

# Numerical Methods

## for Computational Science and Engineering

(“Numerische Methoden für CSE”, 401-0663-00 V)

Prof. Ralf Hiptmair and Dr. Vasile Gradinaru

Draft version November 5, 2011, Subversion 38286

(C) Seminar für Angewandte Mathematik, ETH Zürich

<http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE11.pdf>

Always under construction



# Contents

<b>1</b>	<b>Computing with Matrices and Vectors [48, Sect. 2.1&amp;2.2], [27, Sect. 2.1&amp;2.2]</b>	<b>28</b>
1.1	Notations . . . . .	29
1.1.1	Vectors . . . . .	29
1.1.2	Matrices . . . . .	31
1.2	Elementary operations . . . . .	40
1.3	Complexity/computational effort . . . . .	54
1.4	BLAS . . . . .	63
<b>2</b>	<b>Direct Methods for Linear Systems of Equations</b>	<b>85</b>
2.1	Gaussian Elimination . . . . .	89
2.2	LU-Decomposition/LU-Factorization . . . . .	106
2.3	Pivoting . . . . .	124
2.4	Supplement: Machine Arithmetic . . . . .	138



2.5	Stability of Gaussian Elimination . . . . .	148
2.5.1	Vector norms and matrix norms [13, Sect. 2.1.2], [35, Sect. 1.2], [51, Sect. 1.11] . . . . .	148
2.5.2	Numerical Stability [13, Sect. 2.3] . . . . .	152
2.5.3	Roundoff analysis of Gaussian elimination . . . . .	155
2.5.4	Conditioning . . . . .	166
2.5.5	Sensitivity of linear systems . . . . .	172
2.6	Sparse Matrices . . . . .	175
2.6.1	Sparse matrix storage formats . . . . .	181
2.6.2	Sparse matrices in MATLAB . . . . .	183
2.6.3	LU-factorization of sparse matrices . . . . .	195
2.6.4	Banded matrices [13, Sect. 3.7] . . . . .	209
2.7	Stable Gaussian elimination without pivoting . . . . .	223
2.8	QR-factorization/decomposition [35, Sect. 13], [27, Sect. 7.3] . . . . .	240
2.9	Modification Techniques . . . . .	267
2.9.0.1	Rank-1-modifications . . . . .	270
2.9.0.2	Adding a column . . . . .	281
2.9.0.3	Adding a row . . . . .	286

<b>3</b>	<b>Data Interpolation in 1D</b>	<b>289</b>	
3.1	Abstract interpolation . . . . .	290	NumCSE, autumn 2010
3.2	Polynomials . . . . .	300	
3.3	Polynomial Interpolation: Theory . . . . .	302	
3.4	Polynomial Interpolation: Algorithms . . . . .	310	
3.4.1	Multiple evaluations . . . . .	311	
3.4.2	Single evaluation [13, Sect. 8.2.2] . . . . .	314	
3.4.3	Extrapolation to zero . . . . .	319	
3.4.4	Newton basis and divided differences [13, Sect. 8.2.4], [51, Sect. 8.2] . . . . .	329	
3.5	Polynomial Interpolation: Sensitivity [51, Sect. 8.1.3] . . . . .	337	
3.6	Shape preserving interpolation . . . . .	345	R. Hiptmair
3.6.1	Piecewise linear interpolation . . . . .	351	rev 38286, November 5, 2011
3.7	Cubic Hermite Interpolation . . . . .	355	
3.7.1	Definition and algorithms . . . . .	356	
3.7.2	Shape preserving Hermite interpolation . . . . .	361	
3.8	Splines [13, Ch. 9] . . . . .	367	
3.8.1	Cubic spline interpolation [35, XIII, 46], [51, Sect. 8.6.1] . . . . .	369	
3.8.2	Shape Preserving Spline Interpolation . . . . .	385	

<b>4</b>	<b>Iterative Methods for Non-Linear Systems of Equations</b>	<b>401</b>	
4.1	Iterative methods . . . . .	404	NumCSE, autumn 2010
4.1.1	Speed of convergence . . . . .	409	
4.1.2	Termination criteria . . . . .	418	
4.2	Fixed Point Iterations [13, Sect. 5.3], [51, Sect. 6.3] . . . . .	425	
4.2.1	Consistent fixed point iterations . . . . .	426	
4.2.2	Convergence of fixed point iterations . . . . .	430	
4.3	Zero Finding . . . . .	443	
4.3.1	Bisection [13, Sect. 5.5.1] . . . . .	444	
4.3.2	Model function methods . . . . .	446	
4.3.2.1	Newton method in scalar case [35, Sect. 18.1], [13, Sect. 5.5.2] . . . . .	447	R. Hiptmair
4.3.2.2	Special one-point methods . . . . .	449	rev 38286, November 5, 2011
4.3.2.3	Multi-point methods . . . . .	458	
4.3.3	Note on Efficiency . . . . .	466	
4.4	Newton's Method [35, Sect. 19], [13, Sect. 5.6] . . . . .	470	
4.4.1	The Newton iteration . . . . .	470	
4.4.2	Convergence of Newton's method . . . . .	477	
4.4.3	Termination of Newton iteration . . . . .	482	
4.4.4	Damped Newton method [13, pp. 200] . . . . .	485	0.0
4.4.5	Quasi-Newton Method [51, Sect. 7.1.4] . . . . .	492	p. 5

<b>5</b>	<b>Krylov Methods for Linear Systems of Equations</b>	<b>501</b>	
5.1	Descent Methods [51, Sect. 4.3.3]	503	NumCSE, autumn 2010
5.1.1	Quadratic minimization context	504	
5.1.2	Abstract steepest descent	507	
5.1.3	Gradient method for s.p.d. linear system of equations	510	
5.1.4	Convergence of the gradient method	513	
5.2	Conjugate gradient method (CG) [35, Ch. 9], [13, Sect. 13.4], [51, Sect. 4.3.4]	522	
5.2.1	Krylov spaces	526	
5.2.2	Implementation of CG	528	
5.2.3	Convergence of CG	536	
5.3	Preconditioning [13, Sect. 13.5], [35, Ch. 10], [51, Sect. 4.3.5]	550	
5.4	Survey of Krylov Subspace Methods	566	
5.4.1	Minimal residual methods	566	
5.4.2	Iterations with short recursions [51, Sect. 4.5]	569	
<b>6</b>	<b>Eigenvalues</b>	<b>574</b>	R. Hiptmair
6.1	Theory of eigenvalue problems [48, Ch. 7], [27, Ch. 9], [51, Sect. 1.7]	583	rev 38286, November 5, 2011
6.2	“Direct” Eigensolvers	589	
6.3	Power Methods	599	
6.3.1	Direct power method [13, Sect. 7.5], [51, Sect. 5.3.1], [51, Sect. 5.3]	599	
6.3.2	Inverse Iteration [13, Sect. 7.6], [51, Sect. 5.3.2]	623	
6.3.3	Preconditioned inverse iteration (PINVIT)	652	
6.3.4	Subspace iterations	658	
6.3.4.1	Orthogonalization	671	
6.3.4.2	Ritz projection	683	
6.4	Krylov Subspace Methods [35, Sect. 30]	694	0.0
6.5	Singular Value Decomposition	719	p. 6

<b>7</b>	<b>Least Squares</b>	<b>764</b>	
7.1	Normal Equations [13, Sect. 4.2], [35, Ch. 11]	778	
7.2	Orthogonal Transformation Methods [13, Sect. 4.4.2]	784	
7.3	Total Least Squares	801	
7.4	Constrained Least Squares	804	
7.4.1	Solution via normal equations	805	
7.4.2	Solution via SVD	809	
7.5	Non-linear Least Squares [13, Ch. 6]	810	
7.5.1	(Damped) Newton method	812	
7.5.2	Gauss-Newton method	814	
7.5.3	Trust region method (Levenberg-Marquardt method)	822	
<b>8</b>	<b>Filtering Algorithms</b>	<b>824</b>	
8.1	Discrete convolutions	826	
8.2	Discrete Fourier Transform (DFT)	845	
8.2.1	Discrete convolution via DFT	857	
8.2.2	Frequency filtering via DFT	859	
8.2.3	Real DFT	875	
8.2.4	Two-dimensional DFT	877	
8.2.5	Semi-discrete Fourier Transform [51, Sect. 10.11]	885	
8.3	Fast Fourier Transform (FFT) [13, Sect. 8.7.3], [35, Sect. 53], [51, Sect. 10.9.2]	904	
8.4	Trigonometric transformations [35, Sect. 55]	916	
8.4.1	Sine transform	917	
8.4.2	Cosine transform	926	
8.5	Toeplitz matrix techniques	929	
8.5.1	Toeplitz matrix arithmetic	933	
8.5.2	The Levinson algorithm	935	

<b>9</b>	<b>Approximation of Functions in 1D</b>	<b>940</b>	
9.1	Error estimates for polynomial interpolation . . . . .	943	NumCSE, autumn 2010
9.2	Chebychev Interpolation . . . . .	957	
9.2.1	Motivation and definition . . . . .	957	
9.2.2	Chebychev interpolation error estimates . . . . .	966	
9.2.3	Chebychev interpolation: computational aspects . . . . .	976	
9.3	Trigonometric interpolation [13, Sect. 8.5] . . . . .	983	
9.4	Approximation by piecewise polynomials . . . . .	1001	
9.4.1	Piecewise Lagrange interpolation . . . . .	1003	
9.4.2	Cubic Hermite interpolation: error estimates . . . . .	1012	
9.4.3	Cubic spline interpolation: error estimates [35, Ch. 47] . . . . .	1019	
<b>10</b>	<b>Numerical Quadrature [35, VII], [13, Ch. 10]</b>	<b>1023</b>	R. Hiptmair
10.1	Quadrature Formulas . . . . .	1026	rev 38286, November 5, 2011
10.2	Polynomial Quadrature Formulas [13, Sect. 10.2] . . . . .	1030	
10.3	Composite Quadrature . . . . .	1037	
10.4	Gauss Quadrature [35, Ch. 40-41], [13, Sect.10.3] . . . . .	1063	
10.5	Adaptive Quadrature . . . . .	1085	
<b>11</b>	<b>Clustering Techniques</b>	<b>1093</b>	
11.1	Kernel Matrices . . . . .	1093	
11.2	Local Separable Approximation . . . . .	1096	
11.3	Cluster Trees . . . . .	1109	0.0
11.4	Algorithm . . . . .	1118	p. 8

<b>12 Single Step Methods</b>	<b>1136</b>	
12.1 Initial value problems (IVP) for ODEs . . . . .	1136	NumCSE, autumn 2010
12.1.1 Examples . . . . .	1137	
12.1.2 Theory [51, Sect. 11.1], [13, Sect. 11.3] . . . . .	1147	
12.2 Euler methods . . . . .	1156	
12.2.1 Explicit Euler method . . . . .	1157	
12.2.2 Implicit Euler method . . . . .	1160	
12.2.3 Abstract single step methods . . . . .	1164	
12.3 Convergence of single step methods [13, Sect. 11.5] [51, Sect. 11.3] . . . . .	1168	
12.4 Runge-Kutta methods [13, Sect. 11.6], [35, Ch. 76], [51, Sect. 11.8] . . . . .	1177	
12.5 Stepsize control [13, Sect. 11.7], [51, Sect. 11.8.2] . . . . .	1189	
<b>13 Stiff Integrators [13, Sect. 11.9]</b>	<b>1220</b>	R. Hiptmair rev 38286, October 21, 2011
13.1 Model problem analysis [35, Ch. 77], [51, Sect. 11.3.3] . . . . .	1222	
13.2 Stiff problems [51, Sect. 11.10] . . . . .	1231	
13.3 Implicit Runge-Kutta methods [13, Sect. 11.6.2], [51, Sect. 11.8.3] . . . . .	1248	
13.4 Semi-implicit Runge-Kutta methods [35, Ch. 80] . . . . .	1256	
<b>14 Structure Preservation [31]</b>	<b>1261</b>	
14.1 Dissipative Evolutions . . . . .	1261	
14.2 Quadratic Invariants . . . . .	1261	
14.3 Reversible Integrators . . . . .	1261	0.0
14.4 Symplectic Integrators . . . . .	1261	p. 9

## Focus

- ▷ on **algorithms** (principles, scope, and limitations),
- ▷ on (efficient, stable) **implementation** in MATLAB,
- ▷ on **numerical experiments** (design and interpretation).

no emphasis on

- theory and proofs (unless essential for understanding of algorithms)
- hardware-related issues (e.g. parallelization, vectorization, memory access)  
(These aspect will be covered in the new course “Parallel Computing for Scientific Simulation” offered by D-INFK from autumn term 2012)



Linear systems of  
equations

Eigenvalue  
problems

Least squares  
problems

Interpolation

Quadrature

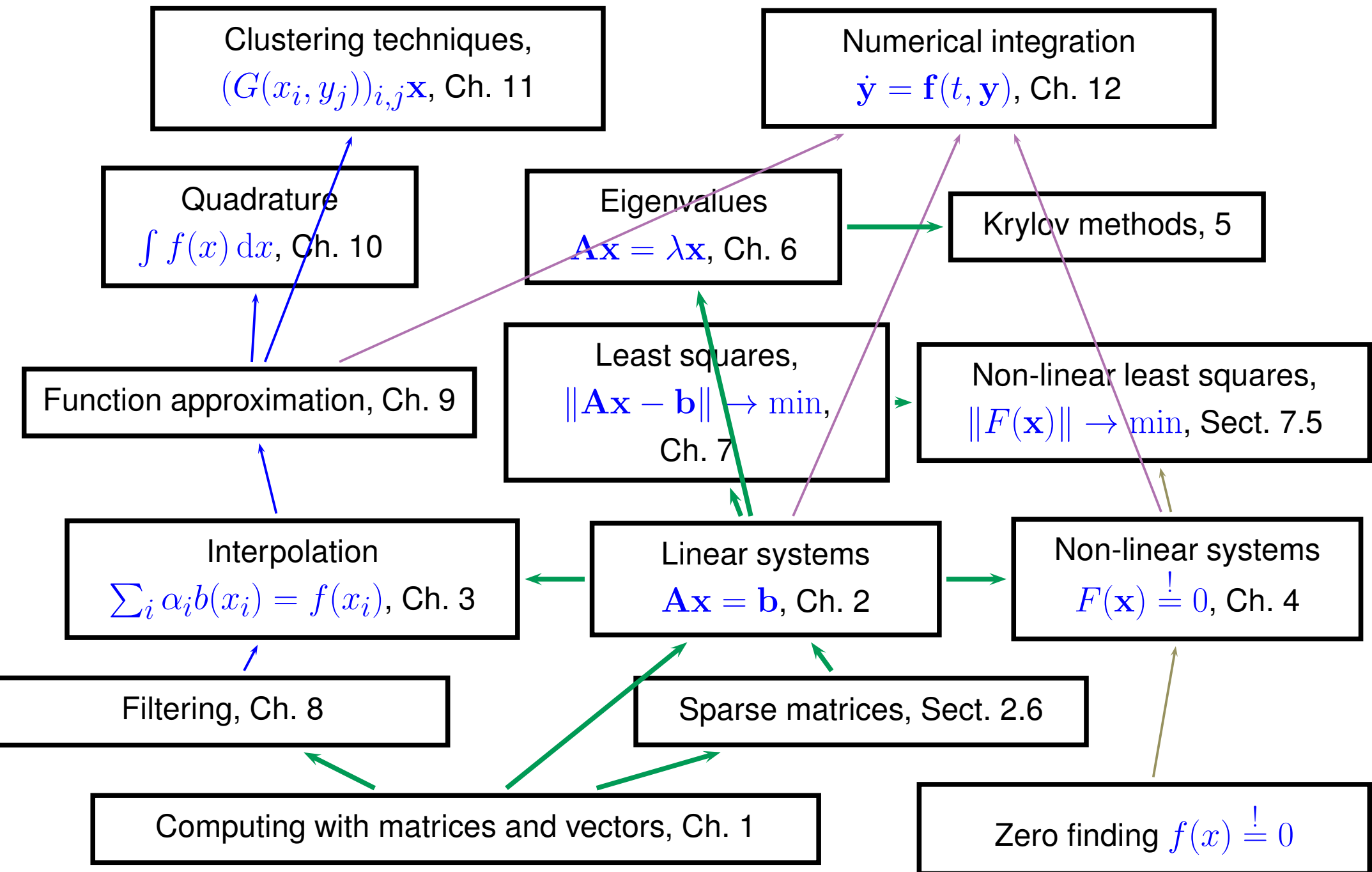
Numerical  
integration of ODEs



Analysis

Linear algebra

Programming



I am a student of computer science. After the exam, may I safely forget everything I have learned in this mandatory “numerical methods” course?

singular value decomposition }  
least squares } ▶ Computational statistics, machine learning

function approximation }  
numerical quadrature } ▶ Numerical methods for PDEs  
numerical integration }

interpolation }  
least squares } ▶ Computer graphics

eigensolvers }  
sparse linear systems } ▶ Graph theoretic algorithms

numerical integration } ▶ Computer animation

and many more applications of fundamental numerical methods . . . .

# Goals

- Knowledge of the fundamental algorithms in numerical mathematics
- Knowledge of the essential terms in numerical mathematics and the techniques used for the analysis of numerical algorithms
- Ability to choose the appropriate numerical method for concrete problems
- Ability to interpret numerical results
- Ability to implement numerical algorithms efficiently in MATLAB

Indispensable:

Learning by doing (→ exercises)

## Reading instructions

These course materials are neither a textbook nor comprehensive lecture notes.  
They are meant to be supplemented by explanations given in class.

## Some pieces of advice:

- these lecture slides are not designed to be self-contained, but to supplement explanations in class.
- this document is not meant for mere reading, but for working with,
- turn pages all the time and follow the numerous cross-references,
- study the relevant section of the course material when doing homework problems,
- study referenced literature to refresh prerequisite knowledge and for alternative presentation of the material (from a different angle, maybe).

## What to expect

- The course is difficult and demanding (*ie.* ETH level)
- Do **not** expect to understand everything in class. The average student will
  - understand about one third of the material when attending the lectures,
  - understand another third when making a *serious effort* to solve the homework problems,
  - hopefully understand the remaining third when studying for the examination after the end of the course.

Perseverance will be rewarded!

Parts of the following textbooks may be used as supplementary reading for this course. References to relevant sections will be provided in the course material.

- [[13]] W. DAHMEN AND A. REUSKEN, *Numerik für Ingenieure und Naturwissenschaftler*, Springer, Heidelberg, 2006.

Good reference for large parts of this course; provides a lot of simple examples and lucid explanations, but also rigorous mathematical treatment. (Target audience: undergraduate students in science and engineering)

- [[35]] M. HANKE-BOURGEOIS, *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*, Mathematische Leitfäden, B.G. Teubner, Stuttgart, 2002.

Gives detailed description and mathematical analysis of algorithms and relies on MATLAB. Profound treatment of theory way beyond the scope of this course. (Target audience: undergraduates in mathematics)

- [[51]] A. QUARTERONI, R. SACCO, AND F. SALERI, *Numerical mathematics*, vol. 37 of *Texts in Applied Mathematics*, Springer, New York, 2000.

Classical introductory numerical analysis text with many examples and detailed discussion of algorithms. (Target audience: undergraduates in mathematics and engineering)

- [[17]] P. DEUFLHARD AND A. HOHMANN, *Numerische Mathematik. Eine algorithmisch orientierte Einführung*, DeGruyter, Berlin, 1 ed., 1991.

Modern discussion of numerical methods with profound treatment of theoretical aspects (Target audience: undergraduate students in mathematics).

Essential prerequisite for this course is a solid knowledge in linear algebra and calculus. Familiarity with the topics covered in the first semester courses is taken for granted, see

- [[48]] K. NIPP AND D. STOFFER, *Lineare Algebra*, vdf Hochschulverlag, Zürich, 5 ed., 2002.
- [[27]] M. GUTKNECHT, *Lineare algebra*, lecture notes, SAM, ETH Zürich, 2009.  
<http://www.sam.math.ethz.ch/~mhg/unt/LA/HS07/>.
- [[63]] M. STRUWE, *Analysis für Informatiker*. Lecture notes, ETH Zürich, 2009.  
<https://moodle-app1.net.ethz.ch/lms/mod/resource/index.php?id=145>.

# General information

Lecturer: Prof. Ralf Hiptmair HG G 58.2, ☎ 044 632 3404, [hiptmair@sam.math.ethz.ch](mailto:hiptmair@sam.math.ethz.ch)

Assistants: Jonas Šukys, HG E 62.1, ☎ 044 632 0401, [jonas.sukys@sam.math.ethz.ch](mailto:jonas.sukys@sam.math.ethz.ch)  
Rafaela Guberovic, HG J 47, ☎ 044 632 4320, [rafaela.guberovic@sam.math.ethz.ch](mailto:rafaela.guberovic@sam.math.ethz.ch)  
Konstantin Grella, HG J 59, ☎ 044 632 6362, [konstantin.grella@sam.math.ethz.ch](mailto:konstantin.grella@sam.math.ethz.ch)  
Robert Carnecky, CAB G 66.2, ☎ 044 632 5527, [crobi@inf.ethz.ch](mailto:crobi@inf.ethz.ch)  
Sebastian Lienert, [lienerts@student.ethz.ch](mailto:lienerts@student.ethz.ch)  
Simon Härdi, [shaerdi@student.ethz.ch](mailto:shaerdi@student.ethz.ch)

Classes: Mon, 10.15-12.00 (HG F3), Thu, 10.15-12.00 (HG G5)

Tutorials: Mon, 13.15-15.00 (LFO C 13, ML H 34.3)

Thu, 8.15-10.00 (HG G 26.3, HG G 26.5, LFW E 13)

Thu, 13.15-15.00 (ML F 40)

Please register (in course website) for tutorial groups until September 25th:

[http://www.math.ethz.ch/education/bachelor/lectures/hs2011/math/nummath\\_cse](http://www.math.ethz.ch/education/bachelor/lectures/hs2011/math/nummath_cse)

Consulting hours: Tue 12-13 Rafaela Guberovic, Jonas Šukys (HG E 62.1)

Wed 12-13 Sebastian Lienert, Simon Härdi (in front of HG J 54)

Fri 12-13 Konstantin Grella, Robert Carnecky (in front of HG J 54)



# Assignments

- The assignment sheets will be uploaded on the course webpage on Monday every week.
- The attempted solutions have to be handed in until the following tutorial class (give them to the assistant during the tutorial class itself or put them into the plexiglass trays in HG G 53/5).
- To earn a “Testat” you are expected to
  - present your solutions of **two** homework problems at the **blackboard** in your tutorial class. In the beginning of the semester you are supposed to sign up for two homework problems. Sign up links are available on the course website. Deadline: September 25th.
  - make a serious effort to solve the exercises and hand in the attempted solutions. In order to obtain the “Testat” at least **80%** of the *core homework problems* have to be solved.

## Self-grading

For each exercise sheet the students have to state which (sub)problems they have solved or partially solved by ticking the fields *solved* (100%), *partly solved* (50%) or *not solved* (0%) in a list for each sub-problem. Random checks will be conducted to ensure honest self-grading and cheaters will be

punished. In case a student is caught cheating (i.e., stating as solved exercises that he/she didn't solve) the stricter requirements for the "Testat" will apply to the perpetrator.

However, you may request thorough corrections for up to five individual homework problems during the term. Indicate your requests clearly in the self-grading sheet.

**Website:**

[http://www.math.ethz.ch/education/bachelor/lectures/hs2011/math/nummath\\_cse](http://www.math.ethz.ch/education/bachelor/lectures/hs2011/math/nummath_cse)

## **Online homework submission (optional)**

You can use the online homework submission tool:

<http://www.math.ethz.ch/~grsam/submit/?VL=02>

- Three-hour written examination involving coding problems to be done at the computer on

TBA

- Dry-run for computer based examination:

TBA, registration via course website

- Pre-exam question session:

TBA

- Subjects of examination:

- All topics, which have been addressed in class or in a homework problem.

- Lecture documents will be available as PDF during the examination. The corresponding final version of the lecture documents will be made available on TBA

- The exam questions will be asked both in German and in English.

# Changes in course documentation during HS11

These course documents are being altered all the time in response to new ideas and topics and experiences in class . The latter sometimes suggest significant changes after the material has been taught. This is not completely desirable, but may be better than keeping poorly devised parts.

The following significant changes were made to the lecture material after it had been made available in HS 2011:

- The policy for partial pivoting in Sect. 2.3, Formula (2.3.7) was changed to match that of [48, Sect. 2.5].
- The previous subsection 3.3.2 has been moved and promoted to the new Section 3.5. Only slight alterations have been made in the process.

## Reporting errors

Please report errors in the electronic lecture notes via a [wiki page](#) !

(Password: CSE, please choose **EDIT** menu to enter information)

Please supply the following information:

- (sub)section where the error has been found,
- precise location (e.g, after Equation (4), Thm. 2.3.3, etc. ). Refrain from giving page numbers,
- brief description of the error.

Alternative (for people not savvy with wikis): E-mail an `hiptmair@sam.math.ethz.ch`, **Sub-**  
**ject:** NUMCSE

## Extra questions for course evaluation

Course number (LV-ID): 401-0663-00

Date of evaluation: TBA

- D1:** I try to do all programming exercises. (HS 10: 2-9-25-39-26, 3.8)
- D2:** The programming exercises help understand the numerical methods. (HS 10: 2-9-33-42-12, 3.6)
- D3:** The programming exercises offer too little benefit for the effort spent on them. (HS 10: 7-16-21-25-29, 3.5)
- D4:** Scope and limitations of methods are properly addressed in the course. (HS 10: 9-12-19-30-19, 3.4)
- D5:** Numerical examples in class provide useful insights and motivation. (HS 10: 13-13-22-25-25, 3.4)
- D6:** There should be more examples presented and discussed in class. (HS 10: 2-15-20-28-31, 3.8)
- D7:** Too much information is crammed onto the lecture slides (HS 10: 2-7-13-16-62, 4.3)
- D8:** The course requires too much prior knowledge in linear algebra (HS 10: 18-24-24-22-13, 2.9)
- D9:** The course requires too much prior knowledge in analysis (HS 10: 25-30-21-16-5, 2.5)
- D10:** My prior knowledge of MATLAB was insufficient for the course (HS 10: 31-29-13-22-5, 2.4)
- D11:** More formal proofs would be desirable (HS 10: 56-19-15-7-4, 1.9)
- D12:** The explanations on the blackboard promote understanding (HS 10: 13-36-25-20-5, 2.7)
- D13:** The codes included in the lecture material convey useful information (HS 10: 5-20-18-25-29, 3.5)
- D14:** The master solutions for exercise problems offer too little guidance. (HS 10: 5-13-22-15-13, 3.2)
- D15:** The relevance of the numerical methods taught in the course is convincingly conveyed. (HS 10: 9-15-27-22-11, 3.1)

16: I would not mind the course being taught in English. (HS 10: 25-14-23-23-13, 2.8)

- Scoring:
- 6: I agree fully
  - 5: I agree to a large extent
  - 4: I agree partly
  - 3: I do not quite agree
  - 2: I disagree
  - 1: I disagree strongly

Evaluation of assistants:

Assistant	shortcut

Please enter the shortcut code after the LV-ID in the three separate boxes.

# MATLAB

- MATLAB** ("Matrix Laboratory"):
- full fledged high level *programming language* (for numerical algorithms)
  - integrated *programming environment*
  - versatile collection of *numerical libraries*

- in this course used for
- ▷ demonstrating (implementation of) algorithms
  - ▷ numerical experiments
  - ▷ programming assignments

This course assumes *familiarity with MATLAB*  
as acquired in the introductory linear algebra courses.

Proficiency in MATLAB through "Learning on the job"  
(using the very detailed online help facilities)



```

Terminal
File Edit View Terminal Tabs Help

>> help surf
SURF 3-D colored surface.
SURF(X,Y,Z,C) plots the colored parametric surface defined by
four matrix arguments. The view point is specified by VIEW.
The axis labels are determined by the range of X, Y and Z,
or by the current setting of AXIS. The color scaling is determined
by the range of C, or by the current setting of CAXIS. The scaled
color values are used as indices into the current COLORMAP.
The shading model is set by SHADING.

SURF(X,Y,Z) uses C = Z, so color is proportional to surface height.

SURF(x,y,Z) and SURF(x,y,Z,C), with two vector arguments replacing
the first two matrix arguments, must have length(x) = n and
length(y) = m where [m,n] = size(Z). In this case, the vertices
of the surface patches are the triples (x(j), y(i), Z(i,j)).
Note that x corresponds to the columns of Z and y corresponds to
the rows.

SURF(Z) and SURF(Z,C) use x = 1:n and y = 1:m. In this case,
the height, Z, is a single-valued function, defined over a
geometrically rectangular grid.

SURF(...,'PropertyName',PropertyValue,...) sets the value of the
specified surface property. Multiple property values can be set
with a single statement.

SURF(AX,...) plots into AX instead of GCA.

SURF returns a handle to a surface plot object.

AXIS, CAXIS, COLORMAP, HOLD, SHADING and VIEW set figure, axes, and
surface properties which affect the display of the surface.

Backwards compatibility
SURF('v6',...) creates a surface object instead of a surface plot
object for compatibility with MATLAB 6.5 and earlier.

See also SURFC, SURFL, MESH, SHADING.

>> 

```

Useful links:

[Matlab Online Documentation](#)

[MATLAB guide](#)

[MATLAB Primer](#)

Miscellaneous internet resources:

- [MATLAB classroom resources](#)

- 

[Mathematical Methods for Simulation with MATLAB - Course material developed by Professor Gilbert](#)

- [Numerical algorithms](#)

- [NUMAWWW](#)

# 1 Computing with Matrices and Vectors

## [48, Sect. 2.1&2.2], [27, Sect. 2.1&2.2]

The implementation of most numerical algorithms relies on array type data structures modelling concepts from linear algebra (matrices and vectors). Related information can be found in [23, Ch. 1].

Learning outcomes for this chapter:

- Repetition of basic vector and matrix operations from linear algebra and awareness of their computational cost.
- Understanding of the concepts of computational effort/cost and asymptotic complexity.

- Ability to determine the (asymptotic) computational effort for a concrete (numerical linear algebra) algorithm.
- Grasp of the importance of key matrix properties (low rank or bad structure) for efficient implementation.
- Ability to manipulate simple expressions involving matrices and vectors in order to reduce the computational cost for their evaluation.
- Knowledge about BLAS and the benefit of using BLAS routines.

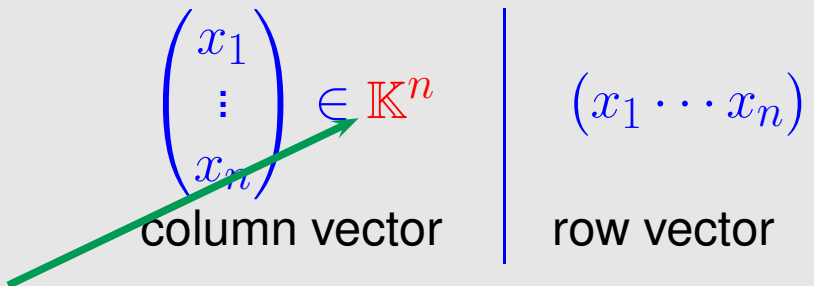
## 1.1 Notations

### 1.1.1 Vectors

● **Vectors** = are  $n$ -tuples ( $n \in \mathbb{N}$ ) with components  $x_i \in \mathbb{K}$ , over field  $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ .

vector = one-dimensional array (of real/complex numbers)

● Default in this lecture: vectors = **column vectors**



vector space of column vectors with  $n$  components

✎ notation for column vectors: **bold** small roman letters, e.g. **x, y, z**

● Initialization of vectors in **MATLAB**:

```
column vectors x = [1; 2; 3];
row vectors    y = [1, 2, 3];
```

● **Transposing:**  $\begin{cases} \text{column vector} \mapsto \text{row vector} \\ \text{row vector} \mapsto \text{column vector} \end{cases}$

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}^T = (x_1 \cdots x_n) \quad , \quad (x_1 \cdots x_n)^T = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

Transposing in **MATLAB**:  $x\_T = x'$  ;

## Addressing vector components:

two notations:  $\mathbf{x} = (x_1 \dots x_n)^\top \rightarrow x_i, \quad i = 1, \dots, n$   
 $\mathbf{x} \in \mathbb{K}^n \rightarrow (\mathbf{x})_i, \quad i = 1, \dots, n$

MATLAB: `x(i)` selects  $i$ -th component of vector `x`

## Selecting sub-vectors:

notation:  $\mathbf{x} = (x_1 \dots x_n)^\top \triangleright (\mathbf{x})_{k:l} = (x_k, \dots, x_l)^\top, \quad 1 \leq k \leq l \leq n$

MATLAB: `x(4:7)`  $\longleftrightarrow$  `[x(4);x(5);x(6);x(7)]`,  
`x(7:-1:4)`  $\longleftrightarrow$  `[x(7);x(6);x(5);x(4)]`,  
`x(3:2:10)`  $\longleftrightarrow$  `[x(3);x(5);x(7);x(9)]`

$j$ -th unit vector:  $\mathbf{e}_j = (0, \dots, 1, \dots, 0)^\top, \quad e_i = \delta_{ij}$ .

notation: **Kronecker symbol**  $\delta_{ij} := 1, \text{ if } i = j, \delta_{ij} := 0, \text{ if } i \neq j.$

## 1.1.2 Matrices

● **Matrices** = two-dimensional arrays of real/complex numbers

$$\mathbf{A} := \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \in \mathbb{K}^{n,m}, \quad n, m \in \mathbb{N}.$$

vector space of  $n \times m$ -matrices: ( $n \hat{=}$  number of **rows**,  $m \hat{=}$  number of **columns**)

✎ notation: **bold** capital roman letters, e.g., **A**, **S**, **Y**

$$\mathbb{K}^{n,1} \leftrightarrow \text{column vectors}, \quad \mathbb{K}^{1,n} \leftrightarrow \text{row vectors}$$

**MATLAB**: matrix as fundamental data structure

▷ vectors are  $1 \times n/n \times 1$ -matrices

▷ initialization:  $\mathbf{A} = [1, 2; 3, 4; 5, 6]; \rightarrow 3 \times 2$  matrix  $\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$ .

Note: the treatment of vectors as special matrices is consistent with the basic operations from matrix calculus, see Sect. 1.2.

Assembling a matrix from vectors:

$$\mathbf{c}_i \in \mathbb{K}^n, \quad i = 1, \dots, m \quad \triangleright \quad \mathbf{A} = (\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m) \in \mathbb{K}^{n,m},$$

$$\mathbf{r}_i \in \mathbb{K}^m, \quad i = 1, \dots, n \quad \triangleright \quad \mathbf{A} = \begin{pmatrix} \mathbf{r}_1^\top \\ \vdots \\ \mathbf{r}_n^\top \end{pmatrix} \in \mathbb{K}^{n,m}.$$

MATLAB:

$$A = [c1, c2, c3]; \quad , \quad A = [r1'; r2'; r3'];$$

Accessing matrix entries & sub-matrices (📎 notations):

$$\mathbf{A} := \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix}$$

$$\rightarrow \text{entry } (\mathbf{A})_{i,j} = a_{ij}, \quad 1 \leq i \leq n, 1 \leq j \leq m,$$

$$\rightarrow i\text{-th row, } 1 \leq i \leq n: \quad a_{i,:} = (\mathbf{A})_{i,:},$$

$$\rightarrow j\text{-th column, } 1 \leq j \leq m: \quad a_{:,j} = (\mathbf{A})_{:,j},$$

$$\rightarrow \text{matrix block } (a_{ij})_{\substack{i=k,\dots,l \\ j=r,\dots,s}} = (\mathbf{A})_{k:l,r:s}, \quad \begin{matrix} 1 \leq k \leq l \leq n, \\ 1 \leq r \leq s \leq m. \end{matrix}$$

(sub-matrix)


MATLAB: Matrix  $\mathbf{A}$   $\mapsto$  entry at position  $(i, j)$   $= A(i, j)$   
 $\mapsto i$ -th row  $= A(i,:)$   
 $\mapsto j$ -th column  $= A(:,j)$   
 $\mapsto$  matrix block  $(a_{ij})_{\substack{i=k,\dots,l \\ j=r,\dots,s}} = (\mathbf{A})_{k:l,r:s} = A(k:l,r:s)$   
(sub-matrix)

● **Transposed matrix:**

$$\mathbf{A}^{\top} = \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix}^{\top} := \begin{pmatrix} a_{11} & \dots & a_{n1} \\ \vdots & & \vdots \\ a_{1m} & \dots & a_{mn} \end{pmatrix} \in \mathbb{K}^{m,n}.$$

● **Adjoint matrix** (Hermitian transposed):

$$\mathbf{A}^{\text{H}} := \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix}^{\text{H}} := \begin{pmatrix} \bar{a}_{11} & \dots & \bar{a}_{n1} \\ \vdots & & \vdots \\ \bar{a}_{1m} & \dots & \bar{a}_{mn} \end{pmatrix} \in \mathbb{K}^{m,n}.$$

 notation:  $\bar{a}_{ij} = \Re(a_{ij}) - i\Im(a_{ij})$  complex conjugate of  $a_{ij}$ .

● **Special matrices:**


**Identity matrix:**  $\mathbf{I} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix} \in \mathbb{K}^{n,n}$ , MATLAB:  $\mathbf{I} = \text{eye}(n)$ ;

**Zero matrix:**  $\mathbf{O} = \begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix} \in \mathbb{K}^{n,m}$ , MATLAB:  $\mathbf{O} = \text{zeros}(n,m)$ ;

**Diagonal matrix:**  $\mathbf{D} = \begin{pmatrix} d_1 & & 0 \\ & \ddots & \\ 0 & & d_n \end{pmatrix} \in \mathbb{K}^{n,n}$ , MATLAB:  $\mathbf{D} = \text{diag}(d)$ ; with vector  $d$



Remark 1.1.1 (Matrix storage formats). (for dense/full matrices, cf. Sect. 2.6)

$A \in \mathbb{K}^{m,n}$   linear array (size  $mn$ ) + index computations  
(Note: two options, *row major*, *column major*)

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Row major (C-arrays, bitmaps, Python):

A_arr	1	2	3	4	5	6	7	8	9
-------	---	---	---	---	---	---	---	---	---

Column major (Fortran, MATLAB, OpenGL):

A_arr	1	4	7	2	5	8	3	6	9
-------	---	---	---	---	---	---	---	---	---

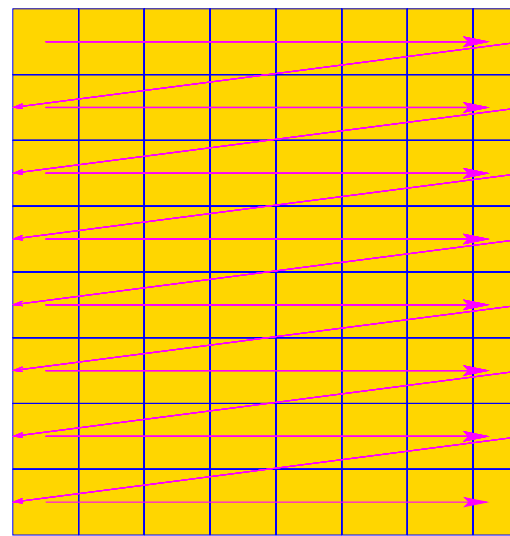
Access to entry  $a_{ij}$  of  $\mathbf{A} \in \mathbb{K}^{n,m}$ ,  $i = 1, \dots, n$ ,  
 $j = 1, \dots, m$ :

row major:

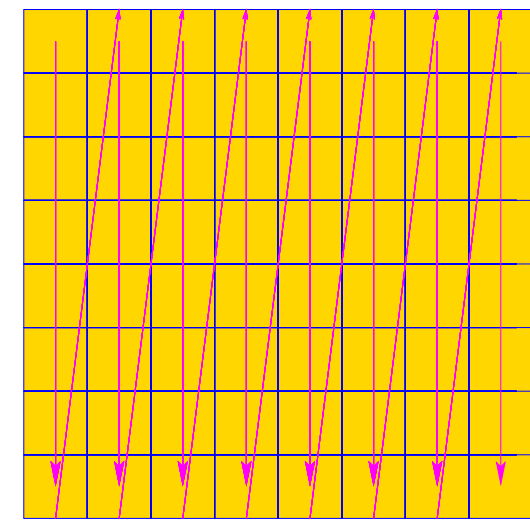
$$a_{ij} \leftrightarrow \text{A\_arr}(m*(i-1)+(j-1))$$

column major:

$$a_{ij} \leftrightarrow \text{A\_arr}(n*(j-1)+(i-1))$$



row major



column major



*Example 1.1.2* (Impact of data access patterns on runtime).

Cache hierarchies  $\rightsquigarrow$  slow access of “remote” memory sites !

column oriented

```
A = randn(n,n);
for j = 1:n-1,
    A(:,j+1) = A(:,j+1) - A(:,j);
end
```

access

row oriented

```
A = randn(n,n);
for i = 1:n-1,
    A(i+1,:) = A(i+1,:) - A(i,:);
end
```

access

## Code 1.1.3: timing for row and column oriented matrix access in MATLAB

NumCSE,  
autumn  
2010

```

1 % Timing for row/column operations on matrices
2 % We conduct K runs in order to reduce the risk of skewed measurements
3 % due to OS activity during MATLAB run.
4 K = 3; res = [];
5 for n=2.(4:13)
6     A = randn(n,n);
7
8     t1 = realmax;
9     for k=1:K, tic;
10        for j = 1:n-1, A(:,j+1) = A(:,j+1) - A(:,j); end;
11        t1 = min(toc,t1);
12    end
13    t2 = realmax;
14    for k=1:K, tic;
15        for i = 1:n-1, A(i+1,:) = A(i+1,:) - A(i,:); end;
16        t2 = min(toc,t2);
17    end
18    res = [res; n, t1 , t2];
19 end
20
21 % Plot runtimes versus matrix sizes
22 figure; plot(res(:,1),res(:,2),'r+', res(:,1),res(:,3),'m*');
23 xlabel('\bf n','fontsize',14);

```

R. Hiptmair  
rev 38286,  
February  
24, 2009

```
24 ylabel ('{\bf runtime [s]}', 'fontsize', 14);
25 legend ('A(:, j+1) = A(:, j+1) - A(:, j)', 'A(i+1, :) = A(i+1, :) -
    A(i, :)', ...
26         'location', 'northwest');
27 print -depsc2 '../PICTURES/accessrtlin.eps';
28
29 figure; loglog (res(:, 1), res(:, 2), 'r+', res(:, 1), res(:, 3), 'm*');
30 xlabel ('{\bf n}', 'fontsize', 14);
31 ylabel ('{\bf runtime [s]}', 'fontsize', 14);
32 legend ('A(:, j+1) = A(:, j+1) - A(:, j)', 'A(i+1, :) = A(i+1, :) -
    A(i, :)', ...
33         'location', 'northwest');
34 print -depsc2 '../PICTURES/accessrtlog.eps';
```

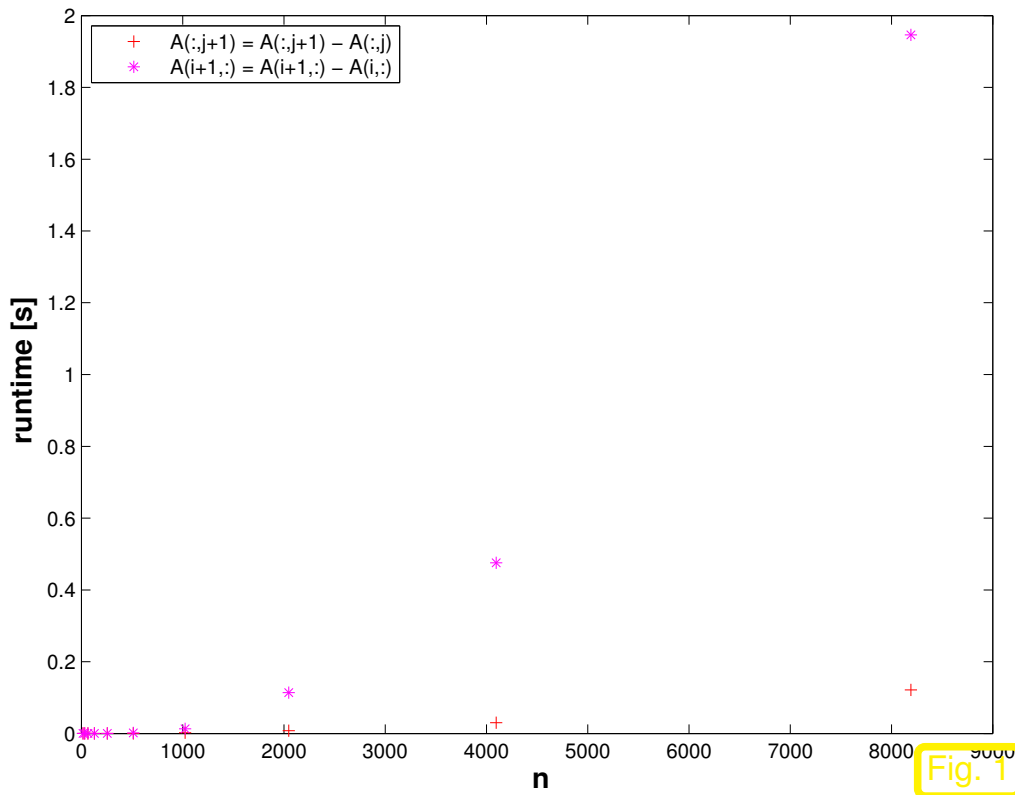


Fig. 1

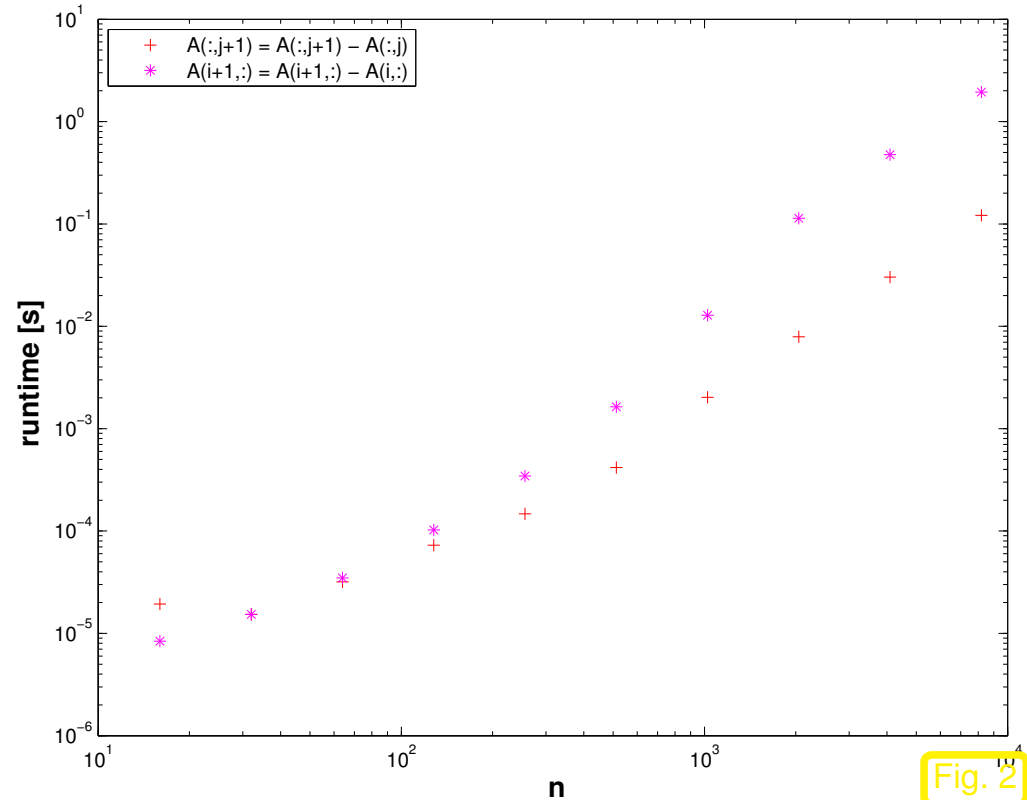


Fig. 2

(Intel Core i7, 2.66 GHz, MacOS X 1.6, MATLAB 7.12.0.635 (R2011a))

Glaring discrepancy of CPU time required for accessing entries of a matrix in rowwise or columnwise fashion. This reflects the impact of features of the underlying hardware architecture, like cache size and memory bandwidth:

Interpretation of timings: Since matrices in MATLAB are stored column major all the matrix elements in a row occupy contiguous memory locations, which will all reside in the cache together. Hence,

column oriented access will mainly operate on data in the cache even for large matrices. Conversely, row oriented access addresses matrix entries that are stored in distant memory locations, which incurs frequent cash misses (**cache thrashing**).

The impact of hardware architecture on the performance of algorithms will **not** be taken into account in this course, because hardware features tend to be both intricate and ephemeral. However, for modern high performance computing it is essential to adapt implementations to the hardware on which the code is supposed to run.



## 1.2 Elementary operations

## What you should know from linear algebra:

- vector space operations in  $\mathbb{K}^{m,n}$  (addition, multiplication with scalars)

- dot product:**  $\mathbf{x}, \mathbf{y} \in \mathbb{K}^n, n \in \mathbb{N}: \mathbf{x} \cdot \mathbf{y} := \mathbf{x}^H \mathbf{y} = \sum_{i=1}^n \bar{x}_i y_i \in \mathbb{K}$

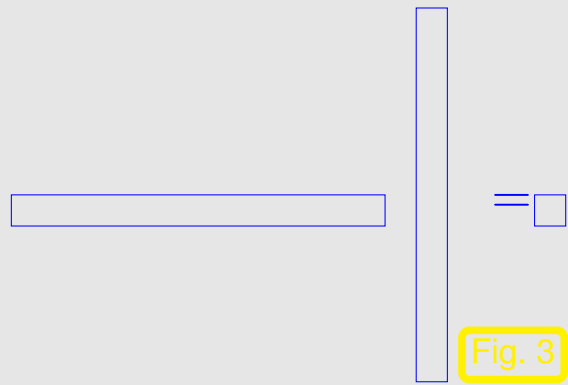
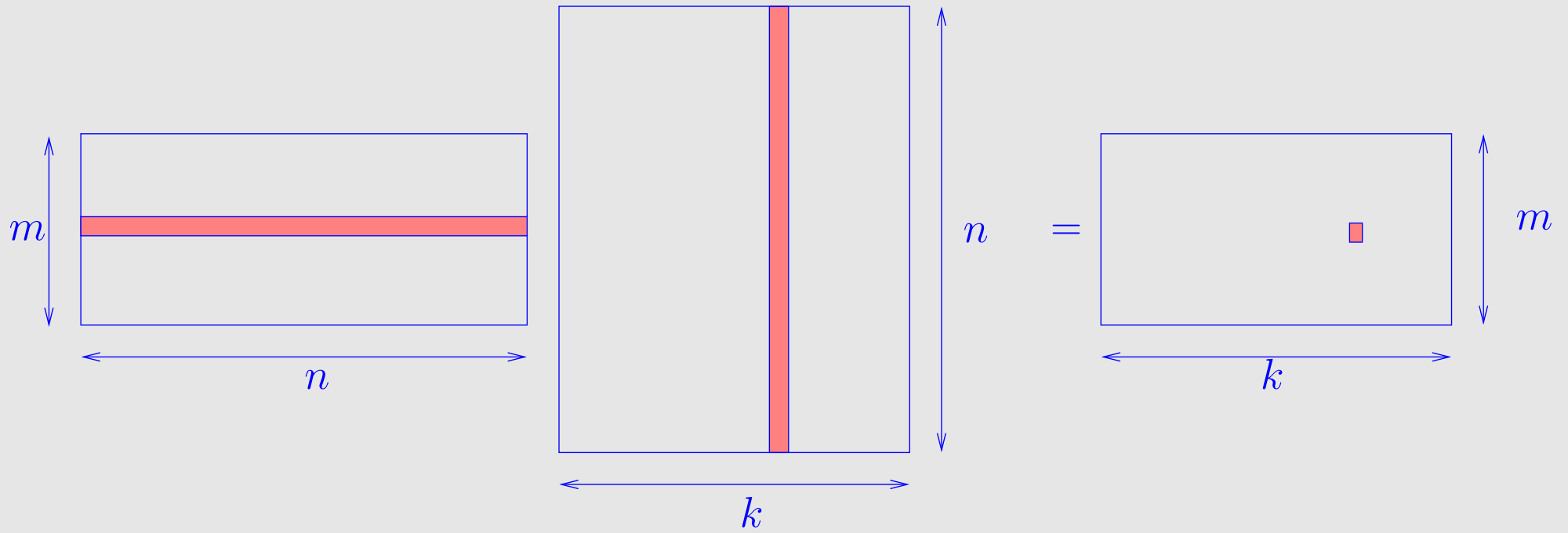
(in MATLAB: `dot(x, y)`)

- tensor product:**  $\mathbf{x} \in \mathbb{K}^m, \mathbf{y} \in \mathbb{K}^n, n \in \mathbb{N}: \mathbf{x} \mathbf{y}^H = (x_i \bar{y}_j)_{\substack{i=1, \dots, m \\ j=1, \dots, n}} \in \mathbb{K}^{m,n}$

- All are special cases of the **matrix product:**

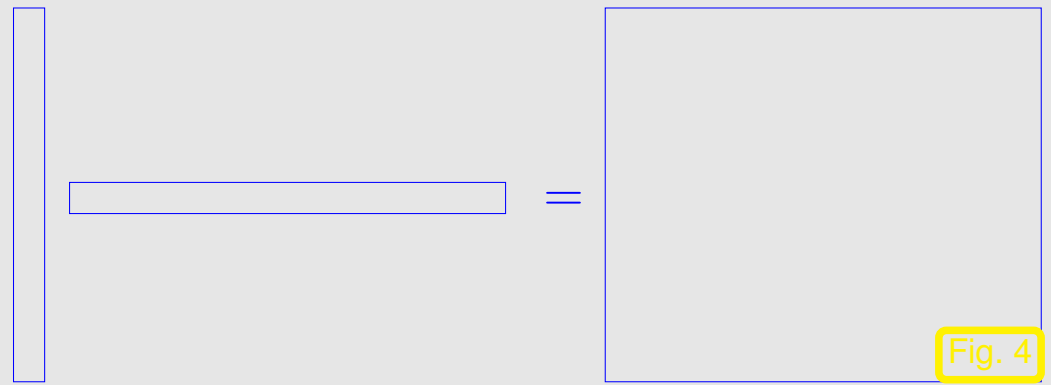
$$\mathbf{A} \in \mathbb{K}^{m,n}, \mathbf{B} \in \mathbb{K}^{n,k} : \mathbf{AB} = \left( \sum_{j=1}^n a_{ij} b_{jl} \right)_{\substack{i=1, \dots, m \\ l=1, \dots, k}} \in \mathbb{R}^{m,k}. \quad (1.2.1)$$

“Visualization” of special matrix products:



dot product

Fig. 3



tensor product

Fig. 4



*Remark 1.2.2* (Row-wise & column-wise view of matrix product).

To understand what is going on when forming a matrix product, it is often useful to decompose it into matrix  $\times$  vector operations in one of the following two ways:

$\mathbf{A} \in \mathbb{K}^{m,n}$ ,  $\mathbf{B} \in \mathbb{K}^{n,k}$ :

$$\mathbf{AB} = \left( \underset{\downarrow}{\mathbf{A}(\mathbf{B})_{:,1}} \quad \dots \quad \mathbf{A}(\mathbf{B})_{:,k} \right), \quad \mathbf{AB} = \begin{pmatrix} (\mathbf{A})_{1,:}\mathbf{B} \\ \vdots \\ (\mathbf{A})_{m,:}\mathbf{B} \end{pmatrix}. \quad (1.2.3)$$

matrix assembled from columns      matrix assembled from rows

For notations refer to Sect. 1.1.2.



*Remark 1.2.4* (Understanding the structure of product matrices).

A “mental image” of matrix multiplication is useful for telling special properties of product matrices.

For instance, zero blocks of the product matrix can be predicted easily in the following situations using

the idea explained in Rem. 1.2.2 (try to understand how):

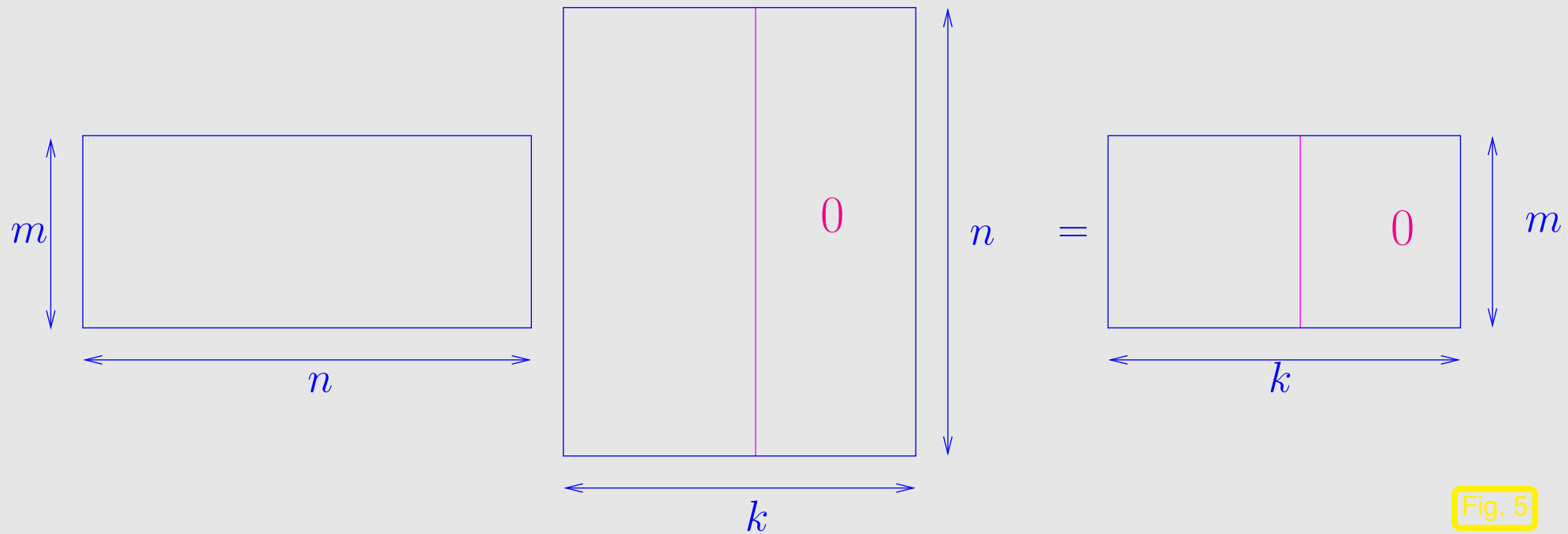


Fig. 5

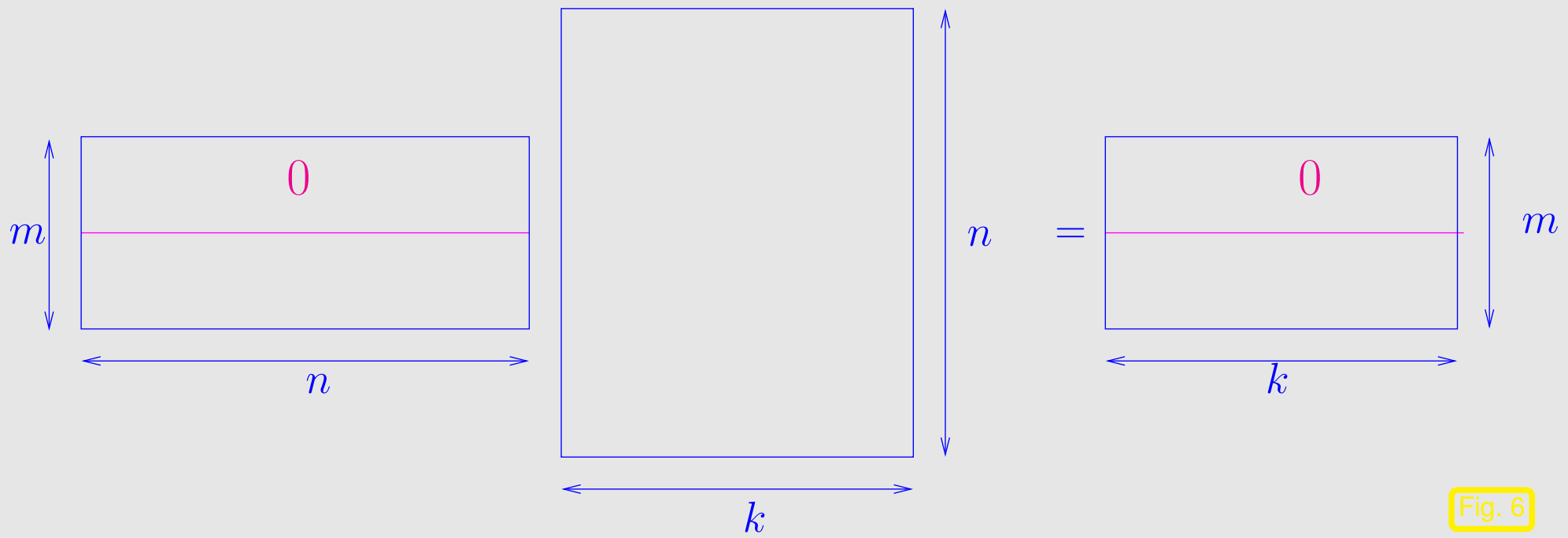
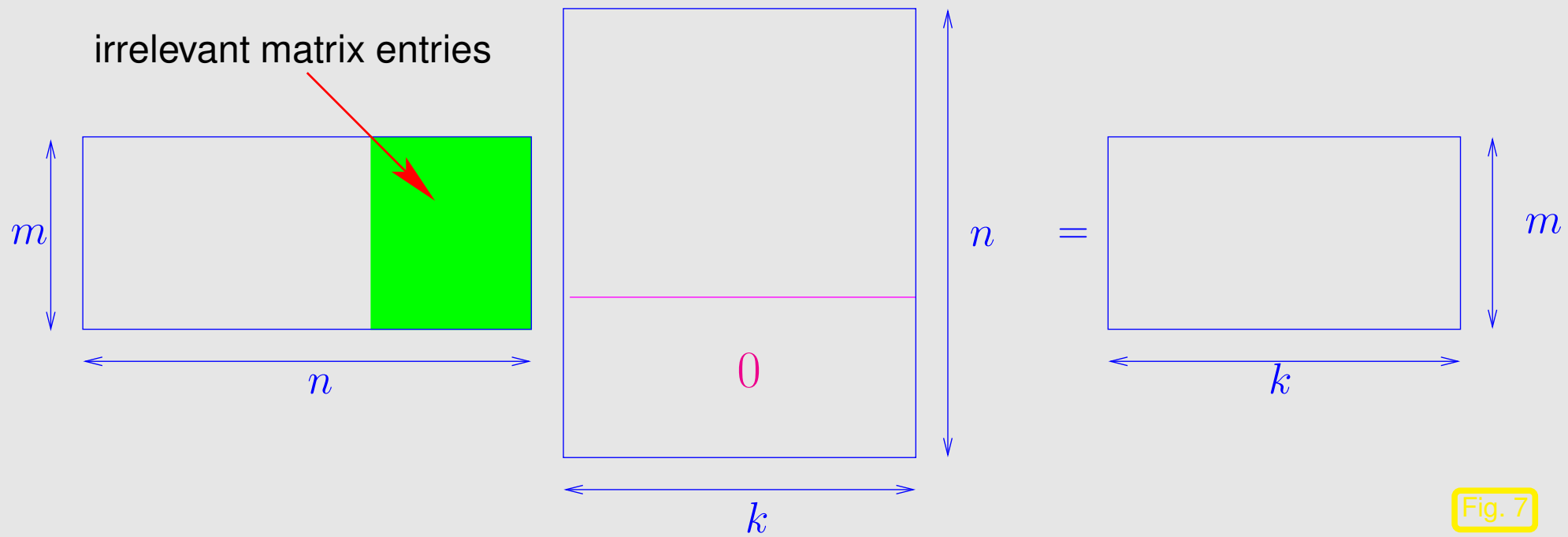
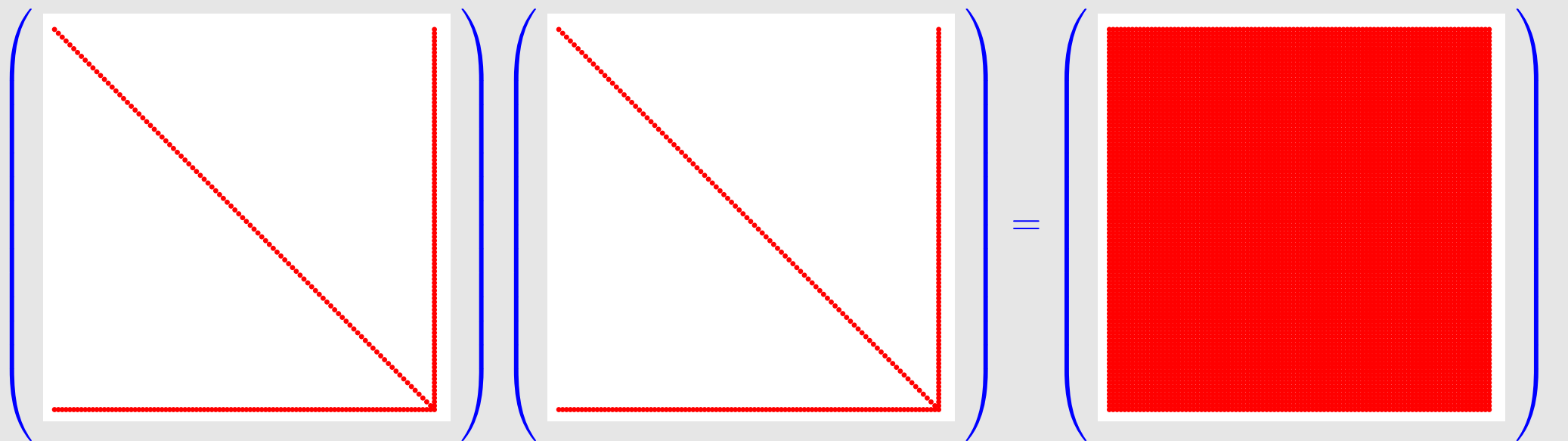
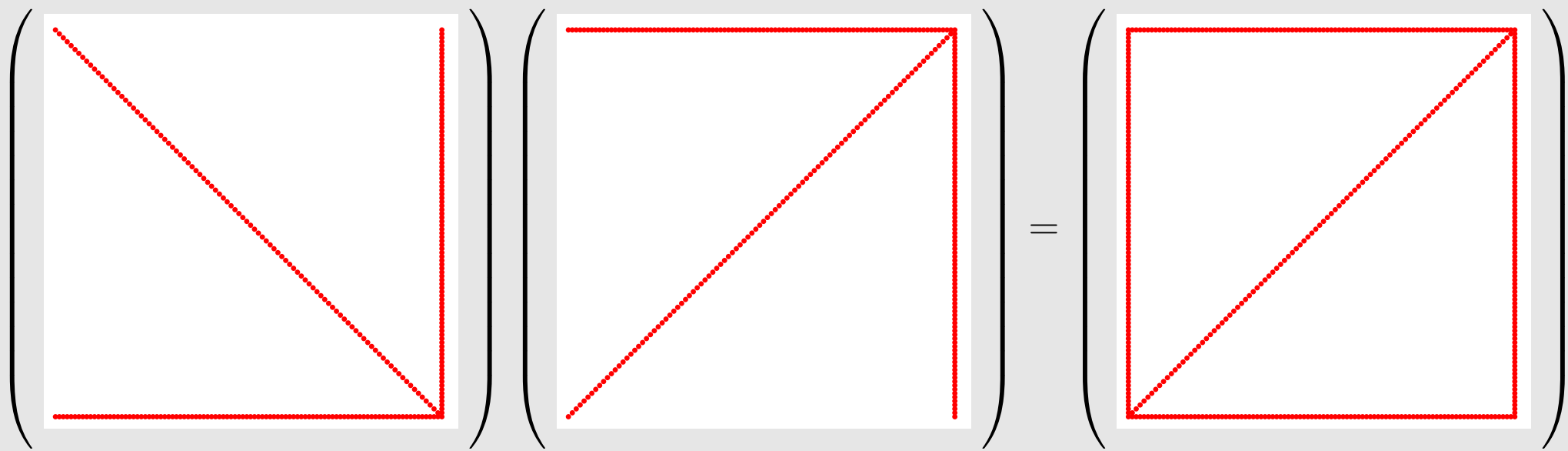


Fig. 6



“Seeing” the structure/pattern of a matrix product:





MATLAB-command for visualizing the structure of a matrix: `spy(M)`

### Code 1.2.5: visualizing structure of matrices

```

1 n = 100; A = [diag(1:n-1), (1:n-1)'; (1:n) ]; B = A(n:-1:1,:);
2 C = A*A; D = A*B;
3 figure; spy(A, 'r'); axis off; print -depsc2 '../PICTURES/Aspy.eps';
4 figure; spy(B, 'r'); axis off; print -depsc2 '../PICTURES/Bspy.eps';
5 figure; spy(C, 'r'); axis off; print -depsc2 '../PICTURES/Cspy.eps';
6 figure; spy(D, 'r'); axis off; print -depsc2 '../PICTURES/Dspy.eps';

```

Remark 1.2.6 (Scalings).

**Scaling** = multiplication with diagonal matrices (with non-zero diagonal entries):

It is important to know the different effect of multiplying with a diagonal matrix from left or right:

• multiplication with diagonal matrix *from left* ➤ **row scaling**

$$\begin{pmatrix} d_1 & 0 & & 0 \\ 0 & d_2 & & 0 \\ & & \dots & \\ 0 & 0 & & d_n \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & & a_{2m} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} = \begin{pmatrix} d_1 a_{11} & d_1 a_{12} & \dots & d_1 a_{1m} \\ d_2 a_{21} & d_2 a_{22} & \dots & d_2 a_{2m} \\ \vdots & & & \vdots \\ d_n a_{n1} & d_n a_{n2} & \dots & d_n a_{nm} \end{pmatrix} = \begin{pmatrix} d_1(\mathbf{A})_{1,:} \\ \vdots \\ d_n(\mathbf{A})_{n,:} \end{pmatrix} .$$

• multiplication with diagonal matrix *from right* ➤ **column scaling**

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & & a_{2m} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \begin{pmatrix} d_1 & 0 & & 0 \\ 0 & d_2 & & 0 \\ & & \dots & \\ 0 & 0 & & d_m \end{pmatrix} = \begin{pmatrix} d_1 a_{11} & d_2 a_{12} & \dots & d_m a_{1m} \\ d_1 a_{21} & d_2 a_{22} & \dots & d_m a_{2m} \\ \vdots & & & \vdots \\ d_1 a_{n1} & d_2 a_{n2} & \dots & d_m a_{nm} \end{pmatrix} \\ = \begin{pmatrix} d_1(\mathbf{A})_{:,1} & \dots & d_m(\mathbf{A})_{:,m} \end{pmatrix} .$$

Multiplication with a scaling matrix  $D = \text{diag}(d_1, \dots, d_n) \in \mathbb{R}^{n,n}$  in MATLAB:

$$y = \text{diag}(d) * x;$$

or

$$y = d .* x;$$

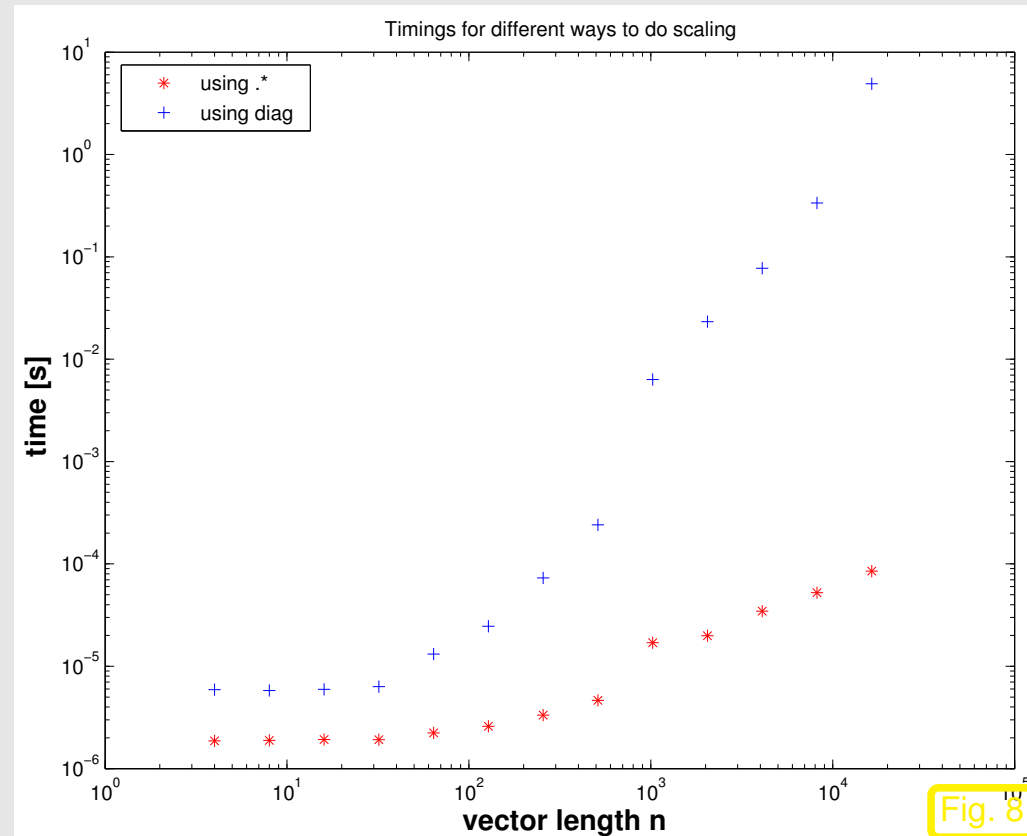


Fig. 8

### Code 1.2.7: Timing multiplication with scaling matrix

```

1 % MATLAB script for timing a smart and foolish way to carry out multiplication
2 % with a scaling matrix in MATLAB, see Rem. 1.2.6.
3 nruns = 3; timings = [];
4 for n=2.(2:14)
5     d = rand(n,1); x = rand(n,1);
6     tbad = realmax; tgood = realmax;
7     for j=1:nruns
8         tic; y = diag(d) * x; tbad = min(tbad, toc);

```

```
9     tic; y = d.*x; tgood = min(tgood,toc);
0     end
1     timings = [timings; n, tgood, tbad];
2 end
3
4 figure('name','scaletimings');
5 loglog(timings(:,1),timings(:,2),'r*',...
6        timings(:,1),timings(:,3),'b+');
7 xlabel('\bf vector length n','fontsize',14);
8 ylabel('\bf time [s]','fontsize',14);
9 title('Timings for different ways to do scaling');
0 legend('using .*','using diag','location','best');
1
2 print -depsc2 '../PICTURES/scaletiming.eps';
```



Example 1.2.8 (Row and column transformations).

Simple operations on rows/columns of matrices, *cf.* what was done in Ex. 1.1.2, can often be expressed as multiplication with special matrices:

Given  $\mathbf{A} \in \mathbb{K}^{n,m}$  obtain  $\mathbf{B}$  by adding row  $(\mathbf{A})_{j,:}$  to row  $(\mathbf{A})_{j+1,:}$ ,  $1 \leq j < n$

realisation through matrix product



$$\mathbf{B} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & 1 & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix} \mathbf{A} .$$

left-multiplication  
right-multiplication

with transformation matrices



row transformations  
column transformations

row/column transformations will play a central role in Sect. 2.1



Recall: rules of matrix multiplication, for all  $\mathbb{K}$ -matrices  $\mathbf{A}, \mathbf{B}, \mathbf{C}$  (of suitable sizes),  $\alpha, \beta \in \mathbb{K}$

associative:  $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$ ,

bi-linear:  $(\alpha\mathbf{A} + \beta\mathbf{B})\mathbf{C} = \alpha(\mathbf{AC}) + \beta(\mathbf{BC})$ ,  $\mathbf{C}(\alpha\mathbf{A} + \beta\mathbf{B}) = \alpha(\mathbf{CA}) + \beta(\mathbf{CB})$ ,

non-commutative:  $\mathbf{AB} \neq \mathbf{BA}$  in general.

*Remark 1.2.9 (Matrix algebra).*

A vector space  $(V, \mathbb{K}, +, \cdot)$ , where  $V$  is additionally equipped with a **bi-linear** and associative “multiplication” is called an **algebra**. Hence, the vector space of square matrices  $\mathbb{K}^{n,n}$  with matrix multiplication is an algebra with *unit element*  $\mathbf{I}$ .



*Remark 1.2.10 (Block matrix product).*

Given matrix dimensions  $M, N, K \in \mathbb{N}$  block sizes  $1 \leq n < N$  ( $n' := N - n$ ),  $1 \leq m < M$

$(m' := M - m), 1 \leq k < K (k' := K - k)$  assume

$$\begin{matrix} \mathbf{A}_{11} \in \mathbb{K}^{m,n} & \mathbf{A}_{12} \in \mathbb{K}^{m,n'} & \mathbf{B}_{11} \in \mathbb{K}^{n,k} & \mathbf{B}_{12} \in \mathbb{K}^{n,k'} \\ \mathbf{A}_{21} \in \mathbb{K}^{m',n} & \mathbf{A}_{22} \in \mathbb{K}^{m',n'} & \mathbf{B}_{21} \in \mathbb{K}^{n',k} & \mathbf{B}_{22} \in \mathbb{K}^{n',k'} \end{matrix},$$

► 
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{pmatrix}. \quad (1.2.11)$$

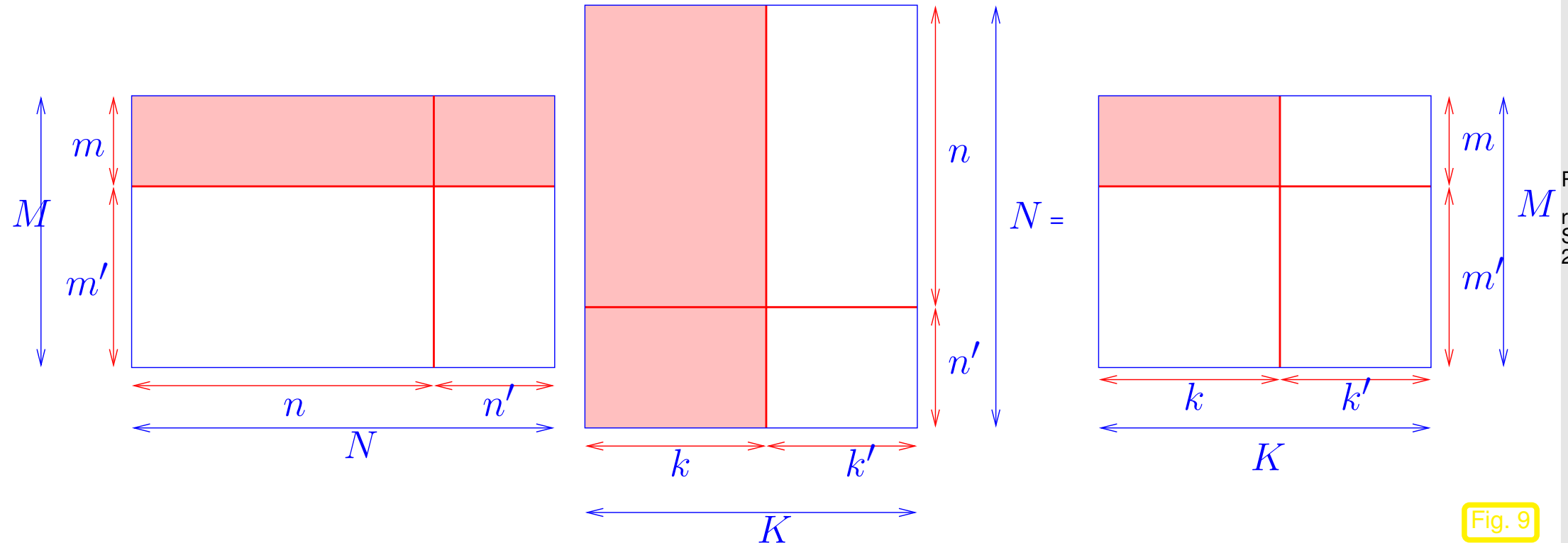


Fig. 9

Bottom line: one can compute with block-structured matrices in *almost* (\*) the same ways as with matrices with real/complex entries, see [51, Sect. 1.3.3].



(\*): you must not use the commutativity of multiplication (because matrix multiplication is not commutative).



## 1.3 Complexity/computational effort

complexity/computational effort of an algorithm  $:\Leftrightarrow$  number of elementary operators


additions/multiplications

Crucial: dependence of (worst case) complexity of an algorithm on (integer) **problem size parameters** (worst case  $\leftrightarrow$  maximum for all possible data)

Usually studied: **asymptotic complexity**  $\hat{=}$  “leading order term” of complexity w.r.t *large* problem size parameters

The usual choice of problem size parameters in numerical linear algebra is the number of independent real variables needed to describe the input data (vector length, matrix sizes).

operation	description	#mul/div	#add/sub	asympt. complexity
dot product	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}^H \mathbf{y}$	$n$	$n - 1$	$O(n)$
tensor product	$(\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x} \mathbf{y}^H$	$nm$	$0$	$O(mn)$
matrix product <sup>(*)</sup>	$(\mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{B} \in \mathbb{R}^{n,k}) \mapsto \mathbf{AB}$	$mnk$	$mk(n - 1)$	$O(mnk)$

 notation (“Landau-O”):  $f(n) = O(g(n)) \Leftrightarrow \exists C > 0, N > 0: |f(n)| \leq Cg(n)$  for all  $n > N$ .

*Remark 1.3.1* (Meaningful “O-bounds” for complexity).

Of course, the definition of the Landau symbol leaves ample freedom for stating meaningless bounds; an algorithm that runs with linear complexity  $O(n)$  can be correctly labelled as possessing  $O(\exp(n))$  complexity.

Yet, whenever the Landau notation is used to describe asymptotic complexities, the bounds have to be sharp in the sense that no function with slower asymptotic growth will be possible inside the  $O$ , for instance

- complexity  $O(n)$  means that the complexity is not  $O(n^\alpha)$  for any  $\alpha < 1$ ,
- complexity  $O(\exp(n))$  excludes asymptotic complexity  $O(n^p)$  for any  $p \in \mathbb{R}$ .



*Remark 1.3.2* (“Fast” matrix multiplication).

(\*) :  $O(mnk)$  complexity bound applies to “straightforward” matrix multiplication according to (1.2.1).

For  $m = n = k$  there are (sophisticated) variants with better asymptotic complexity, e.g., the **divide-and-conquer Strassen algorithm** [62] with asymptotic complexity  $O(n^{\log_2 7})$ :

Start from  $\mathbf{A}, \mathbf{B} \in \mathbb{K}^{n,n}$  with  $n = 2\ell$ ,  $\ell \in \mathbb{N}$ . The idea relies on the block matrix product (1.2.11) with  $\mathbf{A}_{ij}, \mathbf{B}_{ij} \in \mathbb{K}^{\ell,\ell}$ ,  $i, j \in \{1, 2\}$ . Let  $\mathbf{C} := \mathbf{A}\mathbf{B}$  be partitioned accordingly:  $\mathbf{C} = \begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{22} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{pmatrix}$ . Then tedious elementary computations reveal

$$\mathbf{C}_{11} = \mathbf{Q}_0 + \mathbf{Q}_3 - \mathbf{Q}_4 + \mathbf{Q}_6,$$

$$\mathbf{C}_{21} = \mathbf{Q}_1 + \mathbf{Q}_3,$$

$$\mathbf{C}_{12} = \mathbf{Q}_2 + \mathbf{Q}_4,$$

$$\mathbf{C}_{22} = \mathbf{Q}_0 + \mathbf{Q}_2 - \mathbf{Q}_1 + \mathbf{Q}_5,$$

where the  $\mathbf{Q}_k \in \mathbb{K}^{\ell,\ell}$ ,  $k = 1, \dots, 7$  are obtained from

$$\mathbf{Q}_0 = (\mathbf{A}_{11} + \mathbf{A}_{22}) * (\mathbf{B}_{11} + \mathbf{B}_{22}),$$

$$\mathbf{Q}_1 = (\mathbf{A}_{21} + \mathbf{A}_{22}) * \mathbf{B}_{11},$$

$$\mathbf{Q}_2 = \mathbf{A}_{11} * (\mathbf{B}_{12} - \mathbf{B}_{22}),$$

$$\mathbf{Q}_3 = \mathbf{A}_{22} * (-\mathbf{B}_{11} + \mathbf{B}_{21}),$$

$$\mathbf{Q}_4 = (\mathbf{A}_{11} + \mathbf{A}_{12}) * \mathbf{B}_{22},$$

$$\mathbf{Q}_5 = (-\mathbf{A}_{11} + \mathbf{A}_{21}) * (\mathbf{B}_{11} + \mathbf{B}_{12}),$$

$$\mathbf{Q}_6 = (\mathbf{A}_{12} - \mathbf{A}_{22}) * (\mathbf{B}_{21} + \mathbf{B}_{22}).$$

Beside a considerable number of matrix additions ( computational effort  $O(n^2)$  ) it takes only **7** multiplications of matrices of size  $n/2$  to compute  $\mathbf{C}$ ! Strassen's algorithm boils down to the *recursive application* of these formulas for  $n = 2^k$ ,  $k \in \mathbb{N}$ .

A refined algorithm of this type can achieve complexity  $O(n^{2.36})$ , see [12].



Example 1.3.3 (Efficient associative matrix multiplication).

$\mathbf{a} \in \mathbb{K}^m, \mathbf{b} \in \mathbb{K}^n, \mathbf{x} \in \mathbb{K}^n$ :

$$\mathbf{y} = (\mathbf{a}\mathbf{b}^\top)\mathbf{x}. \quad (1.3.4)$$

$$\mathbf{T} = \mathbf{a}\mathbf{b}^\top; \quad \mathbf{y} = \mathbf{T}\mathbf{x};$$

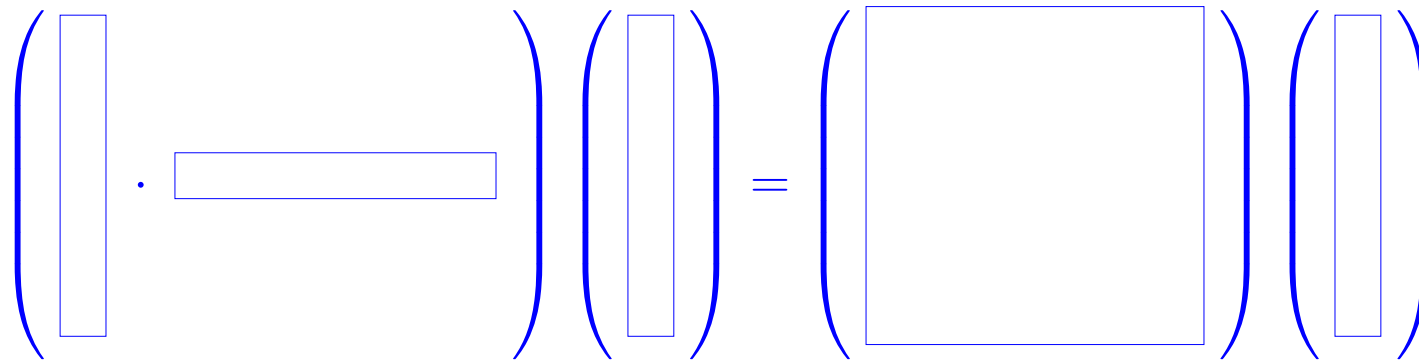
➤ complexity  $O(mn)$

$$\mathbf{y} = \mathbf{a}(\mathbf{b}^\top\mathbf{x}). \quad (1.3.5)$$

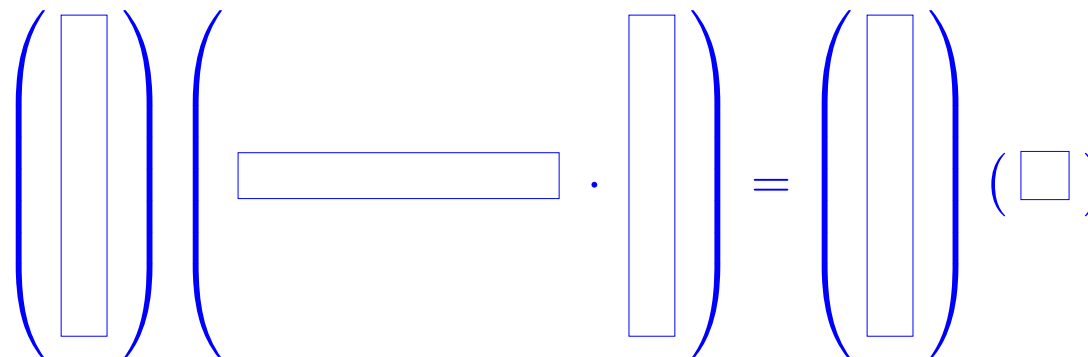
$$\mathbf{t} = \mathbf{b}^\top\mathbf{x}; \quad \mathbf{y} = \mathbf{a}\mathbf{t};$$

➤ complexity  $O(n + m)$  (“linear complexity”)

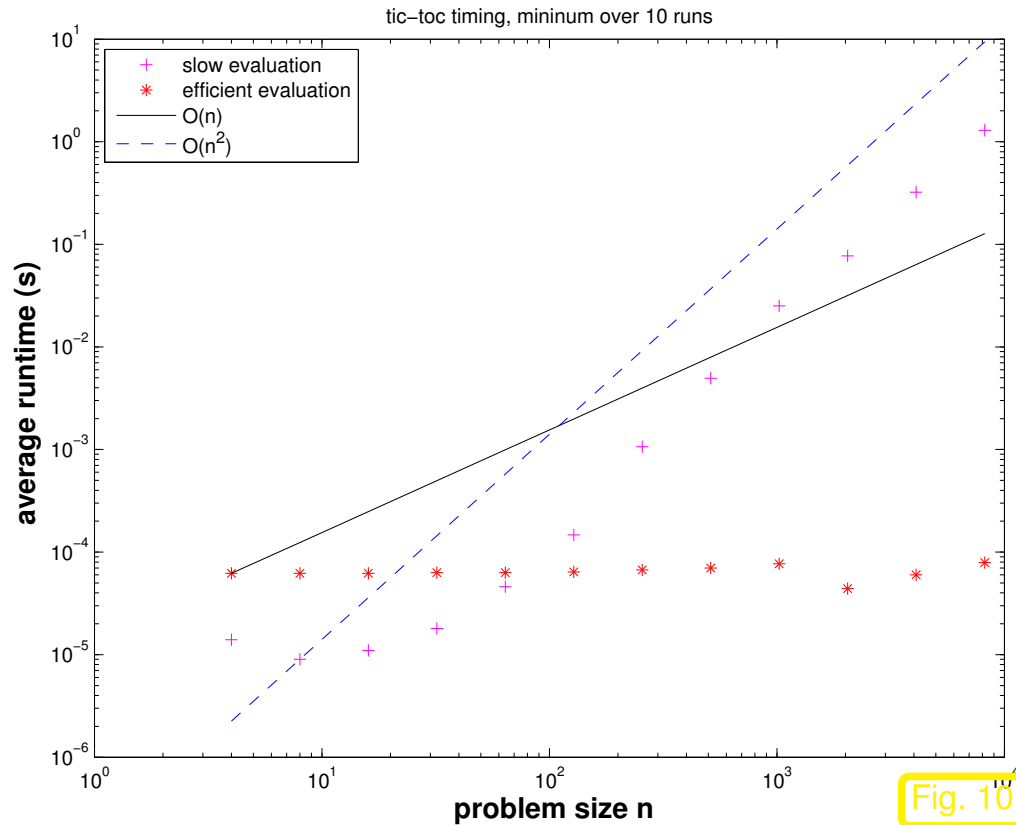
Visualization of evaluation according to (1.3.4):



Visualization of evaluation according to (1.3.5):







◁ average runtimes for efficient/inefficient matrix×vector multiplication with rank-1 matrices ( MATLAB `tic-toc` timing)

Platform:

- MATLAB7.4.0.336 (R2007a)
- Genuine Intel(R) CPU T2500 @ 2.00GHz
- Linux 2.6.16.27-0.9-smp

### Code 1.3.6: MATLAB code for Ex. 1.3.3

```

1 function dottenstiming(N, nruns)
2 % This function compares the runtimes for the multiplication of a vector with a
3 % rank-1 matrix  $\mathbf{ab}^T$ ,  $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$  using different associative
4 % evaluations measurements consider minimal time for several (nruns) runs
5

```

```
6 if (nargin < 1), N = 2.^(2:13); end
7 if (nargin < 2), nruns = 10; end
8
9 times = []; % matrix for storing recorded runtimes
10 for n=N
11     % Initialize dense vectors a, b, x (column vectors!)
12     a = (1:n)'; b = (n:-1:1)'; x = rand(n,1);
13
14     % Measuring times using MATLAB tic-toc commands
15     tfool = realmax; for i=1:nruns, tic; y = (a*b')*x; tfool =
16         min(tfool,toc); end;
17     tsmart = realmax; for i=1:nruns, tic; y = a*dot(b',x); tsmart
18         = min(tsmart,toc); end;
19     times = [times;n, tfool, tsmart];
20 end
21
22 % log-scale plot for investigation of asymptotic complexity
23 figure ('name', 'dottenstiming');
24 loglog(times(:,1),times(:,2), 'm+', ...
25         times(:,1),times(:,3), 'r*', ...
26         times(:,1),times(:,1)*times(1,3)/times(1,1), 'k-', ...
27         times(:,1), (times(:,1).^2)*times(2,2)/(times(2,1)^2), 'b--');
28 xlabel ('{\bf problem size n}', 'fontsize', 14);
```

```

27 ylabel ('{\bf average runtime (s)}', 'fontsize', 14);
28 title ('tic-toc timing, minimum over 10 runs');
29 legend ('slow evaluation', 'efficient evaluation', ...
30         'O(n)', 'O(n^2)', 'location', 'northwest');
31 print -depsc2 '../PICTURES/dottenstiming.eps';

```



*Remark 1.3.7* (Reading off polynomial complexity).

Available: “Measurements”  $t_i = t_i(n_i)$  for different  $n_1, n_2, \dots, n_N, n_i \in \mathbb{N}$

Conjectured: **Algebraic dependence**  $t_i = Cn_i^\alpha, \alpha \in \mathbb{R}$

$$t_i = Cn_i^\alpha \Rightarrow \log(t_i) = \log C + \alpha \log(n_i), \quad i = 1, \dots, N.$$

► If the conjecture holds true, then the points  $(n_i, t_i)$  will lie on a *straight line* with *slope*  $\alpha$  in a **doubly logarithmic plot**.

➤ quick “visual test” of conjectured asymptotic complexity



*Remark 1.3.8* (Relevance of asymptotic complexity).

Runtimes in Ex. 1.3.3 illustrate that the

asymptotic complexity of an algorithm need not be closely correlated with its overall runtime on a particular platform,

because on modern computer architectures with multi-level memory hierarchies the *memory access pattern* may be more important for efficiency than the mere number of floating point operations, see [40].

Then, why do we pay so much attention to asymptotic complexity in this course ?

- To a certain extent, the asymptotic complexity allows to predict the dependence of the runtime *of a particular implementation* of an algorithm on the problem size (for large problems). For instance, an algorithm with asymptotic complexity  $O(n^2)$  is likely to take  $4\times$  as much time when the problem size is doubled.



## 1.4 BLAS

**BLAS** = basic linear algebra subroutines

BLAS provides a library of routines with standardized (FORTRAN 77 style) interfaces. These routines have been implemented efficiently on various platforms and operating systems.

MATLAB heavily relies on BLAS library calls for performing linear algebra operations.

**Example 1.4.1** (Different implementations of matrix multiplication in MATLAB).**Code 1.4.2:** Timing different implementations of matrix multiplication

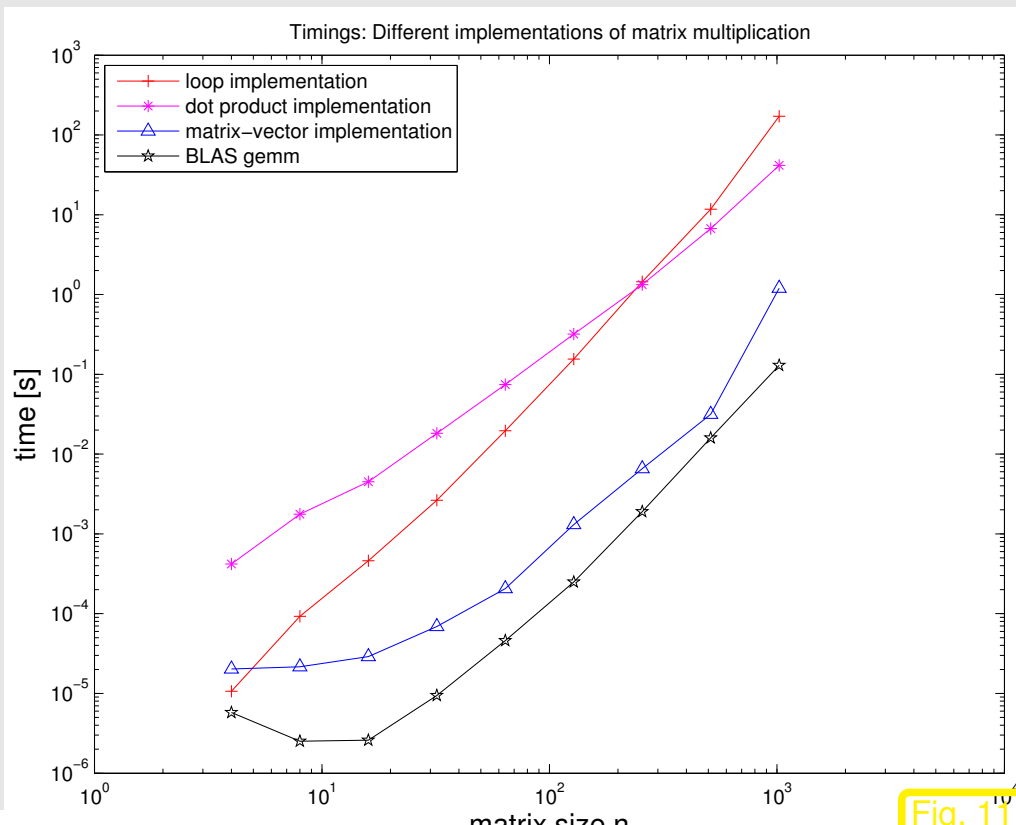
```
1 % MATLAB script for timing different implementations of matrix multiplications
2 nruns = 3; times = [];
3 for n=2.^(2:10) % n = size of matrices
4     fprintf('matrix size n = %d\n',n);
5     A = rand(n,n); B = rand(n,n); C = zeros(n,n);
6     t1 = realmax;
7     % loop based implementation (no BLAS)
8     for l=1:nruns
9         tic;
10        for i=1:n, for j=1:n
11            for k=1:n, C(i,j) = C(i,j) + A(i,k)*B(k,j); end
12        end, end
13        t1 = min(t1,toc);
14    end
15    t2 = realmax;
16    % dot product based implementation (BLAS level 1)
17    for l=1:nruns
18        tic;
19        for i=1:n
20            for j=1:n, C(i,j) = dot(A(i,:),B(:,j)); end
21        end
```

```
22     t2 = min(t2, toc);
23 end
24 t3 = realmax;
25 % matrix-vector based implementation (BLAS level 2)
26 for l=1:nruns
27     tic;
28     for j=1:n, C(:,j) = A*B(:,j); end
29     t3 = min(t3, toc);
30 end
31 t4 = realmax;
32 % BLAS level 3 matrix multiplication
33 for l=1:nruns
34     tic; C = A*B; t4 = min(t4, toc);
35 end
36 times = [ times; n t1 t2 t3 t4];
37 end
38
39 figure ('name', 'mmtiming');
40 loglog (times(:,1), times(:,2), 'r+-', ...
41         times(:,1), times(:,3), 'm*-', ...
42         times(:,1), times(:,4), 'b^-', ...
43         times(:,1), times(:,5), 'kp-');
44 title ('Timings: Different implementations of matrix
```

```

multiplication'
45 xlabel 'matrix size n' 'fontsize'
46 ylabel 'time [s]' 'fontsize'
47 legend 'loop implementation' 'dot product implementation'
48 'matrix-vector implementation' 'BLAS gemm (MATLAB *)'
49 'location' 'northwest'
50
51 print      '../PICTURES/mvtiming.eps'

```



### Platform:

- Mac OS X.6
- Intel Core 7, 2.66 GHz
- L2 256 kB, L3 4 MB, Mem 4 GB
- MATLAB 7.10.0 (R 2010a)

▶ Tremendous gain in execution speed when relying on high-level BLAS routines.





Grouping of BLAS routines (“levels”) according to asymptotic complexity, see [23, Sect. 1.1.12]:

- **Level 1:** vector operations such as scalar products and vector norms.  
asymptotic complexity  $O(n)$ , (with  $n \hat{=}$  vector length),  
e.g.: dot product:  $\rho = \mathbf{x}^\top \mathbf{y}$
- **Level 2:** vector-matrix operations such as matrix-vector multiplications.  
asymptotic complexity  $O(mn)$ , (with  $(m, n) \hat{=}$  matrix size),  
e.g.: matrix  $\times$  vector multiplication:  $\mathbf{y} = \alpha \mathbf{A}\mathbf{x} + \beta \mathbf{y}$
- **Level 3:** matrix operations such as matrix additions or multiplications.  
asymptotic complexity  $O(nmk)$ , (with  $(n, m, k) \hat{=}$  matrix sizes),  
e.g.: matrix product:  $\mathbf{C} = \mathbf{A}\mathbf{B}$

### Syntax of BLAS calls:

The functions have been implemented for different types, and are distinguished by the first letter of the function name. E.g. *sdot* is the dot product implementation for single precision and *ddot* for double precision.

• **BLAS LEVEL 1:** vector operations, asymptotic complexity  $O(n)$ ,  $n \hat{=}$  vector length

- dot product  $\rho = \mathbf{x}^\top \mathbf{y}$

**x**DOT (N, X, INCX, Y, INCY)

- $\mathbf{x} \in \{S, D\}$ , scalar type:  $S \hat{=}$  type float,  $D \hat{=}$  type double
- $N \hat{=}$  length of vector (modulo stride INCX)
- $X \hat{=}$  vector  $\mathbf{x}$ : array of type  $\mathbf{x}$
- INCX  $\hat{=}$  stride for traversing vector  $X$
- $Y \hat{=}$  vector  $\mathbf{y}$ : array of type  $\mathbf{x}$
- INCY  $\hat{=}$  stride for traversing vector  $Y$

- vector operations  $\mathbf{y} = \alpha \mathbf{x} + \mathbf{y}$

**x**AXPY (N, ALPHA, X, INCX, Y, INCY)

- $\mathbf{x} \in \{S, D, C, Z\}$ ,  $S \hat{=}$  type float,  $D \hat{=}$  type double,  $C \hat{=}$  type complex
- $N \hat{=}$  length of vector (modulo stride INCX)
- ALPHA  $\hat{=}$  scalar  $\alpha$
- $X \hat{=}$  vector  $\mathbf{x}$ : array of type  $\mathbf{x}$
- INCX  $\hat{=}$  stride for traversing vector  $X$
- $Y \hat{=}$  vector  $\mathbf{y}$ : array of type  $\mathbf{x}$
- INCY  $\hat{=}$  stride for traversing vector  $Y$

• **BLAS LEVEL 2:** matrix-vector operations, asymptotic complexity  $O(mn)$ ,  $(m, n) \hat{=}$  matrix size

- matrix  $\times$  vector multiplication  $\mathbf{y} = \alpha \mathbf{A}\mathbf{x} + \beta \mathbf{y}$

**x**GEMV (TRANS, M, N, ALPHA, A, LDA, X,  
INCX, BETA, Y, INCY)

- **x**  $\in \{S, D, C, Z\}$ , scalar type: S  $\hat{=}$  type float, D  $\hat{=}$  type double, C  $\hat{=}$  type complex
- M, N  $\hat{=}$  size of matrix **A**
- ALPHA  $\hat{=}$  scalar parameter  $\alpha$
- A  $\hat{=}$  matrix **A** stored in *linear array* of length  $M \cdot N$  (column major arrangement)

$$(\mathbf{A})_{i,j} = A[N * (j - 1) + i] .$$

- LDA  $\hat{=}$  “leading dimension” of  $\mathbf{A} \in \mathbb{K}^{n,m}$ , that is, the number  $n$  of rows.
- X  $\hat{=}$  vector **x**: array of type **x**
- INCX  $\hat{=}$  stride for traversing vector X
- BETA  $\hat{=}$  scalar parameter  $\beta$

- $Y \hat{=}$  vector  $y$ : array of type  $x$
- $INCY \hat{=}$  stride for traversing vector  $Y$
- **BLAS LEVEL 3**: matrix-matrix operations, asymptotic complexity  $O(mnk)$ ,  $(m, n, k) \hat{=}$  matrix sizes
  - matrix  $\times$  matrix multiplication  $C = \alpha AB + \beta C$

$x$ GEMM (TRANSA, TRANSB, M, N, K,  
ALPHA, A, LDA, X, B, LDB,  
BETA, C, LDC)

( meaning of arguments as above)

*Remark 1.4.3* (BLAS calling conventions).

The BLAS calling syntax seems queer in light of modern object oriented programming paradigms, but it is a legacy of FORTRAN77, which was (and partly still is) the programming language, in which the BLAS routines were coded.

It is a very common situation in scientific computing that one has to rely on old codes and libraries implemented in an old-fashioned style.

This example demonstrates the gain in efficiency through BLAS calls from C++. Thus calling BLAS and using high-level numerical linear algebra operations is advisable not only in MATLAB, but also when using regular programming languages. The test case is matrix multiplication conducted in various ways as in Ex. 1.4.1.

### Code 1.4.5: ColumnMajor Matrix Class in C++

```
1  /*  Author: Manfred Quack
2     *  ColumnMajorMatrix.h
3     *  This Class Implements a ColumnMajor Matrix Structure in C++
4     *  - it provides an access operator () using ColumnMajor Access
5     *  - it also provides 4 different implementations of a
6       Matrix-Matrix Multiplication
7     */
7  #include <iostream>
8  #include <cstdlib>
9  #include <stdio.h>
10 #include <cassert>
11 #ifndef _USE_MKL
12 #ifdef _MAC_OS
13 #include <vecLib/cblas.h> //part of Accelerate Framework
14 #endif
```

```
15 #ifdef _LINUX
16 extern "C" {
17 #include <cbblas.h>
18 }
19 #endif
20 #else
21 #include <mkl.h>
22 #endif
23 #include <cmath>
24 typedef double Real;
25 using namespace std;
26
27 class ColumnMajorMatrix {
28 private: //Data Members:
29     Real* data;
30     int n,m;
31 public:
32     //----Class Managment-----:
33     //Constructors
34     ColumnMajorMatrix(int _n, int _m);
35     ColumnMajorMatrix(const ColumnMajorMatrix &B);
36     //Destructor
37     ~ColumnMajorMatrix();
```

```
38 //Assignment operator
39 ColumnMajorMatrix & operator=(const ColumnMajorMatrix &B);
40 //Access for ColumnMajorArrangement
41 inline Real& operator()(int i, int j)
42 { assert(i<n && j<m); return data[n*j+i];}
43 //---Different Implementations for the Multiplication---/
44 // All of these implementations have only been checked for Square
   Matrices
45 //straightforward implementation for a Matrix Multiplication
46 ColumnMajorMatrix standardMultiply( ColumnMajorMatrix &B);
47 //Implementations using DOT-, GEMV- and GEMM from BLAS:
48 ColumnMajorMatrix dotMultiply( ColumnMajorMatrix &B);
49 ColumnMajorMatrix gemvMultiply( ColumnMajorMatrix &B);
50 ColumnMajorMatrix gemmMultiply( ColumnMajorMatrix &B);
51 //----Other Functions-----:
52 //Function to initialize matrix with 1,2,3...
53 void initGrow();
54 void initRand();
55 void print();
56 Real CalcErr(const ColumnMajorMatrix &B);
57 };
```

### Code 1.4.6: A straightforward implementation for Matrix Multiplications in C++

```
1 /* ColumnMajorMatrix_Multiplication: straightforward standard
```

```
implementation for a Matrix Multiplication (3 loops) */
2 #include "ColumnMajorMatrix.h"
3 ColumnMajorMatrix ColumnMajorMatrix::standardMultiply(
  ColumnMajorMatrix &B)
4 {
5   assert(m==B.n); // only support square matrices
6   ColumnMajorMatrix C(n,B.m); //important: must be zero: (done in
  constructor)
7   for (int j=0; j<B.m; ++j)
8     for (int i=0; i<n; ++i)
9       for (int k=0; k<m; ++k)
10        C(i, j) += operator()(i, k) * B(k, j);
11   return C;
12 }
```

Code 1.4.8: Matrix Multiplication using DOT from BLAS and two nested loop, parameters are described above

```
1 /* ColumnMajorMatrix_Multiplication.cpp using the DOT-routine
   from BLAS (and 2 loops) */
2 #include "ColumnMajorMatrix.h"
3 ColumnMajorMatrix ColumnMajorMatrix::dotMultiply(
  ColumnMajorMatrix &B)
```



```

4 {
5     assert (m==B.n);
6     ColumnMajorMatrix C(n,B.m);
7     for (int j=0; j<B.m; ++j)
8         for (int i=0; i<n; ++i)
9             C(i,j)=cblas_ddot(this->m, &(operator()(i,0)), n,
10                &(B(0,j)), 1);
11 return C;
12 }

```

Code 1.4.10: Matrix Multiplication using GEMV from BLAS and one loop, *CblasColMajor* and *Cblas-NoTrans* are cblas specific flags to toggle between column and rowmajor format and transpose matrix. Other *gemv* parameters are described above.

```

1 /* ColumnMajorMatrix_Multiplication using the GEMV-routine from
2    BLAS (+1 loop) */
3 #include "ColumnMajorMatrix.h"
4 ColumnMajorMatrix ColumnMajorMatrix::gemvMultiply(
5     ColumnMajorMatrix &B)
6 {
7     assert (m==B.n);
8     ColumnMajorMatrix C(n,B.m); //important: must be zero: (done in
9     constructor)

```

```

7  double alpha(1.0), beta(1.0);
8  for (int j=0; j<m; ++j)
9      cblas_dgemv(CblasColMajor, CblasNoTrans, m, n, alpha, data,
10             n, &(B(0,j)), 1, beta, &C(0,j), 1);
11 return C;

```

Code 1.4.12: Matrix Multiplication using GEMM from BLAS, *CblasColMajor* and *CblasNoTrans* are *cblas* specific flags to toggle between column and rowmajor format and transpose matrix. Other *gemm* parameters are described above.

```

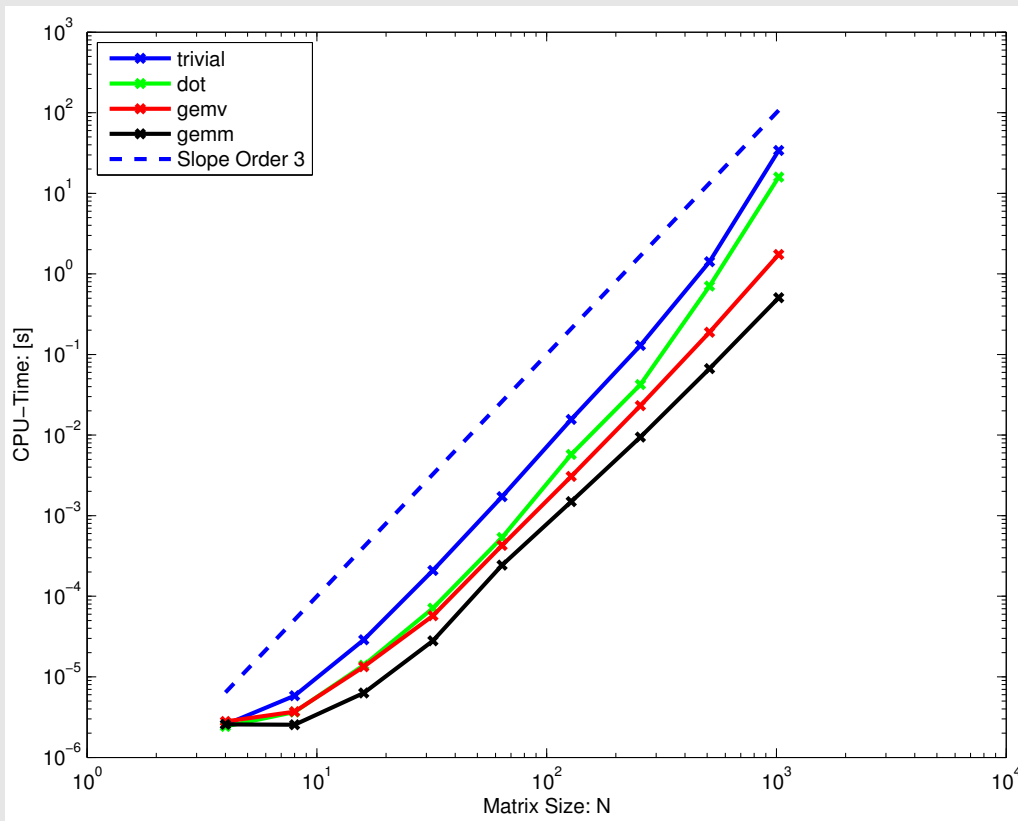
1  /* ColumnMajorMatrix_Multiplication using the GEMM-routine from
   BLAS */
2  #include "ColumnMajorMatrix.h"
3  ColumnMajorMatrix ColumnMajorMatrix::gemmMultiply(
   ColumnMajorMatrix &B)
4  {
5      assert (m==B.n);
6      ColumnMajorMatrix C(n,B.m); //important: must be zero: (done in
   constructor)
7      double alpha(1.0), beta(1.0);
8      cblas_dgemm ( CblasColMajor, CblasNoTrans, CblasNoTrans, n, m,
   B.m, alpha, data, n, B.data, B.n, beta, C.data, C.n );
9      return C;

```

## Code 1.4.13: Timings of different Matrix Multiplications in C++

```
10 }
1 #include <iostream>
2 #include <cstdio>
3 #include <cstdlib>
4 #include <cassert>
5 #include "simpleTimer.h"
6 #include "unixTimer.h"
7 #include "ColumnMajorMatrix.h"
8 /* Main Routine for the timing of different
9  * Matrix Matrix Multiplication implementations */
10 int main (int argc, char * const argv[]) {
11
12     double T0(1e20), T1(1e20), T2(1e20), T3(1e20);
13     simpleTimer watch;
14     int rep(1), n(5);
15     if (argc>1) n=atoi(argv[1]);
16     if (argc>2) rep=atoi(argv[2]);
17     //Declare Input Data
18     ColumnMajorMatrix A(n,n);
19     A.initRand(); //A.initGrow();
20     ColumnMajorMatrix B(A);
21     //The Results:
```

```
22 ColumnMajorMatrix C(n,n),D(n,n),E(n,n),F(n,n);
23 //loop for repetitions (always take timing results over several
    measurements!)
24 for (int r=0;r<rep;++r)
25 {
26 watch.start(); C=A.standardMultiply(B);
    T0=std::min(T0,watch.getTime()); watch.reset();
27 watch.start(); D=A.dotMultiply(B);
    T1=std::min(T1,watch.getTime()); watch.reset();
28 watch.start(); E=A.gemvMultiply(B);
    T2=std::min(T2,watch.getTime()); watch.reset();
29 watch.start(); F=A.gemmMultiply(B);
    T3=std::min(T3,watch.getTime()); watch.reset();
30 }
31 printf("Timing Results: (min. of : %i Repetitions) \n",rep);
32 printf("N: %i StraightForward: %g \n",n,T0);
33 printf("N: %i dotMultiply: %g ,error: %g
    \n",n,T1,D.CalcErr(C));
34 printf("N: %i gemvMultiply: %g, error: %g
    \n",n,T2,E.CalcErr(C));
35 printf("N: %i gemmMultiply: %g, error: %g
    \n",n,T3,F.CalcErr(C));
36 }
```



◁ timings for different implementations of matrix multiplication (see C++-codes above)

OS: Mac OS X

Processor: Intel Core 2 Duo 2GB 667 MHz DDR2  
SDRAM

Compiler: intel v.11.0 (-O3 option)

◇ R. Hiptmair  
rev 38286,  
September  
22, 2011

## Available BLAS implementations:

Below a list of the most common BLAS implementations:

- Reference implementations in C and Fortran (open-source):

<http://www.netlib.org/blas/>

- ATLAS: Automatically tuned linear algebra software (open-source):

<http://math-atlas.sourceforge.net/>

- uBLAS: generic C++ template library (part of Boost)  
[www.boost.org](http://www.boost.org)
- Intel MKL: vendor-specific implementation

## Installation:

- Linux distributions: ATLAS is available in many package-management systems.  
e.g. in Ubuntu, type: `sudo apt-get install libatlas-base-dev`
- Mac OS: BLAS is part of the Accelerate framework which comes with the Developer Tools:  
<http://developer.apple.com/technology/xcode.html>
- Windows: for the exercises it is recommend to use a linux emulator like cygwin or VirtualBox:  
<http://www.virtualbox.org/> <http://www.cygwin.com/>

## Time Measurement:

In order to compare the efficiency of different implementations we need to be able to measure the time spent on a computation. The following definitions are commonly used in this context:

- the *wall time* or *real time* denotes the time an observer would measure between the program start and end. (c.f. wall clock)

- the *user time* denotes the cpu-time spent in executing the user's code.
- the *system time* denotes the cpu-time that the system spent on behalf of the user's code (e.g. memory allocation, i/o handling etc.)

Unix-based systems provide the *time* command for measuring the time of a whole runnable, e.g.: *time ./runnable*. For the measurement of the runtimes in c++, the *clock()*-command provided in the *time.h* can be used. These methods will not provide correct results for the time-measurement of parallelized code, where the routines from the parallelization framework should be used. (e.g. *MPI\_WTIME* for MPI-programs)

#### Code 1.4.14: Measuring CPU time from C++ using clock command

```
1 #include <iostream>
2 #include <time.h> //header for clock()
3
4 /*
5  * simple Timer Class, using clock()-command from the time.h
6  * (should work on all platforms)
7  * this class will only report the cputime (not walltime)
8  */
9 class simpleTimer {
10 public:
11     simpleTimer():time(0),bStarted(false)
```

```
11     {}
12 void start ()
13 {
14     time=clock ();
15     bStarted=true;
16 }
17 double getTime ()
18 {
19     assert (bStarted);
20     return (clock () - time) / (double)CLOCKS_PER_SEC;;
21 }
22 void reset ()
23 {
24     time=0;
25     bStarted=false;
26 }
27 private:
28     double time;
29     bool bStarted;
30 };
```

Code 1.4.15: Measuring real, user and system time from C++ on unix based systems



```
2 * unixTimer Class using times()-command from the unixbased
   times.h
3 * this class will report the user, system and real time.
4 */
5 #include <sys/param.h>
6 #include <sys/times.h>
7 #include <sys/types.h>
8 class unixtimer {
9 public:
10     unixtimer():utime(0),stime(0),rtime(0),bStarted(false)
11     {}
12
13     void start() {rt0=times(&t0);    bStarted=true; }
14
15     double stop() {
16         tms t1;
17         long rt1;
18         assert(bStarted);
19         rt1=times(&t1);
20         utime=((double)(t1.tms_utime-t0.tms_utime))/
            CLOCKS_PER_SEC*10000;
21         stime=((double)(t1.tms_stime-t0.tms_stime))/
            CLOCKS_PER_SEC*10000;
```

```
22     rtime=((double) (rt1-rt0)) / CLOCKS_PER_SEC*10000;  
23     bStarted=false;  
24     return rtime;  
25 }  
  
26  
27 double user() { assert(!bStarted); return utime;}  
28 double system(){assert(!bStarted); return stime;}  
29 double real () {assert(!bStarted); return rtime;}  
30  
31 private:  
32     double utime,stime,rtime;  
33     tms t0;  
34     long rt0;  
35     bool bStarted;  
36 };
```

# 2

## Direct Methods for Linear Systems of Equations

### Learning outcomes:

- Familiarity with the algorithm of Gaussian elimination without and with (partial) pivoting.
- Understanding the rationale behind pivoting and knowing when it can be dispensed with.
- Knowledge about various matrix factorizations like LU, QR, and Cholesky and their use for solving linear systems of equations.
- Awareness of the computational cost of Gaussian elimination and matrix factorizations.
- Sparse matrices: data structures, exploiting sparsity, and problems haunting factorizations of sparse matrices.

The fundamental task:

Given : square matrix  $\mathbf{A} \in \mathbb{K}^{n,n}$ , vector  $\mathbf{b} \in \mathbb{K}^n$ ,  $n \in \mathbb{K}$

Sought : solution vector  $\mathbf{x} \in \mathbb{K}^n$ :  $\mathbf{Ax} = \mathbf{b}$  ← (square) linear system of equations (LSE)  
(ger.: lineares Gleichungssystem)

(Terminology:  $\mathbf{A} \hat{=}$  system matrix,  $\mathbf{b} \hat{=}$  right hand side, ger.: Rechte-Seite-Vektor )

Linear systems of equations are ubiquitous in computational science: they are encountered

- with discrete linear models in network theory (see Ex. 2.6.3), control, statistics;
- in the case of *discretized* boundary value problems for ordinary and partial differential equations (→ course “Numerical methods for partial differential equations”, 4th semester);
- as a result of linearization (e.g, “Newton’s method” → Sect. 4.4).

# Theory

Known from linear algebra [48, Sect. 1.2], [27, Sect. 1.3]:

**Definition 2.0.1** (Invertible matrix).  $\rightarrow$  [48, Sect. 2.3]

$$\mathbf{A} \in \mathbb{K}^{n,n} \text{ invertible / regular} \Leftrightarrow \exists_! \mathbf{B} \in \mathbb{K}^{n,n}: \mathbf{AB} = \mathbf{BA} = \mathbf{I}.$$

$\mathbf{B} \hat{=} \textit{inverse of } \mathbf{A}$ , (pencil notation  $\mathbf{B} = \mathbf{A}^{-1}$ )

**Definition 2.0.2** (Rank of a matrix).  $\rightarrow$  [48, Sect. 2.4], [51, Sect. 1.5]

The *rank* of a matrix  $\mathbf{M} \in \mathbb{K}^{m,n}$ , denoted by  $\text{rank}(\mathbf{M})$ , is the maximal number of linearly independent rows/columns of  $\mathbf{M}$ .

**Theorem 2.0.3** (Criteria for invertibility of matrix).  $\rightarrow$  [48, Sect. 2.3 & Cor. 3.8]

A matrix  $\mathbf{A} \in \mathbb{K}^{n,n}$  is *invertible/regular* if one of the following equivalent conditions is satisfied:

1.  $\exists \mathbf{B} \in \mathbb{K}^{n,n}$ :  $\mathbf{BA} = \mathbf{AB} = \mathbf{I}$ ,
2.  $\mathbf{x} \mapsto \mathbf{Ax}$  defines an endomorphism of  $\mathbb{K}^n$ ,
3. the columns of  $\mathbf{A}$  are linearly independent (full column rank),
4. the rows of  $\mathbf{A}$  are linearly independent (full row rank),
5.  $\det \mathbf{A} \neq 0$  (non-vanishing determinant),
6.  $\text{rank}(\mathbf{A}) = n$  (full rank).

Formal way to denote solution of LSE:

$$\mathbf{A} \in \mathbb{K}^{n,n} \text{ regular} \quad \& \quad \mathbf{Ax} = \mathbf{b} \quad \Rightarrow \quad \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}.$$

matrix inverse

MATLAB: inverse of a matrix  $\mathbf{A}$  available through `inv(A)`



Always avoid computing the inverse of a matrix (which can almost always be avoided)!

In particular, never ever even contemplate using  $x = \text{inv}(A) * b$  to solve the linear system of equations  $Ax = b$ . The next sections present a sound way to do this.

## 2.1 Gaussian Elimination

! Exceptional feature of linear systems of equations (LSE):

☞ “exact” solution computable with finitely many elementary operations

Algorithm: **Gaussian elimination** (→ secondary school, linear algebra,)

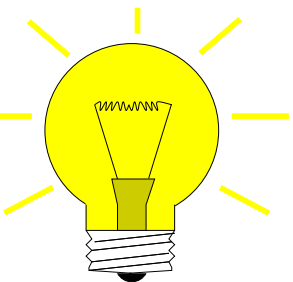
Familiarity with the algorithm of Gaussian elimination for a square linear system of equations will be taken for granted.

In case you cannot remember it, very detailed accounts and examples can be found in

- M. Gutknecht's lecture notes [27, Ch. 1],
- the textbook by Nipp & Stoffer [48, Ch. 1],
- the numerical analysis text by Quarteroni et al. [51, Sects. 3.2 & 3.3],

and, to some extent, below, see Ex. 2.1.1.

Wikipedia: Although the method is named after mathematician **Carl Friedrich Gauss**, the earliest presentation of it can be found in the important Chinese mathematical text *Jiuzhang suanshu* or The Nine Chapters on the Mathematical Art, dated approximately 150 B.C.E, and commented on by **Liu Hui** in the 3rd century.



Idea: transformation to “simpler”, but equivalent LSE by means of successive (invertible) *row transformations*



Ex. 1.2.8: row transformations  $\leftrightarrow$  left-multiplication with transformation matrix

Obviously, left multiplication with a regular matrix does not affect the solution of an LSE: for any regular  $\mathbf{T} \in \mathbb{K}^{n,n}$

$$\mathbf{Ax} = \mathbf{b} \Rightarrow \mathbf{A}'\mathbf{x} = \mathbf{b}' \quad , \text{ if } \mathbf{A}' = \mathbf{TA}, \mathbf{b}' = \mathbf{Tb} .$$

So we may try to convert the linear system of equations to a form that can be solved more easily by multiplying with regular matrices from left, which boils down to applying row transformations. A suitable target format is a diagonal linear system of equations, for which all equations are completely decoupled. This is the gist of Gaussian elimination.

*Example 2.1.1* (Gaussian elimination).

① (Forward) elimination:

$$\begin{pmatrix} 1 & 1 & 0 \\ 2 & 1 & -1 \\ 3 & -1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ -3 \end{pmatrix} \iff \begin{array}{rcl} x_1 + x_2 & = & 4 \\ 2x_1 + x_2 - x_3 & = & 1 \\ 3x_1 - x_2 - x_3 & = & -3 \end{array} .$$

$$\begin{pmatrix} 1 & 1 & 0 \\ 2 & 1 & -1 \\ 3 & -1 & -1 \end{pmatrix} \begin{pmatrix} 4 \\ 1 \\ -3 \end{pmatrix} \quad \begin{pmatrix} \mathbf{1} & 1 & 0 \\ 0 & -1 & -1 \\ 3 & -1 & -1 \end{pmatrix} \begin{pmatrix} 4 \\ -7 \\ -3 \end{pmatrix} \quad \begin{pmatrix} \mathbf{1} & 1 & 0 \\ 0 & -1 & -1 \\ 0 & -4 & -1 \end{pmatrix} \begin{pmatrix} 4 \\ -7 \\ -15 \end{pmatrix}$$

$$\underbrace{\begin{pmatrix} 1 & 1 & 0 \\ 0 & \mathbf{-1} & -1 \\ 0 & 0 & 3 \end{pmatrix}}_{=U} \begin{pmatrix} 4 \\ -7 \\ 13 \end{pmatrix}$$

 = pivot row, pivot element **bold**.

▶ transformation of LSE to **upper triangular form**

② Solve by **back substitution**: back substitution = Rücksubstitution

$$\begin{aligned} x_1 + x_2 &= 4 & x_3 &= \frac{13}{3} \\ -x_2 - x_3 &= -7 & \Rightarrow x_2 &= 7 - \frac{13}{3} = \frac{8}{3} \\ 3x_3 &= 13 & x_1 &= 4 - \frac{8}{3} = \frac{4}{3} \end{aligned}$$

More detailed examples: [27, Sect. 1.1], [48, Sect. 1.1].

More general:

$$\begin{array}{cccccccc}
 a_{11} x_1 & + & a_{12} x_2 & + & \cdots & + & a_{1n} x_n & = & b_1 \\
 a_{21} x_1 & + & a_{22} x_2 & + & \cdots & + & a_{2n} x_n & = & b_2 \\
 \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
 \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
 a_{n1} x_1 & + & a_{n2} x_2 & + & \cdots & + & a_{nn} x_n & = & b_n
 \end{array}$$

- $i$ -th row -  $l_{i1} \cdot$  1st row (**pivot row**),  $l_{i1} := a_{i1}/a_{11}$ ,  $i = 2, \dots, n$

$$\begin{array}{cccccccc}
 a_{11} x_1 & + & a_{12} x_2 & + & \cdots & + & a_{1n} x_n & = & b_1 \\
 a_{22}^{(1)} x_2 & + & \cdots & + & a_{2n}^{(1)} x_n & = & b_2^{(1)} \\
 \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
 \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
 a_{n2}^{(1)} x_2 & + & \cdots & + & a_{nn}^{(1)} x_n & = & b_n^{(1)}
 \end{array}
 \quad \text{with} \quad
 \begin{array}{l}
 a_{ij}^{(1)} = a_{ij} - a_{1j} l_{i1}, \quad i, j = 2, \dots, n \\
 b_i^{(1)} = b_i - b_1 l_{i1}, \quad i = 2, \dots, n.
 \end{array}$$

- $i$ -th row -  $l_{i1} \cdot$  2nd row (**pivot row**),  $l_{i2} := a_{i2}^{(1)}/a_{22}^{(1)}$ ,  $i = 3, \dots, n$ .

$$\begin{array}{cccccccc}
 a_{11} x_1 & + & a_{12} x_2 & + & a_{13} x_3 & + & \cdots & + & a_{1n} x_n & = & b_1 \\
 a_{22}^{(1)} x_2 & + & a_{23}^{(1)} x_3 & + & \cdots & + & a_{2n}^{(1)} x_n & = & b_2^{(1)} \\
 & & a_{33}^{(2)} x_3 & + & \cdots & + & a_{3n}^{(2)} x_n & = & b_3^{(2)} \\
 & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
 & & a_{n2}^{(2)} x_2 & + & \cdots & + & a_{nn}^{(2)} x_n & = & b_n^{(2)}
 \end{array}$$

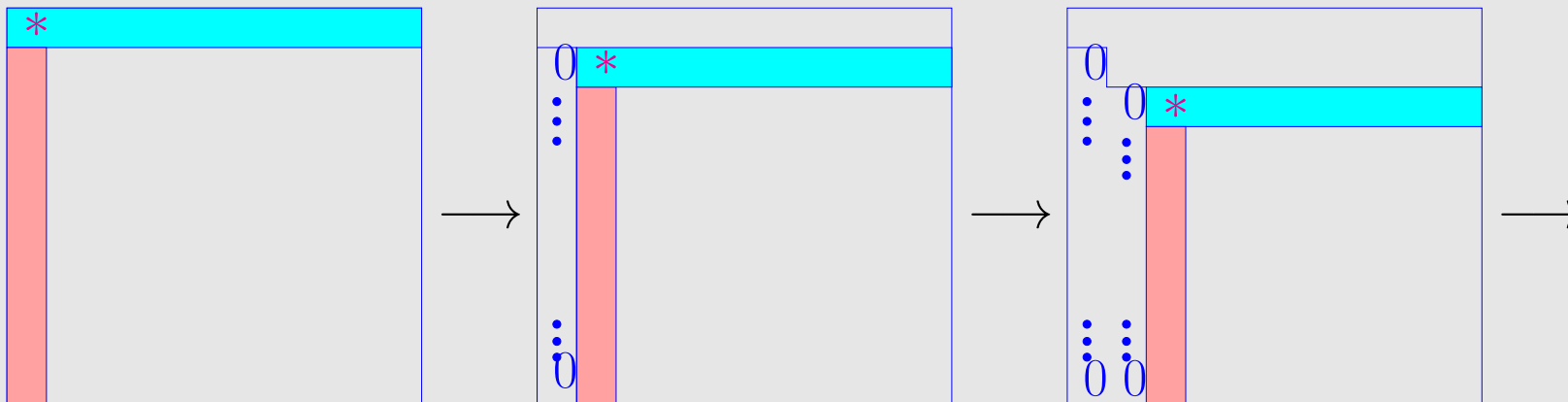
After  $n - 1$  steps:

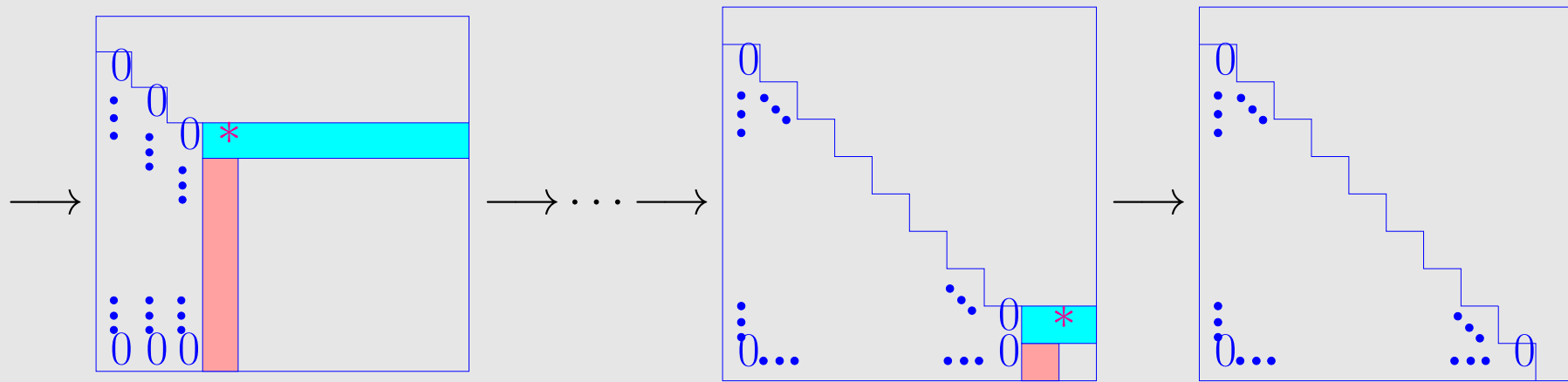
linear systems of equations in **upper triangular form**

$$\begin{array}{rcccccccc}
 a_{11} x_1 + a_{12} x_2 + a_{13} x_3 + \cdots + a_{1n} x_n & = & b_1 \\
 & a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 + \cdots + a_{2n}^{(1)} x_n & = & b_2^{(1)} \\
 & & a_{33}^{(2)} x_3 + \cdots + a_{3n}^{(2)} x_n & = & b_3^{(2)} \\
 & & & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\
 & & & & & & & \vdots & \vdots & \vdots \\
 & & & & & & & & a_{nn}^{(n-1)} x_n & = & b_n^{(n-1)}
 \end{array}$$

Terminology:  $a_{11}, a_{22}^{(1)}, a_{33}^{(2)}, \dots, a_{n-1,n-1}^{(n-2)}$  = **pivots/pivot elements**

Graphical depiction:





\*  $\hat{=}$  pivot (necessarily  $\neq 0$   $\rightarrow$  here: assumption),  = pivot row

In  $k$ -th step (starting from  $\mathbf{A} \in \mathbb{K}^{n,n}$ ,  $1 \leq k < n$ , pivot row  $\mathbf{a}_k^\top$ ):

transformation:  $\mathbf{Ax} = \mathbf{b} \rightarrow \mathbf{A}'\mathbf{x} = \mathbf{b}'$ .

with

$$a'_{ij} := \begin{cases} a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj} & \text{for } k < i, j \leq n, \\ 0 & \text{for } k < i \leq n, j = k, \\ a_{ij} & \text{else,} \end{cases} \quad b'_i := \begin{cases} b_i - \frac{a_{ik}}{a_{kk}} b_k & \text{for } k < i \leq n, \\ b_i & \text{else.} \end{cases} \quad (2.1.2)$$

multipliers  $l_{ik}$

Code 2.1.4: Solving LSE  $Ax = b$  with Gaussian elimination*Algorithm 2.1.3.*

Direct MATLAB implementation of Gaussian elimination for LSE  $Ax = b$ : grossly inefficient!

Line 4: right hand side vector set as last column of matrix, facilitates simultaneous row transformations of matrix and r.h.s.

Variable  $fac \hat{=}$  multiplier

▷ Line 19: extract solution from last column of transformed matrix.

```

1 function x = gausselimsolve(A,b)
2 % Gauss elimination without pivoting, x = A\b
3 % A must be an n x n-matrix, b an n-vector
4 n = size(A,1); A = [A,b]; %
5 % Forward elimination (cf. step ① in Ex. 2.1.1)
6 for i=1:n-1, pivot = A(i,i);
7   for k=i+1:n, fac = A(k,i)/pivot;
8     A(k,i+1:n+1) = A(k,i+1:n+1) -
9       fac*A(i,i+1:n+1); %
9   end
10 end
11 % Back substitution (cf. step ② in Ex. 2.1.1)
12 A(n,n+1) = A(n,n+1) /A(n,n);
13 for i=n-1:-1:1
14   for l=i+1:n
15     A(i,n+1) = A(i,n+1) -
16       A(l,n+1)*A(i,l);
16   end
17   A(i,n+1) = A(i,n+1)/A(i,i);
18 end
19 x = A(:,n+1); %

```

Forward elimination: three nested loops (note: compact vector operation in line 8 involves another loop from  $i + 1$  to  $m$ )

Back substitution: two nested loops

computational cost ( $\leftrightarrow$  number of elementary operations) of Gaussian elimination [48, Sect. 1.3]:

$$\text{elimination : } \sum_{i=1}^{n-1} (n-i)(2(n-i)+3) = n(n-1)\left(\frac{2}{3}n + \frac{7}{6}\right) \text{ Ops. ,} \quad (2.1.5)$$

$$\text{back substitution : } \sum_{i=1}^n 2(n-i) + 1 = n^2 \text{ Ops. .}$$

asymptotic complexity ( $\rightarrow$  Sect. 1.3) of Gaussian elimination  
(without pivoting) for generic LSE  $\mathbf{Ax} = \mathbf{b}$ ,  $\mathbf{A} \in \mathbb{R}^{n,n}$   $= \frac{2}{3}n^3 + O(n^2) = O(n^3)$

*Example 2.1.6* (Runtime of Gaussian elimination).

## Code 2.1.7: Measuring runtimes of Code 2.1.6 vs. MATLAB \-operator

```

1 % MATLAB script for timing numerical solution of linear systems
2 nruns = 3; times = [];
3 for n = 2.^(3:12)
4     fprintf('Matrix size n = %d\n',n);
5     % Initialized random matrix and right hand side
6     A = rand(n,n) + n*eye(n); b = rand(n,1);
7     t1 = realmax; t2 = realmax;
8     for j=1:nruns
9         tic; x1 = gausselimsolve(A,b); t1 = min(t1, toc);
10        tic; x2 = A\b; t2 = min(t2, toc);
11        norm(x1-x2),
12    end
13    times = [times; n t1 t2];
14 end
15
16 figure('name','gausstiming');
17 loglog(times(:,1),times(:,2),'r+',times(:,1),times(:,3),'m-*',...
18        times(:,1),times(:,1).^3*(1E-5/(times(1,1)^3)),'k-');
19 xlabel('matrix size n','fontsize',14);
20 ylabel('execution time [s]','fontsize',14);
21 legend('gausselimsolve','backslash
    solve','O(n^3)','location','northwest');

```



```
print -depsc2 '../PICTURES/gausstiming.eps';
```

MATLAB \

▷ based on LAPACK

▷ based on BLAS (→ Sect. 1.4)

▶ \ about two orders of magnitude faster than a direct implementation

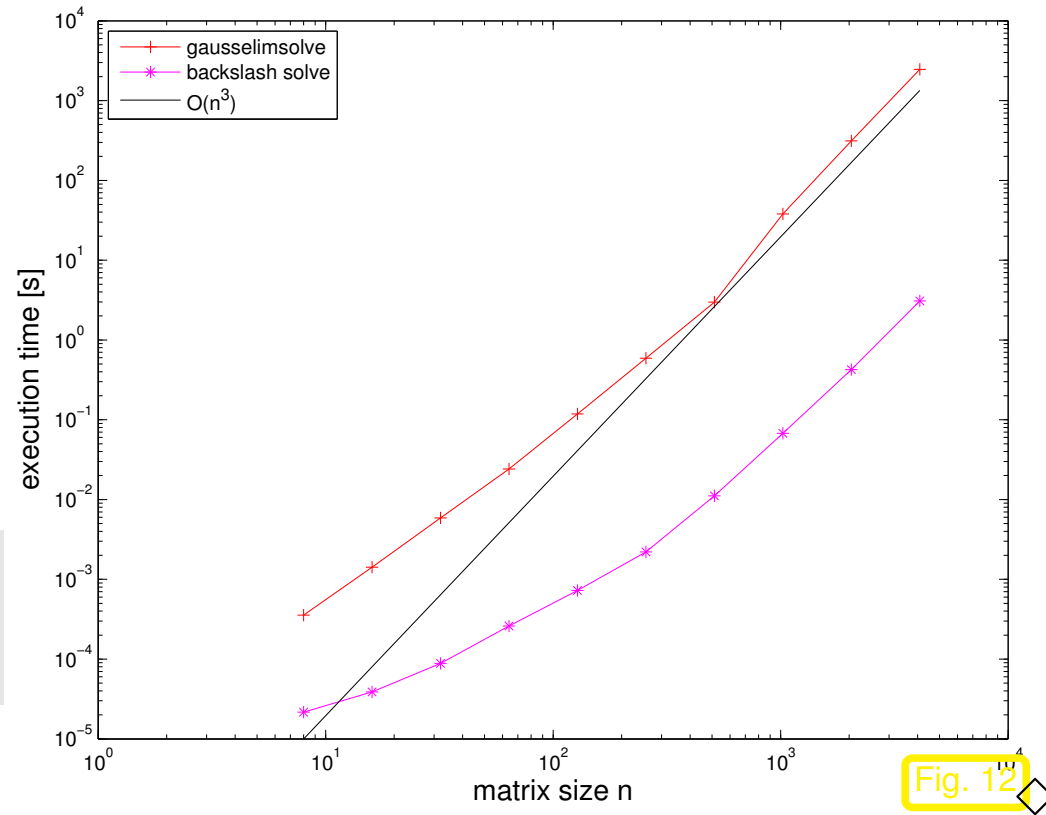


Fig. 12

**Never implement Gaussian elimination yourself !**

**use numerical libraries (LAPACK) or MATLAB ! (MATLAB operator: \)**

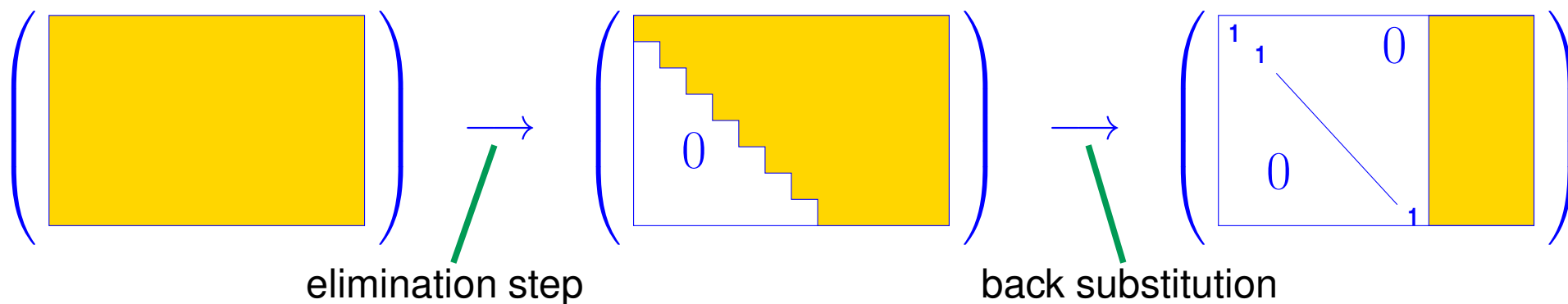
A concise list of libraries for numerical linear algebra and related problems can be found here.

*Remark 2.1.8* (Gaussian elimination for non-square matrices).

In Code 2.1.6: the right hand side vector  $\mathbf{b}$  was first appended to matrix  $\mathbf{A}$  as rightmost column, and then forward elimination and back substitution were carried out on the resulting matrix.

➤ Gaussian elimination for  $\mathbf{A} \in \mathbb{K}^{n,n+1}$ !

“fat matrix”:  $\mathbf{A} \in \mathbb{K}^{n,m}$ ,  $m > n$ :



Recall Code 2.1.6 ( $m = n + 1$ ): the solution vector  $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$  was recovered as the rightmost column of the augmented matrix  $(\mathbf{A}, \mathbf{b})$  after forward elimination and back substitution. In the above cartoon it would be contained in the yellow part of the matrix on the right.

Simultaneous solving of  
LSE with multiple right hand sides

Given regular  $\mathbf{A} \in \mathbb{K}^{n,n}$ ,  $\mathbf{B} \in \mathbb{K}^{n,k}$ ,  
seek  $\mathbf{X} \in \mathbb{K}^{n,k}$

$$\mathbf{AX} = \mathbf{B} \Leftrightarrow \mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$$

MATLAB:

$$\mathbf{X} = \mathbf{A} \backslash \mathbf{B};$$

asymptotic complexity:  $O(n^2(n+k))$

### Code 2.1.9: Gaussian elimination with multiple r.h.s.

```

1 function X = gausselimsolvemult(A,B)
2 % Gauss elimination without pivoting, X = A\B
3 n = size(A,1); m = n + size(B,2); A =
   [A,B];
4 for i=1:n-1, pivot = A(i,i);
5   for k=i+1:n, fac = A(k,i)/pivot;
6     A(k,i+1:m) = A(k,i+1:m) -
       fac*A(i,i+1:m);
7   end
8 end
9 A(n,n+1:m) = A(n,n+1:m) /A(n,n);
10 for i=n-1:-1:1
11   for l=i+1:n
12     A(i,n+1:m) = A(i,n+1:m) -
       A(l,n+1:m)*A(i,l);
13   end
14   A(i,n+1:m) = A(i,n+1:m)/A(i,i);
15 end
16 X = A(:,n+1:m);

```

Next two remarks: For understanding or analyzing special variants of Gaussian elimination, it is useful to be aware of

- the effects of elimination steps on the level of *matrix blocks*, cf. Rem. 1.2.10,
- and of the *recursive nature* of Gaussian elimination.

*Remark 2.1.10* (Gaussian elimination via rank-1 modifications).

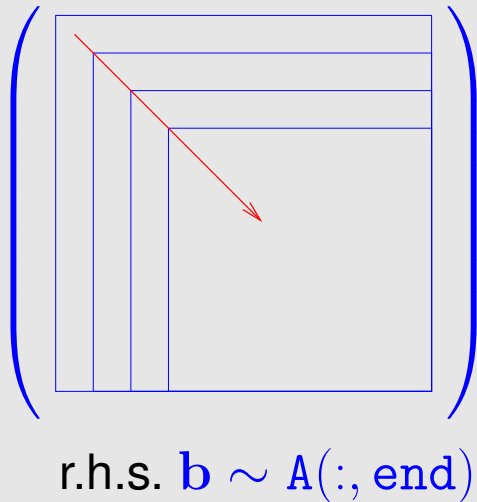
Block perspective (first step of Gaussian elimination with pivot  $\alpha \neq 0$ ), cf. (2.1.2):

$$\mathbf{A} := \left( \begin{array}{c|c} \alpha & \mathbf{c}^\top \\ \hline \mathbf{d} & \mathbf{C} \end{array} \right) \rightarrow \mathbf{A}' := \left( \begin{array}{c|c} \alpha & \mathbf{c}^\top \\ \hline \mathbf{0} & \mathbf{C}' := \mathbf{C} - \frac{\mathbf{d}\mathbf{c}^\top}{\alpha} \end{array} \right). \quad (2.1.11)$$

rank-1 modification of  $\mathbf{C}$

Adding a tensor product of two vectors to a matrix is called a **rank-1 modification** of that matrix.

(2.1.11) suggests a *recursive* variant of Gaussian elimination:



```

1 function A = blockgs(A)
2 %in-situ recursive Gaussian elimination, no pivoting
3 %right hand side in rightmost column of A: A(:,end)
4 n=size(A,1);
5 if (n>1)
6   C=blockgs(A(2:end,2:end)-A(2:end,1) ...
7     *A(1,2:end)/A(1,1));
8   A=[A(1,:);zeros(n-1,1),C];
9 end

```

In this code the Gaussian elimination is carried out **in situ**: the matrix  $\mathbf{A}$  is replaced with the transformed matrices during elimination. If the matrix is not needed later this offers maximum efficiency.  $\triangle$

**Remark 2.1.12** (Block Gaussian elimination).

Recall “principle” from Ex. ??: deal with block matrices (“matrices of matrices”) like regular matrices (except for commutativity of multiplication!).

Given: regular matrix  $\mathbf{A} \in \mathbb{K}^{n,n}$  with sub-matrices  $\mathbf{A}_{11} := (\mathbf{A})_{1:k,1:k}$ ,  $\mathbf{A}_{22} = (\mathbf{A})_{k+1:n,k+1:n}$ ,  
 $\mathbf{A}_{12} = (\mathbf{A})_{1:k,k+1:n}$ ,  $\mathbf{A}_{21} := (\mathbf{A})_{k+1:n,1:k}$ ,  $k < n$ ,  
 right hand side vector  $\mathbf{b} \in \mathbb{K}^n$ ,  $\mathbf{b}_1 = (\mathbf{b})_{1:k}$ ,  $\mathbf{b}_2 = (\mathbf{b})_{k+1:n}$

$$\begin{aligned} \left( \begin{array}{cc|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{b}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{b}_2 \end{array} \right) & \xrightarrow{\textcircled{1}} \left( \begin{array}{cc|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{b}_1 \\ 0 & \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} & \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1 \end{array} \right) \\ & \xrightarrow{\textcircled{2}} \left( \begin{array}{cc|c} \mathbf{I} & 0 & \mathbf{A}_{11}^{-1}(\mathbf{b}_1 - \mathbf{A}_{12}\mathbf{S}^{-1}\mathbf{b}_S) \\ 0 & \mathbf{I} & \mathbf{S}^{-1}\mathbf{b}_S \end{array} \right), \end{aligned}$$

where  $\mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$  (Schur complement, see Rem. 2.2.16),  $\mathbf{b}_S := \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1$

$\textcircled{1}$ : elimination step,  $\textcircled{2}$ : backsubstitution step

*Assumption:* (sub-)matrices regular, if required.



## 2.2 LU-Decomposition/LU-Factorization

A **matrix factorization** (*ger.* Matrixzerlegung) expresses a general matrix  $A$  as product of two *special* (factor) matrices. Requirements for these special matrices define the matrix factorization.

Mathematical issue: existence & uniqueness

Numerical issue: algorithm for computing factor matrices

### Matrix factorizations

- ☞ often capture the essence of algorithms in compact form (here: Gaussian elimination),
- ☞ are important building blocks for complex algorithms,
- ☞ are key theoretical tools for algorithm analysis.



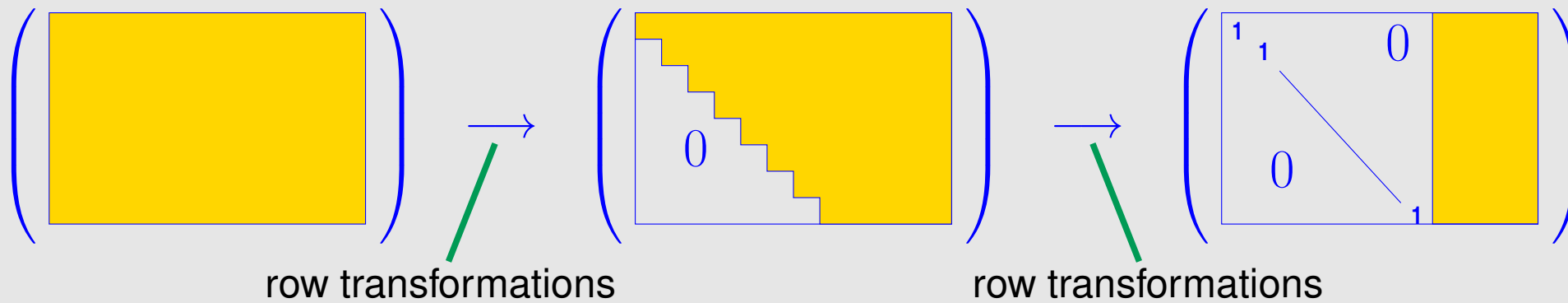
In this section: forward elimination step of Gaussian elimination will be related to a special matrix factorization, the so-called LU-factorization.

This should be well known from the introductory linear algebra course. In case you need to refresh your knowledge, please consult one of the following:

- textbook by Nipp & Stoffer [48, Sect. 2.4],
- book by M. Hanke-Bourgeois [35, II.4],
- linear algebra lecture notes by M. Gutknecht [27, Sect. 3.1],
- textbook by Quarteroni et al. [51, Sect.3.3.1],
- Sect. 3.5 of the book by Dahmen & Reusken.

See also (2.2.1) below.

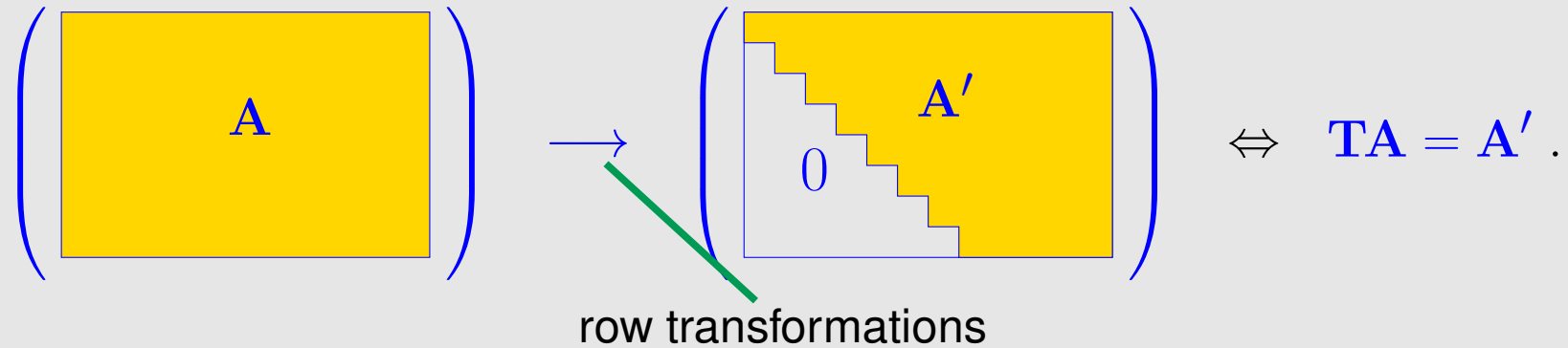
The gist of Gaussian elimination:



Here: **row transformation** = adding a multiple of a matrix row to another row, or multiplying a row with a non-zero scalar (number)  
(more special than row transformations discussed in Ex. 1.2.8)

Note: these row transformations preserve regularity of a matrix (why ?)  
▷ suitable for transforming linear systems of equations  
(solution will not be affected)

Ex. 1.2.8: row transformations can be realized by multiplication *from left* with suitable transformation matrices. When multiplying these transformation matrices we can emulate the effect to successive row transformations through left multiplication with a matrix **T**:



Now we want to determine the  $T$  for the forward elimination step of Gaussian elimination.

*Example 2.2.1* (Gaussian elimination and LU-factorization).  $\rightarrow$  [48, Sect. 2.4], [35, II.4], [27, Sect. 3.1]

LSE from Ex. 2.1.1: consider (forward) Gaussian elimination:

$$\begin{pmatrix} 1 & 1 & 0 \\ 2 & 1 & -1 \\ 3 & -1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ -3 \end{pmatrix} \iff \begin{cases} x_1 + x_2 & = & 4 \\ 2x_1 + x_2 - x_3 & = & 1 \\ 3x_1 - x_2 - x_3 & = & -3 \end{cases}$$

$$\begin{pmatrix} 1 & & \\ & 1 & \\ & & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 2 & 1 & -1 \\ 3 & -1 & -1 \end{pmatrix} \begin{pmatrix} 4 \\ 1 \\ -3 \end{pmatrix} \blacktriangleright \begin{pmatrix} 1 & & \\ 2 & 1 & \\ & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & -1 \\ 3 & -1 & -1 \end{pmatrix} \begin{pmatrix} 4 \\ -7 \\ -3 \end{pmatrix} \blacktriangleright$$

$$\begin{pmatrix} 1 & & & \\ 2 & 1 & & \\ 3 & 0 & 1 & \end{pmatrix} \begin{pmatrix} \mathbf{1} & 1 & 0 \\ 0 & -1 & -1 \\ 0 & -4 & -1 \end{pmatrix} \begin{pmatrix} 4 \\ -7 \\ -15 \end{pmatrix} \rightarrow \underbrace{\begin{pmatrix} 1 & & & \\ 2 & 1 & & \\ 3 & 4 & 1 & \end{pmatrix}}_{=L} \underbrace{\begin{pmatrix} 1 & 1 & 0 \\ 0 & \mathbf{-1} & -1 \\ 0 & 0 & 3 \end{pmatrix}}_{=U} \begin{pmatrix} 4 \\ -7 \\ 13 \end{pmatrix}$$

 = pivot row, pivot element **bold**, negative multipliers red



Details: link between Gaussian elimination and **matrix factorization** → Ex. 2.2.1  
(row transformation = multiplication with elimination matrix)

$$a_1 \neq 0 \rightarrow \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ -\frac{a_2}{a_1} & 1 & & & 0 \\ -\frac{a_3}{a_1} & & \dots & & \\ \vdots & & & \dots & \\ -\frac{a_n}{a_1} & 0 & & & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad (2.2.2)$$

►  $n - 1$  steps of Gaussian elimination: ► matrix factorization ( $\rightarrow$  Ex. 2.1.1)  
(non-zero pivot elements assumed)

$A = L_1 \cdots L_{n-1} U$  with elimination matrices  $L_i, i = 1, \dots, n - 1$ ,  
upper triangular matrix  $U \in \mathbb{R}^{n,n}$ .

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ l_2 & 1 & & & 0 \\ l_3 & & \ddots & & \\ \vdots & & & \ddots & \\ l_n & 0 & & & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & & & 0 \\ 0 & h_3 & 1 & & \\ \vdots & \vdots & & \ddots & \\ 0 & h_n & 0 & & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ l_2 & 1 & & & 0 \\ l_3 & h_3 & 1 & & \\ \vdots & \vdots & & \ddots & \\ l_n & h_n & 0 & & 1 \end{pmatrix}$$

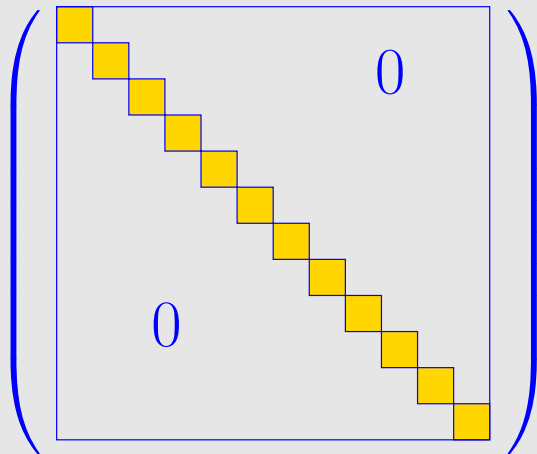
►  $L_1 \cdots L_{n-1}$  are **normalized lower triangular matrices**  
(entries = multipliers  $-\frac{a_{ik}}{a_{kk}}$  from (2.1.2)  $\rightarrow$  Ex. 2.1.1)

**Definition 2.2.3** (Types of matrices).

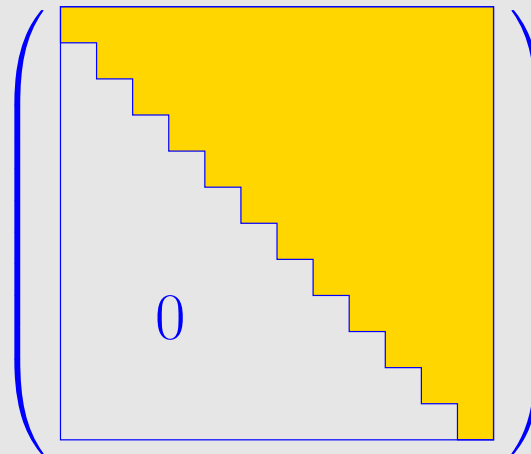
A matrix  $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m,n}$  is

- *diagonal matrix*, if  $a_{ij} = 0$  for  $i \neq j$ ,
- *upper triangular matrix* if  $a_{ij} = 0$  for  $i > j$ ,
- *lower triangular matrix* if  $a_{ij} = 0$  for  $i < j$ .

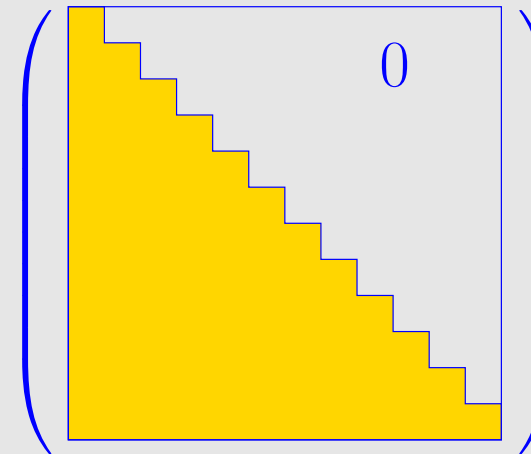
A triangular matrix is *normalized*, if  $a_{ii} = 1, i = 1, \dots, \min\{m, n\}$ .



diagonal matrix



upper triangular



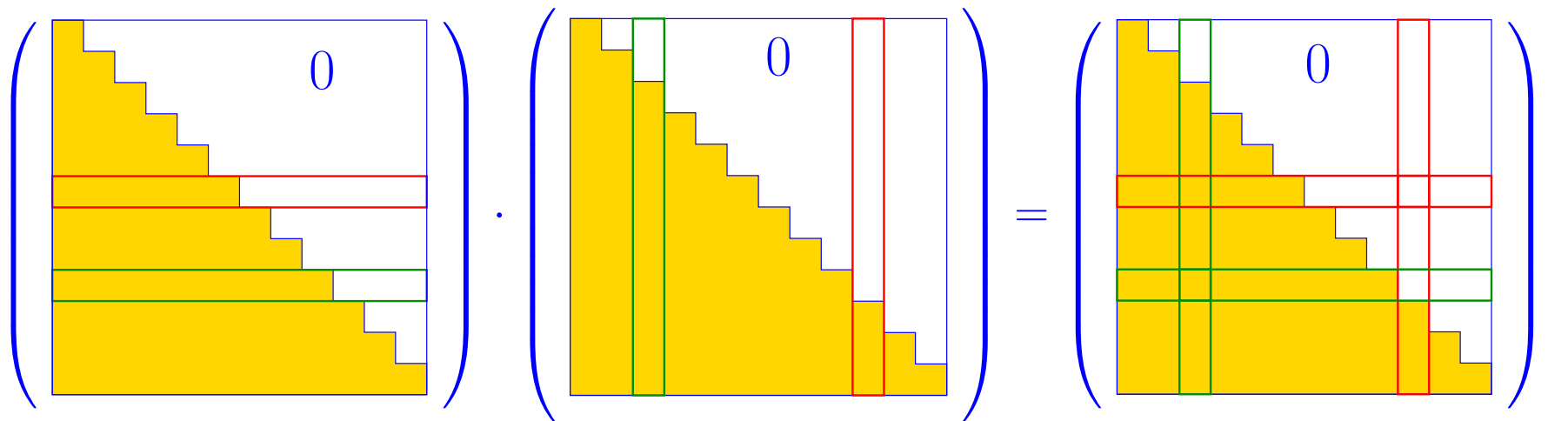
lower triangular

**Lemma 2.2.4** (Group of regular diagonal/triangular matrices).

$$\mathbf{A}, \mathbf{B} \begin{cases} \text{diagonal} \\ \text{upper triangular} \\ \text{lower triangular} \end{cases} \Rightarrow \mathbf{AB} \text{ and } \mathbf{A}^{-1} \begin{cases} \text{diagonal} \\ \text{upper triangular} \\ \text{lower triangular} \end{cases} .$$

(assumes that  $\mathbf{A}$  is regular)

“Proof by visualization” → Rem. 1.2.4



The (forward) Gaussian elimination (without pivoting), for  $\mathbf{Ax} = \mathbf{b}$ ,  $\mathbf{A} \in \mathbb{R}^{n,n}$ , if possible, is algebraically equivalent to an **LU-factorization**/LU-decomposition  $\mathbf{A} = \mathbf{LU}$  of  $\mathbf{A}$  into a normalized lower triangular matrix  $\mathbf{L}$  and an upper triangular matrix  $\mathbf{U}$ , [13, Thm. 3.2.1], [48, Thm. 2.10], [27, Sect. 3.1].

Algebraically equivalent  $\hat{=}$  when carrying out the forward elimination in situ as in Code 2.1.6 and storing the multipliers in a lower triangular matrix as in Ex. 2.2.1, then the latter will contain the **L**-factor and the original matrix will be replaced with the **U**-factor.

**Lemma 2.2.5** (Existence of  $LU$ -decomposition).

The  $LU$ -decomposition of  $\mathbf{A} \in \mathbb{K}^{n,n}$  exists, if all submatrices  $(\mathbf{A})_{1:k,1:k}$ ,  $1 \leq k \leq n$ , are regular.

*Proof.* by block matrix perspective ( $\rightarrow$  Rem. 1.2.10) and induction w.r.t.  $n$ :

$n = 1$ : assertion trivial

$n-1 \rightarrow n$ : Induction hypothesis ensures existence of normalized lower triangular matrix  $\tilde{\mathbf{L}}$  and regular upper triangular matrix  $\tilde{\mathbf{U}}$  such that  $\tilde{\mathbf{A}} = \tilde{\mathbf{L}}\tilde{\mathbf{U}}$ , where  $\tilde{\mathbf{A}}$  is the upper left  $(n-1) \times (n-1)$  block of  $\mathbf{A}$ :

$$\left( \begin{array}{c|c} \tilde{\mathbf{A}} & \mathbf{b} \\ \mathbf{a}^\top & \alpha \end{array} \right) = \left( \begin{array}{c|c} \tilde{\mathbf{L}} & 0 \\ \mathbf{x}^\top & 1 \end{array} \right) \left( \begin{array}{c|c} \tilde{\mathbf{U}} & \mathbf{y} \\ 0 & \xi \end{array} \right) =: \mathbf{LU}.$$

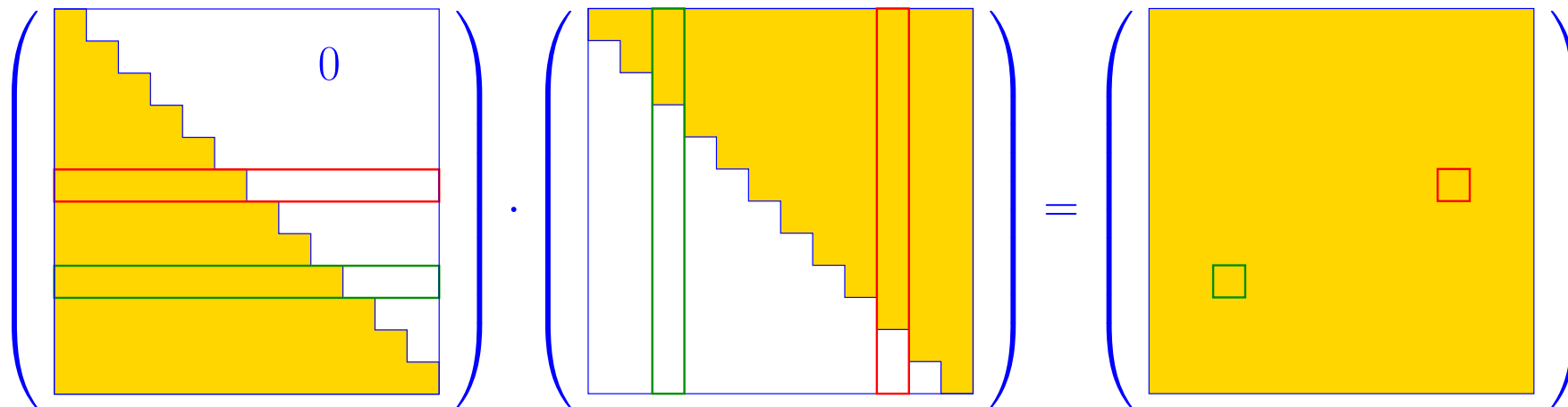


Then solve

- ①  $\tilde{\mathbf{L}}\mathbf{y} = \mathbf{b}$  → provides  $\mathbf{y} \in \mathbb{K}^n$ ,
- ②  $\mathbf{x}^\top \tilde{\mathbf{U}} = \mathbf{a}^\top$  → provides  $\mathbf{x} \in \mathbb{K}^n$ ,
- ③  $\mathbf{x}^\top \mathbf{y} + \xi = \alpha$  → provides  $\xi \in \mathbb{K}$ .

Regularity of  $\mathbf{A}$  involves  $\xi \neq 0$  (why?) so that  $\mathbf{U}$  will be regular, too.

A direct way to  $LU$ -decomposition [27, Sect. 3.1], [51, Sect. 3.3.3]:



$$\mathbf{LU} = \mathbf{A} \Rightarrow a_{ik} = \sum_{j=1}^{\min\{i,k\}} l_{ij}u_{jk} = \begin{cases} \sum_{j=1}^{i-1} l_{ij}u_{jk} + 1 \cdot u_{ik} & , \text{ if } i \leq k , \\ \sum_{j=1}^{k-1} l_{ij}u_{jk} + l_{ik}u_{kk} & , \text{ if } i > k . \end{cases} \quad (2.2.6)$$

- • row by row computation of **U**
- column by column computation of **L**

Entries of **A** can be replaced with those of **L**, **U** !  
(so-called **in situ**/in place computation)

(Crout's algorithm, [27, Alg. 3.1])

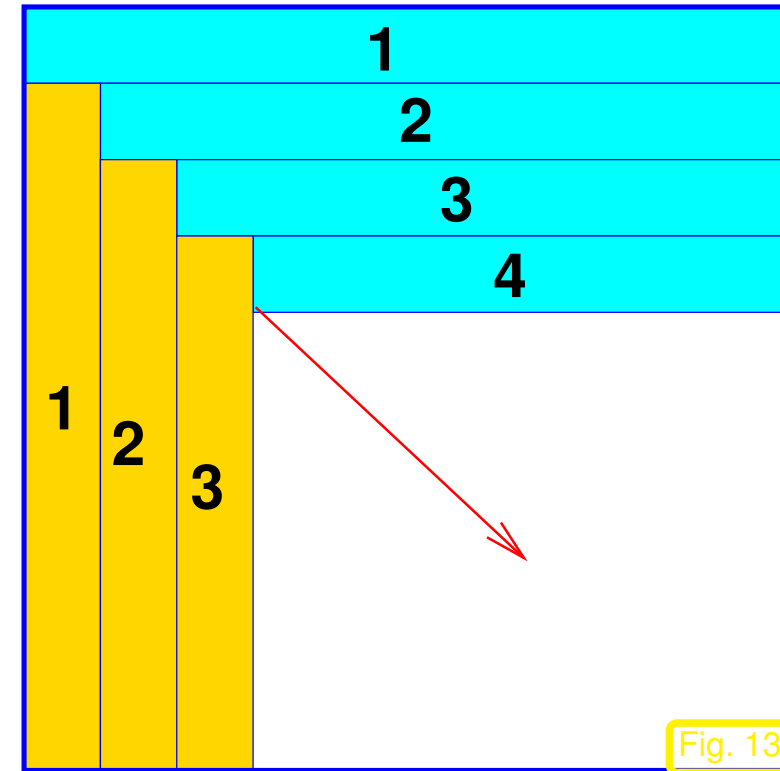
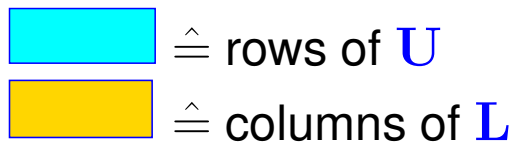


Fig. 13

LU-factorization = "inversion" of matrix multiplication:

## Code 2.2.7: LU-factorization

```

1 function [L,U] = lufak(A)
2 % Algorithm of Crout:
3 % LU-factorization of  $A \in \mathbb{K}^{n,n}$ 
4 n = size(A,1); if (size(A,2)
   ~ = n), error('n ~= m'); end
5 L = eye(n); U = zeros(n,n);
6 for k=1:n
7     % Compute row of U
8     for j=k:n
9         U(k,j) = A(k,j) -
           L(k,1:k-1)*U(1:k-1,j);
10    end
11    % Compute column of L
12    for i=k+1:n
13        L(i,k) = (A(i,k) -
                  L(i,1:k-1)*U(1:k-1,k))
                  /U(k,k);
14    end
15 end

```

Code 2.2.8: matrix multiplication  $L \cdot U$ 

```

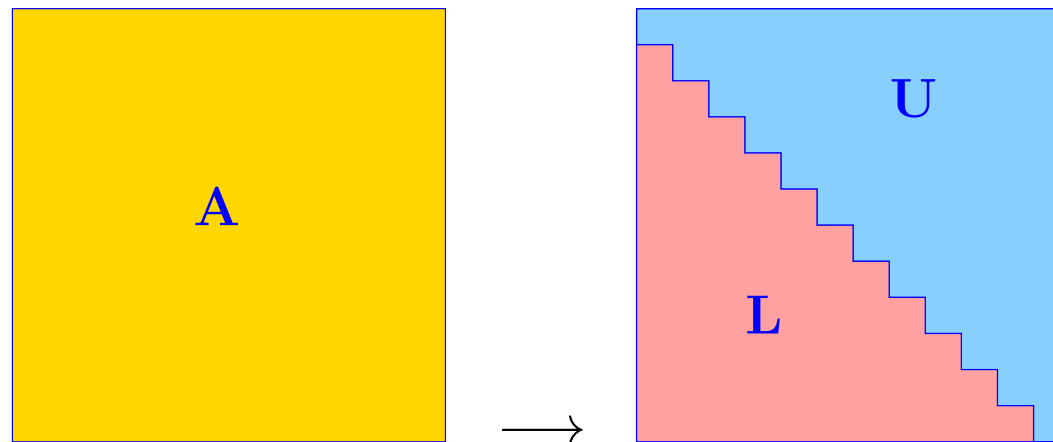
1 function A = lumult(L,U)
2 % Multiplication of normalized
3 lower/upper triangular matrices
4 n = size(L,1); A = zeros(n,n);
5 if ((size(L,2) ~= n) ||
   (size(U,1) ~= n) ||
   (size(U,2) ~= n))
6     error('size mismatch');
7 end
8 for k=n:-1:1
9     for j=k:n
10        A(k,j) = U(k,j) +
               L(k,1:k-1)*U(1:k-1,j);
11    end
12    for i=k+1:n
13        A(i,k) =
               L(i,1:k-1)*U(1:k-1,k)
               + L(i,k)*U(k,k);
14    end
15 end

```

gular and normalized lower triangular matrix ( $\rightarrow$  Code 2.2.7) yields the algorithm for LU-factorization ( $\rightarrow$  Code 2.2.6).

$$\text{asymptotic complexity of LU-factorization of } \mathbf{A} \in \mathbb{R}^{n,n} = \frac{1}{3}n^3 + O(n^2) = O(n^3) \quad (2.2.9)$$

*Remark 2.2.10* (In-situ LU-decomposition).



Replace entries of  $\mathbf{A}$  with entries of  $\mathbf{L}$  (strict lower triangle) and  $\mathbf{U}$  (upper triangle).

*Remark 2.2.11* (Recursive LU-factorization).

Recall: recursive view of Gaussian elimination → Rem. 2.1.10

In light of the close relationship between Gaussian elimination and LU-factorization there will also be a recursive version of LU-factorization.

Recursive **in situ** (in place) LU-decomposition  
of  $A \in \mathbb{R}^{n,n}$  (without pivoting):

**L, U** stored in place of **A**:

```

1 function [L,U] =
    lurecdriver(A)
2 A = lurec(A);
3 % post-processing: extract L and
  U
4 U = triu(A);
5 L = tril(A,-1) +
    eye(size(A));

```

```

1 function A = lurec(A)
2 % insitu recursive LU-factorization
3 if (size(A,1)>1)
4   fac = A(2:end,1)/A(1,1);
5   C = lurec(A(2:end,2:end)...
6     -fac*A(1,2:end));
7   A=[A(1,:);fac,C];
8 end

```

Refer to (2.1.11) to understand `lurec`: the rank-1 modification of the lower  $(n-1) \times (n-1)$ -block

of the matrix is done in lines 5-6 of the code.

---

Solving a linear system of equations by LU-factorization:

*Algorithm 2.2.12* (Using LU-factorization to solve a linear system of equations).

- $\mathbf{Ax} = \mathbf{b}$  :
- ① *LU*-decomposition  $\mathbf{A} = \mathbf{LU}$ , #elementary operations  $\frac{1}{3}n(n-1)(n+1)$
  - ② **forward substitution**, solve  $\mathbf{Lz} = \mathbf{b}$ , #elementary operations  $\frac{1}{2}n(n-1)$
  - ③ **backward substitution**, solve  $\mathbf{Ux} = \mathbf{z}$ , #elementary operations  $\frac{1}{2}n(n+1)$

► asymptotic complexity: (in leading order) the same as for Gaussian elimination

Gauss elimination and LU-factorization are equivalent and only differ in the ordering of the steps.

Then, why is it important to know about LU-factorization?

Because in the case of LU-factorization the expensive forward elimination and the less expensive (forward/backward) substitutions are separated, which sometimes can be exploited to reduce computational cost, as highlighted in the next remark.

*Remark 2.2.13* (Many sequential solutions of LSE).

Given: regular matrix  $\mathbf{A} \in \mathbb{K}^{n,n}$ ,  $n \in \mathbb{N}$ , and  $N \in \mathbb{N}$ , both  $n, N$  large

**foolish !**

```

1 % Setting:  $N \gg 1$ , large matrix  $\mathbf{A}$ 
2 for j=1:N
3     x = A\b;
4     b = some_function(x);
5 end

```

computational effort  $O(Nn^3)$

**smart !**

```

1 % Setting:  $N \gg 1$ , large matrix  $\mathbf{A}$ 
2 [L,U] = lu(A);
3 for j=1:N
4     x = U\ (L\b);
5     b = some_function(x);
6 end

```

computational effort  $O(n^3 + Nn^2)$

Efficient implementation requires *one* LU-decomposition of  $\mathbf{A}$  (cost  $O(n^3)$ ) +  $N$  forward substitutions +  $N$  backward substitutions (cost  $Nn^2$ )

Remark 2.2.14 (“Partial  $LU$ -decompositions” of principal minors).

Principal minor  $\hat{=}$  left upper block of a matrix

The following “visual rule” help identify the structure of the  $LU$ -factors of a matrix.

$$\begin{pmatrix} \square & \square \\ \square & \square \end{pmatrix} = \begin{pmatrix} \color{cyan}\triangle & \square \\ \square & 0 \end{pmatrix} \begin{pmatrix} \square & \color{cyan}\triangle \\ 0 & \square \end{pmatrix} \tag{2.2.15}$$

The left-upper blocks of both  $\mathbf{L}$  and  $\mathbf{U}$  in the  $LU$ -factorization of  $\mathbf{A}$  depend only on the corresponding left-upper block of  $\mathbf{A}$ !



Remark 2.2.16 (Block LU-factorization).

In the spirit of Rem. 1.2.10: block perspective of LU-factorization.

Natural in light of the close connection between matrix multiplication and matrix factorization, cf. Codes 2.2.7, 2.2.7:

Block matrix multiplication (1.2.11)  $\cong$  block  $LU$ -decomposition:

With  $\mathbf{A}_{11} \in \mathbb{K}^{n,n}$  regular,  $\mathbf{A}_{12} \in \mathbb{K}^{n,m}$ ,  $\mathbf{A}_{21} \in \mathbb{K}^{m,n}$ ,  $\mathbf{A}_{22} \in \mathbb{K}^{m,m}$ :

$$\underbrace{\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix}}_{\text{block LU-factorization}} = \begin{pmatrix} \mathbf{I} & 0 \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{S} \end{pmatrix}, \quad \begin{array}{l} \text{Schur complement} \\ \mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \end{array} \quad (2.2.17)$$

→ block Gaussian elimination, see Rem. 2.1.12.

△

# 2.3 Pivoting

Known from linear algebra [48, Sect. 1.1]:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

breakdown of Gaussian elimination  
pivot element = 0

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_2 \\ b_1 \end{pmatrix}$$

Gaussian elimination feasible

Idea (in linear algebra): Avoid zero pivot elements by **swapping rows**

*Example 2.3.1* (Pivoting and numerical stability). → [13, Bsp. 3.2.3]

```
1 % Example: numerical instability without  
  pivoting  
2 A = [5.0E-17 , 1; 1 , 1];  
3 b = [1;2];  
4 x1 = A\b,  
5 x2 =gausselim(A,b), % see Code 2.1.8  
6 [L,U] = lufak(A); % see Code 2.2.6  
7 z = L\b; x3 = U\z,
```

Ouput of MATLAB run:

```
x1 = 1  
    1  
x2 = 0  
    1  
x3 = 0  
    1
```

$$\mathbf{A} = \begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \Rightarrow \mathbf{x} = \begin{pmatrix} \frac{1}{1-\epsilon} \\ \frac{1-2\epsilon}{1-\epsilon} \end{pmatrix} \approx \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{for } |\epsilon| \ll 1.$$

What is wrong with MATLAB? Needed: insight into **roundoff errors** → Sect. 2.4

Armed with knowledge about the behavior of machine numbers and roundoff errors we can now understand what is going on in Ex. 2.3.1

Straightforward LU-factorization: if  $\epsilon \leq \frac{1}{2}\text{eps}$ ,  $\text{eps} \hat{=} \text{machine precision}$ ,

$$\blacktriangleright \quad \mathbf{L} = \begin{pmatrix} 1 & 0 \\ \epsilon^{-1} & 1 \end{pmatrix}, \quad \mathbf{U} = \begin{pmatrix} \epsilon & 1 \\ 0 & 1 - \epsilon^{-1} \end{pmatrix} \stackrel{(*)}{=} \tilde{\mathbf{U}} := \begin{pmatrix} \epsilon & 1 \\ 0 & -\epsilon^{-1} \end{pmatrix} \quad \text{in } \mathbb{M}! \quad (2.3.2)$$

(\*): because  $1 + 2/\text{eps} = 2/\text{eps}$ , see Rem. 2.4.13.

$$\blacktriangleright \quad \text{Solution of } \mathbf{L}\tilde{\mathbf{U}}\mathbf{x} = \mathbf{b}: \quad \mathbf{x} = \begin{pmatrix} 2\epsilon \\ 1 - 2\epsilon \end{pmatrix} \quad (\text{meaningless result !})$$

LU-factorization after swapping rows:

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix} \Rightarrow \mathbf{L} = \begin{pmatrix} 1 & 0 \\ \epsilon & 1 \end{pmatrix}, \quad \mathbf{U} = \begin{pmatrix} 1 & 1 \\ 0 & 1 - \epsilon \end{pmatrix} = \tilde{\mathbf{U}} := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{in } \mathbb{M}. \quad (2.3.3)$$

► Solution of  $\mathbf{L}\tilde{\mathbf{U}}\mathbf{x} = \mathbf{b}$ :  $\mathbf{x} = \begin{pmatrix} 1 + 2\epsilon \\ 1 - 2\epsilon \end{pmatrix}$  (sufficiently accurate result !)

no row swapping,  $\rightarrow$  (2.3.2):  $\mathbf{L}\tilde{\mathbf{U}} = \mathbf{A} + \mathbf{E}$  with  $\mathbf{E} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$  ► unstable !

after row swapping,  $\rightarrow$  (2.3.3):  $\mathbf{L}\tilde{\mathbf{U}} = \tilde{\mathbf{A}} + \mathbf{E}$  with  $\mathbf{E} = \begin{pmatrix} 0 & 0 \\ 0 & \epsilon \end{pmatrix}$  ► stable !

Introduction to the notion of **stability**  $\rightarrow$  Sect. 2.5.2, Def. 2.5.11, see also [13, Sect. 2.3].



Suitable pivoting essential for controlling impact of roundoff errors  
on Gaussian elimination ( $\rightarrow$  Sect. 2.5.2, [48, Sect. 2.5])

*Example 2.3.4* (Gaussian elimination with pivoting for  $3 \times 3$ -matrix).

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 2 \\ 2 & -3 & 2 \\ 1 & 24 & 0 \end{pmatrix} \xrightarrow{\textcircled{1}} \begin{pmatrix} 2 & -3 & 2 \\ 1 & 2 & 2 \\ 1 & 24 & 0 \end{pmatrix} \xrightarrow{\textcircled{2}} \begin{pmatrix} 2 & -3 & 2 \\ 0 & 3.5 & 1 \\ 0 & 25.5 & -1 \end{pmatrix} \xrightarrow{\textcircled{3}} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 25.5 & -1 \\ 0 & 3.5 & 1 \end{pmatrix} \xrightarrow{\textcircled{4}} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 25.5 & -1 \\ 0 & 0 & 1.373 \end{pmatrix}$$

①: swap rows 1 & 2.

②: elimination with top row as pivot row

③: swap rows 2 & 3

④: elimination with 2nd row as pivot row



*Algorithm 2.3.5* (Gaussian elimination with partial pivoting).

Code 2.3.6: Gaussian elimination with pivoting: extension of Code 2.1.6

```

1 function x = gepiv(A,b)
2 % Solving an LSE  $\mathbf{Ax}=\mathbf{b}$  by Gaussian elimination with partial pivoting
3 n = size(A,1); A = [A,b]; %
4 % Forward elimination by rank-1 modification, see Rem. 2.1.10
5 for k=1:n-1
6     [p, j] = max(abs(A(k:n,k)) ./ max(abs(A(k:n,k:n))')) %
7     if (p < eps*norm(A(k:n,k:n),1)), %
8         disp('A nearly singular'); end
9     A([k, j+k-1], k:n+1) = A([j+k-1, k], k:n+1); %
10    A(k+1:n, k+1:n+1) =
        A(k+1:n, k+1:n+1) - (A(k+1:n, k)*A(k, k+1:n+1))/A(k, k); %

```

```

11 end
12 % Back substitution (same as in Code 2.1.6)
13 A(n,n+1) = A(n,n+1) /A(n,n);
14 for i=n-1:-1:1
15     A(i,n+1) = (A(i,n+1) - A(i,i+1:n)*A(i+1:n,n+1)) /A(i,i);
16 end
17 x = A(:,n+1); %

```

choice of pivot row index  $j$  (Line 6 of code): **relatively largest** pivot [48, Sect. 2.5],

$$j \in \{k, \dots, n\} \text{ such that } \frac{|a_{ji}|}{\max\{|a_{jl}|, l = k, \dots, n\}} \rightarrow \max \quad (2.3.7)$$

for  $k = j, k \in \{i, \dots, n\}$ : **partial pivoting**

Explanations to Code 2.3.5:

Line 3: Augment matrix  $\mathbf{A}$  by right hand side vector  $\mathbf{b}$ , see comments on Code 2.1.6 for explanations.

Line 6: Select index  $j$  for pivot row according to the recipe of **partial pivoting**, see (2.3.7).

Note: Inefficient implementation above (too many comparisons)! Try to do better!

Line 7: If the pivot element is still very small relative to the norm of the matrix, then we have encountered an entire column that is close to zero. Gaussian elimination may not be possible in a stable fashion for this matrix; warn user and terminate.

Line 9: A way to swap rows of a matrix in MATLAB.

Line 10: Forward elimination by means of rank-1-update, see (2.1.11).

Line 17: As in Code 2.1.6: after back substitution last column of augmented matrix supplies solution  $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ .

## Algorithm 2.3.8.

Recall: close relationship between Gaussian elimination and LU-factorization

- LU-factorization with pivoting? Of course, just by rearranging the operations of Gaussian forward elimination with pivoting.

MATLAB-code for *recursive* in place LU-factorization of  $A \in \mathbb{R}^{n,n}$  with **partial pivoting** (*ger.:* Spaltenpivotsuche):

Code 2.3.9: recursive LU-factorization with partial pivoting

```

1 function A = gsrecpiv(A)
2 n = size (A,1);
3 if (n > 1)
4     [p, j] = max (abs (A(:,1)) ./ max (abs (A)')')'; %
5     if (p < eps*norm (A(:,1:n),1)), disp ('A nearly singular'); end %
6     A([1, j], :) = A([j, 1], :); %
7     fac = A(2:end,1)/A(1,1); %
8     C = gsrecpiv(A(2:end,2:end) - fac*A(1,2:end)); %
9     A = [A(1, :) ; -fac, C ]; %
10 end

```



## Explanations to Code 2.3.8:

- Line 4: Find the *relatively largest* pivot element  $p$  and the index  $j$  of the corresponding row of the matrix, see (2.3.7)
- Line 5: If the pivot element is still very small relative to the norm of the matrix, then we have encountered an entire column that is close to zero. The matrix is (close to) singular and LU-factorization does not exist.
- Line 6: Swap the first and the  $j$ -th row of the matrix.
- Line 7: Initialize the vector of multiplier.
- Line 8: Call the routine for the upper right  $(n - 1) \times (n - 1)$ -block of the matrix after subtracting suitable multiples of the first row from the other rows, *cf.* Rem. 2.1.10 and Rem. 2.2.11.
- Line 9: Reassemble the parts of the LU-factors. The vector of multipliers yields a column of  $\mathbf{L}$ , see Ex. 2.2.1.

**Algorithm 2.3.10.**

C++-code (non-recursive implementation, of course) for in-situ LU-factorization ( $\rightarrow$  Sect. 2.2) of  $A \in \mathbb{R}^{n,n}$  with **partial pivoting**  $\blacktriangleright$

Row permutations recorded in vector  $p$  !

Usual choice of pivot,  
pivot row index  $j$  according to (2.3.7)

Note: row swapping  $\leftrightarrow$  pointer swapping

```
template<class Matrix>
void LU(Matrix &A, std::vector<int> &p) {
    int n = A.dim();
    for(int i=1; i<=n; i++) p[i] = i;
    for(int i=1; i<n; i++) {
        Choose index  $j \in \{i, \dots, n\}$  of pivot row
        std::swap(p[i], p[j]);
        for(int k=i+1; k<=n; k++) {
            A(p[k], i) /= A(p[i], i);
            for(int l=i+1; l<=n; l++) {
                A(p[k], l) -= A(p[k], i) * A(p[i], l);
            }
        }
    }
}
```

**Example 2.3.11** (Rationale for partial pivoting policy (2.3.7)).  $\rightarrow$  [48, Page 47]

Why *relatively* largest pivot element in (2.3.7)? **scaling invariance** desirable

Scale linear system of equations from Ex. 2.3.1:

$$\begin{pmatrix} 2/\epsilon & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 & 2/\epsilon \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2/\epsilon \\ 1 \end{pmatrix}$$

No row swapping, if absolutely largest pivot element is used:

$$\begin{pmatrix} 2 & 2/\epsilon \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 2/\epsilon \\ 0 & 1 - 2/\epsilon \end{pmatrix} \doteq \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 2/\epsilon \\ 0 & -2/\epsilon \end{pmatrix} \quad \text{in } \mathbb{M}.$$

```

1 % Example: importance of scale-invariant
  pivoting
2 epsilon = 5.0E-17;
3 A = [epsilon , 1; 1 , 1]; b = [1;2];
4 D = [1/epsilon, 0; 0 ,1];
5 A = D*A; b = D*b;
6 x1 = A\b, % MATLAB internal Gaussian
  elimination
7 x2 =gausselim(A,b), % see Code 2.1.8
8 [L,U] = lufak(A); % see Code 2.2.6
9 z = L\b; x3 = U\z,
```

Ouput of MATLAB run:

```

x1 = 1
      1
x2 = 0
      1
x3 = 0
      1
```



**Pivoting: Theoretical perspective**

**Definition 2.3.12** (Permutation matrix).

An  $n$ -*permutation*,  $n \in \mathbb{N}$ , is a bijective mapping  $\pi : \{1, \dots, n\} \mapsto \{1, \dots, n\}$ . The corresponding *permutation matrix*  $\mathbf{P}_\pi \in \mathbb{K}^{n,n}$  is defined by

$$(\mathbf{P}_\pi)_{ij} = \begin{cases} 1 & , \text{ if } j = \pi(i) , \\ 0 & \text{ else.} \end{cases}$$

permutation  $(1, 2, 3, 4) \mapsto (1, 3, 2, 4) \hat{=} \mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} .$

- Note:
- $\mathbf{P}^\top = \mathbf{P}^{-1}$  for any permutation matrix  $\mathbf{P}$  ( $\rightarrow$  permutation matrices orthogonal/unitary)
  - $\mathbf{P}_\pi \mathbf{A}$  effects  $\pi$ -permutation of rows of  $\mathbf{A} \in \mathbb{K}^{n,m}$
  - $\mathbf{A} \mathbf{P}_\pi$  effects  $\pi$ -permutation of columns of  $\mathbf{A} \in \mathbb{K}^{m,n}$

**Lemma 2.3.13** (Existence of LU-factorization with pivoting).  $\rightarrow$  [13, Thm. 3.25], [35, Thm. 4.4]

For any regular  $\mathbf{A} \in \mathbb{K}^{n,n}$  there is a permutation matrix ( $\rightarrow$  Def. 2.3.12)  $\mathbf{P} \in \mathbb{K}^{n,n}$ , a normalized lower triangular matrix  $\mathbf{L} \in \mathbb{K}^{n,n}$ , and a regular upper triangular matrix  $\mathbf{U} \in \mathbb{K}^{n,n}$  ( $\rightarrow$  Def. 2.2.3), such that  $\mathbf{PA} = \mathbf{LU}$ .

*Proof.* (by induction)

Every regular matrix  $\mathbf{A} \in \mathbb{K}^{n,n}$  admits a row permutation encoded by the permutation matrix  $\mathbf{P} \in \mathbb{K}^{n,n}$ , such that  $\mathbf{A}' := (\mathbf{A})_{1:n-1,1:n-1}$  is regular (why?).

By induction assumption there is a permutation matrix  $\mathbf{P}' \in \mathbb{K}^{n-1,n-1}$  such that  $\mathbf{P}'\mathbf{A}'$  possesses a LU-factorization  $\mathbf{A}' = \mathbf{L}'\mathbf{U}'$ . There are  $\mathbf{x}, \mathbf{y} \in \mathbb{K}^{n-1}$ ,  $\gamma \in \mathbb{K}$  such that

$$\begin{pmatrix} \mathbf{P}' & 0 \\ 0 & 1 \end{pmatrix} \mathbf{PA} = \begin{pmatrix} \mathbf{P}' & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A}' & \mathbf{x} \\ \mathbf{y}^\top & \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{L}'\mathbf{U}' & \mathbf{x} \\ \mathbf{y}^\top & \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{L}' & 0 \\ \mathbf{c}^\top & 1 \end{pmatrix} \begin{pmatrix} \mathbf{U} & \mathbf{d} \\ 0 & \alpha \end{pmatrix},$$

if we choose

$$\mathbf{d} = (\mathbf{L}')^{-1}\mathbf{x} \quad , \quad \mathbf{c} = (\mathbf{u}')^{-T}\mathbf{y} \quad , \quad \alpha = \gamma - \mathbf{c}^\top\mathbf{d} \quad ,$$

which is always possible. □

Example 2.3.14 (Ex. 2.3.4 cnt'd).

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 2 \\ 2 & -3 & 2 \\ 1 & 24 & 0 \end{pmatrix} \xrightarrow{\textcircled{1}} \begin{pmatrix} 2 & -3 & 2 \\ 1 & 2 & 2 \\ 1 & 24 & 0 \end{pmatrix} \xrightarrow{\textcircled{2}} \begin{pmatrix} 2 & -3 & 2 \\ 0 & 3.5 & 1 \\ 0 & 25.5 & -1 \end{pmatrix} \xrightarrow{\textcircled{3}} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 25.5 & -1 \\ 0 & 3.5 & 1 \end{pmatrix} \xrightarrow{\textcircled{4}} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 25.5 & -1 \\ 0 & 0 & 1.373 \end{pmatrix}$$

$$\mathbf{U} = \begin{pmatrix} 2 & -3 & 2 \\ 0 & 25.5 & -1 \\ 0 & 0 & 1.1373 \end{pmatrix}, \quad \mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0.1373 & 1 \end{pmatrix}, \quad \mathbf{P} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

Two permutations: in step  $\textcircled{1}$  swap rows #1 and #2, in step  $\textcircled{3}$  swap rows #2 and #3. Apply these swaps to the identity matrix and you will recover  $\mathbf{P}$ . See also [13, Ex. 3.30].



MATLAB function:  $[\mathbf{L}, \mathbf{U}, \mathbf{P}] = \text{lu}(\mathbf{A})$  ( $\mathbf{P}$  = permutation matrix)

Remark 2.3.15 (Row swapping commutes with forward elimination).

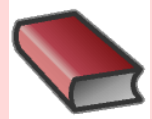
Any kind of pivoting only involves comparisons and row/column permutations, but no arithmetic operations on the matrix entries. This makes the following observation plausible:

The LU-factorization of  $\mathbf{A} \in \mathbb{K}^{n,n}$  with partial pivoting by Alg. 2.3.8 is *numerically equivalent* to the LU-factorization of  $\mathbf{PA}$  without pivoting ( $\rightarrow$  Code 2.2.6), when  $\mathbf{P}$  is a permutation matrix gathering the row swaps entailed by partial pivoting.

*numerically equivalent*  $\hat{=}$  same result when executed with the same machine arithmetic

► The above statement means that whenever we study the impact of roundoff errors on LU-factorization it is safe to consider only the basic version without pivoting, because we can always assume that row swaps have been conducted beforehand.





### Supplementary and further reading:

Concise textbook style presentations of machine arithmetic can be found in [13, Sect. 2.2] and [51, Sect. 2.5].

A very detailed exposition and in-depth discussion of the all the material in this section can be found in [36]:

- [36, Ch. 1]: excellent collection of examples concerning the impact of roundoff errors.
- [36, Ch. 2]: floating point arithmetic, see Def. 2.4.2 below and the remarks following it.

Computer = finite automaton



can handle only *finitely many* numbers, not  $\mathbb{R}$

machine numbers, set  $M$



Essential property:

$\mathbb{M}$  is a **discrete** subset of  $\mathbb{R}$

$\mathbb{M}$  not closed under elementary arithmetic operations  $+, -, \cdot, /$ .



**roundoff errors** (*ger.:* Rundungsfehler) are inevitable

The impact of roundoff means that mathematical identities may not carry over to the computational realm. Putting it bluntly,

Computers cannot compute “properly” !



**numerical computations**  $\neq$  **analysis**  
**linear algebra**

This introduces a *new and important* aspect in the study of numerical algorithms!

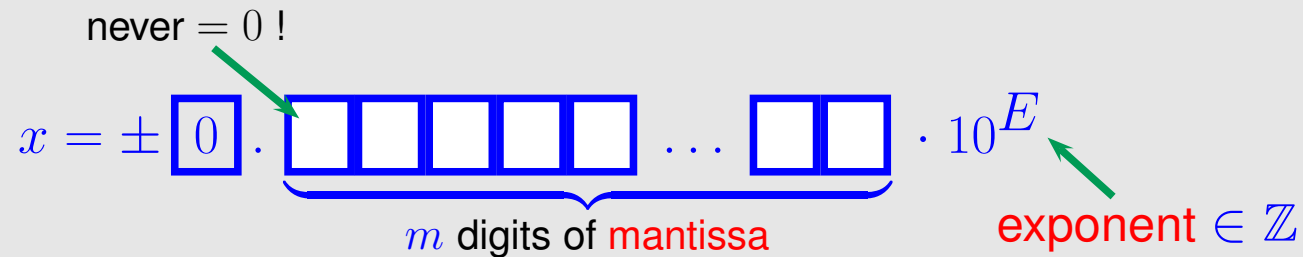
“Computers use floating point numbers (scientific notation)”

Example 2.4.1 (Decimal floating point numbers).

3-digit normalized decimal floating point numbers:

valid:  $0.723 \cdot 10^2$  ,  $0.100 \cdot 10^{-20}$  ,  $-0.801 \cdot 10^5$   
 invalid:  $0.033 \cdot 10^2$  ,  $1.333 \cdot 10^{-4}$  ,  $-0.002 \cdot 10^3$

General form of  $m$ -digit normalized decimal floating point number:

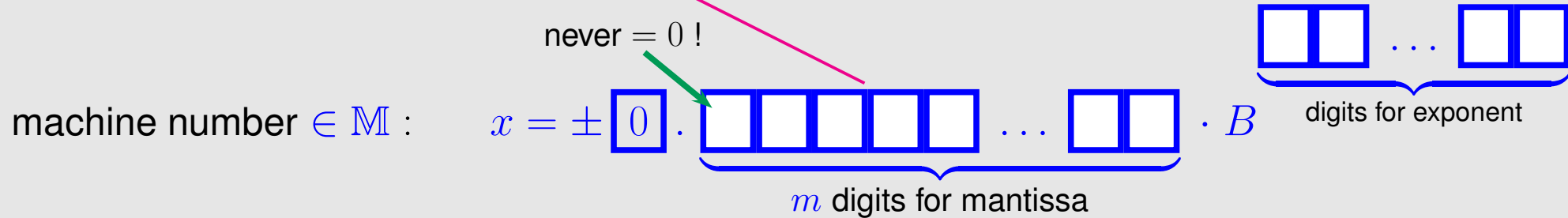


Of course, computers are restricted to a *finite range* of exponents.

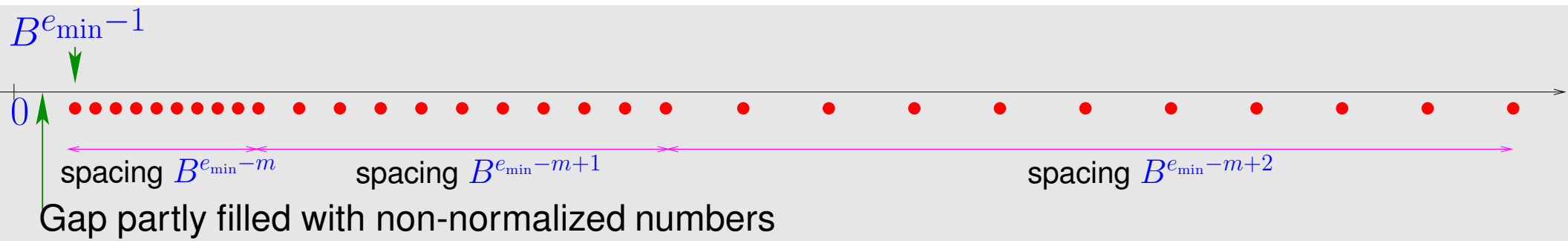
**Definition 2.4.2** (Machine numbers/floating point numbers).

- Given
- ☞ **basis**  $B \in \mathbb{N} \setminus \{1\}$ ,
  - ☞ **exponent range**  $\{e_{\min}, \dots, e_{\max}\}$ ,  $e_{\min}, e_{\max} \in \mathbb{Z}$ ,  $e_{\min} < e_{\max}$ ,
  - ☞ **number**  $m \in \mathbb{N}$  of digits (for **mantissa**),
- the corresponding set of **machine numbers** is

$$\mathbb{M} := \{d \cdot B^E : d = i \cdot B^{-m}, i = B^{m-1}, \dots, B^m - 1, E \in \{e_{\min}, \dots, e_{\max}\}\}$$



- ▶ Largest machine number (in modulus) :  $x_{\max} = (1 - B^{-m}) \cdot B^{e_{\max}}$
- Smallest machine number (in modulus) :  $x_{\min} = B^{-1} \cdot B^{e_{\min}}$



Example 2.4.3 (IEEE standard 754 for machine numbers). → [49], → link

No surprise: for modern computers  $B = 2$  (binary system)

single precision :  $m = 24^*$ ,  $E \in \{-125, \dots, 128\}$  ➤ 4 bytes

double precision :  $m = 53^*$ ,  $E \in \{-1021, \dots, 1024\}$  ➤ 8 bytes

\*: including bit indicating sign



Remark 2.4.4 (Special cases in IEEE standard).

```

1 >> x = exp(1000), y = 3/x, z = x*sin(pi), w =
  x*log(1)
2     x = Inf
3     y = 0
4     z = Inf
5     w = NaN

```



$E = e_{\max}, M \neq 0 \hat{=} \text{NaN} = \text{Not a number} \rightarrow \text{exception}$


$E = e_{\max}, M = 0 \hat{=} \text{Inf} = \text{Infinity} \rightarrow \text{overflow}$

$E = 0 \hat{=} \text{Non-normalized numbers} \rightarrow \text{underflow}$

$E = 0, M = 0 \hat{=} \text{number } 0$



*Example 2.4.5* (Characteristic parameters of IEEE floating point numbers (double precision)).

 MATLAB *always* uses double precision

```

1 >> format hex; realmin, format long; realmin
2 ans = 001000000000000000
3 ans = 2.225073858507201e-308

```

```

4 >> format hex; realmax, format long; realmax
5 ans = 7fefffffffffffffffff
6 ans = 1.797693134862316e+308

```

*Example 2.4.6* (Input errors and roundoff errors).

#### Code 2.4.7: input errors and roundoff errors

```

1 >> format long;
2 >> a = 4/3; b = a-1; c = 3*b; e = 1-c
3 e = 2.220446049250313e-16
4 >> a = 1012/113; b = a-9; c = 113*b; e = 5+c
5 e = 6.750155989720952e-14
6 >> a = 83810206/6789; b = a-12345; c = 6789*b; e = c-1
7 e = -1.607986632734537e-09

```



correct rounding:

$$\text{rd}(x) = \arg \min_{\tilde{x} \in \mathbb{M}} |x - \tilde{x}|$$

(if non-unique, round to larger (in modulus)  $\tilde{x} \in \mathbb{M}$ : “rounding up”)

For any reasonable  $\mathbb{M}$ : small relative rounding error

$$\exists \text{eps} \ll 1: \frac{|\text{rd}(x) - x|}{|x|} \leq \text{eps} \quad \forall x \in \mathbb{R}. \quad (2.4.8)$$

► Realization of  $\tilde{+}, \tilde{-}, \tilde{\cdot}, \tilde{/}$ :

$$\star \in \{+, -, \cdot, /\}: \quad x \tilde{\star} y := \text{rd}(x \star y) \quad (2.4.9)$$

**Assumption 2.4.10** (“Axiom” of roundoff analysis).

There is a small positive number  $\text{eps}$ , the *machine precision*, such that for the elementary arithmetic operations  $\star \in \{+, -, \cdot, /\}$  and “hard-wired” functions\*  $f \in \{\exp, \sin, \cos, \log, \dots\}$  holds

$$x \tilde{\star} y = (x \star y)(1 + \delta) \quad , \quad \tilde{f}(x) = f(x)(1 + \delta) \quad \forall x, y \in \mathbb{M} ,$$

with  $|\delta| < \text{eps}$ .

\*: this is an ideal, which may not be accomplished even by modern CPUs.

▶ relative roundoff errors of elementary steps in a program bounded by machine precision !

Example 2.4.11 (Machine precision for MATLAB). (CPU Intel Pentium)

Code 2.4.12: Finding out `eps` in MATLAB

```
1 >> format hex; eps, format long; eps
2 ans = 3cb0000000000000
3 ans = 2.220446049250313e-16
```



Remark 2.4.13 (Adding `eps` to 1).

`eps` is the smallest positive number  $\in \mathbb{M}$  for which  $1 + \epsilon \neq 1$  (in  $\mathbb{M}$ ):

Code 2.4.14:  $1 + \epsilon$  in MATLAB

```
1 >> fprintf (' %30.25f\n' , 1+0.5*eps)
2 1.000000000000000000000000000000
3 >> fprintf (' %30.25f\n' , 1-0.5*eps)
4 0.999999999999999999998889776975
5 >>
6 fprintf (' %30.25f\n' , (1+2/eps) - 2/eps);
6 0.000000000000000000000000000000
```



In fact  $1 \tilde{+} \text{eps} = 1$  would comply with the “axiom” of roundoff error analysis, Ass. 2.4.10:

$$1 = (1 + \text{eps})(1 + \delta) \Rightarrow |\delta| = \left| \frac{\text{eps}}{1 + \text{eps}} \right| \leq \text{eps} ,$$

$$\frac{2}{\text{eps}} = \left(1 + \frac{2}{\text{eps}}\right)(1 + \delta) \Rightarrow |\delta| = \left| \frac{\text{eps}}{2 + \text{eps}} \right| \leq \text{eps} .$$



Do we have to worry about these tiny roundoff errors ?



**YES**

(→ Sect. 2.3):

- accumulation of roundoff errors
- amplification of roundoff errors

# 2.5 Stability of Gaussian Elimination

Issue: Gauge impact of roundoff errors on Gaussian elimination with partial pivoting !

## 2.5.1 Vector norms and matrix norms [13, Sect. 2.1.2], [35, Sect. 1.2], [51, Sect. 1.11]

Norms provide tools for measuring errors. Recall from linear algebra and calculus [48, Sect. 4.3], [27, Sect. 6.1]:

### Definition 2.5.1 (Norm).

$X$  = vector space over field  $\mathbb{K}$ ,  $\mathbb{K} = \mathbb{C}, \mathbb{R}$ . A map  $\|\cdot\| : X \mapsto \mathbb{R}_0^+$  is a *norm* on  $X$ , if it satisfies

- (i)  $\forall \mathbf{x} \in X: \mathbf{x} \neq 0 \Leftrightarrow \|\mathbf{x}\| > 0$  (definite),
- (ii)  $\|\lambda \mathbf{x}\| = |\lambda| \|\mathbf{x}\| \quad \forall \mathbf{x} \in X, \lambda \in \mathbb{K}$  (homogeneous),
- (iii)  $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in X$  (triangle inequality).

Examples: (for vector space  $\mathbb{K}^n$ , vector  $\mathbf{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{K}^n$ )

name	:	definition	MATLAB function
Euclidean norm	:	$\ \mathbf{x}\ _2 := \sqrt{ x_1 ^2 + \dots +  x_n ^2}$	<code>norm(x)</code>
1-norm	:	$\ \mathbf{x}\ _1 :=  x_1  + \dots +  x_n $	<code>norm(x, 1)</code>
$\infty$ -norm, max norm	:	$\ \mathbf{x}\ _\infty := \max\{ x_1 , \dots,  x_n \}$	<code>norm(x, inf)</code>

Simple explicit norm equivalences: for all  $\mathbf{x} \in \mathbb{K}^n$

$$\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1 \leq \sqrt{n} \|\mathbf{x}\|_2, \quad (2.5.2)$$

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{n} \|\mathbf{x}\|_\infty, \quad (2.5.3)$$

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_1 \leq n \|\mathbf{x}\|_\infty. \quad (2.5.4)$$

**Definition 2.5.5** (Matrix norm).

Given a vector norm  $\|\cdot\|$  on  $\mathbb{R}^n$ , the associated *matrix norm* is defined by

$$\mathbf{M} \in \mathbb{R}^{m,n}: \quad \|\mathbf{M}\| := \sup_{\mathbf{x} \in \mathbb{R}^n \setminus \{0\}} \frac{\|\mathbf{M}\mathbf{x}\|}{\|\mathbf{x}\|}.$$

sub-multiplicative:  $\mathbf{A} \in \mathbb{K}^{n,m}, \mathbf{B} \in \mathbb{K}^{m,k}: \|\mathbf{AB}\| \leq \|\mathbf{A}\| \|\mathbf{B}\|$

notation:  $\|\mathbf{x}\|_2 \rightarrow \|\mathbf{M}\|_2, \|\mathbf{x}\|_1 \rightarrow \|\mathbf{M}\|_1, \|\mathbf{x}\|_\infty \rightarrow \|\mathbf{M}\|_\infty$

*Example 2.5.6* (Matrix norm associated with  $\infty$ -norm and 1-norm).

e.g. for  $\mathbf{M} \in \mathbb{K}^{2,2}$ :

$$\begin{aligned} \|\mathbf{M}\mathbf{x}\|_\infty &= \max\{|m_{11}x_1 + m_{12}x_2|, |m_{21}x_1 + m_{22}x_2|\} \\ &\leq \max\{|m_{11}| + |m_{12}|, |m_{21}| + |m_{22}|\} \|\mathbf{x}\|_\infty, \\ \|\mathbf{M}\mathbf{x}\|_1 &= |m_{11}x_1 + m_{12}x_2| + |m_{21}x_1 + m_{22}x_2| \\ &\leq \max\{|m_{11}| + |m_{21}|, |m_{12}| + |m_{22}|\} (|x_1| + |x_2|). \end{aligned}$$

For general  $\mathbf{M} \in \mathbb{K}^{m,n}$

➤ matrix norm  $\leftrightarrow \|\cdot\|_\infty =$  **row sum norm**  $\|\mathbf{M}\|_\infty := \max_{i=1,\dots,m} \sum_{j=1}^n |m_{ij}|, \quad (2.5.7)$

➤ matrix norm  $\leftrightarrow \|\cdot\|_1 =$  **column sum norm**  $\|\mathbf{M}\|_1 := \max_{j=1,\dots,n} \sum_{i=1}^m |m_{ij}|. \quad (2.5.8)$

◇

**Lemma 2.5.9** (Formula for Euclidean norm of a Hermitian matrix).

$$\mathbf{A} \in \mathbb{K}^{n,n}, \mathbf{A} = \mathbf{A}^H \Rightarrow \|\mathbf{A}\|_2 = \max_{\mathbf{x} \neq 0} \frac{|\mathbf{x}^H \mathbf{A} \mathbf{x}|}{\|\mathbf{x}\|_2^2}.$$

*Proof.* Recall from linear algebra: Hermitian matrices (a special class of normal matrices) enjoy unitary similarity to diagonal matrices:

$$\exists \mathbf{U} \in \mathbb{K}^{n,n}, \text{ diagonal } \mathbf{D} \in \mathbb{R}^{n,n}: \mathbf{U}^{-1} = \mathbf{U}^H \text{ and } \mathbf{A} = \mathbf{U}^H \mathbf{D} \mathbf{U}.$$

Since multiplication with an unitary matrix preserves the 2-norm of a vector, we conclude

$$\|\mathbf{A}\|_2 = \left\| \mathbf{U}^H \mathbf{D} \mathbf{U} \right\|_2 = \|\mathbf{D}\|_2 = \max_{i=1, \dots, n} |d_i|, \quad \mathbf{D} = \text{diag}(d_1, \dots, d_n).$$

On the other hand, for the same reason:

$$\max_{\|\mathbf{x}\|_2=1} \mathbf{x}^H \mathbf{A} \mathbf{x} = \max_{\|\mathbf{x}\|_2=1} (\mathbf{U} \mathbf{x})^H \mathbf{D} (\mathbf{U} \mathbf{x}) = \max_{\|\mathbf{y}\|_2=1} \mathbf{y}^H \mathbf{D} \mathbf{y} = \max_{i=1, \dots, n} |d_i|. \quad \square$$

**Corollary 2.5.10** (2-Matrixnorm and eigenvalues).

For  $\mathbf{A} \in \mathbb{K}^{m,n}$  the 2-Matrixnorm  $\|\mathbf{A}\|_2$  is the square root of the largest (in modulus) eigenvalue of  $\mathbf{A}^H \mathbf{A}$ .



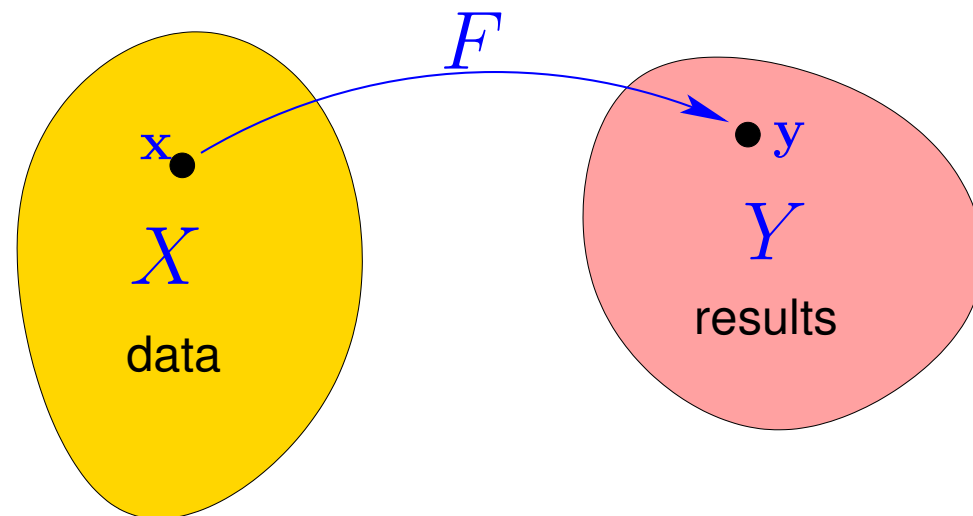
### Supplementary and further reading:

Also see the article [7] for a simplified but very instructive presentation of stability analysis of numerical algorithms.

Abstract point of view:

Our notion of “**problem**”:

- data space  $X$ , usually  $X \subset \mathbb{R}^n$
- result space  $Y$ , usually  $Y \subset \mathbb{R}^m$
- mapping (problem function)  $F : X \mapsto Y$



Application to linear system of equations  $\mathbf{Ax} = \mathbf{b}$ ,  $\mathbf{A} \in \mathbb{K}^{n,n}$ ,  $\mathbf{b} \in \mathbb{K}^n$ :

- “The problem:”
- data  $\hat{=}$  system matrix  $\mathbf{A} \in \mathbb{R}^{n,n}$ , right hand side vector  $\mathbf{b} \in \mathbb{R}^n$ 
    - data space  $X = \mathbb{R}^{n,n} \times \mathbb{R}^n$  with vector/matrix norms ( $\rightarrow$  Defs. 2.5.1, 2.5.5)
  - problem mapping  $(\mathbf{A}, \mathbf{b}) \mapsto F(\mathbf{A}, \mathbf{b}) := \mathbf{A}^{-1}\mathbf{b}$ , (for regular  $\mathbf{A}$ )

Stability is a property of a particular algorithm for a problem

Numerical algorithm = Specific sequence of elementary operations  
( $\rightarrow$  programme in C++ or FORTRAN)

Below:  $X, Y$  = normed vector spaces, e.g.,  $X = \mathbb{R}^n$ ,  $Y = \mathbb{R}^m$

**Definition 2.5.11** (Stable algorithm).

An algorithm  $\tilde{F}$  for solving a problem  $F : X \mapsto Y$  is **numerically stable**, if for all  $x \in X$  its result  $\tilde{F}(\mathbf{x})$  (affected by roundoff) is the exact result for “slightly perturbed” data:

$$\exists C \approx 1: \forall \mathbf{x} \in X: \exists \tilde{\mathbf{x}} \in X: \|\mathbf{x} - \tilde{\mathbf{x}}\| \leq C \text{ eps} \|\mathbf{x}\| \quad \wedge \quad \tilde{F}(\mathbf{x}) = F(\tilde{\mathbf{x}}) .$$

- Judgement about the stability of an algorithm depends on the chosen norms !
- What is meant by  $C \approx 1$  in practice ?  
 $C \approx 1 \leftrightarrow C \approx$  no. of elementary operations for computing  $\tilde{F}(\mathbf{x})$ : The longer a computation takes the more accumulation of roundoff errors we are willing to tolerate.
- The use of computer arithmetic involves inevitable relative **input errors** ( $\rightarrow$  Ex. 2.4.6) of the size of **eps**. Moreover, in most applications the input data are also affected by other (e.g, measurement) errors. Hence stability means that

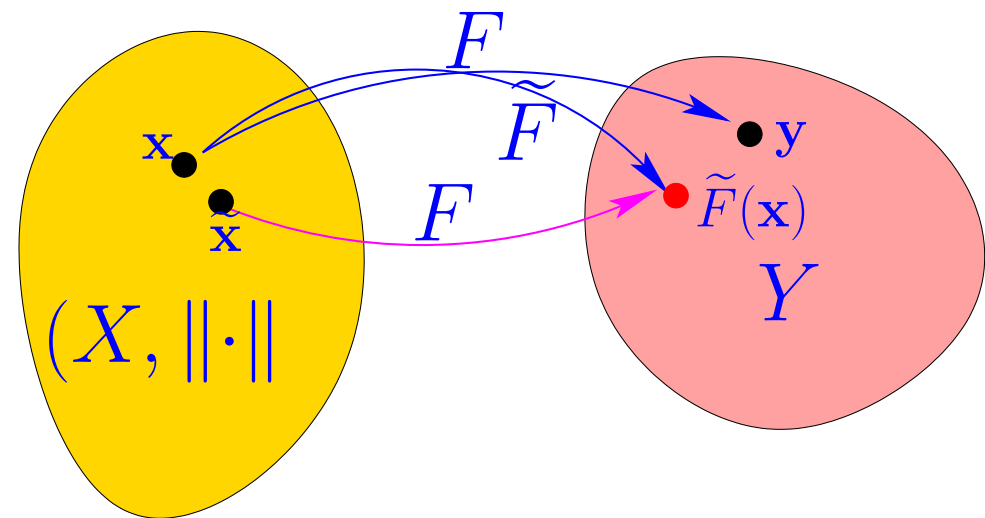
Roundoff errors affect the result in the same way as inevitable data errors.

➤ for stable algorithms roundoff errors are “harmless”.



Terminology:

Def. 2.5.11 introduces stability in the sense of **backward error analysis**





## 2.5.3 Roundoff analysis of Gaussian elimination

Simplification: equivalence of Gaussian elimination and LU-factorization extends to machine arithmetic, *cf.* Sect. 2.2

**Lemma 2.5.12** (Equivalence of Gaussian elimination and LU-factorization).

The following algorithms for solving the LSE  $\mathbf{Ax} = \mathbf{b}$  ( $\mathbf{A} \in \mathbb{K}^{n,n}$ ,  $\mathbf{b} \in \mathbb{K}^n$ ) are numerically equivalent:

- ➊ Gaussian elimination (forward elimination and back substitution) without pivoting, see Algorithm 2.1.3.
- ➋ LU-factorization of  $\mathbf{A}$  ( $\rightarrow$  Code 2.2.6) followed by forward and backward substitution, see Algorithm 2.2.12.

Rem. 2.3.15 ➤ sufficient to consider LU-factorization without pivoting

A profound roundoff analysis of Gaussian elimination/LU-factorization can be found in [23, Sect. 3.3 & 3.5] and [36, Sect. 9.3]. A less rigorous, but more lucid discussion is given in [65, Lecture 22].

Here we only quote a result due to Wilkinson, [36, Thm. 9.5]:

**Theorem 2.5.13** (Stability of Gaussian elimination with partial pivoting).

Let  $\mathbf{A} \in \mathbb{R}^{n,n}$  be regular and  $\mathbf{A}^{(k)} \in \mathbb{R}^{n,n}$ ,  $k = 1, \dots, n-1$ , denote the intermediate matrix arising in the  $k$ -th step of Algorithm 2.3.8 (Gaussian elimination with partial pivoting) when carried out with exact arithmetic.

For the approximate solution  $\tilde{\mathbf{x}} \in \mathbb{R}^n$  of the LSE  $\mathbf{A}\mathbf{x} = \mathbf{b}$ ,  $\mathbf{b} \in \mathbb{R}^n$ , computed by Algorithm 2.3.8 (based on machine arithmetic with machine precision  $\text{eps}$ ,  $\rightarrow$  Ass. 2.4.10) there is  $\Delta\mathbf{A} \in \mathbb{R}^{n,n}$  with

$$\|\Delta\mathbf{A}\|_{\infty} \leq n^3 \frac{3\text{eps}}{1 - 3n\text{eps}} \rho \|\mathbf{A}\|_{\infty}, \quad \rho := \frac{\max_{i,j,k} |(\mathbf{A}^{(k)})_{ij}|}{\max_{i,j} |(\mathbf{A})_{ij}|},$$

$$\text{such that } (\mathbf{A} + \Delta\mathbf{A})\tilde{\mathbf{x}} = \mathbf{b}.$$

$\rho$  “small”  $\rightarrow$  Gaussian elimination with partial pivoting is stable ( $\rightarrow$  Def. 2.5.11)

If  $\rho$  is “small”, the computed solution of a LSE can be regarded as the exact solution of a LSE with “slightly perturbed” system matrix (perturbations of size  $O(n^3 \text{eps})$ ).



Bad news:

exponential growth  $\rho \sim 2^n$  is possible !

Example 2.5.14 (Wilkinson’s counterexample).

$$a_{ij} = \begin{cases} 1 & , \text{ if } i = j \vee j = n , \\ -1 & , \text{ if } i > j , \\ 0 & \text{ else.} \end{cases} ,$$

$n=10:$

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

Partial pivoting does not trigger row permutations !

$$\blacktriangleright \quad \mathbf{A} = \mathbf{LU}, \quad l_{ij} = \begin{cases} 1 & , \text{ if } i = j, \\ -1 & , \text{ if } i > j, \\ 0 & \text{ else} \end{cases}, \quad u_{ij} = \begin{cases} 1 & , \text{ if } i = j, \\ 2^{i-1} & , \text{ if } j = n, \\ 0 & \text{ else.} \end{cases}$$

$\blacktriangleright$  Exponential blow-up of entries of  $\mathbf{U}$  !



Observation: In practice  $\rho$  (almost) always grows only mildly (like  $O(\sqrt{n})$ ) with  $n$

Discussion in [65, Lecture 22]: growth factors larger than the order  $O(\sqrt{n})$  are exponentially rare in certain relevant classes of *random matrices*.

*Example 2.5.15* (Stability by small random perturbations).

```

1 % Curing Wilkinson's counterexample by random perturbation
2 % Theory: Spielman and Teng
3 res = [];
4 for n=10:10:200
5     % Build Wilkinson matrix
6     A = [ tril(-ones(n,n-1)) + 2*eye(n-1);
```

```
7      zeros (1, n-1) ], ones (n, 1) ] ;
8      % imposed solution
9      x = ((-1) .^(1:n))' ;
10     relerr = norm (A \ (A*x) - x) / norm (x) ;
11     % Randomly perturbed Wilkinson matrix by matrix with iid
12     % N(0,eps) distributed entries
13     Ap = A + eps*randn (size (A)) ;
14     relerrp = norm (Ap \ (A*x) - x) / norm (x) ;
15     res = [res; n relerr relerrp];
16 end
17 semilogy (res(:,1), res(:,2), 'm-*', res(:,1), res(:,3), 'r-+') ;
18 xlabel ('matrix size n', 'fontsize', 14) ;
19 ylabel ('relative error', 'fontsize', 14) ;
20 legend ('unperturbed matrix', 'randn perturbed
21         matrix', 'location', 'west') ;
22 print -depsc2 '../PICTURES/wilkpert.eps' ;
```

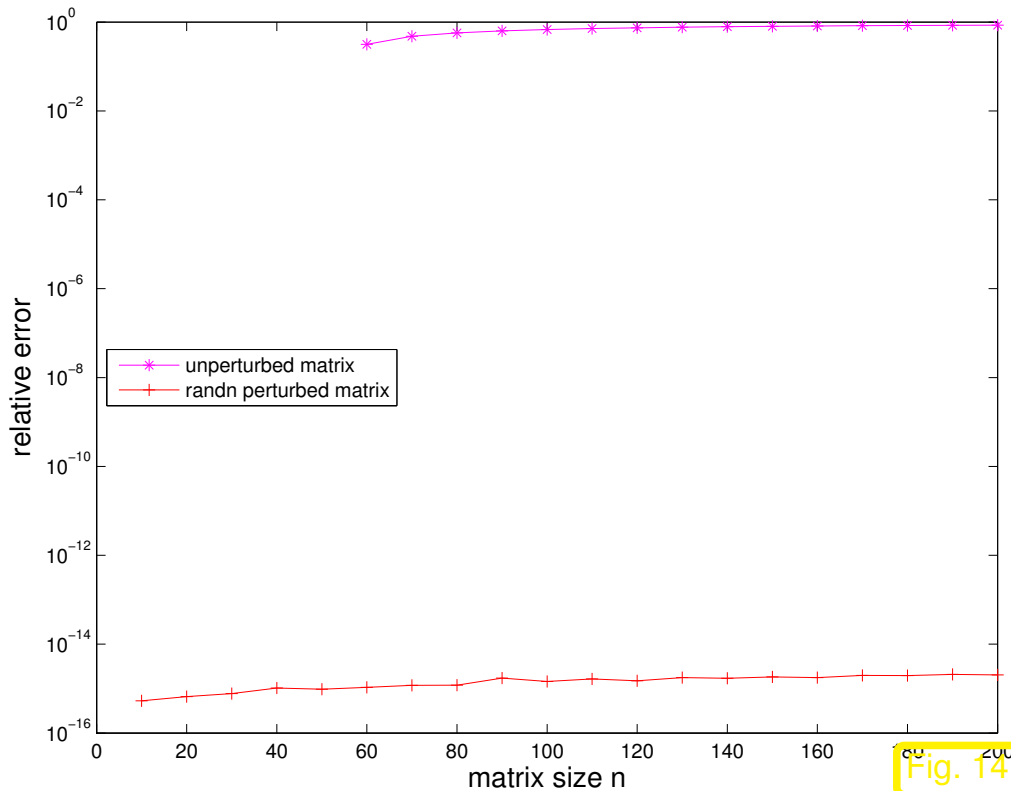


Fig. 14

Recall statement in Sect. 2.5.3 about “improbability” of matrices for which Gaussian elimination with partial pivoting is unstable. This is now matched by the observation that a *tiny random* perturbation of the matrix (almost certainly) cures the problem. This is investigated by the brand-new field of **smoothed analysis** of numerical algorithms, see [56].



*Gaussian elimination/LU-factorization with partial pivoting is stable (\*)*  
(for all practical purposes) !

(\*): stability refers to maximum norm  $\|\cdot\|_{\infty}$ .

In practice *Gaussian elimination/LU-factorization with partial pivoting* produces “relatively **small residuals**”

**Definition 2.5.16** (Residual).

Given an approximate solution  $\tilde{\mathbf{x}} \in \mathbb{K}^n$  of the LSE  $\mathbf{Ax} = \mathbf{b}$  ( $\mathbf{A} \in \mathbb{K}^{n,n}$ ,  $\mathbf{b} \in \mathbb{K}^n$ ), its *residual* is the vector

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}} .$$

Simple consideration:

$$(\mathbf{A} + \Delta\mathbf{A})\tilde{\mathbf{x}} = \mathbf{b} \Rightarrow \mathbf{r} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}} = \Delta\mathbf{A}\tilde{\mathbf{x}} \Rightarrow \|\mathbf{r}\| \leq \|\Delta\mathbf{A}\| \|\tilde{\mathbf{x}}\| ,$$

for any vector norm  $\|\cdot\|$ .

*Example 2.5.17* (Small residuals by Gaussian elimination).

## Code 2.5.18: small residuals for GE

Numerical experiment with *nearly singular matrix*

$$\mathbf{A} = \mathbf{u}\mathbf{v}^T + \epsilon\mathbf{I},$$

singular rank-1 matrix

with

$$\mathbf{u} = \frac{1}{3}(1, 2, 3, \dots, 10)^T,$$

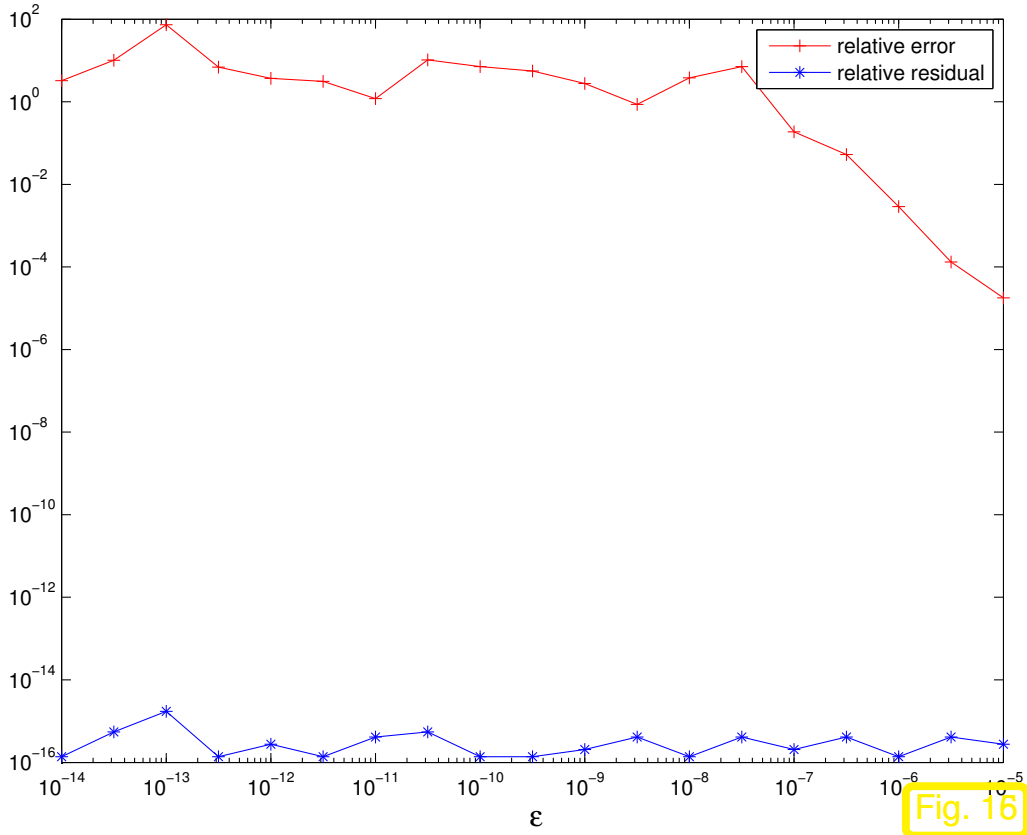
$$\mathbf{v} = \left(-1, \frac{1}{2}, -\frac{1}{3}, \frac{1}{4}, \dots, \frac{1}{10}\right)^T$$

```

1 n = 10; u = (1:n)' / 3; v =
  (1./u) .* (-1) .^ ((1:n)');
2 x = ones(10,1); nx = norm(x, 'inf');
3
4 result = [];
5 for epsilon = 10.^(-5:-0.5:-14)
6   A = u*v' + epsilon*eye(n);
7   b = A*x; nb = norm(b, 'inf');
8   xt = A\b; % Gaussian elimination
9   r = b - A*xt; % residual
10  result = [result; epsilon,
             norm(x-xt, 'inf')/nx,
             norm(r, 'inf')/nb, cond(A, 'inf')];
11 end

```





Observations (w.r.t  $\|\cdot\|_\infty$ -norm)

- for  $\epsilon \ll 1$  large relative error in computed solution  $\tilde{\mathbf{x}}$
- small residuals for any  $\epsilon$

How can a *large* relative error be reconciled with a *small* relative residual ?

$$\mathbf{Ax} = \mathbf{b} \quad \leftrightarrow \quad \mathbf{A}\tilde{\mathbf{x}} \approx \mathbf{b}$$

$$\begin{cases} \mathbf{A}(\mathbf{x} - \tilde{\mathbf{x}}) = \mathbf{r} \Rightarrow \|\mathbf{x} - \tilde{\mathbf{x}}\| \leq \|\mathbf{A}^{-1}\| \|\mathbf{r}\| \\ \mathbf{Ax} = \mathbf{b} \Rightarrow \|\mathbf{b}\| \leq \|\mathbf{A}\| \|\mathbf{x}\| \end{cases} \Rightarrow \frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}\| \|\mathbf{A}^{-1}\| \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}. \quad (2.5.19)$$

➤ If  $\|\mathbf{A}\| \|\mathbf{A}^{-1}\| \gg 1$ , then a small relative residual may not imply a small relative error.

*Example 2.5.20* (Instability of multiplication with inverse).

Nearly singular matrix from Ex. 2.5.17

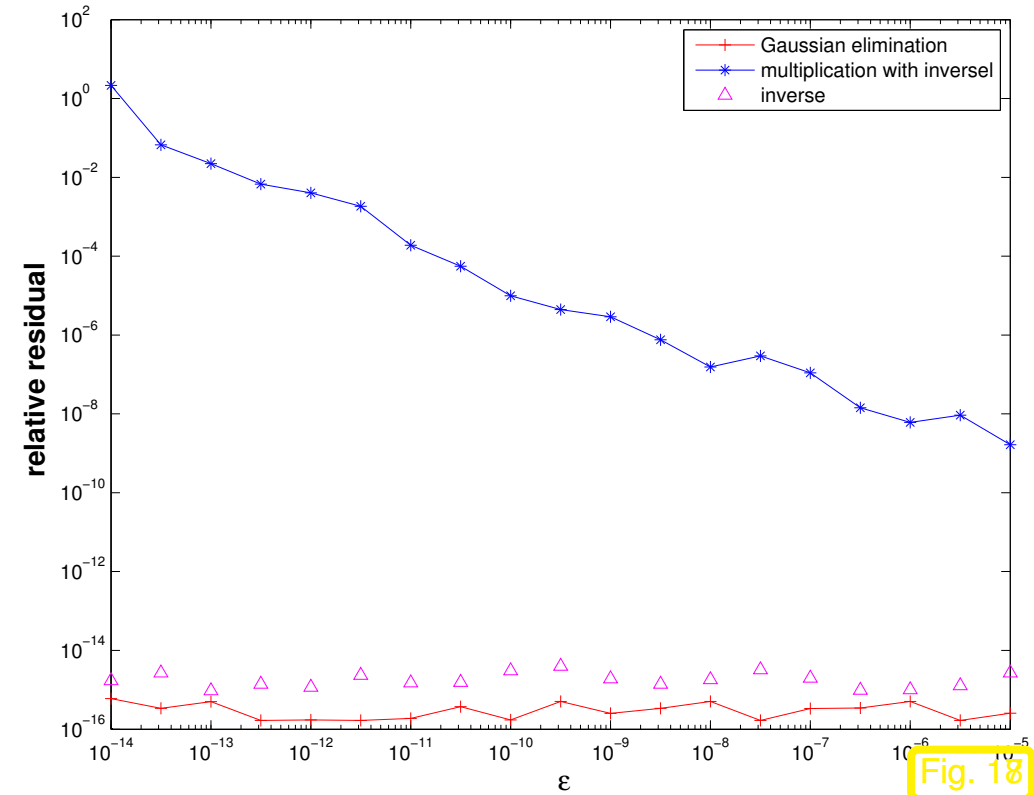
## Code 2.5.22: instability of multiplication with inverse

```

1 n = 10; u = (1:n)' / 3; v =
  (1./u) .* (-1).^u;
2 x = ones(10,1); nx =
  norm(x, 'inf');
3
4 result = [];
5 for epsilon = 10.^(-5:-0.5:-14)
6   A = u*v' +
     epsilon*rand(n,n);
7   b = A*x; nb = norm(b, 'inf');
8   xt = A\b;      % Gaussian
                   elimination
9   r = b - A*xt;  % residualB
10  B = inv(A); xi = B*b;
11  ri = b - A*xi; % residual
12  R = eye(n) - A*B; % residual
13  result = [result; epsilon,
             norm(r, 'inf')/nb,
             norm(ri, 'inf')/nb,
             norm(R, 'inf')/norm(B, 'inf')
            ];

```

14 end



Computation of the inverse  $\mathbf{B} := \text{inv}(\mathbf{A})$  is affected by roundoff errors, but does not benefit from favorable compensation of roundoff errors as does Gaussian elimination.



## 2.5.4 Conditioning

Considered: linear system of equations  $\mathbf{Ax} = \mathbf{b}$ ,  $\mathbf{A} \in \mathbb{R}^{n,n}$  regular,  $\mathbf{b} \in \mathbb{R}^n$   
 $\hat{\mathbf{x}} \in \mathbb{M}^n \hat{=} \text{computed solution (by Gaussian elimination with partial pivoting)}$

Question: implications of stability results ( $\rightarrow$  previous section) for

(normwise) **relative error**: 
$$\epsilon_r := \frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} .$$

( $\|\cdot\| \hat{=} \text{suitable vector norm, e.g., maximum norm } \|\cdot\|_\infty$ )

Perturbed linear system:

$$\mathbf{Ax} = \mathbf{b} \iff (\mathbf{A} + \Delta\mathbf{A})\tilde{\mathbf{x}} = \mathbf{b} + \Delta\mathbf{b} \blacktriangleright (\mathbf{A} + \Delta\mathbf{A})(\tilde{\mathbf{x}} - \mathbf{x}) = \Delta\mathbf{b} - \Delta\mathbf{Ax} . \quad (2.5.23)$$

**Theorem 2.5.24** (Conditioning of LSEs).  $\rightarrow$  [51, Thm. 3.1]

If  $\mathbf{A}$  regular,  $\|\Delta\mathbf{A}\| < \|\mathbf{A}^{-1}\|^{-1}$  and (2.5.23), then

$$\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\|\mathbf{A}^{-1}\| \|\mathbf{A}\|}{1 - \|\mathbf{A}^{-1}\| \|\mathbf{A}\| \|\Delta\mathbf{A}\| / \|\mathbf{A}\|} \left( \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \right).$$

*relative error*
*relative perturbations*

The proof is based on the following fundamental result:

**Lemma 2.5.25** (Perturbation lemma).  $\rightarrow$  [51, Thm. 1.5]

$$\mathbf{B} \in \mathbb{R}^{n,n}, \|\mathbf{B}\| < 1 \Rightarrow \mathbf{I} + \mathbf{B} \text{ regular} \wedge \left\| (\mathbf{I} + \mathbf{B})^{-1} \right\| \leq \frac{1}{1 - \|\mathbf{B}\|}.$$

*Proof.*  $\triangle$ -inequality  $\triangleright \|\mathbf{I} + \mathbf{B}\mathbf{x}\| \geq (1 - \|\mathbf{B}\|) \|\mathbf{x}\|, \forall \mathbf{x} \in \mathbb{R}^n \triangleright \mathbf{I} + \mathbf{B}$  regular.

$$\blacktriangleright \left\| (\mathbf{I} + \mathbf{B})^{-1} \right\| = \sup_{\mathbf{x} \in \mathbb{R}^n \setminus \{0\}} \frac{\|(\mathbf{I} + \mathbf{B})^{-1}\mathbf{x}\|}{\|\mathbf{x}\|} = \sup_{\mathbf{y} \in \mathbb{R}^n \setminus \{0\}} \frac{\|\mathbf{y}\|}{\|(\mathbf{I} + \mathbf{B})\mathbf{y}\|} \leq \frac{1}{1 - \|\mathbf{B}\|}$$

*Proof* (of Thm. 2.5.24) Lemma 2.5.25  $\triangleright \left\| (\mathbf{A} + \Delta\mathbf{A})^{-1} \right\| \leq \frac{\left\| \mathbf{A}^{-1} \right\|}{1 - \left\| \mathbf{A}^{-1} \Delta\mathbf{A} \right\|} \quad \& \quad (2.5.23)$

$$\Rightarrow \|\Delta\mathbf{x}\| \leq \frac{\|\mathbf{A}^{-1}\|}{1 - \|\mathbf{A}^{-1}\Delta\mathbf{A}\|} (\|\Delta\mathbf{b}\| + \|\Delta\mathbf{A}\mathbf{x}\|) \leq \frac{\|\mathbf{A}^{-1}\| \|\mathbf{A}\|}{1 - \|\mathbf{A}^{-1}\| \|\Delta\mathbf{A}\|} \left( \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{A}\| \|\mathbf{x}\|} + \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \right) \|\mathbf{x}\|$$

**Definition 2.5.26** (Condition (number) of a matrix).

*Condition* (number) of a matrix  $\mathbf{A} \in \mathbb{R}^{n,n}$ :

$$\text{cond}(\mathbf{A}) := \left\| \mathbf{A}^{-1} \right\| \|\mathbf{A}\|$$

Note:  $\text{cond}(\mathbf{A})$  depends on  $\|\cdot\|$  !

Rewriting estimate of Thm. 2.5.24 with  $\Delta\mathbf{b} = 0$ ,

$$\epsilon_r := \frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\text{cond}(\mathbf{A})\delta_A}{1 - \text{cond}(\mathbf{A})\delta_A}, \quad \delta_A := \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}. \quad (2.5.27)$$

- (2.5.27) ➤
- If  $\text{cond}(\mathbf{A}) \gg 1$ , small perturbations in  $\mathbf{A}$  can lead to large relative errors in the solution of the LSE.
  - If  $\text{cond}(\mathbf{A}) \gg 1$ , a stable algorithm ( $\rightarrow$  Def. 2.5.11) can produce solutions with large relative error !

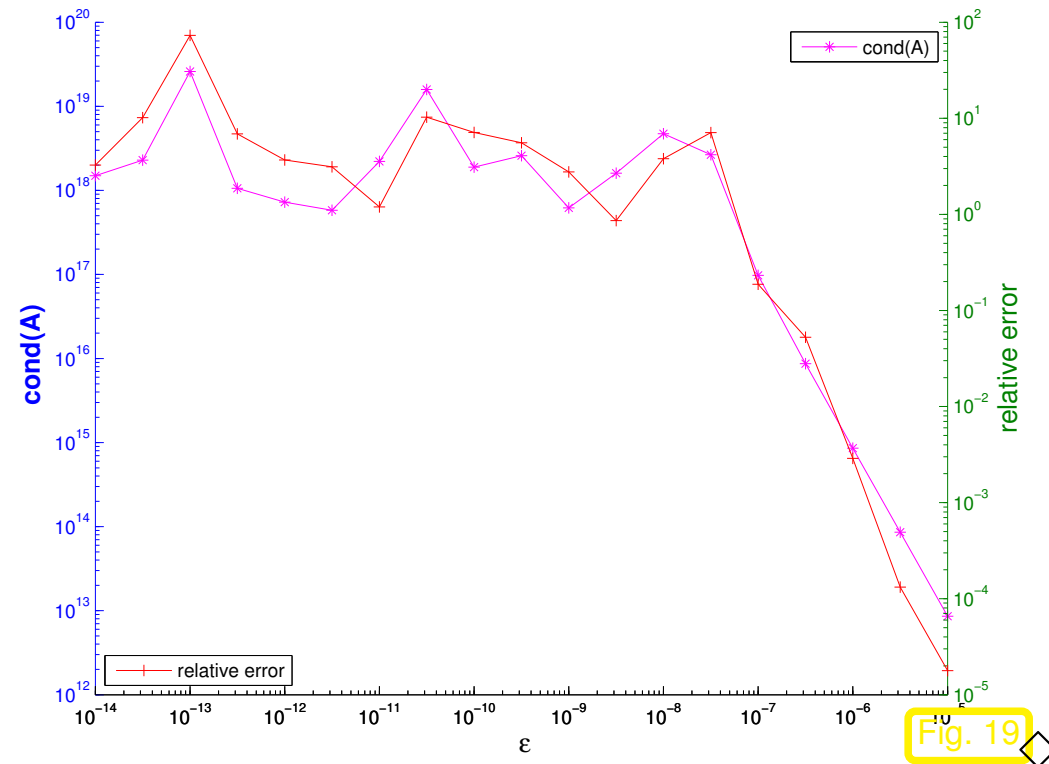
Example 2.5.28 (Conditioning and relative error).  $\rightarrow$  Ex. 2.5.17 cnt'd

Numerical experiment with *nearly singular matrix* from Ex. 2.5.17

$$\mathbf{A} = \mathbf{u}\mathbf{v}^T + \epsilon\mathbf{I},$$

$$\mathbf{u} = \frac{1}{3}(1, 2, 3, \dots, 10)^T,$$

$$\mathbf{v} = \left(-1, \frac{1}{2}, -\frac{1}{3}, \frac{1}{4}, \dots, \frac{1}{10}\right)^T$$



Example 2.5.29 (Wilkinson's counterexample cnt'd).  $\rightarrow$  Ex. 2.5.14

## Code 2.5.30: GE for “Wilkinson system”

```

1 res = [];
2 for n=10:10:1000
3     A =
4         [ tril (-ones (n, n-1)) + 2 * [ eye (n-1);
5             zeros (1, n-1) ], ones (n, 1) ];
6     x = ((-1) .^(1:n))';
7     relerr = norm (A \ (A*x) - x) / norm (x);
8     res = [res; n, relerr];
9 end
10 plot (res (:, 1), res (:, 2), 'm-*');

```

Blow-up of entries of  $\mathbf{U}$  !

↕ (\*)

However,  $\text{cond}_2(\mathbf{A})$  is small!

▷ **Instability** of Gaussian elimination !

(\*) If  $\text{cond}_2(\mathbf{A})$  was huge, then big errors in the solution of a linear system can be caused by small perturbations of either the system matrix or the right hand side vector, see (2.5.19) and the message of Thm. 2.5.24, (2.5.27). In this case, a stable algorithm can obviously produce a grossly “wrong” solution, as was already explained after (2.5.27).

Hence, lack of stability of Gaussian elimination will only become apparent for linear systems with well-conditioned system matrices.



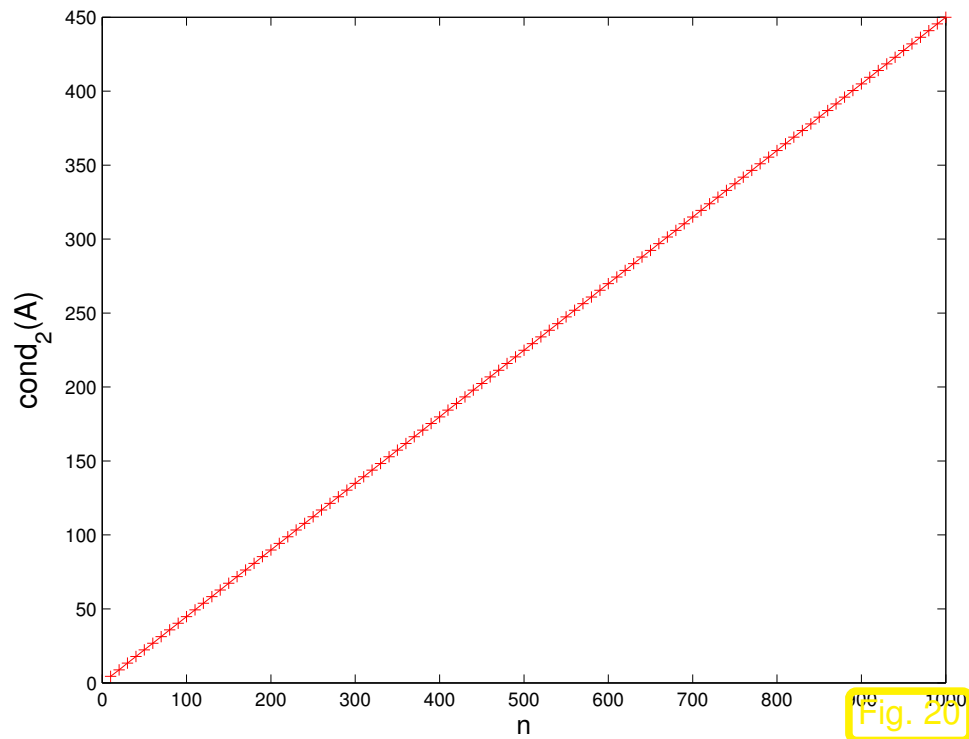


Fig. 20

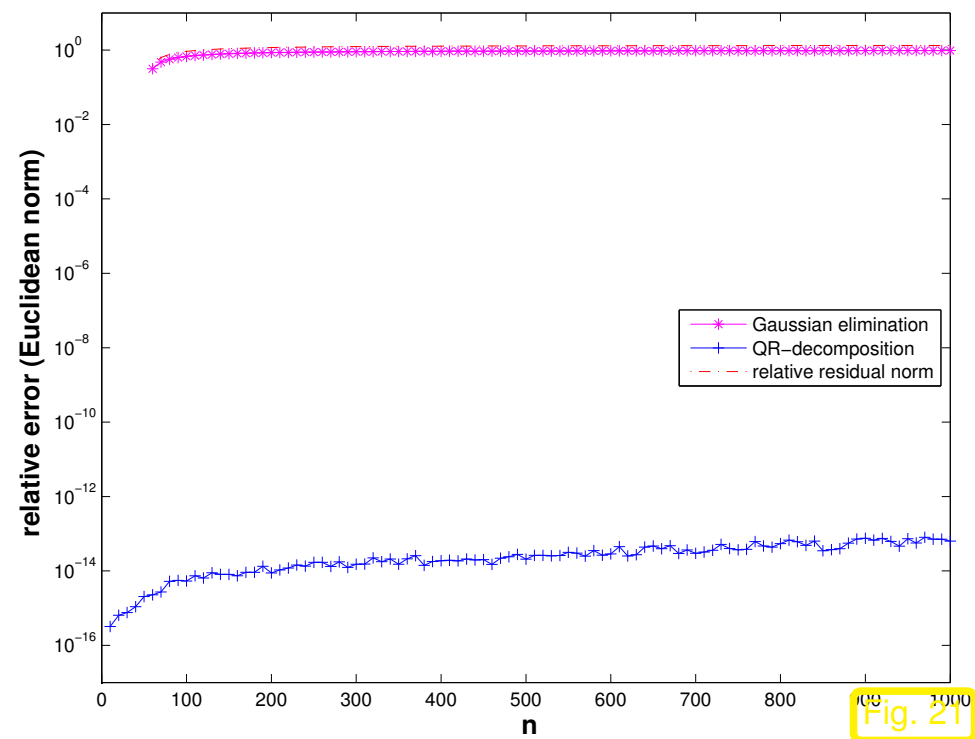


Fig. 21

These observations match Thm. 2.5.13, because in this case we encounter an exponential growth of  $\rho = \rho(n)$ , see Ex. 2.5.14.



## 2.5.5 Sensitivity of linear systems

Recall Thm. 2.5.24: for regular  $\mathbf{A} \in \mathbb{K}^{n,n}$ , small  $\Delta\mathbf{A}$ , generic vector/matrix norm  $\|\cdot\|$

$$\begin{aligned} \mathbf{Ax} = \mathbf{b} \\ (\mathbf{A} + \Delta\mathbf{A})\tilde{\mathbf{x}} = \mathbf{b} + \Delta\mathbf{b} \end{aligned} \Rightarrow \frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\text{cond}(\mathbf{A})}{1 - \text{cond}(\mathbf{A}) \|\Delta\mathbf{A}\| / \|\mathbf{A}\|} \left( \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \right). \quad (2.5.31)$$

►  $\text{cond}(\mathbf{A}) \gg 1$  ► small relative changes of data  $\mathbf{A}, \mathbf{b}$  may effect huge relative changes in solution.

**Sensitivity** of a problem (for given data) gauges impact of small perturbations of the data on the result.

►  $\text{cond}(\mathbf{A})$  indicates sensitivity of “LSE problem”  $(\mathbf{A}, \mathbf{b}) \mapsto \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$  (as “amplification factor” of relative perturbations in the data  $\mathbf{A}, \mathbf{b}$ ).

Terminology:

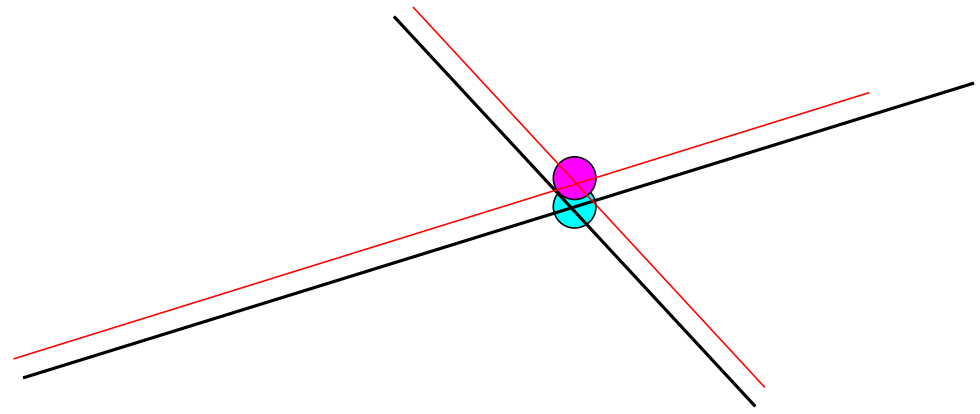
Small changes of data  $\Rightarrow$  small perturbations of result : well-conditioned problem

Small changes of data  $\Rightarrow$  large perturbations of result : ill-conditioned problem

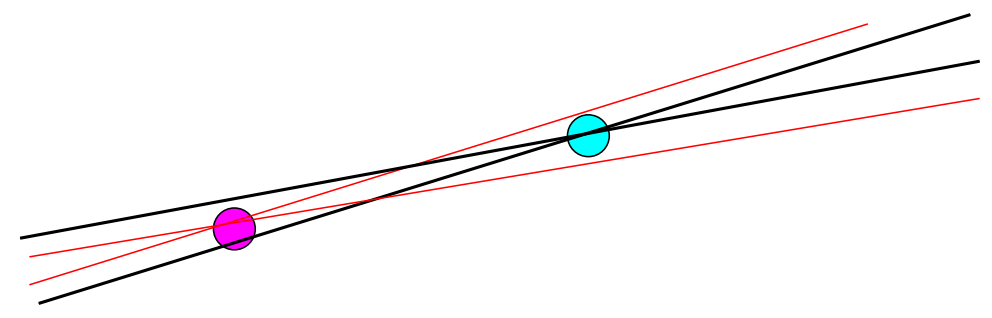
Note: sensitivity gauge depends on the chosen norm !

Example 2.5.32 (Intersection of lines in 2D).

In distance metric:



nearly orthogonal intersection: well-conditioned



glancing intersection: ill-conditioned

Hessian normal form of line # $i$ ,  $i = 1, 2$ :

$$L_i = \{ \mathbf{x} \in \mathbb{R}^2 : \mathbf{x}^T \mathbf{n}_i = d_i \}, \quad \mathbf{n}_i \in \mathbb{R}^2, d_i \in \mathbb{R} .$$

$\blacktriangleright$  intersection:  $\underbrace{\begin{pmatrix} \mathbf{n}_1^T \\ \mathbf{n}_2^T \end{pmatrix}}_{=: \mathbf{A}} \mathbf{x} = \underbrace{\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}}_{=: \mathbf{b}},$

$\mathbf{n}_i \hat{=}$  (unit) direction vectors,  $d_i \hat{=}$  distance to origin.

Code 2.5.34: condition numbers of  $2 \times 2$  matrices

```

1 r = [];
2 for phi=pi/200:pi/100:pi/2
3     A = [1, cos(phi);
4         0, sin(phi)];
5     r = [r; phi,
6         cond(A), cond(A, 'inf')];
7 end
8 plot(r(:,1), r(:,2), 'r-',
9     r(:,1), r(:,3), 'b--');
10 xlabel('{\bf angle of n_1,
11     n_2}', 'fontsize', 14);
12 ylabel('{\bf condition
13     numbers}', 'fontsize', 14);
14 legend('2-norm', 'max-norm');
15 print -depsc2
16     '../PICTURES/linesec.eps';

```

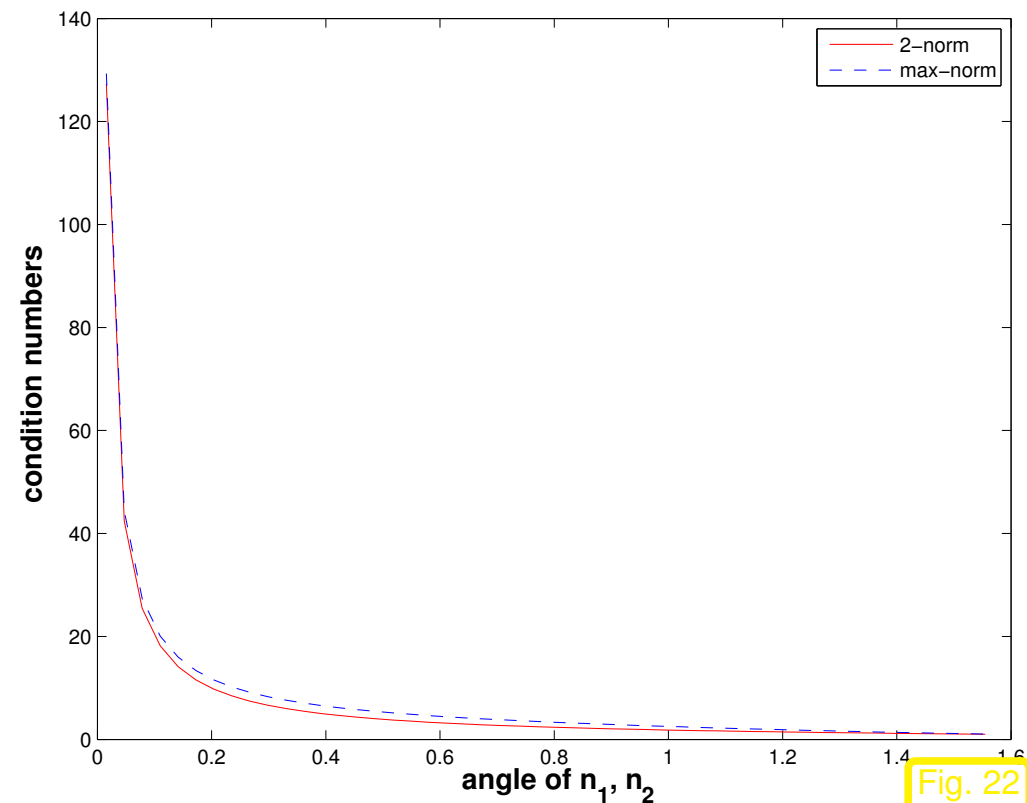


Fig. 22

Heuristics:

$$\text{cond}(\mathbf{A}) \gg 1 \iff \text{columns/rows of } \mathbf{A} \text{ "almost linearly dependent"}$$

# 2.6 Sparse Matrices

A classification of matrices:

Dense matrices (*ger.:* vollbesetzt)



sparse matrices (*ger.:* dünnbesetzt)

**Notion 2.6.1** (Sparse matrix).  $\mathbf{A} \in \mathbb{K}^{m,n}$ ,  $m, n \in \mathbb{N}$ , is *sparse*, if

$$\text{nnz}(\mathbf{A}) := \#\{(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\} : a_{ij} \neq 0\} \ll mn .$$

Sloppy parlance: matrix *sparse*  $:\Leftrightarrow$  “almost all” entries  $= 0$  / “only a few percent of” entries  $\neq 0$

A more rigorous “mathematical” definition:

**Definition 2.6.2** (Sparse matrices).

Given a strictly increasing sequences  $m : \mathbb{N} \mapsto \mathbb{N}$ ,  $n : \mathbb{N} \mapsto \mathbb{N}$ , a family  $(\mathbf{A}^{(l)})_{l \in \mathbb{N}}$  of matrices with  $\mathbf{A}^{(l)} \in \mathbb{K}^{m_l, n_l}$  is *sparse* (opposite: dense), if

$$\text{nnz}(\mathbf{A}^{(l)}) := \#\{(i, j) \in \{1, \dots, m_i\} \times \{1, \dots, n_i\} : a_{ij}^{(l)} \neq 0\} = O(n_i + m_i) \quad \text{for } i \rightarrow \infty .$$

Simple example: families of diagonal matrices ( $\rightarrow$  Def. 2.2.3)

*Example 2.6.3* (Nodal analysis of (linear) electric circuit ). [51, Sect. 4.7.1],

Now we study an very important application of numerical simulation, where (large, sparse) linear systems of equations play a central role: **Numerical circuit analysis**. We begin with *linear circuits* in the *frequency domain*, which are directly modelled by complex linear systems of equations. Later we tackle circuits with non-linear elements, see Ex. 4.0.1, and, finally, will learn about numerical methods for computing the transient (time-dependent) behavior of circuits, see Ex. 12.1.8.

Modeling of simple linear circuits takes only elementary physical laws as covered in any introductory course of physics (or even in secondary school physics). There is no sophisticated physics or mathematics involved.

**Node** (ger.: Knoten)  $\hat{=}$  junction of wires

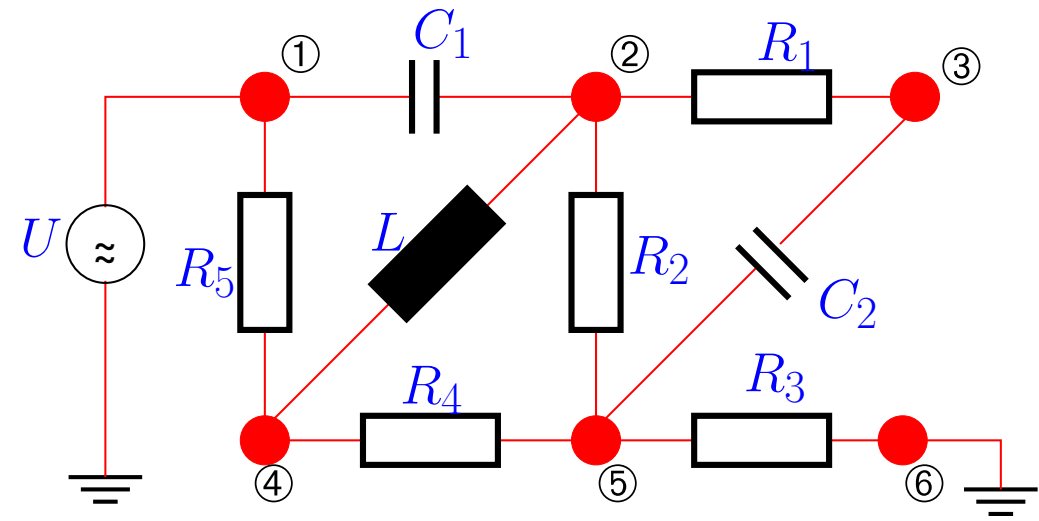
☞ number nodes  $1, \dots, n$

$I_{kj}$ : current from node  $k \rightarrow$  node  $j$ ,  $I_{kj} = -I_{jk}$

**Kirchhoff current law** (KCL) : sum of node currents

= 0:

$$\forall k \in \{1, \dots, n\}: \sum_{j=1}^n I_{kj} = 0. \quad (2.6.4)$$



Unknowns: **nodal potentials**  $U_k$ ,  $k = 1, \dots, n$ .

(some may be known: grounded nodes, voltage sources)

Constitutive relations for circuit elements: (in *frequency domain* with angular frequency  $\omega > 0$ ):

- Ohmic resistor:  $I = \frac{U}{R}$ ,  $[R] = 1\text{VA}^{-1}$
- capacitor:  $I = \omega C U$ , capacitance  $[C] = 1\text{AsV}^{-1}$
- coil/inductor :  $I = \frac{U}{\omega L}$ , inductance  $[L] = 1\text{VsA}^{-1}$

$$\blacktriangleright I_{kj} = \begin{cases} R^{-1}(U_k - U_j), \\ \omega C(U_k - U_j), \\ -\omega^{-1}L^{-1}(U_k - U_j). \end{cases}$$

notation:  $\mathbf{i} \hat{=} \text{imaginary unit}$  “ $\mathbf{i} := \sqrt{-1}$ ”,  $\mathbf{i} = \exp(\mathbf{i}\pi/2)$

Here we face the special case of a **linear circuit**: all relationships between branch currents and voltages are of the form

$$I_{kj} = \alpha_{kj}(U_k - U_j) \quad \text{with} \quad \alpha_{kj} \in \mathbb{C} .$$

The concrete value of  $\alpha_{kj}$  is determined by the circuit element connecting node  $k$  and node  $j$ .

These constitutive relations are derived by assuming a harmonic time-dependence of all quantities, which is termed circuit analysis in the **frequency domain**.

$$\text{voltage: } u(t) = \text{Re}\{U \exp(\mathbf{i}\omega t)\} \quad , \quad \text{current: } i(t) = \text{Re}\{I \exp(\mathbf{i}\omega t)\} . \quad (2.6.5)$$

Here  $U, I \in \mathbb{C}$  are called complex amplitudes. This implies for temporal derivatives (denoted by a dot):

$$\frac{du}{dt}(t) = \text{Re}\{\mathbf{i}\omega U \exp(\mathbf{i}\omega t)\} \quad , \quad \frac{di}{dt}(t) = \text{Re}\{\mathbf{i}\omega I \exp(\mathbf{i}\omega t)\} . \quad (2.6.6)$$



For a capacitor the total charge is proportional to the applied voltage:

$$q(t) = Cu(t) \quad \begin{array}{l} i(t) = \dot{q}(t) \\ \Rightarrow \end{array} \quad i(t) = C\dot{u}(t) .$$

For a coil the voltage is proportional to the rate of change of current:  $u(t) = Li(t)$ . Combined with (2.6.5) and (2.6.6) this leads to the above constitutive relations.

Constitutive relations + (2.6.4)  $\blacktriangleright$  linear system of equations:

$$\textcircled{2} : \quad \boldsymbol{\omega}C_1(U_2 - U_1) + R_1^{-1}(U_2 - U_3) - \boldsymbol{\omega}^{-1}L^{-1}(U_2 - U_4) + R_2^{-1}(U_2 - U_5) = 0 ,$$

$$\textcircled{3} : \quad R_1^{-1}(U_3 - U_2) + \boldsymbol{\omega}C_2(U_3 - U_5) = 0 ,$$

$$\textcircled{4} : \quad R_5^{-1}(U_4 - U_1) - \boldsymbol{\omega}^{-1}L^{-1}(U_4 - U_2) + R_4^{-1}(U_4 - U_5) = 0 ,$$

$$\textcircled{5} : \quad R_2^{-1}(U_5 - U_2) + \boldsymbol{\omega}C_2(U_5 - U_3) + R_4^{-1}(U_5 - U_4) + R_3^{-1}(U_5 - U_6) = 0 ,$$

$$U_1 = U \quad , \quad U_6 = 0 .$$

$$\begin{pmatrix} \boldsymbol{\omega}C_1 + \frac{1}{R_1} - \frac{i}{\boldsymbol{\omega}L} + \frac{1}{R_2} & -\frac{1}{R_1} & \frac{i}{\boldsymbol{\omega}L} & -\frac{1}{R_2} \\ -\frac{1}{R_1} & \frac{1}{R_1} + \boldsymbol{\omega}C_2 & 0 & -\boldsymbol{\omega}C_2 \\ \frac{i}{\boldsymbol{\omega}L} & 0 & \frac{1}{R_5} - \frac{i}{\boldsymbol{\omega}L} + \frac{1}{R_4} & -\frac{1}{R_4} \\ -\frac{1}{R_2} & -\boldsymbol{\omega}C_2 & -\frac{1}{R_4} & \frac{1}{R_2} + \boldsymbol{\omega}C_2 + \frac{1}{R_4} + R_3^{-1} \end{pmatrix} \begin{pmatrix} U_2 \\ U_3 \\ U_4 \\ U_5 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\omega}C_1 U \\ 0 \\ \frac{1}{R_5} U \\ 0 \end{pmatrix}$$

This is a linear system of equations with *complex* coefficients:  $\mathbf{A} \in \mathbb{C}^{4,4}$ ,  $\mathbf{b} \in \mathbb{C}^4$ . For the algorithms to be discussed below this does not matter, because they work alike for real and complex numbers.



*Example 2.6.7* (Sparse LSE in circuit modelling).

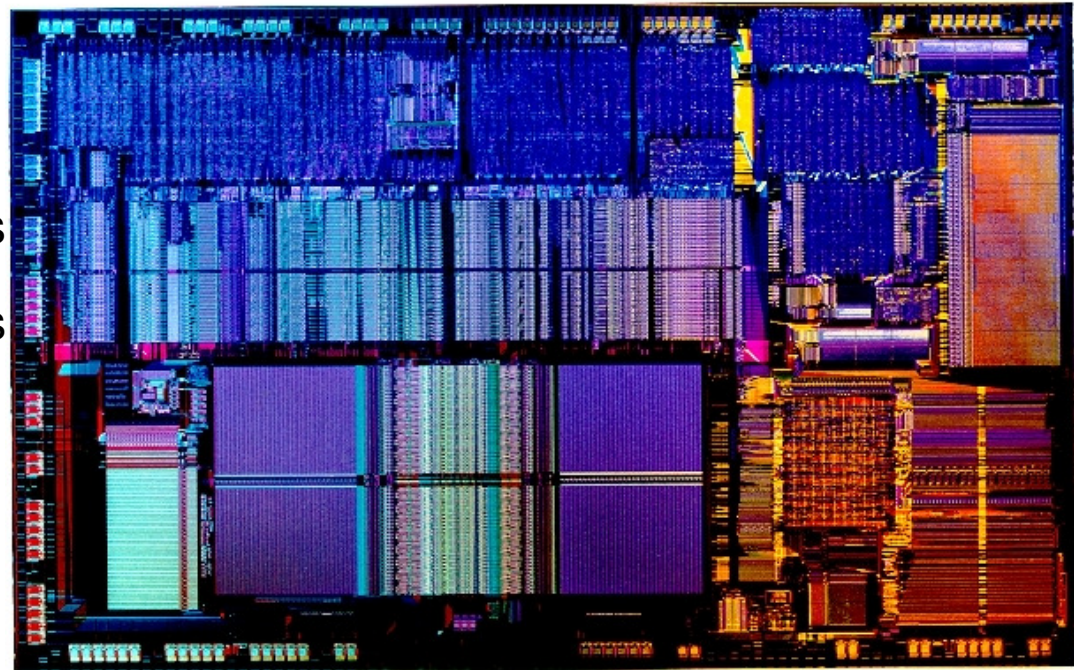
Modern electric circuits (VLSI chips):  
 $10^5 - 10^7$  circuit elements

- Each element is connected to only *a few* nodes
- Each node is connected to only *a few* elements

[In the case of a linear circuit]



nodal analysis ➤ **sparse** circuit matrix



Another important context in which sparse matrices usually arise:

- discretization of boundary value problems for partial differential equations (→ 4th semester course “Numerical treatment of PDEs”)

## 2.6.1 Sparse matrix storage formats

Special **sparse matrix storage formats** store *only* non-zero entries:

(➤ usually  $O(n + m)$  storage required for sparse  $n \times m$ -matrix)

- Compressed Row Storage (CRS)
- Compressed Column Storage (CCS) → used by MATLAB
- Block Compressed Row Storage (BCRS)
- Compressed Diagonal Storage (CDS)
- Jagged Diagonal Storage (JDS)
- Skyline Storage (SKS)

mandatory for large sparse matrices.

Example 2.6.8 (Compressed row-storage (CRS) format).

Data for matrix  $\mathbf{A} = (a_{ij}) \in \mathbb{K}^{n,n}$  kept in three arrays

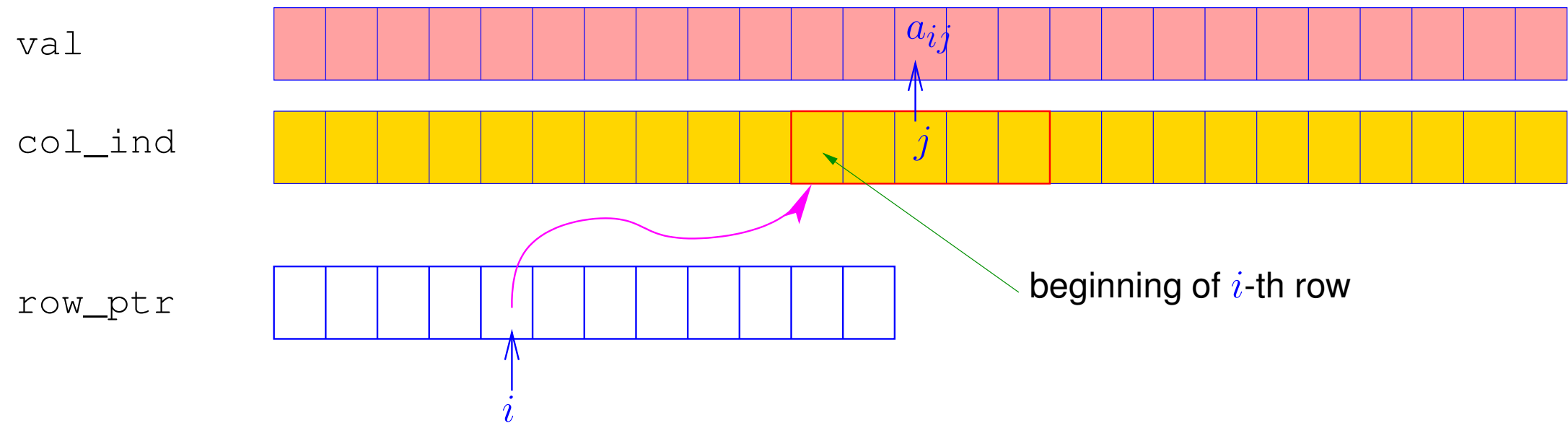
```

double * val           size nnz(A) := #{(i, j) ∈ {1, ..., n}^2, aij ≠ 0}
unsigned int * col_ind size nnz(A)
unsigned int * row_ptr  size n + 1 & row_ptr[n + 1] = nnz(A) + 1
    
```

$\text{nnz}(\mathbf{A}) \hat{=}$  (number of nonzeros) of  $\mathbf{A}$

Access to matrix entry  $a_{ij} \neq 0, 1 \leq i, j \leq n$ :

$$\text{val}[k] = a_{ij} \Leftrightarrow \begin{cases} \text{col\_ind}[k] = j, \\ \text{row\_ptr}[i] \leq k < \text{row\_ptr}[i + 1], \end{cases} \quad 1 \leq k \leq \text{nnz}(\mathbf{A}).$$



$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

val	10	-2	3	9	3	7	8	7	3 ... 9	13	4	2	-1
col_ind	1	5	1	2	6	2	3	4	1 ... 5	6	2	5	6
row_ptr	1	3	6	9	13	17	20						

Option: diagonal CRS format (matrix diagonal stored in separate array)



## 2.6.2 Sparse matrices in MATLAB



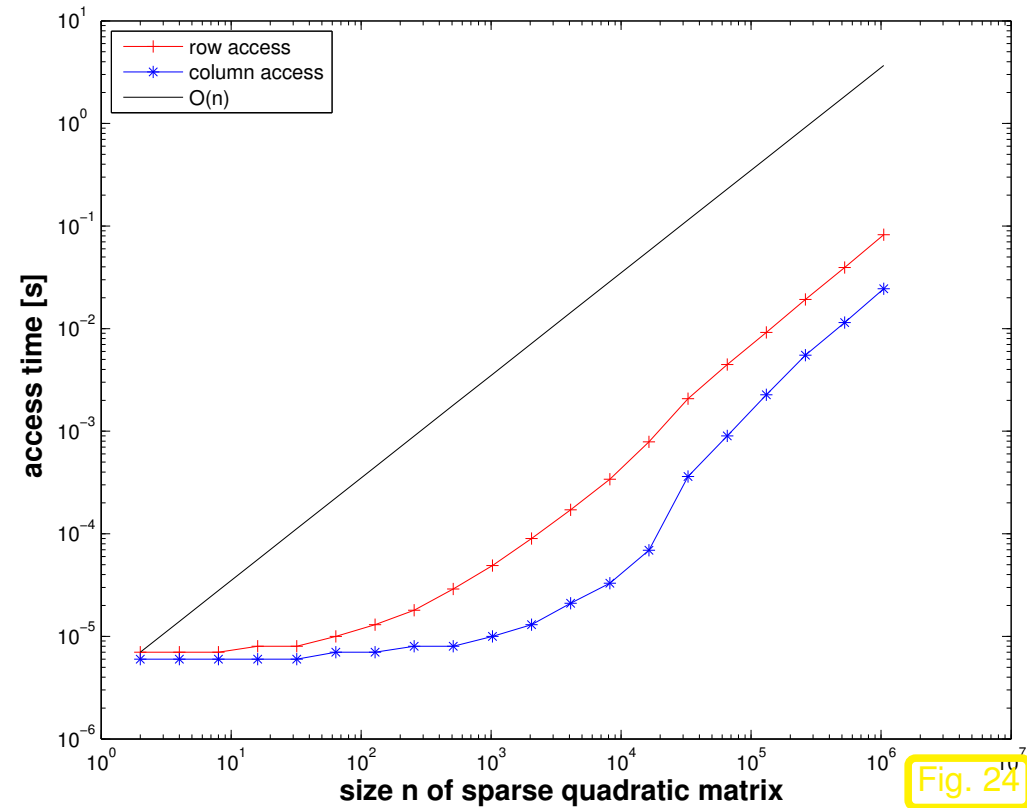
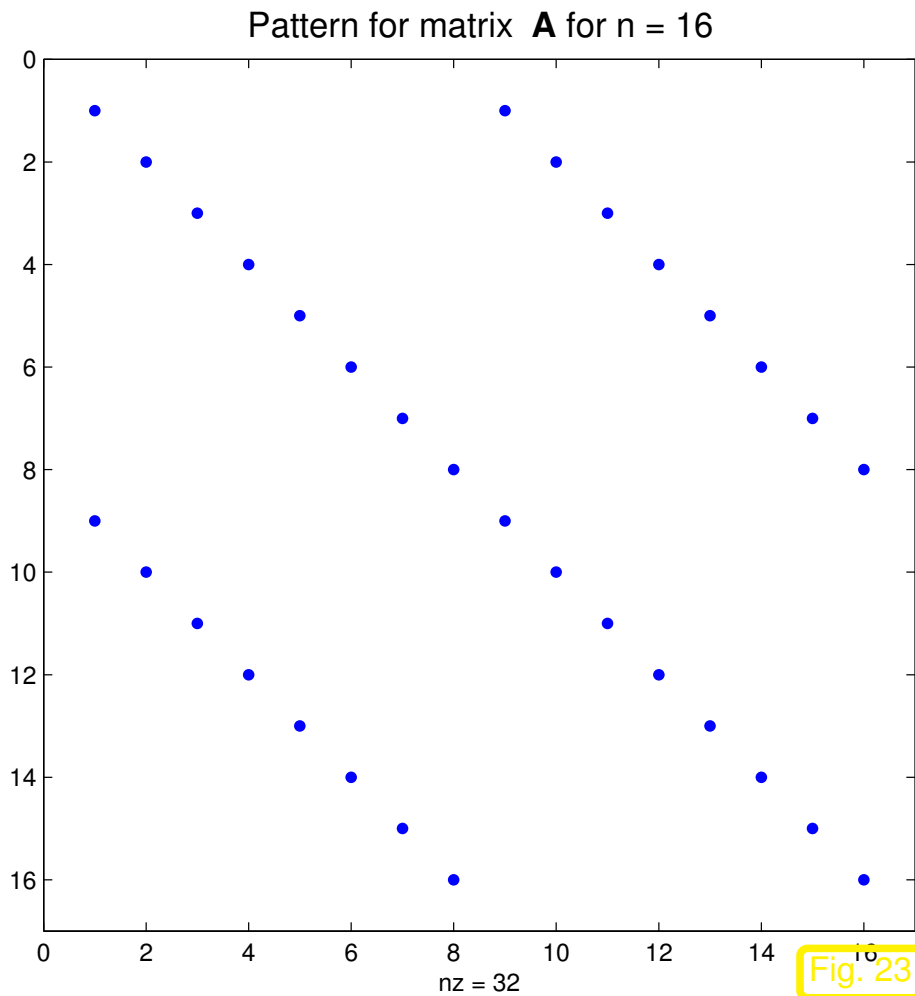
## Supplementary and further reading:

A detailed discussion of sparse matrix formats and how to work with them efficiently is given in [22].

An interesting related article on MATLAB-central can be found here.

```
Initialization: A = sparse(m,n); A = spalloc(m,n,nnz)
               A = sparse(i,j,s,m,n);
               A = spdiags(B,d,m,n); A = speye(n); A = spones(S);
```

*Example 2.6.9* (Accessing rows and columns of sparse matrices).



Code 2.6.10: timing access to rows/columns of a sparse matrix

```

1 figure; spy(spdiags(repmat([-1 2 5],16,1),[-8,0,8],16,16)); %
2 title('Pattern for matrix {\bf A} for n = 16','fontsize',14);
3 print -depsc2 '../PICTURES/spdiagsmatspy.eps';
4
5 t = [];
6 for i=1:20
7     n = 2^i; m = n/2;

```

```
8  A = spdiags(repmat([-1 2 5],n,1),[-n/2,0,n/2],n,n); %
9
10 t1 = inf; for j=1:5, tic; v = A(m,:)+j; t1 = min(t1,toc); end
11 t2 = inf; for j=1:5, tic; v = A(:,m)+j; t2 = min(t2,toc); end
12 t = [t; size(A,1), nnz(A), t1, t2 ];
13 end
14
15 figure;
16 loglog(t(:,1),t(:,3),'r+-', t(:,1),t(:,4),'b*-',...
17         t(:,1),t(1,3)*t(:,1)/t(1,1),'k-');
18 xlabel('\bf size n of sparse quadratic matrix','fontsize',14);
19 ylabel('\bf access time [s]','fontsize',14);
20 legend('row access','column
        access','O(n)','location','northwest');
21
22 print -depsc2 '../PICTURES/sparseaccess.eps';
```

MATLAB uses compressed column storage (CCS), which entails  $O(n)$  searches for index  $j$  in the index array when accessing all elements of a matrix row. Conversely, access to a column does not involve any search operations.



[ Acknowledgment: this observation was made by Andreas Növer, 8.10.2009 ]

Note the use of the MATLABcommand `repmat` in lines 1 and 8 of the above code. It can be used to build structured matrices. Consult the MATLABdocumentation for details.



*Example 2.6.11* (Efficient Initialization of sparse matrices in MATLAB).

Code 2.6.12: Initialization of sparse matrices: version I

```
1 A1 = sparse (n, n) ;  
2 for i=1:n  
3     for j=1:n  
4         if (abs (i-j) == 1) , A1 (i, j) = A1 (i, j) + 1; end ;  
5         if (abs (i-j) == round (n/3)) , A1 (i, j) = A1 (i, j) -1; end ;  
6 end ; end
```

Code 2.6.13: Initialization of sparse matrices: version II

```
1 dat = [] ;  
2 for i=1:n  
3     for j=1:n
```

```

4     if (abs(i-j) == 1), dat = [dat; i, j, 1.0]; end;
5     if (abs(i-j) == round(n/3)), dat = [dat; i, j, -1.0];
6 end; end; end;
7 A2 = sparse(dat(:,1), dat(:,2), dat(:,3), n, n);

```

## Code 2.6.14: Initialization of sparse matrices: version III

```

1 dat = zeros(6*n, 3); k = 0;
2 for i=1:n
3     for j=1:n
4         if (abs(i-j) == 1), k=k+1; dat(k, :) = [i, j, 1.0];
5         end;
6         if (abs(i-j) == round(n/3))
7             k=k+1; dat(k, :) = [i, j, -1.0];
8         end;
9     end; end;
10 A3 = sparse(dat(1:k, 1), dat(1:k, 2), dat(1:k, 3), n, n);

```

## Code 2.6.15: Initialization of sparse matrices: driver script

```

1 % Driver routine for initialization of sparse matrices
2 K = 3; r = [];
3 for n=2.(8:14)
4     t1= 1000; for k=1:K, fprintf('sparse1, %d, %d\n', n, k); tic;
5     sparse1; t1 = min(t1, toc); end
6     t2= 1000; for k=1:K, fprintf('sparse2, %d, %d\n', n, k); tic;
7     sparse2; t2 = min(t2, toc); end

```

```
6     t3= 1000; for k=1:K, fprintf ('sparse3, %d, %d\n',n,k); tic;  
       sparse3; t3 = min(t3,toc); end  
7     r = [r; n, t1 , t2, t3];  
8 end  
9  
10 loglog (r(:,1),r(:,2), 'r*',r(:,1),r(:,3), 'm+',r(:,1),r(:,4), 'b^');  
11 xlabel ('\bf matrix size n', 'fontsize',14);  
12 ylabel ('\bf time [s]', 'fontsize',14);  
13 legend ('Initialization I', 'Initialization II', 'Initialization  
      III', ...  
14         'location', 'northwest');  
15 print -depsc2 './PICTURES/sparseinit.eps';
```

## Timings:

- Linux lions 2.6.16.27-0.9-smp #1 SMP Tue Feb 13 09:35:18 UTC 2007 i686 i686 i386 GNU/Linux
- CPU: Genuine Intel(R) CPU T2500 2.00GHz
- MATLAB 7.4.0.336 (R2007a)

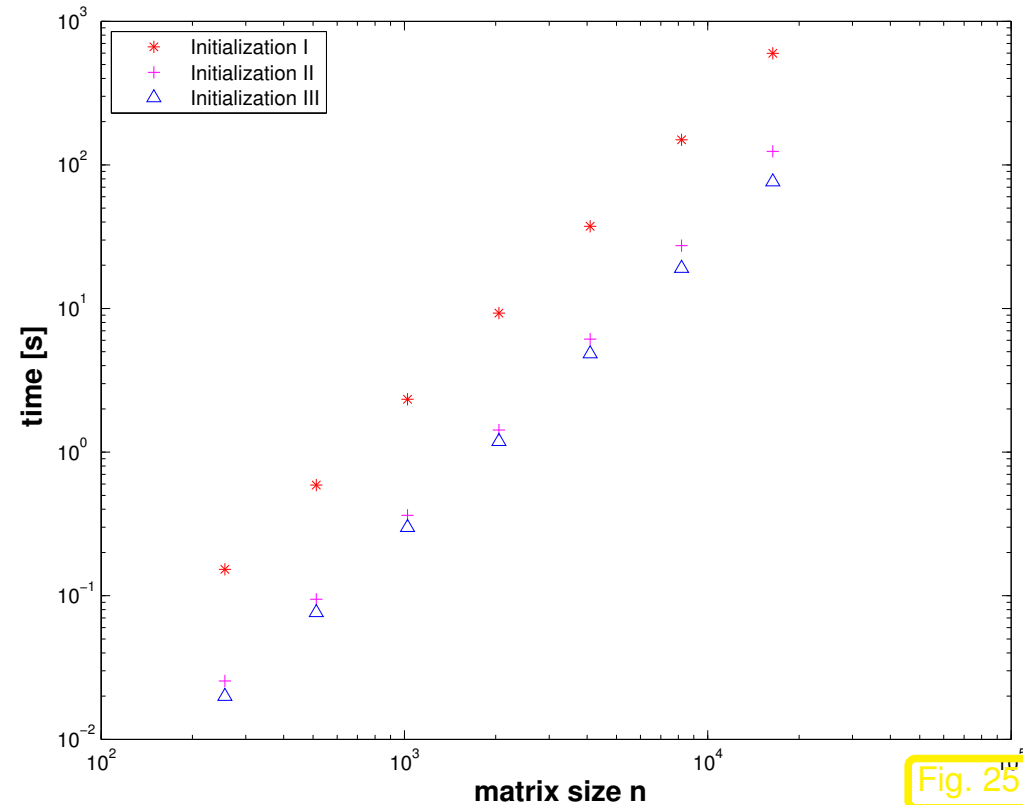


Fig. 25

- It is grossly inefficient to initialize a matrix in CCS format ( $\rightarrow$  Ex. 2.6.8) by setting individual entries one after another, because this usually entails moving large chunks of memory to create space for new non-zero entries.

Instead calls like

```
sparse(dat(1:k,1), dat(1:k,2), dat(1:k,3), n, n);
```

where

$$\text{dat}(1:k, 1) = i \quad \text{and} \quad \text{dat}(1:k, 2) = j \quad \Rightarrow \quad a_{ij} = \text{dat}(1:k, 3),$$

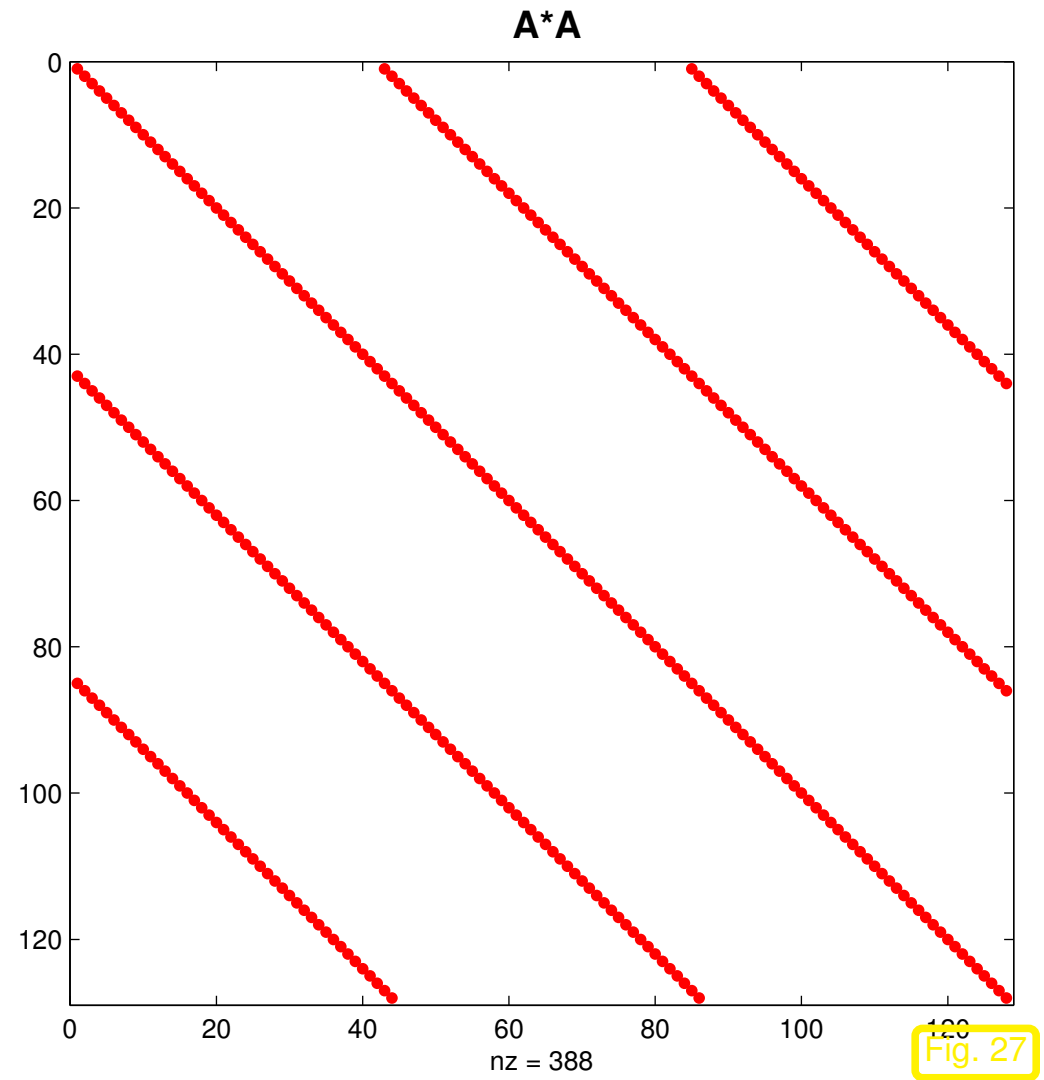
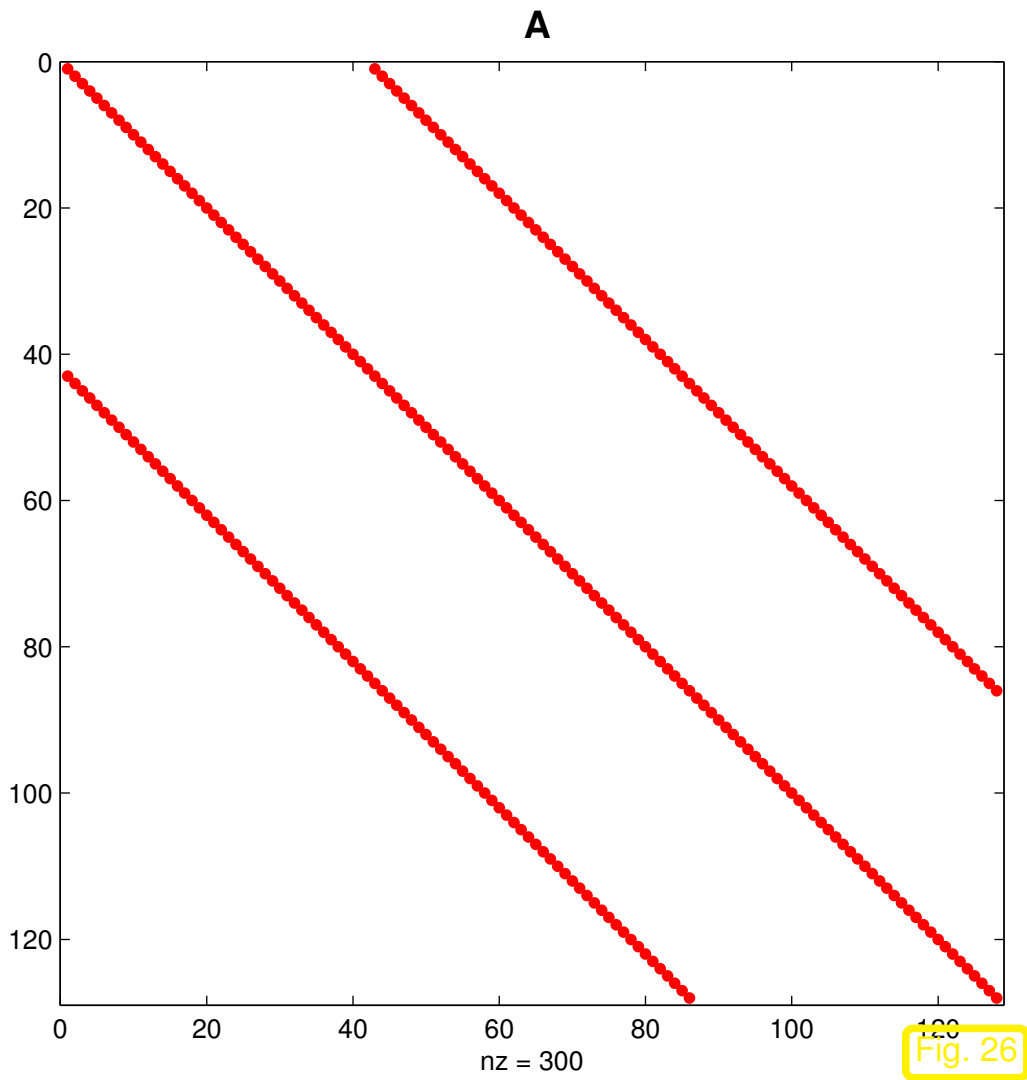
allow MATLAB to allocate memory and initialize the arrays in one sweep.



*Example 2.6.16* (Multiplication of sparse matrices).

Sparse matrix  $\mathbf{A} \in \mathbb{R}^{n,n}$  initialized by

```
A = spdiags([ (1:n)', ones(n,1), (n:-1:1)' ], ...  
            [-floor(n/3), 0, floor(n/3)], n, n);
```



➤  $A^2$  is still a sparse matrix ( $\rightarrow$  Notion 2.6.1)

runtimes for matrix multiplication  $A * A$  in MATLAB  
(`tic/toc` timing)  $\triangleright$

(same platform as in Ex. 2.6.11)

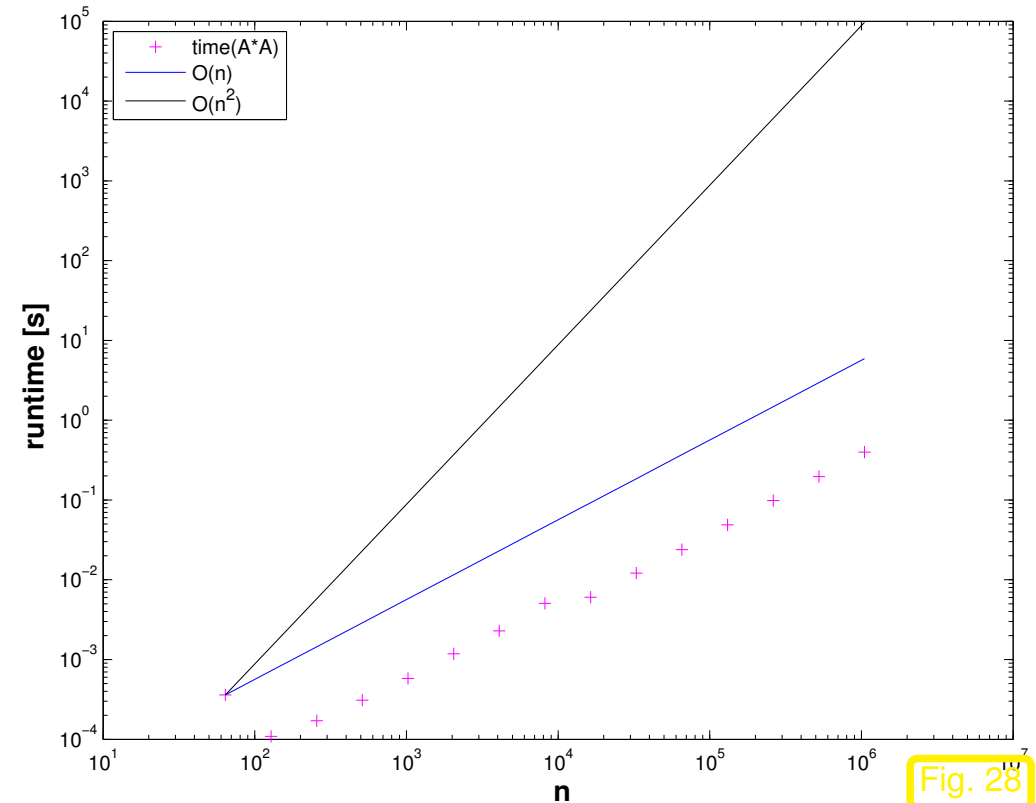


Fig. 28

$\blacktriangleright$   $O(n)$  asymptotic complexity: “optimal” implementation !  $\diamond$

Remark 2.6.17 (Silly MATLAB).

A strange behavior of MATLAB from version R2010a:

```

1 % MATLAB script demonstrating the awkward effects of treating entries of
2 % sparse matrices as sparse matrices themselves.
3 A = spdiags(ones(100,3), -1:1, 100, 100);
4 b = A(1,1),

```

```
5 c = full(b),  
6 whos('b','c');  
7 sum=0; tic; for i=1:1e6, sum = sum + b; end, toc  
8 sum=0; tic; for i=1:1e6, sum = sum + c; end, toc
```

Output ( MATLAB-version 7.12.0.635 (R2011a), MacOS X 10.6, Intel Core i7):

```
» sparseentry
```

```
b =  
(1,1) 1
```

```
c =  
1
```

```
Name Size Bytes Class Attributes
```

```
b 1x1 32 double sparse
```

```
c 1x1 8 double
```



```
Elapsed time is 2.962332 seconds.  
Elapsed time is 0.514712 seconds.
```

When extracting a single entry from a sparse matrix, this entry will be stored in sparse format though it is a mere number! This will considerably slow down all operations on that entry.

**Change in Indexing for Sparse Matrix Input.** Now subscripted reference into a sparse matrix always returns a sparse matrix. In previous versions of MATLAB, using a double scalar to index into a sparse matrix resulted in full scalar output.



### 2.6.3 LU-factorization of sparse matrices

In Sect. 2.6.1 we have seen, how sparse matrices can be stored requiring  $O(\text{nnz}(\mathbf{A}))$  memory.

In Ex. 2.6.16 we found that (sometimes) matrix multiplication of sparse matrices can also be carried out with optimal complexity, that is, with computational effort proportional to the total number of non-zero entries of all matrices involved.

Does this carry over to the solution of linear systems of equations with sparse system matrices?

*Example 2.6.18 (LU-factorization of sparse matrices).*

$$\mathbf{A} = \left( \begin{array}{cccc|cccc} 3 & -1 & & & -1 & & & \\ -1 & \ddots & \ddots & & & \ddots & & \\ & \ddots & \ddots & -1 & 3 & & & \\ -1 & & & & & & -1 & \\ \hline & & & & 3 & -1 & & \\ & & & & -1 & \ddots & \ddots & \\ & & & & & \ddots & \ddots & -1 \\ & & & & & & -1 & 3 \end{array} \right) \in \mathbb{R}^{n,n}, n \in \mathbb{N}$$

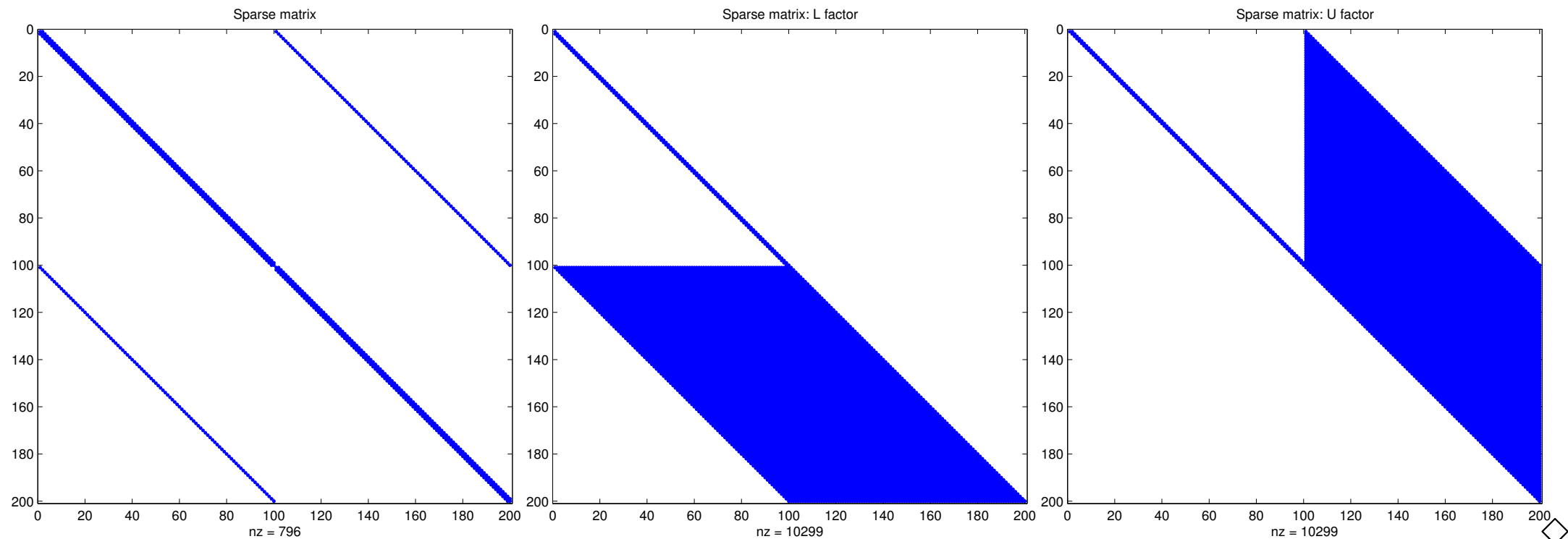
Code 2.6.19: LU-factorization of sparse matrix

```
1 % Demonstration of fill-in for LU-factorization of sparse matrices
2 n = 100;
3 A = [gallery('tridiag', n, -1, 3, -1), speye(n); speye(n) ,
      gallery('tridiag', n, -1, 3, -1)]; [L,U,P] = lu(A);
```

```

4 figure; spy (A); title ('Sparse matrix'); print -depsc2
   './PICTURES/sparseA.eps';
5 figure; spy (L); title ('Sparse matrix: L factor'); print -depsc2
   './PICTURES/sparseL.eps';
6 figure; spy (U); title ('Sparse matrix: U factor'); print -depsc2
   './PICTURES/sparseU.eps';

```



$A$  sparse  $\nrightarrow$   $LU$ -factors sparse

**Definition 2.6.20** (Fill-in).

Let  $\mathbf{A} = \mathbf{LU}$  be an  $LU$ -factorization ( $\rightarrow$  Sect. 2.2) of  $\mathbf{A} \in \mathbb{K}^{n,n}$ . If  $l_{ij} \neq 0$  or  $u_{ij} \neq 0$  though  $a_{ij} = 0$ , then we encounter **fill-in** at position  $(i, j)$ .

**Example 2.6.21** (Sparse  $LU$ -factors).

Ex. 2.6.18  $\triangleright$  massive fill-in can occur for sparse matrices

This example demonstrates that fill-in can be largely avoided, if the matrix has favorable structure. In this case a LSE with this particular system matrix  $\mathbf{A}$  can be solved efficiently, that is, with a computational effort  $O(\text{nnz}(\mathbf{A}))$  by Gaussian elimination.

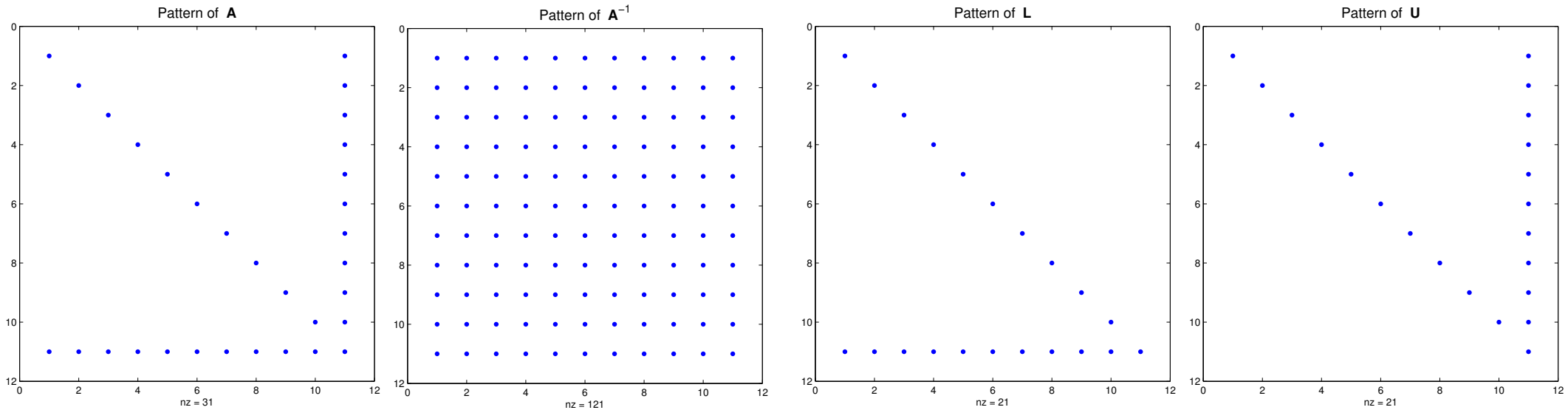
```

1 % Simple example for dense inverse despite sparse LU-factors
2 A = [diag(1:10), ones(10,1); ones(1,10), 2];
3 [L,U] = lu(A); spy(A); spy(L); spy(U); spy(inv(A));

```

$A$  is called an “arrow matrix”, see the pattern of non-zero entries below.

Recalling Rem. 2.2.14 it is easy to see that the LU-factors of  $A$  will be sparse and that their sparsity patterns will be as depicted below.



$L, U$  sparse  $\not\Rightarrow A^{-1}$  sparse !



Besides stability issues, see Ex. 2.5.17, this is another reason why using  $\mathbf{x} = \text{inv}(A) * \mathbf{y}$  instead of  $\mathbf{y} = A \setminus \mathbf{b}$  is usually a major blunder.



Example 2.6.22 (“arrow matrix”).

$$\mathbf{A} = \left( \begin{array}{c|cc} \alpha & & \mathbf{b}^\top \\ \hline & & \\ \mathbf{c} & & \mathbf{D} \end{array} \right), \quad \begin{array}{l} \alpha \in \mathbb{R}, \\ \mathbf{b}, \mathbf{c} \in \mathbb{R}^{n-1}, \\ \mathbf{D} \in \mathbb{R}^{n-1, n-1} \text{ regular diagonal matrix, } \rightarrow \text{Def. 2.2.3} \end{array} \quad (2.6.23)$$

R. Hiptmair  
rev 38286,  
October 7,  
2011

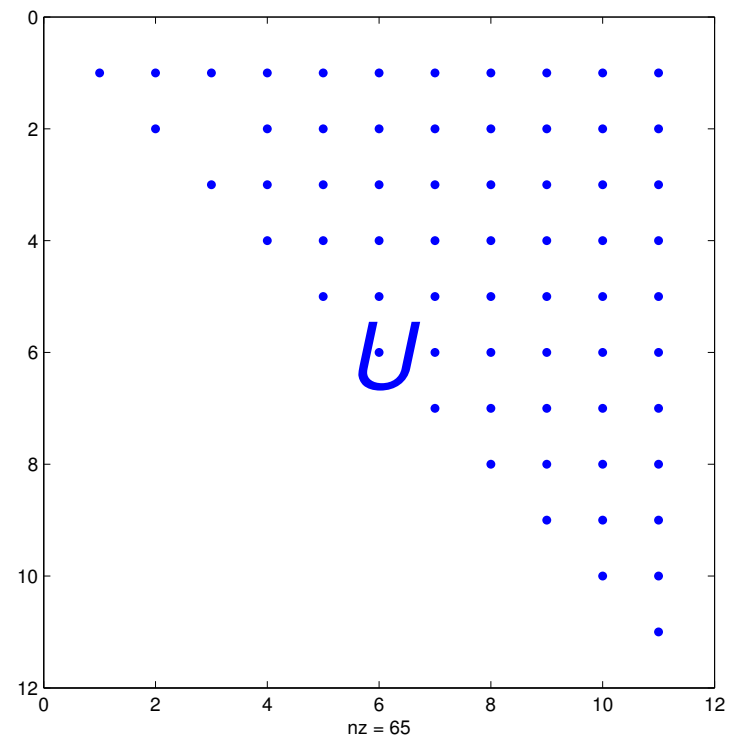
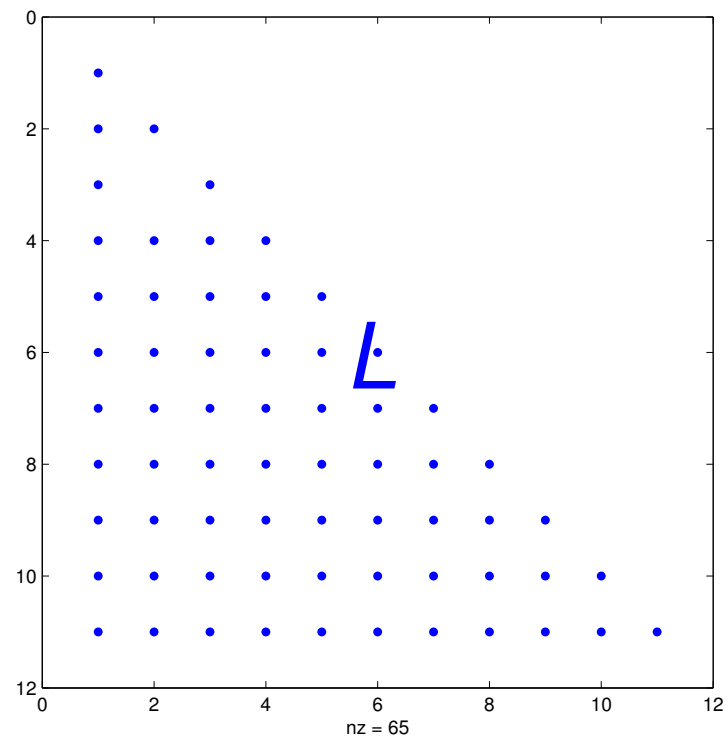
Run algorithm 2.3.8 (Gaussian elimination without pivoting):

- factor matrices with  $O(n^2)$  non-zero entries.
- computational costs:  $O(n^3)$

Code 2.6.24: LU-factorization of arrow matrix

```
1 n = 10; A = [ n+1, (n:-1:1) ;
               ones(n,1), eye(n,n) ];
2 [L,U,P] = lu(A); spy(L); spy(U);
```

Obvious fill-in ( $\rightarrow$  Def. 2.6.20)



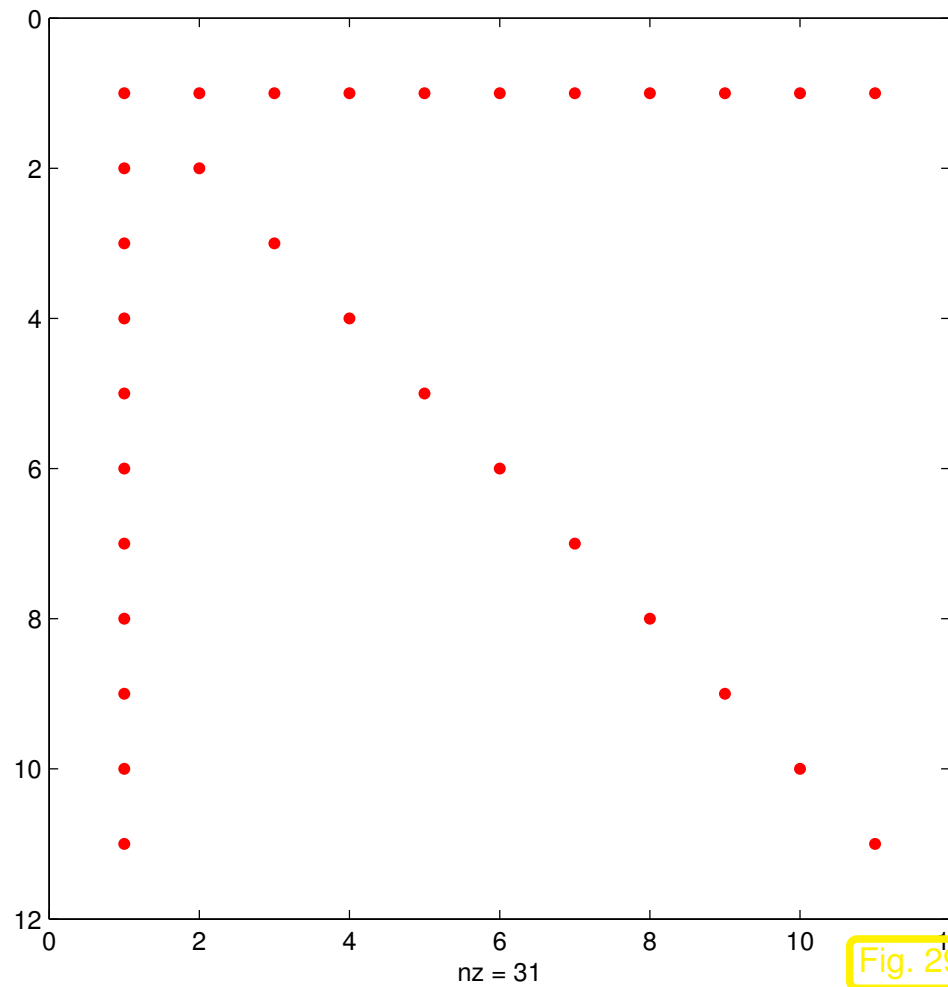


Fig. 29

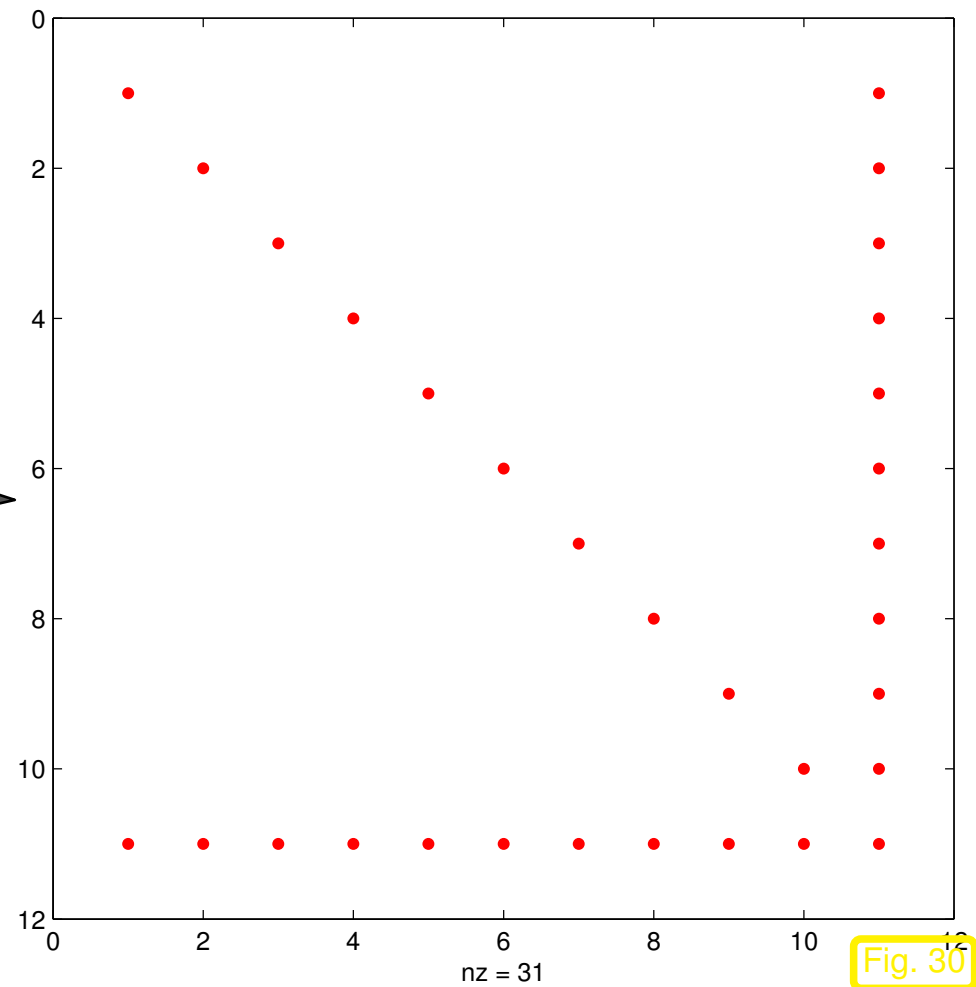


Fig. 30

Code 2.6.25: Permuting arrow matrix, see Figs. 29, 30

```

1 n = 10; A = [ n+1, (n:-1:1) ; ones(n,1), eye(n,n) ];
2 % Permutation matrix (→ Def. 2.3.12) encoding cyclic permutation
3 P = [ zeros(n,1), eye(n) ; 1, zeros(1,n) ];
4
5 figure('name','A');
6 spy(A,'r. '); print -depsc '../PICTURES/InvArrowSpy.eps';
7 figure('name','PAPT');

```



```
spy (P*A*P', 'r. '); print -depsc '../PICTURES/ArrowSpy.eps';
```

Perform cyclic permutation of rows/columns:

- 1st row/column  $\rightarrow n$ -th row/column
- $i$ -th row/column  $\rightarrow i - 1$ -th row/column,  
 $i = 2, \dots, n$

➤ LU-factorization (*without pivoting*) requires  $O(n)$  operations, see Ex. 2.6.21.

$$\mathbf{A} = \left( \begin{array}{c|c} \mathbf{D} & \mathbf{c} \\ \hline \mathbf{b}^\top & \alpha \end{array} \right) \quad (2.6.26)$$

After permuting rows of  $\mathbf{A}$  from (2.6.26), cf. (2.2.17):

$$\mathbf{L} = \left( \begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline \mathbf{b}^\top \mathbf{D}^{-1} & 1 \end{array} \right), \quad \mathbf{U} = \left( \begin{array}{c|c} \mathbf{D} & \mathbf{c} \\ \hline \mathbf{0} & \sigma \end{array} \right), \quad \sigma := \alpha - \mathbf{b}^\top \mathbf{D}^{-1} \mathbf{c}.$$

➤ No more fill-in, costs merely  $O(n)$  !

Solving LSE  $Ax = y$  with  $A$  from 2.6.23: two MATLAB codes

“naive” implementation via “\”:

Code 2.6.28: LSE with arrow matrix, implementation I

```
1 function x =
   sa1(alpha,b,c,d,y)
2 A = [alpha, b'; c, diag(d)];
3 x = A\y;
```

“structure aware” implementation:

Code 2.6.30: LSE with arrow matrix, implementation II

```
1 function x =
   sa2(alpha,b,c,d,y)
2 z = b./d;
3 xi = (y(1) -
   dot(z,y(2:end)))...
4       / (alpha-dot(z,c));
5 x = [xi; (y(2:end)-xi*c) ./d];
```

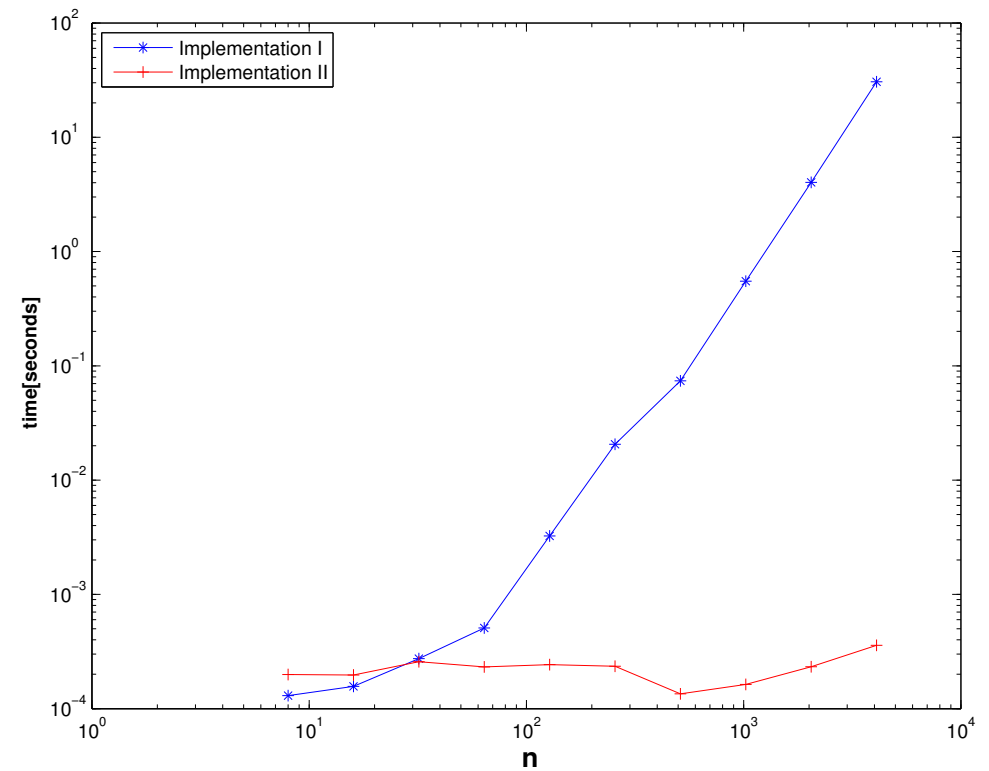
Code 2.6.26 is afflicted by a common plunder in MATLAB coding: inadvertently a dense matrix is created, where a sparse matrix should have been used. Here this happens through the inappropriate use of `diag`.

Measuring run times:

```

t = [];
for i=3:12
    n = 2^n; alpha = 2;
    b = ones(n,1); c = (1:n)';
    d = -ones(n,1); y = (-1).^(1:(n+1))';
    tic; x1 = sa1(alpha,b,c,d,y); t1 = toc;
    tic; x2 = sa2(alpha,b,c,d,y); t2 = toc;
    t = [t; n t1 t2];
end
loglog(t(:,1),t(:,2), ...
... 'b-*',t(:,1),t(:,3),'r-+');

```



Platform as in Ex. 2.6.11

MATLAB can do much better !

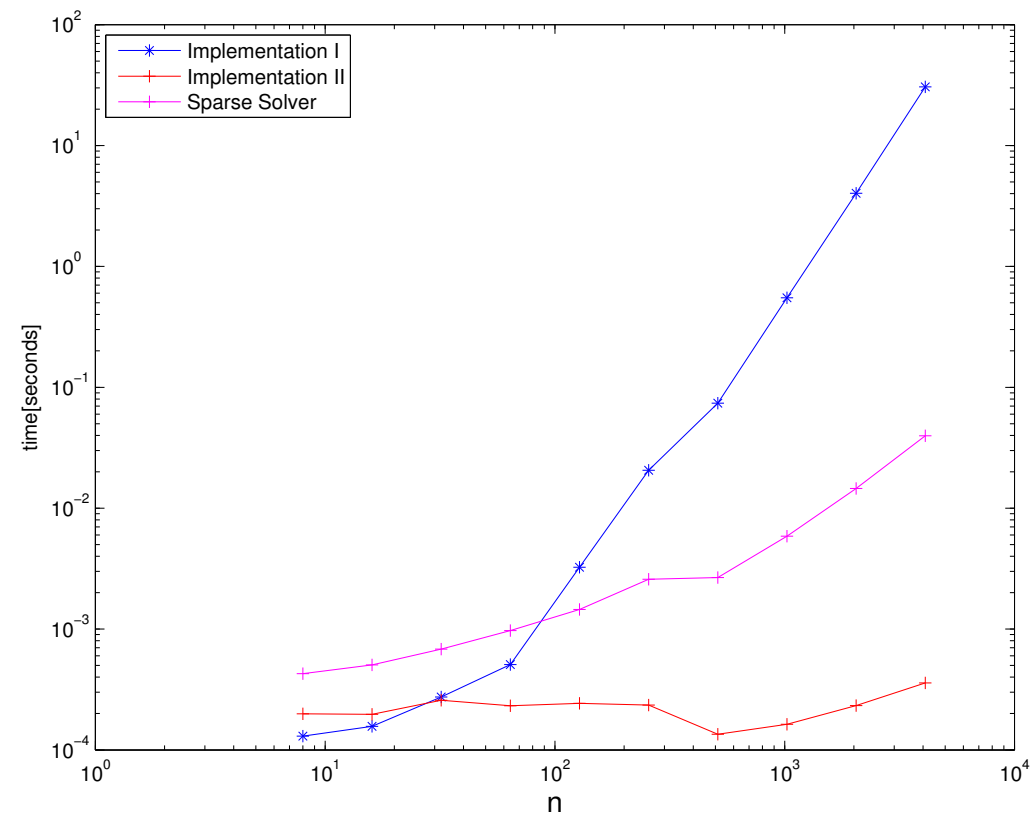
Use sparse matrix format:

Code 2.6.31: sparse solver for arrow matrix

```

1 function x =
   sa3(alpha,b,c,d,y)
2 n = length(d);
3 A = [alpha, b'; ...
4      c, spdiags(d,0,n,n)];
5 x = A\y;

```



Exploit structure of (sparse) linear systems of equations !

The vector based implementation of the solver of Code 2.6.26 can be vulnerable to roundoff errors, because it clearly amounts to Gaussian elimination without pivoting, *cf.* Sect. 2.3, Ex. 2.3.1.



Caution:

stability at risk

Example 2.6.32 (Pivoting destroys sparsity).

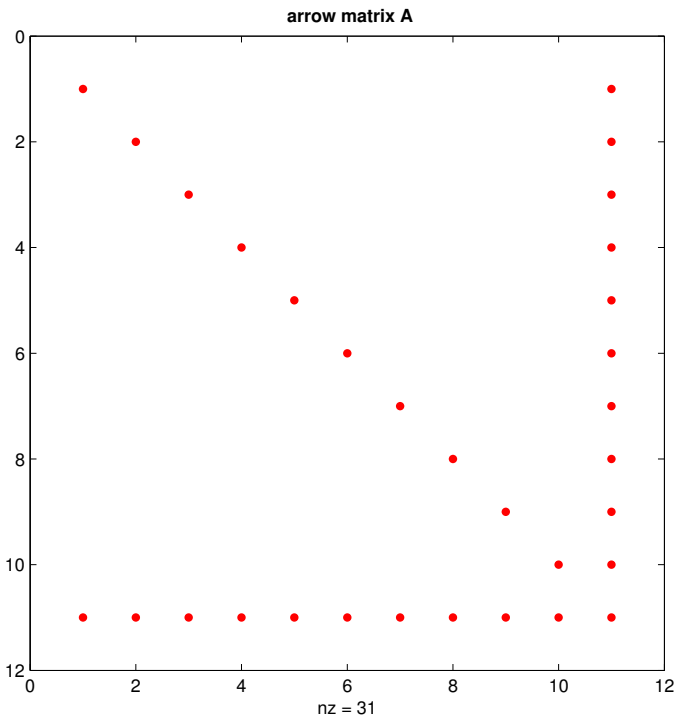
### Code 2.6.33: fill-in due to pivoting

```

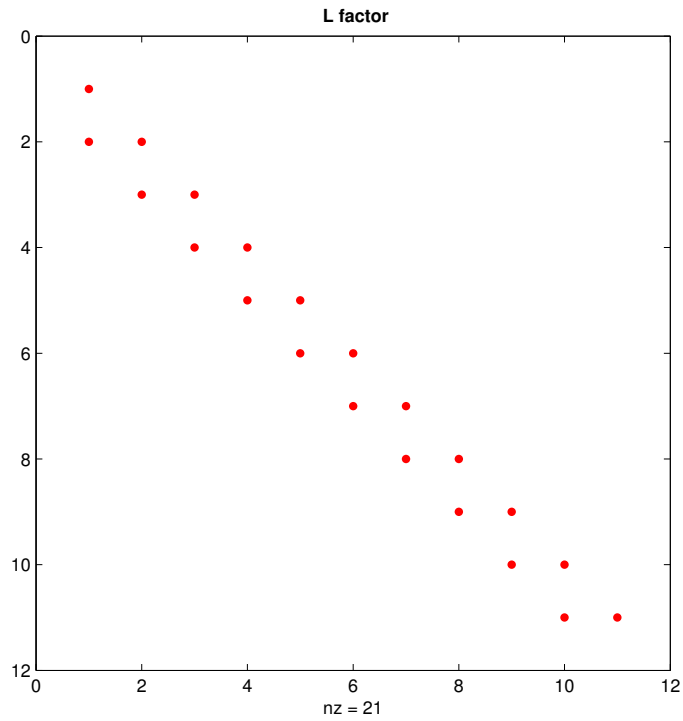
1 % Study of fill-in with LU-factorization due to pivoting
2 n = 10; D = diag(1./(1:n));
3 A = [ D , 2*ones(n,1); 2*ones(1,n) , 2];
4 [L,U,P] = lu(A);
5 figure; spy(A,'r'); title('{\bf arrow matrix A}');
6 print -depsc2 '../PICTURES/fillinpivotA.eps';
7 figure; spy(L,'r'); title('{\bf L factor}');
8 print -depsc2 '../PICTURES/fillinpivotL.eps';
9 figure; spy(U,'r'); title('{\bf U factor}');
10 print -depsc2 '../PICTURES/fillinpivotU.eps';

```

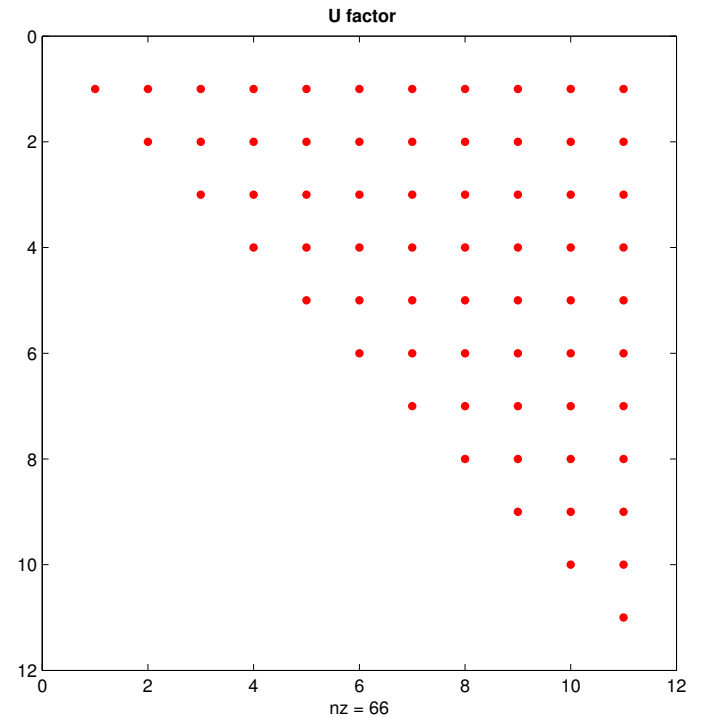
$$\mathbf{A} = \begin{pmatrix} 1 & & & 2 \\ & \frac{1}{2} & & 2 \\ & & \cdots & \vdots \\ & & & \frac{1}{10} & 2 \\ 2 & \cdots & & & 2 \end{pmatrix} \rightarrow \text{arrow matrix, Ex. 2.6.21}$$



**A**



**L**



**U**

In this case the solution of a LSE with system matrix  $\mathbf{A} \in \mathbb{R}^{n,n}$  of the above type by means of Gaussian elimination with partial pivoting would incur costs of  $O(n^3)$ .

☛ a special class of sparse matrices with extra structure:

**Definition 2.6.34** (bandwidth).

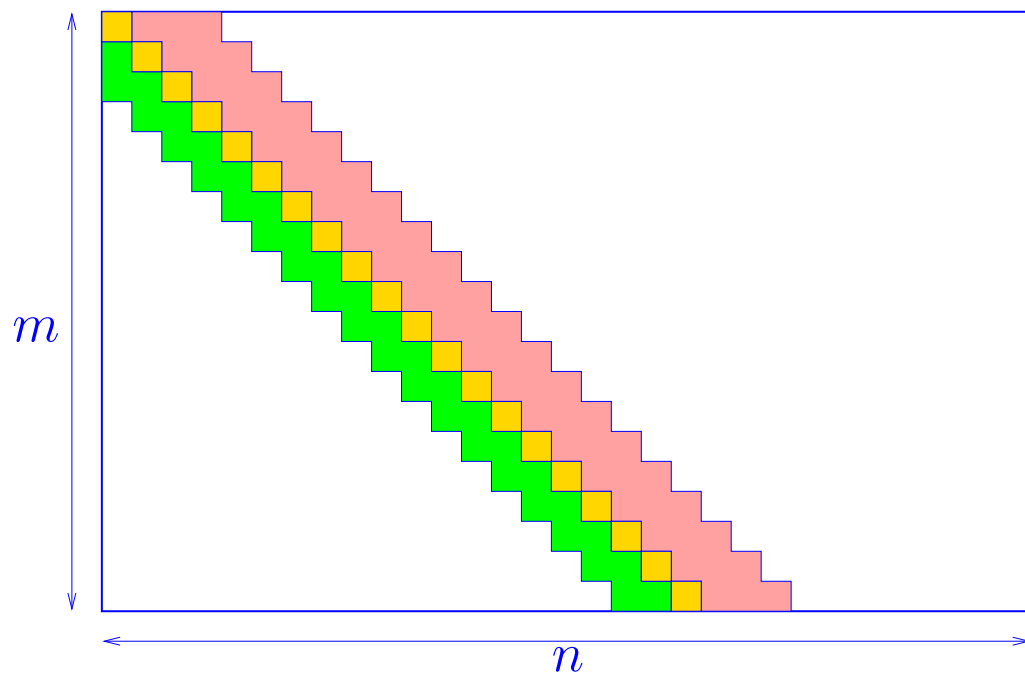
For  $\mathbf{A} = (a_{ij})_{i,j} \in \mathbb{K}^{m,n}$  we call

$$\overline{m}(\mathbf{A}) := \min\{k \in \mathbb{N} : j - i > k \Rightarrow a_{ij} = 0\} \text{ upper bandwidth ,}$$

$$\underline{m}(\mathbf{A}) := \min\{k \in \mathbb{N} : i - j > k \Rightarrow a_{ij} = 0\} \text{ lower bandwidth .}$$

$$m(\mathbf{A}) := \overline{m}(\mathbf{A}) + \underline{m}(\mathbf{A}) + 1 = \text{bandwidth von } \mathbf{A} \text{ (ger.: Bandbreite)}$$

- $m(\mathbf{A}) = 1 \quad \triangleright \quad \mathbf{A}$  diagonal matrix,  $\rightarrow$  Def. 2.2.3
- $\overline{m}(\mathbf{A}) = \underline{m}(\mathbf{A}) = 1 \quad \triangleright \quad \mathbf{A}$  tridiagonal matrix
- More general:  $\mathbf{A} \in \mathbb{R}^{n,n}$  with  $m(\mathbf{A}) \ll n \hat{=} \text{banded matrix}$



- : diagonal
- : super-diagonals
- : sub-diagonals

$$\triangleleft \quad \overline{m}(\mathbf{A}) = 3, \underline{m}(\mathbf{A}) = 2$$

► for banded matrix  $\mathbf{A} \in \mathbb{K}^{m,n}$ :  $\text{nnz}(\mathbf{A}) \leq \min\{m, n\}m(\mathbf{A})$

MATLAB function for creating banded matrices:

dense matrix : `X=diag(v);`

sparse matrix : `X=spdiags(B,d,m,n);` (sparse storage !)

tridiagonal matrix : `X=gallery('tridiag',c,d,e);` (sparse storage !)



We now examine a generalization of the concept of a banded matrix that is particularly useful in the context of Gaussian elimination:

**Definition 2.6.35** (Matrix envelope (*ger.:* Hülle)).

For  $\mathbf{A} \in \mathbb{K}^{n,n}$  define

*row bandwidth*  $m_i^R(\mathbf{A}) := \max\{0, i - j : a_{ij} \neq 0, 1 \leq j \leq n\}, i \in \{1, \dots, n\}$

*column bandwidth*  $m_j^C(\mathbf{A}) := \max\{0, j - i : a_{ij} \neq 0, 1 \leq i \leq n\}, j \in \{1, \dots, n\}$

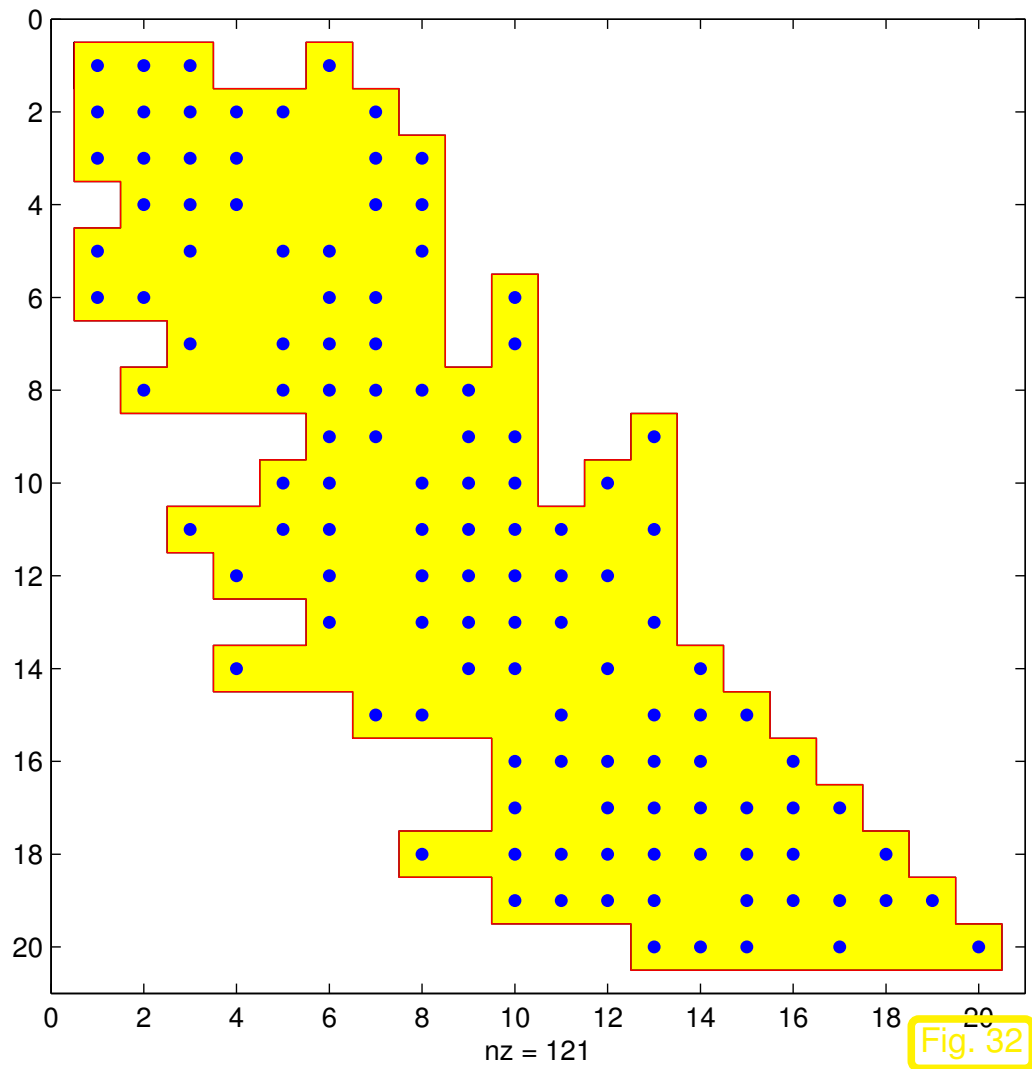
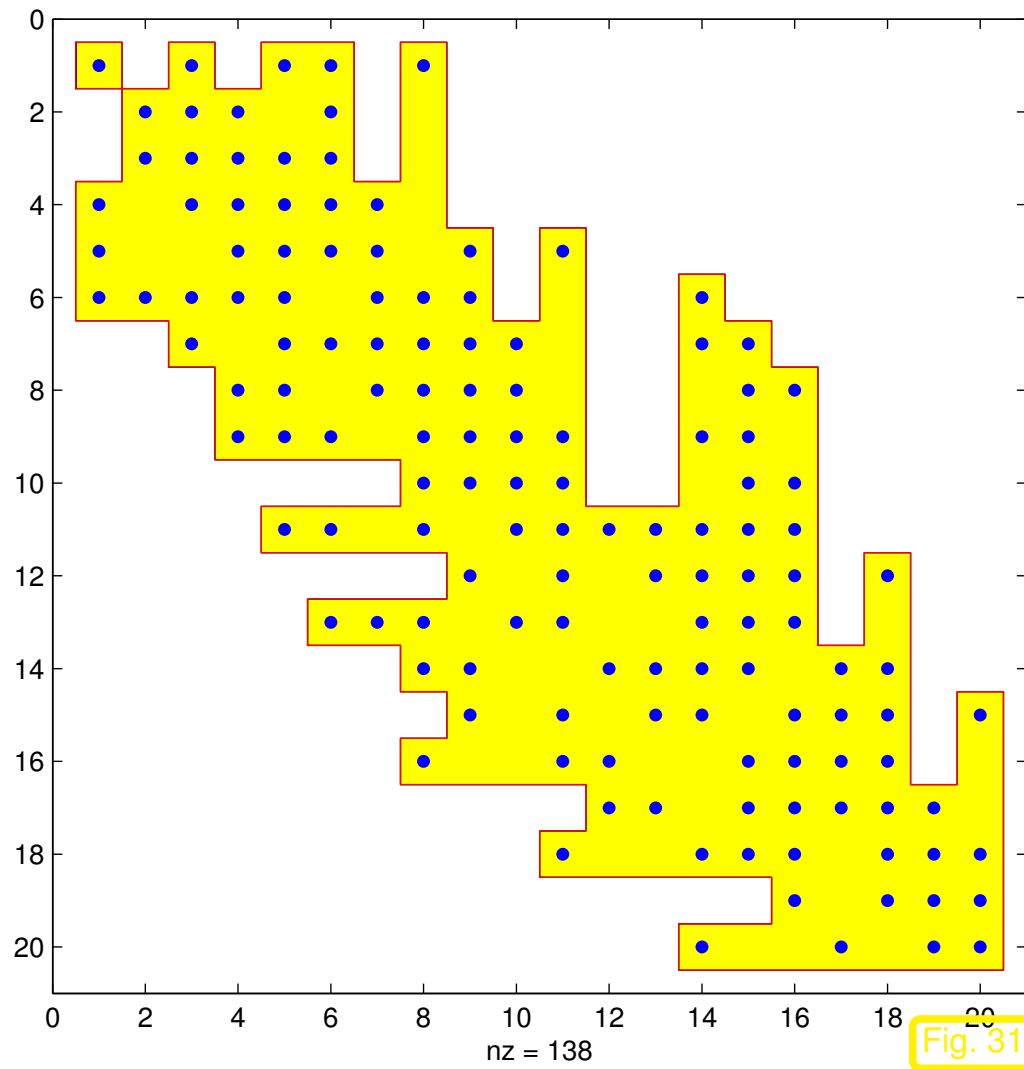
*envelope*  $\text{env}(\mathbf{A}) := \left\{ (i, j) \in \{1, \dots, n\}^2 : \begin{array}{l} i - m_i^R(\mathbf{A}) \leq j \leq i, \\ j - m_j^C(\mathbf{A}) \leq i \leq j \end{array} \right\}$

**Example 2.6.36** (Envelope of a matrix).

$$\mathbf{A} = \begin{pmatrix} * & 0 & * & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & * & 0 & 0 \\ * & 0 & * & 0 & 0 & 0 & * \\ 0 & 0 & 0 & * & * & 0 & * \\ 0 & * & 0 & * & * & * & 0 \\ 0 & 0 & 0 & 0 & * & * & 0 \\ 0 & 0 & * & * & 0 & 0 & * \end{pmatrix} \begin{array}{l} m_1^R(A) = 0 \\ m_2^R(A) = 0 \\ m_3^R(A) = 2 \\ m_4^R(A) = 0 \\ m_5^R(A) = 3 \\ m_6^R(A) = 1 \\ m_7^R(A) = 4 \end{array}$$

$\text{env}(A) =$  red elements

$*$   $\hat{=}$  non-zero matrix entry  $a_{ij} \neq 0$



Note: the envelope of the arrow matrix from Ex. 2.6.21 is just the set of index pairs of its non-zero entries. Hence, the following theorem provides another reason for the sparsity of the LU-factors in that example.

**Theorem 2.6.37** (Envelope and fill-in).  $\rightarrow$  [51, Sect. 3.9]

If  $\mathbf{A} \in \mathbb{K}^{n,n}$  is regular with LU-factorization  $\mathbf{A} = \mathbf{L}\mathbf{U}$ , then fill-in ( $\rightarrow$  Def. 2.6.20) is confined to  $\text{env}(\mathbf{A})$ .

Gaussian elimination *without pivoting*

*Proof.* (by induction, version I) Examine first step of Gaussian elimination without pivoting,  $a_{11} \neq 0$

$$\mathbf{A} = \begin{pmatrix} a_{11} & \mathbf{b}^\top \\ \mathbf{c} & \tilde{\mathbf{A}} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 \\ -\frac{\mathbf{c}}{a_{11}} & \mathbf{I} \end{pmatrix}}_{\mathbf{L}^{(1)}} \underbrace{\begin{pmatrix} a_{11} & \mathbf{b}^\top \\ 0 & \tilde{\mathbf{A}} - \frac{\mathbf{c}\mathbf{b}^\top}{a_{11}} \end{pmatrix}}_{\mathbf{U}^{(1)}}$$

$$\text{If } (i, j) \notin \text{env}(\mathbf{A}) \Rightarrow \begin{cases} c_{i-1} = 0, & \text{if } i > j, \\ b_{j-1} = 0, & \text{if } i < j. \end{cases}$$

$$\Rightarrow \text{env}(\mathbf{L}^{(1)}) \subset \text{env}(\mathbf{A}), \quad \text{env}(\mathbf{U}^{(1)}) \subset \text{env}(\mathbf{A}).$$

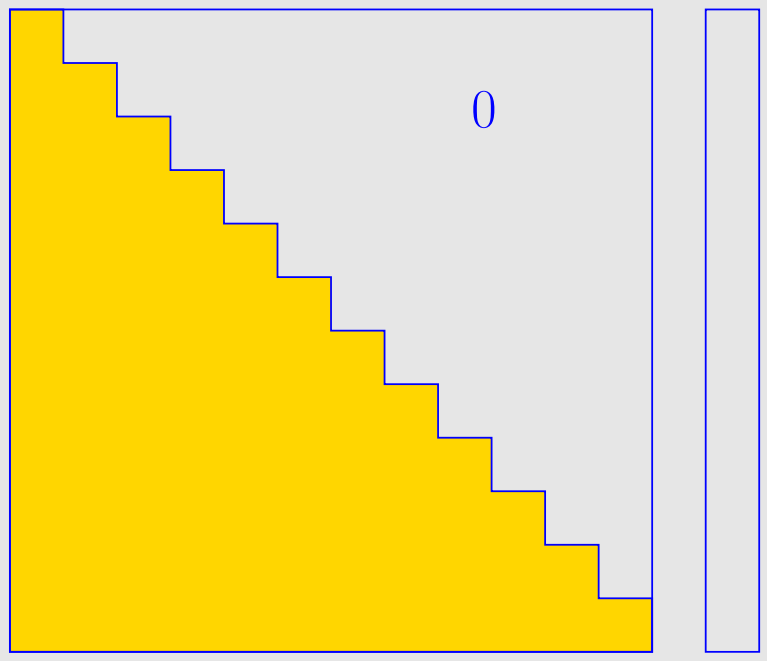
Moreover,  $\text{env}(\tilde{\mathbf{A}} - \frac{\mathbf{c}\mathbf{b}^\top}{a_{11}}) = \text{env}(\mathbf{A}(2:n, 2:n))$

□

*Proof.* (by induction, version II) Use block-LU-factorization, cf. Rem. 2.2.16 and proof of Lemma 2.2.5:

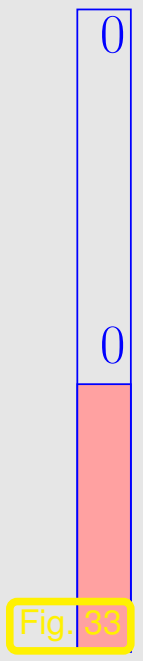
$$\left( \begin{array}{c|c} \tilde{\mathbf{A}} & \mathbf{b} \\ \hline \mathbf{c}^\top & \alpha \end{array} \right) = \left( \begin{array}{c|c} \tilde{\mathbf{L}} & 0 \\ \hline \mathbf{1}^\top & 1 \end{array} \right) \left( \begin{array}{c|c} \tilde{\mathbf{U}} & \mathbf{u} \\ \hline 0 & \xi \end{array} \right) \Rightarrow \begin{array}{l} \tilde{\mathbf{U}}^\top \mathbf{l} = \mathbf{c} \\ \tilde{\mathbf{L}}\mathbf{u} = \mathbf{b} \end{array} \quad (2.6.38)$$

From Def. 2.6.35:



If  $m_n^R(\mathbf{A}) = m$ , then  $c_1, \dots, c_{n-m} = 0$  (entries of  $\mathbf{c}$  from (2.6.38))

If  $m_n^C(\mathbf{A}) = m$ , then  $b_1, \dots, b_{n-m} = 0$  (entries of  $\mathbf{b}$  from (2.6.38))



◁ for lower triangular LSE:

- If  $c_1, \dots, c_k = 0$  then  $l_1, \dots, l_k = 0$
- If  $b_1, \dots, b_k = 0$ , then  $u_1, \dots, u_k = 0$



assertion of the theorem ◻

Thm. 2.6.37 immediately suggests a policy for saving computational effort when solving linear system whose system matrix  $\mathbf{A} \in \mathbb{K}^{n,n}$  is sparse due to *small envelope*:

$$\# \text{env}(\mathbf{A}) \ll n^2 :$$



Policy

Confine elimination to envelope!

Details will be given now:



**Envelope-aware LU-factorization:**

Code 2.6.39: computing row bandwidths,  $\rightarrow$  Def. 2.6.35

```

1 function mr = rowbandwidth(A)
2 % computes row bandwidth numbers  $m_i^R(\mathbf{A})$  of  $\mathbf{A}$ 
3 n = size(A,1); mr = zeros(n,1);
4 for i=1:n, mr(i) = max(0, i-min(find(A(i,:) ~= 0))); end

```

## Code 2.6.40: envelope aware forward substitution

```

1 function y = substenv(L, y, mc)
2 % envelope aware forward substitution for  $\mathbf{Lx} = \mathbf{y}$ 
3 % ( $\mathbf{L}$  = lower triangular matrix)
4 % argument mc: column bandwidth vector
5 n = size(L, 1); y(1) = y(1)/L(1, 1);
6 for i=2:n
7     if (mr(i) > 0)
8         zeta =
9             L(i, i-mr(i):i-1)*y(i-mr(i):i-1);
10        y(i) = (y(i) - zeta)/L(i, i);
11    else y(i) = y(i)/L(i, i); end
12 end

```

Asymptotic complexity  
of envelope aware  
forward substitution, *cf.*  
Alg. 2.2.12, for  $\mathbf{Lx} = \mathbf{y}$ ,  
 $\mathbf{L} \in \mathbb{K}^{n,n}$  regular  
lower triangular matrix is

$$O(\#\text{env}(\mathbf{L}))!$$

By block LU-factorization  $\rightarrow$  Rem. 2.2.16:

$$\left( \begin{array}{c|c} (\mathbf{A})_{1:n-1,1:n-1} & (\mathbf{A})_{1:n-1,n} \\ \hline (\mathbf{A})_{n,1:n-1} & (\mathbf{A})_{n,n} \end{array} \right) = \left( \begin{array}{c|c} \mathbf{L}_1 & 0 \\ \hline \mathbf{I}^\top & 1 \end{array} \right) \left( \begin{array}{c|c} \mathbf{U}_1 & \mathbf{u} \\ \hline 0 & \gamma \end{array} \right), \quad (2.6.41)$$

$$\Rightarrow (\mathbf{A})_{1:n-1,1:n-1} = \mathbf{L}_1 \mathbf{U}_1, \quad \mathbf{L}_1 \mathbf{u} = (\mathbf{A})_{1:n-1,n}, \quad \mathbf{U}_1^\top \mathbf{l} = (\mathbf{A})_{n,1:n-1}^\top, \quad \mathbf{l}^\top \mathbf{u} + \gamma = (\mathbf{A})_{n,n}. \quad (2.6.42)$$

## Code 2.6.43: envelope aware recursive LU-factorization

```

1 function [L,U] = luenv(A)
2 % envelope aware recursive LU-factorization
3 % of structurally symmetric matrix
4 n = size (A,1);
5 if (size (A,2) ~= n),
6     error ('A must be square'); end
7 if (n == 1), L = eye (1); U = A;
8 else
9     mr = rowbandwidth(A);
10    [L1,U1] = luenv(A(1:n-1,1:n-1));
11    u = substenv(L1,A(1:n-1,n),mr);
12    l = substenv(U1',A(n,1:n-1)',mr);
13    if (mr(n) > 0)
14        gamma = A(n,n) -
15            l(n-mr(n):n-1)' * u(n-mr(n):n-1);
16    else gamma = A(n,n); end
17    L = [L1,zeros (n-1,1); l' , 1];
18    U = [U1,u;zeros (1,n-1) , gamma];
19 end

```

◁ recursive implementation  
of envelope aware recur-  
sive LU-factorization (**no  
pivoting !**)

Assumption:

$$\mathbf{A} \in \mathbb{K}^{n,n} \text{ is}$$

**structurally symmetric**

Asymptotic complexity ( $\mathbf{A} \in$   
 $\mathbb{K}^{n,n}$ )

$$O(n \cdot \# \text{env}(\mathbf{A})) .$$

**Definition 2.6.44** (Structurally symmetric matrix).

$\mathbf{A} \in \mathbb{K}^{n,n}$  is *structurally symmetric*, if

$$(\mathbf{A})_{i,j} \neq 0 \Leftrightarrow (\mathbf{A})_{j,i} \neq 0 \quad \forall i, j \in \{1, \dots, n\}.$$

- ▶ Store only  $a_{ij}$ ,  $(i, j) \in \text{env}(\mathbf{A})$  when computing (in situ) LU-factorization of *structurally symmetric*  $\mathbf{A} \in \mathbb{K}^{n,n}$ 
  - ▶ Storage required:  $n + 2 \sum_{i=1}^n m_i(\mathbf{A})$  floating point numbers
  - ▶ **envelope oriented matrix storage**

*Example 2.6.45* (Envelope oriented matrix storage).

Linear envelope oriented matrix storage of *symmetric*  $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$ :



Two arrays:

double \* val size  $P$ ,  
unsigned int \* dptr size  $n$

$$P := n + \sum_{i=1}^n m_i(A) . \tag{2.6.46}$$

$$\mathbf{A} = \begin{pmatrix} * & 0 & * & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & * & 0 & 0 \\ * & 0 & * & 0 & 0 & 0 & * \\ 0 & 0 & 0 & * & * & 0 & * \\ 0 & * & 0 & * & * & * & 0 \\ 0 & 0 & 0 & 0 & * & * & 0 \\ 0 & 0 & * & * & 0 & 0 & * \end{pmatrix}$$

Indexing rule:

$$\text{dptr}[j] = k$$



$$\text{val}[k] = a_{jj}$$

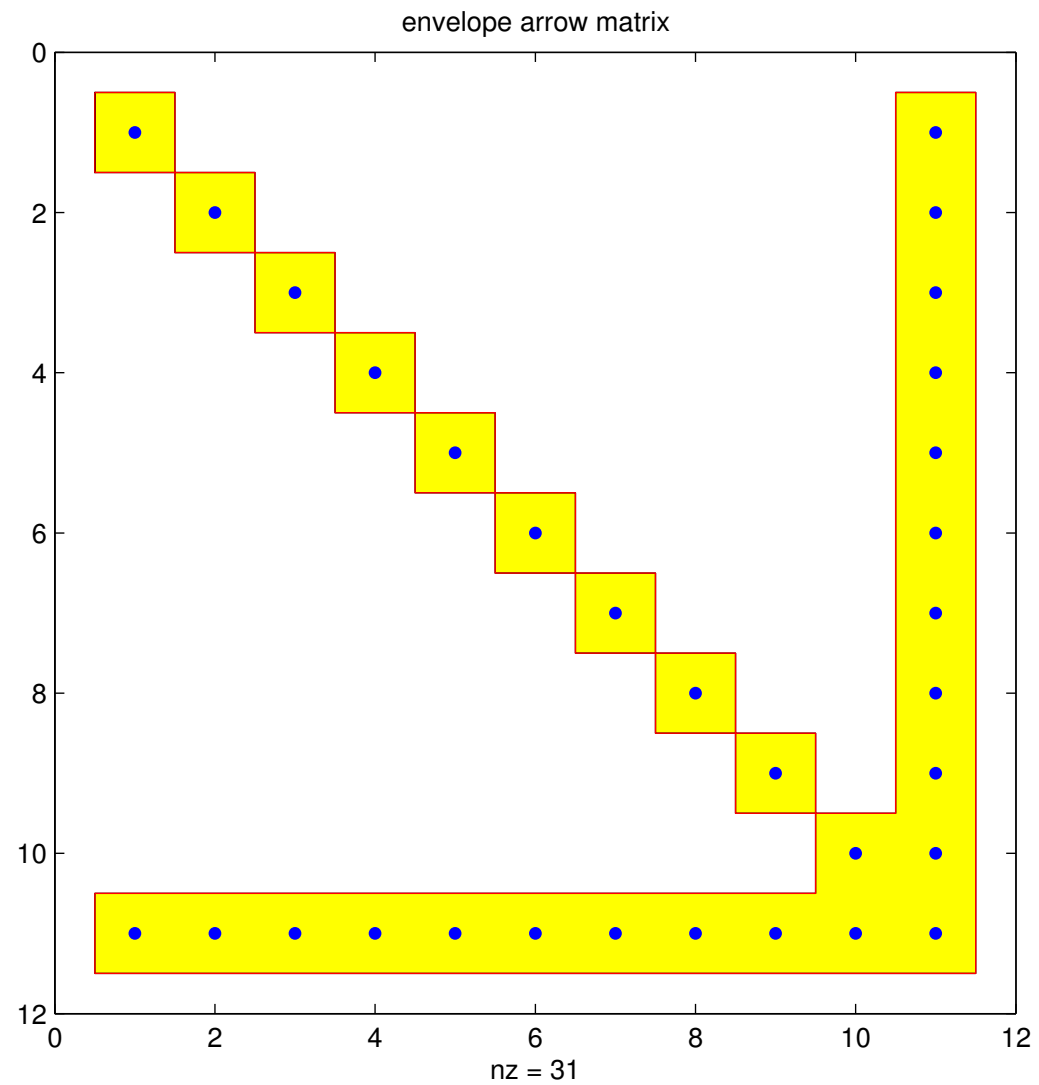
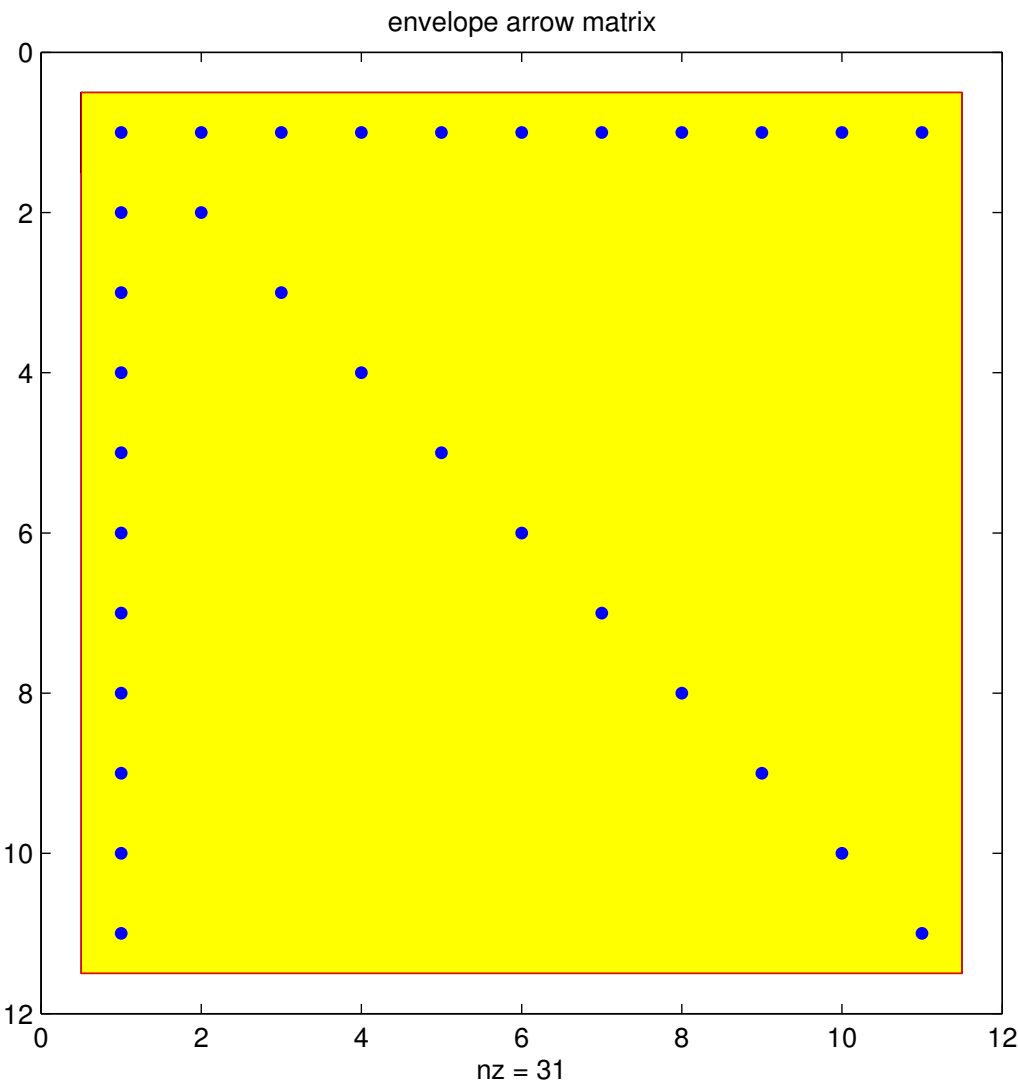
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
val		$a_{11}$	$a_{22}$	$a_{31}$	$a_{32}$	$a_{33}$	$a_{44}$	$a_{52}$	$a_{53}$	$a_{54}$	$a_{55}$	$a_{65}$	$a_{66}$	$a_{73}$	$a_{74}$	$a_{75}$	$a_{76}$	$a_{77}$
dptr	0	1	2	5	6	10	12	17										

### Minimizing bandwidth/envelope:

Goal: Minimize  $m_i(\mathbf{A})$ ,  $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{N,N}$ , by permuting rows/columns of  $\mathbf{A}$

*Example 2.6.47* (Reducing bandwidth by row/column permutations).

Recall: cyclic permutation of rows/columns of arrow matrix  $\rightarrow$  Ex. 2.6.22



Another example: Reflection at cross diagonal ➤ reduction of  $\# \text{env}(\mathbf{A})$

$$\begin{pmatrix} * & 0 & 0 & * & * & * \\ 0 & * & 0 & 0 & 0 & 0 \\ * & 0 & * & 0 & 0 & 0 \\ * & 0 & 0 & * & * & * \end{pmatrix} \rightarrow \begin{pmatrix} * & * & * & 0 & 0 & * \\ * & * & * & 0 & 0 & * \\ 0 & 0 & 0 & * & 0 & 0 \\ * & * & * & 0 & 0 & * \end{pmatrix}$$

$$i \leftarrow N + 1 - i$$

$$\# \text{env}(\mathbf{A}) = 30$$

$$\# \text{env}(\mathbf{A}) = 22$$



Example 2.6.48 (Reducing fill-in by reordering).

**M**: 114×114 symmetric matrix (from computational PDEs)

Code 2.6.49: preordering in MATLAB

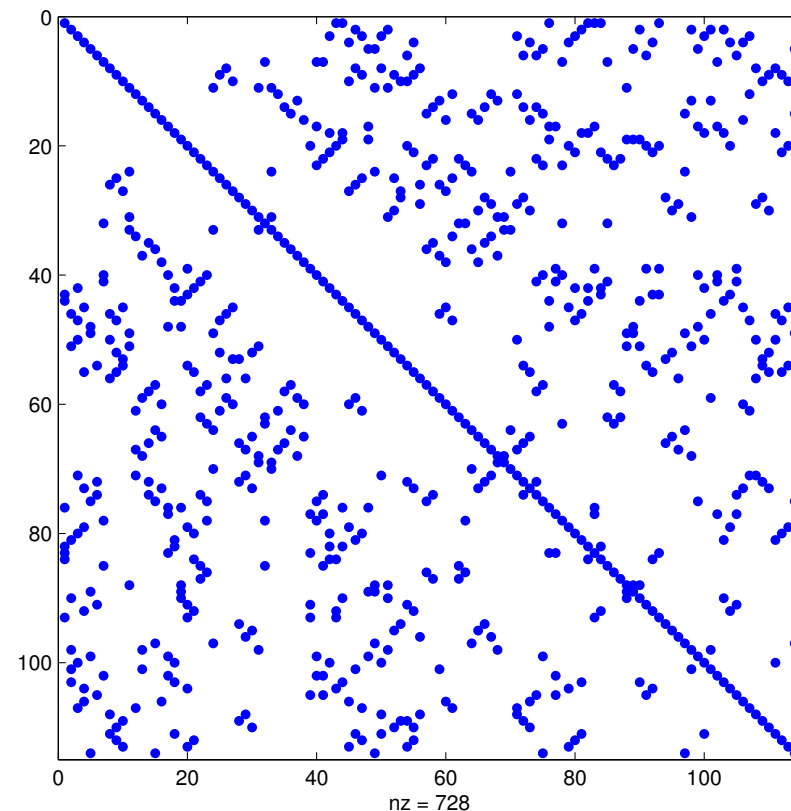
```

1 spy (M) ;
2 [L,U] = lu (M) ; spy (U) ;
3 r = symrcm (M) ;
4 [L,U] = lu (M(r,r)) ; spy (U) ;
5 m = symamd (M) ;
6 [L,U] = lu (M(m,m)) ; spy (U) ;

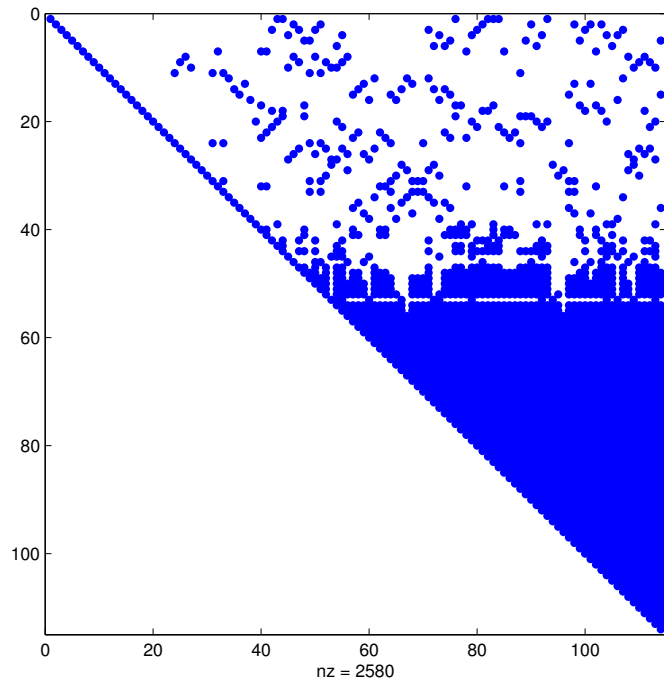
```

Pattern of **M** →

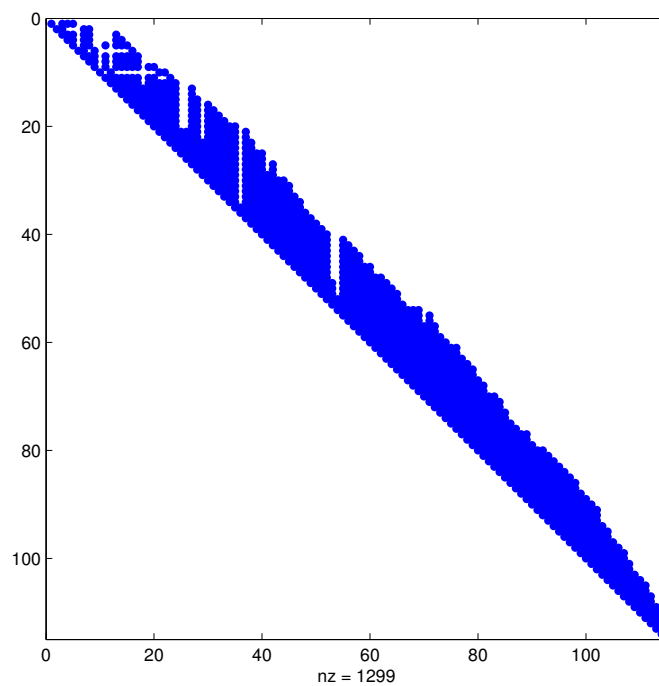
(Here: no row swaps from pivoting !)



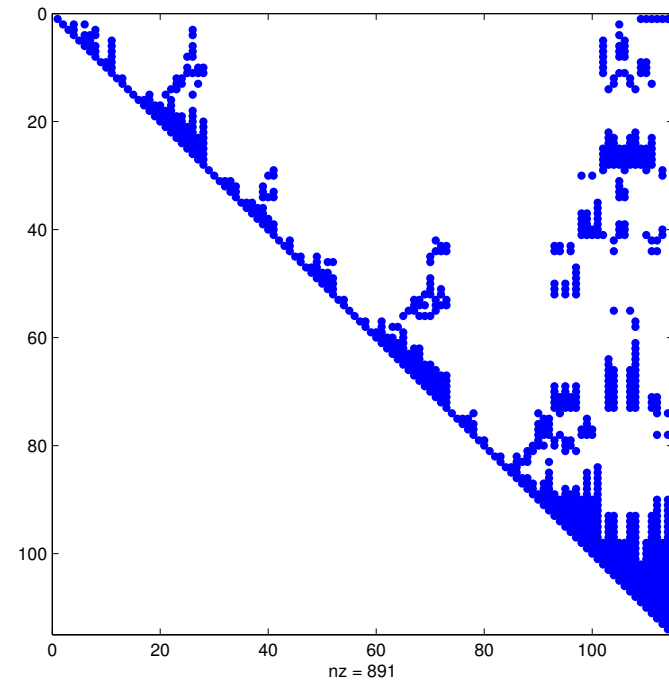
Examine patterns of LU-factors (→ Sect. 2.2) after reordering:



no reordering



reverse Cuthill-McKee



approximate minimum degree

Cuthill-McKee algorithm → [51, Sect. 3.9.1]



Advice: Use numerical libraries for solving LSE with sparse system matrices !

- **SuperLU** (<http://www.cs.berkeley.edu/~demmel/SuperLU.html>)
- **UMFPACK** (<http://www.cise.ufl.edu/research/sparse/umfpack/>)
- **Pardiso** (<http://www.pardiso-project.org/>)
- **Matlab-\** (on sparse storage formats)

## 2.7 Stable Gaussian elimination without pivoting

Thm. 2.6.37  $\triangleright$  special structure of the matrix helps avoid fill-in in Gaussian elimination/LU-factorization *without* pivoting.

Ex. 2.6.32  $\triangleright$  pivoting can trigger huge fill-in that would not occur without it.

Ex. 2.6.48  $\triangleright$  fill-in reducing effect of reordering can be thwarted by later row swapping in the course of pivoting.

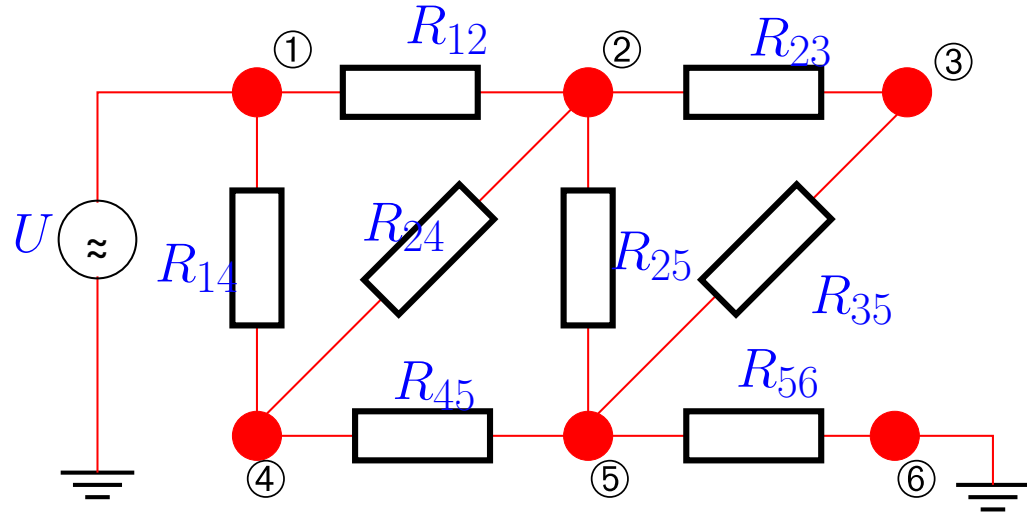
Sect. 2.5.3: pivoting essential for stability of Gaussian elimination/LU-factorization

► Very desirable: a priori criteria, when Gaussian elimination/LU-factorization remains stable even without pivoting. This can help avoid the extra work for partial pivoting and makes it possible to exploit structure without worrying about stability.

This section will introduce classes of matrices that allow Gaussian elimination without pivoting. Fortunately, linear systems of equations featuring system matrices from these classes are very common in applications.

Example 2.7.1 (Diagonally dominant matrices from nodal analysis). → Ex. 2.6.3

Consider:  
electrical circuit entirely composed of  
Ohmic resistors.  
Circuit equations from nodal analysis, see  
Ex. 2.6.3:



$$\begin{aligned}
 \textcircled{2} : & \quad R_{12}^{-1}(U_2 - U_1) + R_{23}^{-1}(U_2 - U_3) - R_{24}^{-1}(U_2 - U_4) + R_{25}^{-1}(U_2 - U_5) = 0, \\
 \textcircled{3} : & \quad R_{23}^{-1}(U_3 - U_2) + R_{35}^{-1}(U_3 - U_5) = 0, \\
 \textcircled{4} : & \quad R_{14}^{-1}(U_4 - U_1) - R_{24}^{-1}(U_4 - U_2) + R_{45}^{-1}(U_4 - U_5) = 0, \\
 \textcircled{5} : & \quad R_{25}^{-1}(U_5 - U_2) + R_{35}^{-1}(U_5 - U_3) + R_{45}^{-1}(U_5 - U_4) + R_{56}(U_5 - U_6) = 0, \\
 & \quad U_1 = U \quad , \quad U_6 = 0 .
 \end{aligned}$$



$$\begin{pmatrix} \frac{1}{R_{12}} + \frac{1}{R_{23}} + \frac{1}{R_{24}} + \frac{1}{R_{25}} & -\frac{1}{R_{23}} & -\frac{1}{R_{24}} & -\frac{1}{R_{25}} \\ & \frac{1}{R_{23}} + \frac{1}{R_{35}} & 0 & -\frac{1}{R_{35}} \\ & -\frac{1}{R_{24}} & \frac{1}{R_{24}} + \frac{1}{R_{45}} & -\frac{1}{R_{45}} \\ & -\frac{1}{R_{25}} & -\frac{1}{R_{35}} & \frac{1}{R_{22}} + \frac{1}{R_{35}} + \frac{1}{R_{45}} + \frac{1}{R_{56}} \end{pmatrix} \begin{pmatrix} U_2 \\ U_3 \\ U_4 \\ U_5 \end{pmatrix} = \begin{pmatrix} \frac{1}{R_{12}} \\ 0 \\ \frac{1}{R_{14}} \\ 0 \end{pmatrix} U$$

➤ Matrix  $\mathbf{A} \in \mathbb{R}^{n,n}$  arising from nodal analysis satisfies

$$\bullet \quad \mathbf{A} = \mathbf{A}^\top \quad , \quad a_{kk} > 0 \quad , \quad a_{kj} \leq 0 \text{ for } k \neq j \quad , \quad (2.7.2)$$

$$\bullet \quad \sum_{j=1}^n a_{kj} \geq 0 \quad , \quad k = 1, \dots, n \quad , \quad (2.7.3)$$

- $\mathbf{A}$  is regular.

(2.7.4)

All these properties are obvious except for the fact that  $\mathbf{A}$  is regular.

Proof of (2.7.4): By Thm. 2.0.3 it suffices to show that the nullspace of  $\mathbf{A}$  is trivial:  $\mathbf{Ax} = 0 \Rightarrow \mathbf{x} = 0$ .

Pick  $\mathbf{x} \in \mathbb{R}^n$ ,  $\mathbf{Ax} = 0$ , and  $i \in \{1, \dots, n\}$  so that

$$|x_i| = \max\{|x_j|, j = 1, \dots, n\} .$$

Intermediate goal: show that all entries of  $\mathbf{x}$  are the same

$$\mathbf{Ax} = 0 \Rightarrow x_i = \sum_{j \neq i} \frac{a_{ij}}{a_{ii}} x_j \Rightarrow |x_i| \leq \sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} |x_j| . \quad (2.7.5)$$

By (2.7.3) and the sign condition from (2.7.2) we conclude

$$\sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} \leq 1 . \quad (2.7.6)$$



Hence, (2.7.6) combined with the above estimate (2.7.5) that tells us that the maximum is smaller equal than a mean implies  $|x_j| = |x_i|$  for all  $j = 1, \dots, n$ . Finally, the sign condition  $a_{kj} \leq 0$  for  $k \neq j$  enforces the same sign of all  $x_i$ . Thus, we conclude, w.l.o.g.,  $x_1 = x_2 = \dots = x_n$ . As

$$\exists i \in \{1, \dots, n\}: \sum_{j=1}^n a_{ij} > 0 \quad (\text{strict inequality}),$$

$\mathbf{Ax} = 0$  is only possible for  $\mathbf{x} = 0$ .



**Definition 2.7.7** (Diagonally dominant matrix).  $\rightarrow [51, \text{Def. 1.24}]$

$\mathbf{A} \in \mathbb{K}^{n,n}$  is *diagonally dominant*, if

$$\forall k \in \{1, \dots, n\}: \sum_{j \neq k} |a_{kj}| \leq |a_{kk}|.$$

The matrix  $\mathbf{A}$  is called *strictly diagonally dominant*, if

$$\forall k \in \{1, \dots, n\}: \sum_{j \neq k} |a_{kj}| < |a_{kk}|.$$

**Lemma 2.7.8** (LU-factorization of diagonally dominant matrices).

$$\mathbf{A} \text{ regular, diagonally dominant} \iff \left\{ \begin{array}{l} \mathbf{A} \text{ has LU-factorization} \\ \text{Gaussian elimination feasible without pivoting}^{(*)} \end{array} \right.$$

$\Updownarrow$

*with positive diagonal*

(\*) : partial pivoting & diagonally dominant matrices  $\triangleright$  triggers no row permutations !

((2.3.7) will always be satisfied for  $j = k$ )

$\triangleright$  Pivoting can be dispensed with without compromising stability.

*Proof.*(of Lemma 2.7.8)

(2.1.11)  $\rightarrow$  induction w.r.t.  $n$ :

Clear: partial pivoting in the first step selects  $a_{11}$  as pivot element, *cf.* (2.3.7).

After 1st step of elimination:

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j}, \quad i, j = 2, \dots, n \Rightarrow a_{ii}^{(1)} > 0.$$

$$\begin{aligned} \blacktriangleright \quad |a_{ii}^{(1)}| - \sum_{\substack{j=2 \\ j \neq i}}^n |a_{ij}^{(1)}| &= \left| a_{ii} - \frac{a_{i1}}{a_{11}}a_{1i} \right| - \sum_{\substack{j=2 \\ j \neq i}}^n \left| a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j} \right| \\ &\geq a_{ii} - \frac{|a_{i1}||a_{1i}|}{a_{11}} - \sum_{\substack{j=2 \\ j \neq i}}^n |a_{ij}| - \frac{|a_{i1}|}{a_{11}} \sum_{\substack{j=2 \\ j \neq i}}^n |a_{1j}| \\ &\geq a_{ii} - \frac{|a_{i1}||a_{1i}|}{a_{11}} - \sum_{\substack{j=2 \\ j \neq i}}^n |a_{ij}| - |a_{i1}| \frac{a_{11} - |a_{1i}|}{a_{11}} \geq a_{ii} - \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}| \geq 0. \end{aligned}$$

**A** regular, diagonally dominant  $\Rightarrow$  partial pivoting according to (2.3.7) selects  $i$ -th row in  $i$ -th step.

There is another important class of matrices permitting stable Gaussian elimination without pivoting:

**Definition 2.7.9** (Symmetric positive definite (s.p.d.) matrices).  $\rightarrow$  [13, Def. 3.31], [51, Def. 1.22]

$\mathbf{M} \in \mathbb{K}^{n,n}$ ,  $n \in \mathbb{N}$ , is *symmetric (Hermitian) positive definite* (s.p.d.), if

$$\mathbf{M} = \mathbf{M}^H \quad \wedge \quad \mathbf{x}^H \mathbf{M} \mathbf{x} > 0 \quad \Leftrightarrow \quad \mathbf{x} \neq \mathbf{0}.$$

If  $\mathbf{x}^H \mathbf{M} \mathbf{x} \geq 0$  for all  $\mathbf{x} \in \mathbb{K}^n$   $\triangleright$   $\mathbf{M}$  *positive semi-definite*.

**Lemma 2.7.10** (Necessary conditions for s.p.d.).  $\rightarrow$  [13, Satz 3.33], [51, Prop. 1.18]

For a symmetric/Hermitian positive definite matrix  $\mathbf{M} = \mathbf{M}^H \in \mathbb{K}^{n,n}$  holds true:

1.  $m_{ii} > 0$ ,  $i = 1, \dots, n$ ,
2.  $m_{ii}m_{jj} - |m_{ij}|^2 > 0 \quad \forall 1 \leq i < j \leq n$ ,
3. all eigenvalues of  $\mathbf{M}$  are positive. ( $\leftarrow$  also sufficient for symmetric/Hermitian  $\mathbf{M}$ )

*Remark 2.7.11 (S.p.d. Hessians).*

Recall from analysis: in a local minimum  $x^*$  of a  $C^2$ -function  $f : \mathbb{R}^n \mapsto \mathbb{R}$   $\blacktriangleright$  Hessian  $D^2f(x^*)$   
s.p.d.

To compute the minimum of a  $C^2$ -function iteratively by means of Newton's method ( $\rightarrow$  Sect. 4.4) a linear system of equations with the s.p.d. Hessian as system matrix has to be solved in each step.

The solutions of many equations in science and engineering boils down to finding the minimum of some (energy, entropy, etc.) function, which accounts for the prominent role of s.p.d. linear systems in applications.



**Lemma 2.7.12.** *A diagonally dominant Hermitian/symmetric matrix with non-negative diagonal entries is positive semi-definite.*

*A strictly diagonally dominant Hermitian/symmetric matrix with positive diagonal entries is positive definite.*

*Proof.* For  $\mathbf{A} = \mathbf{A}^H$  diagonally dominant, use inequality between arithmetic and geometric mean (AGM)  $ab \leq \frac{1}{2}(a^2 + b^2)$ :

$$\begin{aligned}
 \mathbf{x}^H \mathbf{A} \mathbf{x} &= \sum_{i=1}^n \left( a_{ii} |x_i|^2 + \sum_{i \neq j} a_{ij} \bar{x}_i x_j \right) \geq \sum_{i=1}^n \left( a_{ii} |x_i|^2 - \sum_{i \neq j} |a_{ij}| |x_i| |x_j| \right) \\
 &\stackrel{\text{AGM}}{\geq} \sum_{i=1}^n a_{ii} |x_i|^2 - \frac{1}{2} \sum_{i \neq j} |a_{ij}| (|x_i|^2 + |x_j|^2) \\
 &\geq \frac{1}{2} \left( \sum_{i=1}^n \{ a_{ii} |x_i|^2 - \sum_{j \neq i} |a_{ij}| |x_i|^2 \} \right) + \frac{1}{2} \left( \sum_{j=1}^n \{ a_{ii} |x_j|^2 - \sum_{i \neq j} |a_{ij}| |x_j|^2 \} \right) \\
 &\geq \sum_{i=1}^n |x_i|^2 \left( a_{ii} - \sum_{j \neq i} |a_{ij}| \right) \geq 0.
 \end{aligned}$$

**Theorem 2.7.13** (Gaussian elimination for s.p.d. matrices).

*Every symmetric/Hermitian positive definite matrix ( $\rightarrow$  Def. 2.7.9) possesses an LU-decomposition ( $\rightarrow$  Sect. 2.2).*

Equivalent to assertion of theorem: Gaussian elimination feasible *without pivoting*

In fact, this theorem is a corollary of Lemma 2.2.5, because all principal minors of an s.p.d. matrix are s.p.d. themselves.

*Sketch of alternative self-contained proof.*

Proof by induction: consider first step of elimination

$$\mathbf{A} = \begin{pmatrix} a_{11} & \mathbf{b}^\top \\ \mathbf{b} & \tilde{\mathbf{A}} \end{pmatrix} \xrightarrow[\text{Gaussian elimination}]{\text{1. step}} \begin{pmatrix} a_{11} & \mathbf{b}^\top \\ 0 & \tilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^\top}{a_{11}} \end{pmatrix} .$$

➤ to show:  $\tilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^\top}{a_{11}}$  s.p.d. ( $\rightarrow$  step of induction argument)

Evident: symmetry of  $\tilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^\top}{a_{11}} \in \mathbb{R}^{n-1,n-1}$

As  $\mathbf{A}$  s.p.d. ( $\rightarrow$  Def. 2.7.9), for every  $\mathbf{y} \in \mathbb{R}^{n-1} \setminus \{0\}$

$$0 < \begin{pmatrix} -\frac{\mathbf{b}^\top \mathbf{y}}{a_{11}} \\ \mathbf{y} \end{pmatrix}^\top \begin{pmatrix} a_{11} & \mathbf{b}^\top \\ \mathbf{b} & \tilde{\mathbf{A}} \end{pmatrix} \begin{pmatrix} -\frac{\mathbf{b}^\top \mathbf{y}}{a_{11}} \\ \mathbf{y} \end{pmatrix} = \mathbf{y}^\top \left( \tilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^\top}{a_{11}} \right) \mathbf{y} .$$

►  $\tilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^\top}{a_{11}}$  positive definite. □

The proof can also be based on the identities

$$\left( \begin{array}{c|c} (\mathbf{A})_{1:n-1,1:n-1} & (\mathbf{A})_{1:n-1,n} \\ \hline (\mathbf{A})_{n,1:n-1} & (\mathbf{A})_{n,n} \end{array} \right) = \left( \begin{array}{c|c} \mathbf{L}_1 & 0 \\ \hline \mathbf{1}^\top & 1 \end{array} \right) \left( \begin{array}{c|c} \mathbf{U}_1 & \mathbf{u} \\ \hline 0 & \gamma \end{array} \right) , \quad (2.6.41)$$

$$\Rightarrow (\mathbf{A})_{1:n-1,1:n-1} = \mathbf{L}_1 \mathbf{U}_1 , \quad \mathbf{L}_1 \mathbf{u} = (\mathbf{A})_{1:n-1,n} , \quad \mathbf{U}_1^\top \mathbf{1} = (\mathbf{A})_{n,1:n-1}^\top , \quad \mathbf{1}^\top \mathbf{u} + \gamma = (\mathbf{A})_{n,n} ,$$

noticing that the principal minor  $(\mathbf{A})_{1:n-1,1:n-1}$  is also s.p.d. This allows a simple induction argument.

Note:

no pivoting required ( $\rightarrow$  Sect. 2.3)  
(partial pivoting always picks current pivot row)



**Lemma 2.7.14** (Cholesky decomposition for s.p.d. matrices).  $\rightarrow$  [27, Sect. 3.4], [35, Sect. II.5], [51, Thm. 3.6]

For any s.p.d.  $\mathbf{A} \in \mathbb{K}^{n,n}$ ,  $n \in \mathbb{N}$ , there is a unique upper triangular matrix  $\mathbf{R} \in \mathbb{K}^{n,n}$  with  $r_{ii} > 0$ ,  $i = 1, \dots, n$ , such that  $\mathbf{A} = \mathbf{R}^H \mathbf{R}$  (*Cholesky decomposition*).

Thm. 2.7.13  $\Rightarrow \mathbf{A} = \mathbf{L}\mathbf{U}$  (unique  $LU$ -decomposition of  $\mathbf{A}$ , Lemma 2.2.5)

$$\mathbf{A} = \mathbf{L}\mathbf{D}\tilde{\mathbf{U}}, \quad \begin{array}{l} \mathbf{D} \hat{=} \text{diagonal of } \mathbf{U}, \\ \tilde{\mathbf{U}} \hat{=} \text{normalized upper triangular matrix} \rightarrow \text{Def. 2.2.3} \end{array}$$

Due to uniqueness of  $LU$ -decomposition

$$\mathbf{A} = \mathbf{A}^T \Rightarrow \mathbf{U} = \mathbf{D}\mathbf{L}^T \Rightarrow \boxed{\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T},$$

with unique  $\mathbf{L}$ ,  $\mathbf{D}$  (diagonal matrix)

$$\mathbf{x}^T \mathbf{A} \mathbf{x} > 0 \quad \forall \mathbf{x} \neq 0 \Rightarrow \mathbf{y}^T \mathbf{D} \mathbf{y} > 0 \quad \forall \mathbf{y} \neq 0.$$

►  $\mathbf{D}$  has positive diagonal  $\Rightarrow \mathbf{R} = \sqrt{\mathbf{D}}\mathbf{L}^T.$  □

Formulas analogous to (2.2.6)

$$\mathbf{R}^H \mathbf{R} = \mathbf{A} \Rightarrow a_{ik} = \sum_{j=1}^{\min\{i,k\}} \bar{r}_{ji} r_{jk} = \begin{cases} \sum_{j=1}^{i-1} \bar{r}_{ji} r_{jk} + \bar{r}_{ii} r_{ik} & , \text{ if } i < k , \\ \sum_{j=1}^{i-1} |r_{ji}|^2 + r_{ii}^2 & , \text{ if } i = k . \end{cases} \quad (2.7.15)$$

Code 2.7.16: simple Cholesky factorization

```

1 function R = cholfac(A)
2 % simple Cholesky factorization
3 n = size(A,1);
4 for k = 1:n
5     for j=k+1:n
6         A(j,j:n) = A(j,j:n) -
7             A(k,j:n)*A(k,j)/A(k,k);
8     end
9     A(k,k:n) =
10        A(k,k:n) / sqrt(A(k,k));
end
R = triu(A);

```

Computational costs (#

elementary arithmetic operations) of Cholesky

decomposition:  $\frac{1}{6}n^3 + O(n^2)$ 

(➤ “half the costs” of LU-factorization, cf. Code. 2.2.6, but this does not mean “twice as fast” in a concrete implementation, because memory access patterns will have a crucial impact, see Rem. 1.3.8.)

Gains in efficiency hardly justify the use of Cholesky decomposition in modern numerical algorithms.

Savings in memory compared to standard LU-factorization (only one factor  $\mathbf{R}$  has to be stored) offer a stronger reason to prefer the Cholesky decomposition.

MATLAB function:

$$\mathbf{R} = \text{chol}(\mathbf{A})$$

The computation of Cholesky-factorization by means of the algorithm of Code 2.7.15 is **numerically stable** ( $\rightarrow$  Def. 2.5.11)!

Reason: recall Thm. 2.5.13: Numerical instability of Gaussian elimination (with any kind of pivoting) manifests itself in massive growth of the entries of intermediate elimination matrices  $\mathbf{A}^{(k)}$ .

Use the relationship between LU-factorization and Cholesky decomposition, which tells us that we only have to monitor the growth of entries of intermediate upper triangular “Cholesky factorization matrices”  $\mathbf{A} = (\mathbf{R}^{(k)})^H \mathbf{R}^{(k)}$ .

Consider: Euclidean vector norm/matrix norm ( $\rightarrow$  Def. 2.5.5)  $\|\cdot\|_2$

$$\mathbf{A} = \mathbf{R}^H \mathbf{R} \Rightarrow \|\mathbf{A}\|_2 = \sup_{\|\mathbf{x}\|_2=1} \mathbf{x}^H \mathbf{R}^H \mathbf{R} \mathbf{x} = \sup_{\|\mathbf{x}\|_2=1} (\mathbf{R} \mathbf{x})^H (\mathbf{R} \mathbf{x}) = \|\mathbf{R}\|_2^2 .$$

$$\left\| (\mathbf{R}^{(k)})^H \right\|_2 = \left\| \mathbf{R}^{(k)} \right\|_2 = \|\mathbf{A}\|_2^{1/2}$$

 $\mathbf{R}^{(k)}$ 

Computation of the Cholesky decomposition largely agrees with the computation of LU-factorization (without pivoting). Using the latter together with forward and backward substitution ( $\rightarrow$  Sect. 2.2) to solve a linear system of equations is algebraically and numerically equivalent to using Gaussian elimination without pivoting.

From these equivalences we conclude:

Solving LSE with s.p.d. system matrix via

Cholesky decomposition + forward & backward substitution

is numerically stable ( $\rightarrow$  Def. 2.5.11)



Gaussian elimination *without pivoting* is a *numerically stable* way to solve LSEs with s.p.d. system matrix.

*Remark 2.7.17* (Telling MATLAB about matrix properties).

MATLAB-\ assumes generic matrix, cannot detect special properties of (fully populated) matrix (e.g. symmetric, s.p.d., triangular).



Use `y = linsolve(A,b,opts)`

$opts \in \{$ 

LT	$\leftrightarrow$	A lower triangular matrix
UT	$\leftrightarrow$	A upper triangular matrix
UHES	$\leftrightarrow$	A upper Hessenberg matrix
SYM	$\leftrightarrow$	A Hermitian matrix
POSDEF	$\leftrightarrow$	A positive definite matrix

 $\}$



## 2.8 QR-factorization/decomposition [35, Sect. 13], [27, Sect. 7.3]

*Remark 2.8.1* (Sensitivity of linear mappings).

Consider problem map ( $\rightarrow$  Sect. 2.5.2)

$$F : \begin{cases} \mathbb{K}^n \mapsto \mathbb{K}^n \\ \mathbf{x} \mapsto \mathbf{Ax} \end{cases} \quad \text{for given regular } \mathbf{A} \in \mathbb{K}^{n,n} \quad \triangleright \quad \mathbf{x} \hat{=} \text{“data”}$$

Goal: Estimate relative perturbations in  $F(\mathbf{x})$  due to relative perturbations in  $\mathbf{x}$ .

(*cf.* the same investigations for linear systems of equations in Sect. 2.5.5 and Thm. 2.5.24)

We assume that  $\mathbb{K}^n$  is equipped with *some* vector norm ( $\rightarrow$  Def. 2.5.1) and we use the induced matrix norm ( $\rightarrow$  Def. 2.5.5) on  $\mathbb{K}^{n,n}$ .

Using linearity and the elementary estimate  $\|\mathbf{Mx}\| \leq \|\mathbf{M}\| \|\mathbf{x}\|$ , which is a direct consequence of the definition of an induced matrix norm, we obtain

$$\begin{aligned} \mathbf{Ax} = \mathbf{y} &\Rightarrow \|\mathbf{x}\| \leq \|\mathbf{A}^{-1}\| \|\mathbf{y}\| \\ \mathbf{A}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{y} + \Delta\mathbf{y} &\Rightarrow \mathbf{A}\Delta\mathbf{x} = \Delta\mathbf{y} \Rightarrow \|\Delta\mathbf{y}\| \leq \|\mathbf{A}\| \|\Delta\mathbf{x}\| \end{aligned}$$

$$\Rightarrow \frac{\|\Delta \mathbf{y}\|}{\|\mathbf{y}\|} \leq \frac{\|\mathbf{A}\| \|\Delta \mathbf{x}\|}{\|\mathbf{A}^{-1}\|^{-1} \|\mathbf{x}\|} = \text{cond}(\mathbf{A}) \frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} \quad (2.8.2)$$

relative perturbation in result

relative perturbation in data

➤ Condition number  $\text{cond}(\mathbf{A})$  ( $\rightarrow$  Def. 2.5.26) bounds amplification of relative error in argument vector in matrix  $\times$  vector-multiplication  $\mathbf{x} \mapsto \mathbf{Ax}$ . △

Example 2.8.3 (Conditioning of row transformations).

$2 \times 2$  Row transformation matrix (*cf.* elimination matrices of Gaussian elimination, Sect. 2.2):

$$\mathbf{T}(\mu) = \begin{pmatrix} 1 & 0 \\ \mu & 1 \end{pmatrix}$$

Condition numbers of  $\mathbf{T}(\mu)$  ▷

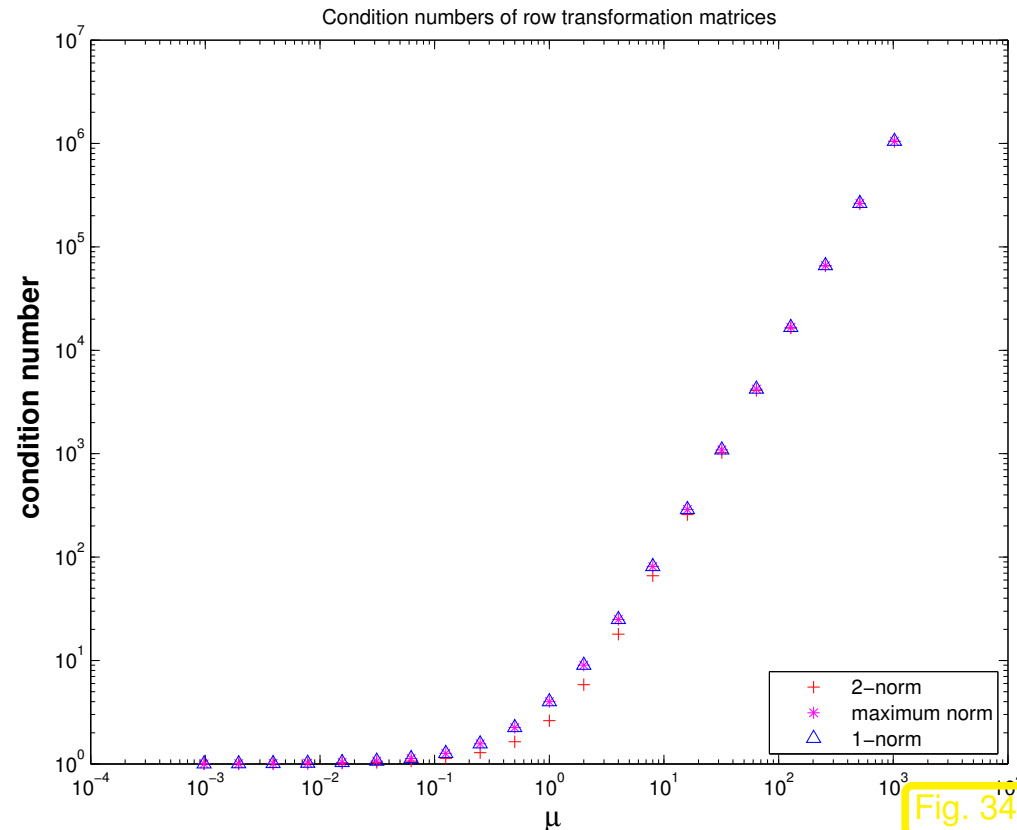


Fig. 34

## Code 2.8.4: computing different condition numbers of row transformation matrices

```
1 T = eye(2); res = [];  
2 for mult = 2.^(-10:10)  
3     T(1,2) = mult;  
4     res = [res; mult, cond(T,2), cond(T,'inf'), cond(T,1)];  
5 end  
6 figure;  
7 loglog(res(:,1),res(:,2),'r+', res(:,1),res(:,3),'m*',  
8     res(:,1),res(:,4),'b^');  
9 xlabel('\bf \mu','fontsize',14);  
0 ylabel('\bf condition number','fontsize',14);  
1 title('Condition numbers of row transformation matrices');  
2 legend('2-norm', 'maximum norm', '1-norm', 'location',  
3     'southeast');  
4 print -depsc2 '../PICTURES/rowtrfcond.eps';
```

Observation:  $\text{cond}(\mathbf{T}(\mu))$  large for large  $\mu$

As explained in Sect. 2.2, Gaussian (forward) elimination can be viewed as successive multiplication



with elimination matrices. If an elimination matrix has a large condition number, then small relative errors in the entries of intermediate matrices caused by earlier roundoff errors can experience massive amplification and, thus, spoil all further steps (► loss of numerical stability, Def. 2.5.11).

Therefore, the entries of elimination matrices should be kept small, and this is the main rationale behind (partial) pivoting (→ Sect. 2.3), which ensures that multipliers have modulus  $\leq 1$  throughout forward elimination.



There is a class of transformation that avoids any amplification of relative errors (measured in Euclidean vector norm)!

Recall from linear algebra [27, Sect. 2.8]:

**Definition 2.8.5** (Unitary and orthogonal matrices).

- $Q \in \mathbb{K}^{n,n}$ ,  $n \in \mathbb{N}$ , is *unitary*, if  $Q^{-1} = Q^H$ .
- $Q \in \mathbb{R}^{n,n}$ ,  $n \in \mathbb{N}$ , is *orthogonal*, if  $Q^{-1} = Q^T$ .

**Theorem 2.8.6** (Criteria for Unitarity).

$$\mathbf{Q} \in \mathbb{C}^{n,n} \text{ unitary} \Leftrightarrow \|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2 \quad \forall \mathbf{x} \in \mathbb{K}^n.$$

$$\mathbf{Q} \text{ unitary} \Rightarrow \text{cond}(\mathbf{Q}) = 1$$



(2.8.2)

unitary transformations enhance (numerical) stability

If  $\mathbf{Q} \in \mathbb{K}^{n,n}$  unitary, then



- all rows/columns (regarded as vectors  $\in \mathbb{K}^n$ ) have Euclidean norm = 1,
- all rows/columns are pairwise orthogonal (w.r.t. Euclidean inner product),
- $|\det \mathbf{Q}| = 1$ , and all eigenvalues  $\in \{z \in \mathbb{C} : |z| = 1\}$ .
- $\|\mathbf{Q}\mathbf{A}\|_2 = \|\mathbf{A}\|_2$  for any matrix  $\mathbf{A} \in \mathbb{K}^{n,m}$

Drawbacks of  $LU$ -factorization:

-  often pivoting required ( $\rightarrow$  destroys structure, Ex. 2.6.32, leads to fill-in)
-  Possible (theoretical) instability of partial pivoting  $\rightarrow$  Ex. 2.5.14

Stability problems of Gaussian elimination without pivoting are due to the fact that row transformations can convert well-conditioned matrices to ill-conditioned matrices, *cf.* Ex. 2.5.14

Which bijective row transformations preserve the Euclidean condition number of a matrix ?

-  transformations that preserve the Euclidean norm of a vector !
-  Investigate algorithms that use orthogonal/unitary row transformations to convert a matrix to upper triangular form.

Goal: find unitary row transformation rendering certain matrix elements zero.

$$Q \begin{pmatrix} \text{yellow square} \end{pmatrix} = \begin{pmatrix} \text{white square with 0} \quad \text{yellow square} \end{pmatrix} \quad \text{with} \quad Q^H = Q^{-1}.$$

This “annihilation of column entries” is the key operation in Gaussian forward elimination, where it is achieved by means of non-unitary row transformations, see Sect. 2.2. Now we want to find a counterpart of Gaussian elimination based on unitary row transformations on behalf of numerical stability.

*Example 2.8.7* (“Annihilating” orthogonal transformations in 2D).

In 2D: two possible orthogonal transformations make 2nd component of  $\mathbf{a} \in \mathbb{R}^2$  vanish:

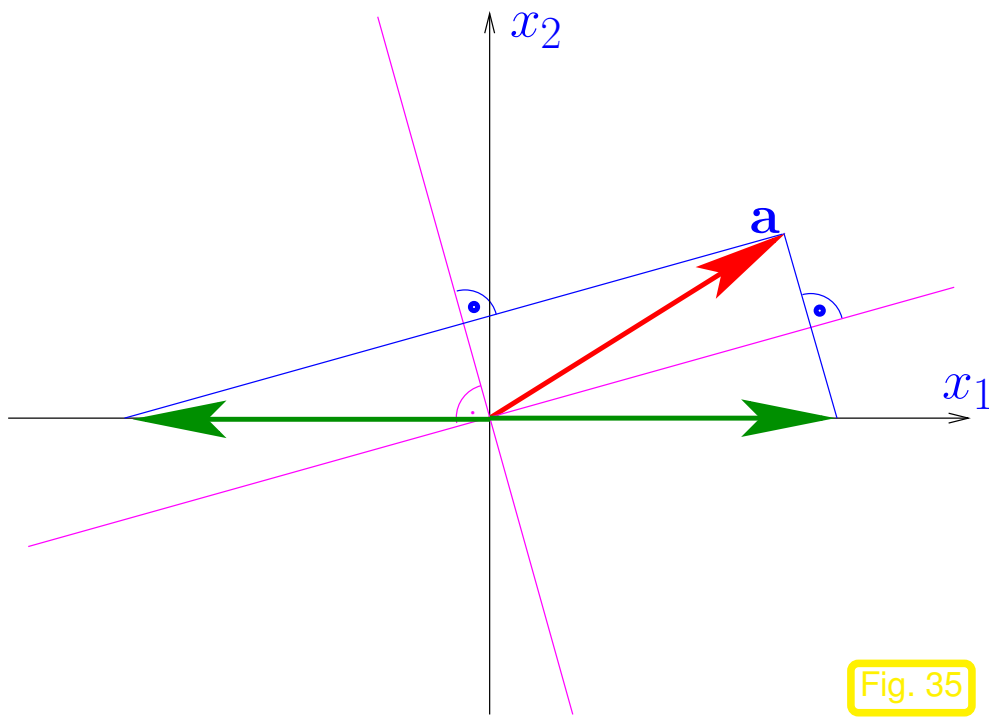


Fig. 35

reflection at angle bisector,

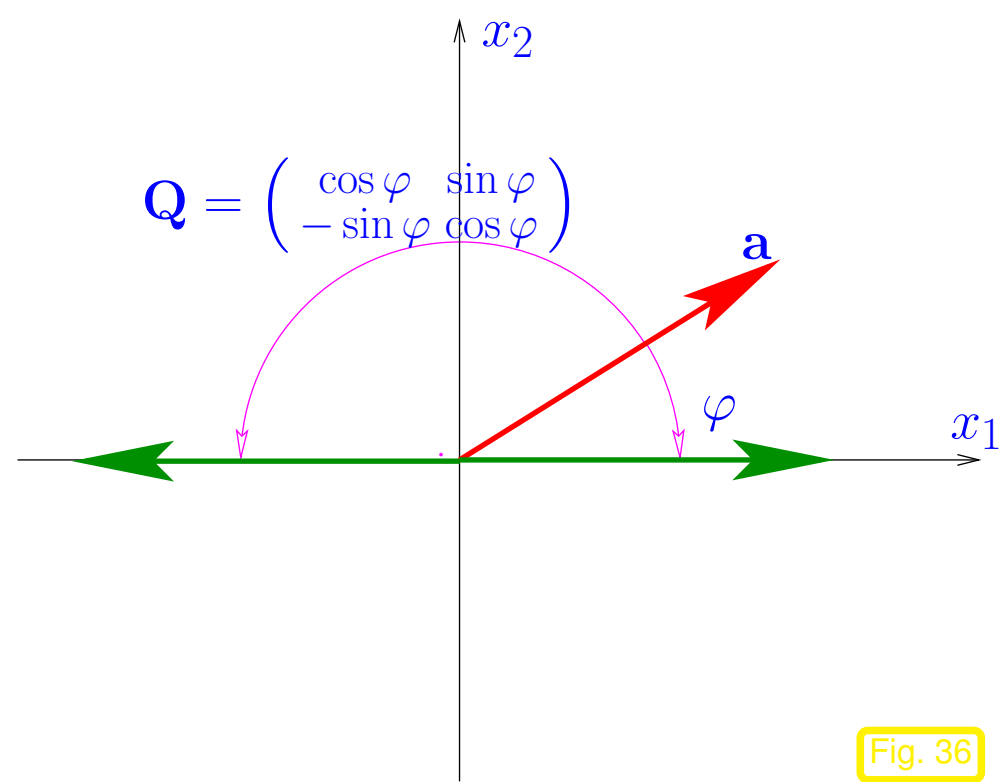


Fig. 36

rotation turning **a** onto  $x_1$ -axis.

$$Q = \begin{pmatrix} \cos \varphi & \sin \varphi \\ -\sin \varphi & \cos \varphi \end{pmatrix}$$

Note: two possible reflections/rotations



In  $n$ D: given  $\mathbf{a} \in \mathbb{R}^n$  find orthogonal matrix  $Q \in \mathbb{R}^{n,n}$  such that  $Q\mathbf{a} = \|\mathbf{a}\|_2 \mathbf{e}_1$ ,  $\mathbf{e}_1 \hat{=} 1$ st unit vector.

Choice 1: Householder reflections

$$Q = \mathbf{H}(\mathbf{v}) := \mathbf{I} - 2 \frac{\mathbf{v}\mathbf{v}^H}{\mathbf{v}^H\mathbf{v}} \quad \text{with} \quad \mathbf{v} = \frac{1}{2}(\mathbf{a} \pm \|\mathbf{a}\|_2 \mathbf{e}_1). \quad (2.8.8)$$

# “Geometric derivation” of Householder reflection, see Figure 35

Given  $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$  with  $\|\mathbf{a}\| = \|\mathbf{b}\|$ , the difference vector  $\mathbf{v} = \mathbf{b} - \mathbf{a}$  is orthogonal to the bisector.

$$\begin{aligned} \mathbf{b} &= \mathbf{a} - (\mathbf{a} - \mathbf{b}) = \mathbf{a} - \mathbf{v} \frac{\mathbf{v}^T \mathbf{v}}{\mathbf{v}^T \mathbf{v}} \\ &= \mathbf{a} - 2\mathbf{v} \frac{\mathbf{v}^T \mathbf{a}}{\mathbf{v}^T \mathbf{v}} = \mathbf{a} - 2 \frac{\mathbf{v} \mathbf{v}^T}{\mathbf{v}^T \mathbf{v}} \mathbf{a} = \mathbf{H}(\mathbf{v}) \mathbf{a}, \end{aligned}$$

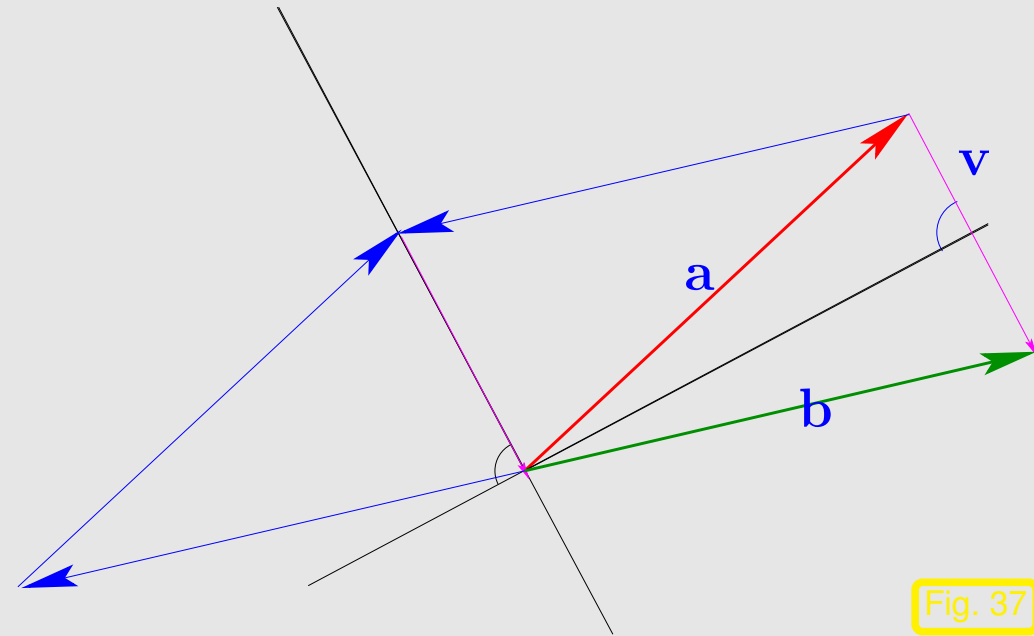


Fig. 37

because, due to orthogonality  $(\mathbf{a} - \mathbf{b}) \perp (\mathbf{a} + \mathbf{b})$

$$(\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b}) = (\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b} + \mathbf{a} + \mathbf{b}) = 2(\mathbf{a} - \mathbf{b})^T \mathbf{a}.$$

*Remark 2.8.9* (Details of Householder reflections).

- Practice: for the sake of numerical stability (in order to avoid so-called *cancellation*) choose

$$\mathbf{v} = \begin{cases} \frac{1}{2}(\mathbf{a} + \|\mathbf{a}\|_2 \mathbf{e}_1) & , \text{ if } a_1 > 0 , \\ \frac{1}{2}(\mathbf{a} - \|\mathbf{a}\|_2 \mathbf{e}_1) & , \text{ if } a_1 \leq 0 . \end{cases}$$

However, this is not really needed [36, Sect. 19.1] !

- If  $\mathbb{K} = \mathbb{C}$  and  $a_1 = |a_1| \exp(i\varphi)$ ,  $\varphi \in [0, 2\pi]$ , then choose

$$\mathbf{v} = \frac{1}{2}(\mathbf{a} \pm \|\mathbf{a}\|_2 \mathbf{e}_1 \exp(-i\varphi)) \quad \text{in (2.8.8).}$$

- efficient storage of Householder matrices  $\rightarrow$  [5]

Choice 2: successive Givens rotations [35, Sect. 14] ( $\rightarrow$  2D case)

$$\mathbf{G}_{1k}(a_1, a_k)\mathbf{a} := \begin{pmatrix} \bar{\gamma} & \cdots & \bar{\sigma} & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots \\ -\sigma & \cdots & \gamma & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_k \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_1^{(1)} \\ \vdots \\ 0 \\ \vdots \\ a_n \end{pmatrix}, \text{ if } \begin{cases} \gamma = \frac{a_1}{\sqrt{|a_1|^2 + |a_k|^2}}, \\ \sigma = \frac{a_k}{\sqrt{|a_1|^2 + |a_k|^2}}. \end{cases} \quad (2.8.10)$$

MATLAB-Function: `[G, x] = planerot(a);`

## Code 2.8.11: (plane) Givens rotation

```

1 function [G, x] = planerot(a)
2 % plane Givens rotation.
3 if (a(2) ~= 0)
4     r = norm(a); G = [a'; -a(2) a(1)]/r; x = [r; 0];
5 else, G = eye(2); end

```

Again, two options for annihilating rotation, see Ex. 2.8.7: either can be chosen and guarantees numerical stability.

So far, we know how to annihilate a single component of a vector by means of a Givens rotation that targets that component and some other (the first in (2.8.10)).



However, we aim to map *all* components to zero except for the first.

☞ This can be achieved by  $n - 1$  successive Givens rotations.

Mapping  $\mathbf{a} \in \mathbb{K}^n$  to a multiple of  $\mathbf{e}_1$  by  $n - 1$  successive Givens rotations:

$$\begin{pmatrix} a_1 \\ \vdots \\ \vdots \\ \vdots \\ a_n \end{pmatrix} \xrightarrow{\mathbf{G}_{12}(a_1, a_2)} \begin{pmatrix} a_1^{(1)} \\ 0 \\ a_3 \\ \vdots \\ a_n \end{pmatrix} \xrightarrow{\mathbf{G}_{13}(a_1^{(1)}, a_3)} \begin{pmatrix} a_1^{(2)} \\ 0 \\ 0 \\ a_4 \\ \vdots \\ a_n \end{pmatrix} \xrightarrow{\mathbf{G}_{14}(a_1^{(2)}, a_4)} \dots \xrightarrow{\mathbf{G}_{1n}(a_1^{(n-2)}, a_n)} \begin{pmatrix} a_1^{(n-1)} \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} \quad (2.8.12)$$

R. Hiptmair  
rev 38286,  
September  
7, 2011

☞ Notation:  $\mathbf{G}_{ij}(a_1, a_2) \hat{=}$  Givens rotation aimed at rows  $i$  and  $j$  of the matrix.

## Code 2.8.13: Transformation of a vector according to (2.8.12)

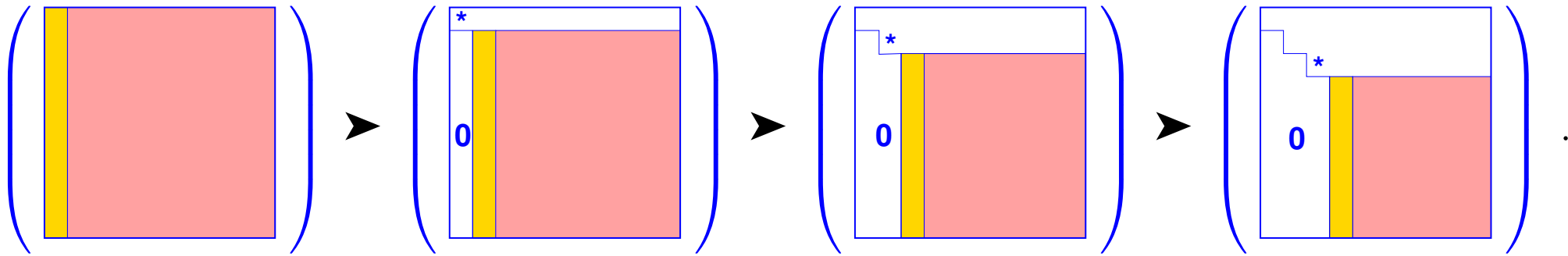
```
1 function [a,Q] = givenscoltrf(a)
2 % Orthogonal transformation of a (column) vector into a multiple of the
3 % first unit vector by successive Givens transformations
4 n=length(a);
5 Q=eye(n); % Assemble rotations in matrix, alternative see Rem. 2.8.21
6 for j=2:n
7     G=planerot([a(1);a(j)]); % see Code 2.8.10
8     a([1,j])=G*a([1,j]);
9     Q(:, [1,j])=Q(:, [1,j])*G';
10 end
```



Transformation to *upper triangular form* ( $\rightarrow$  Def. 2.2.3) by successive unitary transformations:

We may use either Householder reflections or successive Givens rotations as explained above.

Visualization of annihilation of lower triangular part of a matrix by suitable successive Householder reflections (2.8.8):



 = "target column **a**" (determines unitary transformation),

 = modified in course of transformations.



$$Q_{n-1} Q_{n-2} \cdots Q_1 A = R,$$

**QR-factorization**  
(QR-decomposition)

of  $A \in \mathbb{C}^{n,n}$ :  $A = QR$ ,  $Q := Q_1^H \cdots Q_{n-1}^H$  unitary matrix,  
 $R$  upper triangular matrix.

The same can be achieved by suitable successive Givens rotations:

## Code 2.8.14: QR decomposition by successive Givens rotations

```
1 function [Q,A] = qrgivens(A)
2 % in situ QR decomposition of square matrix A, same semantics as
3 % MATLAB built qr() function
4 n=size(A,1);
5 Q=eye(n); % Assemble rotations in matrix, alternative see Rem. 2.8.21
6 for i=1:(n-1)
7     for j=n:-1:(i+1)
8         G=planerot([A(j-1,i);A(j,i)]); % see Code 2.8.10
9         A([j-1,j],:)=G*A([j-1,j],:);
10        Q(:,[j-1,j])=Q(:,[j-1,j])*G';
11    end
12 end
```

➡ Obvious asymptotic computational complexity:  $O(n^3)$  (for  $\mathbf{A} \in \mathbb{R}^{n,n}$ )

Generalization to  $\mathbf{A} \in \mathbb{K}^{m,n}$ :

$$m > n: \quad \left( \begin{array}{c} \mathbf{A} \end{array} \right) = \left( \begin{array}{c} \mathbf{Q} \end{array} \right) \left( \begin{array}{c} \mathbf{R} \end{array} \right), \quad \mathbf{A} = \mathbf{QR}, \quad \begin{array}{l} \mathbf{Q} \in \mathbb{K}^{m,n}, \\ \mathbf{R} \in \mathbb{K}^{n,n}, \end{array}$$

(2.8.15)

where  $\mathbf{Q}^H \mathbf{Q} = \mathbf{I}$  (orthonormal columns),  $\mathbf{R}$  upper triangular matrix.

**Lemma 2.8.16** (Uniqueness of QR-factorization).

The “economical” QR-factorization (2.8.15) of  $\mathbf{A} \in \mathbb{K}^{m,n}$ ,  $m \geq n$ , with  $\text{rank}(\mathbf{A}) = n$  is unique, if we demand  $r_{ii} > 0$ .

R. Hiptmair  
rev 38286,  
September  
7, 2011

*Proof.* we observe that  $\mathbf{R}$  is regular, if  $\mathbf{A}$  has full rank  $n$ . Since the regular upper triangular matrices form a group under multiplication:

$$\mathbf{Q}_1 \mathbf{R}_1 = \mathbf{Q}_2 \mathbf{R}_2 \quad \Rightarrow \quad \mathbf{Q}_1 = \mathbf{Q}_2 \mathbf{R} \quad \text{with upper triangular } \mathbf{R} := \mathbf{R}_2 \mathbf{R}_1^{-1}.$$

$$\blacktriangleright \quad \mathbf{I} = \mathbf{Q}_1^H \mathbf{Q}_1 = \mathbf{R}^H \underbrace{\mathbf{Q}_2^H \mathbf{Q}_2}_{=\mathbf{I}} \mathbf{R} = \mathbf{R}^H \mathbf{R} .$$

The assertion follows by uniqueness of Cholesky decomposition, Lemma 2.7.14. □

$$m < n: \quad \left( \begin{array}{|c|} \hline \mathbf{A} \\ \hline \end{array} \right) = \left( \begin{array}{|c|} \hline \mathbf{Q} \\ \hline \end{array} \right) \left( \begin{array}{|c|} \hline \mathbf{R} \\ \hline \end{array} \right) ,$$

$$\mathbf{A} = \mathbf{QR} \quad , \quad \mathbf{Q} \in \mathbb{K}^{m,m} \quad , \quad \mathbf{R} \in \mathbb{K}^{m,n} \quad ,$$

where **Q** unitary, **R** upper triangular matrix.

*Remark 2.8.17* (Choice of unitary/orthogonal transformation).

When to use which unitary/orthogonal transformation for QR-factorization ?

$\blacktriangleright$  Householder reflections advantageous for fully populated target columns (dense matrices).



MATLAB functions:

$$[Q, R] = \text{qr}(A) \quad \mathbf{Q} \in \mathbb{K}^{m,m}, \mathbf{R} \in \mathbb{K}^{m,n} \text{ for } \mathbf{A} \in \mathbb{K}^{m,n}$$

$$[Q, R] = \text{qr}(A, 0) \quad \mathbf{Q} \in \mathbb{K}^{m,n}, \mathbf{R} \in \mathbb{K}^{n,n} \text{ for } \mathbf{A} \in \mathbb{K}^{m,n}, m > n$$

(*economical* QR-factorization)

Computational effort for Householder QR-factorization of  $\mathbf{A} \in \mathbb{K}^{m,n}, m > n$ :

$$[Q, R] = \text{qr}(A) \quad \rightarrow \text{Costs: } O(m^2n)$$

$$[Q, R] = \text{qr}(A, 0) \quad \rightarrow \text{Costs: } O(mn^2)$$

*Example 2.8.18* (Complexity of Householder QR-factorization).

Code 2.8.19: timing MATLAB QR-factorizations

```
1 % Timing QR factorizations
2
3 K = 3; r = [];
```

```
4 for n=2.^(2:6)
5     m = n*n;
6
7     A = (1:m)'*(1:n) + [eye(n);ones(m-n,n)];
8     t1 = 1000; for k=1:K, tic; [Q,R] = qr(A); t1 = min(t1,toc);
9         clear Q,R; end
10    t2 = 1000; for k=1:K, tic; [Q,R] = qr(A,0); t2 = min(t2,toc);
11        clear Q,R; end
12    t3 = 1000; for k=1:K, tic; R = qr(A); t3 = min(t3,toc); clear
13        R; end
14    r = [r; n , m , t1 , t2 , t3];
15 end
```



tic-toc-timing of different variants of QR-factorization in MATLAB

► Use  $[Q, R] = \text{qr}(A, 0)$ , if output sufficient!

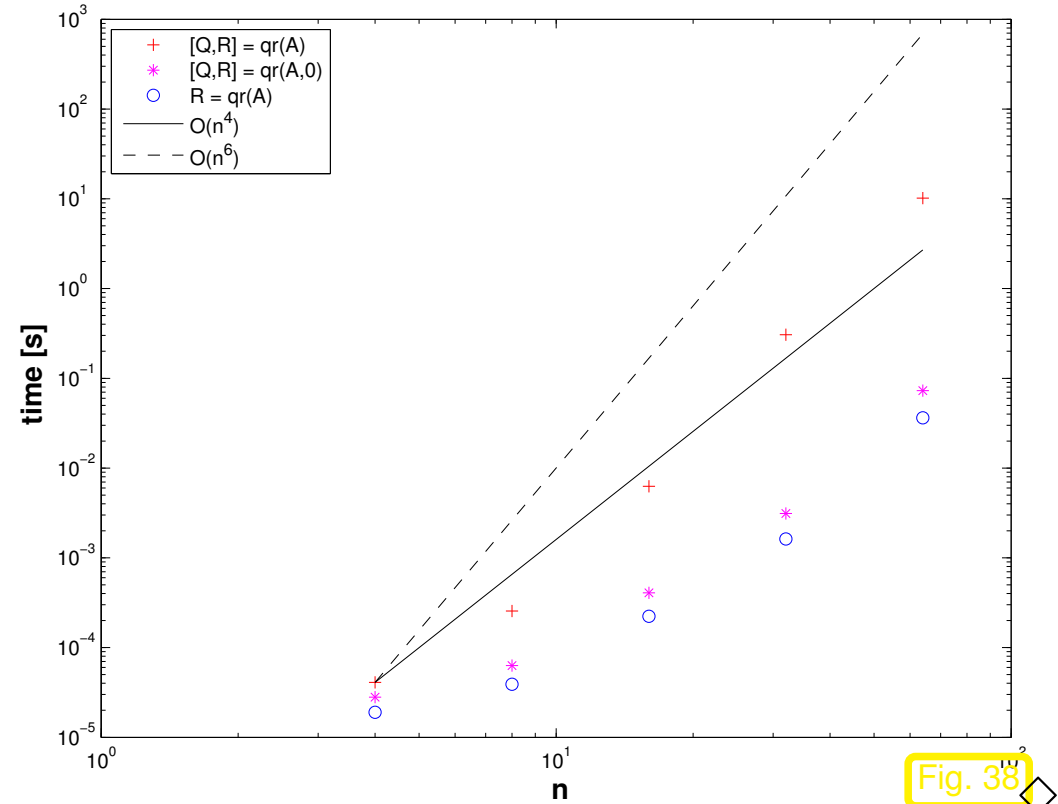


Fig. 38

Remark 2.8.20 (QR-orthogonalization).

$$\begin{pmatrix} \text{A} \end{pmatrix} = \begin{pmatrix} \text{Q} \end{pmatrix} \begin{pmatrix} \text{R} \end{pmatrix}, \quad \text{A}, \text{Q} \in \mathbb{K}^{m,n}, \text{R} \in \mathbb{K}^{n,n}.$$

If  $m > n$ ,  $\text{rank}(\mathbf{R}) = \text{rank}(\mathbf{A}) = n$  (full rank)

➤  $\{\mathbf{q}_{\cdot,1}, \dots, \mathbf{q}_{\cdot,n}\}$  is **orthonormal basis** of  $\text{Im}(\mathbf{A})$  with  
 $\text{Span}\{\mathbf{q}_{\cdot,1}, \dots, \mathbf{q}_{\cdot,k}\} = \text{Span}\{\mathbf{a}_{\cdot,1}, \dots, \mathbf{a}_{\cdot,k}\}, 1 \leq k \leq n$ .



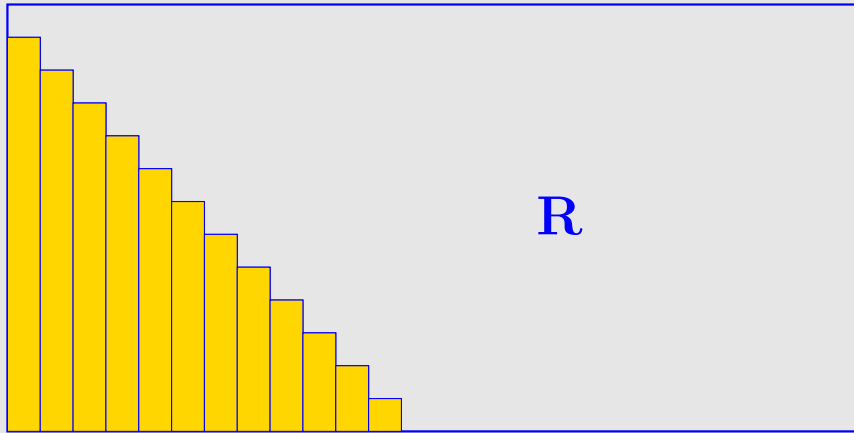
*Remark 2.8.21* (Keeping track of unitary transformations).

How to store  $\mathbf{G}_{i_1 j_1}(a_1, b_1) \cdots \mathbf{G}_{i_k j_k}(a_k, b_k), \quad ?$   
 $\mathbf{H}(\mathbf{v}_1) \cdots \mathbf{H}(\mathbf{v}_k)$

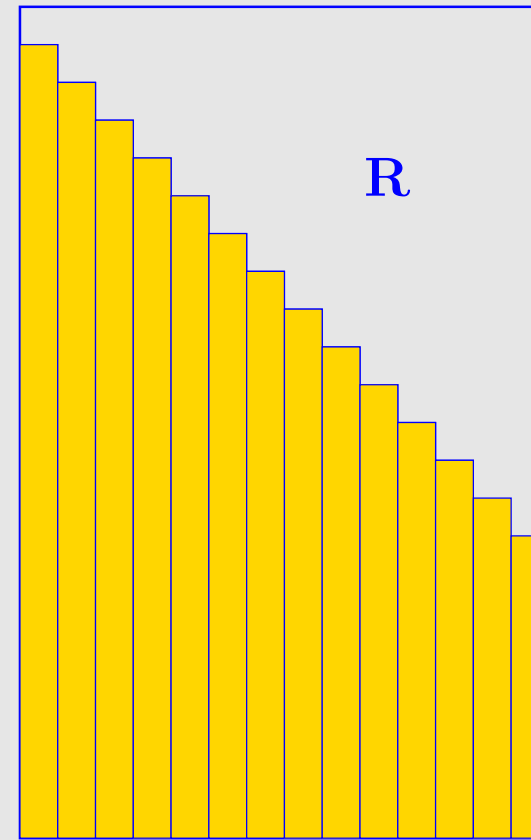
☛ For Householder reflections

$\mathbf{H}(\mathbf{v}_1) \cdots \mathbf{H}(\mathbf{v}_k)$ : store  $\mathbf{v}_1, \dots, \mathbf{v}_k$


For in place QR-factorization of  $\mathbf{A} \in \mathbb{K}^{m,n}$ : store "Householder vectors"  $\mathbf{v}_j$  (decreasing size  
 !) in lower triangle of  $\mathbf{A}$



↑ Case  $m < n$



Case  $m > n \rightarrow$

 = Householder vectors

☛ Convention for Givens rotations ( $\mathbb{K} = \mathbb{R}$ )

$$\mathbf{G} = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \Rightarrow \text{store } \rho := \begin{cases} 1 & , \text{ if } \gamma = 0 , \\ \frac{1}{2} \text{sign}(\gamma)\sigma & , \text{ if } |\sigma| < |\gamma| , \\ 2 \text{sign}(\sigma)/\gamma & , \text{ if } |\sigma| \geq |\gamma| . \end{cases}$$

$$\blacktriangleright \begin{cases} \rho = 1 & \Rightarrow \gamma = 0 , \quad \sigma = 1 \\ |\rho| < 1 & \Rightarrow \sigma = 2\rho , \quad \gamma = \sqrt{1 - \sigma^2} \\ |\rho| > 1 & \Rightarrow \gamma = 2/\rho , \quad \sigma = \sqrt{1 - \gamma^2} . \end{cases}$$

Then store  $\mathbf{G}_{ij}(a, b)$  as triple  $(i, j, \rho)$ . The parameter  $\rho$  forgets the sign of the matrix  $\mathbf{G}_{ij}$ , so the signs of the corresponding rows in the transformed matrix  $\mathbf{R}$  have to be changed accordingly.

The rationale for this convention is to curb the impact of roundoff errors.

Storing orthogonal transformation matrices is usually inefficient !

*Algorithm 2.8.22* (Solving linear system of equations by means of QR-decomposition).

① QR-decomposition  $\mathbf{A} = \mathbf{QR}$ , computational costs  $\frac{2}{3}n^3 + O(n^2)$  (about twice as expensive as *LU*-decomposition without pivoting)

$\mathbf{Ax} = \mathbf{b}$  : ② orthogonal transformation  $\mathbf{z} = \mathbf{Q}^H \mathbf{b}$ , computational costs  $4n^2 + O(n)$  (in the case of *compact storage* of reflections/rotations)

③ **Backward substitution**, solve  $\mathbf{Rx} = \mathbf{z}$ , computational costs  $\frac{1}{2}n(n + 1)$

- ✌ Computing the generalized QR-decomposition  $A = QR$  by means of Householder reflections or Givens rotations is (numerically stable) for any  $A \in \mathbb{C}^{m,n}$ .
- ✌ For *any* regular system matrix an LSE can be solved by means of  
 QR-decomposition + orthogonal transformation + backward substitution  
 in a stable manner.

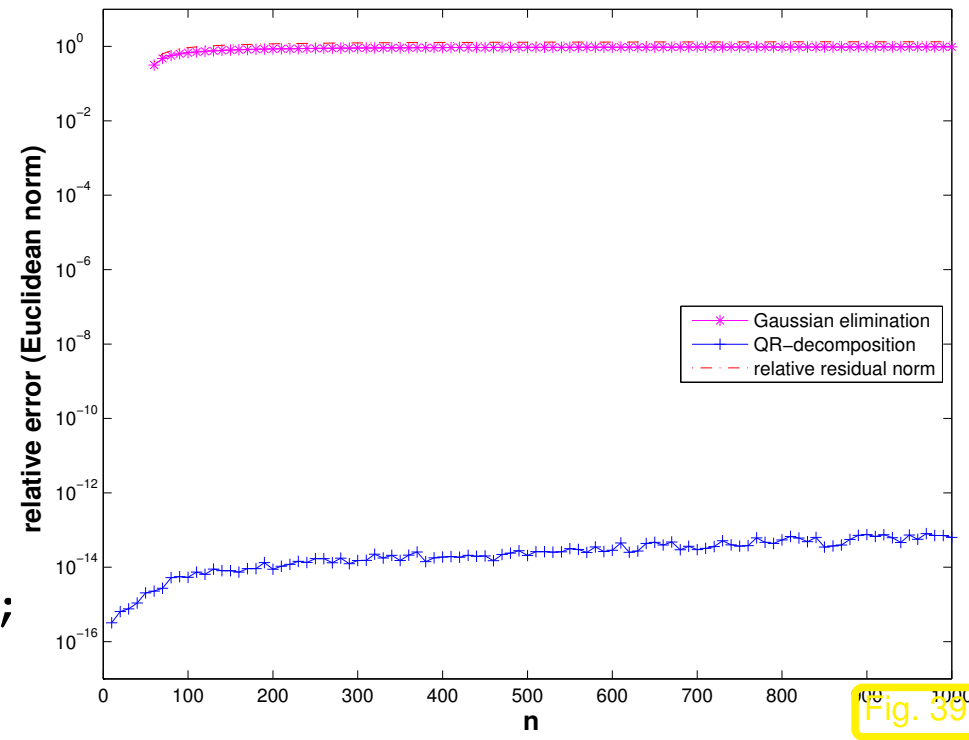
Example 2.8.23 (Stable solution of LSE by means of QR-decomposition). → Ex. 2.5.14

Code 2.8.24: QR-fac. ↔ Gaussian elimination

```

1 res = [];
2 for n=10:10:1000
3     A=[ tril (-ones (n, n-1)) ...
4         +2* [ eye (n-1) ; ...
5             zeros (1, n-1) ], ones (n, 1) ];
6     x= (-1) .^ (1:n)';
7     b=A*x;
8     [Q,R]=qr (A) ;
9     errlu=norm (A\b-x) / norm (x) ;
10    errqr=norm (R\ (Q' *b) -x) / norm (x) ;
11    res=[res; n, errlu, errqr];
12 end
13 semilogy (res (:, 1), res (:, 2), 'm-*',
14            res (:, 1), res (:, 3), 'b-+') ;

```



superior stability of QR-decomposition !

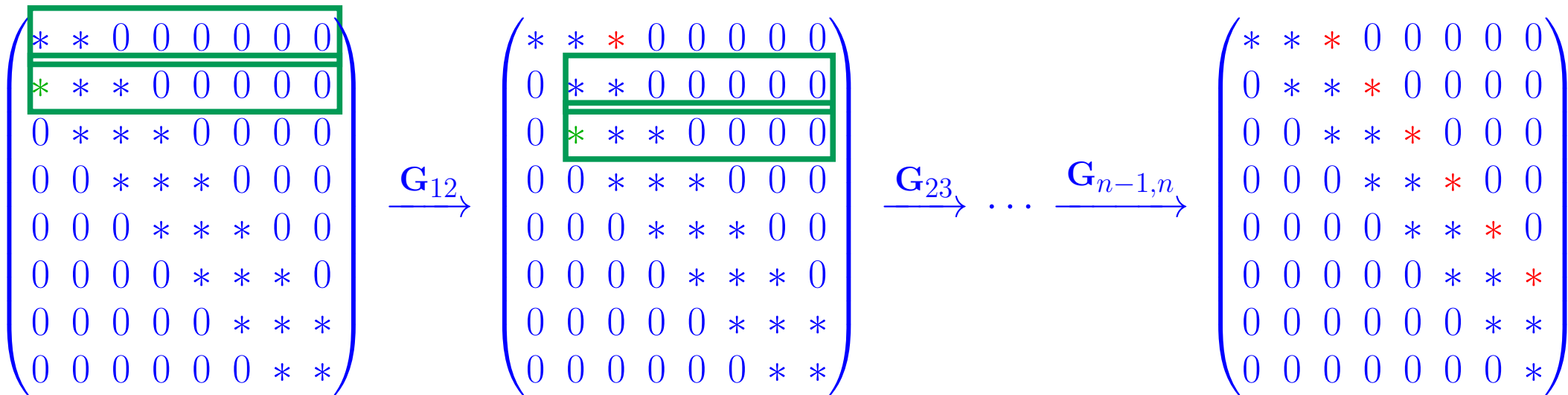
# Fill-in for QR-decomposition ?

bandwidth

$$\mathbf{A} \in \mathbb{C}^{n,n} \text{ with QR-decomposition } \mathbf{A} = \mathbf{QR} \Rightarrow m(\mathbf{R}) \leq m(\mathbf{A}) \quad (\rightarrow \text{Def. 2.6.34})$$

Example 2.8.25 (QR-based solution of tridiagonal LSE).

Elimination of Sub-diagonals by  $n - 1$  successive Givens rotations:



MATLAB code ( $c, d, e, b$  = column vectors of length  $n, n \in \mathbb{N}, e(n), c(n)$  not used):

$$\mathbf{A} = \begin{pmatrix} d_1 & c_1 & 0 & \dots & 0 \\ e_1 & d_2 & c_2 & & \vdots \\ 0 & e_2 & d_3 & c_3 & \\ \vdots & \dots & \dots & \dots & c_{n-1} \\ 0 & \dots & 0 & e_{n-1} & d_n \end{pmatrix} \Leftrightarrow \text{spdiags}([e, d, c], [-1 \ 0 \ 1], n, n)$$

```

1 function y = tridiaggqr(c,d,e,b)
2 n = length(d); t = norm(d)+norm(e)+norm(c);
3 for k=1:n-1
4     [R,z] = planerot([d(k);e(k)]);
5     if (abs(z(1))/t < eps), error('Matrix singular'); end;
6     d(k) = z(1); b(k:k+1) = R*b(k:k+1);
7     Z = R*[c(k), 0;d(k+1), c(k+1)];
8     c(k) = Z(1,1); d(k+1) = Z(2,1);
9     e(k) = Z(1,2); c(k+1) = Z(2,2);
10 end
11 A = spdiags([d, [0;c(1:end-1)], [0;0;e(1:end-2)]], [0 1 2], n, n);
12 y = A\b;

```

Asymptotic complexity  $O(n)$



**Remark 2.8.27** (Storing the **Q**-factor).

The previous example (Code 2.8.25) showed that assembly of the **Q**-factor in the QR-factorization of **A** is *not needed*, when the linear system of equations  $\mathbf{Ax} = \mathbf{b}$  is to be solved by means of QR-factorization: the orthogonal transformations can simply be applied to the right hand side(s) whenever they are applied to the columns of **A**.

Discussion of Rem. 2.2.13 for QR-factorization:

Inefficient (!) code

- $\mathbf{Q} \in \mathbb{R}^{n,n}$  dense matrix

- $\mathbf{R} \in \mathbb{R}^{n,n}$  dense matrix

➤  $O(n^2)$  computational effort for executing loop body

Remedies:

- Store **R** in sparse matrix format, see Code 2.8.25, Sect. 2.6.2.

```

1 % Setting: N >> 1,
2 % large tridiagonal matrix A ∈ ℝn,n
3 [Q,R] = qr(A);
4 for j=1:N
5     x = R \ (Q' * b);
6     b = some_function(x);
7 end

```



- Store Givens rotations contained in  $\mathbf{Q}$  as array of triplets:  $G_{lk}$  is coded as

$$[l, k, \text{rho}] ,$$

where  $\text{rho} (\hat{=} \rho)$  is chosen as in Rem. 2.8.21.



*Remark 2.8.28* (Testing for near singularity of a matrix).

Very small (w.r.t. matrix norm) element  $r_{ii}$  in QR-factor  $\mathbf{R} \leftrightarrow \mathbf{A}$  “nearly singular”



## 2.9 Modification Techniques

In quite a few applications one has to solve many linear systems, whose matrices can be transformed into each other by small changes, however.

Issue: can we use this information to solve the LSEs more efficiently

$\iff$  if  $\tilde{\mathbf{A}}$  arises from  $\mathbf{A}$  through a “small alteration” (modification), can we reuse the Cholesky/LU-/QR-factorization of  $\mathbf{A}$  to obtain that of  $\tilde{\mathbf{A}}$  with less effort than computing them anew.

This section will present algorithms that can accomplish this.

*Example 2.9.1* (Resistance to currents map).

Large (linear) electric circuit (modelling  
→ Ex. 2.6.3) ▷

Sought:

Dependence of (certain) branch currents on “continuously varying” resistance  $R_x$

(▷ currents for many different values of  $R_x$ )

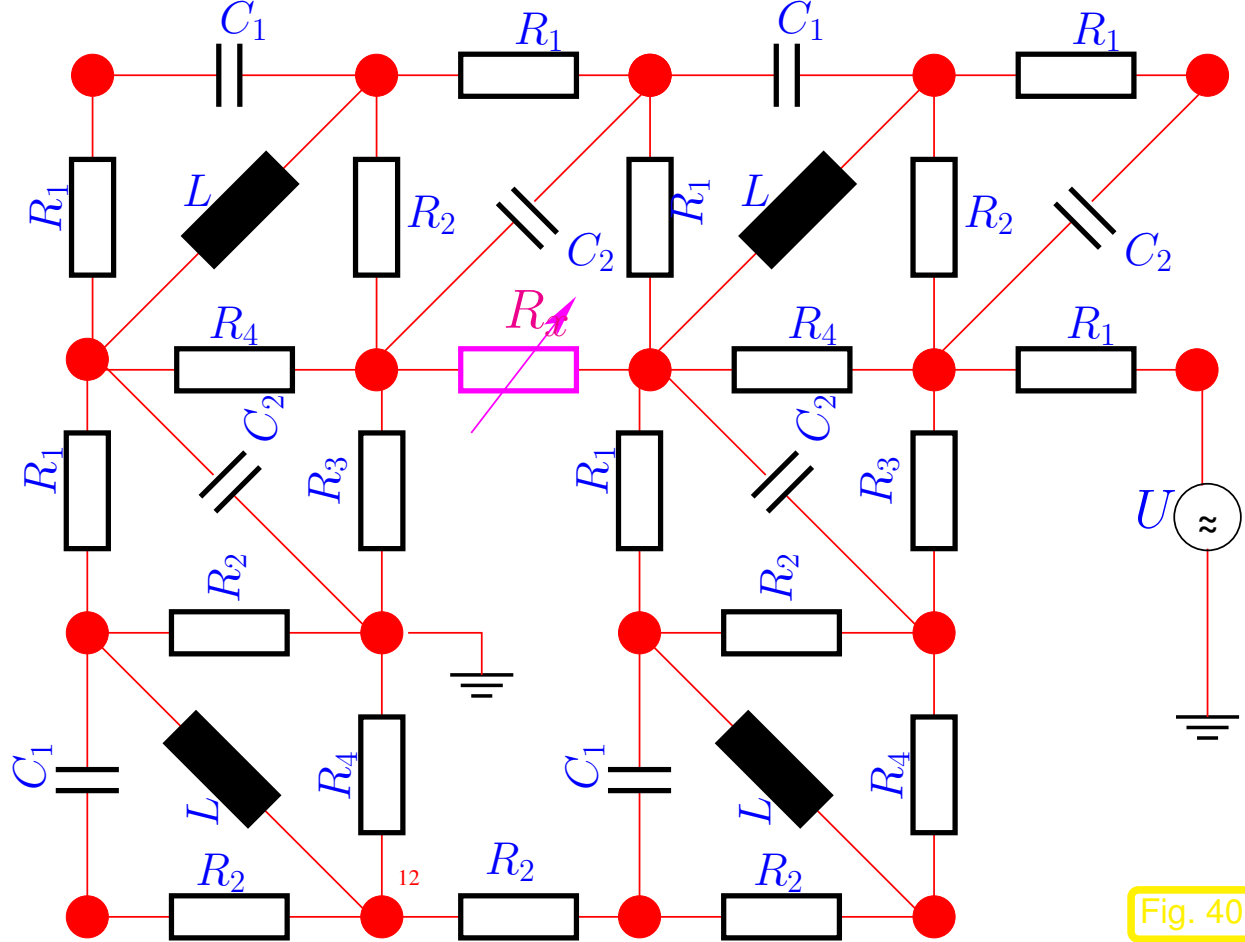


Fig. 40

▶ Only a few entries of the nodal analysis matrix  $\mathbf{A}$  (→ Ex. 2.6.3) are affected by variation of  $R_x$ !  
(If  $R_x$  connects nodes  $i$  &  $j$  ⇒ only entries  $a_{ii}, a_{jj}, a_{ij}, a_{ji}$  of  $\mathbf{A}$  depend on  $R_x$ )

▶ Repeating Gaussian elimination/LU-factorization for each value of  $R_x$  from scratch seems wasteful.

- Idea:
- compute (sparse) LU-factorization of  $\mathbf{A}$  *once*
  - Repeat: **update** LU-factors for modified  $\mathbf{A}$

+

(partial) forward and backward substitution



Problem: Efficient *update* of matrix factorizations in the case of ‘slight’ changes of the matrix [23, Sect. 12.6], [61, Sect. 4.9].

### 2.9.0.1 Rank-1-modifications

*Example 2.9.2* (Changing entries/rows/columns of a matrix).

Changing a single entry: given  $x \in \mathbb{K}$

$$\mathbf{A}, \tilde{\mathbf{A}} \in \mathbb{K}^{n,n}: \tilde{a}_{ij} = \begin{cases} a_{ij} & , \text{ if } (i, j) \neq (i^*, j^*), \\ x + a_{ij} & , \text{ if } (i, j) = (i^*, j^*), \end{cases} \quad , \quad i^*, j^* \in \{1, \dots, n\} . \quad (2.9.3)$$

$$\blacktriangleright \quad \boxed{\tilde{\mathbf{A}} = \mathbf{A} + x \cdot \mathbf{e}_{i^*} \mathbf{e}_{j^*}^T} . \quad (2.9.4)$$

Recall:  $\mathbf{e}_i \hat{=} i$ -th unit vector

Changing a single row: given  $\mathbf{x} \in \mathbb{K}^n$

$$\mathbf{A}, \tilde{\mathbf{A}} \in \mathbb{K}^{n,n}: \quad \tilde{a}_{ij} = \begin{cases} a_{ij} & , \text{ if } i \neq i^* , \\ x_j + a_{ij} & , \text{ if } i = i^* , \end{cases} \quad , \quad i^*, j^* \in \{1, \dots, n\} .$$

$$\blacktriangleright \quad \boxed{\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{e}_{i^*} \mathbf{x}^T} . \quad (2.9.5)$$

Both matrix modifications (2.9.3) and (2.9.5) are specimens of a rank-1-modifications.



$$\mathbf{A} \in \mathbb{K}^{n,n} \quad \mapsto \quad \tilde{\mathbf{A}} := \mathbf{A} + \boxed{\mathbf{u} \mathbf{v}^H} , \quad \mathbf{u}, \mathbf{v} \in \mathbb{K}^n \quad (2.9.6)$$

general rank-1-matrix

*Remark 2.9.7* (Solving LSE in the case of rank-1-modification).

*Lemma 2.9.8* (Sherman-Morrison-Woodbury formula).

For regular  $\mathbf{A} \in \mathbb{K}^{n,n}$ , and  $\mathbf{U}, \mathbf{V} \in \mathbb{K}^{n,k}$ ,  $n, k \in \mathbb{N}$ ,  $k \leq n$ , holds

$$(\mathbf{A} + \mathbf{U}\mathbf{V}^H)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^H\mathbf{A}^{-1},$$

if  $\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U}$  regular.

*Proof.* Straightforward algebra:

$$\begin{aligned} \left( \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^H\mathbf{A}^{-1} \right) (\mathbf{A} + \mathbf{U}\mathbf{V}^H) = \\ \mathbf{I} - \mathbf{A}^{-1}\mathbf{U} \underbrace{(\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})^{-1}(\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})}_{=\mathbf{I}} \mathbf{V}^H + \mathbf{A}^{-1}\mathbf{U}\mathbf{V}^H = \mathbf{I}. \end{aligned}$$

Uniqueness of the inverse settles the case. □

Task: Solve  $\tilde{\mathbf{A}}\mathbf{x} = \mathbf{b}$ , when LU-factorization  $\mathbf{A} = \mathbf{L}\mathbf{U}$  already known

Apply Lemma 2.9.8 for  $k = 1$ :

$$\mathbf{x} = \left( \mathbf{I} - \frac{\mathbf{A}^{-1} \mathbf{u} \mathbf{v}^H}{1 + \mathbf{v}^H \mathbf{A}^{-1} \mathbf{u}} \right) \mathbf{A}^{-1} \mathbf{b}.$$

Efficient implementation !

Asymptotic complexity  $O(n^2)$   
(back substitutions)



### Code 2.9.9: solving a rank-1 modified LSE

```

1 function x = smw(L,U,u,v,b)
2 t = L\b; z = U\t;
3 t = L\u; w = U\t;
4 alpha = 1+dot(v,w);
5 if (abs(alpha) <
      eps*norm(U,1)), error('Nearly
      singular matrix'); end;
6 x = z - w*dot(v,z)/alpha;

```



The approach of Rem. 2.9.7 is certainly efficient, but may *suffer from instability* similar to Gaussian elimination without pivoting, cf. Ex. 2.3.1.

This can be avoided by using QR-factorization (→ Sect. 2.8) and corresponding update techniques. This was a key rationale for studying QR-factorization for the solution of linear system of equations.

Other important applications of QR-factorization will be discussed later in Chapter 7.

Task: Efficient computation of QR-factorization ( $\rightarrow$  Sect. 2.8)  $\tilde{\mathbf{A}} = \tilde{\mathbf{Q}}\tilde{\mathbf{R}}$  of  $\tilde{\mathbf{A}}$  from (2.9.6), when QR-factorization  $\mathbf{A} = \mathbf{Q}\mathbf{R}$  already known

① With  $\mathbf{w} := \mathbf{Q}^H \mathbf{u}$ :  $\mathbf{A} + \mathbf{u}\mathbf{v}^H = \mathbf{Q}(\mathbf{R} + \mathbf{w}\mathbf{v}^H)$

$\rightarrow$  Asymptotic complexity  $O(n^2)$  (depends on how  $\mathbf{Q}$  is stored  $\rightarrow$  Rem. 2.8.21)

② Objective:  $\mathbf{w} \rightarrow \|\mathbf{w}\| \mathbf{e}_1$   $\rightarrow$  via  $n - 1$  Givens rotations, see (2.8.10).

$$\mathbf{w} = \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ * \\ * \\ * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-1,n}} \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ * \\ * \\ 0 \end{pmatrix} \xrightarrow{\mathbf{G}_{n-2,n-1}} \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ 0 \\ 0 \end{pmatrix} \xrightarrow{\mathbf{G}_{n-3,n-2}} \dots \xrightarrow{\mathbf{G}_{1,2}} \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (2.9.10)$$

Note the difference between this arrangement of successive Givens rotations to turn  $\mathbf{w}$  into a multiple of the first unit vector  $\mathbf{e}_1$ , and the different sequence of Givens rotations discussed in Sect. 2.8. Both serve the same purpose, but we shall see in a moment that the smart selection of Givens rotations is crucial in the current context.



Note: rotations affect **R** !

$$\begin{aligned}
 \mathbf{R} = & \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-1,n}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ \vdots & \ddots & & & & & \vdots \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-2,n-1}} \\
 & \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-3,n-2}} \cdots \xrightarrow{\mathbf{G}_{1,2}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ * & * & \cdots & * & * & * & * \\ & & \ddots & & & & \vdots \\ 0 & \cdots & * & * & * & * & * \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} =: \mathbf{R}_1
 \end{aligned}$$

upper Hessenberg matrix: Entry  $(i, j) = 0$ , if  $i > j + 1$ .

►  $\mathbf{A} + \mathbf{u}\mathbf{v}^H = \mathbf{Q}\mathbf{Q}_1^H \left( \underbrace{\mathbf{R}_1 + \|\mathbf{w}\|_2 \mathbf{e}_1 \mathbf{v}^H}_{\text{upper Hessenberg matrix}} \right)$  with unitary  $\mathbf{Q}_1 := \mathbf{G}_{12} \cdots \mathbf{G}_{n-1,n}$ .

➔ Asymptotic complexity  $O(n^2)$

Imagine that in (2.9.10) we had chosen to annihilate the components  $2, \dots, n$  of  $\mathbf{w}$  by the product of Givens rotations  $\mathbf{G}_{12}\mathbf{G}_{13}\cdots\mathbf{G}_{1,n-1}$ . This would have resulted in a fully populated matrix  $\mathbf{R}_1$ !

In this case, the next step could be carried out with an effort  $O(n^3)$  only.

③ Successive Givens rotations:  $\mathbf{R}_1 + \|\mathbf{w}\|_2 \mathbf{e}_1 \mathbf{v}^H \mapsto$  upper triangular form

$$\begin{aligned}
 \mathbf{R}_1 + \|\mathbf{w}\|_2 \mathbf{e}_1 \mathbf{v}^H &= \begin{pmatrix} * & * & \cdots & * & * & * & * \\ * & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & * & * & * & * & * \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{12}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & * & * & * & * & * \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{23}} \cdots \\
 &\xrightarrow{\mathbf{G}_{n-2,n-1}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-1,n}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \end{pmatrix} =: \tilde{\mathbf{R}}. \quad (2.9.11)
 \end{aligned}$$

➔ Asymptotic complexity  $O(n^2)$



$$\mathbf{A} + \mathbf{u}\mathbf{v}^H = \tilde{\mathbf{Q}}\tilde{\mathbf{R}} \quad \text{mit } \tilde{\mathbf{Q}} = \mathbf{Q}\mathbf{Q}_1^H \mathbf{G}_{n-1,n}^H \cdots \mathbf{G}_{12}^H.$$

MATLAB-function: `[Q1,R1] = qrupdate(Q,R,u,v);`

Special case: rank-1-modifications preserving symmetry & *positivity* ( $\rightarrow$  Def. 2.7.9):

$$\mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n} \quad \mapsto \quad \tilde{\mathbf{A}} := \mathbf{A} + \alpha \mathbf{v}\mathbf{v}^H, \quad \mathbf{v} \in \mathbb{K}^n, \alpha > 0. \quad (2.9.12)$$

If the modified matrix is known to be s.p.d. ➤ Cholesky factorization will be stable. Thus, efficient modification of the Cholesky factor is of practical relevance.

Task: Efficient computation of Cholesky factorization  $\tilde{\mathbf{A}} = \tilde{\mathbf{R}}^H \tilde{\mathbf{R}}$  ( $\rightarrow$  Lemma 2.7.14) of  $\tilde{\mathbf{A}}$  from (2.9.12), when Cholesky factorization  $\mathbf{A} = \mathbf{R}^H \mathbf{R}$  of  $\mathbf{A}$  already known

With  $\mathbf{w} := \mathbf{R}^{-H} \mathbf{v}$ :  $\mathbf{A} + \alpha \mathbf{v} \mathbf{v}^H = \mathbf{R}^H (\mathbf{I} + \alpha \mathbf{w} \mathbf{w}^H) \mathbf{R}$ .

➔ Asymptotic complexity  $O(n^2)$  (backward substitution !)

② Idea: formal Gaussian elimination: with  $\tilde{\mathbf{w}} = (w_2, \dots, w_n)^T \rightarrow$  see (2.1.11)

$$\mathbf{I} + \alpha \mathbf{w} \mathbf{w}^H = \left( \begin{array}{c|c} 1 + \alpha w_1^2 & \alpha w_1 \tilde{\mathbf{w}}^H \\ \hline \alpha w_1 \tilde{\mathbf{w}} & \mathbf{I} + \alpha \tilde{\mathbf{w}} \tilde{\mathbf{w}}^H \end{array} \right) \rightarrow \left( \begin{array}{c|c} 1 + \alpha w_1^2 & \alpha w_1 \tilde{\mathbf{w}}^H \\ \hline 0 & \mathbf{I} + \alpha^{(1)} \tilde{\mathbf{w}} \tilde{\mathbf{w}}^H \end{array} \right) \quad (2.9.13)$$

where  $\alpha^{(1)} := \alpha - \frac{\alpha^2 w_1^2}{1 + \alpha w_1^2}$ .

same structure ➤ recursion

Proceeding in this way, we obtain the  $\mathbf{U}$  matrix of the LU-decomposition  $\mathbf{I} + \alpha \mathbf{w} \mathbf{w}^H = \mathbf{L} \mathbf{U}$ . In order to obtain the Cholesky decomposition we divide the lines of  $\mathbf{U}$  by the square root of its diagonal terms, i.e.,  $\mathbf{I} + \alpha \mathbf{w} \mathbf{w}^H = \mathbf{L} \mathbf{U} = (\mathbf{L} \mathbf{D}^{1/2}) (\mathbf{D}^{-1/2} \mathbf{U}) = \mathbf{R}^H \mathbf{R}$ , where

$$\mathbf{D} = \text{diag}(\text{diag}(\mathbf{R})) = \text{diag}(1 + \alpha w_1^2, 1 + \alpha^{(1)} w_2^2, \dots).$$

► Computation of Cholesky-factorization

$$\mathbf{I} + \alpha \mathbf{w} \mathbf{w}^H = \mathbf{R}_1^H \mathbf{R}_1 .$$

Motivation: “recursion”  
(2.9.13).

→ asymptotic complexity  $O(n^2)$   
( $O(n)$ , if only  $\mathbf{d}, \mathbf{s}$  computed  
→ (2.9.16))

Code 2.9.15: Cholesky factorization of rank-1-modified identity matrix

```

1 function [d,s] = roid(alpha,w)
2 n = length(w);
3 d = []; s = [];
4 for i=1:n
5     t = alpha*w(i);
6     d = [d; sqrt(1+t*w(i))];
7     s = [s; t/d(i)];
8     alpha = alpha - s(i)^2;
9 end
    
```

③ Special structure of  $\mathbf{R}_1$ :

$$\mathbf{R}_1 = \begin{pmatrix} d_1 & & & & & \\ & \ddots & & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & & & & & d_n \end{pmatrix} + \begin{pmatrix} s_1 & & & & & \\ & \ddots & & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & & & & & s_n \end{pmatrix} \begin{pmatrix} 0 & w_2 & w_3 & \cdots & \cdots & w_n \\ 0 & 0 & w_3 & \cdots & \cdots & w_n \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & & 0 & w_{n-1} & w_n \\ 0 & \cdots & \cdots & & 0 & w_n \\ 0 & \cdots & & & \cdots & 0 \end{pmatrix} \quad (2.9.16)$$

$$\mathbf{R}_1 = \begin{pmatrix} d_1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & d_n \end{pmatrix} + \begin{pmatrix} s_1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & s_n \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & \cdots & \cdots & 1 \\ 0 & 0 & 1 & \cdots & \cdots & 1 \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & 0 & 1 & 1 \\ 0 & \cdots & \cdots & 0 & 1 & \\ 0 & \cdots & \cdots & \cdots & 0 & 0 \end{pmatrix} \begin{pmatrix} w_1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & w_n \end{pmatrix}$$

Smart  
multiplication

$$\tilde{\mathbf{R}} := \mathbf{R}_1 \mathbf{R}$$

→ Complexity  $O(n^2)$

$$\mathbf{A} + \alpha \mathbf{v} \mathbf{v}^H = \tilde{\mathbf{R}}^H \tilde{\mathbf{R}}$$

Code 2.9.18: Update of Cholesky factorization in the case of s.p.d. preserving rank-1-modification

```

1  function Rt = roudchol(R,alpha,v)
2  w = R' \ v;
3  [d,s] = roid(alpha,w);
4  T = zeros(1,n);
5  for j=n-1:-1:1
6      T = [w(j+1)*R(j+1,:)+T(1,:);T];
7  end
8  Rt = spdiags(d,0,n,n)*R+spdiags(s,0,n,n)*T;

```

Let us adopt an academic point of view: Before we have seen how to update a QR-factorization in the case of rank-1-modification of a *square* matrix.

However, the QR-factorization makes sense for an arbitrary *rectangular* matrix. A possible modification of rectangular matrices is achieved by adding a row or a column. How can QR-factors be updated efficiently for these kinds of modifications.

An application of these modification techniques will be given in Chapter 7.

$$\mathbf{A} \in \mathbb{K}^{m,n} \mapsto \tilde{\mathbf{A}} = [(\mathbf{A})_{:,1}, \dots, (\mathbf{A})_{:,k-1}, \mathbf{v}, (\mathbf{A})_{:,k}, \dots, (\mathbf{A})_{:,n}] , \quad \mathbf{v} \in \mathbb{K}^m . \quad (2.9.19)$$

Known: QR-factorization  $\mathbf{A} = \mathbf{QR}$ ,  $\mathbf{Q} \in \mathbb{K}^{m,m}$  unitary  $\mathbf{R} \in \mathbb{K}^{m,n}$  upper triangular matrix.

Task: Efficient computation of QR-factorization  $\tilde{\mathbf{A}} = \tilde{\mathbf{Q}}\tilde{\mathbf{R}}$  of  $\tilde{\mathbf{A}}$  from (2.9.19),  $\tilde{\mathbf{Q}} \in \mathbb{K}^{m,m}$  unitary,  $\tilde{\mathbf{R}} \in \mathbb{K}^{m,n+1}$  upper triangular

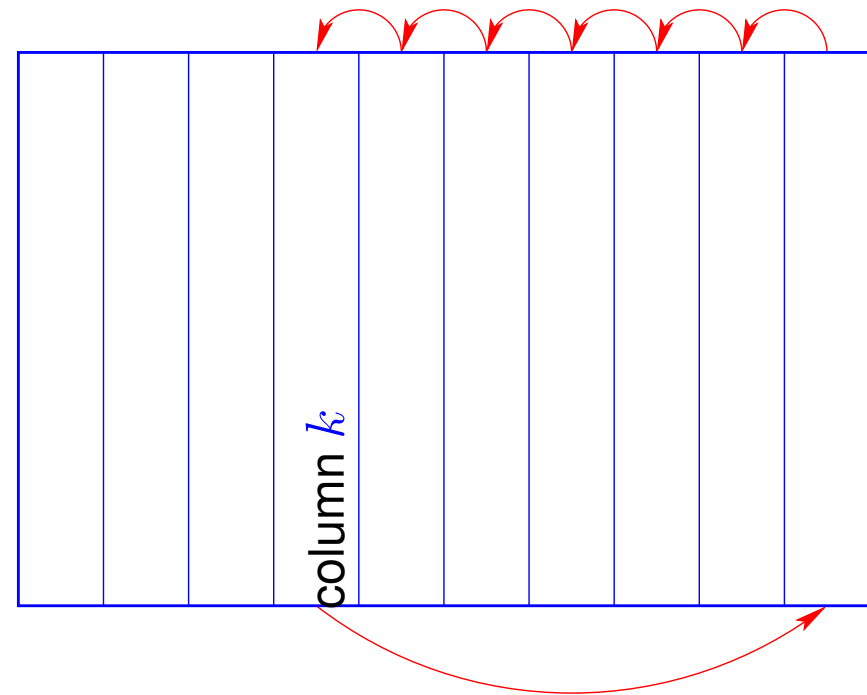
Idea: Easy, if  $k = n + 1$  (adding last column)

▶  $\exists$  column permutation

$$k \mapsto n + 1, i \mapsto i - 1, i = k + 1, \dots, n + 1$$

$\sim$  permutation matrix

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & \ddots & & & \\ & & 1 & 0 & \\ \vdots & & & 0 & 1 \\ \vdots & & & & \ddots \\ 0 & \dots & & 1 & 0 \end{pmatrix} \in \mathbb{R}^{n+1, n+1}$$



$$\tilde{\mathbf{A}} \rightarrow \mathbf{A}_1 = \tilde{\mathbf{A}}\mathbf{P} = [\mathbf{a}_1, \dots, \mathbf{a}_n, \mathbf{v}] = \mathbf{Q} \begin{pmatrix} \mathbf{R} & \mathbf{Q}^H \mathbf{v} \end{pmatrix} = \mathbf{Q}$$

column  $\mathbf{Q}^H \mathbf{v}$   
case  $m > n + 1$

① If  $m > n + 1$ :  $\exists$  orthogonal transformation  $\mathbf{Q}_1 \in \mathbb{K}^{m, m}$  (Householder reflection) with

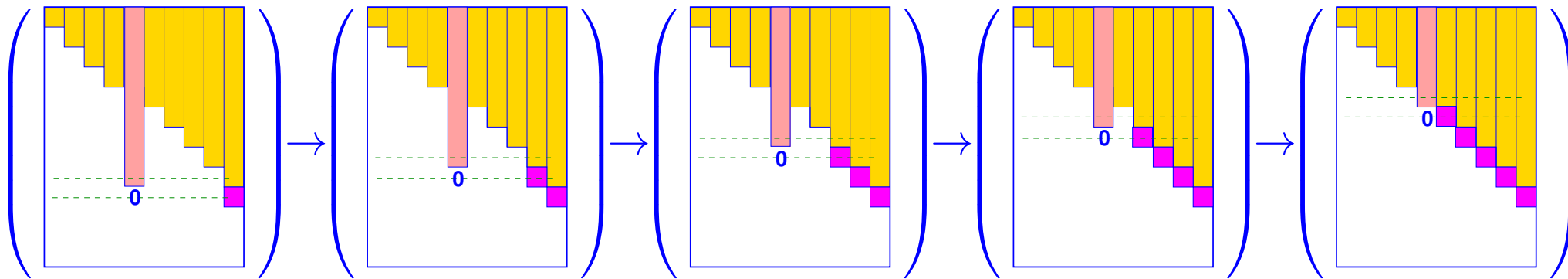




$$\mathbf{Q}_1 \mathbf{Q}^H \tilde{\mathbf{A}} = \mathbf{Q}_1 \mathbf{Q}^H \mathbf{A}_1 \mathbf{P}^H = \begin{pmatrix} * & \dots & * & \dots & * \\ 0 & * & * & \dots & \vdots \\ \vdots & & \ddots & \vdots & \vdots \\ & & & \vdots & \ddots \\ \vdots & & * & * & * \\ 0 & \dots & 0 & \dots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & \dots & 0 & \dots & 0 \end{pmatrix} = \begin{pmatrix} \text{yellow blocks} & & & & \\ & \text{yellow blocks} & & & \\ & & \text{yellow blocks} & & \\ & & & \text{red block} & \\ & & & & \text{yellow blocks} \end{pmatrix}$$

②  $n + 1 - k$  successive Givens rotations  $\Rightarrow$  upper triangular matrix  $\tilde{\mathbf{R}}$

$$\mathbf{Q}_1 \mathbf{Q}^H \mathbf{A}_1 = \begin{pmatrix} * & \dots & * & \dots & * \\ 0 & * & * & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ & & & * & \vdots \\ & & & * & * \\ & & & * & * \\ \vdots & & & * & 0 \\ 0 & \dots & 0 & \dots & \vdots \\ \vdots & & \vdots & & \vdots \\ 0 & \dots & 0 & \dots & 0 \end{pmatrix} \xrightarrow{\mathbf{G}_{n,n+1}} \dots \xrightarrow{\mathbf{G}_{k,k+1}} \begin{pmatrix} * & \dots & * & \dots & * \\ 0 & * & * & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ & & & * & \vdots \\ & & & 0 & \dots \\ & & & 0 & * \\ \vdots & & & 0 & \vdots \\ 0 & \dots & 0 & \dots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & \dots & 0 & \dots & 0 \end{pmatrix}$$



---  $\hat{=}$  rows targeted by plane rotations, ■  $\hat{=}$  new entries  $\neq 0$

➔ Asymptotic complexity  $O((n - k)^2)$

$$\mathbf{A} \in \mathbb{K}^{m,n} \mapsto \tilde{\mathbf{A}} = \begin{bmatrix} (\mathbf{A})_{1,:} \\ \vdots \\ (\mathbf{A})_{k-1,:} \\ \mathbf{v}^T \\ (\mathbf{A})_{k,:} \\ \vdots \\ (\mathbf{A})_{m,:} \end{bmatrix}, \quad \mathbf{v} \in \mathbb{K}^n. \quad (2.9.20)$$

Given: QR-factorization  $\mathbf{A} = \mathbf{Q}\mathbf{R}$ ,  $\mathbf{Q} \in \mathbb{K}^{m+1,m+1}$  unitary,  $\mathbf{R} \in \mathbb{K}^{m,n}$  upper triangular matrix.

Task: efficient computation of QR-factorization  $\tilde{\mathbf{A}} = \tilde{\mathbf{Q}}\tilde{\mathbf{R}}$  of  $\tilde{\mathbf{A}}$  from (2.9.20),  $\tilde{\mathbf{Q}} \in \mathbb{K}^{m+1,m+1}$  unitary,  $\tilde{\mathbf{R}} \in \mathbb{K}^{m+1,n+1}$  upper triangular matrix.

- ①  $\exists$  (partial) cyclic row permutation  $m + 1 \leftarrow k, i \leftarrow i + 1, i = k, \dots, m$ :  
 $\rightarrow$  **unitary** permutation matrix ( $\rightarrow$  Def. 2.3.12)  $\mathbf{P} \in \{0, 1\}^{m+1, m+1}$

$$\mathbf{P}\tilde{\mathbf{A}} = \begin{pmatrix} \mathbf{A} \\ \mathbf{v}^T \end{pmatrix} \blacktriangleright \begin{pmatrix} \mathbf{Q}^H & 0 \\ 0 & 1 \end{pmatrix} \mathbf{P}\tilde{\mathbf{A}} = \begin{pmatrix} \mathbf{R} \\ \mathbf{v}^T \end{pmatrix} = \begin{pmatrix} \text{[Upper triangular matrix with yellow diagonal]} \\ \mathbf{v}^T \end{pmatrix}.$$

case  $m = n$

- ② Transform into upper triangular form by  $m$  successive Givens rotations:

$$\begin{pmatrix} * & \dots & & \dots & * \\ 0 & * & & & \vdots \\ \vdots & 0 & \dots & & \\ \vdots & \vdots & & * & \vdots \\ 0 & 0 & & 0 & * & * \\ 0 & 0 & & 0 & 0 & * \\ * & \dots & \dots & * & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{1, m+1}} \begin{pmatrix} * & \dots & & \dots & * \\ 0 & * & & & \vdots \\ \vdots & 0 & \dots & & \\ \vdots & \vdots & & * & \vdots \\ 0 & 0 & & 0 & * & * \\ 0 & 0 & & 0 & 0 & * \\ 0 & * & \dots & * & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{2, m+1}} \dots$$

$$\dots \xrightarrow{\mathbf{G}_{m-1,m+1}} \begin{pmatrix} * & \dots & & \dots & * \\ 0 & * & & & \vdots \\ \vdots & 0 & \ddots & & \\ \vdots & \vdots & & * & \vdots \\ 0 & 0 & & 0 & * \\ 0 & 0 & & 0 & * \\ 0 & \dots & & \dots & 0 \\ 0 & \dots & & \dots & * \end{pmatrix} \xrightarrow{\mathbf{G}_{m,m+1}} \begin{pmatrix} * & \dots & & \dots & * \\ 0 & * & & & \vdots \\ \vdots & 0 & \ddots & & \\ \vdots & \vdots & & * & \vdots \\ 0 & 0 & & 0 & * \\ 0 & 0 & & 0 & * \\ 0 & \dots & & \dots & 0 \\ 0 & \dots & & \dots & 0 \end{pmatrix} := \tilde{\mathbf{R}} \quad (2.9.21)$$

③ With  $\mathbf{Q}_1 = \mathbf{G}_{m,m+1} \cdots \mathbf{G}_{1,m+1}$

$$\tilde{\mathbf{A}} = \mathbf{P}^T \begin{pmatrix} \mathbf{Q} & 0 \\ 0 & 1 \end{pmatrix} \mathbf{Q}_1^H \tilde{\mathbf{R}} = \tilde{\mathbf{Q}} \tilde{\mathbf{R}} \quad \text{with unitary } \tilde{\mathbf{Q}} \in \mathbb{K}^{m+1,m+1}.$$

☞ Similar update algorithms exist for modifications arising from dropping one row or column of a matrix.

# 3

## Data Interpolation in 1D

Learning outcomes:

- Understand the use of basis functions for representing functions on a computer.
- Know the concept of a interpolation operator and what its linearity means.
- Know the connection between linear interpolation operators and linear systems of equations.
- Be familiar with efficient algorithms for polynomial interpolation in different settings.
- Know the meaning and significance of “sensitivity” in the context of interpolation.
- Be familiar with the notions of “shape preservation” for an interpolation scheme and its different aspects (monotonicity, curvature).
- Know the details of cubic Hermite interpolation and how to ensure that it is monotonicity preserving.
- Know what splines are and how cubic spline interpolation works.

# 3.1 Abstract interpolation

Problem: (multidimensional) **data interpolation** (point interpolation):

Given: data points  $(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots, m,$   $\mathbf{x}_i \in D \subset \mathbb{R}^n,$   $\mathbf{y}_i \in \mathbb{R}^d$

Goal: *reconstruction of a (continuous) function*  $\mathbf{f} : D \mapsto \mathbb{R}^d$  satisfying **interpolation conditions**

$$\mathbf{f}(\mathbf{x}_i) = \mathbf{y}_i, \quad i = 1, \dots, m$$

Additional requirements:

- smoothness of  $\mathbf{f}$ , e.g.  $\mathbf{f} \in C^1$ , etc.
- shape of  $\mathbf{f}$  (positivity, monotonicity, convexity  $\rightarrow$  Sect. 3.6)

Focus in this chapter:  $n = 1 \leftrightarrow$  interpolation of data depending on *one* parameter ( $t \in \mathbb{R}$ ):

$$\text{interpolation conditions: } \mathbf{f}(t_i) = \mathbf{y}_i, \quad i = 1, \dots, m. \quad (3.1.1)$$

The next example addresses an important application of data interpolation in 1D.

*Example 3.1.2* (Constitutive relations (*ger.* Kennlinien) from measurements).



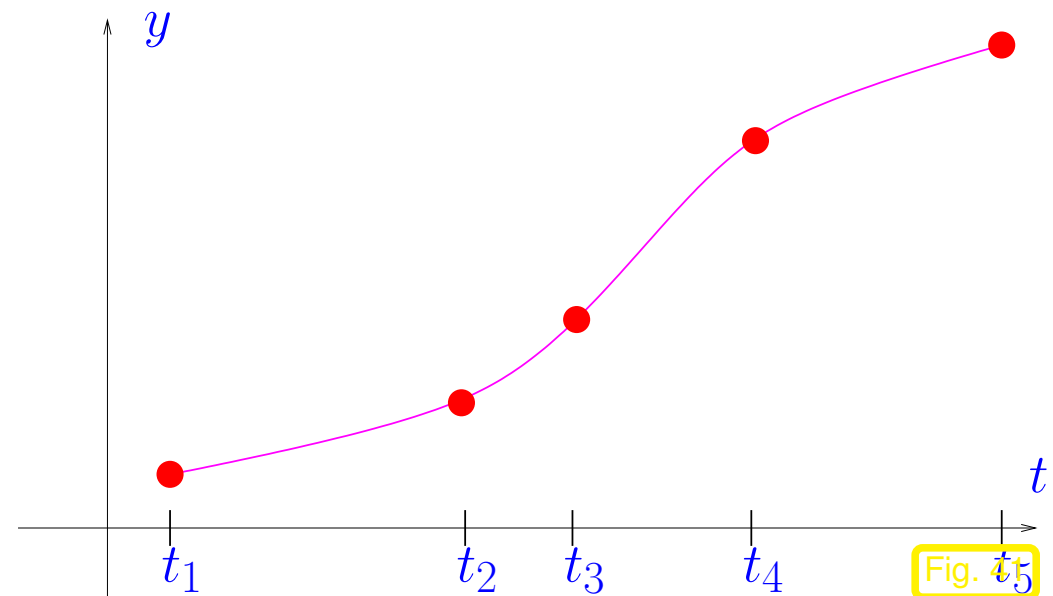
In this context:  $t, y \hat{=}$  two state variables of a physical system, a functional dependence  $y = y(t)$  is assumed.

Known: several *accurate* measurements

$$(t_i, y_i), \quad i = 1, \dots, m$$

Examples:

$t$	$y$
voltage $U$	current $I$
pressure $p$	density $\rho$
magnetic field $H$	magnetic flux $B$
...	...



Meaning of attribute “accurate”: justification for interpolation. If measured values  $y_i$  were affected by considerable errors, one would not impose the interpolation conditions (3.1.1), but opt for **data fitting** ( $\rightarrow$  Ex. 7.0.1).



! General “mathematical” function  $f : D \subset \mathbb{R} \mapsto \mathbb{K}^d$ ,  $D$  interval, contains an “infinite amount of information”.

What is a function  $f : I \subset \mathbb{R} \mapsto \mathbb{R}$  “from the perspective of a computer”?

```
1 class Function {  
2   private :  
3     // various internal data describing  $f$   
4   public :  
5     // Constructor: accepts information necessary for specifying the  
6     // function  
7     Function(/* .... */);  
8     // Evaluation operator  
9     double operator () (double t) const;  
};
```

? How to represent  $f$  on a computer?

➔ Idea: **parametrization**, a finite number of parameters  $c_1, \dots, c_n$ ,  $n \in \mathbb{N}$ , characterizes  $f$ .

Special case: Representation with *finite linear combination* of **basis functions**

$b_j : D \subset \mathbb{R} \mapsto \mathbb{K}$ ,  $j = 1, \dots, n$ :

$$f = \sum_{j=1}^n c_j b_j \quad , \quad c_j \in \mathbb{K}^d . \quad (3.1.4)$$

➔  $f \in$  finite dimensional **function space**  $V_n := \text{Span} \{b_1, \dots, b_n\}$ .

☞ Focus in this chapter will be on this special case.



*Example 3.1.5* (Piecewise linear interpolation). See also Sect. 3.6.1

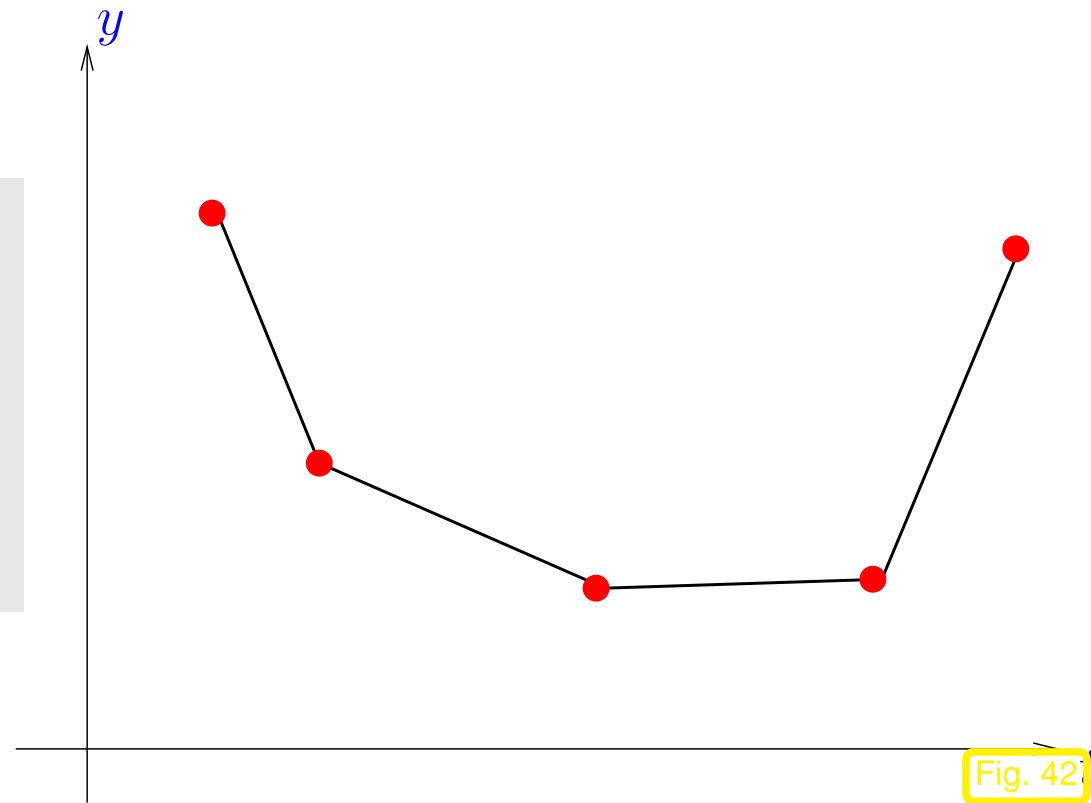
Recall: linear function in 1D = function of the form  $x \mapsto a + bx$ ,  $a, b \in \mathbb{R}$  (polynomial of degree 1).

Piecewise linear interpolation = connect data points  $(t_i, y_i)$ ,  $i = 1, \dots, m$ ,  $t_{i-1} < t_i$ , by line segments

➤ interpolating polygon

Convenient/canonical basis in this case?

Piecewise linear interpolant of data



“Tent function” (“hat function”) basis:

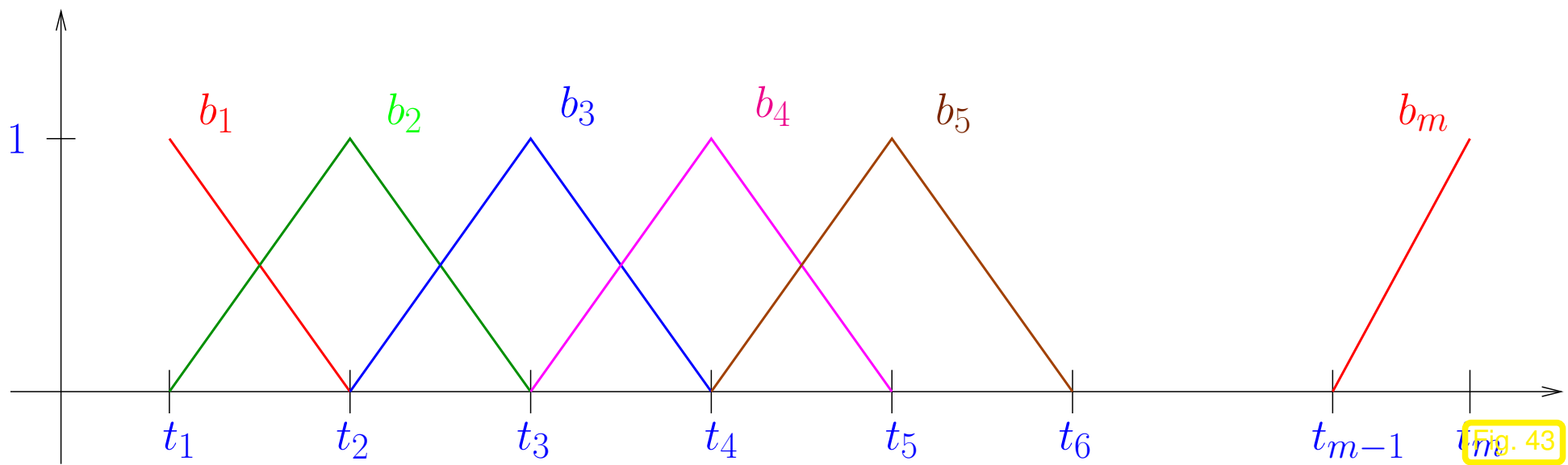


Fig. 43

Note: in Fig. 43 the basis functions have to be extended by zero outside the  $t$ -range where they are drawn.

Formulas for the basis functions:

$$\begin{aligned}
 b_1(t) &= \begin{cases} 1 - \frac{t-t_1}{t_2-t_1} & \text{for } t_1 \leq t < t_2, \\ 0 & \text{for } t \geq t_2. \end{cases} \\
 b_j(t) &= \begin{cases} 1 - \frac{t_j-t}{t_j-t_{j-1}} & \text{for } t_{j-1} \leq t < t_j, \\ 1 - \frac{t-t_j}{t_{j+1}-t_j} & \text{for } t_j \leq t < t_{j+1}, \\ 0 & \text{elsewhere in } [t_1, t_m]. \end{cases}, \quad j = 2, \dots, m-1, \quad (3.1.6)
 \end{aligned}$$

Basis functions are *uniquely* determined by the conditions

- $b_j$  is **continuous** on  $[t_1, t_m]$ ,
- $b_j$  is linear on each interval  $[t_{i-1}, t_i]$ ,  $i = 2, m$ ,
- $b_j(t_i) = \delta_{ij} := \begin{cases} 1 & , \text{ if } i = j , \\ 0 & \text{ else.} \end{cases} \quad \triangleright \quad \text{cardinal basis}$

This last condition implies a simple basis representation of a (the ?) piecewise linear interpolant of  $(t_i, y_i)$ ,  $i = 1, m$ :

$$f(t) = \sum y_j b_j(t) , \quad t_1 \leq t \leq t_m , \quad (3.1.7)$$

where the  $b_j$  are given by (3.1.6).



*Remark 3.1.8* (Interpolation as linear mapping).

Setting: interpolation based on finite linear combinations of basis functions, see Rem. 3.1.3:

$$(3.1.1) \ \& \ (3.1.4) \ \Rightarrow \ f(t_i) = \sum_{j=1}^n c_j b_j(t_i) = y_i, \quad i = 1, \dots, m, \quad (3.1.9)$$

$$\Updownarrow$$

$$\mathbf{A} \mathbf{c} := \begin{pmatrix} b_1(t_1) & \dots & b_n(t_1) \\ \vdots & & \vdots \\ b_1(t_m) & \dots & b_n(t_m) \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} =: \mathbf{y}. \quad (3.1.10)$$

=  $m \times n$  linear system of equations !

**Necessary condition** for unique solvability of interpolation problem (3.1.9) :  $m = n$



If  $m = n$  and  $\mathbf{A}$  from (3.1.10) regular ( $\rightarrow$  Def. 2.0.1), then

For *fixed*  $t_i$  the interpolation problem (3.1.9) defines linear mapping

$$I : \begin{cases} \mathbb{K}^n \mapsto V_n \\ \mathbf{y} \mapsto f \end{cases}$$

data space

function space

More precisely:

An **interpolation operator**  $I : \mathbb{R}^m \mapsto C^0([t_1, t_m])$  for the given nodes  $t_1 < t_2 < \dots < t_m$  is called *linear*, if

$$I(c\mathbf{y} + \beta\mathbf{z}) = cI(\mathbf{y}) + \beta I(\mathbf{z}) \quad \forall \mathbf{y}, \mathbf{z} \in \mathbb{R}^{n+1}, c, \beta \in \mathbb{R}.$$

 Notation:  $C^0([t_1, t_m]) \hat{=}$  vector space of continuous functions on  $[t_1, t_m]$



*Remark 3.1.11* (“Software solution” of interpolation problem).

```

1 class Interpolant {
2   private :
3     // various internal data describing f
4   public :
5     // Constructor: computation of coefficients  $c_j$  of representation (3.1.4)
6     Interpolant(const vector<double> &t, const vector<double> &y);
7     // Evaluation operator for interpolant f
8     double operator () (double t) const;
9 };

```



Practical object oriented implementation of interpolation operator:

- Constructor: “setup phase”, e.g. building and solving linear system of equations (3.1.10)
- Evaluation operator, e.g., implemented as evaluation of linear combination (3.1.4)

Crucial issue: computational effort for evaluation of interpolant at single point:  $O(1)$  or  $O(m)$  (or in between) ?



# 3.2 Polynomials

Notation: Vector space of the polynomials of degree  $\leq k$ ,  $k \in \mathbb{N}$ :

$$\mathcal{P}_k := \{t \mapsto \alpha_k t^k + \alpha_{k-1} t^{k-1} + \dots + \alpha_1 t + \alpha_0, \alpha_j \in \mathbb{K}\}. \quad (3.2.1)$$

leading coefficient

Terminology: the functions  $t \mapsto t^k$ ,  $k \in \mathbb{N}_0$ , are called **monomials**

$$t \mapsto \alpha_k t^k + \alpha_{k-1} t^{k-1} + \dots + \alpha_0 = \text{monomial representation of a polynomial.}$$

Monomial representation is a linear combination of basis functions  $t \mapsto t^k$ , see Rem. 3.1.3.

Obvious:  $\mathcal{P}_k$  is a vector space, see [48, Sect. 4.2, Bsp. 4]. What is its dimension?

**Theorem 3.2.2** (Dimension of space of polynomials).

$$\dim \mathcal{P}_k = k + 1 \quad \text{and} \quad \mathcal{P}_k \subset C^\infty(\mathbb{R}).$$

*Proof.* Dimension formula by linear independence of monomials.

Why are polynomials important in computational mathematics ?

- Easy to compute (only elementary operations required), integrate and differentiate
- Vector space & algebra
- Analysis: Taylor polynomials & power series

*Remark 3.2.3* (Polynomials in Matlab).

**MATLAB:**  $\alpha_k t^k + \alpha_{k-1} t^{k-1} + \dots + \alpha_0 \rightarrow$  Vector  $(\alpha_k, \alpha_{k-1}, \dots, \alpha_0)$  (ordered!).



*Remark 3.2.4* (Horner scheme).  $\rightarrow$  [13, Bem. 8.11]

Evaluation of a polynomial in monomial representation:

**Horner scheme**

$$p(t) = t(\dots t(t(\alpha_n t + \alpha_{n-1}) + \alpha_{n-2}) + \dots + \alpha_1) + \alpha_0 . \quad (3.2.5)$$

```
1 function y = polyval(p,x)
2 y = p(1); for i=2:length(p), y = x*y+p(i); end
```

Asymptotic complexity:  $O(n)$ Use: MATLAB “built-in”-function `polyval(p,x);`

## 3.3 Polynomial Interpolation: Theory

→ [13, Sect. 8.2.1], [51, Sect. 8.1]

Interpolation problem ( $\rightarrow$  Sect. 3.1): (re-)construction of a polynomial through points  $(t_i, y_i)$ .

*Lagrange polynomial interpolation problem*

Given the **simple nodes**  $t_0, \dots, t_n$ ,  $n \in \mathbb{N}$ ,  $-\infty < t_0 < t_1 < \dots < t_n < \infty$  and the values  $y_0, \dots, y_n \in \mathbb{K}$  compute  $p \in \mathcal{P}_n$  such that

$$p(t_j) = y_j \quad \text{for } j = 0, \dots, n. \quad (3.3.1)$$

For **nodes**  $t_0 < t_1 < \dots < t_n$  ( $\rightarrow$  Lagrange interpolation) consider

**Lagrange polynomials**  $L_i(t) := \prod_{\substack{j=0 \\ j \neq i}}^n \frac{t - t_j}{t_i - t_j}, \quad i = 0, \dots, n. \quad (3.3.2)$

R. Hiptmair  
rev 38286,  
October 26,  
2011



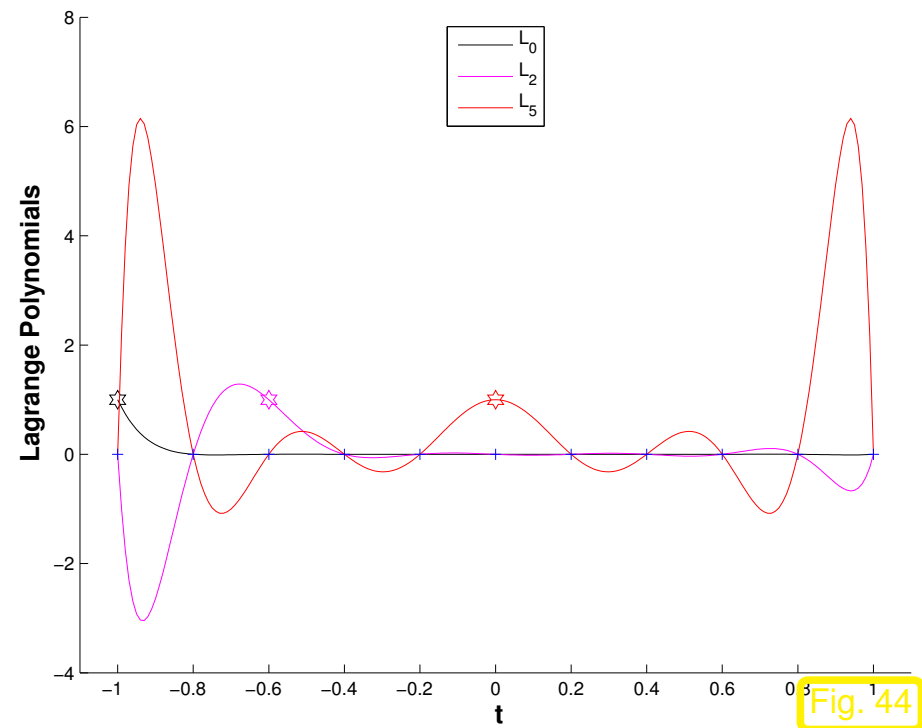
$$L_i \in \mathcal{P}_n \quad \text{and} \quad L_i(t_j) = \delta_{ij}$$

Recall the Kronecker symbol  $\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{else.} \end{cases}$

*Example 3.3.3.* Lagrange polynomials for uniformly spaced nodes

$$\mathcal{T} := \left\{ t_j = -1 + \frac{2}{n} j \right\}, \\ j = 0, \dots, n.$$

Plot  $n = 10$ ,  $j = 0, 2, 5 \rightarrow$



The Lagrange polynomials  $L_0, \dots, L_n$  form a *basis* of  $\mathcal{P}_n$ , because they are obviously (?) linearly independent.

The Lagrange interpolation polynomial  $p$  for data  $(t_i, y_i)_{i=0}^n$  has the representation:

$$p(t) = \sum_{i=0}^n y_i L_i(t), \quad \Rightarrow \quad p \in \mathcal{P}_n \quad \text{and} \quad p(t_i) = y_i. \quad (3.3.4)$$

**Theorem 3.3.5** (Existence & uniqueness of Lagrange interpolation polynomial).  $\rightarrow$  [51, Thm. 8.1], [13, Satz 8.3]

The general polynomial interpolation problem (3.3.1) admits a unique solution  $p \in \mathcal{P}_n$ .

*Proof.* Consider the linear evaluation operator

$$\text{eval}_{\mathcal{T}} : \begin{cases} \mathcal{P}_n \mapsto \mathbb{R}^{n+1}, \\ p \mapsto (p(t_i))_{i=0}^n, \end{cases}$$

which maps between finite-dimensional vector spaces of the same dimension, see Thm. 3.2.2.

Representation (3.3.4)  $\Rightarrow$  existence of interpolating polynomial  
 $\Rightarrow$   $\text{eval}_{\mathcal{T}}$  is **surjective** (“onto”)

Known from linear algebra: for a linear mapping  $T : V \mapsto W$  between finite-dimensional vector spaces with  $\dim V = \dim W$  holds the equivalence

$$T \text{ surjective} \Leftrightarrow T \text{ bijective} \Leftrightarrow T \text{ injective.}$$

Applying this equivalence to  $\text{eval}_{\mathcal{T}}$  yields the assertion of the theorem □

Lagrangian polynomial interpolation leads to linear systems of equations: with **monomial basis** representation

$$p(t_j) = y_j \iff \sum_{i=0}^n a_i t_j^i = y_j, j = 0, \dots, n$$

$\iff$  solution of  $(n + 1) \times (n + 1)$  linear system  $\mathbf{V}\mathbf{a} = \mathbf{y}$  with matrix

$$\mathbf{V} = \begin{pmatrix} 1 & t_0 & t_0^2 & \cdots & t_0^n \\ 1 & t_1 & t_1^2 & \cdots & t_1^n \\ 1 & t_2 & t_2^2 & \cdots & t_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^n \end{pmatrix}. \quad (3.3.6)$$

A matrix in the form of  $\mathbf{V}$  is called **Vandermonde matrix**.

Given a column vector  $\mathbf{t}$ , the corresponding Vandermonde matrix can be generated by



- for  $j = 1 : \text{length}(t)$ ;  $V(j,:) = t(j).\wedge[0 : \text{length}(t) - 1]$ ; end;
- for  $j = 1 : \text{length}(t)$ ;  $V(:,j) = t.\wedge(j - 1)$ ; end;
- `fliplr(vander(t))`

**Theorem 3.3.7** (Lagrange interpolation as linear mapping).  $\rightarrow$  *Rem. 3.1.8*

The polynomial interpolation in the nodes  $\mathcal{T} := \{t_j\}_{j=0}^n$  defines a linear operator

$$I_{\mathcal{T}} : \begin{cases} \mathbb{K}^{n+1} & \rightarrow \mathcal{P}_n, \\ (y_0, \dots, y_n)^T & \mapsto \text{interpolating polynomial } p. \end{cases} \quad (3.3.8)$$

*Remark 3.3.9* (Matrix representation of interpolation operator).

In the case of Lagrange interpolation:

- if Lagrange polynomials are chosen as basis for  $\mathcal{P}_n$ ,  $\rightarrow I_{\mathcal{T}}$  is represented by the identity matrix;

- if monomials are chosen as basis for  $\mathcal{P}_n$ ,  $\rightarrow I_{\mathcal{T}}$  is represented by the inverse of the Vandermonde matrix  $\mathbf{V}$ , see (3.3.6).



*Remark 3.3.10* (Generalized polynomial interpolation).  $\rightarrow$  [13, Sect. 8.2.7], [51, Sect. 8.4]

Generalization: Beside function values also prescribe **derivatives** up to a certain order for interpolating polynomial at given nodes.

Convention: indicate occurrence of derivatives as interpolation conditions by **multiple nodes**.

### *Generalized polynomial interpolation problem*

Given the **(possibly multiple) nodes**  $t_0, \dots, t_n$ ,  $n \in \mathbb{N}$ ,  $-\infty < t_0 \leq t_1 \leq \dots \leq t_n < \infty$  and the values  $y_0, \dots, y_n \in \mathbb{K}$  compute  $p \in \mathcal{P}_n$  such that

$$\frac{d^k}{dt^k} p(t_j) = y_j \quad \text{for } k = 0, \dots, l_j \quad \text{and } j = 0, \dots, n, \quad (3.3.11)$$

When all the multiplicities are equal to 2: **Hermite interpolation** (or osculatory interpolation) [51, Ex. 8.6],

$$t_0 = t_1 < t_2 = t_3 < \cdots < t_{n-1} = t_n \quad \rightarrow \quad p(t_{2j}) = y_{2j}, p'(t_{2j}) = y_{2j+1}, \quad (\text{double nodes}).$$

*Theorem 3.3.12 (Existence & uniqueness of generalized Lagrange interpolation polynomials).  
The general polynomial interpolation problem (3.3.11) admits a unique solution  $p \in \mathcal{P}_n$ .*

*Definition 3.3.13 (Generalized Lagrange polynomials).*

The **generalized Lagrange polynomials** on the nodes  $\mathcal{T} = \{t_j\}_{j=0}^n \subset \mathbb{R}$  are  $L_i := l_{\mathcal{T}}(\mathbf{e}_{i+1})$ ,  $i = 0, \dots, n$ , where  $\mathbf{e}_i = (0, \dots, 0, 1, 0, \dots, 0)^T \in \mathbb{R}^{n+1}$  are the unit vectors.

Note: the linear interpolation operator  $l_{\mathcal{T}}$  in this definition refers to generalized Lagrangian interpolation. Its existence is guaranteed by Thm. 3.3.12.

*Example 3.3.14.* (Generalized Lagrange polynomials for Hermite Interpolation) mode details to

Sect. 3.7

double nodes

$$t_0 = 0, t_1 = 0, t_2 = 1, t_3 = 1 \Rightarrow n = 3$$

(cubic Hermite interpolation).

Explicit formulas for the polynomials  $\rightarrow$  see (3.7.2).

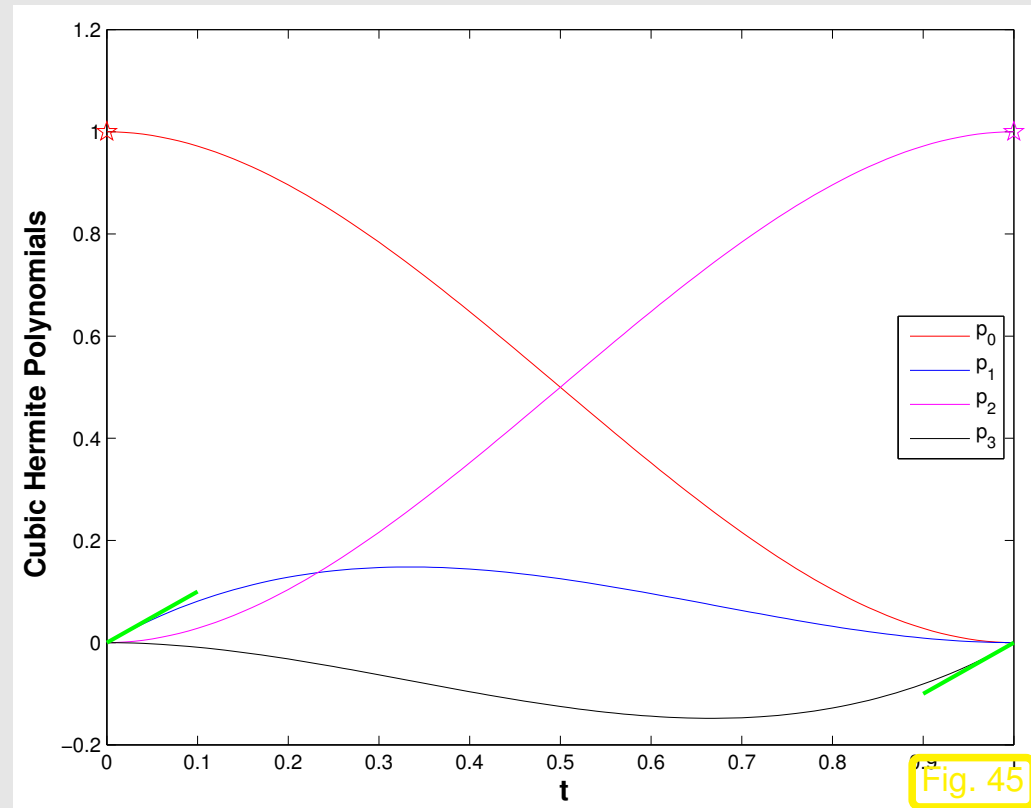


Fig. 45

## 3.4 Polynomial Interpolation: Algorithms

Given: nodes  $\mathcal{T} := \{-\infty < t_0 < t_1 < \dots < t_n < \infty\}$ ,  
values  $\mathbf{y} := \{y_0, y_1, \dots, y_n\}$ ,

define:  $p := I_{\mathcal{T}}(\mathbf{y})$  as the unique Lagrange interpolation polynomial given by Theorem 3.3.5.

### 3.4.1 Multiple evaluations

Task: For *fixed* set  $\{t_0, \dots, t_n\}$  of nodes  
and *many* given data values  $y_i^k$  and arguments  $x_k$ ,  $k = 1, \dots, N$ ,  $N \gg 1$ ,  
efficiently evaluate  $p(x_k)$  for  $p \in \mathcal{P}_n$  interpolating in  $(t_i, y_i^k)$ ,  $i = 0, \dots, n$ .

```
1 class PolyInterp {  
2   private :  
3     // various internal data describing p  
4   public :  
5     // Constructor taking node vector (t_0, ..., t_n) as argument  
6     PolyInterp(const vector<double> &t);
```

```

7 // Evaluation operator at x for data (y0,...,yn)
8 double eval(double x, const vector<double> &y) const;
9 };

```

- Interpolation with Lagrange polynomials (3.3.2) , (3.3.4) is not efficient:  $O(n^2)$  operations for every invocation of `eval(x, y)`.
- More efficient formula:

Simple manipulations:

$$p(t) = \sum_{i=0}^n L_i(t) y_i = \sum_{i=0}^n \prod_{\substack{j=0 \\ j \neq i}}^n \frac{t - t_j}{t_i - t_j} y_i = \sum_{i=0}^n \lambda_i \prod_{\substack{j=0 \\ j \neq i}}^n (t - t_j) y_i = \prod_{j=0}^n (t - t_j) \cdot \sum_{i=0}^n \frac{\lambda_i}{t - t_i} y_i .$$

with 
$$\lambda_i = \frac{1}{(t_i - t_0) \cdots (t_i - t_{i-1})(t_i - t_{i+1}) \cdots (t_i - t_n)}, i = 0, \dots, n.$$

From above formula, with  $p(t) \equiv 1, y_i = 1$ :

$$1 = \prod_{j=0}^n (t - t_j) \sum_{i=0}^n \frac{\lambda_i}{t - t_i} \quad \Rightarrow \quad \prod_{j=0}^n (t - t_j) = \frac{1}{\sum_{i=0}^n \frac{\lambda_i}{t - t_i}}$$

Barycentric interpolation formula

$$p(t) = \frac{\sum_{i=0}^n \frac{\lambda_i}{t - t_i} y_i}{\sum_{i=0}^n \frac{\lambda_i}{t - t_i}}. \quad (3.4.1)$$

with  $\lambda_i = \frac{1}{(t_i - t_0) \cdots (t_i - t_{i-1})(t_i - t_{i+1}) \cdots (t_i - t_n)}$ ,  $i = 0, \dots, n$  → precompute !

Computational effort: • computation of  $\lambda_i$ :  $O(n^2)$  (only once),

• every subsequent evaluation of  $p$ :  $O(n)$ ,

⇒ total effort  $O(Nn) + O(n^2)$

Code 3.4.2: Evaluation of the interpolation polynomials with barycentric formula

```

1 function p = intpolyval(t, y, x)
2 % t: row vector of nodes  $t_0, \dots, t_n$ 
3 % y: row vector of data  $y_0, \dots, y_n$ 
4 % x: row vector of evaluation points  $x_1, \dots, x_N$ 
5 n = length(t); % number of interpolation nodes = degree of polynomial - 1
6 N = length(x); % Number of evaluation points stored in x
7 % Precompute the weights  $\lambda_i$  with effort  $O(n^2)$ 
8 for k = 1:n
9     lambda(k) = 1 / prod(t(k) - t([1:k-1, k+1:n])); end;
10 for i = 1:N

```

```
11 % Compute quotient of weighted sums of  $\frac{\lambda_i}{t-t_i}$ , effort  $O(n)$ 
12 z = (x(i)-t); j = find (z == 0);
13 if (~isempty (j)), p(i) = y(j); % avoid division by zero
14 else
15     mu = lambda./z; p(i) = sum (mu.*y) / sum (mu);
16 end
17 end
```

tic-toc-computational time, Matlab polyval vs. barycentric formula → Ex. 3.4.5.

### 3.4.2 Single evaluation [13, Sect. 8.2.2]



Task: evaluation of  $p$  in few points,

## Aitken-Neville scheme

Given: nodes  $\mathcal{T} := \{t_j\}_{j=0}^n \subset \mathbb{R}$ , pairwise different,  $t_i \neq t_j$  for  $i \neq j$ ,  
values  $y_0, \dots, y_n$ ,  
*one* evaluation point  $t \in \mathbb{R}$ .

For  $\{i_0, \dots, i_m\} \subset \{0, \dots, n\}$ ,  $0 \leq m \leq n$ :

$p_{i_0, \dots, i_m}$  = interpolation polynomial of degree  $m$  through  $(t_{i_0}, y_{i_0}), \dots, (t_{i_m}, y_{i_m})$ ,

recursive definition:

$$\begin{aligned}
 p_i(t) &\equiv y_i, & i = 0, \dots, n, \\
 p_{i_0, \dots, i_m}(t) &= \frac{(t - t_{i_0})p_{i_1, \dots, i_m}(t) - (t - t_{i_m})p_{i_0, \dots, i_{m-1}}(t)}{t_{i_m} - t_{i_0}} \\
 &= \text{TODO!} .
 \end{aligned} \tag{3.4.3}$$

Aitken-Neville algorithm:

	$n = 0$	1	2	3
$t_0$	$y_0 =: p_0(x)$	$\rightarrow p_{01}(x)$	$\rightarrow p_{012}(x)$	$\rightarrow p_{0123}(x)$
$t_1$	$y_1 =: p_1(x)$	$\rightarrow p_{12}(x)$	$\rightarrow p_{123}(x)$	
$t_2$	$y_2 =: p_2(x)$	$\rightarrow p_{23}(x)$		
$t_3$	$y_3 =: p_3(x)$			

Code 3.4.4: Aitken-Neville algorithm

```

1 function v = ANipoleval(t,y,x)
2 for i=1:length(y)
3     for k=i-1:-1:1
4         y(k) =
5             y(k+1) + (y(k+1) - y(k)) * ...
6                 (x - t(i)) / (t(i) - t(k));
7     end
8 end
v = y(1);

```

```

1 class PolyEval {
2     private :
3         // evaluation point and various internal data describing the polynomials
4     public :
5         // Constructor taking the evaluation point as argument
6         PolyEval(double x);
7         // Add another data point and update internal information
8         void addPoint(t,y);
9         // Value of current interpolating polynomial at x
10        double eval(void) const;
11 };

```

Example 3.4.5 (Timing polynomial evaluations).

Comparison of the computational time needed for polynomial interpolation of

$$\{t_i = i\}_{i=1,\dots,n}, \quad \{y_i = \sqrt{i}\}_{i=1,\dots,n},$$

$n = 3, \dots, 200$ , and evaluation in a point  $x \in [0, n]$ .

Minimum `tic-toc`-computational time  
over 100 runs →

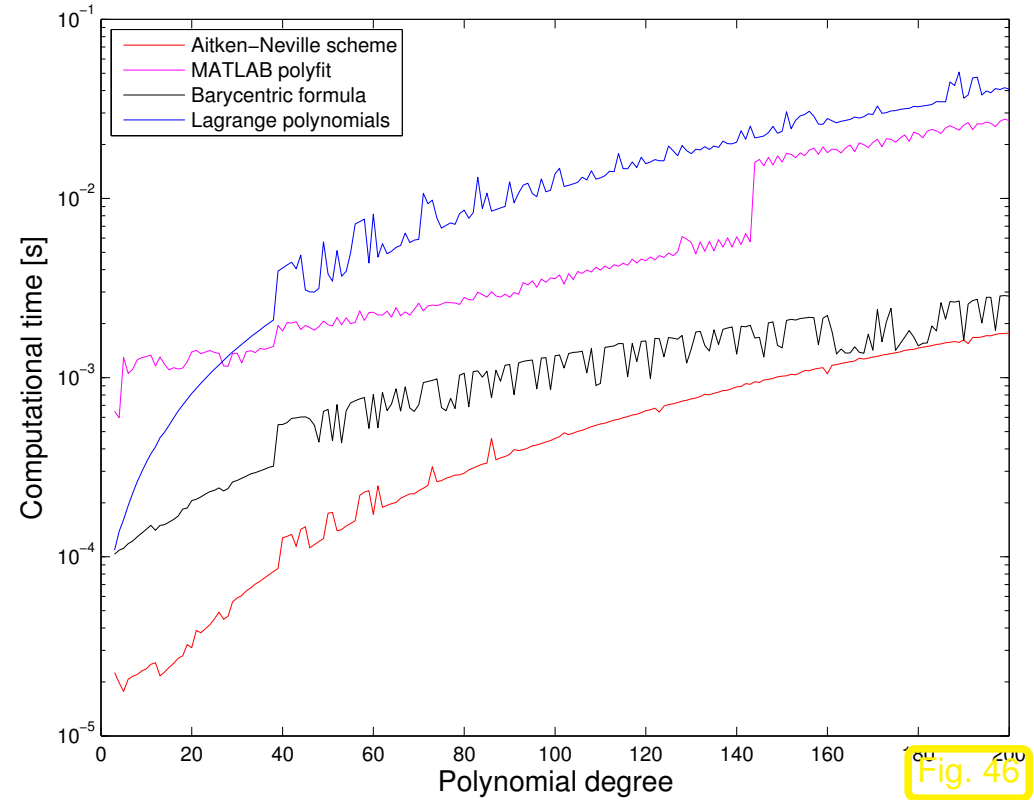


Fig. 46

## Code 3.4.6: Timing polynomial evaluations

```

1 time=zeros(1,4);
2 f=@(x) sqrt(x);           % function to interpolate:
3 for k=1:100
4     res = [];
5     for n=3:1:200         % n = increasing polynomial degree
6         t = (1:n);      y = f(t);      x=n*rand;
7         tic;    v1 = ANipoleval(t,y,x);    time(1) = toc;
8         tic;    v2 = ipoleval(t,y,x);      time(2) = toc;
9         tic;    v3 = intpolyval(t,y,x);    time(3) = toc;
10        tic;    v4 = intpolyval_lag(t,y,x); time(4) = toc;
11        res = [res; n,time];
12    end
13    if (k == 1), finres = res;
14    else, finres = min(finres,res); end
15end
16figure
17semilogy(finres(:,1),finres(:,2),'r-',finres(:,1),finres(:,3),'m-',
18         finres(:,1),finres(:,4),'k-', finres(:,1),finres(:,5),'b-');
19xlabel('Polynomial degree','FontSize',14);
20ylabel('Computational time [s]','FontSize',14);
21legend('Aitken-Neville scheme','MATLAB polyfit', 'Barycentric formula',
22       'Lagrange polynomials',2);

```

R. Hiptmair

rev 38286,  
October 26,  
2011

This uses functions given in Code 3.4.1, Code 3.4.3 and the MATLAB function `polyfit` (with a clearly greater computational effort !)

### Code 3.4.7: MATLAB polynomial evaluation using built-in function `polyfit`

```
1 function v=ipoleval(t,y,x)
2   p = polyfit(t,y,length(y)-1);
3   v=polyval(p,x);
```

### Code 3.4.8: Lagrange polynomial interpolation and evaluation

```
1 function p = intpolyval_lag(t,y,x)
2 p=zeros(size(x));
3 for k=1:length(t); p=p + y(k)*lagrangepoly(x, k-1, t); end
4
5 function L=lagrangepoly(x, index, nodes)
6 L=1;
7 for j=[0:index-1, index+1:length(nodes)-1];
8     L = L .* (x-nodes(j+1)) ./ ( nodes(index+1)-nodes(j+1) );
9 end
```



## 3.4.3 Extrapolation to zero

Extrapolation is the same as interpolation but the evaluation point  $t$  is outside the interval  $[\inf_{j=0,\dots,n} t_j, \sup_{j=0,\dots,n} t_j]$ . W.l.o.g. assume  $t = 0, t_i > 0$ .

Problem: compute  $\lim_{t \rightarrow 0} f(t)$  with prescribed precision, when the evaluation of the function  $y = f(t)$  is numerically unstable ( $\rightarrow$  Sect. 2.5.2) for  $|t| \ll 1$ .

The extrapolation technique introduced below works well, if

- $f$  is an *even function* of its argument:  $f(t) = f(-t)$ ,
- $f$  behaves “nicely” around  $t = 0$ .

Rigorous: existence of an **asymptotic expansion** in  $h^2$

$$f(h) = f(0) + A_1 h^2 + A_2 h^4 + \dots + A_n h^{2n} + R(h) \quad , \quad A_k \in \mathbb{K} \quad ,$$

with **remainder estimate**  $|R(h)| = O(h^{2n+2})$  for  $h \rightarrow 0$ .

Idea:



- ① evaluation of  $f(t_i)$  for different  $t_i$ ,  $i = 0, \dots, n$ ,  $|t_i| > 0$ .
- ②  $f(0) \approx p(0)$  with interpolation polynomial  $p \in \mathcal{P}_n$ ,  $p(t_i) = f(t_i)$ .

*Example 3.4.9* (Numeric differentiation through extrapolation).

Given: smooth function  $f : I \subset \mathbb{R} \mapsto \mathbb{R}$  in **procedural form**: function  $y = f(x)$

Sought: (approximation of)  $f'(x)$ ,  $x \in I$ .

Natural idea: approximation of derivative by (symmetric) **difference quotient**

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \quad (3.4.10)$$

► straightforward implementation:

MATLAB-CODE : Numeric differentiation through finite differences & relative errors.

```
x=1.1; h=2.[-1:-5:-36];
atanerr = abs(dirnumdiff(atan,x,h)-1/(1+x^2))*(1+x^2);
sqrterr = abs(dirnumdiff(sqrt,x,h)-1/(2*sqrt(x)))*(2*sqrt(x));
experr = abs(dirnumdiff(exp,x,h)-exp(x))/exp(x);

function[df]=dirnumdiff(f,x,h)
df=(f(x+h)-f(x))./h;
end
```

$$f(x) = \arctan(x)$$

$h$	Relative error
$2^{-1}$	0.20786640808609
$2^{-6}$	0.00773341103991
$2^{-11}$	0.00024299312415
$2^{-16}$	0.00000759482296
$2^{-21}$	0.00000023712637
$2^{-26}$	0.00000001020730
$2^{-31}$	0.00000005960464
$2^{-36}$	0.00000679016113

$$f(x) = \sqrt{x}$$

$h$	Relative error
$2^{-1}$	0.09340033543136
$2^{-6}$	0.00352613693103
$2^{-11}$	0.00011094838842
$2^{-16}$	0.00000346787667
$2^{-21}$	0.00000010812198
$2^{-26}$	0.00000001923506
$2^{-31}$	0.00000001202188
$2^{-36}$	0.00000198842224

$$f(x) = \exp(x)$$

$h$	Relative error
$2^{-1}$	0.29744254140026
$2^{-6}$	0.00785334954789
$2^{-11}$	0.00024418036620
$2^{-16}$	0.00000762943394
$2^{-21}$	0.00000023835113
$2^{-26}$	0.00000000429331
$2^{-31}$	0.00000012467100
$2^{-36}$	0.00000495453865

Effect of **roundoff errors**:  $h \rightarrow 0$  does not achieve arbitrarily high accuracy. Rather, fewer correct digits for very small  $h$ ! (This is due to an effect called **cancellation**) that haunts the denominator of the difference quotient. More detailed discussion in [13, Sect. 8.2.6].

Extrapolation offers a numerically stable ( $\rightarrow$  Sect. 2.5.2) alternative, because for a  $2(n+1)$ -times continuously differentiable function  $f : D \subset \mathbb{R} \mapsto \mathbb{R}$ ,  $x \in D$  (Taylor sum in  $x$  with Lagrange residual)

$$T(h) := \frac{f(x+h) - f(x-h)}{2h} \sim f'(x) + \sum_{k=1}^n \frac{1}{(2k)!} \frac{d^{2k}f}{dx^{2k}}(x) h^{2k} + \frac{1}{(2n+2)!} f^{(2n+2)}(\xi(x)) .$$



Since  $\lim_{h \rightarrow 0} T(h) = f'(x) \rightarrow$  approximate  $f'(x)$  by interpolation of  $T$  in points  $h_i$ .

### Code 3.4.11: Numerical differentiation by extrapolation to zero

```

1 function d = diffex(f,x,h0,tol)
2 % f: handle of to a function defined in a neighborhood of  $x \in \mathbb{R}$ ,
3 % x: point at which approximate derivative is desired,
4 % h0: initial distance from x,
5 % tol: relative target tolerance
6 h = h0;
7 % Aitken-Neville scheme, see Code 3.4.3 ( $x = 0!$ )
8 y(1) = (f(x+h0)-f(x-h0))/(2*h0);
9 for i=2:10
10     h(i) = h(i-1)/2;
11     y(i) = (f(x+h(i))-f(x-h(i)))/h(i-1);
12     for k=i-1:-1:1
13         y(k) = y(k+1) - (y(k+1)-y(k))*h(i)/(h(i)-h(k));
14     end
15     % termination of extrapolation, when desired tolerance is achieved
16     if (abs(y(2)-y(1)) < tol*abs(y(1))), break; end %
17 end
18 d = y(1);

```

diffex2 (@atan, 1.1, 0.5) diffex2 (@sqrt, 1.1, 0.5) diffex2 (@exp, 1.1, 0.5)

Degree	Relative error	Degree	Relative error	Degree	Relative error
0	0.04262829970946	0	0.02849215135713	0	0.04219061098749
1	0.02044767428982	1	0.01527790811946	1	0.02129207652215
2	0.00051308519253	2	0.00061205284652	2	0.00011487434095
3	0.00004087236665	3	0.00004936258481	3	0.00000825582406
4	0.00000048930018	4	0.00000067201034	4	0.00000000589624
5	0.00000000746031	5	0.00000001253250	5	0.00000000009546
6	0.00000000001224	6	0.00000000004816	6	0.00000000000002
		7	0.00000000000021		

advantage: guaranteed accuracy → efficiency



Additional explanations concerning the problems encountered in the numerical evaluation of difference quotients:

*Example 3.4.12* (Roundoff errors and difference quotients). → [13, Sect. 8.2.6] Ex. 3.4.9

Approximate derivative by difference quotient:  $f'(x) \approx \frac{f(x+h) - f(x)}{h}$ .

Calculus: better approximation for smaller  $h > 0$ , isn't it ?

MATLAB-CODE: Numerical differentiation of  $\exp(x)$

```
h = 0.1; x = 0.0;
for i = 1:16
    df = (exp(x+h) - exp(x)) / h;
    fprintf('%d %16.14f\n', i, df-1);
    h = h*0.1;
end
```

Recorded relative error,  $f(x) = e^x$ ,  $x = 0$  ▷

$\log_{10}(h)$	relative error
-1	0.05170918075648
-2	0.00501670841679
-3	0.00050016670838
-4	0.00005000166714
-5	0.00000500000696
-6	0.00000049996218
-7	0.00000004943368
-8	-0.00000000607747
-9	0.00000008274037
-10	0.00000008274037
-11	0.00000008274037
-12	0.00008890058234
-13	-0.00079927783736
-14	-0.00079927783736
-15	0.11022302462516
-16	-1.00000000000000

Note: An analysis based on expressions for remainder terms of Taylor expansions shows that the **approximation error** cannot be blamed for the loss of accuracy as  $h \rightarrow 0$  (as expected).

Explanation relying on roundoff error analysis, see Sect. 2.4:

MATLAB-CODE: Numerical differentiation of exp(x)

```
h = 0.1; x = 0.0;
for i = 1:16
    df = (exp(x+h) - exp(x)) / h;
    fprintf('%d %16.14f\n', i, df-1);
    h = h*0.1;
end
```

Obvious **cancellation** → error amplification

$\log_{10}(h)$	relative error
-1	0.05170918075648
-2	0.00501670841679
-3	0.00050016670838
-4	0.00005000166714
-5	0.00000500000696
-6	0.00000049996218
-7	0.00000004943368
-8	-0.00000000607747
-9	0.00000008274037
-10	0.00000008274037
-11	0.00000008274037
-12	0.00008890058234
-13	-0.00079927783736
-14	-0.00079927783736
-15	0.11022302462516
-16	-1.00000000000000

R. Hiptmair  
rev 38286,  
October 26,  
2011

$$f'(x) - \frac{f(x+h) - f(x)}{h} \rightarrow 0 \quad \left. \vphantom{\frac{f(x+h) - f(x)}{h}} \right\} \text{ for } h \rightarrow 0.$$

Impact of roundoff  $\rightarrow \infty$

Analysis for  $f(x) = \exp(x)$ :

$$\text{df} = \frac{e^{x+h} (1 + \delta_1) - e^x (1 + \delta_2)}{h}$$

correction factors take into account roundoff:  
(→ "axiom of roundoff analysis", Ass. 2.4.10)

$$= e^x \left( \frac{e^h - 1}{h} + \frac{\delta_1 e^h - \delta_2}{h} \right) \quad |\delta_1|, |\delta_2| \leq \text{eps} .$$

$$\Rightarrow |\text{df}| \leq e^x \left( \frac{e^h - 1}{h} + \text{eps} \frac{1 + e^h}{h} \right)$$

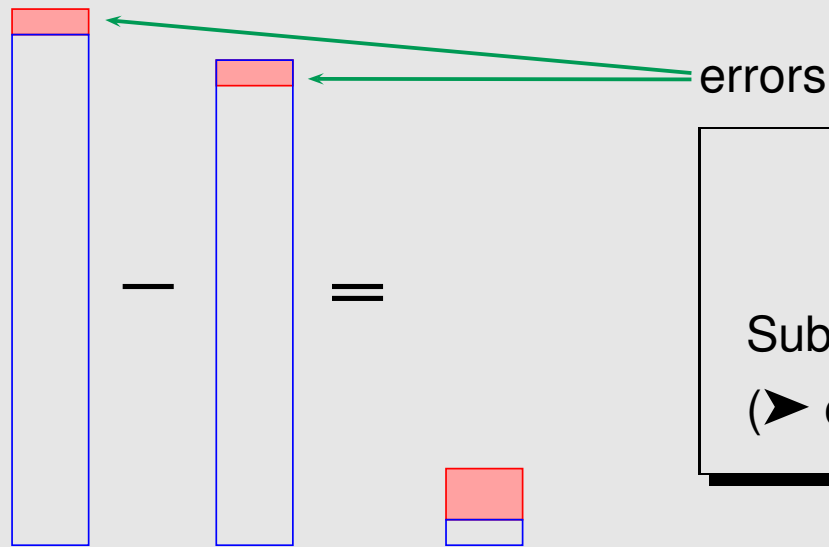
$1 + O(h) \quad O(h^{-1}) \quad \text{für } h \rightarrow 0$

▶ relative error:  $\left| \frac{e^x - \text{df}}{e^x} \right| \approx h + \frac{2\text{eps}}{h} \rightarrow \min \quad \text{for } h = \sqrt{2\text{eps}} .$

In double precision:  $\sqrt{2\text{eps}} = 2.107342425544702 \cdot 10^{-8}$



# What is this mysterious cancellation (*ger.:* Auslöschung) ?



**Cancellation**

≐

Subtraction of almost equal numbers  
 (▶ extreme amplification of relative errors)

*Example 3.4.13* (cancellation in decimal floating point arithmetic).

$x, y$  afflicted with relative errors  $\approx 10^{-7}$ :

$$\begin{array}{r}
 x = 0.123467* \\
 y = 0.123456* \\
 \hline
 x - y = 0.000011* = 0.11*000 \cdot 10^{-4}
 \end{array}$$

← 7th digit perturbed  
 ← 7th digit perturbed  
 ← 3rd digit perturbed

↑  
 padded zeroes



### 3.4.4 Newton basis and divided differences [13, Sect. 8.2.4], [51, Sect. 8.2]

Drawback of the Lagrange basis: adding another data point affects *all* basis polynomials!

```
1 class PolyEval {  
2   private :  
3   // evaluation point and various internal data describing the polynomials  
4   public :  
5     // Idle Constructor  
6     PolyEval();  
7     // Add another data point and update internal information  
8     void addPoint(t,y);  
9     // Evaluation of current interpolating polynomial at x  
10    double operator () (double x) const;  
11 };
```

The challenge: Both `addPoint()` and the evaluation operator may be called many times and the implementation has to remain efficient under these circumstances.

Tool: “update friendly” representation: **Newton basis** for  $\mathcal{P}_n$

$$N_0(t) := 1, \quad N_1(t) := (t - t_0), \quad \dots, \quad N_n(t) := \prod_{i=0}^{n-1} (t - t_i). \quad (3.4.14)$$

Note:  $N_n \in \mathcal{P}_n$  with *leading coefficient* 1.

➤ LSE for polynomial interpolation problem in Newton basis:

$$a_j \in \mathbb{R}: \quad a_0 N_0(t_j) + a_1 N_1(t_j) + \dots + a_n N_n(t_j) = y_j, \quad j = 0, \dots, n.$$

⇔ triangular linear system

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & (t_1 - t_0) & \dots & \vdots \\ \vdots & \vdots & \dots & 0 \\ 1 & (t_n - t_0) & \dots & \prod_{i=0}^{n-1} (t_n - t_i) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$



Solution of the system with forward substitution:

$$\begin{aligned}
 a_0 &= y_0, \\
 a_1 &= \frac{y_1 - a_0}{t_1 - t_0} = \frac{y_1 - y_0}{t_1 - t_0}, \\
 a_2 &= \frac{y_2 - a_0 - (t_2 - t_0)a_1}{(t_2 - t_0)(t_2 - t_1)} = \frac{y_2 - y_0 - (t_2 - t_0)\frac{y_1 - y_0}{t_1 - t_0}}{(t_2 - t_0)(t_2 - t_1)} = \frac{\frac{y_2 - y_0}{t_2 - t_0} - \frac{y_1 - y_0}{t_1 - t_0}}{t_2 - t_1}, \\
 &\vdots
 \end{aligned}$$

Observation: same quantities computed again and again !

In order to find a better algorithm, we turn to a new interpretation of the coefficients  $a_j$  of the interpolating polynomials in Newton basis.

Newton basis polynomial  $N_j(t)$ : degree  $j$  and leading coefficient 1  
 $\Rightarrow a_j$  is the leading coefficient of the interpolating polynomial  $p_{0,\dots,j}$

(notation  $p_{0,\dots,j}$  introduced in Sect. 3.4.2, see (3.4.3))

- Recursion (3.4.3) implies recursion for leading coefficients  $a_{\ell, \dots, m}$  of interpolating polynomials  $p_{\ell, \dots, m}$ ,  $0 \leq \ell \leq m \leq n$ :

$$a_{\ell, \dots, m} = \frac{a_{\ell+1, \dots, m} - a_{\ell, \dots, m-1}}{t_m - t_\ell}.$$

Simpler and more efficient algorithm using **divided differences**:

$$\begin{aligned} y[t_i] &= y_i \\ y[t_i, \dots, t_{i+k}] &= \frac{y[t_{i+1}, \dots, t_{i+k}] - y[t_i, \dots, t_{i+k-1}]}{t_{i+k} - t_i} \quad (\text{recursion}) \end{aligned} \quad (3.4.15)$$

Recursive calculation by **divided differences scheme**, cf. Aitken-Neville scheme, Code 3.4.3:

$$\begin{array}{l|l} t_0 & y[t_0] \\ & > y[t_0, t_1] \\ t_1 & y[t_1] \\ & > y[t_0, t_1, t_2] \\ & > y[t_1, t_2] \\ & > y[t_0, t_1, t_2, t_3], \\ t_2 & y[t_2] \\ & > y[t_1, t_2, t_3] \\ & > y[t_2, t_3] \\ t_3 & y[t_3] \end{array} \quad (3.4.16)$$

the elements are computed from left to right, every “>” means recursion (3.4.15).

If a new datum  $(t_{n+1}, y_{n+1})$  is added, it is enough to compute  $n + 2$  new terms

$$y[t_{n+1}], y[t_n, t_{n+1}], \dots, y[t_0, \dots, t_{n+1}].$$

Code 3.4.17: Divided differences, recursive implementation, in situ computation

```

1 function y = divdiff(t, y)
2 n = length(y) - 1;
3 if (n > 0)
4     y(1:n) = divdiff(t(1:n), y(1:n));
5     for j=0:n-1
6         y(n+1) = (y(n+1) - y(j+1)) / (t(n+1) - t(j+1));
7     end
8 end

```

By derivation: computed finite differences are the coefficients of interpolating polynomials in Newton basis:

$$p(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)(t - t_1) + \dots + a_n \prod_{j=0}^{n-1} (t - t_j) \quad (3.4.18)$$

$$a_0 = y[t_0], \quad a_1 = y[t_0, t_1], \quad a_2 = y[t_0, t_1, t_2], \quad \dots$$

“Backward evaluation” of  $p(t)$  in the spirit of Horner’s scheme ( $\rightarrow$  Rem. 3.2.4, [13, Alg. 8.20]):

$$p \leftarrow a_n, \quad p \leftarrow (t - t_{n-1})p + a_{n-1}, \quad p \leftarrow (t - t_{n-2})p + a_{n-2}, \quad \dots$$

Code 3.4.19: Divided differences evaluation by modified Horner scheme

```

1 function p = evaldivdiff(t, y, x)
2 dd=divdiff(t, y); % Compute divided differences, see Code 3.4.16
3 n = length(y)-1;
4 p=dd(n+1);
5 for j=n:-1:1, p = (x-t(j)).*p+dd(j); end

```

Computational effort:

- $O(n^2)$  for computation of divided differences,
- $O(n)$  for every single evaluation of  $p(t)$ .

*Example 3.4.20* (Class PolyEval).

Implementation of a C++ class supporting the efficient update and evaluation of an interpolating polynomial making use of

- presentation in Newton basis (3.4.14),
- computation of representation coefficients through divided difference scheme (3.4.16), see Code 3.4.16,
- evaluation by means of Horner scheme, see Code 3.4.18.

#### Code 3.4.21: “Update friendly” polynomial interpolant

```
1 class PolyEval {  
2 private :  
3     std::vector<double> t;    // Interpolation nodes  
4     std::vector<double> y;    // Coefficients in Newton representation  
5 public :  
6     PolyEval(); // Idle constructor  
7     void addPoint(double t, double y); // Add another data point  
8     // evaluate value of current interpolating polynomial at x,  
9     double operator() (double x) const;  
10 private :  
11     // Update internal representation, called by addPoint()  
12     void divdiff();
```

```
13 };
14
15 PolyEval::PolyEval() {}
16
17 void PolyEval::addPoint(double td, double yd)
18 { t.push_back(td); y.push_back(yd); divdiff(); }
19
20 // Update coefficients in Newton basis representation, cf. Code 3.4.16
21 void PolyEval::divdiff() {
22     int n = t.size();
23     for(int j=0; j<n-1; j++) y[n-1] =
24         ((y[n-1]-y[j]) / (t[n-1]-t[j]));
25 }
26
27 double PolyEval::operator() (double x) const {
28     double s = y.back();
29     for(int i = y.size()-2; i >= 0; --i) s = s*(x-t[i])+y[i];
30     return s;
31 }
```



*Remark 3.4.22* (Divided differences and derivatives).

If  $y_0, \dots, y_n$  are the values of a smooth function  $f$  in the points  $t_0, \dots, t_n$ , that is,  $y_j := f(t_j)$ , then

$$y[t_i, \dots, t_{i+k}] = \frac{f^{(k)}(\xi)}{k!}$$

for a certain  $\xi \in [t_i, t_{i+k}]$ , see [13, Thm. 8.21].



## 3.5 Polynomial Interpolation: Sensitivity [51, Sect. 8.1.3]

This section addresses a *major shortcoming of polynomial interpolation* in case the interpolation knots  $t_i$  are imposed, which is usually the case when given data points have to be interpolated, *cf.* Ex. 3.1.2.

The next example gives a glimpse of the problems haunting polynomial interpolation.

Example 3.5.1 (Oscillating interpolation polynomial: Runge's counterexample). → [13, Sect. 8.3], [51, Ex. 8.1]

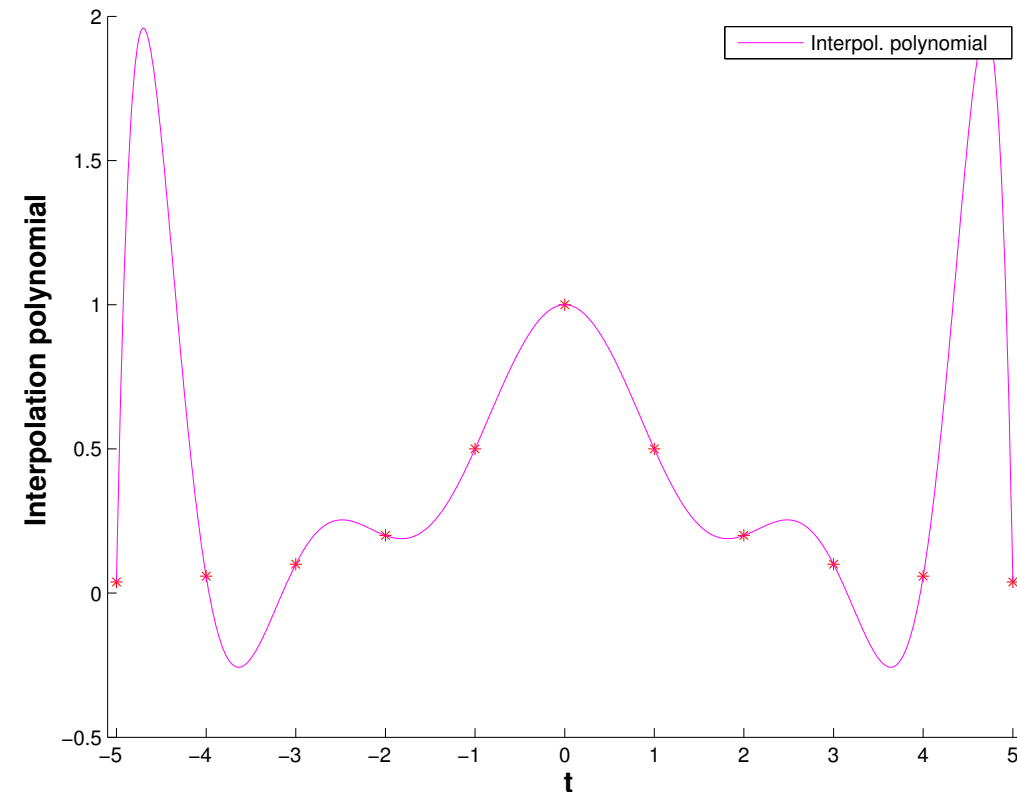
Interpolation polynomial with uniformly spaced nodes:

$$\mathcal{T} := \left\{ -5 + \frac{10}{n} j \right\}_{j=0}^n,$$

$$y_j = \frac{1}{1 + t_j^2}, \quad j = 0, \dots, n.$$

Plot  $n = 10 \rightarrow$

See example 9.1.5.



possible strong oscillations of interpolating polynomials  
of *high degree* on *uniformly spaced* nodes!





Sect. 2.5.4 discussed the conditioning/sensitivity of linear systems of equations: This concepts tries to capture how perturbations in the data affect the output of a problem map.

Here: data  $\leftrightarrow$  values  $y_i$   
problem map  $\leftrightarrow$  polynomial interpolation mapping  $I_{\mathcal{T}}$ , see Thm. 3.3.7

“The (pointwise) conditioning of interpolation tells us to what extent perturbations in the  $y$ -data will affect the values of the interpolating function somewhere else. In the case of ill-conditioning small perturbations in the data can cause big variations in some function values, which is clearly undesirable.”

Necessary for studying the conditioning: norms ( $\rightarrow$  Def. 2.5.1) on vector space of continuous functions  $C(I)$ ,  $I \subset \mathbb{R}$

Note relationship with vector norms  $\rightarrow$  Sect. 2.5.1.

$$\text{supremum norm} \quad \|f\|_{L^\infty(I)} := \sup\{|f(t)| : t \in I\}, \quad (3.5.2)$$

$$\text{\textit{L}^2\text{-norm}} \quad \|f\|_{L^2(I)}^2 := \int_I |f(t)|^2 dt, \quad (3.5.3)$$

$$\text{\textit{L}^1\text{-norm}} \quad \|f\|_{L^1(I)} := \int_I |f(t)| dt. \quad (3.5.4)$$

**Lemma 3.5.5** (Absolute conditioning of polynomial interpolation).

Given a mesh  $\mathcal{T} \subset \mathbb{R}$  with generalized Lagrange polynomials  $L_i, i = 0, \dots, n$ , and fixed  $I \subset \mathbb{R}$ , the norm of the interpolation operator satisfies

$$\|\mathcal{I}_{\mathcal{T}}\|_{\infty \rightarrow \infty} := \sup_{\mathbf{y} \in \mathbb{K}^{n+1} \setminus \{0\}} \frac{\|\mathcal{I}_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)}}{\|\mathbf{y}\|_\infty} = \left\| \sum_{i=0}^n |L_i| \right\|_{L^\infty(I)}, \quad (3.5.6)$$

$$\|\mathcal{I}_{\mathcal{T}}\|_{2 \rightarrow 2} := \sup_{\mathbf{y} \in \mathbb{K}^{n+1} \setminus \{0\}} \frac{\|\mathcal{I}_{\mathcal{T}}(\mathbf{y})\|_{L^2(I)}}{\|\mathbf{y}\|_2} \leq \left( \sum_{i=0}^n \|L_i\|_{L^2(I)}^2 \right)^{\frac{1}{2}}. \quad (3.5.7)$$

*Proof.* (for the  $L^\infty$ -Norm) By  $\triangle$ -inequality

$$\|\mathcal{I}_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)} = \left\| \sum_{j=0}^n y_j L_j \right\|_{L^\infty(I)} \leq \sup_{t \in I} \sum_{j=0}^n |y_j| |L_j(t)| \leq \|\mathbf{y}\|_\infty \left\| \sum_{i=0}^n |L_i| \right\|_{L^\infty(I)},$$

equality in (3.5.6) for  $\mathbf{y} := (\text{sgn}(L_j(t^*)))_{j=0}^n$ ,  $t^* := \text{argmax}_{t \in I} \sum_{i=0}^n |L_i(t)|$ . □

*Proof.* (for the  $L^2$ -Norm) By  $\triangle$ -inequality and Cauchy-Schwarz inequality

$$\|\mathbf{l}_{\mathcal{T}}(\mathbf{y})\|_{L^2(I)} \leq \sum_{j=0}^n |y_j| \|L_j\|_{L^2(I)} \leq \left( \sum_{j=0}^n |y_j|^2 \right)^{\frac{1}{2}} \left( \sum_{j=0}^n \|L_j\|_{L^2(I)}^2 \right)^{\frac{1}{2}}. \quad \square$$

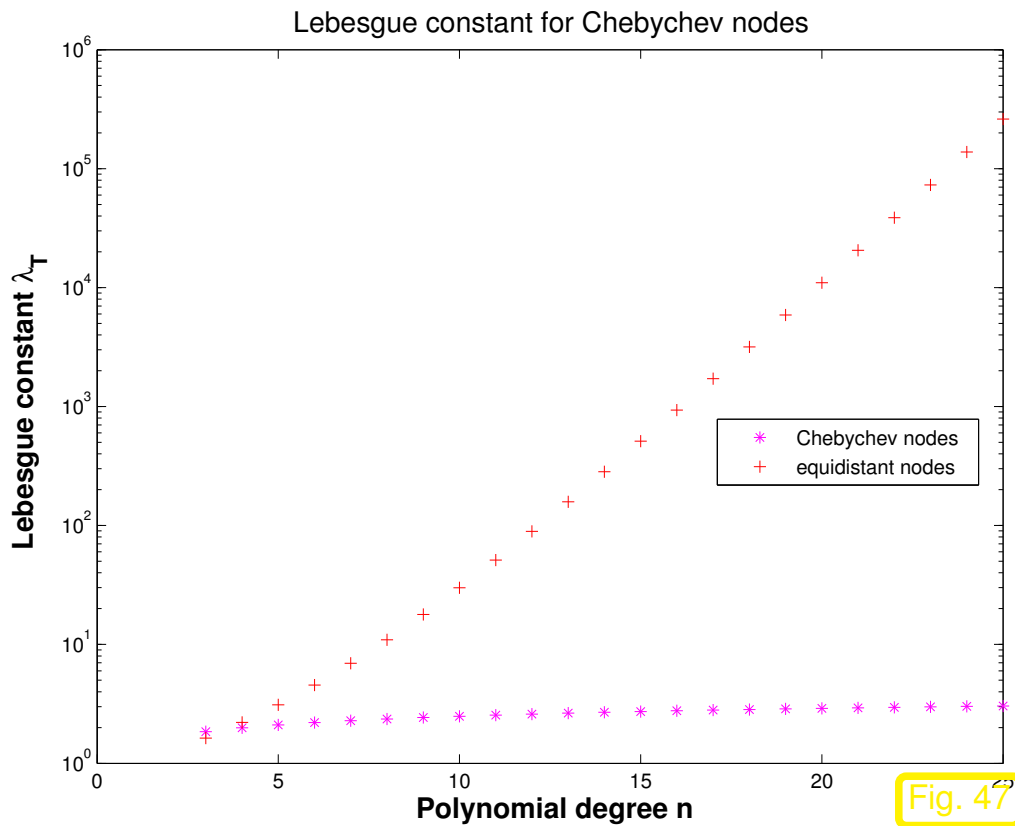
Terminology: **Lebesgue constant** of  $\mathcal{T}$ :  $\lambda_{\mathcal{T}} := \left\| \sum_{i=0}^n |L_i| \right\|_{L^\infty(I)}$

*Remark 3.5.8* (Lebesgue constant for equidistant nodes).

$$I = [-1, 1], \quad \mathcal{T} = \left\{ -1 + \frac{2k}{n} \right\}_{k=0}^n \quad (\text{uniformly spaced nodes})$$

Asymptotic estimate (with (3.3.2) and Stirling formula): for  $n = 2m$

$$|L_m(1 - \frac{1}{n})| = \frac{\frac{1}{n} \cdot \frac{1}{n} \cdot \frac{3}{n} \dots \frac{n-3}{n} \cdot \frac{n+1}{n} \dots \frac{2n-1}{n}}{\left(\frac{2}{n} \cdot \frac{4}{n} \dots \frac{n-2}{n} \cdot 1\right)^2} = \frac{(2n)!}{(n-1)2^{2n}((n/2)!)^2 n!} \sim \frac{2^{n+3/2}}{\pi(n-1)n}$$



Theory [11]: for uniformly spaced nodes

$$\lambda_{\mathcal{T}} \geq C e^{n/2}$$

for  $C > 0$  independent of  $n$ .

## Numerical evaluation of the expression

$$\lambda_{\mathcal{T}} = \left\| \sum_{i=0}^n |L_i| \right\|_{L^\infty(I)},$$

for the Lebesgue constant of polynomial interpolation, see Lemma 3.5.5.

Code 3.5.9: MATLAB code for *approximate* computation of Lebesgue constants

```

1 function l = lebesgue(t,N)
2 % Computation of Lebesgue constant of polynomial interpolation with
3 % knots  $t_i$  passed in the row vector  $t$  based on
4 % (3.5.6).  $N$  specifies the number of sampling points for the approximate
5 % computation of the maximum norm of the Lagrange polynomial on the interval
6 %  $[-1,1]$ .
7 n = length(t);
8 den = []; % denominators of normalized Lagrange polynomials relative to the
   nodes t
9 for i=1:n
10     den = [den; prod(t(i)-t([1:i-1,i+1:end]))];
11 end
12
13 % Default argument
14 if (nargin < 2), N = 1E5; end
15 l = 0; % return value
16 for x=-1:2/N:1 % sampling points for approximate computation of  $\|\cdot\|_{L^\infty([-1,1])}$ 
17     s = 0;
```

```
18 for i = 1:n
19     % v provides value of the normalized Lagrange polynomials
20     v = prod(x-t([1:i-1,i+1:end]))/den(i);
21     s = s+abs(v); % sum over the modulus of the polynomials
22 end
23 l = max(l,s); % maximum of sampled values
24 end
```

Note: Approximation of  $\|L_i\|_{L^\infty(I)}$  maximum modulus of point vales.



In Ex. 3.1.2 we learned that interpolation is an important technique for obtaining a mathematical (and algorithmic) description of a constitutive relationship from measured data.

If the interpolation operator is poorly conditioned, tiny measurement errors will lead to big (local) deviations of the interpolant from its “true” form.

Since measurement errors are inevitable, poorly conditioned interpolation procedures are useless for determining constitutive relationships from measurements.

## 3.6 Shape preserving interpolation

When reconstructing a quantitative dependence of quantities from measurements, first principles from physics often stipulate qualitative constraints, which translate into *shape properties* of the function  $f$ , e.g., when modelling the material law for a gas:

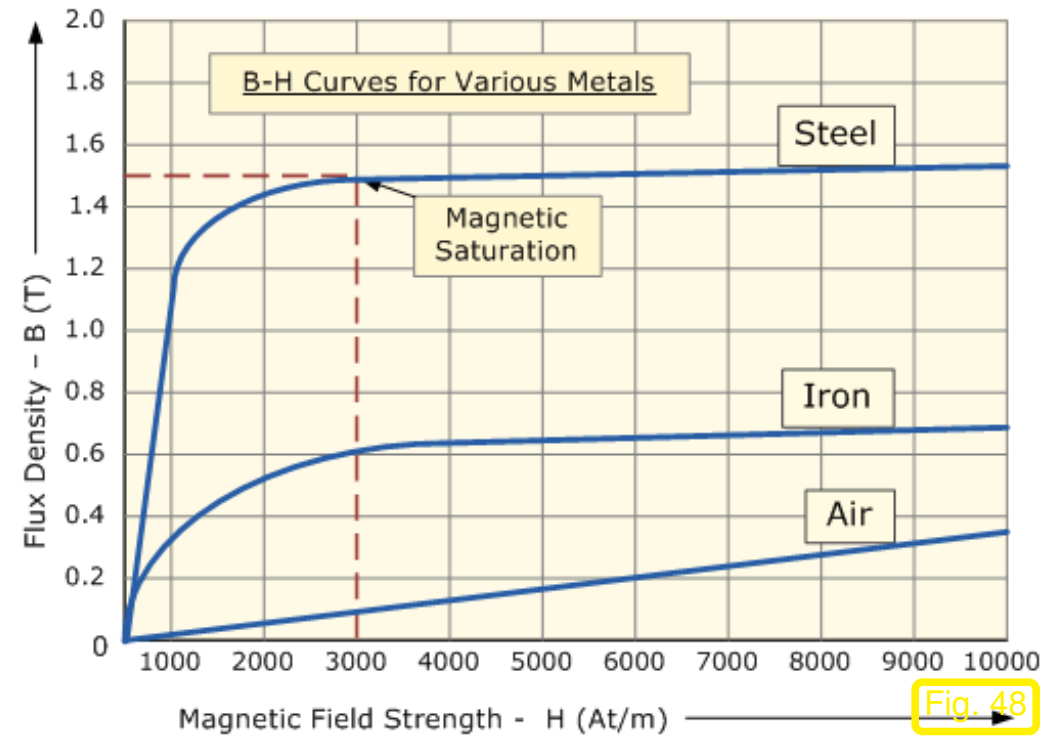
$t_i$  pressure values,  $y_i$  densities  $\triangleright f$  positive & monotone.

Notation: given data:  $(t_i, y_i) \in \mathbb{R}^2, i = 0, \dots, n, n \in \mathbb{N}, t_0 < t_1 < \dots < t_n.$

**Example 3.6.1** (Magnetization curves).

For many materials physics stipulates properties of the functional dependence of magnetic flux  $B$  from magnetic field strength  $H$ :

- $H \mapsto B(H)$  monotone (increasing),
- $H \mapsto B(H)$  concave

**Definition 3.6.2** (monotonic data).

The data  $(t_i, y_i)$  are called *monotonic* when  $y_i \geq y_{i-1}$  or  $y_i \leq y_{i-1}$ ,  $i = 1, \dots, n$ .



**Definition 3.6.3** (Convex/concave data).

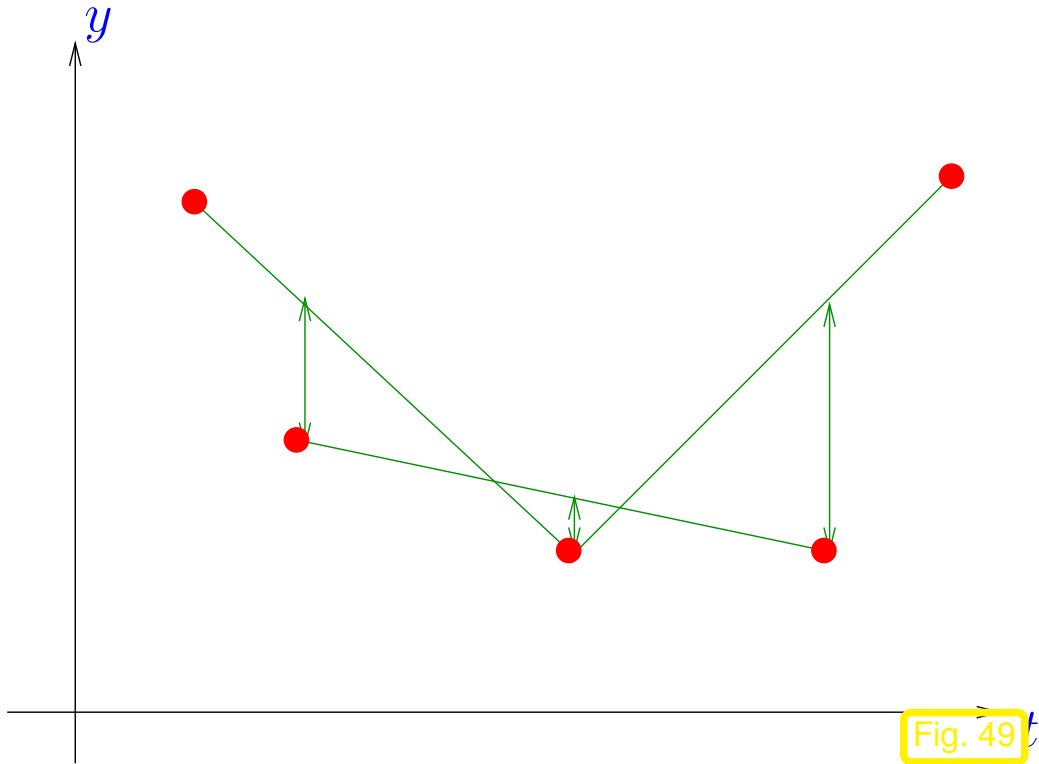
The data  $\{(t_i, y_i)\}_{i=0}^n$  are called **convex** (**concave**) if

$$\Delta_j \stackrel{(\geq)}{\leq} \Delta_{j+1}, \quad j = 1, \dots, n-1, \quad \Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}}, \quad j = 1, \dots, n.$$

Mathematical characterization of convex data:

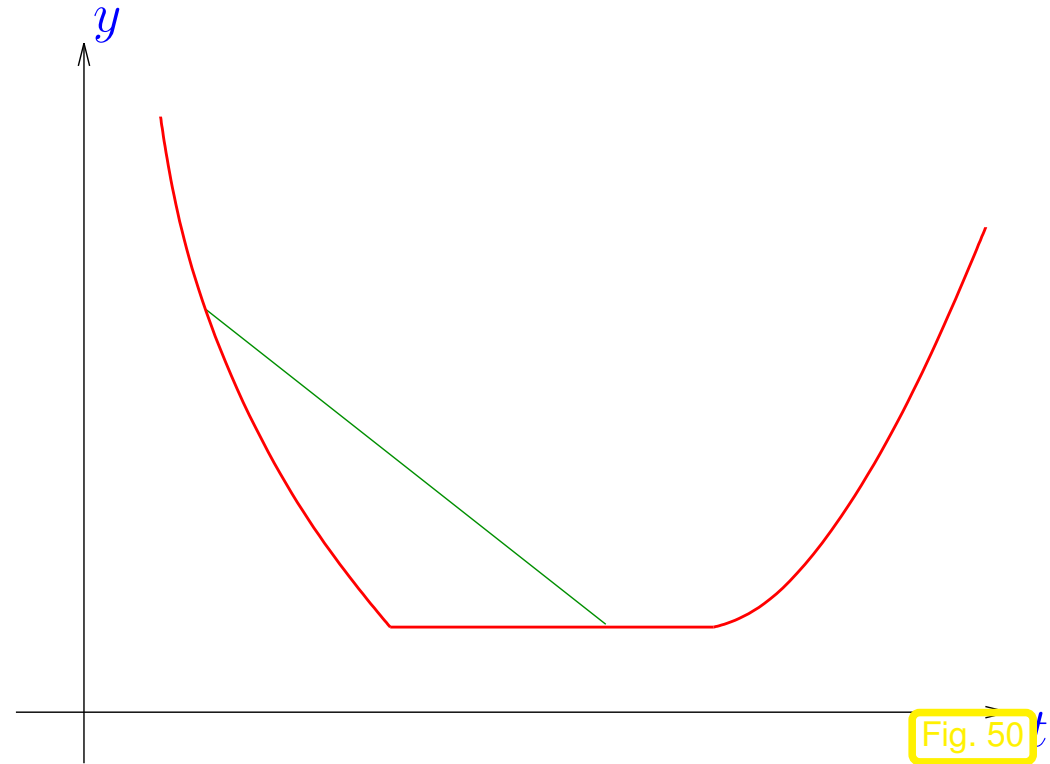
$$y_i \leq \frac{(t_{i+1} - t_i)y_{i-1} + (t_i - t_{i-1})y_{i+1}}{t_{i+1} - t_{i-1}} \quad \forall i = 1, \dots, n-1,$$

i.e., each data point lies below the line segment connecting the other data, *cf.* definition of convexity of a function [63, Def. 5.5.2].



Convex data

Fig. 49



Convex function

Fig. 50

**Definition 3.6.4** (Convex/concave function). [63, Def. 5.5.2]  $\rightarrow$

$$\begin{array}{l}
 f : I \subset \mathbb{R} \mapsto \mathbb{R} \\
 \text{convex} \\
 \text{concave}
 \end{array}
 \Leftrightarrow
 \begin{array}{l}
 f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y) \\
 f(\lambda x + (1 - \lambda)y) \geq \lambda f(x) + (1 - \lambda)f(y)
 \end{array}
 \quad \forall 0 \leq \lambda \leq 1, \forall x, y \in I.$$

Now consider interpolation: data  $(t_i, y_i), i = 0, \dots, n$   $\longrightarrow$  interpolant  $f$

Goal: **shape preserving interpolation:**

positive data	$\longrightarrow$	positive interpolant $f$ ,
monotonic data	$\longrightarrow$	monotonic interpolant $f$ ,
convex data	$\longrightarrow$	convex interpolant $f$ .

More ambitious goal: **local shape preserving interpolation:** for each subinterval  $I = (t_i, t_{i+j})$

positive data in $I$	$\longrightarrow$	locally positive interpolant $f _I$ ,
monotonic data in $I$	$\longrightarrow$	locally monotonic interpolant $f _I$ ,
convex data in $I$	$\longrightarrow$	locally convex interpolant $f _I$ .

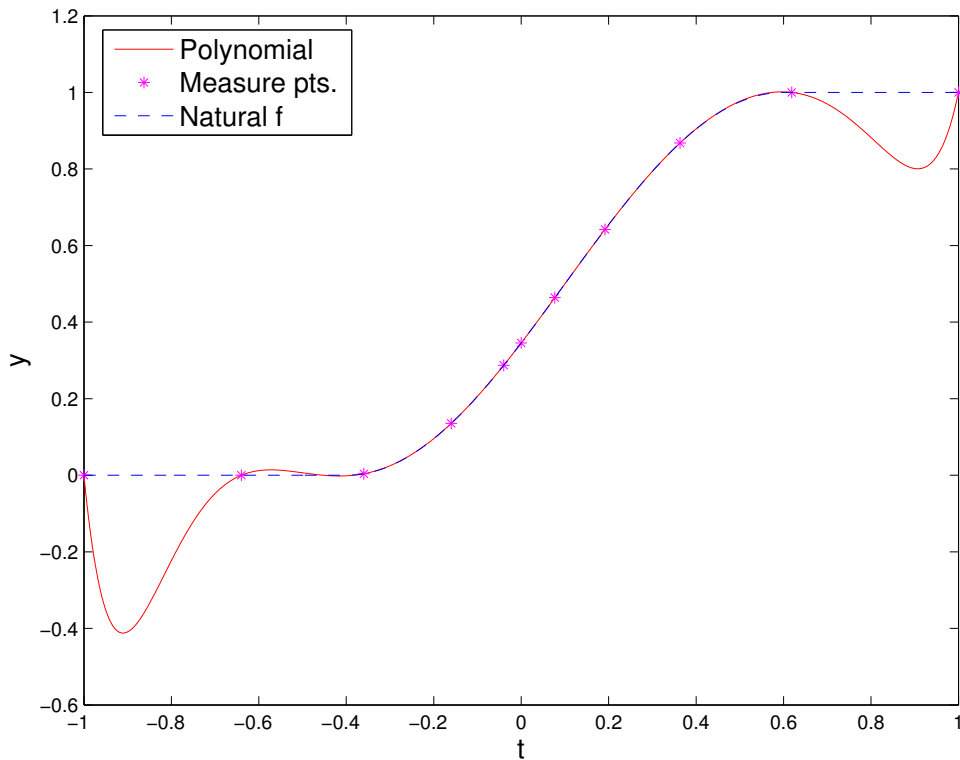
*Example 3.6.5* (Bad behavior of global polynomial interpolants).

Positive and monotonic data:

$t_i$	-1.0000	-0.6400	-0.3600	-0.1600	-0.0400	0.0000	0.0770	0.1918	0.3631	0.6187	1.0000
$y_i$	0.0000	0.0000	0.0039	0.1355	0.2871	0.3455	0.4639	0.6422	0.8678	1.0000	1.0000

created by taking points on the graph of

$$f(t) = \begin{cases} 0 & \text{if } t < -\frac{2}{5}, \\ \frac{1}{2}(1 + \cos(\pi(t - \frac{3}{5}))) & \text{if } -\frac{2}{5} < t < \frac{3}{5}, \\ 1 & \text{otherwise.} \end{cases}$$



← Interpolating polynomial, degree = 10

Oscillations at the endpoints of the interval (see Ex. 9.1.5)

- No locality
- No positivity
- No monotonicity
- No local conservation of the curvature

## 3.6.1 Piecewise linear interpolation

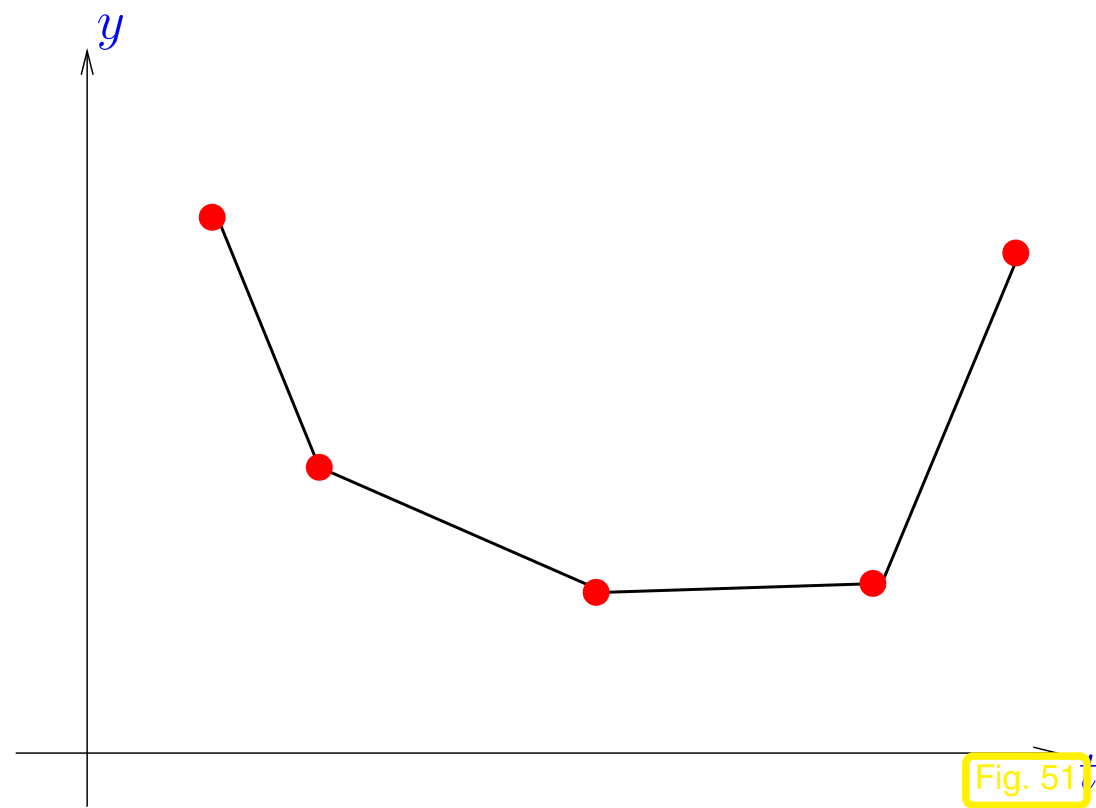
There is a very simple method of achieving perfect shape preservation by means of a linear ( $\rightarrow$  Rem. 3.1.8) interpolation operator into the space of continuous functions:

Data:  $(t_i, y_i) \in \mathbb{R}^2, i = 0, \dots, n, n \in \mathbb{N}, t_0 < t_1 < \dots < t_n.$

Piecewise linear interpolant, *cf.* Ex. 3.1.5:

$$s(x) = \frac{(t_{i+1} - t)y_i + (t - t_i)y_{i+1}}{t_{i+1} - t_i} \quad t \in [t_i, t_{i+1}].$$

Piecewise linear interpolant of data in Fig. 49:



Piecewise linear interpolation means simply “connect the data points in  $\mathbb{R}^2$  using straight lines”.

Obvious: linear interpolation is **linear** (as mapping  $\mathbf{y} \mapsto s$ ) and **local**:

$$y_j = \delta_{ij} , \quad i, j = 0, \dots, n \quad \Rightarrow \quad \text{supp}(s) \subset [t_{i-1}, t_{i+1}] .$$

**Theorem 3.6.6** (**Local shape preservation** by piecewise linear interpolation).

Let  $s \in C([t_0, t_n])$  be the piecewise linear interpolant of  $(t_i, y_i) \in \mathbb{R}^2$ ,  $i = 0, \dots, n$ , for every subinterval  $I = [t_j, t_k] \subset [t_0, t_n]$ :

if  $(t_i, y_i)|_I$  are positive/negative  $\Rightarrow s|_I$  is positive/negative,  
 if  $(t_i, y_i)|_I$  are monotonic (increasing/decreasing)  $\Rightarrow s|_I$  is monotonic (increasing/decreasing),  
 if  $(t_i, y_i)|_I$  are convex/concave  $\Rightarrow s|_I$  is convex/concave.

Local shape preservation = perfect shape preservation!

Bad news: none of these properties carries over to local polynomial interpolation of higher polynomial degree  $d > 1$ .

*Example 3.6.7* (Piecewise quadratic interpolation).

Known (Thm. 3.3.5): Parabola (polynomial of degree 2) uniquely determined by 3 data points  $\Rightarrow$  form groups of three adjacent data points and interpolate each of these triplets by a 2nd-degree polynomial (parabola).

Assume:  $n = 2m$  even

► **piecewise quadratic interpolant**  $q : [\min\{t_i\}, \max\{t_i\}] \mapsto \mathbb{R}$  defined by

$$q_j := q|_{[t_{2j-2}, t_{2j}]} \in \mathcal{P}_2 \quad , \quad q_j(t_i) = y_i \quad , \quad i = 2j - 2, 2j - 1, 2j \quad , \quad j = 1, \dots, m \quad . \quad (3.6.8)$$

Nodes as in Ex. 3.6.5

Piecewise linear/quadratic interpolation



No shape preservation for piecewise quadratic interpolant

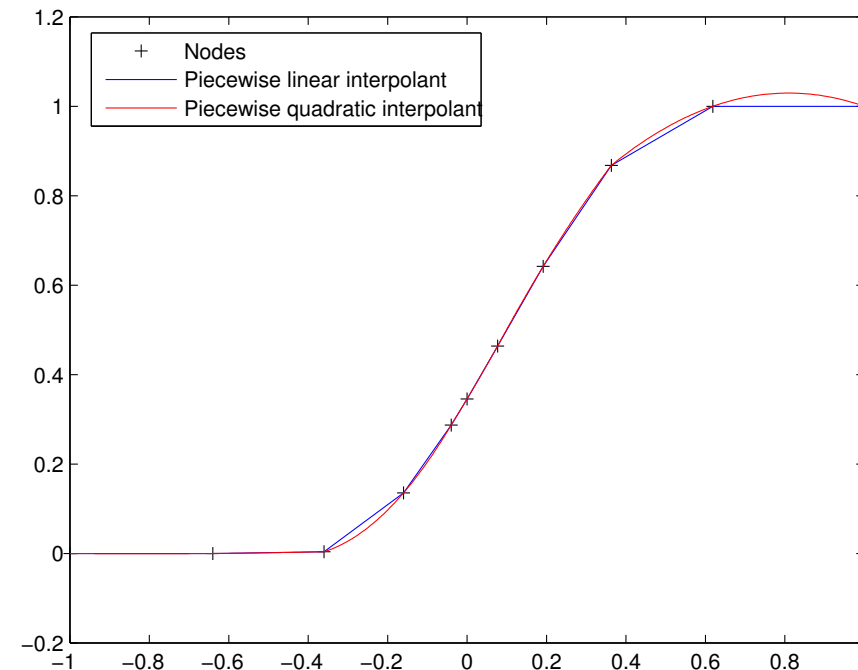


Fig. 52





The “only” drawback of piecewise linear interpolation:

interpolant is only  $C^0$  but not  $C^1$  (no continuous derivative).

Note: Interpolant usually serves as input for other numerical methods like a Newton-method for solving non-linear systems of equations, see Sect. 4.4, which requires derivatives.

## 3.7 Cubic Hermite Interpolation

Aim: construct local shape-preserving ( $\rightarrow$  Sect. 3.6) (linear ?) interpolation operator that fixes shortcoming of piecewise linear interpolation by ensuring  $C^1$ -smoothness of the interpolant.

 notation:  $C^1([a, b]) \hat{=}$  space of *continuously differentiable* functions  $[a, b] \mapsto \mathbb{R}$ .

### 3.7.1 Definition and algorithms

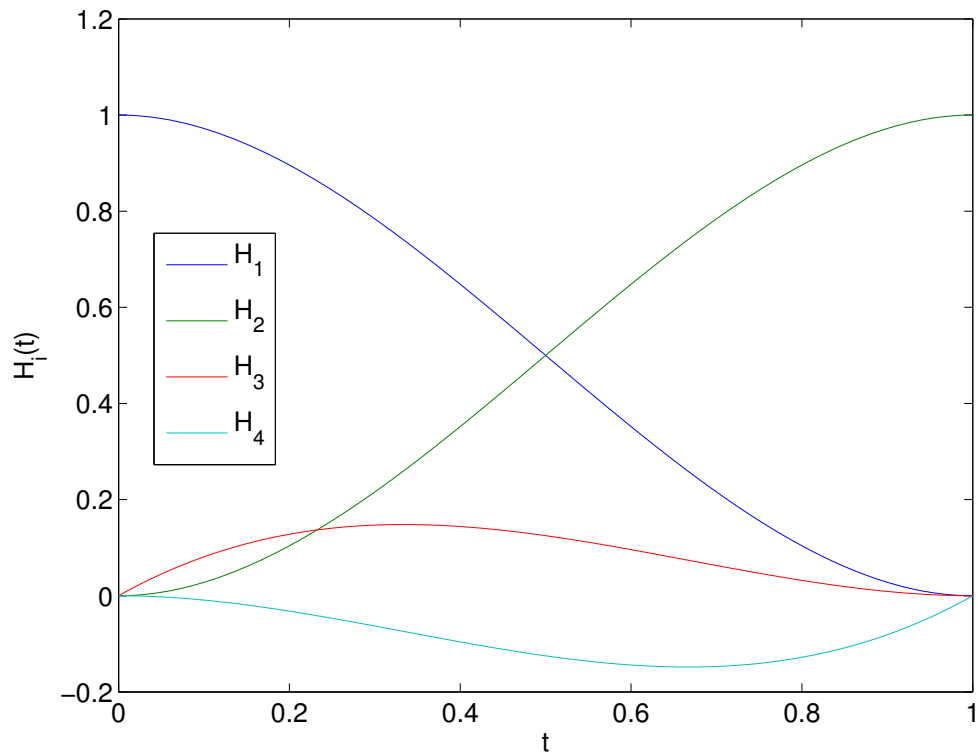
Given: mesh points  $(t_i, y_i) \in \mathbb{R}^2, i = 0, \dots, n, t_0 < t_1 < \dots < t_n$

Goal: function  $f \in C^1([t_0, t_n]), f(t_i) = y_i, i = 0, \dots, n$

→ Piecewise **cubic Hermite interpolation polynomial**  $s \in C^1([t_0, t_n])$ :  
with given **slopes**  $c_i \in \mathbb{R}, i = 0, \dots, n$

$$s|_{[t_{i-1}, t_i]} \in \mathcal{P}_3, \quad i = 1, \dots, n, \quad \boxed{s(t_i) = y_i}, \quad \boxed{s'(t_i) = c_i}, \quad i = 0, \dots, n.$$

$$\blacktriangleright \quad s(t) = y_{i-1}H_1(t) + y_iH_2(t) + c_{i-1}H_3(t) + c_iH_4(t), \quad t \in [t_{i-1}, t_i], \quad (3.7.1)$$



$$\begin{aligned}
 H_1(t) &:= \phi\left(\frac{t_i-t}{h_i}\right), & H_2(t) &:= \phi\left(\frac{t-t_{i-1}}{h_i}\right), \\
 H_3(t) &:= -h_i\psi\left(\frac{t_i-t}{h_i}\right), & H_4(t) &:= h_i\psi\left(\frac{t-t_{i-1}}{h_i}\right), \\
 h_i &:= t_i - t_{i-1}, \\
 \phi(\tau) &:= 3\tau^2 - 2\tau^3, \\
 \psi(\tau) &:= \tau^3 - \tau^2.
 \end{aligned}
 \tag{3.7.2}$$

◁ Local basis polynomial on  $[0, 1]$

By tedious, but straightforward computations using the chain rule:

	$H(t_{i-1})$	$H(t_i)$	$H'(t_{i-1})$	$H'(t_i)$
$H_1$	1	0	0	0
$H_2$	0	1	0	0
$H_3$	0	0	1	0
$H_4$	0	0	0	1

Code 3.7.3: Local evaluation of cubic Hermite polynomial

Piecewise cubic polynomial  $s$  on  $t_1, t_2$  with  $s(t_1) = y_1, s(t_2) = y_2, s'(t_1) = c_1, s'(t_2) = c_2$ :

efficient local evaluation



```

1 function
   s=hermloceval(t,t1,t2,y1,y2,c1,c2)
2 % y1, y2: data values, c1, c2: slopes
3 h = t2-t1; t = (t-t1)/h;
4 a1 = y2-y1; a2 = a1-h*c1;
5 a3 = h*c2-a1-a2;
6 s = y1+(a1+(a2+a3*t).* (t-1)).*t;

```

How to choose the slopes  $c_i$  ?

Natural attempt: (weighted) average of local slopes:

$$c_i = \begin{cases} \Delta_1 & , \text{ for } i = 0 , \\ \Delta_n & , \text{ for } i = n , \\ \frac{t_{i+1}-t_i}{t_{i+1}-t_{i-1}}\Delta_i + \frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}}\Delta_{i+1} & , \text{ if } 1 \leq i < n . \end{cases} , \quad \Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}} , j = 1, \dots, n .$$

(3.7.4)



**Linear** local interpolation operator

See (3.7.8) for a different choice of the slopes.

*Example 3.7.5* (Piecewise cubic Hermite interpolation).

Data points:

- 11 equispaced nodes

$$t_j = -1 + 0.2 j, \quad j = 0, \dots, 10.$$

in the interval  $I = [-1, 1]$ ,

- $y_i = f(t_i)$  with

$$f(x) := \sin(5x) e^x.$$

Use of weighted averages of slopes as  
in (3.7.4).

See Code 3.7.5.

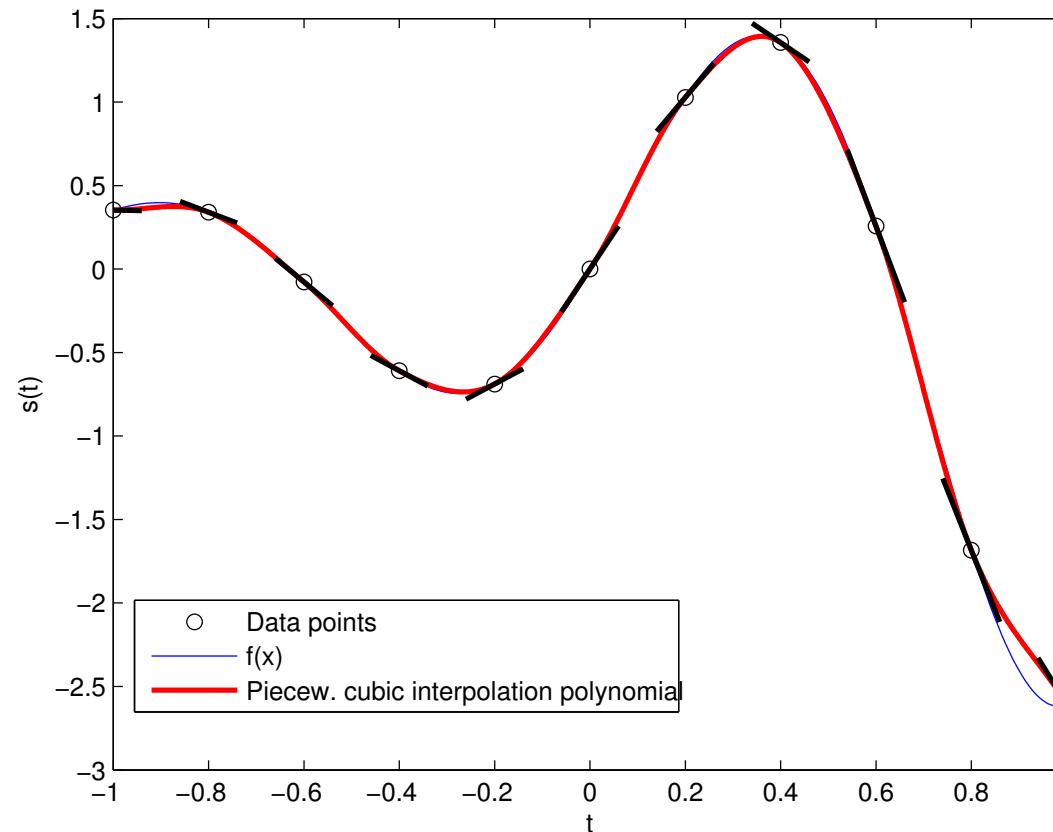


Fig. 53

No preservation of monotonicity!

## Code 3.7.6: Piecewise cubic Hermite interpolation

```

1 function hermintp1(f,t)
2 % compute and plot the cubic Hermite interpolant of the function f in the nodes
  t
3 % using weighted averages according to (3.7.4) as local slopes
4 n = length (t); h = diff (t); % computes lengths of intervals between
  nodes:  $h_i := t_{i+1} - t_i$ 
5 y = f(t); % f must support collective evaluation for row vector argument
6 delta = diff (y) ./ h; % slopes of piecewise linear interpolant
7 c = [delta(1), ...
      ((h(2:end) .* delta(1:end-1) + h(1:end-1) .* delta(2:end)) ...
      ./ (t(3:end) - t(1:end-2)) ], ...
      delta(end)] ; % slopes from weighted average, see (3.7.4)
11
12 figure ('Name', 'Hermite Interpolation');
13 plot (t,y,'ko'); hold on; % plot data points
14 fplot (f, [t(1), t(n)]);
15 for j=1:n-1 % compute and plot the Hermite interpolant with slopes c
16 vx = linspace (t(j),t(j+1), 100);
17 plot (vx,hermloceval(vx,t(j),t(j+1),y(j),y(j+1),c(j),c(j+1))),'r-',
      'LineWidth',2);
18 end
19 for j=2:n-1 % plot segments indicating the slopes  $c_i$ 

```

```
20     plot ([t(j)-0.3*h(j-1),t(j)+0.3*h(j)],...
21           [y(j)-0.3*h(j-1)*c(j),y(j)+0.3*h(j)*c(j)],'k-','LineWidth',2)
22 end
23 plot ([t(1),t(1)+0.3*h(1)], [y(1),y(1)+0.3*h(1)*c(1)], 'k-',
24       'LineWidth',2);
25 plot ([t(end)-0.3*h(end),t(end)], [y(end)-0.3*h(end)*c(end),y(end)], 'k-',
26       'LineWidth',2);
27 xlabel ('t'); ylabel ('s(t)');
28 legend ('Data points','f(x)','Piecew. cubic interpolation
29         polynomial ');
30 hold off;
```

Invocation: `hermintp1 (@(x) sin(5*x).*exp(x), [-1:0.2:1]);`

### 3.7.2 Shape preserving Hermite interpolation

Slopes according to (3.7.4)  $\Rightarrow$  Hermite interpolation does not preserve monotonicity.

Remedy: choice of the slopes  $c_i$  via “limiter”  $\rightarrow$  Preservation of monotonicity.

Geometric considerations lead to the necessary condition (If  $C^1$ -interpolant was not flat, an “overshoot” at a local maximum of the data could not be avoided):

$$c_i = \begin{cases} 0 & , \text{ if } \text{sgn}(\Delta_i) \neq \text{sgn}(\Delta_{i+1}) , \\ \text{weighted average of } \Delta_i, \Delta_{i+1} & \text{otherwise} \end{cases} , \quad i = 1, \dots, n - 1 . \quad (3.7.7)$$

R. Hiptmair  
rev 38286,  
September  
26, 2011

However, we also want the reconstruction slope  $c_i \rightarrow 0$  if one of the adjacent slopes tends to zero!

Which kind of average ?

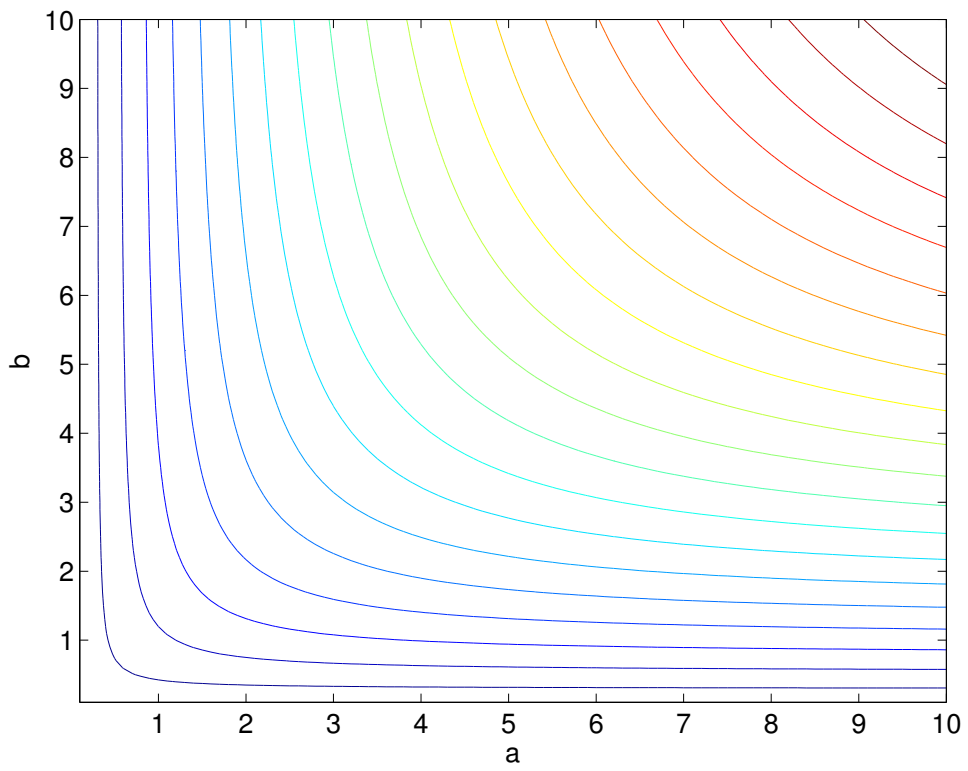
$$c_i = \frac{1}{\frac{w_a}{\Delta_i} + \frac{w_b}{\Delta_{i+1}}} \quad (3.7.8)$$



= weighted **harmonic mean** of the slopes with weights  $w_a, w_b, (w_a + w_b = 1)$ .

Harmonic mean = “smoothed  $\min(\cdot, \cdot)$ -function”.

Contour plot of the harmonic mean of  $a$  and  $b \rightarrow (w_a = w_b = 1/2)$ .



Concrete choice of the weights:

$$w_a = \frac{2h_{i+1} + h_i}{3(h_{i+1} + h_i)}, \quad w_b = \frac{h_{i+1} + 2h_i}{3(h_{i+1} + h_i)},$$

$$\underset{\rightarrow}{\text{sgn}(\Delta_1)=\text{sgn}(\Delta_2)} \quad c_i = \begin{cases} \Delta_1 & , \text{ if } i = 0 , \\ \frac{3(h_{i+1}+h_i)}{\frac{2h_{i+1}+h_i}{\Delta_i} + \frac{2h_i+h_{i+1}}{\Delta_{i+1}}} & , \text{ for } i \in \{1, \dots, n-1\} , \\ \Delta_n & , \text{ if } i = n , \end{cases} \quad h_i := t_i - t_{i-1} .$$

(3.7.9)

**Example 3.7.10** (Monotonicity preserving piecewise cubic polynomial interpolation).

Data from ex. 3.6.5

MATLAB-function:

```
v = pchip(t,y,x);
```

t: Sampling points

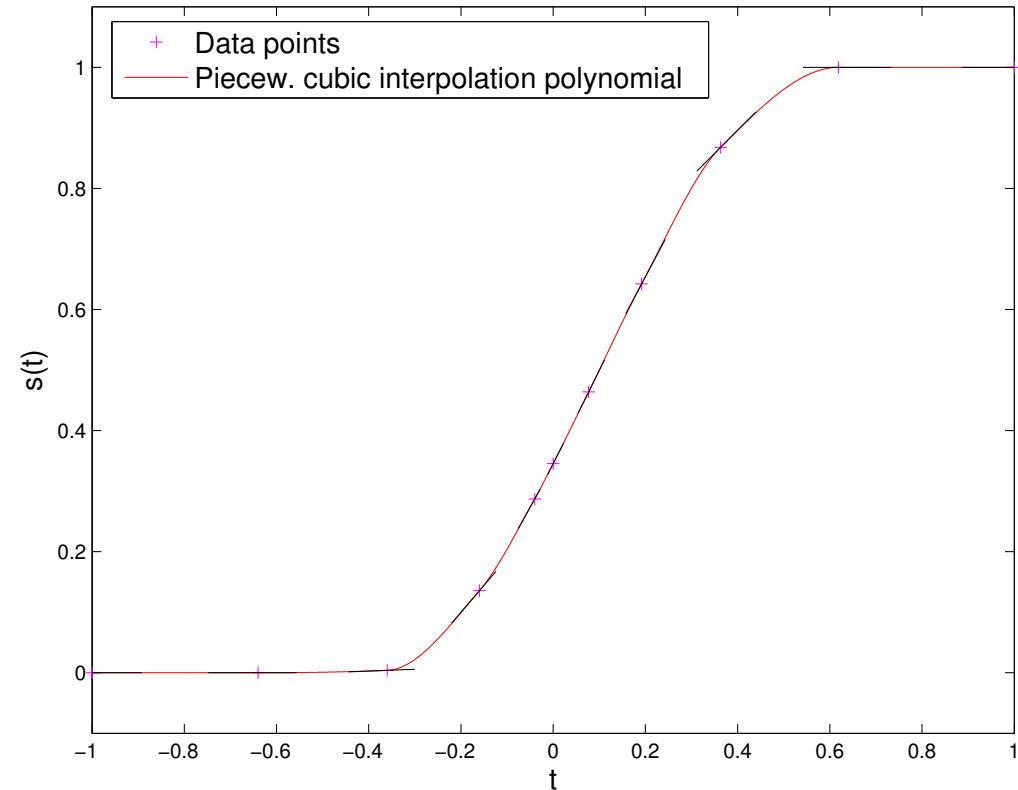
y: Sampling values

x: Evaluation points

v: Vector  $s(x_i)$ 

Local interpolation operator

! Non linear interpolation operator

**Theorem 3.7.11** (Monotonicity preservation of limited cubic Hermite interpolation).

*The cubic Hermite interpolation polynomial with slopes as in (3.7.9) provides a local monotonicity-preserving  $C^1$ -interpolant.*

*Proof.* See F. FRITSCH UND R. CARLSON, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal., 17 (1980), S. 238–246. □

*Remark 3.7.12* (Non-linear interpolation).

The monotonicity preserving cubic Hermite interpolation is **non-linear** !

Obviously, (3.1.8) is not satisfied, because linearity is clearly violated by the slope formulas (3.7.7), (3.7.9). △

Calculation of the  $c_i$  in `pchip` (details in [20]):

Code 3.7.13: Monotonicity preserving slopes in `pchip`

```

1 function c = pchipslopes(t, y)
2 % Calculation of local slopes  $c_i$  for shape preserving cubic Hermite
  interpolation, see (3.7.7), (3.7.9)
3 %  $t, y$  are row vectors passing the data points

```

```
4 n = length(t); h = diff(t); delta = diff(y)./h; % linear slopes
5 c = zeros(size(h));
6 k = find(sign(delta(1:n-2)).*sign(delta(2:n-1))>0)+1;
7 % Compute reconstruction slope according to (3.7.9)
8 w1 = 2*h(k)+h(k-1); w2 = h(k)+2*h(k-1);
9 c(k) = (w1+w2)./(w1./delta(k-1) + w2./delta(k));
0 % Special slopes at endpoints, beyond (3.7.9)
1 c(1) = pchipend(h(1),h(2),delta(1),delta(2));
2 c(n) = pchipend(h(n-1),h(n-2),delta(n-1),delta(n-2));
3
4 function d = pchipend(h1,h2,del1,del2)
5 % Noncentered, shape-preserving, three-point formula.
6 d = ((2*h1+h2)*del1 - h1*del2)/(h1+h2);
7 if sign(d) ~= sign(del1), d = 0;
8 elseif (sign(del1)~=sign(del2)) (abs(d)>abs(3*del1)), d = 3*del1;
   end
```

Cubic Hermite Interpolation of Sect. 3.7 entailed determining reconstruction slopes  $c_i$ .

Now we learn about a way how to do piecewise polynomial interpolation, which results in  $C^k$ -interpolants,  $k > 0$ , and dispenses with auxiliary slopes. The idea is to obtain the missing conditions implicitly from extra continuity conditions.

**Definition 3.8.1** (Spline space).  $\rightarrow [51, \text{Def. 8.1}]$

Given an interval  $I := [a, b] \subset \mathbb{R}$  and a **knot set/mesh**  $\mathcal{M} := \{a = t_0 < t_1 < \dots < t_{n-1} < t_n = b\}$ , the vector space  $\mathcal{S}_{d, \mathcal{M}}$  of the **spline functions** of degree  $d$  (or order  $d + 1$ ) is defined by

$$\mathcal{S}_{d, \mathcal{M}} := \{s \in C^{d-1}(I) : s_j := s|_{[t_{j-1}, t_j]} \in \mathcal{P}_d \forall j = 1, \dots, n\} .$$

$d - 1$ -times continuously differentiable locally polynomial of degree  $d$

Spline spaces mapped onto each other by differentiation & integration:

$$s \in \mathcal{S}_{d,\mathcal{M}} \Rightarrow s' \in \mathcal{S}_{d-1,\mathcal{M}} \quad \wedge \quad \int_a^t s(\tau) \, d\tau \in \mathcal{S}_{d+1,\mathcal{M}} .$$

- $d = 0$  :  $\mathcal{M}$ -piecewise constant *discontinuous* functions
- $d = 1$  :  $\mathcal{M}$ -piecewise linear *continuous* functions
- $d = 2$  : *continuously differentiable*  $\mathcal{M}$ -piecewise quadratic functions

Dimension of spline space by **counting argument** (heuristic):

$$\dim \mathcal{S}_{d,\mathcal{M}} = n \cdot \dim \mathcal{P}_d - \#\{C^{d-1} \text{ continuity constraints}\} = n \cdot (d + 1) - (n - 1) \cdot d = n + d .$$

Note special case: interpolation in  $\mathcal{S}_{1,\mathcal{M}}$  = piecewise linear interpolation.

## 3.8.1 Cubic spline interpolation [35, XIII, 46], [51, Sect. 8.6.1]

Cognitive psychology:  $C^2$ -functions are perceived as “smooth”.

→  $C^2$ -spline interpolants  $\leftrightarrow d = 3$  received special attention in CAD.

Another special case: **cubic spline interpolation**,  $d = 3$  (related to Hermite interpolation, Sect. 3.7)

Task: Given mesh  $\mathcal{M} := \{t_0 < t_1 < \dots < t_n\}$ ,  $n \in \mathbb{N}$ , “find” cubic spline  $s \in \mathcal{S}_{3,\mathcal{M}}$  such that

$$s(t_j) = y_j \quad , \quad j = 0, \dots, n . \quad (3.8.2)$$

$\hat{=}$  interpolation at knots !

From **dimensional considerations** it is clear that the interpolation conditions will fail to fix the interpolating cubic spline uniquely:

$$\dim \mathcal{S}_{3,\mathcal{M}} - \#\{\text{interpolation conditions}\} = (n + 3) - (n + 1) = 2 \text{ free d.o.f.}$$

“two conditions are missing”

Algorithmic approach to finding  $s$ :

Reuse representation through cubic Hermite basis polynomials from (3.7.2):

$$(3.7.1) \quad \blacktriangleright \quad s|_{[t_{j-1}, t_j]}(t) = \begin{aligned} & s(t_{j-1}) \cdot (1 - 3\tau^2 + 2\tau^3) + \\ & s(t_j) \cdot (3\tau^2 - 2\tau^3) + \\ & h_j s'(t_{j-1}) \cdot (\tau - 2\tau^2 + \tau^3) + \\ & h_j s'(t_j) \cdot (-\tau^2 + \tau^3), \end{aligned} \quad (3.8.3)$$

with  $h_j := t_j - t_{j-1}$ ,  $\tau := (t - t_{j-1})/h_j$ .



➤ Task of cubic spline interpolation boils down to finding slopes  $s'(t_j)$  in nodes of the mesh.

Once these slopes are known, the efficient local evaluation of a cubic spline function can be done as for a cubic Hermite interpolant, see Sect. 3.7.1, Code 3.7.2.

Note: if  $s(t_j)$ ,  $s'(t_j)$ ,  $j = 0, \dots, n$ , are fixed, then the representation (3.8.3) already guarantees  $s \in C^1([t_0, t_n])$ , cf. the discussion for cubic Hermite interpolation, Sect. 3.7.

➤ only continuity of  $s''$  

- has to be enforced by choice of  $s'(t_j)$
- ↕
- will yield extra conditions to fix the  $s'(t_j)$

However, do the

- interpolation conditions (3.8.2)  $s(t_j) = y_j$ ,  $j = 0, \dots, n$ , and the
- regularity constraint  $s \in C^2([t_0, t_n])$

uniquely determine the **unknown slopes**  $c_j := s'(t_j)$  ?

$s \in C^2([t_0, t_n]) \Rightarrow n - 1$  continuity constraints for  $s''(t)$  at the internal nodes

$$s''_{|[t_{j-1}, t_j]}(t_j) = s''_{|[t_j, t_{j+1}]}(t_j), \quad j = 1, \dots, n - 1. \quad (3.8.4)$$

Based on (3.8.3), we express (3.8.4) in concrete terms, using

$$s''_{|[t_{j-1}, t_j]}(t) = s(t_{j-1})h_j^{-2}6(-1 + 2\tau) + s(t_j)h_j^{-2}6(1 - 2\tau) \\ + h_j^{-1}s'(t_{j-1})(-4 + 6\tau) + h_j^{-1}s'(t_j)(-2 + 6\tau), \quad (3.8.5)$$

which can be obtained by the chain rule and from  $\frac{d\tau}{dt} = h_j^{-1}$ .

$$(3.8.5) \Rightarrow s''_{|[t_{j-1}, t_j]}(t_{j-1}) = -6 \cdot s(t_{j-1})h_j^{-2} + 6 \cdot s(t_j)h_j^{-2} - 4 \cdot h_j^{-1}s'(t_{j-1}) - 2 \cdot h_j^{-1}s'(t_j), \\ s''_{|[t_{j-1}, t_j]}(t_j) = 6 \cdot s(t_{j-1})h_j^{-2} - 6 \cdot s(t_j)h_j^{-2} + 2 \cdot h_j^{-1}s'(t_{j-1}) + 4 \cdot h_j^{-1}s'(t_j).$$

(3.8.4)  $\rightarrow n - 1$  linear equations for  $n$  slopes  $c_j := s'(t_j)$

$$\frac{1}{h_j}c_{j-1} + \left( \frac{2}{h_j} + \frac{2}{h_{j+1}} \right) c_j + \frac{1}{h_{j+1}}c_{j+1} = 3 \left( \frac{y_j - y_{j-1}}{h_j^2} + \frac{y_{j+1} - y_j}{h_{j+1}^2} \right), \quad (3.8.6)$$

for  $j = 1, \dots, n - 1$ .

(3.8.6)  $\Leftrightarrow$  *underdetermined*  $(n - 1) \times (n + 1)$  linear system of equations

$n - 1$ :no. of interpolation conditions

$n + 1$ :dimension of cubic spline space on knot set  $\{t_0 < t_1 < \dots < t_n\}$

$$\begin{pmatrix} b_0 & a_1 & b_1 & 0 & \cdots & \cdots & 0 \\ 0 & b_1 & a_2 & b_2 & & & \vdots \\ & 0 & \cdots & \cdots & \cdots & & \\ \vdots & & & & & & \\ & & & \cdots & a_{n-2} & b_{n-2} & 0 \\ 0 & \cdots & \cdots & 0 & b_{n-2} & a_{n-1} & b_{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} 3 \left( \frac{y_1 - y_0}{h_1^2} + \frac{y_2 - y_1}{h_2^2} \right) \\ \vdots \\ 3 \left( \frac{y_{n-1} - y_{n-2}}{h_{n-1}^2} + \frac{y_n - y_{n-1}}{h_n^2} \right) \end{pmatrix} . \quad (3.8.7)$$

with

$$b_i := \frac{1}{h_{i+1}}, \quad i = 0, 1, \dots, n - 1 ,$$

$$a_i := \frac{2}{h_i} + \frac{2}{h_{i+1}}, \quad i = 0, 1, \dots, n - 1 .$$

$$\left[ \quad b_i, a_i > 0 \quad , \quad a_i = 2(b_i + b_{i-1}) \quad \right]$$

→ two additional constraints are required, (at least) three different choices are possible:

① **Complete cubic spline interpolation:**  $s'(t_0) = c_0, s'(t_n) = c_n$  prescribed.

② **Natural cubic spline interpolation:**  $s''(t_0) = s''(t_n) = 0$

$$\frac{2}{h_1}c_0 + \frac{1}{h_1}c_1 = 3 \frac{y_1 - y_0}{h_1^2} \quad , \quad \frac{1}{h_n}c_{n-1} + \frac{2}{h_n}c_n = 3 \frac{y_n - y_{n-1}}{h_n^2} .$$

➤ Linear system of equations with tridiagonal s.p.d. (→ Def. 2.7.9, Lemma 2.7.12) coefficient matrix →  $c_0, \dots, c_n$

Thm. 2.6.37 ⇒ computational effort for the solution =  $O(n)$

③ **Periodic cubic spline interpolation:**  $s'(t_0) = s'(t_n), s''(t_0) = s''(t_n)$

$n \times n$ -linear system with s.p.d. coefficient matrix

$$\mathbf{A} := \begin{pmatrix} a_1 & b_1 & 0 & \cdots & 0 & b_0 \\ b_1 & a_2 & b_2 & & & 0 \\ 0 & \cdots & \cdots & \cdots & & \vdots \\ \vdots & & \cdots & \cdots & \cdots & 0 \\ 0 & & & \cdots & a_{n-1} & b_{n-1} \\ b_0 & 0 & \cdots & 0 & b_{n-1} & a_0 \end{pmatrix}, \quad \begin{aligned} b_i &:= \frac{1}{h_{i+1}}, \quad i = 0, 1, \dots, n-1, \\ a_i &:= \frac{2}{h_i} + \frac{2}{h_{i+1}}, \quad i = 0, 1, \dots, n-1. \end{aligned}$$

Solved with rank-1-modifications technique (see Section 2.9.0.1, Lemma 2.9.8) + tridiagonal elimination, computational effort  $O(n)$

MATLAB-function: `v = spline(t, y, x)`: natural / complete spline interpolation  
(see spline-toolbox in MATLAB)

*Remark 3.8.8* (Piecewise cubic interpolation schemes).

- Piecewise cubic local Lagrange interpolation
  - Extra degrees of freedom fixed by putting four nodes in one interval
- Cubic Hermite interpolation
  - Extra degrees of freedom fixed by reconstruction slopes
- Cubic spline interpolation
  - Extra degrees of freedom fixed by  $C^2$ -smoothness, complete/natural/periodic constraint.



## Structural properties of cubic spline interpolants

*Remark 3.8.9* (Extremal properties of natural cubic spline interpolants). → [51, Sect. 8.6.1, Property 8.2]

For  $f : [a, b] \mapsto \mathbb{R}$ ,  $f \in C^2([a, b])$ :  $\frac{1}{2} \int_a^b |f''(t)|^2 dt$  = elastic bending energy of a rod, whose shape is described by the graph of  $f$  (Soundness check: zero bending energy for straight rod).

Given: mesh  $\mathcal{M} := \{a = t_0 < t_1 < \dots < t_n = b\}$

Set  $s \in \mathcal{S}_{3, \mathcal{M}} :=$  **natural** cubic spline interpolant of  $(t_i, y_i) \in \mathbb{R}^2$ ,  $i = 0, \dots, n$ .

*Theorem 3.8.10 (Optimality of natural cubic spline interpolant).*

*The natural cubic spline interpolant minimizes the elastic curvature energy among all interpolating functions in  $C^2([a, b])$ .*

Idea of proof: **variational calculus**

We show that any small perturbation of  $s$  such that the perturbed spline still satisfies the interpolation conditions leads to an increase in elastic energy.

Pick *perturbation direction*  $k \in C^2([t_0, t_n])$  satisfying  $k(t_i) = 0, i = 0, \dots, n$ :

$$\begin{aligned}
 E_{\text{bend}}(s + k) &= \frac{1}{2} \int_a^b |s'' + \lambda k''|^2 dt & (3.8.11) \\
 &= E_{\text{bend}}(s) + \underbrace{\int_a^b s''(t)k''(t) dt}_{:=I} + \frac{1}{2} \underbrace{\int_a^b |k''|^2 dt}_{\geq 0} .
 \end{aligned}$$

Scrutiny of  $I$ : split in interval contributions, integrate by parts twice, and use  $s^{(4)} \equiv 0$ :

$$\begin{aligned}
 I &= \sum_{j=1}^n \int_{t_{j-1}}^{t_j} s''(t)k''(t) dt \\
 &= - \sum_{j=1}^n \left( \underbrace{s'''(t_j^-)}_{=0} \underbrace{k(t_j)}_{=0} - s'''(t_{j-1}^+) \underbrace{k(t_{j-1})}_{=0} \right) + \underbrace{s''(t_n)}_{=0} k'(t_n) - \underbrace{s''(t_0)}_{=0} k'(t_0) = 0 .
 \end{aligned}$$

In light of (3.8.11): non perturbation compatible with interpolation conditions can make the bending energy of  $s$  decrease!



*Remark 3.8.12* (Origin of the term “Spline”).

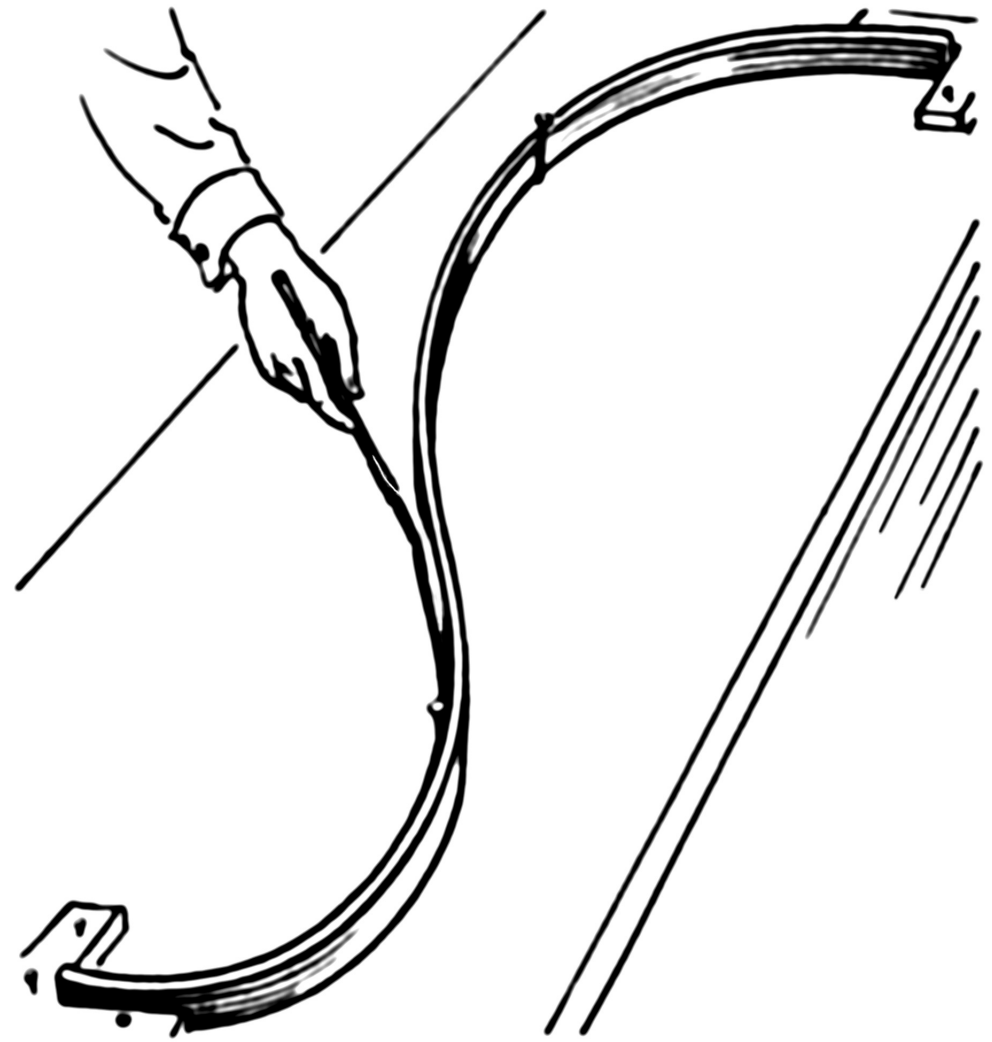
Rem. 3.8.9: (Natural) cubic spline interpolant provides  $C^2$ -curve of minimal elastic bending energy that travels through prescribed points.



Nature: A thin elastic rod fixed at certain points attains a shape that minimizes its potential bending energy (virtual work principle of statics).

► Cubic spline interpolation approximates shape of elastic rods. Such rods were in fact used in the manufacturing of ship hulls as “analog computers” for “interpolating points” that were specified by the designer of the ship.

Cubic spline interpolation  
before MATLAB



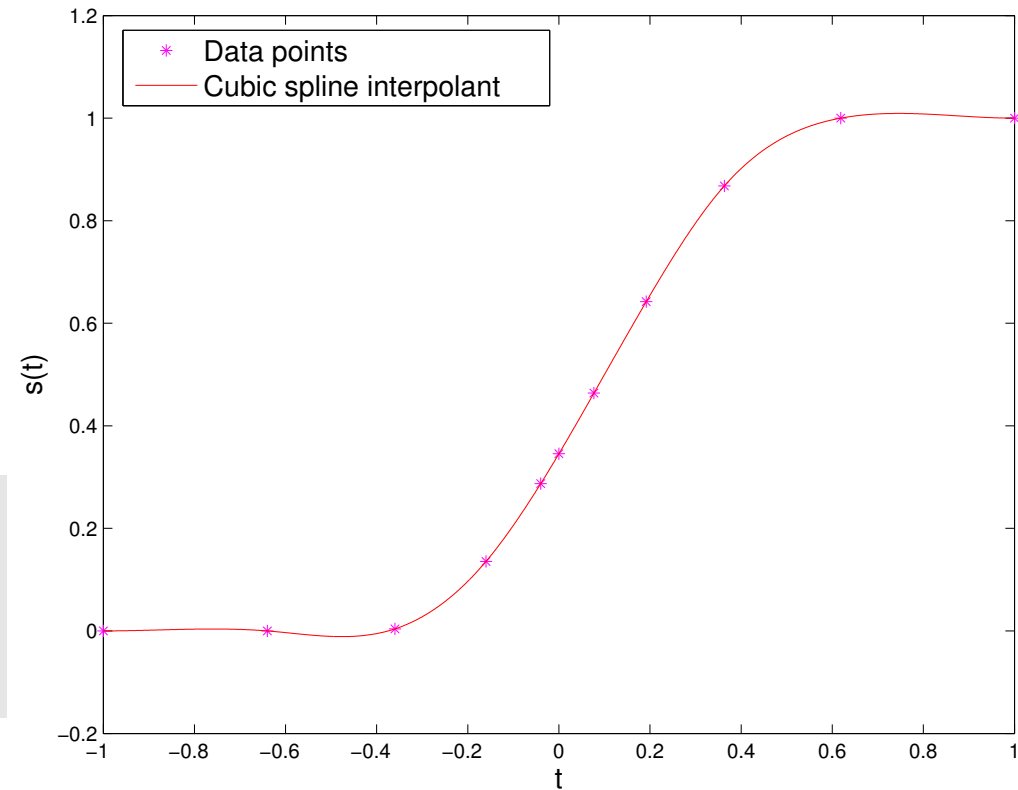
*Remark 3.8.13* (Shape preservation).

Data  $s(t_j) = y_j$  from Ex. 3.6.5 and

$$c_0 := \frac{y_1 - y_0}{t_1 - t_0},$$

$$c_n := \frac{y_n - y_{n-1}}{t_n - t_{n-1}}.$$

The cubic spline interpolant is **not** monotonicity- or curvature-preserving  
(cubic spline interpolation is linear!)



Another desirable structural property of an interpolation scheme: **locality**

- locality measures the impact of a perturbation of a data value  $y_i$  at points  $t \in [t_0, t_n]$  as a function of  $|t - t_i|$ .
- interpolation is well localized, if the impact of the perturbation of  $y_i$  displays a rapid (e.g. exponential) decay as  $|t - t_i|$  increases.

For a *linear* interpolation scheme ( $\rightarrow$  Rem. 3.1.8) locality can be deduced from the decay of the **cardinal interpolants/cardinal basis functions** ( $\rightarrow$  Ex. ??, cf. Lagrange polynomials of Sect. 3.3), that is, the functions  $l(\mathbf{e}_j)$ , where  $\mathbf{e}_j$  is the  $j$ -th unit vector, and  $l$  the interpolation operator.

Note: Lagrange polynomials from (3.3.2) provide cardinal interpolants for polynomial interpolation  $\rightarrow$  Sect. ??.

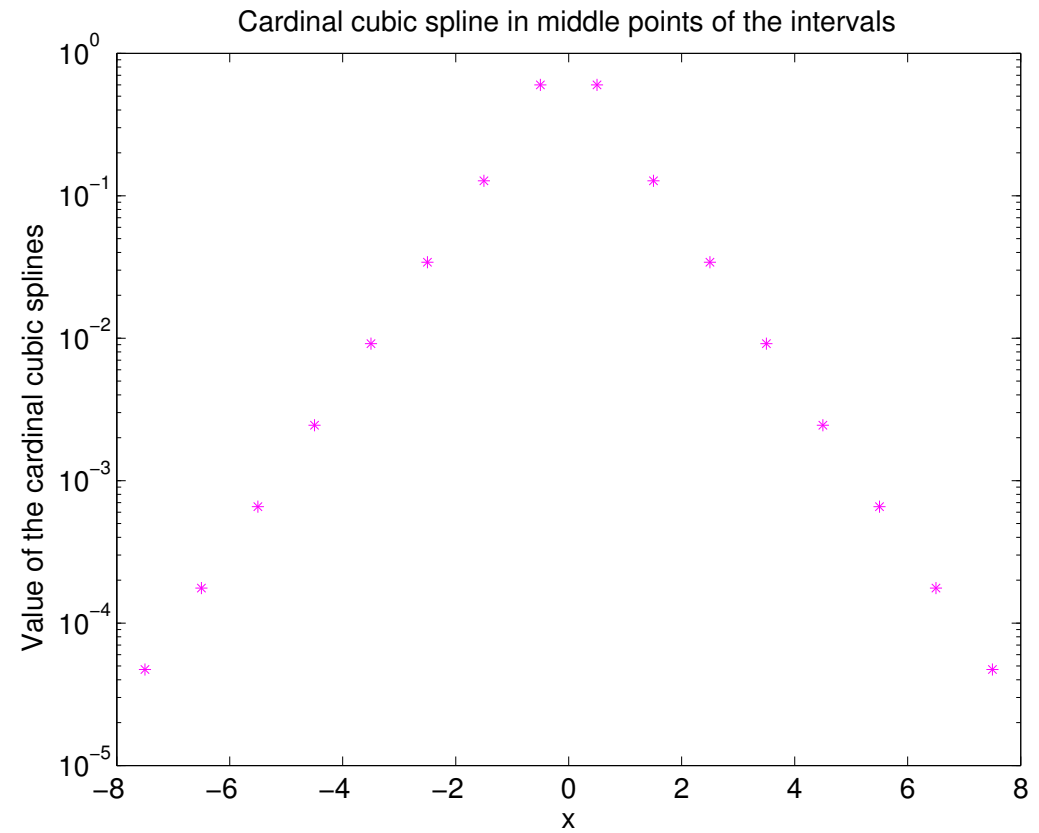
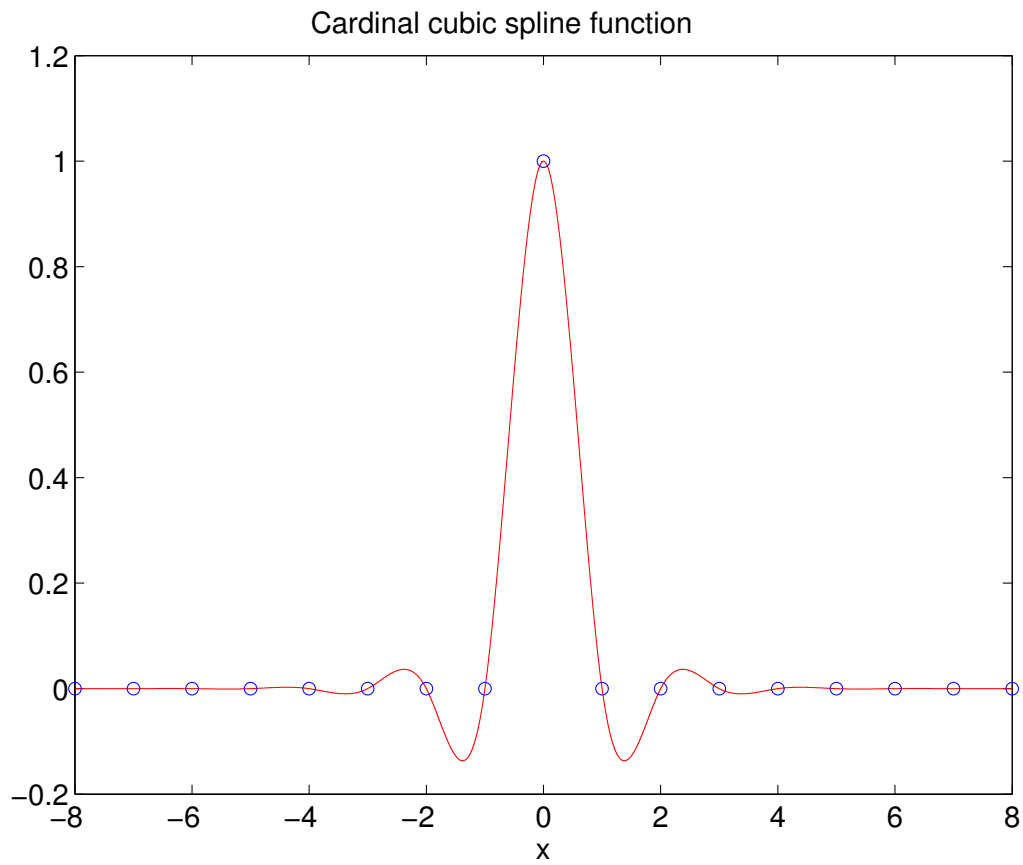
*Example 3.8.14* (Locality of the natural cubic spline interpolation).

Given a grid  $\mathcal{M} := \{t_0 < t_1 < \dots < t_n\}$  the  $i$ th natural **cardinal spline** is defined as

$$L_i \in \mathcal{S}_{3,\mathcal{M}}, \quad L_i(t_j) = \delta_{ij}, \quad L_i''(t_0) = L_i''(t_n) = 0. \quad (3.8.15)$$

Natural spline interpolant: 
$$s(t) = \sum_{j=0}^n y_j L_j(t).$$

Decay of  $L_i$   $\leftrightarrow$  **locality** of the cubic spline interpolation.



Exponential decay of the cardinal splines → cubic spline interpolation is “well localized”



Rem. 3.8.13: cubic spline interpolation is not monotonicity preserving/curvature preserving. Necessarily so, because it is a *linear* interpolation scheme, see Rem. 3.1.8; *only non-linear interpolation schemes into  $C^1$ -functions can ensure shape preservation, cf. Sect.3.7.2.*

This section presents a non-linear *quadratic spline* ( $\rightarrow$  Def. 3.8.1,  $C^1$ -functions) based interpolation scheme that manages to preserve both monotonicity and curvature of data even in a local sense, *cf. Sect. 3.6.*

R. Hiptmair

rev 38286,  
October 1,  
2011

Given: data points  $(t_i, y_i) \in \mathbb{R}^2, i = 0, \dots, n,$  assume ordering  $t_0 < t_1 < \dots < t_n.$

Sought:

- knot set  $\mathcal{M} \subset [t_0, t_n]$  ( $\rightarrow$  Def 3.8.1),
- an interpolating **quadratic spline function**  $s \in \mathcal{S}_{2, \mathcal{M}}, s(t_i) = y_i, i = 0, \dots, n$  that preserves the “shape” of the data ( $\rightarrow$  Sect. 3.6)

Notice that here  $\mathcal{M} \neq \{t_j\}_{j=0}^n$ :  $s$  interpolates the data in the points  $t_i$  but is piecewise polynomial on  $\mathcal{M}$ !

We proceed in four steps:

- ① Shape preserving choice of slopes  $c_i, i = 0, \dots, n$  [43, 50], analogous to Sect. 3.7.2

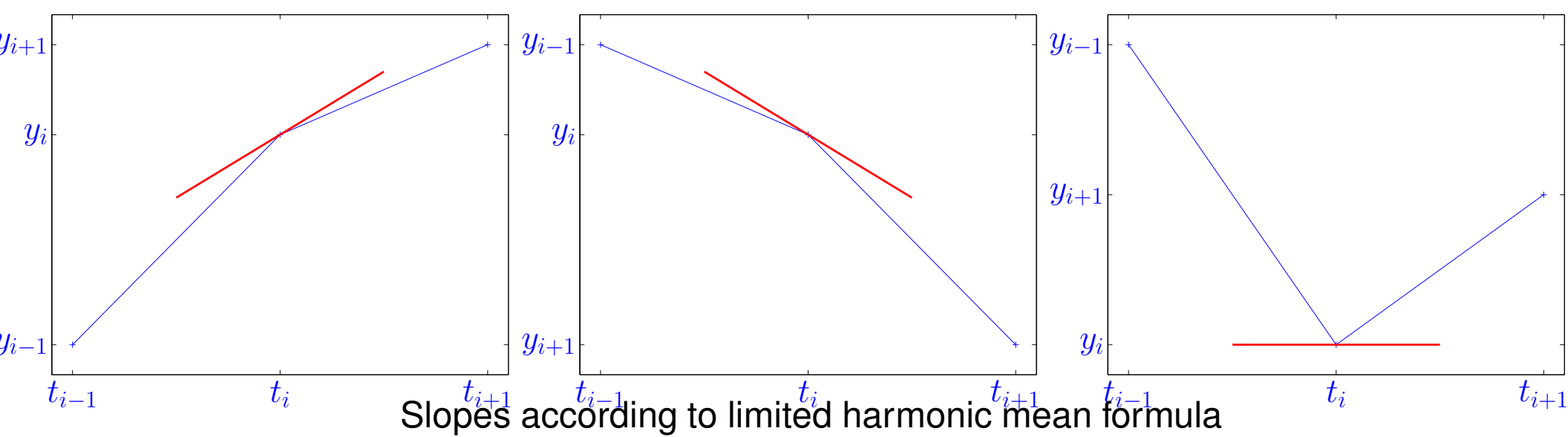
Recall (3.7.7) and (3.7.9): we fix the slopes  $c_i$  in the nodes using the harmonic mean of data slopes  $\Delta_j$ , the final interpolant will be tangent to these segments in the points  $(t_i, y_i)$ . If  $(t_i, y_i)$  is a local maximum or minimum of the data,  $c_j$  is set to zero.

$$\text{Limiter } c_i := \begin{cases} \frac{2}{\Delta_i^{-1} + \Delta_{i+1}^{-1}} & , \text{ if } \text{sign}(\Delta_i) = \text{sign}(\Delta_{i+1}) , \\ 0 & \text{otherwise,} \end{cases} \quad i = 1, \dots, n - 1 .$$

$$c_0 := 2\Delta_1 - c_1 , \quad c_n := 2\Delta_n - c_{n-1} ,$$

where  $\Delta_j = \frac{y_j - y_{j-1}}{t_j - t_{j-1}}$ .





② Choice of “extra knots”  $p_i \in ]t_{i-1}, t_i], i = 1, \dots, n$ :

$$p_i = \begin{cases} \text{intersection of the two straight lines} \\ \text{resp. through points } (t_{i-1}, y_{i-1}), (t_i, y_i) & \text{if the intersection point belongs to } (t_{i-1}, t_i] , \\ \text{with slopes } c_{i-1}, c_i, \text{ resp.} & \\ \frac{1}{2}(t_{i-1} + t_i) & \text{otherwise .} \end{cases}$$

These points will be used to build the knot set for the final **quadratic spline**:

$$\mathcal{M} = \{t_0 < p_1 \leq t_1 < p_2 \leq \dots < p_n \leq t_n\} .$$

```

p = (t(1)-1)*ones(1,length(t)-1);
for j=1:n-1
    if (c(j) ~= c(j+1))
        p(j)=(y(j+1)-y(j)+...
            t(j)*c(j)-t(j+1)*c(j+1))/...
            (c(j)-c(j+1));
    end
    if ((p(j)<t(j)) | (p(j)>t(j+1)))
        p(j) = 0.5*(t(j)+t(j+1));
    end;
end
end

```

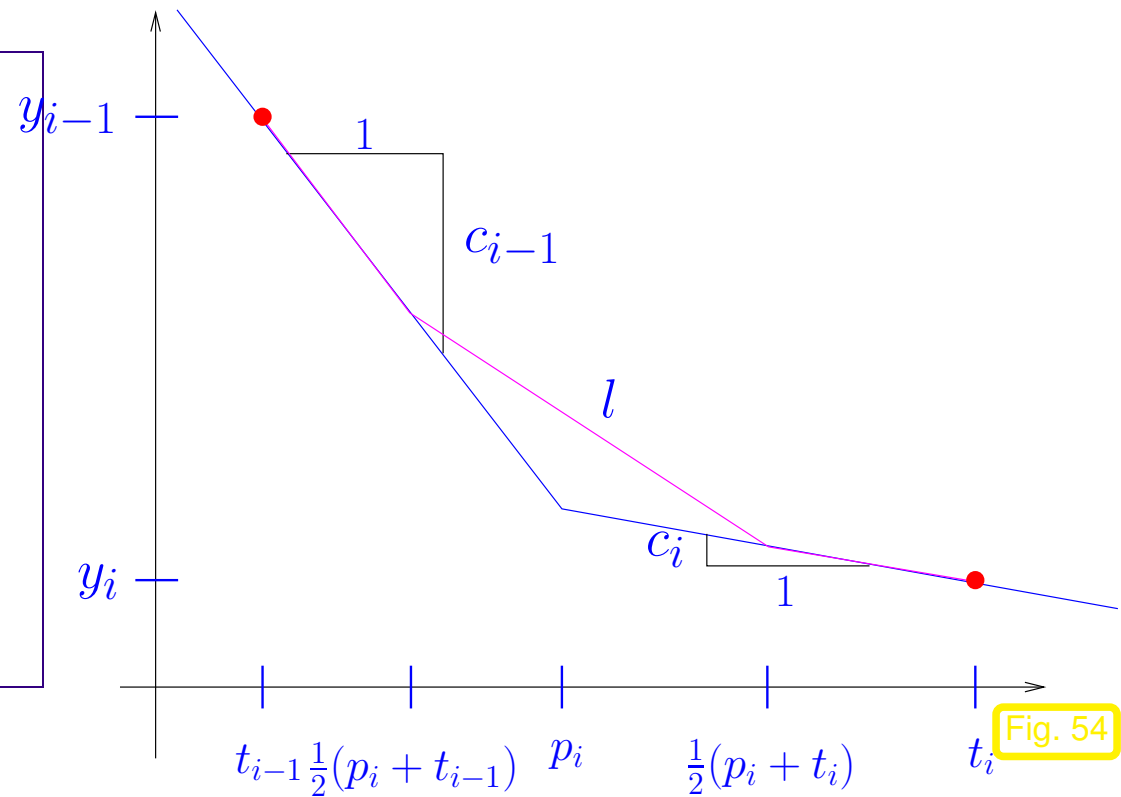


Fig. 54

③ Set  $l$  = linear spline (polygon) on the knot set  $\mathcal{M}'$  (middle points of  $\mathcal{M}$ )

$$\mathcal{M}' = \{t_0 < \frac{1}{2}(t_0 + p_1) < \frac{1}{2}(p_1 + t_1) < \frac{1}{2}(t_1 + p_2) < \dots < \frac{1}{2}(t_{n-1} + p_n) < \frac{1}{2}(p_n + t_n) < t_n\}$$

with  $l(t_i) = y_i$  ,  $l'(t_i) = c_i$ .

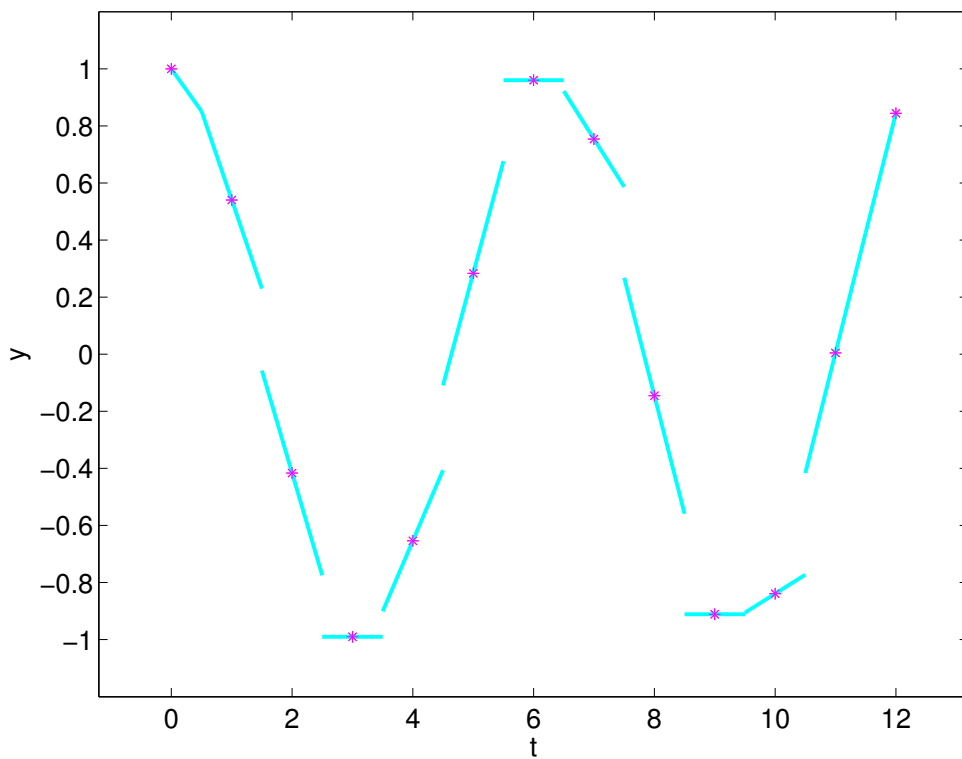
▶ In each interval  $(\frac{1}{2}(p_j + t_j), \frac{1}{2}(t_j + p_{j+1}))$  the spline corresponds to the segment of slope  $c_j$  passing through the data node  $(t_j, y_j)$ .

▶ In each interval  $(\frac{1}{2}(t_j + p_{j+1}), \frac{1}{2}(p_{j+1} + t_{j+1}))$  the spline corresponds to the segment connecting the previous ones, see Fig. 54.

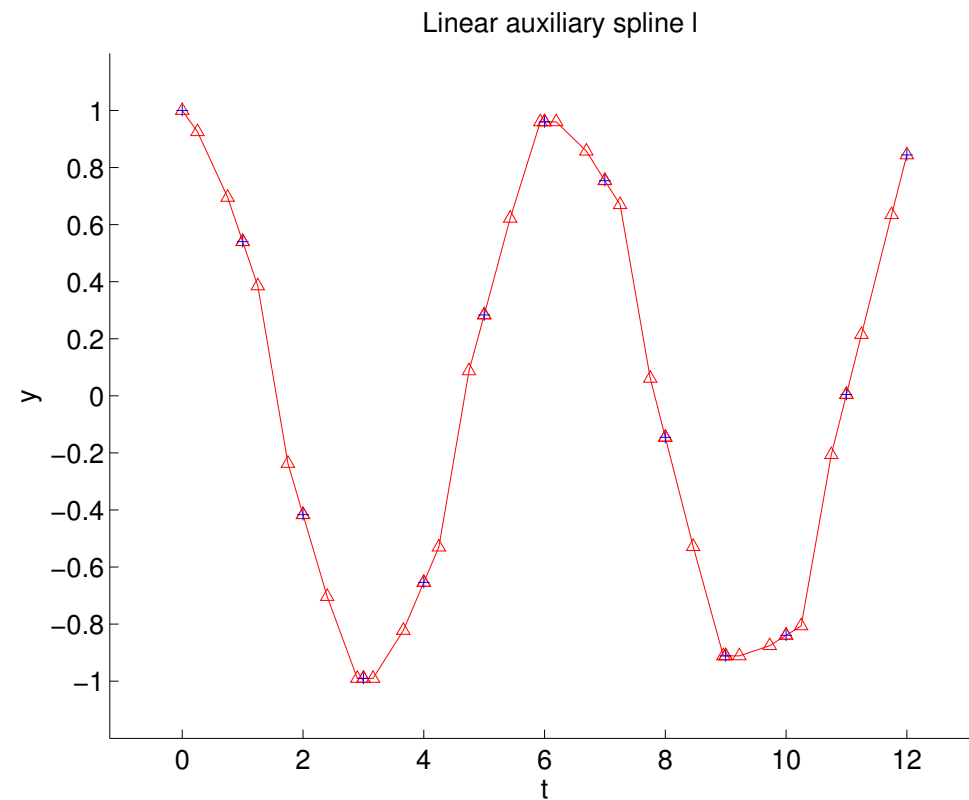
$l$  “inherits” local monotonicity and curvature from the data.

*Example 3.8.16* (Auxiliary construction for shape preserving quadratic spline interpolation).

Data points: `t = (0:12); y = cos(t);`



Local slopes  $c_i, i = 0, \dots, n$



Linear auxiliary spline  $l$



④ Local quadratic approximation / interpolation of  $l$ :

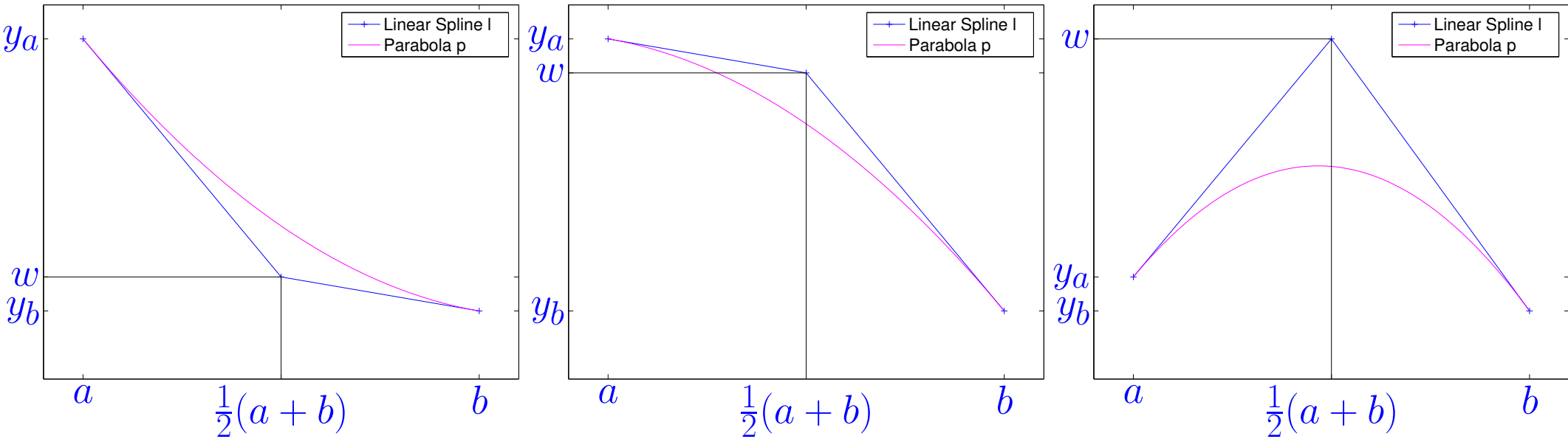
If  $g$  is a linear spline through three points  $(a, y_a), (\frac{1}{2}(a+b), w), (b, y_b), a < b, y_a, y_b, w \in \mathbb{R}$ ,

the parabola  $p(t) := (y_a(b-t)^2 + 2w(t-a)(b-t) + y_b(t-a)^2)/(b-a)^2, a \leq t \leq b,$

satisfies  $p(a) = y_a, p(b) = y_b, p'(a) = g'(a), p'(b) = g'(b).$

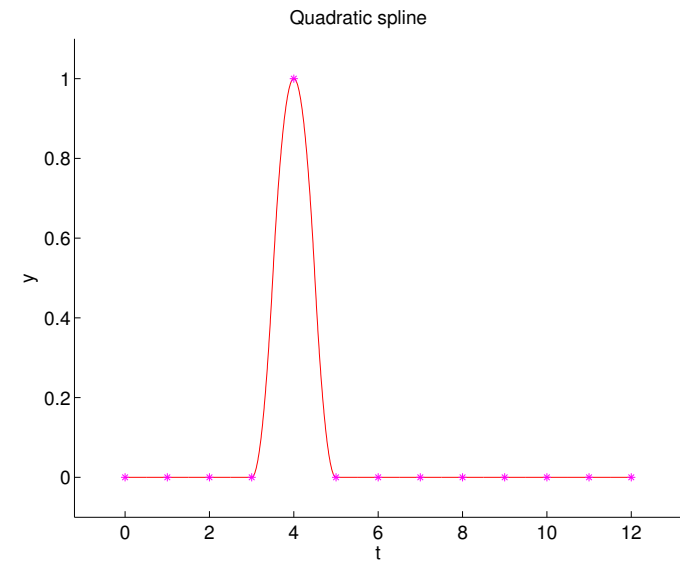
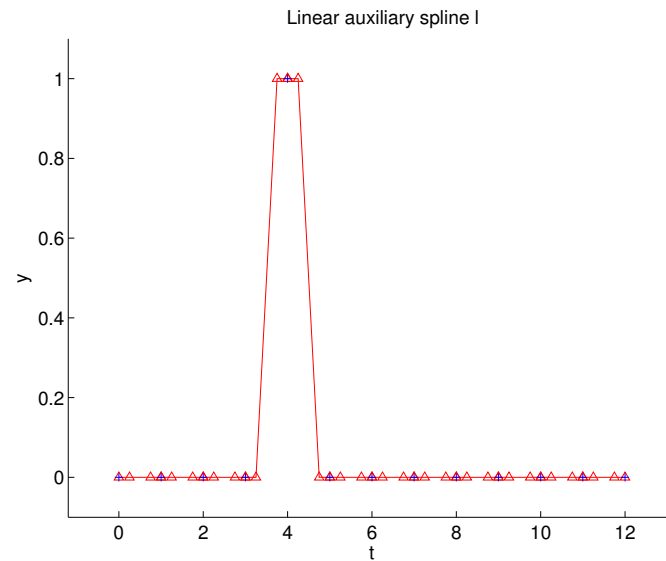
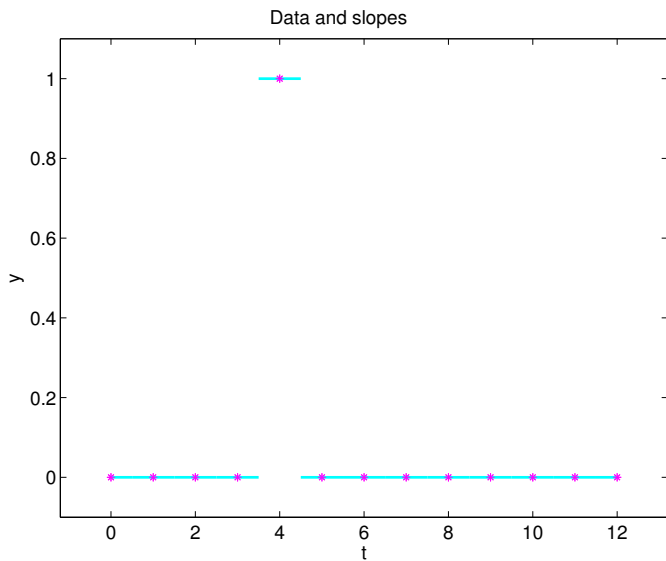
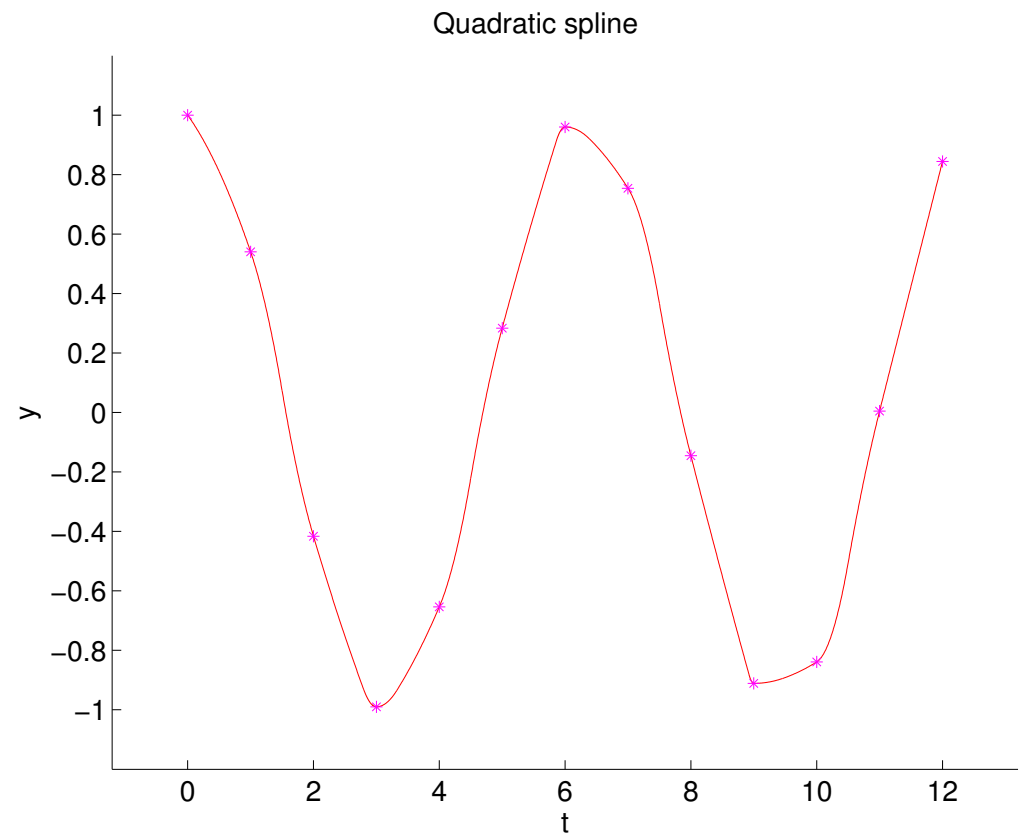
$g$  monotonic increasing / decreasing  $\Rightarrow p$  monotonic increasing / decreasing

$g$  convex / concave  $\Rightarrow p$  convex / concave



This implies that the final quadratic spline that passes through the points  $(t_j, y_j)$  with slopes  $c_j$  can be built locally as  $p$  using the linear spline  $l$ , in place of  $g$ .

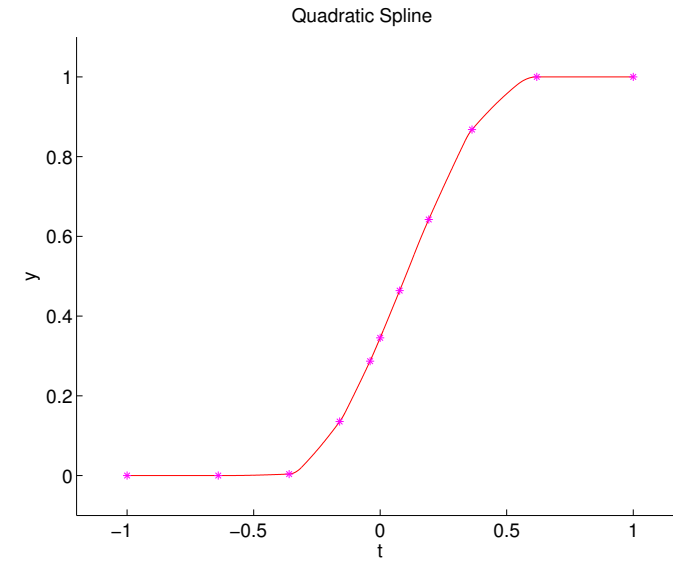
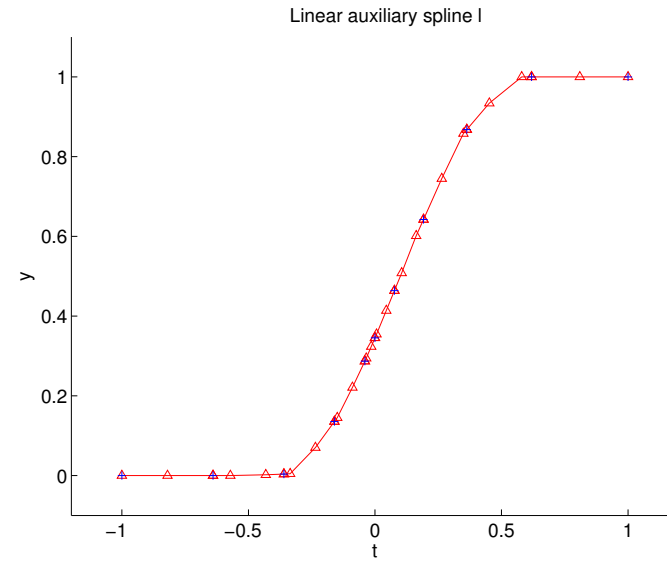
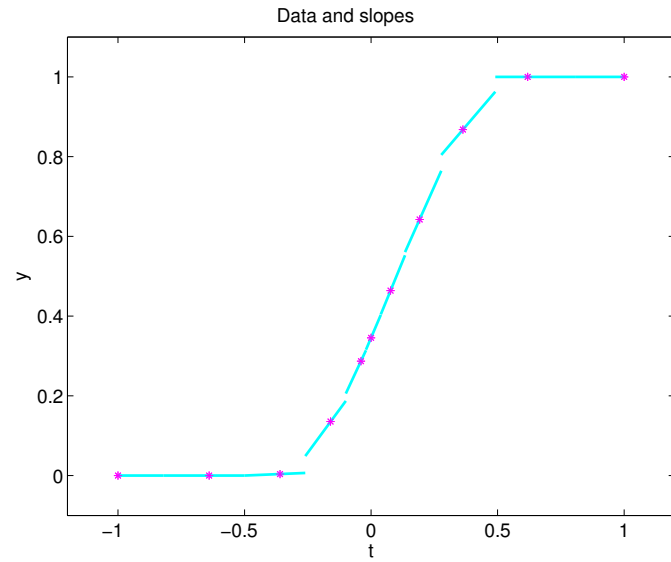
Continuation of Ex. 3.8.16:  
Interpolating quadratic spline



Shape preserving quadratic spline interpolation = **local but not linear**

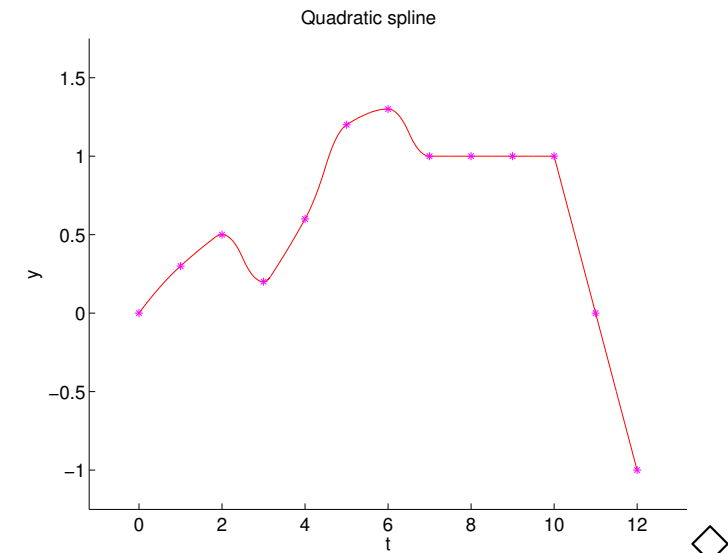
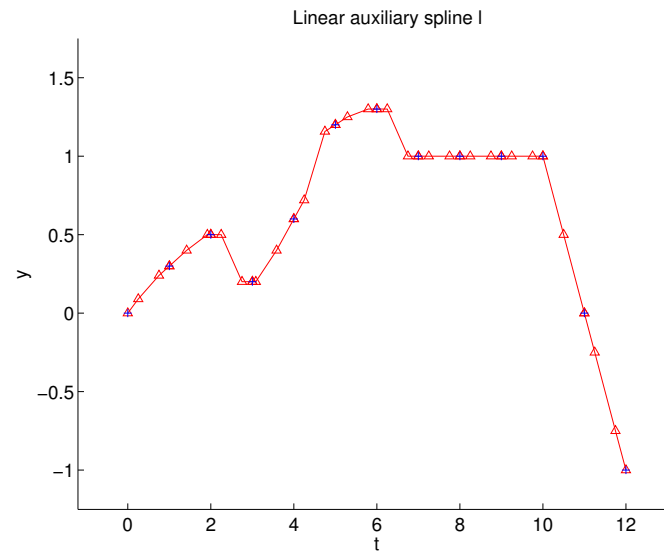
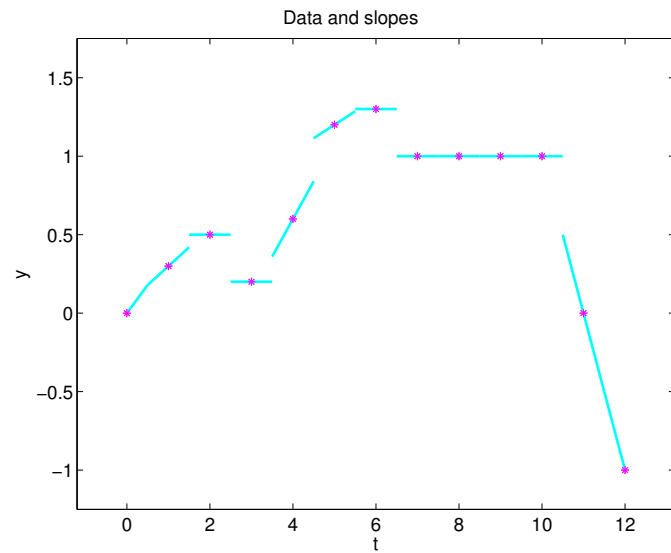
# Example 3.8.17 (Shape preserving quadratic spline interpolation).

Data from Ex. 3.6.5:



Data from [43]:

$t_i$	0	1	2	3	4	5	6	7	8	9	10	11	12
$y_i$	0	0.3	0.5	0.2	0.6	1.2	1.3	1	1	1	1	0	-1



### Code 3.8.18: Step by step shape preserving spline interpolation

```

1 % 22.06.2009 shapepresintp.m
2 % Shape preserving interpolation through nodes (t,y)
3 % Build a quadratic spline interpolation without overshooting
4 % In 4 steps, see the comments in each step
5 %
6 % example:
7 % shapepresintp([1, 1.5,3:10], [ 1 2.5 4 3.8 3 2 1 4 2 1])
8 % shapepresintp([0:0.05:1], cosf([0:0.05:1]))
9 % shapepresintp([0:0.05:1], [zeros(1,5),1,zeros(1,15) ] )
0 % shapepresintp([0:12], [0 0.3 0.5 0.2 0.6 1.2 1.3 1 1 1 1 0 -1])

```



```
1
2 function shapepresintp(t,y)
3
4 n=length (t) -1;
5 % plot the data and prepare the figure
6 figure;
7 hplot(1)=plot (t,y,'k*-');
8 hold on;
9 newaxis=axis;
10 newaxis([3,4])=[newaxis(3)-0.5, newaxis(4)+0.5];
11 axis (newaxis);           % enlarge the vertical size of the plot
12 title ('Data points - Press enter to continue')
13 plot (t,ones(1,n+1)* (newaxis(3)+0.25),'k. ');
14 set (gca, 'XTick', t)
15 leg={'Data','Slopes','Middle points','Linear spline' ,'Sh. pres.
    spline'};
16 legend (hplot(1),leg{1});
17 pause;
18
19 % ===== Step 1: choice of slopes =====
20 % shape-faithful slopes (c) in the nodes using harmonic mean of data slopes
21 % the final interpolant will be tangents to these segments
```

```
3 disp('STEP 1')
4 title('Shape-faithful slopes - Press enter to continue')
5 h=diff(t);
6 delta = diff(y) ./ h;           % slopes of data
7 c=zeros(size(t));
8 for j=1:n-1
9     if (delta(j)*delta(j+1) >0)
10        c(j+1) = 2/(1/delta(j) + 1/delta(j+1));
11    end
12 end
13 c(1)=2*delta(1)-c(2);   c(n+1)=2*delta(n)-c(n);
14
15 % plot segments indicating the slopes c(i):
16 % use (vector) plot handle 'hplot' to reduce the linewidth in step 2
17 hplots=zeros(1,n+1);
18 for j=2:n
19     hplots1(j)=plot([t(j)-0.3*h(j-1),t(j)+0.3*h(j)],
20                    [y(j)-0.3*h(j-1)*c(j),y(j)+0.3*h(j)*c(j)],'-','linewidth',2);
21 end
22 hplots1(1)=plot([t(1),t(1)+0.3*h(1)], [y(1),y(1)+0.3*h(1)*c(1)],
23                '-','linewidth',2);
24 hplots1(n+1)= plot([t(end)-0.3*h(end),t(end)],
25                  [y(end)-0.3*h(end)*c(end),y(end)], '-','linewidth',2);
```

```
3 legend ([hplot(1), hplots1(1)], leg{1:2});
4 pause;
5
6 % ===== Step 2: choice of middle points =====
7 % fix points p(j) in [t(j), t(j+1)], depending on the slopes c(j), c(j+1)
8
9 disp('STEP 2')
10 title('Middle points - Press enter to continue')
11 set(hplots1,'linewidth',1)
12
13 p = (t(1)-1)*ones(1,length(t)-1);
14 for j=1:n
15     if (c(j) ~= c(j+1))
16         p(j) = (y(j+1)-y(j) + t(j)*c(j) - t(j+1)*c(j+1)) / (c(j) - c(j+1));
17     end
18     % check and repair if p(j) is outside its interval:
19     if ((p(j) < t(j)) || (p(j) > t(j+1))); p(j) = 0.5*(t(j)+t(j+1));
20     end;
21 end
22 hplot(2) = plot(p, ones(1,n)*(newaxis(3)+0.25), 'go');
23 legend([hplot(1), hplots1(1), hplot(2)], leg{1:3});
24 pause;
```

```
5
6 % ===== Step 3: auxiliary linear spline =====
7 % build the linear spline with nodes in:
8 % -t(j)
9 % -the middle points between t(j) and p(j)
0 % -the middle points between p(j) and t(j+1)
1 % -t(j+1)
2 % and with slopes c(j) in t(j), for every j
3
4 disp ('STEP 3')
5 title ('Auxiliary linear spline - Press enter to continue')
6
7 for j=1:n
8     hplot(3)=plot ([t(j) 0.5*(p(j)+t(j)) 0.5*(p(j)+t(j+1)) t(j+1)],
9                 [y(j) y(j)+0.5*(p(j)-t(j))*c(j)
10                y(j+1)+0.5*(p(j)-t(j+1))*c(j+1) y(j+1)], 'm-^');
11     plot ([t(j) 0.5*(p(j)+t(j)) 0.5*(p(j)+t(j+1)) t(j+1)],
12          ones(1,4)*(newaxis(3)+0.25) , 'm^');
13 end
14 legend ([hplot(1), hplotsl(1), hplot(2), hplot(3)], leg{1:4});
15 pause;
16
17 % ===== Step 4: quadratic spline =====
```

```
5 % final quadratic shape preserving spline
6 % quadratic polynomial in the intervals [t(j), p(j)] and [p(j), t(j)]
7 % tangent in t(j) and p(j) to the linear spline of step 3
8
9 disp('STEP 4')
0 title('Quadratic spline')
1
2 % for every interval 2 quadratic interpolations
3 % a, b, ya, yb = extremes and values in the subinterval
4 % w = value in middle point that gives the right slope
5 for j=1:n
6     a=t(j);
7     b=p(j);
8     ya = y(j);
9     w = y(j)+0.5*(p(j)-t(j))*c(j);
0     yb = ((t(j+1)-p(j))*(y(j)+0.5*(p(j)-t(j))*c(j))+...
1         (p(j)-t(j))*(y(j+1)+0.5*(p(j)-t(j+1))*c(j+1)))/(t(j+1)-t(j));
2     x=linspace(a,b,100);
3     pb = (ya*(b-x).^2 + 2*w*(x-a).*(b-x)+yb*(x-a).^2)/((b-a)^2);
4     hplot(4)=plot(x,pb,'r-', 'linewidth',2);
5
6     a = b;
7     b = t(j+1);
```

```
8 ya = yb;  
9 yb = y(j+1);  
0 w = y(j+1)+0.5*(p(j)-t(j+1))*c(j+1);  
1 x = (a:(b-a)/100:b);  
2 pb = (ya*(b-x).^2 + 2*w*(x-a).*(b-x)+yb*(x-a).^2) / ((b-a)^2);  
3 plot(x,pb,'r-', 'linewidth',2);  
4  
5 plot(p(j),ya,'go');  
6 end  
7  
8 % replot initial nodes over the other plots:  
9 plot(t,y,'k*');  
0 % plot(p,yb,'go')  
1 legend([hplot(1), hplotsl(1), hplot(2:4)], leg);  
2 title('Shape preserving interpolation')
```

# 4 Iterative Methods for Non-Linear Systems of Equations

# 4

Example 4.0.1 (Non-linear electric circuit).

Schmitt trigger circuit

NPN bipolar junction transistor:

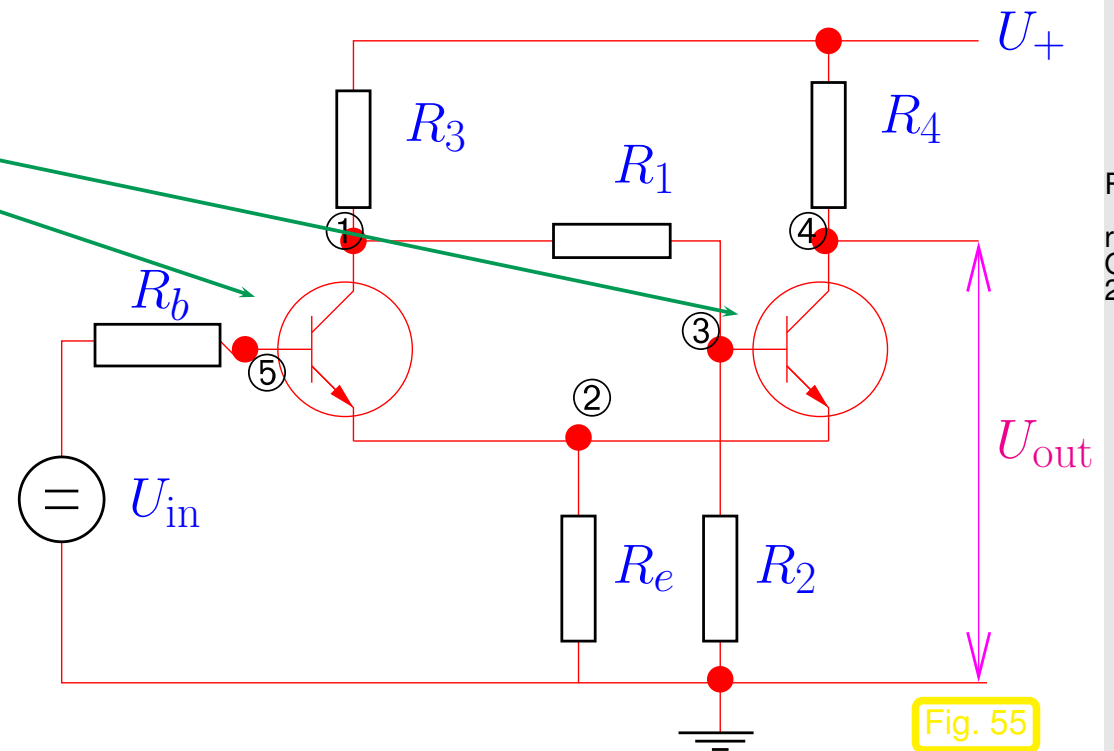
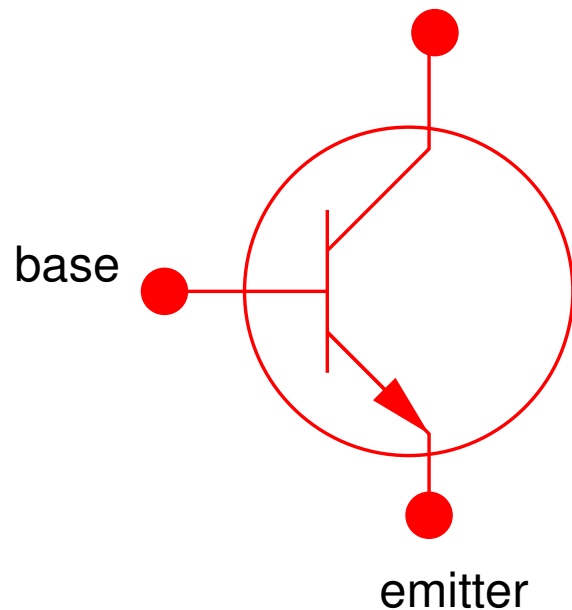


Fig. 55

Ebers-Moll model (large signal approximation):

$$\begin{aligned}
 I_C &= I_S \left( e^{\frac{U_{BE}}{U_T}} - e^{\frac{U_{BC}}{U_T}} \right) - \frac{I_S}{\beta_R} \left( e^{\frac{U_{BC}}{U_T}} - 1 \right) = I_C(U_{BE}, U_{BC}) , \\
 I_B &= \frac{I_S}{\beta_F} \left( e^{\frac{U_{BE}}{U_T}} - 1 \right) + \frac{I_S}{\beta_R} \left( e^{\frac{U_{BC}}{U_T}} - 1 \right) = I_B(U_{BE}, U_{BC}) , \\
 I_E &= I_S \left( e^{\frac{U_{BE}}{U_T}} - e^{\frac{U_{BC}}{U_T}} \right) + \frac{I_S}{\beta_F} \left( e^{\frac{U_{BE}}{U_T}} - 1 \right) = I_E(U_{BE}, U_{BC}) .
 \end{aligned}
 \tag{4.0.2}$$

$I_C, I_B, I_E$ : current in collector/base/emitter,

$U_{BE}, U_{BC}$ : potential drop between base-emitter, base-collector.

( $\beta_F$  is the forward common emitter current gain (20 to 500),  $\beta_R$  is the reverse common emitter current gain (0 to 20),  $I_S$  is the reverse saturation current (on the order of  $10^{-15}$  to  $10^{-12}$  amperes),  $U_T$  is the thermal voltage (approximately 26 mV at 300 K).)



**Non-linear system of equations** from nodal analysis ( $\rightarrow$  Ex. 2.6.3):

$$\begin{aligned}
 \textcircled{1} : & R_3(U_1 - U_+) + R_1(U_1 - U_3) + I_B(U_5 - U_1, U_5 - U_2) = 0, \\
 \textcircled{2} : & R_e U_2 + I_E(U_5 - U_1, U_5 - U_2) + I_E(U_3 - U_4, U_3 - U_2) = 0, \\
 \textcircled{3} : & R_1(U_3 - U_1) + I_B(U_3 - U_4, U_3 - U_2) = 0, \\
 \textcircled{4} : & R_4(U_4 - U_+) + I_C(U_3 - U_4, U_3 - U_2) = 0, \\
 \textcircled{5} : & R_b(U_5 - U_{in}) + I_B(U_5 - U_1, U_5 - U_2) = 0.
 \end{aligned}
 \tag{4.0.3}$$

5 equations  $\leftrightarrow$  5 unknowns  $U_1, U_2, U_3, U_4, U_5$

Formally:

$$(4.0.3) \iff F(\mathbf{u}) = 0$$



A **non-linear system of equations** is a concept almost *too abstract to be useful*, because it covers an extremely wide variety of problems. Nevertheless in this chapter we will mainly look at “generic” methods for such systems. This means that every method discussed may take a good deal of fine-tuning before it will really perform satisfactorily for a given non-linear system of equations.

Given: function  $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ ,  $n \in \mathbb{N}$



Possible meaning: Availability of a **procedure** function  $y=F(x)$  evaluating  $F$

Sought: solution of **non-linear equation**

$$F(\mathbf{x}) = 0$$

Note:  $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n \iff$  “same number of equations and unknowns”

In general no existence & uniqueness of solutions

## 4.1 Iterative methods

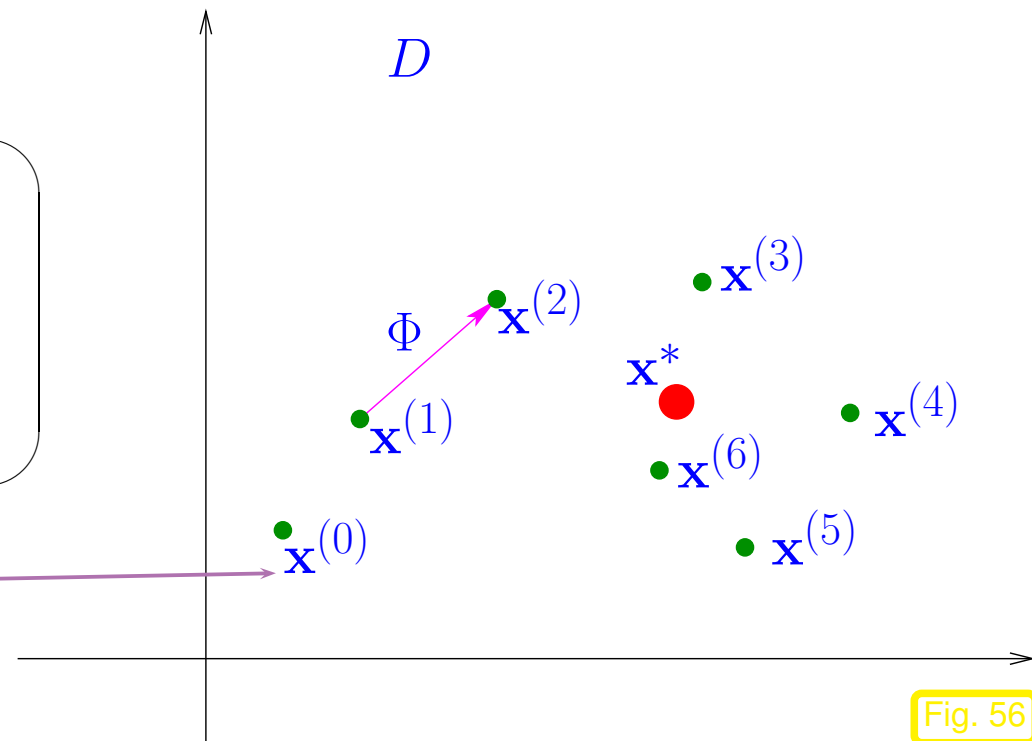
**Remark 4.1.1** (Necessity of iterative approximation).

Gaussian elimination ( $\rightarrow$  Sect. 2.1) provides an algorithm that, if carried out in exact arithmetic, computes the solution of a linear system of equations with a *finite* number of elementary operations. However, linear systems of equations represent an exceptional case, because it is hardly ever possible to solve general systems of non-linear equations using only finitely many elementary operations. Certainly this is the case whenever irrational numbers are involved.



An **iterative method** for (approximately) solving the non-linear equation  $F(\mathbf{x}) = 0$  is an algorithm generating a sequence  $(\mathbf{x}^{(k)})_{k \in \mathbb{N}_0}$  of **approximate solutions**.

Initial guess  $\rightarrow$



Fundamental concepts: **convergence**  $\rightarrow$  **speed of convergence**  
**consistency**

- iterate  $\mathbf{x}^{(k)}$  depends on  $F$  and (one or several)  $\mathbf{x}^{(n)}$ ,  $n < k$ , e.g.,

$$\mathbf{x}^{(k)} = \underbrace{\Phi_F(\mathbf{x}^{(k-1)}, \dots, \mathbf{x}^{(k-m)})}_{\text{iteration function for } m\text{-point method}} \quad (4.1.2)$$

- $\mathbf{x}^{(0)}, \dots, \mathbf{x}^{(m-1)}$  = **initial guess(es)** (*ger.:* Anfangsnäherung)

**Definition 4.1.3** (Convergence of iterative methods).

An iterative method **converges** (for fixed initial guess(es))  $:\Leftrightarrow \mathbf{x}^{(k)} \rightarrow \mathbf{x}^*$  and  $F(\mathbf{x}^*) = 0$ .

**Definition 4.1.4** (Consistency of iterative methods).

An iterative method is **consistent** with  $F(\mathbf{x}) = 0$

$$:\Leftrightarrow \Phi_F(\mathbf{x}^*, \dots, \mathbf{x}^*) = \mathbf{x}^* \Leftrightarrow F(\mathbf{x}^*) = 0$$

Terminology: **error** of iterates  $\mathbf{x}^{(k)}$  is defined as:  $\mathbf{e}^{(k)} := \mathbf{x}^{(k)} - \mathbf{x}^*$

**Definition 4.1.5** (Local and global convergence).  $\rightarrow$  [35, Def. 17.1]

An iterative method **converges locally** to  $\mathbf{x}^* \in \mathbb{R}^n$ , if there is a neighborhood  $U \subset D$  of  $\mathbf{x}^*$ , such that

$$\mathbf{x}^{(0)}, \dots, \mathbf{x}^{(m-1)} \in U \Rightarrow \mathbf{x}^{(k)} \text{ well defined} \wedge \lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x}^*$$

for the sequences  $(\mathbf{x}^{(k)})_{k \in \mathbb{N}_0}$  of iterates.

If  $U = D$ , the iterative method is **globally convergent**.

local convergence



(Only initial guesses “sufficiently close” to  $\mathbf{x}^*$  guarantee convergence.)

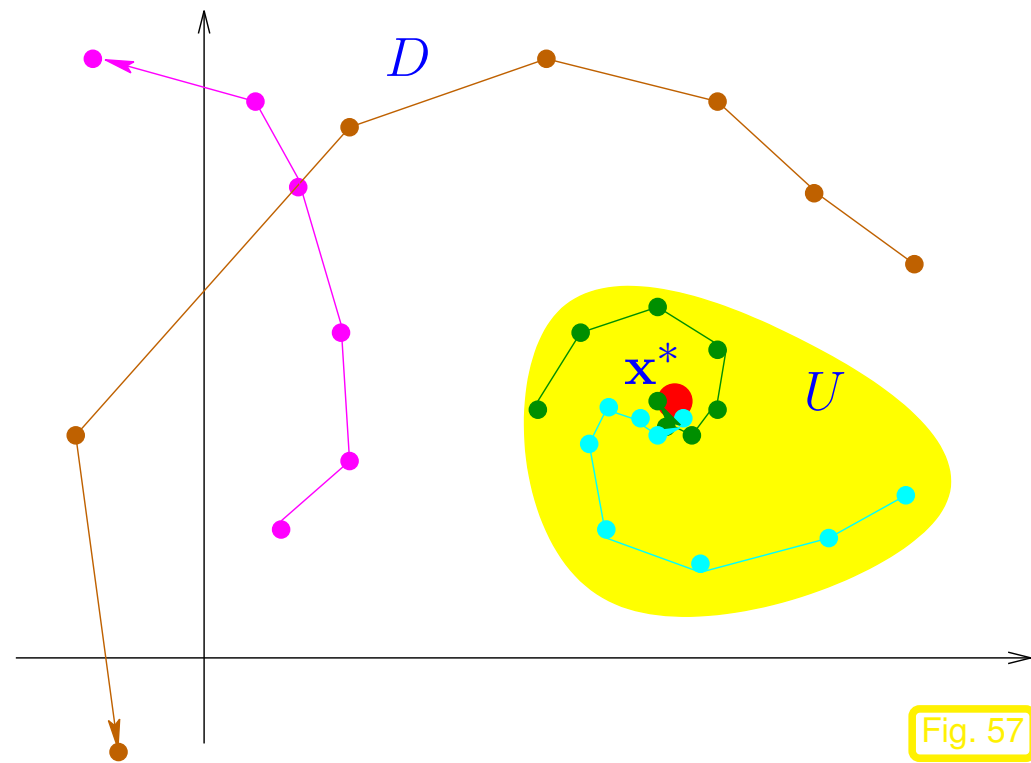


Fig. 57

Goal: Find iterative methods that converge (locally) to a solution of  $F(\mathbf{x}) = 0$ .

Two general questions: How to measure the speed of convergence?  
When to terminate the iteration?

## 4.1.1 Speed of convergence

Here and in the sequel,  $\|\cdot\|$  designates a generic vector norm, see Def. 2.5.1. Any occurring matrix norm is induced by this vector norm, see Def. 2.5.5.

It is important to be aware which statements depend on the choice of norm and which do not!

“*Speed of convergence*”  $\leftrightarrow$  decrease of norm (see Def. 2.5.1) of iteration error

**Definition 4.1.6** (Linear convergence).

A sequence  $\mathbf{x}^{(k)}$ ,  $k = 0, 1, 2, \dots$ , in  $\mathbb{R}^n$  *converges linearly* to  $\mathbf{x}^* \in \mathbb{R}^n$ , if

$$\exists L < 1: \quad \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\| \leq L \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| \quad \forall k \in \mathbb{N}_0 .$$

Terminology: least upper bound for  $L$  gives the *rate of convergence*

*Remark 4.1.7* (Impact of choice of norm).

<i>Fact of convergence</i> of iteration is	independent	of choice of norm
<i>Fact of linear convergence</i>	depends	on choice of norm
<i>Rate of linear convergence</i>	depends	on choice of norm

Recall: equivalence of all norms on finite dimensional vector space  $\mathbb{K}^n$ :

*Definition 4.1.8* (Equivalence of norms).

Two norms  $\|\cdot\|_1$  and  $\|\cdot\|_2$  on a vector space  $V$  are equivalent if

$$\exists \underline{C}, \overline{C} > 0: \quad \underline{C} \|v\|_1 \leq \|v\|_2 \leq \overline{C} \|v\|_1 \quad \forall v \in V .$$

*Theorem 4.1.9* (Equivalence of all norms on finite dimensional vector spaces).

If  $\dim V < \infty$  all norms ( $\rightarrow$  Def. 2.5.1) on  $V$  are equivalent ( $\rightarrow$  Def. 4.1.8).



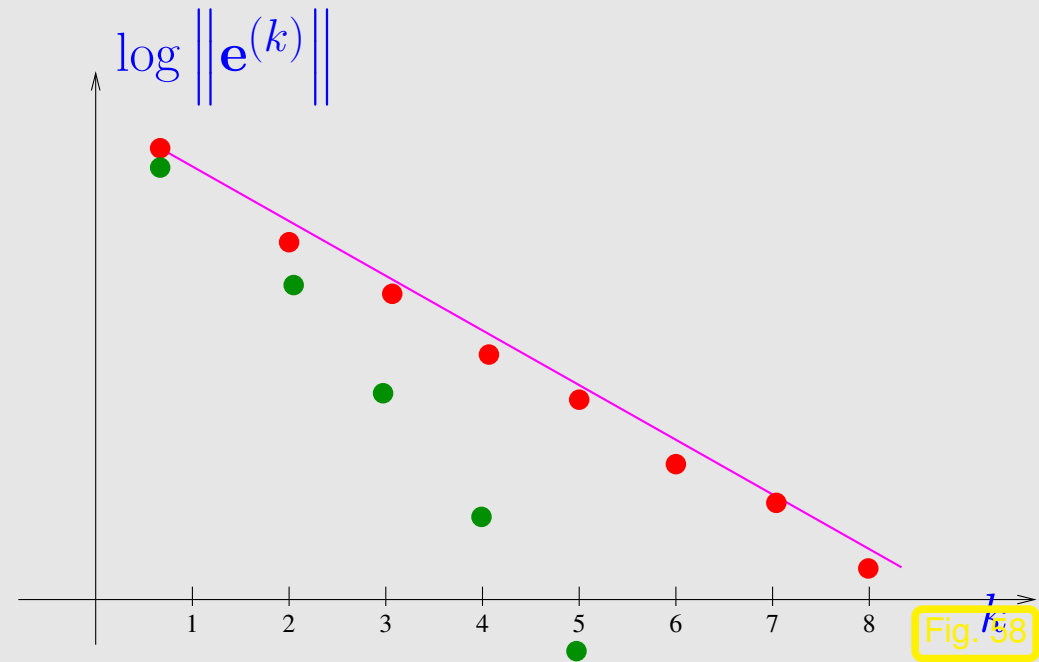
Remark 4.1.10 (Detecting linear convergence).

norms of iteration errors  
 $\updownarrow$   
 $\sim$  straight line in **lin-log** plot

$$\|e^{(k)}\| \leq L^k \|e^{(0)}\| ,$$

$$\log(\|e^{(k)}\|) \leq k \log(L) + \log(\|e^{(0)}\|) .$$

(●: Any “faster” convergence also qualifies as linear !)



Let us abbreviate the error norm in step  $k$  by  $\epsilon_k := \|\mathbf{x}^{(k)} - \mathbf{x}^*\|$ . In the case of linear convergence (see Def. 4.1.6) assume (with  $0 < L < 1$ )

$$\epsilon_{k+1} \approx L\epsilon_k \Rightarrow \log \epsilon_{k+1} \approx \log L + \log \epsilon_k \Rightarrow \log \epsilon_k \approx k \log L + \log \epsilon_0 . \quad (4.1.11)$$

We conclude that  $\log L < 0$  describes slope of graph in lin-log error chart.

Also recall: guessing time complexity  $O(n^\alpha)$  of an algorithm from measurements, see Rem. 1.3.7.  $\triangle$

Example 4.1.12 (Linearly convergent iteration).

Iteration ( $n = 1$ ):

$$x^{(k+1)} = x^{(k)} + \frac{\cos x^{(k)} + 1}{\sin x^{(k)}}.$$

$x$  has to be initialized with the different values for  $x_0$ .

Code 4.1.13: simple fixed point iteration

```

1 y = [ ];
2 for i = 1:15
3     x = x + (cos(x)+1)/sin(x);
4     y = [y, x];
5 end
6 err = y - x;
7 rate = err(2:15) ./ err(1:14);

```

Note:  $x^{(15)}$  replaces the exact solution  $x^*$  in the computation of the rate of convergence.

$k$	$x^{(0)} = 0.4$		$x^{(0)} = 0.6$		$x^{(0)} = 1$	
	$x^{(k)}$	$\frac{ x^{(k)} - x^{(15)} }{ x^{(k-1)} - x^{(15)} }$	$x^{(k)}$	$\frac{ x^{(k)} - x^{(15)} }{ x^{(k-1)} - x^{(15)} }$	$x^{(k)}$	$\frac{ x^{(k)} - x^{(15)} }{ x^{(k-1)} - x^{(15)} }$
2	3.3887	0.1128	3.4727	0.4791	2.9873	0.4959
3	3.2645	0.4974	3.3056	0.4953	3.0646	0.4989
4	3.2030	0.4992	3.2234	0.4988	3.1031	0.4996
5	3.1723	0.4996	3.1825	0.4995	3.1224	0.4997
6	3.1569	0.4995	3.1620	0.4994	3.1320	0.4995
7	3.1493	0.4990	3.1518	0.4990	3.1368	0.4990
8	3.1454	0.4980	3.1467	0.4980	3.1392	0.4980

Linear convergence as in Def. 4.1.6



error graphs = straight lines in lin-log scale

→ Rem. 4.1.10

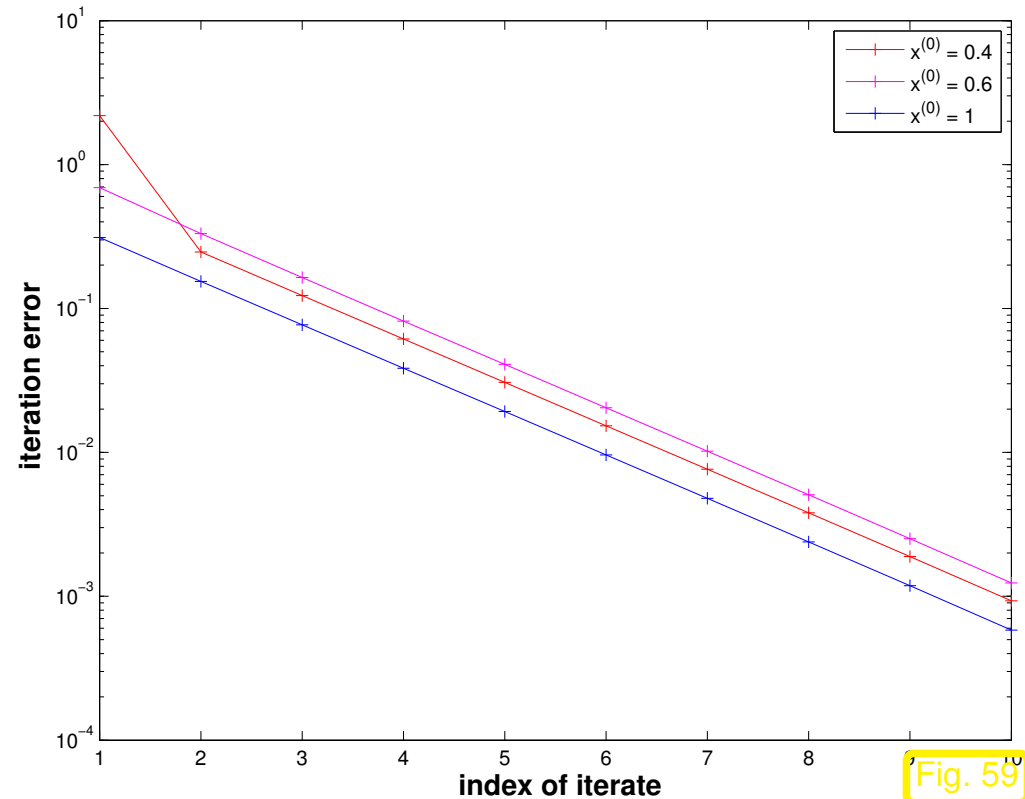


Fig. 59

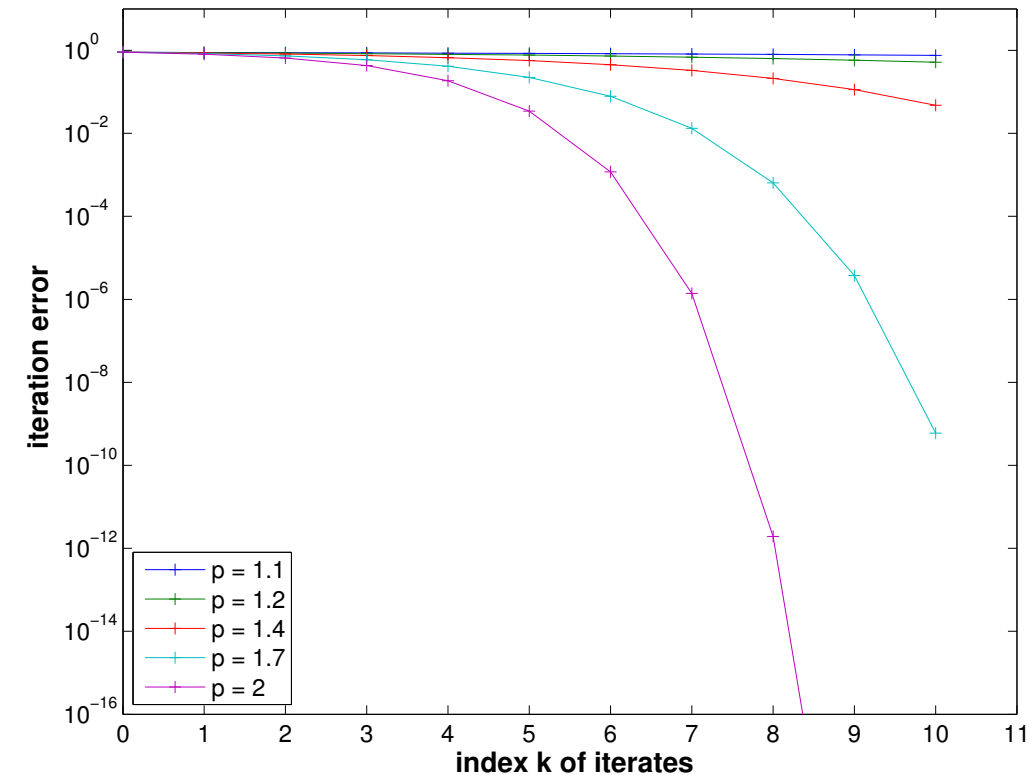
**Definition 4.1.14** (Order of convergence). → [35, Sect. 17.2], [13, Def. 5.14], [51, Def. 6.1]

A **convergent** sequence  $\mathbf{x}^{(k)}$ ,  $k = 0, 1, 2, \dots$ , in  $\mathbb{R}^n$  converges with **order**  $p$  to  $\mathbf{x}^* \in \mathbb{R}^n$ , if

$$\exists C > 0: \quad \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\| \leq C \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\|^p \quad \forall k \in \mathbb{N}_0,$$

and, in addition,  $C < 1$  in the case  $p = 1$  (linear convergence → Def. 4.1.6).

Of course, the order  $p$  of convergence of an iterative method refers to the largest possible  $p$  in the definition, that is, the error estimate will in general not hold, if  $p$  is replaced with  $p + \epsilon$  for any  $\epsilon > 0$ , cf. Rem. 1.3.1.



◁ Qualitative error graphs for convergence of order  $p$  (lin-log scale)

In the case of convergence of order  $p$  ( $p > 1$ ) (see Def. 4.1.14):

$$\begin{aligned} \epsilon_{k+1} \approx C \epsilon_k^p &\Rightarrow \log \epsilon_{k+1} = \log C + p \log \epsilon_k \Rightarrow \log \epsilon_{k+1} = \log C \sum_{l=0}^k p^l + p^{k+1} \log \epsilon_0 \\ &\Rightarrow \log \epsilon_{k+1} = -\frac{\log C}{p-1} + \left( \frac{\log C}{p-1} + \log \epsilon_0 \right) p^{k+1}. \end{aligned}$$

In this case, the error graph is a concave power curve (for sufficiently small  $\epsilon_0$  !)

*Remark 4.1.15* (Detecting order of convergence).

How to guess the order of convergence ( $\rightarrow$  Def. 4.1.14) from a numerical experiment?

Abbreviate  $\epsilon_k := \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\|$  (norm of iteration error):

$$\text{assume } \epsilon_{k+1} \approx C \epsilon_k^p \Rightarrow \log \epsilon_{k+1} \approx \log C + p \log \epsilon_k \Rightarrow \frac{\log \epsilon_{k+1} - \log \epsilon_k}{\log \epsilon_k - \log \epsilon_{k-1}} \approx p.$$

➤ monitor the quotients  $(\log \epsilon_{k+1} - \log \epsilon_k) / (\log \epsilon_k - \log \epsilon_{k-1})$  over several steps of the iteration.



Example 4.1.16 (quadratic convergence). (= convergence of order 2)

Iteration for computing  $\sqrt{a}$ ,  $a > 0$ :

$$x^{(k+1)} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right) \Rightarrow |x^{(k+1)} - \sqrt{a}| = \frac{1}{2x^{(k)}}|x^{(k)} - \sqrt{a}|^2. \quad (4.1.17)$$

By the arithmetic-geometric mean inequality (AGM)  $\sqrt{ab} \leq \frac{1}{2}(a + b)$  we conclude:  $x^{(k)} > \sqrt{a}$  for  $k \geq 1$ .

$\Rightarrow$  sequence from (4.1.17) converges with order 2 to  $\sqrt{a}$

Note:  $x^{(k+1)} < x^{(k)}$  for all  $k \geq 2 \Rightarrow (x^{(k)})_{k \in \mathbb{N}_0}$  converges as a decreasing sequence that is bounded from below ( $\rightarrow$  analysis course)

Numerical experiment: iterates for  $a = 2$ :

$k$	$x^{(k)}$	$e^{(k)} := x^{(k)} - \sqrt{2}$	$\log \frac{ e^{(k)} }{ e^{(k-1)} } : \log \frac{ e^{(k-1)} }{ e^{(k-2)} }$
0	2.000000000000000000	0.58578643762690485	
1	1.500000000000000000	0.08578643762690485	
2	1.416666666666666652	0.00245310429357137	1.850
3	1.41421568627450966	0.00000212390141452	1.984
4	1.41421356237468987	0.00000000000159472	2.000
5	1.41421356237309492	0.00000000000000022	0.630

Note the **doubling** of the number of significant digits in each step !

[impact of roundoff !]

The doubling of the number of significant digits for the iterates holds true for any convergent second-order iteration:

Indeed, denoting the relative error in step  $k$  by  $\delta_k$ , we have:

$$\begin{aligned}
 x^{(k)} &= x^*(1 + \delta_k) \Rightarrow x^{(k)} - x^* = \delta_k x^* . \\
 \Rightarrow |x^* \delta_{k+1}| &= |x^{(k+1)} - x^*| \leq C|x^{(k)} - x^*|^2 = C|x^* \delta_k|^2 \\
 &\Rightarrow |\delta_{k+1}| \leq C|x^*| \delta_k^2 . \tag{4.1.18}
 \end{aligned}$$

Note:  $\delta_k \approx 10^{-\ell}$  means that  $\mathbf{x}^{(k)}$  has  $\ell$  significant digits.

Also note that if  $C \approx 1$ , then  $\delta_k = 10^{-\ell}$  and (4.1.16) implies  $\delta_{k+1} \approx 10^{-2\ell}$ .

## 4.1.2 Termination criteria

Usually (even without roundoff errors) the iteration will never arrive at an/the exact solution  $\mathbf{x}^*$  after finitely many steps. Thus, we can only hope to compute an *approximate* solution by accepting  $\mathbf{x}^{(K)}$  as result for some  $K \in \mathbb{N}_0$ . Termination criteria (*ger.:* Abbruchbedingungen) are used to determine a suitable value for  $K$ .

For the sake of efficiency: ▷ stop iteration when iteration error is just “small enough”

“small enough” depends on concrete setting:



Usual goals:  $\left\| \mathbf{x}^{(K)} - \mathbf{x}^* \right\| \leq \tau_{\text{abs}}, \quad \tau_{\text{abs}} \hat{=} \text{prescribed (absolute) tolerance.}$

or

$\left\| \mathbf{x}^{(K)} - \mathbf{x}^* \right\| \leq \tau_{\text{rel}} \left\| \mathbf{x}^* \right\|, \quad \tau_{\text{rel}} \hat{=} \text{prescribed (relative) tolerance.}$

Ideal: STOP at step  $K = \operatorname{argmin}\{k \in \mathbb{N}_0 : \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| \leq \begin{cases} \tau_{\text{abs}} \\ \text{or} \\ \tau_{\text{rel}} \left\| \mathbf{x}^* \right\| \end{cases} . \quad (4.1.19)$

① **A priori termination:** stop iteration after fixed number of steps (possibly depending on  $\mathbf{x}^{(0)}$ ).



Drawback: hardly ever possible !

Invoking additional properties of either the non-linear system of equations  $F(\mathbf{x}) = 0$  or the iteration it is sometimes possible to tell that for sure  $\left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| \leq \tau$  for all  $k \geq K$ , though this  $K$  may be (significantly) larger than the optimal termination index from (4.1.19), see Rem. 4.1.23.

Alternative:

**A posteriori** termination criteria

use already computed iterates to decide when to stop

②

Reliable termination: stop iteration  $\{\mathbf{x}^{(k)}\}_{k \in \mathbb{N}_0}$  with limit  $\mathbf{x}^*$ , when

$$\left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| \leq \tau, \quad \tau \hat{=} \text{prescribed (absolute) tolerance} > 0. \quad (4.1.20)$$



$\mathbf{x}^*$  not known !

Sometimes extra knowledge about the type/speed of convergence allows to achieve **reliable termination** in the sense that (4.1.20) can be guaranteed though the number of iterations might be (slightly) too large.

For an example see Rem. 4.2.17, in particular (4.2.19), (4.2.20).

③ use that  $M$  is finite! ( $\rightarrow$  Sect. 2.4)

➤ possible to wait until (convergent) iteration becomes stationary

possibly grossly inefficient!  
(always computes “up to machine precision”)



Code 4.1.22: stationary iteration in  $M$ ,  $\rightarrow$

Ex. 4.1.16

```

1 function x = sqrtit(a)
2 x_old = -1; x = a;
3 while (x_old ~= x)
4     x_old = x;
5     x = 0.5*(x+a/x);
6 end

```

④ **Residual based** termination: STOP convergent iteration  $\{\mathbf{x}^{(k)}\}_{k \in \mathbb{N}_0}$ , when

$$\|F(\mathbf{x}^{(k)})\| \leq \tau, \quad \tau \hat{=} \text{prescribed tolerance} > 0.$$



no guaranteed accuracy

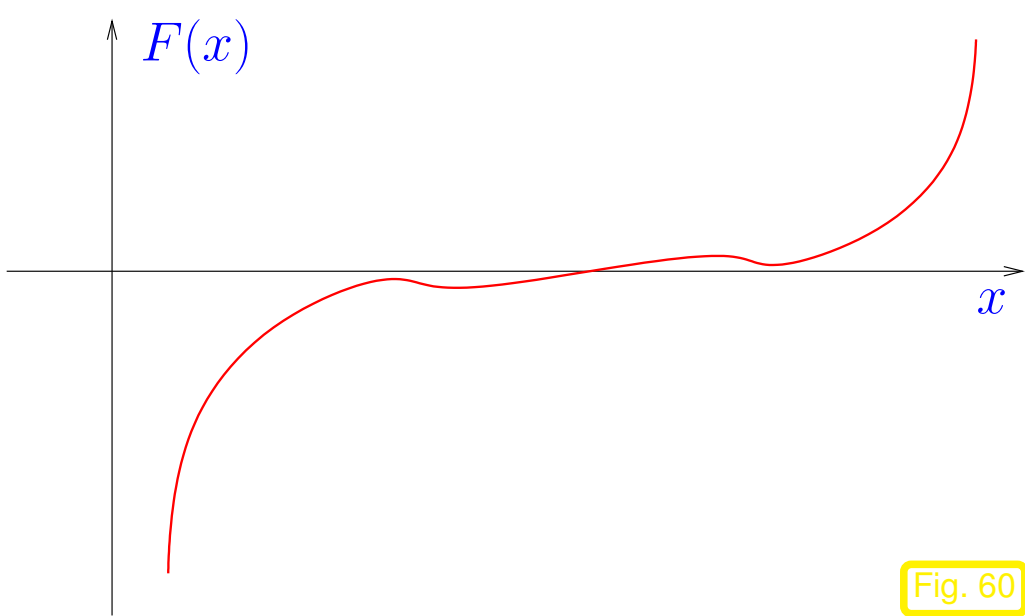


Fig. 60

$$\|F(\mathbf{x}^{(k)})\| \text{ small } \not\Rightarrow |x - x^*| \text{ small}$$

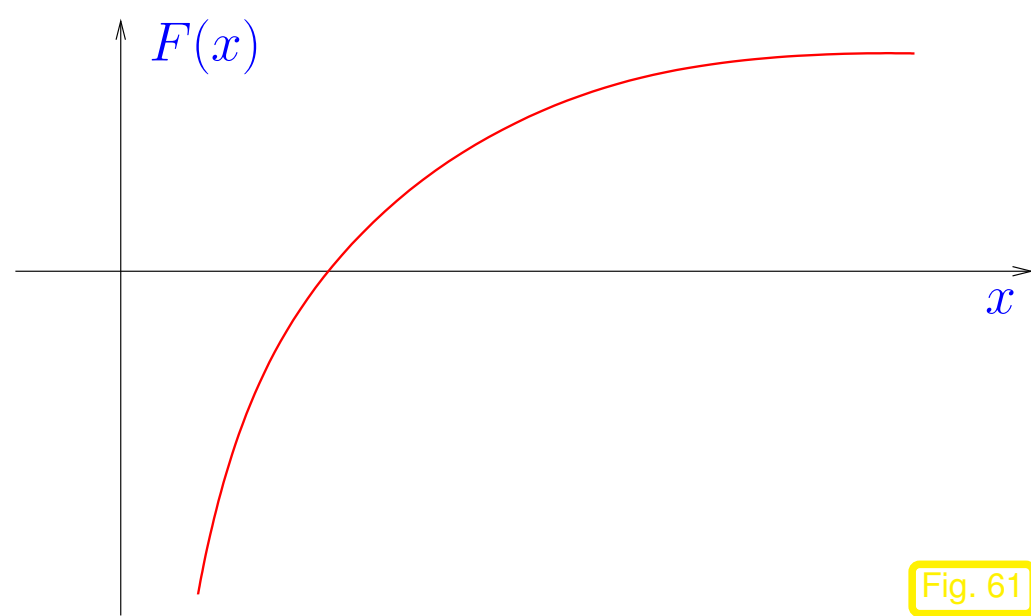


Fig. 61

$$\|F(\mathbf{x}^{(k)})\| \text{ small } \Rightarrow |x - x^*| \text{ small}$$

⑤

**Correction based** termination: STOP convergent iteration  $\{\mathbf{x}^{(k)}\}_{k \in \mathbb{N}_0}$ , when

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq \begin{cases} \tau_{\text{abs}} \\ \text{or} \\ \tau_{\text{rel}} \|\mathbf{x}^*\| \end{cases}, \quad \begin{matrix} \tau_{\text{abs}} \\ \tau_{\text{rel}} \end{matrix} \text{ prescribed absolute relative } \text{tolerances} > 0.$$

*Remark 4.1.23* (A posteriori termination criterion for linearly convergent iterations).  $\rightarrow$  [13, Lemma 5.

Known: iteration linearly convergent with rate of convergence  $0 < L < 1$ :

Derivation of a posteriori termination criterion for linearly convergent iterations with rate of convergence  $0 < L < 1$ :

$$\left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| \stackrel{\Delta\text{-inequ.}}{\leq} \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\| + \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\| \leq \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\| + L \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| .$$

Iterates satisfy:

$$\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\| \leq \frac{L}{1-L} \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\| . \quad (4.1.24)$$

This suggests that we take the right hand side of (4.1.24) as a posteriori error bound.



*Example 4.1.25* (A posteriori error bound for linearly convergent iteration).

Iteration of Example 4.1.12:

$$x^{(k+1)} = x^{(k)} + \frac{\cos x^{(k)} + 1}{\sin x^{(k)}} \Rightarrow x^{(k)} \rightarrow \pi \quad \text{for } x^{(0)} \text{ close to } \pi .$$

Observed rate of convergence:  $L = 1/2$

Error and error bound for  $x^{(0)} = 0.4$ :

$k$	$ x^{(k)} - \pi $	$\frac{L}{1-L} x^{(k)} - x^{(k-1)} $	slack of bound
1	2.191562221997101	4.933154875586894	2.741592653589793
2	0.247139097781070	1.944423124216031	1.697284026434961
3	0.122936737876834	0.124202359904236	0.001265622027401
4	0.061390835206217	0.061545902670618	0.000155067464401
5	0.030685773472263	0.030705061733954	0.000019288261691
6	0.015341682696235	0.015344090776028	0.000002408079792
7	0.007670690889185	0.007670991807050	0.000000300917864
8	0.003835326638666	0.003835364250520	0.000000037611854
9	0.001917660968637	0.001917665670029	0.000000004701392
10	0.000958830190489	0.000958830778147	0.000000000587658
11	0.000479415058549	0.000479415131941	0.000000000073392
12	0.000239707524646	0.000239707533903	0.000000000009257
13	0.000119853761949	0.000119853762696	0.000000000000747
14	0.000059926881308	0.000059926880641	0.000000000000667
15	0.000029963440745	0.000029963440563	0.000000000000181

Hence: the a posteriori error bound is highly accurate in this case!



Note: If  $L$  not known then using  $\tilde{L} > L$  in error bound is playing safe.

## 4.2 Fixed Point Iterations [13, Sect. 5.3], [51, Sect. 6.3]

Non-linear system of equations  $F(\mathbf{x}) = 0$ ,  $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ ,

R. Hiptmair  
rev 38286,  
October 3,  
2011

A **fixed point iteration** is defined by **iteration function**  $\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ :

iteration function  $\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$

initial guess  $\mathbf{x}^{(0)} \in U$

$\triangleright$  iterates  $(\mathbf{x}^{(k)})_{k \in \mathbb{N}_0}$ :  $\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})$  .  
 $\rightarrow$  1-point method, cf. (4.1.2)

Sequence of iterates need not be well defined:  $\mathbf{x}^{(k)} \notin U$  possible !

## 4.2.1 Consistent fixed point iterations

**Definition 4.2.1** (Consistency of fixed point iterations, *c.f.* Def. 4.1.4).

A fixed point iteration  $\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)})$  is **consistent** with  $F(\mathbf{x}) = 0$ , if

$$F(\mathbf{x}) = 0 \quad \text{and} \quad \mathbf{x} \in U \cap D \quad \Leftrightarrow \quad \Phi(\mathbf{x}) = \mathbf{x} .$$

R. Hiptmair  
rev 38286,  
October 3,  
2011

Note: iteration function  $\Phi$  continuous      **and**      fixed point iteration (locally) convergent to  $\mathbf{x}^* \in U$        $\Rightarrow$        $\mathbf{x}^*$  is a **fixed point** of  $\Phi$ .



General construction of fixed point iterations that is consistent with  $F(\mathbf{x}) = 0$ :

rewrite  $F(\mathbf{x}) = 0 \Leftrightarrow \Phi(\mathbf{x}) = \mathbf{x}$  and then

use the **fixed point iteration**  $\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})$ . (4.2.2)

Note: there are *many* ways to transform  $F(\mathbf{x}) = 0$  into a fixed point form !

*Example 4.2.3* (Options for fixed point iterations).

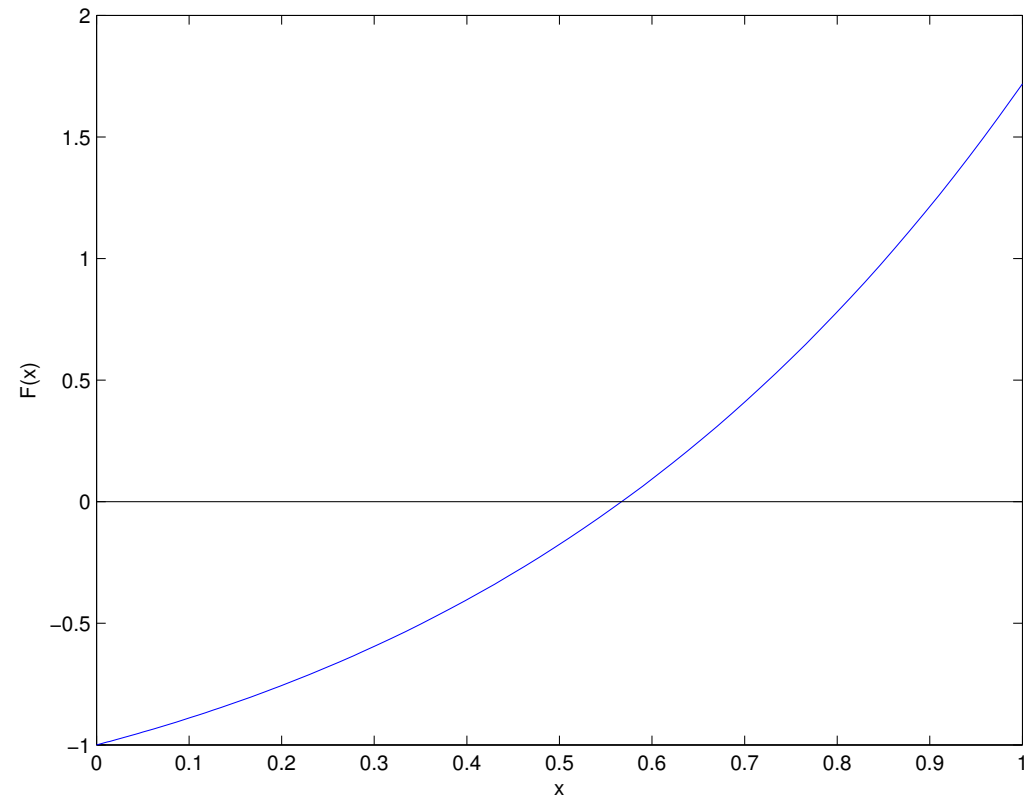
$$F(x) = xe^x - 1, \quad x \in [0, 1].$$

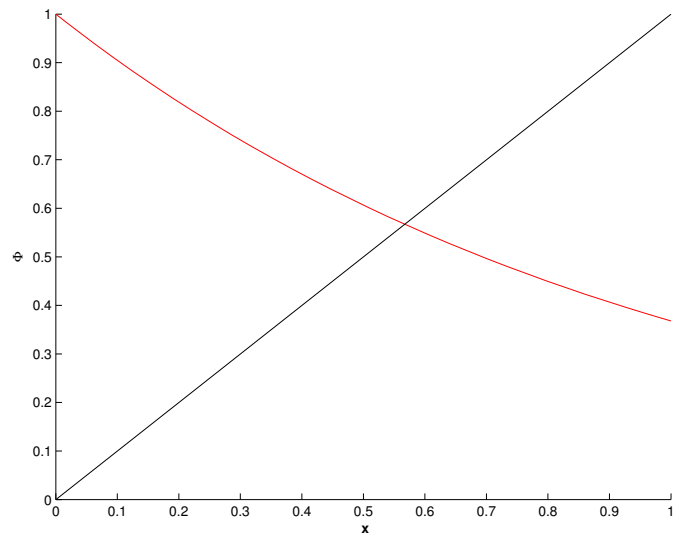
Different fixed point forms:

$$\Phi_1(x) = e^{-x},$$

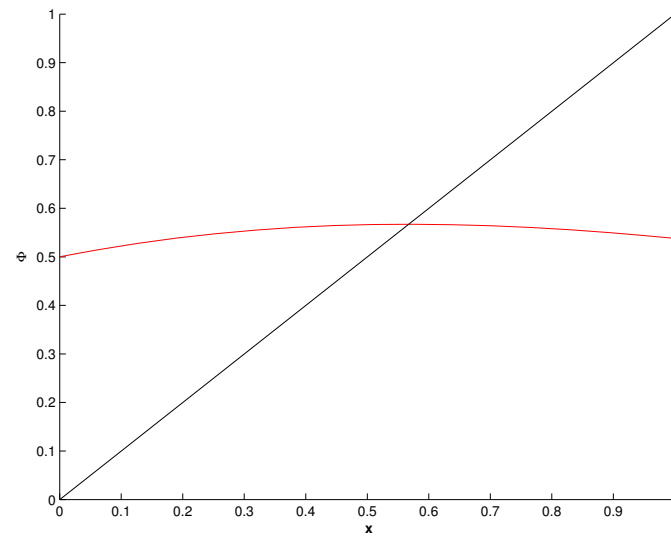
$$\Phi_2(x) = \frac{1+x}{1+e^x},$$

$$\Phi_3(x) = x + 1 - xe^x.$$

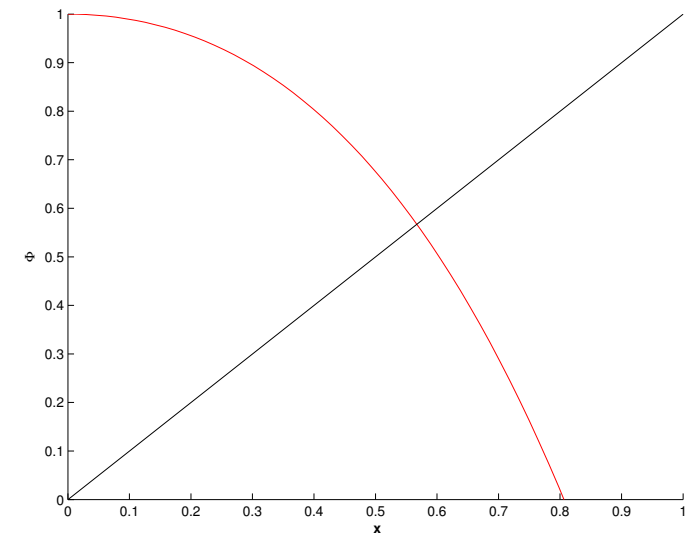




function  $\Phi_1$



function  $\Phi_2$



function  $\Phi_3$

$k$	$x^{(k+1)} := \Phi_1(x^{(k)})$	$x^{(k+1)} := \Phi_2(x^{(k)})$	$x^{(k+1)} := \Phi_3(x^{(k)})$
0	0.5000000000000000	0.5000000000000000	0.5000000000000000
1	0.606530659712633	0.566311003197218	0.675639364649936
2	0.545239211892605	0.567143165034862	0.347812678511202
3	0.579703094878068	0.567143290409781	0.855321409174107
4	0.560064627938902	0.567143290409784	-0.156505955383169
5	0.571172148977215	0.567143290409784	0.977326422747719
6	0.564862946980323	0.567143290409784	-0.619764251895580
7	0.568438047570066	0.567143290409784	0.713713087416146
8	0.566409452746921	0.567143290409784	0.256626649129847
9	0.567559634262242	0.567143290409784	0.924920676910549
10	0.566907212935471	0.567143290409784	-0.407422405542253

$k$	$ x_1^{(k+1)} - x^* $	$ x_2^{(k+1)} - x^* $	$ x_3^{(k+1)} - x^* $
0	0.067143290409784	0.067143290409784	0.067143290409784
1	0.039387369302849	0.000832287212566	0.108496074240152
2	0.021904078517179	0.000000125374922	0.219330611898582
3	0.012559804468284	0.0000000000000003	0.288178118764323
4	0.007078662470882	0.0000000000000000	0.723649245792953
5	0.004028858567431	0.0000000000000000	0.410183132337935
6	0.002280343429460	0.0000000000000000	1.186907542305364
7	0.001294757160282	0.0000000000000000	0.146569797006362
8	0.000733837662863	0.0000000000000000	0.310516641279937
9	0.000416343852458	0.0000000000000000	0.357777386500765
10	0.000236077474313	0.0000000000000000	0.974565695952037

Observed: linear convergence of  $x_1^{(k)}$ , quadratic convergence of  $x_2^{(k)}$ ,  
no convergence (erratic behavior of  $x_3^{(k)}$ ),  $x_i^{(0)} = 0.5$ .

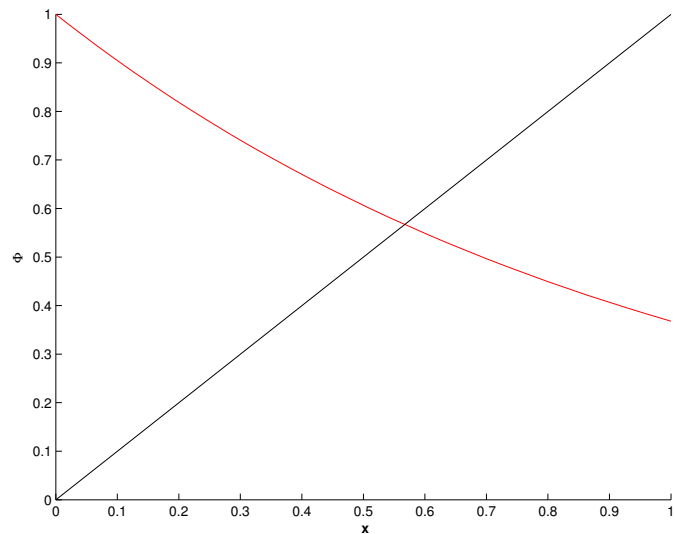


Question: can we explain/forecast the behaviour of the iteration?

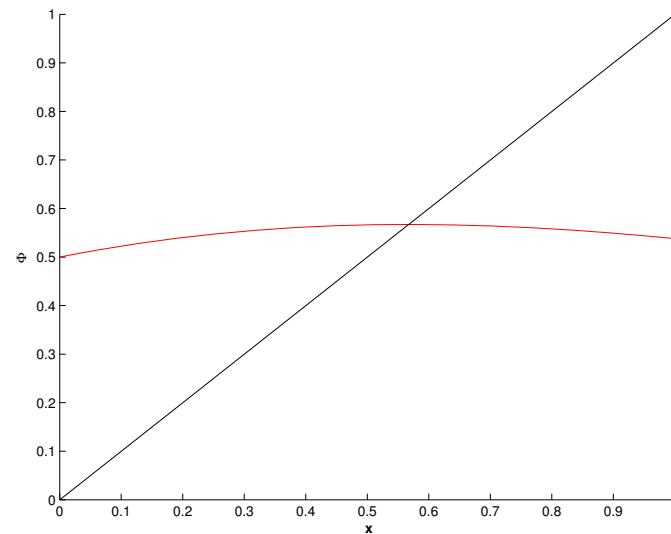
## 4.2.2 Convergence of fixed point iterations

In this section we will try to find easily verifiable conditions that ensure convergence (of a certain order) of fixed point iterations. It will turn out that these conditions are surprisingly simple and general.

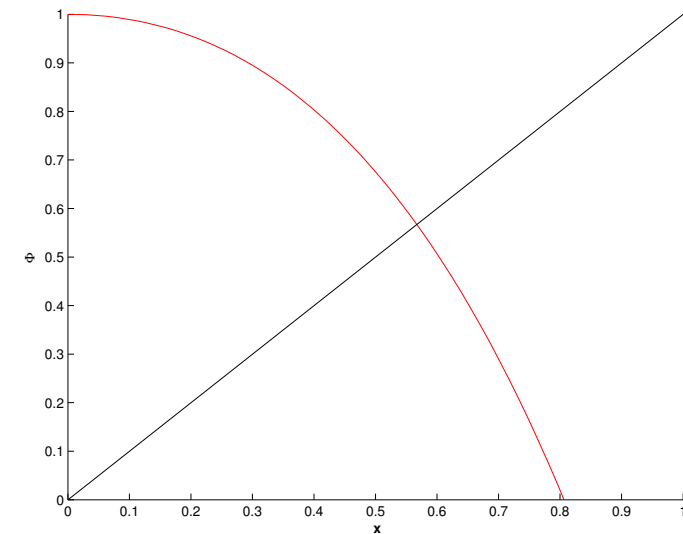
Ex. 4.2.3 revisited: vastly different behavior of different fixed point iterations for  $n = 1$ :



$\Phi_1$ : linear convergence ?



$\Phi_2$ : quadratic convergence ?



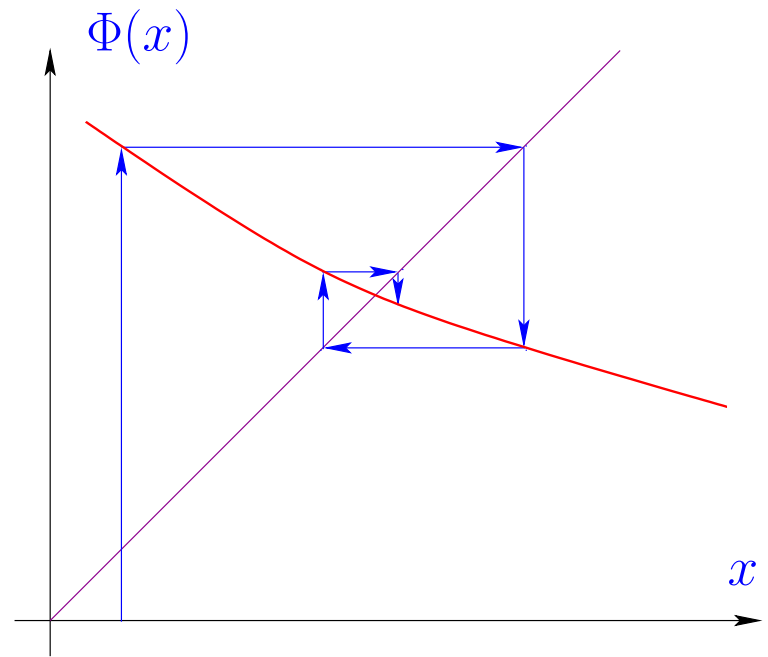
function  $\Phi_3$ : no convergence

Example 4.2.4 (Fixed point iteration in 1D).

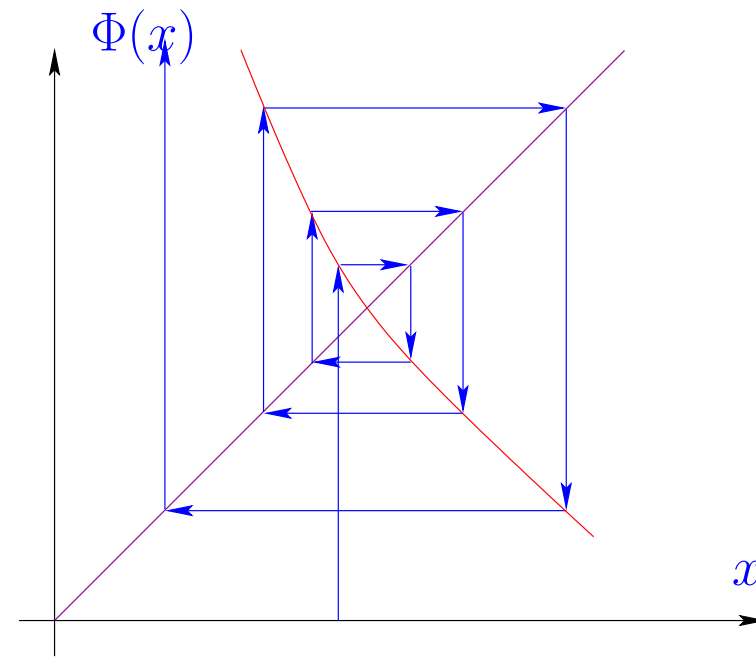
1D setting ( $n = 1$ ):  $\Phi : \mathbb{R} \mapsto \mathbb{R}$  continuously differentiable,  $\Phi(x^*) = x^*$

fixed point iteration:  $x^{(k+1)} = \Phi(x^{(k)})$

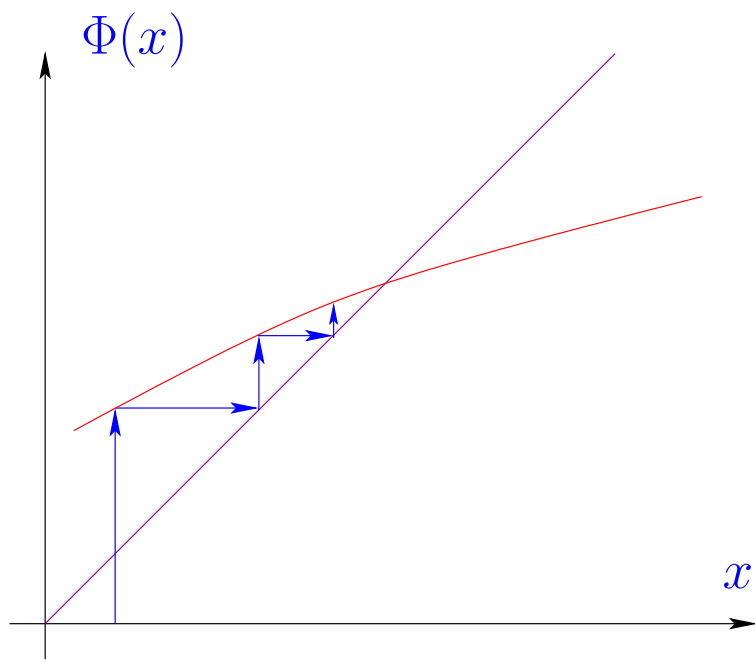
Visualization of different convergence behavior of fixed point iterations:



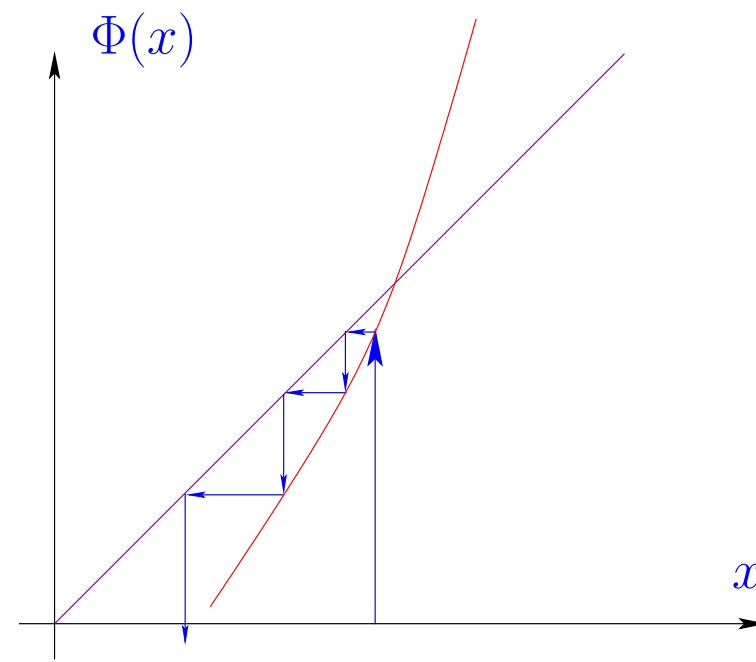
$-1 < \Phi'(x^*) \leq 0 \Rightarrow$  convergence



$\Phi'(x^*) < -1 \Rightarrow$  divergence



$0 \leq \Phi'(x^*) < 1 \quad \Rightarrow$  convergence



$1 < \Phi'(x^*) \quad \Rightarrow$  divergence

Numerical examples  $\Rightarrow$  Ex. 4.2.3, iteration functions  $\Phi_1$  and  $\Phi_3$



**Definition 4.2.5** (Contractive mapping).

$\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$  is **contractive** (w.r.t. norm  $\|\cdot\|$  on  $\mathbb{R}^n$ ), if

$$\exists L < 1: \quad \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in U. \quad (4.2.6)$$

A simple consideration: if  $\Phi(\mathbf{x}^*) = \mathbf{x}^*$  (fixed point), then a fixed point iteration induced by a contractive mapping  $\Phi$  satisfies

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| = \|\Phi(\mathbf{x}^{(k)}) - \Phi(\mathbf{x}^*)\| \stackrel{(4.2.8)}{\leq} L \|\mathbf{x}^{(k)} - \mathbf{x}^*\|,$$

that is, the iteration **converges** (at least) **linearly** ( $\rightarrow$  Def. 4.1.6).

Note that

$\Phi$  contractive  $\Rightarrow$   $\Phi$  has **at most one** fixed point.

*Remark 4.2.7* (Banach's fixed point theorem).  $\rightarrow$  [63, Satz 6.5.2], [13, Satz 5.8]

A key theorem in calculus (also functional analysis):

*Theorem 4.2.8* (Banach's fixed point theorem).

If  $D \subset \mathbb{K}^n$  ( $\mathbb{K} = \mathbb{R}, \mathbb{C}$ ) closed and bounded and  $\Phi : D \mapsto D$  satisfies

$$\exists L < 1: \quad \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in D,$$

then there is a unique fixed point  $\mathbf{x}^* \in D$ ,  $\Phi(\mathbf{x}^*) = \mathbf{x}^*$ , which is the limit of the sequence of iterates  $\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})$  for any  $\mathbf{x}^{(0)} \in D$ .

*Proof.* Proof based on 1-point iteration  $\mathbf{x}^{(k)} = \Phi(\mathbf{x}^{(k-1)})$ ,  $\mathbf{x}^{(0)} \in D$ :

$$\begin{aligned} \left\| \mathbf{x}^{(k+N)} - \mathbf{x}^{(k)} \right\| &\leq \sum_{j=k}^{k+N-1} \left\| \mathbf{x}^{(j+1)} - \mathbf{x}^{(j)} \right\| \leq \sum_{j=k}^{k+N-1} L^j \left\| \mathbf{x}^{(1)} - \mathbf{x}^{(0)} \right\| \\ &\leq \frac{L^k}{1-L} \left\| \mathbf{x}^{(1)} - \mathbf{x}^{(0)} \right\| \xrightarrow{k \rightarrow \infty} 0. \end{aligned}$$



$(\mathbf{x}^{(k)})_{k \in \mathbb{N}_0}$  Cauchy sequence  $\blacktriangleright$  convergent  $\mathbf{x}^{(k)} \xrightarrow{k \rightarrow \infty} \mathbf{x}^*$ .  
 Continuity of  $\Phi$   $\blacktriangleright$   $\Phi(\mathbf{x}^*) = \mathbf{x}^*$ . Uniqueness of fixed point is evident.  $\square$

A simple criterion for a differentiable  $\Phi$  to be contractive:

**Lemma 4.2.9** (Sufficient condition for local linear convergence of fixed point iteration).  $\rightarrow$  [35, Thm. 17.2], [13, Cor. 5.12]

If  $\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ ,  $\Phi(\mathbf{x}^*) = \mathbf{x}^*$ ,  $\Phi$  differentiable in  $\mathbf{x}^*$ , and  $\|D\Phi(\mathbf{x}^*)\| < 1$ , then the fixed point iteration (4.2.2) converges locally and at least linearly.

*matrix norm, Def. 2.5.5 !*

$\pencil$  notation:  $D\Phi(\mathbf{x}) \hat{=}$  **Jacobian** (*ger.:* Jacobi-Matrix) of  $\Phi$  at  $\mathbf{x} \in D$   
 $\rightarrow$  [63, Sect. 7.6]

“Visualization” of the statement of Lemma 4.2.9 in Ex. 4.2.4: The iteration converges *locally*, if  $\Phi$  is flat in a neighborhood of  $x^*$ , it will diverge, if  $\Phi$  is steep there.

*Proof.* (of Lemma 4.2.9) By definition of derivative

$$\|\Phi(\mathbf{y}) - \Phi(\mathbf{x}^*) - D\Phi(\mathbf{x}^*)(\mathbf{y} - \mathbf{x}^*)\| \leq \psi(\|\mathbf{y} - \mathbf{x}^*\|) \|\mathbf{y} - \mathbf{x}^*\| ,$$

with  $\psi : \mathbb{R}_0^+ \mapsto \mathbb{R}_0^+$  satisfying  $\lim_{t \rightarrow 0} \psi(t) = 0$ .

Choose  $\delta > 0$  such that

$$L := \psi(t) + \|D\Phi(\mathbf{x}^*)\| \leq \frac{1}{2}(1 + \|D\Phi(\mathbf{x}^*)\|) < 1 \quad \forall 0 \leq t < \delta .$$

By inverse triangle inequality we obtain for fixed point iteration

$$\begin{aligned} & \|\Phi(\mathbf{x}) - \mathbf{x}^*\| - \|D\Phi(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*)\| \leq \psi(\|\mathbf{x} - \mathbf{x}^*\|) \|\mathbf{x} - \mathbf{x}^*\| \\ \blacktriangleright \quad & \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\| \leq (\psi(t) + \|D\Phi(\mathbf{x}^*)\|) \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| \leq L \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| , \end{aligned}$$

if  $\left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| < \delta$ .

□

**Lemma 4.2.10** (Sufficient condition for linear convergence of fixed point iteration).

Let  $U$  be convex and  $\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$  be continuously differentiable with  $L := \sup_{\mathbf{x} \in U} \|D\Phi(\mathbf{x})\| < 1$ . If  $\Phi(\mathbf{x}^*) = \mathbf{x}^*$  for some interior point  $\mathbf{x}^* \in U$ , then the fixed point iteration  $\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)})$  converges to  $\mathbf{x}^*$  at least linearly.

Recall:  $U \subset \mathbb{R}^n$  convex  $:\Leftrightarrow (t\mathbf{x} + (1-t)\mathbf{y}) \in U$  for all  $\mathbf{x}, \mathbf{y} \in U, 0 \leq t \leq 1$

*Proof.* (of Lemma 4.2.10) By the mean value theorem

$$\begin{aligned} \Phi(\mathbf{x}) - \Phi(\mathbf{y}) &= \int_0^1 D\Phi(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x}))(\mathbf{y} - \mathbf{x}) \, d\tau \quad \forall \mathbf{x}, \mathbf{y} \in \text{dom}(\Phi) . \\ &\Rightarrow \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \leq L \|\mathbf{y} - \mathbf{x}\| , \\ &\Rightarrow \left\| (\mathbf{x})^{(k+1)} - \mathbf{x}^* \right\| \leq L \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| . \end{aligned}$$

We find that  $\Phi$  is contractive on  $U$  with unique fixed point  $\mathbf{x}^*$ , to which  $\mathbf{x}^{(k)}$  converges linearly for  $k \rightarrow \infty$ .

*Remark 4.2.11* (Bound for asymptotic rate of linear convergence).

If  $0 < \|D\Phi(\mathbf{x}^*)\| < 1$ ,  $\mathbf{x}^{(k)} \approx \mathbf{x}^*$  then the (worst) **asymptotic** rate of linear convergence is  $L = \|D\Phi(\mathbf{x}^*)\|$



*Example 4.2.12* (Multidimensional fixed point iteration).

$$\begin{array}{l} \text{System of equations} \qquad \qquad \qquad \text{in} \qquad \qquad \qquad \text{fixed point form:} \\ \left\{ \begin{array}{l} x_1 - c(\cos x_1 - \sin x_2) = 0 \\ (x_1 - x_2) - c \sin x_2 = 0 \end{array} \right. \Rightarrow \left\{ \begin{array}{l} c(\cos x_1 - \sin x_2) = x_1 \\ c(\cos x_1 - 2 \sin x_2) = x_2 \end{array} \right. \\ \text{Define: } \Phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = c \begin{pmatrix} \cos x_1 - \sin x_2 \\ \cos x_1 - 2 \sin x_2 \end{pmatrix} \Rightarrow D\Phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = -c \begin{pmatrix} \sin x_1 & \cos x_2 \\ \sin x_1 & 2 \cos x_2 \end{pmatrix}. \end{array}$$

Choose *appropriate* norm:  $\|\cdot\| = \infty\text{-norm } \|\cdot\|_\infty$  ( $\rightarrow$  Example 2.5.6);

$$\text{if } c < \frac{1}{3} \Rightarrow \|D\Phi(\mathbf{x})\|_\infty < 1 \quad \forall \mathbf{x} \in \mathbb{R}^2,$$

➤ (at least) linear convergence of the fixed point iteration.

The existence of a fixed point is also guaranteed, because  $\Phi$  maps into the closed set  $[-3, 3]^2$ . Thus, the Banach fixed point theorem, Thm. 4.2.8, can be applied.

What about higher order convergence ( $\rightarrow$  Def. 4.1.14, cf.  $\Phi_2$  in Ex. 4.2.3) ?

Refined convergence analysis for  $n = 1$  (scalar case,  $\Phi : \text{dom}(\Phi) \subset \mathbb{R} \mapsto \mathbb{R}$ ):

**Theorem 4.2.13** (Taylor's formula).  $\rightarrow$  [63, Sect. 5.5]

If  $\Phi : U \subset \mathbb{R} \mapsto \mathbb{R}$ ,  $U$  interval, is  $m + 1$  times continuously differentiable,  $x \in U$

$$\Phi(y) - \Phi(x) = \sum_{k=1}^m \frac{1}{k!} \Phi^{(k)}(x)(y-x)^k + O(|y-x|^{m+1}) \quad \forall y \in U. \quad (4.2.14)$$

Apply Taylor expansion (4.2.14) to iteration function  $\Phi$ :

If  $\Phi(x^*) = x^*$  and  $\Phi : \text{dom}(\Phi) \subset \mathbb{R} \mapsto \mathbb{R}$  is “sufficiently smooth”

$$x^{(k+1)} - x^* = \Phi(x^{(k)}) - \Phi(x^*) = \sum_{l=1}^m \frac{1}{l!} \Phi^{(l)}(x^*)(x^{(k)} - x^*)^l + O(|x^{(k)} - x^*|^{m+1}). \quad (4.2.15)$$

**Lemma 4.2.16** (Higher order local convergence of fixed point iterations).

If  $\Phi : U \subset \mathbb{R} \mapsto \mathbb{R}$  is  $m + 1$  times continuously differentiable,  $\Phi(x^*) = x^*$  for some  $x^*$  in the interior of  $U$ , and  $\Phi^{(l)}(x^*) = 0$  for  $l = 1, \dots, m$ ,  $m \geq 1$ , then the fixed point iteration (4.2.2) converges locally to  $x^*$  with **order**  $\geq m + 1$  ( $\rightarrow$  Def. 4.1.14).

*Proof.* For neighborhood  $\mathcal{U}$  of  $x^*$

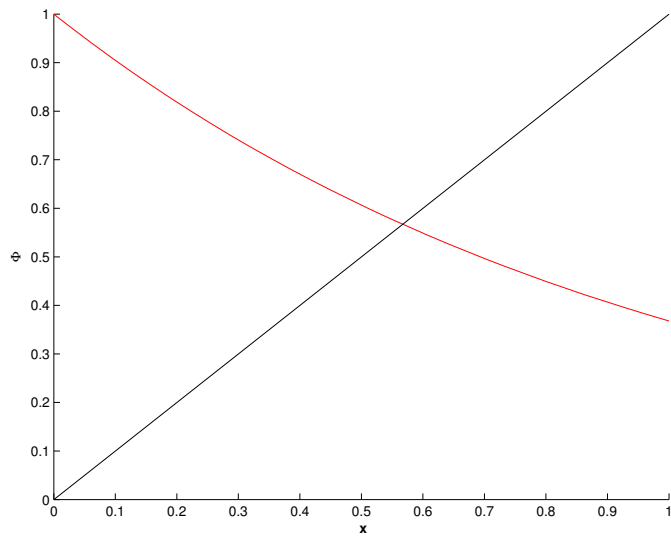
$$(4.2.15) \Rightarrow \exists C > 0: |\Phi(y) - \Phi(x^*)| \leq C |y - x^*|^{m+1} \quad \forall y \in \mathcal{U} .$$

$$\delta^m C < 1/2 : |x^{(0)} - x^*| < \delta \Rightarrow |x^{(k)} - x^*| < 2^{-k} \delta \quad \triangleright \text{local convergence} .$$

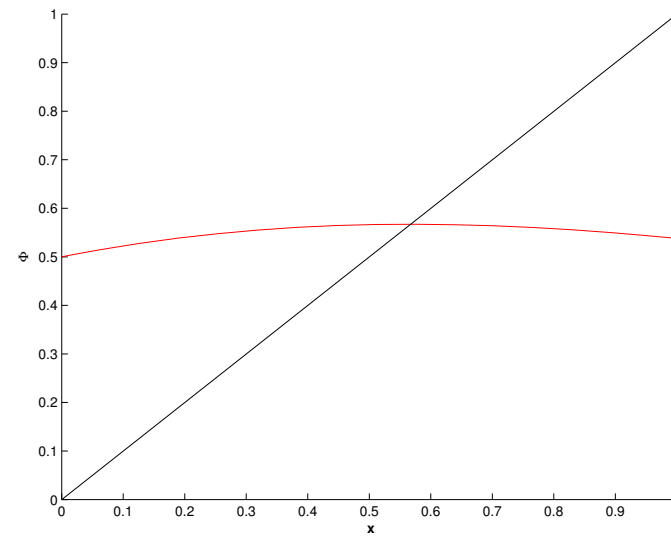
Then appeal to (4.2.15)

□

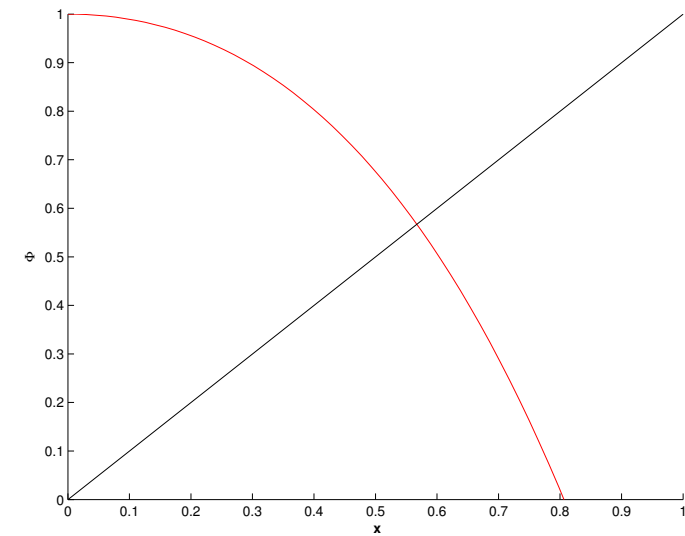
Example 4.2.3 continued:



function  $\Phi_1$



function  $\Phi_2$



function  $\Phi_3$

$$\Phi_2'(x) = \frac{1 - xe^x}{(1 + e^x)^2} = 0 \quad , \text{ if } \quad xe^x - 1 = 0 \quad \text{hence quadratic convergence ! .}$$

Example 4.2.3 continued: Since  $x^*e^{x^*} - 1 = 0$

$$\Phi_1'(x) = -e^{-x} \quad \Rightarrow \quad \Phi_1'(x^*) = -x^* \approx -0.56 \quad \text{hence local linear convergence .}$$

$$\Phi_3'(x) = 1 - xe^x - e^x \quad \Rightarrow \quad \Phi_3'(x^*) = -\frac{1}{x^*} \approx -1.79 \quad \text{hence no convergence .}$$

*Remark 4.2.17* (Termination criterion for contractive fixed point iteration).

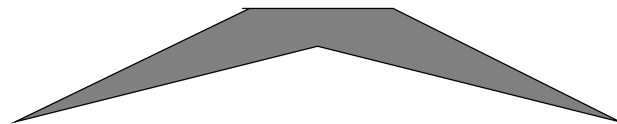
Recap of Rem. 4.1.23:

Termination criterion for contractive fixed point iteration, *c.f.* (4.2.8), with contraction factor  $0 \leq L < 1$ :

$$\begin{aligned} \left\| \mathbf{x}^{(k+m)} - \mathbf{x}^{(k)} \right\| &\stackrel{\Delta\text{-ineq.}}{\leq} \sum_{j=k}^{k+m-1} \left\| \mathbf{x}^{(j+1)} - \mathbf{x}^{(j)} \right\| \leq \sum_{j=k}^{k+m-1} L^{j-k} \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\| \\ &= \frac{1 - L^m}{1 - L} \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\| \leq \frac{1 - L^m}{1 - L} L^{k-l} \left\| \mathbf{x}^{(l+1)} - \mathbf{x}^{(l)} \right\|. \end{aligned}$$

hence for  $m \rightarrow \infty$ , with  $\mathbf{x}^* := \lim_{k \rightarrow \infty} \mathbf{x}^{(k)}$ :

$$\left\| \mathbf{x}^* - \mathbf{x}^{(k)} \right\| \leq \frac{L^{k-l}}{1 - L} \left\| \mathbf{x}^{(l+1)} - \mathbf{x}^{(l)} \right\|. \quad (4.2.18)$$





Set  $l = 0$  in (4.2.18)

a priori termination criterion

$$\left\| \mathbf{x}^* - \mathbf{x}^{(k)} \right\| \leq \frac{L^k}{1-L} \left\| \mathbf{x}^{(1)} - \mathbf{x}^{(0)} \right\| \quad (4.2.19)$$

Set  $l = k - 1$  in (4.2.18)

a posteriori termination criterion

$$\left\| \mathbf{x}^* - \mathbf{x}^{(k)} \right\| \leq \frac{L}{1-L} \left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \right\| \quad (4.2.20)$$



## 4.3 Zero Finding

Now, focus on scalar case  $n = 1$ : $F : I \subset \mathbb{R} \mapsto \mathbb{R}$  **continuous**,  $I$  interval

Sought:

$$x^* \in I: \quad F(x^*) = 0$$

## 4.3.1 Bisection [13, Sect. 5.5.1]

Idea: use ordering of real numbers & intermediate value theorem

Input:  $a, b \in I$  such that  $F(a)F(b) < 0$   
(different signs !)

$\Rightarrow \exists x^* \in ]\min\{a, b\}, \max\{a, b\}[$ :  
 $F(x^*) = 0$ .

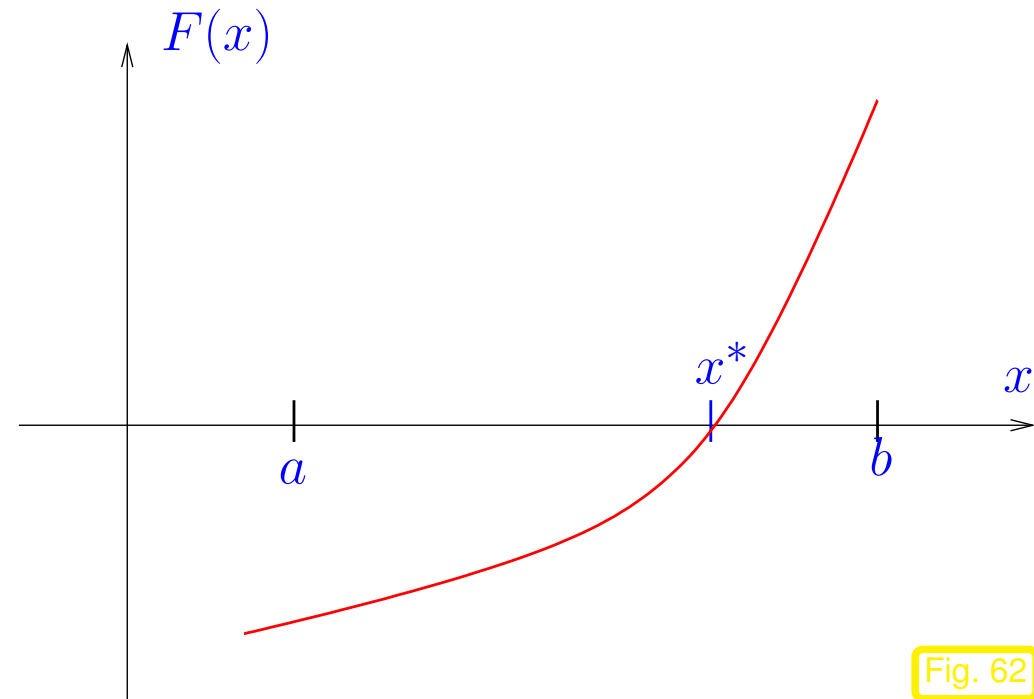


Fig. 62

*Algorithm 4.3.1* (Bisection method).

MATLAB-CODE: bisection method


```
function x = bisect(F,a,b,tol)
% Searching zero by bisection
if (a>b), t=a; a=b; b=t; end;
fa = F(a); fb = F(b);
if (fa*fb>0)
    error('f(a), f(b) same sign'); end;
if (fa > 0), v=-1; else v = 1; end
x = 0.5*(b+a);
while((b-a > tol) & ((a<x) & (x<b)))
    if (v*F(x)>0), b=x; else a=x; end;
    x = 0.5*(a+b)
end
```


 $\text{tol} \hat{=} \text{absolute tolerance}$ Handle to MATLAB function providing  $F$ .Avoid infinite loop, if  $\text{tol} < \text{resolution of } \mathbb{M} \text{ at zero } x^*$  (“ $\mathbb{M}$ -based termination criterion”).

This is an example for an algorithm that (in the case of  $\text{tol}=0$ ) uses the properties of machine arithmetic to define an a posteriori termination criterion, see Sect. 4.1.2. The iteration will terminate, when, e.g.,  $a + \frac{1}{2}(b - a) = a$ , which, by the Ass. 2.4.10 can only happen, when

$$\left| \frac{1}{2}(b - a) \right| \leq \text{eps} \cdot |a| .$$

Since the exact zero is located between  $a$  and  $b$ , this condition implies a relative error  $\leq \text{eps}$  of the computed zero.

-  Advantages:
- “foolproof”
  - requires only  $F$  evaluations

-  Drawbacks:
- Merely “linear-type” convergence:  $|x^{(k)} - x^*| \leq 2^{-k}|b - a|$
- ▶  $\log_2 \left( \frac{|b - a|}{\text{tol}} \right)$  steps necessary

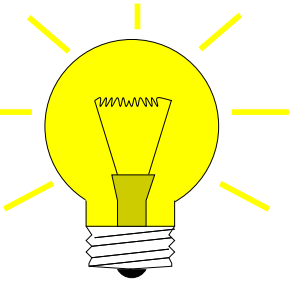
*Remark 4.3.2.* MATLAB function `fzero` is based on bisection approach.



## 4.3.2 Model function methods

$\hat{=}$  class of iterative methods for finding zeroes of  $F$ :

Idea: Given: approximate zeroes  $x^{(k)}, x^{(k-1)}, \dots, x^{(k-m)}$



- ❶ replace  $F$  with **model function**  $\tilde{F}$   
(using function values/derivative values in  $x^{(k)}, x^{(k-1)}, \dots, x^{(k-m)}$ )
- ❷  $x^{(k+1)} :=$  zero of  $\tilde{F}$   
(has to be readily available  $\leftrightarrow$  analytic formula)

Distinguish (see (4.1.2)):

one-point methods :  $x^{(k+1)} = \Phi_F(x^{(k)})$ ,  $k \in \mathbb{N}$  (e.g., fixed point iteration  $\rightarrow$  Sect. 4.2)

multi-point methods :  $x^{(k+1)} = \Phi_F(x^{(k)}, x^{(k-1)}, \dots, x^{(k-m)})$ ,  $k \in \mathbb{N}$ ,  $m = 2, 3, \dots$

#### 4.3.2.1 Newton method in scalar case [35, Sect. 18.1], [13, Sect. 5.5.2]

Assume:  $F : I \mapsto \mathbb{R}$  continuously differentiable

model function := tangent at  $F$  in  $x^{(k)}$ :

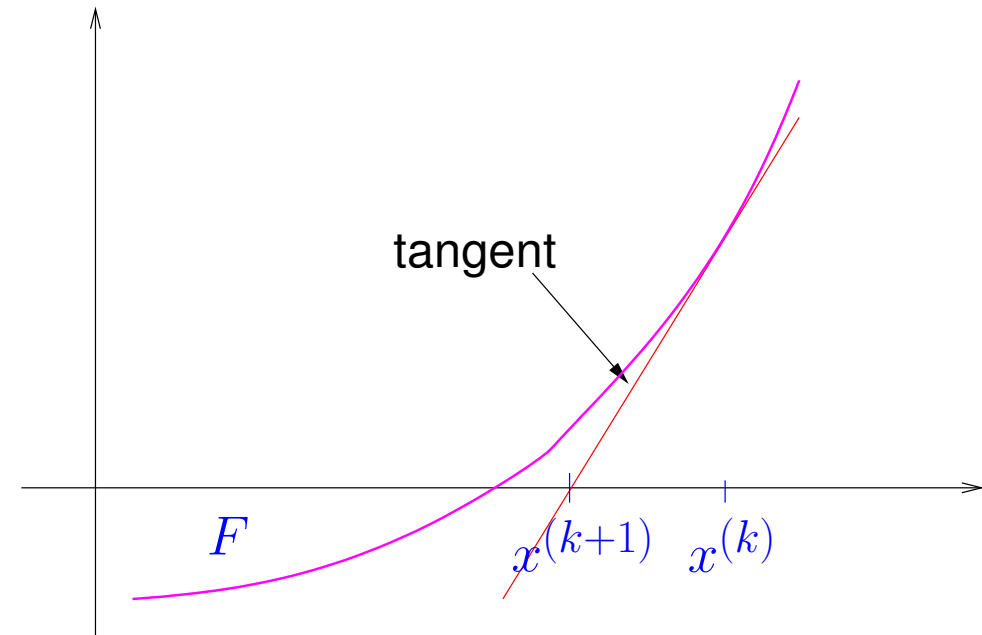
$$\tilde{F}(x) := F(x^{(k)}) + F'(x^{(k)})(x - x^{(k)})$$

take  $x^{(k+1)} :=$  zero of tangent

We obtain **Newton iteration**

$$x^{(k+1)} := x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})}, \quad (4.3.3)$$

that requires  $F'(x^{(k)}) \neq 0$ .



*Example 4.3.4* (Newton method in 1D). ( $\rightarrow$  Ex. 4.2.3)

Newton iterations for two different scalar non-linear equation with the same solution sets:

$$F(x) = xe^x - 1 \Rightarrow F'(x) = e^x(1+x) \Rightarrow x^{(k+1)} = x^{(k)} - \frac{x^{(k)}e^{x^{(k)}} - 1}{e^{x^{(k)}}(1+x^{(k)})} = \frac{(x^{(k)})^2 + e^{-x^{(k)}}}{1+x^{(k)}}$$

$$F(x) = x - e^{-x} \Rightarrow F'(x) = 1 + e^{-x} \Rightarrow x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - e^{-x^{(k)}}}{1 + e^{-x^{(k)}}} = \frac{1 + x^{(k)}}{1 + e^{x^{(k)}}}$$

Ex. 4.2.3 shows quadratic convergence ! ( $\rightarrow$  Def. 4.1.14)

Newton iteration (4.3.3)  $\hat{=}$  fixed point iteration ( $\rightarrow$  Sect.4.2) with iteration function

$$\Phi(x) = x - \frac{F(x)}{F'(x)} \Rightarrow \Phi'(x) = \frac{F(x)F''(x)}{(F'(x))^2} \Rightarrow \Phi'(x^*) = 0, \text{ if } F(x^*) = 0, F'(x^*) \neq 0.$$

From Lemma 4.2.16:

Newton method locally quadratically convergent ( $\rightarrow$  Def. 4.1.14) to zero  $x^*$ , if  $F'(x^*) \neq 0$

### 4.3.2.2 Special one-point methods

Idea underlying other one-point methods: non-linear local approximation

Useful, if *a priori knowledge* about the structure of  $F$  (e.g. about  $F$  being a rational function, see below) is available. This is often the case, because many problems of 1D zero finding are posed for functions given in analytic form with a few parameters.

Prerequisite: Smoothness of  $F$ :  $F \in C^m(I)$  for some  $m > 1$

*Example 4.3.5* (Halley's iteration).  $\rightarrow$  [35, Sect. 18.3]

This example demonstrates that non-polynomial model functions can offer excellent approximation of  $F$ . In this example the model function is chosen as a quotient of two linear function, that is, from the simplest class of true rational functions.

Of course, that this function provides a good model function is merely “a matter of luck”, unless you have some more information about  $F$ . Such information might be available from the application context.



Given  $x^{(k)} \in I$ , next iterate := zero of model function:  $h(x^{(k+1)}) = 0$ , where

$$h(x) := \frac{a}{x+b} + c \quad (\text{rational function}) \text{ such that } F^{(j)}(x^{(k)}) = h^{(j)}(x^{(k)}), \quad j = 0, 1, 2.$$

$$\frac{a}{x^{(k)} + b} + c = F(x^{(k)}), \quad -\frac{a}{(x^{(k)} + b)^2} = F'(x^{(k)}), \quad \frac{2a}{(x^{(k)} + b)^3} = F''(x^{(k)}).$$

$$x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})} \cdot \frac{1}{1 - \frac{1}{2} \frac{F(x^{(k)})F''(x^{(k)})}{F'(x^{(k)})^2}}.$$

Halley's iteration for  $F(x) = \frac{1}{(x+1)^2} + \frac{1}{(x+0.1)^2} - 1$ ,  $x > 0$ : and  $x^{(0)} = 0$

$k$	$x^{(k)}$	$F(x^{(k)})$	$x^{(k)} - x^{(k-1)}$	$x^{(k)} - x^*$
1	0.19865959351191	10.90706835180178	-0.19865959351191	-0.84754290138257
2	0.69096314049024	0.94813655914799	-0.49230354697833	-0.35523935440424
3	1.02335017694603	0.03670912956750	-0.33238703645579	-0.02285231794846
4	1.04604398836483	0.00024757037430	-0.02269381141880	-0.00015850652965
5	1.04620248685303	0.00000001255745	-0.00015849848821	-0.00000000804145

Compare with Newton method (4.3.3) for the same problem:

$k$	$x^{(k)}$	$F(x^{(k)})$	$x^{(k)} - x^{(k-1)}$	$x^{(k)} - x^*$
1	0.04995004995005	44.38117504792020	-0.04995004995005	-0.99625244494443
2	0.12455117953073	19.62288236082625	-0.07460112958068	-0.92165131536375
3	0.23476467495811	8.57909346342925	-0.11021349542738	-0.81143781993637
4	0.39254785728080	3.63763326452917	-0.15778318232269	-0.65365463761368
5	0.60067545233191	1.42717892023773	-0.20812759505112	-0.44552704256257
6	0.82714994286833	0.46286007749125	-0.22647449053641	-0.21905255202615
7	0.99028203077844	0.09369191826377	-0.16313208791011	-0.05592046411604
8	1.04242438221432	0.00592723560279	-0.05214235143588	-0.00377811268016
9	1.04618505691071	0.00002723158211	-0.00376067469639	-0.00001743798377
10	1.04620249452271	0.00000000058056	-0.00001743761199	-0.00000000037178

Note that Halley's iteration is superior in this case, since  $F$  is a rational function.

! Newton method converges more slowly, but also needs less effort per step ( $\rightarrow$  Sect. 4.3.3)  $\diamond$

In the previous example Newton's method performed rather poorly. Often its convergence can be boosted by converting the non-linear equation to an equivalent one (that is, one with the same solutions) for another function  $g$ , which is "closer to a linear function":

Assume  $F \approx \widehat{F}$ , where  $\widehat{F}$  is invertible with an inverse  $\widehat{F}^{-1}$  that can be evaluated with little effort.

$$\blacktriangleright \quad g(x) := \widehat{F}^{-1}(F(x)) \approx x .$$

Then apply Newton's method to  $g(x)$ , using the formula for the derivative of the inverse of a function

$$\frac{d}{dy}(\widehat{F}^{-1})(y) = \frac{1}{\widehat{F}'(\widehat{F}^{-1}(y))} \quad \Rightarrow \quad g'(x) = \frac{1}{\widehat{F}'(g(x))} \cdot F'(x) .$$

*Example 4.3.6 (Adapted Newton method).*

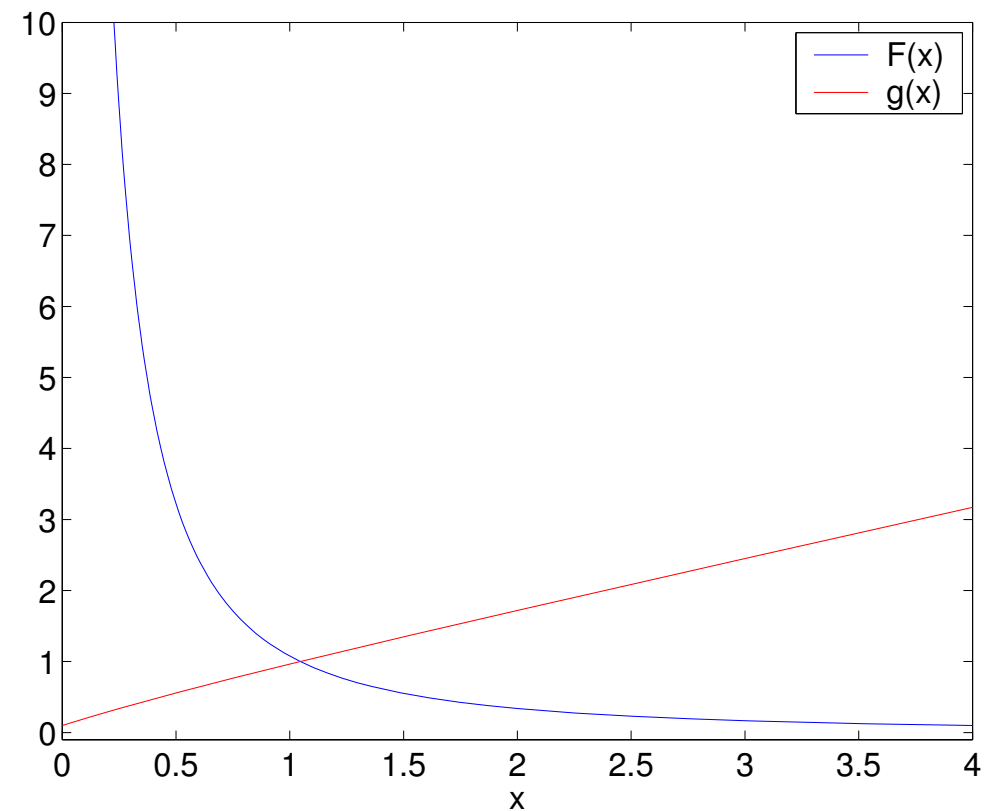
As in Ex. 4.3.5:

$$F(x) = \frac{1}{(x+1)^2} + \frac{1}{(x+0.1)^2} - 1, \quad x > 0 :$$

Observation:

$$F(x) + 1 \approx 2x^{-2} \text{ for } x \gg 1$$

and so  $g(x) := \frac{1}{\sqrt{F(x) + 1}}$  “almost” linear for  $x \gg 1$



Idea: instead of  $F(x) \stackrel{!}{=} 0$  tackle  $g(x) \stackrel{!}{=} 1$  with Newton's method (4.3.3).

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - \frac{g(x^{(k)}) - 1}{g'(x^{(k)})} = x^{(k)} + \left( \frac{1}{\sqrt{F(x^{(k)}) + 1}} - 1 \right) \frac{2(F(x^{(k)}) + 1)^{3/2}}{F'(x^{(k)})} \\ &= x^{(k)} + \frac{2(F(x^{(k)}) + 1)(1 - \sqrt{F(x^{(k)}) + 1})}{F'(x^{(k)})}. \end{aligned}$$

Convergence recorded for  $x^{(0)} = 0$ :

$k$	$x^{(k)}$	$F(x^{(k)})$	$x^{(k)} - x^{(k-1)}$	$x^{(k)} - x^*$
1	0.91312431341979	0.24747993091128	0.91312431341979	-0.13307818147469
2	1.04517022155323	0.00161402574513	0.13204590813344	-0.00103227334125
3	1.04620244004116	0.00000008565847	0.00103221848793	-0.00000005485332
4	1.04620249489448	0.00000000000000	0.00000005485332	-0.00000000000000



For zero finding there is wealth of iterative methods that offer higher order of convergence.

One idea: **consistent modification** of the Newton-Iteration:

► fixed point iteration :  $\Phi(x) = x - \frac{F(x)}{F'(x)}H(x)$  with "proper"  $H : I \mapsto \mathbb{R}$ .

Aim: find  $H$  such that the method is of  $p$ -th order; tool: Lemma 4.2.16.

Assume:  $F$  smooth "enough" and  $\exists x^* \in I: F(x^*) = 0, F'(x^*) \neq 0$ .

$$\Phi = x - uH \quad , \quad \Phi' = 1 - u'H - uH' \quad , \quad \Phi'' = -u''H - 2u'H - uH'' \quad ,$$

$$\text{with } u = \frac{F}{F'} \Rightarrow u' = 1 - \frac{FF''}{(F')^2}, \quad u'' = -\frac{F''}{F'} + 2\frac{F(F'')^2}{(F')^3} - \frac{FF'''}{(F')^2}.$$

$$F(x^*) = 0 \quad \blacktriangleright \quad u(x^*) = 0, u'(x^*) = 1, u''(x^*) = -\frac{F''(x^*)}{F'(x^*)}.$$

$$\blacktriangleright \quad \Phi'(x^*) = 1 - H(x^*) \quad , \quad \Phi''(x^*) = \frac{F''(x^*)}{F'(x^*)}H(x^*) - 2H'(x^*) . \quad (4.3.7)$$

Lemma 4.2.16  $\blacktriangleright$  **Necessary** conditions for local convergence of order  $p$ :

$$p = 2 \text{ (quadratical convergence): } H(x^*) = 1 ,$$

$$p = 3 \text{ (cubic convergence): } H(x^*) = 1 \quad \wedge \quad H'(x^*) = \frac{1}{2} \frac{F''(x^*)}{F'(x^*)} .$$

In particular:  $H(x) = G(1 - u'(x))$  with "proper"  $G$

$$\blacktriangleright \quad \text{fixed point iteration} \quad x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})} G \left( \frac{F(x^{(k)})F''(x^{(k)})}{(F'(x^{(k)}))^2} \right) . \quad (4.3.8)$$

**Lemma 4.3.9.** *If  $F \in C^2(I)$ ,  $F(x^*) = 0$ ,  $F'(x^*) \neq 0$ ,  $G \in C^2(U)$  in a neighbourhood  $U$  of 0,  $G(0) = 1$ ,  $G'(0) = \frac{1}{2}$ , then the fixed point iteration (4.3.8) converge locally cubically to  $x^*$ .*

*Proof:* Lemma 4.2.16, (4.3.7) and

$$H(x^*) = G(0) \quad , \quad H'(x^*) = -G'(0)u''(x^*) = G'(0)\frac{F''(x^*)}{F'(x^*)} .$$

*Example 4.3.10* (Example of modified Newton method).

- $G(t) = \frac{1}{1 - \frac{1}{2}t}$      $\Rightarrow$  Halley's iteration ( $\rightarrow$  Ex. 4.3.5)
- $G(t) = \frac{2}{1 + \sqrt{1 - 2t}}$      $\Rightarrow$  Euler's iteration
- $G(t) = 1 + \frac{1}{2}t$      $\Rightarrow$  quadratic inverse interpolation

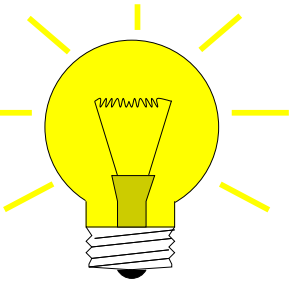
$k$	$e^{(k)} := x^{(k)} - x^*$		
	Halley	Euler	Quad. Inv.
1	2.81548211105635	3.57571385244736	2.03843730027891
2	1.37597082614957	2.76924150041340	1.02137913293045
3	0.34002908011728	1.95675490333756	0.28835890388161
4	0.00951600547085	1.25252187565405	0.01497518178983
5	0.00000024995484	0.51609212477451	0.000000315361454

Numerical experiment:

$$F(x) = xe^x - 1 ,$$

$$x^{(0)} = 5$$

### 4.3.2.3 Multi-point methods



Idea:

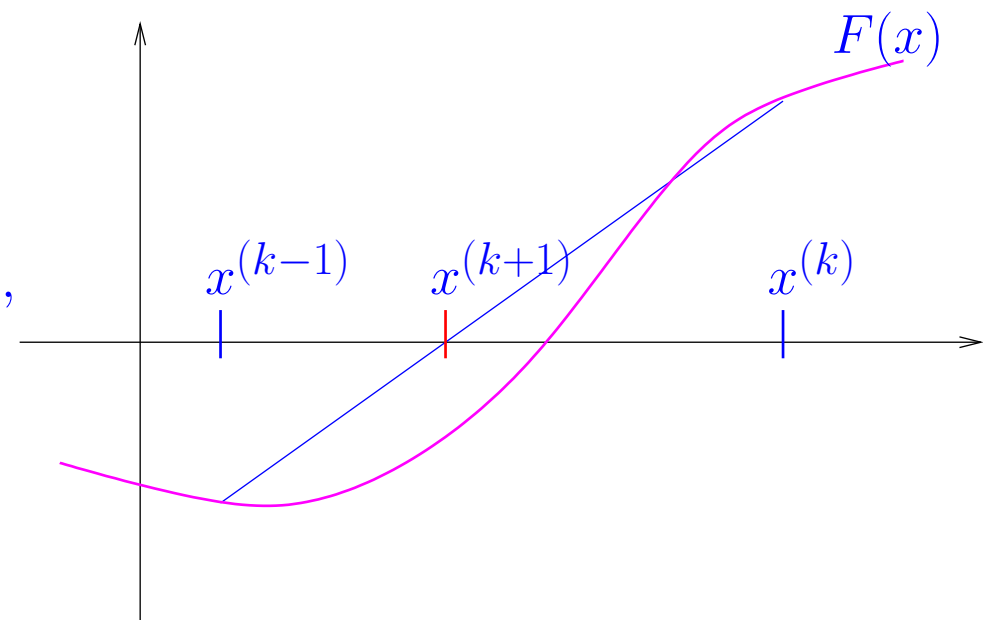
Replace  $F$  with **interpolating polynomial**  
producing interpolatory model function methods

Simplest representative: **secant method** → [35,  
Sect. 18.2], [13, Sect, 5.5.3]

$x^{(k+1)}$  = zero of secant

$$s(x) = F(x^{(k)}) + \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}(x - x^{(k)}), \quad (4.3.11)$$

▶ 
$$x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})(x^{(k)} - x^{(k-1)})}{F(x^{(k)}) - F(x^{(k-1)})}. \quad (4.3.12)$$





## Code 4.3.13: secant method

```
1 function x = secant(x0,x1,F,tol)
2 fo = F(x0);
3 for i=1:MAXIT
4     fn = F(x1);
5     s = fn*(x1-x0)/(fn-fo);
6     x0 = x1; x1 = x1-s;
7     if (abs(s) < tol), x = x1;
8         return; end
9 end
```

secant method  
( MATLAB implementation)

- Only one function evaluation per step
- **no derivatives required !**

Remember:  $F(x)$  may only be available as output of a (complicated) procedure. In this case it is difficult to find a procedure that evaluates  $F'(x)$ . Thus the significance of methods that do not involve evaluations of derivatives.

*Example 4.3.14* (secant method).  $F(x) = xe^x - 1$ ,  $x^{(0)} = 0$ ,  $x^{(1)} = 5$ .

$k$	$x^{(k)}$	$F(x^{(k)})$	$e^{(k)} := x^{(k)} - x^*$	$\frac{\log  e^{(k+1)}  - \log  e^{(k)} }{\log  e^{(k)}  - \log  e^{(k-1)} }$
2	0.00673794699909	-0.99321649977589	-0.56040534341070	
3	0.01342122983571	-0.98639742654892	-0.55372206057408	24.43308649757745
4	0.98017620833821	1.61209684919288	0.41303291792843	2.70802321457994
5	0.38040476787948	-0.44351476841567	-0.18673852253030	1.48753625853887
6	0.50981028847430	-0.15117846201565	-0.05733300193548	1.51452723840131
7	0.57673091089295	0.02670169957932	0.00958762048317	1.70075240166256
8	0.56668541543431	-0.00126473620459	-0.00045787497547	1.59458505614449
9	0.56713970649585	-0.00000990312376	-0.00000358391394	1.62641838319117
10	0.56714329175406	0.00000000371452	0.00000000134427	
11	0.56714329040978	-0.000000000000001	-0.000000000000000	

Rem. 4.1.15: the rightmost column of the table provides an estimate for the order of convergence →  
Def. 4.1.14.

A startling observation: the method seems to have a *fractional* (!) order of convergence, see  
Def. 4.1.14.

*Remark 4.3.15* (Fractional order of convergence of secant method).

Indeed, a fractional order of convergence can be proved for the secant method, see [35, Sect. 18.2].

Here is a crude outline of the reasoning:

**Asymptotic** convergence of secant method: error  $e^{(k)} := x^{(k)} - x^*$

$$e^{(k+1)} = \Phi(x^* + e^{(k)}, x^* + e^{(k-1)}) - x^* \quad , \text{ with } \Phi(x, y) := x - \frac{F(x)(x - y)}{F(x) - F(y)} . \quad (4.3.16)$$

Use MAPLE to find Taylor expansion (assuming  $F$  sufficiently smooth):

```
> Phi := (x, y) -> x - F(x) * (x - y) / (F(x) - F(y));
> F(s) := 0;
> e2 = normal(mtaylor(Phi(s+e1, s+e0) - s, [e0, e1], 4));
```

➤ linearized **error propagation formula**:

$$e^{(k+1)} \doteq \frac{1}{2} \frac{F''(x^*)}{F'(x^*)} e^{(k)} e^{(k-1)} = C e^{(k)} e^{(k-1)} . \quad (4.3.17)$$

Try  $e^{(k)} = K(e^{(k-1)})^p$  to determine the order of convergence ( $\rightarrow$  Def. 4.1.14):

$$\begin{aligned} &\Rightarrow e^{(k+1)} = K^{p+1}(e^{(k-1)})^{p^2} \\ &\Rightarrow (e^{(k-1)})^{p^2-p-1} = K^{-p}C \Rightarrow p^2 - p - 1 = 0 \Rightarrow p = \frac{1}{2}(1 \pm \sqrt{5}). \end{aligned}$$

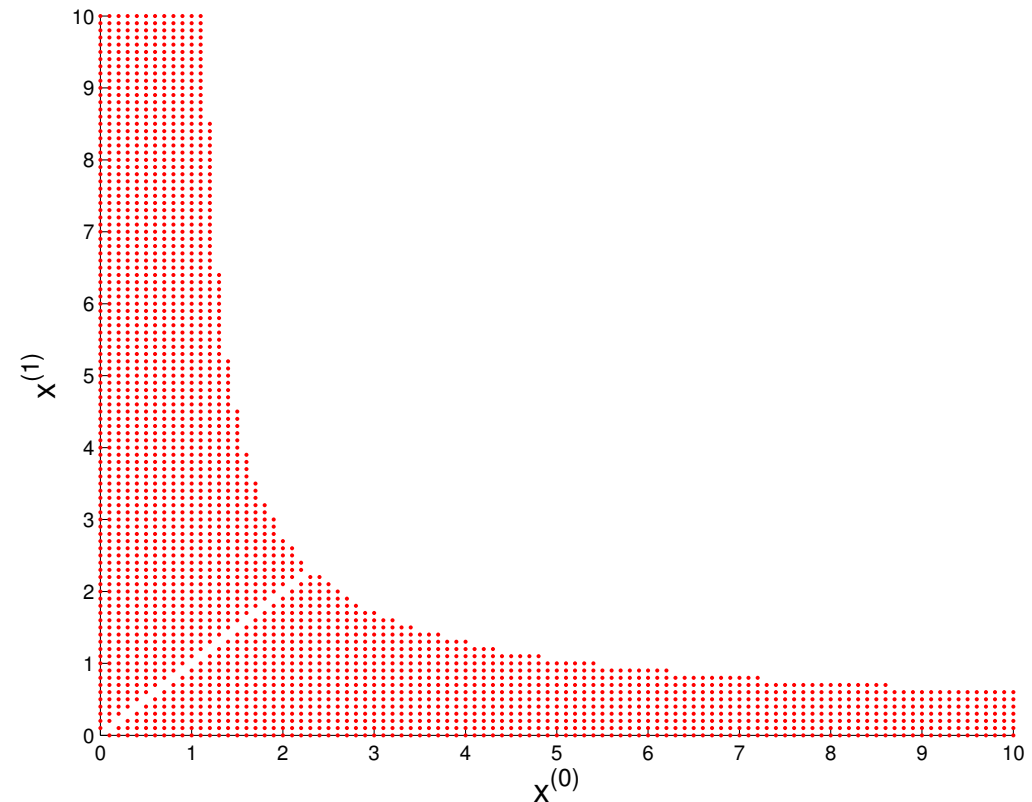
As  $e^{(k)} \rightarrow 0$  for  $k \rightarrow \infty$  we get the order of convergence  $p = \frac{1}{2}(1 + \sqrt{5}) \approx 1.62$  (see Ex. 4.3.14 !)

Example 4.3.18 (local convergence of secant method).

$$F(x) = \arctan(x)$$

•  $\hat{=}$  secant method converges for a pair  $(x^{(0)}, x^{(1)})$  of initial guesses.

= local convergence  $\rightarrow$  Def. 4.1.5



Another class of multi-point methods: *inverse interpolation*

Assume:

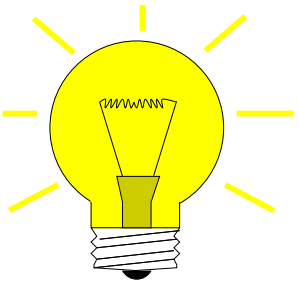
$$F : I \subset \mathbb{R} \mapsto \mathbb{R} \text{ one-to-one}$$

$$F(x^*) = 0 \Rightarrow F^{-1}(0) = x^* .$$

- Interpolate  $F^{-1}$  by polynomial  $p$  of degree  $d$  determined by

$$p(F(x^{(k-m)})) = x^{(k-m)} , \quad m = 0, \dots, d .$$

- New approximate zero  $x^{(k+1)} := p(0)$



$$F(x^*) = 0 \Leftrightarrow F^{-1}(0) = x^*$$

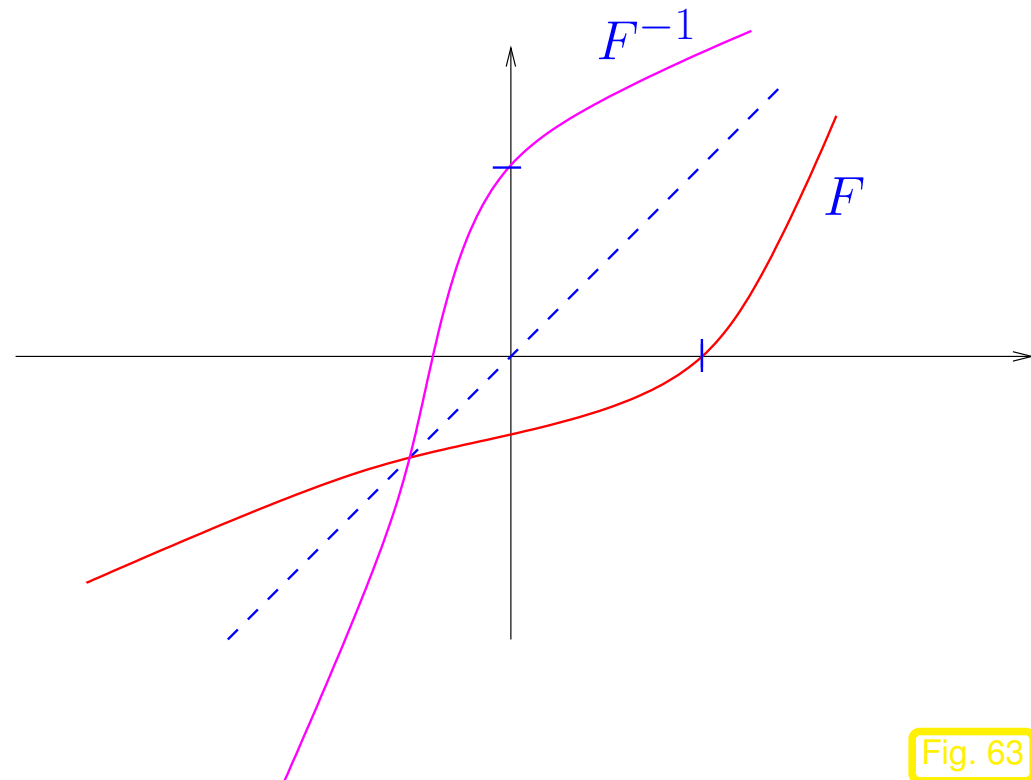


Fig. 63

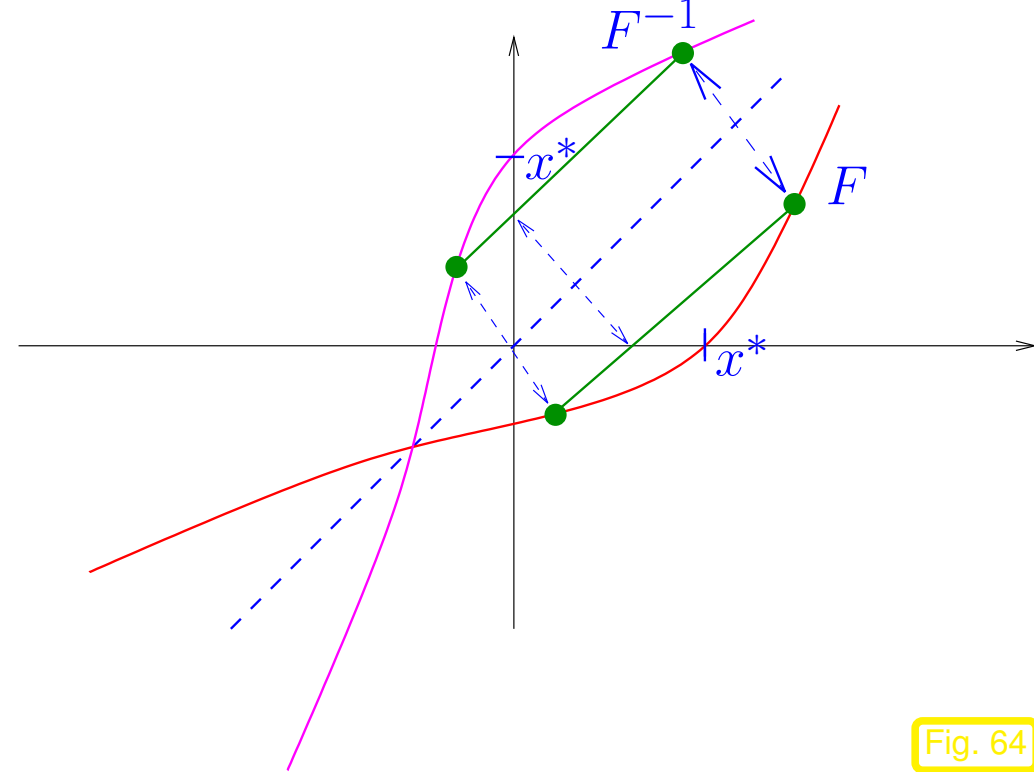


Fig. 64

Case  $d = 1$  ➤ secant method

Case  $d = 2$ : quadratic inverse interpolation, see [45, Sect. 4.5]

MAPLE code: `p := x -> a*x^2+b*x+c;`  
`solve({p(f0)=x0,p(f1)=x1,p(f2)=x2},{a,b,c});`  
`assign(%); p(0);`

$$\blacktriangleright x^{(k+1)} = \frac{F_0^2(F_1x_2 - F_2x_1) + F_1^2(F_2x_0 - F_0x_2) + F_2^2(F_0x_1 - F_1x_0)}{F_0^2(F_1 - F_2) + F_1^2(F_2 - F_0) + F_2^2(F_0 - F_1)}.$$

$$( F_0 := F(x^{(k-2)}), F_1 := F(x^{(k-1)}), F_2 := F(x^{(k)}), x_0 := x^{(k-2)}, x_1 := x^{(k-1)}, x_2 := x^{(k)} )$$

*Example 4.3.19* (quadratic inverse interpolation).  $F(x) = xe^x - 1$ ,  $x^{(0)} = 0$ ,  $x^{(1)} = 2.5$ ,  $x^{(2)} = 5$ .

$k$	$x^{(k)}$	$F(x^{(k)})$	$e^{(k)} := x^{(k)} - x^*$	$\frac{\log  e^{(k+1)}  - \log  e^{(k)} }{\log  e^{(k)}  - \log  e^{(k-1)} }$
3	0.08520390058175	-0.90721814294134	-0.48193938982803	
4	0.16009252622586	-0.81211229637354	-0.40705076418392	3.33791154378839
5	0.79879381816390	0.77560534067946	0.23165052775411	2.28740488912208
6	0.63094636752843	0.18579323999999	0.06380307711864	1.82494667289715
7	0.56107750991028	-0.01667806436181	-0.00606578049951	1.87323264214217
8	0.56706941033107	-0.00020413476766	-0.00007388007872	1.79832936980454
9	0.56714331707092	0.00000007367067	0.00000002666114	1.84841261527097
10	0.56714329040980	0.000000000000003	0.000000000000001	

Also in this case the numerical experiment hints at a fractional rate of convergence, as in the case of the secant method, see Rem. 4.3.15.



Efficiency of an iterative method  
(for solving  $F(\mathbf{x}) = 0$ )  $\leftrightarrow$  computational effort to reach prescribed  
number of significant digits in result.

Abstract:

$W \hat{=}$  computational effort per step

(e.g,  $W \approx \frac{\#\{\text{evaluations of } F\}}{\text{step}} + n \cdot \frac{\#\{\text{evaluations of } F'\}}{\text{step}} + \dots$  )

Crucial: number of steps  $k = k(\rho)$  to achieve *relative reduction of error*

$$\|e^{(k)}\| \leq \rho \|e^{(0)}\|, \quad \rho > 0 \text{ prescribed ?} \quad (4.3.20)$$



Error recursion can be converted into expressions (4.3.21) and (4.3.22) that related the error norm  $\|e^{(k)}\|$  to  $\|e^{(0)}\|$  and lead to quantitative bounds for the number of steps to achieve (4.3.20) for iterative method of order  $p$ ,  $p \geq 1$  ( $\rightarrow$  Def. 4.1.14):

$$\exists C > 0: \quad \|e^{(k)}\| \leq C \|e^{(k-1)}\|^p \quad \forall k \geq 1 \quad (C < 1 \text{ for } p = 1).$$

Assuming  $C \|e^{(0)}\|^{p-1} < 1$  (guarantees convergence):

$$p = 1: \quad \|e^{(k)}\| \stackrel{!}{\leq} C^k \|e^{(0)}\| \quad \text{requires} \quad k \geq \frac{\log \rho}{\log C}, \quad (4.3.21)$$

$$p > 1: \quad \|e^{(k)}\| \stackrel{!}{\leq} C^{\frac{p^k-1}{p-1}} \|e^{(0)}\|^{p^k} \quad \text{requires} \quad p^k \geq 1 + \frac{\log \rho}{\log C/p-1 + \log(\|e^{(0)}\|)}$$

$$\Rightarrow \quad k \geq \log\left(1 + \frac{\log \rho}{\log L_0}\right) / \log p, \quad (4.3.22)$$

$$L_0 := C^{1/p-1} \|e^{(0)}\| < 1.$$

If  $\rho \ll 1$ , then  $\log\left(1 + \frac{\log \rho}{\log L_0}\right) \approx \log |\log \rho| - \log |\log L_0| \approx \log |\log \rho|$ . This simplification will be made in the context of asymptotic considerations  $\rho \rightarrow 0$  below.

Notice:

 $|\log \rho| \leftrightarrow$  No. of significant digits of  $x^{(k)}$ 

Measure for efficiency:

$$\text{Efficiency} := \frac{\text{no. of digits gained}}{\text{total work required}} = \frac{|\log \rho|}{k(\rho) \cdot W}$$

(4.3.23)

► **asymptotic** efficiency w.r.t.  $\rho \rightarrow 0 \rightarrow |\log \rho| \rightarrow \infty$ ):

$$\text{Efficiency}_{|\rho \rightarrow 0} = \begin{cases} \frac{\log C}{W} & , \text{ if } p = 1 , \\ \frac{\log p \cdot |\log \rho|}{W \log(|\log \rho|)} & , \text{ if } p > 1 . \end{cases} \quad (4.3.24)$$

*Example 4.3.25* (Efficiency of iterative methods).

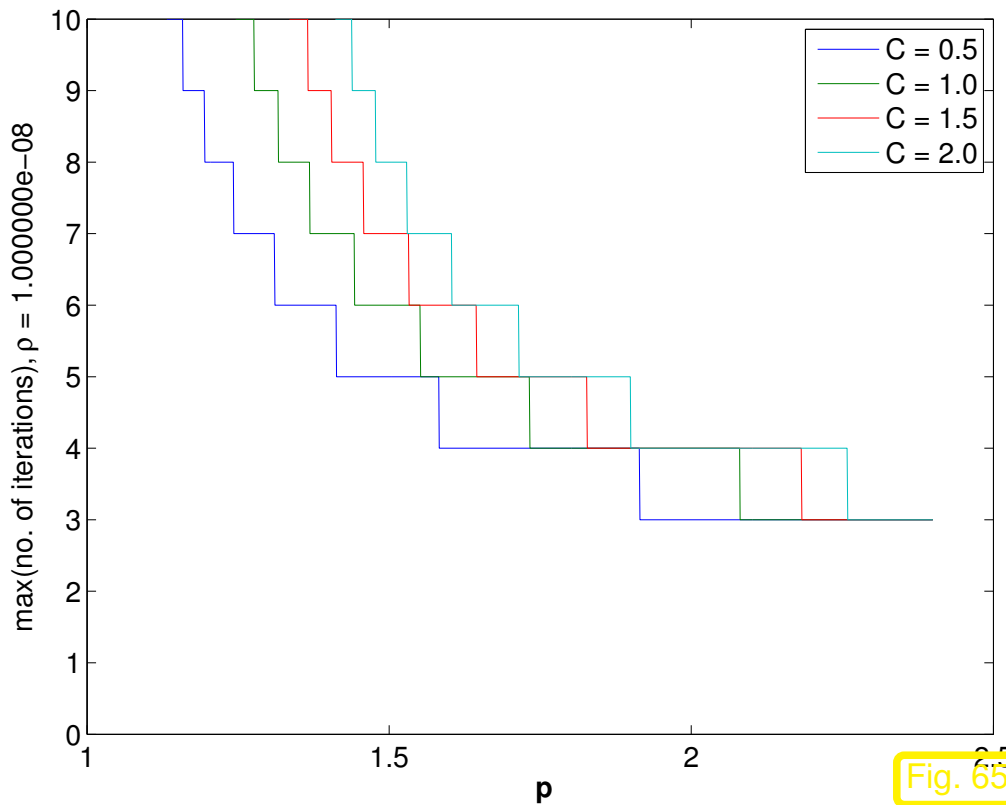


Fig. 65

Evaluation (4.3.22) for  $\|e^{(0)}\| = 0.1, \rho = 10^{-8}$

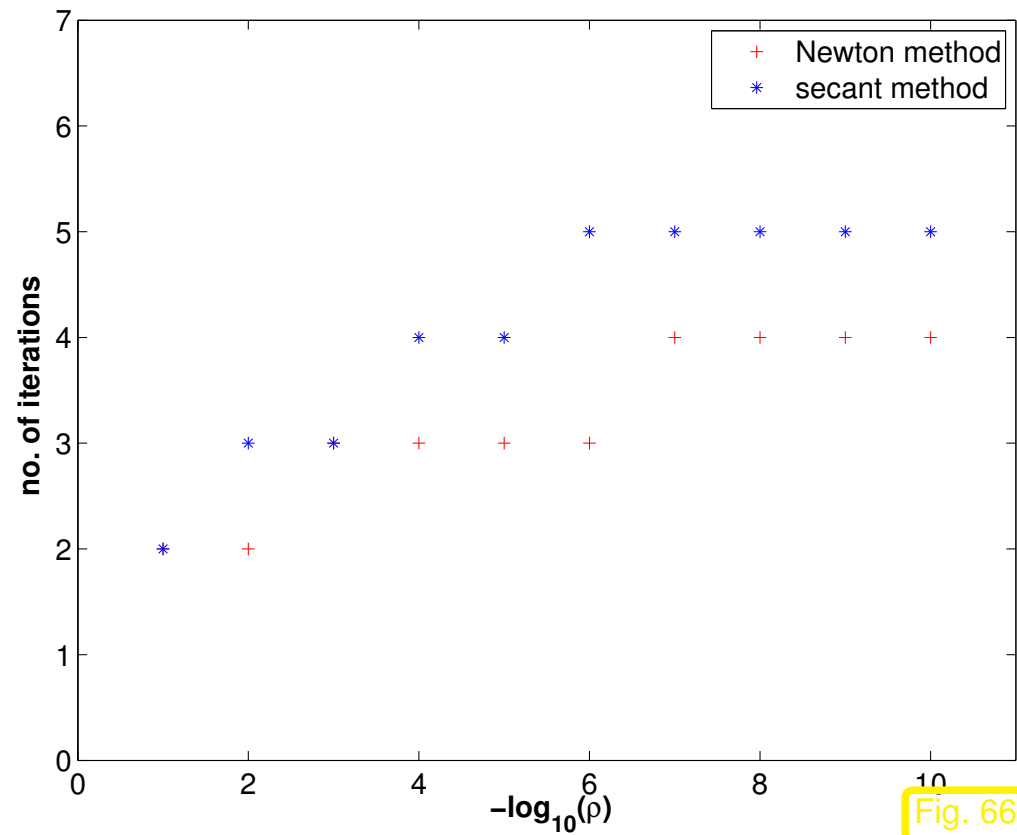


Fig. 66

Newton's method  $\leftrightarrow$  secant method,  $C = 1$ ,  
initial error  $\|e^{(0)}\| = 0.1$

$$W_{\text{Newton}} = 2W_{\text{secant}}, \quad p_{\text{Newton}} = 2, \quad p_{\text{secant}} = 1.62 \quad \Rightarrow \quad \frac{\log p_{\text{Newton}}}{W_{\text{Newton}}} : \frac{\log p_{\text{secant}}}{W_{\text{secant}}} = 0.71 .$$

secant method is more efficient than Newton's method!

# 4.4 Newton's Method [35, Sect. 19], [13, Sect. 5.6]

Non-linear system of equations: for  $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$  find  $\mathbf{x}^* \in D$ :  $F(\mathbf{x}^*) = 0$

Assume:  $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$  continuously differentiable

## 4.4.1 The Newton iteration

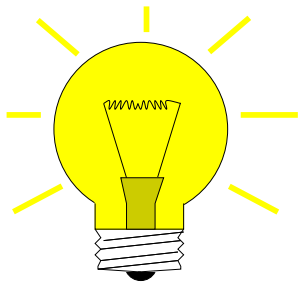
Idea ( $\rightarrow$  Sect. 4.3.2.1):

local linearization:

Given  $\mathbf{x}^{(k)} \in D \succ \mathbf{x}^{(k+1)}$  as zero of affine linear model function

$$F(\mathbf{x}) \approx \tilde{F}(\mathbf{x}) := F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}),$$

$$DF(\mathbf{x}) \in \mathbb{R}^{n,n} = \text{Jacobian (ger.: Jacobi-Matrix)}, DF(\mathbf{x}) = \left( \frac{\partial F_j}{\partial x_k}(\mathbf{x}) \right)_{j,k=1}^n.$$



**Newton iteration:** ( $\leftrightarrow$  (4.3.3) for  $n = 1$ )

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}), \quad [\text{if } DF(\mathbf{x}^{(k)}) \text{ regular}] \quad (4.4.1)$$

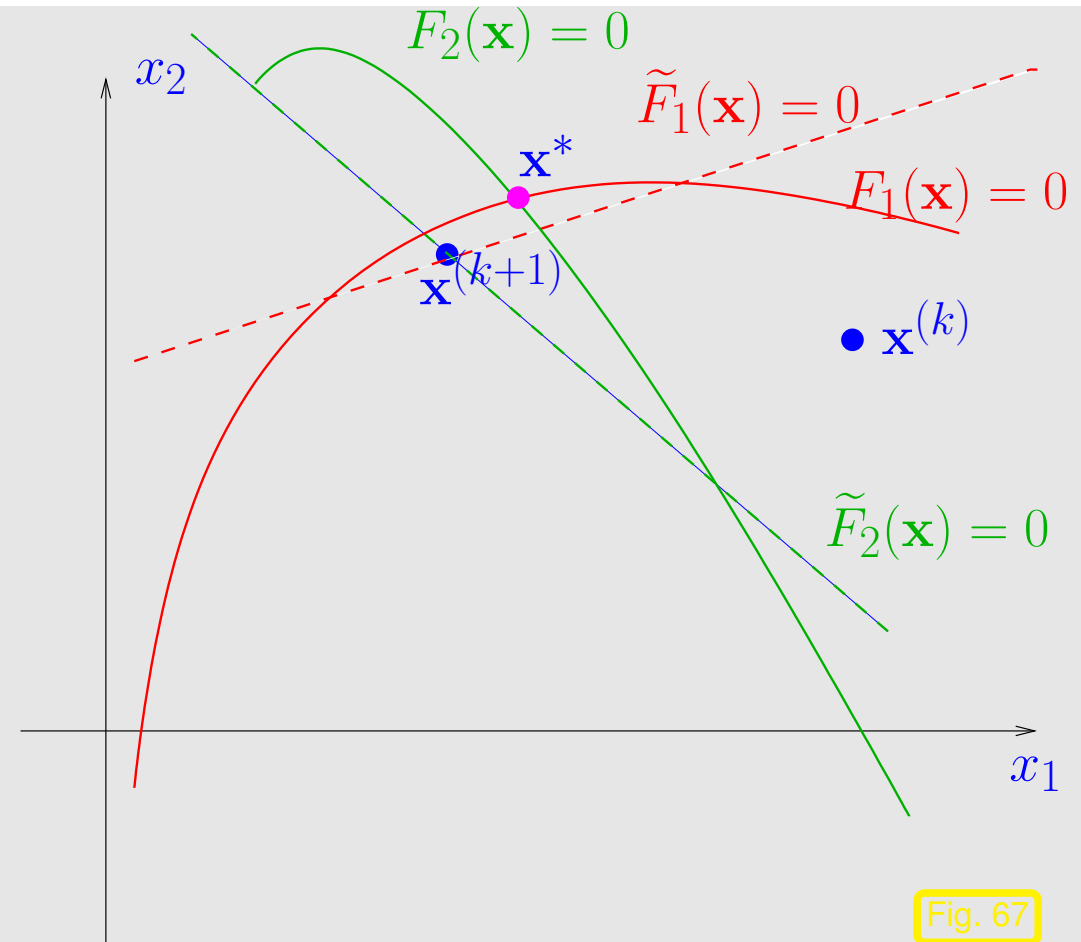
Terminology:  $-DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}) =$  Newton correction

Illustration of idea of Newton's method for  $n = 2$ :

▷

Sought: intersection point  $\mathbf{x}^*$  of the curves  
 $F_1(\mathbf{x}) = 0$  and  $F_2(\mathbf{x}) = 0$ .

Idea:  $\mathbf{x}^{(k+1)}$  = the intersection of two straight lines (= zero sets of the components of the model function, *cf.* Ex. 2.5.32) that are approximations of the original curves



MATLAB template for Newton  
method:

Solve linear system:

$A \setminus b = A^{-1}b \rightarrow$  Chapter 2

$F, DF$ : function handles

A posteriori termination criterion

MATLAB-CODE: Newton's method

```
function x = newton(x, F, DF, tr, ta)
for i=1:MAXIT
    s = DF(x) \ F(x);
    x = x-s;
    if ((norm(s) < tr*norm(x)) || (norm(s) < ta))
        return; end;
end
```

Here a **correction based** a posteriori termination criterion for the Newton iteration is used; it stops the iteration if the *relative* size of the Newton correction drops below the prescribed **relative tolerance**  $t_r$ . If  $\mathbf{x}^* \approx 0$  also the *absolute* size of the Newton correction has to be tested against an **absolute tolerance**  $t_a$  in order to avoid non-termination despite convergence of the iteration.



New aspect for  $n \gg 1$  (compared to  $n = 1$ -dimensional case, section. 4.3.2.1):

Computation of the Newton correction may be expensive!

*Remark 4.4.2* (Affine invariance of Newton method).

An important property of the Newton iteration (4.4.1): **affine invariance**  $\rightarrow$  [15, Sect .1.2.2]

set  $G(\mathbf{x}) := \mathbf{A}F(\mathbf{x})$  with regular  $\mathbf{A} \in \mathbb{R}^{n,n}$  so that  $F(\mathbf{x}^*) = 0 \Leftrightarrow G(\mathbf{x}^*) = 0$ .

**Affine invariance:** Newton iteration for  $G(\mathbf{x}) = 0$  is the same for all regular  $\mathbf{A}$  !

This is a simple computation:

$$DG(\mathbf{x}) = \mathbf{A}DF(\mathbf{x}) \Rightarrow DG(\mathbf{x})^{-1}G(\mathbf{x}) = DF(\mathbf{x})^{-1}\mathbf{A}^{-1}\mathbf{A}F(\mathbf{x}) = DF(\mathbf{x})^{-1}F(\mathbf{x}).$$

Use affine invariance as guideline for

- convergence theory for Newton's method: assumptions and results should be affine invariant, too.
- modifying and extending Newton's method: resulting schemes should preserve affine invariance.



*Remark 4.4.3* (Differentiation rules). → Repetition: basic analysis

Statement of the Newton iteration (4.4.1) for  $F : \mathbb{R}^n \mapsto \mathbb{R}^n$  given as analytic expression entails computing the Jacobian  $DF$ . To avoid cumbersome component-oriented considerations, it is useful to know the *rules of multidimensional differentiation*:

Let  $V, W$  be finite dimensional vector spaces,  $F : D \subset V \mapsto W$  sufficiently smooth. The **differential**  $DF(\mathbf{x})$  of  $F$  in  $\mathbf{x} \in V$  is the *unique*

$$\text{linear mapping } DF(\mathbf{x}) : V \mapsto W ,$$

such that  $\|F(\mathbf{x} + \mathbf{h}) - F(\mathbf{x}) - DF(\mathbf{x})\mathbf{h}\| = o(\|\mathbf{h}\|) \quad \forall \mathbf{h}, \|\mathbf{h}\| < \delta .$

- For  $F : V \mapsto W$  linear, i.e.  $F(\mathbf{x}) = \mathbf{A}\mathbf{x}$ ,  $\mathbf{A}$  matrix →  $DF(\mathbf{x}) = \mathbf{A}$ .
- **Chain rule**:  $F : V \mapsto W, G : W \mapsto U$  sufficiently smooth

$$D(G \circ F)(\mathbf{x})\mathbf{h} = DG(F(\mathbf{x}))(DF(\mathbf{x}))\mathbf{h}, \quad \mathbf{h} \in V, \mathbf{x} \in D. \quad (4.4.4)$$



- **Product rule:**  $F : D \subset V \mapsto W$ ,  $G : D \subset V \mapsto U$  sufficiently smooth,  $b : W \times U \mapsto Z$  **bilinear**

$$T(\mathbf{x}) = b(F(\mathbf{x}), G(\mathbf{x})) \Rightarrow DT(\mathbf{x})\mathbf{h} = b(DF(\mathbf{x})\mathbf{h}, G(\mathbf{x})) + b(F(\mathbf{x}), DG(\mathbf{x})\mathbf{h}), \quad (4.4.5)$$

$$\mathbf{h} \in V, \mathbf{x} \in D.$$

For  $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}$  the **gradient**  $\text{grad } F : D \mapsto \mathbb{R}^n$ , and the **Hessian matrix**  $HF(\mathbf{x}) : D \mapsto \mathbb{R}^{n,n}$  are defined as

$$\text{grad } F(\mathbf{x})^T \mathbf{h} := DF(\mathbf{x})\mathbf{h}, \quad \mathbf{h}_1^T HF(\mathbf{x})\mathbf{h}_2 := D(DF(\mathbf{x})(\mathbf{h}_1))(\mathbf{h}_2), \quad \mathbf{h}, \mathbf{h}_1, \mathbf{h}_2 \in V.$$

A simple example:

$$\Psi : \mathbb{R}^n \mapsto \mathbb{R}, \Psi(\mathbf{x}) := \mathbf{x}^T \mathbf{A} \mathbf{x}, \quad \text{with } \mathbf{A} \in \mathbb{R}^{n,n}$$

We apply the **product rule** (4.4.5) with  $F, G = Id$ , which means  $DF(\mathbf{x}) = DG(\mathbf{x}) = \mathbf{I}$ , and the **bilinear form**  $b(\mathbf{x}, \mathbf{y}) := \mathbf{x}^T \mathbf{A} \mathbf{y}$ :

$$\blacktriangleright D\Psi(\mathbf{x})\mathbf{h} = \mathbf{h}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{A} \mathbf{h} = \underbrace{(\mathbf{x}^T \mathbf{A}^T + \mathbf{x}^T \mathbf{A})}_{=(\text{grad } \Psi(\mathbf{x}))^T} \mathbf{h},$$

also  $\text{grad } \Psi(\mathbf{x}) = (\mathbf{A} + \mathbf{A}^T)\mathbf{x}$  according to the definition of a gradient.

Advice: If you do not feel comfortable with these rules of multidimensional differential calculus, please resort to detailed componentwise/entrywise calculations, though they may be tedious.



Remark 4.4.6 (Simplified Newton method). [13, Sect. 5.6.2]

### Code 4.4.7: simplified Newton method

Simplified Newton Method:

use the same  $DF(\mathbf{x}^{(k)})$  for  
all/several steps

```

1 function x = simpnewton(x,F,DF,rtol,atol)
2 % MATLAB template for simplified Newton method
3 [L,U] = lu(DF(x)); % one LU-decomposition
4 s = U \ (L \ F(x)); x = x-s;
5 % termination based on relative and absolute
   tolerance
6 ns = norm(s); nx = norm(x);
7 while ((ns > rtol*nx) && (ns > atol))
8     s = U \ (L \ F(x)); x = x-s;
9 end

```

Note: reuse of LU-decomposition,  
cf. Rem. 2.2.13

➤ (usually) merely linear convergence instead of quadratic convergence



Remark 4.4.8 (Numerical Differentiation for computation of Jacobian).

If  $DF(\mathbf{x})$  is not available (e.g. when  $F(\mathbf{x})$  is given only as a procedure):

**Numerical Differentiation:**

$$\frac{\partial F_i}{\partial x_j}(\mathbf{x}) \approx \frac{F_i(\mathbf{x} + h\vec{e}_j) - F_i(\mathbf{x})}{h}.$$

## 4.4.2 Convergence of Newton's method

Newton iteration (4.4.1)  $\hat{=}$  fixed point iteration ( $\rightarrow$  Sect. 4.2) with

$$\Phi(\mathbf{x}) = \mathbf{x} - DF(\mathbf{x})^{-1}F(\mathbf{x}) .$$

$$\text{[“product rule” : } D\Phi(\mathbf{x}) = \mathbf{I} - D(DF(\mathbf{x})^{-1})F(\mathbf{x}) - DF(\mathbf{x})^{-1}DF(\mathbf{x}) \text{ ]}$$

$$F(\mathbf{x}^*) = 0 \Rightarrow D\Phi(\mathbf{x}^*) = 0 .$$

Lemma 4.2.16 suggests conjecture:

*Local* quadratic convergence of Newton's method, if  $DF(\mathbf{x}^*)$  regular

Example 4.4.9 (Convergence of Newton's method in 2D).

$$F(\mathbf{x}) = \begin{pmatrix} x_1^2 - x_2^4 \\ x_1 - x_2^3 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2 \quad \text{with solution} \quad F \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0.$$

Jacobian (analytic computation):

$$DF(\mathbf{x}) = \begin{pmatrix} \partial_{x_1} F_1(x) & \partial_{x_2} F_1(x) \\ \partial_{x_1} F_2(x) & \partial_{x_2} F_2(x) \end{pmatrix} = \begin{pmatrix} 2x_1 & -4x_2^3 \\ 1 & -3x_2^2 \end{pmatrix}$$

## Code 4.4.10: Newton iteration in 2D

Realization of Newton iteration (4.4.1):

```

1 F=@(x) [x(1)^2-x(2)^4; x(1)-x(2)^3];
2 DF=@(x) [2*x(1), -4*x(2)^3; 1, -3*x(2)^2];
3 x=[0.7;0.7]; x_ast=[1;1]; tol=1E-10;
4
5 res=[0, x', norm(x-x_ast)];
6 s = DF(x)\F(x); x = x-s;
7 res = [res; 1, x', norm(x-x_ast)]; k=2;
8 while (norm(s) > tol*norm(x))
9     s = DF(x)\F(x); x = x-s;
10    res = [res; k, x', norm(x-x_ast)];
11    k = k+1;
12 end
13
14 ld = diff(log(res(:,4))); %
15 rates = ld(2:end)./ld(1:end-1); %

```

1. Solve LSE

$$\begin{pmatrix} 2x_1 & -4x_2^3 \\ 1 & -3x_2^2 \end{pmatrix} \Delta \mathbf{x}^{(k)} = - \begin{pmatrix} x_1^2 - x_2^4 \\ x_1 - x_2^3 \end{pmatrix},$$

where  $\mathbf{x}^{(k)} = (x_1, x_2)^T$ .

2. Set  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta \mathbf{x}^{(k)}$

Lines 14, 15: estimation of order of convergence, see Rem. 4.1.15.<sup>13</sup>

$k$	$\mathbf{x}^{(k)}$	$\epsilon_k := \ \mathbf{x}^* - \mathbf{x}^{(k)}\ _2$	$\frac{\log \epsilon_{k+1} - \log \epsilon_k}{\log \epsilon_k - \log \epsilon_{k-1}}$
0	$(0.7, 0.7)^T$	4.24e-01	
1	$(0.878500000000000, 1.064285714285714)^T$	1.37e-01	1.69
2	$(1.01815943274188, 1.00914882463936)^T$	2.03e-02	2.23
3	$(1.00023355916300, 1.00015913936075)^T$	2.83e-04	2.15
4	$(1.00000000583852, 1.00000002726552)^T$	2.79e-08	1.77
5	$(0.9999999999999998, 1.0000000000000000)^T$	2.11e-15	
6	$(1, 1)^T$		

☞ (Some) evidence of **quadratic convergence**, see Rem. 4.1.15.




There is a sophisticated theory about the convergence of Newton's method. For example one can find the following theorem in [17, Thm. 4.10], [15, Sect. 2.1]):

**Theorem 4.4.11** (Local quadratic convergence of Newton's method). **If:**

- (A)  $D \subset \mathbb{R}^n$  open and convex,
- (B)  $F : D \mapsto \mathbb{R}^n$  continuously differentiable,
- (C)  $DF(\mathbf{x})$  regular  $\forall \mathbf{x} \in D$ ,
- (D)  $\exists L \geq 0$ :  $\left\| DF(\mathbf{x})^{-1}(DF(\mathbf{x} + \mathbf{v}) - DF(\mathbf{x})) \right\|_2 \leq L \|\mathbf{v}\|_2 \quad \forall \mathbf{v} \in \mathbb{R}^n, \mathbf{v} + \mathbf{x} \in D, \forall \mathbf{x} \in D$ ,
- (E)  $\exists \mathbf{x}^*$ :  $F(\mathbf{x}^*) = 0$  (*existence of solution in  $D$* )
- (F) initial guess  $\mathbf{x}^{(0)} \in D$  satisfies  $\rho := \left\| \mathbf{x}^* - \mathbf{x}^{(0)} \right\|_2 < \frac{2}{L} \wedge B_\rho(\mathbf{x}^*) \subset D$ .

**then the Newton iteration (4.4.1) satisfies:**

- (i)  $\mathbf{x}^{(k)} \in B_\rho(\mathbf{x}^*) := \{\mathbf{y} \in \mathbb{R}^n, \|\mathbf{y} - \mathbf{x}^*\| < \rho\}$  for all  $k \in \mathbb{N}$ ,
- (ii)  $\lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x}^*$ ,
- (iii)  $\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\|_2 \leq \frac{L}{2} \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\|_2^2$  (*local quadratic convergence*).

 notation: ball  $B_\rho(\mathbf{z}) := \{\mathbf{x} \in \mathbb{R}^n: \|\mathbf{x} - \mathbf{z}\|_2 \leq \rho\}$

Terminology: (D)  $\hat{=}$  affine invariant **Lipschitz condition**

Problem: Usually neither  $\omega$  nor  $x^*$  are known !

► In general: a priori estimates as in Thm. 4.4.11 are of little practical relevance.

### 4.4.3 Termination of Newton iteration

A first viable idea:

**Asymptotically** due to quadratic convergence:

$$\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\| \ll \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| \Rightarrow \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\| \approx \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\|. \quad (4.4.12)$$



➤ quit iterating as soon as  $\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\| = \left\| DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}) \right\| < \tau \left\| \mathbf{x}^{(k)} \right\|$ ,  
with  $\tau = \text{tolerance}$

→ uneconomical: one needless update, because  $\mathbf{x}^{(k)}$  already accurate enough !

*Remark 4.4.13.* New aspect for  $n \gg 1$ : computation of Newton correction may be expensive !



Therefore we would like to use an a-posteriori termination criterion that dispenses with computing (and “inverting”) another Jacobian  $DF(\mathbf{x}^{(k)})$  just to tell us that  $\mathbf{x}^{(k)}$  is already accurate enough.



Practical a-posteriori termination criterion for Newton’s method:

$$DF(\mathbf{x}^{(k-1)}) \approx DF(\mathbf{x}^{(k)}): \text{ quit as soon as } \left\| DF(\mathbf{x}^{(k-1)})^{-1} F(\mathbf{x}^{(k)}) \right\| < \tau_{\text{rel}} \left\| \mathbf{x}^{(k)} \right\|$$

affine invariant termination criterion

Justification: we expect  $DF(\mathbf{x}^{(k-1)}) \approx DF(\mathbf{x}^{(k)})$ , when Newton iteration has converged. Then appeal to (4.4.12).

Rationale: LU-decomposition of  $DF(\mathbf{x}^{(k-1)})$  is already available ➤ less effort.

If we used the residual based termination criterion

$$\|F(\mathbf{x}^{(k)})\| \leq \tau,$$

then the resulting algorithm would not be affine invariant, because for  $F(\mathbf{x}) = 0$  and  $\mathbf{A}F(\mathbf{x}) = 0$ ,  $\mathbf{A} \in \mathbb{R}^{n,n}$  regular, the Newton iteration might terminate with different iterates.

Terminology:  $\Delta\bar{\mathbf{x}}^{(k)} := DF(\mathbf{x}^{(k-1)})^{-1}F(\mathbf{x}^{(k)}) \hat{=} \text{**simplified Newton correction**}$

Reuse of LU-factorization ( $\rightarrow$  Rem. 2.2.13) of  $DF(\mathbf{x}^{(k-1)})$  ➤  $\Delta\bar{\mathbf{x}}^{(k)}$  available with  $O(n^2)$  operations

## Summary: The Newton Method



converges *asymptotically* very fast: doubling of number of significant digits in each step



often a very small region of convergence, which requires an initial guess rather close to the solution.

### 4.4.4 Damped Newton method [13, pp. 200]

Potentially big problem: Newton method converges quadratically, but only *locally*, which may render it useless, if convergence is guaranteed only for initial guesses very close to exact solution, see also Ex. 4.3.18.

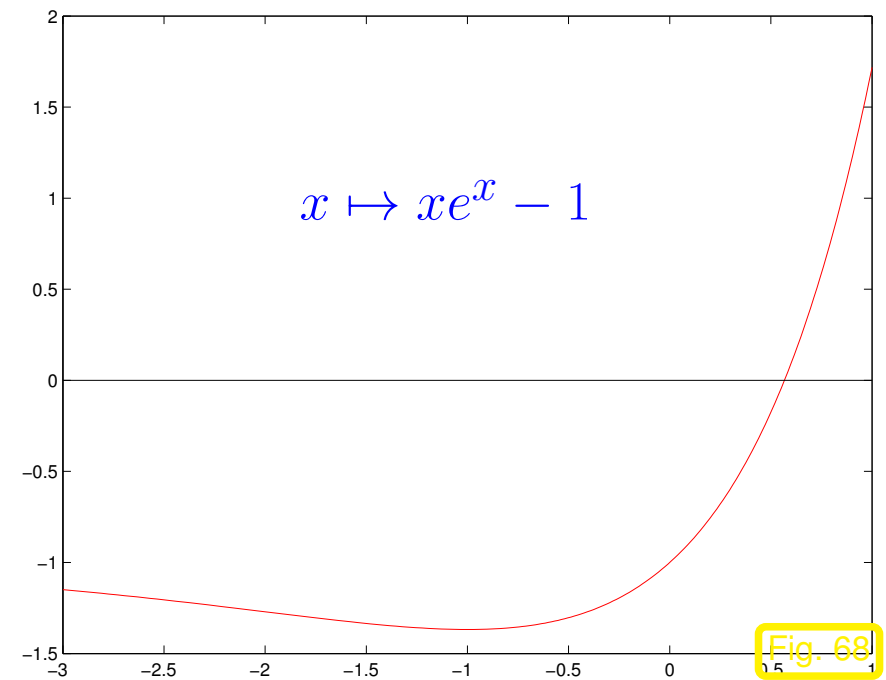
In this section we study a method to enlarge the region of convergence, at the expense of quadratic convergence, of course.

*Example 4.4.14* (Local convergence of Newton's method).

$$F(x) = xe^x - 1 \Rightarrow F'(-1) = 0$$

$$x^{(0)} < -1 \Rightarrow x^{(k)} \rightarrow -\infty$$

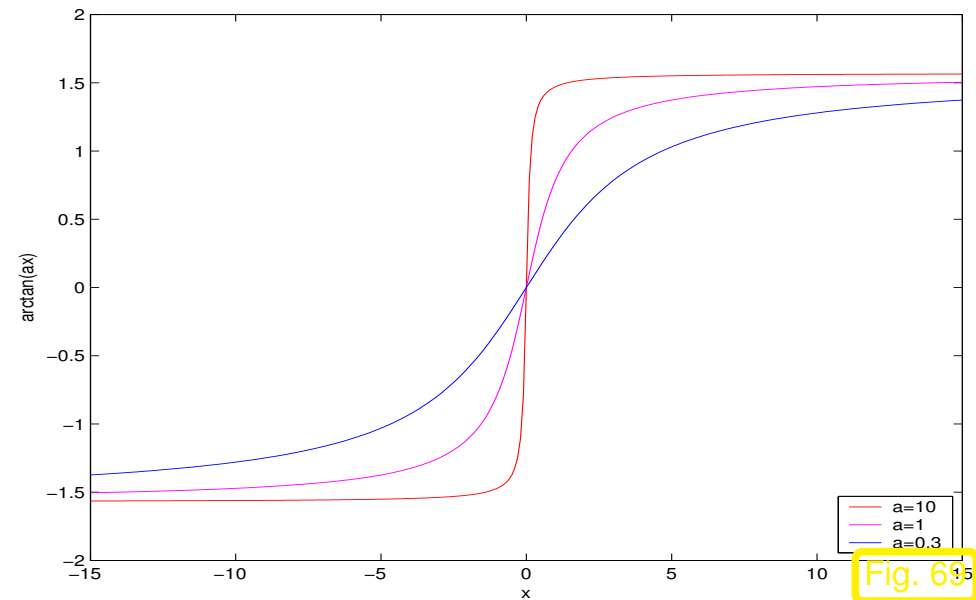
$$x^{(0)} > -1 \Rightarrow x^{(k)} \rightarrow x^*$$

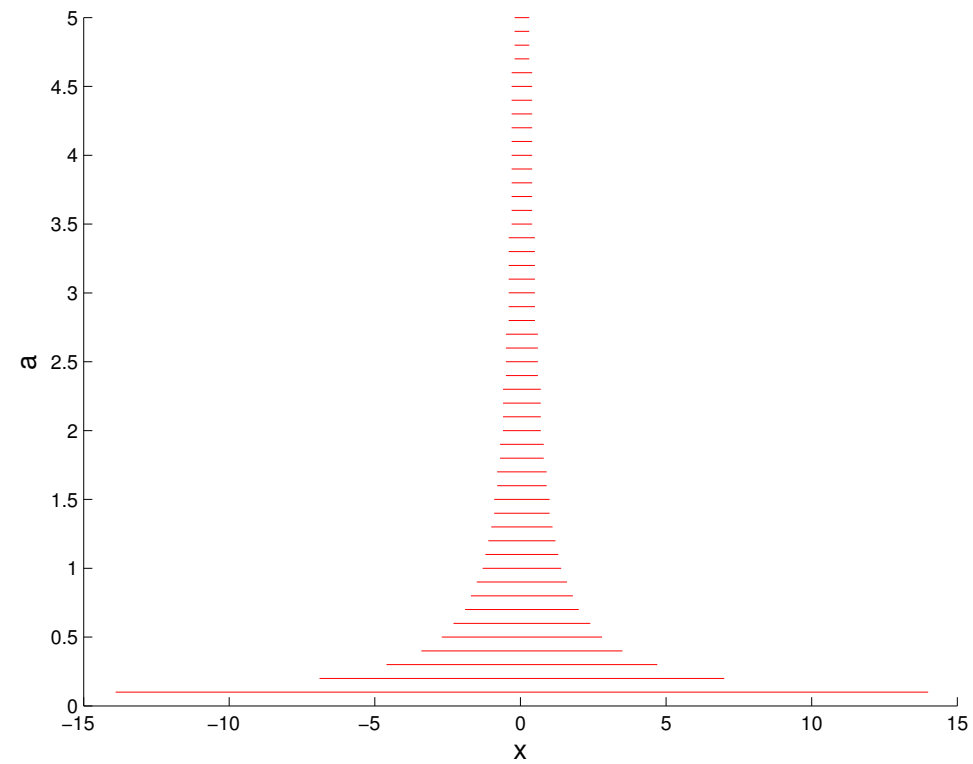
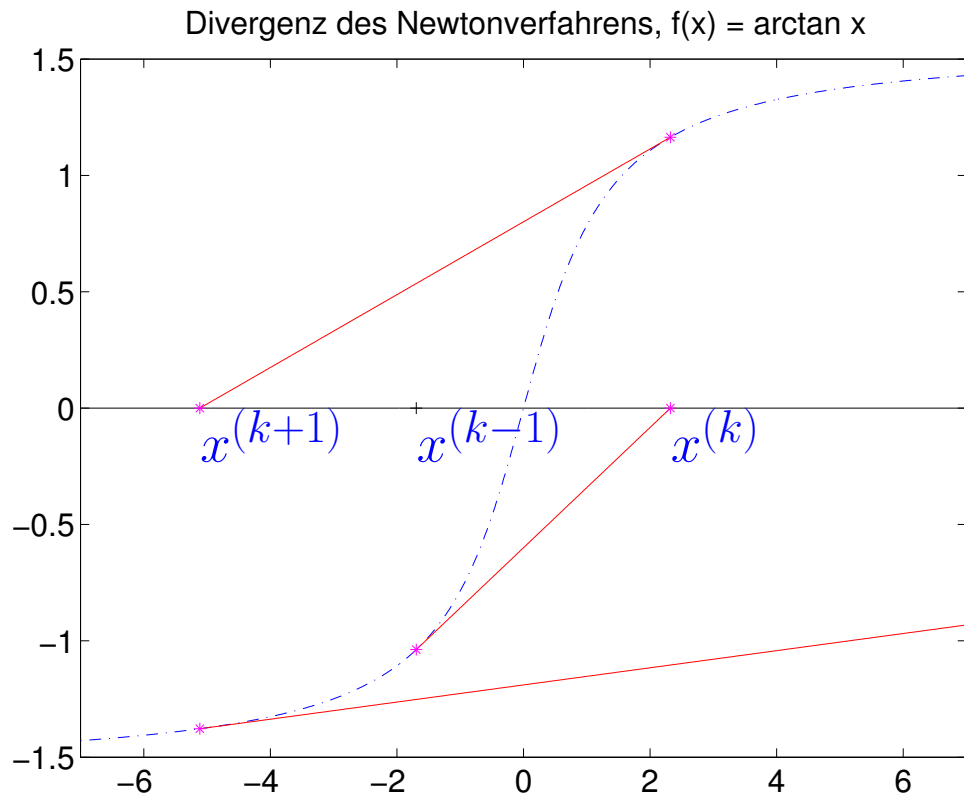


*Example 4.4.15* (Region of convergence of Newton method).

$$F(x) = \arctan(ax), \quad a > 0, x \in \mathbb{R}$$

with zero  $x^* = 0$ .





red zone =  $\{x^{(0)} \in \mathbb{R}, x^{(k)} \rightarrow 0\}$

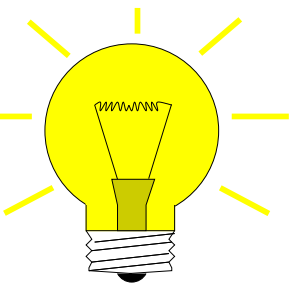
► we observe “overshooting” of Newton correction

Idea:

damping of Newton correction:

$$\text{With } \lambda^{(k)} > 0: \quad \mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \lambda^{(k)} DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}) . \quad (4.4.16)$$

Terminology:  $\lambda^{(k)}$  = damping factor



Choice of damping factor: affine invariant **natural monotonicity test** [15, Ch. 3]

“maximal”  $0 < \lambda^{(k)} \leq 1$ : 
$$\left\| \Delta \bar{\mathbf{x}}(\lambda^{(k)}) \right\| \leq \left(1 - \frac{\lambda^{(k)}}{2}\right) \left\| \Delta \mathbf{x}^{(k)} \right\|_2 \quad (4.4.17)$$

where

$$\Delta \mathbf{x}^{(k)} := DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}) \quad \rightarrow \text{current Newton correction ,}$$

$$\Delta \bar{\mathbf{x}}(\lambda^{(k)}) := DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)} + \lambda^{(k)} \Delta \mathbf{x}^{(k)}) \quad \rightarrow \text{tentative simplified Newton correction .}$$

Heuristics behind control of damping:

- When the method converges  $\Leftrightarrow$  size of Newton correction decreases  $\Leftrightarrow$  (4.4.17) satisfied.
- In the case of strong damping ( $\lambda^{(k)} \ll 1$ ) the size of the Newton correction cannot be expected to shrink significantly, since iterates do not change much  $\succ$  factor  $(1 - \frac{1}{2}\lambda^{(k)})$  in (4.4.17).

Note: As before, reuse of LU-factorization in the computation of  $\Delta \mathbf{x}^{(k)}$  and  $\Delta \bar{\mathbf{x}}(\lambda^{(k)})$ .

## Code 4.4.18: Damped Newton method

```

1 function [x,cvg] =
   dampnewton(x,F,DF,rtol,atol)
2 [L,U] = lu(DF(x)); s = U\(L\F(x));
3 xn = x-s; lambda = 1; cvg = 0;
4 f = F(xn); st = U\(L\f); stn = norm(st);
5 while ((stn>rtol*norm(xn)) && (stn > atol))
6   while (norm(st) > (1-lambda/2)*norm(s))
7     lambda = lambda/2;
8     if (lambda < LMIN), cvg = -1; return;
9     end
10    xn = x-lambda*s; f = F(xn);
11    st = U\(L\f);
12 end
13 x = xn; [L,U] = lu(DF(x)); s = U\(L\f);
14 lambda = min(2*lambda,1);
15 xn = x-lambda*s; f = F(xn); st =
   U\(L\f);
16 end
17 x = xn;

```

Reuse of LU-factorization, see  
Rem. 2.2.13

a-posteriori termination  
criterion (based on  
simplified Newton correction,  
*cf.* Sect. 4.4.3)

Natural monotonicity test  
(4.4.17)

Reduce damping factor  $\lambda$

Policy: Reduce damping factor by a factor  $q \in ]0, 1[$  (usually  $q = \frac{1}{2}$ ) until the affine invariant natural monotonicity test (4.4.17) passed.

*Example 4.4.19* (Damped Newton method). ( $\rightarrow$  Ex. 4.4.15)

$$F(x) = \arctan(x),$$

- $x^{(0)} = 20$

- $q = \frac{1}{2}$

- LMIN = 0.001

Observation: asymptotic  
quadratic convergence

$k$	$\lambda^{(k)}$	$x^{(k)}$	$F(x^{(k)})$
1	0.03125	0.94199967624205	0.75554074974604
2	0.06250	0.85287592931991	0.70616132170387
3	0.12500	0.70039827977515	0.61099321623952
4	0.25000	0.47271811131169	0.44158487422833
5	0.50000	0.20258686348037	0.19988168667351
6	1.00000	-0.00549825489514	-0.00549819949059
7	1.00000	0.00000011081045	0.00000011081045
8	1.00000	-0.000000000000001	-0.000000000000001



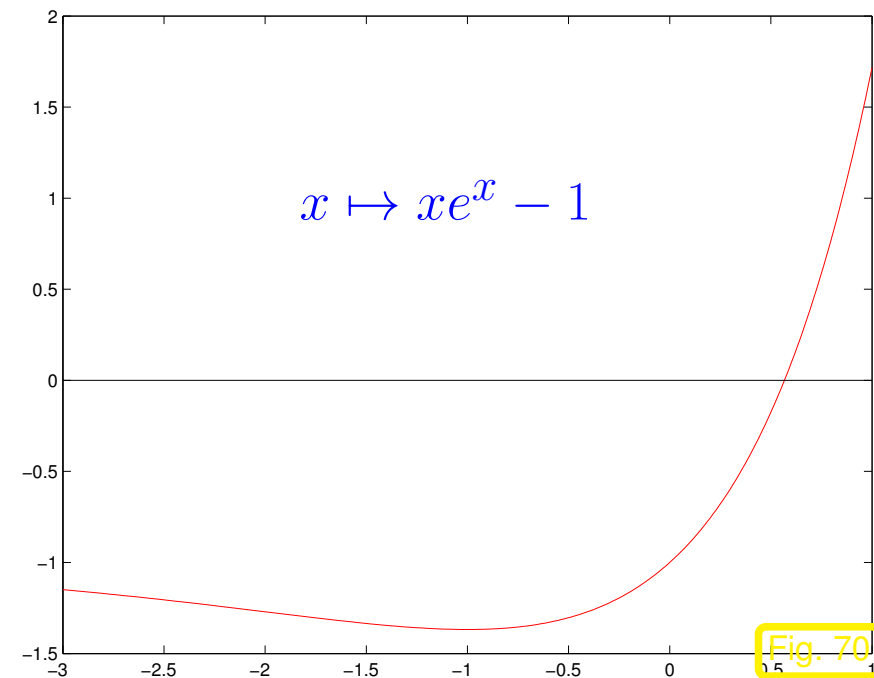
*Example 4.4.20* (Failure of damped Newton method).



- As in Ex. 4.4.14:

$$F(x) = xe^x - 1,$$

- Initial guess for damped Newton method  $x^{(0)} = -1.5$



Observation:

Newton correction pointing in “wrong direction”

➤ no convergence despite damping

$k$	$\lambda^{(k)}$	$x^{(k)}$	$F(x^{(k)})$
1	0.25000	-4.4908445351690	-1.0503476286303
2	0.06250	-6.1682249558799	-1.0129221310944
3	0.01562	-7.6300006580712	-1.0037055902301
4	0.00390	-8.8476436930246	-1.0012715832278
5	0.00195	-10.5815494437311	-1.0002685596314

Bailed out because of  $\lambda < \text{LMIN}$  !



## 4.4.5 Quasi-Newton Method [51, Sect. 7.1.4]

What to do when  $DF(\mathbf{x})$  is not available and numerical differentiation (see remark 4.4.8) is too expensive?

Idea: in one dimension ( $n = 1$ ) apply the secant method (4.3.11) of section 4.3.2.3



$$F'(x^{(k)}) \approx \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \quad \text{"difference quotient"} \quad (4.4.21)$$

already computed !  $\rightarrow$  cheap



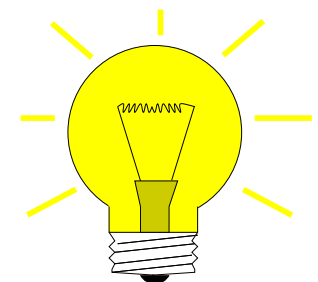
Generalisation for  $n > 1$  ?

Idea: rewrite (4.4.21) as a **secant condition** for the approximation  $\mathbf{J}_k \approx DF(\mathbf{x}^{(k)})$ ,  $\mathbf{x}^{(k)} \hat{=}$  iterate:

$$\mathbf{J}_k(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}) = F(\mathbf{x}^{(k)}) - F(\mathbf{x}^{(k-1)}) . \quad (4.4.22)$$

BUT: many matrices  $\mathbf{J}_k$  fulfill (4.4.22)

Hence: we need more conditions for  $\mathbf{J}_k \in \mathbb{R}^{n,n}$



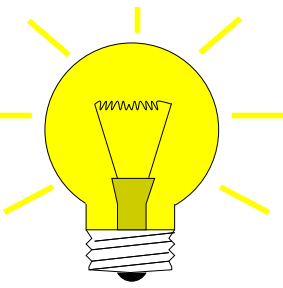
Idea: get

$\mathbf{J}_k$  by a modification of  $\mathbf{J}_{k-1}$

Broyden conditions:  $\mathbf{J}_k \mathbf{z} = \mathbf{J}_{k-1} \mathbf{z} \quad \forall \mathbf{z}: \mathbf{z} \perp (\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})$ . (4.4.23)

i.e.:

$$\mathbf{J}_k := \mathbf{J}_{k-1} + \frac{F(\mathbf{x}^{(k)})(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})^T}{\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_2^2} \quad (4.4.24)$$



Broydens Quasi-Newton Method for solving  $F(\mathbf{x}) = 0$ :

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \Delta \mathbf{x}^{(k)}, \quad \Delta \mathbf{x}^{(k)} := -\mathbf{J}_k^{-1} F(\mathbf{x}^{(k)}),$$

$$\mathbf{J}_{k+1} := \mathbf{J}_k + \frac{F(\mathbf{x}^{(k+1)})(\Delta \mathbf{x}^{(k)})^T}{\|\Delta \mathbf{x}^{(k)}\|_2^2}. \quad (4.4.25)$$

Start: initialize  $\mathbf{J}_0$  e.g. with the exact Jacobi matrix  $DF(\mathbf{x}^{(0)})$ .

*Remark 4.4.26* (Minimal property of Broydens rank 1 modification).

Let  $\mathbf{J} \in \mathbb{R}^{n,n}$  fulfill (4.4.22)  
and  $\mathbf{J}_k, \mathbf{x}^{(k)}$  from (4.4.25) then  $(\mathbf{I} - \mathbf{J}_k^{-1}\mathbf{J})(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = -\mathbf{J}_k^{-1}F(\mathbf{x}^{(k+1)})$

and hence

$$\begin{aligned} \left\| \mathbf{I} - \mathbf{J}_k^{-1}\mathbf{J}_{k+1} \right\|_2 &= \left\| \frac{-\mathbf{J}_k^{-1}F(\mathbf{x}^{(k+1)})\Delta\mathbf{x}^{(k)}}{\left\| \Delta\mathbf{x}^{(k)} \right\|_2^2} \right\|_2 = \left\| \left( \mathbf{I} - \mathbf{J}_k^{-1}\mathbf{J} \right) \frac{\Delta\mathbf{x}^{(k)}(\Delta\mathbf{x}^{(k)})^T}{\left\| \Delta\mathbf{x}^{(k)} \right\|_2^2} \right\|_2 \\ &\leq \left\| \mathbf{I} - \mathbf{J}_k^{-1}\mathbf{J} \right\|_2. \end{aligned}$$

In conclusion,

(4.4.24) gives the  $\|\cdot\|_2$ -minimal relative correction of  $\mathbf{J}_{k-1}$ , such that the secant condition (4.4.22) holds.

△

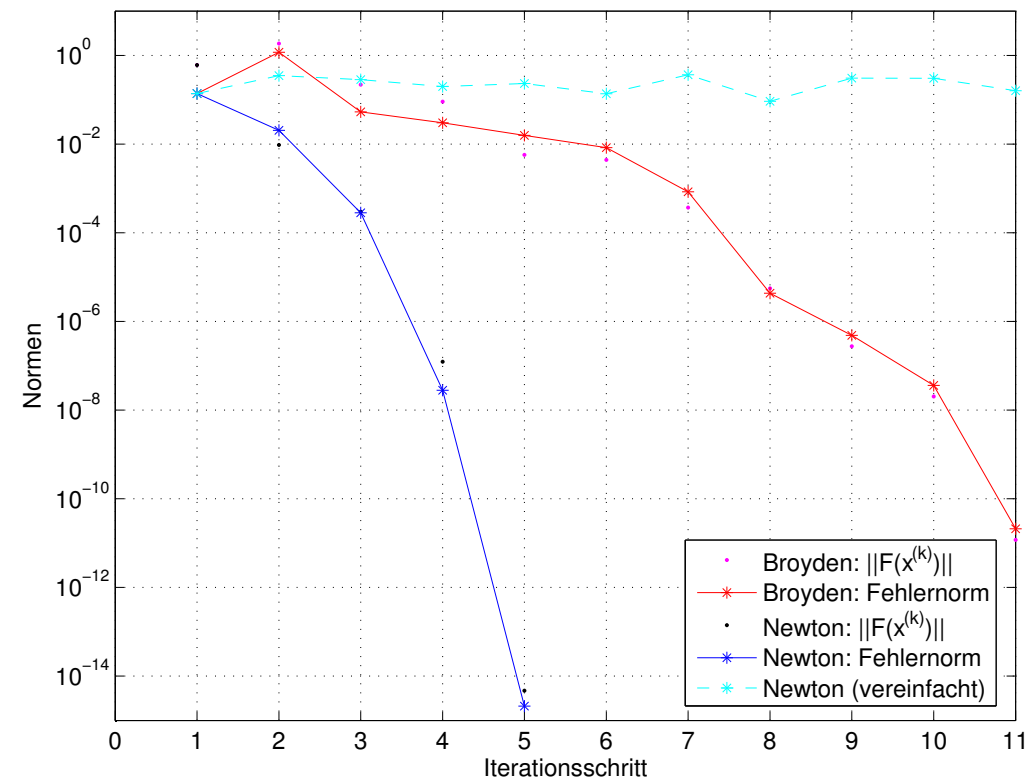
*Example 4.4.27* (Broydens Quasi-Newton Method: Convergence).

- In the non-linear system of the example 4.4.9,  $n = 2$  take  $\mathbf{x}^{(0)} = (0.7, 0.7)^T$  and  $\mathbf{J}_0 = DF(\mathbf{x}^{(0)})$

The numerical example shows that the method is:

slower than Newton method (4.4.1), but

better than simplified Newton method  
(see remark. 4.4.6)



Observation: for all iterative methods for non-linear systems of equations convergence can fail (stall/diverge).

➤ algorithms should warn user of impending failure.

## convergence monitor

=

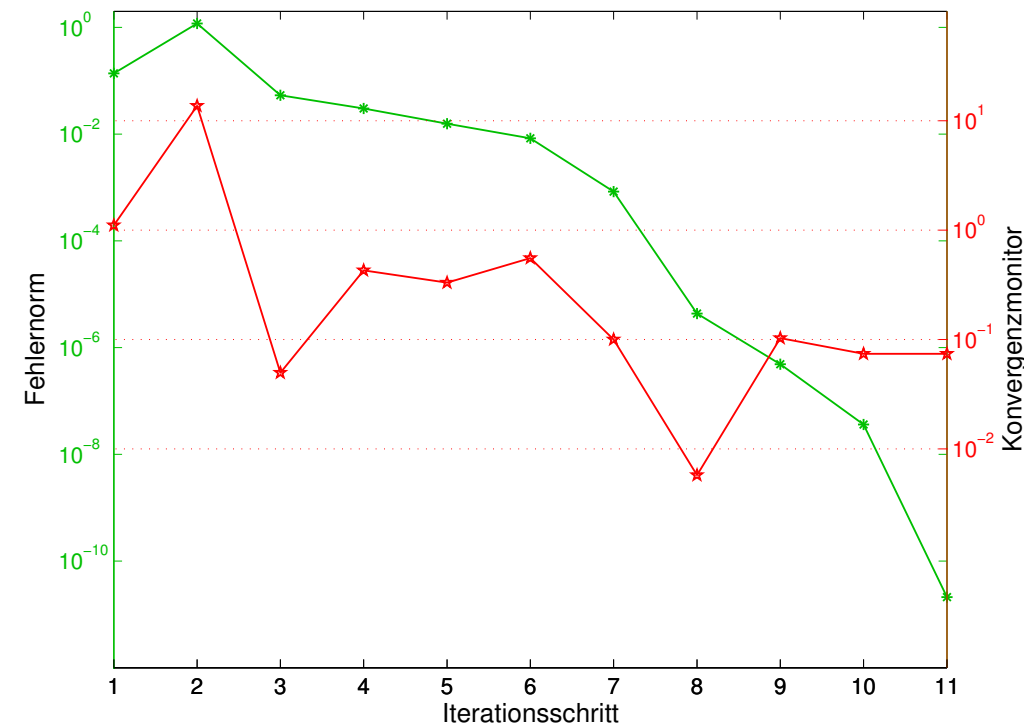
quantity that alerts to difficulties in the convergence of an iteration

Here:

$$\mu := \frac{\left\| \mathbf{J}_{k-1}^{-1} F(\mathbf{x}^{(k)}) \right\|}{\left\| \Delta \mathbf{x}^{(k-1)} \right\|}$$

Heuristics: no convergence whenever  $\mu > 1$

◇ R. Hiptmair  
rev 38286,  
October 26,  
2011



*Remark 4.4.28* (Damped Broyden method).

Option to improve robustness (increase region of local convergence):

damped

 Broyden method      (*cf.* same idea for Newton's method, section 4.4.4)

**Implementation of (4.4.25):**

Observe: (4.4.25) is a **rank-1-update** → Sect. 2.9.0.1

Idea: use Sherman-Morrison-Woodbury update-formula, Lemma 2.9.7

$$\mathbf{J}_{k+1}^{-1} = \left( \mathbf{I} - \frac{\mathbf{J}_k^{-1} F(\mathbf{x}^{(k+1)}) (\Delta \mathbf{x}^{(k)})^T}{\|\Delta \mathbf{x}^{(k)}\|_2^2 + \Delta \mathbf{x}^{(k)} \cdot \mathbf{J}_k^{-1} F(\mathbf{x}^{(k+1)})} \right) \mathbf{J}_k^{-1} = \left( \mathbf{I} + \frac{\Delta \mathbf{x}^{(k+1)} (\Delta \mathbf{x}^{(k)})^T}{\|\Delta \mathbf{x}^{(k)}\|_2^2} \right) \mathbf{J}_k^{-1} \tag{4.4.29}$$

that is well defined, if

$$\left\| \mathbf{J}_k^{-1} F(\mathbf{x}^{(k+1)}) \right\|_2 < \left\| \Delta \mathbf{x}^{(k)} \right\|_2$$

"simplified Quasi-Newton correction"

Iterated application of (4.4.29) pays off, if iteration terminates after a few steps.

```
function x = broyden(F,x,J,tol)
```

```
k = 1;
```

```
[L,U] = lu(J);
```

```
s = U\(L\F(x)); sn = dot(s,s);
```

```
dx = [s]; dxn = [sn];
```

```
x = x - s; f = F(x);
```

```
while (sqrt(sn) > tol), k=k+1
```

```
    w = U\(L\f);
```

```
    for l=2:k-1
```

```
        w = w+dx(:,l)*(dx(:,l-1)'*w) ...  
            /dxn(l-1);
```

```
    end
```

```
    if (norm(w) >= sn)
```

```
        warning('Dubious step %d!',k);
```

```
    end
```

```
    z = s'*w; s = (1+z/(sn-z))*w; sn=s'*s;
```

```
    dx = [dx,s]; dxn = [dxn,sn];
```

```
    x = x - s; f = F(x);
```

```
end
```

unique LU-decomposition !

store  $\Delta \mathbf{x}^{(k)}$ ,  $\|\Delta \mathbf{x}^{(k)}\|_2^2$   
(see (4.4.29))

solve two SLEs

Termination, see 4.4.2

construct  $\mathbf{w} := \mathbf{J}_k^{-1} F(\mathbf{x}^{(k)})$   
( $\rightarrow$  recursion (4.4.29))

convergence monitor

$$\frac{\|\mathbf{J}_{k-1}^{-1} F(\mathbf{x}^{(k)})\|}{\|\Delta \mathbf{x}^{(k-1)}\|} < 1 ?$$

correction  $\mathbf{s} = \mathbf{J}_k^{-1} F(\mathbf{x}^{(k)})$



- Computational cost :  $O(N^2 \cdot n)$  operations with vectors, (Level I)  
 $N$  steps
- 1 LU-decomposition of  $J$ ,  $N \times$  solutions of SLEs, see section 2.2
  - $N$  evaluations of  $F$  !

- Memory cost : LU-factors of  $J$  + auxiliary vectors  $\in \mathbb{R}^n$   
 $N$  steps
- $N$  vectors  $\mathbf{x}^{(k)} \in \mathbb{R}^n$

Example 4.4.30 (Broyden method for a large non-linear system).

$$F(\mathbf{x}) = \begin{cases} \mathbb{R}^n \mapsto \mathbb{R}^n \\ \mathbf{x} \mapsto \text{diag}(\mathbf{x})\mathbf{A}\mathbf{x} - \mathbf{b} \end{cases},$$

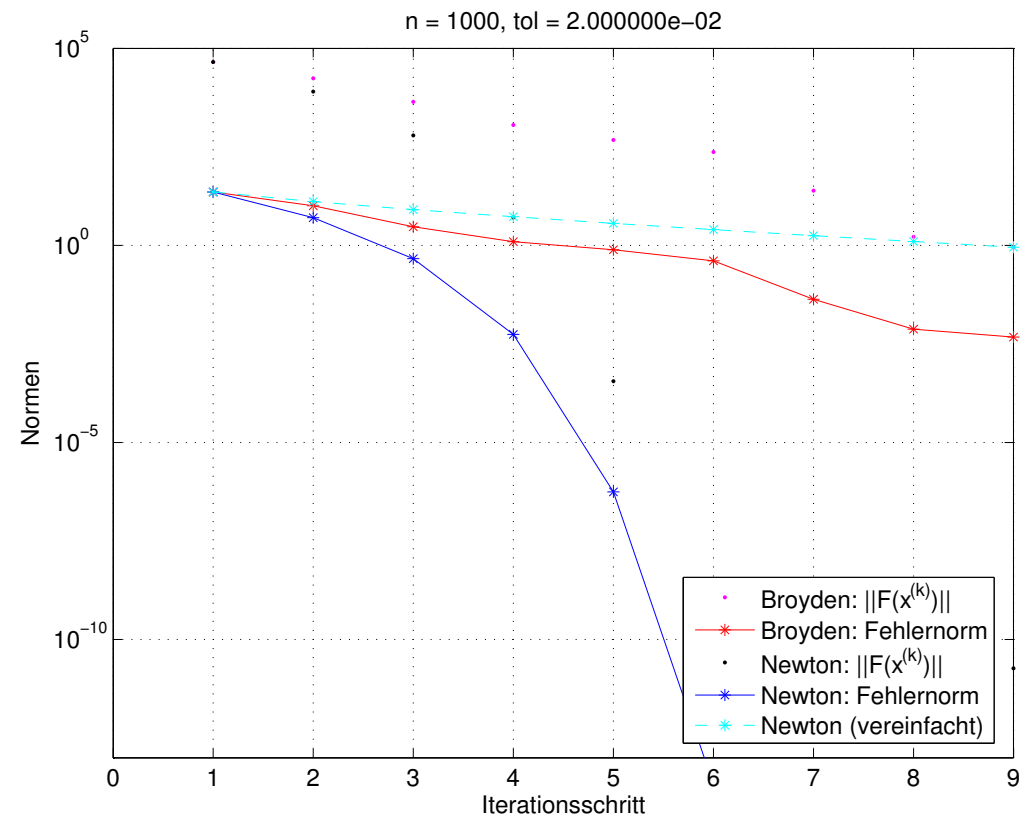
$$\mathbf{b} = (1, 2, \dots, n) \in \mathbb{R}^n,$$

$$\mathbf{A} = \mathbf{I} + \mathbf{a}\mathbf{a}^T \in \mathbb{R}^{n,n},$$

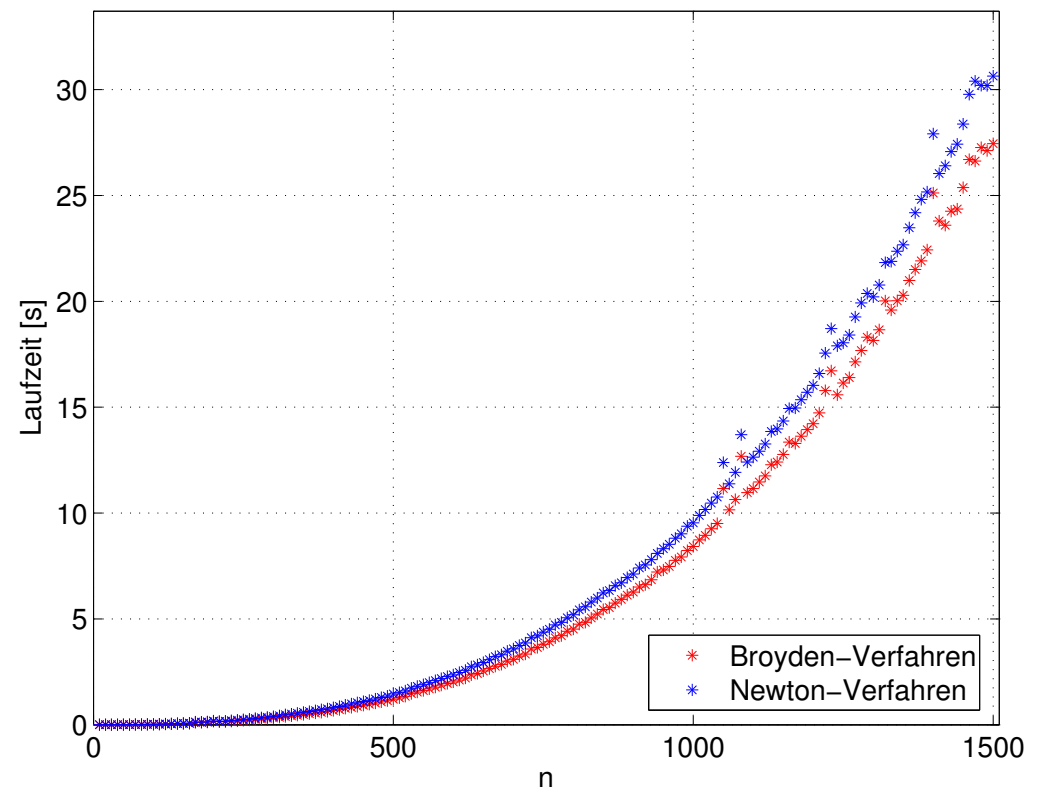
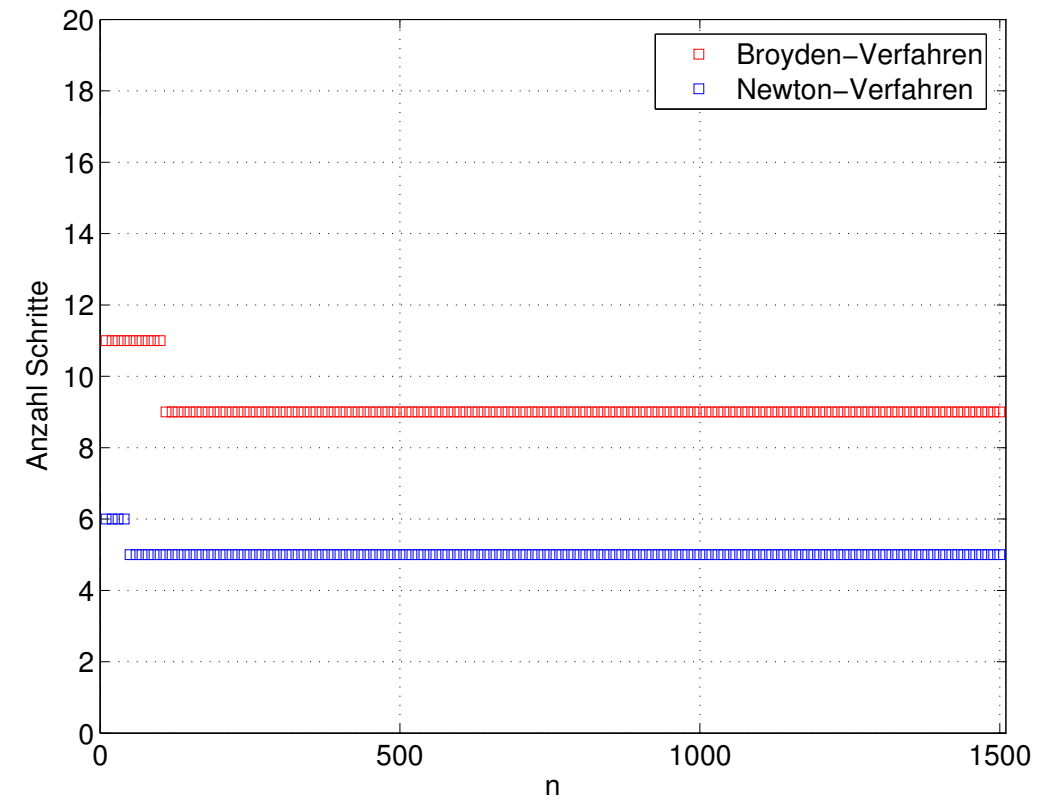
$$\mathbf{a} = \frac{1}{\sqrt{\mathbf{1} \cdot \mathbf{b} - \mathbf{1}}}(\mathbf{b} - \mathbf{1}).$$

The interpretation of the results resemble the example 4.4.27  $\triangleright$

$$h = 2/n; \quad \mathbf{x}_0 = (2:h:4-h)';$$



Efficiency comparison: Broyden method  $\longleftrightarrow$  Newton method:  
 (in case of dimension  $n$  use tolerance  $tol = 2n \cdot 10^{-5}$ ,  $h = 2/n$ ;  $x_0 = (2:h:4-h)'$  ; )



In conclusion,  
 the Broyden method is worthwhile for dimensions  $n \gg 1$  and low accuracy requirements.

# 5

## Krylov Methods for Linear Systems of Equations



### Supplementary and further reading:

There is a wealth of literature on iterative methods for the solution of linear systems of equations: The two books [29] and [55] offer a comprehensive treatment of the topic (the latter is available online for ETH students and staff).

Concise presentations can be found in [51, Ch. 4] and [13, Ch. 13].

## Learning outcomes:

- Understanding when and why iterative solution of linear systems of equations may be preferred to direct solvers based on Gaussian elimination.

= A class of **iterative methods** ( $\rightarrow$  section 4.1) for approximate solution of large linear systems of equations  $\mathbf{Ax} = \mathbf{b}$ ,  $\mathbf{A} \in \mathbb{K}^{n,n}$ .

BUT, we have reliable *direct* methods (Gauss elimination  $\rightarrow$  Sect. 2.1, LU-factorization  $\rightarrow$  Alg. 2.2.12, QR-factorization  $\rightarrow$  Alg. 2.8.22) that provide an (apart from roundoff errors) exact solution with a *finite* number of elementary operations!

Alas, direct elimination may **not** be **feasible**, or may be grossly inefficient, because

- it may be too expensive (e.g. for  $\mathbf{A}$  too large, sparse),  $\rightarrow$  (2.2.9),

- inevitable fill-in may exhaust main memory,
- the system matrix may be available only as procedure  $y = \text{evalA}(x) \leftrightarrow y = Ax$

## 5.1 Descent Methods [51, Sect. 4.3.3]

Focus: Linear system of equations  $Ax = b$ ,  $A \in \mathbb{R}^{n,n}$ ,  $b \in \mathbb{R}^n$ ,  $n \in \mathbb{N}$  given,  
with **symmetric positive definite** (s.p.d.,  $\rightarrow$  Def. 2.7.9) system matrix  $A$

➔  $A$ -inner product  $(x, y) \mapsto x^T Ay \Rightarrow$  “ $A$ -geometry”

**Definition 5.1.1** (Energy norm).  $\rightarrow$  [35, Def. 9.1]

A s.p.d. matrix  $A \in \mathbb{R}^{n,n}$  induces an **energy norm**

$$\|x\|_A := (x^T Ax)^{1/2}, \quad x \in \mathbb{R}^n.$$

*Remark 5.1.2* (Krylov methods for complex s.p.d. system matrices).

In this chapter, for the sake of simplicity, we restrict ourselves to  $\mathbb{K} = \mathbb{R}$ .

However, the (conjugate) gradient methods introduced below also work for LSE  $\mathbf{Ax} = \mathbf{b}$  with  $\mathbf{A} \in \mathbb{C}^{n,n}$ ,  $\mathbf{A} = \mathbf{A}^H$  s.p.d. when  $\top$  is replaced with  $^H$  (Hermitian transposed). Then, all theoretical statements remain valid unaltered for  $\mathbb{K} = \mathbb{C}$ .



## 5.1.1 Quadratic minimization context

**Lemma 5.1.3** (S.p.d. LSE and quadratic minimization problem). [13, (13.37)]

A LSE with  $\mathbf{A} \in \mathbb{R}^{n,n}$  s.p.d. and  $\mathbf{b} \in \mathbb{R}^n$  is equivalent to a minimization problem:

$$\mathbf{Ax} = \mathbf{b} \Leftrightarrow \mathbf{x} = \arg \min_{\mathbf{y} \in \mathbb{R}^n} J(\mathbf{y}), \quad J(\mathbf{y}) := \frac{1}{2} \mathbf{y}^\top \mathbf{A} \mathbf{y} - \mathbf{b}^\top \mathbf{y}. \quad (5.1.4)$$

A quadratic functional

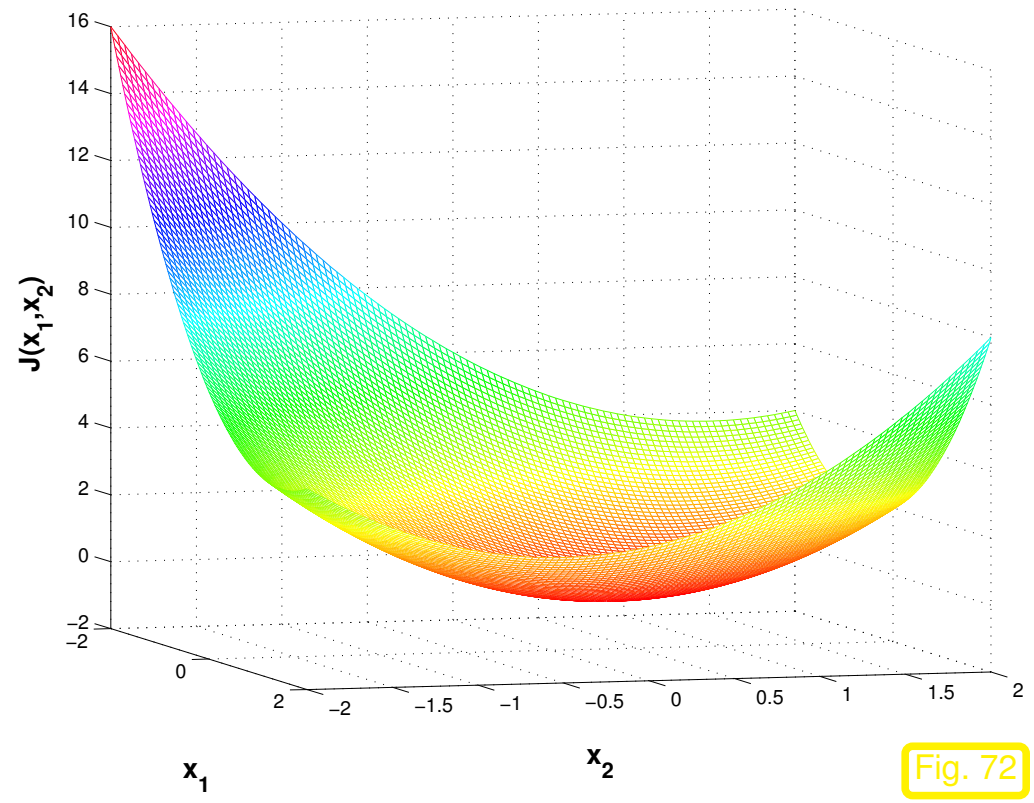
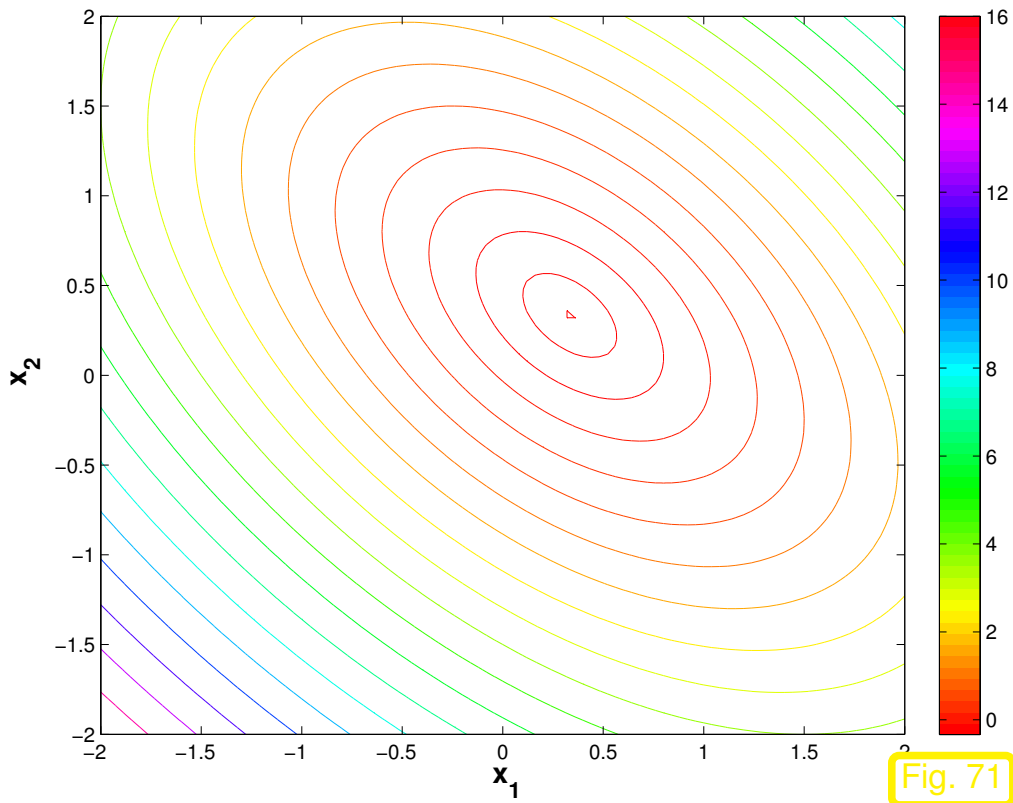
*Proof.* If  $\mathbf{x}^* := \mathbf{A}^{-1}\mathbf{b}$  a straightforward computation using  $\mathbf{A} = \mathbf{A}^T$  shows

$$\begin{aligned} J(\mathbf{x}) - J(\mathbf{x}^*) &= \frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{b}^T \mathbf{x} - \frac{1}{2}(\mathbf{x}^*)^T \mathbf{A} \mathbf{x}^* + \mathbf{b}^T \mathbf{x}^* \\ &\stackrel{\mathbf{b}=\mathbf{A}\mathbf{x}^*}{=} \frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} - (\mathbf{x}^*)^T \mathbf{A} \mathbf{x} + \frac{1}{2}(\mathbf{x}^*)^T \mathbf{A} \mathbf{x}^* \\ &= \frac{1}{2} \|\mathbf{x} - \mathbf{x}^*\|_A^2 . \end{aligned} \tag{5.1.5}$$

Then the assertion follows from the properties of the energy norm.

*Example 5.1.6* (Quadratic functional in 2D).

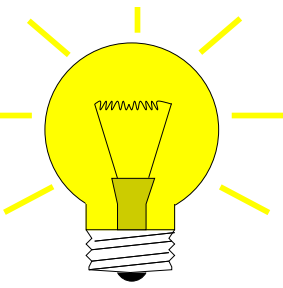
Plot of  $J$  from (5.1.4) for  $\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ ,  $\mathbf{b} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ .



Level lines of quadratic functionals with s.p.d.  $\mathbf{A}$  are (hyper)ellipses







Algorithmic idea: (Lemma 5.1.3  $\rightrightarrows$ ) Solve  $\mathbf{Ax} = \mathbf{b}$  iteratively by **successive** solution of *simpler* minimization problems

## 5.1.2 Abstract steepest descent

Task: Given **continuously differentiable**  $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}$ ,  
find **minimizer**  $\mathbf{x}^* \in D$ :  $\mathbf{x}^* = \underset{\mathbf{x} \in D}{\operatorname{argmin}} F(\mathbf{x})$

Note that a minimizer need not exist, if  $F$  is not bounded from below (e.g.,  $F(x) = x^3$ ,  $x \in \mathbb{R}$ , or  $F(x) = \log x$ ,  $x > 0$ ), or if  $D$  is open (e.g.,  $F(x) = \sqrt{x}$ ,  $x > 0$ ).

The existence of a minimizer is guaranteed if  $F$  is bounded from below and  $D$  is closed ( $\rightarrow$  Analysis).

The most natural iteration:

Initial guess  $\mathbf{x}^{(0)} \in D, k = 0$

**repeat**

$$\mathbf{d}_k := -\mathbf{grad} F(\mathbf{x}^{(k)})$$

$$t^* := \operatorname{argmin}_{t \in \mathbb{R}} F(\mathbf{x}^{(k)} + t\mathbf{d}_k) \quad (\text{line search})$$

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + t^*\mathbf{d}_k$$

$$k := k + 1$$

**until**  $\left( \begin{array}{l} \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \leq \tau_{\text{rel}} \|\mathbf{x}^{(k)}\| \\ \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \leq \tau_{\text{abs}} \end{array} \right)$  **or**

- $\mathbf{d}_k \hat{=}$  direction of steepest descent
- linear search  $\hat{=}$  1D minimization: use Newton's method ( $\rightarrow$  Sect. 4.3.2.1) on derivative
- correction based a posteriori termination criterion, see Sect. 4.1.2 for a discussion.  
( $\tau \hat{=}$  prescribed tolerance)

The **gradient** ( $\rightarrow$  [63, Kapitel 7])

$$\text{grad } F(\mathbf{x}) = \begin{pmatrix} \frac{\partial F}{\partial x_1} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n \quad (5.1.8)$$

provides the direction of **local** steepest ascent/descent of  $F$

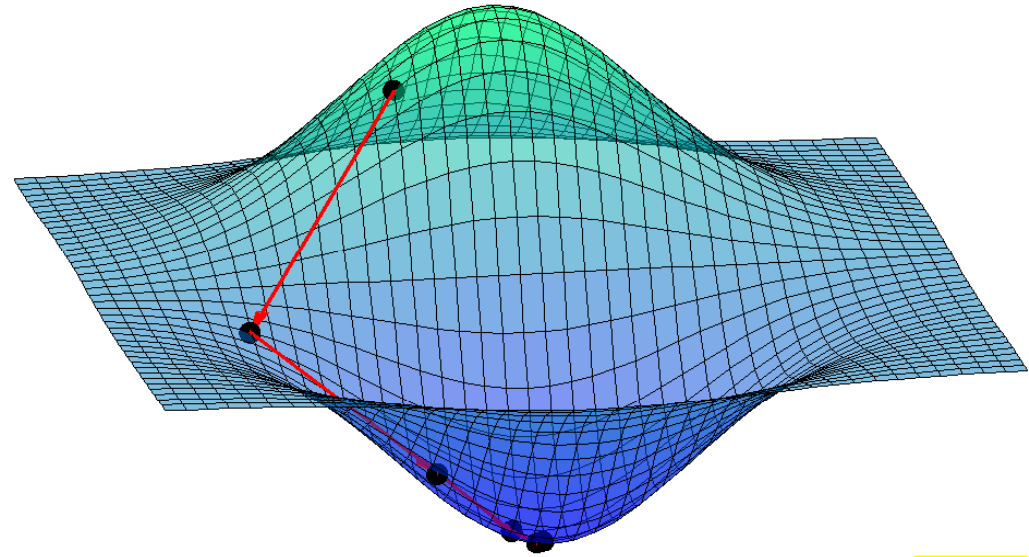


Fig. 73

Of course this very algorithm can encounter plenty of difficulties:

- iteration may get stuck in a *local minimum*,
- iteration may diverge or lead out of  $D$ ,
- line search may not be feasible.

## 5.1.3 Gradient method for s.p.d. linear system of equations

However, for the quadratic minimization problem (5.1.4) Alg. 5.1.7 will converge:

(“Geometric intuition”, see Fig. 71: quadratic functional  $J$  with s.p.d.  $\mathbf{A}$  has unique global minimum,  $\mathbf{grad} J \neq 0$  away from minimum, pointing towards it.)

Adaptation: steepest descent algorithm Alg. 5.1.7 for quadratic minimization problem (5.1.4), see [51, Sect. 7.2.4]:

$$F(\mathbf{x}) := J(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x} \quad \Rightarrow \quad \mathbf{grad} J(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} . \quad (5.1.9)$$

This follows from  $\mathbf{A} = \mathbf{A}^\top$ , the componentwise expression

$$J(\mathbf{x}) = \frac{1}{2} \sum_{i,j=1}^n a_{ij}x_i x_j - \sum_{i=1}^n b_i x_i$$

and the definition (5.1.8) of the gradient.

➤ For the descent direction in Alg. 5.1.7 applied to the minimization of  $J$  from (5.1.4) holds

$$\mathbf{d}_k = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} =: \mathbf{r}_k \quad \text{the residual } (\rightarrow \text{Def. 2.5.16}) \text{ for } \mathbf{x}^{(k-1)} .$$

Alg. 5.1.7 for  $F = J$  from (5.1.4): function to be minimized in line search step:

$$\varphi(t) := J(\mathbf{x}^{(k)} + t\mathbf{d}_k) = J(\mathbf{x}^{(k)}) + t\mathbf{d}_k^\top (\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}) + \frac{1}{2}t^2\mathbf{d}_k^\top \mathbf{A}\mathbf{d}_k \rightarrow \text{a parabola ! .}$$

$$\frac{d\varphi}{dt}(t^*) = 0 \Leftrightarrow t^* = \frac{\mathbf{d}_k^\top \mathbf{d}_k}{\mathbf{d}_k^\top \mathbf{A}\mathbf{d}_k} \quad (\text{unique minimizer}) . \quad (5.1.10)$$

Note:  $\mathbf{d}_k = 0 \Leftrightarrow \mathbf{A}\mathbf{x}^{(k)} = \mathbf{b}$  (solution found !)

Note:  $\mathbf{A}$  s.p.d. ( $\rightarrow$  Def. 2.7.9)  $\Rightarrow \mathbf{d}_k^\top \mathbf{A}\mathbf{d}_k > 0$ , if  $\mathbf{d}_k \neq 0$

►  $\varphi(t)$  is a parabola that is bounded from below (upward opening)

Based on (5.1.9) and (5.1.10) we obtain the following steepest descent method for the minimization problem (5.1.4):

Steepest descent iteration = **gradient method** for LSE  $\mathbf{A}\mathbf{x} = \mathbf{b}$ ,  $\mathbf{A} \in \mathbb{R}^{n,n}$  s.p.d.,  $\mathbf{b} \in \mathbb{R}^n$ :

*Algorithm 5.1.11* (Gradient method for s.p.d. LSE).

Initial guess  $\mathbf{x}^{(0)} \in \mathbb{R}^n$ ,  $k = 0$

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$

**repeat**

$$t^* := \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{r}_k^\top \mathbf{A} \mathbf{r}_k}$$

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + t^* \mathbf{r}_k$$

$$\mathbf{r}_{k+1} := \mathbf{r}_k - t^* \mathbf{A} \mathbf{r}_k$$

$$k := k + 1$$

**until**  $\left( \begin{array}{l} \left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \right\| \leq \tau_{\text{rel}} \left\| \mathbf{x}^{(k)} \right\| \text{ or} \\ \left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \right\| \leq \tau_{\text{abs}} \end{array} \right)$

Code 5.1.12: gradient method for  $\mathbf{A}\mathbf{x} = \mathbf{b}$ ,  $\mathbf{A}$  s.p.d.

```

1 function x =
   gradit(A,b,x,rtol,atol,maxit)
2 r = b-A*x; % residual → Def. 2.5.16
3 for k=1:maxit
4   p = A*r;
5   ts = (r'*r)/(r'*p); % cf. (5.1.10)
6   x = x + ts*r;
7   cn = (abs(ts)*norm(r)); % norm of
   correction
8   if ((cn < tol*norm(x)) ||
9       (cn < atol))
10      return; end
11   r = r - ts*p; %
12 end

```

**Recursion** for residuals, see Line 11 of Code 5.1.11:

$$\mathbf{r}_{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{A}(\mathbf{x}^{(k)} + t^* \mathbf{r}_k) = \mathbf{r}_k - t^* \mathbf{A} \mathbf{r}_k. \quad (5.1.13)$$

One step of gradient method involves

- $A$  single matrix  $\times$  vector product with  $A$ ,
- 2 AXPY-operations ( $\rightarrow$  Sect. 1.4) on vectors of length  $n$ ,
- 2 dot products in  $\mathbb{R}^n$ .

Computational cost (per step) = cost(matrix  $\times$  vector) +  $O(n)$

- If  $A \in \mathbb{R}^{n,n}$  is a sparse matrix ( $\rightarrow$  Sect. 2.6) with “ $O(n)$  nonzero entries”, and the data structures allow to perform the matrix  $\times$  vector product with a computational effort  $O(n)$ , then a single step of the gradient method costs  $O(n)$  elementary operations.
- Gradient method of Alg. 5.1.11 only needs  $A \times$  vector in procedural form  $y = \text{eval}A(x)$ .

## 5.1.4 Convergence of the gradient method


*Example 5.1.14* (Gradient method in 2D).

S.p.d. matrices  $\in \mathbb{R}^{2,2}$ :

$$\mathbf{A}_1 = \begin{pmatrix} 1.9412 & -0.2353 \\ -0.2353 & 1.0588 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} 7.5353 & -1.8588 \\ -1.8588 & 0.5647 \end{pmatrix}$$

Eigenvalues:  $\sigma(\mathbf{A}_1) = \{1, 2\}$ ,

$\sigma(\mathbf{A}_2) = \{0.1, 8\}$

 notation: **spectrum** of a matrix  $\in \mathbb{K}^{n,n}$   $\sigma(\mathbf{M}) := \{\lambda \in \mathbb{C} : \lambda \text{ is eigenvalue of } \mathbf{M}\}$

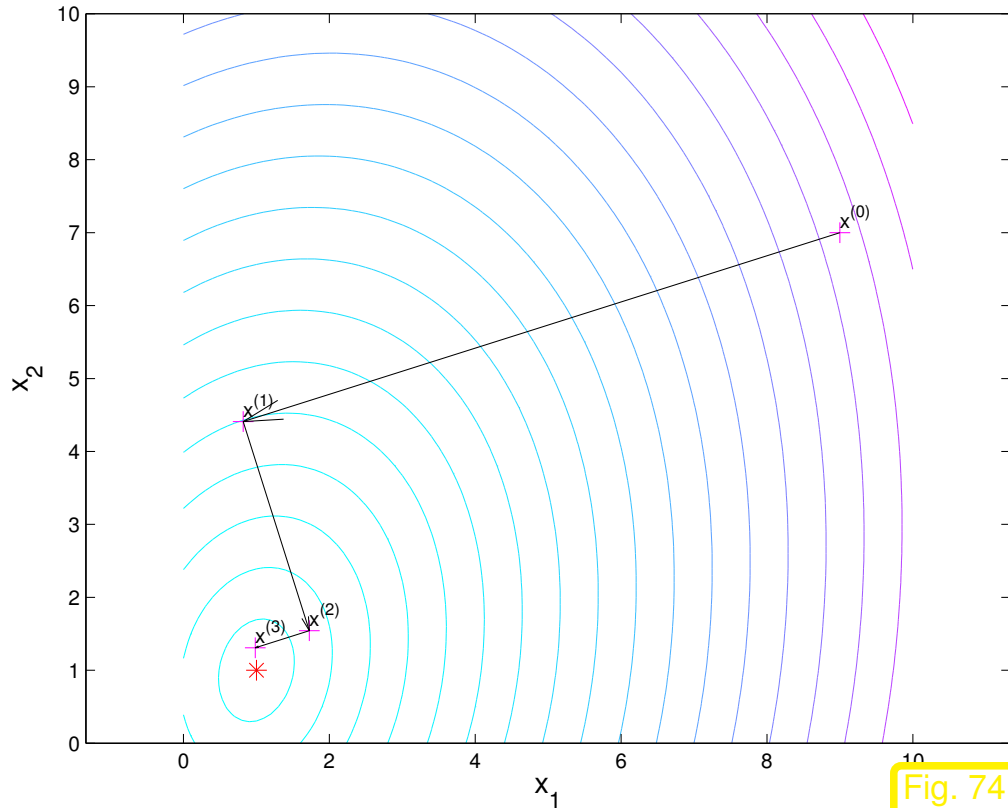


Fig. 74

iterates of Alg. 5.1.11 for  $\mathbf{A}_1$

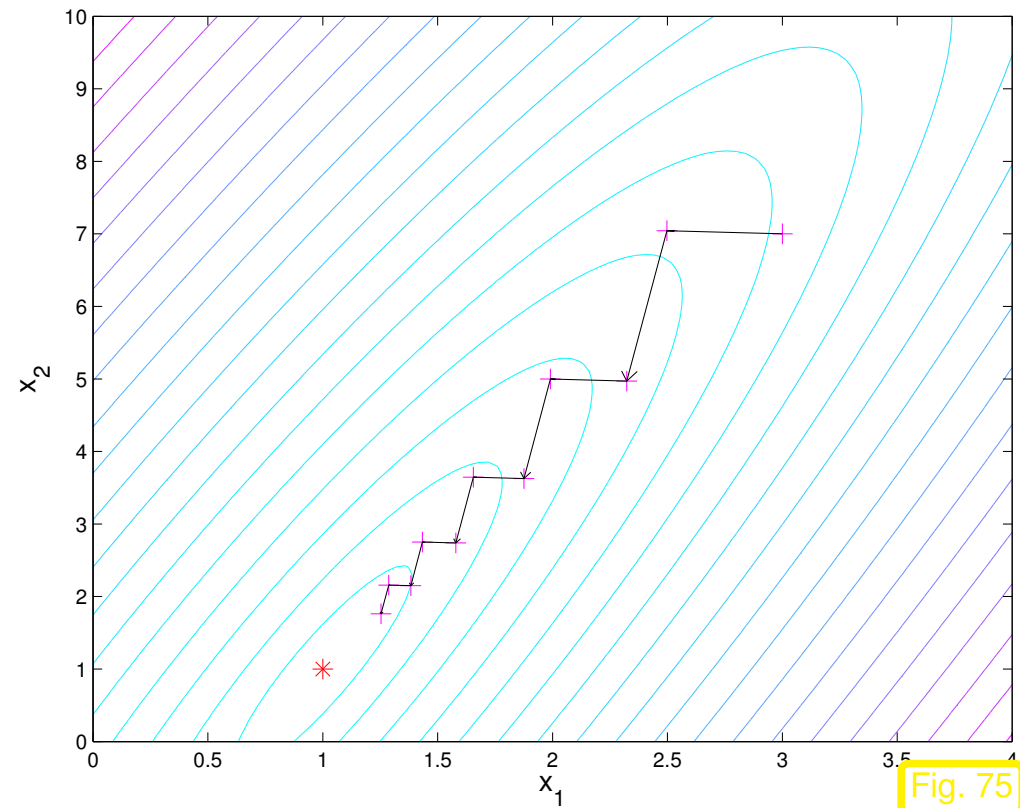


Fig. 75

iterates of Alg. 5.1.11 for  $\mathbf{A}_2$



Recall theorem on principal axis transformation: every real *symmetric* matrix can be diagonalized by *orthogonal* similarity transformations, see Cor. 6.1.9, [48, Thm. 7.8], [27, Satz 9.15],

$$\mathbf{A} = \mathbf{A}^\top \in \mathbb{R}^{n,n} \Rightarrow \exists \mathbf{Q} \in \mathbb{R}^{n,n} \text{ orthogonal: } \mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top, \quad \mathbf{D} = \text{diag}(d_1, \dots, d_n) \in \mathbb{R}^{n,n} \text{ d} \quad (5.1.15)$$

$$J(\mathbf{Q}\hat{\mathbf{y}}) = \frac{1}{2}\hat{\mathbf{y}}^\top \mathbf{D}\hat{\mathbf{y}} - \underbrace{(\mathbf{Q}^\top \mathbf{b})^\top}_{=:\hat{\mathbf{b}}^\top} \hat{\mathbf{y}} = \frac{1}{2} \sum_{i=1}^n d_i \hat{y}_i^2 - \hat{b}_i \hat{y}_i.$$

Hence, a rigid transformation (rotation, reflection) maps the level surfaces of  $J$  from (5.1.4) to ellipses with principal axes  $d_i$ . As  $\mathbf{A}$  s.p.d.  $d_i > 0$  is guaranteed.

Observations:

- Larger spread of spectrum leads to more elongated ellipses as level lines  $\triangleright$  slower convergence of gradient method, see Fig. 75.
- Orthogonality of successive residuals  $\mathbf{r}_k, \mathbf{r}_{k+1}$ .

Clear from definition of Alg. 5.1.11:

$$\mathbf{r}_k^\top \mathbf{r}_{k+1} = \mathbf{r}_k^\top \mathbf{r}_k - \mathbf{r}_k^\top \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{r}_k^\top \mathbf{A} \mathbf{r}_k} \mathbf{A} \mathbf{r}_k = 0. \quad (5.1.16)$$

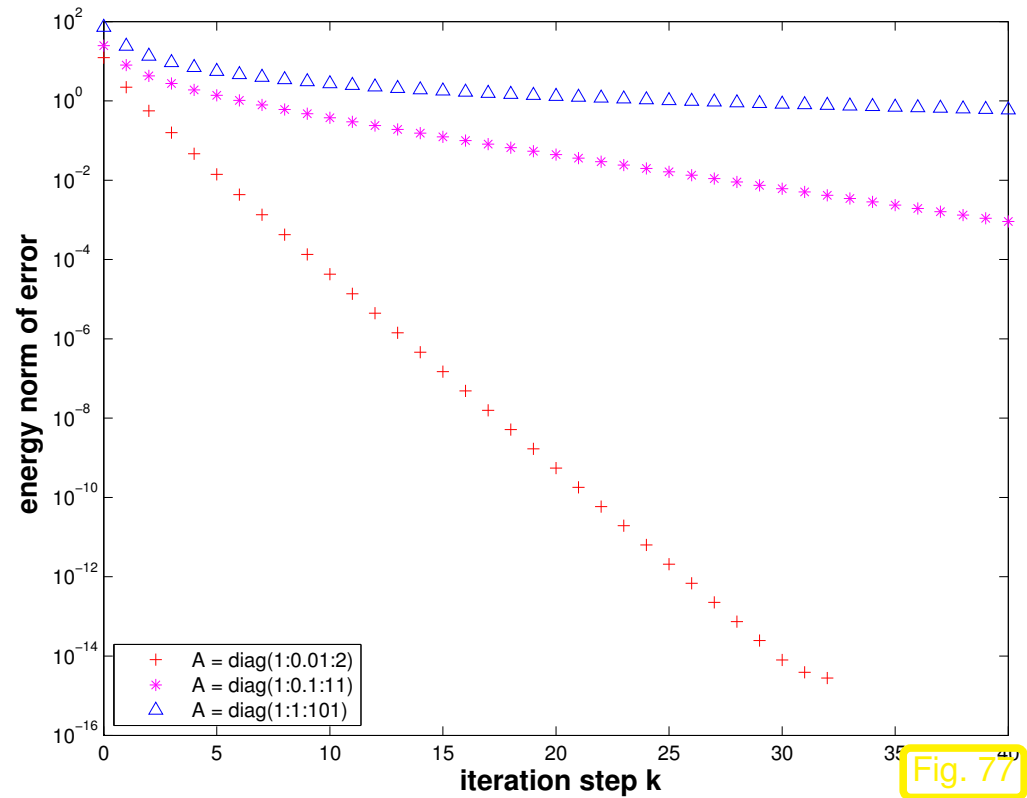
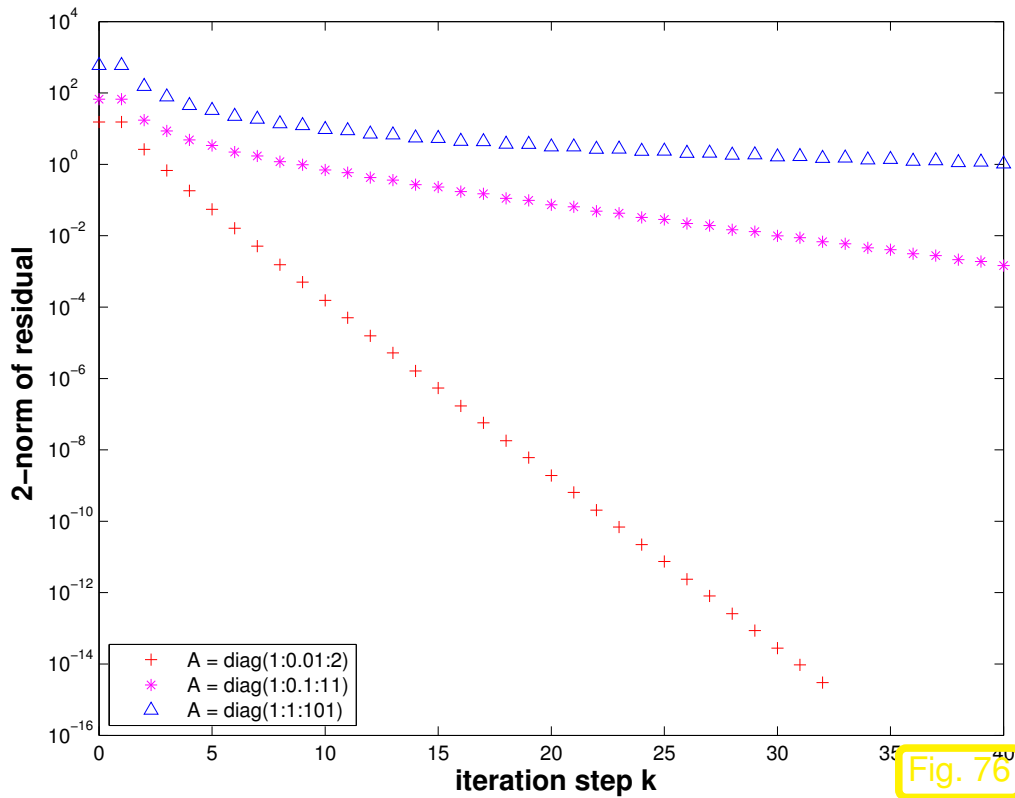
Example 5.1.17 (Convergence of gradient method).

Convergence of gradient method for diagonal matrices,  $\mathbf{x}^* = (1, \dots, 1)^\top$ ,  $\mathbf{x}^{(0)} = \mathbf{0}$ :

```

1   d = 1:0.01:2;   A1 = diag (d) ;
2   d = 1:0.1:11;  A2 = diag (d) ;
3   d = 1:1:101;   A3 = diag (d) ;

```



Note: To study convergence it is *sufficient to consider diagonal matrices*, because

1. (5.1.15): for every  $\mathbf{A} \in \mathbb{R}^{n,n}$  with  $\mathbf{A}^\top = \mathbf{A}$  there is an orthogonal matrix  $\mathbf{Q} \in \mathbb{R}^{n,n}$  such that  $\mathbf{A} = \mathbf{Q}^\top \mathbf{D} \mathbf{Q}$  with a diagonal matrix  $\mathbf{D}$  (principal axis transformation),  $\rightarrow$  Cor. 6.1.9, [48, Thm. 7.8], [27, Satz 9.15],
2. when applying the gradient method Alg. 5.1.11 to both  $\mathbf{A}\mathbf{x} = \mathbf{b}$  and  $\mathbf{D}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} := \mathbf{Q}\mathbf{b}$ , then the iterates  $\mathbf{x}^{(k)}$  and  $\tilde{\mathbf{x}}^{(k)}$  are related by  $\mathbf{Q}\mathbf{x}^{(k)} = \tilde{\mathbf{x}}^{(k)}$ .

With  $\tilde{\mathbf{r}}_k := \mathbf{Q}\mathbf{r}_k$ ,  $\tilde{\mathbf{x}}^{(k)} := \mathbf{Q}\mathbf{x}^{(k)}$ , using  $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$ :

Initial guess  $\mathbf{x}^{(0)} \in \mathbb{R}^n$ ,  $k = 0$

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{Q}^\top \mathbf{D} \mathbf{Q} \mathbf{x}^{(0)}$$

**repeat**

$$t^* := \frac{\mathbf{r}_k^\top \mathbf{Q}^\top \mathbf{Q} \mathbf{r}_k}{\mathbf{r}_k^\top \mathbf{Q}^\top \mathbf{D} \mathbf{Q} \mathbf{r}_k}$$

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + t^* \mathbf{r}_k$$

$$\mathbf{r}_{k+1} := \mathbf{r}_k - t^* \mathbf{Q}^\top \mathbf{D} \mathbf{Q} \mathbf{r}_k$$

$$k := k + 1$$

**until**  $\left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \right\| \leq \tau \left\| \mathbf{x}^{(k)} \right\|$

Initial guess  $\tilde{\mathbf{x}}^{(0)} \in \mathbb{R}^n$ ,  $k = 0$

$$\tilde{\mathbf{r}}_0 := \tilde{\mathbf{b}} - \mathbf{D} \tilde{\mathbf{x}}^{(0)}$$

**repeat**

$$t^* := \frac{\tilde{\mathbf{r}}_k^\top \tilde{\mathbf{r}}_k}{\tilde{\mathbf{r}}_k^\top \mathbf{D} \tilde{\mathbf{r}}_k}$$

$$\tilde{\mathbf{x}}^{(k+1)} := \tilde{\mathbf{x}}^{(k)} + t^* \tilde{\mathbf{r}}_k$$

$$\tilde{\mathbf{r}}_{k+1} := \tilde{\mathbf{r}}_k - t^* \mathbf{D} \tilde{\mathbf{r}}_k$$

$$k := k + 1$$

**until**  $\left\| \tilde{\mathbf{x}}^{(k)} - \tilde{\mathbf{x}}^{(k-1)} \right\| \leq \tau \left\| \tilde{\mathbf{x}}^{(k)} \right\|$

Observation:

- **linear convergence** ( $\rightarrow$  Def. 4.1.6), see also Rem. 4.1.10
- rate of convergence increases ( $\leftrightarrow$  speed of convergence decreases) with spread of spectrum of **A**

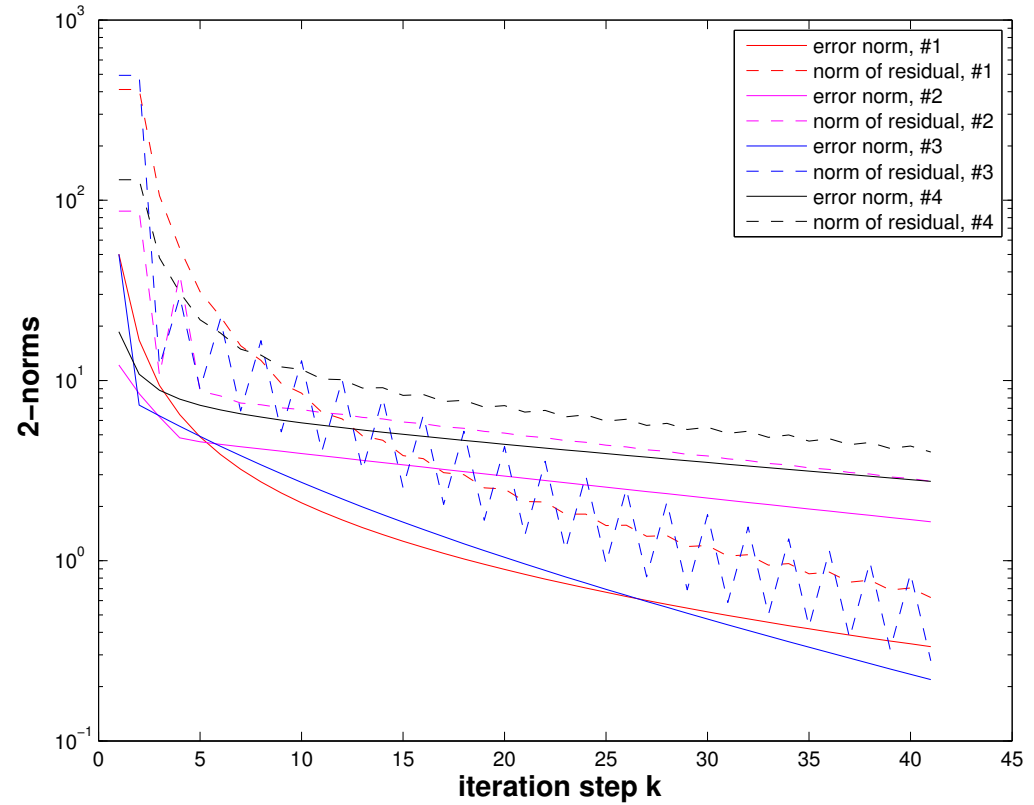
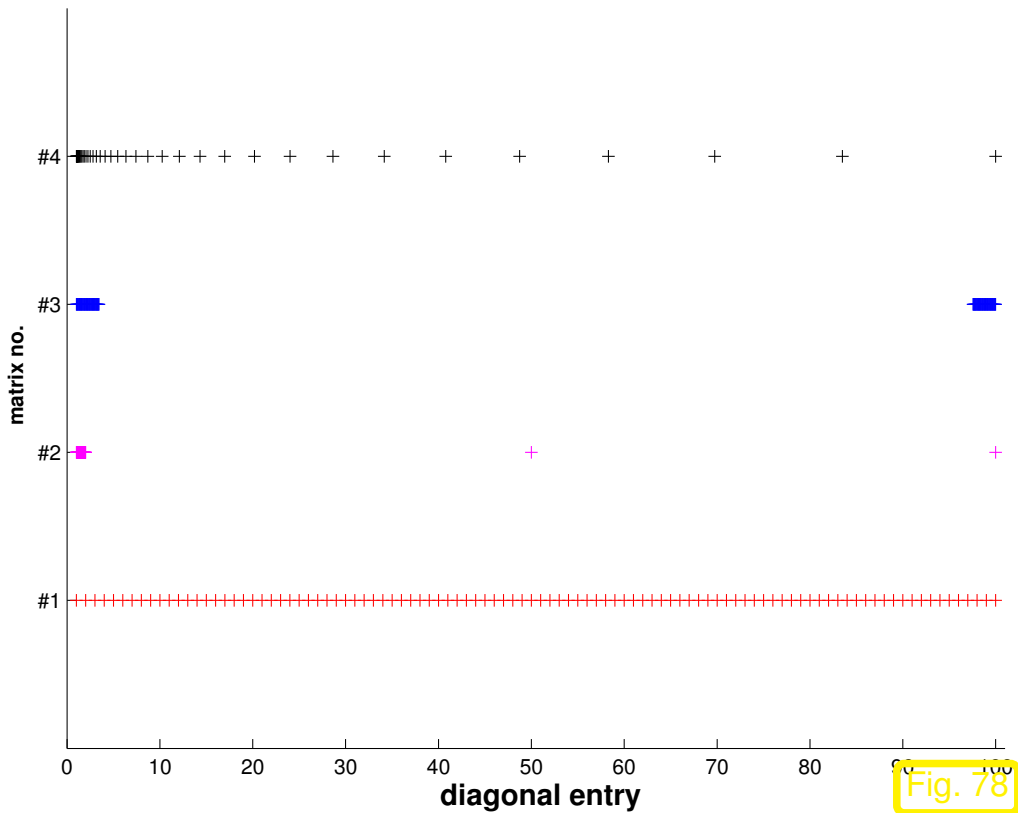
Impact of distribution of diagonal entries ( $\leftrightarrow$  eigenvalues) of (diagonal matrix) **A**  
( $\mathbf{b} = \mathbf{x}^* = 0$ ,  $\mathbf{x}_0 = \cos((1:n)')$ );

Test matrix **#1**:  $A = \text{diag}(d)$ ;  $d = (1:100)$ ;

Test matrix **#2**:  $A = \text{diag}(d)$ ;  $d = [1 + (0:97)/97, 50, 100]$ ;

Test matrix **#3**:  $A = \text{diag}(d)$ ;  $d = [1 + (0:49) * 0.05, 100 - (0:49) * 0.05]$ ;

Test matrix **#4**: eigenvalues exponentially dense at 1



Observation: Matrices #1, #2 & #4  $\triangleright$  little impact of distribution of eigenvalues on *asymptotic* convergence (exception: matrix #2)



**Theorem 5.1.18** (Convergence of gradient method/steepest descent).

*The iterates of the gradient method of Alg. 5.1.11 satisfy*

$$\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\|_A \leq L \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\|_A, \quad L := \frac{\text{cond}_2(\mathbf{A}) - 1}{\text{cond}_2(\mathbf{A}) + 1},$$


*that is, the iteration converges at least linearly ( $\rightarrow$  Def. 4.1.6) w.r.t. energy norm ( $\rightarrow$  Def. 5.1.1).*

 notation:  $\text{cond}_2(\mathbf{A}) \hat{=}$  condition number ( $\rightarrow$  Def. 2.5.26) of  $\mathbf{A}$  induced by 2-norm

*Remark 5.1.19* (2-norm from eigenvalues).  $\rightarrow$  [27, Sect. 10.6], [48, Sect. 7.4]

$$\begin{aligned} \mathbf{A} = \mathbf{A}^\top \quad \Rightarrow \quad & \left\| \mathbf{A} \right\|_2 = \max(|\sigma(\mathbf{A})|), \\ & \left\| \mathbf{A}^{-1} \right\|_2 = \min(|\sigma(\mathbf{A})|)^{-1}, \text{ if } \mathbf{A} \text{ regular.} \end{aligned} \quad (5.1.20)$$

$$\mathbf{A} = \mathbf{A}^\top \quad \Rightarrow \quad \text{cond}_2(\mathbf{A}) = \frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})}, \quad \text{where} \quad \begin{aligned} \lambda_{\max}(\mathbf{A}) &:= \max(|\sigma(\mathbf{A})|), \\ \lambda_{\min}(\mathbf{A}) &:= \min(|\sigma(\mathbf{A})|). \end{aligned} \quad (5.1.21)$$

 other notation  $\kappa(\mathbf{A}) := \frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})} \hat{=} \text{spectral condition number of } \mathbf{A}$

(for general  $\mathbf{A}$ :  $\lambda_{\max}(\mathbf{A})/\lambda_{\min}(\mathbf{A})$  largest/smallest eigenvalue *in modulus*)

These results are an immediate consequence of the fact that

$$\forall \mathbf{A} \in \mathbb{R}^{n,n}, \quad \mathbf{A}^\top = \mathbf{A} \quad \exists \mathbf{U} \in \mathbb{R}^{n,n}, \quad \mathbf{U}^{-1} = \mathbf{U}^\top: \quad \mathbf{U}^\top \mathbf{A} \mathbf{U} \quad \text{is diagonal,}$$

see (5.1.15), Cor. 6.1.9, [48, Thm. 7.8], [27, Satz 9.15].

Please note that for general regular  $\mathbf{M} \in \mathbb{R}^{n,n}$  we *cannot* expect  $\text{cond}_2(\mathbf{M}) = \kappa(\mathbf{M})$ .

## 5.2 Conjugate gradient method (CG) [35, Ch. 9], [13, Sect. 13.4], [51, Sect. 4.3.4]

Again we consider a linear system of equations  $\mathbf{Ax} = \mathbf{b}$  with s.p.d. ( $\rightarrow$  Def. 2.7.9) system matrix  $\mathbf{A} \in \mathbb{R}^{n,n}$  and given  $\mathbf{b} \in \mathbb{R}^n$ .

Liability of gradient method of Sect. 5.1.3:

**NO MEMORY**

1D line search in Alg. 5.1.11 is oblivious of former line searches, which rules out reuse of information gained in previous steps of the iteration. This is a typical drawback of 1-point iterative methods.



Idea:

Replace linear search with **subspace correction**Given: • initial guess  $\mathbf{x}^{(0)}$ • **nested** subspaces  $U_1 \subset U_2 \subset U_3 \subset \dots \subset U_n = \mathbb{R}^n$ ,  $\dim U_k = k$ 

$$\mathbf{x}^{(k)} := \operatorname{argmin}_{\mathbf{x} \in U_k + \mathbf{x}^{(0)}} J(x), \quad (5.2.1)$$

quadratic functional from (5.1.4)

Note: Once the subspaces  $U_k$  and  $\mathbf{x}^{(0)}$  are fixed, the iteration (5.2.1) is well defined, because  $J|_{U_k + \mathbf{x}^{(0)}}$  always possess a unique minimizer.

Obvious (from Lemma 5.1.3):

$$\mathbf{x}^{(n)} = \mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$$

Thanks to (5.1.5), definition (5.2.1) ensures:

$$\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\|_A \leq \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\|_A$$



How to find suitable subspaces  $U_k$  ?

Idea:  $U_{k+1} \leftarrow U_k +$  “local steepest descent direction”

given by  $-\mathbf{grad} J(\mathbf{x}^{(k)}) = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} = \mathbf{r}_k$  (residual  $\rightarrow$  Def. 2.5.16)

$$U_{k+1} = \text{Span} \{U_k, \mathbf{r}_k\}, \quad \mathbf{x}^{(k)} \text{ from (5.2.1)}. \quad (5.2.2)$$

Obvious:  $\mathbf{r}_k = 0 \Rightarrow \mathbf{x}^{(k)} = \mathbf{x}^* := \mathbf{A}^{-1}\mathbf{b}$  done ✓

**Lemma 5.2.3** ( $\mathbf{r}_k \perp U_k$ ).

With  $\mathbf{x}^{(k)}$  according to (5.2.1),  $U_k$  from (5.2.2) the residual  $\mathbf{r}_k := \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$  satisfies

$$\mathbf{r}_k^\top \mathbf{u} = 0 \quad \forall \mathbf{u} \in U_k \quad (\text{“}\mathbf{r}_k \perp U_k\text{”}).$$

Geometric consideration: since  $\mathbf{x}^{(k)}$  is the minimizer of  $J$  over the affine space  $U_k + \mathbf{x}^{(0)}$ , the projection of the steepest descent direction  $\mathbf{grad} J(\mathbf{x}^{(k)})$  onto  $U_k$  has to vanish:

$$\mathbf{x}^{(k)} := \operatorname{argmin}_{\mathbf{x} \in U_k + \mathbf{x}^{(0)}} J(\mathbf{x}) \Rightarrow \mathbf{grad} J(\mathbf{x}^{(k)}) \perp U_k. \quad (5.2.4)$$

$$\psi(t) = J(\mathbf{x}^{(k)} + t\mathbf{u}), \quad \mathbf{u} \in U_k, \quad t \in \mathbb{R}.$$

By (5.2.1),  $t \mapsto \psi(t)$  has a global minimum in  $t = 0$ , which implies

$$\frac{d\psi}{dt}(0) = \mathbf{grad} J(\mathbf{x}^{(k)})^\top \mathbf{u} = (\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b})^\top \mathbf{u} = 0.$$

Since  $\mathbf{u} \in U_k$  was arbitrary, the lemma is proved. □

**Corollary 5.2.5.** *If  $\mathbf{r}_l \neq 0$  for  $l = 0, \dots, k$ ,  $k \leq n$ , then  $\{\mathbf{r}_0, \dots, \mathbf{r}_k\}$  is an *orthogonal basis* of  $U_k$ .*

Lemma 5.2.3 also implies that, if  $U_0 = \{0\}$ , then  $\dim U_k = k$  as long as  $\mathbf{x}^{(k)} \neq \mathbf{x}^*$ , that is, before we have converged to the exact solution.

(5.2.1) and (5.2.2) define the **conjugate gradient method** (CG) for the iterative solution of

$$\mathbf{Ax} = \mathbf{b}$$

(hailed as a “top ten algorithm” of the 20th century, SIAM News, 33(4))

## 5.2.1 Krylov spaces

**Definition 5.2.6** (Krylov space).

For  $\mathbf{A} \in \mathbb{R}^{n,n}$ ,  $\mathbf{z} \in \mathbb{R}^n$ ,  $\mathbf{z} \neq \mathbf{0}$ , the  $l$ -th **Krylov space** is defined as

$$\mathcal{K}_l(\mathbf{A}, \mathbf{z}) := \text{Span} \left\{ \mathbf{z}, \mathbf{Az}, \dots, \mathbf{A}^{l-1}\mathbf{z} \right\} .$$

Equivalent definition:

$$\mathcal{K}_l(\mathbf{A}, \mathbf{z}) = \{p(\mathbf{A})\mathbf{z} : p \text{ polynomial of degree } \leq l\}$$

**Lemma 5.2.7.** *The subspaces  $U_k \subset \mathbb{R}^n$ ,  $k \geq 1$ , defined by (5.2.1) and (5.2.2) satisfy*

$$U_k = \text{Span} \left\{ \mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \dots, \mathbf{A}^{k-1}\mathbf{r}_0 \right\} = \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0),$$

where  $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$  is the initial residual.

*Proof.* (by induction) Obviously  $\mathbf{A}\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0) \subset \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{r}_0)$ . In addition

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}(\mathbf{x}^{(0)} + \mathbf{z}) \quad \text{for some } \mathbf{z} \in U_k \quad \Rightarrow \quad \mathbf{r}_k = \underbrace{\mathbf{r}_0}_{\in \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{r}_0)} - \underbrace{\mathbf{A}\mathbf{z}}_{\in \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{r}_0)}.$$

Since  $U_{k+1} = \text{Span} \{U_k, \mathbf{r}_k\}$ , we obtain  $U_{k+1} \subset \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{r}_0)$ . Dimensional considerations based on Lemma 5.2.3 finish the proof.  $\square$

Assume: **basis**  $\{\mathbf{p}_1, \dots, \mathbf{p}_l\}$ ,  $l = 1, \dots, n$ , of  $\mathcal{K}_l(\mathbf{A}, \mathbf{r})$  available

$$\mathbf{x}^{(l)} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0) \quad \triangleright \quad \text{set} \quad \mathbf{x}^{(l)} = \mathbf{x}^{(0)} + \gamma_1 \mathbf{p}_1 + \dots + \gamma_l \mathbf{p}_l .$$

For  $\psi(\gamma_1, \dots, \gamma_l) := J(\mathbf{x}^{(0)} + \gamma_1 \mathbf{p}_1 + \dots + \gamma_l \mathbf{p}_l)$  holds

$$(5.2.1) \quad \Leftrightarrow \quad \frac{\partial \psi}{\partial \gamma_j} = 0, \quad j = 1, \dots, l .$$

This leads to a linear system of equations by which the coefficients  $\gamma_j$  can be computed:

$$\begin{pmatrix} \mathbf{p}_1^\top \mathbf{A} \mathbf{p}_1 & \dots & \mathbf{p}_1^\top \mathbf{A} \mathbf{p}_l \\ \vdots & & \vdots \\ \mathbf{p}_l^\top \mathbf{A} \mathbf{p}_1 & \dots & \mathbf{p}_l^\top \mathbf{A} \mathbf{p}_l \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_l \end{pmatrix} = \begin{pmatrix} \mathbf{p}_1^\top \mathbf{r} \\ \vdots \\ \mathbf{p}_l^\top \mathbf{r} \end{pmatrix}, \quad \mathbf{r} := \mathbf{b} - \mathbf{A} \mathbf{x}^{(0)}. \quad (5.2.8)$$

Great simplification, if  $\{\mathbf{p}_1, \dots, \mathbf{p}_l\}$  **A-orthogonal basis** of  $\mathcal{K}_l(\mathbf{A}, \mathbf{r})$ :  $\mathbf{p}_j^\top \mathbf{A} \mathbf{p}_i = 0$  for  $i \neq j$ .

Recall: s.p.d.  $\mathbf{A}$  induces an inner product  $\triangleright$  concept of orthogonality [48, Sect. 4.4], [27, Sect. 6.2]. “ $\mathbf{A}$ -geometry” like standard Euclidean space.

Assume:  $\mathbf{A}$ -orthogonal basis  $\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$  of  $\mathbb{R}^n$  available, such that

$$\text{Span}\{\mathbf{p}_1, \dots, \mathbf{p}_l\} = \mathcal{K}_l(\mathbf{A}, \mathbf{r}) .$$

► (Efficient) successive computation of  $\mathbf{x}^{(l)}$  becomes possible, see [13, Lemma 13.24]  
(LSE (5.2.8) becomes diagonal !)

Input: : initial guess  $\mathbf{x}^{(0)} \in \mathbb{R}^n$

Given: :  $\mathbf{A}$ -orthogonal bases  $\{\mathbf{p}_1, \dots, \mathbf{p}_l\}$  of  $\mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)$ ,  $l = 1, \dots, n$

Output: : approximate solution  $\mathbf{x}^{(l)} \in \mathbb{R}^n$  of  $\mathbf{Ax} = \mathbf{b}$

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{Ax}^{(0)};$$

$$\text{for } j = 1 \text{ to } l \text{ do } \left\{ \mathbf{x}^{(j)} := \mathbf{x}^{(j-1)} + \frac{\mathbf{p}_j^\top \mathbf{r}_0}{\mathbf{p}_j^\top \mathbf{A} \mathbf{p}_j} \mathbf{p}_j \right\} \quad (5.2.9)$$

**Task:** Efficient computation of  $\mathbf{A}$ -orthogonal vectors  $\{\mathbf{p}_1, \dots, \mathbf{p}_l\}$  spanning  $\mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)$  during the CG iteration.

Lemma 5.2.3 implies orthogonality  $\mathbf{p}_j \perp \mathbf{r}_m := \mathbf{b} - \mathbf{A}\mathbf{x}^{(m)}$ ,  $1 \leq j \leq m \leq l$ . Also by  $\mathbf{A}$ -orthogonality of the  $\mathbf{p}_k$

$$\mathbf{p}_j^T (\mathbf{b} - \mathbf{A}\mathbf{x}^{(m)}) = \mathbf{p}_j^T \left( \underbrace{\mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}}_{=\mathbf{r}_0} - \sum_{k=1}^m \frac{\mathbf{p}_k^T \mathbf{r}_0}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \mathbf{A}\mathbf{p}_k \right) = 0. \quad (5.2.10)$$

From linear algebra we already know a way to construct orthogonal basis vectors:

(5.2.10)  $\Rightarrow$  Idea: **Gram-Schmidt orthogonalization** [48, Thm. 4.8], [27, Alg. 6.1], of residuals  $\mathbf{r}_j := \mathbf{b} - \mathbf{A}\mathbf{x}^{(j)}$  w.r.t.  $\mathbf{A}$ -inner product:

$$\mathbf{p}_1 := \mathbf{r}_0, \quad \mathbf{p}_{j+1} := \underbrace{(\mathbf{b} - \mathbf{A}\mathbf{x}^{(j)})}_{=\mathbf{r}_j} - \underbrace{\sum_{k=1}^j \frac{\mathbf{p}_k^T \mathbf{A}\mathbf{r}_j}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \mathbf{p}_k}_{(*)}, \quad j = 1, \dots, l-1.$$

(5.2.11)

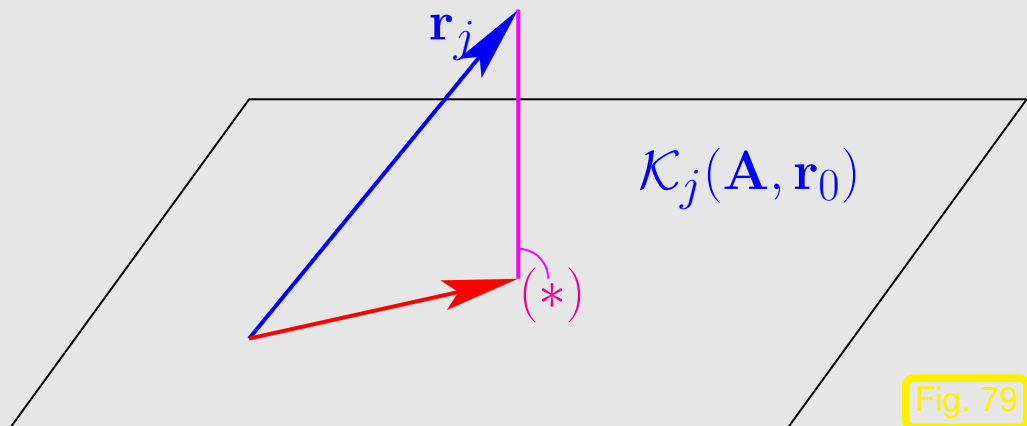


Fig. 79

Geometric interpretation of (5.2.11):

$\hat{\mathbf{x}} \hat{=}$  orthogonal projection of  $\mathbf{r}_j$  on the subspace  $\text{Span} \{ \mathbf{p}_1, \dots, \mathbf{p}_j \}$



**Lemma 5.2.12** (Bases for Krylov spaces in CG).

If they do not vanish, the vectors  $\mathbf{p}_j$ ,  $1 \leq j \leq l$ , and  $\mathbf{r}_j := \mathbf{b} - \mathbf{A}\mathbf{x}^{(j)}$ ,  $0 \leq j \leq l$ , from (5.2.9), (5.2.11) satisfy

- (i)  $\{\mathbf{p}_1, \dots, \mathbf{p}_j\}$  is  $\mathbf{A}$ -orthogonal basis von  $\mathcal{K}_j(\mathbf{A}, \mathbf{r}_0)$ ,
- (ii)  $\{\mathbf{r}_0, \dots, \mathbf{r}_{j-1}\}$  is orthogonal basis of  $\mathcal{K}_j(\mathbf{A}, \mathbf{r}_0)$ , cf. Cor. 5.2.5

*Proof.*  $\mathbf{A}$ -orthogonality of  $\mathbf{p}_j$  by construction, study (5.2.11).

$$(5.2.9) \ \& \ (5.2.11) \ \Rightarrow \ \mathbf{p}_{j+1} = \mathbf{r}_0 - \sum_{k=1}^j \frac{\mathbf{p}_k^\top \mathbf{r}_0}{\mathbf{p}_k^\top \mathbf{A} \mathbf{p}_k} \mathbf{A} \mathbf{p}_k - \sum_{k=1}^j \frac{\mathbf{p}_k^\top \mathbf{A} \mathbf{r}_j}{\mathbf{p}_k^\top \mathbf{A} \mathbf{p}_k} \mathbf{p}_k .$$

$$\Rightarrow \ \mathbf{p}_{j+1} \in \text{Span} \{ \mathbf{r}_0, \mathbf{p}_1, \dots, \mathbf{p}_j, \mathbf{A} \mathbf{p}_1, \dots, \mathbf{A} \mathbf{p}_j \} .$$

A simple induction argument confirms (i)

$$(5.2.11) \ \Rightarrow \ \mathbf{r}_j \in \text{Span} \{ \mathbf{p}_1, \dots, \mathbf{p}_{j+1} \} \ \& \ \mathbf{p}_j \in \text{Span} \{ \mathbf{r}_0, \dots, \mathbf{r}_{j-1} \} . \quad (5.2.13)$$

$$\blacktriangleright \ \boxed{\text{Span} \{ \mathbf{p}_1, \dots, \mathbf{p}_j \} = \text{Span} \{ \mathbf{r}_0, \dots, \mathbf{r}_{j-1} \} = \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)} . \quad (5.2.14)$$

$$(5.2.10) \ \Rightarrow \ \mathbf{r}_j \perp \text{Span} \{ \mathbf{p}_1, \dots, \mathbf{p}_j \} = \text{Span} \{ \mathbf{r}_0, \dots, \mathbf{r}_{j-1} \} . \quad (5.2.15)$$

Orthogonalities from Lemma 5.2.12 ➤ **short recursions** for  $\mathbf{p}_k, \mathbf{r}_k, \mathbf{x}^{(k)}$  !

$$(5.2.10) \Rightarrow (5.2.11) \text{ collapses to } \mathbf{p}_{j+1} := \mathbf{r}_j - \frac{\mathbf{p}_j^\top \mathbf{A} \mathbf{r}_j}{\mathbf{p}_j^\top \mathbf{A} \mathbf{p}_j} \mathbf{p}_j, \quad j = 1, \dots, l.$$

recursion for residuals:

$$(5.2.9) \quad \blacktriangleright \quad \mathbf{r}_j = \mathbf{r}_{j-1} - \frac{\mathbf{p}_j^\top \mathbf{r}_0}{\mathbf{p}_j^\top \mathbf{A} \mathbf{p}_j} \mathbf{A} \mathbf{p}_j.$$

$$\text{Lemma 5.2.12, (i)} \quad \blacktriangleright \quad \mathbf{r}_{j-1}^H \mathbf{p}_j = \left( \mathbf{r}_0 + \sum_{k=1}^{m-1} \frac{\mathbf{r}_0^\top \mathbf{p}_k}{\mathbf{p}_k^\top \mathbf{A} \mathbf{p}_k} \mathbf{A} \mathbf{p}_k \right)^\top \mathbf{p}_j = \mathbf{r}_0^\top \mathbf{p}_j. \quad (5.2.16)$$

The orthogonality (5.2.16) together with (5.2.15) permits us to replace  $\mathbf{r}_0$  with  $\mathbf{r}_{j-1}$  in the actual implementation.



*Algorithm 5.2.17* (CG method for solving  $\mathbf{A} \mathbf{x} = \mathbf{b}$ ,  $\mathbf{A}$  s.p.d.). → [13, Alg. 13.27]

Input : initial guess  $\mathbf{x}^{(0)} \in \mathbb{R}^n$

Output : approximate solution  $\mathbf{x}^{(l)} \in \mathbb{R}^n$

$$\mathbf{p}_1 = \mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)};$$

for  $j = 1$  to  $l$  do {

$$\mathbf{x}^{(j)} := \mathbf{x}^{(j-1)} + \frac{\mathbf{p}_j^T \mathbf{r}_{j-1}}{\mathbf{p}_j^T \mathbf{A}\mathbf{p}_j} \mathbf{p}_j;$$

$$\mathbf{r}_j = \mathbf{r}_{j-1} - \frac{\mathbf{p}_j^T \mathbf{r}_{j-1}}{\mathbf{p}_j^T \mathbf{A}\mathbf{p}_j} \mathbf{A}\mathbf{p}_j;$$

$$\mathbf{p}_{j+1} = \mathbf{r}_j - \frac{(\mathbf{A}\mathbf{p}_j)^T \mathbf{r}_j}{\mathbf{p}_j^T \mathbf{A}\mathbf{p}_j} \mathbf{p}_j;$$

}

Input: initial guess  $\mathbf{x} \hat{=} \mathbf{x}^{(0)} \in \mathbb{R}^n$

tolerance  $\tau > 0$

Output: approximate solution  $\mathbf{x} \hat{=} \mathbf{x}^{(l)}$

$$\mathbf{p} := \mathbf{r}_0 := \mathbf{r} := \mathbf{b} - \mathbf{A}\mathbf{x};$$

for  $j = 1$  to  $l_{\max}$  do {

$$\beta := \mathbf{r}^T \mathbf{r};$$

$$\mathbf{h} := \mathbf{A}\mathbf{p};$$

$$\alpha := \frac{\beta}{\mathbf{p}^T \mathbf{h}};$$

$$\mathbf{x} := \mathbf{x} + \alpha \mathbf{p};$$

$$\mathbf{r} := \mathbf{r} - \alpha \mathbf{h};$$

if  $\|\mathbf{r}\| \leq \tau \|\mathbf{r}_0\|$  then stop;

$$\beta := \frac{\mathbf{r}^T \mathbf{r}}{\beta};$$

$$\mathbf{p} := \mathbf{r} + \beta \mathbf{p};$$

}

In CG algorithm:  $\mathbf{r}_j = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$  agrees with the residual associated with the current iterate (in exact arithmetic, *cf.* Ex. 5.2.21), but computation through short recursion is more efficient.

➤ We find that the CG method possesses all the algorithmic advantages of the gradient method, *cf.* the discussion in Sect. 5.1.3.

1 matrix  $\times$  vector product, 3 dot products, 3 **AXPY-operations** per step:  
If  $\mathbf{A}$  sparse,  $\text{nnz}(\mathbf{A}) \sim n$  ➤ computational effort  $O(n)$  per step

Code 5.2.18: basic CG iteration for solving  $\mathbf{Ax} = \mathbf{b}$ , Alg. 5.2.17

```

1 function x = cg(evalA,b,x,tol,maxit)
2 % x supplies initial guess, maxit maximal number of CG steps
3 % evalA must pass a handle to a MATLAB function realizing A*x
4 r = b - evalA(x); rho = 1; n0 = norm(r);
5 for i = 1 : maxit
6     rho1 = rho; rho = r' * r;
7     if (i == 1), p = r;

```

```

3  else beta = rho/rho1; p = r + beta * p; end
9  q = evalA(p); alpha = rho/(p' * q);
0  x = x + alpha * p;      % update of approximate solution
1  if (norm(b-A*x) <= tol*n0) return; end % termination, see Rem. 5.2.19
2  r = r - alpha * q;      % update of residual
3  end

```

MATLAB-function:

`x=pcg(A,b,tol,maxit,[],[],x0)` : Solve  $\mathbf{Ax} = \mathbf{b}$  with at most `maxit` CG steps:  
stop, when  $\|\mathbf{r}_l\| : \|\mathbf{r}_0\| < \text{tol}$ .

`x=pcg(Afun,b,tol,maxit,[],[],x0)`: `Afun` = handle to function for computing  
 $\mathbf{A} \times \text{vector}$ .

`[x,flag,relr,it,resv] = pcg(...)` : diagnostic information about iteration

*Remark 5.2.19* (A posteriori termination criterion for plain CG).

For any vector norm and associated matrix norm ( $\rightarrow$  Def. 2.5.5) hold (with residual  $\mathbf{r}_l := \mathbf{b} - \mathbf{A}\mathbf{x}^{(l)}$ )

$$\frac{1}{\text{cond}(\mathbf{A})} \frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|} \leq \frac{\|\mathbf{x}^{(l)} - \mathbf{x}^*\|}{\|\mathbf{x}^{(0)} - \mathbf{x}^*\|} \leq \text{cond}(\mathbf{A}) \frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|}. \quad (5.2.20)$$

↑  
relative decrease of iteration error

(5.2.20) can easily be deduced from the error equation  $\mathbf{A}(\mathbf{x}^{(k)} - \mathbf{x}^*) = \mathbf{r}_k$ , see Def. 2.5.16 and (2.5.19).



### 5.2.3 Convergence of CG

Note: CG is a *direct solver*, because (in exact arithmetic)  $\mathbf{x}^{(k)} = \mathbf{x}^*$  for some  $k \leq n$

Example 5.2.21 (Impact of roundoff errors on CG).  $\rightarrow$  [51, Rem. 4.3]

Numerical experiment:  $A = \text{hilb}(20)$ ,  
 $\mathbf{x}^{(0)} = \mathbf{0}$ ,  $\mathbf{b} = (1, \dots, 1)^T$

Hilbert-Matrix: extremely ill-conditioned

residual norms during CG iteration

$$\mathbf{R} = [\mathbf{r}_0, \dots, \mathbf{r}^{(10)}]$$

$\triangleright$ :

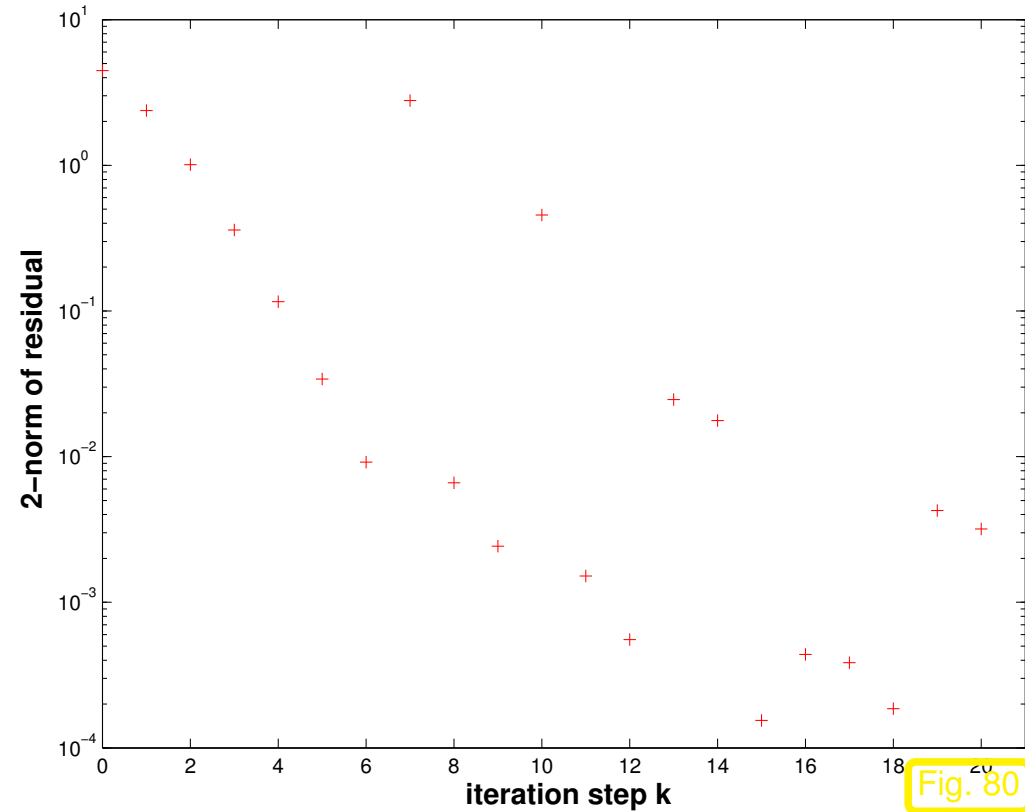


Fig. 80

$$\mathbf{R}^T \mathbf{R} =$$

1.000000	-0.000000	0.000000	-0.000000	0.000000	-0.000000	0.016019	-0.795816	-0.430569	0.348133
-0.000000	1.000000	-0.000000	0.000000	-0.000000	0.000000	-0.012075	0.600068	-0.520610	0.420903
0.000000	-0.000000	1.000000	-0.000000	0.000000	-0.000000	0.001582	-0.078664	0.384453	-0.310577
-0.000000	0.000000	-0.000000	1.000000	-0.000000	0.000000	-0.000024	0.001218	-0.024115	0.019394
0.000000	-0.000000	0.000000	-0.000000	1.000000	-0.000000	0.000000	-0.000002	0.000151	-0.000118
-0.000000	0.000000	-0.000000	0.000000	-0.000000	1.000000	-0.000000	0.000000	-0.000000	0.000000
0.016019	-0.012075	0.001582	-0.000024	0.000000	-0.000000	1.000000	-0.000000	-0.000000	0.000000
-0.795816	0.600068	-0.078664	0.001218	-0.000002	0.000000	-0.000000	1.000000	0.000000	-0.000000
-0.430569	-0.520610	0.384453	-0.024115	0.000151	-0.000000	-0.000000	0.000000	1.000000	0.000000
0.348133	0.420903	-0.310577	0.019394	-0.000118	0.000000	0.000000	-0.000000	0.000000	1.000000

- Roundoff
  - destroys orthogonality of residuals
  - prevents computation of exact solution after  $n$  steps.

▶ Numerical instability ( $\rightarrow$  Def. 2.5.11) ➤ pointless to (try to) use CG as direct solver!





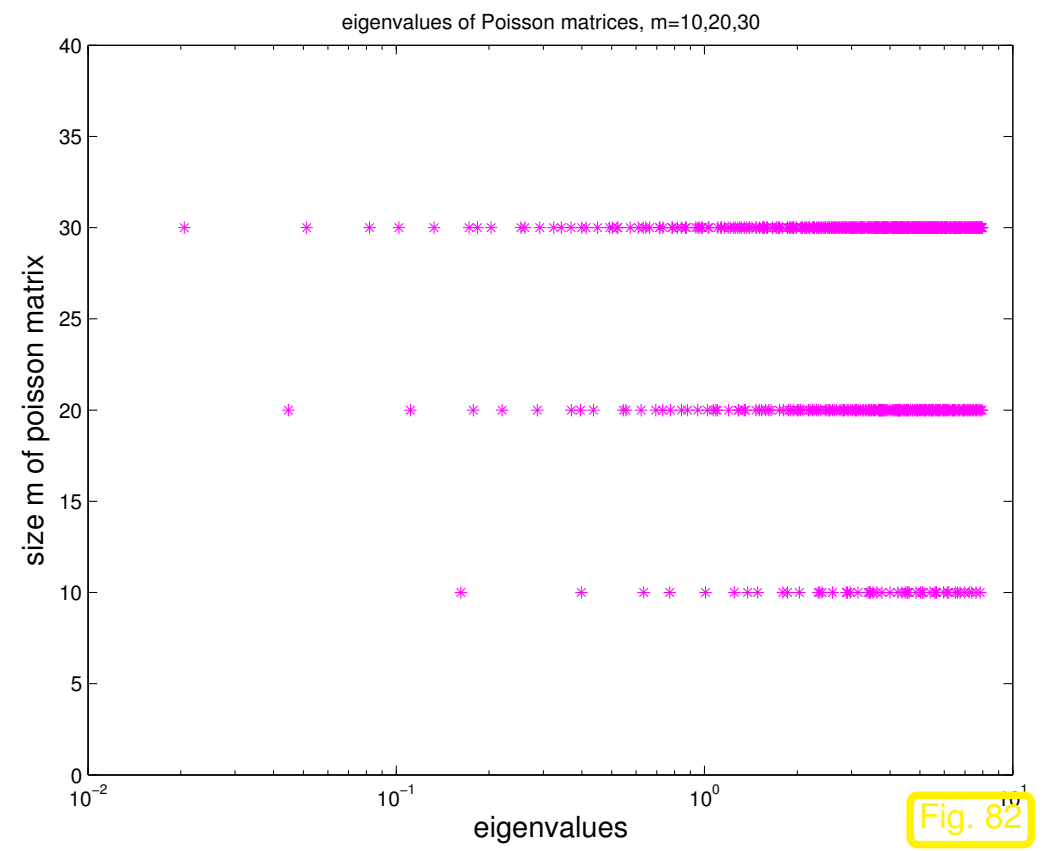
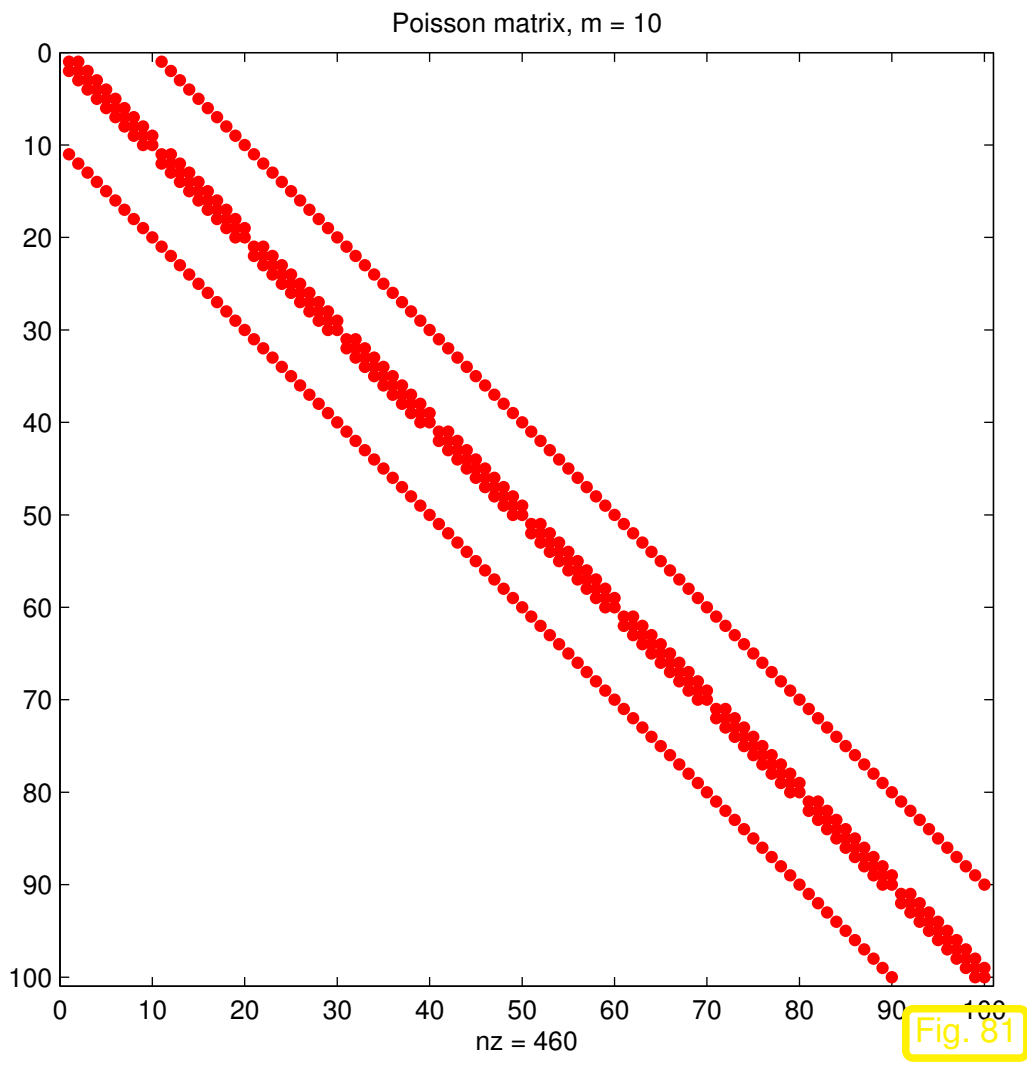
Practice: CG used for large  $n$  as *iterative solver*:  $\mathbf{x}^{(k)}$  for some  $k \ll n$  is expected to provide good approximation for  $\mathbf{x}^*$

*Example 5.2.22* (Convergence of CG as iterative solver).

CG (Code 5.2.17) & gradient method (Code 5.1.11) for LSE with sparse s.p.d. “Poisson matrix”

```
A = gallery('poisson',m); x0 = (1:n)'; b = zeros(n,1);
```

➤  $\mathbf{A} \in \mathbb{R}^{m^2, m^2}$



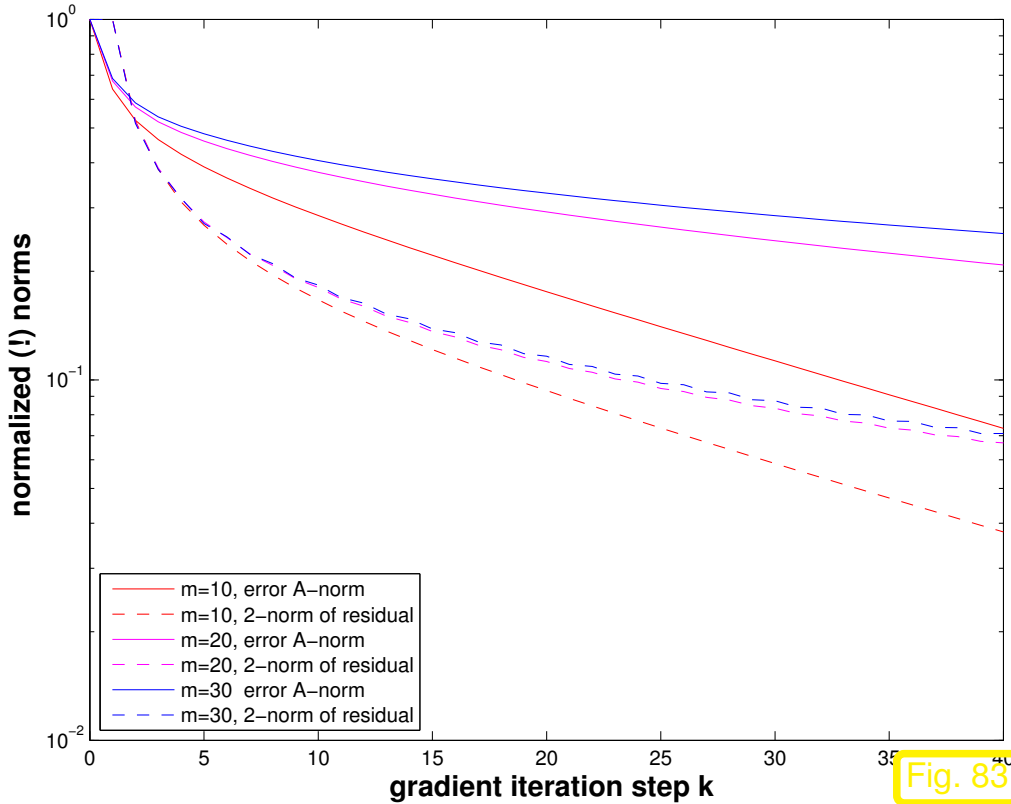


Fig. 83

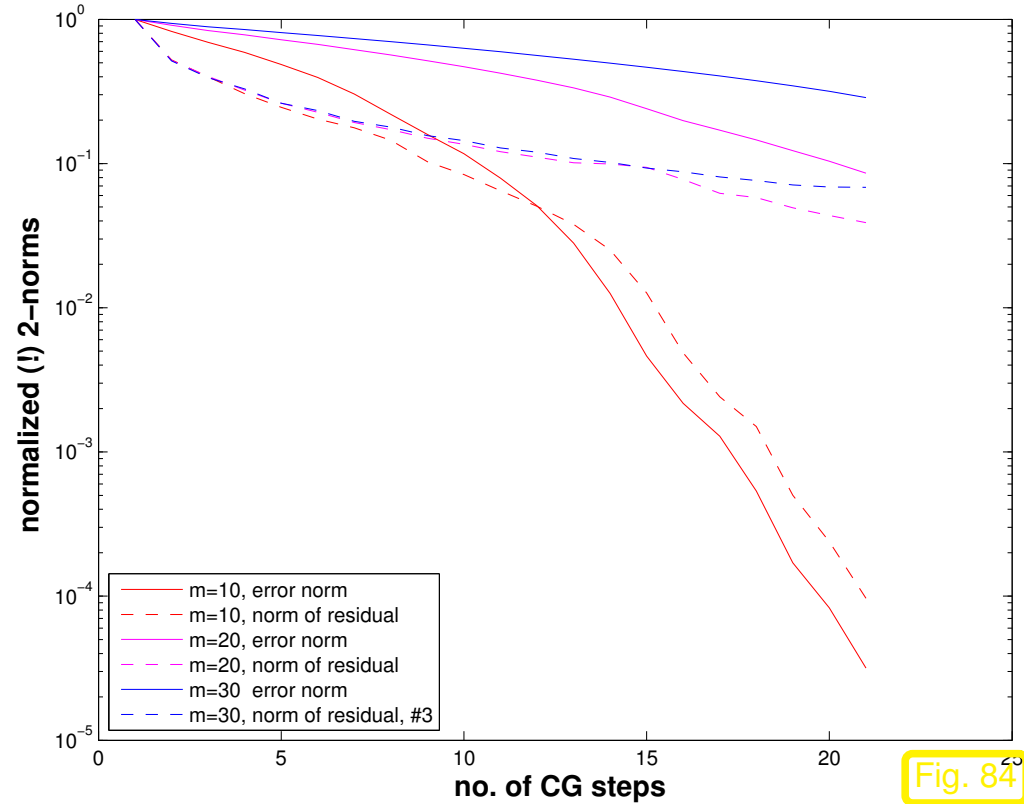


Fig. 84

## Observations:

- CG much faster than gradient method (as expected, because it has “memory”)
- Both, CG and gradient method converge more slowly for larger sizes of Poisson matrices.

**Convergence theory:** [29, Sect. 9.4.3]

A simple consequence of (5.1.5) and (5.2.1):

**Corollary 5.2.23** (“Optimality” of CG iterates).

Writing  $\mathbf{x}^* \in \mathbb{R}^n$  for the exact solution of  $\mathbf{Ax} = \mathbf{b}$  the CG iterates satisfy

$$\left\| \mathbf{x}^* - \mathbf{x}^{(l)} \right\|_A = \min \{ \|\mathbf{y} - \mathbf{x}^*\|_A : \mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0) \} \quad , \quad \mathbf{r}_0 := \mathbf{b} - \mathbf{Ax}^{(0)} .$$

This paves the way for a quantitative convergence estimate:

$$\mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}) \Leftrightarrow \mathbf{y} = \mathbf{x}^{(0)} + \mathbf{A} p(\mathbf{A})(\mathbf{x} - \mathbf{x}^{(0)}) \quad , \quad p = \text{polynomial of degree } \leq l - 1 .$$

$$\blacktriangleright \quad \mathbf{x} - \mathbf{y} = q(\mathbf{A})(\mathbf{x} - \mathbf{x}^{(0)}) \quad , \quad q = \text{polynomial of degree } \leq l \quad , \quad q(0) = 1 .$$

$$\|\mathbf{x} - \mathbf{x}^{(l)}\|_A \leq \min\left\{ \max_{\lambda \in \sigma(\mathbf{A})} |q(\lambda)| : q \text{ polynomial of degree } \leq l, \quad q(0) = 1 \right\} \cdot \|\mathbf{x} - \mathbf{x}^{(0)}\|_A .$$

(5.2.24)

Bound this minimum for  $\lambda \in [\lambda_{\min}(\mathbf{A}), \lambda_{\max}(\mathbf{A})]$  by using suitable “polynomial candidates”

Tool: **Chebyshev polynomials** ( $\rightarrow$  Sect. 9.2.1)  $\triangleright$  lead to the following estimate [28, Satz 9.4.2], [13, Satz 13.29]

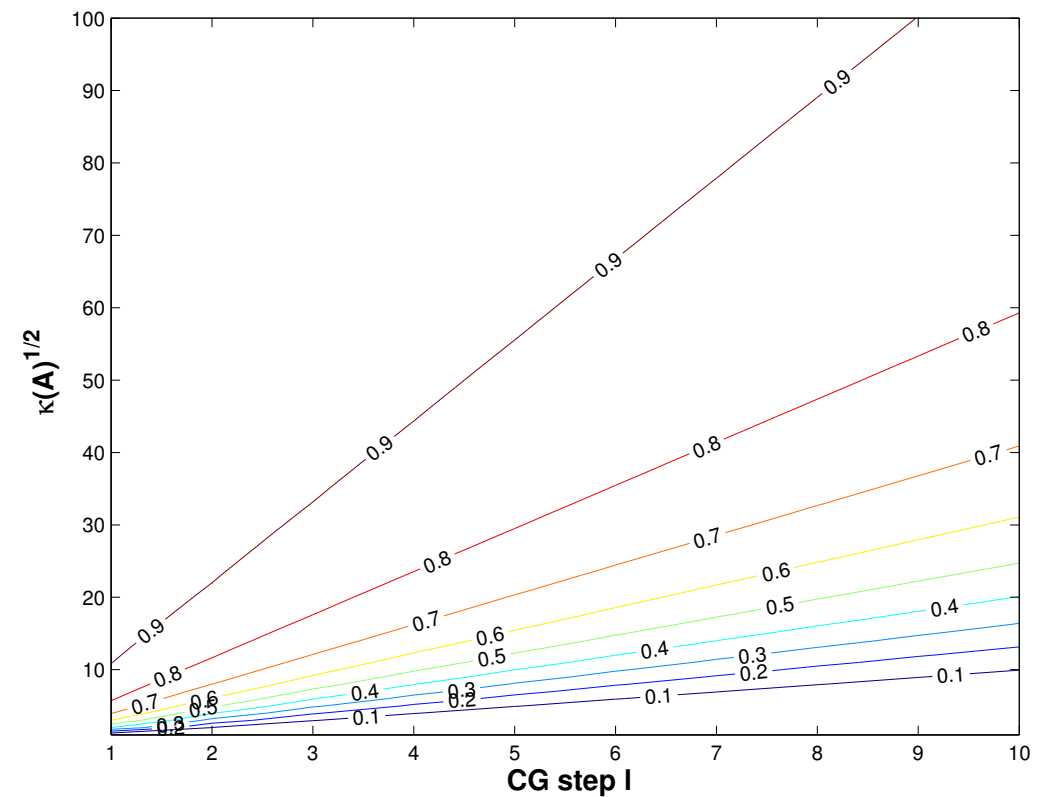
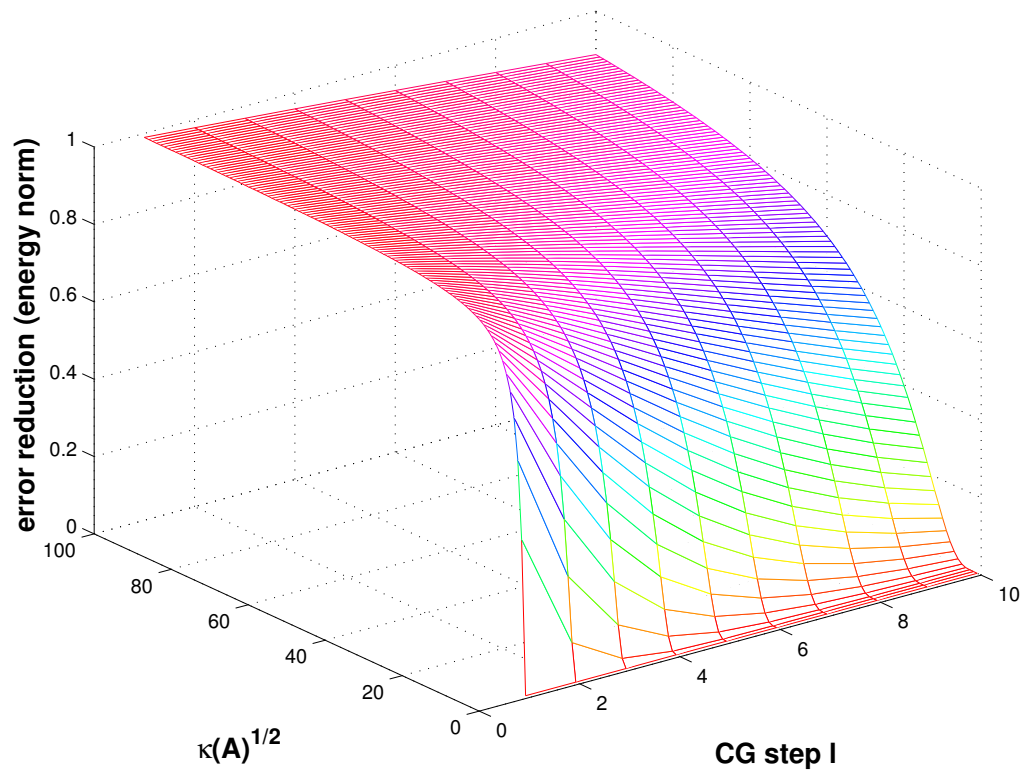
**Theorem 5.2.25** (Convergence of CG method).

The iterates of the CG method for solving  $\mathbf{Ax} = \mathbf{b}$  (see Code 5.2.17) with  $\mathbf{A} = \mathbf{A}^\top$  s.p.d. satisfy

$$\begin{aligned} \|\mathbf{x} - \mathbf{x}^{(l)}\|_A &\leq \frac{2 \left(1 - \frac{1}{\sqrt{\kappa(\mathbf{A})}}\right)^l}{\left(1 + \frac{1}{\sqrt{\kappa(\mathbf{A})}}\right)^{2l} + \left(1 - \frac{1}{\sqrt{\kappa(\mathbf{A})}}\right)^{2l}} \|\mathbf{x} - \mathbf{x}^{(0)}\|_A \\ &\leq 2 \left(\frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1}\right)^l \|\mathbf{x} - \mathbf{x}^{(0)}\|_A. \end{aligned}$$

(recall:  $\kappa(\mathbf{A}) = \text{spectral condition number of } \mathbf{A}$ ,  $\kappa(\mathbf{A}) = \text{cond}_2(\mathbf{A})$ )

The estimate of this theorem confirms *asymptotic linear convergence* of the CG method ( $\rightarrow$  Def. 4.1.6) with a rate of  $\frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1}$



Code 5.2.26: plotting theoretical bounds for CG convergence rate

```

1 function plottheorate
2 % Python: ../PYTHON/ConjugateGradient.py plottheorate()
3
4 [X,Y] = meshgrid(1:10,1:100); R = zeros(100,10);
5 for I=1:100
6     t = 1/I;
7     for j=1:10

```

```
8     R(I, j) = 2*(1-t)^j/((1+t)^(2*j)+(1-t)^(2*j));
9     end
10 end
11
12 figure; view([-45,28]); mesh(X,Y,R); colormap hsv;
13 xlabel('\bf CG step 1','FontSize',14);
14 ylabel('\bf \kappa(A)^{1/2}','FontSize',14);
15 zlabel('\bf error reduction (energy norm)','FontSize',14);
16
17 print -depsc2 '../PICTURES/theorate1.eps';
18
19 figure; [C,h] = contour(X,Y,R); clabel(C,h);
20 xlabel('\bf CG step 1','FontSize',14);
21 ylabel('\bf \kappa(A)^{1/2}','FontSize',14);
22
23 print -depsc2 '../PICTURES/theorate2.eps';
```

*Example 5.2.27* (Convergence rates for CG method).



## Code 5.2.28: CG for Poisson matrix

## Measurement

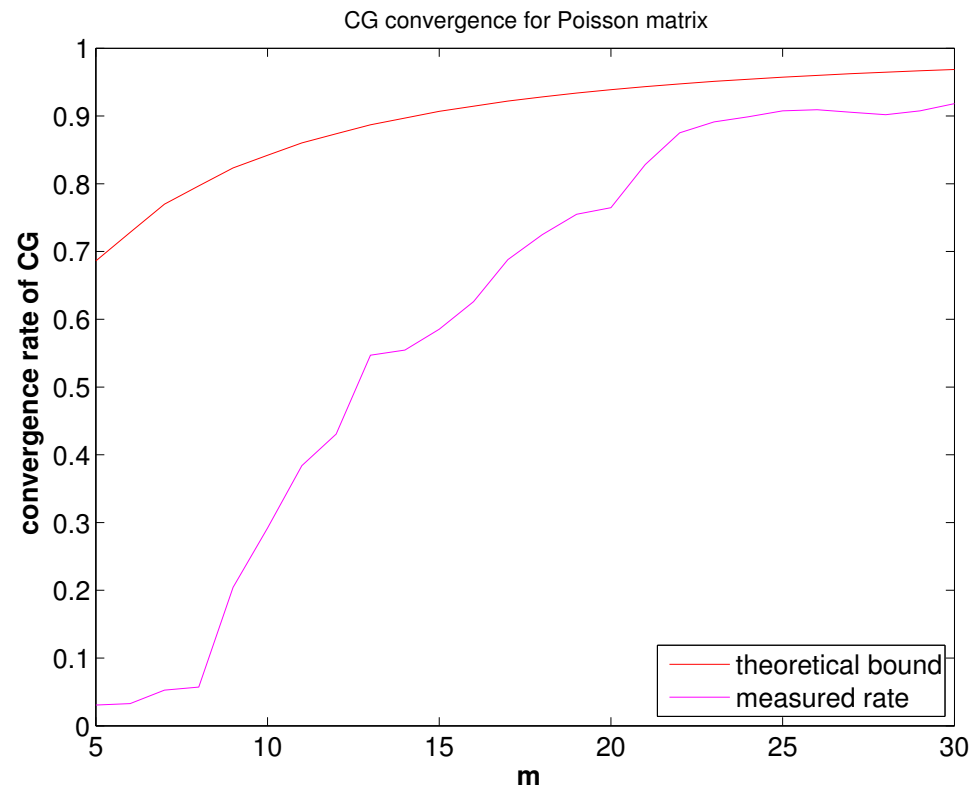
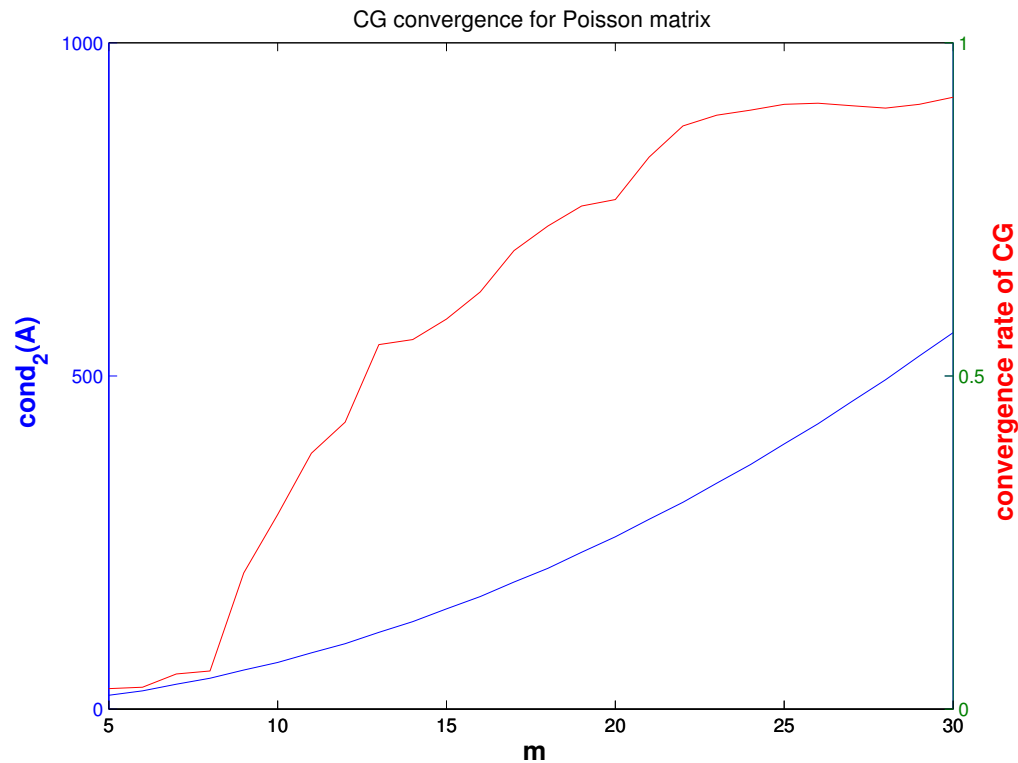
```

1 A = gallery('poisson',m); n = size(A,1);
2 x0 = (1:n)'; b = ones(n,1); maxit = 30;
   tol = 0;
3 [x, flag, relres, iter, resvec] =
   pcg(A,b,tol,maxit,[],[],x0);

```

rate of (linear) convergence:

$$\text{rate} \approx \sqrt[10]{\frac{\|\mathbf{r}_{30}\|_2}{\|\mathbf{r}_{20}\|_2}}. \quad (5.2.29)$$



Justification for estimating the rate of linear convergence ( $\rightarrow$  Def. 4.1.6) of  $\|\mathbf{r}_k\|_2 \rightarrow 0$ :

$$\|\mathbf{r}_{k+1}\|_2 \approx L \|\mathbf{r}_k\|_2 \quad \Rightarrow \quad \|\mathbf{r}_{k+m}\|_2 \approx L^m \|\mathbf{r}_k\|_2 .$$



*Example 5.2.30* (CG convergence and spectrum).  $\rightarrow$  Ex. 5.1.17

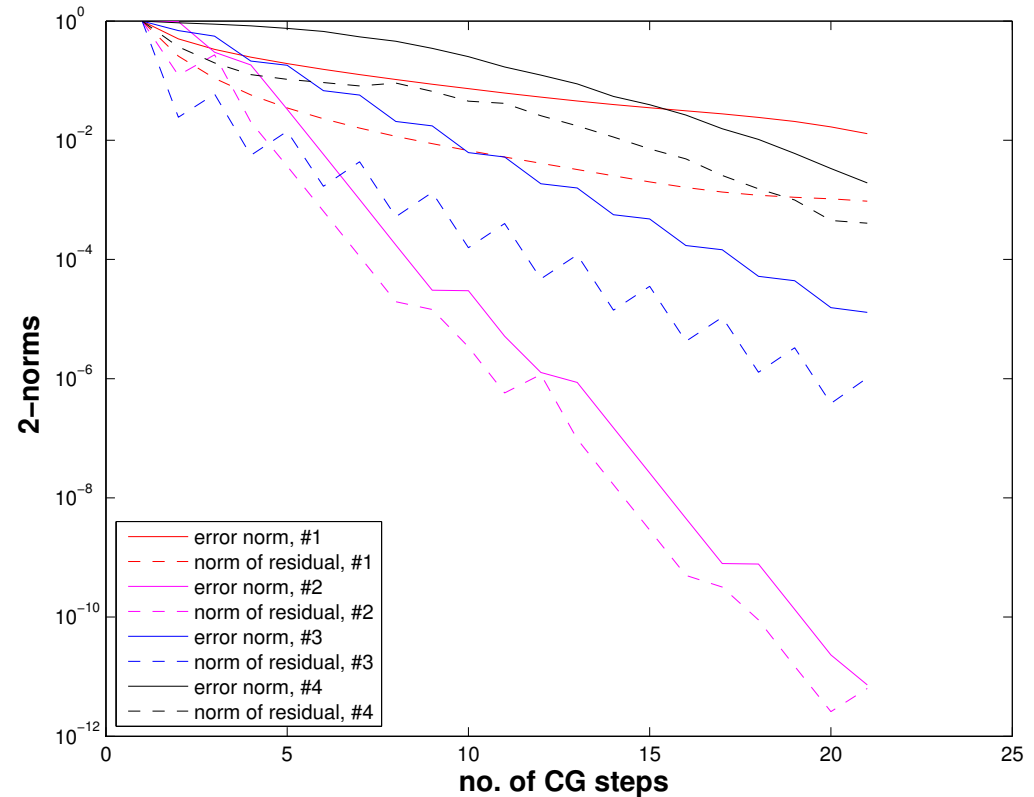
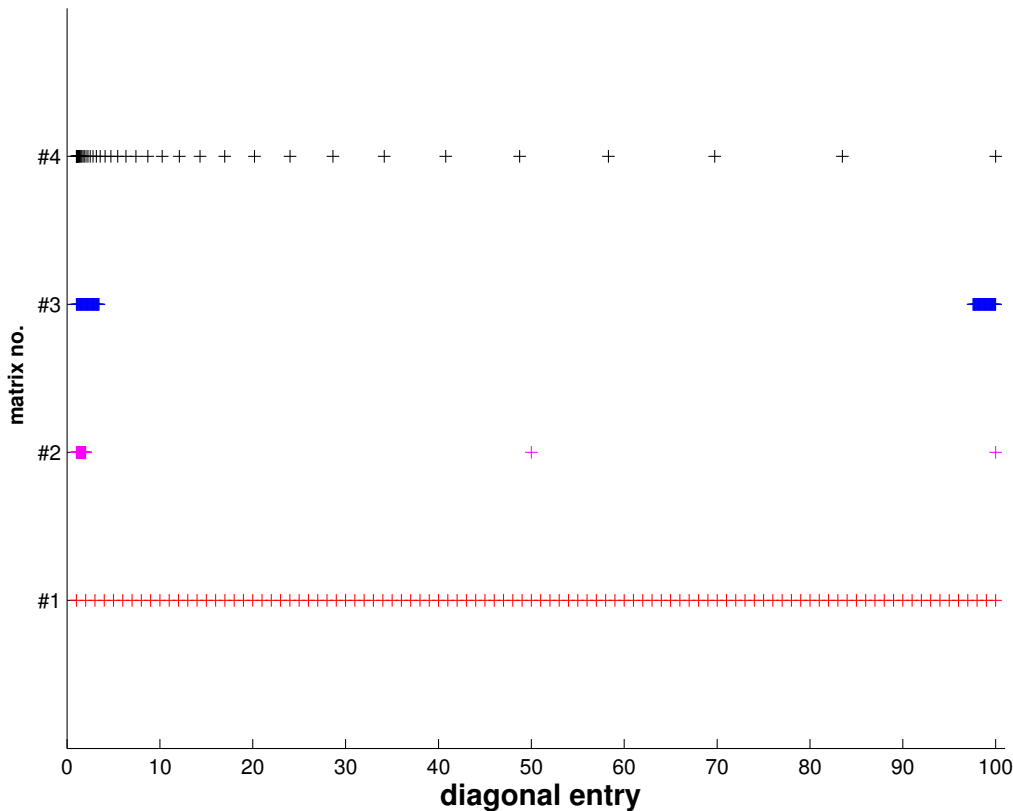
Test matrix **#1**:  $A = \text{diag}(d)$ ;  $d = (1:100)$ ;

Test matrix **#2**:  $A = \text{diag}(d)$ ;  $d = [1 + (0:97)/97, 50, 100]$ ;

Test matrix **#3**:  $A = \text{diag}(d)$ ;  $d = [1 + (0:49) * 0.05, 100 - (0:49) * 0.05]$ ;

Test matrix **#4**: eigenvalues exponentially dense at 1

$x_0 = \cos((1:n)')$ ;  $b = \text{zeros}(n, 1)$ ;



Observations: Distribution of eigenvalues has crucial impact on convergence of CG  
 (This is clear from the convergence theory, because detailed information about the spectrum allows a much better choice of “candidate polynomial” in (5.2.24) than merely using Chebychev polynomials)

➤ Clustering of eigenvalues leads to faster convergence of CG  
 (in stark contrast to the behavior of the gradient method, see Ex. 5.1.17)



## 5.3 Preconditioning [13, Sect. 13.5], [35, Ch. 10], [51, Sect. 4.3.5]

Thm. 5.2.25  $\triangleright$  (Potentially) slow convergence of CG in case  $\kappa(\mathbf{A}) \gg 1$ .

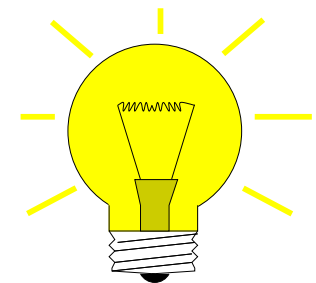
Idea:

Preconditioning

Apply CG method to transformed linear system

$$\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \quad , \quad \tilde{\mathbf{A}} := \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2} \quad , \quad \tilde{\mathbf{x}} := \mathbf{B}^{1/2}\mathbf{x} \quad , \quad \tilde{\mathbf{b}} := \mathbf{B}^{-1/2}\mathbf{b} \quad , \quad (5.3.1)$$

with “small”  $\kappa(\tilde{\mathbf{A}})$ ,  $\mathbf{B} = \mathbf{B}^T \in \mathbb{R}^{N,N}$  s.p.d.  $\hat{=}$  preconditioner.



*Remark 5.3.2* (Square root of a s.p.d. matrix).

What is meant by the “square root”  $\mathbf{B}^{1/2}$  of a s.p.d. matrix  $\mathbf{B}$  ?

Recall (5.1.15) : for every  $\mathbf{B} \in \mathbb{R}^{n,n}$  with  $\mathbf{B}^\top = \mathbf{B}$  there is an orthogonal matrix  $\mathbf{Q} \in \mathbb{R}^{n,n}$  such that  $\mathbf{B} = \mathbf{Q}^\top \mathbf{D} \mathbf{Q}$  with a diagonal matrix  $\mathbf{D}$  ( $\rightarrow$  Cor. 6.1.9, [48, Thm. 7.8], [41, Satz 9.15]). If  $\mathbf{B}$  is s.p.d. the (diagonal) entries of  $\mathbf{D}$  are strictly positive and we can define

$$\mathbf{D} = \text{diag}(\lambda_1, \dots, \lambda_n), \quad \lambda_i > 0 \quad \Rightarrow \quad \mathbf{D}^{1/2} := \text{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_n}) .$$

This is generalized to

$$\mathbf{B}^{1/2} := \mathbf{Q}^\top \mathbf{D}^{1/2} \mathbf{Q} ,$$

and one easily verifies, using  $\mathbf{Q}^\top = \mathbf{Q}^{-1}$ , that  $(\mathbf{B}^{1/2})^2 = \mathbf{B}$  and that  $\mathbf{B}^{1/2}$  is s.p.d. In fact, these two requirements already determine  $\mathbf{B}^{1/2}$  uniquely:

$\mathbf{B}^{1/2}$  is the unique s.p.d. matrix such that  $(\mathbf{B}^{1/2})^2 = \mathbf{B}$ .

**Notion 5.3.3** (Preconditioner).

A s.p.d. matrix  $\mathbf{B} \in \mathbb{R}^{n,n}$  is called a *preconditioner* (ger.: Vorkonditionierer) for the s.p.d. matrix  $\mathbf{A} \in \mathbb{R}^{n,n}$ , if

1.  $\kappa(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2})$  is “small” and
2. the evaluation of  $\mathbf{B}^{-1}\mathbf{x}$  is about as expensive (in terms of elementary operations) as the matrix×vector multiplication  $\mathbf{A}\mathbf{x}$ ,  $\mathbf{x} \in \mathbb{R}^n$ .

Recall: spectral condition number  $\kappa(\mathbf{A}) := \frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})}$ , see (5.1.21)

There are several equivalent ways to express that  $\kappa(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2})$  is “small”:

- $\kappa(\mathbf{B}^{-1}\mathbf{A})$  is “small”,  
because spectra agree  $\sigma(\mathbf{B}^{-1}\mathbf{A}) = \sigma(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2})$  due to similarity ( $\rightarrow$  Lemma 6.1.6)
- $\exists 0 < \gamma < \Gamma$ ,  $\Gamma/\gamma$  “small”:  $\gamma(\mathbf{x}^\top \mathbf{B}\mathbf{x}) \leq \mathbf{x}^\top \mathbf{A}\mathbf{x} \leq \Gamma(\mathbf{x}^\top \mathbf{B}\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^n$ ,

where equivalence is seen by transforming  $\mathbf{y} := \mathbf{B}^{-1/2}\mathbf{x}$  and appealing to the min-max Theorem 6.3.39.

“Reader’s digest” version of notion 5.3.3:

S.p.d.  $\mathbf{B}$  preconditioner  $:\Leftrightarrow \mathbf{B}^{-1}$  = cheap approximate inverse of  $\mathbf{A}$

Problem:  $\mathbf{B}^{1/2}$ , which occurs prominently in (5.3.1) is usually not available with acceptable computational costs.

However, if one formally applies Algorithm 5.2.17 to the transformed system

$$\tilde{\mathbf{A}}\tilde{\mathbf{x}} := \left(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\right) (\mathbf{B}^{1/2}\mathbf{x}) = \tilde{\mathbf{b}} := \mathbf{B}^{-1/2}\mathbf{b} ,$$

from (5.3.1), it becomes apparent that, after suitable transformation of the iteration variables  $\mathbf{p}_j$  and  $\mathbf{r}_j$ ,  $\mathbf{B}^{1/2}$  and  $\mathbf{B}^{-1/2}$  invariably occur in products  $\mathbf{B}^{-1/2}\mathbf{B}^{-1/2} = \mathbf{B}^{-1}$  and  $\mathbf{B}^{1/2}\mathbf{B}^{-1/2} = \mathbf{I}$ . Thus, thanks to this **intrinsic transformation** square roots of  $\mathbf{B}$  are not required for the implementation!

CG for  $\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ 

Input : initial guess  $\tilde{\mathbf{x}}^{(0)} \in \mathbb{R}^n$

Output : approximate solution  $\tilde{\mathbf{x}}^{(l)} \in \mathbb{R}^n$

$$\tilde{\mathbf{p}}_1 := \tilde{\mathbf{r}}_0 := \tilde{\mathbf{b}} - \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\tilde{\mathbf{x}}^{(0)};$$

for  $j = 1$  to  $l$  do {

$$\alpha := \frac{\tilde{\mathbf{p}}_j^T \tilde{\mathbf{r}}_{j-1}}{\tilde{\mathbf{p}}_j^T \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j}$$

$$\tilde{\mathbf{x}}^{(j)} := \tilde{\mathbf{x}}^{(j-1)} + \alpha \tilde{\mathbf{p}}_j;$$

$$\tilde{\mathbf{r}}_j = \tilde{\mathbf{r}}_{j-1} - \alpha \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{1/2}\tilde{\mathbf{p}}_j;$$

$$\tilde{\mathbf{p}}_{j+1} = \tilde{\mathbf{r}}_j - \frac{(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j)^T \tilde{\mathbf{r}}_j}{\tilde{\mathbf{p}}_j^T \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j} \tilde{\mathbf{p}}_j;$$

}

## Equivalent CG with transformed variables

Input : initial guess  $\mathbf{x}^{(0)} \in \mathbb{R}^n$

Output : approximate solution  $\mathbf{x}^{(l)} \in \mathbb{R}^n$

$$\mathbf{B}^{1/2}\tilde{\mathbf{r}}_0 := \mathbf{B}^{1/2}\tilde{\mathbf{b}} - \mathbf{A}\mathbf{B}^{-1/2}\tilde{\mathbf{x}}^{(0)};$$

$$\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_1 := \mathbf{B}^{-1}(\mathbf{B}^{1/2}\tilde{\mathbf{r}}_0);$$

for  $j = 1$  to  $l$  do {

$$\alpha := \frac{(\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j)^T \mathbf{B}^{1/2}\tilde{\mathbf{r}}_{j-1}}{(\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j)^T \mathbf{A}\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j}$$

$$\mathbf{B}^{-1/2}\tilde{\mathbf{x}}^{(j)} := \mathbf{B}^{-1/2}\tilde{\mathbf{x}}^{(j-1)} + \alpha \mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j;$$

$$\mathbf{B}^{1/2}\tilde{\mathbf{r}}_j = \mathbf{B}^{1/2}\tilde{\mathbf{r}}_{j-1} - \alpha \mathbf{A}\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j;$$

$$\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_{j+1} = \mathbf{B}^{-1}(\mathbf{B}^{-1/2}\tilde{\mathbf{r}}_j) - \frac{(\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j)^T \mathbf{A}\mathbf{B}^{-1}(\mathbf{B}^{1/2}\tilde{\mathbf{r}}_j)}{(\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j)^T \mathbf{A}\mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j} \mathbf{B}^{-1/2}\tilde{\mathbf{p}}_j;$$

}



with the transformations:

$$\tilde{\mathbf{x}}^{(k)} = \mathbf{B}^{1/2} \mathbf{x}^{(k)} \quad , \quad \tilde{\mathbf{r}}_k = \mathbf{B}^{-1/2} \mathbf{r}_k \quad , \quad \tilde{\mathbf{p}}_k = \mathbf{B}^{-1/2} \mathbf{p}_k . \quad (5.3.4)$$

*Algorithm 5.3.5 (Preconditioned CG method (PCG)).* [13, Alg. 13.32], [35, Alg. 10.1]]

Input: initial guess  $\mathbf{x} \in \mathbb{R}^n \hat{=} \mathbf{x}^{(0)} \in \mathbb{R}^n$ , tolerance  $\tau > 0$

Output: approximate solution  $\mathbf{x} \hat{=} \mathbf{x}^{(l)}$

$\mathbf{p} := \mathbf{r} := \mathbf{b} - \mathbf{A}\mathbf{x}; \quad \mathbf{p} := \mathbf{B}^{-1}\mathbf{r}; \quad \mathbf{q} := \mathbf{p}; \quad \tau_0 := \mathbf{p}^\top \mathbf{r};$

**for**  $l = 1$  **to**  $l_{\max}$  **do** {

$\beta := \mathbf{r}^\top \mathbf{q}; \quad \mathbf{h} := \mathbf{A}\mathbf{p}; \quad \alpha := \frac{\beta}{\mathbf{p}^\top \mathbf{h}};$

$\mathbf{x} := \mathbf{x} + \alpha \mathbf{p};$

$\mathbf{r} := \mathbf{r} - \alpha \mathbf{h};$

$\mathbf{q} := \mathbf{B}^{-1}\mathbf{r}; \quad \beta := \frac{\mathbf{r}^\top \mathbf{q}}{\beta};$

**if**  $|\mathbf{q}^\top \mathbf{r}| \leq \tau \cdot \tau_0$  **then stop**;

$\mathbf{p} := \mathbf{q} + \beta \mathbf{p};$

}

(5.3.6)

## Code 5.3.7: simple implementation of PCG algorithm Alg. 5.3.5

```
1 function [x, rn, xk] = pcgbase(evalA, b, tol, maxit, invB, x)
2 % evalA must pass a handle to a function implementing A*x
3 % invB is to be a handle to a function providing the action of the
4 % preconditioner on a vector. The other arguments like for MATLAB's pcg.
5 r = b - evalA(x); rho = 1; rn = [];
6 if (nargout > 2), xk = x; end
7 for i = 1 : maxit
8     y = invB(r);
9     rho_old = rho; rho = r' * y; rn = [rn, rho];
10    if (i == 1), p = y; rho0 = rho;
11    elseif (rho < rho0*tol), return;
12    else beta = rho/rho_old; p = y+beta*p; end
13    q = evalA(p); alpha = rho / (p' * q);
14    x = x + alpha * p;
15    r = r - alpha * q;
16    if (nargout > 2), xk = [xk, x]; end
17 end
```

Computational effort per step: 1 evaluation  $\mathbf{A} \times \text{vector}$ ,  
1 evaluation  $\mathbf{B}^{-1} \times \text{vector}$ ,  
3 dot products, 3 **AXPY-operations**

*Remark 5.3.8* (Convergence theory for PCG).

Assertions of Thm. 5.2.25 remain valid with  $\kappa(\mathbf{A})$  replaced with  $\kappa(\mathbf{B}^{-1}\mathbf{A})$  and energy norm based on  $\tilde{\mathbf{A}}$  instead of  $\mathbf{A}$ .



*Example 5.3.9* (Simple preconditioners).

$\mathbf{B}$  = easily invertible “part” of  $\mathbf{A}$

- $\mathbf{B} = \text{diag}(\mathbf{A})$ : **Jacobi preconditioner** (diagonal scaling)

- $(\mathbf{B})_{ij} = \begin{cases} (\mathbf{A})_{ij} & , \text{ if } |i - j| \leq k, \\ 0 & \text{ else,} \end{cases}$  for some  $k \ll n$ .

- **Symmetric Gauss-Seidel preconditioner**

Idea: Solve  $\mathbf{Ax} = \mathbf{b}$  approximately in two stages:

① Approximation  $\mathbf{A}^{-1} \approx \text{tril}(\mathbf{A})$  (lower triangular part):  $\tilde{\mathbf{x}} = \text{tril}(\mathbf{A})^{-1} \mathbf{b}$

② Approximation  $\mathbf{A}^{-1} \approx \text{triu}(\mathbf{A})$  (upper triangular part) and use this to approximately “solve” the **error equation**  $\mathbf{A}(\mathbf{x} - \tilde{\mathbf{x}}) = \mathbf{r}$ , with **residual**  $\mathbf{r} := \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$ :

$$\mathbf{x} = \tilde{\mathbf{x}} + \text{triu}(\mathbf{A})^{-1}(\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}) .$$

With  $\mathbf{L}_A := \text{tril}(\mathbf{A})$ ,  $\mathbf{U}_A := \text{triu}(\mathbf{A})$  one finds

$$\mathbf{x} = (\mathbf{L}_A^{-1} + \mathbf{U}_A^{-1} - \mathbf{U}_A^{-1} \mathbf{A} \mathbf{L}_A^{-1}) \mathbf{b} \quad \blacktriangleright \quad \mathbf{B}^{-1} = \mathbf{L}_A^{-1} + \mathbf{U}_A^{-1} - \mathbf{U}_A^{-1} \mathbf{A} \mathbf{L}_A^{-1} . \quad (5.3.10)$$

For all these approaches the evaluation of  $\mathbf{B}^{-1} \mathbf{r}$  can be done with effort of  $O(n)$  in the case of a sparse matrix  $\mathbf{A}$  (e.g. with  $O(1)$  non-zero entries per row). However, there is absolutely no guarantee that  $\kappa(\mathbf{B}^{-1} \mathbf{A})$  will be reasonably small. It will crucially depend on  $\mathbf{A}$ , if this can be expected.



More complicated preconditioning strategies:

- **Incomplete Cholesky factorization**, MATLAB-`ichol`, [13, Sect. 13.5]
- **Sparse approximate inverse preconditioner (SPAI)**

*Example 5.3.11* (Tridiagonal preconditioning).

Efficacy of preconditioning of sparse LSE with tridiagonal part:

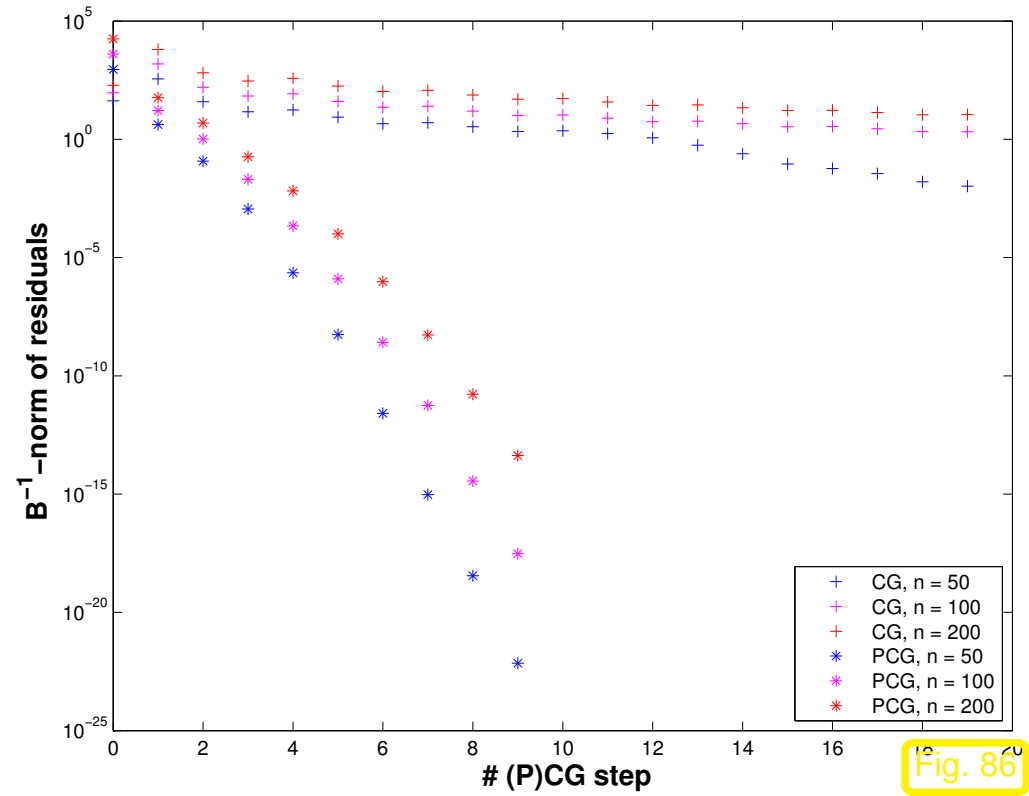
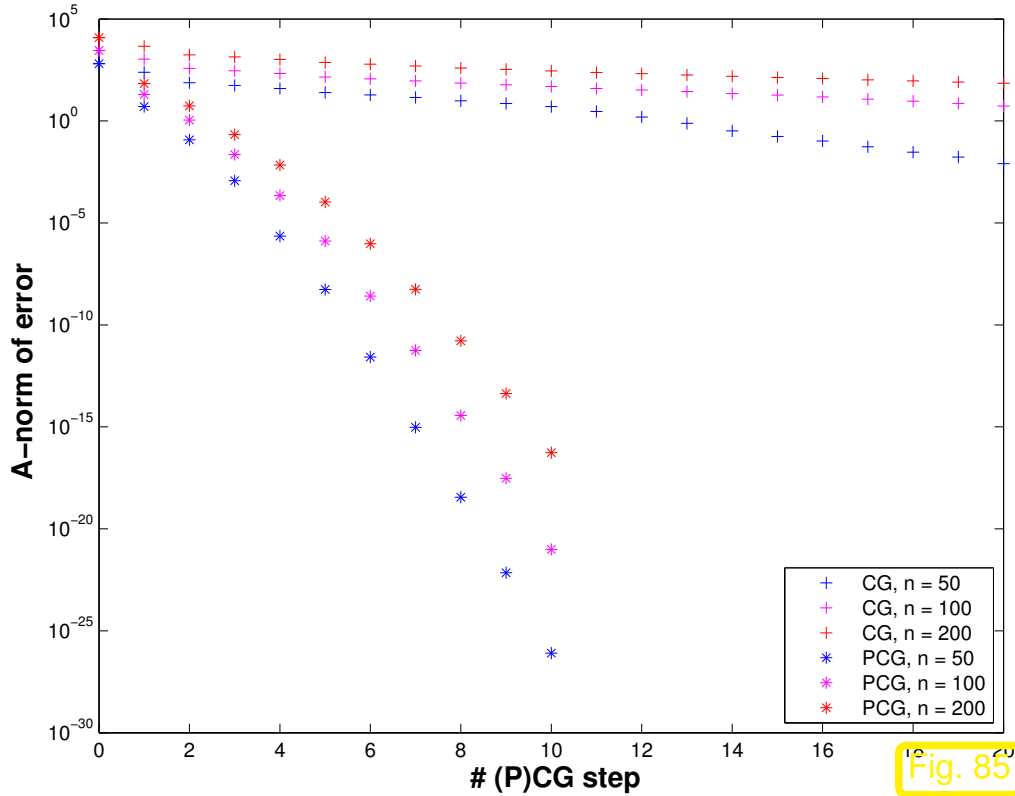
Code 5.3.12: LSE for Ex. 5.3.11

```

1 A =
   spdiags(repmat([1/n, -1, 2+2/n, -1, 1/n], n, 1), [-n/2, -1, 0, 1, n/2], n, n);
2 b = ones(n, 1); x0 = ones(n, 1); tol = 1.0E-4; maxit = 1000;
3 evalA = @(x) A*x;
4
5 % no preconditioning, see Code 5.3.6
6 invB = @(x) x; [x, rn] = pcgbase(evalA, b, tol, maxit, invB, x0);
7
8 % tridiagonal preconditioning, see Code 5.3.6
9 B = spdiags(spdiags(A, [-1, 0, 1]), [-1, 0, 1], n, n);
10 invB = @(x) B\x; [x, rnpc] = pcgbase(evalA, b, tol, maxit, invB, x0); %

```

The Code 5.3.11 highlights the use of a preconditioner in the context of the PCG method; it only takes a function that realizes the application of  $B^{-1}$  to a vector. In Line 10 of the code this function is passed as function handle `invB`.



$n$	# CG steps	# PCG steps
16	8	3
32	16	3
64	25	4
128	38	4
256	66	4
512	106	4
1024	149	4
2048	211	4
4096	298	3
8192	421	3
16384	595	3
32768	841	3

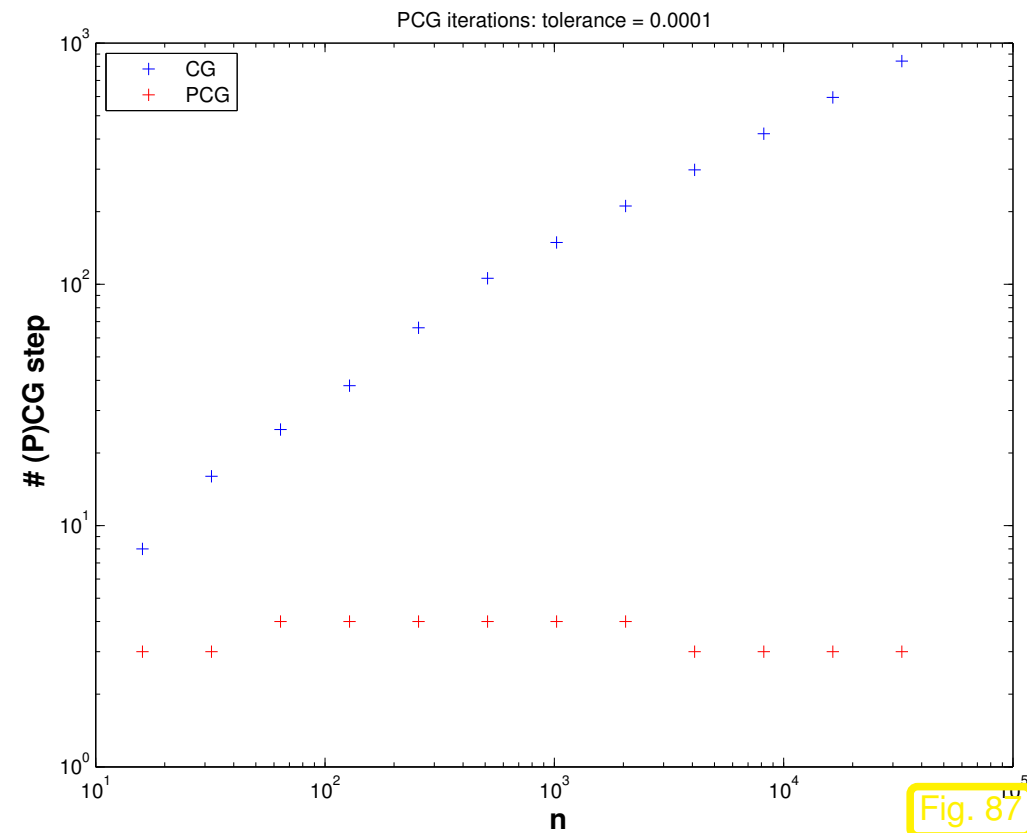


Fig. 87

Clearly in this example the tridiagonal part of the matrix is dominant for large  $n$ . In addition, its condition number grows  $\sim n^2$  as is revealed by a closer inspection of the spectrum.

Preconditioning with the tridiagonal part manages to suppress this growth of the condition number of  $B^{-1}A$  and ensures fast convergence of the preconditioned CG method



Remark 5.3.13 (Termination of PCG).

Recall Rem. 5.2.19, (5.2.20):

$$\frac{1}{\text{cond}(\mathbf{A})} \frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|} \leq \frac{\|\mathbf{x}^{(l)} - \mathbf{x}^*\|}{\|\mathbf{x}^{(0)} - \mathbf{x}^*\|} \leq \text{cond}(\mathbf{A}) \frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|}. \quad (5.2.20)$$

**B** good preconditioner  $\blacktriangleright \text{cond}_2(\mathbf{B}^{-1/2} \mathbf{A} \mathbf{B}^{-1/2})$  small  $(\rightarrow \text{Notion. 5.3.3})$

Idea: consider (5.2.20) for

- Euclidean norm  $\|\cdot\| = \|\cdot\|_2 \leftrightarrow \text{cond}_2$
- transformed quantities  $\tilde{\mathbf{x}}, \tilde{\mathbf{r}}$ , see (5.3.1), (5.3.4)

Monitor 2-norm of transformed residual:

$$\tilde{\mathbf{r}} = \tilde{\mathbf{b}} - \tilde{\mathbf{A}}\tilde{\mathbf{x}} = \mathbf{B}^{-1/2} \mathbf{r} \Rightarrow \|\tilde{\mathbf{r}}\|_2^2 = \mathbf{r}^\top \mathbf{B}^{-1} \mathbf{r}.$$

(5.2.20)  $\blacktriangleright$  estimate for 2-norm of transformed iteration errors:  $\|\tilde{\mathbf{e}}^{(l)}\|_2^2 = (\mathbf{e}^{(l)})^\top \mathbf{B} \mathbf{e}^{(l)}$

Analogous to (5.2.20), estimates for energy norm  $(\rightarrow \text{Def. 5.1.1})$  of error  $\mathbf{e}^{(l)} := \mathbf{x} - \mathbf{x}^{(l)}$ ,  $\mathbf{x}^* := \mathbf{A}^{-1} \mathbf{b}$ :



Use error equation  $\mathbf{A}\mathbf{e}^{(l)} = \mathbf{r}_l$ :

$$\mathbf{r}_l^\top \mathbf{B}^{-1} \mathbf{r}_l = (\mathbf{B}^{-1} \mathbf{A} \mathbf{e}^{(l)})^\top \mathbf{A} \mathbf{e}^{(l)} \leq \lambda_{\max}(\mathbf{B}^{-1} \mathbf{A}) \left\| \mathbf{e}^{(l)} \right\|_A^2,$$

$$\left\| \mathbf{e}^{(l)} \right\|_A^2 = (\mathbf{A} \mathbf{e}^{(l)})^\top \mathbf{e}^{(l)} = \mathbf{r}_l^\top \mathbf{A}^{-1} \mathbf{r}_l = \mathbf{B}^{-1} \mathbf{r}_l^\top \mathbf{B} \mathbf{A}^{-1} \mathbf{r}_l \leq \lambda_{\max}(\mathbf{B} \mathbf{A}^{-1}) (\mathbf{B}^{-1} \mathbf{r}_l)^\top \mathbf{r}_l.$$



available during PCG iteration (5.3.6)

$$\frac{1}{\kappa(\mathbf{B}^{-1} \mathbf{A})} \frac{\left\| \mathbf{e}^{(l)} \right\|_A^2}{\left\| \mathbf{e}^{(0)} \right\|_A^2} \leq \frac{(\mathbf{B}^{-1} \mathbf{r}_l)^\top \mathbf{r}_l}{(\mathbf{B}^{-1} \mathbf{r}_0)^\top \mathbf{r}_0} \leq \kappa(\mathbf{B}^{-1} \mathbf{A}) \frac{\left\| \mathbf{e}^{(l)} \right\|_A^2}{\left\| \mathbf{e}^{(0)} \right\|_A^2} \quad (5.3.14)$$

$\kappa(\mathbf{B}^{-1} \mathbf{A})$  “small”  $\blacktriangleright$   $\mathbf{B}^{-1}$ -energy norm of residual  $\approx$   $\mathbf{A}$ -norm of error !  
( $\mathbf{r}_l \cdot \mathbf{B}^{-1} \mathbf{r}_l = \mathbf{q}^\top \mathbf{r}$  in Algorithm (5.3.6))

MATLAB-function: `[x, flag, relr, it, rv] = pcg(A, b, tol, maxit, B, [], x0);`  
( $A$ ,  $B$  may be handles to functions providing  $Ax$  and  $B^{-1}x$ , resp.)

*Remark 5.3.15* (Termination criterion in MATLAB-`pcg`).  $\rightarrow$  [51, Sect. 4.6]

Implementation (skeleton) of MATLAB built-in `pcg`:

```
function x = pcg(Afun,b,tol,maxit,Binvfun,x0)
x = x0; r = b - feval(Afun,x); rho = 1;
for i = 1 : maxit
    y = feval(Binvfun,r);
    rho1 = rho; rho = r' * y;
    if (i == 1)
        p = y;
    else
        beta = rho / rho1;
        p = y + beta * p;
    end
    q = feval(Afun,p);
    alpha = rho / (p' * q);
    x = x + alpha * p;
    if (norm(b - evalf(Afun,x)) <= tol*b*norm(b)) , return; end
    r = r - alpha * q;
end
```

Dubious termination criterion !



## 5.4.1 Minimal residual methods

Idea: Replace Euclidean inner product in CG with  $\mathbf{A}$ -inner product

$$\|\mathbf{x}^{(l)} - \mathbf{x}\|_{\mathbf{A}} \text{ replaced with } \|\mathbf{A}(\mathbf{x}^{(l)} - \mathbf{x})\|_2 = \|\mathbf{r}_l\|_2$$

▶ **MINRES** method [28, Sect. 9.5.2] (for *any* symmetric matrix !)

**Theorem 5.4.1.** For  $\mathbf{A} = \mathbf{A}^H \in \mathbb{R}^{n,n}$  the residuals  $\mathbf{r}_l$  generated in the MINRES iteration satisfy

$$\|\mathbf{r}_l\|_2 = \min\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\}$$
$$\|\mathbf{r}_l\|_2 \leq \frac{2 \left(1 - \frac{1}{\kappa(\mathbf{A})}\right)^l}{\left(1 + \frac{1}{\kappa(\mathbf{A})}\right)^{2l} + \left(1 - \frac{1}{\kappa(\mathbf{A})}\right)^{2l}} \|\mathbf{r}_0\|_2 .$$

Note: similar formula for (linear) rate of convergence as for CG, see Thm. 5.2.25, but with  $\sqrt{\kappa(\mathbf{A})}$  replaced with  $\kappa(\mathbf{A})$  !

Iterative solver for  $\mathbf{Ax} = \mathbf{b}$  with *symmetric* system matrix  $\mathbf{A}$ :

MATLAB-functions: 

- `[x, flg, res, it, resv] = minres(A, b, tol, maxit, B, [], x0);`
- `[...] = minres(Afun, b, tol, maxit, Binvsfun, [], x0);`

Computational costs: 1  $\mathbf{A}$   $\times$  vector, 1  $\mathbf{B}^{-1}$   $\times$  vector per step, a few dot products & SAXPYs

Memory requirement: a few vectors  $\in \mathbb{R}^n$

Extension to general regular  $\mathbf{A} \in \mathbb{R}^{n,n}$ :

Idea: Solver overdetermined linear system of equations

$$\mathbf{x}^{(l)} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0): \quad \mathbf{A}\mathbf{x}^{(l)} = \mathbf{b}$$

in *least squares sense*,  $\rightarrow$  Chapter 7.

$$\mathbf{x}^{(l)} = \operatorname{argmin}\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\}.$$

➤ **GMRES** method for general matrices  $\mathbf{A} \in \mathbb{R}^{n,n}$   $\rightarrow$  [35, Ch. 16], [51, Sect. 4.4.2]

MATLAB-function: • `[x, flag, relr, it, rv] = gmres(A, b, rs, tol, maxit, B, [], x0);`  
 • `[...] = gmres(Afun, b, rs, tol, maxit, Binvsfun, [], x0);`

Computational costs : 1  $\mathbf{A}$   $\times$  vector, 1  $\mathbf{B}^{-1}$   $\times$  vector per step,  
 :  $O(l)$  dot products & SAXPYs in  $l$ -th step

Memory requirements:  $O(l)$  vectors  $\in \mathbb{K}^n$  in  $l$ -th step

*Remark 5.4.2 (Restarted GMRES).*

After many steps of GMRES we face considerable computational costs and memory requirements for every further step. Thus, the iteration may be *restarted* with the current iterate  $\mathbf{x}^{(l)}$  as initial guess  $\rightarrow$



## 5.4.2 Iterations with short recursions [51, Sect. 4.5]

Iterative methods for *general* regular system matrix  $\mathbf{A}$ :

Idea: Given  $\mathbf{x}^{(0)} \in \mathbb{R}^n$  determine (better) approximation  $\mathbf{x}^{(l)}$  through **Petrov-Galerkin condition**

$$\mathbf{x}^{(l)} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0): \quad \mathbf{p}^H (\mathbf{b} - \mathbf{A}\mathbf{x}^{(l)}) = 0 \quad \forall \mathbf{p} \in W_l,$$

with suitable **test space**  $W_l$ ,  $\dim W_l = l$ , e.g.  $W_l := \mathcal{K}_l(\mathbf{A}^H, \mathbf{r}_0)$  ( $\rightarrow$  bi-conjugate gradients, BiCG)



MATLAB-function: • `[x, flag, r, it, rv] = bicgstab(A, b, tol, maxit, B, [], x0)`  
 • `[...] = bicgstab(Afun, b, tol, maxit, Binvsfun, [], x0);`

Computational costs :  $2 \mathbf{A} \times \text{vector}$ ,  $2 \mathbf{B}^{-1} \times \text{vector}$ , 4 dot products, 6 SAXPYs per step

Memory requirements: 8 vectors  $\in \mathbb{R}^n$

MATLAB-function: • `[x, flag, r, it, rv] = qmr(A, b, tol, maxit, B, [], x0)`  
 • `[...] = qmr(Afun, b, tol, maxit, Binvsfun, [], x0);`

Computational costs :  $2 \mathbf{A} \times \text{vector}$ ,  $2 \mathbf{B}^{-1} \times \text{vector}$ , 2 dot products, 12 SAXPYs per step

Memory requirements: 10 vectors  $\in \mathbb{R}^n$



- little (useful) convergence theory available
- stagnation & “breakdowns” commonly occur

*Example 5.4.3 (Failure of Krylov iterative solvers).*



$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & \cdots & & \cdots & 0 \\ 0 & 0 & 1 & 0 & & & \vdots \\ \vdots & \cdots & \cdots & \cdots & & & \vdots \\ & & & & \cdots & & \vdots \\ \vdots & & & \cdots & \cdots & & 0 \\ 0 & & & & 0 & 1 & \\ 1 & 0 & \cdots & & \cdots & 0 & \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ 1 \end{pmatrix} \quad \blacktriangleright \quad \mathbf{x} = \mathbf{e}_1.$$

$$\mathbf{x}^{(0)} = 0 \quad \blacktriangleright \quad \mathbf{r}_0 = \mathbf{e}_n \quad \blacktriangleright \quad \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0) = \text{Span} \{ \mathbf{e}_n, \mathbf{e}_{n-1}, \dots, \mathbf{e}_{n-l+1} \}$$

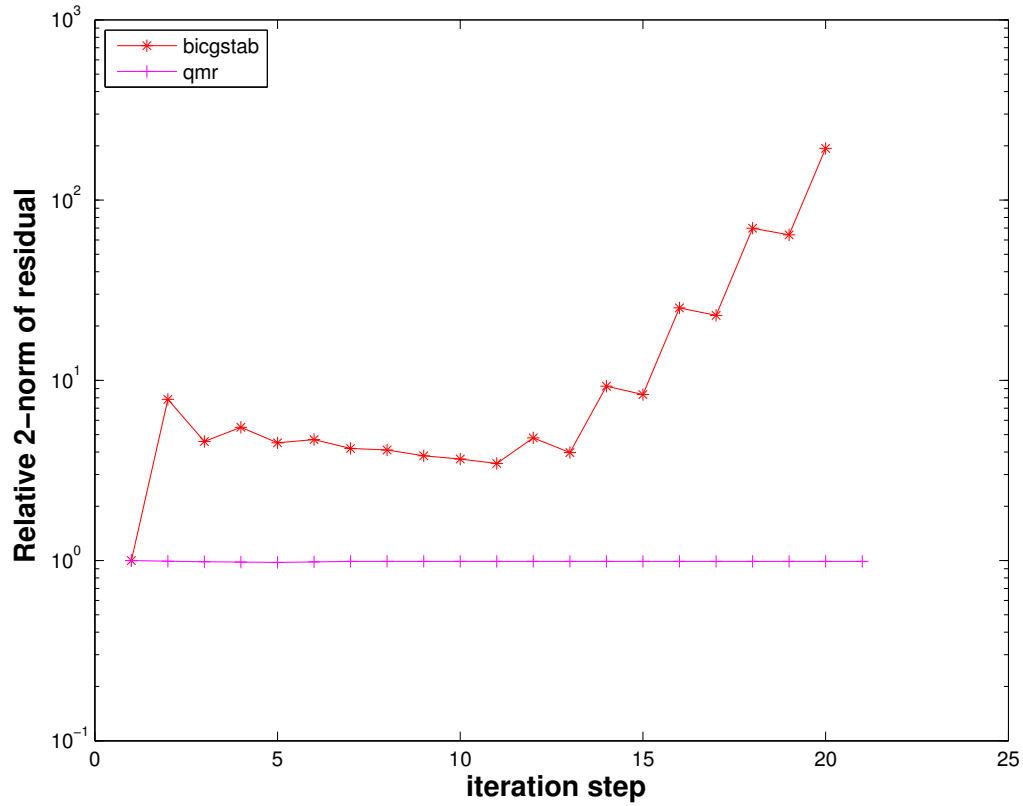
$$\blacktriangleright \quad \min \{ \|\mathbf{y} - \mathbf{x}\|_2 : \mathbf{y} \in \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0) \} = \begin{cases} 1 & , \text{ if } l \leq n, \\ 0 & , \text{ for } l = n. \end{cases}$$

TRY & PRAY

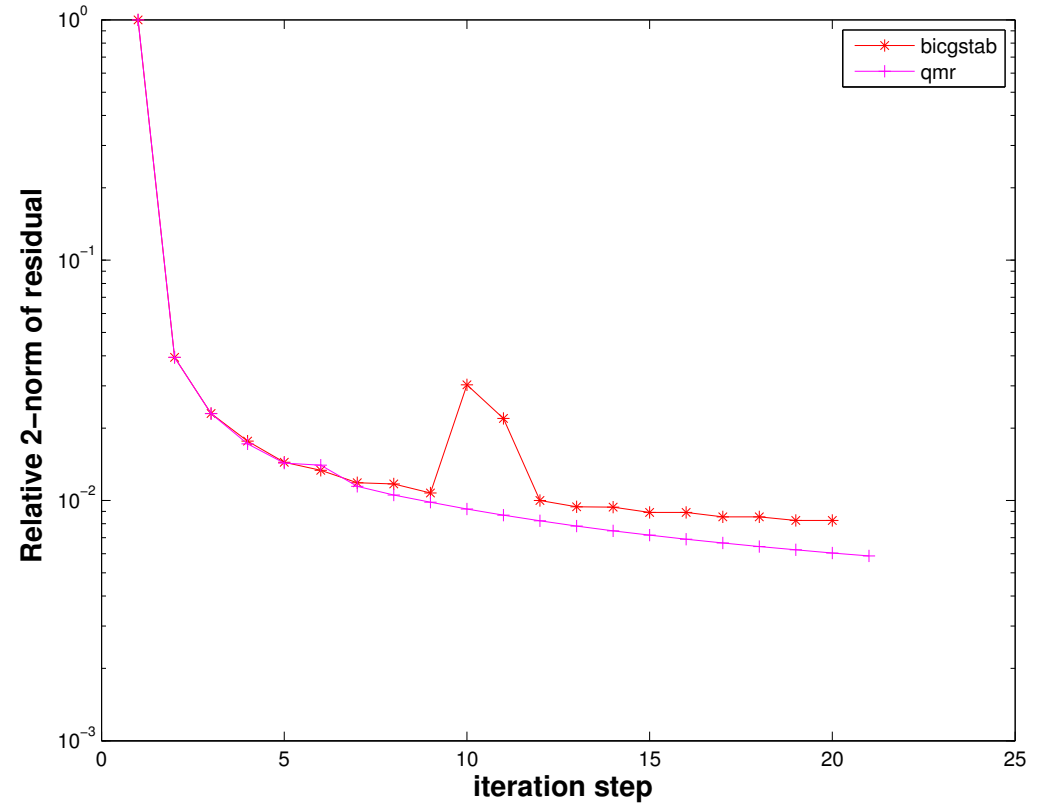
*Example 5.4.4* (Convergence of Krylov subspace methods for non-symmetric system matrix).

```
A = gallery('tridiag', -0.5*ones(n-1,1), 2*ones(n,1), -1.5*ones(n-1,1));
B = gallery('tridiag', 0.5*ones(n-1,1), 2*ones(n,1), 1.5*ones(n-1,1));
```

Plotted:  $\|\mathbf{r}_l\|_2 : \|\mathbf{r}_0\|_2$ :



tridiagonal matrix **A**



tridiagonal matrix **B**



**Summary:**

*Advantages* of Krylov methods vs. direct elimination (**IF** they converge at all/sufficiently fast).

- They require system matrix  $A$  in procedural form  $y = \text{eval}A(x) \leftrightarrow y = Ax$  only.
- They can perfectly exploit sparsity of system matrix.
- They can cash in on low accuracy requirements (**IF** viable termination criterion available).
- They can benefit from a good initial guess.

# 6

## Eigenvalues



### Supplementary and further reading:

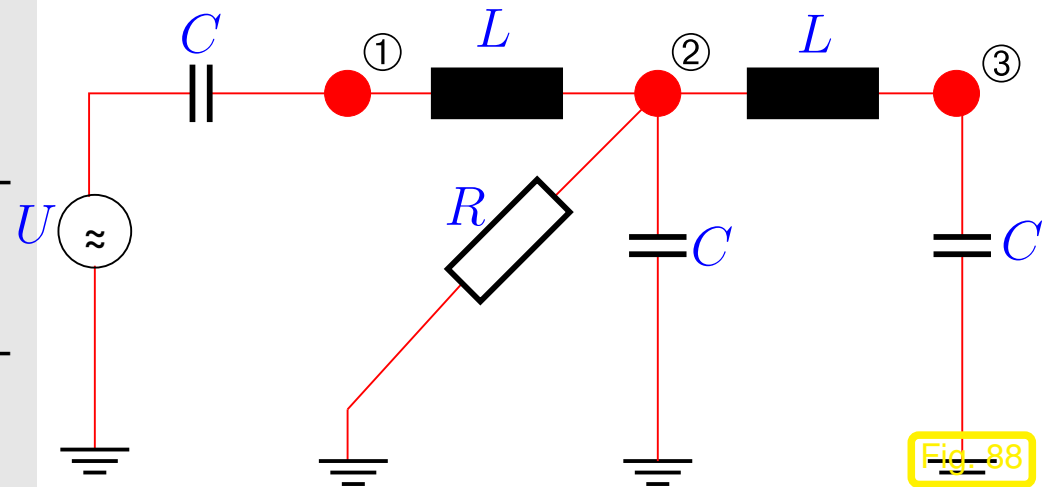
[4] offers comprehensive presentation of numerical methods for the solution of eigenvalue problems from an algorithmic point of view.

R. Hiptmair  
rev 38286,  
November  
11, 2011

*Example 6.0.1* (Resonances of linear electric circuits).

## Simple electric circuit, cf. Ex. 2.6.3

- linear components (resistors, coils, capacitors) only,
- time-harmonic excitation (alternating voltage/current)
- “frequency domain” circuit model



Ex. 2.6.3: nodal analysis of linear ( $\leftrightarrow$  composed of resistors, inductors, capacitors) electric circuit in **frequency domain** (at angular frequency  $\omega > 0$ ), see (2.6.5))

- linear system of equations for nodal potentials with complex system matrix  $\mathbf{A}$

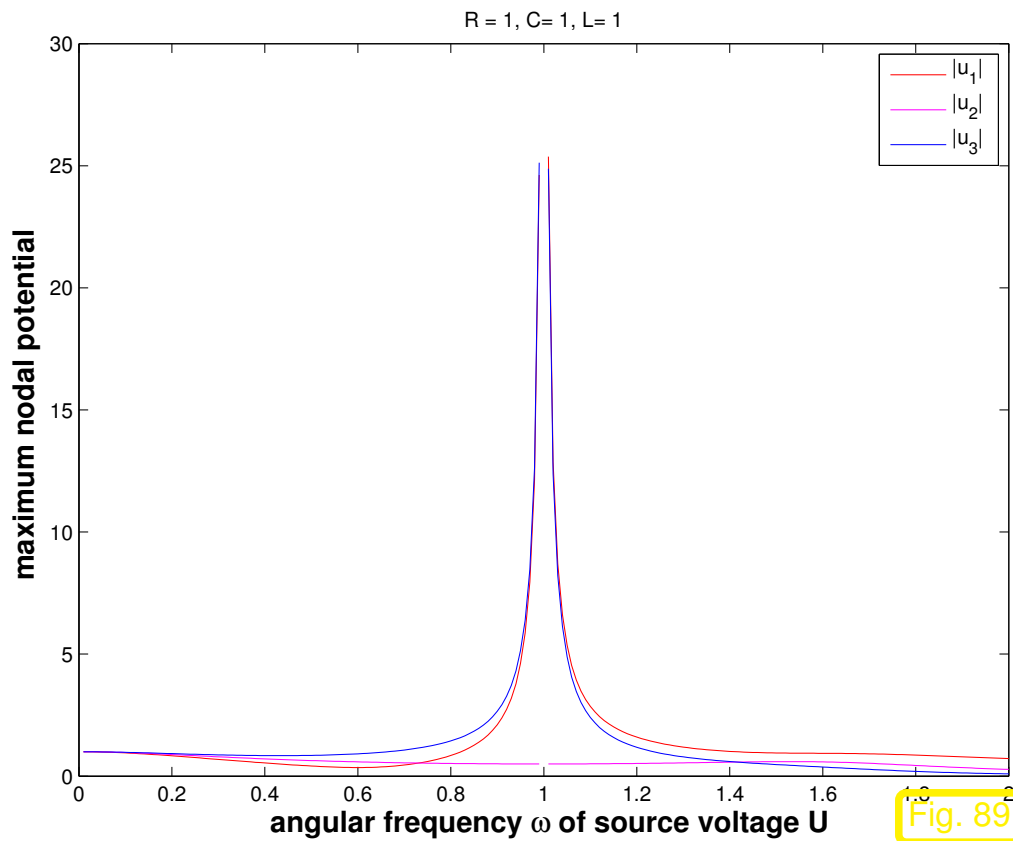
For circuit of Fig. 6.0.2: three unknown nodal potentials

➤ system matrix from nodal analysis at angular frequency  $\omega > 0$ :

$$\mathbf{A} = \begin{pmatrix} i\omega C + \frac{1}{i\omega L} & -\frac{1}{i\omega L} & 0 \\ -\frac{1}{i\omega L} & i\omega C + \frac{1}{R} + \frac{2}{i\omega L} & -\frac{1}{i\omega L} \\ 0 & -\frac{1}{i\omega L} & i\omega C + \frac{1}{i\omega L} \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{R} & 0 \\ 0 & 0 & 0 \end{pmatrix} + i\omega \begin{pmatrix} C & 0 & 0 \\ 0 & C & 0 \\ 0 & 0 & C \end{pmatrix} + 1/i\omega \begin{pmatrix} \frac{1}{L} & -\frac{1}{L} & 0 \\ -\frac{1}{L} & \frac{2}{L} & -\frac{1}{L} \\ 0 & -\frac{1}{L} & \frac{1}{L} \end{pmatrix} .$$

$$\mathbf{A}(\omega) := \mathbf{W} + i\omega\mathbf{C} - i\omega^{-1}\mathbf{S} \quad , \quad \mathbf{W}, \mathbf{C}, \mathbf{S} \in \mathbb{R}^{n,n} \text{ symmetric} . \quad (6.0.2)$$



◁ plot of  $|u_i(U)|$ ,  $i = 1, 2, 3$  for  $R = L = C = 1$   
(scaled model)

*Blow-up* of some nodal potentials for certain  $\omega$  !

### Code 6.0.3: Computation of nodal potential for circuit of Fig. 6.0.2

```

1 function rescirc(R,L,C)
2 % Ex. ??: Numerical nodal analysis of the resonant circuit
3 % R, L, C  $\hat{=}$  network component parameters
4
5 Z = 1/R; K = 1/L;
6
7 % Matrices W, C, S for nodal analysis of circuit

```

```
8 Wmat = [0 0 0; 0 Z 0; 0 0 0];
9 Cmat = [C 0 0; 0 C 0; 0 0 C];
10 Smat = [K -K 0; -K 2*K -K; 0 -K K];
11 % System matrix from nodal analysis
12 Amat = @(w) (Wmat+i*w*Cmat+Smat/(i*w));
13
14 % Scanning source currents
15 res = [];
16 for w=0.01:0.01:2
17     res = [res; w, abs(Amat(w)\[C;0;0])'];
18 end
19
20 figure('name','resonant_circuit');
21 plot(res(:,1),res(:,2),'r-',res(:,1),res(:,3),'m-',res(:,1),res(:,4),'b-');
22 xlabel('\bf angular frequency \omega of source voltage
    U}','fontsize',14);
23 ylabel('\bf maximum nodal potential}','fontsize',14);
24 title(sprintf('R = %d, C= %d, L= %d',R,L,C));
25 legend('|u_1|','|u_2|','|u_3|');
26
27 print -depsc2 '../PICTURES/rescircpot.eps'
28
29 % Solving generalized eigenvalue problem (6.0.5)
```



```
30 Zmat = zeros (3,3); Imat = eye (3,3);
31 % Assemble 6x6-matrices M and B
32 Mmat = [Wmat, Smat; Imat, Zmat];
33 Bmat = [-i*Cmat, Zmat; Zmat, i*Imat];
34 % Solve generalized eigenvalue problem, cf. (6.0.6)
35 omega = eig (Mmat, Bmat);
36
37 figure ('name', 'resonances');
38 plot (real (omega), imag (omega), 'r*'); hold on;
39 ax = axis;
40 plot ([ax(1) ax(2)], [0 0], 'k-');
41 plot ([ 0 0], [ax(3) ax(4)], 'k-');
42 grid on;
43 xlabel ('\bf Re(\omega)', 'fontsize', 14);
44 ylabel ('\bf Im(\omega)', 'fontsize', 14);
45 title (sprintf ('R = %d, C = %d, L = %d', R, L, C));
46 legend ('\omega');
47
48 print -depsc2 '../PICTURES/rescircomega.eps'
```

resonant frequencies =  $\omega \in \{\omega \in \mathbb{R}: \mathbf{A}(\omega) \text{ singular}\}$

If the circuit is operated at a real resonant frequency, the circuit equations will not possess a solution. Of course, the real circuit will always behave in a well-defined way, but the linear model will break down due to extremely large currents and voltages. In an experiment this breakdown manifests itself as a rather explosive meltdown of circuits components. Hence, it is vital to determine resonant frequencies of circuits in order to avoid their destruction.

➔ relevance of numerical methods for solving:

$$\text{Find } \omega \in \mathbb{C} \setminus \{0\}: \quad \mathbf{W} + i\omega\mathbf{C} + \frac{1}{i\omega}\mathbf{S} \text{ singular .}$$

This is a **quadratic eigenvalue problem**: find  $\mathbf{x} \neq 0, \omega \in \mathbb{C} \setminus \{0\}$ ,

$$\mathbf{A}(\omega)\mathbf{x} = (\mathbf{W} + i\omega\mathbf{C} + \frac{1}{i\omega}\mathbf{S})\mathbf{x} = 0 . \quad (6.0.4)$$

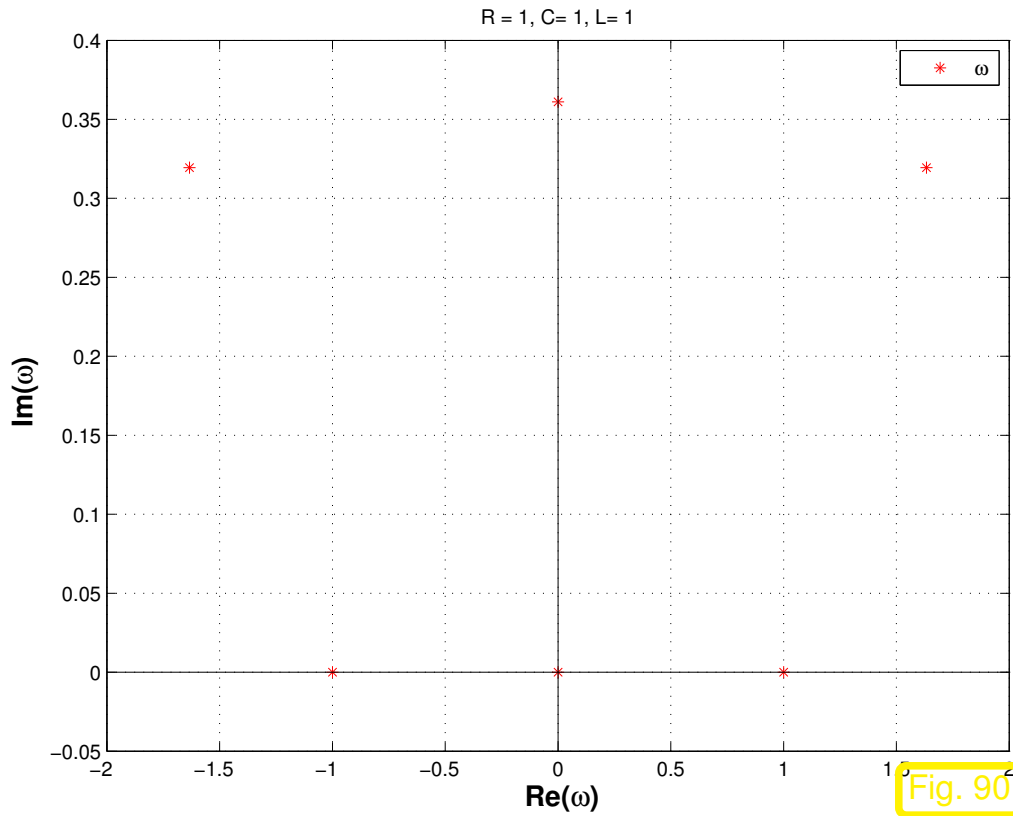
Substitution:  $\mathbf{y} = \frac{1}{i\omega}\mathbf{x} \Leftrightarrow \mathbf{x} = i\omega\mathbf{y}$  [64, Sect. 3.4]:

$$(6.0.4) \Leftrightarrow \underbrace{\begin{pmatrix} \mathbf{W} & \mathbf{S} \\ \mathbf{I} & \mathbf{0} \end{pmatrix}}_{:=\mathbf{M}} \underbrace{\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}}_{:=\mathbf{z}} = \omega \underbrace{\begin{pmatrix} -i\mathbf{C} & \mathbf{0} \\ \mathbf{0} & -i\mathbf{I} \end{pmatrix}}_{:=\mathbf{B}} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \quad (6.0.5)$$

➤ **generalized linear eigenvalue problem** of the form: find  $\omega \in \mathbb{C}, \mathbf{z} \in \mathbb{C}^{2n} \setminus \{0\}$  such that

$$\mathbf{M}\mathbf{z} = \omega\mathbf{B}\mathbf{z} . \quad (6.0.6)$$

In this example one is mainly interested in the **eigenvalues**  $\omega$ , whereas the **eigenvectors**  $\mathbf{z}$  usually need not be computed.



◁ resonant frequencies for circuit from Fig. 6.0.2  
(including decaying modes with  $\text{Im}(\omega) > 0$ )



*Example 6.0.7* (Analytic solution of homogeneous linear ordinary differential equations). → [63, Remark 5.6.1], [27, Sect. 10.1], [48, Sect. 8.1], [13, Ex. 7.3]

Autonomous homogeneous linear ordinary differential equation (ODE):

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \quad , \quad \mathbf{A} \in \mathbb{C}^{n,n} . \tag{6.0.8}$$

$$\mathbf{A} = \mathbf{S} \underbrace{\begin{pmatrix} \lambda_1 & & \\ & \cdots & \\ & & \lambda_n \end{pmatrix}}{=: \mathbf{D}} \mathbf{S}^{-1}, \quad \mathbf{S} \in \mathbb{C}^{n,n} \text{ regular} \implies \left( \dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \quad \begin{array}{c} \mathbf{z} = \mathbf{S}^{-1}\mathbf{y} \\ \longleftrightarrow \\ \dot{\mathbf{z}} = \mathbf{D}\mathbf{z} \end{array} \right).$$

➤ solution of initial value problem:

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y}, \quad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{C}^n \implies \mathbf{y}(t) = \mathbf{S}\mathbf{z}(t), \quad \dot{\mathbf{z}} = \mathbf{D}\mathbf{z}, \quad \mathbf{z}(0) = \mathbf{S}^{-1}\mathbf{y}_0.$$

The initial value problem for the *decoupled* homogeneous linear ODE  $\dot{\mathbf{z}} = \mathbf{D}\mathbf{z}$  has a simple analytic solution

$$\mathbf{z}_i(t) = \exp(\lambda_i t) (\mathbf{z}_0)_i = \exp(\lambda_i t) \left( (\mathbf{S}^{-1})_{i,:}^T \mathbf{y}_0 \right).$$

In light of Rem. 1.2.2:

$$\mathbf{A} = \mathbf{S} \begin{pmatrix} \lambda_1 & & \\ & \cdots & \\ & & \lambda_n \end{pmatrix} \mathbf{S}^{-1} \iff \mathbf{A} ((\mathbf{S})_{:,i}) = \lambda_i ((\mathbf{S})_{:,i}) \quad i = 1, \dots, n. \quad (6.0.9)$$

In order to find the transformation matrix  $\mathbf{S}$  all non-zero solution vectors (= **eigenvectors**)  $\mathbf{x} \in \mathbb{C}^n$  of the **linear eigenvalue problem**

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

have to be found.

## 6.1 Theory of eigenvalue problems [48, Ch. 7], [27, Ch. 9], [51, Sect. 1.7]

**Definition 6.1.1** (Eigenvalues and eigenvectors).  $\rightarrow$  [48, Sects. 7.1,7.2], [27, Sect. 9.1]

- $\lambda \in \mathbb{C}$  **eigenvalue** (ger.: *Eigenwert*) of  $\mathbf{A} \in \mathbb{K}^{n,n}$   $:\Leftrightarrow$   $\underbrace{\det(\lambda\mathbf{I} - \mathbf{A})}_{\text{characteristic polynomial } \chi(\lambda)} = 0$
- **spectrum** of  $\mathbf{A} \in \mathbb{K}^{n,n}$ :  $\sigma(\mathbf{A}) := \{\lambda \in \mathbb{C} : \lambda \text{ eigenvalue of } \mathbf{A}\}$
- **eigenspace** (ger.: *Eigenraum*) associated with eigenvalue  $\lambda \in \sigma(\mathbf{A})$ :  
 $\text{Eig}_{\mathbf{A}}(\lambda) := \text{Ker}(\lambda\mathbf{I} - \mathbf{A})$
- $\mathbf{x} \in \text{Eig}_{\mathbf{A}}(\lambda) \setminus \{0\} \Rightarrow \mathbf{x}$  is **eigenvector**
- **Geometric multiplicity** (ger.: *Vielfachheit*) of an eigenvalue  $\lambda \in \sigma(\mathbf{A})$ :  
 $m(\lambda) := \dim \text{Eig}_{\mathbf{A}}(\lambda)$

Two simple facts:

$$\lambda \in \sigma(\mathbf{A}) \Rightarrow \dim \text{Eig}_{\mathbf{A}}(\lambda) > 0, \quad (6.1.2)$$

$$\det(\mathbf{A}) = \det(\mathbf{A}^T) \quad \forall \mathbf{A} \in \mathbb{K}^{n,n} \Rightarrow \sigma(\mathbf{A}) = \sigma(\mathbf{A}^T). \quad (6.1.3)$$

 notation:  $\rho(\mathbf{A}) := \max\{|\lambda| : \lambda \in \sigma(\mathbf{A})\} \hat{=} \text{spectral radius of } \mathbf{A} \in \mathbb{K}^{n,n}$

**Theorem 6.1.4** (Bound for spectral radius).

For any matrix norm  $\|\cdot\|$  induced by a vector norm ( $\rightarrow$  Def. 2.5.5)

$$\rho(\mathbf{A}) \leq \|\mathbf{A}\|.$$

*Proof.* Let  $\mathbf{z} \in \mathbb{C}^n \setminus \{0\}$  be an eigenvector to the largest (in modulus) eigenvalue  $\lambda$  of  $\mathbf{A} \in \mathbb{C}^{n,n}$ .

Then

$$\|\mathbf{A}\| := \sup_{\mathbf{x} \in \mathbb{C}^{n,n} \setminus \{0\}} \frac{\|\mathbf{Ax}\|}{\|\mathbf{x}\|} \geq \frac{\|\mathbf{Az}\|}{\|\mathbf{z}\|} = |\lambda| = \rho(\mathbf{A}) .$$

**Lemma 6.1.5** (Gershgorin circle theorem).  $\rightarrow$  [13, Thm. 7.13], [35, Thm. 32.1], [51, Sect. 5.1]

For any  $\mathbf{A} \in \mathbb{K}^{n,n}$  holds true

$$\sigma(\mathbf{A}) \subset \bigcup_{j=1}^n \left\{ z \in \mathbb{C} : |z - a_{jj}| \leq \sum_{i \neq j} |a_{ji}| \right\} .$$

**Lemma 6.1.6** (Similarity and spectrum).  $\rightarrow$  [27, Thm. 9.7], [13, Lemma 7.6], [48, Thm. 7.2]

The spectrum of a matrix is invariant with respect to **similarity transformations**:

$$\forall \mathbf{A} \in \mathbb{K}^{n,n} \quad (\sigma = \lambda(\mathbf{A})) \quad (\mathbf{A}) \quad \forall \text{regular } \mathbf{S} \in \mathbb{K}^{n,n}$$

**Lemma 6.1.7.** *Existence of a one-dimensional invariant subspace*

$$\forall \mathbf{C} \in \mathbb{C}^{n,n}: \exists \mathbf{u} \in \mathbb{C}^n: \mathbf{C}(\text{Span}\{\mathbf{u}\}) \subset \text{Span}\{\mathbf{u}\} .$$

**Theorem 6.1.8** (Schur normal form).  $\rightarrow$  [29, Thm .2.8.1]

$$\forall \mathbf{A} \in \mathbb{K}^{n,n}: \exists \mathbf{U} \in \mathbb{C}^{n,n} \text{ unitary: } \mathbf{U}^H \mathbf{A} \mathbf{U} = \mathbf{T} \text{ with } \mathbf{T} \in \mathbb{C}^{n,n} \text{ upper triangular .}$$



**Corollary 6.1.9** (Principal axis transformation).

$$\mathbf{A} \in \mathbb{K}^{n,n}, \mathbf{A}\mathbf{A}^H = \mathbf{A}^H\mathbf{A}: \exists \mathbf{U} \in \mathbb{C}^{n,n} \text{ unitary: } \mathbf{U}^H\mathbf{A}\mathbf{U} = \text{diag}(\lambda_1, \dots, \lambda_n), \lambda_i \in \mathbb{C}.$$

A matrix  $\mathbf{A} \in \mathbb{K}^{n,n}$  with  $\mathbf{A}\mathbf{A}^H = \mathbf{A}^H\mathbf{A}$  is called **normal**.

Examples of normal matrices are

- Hermitian matrices:  $\mathbf{A}^H = \mathbf{A}$   $\blacktriangleright$   $\sigma(\mathbf{A}) \subset \mathbb{R}$
- unitary matrices:  $\mathbf{A}^H = \mathbf{A}^{-1}$   $\blacktriangleright$   $|\sigma(\mathbf{A})| = 1$
- skew-Hermitian matrices:  $\mathbf{A} = -\mathbf{A}^H$   $\blacktriangleright$   $\sigma(\mathbf{A}) \subset i\mathbb{R}$

Normal matrices can be diagonalized by *unitary* similarity transformations

Symmetric real matrices can be diagonalized by *orthogonal* similarity transformations

- In Thm. 6.1.9:
- $\lambda_1, \dots, \lambda_n$  = eigenvalues of  $\mathbf{A}$
  - Columns of  $\mathbf{U}$  = orthonormal basis of eigenvectors of  $\mathbf{A}$

## Eigenvalue

- problems: (EVPs)
- ❶ Given  $\mathbf{A} \in \mathbb{K}^{n,n}$  find **all eigenvalues** (= spectrum of  $\mathbf{A}$ ).
  - ❷ Given  $\mathbf{A} \in \mathbb{K}^{n,n}$  find  $\sigma(\mathbf{A})$  plus **all eigenvectors**.
  - ❸ Given  $\mathbf{A} \in \mathbb{K}^{n,n}$  find **a few** eigenvalues and associated eigenvectors

(Linear) **generalized eigenvalue problem**:

Given  $\mathbf{A} \in \mathbb{C}^{n,n}$ , regular  $\mathbf{B} \in \mathbb{C}^{n,n}$ , seek  $\mathbf{x} \neq 0$ ,  $\lambda \in \mathbb{C}$

$$\mathbf{Ax} = \lambda \mathbf{Bx} \Leftrightarrow \mathbf{B}^{-1} \mathbf{Ax} = \lambda \mathbf{x} . \quad (6.1.10)$$

$\mathbf{x} \hat{=}$  generalized eigenvector,  $\lambda \hat{=}$  generalized eigenvalue

Obviously every generalized eigenvalue problem is equivalent to a standard eigenvalue problem

$$\mathbf{Ax} = \lambda \mathbf{Bx} \Leftrightarrow \mathbf{B}^{-1} \mathbf{Ax} = \lambda \mathbf{x} .$$

However, usually it is not advisable to use this equivalence for numerical purposes!

*Remark 6.1.11* (Generalized eigenvalue problems and Cholesky factorization).

If  $\mathbf{B} = \mathbf{B}^H$  s.p.d. ( $\rightarrow$  Def. 2.7.9) with Cholesky factorization  $\mathbf{B} = \mathbf{R}^H \mathbf{R}$

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x} \Leftrightarrow \tilde{\mathbf{A}}\mathbf{y} = \lambda\mathbf{y} \quad \text{where } \tilde{\mathbf{A}} := \mathbf{R}^{-H} \mathbf{A} \mathbf{R}^{-1}, \mathbf{y} := \mathbf{R}\mathbf{x}.$$

$\rightarrow$  This transformation can be used for efficient computations.



## 6.2 “Direct” Eigensolvers

Purpose: solution of eigenvalue problems ❶, ❷ for **dense** matrices “up to machine precision”

MATLAB-function:

`eig`

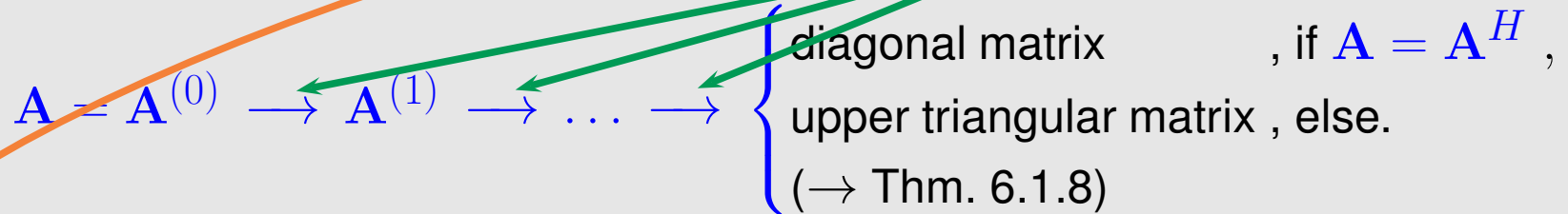
`d = eig(A)` : computes spectrum  $\sigma(\mathbf{A}) = \{d_1, \dots, d_n\}$  of  $\mathbf{A} \in \mathbb{C}^{n,n}$

`[V,D] = eig(A)` : computes  $\mathbf{V} \in \mathbb{C}^{n,n}$ , *diagonal*  $\mathbf{D} \in \mathbb{C}^{n,n}$  such that  $\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{D}$

Remark 6.2.1 (QR-Algorithm). → [23, Sect. 7.5], [48, Sect. 10.3], [35, Ch. 26], [51, Sect. 5.5-5.7]

Note: All “direct” eigensolvers are iterative methods

Idea: Iteration based on successive **unitary** similarity transformations



(superior stability of unitary transformations, see Rem. 2.8.1)

Code 6.2.2: QR-algorithm with shift

```

1 function d = eigqr(A,tol)
2 n = size (A,1);
3 while (norm (tril (A,-1)) > tol*norm (A))
4 % shift by eigenvalue of lower right 2x2 block
  % closest to (A)n,n
5   sc = eig (A(n-1:n,n-1:n));
6   [dummy, si] = min (abs (sc-A(n,n)));
7   shift = sc(si);
8   [Q,R] = qr ( A - shift * eye (n));
9   A = Q' *A*Q;
10 end

```

► QR-algorithm (with shift)

- in general: quadratic convergence

- cubic convergence for normal matrices

(→ [23, Sect. 7.5,8.2])

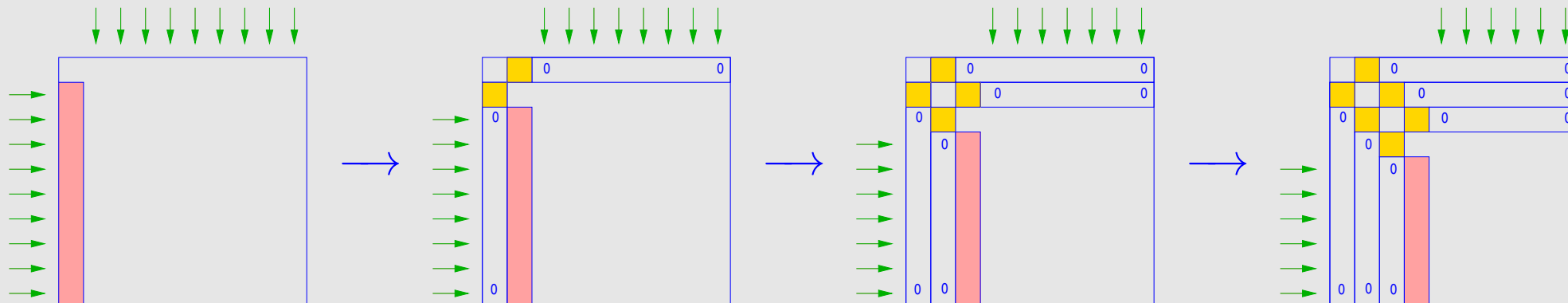
Computational cost:  $O(n^3)$  operations per step of the QR-algorithm

Library implementations of the QR-algorithm provide *numerically stable* eigensolvers ( $\rightarrow$  Def.2.5.11)

*Remark 6.2.3* (Unitary similarity transformation to tridiagonal form).

Successive Householder similarity transformations of  $\mathbf{A} = \mathbf{A}^H$ :

( $\rightarrow \hat{=}$  affected rows/columns,   $\hat{=}$  targeted vector)



► transformation to tridiagonal form ! (for general matrices a similar strategy can achieve a similarity transformation to upper Hessenberg form)

► this transformation is used as a preprocessing step for QR-algorithm  $\succ$  `eig`.

Similar functionality for generalized EVP  $\mathbf{Ax} = \lambda \mathbf{Bx}$ ,  $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{n,n}$

`d = eig(A,B)` : computes all generalized eigenvalues

`[V,D] = eig(A,B)` : computes  $\mathbf{V} \in \mathbb{C}^{n,n}$ , *diagonal*  $\mathbf{D} \in \mathbb{C}^{n,n}$  such that  $\mathbf{AV} = \mathbf{BVD}$

Note: (Generalized) eigenvectors can be recovered as columns of  $\mathbf{V}$ :

$$\mathbf{AV} = \mathbf{VD} \Leftrightarrow \mathbf{A}(\mathbf{V})_{:,i} = (\mathbf{D})_{i,i} \mathbf{V}_{:,i},$$

if  $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$ .

Remark 6.2.4 (Computational effort for eigenvalue computations).

Computational effort (#elementary operations) for `eig()`:

eigenvalues & eigenvectors of $\mathbf{A} \in \mathbb{K}^{n,n}$	$\sim 25n^3 + O(n^2)$	}	$O(n^3)!$
only eigenvalues of $\mathbf{A} \in \mathbb{K}^{n,n}$	$\sim 10n^3 + O(n^2)$		
eigenvalues and eigenvectors $\mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n}$	$\sim 9n^3 + O(n^2)$		
only eigenvalues of $\mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n}$	$\sim \frac{4}{3}n^3 + O(n^2)$		
only eigenvalues of <b>tridiagonal</b> $\mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n}$	$\sim 30n^2 + O(n)$		

Note: `eig` not available for sparse matrix arguments

Exception: `d=eig(A)` for sparse *Hermitian* matrices



Example 6.2.5 (Runtimes of `eig`).

```
1 A = rand(500,500); B = A'*A; C = gallery('tridiag',500,1,3,1);
```

- • **A** generic dense matrix
- **B** symmetric (s.p.d. → Def. 2.7.9) matrix
- **C** s.p.d. *tridiagonal* matrix

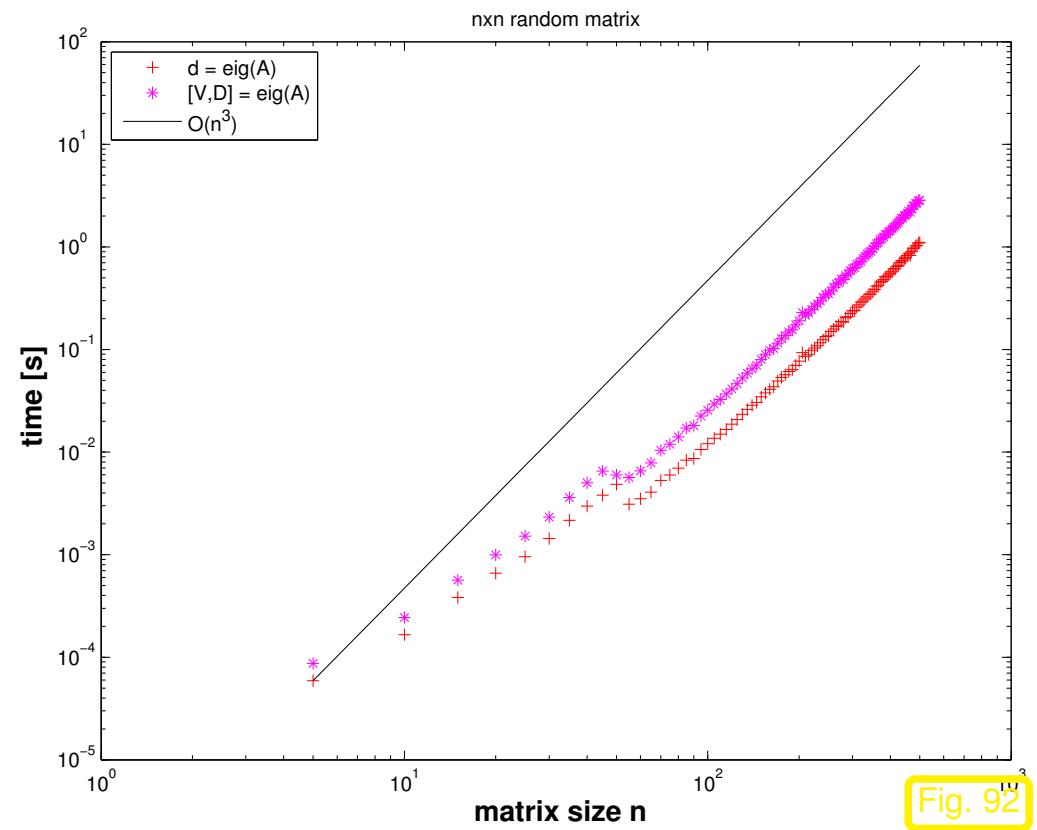
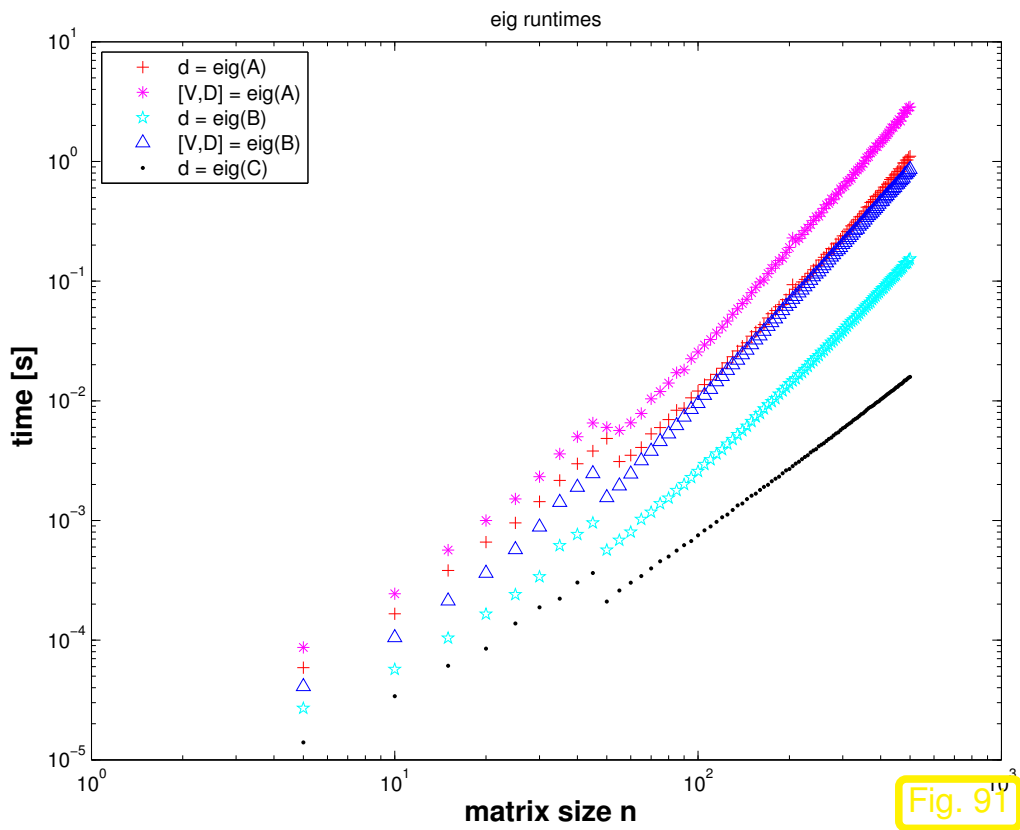
## Code 6.2.6: measuring runtimes of eig

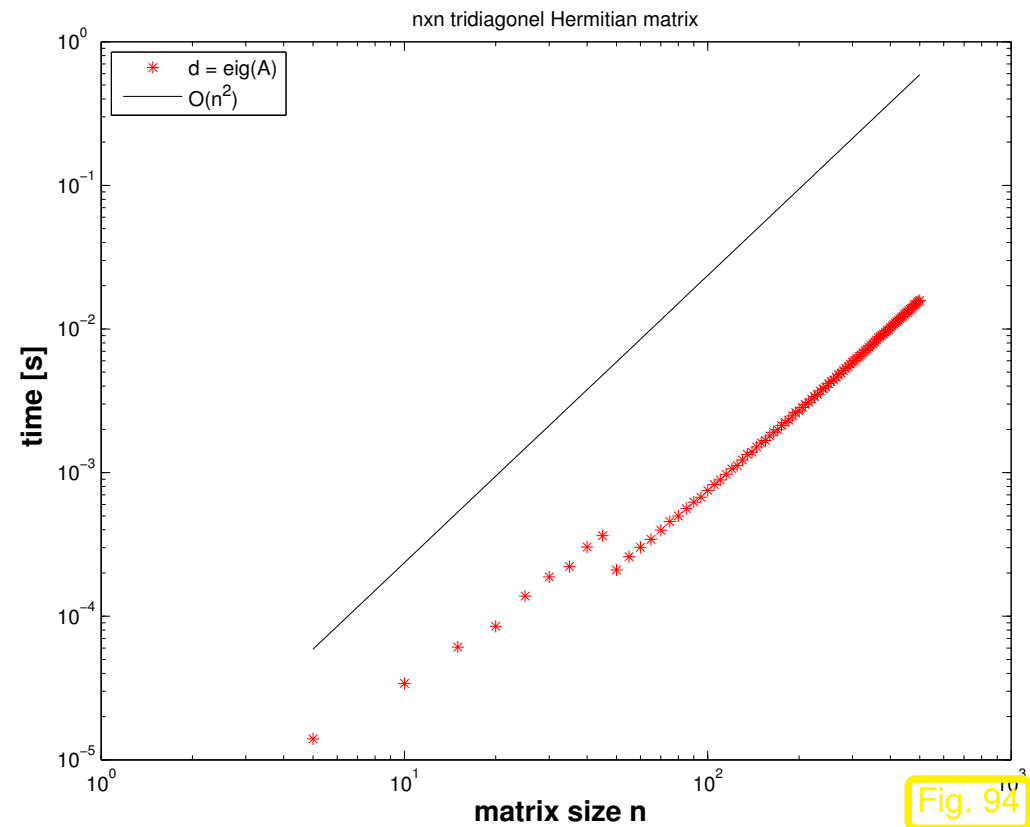
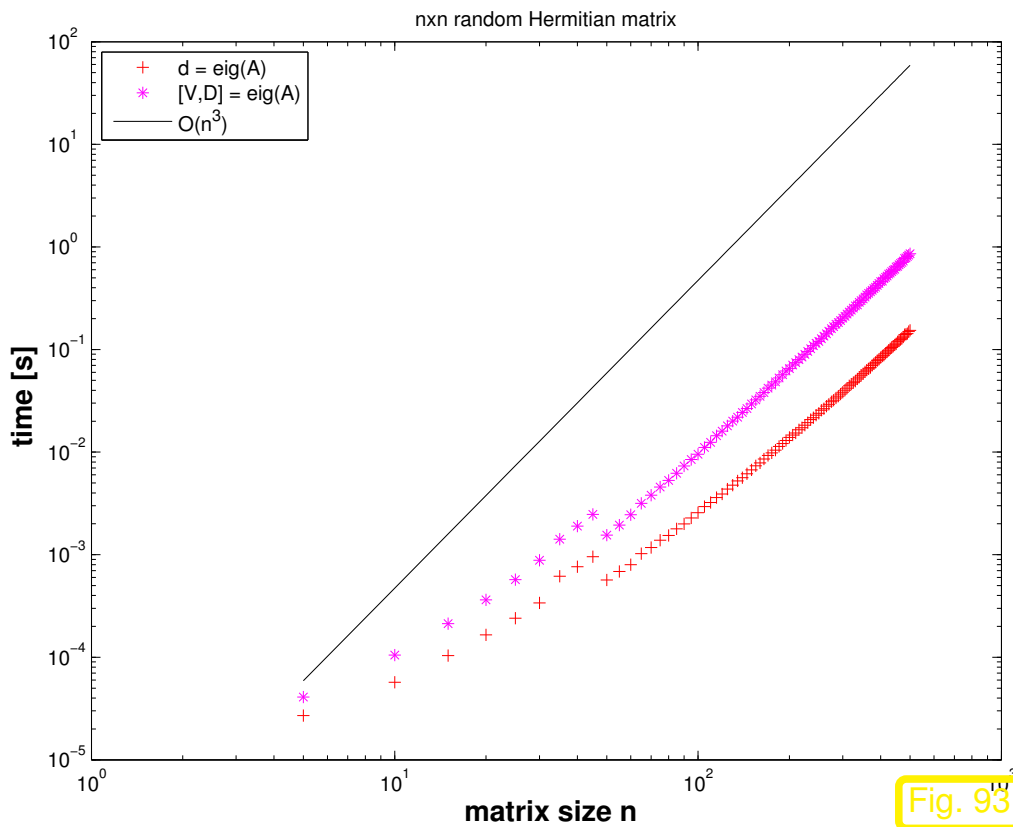
```
1 function eigtiming
2
3 A = rand(500,500); B = A'*A;
4 C = gallery('tridiag',500,1,3,1);
5 times = [];
6 for n=5:5:500
7     An = A(1:n,1:n); Bn = B(1:n,1:n); Cn = C(1:n,1:n);
8     t1 = 1000; for k=1:3, tic; d = eig(An); t1 = min(t1,toc); end
9     t2 = 1000; for k=1:3, tic; [V,D] = eig(An); t2 = min(t2,toc);
10    end
11    t3 = 1000; for k=1:3, tic; d = eig(Bn); t3 = min(t3,toc); end
12    t4 = 1000; for k=1:3, tic; [V,D] = eig(Bn); t4 = min(t4,toc);
13    end
14    t5 = 1000; for k=1:3, tic; d = eig(Cn); t5 = min(t5,toc); end
15    times = [times; n t1 t2 t3 t4 t5];
16 end
17 figure;
18 loglog(times(:,1),times(:,2),'r+', times(:,1),times(:,3),'m*', ...
19         times(:,1),times(:,4),'cp', times(:,1),times(:,5),'b^', ...
20         times(:,1),times(:,6),'k.');
```



```
20 xlabel ('{\bf matrix size n}', 'fontsize', 14);
21 ylabel ('{\bf time [s]}', 'fontsize', 14);
22 title ('eig runtimes');
23 legend ('d = eig(A)', '[V,D] = eig(A)', 'd = eig(B)', '[V,D] =
    eig(B)', 'd = eig(C)', ...
    'location', 'northwest');
24
25
26 print -depsc2 '../PICTURES/eigtimingall.eps'
27
28 figure;
29 loglog (times(:,1), times(:,2), 'r+', times(:,1), times(:,3), 'm*', ...
30         times(:,1), (times(:,1).^3)/(times(1,1)^3)*times(1,2), 'k-');
31 xlabel ('{\bf matrix size n}', 'fontsize', 14);
32 ylabel ('{\bf time [s]}', 'fontsize', 14);
33 title ('nxn random matrix');
34 legend ('d = eig(A)', '[V,D] =
    eig(A)', 'O(n^3)', 'location', 'northwest');
35
36 print -depsc2 '../PICTURES/eigtimingA.eps'
37
38 figure;
39 loglog (times(:,1), times(:,4), 'r+', times(:,1), times(:,5), 'm*', ...
40         times(:,1), (times(:,1).^3)/(times(1,1)^3)*times(1,2), 'k-');
```

```
41 xlabel ('{\bf matrix size n}', 'fontsize', 14);
42 ylabel ('{\bf time [s]}', 'fontsize', 14);
43 title ('nxn random Hermitian matrix');
44 legend ('d = eig(A)', '[V,D] =
    eig(A)', 'O(n^3)', 'location', 'northwest');
45
46 print -depsc2 '../PICTURES/eigtimingB.eps'
47
48 figure;
49 loglog (times(:,1), times(:,6), 'r*', ...
50         times(:,1), (times(:,1).^2) / (times(1,1)^2) * times(1,2), 'k-');
51 xlabel ('{\bf matrix size n}', 'fontsize', 14);
52 ylabel ('{\bf time [s]}', 'fontsize', 14);
53 title ('nxn tridiagonal Hermitian matrix');
54 legend ('d = eig(A)', 'O(n^2)', 'location', 'northwest');
55
56 print -depsc2 '../PICTURES/eigtimingC.eps'
```





☞ For the sake of efficiency: think which information you really need when computing eigenvalues/eigenvectors of dense matrices

Potentially more efficient methods for *sparse matrices* will be introduced below in Sects. 6.3, 6.4.



## 6.3 Power Methods

### 6.3.1 Direct power method [13, Sect. 7.5], [51, Sect. 5.3.1], [51, Sect. 5.3]

*Example 6.3.1* ((Simplified) Page rank algorithm).  $\rightarrow$  [42]

Model: **Random surfer** visits a web page, stays there for fixed time  $\Delta t$ , and then

- ❶ either follows each of  $\ell$  links on a page with probability  $1/\ell$ .
- ❷ or resumes surfing at a randomly (with equal probability) selected page

Option ❷ is chosen with probability  $d$ ,  $0 \leq d \leq 1$ , option ❶ with probability  $1 - d$ .

► Stationary **Markov chain**, state space  $\hat{=}$  set of all web pages

Question: Fraction of time spent by random surfer on  $i$ -th page (= **page rank**  $x_i \in [0, 1]$ )

This number  $\in ]0, 1[$  can be used to gauge the “importance” of a web page, which, in turns, offers a way to sort the hits resulting from a keyword query: the GOOGLE idea.

Method: Stochastic simulation



### Code 6.3.2: stochastic page rank simulation

```

1 function prstochsim(Nhops)
2 % Load web graph data stored in N x N-matrix G
3 load harvard500.mat;
4 N = size (G,1); d = 0.15;
5 count = zeros (1,N); cp = 1;
6 figure ('position', [0 0 1200 1000]); pause;
7 for n=1:Nhops
8     % Find links from current page cp
9     idx = find (G(:,cp)); l = size (idx,1); rn = rand (); %
10    % If no links, jump to any other pages with equal probability
11    if (isempty (idx)), cp = floor (rn*N)+1;
12    % With probability d jump to any other page
13    elseif (rn < d), cp = floor (rn/d*N)+1;
14    % Follow outgoing links with equal probability
15    else cp = idx(floor ((rn-d)/(1-d)*l)+1,1);
16 end
17 count(cp) = count(cp) + 1;

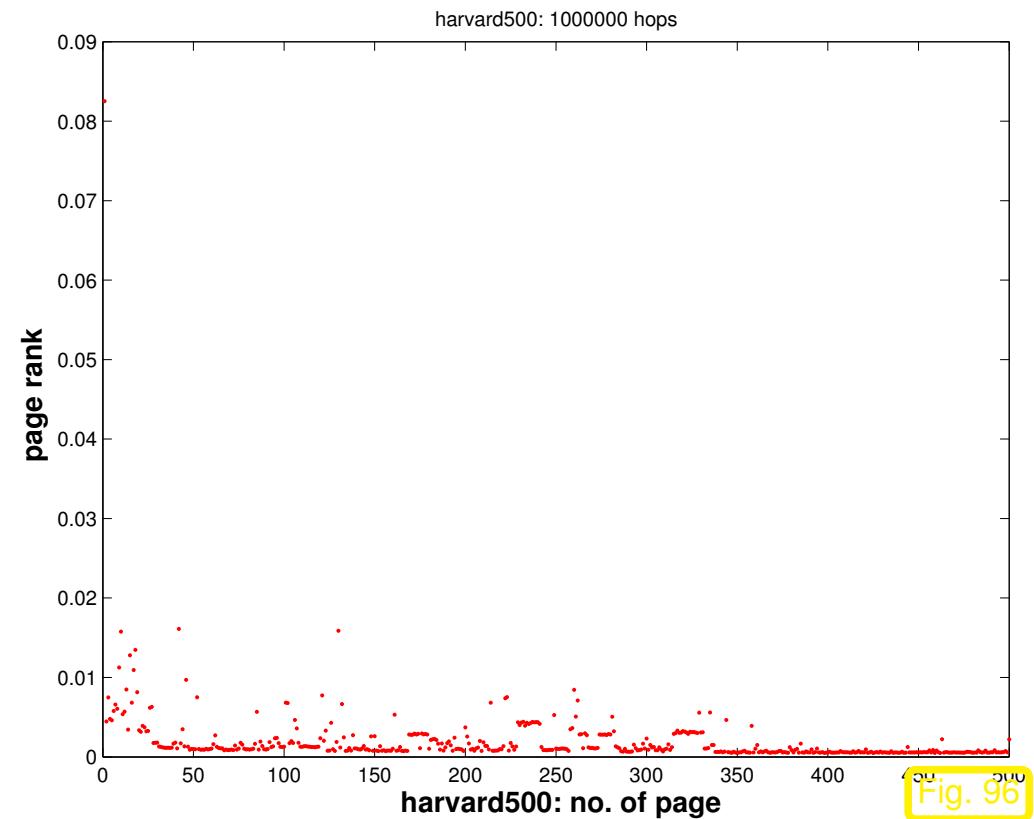
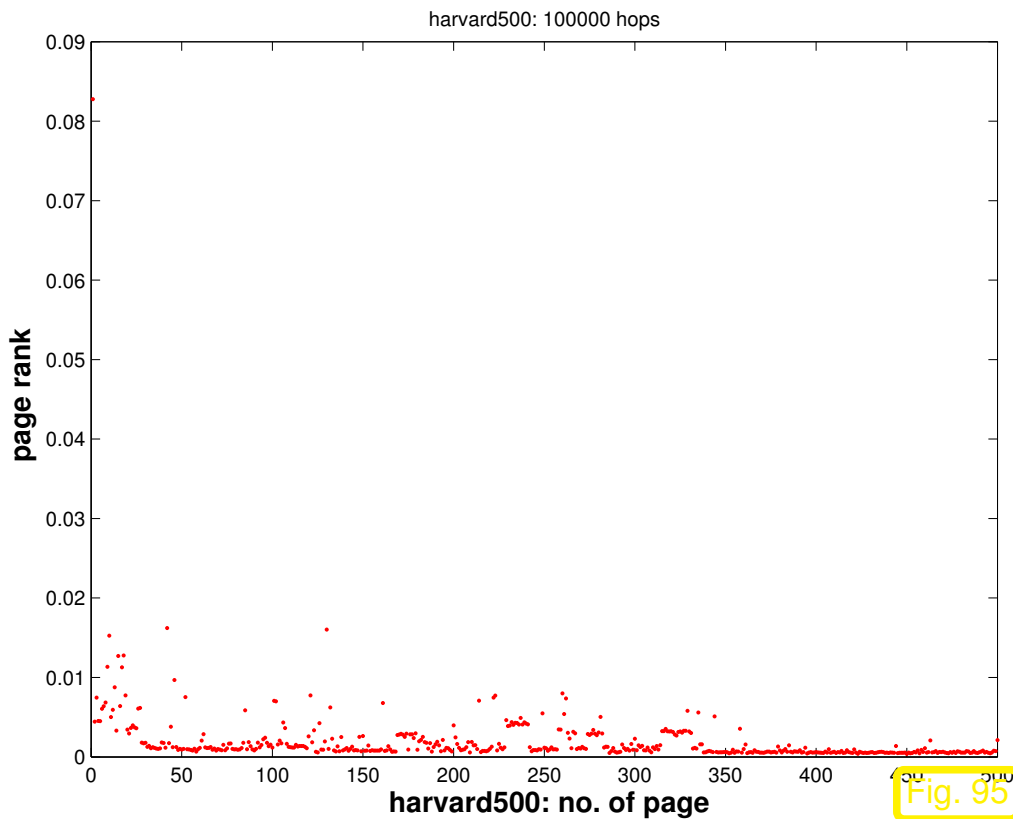
```

```
18 plot (1:N, count/n, 'r. '); axis ([0 N+1 0 0.1]);  
19 xlabel ('{\bf harvard500: no. of page}', 'fontsize', 14);  
20 ylabel ('{\bf page rank}', 'fontsize', 14);  
21 title (sprintf ('{\bf page rank, harvard500: %d  
    hops}', n), 'fontsize', 14);  
22 drawnow;  
23 end
```

## Explanations for Code 6.3.1:

- Line 9: `rand` generates uniformly distributed pseudo-random numbers  $\in [0, 1[$
- Web graph encoded in  $\mathbf{G} \in \{0, 1\}^{N, N}$ :

$$(\mathbf{G})_{ij} = 1 \Rightarrow \text{link } j \rightarrow i ,$$



Observation: relative visit times stabilize as the number of hops in the stochastic simulation  $\rightarrow \infty$ .

The limit distribution is called **stationary distribution**/invariant measure of the Markov chain. This is what we seek.

- Numbering of pages  $1, \dots, N$ ,  $\ell_i \hat{=}$  number of links from page  $i$



•  $N \times N$ -matrix of transition probabilities page  $j \rightarrow$  page  $i$ :  $\mathbf{A} = (a_{ij})_{i,j=1}^N \in \mathbb{R}^{N,N}$

$a_{ij} \in [0, 1] \hat{=}$  probability to jump from page  $j$  to page  $i$ .

$$\Rightarrow \sum_{i=1}^N a_{ij} = 1. \quad (6.3.3)$$

A matrix  $\mathbf{A} \in [0, 1]^{N,N}$  with the property (6.3.3) is called a (column) **stochastic matrix**.

“Meaning” of  $\mathbf{A}$ : given  $\mathbf{x} \in [0, 1]^N$ ,  $\|\mathbf{x}\|_1 = 1$ , where  $x_i$  is the probability of the surfer to visit page  $i$ ,  $i = 1, \dots, N$ , at an instance  $t$  in time,  $\mathbf{y} = \mathbf{A}\mathbf{x}$  satisfies

$$y_j \geq 0, \quad \sum_{j=1}^N y_j = \sum_{j=1}^N \sum_{i=1}^N a_{ji} x_i = \sum_{i=1}^N x_i \underbrace{\sum_{j=1}^N a_{ij}}_{=1} = \sum_{i=1}^N x_i = 1.$$

►  $y_j \hat{=}$  probability for visiting page  $j$  at time  $t + \Delta t$ .

## Transition probability matrix for page rank computation

$$(\mathbf{A})_{ij} = \begin{cases} \frac{1}{N} & , \text{ if } (\mathbf{G})_{ij} = 0 \ \forall i = 1, \dots, N, \\ d/N + (1-d) \frac{(\mathbf{G})_{ij}}{\ell_j} & \text{ else.} \end{cases} \quad (6.3.4)$$

random jump to any other page      follow link


## Code 6.3.5: transition probability matrix for page rank

```

1 function A = prbuildA(G,d)
2 N = size(G,1);
3 l = full(sum(G)); idx = find(l>0);
4 s = zeros(N,1); s(idx) = 1./l(idx);
5 ds = ones(N,1)/N; ds(idx) = d/N;
6 A = ones(N,1)*ones(1,N)*diag(ds) +
   (1-d)*G*diag(s);

```

Note: special treatment  
of zero columns of  $\mathbf{G}$ , cf.  
(6.3.4)!

 Stochastic simulation based on a single surfer is *slow*. Alternatives?

Thought experiment: Instead of a single random surfer we may consider  $m \in \mathbb{N}$ ,  $m \gg 1$ , of them who visit pages independently. The fraction of time  $m \cdot T$  they all together spend on page  $i$  will obviously be the same for  $T \rightarrow \infty$  as that for a single random surfer.

Instead of counting the surfers we watch the proportions of them visiting particular web pages at an instance of time. Thus, after the  $k$ -th hop we can assign a number  $x_i^{(k)} \in [0, 1]$  to web page  $i$ , which gives the proportion of surfers currently on that page:  $x_i^{(k)} := \frac{n_i^{(k)}}{m}$ , where  $n_i^{(k)} \in \mathbb{N}_0$  designates the number of surfers on page  $i$  after the  $k$ -th hop.

R. Hiptmair  
rev 38286,  
November  
4, 2011

Now consider  $m \rightarrow \infty$ . The law of **law of large numbers** suggests that the (“infinitely many”) surfers visiting page  $j$  will move on to other pages proportional to the transition probabilities  $a_{ij}$ : in terms of proportions, for  $m \rightarrow \infty$  the stochastic evolution becomes a deterministic discrete dynamical system and we find

$$x_i^{(k+1)} = \sum_{j=1}^N a_{ij} x_j^{(k)}, \quad (6.3.6)$$

that is, the proportion of surfers ending up on page  $i$  equals the sum of the proportions on the “source pages” weighted with the transition probabilities.

Notice that (6.3.6) amounts to matrix  $\times$  vector. Thus, writing  $\mathbf{x}^{(0)} \in [0, 1]^N$ ,  $\|\mathbf{x}^{(0)}\| = 1$  for the initial distribution of the surfers on the net we find

$$\mathbf{x}^{(k)} = \mathbf{A}^k \mathbf{x}^{(0)}$$

will be their mass distribution after  $k$  hops. If the limit exists, the  $i$ -th component of  $\mathbf{x}^* := \lim_{k \rightarrow \infty} \mathbf{x}^{(k)}$  tells us which fraction of the (infinitely many) surfers will be visiting page  $i$  most of the time. Thus,  $\mathbf{x}^*$  yields the stationary distribution of the Markov chain.

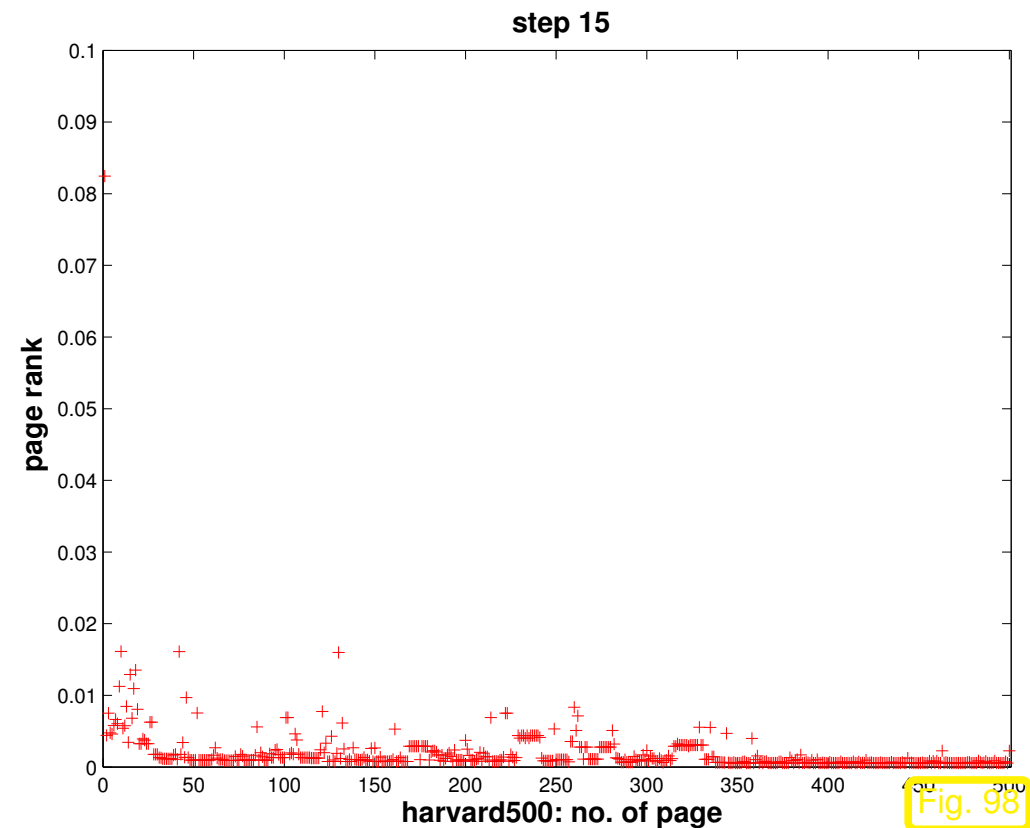
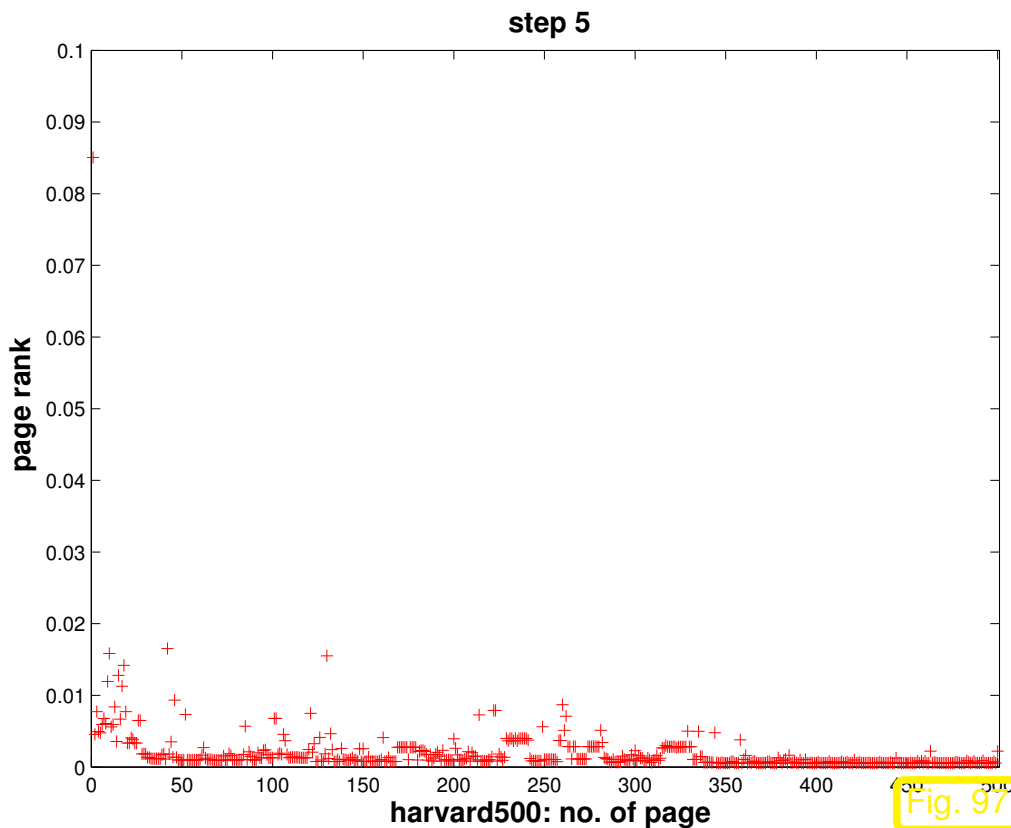
### Code 6.3.7: tracking fractions of many surfers

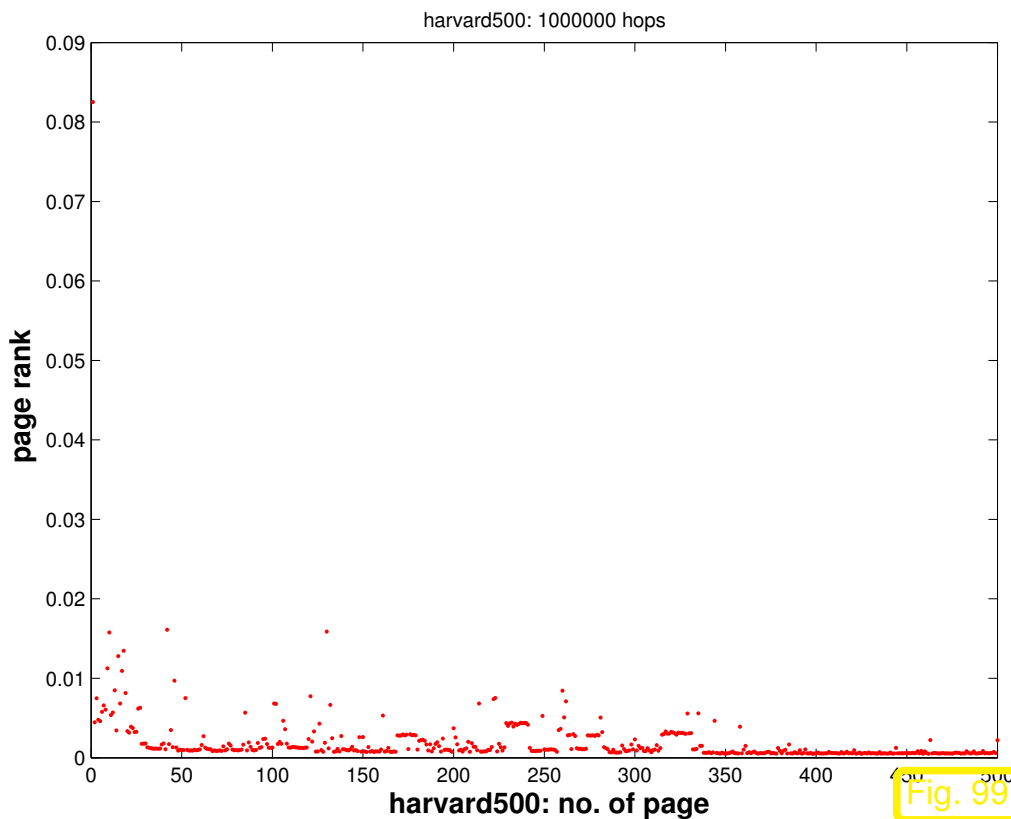
```

1 function prpowitsim(d,Nsteps)
2 % MATLAB way of specifying Default arguments
3 if (nargin < 2), Nsteps = 5; end
4 if (nargin < 1), d = 0.15; end
5 % load connectivity matrix and build transition matrix
6 load harvard500.mat; A = prbuildA(G,d);
7 N = size(A,1); x = ones(N,1)/N;
8
9 figure ('position', [0 0 1200 1000]);
10 plot (1:N, x, 'r+'); axis ([0 N+1 0 0.1]);

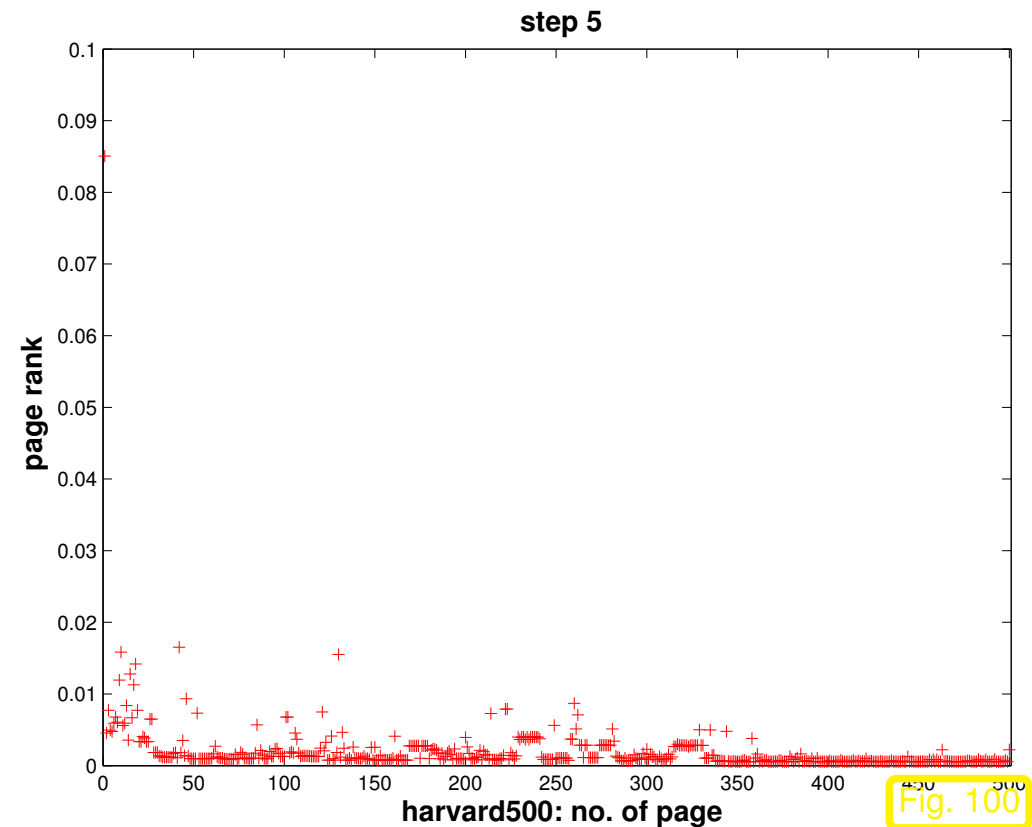
```

```
11 % Plain power iteration for stochastic matrix A
12 for l=1:Nsteps
13     pause; x = A*x; plot(1:N,x,'r+'); axis([0 N+1 0 0.1]);
14     title(sprintf('\bf step %d',l),'fontsize',14);
15     xlabel('\bf harvard500: no. of page','fontsize',14);
16     ylabel('\bf page rank','fontsize',14); drawnow;
17 end
```





Single surfer stochastic simulation



Power method, Code 6.3.6

Observation: Convergence of the  $\mathbf{x}^{(k)} \rightarrow \mathbf{x}^*$ , and the limit must be a fixed point of the iteration function:

$$\triangleright \quad \mathbf{A}\mathbf{x}^* = \mathbf{x}^* \quad \Rightarrow \quad \mathbf{x}^* \in \text{Eig}_{\mathbf{A}}(1).$$

Does  $\mathbf{A}$  possess an eigenvalue  $= 1$ ? Does the associated eigenvector really provide a probability distribution (after scaling), that is, are all of its entries non-negative? Is this probability distribution unique? To answer these questions we have to study the matrix  $\mathbf{A}$ :

For every stochastic matrix  $\mathbf{A}$ , by definition (6.3.3)

$$\begin{aligned} \mathbf{A}^T \mathbf{1} &= \mathbf{1} \quad \stackrel{(6.1.3)}{\Rightarrow} \quad 1 \in \sigma(\mathbf{A}), \\ (2.5.8) \quad \Rightarrow \quad \|\mathbf{A}\|_1 &= 1 \quad \stackrel{\text{Thm 6.1.4}}{\Rightarrow} \quad \rho(\mathbf{A}) = 1, \end{aligned}$$

where  $\rho(\mathbf{A})$  is the spectral radius of the matrix  $\mathbf{A}$ , see Sect. 6.1.

For  $\mathbf{r} \in \text{Eig}_{\mathbf{A}}(1)$ , that is,  $\mathbf{A}\mathbf{r} = \mathbf{r}$ , denote by  $|\mathbf{r}|$  the vector  $(|r_i|)_{i=1}^N$ . Since all entries of  $\mathbf{A}$  are non-negative, we conclude by the triangle inequality that  $\|\mathbf{A}\mathbf{r}\|_1 \leq \|\mathbf{A}|\mathbf{r}|\|_1$

$$\begin{aligned} \Rightarrow \quad 1 = \|\mathbf{A}\|_1 &= \sup_{\mathbf{x} \in \mathbb{R}^N} \frac{\|\mathbf{A}\mathbf{x}\|_1}{\|\mathbf{x}\|_1} \geq \frac{\|\mathbf{A}|\mathbf{r}|\|_1}{\||\mathbf{r}|\|_1} \geq \frac{\|\mathbf{A}\mathbf{r}\|_1}{\|\mathbf{r}\|_1} = 1. \\ \Rightarrow \quad \|\mathbf{A}|\mathbf{r}|\|_1 &= \|\mathbf{A}\mathbf{r}\|_1 \quad \stackrel{\text{if } a_{ij} > 0}{\Rightarrow} \quad |\mathbf{r}| = \pm \mathbf{r}. \end{aligned}$$

Hence, different components of  $\mathbf{r}$  cannot have opposite sign, which means, that  $\mathbf{r}$  can be chosen to have non-negative entries, if the entries of  $\mathbf{A}$  are strictly positive, which is the case for  $\mathbf{A}$  from (6.3.4). After normalization  $\|\mathbf{r}\|_1 = 1$  the eigenvector can be regarded as a probability distribution on  $\{1, \dots, N\}$ .

If  $\mathbf{A}\mathbf{r} = \mathbf{r}$  and  $\mathbf{A}\mathbf{s} = \mathbf{s}$  with  $(\mathbf{r})_i \geq 0$ ,  $(\mathbf{s})_i \geq 0$ ,  $\|\mathbf{r}\|_1 = \|\mathbf{s}\|_1 = 1$ , then  $\mathbf{A}(\mathbf{r} - \mathbf{s}) = \mathbf{r} - \mathbf{s}$ . Hence, by the above considerations, also all the entries of  $\mathbf{r} - \mathbf{s}$  are either non-negative or non-positive. By the

assumptions on  $\mathbf{r}$  and  $\mathbf{s}$  this is only possible, if  $\mathbf{r} - \mathbf{s} = \mathbf{0}$ . We conclude that

$$\mathbf{A} \in ]0, 1]^{N,N} \text{ stochastic} \Rightarrow \dim \text{Eig}_{\mathbf{A}}(1) = 1. \quad (6.3.8)$$

Sorting the pages according to the size of the corresponding entries in  $\mathbf{r}$  yields the famous “page rank”.

Code 6.3.9: computing page rank vector  $\mathbf{r}$  via `eig`

```

1 function prevp
2 load harvard500.mat; d = 0.15;
3 [V,D] = eig(prbuildA(G,d));
4
5 figure; r = V(:,1); N = length(r);
6 plot(1:N, r/sum(r), 'm. '); axis([0 N+1 0 0.1]);
7 xlabel('{\bf harvard500: no. of
   page}', 'fontsize', 14);
8 ylabel('{\bf entry of r-vector}', 'fontsize', 14);
9 title('harvard 500: Perron-Frobenius vector');
10 print -depsc2 '../PICTURES/prevp.eps';

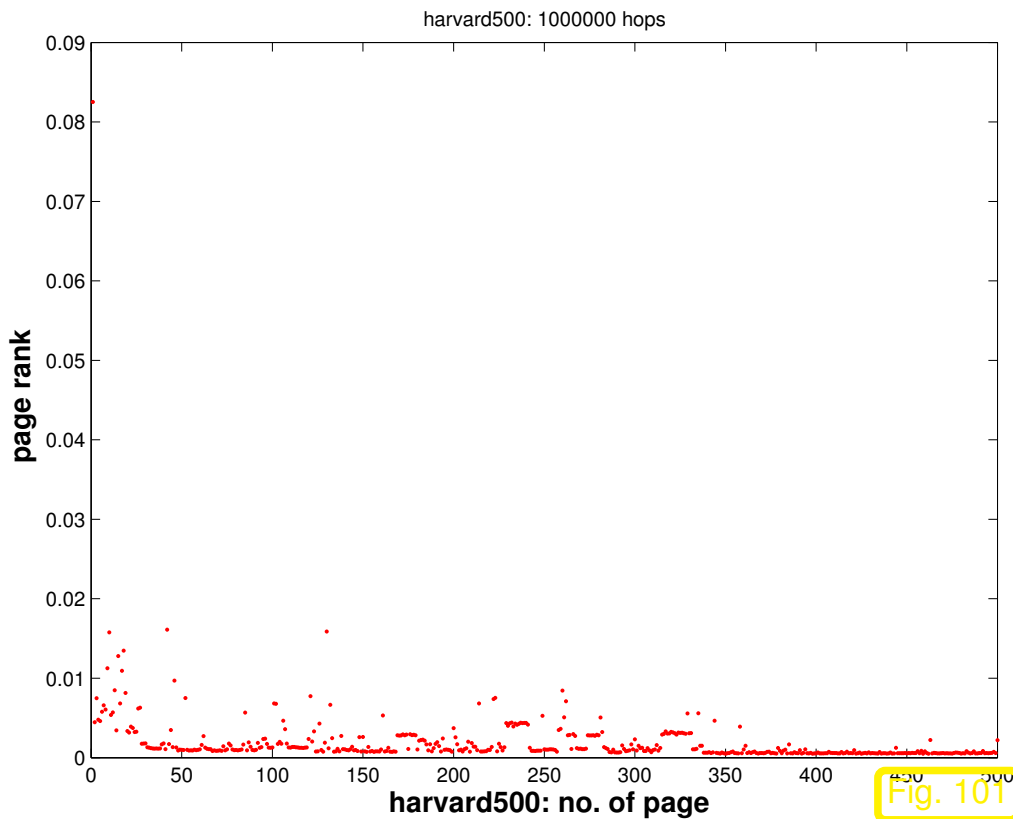
```

Plot of entries of  
*unique* vector  $\mathbf{r} \in \mathbb{R}^N$  with

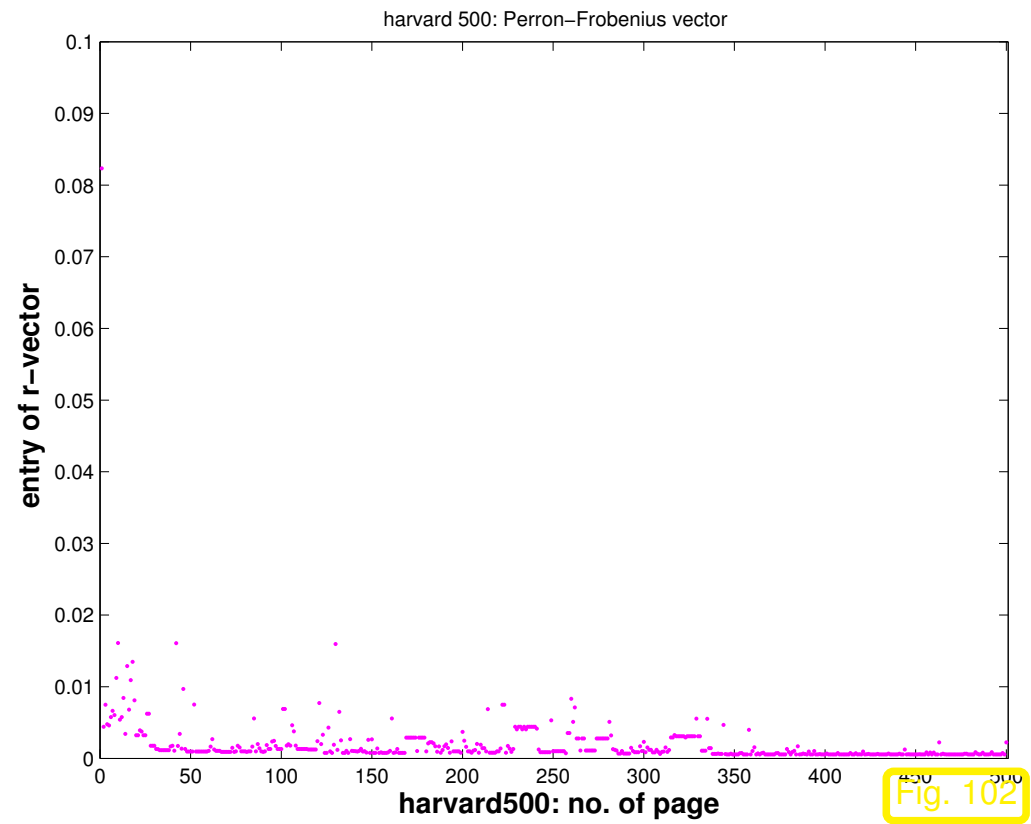
$$0 \leq (\mathbf{r})_i \leq 1, \\ \|\mathbf{r}\|_1 = 1, \\ \boxed{\mathbf{A}\mathbf{r} = \mathbf{r}}.$$

Inefficient imple-  
mentation!





stochastic simulation



eigenvector computation

The possibility to compute the stationary probability distribution of a Markov chain through an eigenvector of the transition probability matrix is due to a property of stationary Markov chains called **ergodicity**.

$\hat{\mathbf{A}}$   $\hat{=}$  page rank transition probability matrix, see Code 6.3.4,  $d = 0.15$ , harvard500 example.

Errors:

$$\left\| \mathbf{A}^k \mathbf{x}_0 - \mathbf{r} \right\|_1,$$

with  $\mathbf{x}_0 = \mathbf{1}/N$ ,  $N = 500$ .

We observe **linear convergence!** ( $\rightarrow$  Def. 4.1.6, iteration error vs. iteration count  $\approx$  straight line lin-log plot)

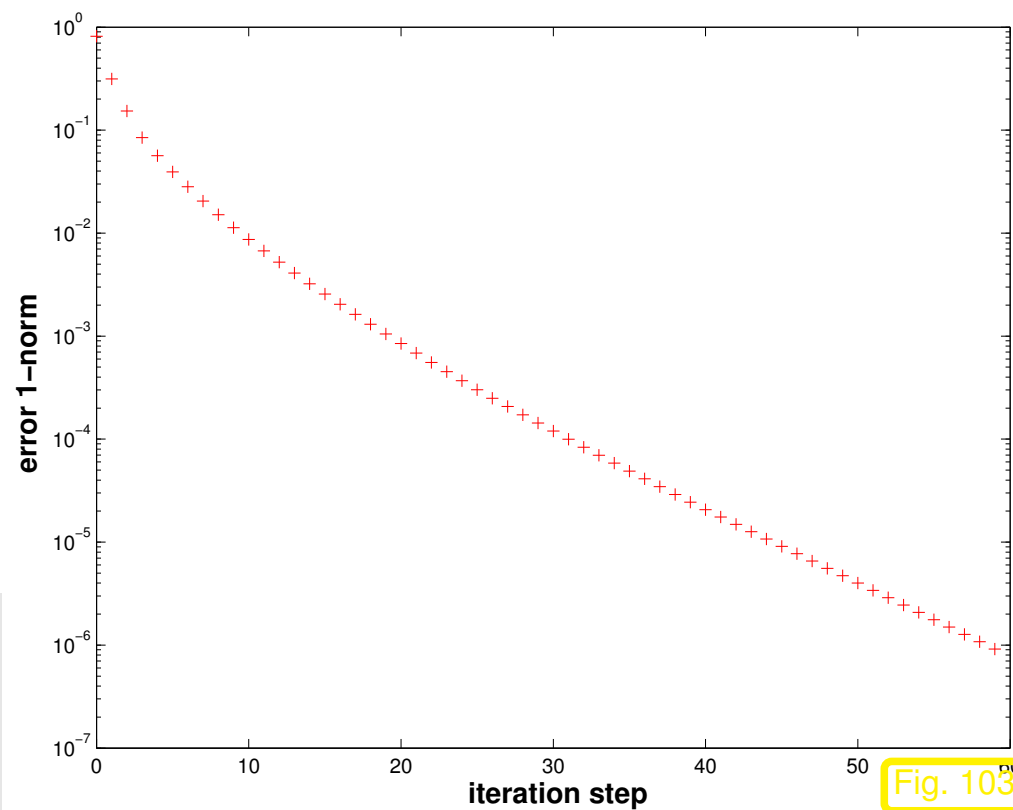


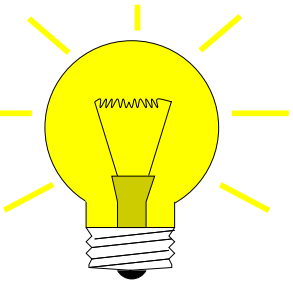
Fig. 103

The computation of page rank amounts to finding the eigenvector of the matrix  $\mathbf{A}$  of transition probabilities that belongs to its *largest* eigenvalue 1. This is addressed by an important class of practical eigenvalue problems:

Task: given  $\mathbf{A} \in \mathbb{K}^{n,n}$ , find **largest** (in modulus) eigenvalue of  $\mathbf{A}$   
and (an) associated eigenvector.

Idea: (suggested by page rank computation, Code 6.3.6)

Iteration:  $\mathbf{z}^{(k+1)} = \mathbf{A}\mathbf{z}^{(k)}$ ,  $\mathbf{z}^{(0)}$  arbitrary



*Example 6.3.10* (Power iteration).  $\rightarrow$  Ex. 6.3.1

Try the above iteration for general  $10 \times 10$ -matrix, largest eigenvalue 10, algebraic multiplicity 1.

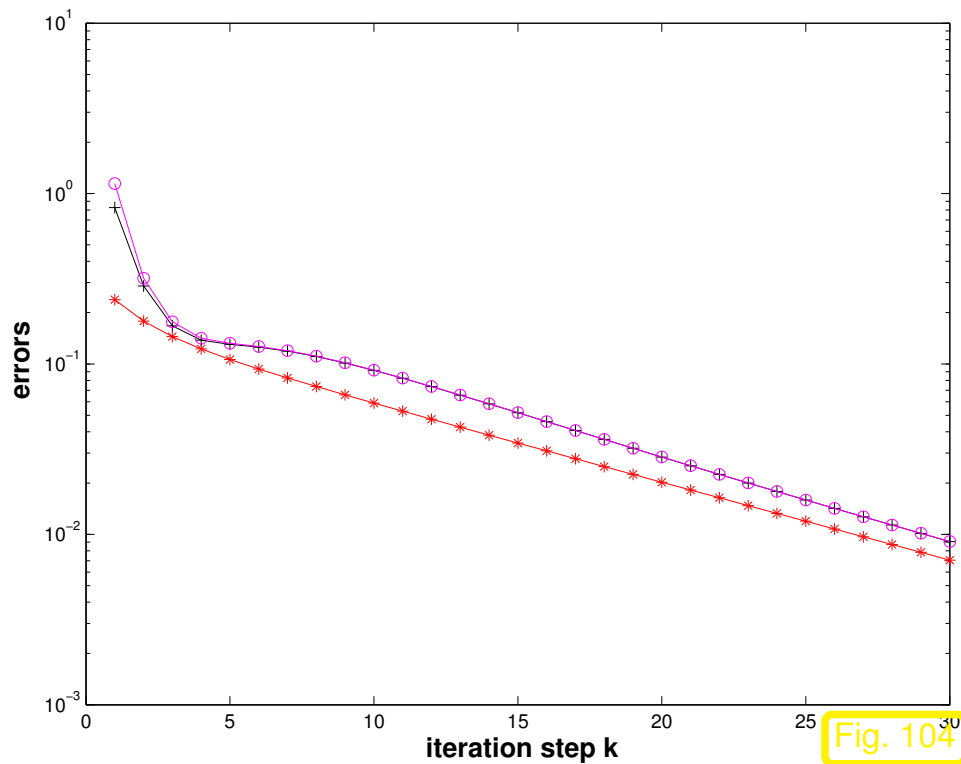


Fig. 104

```
d = (1:10)'; n = length(d);
S = triu(diag(n:-1:1,0)+...
ones(n,n));
A = S*diag(d,0)*inv(S);
```

◁ error norm  $\left\| \frac{\mathbf{z}^{(k)}}{\|\mathbf{z}^{(k)}\|} - (\mathbf{S})_{:,10} \right\|$   
 (Note:  $(\mathbf{S})_{:,10} \hat{=}$  eigenvector for eigenvalue 10)

$\mathbf{z}^{(0)}$  = random vector

Observation: **linear convergence** of (normalized) eigenvectors!



► Suggests **direct power method** (*ger.:* Potenzmethode): iterative method ( $\rightarrow$  Sect. 4.1)

$$\begin{aligned} \text{initial guess: } & \mathbf{z}^{(0)} \text{ "arbitrary" ,} \\ \text{next iterate: } & \mathbf{w} := \mathbf{A}\mathbf{z}^{(k-1)} , \quad \mathbf{z}^{(k)} := \frac{\mathbf{w}}{\|\mathbf{w}\|_2} , \quad k = 1, 2, \dots . \end{aligned} \quad (6.3.11)$$

Note: the “normalization” of the iterates in (6.3.11) does not change anything (in exact arithmetic) and helps *avoid overflow* in floating point arithmetic.

Computational effort:  $1 \times \text{matrix} \times \text{vector}$  per step  $\triangleright$  inexpensive for sparse matrices

A persuasive theoretical justification for the direct power method:

Assume  $\mathbf{A} \in \mathbb{K}^{n,n}$  to be **diagonalizable**:

$$\Leftrightarrow \exists \text{ basis } \{\mathbf{v}_1, \dots, \mathbf{v}_n\} \text{ of eigenvectors of } \mathbf{A}: \quad \mathbf{A}\mathbf{v}_j = \lambda_j \mathbf{v}_j, \lambda_j \in \mathbb{C}.$$

Assume

$$|\lambda_1| \leq |\lambda_2| \leq \cdots \leq |\lambda_{n-1}| < |\lambda_n| \quad , \quad \|\mathbf{v}_j\|_2 = 1 . \quad (6.3.12)$$

Key observations for power iteration (6.3.11)

$$\mathbf{z}^{(k)} = \mathbf{A}^k \mathbf{z}^{(0)} \quad (\rightarrow \text{name "power method"}) \quad (6.3.13)$$

$$\mathbf{z}^{(0)} = \sum_{j=1}^n \zeta_j \mathbf{v}_j \quad \Rightarrow \quad \mathbf{z}^{(k)} = \sum_{j=1}^n \zeta_j \lambda_j^k \mathbf{v}_j . \quad (6.3.14)$$

Due to (6.3.12) for large  $k \gg 1$  ( $\Rightarrow |\lambda_n^k| \gg |\lambda_j^k|$  for  $j \neq n$ ) the contribution of  $\mathbf{v}_n$  (size  $\zeta_n \lambda_n^k$ ) in the *eigenvector expansion* (6.3.14) will be much larger than the contribution (size  $\zeta_n \lambda_j^k$ ) of any other eigenvector (, if  $\zeta_n \neq 0$ ): the eigenvector for  $\lambda_n$  will swamp all other for  $k \rightarrow \infty$ .

Further (6.3.14) nurtures expectation:  $\mathbf{v}_n$  will become dominant in  $\mathbf{z}^{(k)}$  the faster, the better  $|\lambda_n|$  is separated from  $|\lambda_{n-1}|$ , see Thm. 6.3.19 for rigorous statement.

$\mathbf{z}^{(k)}$   $\rightarrow$  eigenvector, but how do we get the associated eigenvalue  $\lambda_n$  ?

When (6.3.11) has converged, two common ways to recover  $\lambda_{\max}$   $\rightarrow$  [13, Alg. 7.20]

- ①  $\mathbf{A}\mathbf{z}^{(k)} \approx \lambda_{\max}\mathbf{z}^{(k)} \quad \triangleright \quad |\lambda_n| \approx \frac{\|\mathbf{A}\mathbf{z}^{(k)}\|}{\|\mathbf{z}^{(k)}\|}$  (modulus only!)
- ②  $\lambda_{\max} \approx \operatorname{argmin}_{\theta \in \mathbb{R}} \left\| \mathbf{A}\mathbf{z}^{(k)} - \theta\mathbf{z}^{(k)} \right\|_2^2 \quad \triangleright \quad \lambda_{\max} \approx \frac{(\mathbf{z}^{(k)})^H \mathbf{A}\mathbf{z}^{(k)}}{\|\mathbf{z}^{(k)}\|_2^2}.$

This latter formula is extremely useful, which has earned it a special name:

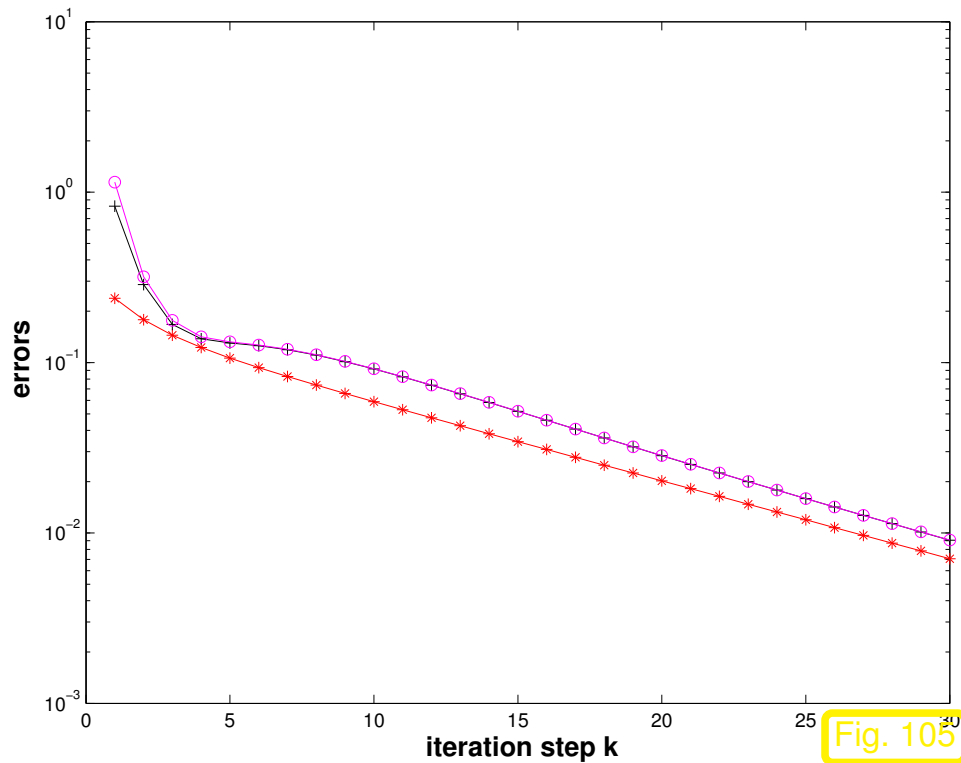
**Definition 6.3.15.** For  $\mathbf{A} \in \mathbb{K}^{n,n}$ ,  $\mathbf{u} \in \mathbb{K}^n$  the *Rayleigh quotient* is defined by

$$\rho_{\mathbf{A}}(\mathbf{u}) := \frac{\mathbf{u}^H \mathbf{A}\mathbf{u}}{\mathbf{u}^H \mathbf{u}}.$$

An immediate consequence of the definitions:

$$\lambda \in \sigma(\mathbf{A}) \quad , \quad \mathbf{z} \in \text{Eig}_\lambda(\mathbf{A}) \quad \Rightarrow \quad \rho_{\mathbf{A}}(\mathbf{z}) = \lambda . \quad (6.3.16)$$

Example 6.3.17 (Direct power method).  $\rightarrow$  Ex. 6.3.17 cnt'd



```
n = length(d);
S = triu(diag(n:-1:1, 0) + ...
        ones(n, n));
A = S*diag(d, 0)*inv(S);
```

```
d = (1:10)';
```

○ : error  $|\lambda_n - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})|$

△ \* : error norm  $\|\mathbf{z}^{(k)} - \mathbf{s}_{\cdot, n}\|$

+ :  $\left| \lambda_n - \frac{\|\mathbf{A}\mathbf{z}^{(k-1)}\|_2}{\|\mathbf{z}^{(k-1)}\|_2} \right|$

$\mathbf{z}^{(0)}$  = random vector



## Test matrices:

$$\textcircled{1} \ d = (1:10)'; \quad \Rightarrow \quad |\lambda_{n-1}| : |\lambda_n| = 0.9$$

$$\textcircled{2} \ d = [\text{ones}(9,1); 2]; \quad \Rightarrow \quad |\lambda_{n-1}| : |\lambda_n| = 0.5$$

$$\textcircled{3} \ d = 1 - 2.^{- (1:0.5:5)'}; \quad \Rightarrow \quad |\lambda_{n-1}| : |\lambda_n| = 0.9866$$

## Code 6.3.18: Investigating convergence of direct power method

```

1 % Demonstration of direct power method for Ex. 6.3.17
2 maxit = 30; d = [1:10]'; n = length(d);
3 % Initialize the matrix A
4 S = triu(diag(n:-1:1,0)+ones(n,n));
5 A = S*diag(d,0)*inv(S);
6 % This calculates the exact eigenvalues (for error calculation)
7 [V,D] = eig(A);
8
9 k = find(d == max(abs(d)));
0 if (length(k) > 1), error('No single largest EV'); end;
1 ev = X(:,k(1)); ev = ev/norm(ev); ev
2 ew = d(k(1)); ew
3 if (ew < 0), sgn = -1; else sgn = 1; end
4
5 z = rand(n,1); z = z/norm(z);
6 s = 1;
7 res = [];
8
9 % Actual direct power iteration

```

```

20 for i=1:maxit
21     w = A*z; l = norm(w); rq = real(dot(w,z)); z = w/l;
22     res = [res;i,l,norm(s*z-ev),abs(l-abs(ew)),abs(sgn*rq-ew)];
23     s = s * sgn;
24 end
25
26 % Plot the result
27 semilogy(res(:,1),res(:,3),'r-*',res(:,1),res(:,4),'k-+',res(:,1),res(:,5),'m-o')
28 xlabel('\bf iteration step k','FontSize',14);
29 ylabel('\bf errors','FontSize',14);
30 print -deps2c '../PICTURES/pm1.eps';

```

$$\rho_{EV}^{(k)} := \frac{\|\mathbf{z}^{(k)} - \mathbf{s}_{:,n}\|}{\|\mathbf{z}^{(k-1)} - \mathbf{s}_{:,n}\|},$$

$$\rho_{EW}^{(k)} := \frac{|\rho_{\mathbf{A}}(\mathbf{z}^{(k)}) - \lambda_n|}{|\rho_{\mathbf{A}}(\mathbf{z}^{(k-1)}) - \lambda_n|}.$$

	①		②		③	
$k$	$\rho_{EV}^{(k)}$	$\rho_{EW}^{(k)}$	$\rho_{EV}^{(k)}$	$\rho_{EW}^{(k)}$	$\rho_{EV}^{(k)}$	$\rho_{EW}^{(k)}$
22	0.9102	0.9007	0.5000	0.5000	0.9900	0.9781
23	0.9092	0.9004	0.5000	0.5000	0.9900	0.9791
24	0.9083	0.9001	0.5000	0.5000	0.9901	0.9800
25	0.9075	0.9000	0.5000	0.5000	0.9901	0.9809
26	0.9068	0.8998	0.5000	0.5000	0.9901	0.9817
27	0.9061	0.8997	0.5000	0.5000	0.9901	0.9825
28	0.9055	0.8997	0.5000	0.5000	0.9901	0.9832
29	0.9049	0.8996	0.5000	0.5000	0.9901	0.9839
30	0.9045	0.8996	0.5000	0.5000	0.9901	0.9844



**Theorem 6.3.19** (Convergence of direct power method).  $\rightarrow$  [13, Thm. 25.1]

Let  $\lambda_n > 0$  be the largest (in modulus) eigenvalue of  $\mathbf{A} \in \mathbb{K}^{n,n}$  and have (algebraic) multiplicity 1. Let  $\mathbf{v}, \mathbf{y}$  be the left and right eigenvectors of  $\mathbf{A}$  for  $\lambda_n$  normalized according to  $\|\mathbf{y}\|_2 = \|\mathbf{v}\|_2 = 1$ . Then there is convergence

$$\|\mathbf{A}\mathbf{z}^{(k)}\|_2 \rightarrow \lambda_n, \quad \mathbf{z}^{(k)} \rightarrow \pm \mathbf{v} \text{ linearly with rate } \frac{|\lambda_{n-1}|}{|\lambda_n|},$$

where  $\mathbf{z}^{(k)}$  are the iterates of the direct power iteration and  $\mathbf{y}^H \mathbf{z}^{(0)} \neq 0$  is assumed.

*Remark 6.3.20* (Initial guess for power iteration).

roundoff errors  $\blacktriangleright$   $\mathbf{y}^H \mathbf{z}^{(0)} \neq 0$  always satisfied in practical computations



Usual (not the best!) choice for  $\mathbf{x}^{(0)}$  = random vector

*Remark 6.3.21* (Termination criterion for direct power iteration). ( $\rightarrow$  Sect. 4.1.2)

Adaptation of a posteriori termination criterion (4.2.20)

“relative change”  $\leq \text{tol}$ :

$$\left\{ \begin{array}{l} \min \left\| \mathbf{z}^{(k)} \pm \mathbf{z}^{(k-1)} \right\| \leq (1/L - 1)\text{tol} , \\ \left| \frac{\|\mathbf{Az}^{(k)}\|}{\|\mathbf{z}^{(k)}\|} - \frac{\|\mathbf{Az}^{(k-1)}\|}{\|\mathbf{z}^{(k-1)}\|} \right| \leq (1/L - 1)\text{tol} \text{ see (4.1.24) .} \end{array} \right.$$

Estimated rate of convergence



## 6.3.2 Inverse Iteration [13, Sect. 7.6], [51, Sect. 5.3.2]

*Example 6.3.22* (Image segmentation).

Given: gray-scale image: intensity matrix  $\mathbf{P} \in \{0, \dots, 255\}^{m,n}$ ,  $m, n \in \mathbb{N}$   
 $((\mathbf{P})_{ij} \leftrightarrow \text{pixel}, 0 \hat{=} \text{black}, 255 \hat{=} \text{white})$

Code 6.3.23: loading and displaying an image

Loading and displaying images in MATLAB

```
1 M = imread('eth.pbm');  
2 [m,n] = size(M);  
3 fprintf('%dx%d grey scale pixel image\n', m, n);  
4 figure; image(M); title('ETH view');  
5 col = [0:1/215:1]' * [1,1,1]; colormap(col);
```

(Fuzzy) task: Local segmentation

Find connected patches of image of the same shade/color

More general segmentation problem (non-local): identify parts of the image, not necessarily connected, with the same texture.

Next: Statement of (rigorously defined) problem, *cf.* Sect. 2.5.2:

Preparation: Numbering of pixels  $1 \dots, mn$ , e.g, **lexicographic numbering**:

- pixel set  $\mathcal{V} := \{1 \dots, nm\}$
- indexing:  $\text{index}(\text{pixel}_{i,j}) = (i - 1)n + j$

 notation:  $p_k := (\mathbf{P})_{ij}$ , if  $k = \text{index}(\text{pixel}_{i,j}) = (i - 1)n + j$ ,  $k = 1, \dots, N := mn$

Local similarity matrix:

$$\mathbf{W} \in \mathbb{R}^{N,N}, \quad N := mn, \quad (6.3.24)$$

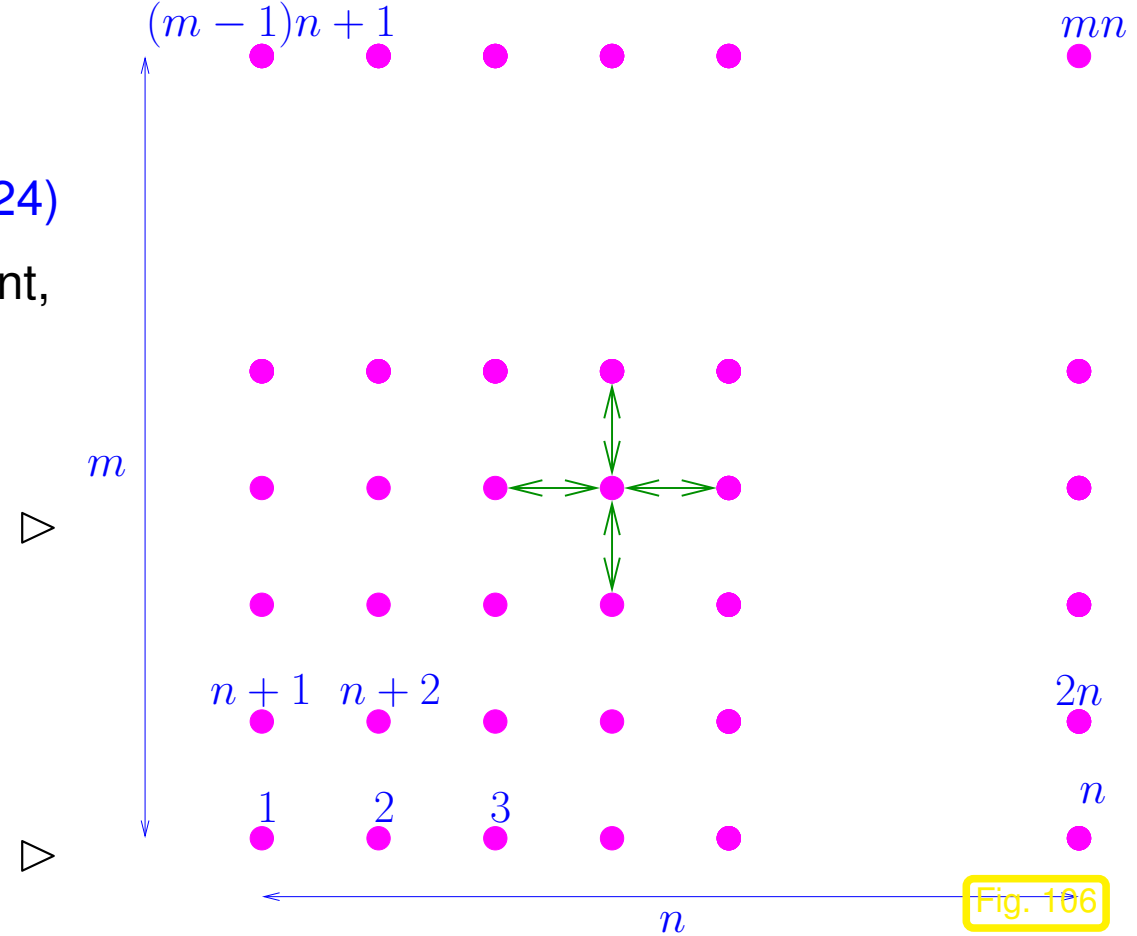
$$(\mathbf{W})_{ij} = \begin{cases} 0 & , \text{ if pixels } i, j \text{ not adjacent,} \\ 0 & , \text{ if } i = j, \\ \sigma(p_i, p_j) & , \text{ if pixels } i, j \text{ adjacent.} \end{cases}$$

$\leftrightarrow \hat{=}$  adjacent pixels

Similarity function, e.g., with  $\alpha > 0$

$$\sigma(x, y) := \exp(-\alpha(x - y)^2), \quad x, y \in \mathbb{R}.$$

Lexicographic numbering



The entries of the matrix  $\mathbf{W}$  measure the “similarity” of neighboring pixels: if  $(\mathbf{W})_{ij}$  is large, they encode (almost) the same intensity, if  $(\mathbf{W})_{ij}$  is close to zero, then they belong to parts of the picture with very different brightness. In the latter case, the boundary of the segment may separate the two pixels.

*Definition 6.3.25 (Normalized cut).* ( $\rightarrow$  [58, Sect. 2])

For  $\mathcal{X} \subset \mathcal{V}$  define the *normalized cut* as

$$\text{Ncut}(\mathcal{X}) := \frac{\text{cut}(\mathcal{X})}{\text{weight}(\mathcal{X})} + \frac{\text{cut}(\mathcal{X})}{\text{weight}(\mathcal{V} \setminus \mathcal{X})},$$

with

$$\text{cut}(\mathcal{X}) := \sum_{i \in \mathcal{X}, j \notin \mathcal{X}} w_{ij}, \quad \text{weight}(\mathcal{X}) := \sum_{i \in \mathcal{X}, j \in \mathcal{X}} w_{ij}.$$

In light of local similarity relationship:

- $\text{cut}(\mathcal{X})$  big  $\Rightarrow$  substantial similarity of pixels across interface between  $\mathcal{X}$  and  $\mathcal{V} \setminus \mathcal{X}$ .
- $\text{weight}(\mathcal{X})$  big  $\Rightarrow$  a lot of similarity of adjacent pixels inside  $\mathcal{X}$ .

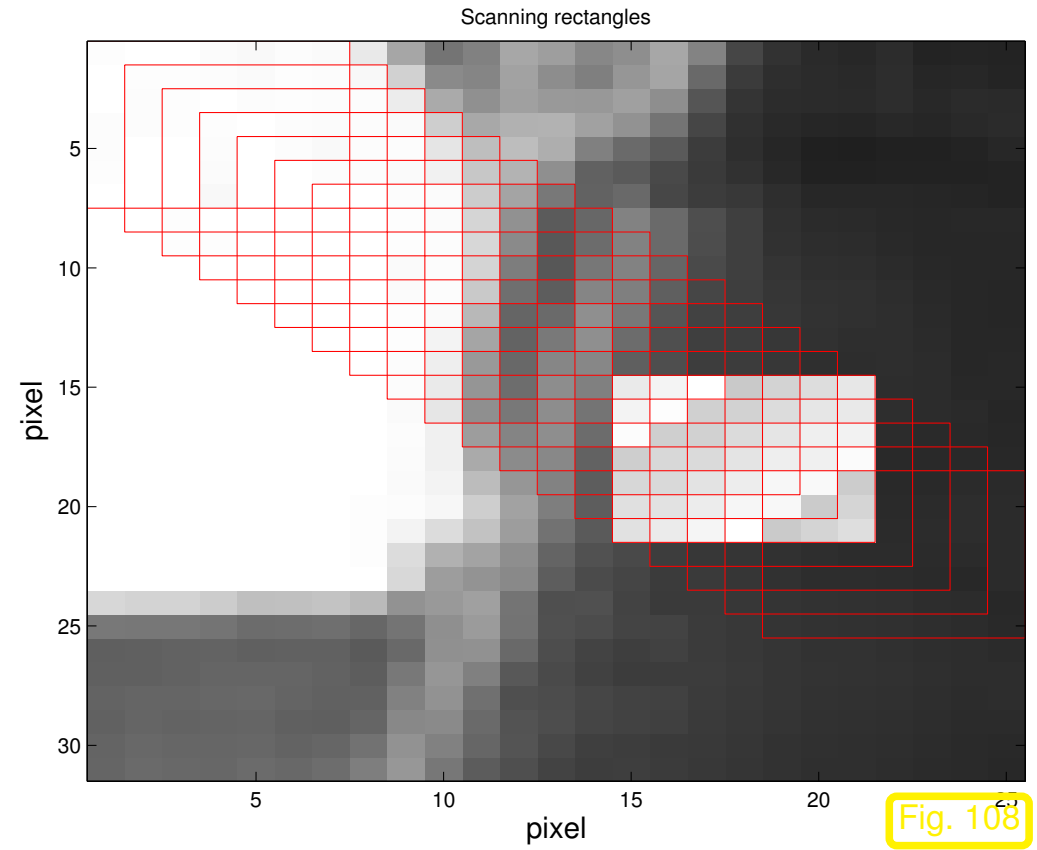
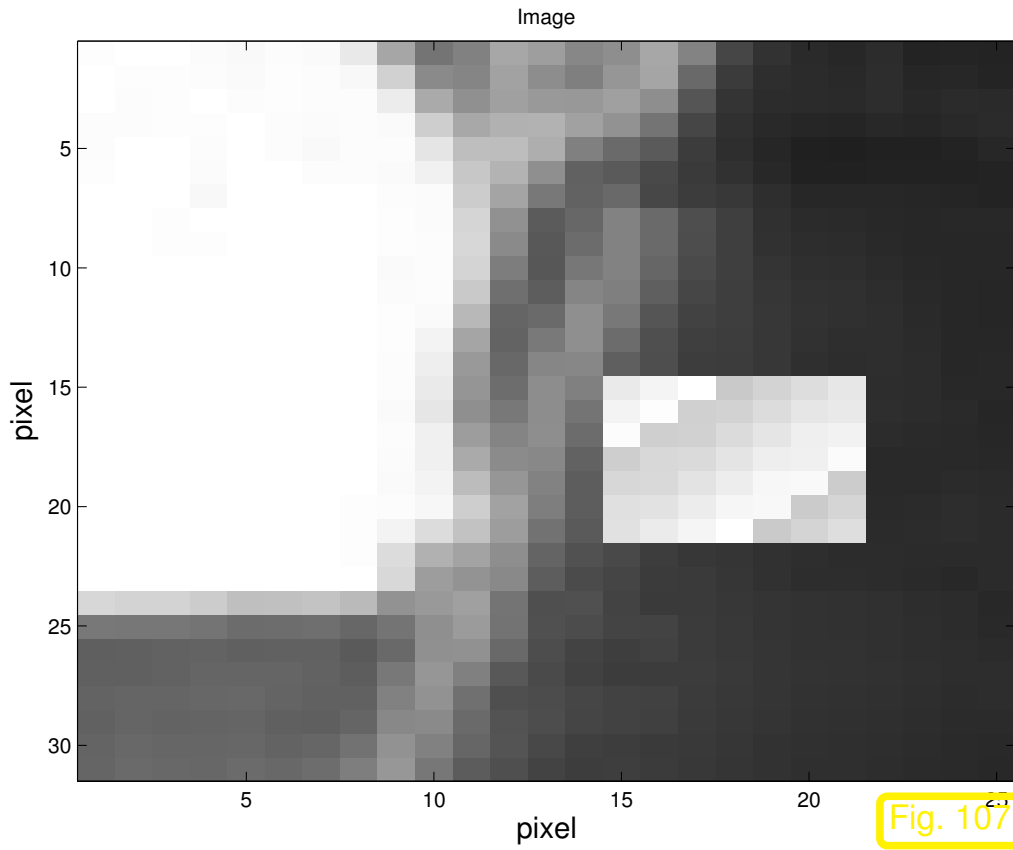
► **Segmentation problem** (rigorous statement):

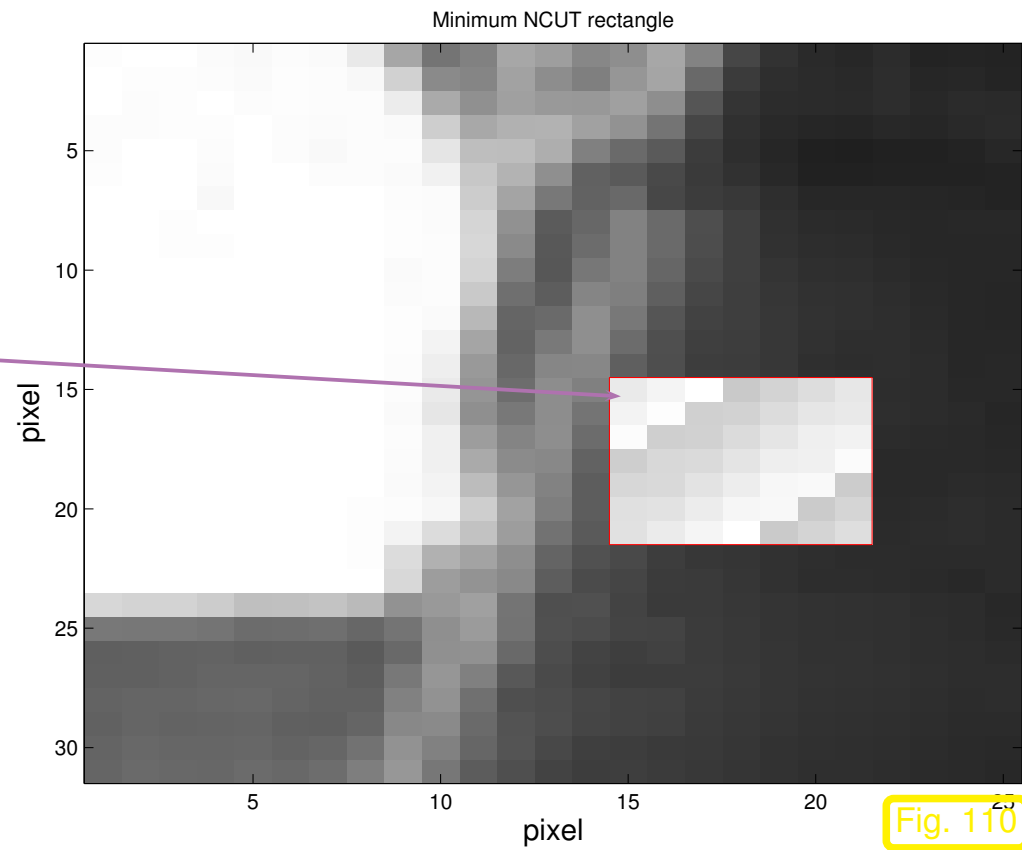
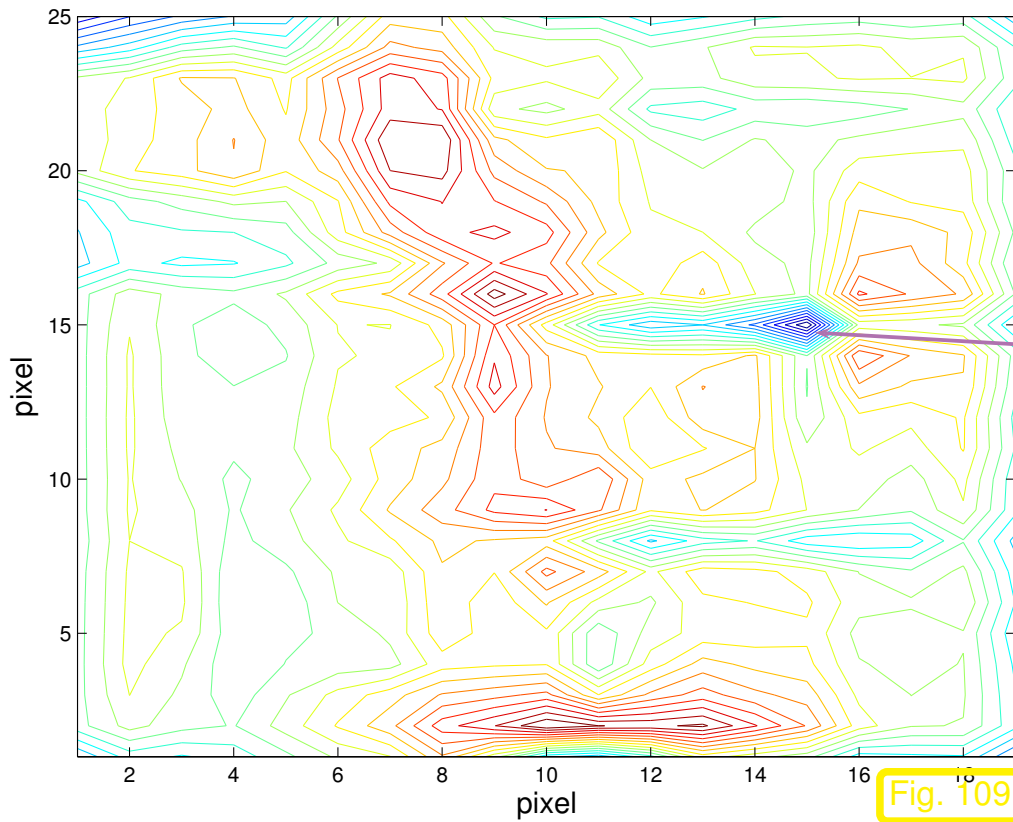
$$\text{find } \mathcal{X}^* \subset \mathcal{V}: \mathcal{X}^* = \underset{\mathcal{X} \subset \mathcal{V}}{\text{argmin}} \text{Ncut}(\mathcal{X}). \quad (6.3.26)$$





# NP-hard combinatorial optimization problem !





△  $N_{\text{cut}}(\mathcal{X})$  for pixel subsets  $\mathcal{X}$  defined by sliding rectangles, see Fig. 108.

*Equivalent reformulation:*

**indicator function:**  $z : \{1, \dots, N\} \mapsto \{-1, 1\}$ ,  $z_i := z(i) = \begin{cases} 1 & , \text{if } i \in \mathcal{X} , \\ -1 & , \text{if } i \notin \mathcal{X} . \end{cases}$  (6.3.27)

$$\blacktriangleright \quad \text{Ncut}(\mathcal{X}) = \frac{\sum_{z_i > 0, z_j < 0} -w_{ij} z_i z_j}{\sum_{z_i > 0} d_i} + \frac{\sum_{z_i > 0, z_j < 0} -w_{ij} z_i z_j}{\sum_{z_i < 0} d_i}, \quad (6.3.28)$$

$$d_i = \sum_{j \in \mathcal{V}} w_{ij} = \text{weight}(\{i\}). \quad (6.3.29)$$

Sparse matrices:

$$\mathbf{D} := \text{diag}(d_1, \dots, d_N) \in \mathbb{R}^{N,N}, \quad \mathbf{A} := \mathbf{D} - \mathbf{W} = \mathbf{A}^\top. \quad (6.3.30)$$

Summary: (obvious) properties of these matrices

- $\mathbf{A}$  has positive diagonal and non-positive off-diagonal entries.
- $\mathbf{A}$  is **diagonally dominant** ( $\rightarrow$  Def. 2.7.7)  $\triangleright$   $\mathbf{A}$  is positive semidefinite by Lemma 2.7.12.
- $\mathbf{A}$  has row sums = 0:

$$\mathbf{1}^\top \mathbf{A} = \mathbf{A} \mathbf{1} = \mathbf{0}. \quad (6.3.31)$$

Code 6.3.32: assembly of **A**, **D**

```
1 function [A,D] = imgsepmat(P)
2 P = double(P); [n,m] = size (P);
3 spdata = zeros (4*n*m,1); spidx = zeros (4*n*m,2);
4 k = 1;
5 for ni=1:n
6     for mi=1:m
7         mni = (mi-1)*n+ni;
8         if (ni-1>0), spidx(k,:) = [mni,mni-1];
9             spdata(k) = Sfun(P(ni,mi),P(ni-1,mi));
10            k = k + 1;
11        end
12        if (ni+1<=n), spidx(k,:) = [mni,mni+1];
13            spdata(k) = Sfun(P(ni,mi),P(ni+1,mi));
14            k = k + 1;
15        end
16        if (mi-1>0), spidx(k,:) = [mni,mni-n];
17            spdata(k) = Sfun(P(ni,mi),P(ni,mi-1));
18            k = k + 1;
19        end
20        if (mi+1<=m), spidx(k,:) = [mni,mni+n];
21            spdata(k) = Sfun(P(ni,mi),P(ni,mi+1));
```

```

22     k = k + 1;
23     end
24 end
25 end
26 % Efficient initialization, see Sect. 2.6.2, Ex. 2.6.11
27 W = sparse (spidx(1:k-1,1), spidx(1:k-1,2), spdata(1:k-1), n*m, n*m);
28 D = spdiags (full (sum (W')'), [0], n*m, n*m);
29 A = D-W;

```

*Lemma 6.3.33 (Ncut and Rayleigh quotient). (→ [58, Sect. 2])*

*With  $\mathbf{z} \in \{-1, 1\}^N$  according to (6.3.27) there holds*

$$\text{Ncut}(\mathcal{X}) = \frac{\mathbf{y}^\top \mathbf{A} \mathbf{y}}{\mathbf{y}^\top \mathbf{D} \mathbf{y}}, \quad \mathbf{y} := (\mathbf{1} + \mathbf{z}) - \beta(\mathbf{1} - \mathbf{z}), \quad \beta := \frac{\sum_{z_i > 0} d_i}{\sum_{z_i < 0} d_i}.$$

generalized Rayleigh quotient  $\rho_{\mathbf{A}, \mathbf{D}}(\mathbf{y})$

*Proof.* Note that by (6.3.27)  $(\mathbf{1} - \mathbf{z})_i = 0 \Leftrightarrow i \in \mathcal{X}$ ,  $(\mathbf{1} + \mathbf{z})_i = 0 \Leftrightarrow i \notin \mathcal{X}$ . Hence, since  $(\mathbf{1} + \mathbf{z})^\top \mathbf{D}(\mathbf{1} - \mathbf{z}) = 0$ ,

$$4 \text{Ncut}(\mathcal{X}) = (\mathbf{1} + \mathbf{z})^\top \mathbf{A}(\mathbf{1} + \mathbf{z}) \left( \frac{1}{\kappa \mathbf{1}^\top \mathbf{D} \mathbf{1}} + \frac{1}{(1 - \kappa) \mathbf{1}^\top \mathbf{D} \mathbf{1}} \right) = \frac{\mathbf{y}^\top \mathbf{A} \mathbf{y}}{\beta \mathbf{1}^\top \mathbf{D} \mathbf{1}},$$

where  $\kappa := \sum_{z_i > 0} d_i / \sum_i d_i = \frac{\beta}{1 + \beta}$ . Also observe

$$\begin{aligned} \mathbf{y}^\top \mathbf{D} \mathbf{y} &= (\mathbf{1} + \mathbf{z})^\top \mathbf{D}(\mathbf{1} + \mathbf{z}) + \beta^2 (\mathbf{1} - \mathbf{z})^\top \mathbf{D}(\mathbf{1} - \mathbf{z}) = \\ &= 4 \left( \sum_{z_i > 0} d_i + \beta^2 \sum_{z_i < 0} d_i \right) = 4\beta \sum_i d_i = 4\beta \mathbf{1}^\top \mathbf{D} \mathbf{1}. \end{aligned}$$

This finishes the proof. □

- (6.3.31)  $\Rightarrow \mathbf{1} \in \text{Eig}_{\mathbf{A}}(0)$

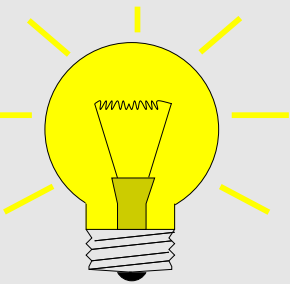
- Lemma 2.7.12:  $\mathbf{A}$  diagonally dominant  $\implies \mathbf{A}$  is positive semidefinite ( $\rightarrow$  Def. 2.7.9)

$N_{\text{cut}}(\mathcal{X}) \geq 0$  and  $0$  is the smallest eigenvalue of  $\mathbf{A}$ .

However, we are by no means interested in a minimizer  $\mathbf{y} \in \text{Span}\{\mathbf{1}\}$  (with constant entries) that does not provide a meaningful segmentation.

Idea: weed out undesirable constant minimizers by imposing **orthogonality constraint** (orthogonality w.r.t. inner product induced by  $\mathbf{D}$ , cf. Sect. 5.1)

$$\mathbf{y} \perp \mathbf{D}\mathbf{1} \iff \mathbf{1}^\top \mathbf{D}\mathbf{y} = 0. \quad (6.3.34)$$



segmentation problem (6.3.26)  $\iff$  
$$\operatorname{argmin}_{\mathbf{y} \in \{2, -2\beta\}^N, \mathbf{1}^\top \mathbf{D}\mathbf{y} = 0} \rho_{\mathbf{A}, \mathbf{D}}(\mathbf{y}). \quad (6.3.35)$$
 still NP-hard

➤ Minimizing  $N_{\text{cut}}(\mathcal{X})$  amounts to minimizing a (generalized) Rayleigh quotient ( $\rightarrow$  Def. 6.3.15) over a *discrete* set of vectors, which is still an NP-hard problem.

Idea:

Relaxation

Discrete optimization problem  $\rightarrow$  continuous optimization problem

$$(6.3.35) \rightarrow \underset{\mathbf{y} \in \mathbb{R}^N, \mathbf{y} \neq 0, \mathbf{1}^\top \mathbf{D} \mathbf{y} = 0}{\operatorname{argmin}} \rho_{\mathbf{A}, \mathbf{D}}(\mathbf{y}) . \quad (6.3.36)$$

Task: (6.3.36)  $\Leftrightarrow$ 

Find minimizer of (generalized) Rayleigh quotient under  
linear constraint

Here: linear constraint on  $\mathbf{y}$ :  $\mathbf{1}^\top \mathbf{D} \mathbf{y} = 0$

The next theorem establishes a link between argument vectors that render the Rayleigh quotient extremal and eigenspace for extremal eigenvalues.



*Theorem 6.3.37 (Rayleigh quotient theorem).*

Let  $\lambda_1 < \lambda_2 < \dots < \lambda_m$ ,  $m \leq n$ , be the sorted sequence of all (real!) eigenvalues of  $\mathbf{A} = \mathbf{A}^H \in \mathbb{C}^{n,n}$ . Then

$$\text{Eig}_{\mathbf{A}}(\lambda_1) = \underset{\mathbf{y} \in \mathbb{C}^{n,n} \setminus \{0\}}{\text{argmin}} \rho_{\mathbf{A}}(\mathbf{y}) \quad \text{and} \quad \text{Eig}_{\mathbf{A}}(\lambda_m) = \underset{\mathbf{y} \in \mathbb{C}^{n,n} \setminus \{0\}}{\text{argmax}} \rho_{\mathbf{A}}(\mathbf{y}).$$

*Remark 6.3.38 (Min-max theorem).*

Thm. 6.3.37 is an immediate consequence of the following more general and fundamentally important result.

*Theorem 6.3.39 (Courant-Fischer min-max theorem). → [23, Thm. 8.1.2]*

Let  $\lambda_1 < \lambda_2 < \dots < \lambda_m$ ,  $m \leq n$ , be the sorted sequence of the (real!) eigenvalues of  $\mathbf{A} = \mathbf{A}^H \in \mathbb{C}^{n,n}$ . Write

$$U_0 = \{0\}, \quad U_\ell := \sum_{j=1}^{\ell} \text{Eig}_{\mathbf{A}}(\lambda_j), \quad \ell = 1, \dots, m \quad \text{and} \quad U_\ell^\perp := \{\mathbf{x} \in \mathbb{C}^n : \mathbf{u}^H \mathbf{x} = 0 \forall \mathbf{u} \in U_\ell\}.$$

Then

$$\min_{\mathbf{y} \in U_\ell} \rho_{\mathbf{A}}(\mathbf{y}) = \lambda_\ell \quad 1 \leq \ell \leq m \quad \text{and} \quad \underset{\mathbf{y} \in U_\ell^\perp}{\text{argmin}} \rho_{\mathbf{A}}(\mathbf{y}) \subset \text{Eig}_{\mathbf{A}}(\lambda_\ell)$$

*Proof.* For diagonal  $\mathbf{A} \in \mathbb{R}^{n,n}$  the assertion of the theorem is obvious. Thus, Cor. 6.1.9 settles everything.  $\square$

A simple conclusion from Thm. 6.3.39: If  $\mathbf{A} = \mathbf{A}^\top \in \mathbb{R}^{n,n}$  with eigenvalues  $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ , then

$$\lambda_1 = \min_{\mathbf{z} \in \mathbb{R}^n} \rho_{\mathbf{A}}(\mathbf{z}) \quad , \quad \lambda_2 = \min_{\mathbf{z} \in \mathbb{R}^n, \mathbf{z} \perp \mathbf{v}_1} \rho_{\mathbf{A}}(\mathbf{z}) \quad , \quad (6.3.40)$$

where  $\mathbf{v}_1 \in \text{Eig}_{\mathbf{A}}(\lambda_1) \setminus \{0\}$ .  $\triangle$

Well, in Lemma 6.3.33 we encounter a generalized Rayleigh quotient  $\rho_{\mathbf{A},\mathbf{D}}(\mathbf{y})$ ! How can Thm. 6.3.37 be applied to it?

► **Transformation** idea:  $\rho_{\mathbf{A},\mathbf{D}}(\mathbf{D}^{-1/2}\mathbf{z}) = \rho_{\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}}(\mathbf{z}) \quad , \quad \mathbf{y} \in \mathbb{R}^n \quad . \quad (6.3.41)$

► Apply Thm. 6.3.39 to transformed matrix  $\tilde{\mathbf{A}} := \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ . *Elementary* manipulations

show

$$\begin{aligned}
 (6.3.36) \quad &\Leftrightarrow \operatorname{argmin}_{\mathbf{1}^\top \mathbf{D} \mathbf{y} = 0} \rho_{\mathbf{A}, \mathbf{D}}(\mathbf{y}) \stackrel{\mathbf{z} = \mathbf{D}^{1/2} \mathbf{y}}{=} \operatorname{argmin}_{\mathbf{1}^\top \mathbf{D}^{1/2} \mathbf{z} = 0} \rho_{\mathbf{A}, \mathbf{D}}(\mathbf{D}^{-1/2} \mathbf{z}) \\
 &= \operatorname{argmin}_{\mathbf{1}^\top \mathbf{D}^{1/2} \mathbf{z} = 0} \rho_{\tilde{\mathbf{A}}}(\mathbf{z}) \quad \text{with} \quad \tilde{\mathbf{A}} := \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}.
 \end{aligned} \tag{6.3.42}$$

Related: transformation of a generalized eigenvalue problem into a standard eigenvalue problem according to

$$\mathbf{A} \mathbf{x} = \lambda \mathbf{B} \mathbf{x} \quad \xrightarrow{\mathbf{z} = \mathbf{B}^{1/2} \mathbf{x}} \quad \mathbf{B}^{-1/2} \mathbf{A} \mathbf{B}^{-1/2} \mathbf{z} = \lambda \mathbf{z}. \tag{6.3.43}$$

$\mathbf{B}^{1/2} \hat{=}$  square root of s.p.d. matrix  $\mathbf{B}$   $\rightarrow$  Rem. 5.3.2.

For segmentation problem:  $\mathbf{B} = \mathbf{D}$  diagonal with positive diagonal entries, see (6.3.30)

$\rightarrow \mathbf{D}^{-1/2} = \operatorname{diag}(d_1^{-1/2}, \dots, d_N^{-1/2})$  and  $\tilde{\mathbf{A}} := \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$  can easily be computed.

► In the sequel consider minimization problem/related eigenvalue problem

$$\mathbf{z}^* = \underset{\mathbf{1}^\top \mathbf{D}^{1/2} \mathbf{z} = 0}{\operatorname{argmin}} \rho_{\tilde{\mathbf{A}}}(\mathbf{z}) \iff \tilde{\mathbf{A}} \mathbf{z} = \lambda \mathbf{z} . \quad (6.3.44)$$

Recover solution  $\mathbf{y}^*$  of (6.3.36) as  $\mathbf{y}^* = \mathbf{D}^{-1/2} \mathbf{z}^*$ .



Still, Thm. 6.3.37 cannot be applied to (6.3.44):

How to deal with *constraint*  $\mathbf{1}^\top \mathbf{D}^{1/2} \mathbf{z} = 0$  ?



Idea:

**Penalization**

Add term  $P(\mathbf{z})$  to  $\rho_{\tilde{\mathbf{A}}}(\mathbf{z})$  that becomes “sufficiently large” in case the constraint is violated.

►  $\mathbf{z}^*$  can only be a minimizer of  $\rho_{\tilde{\mathbf{A}}}(\mathbf{z}) + P(\mathbf{z})$ , if  $P(\mathbf{z}) = 0$ .

How to choose the **penalty function**  $P(\mathbf{z})$  for the segmentation problem ?

$$\left\{ \mathbf{1}^\top \mathbf{D}^{1/2} \mathbf{z} \neq 0 \Rightarrow P(\mathbf{z}) > 0 \right\} \text{ satisfied for } P(\mathbf{z}) = \mu \frac{|\mathbf{1}^\top \mathbf{D}^{1/2} \mathbf{z}|^2}{\|\mathbf{z}\|_2^2},$$

with **penalty parameter**  $\mu > 0$ .

► **penalized minimization problem**

dense rank-1 matrix

$$\begin{aligned} \mathbf{z}^* &= \operatorname{argmin}_{\mathbf{z} \in \mathbb{R}^N \setminus \{0\}} \rho_{\tilde{\mathbf{A}}}(\mathbf{z}) + P(\mathbf{z}) = \operatorname{argmin}_{\mathbf{z} \in \mathbb{R}^N \setminus \{0\}} \rho_{\tilde{\mathbf{A}}}(\mathbf{z}) + \frac{\mathbf{z}^\top (\mathbf{D}^{1/2} \mathbf{1} \mathbf{1}^\top \mathbf{D}^{1/2}) \mathbf{z}}{\mathbf{z}^\top \mathbf{z}} \\ &= \operatorname{argmin}_{\mathbf{z} \in \mathbb{R}^N \setminus \{0\}} \rho_{\hat{\mathbf{A}}}(\mathbf{z}) \quad \text{with} \quad \hat{\mathbf{A}} := \tilde{\mathbf{A}} + \mathbf{D}^{1/2} \mathbf{1} \mathbf{1}^\top \mathbf{D}^{1/2}. \end{aligned} \tag{6.3.45}$$

How to choose the penalty parameter  $\mu$  ?

In general: finding a “suitable” value for  $\mu$  may be difficult or even impossible!

Here we are lucky:

$$(6.3.31) \Rightarrow \mathbf{A}\mathbf{1} = 0 \Rightarrow \tilde{\mathbf{A}}(\mathbf{D}^{1/2}\mathbf{1}) = 0 \Leftrightarrow \mathbf{D}^{1/2}\mathbf{1} \in \text{Eig}_{\tilde{\mathbf{A}}}(0).$$

► Constraint in (6.3.44) means

Minimize over the **orthogonal complement** of an *eigenvector*. (6.3.46)

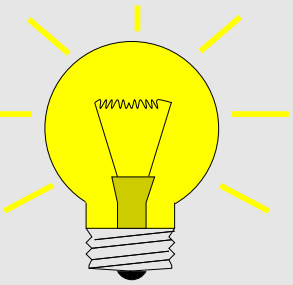
Cor. 6.1.9 ► The orthogonal complement of an eigenvector of a *symmetric* matrix is spanned by the other eigenvectors (orthonormalization of eigenvectors belonging to the same eigenvalue is assumed).

(6.3.46) ► The minimizer of (6.3.44) will be one of the other eigenvectors of  $\tilde{\mathbf{A}}$  that belongs to the smallest eigenvalue.

Note: This eigenvector  $\mathbf{z}^*$  will be orthogonal to  $\mathbf{D}^{1/2}\mathbf{1}$ , it satisfies the constraint, and, thus,  $P(\mathbf{z}^*) = 0!$

Note: eigenspaces of  $\tilde{\mathbf{A}}$  and  $\hat{\mathbf{A}}$  agree.

Note: Lemma 2.7.12  $\implies \tilde{\mathbf{A}}$  is *positive semidefinite* ( $\rightarrow$  Def. 2.7.9) with smallest eigenvalue 0.n



Idea: Choose penalization parameter  $\mu$  in (6.3.45) such that  $\mathbf{D}^{1/2}\mathbf{1}$  is guaranteed not to be an eigenvector belonging to the smallest eigenvalue of  $\hat{\mathbf{A}}$ .

Safe choice: choose  $\mu$  such that  $\mathbf{D}^{1/2}\mathbf{1}$  will belong to the largest eigenvalue of  $\hat{\mathbf{A}}$ : Thm. 6.1.4 suggests

$$\mu = \left\| \tilde{\mathbf{A}} \right\|_{\infty} \stackrel{(2.5.7)}{=} 2. \quad (6.3.47)$$



$$\mathbf{z}^* = \underset{\mathbf{1}^T \mathbf{D}^{1/2} \mathbf{z} = 0}{\operatorname{argmin}} \rho_{\tilde{\mathbf{A}}}(\mathbf{z}) = \underset{\mathbf{z} \neq 0}{\operatorname{argmin}} \rho_{\hat{\mathbf{A}}}(\mathbf{z}). \quad (6.3.48)$$

By Thm. 6.3.37:

$$\begin{aligned} \mathbf{z}^* &= \text{eigenvector belonging to minimal eigenvalue of } \hat{\mathbf{A}}, \\ &\quad \Updownarrow \\ \mathbf{z}^* &= \text{eigenvector } \perp \mathbf{D}^{1/2}\mathbf{1} \text{ belonging to minimal eigenvalue of } \tilde{\mathbf{A}}, \\ &\quad \Updownarrow \\ \mathbf{D}^{-1/2}\mathbf{z}^* &= \text{minimizer for (6.3.36)}. \end{aligned}$$

- ❶ Given similarity function  $\sigma$  compute (sparse!) matrices  $\mathbf{W}, \mathbf{D}, \mathbf{A} \in \mathbb{R}^{N,N}$ , see (6.3.24), (6.3.30).
- ❷ Compute  $\mathbf{y}^*$ ,  $\|\mathbf{y}^*\|_2 = 1$ , as eigenvector belonging to the *smallest eigenvalue* of  $\hat{\mathbf{A}} := \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2} + 2(\mathbf{D}^{1/2} \mathbf{1})(\mathbf{D}^{1/2} \mathbf{1})^\top$ .
- ❸ Set  $\mathbf{x}^* = \mathbf{D}^{-1/2} \mathbf{y}^*$  and define the image segment as pixel set

$$\mathcal{X} := \{i \in \{1, \dots, N\} : x_i^* > \frac{1}{N} \sum_{i=1}^N x_i^*\}. \quad (6.3.50)$$

↑  
mean value of entries of  $\mathbf{x}^*$

Code 6.3.51: 1st stage of segmentation of grayscale image

```

1 % Read image and build matrices, see Code 6.3.31 and (6.3.30)
2 P = imread('image.pbm'); [m,n] = size(P); [A,D] = imgsegmat(P);
3 % Build scaling matrices
4 N = size(A,1); dv = sqrt(spdiags(A,0));
5 Dm = spdiags(1./dv,[0],N,N); % D-1/2
6 Dp = spdiags(dv,[0],N,N); % D-1/2
7 % Build (densely populated!) matrix  $\hat{\mathbf{A}}$ 
8 c = Dp*ones(N,1); Ah = Dm*A*Dm + 2*c*c';

```

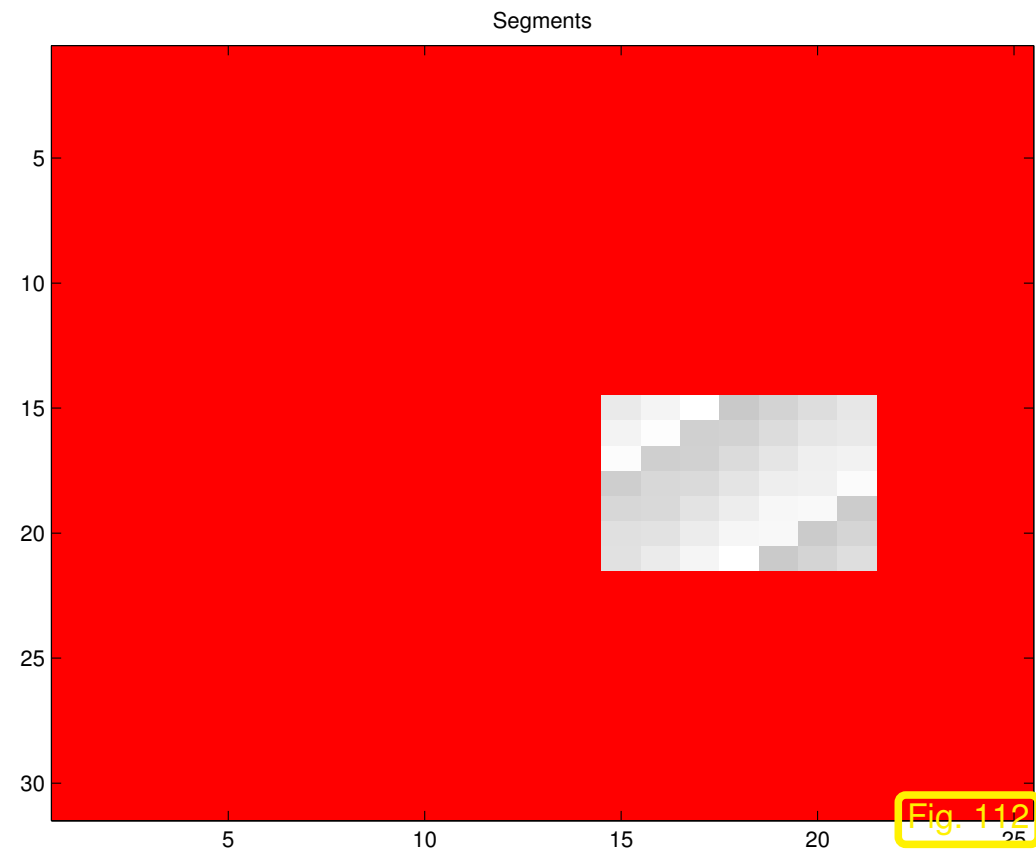
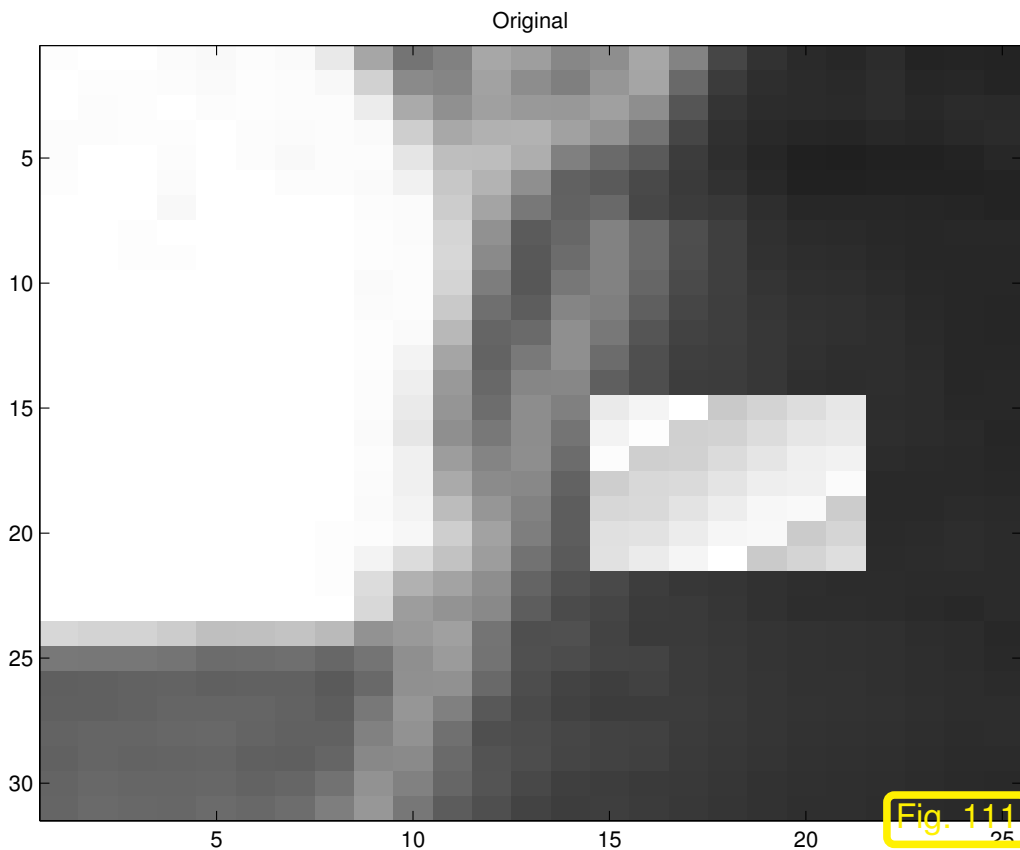


```

9 % Compute and sort eigenvalues; grossly inefficient !
10 [W,E] = eig(full(Ah)); [ev,idx] = sort(diag(E)); W(:,idx) = W;
11 % Obtain eigenvector x* belonging to 2nd smallest generalized
12 % eigenvalue of A and D
13 x = W(:,1); x = Dm*v;
14 % Extract segmented image
15 xs = reshape(x,m,n); Xidx = find(xs > (sum(sum(xs))/(n*m)));

```

1st-stage of segmentation of  $31 \times 25$  grayscale pixel image (root.pbm, red pixels  $\hat{=}$   $\mathcal{X}$ ,  $\sigma(x,y) = \exp(-(x-y/10)^2)$ )



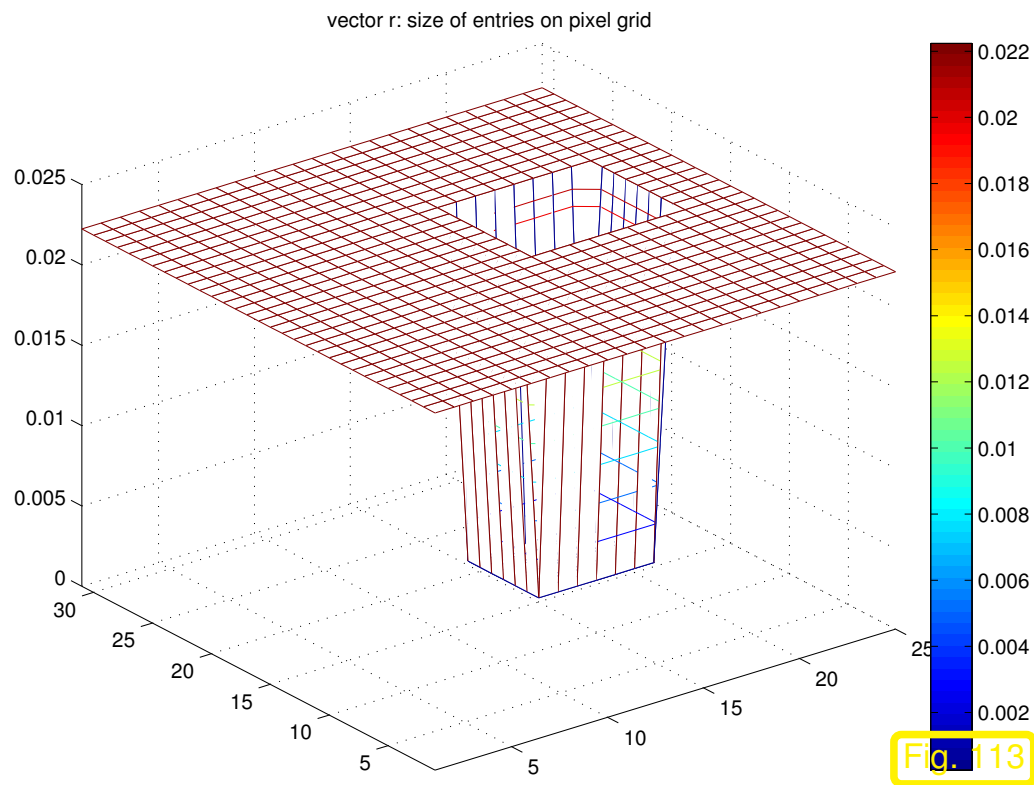


Image from Fig. 111:

◁ eigenvector  $x^*$  plotted on pixel grid

To identify more segments, the same algorithm is *recursively applied* to segment parts of the image already determined.

Practical segmentation algorithms rely on many more steps of which the above algorithm is only one, preceded by substantial preprocessing. Moreover, they dispense with the strictly local perspective adopted above and take into account more distant connections between image parts, often in a randomized fashion [58].

The image segmentation problem falls into the wider class of **graph partitioning problems**. Methods based on (a few of) the eigenvector of the connectivity matrix belonging to the smallest eigenvalues are known as **spectral partitioning methods**. The eigenvector belonging to the smallest non-zero eigenvalue that we computed above is usually called the **Fiedler vector** of the graph, see [59, 1].



The solution of the image segmentation problem by means of `eig` in Code 6.3.50 amounts a tremendous waste of computational resources: we compute *all* eigenvalues/eigenvectors of dense matrices, though only a single eigenvector associated with the smallest eigenvalue is of interest.

This motivates the quest to find *efficient* numerical methods for the following task.

Task: given  $\mathbf{A} \in \mathbb{K}^{n,n}$ , find **smallest** (in modulus) eigenvalue of regular  $\mathbf{A} \in \mathbb{K}^{n,n}$  and (an) associated eigenvector.

If  $\mathbf{A} \in \mathbb{K}^{n,n}$  regular:

$$\text{Smallest (in modulus) EV of } \mathbf{A} = \left( \text{Largest (in modulus) EV of } \mathbf{A}^{-1} \right)^{-1}$$

Direct power method ( $\rightarrow$  Sect. 6.3.1) for  $\mathbf{A}^{-1}$  = **inverse iteration**

Code 6.3.52: inverse iteration for computing  $\lambda_{\min}(\mathbf{A})$  and associated eigenvector

```

1 function [lmin,y] = invit(A,tol)
2 [L,U] = lu(A); % single intial LU-factorization, see Rem. 2.2.13
3 n = size(A,1); x = rand(n,1); x = x/norm(x); % random initial guess
4 y = U\(L\x); lmin = 1/norm(y); y = y*lmin; lold = 0;
5 while (abs(lmin-lold) > tol*lmin) % termination, if small relative change
6     lold = lmin; x = y;
7     y = U\(L\x); % core iteration:  $\mathbf{y} = \mathbf{A}^{-1}\mathbf{x}$ ,
8     lmin = 1/norm(y); % new approximation of  $\lambda_{\min}(\mathbf{A})$ 
9     y = y*lmin; % normalization  $\mathbf{y} := \frac{\mathbf{y}}{\|\mathbf{y}\|_2}$ 
10 end

```

Note: **reuse** of LU-factorization, see Rem. 2.2.13

*Remark 6.3.53* (Shifted inverse iteration).

More general task:

For  $\alpha \in \mathbb{C}$  find  $\lambda \in \sigma(\mathbf{A})$  such that  $|\alpha - \lambda| = \min\{|\alpha - \mu|, \mu \in \sigma(\mathbf{A})\}$

► **Shifted inverse iteration:** [13, Alg .7.24]

$$\mathbf{z}^{(0)} \text{ arbitrary, } \mathbf{w} = (\mathbf{A} - \alpha\mathbf{I})^{-1}\mathbf{z}^{(k-1)}, \quad \mathbf{z}^{(k)} := \frac{\mathbf{w}}{\|\mathbf{w}\|_2}, \quad k = 1, 2, \dots, \quad (6.3.54)$$

where:  $(\mathbf{A} - \alpha\mathbf{I})^{-1}\mathbf{z}^{(k-1)} \hat{=}$  solve  $(\mathbf{A} - \alpha\mathbf{I})\mathbf{w} = \mathbf{z}^{(k-1)}$  based on Gaussian elimination ( $\leftrightarrow$  a **single** LU-factorization of  $\mathbf{A} - \alpha\mathbf{I}$  as in Code 6.3.51).



*Remark 6.3.55* ((Nearly) singular LSE in shifted inverse iteration).

What if “by accident”  $\alpha \in \sigma(\mathbf{A})$  ( $\Leftrightarrow \mathbf{A} - \alpha\mathbf{I}$  singular) ?

Stability of Gaussian elimination/LU-factorization ( $\rightarrow$  Sect. 2.5.3) will ensure that “ $\mathbf{w}$  from (6.3.54) points in the right direction”

In other words, roundoff errors may badly affect the length of the solution  $\mathbf{w}$ , but not its direction.

Practice [24]: If, in the course of Gaussian elimination/LU-factorization a zero pivot element is really encountered, then we just *replace it with* `eps`, in order to avoid `inf` values!

Thm. 6.3.19  $\triangleright$  Convergence of shifted inverse iteration for  $\mathbf{A}^H = \mathbf{A}$ :

Asymptotic **linear convergence**, Rayleigh quotient  $\rightarrow \lambda_j$  with rate

$$\frac{|\lambda_j - \alpha|}{\min\{|\lambda_i - \alpha|, i \neq j\}} \quad \text{with} \quad \lambda_j \in \sigma(\mathbf{A}), \quad |\alpha - \lambda_j| \leq |\alpha - \lambda| \quad \forall \lambda \in \sigma(\mathbf{A}).$$

$\blacktriangleright$  Extremely fast for  $\alpha \approx \lambda_j$  !

Idea:

A posteriori adaptation of shift

Use  $\alpha := \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})$  in  $k$ -th step of inverse iteration.

Algorithm 6.3.56 (Rayleigh quotient iteration). → [35, Alg. 25.2]

Code 6.3.57: Rayleigh quotient iteration (for *normal*  $\mathbf{A} \in \mathbb{R}^{n,n}$ )

```

1 function [z,lmin] = rqui(A,tol,maxit)
2 alpha = 0; n = size(A,1);
3 z = rand(size(A,1),1); z = z/norm(z); %  $\mathbf{z}^{(0)}$ 
4 for i=1:maxit
5     z = (A-alpha*speye(n))\z; %  $\mathbf{z}^{(k+1)} = (\mathbf{A} - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})\mathbf{I})^{-1}\mathbf{x}^{(k)}$ 
6     z = z/norm(z); lmin=dot(A*z,z); % Computation of  $\rho_{\mathbf{A}}(\mathbf{z}^{(k+1)})$ 
7     if (abs(alpha-lmin) < tol*lmin) % Desired relative accuracy reached ?
8         break; end;
9     alpha = lmin;
10 end

```

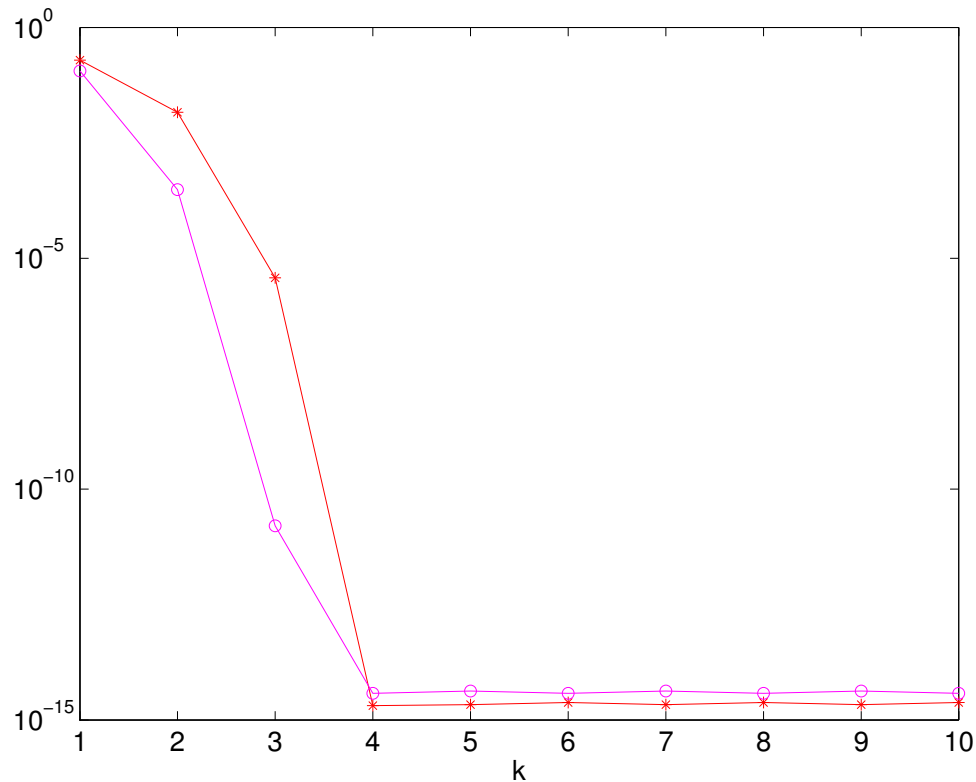
Line 5: note use of `speye` to preserve sparse matrix data format!

- Drawback compared with Code 6.3.51: reuse of LU-factorization no longer possible.
- Even if LSE nearly singular, stability of Gaussian elimination guarantees correct direction of  $\mathbf{z}$ , see discussion in Rem. 6.3.55.

*Example 6.3.58* (Rayleigh quotient iteration).

Monitored: iterates of Rayleigh quotient iteration (6.3.56) for s.p.d.  $\mathbf{A} \in \mathbb{R}^{n,n}$





```
d = (1:10)';
n = length(d);
Z = diag(sqrt(1:n), 0) + ones(n, n);
[Q, R] = qr(Z);
A = Q * diag(d, 0) * Q';
```

- :  $|\lambda_{\min} - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})|$
- \* :  $\|\mathbf{z}^{(k)} - \mathbf{x}_j\|, \lambda_{\min} = \lambda_j, \mathbf{x}_j \in \text{Eig}_{\mathbf{A}}(\lambda_j),$
- :  $\|\mathbf{x}_j\|_2 = 1$

$k$	$ \lambda_{\min} - \rho_{\mathbf{A}}(\mathbf{z}^{(k)}) $	$\ \mathbf{z}^{(k)} - \mathbf{x}_j\ $
1	0.09381702342056	0.20748822490698
2	0.00029035607981	0.01530829569530
3	0.00000000001783	0.00000411928759
4	0.00000000000000	0.00000000000000
5	0.00000000000000	0.00000000000000

*Theorem 6.3.59. → [35, Thm. 25.4]*  
 If  $\mathbf{A} = \mathbf{A}^H$ , then  $\rho_{\mathbf{A}}(\mathbf{z}^{(k)})$  converges locally of order 3 (→ Def. 4.1.14) to the smallest eigenvalue (in modulus), when  $\mathbf{z}^{(k)}$  are generated by the Rayleigh quotient iteration (6.3.56).



## 6.3.3 Preconditioned inverse iteration (PINVIT)

Task: given  $\mathbf{A} \in \mathbb{K}^{n,n}$ , find **smallest** (in modulus) eigenvalue of regular  $\mathbf{A} \in \mathbb{K}^{n,n}$  and (an) associated eigenvector.

► Options: inverse iteration ( $\rightarrow$  Code 6.3.51) and Rayleigh quotient iteration (6.3.56).



What if direct solution of  $\mathbf{Ax} = \mathbf{b}$  not feasible ?

This can happen, in case

- for large sparse  $\mathbf{A}$  the amount of fill-in exhausts memory, despite sparse elimination techniques ( $\rightarrow$  Sect. 2.6.3),
- $\mathbf{A}$  is available only through a routine `evalA(x)` providing  $\mathbf{A} \times \text{vector}$ .

We expect that an approximate solution of the linear systems of equations encountered during inverse iteration should be sufficient, because we are dealing with approximate eigenvectors anyway.

Thus, iterative solvers for solving  $\mathbf{A}\mathbf{w} = \mathbf{z}^{(k-1)}$  may be considered, see Sect. 5. However, the required accuracy is not clear a priori. Here we examine an approach that completely dispenses with an iterative solver and uses a *preconditioner* ( $\rightarrow$  Def. 5.3.3) instead.

Idea: (for inverse iteration without shift,  $\mathbf{A} = \mathbf{A}^H$  s.p.d.)

Instead of solving  $\mathbf{A}\mathbf{w} = \mathbf{z}^{(k-1)}$  compute  $\mathbf{w} = \mathbf{B}^{-1}\mathbf{z}^{(k-1)}$  with  
“inexpensive” s.p.d. **approximate inverse**  $\mathbf{B}^{-1} \approx \mathbf{A}^{-1}$

$\triangleright$   $\mathbf{B} \hat{=} \mathbf{Preconditioner}$  for  $\mathbf{A}$ , see Def. 5.3.3

Possible to replace  $\mathbf{A}^{-1}$  with  $\mathbf{B}^{-1}$  in inverse iteration ?

**NO**, because we are not interested in smallest eigenvalue of  $\mathbf{B}$  !


Replacement  $\mathbf{A}^{-1} \rightarrow \mathbf{B}^{-1}$  possible only when applied to **residual quantity**

residual quantity = quantity that  $\rightarrow 0$  in the case of convergence to exact solution

Natural residual quantity for eigenvalue problem  $\mathbf{Ax} = \lambda\mathbf{x}$ :

$$\mathbf{r} := \mathbf{Az} - \rho_{\mathbf{A}}(\mathbf{z})\mathbf{z} \quad , \quad \rho_{\mathbf{A}}(\mathbf{z}) = \text{Rayleigh quotient} \rightarrow \text{Def. 6.3.15} .$$

Note: only *direction* of  $\mathbf{A}^{-1}\mathbf{z}$  matters in inverse iteration (6.3.54)

$$(\mathbf{A}^{-1}\mathbf{z}) \parallel (\mathbf{z} - \mathbf{A}^{-1}(\mathbf{Az} - \rho_{\mathbf{A}}(\mathbf{z})\mathbf{z})) \Rightarrow \text{defines same next iterate!}$$




[Preconditioned inverse iteration (PINVIT) for s.p.d.  $\mathbf{A}$ ]

$$\mathbf{z}^{(0)} \text{ arbitrary,} \quad \mathbf{w} = \mathbf{z}^{(k-1)} - \mathbf{B}^{-1}(\mathbf{Az}^{(k-1)} - \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{z}^{(k-1)}) , \quad k = 1, 2, \dots . \quad (6.3.60)$$

$$\mathbf{z}^{(k)} = \frac{\mathbf{w}}{\|\mathbf{w}\|_2} ,$$

## Code 6.3.61: preconditioned inverse iteration (6.3.60)

```

1 function [lmin,z,res] =
   pinvit(evalA,n,invB,tol,maxit)
2 % invB  $\hat{=}$  handle to function implementing preconditioner  $B^{-1}$ 
3 z = (1:n)'; z = z/norm(z); % initial guess
4 res = []; rho = 0;
5 for i=1:maxit
6   v = evalA(z); rhon = dot(v,z); % Rayleigh quotient
7   r = v - rhon*z; % residual
8   z = z - invB(r); % iteration according to (6.3.60)
9   z = z/norm(z); % normalization
0   res = [res; rhon]; % tracking iteration
1   if (abs(rho-rhon) < tol*abs(rhon)), break;
2   else rho = rhon; end
3 end
4 lmin = dot(evalA(z),z); res = [res; lmin],

```

Computational  
effort:

1 matrix  $\times$  vector

1 evaluation of  
preconditioner

A few

AXPY-operations

Example 6.3.62 (Convergence of PINVIT).

S.p.d. matrix  $A \in \mathbb{R}^{n,n}$ , tridiagonal preconditioner, see Ex. 5.3.11

```

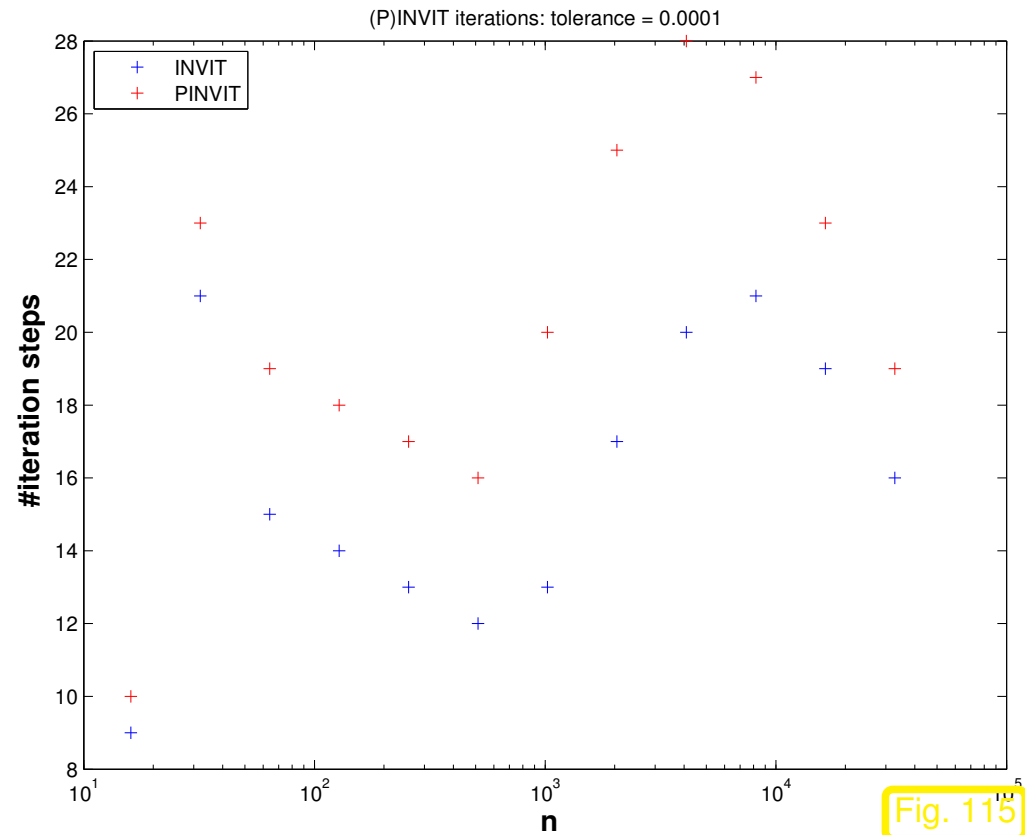
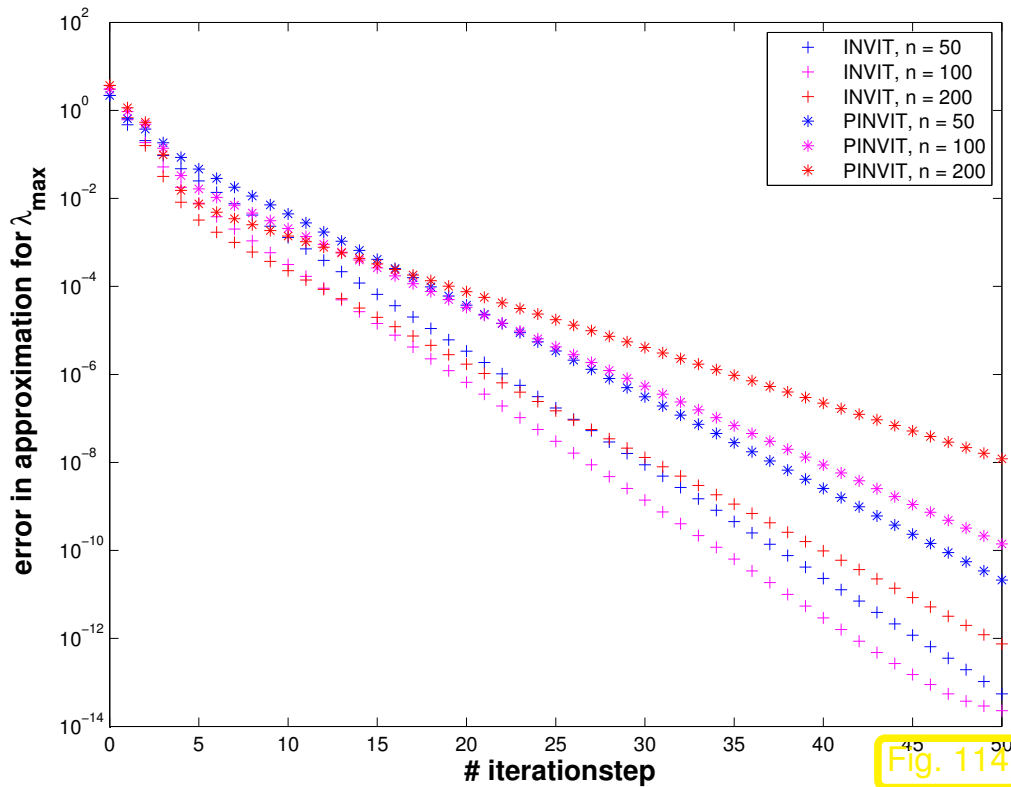
1 A = spdiags(repmat([1/n,-1,2*(1+1/n),-1,1/n],n,1),
   [-n/2,-1,0,1,n/2],n,n);
2 evalA = @(x) A*x;
3 % inverse iteration

```

```

4 invB = @(x) A\x;
5 % tridiagonal preconditioning
6 B = spdiags(spdiags(A, [-1, 0, 1]), [-1, 0, 1], n, n); invB = @(x) B\x;
    
```

Monitored: error decay during iteration of Code 6.3.60:  $|\rho_{\mathbf{A}}(\mathbf{z}^{(k)}) - \lambda_{\min}(\mathbf{A})|$



Observation: linear convergence of eigenvectors also for PINVIT.



- Theory [47, 46]:
- linear convergence of (6.3.60)
  - fast convergence, if spectral condition number  $\kappa(\mathbf{B}^{-1}\mathbf{A})$  small

The theory of PINVIT [47, 46] is based on the identity

$$\mathbf{w} = \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{A}^{-1}\mathbf{z}^{(k-1)} + (\mathbf{I} - \mathbf{B}^{-1}\mathbf{A})(\mathbf{z}^{(k-1)} - \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{A}^{-1}\mathbf{z}^{(k-1)}) . \quad (6.3.63)$$

For small residual  $\mathbf{A}\mathbf{z}^{(k-1)} - \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{z}^{(k-1)}$  PINVIT almost agrees with the regular inverse iteration.

*Remark 6.3.64* (Excited resonances).

Consider the non-autonomous ODE (excited harmonic oscillator)

$$\ddot{y} + \lambda^2 y = \cos(\omega t) , \quad (6.3.65)$$

with general solution

$$y(t) = \begin{cases} \frac{1}{\lambda^2 - \omega^2} \cos(\omega t) + A \cos(\lambda t) + B \sin(\lambda t) & , \text{ if } \lambda \neq \omega , \\ \frac{t}{2\omega} \sin(\omega t) + A \cos(\lambda t) + B \sin(\lambda t) & , \text{ if } \lambda = \omega . \end{cases} \quad (6.3.66)$$

► growing solutions possible in **resonance case**  $\lambda = \omega$  !

Now consider harmonically excited vibration modelled by ODE

$$\ddot{\mathbf{y}} + \mathbf{A}\mathbf{y} = \mathbf{b} \cos(\omega t) , \quad (6.3.67)$$



with *symmetric, positive (semi)definite* matrix  $\mathbf{A} \in \mathbb{R}^{n,n}$ ,  $\mathbf{b} \in \mathbb{R}^n$ . By Cor. 6.1.9 there is an *orthogonal* matrix  $\mathbf{Q} \in \mathbb{R}^{n,n}$  such that

$$\mathbf{Q}^\top \mathbf{A} \mathbf{Q} = \mathbf{D} := \text{diag}(\lambda_1, \dots, \lambda_n) .$$

where the  $0 \leq \lambda_1 < \lambda_2 < \dots < \lambda_n$  are the eigenvalues of  $\mathbf{A}$ .

► Transform ODE as in Ex. 6.0.7: with  $\mathbf{z} = \mathbf{Q}^\top \mathbf{y}$

$$(6.3.67) \quad \text{►} \quad \ddot{\mathbf{z}} + \mathbf{D} \mathbf{z} = \mathbf{Q}^\top \mathbf{b} \cos(\omega t) .$$

We have obtained decoupled linear 2nd-order scalar ODEs of the type (6.3.65).

► (6.3.67) can have growing (with time) solutions, if  $\omega = \sqrt{\lambda_i}$  for some  $i = 1, \dots, n$

If  $\omega = \sqrt{\lambda_j}$  for one  $j \in \{1, \dots, n\}$ , then the solution for the initial value problem for (6.3.67) with  $\mathbf{y}(0) = \dot{\mathbf{y}}(0) = 0$  ( $\leftrightarrow \mathbf{z}(0) = \dot{\mathbf{z}}(0) = 0$ ) is

$$\mathbf{z}(t) \sim \frac{t}{2\omega} \sin(\omega t) \mathbf{e}_j + \text{bounded oscillations}$$

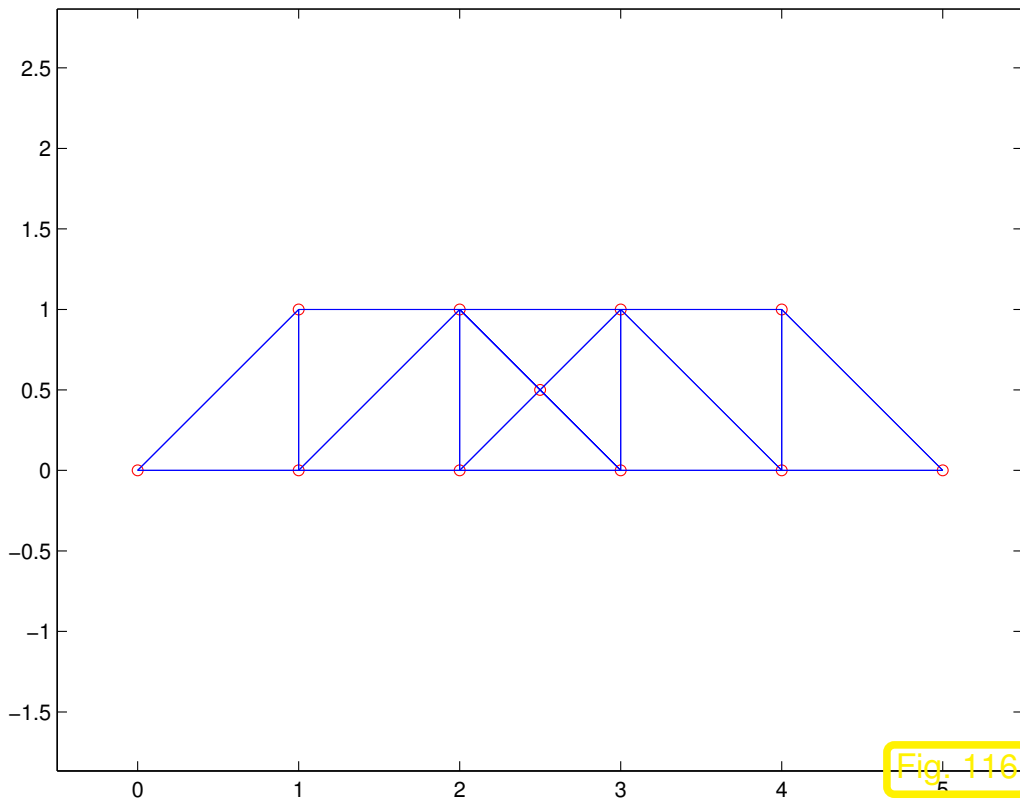
$\Updownarrow$

$$\mathbf{y}(t) \sim \frac{t}{2\omega} \sin(\omega t) (\mathbf{Q})_{:,j} + \text{bounded oscillations} .$$

$j$ -th eigenvector of  $\mathbf{A}$

Example 6.3.68 (Vibrations of a truss structure).

cf. [35, Sect. 3], MATLAB's `truss` demo



A **truss** is a structure composed of (massless) rods and point masses; we consider in-plane (2D) trusses.

Encoding: positions of masses + (sparse) connectivity matrix

R. Hiptmair  
rev 38286,  
November  
10, 2011

Code 6.3.69: Data for “bridge truss”

```
1 % Data for truss structure "bridge"
2 pos = [ 0 0; 1 0; 2 0; 3 0; 4 0; 5 0; 1 1; 2 1; 3 1; 4 1; 2.5 0.5];
```

```

3 con = [1 2;2 3;3 4;4 5;5 6;1 7;2 7;3 8;2 8;4 8;5 9;5 10;6 10;7
      8;8 9;9 10;8 11 ...
4       ; 9 11;3 11;4 11;4 9];
5 n = size(pos,1);
6 top = sparse(con(:,1),con(:,2),ones(size(con,1),1),n,n);
7 top = sign(top+top');

```

- Assumptions:
- Truss in static equilibrium (perfect balance of forces at each point mass).
  - Rods are *perfectly elastic* (i.e., frictionless).

► **Hook's law** holds for force in the direction of a rod:

$$F = \alpha \frac{\Delta l}{l}, \quad (6.3.70)$$

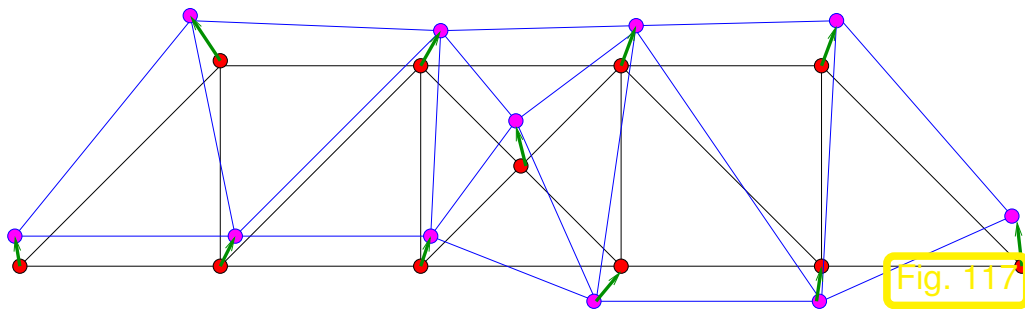
where •  $l$  is the equilibrium length of the rod,

- $\Delta l$  is the elongation of the rod effected by the force  $F$  in the direction of the rod
- $\alpha$  is a material coefficient (Young's modulus).

$n$  point masses are numbered  $1, \dots, n$ :  $\mathbf{p}^i \in \mathbb{R}^2 \hat{=}$  position of  $i$ -th mass

We consider a swaying truss: description by time-dependent displacements  $\mathbf{u}^i(t) \in \mathbb{R}^2$  of point masses:

$$\text{position of } i\text{-th mass at time } t = \mathbf{p}^i + \mathbf{u}^i(t); .$$



◁ deformed truss:

- $\hat{=}$  point masses at positions  $\mathbf{p}^i$
- $\rightarrow$   $\hat{=}$  displacement vectors  $\mathbf{u}^i$
- $\hat{=}$  shifted masses at  $\mathbf{p}^i + \mathbf{u}^i$

Equilibrium length and (time-dependent) elongation of rod connecting point masses  $i$  and  $j$ ,  $i \neq j$ :

$$l_{ij} := \left\| \Delta \mathbf{p}^{ji} \right\|_2, \quad \Delta \mathbf{p}^{ji} := \mathbf{p}^j - \mathbf{p}^i, \quad (6.3.71)$$

$$\Delta l_{ij}(t) := \left\| \Delta \mathbf{p}^{ji} + \Delta \mathbf{u}^{ji}(t) \right\|_2 - l_{ij}, \quad \Delta \mathbf{u}^{ji}(t) := \mathbf{u}^j(t) - \mathbf{u}^i(t). \quad (6.3.72)$$

► Extra (reaction) force on masses  $i$  and  $j$ :

$$F_{ij}(t) = -\alpha_{ij} \frac{\Delta l_{ij}}{l_{ij}} \cdot \frac{\Delta \mathbf{p}^{ji} + \Delta \mathbf{u}^{ji}(t)}{\left\| \Delta \mathbf{p}^{ji} + \Delta \mathbf{u}^{ji}(t) \right\|_2}. \quad (6.3.73)$$

Assumption:

Small displacements

► Possibility of **linearization** by neglecting terms of order  $\left\| \mathbf{u}^i \right\|_2^2$

$$(6.3.71) \quad \blacktriangleright \quad (6.3.72) \quad F_{ij}(t) = \alpha_{ij} \left( \frac{1}{\left\| \Delta \mathbf{p}^{ji} + \Delta \mathbf{u}^{ji}(t) \right\|_2} - \frac{1}{\left\| \Delta \mathbf{p}^{ji} \right\|_2} \right) \cdot (\Delta \mathbf{p}^{ji} + \Delta \mathbf{u}^{ji}(t)). \quad (6.3.74)$$

**Lemma 6.3.75** (Taylor expansion of inverse distance function).

For  $\mathbf{x} \in \mathbb{R}^d \setminus \{0\}$ ,  $\mathbf{y} \in \mathbb{R}^d$ ,  $\|\mathbf{y}\|_2 < \|\mathbf{x}\|_2$  holds for  $\mathbf{y} \rightarrow 0$

$$\frac{1}{\|\mathbf{x} + \mathbf{y}\|_2} = \frac{1}{\|\mathbf{x}\|_2} - \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|_2^3} + O(\|\mathbf{y}\|_2^2).$$

*Proof.* Simple Taylor expansion up to linear term for  $f(\mathbf{x}) = (x_1^2 + \dots + x_d^2)^{-1/2}$  and  $f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + \text{grad } f(\mathbf{x}) \cdot \mathbf{y} + O(\|\mathbf{y}\|_2^2)$ . □

Linearization of force: apply Lemma 6.3.75 to (6.3.74) and drop terms  $O(\|\Delta \mathbf{u}^{ji}\|_2^2)$ :

$$\begin{aligned} \blacktriangleright \quad F_{ij}(t) &\approx -\alpha_{ij} \frac{\Delta \mathbf{p}^{ji} \cdot \Delta \mathbf{u}^{ji}(t)}{l_{ij}^3} \cdot (\Delta \mathbf{p}^{ji} + \Delta \mathbf{u}^{ji}(t)) \\ &\approx -\alpha_{ij} \frac{\Delta \mathbf{p}^{ji} \cdot \Delta \mathbf{u}^{ji}(t)}{l_{ij}^3} \cdot \Delta \mathbf{p}^{ji}. \end{aligned} \tag{6.3.76}$$

Newton's second law of motion: ( $F_i \hat{=}$  total force acting on  $i$ -th mass)

$$m_i \frac{d^2}{dt^2} \mathbf{u}^i(t) = F_i = \sum_{\substack{j=1 \\ j \neq i}}^n -F_{ij}(t), \tag{6.3.77}$$

$m_i \hat{=}$  mass of point mass  $i$ .

$$\blacktriangleright \quad m_i \frac{d^2}{dt^2} \mathbf{u}^i(t) = \sum_{\substack{j=1 \\ j \neq i}}^n \alpha_{ij} \frac{1}{l_{ij}^3} \left( \Delta \mathbf{p}^{ji} (\Delta \mathbf{p}^{ji})^\top \right) (\mathbf{u}^j(t) - \mathbf{u}^i(t)) . \quad (6.3.78)$$

Compact notation: collect all displacements into one vector  $\mathbf{u}(t) = \left( \mathbf{u}^i(t) \right)_{i=1}^n \in \mathbb{R}^{2n}$

$$(6.3.78) \quad \blacktriangleright \quad \boxed{\mathbf{M} \frac{d\mathbf{u}}{dt^2}(t) + \mathbf{A} \mathbf{u}(t) = \mathbf{f}(t)} . \quad (6.3.79)$$

with **mass matrix**  $\mathbf{M} = \text{diag}(m_1, m_1, \dots, m_n, m_n)$

and **stiffness matrix**  $\mathbf{A} \in \mathbb{R}^{2n, 2n}$  with  $2 \times 2$ -blocks

$$(\mathbf{A})_{2i-1:2i, 2i-1:2i} = \sum_{\substack{j=1 \\ j \neq i}}^n \alpha_{ij} \frac{1}{l_{ij}^3} \left( \Delta \mathbf{p}^{ji} (\Delta \mathbf{p}^{ji})^\top \right) , \quad i = 1, \dots, n , \quad (6.3.80)$$

$$(\mathbf{A})_{2i-1:2i, 2j-1:2j} = -\alpha_{ij} \frac{1}{l_{ij}^3} \left( \Delta \mathbf{p}^{ji} (\Delta \mathbf{p}^{ji})^\top \right) , \quad i \neq j .$$

and external forces  $\mathbf{f}(t) = \left( \mathbf{f}^i(t) \right)_{i=1}^n$ .

Note: stiffness matrix  $\mathbf{A}$  is *symmetric, positive semidefinite* ( $\rightarrow$  Def. 2.7.9).

Rem. 6.3.64: if periodic external forces  $\mathbf{f}(t) = \cos(\omega t)\mathbf{f}$ ,  $\mathbf{f} \in \mathbb{R}^{2n}$ , (wind, earthquake) act on the truss they can excite vibrations of (linearly in time) growing amplitude, if  $\omega$  coincides with  $\sqrt{\lambda_j}$  for an eigenvalue  $\lambda_j$  of  $\mathbf{A}$ .

Excited vibrations can lead to the collapse of a truss structure, cf. the notorious Tacoma-Narrows bridge disaster.

► It is essential to know whether eigenvalues of a truss structure fall into a range that can be excited by external forces.

These will typically<sup>(\*)</sup> be the **low modes**  $\leftrightarrow$  a few of the smallest eigenvalues.

(<sup>(\*)</sup> Reason: fast oscillations will quickly be damped due to friction, which was neglected in our model.)



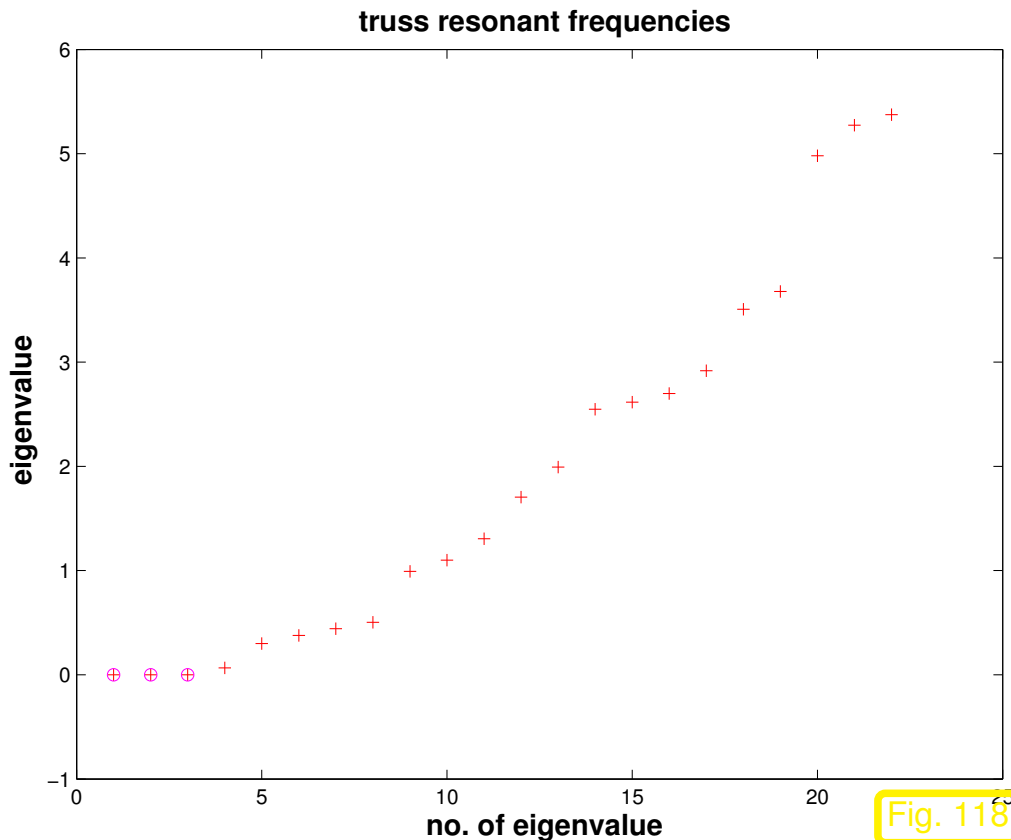
## Code 6.3.81: Computing resonant frequencies and modes of elastic truss

```

1 function [lambda,V] = trussvib(pos,top)
2 % Computes vibration modes of a truss structure, see Ex. 6.3.68. Mass point
3 % positions passed in the  $n \times 2$ -matrix poss and the connectivity encoded in
4 % the sparse symmetric matrix top. In addition top(i,j) also stores the
5 % Young's moduli  $\alpha_{ij}$ .
6 % The  $2n$  resonant frequencies are returned in the vector lambda, the
7 % eigenmodes in the column of V, where entries at odd positions contain the
8 %  $x_1$ -coordinates, entries at even positions the  $x_2$ -coordinates
9 n = size (pos,1); % no. of point masses
10 % Assembly of stiffness matrix according to (6.3.80)
11 A = zeros (2*n,2*n);
12 [Iidx,Jidx] = find (top); idx = [Iidx,Jidx]; % Find connected masses
13 for ij = idx'
14     i = ij(1); j = ij(2);
15     dp = [pos(j,1);pos(j,2)] - [pos(i,1);pos(i,2)]; %  $\Delta \mathbf{p}^{ji}$ 
16     lij = norm (dp); %  $l_{ij}$ 
17     A(2*i-1:2*i,2*j-1:2*j) = -(dp*dp') / (lij^3);
18 end
19 % Set Young's moduli  $\alpha_{ij}$  (stored in top matrix)
20 A = A.*full (kron (top, [1 1;1 1]));
21 % Set  $2 \times 2$  diagonal blocks

```

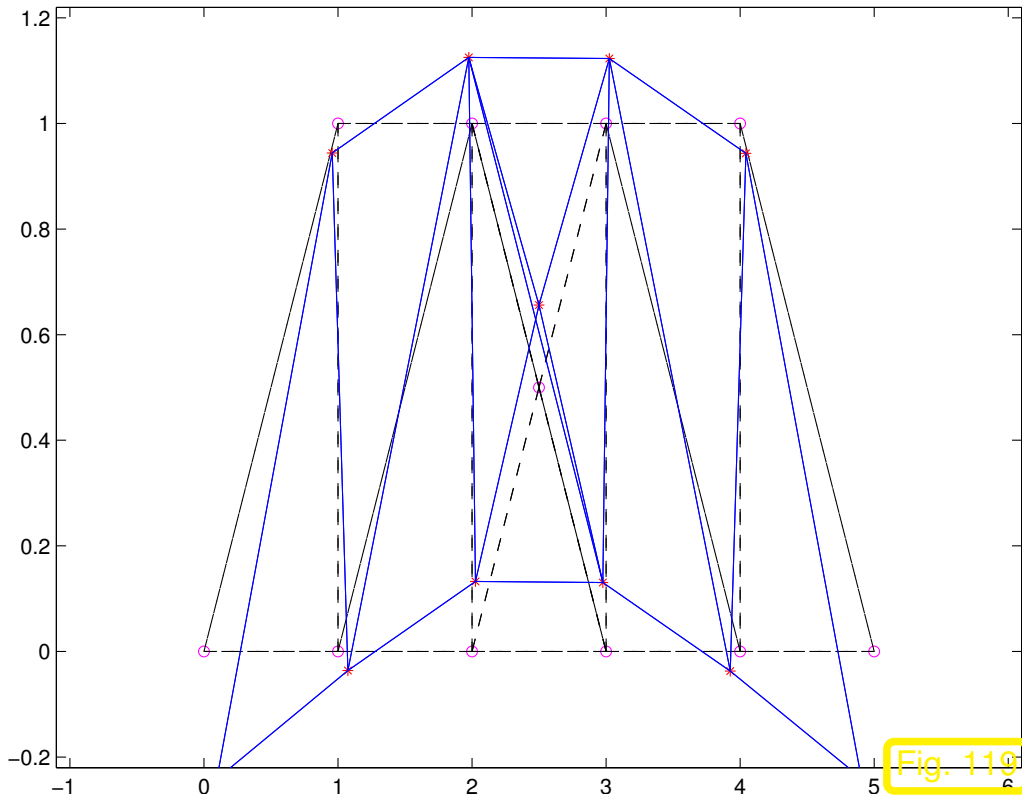
```
22 for i=1:n
23     A(2*i-1:2*i,2*i-1) = -sum(A(2*i-1:2*i,1:2:end)')';
24     A(2*i-1:2*i,2*i) = -sum(A(2*i-1:2*i,2:2:end)')';
25 end
26 % Compute eigenvalues and eigenmodes
27 [V,D] = eig(A); lambda = diag(D);
```



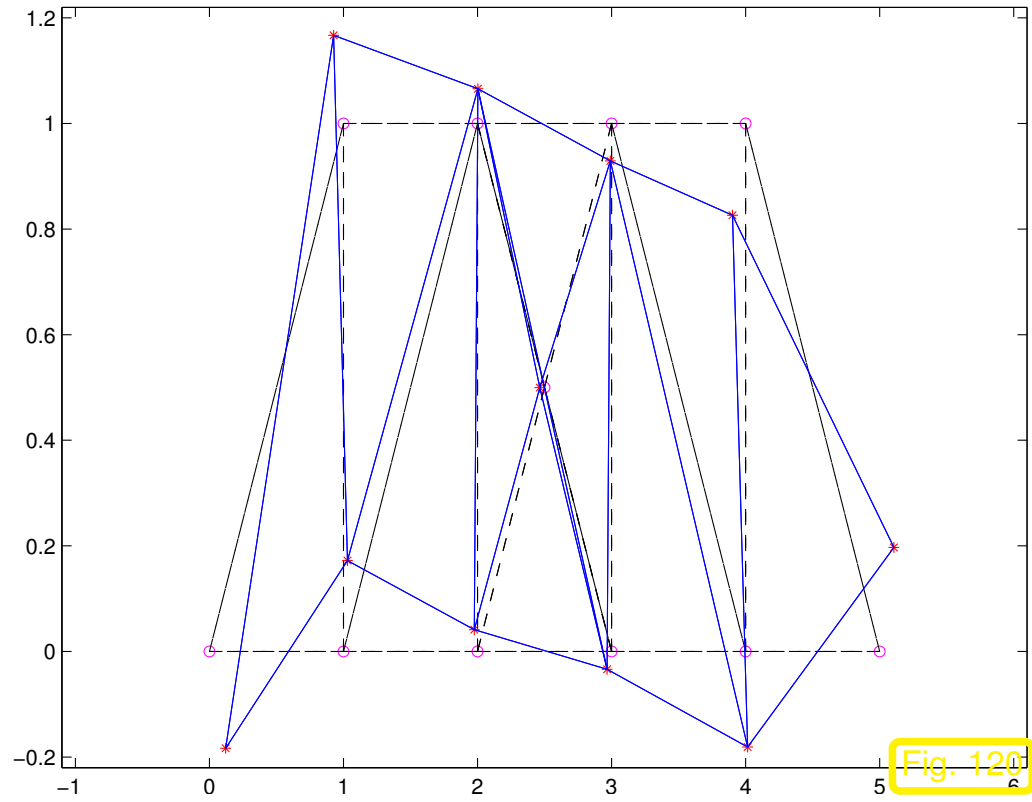
◁ resonant frequencies of bridge truss from Fig. 116.

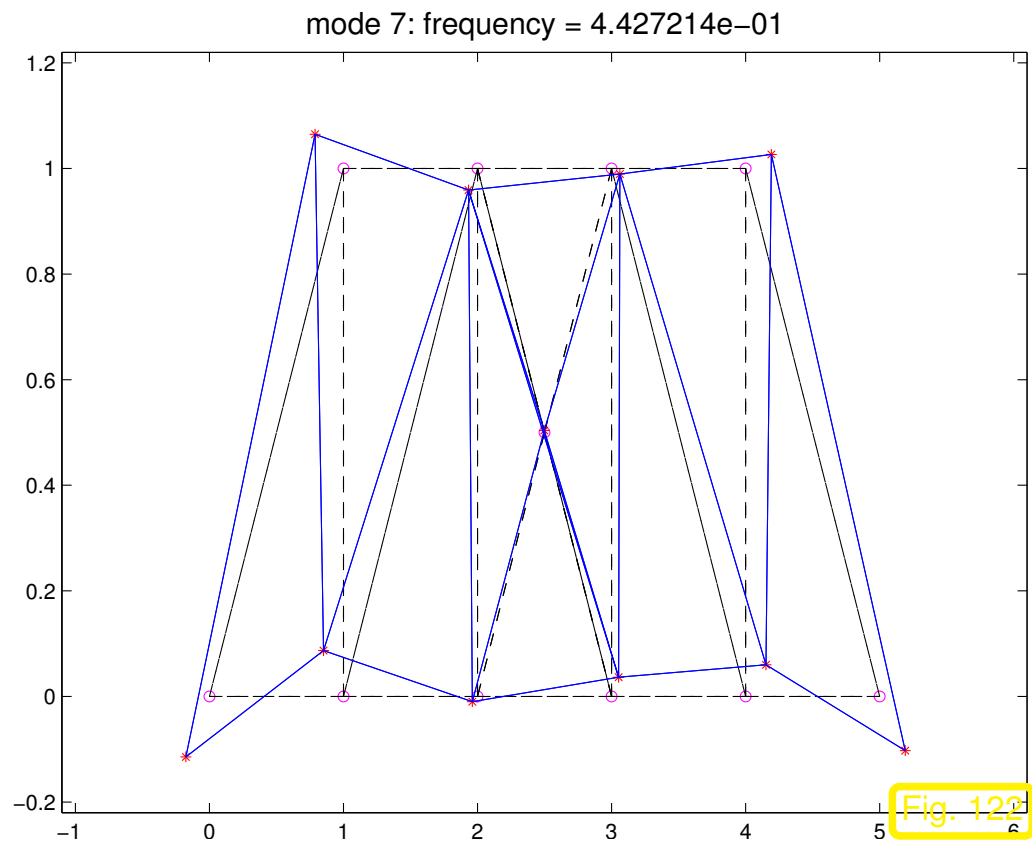
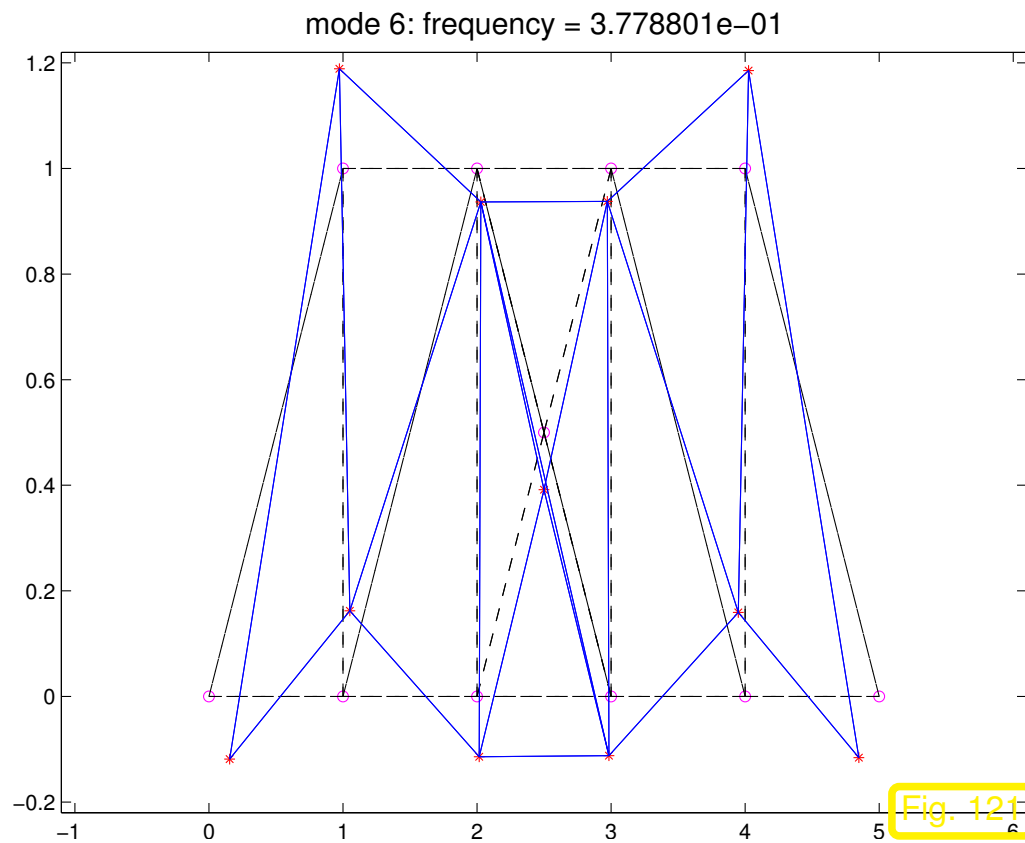
The stiffness matrix will always possess three zero eigenvalues corresponding to **rigid body modes** (= displacements without change of length of the rods)

mode 4: frequency =  $6.606390e-02$



mode 5: frequency =  $3.004450e-01$





To compute *a few* of a truss's lowest resonant frequencies and excitable mode, we need efficient numerical methods for the following tasks. Obviously, Code 6.3.80 cannot be used for large trusses, because `eig` invariably operates on dense matrices and will be prohibitively slow and gobble up huge amounts of memory, also recall the discussion of Code 6.3.50.

Task: Compute  $m$ ,  $m \ll n$ , of the smallest/largest (in modulus) eigenvalues of  $\mathbf{A} = \mathbf{A}^H \in \mathbb{C}^{n,n}$  and associated eigenvectors.

Of course, we aim to tackle this task by iterative methods generalizing power iteration ( $\rightarrow$  Sect. 6.3.1) and inverse iteration ( $\rightarrow$  Sect. 6.3.2).

### 6.3.4.1 Orthogonalization

Preliminary considerations (in  $\mathbb{R}$ ,  $m = 2$ ):

According to Cor. 6.1.9: For  $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$  there is a factorization  $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^T$  with  $\mathbf{D} = \text{diag}(\lambda_1, \dots, \lambda_n)$ ,  $\lambda_j \in \mathbb{R}$ ,  $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ , and  $\mathbf{U}$  orthogonal. Thus,  $\mathbf{u}_j := (\mathbf{U})_{:,j}$ ,  $j = 1, \dots, n$ , are (mutually orthogonal) eigenvectors of  $\mathbf{A}$ .

Assume  $0 \leq \lambda_1 \leq \dots \leq \lambda_{n-2} < \lambda_{n-1} < \lambda_n$  (largest eigenvalues are simple).

If we just carry out the direct power iteration (6.3.11) for two vectors both sequences will converge to the largest (in modulus) eigenvector. However, we recall that all eigenvectors are mutually orthogonal. This suggests that we orthogonalize the iterates of the second power iteration (that is to yield the eigenvector for the second largest eigenvalue) with respect to those of the first. This idea spawns the following iteration, cf. Gram-Schmidt orthogonalization in (5.2.11):

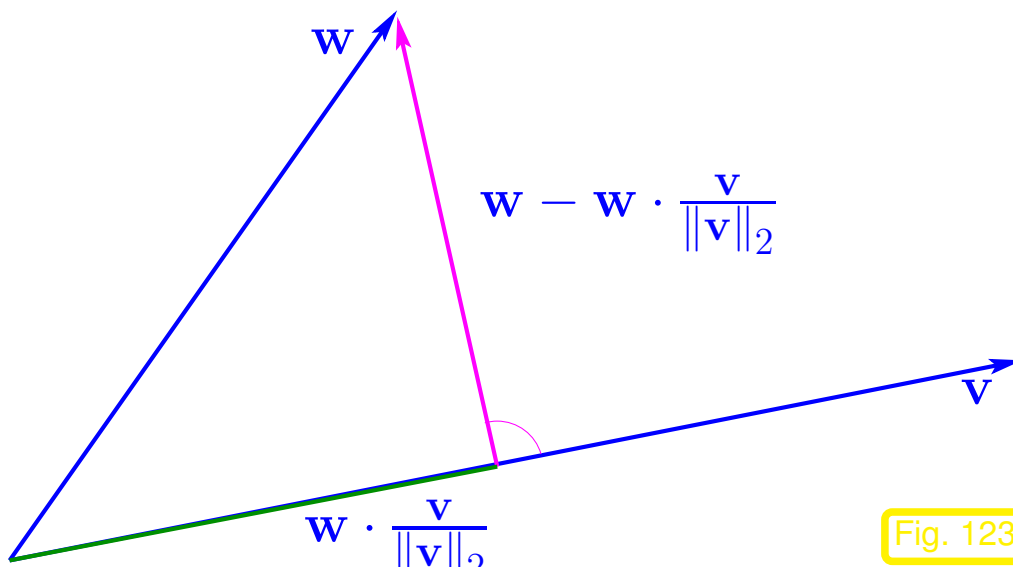
Code 6.3.82: one step of subspace power iteration,  $m = 2$

```

1 function [v,w] = sspowitstep(A,v,w)
2 v = A*v; w = A*w; % "power iteration", cf. (6.3.11)
3 % orthogonalization, cf. Gram-Schmidt orthogonalization (5.2.11)
4 v = v/norm(v); w = w - dot(v,w)*v; w = w/norm(w); % now w ⊥ v

```

R. Hiptmair  
rev 38286,  
November  
10, 2011



◁ Orthogonalization of two vectors  
(see Line 4 of Code 6.3.81)

Fig. 123

Analysis through eigenvector expansions ( $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ ,  $\|\mathbf{v}\|_2 = \|\mathbf{w}\|_2 = 1$ )

$$\mathbf{v} = \sum_{i=1}^n \alpha_j \mathbf{u}_j \quad , \quad \mathbf{w} = \sum_{i=1}^n \beta_j \mathbf{u}_j \quad ,$$

$$\Rightarrow \mathbf{A}\mathbf{v} = \sum_{i=1}^n \lambda_j \alpha_j \mathbf{u}_j \quad , \quad \mathbf{A}\mathbf{w} = \sum_{i=1}^n \lambda_j \beta_j \mathbf{u}_j \quad ,$$

$$\mathbf{v}_0 := \frac{\mathbf{v}}{\|\mathbf{v}\|_2} = \left( \sum_{i=1}^n \lambda_j^2 \alpha_j^2 \right)^{-1/2} \sum_{i=1}^n \lambda_j \alpha_j \mathbf{u}_j \quad ,$$

$$\mathbf{A}\mathbf{w} - (\mathbf{v}_0^\top \mathbf{A}\mathbf{w}) \mathbf{v}_0 = \sum_{i=1}^n \left( \beta_j - \left( \frac{\sum_{i=1}^n \lambda_j^2 \alpha_j \beta_j}{\sum_{i=1}^n \lambda_j^2 \alpha_j^2} \right) \alpha_j \right) \lambda_j \mathbf{u}_j \quad .$$

We notice that  $\mathbf{v}$  is just mapped to the next iterate in the regular direct power iteration (6.3.11). After many steps, it will be very close to  $\mathbf{u}_n$ , and, therefore, we may now assume  $\mathbf{v} = \mathbf{u}_n \Leftrightarrow \alpha_j = \delta_{j,n}$  (Kronecker symbol).

$$\mathbf{z} := \mathbf{A}\mathbf{w} - (\mathbf{v}_0^\top \mathbf{A}\mathbf{w})\mathbf{v}_0 = 0 \cdot \mathbf{u}_n + \sum_{i=1}^{n-1} \lambda_i \beta_i \mathbf{u}_i ,$$

$$\mathbf{w}^{(\text{new})} := \frac{\mathbf{z}}{\|\mathbf{z}\|_2} = \left( \sum_{i=1}^{n-1} \lambda_i^2 \beta_i^2 \right)^{-1/2} \sum_{i=1}^{n-1} \lambda_i \beta_i \mathbf{u}_i .$$

The sequence  $\mathbf{w}^{(k)}$  produced by repeated application of the mapping given by Code 6.3.81 asymptotically (that is, when  $\mathbf{v}^{(k)}$  has already converged to  $\mathbf{u}_n$ ) agrees with the sequence produced by the direct power method for  $\tilde{\mathbf{A}} := \mathbf{U} \text{diag}(\lambda_1, \dots, \lambda_{n-1}, 0)$ . Its convergence will be governed by the relative gap  $\lambda_{n-2}/\lambda_{n-1}$ , see Thm. 6.3.19.

However: if  $\mathbf{v}^{(k)}$  itself converges slowly, this reasoning does not apply.

*Example 6.3.83* (Subspace power iteration with orthogonal projection).

☞ construction of matrix  $\mathbf{A} = \mathbf{A}^\top$  as in Ex. 6.3.58



## Code 6.3.84: power iteration with orthogonal projection for two vectors

```

1 function sppowitdriver(d,maxit)
2 % monitor power iteration with orthogonal projection for finding
3 % the two largest (in modulus) eigenvalues and associated eigenvectors
4 % of a symmetric matrix with prescribed eigenvalues passed in d
5 if (nargin < 10), maxit = 20; end
6 if (nargin < 1), d = (1:10)'; end
7 % Generate matrix
8 n = length(d);
9 Z = diag(sqrt(1:n),0) + ones(n,n);
10 [Q,R] = qr(Z); % generate orthogonal matrix
11 A = Q*diag(d,0)*Q'; % "synthetic"  $A = A^T$  with spectrum  $\sigma(A) = \{d_1, \dots, d_n\}$ 
12 % Compute "exact" eigenvectors and eigenvalues
13 [V,D] = eig(A); [d,idx] = sort(diag(D)),
14 v_ex = V(:,idx(n)); w_ex = V(:,idx(n-1));
15 lv_ex = d(n); lw_ex = d(n-1);
16
17 v = ones(n,1); w = (-1).^v; % (Arbitrary) initial guess for eigenvectors
18 v = v/norm(v); w = w/norm(w);
19 result = [];
20 for k=1:maxit
21     v_new = A*v; w_new = A*w; % "power iteration", cf. (6.3.11)
22     % Rayleigh quotients provide approximate eigenvalues

```

```
3 lv = dot(v_new,v); lw = dot(w_new,w);
4 % orthogonalization, cf. Gram-Schmidt orthogonalization (5.2.11):  $w \perp v$ 
5 v = v_new/norm(v_new); w = w_new - dot(v,w_new)*v; w = w/norm(w);
6 % Record errors in eigenvalue and eigenvector approximations. Note that the
7 % direction of the eigenvectors is not specified.
8 result = [result; k, abs(lv-lv_ex), abs(lw-lw_ex), ...
9           min(norm(v-v_ex), norm(v+v_ex)),
10          min(norm(w-w_ex), norm(w+w_ex))];
11 end
12 figure('name','sspowit');
13 semilogy(result(:,1),result(:,2),'m-+',...
14          result(:,1),result(:,3),'r-*',...
15          result(:,1),result(:,4),'k-^',...
16          result(:,1),result(:,5),'b-p');
17 title('d = [0.5*(1:8),9.5,10]');
18 xlabel('\bf power iteration step','fontsize',14);
19 ylabel('\bf error','fontsize',14);
20 legend('error in \lambda_n','error in \lambda_{n-1}','error in
21 v','error in w','location','northeast');
22 print -depsc2 '../PICTURES/sspowitcvgl.eps';
23 rates = result(2:end,2:end)./result(1:end-1,2:end);
```

```
4 figure ('name', 'rates');
5 plot (result(2:end,1), rates(:,1), 'm-+', ...
6       result(2:end,1), rates(:,2), 'r-*', ...
7       result(2:end,1), rates(:,3), 'k-^', ...
8       result(2:end,1), rates(:,4), 'b-p');
9 axis ([0 maxit 0.5 1]);
10 title ('d = [0.5*(1:8), 9.5, 10]');
11 xlabel ('\bf power iteration step', 'fontsize', 14);
12 ylabel ('\bf error quotient', 'fontsize', 14);
13 legend ('error in \lambda_n', 'error in \lambda_{n-1}', 'error in
14         v', 'error in w', 'location', 'southeast');
15 print -depsc2 '../PICTURES/sspowitcvgrates1.eps';
```

$\sigma(\mathbf{A}) = \{1, 2, \dots, 10\}$ :

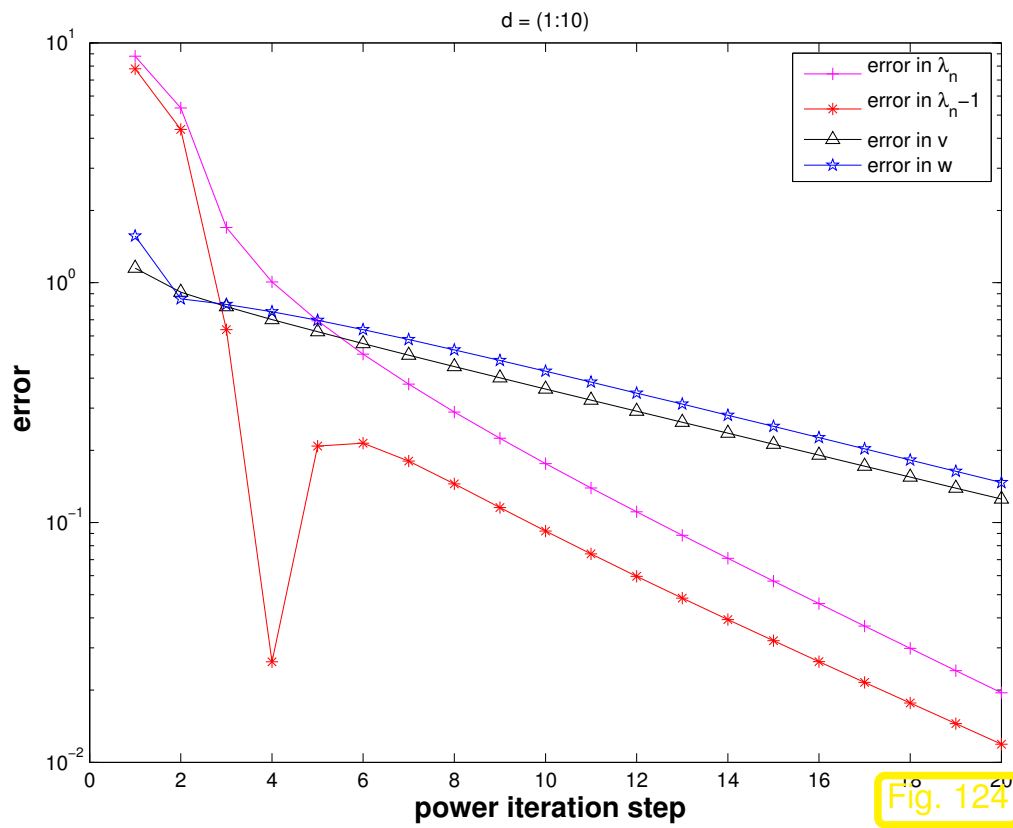


Fig. 124

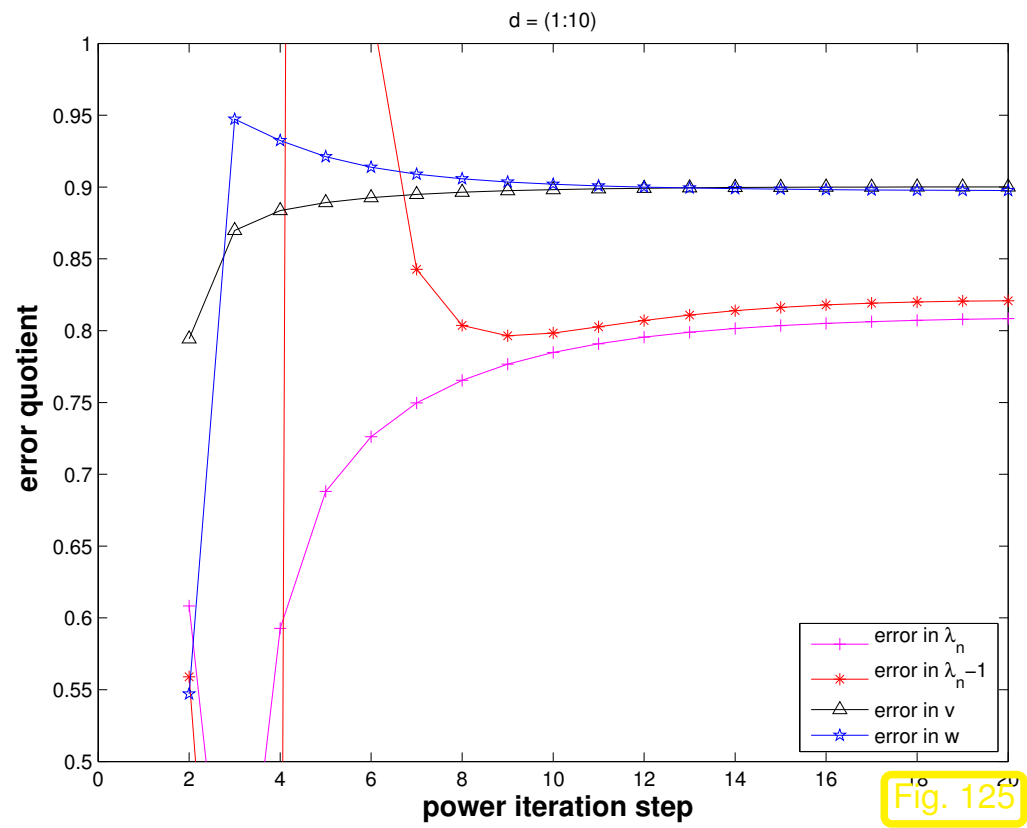


Fig. 125

$\sigma(\mathbf{A}) = \{0.5, 1, \dots, 4, 9.5, 10\}$ :

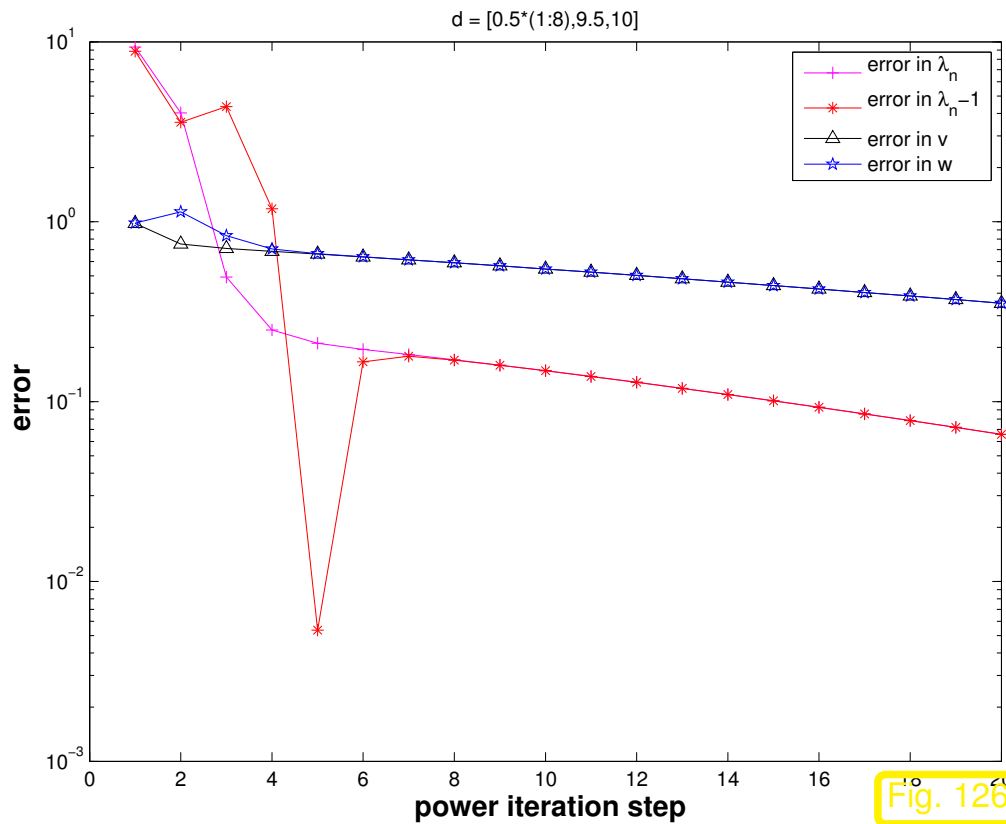


Fig. 126

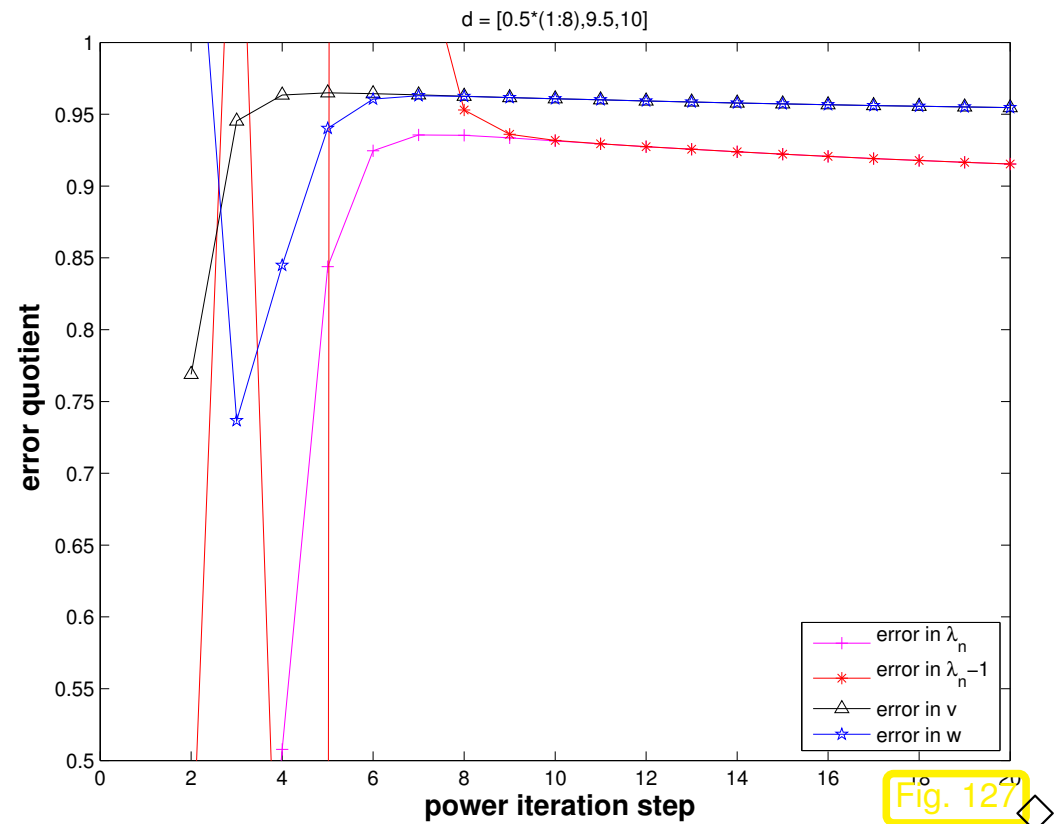


Fig. 127

Issue: generalization of orthogonalization idea to subspaces of dimension  $> 2$

Nothing new:

**Gram-Schmidt orthonormalization**

( $\rightarrow$  [48, Thm. 4.8], [27, Alg. 6.1], [51, Sect. 3.4.3])

Given: linearly independent vectors  $\mathbf{v}_1, \dots, \mathbf{v}_m \in \mathbb{R}^n$ ,  $m \in \mathbb{N}$

Sought: vectors  $\mathbf{q}_1, \dots, \mathbf{q}_m \in \mathbb{R}^n$  such that

$$\textcircled{2} \quad \mathbf{q}_l^\top \mathbf{q}_k = \delta_{lk} \quad (\text{orthonormality}), \quad (6.3.85)$$

$$\textcircled{2} \quad \text{Span} \{ \mathbf{q}_1, \dots, \mathbf{q}_k \} = \text{Span} \{ \mathbf{v}_1, \dots, \mathbf{v}_k \} \quad \text{for all } k = 1, \dots, m. \quad (6.3.86)$$

Constructive proof & algorithm for Gram-Schmidt orthonormalization:

$$\begin{aligned} \mathbf{z}_1 &= \mathbf{v}_1, \\ \mathbf{z}_2 &= \mathbf{v}_2 - \frac{\mathbf{v}_2^\top \mathbf{z}_1}{\mathbf{z}_1^\top \mathbf{z}_1} \mathbf{z}_1, \\ \mathbf{z}_3 &= \mathbf{v}_3 - \frac{\mathbf{v}_3^\top \mathbf{z}_1}{\mathbf{z}_1^\top \mathbf{z}_1} \mathbf{z}_1 - \frac{\mathbf{v}_3^\top \mathbf{z}_2}{\mathbf{z}_2^\top \mathbf{z}_2} \mathbf{z}_2, \\ &\vdots \end{aligned} \quad (6.3.87)$$

$$\text{+ normalization} \quad \mathbf{q}_k = \frac{\mathbf{z}_k}{\|\mathbf{z}_k\|_2}, \quad k = 1, \dots, m. \quad (6.3.88)$$

Easy computation: the vectors  $\mathbf{q}_1, \dots, \mathbf{q}_m$  produced by (6.3.87) satisfy (6.3.85) and (6.3.86).

Code 6.3.89: **Gram-Schmidt orthonormalization** (do not use, unstable algorithm!)

```

1 function Q = gso(V)
2 % Gram-Schmidt orthonormalization of the columns of  $\mathbf{V} \in \mathbb{R}^{n,m}$ , see
3 % (6.3.87). The vectors  $\mathbf{q}_1, \dots, \mathbf{q}_m$  are returned as the columns of
4 % the orthogonal matrix  $\mathbf{Q}$ .
5 m = size(V, 2);

```

```

6 Q = V(:,1)/norm(V(:,1)); % normalization
7 for l=2:m
8     q = V(:,l);
9     % orthogonalization
10    for k=1:l-1
11        q = q - dot(Q(:,k),V(:,l))*Q(:,k);
12    end
13    Q = [Q, q/norm(q)]; % normalization
14 end

```



Warning! Code 6.3.88 provides an unstable implementation of Gram-Schmidt orthonormalization: for large  $n, m$  impact of round-off will destroy the orthogonality of the columns of  $Q$ .

A stable implementation of Gram-Schmidt orthogonalization of the columns of a matrix  $V \in \mathbb{K}^{n,m}$ ,  $m \leq n$ , is provided by the following MATLAB command:

$[Q, R] = \text{qr}(V, 0)$  (Asymptotic computational cost:  $O(m^2n)$ )

dummy return value (for our purposes)

dummy argument

Detailed description of the algorithm behind `qr` and meaning of the return value `R` → Sect. 2.8.

*Example 6.3.90* (`qr` based orthogonalization,  $m = 2$ ).

The following two MATLAB code snippets perform the same function, *cf.* Code 6.3.81:

```
1 v = v/norm(v);
2 w = w - dot(v,w)*v;
3 w = w/norm(w);
```

```
1 [Q,R] = qr([v,w],0);
2 v = Q(:,1);
3 w = Q(:,2);
```

R. Hiptmair  
rev 38286,  
November  
10, 2011

Explanation ➤ discussion of Gram-Schmidt orthonormalization.



Code 6.3.91: General subspace power iteration step with `qr` based orthonormalization

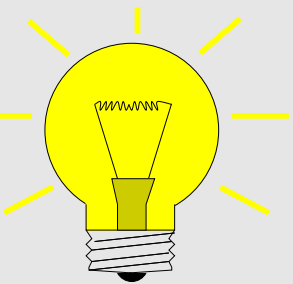
```
1 function V = sspowitstep(A,V)
2 % power iteration with orthonormalization for  $A = A^T$ .
3 % columns of matrix V span subspace for power iteration.
4 V = A*V; % actual power iteration on individual columns
5 [V,R] = qr(V,0); % Gram-Schmidt orthonormalization (6.3.87)
```



Observations on Code 6.3.81:

- the first column of  $\mathbf{V}$ ,  $(\mathbf{V})_{:,1}$ , is a sequence of vectors created by the standard direct power method (6.3.11).
- reasoning: the other columns of  $\mathbf{V}$ , after each multiplication with  $\mathbf{A}$  can be expected to contain a significant component in the direction of the eigenvector associated with the eigenvalue of largest modulus.

R. Hiptmair  
rev 38286,  
November  
10, 2011



Idea: use information in  $(\mathbf{V})_{:,2}, \dots, (\mathbf{V})_{:,end}$  to accelerate convergence of  $(\mathbf{V})_{:,1}$ .

Since the columns of  $\mathbf{V}$  span a *subspace* of  $\mathbb{R}^n$ , this idea can be recast as the following task:

Task: given  $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{K}^n$ ,  $k \ll n$ , extract (good approximations of) eigenvectors of  $\mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n}$  contained in  $\text{Span}\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$ .

We take for granted that  $\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$  is linearly independent.

Assumption:  $\text{Eig}_{\mathbf{A}}(\lambda) \cap V := \text{Span}\{\mathbf{v}_1, \dots, \mathbf{v}_m\} \neq \{0\}$

$\Leftrightarrow$   $V$  contains an eigenvector of  $\mathbf{A}$

$$\Leftrightarrow \exists \mathbf{w} \in V \setminus \{0\}: \mathbf{A}\mathbf{w} = \lambda\mathbf{w}$$

$$\Leftrightarrow \exists \mathbf{u} \in \mathbb{K}^m \setminus \{0\}: \mathbf{A}\mathbf{V}\mathbf{u} = \lambda\mathbf{V}\mathbf{u}$$

$$\Rightarrow \exists \mathbf{u} \in \mathbb{K}^m \setminus \{0\}: \mathbf{V}^H \mathbf{A}\mathbf{V}\mathbf{u} = \lambda \mathbf{V}^H \mathbf{V}\mathbf{u}, \tag{6.3.92}$$

where  $\mathbf{V} := (\mathbf{v}_1, \dots, \mathbf{v}_m) \in \mathbb{K}^{n,m}$  and we used

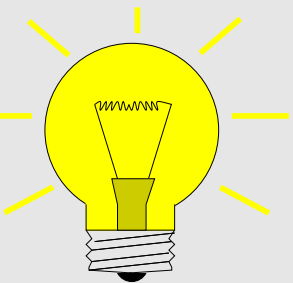
$$V = \{\mathbf{V}\mathbf{u} : \mathbf{u} \in \mathbb{K}^m\} \quad (\text{linear combinations of the } \mathbf{v}_i).$$

(6.3.92)  $\Rightarrow$   $\mathbf{u} \in \mathbb{K}^m \setminus \{0\}$  solves  $m \times m$  **generalized eigenvalue problem**

$$(\mathbf{V}^H \mathbf{A}\mathbf{V})\mathbf{u} = \lambda(\mathbf{V}^H \mathbf{V})\mathbf{u}. \tag{6.3.93}$$

Note:  $\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$  linearly independent  
 $\Updownarrow$   
 $\mathbf{V}$  has full rank  $m$  ( $\rightarrow$  Def. 2.0.2)  
 $\Updownarrow$   
 $\mathbf{V}^H \mathbf{V}$  is symmetric positive definite ( $\rightarrow$  Def. 2.7.9)

If our initial assumption holds true and  $\mathbf{u}$  solves (6.3.93) and is a simple eigenvalue, a corresponding  $\mathbf{x} \in \text{Eig}_{\mathbf{A}}(\lambda)$  can be recovered as  $\mathbf{x} = \mathbf{V}\mathbf{u}$ .



Idea: Given a subspace  $V = \text{Im}(\mathbf{V}) \subset \mathbb{K}^n$ ,  $\mathbf{V} \in \mathbb{K}^{n,m}$ ,  $\dim(V) = m$ , obtain *approximations* of (a few) eigenvalues and eigenvectors  $\mathbf{x}_1, \dots, \mathbf{x}_m$  of  $\mathbf{A}$  by

① solving the generalized eigenvalue problem (6.3.93)

$\rightarrow$  eigenvectors  $\mathbf{u}_1, \dots, \mathbf{u}_k \in \mathbb{K}^m$  (linearly independent),

② and transforming them back according to  $\mathbf{x}_k = \mathbf{V}\mathbf{u}_k$ ,  $k = 1, \dots, m$ .

Terminology: (6.3.93) is called the **Ritz projection** of EVP  $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$  onto  $V$

Terminology:  $\sigma(\mathbf{V}^H\mathbf{A}\mathbf{V}) \hat{=} \mathbf{Ritz\ values}$ ,  
eigenvectors of  $\mathbf{V}^H\mathbf{A}\mathbf{V} \hat{=} \mathbf{Ritz\ vectors}$

Example: Ritz projection of  $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$  onto  $\text{Span}\{\mathbf{v}, \mathbf{w}\}$ :

$$(\mathbf{v}, \mathbf{w})^H \mathbf{A} (\mathbf{v}, \mathbf{w}) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \lambda (\mathbf{v}, \mathbf{w})^H (\mathbf{v}, \mathbf{w}) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} .$$

Note: If  $\mathbf{V}$  is *unitary* ( $\rightarrow$  Def. 2.8.5), then the generalized eigenvalue problem (6.3.93) will become a standard linear eigenvalue problem.

*Remark 6.3.94* (Justification of Ritz projection by min-max theorem).

We revisit  $m = 2$ , see Code 6.3.81. Recall that by the min-max theorem Thm. 6.3.39

$$\mathbf{u}_n = \operatorname{argmax}_{\mathbf{x} \in \mathbb{R}^n} \rho_{\mathbf{A}}(\mathbf{x}) \quad , \quad \mathbf{u}_{n-1} = \operatorname{argmax}_{\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \perp \mathbf{u}_n} \rho_{\mathbf{A}}(\mathbf{x}) . \quad (6.3.95)$$

Idea: maximize Rayleigh quotient over  $\text{Span}\{\mathbf{v}, \mathbf{w}\}$ , where  $\mathbf{v}$ ,  $\mathbf{w}$  are output by Code 6.3.81. This leads to the optimization problem

$$(\alpha^*, \beta^*) := \underset{\alpha, \beta \in \mathbb{R}, \alpha^2 + \beta^2 = 1}{\operatorname{argmax}} \rho_{\mathbf{A}}(\alpha \mathbf{v} + \beta \mathbf{w}) = \underset{\alpha, \beta \in \mathbb{R}, \alpha^2 + \beta^2 = 1}{\operatorname{argmax}} \rho_{(\mathbf{v}, \mathbf{w})^\top \mathbf{A}(\mathbf{v}, \mathbf{w})} \left( \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \right). \quad (6.3.96)$$

Then a better approximation for the eigenvector to the largest eigenvalue is

$$\mathbf{v}^* := \alpha^* \mathbf{v} + \beta^* \mathbf{w}.$$

Note that  $\|\mathbf{v}^*\|_2 = 1$ , if both  $\mathbf{v}$  and  $\mathbf{w}$  are normalized, which is guaranteed in Code 6.3.81.

Then, orthogonalizing  $\mathbf{w}$  w.r.t  $\mathbf{v}^*$  will produce a new iterate  $\mathbf{w}^*$ .

Again the min-max theorem Thm. 6.3.39 tells us that we can find  $(\alpha^*, \beta^*)^\top$  as eigenvector to the largest eigenvalue of

$$(\mathbf{v}, \mathbf{w})^\top \mathbf{A}(\mathbf{v}, \mathbf{w}) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \lambda \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \quad (6.3.97)$$

Since eigenvectors of symmetric matrices are mutually orthogonal, we find  $\mathbf{w}^* = \alpha_2 \mathbf{v} + \beta_2 \mathbf{w}$ , where  $(\alpha_2, \beta_2)^\top$  is the eigenvector of (6.3.97) belonging to the smallest eigenvalue. This assumes orthonormal vectors  $\mathbf{v}$ ,  $\mathbf{w}$ .

## Code 6.3.98: one step of subspace power iteration with Ritz projection, matrix version

NumCSE,  
autumn  
2010

```
1 function V = sspowitsteprp(A,V)
2 V = A*V;           % power iteration applied to columns of V
3 [Q,R] = qr(V,0);  % orthonormalization, see Sect. 6.3.4.1
4 [U,D] = eig(Q'*A*Q); % Solve Ritz projected  $m \times m$  eigenvalue problem
5 V = Q*U;          % recover approximate eigenvectors
6 ev = diag(D);     % approximate eigenvalues
```

Note that the orthogonalization step in Code 6.3.97 is actually redundant, if exact arithmetic could be employed, because the Ritz projection could also be realized by solving the generalized eigenvalue problem.

R. Hiptmair  
rev 38286,  
November  
10, 2011

However, prior orthogonalization is essential for numerical stability ( $\rightarrow$  Def. 2.5.11), *cf.* the discussion in Sect. 2.8.

*Example 6.3.99* (Power iteration with Ritz projection).

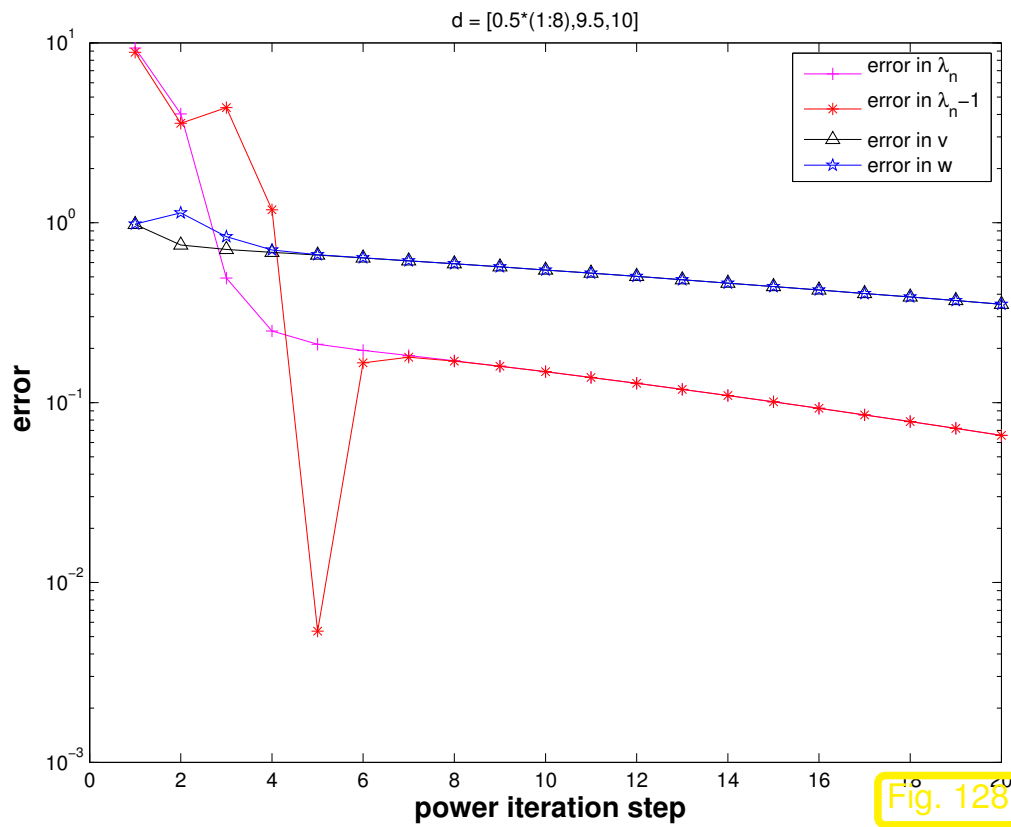
## Code 6.3.100: Main loop: power iteration with Ritz projection for two eigenvectors

```

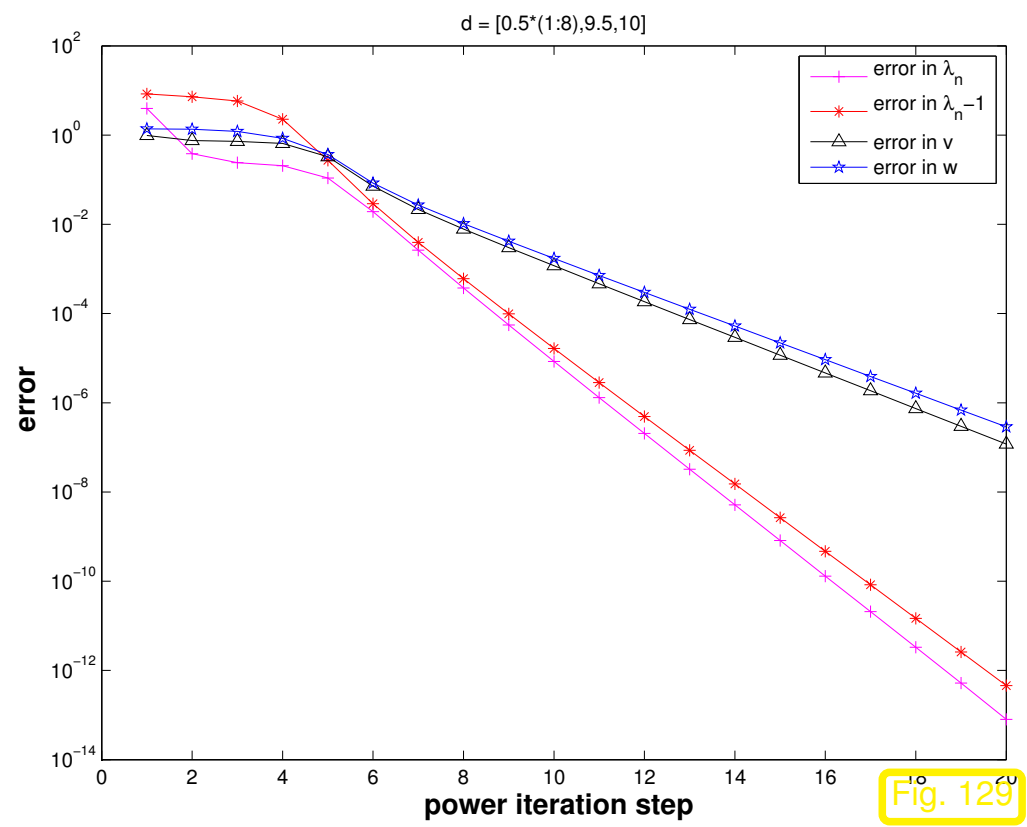
1 % See Code 6.3.83 for generation of matrix A and output
2 for k=1:maxit
3     v_new = A*v; w_new = A*w;      % "power iteration", cf. (6.3.11)
4     [Q,R] = qr([v_new,w_new],0); % orthogonalization, see Sect. 6.3.4.1
5     [U,D] = eig(Q'*A*Q);          % Solve Ritz projected eigenvalue problem
6     [ev,idx] = sort(abs(diag(D))), % Sort eigenvalues
7     w = Q*U(:,idx(1)); v = Q*U(:,idx(2)); % Recover approximate
        eigenvectors
8
9     % Record errors in eigenvalue and eigenvector approximations. Note that the
10    % direction of the eigenvectors is not specified.
11    result = [result; k, abs(ev(2)-lv_ex), abs(ev(1)-lw_ex), ...
12             min(norm(v-v_ex), norm(v+v_ex)),
13             min(norm(w-w_ex), norm(w+w_ex))];
14 end

```

Matrix as in Ex. 6.3.83,  $\sigma(\mathbf{A}) = \{0.5, 1, \dots, 4, 9.5, 10\}$ :

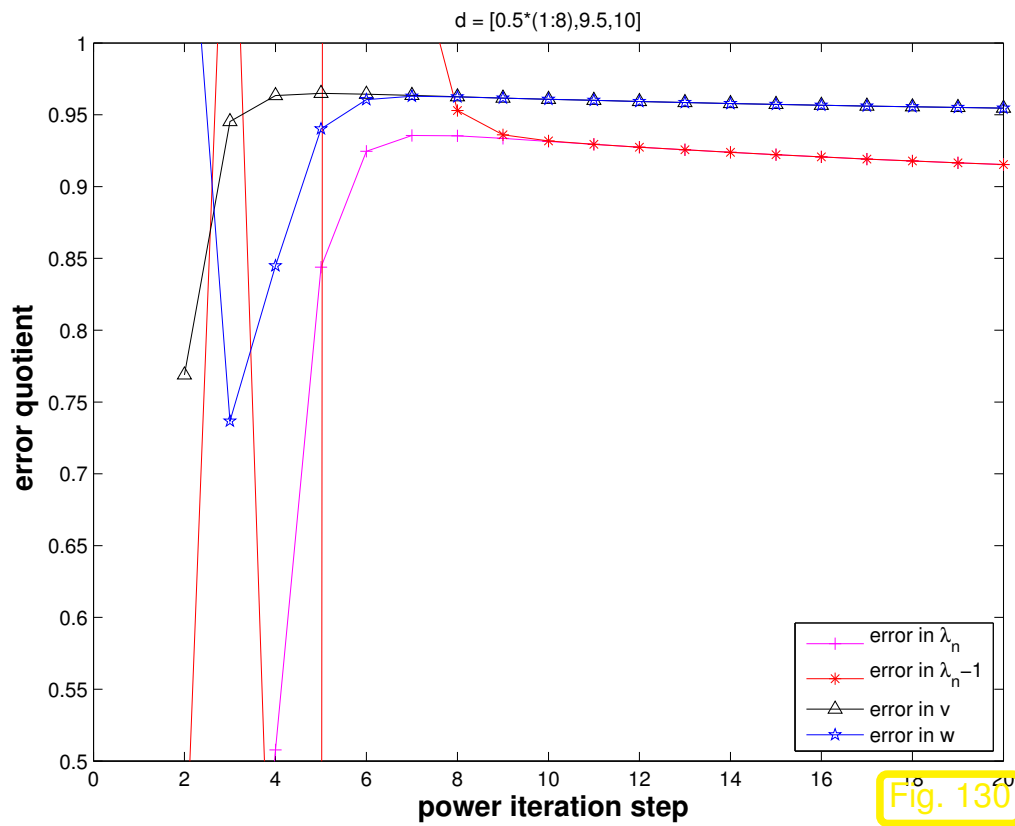


simple orthonormalization, Ex. 6.3.83

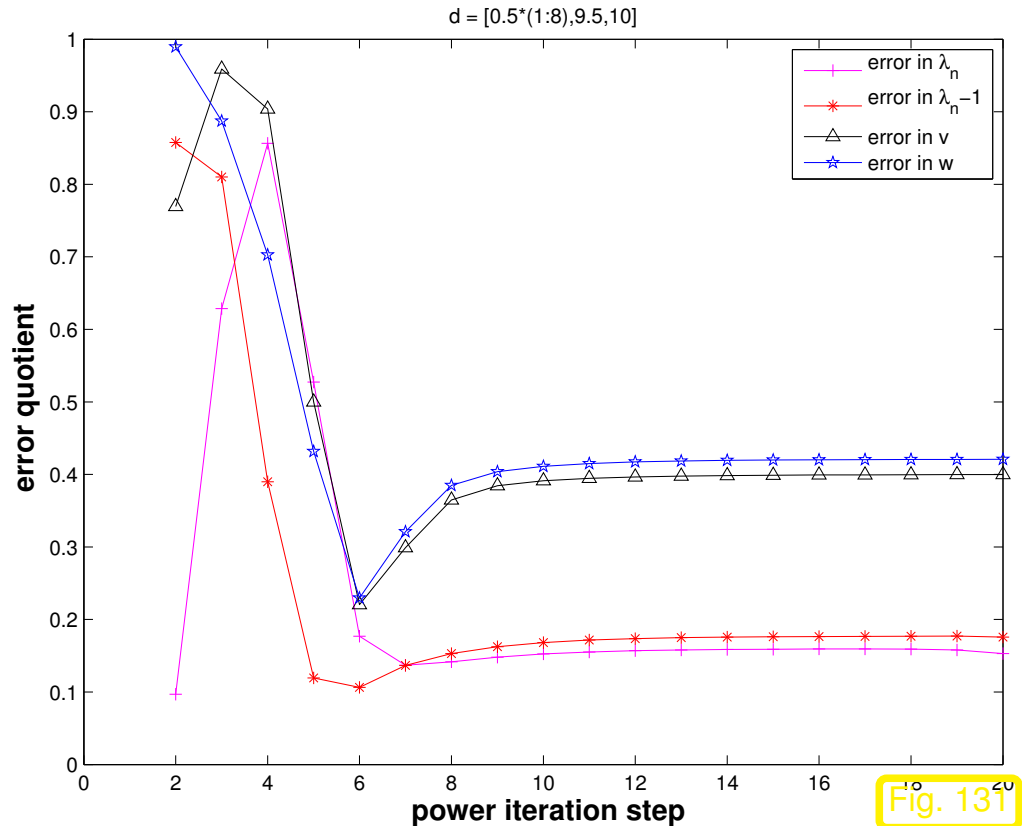


with Ritz projection, Code 6.3.97





simple orthonormalization, Ex. 6.3.83



with Ritz projection, Code 6.3.97

Observation: tremendous acceleration of power iteration through Ritz projection, convergence still linear but with much better rates.



In Code: 6.3.97: diagonal entries of  $\mathbf{D}$  provide approximations of eigenvalues. Their (relative) changes can be used as a termination criterion.

Algorithm 6.3.101 (Subspace variant of direct power method with Ritz projection).

Code 6.3.102: Subspace power iteration with Ritz projection

```

1 function [ev,V] = sspowitrp(A,k,m,tol,maxit)
2 % Power iteration with Ritz projection for matrix  $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$ :
3 % Subspace of dimension  $m \leq n$  is used to compute the  $k \leq m$  largest
4 % eigenvalues of  $\mathbf{A}$  and associated eigenvectors.
5 n = size(A,1); V = eye(n,m); d = zeros(m,1); % (Arbitrary) initial
   eigenvectors
6 % The approximate eigenvectors are stored in the columns of  $\mathbf{V} \in \mathbb{R}^{n,m}$ 
7 for i=1:maxit
8     [Q,R] = qr(A*V,0); % Power iteration and orthonormalization
9     [U,D] = eig(Q'*A*Q); % Small  $m \times m$  eigenvalue problem for Ritz projection
10    [ev,idx] = sort(diag(D)); % eigenvalue approximations in diagonal of  $\mathbf{D}$ 
11    V = Q*U; % 2nd part of Ritz projection
12    if (abs(ev-d) < tol*max(abs(ev))), break; end

```

```

3   d = ev;
4 end
5   ev = ev(m-k+1:end);
6   V = V(:,idx(m-k+1:end));

```

*Example 6.3.103* (Convergence of subspace variant of direct power method).

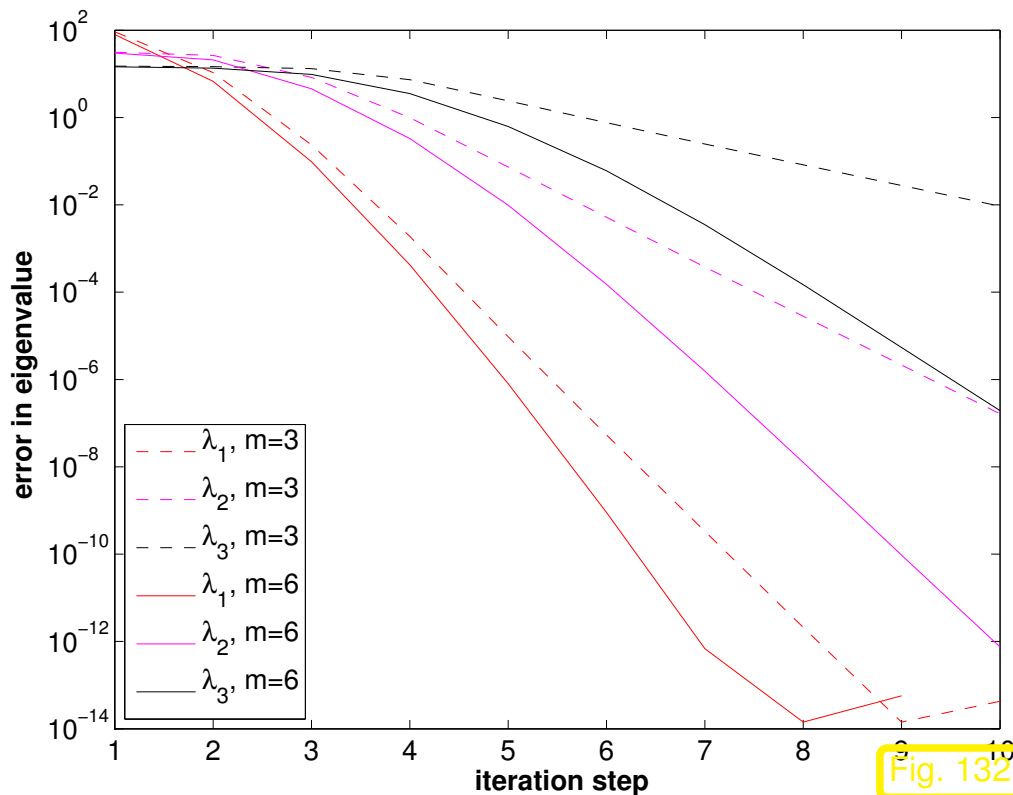


Fig. 132

S.p.d. test matrix:  $a_{ij} := \min\{\frac{i}{j}, \frac{j}{i}\}$

$n=200$ ;  $A = \text{gallery}('lehmer', n)$ ;

“Initial eigenvector guesses”:

$V = \text{eye}(n, m)$ ;

- Observation:  
linear convergence of eigenvalues
- choice  $m > k$  boosts convergence  
of eigenvalues



*Remark 6.3.104* (Subspace power methods).

Analogous to Alg. 6.3.101: construction of subspace variants of inverse iteration ( $\rightarrow$  Code 6.3.51), PINVIT (6.3.60), and Rayleigh quotient iteration (6.3.56).



## 6.4 Krylov Subspace Methods [35, Sect. 30]

All power methods ( $\rightarrow$  Sect. 6.3) for the eigenvalue problem (EVP)  $\mathbf{Ax} = \lambda\mathbf{x}$  only rely on the last iterate to determine the next one (1-point methods, *cf.* (4.1.2))

➤ NO MEMORY, *cf.* discussion in the beginning of Sect. 5.2.

“Memory for power iterations”: pursue same idea that led from the gradient method, Alg. 5.1.11, to the conjugate gradient method, Alg. 5.2.17: use information from previous iterates to achieve efficient minimization over larger and larger **subspaces**.

Min-max theorem, Thm. 6.3.39 :  $\mathbf{A} = \mathbf{A}^H \Rightarrow$  EVPs  $\Leftrightarrow$  Finding extrema/stationary points of Rayleigh quotient ( $\rightarrow$  Def. 6.3.15)

Setting: EVP  $\mathbf{Ax} = \lambda\mathbf{x}$  for real s.p.d. ( $\rightarrow$  Def. 2.7.9) matrix  $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$

notations used below:  $0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ : eigenvalues of  $\mathbf{A}$ , counted with multiplicity, see Def. 6.1.1,

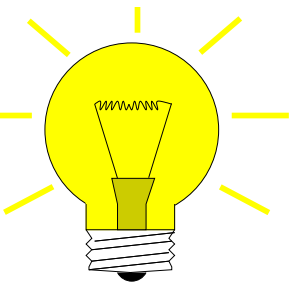
$\mathbf{u}_1, \dots, \mathbf{u}_n \hat{=}$  corresponding orthonormal eigenvectors, *cf.* Cor. 6.1.9.

►  $\mathbf{AU} = \mathbf{DU}$  ,  $\mathbf{U} = (\mathbf{u}_1, \dots, \mathbf{u}_n) \in \mathbb{R}^{n,n}$  ,  $\mathbf{D} = \text{diag}(\lambda_1, \dots, \lambda_n)$  .

We recall

- the direct power method (6.3.11) from Sect. 6.3.1
- and the inverse iteration from Sect. 6.3.2

and how they produce sequences  $(\mathbf{z}^{(k)})_{k \in \mathbb{N}_0}$  of vectors that are supposed to converge to a vector  $\in \text{Eig}_{\mathbf{A}}(\lambda_1)$  or  $\in \text{Eig}_{\mathbf{A}}(\lambda_n)$ , respectively.



Idea: Better  $\mathbf{z}^{(k)}$  from Ritz projection onto  $V := \text{Span} \{ \mathbf{z}^{(0)}, \dots, \mathbf{z}^{(k)} \}$   
(= space spanned by previous iterates)

Recall ( $\rightarrow$  Code 6.3.97) **Ritz projection** of an EVP  $\mathbf{Ax} = \lambda\mathbf{x}$  onto a subspace  $V := \text{Span} \{ \mathbf{v}_1, \dots, \mathbf{v}_m \}$ ,  $m < n$   $\rightarrow$  smaller  $m \times m$  generalized EVP

$$\underbrace{\mathbf{V}^T \mathbf{A} \mathbf{V}}_{:= \mathbf{H}} \mathbf{x} = \lambda \mathbf{V}^T \mathbf{V} \mathbf{x} \quad , \quad \mathbf{V} := (\mathbf{v}_1, \dots, \mathbf{v}_m) \in \mathbb{R}^{n,m} . \quad (6.4.1)$$

From Rayleigh quotient Thm. 6.3.37 and considerations in Sect. 6.3.4.2:

$$\begin{aligned} \mathbf{u}_n \in V &\Rightarrow \text{largest eigenvalue of (6.4.1)} = \lambda_{\max}(\mathbf{A}) , \\ \mathbf{u}_1 \in V &\Rightarrow \text{smallest eigenvalue of (6.4.1)} = \lambda_{\min}(\mathbf{A}) . \end{aligned}$$

Intuition: If  $\mathbf{u}_n$  ( $\mathbf{u}_1$ ) “well captured” by  $V$  (that is, the angle between the vector and the space  $V$  is small), then we can expect that the largest (smallest) eigenvalue of (6.4.1) is a good approximation for  $\lambda_{\max}(\mathbf{A})$  ( $\lambda_{\min}(\mathbf{A})$ ), and that, assuming normalization

$$\mathbf{V} \mathbf{w} \approx \mathbf{u}_1 \quad (\text{or } \mathbf{V} \mathbf{w} \approx \mathbf{u}_n) ,$$

where  $\mathbf{w}$  is the corresponding eigenvector of (6.4.1).

For direct power method (6.3.11):

$$\mathbf{z}^{(k)} \parallel \mathbf{A}^k \mathbf{z}^{(0)}$$

$$V = \text{Span} \left\{ \mathbf{z}^{(0)}, \mathbf{A}\mathbf{z}^{(0)}, \dots, \mathbf{A}^{(k)}\mathbf{z}^{(0)} \right\} = \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{z}^{(0)}) \text{ a Krylov space, } \rightarrow \text{Def. 5.2.6 . (6.4.2)}$$

Code 6.4.3: Ritz projections onto Krylov space (6.4.2)

```

1 function [V,D] = kryleig(A,m)
2 % Ritz projection onto Krylov subspace. An
   orthonormal basis of  $\mathcal{K}_m(\mathbf{A}, \mathbf{1})$  is assembled into
   the columns of  $\mathbf{V}$ .
3 n = size (A,1); V = (1:n)'; V = V/norm (V);
4 for l=1:m-1
5     V = [V,A*V(:,end)] ; [Q,R] = qr (V,0) ;
6     [W,D] = eig (Q'*A*Q) ; V = Q*W;
7 end

```

◁ direct power method with Ritz projection onto Krylov space from (6.4.2), cf. Alg 6.3.101.

Note: implementation for demonstration purposes only (inefficient for sparse matrix **A**!)

*Example 6.4.4 (Ritz projections onto Krylov space).*

```

1 n=100;
2 M=gallery ('tridiag', -0.5*ones(n-1,1), 2*ones(n,1), -1.5*ones(n-1,1)) ;
3 [Q,R]=qr (M) ; A=Q'*diag (1:n)*Q; % synthetic matrix,  $\sigma(\mathbf{A}) = \{1, 2, 3, \dots, 100\}$ 

```

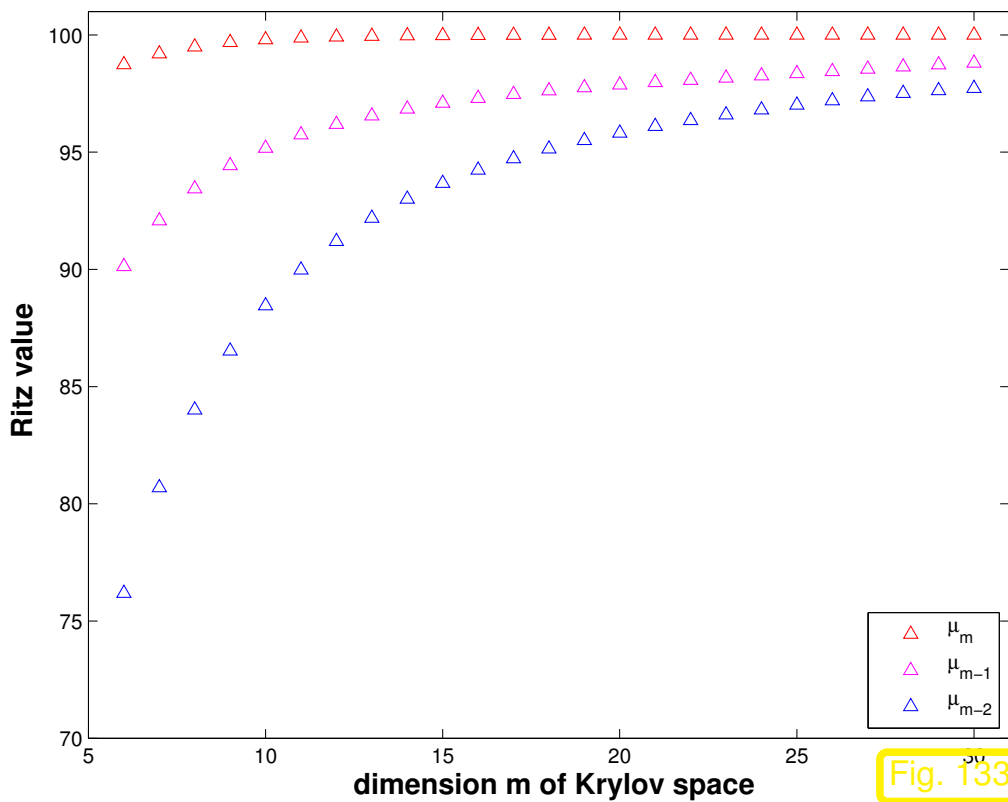


Fig. 133

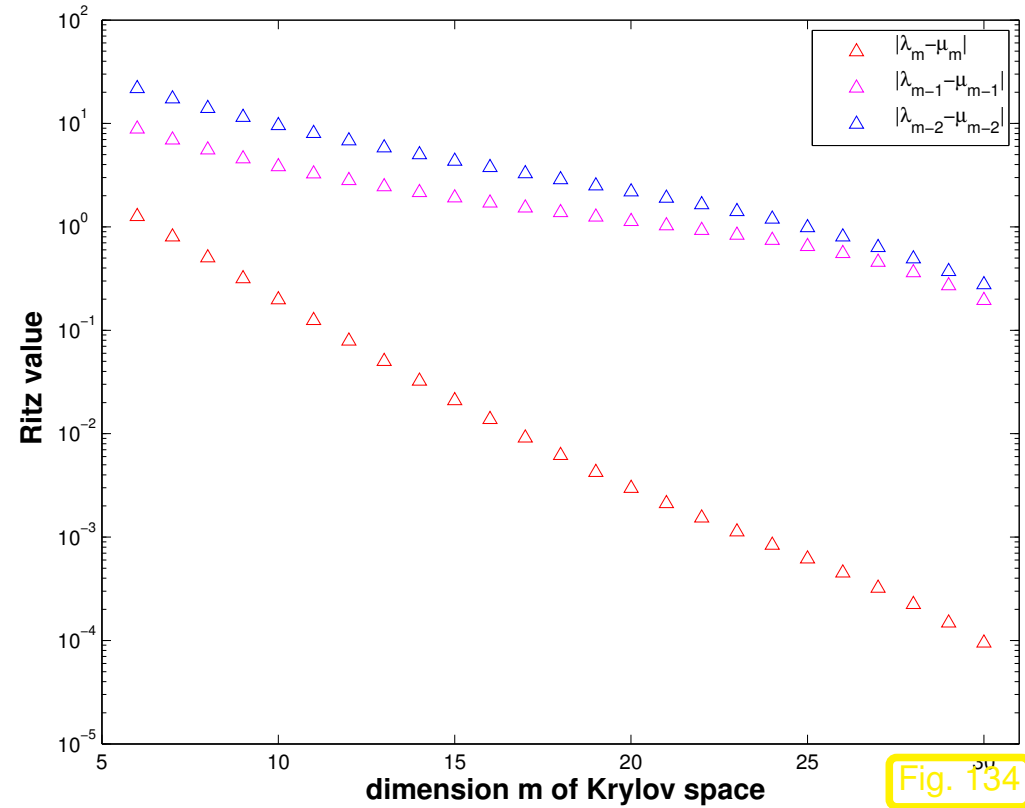
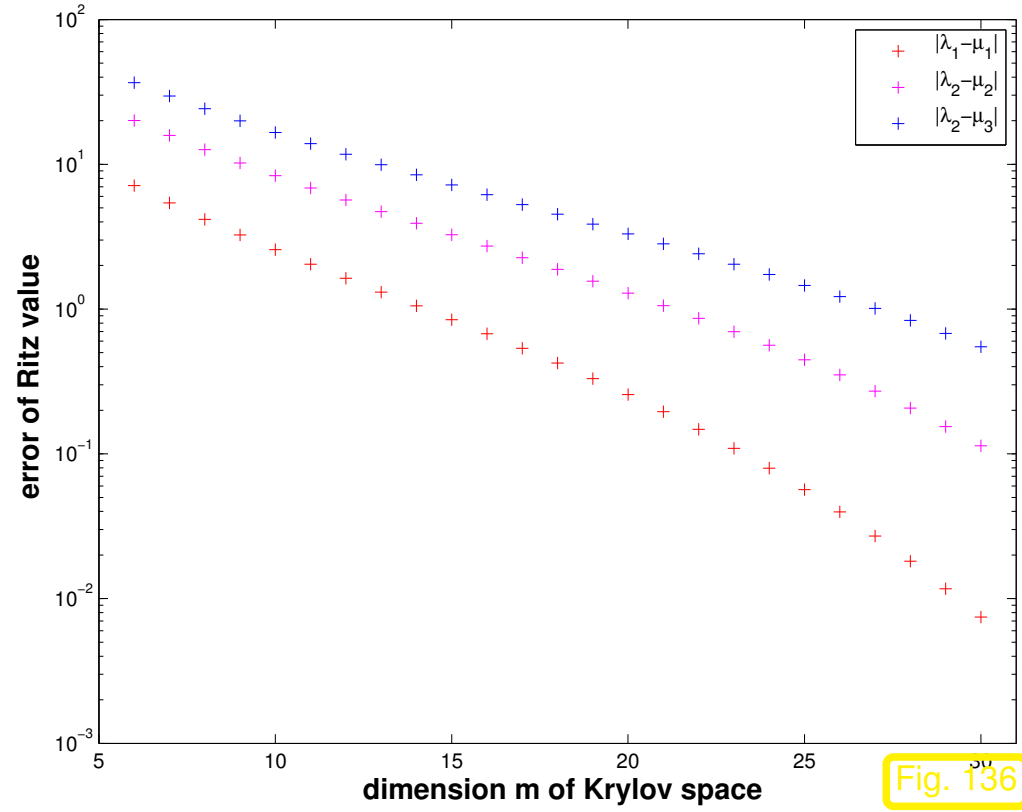
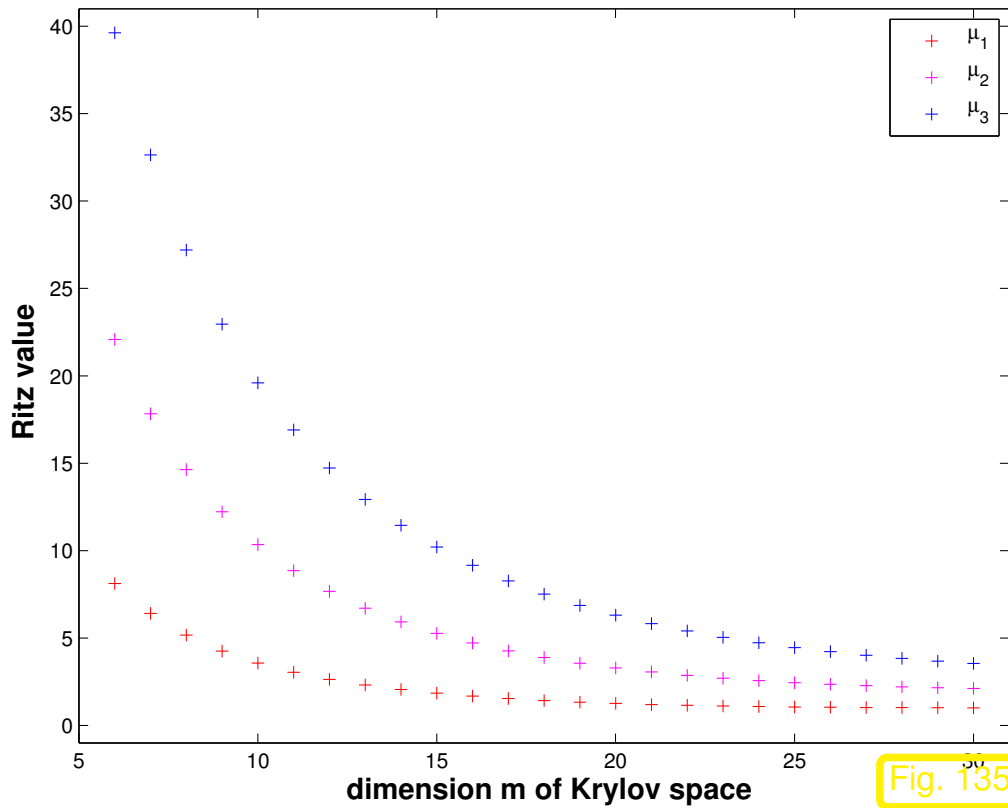


Fig. 134

Observation: “vaguely linear” convergence of largest Ritz values (notation  $\mu_i$ ) to largest eigenvalues.  
Fastest convergence of largest Ritz value  $\rightarrow$  largest eigenvalue of  $A$





Observation: *Also the smallest Ritz values converge “vaguely linearly” to the smallest eigenvalues of  $A$ .* Fastest convergence of smallest Ritz value  $\rightarrow$  smallest eigenvalue of  $A$ .



# Why do smallest Ritz values converge to smallest eigenvalues of $\mathbf{A}$ ?

Consider direct power method (6.3.11) for  $\tilde{\mathbf{A}} := \nu\mathbf{I} - \mathbf{A}$ ,  $\nu > \lambda_{\max}(\mathbf{A})$ :

$$\mathbf{z}^{(0)} \text{ arbitrary, } \tilde{\mathbf{z}}^{(k+1)} = \frac{(\nu\mathbf{I} - \mathbf{A})\tilde{\mathbf{z}}^{(k)}}{\left\|(\nu\mathbf{I} - \mathbf{A})\tilde{\mathbf{z}}^{(k)}\right\|_2} \quad (6.4.5)$$

As  $\sigma(\nu\mathbf{I} - \mathbf{A}) = \nu - \sigma(\mathbf{A})$  and eigenspaces agree, we infer from Thm. 6.3.19

$$\lambda_1 < \lambda_2 \Rightarrow \mathbf{z}^{(k)} \xrightarrow{k \rightarrow \infty} \mathbf{u}_1 \quad \& \quad \rho_{\mathbf{A}}(\mathbf{z}^{(k)}) \xrightarrow{k \rightarrow \infty} \lambda_1 \text{ linearly.} \quad (6.4.6)$$

By the binomial theorem (also applies to matrices, if they commute)

$$(\nu\mathbf{I} - \mathbf{A})^k = \sum_{j=0}^k \binom{k}{j} \nu^{k-j} \mathbf{A}^j \Rightarrow (\nu\mathbf{I} - \mathbf{A})^k \tilde{\mathbf{z}}^{(0)} \in \mathcal{K}_k(\mathbf{A}, \mathbf{z}^{(0)}),$$

$$\mathcal{K}_k(\nu\mathbf{I} - \mathbf{A}, \mathbf{x}) = \mathcal{K}_k(\mathbf{A}, \mathbf{x}). \quad (6.4.7)$$

➤  $\mathbf{u}_1$  can also be expected to be “well captured” by  $\mathcal{K}_k(\mathbf{A}, \mathbf{x})$  and the smallest Ritz value should provide a good approximation for  $\lambda_{\min}(\mathbf{A})$ .

Recall from Sect. 5.2.2 , Lemma 5.2.12:

Residuals  $\mathbf{r}_0, \dots, \mathbf{r}_{m-1}$  generated in CG iteration, Alg. 5.2.17 applied to  $\mathbf{Ax} = \mathbf{z}$  with  $\mathbf{x}^{(0)} = 0$ , provide *orthogonal basis* for  $\mathcal{K}_m(\mathbf{A}, \mathbf{z})$  (, if  $\mathbf{r}_k \neq 0$ ).

► Inexpensive Ritz projection of  $\mathbf{Ax} = \lambda \mathbf{x}$  onto  $\mathcal{K}_m(\mathbf{A}, \mathbf{z})$ : orthogonal matrix

$$\mathbf{V}_m^T \mathbf{A} \mathbf{V}_m \mathbf{x} = \lambda \mathbf{x}, \quad \mathbf{V}_m := \left( \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|}, \dots, \frac{\mathbf{r}_{m-1}}{\|\mathbf{r}_{m-1}\|} \right) \in \mathbb{R}^{n,m}. \quad (6.4.8)$$

recall: residuals generated by *short recursions*, see Alg. 5.2.17

**Lemma 6.4.9** (Tridiagonal Ritz projection from CG residuals).

$\mathbf{V}_m^T \mathbf{A} \mathbf{V}_m$  is a *tridiagonal matrix*.

*Proof.* Lemma 5.2.12:  $\{\mathbf{r}_0, \dots, \mathbf{r}_{\ell-1}\}$  is an orthogonal basis of  $\mathcal{K}_\ell(\mathbf{A}, \mathbf{r}_0)$ , if all the residuals are non-zero. As  $\mathbf{A}\mathcal{K}_{\ell-1}(\mathbf{A}, \mathbf{r}_0) \subset \mathcal{K}_\ell(\mathbf{A}, \mathbf{r}_0)$ , we conclude the orthogonality  $\mathbf{r}_m^T \mathbf{A} \mathbf{r}_j$  for all  $j = 0, \dots, m-2$ . Since

$$\left( \mathbf{V}_m^T \mathbf{A} \mathbf{V}_m \right)_{ij} = \mathbf{r}_{i-1}^T \mathbf{A} \mathbf{r}_{j-1}, \quad 1 \leq i, j \leq m,$$



Algorithm for computing  $V_l$  and  $T_l$ :

## Lanczos process

Computational effort/step:

1 ×  $A \times$  vector

2 dot products

2 AXPY-operations

1 division

Closely related to CG iteration,

Alg. 5.2.17, Code 5.2.17.

```

1 function [V,alph,bet] = lanczos(A,k,z0)
2 % Note: this implementation of the Lanczos process
  also records the orthonormal CG residuals in the
  columns of the matrix V, which is not needed when
  only eigenvalue approximations are desired.
3 V = z0/norm(z0);
4 % Vectors storing entries of tridiagonal matrix (6.4.10)
5 alph=zeros(k,1); bet = zeros(k,1);
6 for j=1:k
7     q = A*V(:,j); alph(j) = dot(q,V(:,j));
8     w = q - alph(j)*V(:,j);
9     if (j > 1), w = w - bet(j-1)*V(:,j-1); end
10    bet(j) = norm(w); V = [V,w/bet(j)];
11 end
12 bet = bet(1:end-1);

```

Total computational effort for  $l$  steps of Lanczos process, if  $A$  has at most  $k$  non-zero entries per row: $O(nkl)$ 

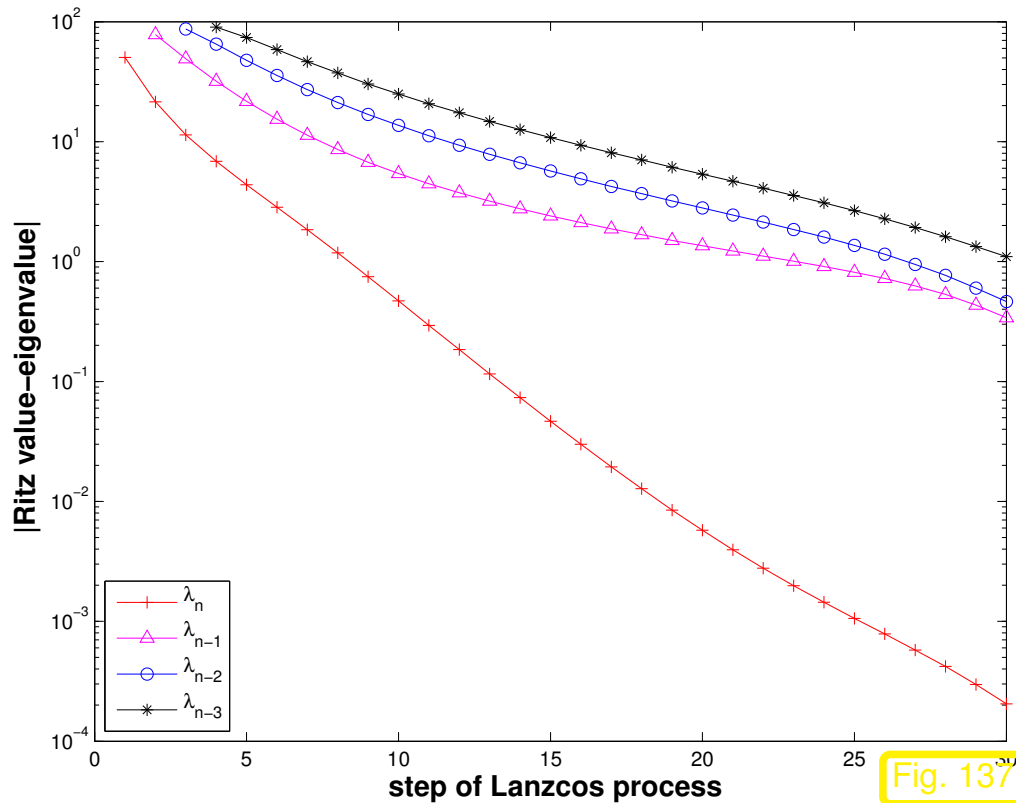
Note: Code 6.4.10 assumes that no residual vanishes. This could happen, if  $z_0$  exactly belonged to the span of a few eigenvectors. However, in practical computations inevitable round-off errors will always ensure that the iterates do not stay in an invariant subspace of  $A$ , cf. Rem. 6.3.20.

Convergence (what we expect from the above considerations)  $\rightarrow$  [17, Sect. 8.5]

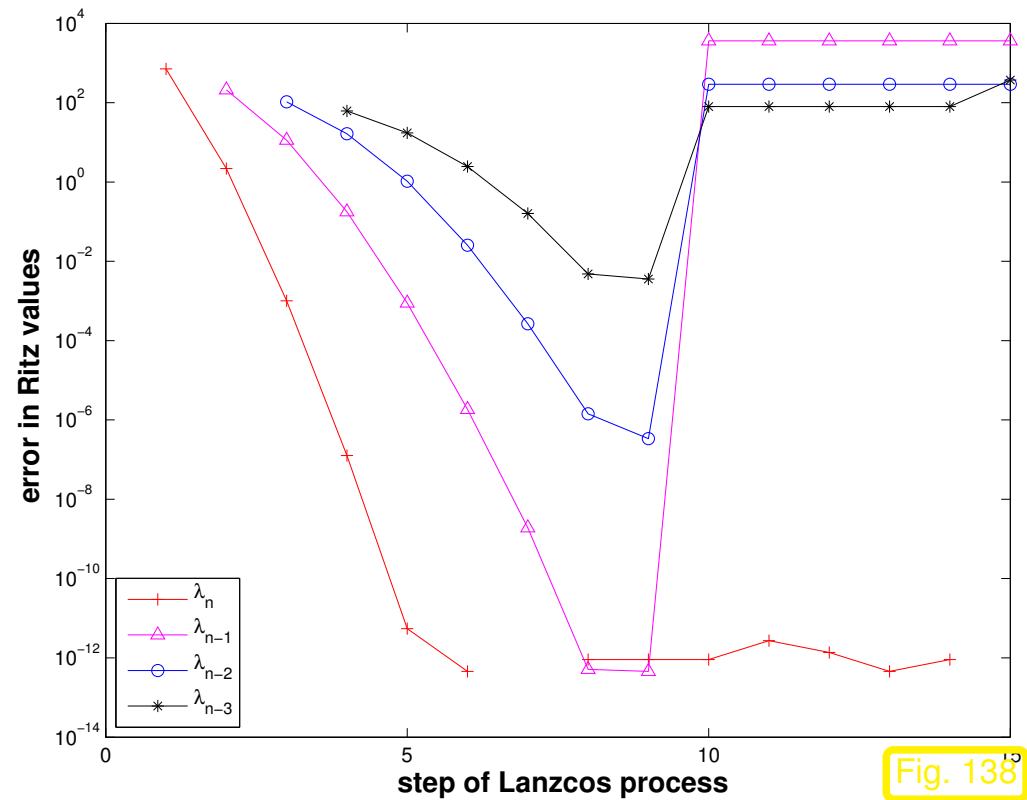
In  $l$ -th step:  $\lambda_n \approx \mu_l^{(l)}, \lambda_{n-1} \approx \mu_{l-1}^{(l)}, \dots, \lambda_1 \approx \mu_1^{(l)}$ ,  
 $\sigma(\mathbf{T}_l) = \{\mu_1^{(l)}, \dots, \mu_l^{(l)}\}, \mu_1^{(l)} \leq \mu_2^{(l)} \leq \dots \leq \mu_l^{(l)}$ .

Example 6.4.12 (Lanczos process for eigenvalue computation).

**A** from Ex. 6.4.4



`A = gallery('minij', 100);`



Observation: same as in Ex. 6.4.4, linear convergence of Ritz values to eigenvalues.

However for  $\mathbf{A} \in \mathbb{R}^{10,10}$ ,  $a_{ij} = \min\{i, j\}$  good initial convergence, but sudden “jump” of Ritz values off eigenvalues!

Conjecture: Impact of roundoff errors, *cf.* Ex. 5.2.21



*Example 6.4.13* (Impact of roundoff on Lanczos process).

$$\mathbf{A} \in \mathbb{R}^{10,10}, \quad a_{ij} = \min\{i, j\}. \quad \text{A} = \text{gallery}('minij', 10);$$





► Uncanny cluster of computed eigenvalues of **T** (“ghost eigenvalues”, [23, Sect. 9.2.5])

$$V^H V = \begin{pmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000251 & 0.258801 & 0.883711 \\ 0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000106 & 0.109470 & 0.373799 \\ 0.000000 & -0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000005 & 0.005373 & 0.018347 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000096 & 0.000328 \\ 0.000000 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000001 & 0.000003 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 \\ 0.000251 & 0.000106 & 0.000005 & 0.000000 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 \\ 0.258801 & 0.109470 & 0.005373 & 0.000096 & 0.000001 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & 0.000000 \\ 0.883711 & 0.373799 & 0.018347 & 0.000328 & 0.000003 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{pmatrix}$$

► Loss of orthogonality of residual vectors due to roundoff  
(compare: impact of roundoff on CG iteration, Ex. 5.2.21)

$l$	$\sigma(\mathbf{T}_l)$									
1										38.500000
2								3.392123	44.750734	
3							1.117692	4.979881	44.766064	
4						0.597664	1.788008	5.048259	44.766069	
5					0.415715	0.925441	1.870175	5.048916	44.766069	
6				0.336507	0.588906	0.995299	1.872997	5.048917	44.766069	
7			0.297303	0.431779	0.638542	0.999922	1.873023	5.048917	44.766069	
8		0.276160	0.349724	0.462449	0.643016	1.000000	1.873023	5.048917	44.766069	
9		0.276035	0.349451	0.462320	0.643006	1.000000	1.873023	3.821426	5.048917	44.766069
10	0.263867	0.303001	0.365376	0.465199	0.643104	1.000000	1.873023	5.048917	44.765976	44.766069



Idea: • do not rely on orthogonality relations of Lemma 5.2.12

- use explicit **Gram-Schmidt orthogonalization** [48, Thm. 4.8], [27, Alg .6.1]

Details: inductive approach: given  $\{\mathbf{v}_1, \dots, \mathbf{v}_l\}$  ONB of  $\mathcal{K}_l(\mathbf{A}, \mathbf{z})$

$$\blacktriangleright \quad \tilde{\mathbf{v}}_{l+1} := \mathbf{A}\mathbf{v}_l - \sum_{j=1}^l (\mathbf{v}_j^H \mathbf{A}\mathbf{v}_l) \mathbf{v}_j, \quad \mathbf{v}_{l+1} := \frac{\tilde{\mathbf{v}}_{l+1}}{\|\tilde{\mathbf{v}}_{l+1}\|_2} \Rightarrow \mathbf{v}_{l+1} \perp \mathcal{K}_l(\mathbf{A}, \mathbf{z}). \quad (6.4.14)$$

orthogonal

(Gram-Schmidt, cf. (5.2.11))

**Arnoldi process:** In step  $l$ :

- $1 \times \mathbf{A} \times \text{vector}$
- $l + 1$  dot products
- $l$  AXPY-operations
- $n$  divisions

➤ Computational cost for  $l$  steps, if at most  $k$  non-zero entries in each row of  $\mathbf{A}$ :  $O(nkl^2)$

### Code 6.4.15: Arnoldi process

```

1 function [V,H] = arnoldi(A,k,v0)
2 % Columns of V store orthonormal basis of Krylov spaces  $\mathcal{K}_l(\mathbf{A}, \mathbf{v}_0)$ .
3 % H returns Hessenberg matrix, see Lemma 6.4.16.

```

```

4 V = [v0/norm(v0)];
5 H = zeros(k+1,k);
6 for l=1:k
7     vt = A*V(:,l); % "power iteration", next basis vector
8     for j=1:l
9         % Gram-Schmidt orthogonalization, cf. Sect. 6.3.4.1
10        H(j,l) = dot(V(:,j),vt);
11        vt = vt - H(j,l)*V(:,j);
12    end
13    H(l+1,l) = norm(vt);
14    if (H(l+1,l) == 0), break; end % "theoretical" termination
15    V = [V, vt/H(l+1,l)];
16 end

```

If it does not stop prematurely, the Arnoldi process of Code 6.4.14 will yield an *orthonormal basis* (ONB) of  $\mathcal{K}_{k+1}(\mathbf{A}, \mathbf{v}_0)$  for a **general**  $\mathbf{A} \in \mathbb{C}^{n,n}$ .

Arnoldi process

6.4.14

$$\mathbf{V}_l = [\mathbf{v}_1, \dots, \mathbf{v}_l] : \mathbf{A}\mathbf{V}_l = \mathbf{V}_{l+1}\tilde{\mathbf{H}}_l, \quad \tilde{\mathbf{H}}_l \in \mathbb{K}^{l+1,l} \text{ mit } \tilde{h}_{ij} = \begin{cases} \mathbf{v}_i^H \mathbf{A}\mathbf{v}_j & , \text{ if } i \leq j, \\ \|\tilde{\mathbf{v}}_i\|_2 & , \text{ if } i = j + 1, \\ 0 & \text{else.} \end{cases}$$





*Remark 6.4.17* (Arnoldi process and Ritz projection).

Interpretation of Lemma 6.4.16 (iii) & (i):

$\mathbf{H}_l \mathbf{x} = \lambda \mathbf{x}$  is a (generalized) Ritz projection of EVP  $\mathbf{A} \mathbf{x} = \lambda \mathbf{x}$ , cf. Sect. 6.3.4.2.



► Eigenvalue approximation for general EVP  $\mathbf{A} \mathbf{x} = \lambda \mathbf{x}$  by Arnoldi process:

$$\begin{aligned} \text{In } l\text{-th step: } \lambda_n \approx \mu_l^{(l)}, \lambda_{n-1} \approx \mu_{l-1}^{(l)}, \dots, \lambda_1 \approx \mu_1^{(l)}, \\ \sigma(\mathbf{H}_l) = \{\mu_1^{(l)}, \dots, \mu_l^{(l)}\} \quad (\text{Ritz values}), \quad |\mu_1^{(l)}| \leq |\mu_2^{(l)}| \leq \dots \leq |\mu_l^{(l)}|. \end{aligned}$$

## Code 6.4.18: Arnoldi eigenvalue approximation

```

1 function [dn,V,Ht] = arnoldieig(A,v0,k,tol)
2 n = size(A,1); V = [v0/norm(v0)];
3 H = zeros(1,0); dn = zeros(k,1);
4 for l=1:n
5     d = dn;
6     Ht = [Ht, zeros(1,1); zeros(1,1)];
7     vt = A*V(:,l);
8     for j=1:l
9         Ht(j,l) = dot(V(:,j),vt);
10        vt = vt - Ht(j,l)*V(:,j);
11    end
12    ev = sort(eig(Ht(1:l,1:l)));
13    dn(1:min(l,k)) =
14        ev(end:-1:end-min(l,k)+1);
15    if (norm(d-dn) < tol*norm(dn)), break;
16    end;
17    Ht(l+1,l) = norm(vt);
18    V = [V, vt/Ht(l+1,l)];
19 end

```

Arnoldi process for computing  
the  $k$  largest (in modulus)  
eigenvalues of  $A \in \mathbb{C}^{n,n}$

1  $A \times$  vector per step  
( $\triangleright$  attractive for sparse  
matrices)

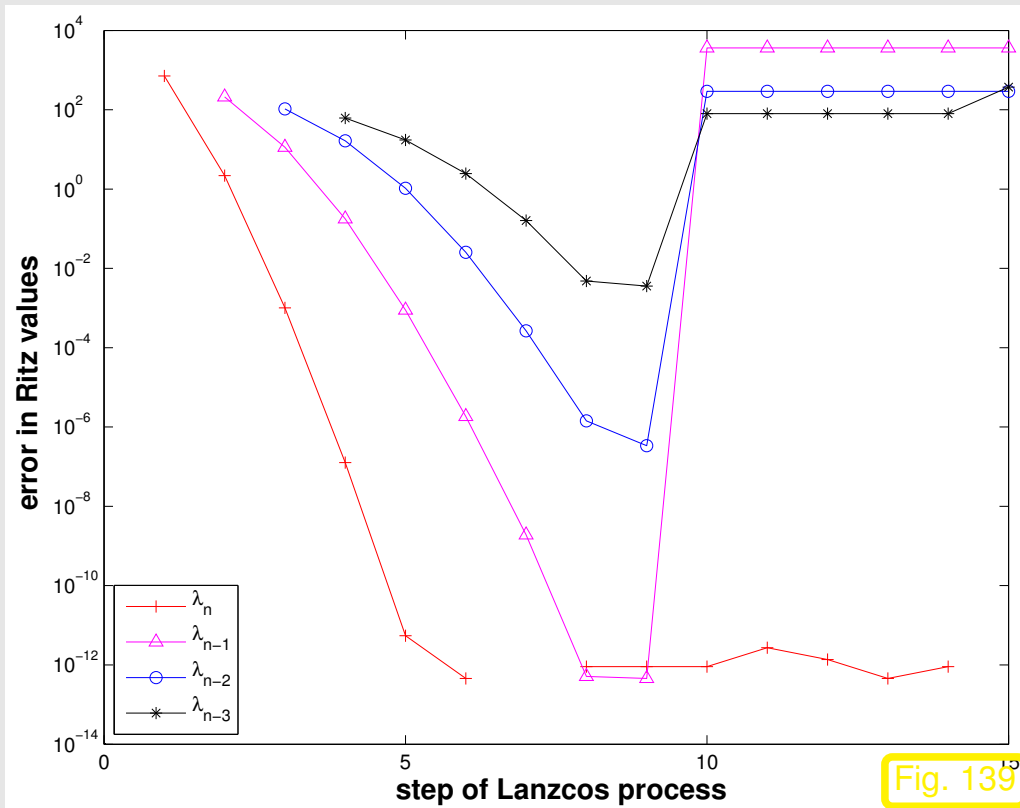
However: required storage in-  
creases with number of steps,  
*cf.* situation with GMRES,  
Sect. 5.4.1.

Heuristic termination criterion

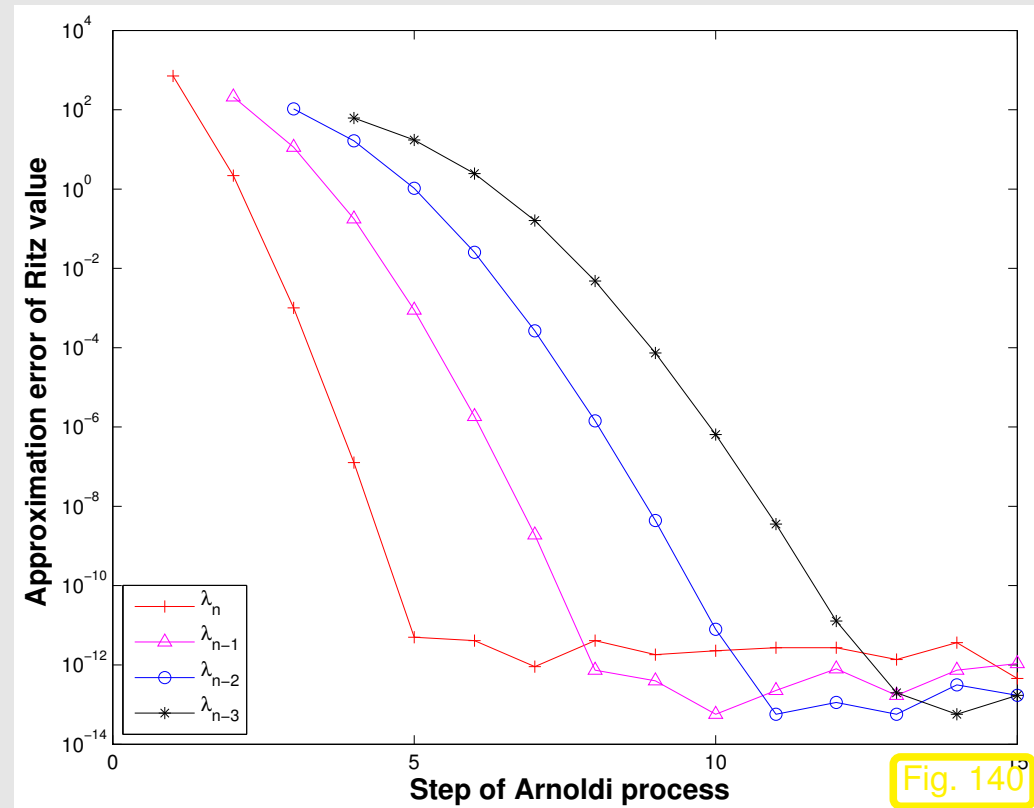
Example 6.4.19 (Stability of Arnoldi process).

$$A \in \mathbb{R}^{100,100}, \quad a_{ij} = \min\{i, j\}.$$

```
A = gallery('minij', 100);
```



Lanczos process: Ritz values



Arnoldi process: Ritz values

Ritz values during Arnoldi process for `A = gallery('minij', 10);` ↔ Ex. 6.4.12



$l$	$\sigma(\mathbf{H}_l)$									
1										38.500000
2									3.392123	44.750734
3								1.117692	4.979881	44.766064
4							0.597664	1.788008	5.048259	44.766069
5						0.415715	0.925441	1.870175	5.048916	44.766069
6					0.336507	0.588906	0.995299	1.872997	5.048917	44.766069
7				0.297303	0.431779	0.638542	0.999922	1.873023	5.048917	44.766069
8			0.276159	0.349722	0.462449	0.643016	1.000000	1.873023	5.048917	44.766069
9		0.263872	0.303009	0.365379	0.465199	0.643104	1.000000	1.873023	5.048917	44.766069
10	0.255680	0.273787	0.307979	0.366209	0.465233	0.643104	1.000000	1.873023	5.048917	44.766069

Observation: (almost perfect approximation of spectrum of  $\mathbf{A}$ )

For the above examples both the Arnoldi process and the Lanczos process are *algebraically equivalent*, because they are applied to a symmetric matrix  $\mathbf{A} = \mathbf{A}^T$ . However, they behave strikingly differently, which indicates that they are *not numerically equivalent*.

The Arnoldi process is much less affected by roundoff than the Lanczos process, because it does not take for granted orthogonality of the “residual vector sequence”. Hence, the Arnoldi process enjoys superior numerical stability ( $\rightarrow$  Sect. 2.5.2, Def. 2.5.11) compared to the Lanczos process.



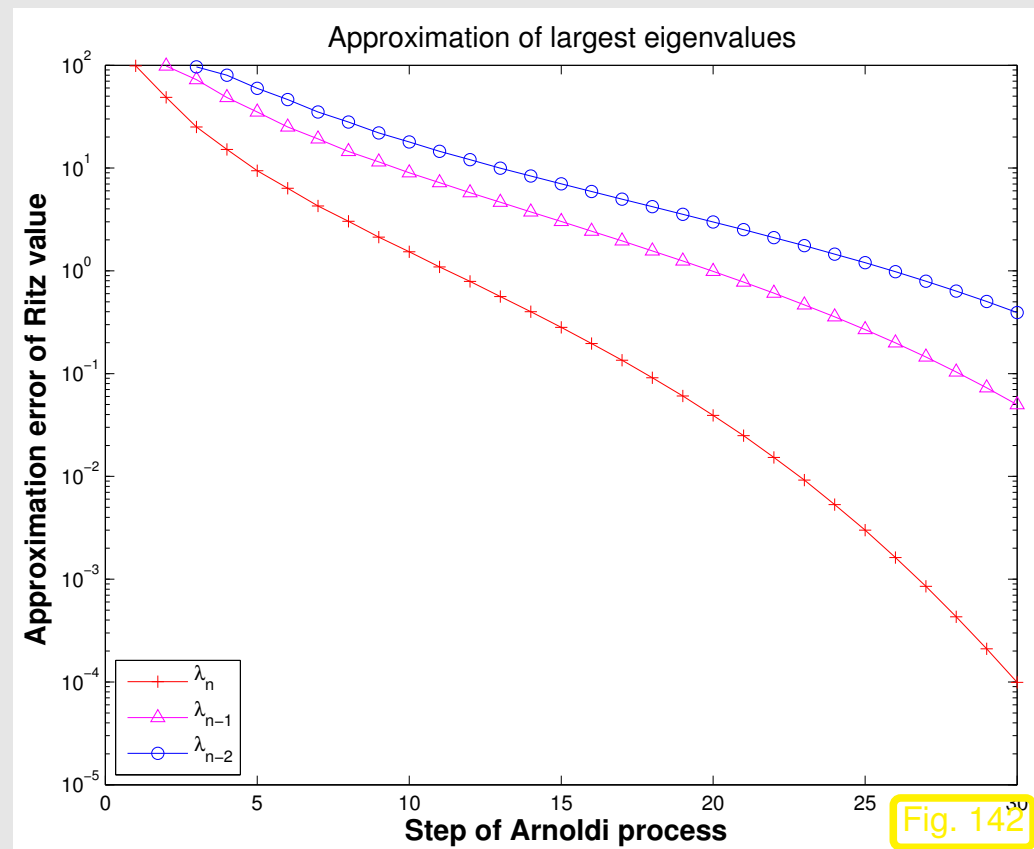
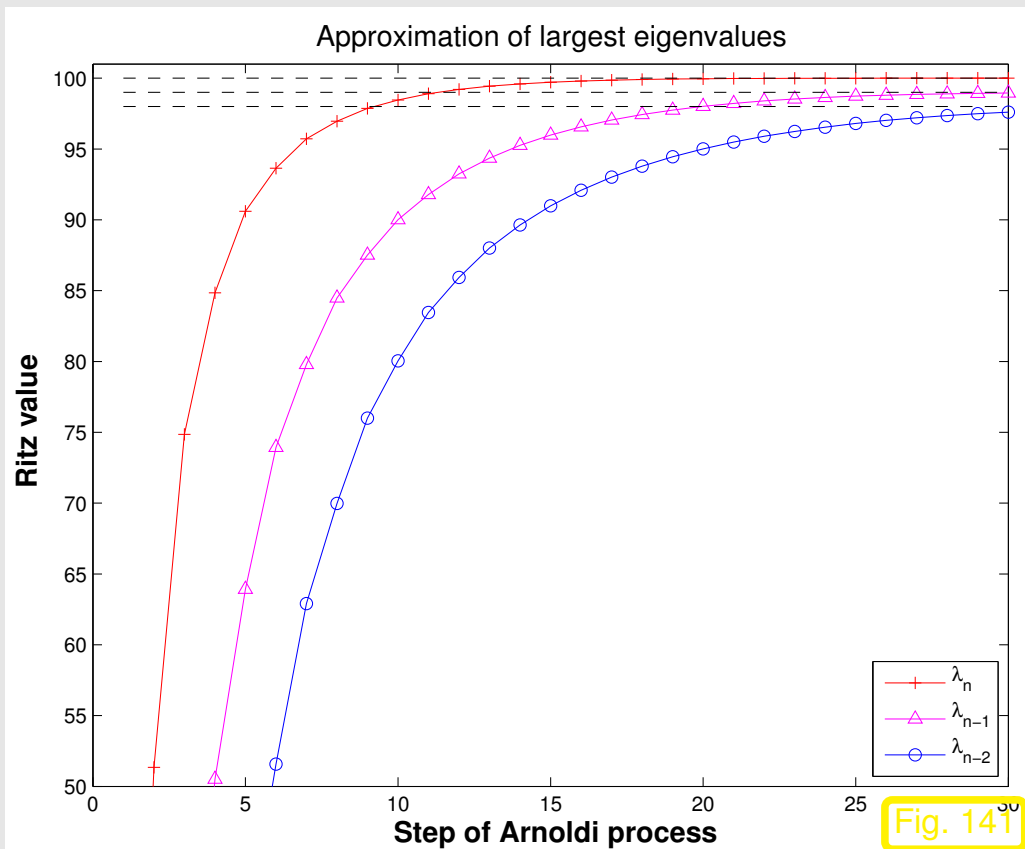
Example 6.4.20 (Eigenvalue computation with Arnoldi process).

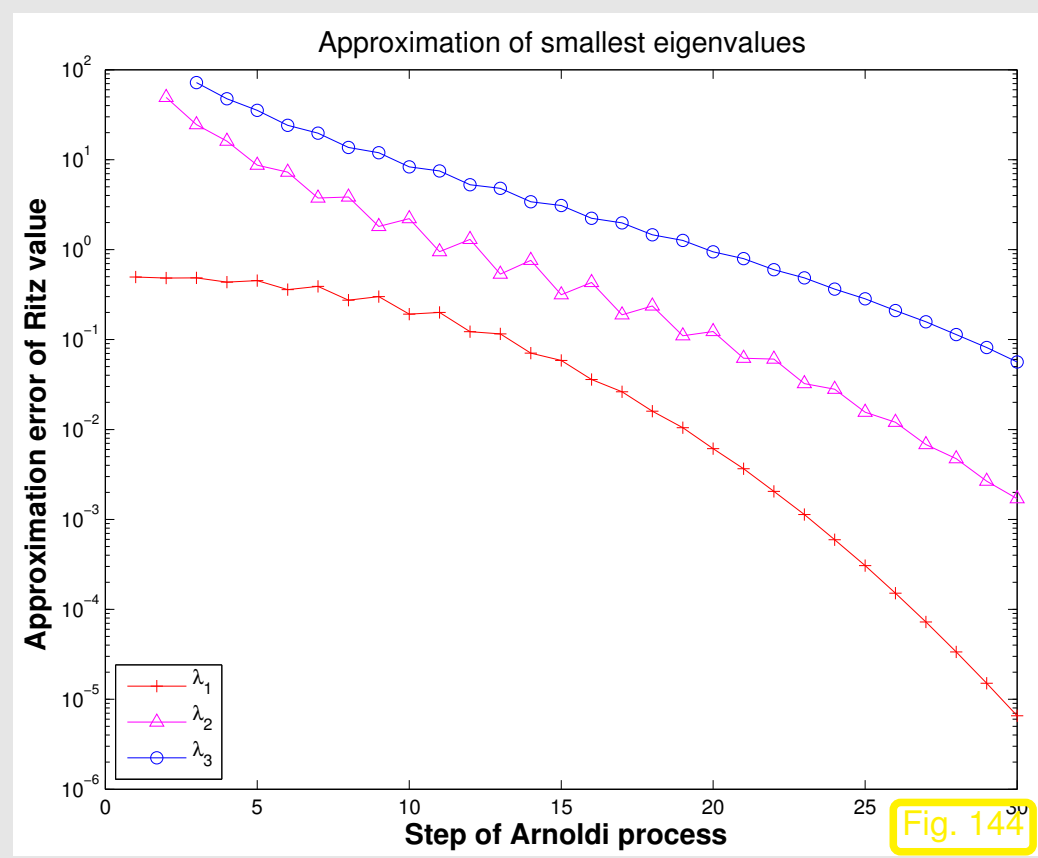
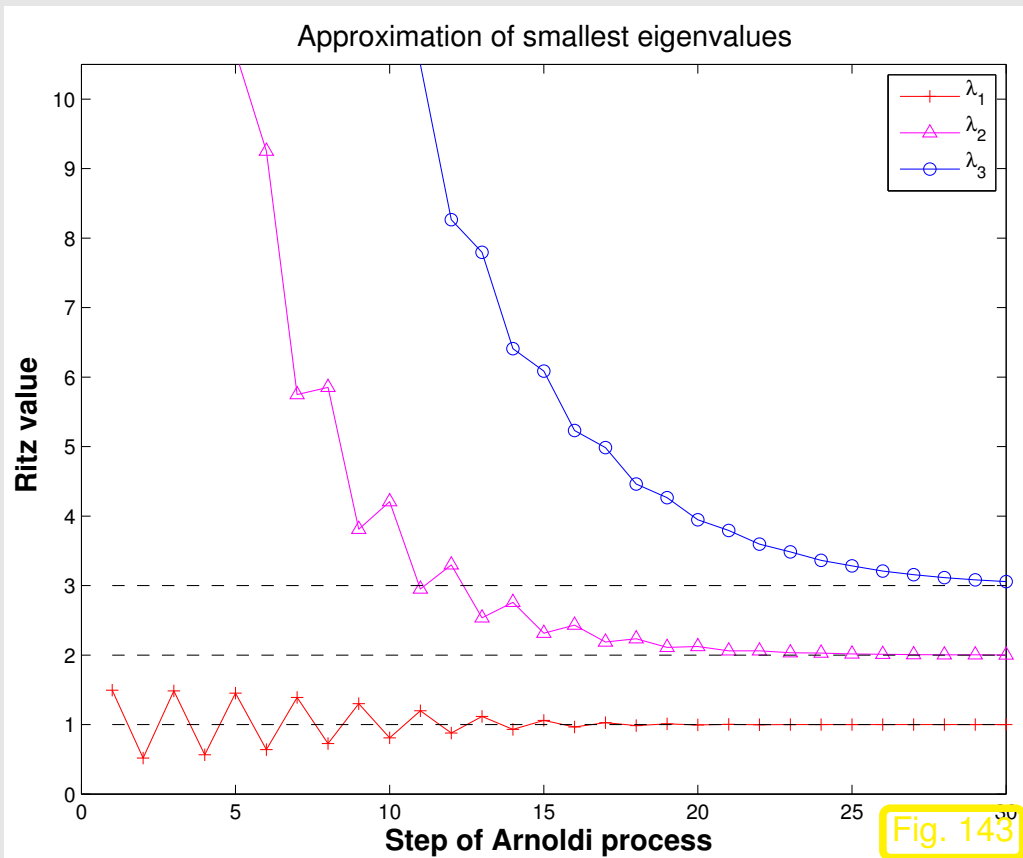
Eigenvalue approximation from Arnoldi process for *non-symmetric*  $A$ , initial vector  $\text{ones}(100, 1)$  ;

```

1 n=100;
2 M=full(gallery('tridiag', -0.5*ones(n-1, 1), 2*ones(n, 1), -1.5*ones(n-1, 1)));
3 A=M*diag(1:n)*inv(M);

```





Observation: “vaguely linear” convergence of largest and smallest eigenvalues, *cf.* Ex. 6.4.4.



Krylov subspace iteration methods (= Arnoldi process, Lanczos process) attractive for computing *a few* of the largest/smallest eigenvalues and associated eigenvectors of *large sparse matrices*.

*Remark 6.4.21* (Krylov subspace methods for generalized EVP).

Adaptation of Krylov subspace iterative eigensolvers to generalized EVP:  $\mathbf{Ax} = \lambda\mathbf{Bx}$ ,  $\mathbf{B}$  s.p.d.:  
replace Euclidean inner product with “**B-inner product**”  $(\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x}^H \mathbf{B} \mathbf{y}$ .

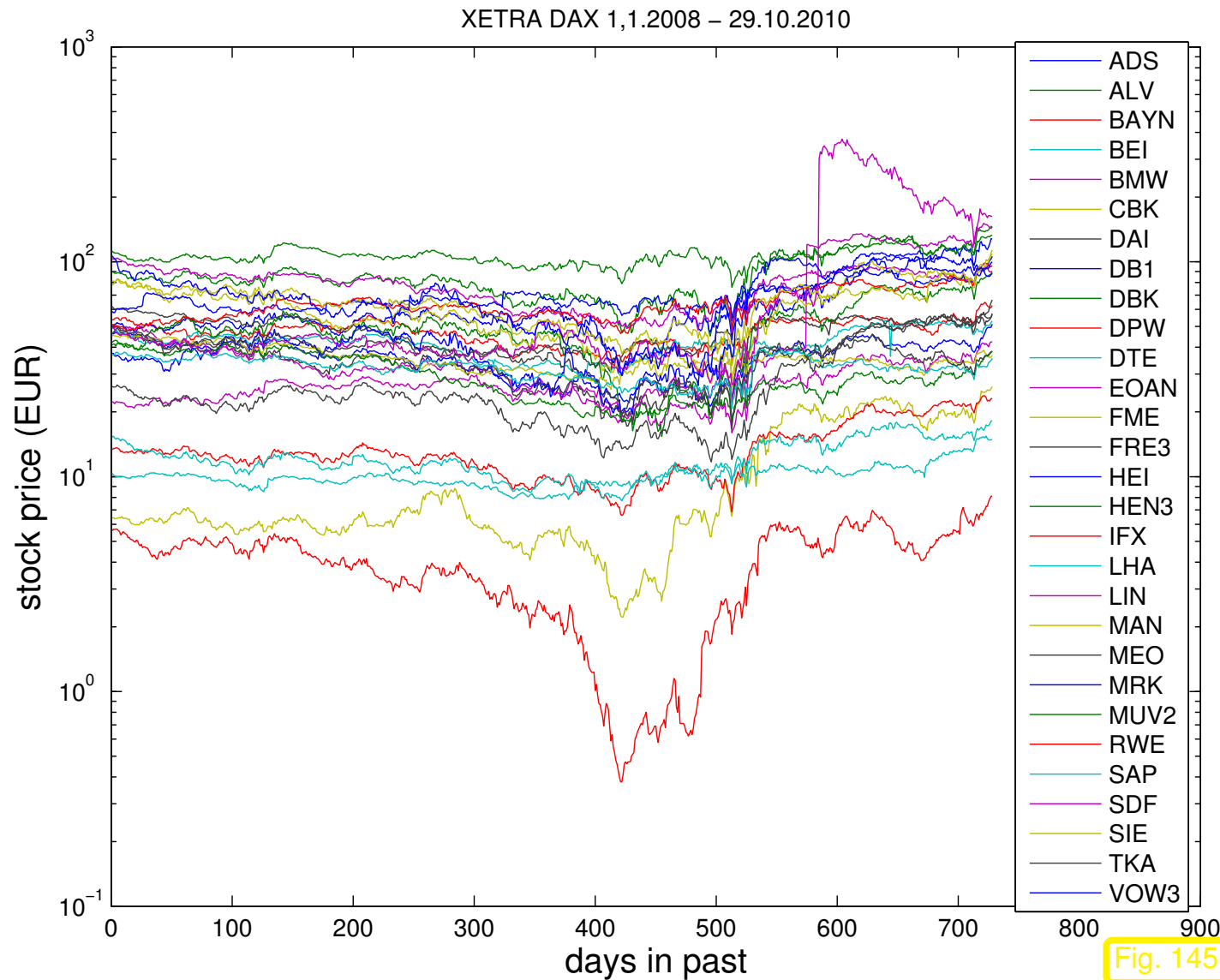


MATLAB-functions:

`d = eigs(A, k, sigma)` :  $k$  largest/smallest eigenvalues of  $\mathbf{A}$   
`d = eigs(A, B, k, sigma)` :  $k$  largest/smallest eigenvalues for generalized EVP  $\mathbf{Ax} = \lambda\mathbf{Bx}$ ,  $\mathbf{B}$  s.p.d.  
`d = eigs(Afun, n, k)` : `Afun` = handle to function providing matrix×vector for  $\mathbf{A}/\mathbf{A}^{-1}/\mathbf{A} - \alpha\mathbf{I}/(\mathbf{A} - \alpha\mathbf{B})^{-1}$ . (Use flags to tell `eigs` about special properties of matrix behind `Afun`.)

# 6.5 Singular Value Decomposition

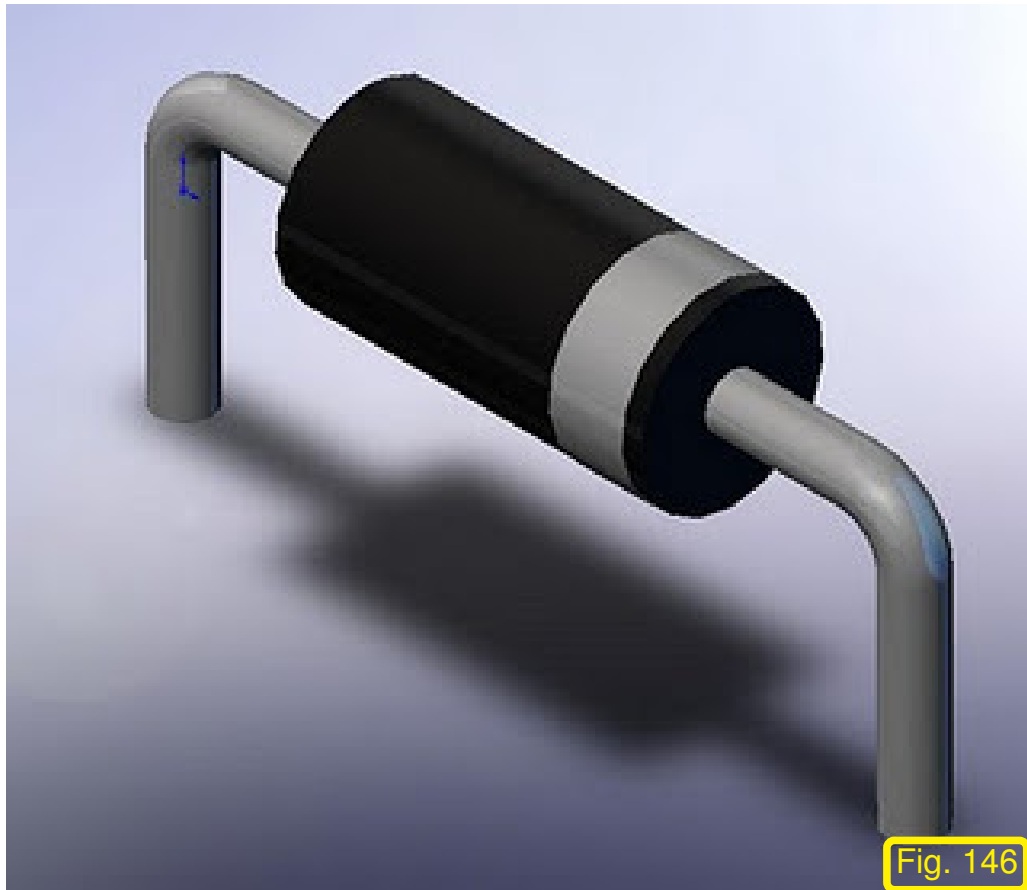
Example 6.5.1 (Trend analysis).



◁ (end of day) stock prizes

Are there underlying governing trends ?



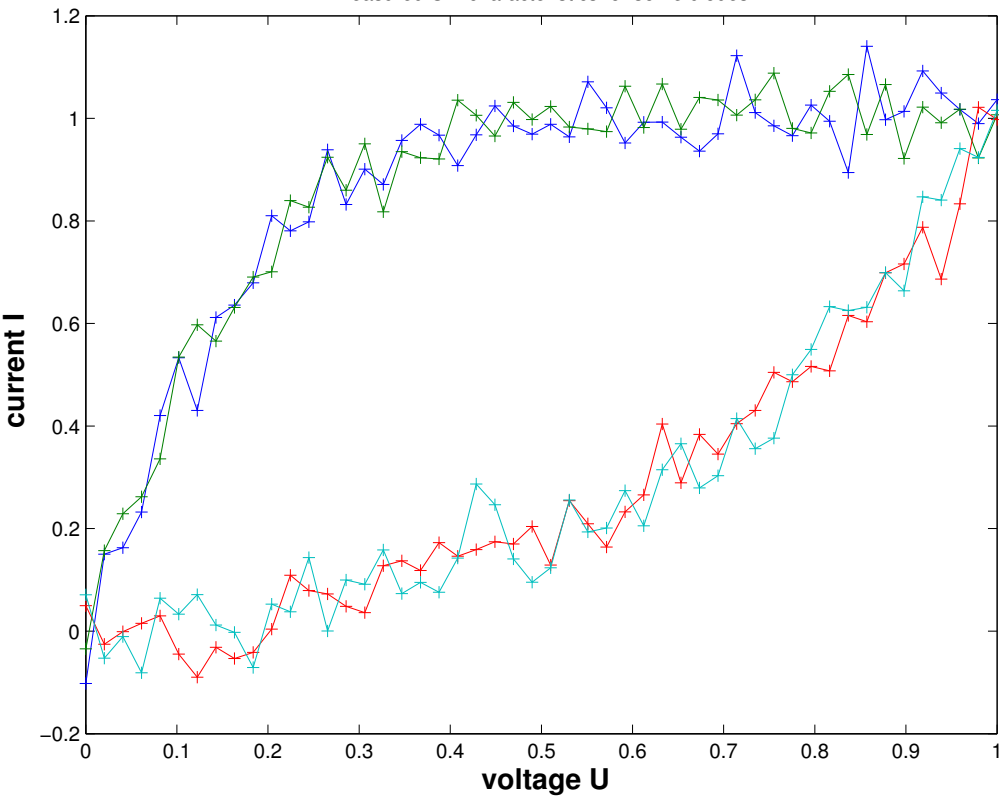


Given: measured  $U$ - $I$  characteristics of diodes

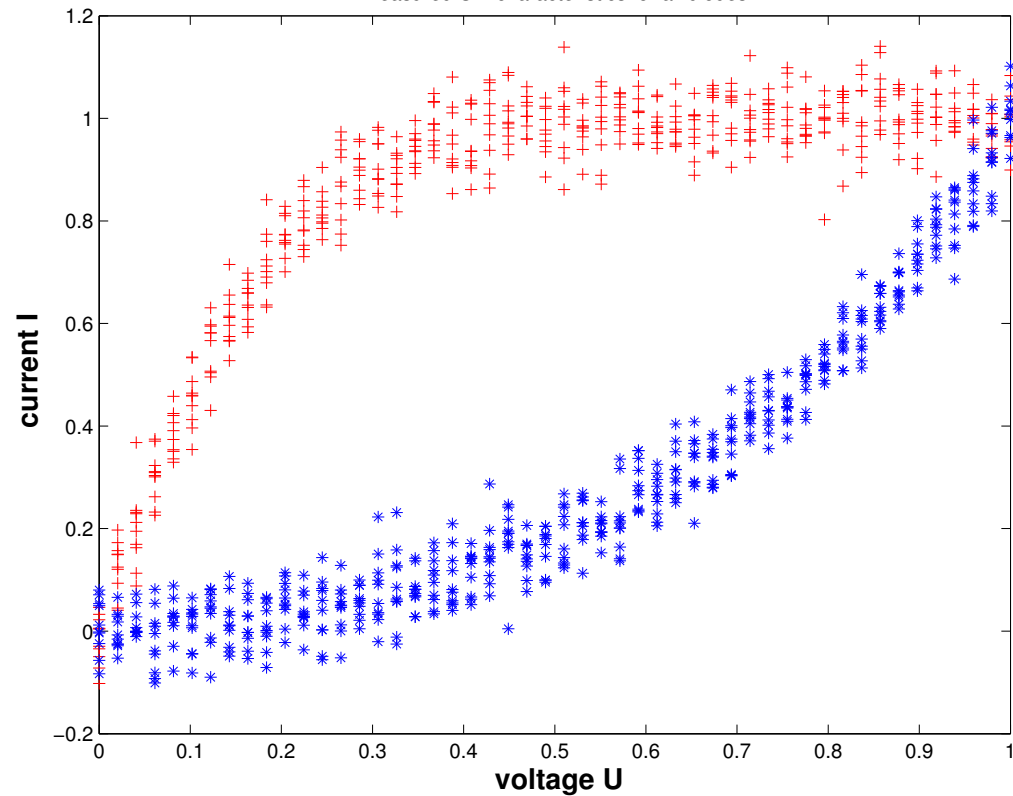
Find out

- how many types,
- the  $U$ - $I$  characteristic of each type.

measured U-I characteristics for some diodes



measured U-I characteristics for all diodes



**Remark 6.5.3 (Principal component analysis (PCA)).**

Given:  $n$  data points  $\mathbf{a}_j \in \mathbb{R}^m$ ,  $j = 1, \dots, n$ , in  $m$ -dimensional (feature) space  
(e.g.,  $\mathbf{a}_j$  may represent a finite *time series* or a *measured relationship* of physical quantities)

In Ex. 6.5.1:  $n \hat{=}$  number of stocks

$m \hat{=}$  number of days, for which stock prices are recorded

- Extreme case: all stocks follow exactly *one* trend

$$\Leftrightarrow \mathbf{a}_j \in \text{Span} \{ \mathbf{u} \} \quad \forall j = 1, \dots, n,$$

for a **trend vector**  $\mathbf{u} \in \mathbb{R}^m$ ,  $\|\mathbf{u}\|_2 = 1$ .

- Unlikely case: all stocks prices are governed by  $p < n$  trends:

$$\Leftrightarrow \mathbf{a}_j \in \text{Span} \{ \mathbf{u}_1, \dots, \mathbf{u}_p \} \quad \forall j = 1, \dots, m, \quad (6.5.4)$$

with **orthonormal trend vectors**  $\mathbf{u}_i \in \mathbb{R}^m$ ,  $i = 1, \dots, p$ .

Why orthonormal ? Trends should be as “independent as possible” (minimally correlated)

Perspective of linear algebra:

$$(6.5.4) \Leftrightarrow \text{rank}(\mathbf{A}) = p \text{ for } \mathbf{A} := (\mathbf{a}_1, \dots, \mathbf{a}_n) \in \mathbb{R}^{m,n}, \quad \text{Im}(\mathbf{A}) = \text{Span} \{ \mathbf{u}_1, \dots, \mathbf{u}_p \} \quad (6.5.5)$$

- Realistic: stock prizes *approximately* follow a few trends

$$\mathbf{a}_j \in \text{Span} \{ \mathbf{u}_1, \dots, \mathbf{u}_p \} + \text{“small perturbations”} \quad \forall j = 1, \dots, m,$$

with orthonormal trend vectors  $\mathbf{u}_i$ ,  $i = 1, \dots, p$ .



Task (PCA): determine (minimal)  $p$  and orthonormal trend vectors  $\mathbf{u}_i, i = 1, \dots, p$

Issue: how to distinguish between trends and perturbations ?

**Theorem 6.5.6** (singular value decomposition).  $\rightarrow$  [48, Thm. 9.6], [27, Thm. 11.1]

For any  $\mathbf{A} \in \mathbb{K}^{m,n}$  there are unitary matrices  $\mathbf{U} \in \mathbb{K}^{m,m}$ ,  $\mathbf{V} \in \mathbb{K}^{n,n}$  and a (generalized) diagonal <sup>(\*)</sup> matrix  $\mathbf{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m,n}$ ,  $p := \min\{m, n\}$ ,  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$  such that

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H.$$

(\*) :  $\mathbf{\Sigma}$  (generalized) diagonal matrix  $:\Leftrightarrow (\mathbf{\Sigma})_{i,j} = 0$ , if  $i \neq j, 1 \leq i \leq m, 1 \leq j \leq n$ .

$$\begin{pmatrix} \boxed{\text{A}} \end{pmatrix} = \begin{pmatrix} \boxed{\text{U}} \end{pmatrix} \begin{pmatrix} \boxed{\Sigma} \end{pmatrix} \begin{pmatrix} \boxed{\text{V}^H} \end{pmatrix}$$

The diagram illustrates the singular value decomposition of a matrix A. Matrix A is shown as a tall rectangle. It is equal to the product of three matrices: U (a tall rectangle), Σ (a tall rectangle with a diagonal of yellow squares), and V<sup>H</sup> (a wide rectangle).

$$\begin{pmatrix} \boxed{\text{A}} \end{pmatrix} = \begin{pmatrix} \boxed{\text{U}} \end{pmatrix} \begin{pmatrix} \boxed{\Sigma} \end{pmatrix} \begin{pmatrix} \boxed{\text{V}^H} \end{pmatrix}$$

The diagram illustrates the compact singular value decomposition of a matrix A. Matrix A is shown as a wide rectangle. It is equal to the product of three matrices: U (a wide rectangle), Σ (a wide rectangle with a diagonal of yellow squares), and V<sup>H</sup> (a wide rectangle).

*Proof.* (of Thm. 6.5.6, by induction)

[63, Thm. 4.2.3]: Continuous functions attain extremal values on compact sets (here the unit ball  $\{\mathbf{x} \in \mathbb{R}^n: \|\mathbf{x}\|_2 \leq 1\}$ )

$$\blacktriangleright \exists \mathbf{x} \in \mathbb{K}^n, \mathbf{y} \in \mathbb{K}^m, \quad \|\mathbf{x}\| = \|\mathbf{y}\|_2 = 1 : \mathbf{A}\mathbf{x} = \sigma\mathbf{y}, \quad \sigma = \|\mathbf{A}\|_2,$$

where we used the definition of the matrix 2-norm, see Def. 2.5.5. By Gram-Schmidt orthogonalization:  $\exists \tilde{\mathbf{V}} \in \mathbb{K}^{n,n-1}, \tilde{\mathbf{U}} \in \mathbb{K}^{m,m-1}$  such that

$$\mathbf{V} = (\mathbf{x} \ \tilde{\mathbf{V}}) \in \mathbb{K}^{n,n}, \quad \mathbf{U} = (\mathbf{y} \ \tilde{\mathbf{U}}) \in \mathbb{K}^{m,m} \quad \text{are unitary.}$$

$$\blacktriangleright \mathbf{U}^H \mathbf{A} \mathbf{V} = (\mathbf{y} \ \tilde{\mathbf{U}})^H \mathbf{A} (\mathbf{x} \ \tilde{\mathbf{V}}) = \left( \begin{array}{c|c} \mathbf{y}^H \mathbf{A} \mathbf{x} & \mathbf{y}^H \mathbf{A} \tilde{\mathbf{V}} \\ \hline \tilde{\mathbf{U}}^H \mathbf{A} \mathbf{x} & \tilde{\mathbf{U}}^H \mathbf{A} \tilde{\mathbf{V}} \end{array} \right) = \left( \begin{array}{c|c} \sigma & \mathbf{w}^H \\ \hline 0 & \mathbf{B} \end{array} \right) =: \mathbf{A}_1.$$

Since

$$\left\| \mathbf{A}_1 \begin{pmatrix} \sigma \\ \mathbf{w} \end{pmatrix} \right\|_2^2 = \left\| \begin{pmatrix} \sigma^2 + \mathbf{w}^H \mathbf{w} \\ \mathbf{B} \mathbf{w} \end{pmatrix} \right\|_2^2 = (\sigma^2 + \mathbf{w}^H \mathbf{w})^2 + \|\mathbf{B} \mathbf{w}\|_2^2 \geq (\sigma^2 + \mathbf{w}^H \mathbf{w})^2,$$

we conclude

$$\|\mathbf{A}_1\|_2^2 = \sup_{0 \neq \mathbf{x} \in \mathbb{K}^n} \frac{\|\mathbf{A}_1 \mathbf{x}\|_2^2}{\|\mathbf{x}\|_2^2} \geq \frac{\|\mathbf{A}_1 \begin{pmatrix} \sigma \\ \mathbf{w} \end{pmatrix}\|_2^2}{\|\begin{pmatrix} \sigma \\ \mathbf{w} \end{pmatrix}\|_2^2} \geq \frac{(\sigma^2 + \mathbf{w}^H \mathbf{w})^2}{\sigma^2 + \mathbf{w}^H \mathbf{w}} = \sigma^2 + \mathbf{w}^H \mathbf{w}. \quad (6.5.7)$$

$$\sigma^2 = \|\mathbf{A}\|_2^2 = \left\| \mathbf{U}^H \mathbf{A} \mathbf{V} \right\|_2^2 = \|\mathbf{A}_1\|_2^2 \stackrel{(6.5.7)}{\implies} \|\mathbf{A}_1\|_2^2 = \|\mathbf{A}_1\|_2^2 + \|\mathbf{w}\|_2^2 \implies \mathbf{w} = \mathbf{0} .$$

$$\blacktriangleright \mathbf{A}_1 = \left( \begin{array}{c|c} \sigma & 0 \\ \hline 0 & \mathbf{B} \end{array} \right) .$$

Then apply induction argument to  $\mathbf{B}$

□.

**Definition 6.5.8** (Singular value decomposition (SVD)).

The decomposition  $\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^H$  of Thm. 6.5.6 is called *singular value decomposition (SVD)* of  $\mathbf{A}$ . The diagonal entries  $\sigma_i$  of  $\mathbf{\Sigma}$  are the *singular values* of  $\mathbf{A}$ .

**Lemma 6.5.9.** The squares  $\sigma_i^2$  of the non-zero singular values of  $\mathbf{A}$  are the non-zero eigenvalues of  $\mathbf{A}^H \mathbf{A}$ ,  $\mathbf{A} \mathbf{A}^H$  with associated eigenvectors  $(\mathbf{V})_{:,1}, \dots, (\mathbf{V})_{:,p}$ ,  $(\mathbf{U})_{:,1}, \dots, (\mathbf{U})_{:,p}$ , respectively.

*Proof.*  $\mathbf{A}\mathbf{A}^H$  and  $\mathbf{A}^H\mathbf{A}$  are similar ( $\rightarrow$  Lemma 6.1.6) to diagonal matrices with non-zero diagonal entries  $\sigma_i^2$  ( $\sigma_i \neq 0$ ), e.g.,

$$\mathbf{A}\mathbf{A}^H = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H\mathbf{V}\mathbf{\Sigma}^H\mathbf{U}^H = \mathbf{U} \underbrace{\mathbf{\Sigma}\mathbf{\Sigma}^H}_{\text{diagonal matrix}} \mathbf{U}^H. \quad \square$$

*Remark 6.5.10* (SVD and additive rank-1 decomposition).  $\rightarrow$  [27, Cor. 11.2], [48, Thm. 9.8]

Recall from linear algebra: rank-1 matrices are tensor products of vectors

$$\mathbf{A} \in \mathbb{K}^{m,n} \quad \text{and} \quad \text{rank}(\mathbf{A}) = 1 \quad \Leftrightarrow \quad \exists \mathbf{u} \in \mathbb{K}^m, \mathbf{v} \in \mathbb{K}^n: \quad \mathbf{A} = \mathbf{u}\mathbf{v}^H, \quad (6.5.11)$$

because  $\text{rank}(\mathbf{A}) = 1$  means that  $\mathbf{A}\mathbf{x} = \mu(\mathbf{x})\mathbf{u}$  for some  $\mathbf{u} \in \mathbb{K}^m$  and linear form  $\mathbf{x} \mapsto \mu(\mathbf{x})$ . By the Riesz representation theorem the latter can be written as  $\mu(\mathbf{x}) = \mathbf{v}^H\mathbf{x}$ .

► Singular value decomposition provides additive decomposition into rank-1 matrices:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H = \sum_{j=1}^p \sigma_j (\mathbf{U})_{:,j} (\mathbf{V})_{:,j}^H. \quad (6.5.12)$$

*Remark 6.5.13 (Uniqueness of SVD).*

SVD of Def. 6.5.8 is not (necessarily) unique, but the singular values are.

Proof by contradiction: assume that  $\mathbf{A}$  has two singular value decompositions

$$\mathbf{A} = \mathbf{U}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^H = \mathbf{U}_2 \mathbf{\Sigma}_2 \mathbf{V}_2^H \Rightarrow \mathbf{U}_1 \underbrace{\mathbf{\Sigma}_1 \mathbf{\Sigma}_1^H}_{=\text{diag}(s_1^1, \dots, s_m^1)} \mathbf{U}_1^H = \mathbf{A} \mathbf{A}^H = \mathbf{U}_2 \underbrace{\mathbf{\Sigma}_2 \mathbf{\Sigma}_2^H}_{=\text{diag}(s_1^2, \dots, s_m^2)} \mathbf{U}_2^H.$$

Two similar diagonal matrices with non-increasing diagonal entries are equal !

□



R. Hiptmair

rev 38286,  
November  
29, 2011

MATLAB-functions (for algorithms see [23, Sect. 8.3]):

- $\mathbf{s} = \text{svd}(\mathbf{A})$  : computes singular values of matrix  $\mathbf{A}$
- $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{svd}(\mathbf{A})$  : computes singular value decomposition according to Thm. 6.5.6
- $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{svd}(\mathbf{A}, 0)$  : “economical” singular value decomposition for  $m > n$ : :  $\mathbf{U} \in \mathbb{K}^{m,n}$ ,  $\mathbf{\Sigma} \in \mathbb{R}^{n,n}$ ,  $\mathbf{V} \in \mathbb{K}^{n,n}$
- $\mathbf{s} = \text{svds}(\mathbf{A}, k)$  :  $k$  largest singular values (important for sparse  $\mathbf{A} \rightarrow$  Def. 2.6.1)
- $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{svds}(\mathbf{A}, k)$  : partial singular value decomposition:  $\mathbf{U} \in \mathbb{K}^{m,k}$ ,  $\mathbf{V} \in \mathbb{K}^{n,k}$ ,  $\mathbf{\Sigma} \in \mathbb{R}^{k,k}$  diagonal with  $k$  largest singular values of  $\mathbf{A}$ .

Explanation: “economical” singular value decomposition:

$$\begin{pmatrix} \text{A} \end{pmatrix} = \begin{pmatrix} \text{U} \end{pmatrix} \begin{pmatrix} \Sigma \end{pmatrix} \begin{pmatrix} \text{V}^H \end{pmatrix}$$

(MATLAB) algorithm for computing SVD is (numerically) stable  $\rightarrow$  Def. 2.5.11

Complexity:

$$\begin{aligned} & 2mn^2 + 2n^3 + O(n^2) + O(mn) \quad \text{for } s = \text{svd}(A), \\ & 4m^2n + 22n^3 + O(mn) + O(n^2) \quad \text{for } [U, S, V] = \text{svd}(A), \\ & O(mn^2) + O(n^3) \quad \text{for } [U, S, V] = \text{svd}(A, 0), m \gg n. \end{aligned}$$

- Application of SVD: computation of rank ( $\rightarrow$  Def. 2.0.2), kernel and range of a matrix

**Lemma 6.5.14** (SVD and rank of a matrix).  $\rightarrow$  [48, Cor. 9.7],

If the singular values of  $\mathbf{A}$  satisfy  $\sigma_1 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_p = 0$ , then

- $\text{rank}(\mathbf{A}) = r$  ,
- $\text{Ker}(\mathbf{A}) = \text{Span} \{ (\mathbf{V})_{:,r+1}, \dots, (\mathbf{V})_{:,n} \}$  ,
- $\text{Im}(\mathbf{A}) = \text{Span} \{ (\mathbf{U})_{:,1}, \dots, (\mathbf{U})_{:,r} \}$  .

Illustration:

columns = ONB of  $\text{Im}(\mathbf{A})$

rows = ONB of  $\text{Ker}(\mathbf{A})$

$$\begin{pmatrix} \mathbf{A} \end{pmatrix} = \begin{pmatrix} \mathbf{U} \end{pmatrix} \begin{pmatrix} \Sigma_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{V}^H \end{pmatrix} \tag{6.5.15}$$

Code 6.5.16: Computing an ONB of the kernel of a matrix

```

function V = kerncomp(A,tol)
% computes an orthonormal basis of Ker(A) using Lemma 6.5.14

```



```
3 % kernel selection with relative tolerance tol
4 if (nargin < 2), tol = eps; end
5 [U,S,V] = svd(A); % Singular value decomposition
6 s = diag(S); % Extract vector of singular values
7 % find singular values of relative (w.r.t.  $\sigma_1$ ) size  $\leq$  tol
8 r = min(find(s/s(1) <= tol)); % "Numerical rank" +1
9 V = V(:,r:end); % rightmost columns of V
```

Remark: MATLAB function `r=rank(A)` relies on `svd(A)`

- Application of SVD: PCA by SVD ( $\rightarrow$  Rem. 6.5.3)

By Rem. 6.5.10, see (6.5.12), if  $\mathbf{u}_j \hat{=}$  columns of  $\mathbf{U}$ ,  $\mathbf{v}_j \hat{=}$  columns of  $\mathbf{V}$ ,

$$\begin{pmatrix} \boxed{\mathbf{A}} \end{pmatrix} = \sigma_1 \begin{pmatrix} \boxed{\mathbf{u}_1} \end{pmatrix} \left( \boxed{\mathbf{v}_1^\top} \right) + \sigma_2 \begin{pmatrix} \boxed{\mathbf{u}_2} \end{pmatrix} \left( \boxed{\mathbf{v}_2^\top} \right) + \dots$$

❶ no perturbations:

SVD:  $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$  satisfies  $\sigma_1 \geq \sigma_2 \geq \dots \sigma_p > \sigma_{p+1} = \dots = \sigma_{\min\{m,n\}} = 0$  ,  
 $V = \text{Span} \underbrace{\{(\mathbf{U})_{:,1}, \dots, (\mathbf{U})_{:,p}\}}_{\text{orthonormal trend vectors}} .$

❷ with perturbations:

SVD:  $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$  satisfies  $\sigma_1 \geq \sigma_2 \geq \dots \sigma_p \gg \sigma_{p+1} \approx \dots \approx \sigma_{\min\{m,n\}} \approx 0$  ,  
 $V = \text{Span} \underbrace{\{(\mathbf{U})_{:,1}, \dots, (\mathbf{U})_{:,p}\}}_{\text{orthonormal trend vectors}} .$



$j$ -th data set ( $\leftrightarrow$  time series # $j$ ) in  $j$ -th column of  $\mathbf{A}$

$$(6.5.12) \Rightarrow (\mathbf{A})_{:,j} = \sigma_1 \mathbf{u}_1 (\mathbf{v}_1)_j + \sigma_2 \mathbf{u}_2 (\mathbf{v}_2)_j + \dots$$

► The  $j$ -th row of  $\mathbf{V}$  (up to the  $p$ -th component) gives the weights with which the  $p$  identified trends contribute to data set  $j$ .

Example 6.5.17 (Data points confined to a subspace).

### Code 6.5.18: PCA in three dimensions via SVD

```

1 % Use of SVD for PCA with perturbations
2
3 V = [1 , -1; 0 , 0.5; -1 , 0]; A = V*rand(2,20)+0.1*rand(3,20);
4 [U,S,V] = svd(A,0);
5
6 figure; sv = diag(S(1:3,1:3))
7
8 [X,Y] = meshgrid(-2:0.2:0,-1:0.2:1); n = size(X,1); m = size(X,2);
9 figure; plot3(A(1,:),A(2,:),A(3,:),'r*'); grid on; hold on;
0 M =
   U(:,1:2) * [reshape(X,1,prod(size(X))); reshape(Y,1,prod(size(Y)))] ;

```

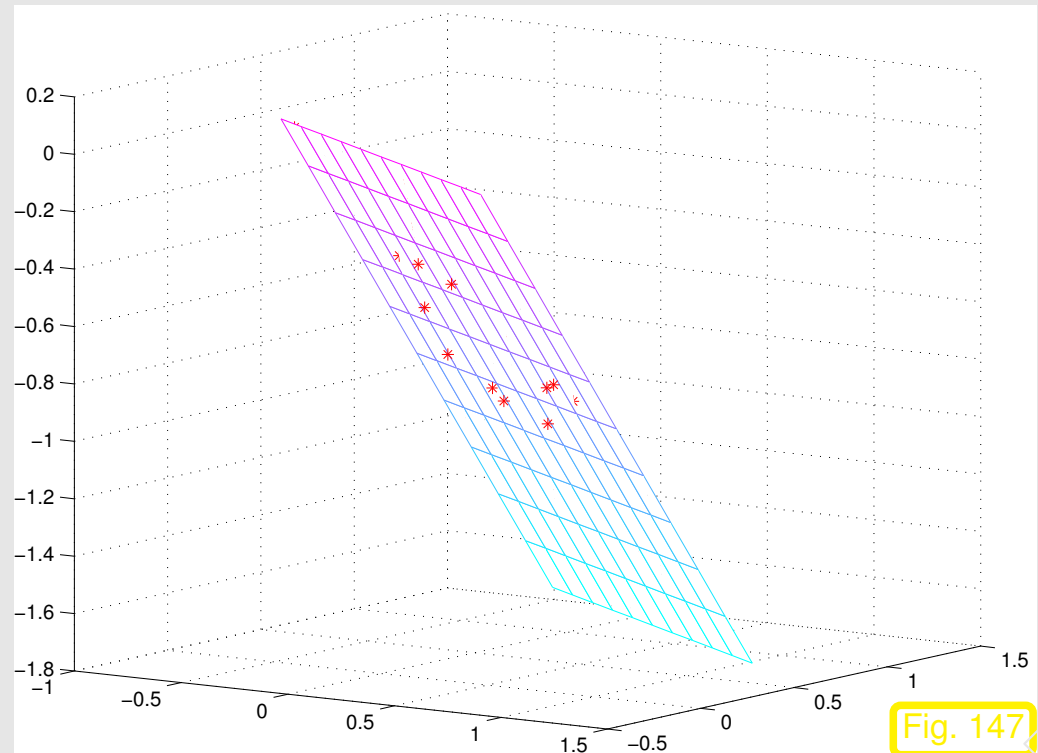
```

1 mesh ( reshape ( M ( 1 , : ) , n , m ) , reshape ( M ( 2 , : ) , n , m ) , reshape ( M ( 3 , : ) , n , m ) ) ;
2 colormap ( cool ) ; view ( 35 , 10 ) ;
3
4 print -depsc2 ' ../PICTURES/svdpca.eps ' ;

```

3.1378  
1.8092  
-----  
0.1792

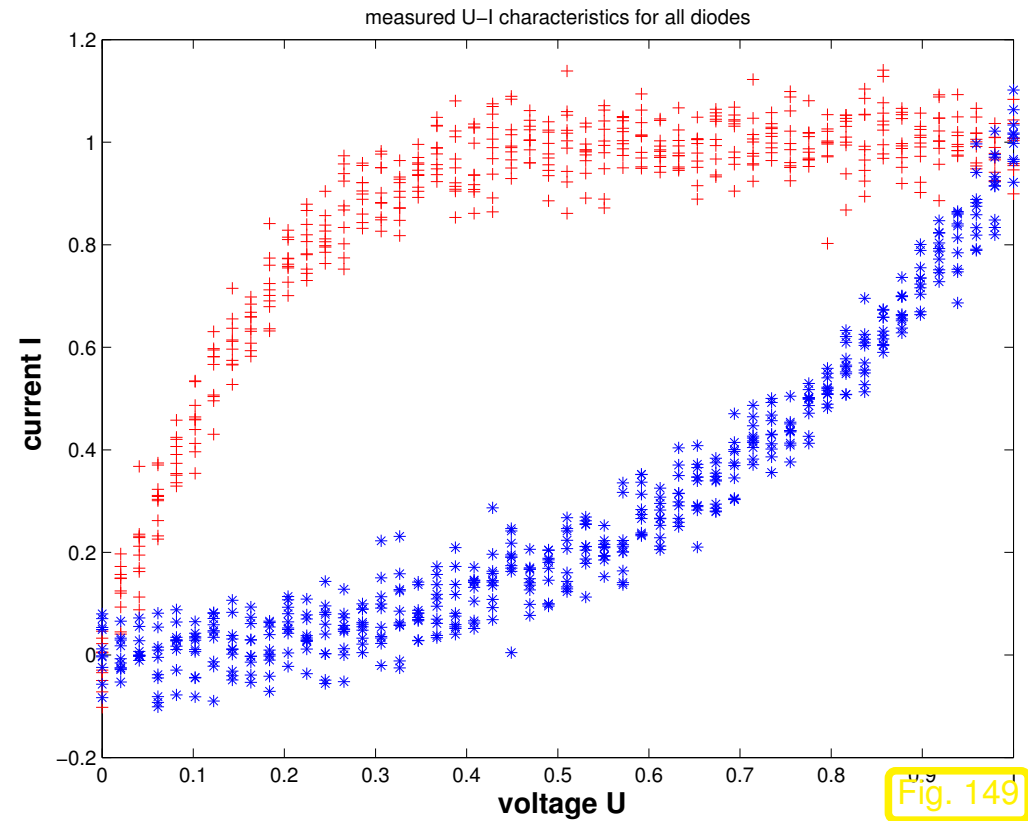
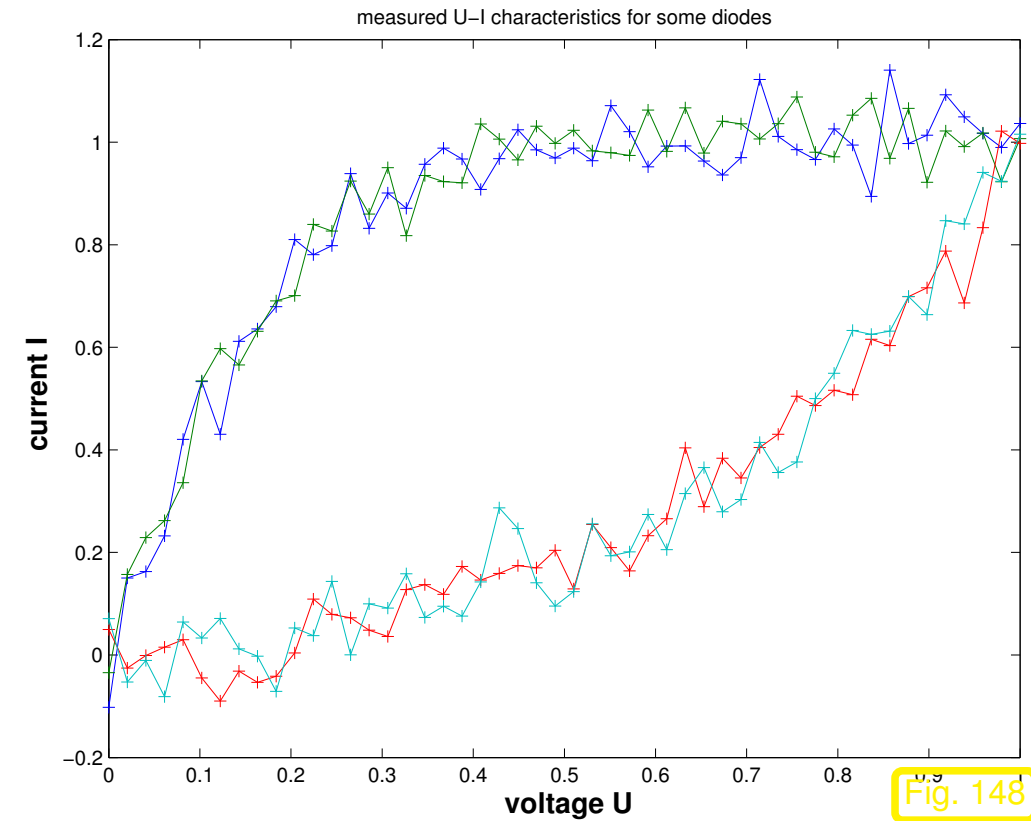
We observe a gap between the second and third singular value ➤ the data points essentially lie in a 2D subspace.



Example 6.5.19 (Principal component analysis for data classification). → Ex. 6.5.2 cnt'd

Given: measured  $U$ - $I$  characteristics of  $n = 20$  unknown diodes,  $I(U)$  available for  $m = 50$  volt-ages.

Sought: Number of different types of diodes in batch and reconstructed  $U$ - $I$  characteristic for each type.



Code 6.5.20: Generation of synthetic perturbed  $U$ - $I$  characteristics

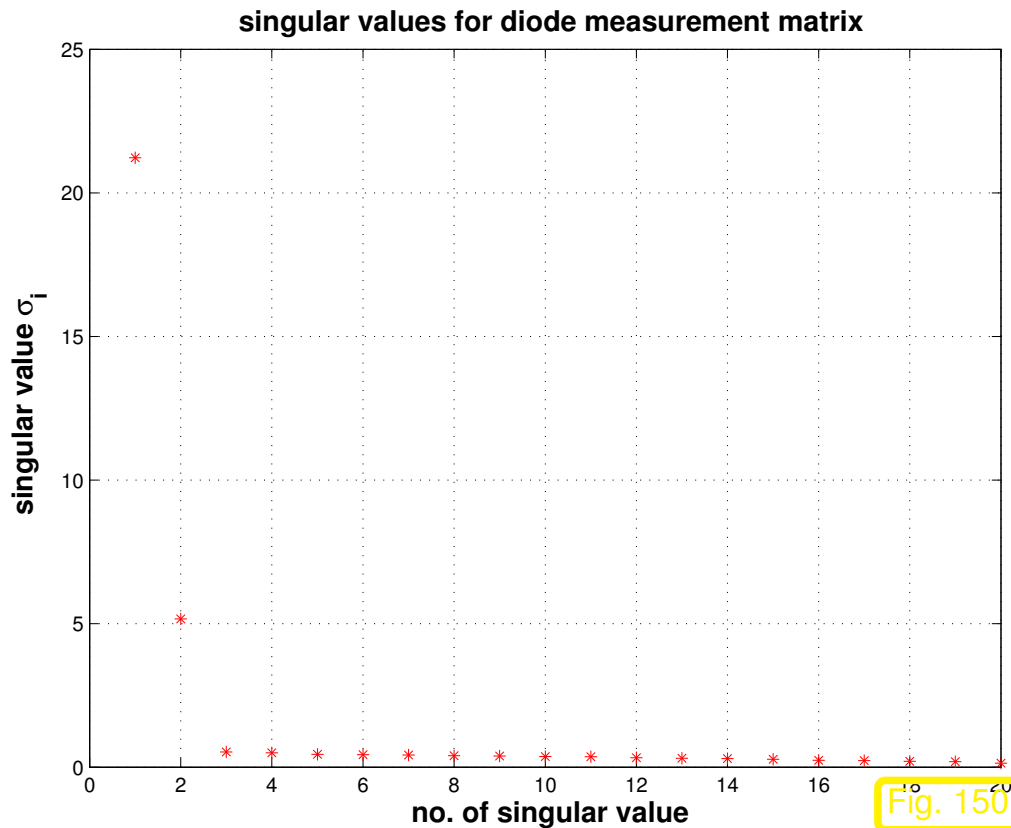
```
% Generate synthetic measurement curves with random multiplicative and additive
```

```
2 % perturbations supposed to reflect measurement errors and manufacturing
  tolerances
3
4 % Voltage-current characteristics of both kinds of diodes
5 i1 = @(u) (2./(1+exp(-10*u)) - 1);
6 i2 = @(u) ((exp(3*u)-1)/(exp(3)-1));
7 % Simulated measurements
8 m = 50; % number of measurements for different input voltages
9 n = 10; % no. of diodes of each kind
0 na = 0.05; % level of additive noise (normally distributed)
1 nm = 0.02; % level of multiplicative noise (normally distributed)
2
3 uvals = (0:1/(m-1):1);
4 D1 = (1+nm*randn(n,m)).*(i1(repmat(uvals,n,1)))+na*randn(n,m);
5 D2 = (1+nm*randn(n,m)).*(i2(repmat(uvals,n,1)))+na*randn(n,m);
6 A = ([D1;D2])'; A = A(1:size(A,1),randperm(1:size(A,2)));
```

Data matrix  $\mathbf{A} \in \mathbb{R}^{m,n}$ ,  $m \gg n$ :

Columns  $\mathbf{A}$   $\rightarrow$  series of measurements for different diodes (times/locations etc.),  
Rows of  $\mathbf{A}$   $\rightarrow$  measured values corresponding to one diode (time/location etc.).

Goal of PCA:

detect linear correlations between columns of  $A$ 

← distribution of singular values of matrix

two dominant singular values !

measurements display linear correlation with **two**  
principal components

**two** types of diodes in batch

R. Hiptmair  
rev 38286,  
November  
29, 2011Code 6.5.21: PCA for measured  $U$ - $I$  characteristics

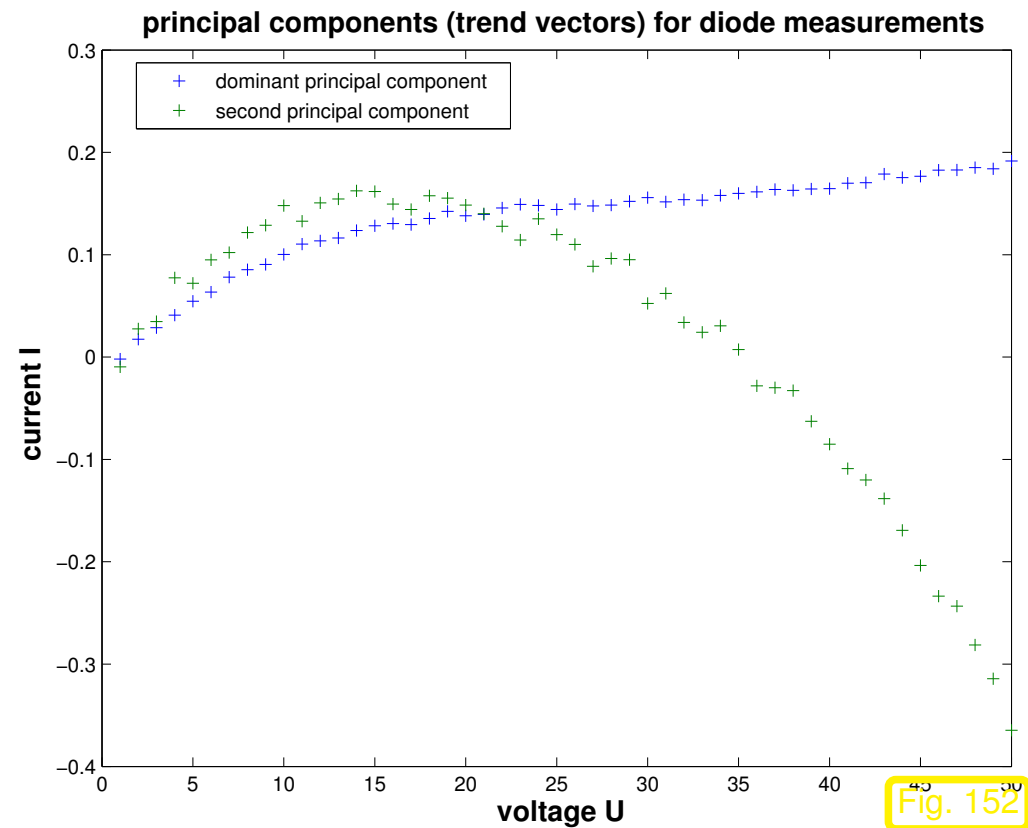
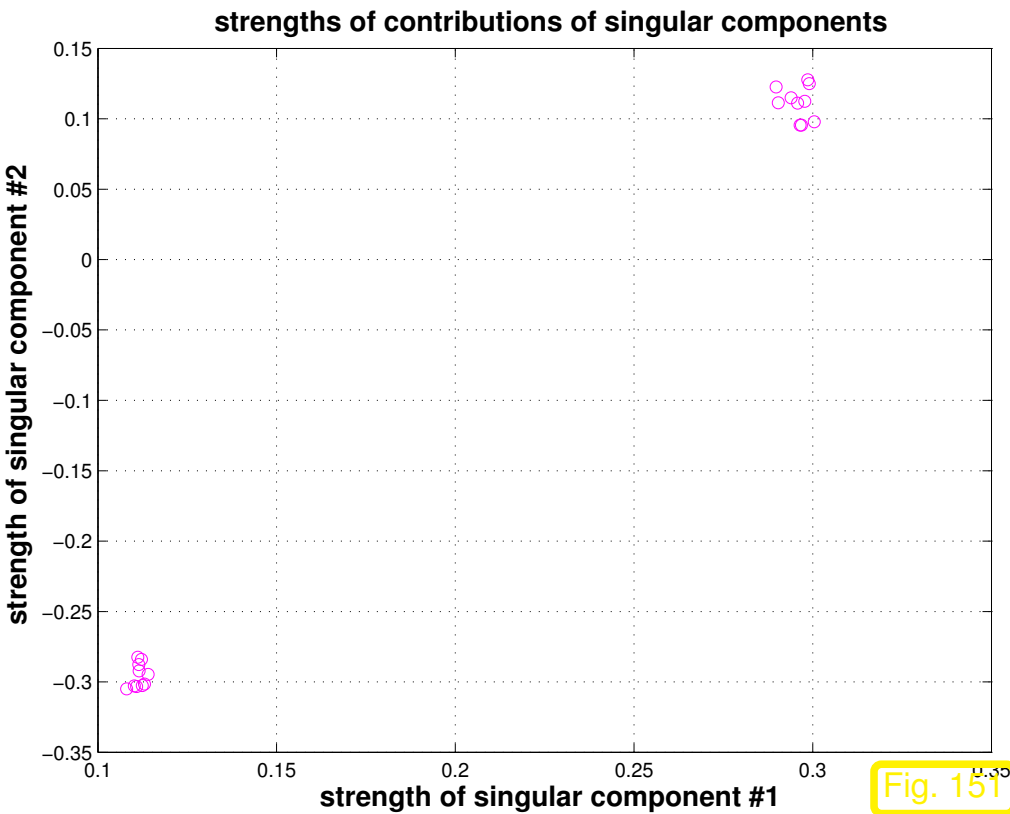
```

1 clear all; diodedata;
2
3 figure('name','U-I characteristics for a few diodes');
4 plot(uvals,(D(:,[1 7 13 17]))','-+');
```



```
5 xlabel ('{\bf voltage U}', 'fontsize', 14);
6 ylabel ('{\bf current I}', 'fontsize', 14);
7 title ('measured U-I characteristics for some diodes');
8 print -depsc2 '../PICTURES/diodepcafewmeas.eps';
9
0 figure ('name', 'measured U-I characteristics');
1 plot (uvals, D1', 'r+'); hold on;
2 plot (uvals, D2', 'b*');
3 xlabel ('{\bf voltage U}', 'fontsize', 14);
4 ylabel ('{\bf current I}', 'fontsize', 14);
5 title ('measured U-I characteristics for all diodes');
6 print -depsc2 '../PICTURES/diodepcameas.eps';
7
8 % Perform SVD based PCA
9 [U, S, V] = svd(D);
0
1 figure ('name', 'singular values');
2 sv = diag(S(1:2*n, 1:2*n));
3 plot(1:2*n, sv, 'r*'); grid on;
4 xlabel ('{\bf index i of singular value}', 'fontsize', 14);
5 ylabel ('{\bf singular value \sigma_i}', 'fontsize', 14);
6 title ('{\bf singular values for diode measurement
7 matrix}', 'fontsize', 14);
```

```
7 print -depsc2 '../PICTURES/diodepcasv.eps';
8
9 figure ('name', 'trend vectors');
0 plot (1:m, U(:, 1:2), '+');
1 xlabel ('\bf voltage U', 'fontsize', 14);
2 ylabel ('\bf current I', 'fontsize', 14);
3 title ('\bf principal components (trend vectors) for diode
  measurements', 'fontsize', 14);
4 legend ('dominant principal component', 'second principal
  component', 'location', 'best');
5 print -depsc2 '../PICTURES/diodepcav.eps';
6
7 figure ('name', 'strength');
8 plot (V(:, 1), V(:, 2), 'mo'); grid on;
9 xlabel ('\bf strength of singular component #1', 'fontsize', 14);
0 ylabel ('\bf strength of singular component #2', 'fontsize', 14);
1 title ('\bf strengths of contributions of singular
  components', 'fontsize', 14);
2 print -depsc2 '../PICTURES/diodepcav.eps';
```



## Observations:

- First two rows of  $\mathbf{V}$ -matrix specify strength of contribution of the two leading principal components to each measurement
- Points  $(\mathbf{V})_{:,1:2}$ , which correspond to different diodes are neatly clustered in  $\mathbb{R}^2$ . To determine the type of diode  $i$ , we have to identify the cluster to which the point  $((\mathbf{V})_{i,1}, \mathbf{V}_{i,2})$  belongs ( $\rightarrow$  **cluster analysis**, course “machine learning”, next Remark 6.5.22).

- The principal components themselves do not carry much useful information in this example.



*Remark 6.5.22. Algorithm for cluster analysis*

- Given:
- $N$  data points  $\mathbf{x}_i \in \mathbb{R}^k$ ,  $i = 1, \dots, N$ ,
  - No.  $n$  of desired clusters.

Sought: Partitioning of index set  $\{1, \dots, N\} = I_1 \cup \dots \cup I_n$ , achieving minimal **mean least squares error**

$$\text{mlse} := \sum_{l=1}^n \sum_{i \in I_l} \|\mathbf{x}_i - \mathbf{m}_l\|_2^2, \quad \mathbf{m}_l = \frac{1}{\#I_l} \sum_{i \in I_l} \mathbf{x}_i. \quad (6.5.23)$$

The subsets  $\{\mathbf{x}_i : i \in I_l\}$  are called the **clusters**. The points  $\mathbf{m}_l$  are their **centers of gravity**.

The Algorithm involves two components:

❶ Improvement of clusters using the **Lloyd-Max algorithm**, see Code 6.5.27. It involves two steps in turns:

(a) Given centers of gravity  $\mathbf{m}_l$  redistribute points according to

$$I_l := \{i \in \{1, \dots, N\} : \|\mathbf{x}_i - \mathbf{m}_l\|_2 \leq \|\mathbf{x}_i - \mathbf{m}_k\|_2 \quad \forall k \neq l\} . \quad (6.5.24)$$

(b) Recompute centers of gravity

$$\mathbf{m}_l = \frac{1}{\#I_l} \sum_{i \in I_l} \mathbf{x}_i . \quad (6.5.25)$$

❷ Splitting of cluster by separation along its **principal axis**, see Code 6.5.26:

$$\mathbf{a}_l := \operatorname{argmax}_{\|\mathbf{v}\|_2=1} \left\{ \sum_{i \in I_l} |(\mathbf{x}_i - \mathbf{m}_l)^\top \mathbf{v}|^2 \right\} \quad (6.5.26)$$

### Code 6.5.27: Principal axis point set separation

NumCSE,  
autumn  
2010

```
1 function [i1,i2] = princaxissep(X)
2 % Separation of a set of points whose coordinates are stored in the columns of
3 % X according to their location w.r.t. the principal axis
4
5 N = size (X,2); % no. of. points
6 g = sum (X')' /N; % Compute center of gravity, cf. (6.5.25)
7 Y = X - repmat(g,1,N); % Normalize point coordinates.
8
9 % Compute principal axes, cf. (6.5.26) and (6.5.33). Note
0 % that the SVD of a symmetric matrix is available through an orthonormal basis of
1 % eigenvectors.
2 [V,D] = eig (Y*Y');
3 a = V(:,end); % Major principal axis
4 c = a'*Y; % Coordinates of points w.r.t. to major principal axis
5 % Split point set according to locations of projections on principal axis
6 i1 = find (c < 0); i2 = find (c >= 0);
```

R. Hiptmair  
rev 38286,  
November  
29, 2011

### Code 6.5.28: Lloyd-Max algorithm for cluster identification

```
1 function [C,idx,cds] = lloydmax (X,C,tol)
2 % Lloyd-Max iterative vector quantization algorithm for discrete point sets; the
3 % columns of X contain the points  $x_i$ , the columns of
4 % C initial approximations for the centers of the clusters. The final centers
5 % are returned in C, the index vector idx specifies the association
```

6.5

p. 744

```
6 % of points with centers.
7 k = size(X,1); % dimension of space
8 N = size(X,2); % no. of points
9 n = size(C,2); % no. of clusters
0 if (k ~= size(C,1)), error('dimension mismatch'); end
1 if (nargin < 3), tol = 0.0001; end
2
3 sd_old = realmax;
4 [sd,idx] = distcomp(X,C),
5 % Terminate, if sum of squared minimal distances has not changed much
6 while ((sd_old-sd)/sd > tol)
7     % Compute new centers of gravity according to (6.5.25)
8     for j=1:n
9         idj = find(idx == j);
0         nj = length(idj);
1         if (nj > 0), C(:,j) = sum(X(:,idj)')')/nj; end
2     end
3     sd_old = sd;
4     [sd,idx,cds] = distcomp(X,C),
5 end
6
7 end
```

```
9 function [sumd,idx,cds] = distcomp(X,C)
0 % Compute squared distances
1 d = [];
2 for j=1:size(C,2)
3     Dv = X - repmat(C(:,j),1,size(X,2));
4     d = [d; sum(Dv.*Dv)];
5 end
6 % Compute minimum distance point association and sum of minimal squared
7 % distances
8 [mx,idx] = min(d);
9 sumd = sum(mx);
0 % Computer sum of squared distances within each cluster
1 for j=1:size(C,2)
2     cds(j) = sum(mx(find(idx == j)));
3 end
4 end
```

### Code 6.5.29: Clustering of point set

```
1 function [C,idx] = pointcluster(X,n)
2 % n-quantization of point set in k-dimensional space based on minimizing the
3 % mean square
4 % error of Euclidean distances. The columns of the matrix X contain the point
5 % coordinates, n specifies the desired number of clusters.
6 N = size(X,2); % no. of points
```



```
7 k = size (X,1); % dimension of space
8
9 % Start with two clusters obtained by principal axis separation
0 nc = 1; % Current number of clusters
1 Ibig = 1:N; % Initial single cluster encompassing all points
2 nbig = 1; % Index of largest cluster
3 C = sum (X')' /N; % center of gravity
4 idx = ones (1,N);
5
6 while (nc < n)
7     % Split largest cluster into two using the principal axis separation
8     % algorithm
9     [i1,i2] = princaxissep (X(:,Ibig));
0     i1 = Ibig(i1); i2 = Ibig(i2);
1     n1 = length (i1); n2 = length (i2);
2     c1 = sum (X(:,i1)')' /n1; c2 = sum (X(:,i2)')' /n2;
3     C(:,nbig) = c1; C = [C,c2];
4     nc = nc+1;
5     % Improve clusters by Lloyd-Max iteration
6     [C,idx,cds] = lloydmax (X,C);
7     % Identify cluster with biggest contribution to mean square error
8     [cdm,nbig] = max (cds);
9     Ibig = find (idx == nbig);
```

*Example 6.5.30* (PCA of stock prices). → Ex. 6.5.1

columns of  $A$  → time series of end of day stock prices of individual stocks  
rows of  $A$  → closing prices of DAX stocks on a particular day

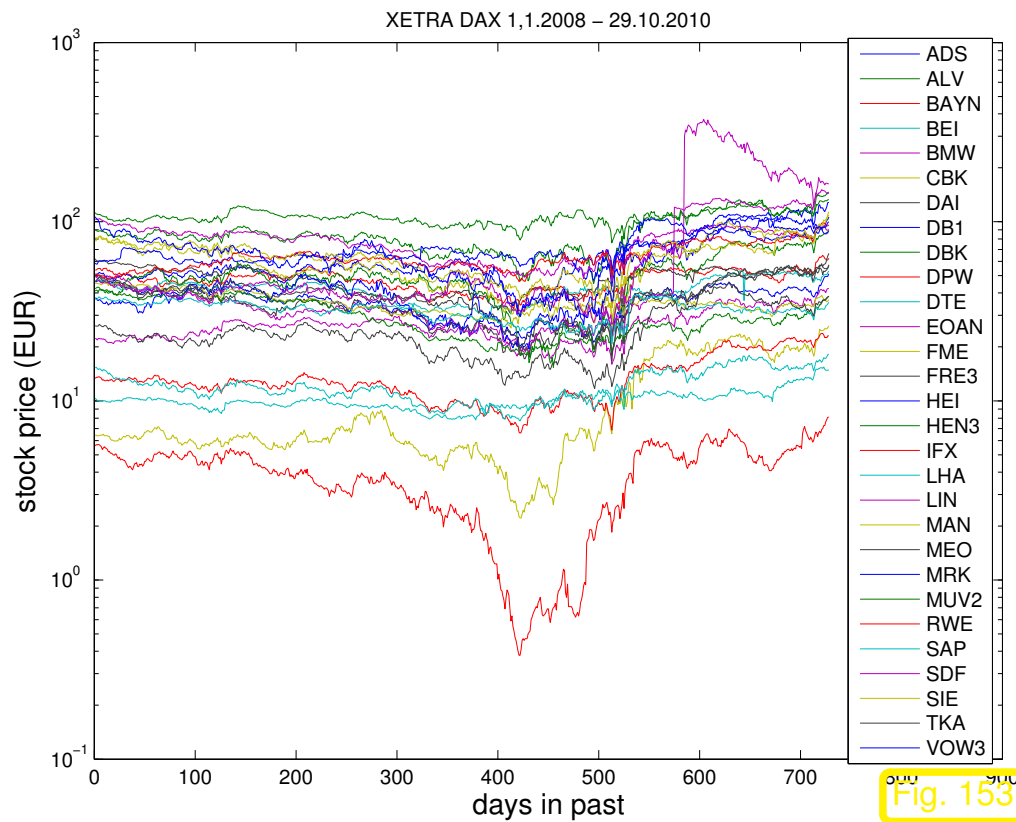


Fig. 153

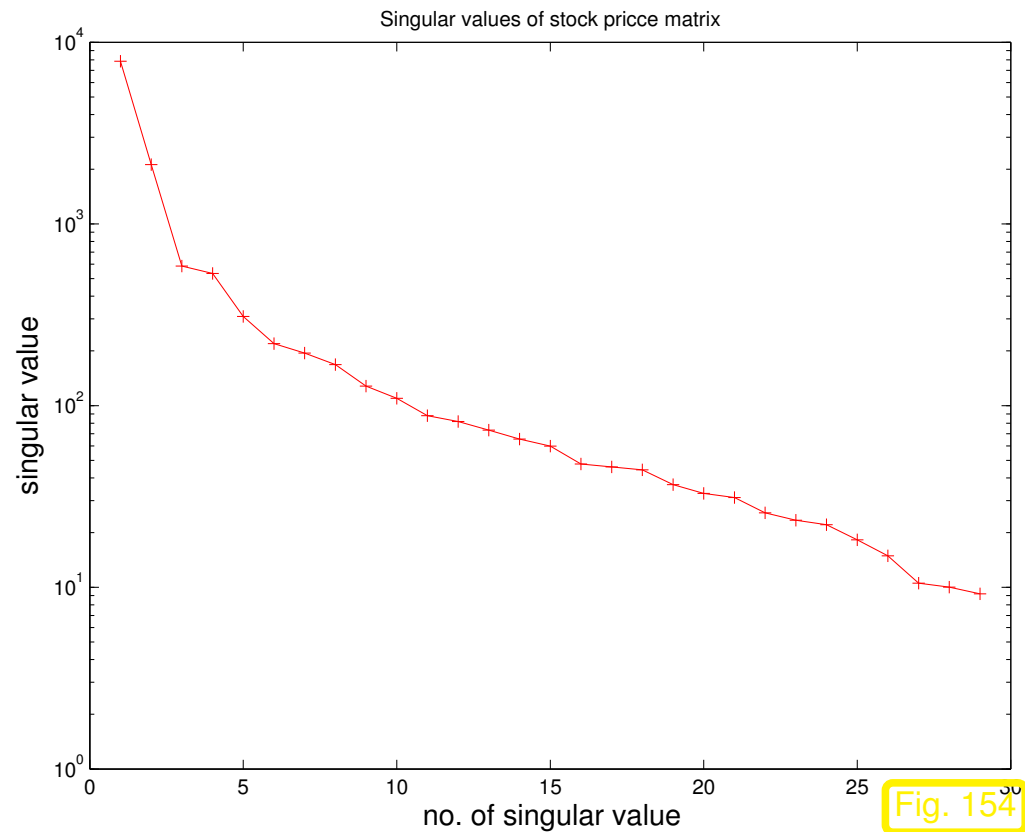
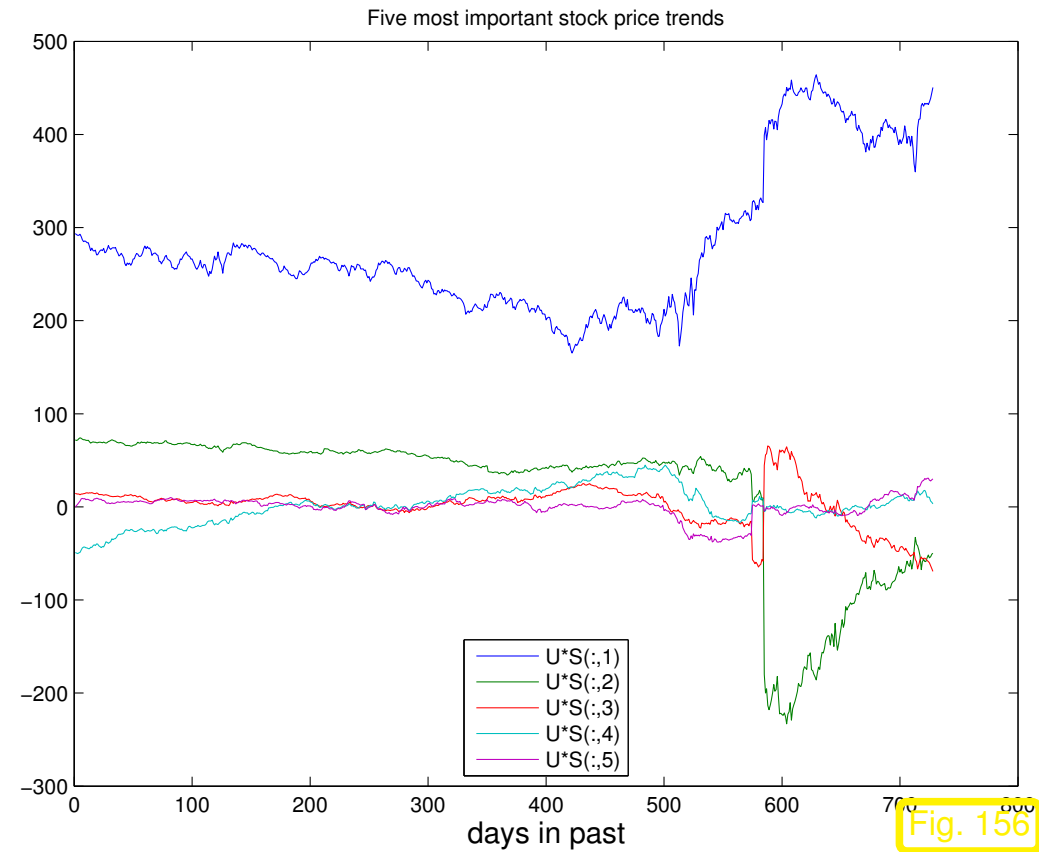
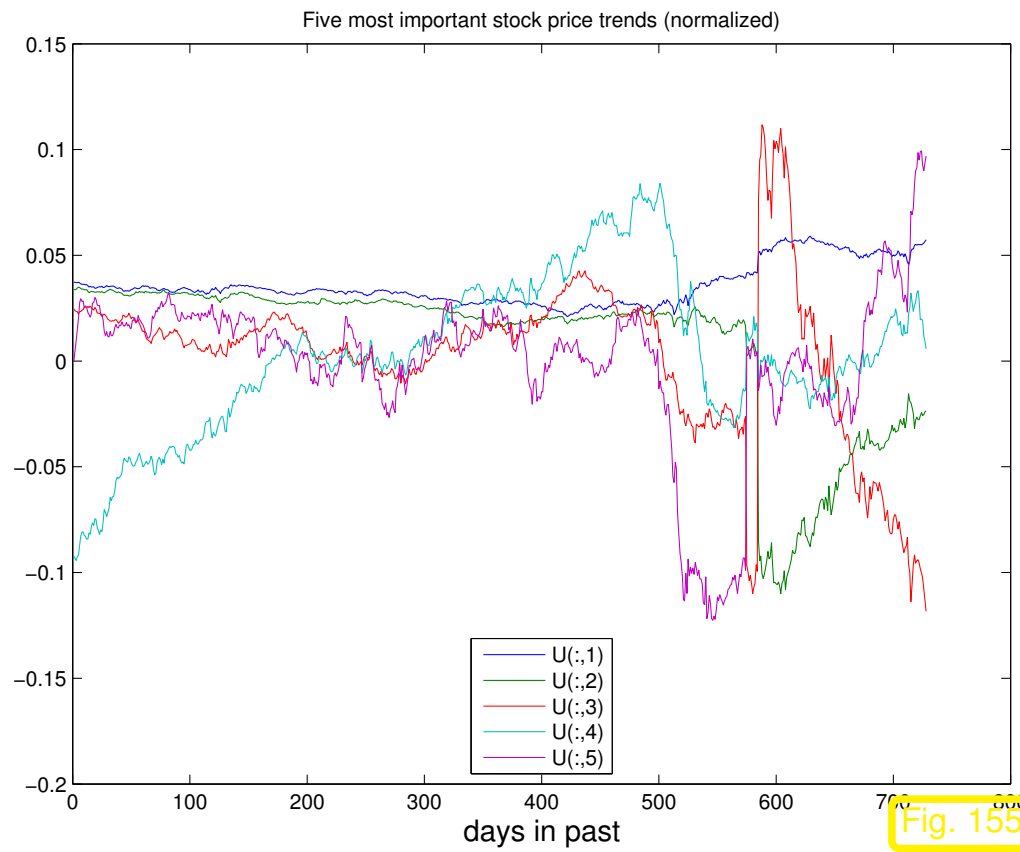


Fig. 154

We observe a pronounced decay of the singular values ( $\approx$  exponential decay, logarithmic scale in Fig. 154)

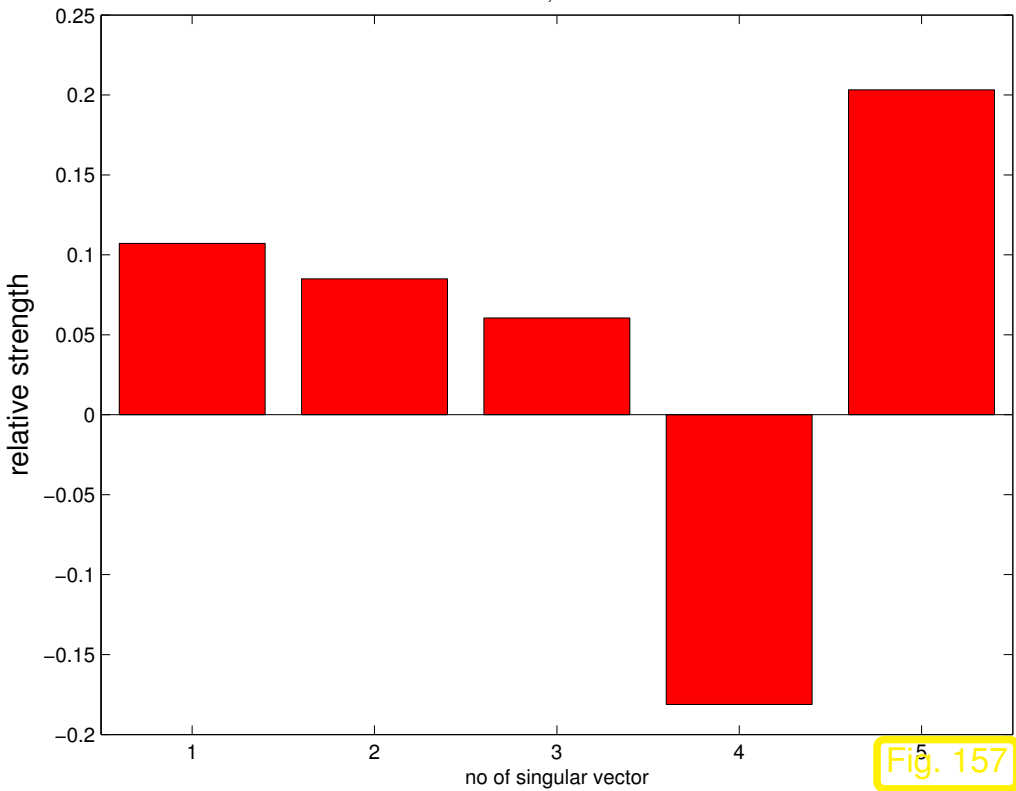
➤ a few trends (corresponding to a few of the largest singular values) govern the time series.



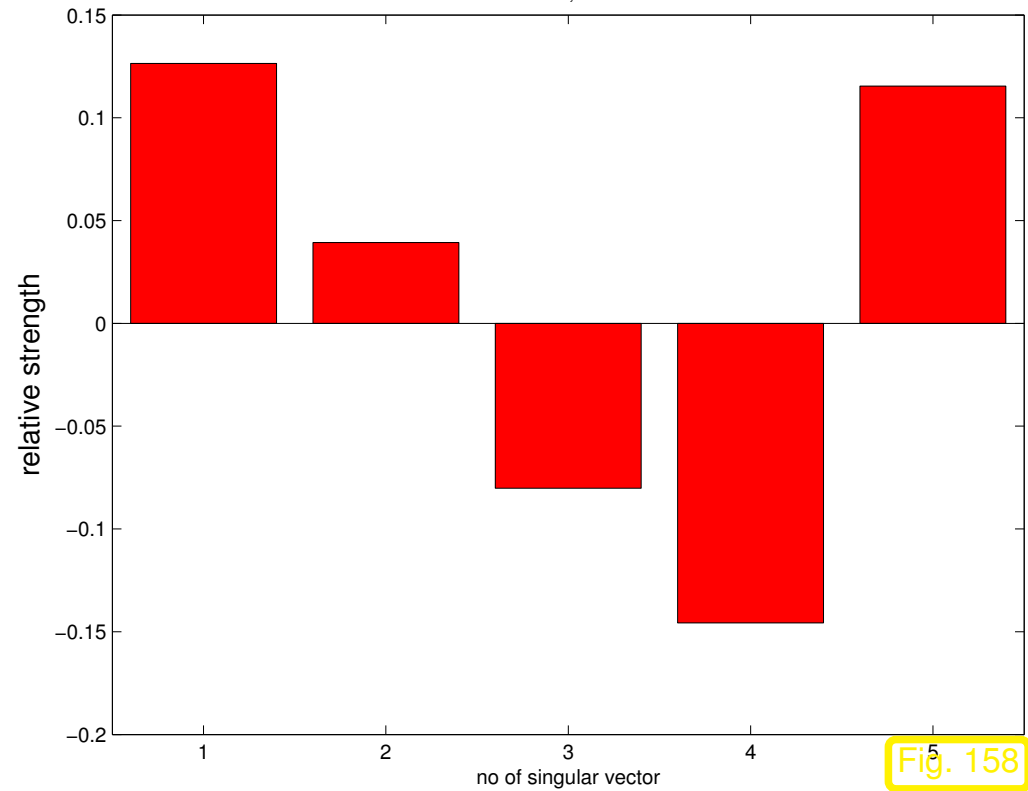
Columns of  $\mathbf{U}$  ( $\rightarrow$  Fig. 155) in SVD  $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$  provide trend vectors, *cf.* Rem. 6.5.3 & Ex. ??.

When weighted with the corresponding singular value, the importance of a trend contribution emerges, see Fig. 156

Trends in BMW stock, 1.1.2008 – 29.10.2010



Trends in Daimler stock, 1.1.2008 – 29.10.2010



Stocks of companies from the same sector of the economy should display similar contributions of major trend vectors, because their prices can be expected to be more closely correlated than stock prices in general.

## Data obtained from Yahoo Finance:

```

1 #!/bin/csh
2 foreach i (ADS ALV BAYN BEI BMW CBK DAI DBK DB1 LHA DPW DTE EOAN FRE3 \
3           FME HEI HEN3 IFX SDF LIN MAN MRK MEO MUV2 RWE SAP SIE TKA VOW3)
4 wget -O "i".csv "http://ichart.finance.yahoo.com/table.csv?s=i.DE&a=00&b=1&
5           c=2008&d=09&e=30&f=2010&g=d&ignore=.csv"
6 sed -i -e 's/-/,/g' "i".csv
7 end

```

## Code 6.5.31: PCA of stock prices in MATLAB

```

1 % Read end of day XETRA-DAX stock quotes into matrices
2 k = 1;
3 M{k} = dlmread ('ADS.csv', ',', 1, 0); leg{k} = 'ADS'; k = k + 1;
4 M{k} = dlmread ('ALV.csv', ',', 1, 0); leg{k} = 'ALV'; k = k + 1;
5 M{k} = dlmread ('BAYN.csv', ',', 1, 0); leg{k} = 'BAYN'; k = k + 1;
6 M{k} = dlmread ('BEI.csv', ',', 1, 0); leg{k} = 'BEI'; k = k + 1;
7 M{k} = dlmread ('BMW.csv', ',', 1, 0); leg{k} = 'BMW'; k = k + 1;
8 M{k} = dlmread ('CBK.csv', ',', 1, 0); leg{k} = 'CBK'; k = k + 1;
9 M{k} = dlmread ('DAI.csv', ',', 1, 0); leg{k} = 'DAI'; k = k + 1;
0 M{k} = dlmread ('DB1.csv', ',', 1, 0); leg{k} = 'DB1'; k = k + 1;
1 M{k} = dlmread ('DBK.csv', ',', 1, 0); leg{k} = 'DBK'; k = k + 1;
2 M{k} = dlmread ('DPW.csv', ',', 1, 0); leg{k} = 'DPW'; k = k + 1;
3 M{k} = dlmread ('DTE.csv', ',', 1, 0); leg{k} = 'DTE'; k = k + 1;
4 M{k} = dlmread ('EOAN.csv', ',', 1, 0); leg{k} = 'EOAN'; k = k + 1;
5 M{k} = dlmread ('FME.csv', ',', 1, 0); leg{k} = 'FME'; k = k + 1;
6 M{k} = dlmread ('FRE3.csv', ',', 1, 0); leg{k} = 'FRE3'; k = k + 1;
7 M{k} = dlmread ('HEI.csv', ',', 1, 0); leg{k} = 'HEI'; k = k + 1;
8 M{k} = dlmread ('HEN3.csv', ',', 1, 0); leg{k} = 'HEN3'; k = k + 1;
9 M{k} = dlmread ('IFX.csv', ',', 1, 0); leg{k} = 'IFX'; k = k + 1;

```

```
20 M{k} = dlmread ('LHA.csv', ',', 1, 0); leg{k} = 'LHA'; k = k + 1;
21 M{k} = dlmread ('LIN.csv', ',', 1, 0); leg{k} = 'LIN'; k = k + 1;
22 M{k} = dlmread ('MAN.csv', ',', 1, 0); leg{k} = 'MAN'; k = k + 1;
23 M{k} = dlmread ('MEO.csv', ',', 1, 0); leg{k} = 'MEO'; k = k + 1;
24 M{k} = dlmread ('MRK.csv', ',', 1, 0); leg{k} = 'MRK'; k = k + 1;
25 M{k} = dlmread ('MUV2.csv', ',', 1, 0); leg{k} = 'MUV2'; k = k + 1;
26 M{k} = dlmread ('RWE.csv', ',', 1, 0); leg{k} = 'RWE'; k = k + 1;
27 M{k} = dlmread ('SAP.csv', ',', 1, 0); leg{k} = 'SAP'; k = k + 1;
28 M{k} = dlmread ('SDF.csv', ',', 1, 0); leg{k} = 'SDF'; k = k + 1;
29 M{k} = dlmread ('SIE.csv', ',', 1, 0); leg{k} = 'SIE'; k = k + 1;
30 M{k} = dlmread ('TKA.csv', ',', 1, 0); leg{k} = 'TKA'; k = k + 1;
31 M{k} = dlmread ('VOW3.csv', ',', 1, 0); leg{k} = 'VOW3';
32 % Data format of matrices M:
33 % M(:,1): year, M(:,2): month, M(:,3): day, M(:,6): end of day stock quote
34 % Clean data: remove/interpolate days without trading
35 dv = [];
36 for j=1:k
37     M{j}(:,1) = round(366*M{j}(:,1)+31*M{j}(:,2)+M{j}(:,3));
38     dv = [dv; M{j}(:,1)];
39 end
40 dv = unique(dv); mxd = max(dv)+1;
41 dv = mxd - dv; mdays = max(dv);
42 A = zeros(mdays, k);
43 for j=1:k, A(mxd - M{j}(:,1), j) = M{j}(:,6); end
44 idx = find(sum(A') ~= 0); A = A(idx, :);
45
46 for j=size(A, 1):-1:2
47     zidx = find(A(j-1, :) == 0);
48     A(j-1, zidx) = A(j, zidx);
```

```
49 end
50 for j=2:size(A,1)
51     zidx = find(A(j,:) == 0);
52     A(j,zidx) = A(j-1,zidx);
53 end
54
55 figure('name','DAX');
56 semilogy(A); set(gca,'xlim',[0 900]);
57 legend(leg,'location','east');
58 xlabel('days in past','fontsize',14);
59 ylabel('stock price (EUR)','fontsize',14);
60 title('XETRA DAX 1,1.2008 - 29.10.2010');
61 print -depsc2 '../../PICTURES/stockprizes.eps';
62
63 [U,S,V] = svd(A,0);
64 figure('name','singular values');
65 semilogy(diag(S),'r-+');
66 xlabel('no. of singular value','fontsize',14);
67 ylabel('singular value','fontsize',14);
68 title('Singular values of stock price matrix');
69 print -depsc2 '../../PICTURES/stocksingval.eps';
70
71 figure('name','trend vectors');
72 plot(U(:,1:5));
73 xlabel('days in past','fontsize',14);
74 title('Five most important stock price trends (normalized)');
75 legend('U(:,1)','U(:,2)','U(:,3)','U(:,4)','U(:,5)','location','south');
76 print -depsc2 '../../PICTURES/stocktrendsn.eps';
77
```



```
78 figure ('name', 'trend vectors');
79 plot (U(:, 1:5)*S(1:5, 1:5));
80 xlabel ('days in past', 'fontsize', 14);
81 title ('Five most important stock price trends');
82 legend ('U*S(:, 1)', 'U*S(:, 2)', 'U*S(:, 3)', 'U*S(:, 4)', 'U*S(:, 5)', 'location', 'south')
83 print -depsc2 '../../PICTURES/stocktrends.eps';
84
85 figure ('name', 'trend components BMW');
86 trendcomp (V, 5, 5);
87 title ('Trends in BMW stock, 1.1.2008 - 29.10.2010');
88 print -depsc2 '../../PICTURES/bmw.eps';
89
90 figure ('name', 'trend components DAI');
91 trendcomp (V, 7, 5);
92 title ('Trends in Daimler stock, 1.1.2008 - 29.10.2010');
93 print -depsc2 '../../PICTURES/dai.eps';
94
95 figure ('name', 'trend components DBK');
96 trendcomp (V, 9, 5);
97 title ('Trends in Deutsche Bank stock, 1.1.2008 - 29.10.2010');
98 print -depsc2 '../../PICTURES/dbk.eps';
99
100 figure ('name', 'trend components CBK');
101 trendcomp (V, 6, 5);
102 title ('Trends in Commerzbank stock, 1.1.2008 - 29.10.2010');
103 print -depsc2 '../../PICTURES/cbk.eps';
```



- Application of SVD: extrema of quadratic forms on the unit sphere

A minimization problem on the Euclidean unit sphere  $\{\mathbf{x} \in \mathbb{K}^n: \|\mathbf{x}\|_2 = 1\}$ :

$$\text{given } \mathbf{A} \in \mathbb{K}^{m,n}, m > n, \text{ find } \mathbf{x} \in \mathbb{K}^n, \|\mathbf{x}\|_2 = 1, \|\mathbf{Ax}\|_2 \rightarrow \min. \quad (6.5.32)$$

Use that multiplication with unitary matrices preserves the 2-norm ( $\rightarrow$  Thm. 2.8.6) and the singular value decomposition  $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$  ( $\rightarrow$  Def. 6.5.8):

$$\begin{aligned} \min_{\|\mathbf{x}\|_2=1} \|\mathbf{Ax}\|_2^2 &= \min_{\|\mathbf{x}\|_2=1} \|\mathbf{U}\Sigma\mathbf{V}^H\mathbf{x}\|_2^2 = \min_{\|\mathbf{V}^H\mathbf{x}\|_2=1} \|\mathbf{U}\Sigma(\mathbf{V}^H\mathbf{x})\|_2^2 \\ &= \min_{\|\mathbf{y}\|_2=1} \|\Sigma\mathbf{y}\|_2^2 = \min_{\|\mathbf{y}\|_2=1} (\sigma_1^2 y_1^2 + \dots + \sigma_n^2 y_n^2) \geq \sigma_n^2. \end{aligned}$$

The minimum  $\sigma_n^2$  is attained for  $\mathbf{y} = \mathbf{e}_n \Rightarrow$  minimizer  $\mathbf{x} = \mathbf{V}\mathbf{e}_n = (\mathbf{V})_{:,n}$ .

By similar arguments:

$$\sigma_1 = \max_{\|\mathbf{x}\|_2=1} \|\mathbf{Ax}\|_2, \quad (\mathbf{V})_{:,1} = \operatorname{argmax}_{\|\mathbf{x}\|_2=1} \|\mathbf{Ax}\|_2. \quad (6.5.33)$$

Recall: 2-norm of the matrix  $\mathbf{A}$  ( $\rightarrow$  Def. 2.5.5) is defined as the maximum in (6.5.33). Thus we have proved the following theorem:

**Lemma 6.5.34** (SVD and Euclidean matrix norm).

- $\forall \mathbf{A} \in \mathbb{K}^{m,n}$ :  $\|\mathbf{A}\|_2 = \sigma_1(\mathbf{A})$ ,
- $\forall \mathbf{A} \in \mathbb{K}^{n,n}$  regular:  $\text{cond}_2(\mathbf{A}) = \sigma_1/\sigma_n$ .

Remark: MATLAB functions `norm(A)` and `cond(A)` rely on `svd(A)`

- Application of SVD: *best low rank approximation*

**Definition 6.5.35** (Frobenius norm).

The *Frobenius norm* of  $\mathbf{A} \in \mathbb{K}^{m,n}$  is defined as

$$\|\mathbf{A}\|_F^2 := \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 .$$

Obvious (why?):  $\|\mathbf{A}\|_F$  invariant under unitary transformations of  $\mathbf{A}$

Frobenius norm and SVD:

$$\|\mathbf{A}\|_F^2 = \sum_{j=1}^p \sigma_j^2 \quad (6.5.36)$$

 notation:  $\mathcal{R}_k(m, n) := \{\mathbf{A} \in \mathbb{K}^{m,n} : \text{rank}(\mathbf{A}) \leq k\}, m, n, k \in \mathbb{N}$

**Theorem 6.5.37** (best low rank approximation).  $\rightarrow [27, \text{Thm. 11.6}]$

Let  $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$  be the SVD of  $\mathbf{A} \in \mathbb{K}^{m,n}$  ( $\rightarrow$  Thm. 6.5.6). For  $1 \leq k \leq \text{rank}(\mathbf{A})$  set  $\mathbf{U}_k := [\mathbf{u}_{\cdot,1}, \dots, \mathbf{u}_{\cdot,k}] \in \mathbb{K}^{m,k}$ ,  $\mathbf{V}_k := [\mathbf{v}_{\cdot,1}, \dots, \mathbf{v}_{\cdot,k}] \in \mathbb{K}^{n,k}$ ,  $\mathbf{\Sigma}_k := \text{diag}(\sigma_1, \dots, \sigma_k) \in \mathbb{K}^{k,k}$ . Then, for  $\|\cdot\| = \|\cdot\|_F$  and  $\|\cdot\| = \|\cdot\|_2$ , holds true

$$\|\mathbf{A} - \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^H\| \leq \|\mathbf{A} - \mathbf{F}\| \quad \forall \mathbf{F} \in \mathcal{R}_k(m, n).$$

Thm 6.5.37: the rank- $k$ -matrix that is *closest* to  $\mathbf{A}$  (rank- $k$  best approximation) in either the Euclidean matrix norm or the Frobeniusnorm ( $\rightarrow$  Def. 6.5.35) can be obtained by truncating the rank-1 sum expansion (6.5.12) from the SVD of  $\mathbf{A}$  after  $k$  terms.

*Proof.* Write  $\mathbf{A}_k = \mathbf{U}_k \Sigma_k \mathbf{V}_k^H$ . Obviously, with  $r = \text{rank } \mathbf{A}$ ,

$$\text{rank } \mathbf{A}_k = k \quad \text{and} \quad \|\mathbf{A} - \mathbf{A}_k\| = \|\Sigma - \Sigma_k\| = \begin{cases} \sigma_{k+1} & , \text{ for } \|\cdot\| = \|\cdot\|_2 , \\ \sqrt{\sigma_{k+1}^2 + \dots + \sigma_r^2} & , \text{ for } \|\cdot\| = \|\cdot\|_F . \end{cases}$$

① Pick  $\mathbf{B} \in \mathbb{K}^{n,n}$ ,  $\text{rank } \mathbf{B} = k$ .

$$\blacktriangleright \quad \dim \text{Ker}(\mathbf{B}) = n - k \quad \Rightarrow \quad \text{Ker}(\mathbf{B}) \cap \text{Span} \{\mathbf{v}_1, \dots, \mathbf{v}_{k+1}\} \neq \{0\} ,$$

where  $\mathbf{v}_i$ ,  $i = 1, \dots, n$  are the columns of  $\mathbf{V}$ . For  $\mathbf{x} \in \text{Ker}(\mathbf{B}) \cap \text{Span} \{\mathbf{v}_1, \dots, \mathbf{v}_{k+1}\}$ ,  $\|\mathbf{x}\|_2 = 1$

$$\mathbf{x} = \sum_{j=1}^{k+1} (\mathbf{v}_j^H \mathbf{x}) \mathbf{v}_j ,$$

$$\|\mathbf{A} - \mathbf{B}\|_2^2 \geq \|(\mathbf{A} - \mathbf{B})\mathbf{x}\|_2^2 = \|\mathbf{A}\mathbf{x}\|_2^2 = \left\| \sum_{j=1}^{k+1} \sigma_j (\mathbf{v}_j^H \mathbf{x}) \mathbf{u}_j \right\|_2^2 = \sum_{j=1}^{k+1} \sigma_j^2 (\mathbf{v}_j^H \mathbf{x})^2 \geq \sigma_{j+1}^2 ,$$

because  $\sum_{j=1}^{k+1} (\mathbf{v}_j^H \mathbf{x})^2 = \|\mathbf{x}\|_2^2 = 1$ .

② Find ONB  $\{\mathbf{z}_1, \dots, \mathbf{z}_{n-k}\}$  of  $\text{Ker}(\mathbf{B})$  and assemble it into a matrix  $\mathbf{Z} = [\mathbf{z}_1 \dots \mathbf{z}_{n-k}] \in \mathbb{K}^{n,n-k}$

$$\|\mathbf{A} - \mathbf{B}\|_F^2 \geq \|(\mathbf{A} - \mathbf{B})\mathbf{Z}\|_F^2 = \|\mathbf{AZ}\|_F^2 = \sum_{i=1}^{n-k} \|\mathbf{Az}_i\|_2^2 = \sum_{i=1}^{n-k} \sum_{j=1}^r \sigma_j^2 (\mathbf{v}_j^H \mathbf{z}_i)^2 \quad \square$$

Since matrix norms  $\|\cdot\|_2$  and  $\|\cdot\|_F$  are invariant under multiplication with orthogonal (unitary) matrices,

$$\left\| \mathbf{A} - \mathbf{U}_k \Sigma_k \mathbf{V}_k^H \right\|_2 = \sigma_{k+1}, \quad (6.5.38)$$

$$\left\| \mathbf{A} - \mathbf{U}_k \Sigma_k \mathbf{V}_k^H \right\|_F^2 = \sum_{j=k+1}^{\min\{m,n\}} \sigma_j^2. \quad (6.5.39)$$

This provides precise information about the best approximation error for rank- $k$ -matrices.

Note: information content of a rank- $k$  matrix  $\mathbf{M} \in \mathbb{K}^{m,n}$  is equivalent to  $k(m+n)$  numbers!

Approximation by low-rank matrices  $\leftrightarrow$  **matrix compression**

*Example 6.5.40* (Image compression).

Image composed of  $m \times n$  pixels (greyscale, BMP format)  $\leftrightarrow$  matrix  $\mathbf{A} \in \mathbb{R}^{m,n}$ ,  $a_{ij} \in \{0, \dots, 255\}$ ,  
see Ex. 6.3.22

► Thm. 6.5.37 ➤ best low rank approximation of image:  $\tilde{\mathbf{A}} = \mathbf{U}_k \Sigma_k \mathbf{V}^\top$

Code 6.5.41: SVD based image compression

```

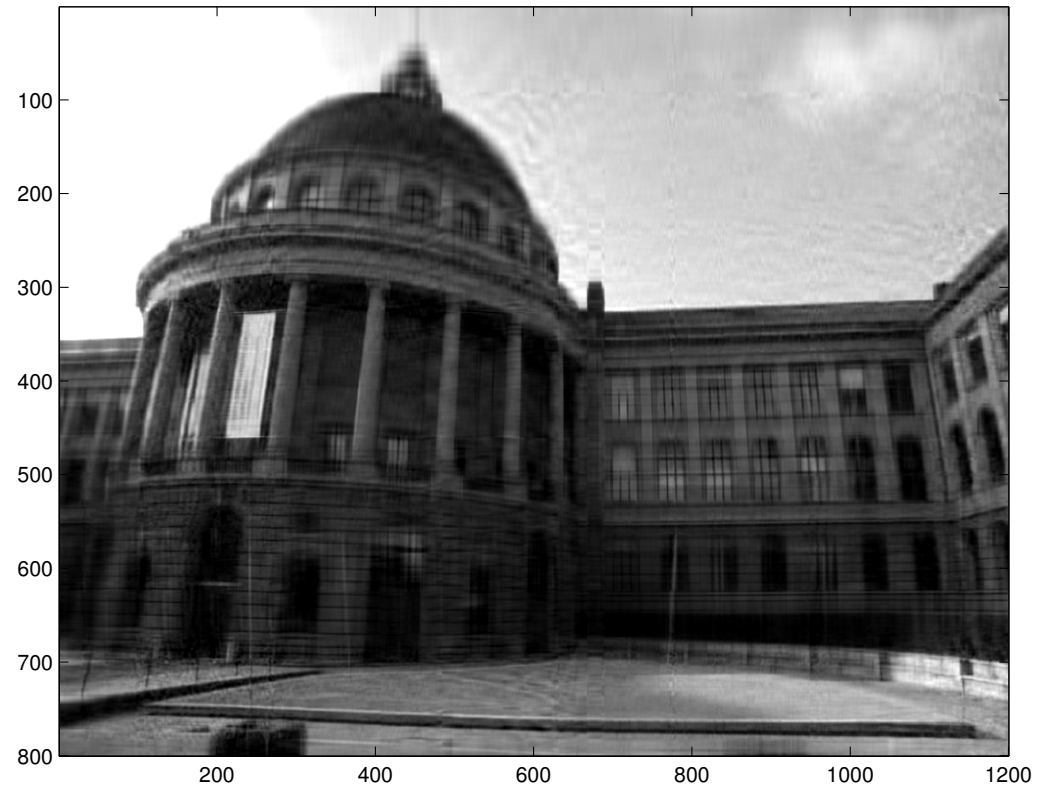
1 P = double(imread('eth.pbm'));
2 [m,n] = size(P); [U,S,V] = svd(P); s = diag(S);
3 k = 40; S(k+1:end,k+1:end) = 0; PC = U*S*V';
4
5 figure('position',[0 0 1600 1200]); col = [0:1/215:1]'*[1,1,1];
6 subplot(2,2,1); image(P); title('original image'); colormap(col);
7 subplot(2,2,2); image(PC); title('compressed (40 S.V.)');
  colormap(col);
8 subplot(2,2,3); image(abs(P-PC)); title('difference');
  colormap(col);
9 subplot(2,2,4); cla; semilogy(s); hold on; plot(k,s(k),'ro');

```

View of ETH Zurich main building

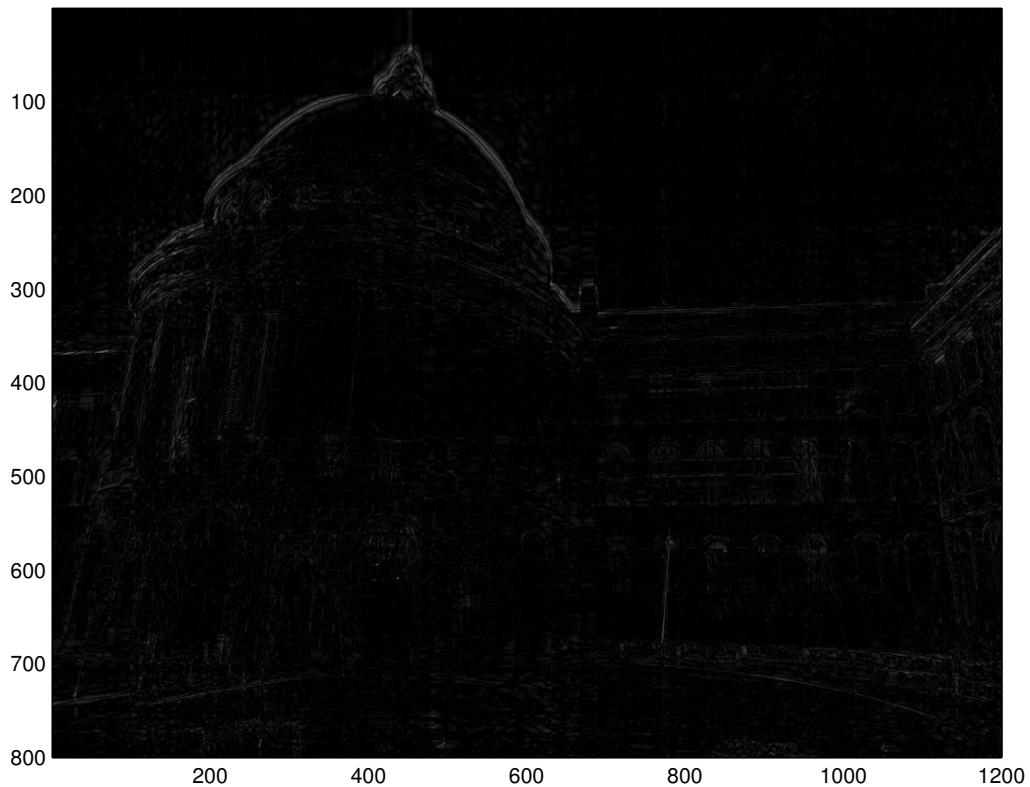


Compressed image, 40 singular values used

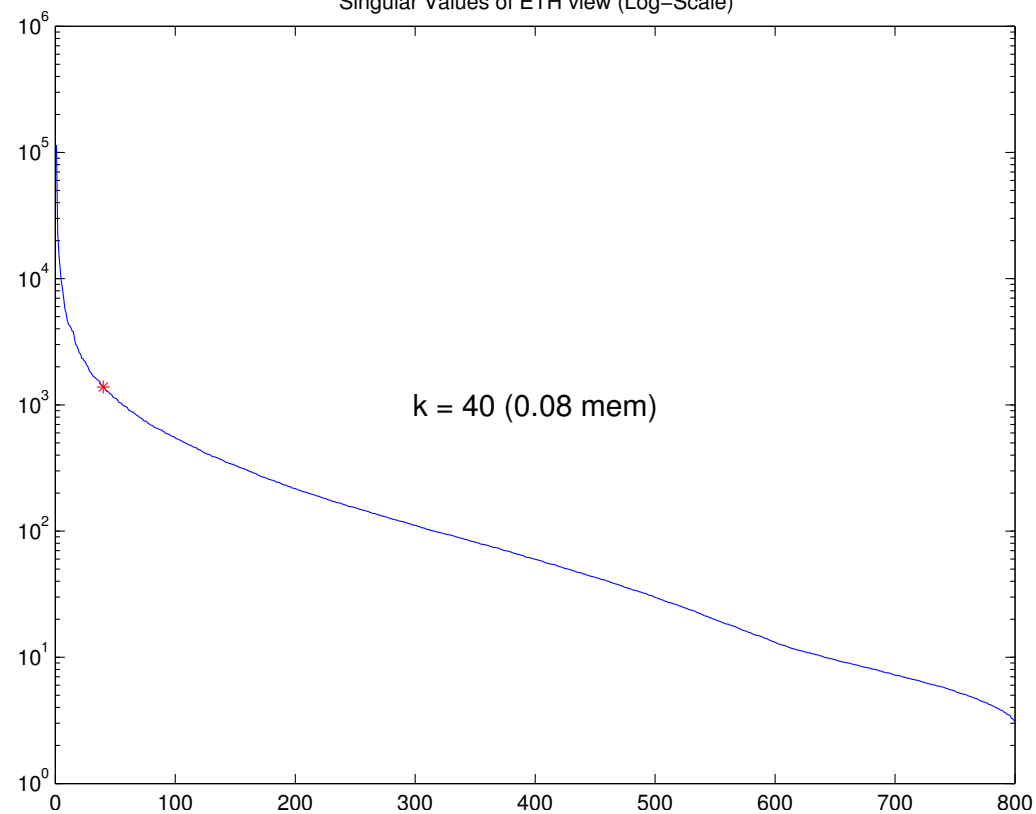




Difference image: |original – approximated|



Singular Values of ETH view (Log-Scale)



However: there are better and faster ways to compress images than SVD (JPEG, Wavelets, etc.)



# 7

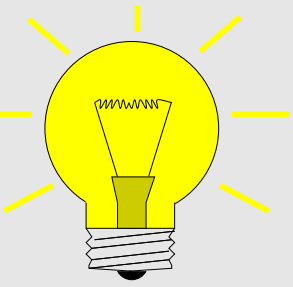
## Least Squares

*Example 7.0.1* (Least squares data fitting).

In Sect. 3.1 we discussed the reconstruction of a *parameterized function*  $f(x_1, \dots, x_n; \cdot) : D \subset \mathbb{R} \mapsto \mathbb{R}$  from data points  $(t_i, y_i)$ ,  $i = 1, \dots, n$ , by imposing interpolation conditions (3.1.1). Necessary was that the number  $n$  of parameters agreed with number of data points. The interpolation approach is justified in the case of *highly accurate data*.

Frequently encountered: *inaccurate data* (due to *measurement errors*)

➤ interpolation approach dubious (impact of “outliers”!)



Mitigate impact of data uncertainty by  
choosing fewer parameters than data points

(measurement errors can “average out”)

Non-linear least squares fitting problem: [13, Sect. 6.1]

- Given:
- data points  $(t_i, y_i)$ ,  $i = 1, \dots, m$
  - (symbolic formula) for parameterized function

$$f(x_1, \dots, x_n; \cdot) : D \subset \mathbb{R} \mapsto \mathbb{R}, \quad n < m$$

Sought: parameter values  $x_1^*, \dots, x_n^* \in \mathbb{R}$  such that

$$(x_1^*, \dots, x_n^*) = \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^m |f(x_1, \dots, x_n; t_i) - y_i|^2. \quad (7.0.2)$$

Example 7.0.3 (Linear data fitting).  $\rightarrow$  [13, Sect. 4.1]  $\rightarrow$  [13, Ex. 4.2 & Ex. 4.3]

Special case, *cf.* in Sect. 3.1, (3.1.4): Representation of  $f$  by *finite linear combination* of **basis functions**  $b_j : D \subset \mathbb{R} \mapsto \mathbb{R}$ ,  $j = 1, \dots, n$ :

$$f(t) = \sum_{j=1}^n x_j b_j(t) \quad , \quad x_j \in \mathbb{R}^d . \quad (7.0.4)$$

→  $f \in$  finite dimensional **function space**  $V_n := \text{Span} \{b_1, \dots, b_n\}$ .

### Linear least squares fitting problem:

- Given:
- data points  $(t_i, y_i)$ ,  $i = 1, \dots, m$
  - basis functions  $b_j : I \mapsto \mathbb{K}$ ,  $j = 1, \dots, n$ ,  $n < m$

Sought: coefficients  $x_j^* \in \mathbb{R}$ ,  $j = 1, \dots, n$ , such that

$$(x_1^*, \dots, x_n^*) = \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^m \left| \sum_{j=1}^n x_j b_j(t_i) - y_i \right|^2 . \quad (7.0.5)$$

*Example 7.0.6* (Polynomial fitting).

Special variant of linear data fitting ( $\rightarrow$  Ex. 7.0.3): choose  $f$  as polynomial of degree  $n - 1$ ,

$$V_n = \mathcal{P}_{n-1} \quad , \quad \text{e.g.} \quad b_j(t) = t^{j-1} \quad (\text{monomial basis, Sect. 3.2}) .$$

$\blackrightarrow$  MATLAB-function for solving (7.0.2) in this case:

```
p = polyfit(t, y, d);
```

$d \hat{=}$  polynomial degree,  $t \hat{=}$  vector  $(t_i)_{i=1}^n$ ,  $y \hat{=}$  vector  $(y_i)_{i=1}^n$

Note:  $p \hat{=}$  vector of monomial coefficients in MATLAB convention, see Rem. 3.2.3.

*Example 7.0.7* (Polybomial interpolation vs. polynomial fitting).

## Code 7.0.8: Fitting and interpolating polynomial

```
% Comparison of polynomial interpolation and polynomial fitting
```

```
2 % ("Quick and dirty" MATLAB implementation, see 3.4)
3 f = @(x) 1./(1+x.^2); % Function providing data points
4 d = 10; % Polynomial degree
5 tip = -5+(0:d)*10/d; % d+1 nodes for interpolation
6 tft = -5+(0:3*d)*10/(3*d); % 3d+1 Nodes for polynomial fitting
7 pip = polyfit(tip, f(tip), d); % Interpolating polynomial (degree d)
8 pft = polyfit(tft, f(tft), d); % Fitting polynomial (degree d)
9 x = -5+(0:1000)/100;
0 h = plot(x, f(x), 'g--', ...
1         x, polyval(pip, x), 'b-', ...
2         x, polyval(pft, x), 'r-', ...
3         tip, f(tip), 'b*');
4 set(h(1), 'linewidth', 2);
5 xlabel('\bf t', 'fontsize', 14);
6 ylabel('\bf y', 'fontsize', 14);
7 legend('function f', 'interpolating polynomial', 'fitting
8 polynomial', 'location', 'north');
9 print -depsc2 '../PICTURES/interpfit.eps';
```

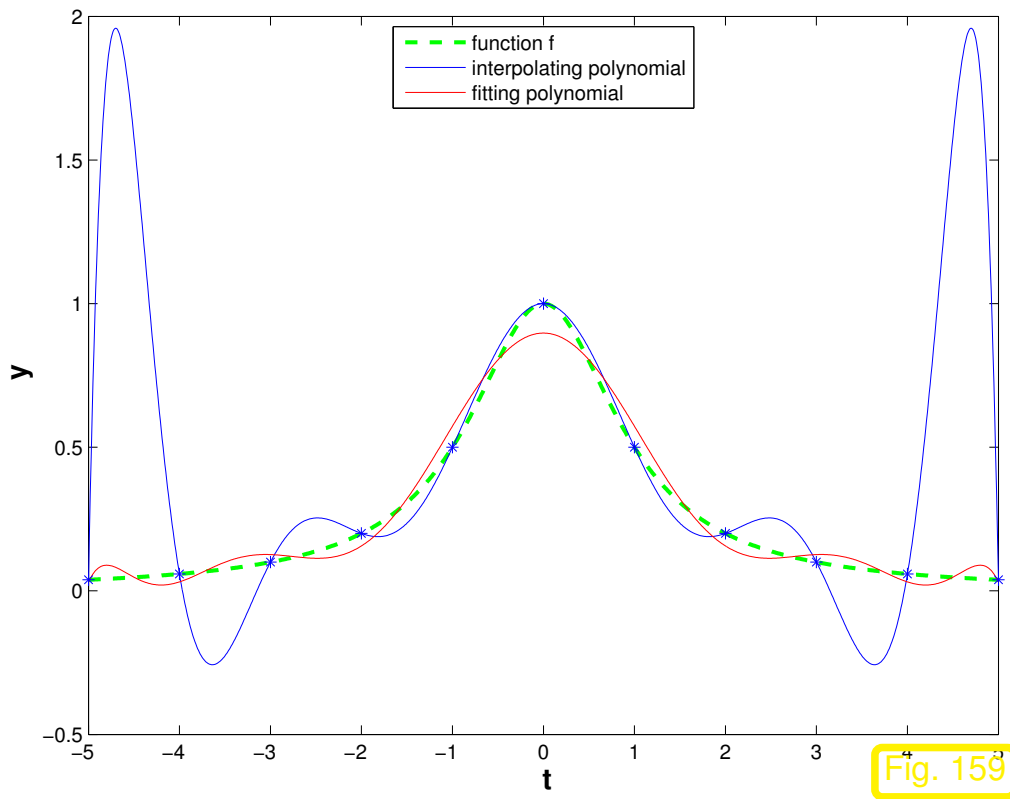


Fig. 159

Data from function  $f(t) = \frac{1}{1+t^2}$ ,

- polynomial degree  $d = 10$ ,
- interpolation through data points  $(t_j, f(t_j))$ ,  $j = 0, \dots, d$ ,  $t_j = -5 + j$ , see Ex. 3.5.1,
- fitting to data points  $(t_j, f(t_j))$ ,  $j = 0, \dots, 3d$ ,  $t_j = -5 + j/3$ .

Fitting helps curb oscillations that haunt polynomial interpolation!



Example 7.0.9 (linear regression). → [13, Ex. 4.1]

☞ example for multidimensional linear least squares data fitting:

Given: *measured* data points  $(\mathbf{x}_i, y_i)$ ,  $\mathbf{x}_i \in \mathbb{R}^n$ ,  $y_i \in \mathbb{R}$ ,  $i = 1, \dots, m$ ,  $m \geq n + 1$   
( $y_i$ ,  $\mathbf{x}_i$  affected by measurement errors).

Known: without measurement errors data would satisfy  
*affine linear relationship*  $y = \mathbf{a}^\top \mathbf{x} + c$ ,  $\mathbf{a} \in \mathbb{R}^n$ ,  $c \in \mathbb{R}$ .

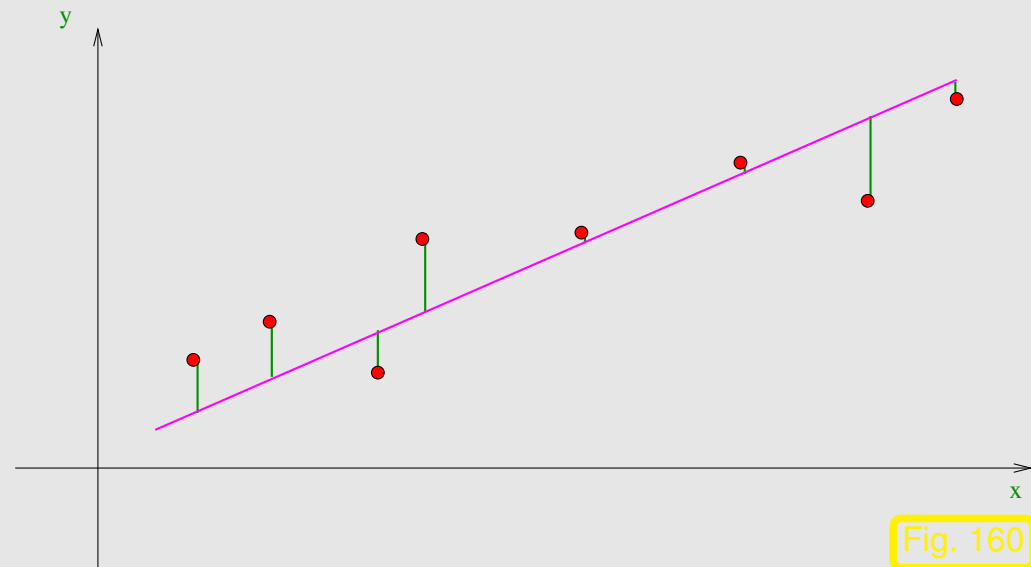
Goal: *estimate* parameters  $\mathbf{a}$ ,  $c$ .

*least squares* estimate

$$(\mathbf{a}, c) = \underset{\mathbf{p} \in \mathbb{R}^n, q \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^m |y_i - \mathbf{p}^\top \mathbf{x}_i - q|^2 \quad (7.0.10)$$

linear regression for  $n = 2, m = 8$

▷.



Remark, see [13, Sect. 4.5]: In statistics we learn that the least squares estimate provides a maximum likelihood estimate, if the measurement errors are uniformly and independently normally



distributed.

*Remark 7.0.11* (Overdetermined linear systems).

In Ex. 7.0.9 we could try to find  $\mathbf{a}$ ,  $c$  by solving the linear system of equations

$$\begin{pmatrix} \mathbf{x}_1^\top & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^\top & 1 \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}, \quad (7.0.12)$$

but in case  $m > n + 1$  we encounter more equations than unknowns.

In Ex. 7.0.3 the same idea leads to the linear system

$$\begin{pmatrix} b_1(t_1) & \dots & b_n(t_1) \\ \vdots & & \vdots \\ b_1(t_m) & \dots & b_n(t_m) \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}, \quad (7.0.13)$$

with the same problem in case  $m > n$ .

Now we elaborate the common mathematical structure underlying the linear least squares fitting problem from Ex. 7.0.3 and Ex. 7.0.9.

(Full rank linear) **least squares problem**: [13, Sect. 4.2]

given:  $\mathbf{A} \in \mathbb{R}^{m,n}$ ,  $m, n \in \mathbb{N}$ ,  $m \geq n$ ,  $\text{rank}(\mathbf{A}) = n$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,

find:  $\mathbf{x} \in \mathbb{R}^n$  such that  $\|\mathbf{Ax} - \mathbf{b}\|_2 = \inf\{\|\mathbf{Ay} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbb{R}^n\}$  (7.0.14)

$$\begin{aligned} & \Downarrow \\ & \mathbf{x} = \underset{\mathbf{y} \in \mathbb{R}^n}{\text{argmin}} \|\mathbf{Ay} - \mathbf{b}\|_2 \end{aligned}$$

Sloppy notation for the minimization problem (7.0.14):

$$\|\mathbf{Ax} - \mathbf{b}\|_2 \rightarrow \min$$

Recast as linear least squares problem, *cf.* Rem. 7.0.11:

$$\text{Ex. 7.0.9: } \mathbf{A} = \begin{pmatrix} \mathbf{x}_1^\top & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^\top & 1 \end{pmatrix} \in \mathbb{R}^{m, n+1}, \quad \mathbf{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \in \mathbb{R}^m, \quad \mathbf{x} = \begin{pmatrix} \mathbf{a} \\ c \end{pmatrix} \in \mathbb{R}^{n+1}.$$

$$\text{Ex. 7.0.3: } \mathbf{A} = \begin{pmatrix} b_1(t_1) & \dots & b_n(t_1) \\ \vdots & & \vdots \\ b_1(t_m) & \dots & b_n(t_m) \end{pmatrix} \in \mathbb{R}^{m, n}, \quad \mathbf{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \in \mathbb{R}^m, \quad \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n.$$

In both cases the **residual norm**  $\|\mathbf{b} - \mathbf{Ax}\|_2$  allows to gauge the quality of the model.

What if  $\text{rank } \mathbf{A} < n$  in (7.0.14) ?

$$\mathbf{A} \in \mathbb{R}^{m, n}, \quad m \geq n, \quad \text{rank } \mathbf{A} < n \quad \xRightarrow{[48, \text{Sect. 6.1}]} \quad \exists \mathbf{z} \neq 0: \mathbf{Az} = 0. \quad (7.0.15)$$

► Solution of (7.0.14) cannot be unique!

➤ We need another condition to single out a unique minimizer: **minimum norm condition**

(General linear) **least squares problem**:

given:  $\mathbf{A} \in \mathbb{R}^{m,n}$ ,  $m, n \in \mathbb{N}$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,

find:  $\mathbf{x} \in \mathbb{R}^n$  such that

$$(i) \quad \|\mathbf{Ax} - \mathbf{b}\|_2 = \inf\{\|\mathbf{Ay} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbb{R}^n\}, \quad (7.0.16)$$

(ii)  $\|\mathbf{x}\|_2$  is minimal under the condition (i).

**Lemma 7.0.17** (Existence & uniqueness of solutions of the least squares problem).

*The least squares problem for  $\mathbf{A} \in \mathbb{K}^{m,n}$ ,  $\mathbf{A} \neq 0$ , has a unique solution for every  $\mathbf{b} \in \mathbb{K}^m$ .*

*Proof.* The proof is given by formula (7.2.6) and its derivation, see Sect. 7.2. □

MATLAB “black-box” solver for general linear least squares problems (7.0.16)

$\mathbf{x} = \mathbf{A} \backslash \mathbf{b}$  (“backslash”) solves (7.0.16) for  $\mathbf{A} \in \mathbb{K}^{m,n}$ ,  $m \neq n$ .

Reassuring: stable ( $\rightarrow$  Def.2.5.11) implementation (for dense matrices).

*Remark 7.0.18* (Rank defect in linear least squares problems).

Consider linear least squares fitting problem (7.0.5) with  $\mathbf{A} \in \mathbb{R}^{m,n}$  from (7.0.13).

$\text{rank}(\mathbf{A}) < n \Leftrightarrow$  columns of  $\mathbf{A}$  are linearly dependent

- $\Leftrightarrow$  at least one of the basis functions  $b_j$  is **redundant**, because it can not be distinguished from a linear combination of some others on  $\{t_i\}$
- $\triangleright$  too many basis functions  $\leftrightarrow$  too few data points

Usually  $\text{rank}(\mathbf{A}) < n$  in linear fitting hints at inadequate modelling



*Remark 7.0.19 (Pseudoinverse).* → [35, Ch. 12]

By Lemma 7.0.17 the solution operator of the least squares problem (7.0.16) defines a linear mapping  $\mathbf{b} \mapsto \mathbf{x}$ , which has a matrix representation.

*Definition 7.0.20 (Pseudoinverse).* The **pseudoinverse**  $\mathbf{A}^+ \in \mathbb{K}^{n,m}$  of  $\mathbf{A} \in \mathbb{K}^{m,n}$  is the matrix representation of the (linear) solution operator  $\mathbb{R}^m \mapsto \mathbb{R}^n$ ,  $\mathbf{b} \mapsto \mathbf{x}$  of the least squares problem (7.0.16)  $\|\mathbf{Ax} - \mathbf{b}\| \rightarrow \min, \|\mathbf{x}\| \rightarrow \min$ .

MATLAB:

`P = pinv(A)` computes the pseudoinverse.



*Remark 7.0.21 (Conditioning of the least squares problem).* → [13, Sect. 4.3]

*Definition 7.0.22* (Generalized condition (number) of a matrix,  $\rightarrow$  Def. 2.5.26).

Let  $\sigma_1 \geq \sigma_2 \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_p = 0$ ,  $p := \min\{m, n\}$ , be the singular values ( $\rightarrow$  Def. 6.5.8) of  $\mathbf{A} \in \mathbb{K}^{m,n}$ . Then

$$\text{cond}_2(\mathbf{A}) := \frac{\sigma_1}{\sigma_r}$$

is the *generalized condition (number)* (w.r.t. the 2-norm) of  $\mathbf{A}$ .

*Theorem 7.0.23.* For  $m \geq n$ ,  $\mathbf{A} \in \mathbb{K}^{m,n}$ ,  $\text{rank}(\mathbf{A}) = n$ , let  $\mathbf{x} \in \mathbb{K}^n$  be the solution of the least squares problem  $\|\mathbf{Ax} - \mathbf{b}\| \rightarrow \min$  and  $\hat{\mathbf{x}}$  the solution of the perturbed least squares problem  $\|(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} - \mathbf{b}\| \rightarrow \min$ . Then

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2} \leq \left( 2 \text{cond}_2(\mathbf{A}) + \text{cond}_2^2(\mathbf{A}) \frac{\|\mathbf{r}\|_2}{\|\mathbf{A}\|_2 \|\mathbf{x}\|_2} \right) \frac{\|\Delta\mathbf{A}\|_2}{\|\mathbf{A}\|_2}$$

holds, where  $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$  is the *residual*.

This means: if  $\|\mathbf{r}\|_2 \ll 1$   $\blacktriangleright$  condition of the least squares problem  $\approx \text{cond}_2(\mathbf{A})$   
if  $\|\mathbf{r}\|_2$  “large”  $\blacktriangleright$  condition of the least squares problem  $\approx \text{cond}_2^2(\mathbf{A})$

# 7.1 Normal Equations [13, Sect. 4.2], [35, Ch. 11]

Setting (7.0.14):  $\mathbf{A} \in \mathbb{R}^{m,n}$ ,  $m \geq n$ , with full rank  $\text{rank}(\mathbf{A}) = n$ :

$$\|\mathbf{Ax} - \mathbf{b}\|_2 \rightarrow \min .$$

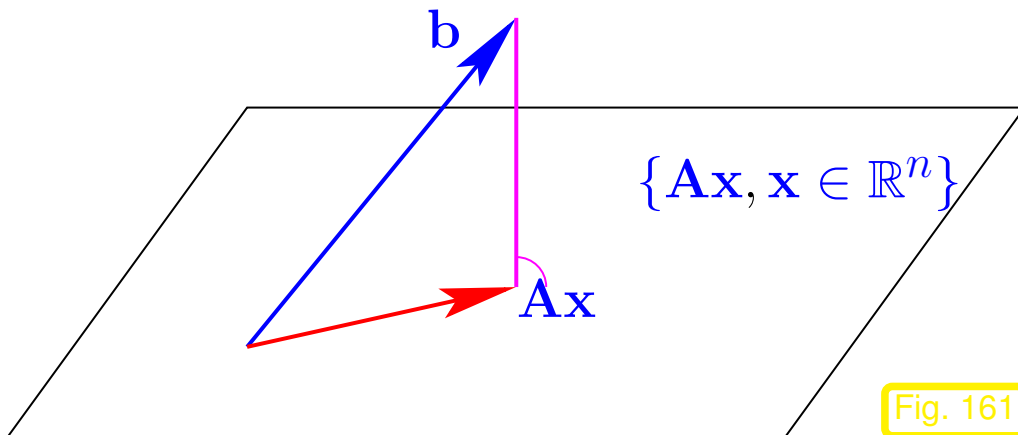


Fig. 161

Geometric interpretation of linear least squares problem (7.0.16):

$\hat{\mathbf{x}} \hat{=}$  orthogonal projection of  $\mathbf{b}$  on the subspace  $\text{Im}(\mathbf{A}) := \text{Span} \{(\mathbf{A})_{:,1}, \dots, (\mathbf{A})_{:,n}\}$ .



Geometric interpretation: the least squares problem (7.0.16) amounts to searching the point  $\mathbf{p} \in \text{Im}(\mathbf{A})$  nearest (w.r.t. Euclidean distance) to  $\mathbf{b} \in \mathbb{R}^m$ .

Geometric intuition, see Fig. 161:  $\mathbf{p}$  is the orthogonal projection of  $\mathbf{b}$  onto  $\text{Im}(\mathbf{A})$ , that is  $\mathbf{b} - \mathbf{p} \perp \text{Im}(\mathbf{A})$ . Note the equivalence

$$\mathbf{b} - \mathbf{p} \perp \text{Im}(\mathbf{A}) \Leftrightarrow \mathbf{b} - \mathbf{p} \perp (\mathbf{A})_{:,j}, \quad j = 1, \dots, n \Leftrightarrow \mathbf{A}^H(\mathbf{b} - \mathbf{p}) = 0,$$

Representation  $\mathbf{p} = \mathbf{A}\mathbf{x}$  leads to normal equations (7.1.2). General discussion in [13, Sect. 4.6].

Solve (7.0.16) for  $\mathbf{b} \in \mathbb{R}^m$

$$\mathbf{x} \in \mathbb{R}^n: \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \rightarrow \min \Leftrightarrow f(\mathbf{x}) := \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \rightarrow \min. \quad (7.1.1)$$

A quadratic functional, cf. Sect. 5.1.1, (5.1.4)

$$f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 = \mathbf{x}^H(\mathbf{A}^H\mathbf{A})\mathbf{x} - 2\mathbf{b}^H\mathbf{A}\mathbf{x} + \mathbf{b}^H\mathbf{b}.$$

Minimization problem for  $f$   $\triangleright$  study gradient, cf. (5.1.9)


$$\text{grad } f(\mathbf{x}) = 2(\mathbf{A}^H \mathbf{A})\mathbf{x} - 2\mathbf{A}^H \mathbf{b} .$$

$$\text{grad } f(\mathbf{x}) \stackrel{!}{=} 0: \quad \boxed{\mathbf{A}^H \mathbf{A} \mathbf{x} = \mathbf{A}^H \mathbf{b}} \quad = \text{normal equation of (7.1.1)}$$

Notice:  $\text{rank}(\mathbf{A}) = n \Rightarrow \mathbf{A}^H \mathbf{A} \in \mathbb{R}^{n,n}$  s.p.d. ( $\rightarrow$  Def. 2.7.9, [13, Rem. 4.6])

*Remark 7.1.3* (Conditioning of normal equations). [13, pp. 128]

Cauton: danger of instability, with SVD  $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$



$$\text{cond}_2(\mathbf{A}^H \mathbf{A}) = \text{cond}_2(\mathbf{V}\Sigma^H\mathbf{U}^H\mathbf{U}\Sigma\mathbf{V}^H) = \text{cond}_2(\Sigma^H\Sigma) = \frac{\sigma_1^2}{\sigma_n^2} = \text{cond}_2(\mathbf{A})^2 .$$

$\triangleright$  For fairly ill-conditioned  $\mathbf{A}$  using the normal equations (7.1.2) to solve the linear least squares problem (7.1.1) numerically may run the risk of huge amplification of roundoff errors incurred during the computation of the right hand side  $\mathbf{A}^H \mathbf{b}$ : **potential instability** ( $\rightarrow$  Def. 2.5.11) of normal equation approach.

Example 7.1.4 (Instability of normal equations). → [13, Ex. 4.12]

Caution: loss of information in the computation of  $\mathbf{A}^H \mathbf{A}$ , e.g.



$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ \delta & 0 \\ 0 & \delta \end{pmatrix}$$

$$\Rightarrow \mathbf{A}^H \mathbf{A} = \begin{pmatrix} 1 + \delta^2 & 1 \\ 1 & 1 + \delta^2 \end{pmatrix}$$

```

1 >> A = [1 1; ...
2         sqrt(eps) 0; ...
3         0 sqrt(eps)];
4 >> rank(A)
5     ans = 2
6 >> rank(A' * A)
7     ans = 1


```

If  $\delta < \sqrt{\text{eps}}$   $\Rightarrow 1 + \delta^2 = 1$  in  $\mathbb{M}$ , i.e.  $\mathbf{A}^H \mathbf{A}$  “numeric singular”, though  $\text{rank}(\mathbf{A}) = 2$ , see Sect. 2.4, in particular Rem. 2.4.13.

*Remark 7.1.5* (Loss of sparsity when forming normal equations).

Another *reason not to compute*  $\mathbf{A}^H \mathbf{A}$ , when both  $m, n$  large:

$$\mathbf{A} \text{ sparse} \not\Rightarrow \mathbf{A}^T \mathbf{A} \text{ sparse}$$

- ▶ • Potential memory overflow, when computing  $\mathbf{A}^T \mathbf{A}$
- Squanders possibility to use efficient sparse direct elimination techniques, see Sect. 2.6.3 

*Remark 7.1.6* (Extended normal equations).

A way to avoid the computation of  $\mathbf{A}^H \mathbf{A}$ :

Extend normal equations (7.1.2): introduce **residual**  $\mathbf{r} := \mathbf{Ax} - \mathbf{b}$  as new unknown:

$$\mathbf{A}^H \mathbf{Ax} = \mathbf{A}^H \mathbf{b} \quad \Leftrightarrow \quad \mathbf{B} \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \end{pmatrix} := \begin{pmatrix} -\mathbf{I} & \mathbf{A} \\ \mathbf{A}^H & 0 \end{pmatrix} \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix}. \quad (7.1.7)$$

More general substitution  $\mathbf{r} := \alpha^{-1}(\mathbf{Ax} - \mathbf{b})$ ,  $\alpha > 0$  to improve the conditioning:

$$\mathbf{A}^H \mathbf{Ax} = \mathbf{A}^H \mathbf{b} \quad \Leftrightarrow \quad \mathbf{B}_\alpha \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \end{pmatrix} := \begin{pmatrix} -\alpha \mathbf{I} & \mathbf{A} \\ \mathbf{A}^H & 0 \end{pmatrix} \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix}. \quad (7.1.8)$$

For  $m, n \gg 1$ ,  $\mathbf{A}$  sparse, both (7.1.7) and (7.1.8) lead to large sparse linear systems of equations, amenable to sparse direct elimination techniques, see Sect. 2.6.3.

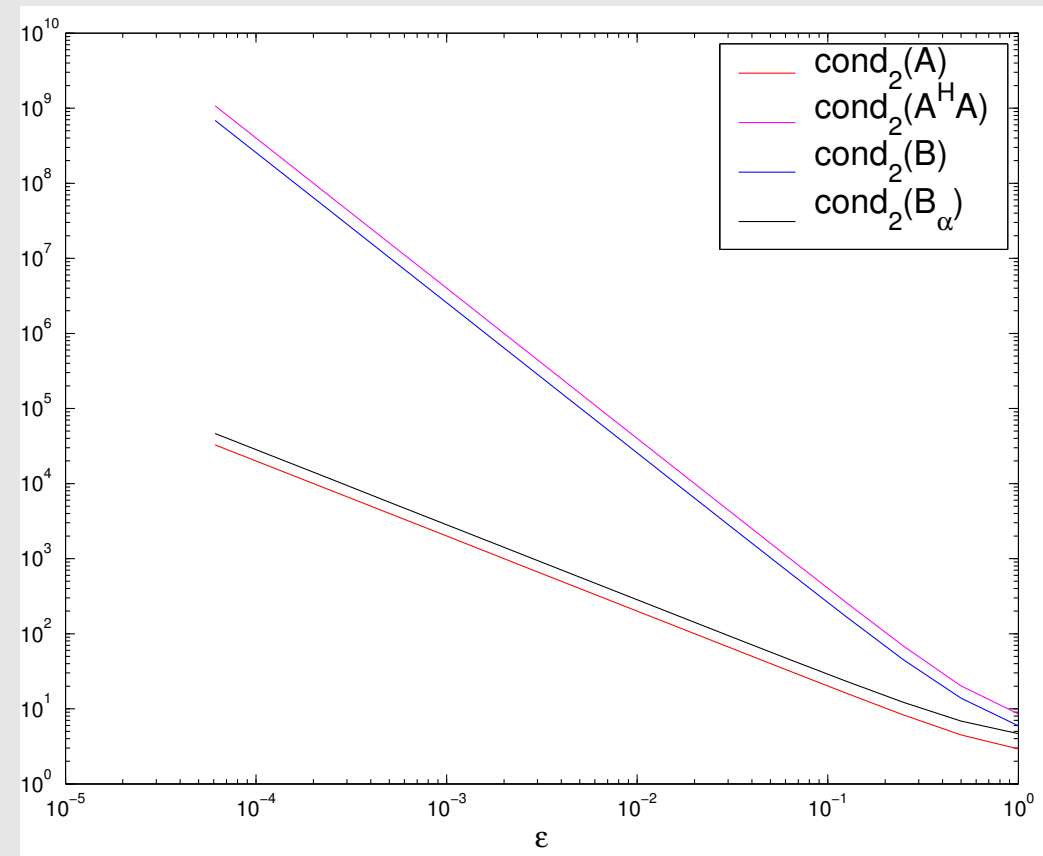


*Example 7.1.9* (Conditioning of the extended normal equations).

Consider (7.1.7), (7.1.8) for

$$\mathbf{A} = \begin{pmatrix} 1 + \epsilon & 1 \\ 1 - \epsilon & 1 \\ \epsilon & \epsilon \end{pmatrix}.$$

Plot of different condition numbers  
in dependence on  $\epsilon$   
( $\alpha = \epsilon \|\mathbf{A}\|_2 / \sqrt{2}$ )



## 7.2 Orthogonal Transformation Methods [13, Sect. 4.4.2]

We consider the linear least squares problem (7.0.14)

$$\text{given } \mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{b} \in \mathbb{R}^m \text{ find } \mathbf{x} = \underset{\mathbf{y} \in \mathbb{R}^n}{\operatorname{argmin}} \|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2. \quad (7.0.14)$$

Assumption:  $m \geq n$  and  $\mathbf{A}$  has full (maximum) rank:  $\operatorname{rank}(\mathbf{A}) = n$ .

Recall: rationale behind Gaussian elimination ( $\rightarrow$  Sect. 2.1, Ex. 2.1.1)

$\rightarrow$  Row transformations of LSE  $\mathbf{A}\mathbf{x} = \mathbf{b}$  to *equivalent* (in terms of solution) LSE  $\mathbf{U}\mathbf{x} = \tilde{\mathbf{b}}$ , which is easier to solve because it has triangular form.

How to adapt this policy to linear least squares problem (7.0.16) ?

- Two questions:
- ❶ Which linear least squares problems are “easy to solve” ?
  - ❷ How do we get them by *equivalent transformations* of (7.0.14) ?

Answer to question ❶ (same as for LSE):

Linear least squares problems (7.0.14) with upper *triangular*  $\mathbf{A}$  are easy to solve!

$$\left\| \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} - \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \right\|_2 \rightarrow \min \stackrel{(*)}{\Rightarrow} \mathbf{x} = \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

How can we draw the conclusion  $(*)$ ? Obviously, the components  $n + 1, \dots, m$  of the vector inside the norm are fixed and do not depend on  $\mathbf{x}$ . All we can do is to make the first components  $1, \dots, n$  vanish, by choosing a suitable  $\mathbf{x}$ , see [13, Thm. 4.13].

Note: since  $\mathbf{A}$  has full rank  $n$  the triangular part  $\mathbf{R} \in \mathbb{R}^{n,n}$  of  $\mathbf{A}$  is regular!

Answer to question ②:

Recall Thm. 2.8.6: orthogonal (unitary) transformations ( $\rightarrow$  Def. 2.8.5) leave 2-norm invariant.



Idea: Transformation of  $\mathbf{Ax} - \mathbf{b}$  to simpler form by *orthogonal* row transformations:

$$\operatorname{argmin}_{\mathbf{y} \in \mathbb{R}^n} \|\mathbf{Ay} - \mathbf{b}\|_2 = \operatorname{argmin}_{\mathbf{y} \in \mathbb{R}^n} \|\tilde{\mathbf{A}}\mathbf{y} - \tilde{\mathbf{b}}\|_2 ,$$

where  $\tilde{\mathbf{A}} = \mathbf{QA}$  ,  $\tilde{\mathbf{b}} = \mathbf{Qb}$  with *orthogonal*  $\mathbf{Q} \in \mathbb{R}^{m,m}$  .

equivalence !

As in the case of LSE ( $\rightarrow$  Sect. 2.8): “simpler form” = triangular form, see ❶ above.

Is orthogonal transformation to upper triangular form always possible ?

Recall: **Gram-Schmidt orthonormalization** ( $\rightarrow$  Sect. 6.3.4.1)

A close scrutiny of (6.3.87), (6.3.88) and Code 6.3.88 (with roles of  $m$  and  $n$  swapped) reveals

$$\begin{aligned} \mathbf{q}_1 &= t_{11}\mathbf{a}_1 \\ \mathbf{q}_2 &= t_{12}\mathbf{a}_1 + t_{22}\mathbf{a}_2 \\ \mathbf{q}_3 &= t_{13}\mathbf{a}_1 + t_{23}\mathbf{a}_2 + t_{33}\mathbf{a}_3 \\ &\vdots \\ \mathbf{q}_n &= t_{1n}\mathbf{a}_1 + t_{2n}\mathbf{a}_2 + \cdots + t_{nn}\mathbf{a}_n . \end{aligned}$$



$$\blacktriangleright \quad \mathbf{A} = \tilde{\mathbf{Q}} \begin{pmatrix} \mathbf{R} \\ 0 \end{pmatrix} \Leftrightarrow \begin{pmatrix} \mathbf{A} \end{pmatrix} = \begin{pmatrix} \mathbf{Q} \end{pmatrix} \begin{pmatrix} \mathbf{R} \\ 0 \end{pmatrix}$$

$$\Leftrightarrow \tilde{\mathbf{Q}}^\top \mathbf{A} = \begin{pmatrix} \mathbf{R} \\ 0 \end{pmatrix} .$$

Finally, let the columns of  $\mathbf{A}$  play the roles of the  $\mathbf{a}_i$  in Gram-Schmidt orthonormalization.

Concrete realization of the “equivalent orthonormal transformation to upper triangular form”-idea by means of **QR-decomposition** ([13, Sect. 4.4.2], recall Sect. 2.8).

**QR-decomposition:**  $\mathbf{A} = \mathbf{Q}\mathbf{R}$ ,  $\mathbf{Q} \in \mathbb{K}^{m,m}$  unitary,  $\mathbf{R} \in \mathbb{K}^{m,n}$  (regular) upper triangular matrix.

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \left\| \mathbf{Q}(\mathbf{R}\mathbf{x} - \mathbf{Q}^H\mathbf{b}) \right\|_2 = \left\| \mathbf{R}\mathbf{x} - \tilde{\mathbf{b}} \right\|_2, \quad \tilde{\mathbf{b}} := \mathbf{Q}^H\mathbf{b} .$$

$$\| \mathbf{Ax} - \mathbf{b} \|_2 \rightarrow \min \Leftrightarrow \left\| \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} - \begin{pmatrix} \tilde{b}_1 \\ \vdots \\ \tilde{b}_m \end{pmatrix} \right\|_2 \rightarrow \min .$$

$$\mathbf{x} = \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} \tilde{b}_1 \\ \vdots \\ \tilde{b}_n \end{pmatrix}, \quad \text{residuum } \mathbf{r} = \mathbf{Q} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \tilde{b}_{n+1} \\ \vdots \\ \tilde{b}_m \end{pmatrix} .$$

Note: by Thm. 2.8.6 residual norm readily available  $\|\mathbf{r}\|_2 = \sqrt{\tilde{b}_{n+1}^2 + \cdots + \tilde{b}_m^2}$ .

### Code 7.2.1: QR-based solver for full rank linear least squares problem (7.0.14)

```

1 function [x,res] = qr_qls_solve(A,b)
2 % Solution of linear least squares problem (7.0.14) by means of QR-decomposition
3 % Note:  $\mathbf{A} \in \mathbb{R}^{m,n}$  with  $m > n$ ,  $\text{rank}(\mathbf{A}) = n$  is assumed
4 [m,n] = size (A) ;
5 R = triu (qr ([A,b],0) ), % economical QR-decomposition of extended matrix
6 x = R(1:n,1:n) \ R(1:n,n+1); %  $\hat{\mathbf{x}} = (\mathbf{R})_{1:n,1:n}^{-1} (\mathbf{Q}^T \mathbf{b})_{1:n}$ 
7 res = R(n+1,n+1); % =  $\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_2$  (why ?)

```

Discussion of (some) details of implementation in Code 7.2.0:

- Gram-Schmidt orthogonalization is done in a numerically stable way by the MATLAB command `qr`; the call `triu(qr(A))`,  $\mathbf{A} \in \mathbb{K}^{m,n}$  with full rank,  $m \geq n$ , returns the (generalized) upper triangular matrix  $\mathbf{Q}^H \mathbf{A} \in \mathbb{K}^{m,n}$ , where  $\mathbf{Q}$  is an orthogonal matrix, whose first  $n$  columns agree with the vectors produced by Gram-Schmidt orthonormalization of the columns of  $\mathbf{A}$ .
- Asymptotic computational cost of `qr(A)`:  $O(mn^2)$

- Algorithmically, the orthogonal/unitary transformations are effected by successive orthogonal row transformations (by means of Householder reflections (2.8.8) for general matrices, and Givens rotations (2.8.10) for banded matrices, see Sect. 2.8 for details).
- The orthogonal/unitary transformation with  $\mathbf{Q}^H$  is applied to the augmented matrix  $(\mathbf{A}, \mathbf{b}) \in \mathbb{R}^{m, n+1}$  (Line 5 of Code 7.2.0), which is transformed into  $(\mathbf{R}, \tilde{\mathbf{b}})$ . Thus, the matrix  $\mathbf{Q}$  *need not be stored* to compute the transformed vector  $\tilde{\mathbf{b}} = \mathbf{Q}^H \mathbf{b}$ . Same idea is used for Gaussian elimination, see Code 2.1.6, Alg. 2.1.3, and also Alg. 2.8.22.
- Line 7: note that  $(\mathbf{Q}^H(\mathbf{A}, \mathbf{b}))_{n+2:m,:} = 0$  so that only the first  $n + 1$  components of  $\tilde{\mathbf{b}}$  do not vanish.

► A QR-based algorithm is implemented in the least-squares-solver of the MATLAB-operator “\” (for dense matrices).

Alternative: Solving linear least squares problem (7.0.16) by SVD ( $\rightarrow$  Def. 6.5.8)

Most general setting:  $\mathbf{A} \in \mathbb{K}^{m, n}$ ,  $\text{rank}(\mathbf{A}) = r \leq \min\{m, n\}$ :

Here we drop the assumption of full rank of  $\mathbf{A}$ . This means that condition (ii) in the definition (7.0.16) of a linear least squares problem may be required for singling out a unique solution.

SVD:  $\mathbf{A} = [\mathbf{U}_1 \quad \mathbf{U}_2] \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{pmatrix}$

(7.2.2)

with  $\mathbf{U}_1 \in \mathbb{K}^{m,r}$ ,  $\mathbf{U}_2 \in \mathbb{K}^{m,m-r}$  with *orthonormal* columns,  
 $\mathbf{\Sigma}_r = \text{diag}(\sigma_1, \dots, \sigma_r) \in \mathbb{R}^{r,r}$  (singular values, Def. 6.5.8),  
 $\mathbf{V}_1 \in \mathbb{K}^{n,r}$ ,  $\mathbf{V}_2 \in \mathbb{K}^{n,n-r}$  with *orthonormal* columns.

Then we use the invariance of the 2-norm of a vector with respect to multiplication with  $\mathbf{U} = [\mathbf{U}_1, \mathbf{U}_2]$ , see Thm. 2.8.6, together with the fact that  $\mathbf{U}$  is unitary, see Def. 2.8.5:

$$[\mathbf{U}_1, \mathbf{U}_2] \cdot \begin{bmatrix} \mathbf{U}_1^H \\ \mathbf{U}_2^H \end{bmatrix} = \mathbf{I}.$$

$$\|\mathbf{Ax} - \mathbf{b}\|_2 = \left\| [\mathbf{U}_1 \ \mathbf{U}_2] \begin{pmatrix} \mathbf{\Sigma}_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{pmatrix} \mathbf{x} - \mathbf{b} \right\|_2 = \left\| \begin{pmatrix} \mathbf{\Sigma}_r \mathbf{V}_1^H \mathbf{x} \\ 0 \end{pmatrix} - \begin{pmatrix} \mathbf{U}_1^H \mathbf{b} \\ \mathbf{U}_2^H \mathbf{b} \end{pmatrix} \right\|_2 \quad (7.2.3)$$



Logical strategy: choose  $\mathbf{x}$  such that the first  $r$  components of  $\begin{pmatrix} \Sigma_r \mathbf{V}_1^H \mathbf{x} \\ 0 \end{pmatrix} - \begin{pmatrix} \mathbf{U}_1^H \mathbf{b} \\ \mathbf{U}_2^H \mathbf{b} \end{pmatrix}$  vanish:

$$\triangleright \text{ (possibly underdetermined) } r \times n \text{ linear system } \quad \Sigma_r \mathbf{V}_1^H \mathbf{x} = \mathbf{U}_1^H \mathbf{b} . \quad (7.2.4)$$

To fix a unique solution in the case  $r < n$  we appeal to the **minimal norm condition** in (7.0.16): solution  $\mathbf{x}$  of (7.2.4) is unique up to contributions from

$$\text{Ker}(\mathbf{V}_1^H) = \text{Im}(\mathbf{V}_1)^\perp = \text{Im}(\mathbf{V}_2) . \quad (7.2.5)$$

Since  $\mathbf{V}$  is unitary, the minimal norm solution is obtained by setting contributions from  $\text{Im}(\mathbf{V}_2)$  to zero, which amounts to choosing  $\mathbf{x} \in \text{Im}(\mathbf{V}_1)$ . This converts (7.2.4) into

$$\Sigma_r \underbrace{\mathbf{V}_1^H \mathbf{V}_1}_{=\mathbf{I}} \mathbf{z} = \mathbf{U}_1^H \mathbf{b} \quad \Rightarrow \quad \mathbf{z} = \Sigma_r^{-1} \mathbf{U}_1^H \mathbf{b} .$$

$$\blacktriangleright \text{ solution } \boxed{\mathbf{x} = \mathbf{V}_1 \Sigma_r^{-1} \mathbf{U}_1^H \mathbf{b}} , \quad \|\mathbf{r}\|_2 = \left\| \mathbf{U}_2^H \mathbf{b} \right\|_2 . \quad (7.2.6)$$

## Code 7.2.7: Solving LSQ problem via SVD

```

1 function y = lsqsvd(A,b)
2 [U,S,V] = svd(A,0);
3 sv = diag(S);
4 r = max(find(sv/sv(1) > eps));
5 y = V(:,1:r) * (diag(1./sv(1:r)) * ...
6           (U(:,1:r)' * b));

```

Practical implementation:

“numerical rank” test:

$$r = \max\{i: \sigma_i/\sigma_1 > \text{tol}\}$$



*Remark 7.2.8* (Pseudoinverse and SVD). → Rem. 7.0.19, [35, Ch. 12], [13, Sect. 4.7]

The solution formula (7.2.6) directly yields a representation of the pseudoinverse  $\mathbf{A}^+$  (→ Def. 7.0.20) of any matrix  $\mathbf{A}$ :

*Theorem 7.2.9* (Pseudoinverse and SVD).

If  $\mathbf{A} \in \mathbb{K}^{m,n}$  has the SVD decomposition (7.2.2), then  $\mathbf{A}^+ = \mathbf{V}_1 \Sigma_r^{-1} \mathbf{U}_1^H$  holds.

*Remark 7.2.10* (Normal equations vs. orthogonal transformations method).

Superior numerical stability ( $\rightarrow$  Def. 2.5.11) of orthogonal transformations methods:

- ▶ Use orthogonal transformations methods for least squares problems (7.0.16), whenever  $\mathbf{A} \in \mathbb{R}^{m,n}$  *dense* and  $n$  small.

SVD/QR-factorization cannot exploit sparsity:

- ▶ Use normal equations in the expanded form (7.1.7)/(7.1.8), when  $\mathbf{A} \in \mathbb{R}^{m,n}$  *sparse* ( $\rightarrow$  Def. 2.6.1) and  $m, n$  big.



*Example 7.2.11* (Fit of hyperplanes).

This example studies the power and versatility of orthogonal transformations in the context of (generalized) least squares minimization problems.

The **Hesse normal form** of a hyperplane  $\mathcal{H}$  (= affine subspace of dimension  $d - 1$ ) in  $\mathbb{R}^d$  is:

$$\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^d: c + \mathbf{n}^\top \mathbf{x} = 0\}, \quad \|\mathbf{n}\|_2 = 1. \quad (7.2.12)$$

► Euclidean distance of  $\mathbf{y} \in \mathbb{R}^d$  from the plane:  $\text{dist}(\mathcal{H}, \mathbf{y}) = |c + \mathbf{n}^\top \mathbf{y}|. \quad (7.2.13)$

Goal: given the points  $\mathbf{y}_1, \dots, \mathbf{y}_m, m > d$ , find  $\mathcal{H} \leftrightarrow \{c \in \mathbb{R}, \mathbf{n} \in \mathbb{R}^d, \|\mathbf{n}\|_2 = 1\}$ , such that

$$\sum_{j=1}^m \text{dist}(\mathcal{H}, \mathbf{y}_j)^2 = \sum_{j=1}^m |c + \mathbf{n}^\top \mathbf{y}_j|^2 \rightarrow \min. \quad (7.2.14)$$

Note: (7.2.14)  $\neq$  linear least squares problem due to **constraint**  $\|\mathbf{n}\|_2 = 1$ .

$$(7.2.14) \Leftrightarrow \left\| \underbrace{\begin{pmatrix} 1 & y_{1,1} & \cdots & y_{1,d} \\ 1 & y_{2,1} & \cdots & y_{2,d} \\ \vdots & \vdots & & \vdots \\ 1 & y_{m,1} & \cdots & y_{m,d} \end{pmatrix}}_{=: \mathbf{A}} \begin{pmatrix} c \\ n_1 \\ \vdots \\ n_d \end{pmatrix} \right\|_2 \rightarrow \min \quad \text{under constraint} \quad \|\mathbf{n}\|_2 = 1.$$

Step ❶: QR-decomposition (→ Section 2.8)

$$\mathbf{A} := \begin{pmatrix} 1 & y_{1,1} & \cdots & y_{1,d} \\ 1 & y_{2,1} & \cdots & y_{2,d} \\ \vdots & \vdots & & \vdots \\ 1 & y_{m,1} & \cdots & y_{m,d} \end{pmatrix} = \mathbf{Q}\mathbf{R} \quad , \quad \mathbf{R} := \begin{pmatrix} r_{11} & r_{12} & \cdots & \cdots & r_{1,d+1} \\ 0 & r_{22} & \cdots & \cdots & r_{2,d+1} \\ \vdots & & \ddots & & \vdots \\ 0 & & & & r_{d+1,d+1} \\ 0 & \cdots & & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{m,d+1} .$$

$$\|\mathbf{A}\mathbf{x}\|_2 \rightarrow \min \quad \Leftrightarrow \quad \|\mathbf{R}\mathbf{x}\|_2 = \left\| \begin{pmatrix} r_{11} & r_{12} & \cdots & \cdots & r_{1,d+1} \\ 0 & r_{22} & \cdots & \cdots & r_{2,d+1} \\ \vdots & & \ddots & & \vdots \\ 0 & & & & r_{d+1,d+1} \\ 0 & \cdots & & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & & \cdots & 0 \end{pmatrix} \begin{pmatrix} c \\ n_1 \\ \vdots \\ n_d \end{pmatrix} \right\|_2 \rightarrow \min . \quad (7.2.15)$$

Step ❷ Note that necessarily (why?)

$$c \cdot r_{11} + n_1 \cdot r_{12} + \cdots + r_{1,d+1} \cdot n_d = 0 .$$

This insight converts (7.2.15) to

$$\left\| \begin{pmatrix} r_{22} & r_{23} & \cdots & \cdots & r_{2,d+1} \\ 0 & r_{33} & \cdots & \cdots & r_{3,d+1} \\ \vdots & & \ddots & & \vdots \\ 0 & & & & r_{d+1,d+1} \end{pmatrix} \begin{pmatrix} n_1 \\ \vdots \\ \vdots \\ n_d \end{pmatrix} \right\|_2 \rightarrow \min, \quad \|\mathbf{n}\|_2 = 1. \quad (7.2.16)$$

(7.2.16) = problem of type (6.5.32), minimization on the Euclidean sphere.

➤ Solve (7.2.16) using SVD !

Note: Since  $r_{11} = \|(\mathbf{A})_{:,1}\|_2 = \sqrt{m} \neq 0 \Rightarrow c = -r_{11}^{-1} \sum_{j=1}^d r_{1,j+1} n_j$ .

Code 7.2.17: (Generalized) distance fitting of a hyperplane: solution of (7.2.18)

```

1 function [c,n] = clsq(A,dim);
2 % Solves constrained linear least squares problem (7.2.18) with dim passing d
3 [m,p] = size (A);
4 if p < dim+1, error ('not enough unknowns'); end;
5 if m < dim, error ('not enough equations'); end;
6 m = min (m, p);
7 R = triu (qr (A)); % First step: orthogonal transformation, see Code 7.2.0
8 [U,S,V] = svd (R(p-dim+1:m,p-dim+1:p)); % Solve (7.2.16)
9 n = V(:,dim);
10 c = -R(1:p-dim,1:p-dim)\R(1:p-dim,p-dim+1:p)*n;

```

Code 7.2.18 solves the general problem: For  $\mathbf{A} \in \mathbb{K}^{m,n}$  find  $\mathbf{n} \in \mathbb{R}^d$ ,  $\mathbf{c} \in \mathbb{R}^{n-d}$  such that

$$\left\| \mathbf{A} \begin{pmatrix} \mathbf{c} \\ \mathbf{n} \end{pmatrix} \right\|_2 \rightarrow \min \quad \text{with constraint } \|\mathbf{n}\|_2 = 1. \quad (7.2.18)$$



## 7.3 Total Least Squares

Given: overdetermined linear system of equations  $\mathbf{Ax} = \mathbf{b}$ ,  $\mathbf{A} \in \mathbb{R}^{m,n}$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,  $m \geq n$ .

Known: LSE solvable  $\Leftrightarrow \mathbf{b} \in \text{Im}(\mathbf{A})$ , if  $\mathbf{A}$ ,  $\mathbf{b}$  were not perturbed,

but  $\mathbf{A}$ ,  $\mathbf{b}$  are perturbed (measurement errors).

Sought: Solvable overdetermined system of equations  $\hat{\mathbf{A}}\mathbf{x} = \hat{\mathbf{b}}$ ,  $\hat{\mathbf{A}} \in \mathbb{R}^{m,n}$ ,  $\hat{\mathbf{b}} \in \mathbb{R}^m$ ,  
“nearest” to  $\mathbf{Ax} = \mathbf{b}$ .

☞ least squares problem “turned upside down”: now we are allowed to tamper with system matrix and right hand side vector!

Total least squares problem:

Given:  $\mathbf{A} \in \mathbb{R}^{m,n}$ ,  $m \geq n$ ,  $\text{rank}(\mathbf{A}) = n$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,

find:  $\hat{\mathbf{A}} \in \mathbb{R}^{m,n}$ ,  $\hat{\mathbf{b}} \in \mathbb{R}^m$  with

$$\left\| \underbrace{\begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix}}_{=: \mathbf{C}} - \underbrace{\begin{bmatrix} \hat{\mathbf{A}} & \hat{\mathbf{b}} \end{bmatrix}}_{=: \hat{\mathbf{C}}} \right\|_F \rightarrow \min, \quad \hat{\mathbf{b}} \in \text{Im}(\hat{\mathbf{A}}).$$

(7.3.1)

$$\hat{\mathbf{b}} \in \text{Im}(\hat{\mathbf{A}}) \Rightarrow \text{rank}(\hat{\mathbf{C}}) = n \quad \blacktriangleright \quad (7.3.1) \Rightarrow \min_{\text{rank}(\hat{\mathbf{C}})=n} \left\| \mathbf{C} - \hat{\mathbf{C}} \right\|_F.$$

R. Hiptmair  
rev 38286,  
November  
18, 2011

☞  $\hat{\mathbf{C}}$  is the rank- $n$  best approximation of  $\mathbf{C}$ !

Thm. 6.5.37  $\blacktriangleright$  use the SVD decomposition of  $\mathbf{C}$  to construct  $\hat{\mathbf{C}}$ :

$$\mathbf{C} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H = \sum_{j=1}^{n+1} \sigma_j (\mathbf{U})_{:,j} (\mathbf{V})_{:,j}^H \xrightarrow{\text{Thm. 6.5.37}} \hat{\mathbf{C}} = \sum_{j=1}^n \sigma_j (\mathbf{U})_{:,j} (\mathbf{V})_{:,j}^H. \quad (7.3.2)$$



$$\mathbf{V} \text{ orthogonal} \implies \widehat{\mathbf{C}}(\mathbf{V})_{:,n+1} = 0. \quad (7.3.3)$$

Recall interpretation:  $\widehat{\mathbf{A}} = (\widehat{\mathbf{C}})_{1:n,1:n}$ ,  $\widehat{\mathbf{b}} = \widehat{\mathbf{C}}_{1:n,n+1}$

► (7.3.3) provides solution

$$\mathbf{x} := \widehat{\mathbf{A}}^{-1}\widehat{\mathbf{b}} = (\mathbf{V})_{:,n+1}/(\mathbf{V})_{n+1,n+1}. \quad (7.3.4)$$

### Code 7.3.5: Total least squares via SVD

```

1 function x = lsqtotal(A,b);
2 % computes only solution x of fitted consistent LSE
3 [m,n]=size(A);
4 [U, Sigma, V] = svd([A,b]); % see (7.3.2)
5 s = V(n+1,n+1);
6 if s == 0, error('No solution'); end
7 x = -V(1:n,n+1)/s; % see (7.3.4)

```

# 7.4 Constrained Least Squares

Sect. 3.1: interpolation ➤ linear system of equations (3.1.10),  
Ex. 7.0.3: linear data fitting ➤ linear least squares problem (7.0.5).

What if *some* data points are considered accurate?

- ▶ mix interpolation and linear least squares fitting!
- ▶ linear least squares problem with **linear constraint**

**Linear least squares problem with linear constraint:**

Given:  $\mathbf{A} \in \mathbb{R}^{m,n}$ ,  $m \geq n$ ,  $\text{rank}(\mathbf{A}) = n$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,  
 $\mathbf{C} \in \mathbb{R}^{p,n}$ ,  $p < n$ ,  $\text{rank}(\mathbf{C}) = p$ ,  $\mathbf{d} \in \mathbb{R}^p$

Find:  $\mathbf{x} \in \mathbb{R}^n$  with  $\|\mathbf{Ax} - \mathbf{b}\|_2 \rightarrow \min$ ,  $\mathbf{Cx} = \mathbf{d}$ .  
 (7.4.1)  
 Linear constraint

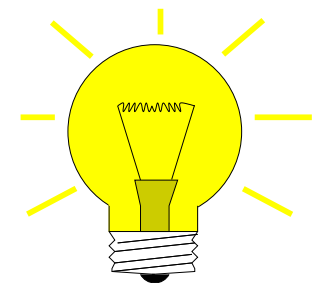
**7.4.1 Solution via normal equations**

Recall important technique from multidimensional calculus: Lagrange multipliers, see [63, Sect. 7.9].

Idea: coupling the constraint using the **Lagrange multiplier**  $\mathbf{m} \in \mathbb{R}^p$

$$\mathbf{x} = \underset{\mathbf{x} \in \mathbb{R}^n}{\text{argmin}} \max_{\mathbf{m} \in \mathbb{R}^p} L(\mathbf{x}, \mathbf{m}), \quad (7.4.2)$$

$$L(\mathbf{x}, \mathbf{m}) := \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|^2 + \mathbf{m}^H (\mathbf{Cx} - \mathbf{d}). \quad (7.4.3)$$



The simple heuristics behind Lagrange multipliers:

$$\max_{\mathbf{m} \in \mathbb{R}^p} L(\mathbf{x}, \mathbf{m}) = \infty, \quad \text{in case } \mathbf{C}\mathbf{x} \neq \mathbf{d}!$$

➔ A minimum in (7.4.2) can only be attained, if the constraint is satisfied!

(7.4.2) is called a **saddle point problem**.

Solution of min-max problem:

saddle point

$$F(x, m) = x^2 - 2xm$$



Note that the function is “flat” in the saddle point ●

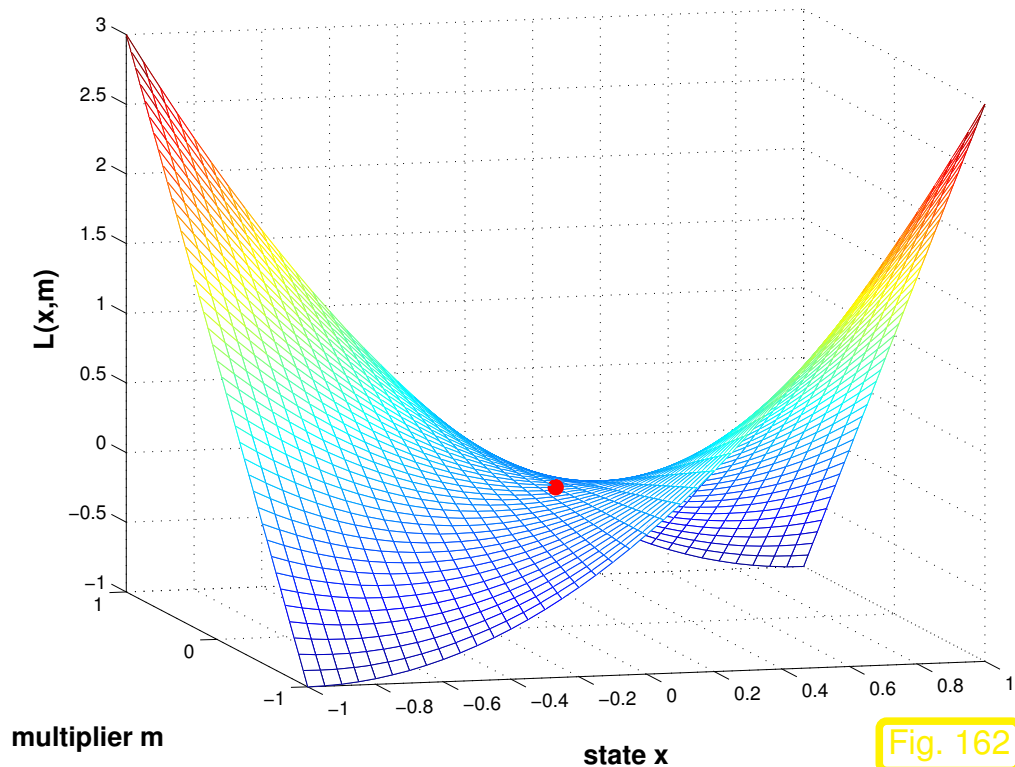


Fig. 162

Necessary (and sufficient) condition for the minimizer (→ Section 7.1)

$$\frac{\partial L}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{m}) = \mathbf{A}^H(\mathbf{A}\mathbf{x} - \mathbf{b}) + \mathbf{C}^H\mathbf{m} \stackrel{!}{=} \mathbf{0} \quad , \quad \frac{\partial L}{\partial \mathbf{m}}(\mathbf{x}, \mathbf{m}) = \mathbf{C}\mathbf{x} - \mathbf{d} \stackrel{!}{=} \mathbf{0}. \quad (7.4.4)$$

$$\begin{pmatrix} \mathbf{A}^H\mathbf{A} & \mathbf{C}^H \\ \mathbf{C} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{m} \end{pmatrix} = \begin{pmatrix} \mathbf{A}^H\mathbf{b} \\ \mathbf{d} \end{pmatrix} \quad \text{Augmented normal equations} \\ \text{(matrix saddle point problem)} \quad (7.4.5)$$

Algorithm for (7.4.5) (based on block-LU-decomposition):

$$\begin{pmatrix} \mathbf{A}^H \mathbf{A} & \mathbf{C}^H \\ \mathbf{C} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{R}^H & 0 \\ \mathbf{G} & -\mathbf{S}^H \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{G}^H \\ 0 & \mathbf{S} \end{pmatrix}, \quad \begin{array}{l} \mathbf{R}, \mathbf{S} \in \mathbb{R}^{n,n} \text{ upper triangular matrix,} \\ \mathbf{G} \in \mathbb{R}^{p,n}. \end{array}$$

$\mathbf{R}$  from  $\mathbf{R}^H \mathbf{R} = \mathbf{A}^H \mathbf{A} \rightarrow$  Cholesky decomposition  $\rightarrow$  Sect. 2.7,  
 $\mathbf{G}$  from  $\mathbf{R}^H \mathbf{G}^H = \mathbf{C}^H \rightarrow n$  forward substitution  $\rightarrow$  Sect. 2.2,  
 $\mathbf{S}$  from  $\mathbf{S}^H \mathbf{S} = \mathbf{G} \mathbf{G}^H \rightarrow$  Cholesky decomposition  $\rightarrow$  Sect. 2.7.

Caution      Sect. 7.1: the computation of  $\mathbf{A}^H \mathbf{A}$  can be expensive and problematic!  
 (remedy through introduction of a new unknown  $\mathbf{r} = \mathbf{A} \mathbf{x} - \mathbf{b}$ , cf. (7.1.7))

$$\begin{pmatrix} -\mathbf{I} & \mathbf{A} & 0 \\ \mathbf{A}^H & 0 & \mathbf{C}^H \\ 0 & \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \\ \mathbf{m} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \\ \mathbf{d} \end{pmatrix}. \quad (7.4.6)$$

Idea: identify the subspace in which the solution can vary without violating the constraint. Since  $\mathbf{C}$  has *full rank*, this subspace agrees with the nullspace/kernel of  $\mathbf{C}$ .

Lemma 6.5.14 ➤ SVD can be used to compute  $\text{Ker}(\mathbf{C})$

① Compute orthonormal basis of  $\text{Ker}(\mathbf{C})$  using SVD ( $\rightarrow$  Lemma 6.5.14, (7.2.2)):

$$\mathbf{C} = \mathbf{U} \begin{bmatrix} \Sigma & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{bmatrix}, \quad \mathbf{U} \in \mathbb{R}^{p,p}, \Sigma \in \mathbb{R}^{p,p}, \mathbf{V}_1 \in \mathbb{R}^{n,p}, \mathbf{V}_2 \in \mathbb{R}^{n,n-p}$$

▶  $\text{Ker}(\mathbf{C}) = \text{Im}(\mathbf{V}_2)$ .

and the **particular solution** of the constraint equation

$$\mathbf{x}_0 := \mathbf{V}_1 \Sigma^{-1} \mathbf{U}^H \mathbf{d}.$$

Representation of the solution  $\mathbf{x}$  of (7.4.1):  $\mathbf{x} = \mathbf{x}_0 + \mathbf{V}_2 \mathbf{y}, \quad \mathbf{y} \in \mathbb{R}^{n-p}.$

② Insert this representation in (7.4.1) ➤ standard linear least squares

$$\|\mathbf{A}(\mathbf{x}_0 + \mathbf{V}_2 \mathbf{y}) - \mathbf{b}\|_2 \rightarrow \min \Leftrightarrow \|\mathbf{A} \mathbf{V}_2 \mathbf{y} - (\mathbf{b} - \mathbf{A} \mathbf{x}_0)\| \rightarrow \min.$$

# 7.5 Non-linear Least Squares [13, Ch. 6]

*Example 7.5.1* (Non-linear data fitting (parametric statistics)). → Ex. 7.0.1 revisited

Given: data points  $(t_i, y_i)$ ,  $i = 1, \dots, m$  with measurement errors.

Known:  $y = f(\mathbf{x}, t)$  through a function  $f : \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}$  depending non-linearly and smoothly on parameters  $\mathbf{x} \in \mathbb{R}^n$ .

Example:  $f(t) = x_1 + x_2 \exp(-x_3 t)$ ,  $n = 3$ .

Determine parameters by non-linear **least squares data fitting**:

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^m |f(\mathbf{x}, t_i) - y_i|^2 = \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \|F(\mathbf{x})\|_2^2, \quad (7.5.2)$$

$$\text{with } F(\mathbf{x}) = \begin{pmatrix} f(\mathbf{x}, t_1) - y_1 \\ \vdots \\ f(\mathbf{x}, t_m) - y_m \end{pmatrix}.$$





## Non-linear least squares problem

Given:  $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^m$ ,  $m, n \in \mathbb{N}$ ,  $m > n$ .

Find:  $\mathbf{x}^* \in D$ :  $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x} \in D} \Phi(\mathbf{x})$ ,  $\Phi(\mathbf{x}) := \frac{1}{2} \|F(\mathbf{x})\|_2^2$ . (7.5.3)

Terminology:  $D \hat{=}$  parameter space,  $x_1, \dots, x_n \hat{=}$  parameter.

As in the case of linear least squares problems ( $\rightarrow$  Rem. 7.0.11): a non-linear least squares problem is related to an overdetermined non-linear system of equations  $F(\mathbf{x}) = 0$ .

As for non-linear systems of equations ( $\rightarrow$  Chapter 4): existence and uniqueness of  $\mathbf{x}^*$  in (7.5.3) has to be established in each concrete case!

We require “independence for each parameter”:  $\rightarrow$  Rem. 7.0.18

$\exists$  neighbourhood  $\mathcal{U}(\mathbf{x}^*)$  such that  $DF(\mathbf{x})$  has full rank  $n \quad \forall \mathbf{x} \in \mathcal{U}(\mathbf{x}^*)$ . (7.5.4)

(It means: the columns of the Jacobi matrix  $DF(\mathbf{x})$  are linearly independent.)

If (7.5.4) is not satisfied, then the parameters are redundant in the sense that fewer parameters would be enough to model the same dependence (locally at  $\mathbf{x}^*$ ), cf. Rem. 7.0.18.

## 7.5.1 (Damped) Newton method

$$\Phi(\mathbf{x}^*) = \min \Rightarrow \mathbf{grad} \Phi(\mathbf{x}) = 0, \quad \mathbf{grad} \Phi(\mathbf{x}) := \left( \frac{\partial \Phi}{\partial x_1}(\mathbf{x}), \dots, \frac{\partial \Phi}{\partial x_n}(\mathbf{x}) \right)^T \in \mathbb{R}^n.$$

Simple idea: use Newton's method ( $\rightarrow$  Sect. 4.4) to determine a zero of  $\mathbf{grad} \Phi : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ .

Newton iteration (4.4.1) for non-linear system of equations  $\mathbf{grad} \Phi(\mathbf{x}) = 0$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - H\Phi(\mathbf{x}^{(k)})^{-1} \mathbf{grad} \Phi(\mathbf{x}^{(k)}), \quad (H\Phi(\mathbf{x}) = \mathbf{Hessian\ matrix}). \quad (7.5.5)$$

Expressed in terms of  $F : \mathbb{R}^n \mapsto \mathbb{R}^m$  from (7.5.3):

chain rule (4.4.4)    ➤     $\mathbf{grad} \Phi(\mathbf{x}) = DF(\mathbf{x})^T F(\mathbf{x})$  ,

product rule (4.4.5)    ➤     $H\Phi(\mathbf{x}) := D(\mathbf{grad} \Phi)(\mathbf{x}) = DF(\mathbf{x})^T DF(\mathbf{x}) + \sum_{j=1}^m F_j(\mathbf{x}) D^2 F_j(\mathbf{x})$  ,

⇕

$$(H\Phi(\mathbf{x}))_{i,k} = \sum_{j=1}^m \frac{\partial^2 F_j}{\partial x_i \partial x_k}(\mathbf{x}) F_j(\mathbf{x}) + \frac{\partial F_j}{\partial x_k}(\mathbf{x}) \frac{\partial F_j}{\partial x_i}(\mathbf{x}) .$$

Recommendation, *cf.* Rem. 4.4.3: when in doubt, differentiate components of matrices and vectors!

The above derivative formulas allow to rewrite (7.5.5) in concrete terms:

► For Newton iterate  $\mathbf{x}^{(k)}$ : Newton correction  $\mathbf{s} \in \mathbb{R}^n$  from LSE

$$\underbrace{\left( DF(\mathbf{x}^{(k)})^T DF(\mathbf{x}^{(k)}) + \sum_{j=1}^m F_j(\mathbf{x}^{(k)}) D^2 F_j(\mathbf{x}^{(k)}) \right)}_{=H\Phi(\mathbf{x}^{(k)})} \mathbf{s} = - \underbrace{DF(\mathbf{x}^{(k)})^T F(\mathbf{x}^{(k)})}_{=\mathbf{grad} \Phi(\mathbf{x}^{(k)})} . \quad (7.5.6)$$

*Remark 7.5.7* (Newton method and minimization of quadratic functional).

Newton's method (7.5.5) for (7.5.3) can be read as *successive minimization* of a local **quadratic approximation** of  $\Phi$ :

$$\Phi(\mathbf{x}) \approx Q(\mathbf{s}) := \Phi(\mathbf{x}^{(k)}) + \mathbf{grad} \Phi(\mathbf{x}^{(k)})^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T H\Phi(\mathbf{x}^{(k)}) \mathbf{s}, \quad (7.5.8)$$

$$\mathbf{grad} Q(\mathbf{s}) = 0 \Leftrightarrow H\Phi(\mathbf{x}^{(k)}) \mathbf{s} + \mathbf{grad} \Phi(\mathbf{x}^{(k)}) = 0 \Leftrightarrow (7.5.6).$$

- So we deal with yet another model function method ( $\rightarrow$  Sect. 4.3.2) with quadratic model function for  $Q$ .



## 7.5.2 Gauss-Newton method

Idea: **local linearization** of  $F$ :  $F(\mathbf{x}) \approx F(\mathbf{y}) + DF(\mathbf{y})(\mathbf{x} - \mathbf{y})$

- sequence of *linear* least squares problems

$$\operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \|F(\mathbf{x})\|_2 \quad \text{is approximated by} \quad \underbrace{\operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \|F(\mathbf{x}_0) + DF(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)\|_2}_{(\spadesuit)},$$

where  $\mathbf{x}_0$  is an approximation of the solution  $\mathbf{x}^*$  of (7.5.3).

$$(\spadesuit) \Leftrightarrow \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\| \quad \text{with} \quad \mathbf{A} := DF(\mathbf{x}_0) \in \mathbb{R}^{m,n}, \quad \mathbf{b} := -F(\mathbf{x}_0) + DF(\mathbf{x}_0)\mathbf{x}_0 \in \mathbb{R}^m.$$

This is a linear least squares problem of the form (7.0.16).

Note: (7.5.4)  $\Rightarrow$   $\mathbf{A}$  has full rank, if  $\mathbf{x}_0$  sufficiently close to  $\mathbf{x}^*$ .

Note: This approach is different from local quadratic approximation of  $\Phi$  underlying Newton's method for (7.5.3), see Sect. 7.5.1, Rem. 7.5.7.

Initial guess  $\mathbf{x}^{(0)} \in D$

$$\mathbf{x}^{(k+1)} := \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \left\| F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) \right\|_2. \quad (7.5.9)$$

linear least squares problem

MATLAB-\`\` used to solve linear least squares problem in each step:

for  $\mathbf{A} \in \mathbb{R}^{m,n}$

$$\mathbf{x} = \mathbf{A} \backslash \mathbf{b}$$



$\mathbf{x}$  minimizer of  $\|\mathbf{Ax} - \mathbf{b}\|_2$   
with minimal 2-norm

Code 7.5.10: template for Gauss-Newton method

```

1 function x = gn(x,F,J,tol)
2 s = J(x) \ F(x); %
3 x = x-s;
4 while (norm(s) > tol*norm(x)) %
5     s = J(x) \ F(x); %
6     x = x-s;
7 end
    
```

Comments on Code 7.5.9:

☞ Argument `x` passes initial guess  $\mathbf{x}^{(0)} \in \mathbb{R}^n$ , argument `F` must be a *handle* to a function  $F : \mathbb{R}^n \mapsto \mathbb{R}^m$ , argument `J` provides the Jacobian of  $F$ , namely  $DF : \mathbb{R}^n \mapsto \mathbb{R}^{m,n}$ , argument `tol` specifies the tolerance for termination

☞ Line 4: iteration terminates if relative norm of correction is below threshold specified in `tol`.

Note: Code 7.5.9 also implements Newton's method ( $\rightarrow$  Sect. 4.4.1) in the case  $m = n$ !

Summary:

Advantage of the Gauss-Newton method : second derivative of  $F$  not needed.

Drawback of the Gauss-Newton method : no local quadratic convergence.

*Example 7.5.11* (Non-linear data fitting (II)).  $\rightarrow$  Ex. 7.5.1

Non-linear data fitting problem (7.5.2) for  $f(t) = x_1 + x_2 \exp(-x_3 t)$ .

$$F(\mathbf{x}) = \begin{pmatrix} x_1 + x_2 \exp(-x_3 t_1) - y_1 \\ \vdots \\ x_1 + x_2 \exp(-x_3 t_m) - y_m \end{pmatrix} : \mathbb{R}^3 \mapsto \mathbb{R}^m, \quad DF(\mathbf{x}) = \begin{pmatrix} 1 & e^{-x_3 t_1} & -x_2 t_1 e^{-x_3 t_1} \\ \vdots & \vdots & \vdots \\ 1 & e^{-x_3 t_m} & -x_2 t_m e^{-x_3 t_m} \end{pmatrix}$$

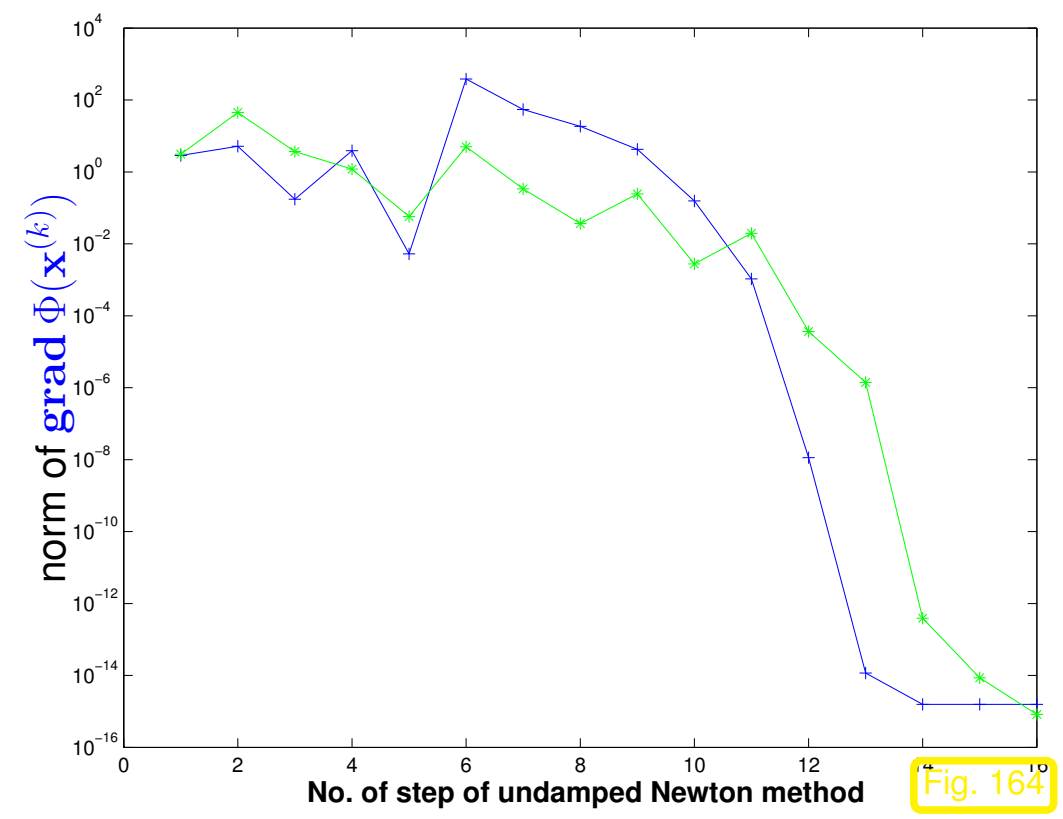
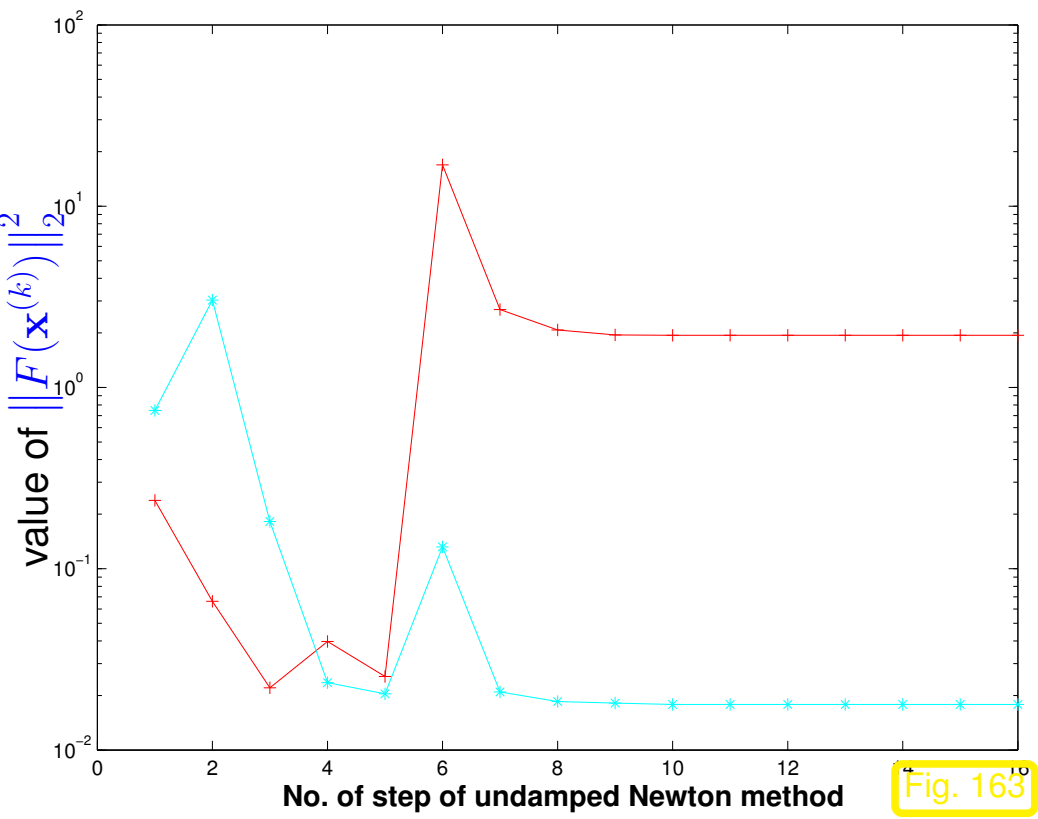
Numerical experiment:

convergence of the Newton method,  
damped Newton method ( $\rightarrow$  Section  
4.4.4) and Gauss-Newton method for  
different initial values

```
rand('seed', 0);
t = (1:0.3:7)';
y = x(1) + x(2)*exp(-x(3)*t);
y = y+0.1*(rand(length(y), 1)-0.5);
```

- initial value  $(1.8, 1.8, 0.1)^T$  (red curves, blue curves)
- initial value  $(1.5, 1.5, 0.1)^T$  (cyan curves, green curves)

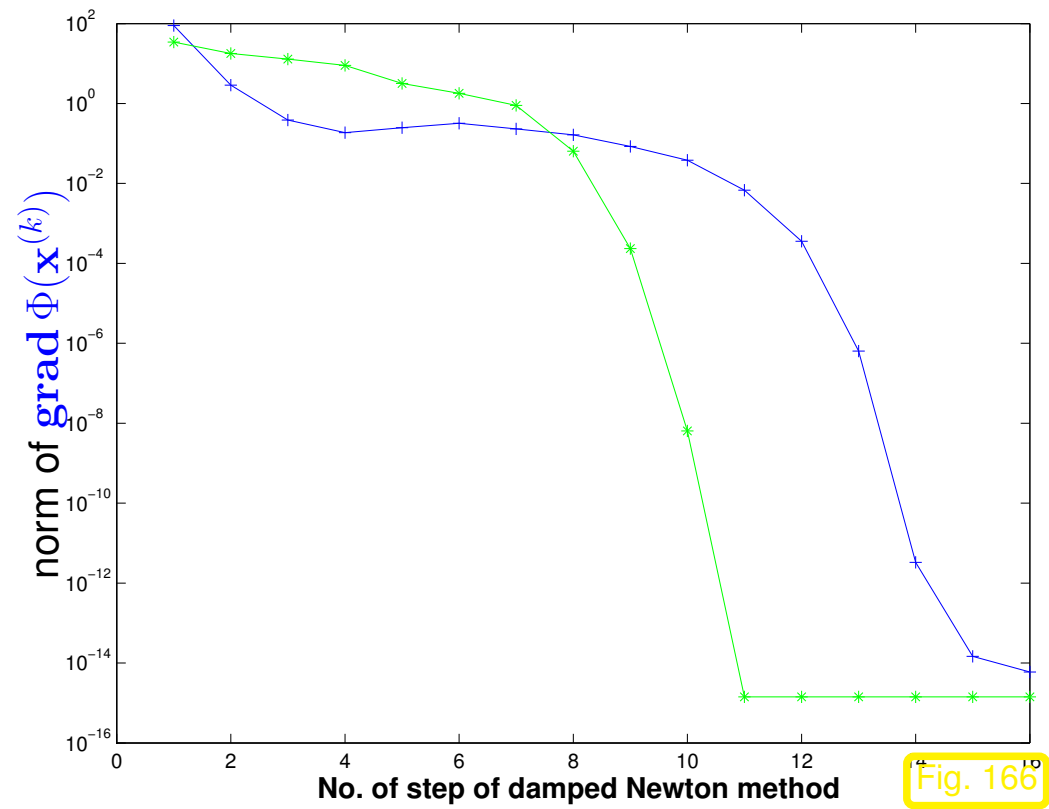
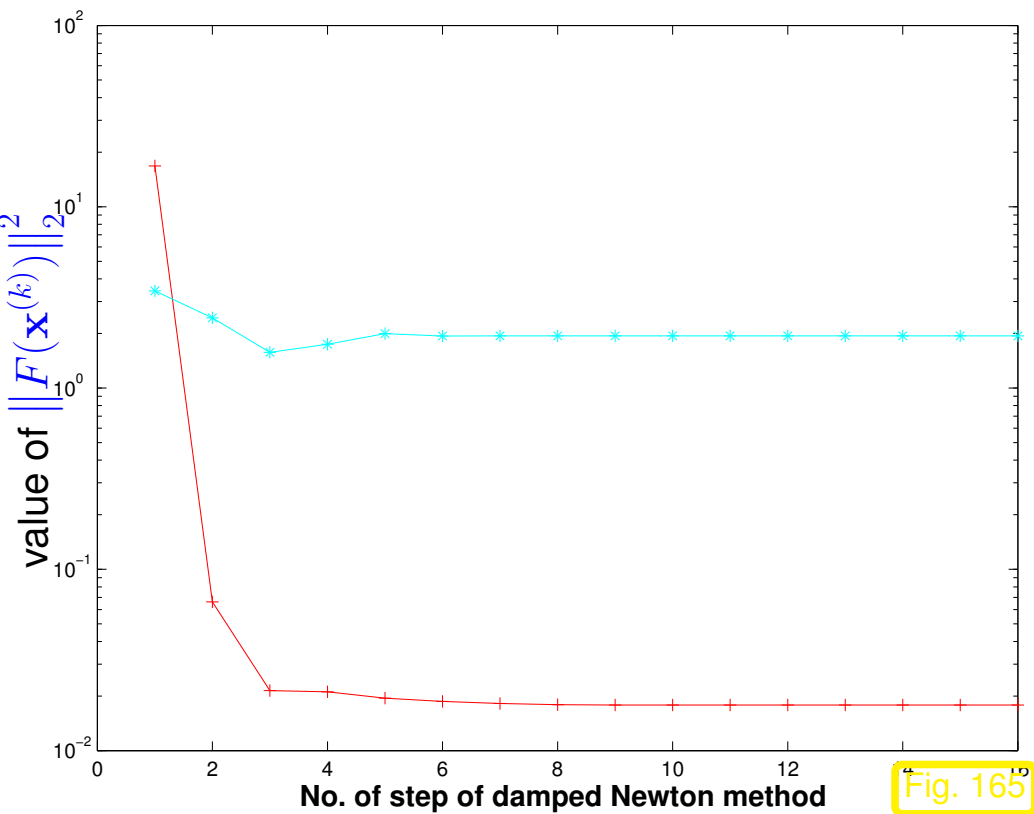
First experiment (→ Sect. 7.5.1): iterative solution of non-linear least squares data fitting problem by means of the Newton method (7.5.6) and the damped Newton method from Code 4.4.17





# Convergence behaviour of plain Newton method:

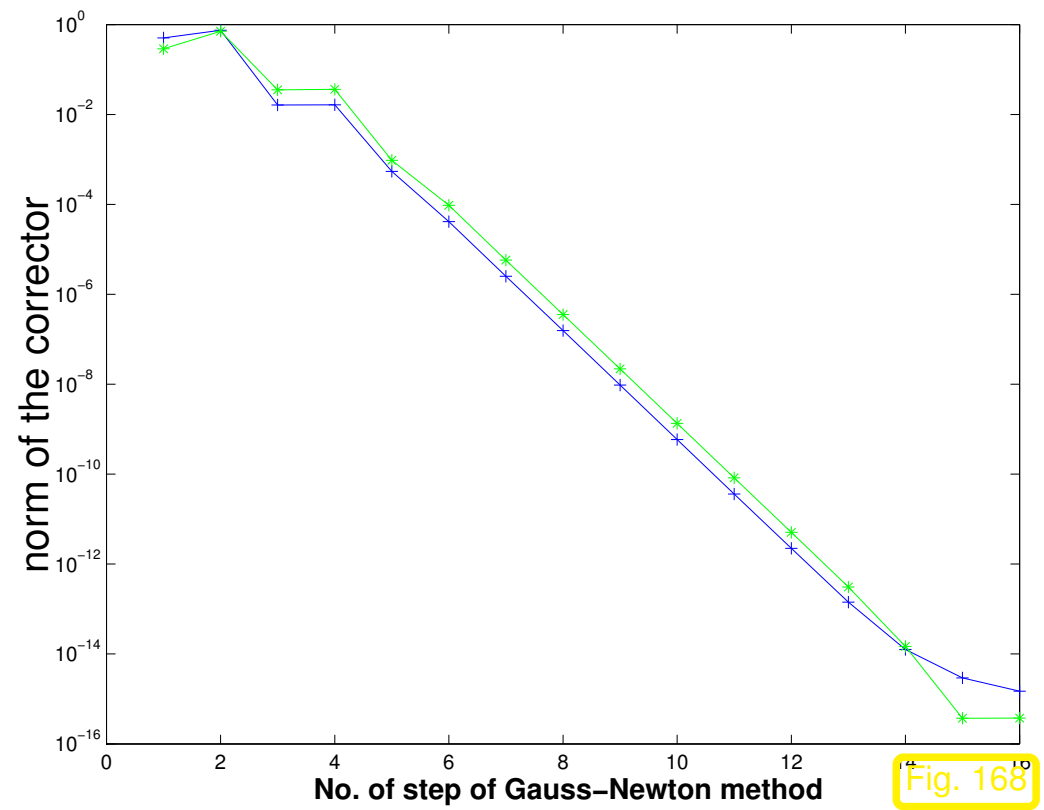
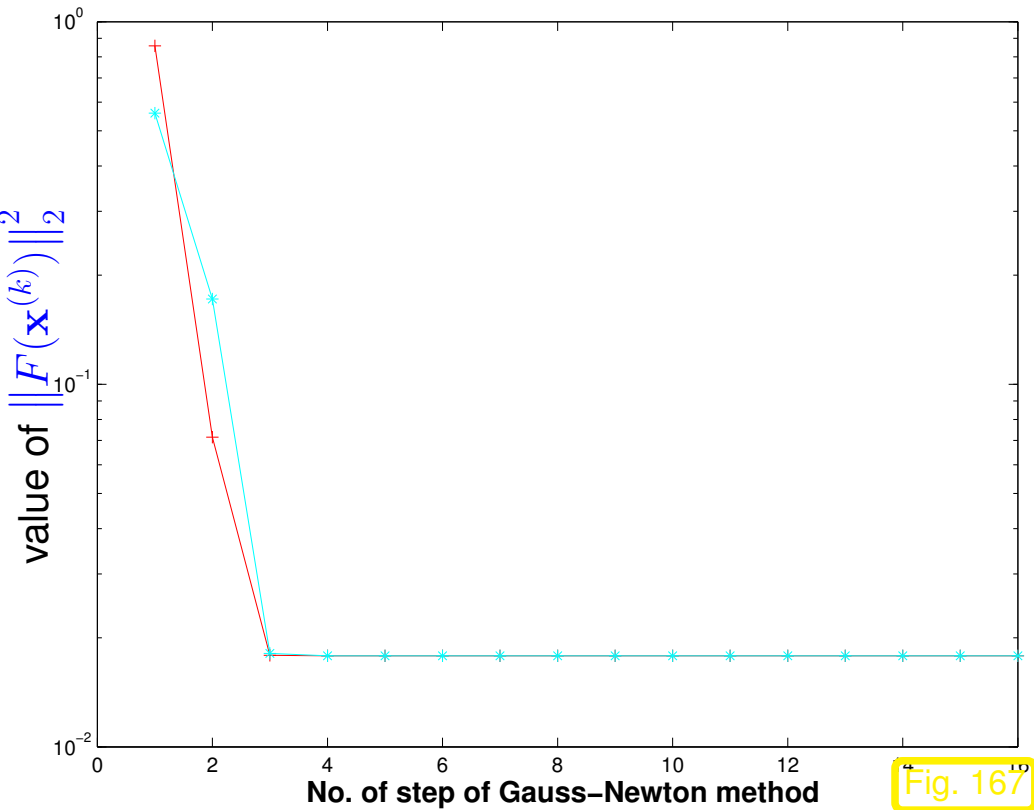
initial value  $(1.8, 1.8, 0.1)^T$  (red curve) ➤ Newton method caught in **local minimum**,  
 initial value  $(1.5, 1.5, 0.1)^T$  (cyan curve) ➤ fast (locally quadratic) convergence.



## Convergence behavior of damped Newton method:

initial value  $(1.8, 1.8, 0.1)^T$  (red curve) ➤ fast (locally quadratic) convergence,  
initial value  $(1.5, 1.5, 0.1)^T$  (cyan curve) ➤ Newton method caught in **local minimum**.

Second experiment: iterative solution of non-linear least squares data fitting problem by means of the Gauss-Newton method (7.5.9), see Code 7.5.9.



We observe: **linear convergence** for all initial values, *cf.* Def. 4.1.6, Rem. 4.1.10.



## 7.5.3 Trust region method (Levenberg-Marquardt method)

As in the case of Newton's method for non-linear systems of equations, see Sect. 4.4.4: often overshooting of Gauss-Newton corrections occurs.

Remedy as in the case of Newton's method: **damping**.

Idea: damping of the Gauss-Newton correction in (7.5.9) using a **penalty term**

instead of  $\left\| F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})\mathbf{s} \right\|^2$  minimize  $\left\| F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})\mathbf{s} \right\|^2 + \lambda \|\mathbf{s}\|_2^2$  .

$\lambda > 0 \hat{=}$  penalty parameter (how to choose it ?  $\rightarrow$  heuristic)

$$\lambda = \gamma \left\| F(\mathbf{x}^{(k)}) \right\|_2, \quad \gamma := \begin{cases} 10 & , \text{ if } \left\| F(\mathbf{x}^{(k)}) \right\|_2 \geq 10, \\ 1 & , \text{ if } 1 < \left\| F(\mathbf{x}^{(k)}) \right\|_2 < 10, \\ 0.01 & , \text{ if } \left\| F(\mathbf{x}^{(k)}) \right\|_2 \leq 1. \end{cases}$$

► Modified (regularized) equation for the corrector  $\mathbf{s}$ :

$$\left( DF(\mathbf{x}^{(k)})^T DF(\mathbf{x}^{(k)}) + \lambda \mathbf{I} \right) \mathbf{s} = -DF(\mathbf{x}^{(k)})F(\mathbf{x}^{(k)}) . \quad (7.5.12)$$

## 8

## Filtering Algorithms

This chapter continues the theme of *numerical linear algebra*, earlier covered in Chapters 1, 2, 5. We will come across very special linear transformations ( $\leftrightarrow$  matrices) and related algorithms. Surprisingly, these form the basis of a host of very important numerical methods for **signal processing**.

R. Hiptmair  
rev 38286,  
November  
17, 2011

Perspective of **signal processing**:

vector  $\mathbf{x} \in \mathbb{R}^n$   $\leftrightarrow$  finite discrete (= sampled) signal.

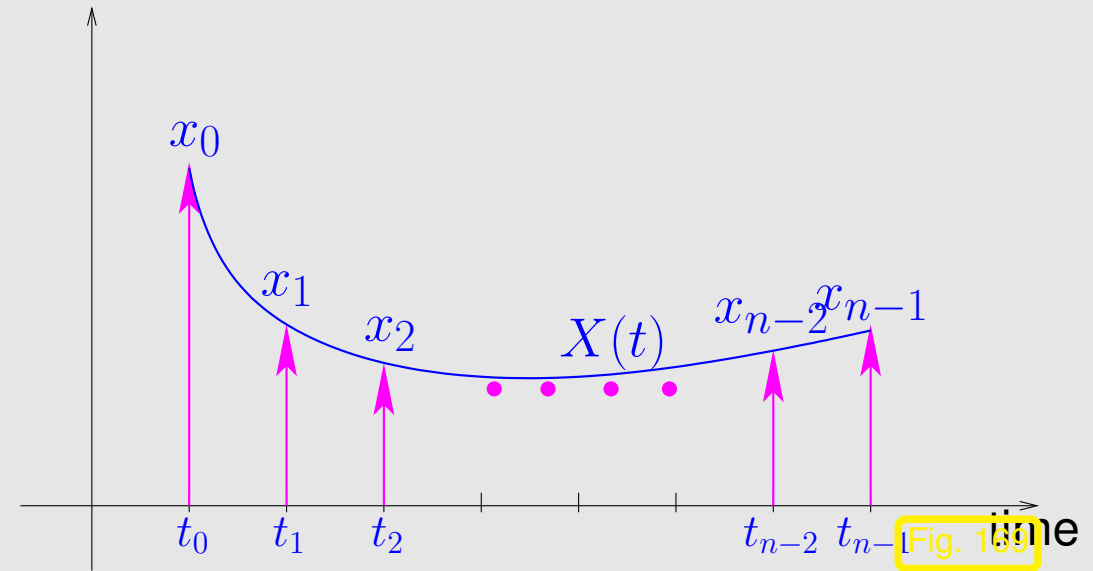
**Sampling:**

$X = X(t) \hat{=}$  *time-continuous* signal,  $0 \leq t \leq T$ ,

“sampling”:  $x_j = X(j\Delta t)$ ,  $j = 0, \dots, n-1$ ,  
 $n \in \mathbb{N}$ ,  $n\Delta t \leq T$ .

$\Delta t > 0 \hat{=}$  time between samples.

Sampled values arranged in a vector  $\mathbf{x} = (x_0, \dots, x_{n-1})^\top \in \mathbb{R}^n$ .



Note: vector indices  $0, \dots, n-1$  ! (“C-style indexing”).

Generalization: (bi-infinite) time-discrete signals of infinite duration

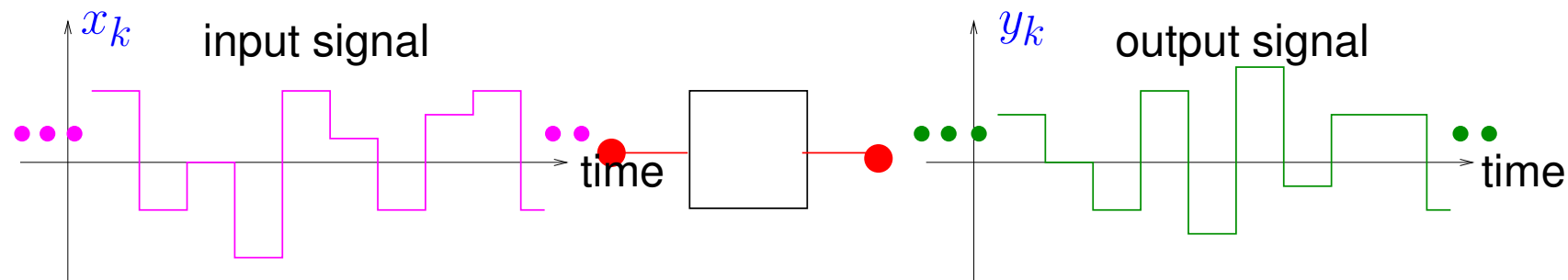
$\Updownarrow$   
sequence  $(x_k)_{k \in \mathbb{Z}}$

# 8.1 Discrete convolutions

*Example 8.1.1* (Discrete finite linear time-invariant causal channel (filter)).

We study a **finite linear time-invariant causal channel (filter)**:  
(widely used model for digital communication channels, e.g. in wireless communication theory)

Mathematically speaking, a (discrete) channel/filter is a mapping from the space of input sequences  $\{x_i\}_{i \in \mathbb{Z}}$  to output sequences  $\{y_i\}_{i \in \mathbb{Z}}$ .

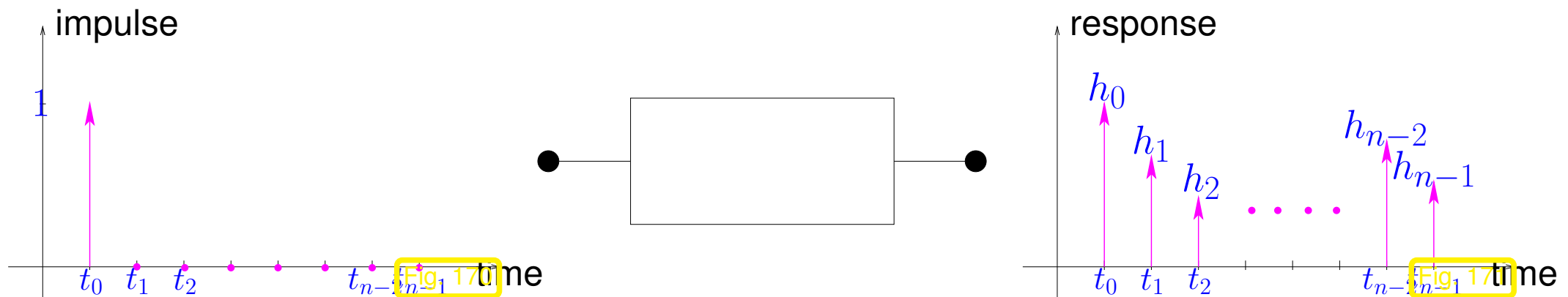




**Impulse response** = output when filter is fed with a single impulse of strength one, corresponding to input  $\mathbf{e}_1$  (first unit vector).

Impulse response of channel (filter):

$$\mathbf{h} = (h_0, \dots, h_{n-1})^T$$



In order to link digital filters to linear algebra, we have to assume certain properties that are indicated by the attributes “finite”, “linear”, “time-invariant” and “causal”:

**finite**: impulse response of finite duration  $\Rightarrow$  it can be described by a vector  $\mathbf{h}$  of finite length  $n$ .

**time-invariant**: when input is shifted in time, output is shifted by the same amount of time.

**linear**: input  $\mapsto$  output-map is linear

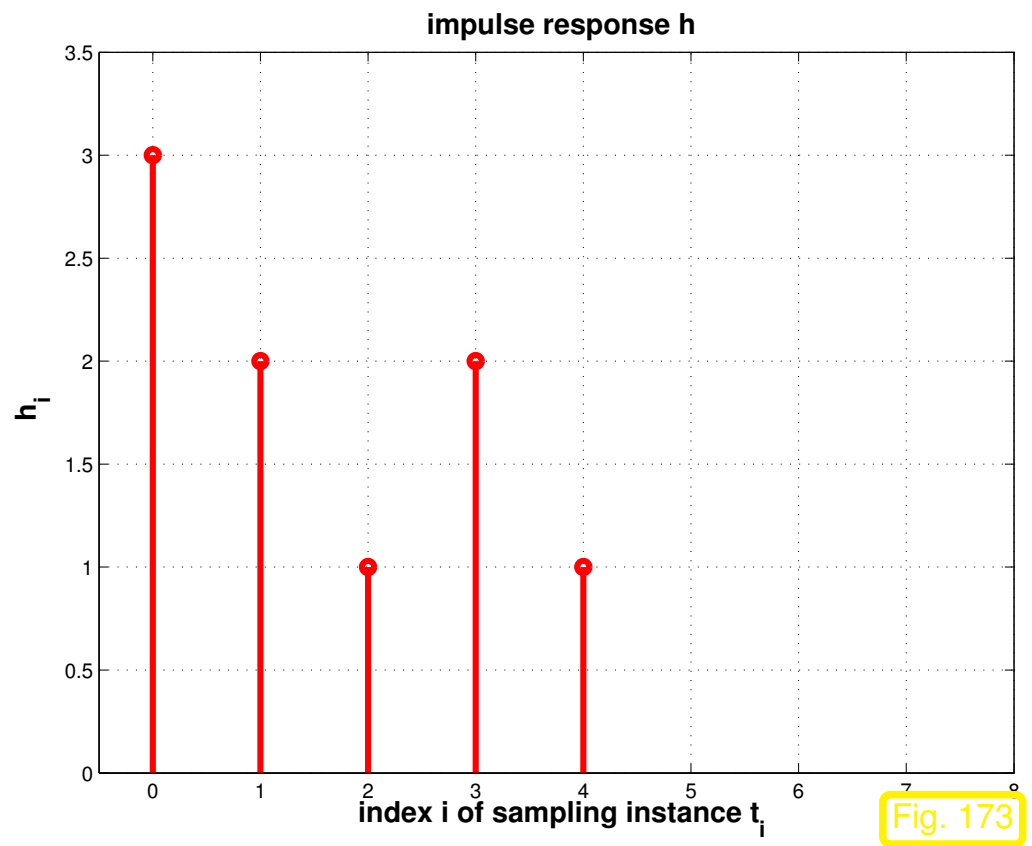
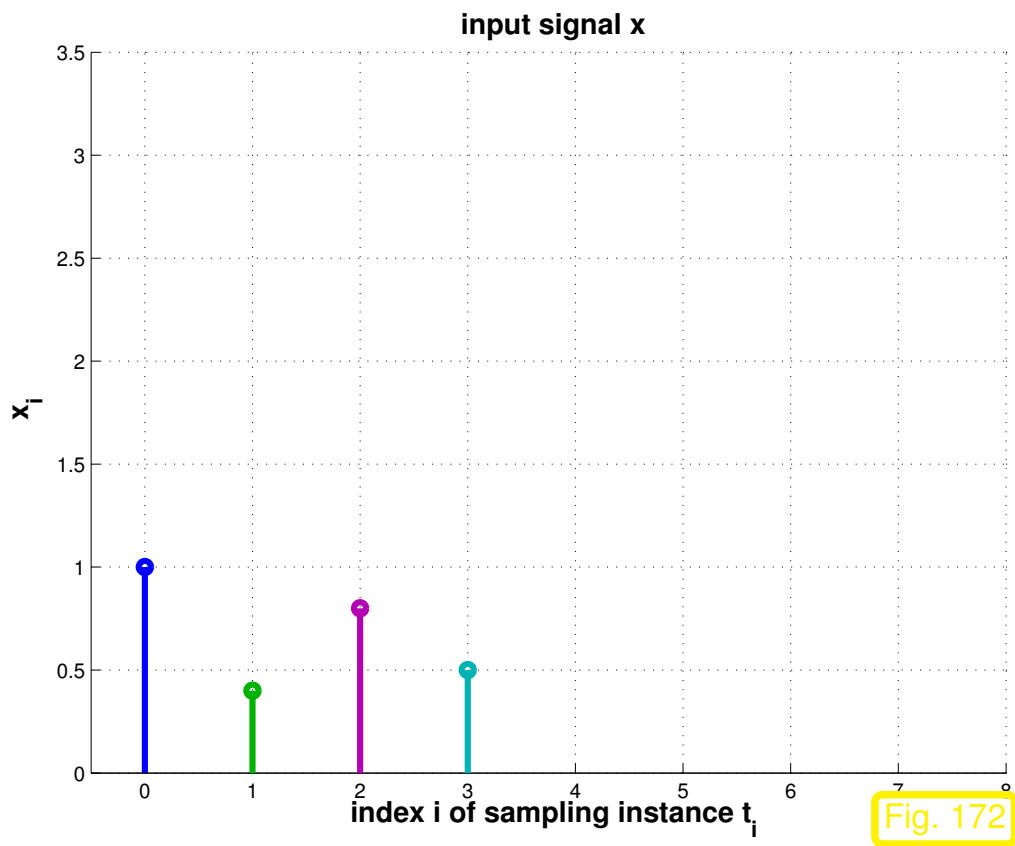
$$\text{output}(\mu \cdot \text{signal 1} + \lambda \cdot \text{signal 2}) = \mu \cdot \text{output}(\text{signal 1}) + \lambda \cdot \text{output}(\text{signal 2}) . \quad (8.1.2)$$

**causal** (or physical, or nonanticipative): output depends only on past and present inputs, not on the future.

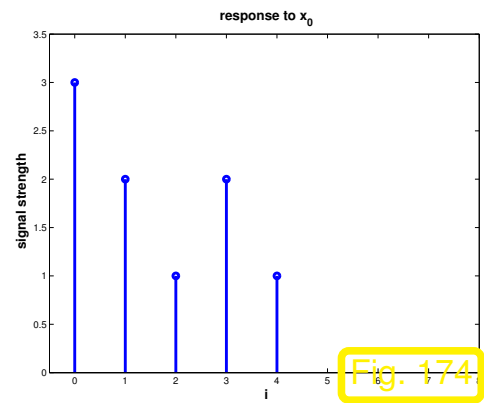
(Note: the impulse response depicted in Fig. 171 is finite and causal)

► The output  $(y_1, y_2, \dots)^\top$  for finite length input  $\mathbf{x} = (x_0, \dots, x_{n-1})^\top \in \mathbb{R}^n$  is  
a superposition of  $x_j$ -weighted  $j\Delta t$ -shifted impulse responses

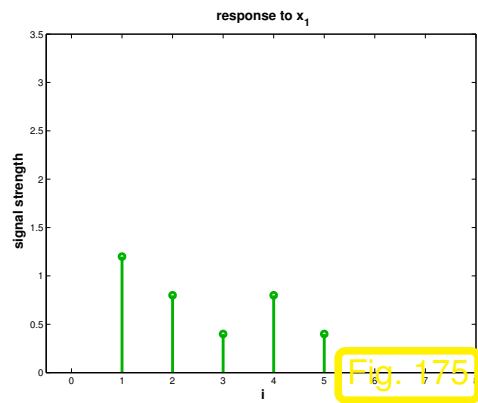
Understand, why this is a consequence of linearity (8.1.2).



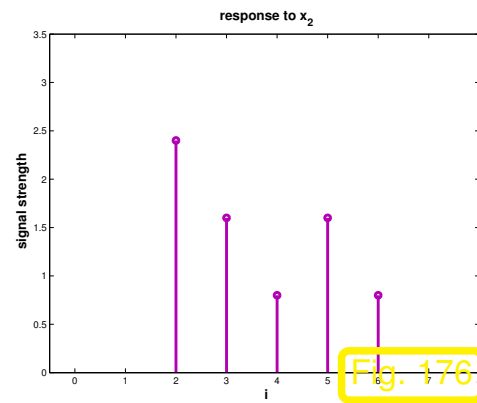
Output = linear superposition of impulse responses:



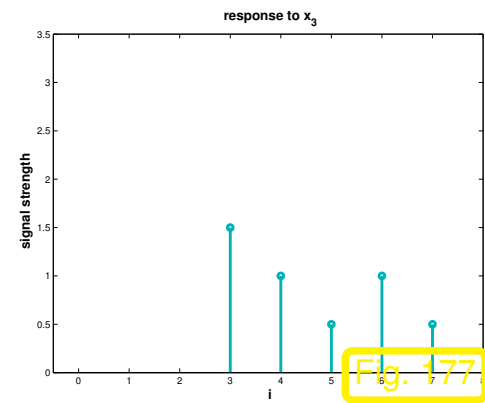
+

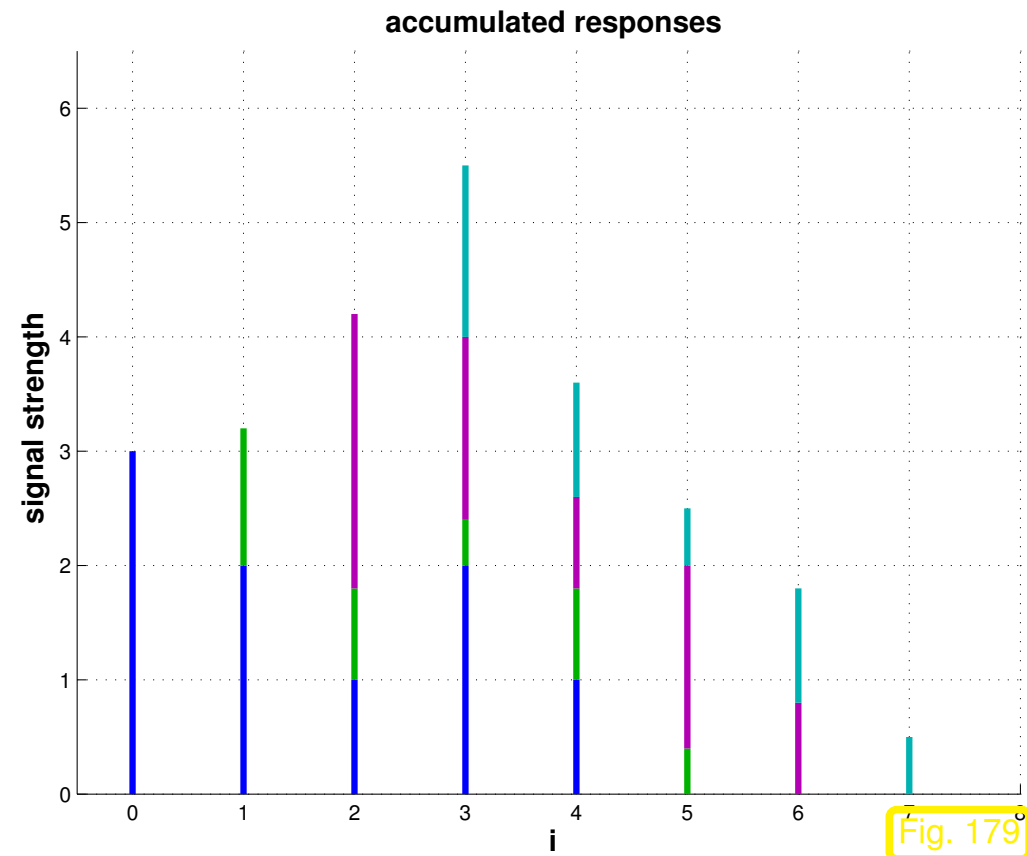
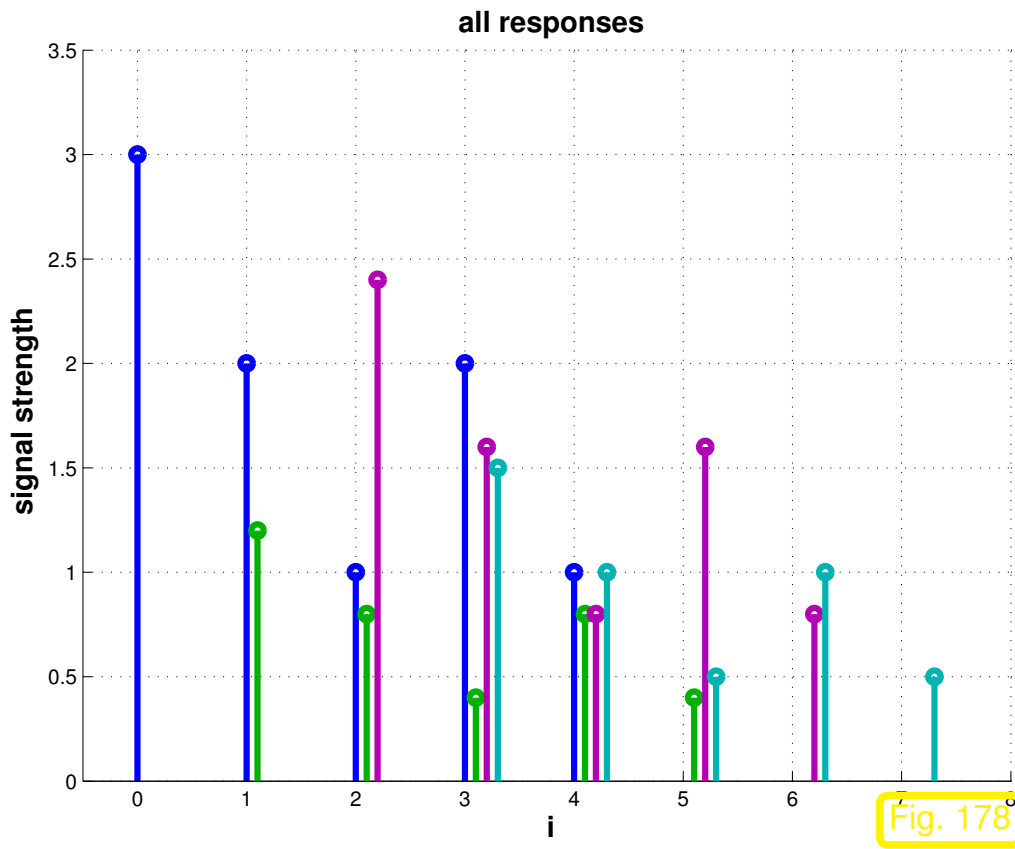


+



+





“Duration” of output signal = “duration” of input signal + “duration” of impulse response

► General formula:

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \\ \vdots \\ y_{2n-3} \\ y_{2n-2} \end{pmatrix} = x_0 \begin{pmatrix} h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} + x_1 \begin{pmatrix} 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \cdots + x_{n-1} \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \end{pmatrix} .$$

channel is causal!

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j, \quad k = 0, \dots, 2n-2 \quad (h_j := 0 \text{ for } j < 0 \text{ and } j \geq n) . \quad (8.1.3)$$

$\mathbf{x} = (x_0, \dots, x_{n-1})^\top \in \mathbb{R}^n \hat{=} \text{input signal} \quad \mapsto \quad \mathbf{y} = (y_0, \dots, y_{2n-2})^\top \in \mathbb{R}^{2n-1} \hat{=} \text{output signal} .$

Matrix notation of (8.1.3):

$$\begin{pmatrix} y_0 \\ \vdots \\ y_{2n-2} \end{pmatrix} = \begin{pmatrix} h_0 & 0 & \cdots & 0 \\ h_1 & & & \\ & \ddots & & \\ & & h_1 & h_0 \\ & & & \\ & & 0 & \\ & & & \ddots & \\ & & & & 0 & h_{n-1} \\ & & & & & \\ & & & & & 0 & h_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix}. \tag{8.1.4}$$

◇

Example 8.1.5 (Multiplication of polynomials).

“Surprisingly” the bilinear operation (8.1.3) that takes two input vectors and produces an output vector with double the number of entries ( $-1$ ) also governs the multiplication of polynomials:

$$p(z) = \sum_{k=0}^{n-1} a_k z^k, \quad q(z) = \sum_{k=0}^{n-1} b_k z^k \quad \blacktriangleright \quad (pq)(z) = \sum_{k=0}^{2n-2} \underbrace{\left( \sum_{j=0}^k a_j b_{k-j} \right)}_{=:c_k} z^k \quad (8.1.6)$$

Here the roles of  $h_k, x_k$  are played by the  $a_k$  and  $b_k$ .

➤ coefficients of product polynomial by **discrete convolution** of coefficients of polynomial factors!



Both in (8.1.3) and (8.1.6) we recognize the same pattern of a particular *bi-linear* combination of

- discrete signals in Ex. 8.1.1,
- polynomial coefficient sequences in Ex. 8.1.5.

**Definition 8.1.7** (Discrete convolution).

Given  $\mathbf{x} = (x_0, \dots, x_{n-1})^\top \in \mathbb{K}^n$ ,  $\mathbf{h} = (h_0, \dots, h_{n-1})^\top \in \mathbb{K}^n$  their **discrete convolution** (ger.: *diskrete Faltung*) is the vector  $\mathbf{y} \in \mathbb{K}^{2n-1}$  with components

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j, \quad k = 0, \dots, 2n-2 \quad (h_j := 0 \text{ for } j < 0). \quad (8.1.8)$$

 Notation for discrete convolution (8.1.8):

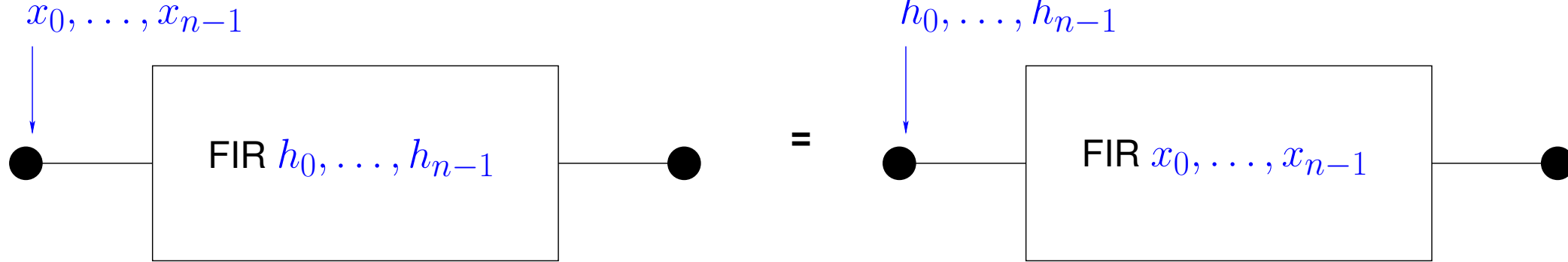
$$\mathbf{y} = \mathbf{h} * \mathbf{x}.$$

Defining  $x_j := 0$  for  $j < 0$ , we find that *discrete convolution is commutative*:

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j = \sum_{l=0}^{n-1} h_l x_{k-l}, \quad k = 0, \dots, 2n-2, \quad (\text{that is, } \mathbf{h} * \mathbf{x} = \mathbf{x} * \mathbf{h} \text{ )},$$

obtained by index transformation  $l \leftarrow k - j$ .





*Remark 8.1.9* (Convolution of sequences).

The notion of a discrete convolution of Def. 8.1.7 naturally extends to sequences  $\mathbb{N}_0 \mapsto \mathbb{K}$ : the (discrete) convolution of two sequences  $(x_j)_{j \in \mathbb{N}_0}$ ,  $(y_j)_{j \in \mathbb{N}_0}$  is the sequence  $(z_j)_{j \in \mathbb{N}_0}$  defined by

$$z_k := \sum_{j=0}^k x_{k-j} y_j = \sum_{j=0}^k x_j y_{k-j}, \quad k \in \mathbb{N}_0.$$

In this context recall: product formula for power series, **Cauchy product**



*Example 8.1.10* (Linear filtering of periodic signals).

$n$ -periodic signal ( $n \in \mathbb{N}$ ) = sequence  $(x_j)_{j \in \mathbb{Z}}$  with  $x_{j+n} = x_j \quad \forall j \in \mathbb{Z}$

➤  $n$ -periodic signal  $(x_j)_{j \in \mathbb{Z}}$  fixed by  $x_0, \dots, x_{n-1} \leftrightarrow$  vector  $\mathbf{x} = (x_0, \dots, x_{n-1})^\top \in \mathbb{R}^n$ .

Whenever the input signal of a time-invariant filter is  $n$ -periodic, so will be the output signal. Thus, in the  $n$ -periodic setting, a causal *linear* time-invariant filter will give rise to a *linear* mapping  $\mathbb{R}^n \mapsto \mathbb{R}^n$  according to

$$y_k = \sum_{j=0}^{n-1} p_{k-j} x_j \quad \text{for some } p_0, \dots, p_{n-1} \in \mathbb{R}, \quad p_k := p_{k-n} \text{ for all } k \in \mathbb{Z}. \quad (8.1.11)$$

R. Hiptmair  
rev 38286,  
November  
17, 2011

Note:  $p_0, \dots, p_{n-1}$  does **not** agree with the impulse response of the filter.

Matrix notation:

$$\begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} p_0 & p_{n-1} & p_{n-2} & \cdots & \cdots & p_1 \\ p_1 & p_0 & p_{n-1} & & & \vdots \\ p_2 & p_1 & p_0 & \ddots & & \\ \vdots & & \ddots & \ddots & \ddots & \\ \vdots & & & \ddots & \ddots & \\ p_{n-1} & \cdots & & & \cdots & p_1 & p_0 \end{pmatrix}}{=: \mathbf{P}} \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix}. \quad (8.1.12)$$

$$(\mathbf{P})_{ij} = p_{i-j}, \quad 1 \leq i, j \leq n, \quad \text{with } p_j := p_{j+n} \text{ for } 1 - n \leq j < 0.$$

**Definition 8.1.13** (Discrete periodic convolution).

The *discrete periodic convolution* of two  $n$ -periodic sequences  $(x_k)_{k \in \mathbb{Z}}$ ,  $(y_k)_{k \in \mathbb{Z}}$  yields the  $n$ -periodic sequence

$$(z_k) := (x_k) *_n (y_k) \quad , \quad z_k := \sum_{j=0}^{n-1} x_{k-j} y_j = \sum_{j=0}^{n-1} y_{k-j} x_j \quad , \quad k \in \mathbb{Z} .$$

notation for discrete periodic convolution:  $(x_k) *_{n} (y_k)$

Since  $n$ -periodic sequences can be identified with vectors in  $\mathbb{K}^n$  (see above), we can also introduce the discrete periodic convolution of vectors:

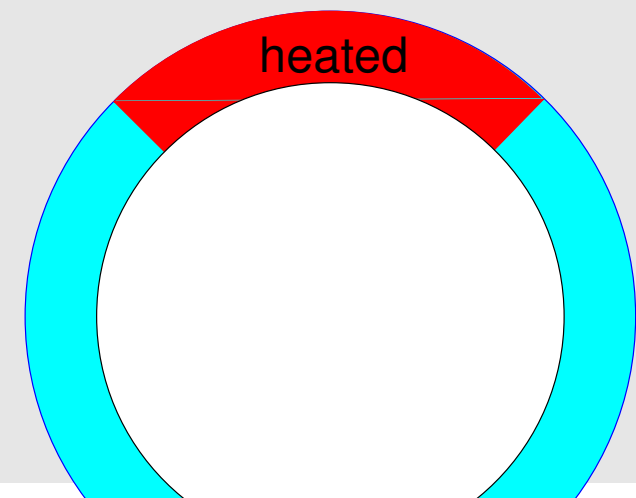
Def. 8.1.13  $\triangleright$  discrete periodic convolution of vectors:  $\mathbf{z} = \mathbf{x} *_{n} \mathbf{y} \in \mathbb{K}^n, \quad \mathbf{x}, \mathbf{y} \in \mathbb{K}^n.$

*Example 8.1.14* (Radiative heat transfer).

Beyond signal processing discrete periodic convolutions occur in mathematical models:

An engineering problem:

- cylindrical pipe,
- heated on part  $\Gamma_H$  of its perimeter ( $\rightarrow$  prescribed heat flux),
- cooled on remaining perimeter  $\Gamma_K$  ( $\rightarrow$  constant heat flux).



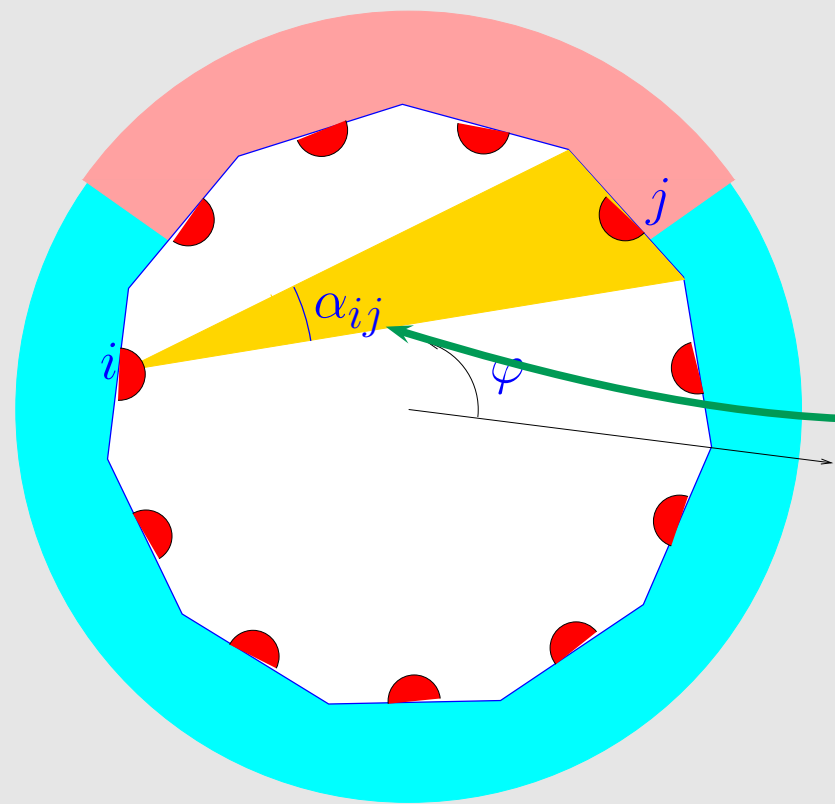
**Modeling** (discretization):

- approximation by regular  $n$ -polygon, edges  $\Gamma_j$ ,
- isotropic radiation of each edge  $\Gamma_j$  (power  $I_j$ ),

radiative heat flow  $\Gamma_j \rightarrow \Gamma_i: P_{ji} := \frac{\alpha_{ij}}{\pi} I_j$ ,

opening angle:  $\alpha_{ij} = \pi \gamma_{|i-j|}, 1 \leq i, j \leq n$ ,

power balance: 
$$\underbrace{\sum_{i=1, i \neq j}^n P_{ji}}_{=I_j} - \sum_{i=1, i \neq j}^n P_{ij} = Q_j . \quad (8.1.15)$$



$Q_j \hat{=}$  heat flux through  $\Gamma_j$ , satisfies

$$Q_j := \int_{\frac{2\pi}{n}(j-1)}^{\frac{2\pi}{n}j} q(\varphi) d\varphi, \quad q(\varphi) := \begin{cases} \text{local heating} & , \text{ if } \varphi \in \Gamma_H, \\ -\frac{1}{|\Gamma_K|} \int_{\Gamma_H} q(\varphi) d\varphi & (\text{const.}), \text{ if } \varphi \in \Gamma_K. \end{cases}$$

(8.1.15)  $\Rightarrow$  LSE: 
$$I_j - \sum_{i=1, i \neq j}^n \frac{\alpha_{ij}}{\pi} I_i = Q_j, \quad j = 1, \dots, n .$$

$$n = 8: \begin{pmatrix} 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 \\ -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 \\ -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 \\ -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_2 & -\gamma_4 \\ -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 \\ -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 \\ -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 \\ -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \\ I_8 \end{pmatrix} = \begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \\ Q_7 \\ Q_8 \end{pmatrix}. \quad (8.1.16)$$

This is a linear system of equations with symmetric, singular, and (by Thm. 6.1.5,  $\sum \gamma_i \leq 1$ ) positive semidefinite ( $\rightarrow$  Def. 2.7.9) system matrix.

Note: matrices from (8.1.12) and (8.1.16) have the same structure !

Observe: LSE from (8.1.16) can be written by means of the discrete periodic convolution ( $\rightarrow$  Def. 8.1.13) of vectors  $\mathbf{y} = (1, -\gamma_1, -\gamma_2, -\gamma_3, -\gamma_4, -\gamma_3, -\gamma_2, -\gamma_1)$ ,  $\mathbf{x} = (I_1, \dots, I_8)$

$$(8.1.16) \Leftrightarrow \mathbf{y} *_8 \mathbf{x} = (Q_1, \dots, Q_8)^\top.$$

**Definition 8.1.17** (Circulant matrix).  $\rightarrow$  [35, Sect. 54]

A matrix  $\mathbf{C} = (c_{ij})_{i,j=1}^n \in \mathbb{K}^{n,n}$  is **circulant** (ger.: zirkulant)

$$:\Leftrightarrow \exists (u_k)_{k \in \mathbb{Z}} \text{ } n\text{-periodic sequence: } c_{ij} = u_{j-i}, 1 \leq i, j \leq n.$$

☞ Circulant matrix has constant (main, sub- and super-) diagonals (for which indices  $j - i = \text{const.}$ ).

☞ columns/rows arise by *cyclic permutation* from first column/row.

Similar to the case of banded matrices ( $\rightarrow$  Sect. 2.6.4):

“information content” of circulant matrix  $\mathbf{C} \in \mathbb{K}^{n,n}$  =  $n$  numbers  $\in \mathbb{K}$ .  
(obviously, one vector  $\mathbf{u} \in \mathbb{K}^n$  enough to define circulant matrix  $\mathbf{C} \in \mathbb{K}^{n,n}$ )

Structure of circulant matrix



$$\begin{pmatrix} u_0 & u_1 & u_2 & \cdots & \cdots & u_{n-1} \\ u_{n-1} & u_0 & & & & u_{n-2} \\ u_{n-2} & & & & & \vdots \\ \vdots & & & & & \vdots \\ \vdots & & & & & \vdots \\ u_2 & & & & & u_1 \\ u_1 & u_2 & \cdots & \cdots & u_{n-1} & u_0 \end{pmatrix}$$

Write  $\mathbf{Z}((u_k)) \in \mathbb{K}^{n,n}$  for the circulant matrix generated by the  $n$ -periodic sequence  $(u_k)_{k \in \mathbb{Z}}$ . Denote by  $\mathbf{y} := (y_0, \dots, y_{n-1})^\top$ ,  $\mathbf{x} = (x_0, \dots, x_{n-1})^\top$  the vectors associated to  $n$ -periodic sequences.

Then the commutativity of the discrete periodic convolution ( $\rightarrow$  Def. 8.1.13) ensures

$$\mathbf{Z}((x_k))\mathbf{y} = \mathbf{Z}((y_k))\mathbf{x} . \tag{8.1.18}$$

*Remark 8.1.19* (Reduction to periodic convolution).

Recall discrete convolution ( $\rightarrow$  Def. 8.1.7) of  $\mathbf{a} = (a_0, \dots, a_{n-1})^\top \in \mathbb{K}^n$ ,  $\mathbf{b} = (b_0, \dots, b_{n-1})^\top \in$

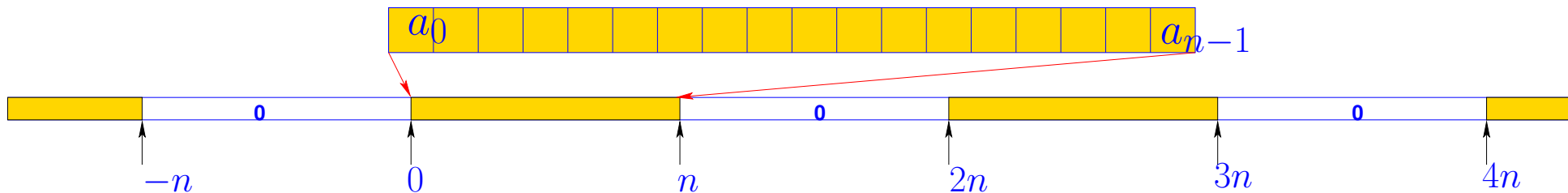


$$(\mathbf{a} * \mathbf{b})_k = \sum_{j=0}^{n-1} a_j b_{k-j}, \quad k = 0, \dots, 2n - 2.$$

Expand  $a_0, \dots, a_{n-1}$  and  $b_0, \dots, b_{n-1}$  to  $2n - 1$ -periodic sequences by **zero padding**:

$$x_k := \begin{cases} a_k & , \text{ if } 0 \leq k < n, \\ 0 & , \text{ if } n \leq k < 2n - 1 \end{cases}, \quad y_k := \begin{cases} b_k & , \text{ if } 0 \leq k < n, \\ 0 & , \text{ if } n \leq k < 2n - 1 \end{cases}, \quad (8.1.20)$$

and periodic extension:  $x_k = x_{2n-1+k}$ ,  $y_k = y_{2n-1+k}$  for all  $k \in \mathbb{Z}$ .



$$\blacktriangleright \quad (\mathbf{a} * \mathbf{b})_k = (\mathbf{x} *_{2n-1} \mathbf{y})_k, \quad k = 0, \dots, 2n - 2. \quad (8.1.21)$$

$$\begin{pmatrix} y_0 \\ \vdots \\ \vdots \\ y_{2n-2} \end{pmatrix} = \underbrace{\begin{pmatrix} h_0 & 0 & \cdots & 0 & h_{n-1} & \cdots & h_1 & h_0 \\ h_1 & & & 0 & & & & \\ \vdots & & & \vdots & & & & \\ h_{n-1} & \cdots & h_1 & h_0 & 0 & \cdots & 0 & h_{n-1} \\ 0 & & & h_1 & h_0 & & & \\ \vdots & & & \vdots & & & & \\ 0 & \cdots & 0 & h_{n-1} & \cdots & h_1 & h_0 & \end{pmatrix}}_{\text{a } (2n-1) \times (2n-1) \text{ circulant matrix!}} \begin{pmatrix} x_0 \\ \vdots \\ \vdots \\ x_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{8.1.22}$$



## 8.2 Discrete Fourier Transform (DFT)

*Example 8.2.1* (Eigenvectors of circulant matrices).

Now we are about to discover a very deep truth . . .

Code 8.2.2: Eigenvectors of random circulant matrices

```
1 function circeig
2 % Computing the eigenvectors of a random circulant matrix
3 n = 8;
4 % The MATLAB way to generate a circulant matrix
5 C = gallery ('circul', rand (n,1)); [V1,D1] = eig (C);
6
7 for j=1:n
8     figure; bar (1:n, [real (V1 (:, j)), imag (V1 (:, j))], 1, 'grouped');
9     title (sprintf ('Circulant matrix 1, eigenvector %d', j));
10    xlabel ('{\bf vector component index}', 'fontsize', 14);
11    ylabel ('{\bf vector component value}', 'fontsize', 14);
12    legend ('real part', 'imaginary part', 'location', 'southwest');
13    print ('-depsc2', sprintf ('../PICTURES/circeiglev%d.eps', j));
14 end
```

```
15
16 C = gallery('circul', rand(n,1)); [V2,D2] = eig(C);
17
18 for j=1:n
19     figure; bar(1:n, [real(V2(:,j)), imag(V2(:,j))], 1, 'grouped');
20     title(sprintf('Circulant matrix 2, eigenvector %d', j));
21     xlabel('\bf vector component index', 'fontsize', 14);
22     ylabel('\bf vector component value', 'fontsize', 14);
23     legend('real part', 'imaginary part', 'location', 'southwest');
24     print('-depsc2', sprintf('../PICTURES/circeig2ev%d.eps', j));
25 end
26
27 figure; plot(1:n, real(diag(D1)), 'r+', 1:n, imag(diag(D1)), 'b+', ...
28             1:n, real(diag(D2)), 'm*', 1:n, imag(diag(D2)), 'k*');
29 ax = axis; axis([0 n+1 ax(3) ax(4)]);
30 xlabel('\bf index of eigenvalue', 'fontsize', 14);
31 ylabel('\bf eigenvalue', 'fontsize', 14);
32 legend('C_1: real(ev)', 'C_1: imag(ev)', 'C_2: real(ev)', 'C_2:
33         imag(ev)', 'location', 'northeast');
34 print -depsc2 '../PICTURES/circeigev.eps';
```

Random  $8 \times 8$  circulant matrices  $C_1, C_2$  ( $\rightarrow$   
Def. 8.1.17)

eigenvalues (real part)  $\triangleright$

Generated by MATLAB-command:

```
C = gallery('circul',rand(n,1));
```

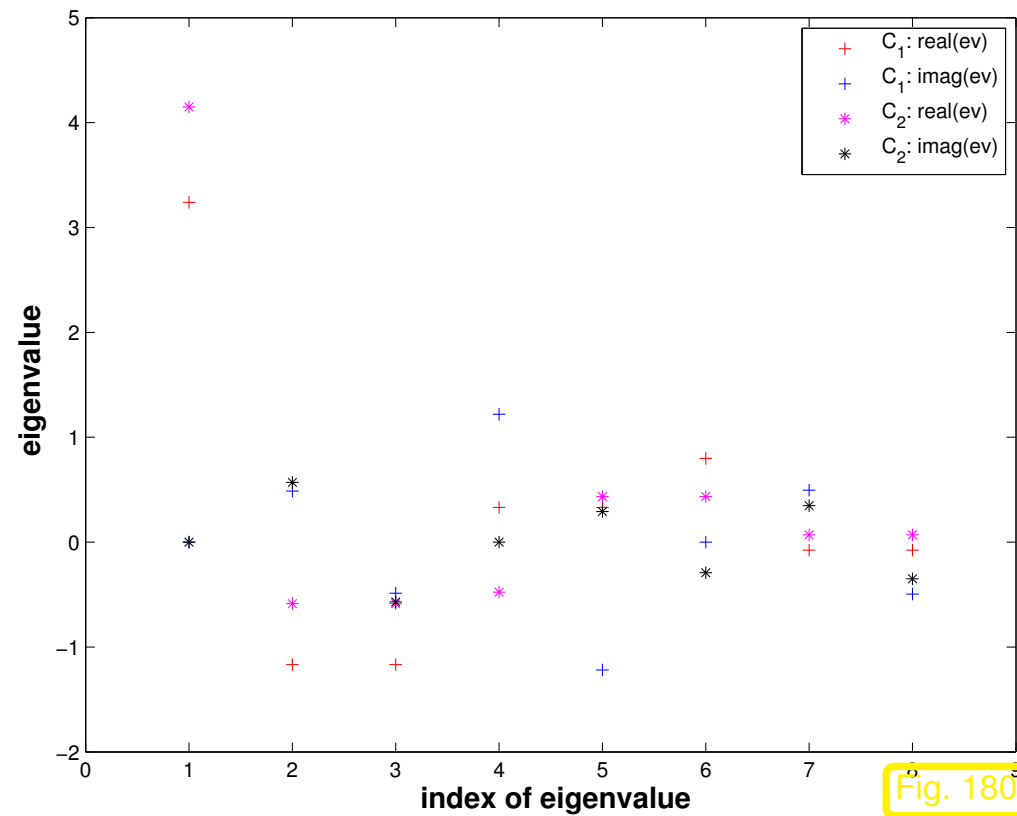
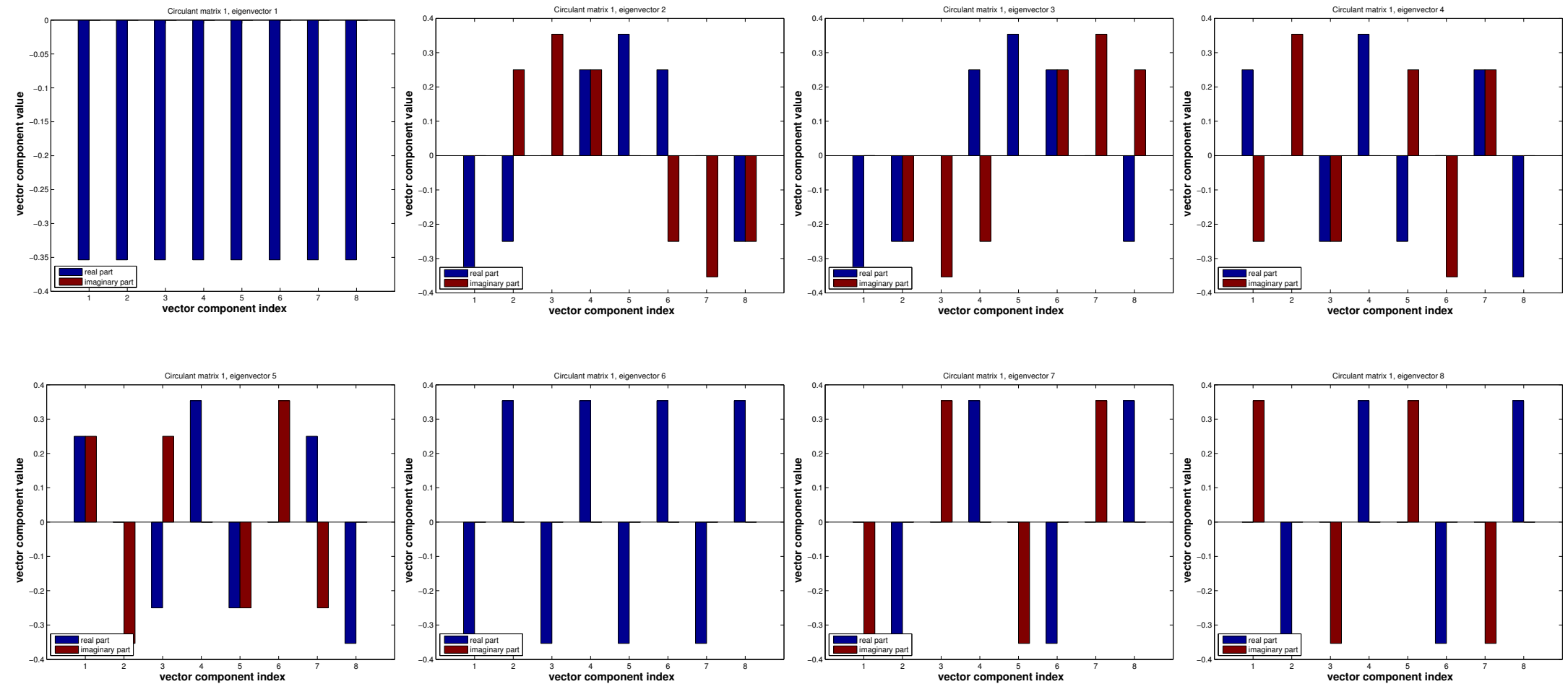


Fig. 180

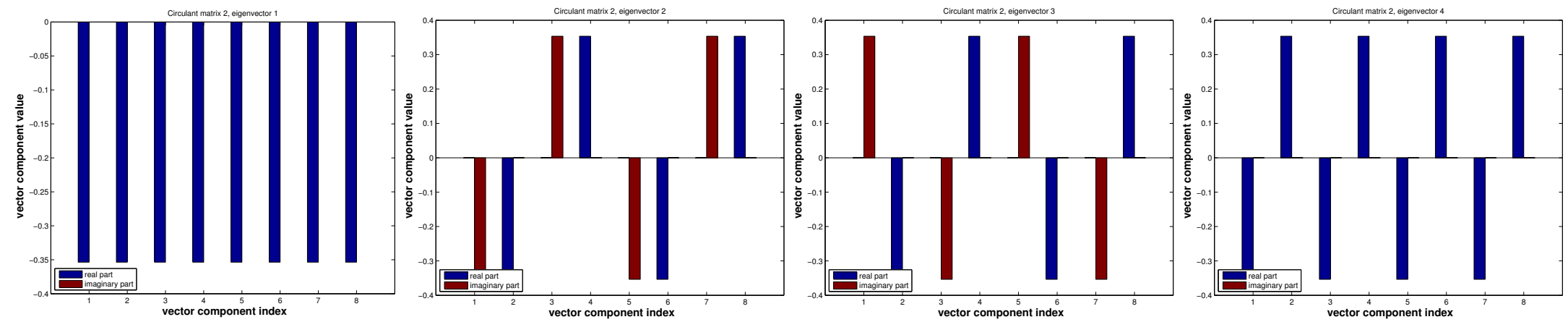
Little relationship between (complex!) eigenvalues can be observed, as can be expected from random matrices with entries  $\in [0, 1]$ .

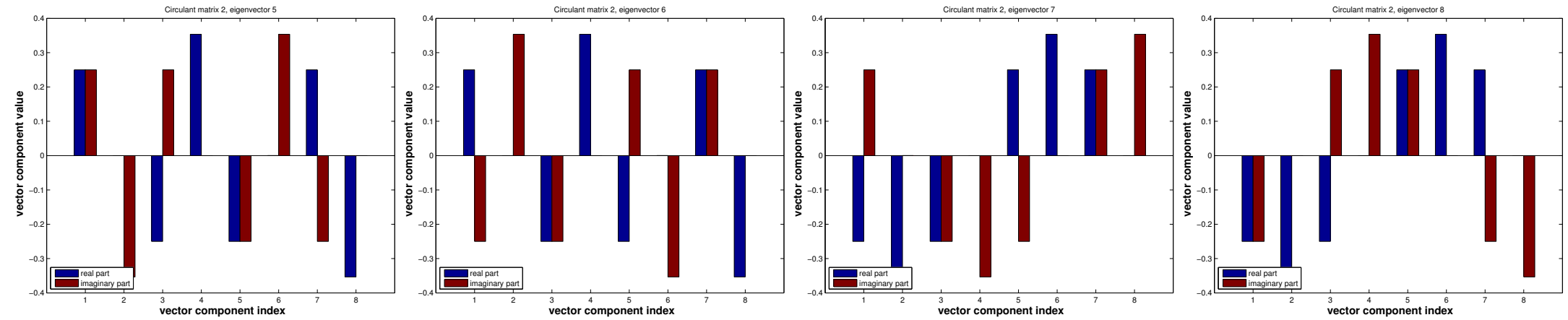
Now: the **surprise** . . .

Eigenvectors of matrix  $C_1$ :



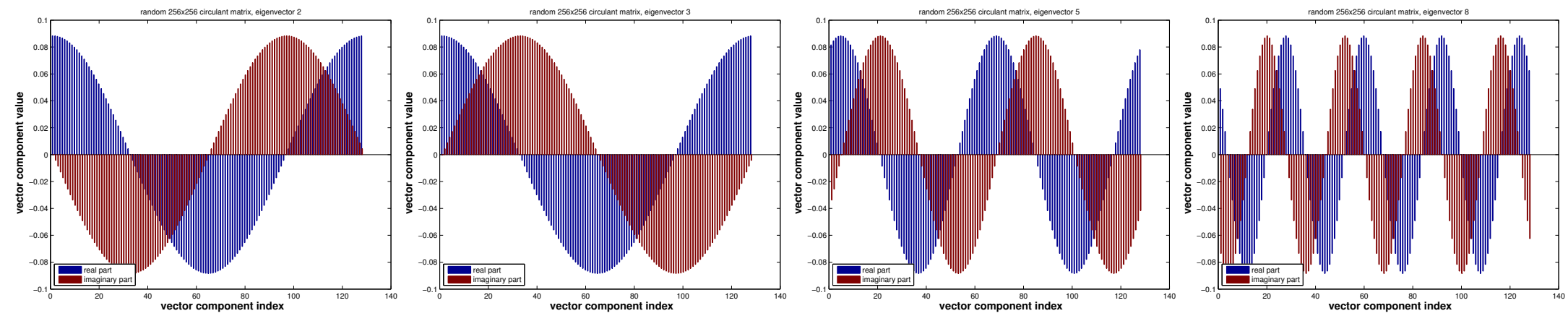
## Eigenvectors of matrix $C_2$





Observation: different random circulant matrices have the **same eigenvectors!**

Eigenvectors of `C = gallery('circul',(1:128)); :`



The eigenvectors remind us of sampled *trigonometric functions*  $\cos(k/n), \sin(k/n), k = 0, \dots, n-1!$



*Remark 8.2.3* (Why using  $\mathbb{K} = \mathbb{C}$ ?).

Ex. 8.2.1: *complex* eigenvalues/eigenvectors for general circulant matrices.

R. Hiptmair  
rev 38286,  
November  
17, 2011

Recall from analysis: unified treatment of trigonometric functions via *complex exponential function*

$$\exp(it) = \cos(t) + i \sin(t), \quad t \in \mathbb{R}.$$

**C!** The field of complex numbers  $\mathbb{C}$  is the *natural framework* for the analysis of linear, time-invariant filters, and the development of algorithms for circulant matrices.





notation:  $n$ th root of unity  $\omega_n := \exp(-2\pi i/n) = \cos(2\pi/n) - i \sin(2\pi/n)$ ,  $n \in \mathbb{N}$

satisfies  $\bar{\omega}_n = \omega_n^{-1}$ ,  $\omega_n^n = 1$ ,  $\omega_n^{n/2} = -1$ ,  $\omega_n^k = \omega_n^{k+n} \quad \forall k \in \mathbb{Z}$ , (8.2.4)

$$\sum_{k=0}^{n-1} \omega_n^{kj} = \begin{cases} n & , \text{ if } j = 0 \pmod n, \\ 0 & , \text{ if } j \neq 0 \pmod n. \end{cases} \quad (8.2.5)$$

(8.2.5) is a simple consequence of the geometric sum formula

$$\sum_{k=0}^{n-1} q^k = \frac{1 - q^n}{1 - q} \quad \forall q \in \mathbb{C} \setminus \{1\}, \quad n \in \mathbb{N}. \quad (8.2.6)$$

$$\Rightarrow \sum_{k=0}^{n-1} \omega_n^{kj} = \frac{1 - \omega_n^{nj}}{1 - \omega_n^j} = \frac{1 - \exp(-2\pi i j)}{1 - \exp(-2\pi i j/n)} = 0,$$

because  $\exp(-2\pi i j) = \omega_n^{nj} = (\omega_n^n)^j = 1$  for all  $j \in \mathbb{Z}$ .

Now we want to confirm the conjecture gleaned from Ex. 8.2.1 that vectors with powers of roots of unity are eigenvectors for any circulant matrix. We do this by simple and straightforward computations:

Consider:  $\mathbf{C} \in \mathbb{C}^{n,n}$  circulant matrix ( $\rightarrow$  Def. 8.1.17),  $c_{ij} = u_{i-j}$ , for  $n$ -periodic sequence  $(u_k)_{k \in \mathbb{Z}}$ ,  $u_k \in \mathbb{C}$

$\mathbf{v}_k \in \mathbb{C}^n$  with  $\mathbf{v}_k := (\omega_n^{jk})_{j=0}^{n-1} \in \mathbb{C}^n$ ,  $k \in \{0, \dots, n-1\}$ .

$(u_{j-l} \omega_n^{lk})_{l \in \mathbb{Z}}$  is  $n$ -periodic!

$$(\mathbf{C}\mathbf{v}_k)_j = \sum_{l=0}^{n-1} u_{j-l} \omega_n^{lk} = \sum_{l=j-n+1}^j u_{j-l} \omega_n^{lk} \tag{8.2.7}$$

$$= \sum_{l=0}^{n-1} u_l \omega_n^{(j-l)k} = \omega_n^{jk} \left[ \sum_{l=0}^{n-1} u_l \omega_n^{-lk} \right] = \lambda_k \cdot \omega_n^{jk} = \lambda_k \cdot (\mathbf{v}_k)_j$$

change of summation index

independent of  $j$  !

$\mathbf{v}$  is eigenvector of  $\mathbf{C}$  to eigenvalue  $\lambda_k = \sum_{l=0}^{n-1} u_l \omega_n^{-lk}$ .

Orthogonal trigonometric basis of  $\mathbb{C}^n$  = eigenvector basis for circulant matrices

$$\left\{ \begin{pmatrix} \omega_n^0 \\ \vdots \\ \omega_n^0 \end{pmatrix}, \begin{pmatrix} \omega_n^0 \\ \omega_n^1 \\ \vdots \\ \omega_n^{n-1} \end{pmatrix}, \dots, \begin{pmatrix} \omega_n^0 \\ \omega_n^{n-2} \\ \omega_n^{2(n-2)} \\ \vdots \\ \omega_n^{(n-1)(n-2)} \end{pmatrix}, \begin{pmatrix} \omega_n^0 \\ \omega_n^{n-1} \\ \omega_n^{2(n-1)} \\ \vdots \\ \omega_n^{(n-1)^2} \end{pmatrix} \right\}.$$

(8.2.5)  $\Rightarrow$  orthogonality of basis vectors:

$$\mathbf{v}_k := (\omega_n^{jk})_{j=0}^{n-1} \in \mathbb{C}^n: \quad \mathbf{v}_k^H \mathbf{v}_m = \sum_{j=0}^{n-1} \omega_n^{-jk} \omega_n^{jm} = \sum_{j=0}^{n-1} \omega_n^{(m-k)j} \stackrel{(8.2.5)}{=} 0, \text{ if } k \neq m. \quad (8.2.8)$$

R. Hiptmair  
rev 38286,  
November  
17, 2011

Matrix of change of basis trigonometrical basis  $\rightarrow$  standard basis: **Fourier-matrix**

$$\mathbf{F}_n = \begin{pmatrix} \omega_n^0 & \omega_n^0 & \dots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \dots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \dots & \omega_n^{2(n-2)} \\ \vdots & \vdots & & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \dots & \omega_n^{(n-1)^2} \end{pmatrix} = \left( \omega_n^{lj} \right)_{l,j=0}^{n-1} \in \mathbb{C}^{n,n}. \quad (8.2.9)$$

**Lemma 8.2.10** (Properties of Fourier matrix).

The scaled Fourier-matrix  $\frac{1}{\sqrt{n}}\mathbf{F}_n$  is unitary ( $\rightarrow$  Def. 2.8.5):  $\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^H = \frac{1}{n}\overline{\mathbf{F}_n}$ .

*Proof.* The lemma is immediate from (8.2.8) and (8.2.5), because

$$\left(\mathbf{F}_n\mathbf{F}_n^H\right)_{l,j} = \sum_{k=0}^{n-1} \omega_n^{(l-1)k} \overline{\omega_n^{(j-1)k}} = \sum_{k=0}^{n-1} \omega_n^{(l-1)k} \omega_n^{-(j-1)k} = \sum_{k=0}^{n-1} \omega_n^{k(l-j)}, \quad 1 \leq l, j \leq n.$$

*Remark 8.2.11* (Spectrum of Fourier matrix).

$$\frac{1}{n^2}\mathbf{F}_n^4 = I \quad \Rightarrow \quad \sigma\left(\frac{1}{\sqrt{n}}\mathbf{F}_n\right) \subset \{1, -1, i, -i\},$$

because, if  $\lambda \in \mathbb{C}$  is an eigenvalue of  $\mathbf{F}_n$ , then there is an eigenvector  $\mathbf{x} \in \mathbb{C}^n \setminus \{0\}$  such that  $\mathbf{F}_n\mathbf{x} = \lambda\mathbf{x}$ , see Def. 6.1.1.

△

**Lemma 8.2.12** (Diagonalization of circulant matrices ( $\rightarrow$  Def. 8.1.17)).

For any circulant matrix  $\mathbf{C} \in \mathbb{K}^{n,n}$ ,  $c_{ij} = u_{i-j}$ ,  $(u_k)_{k \in \mathbb{Z}}$   $n$ -periodic sequence, holds true

$$\mathbf{C}\bar{\mathbf{F}}_n = \bar{\mathbf{F}}_n \text{diag}(d_1, \dots, d_n) \quad , \quad \mathbf{d} = \mathbf{F}_n(u_0, \dots, u_{n-1})^\top .$$

*Proof.* Straightforward computation, see (8.2.7). □

Conclusion (from  $\bar{\mathbf{F}}_n = n\mathbf{F}_n^{-1}$ ):

$$\mathbf{C} = \mathbf{F}_n^{-1} \text{diag}(d_1, \dots, d_n) \mathbf{F}_n . \quad (8.2.13)$$

Lemma 8.2.12, (8.2.13)  $\Rightarrow$  multiplication with Fourier-matrix will be crucial operation in algorithms for circulant matrices and discrete convolutions.

Therefore this operation has been given a special name:

**Definition 8.2.14** (Discrete Fourier transform (DFT)).

The linear map  $\mathcal{F}_n : \mathbb{C}^n \mapsto \mathbb{C}^n$ ,  $\mathcal{F}_n(\mathbf{y}) := \mathbf{F}_n \mathbf{y}$ ,  $\mathbf{y} \in \mathbb{C}^n$ , is called *discrete Fourier transform (DFT)*, i.e. for  $\mathbf{c} := \mathcal{F}_n(\mathbf{y})$

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{kj} \quad , \quad k = 0, \dots, n-1 . \quad (8.2.15)$$

Recall the convention also adopted for the discussion of the DFT: vector indexes range from 0 to  $n-1$ !

Terminology:  $\mathbf{c} = \mathbf{F}_n \mathbf{y}$  is also called the (discrete) Fourier transform of  $\mathbf{y}$

From  $\mathbf{F}_n^{-1} = \frac{1}{n} \overline{\mathbf{F}_n}$  ( $\rightarrow$  Lemma 8.2.10) we find the inverse discrete Fourier transform

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{kj} \quad \Leftrightarrow \quad y_j = \frac{1}{n} \sum_{k=0}^{n-1} c_k \omega_n^{-kj} \quad (8.2.16)$$

MATLAB-functions for discrete Fourier transform (and its inverse):

$$\text{DFT: } c = \text{fft}(y) \quad \leftrightarrow \quad \text{inverse DFT: } y = \text{ifft}(c);$$

## 8.2.1 Discrete convolution via DFT

Recall discrete periodic convolution  $z_k = \sum_{j=0}^{n-1} u_{k-j} x_j$  ( $\rightarrow$  Def. 8.1.13),  $k = 0, \dots, n-1$

$$\updownarrow$$

multiplication with circulant matrix ( $\rightarrow$  Def. 8.1.17)  $\mathbf{z} = \mathbf{C}\mathbf{x}$ ,  $\mathbf{C} := (u_{i-j})_{i,j=1}^n$ .

Idea: (8.2.13)  $\triangleright \mathbf{z} = \mathbf{F}_n^{-1} \text{diag}(\mathbf{F}_n \mathbf{u}) \mathbf{F}_n \mathbf{x}$

Code 8.2.18: discrete periodic convolution:  
straightforward implementation

```

1 function z=pconv(u,x)
2 n = length(x); z = zeros(n,1);
3 for i=1:n, z(i)=dot(conj(u),
   x([i:-1:1,n:-1:i+1]));
4 end

```

Code 8.2.20: discrete periodic convolution:  
DFT implementation

```

1 function z=pconvfft(u,x)
2 % Implementation of (8.2.13), cf.
   % Lemma 8.2.12
3 z = ifft(fft(u) .* fft(x));

```

Rem. 8.1.19: discrete convolution of  $n$ -vectors ( $\rightarrow$  Def. 8.1.7) by *periodic* discrete convolution of  $2n - 1$ -vectors (obtained by zero padding, see Rem. 8.1.19):

Implementation of  
discrete convolution ( $\rightarrow$   
Def. 8.1.7) based on  
periodic discrete convolution  $\triangleright$

Built-in MATLAB-function:

`y = conv(h,x);`

Code 8.2.21: discrete convolution: DFT implementation

```

1 function y = myconv(h,x)
2 n = length(h);
3 % Zero padding, cf. (8.1.20)
4 h = [h;zeros(n-1,1)];
5 x = [x;zeros(n-1,1)];
6 % Periodic discrete convolution of length 2n - 1
7 y = pconvfft(h,x);

```



## 8.2.2 Frequency filtering via DFT

The trigonometric basis vectors, when interpreted as time-periodic signals, represent harmonic oscillations. This is illustrated when plotting some vectors of the trigonometric basis ( $n = 16$ ):

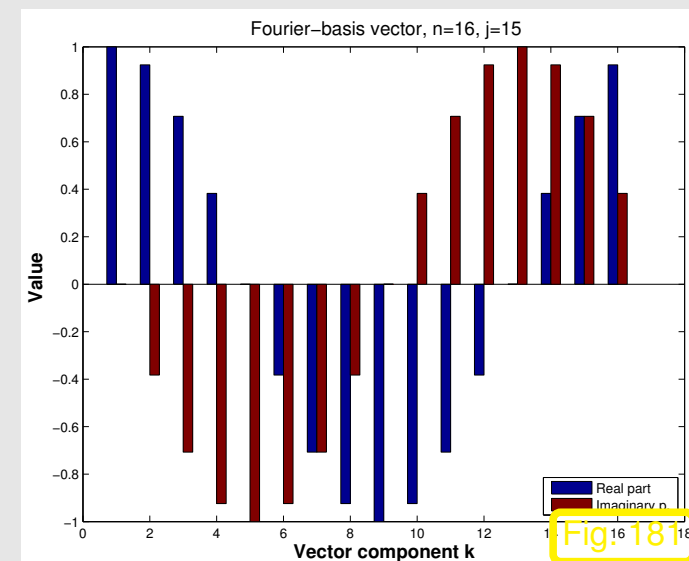
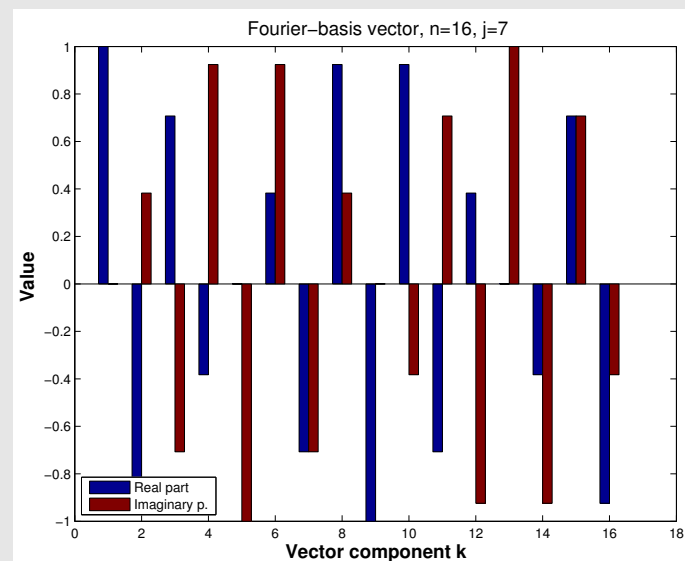
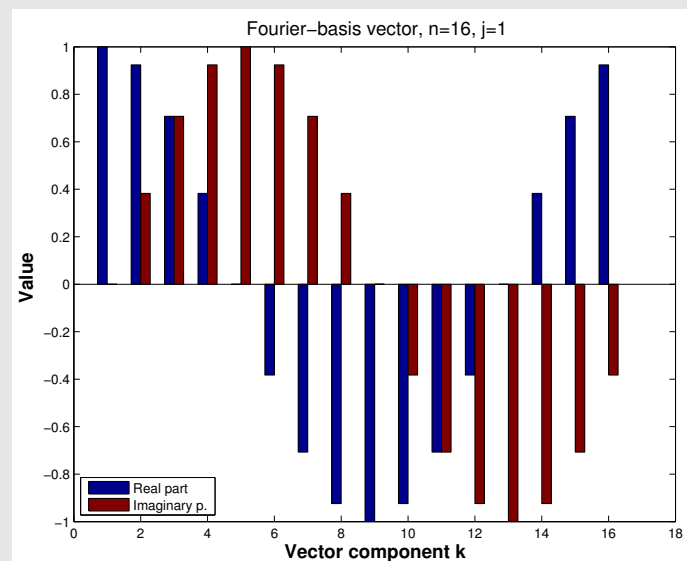


Fig 16 18 16

“slow oscillation/low frequency”    “fast oscillation/high frequency”    “slow oscillation/low frequency”

► Dominant coefficients of a signal after transformation to trigonometric basis indicate dominant frequency components.

Terminology: coefficients of a signal w.r.t. trigonometric basis = signal in **frequency domain** (ger.: Frequenzbereich), original signal = **time domain** (ger.: Zeitbereich).

Recall: DFT (8.2.15) and inverse DFT (8.2.16)

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{kj} \quad \Leftrightarrow \quad y_j = \frac{1}{n} \sum_{k=0}^{n-1} c_k \omega_n^{-kj} \quad (8.2.16)$$

Consider  $y_k \in \mathbb{R} \Rightarrow c_k = \bar{c}_{n-k}$ , because  $\omega_n^{kj} = \bar{\omega}_n^{(n-k)j}$ , and  $n = 2m + 1$

$$\begin{aligned} \blacktriangleright \quad ny_j &= c_0 + \sum_{k=1}^m c_k \omega_n^{-kj} + \sum_{k=m+1}^{2m} c_k \omega_n^{-kj} = c_0 + \sum_{k=1}^m c_k \omega_n^{-kj} + c_{n-k} \omega_n^{(k-n)j} \\ &= c_0 + 2 \sum_{k=1}^m \operatorname{Re}(c_k) \cos(2\pi kj/n) + \operatorname{Im}(c_k) \sin(2\pi kj/n), \end{aligned}$$

since  $\omega_n^\ell = \cos(2\pi\ell/n) + i \sin(2\pi\ell/n)$ .

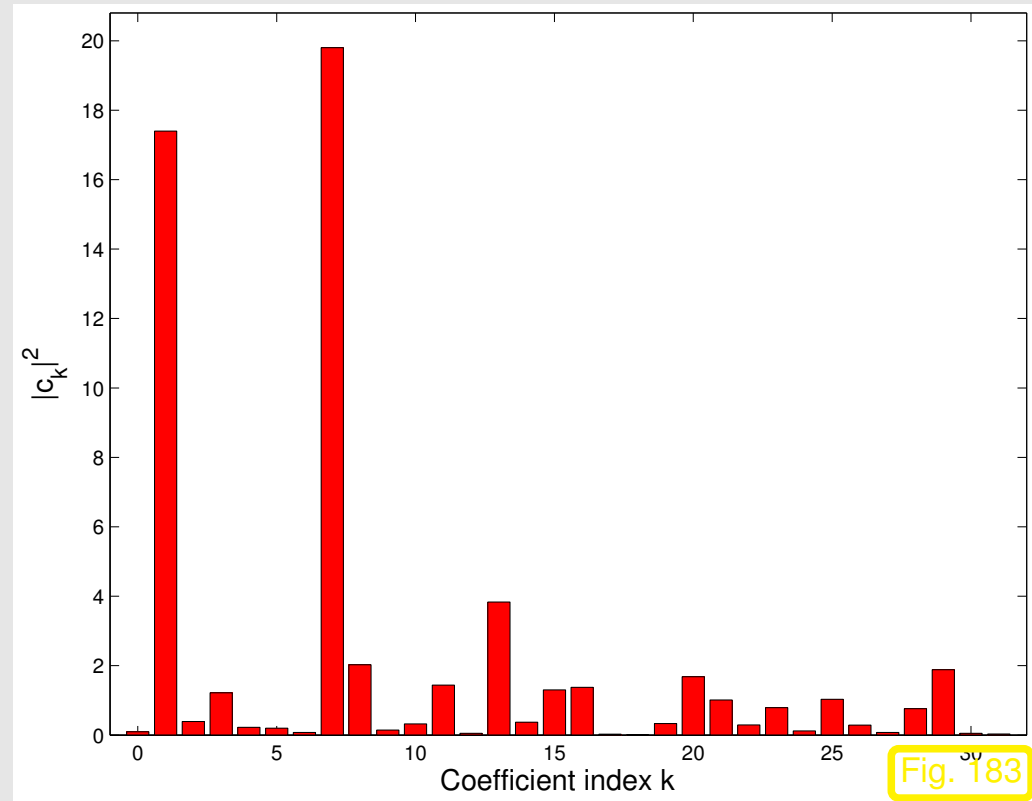
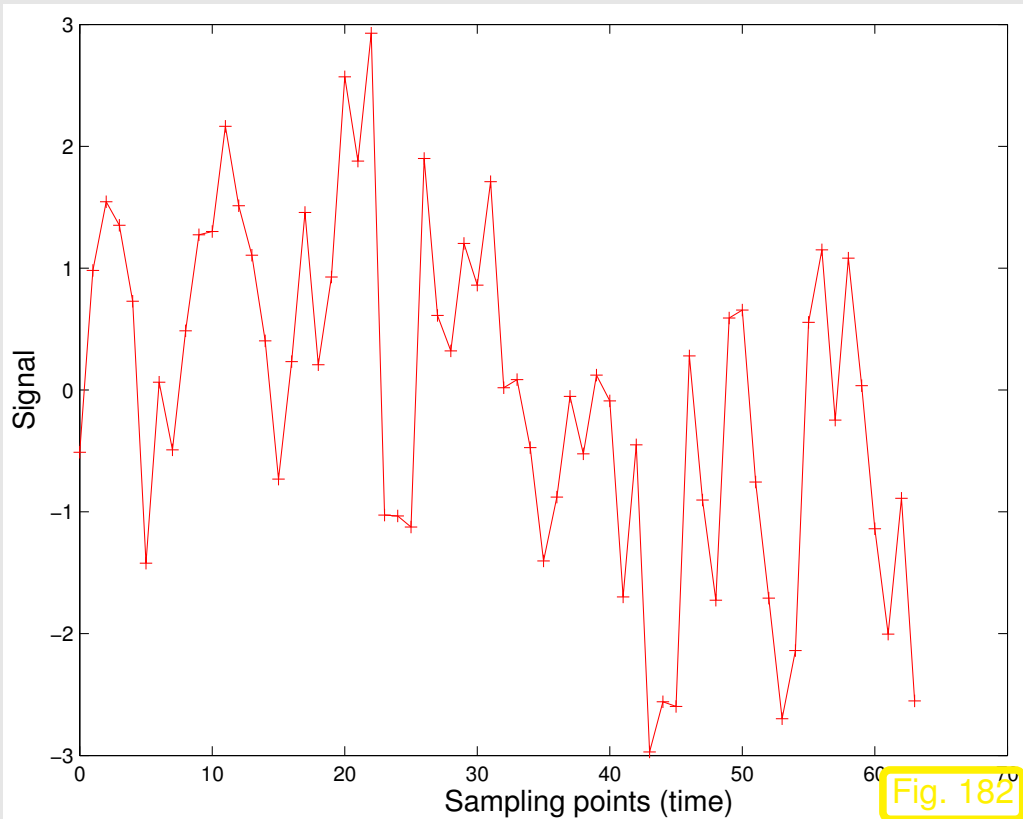
➤  $|c_k|, |c_{n-k}|$  measures the strength with which an oscillation with frequency  $k$  is represented in the signal,  $0 \leq k \leq \lfloor \frac{n}{2} \rfloor$ .

Example 8.2.22 (Frequency identification with DFT).

Extraction of characteristic frequencies from a distorted discrete periodical signal:

```

1 t = 0:63; x = sin(2*pi*t/64) + sin(7*2*pi*t/64);
2 y = x + randn(size(t)); %distortion
    
```



```

1 c = fft (y); p = (c.*conj (c)) / 64;
2
3 figure ('Name', 'power spectrum');
4 bar (0:31, p (1:32), 'r');
5 set (gca, 'fontsize', 14);
6 axis ([-1 32 0 max (p)+1]);
7 xlabel ('{\bf index k of Fourier coefficient}', 'FontSize', 14);
8 ylabel ('{\bf |c_k|^2}', 'FontSize', 14);

```

Frequencies present in unperturbed signal become evident in frequency domain.



*Example 8.2.23 (Detecting periodicity in data).*

DFT: a computer's eye for periodic patterns in data

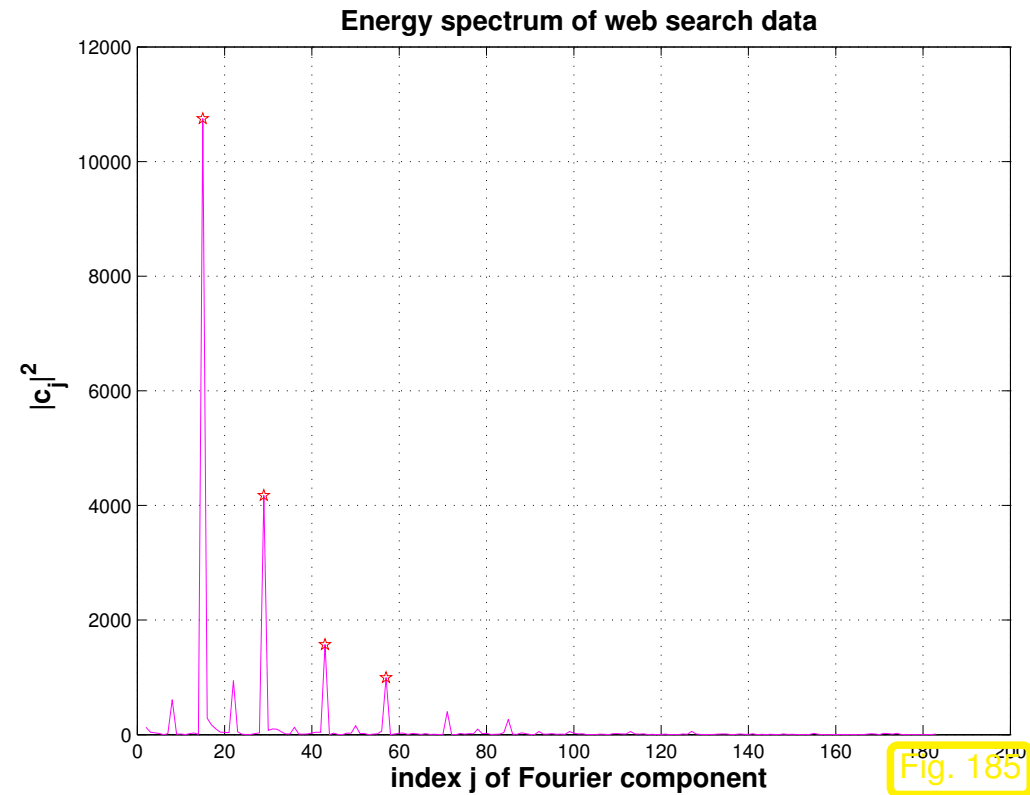
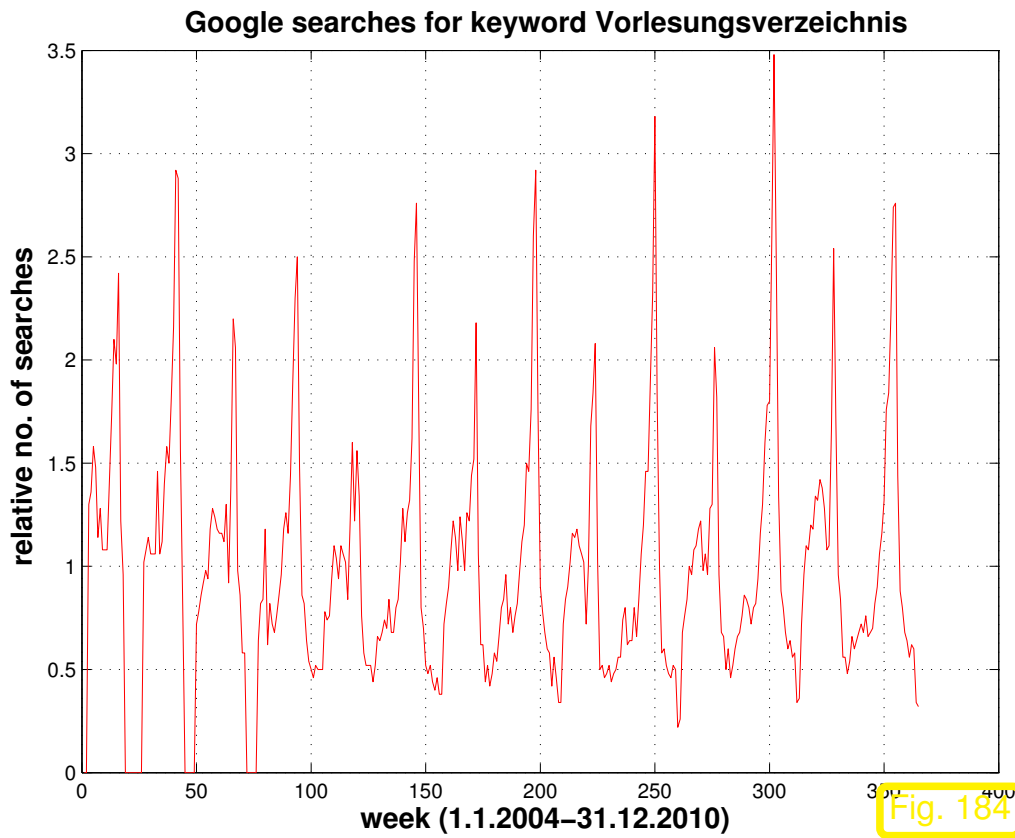
### Code 8.2.24: Extraction of periodic patterns by DFT

```

1 % Tracking of periodicity in data
2 % Data obtained from Google Trends, keyword "Vorlesungsverzeichnis"
3 % Exported as .csv-file, non data lines removed, preprocessed by command cut -f
  2 -d,

```

```
4
5 % read ASCII data from file
6 y = dlmread ('trend.dat'); n = length (y);
7
8 figure ('name', 'data');
9 plot (y, 'r-'); grid on;
0 title ('{\bf Google searches for keyword
   Vorlesungsverzeichnis}', 'fontsize', 14);
1 xlabel ('{\bf week (1.1.2004-31.12.2010) sa}', 'fontsize', 14);
2 ylabel ('{\bf relative no. of searches}', 'fontsize', 14);
3 print -depsc2 '../PICTURES/searchdata.eps';
4
5 % Periodicity analysis by means of DFT
6 c = fft (y);
7 p = abs (c(2:floor ((n+1)/2))) .^2; % Power spectrum
8 figure ('name', 'Fourier spectrum');
9 plot (2:floor ((n+1)/2), p, 'm-'); grid on; hold on;
0 [mx, idx] = sort (p, 'descend');
1 plot (1+idx(1:4), p(idx(1:4)), 'rp');
2 xlabel ('{\bf index j of Fourier component}', 'fontsize', 14);
3 ylabel ('{\bf |c_j|^2}', 'fontsize', 14);
4 title ('{\bf Energy spectrum of web search data}', 'fontsize', 14);
5 print -depsc2 '../PICTURES/fourierdata.eps';
```

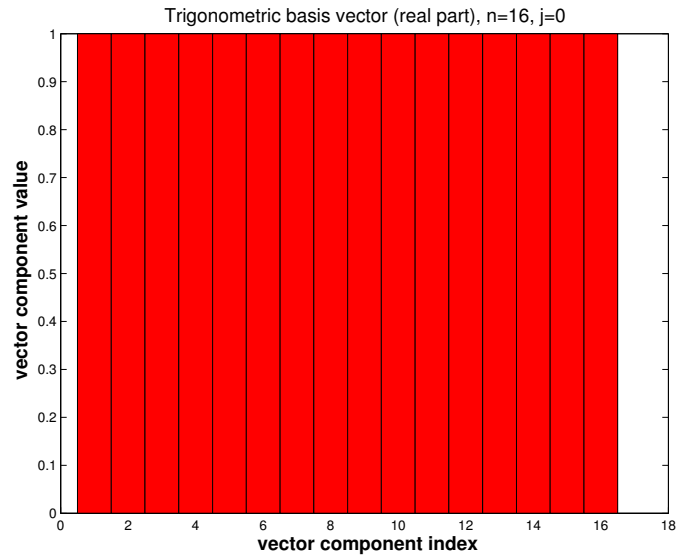


Pronounced peaks in the power spectrum point to periodic structure of the data. Location of peaks tells lengths of dominant periods.

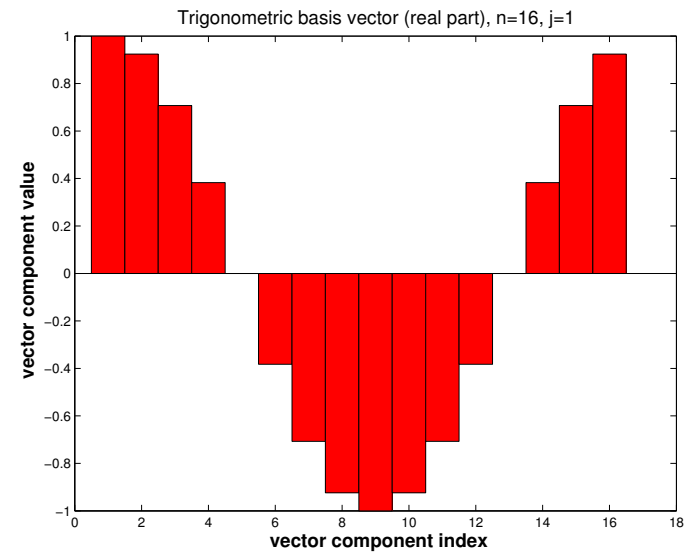


Remark 8.2.25 (“Low” and “high” frequencies).

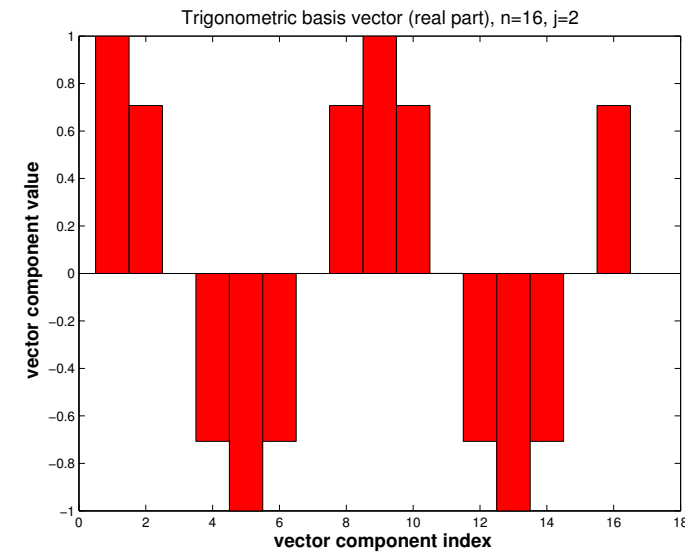
Plots of real parts of trigonometric basis vectors  $(\mathbf{F}_n)_{:,j}$  (= columns of Fourier matrix),  $n = 16$ .



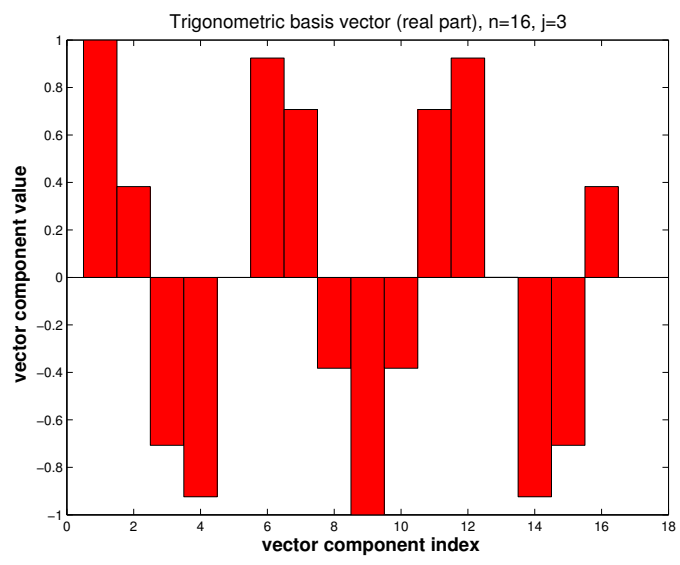
$\text{Re}(\mathbf{F}_{16})_{:,0}$



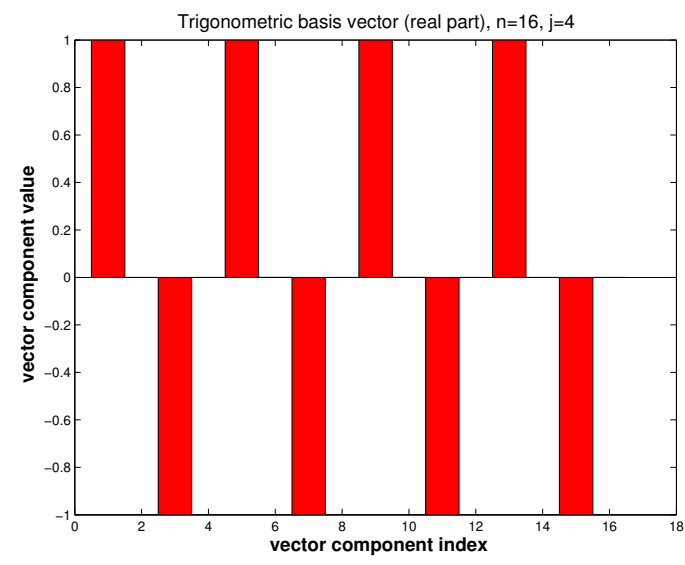
$\text{Re}(\mathbf{F}_{16})_{:,1}$



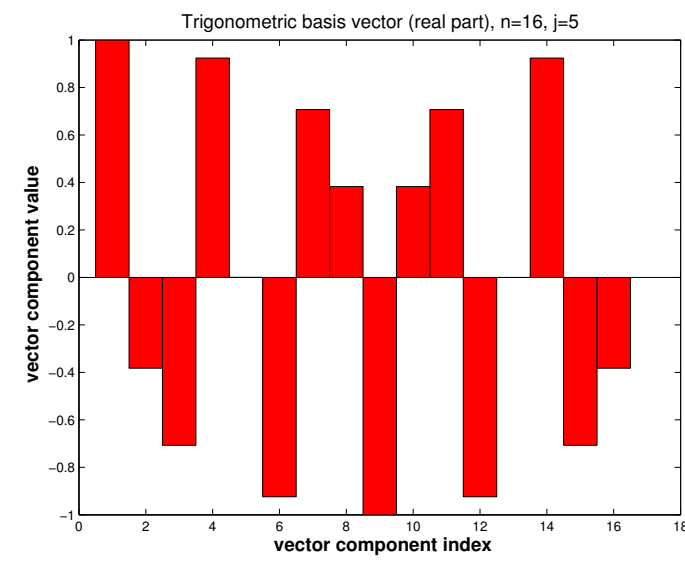
$\text{Re}(\mathbf{F}_{16})_{:,2}$



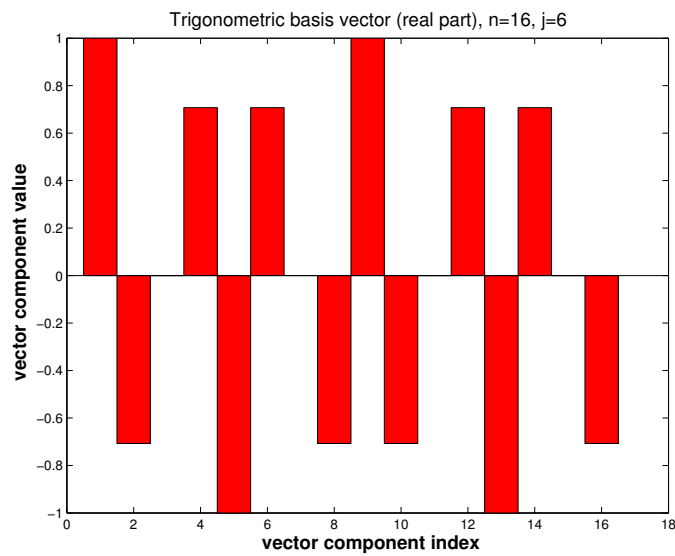
$\text{Re}(\mathbf{F}_{16})_{:,3}$



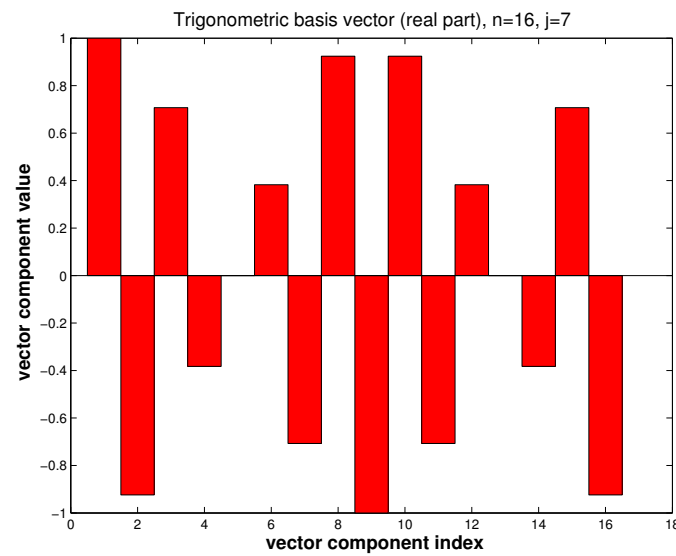
$\text{Re}(\mathbf{F}_{16})_{:,4}$



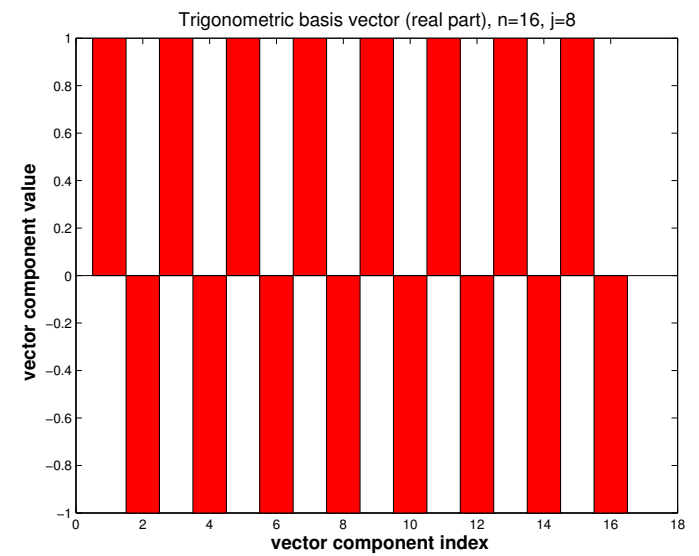
$\text{Re}(\mathbf{F}_{16})_{:,5}$



$\text{Re}(\mathbf{F}_{16})_{:,6}$



$\text{Re}(\mathbf{F}_{16})_{:,7}$



$\text{Re}(\mathbf{F}_{16})_{:,8}$



By elementary trigonometric identities:

$$\text{Re}(\mathbf{F}_n)_{:,j} = \left( \text{Re} \omega_n^{(j-1)k} \right)_{k=0}^{n-1} = \left( \text{Re} \exp(2\pi i(j-1)k/n) \right)_{k=0}^{n-1} = \left( \cos(2\pi(j-1)x) \right)_{x=0, \frac{1}{n}, \dots, 1-\frac{1}{n}} \cdot$$

Slow oscillations/low frequencies  $\leftrightarrow j \approx 1$  and  $j \approx n$ .

Fast oscillations/high frequencies  $\leftrightarrow j \approx n/2$ .



► Frequency filtering of real discrete periodic signals by suppressing certain “**Fourier coefficients**”.

Code 8.2.26: DFT-based frequency filtering

```

1 function [low,high] =
   freqfilter(y,k)
2 m = length(y)/2; c = fft(y);
3 c_low = c; c_low(m+1-k:m+1+k) = 0;
4 c_high = c-c_low;
5 low = real(ifft(c_low));
6 high = real(ifft(c_high));

```

(can be optimised exploiting  $y_j \in \mathbb{R}$  and  $c_{n/2-k} = \bar{c}_{n/2+k}$ )

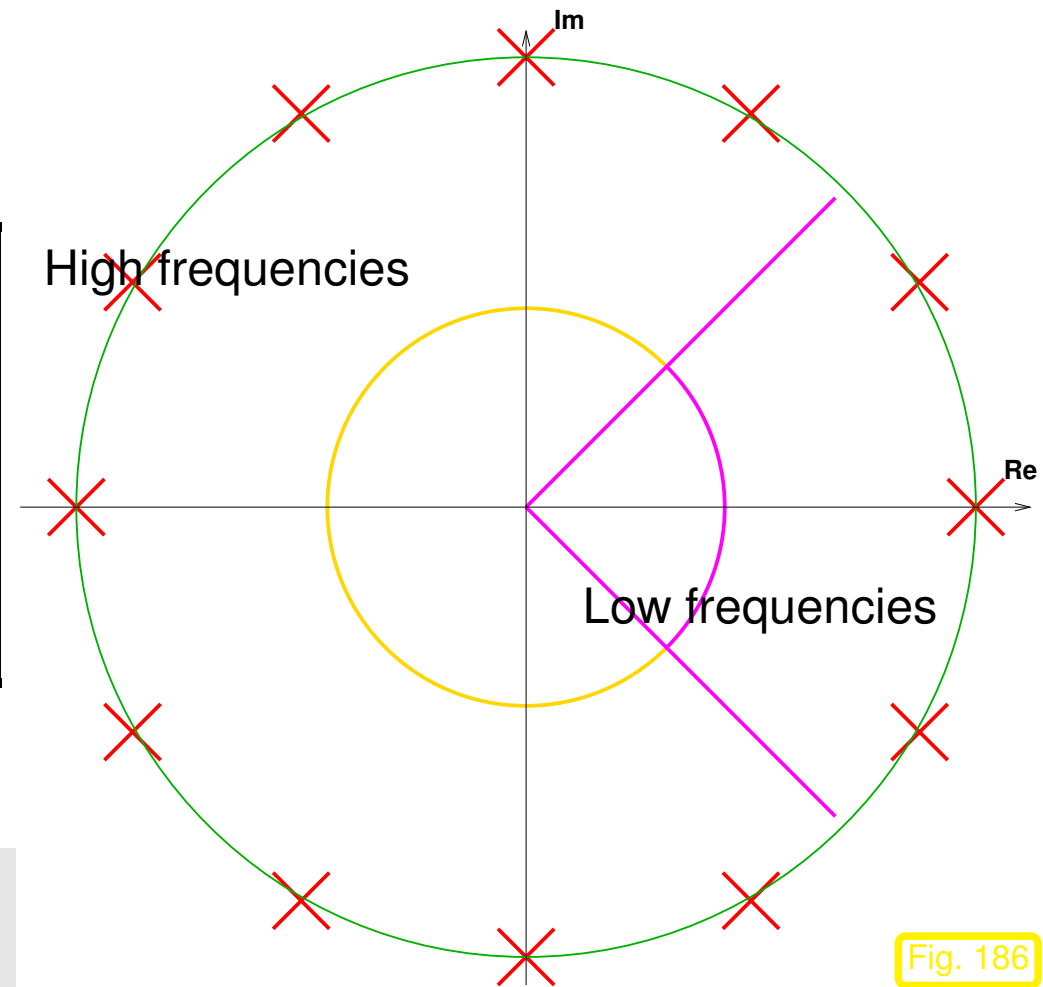


Fig. 186

Map  $y \mapsto \text{low}$  (in Code 8.2.25)  $\hat{=}$  **low pass filter** (*ger.*: Tiefpass).

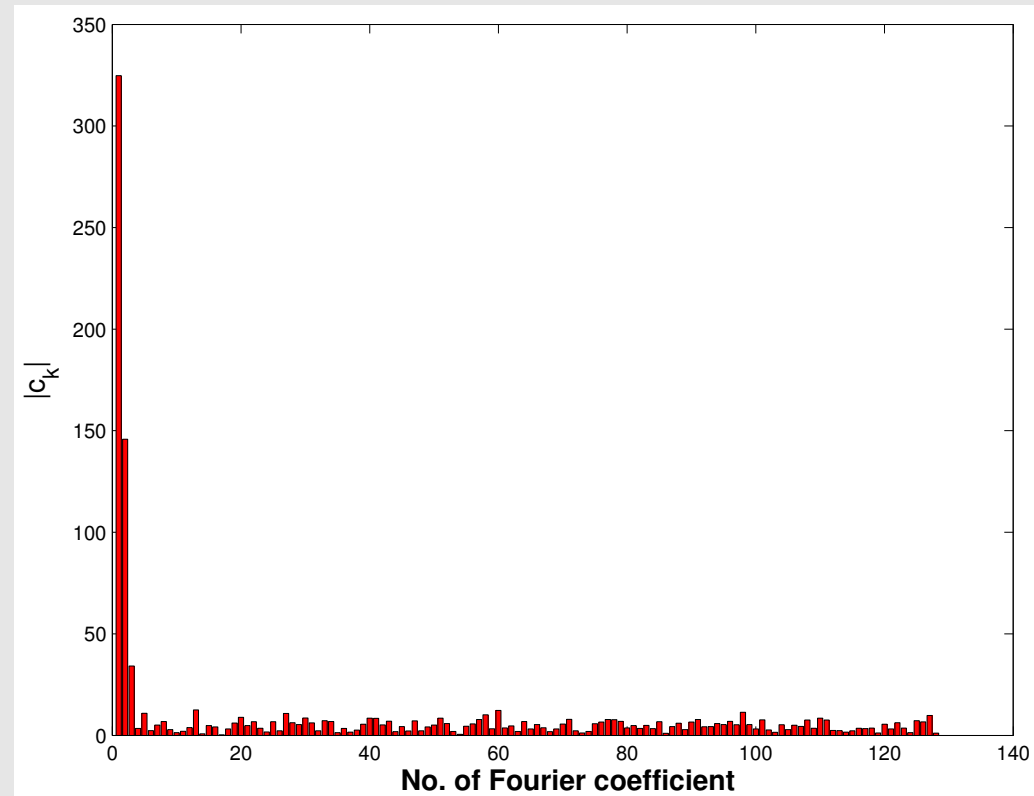
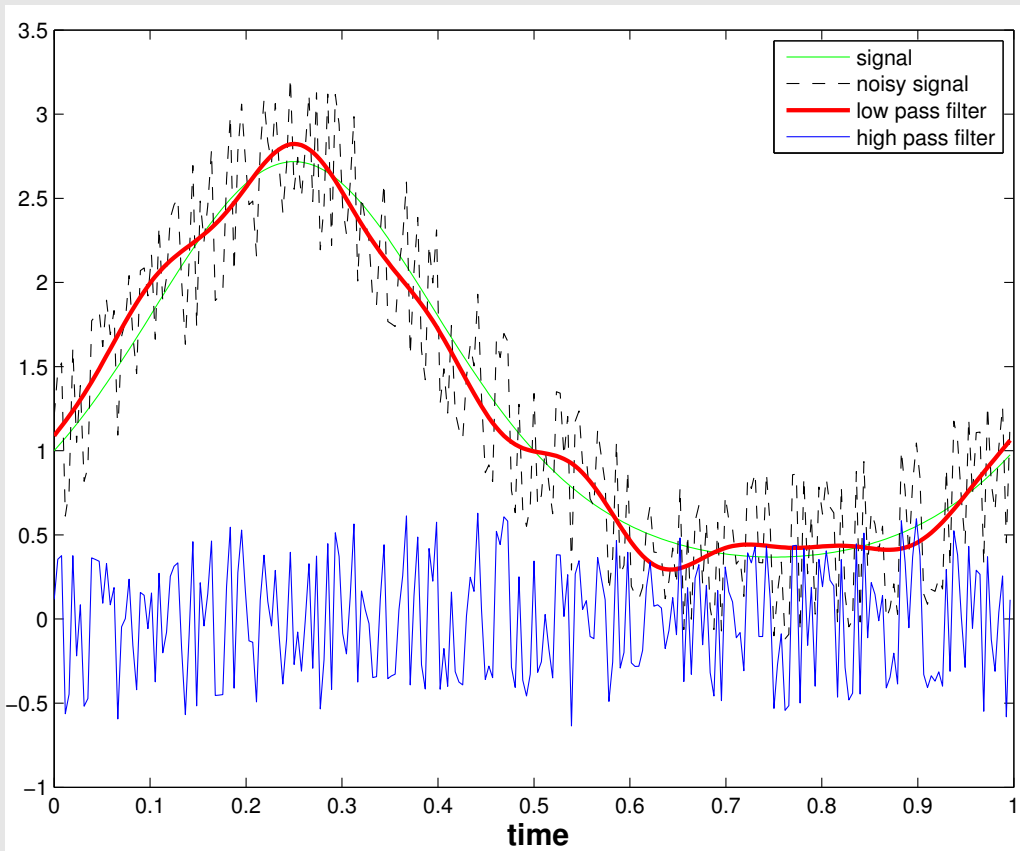
Map  $y \mapsto \text{high}$  (in Code 8.2.25)  $\hat{=}$  **high pass filter** (*ger.*: Hochpass).

Example 8.2.27 (Frequency filtering by DFT).

Noisy signal:

```
n = 256; y = exp(sin(2*pi*((0:n-1)')/n)) + 0.5*sin(exp(1:n)');
```

Frequency filtering by Code 8.2.25 with  $k = 120$ .





### Example 8.2.28 (Sound filtering by DFT).

#### Code 8.2.29: DFT based sound compression

```
1 % Low pass sound filtering by DFT
2
3 % Read sound data
4 [y,Fs,nbits] = wavread('hello.wav');
5 sound(y,Fs);
6
7 n = length(y);
8 fprintf('Read wav File: %d samples, rate = %d/s, nbits = %d\n',
    n,Fs,nbits);
9 k = 1; s{k} = y; leg{k} = 'Sampled signal';
0
1 c = fft(y);
2
3 figure('name','sound signal');
4 plot((22000:44000)/Fs,s{1}(22000:44000),'r-');
```

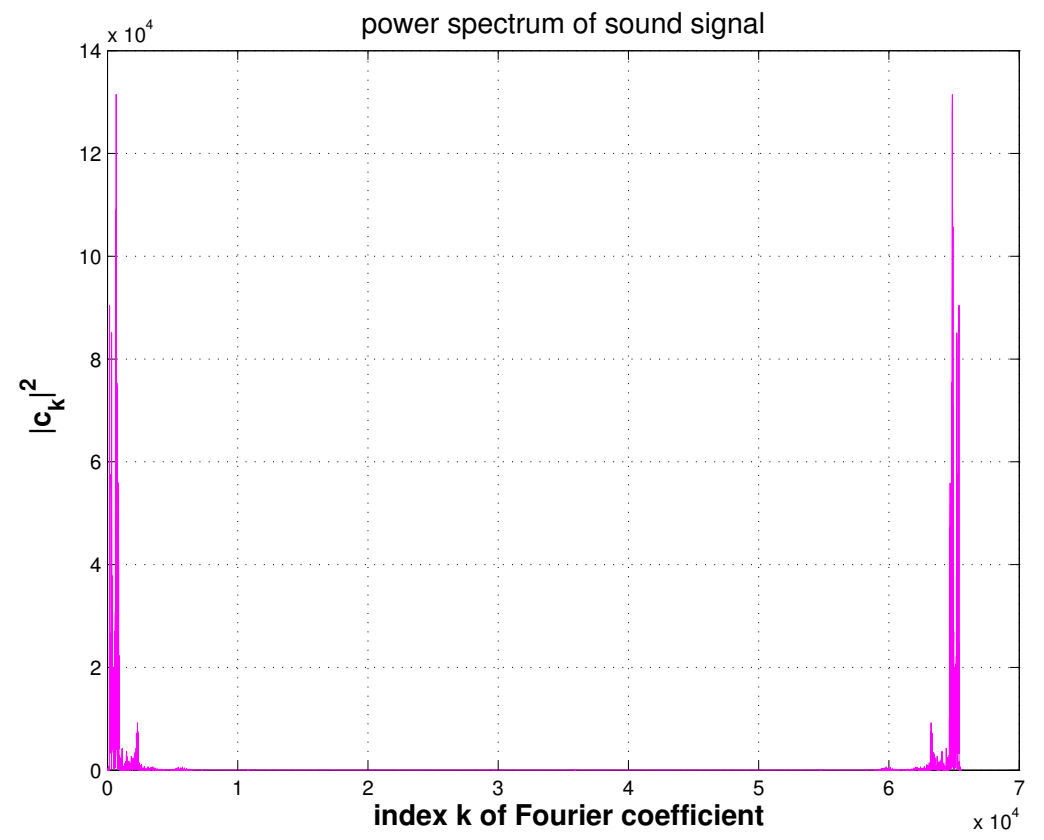
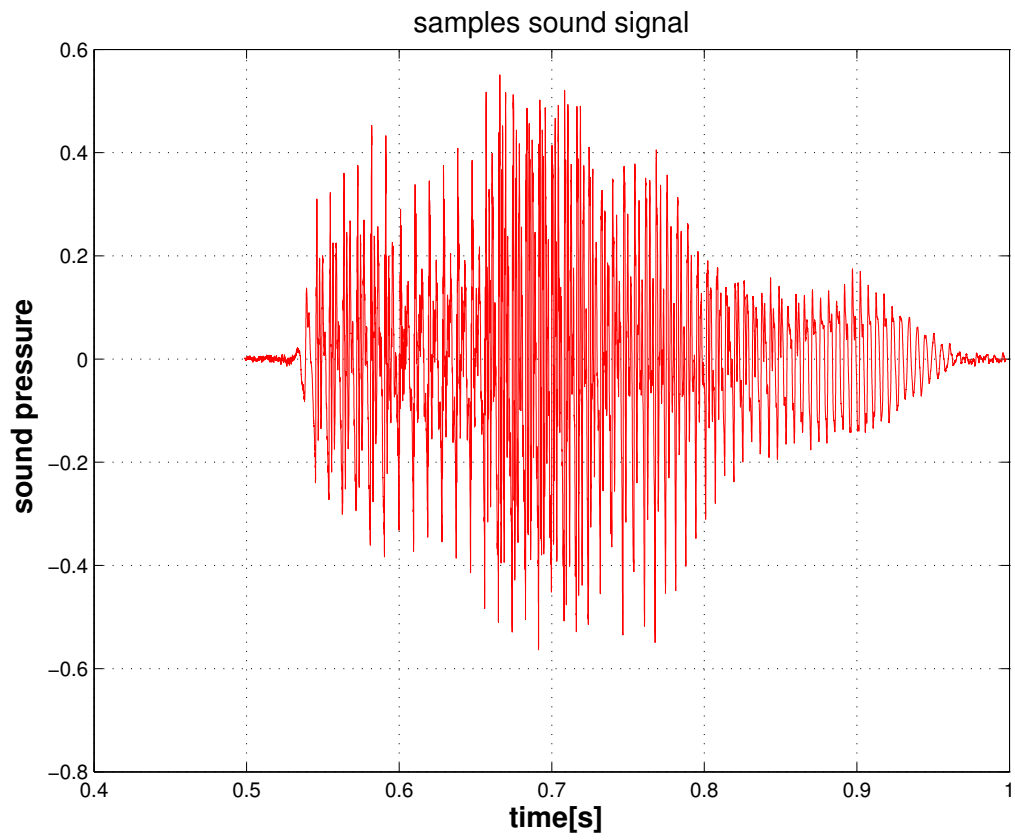
```
5 title ('samples sound signal','fontsize',14);
6 xlabel ('\bf time[s]','fontsize',14);
7 ylabel ('\bf sound pressure','fontsize',14);
8 grid on;
9
10 % print -depsc2 '../PICTURES/soundsignal.eps';
11
12 figure ('name','sound frequencies');
13 plot (1:n,abs(c).^2,'m-');
14 title ('power spectrum of sound signal','fontsize',14);
15 xlabel ('\bf index k of Fourier coefficient','fontsize',14);
16 ylabel ('\bf |c_k|^2','fontsize',14);
17 grid on;
18
19 % print -depsc2 '../PICTURES/soundpower.eps';
20
21 figure ('name','sound frequencies');
22 plot (1:3000,abs(c(1:3000)).^2,'b-');
23 title ('low frequency power spectrum','fontsize',14);
24 xlabel ('\bf index k of Fourier coefficient','fontsize',14);
25 ylabel ('\bf |c_k|^2','fontsize',14);
26 grid on;
27
```

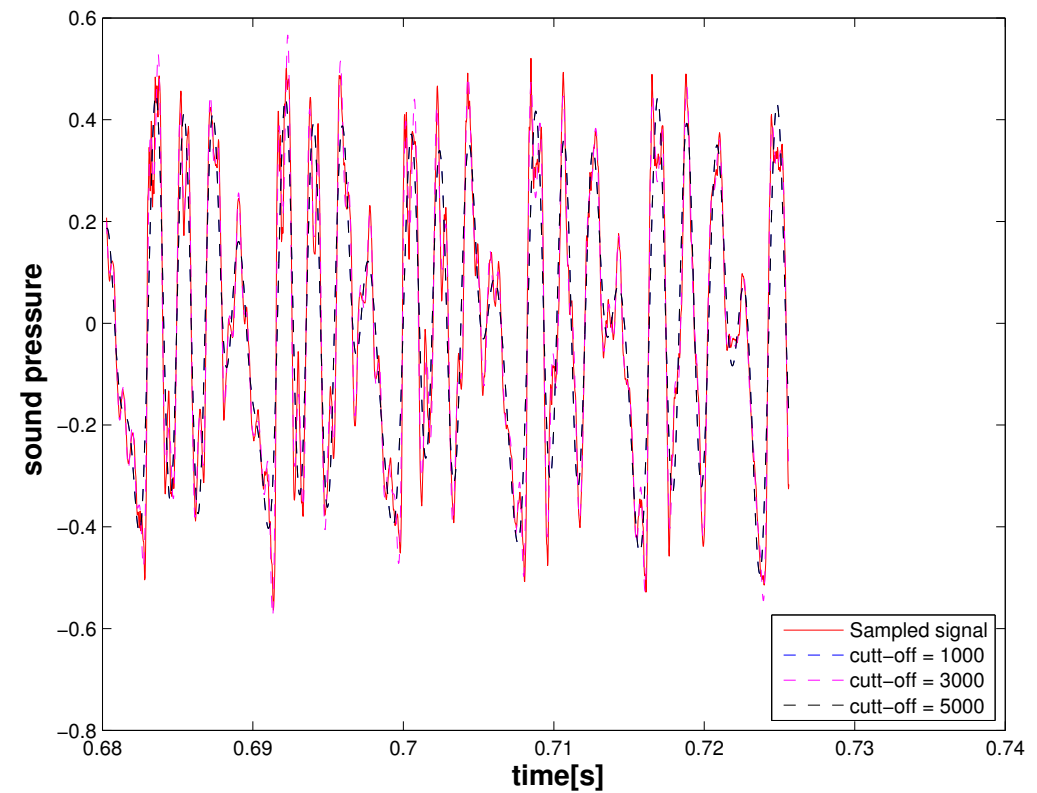
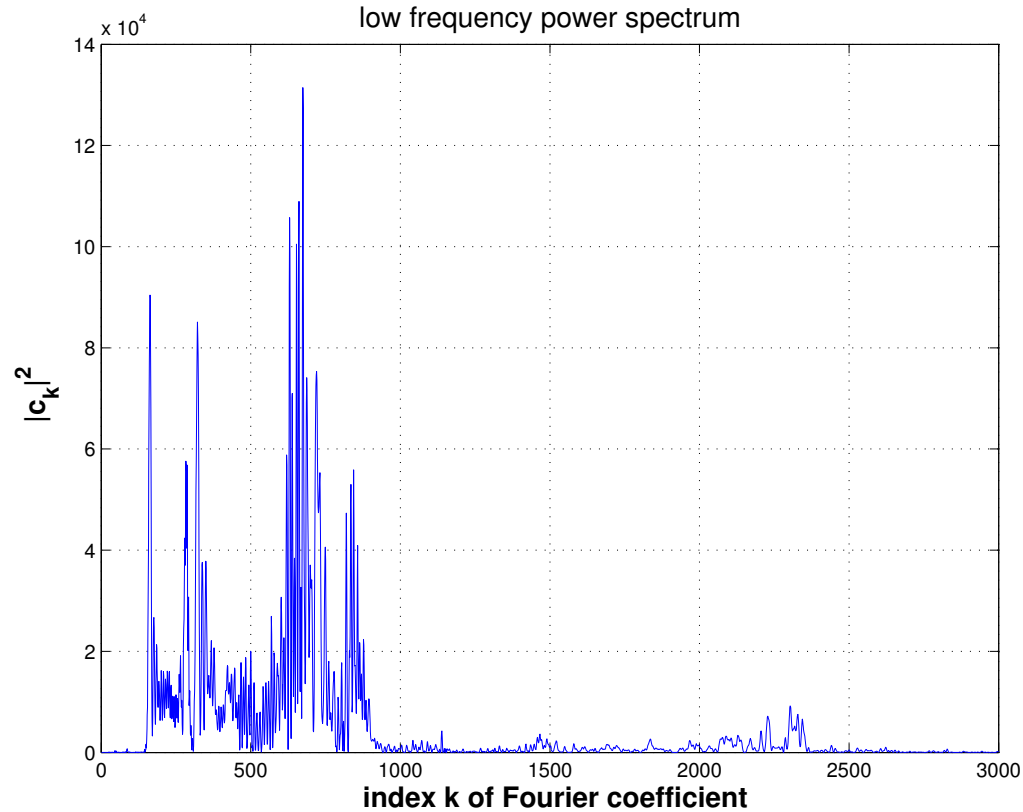
```
8 % print -depsc2 '../PICTURES/soundlowpower.eps';
9
0 for m=[1000,3000,5000]
1
2     % Low pass filtering
3     cf = zeros(1,n);
4     cf(1:m) = c(1:m); cf(n-m+1:end) = c(n-m+1:end);
5
6     % Reconstruct filtered signal
7     yf = ifft(cf);
8     % No idea why this is necessary
9     wavwrite(yf,Fs,nbits,sprintf('hellof%d.wav',m));
0     cy = wavread(sprintf('hellof%d.wav',m));
1     sound(cy,Fs,nbits);
2
3     k = k+1;
4     s{k} = real(yf);
5     leg{k} = sprintf('cutt-off = %d',m');
6 end
7
8 % Plot original signal and filtered signals
9 figure('name','sound filtering');
0 plot((30000:32000)/Fs,s{1}(30000:32000),'r-',...
```

```
1 (30000:32000)/Fs, s{2} (30000:32000), 'b--', ...
2 (30000:32000)/Fs, s{3} (30000:32000), 'm--', ...
3 (30000:32000)/Fs, s{2} (30000:32000), 'k--' );
4 xlabel ('{\bf time[s]}', 'fontsize', 14);
5 ylabel ('{\bf sound pressure}', 'fontsize', 14);
6 legend (leg, 'location', 'southeast');
7
8 % print -depsc2 '../PICTURES/soundfiltered.eps';
```

———— DFT based low pass frequency filtering of sound ————

```
[y,sf,nb] = wavread('hello.wav');
c = fft(y); c(m+1:end-m) = 0;
wavwrite(ifft(c),sf,nb,'filtered.wav');
```





The **power spectrum** of a signal  $\mathbf{y} \in \mathbb{C}^n$  is the vector  $\left(|c_j|^2\right)_{j=0}^{n-1}$ , where  $\mathbf{c} = \mathbf{F}_n \mathbf{y}$  is the discrete Fourier transform of  $\mathbf{y}$ .





Signal obtained from sampling a time-dependent voltage: a **real** vector.

Aim: efficient DFT (Def. 8.2.14)  $(c_0, \dots, c_{n-1})$  for *real* coefficients  $(y_0, \dots, y_{n-1})^\top \in \mathbb{R}^n$ ,  $n = 2m$ ,  $m \in \mathbb{N}$ .

If  $y_j \in \mathbb{R}$  in DFT formula (8.2.15), we obtain redundant output

$$\omega_n^{(n-k)j} = \overline{\omega_n^{kj}}, \quad k = 0, \dots, n-1,$$

$$\Rightarrow \bar{c}_k = \sum_{j=0}^{n-1} y_j \overline{\omega_n^{kj}} = \sum_{j=0}^{n-1} y_j \omega_n^{(n-k)j} = c_{n-k}, \quad k = 1, \dots, n-1.$$

**Idea:** map  $\mathbf{y} \in \mathbb{R}^n$ , to  $\mathbb{C}^m$  and use DFT of length  $m$ .

$$h_k = \sum_{j=0}^{m-1} (y_{2j} + iy_{2j+1}) \omega_m^{jk} = \boxed{\sum_{j=0}^{m-1} y_{2j} \omega_m^{jk}} + i \cdot \boxed{\sum_{j=0}^{m-1} y_{2j+1} \omega_m^{jk}}, \quad (8.2.30)$$

$$\bar{h}_{m-k} = \sum_{j=0}^{m-1} \frac{1}{y_{2j} + iy_{2j+1} \omega_m^{j(m-k)}} \omega_m^{j(m-k)} = \boxed{\sum_{j=0}^{m-1} y_{2j} \omega_m^{jk}} - i \cdot \boxed{\sum_{j=0}^{m-1} y_{2j+1} \omega_m^{jk}}. \quad (8.2.31)$$

$$\Rightarrow \boxed{\sum_{j=0}^{m-1} y_{2j} \omega_m^{jk}} = \frac{1}{2}(h_k + \bar{h}_{m-k}), \quad \boxed{\sum_{j=0}^{m-1} y_{2j+1} \omega_m^{jk}} = -\frac{1}{2}i(h_k - \bar{h}_{m-k}).$$

Use simple identities for roots of unity:

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{jk} = \boxed{\sum_{j=0}^{m-1} y_{2j} \omega_m^{jk}} + \omega_n^k \cdot \boxed{\sum_{j=0}^{m-1} y_{2j+1} \omega_m^{jk}}. \quad (8.2.32)$$

$$\Rightarrow \begin{cases} c_k = \frac{1}{2}(h_k + \bar{h}_{m-k}) - \frac{1}{2}i\omega_n^k(h_k - \bar{h}_{m-k}), & k = 0, \dots, m-1, \\ c_m = \operatorname{Re}\{h_0\} - \operatorname{Im}\{h_0\}, \\ c_k = \bar{c}_{n-k}, & k = m+1, \dots, n-1. \end{cases} \quad (8.2.33)$$

MATLAB-Implementation  
(by a DFT of length  $n/2$ ):

```

1 function c = fftreal(y)
2 n = length(y); m = n/2;
3 if (mod(n,2) ~= 0), error('n must be even');
   end
4 y = y(1:2:n)+i*y(2:2:n); h = fft(y); h =
   [h;h(1)];
5 c = 0.5*(h+conj(h(m+1:-1:1))) - ...
6     (0.5*i*exp(-2*pi*i/n).^((0:m)')) .* ...
7     (h-conj(h(m+1:-1:1)));
8 c = [c;conj(c(m:-1:2))];

```

(Note: not really optimal  
MATLAB implementation)

## 8.2.4 Two-dimensional DFT

A natural analogy:

one-dimensional data (“signal”)  $\longleftrightarrow$  vector  $\mathbf{y} \in \mathbb{C}^n$ ,

two-dimensional data (“image”)  $\longleftrightarrow$  matrix.  $\mathbf{Y} \in \mathbb{C}^{m,n}$

Two-dimensional trigonometric basis of  $\mathbb{C}^{m,n}$ :

tensor product matrices  $\left\{ (\mathbf{F}_m)_{:,j} (\mathbf{F}_n)_{:,l}^\top, 1 \leq j \leq m, 1 \leq l \leq n \right\}$  . (8.2.35)

Basis transform: for  $y_{j_1, j_2} \in \mathbb{C}, 0 \leq j_1 < m, 0 \leq j_2 < n$  compute (nested DFTs !)

$$c_{k_1, k_2} = \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} y_{j_1, j_2} \omega_m^{j_1 k_1} \omega_n^{j_2 k_2}, \quad 0 \leq k_1 < m, 0 \leq k_2 < n .$$

MATLAB function: `fft2(Y)` .

Two-dimensional DFT by *nested one-dimensional DFTs* (8.2.15):

$$\text{fft2}(Y) = \text{fft}(\text{fft}(Y) \cdot') \cdot'$$

Here: `.'` simply transposes the matrix (no complex conjugation)

Example 8.2.36 (Deblurring by DFT).

Gray-scale pixel image  $\mathbf{P} \in \mathbb{R}^{m,n}$ , actually  $\mathbf{P} \in \{0, \dots, 255\}^{m,n}$ , see Ex. 6.3.22.

$(p_{l,k})_{l,k \in \mathbb{Z}} \hat{=}$  periodically extended image:

$$p_{l,j} = (\mathbf{P})_{l+1,j+1} \quad \text{for } 1 \leq l \leq m, 1 \leq j \leq n, \quad p_{l,j} = p_{l+m,j+n} \quad \forall l, k \in \mathbb{Z}.$$

**Blurring** = pixel values get replaced by weighted averages of near-by pixel values  
(effect of distortion in optical transmission systems)

$$c_{l,j} = \sum_{k=-L}^L \sum_{q=-L}^L s_{k,q} p_{l+k,j+q}, \quad \begin{array}{l} 0 \leq l < m, \\ 0 \leq j < n, \end{array} \quad L \in \{1, \dots, \min\{m, n\}\}. \quad (8.2.37)$$

↓ blurred image      ↑ point spread function

Does this ring a bell? Hidden in (8.2.37) are two (periodic) discrete convolutions, see Def. 8.1.13.

Hardly surprising that DFT comes handy for reversing the effect of the blurring!

Usually:  $L$  small,  $s_{k,m} \geq 0$ ,  $\sum_{k=-L}^L \sum_{q=-L}^L s_{k,q} = 1$  (an averaging)

Used in test calculations:  $L = 5$

$$s_{k,q} = \frac{1}{1 + k^2 + q^2}.$$

Code 8.2.38: point spread function

```

1 function S = psf(L)
2 [X,Y] = meshgrid (-L:1:L, -L:1:L) ;
3 S = 1 ./ (1+X.*X+Y.*Y) ;
4 S = S/sum(sum(S)) ;

```

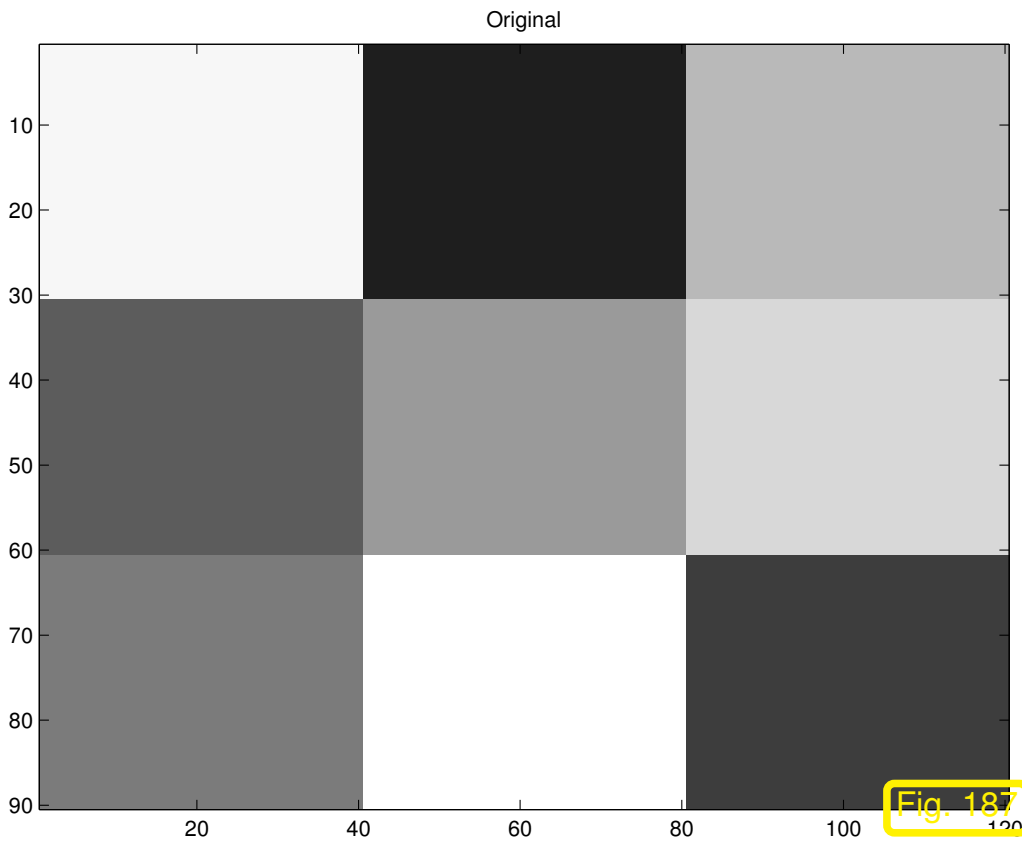


Fig. 187

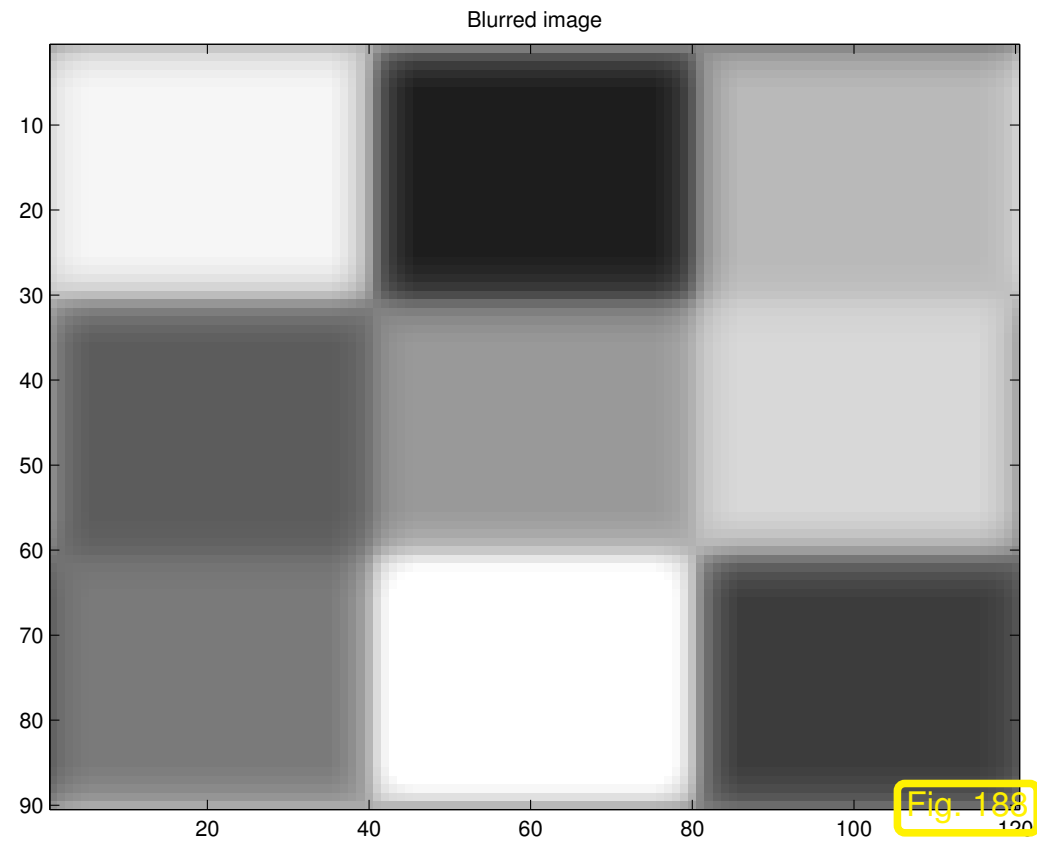


Fig. 188

## Code 8.2.39: MATLAB deblurring experiment

```
1 % Generate artificial "image"
2 P = kron(magic(3), ones(30, 40)) * 31;
3 col = [0:1/254:1]' * [1, 1, 1];
4 figure; image(P); colormap(col); title('Original');
5 print -depsc2 '../PICTURES/dborigimage.eps';
6 % Generate point spread function
7 L = 5; S = psf(L);
8 % Blur image
9 C = blur(P, S);
0 figure; image(floor(C)); colormap(col); title('Blurred image');
1 print -depsc2 '../PICTURES/dbblurredimage.eps';
2 % Deblur image
3 D = deblur(C, S);
4 figure; image(floor(real(D))); colormap(col);
5 fprintf('Difference of images (Frobenius norm): %f\n', norm(P-D));
```

## Code 8.2.40: blurring operator

```

1 function C = blur(P,S)
2 [m,n] = size (P); [M,N] = size (S);
3 if (M ~= N), error ('S not quadratic'); end
4 L = (M-1)/2; C = zeros (m,n);
5 for l=1:m, for j=1:n
6     s = 0;
7     for k=1:(2*L+1), for q=1:(2*L+1)
8         kl = l+k-L-1;
9         if (kl < 1), kl = kl + m; end
10        if (kl > m), kl = kl - m; end
11        jm = j+q-L-1;
12        if (jm < 1), jm = jm + n; end
13        if (jm > n), jm = jm - n; end
14        s = s+P(kl,jm)*S(k,q);
15        end, end
16    C(l,j) = s;
17 end, end

```

Note:

(8.2.37) defines a linear  
operator

$$\mathcal{B} : \mathbb{R}^{m,n} \mapsto \mathbb{R}^{m,n}$$

(“blurring operator”)

Note: more efficient im-  
plementation via MAT-  
LAB function `conv2(P,S)`



Recall: derivation of (8.2.7) and Lemma 8.2.12. Try this in 2D!

$$\left( \mathcal{B} \left( \left( \omega_m^{\nu k} \omega_n^{\mu q} \right)_{k,q \in \mathbb{Z}} \right) \right)_{l,j} = \sum_{k=-L}^L \sum_{q=-L}^L s_{k,q} \omega_m^{\nu(l+k)} \omega_n^{\mu(j+q)} = \omega_m^{\nu l} \omega_n^{\mu j} \sum_{k=-L}^L \sum_{q=-L}^L s_{k,q} \omega_m^{\nu k} \omega_n^{\mu q} .$$

►  $\mathbf{V}_{\nu,\mu} := \left( \omega_m^{\nu k} \omega_n^{\mu q} \right)_{k,q \in \mathbb{Z}}$ ,  $0 \leq \mu < m$ ,  $0 \leq \nu < n$  are the eigenvectors of  $\mathcal{B}$ :

$$\mathcal{B} \mathbf{V}_{\nu,\mu} = \lambda_{\nu,\mu} \mathbf{V}_{\nu,\mu} \quad , \quad \text{eigenvalue} \quad \lambda_{\nu,\mu} = \underbrace{\sum_{k=-L}^L \sum_{q=-L}^L s_{k,q} \omega_m^{\nu k} \omega_n^{\mu q}}_{\text{2-dimensional DFT of point spread function !}} \quad (8.2.41)$$

2-dimensional DFT of point spread function !

## Code 8.2.42: DFT based deblurring

```

1 function D = deblur(C,S,tol)
2 [m,n] = size (C); [M,N] = size (S);
3 if (M ~= N), error ('S not quadratic'); end
4 L = (M-1)/2; Spad = zeros (m,n);
5 % Zero padding
6 Spad(1:L+1,1:L+1) = S(L+1:end,L+1:end);
7 Spad(m-L+1:m,n-L+1:n) = S(1:L,1:L);
8 Spad(1:L+1,n-L+1:n) = S(L+1:end,1:L);
9 Spad(m-L+1:m,1:L+1) = S(1:L,L+1:end);
10 % Inverse of blurring operator
11 SF = fft2 (Spad);
12 % Test for invertibility
13 if (nargin < 3), tol = 1E-3; end
14 if (min (min (abs (SF))) < tol*max (max (abs (SF))))
15     error ('Deblurring impossible');
16 end
17 % DFT based deblurring
18 D = fft2 (ifft2 (C) ./SF);

```

Inversion of blurring  
operator



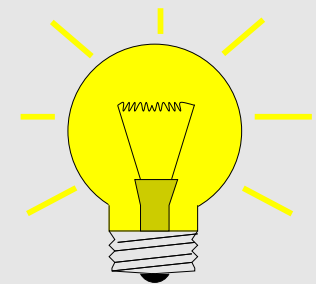
componentwise scaling  
in “Fourier domain”



## 8.2.5 Semi-discrete Fourier Transform [51, Sect. 10.11]

Starting from Ex. 8.1.10 we mainly looked at time-discrete  $n$ -periodic signals, which can be mapped to vectors  $\in \mathbb{R}^n$ . This led to discrete periodic convolution ( $\rightarrow$  Def. 8.1.13) and the discrete Fourier transform (DFT) ( $\rightarrow$  Def. 8.2.14) as (bi-)linear mappings in  $\mathbb{C}^n$ .

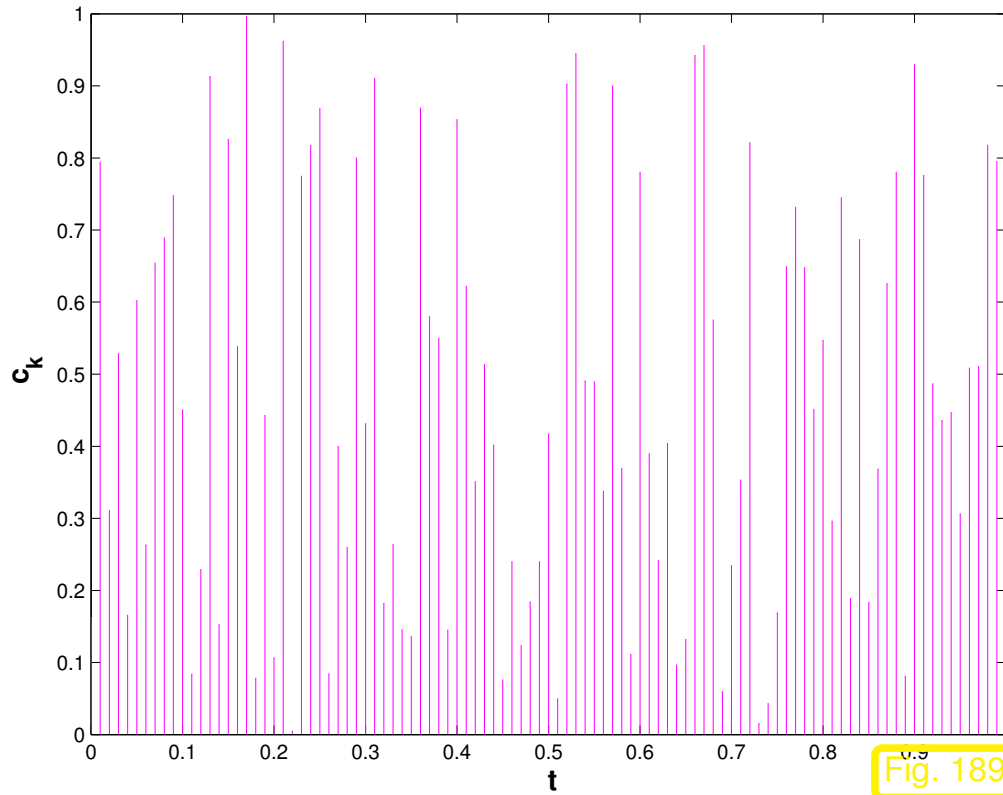
In this section we are concerned with **non-periodic** signals of **infinite duration**.



Idea: Study the limit  $n \rightarrow \infty$  for the  $n$ -periodic setting and DFT.

$(y_k)_{k \in \mathbb{Z}}$ :  $n$ -periodic sequence (signal),  $n = 2m + 1$ ,  $m \in \mathbb{N}$ :

$$\text{DFT: } c_k = \sum_{j=-m}^m y_j \exp(-2\pi i \frac{kj}{n}), \quad k = 0, \dots, n-1. \quad (8.2.43)$$



◁ “Squeezing” a vector  $\in \mathbb{R}^n$  into  $[0, 1[$ .

$$c_k \leftrightarrow c(t_k);,$$

$$t_k = \frac{k}{n}, \quad k = 0, \dots, n - 1.$$

Now we associate a point  $t_k \in [0, 1[$  with each index  $k$

$$k \in \{0, \dots, n - 1\} \longleftrightarrow t_k := \frac{k}{n}. \quad (8.2.44)$$

Thus, formally, we can rewrite (8.2.43) as

$$\text{DFT:} \quad c(t_k) := c_k = \sum_{j=-m}^m y_j \exp(-2\pi i j t_k), \quad k = 0, \dots, n - 1. \quad (8.2.45)$$

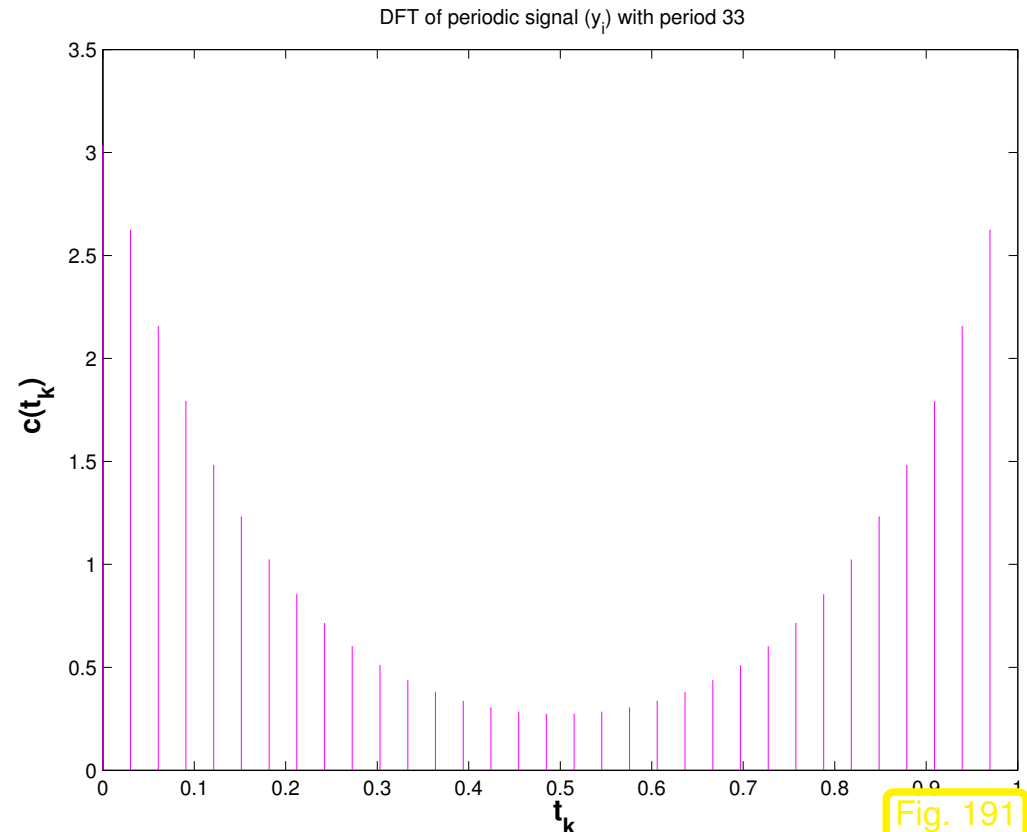
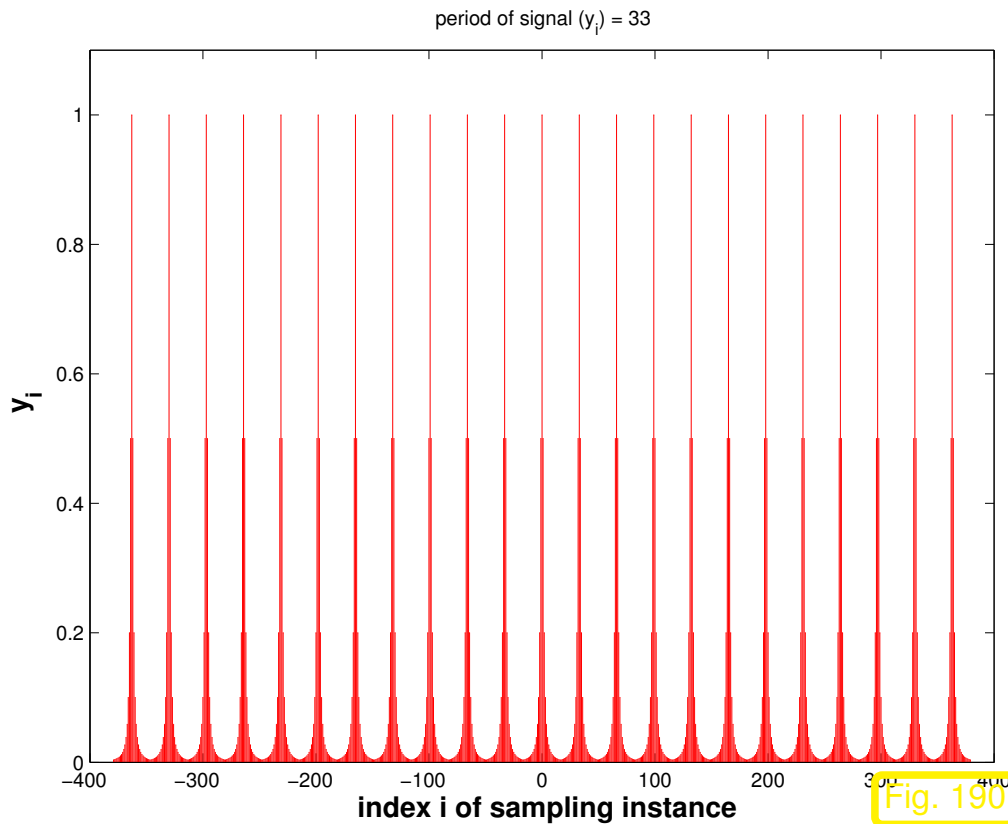
The notation indicates that we read  $c_k$  as the value of a *function*  $c : [0, 1[ \mapsto \mathbb{C}$  for argument  $t_k$ .

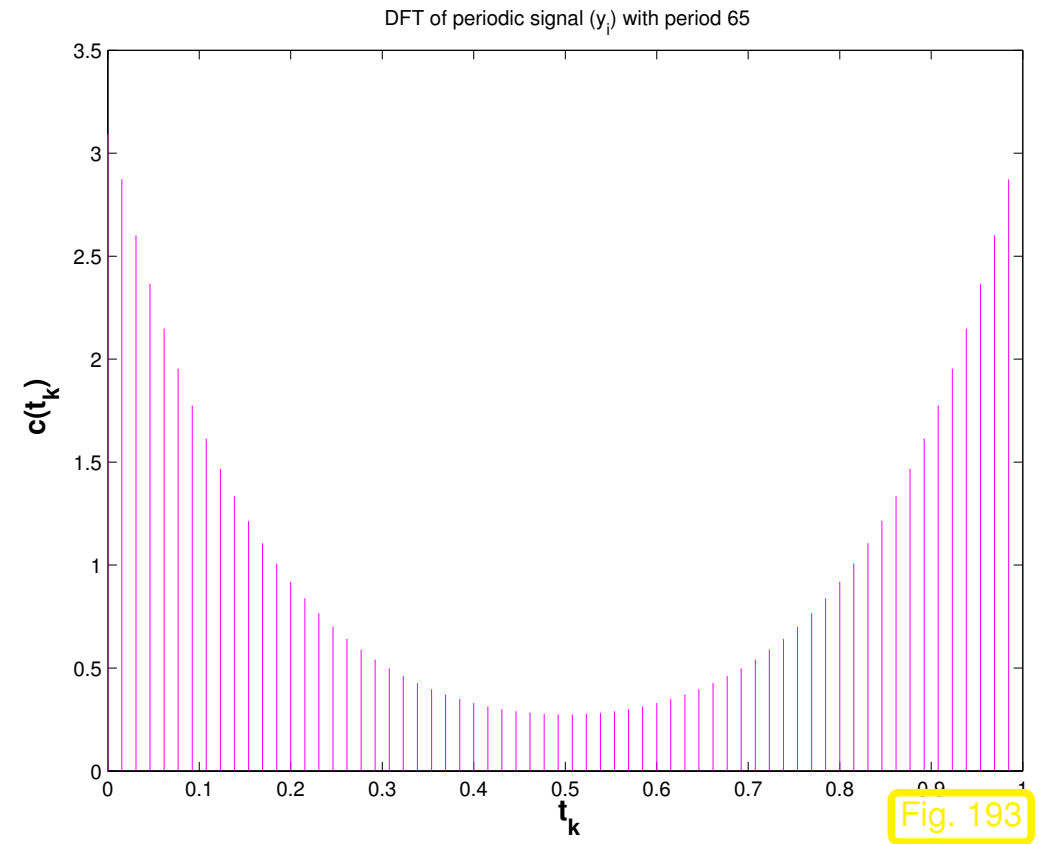
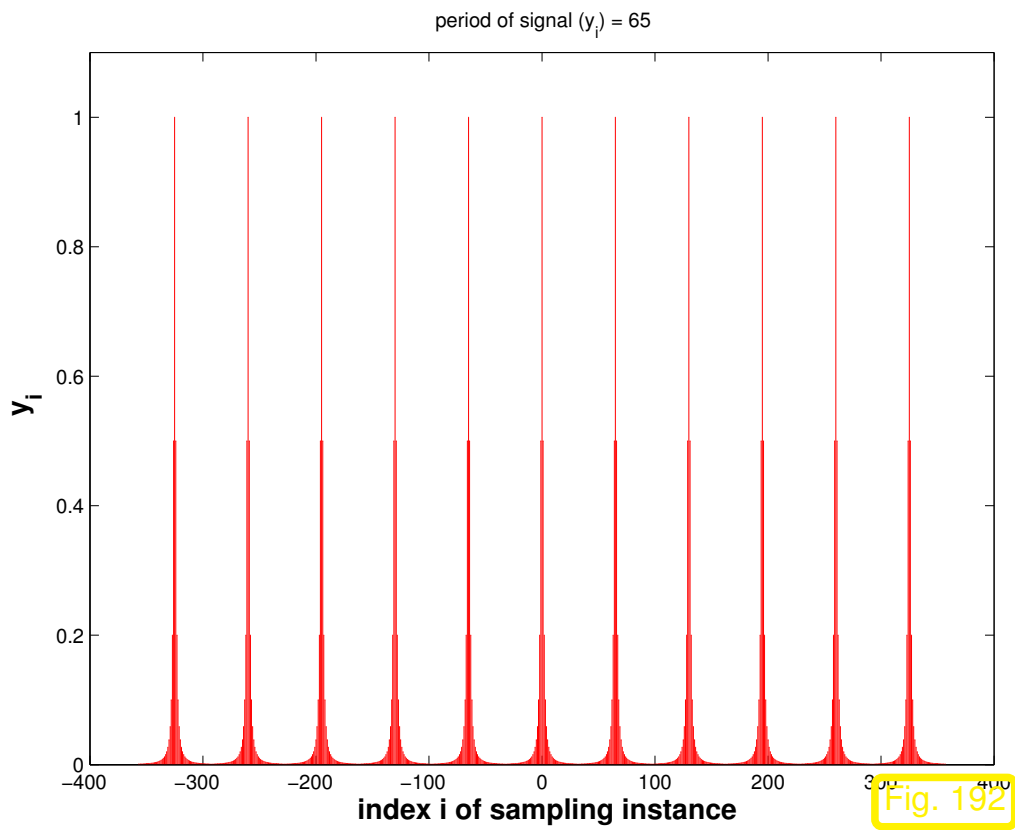
*Example 8.2.46* (“Squeezed” DFT of a periodically truncated signal).

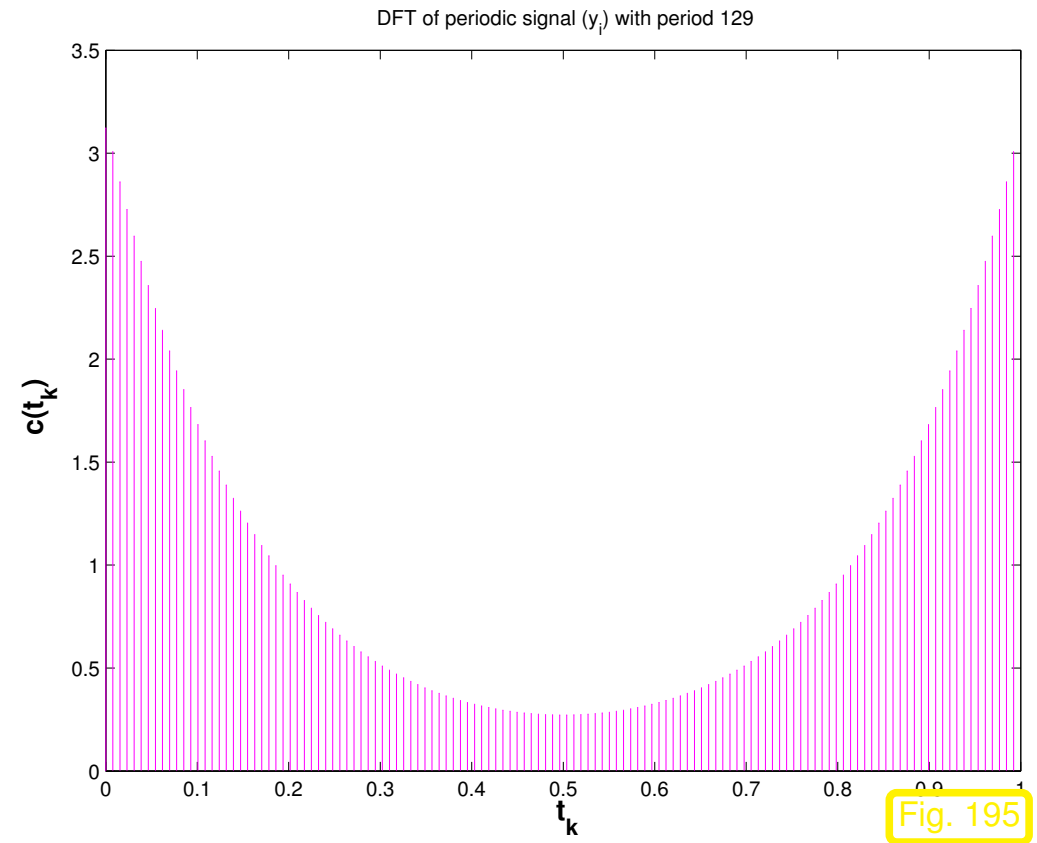
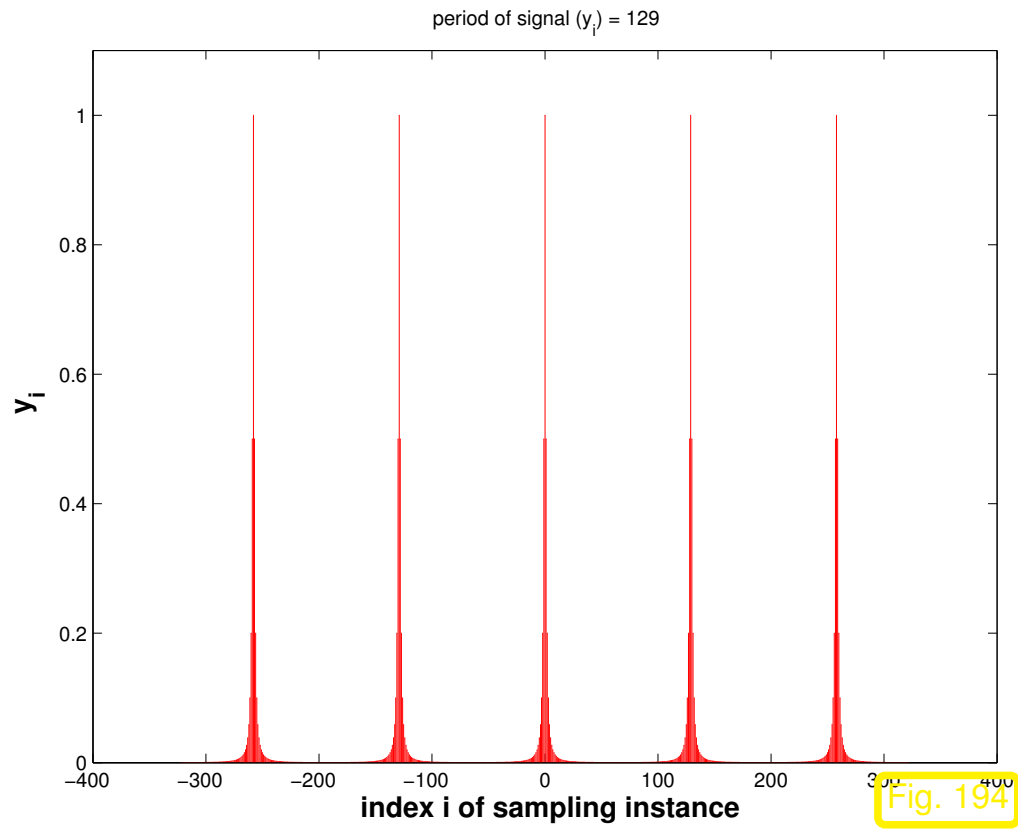
Bi-infinite discrete signal, “concentrated around 0”:

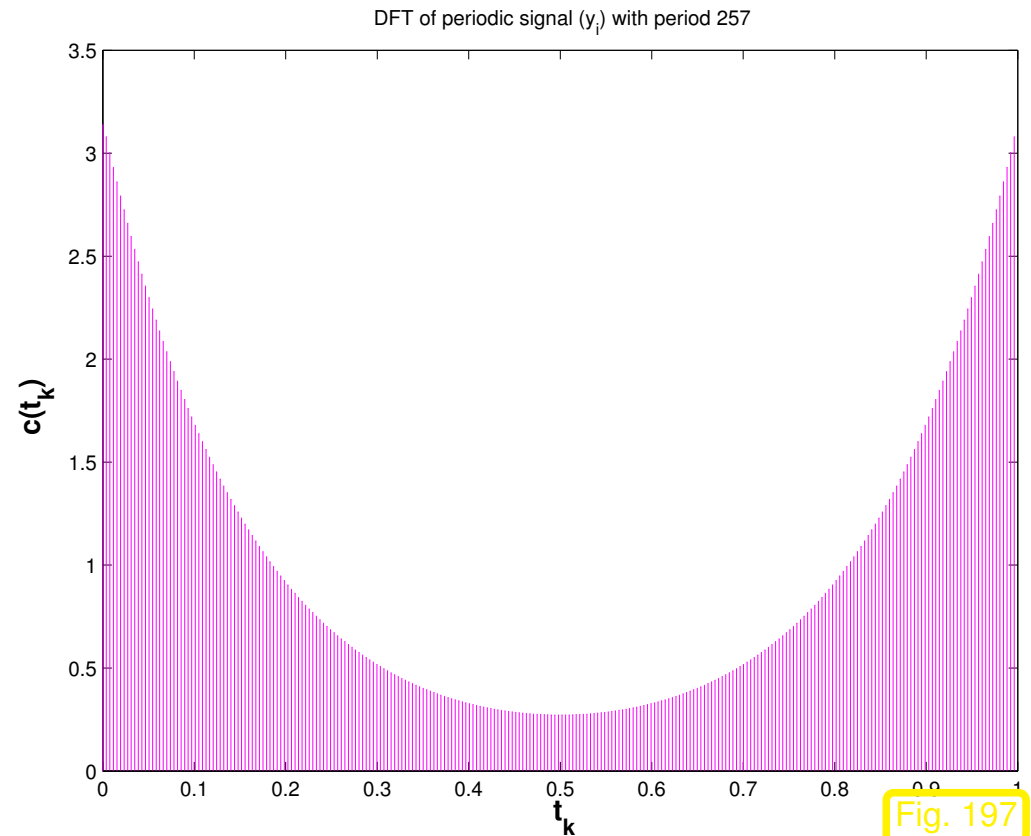
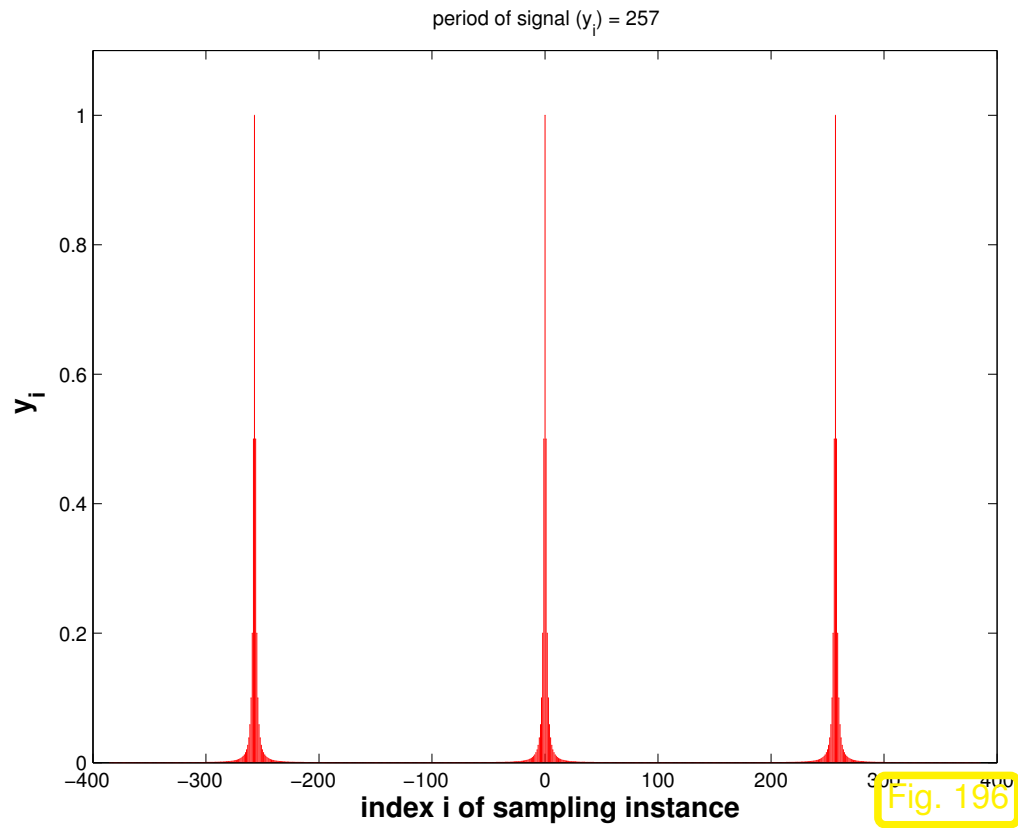
$$y_j = \frac{1}{1 + j^2}, \quad j \in \mathbb{Z}.$$

We examine the DFT of the  $2m + 1$ -periodic signal obtained by periodic extension of  $(y_k)_{k=-m}^m$ .









Observation:  $c(t_k)$  “converge” to a function  $c : [0, 1[ \mapsto \mathbb{R}$





## Code 8.2.47: Plotting a periodically truncated signal and its DFT

```

1 % Visualize limit  $m \rightarrow \infty$  for a  $2m+1$ -periodic signal and
2 % its discrete Fourier transform "squeezed" into  $[0,1]$ .
3
4 % range of plot for visualization of discrete signal
5 Npow = 8; N = 3*(2^Npow+1);
6 % function defining discrete signal
7 yfn = @(k) 1./(1+k.*k);
8 % loop over different periods  $2^l+1$ 
9 for mpow = [4 5 6 7]
10     m = 2^mpow;
11     ybas = yfn([-m:-1], 0, (1:m));
12     Ncp = floor(N/(2*m+1));
13     y = repmat(ybas, 1, Ncp);
14
15     figure; hy = stem((1:length(y)) - (length(y)+1)/2, y, 'r');
16     ax = axis; axis([ax(1) ax(2) 0 1.1]); hold on;
17     set(hy, 'markersize', 0);
18     xlabel('{\bf index i of sampling instance}', 'fontsize', 14);
19     ylabel('{\bf y_{i}}', 'fontsize', 14);
20     title(sprintf('period of signal (y_{i}) = %d', 2*m+1));
21     print('-depsc2', sprintf(' ../PICTURES/persig%d.eps', mpow));
22

```

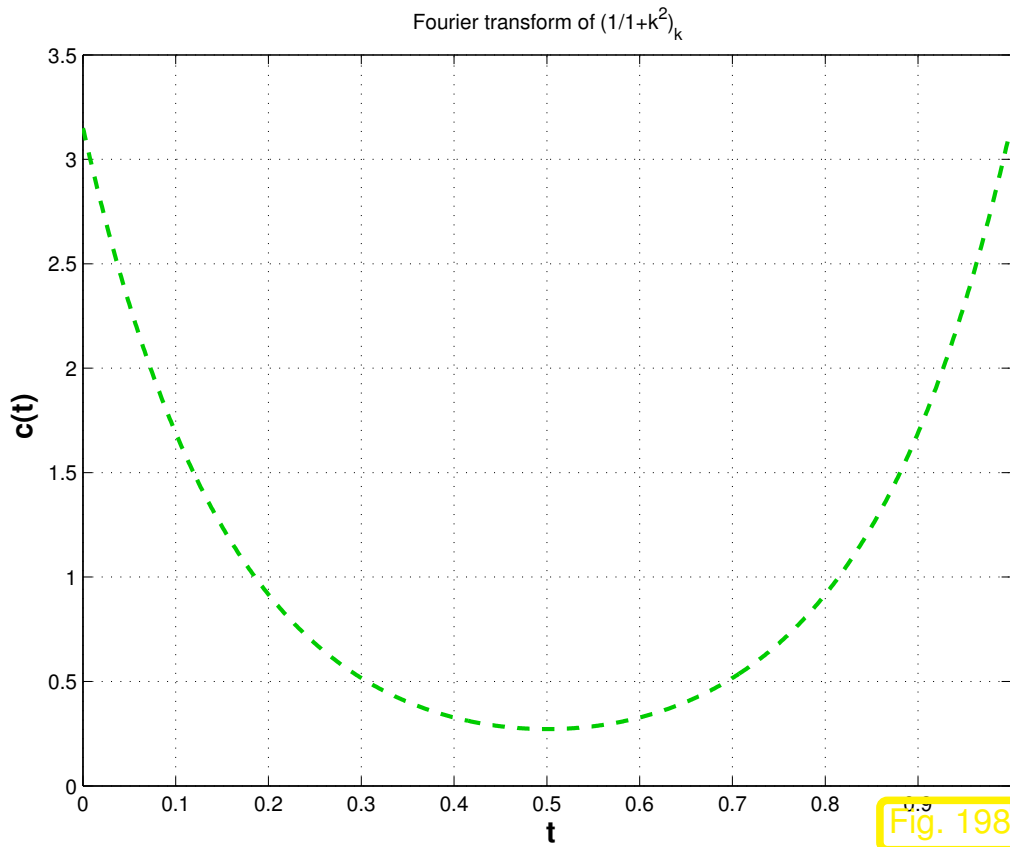
```
3 % DFT of wrapped signal (one period)
4 c = fft([ybas(m+1:end), ybas(1:m)]);
5 hc = 1/(2*m+1);
6 figure; hc = stem(0:hc:1-hc, c, 'm');
7 set(hc, 'markersize', 0);
8 xlabel('\bf t_{k}', 'fontsize', 14);
9 ylabel('\bf c(t_{k})', 'fontsize', 14);
0 title(sprintf('DFT of periodic signal (y_{i}) with period
   %d', 2*m+1));
1 print('-depsc2', sprintf('../PICTURES/persigdft%d.eps', mpow));
2 end
```

Now: pass to the limit  $m \rightarrow \infty$  (and keep the function perspective)

Note: passing to the limit amounts to dropping the assumption of periodicity!

$$c(t) = \sum_{k \in \mathbb{Z}} y_k \exp(-2\pi i k t) . \quad (8.2.48)$$

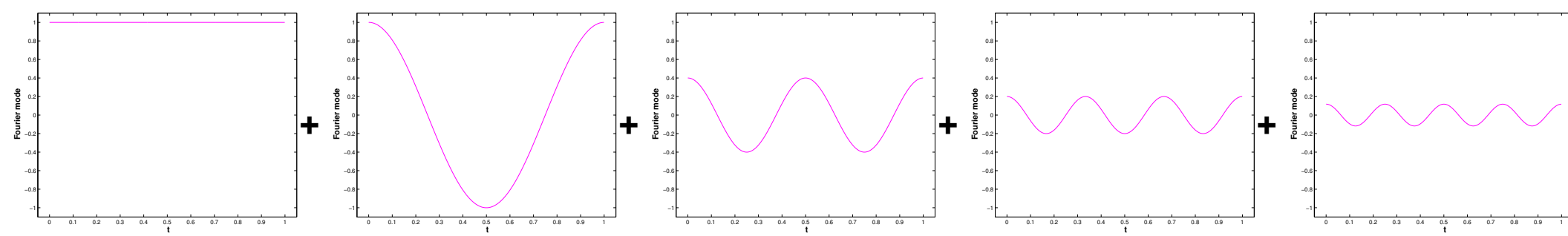
Terminology: the series (= infinite sum) on the right hand side of (8.2.48) is called a **Fourier series**  
 the function  $c : [0, 1[ \mapsto \mathbb{C}$  defined by (8.2.48) is called the **Fourier transform** of the sequence  $(y_k)_{k \in \mathbb{Z}}$  (, if the series converges).



◁ Fourier transform of  $y_k := \frac{1}{1+k^2}$

Fourier transform  
 =  
 sum of Fourier modes





*Remark 8.2.49* (Decay conditions for bi-infinite signals).

The considerations above were based on

- truncation of  $(y_k)_{k \in \mathbb{Z}}$  to  $(y_k)_{k=-m}^m$  and
- periodic continuation to an  $2m + 1$ -periodic signal.

Obviously, only if the signal is *concentrated around*  $k = 0$  this procedure will not lose essential information contained in the signal.

Minimal requirement: 
$$\lim_{k \rightarrow \infty} |y_k| = 0, \tag{8.2.50}$$

Stronger requirement: 
$$\sum_{k \in \mathbb{Z}} |y_k| < \infty. \tag{8.2.51}$$

- (8.2.51)  $\Rightarrow$  Fourier series (8.2.48) (link) converges **uniformly** [63, Def. 4.8.1]
- $\Rightarrow c : [0, 1[ \mapsto \mathbb{C}$  is continuous [63, Thm. 4.8.1].



*Remark 8.2.52* (Numerical summation of Fourier series).

Assuming sufficiently fast decay of the signal  $(y_k)_{k \in \mathbb{Z}}$  for  $k \rightarrow \infty$  ( $\rightarrow$  Rem. 8.2.49), we can *approximate* the Fourier series (8.2.48) by a **Fourier sum**

$$c(t) \approx c_M(t) := \sum_{k=-M}^M y_k \exp(-2\pi i k t), \quad M \gg 1. \quad (8.2.53)$$

Task: Evaluation of  $c(t)$  at  $N$  equidistant points  $t_j := \frac{j}{N}$ ,  $j = 0, \dots, N$  (e.g., for plotting it).

$$c(t_j) = \lim_{M \rightarrow \infty} \sum_{k=-M}^M y_k \exp(-2\pi i k t_j) \approx \sum_{k=-M}^M y_k \exp(-2\pi i \frac{kj}{N}), \quad (8.2.54)$$

for  $j = 0, \dots, N - 1$ .

Note: If  $N = M \gg 1$  (8.2.54) is a **discrete Fourier transform** ( $\rightarrow$  Def. 8.2.14).

Code 8.2.55: FFT-based evaluation of Fourier sum at equidistant points

```

1 function c = foursum(signal, M, N)
2 % Approximate evaluation of Fourier series, signal is a handle to a function of
3 % type @(k) providing the  $y_k$ , M specifies truncation

```

```

4 % of series according to (8.2.53), N is the number of equidistant
5 % evaluation points for c in [0,1].
6
7 y = signal(-M:M); % Sample signal from -M to M
8 m = 2*M+1; % Length of signal
9 % Ensure that there are more sampling points than terms in series
0 if (m > N), l = ceil(m/N); N = l*N; else l = 1; end
1
2 % Zero padding and wrapping of signal, see Code 8.2.25
3 y_ext = zeros(1,N); y_ext(1:M+1) = y(M+1:end); y_ext(N-M+1:N) =
  y(1:M);
4
5 % Perform DFT and decimate output vector
6 c = fft(y_ext); c = c(1:l:end);

```

### Code 8.2.56: Inefficient direct evaluation of Fourier sum at equidistant points

```

1 function c = foursumnaive(signal,M,N)
2 % Approximate direct evaluation of Fourier sum according to the defining
3 % formula (8.2.53), signal is a handle to a function of
4 % type @(k) providing the  $y_k$ , M specifies truncation
5 % of series according to (8.2.53), N is the number of equidistant
6 % evaluation points for c in [0,1].
7
8 t = 0:1/N:1-1/N; % Evaluation points for Fourier sum c

```

```
9 c = signal(0)*ones(1,N);
0 omega = exp(-2*pi*i*t);
1 omp = omega; omm = 1./omega;
2
3 % Inefficient direct summation of Fourier series
4 for k=1:M
5     c = c+signal(k)*omp;
6     c = c+signal(-k)*omm;
7     omp = omp.*omega;
8     omm = omm./omega;
9 end
```

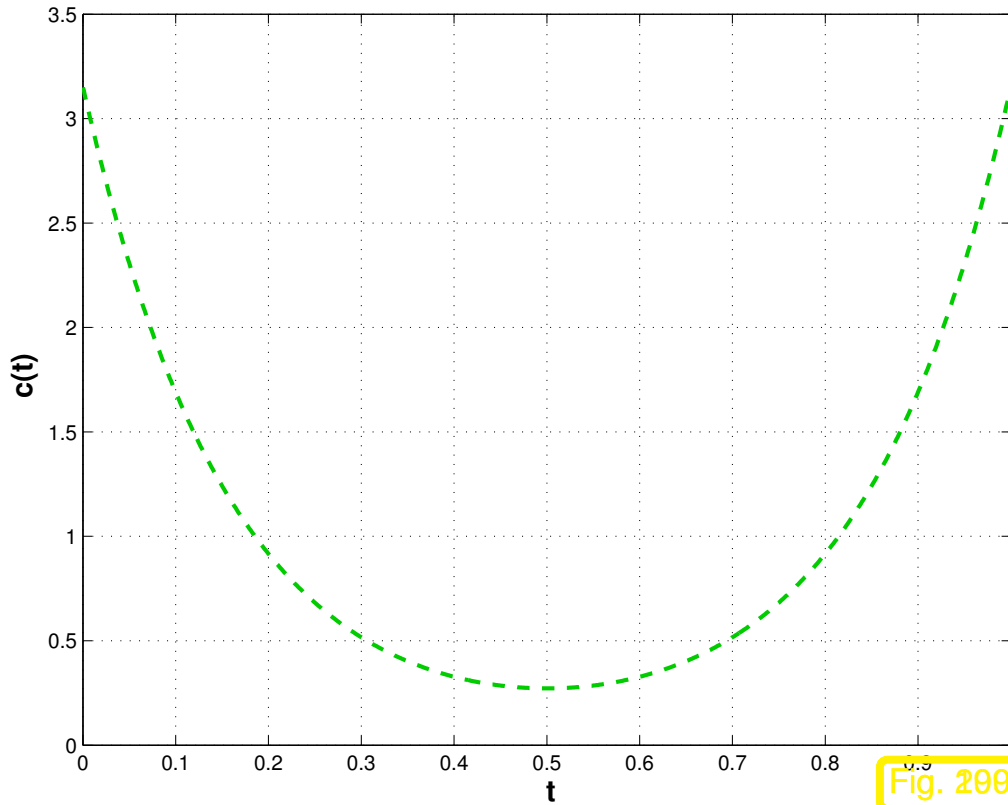


*Example 8.2.57* (Convergence of Fourier sums).

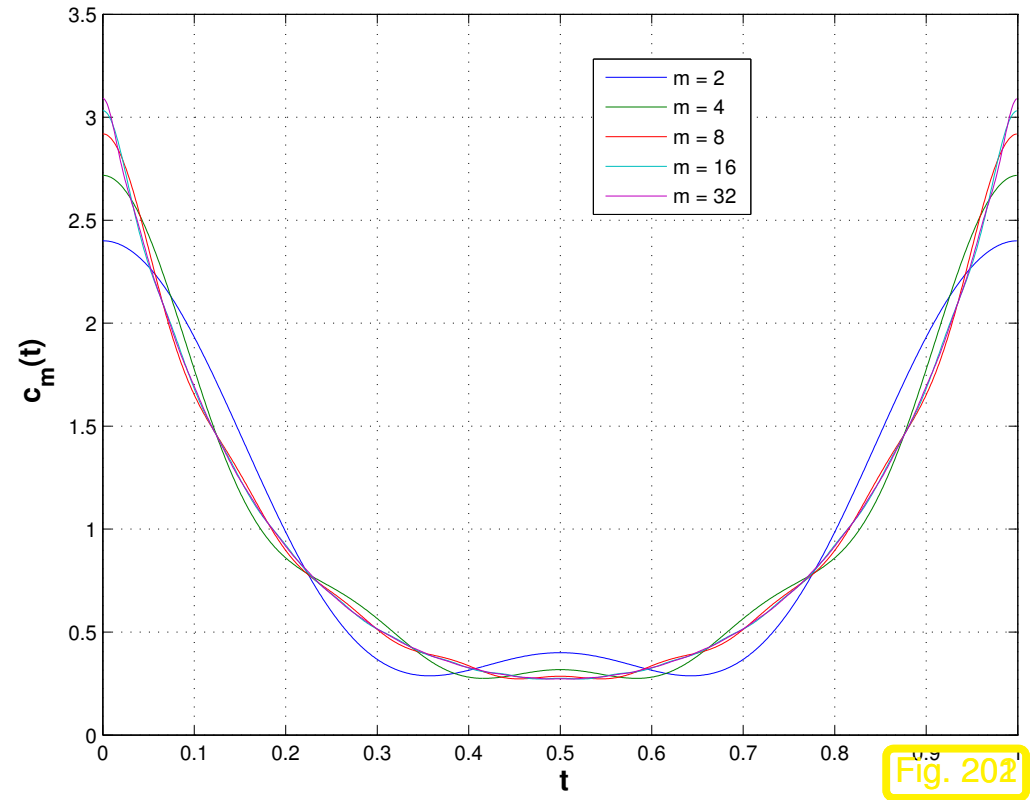
Infinite signal, satisfying decay condition (8.2.51):  $y_k = \frac{1}{1+k^2}$ , see Ex. 8.2.46.

Monitored: approximation of Fourier transform  $c(t)$  by Fourier sums  $c_m(t)$ , see (8.2.53).

Fourier transform of  $(1/(1+k^2))_k$



Fourier sum approximations with  $2m+1$  terms,  $y_k = 1/(1+k^2)$



Observation: Convergence of Fourier sums in “eyeball norm”; quantitative statements about convergence can be deduced from Thm. ??.





Same limit considerations as above for the inverse DFT (8.2.16)

$$y_j = \frac{1}{n} \sum_{k=0}^{n-1} c_k \exp(2\pi i \frac{jk}{n}), \quad j = -m, \dots, m. \quad (8.2.58)$$

Adopt function perspective as before:  $c_k \leftrightarrow c(t_k)$ , cf. (8.2.44).

$$y_j = \frac{1}{n} \sum_{k=0}^{n-1} c(t_k) \exp(2\pi i j t_k), \quad j = -m, \dots, m. \quad (8.2.59)$$

Now: pass to the limit  $m \rightarrow \infty$  in (8.2.59)

Idea: right hand side of (8.2.59) = **Riemann sum**, cf. [63, Sect. 6.2]



in the limit  $m \rightarrow \infty$  the sum becomes an **integral!**

$$y_j = \int_0^1 c(t) \exp(2\pi i j t) dt. \quad (8.2.60)$$

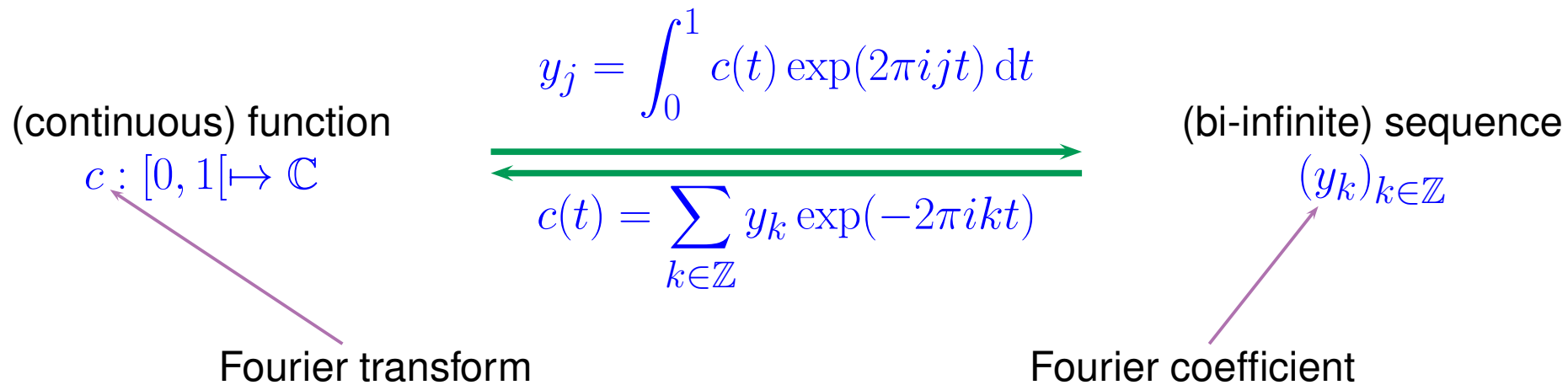
The formula (8.2.60) allows to recover the signal  $(y_k)_{k \in \mathbb{Z}}$  from its Fourier transform  $c(t)$ .

Terminology:  $y_j$  from (8.2.60) is called the  $j$ -th **Fourier coefficient** of the function  $c$ .

Note: Both, the mapping  $(y_k)_{k \in \mathbb{Z}} \mapsto c$  from (8.2.48), and the mapping  $c \mapsto (y_k)_{k \in \mathbb{Z}}$  from (8.2.60) are *linear*!

(Recall the concept of a linear mapping as addressed in Rem. 3.1.8)

Summary:



*Remark 8.2.61* (Filtering in Fourier domain).

Consider (bi-)infinite signal  $(x_k)_{k \in \mathbb{Z}}$  sent through a finite linear time-invariant causal channel with impulse response  $(h_0, \dots, h_{n-1})^\top$  ( $\rightarrow$  Ex. 8.1.1).

➤ output signal, see (8.1.3)

$$y_k = \sum_{j=0}^{n-1} h_j x_{k-j}, \quad k \in \mathbb{Z}. \quad (8.2.62)$$

Fourier transforms of signals:

$$(y_k)_{k \in \mathbb{Z}} \leftrightarrow c(t), \quad (x_j)_{j \in \mathbb{Z}} \leftrightarrow b(t)$$

(Assume that  $(x_k)_{k \in \mathbb{Z}}$  satisfies (8.2.51))

$$\begin{aligned} c(t) &= \sum_{k \in \mathbb{Z}} y_k \exp(-2\pi ikt) = \sum_{k \in \mathbb{Z}} \sum_{j=0}^{n-1} h_j x_{k-j} \exp(-2\pi ikt) \\ \text{[shift summation index } k] &= \sum_{j=0}^{n-1} \sum_{k \in \mathbb{Z}} h_j x_k \exp(-2\pi jt) \exp(-2\pi ikt) \\ &= \underbrace{\left( \sum_{j=0}^{n-1} h_j \exp(-2\pi jt) \right)}_{\text{trigonometric polynomial of degree } n-1} b(t). \end{aligned} \quad (8.2.63)$$

➤ Discrete convolution of a signal with finite impulse response  $\leftrightarrow$  multiplication of Fourier transform with trigonometric polynomial.

## Conservation of power through Fourier transform:

Lemma 8.2.10 ➤ for Fourier matrix  $\mathbf{F}_n$ , see (8.2.9),  $\frac{1}{\sqrt{n}}\mathbf{F}_n$  is *unitary* ( $\rightarrow$  Def. 2.8.5)

$$\begin{array}{c} \text{Thm. 2.8.6} \\ \blacktriangleright \end{array} \quad \left\| \frac{1}{\sqrt{n}}\mathbf{F}_n \mathbf{y} \right\|_2 = \|\mathbf{y}\|_2 \quad (8.2.64)$$

Since DFT boils down to multiplication with  $\mathbf{F}_n$  ( $\rightarrow$  Def. 8.2.14), we conclude from (8.2.64)

$$c_k \text{ from (8.2.43)} \quad \Rightarrow \quad \frac{1}{n} \sum_{k=0}^{n-1} |c_k|^2 = \sum_{j=-m}^m |y_j|^2 \quad (8.2.65)$$

Now: function perspective  $c_k \leftrightarrow c(t_k)$

+ passing to the limit  $m \rightarrow \infty$

+ Riemann summation (see above)

$$(8.2.65) \quad \xrightarrow{m \rightarrow \infty} \int_0^1 |c(t)|^2 dt = \sum_{j \in \mathbb{Z}} |y_j|^2. \quad (8.2.66)$$

**Theorem 8.2.67** (Isometry property of Fourier transform).

If  $\sum_{k \in \mathbb{Z}} |y_k|^2 < \infty$ , then

$$c(t) = \sum_{k \in \mathbb{Z}} y_k \exp(-2\pi ikt) \Rightarrow \int_0^1 |c(t)|^2 dt = \sum_{k \in \mathbb{Z}} |y_k|^2.$$

Recalling the concept of the  $L^2$ -norm of a function, see (3.5.3), the theorem can be stated as follows:

Thm. 8.2.67  $\leftrightarrow$  The  $L^2$ -norm of a Fourier transform agrees with the Euclidean norm of the corresponding signal.

Note: Euclidean norm of a sequence  $\|(y_k)_{k \in \mathbb{Z}}\|_2^2 := \sum_{k \in \mathbb{Z}} |y_k|^2$ .

## 8.3 Fast Fourier Transform (FFT) [13, Sect. 8.7.3], [35, Sect. 53], [51, Sect. 10.9.2]

At first glance (at (8.2.15)): DFT in  $\mathbb{C}^n$  seems to require asymptotic computational effort of  $O(n^2)$  (matrix  $\times$  vector multiplication with dense matrix).

*Example 8.3.1* (Efficiency of `fft`).

`tic-toc`-timing in MATLAB: compare `fft`, loop based implementation, and direct matrix multiplication  
(MATLAB V6.5, Linux, Mobile Intel Pentium 4 - M CPU 2.40GHz, minimum over 5 runs)

Code 8.3.2: timing of different implementations of DFT

```
1 res = [];  
2 for n=1:1:3000, y = rand(n,1); c = zeros(n,1);  
3   t1 = realmax; for k=1:5, tic;
```

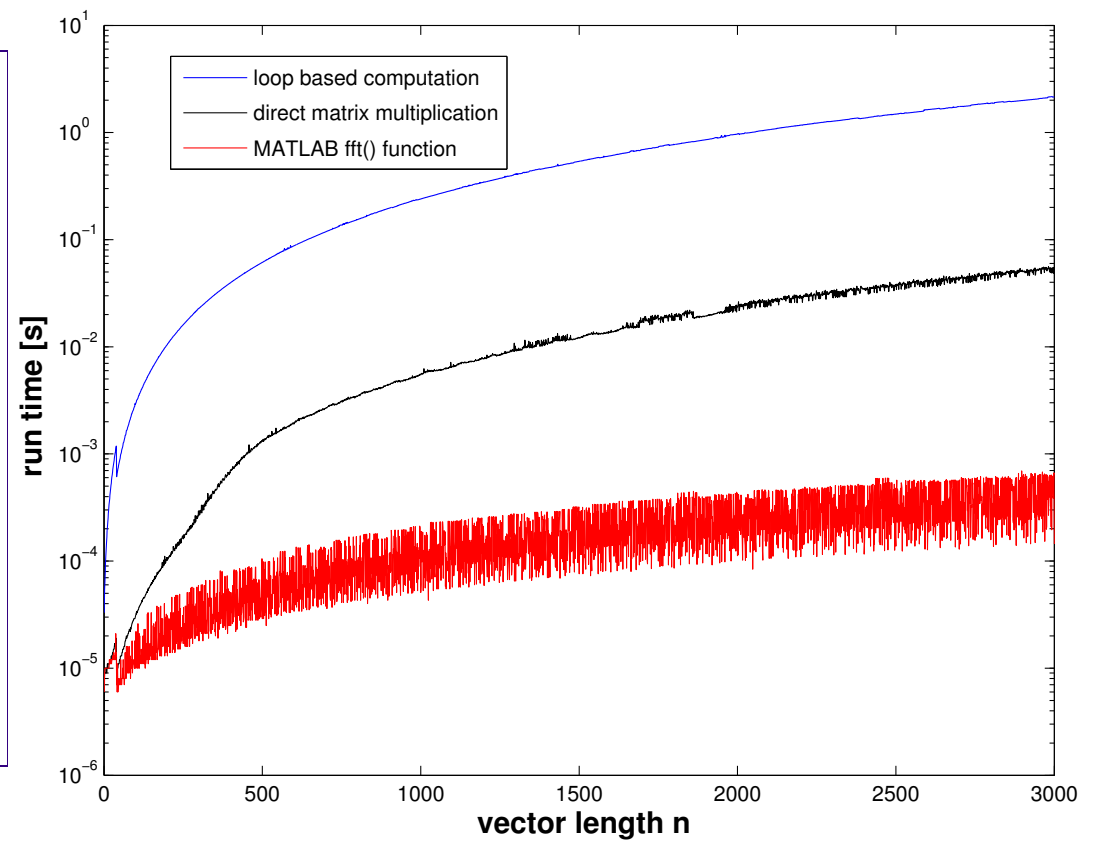
```
4  omega = exp(-2*pi*i/n); c(1) = sum(y); s = omega;
5  for j=2:n, c(j) = y(n);
6      for k=n-1:-1:1, c(j) = c(j)*s+y(k); end
7      s = s*omega;
8  end
9  t1 = min(t1, toc);
0  end
1  [I,J] = meshgrid(0:n-1,0:n-1); F = exp(-2*pi*i*I.*J/n);
2  t2 = realmax; for k=1:5, tic; c = F*y; t2 = min(t2, toc); end
3  t3 = realmax; for k=1:5, tic; d = fft(y); t3 = min(t3, toc); end
4  res = [res; n t1 t2 t3];
5  end
6
7  figure('name','FFT timing');
8  semilogy(res(:,1),res(:,2),'b-',res(:,1),res(:,3),'k-',
9      res(:,1),res(:,4),'r-');
10 ylabel('\bf run time [s]','FontSize',14);
11 xlabel('\bf vector length n','FontSize',14);
12 legend('loop based computation','direct matrix
13     multiplication','MATLAB fft() function',1);
14 print -deps2c '../PICTURES/fftttime.eps'
```

## MATLAB-CODE naive DFT-implementation

```

c = zeros(n,1);
omega = exp(-2*pi*i/n);
c(1) = sum(y); s = omega;
for j=2:n
    c(j) = y(n);
    for k=n-1:-1:1
        c(j) = c(j)*s+y(k);
    end
    s = s*omega;
end

```



Incredible! The MATLAB `fft()`-function clearly beats the  $O(n^2)$  asymptotic complexity of the other implementations. Note the logarithmic scale!





The secret of MATLAB's `fft()`:

the **Fast Fourier Transform** algorithm [18]

(discovered by C.F. Gauss in 1805, rediscovered by Cooley & Tuckey in 1965,  
 one of the “top ten algorithms of the century”).

An elementary manipulation of (8.2.15) for  $n = 2m$ ,  $m \in \mathbb{N}$ :

$$\begin{aligned}
 c_k &= \sum_{j=0}^{n-1} y_j e^{-\frac{2\pi i}{n} jk} \\
 &= \sum_{j=0}^{m-1} y_{2j} e^{-\frac{2\pi i}{n} 2jk} + \sum_{j=0}^{m-1} y_{2j+1} e^{-\frac{2\pi i}{n} (2j+1)k} \\
 &= \underbrace{\sum_{j=0}^{m-1} y_{2j} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}}}_{=:\tilde{c}_k^{\text{even}}} + e^{-\frac{2\pi i}{n} k} \cdot \underbrace{\sum_{j=0}^{m-1} y_{2j+1} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}}}_{=:\tilde{c}_k^{\text{odd}}}.
 \end{aligned} \tag{8.3.3}$$

Note  $m$ -periodicity:  $\tilde{c}_k^{\text{even}} = \tilde{c}_{k+m}^{\text{even}}$ ,  $\tilde{c}_k^{\text{odd}} = \tilde{c}_{k+m}^{\text{odd}}$ .

Note:  $\tilde{c}_k^{\text{even}}$ ,  $\tilde{c}_k^{\text{odd}}$  from DFTs of length  $m$ !

$$\text{with } \mathbf{y}_{\text{even}} := (y_0, y_2, \dots, y_{n-2})^\top \in \mathbb{C}^m: \quad (\tilde{c}_k^{\text{even}})_{k=0}^{m-1} = \mathbf{F}_m \mathbf{y}_{\text{even}},$$

$$\text{with } \mathbf{y}_{\text{odd}} := (y_1, y_3, \dots, y_{n-1})^\top \in \mathbb{C}^m: \quad (\tilde{c}_k^{\text{odd}})_{k=0}^{m-1} = \mathbf{F}_m \mathbf{y}_{\text{odd}}.$$

(8.3.3):

DFT of length  $2m = 2 \times$  DFT of length  $m$  +  $2m$  additions & multiplications

Idea:

divide & conquer recursion

(for DFT of length  $n = 2^L$ )

FFT-algorithm

Code 8.3.4: Recursive FFT

```

1 function c = fftrec(y)
2 n = length(y);
3 if (n == 1), c = y; return;
4 else
5     c1 = fftrec(y(1:2:n));
6     c2 = fftrec(y(2:2:n));
7     c = [c1;c1] +
          (exp(-2*pi*i/n) .^ ((0:n-1)'))
          .* [c2;c2];
8 end

```

Computational cost of `fftrec`:

1 × DFT of length  $2^L$ 2 × DFT of length  $2^{L-1}$ 4 × DFT of length  $2^{L-2}$  $2^L$  × DFT of length 1

Code 8.3.3: each level of the recursion requires  $O(2^L)$  elementary operations.

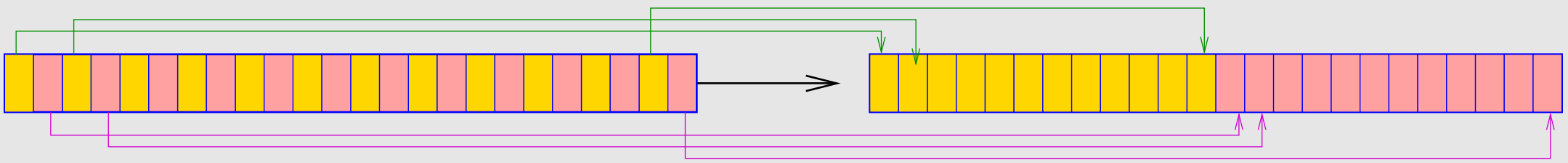
Asymptotic complexity of FFT algorithm,  $n = 2^L$ :  $O(L2^L) = O(n \log_2 n)$

( MATLAB `fft`-function: cost  $\approx 5n \log_2 n$  ).

*Remark 8.3.5* (FFT algorithm by matrix factorization).

For  $n = 2m$ ,  $m \in \mathbb{N}$ ,

permutation  $P_m^{\text{OE}}(1, \dots, n) = (1, 3, \dots, n-1, 2, 4, \dots, n)$ .



$$\omega_n^{2j} = \omega_m^j$$

permutation of rows  $P_m^{\text{OE}} \mathbf{F}_n =$

$$\left( \begin{array}{c|c} \mathbf{F}_m & \mathbf{F}_m \\ \hline \mathbf{F}_m \begin{pmatrix} \omega_n^0 \\ \omega_n^1 \\ \dots \\ \omega_n^{n/2-1} \end{pmatrix} & \mathbf{F}_m \begin{pmatrix} \omega_n^{n/2} \\ \omega_n^{n/2+1} \\ \dots \\ \omega_n^{n-1} \end{pmatrix} \end{array} \right)$$

$$\left( \begin{array}{c|c} \mathbf{F}_m & \\ \hline & \mathbf{F}_m \end{array} \right) \left( \begin{array}{c|c} \mathbf{I} & \mathbf{I} \\ \hline \omega_n^0 & -\omega_n^0 \\ \omega_n^1 & -\omega_n^1 \\ \dots & \dots \\ \omega_n^{n/2-1} & -\omega_n^{n/2-1} \end{array} \right)$$

Example: factorization of Fourier matrix for  $n = 10$

$$P_5^{\text{OE}} \mathbf{F}_{10} = \left( \begin{array}{ccccc|ccccc} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^8 & \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^8 \\ \omega^0 & \omega^4 & \omega^8 & \omega^2 & \omega^6 & \omega^0 & \omega^4 & \omega^8 & \omega^2 & \omega^6 \\ \omega^0 & \omega^6 & \omega^2 & \omega^8 & \omega^4 & \omega^0 & \omega^6 & \omega^2 & \omega^8 & \omega^4 \\ \omega^0 & \omega^8 & \omega^6 & \omega^4 & \omega^2 & \omega^0 & \omega^8 & \omega^6 & \omega^4 & \omega^2 \\ \hline \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 & \omega^8 & \omega^9 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 & \omega^2 & \omega^5 & \omega^8 & \omega^1 & \omega^4 & \omega^7 \\ \omega^0 & \omega^5 & \omega^0 & \omega^5 & \omega^0 & \omega^5 & \omega^0 & \omega^5 & \omega^0 & \omega^5 \\ \omega^0 & \omega^7 & \omega^4 & \omega^1 & \omega^8 & \omega^5 & \omega^2 & \omega^9 & \omega^6 & \omega^3 \\ \omega^0 & \omega^9 & \omega^8 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{array} \right), \quad \omega := \omega_{10}.$$



What if  $n \neq 2^L$ ? Quoted from MATLAB manual:

To compute an  $n$ -point DFT when  $n$  is composite (that is, when  $n = pq$ ), the FFTW library decomposes the problem using the Cooley-Tukey algorithm, which first computes  $p$  transforms of size  $q$ ,

and then computes  $q$  transforms of size  $p$ . The decomposition is applied recursively to both the  $p$ - and  $q$ -point DFTs until the problem can be solved using one of several machine-generated fixed-size "codelets." The codelets in turn use several algorithms in combination, including a variation of Cooley-Tukey, a prime factor algorithm, and a split-radix algorithm. The particular factorization of  $n$  is chosen heuristically.

The execution time for fft depends on the length of the transform. It is fastest for powers of two. It is almost as fast for lengths that have only small prime factors. It is typically several times slower for lengths that are prime or which have large prime factors → Ex. 8.3.1.

*Remark 8.3.6* (FFT based on general factorization).

Fast Fourier transform algorithm for DFT of length  $n = pq$ ,  $p, q \in \mathbb{N}$  (Cooley-Tuckey-Algorithm)

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{jk} \stackrel{[j=:lp+m]}{=} \sum_{m=0}^{p-1} \sum_{l=0}^{q-1} y_{lp+m} e^{-\frac{2\pi i}{pq}(lp+m)k} = \sum_{m=0}^{p-1} \omega_n^{mk} \sum_{l=0}^{q-1} y_{lp+m} \omega_q^{l(k \bmod q)}. \quad (8.3.7)$$

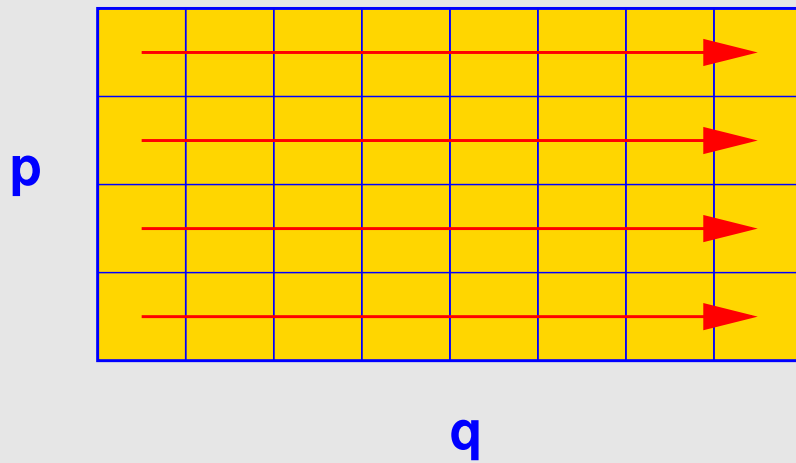
Step I: perform  $p$  DFTs of length  $q$   $z_{m,k} := \sum_{l=0}^{q-1} y_{lp+m} \omega_q^{lk}$ ,  $0 \leq m < p, 0 \leq k < q$ .

Step II: for  $k =: rq + s$ ,  $0 \leq r < p$ ,  $0 \leq s < q$

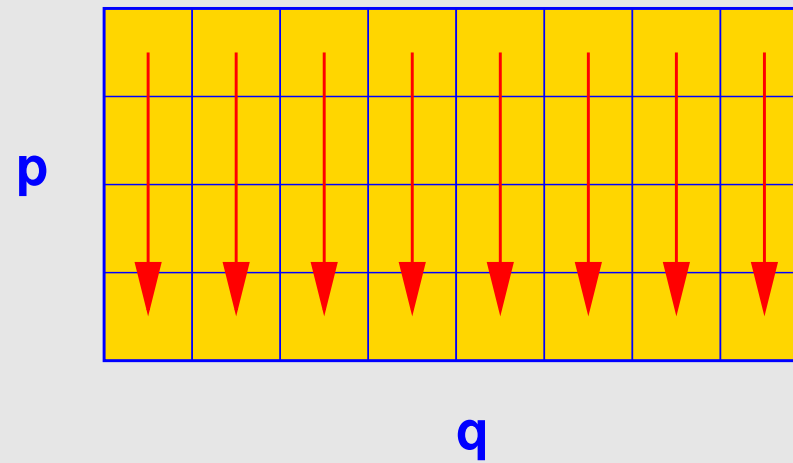
$$c_{rq+s} = \sum_{m=0}^{p-1} e^{-\frac{2\pi i}{pq}(rq+s)m} z_{m,s} = \sum_{m=0}^{p-1} (\omega_p^{ms} z_{m,s}) \omega_p^{mr}$$

and hence  $q$  DFTs of length  $p$  give all  $c_k$ .

Step I



Step II



*Remark 8.3.8* (FFT for prime  $n$ ).

When  $n \neq 2^L$ , even the Cooley-Tukey algorithm of Rem. 8.3.6 will eventually lead to a DFT for a vector with prime length.

Quoted from the MATLAB manual:

When  $n$  is a prime number, the FFTW library first decomposes an  $n$ -point problem into three  $(n-1)$ -point problems using Rader's algorithm [52]. It then uses the Cooley-Tukey decomposition described above to compute the  $(n-1)$ -point DFTs.

Details of Rader's algorithm: a theorem from number theory:

$$\forall p \in \mathbb{N} \text{ prime} \quad \exists g \in \{1, \dots, p-1\}: \{g^k \pmod{p} : k = 1, \dots, p-1\} = \{1, \dots, p-1\},$$

► permutation  $P_{p,g} : \{1, \dots, p-1\} \mapsto \{1, \dots, p-1\}$ ,  $P_{p,g}(k) = g^k \pmod{p}$ ,  
reversing permutation  $P_k : \{1, \dots, k\} \mapsto \{1, \dots, k\}$ ,  $P_k(i) = k - i + 1$ .

For Fourier matrix  $\mathbf{F} = (f_{ij})_{i,j=1}^p$ :  $P_{p-1} P_{p,g} (f_{ij})_{i,j=2}^p P_{p,g}^\top$  is circulant.



Example for  $p = 13$ :  
 $g = 2$  , permutation:  $(2\ 4\ 8\ 3\ 6\ 12\ 11\ 9\ 5\ 10\ 7\ 1)$  .

$$\mathbf{F}_{13} \longrightarrow \begin{array}{c|cccccccccccc} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \hline \omega^0 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 \\ \omega^0 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 \\ \omega^0 & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} \\ \omega^0 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 \\ \omega^0 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 \\ \omega^0 & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} \\ \omega^0 & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} \\ \omega^0 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 \\ \omega^0 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 \\ \omega^0 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 \\ \omega^0 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 \\ \omega^0 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 \end{array}$$

Then apply fast (FFT based!) algorithms for multiplication with circulant matrices to right lower  $(n - 1) \times (n - 1)$  block of permuted Fourier matrix .



Asymptotic complexity of  $\text{c=fft}(y)$  for  $y \in \mathbb{C}^n = O(n \log n)$ .

Asymptotic complexity of discrete periodic convolution/multiplication with circulant matrix, see Code 8.2.18:

$$\text{Cost}(z = \text{pconvfft}(u, x), \mathbf{u}, \mathbf{x} \in \mathbb{C}^n) = O(n \log n).$$

Asymptotic complexity of discrete convolution, see Code 8.2.20:

$$\text{Cost}(z = \text{myconv}(h, x), \mathbf{h}, \mathbf{x} \in \mathbb{C}^n) = O(n \log n).$$

## 8.4 Trigonometric transformations [35, Sect. 55]

Keeping in mind  $\exp(2\pi i x) = \cos(2\pi x) + i \sin(2\pi x)$  we may also consider the real/imaginary parts of the Fourier basis vectors  $(\mathbf{F}_n)_{:,j}$  as bases of  $\mathbb{R}^n$  and define the corresponding basis transformation. They can all be realized by means of `fft` with an asymptotic computational effort of  $O(n \log n)$ .

Details are given in the sequel.

## 8.4.1 Sine transform

Another trigonometric basis transform in  $\mathbb{R}^{n-1}$ ,  $n \in \mathbb{N}$ :

$$\left\{ \begin{array}{c} \left( \begin{array}{c} 1 \\ 0 \\ \vdots \\ 0 \end{array} \right) \left( \begin{array}{c} 0 \\ 1 \\ \vdots \\ 0 \end{array} \right) \dots \left( \begin{array}{c} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{array} \right) \left( \begin{array}{c} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{array} \right) \end{array} \right\} \leftarrow \left\{ \begin{array}{c} \left( \begin{array}{c} \sin(\frac{\pi}{n}) \\ \sin(\frac{2\pi}{n}) \\ \vdots \\ \sin(\frac{(n-1)\pi}{n}) \end{array} \right) \left( \begin{array}{c} \sin(\frac{2\pi}{n}) \\ \sin(\frac{4\pi}{n}) \\ \vdots \\ \sin(\frac{2(n-1)\pi}{n}) \end{array} \right) \dots \left( \begin{array}{c} \sin(\frac{(n-1)\pi}{n}) \\ \sin(\frac{2(n-1)\pi}{n}) \\ \vdots \\ \sin(\frac{(n-1)^2\pi}{n}) \end{array} \right) \end{array} \right\}$$

Basis transform matrix (sine basis  $\rightarrow$  standard basis):  $\mathbf{S}_n := (\sin(jk\pi/n))_{j,k=1}^{n-1} \in \mathbb{R}^{n-1, n-1}$ .

**Lemma 8.4.1** (Properties of the sine matrix).

$\sqrt{2/n} \mathbf{S}_n$  is real, symmetric and orthogonal ( $\rightarrow$  Def. 2.8.5)

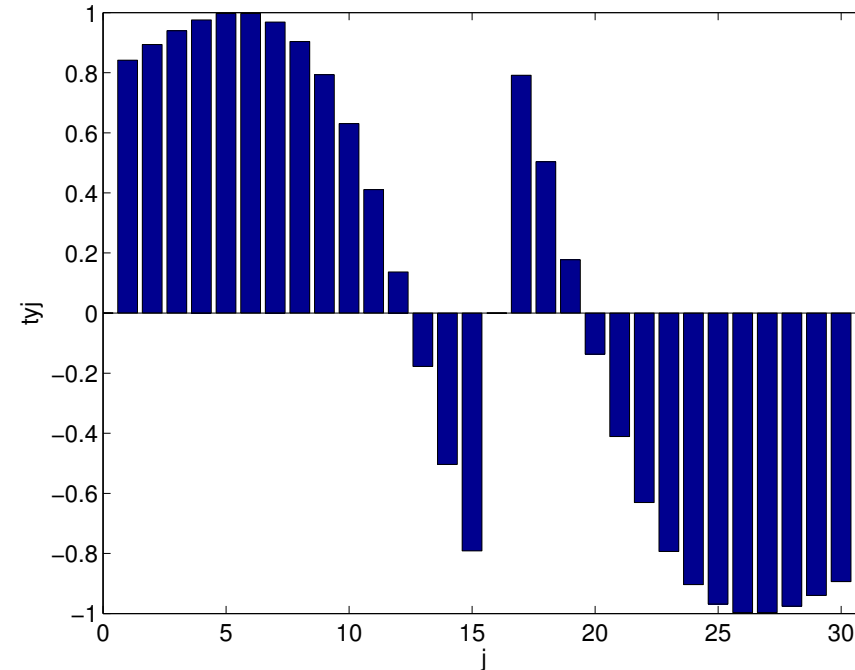
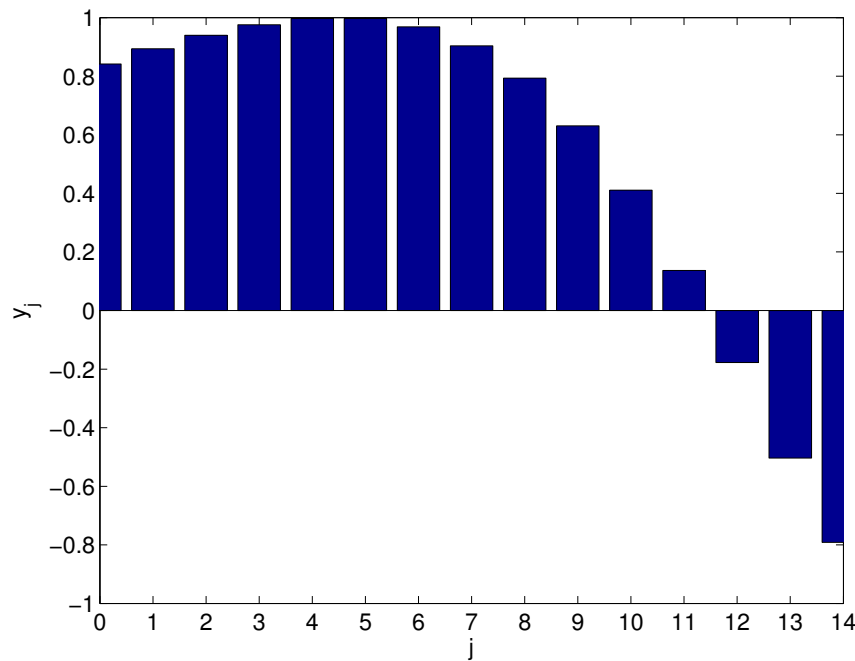
Sine transform:

$$s_k = \sum_{j=1}^{n-1} y_j \sin(\pi jk/n), \quad k = 1, \dots, n-1.$$

(8.4.2)

DFT-based algorithm for the sine transform ( $\hat{=}$   $\mathbf{S}_n \times \text{vector}$ ):

“wrap around”:  $\tilde{\mathbf{y}} \in \mathbb{R}^{2n}$ :  $\tilde{y}_j = \begin{cases} y_j & , \text{ if } j = 1, \dots, n-1, \\ 0 & , \text{ if } j = 0, n, \\ -y_{2n-j} & , \text{ if } j = n+1, \dots, 2n-1. \end{cases}$  ( $\tilde{\mathbf{y}}$  “odd”)



$$\begin{aligned}
 (\mathbf{F}_{2n}\tilde{\mathbf{y}})_k &\stackrel{(8.2.15)}{=} \sum_{j=1}^{2n-1} \tilde{y}_j e^{-\frac{2\pi}{2n}kj} \\
 &= \sum_{j=1}^{n-1} y_j e^{-\frac{\pi}{n}kj} - \sum_{j=n+1}^{2n-1} y_{2n-j} e^{-\frac{\pi}{n}kj} \\
 &= \sum_{j=1}^{n-1} y_j (e^{-\frac{\pi}{n}kj} - e^{\frac{\pi}{n}kj}) \\
 &= -2i (\mathbf{S}_n \mathbf{y})_k, \quad k = 1, \dots, n-1.
 \end{aligned}$$

Wrap-around implementation

```

function c = sinetrans(y)
n = length(y)+1;
yt = [0,y,0,-y(end:-1:1)];
ct = fft(yt);
c = -ct(2:n)/(2*i);

```

MATLAB-CODE sine transform

**Remark 8.4.3** (Sine transform via DFT of half length).

Step ①: transform of the coefficients

$$\tilde{y}_j = \sin(j\pi/n)(y_j + y_{n-j}) + \frac{1}{2}(y_j - y_{n-j}), \quad j = 1, \dots, n-1, \quad \tilde{y}_0 = 0.$$

Step ②: real DFT ( $\rightarrow$  Sect. 8.2.3) of  $(\tilde{y}_0, \dots, \tilde{y}_{n-1}) \in \mathbb{R}^n$ :

$$c_k := \sum_{j=0}^{n-1} \tilde{y}_j e^{-\frac{2\pi i}{n}jk}$$

Hence

$$\begin{aligned} \operatorname{Re}\{c_k\} &= \sum_{j=0}^{n-1} \tilde{y}_j \cos\left(-\frac{2\pi i}{n}jk\right) = \sum_{j=1}^{n-1} (y_j + y_{n-j}) \sin\left(\frac{\pi j}{n}\right) \cos\left(\frac{2\pi i}{n}jk\right) \\ &= \sum_{j=0}^{n-1} 2y_j \sin\left(\frac{\pi j}{n}\right) \cos\left(\frac{2\pi i}{n}jk\right) = \sum_{j=0}^{n-1} y_j \left( \sin\left(\frac{2k+1}{n}\pi j\right) - \sin\left(\frac{2k-1}{n}\pi j\right) \right) \\ &= s_{2k+1} - s_{2k-1} . \\ \operatorname{Im}\{c_k\} &= \sum_{j=0}^{n-1} \tilde{y}_j \sin\left(-\frac{2\pi i}{n}jk\right) = -\sum_{j=1}^{n-1} \frac{1}{2}(y_j - y_{n-j}) \sin\left(\frac{2\pi i}{n}jk\right) = -\sum_{j=1}^{n-1} y_j \sin\left(\frac{2\pi i}{n}jk\right) \\ &= -s_{2k} . \end{aligned}$$

Step ③: extraction of  $s_k$

$$s_{2k+1}, \quad k = 0, \dots, \frac{n}{2} - 1 \quad \blacktriangleright \quad \text{from recursion } s_{2k+1} - s_{2k-1} = \operatorname{Re}\{c_k\}, \quad s_1 = \sum_{j=1}^{n-1} y_j \sin(\pi j/n),$$

$$s_{2k}, \quad k = 1, \dots, \frac{n}{2} - 2 \quad \blacktriangleright \quad s_{2k} = -\operatorname{Im}\{c_k\} .$$

MATLAB-Implementation (via a `fft` of length  $n/2$ ):

```

function s = sinetrans(y)
n = length(y)+1;
sinevals = imag(exp(i*pi/n).^(1:n-1));
yt = [0 (sinevals.*(y+y(end:-1:1)) + 0.5*(y-y(end:-1:1)))]';
c = fftreal(yt);
s(1) = dot(sinevals,y);
for k=2:N-1
if (mod(k,2) == 0), s(k) = -imag(c(k/2+1));
else, s(k) = s(k-2) + real(c((k-1)/2+1)); end
end

```



Application: diagonalization of local translation invariant linear operators.

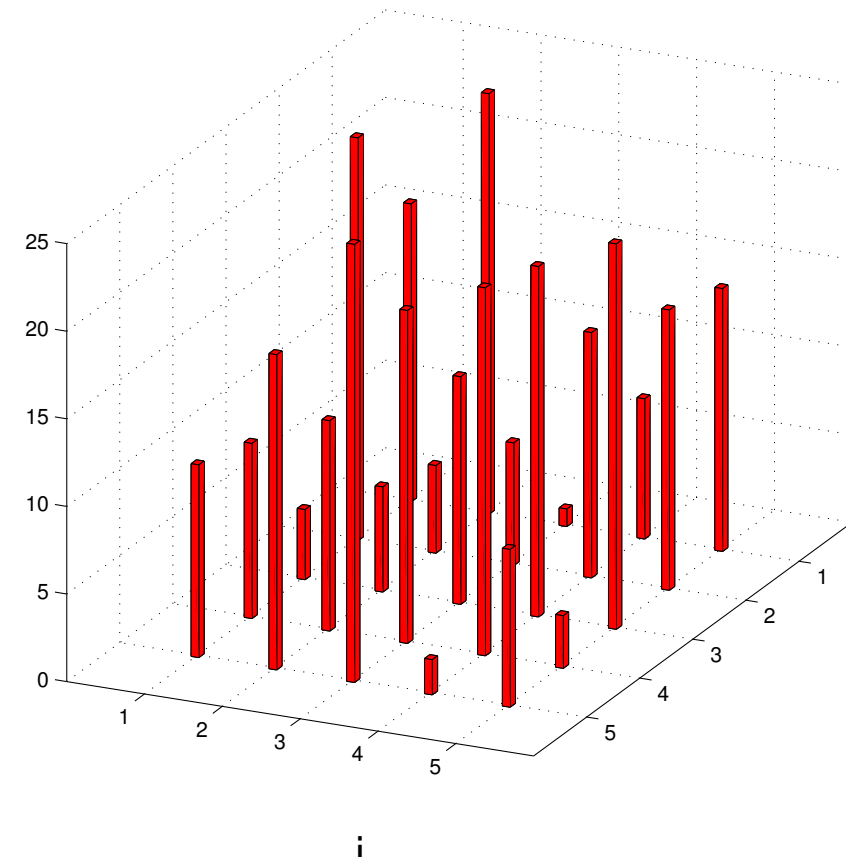
**5-points-stencil**-operator on  $\mathbb{R}^{n,n}$ ,  $n \in \mathbb{N}$ , in **grid representation**:

$$T : \mathbb{R}^{n,n} \mapsto \mathbb{R}^{n,n}, \quad \mathbf{X} \mapsto T(\mathbf{X})$$

$$(T(\mathbf{X}))_{ij} := c_x x_{ij} + c_y x_{i,j+1} + c_y x_{i,j-1} + c_x x_{i+1,j} + c_x x_{i-1,j}$$

with  $c, c_y, c_x \in \mathbb{R}$ , convention:  $x_{ij} := 0$  for  $(i, j) \notin \{1, \dots, n\}^2$ .

$\mathbf{X} \in \mathbb{R}^{n,n}$   
 $\updownarrow$   
 grid function  $\in \{1, \dots, n\}^2 \mapsto \mathbb{R}$

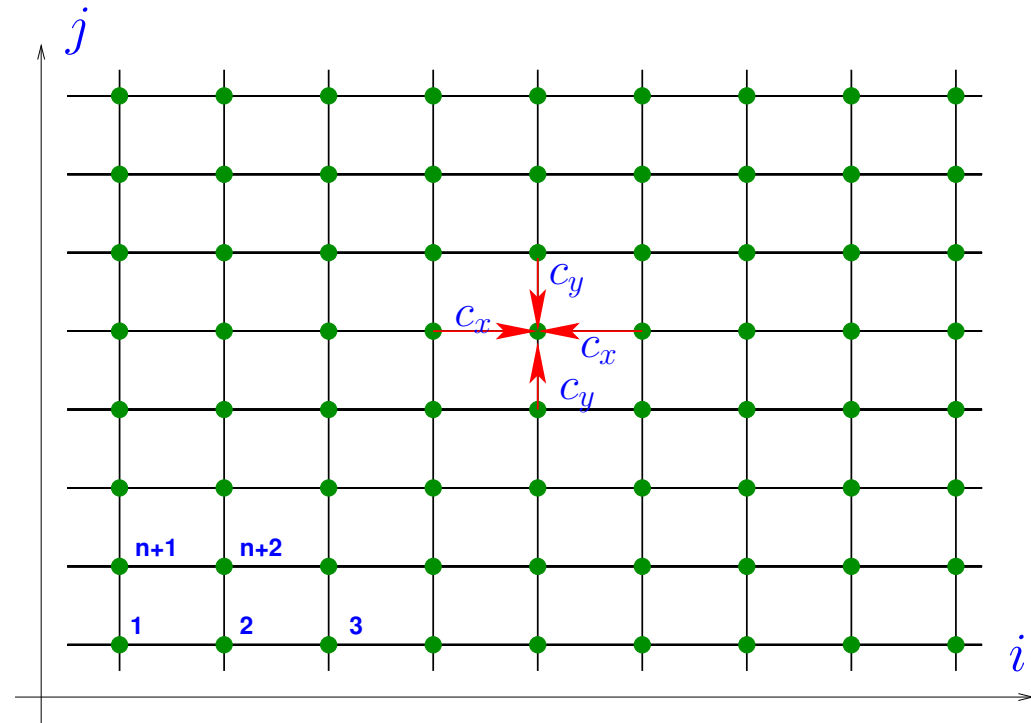


Identification  $\mathbb{R}^{n,n} \cong \mathbb{R}^{n^2}$ ,  $x_{ij} \sim \tilde{x}_{(j-1)n+i}$  gives matrix representation  $\mathbf{T} \in \mathbb{R}^{n^2, n^2}$  of  $T$ :



$$\mathbf{T} = \begin{pmatrix} \mathbf{C} & c_y \mathbf{I} & 0 & \dots & \dots & 0 \\ c_y \mathbf{I} & \mathbf{C} & c_y \mathbf{I} & & & \vdots \\ 0 & \dots & \dots & \dots & & \\ \vdots & & & & c_y \mathbf{I} & \mathbf{C} & c_y \mathbf{I} \\ 0 & \dots & \dots & 0 & c_y \mathbf{I} & \mathbf{C} \end{pmatrix} \in \mathbb{R}^{n^2, n^2},$$

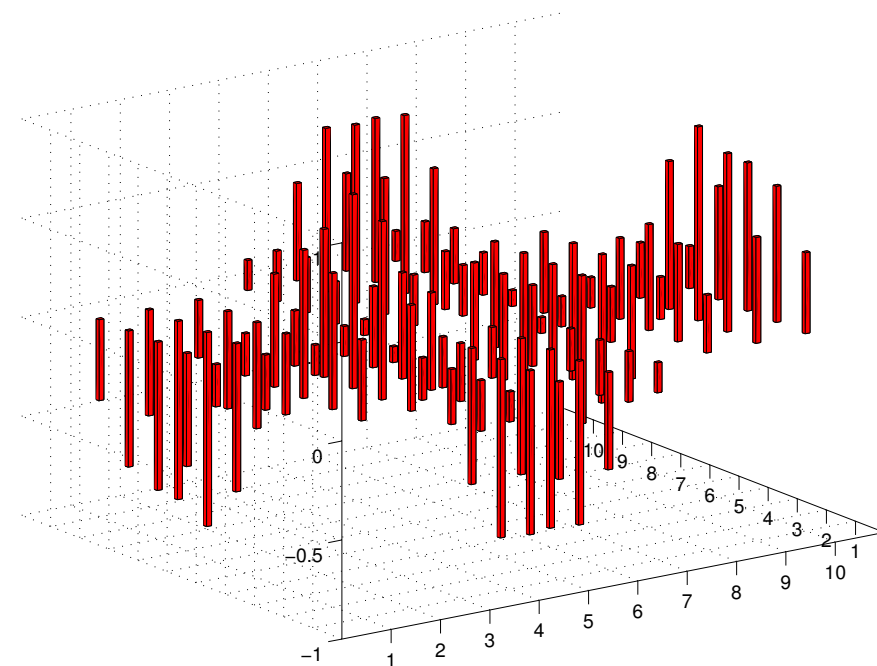
$$\mathbf{C} = \begin{pmatrix} c & c_x & 0 & \dots & \dots & 0 \\ c_x & c & c_x & & & \vdots \\ 0 & \dots & \dots & \dots & & \\ \vdots & & & & c_x & c & c_x \\ 0 & \dots & \dots & 0 & c_x & c \end{pmatrix} \in \mathbb{R}^{n, n}.$$



Sine basis of  $\mathbb{R}^{n, n}$ :

$$\mathbf{B}^{kl} = \left( \sin\left(\frac{\pi}{n+1}ki\right) \sin\left(\frac{\pi}{n+1}lj\right) \right)_{i,j=1}^n. \quad (8.4.4)$$

$n = 10$ : grid function  $\mathbf{B}^{2,3}$



$$\begin{aligned}
(T(\mathbf{B}^{kl}))_{ij} &= c \sin\left(\frac{\pi}{n}ki\right) \sin\left(\frac{\pi}{n}lj\right) + c_y \sin\left(\frac{\pi}{n}ki\right) \left( \sin\left(\frac{\pi}{n+1}l(j-1)\right) + \sin\left(\frac{\pi}{n+1}l(j+1)\right) \right) + \\
&\quad c_x \sin\left(\frac{\pi}{n}lj\right) \left( \sin\left(\frac{\pi}{n+1}k(i-1)\right) + \sin\left(\frac{\pi}{n+1}k(i+1)\right) \right) \\
&= \sin\left(\frac{\pi}{n}ki\right) \sin\left(\frac{\pi}{n}lj\right) \left( c + 2c_y \cos\left(\frac{\pi}{n+1}l\right) + 2c_x \cos\left(\frac{\pi}{n+1}k\right) \right)
\end{aligned}$$

Hence  $\mathbf{B}^{kl}$  is **eigenvector** of  $T \leftrightarrow \mathbf{T}$  corresponding to eigenvalue  $c + 2c_y \cos\left(\frac{\pi}{n+1}l\right) + 2c_x \cos\left(\frac{\pi}{n+1}k\right)$ .

Algorithm for basis transform:

$$\mathbf{X} = \sum_{k=1}^n \sum_{l=1}^n y_{kl} \mathbf{B}^{kl} \quad \Rightarrow \quad x_{ij} = \sum_{k=1}^n \sin\left(\frac{\pi}{n+1}ki\right) \sum_{l=1}^n y_{kl} \sin\left(\frac{\pi}{n+1}lj\right).$$

MATLAB-CODE two dimensional sine tr.

```

function C = sinft2d(Y)
[m,n] = size(Y);
C = fft([zeros(1,n); Y;...
        zeros(1,n);...
        -Y(end:-1:1,:)]);
C = i*C(2:m+1,:)/2;
C = fft([zeros(1,m); C;...
        zeros(1,m);...
        -C(end:-1:1,:)]);
C = i*C(2:n+1,:)/2;

```

Hence nested sine transforms ( $\rightarrow$  Sect. 8.2.4)  
for rows/columns of  $\mathbf{Y} = (y_{kl})_{k,l=1}^n$ .

Here: implementation of sine transform (8.4.2)  
with “wrapping”-technique.

```
function X = fftsolve(B,c,cx,cy)
[m,n] = size(B);
[I,J] = meshgrid(1:m,1:n);
X = 4*sinft2d(sinft2d(B)...
    ./ (c+2*cx*cos(pi/(n+1)*I)+...
        2*cy*cos(pi/(m+1)*J)))...
    / ((m+1)*(n+1));
```

translation invariant linear operators

## Diagonalization of $\mathbf{T}$ via 2D sine transform



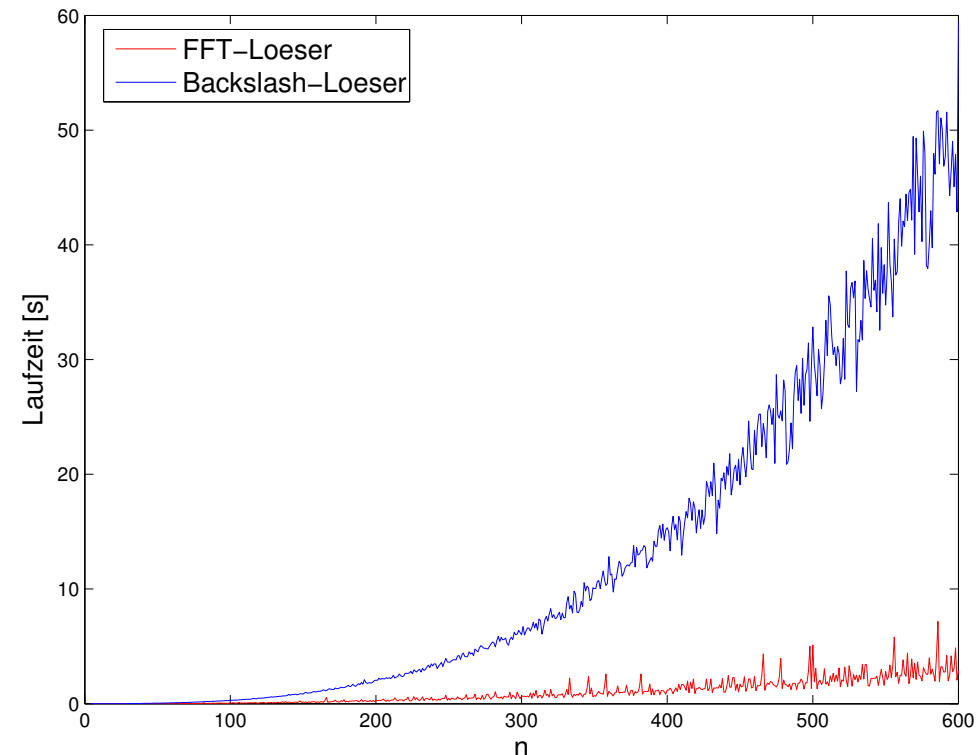
efficient algorithm

for solving linear system of equations  $\mathbf{T}(\mathbf{X}) = \mathbf{B}$ computational cost  $O(n^2 \log n)$ .

*Example 8.4.5* (Efficiency of FFT-based LSE-solver).

tic-toc-timing (MATLAB V7, Linux, Intel Pentium 4 Mobile CPU 1.80GHz)

```
A = gallery('poisson',n);
B = magic(n);
b = reshape(B,n*n,1);
tic;
C = fftsolve(B,4,-1,-1);
t1 = toc;
tic; x = A\b; t2 = toc;
```



## 8.4.2 Cosine transform

Another trigonometric basis transform in  $\mathbb{R}^n$ ,  $n \in \mathbb{N}$ :

standard basis of  $\mathbb{R}^n$

“cosine basis”

$$\left\{ \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\} \leftarrow \left\{ \begin{pmatrix} 2^{-1/2} \\ \cos(\frac{\pi}{2n}) \\ \cos(\frac{2\pi}{2n}) \\ \vdots \\ \cos(\frac{(n-1)\pi}{2n}) \end{pmatrix}, \begin{pmatrix} 2^{-1/2} \\ \cos(\frac{3\pi}{2n}) \\ \cos(\frac{6\pi}{2n}) \\ \vdots \\ \cos(\frac{3(n-1)\pi}{2n}) \end{pmatrix}, \dots, \begin{pmatrix} 2^{-1/2} \\ \cos(\frac{(2n-1)\pi}{2n}) \\ \cos(\frac{2(2n-1)\pi}{2n}) \\ \vdots \\ \cos(\frac{(n-1)(2n-1)\pi}{2n}) \end{pmatrix} \right\}.$$

R. Hiptmair  
rev 38286,  
November  
17, 2011

Basis transform matrix (cosine basis  $\rightarrow$  standard basis):

$$\mathbf{C}_n = (c_{ij}) \in \mathbb{R}^{n,n} \quad \text{with} \quad c_{ij} = \begin{cases} 2^{-1/2} & , \text{ if } i = 1 , \\ \cos((i-1)\frac{2j-1}{2n}\pi) & , \text{ if } i > 1 . \end{cases}$$

**Lemma 8.4.6** (Properties of cosine matrix).

$$\sqrt{2/n} \mathbf{C}_n \text{ is real and orthogonal } (\rightarrow \text{Def. 2.8.5}).$$

Note:  $C_n$  is not symmetric

cosine transform:

$$c_k = \sum_{j=0}^{n-1} y_j \cos\left(k \frac{2j+1}{2n} \pi\right), \quad k = 1, \dots, n-1, \quad (8.4.7)$$

$$c_0 = \frac{1}{\sqrt{2}} \sum_{j=0}^{n-1} y_j.$$

MATLAB-implementation of  $Cy$  ("wrapping"-technique):

MATLAB-CODE cosine transform

```
function c = costrans(y)
n = length(y);
z = fft([y, y(end:-1:1)]);
c = real([z(1)/(2*sqrt(2)), ...
         0.5*(exp(-i*pi/(2*n)).^(1:n-1)).*z(2:n)]);
```

MATLAB-implementation of  $C_n^{-1}y$  ("Wrapping"-technique):

MATLAB-CODE : Inverse cosine transform

```
function y=icostrans(c)
n = length(c);
y = [sqrt(2)*c(1), (exp(i*pi/(2*n)).^(1:n-1)).*c(2:end)];
y = ifft([y,0,conj(y(end:-1:2))]);
y = 2*y(1:n);
```

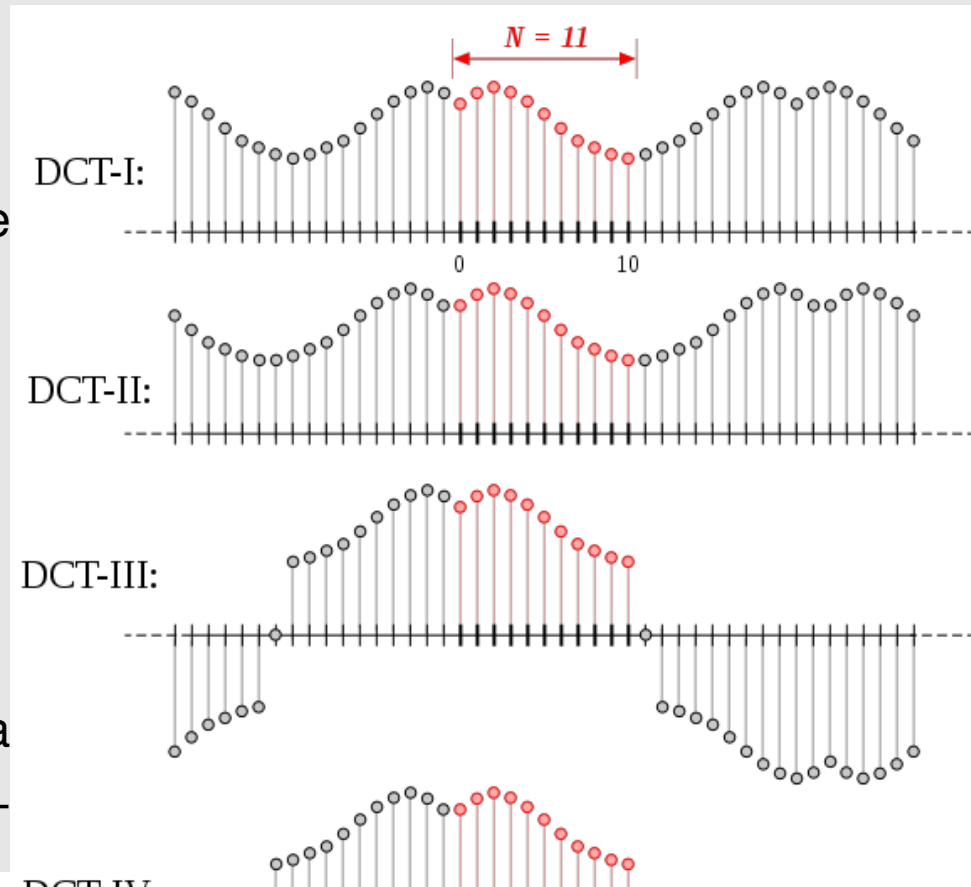
Remark 8.4.8 (Cosine transforms for compression).

The cosine transforms discussed above are named DCT-II and DCT-III.

Various cosine transforms arise by imposing various boundary conditions:

- DCT-II: even around  $-1/2$  and  $N - 1/2$
- DCT-III: even around  $0$  and odd around  $N$

DCT-II is used in JPEG-compression while a slightly modified DCT-IV makes the main component of MP3, AAC, and WMA formats



# 8.5 Toeplitz matrix techniques

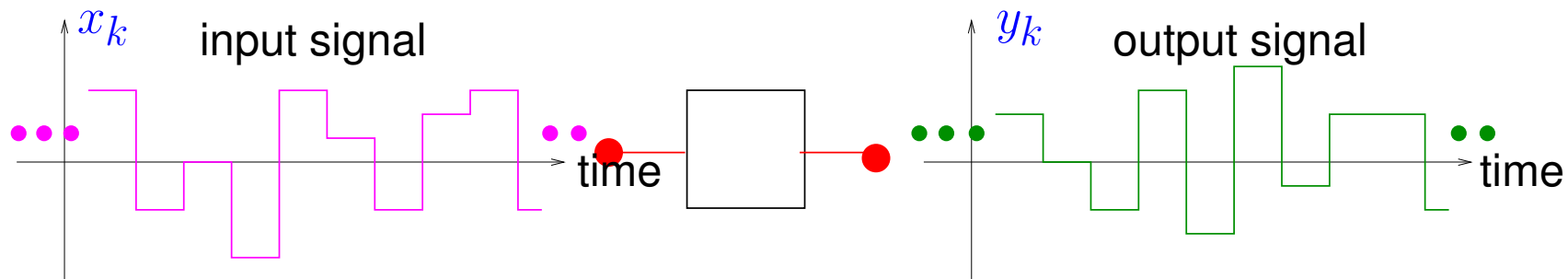
Example 8.5.1 (Parameter identification for linear time-invariant filters).

- $(x_k)_{k \in \mathbb{Z}}$   $m$ -periodic discrete signal = *known* input
- $(y_k)_{k \in \mathbb{Z}}$   $m$ -periodic *measured*<sup>(\*)</sup> output signal of a **linear time-invariant filter**, see Ex. 8.1.1.  
 (\*)  $\rightarrow$  measurement errors !
- Known: impulse response of filter has maximal duration  $n\Delta t$ ,  $n \in \mathbb{N}$ ,  $n \leq m$

cf. (8.1.3)



$$\exists \mathbf{h} = (h_0, \dots, h_{n-1})^T \in \mathbb{R}^n, \quad n \leq m : \quad y_k = \sum_{j=0}^{n-1} h_j x_{k-j} . \quad (8.5.2)$$



If the  $y_k$  were exact, we could retrieve  $h_0, \dots, h_{n-1}$  by examining only  $y_0, \dots, y_{n-1}$  and inverting the discrete periodic convolution ( $\rightarrow$  Def. 8.1.13) using (8.2.13).

However, in case the  $y_k$  are affected by measurements errors we have to use all available  $y_k$  for a **least squares estimate** of the impulse response.

Parameter identification problem: seek  $\mathbf{h} = (h_0, \dots, h_{n-1})^\top \in \mathbb{R}^n$  with

$$\|\mathbf{A}\mathbf{h} - \mathbf{y}\|_2 = \left\| \begin{pmatrix} x_0 & x_{-1} & \cdots & \cdots & x_{1-n} \\ x_1 & x_0 & x_{-1} & & \vdots \\ \vdots & x_1 & x_0 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ x_{n-1} & & & x_1 & x_0 \\ x_n & x_{n-1} & & & x_1 \\ \vdots & & & & \vdots \\ x_{m-1} & \cdots & & \cdots & x_{m-n} \end{pmatrix} \begin{pmatrix} h_0 \\ \vdots \\ h_{n-1} \end{pmatrix} - \begin{pmatrix} y_0 \\ \vdots \\ y_{m-1} \end{pmatrix} \right\|_2 \rightarrow \min .$$

➤ **Linear least squares problem**,  $\rightarrow$  Ch. 7 with Toeplitz matrix  $\mathbf{A}$ :  $(\mathbf{A})_{ij} = x_{i-j}$ .



System matrix of normal equations ( $\rightarrow$  Sect. 7.1)

$$\mathbf{M} := \mathbf{A}^H \mathbf{A} \quad , \quad (\mathbf{M})_{ij} = \sum_{k=1}^m x_{k-i} x_{k-j} = z_{i-j} \quad \text{due to periodicity of } (x_k)_{k \in \mathbb{Z}} .$$



$\mathbf{M} \in \mathbb{R}^{n,n}$  is a *matrix with constant diagonals* & s.p.d.  
 (“constant diagonals”  $\Leftrightarrow$   $(\mathbf{M})_{i,j}$  depends only on  $i - j$ )



*Example 8.5.3* (Linear regression for stationary Markov chains).

Sequence of scalar random variables:  $(Y_k)_{k \in \mathbb{Z}} =$  Markov chain

Assume: **stationary** (time-independent) **correlation**

Expectation  $\mathcal{E}(Y_{i-j} Y_{i-k}) = u_{k-j} \quad \forall i, j, k \in \mathbb{Z} , \quad u_i = u_{-i} .$

Model: finite linear relationship

$$\exists \mathbf{x} = (x_1, \dots, x_n)^\top \in \mathbb{R}^n: Y_k = \sum_{j=1}^n x_j Y_{k-j} \quad \forall k \in \mathbb{Z}.$$

with *unknown* parameters  $x_j, j = 1, \dots, n$ : for fixed  $i \in \mathbb{Z}$

$$\text{Estimator} \quad \mathbf{x} = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}} E \left| Y_i - \sum_{j=1}^n x_j Y_{i-j} \right|^2 \quad (8.5.4)$$

$$\blacktriangleright \quad E|Y_i|^2 - 2 \sum_{j=1}^n x_j u_k + \sum_{k,j=1}^n x_k x_j u_{k-j} \rightarrow \min .$$

$$\blacktriangleright \quad \mathbf{x}^\top \mathbf{A} \mathbf{x} - 2\mathbf{b}^\top \mathbf{x} \rightarrow \min \quad \text{with} \quad \mathbf{b} = (u_k)_{k=1}^n, \quad \mathbf{A} = (u_{i-j})_{i,j=1}^n .$$

Lemma 5.1.3  $\Rightarrow$   $\mathbf{x}$  solves  $\mathbf{A} \mathbf{x} = \mathbf{b}$  (**Yule-Walker-equation**, see below)

$\mathbf{A} \hat{=}$  **Covariance matrix**: s.p.d. matrix with constant diagonals.

Matrices with constant diagonals occur frequently in mathematical models. They generalize circulant matrices  $\rightarrow$  Def. 8.1.17.

Note: “Information content” of a matrix  $\mathbf{M} \in \mathbb{K}^{m,n}$  with constant diagonals, that is,  $(\mathbf{M})_{i,j} = m_{i-j}$ , is  $m + n - 1$  numbers  $\in \mathbb{K}$ .

**Definition 8.5.5** (Toeplitz matrix).

$\mathbf{T} = (t_{ij})_{i,j=1}^n \in \mathbb{K}^{m,n}$  is a **Toeplitz matrix**, if there is a vector  $\mathbf{u} = (u_{-m+1}, \dots, u_{n-1}) \in \mathbb{K}^{m+n-1}$  such that  $t_{ij} = u_{j-i}$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ .

$$\mathbf{T} = \begin{pmatrix} u_0 & u_1 & \cdots & \cdots & u_{n-1} \\ u_{-1} & u_0 & u_1 & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & u_1 \\ u_{1-m} & \cdots & \cdots & u_{-1} & u_0 \end{pmatrix}$$

R. Hiptmair  
rev 38286,  
November  
17, 2011

## 8.5.1 Toeplitz matrix arithmetic

$\mathbf{T} = (u_{j-i}) \in \mathbb{K}^{m,n}$  = Toeplitz matrix with generating vector  $\mathbf{u} = (u_{-m+1}, \dots, u_{n-1}) \in \mathbb{C}^{m+n-1}$

Task: efficient evaluation of matrix  $\times$  vector product  $\mathbf{T}\mathbf{x}$ ,  $\mathbf{x} \in \mathbb{K}^n$

Note: the following extended matrix is **circulant** ( $\rightarrow$  Def. 8.1.17)

$$\mathbf{C} = \begin{pmatrix} \mathbf{T} & \mathbf{S} \\ \mathbf{S} & \mathbf{T} \end{pmatrix} = \left( \begin{array}{cccccc|cccccc} u_0 & u_1 & \cdots & & \cdots & u_{n-1} & 0 & u_{1-n} & \cdots & & \cdots & u_{-1} \\ u_{-1} & u_0 & u_1 & & & \vdots & u_{n-1} & 0 & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots & \vdots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots & \vdots & & & & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & u_1 & \vdots & & & \ddots & \ddots & u_{1-n} \\ u_{1-n} & \cdots & & \cdots & u_{-1} & u_0 & u_1 & & & & u_{n-1} & 0 \\ \hline 0 & u_{1-n} & \cdots & & \cdots & u_{-1} & u_0 & u_1 & \cdots & & \cdots & u_{n-1} \\ u_{n-1} & 0 & \ddots & & & \vdots & u_{-1} & u_0 & u_1 & & & \vdots \\ \vdots & \ddots & \ddots & & & \vdots & \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \vdots & \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \cdots & \ddots & u_{1-n} & \vdots & & & \ddots & \ddots & u_1 \\ u_1 & & & & u_{n-1} & 0 & u_{1-n} & \cdots & & \cdots & u_{-1} & u_0 \end{array} \right)$$

△ This example demonstrates the case  $m = n$ .

Remember Rem. 8.1.19: extension to a circulant matrix to convert general discrete convolution ( $\rightarrow$  Def. 8.1.7) to periodic discrete convolution ( $\rightarrow$  Def. 8.1.13), compare (8.1.22) and the above matrix augmentation.

In general:

```
T = toeplitz(u(0:-1:1-m), u(0:n-1));
S = toeplitz([0, u(n-1:-1:n-m+1)], [0, u(1-m:1:-1)]);
```

zero padding  $\xrightarrow{\blacktriangleright}$   $\mathbf{C} \begin{pmatrix} \mathbf{x} \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{T}\mathbf{x} \\ \mathbf{S}\mathbf{x} \end{pmatrix}$

$\blacktriangleright$  Computational effort  $O(n \log n)$  for computing  $\mathbf{T}\mathbf{x}$  (FFT based, Sect. 8.3)

## 8.5.2 The Levinson algorithm

Given: **S.p.d.** Toeplitz matrix  $\mathbf{T} = (u_{j-i})_{i,j=1}^n$ , generating vector  $\mathbf{u} = (u_{-n+1}, \dots, u_{n-1}) \in \mathbb{C}^{2n-1}$   
 (Symmetry  $\leftrightarrow u_{-k} = u_k$ , w.l.o.g  $u_0 = 1$ )

Task: efficient solution algorithm for LSE  $\mathbf{T}\mathbf{x} = \mathbf{b}$ ,  $\mathbf{b} \in \mathbb{C}^n$   
 (**Yule-Walker problem**)

*Recursive* (inductive) solution strategy:

Define:

- $\mathbf{T}_k := (u_{j-i})_{i,j=1}^k \in \mathbb{K}^{k,k}$  (left upper block of  $\mathbf{T}$ )  $\triangleright$   $\mathbf{T}_k$  is s.p.d. Toeplitz matrix,
- $\mathbf{x}^k \in \mathbb{K}^k$ :  $\mathbf{T}_k \mathbf{x}^k = (b_1, \dots, b_k)^\top \Leftrightarrow \mathbf{x}^k = \mathbf{T}_k^{-1} \mathbf{b}^k$ ,
- $\mathbf{u}^k := (u_1, \dots, u_k)^\top$

Block-partitioned LSE, cf. Rem. 2.1.12, Rem. 2.2.16

$$\mathbf{T}_{k+1} \mathbf{x}^{k+1} = \left( \begin{array}{c|c} \mathbf{T}_k & \begin{matrix} u_k \\ \vdots \\ u_1 \end{matrix} \\ \hline u_k \cdots u_1 & 1 \end{array} \right) \begin{pmatrix} \tilde{\mathbf{x}}^{k+1} \\ \hline x_{k+1}^{k+1} \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_k \\ \hline b_{k+1} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{b}}^{k+1} \\ \hline b_{k+1} \end{pmatrix} \quad (8.5.6)$$

Reversing permutation:  $P_k : \{1, \dots, k\} \mapsto \{1, \dots, k\}$ ,  $P_k(i) := k - i + 1$

$$\begin{aligned} \blacktriangleright \quad \tilde{\mathbf{x}}_{k+1} &= \mathbf{T}_k^{-1} (\tilde{\mathbf{b}}^{k+1} - x_{k+1}^{k+1} P_k \mathbf{u}^k) = \mathbf{x}^k - x_{k+1}^{k+1} \mathbf{T}_k^{-1} P_k \mathbf{u}^k, \\ x_{k+1}^{k+1} &= b_{k+1} - P_k \mathbf{u}^k \cdot \tilde{\mathbf{x}}^{k+1} = b_{k+1} - P_k \cdot \mathbf{x}^k + x_{k+1}^{k+1} P_k \cdot \mathbf{T}_k^{-1} P_k \mathbf{u}^k. \end{aligned} \quad (8.5.7)$$

Efficient algorithm by using *auxiliary vectors*:  $\mathbf{y}^k := \mathbf{T}_k^{-1} P_k \mathbf{u}^k$

$$\mathbf{x}^{k+1} = \begin{pmatrix} \tilde{\mathbf{x}}^{k+1} \\ x_{k+1}^{k+1} \end{pmatrix} \quad \text{with} \quad \begin{aligned} x_{k+1}^{k+1} &= (b_{k+1} - P_k \mathbf{u}^k) / \sigma_k \\ \tilde{\mathbf{x}}^{k+1} &= \mathbf{x}^k - x_{k+1}^{k+1} \mathbf{y}^k \end{aligned}, \quad \sigma_k := 1 - P_k \mathbf{u}^k \cdot \mathbf{y}^k. \quad (8.5.8)$$

### Code 8.5.9: Levinson algorithm

```

1 function [x, y] = levinson (u, b)
2 k = length (u) - 1;
3 if (k == 0), x = b(1); y = u(1);
   return; end
4 [xk, yk] =
   levinson (u(1:k), b(1:k));
5 sigma = 1 - dot (u(1:k), yk);
6 t =
   (b(k+1) - dot (u(k:-1:1), xk)) / sigma;
7 x = [ xk - t * yk(k:-1:1); t];
8 s =
   (u(k+1) - dot (u(k:-1:1), yk)) / sigma;
9 y = [ yk - s * yk(k:-1:1); s];

```

### Levinson algorithm

(recursive,  $u_{n+1}$  not used!)

Linear recursion:

Computational cost  $\sim (n - k)$  on level  $k$ ,  $k = 0, \dots, n - 1$

➤ Asymptotic complexity  $O(n^2)$

*Remark 8.5.10* (Fast Toeplitz solvers).

FFT-based algorithms for solving  $\mathbf{T}\mathbf{x} = \mathbf{b}$  with asymptotic complexity  $O(n \log^3 n)$  [60] !



## Supplementary and further reading:

[13, Sect. 8.5]: Very detailed and elementary presentation, but the discrete Fourier transform through trigonometric interpolation, which is not covered in this chapter. Hardly addresses discrete convolution.

[35, Ch. IX] presents the topic from a mathematical point of view stressing approximation and trigonometric interpolation. Good reference for algorithms for circulant and Toeplitz matrices.

[57, Ch. 10] also discusses the discrete Fourier transform with emphasis on interpolation and (least squares) approximation. The presentation of signal processing differs from that of the course.



There is a vast number of books and survey papers dedicated to discrete Fourier transforms, see, for instance, [18, 9]. Issues and technical details way beyond the scope of the course are treated there.

## 9

## Approximation of Functions in 1D

Problem of **function approximation**:

Given: function  $\mathbf{f} : D \subset \mathbb{R}^n \mapsto \mathbb{R}^d$  (often in procedural form  $y = \text{f eval}(x)$ )

Goal: Find a “simple”<sup>(\*)</sup> function  $\tilde{\mathbf{f}} : D \mapsto \mathbb{R}^d$  such that the difference  $\mathbf{f} - \tilde{\mathbf{f}}$  is “small”<sup>(♣)</sup>

<sup>(\*)</sup>: “simple”  $\sim$  described by small amount of information, easy to evaluate (e.g, polynomial or piece-wise polynomial  $\tilde{\mathbf{f}}$ )

<sup>(♣)</sup> “small”  $\sim$   $\|\mathbf{f} - \tilde{\mathbf{f}}\|$  small for some norm  $\|\cdot\|$  on space  $C(D)$  of continuous functions, e.g.,

- $L^2$ -norm  $\|\mathbf{g}\|_2^2 := \|\mathbf{g}\|_{L^2(D)}^2 = \int_D |\mathbf{g}(x)|^2 dx$ , see (3.5.3),
- supremum norm  $\|\mathbf{g}\|_\infty := \|\mathbf{g}\|_{L^\infty(D)} := \max_{x \in D} |\mathbf{g}(x)|$ , see (3.5.2)

Below we only consider the case  $n = d = 1$ : approximation of scalar valued functions defined on an interval. The techniques are applied componentwise in order to cope with  $d > 1$ .

*Example 9.0.1* (Taylor approximation). → [63, Sect. 5.5]

For  $f \in C^k(I)$ ,  $k \in \mathbb{N}$ ,  $I \subset \mathbb{R}$  an interval,

$$f(t) \approx \underbrace{\sum_{j=0}^k \frac{f^{(j)}(t_0)}{j!} (t - t_0)^j}_{=: T_k(t)}, \quad t_0 \in I.$$


The Taylor polynomial  $T_k$  of degree  $k$  approximates  $f$  in a neighbourhood  $J \subset I$  of  $t_0$  ( $J$  can be small!). The Taylor approximation is easy and direct but inefficient: a polynomial of lower degree often gives the same accuracy.



Another technique:

Approximation by interpolation


$$\mathbf{f} \xrightarrow{\text{sampling}} (t_i, \mathbf{y}_i := f(t_i))_{i=1}^m \xrightarrow{\text{interpolation}} \tilde{\mathbf{f}}: \tilde{\mathbf{f}}(t_i) = \mathbf{y}_i .$$


  
 free choice of nodes  $t_i$

This is a new freedom compared to data interpolation ( $\rightarrow$  Ch. 3): we can choose the interpolation nodes in a way to obtain an accurate interpolant  $\tilde{\mathbf{f}}$ .

*Remark 9.0.2* (Interpolation and approximation: enabling technologies).

Approximation and interpolation ( $\rightarrow$  Ch. 3) are key components of many numerical methods, like for integration, differentiation and computation of the solutions of differential equations, as well as for computer graphics and generation of smooth curves and surfaces.

 this is a “foundations” part of the course



# 9.1 Error estimates for polynomial interpolation

Fcous: **approximation** of a function by global polynomial interpolation ( $\rightarrow$  Sect. 3.3)

*Remark 9.1.1* (Approximation by polynomials).

? Is it always possible to approximate a continuous function by polynomials?

✓ Yes! Recall the **Weierstrass theorem**:  
A continuous function  $f$  on the interval  $[a, b] \subset \mathbb{R}$  can be uniformly approximated by polynomials.

! But not by the interpolation on a fixed mesh [51, pag. 331]:

Given a sequence of meshes of increasing size  $\{\mathcal{T}_j\}_{j=1}^{\infty}$ ,  $\mathcal{T}_j = \{x_1^{(j)}, \dots, x_j^{(j)}\} \subset [a, b]$ ,  
 $a \leq x_1^{(j)} < x_2^{(j)} < \dots < x_j^{(j)} \leq b$ , there exists a continuous function  $f$  such that  
the sequence interpolating polynomials of  $f$  on  $\mathcal{T}_j$  does not converge uniformly to  $f$  as  
 $j \rightarrow \infty$ .

We consider Lagrangian polynomial interpolation on node set

$$\mathcal{T} := \{t_0, \dots, t_n\} \subset I, I \subset \mathbb{R}, \text{ interval of length } |I|.$$

Notation: For a continuous function  $f : I \mapsto \mathbb{K}$  we define the polynomial interpolation operator, see Thm. 3.3.7

$$I_{\mathcal{T}}(f) := I_{\mathcal{T}}(\mathbf{y}) \in \mathcal{P}_n \quad \text{with} \quad \mathbf{y} := (f(t_0), \dots, f(t_n))^T \in \mathbb{K}^{n+1}.$$

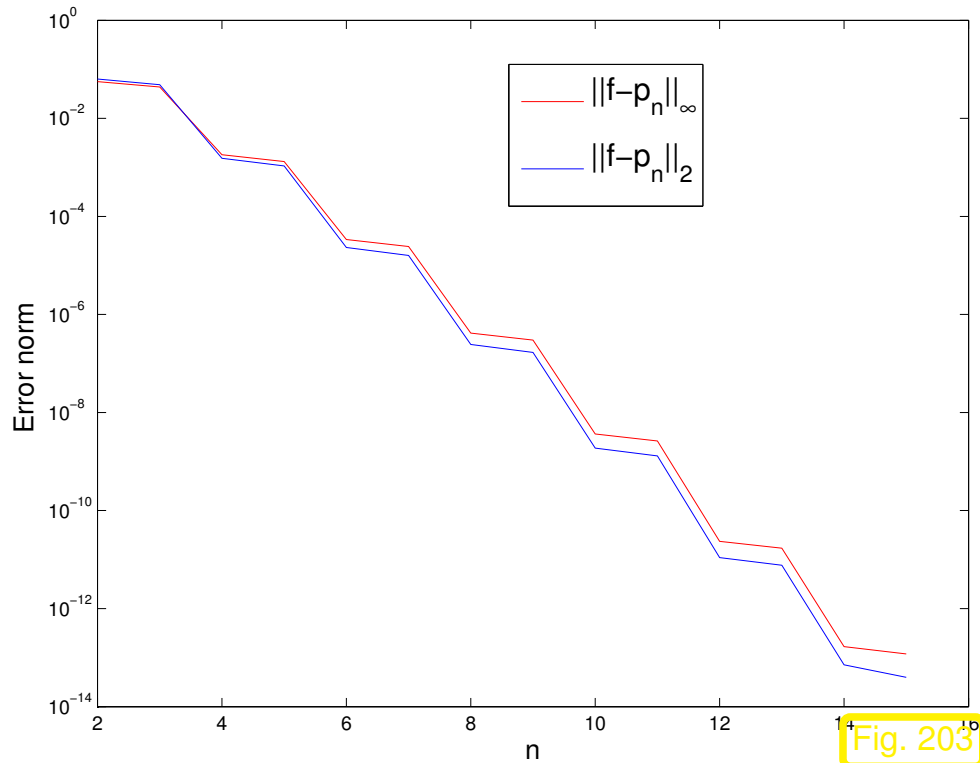
Goal: estimate of the **interpolation error** norm  $\|f - I_{\mathcal{T}}f\|$  (for some norm on  $C(I)$ ).

Focus: **asymptotic** behavior of interpolation error for  $n \rightarrow \infty$

**Example 9.1.2** (Asymptotic behavior of polynomial interpolation error).

Interpolation of  $f(t) = \sin t$  on equispaced nodes in  $I = [0, \pi]$ :  $\mathcal{T} = \{j\pi/n\}_{j=0}^n$ .

Interpolating polynomial  $p := I_{\mathcal{T}}f \in \mathcal{P}_n$ .



MATLAB-experiment: computation of the norms.

- $L^{\infty}$ -norm: sampling on a grid of meshsize  $\pi/1000$ .
- $L^2$ -norm: numerical quadrature ( $\rightarrow$  Chapter 10) with trapezoidal rule (10.3.3) on a grid of meshsize  $\pi/1000$ .



In the previous experiment we observed a clearly visible qualitative behavior of  $\|f - I_{\mathcal{T}}f\|$  as we increased the polynomial degree  $n$ . The prediction of the decay law for  $\|f - I_{\mathcal{T}}f\|$  is one goal in the study of interpolation errors.

Often this goal can be achieved, even if a rigorous quantitative bound for a norm of the interpolation error remains elusive.

Important terminology for the qualitative description of  $\|f - I_{\mathcal{T}}f\|$  as a function of the polynomial degree  $n$ :

$$\exists C \neq C(n) > 0: \|f - I_{\mathcal{T}}f\| \leq CT(n) \quad \text{for } n \rightarrow \infty. \quad (9.1.3)$$

Classification (best bound for  $T(n)$ ):

$$\begin{aligned} \exists p > 0: \quad T(n) \leq n^{-p} & : \text{ algebraic convergence, with rate } p > 0, \quad \forall n \in \mathbb{N}. \\ \exists 0 < q < 1: \quad T(n) \leq q^n & : \text{ exponential convergence,} \end{aligned}$$



Convergence behavior of interpolation error is often expressed by means of the Landau-O-notation:

$$\begin{array}{ll} \text{Algebraic convergence:} & \|f - I_{\mathcal{T}}f\| = O(n^{-p}) \\ \text{Exponential convergence:} & \|f - I_{\mathcal{T}}f\| = O(q^n) \end{array} \quad \text{for } n \rightarrow \infty \text{ ("asymptotic!")}$$

*Remark 9.1.4* (Exploring convergence).

Given: pairs  $(n_i, \epsilon_i)$ ,  $i = 1, 2, 3, \dots$ ,  $n_i \hat{=}$  polynomial degrees,  $\epsilon_i \hat{=}$  norms of interpolation error

❶ Conjectured: **algebraic convergence**:  $\epsilon_i \approx Cn^{-p}$


$$\log(\epsilon_i) \approx \log(C) - p \log n_i \quad (\text{affine linear in log-log scale}).$$

Apply linear regression ( MATLAB `polyfit`) to points  $(\log n_i, \log \epsilon_i) \triangleright$  estimate for rate  $p$ .

❶ Conjectured: **exponential convergence**:  $\epsilon_i \approx C \exp(-\beta n_i)$

$$\log \epsilon_i \approx \log(C) - \beta n_i \quad (\text{affine linear in lin-log scale}).$$

Apply linear regression (Ex. 7.0.9, MATLAB `polyfit`, Ex. 7.0.9) to points  $(n_i, \log \epsilon_i)$   $\triangleright$  estimate for  $q := \exp(-\beta)$ .

 Fig. 203: we suspect exponential convergence in Ex. 9.1.2.

Beware: same concept  $\leftrightarrow$  different meanings:

- **convergence** of a sequence (e.g. of iterates  $\mathbf{x}^{(k)} \rightarrow$  Sect. 4.1 )
- **convergence** of an approximation (dependent on an approximation parameter, e.g.  $n$ )

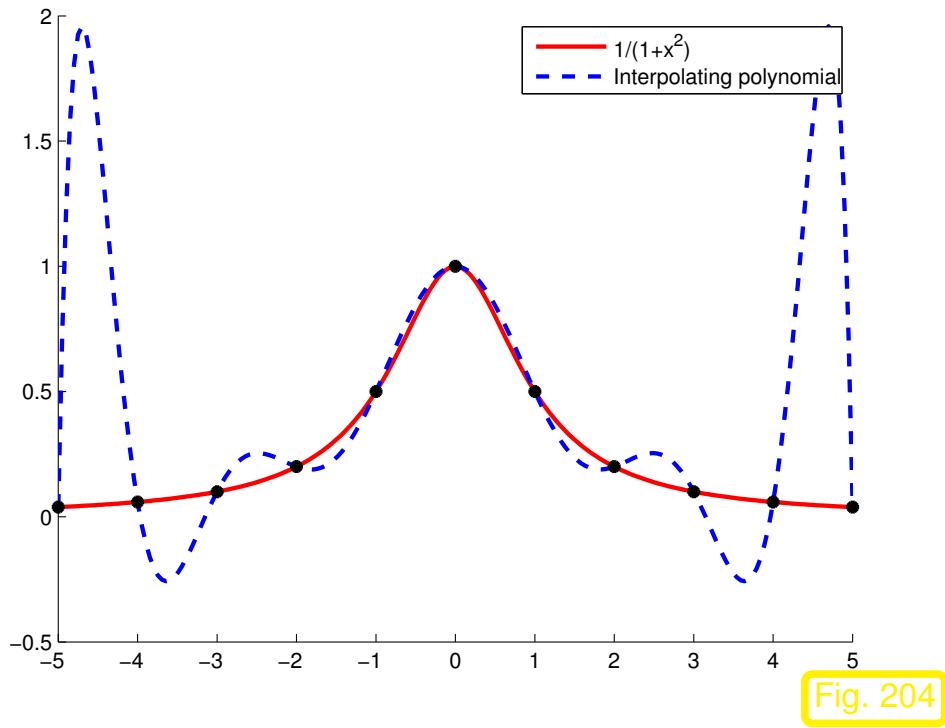
*Example 9.1.5* (Runge's example).  $\rightarrow$  Ex. 3.5.1

Polynomial interpolation of  $f(t) = \frac{1}{1+t^2}$  with equispaced nodes:

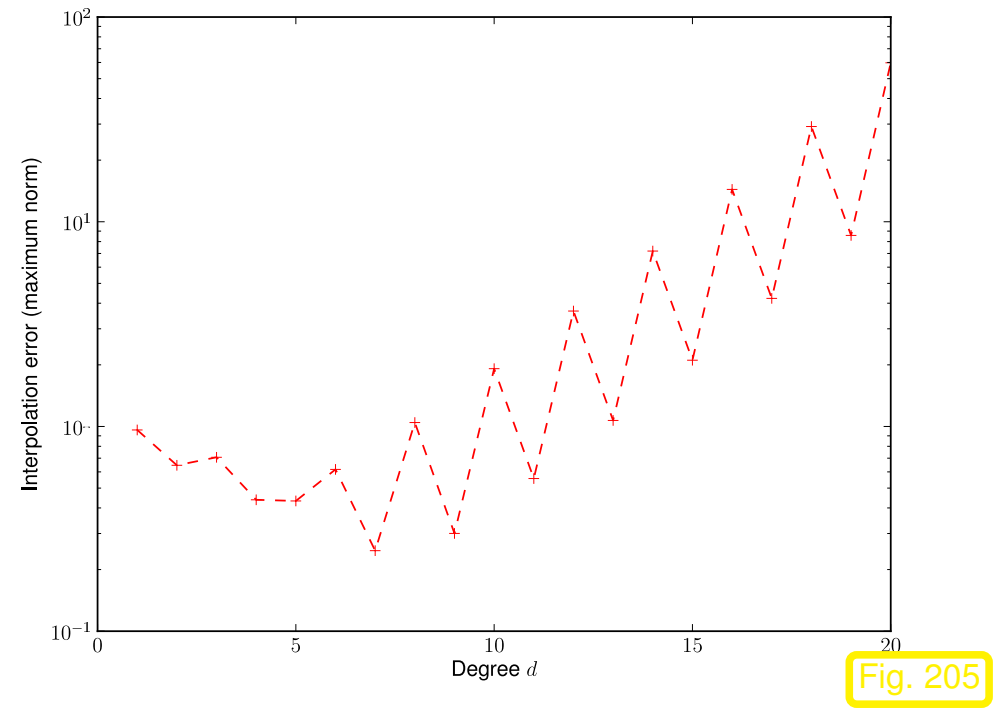
$$\mathcal{T} := \left\{ t_j := -5 + \frac{10}{n} j \right\}_{j=0}^n, \quad y_j = \frac{1}{1+t_j^2}, j = 0, \dots, n.$$

## Code 9.1.6: Computing the interpolation error for Runge's example

```
1 % interpolation error plot for Runge's example
2 % ("Quick and dirty" MATLAB implementation, see 3.4)
3 f = @(x) (1./(1+x.^2));
4 x = -5:0.01:5; % sampling point for approximate maximum norm
5 fv = f(x);
6 err = [];
7 for d=1:20
8     t = -5+(0:d)*10/d;
9     p = polyfit(t, f(t), d);
10    y = polyval(p, x);
11    err = [err; d, max(abs(y-fv))];
12 end
13
14 figure; semilogy(err(:,1), err(:,2), 'r--+');
15 xlabel('\bf degree d', 'fontsize', 14);
16 ylabel('\bf interpolation error (maximum norm)', 'fontsize', 14);
17 print -depsc2 '../PICTURES/rungeerrmax.eps';
```



Interpolating polynomial,  $n = 10$



Approximate  $\|f - \mathcal{I}_{\mathcal{T}}f\|_{\infty}$  on  $[-5, 5]$

Note: approximation of  $\|f - \mathcal{I}_{\mathcal{T}}f\|_{\infty}$  by *sampling* in 1000 equidistant points.

Observation: Strong oscillations of  $\mathcal{I}_{\mathcal{T}}f$  near the endpoints of the interval:

$$\|f - \mathcal{I}_{\mathcal{T}}f\|_{L^{\infty}(-5,5]} \xrightarrow{n \rightarrow \infty} \infty .$$

**Theorem 9.1.7** (Representation of interpolation error). [13, Thm. 8.22], [35, Thm. 37.4]

$f \in C^{n+1}(I): \forall t \in I: \exists \tau_t \in ]\min\{t, t_0, \dots, t_n\}, \max\{t, t_0, \dots, t_n\}[:$

$$f(t) - l_{\mathcal{T}}(f)(t) = \frac{f^{(n+1)}(\tau_t)}{(n+1)!} \cdot \prod_{j=0}^n (t - t_j). \quad (9.1.8)$$

*Proof.* Write  $q(t) := \prod_{j=0}^n (t - t_j) \in \mathcal{P}_{n+1}$  and fix  $t \in I$ .

$$t \neq t_j \Rightarrow q(t) \neq 0 \Rightarrow \exists c(t) \in \mathbb{R}: f(t) - l_{\mathcal{T}}(f)(t) = cq(t)$$

$\varphi(x) := f(x) - l_{\mathcal{T}}(f)(x) - cq(x)$  has  $n+2$  *distinct* zeros  $t_0, \dots, t_n, t$ . By iterated application of the **mean value theorem** [63, Thm .5.2.1], we conclude

$\varphi^{(m)}$  has  $n+2-m$  distinct zeros in  $I$ .

$$\Rightarrow \exists \tau_t \in I: \varphi^{(n+1)}(x) = f^{(n+1)}(\tau_t) - c(n+1)! = 0.$$

This fixes the value of  $c = \frac{f^{(n+1)}(\tau_t)}{(n+1)!}$ . □

*Remark 9.1.9* (Explicit representation of error of polynomial interpolation).

The theorem can also be proved using the following lemma.

*Lemma 9.1.10* (Error of the polynomial interpolation). For  $f \in C^{n+1}(I)$ :  $\forall t \in I$ :

$$f(t) - l_{\mathcal{T}}(f)(t) = \int_0^1 \int_0^{\tau_1} \cdots \int_0^{\tau_{n-1}} \int_0^{\tau_n} f^{(n+1)}(t_0 + \tau_1(t_1 - t_0) + \cdots + \tau_n(t_n - t_{n-1}) + \tau(t - t_n)) d\tau d\tau_n \cdots d\tau_1 \cdot \prod_{j=0}^n (t - t_j).$$

*Proof.* By induction on  $n$ , use (3.4.3) and the fundamental theorem of calculus [53, Sect. 3.1]: △



*Remark 9.1.11.* Lemma 9.1.10 holds also for general polynomial interpolation with multiple nodes, see (3.3.11).

In the *equation* (9.1.8) we can

- first bound the right hand side via  $f^{(n+1)}(\tau_t) \leq \left\| f^{(n+1)} \right\|_{L^\infty(I)}$ ,
- then increase the right hand side further by switching to the maximum (in modulus) w.r.t.  $t$  (the resulting bound does no longer depend on  $t!$ ),
- and, finally, take the maximum w.r.t.  $t$  on the left of  $\leq$ .

This yields the following **interpolation error estimate**:

$$\text{Thm. 9.1.7} \quad \Rightarrow \quad \|f - I_{\mathcal{T}}f\|_{L^\infty(I)} \leq \frac{\left\| f^{(n+1)} \right\|_{L^\infty(I)}}{(n+1)!} \max_{t \in I} |(t - t_0) \cdots (t - t_n)|. \quad (9.1.12)$$

## Interpolation error estimate requires smoothness!

*Example 9.1.13* (Error of polynomial interpolation). Ex. 9.1.2 cnt'd

Theoretical explanation for exponential convergence observed for polynomial interpolation of  $f(t) = \sin(t)$  on equidistant nodes: by Thm. 9.1.10 and (9.1.12)

$$\left\| f^{(k)} \right\|_{L^\infty(I)} \leq 1, \quad \forall k \in \mathbb{N}_0 \quad \Rightarrow \quad \begin{aligned} \|f - p\|_{L^\infty(I)} &\leq \frac{1}{(1+n)!} \max_{t \in I} \left| (t-0) \left(t - \frac{\pi}{n}\right) \left(t - \frac{2\pi}{n}\right) \cdots (t - \pi) \right| \\ &\leq \frac{1}{n+1} \left(\frac{\pi}{n}\right)^{n+1}. \end{aligned}$$

→ **Uniform asymptotic exponential convergence** of the interpolation polynomials (independently of the set of nodes  $\mathcal{T}$ . In fact,  $\|f - p\|_{L^\infty(I)}$  decays even faster than exponential!)





Example 9.1.14 (Runge's example). Ex. 9.1.5 cnt'd

How can the blow-up of the interpolation error observed in Ex. 9.1.5 be reconciled with Thm. 9.1.10 ?

Here  $f(t) = \frac{1}{1+t^2}$  allows only to conclude  $|f^{(n)}(t)| = 2^n n! \cdot O(|t|^{-2-n})$  for  $n \rightarrow \infty$ .

→ Possible blow-up of error bound from Thm. 9.1.7  $\rightarrow \infty$  for  $n \rightarrow \infty$ .



Remark 9.1.15 ( $L^2$ -error estimates for polynomial interpolation).

Thm. 9.1.7 gives error estimates for the  $L^\infty$ -Norm. And the other norms?

From Lemma. 9.1.10 using Cauchy-Schwarz inequality:

$$\|f - I_{\mathcal{T}}(f)\|_{L^2(I)}^2 = \int_I \left| \int_0^1 \int_0^{\tau_1} \cdots \int_0^{\tau_{n-1}} \int_0^{\tau_n} f^{(n+1)}(\dots) d\tau_n \cdots d\tau_1 \cdot \underbrace{\prod_{j=0}^n (t - t_j)}_{|t-t_i| \leq |I|} \right|^2 dt$$

$$\begin{aligned}
 &\leq \int_I |I|^{2n+2} \underbrace{\text{vol}_{(n+1)}(S_{n+1})}_{=1/(n+1)!} \int_{S_{n+1}} |f^{(n+1)}(\dots)|^2 d\boldsymbol{\tau} dt \\
 &= \int_I \frac{|I|^{2n+2}}{(n+1)!} \int_I \underbrace{\text{vol}_{(n)}(C_{t,\tau})}_{\leq 2^{(n-1)/2}/n!} |f^{(n+1)}(\tau)|^2 d\tau dt,
 \end{aligned}$$

$S_{n+1} := \{\mathbf{x} \in \mathbb{R}^{n+1} : 0 \leq x_n \leq x_{n-1} \leq \dots \leq x_1 \leq 1\}$  (unit simplex),

$C_{t,\tau} := \{\mathbf{x} \in S_{n+1} : t_0 + x_1(t_1 - t_0) + \dots + x_n(t_n - t_{n-1}) + x_{n+1}(t - t_n) = \tau\}$ .

This gives the bound for the  $L^2$ -norm of the error:

$$\Rightarrow \|f - \mathcal{I}_{\mathcal{T}}(f)\|_{L^2(I)} \leq \frac{2^{(n-1)/4} |I|^{n+1}}{\sqrt{(n+1)!n!}} \left( \int_I |f^{(n+1)}(\tau)|^2 d\tau \right)^{1/2}. \quad (9.1.16)$$

Notice:  $f \mapsto \|f^{(n)}\|_{L^2(I)}$  defines a **seminorm** on  $C^{n+1}(I)$   
 (**Sobolev-seminorm**, measure of the smoothness of a function).

Estimates like (9.1.16) play a key role in the analysis of numerical methods for solving partial differential equations ( $\rightarrow$  course “Numerical methods for partial differential equations”).

Perspective: function **approximation** by polynomial interpolation



Freedom to choose interpolation nodes judiciously

## 9.2.1 Motivation and definition

Setting: Mesh of nodes:  $\mathcal{T} := \{t_0 < t_1 < \dots < t_{n-1} < t_n\}$ ,  $n \in \mathbb{N}$ ,  
function  $f : I \rightarrow \mathbb{R}$  continuous; without loss of generality  $I = [-1, 1]$ .

Thm. 9.1.7:

$$\|f - p\|_{L^\infty(I)} \leq \frac{1}{(n+1)!} \|f^{(n+1)}\|_{L^\infty(I)} \|w\|_{L^\infty(I)},$$
$$w(t) := (t - t_0) \cdot \dots \cdot (t - t_n).$$

Idea: choose nodes  $t_0, \dots, t_n$  such that  $\|w\|_{L^\infty(I)}$  is minimal!

Equivalent to finding  $q \in \mathcal{P}_{n+1}$ , with

- leading coefficient = 1,
- such that  $\|q\|_{L^\infty(I)}$  is minimal.

► Choice of  $t_0, \dots, t_n =$  zeros of  $q$  (caution:  $t_j$  must belong to  $I$ ).

- Heuristic:
- $t^*$  extremal point of  $q \rightarrow |q(t^*)| = \|q\|_{L^\infty(I)}$ ,
  - $q$  has  $n + 1$  zeros in  $I$ ,
  - $|q(-1)| = |q(1)| = \|q\|_{L^\infty(I)}$ .

**Definition 9.2.1** (Chebychev polynomial).  $\rightarrow [35, \text{Ch. } 32]$

The  $n^{\text{th}}$  *Chebychev polynomial* is  $T_n(t) := \cos(n \arccos t)$ ,  $-1 \leq t \leq 1$ .

Remark 9.2.2 (3-term recursion for Chebychev polynomial). → [35, (32.2)]

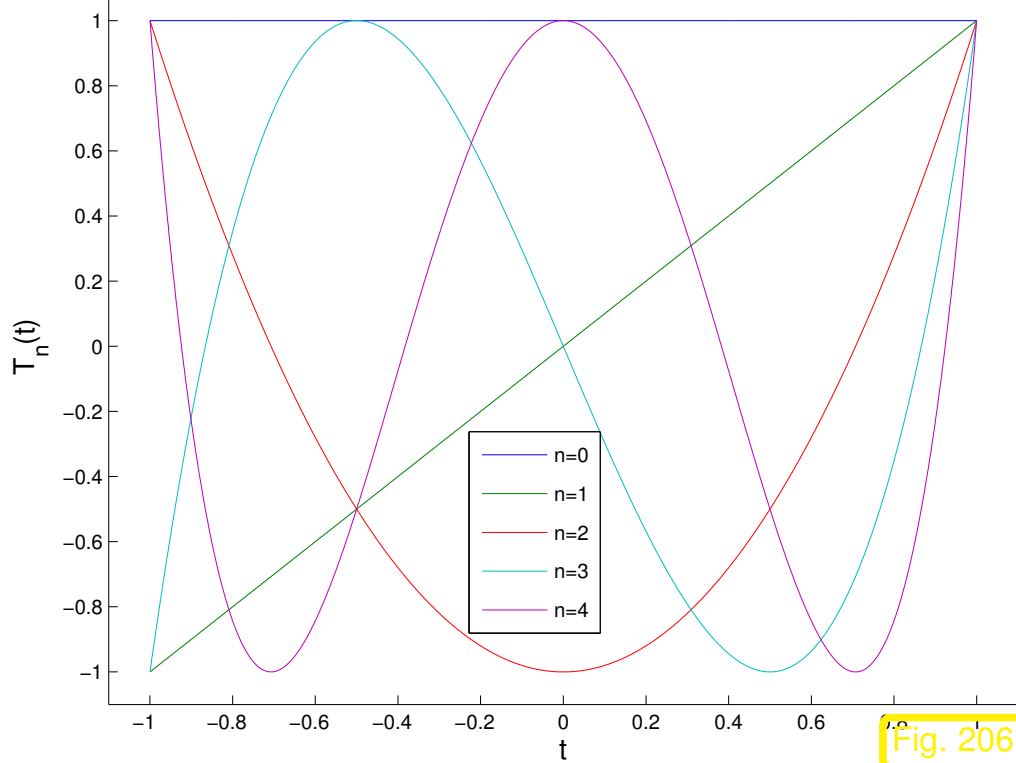
**3-term recursion** by  $\cos(n+1)x = 2\cos nx \cos x - \cos(n-1)x$  with  $\cos x = t$ :

$$T_{n+1}(t) = 2tT_n(t) - T_{n-1}(t) \quad , \quad T_0 \equiv 1 \quad , \quad T_1(t) = t \quad , \quad n \in \mathbb{N} . \quad (9.2.3)$$

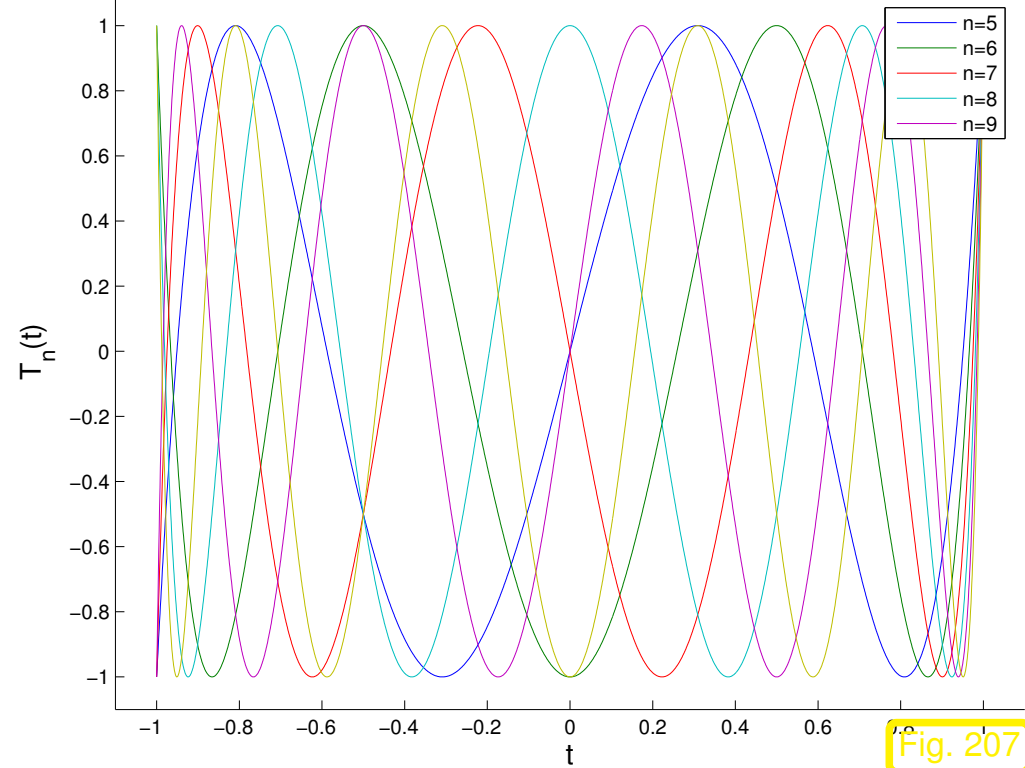
- This implies:
- $T_n \in \mathcal{P}_n$ ,
  - leading coefficients equal to  $2^{n-1}$ ,
  - $T_n$  linearly independent,
  - $T_n$  basis of  $\mathcal{P}_n = \text{Span} \{T_0, \dots, T_n\}$ ,  $n \in \mathbb{N}_0$ .

See Code 9.2.3 for algorithmic use of 3-term recursion (9.2.3).





Chebyshev polynomials  $T_0, \dots, T_4$



Chebyshev polynomials  $T_5, \dots, T_9$

Code 9.2.4: Efficient evaluation of Chebyshev polynomials up to a certain degree

```

1 function V = chebpolmult(d,x)
2 % Computes the values of the Chebyshev polynomials  $T_0, \dots, T_d$ ,  $d \geq 2$ 
3 % at points passed in x using the 3-term recursion (9.2.3).
4 % The values  $T_k(x_j)$ , are returned in row  $k+1$  of V.
5 V = ones(1, length(x)); %  $T_0 \equiv 1$ 
6 V = [V; x]; %  $T_1(x) = x$ 

```

```

7 for k=1:d-1
8     V = [V; 2*x.*V(end, :) - V(end-1, :)]; % 3-term recursion
9 end

```

Code 9.2.5: Plotting Chebychev polynomials, see Fig. 206, 207

```

1 function chebpolplot(nmax)
2 % plots Chebychev polybomials up to degree nmax on [-1,1]
3 N = 1000; x = -1:2/N:1;
4 V = chebpolmult(nmax,x); % compute values of Chebychev polynomials
5 for j=0:nmax, leg{j} = sprintf('n=%i', j); end
6 plot(x,V', '-');
7 legend(leg);
8 xlabel('{\bf t}', 'FontSize', 14);
9 ylabel('{\bf T_n(t)}', 'FontSize', 14);
0 axis([-1.1 1.1 -1.1 1.1]);

```

Extrema (with value  $\pm 1$ , alternating signs) of  $T_n$ :

$$|T_n(\bar{t}_k)| = 1 \Leftrightarrow \exists k = 0, \dots, n: \bar{t}_k = \cos \frac{k\pi}{n}, \quad \|T_n\|_{L^\infty([-1,1])} = 1. \quad (9.2.6)$$

**Theorem 9.2.7** (Minimax property of the Chebychev polynomials). [17, Section 7.1.4.], [35, Thm. 32.2]

$$\|T_n\|_{L^\infty([-1,1])} = \inf\{\|p\|_{L^\infty([-1,1])} : p \in \mathcal{P}_n, p(t) = 2^{n-1}t^n + \dots\}, \quad \forall n \in \mathbb{N}.$$

*Proof.* (indirect) Assume

$$\exists q \in \mathcal{P}_n, \text{ leading coefficient} = 2^{n-1}: \quad \|q\|_{L^\infty([-1,1])} < \|T_n\|_{L^\infty([-1,1])}.$$

▶  $(T_n - q)(x) > 0$  in local maxima of  $T_n$

$(T_n - q)(x) < 0$  in local minima of  $T_n$

From knowledge of local extrema of  $T_n$ , see (9.2.6):

$T_n - q$  changes sign at least  $n + 1$  times

$\Rightarrow T_n - q$  has at least  $n$  zeros

$T_n - q \equiv 0$ , because  $T_n - q \in \mathcal{P}_{n-1}$  (same leading coefficient!)

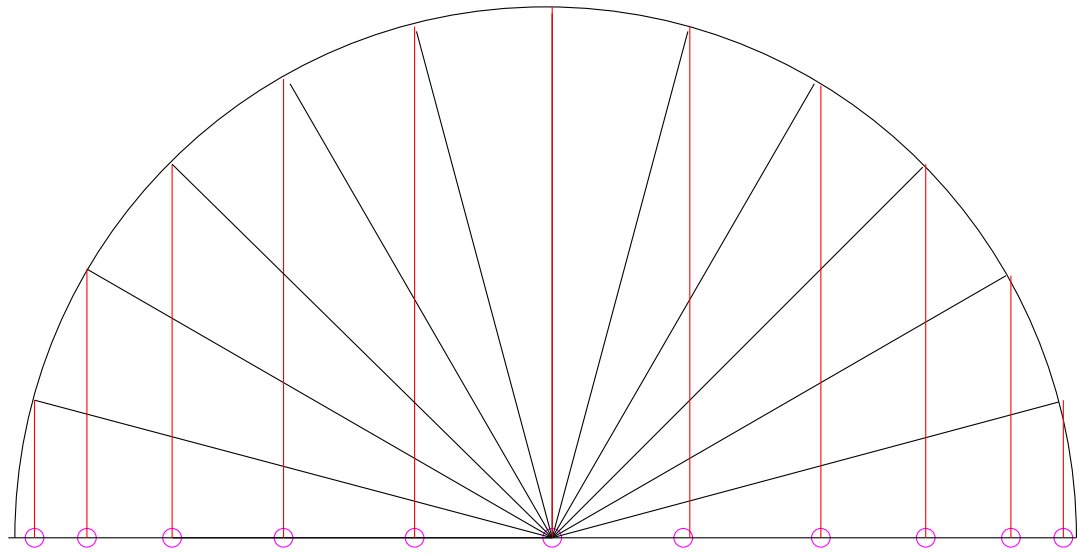
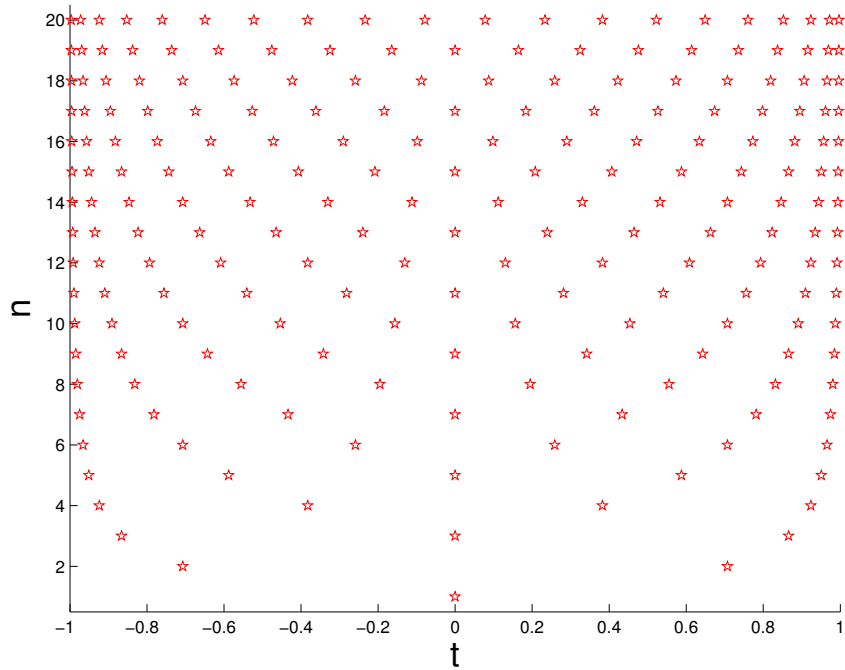


$$\text{Zeros of } T_n: \quad t_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \dots, n. \quad (9.2.8)$$

To see this, notice

$$\begin{aligned} T_n(t) = 0 & \stackrel{\text{zeros of } \cos}{\Leftrightarrow} n \arccos t \in (2\mathbb{Z} + 1)\frac{\pi}{2} \\ & \stackrel{\arccos \in [0, \pi]}{\Leftrightarrow} t \in \left\{ \cos\left(\frac{2k+1}{n}\frac{\pi}{2}\right), k = 0, \dots, n-1 \right\}. \end{aligned}$$

Location of **Chebyshev nodes**  $t_k$  from (9.2.8):



Use of Chebychev nodes for polynomial interpolation: **Chebychev interpolation**

- For  $I = [-1, 1]$
- “optimal” interpolation nodes  $\mathcal{T} = \left\{ \cos \left( \frac{2k+1}{2(n+1)} \pi \right) , k = 0, \dots, n \right\}$ ,
  - $w(t) = (t - t_0) \cdots (t - t_{n+1}) = 2^{-n} T_{n+1}(t)$  ,  $\|w\|_{L^\infty(I)} = 2^{-n}$ ,
- with leading coefficient 1.

Then, by Thm. 9.1.7,

$$\|f - \mathcal{I}_{\mathcal{T}}(f)\|_{L^\infty([-1,1])} \leq \frac{2^{-n}}{(n+1)!} \|f^{(n+1)}\|_{L^\infty([-1,1])}. \quad (9.2.9)$$

*Remark 9.2.10* (Chebychev polynomials on arbitrary interval).

How to use Chebychev polynomial interpolation on an arbitrary interval?

**Scaling argument:** interval transformation requires the transport of the functions

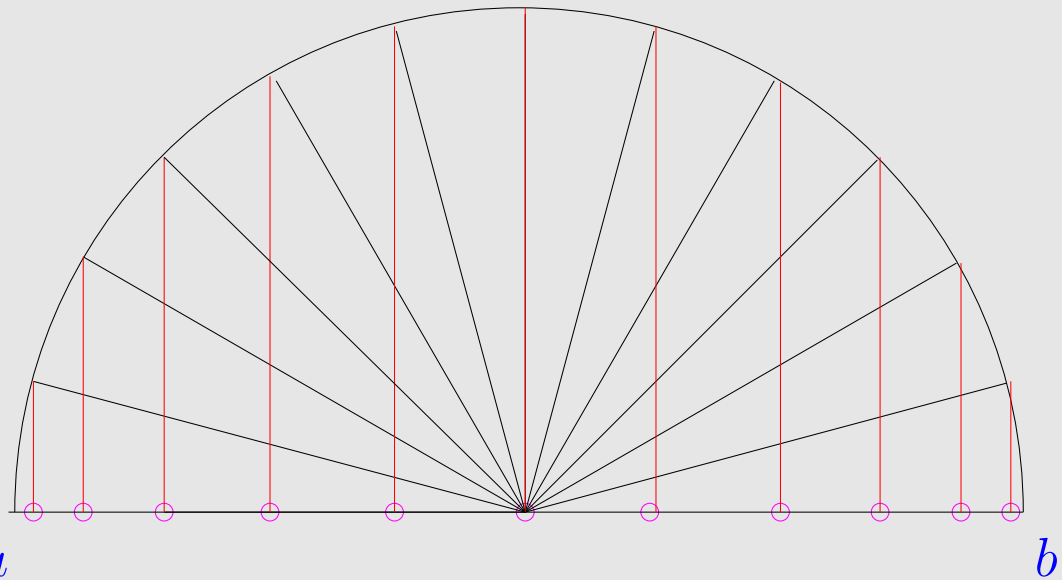
$$[-1, 1] \xrightarrow{\hat{t} \mapsto t := a + \frac{1}{2}(\hat{t} + 1)(b - a)} [a, b] \quad \leftrightarrow \quad \hat{f}(\hat{t}) := f(t).$$

$$p \in \mathcal{P}_n \quad \wedge \quad p(t_j) = f(t_j) \quad \Leftrightarrow \quad \hat{p} \in \mathcal{P}_n \quad \wedge \quad \hat{p}(\hat{t}_j) = \hat{f}(\hat{t}_j).$$

With transformation formula for the integrals &  $\frac{d^n \hat{f}}{d\hat{t}^n}(\hat{t}) = \left(\frac{1}{2}|I|\right)^n \frac{d^n f}{dt^n}(t)$ :

$$\|f - \mathcal{I}_{\mathcal{T}}(f)\|_{L^\infty(I)} = \|\hat{f} - \mathcal{I}_{\hat{\mathcal{T}}}(\hat{f})\|_{L^\infty([-1,1])} \leq \frac{2^{-n}}{(n+1)!} \left\| \frac{d^{n+1} \hat{f}}{d\hat{t}^{n+1}} \right\|_{L^\infty([-1,1])}$$

$$\leq \frac{2^{-2n-1}}{(n+1)!} |I|^{n+1} \left\| f^{(n+1)} \right\|_{L^\infty(I)} . \tag{9.2.11}$$



The **Chebyshev nodes** in the interval  $I = [a, b]$  are

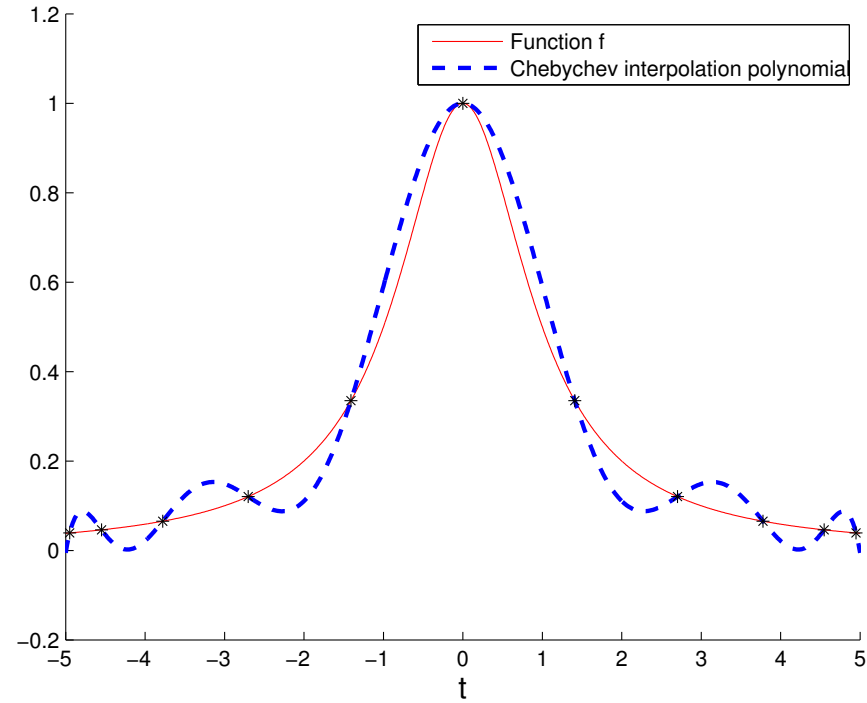
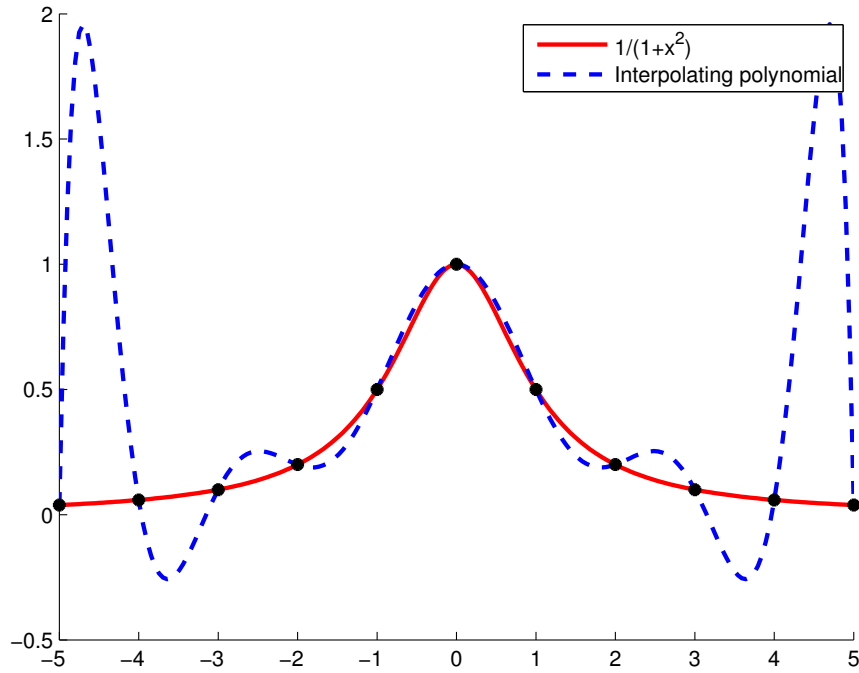
$$t_k := a + \frac{1}{2}(b - a) \left( \cos\left(\frac{2k + 1}{2(n + 1)} \pi\right) + 1 \right) , \tag{9.2.12}$$

$k = 0, \dots, n .$

## 9.2.2 Chebyshev interpolation error estimates

*Example 9.2.13* (Polynomial interpolation: Chebyshev nodes versus equidistant nodes).

Runge's function  $f(t) = \frac{1}{1+t^2}$ , see Ex. 9.1.5, polynomial interpolation based on uniformly spaced nodes and Chebychev nodes:



*Remark 9.2.14* (Lebesgue Constant for Chebychev nodes).

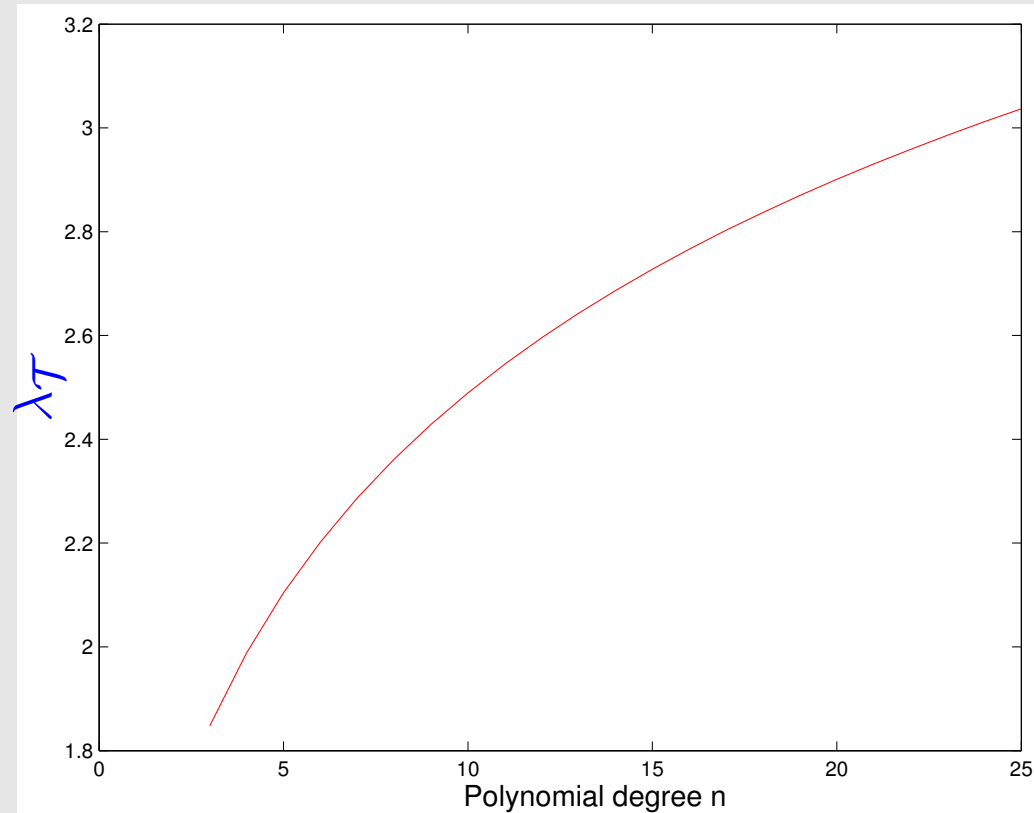
→ Sect. 3.5

Theory [10, 67, 66]:

$$\lambda_{\mathcal{T}} \sim \frac{2}{\pi} \log(1+n) + o(1),$$

$$\lambda_{\mathcal{T}} \leq \frac{2}{\pi} \log(1+n) + 1. \quad (9.2.15)$$

Compare with Lebesgue constant for equidistant nodes, see Rem. 3.5.8.



*Example 9.2.16* (Chebychev interpolation error).

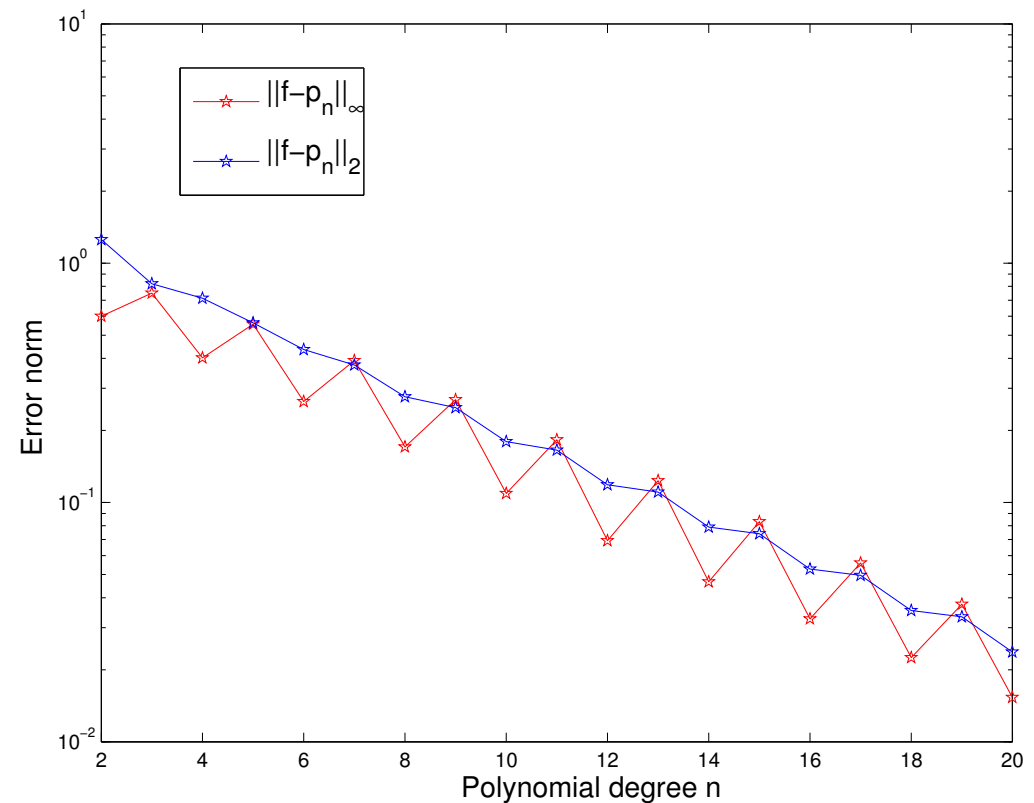
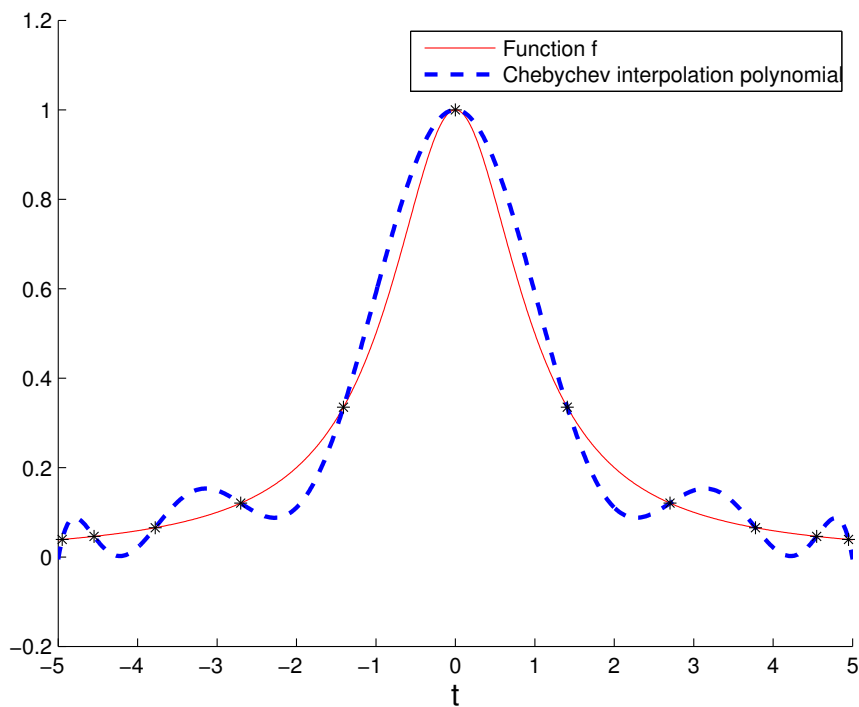
For  $I = [a, b]$  let  $x_l := a + \frac{b-a}{N}l$ ,  $l = 0, \dots, N$ ,  $N = 1000$  we approximate the norms of the error

$$\|f - p\|_\infty \approx \max_{0 \leq l \leq N} |f(x_l) - p(x_l)| \quad (9.2.17)$$

$$\|f - p\|_2^2 \approx \frac{b-a}{2N} \sum_{0 \leq l < N} \left( |f(x_l) - p(x_l)|^2 + |f(x_{l+1}) - p(x_{l+1})|^2 \right) \quad (9.2.18)$$

①  $f(t) = (1 + t^2)^{-1}$ ,  $I = [-5, 5]$  (see Ex. 9.1.5)

Interpolation with  $n = 10$  Chebychev nodes (plot on the left).

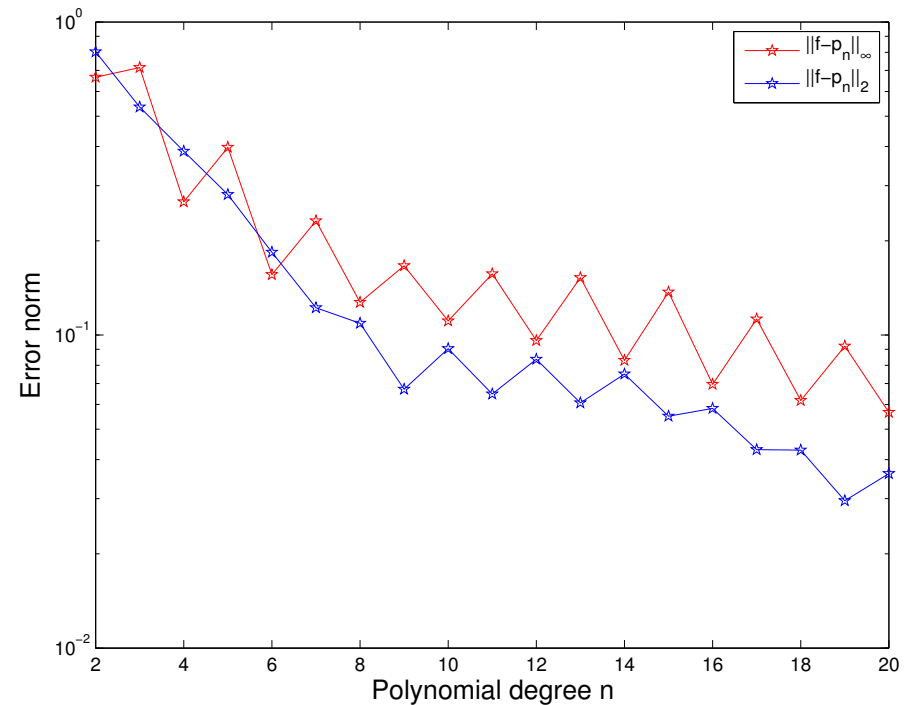
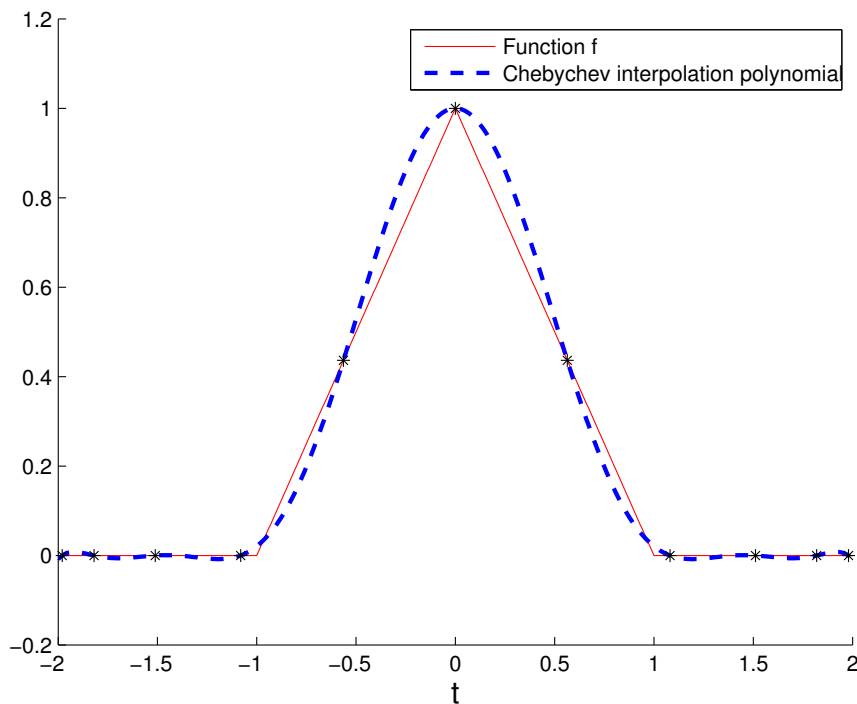


Notice: exponential convergence of the Chebychev interpolation:

$$p_n \rightarrow f, \quad \|f - I_n f\|_{L^\infty([-5,5])} \approx 0.8^n$$

②  $f(t) = \max\{1 - |t|, 0\}$ ,  $I = [-2, 2]$ ,  $n = 10$  nodes (plot on the left).

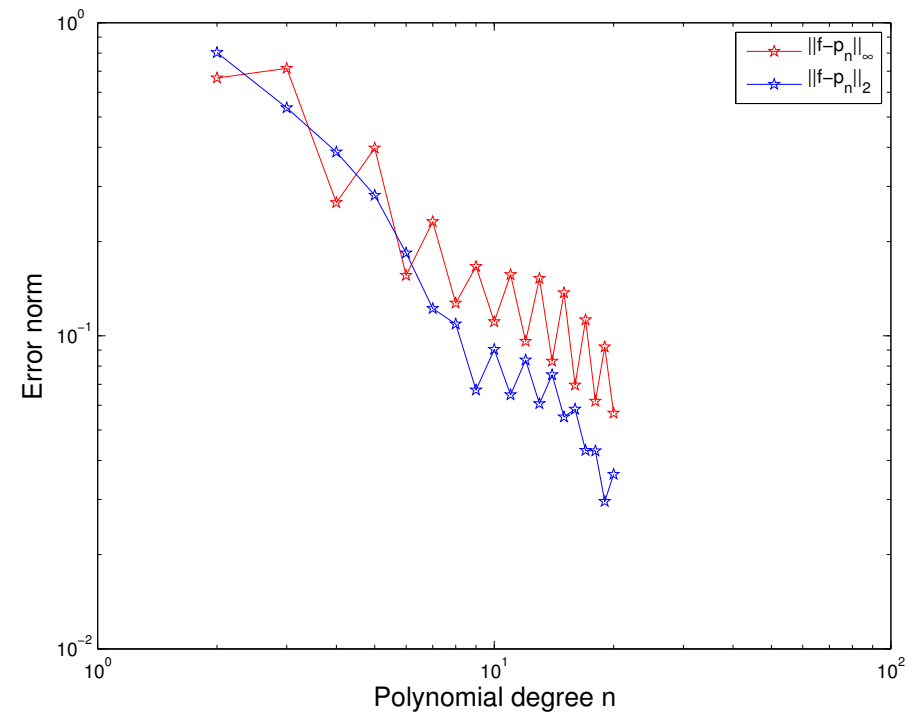
Now  $f \in C^0(I)$  but  $f \notin C^1(I)$ .



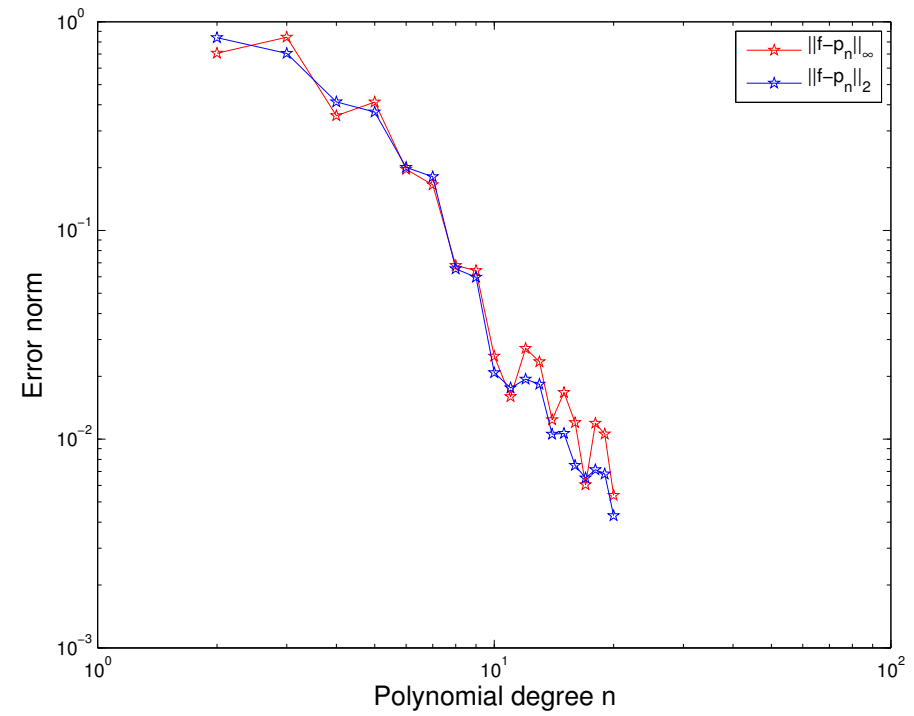
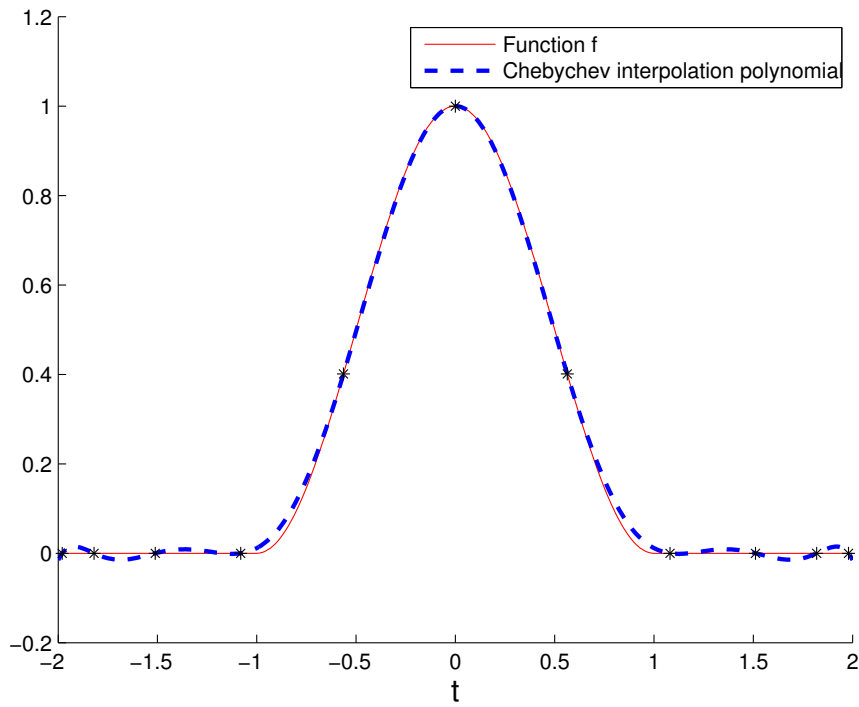


From the doubly logarithmic plot we conclude →

- no exponential convergence
- algebraic convergence (?)



③  $f(t) = \begin{cases} \frac{1}{2}(1 + \cos \pi t) & |t| < 1 \\ 0 & 1 \leq |t| \leq 2 \end{cases} \quad I = [-2, 2], \quad n = 10 \quad (\text{plot on the left}).$



Notice: only (vaguely) algebraic convergence.



Summary of observations, *cf.* Rem. 9.1.4:

- Essential role of smoothness of  $f$ : slow convergence of approximation error of the Cheychev interpolant if  $f$  enjoys little smoothness, *cf.* also (9.1.12),
- for smooth  $f \in C^\infty$  approximation error of the Cheychev interpolant seems to decay to zero exponentially in the polynomial degree  $n$ .

*Remark 9.2.19* (Chebychev interpolation of analytic functions).  $\rightarrow$  complex analysis

Change of perspective:

Consider the interpoland  $f : I \subset \mathbb{R} \mapsto \mathbb{R}$  as the restriction to  $I$  of a complex valued function  $f : D \subset \mathbb{C} \mapsto \mathbb{C}$ , where  $D$  is an open neighborhood of  $I$  in  $\mathbb{C}$ .

Example:  $f(t) = \frac{1}{1+x^2}$  on  $I \subset \mathbb{R}$

$$\updownarrow$$

$$f : \mathbb{C} \setminus \{+i, -i\} \mapsto \mathbb{C} \text{ with } f(z) = \frac{1}{1+z^2}$$

*Definition 9.2.20 (Analytic function).*

A function  $f : D \subset \mathbb{C} \mapsto \mathbb{C}$  is called **analytic** ( $\Leftrightarrow$  holomorphic) (in  $D$ ), if (in the sense of uniform convergence)

$$\forall z \in D: \exists r > 0, (\alpha_k)_{k=0}^{\infty} \in \mathbb{C}^{\mathbb{N}_0}: |w - z| < r \Rightarrow f(w) = \sum_{k=0}^{\infty} \alpha_k (w - z)^k .$$

Analyticity  $\Leftrightarrow$  locally representable by convergent power series

Examples for functions that are analytic in  $\mathbb{C}$ :  $\exp, \sin, \cos, \sinh, \cosh$

Rational functions ( $\hat{=}$  quotients of polynomials) are analytic everywhere away from their poles ( $\hat{=}$  zeros of denominator).

**Result.** Chebychev interpolants of  $f : I \mapsto \mathbb{C}$ ,  $I \subset \mathbb{R}$  interval, converge exponentially in the degree, if  $f$  is analytic in an open  $\mathbb{C}$ -neighborhood of  $I$ . The rate of exponential convergence increases with the size of this neighborhood relative to the length of  $I$ .

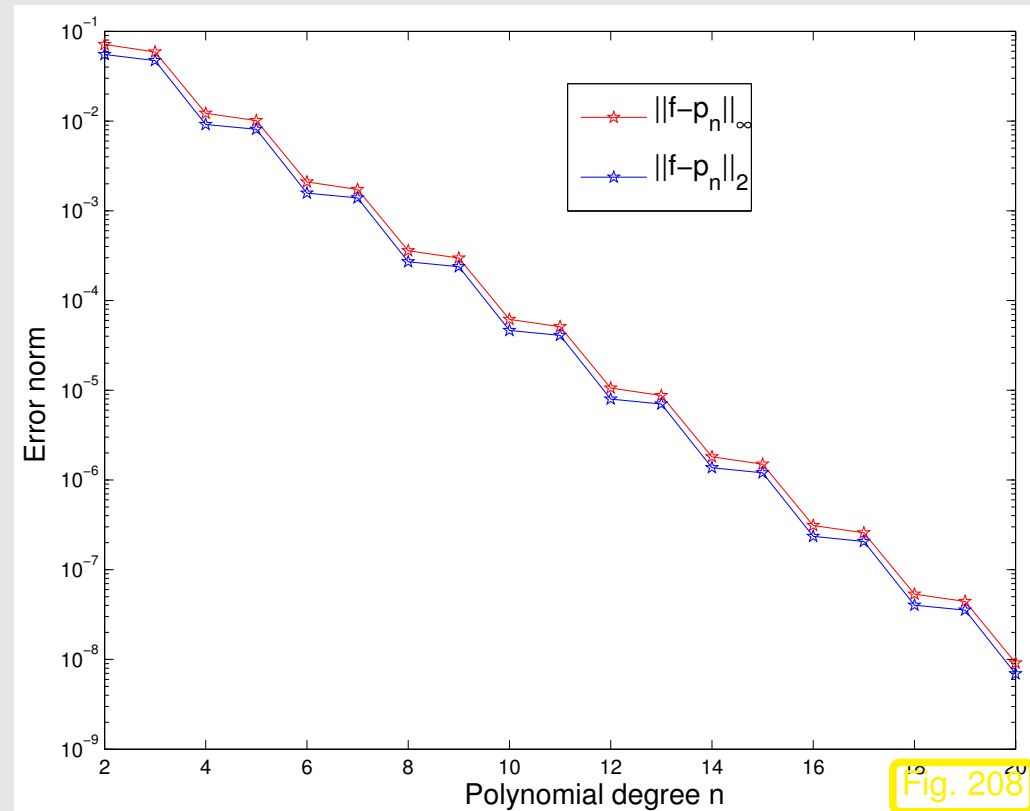


Example 9.2.21 (Chebychev interpolation of analytic function). → Ex. 9.2.16 cnt'd

Modification: the same function  $f(t) = (1 + t^2)^{-1}$  on a smaller interval  $I = [-1, 1]$ .

(Faster) exponential convergence than on  $I = ] -5, 5[$ :

$$\|f - I_n f\|_{L^2([-1,1])} \approx 0.42^n.$$



Explanation, cf. Rem. 9.2.19: for  $I = [-1, 1]$  the poles  $\pm i$  of  $f$  are farther away relative to the size of the interval than for  $I = [-5, 5]$ .



### 9.2.3 Chebychev interpolation: computational aspects

Task: Given: given degree  $n \in \mathbb{N}$ , continuous function  $f : [-1, 1] \mapsto \mathbb{R}$

Sought: **efficient** representation/evaluation of interpolating polynomial  $p \in \mathcal{P}_n$  in Chebychev nodes (9.2.12) on  $[-1, 1]$

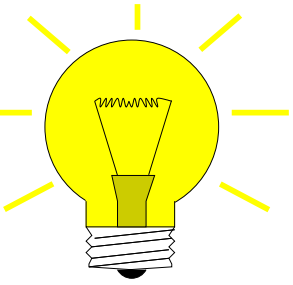
```

1 class ChebInterp {
2   private :
3     // various internal data describing Chebychev interpolating polynomial p
4   public :
5     // Constructor taking function f and degree n as arguments
6     PolyInterp(const Function &f, unsigned int n);
7     // Evaluation operator:  $y_j = p(x_j)$ ,  $j = 1, \dots, m$  ( $m$  "large")
8     double eval(const vector<double> &x, vector <double> &y) const;
9 };

```

Trick: expand  $p$  into Chebychev polynomials

$$p = \sum_{j=0}^n \alpha_j T_j, \quad \alpha_j \in \mathbb{R}. \quad (9.2.22)$$



The representation (9.2.22) is always possible, because  $\{T_0, \dots, T_n\}$  is a *basis* of  $\mathcal{P}_n$

*Remark* 9.2.23 (Fast evaluation of Chebychev expansion).  $\rightarrow$  [35, Alg. 32.1]

Use the 3-term recurrence (9.2.3)

$$T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x), \quad j = 2, 3, \dots,$$

to rewrite (9.2.22) as

$$p(x) = \sum_{j=0}^{n-1} \tilde{\alpha}_j T_j \quad \text{with} \quad \tilde{\alpha}_j = \begin{cases} \alpha_j + 2x\alpha_{j+1} & , \text{ if } j = n - 1, \\ \alpha_j - \alpha_{j+2} & , \text{ if } j = n - 2, \\ \alpha_j & \text{ else.} \end{cases} \quad (9.2.24)$$

▶ recursive algorithm, see Code 9.2.24.

## Code 9.2.25: Recursive evaluation of Chebychev expansion (9.2.22)

```

1 function y = recclenshaw(a,x)
2 % Recursive evaluation of a polynomial  $p = \sum_{j=1}^{n+1} a_j T_{j-1}$  at point x, see (9.2.24)
3 % The coefficients  $a_j$  have to be passed in a row vector.
4 n = length(a)-1;
5 if (n<2), y = a(1)+x*a(2);
6 else
7     y = recclenshaw([a(1:n-2), a(n-1)-a(n+1), a(n)+2*x*a(n+1)],x);
8 end

```

Non-recursive version: **Clenshaw algorithm**

## Code 9.2.26: Clenshaw algorithm for evaluation of Chebychev expansion (9.2.22)

```

1 function y = clenshaw(a,x)
2 % Clenshaw algorithm for evaluating  $p = \sum_{j=1}^{n+1} a_j T_{j-1}$  at points passed in vector x
3 n = length(a)-1; % degree of polynomial
4 d = repmat(reshape(a,n+1,1),1,length(x));
5 for j=n:-1:2
6     d(j,:) = d(j,:) + 2*x.*d(j+1,:); % see (9.2.24)
7     d(j-1,:) = d(j-1,:) - d(j+1,:);
8 end
9 y = d(1,:) + x.*d(2,:);

```





Computational effort :  $O(nm)$  for evaluation at  $m$  points



Issue: How to compute the Chebychev expansion coefficients  $\alpha_j$  in (9.2.22) efficiently from the interpolation conditions

$$p(t_k) = f(t_k), \quad k = 0, \dots, n, \quad \text{for } t_k := \cos \left( \frac{2k+1}{2(n+1)} \pi \right). \quad (9.2.27)$$

 Chebychev nodes

Trick: transformation of  $p$  into a 1-periodic function:

$$\begin{aligned}
 q(s) &:= p(\cos 2\pi s) = \sum_{j=0}^n \alpha_j T_j(\cos 2\pi s) \exp(-2\pi i n s) \stackrel{\text{Def. 9.2.1}}{=} \sum_{j=0}^n \alpha_j \cos(2\pi j s) \\
 &= \sum_{j=0}^n \frac{1}{2} \alpha_j (\exp(2\pi i j s) + \exp(-2\pi i j s)) \\
 &= \sum_{j=-n}^{n+1} \beta_j \exp(-2\pi i j s), \quad \beta_j := \begin{cases} 0 & , \text{ for } j = n + 1, \\ \frac{1}{2} \alpha_j & , \text{ for } j = 1, \dots, n, \\ \alpha_0 & , \text{ for } j = 0, \\ \frac{1}{2} \alpha_{n-j} & , \text{ for } j = -n, \dots, -1. \end{cases}
 \end{aligned} \tag{9.2.28}$$

Transformed interpolation conditions (9.2.27):

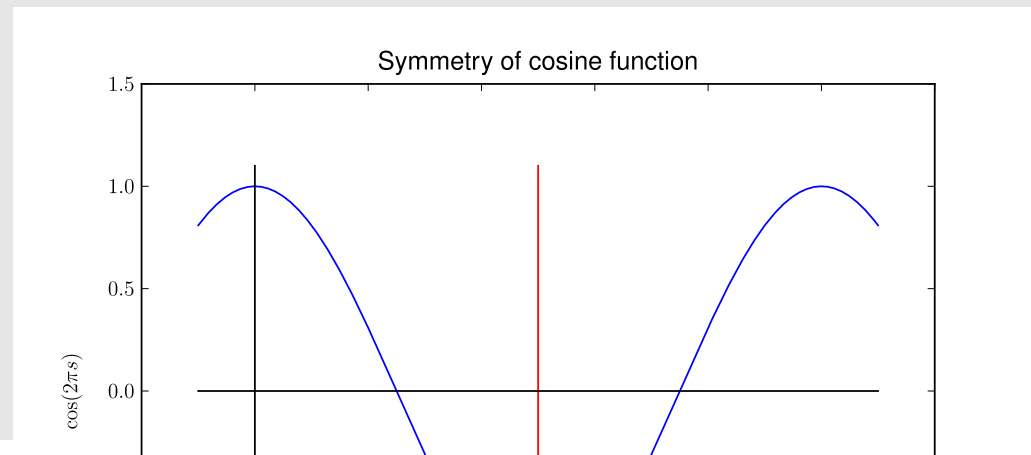
$$t = \cos(2\pi s) \stackrel{(9.2.27)}{\implies} q\left(\frac{2k+1}{4(n+1)}\right) = f(t_k), \quad k = 0, \dots, n. \tag{9.2.29}$$

Observe **symmetry**

$$q(s) = q(1-s)$$

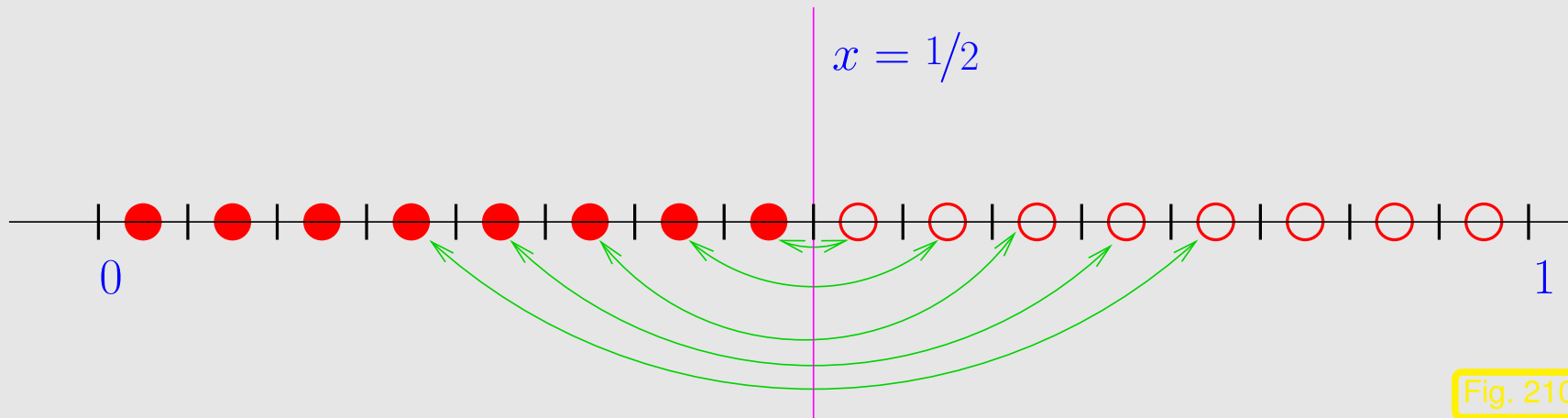
$$\Downarrow \leftarrow (9.2.29)$$

$$\frac{2k+1}{4(n+1)}$$



Extend interpolation conditions (9.2.29) by symmetry, see Fig. 210

$$q\left(\frac{k}{2(n+1)} + \frac{1}{4(n+1)}\right) = z_k := \begin{cases} y_k & , \text{ for } k = 0, \dots, n, \\ y_{2n+1-k} & , \text{ for } k = n+1, \dots, 2n+1. \end{cases} \quad (9.2.30)$$



$\iff$  linear system of equations (at last):

$$q\left(\frac{k}{2(n+1)} + \frac{1}{4(n+1)}\right) = \sum_{j=-n}^{n+1} \left( \beta_j \exp\left(-\frac{2\pi i j}{4(n+1)}\right) \right) \exp\left(-\frac{2\pi i}{2n+1} k j\right) = z_k .$$

$\iff$

$$\sum_{j=0}^{2n+1} \left( \beta_j \exp\left(-\frac{2\pi i(j-n)}{4(n+1)}\right) \right) \underbrace{\exp\left(-\frac{2\pi i}{2n+1} k j\right)}_{\omega_{2(n+1)}^{kj}} = \exp\left(-\pi i \frac{nk}{n+1}\right) z_k , \quad k = 0, \dots, 2n+1 .$$

► solve (9.2.31) with inverse discrete Fourier transform, see 8.2:

asymptotic complexity  $O(n \log n)$  ( $\rightarrow$  Sect. 8.3)

Note: by symmetry of  $\mathbf{z}$   $\Rightarrow \beta_{2n+1} = 0$ , cf. (9.2.28)!

Code 9.2.32: Efficient computation of Chebychev expansion coefficient of Chebychev interpolant

```

1 function a = chebexp(y)
2 % Efficiently compute coefficients  $\alpha_j$  in the Chebychev expansion
3 %  $p = \sum_{j=0}^n \alpha_j T_j$  of  $p \in \mathcal{P}_n$  based on values  $y_k$ ,
4 %  $k = 0, \dots, n$ , in Chebychev nodes  $t_k$ ,  $k = 0, \dots, n$ . These values are
5 % passed in the row vector y.
6 n = length(y) - 1; % degree of polynomial
7 % create vector  $\mathbf{z}$  by wrapping and componentwise scaling
8 z = exp(-pi*i*n/(n+1)*(0:2*n+1)).*[y,y(end:-1:1)]; % r.h.s. vector
9 c = ifft(z); % Solve linear system (9.2.31) with effort  $O(n \log n)$ 
10 b = real(exp(0.5*pi*i/(n+1)*(-n:n+1)).*c); % recover  $\beta_j$ , see (9.2.31)
11 a = [b(n+1), 2*b(n+2:2*n+1)]; % recover  $\alpha_j$ , see (9.2.28)

```

*Remark 9.2.33* (Chebychev representation of built-in functions).

Computers use approximation by sums of Chebychev polynomials in the computation of functions like  $\log$ ,  $\exp$ ,  $\sin$ ,  $\cos$ ,  $\dots$ . The evaluation by means of Clenshaw algorithm according to Code 9.2.25 is more efficient and stable than for approximation by Taylor polynomials.

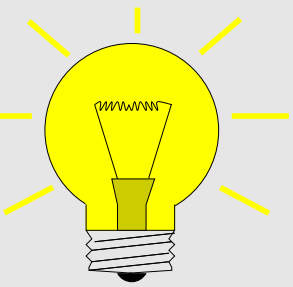


## 9.3 Trigonometric interpolation [13, Sect. 8.5]

Task: approximation of continuous **1-periodic** function

$$f \in C^0(\mathbb{R}) \quad , \quad f(t+1) = f(t) \quad \forall t \in \mathbb{R} .$$

Dubious: approximation by global polynomials, because they are not 1-periodic  $\triangleright$  global polynomial interpolant will lack essential structural property.



Idea: approximation by interpolation based on spaces of periodic functions

Natural candidates: **trigonometric functions** (sines and cosines)

*Trigonometric interpolation* [13, pp. 304]

Given  $t_0 < t_1 < \dots < t_{2n}$ ,  $t_k \in [0, 1[$ , and  $y_k \in \mathbb{R}$ ,  $k = 0, \dots, 2n$  find

$$q \in \mathcal{P}_{2n}^T := \text{Span} \{t \mapsto \cos(2\pi jt), t \mapsto \sin(2\pi jt)\}_{j=0}^n, \quad (9.3.1)$$

$$\text{with } q(t_k) = y_k \text{ for all } k = 0, \dots, 2n. \quad (9.3.2)$$

Terminology:  $\mathcal{P}_{2n}^T \hat{=}$  space of **trigonometric polynomials** of degree  $2n$ .

Trigonometric interpolation = linear interpolation mapping ( $\rightarrow$  Rem. 3.1.8) into function space  $\mathcal{P}_{2n}^T$

- $\dim \mathcal{P}_{2n}^T = 2n + 1$ , because functions are linearly independent, sine dropped for  $j = 0$
- functions in (9.3.1) provide basis functions for interpolation, cf. (3.1.4)

Coefficient representation of  $q \in \mathcal{P}_{2n}^T$

$$q(t) = \alpha_0 + \sum_{j=1}^n \alpha_j \cos(2\pi jt) + \beta_j \sin(2\pi jt), \quad \alpha_j, \beta_j \in \mathbb{R}. \quad (9.3.3)$$

Why is  $\mathcal{P}_{2n}^T$  called a space of trigonometric *polynomials* ?

Recall expressions for trigonometric functions via **complex** exponentials:

$$e^{it} = \cos t + i \sin t \quad \Rightarrow \quad \begin{cases} \cos t = \frac{1}{2}(e^{it} + e^{-it}) \\ \sin t = \frac{1}{2i}(e^{it} - e^{-it}) \end{cases}. \quad (9.3.4)$$

► representation of trigonometric polynomials by means of complex exponentials:

$$\begin{aligned} q(t) &= \alpha_0 + \sum_{j=1}^n \alpha_j \cos(2\pi jt) + \beta_j \sin(2\pi jt) \\ &= \alpha_0 + \frac{1}{2} \left\{ \sum_{j=1}^n (\alpha_j - i\beta_j) e^{2\pi ijt} + (\alpha_j + i\beta_j) e^{-2\pi ijt} \right\} \end{aligned}$$

$$\begin{aligned}
 &= \alpha_0 + \frac{1}{2} \sum_{j=-n}^{-1} (\alpha_{-j} + i\beta_{-j})e^{2\pi ijt} + \frac{1}{2} \sum_{j=1}^n (\alpha_j - i\beta_j)e^{2\pi ijt} \\
 &= e^{-2\pi int} \sum_{j=0}^{2n} \gamma_j e^{2\pi ijt}, \quad \text{with } \gamma_j = \begin{cases} \frac{1}{2}(\alpha_{n-j} + i\beta_{n-j}) & \text{for } j = 0, \dots, n-1, \\ \alpha_0 & \text{for } j = n, \\ \frac{1}{2}(\alpha_{j-n} - i\beta_{j-n}) & \text{for } j = n+1, \dots, 2n. \end{cases} \quad (9.3.5)
 \end{aligned}$$

Note:  $\gamma_j \in \mathbb{C}$   $\triangleright$  work in  $\mathbb{C}$  in the context of trigonometric interpolation! Admit  $y_k \in \mathbb{C}$ .

$$q \in \mathcal{P}_{2n+1}^T \Rightarrow q(t) = e^{-2\pi int} \cdot p(e^{2\pi it}) \quad \text{with } p(z) = \sum_{j=0}^{2n} \gamma_j z^j \in \mathcal{P}_{2n}, \quad (9.3.6)$$

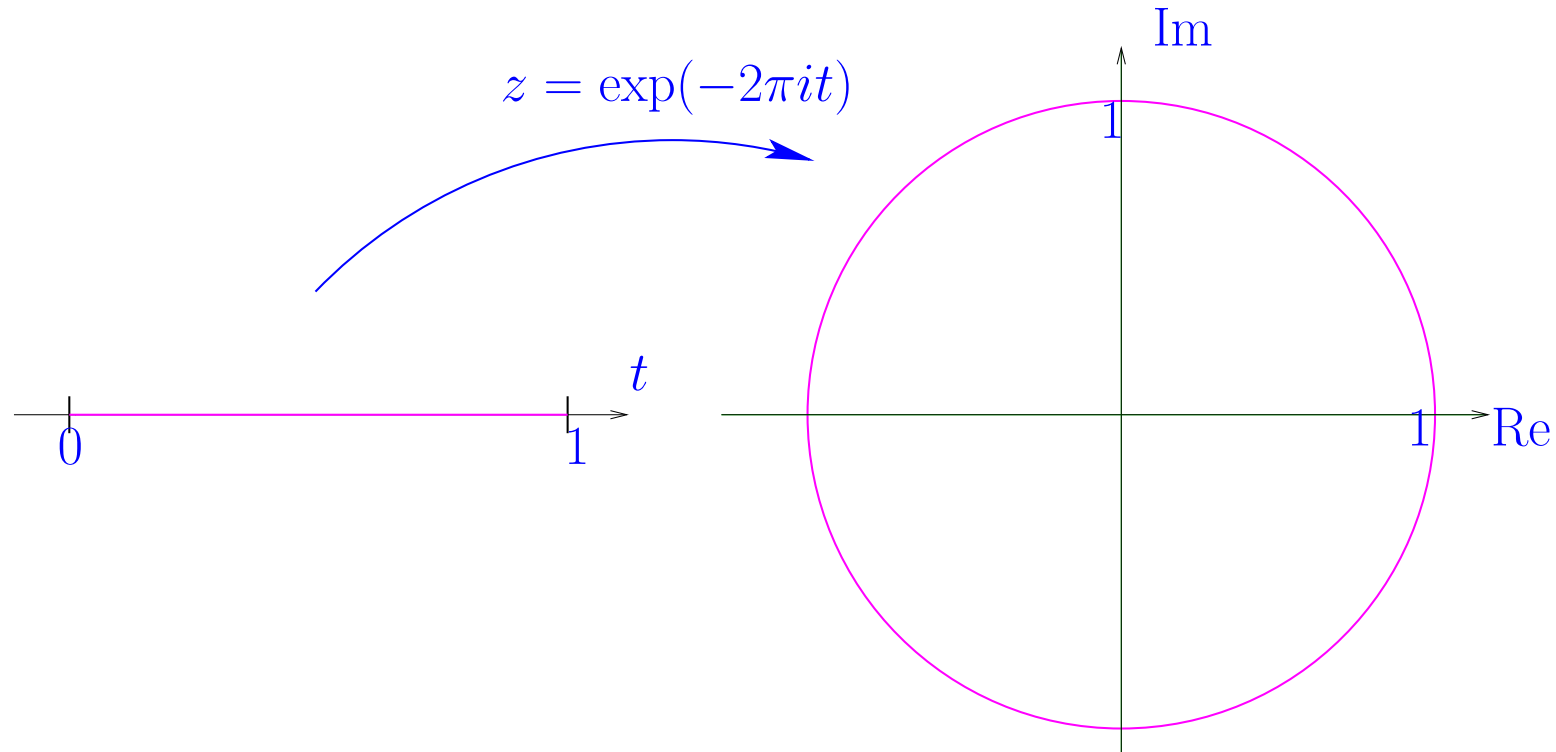
a polynomial !

and  $\gamma_j$  from (9.3.5).

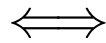
$t \rightarrow \exp(2\pi int)q(t)$  is a polynomial  $p \in \mathcal{P}_{2n}$  restricted to the unit circle  $\mathbb{S}^1$  in  $\mathbb{C}$ .



notation:  $\mathbb{S}^1 := \{z \in \mathbb{C} : |z| = 1\}$



Trigonometric interpolation  
through data points  $(t_k, y_k)$



Polynomial interpolation  
through data points  $(e^{2\pi i t_k}, y_k)$

Trigonometric interpolation = polynomial interpolation on  $\mathbb{S}^1$

► All theoretical results and algorithms from polynomial interpolation carry over to trigonometric interpolation

- Existence and uniqueness of trigonometric interpolation polynomial, see Thm. 3.3.5,
- Concept of Lagrange polynomials, see (3.3.2),
- the algorithms and representations discussed in Sect. 3.4.

Code 9.3.7: Evaluation of trigonometric interpolation polynomial in many points

```

1 function q = trigpolyval(t,y,x)
2 % Evaluation of trigonometric interpolant at numerous points
3 % t: row vector of nodes  $t_0, \dots, t_n \in [0, 1[$ 
4 % y: row vector of data  $y_0, \dots, y_n$ 
5 % x: row vector of evaluation points  $x_1, \dots, x_N$ 
6 N = length(y); if (mod(N,2)~=1), error(' #pts odd required'); end
7 n = (N-1)/2;
8 tc = exp(2*pi*i*t); % Interpolation nodes on unit circle
9 z = exp(2*pi*i*n*t).*y; % Rescaled values, according to  $q(t) = e^{-2\pi i n t} \cdot p(e^{2\pi i t})$ 
10 % Evaluation of interpolating polynomial on unit circle, see Code 3.4.1

```

```
11 p = intpolyval(tc, z, exp(2*pi*i*x));  
12 q = exp(-2*pi*i*n*x) .* p; % Undo the scaling, see (9.3.6)
```

## Back to **Approximation of 1-periodic functions by trigonometric interpolant**

Remember: additional freedom to choose interpolation nodes in the setting of function approximation

Reasonable choice for generic 1-periodic  $f$ : **uniformly distributed nodes**  $t_k = \frac{k}{2n+1}$ ,  $k = 0, \dots, 2n$

Justification: the circle is utterly “isotropic”!

►  $(2n+1) \times (2n+1)$  linear system of equations:

$$\sum_{j=0}^{2n} \gamma_j \exp\left(2\pi i \frac{jk}{2n+1}\right) = z_k := \exp\left(2\pi i \frac{nk}{2n+1}\right) y_k, \quad k = 0, \dots, 2n.$$

$$\Leftrightarrow$$

$$\bar{\mathbf{F}}_{2n+1} \mathbf{c} = \mathbf{z}, \quad \mathbf{c} = (\gamma_0, \dots, \gamma_{2n})^T \quad \xRightarrow{\text{Lemma 8.2.10}} \quad \mathbf{c} = \frac{1}{2n+1} \mathbf{F}_{2n} \mathbf{z}. \quad (9.3.8)$$

$(2n+1) \times (2n+1)$  (conjugate) **Fourier matrix**, see (8.2.9)

Fast solution by means of FFT:  $O(n \log n)$  asymptotic complexity, see Sect. 8.3

Code 9.3.10: Efficient computation of coefficient of trigonometric interpolation polynomial (*equidistant nodes*)

```

1 function [a,b] = trigipequid(y)
2 % Efficient computation of coefficients in expansion (9.3.3) for a trigonometric
3 % interpolation polynomial in equidistant points  $(\frac{j}{2n+1}, y_j)$ ,  $j = 0, \dots, 2n$ 
4 % y has to be a row vector of odd length, return values are column vectors
5 N = length(y); if (mod(N,2)~=1), error(' #pts odd!'); end;
6 n = (N-1)/2;
7 c = fft(exp(2*pi*i*(n/N)*(0:2*n)) .* y) / N; % see (9.3.8)
8 % From (9.3.5):  $\alpha_j = \frac{1}{2}(\gamma_{n-j} + \gamma_{n+j})$  and  $\beta_j = \frac{1}{2i}(\gamma_{n-j} - \gamma_{n+j})$ ,  $j = 1, \dots, n$ ,  $\alpha_0 = \gamma_n$ 

```

```

9 a = transpose([c(n+1), c(n:-1:1)+c(n+2:N)]);
10 b = transpose(-i*[c(n:-1:1)-c(n+2:N)]);

```

### Code 9.3.11: Computation of coefficients of trigonometric interpolation polynomial, general nodes

```

1 function [a,b] = trigpolycoeff(t,y)
2 % Computes expansion coefficients of trigonometric polynomials (9.3.3)
3 % t: row vector of nodes  $t_0, \dots, t_n \in [0, 1[$ 
4 % y: row vector of data  $y_0, \dots, y_n$ 
5 % return values are column vectors of expansion coefficients  $\alpha_j, \beta_j$ 
6 N = length(y); if (mod(N,2)~=1), error('#pts odd!'); end
7 n = (N-1)/2;
8 M = [cos(2*pi*t'*(0:n)), sin(2*pi*t'*(1:n))];
9 c = M\y';
10 a = c(1:n+1); b = c(n+2:end);

```

*Example 9.3.12* (Runtime comparison for computation of coefficient of trigonometric interpolation polynomials).

tic-toc-timings



MATLAB 7.10.0.499 (R2010a)

CPU: Intel Core i7, 2.66 GHz, 2 cores, L2 256 KB,  
L3 4 MB

OS: Mac OS X, Darwin Kernel Version 10.5.0

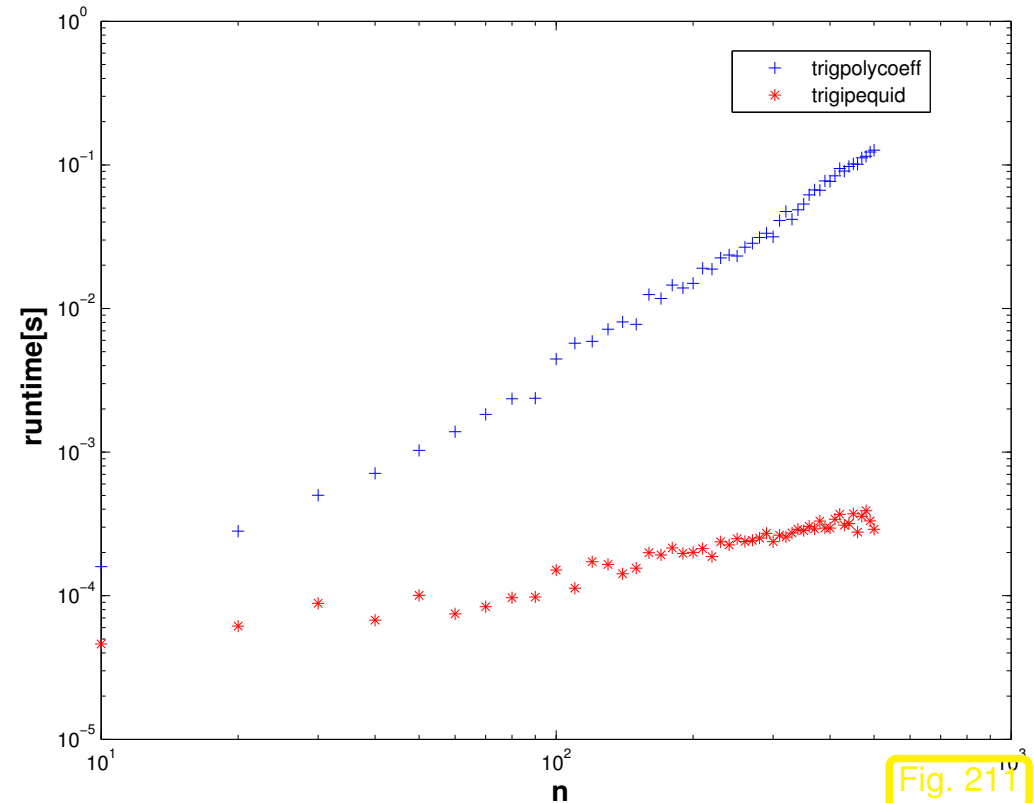


Fig. 211

## Code 9.3.13: Runtime comparison

```

1 function trigipequidtiming
2 % Runtime comparison between efficient (→ Code 9.3.8) and direct computation
3 % (→ Code 9.3.10 of coefficients of trigonoetric interpolation polynomial in
4 % equidistant points.
5 Nruns = 3; times = [];
6 for n = 10:10:500
7     disp(n)
8     N = 2*n+1; t = 0:1/N:(1-1/N); y = exp(cos(2*pi*t));
9     t1 = realmax; t2 = realmax;

```

```
0   for k=1:Nruns
1       tic; [a,b] = trigpolycoeff(t,y); t1 = min(t1,toc);
2       tic; [a,b] = trigipequid(y); t2 = min(t2,toc);
3   end
4   times = [times; n , t1 , t2];
5 end
6
7 figure; loglog(times(:,1),times(:,2),'b+',...
8               times(:,1),times(:,3),'r*');
9 xlabel('\bf n','fontsize',14);
10 ylabel('\bf runtime[s]','fontsize',14);
11 legend('trigpolycoeff','trigipequid','location','best');
12
13 print -depsc2 '../PICTURES/trigipequidtiming.eps';
```

Same observation as in Ex. 8.3.1: massive gain in efficiency through relying on FFT.



*Remark 9.3.14* (Efficient evaluation of trigonometric interpolation polynomials).

Task: evaluation of trigonometric polynomial (9.3.3) at *equidistant* points  $\frac{k}{N}$ ,  $N > 2n$ .  $k = 0, \dots, N - 1$ .

$$(9.3.5) \quad \blacktriangleright \quad q(k/N) = e^{-2\pi i k/N} \sum_{j=0}^{2n} \gamma_j \exp(2\pi i \frac{kj}{N}), \quad k = 0, \dots, N - 1.$$

$$\blacktriangleright \quad q(k/N) = e^{-2\pi i kn/N} v_j \quad \text{with} \quad \mathbf{v} = \bar{\mathbf{F}}_N \tilde{\mathbf{c}}, \quad (9.3.15)$$

Fourier matrix, see (8.2.9).

where  $\tilde{\mathbf{c}} \in \mathbb{C}^N$  is obtained by **zero padding** of  $\mathbf{c} := (\gamma_0, \dots, \gamma_{2n})^T$ :

$$(\tilde{\mathbf{c}})_k = \begin{cases} \gamma_j & , \text{ for } k = 0, \dots, 2n, \\ 0 & , \text{ for } k = 2n + 1, \dots, N - 1. \end{cases}$$

### Code 9.3.16: Fast evaluation of trigonometric polynomial at *equidistant* points

```

1 function q = trigipequidcomp(a,b,N)
2 % Efficient evaluation of trigonometric polynomial at equidistant points
3 % column vectors a and b pass coefficients  $\alpha_j, \beta_j$  in
4 % representation (9.3.3)
5 n = length(a)-1; if (N < (2*n-1)), error('N too small'); end;
6 gamma = transpose(0.5*[a(end:-1:2)+i*b(end:-1:1);...
7     2*a(1);a(2:end)-i*b(1:end)]);
8 ch = [gamma, zeros(1,N-(2*n+1))]; % zero padding

```



```

9 v = conj(fft(conj(ch))); % Multiplication with conjugate Fourier matrix
10 q = exp(-2*pi*i*n*(0:N-1)/N).*v; % undo rescaling

```

Code 9.3.17: *Equidistant* points: fast on the fly evaluation of trigonometric interpolation polynomial

```

1 function q = trigpolyvalequid(y,N)
2 % Evaluation of trigonometric interpolation polynomial through  $(\frac{j}{2n+1}, y_j)$ ,
   %  $j = 0, \dots, 2n$ 
3 % in equidistant points  $\frac{k}{N}$ ,  $k = 0, N-1$ 
4 % y has to be a row vector of odd length, values returned as row vector, too.
5 N = length(y); if (mod(N,2)~=1), error('#pts odd!'); end;
6 n = (N-1)/2;
7 % Compute coefficients  $\gamma_j$  in (9.3.5), see (9.3.8)
8 c = fft(exp(2*pi*i*(n/N)*(0:2*n)).*y)/N;
9 ch = [gamma, zeros(1,N-(2*n+1))]; % zero padding
10 v = conj(fft(conj(ch))); % Evaluate multiplication with conjugate Fourier
   % matrix
11 q = exp(-2*pi*i*n*(0:N-1)/N).*v; % undo rescaling

```

Now we study the asymptotic behavior of the error of trigonometric interpolation as  $n \rightarrow \infty$ .

*Example 9.3.18* (Interpolation error: trigonometric interpolation).

#1 Step function:  $f(t) = 0$  for  $|t - \frac{1}{2}| > \frac{1}{4}$ ,  $f(t) = 1$  for  $|t - \frac{1}{2}| \leq \frac{1}{4}$

#2  $C^\infty$  periodic function:  $f(t) = \frac{1}{\sqrt{1 + \frac{1}{2} \sin(2\pi t)}}$ .

#3 “wedge function”:  $f(t) = |t - \frac{1}{2}|$

Approximate computation of norms of interpolation errors on equidistant grid with 4096 points.

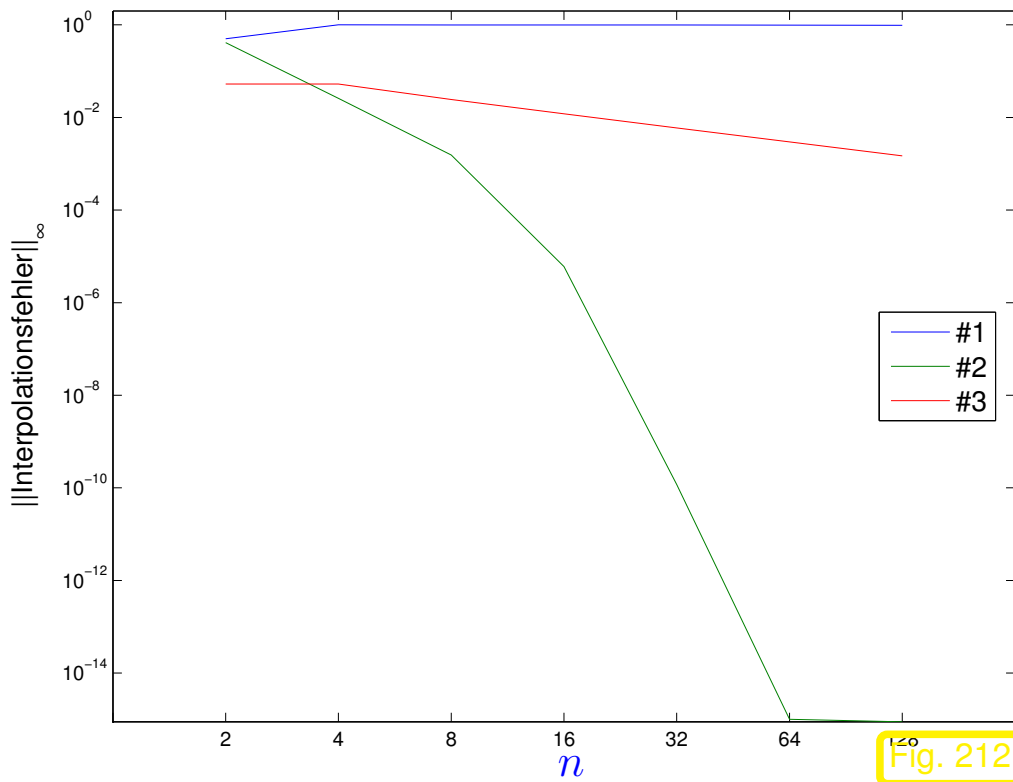


Fig. 212

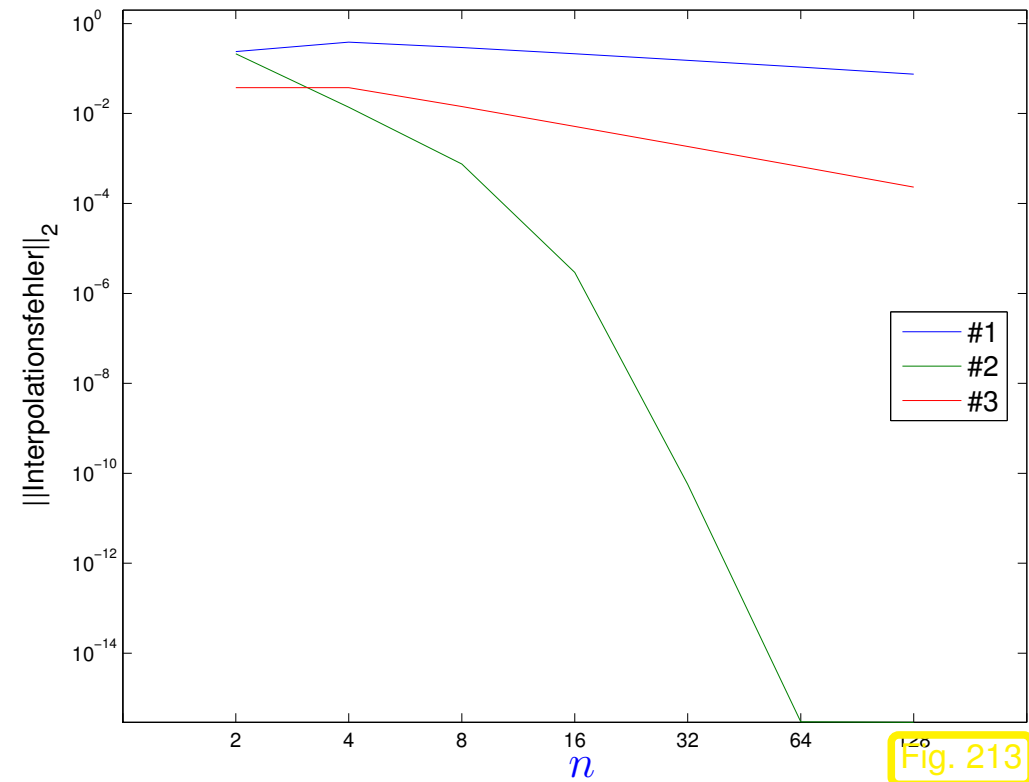


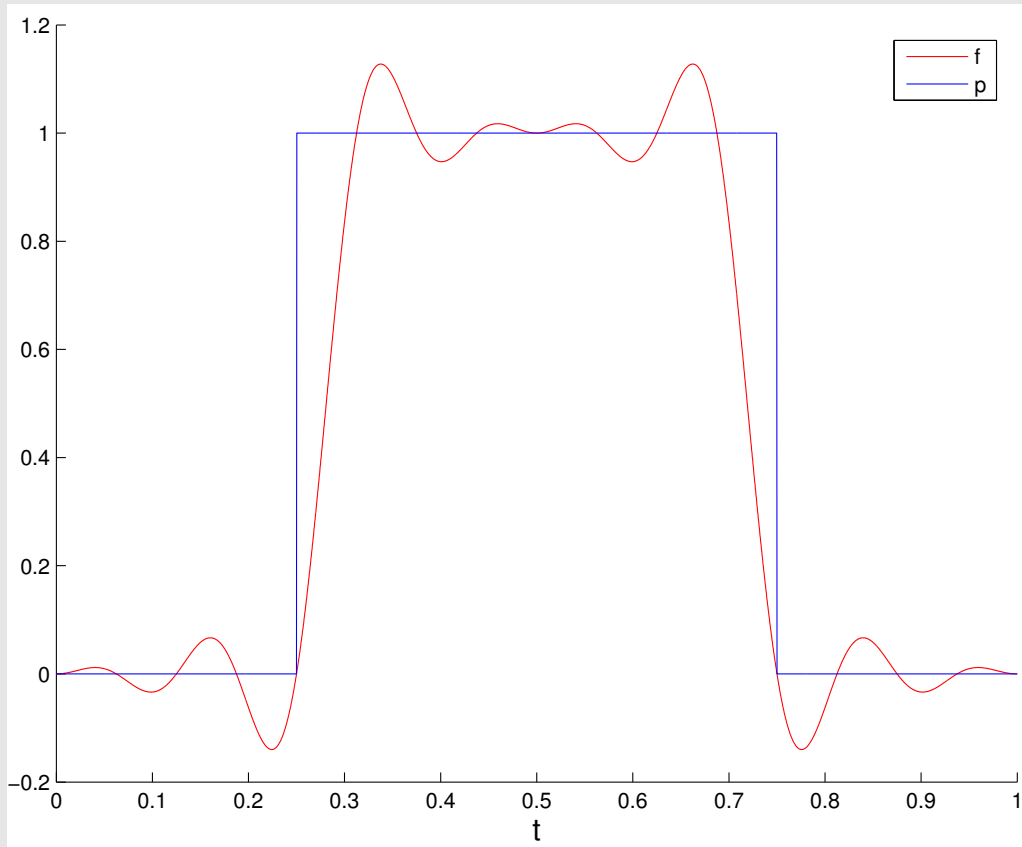
Fig. 213

Observation: Function #1: no convergence in  $L^{\infty}$ -norm, algebraic convergence  
 Function #3: algebraic convergence in both norms  
 Function #2: exponential convergence in both norms

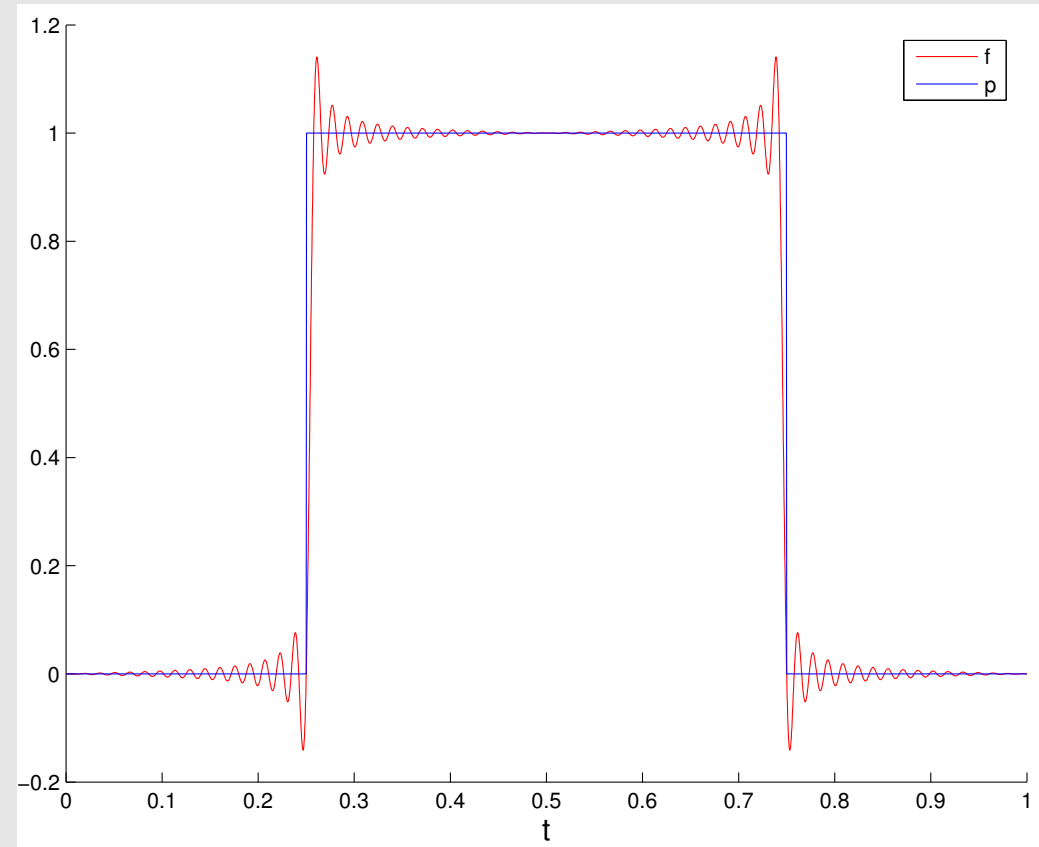


Example 9.3.19 (Gibbs phenomenon).

Trigonometric interpolants of step function fail to converge in  $L^\infty$ -norm in Ex. 9.3.18. A closer inspection of the interpolants shows why:



$n = 16$



$n = 128$

Observation: overshooting in neighborhood of discontinuity: **Gibbs phenomenon**



Example 9.3.20 (Trigonometric interpolation of analytic functions).

$$f(t) = \frac{1}{\sqrt{1 - \alpha \sin(2\pi t)}} \quad \text{auf } I = [0, 1] . \quad (9.3.21)$$

Approximative computations of error norms by “oversampling” in 4096 points.

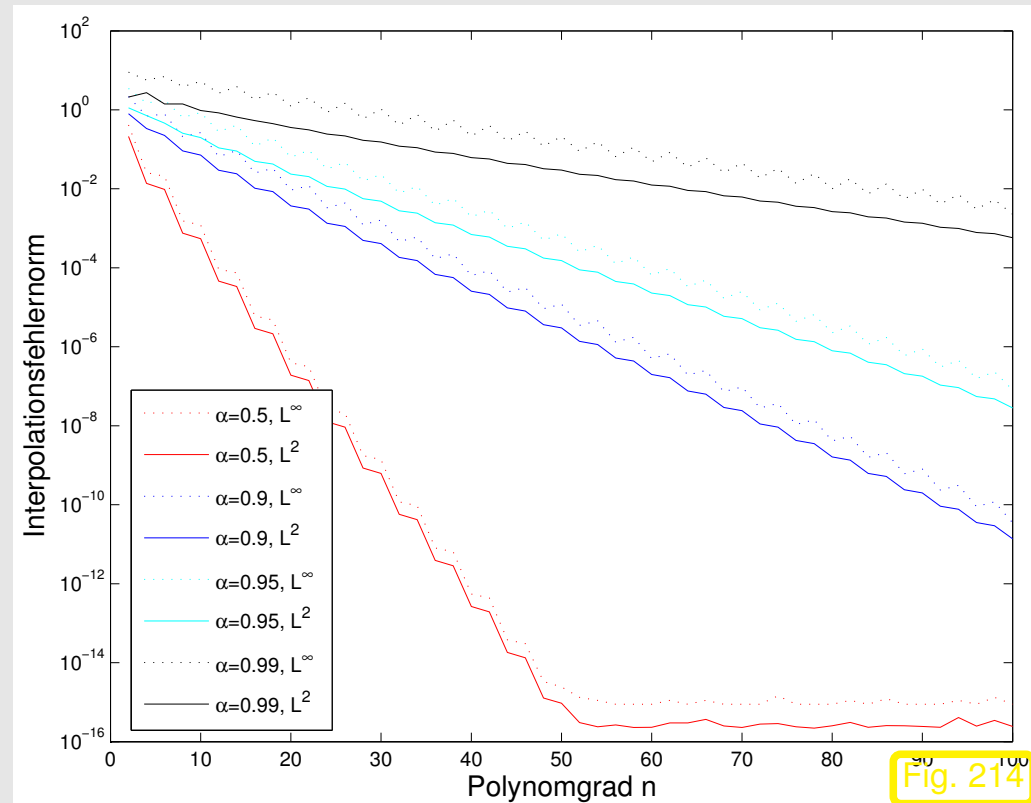


Fig. 214

➤ Observation: exponential convergence in  $n$ , faster for smaller  $\alpha$

Explanation, cf. Rem. 9.2.19:

Similar to Chebychev interpolants, also trigonometric interpolants converge exponentially fast, if the interpoland  $f$  is 1-periodic *analytic* ( $\rightarrow$  Def. 9.2.20) in a strip around the real axis in  $\mathbb{C}$ . The speed of convergence depends on the width of the strip.

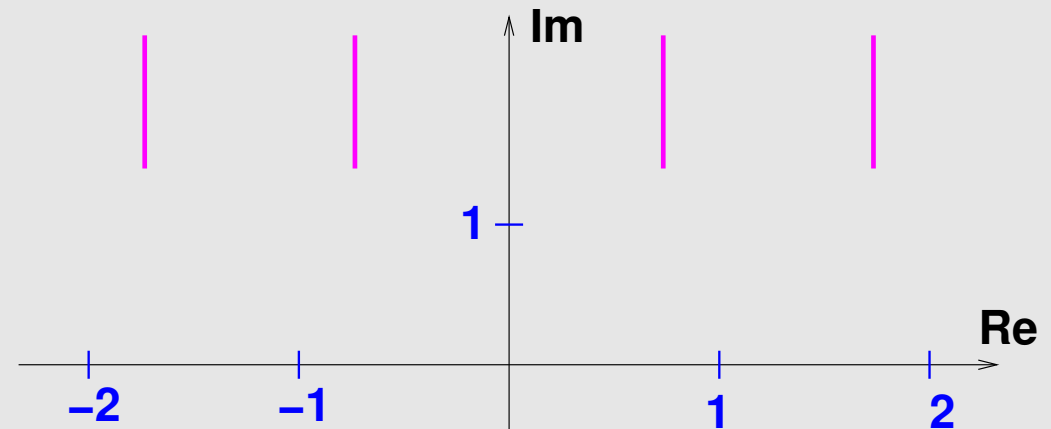
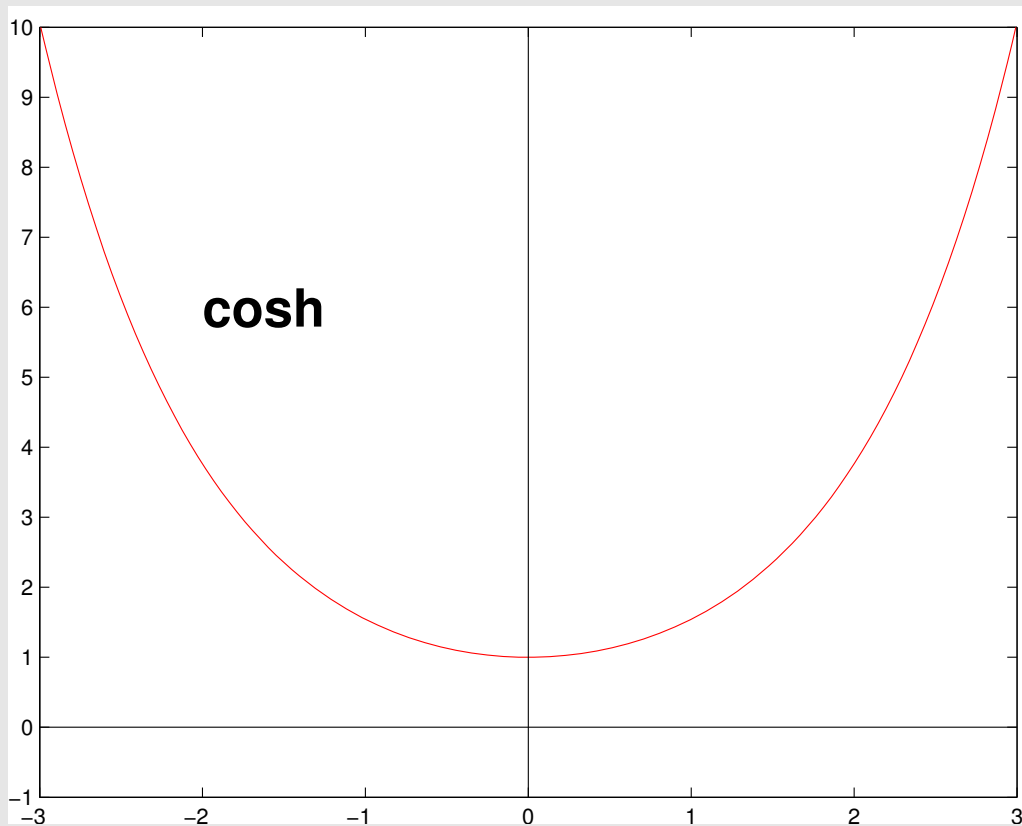
Note that  $f$  from (9.3.21) is holomorphic, where

$$1 + \alpha \sin(2\pi z) \notin \mathbb{R}_0^- \Leftrightarrow \sin(2\pi z) = \sin(2\pi x) \cosh(2\pi y) + i \cos(2\pi x) \sinh(2\pi y) \notin ] -\infty, -1 - \frac{1}{\alpha}] ,$$

because the square root is holomorphic in  $\mathbb{C} \setminus \mathbb{R}_0^-$ .

Domain of analyticity of  $f$ :

$$\mathbb{C} \setminus \bigcup_{k \in \mathbb{Z}} \left( \frac{k}{2} + \frac{1}{4} + i(\mathbb{R} \setminus ] - \zeta, \zeta[) \right), \quad \zeta \in \mathbb{R}^+, \quad \cosh(2\pi\zeta) = 1 + \frac{1}{\alpha} .$$



➤  $f$  analytic in **strip**

$$S := \{z \in \mathbb{C} : -\zeta < \text{Im}(z) < \zeta\} .$$

➤ As  $\alpha$  decreases the strip of analyticity becomes wider, since  $x \rightarrow \cosh(x)$  is increasing for  $x > 0$ .

## 9.4 Approximation by piecewise polynomials

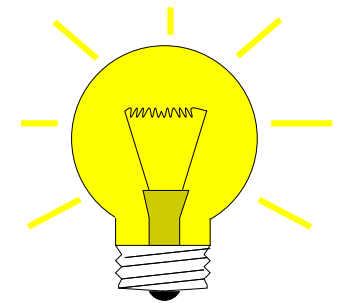
Alternatives to interpolation by global polynomials discussed in Ch. 3:

- piecewise linear/quadratic interpolation → Sect. 3.6.1, Ex. 3.6.7,
- cubic Hermite interpolation → Sect. 3.7,
- (cubic) spline interpolation → 3.8.1.

☞ All these interpolation schemes rely on **piecewise polynomials** (of different global smoothness)

Focus in this section: function approximation by *piecewise polynomial* interpolants

“piecewise” ➤ refer to partitioning of interval on which we aim to approximate:



Idea: use **piecewise polynomials** with respect to a **grid/mesh**

$$\mathcal{M} := \{a = x_0 < x_1 < \dots < x_{m-1} < x_m = b\} \quad (9.4.1)$$

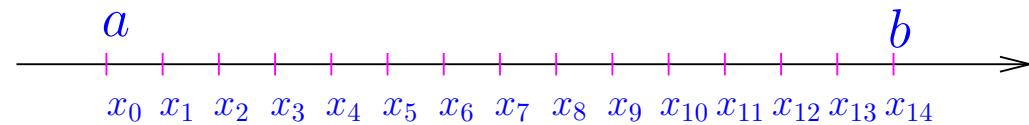
to approximate function  $f : [a, b] \mapsto \mathbb{R}$ ,  $a < b$ .

Spline terminology, *cf.* Def. 3.8.1: mesh is also called knot set.



## Terminology:

- $x_j \hat{=}$  **nodes** of the mesh  $\mathcal{M}$ ,
- $[x_{j-1}, x_j[ \hat{=}$  **intervals/cells** of the mesh,
- $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}| \hat{=}$  **mesh width**,
- If  $x_j = a + jh \hat{=}$  **equidistant** (uniform) mesh with meshwidth  $h > 0$



## 9.4.1 Piecewise Lagrange interpolation

Recall theory of polynomial interpolation  $\rightarrow$  Sect. 3.3:  $n + 1$  data points needed to fix interpolating polynomial, see Thm. 3.3.5.

► Approach to *local* Lagrange interpolation ( $\rightarrow$  (3.3.1)) of  $f \in C(I)$  on mesh  $\mathcal{M} := \{a = x_0 < x_1 < \dots < x_{m-1} < x_m = b\}$ :

❶ choose set of *local* interpolation points


$$\mathcal{T}^j := \{t_0^j, \dots, t_{n_j}^j\} \subset I_j = [x_{j-1}, x_j], \quad j = 1, \dots, m,$$

for each grid interval, and prescribed **local degree**  $n_j \in \mathbb{N}_0$ .

② define piecewise interpolating polynomial  $s : [x_0, x_m] \rightarrow \mathbb{K}$ :

$$s_j := s|_{I_j} \in \mathcal{P}_{n_j} \quad \text{and} \quad s_j(t_i^j) = f(t_i^j) \quad i = 0, \dots, n_j, \quad j = 1, \dots, m. \quad (9.4.2)$$

- The mapping  $f \mapsto s$  defines a **linear** operator  $\mathbf{l}_{\mathcal{M}} : C^0([a, b]) \mapsto C_{\mathcal{M}, \text{pw}}^0([a, b])$ .
- $\mathbf{l}_{\mathcal{M}}$  depends on  $\mathcal{M}$ , the local degrees  $n_j$ , and the sets  $\mathcal{T}_j$  of local interpolation points (the latter two are suppressed in notation).

 notation:  $C_{\mathcal{M}, \text{pw}}^0(I) \hat{=}$  space of piecewise continuous functions on interval  $I$

Obvious: sufficient condition for global **continuity** of piecewise polynomial interpolant:

$$t_{n_j}^j = t_0^{j+1} \quad \forall j = 1, \dots, m-1 \quad \Rightarrow \quad s \in C^0([a, b]). \quad (9.4.3)$$

Focus: **asymptotic** behavior of (some norm of) interpolation error

$$\|f - I_{\mathcal{M}}f\| \leq CT(N) \quad \text{for } N \rightarrow \infty, \quad (9.4.4)$$

where  $N := \sum_{j=1}^m (n_j + 1)$ .

The decays of the bound  $T(N)$  will characterize the type of convergence: algebraic convergence, exponential convergence, see Sect. 9.1.

But why do we choose this strange number  $N$  ?

Because it agrees with the dimension of the space

$$\{q \in C_{\mathcal{M},\text{pw}}^0 : q|_{I_j} \in \mathcal{P}_{n_j} \quad \forall j = 1, \dots, m\} \quad !$$

this dimension tells us the number of reals we need to describe the interpolant  $s$ , that is, the “information cost” of  $s$ .

$N$  is proportional to the number of interpolation conditions = number of  $f$ -evaluations needed to compute  $s$  (why only proportional in general?).

Special case: **uniform** polynomial degree  $n_j = n$  for all  $j = 1, \dots, m$ .

▶ Study estimates  $\|f - I_{\mathcal{M}}f\| \leq CT(h_{\mathcal{M}})$  for  $h_{\mathcal{M}} \rightarrow 0$

in terms of meshwidth  $h_{\mathcal{M}}$ .

Terminology: study of  **$h$ -convergence**

*Example 9.4.5* ( $h$ -convergence of piecewise polynomial interpolation).

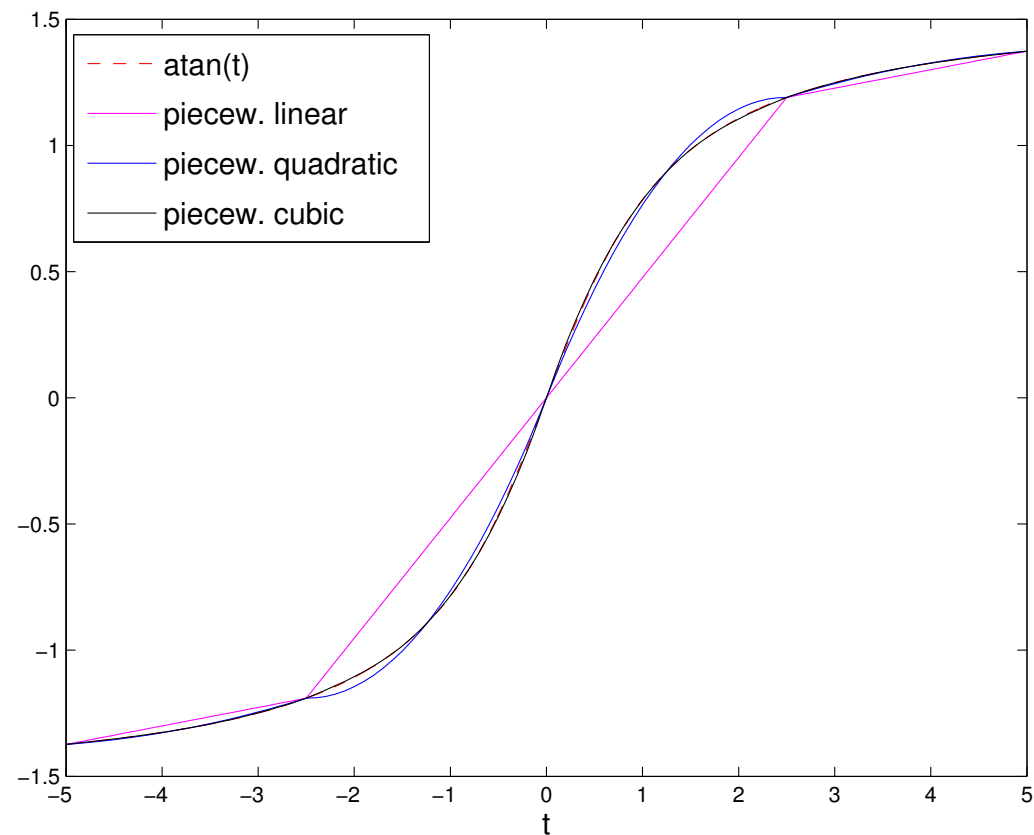
Compare Ex. 3.6.5:

$$f(t) = \arctan t, I = [-5, 5]$$

$$\text{Grid } \mathcal{M} := \left\{-5, -\frac{5}{2}, 0, \frac{5}{2}, 5\right\}$$

$\mathcal{T}^j$  equidistant in  $I_j$ .

Plots of the piecewise linear, quadratic and cubic polynomial interpolants →

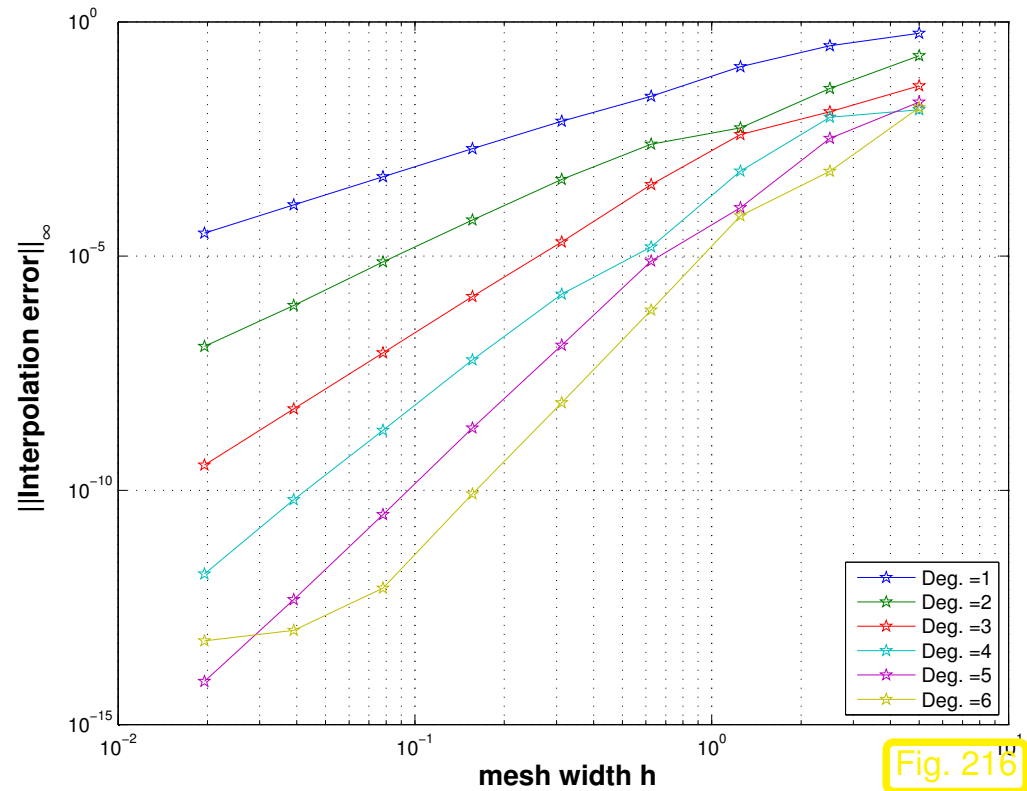
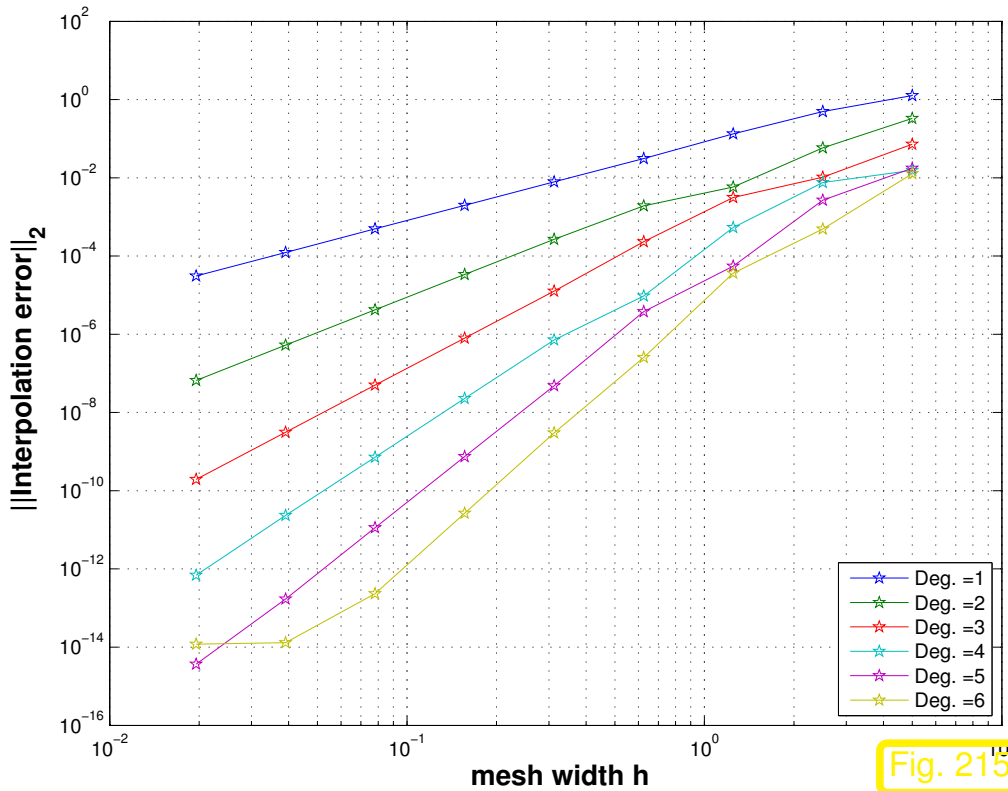


- Sequence of (equidistant) meshes:  $\mathcal{M}_i := \{-5 + j 2^{-i} 10\}_{j=0}^{2^i}, \quad i = 1, \dots, 6.$
- Equidistant local interpolation nodes (endpoints of grid intervals included).

Monitored: interpolation error in (approximate)  $L^\infty$ - and  $L^2$ -norms, see (9.2.18), (9.2.17)

$$\|g\|_{L^\infty([-5,5])} \approx \max_{j=0,\dots,1000} |g(-5 + j/100)|,$$

$$\|g\|_{L^2([-5,5])} \approx \sqrt{\frac{1}{1000}} \cdot \left( \frac{1}{2}g(-5)^2 + \sum_{j=1}^{999} |g(-5 + j/100)|^2 + \frac{1}{2}g(5)^2 \right)^{1/2}.$$



Observation:

Algebraic convergence in meshwidth

(nearly linear error norm graphs in doubly logarithmic scale, see Rem. 9.1.4)

Observation: rate of algebraic convergence increases with polynomial degree  $n$

Rates  $\alpha$  of algebraic convergence  $O(h_{\mathcal{M}}^{\alpha})$  of norms of interpolation error:

$n$	1	2	3	4	5	6
w.r.t. $L^2$ -norm	1.9957	2.9747	4.0256	4.8070	6.0013	5.2012
w.r.t. $L^{\infty}$ -norm	1.9529	2.8989	3.9712	4.7057	5.9801	4.9228

➤ Higher polynomial degree provides faster algebraic decrease of interpolation error norms.

Here: rates estimated by linear regression ( $\rightarrow$  Ex. 7.0.9) based on MATLAB's `polyfit` and the interpolation errors for meshwidth  $h \leq 10 \cdot 2^{-5}$ . This was done in order to avoid erratic “preasymptotic”, that is, for large meshwidth  $h$ , behavior of the error.

The bad rates for  $n = 6$  are due to the impact of roundoff, because the norms of the interpolation error had dropped below machine precision, see Figs. 215, 216.

These observations are easily explained by applying the polynomial interpolation error estimates of Sect. 9.1 *locally* on the mesh intervals: for constant polynomial degree  $n = n_j$ ,  $j = 1, \dots, m$

$$(9.1.12) \Rightarrow \|f - s\|_{L^\infty([x_0, x_m])} \leq \frac{h^{n+1}}{(n+1)!} \|f^{(n+1)}\|_{L^\infty([x_0, x_m])}, \quad (9.4.6)$$

with **mesh width**  $h := \max\{|x_j - x_{j-1}|: j = 1, \dots, m\}$ .

R. Hiptmair  
rev 38286,  
December  
8, 2010

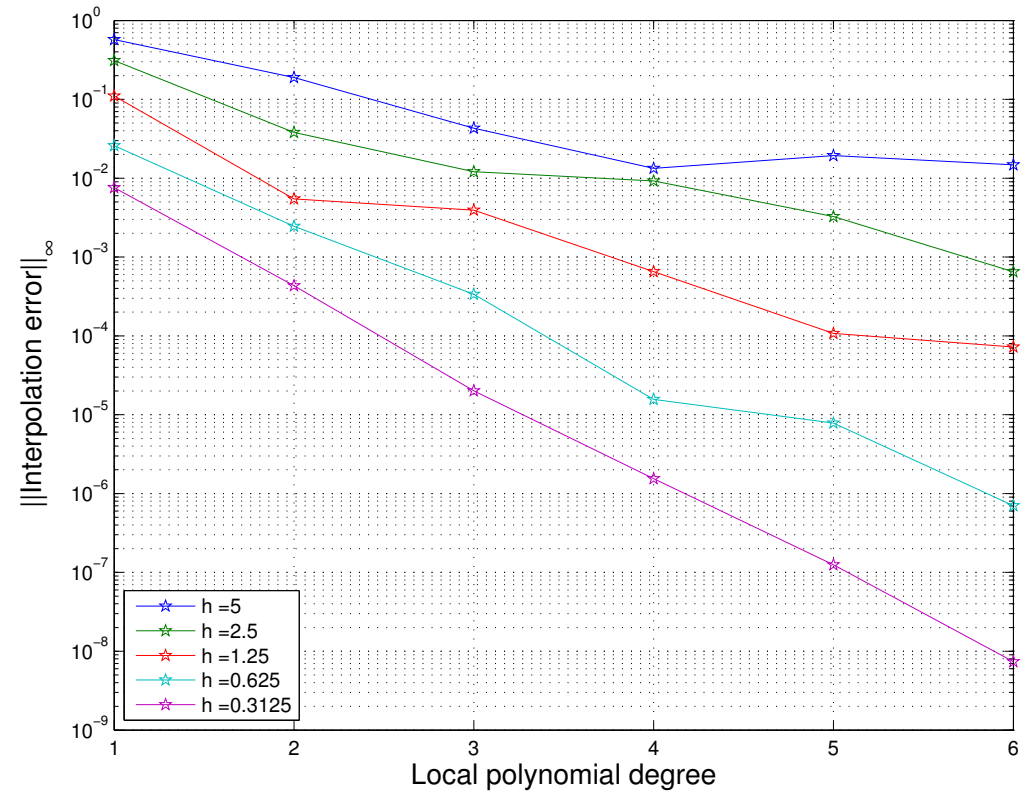
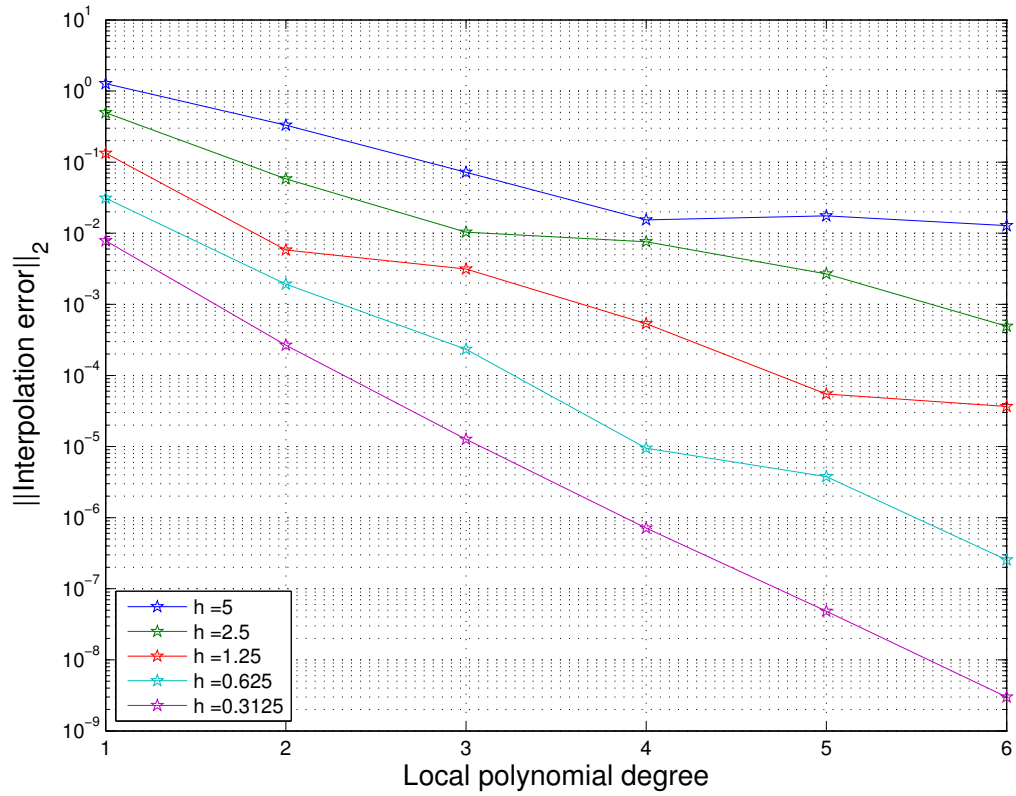
Another special case: fixed mesh  $\mathcal{M}$ , uniform polynomial degree  $n$

Study estimates  $\|f - I_{\mathcal{M}}f\| \leq CT(n)$  for  $n \rightarrow \infty$ .

Terminology: investigation of ***p*-convergence**



Example 9.4.7 ( $p$ -convergence of piecewise polynomial interpolation).



Observation: (apparent) exponential convergence in polynomial degree

Note: in the case of  $p$ -convergence the situation is the same as for standard polynomial interpolation, see 9.1.

In this example we deal with an analytic function, see Rem. 9.2.19. Though equidistant local interpolation nodes are used *cf.* Ex. 9.1.5, the mesh intervals seems to be small enough that even in this case exponential convergence prevails.



## 9.4.2 Cubic Hermite interpolation: error estimates

See Sect. 3.7 for definition and algorithms for cubic Hermite interpolation of data points.

*Example 9.4.8* (Convergence of Hermite interpolation with exact slopes).

Piecewise cubic Hermite interpolation of

$$f(x) = \arctan(x) .$$

- domain:  $I = (-5, 5)$
- mesh  $\mathcal{T} = \{-5 + hj\}_{j=0}^n \subset I, h = \frac{10}{n}$ ,
- exact slopes  $c_i = f'(t_i), i = 0, \dots, n$

▶ algebraic convergence  $O(h^4)$

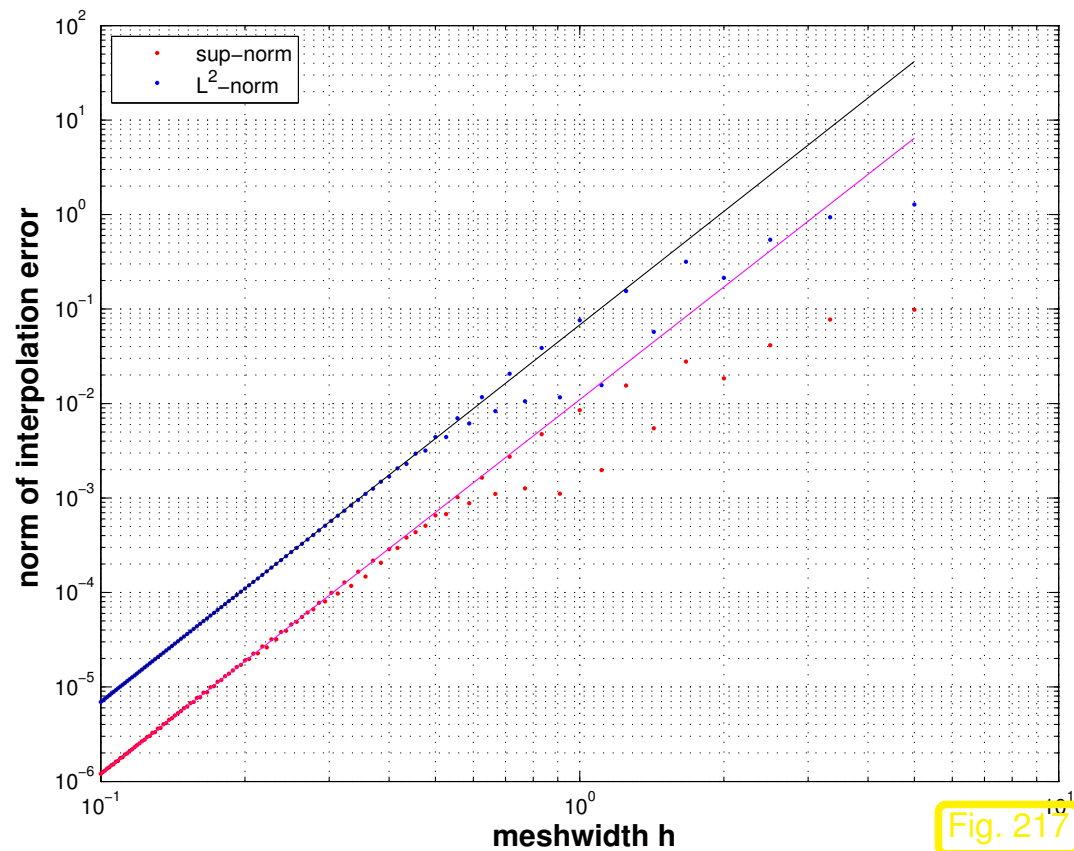


Fig. 217

Approximate computation of error norms analogous to Ex. 9.4.5.

Code 9.4.9: Hermite approximation and orders of convergence with exact slopes

```
1 function hermiteapprox1(f, df, a, b, N)
```

```
2 % Investigation of interpolation error norms for cubic Hermite interpolation of
  f (handle f)
3 % on [a,b] with slopes given by f' (function handle df).
4 % N gives the maximum number of mesh intervals
5 err = [];
6 for j=2:N
7     xx=a; % xx is the fine mesh on which the error norms are computed
8     val=f(a); % function values of mesh xx
9
10    t = a:(b-a)/j:b; % mesh nodes
11    y = f(t); c=df(t); % function values and slopes provide coefficients
12
13    for k=1:j-1
14        vx = linspace(t(k),t(k+1), 100);
15        % See Code 3.7.2 for local evaluation of Hermite interpolant
16        locval=hermloceval(vx,t(k),t(k+1),y(k),y(k+1),c(k),c(k+1));
17        xx=[xx, vx(2:100)];
18        val=[val,locval(2:100)];
19    end
20    d = abs(feval(f,xx)-val); % pointwise error on mesh xx
21    h = (b-a)/j;
22    % compute L^2 norm of the error using trapezoidal rule
23    l2 = sqrt(h*(sum(d(2:end-1).^2)+(d(1)^2+d(end)^2)/2));
24    % columns of err = meshwidth, sup-norm error, L^2 error:
25    err = [err; h, max(d),l2];
26 end
27
28 figure('Name','Hermite approximation');
29 loglog(err(:,1),err(:,2),'r.',err(:,1),err(:,3),'b.');
```

```
30 grid on;
31 xlabel ('{\bf meshwidth h}', 'fontsize', 14);
32 ylabel ('{\bf norm of interpolation error}', 'fontsize', 14);
33 legend ('sup-norm', 'L^2-norm', 'location', 'northwest');
34
35 % compute estimates for algebraic orders of convergence
36 % using linear regression on half of the data points
37 pI = polyfit (log (err (ceil (N/2) : N-2, 1)), log (err (ceil (N/2) : N-2, 2)), 1);
    exp_rate_Linf = pI(1)
38 pL2 = polyfit (log (err (ceil (N/2) : N-2, 1)), log (err (ceil (N/2) : N-2, 3)), 1);
    exp_rate_L2 = pL2(1)
39 hold on;
40 plot ([err(1,1), err(N-1,1)], [err(1,1), err(N-1,1)].^pI(1) * exp (pI(2)), 'm')
41 plot ([err(1,1), err(N-1,1)], [err(1,1), err(N-1,1)].^pL2(1) * exp (pL2(2)), 'k')
42
43 print -depsc2 '../PICTURES/hermiperrslopes.eps';
```

```
Try: hermiteapprox1 (@atan, @(x) 1./(1+x.^2), -5, 5, 100);
```



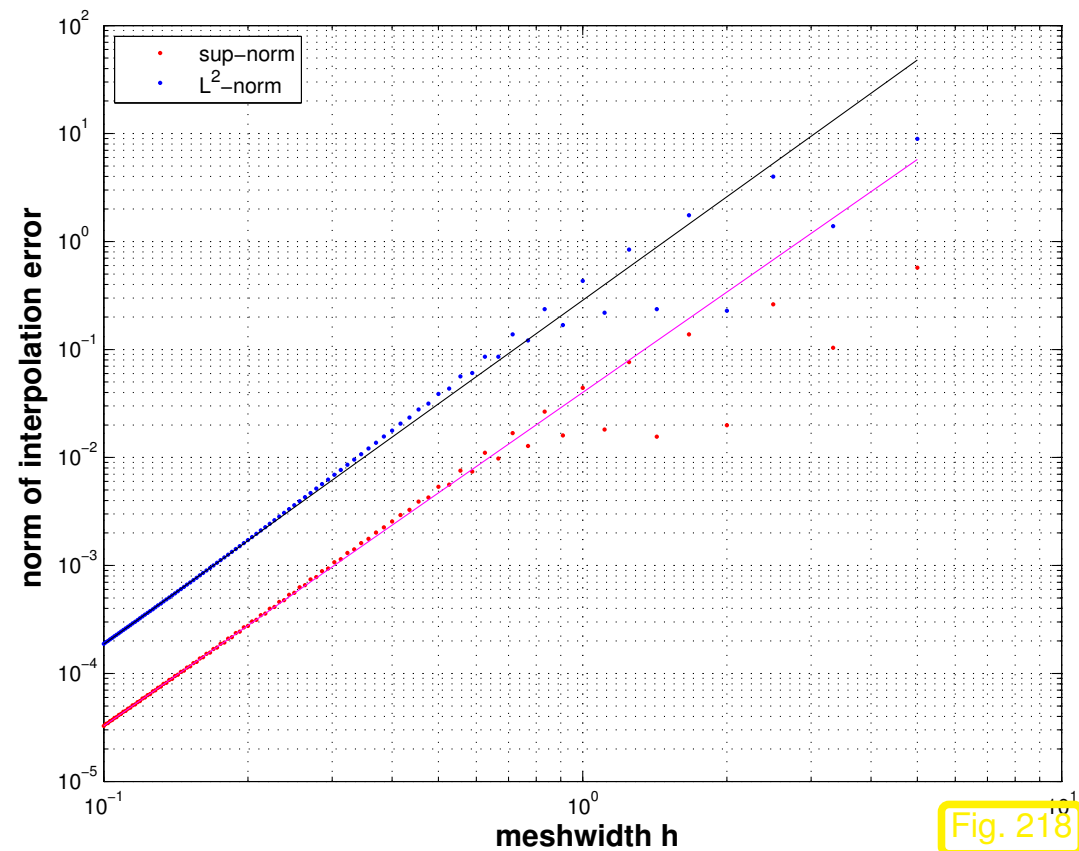
Example 9.4.10 (Convergence of Hermite interpolation with averaged slopes).

Piecewise cubic Hermite interpolation of

$$f(x) = \arctan(x) .$$

- domain:  $I = (-5, 5)$
- equidistant mesh  $\mathcal{T}$  in  $I$ , see Ex. 9.4.8,
- averaged local slopes, see (3.7.4)

► algebraic convergence  $O(h^3)$  in meshwidth  
(see Code 9.4.10)



Lower order of convergence due to the choice of the slopes (3.7.4):  
from the plot  $L^\infty$ -norm  $\sim L^2$ -norm  $\sim O(h^3)$



## Code 9.4.11: Hermite approximation and orders of convergence

```
1 % 16.11.2009 hermiteapprox.m
2 % Plot convergence of approximation error of cubic hermite interpolation
3 % with respect to the meshwidth
4 % print the algebraic order of convergence in sup and  $L^2$  norms
5 % Slopes: weighted average of local slopes
6 %
7 % inputs: f function to be interpolated
8 % a, b left and right extremes of the interval
9 % N maximum number of subdomains
0
1 function hermiteapprox(f,a,b,N)
2 % Investigation of interpolation error norms for cubic Hermite interpolation of
   f (handle f)
3 % on [a,b] with linearly averaged slopes according to (3.7.4).
4 % N gives the maximum number of mesh intervals
5 err = [];
6 for j=2:N
7     xx=a;      % xx is the fine mesh on which the error norms are computed
8     val=f(a); % function values on xx
9
10    t = a:(b-a)/j:b;      % mesh nodes
11    y = f(t); c=slopes1(t,y); % coefficients for Hermit polynomial representation
12
13    for k=1:j-1
14        vx = linspace(t(k),t(k+1), 100);
15        locval=hermloceval(vx,t(k),t(k+1),y(k),y(k+1),c(k),c(k+1));
16        xx=[xx, vx(2:100)];
17        val=[val, locval(2:100)];
```

```
28 end
29 d = abs( feval(f,xx) - val);
30 h = (b-a)/j;
31 % compute  $L^2$  norm of the error using trapezoidal rule
32 l2 = sqrt(h*(sum(d(2:end-1).^2)+(d(1)^2+d(end)^2)/2) );
33 % columns of err = meshwidth, sup-norm error,  $L^2$  error:
34 err = [err; h, max(d), l2];
35 end
36
37 figure('Name','Hermite approximation');
38 loglog(err(:,1),err(:,2),'r.',err(:,1),err(:,3),'b.');
```

```
39 grid on;
40 xlabel('\bf meshwidth h','fontsize',14);
41 ylabel('\bf norm of interpolation error','fontsize',14);
42 legend('sup-norm','L^2-norm','location','northwest');
```

43

```
44 % compute estimates for algebraic orders of convergence
45 % using linear regression on half of the data points
46 pI = polyfit(log(err(ceil(N/2):N-2,1)),log(err(ceil(N/2):N-2,2)),1);
    exp_rate_Linf=pI(1)
47 pL2 = polyfit(log(err(ceil(N/2):N-2,1)),log(err(ceil(N/2):N-2,3)),1);
    exp_rate_L2=pL2(1)
48 hold on;
49 plot([err(1,1),err(N-1,1)], [err(1,1),err(N-1,1)].^pI(1)*exp(pI(2)),'m')
50 plot([err(1,1),err(N-1,1)], [err(1,1),err(N-1,1)].^pL2(1)*exp(pL2(2)),'k')
51
52 print -depsc2 '../PICTURES/hermiperravgsl.eps';
53
54 %-----
```



```

55 function c=slopes1(t,y)
56 h = diff(t); % increments in t
57 delta = diff(y)./h; % slopes of piecewise linear
   interpolant
58 c = [delta(1),...
59      ((h(2:end).*delta(1:end-1)+h(1:end-1).*delta(2:end))...
60      ./ (t(3:end) - t(1:end-2)) ),...
61      delta(end)];

```

Try: `hermiteapprox(@atan, -5, 5, 100);`

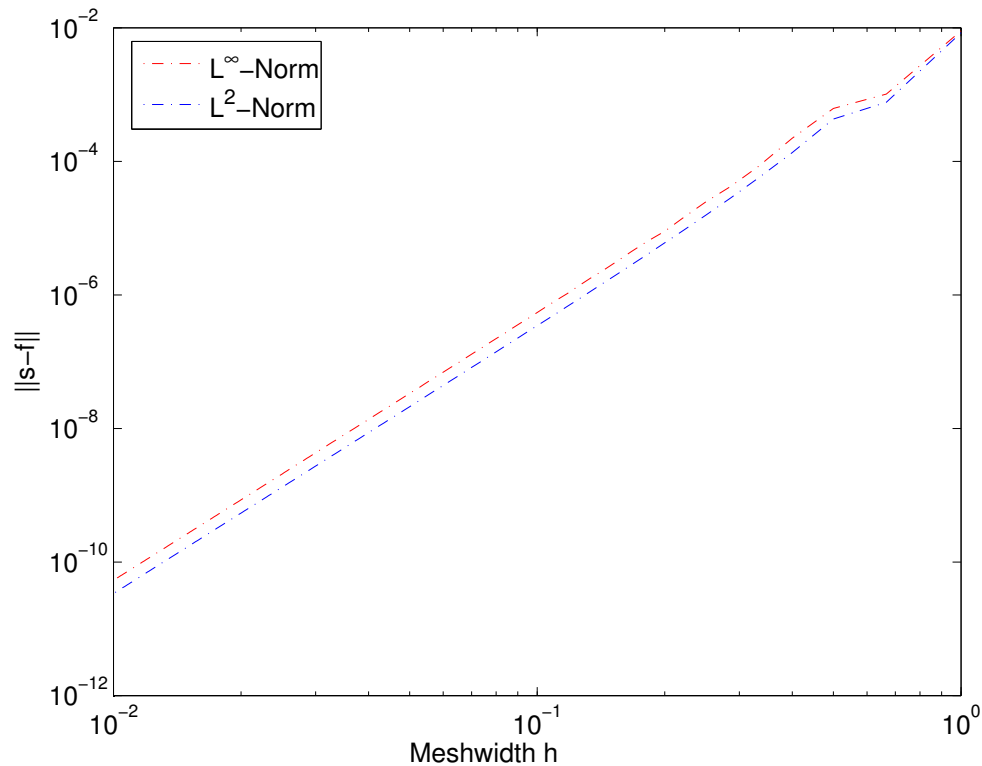
### 9.4.3 Cubic spline interpolation: error estimates [35, Ch. 47]

Recall: concept and algorithms for cubic spline interpolation from Sect. 3.8.1.

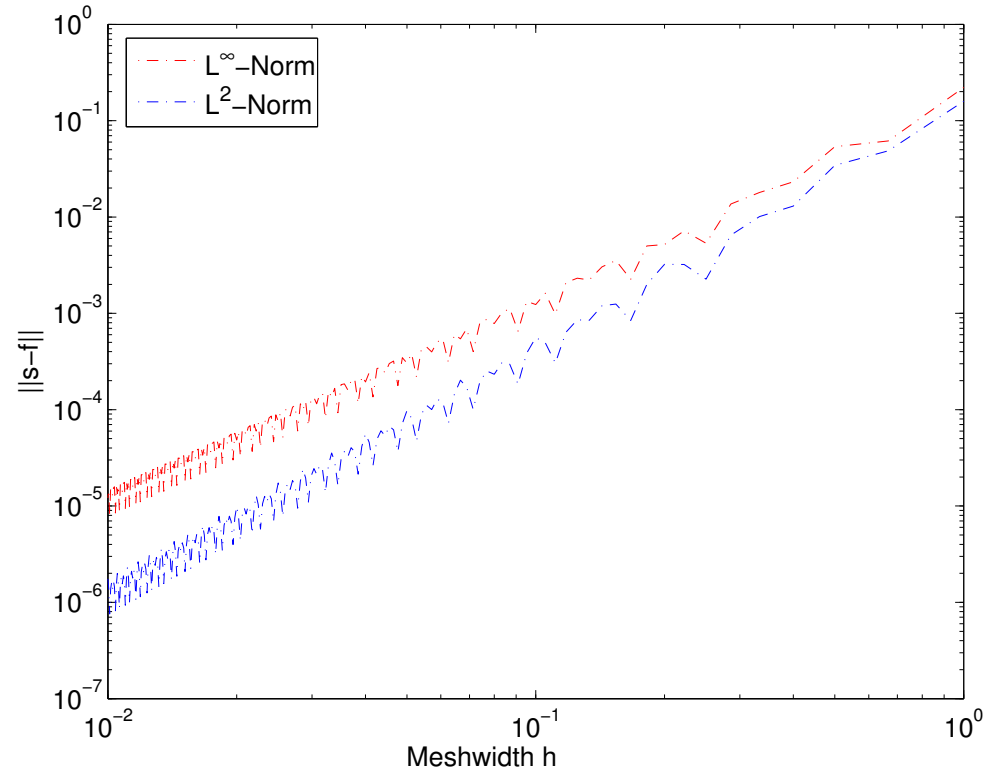
*Example 9.4.12* (Approximation by complete cubic spline interpolants).

Grid (knot set)  $\mathcal{M} := \{-1 + \frac{2}{n}j\}_{j=0}^n, n \in \mathbb{N} \rightarrow$  meshwidth  $h = 2/n, I = [-1, 1]$

$$f_1(t) = \frac{1}{1 + e^{-2t}} \in C^\infty(I) \quad , \quad f_2(t) = \begin{cases} 0 & , \text{if } t < -\frac{2}{5} , \\ \frac{1}{2}(1 + \cos(\pi(t - \frac{3}{5}))) & , \text{if } -\frac{2}{5} < t < \frac{3}{5} , \\ 1 & \text{otherwise.} \end{cases} \in C^1(I) .$$



$$\|f_1 - s\|_{L^\infty([-1,1])} = O(h^4)$$



$$\|f_2 - s\|_{L^\infty([-1,1])} = O(h^2)$$

Algebraic order of convergence in  $h = \min \{ 1 + \text{regularity of } f, 4 \}$ .

Theory [34]:  $f \in C^4([t_0, t_n]) \rightarrow \|f - s\|_{L^\infty([t_0, t_n])} \leq \frac{5}{384} h^4 \left\| f^{(4)} \right\|_{L^\infty([t_0, t_n])}$

See also [13, Rem. 9.2],



### Code 9.4.13: Spline approximation error

```
1 function splineapprox(f,df,a,b,N)
2 % Study of interpolation error norms for complete cubic spline interpolation of
3 f
4 % on equidistant knot set.
5 x = a:0.00025:b; fv = feval(f,x); % fine mesh for norm evaluation
6 dfa = feval(df,a); dfb = feval(df,b); % Slopes at endpoints
7 err = [];
8 for j=2:N
9     t = a:(b-a)/j:b;           % spline knots
10    y = [dfa, feval(f,t),dfb];
11    % compute complete spline imposing exact first derivative at the endpoints
12    % Please refer to MATLAB documentation of spline
13    v = spline(t,y,x);
14    d = abs(fv-v);
15    h = x(2:end)-x(1:end-1);
```

R. Hiptmair  
rev 38286,  
December  
8, 2010

```

5 % compute  $L^2$  norm of the error using trapezoidal rule on mesh x
6 l2 = sqrt(0.5*dot(h, (d(1:end-1).^2+d(2:end).^2)));
7 % columns of err = meshwidth,  $L^\infty$  error,  $L^2$  error:
8 err = [err; (b-a)/j, max(d), l2];
9 end
10
11 figure('Name','Spline interpolation');
12 plot(t, y(2:end-1), 'm*', x, fv, 'b-', x, v, 'r-');
13 xlabel('\bf t', 'fontsize', 14); ylabel('\bf s(t)', 'fontsize', 14);
14 legend('Data points', 'f', 'Cubic spline interpolant', 'location', 'best');
15
16 figure('Name','Spline approximation error');
17 loglog(err(:,1), err(:,2), 'r.-', err(:,1), err(:,3), 'b.-');
18 grid on; xlabel('Meshwidth h'); ylabel('||s-f||');
19 legend('sup-norm', 'L^2-norm', 'Location', 'NorthWest');
20 %compute algebraic orders of convergence using polynomial fit
21 p = polyfit(log(err(:,1)), log(err(:,2)), 1); exp_rate_Linf=p(1)
22 p = polyfit(log(err(:,1)), log(err(:,3)), 1); exp_rate_L2=p(1)

```

Try:

```

1 splineapprox(@atan, @(x) 1./(1+x.^2), -5, 5, 100);
2 splineapprox(@(x) 1./(1+exp(-2*x)), ...
3             @(x) 2*exp(-2*x)./(1+exp(-2*x)).^2, -1, 1, 100);

```

# 10

## Numerical Quadrature [35, VII], [13, Ch. 10]

### Numerical quadrature

= Approximate numerical evaluation of  $\int_{\Omega} f(\mathbf{x}) \, d\mathbf{x}$ , integration domain  $\Omega \subset \mathbb{R}^d$

Setting: continuous function  $f : \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}$  only available as

`function y = f(x)`

(point evaluation, available as *possibly expensive* “black-box” function)

Here: focus on special case  $d = 1$ :  $\Omega = [a, b]$  (interval)

*Remark 10.0.1* (Importance of numerical quadrature).

☞ Numerical quadrature methods are key building blocks for methods for the numerical treatment of partial differential equations (➤ Course “Numerical treatment of partial differential equations”) ☜

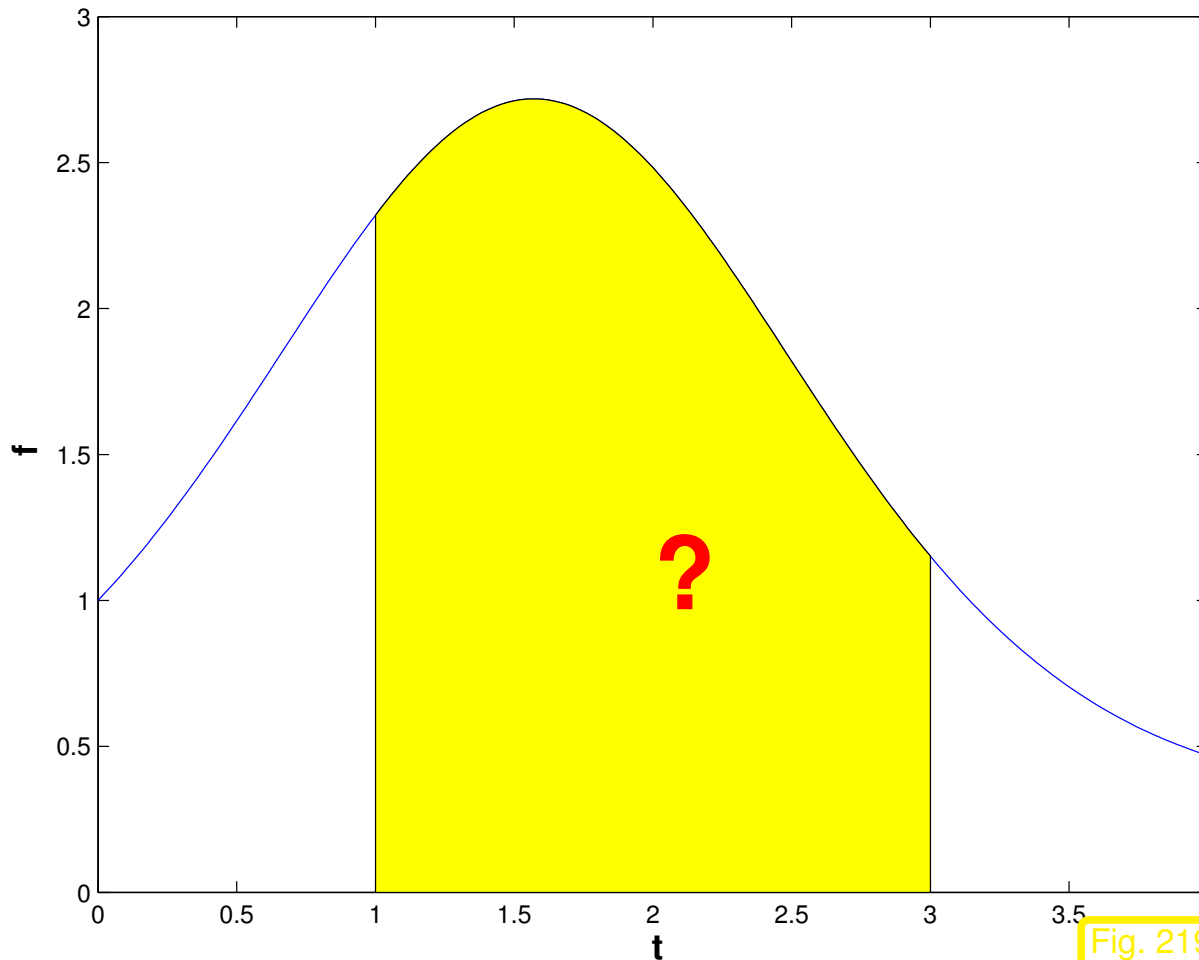


Fig. 219

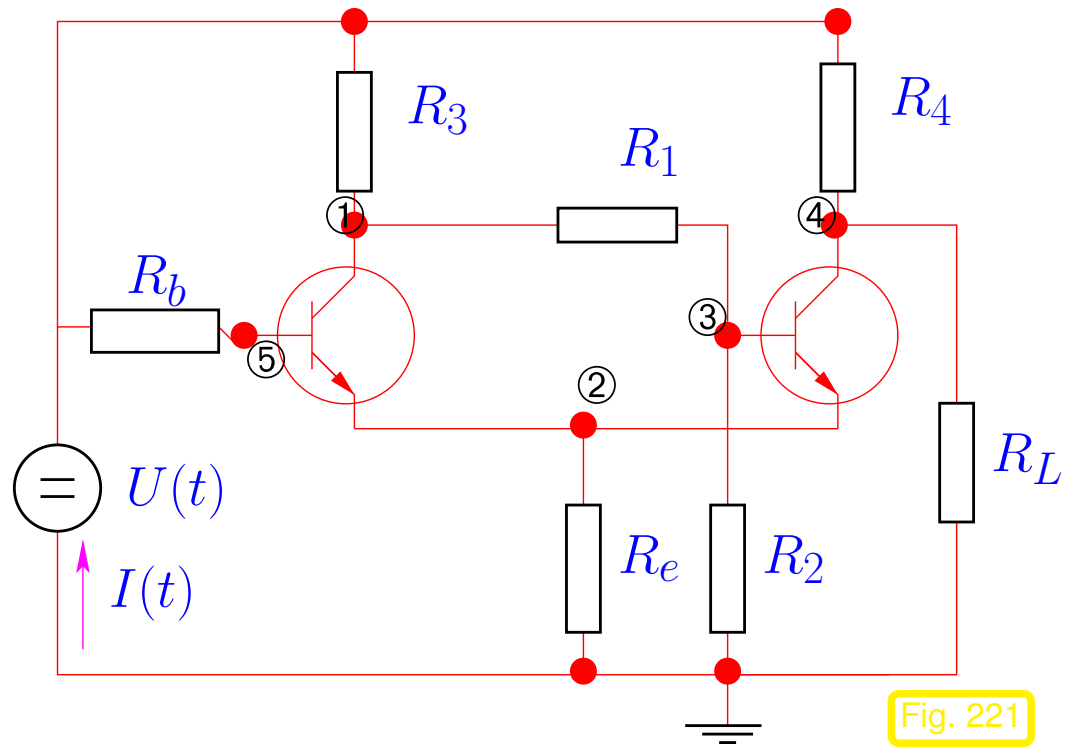
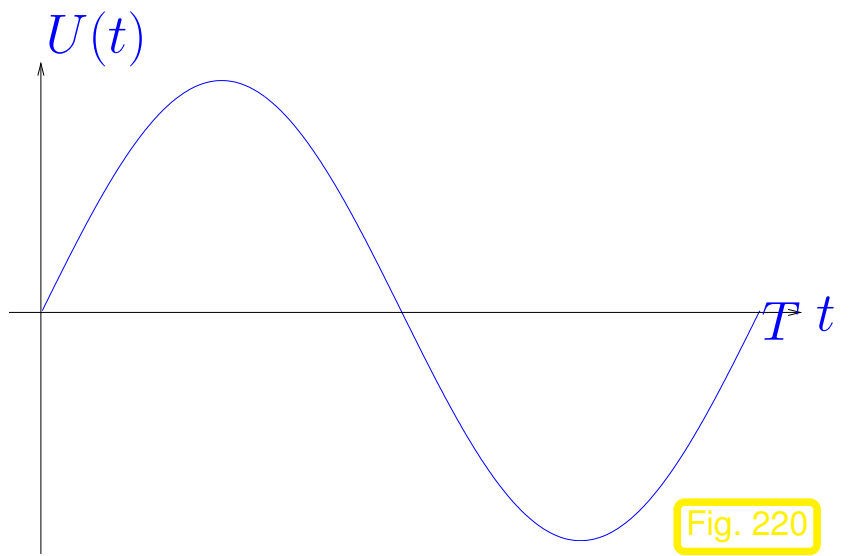
Numerical quadrature methods

*approximate*

$$\int_a^b f(t) dt$$

Example 10.0.2 (Heating production in electrical circuits).

Time-harmonic excitation:



Integrating power  $P = UI$  over period  $[0, T]$  yields heat production per period:

$$W_{\text{therm}} = \int_0^T U(t)I(t) dt, \text{ where } I = I(U).$$

function  $I = \text{current}(U)$  involves solving non-linear system of equations, see Ex. 4.0.1!

# 10.1 Quadrature Formulas

$n$ -point **quadrature formula** on  $[a, b]$ :  
( $n$ -point quadrature rule)

$$\int_a^b f(t) dt \approx Q_n(f) := \sum_{j=1}^n w_j^n f(c_j^n). \quad (10.1.1)$$

$w_j^n$  : **quadrature weights**  $\in \mathbb{R}$  (ger.: Quadraturgewichte)  
 $c_j^n$  : **quadrature nodes**  $\in [a, b]$  (ger.: Quadraturknoten)

Note: (10.1.1) compatible with integrand  $f$  given in **procedural form** as `function y = f(x)`.

Code 10.1.2: MATLAB template implementing generic quadrature formula

```
1 function I = quadformula(f, c, w)
2 % Generic numerical quadrature routine implementing (10.1.1):
3 % f is a handle to a function of type f = @(x) ....
4 % c, w pass quadrature nodes  $c_j \in [a, b]$ , and weights  $w_j > 0$ 
5 n = length(c); If (length(w) ~= n), error(' #weights != #nodes ');
6 end
7 I = 0; for j=1:n, I = I + w(j)*f(c(j)); end
```

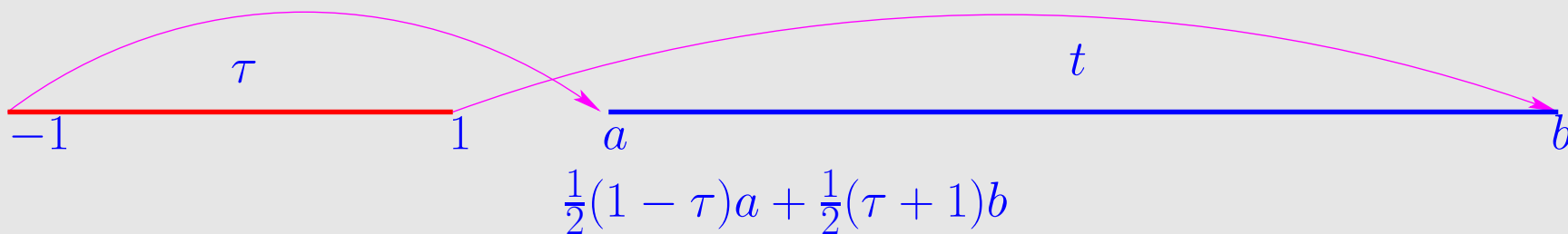
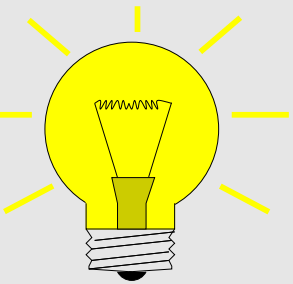


**Remark 10.1.3** (Transformation of quadrature rules).

Given: quadrature formula  $(\hat{c}_j, \hat{w}_j)_{j=1}^n$  on **reference interval**  $[-1, 1]$

Idea: transformation formula for integrals

$$\int_a^b f(t) dt = \frac{1}{2}(b-a) \int_{-1}^1 \hat{f}(\tau) d\tau, \quad \hat{f}(\tau) := f\left(\frac{1}{2}(1-\tau)a + \frac{1}{2}(\tau+1)b\right). \quad (10.1.4)$$



► quadrature formula for general interval  $[a, b]$ ,  $a, b \in \mathbb{R}$ :

$$\int_a^b f(t) dt \approx \frac{1}{2}(b-a) \sum_{j=1}^n \hat{w}_j \hat{f}(\hat{c}_j) = \sum_{j=1}^n w_j f(c_j) \quad \text{with} \quad \begin{aligned} c_j &= \frac{1}{2}(1-\hat{c}_j)a + \frac{1}{2}(1+\hat{c}_j)b, \\ w_j &= \frac{1}{2}(b-a)\hat{w}_j. \end{aligned}$$

▶ A 1D quadrature formula on arbitrary intervals can be specified by providing its weights  $\hat{w}_j$  /nodes  $\hat{c}_j$  for integration domain  $[-1, 1]$ . Then the above transformation is assumed.

Other common choice of reference interval:  $[0, 1]$



In general the quadrature formula (10.1.1) will only provide an approximate value for the integral.

➤ Inevitable for generic integrand:

quadrature error  $E(n) := \left| \int_a^b f(t) dt - Q_n(f) \right|$

Our focus (*cf.* interpolation error estimates, Sect. 9.1):

given families of quadrature rules  $\{Q_n\}_n$  described by

- quadrature weights  $\{w_j^n, j = 1, \dots, n\}_{n \in \mathbb{N}}$  and
- quadrature nodes  $\{c_j^n, j = 1, \dots, n\}_{n \in \mathbb{N}}$ , we

study the *asymptotic* behavior of the quadrature error  $E(n)$  for  $n \rightarrow \infty$

Qualitative distinction, see (9.1.3):

- ▷ algebraic convergence  $E(n) = O(n^{-p}), p > 0$
- ▷ exponential convergence  $E(n) = O(q^n), 0 \leq q < 1$

Note that the number  $n$  of nodes agrees with the number of  $f$ -evaluations required for evaluation of the quadrature formula. This is usually used as a *measure for the cost* of computing  $Q_n(f)$ .

Therefore we consider the quadrature error as a function of  $n$ .

# 10.2 Polynomial Quadrature Formulas [13, Sect. 10.2]

*Remark 10.2.1* (Approximation and quadrature).

There is a close relationship

function approximation (by interpolation)  $\longrightarrow$  numerical quadrature

because, once we know a good approximation of the integrand as a linear combination of basis functions,

$$f(t) \approx \sum_{j=1}^m \gamma_j b_j(t), \quad b_j(t) \hat{=} \text{basis functions, see (3.1.4),}$$

we can expect that

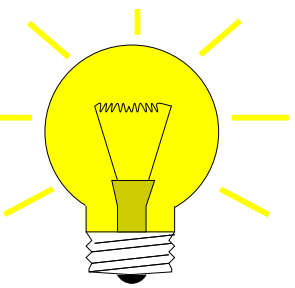
$$\sum_{j=1}^m \gamma_j \int_a^b b_j(t) dt \approx \int_a^b f(t) dt \quad (10.2.2)$$

provides a good approximation for the integral.

If the coefficients  $\gamma_j$  are computed relying on point evaluations, e.g. in the case of interpolation, then (10.2.2) boils down to a quadrature formula.

We elaborate this idea for function approximation by Lagrangian interpolation polynomials, see Sect. 3.3 and Sect. 9.1.

Idea: replace integrand  $f$  with  $p_{n-1} \in \mathcal{P}_{n-1}$  = polynomial interpolant of  $f$  for given interpolation nodes  $\{t_0, \dots, t_{n-1}\} \subset [a, b]$

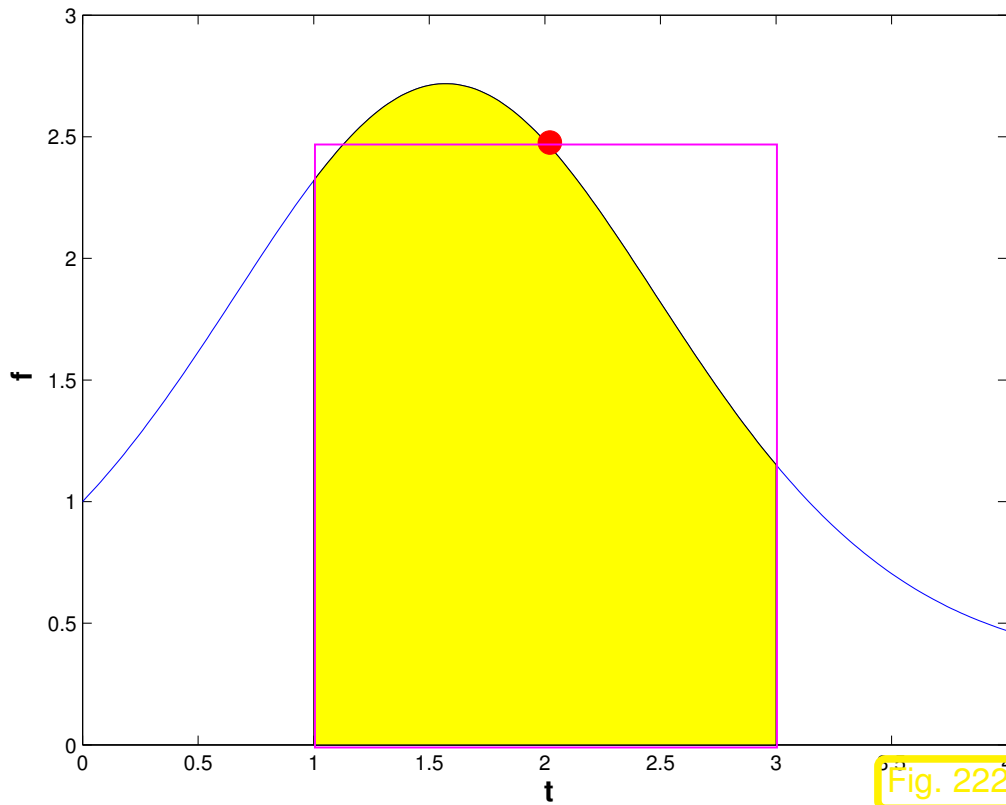


$$\blacktriangleright \int_a^b f(t) dt \approx Q_n(f) := \int_a^b p_{n-1}(t) dt . \quad (10.2.3)$$

Lagrange polynomials:  $L_i(t) := \prod_{\substack{j=0 \\ j \neq i}}^{n-1} \frac{t - t_j}{t_i - t_j}, \quad i = 0, \dots, n-1 \quad (3.3.4) \quad \blacktriangleright \quad p_{n-1}(t) = \sum_{i=0}^{n-1} f(t_i) L_i(t) .$

$$\int_a^b p_{n-1}(t) dt = \sum_{i=0}^{n-1} f(t_i) \int_a^b L_i(t) dt \quad \blacktriangleright \quad \begin{array}{l} \text{nodes } c_i = t_{i-1}, \\ \text{weights } w_i := \int_a^b L_{i-1}(t) dt. \end{array} \quad (10.2.4)$$

Example 10.2.5 (Midpoint rule).



1-point quadrature formula:

$$\int_a^b f(t) dt \approx Q_{\text{mp}}(f) = (b - a) f\left(\frac{1}{2}(a + b)\right) .$$

“midpoint”



Example 10.2.6 (Newton-Cotes formulas). [35, Ch. 38]

Equidistant quadrature nodes

$$t_j := a + hj, \quad h := \frac{b-a}{n}, \quad j = 0, \dots, n:$$

Symbolic computation of quadrature formulas on  $[0, 1]$  using MAPLE:

```
> newtoncotes := n -> factor(int(interp([seq(i/n, i=0..n)],
    [seq(f(i/n), i=0..n)], z), z=0..1)):
```

- $n = 1$ : Trapezoidal rule

```
> trapez := newtoncotes(1);
```

$$\hat{Q}_{\text{trp}}(f) := \frac{1}{2} (f(0) + f(1)) \quad (10.2.7)$$

$$\left( \int_a^b f(t) dt \approx \frac{b-a}{2} (f(a) + f(b)) \right)$$

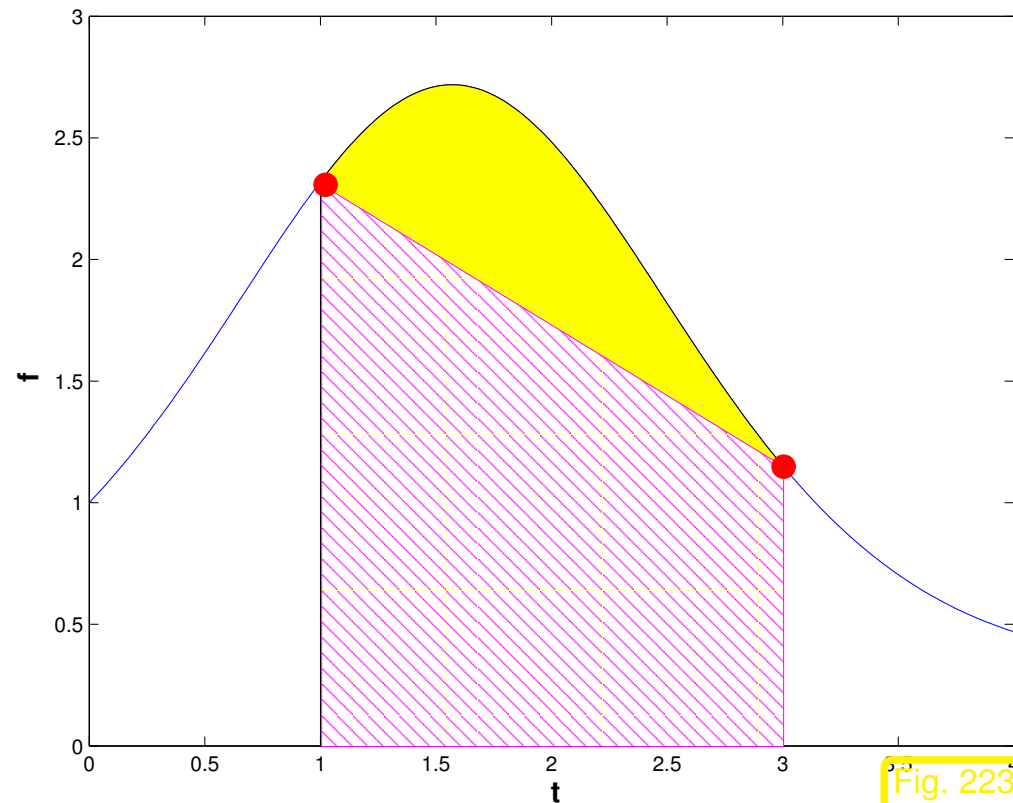


Fig. 223

- $n = 2$ : **Simpson rule**

```
> simpson := newtoncotes(2);
```

$$\frac{h}{6} \left( f(0) + 4f\left(\frac{1}{2}\right) + f(1) \right) \quad \left( \int_a^b f(t) dt \approx \frac{b-a}{6} \left( f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right) \right) \quad (10.2.8)$$

- $n = 4$ : **Milne rule**

```
> milne := newtoncotes(4);
```

$$\frac{1}{90} h \left( 7f(0) + 32f\left(\frac{1}{4}\right) + 12f\left(\frac{1}{2}\right) + 32f\left(\frac{3}{4}\right) + 7f(1) \right)$$

$$\left( \frac{b-a}{90} \left( 7f(a) + 32f\left(a + \frac{b-a}{4}\right) + 12f\left(a + \frac{b-a}{2}\right) + 32f\left(a + \frac{3(b-a)}{4}\right) + 7f(b) \right) \right)$$

- $n = 6$ : **Weddle rule**



```
> weddle := newtoncotes(6);
```

$$\frac{1}{840} h \left( 41 f(0) + 216 f\left(\frac{1}{6}\right) + 27 f\left(\frac{1}{3}\right) + 272 f\left(\frac{1}{2}\right) \right. \\ \left. + 27 f\left(\frac{2}{3}\right) + 216 f\left(\frac{5}{6}\right) + 41 f(1) \right)$$

- $n \geq 8$ : quadrature formulas with *negative* weights

```
> newtoncotes(8);
```

$$\frac{1}{28350} h \left( 989 f(0) + 5888 f\left(\frac{1}{8}\right) - 928 f\left(\frac{1}{4}\right) + 10496 f\left(\frac{3}{8}\right) \right. \\ \left. - 4540 f\left(\frac{1}{2}\right) + 10496 f\left(\frac{5}{8}\right) - 928 f\left(\frac{3}{4}\right) + 5888 f\left(\frac{7}{8}\right) + 989 f(1) \right)$$

Negative weights compromise numerical stability ( $\rightarrow$  Def. 2.5.11) !

Alternative: If  $t_j =$  Chebychev nodes (9.2.8)  $\blacktriangleright$  Clenshaw-Curtis rule

*Remark 10.2.9* (Error estimates for polynomial quadrature).

Quadrature error estimates directly from  $L^\infty$ -interpolation error estimates for Lagrangian interpolation with polynomial of degree  $n - 1$ , see Thm. 9.1.7:

$$f \in C^n([a, b]) \Rightarrow \left| \int_a^b f(t) dt - Q_n(f) \right| \leq \frac{1}{n!} (b - a)^{n+1} \|f^{(n)}\|_{L^\infty([a, b])}. \quad (10.2.10)$$

(More refined estimates for Clenshaw-Curtis rules, see (9.2.9), and, in particular, for analytic integrands, see Rem. 9.2.19.)

△

# 10.3 Composite Quadrature

Sect. 9.4.1: approximation by piecewise polynomial interpolants

Recall Rem. 10.2.1 ➤ piecewise polynomial approximation also gives rise to quadrature formulas:  
**composite quadrature formulas.**

Recall: piecewise polynomial interpolation is based on a grid/mesh

$$\mathcal{M} := \{a = x_0 < x_1 < \dots < x_{m-1} < x_m = b\} \quad ((9.4.1))$$

➤ composite quadrature formulas also rely on a mesh of the integration interval.

A motivation for composite quadrature, similar to the motivation for piecewise Lagrangian interpolation:

With  $a = x_0 < x_1 < \dots < x_{m-1} < x_m = b$

$$\int_a^b f(t) dt = \sum_{j=1}^m \int_{x_{j-1}}^{x_j} f(t) dt . \quad (10.3.1)$$

Recall (10.2.10): for polynomial quadrature rule (10.2.3) and  $f \in C^n([a, b])$  quadrature error shrinks with  $n + 1$ st power of length of integration interval.

► Reduction of quadrature error can be achieved by

- splitting of the integration interval according to (10.3.1),
- using the intended quadrature formula on each sub-interval  $[x_{j-1}, x_j]$ .

Note: Increase in total no. of  $f$ -evaluations incurred, which has to be balanced with the gain in accuracy to achieve optimal efficiency, *cf.* Sect. 4.3.3 and Sect. 10.5 for algorithmic realization.

Idea: • Partition integration domain  $[a, b]$  by **mesh** (grid,  $\rightarrow$  Sect.9.4)

$$\mathcal{M} := \{a = x_0 < x_1 < \dots < x_m = b\}$$

- Apply quadrature formulas from Sects. 10.2, 10.4 **locally** on mesh intervals

$I_j := [x_{j-1}, x_j], j = 1, \dots, m$ , and sum up.

**composite quadrature rule**

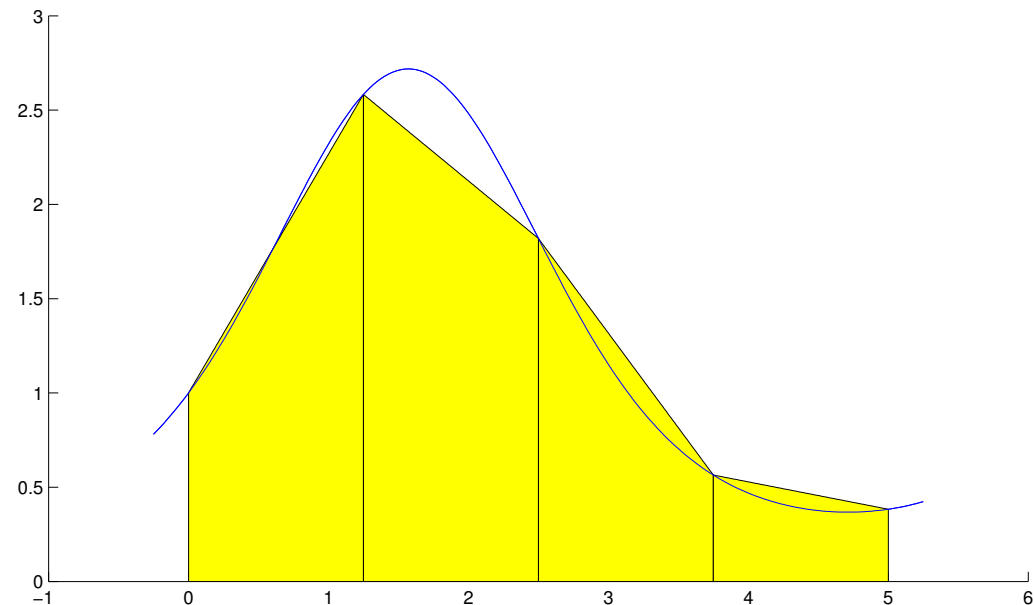
Analogy: global polynomial interpolation  $\longleftrightarrow$  piecewise polynomial interpolation  
( $\rightarrow$  Sect. 9.4)

Note: Here we only consider one and the same quadrature formula (**local quadrature formula**) applied on all sub-intervals.

*Example 10.3.2* (Simple composite polynomial quadrature rules).

Composite trapezoidal rule, *cf.* (10.2.7)

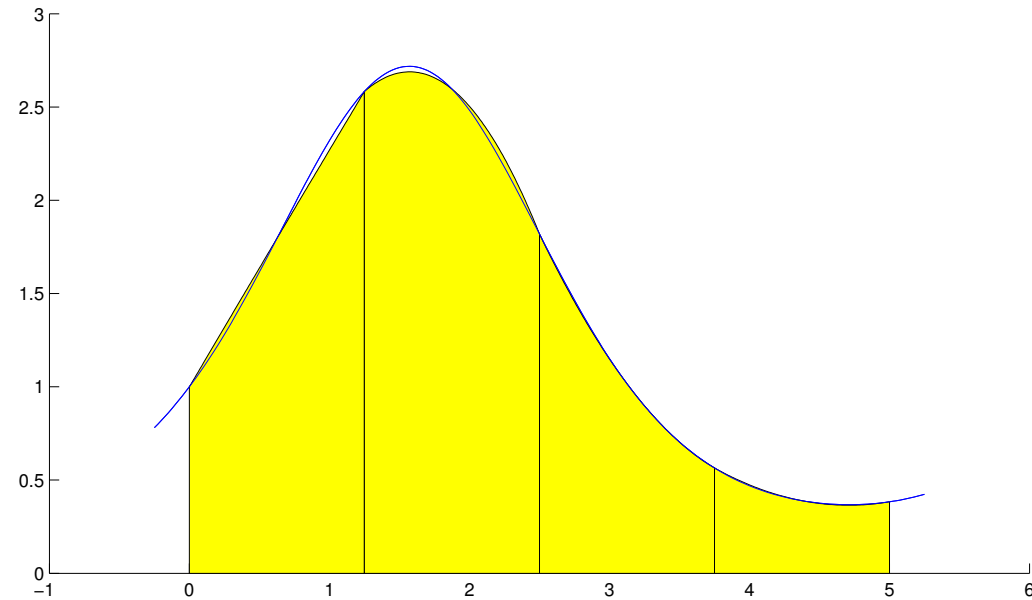
$$\int_a^b f(t) dt = \frac{1}{2}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \frac{1}{2}(x_{j+1} - x_{j-1})f(x_j) + \frac{1}{2}(x_m - x_{m-1})f(b) . \quad (10.3.3)$$



➤ arising from piecewise linear interpolation of  $f$ .

Composite Simpson rule, *cf.* (10.2.8)

$$\int_a^b f(t) dt = \frac{1}{6}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \frac{1}{6}(x_{j+1} - x_{j-1})f(x_j) + \sum_{j=1}^m \frac{2}{3}(x_j - x_{j-1})f\left(\frac{1}{2}(x_j + x_{j-1})\right) + \frac{1}{6}(x_m - x_{m-1})f(b). \quad (10.3.4)$$



➤ related to piecewise quadratic Lagrangian interpolation.

Formulas (10.3.3), (10.3.4) directly suggest efficient implementation with minimal number of  $f$ -evaluations.

How to rate the “quality” of a composite quadrature formula ?

Clear: It is impossible to predict the quadrature error, unless the integrand is known.

Possible:

*asymptotic perspective*

Predict decay of quadrature error as  $m \rightarrow \infty$  (asymptotic perspective) for certain classes of integrands and “uniform” meshes.

This asymptotic perspective will be adopted in the following numerical experiments, which study the decay of the quadrature error for

- fixed integrand  $f$ ,
- fixed local (polynomial) quadrature rule,
- for a *sequence* of ever finer equidistant grids.

*Example 10.3.5* (Quadrature errors for composite quadrature rules).

Composite quadrature rules based on

- trapezoidal rule (10.2.7)  $\triangleright$  local order 2 (exact for linear functions),
- Simpson rule (10.2.8)  $\triangleright$  local order 3 (exact for quadratic polynomials)

on equidistant mesh  $\mathcal{M} := \{jh\}_{j=0}^n$ ,  $h = 1/n$ ,  $n \in \mathbb{N}$ .

## Code 10.3.6: composite trapezoidal rule (10.3.3)

```

1 function res = trapezoidal(fnct, a, b, N)
2 % Numerical quadrature based on trapezoidal rule
3 % fnct handle to y = f(x)
4 % a, b bounds of integration interval
5 % N+1 = number of equidistant integration points (can be a vector)
6 res = [];
7 for n = N
8     h = (b-a)/n; x = (a:h:b); w = [0.5 ones(1, n-1) 0.5];
9     res = [res; h, h*dot(w, feval(fnct, x))];
0 end

```



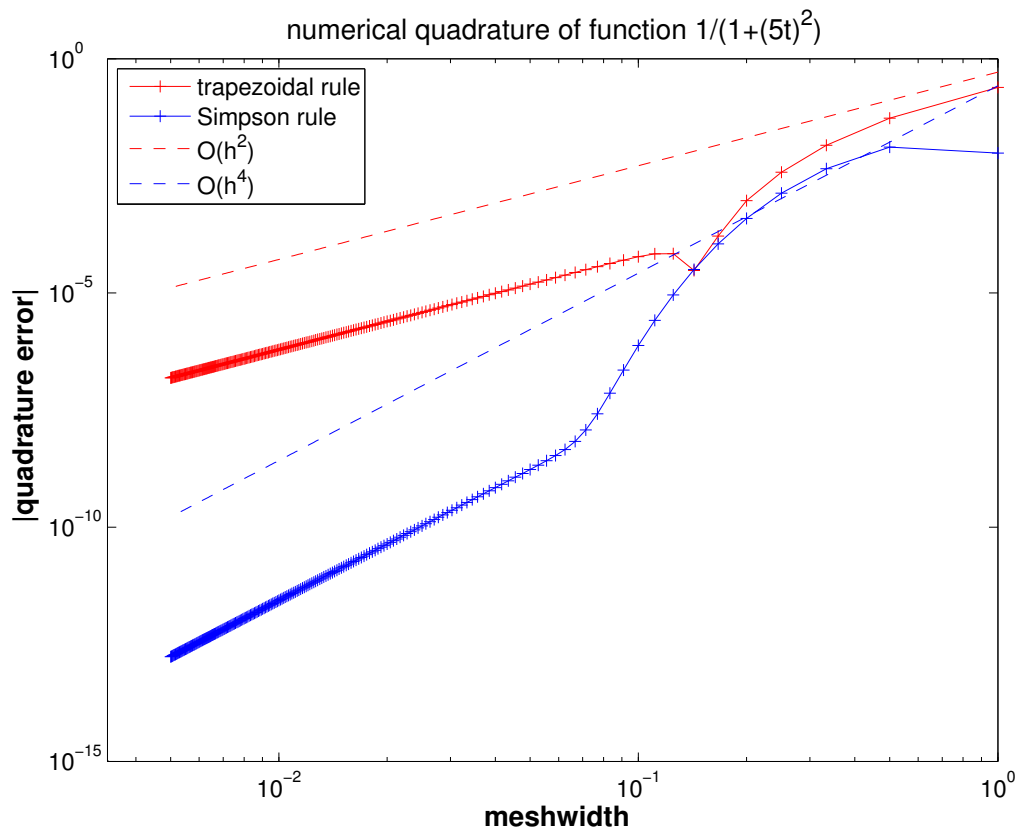
## Code 10.3.7: composite Simpson rule (10.3.4)

NumCSE,  
autumn  
2010

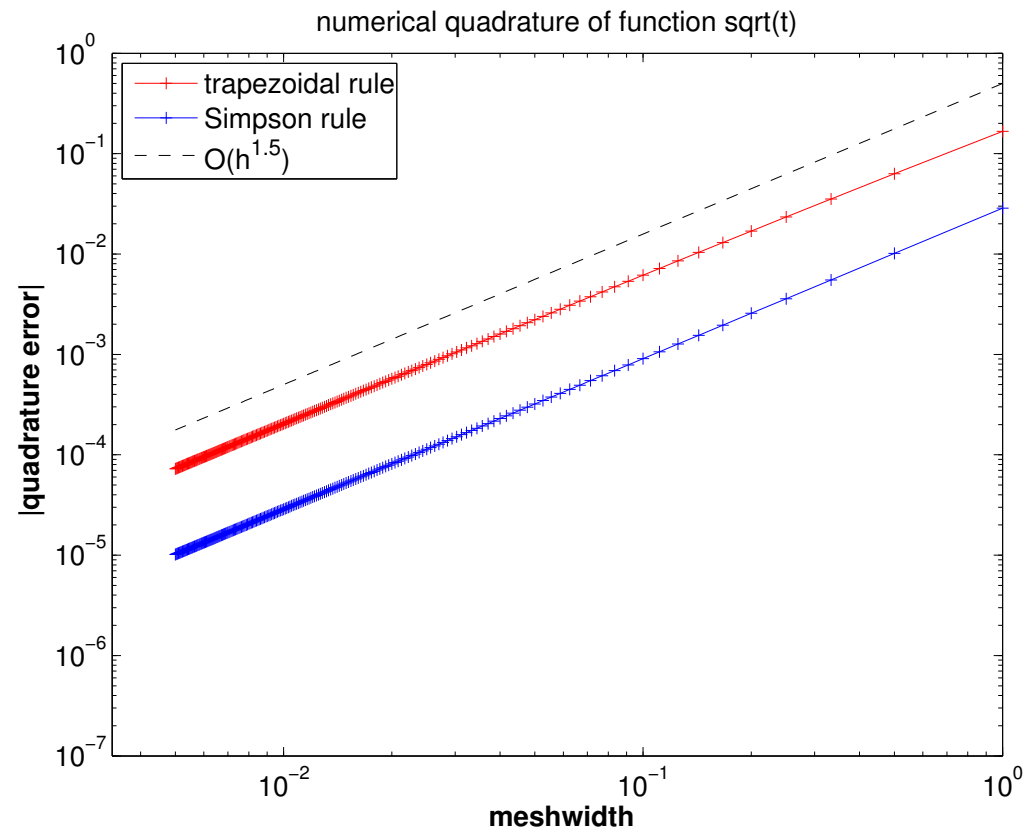
```
1 function res = simpson(fnct,a,b,N)
2 % Numerical quadrature based on Simpson rule
3 % fnct handle to y = f(x)
4 % a,b bounds of integration interval
5 % N+1 = number of equidistant integration points (can be a vector)
6
7 res = [];
8 for n = N
9     h = (b-a)/n;
0     x = (a:h/2:b);
1     fv = feval(fnct,x);
2     val = sum(h*(fv(1:2:end-2)+4*fv(2:2:end-1)+fv(3:2:end)))/6;
3     res = [res; h, val];
4 end
```

R. Hiptmair  
rev 38286,  
November  
8, 2011

Note: `fnct` is supposed to accept vector arguments and return the function value for each vector component!



quadrature error,  $f_1(t) := \frac{1}{1+(5t)^2}$  on  $[0, 1]$



quadrature error,  $f_2(t) := \sqrt{t}$  on  $[0, 1]$

Asymptotic behavior of quadrature error  $E(n) := \left| \int_0^1 f(t) dt - Q_n(f) \right|$  for meshwidth " $h \rightarrow 0$ "

• algebraic convergence  $E(n) = O(h^\alpha)$  of order  $\alpha > 0$ ,  $n = h^{-1}$

➤ Sufficiently smooth integrand  $f_1$ : trapezoidal rule  $\rightarrow \alpha = 2$ , Simpson rule  $\rightarrow \alpha = 4$  !?

➤ singular integrand  $f_2$ :  $\alpha = 3/2$  for trapezoidal rule & Simpson rule !

(lack of) smoothness of integrand limits convergence !

Composite Simpson rule: rate = 4 ? investigate *local* quadrature error on  $[0, h]$  with MAPLE

```
> rule := 1/3*h*(f(2*h)+4*f(h)+f(0))
> err := taylor(rule - int(f(x), x=0..2*h), h=0, 6);
```

$$err := \left( \frac{1}{90} \left( D^{(4)} \right) (f) (0) h^5 + O \left( h^6 \right), h, 6 \right)$$

➤ Composite Simpson rule converges with rate 4, indeed !

Code 10.3.8: errors of composite trapezoidal and Simpson rule

```
1 function comruleerrs()
2 % Numerical quadrature on [0,1]
3
4 figure ('Name', '1/(1+(5t)^2)');
```

```
5 exact = atan(5)/5;
6 trres = trapezoidal(inline('1./(1+(5*x).^2)'),0,1,1:200);
7 smres = simpson(inline('1./(1+(5*x).^2)'),0,1,1:200);
8 loglog(trres(:,1),abs(trres(:,2)-exact),'r+-',...
9         smres(:,1),abs(smres(:,2)-exact),'b+-',...
0         trres(:,1),trres(:,1).^2*(trres(1,2)/trres(1,1)^2),'r--',...
1         smres(:,1),smres(:,1).^4*(smres(1,2)/smres(1,1)^2),'b--');
2 set(gca,'fontsize',12);
3 title('numerical quadrature of function
4       1/(1+(5t)^2)','fontsize',14);
5 xlabel('\bf meshwidth','fontsize',14);
6 ylabel('\bf |quadrature error|','fontsize',14);
7 legend('trapezoidal rule','Simpson rule','O(h^2)','O(h^4)',2);
8 axis([1/300 1 10^(-15) 1]);
9 trp1 =
10     polyfit(log(trres(end-100:end,1)),log(abs(trres(end-100:end,2)-exact)),2);
11 smp1 =
12     polyfit(log(smres(end-100:end,1)),log(abs(smres(end-100:end,2)-exact)),2);
13 print -dpsc2 '../PICTURES/compruleerr1.eps';
14 figure('Name','sqrt(t)');
15 exact = 2/3;
16 trres = trapezoidal(inline('sqrt(x)'),0,1,1:200);
```

```
5 smres = simpson(inline('sqrt(x)'),0,1,1:200);
6 loglog (trres(:,1), abs(trres(:,2)-exact), 'r+-', ...
7         smres(:,1), abs(smres(:,2)-exact), 'b+-', ...
8         trres(:,1), trres(:,1).^(1.5)*(trres(1,2)/trres(1,1)^2), 'k--')
9 set (gca, 'fontsize', 14);
10 title ('numerical quadrature of function sqrt(t)', 'fontsize', 14);
11 xlabel ('\bf meshwidth', 'fontsize', 14);
12 ylabel ('\bf |quadrature error|', 'fontsize', 14);
13 legend ('trapezoidal rule', 'Simpson rule', 'O(h^{1.5})', 2);
14 axis ([1/300 1 10^(-7) 1]);
15 trp2 =
16     polyfit (log (trres (end-100:end, 1)), log (abs (trres (end-100:end, 2) - exact), 2);
17 smp2 =
18     polyfit (log (smres (end-100:end, 1)), log (abs (smres (end-100:end, 2) - exact), 2);
19 print -dpsc2 '../PICTURES/compruleerr2.eps';
```



Composite quadrature formulas based on a grid  $\mathcal{M} := \{a = x_0 < x_1 < \dots < x_m = b\}$  and a single local quadrature formula (usually) display *algebraic convergence*  $O(h^p)$  of the quadrature error in terms of the meshwidth  $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}|$ .

Is it possible to predict the rate of algebraic convergence (for *sufficiently smooth*) integrands?

**YES**, and a simple criterion for the local quadrature rule will do the job!

Gauge for “quality” of a quadrature formula  $Q_n$ :

$$\text{Order}(Q_n) := \max\{n \in \mathbb{N}_0: Q_n(p) = \int_a^b p(t) dt \quad \forall p \in \mathcal{P}_n\} + 1$$

By construction: polynomial quadrature formulas (10.2.3) exact for  $f \in \mathcal{P}_{n-1}$   
 $\Rightarrow$   $n$ -point polynomial quadrature formula has **at least** order  $n$

*Remark 10.3.9* (Orders of simple polynomial quadrature formulas).

$n$		Order
0	midpoint rule	2
1	trapezoidal rule (10.2.7)	2
2	Simpson rule (10.2.8)	4
3	$\frac{3}{8}$ -rule	4
4	Milne rule	6



Focus: *asymptotic* behavior of quadrature error for

mesh width  $h := \max_{j=1,\dots,m} |x_j - x_{j-1}| \rightarrow 0$

For *fixed* local  $n$ -point quadrature rule:  $O(mn)$   $f$ -evaluations for composite quadrature (“total cost”)

➤ If mesh equidistant ( $|x_j - x_{j-1}| = h$  for all  $j$ ), then total cost for composite numerical quadrature =  $O(h^{-1})$ .

**Theorem 10.3.10** (Convergence of composite quadrature formulas).

For a composite quadrature formula  $Q$  based on a local quadrature formula of order  $p \in \mathbb{N}$  holds

$$\exists C > 0: \left| \int_I f(t) dt - Q(f) \right| \leq Ch^p \left\| f^{(p)} \right\|_{L^\infty(I)} \quad \forall f \in C^p(I), \forall \mathcal{M}.$$

*Proof.* Let  $s$  be the *piecewise polynomial interpolant* of local degree  $p - 1$  of  $f$  on the mesh  $\mathcal{M}$ , which relies on the local quadrature nodes ( $p$  for each interval of the mesh) as interpolation points, Sect. 9.4.1 and (9.4.2) for details.

▶  $Q(f) = q(s)$ , because  $s$  and  $f$  agree in all quadrature nodes,



Order  $p$  of local quadrature formulas  $\triangleright$  local quadrature formulas will yield exact evaluation of

$$\int_{x_{j-1}^{x_j}} s(t) dt!$$

$$\Rightarrow Q(s) = \int_a^b s(t) dt$$

$$\Rightarrow \left| \int_a^b f(t) dt - Q(f) \right| = \left| \int_a^b f(t) dt - \int_a^b s(t) dt + \underbrace{Q(s) - Q(f)}_{=0} \right| \leq (b-a) \|f - s\|_{L^\infty([a,b])} \cdot$$

Then apply the interpolation error estimate (9.4.6) for piecewise polynomial interpolation.  $\square$

Again essential, *cf.* Sect. 9.1: sufficient **smoothness** of integrand required to achieve algebraic convergence in  $h$  as predicted by order of local quadrature rule.

*Remark 10.3.11* (Removing a singularity by transformation).

Ex. 10.3.5  $\triangleright$  lack of smoothness of integrand limits rate of algebraic convergence of composite quadrature rule for meshwidth  $h \rightarrow 0$ .

Idea: recover integral with smooth integrand by “analytic preprocessing”

Here is an example:

For  $f \in C^\infty([0, b])$  compute  $\int_0^b \sqrt{t} f(t) dt$  via quadrature rule ( $\rightarrow$  Ex. 10.3.5)

substitution  $s = \sqrt{t}$ : 
$$\int_0^b \sqrt{t} f(t) dt = \int_0^{\sqrt{b}} \boxed{2s^2 f(s^2)} ds. \quad (10.3.12)$$

Then: Apply quadrature rule to smooth integrand



Sometimes there are surprises: convergence of a composite quadrature rule is much better than predicted by the order of the local quadrature formula, see [38] for an explanation.

Equidistant trapezoidal rule (order 2), see (10.3.3)

$$\int_a^b f(t) dt \approx T_m(f) := h \left( \frac{1}{2}f(a) + \sum_{k=1}^{m-1} f(kh) + \frac{1}{2}f(b) \right), \quad h := \frac{b-a}{m}. \quad (10.3.14)$$

### Code 10.3.15: equidistant trapezoidal quadrature formula

```

1 function res = trapezoidal(fnct,a,b,N)
2 % Numerical quadrature based on trapezoidal rule
3 % fnct handle to y = f(x)
4 % a,b bounds of integration interval
5 % N+1 = number of equidistant integration points (can be a vector)
6 res = [];
7 for n = N
8     h = (b-a)/n;   x = (a:h:b); w = [0.5 ones(1,n-1) 0.5];

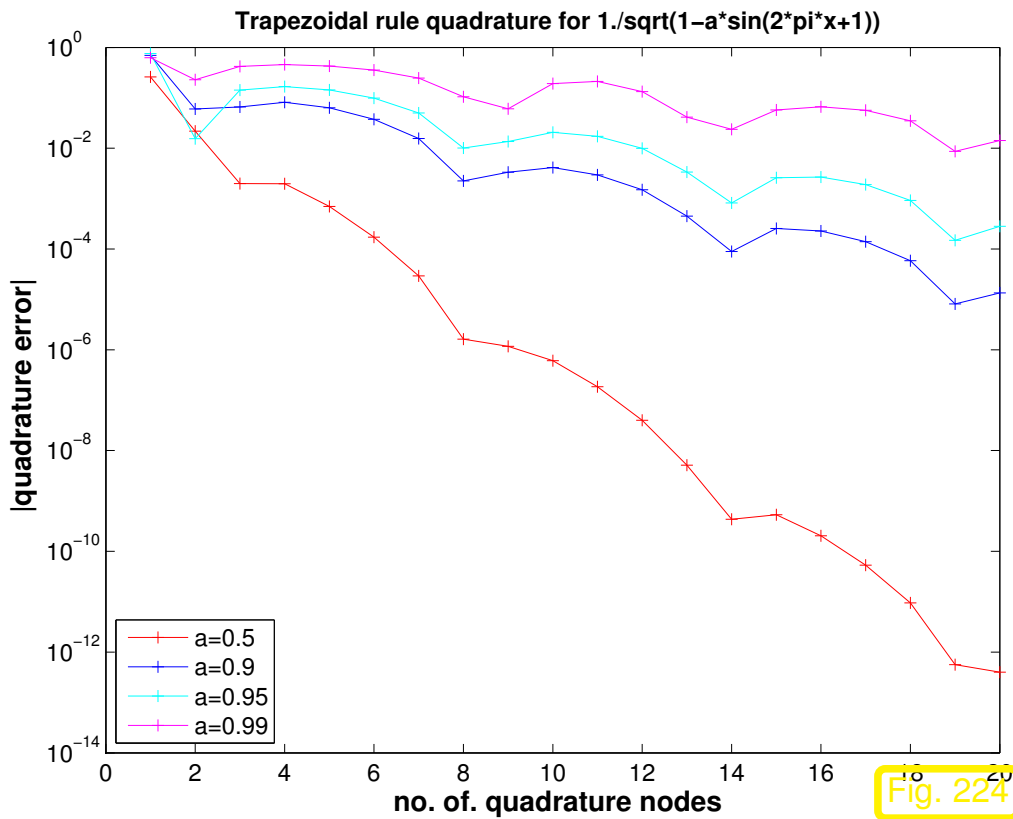
```

```
9 | res = [res; h, h*dot(w, feval(fnct,x))];  
0 | end
```

1-periodic smooth (**analytic**) integrand

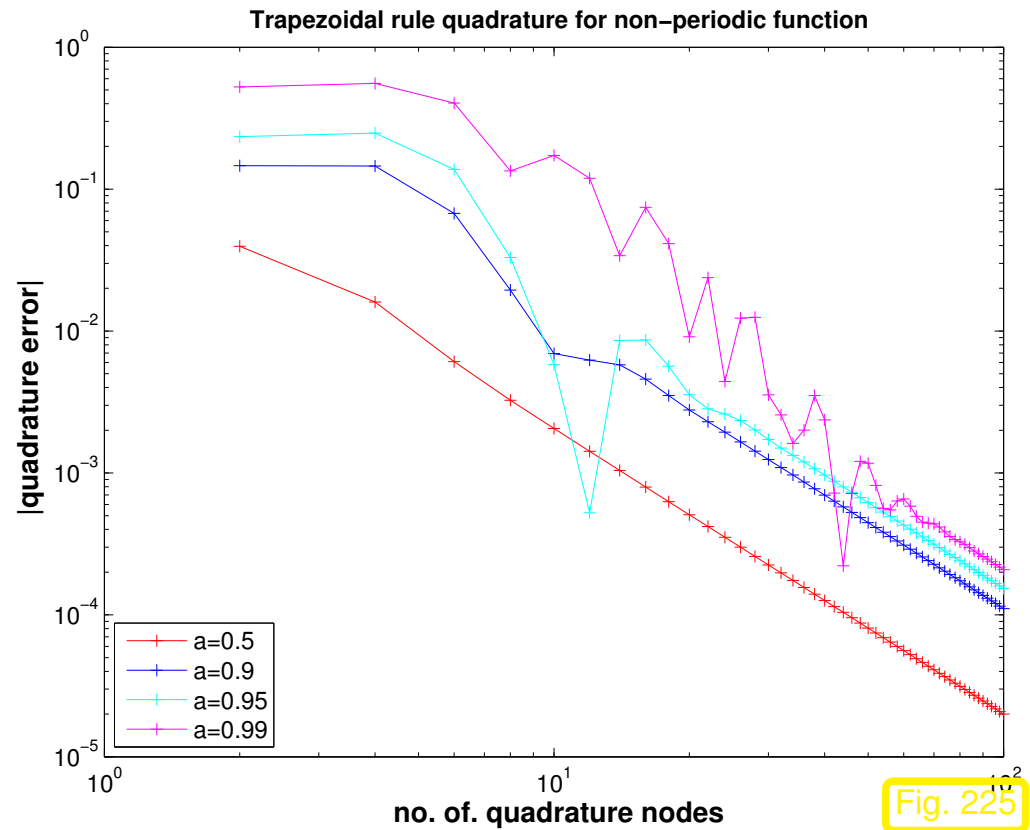
$$f(t) = \frac{1}{\sqrt{1 - a \sin(2\pi t - 1)}}, \quad 0 < a < 1.$$

(“exact value of integral”: use  $T_{500}$ )



quadrature error for  $T_n(f)$  on  $[0, 1]$

exponential convergence !!



quadrature error for  $T_n(f)$  on  $[0, \frac{1}{2}]$

merely algebraic convergence

Code 10.3.16: tracking error of equidistant trapezoidal quadrature formula

```

1 function traperr()
2
3 clear a;

```

```
4 global a;  
5 l = 0; r = 0.5; % integration interval  
6 N = 50;  
7 a = 0.5; res05 = trapezoidal(@issin,l,r,1:N);  
8 ex05 = trapezoidal(@issin,l,r,500); ex05 = ex05(1,2);  
9 a = 0.9; res09 = trapezoidal(@issin,l,r,1:N);  
0 ex09 = trapezoidal(@issin,l,r,500); ex09 = ex09(1,2);  
1 a = 0.95; res95 = trapezoidal(@issin,l,r,1:N);  
2 ex95 = trapezoidal(@issin,l,r,500); ex95 = ex95(1,2);  
3 a = 0.99; res99 = trapezoidal(@issin,l,r,1:N);  
4 ex99 = trapezoidal(@issin,l,r,500); ex99 = ex99(1,2);  
5 figure ('name','trapezoidal rule for non-periodic function');  
6 loglog (1./res05(:,1), abs(res05(:,2)-ex05), 'r+-', ...  
7         1./res09(:,1), abs(res09(:,2)-ex09), 'b+-', ...  
8         1./res95(:,1), abs(res95(:,2)-ex95), 'c+-', ...  
9         1./res99(:,1), abs(res99(:,2)-ex99), 'm+-');  
0 set (gca, 'fontsize', 12);  
1 legend ('a=0.5', 'a=0.9', 'a=0.95', 'a=0.99', 3);  
2 xlabel ('\bf no. of. quadrature nodes', 'fontsize', 14);  
3 ylabel ('\bf |quadrature error|', 'fontsize', 14);  
4 title ('\bf Trapezoidal rule quadrature for non-periodic  
   function', 'fontsize', 12);
```

```
6 print -depsc2 '../PICTURES/traperr2.eps';
7
8 clear a;
9 global a;
10 l = 0; r = 1; % integration interval
11 N = 20;
12 a = 0.5; res05 = trapezoidal(@issin,l,r,1:N);
13 ex05 = trapezoidal(@issin,l,r,500); ex05 = ex05(1,2);
14 a = 0.9; res09 = trapezoidal(@issin,l,r,1:N);
15 ex09 = trapezoidal(@issin,l,r,500); ex09 = ex09(1,2);
16 a = 0.95; res95 = trapezoidal(@issin,l,r,1:N);
17 ex95 = trapezoidal(@issin,l,r,500); ex95 = ex95(1,2);
18 a = 0.99; res99 = trapezoidal(@issin,l,r,1:N);
19 ex99 = trapezoidal(@issin,l,r,500); ex99 = ex99(1,2);
20 figure ('name','trapezoidal rule for periodic function');
21 semilogy (1./res05(:,1), abs(res05(:,2)-ex05), 'r+-', ...
22           1./res09(:,1), abs(res09(:,2)-ex09), 'b+-', ...
23           1./res95(:,1), abs(res95(:,2)-ex95), 'c+-', ...
24           1./res99(:,1), abs(res99(:,2)-ex99), 'm+-');
25 set (gca, 'fontsize', 12);
26 legend ('a=0.5', 'a=0.9', 'a=0.95', 'a=0.99', 3);
27 xlabel ('\bf no. of. quadrature nodes', 'fontsize', 14);
28 ylabel ('\bf |quadrature error|', 'fontsize', 14);
```

```

9 title ('{\bf Trapezoidal rule quadrature for
10 1./sqrt(1-a*sin(2*pi*x+1))}', 'fontsize', 12);
11 print -depsc2 '../PICTURES/traperr1.eps';

```

Explanation:

$$f(t) = e^{2\pi ikt} \quad \blacktriangleright \quad \begin{cases} \int_0^1 f(t) dt = \begin{cases} 0 & , \text{if } k \neq 0, \\ 1 & , \text{if } k = 0. \end{cases} \\ T_m(f) = \frac{1}{m} \sum_{l=0}^{m-1} e^{\frac{2\pi i}{m}lk} \stackrel{(8.2.5)}{=} \begin{cases} 0 & , \text{if } k \notin m\mathbb{Z}, \\ 1 & , \text{if } k \in m\mathbb{Z}. \end{cases} \end{cases}$$

Equidistant trapezoidal rule  $T_m$  is exact for trigonometric polynomials of degree  $< 2m$  !

It takes sophisticated tools from complex analysis to conclude exponential convergence for analytic integrands from the above observation.



*Remark 10.3.17* (Approximate computation of Fourier coefficients).

Recall from Sect. 8.2.5: recovery of signal  $(y_k)_{k \in \mathbb{Z}}$  from its Fourier transform  $c(t)$

$$y_j = \int_0^1 c(t) \exp(2\pi i j t) dt . \quad (8.2.60)$$

Task: approximate computation of  $y_j$


Recall:  $c(t)$  obtained from  $(y_k)_{k \in \mathbb{Z}}$  through Fourier series

$$c(t) = \sum_{k \in \mathbb{Z}} y_k \exp(-2\pi i k t) . \quad (8.2.48)$$

➤  $c(t)$  smooth & 1-periodic for finite/rapidly decaying  $(y_k)_{k \in \mathbb{Z}}$ .

Ex. 10.3.13 

use **equidistant trapezoidal rule** (10.3.14)  
for approximate evaluation of integral in (8.2.60).

 Boils down to inverse DFT (8.2.16); hardly surprising in light of the derivation of (8.2.60) in Sect. 8.2.5.

## Code 10.3.18: DFT-based approximate computation of Fourier coefficients

NumCSE,  
autumn  
2010

```

1 function y = fourcoeffcomp(c,m,ovsmp1)
2 % Compute the Fourier coefficients  $y_{-m}, \dots, y_m$  of the function
3 %  $c: [0,1[ \rightarrow \mathbb{C}$  using an oversampling factor ovsmp1.
4 % c must be a handle to a function  $\varrho(t)$ , which accepts row
5 % vector arguments
6 if (nargin < 3), ovsmp1 = 2; else ovsmp1 = ceil(ovsmp1); end
7 N = (2*m+1)*ovsmp1; h = 1/N; % Number of quadrature points
8 % (Inverse) discrete Fourier transform
9 y = ifft(c(0:h:1-h));
0 % Undo oversampling and wrapping of Fourier coefficient array
1 y = [y(N-m+1:N), y(1:m+1)];

```

R. Hiptmair  
rev 38286,  
November  
8, 2011

*Remark 10.3.19* (Choice of (local) quadrature weights).

So far we have constructed quadrature formulas from (piecewise) polynomial interpolation  $\rightarrow$  (local) Newton-Cotes formulas from Ex. 10.2.6. The degree of the Lagrange interpolation polynomials determined the order of the quadrature formula.

10.3  
p. 1060

Question: given arbitrary quadrature nodes, how can be achieve a certain order by choosing the weights appropriately?

Given: arbitrary nodes  $\xi_1, \dots, \xi_n$  for  $n$ -point (local) quadrature formula on  $[a, b]$

Take cue from polynomial quadrature formulas: choice of weights  $\omega_j$  according to (10.2.4) ensures order  $\geq n$ .

There is a more direct way without detour via Lagrange polynomials:

If  $p_0, \dots, p_{n-1}$  is a basis of  $\mathcal{P}_n$ , then, thanks to the linearity of the integral and quadrature formulas,

$$Q_n(p_j) = \int_a^b p_j(t) dt \quad \forall j = 0, \dots, n-1 \Leftrightarrow Q_n \text{ has order } \geq n. \quad (10.3.20)$$

➤  $n \times n$  linear system of equations, see (10.4.2) for an example:

$$\begin{pmatrix} p_0(\xi_1) & \cdots & p_0(\xi_n) \\ \vdots & & \vdots \\ p_{n-1}(\xi_1) & \cdots & p_{n-1}(\xi_n) \end{pmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_n \end{pmatrix} = \begin{pmatrix} \int_a^b p_0(t) dt \\ \vdots \\ \int_a^b p_{n-1}(t) dt \end{pmatrix}. \quad (10.3.21)$$

For instance, for the computation of quadrature weights, one may choose the monomial basis  $p_j(t) = t^j$ .



Natural question: What is the maximal order for an  $n$ -point quadrature formula ?

**Lemma 10.3.22** (Bound for order of quadrature formula).

*There is no  $n$ -point quadrature formula of order  $2n + 1$*

*Proof.* (indirect) Assume there was an  $n$ -point quadrature formula with nodes  $a \leq \xi_1 < \xi_2 < \dots < \xi_n \leq b$  of order  $2n + 1$ .

► Construct polynomial  $p(t) := \prod_{j=1}^n (t - \xi_j)^2 \in \mathcal{P}_{2n}$

$$\blacktriangleright \quad Q_n(p) = 0 \quad \text{but} \quad \int_a^b p(t) dt > 0 .$$

Thus, the assumption leads to a contradiction. □

## 10.4 Gauss Quadrature [35, Ch. 40-41], [13, Sect.10.3]

Natural question: Are there  $n$ -point quadrature formulas of maximal order  $2n$  ?

Heuristics: A quadrature formula has order  $m \in \mathbb{N}$  already, if it is exact for  $m$  polynomials  $\in \mathcal{P}_{m-1}$  that form a basis of  $\mathcal{P}_{m-1}$  (recall Thm. 3.2.2).



An  $n$ -point quadrature formula has  $2n$  “degrees of freedom” ( $n$  node positions,  $n$  weights).

“No. of equations = No. of unknowns”

*Example 10.4.1* (2-point quadrature rule of order 4).

Necessary & sufficient conditions for order 4, cf. (10.3.21):

$$Q_n(p) = \int_a^b p(t) dt \quad \forall p \in \mathcal{P}_3 \quad \Leftrightarrow \quad Q_n(t^q) = \frac{1}{q+1}(b^{q+1} - a^{q+1}), \quad q = 0, 1, 2, 3.$$



4 equations for weights  $\omega_j$  and nodes  $\xi_j$ ,  $j = 1, 2$  ( $a = -1, b = 1$ ), cf. Rem. 10.3.19

$$\begin{aligned} \int_{-1}^1 1 dt = 2 = 1\omega_1 + 1\omega_2, & \quad \int_{-1}^1 t dt = 0 = \xi_1\omega_1 + \xi_2\omega_2 \\ \int_{-1}^1 t^2 dt = \frac{2}{3} = \xi_1^2\omega_1 + \xi_2^2\omega_2, & \quad \int_{-1}^1 t^3 dt = 0 = \xi_1^3\omega_1 + \xi_2^3\omega_2. \end{aligned} \tag{10.4.2}$$

## Solve using MAPLE:

```
> eqns := seq(int(x^k, x=-1..1) = w[1]*xi[1]^k+w[2]*xi[2]^k, k=0..3);  
> sols := solve(eqns, indets(eqns, name));  
> convert(sols, radical);
```

➤ weights & nodes:  $\left\{ \omega_2 = 1, \omega_1 = 1, \xi_1 = 1/3\sqrt{3}, \xi_2 = -1/3\sqrt{3} \right\}$

▶ quadrature formula:  $\int_{-1}^1 f(x) dx \approx f\left(\frac{1}{\sqrt{3}}\right) + f\left(-\frac{1}{\sqrt{3}}\right)$  (10.4.3)



Optimist's **assumption**:  $\exists$  **family** of  $n$ -point quadrature formulas

$$Q_n(f) := \sum_{j=1}^n \omega_j^n f(\xi_j^n) \approx \int_{-1}^1 f(t) dt, \quad n \in \mathbb{N},$$

of order  $2n \iff$  exact for polynomials  $\in \mathcal{P}_{2n-1}$ . (10.4.4)

Define  $\bar{P}_n(t) := (t - \xi_1^n) \cdots (t - \xi_n^n), \quad t \in \mathbb{R} \implies \bar{P}_n \in \mathcal{P}_n$ .

Note:  $\bar{P}_n$  has leading coefficient = 1.

By assumption on the order of  $Q_n$ : for any  $q \in \mathcal{P}_{n-1}$

$$\int_{-1}^1 \underbrace{q(t)\bar{P}_n(t)}_{\in \mathcal{P}_{2n-1}} dt \stackrel{(10.4.4)}{=} \sum_{j=1}^n \omega_j^n q(\xi_j^n) \underbrace{\bar{P}_n(\xi_j^n)}_{=0} = 0.$$

$$\implies \text{orthogonality} \quad \int_{-1}^1 q(t)\bar{P}_n(t) dt = 0 \quad \forall q \in \mathcal{P}_{n-1}. \quad (10.4.5)$$

$L^2([-1, 1])$ -inner product of  $q$  and  $\bar{P}_n$

Recall:  $(f, g) \mapsto \int_a^b f(t)g(t) dt$  is an inner product on  $C^0([a, b])$ , the  $L^2$ -inner product, see [48, Sect. 4.4, Ex. 2], [27, Ex. 6.5]



- Treat space of polynomials  $\mathcal{P}_n$  as a vector space equipped with an inner product.
- Abstract techniques for vector spaces with inner product can be applied to polynomials, for instance **Gram-Schmidt orthogonalization**, cf. (5.2.11), [48, Thm. 4.8], [27, Alg. 6.1].

Abstract Gram-Schmidt orthogonalization: in a vector space with inner product  $\cdot$  orthogonal vectors  $q_0, q_1, \dots$  spanning the same subspaces as the linearly independent vectors  $v_0, v_1, \dots$  are constructed recursively via

$$q_{n+1} := v_{n+1} - \sum_{k=0}^n \frac{v_{n+1} \cdot q_k}{q_k \cdot q_k} q_k \quad , \quad q_0 := v_0 \quad . \quad (10.4.6)$$

- Construction of  $\bar{P}_n$  by **Gram-Schmidt orthogonalization** of monomial basis  $\{1, t, t^2, \dots, t^{n-1}\}$  of  $\mathcal{P}_{n-1}$  w.r.t.  $L^2([-1, 1])$ -inner product:

$$\bar{P}_0(t) := 1 \quad , \quad \bar{P}_{n+1}(t) = t^n - \sum_{k=0}^n \frac{\int_{-1}^1 t^n \bar{P}_k(t) dt}{\int_{-1}^1 \bar{P}_k^2(t) dt} \cdot \bar{P}_k(t) \quad (10.4.7)$$

The considerations so far only reveal constraints on the nodes of an  $n$ -point quadrature rule of order  $2n$ .

They do by no means confirm the existence of such rules, but offer a clear hint on how to construct them:

**Theorem 10.4.8** (Existence of  $n$ -point quadrature formulas of order  $2n$ ).

Let  $\{\bar{P}_n\}_{n \in \mathbb{N}_0}$  be a family of non-zero polynomials that satisfies

- $\bar{P}_n \in \mathcal{P}_n$ ,
- $\int_{-1}^1 q(t) \bar{P}_n(t) dt = 0$  for all  $q \in \mathcal{P}_{n-1}$  ( $L^2([-1, 1])$ -*orthogonality*),
- The set  $\{\xi_j^n\}_{j=1}^m$ ,  $m \leq n$ , of real zeros of  $\bar{P}_n$  is contained in  $[-1, 1]$ .

Then

$$Q_n(f) := \sum_{j=1}^m \omega_j^n f(\xi_j^n)$$

with weights chosen according to Rem. 10.3.19 provides a quadrature formula of order  $2n$  on  $[-1, 1]$ .

*Proof.* Conclude from the orthogonality of the  $\bar{P}_n$  that  $\{\bar{P}_k\}_{k=0}^n$  is a basis of  $\mathcal{P}_n$  and

$$\int_{-1}^1 h(t) \bar{P}_n(t) dt = 0 \quad \forall h \in \mathcal{P}_{n-1}. \quad (10.4.9)$$

Recall division of polynomials with remainder (Euclid's algorithm  $\rightarrow$  Course "Diskrete Mathematik"):  
for any  $p \in \mathcal{P}_{2n-1}$

$$p(t) = h(t)\bar{P}_n(t) + r(t), \quad \text{for some } h \in \mathcal{P}_{n-1}, r \in \mathcal{P}_{n-1}. \quad (10.4.10)$$

Apply this representation to the integral:

$$\int_{-1}^1 p(t) dt = \underbrace{\int_{-1}^1 h(t)\bar{P}_n(t) dt}_{=0 \text{ by (10.4.9)}} + \int_{-1}^1 r(t) dt \stackrel{(*)}{=} \sum_{j=1}^m \omega_j^n r(\xi_j^n), \quad (10.4.11)$$

(\*) : by choice of weights according to Rem. 10.3.19  $Q_n$  is exact for polynomials of degree  $\leq n-1$ !

By choice of nodes as zeros of  $\bar{P}_n$  using (10.4.9):

$$\sum_{j=1}^m \omega_j^n p(\xi_j^n) \stackrel{(10.4.10)}{=} \sum_{j=1}^m \omega_j^n h(\xi_j^n) \underbrace{\bar{P}_n(\xi_j^n)}_{=0} + \sum_{j=1}^m \omega_j^n r(\xi_j^n) \stackrel{(10.4.11)}{=} \int_{-1}^1 p(t) dt. \quad \square$$

The family of polynomials  $\{\bar{P}_n\}_{n \in \mathbb{N}_0}$  are so-called **orthogonal polynomials** w.r.t. the  $L^2([-1, 1])$ -inner product. They play a key role in analysis.

**Definition 10.4.12** (Legendre polynomials).  
The  $n$ -th **Legendre polynomial**  $P_n$  is defined by

- $P_n \in \mathcal{P}_n$ ,
- $\int_{-1}^1 P_n(t)q(t) dt = 0 \quad \forall q \in \mathcal{P}_{n-1}$ ,
- $P_n(1) = 1$ .

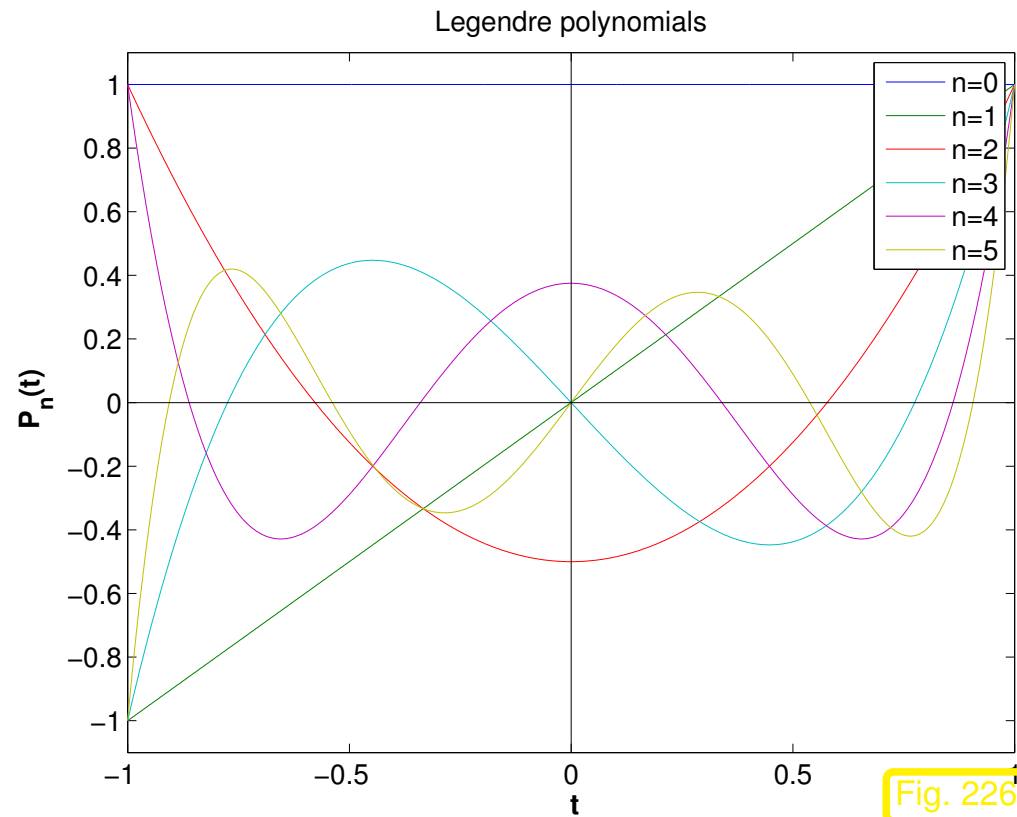


Fig. 226

Legendre polynomials  $P_0, \dots, P_5$



Notice: the polynomials  $\bar{P}_n$  defined by (10.4.7) and the Legendre polynomials  $P_n$  of Def. 10.4.12 (merely) differ by a constant factor!



Gauss points  $\xi_j^n$  = zeros of Legendre polynomial  $P_n$

Note: the above considerations, recall (10.4.5), show that the nodes of an  $n$ -point quadrature formula of order  $2n$  on  $[-1, 1]$  must agree with the zeros of  $L^2([-1, 1])$ -orthogonal polynomials.



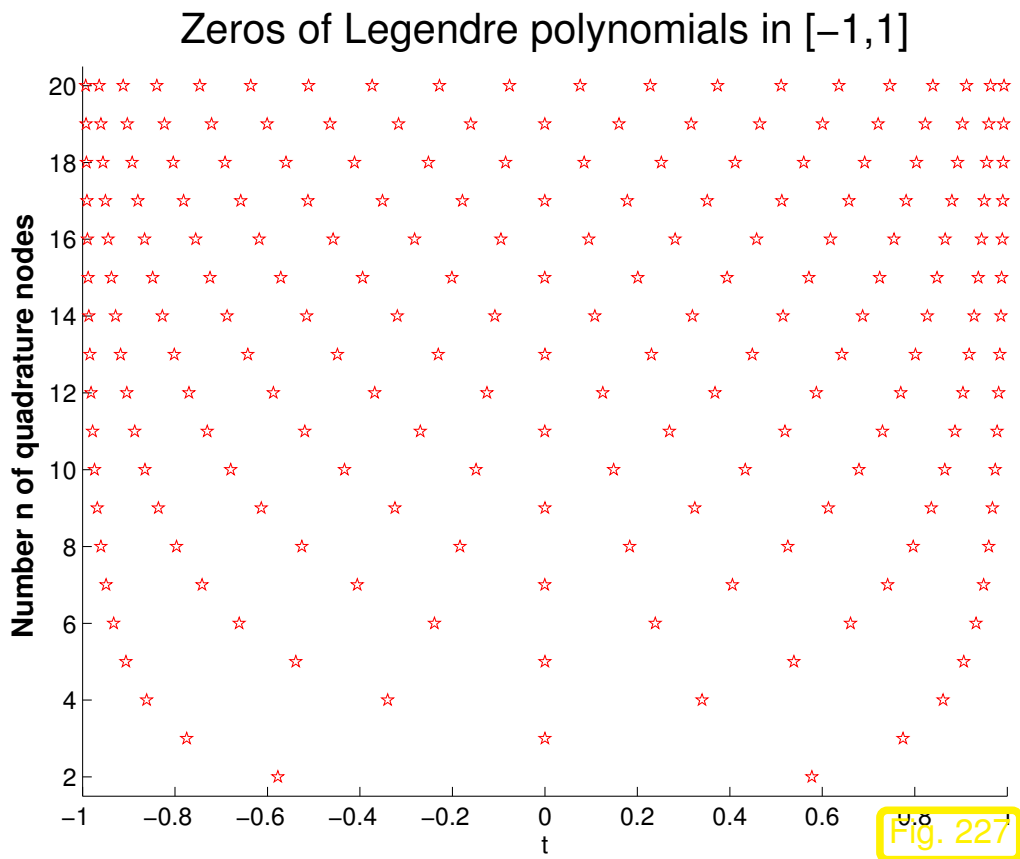
$n$ -point quadrature formulas of order  $2n$  are **unique**

This is not surprising in light of “ $2n$  equations for  $2n$  degrees of freedom”.



We are not done yet: the zeros of  $\bar{P}_n$  from (10.4.7) may lie outside  $[-1, 1]$ .  
In principle  $\bar{P}_n$  could also have less than  $n$  real zeros.

The next lemma shows that all this cannot happen.



◁ Obviously:

**Lemma 10.4.13** (Zeros of Legendre polynomials).  
 $P_n$  has  $n$  distinct zeros in  $] - 1, 1[$ .

Zeros of Legendre polynomials = Gauss points

*Proof.* (indirect) Assume that  $P_n$  has only  $m < n$  zeros  $\zeta_1, \dots, \zeta_m$  in  $] - 1, 1[$  at which it changes sign. Define

$$q(t) := \prod_{j=1}^m (t - \zeta_j) \Rightarrow qP_n \geq 0 \quad \text{or} \quad qP_n \leq 0 .$$

$$\Rightarrow \int_{-1}^1 q(t)P_n(t) dt \neq 0 .$$

As  $q \in \mathcal{P}_{n-1}$ , this contradicts (10.4.9). □

Quadrature formula from Thm. 10.4.8: **Gauss-Legendre quadrature**  
(nodes  $\xi_j^n$  = Gauss points)

Obviously ▷

**Lemma 10.4.14.** (*Positivity of Gauss-Legendre quadrature weights*)

*The weights of  
Gauss-Legendre quadrature formulas  
are positive.*

**Gauss-Legendre weights for [-1,1]**

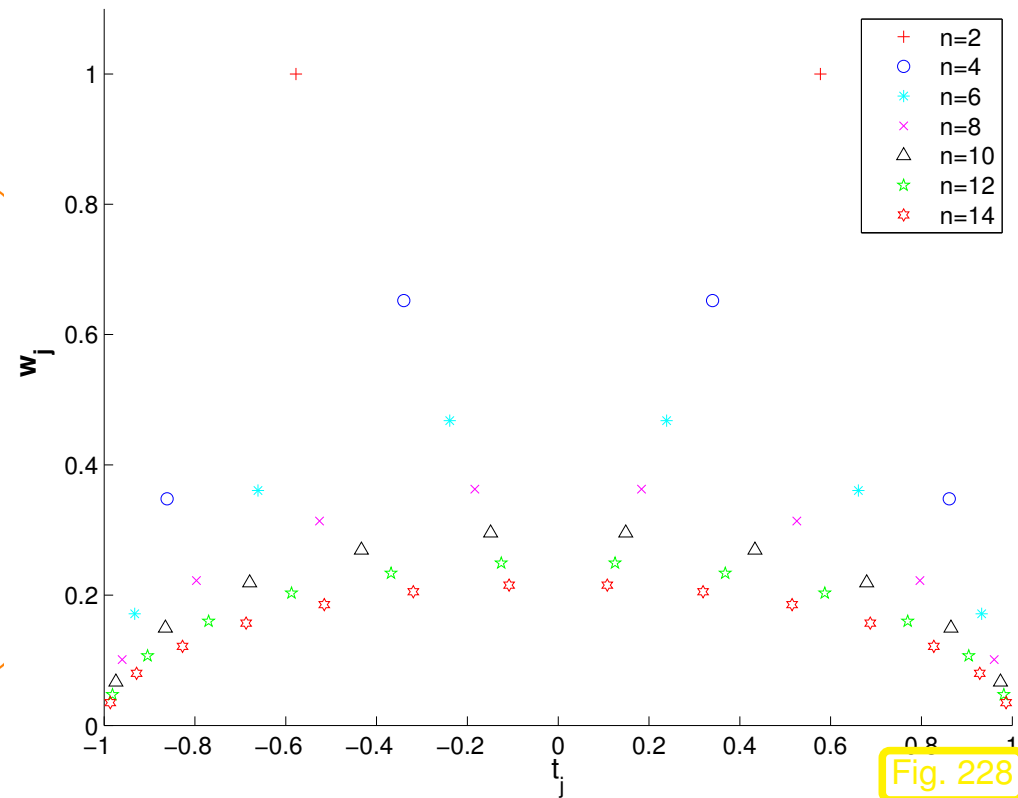


Fig. 228

*Proof.* Writing  $\xi_j^n$ ,  $j = 1, \dots, n$ , for the nodes (Gauss points) of the  $n$ -point Gauss-Legendre quadrature formula,  $n \in \mathbb{N}$ , we define

$$q_k(t) = \prod_{\substack{j=1 \\ j \neq k}}^n (t - \xi_j^n)^2 \quad \Rightarrow \quad q_k \in \mathcal{P}_{2n-2}.$$

This polynomial is integrated exactly by the quadrature rule: since  $q_k(\xi_j^n) = 0$  for  $j \neq k$

$$0 < \int_{-1}^1 q(t) dt = \omega_k^n \underbrace{q(\xi_k^n)}_{>0},$$

where  $\omega_j^n$  are the quadrature weights. □

*Remark 10.4.15* (3-Term recursion for Legendre polynomials).

Note: polynomials  $\bar{P}_n$  from (10.4.7) are uniquely characterized by the two properties (try a proof!)

- $\bar{P}_n \in \mathcal{P}_n$  with leading coefficient 1:  $\bar{P}(t) = t^n + \dots$ ,



- $\int_{-1}^1 \bar{P}_k(t) \bar{P}_j(t) dt = 0$ , if  $j \neq k$  ( $L^2([-1, 1])$ -orthogonality).

➤ same polynomials  $\bar{P}_n$  by another Gram-Schmidt orthogonalization procedure, cf. (10.4.6),

$$\bar{P}_{n+1}(t) = t\bar{P}_n(t) - \sum_{k=0}^n \frac{\int_{-1}^1 \tau \bar{P}_n(\tau) \bar{P}_k(\tau) d\tau}{\int_{-1}^1 \bar{P}_k^2(\tau) d\tau} \cdot \bar{P}_k(t)$$

By orthogonality (10.4.9) the sum collapses, since  $\int_{-1}^1 \tau \bar{P}_n(\tau) \bar{P}_k(\tau) d\tau = \int_{-1}^1 \bar{P}_n(\tau) \underbrace{(\tau \bar{P}_k(\tau))}_{\in \mathcal{P}_{k+1}} d\tau = 0$ , if  $k + 1 < n$ :

$$\bar{P}_{n+1}(t) = t\bar{P}_n(t) - \frac{\int_{-1}^1 \tau \bar{P}_n(\tau) \bar{P}_n(\tau) d\tau}{\int_{-1}^1 \bar{P}_n^2(\tau) d\tau} \cdot \bar{P}_n(t) - \frac{\int_{-1}^1 \tau \bar{P}_n(\tau) \bar{P}_{n-1}(\tau) d\tau}{\int_{-1}^1 \bar{P}_{n-1}^2(\tau) d\tau} \cdot \bar{P}_{n-1}(t) . \tag{10.4.16}$$

After rescaling (tedious!): **3-term recursion** for Legendre polynomials

$$P_{n+1}(t) := \frac{2n+1}{n+1} t P_n(t) - \frac{n}{n+1} P_{n-1}(t) \quad , \quad P_0 := 1 \quad , \quad P_1(t) := t . \tag{10.4.17}$$

Reminder (→ Sect. 9.2.1): similar 3-term recursion (9.2.3) for Chebychev polynomials. Coincidence? Of course not, nothing in mathematics holds “by accident”. 3-term recursions are a distinguishing feature of so-called families of **orthogonal polynomials**, to which the Chebychev polynomials belong

as well, spawned by Gram-Schmidt orthogonalization with respect to a weighted  $L^2$ -inner product, however, see [35, VI].

- Efficient and *stable* evaluation of Legendre polynomials by means of 3-term recursion (10.4.17), cf. the analogous algorithm for Chebychev polynomials given in Code 9.2.3.

Code 10.4.18: computing Legendre polynomials

```
1 function V= legendre (n, x)
2 V = ones(size (x)); V = [V; x];
3 for j=1:n-1
4   V = [V; ((2*j+1)/(j+1)).*x.*V(end, :) - j/(j+1)*V(end-1, :)]; end
```

Comments on Code 10.4.17:

- ☞ return value: matrix  $\mathbf{V}$  with  $(\mathbf{V})_{ij} = P_i(x_j)$
- ☞ line 2: takes into account initialization of Legendre 3-term recursion (10.4.17)

Remark 10.4.19 (Computing Gauss nodes and weights).

Code 10.4.20: Golub-Welsch algorithm

```

1 function [x,w]=gaussquad(n)
2 % octave-compatible version
3 if n==1
4     x = 0;
5     w = 2;
6 else
7     b = zeros (n-1, 1);
8     for i=1:(n-1), b(i)=i/sqrt(4*i*i-1); end
9     J=diag(b,-1)+diag(b,1); [ev,ew]=eig(J);
10    x=diag(ew); w=(2*(ev(1,:).*ev(1,:)))';
11 end

```

Compute nodes/weights of Gaussian quadrature by solving an eigenvalue problem! (Golub-Welsch algorithm [21, Sect. 3.5.4])

In codes:  $\xi_j, \omega_j$  from tables!

Justification: rewrite 3-term recurrence (10.4.17) for **scaled** Legendre polynomials  $\tilde{P}_n = \frac{1}{\sqrt{n+1/2}}P_n$

$$t\tilde{P}_n(t) = \underbrace{\frac{n}{\sqrt{4n^2-1}}}_{=:\beta_n} \tilde{P}_{n-1}(t) + \underbrace{\frac{n+1}{\sqrt{4(n+1)^2-1}}}_{=:\beta_{n+1}} \tilde{P}_{n+1}(t). \quad (10.4.21)$$

For fixed  $t \in \mathbb{R}$  (10.4.21) can be expressed as

$$\underbrace{t \begin{pmatrix} \tilde{P}_0(t) \\ \tilde{P}_1(t) \\ \vdots \\ \tilde{P}_{n-1}(t) \end{pmatrix}}_{=:\mathbf{p}(t) \in \mathbb{R}^n} = \underbrace{\begin{pmatrix} 0 & \beta_1 & & & & \\ \beta_1 & 0 & \beta_2 & & & \\ & \beta_2 & \cdots & \cdots & & \\ & & \cdots & \cdots & \cdots & \\ & & & & 0 & \beta_{n-1} \\ & & & & \beta_{n-1} & 0 \end{pmatrix}}_{=:\mathbf{J}_n \in \mathbb{R}^{n,n}} \begin{pmatrix} \tilde{P}_0(t) \\ \tilde{P}_1(t) \\ \vdots \\ \tilde{P}_{n-1}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \beta_n \tilde{P}_n(t) \end{pmatrix}$$

$$\blacktriangleright \quad \tilde{P}_n(\xi) = 0 \quad \Leftrightarrow \quad \xi \mathbf{p}(\xi) = \mathbf{J}_n \mathbf{p}(\xi) .$$

$\blacktriangleright$  The zeros of  $P_n$  can be obtained as the  $n$  real eigenvalues of the symmetric tridiagonal matrix  $\mathbf{J}_n \in \mathbb{R}^{n,n}$ !

This matrix  $\mathbf{J}_n$  is initialized in line 4 of Code 10.4.19. The computation of the weights in line 6 of Code 10.4.19 is explained in [21, Sect. 3.5.4].

The initial rationale for introducing Gauss quadrature was to obtain a high-order local quadrature rule as building block for composite quadrature rules ( $\rightarrow$  Sect. 10.3).

However, once we have the family of Gauss quadrature rules, we may also employ them as global quadrature rules on the entire integration interval  $[a, b]$  and achieve high accuracy by using more and more quadrature nodes.

This kind of use of Gaussian quadrature is explored in the next numerical experiments.

*Example 10.4.22* (Error of (non-composite) quadratures).

#### Code 10.4.23: important polynomial quadrature rules

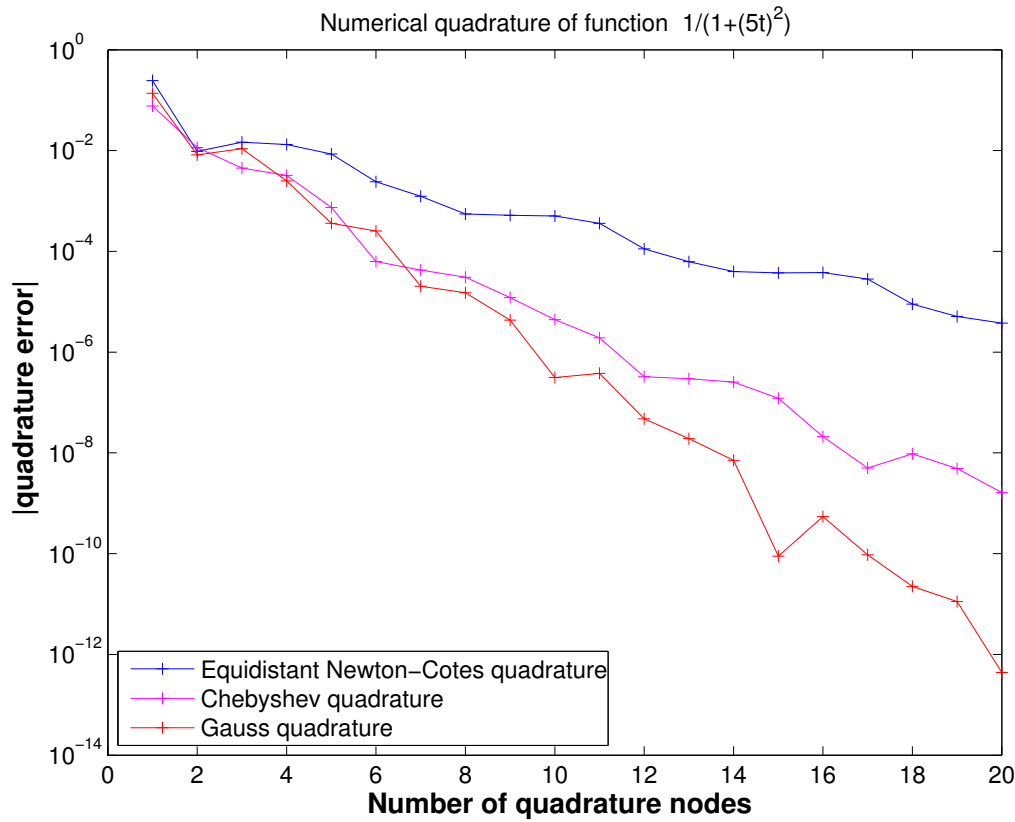
```
1 function res = numquad(f, a, b, N, mode)
2 % Numerical quadrature on [a,b] by polynomial quadrature formula
3 % f -> function to be integrated (handle), must support vector arguments
4 % a,b -> integration interval [a,b] (endpoints included)
5 % N -> Maximal degree of polynomial
```

```
6 | % mode (equidistant, Chebychev = Clenshaw-Curtis, Gauss) selects quadrature
   | rule
7 | if (nargin < 5), mode = 'equidistant'; end
8 | res = [];
9 |
0 | if strcmp(mode, 'Gauss')
1 |     for deg=1:N
2 |         [gx,w] = gaussQuad(deg);
3 |         % Gauss points for [a,b]
4 |         x = 0.5*(b-a)*gx+0.5*(a+b);
5 |         y = feval(f,x);
6 |         res = [res; deg, 0.5*(b-a)*dot(w,y)];
7 |     end
8 | else
9 |     p = (N+1:-1:1);
0 |     w = (b.^p - a.^p) ./ p;
1 |     for deg=1:N
2 |         if strcmp(mode, 'Chebychev')
3 |             % Chebychev nodes on [a,b], see (9.2.12)
4 |             x = 0.5*(b-a)*cos((2*(0:deg)+1)/(2*deg+2)*pi)+0.5*(a+b);
5 |         else
6 |             x = (a:(b-a)/deg:b);
7 |         end
8 |         % "Quick and dirty" implementation through polynomial interpolation
```

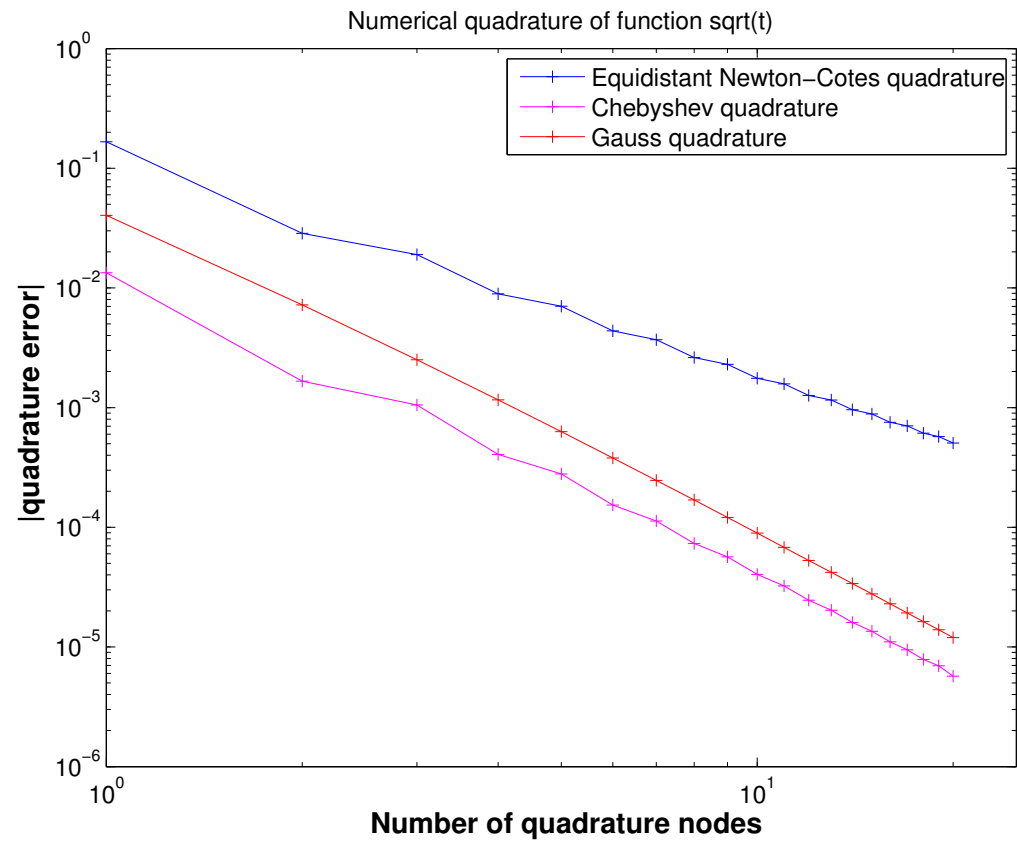
```

9  y = feval (f, x) ;
0  poly = polyfit (x, y, deg) ;
1  res = [res; deg, dot (w (N+1-deg:N+1) , poly) ] ;
2  end
3  end

```



quadrature error,  $f_1(t) := \frac{1}{1+(5t)^2}$  on  $[0, 1]$



quadrature error,  $f_2(t) := \sqrt{t}$  on  $[0, 1]$

Asymptotic behavior of quadrature error  $\epsilon_n := \left| \int_0^1 f(t) dt - Q_n(f) \right|$  for " $n \rightarrow \infty$ ":

➤ exponential convergence  $\epsilon_n \approx O(q^n)$ ,  $0 < q < 1$ , for  $C^\infty$ -integrand  $f_1 \rightsquigarrow$ : Newton-Cotes quadrature :  $q \approx 0.61$ , Clenshaw-Curtis quadrature :  $q \approx 0.40$ , Gauss-Legendre quadrature :  $q \approx 0.27$

➤ algebraic convergence  $\epsilon_n \approx O(n^{-\alpha})$ ,  $\alpha > 0$ , for integrand  $f_2$  with **singularity** at  $t = 0 \rightsquigarrow$   
Newton-Cotes quadrature :  $\alpha \approx 1.8$ , Clenshaw-Curtis quadrature :  $\alpha \approx 2.5$ , Gauss-Legendre quadrature :  $\alpha \approx 2.7$

Code 10.4.24: tracking errors of quadrature rules

```

1 function numquaderrs ()
2 % Numerical quadrature on [0,1]
3 N = 20;
4
5 figure ('Name', '1 / (1 + (5t)^2)');
6 exact = atan(5)/5;
7 eqdres = numquad(inline('1./(1+(5*x).^2)'), 0, 1, N, 'equidistant');
```



```
8 chbres = numquad(inline('1./(1+(5*x).^2)'),0,1,N,'Chebychev');
9 gaures = numquad(inline('1./(1+(5*x).^2)'),0,1,N,'Gauss');
0 semilogy(eqdres(:,1),abs(eqdres(:,2)-exact),'b+-',...
1     chbres(:,1),abs(chbres(:,2)-exact),'m+-',...
2     gaures(:,1),abs(gaures(:,2)-exact),'r+-');
3 set(gca,'fontsize',12);
4 title('Numerical quadrature of function 1/(1+(5t)^2)');
5 xlabel('{\bf Number of quadrature nodes}','fontsize',14);
6 ylabel('{\bf |quadrature error|}','fontsize',14);
7 legend('Equidistant Newton-Cotes quadrature',...
8     'Clenshaw-Curtis quadrature',...
9     'Gauss quadrature',3);
0 eqdp1 = polyfit(eqdres(:,1),log(abs(eqdres(:,2)-exact)),1)
1 chbp1 = polyfit(chbres(:,1),log(abs(chbres(:,2)-exact)),1)
2 gaup1 = polyfit(gaures(:,1),log(abs(gaures(:,2)-exact)),1)
3 print -depsc2 '../PICTURES/numquader1.eps';
4
5 figure('Name','sqrt(t)');
6 exact = 2/3;
7 eqdres = numquad(inline('sqrt(x)'),0,1,N,'equidistant');
8 chbres = numquad(inline('sqrt(x)'),0,1,N,'Chebychev');
9 gaures = numquad(inline('sqrt(x)'),0,1,N,'Gauss');
0 loglog(eqdres(:,1),abs(eqdres(:,2)-exact),'b+-',...
```

```
1     chbres(:,1), abs(chbres(:,2)-exact), 'm+-', ...
2     gaures(:,1), abs(gaures(:,2)-exact), 'r+-');
3 set(gca, 'fontsize', 12);
4 axis([1 25 0.000001 1]);
5 title('Numerical quadrature of function sqrt(t)');
6 xlabel('{\bf Number of quadrature nodes}', 'fontsize', 14);
7 ylabel('{\bf |quadrature error|}', 'fontsize', 14);
8 legend('Equidistant Newton-Cotes quadrature', ...
9         'Clenshaw-Curtis quadrature', ...
0         'Gauss quadrature', 1);
1 eqdp2 = polyfit(log(eqdres(:,1)), log(abs(eqdres(:,2)-exact)), 1)
2 chbp2 = polyfit(log(chbres(:,1)), log(abs(chbres(:,2)-exact)), 1)
3 gaup2 = polyfit(log(gaures(:,1)), log(abs(gaures(:,2)-exact)), 1)
4 print -depsc2 '../PICTURES/numquader2.eps';
```



# 10.5 Adaptive Quadrature

*Example 10.5.1* (Rationale for adaptive quadrature).

Consider composite trapezoidal rule (10.3.3) on mesh  $\mathcal{M} := \{a = x_0 < x_1 < \dots < x_m = b\}$ :

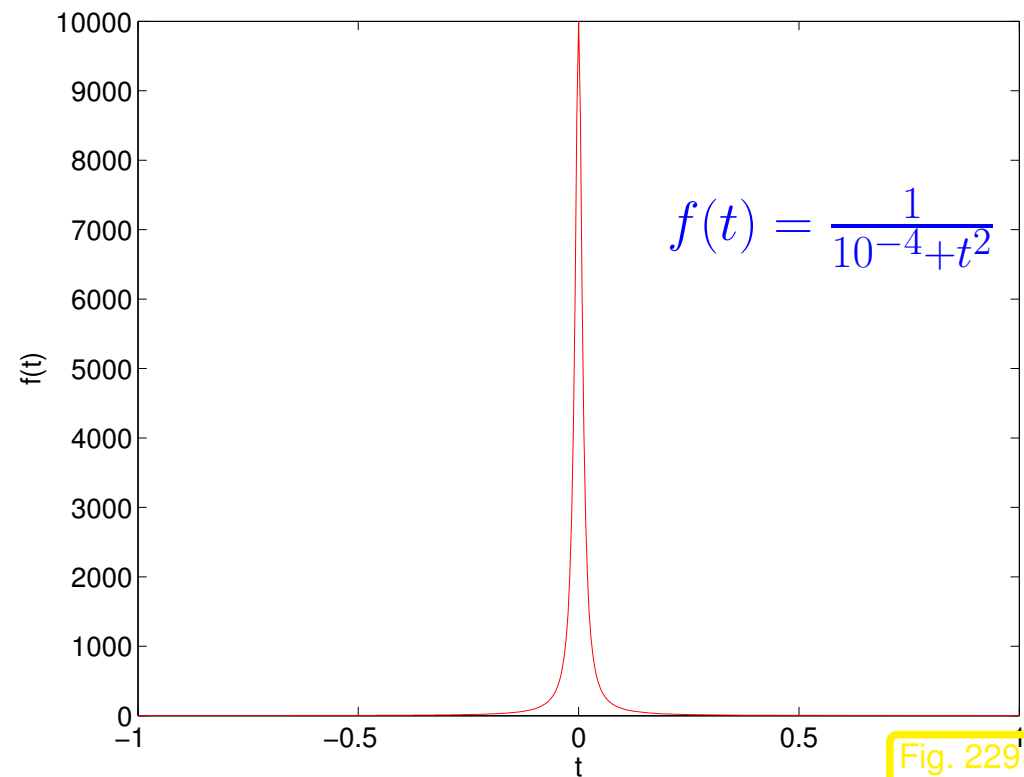
**Local** quadrature error (for  $f \in C^2([a, b])$ ):

$$\int_{x_{k-1}}^{x_k} f(t) dt - \frac{1}{2}(f(x_{k-1}) + f(x_k))$$
$$\leq (x_k - x_{k-1})^3 \|f''\|_{L^\infty([x_{k-1}, x_k])} \cdot$$

➤ Do not use equidistant mesh !

Refine  $\mathcal{M}$ , where  $|f''|$  large !

Makes sense, e.g., for “spike function”



**Goal:** *Equilibrate error contributions of all mesh intervals*

**Tool:** Local **a posteriori error estimation**  
(Estimate contributions of mesh intervals from intermediate results)

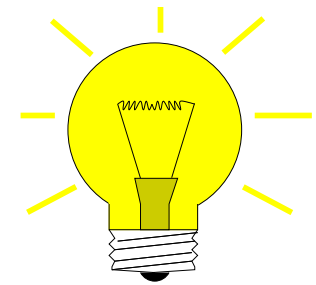
**Policy:** **Local mesh refinement**

► *Adaptive multigrid quadrature* → [17, Sect. 9.7]

Idea: local error estimation by comparing local results of two quadrature formulas  $Q_1, Q_2$  of *different* order → local error estimates

heuristics:  $\text{error}(Q_2) \ll \text{error}(Q_1) \Rightarrow \text{error}(Q_1) \approx Q_2(f) - Q_1(f)$ .

Now:  $Q_1 = \text{trapezoidal rule (order 2)} \Leftrightarrow Q_2 = \text{Simpson rule (order 4)}$



Given: mesh  $\mathcal{M} := \{a = x_0 < x_1 < \dots < x_m = b\}$

① (error estimation)

For  $I_k = [x_{k-1}, x_k]$ ,  $k = 1, \dots, m$  (midpoints  $p_k := \frac{1}{2}(x_{k-1} + x_k)$ )

$$\text{EST}_k := \left| \underbrace{\frac{h_k}{6}(f(x_{k-1}) + 4f(p_k) + f(x_k))}_{\text{Simpson rule}} - \underbrace{\frac{h_k}{4}(f(x_{k-1}) + 2f(p_k) + f(x_k))}_{\text{trapezoidal rule on split mesh interval}} \right|. \quad (10.5.2)$$

## ② (Termination)

Simpson rule on  $\mathcal{M}$   $\Rightarrow$  preliminary result  $I$

$$\text{If } \sum_{k=1}^m \text{EST}_k \leq \text{RTOL} \cdot I \quad (\text{RTOL} := \text{prescribed tolerance}) \Rightarrow \mathbf{STOP} \quad (10.5.3)$$

## ③ (local mesh refinement)

$$\mathcal{S} := \{k \in \{1, \dots, m\} : \text{EST}_k \geq \eta \cdot \frac{1}{m} \sum_{j=1}^m \text{EST}_j\}, \quad \eta \approx 0.9. \quad (10.5.4)$$

► new mesh:  $\mathcal{M}^* := \mathcal{M} \cup \{p_k : k \in \mathcal{S}\}$ .

Then continue with step ① and mesh  $\mathcal{M} \leftarrow \mathcal{M}^*$ .

Non-optimal recursive MATLAB implementation:

Code 10.5.5:  $h$ -adaptive numerical quadrature

```
1 function I = adaptquad(f, M, rtol, abstol)
```

```

2 h = diff(M); %
3 mp = 0.5*(M(1:end-1)+M(2:end)); %
4 fx = f(M); fm = f(mp); %
5 trp_loc = h.*(fx(1:end-1)+2*fm+fx(2:end))/4; %
6 simp_loc = h.*(fx(1:end-1)+4*fm+fx(2:end))/6; %
7 I = sum(simp_loc); %
8 est_loc = abs(simp_loc -trp_loc); %
9 err_tot = sum(est_loc); %
0 %
1 if ((err_tot > rtol*abs(I)) and (err_tot > abstol))
2     refcells = find(est_loc > 0.9*sum(est_loc)/length(h));
3     I = adaptquad(f, sort([M,mp(refcells)]),rtol,abstol); %
4 end

```

### Comments on Code 10.5.4:

- Arguments:  $f \hat{=}$  handle to function  $f$ ,  $M \hat{=}$  initial mesh,  $rtol \hat{=}$  relative tolerance for termination,  $abstol \hat{=}$  absolute tolerance for termination, necessary in case the exact integral value = 0, which renders a relative tolerance meaningless.
- line 2: compute lengths of mesh-intervals  $[x_{j-1}, x_j]$ ,

- line 3: store positions of midpoints  $p_j$ ,
- line 4: evaluate function (vector arguments!),
- line 5: local composite trapezoidal rule (10.3.3),
- line 6: local simpson rule (10.2.8),
- line 7: value obtained from composite simpson rule is used as intermediate approximation for integral value,
- line 8: difference of values obtained from local composite trapezoidal rule ( $\sim Q_1$ ) and local simpson rule ( $\sim Q_2$ ) is used as an estimate for the local quadrature error.
- line 9: estimate for global error by summing up **moduli** of local error contributions,
- line 10: terminate, once the estimated total error is below the relative or absolute error threshold,
- line 13 otherwise, add midpoints of mesh intervals with large error contributions according to (10.5.4) to the mesh and continue.

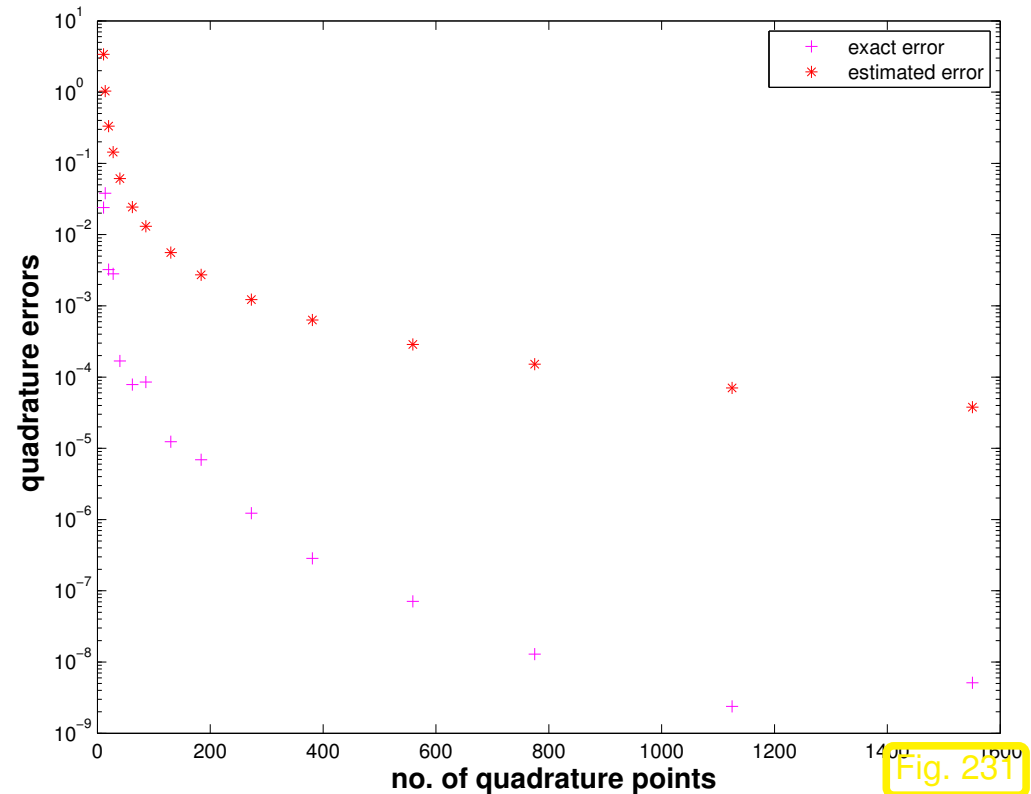
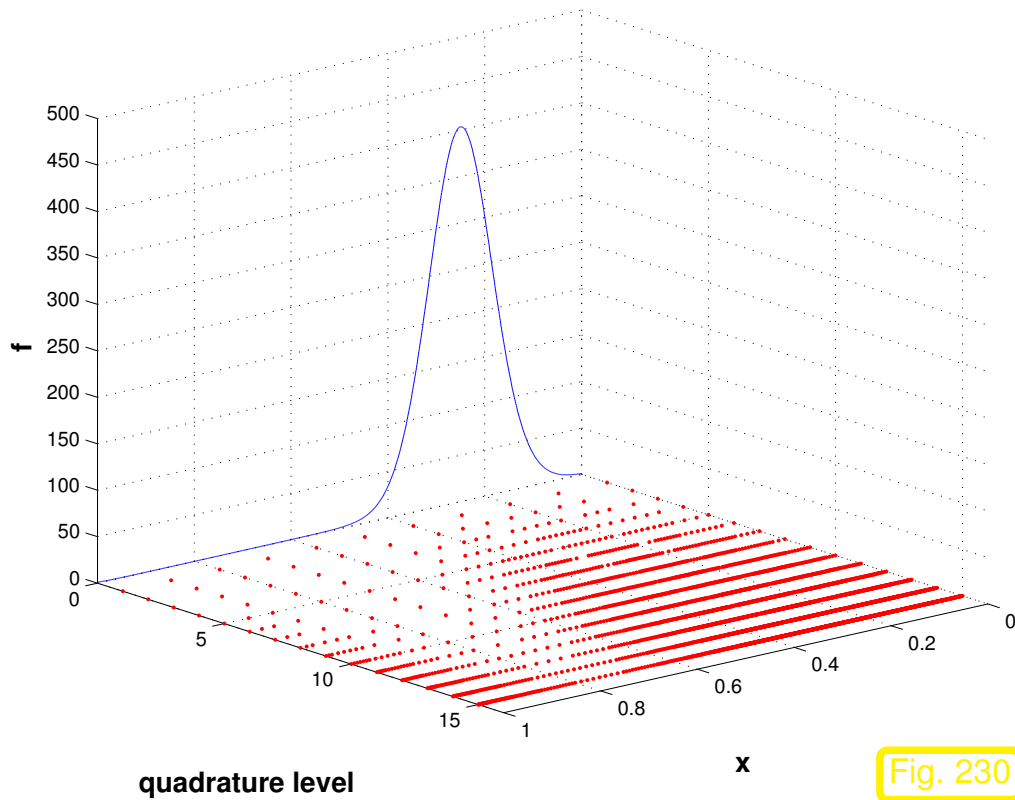
*Example 10.5.6* ( $h$ -adaptive numerical quadrature).

- approximate  $\int_0^1 \exp(6 \sin(2\pi t)) dt$ , initial mesh  $\mathcal{M}_0 = \{j/10\}_{j=0}^{10}$

Algorithm: adaptive quadrature, Code 10.5.4

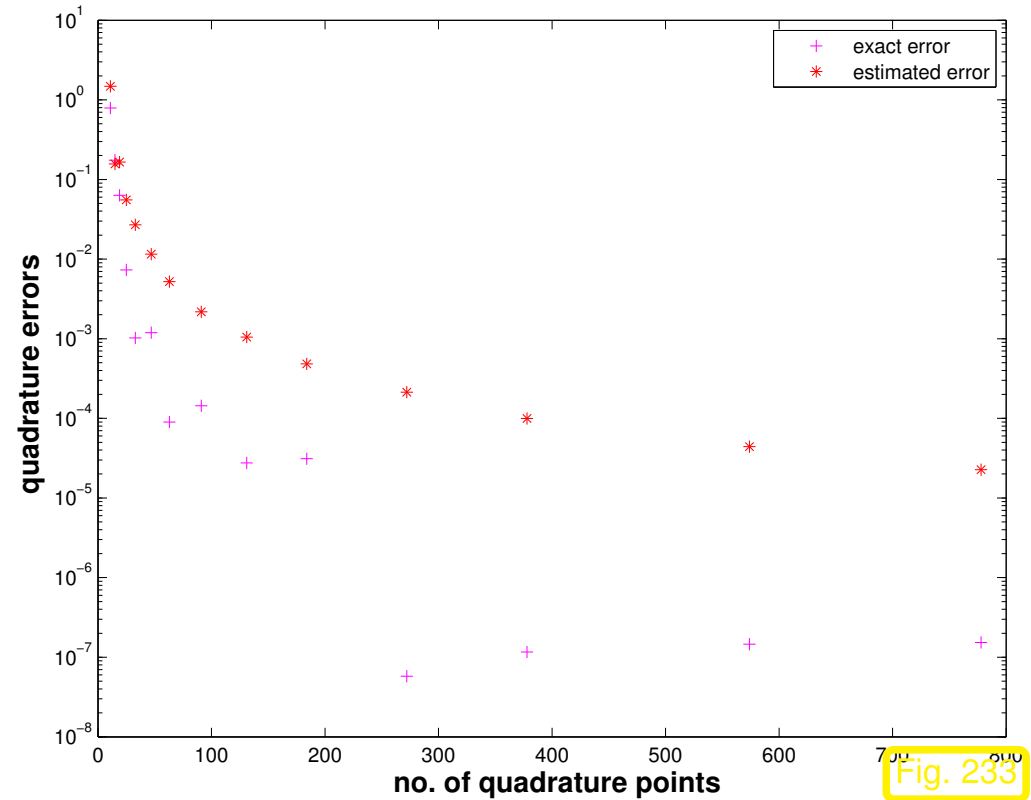
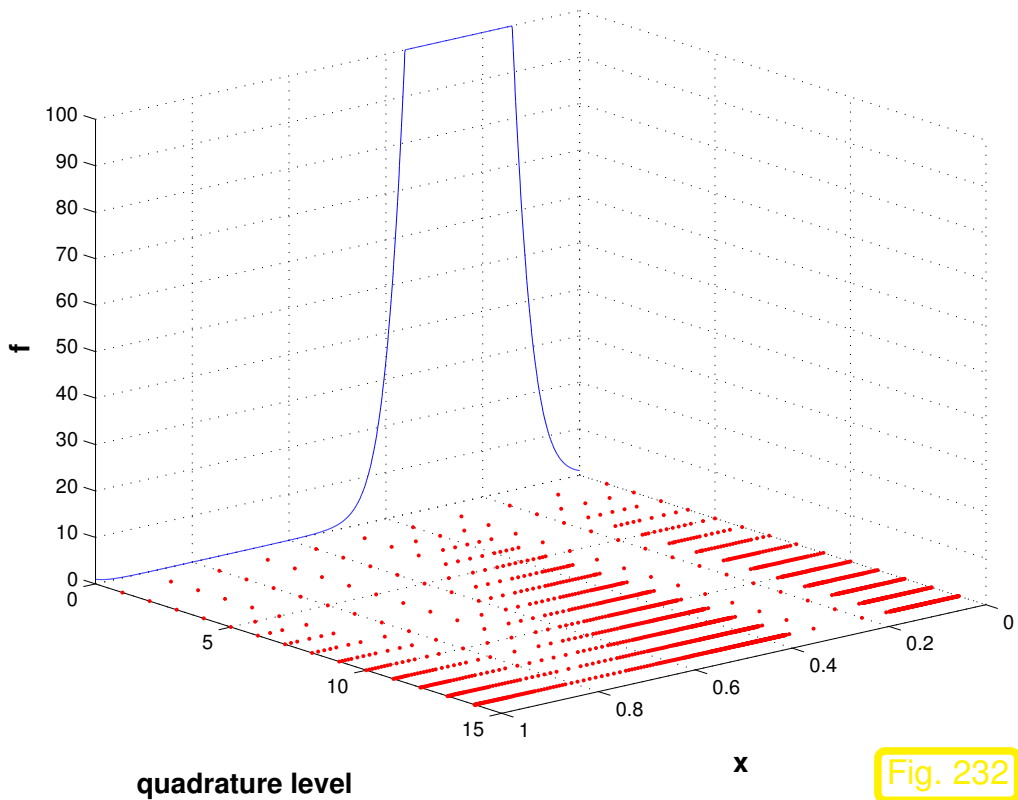
Tolerances:  $rtol = 10^{-6}$ ,  $abstol = 10^{-10}$

We monitor the distribution of quadrature points during the adaptive quadrature and the true and estimated quadrature errors. The “exact” value for the integral is computed by composite Simpson rule on an equidistant mesh with  $10^7$  intervals.





approximate  $\int_0^1 \min\{\exp(6 \sin(2\pi t)), 100\} dt$ , initial mesh as above



Observation:

Adaptive quadrature locally decreases meshwidth where integrand features variations or kinks.

- Trend for estimated error mirrors behavior of true error.
- Overestimation may be due to taking the modulus in (10.5.2)

However, the important information we want to glean from  $EST_k$  is about the *distribution* of the quadrature error.



*Remark 10.5.7* (Adaptive quadrature in MATLAB).

`q = quad(fun, a, b, tol)`: adaptive multigrid quadrature  
(local low order quadrature formulas)

`q = quadl(fun, a, b, tol)`: adaptive Gauss-Lobatto quadrature



## 11

## Clustering Techniques

## 11.1 Kernel Matrices

TASK:

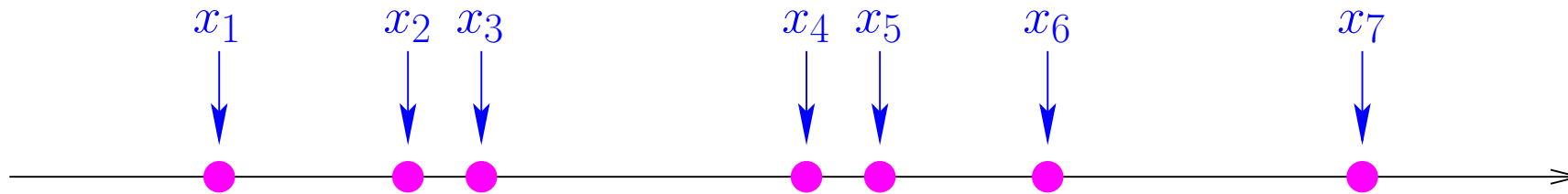
Given: **Kernel Function**  $G : I \times J \mapsto \mathbb{C}$ ,  $I, J \subset \mathbb{R}$  Interval:  $G(x, y)$  smooth for  $x \neq y$ Collocation Points  $x_1 < x_2 < \dots < x_n$ ,  $x_j \in I$ ,  $y_1 < y_2 < \dots < y_m$ ,  $y_j \in J$ 

**Collocation Matrix:**  $\mathbf{M} \in \mathbb{C}^{n,m}$   $:\Leftrightarrow (\mathbf{M})_{ij} := G(x_i, y_j)$ ,  $1 \leq i \leq n, 1 \leq j \leq m$ .

(11.1.1)

We have to find: *Efficient* algorithms for **approximate** evaluation of  $\mathbf{M} \times \text{Vector}$ (Note: Computational Effort  $O(mn)$  !)

## Example 11.1.2 (Interaction calculations for many body systems).



$n$  parallel wires with current flowing through them.

Wire  $j$  has current  $c_j \in \mathbb{R}$ , and is at position  $x_j \in \mathbb{R}$

Our Aim : To compute magnetic force on *each* wire

- Force on wire  $j$  due to all wires:  $f_j = \sum_{\substack{k=1 \\ k \neq j}}^n \frac{1}{|x_j - x_k|} c_k c_j$ ,  $j = 1, \dots, n$ .

- Force on every wire is given by vector  $\mathbf{f} = \text{diag}(c_1, \dots, c_n) \mathbf{M} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$ ,

where  $\mathbf{M} = (m_{ij})_{i,j=1}^n$ ,  $m_{ij} = \begin{cases} \frac{1}{|x_j - x_i|} & \text{for } i \neq j, \\ 0 & \text{for } i = j. \end{cases}$

Collocation matrix  $\mathbf{M}$  will be formed using **kernel function**  $G(x, y) = \frac{1}{|x - y|}$

*Example 11.1.3* (Gravitational forces in galaxy).

Number of stars in galaxy  $n$  ( $\approx 10^9$ ) with position  $\mathbf{x}_i \in \mathbb{R}^3$  and mass  $m_i$ ,  $i = 1, \dots, n$ .

Gravitational force on each star is required for simulation of dynamics of galaxy

Our Aim : To compute gravitational force on each star

- Gravitational force on star  $j$  :

$$f_j = \frac{G}{4\pi} \sum_{\substack{i \neq j \\ j \in \{1, \dots, n\}}} \frac{1}{\|\mathbf{x}_i - \mathbf{y}_j\|} m_i m_j ,$$



- Gravitational force on every star is given by

$$\mathbf{f} = \text{diag}(m_1, \dots, m_n) \mathbf{M} \begin{pmatrix} m_1 \\ \vdots \\ m_n \end{pmatrix} , \quad m_{ij} := \begin{cases} \frac{G}{4\pi} \frac{1}{\|\mathbf{x}_i - \mathbf{y}_j\|} & \text{for } i \neq j , \\ 0 & \text{for } i = j , \end{cases} \quad 1 \leq i, j \leq n .$$

The above example is a 3D generalization of our original task.

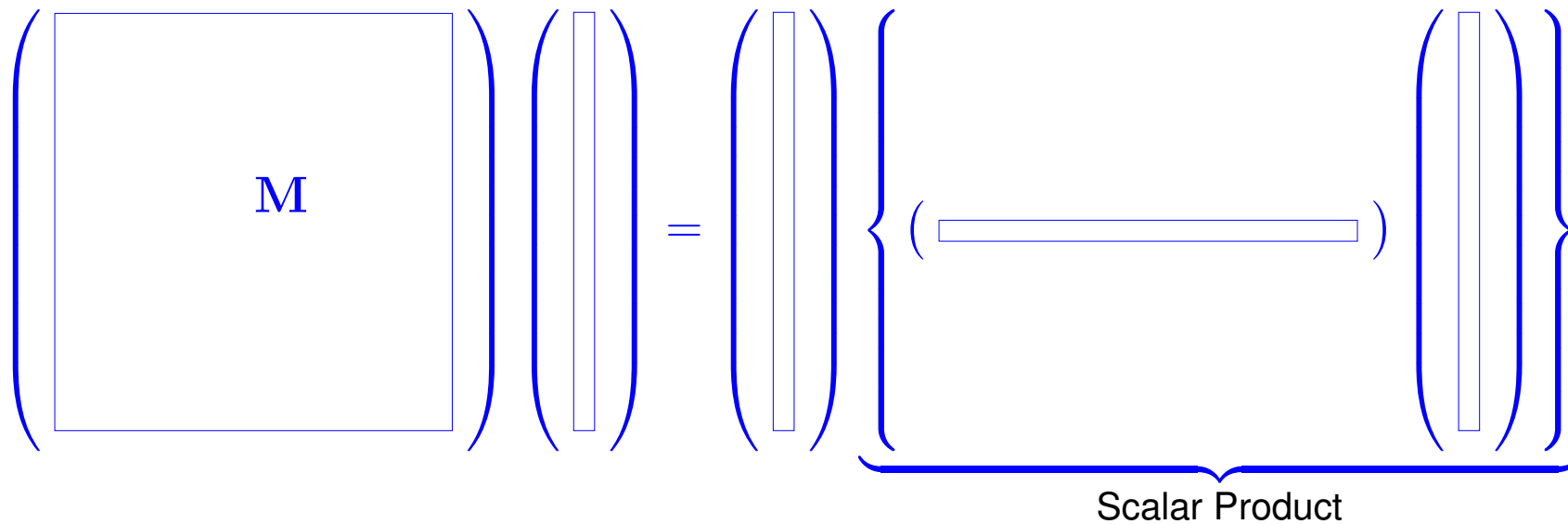


# 11.2 Local Separable Approximation

If kernel function is **separable** i.e. :  $G(x, y) = g(x)h(y)$ ,  $g : I \mapsto \mathbb{C}$ ,  $h : J \mapsto \mathbb{C}$

$$\mathbf{M} = \left( g(x_j) \right)_{j=1}^n \cdot \left( h(y_j) \right)_{j=1}^m {}^T \quad \text{rank}(\mathbf{M}) = 1 .$$

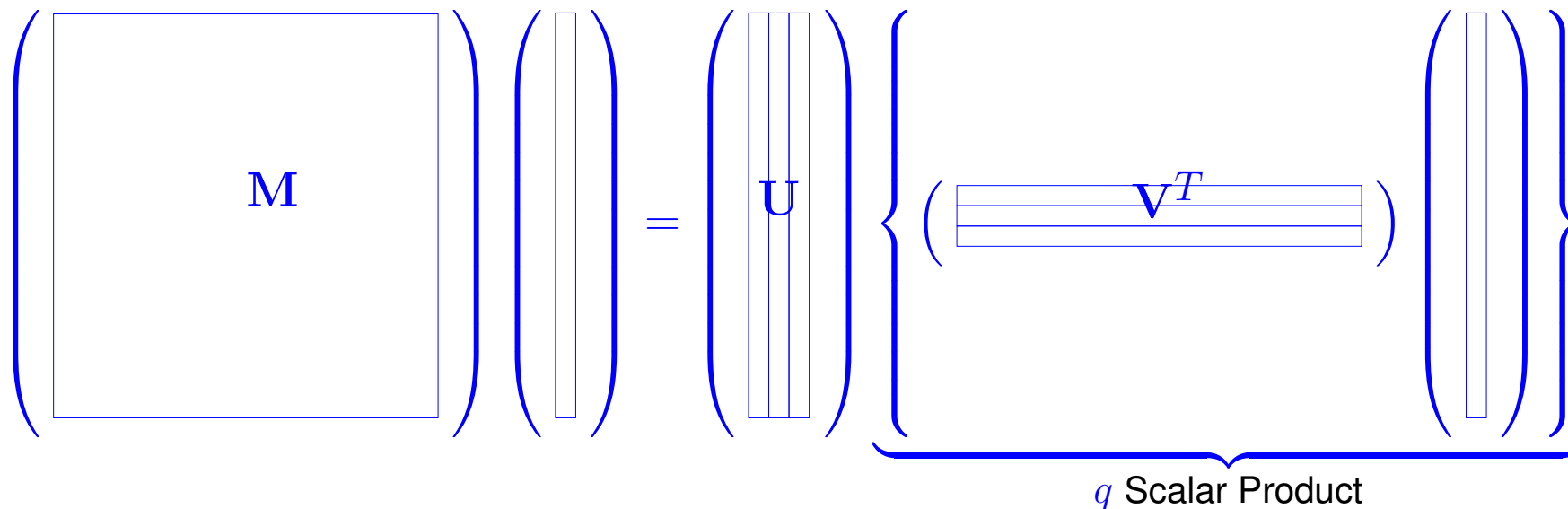
Computational Effort ( $\mathbf{M} \times \text{Vector}$ ) =  $m + n$



Generalization:  $G(x, y) = \sum_{j=1}^q g_j(x)h_j(y)$ ,  $g_j : I \mapsto \mathbb{C}$ ,  $h_j : J \mapsto \mathbb{C}$ ,  $q \in \mathbb{N}$

$$\mathbf{M} = \mathbf{U}\mathbf{V}^T, \quad \begin{array}{l} \mathbf{U} \in \mathbb{R}^{n,q}, \quad u_{ij} = g_j(x_i), \quad j \in \{1, \dots, q\}, i \in \{1, \dots, n\}, \\ \mathbf{V} \in \mathbb{R}^{q,m}, \quad v_{ij} = h_i(y_j), \quad i \in \{1, \dots, q\}, j \in \{1, \dots, m\}. \end{array}$$

If  $\text{rank}(\mathbf{M}) = q$  then Computational Effort ( $\mathbf{M} \times \text{Vector}$ ) =  $q(m + n)$



Idea: Global approximation of  $G$  through sum of separated kernel function:

$$G(x, y) \approx \tilde{G}(x, y) := \sum_{l=0}^d \sum_{k=0}^d \kappa_{l,k} g_l(x) h_k(y), \quad \kappa_{l,j} \in \mathbb{C}, \quad (x, y) \in I \times J. \quad (11.2.1)$$

As  $G$  is approximated by  $\tilde{G}$  therefore  $\mathbf{M}$  is approximated by  $\tilde{\mathbf{M}}$ , where  $\tilde{m}_{ij} = \tilde{G}(x_i, y_j)$

*Remark 11.2.2* (Quality measure for kernel approximation).

$$\|\mathbf{M} - \widetilde{\mathbf{M}}\|_2^2 \leq \|\mathbf{M} - \widetilde{\mathbf{M}}\|_F^2 = \sum_{i,j} (m_{ij} - \tilde{m}_{ij})^2 = \boxed{\sum_{i,j} (G(x_i, y_j) - \tilde{G}(x_i, y_j))^2} .$$

Normalizing quality measure: 
$$\frac{1}{mn} \sum_{i,j} (G(x_i, y_j) - \tilde{G}(x_i, y_j))^2 . \quad (11.2.3)$$

△

Now, our aim is to find separable kernel approximation

**Definition 11.2.4** (Tensor product interpolation polynomial).  $L_j^x \in \mathcal{P}_n$  ( $L_k^y \in \mathcal{P}_m$ ),  $j = 0, \dots, n$  ( $k = 0, \dots, m$ ), Lagrange polynomial on nodes of mesh  $\mathcal{X} := \{x_j\}_{j=0}^n \subset I$  ( $\mathcal{Y} := \{y_k\}_{k=0}^m \subset J$ ),  $I, J \subset \mathbb{R}$  interval. Continuous  $f : I \times J \mapsto \mathbb{C}$  defined by

$$(I_{\mathcal{X} \times \mathcal{Y}} f)(x, y) := \sum_{j=0}^n \sum_{k=0}^m f(x_j, y_k) L_j^x(x) L_k^y(y) , \quad x, y \in \mathbb{R} ,$$

This is *tensor product interpolation polynomial*.



$$G(x, y) \approx \tilde{G}(x, y) := \sum_{j=0}^d \sum_{k=0}^d G(t_j^x, t_k^y) L_j^x(x) L_k^y(y), \quad (11.2.5)$$

$t_0^x, \dots, t_d^x / t_0^y, \dots, t_d^y$  Chebyshev-nodes in  $I/J$ , and  $L_j^x, L_k^y$  subordinate Lagrange-polynomial.

# Example 11.2.6 (Global separable approximation by smooth kernel function).

- Smooth (even analytic) kernel function  $G(x, y) = \frac{1}{1 + |x - y|^2}$ , collocation points  $x_i = y_i = \frac{i-1}{n}, i = 1, \dots, n$ .
- $\tilde{G}$  approximated by tensor product Chebyshev interpolation polynomial (11.2.10) of degree  $d \in \mathbb{N}$ .

$$\frac{1}{n^2} \sum_{i,j} (G(x_i, y_j) - \tilde{G}(x_i, y_j))^2$$

as function of  $d$  for  $n \in \{100, 200, 400\}$

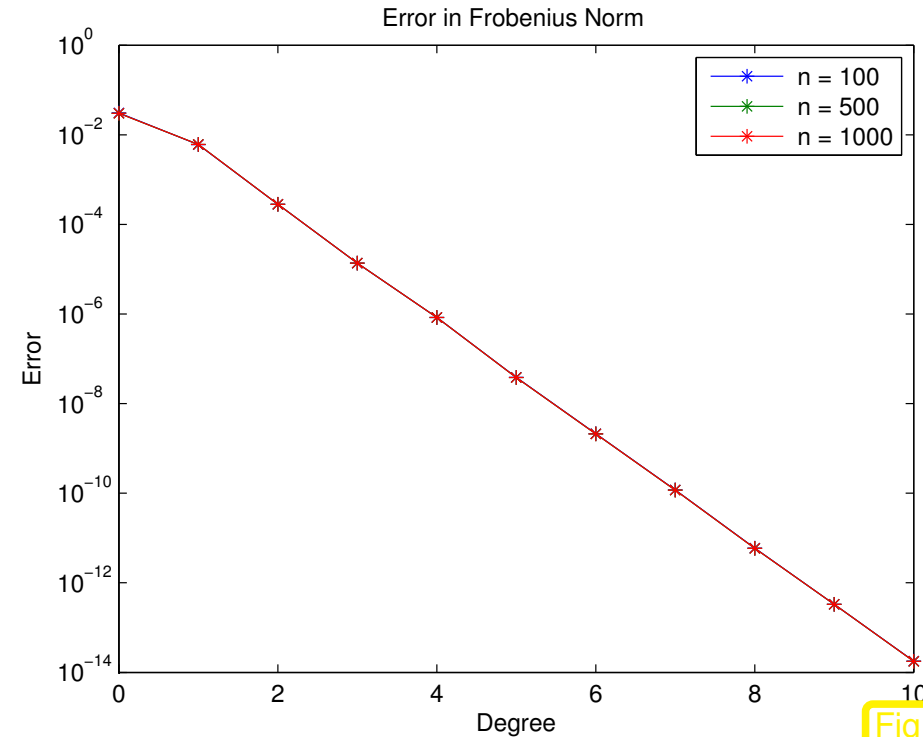


Fig. 234

In file `changeable.c`

- put the admissibility condition as return `true` in `admissible_ClusterPair`
- put kernel function as `1.0/(1.0 + pow(fabs(dX - dY),2))` in `kernel_function`
- put `NONE` as return values in `get_iOperation`
  - Run `main.cpp`, input as follows  
Choice - 2, NumberofPts - 100 1000 0,  
Number of Increments - 2, Increment 1 - 400, Increment 2 - 500  
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1  
Degree - 0 10 1, Admissibility Coefficient - 0 0 1  
Output File from C++: `Error.txt`
  - OutputFile from MATLAB : `Error_in_Frobenius_Norm_Pts.eps`
  - Run `error_plot_NumberofPts.m` from Matlab

Note: Exponential Convergence  $\|M - \tilde{M}\|_F \rightarrow 0$  in dependence of  $d$



Example 11.2.7 (Global separable approximation by non-smooth kernel function).

Analysis as in ex. 11.2.6, now for the kernel function

$$G(x, y) = \begin{cases} \frac{1}{|x-y|} & \text{if } x \neq y, \\ 0 & \text{otherwise.} \end{cases} \quad (11.2.8)$$

from ex. 11.1.2.

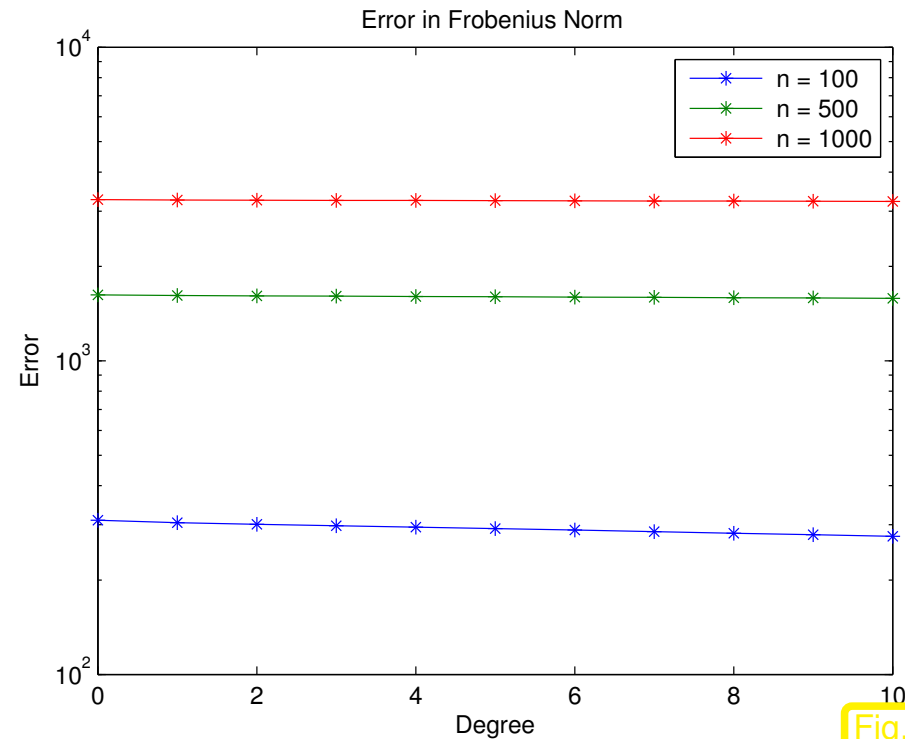


Fig. 235

In file changeable.c

- a) put the admissibility condition as return **true** in `admissible_ClusterPair`
- b) put kernel function as **0.0** for `fabs(dX-dY) < EPS` as **1.0/fabs(dX - dY)** otherwise in `kernel_function`
- c) put **NONE** as return values in `get_iOperation`
  - 1) Run `main.cpp`, input as follows
    - Choice - 2, NumberofPts - 100 1000 0,
    - Number of Increments - 2, Increment 1 - 400, Increment 2 - 500
    - Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1
    - Degree - 0 10 1, Admissibility Coefficient - 0 0 1
  - Output File from C++: `Error.txt`
  - OutputFile from MATLAB : `Error_in_Frobenius_Norm_Pts.eps`
  - 2) Run `error_plot_NumberofPts.m` from Matlab



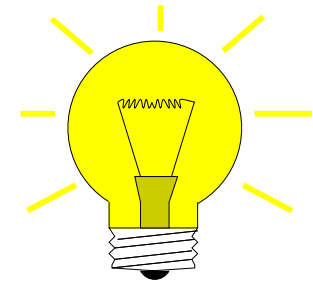
Note: (Virtually) No Convergence  $\left\| \mathbf{M} - \widetilde{\mathbf{M}} \right\|_F \rightarrow 0$  for  $d \rightarrow \infty$

Reason: Missing *global* smoothness

Poor approximation of  $G(x, y) := |x - y|^{-1}$  in region of  $\{(x, y) \in I \times J : x = y\}$ .

However  $G(x, y)$  from (11.2.8) is smooth (even analytic, Def. 9.2.20) at “large distances” from  $\{(x, y) \in I \times J : x = y\}$ .

Idea: **Local** approximation of  $G$  through sum of separated kernel function




$$G(x, y) \approx \sum_{l=0}^d \sum_{k=0}^d \kappa_{l,k} g_l(x) h_k(y), \quad \kappa_{l,j} \in \mathbb{C}, \quad (11.2.9)$$

$$(x, y) \in \widetilde{I} \times \widetilde{Y}, \quad \widetilde{I} \subset I, \quad \widetilde{J} \subset J,$$

with  $\widetilde{I} \times \widetilde{J} \cap \{(x, y) : x = y\} = \emptyset$ .

Actually:

Local approximation of  $G$  on rectangle  $\tilde{I} \times \tilde{J}$  which is a *partition* of  $I \times J$

 Possible partition for separable kernel approximation through local tensor product Chebyshev polynomial interpolation (11.2.9)  
(Admissible rectangles)

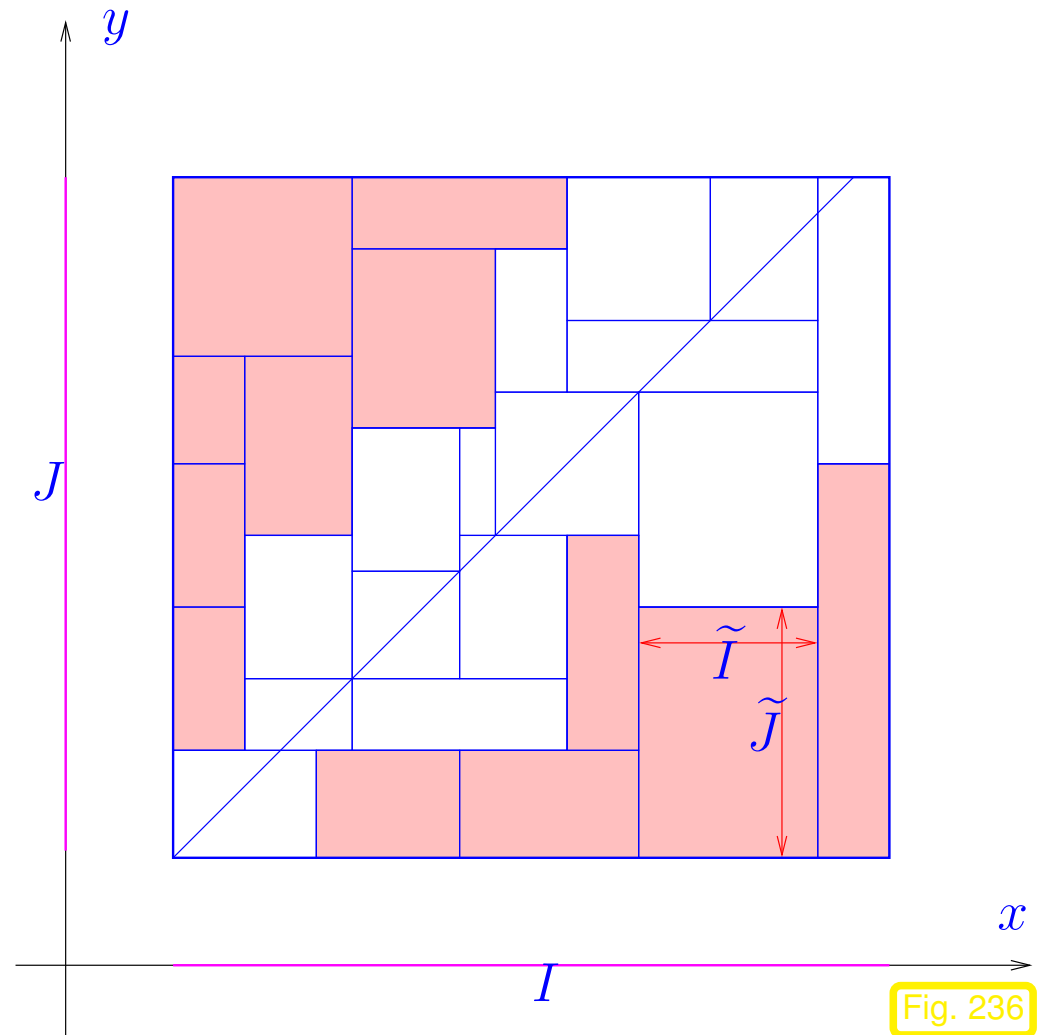
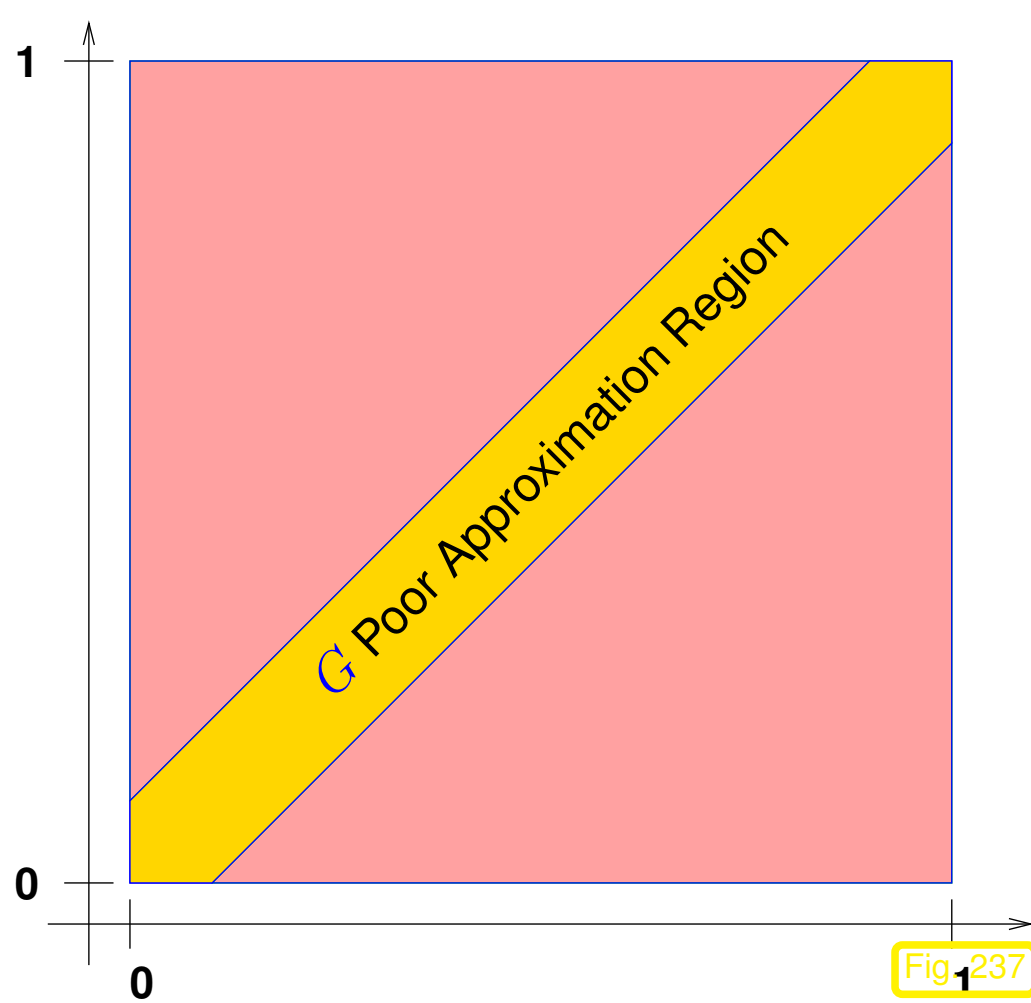


Fig. 236



Terminology:

 Near Field:  $G$  Poor approximation

 Far Field:  $G$  Good approximation

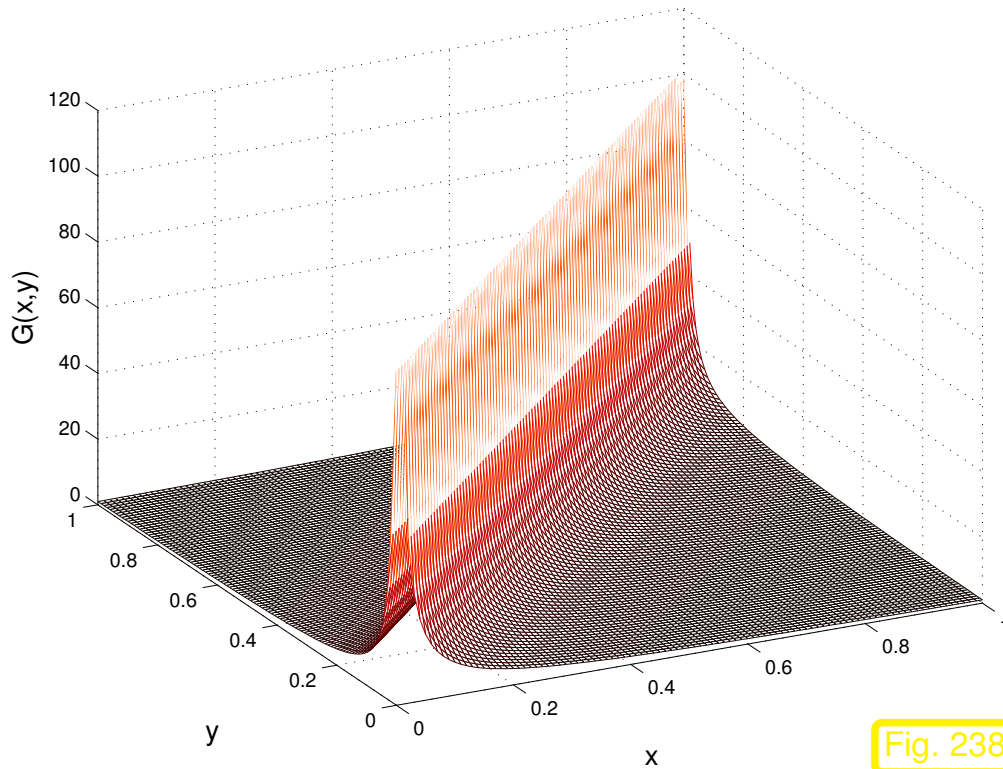


Fig. 238

Kernel function (11.2.8) for  $[0, 1]^2$

$$G(x, y) = \frac{1}{|x - y|}$$

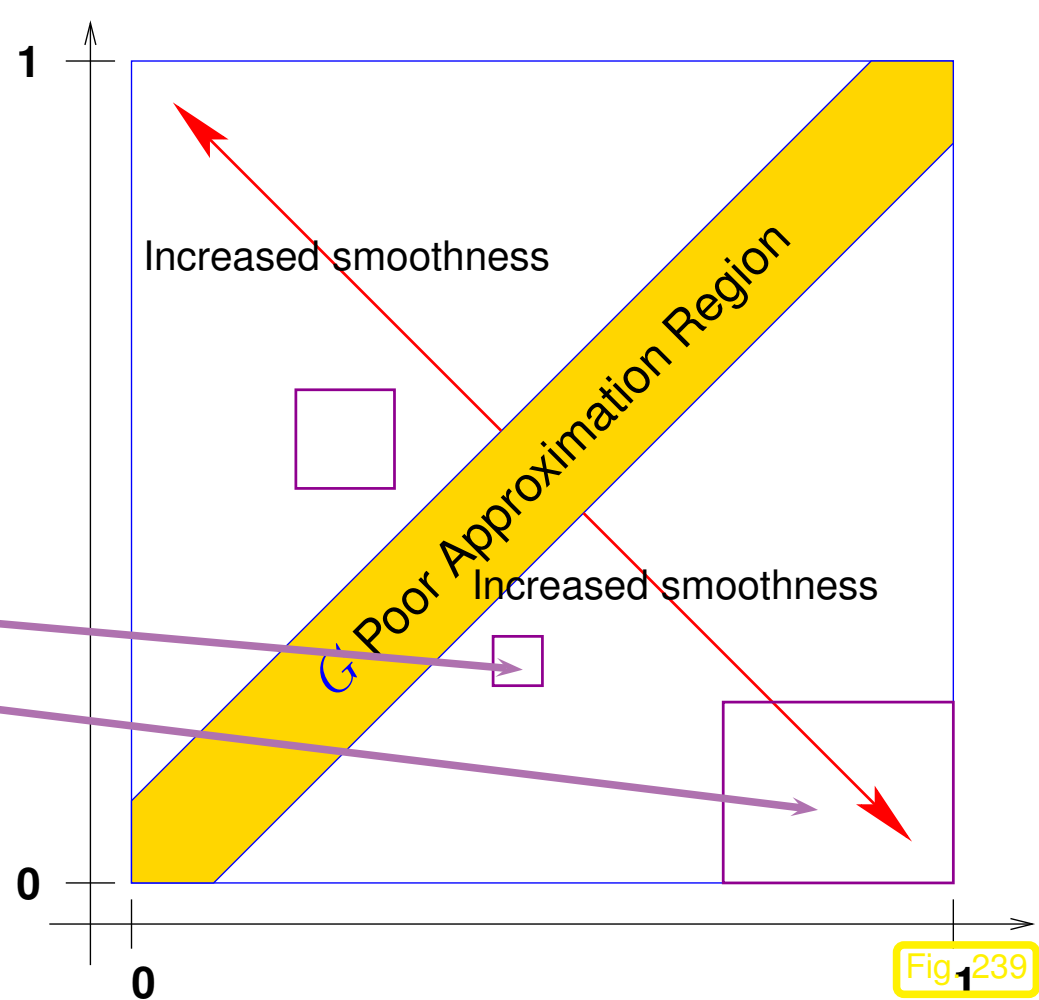
- Singular for  $x = y$
- Analytic far from  $\{(x, y): x = y\}$ , analyticity ( $\rightarrow$  Chapter ??) increases with  $|x - y|$

Similar kernel function:  $G(x, y) = \log |x - y|$ ,  $G(x, y) = \frac{\partial}{\partial x} \frac{1}{|x - y|}$ , etc.

“Smoothness of  $G(x, y) = \frac{1}{|x-y|}$  increases with growing distance from diagonal  $\{x = y\}$ ”:

- No approximation, when  $|x - y|$  is “small”
- Tensor product interpolation polynomial for
  - Small rectangles near diagonal
  - Large rectangles far from diagonal

Now, our aim is to find appropriate sizes of above mentioned rectangles



*Example 11.2.10* (Tensor product Chebyshev interpolation on rectangles).

$I = J = [0, 1]^2$ ,  $G(x, y) = |x - y|^{-1}$  from (11.2.8), (Approximate) Max norm of the error of local tensor product Chebyshev interpolation polynomial of  $G$  on rectangles of constant size, but with growing distance from  $\{(x, y): x = y\}$

$$\tilde{I}_k = [0.55 + k \cdot 0.05, 0.75 + k \cdot 0.05] \quad , \quad \tilde{J}_k = [0.25 - k \cdot 0.05, 0.45 - k \cdot 0.05] \quad , \quad k \in \{0, \dots, 5\} .$$



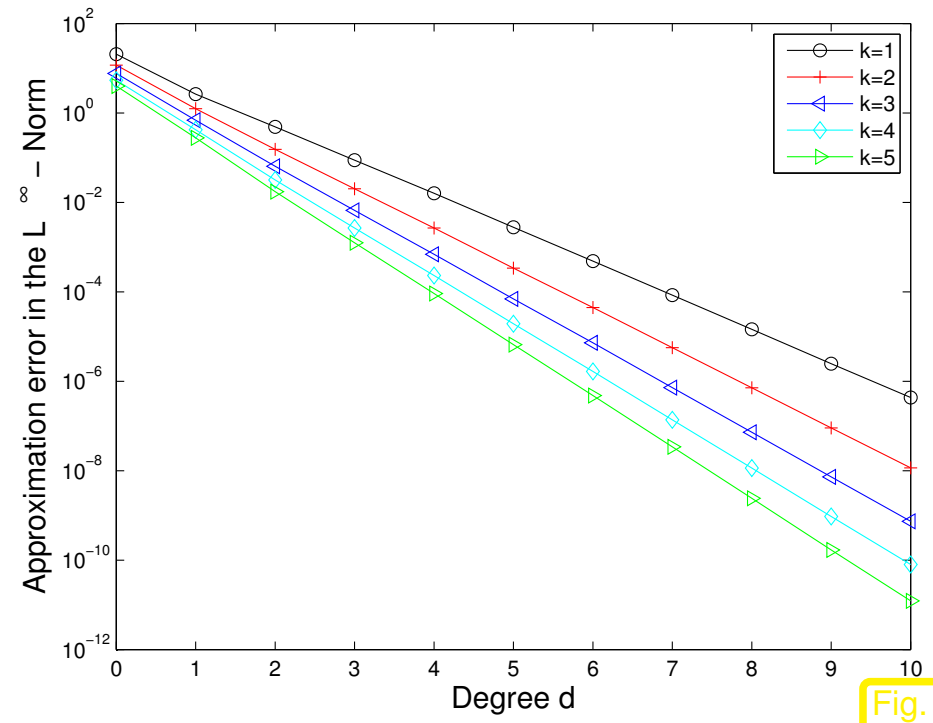
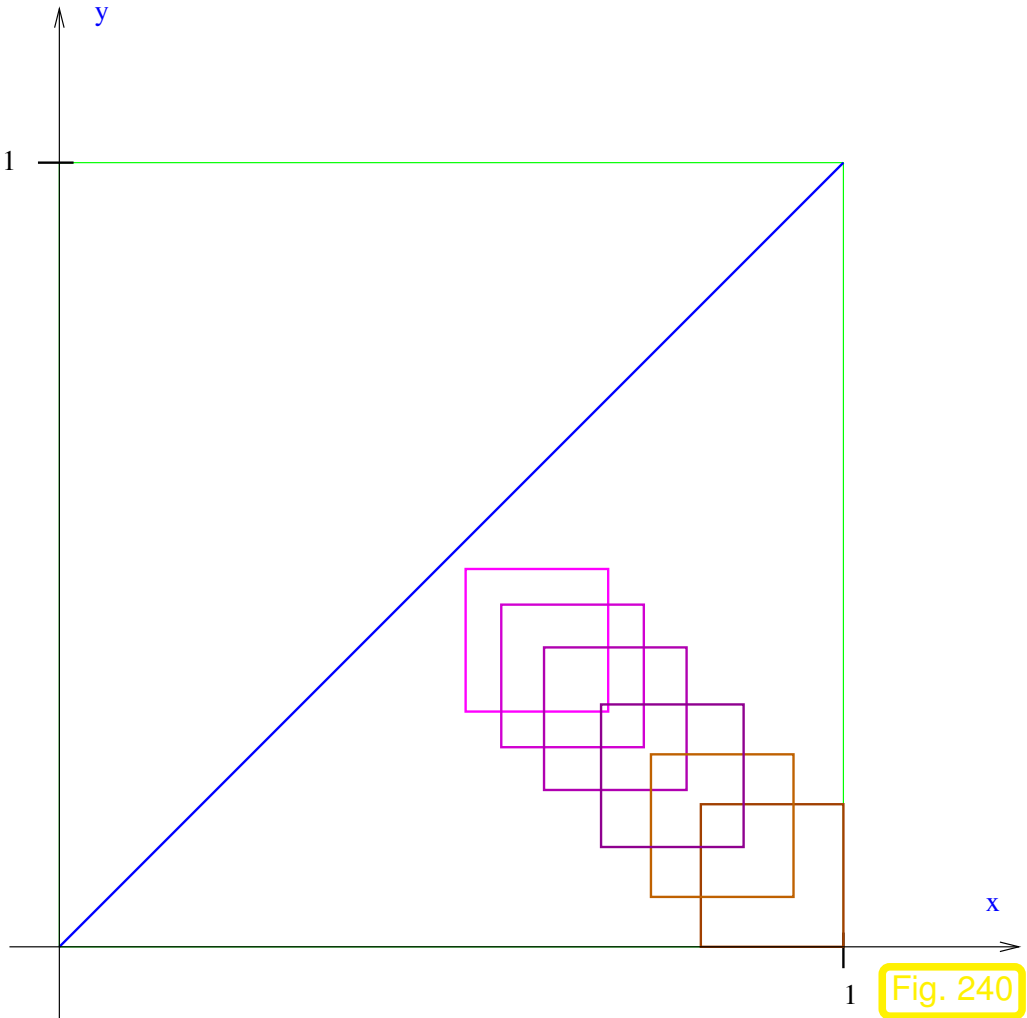


Fig. 241

**NOTE:** Decreasing interpolation errors with increasing distance from  $\{(x, y): x = y\}$



Example 11.2.11 (Tensor product Chebyshev interpolation for variable rectangle sizes).

Taking the variable rectangle sizes as

$$\left[\frac{1}{2}(\sqrt{2} - 1)\xi + \frac{1}{2}, \frac{1}{2}(\sqrt{2} + 1)\xi + \frac{1}{2}\right] \times \left[-\frac{1}{2}(\sqrt{2} - 1)\xi + \frac{1}{2}, -\frac{1}{2}(\sqrt{2} + 1)\xi + \frac{1}{2}\right], \quad 0.05 \leq \xi \leq \frac{1}{1 + \sqrt{2}}$$

that is, Size of rectangles  $\sim$  Distance from diagonal

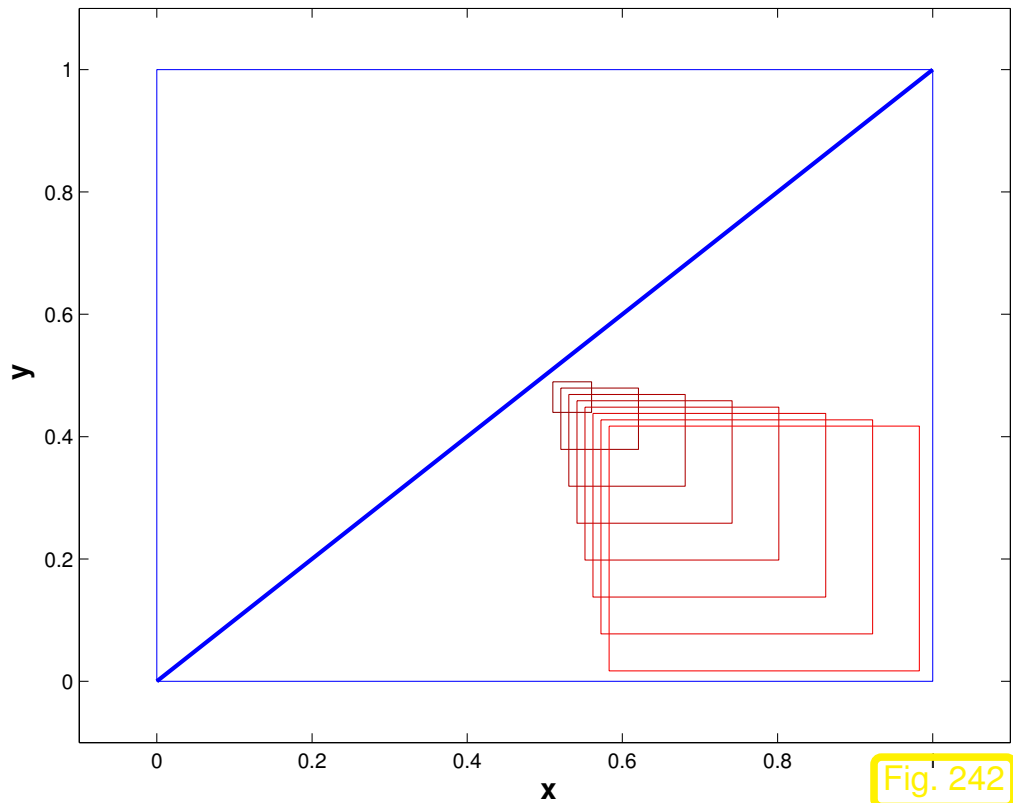


Fig. 242

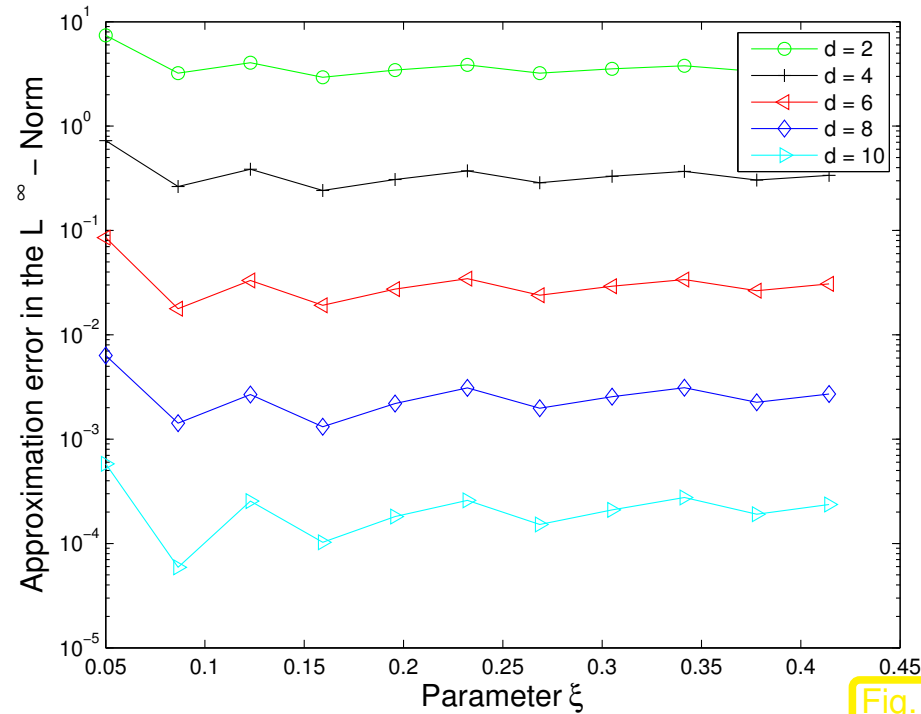


Fig. 243

**NOTE:** Controlled decrease of interpolation errors with increase in size of rectangles



**Acceptability Criteria**

$[a, b] \times [c, d]$  is called  $\eta$ -admissible,  $\eta > 0$ , if

$$\eta \text{dist}([a, b], [c, d]) \geq \max\{b - a, d - c\} . \tag{11.2.12}$$

( $\text{dist}([a, b], [c, d]) := \min\{|x - y| : x \in [a, b], y \in [c, d]\}$ )

# 11.3 Cluster Trees

TASK: Given  $\eta > 0$ , find (efficient algorithms) partitioning of  $I \times J$  "far from diagonal" which are  $\eta$ -admissible rectangles.

Remark 11.3.1. Perspective:

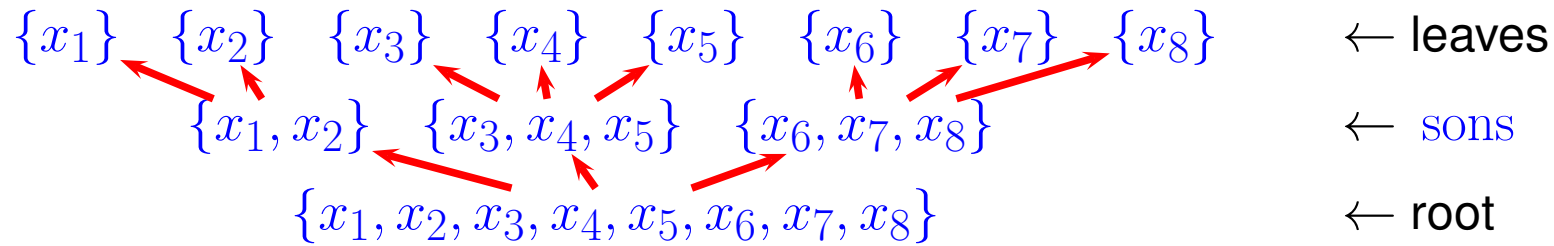
Partitioning of  $I \times J \implies$  Partitioning of  $\{x_i\}_{i=1}^n \times \{y_j\}_{j=1}^m \iff$  Partitioning of index mesh  $\{1, \dots, n\} \times \{1, \dots, m\}$



**Definition 11.3.2** (Cluster Tree). A Tree ( $\rightarrow$  Computer Science, Graph theory)  $T$  is called **Cluster Tree** on  $\mathbb{P} := \{x_1, \dots, x_n\} \subset \mathbb{R} \iff$

- The nodes of the Tree  $T$  (= **Cluster**) are subset of  $\mathbb{P}$ .
- $\text{root}(T) = \mathbb{P}$ .
- For every node  $\sigma$  of  $T$ :  $\{\sigma' : \sigma' \in \text{sons}(\sigma)\}$  is Partitioning of  $\sigma$ .

**Bounding Box** of Cluster  $\sigma \in T$ :  $\Gamma(\sigma) := [\min \{x\}_{x \in \sigma}, \max \{x\}_{x \in \sigma}]$



← leaves

← sons

← root

Terminology ( $\rightarrow$  Graph theory):

Cluster Tree  $T$ :  $\sigma \in T: \text{sons}(\sigma) = \emptyset \iff \sigma$  leaf

Cluster Tree  $T$ :

Recursive Construction

MATLAB-Data Structure:

$N \times 6$ -Matrix,  $N = \#T$ ,

Lines  $\hat{=}$  Cluster

$T = [T; i, j, xl, xr, s1, s2]$

$i, j$  :  $\sigma = \{x_i, \dots, x_j\}$

$xl$  :  $xl = \min_{i \leq k \leq j} x_k$

$xr$  :  $xr = \max_{i \leq k \leq j} x_k$

$s1, s2$ : The line indices of son

**NOTE** :  $\#T \leq 2n$

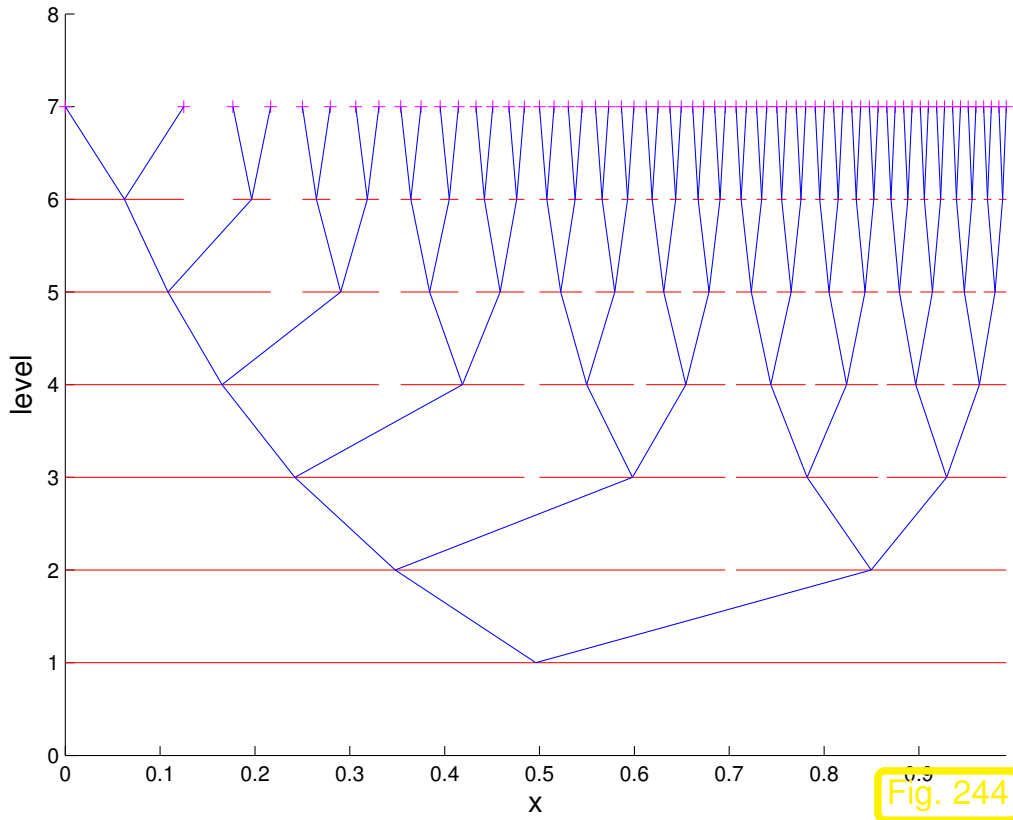
MATLAB-CODE : Construction of Cluster Trees

```
function [T,idx] = ct_rec(x,ofs,m,T)
N = length(x);
if (N <= m)
    T = [T;ofs,ofs+N-1,x(1),x(end),0,0];
else
    n = round(N/2);
    [T,s1] = ct_rec(x(1:n),ofs,m,T);
    [T,s2] = ct_rec(x(n+1:N),ofs+n,m,T);
    T = [T;ofs,ofs+N-1,x(1),x(end),s1,s2];
end
idx = size(T,1);
```

## C++ code: Construction of Cluster Trees

```
1 void clsClusteringApproximation::build_ClusterTree_Linear(int iOffset, int
  iNumberOfPtsInClusterLinear, int* piDimension){
2   int iTempLeftChild, iTempRightChild, iStart, iEnd;
3   int iStartLeft, iPtsInLeft, iStartRight, iPtsInRight;
4
5   iStart = iOffset;
6   iEnd = iOffset + iNumberOfPtsInClusterLinear - 1;
7   if (iNumberOfPtsInClusterLinear == 1){
8     add_ClusterLinear(*piDimension, iStart, iEnd, -1, -1);
9   }
10  else{
11    iStartLeft = iOffset;
12    iPtsInLeft = iNumberOfPtsInClusterLinear - (iNumberOfPtsInClusterLinear/2);
13    build_ClusterTree_Linear(iStartLeft, iPtsInLeft, piDimension);
14    iTempLeftChild = vec2pclClusterLinear[*piDimension].size() - 1;
15
16    iStartRight = iOffset + iPtsInLeft;
17    iPtsInRight = iNumberOfPtsInClusterLinear/2;
18    build_ClusterTree_Linear(iStartRight, iPtsInRight, piDimension);
19    iTempRightChild = vec2pclClusterLinear[*piDimension].size() - 1;
20
21    add_ClusterLinear(*piDimension, iStart, iEnd, iTempLeftChild, iTempRightChild);
22  }
23 }
```

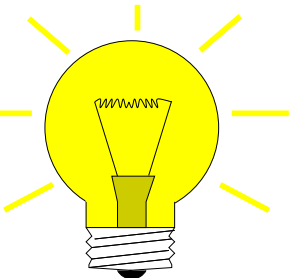
# Example 11.3.3 (Cluster Tree).



← Binary cluster tree:

```
x = sqrt(0:1/64:(1-1/64));
```

- $2^{l-1}$  Cluster on Level  $l$  with  $2^{7-l}$  points.  
(balanced Cluster tree)



Idea: Choose approximation rectangles = Tensor product of Bounding Box of Cluster  $\in T$

Cluster  $\sigma, \mu$  admissible  $:\Leftrightarrow$

$\Gamma(\sigma), \Gamma(\mu)$  admissible

Recursive construction for  
admissible Partitioning:

$P_{near}$  = **Near field**:

The area near the diagonal  
( $\rightarrow$  No approximation)

$P_{far}$  = **Far field**:

Partitioning with  $\eta$ -admissible  
rectangles

MATLAB-CODE : Recursive construction for admissible partitioning

```
function [Pnear,Pfar] = ...
    divide(Tx,Ty,σ,μ,Pnear,Pfar,η)
cls = Tx(σ,:); clm = Ty(μ,:);
if (σ = leaf | μ = leaf)
    Pnear = [Pnear; cls(2),cls(3),clm(2),clm(3)];
elseif ((σ,μ) admissible)
    Pfar = [Pfar; cls(2),cls(3), clm(2),clm(3)];
else
    for s1 = cls(6:7), for s2 = clm(6:7)
        [Pnear,Pfar] = divide(Tx,Ty,s1,s2,Pnear,Pfar,η);
    end; end
end
```

MATLAB-CODE : Parent Code

```
function [Pnear,Pfar] = partition(Tx,Ty,η)
Pnear = []; Pfar = [];
σ = find(Tx(:,1) == min(Tx(:,1)));
μ = find(Ty(:,1) == min(Ty(:,1)));
[Pnear,Pfar] = divide(Tx,Ty,σ,μ,Pnear,Pfar,η);
```

Note: clusters of a far field cluster pair have a common level!



```
1 void clsClusteringApproximation::build_ClusterPair (int iIndex1,  
2     int iIndex2){  
3     clsClusterLinear *pclsClusterLinear1, *pclsClusterLinear2;  
4  
5     pclsClusterLinear1 = vec2pclsClusterLinear[0][iIndex1];  
6     pclsClusterLinear2 = vec2pclsClusterLinear[1][iIndex2];  
7     if (leaf(0,iIndex1) || leaf(1,iIndex2)) {  
8         add_ClusterPairNear(pclsClusterLinear1,  
9             pclsClusterLinear2);  
10        (pclsClusterLinear1->get_ptr_AppearsIn())->push_back(-vecp  
11        (pclsClusterLinear2->get_ptr_AppearsIn())->push_back(-vecp  
12    }  
13    else if (admissible_ClusterPair(iIndex1, iIndex2)) {  
14        add_ClusterPairFar(pclsClusterLinear1,  
15            pclsClusterLinear2);  
16        (pclsClusterLinear1->get_ptr_AppearsIn())->push_back(vecpc  
17        (pclsClusterLinear2->get_ptr_AppearsIn())->push_back(vecpc  
18    }  
19    else {  
20        tree_traverse(pclsClusterLinear1, pclsClusterLinear2);  
    }
```

```
1 void clsClusteringApproximation::preprocess() {
2     int iLoopVari, iLoopVarj;
3     for (iLoopVari = 0; iLoopVari < iDimension; iLoopVari++){
4         build_ClusterTree_Linear(0, veciNumberofPts[iLoopVari],
5             &iLoopVari);
6     }
7     build_ClusterPair(vec2pclsClusterLinear[0].size() - 1,
8         vec2pclsClusterLinear[1].size() - 1);
9 }
```

Cluster Pairs Formed for Number of Pts (64,64) and Admissibility Coefficient : 0.5

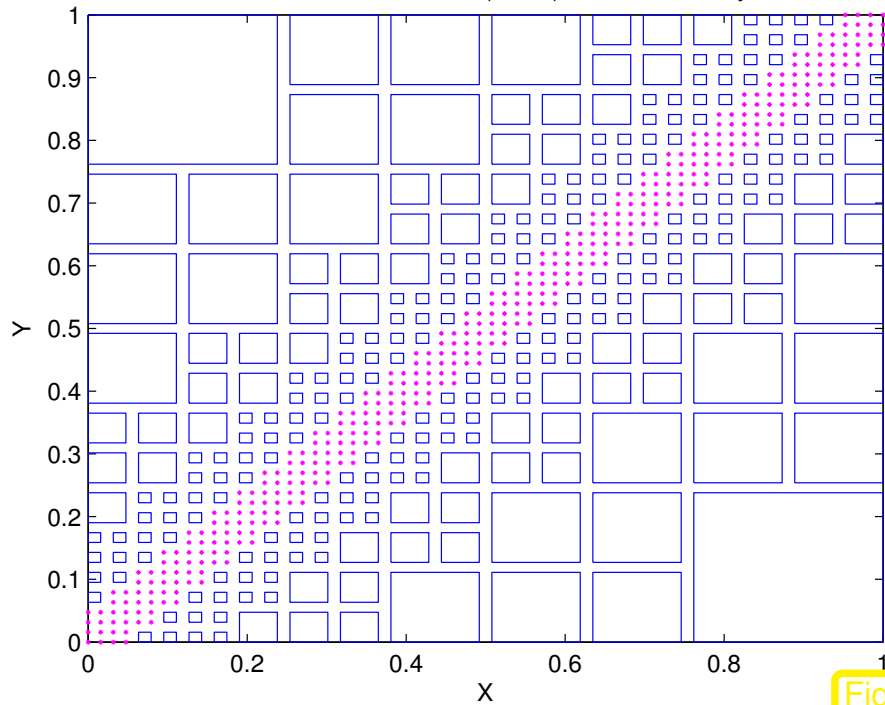


Fig. 245

$x = (0 : 1/64 : (1 - 1/64)) ;$   
In file changeable.c

- a) put the admissibility condition in `admissible_ClusterPair`
- b) put kernel function in `kernel_function`
- c) put **NONE** as return values in `get_iOperation`

1) Run `main.cpp`, input as follows

Choice - 2, NumberofPts - 64 64 1,

Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1

Degree - 0 0 1, Admissibility Coefficient - 0.5 0.5 1

Output File from C++: `ClusterLinear1_AA.txt`, `ClusterLinear2_AA.txt`,

`ClusterFar_AA.txt`, `ClusterLinearNear_AA.txt`,

OutputFile from MATLAB : `ClusterFigure_64_64_0.5.eps`

2) Run `cluster_pair_indiv.m` from Matlab, arguments:

0 0 (argument here is for the Point Number

and Admissibility Coefficient. As 64 Number of Pts is the

first in the iteration over Number of Pts,

therefore argument is given as 0

same holds for Admissibility Coefficient)

Cluster Pairs Formed for Number of Pts (64,64) and Admissibility Coefficient : 0.5

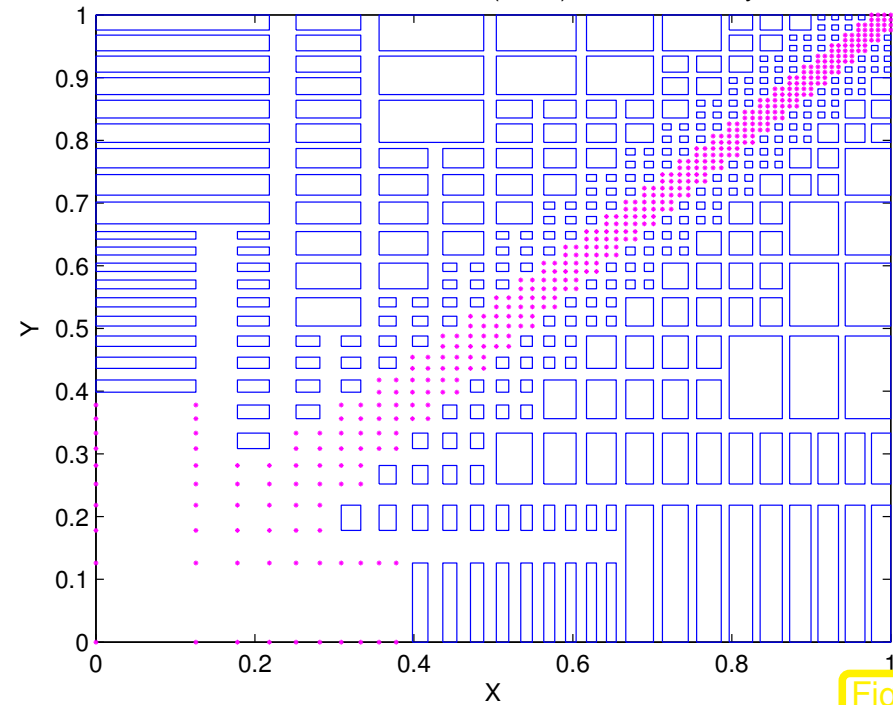


Fig. 246

$x = \text{sqrt}(0 : 1/64 : (1 - 1/64)) ;$   
In file changeable.c

- a) put the admissibility condition in `admissible_ClusterPair`
- b) put kernel function in `kernel_function`
- c) put **SQRT** as return values in `get_iOperation`

1) Run `main.cpp`, input as follows

Choice - 2, NumberofPts - 64 64 1,

Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1

Degree - 0 0 1, Admissibility Coefficient - 0.5 0.5 1

Output File from C++: `ClusterLinear1_AA.txt`, `ClusterLinear2_AA.txt`,

`ClusterFar_AA.txt`, `ClusterLinearNear_AA.txt`,

OutputFile from MATLAB : `ClusterFigure_64_64_0.5.eps`

2) Run `cluster_pair_indiv.m` from Matlab, arguments:

0 0 (argument here is for the Point Number

and Admissibility Coefficient. As 64 Number of Pts is the

first in the iteration over Number of Pts,

therefore argument is given as 0

same holds for Admissibility Coefficient)

Points on mesh(n): \* = Near field points , □ = Partition rectangles (Far field)

**NOTE :** Axis sections of the partition rectangles  $\in \{\Gamma(\sigma) : \sigma \in T\}$ ,  $T \hat{=} \text{Cluster Tree}$  ( $\rightarrow$  Def. 11.3.2)

## 11.4 Algorithm

Local kernel approximation of  
Far field-Rectangles through Tensor  
product Chebyshev interpolation  
polynomial



Low rank approximation of  $\mathbf{M}$  on  
blocks as (11.1.1)

see Figs. 245,246

R. Hiptmair  
rev 38286,  
October 28,  
2009

Notation:

$$\begin{aligned} \sigma &:= \{i_1, \dots, i_2\} \subset \{1, \dots, n\} \\ \mu &:= \{j_1, \dots, j_2\} \subset \{1, \dots, m\} \end{aligned} \Rightarrow \mathbf{M}_{|\sigma \times \mu} := \begin{pmatrix} m_{ij} \end{pmatrix}_{\substack{i=i_1, \dots, i_2 \\ j=j_1, \dots, j_2}} .$$



# Application of partitioning

$$\{x_1, \dots, x_n\} \times \{y_1, \dots, y_m\} = \left( \bigcup_{(\sigma, \mu) \in P^{\text{near}}} \sigma \times \mu \right) \cup \left( \bigcup_{(\sigma, \mu) \in P^{\text{far}}} \sigma \times \mu \right).$$

Algorithm: Analysis of  $\mathbf{f} = \mathbf{M}\mathbf{c}$ ,  $\mathbf{M}$  as (11.1.1)

$$f_i = \sum_{j=1}^m G(x_i, y_j) c_j = \underbrace{\sum_{j \in P^{\text{near}}(i)} G(x_i, y_j) c_j}_{\text{Near field contribution}} + \underbrace{\sum_{\substack{\sigma \in T_x \\ x_i \in \sigma}} \sum_{\substack{\mu \in T_y \\ (\sigma, \mu) \in P^{\text{far}}}} \sum_{\substack{1 \leq j \leq m \\ y_j \in \mu}} G(x_i, y_j) c_j}_{\text{Far field contribution}}.$$

Near field coupling indices  $P^{\text{near}}(i) := \{j \in \{1, \dots, m\} : (\{x_i\} \times \{y_j\}) \in P^{\text{near}}\}$

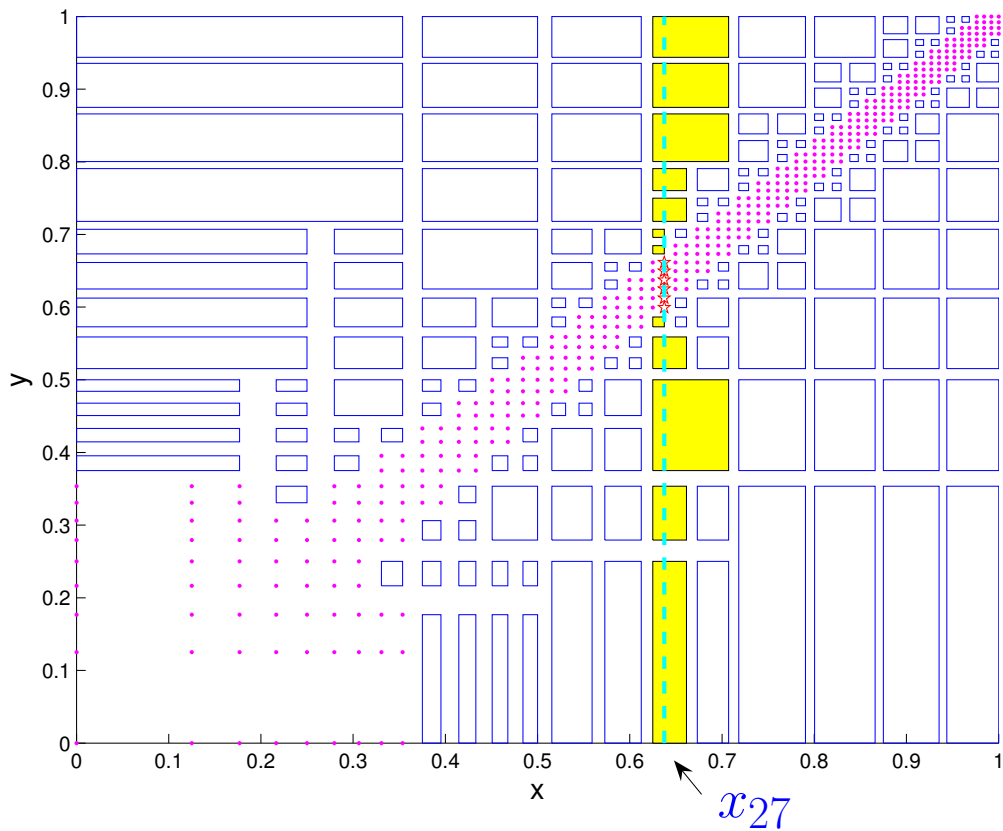


Illustration:

$x = \text{sqrt}(0:1/64:1)$ ; ,  $i = 27$ ,  $\eta = 1$ :

■ → Far field clusters with contribution to  $f_i$

★ → Near field clusters with contribution to  $f_i$

$$\approx \sum_{j \in P^{\text{near}}(i)} G(x_i, y_j) c_j + \sum_{\substack{\sigma \in T_x \\ x_i \in \sigma}} \sum_{\substack{\mu \in T_y \\ (\sigma, \mu) \in P^{\text{far}}}} \sum_{\substack{1 \leq j \leq m \\ y_j \in \mu}} \sum_{l=0}^d \sum_{k=0}^d G(t_l^\sigma, t_k^\mu) L_l^\sigma(x_i) L_k^\mu(y_j) c_j$$

$$\approx \sum_{j \in P^{\text{near}}(i)} G(x_i, y_j) c_j + \sum_{\substack{\sigma \in T_x \\ x_i \in \sigma}} \sum_{\substack{\mu \in T_y \\ (\sigma, \mu) \in P^{\text{far}}}} (\mathbf{V}_\sigma \mathbf{X}_{\sigma, \mu} \mathbf{V}_\mu^T \mathbf{c}_{|\mu})_i ,$$

with

$t_l^\sigma, t_k^\mu :=$  Chebyshev nodes (9.2.12) in  $\Gamma(\sigma), \Gamma(\mu)$ ,  $l = 0, \dots, d$ ,

$$\mathbf{X}_{\sigma, \mu} := (G(t_l^\sigma, t_k^\mu))_{l, k=0}^d \in \mathbb{R}^{d+1, d+1}, \quad (11.4.1)$$

$$\mathbf{V}_\sigma := (L_l^\sigma(x_i))_{\substack{i: x_i \in \sigma \\ l=0, \dots, d}} \in \mathbb{R}^{\#\sigma, d+1}, \quad (11.4.2)$$

$$\mathbf{V}_\mu := (L_k^\mu(y_j))_{\substack{j: y_j \in \mu \\ k=0, \dots, d}} \in \mathbb{R}^{\#\mu, d+1}, \quad (11.4.3)$$

**Analysis of Complexity** (under the assumption  $n \approx m$ ):

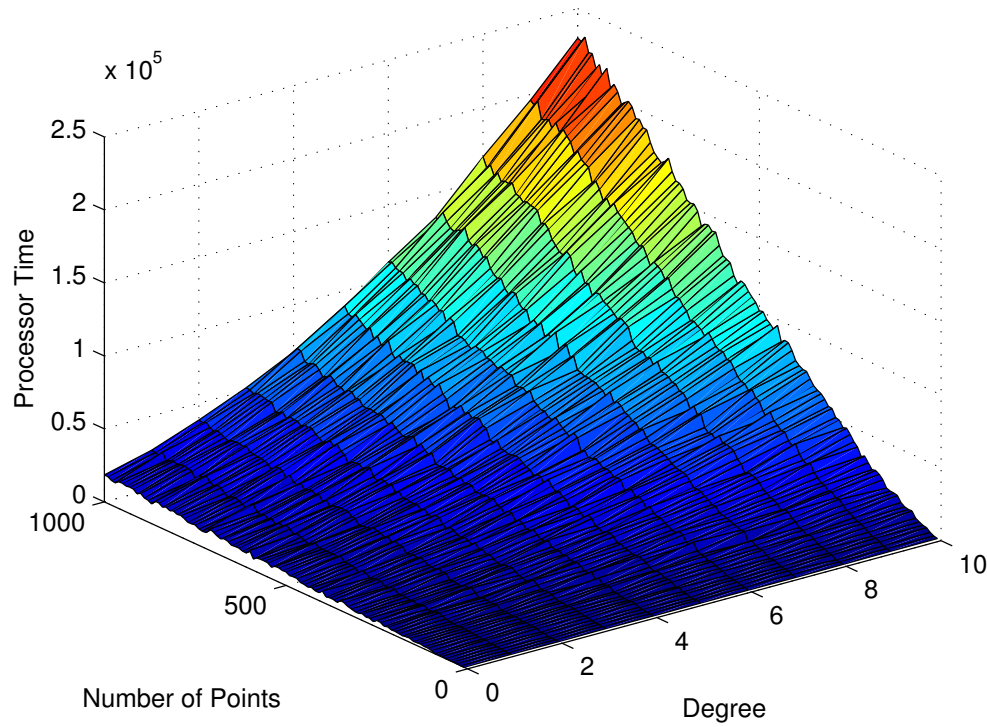
①: Trick: Calculate on above assumption

Compute  $\mathbf{w}_\mu := \mathbf{V}_\mu^T \mathbf{c}|_\mu \in \mathbb{R}^{d+1}$ ,  $\mu \in T_y \rightarrow O(n)$  Operations on every level of  $T_y$

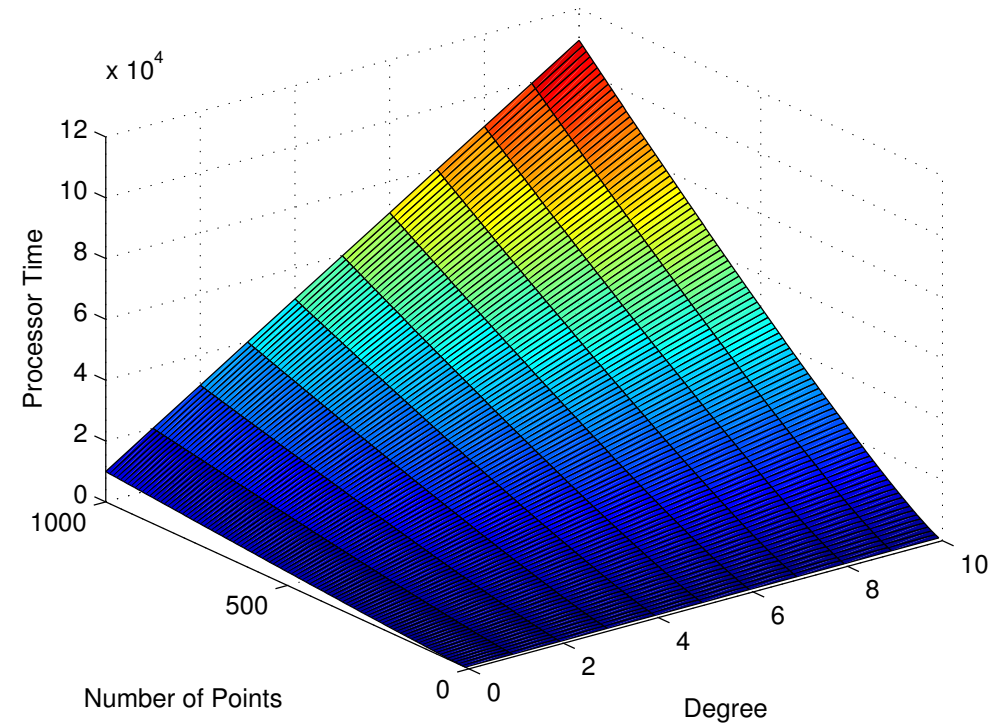
Computational Effort  $O((d+1)n \log_2 n)$ , Memory Required  $O((d+1)n \log_2 n)$



Computational Time – Processor Time Vector W



Computational Time – Processor Time – Theoretical Vector W



In file `changeable.c` -

- a) put the admissibility condition in `admissible_ClusterPair`
- b) put kernel function in `kernel_function`
- c) put **NONE** as return values in `get_iOperation`
- d) put **100** as return values in `get_iNumber of Times`

Output File from C++:

OutputFile from MATLAB :

Time.txt  
 Time\_Vector\_W.eps, Time\_Theoretical\_Vector\_W.eps, Time\_Chebyshev.eps,  
 Time\_Theoretical\_Chebyshev.eps, Time\_Far.eps, Time\_Theoretical\_Far.eps, Time\_Near.eps,  
 Time\_Theoretical\_Near.eps, Time\_All.eps, Time\_Theoretical\_All.eps  
 Choice - 2, NumberofPts - 10 1000 10, Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1  
 Degree - 0 10 1, Admissibility Coefficient - **0.5 0.5 1.0**

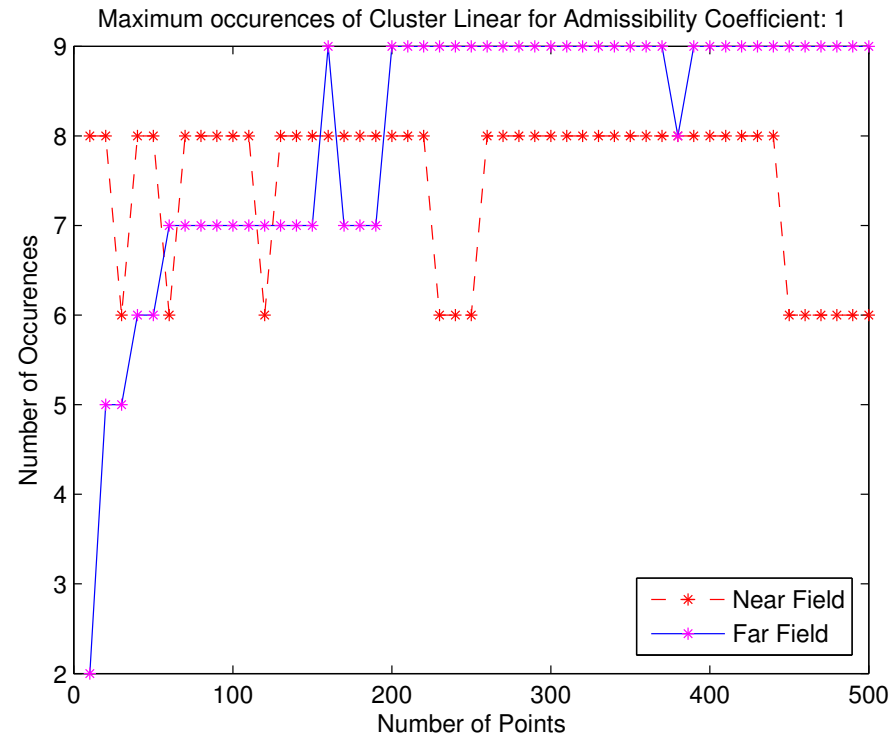
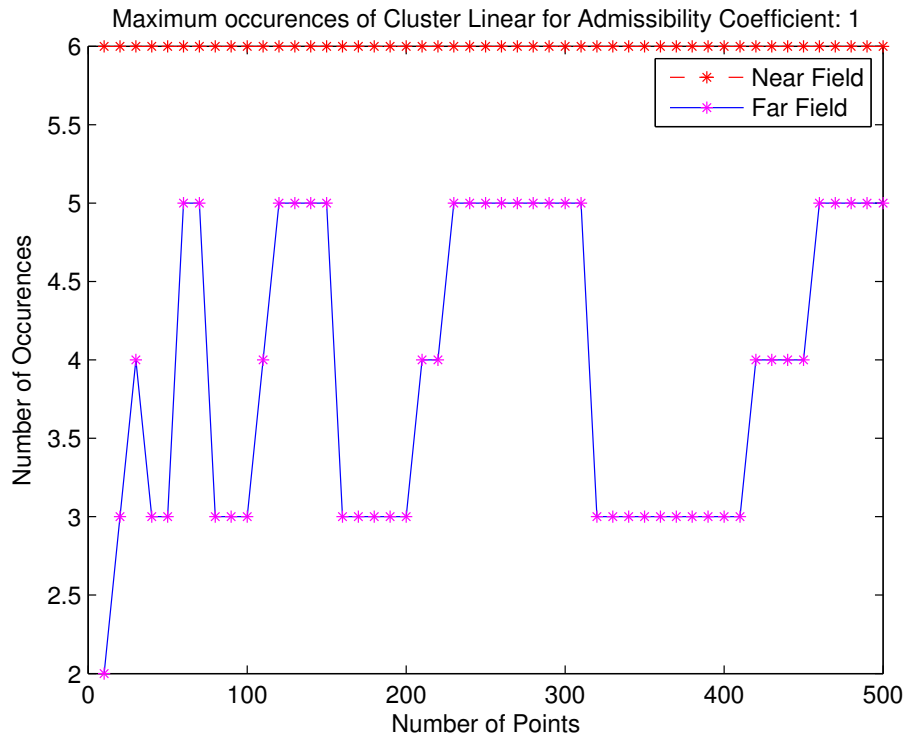
1) Run `main.cpp`, input as follows :

2) Run `time_mesh.m` and `time_mesh_theoretical` from Matlab

②: Compute  $X_{\sigma,\mu}$  as in (11.4.1),  $\sigma \in T_x, \mu \in T_y, (\sigma, \mu) \in P^{\text{far}}$ .

Uniform distribution of points  $\max\{\#\{\mu \in T_y: (\sigma, \mu) \in P^{\text{far}}\}, \sigma \in T_x\} = O(1)$

Example 11.4.4 (Occurrence of clusters in partition rectangles).



$x = (0:1/n:(1-1/n))$ ,  $\eta = 1$   
In file changeable.c

- a) put the admissibility condition in `admissible_ClusterPair`
- b) put kernel function in `kernel_function`
- c) put **NONE** as return values in `get_iOperation`
  - 1) Run `main.cpp`, input as follows  
Choice - 2, NumberofPts - 10 500 10,  
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1  
Degree - 0 0 1, Admissibility Coefficient - 1 1 1  
Output File from C++: `Count.txt`  
OutputFile from MATLAB : `MaxCluster_1.eps`
  - 2) Run `count_indiv.m` from Matlab, arguments:

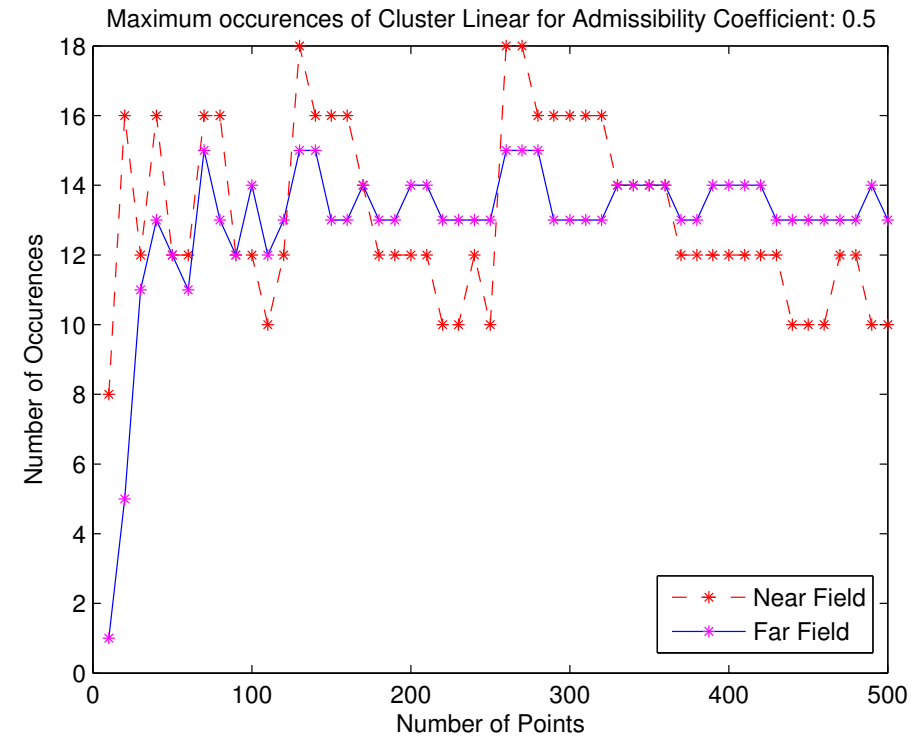
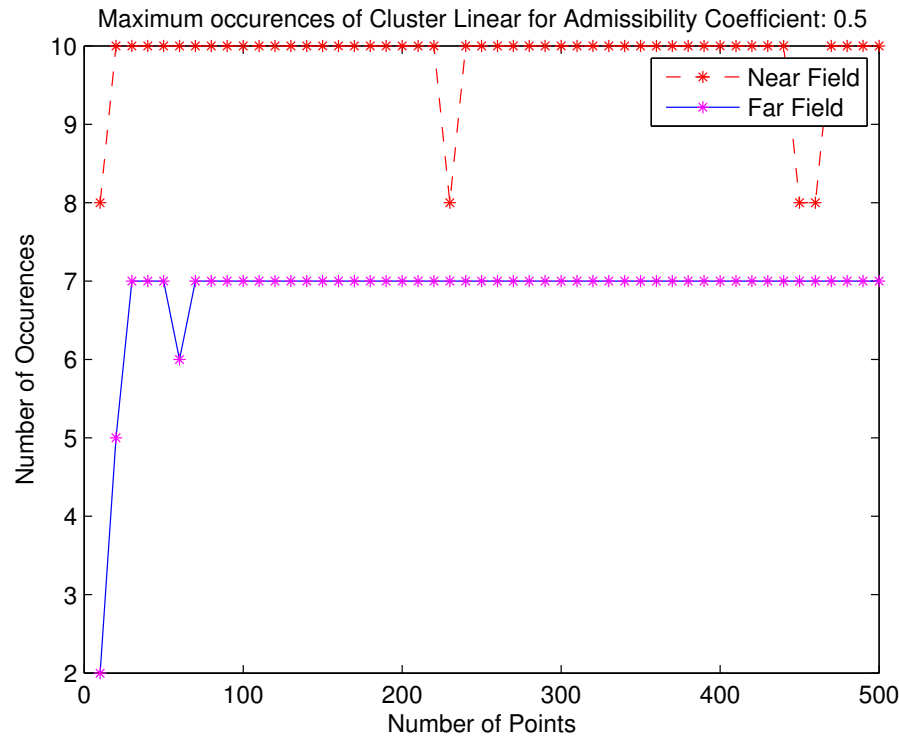
$x = \text{sqrt}(0:1/n:(1-1/n))$ ,  $\eta = 1$   
In file changeable.c

- a) put the admissibility condition in `admissible_ClusterPair`
- b) put kernel function in `kernel_function`
- c) put **SQRT** as return values in `get_iOperation`
  - 1) Run `main.cpp`, input as follows  
Choice - 2, NumberofPts - 10 500 10,  
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1  
Degree - 0 0 1, Admissibility Coefficient - 1 1 1  
Output File from C++: `Count.txt`  
OutputFile from MATLAB : `MaxCluster_1.eps`
  - 2) Run `count_indiv.m` from Matlab, arguments:

Uniform distribution of  $x_i, y_j$ : for every  $\sigma \in T_x$

$$\#\{\mu \in T_y: (\sigma, \mu) \in \text{Pfar}\} = O(1),$$

each cluster contributes only a small number of partitions of rectangle



$x = (0 : 1/n : (1-1/n)) ; \eta = 0.5$   
In file changeable.c

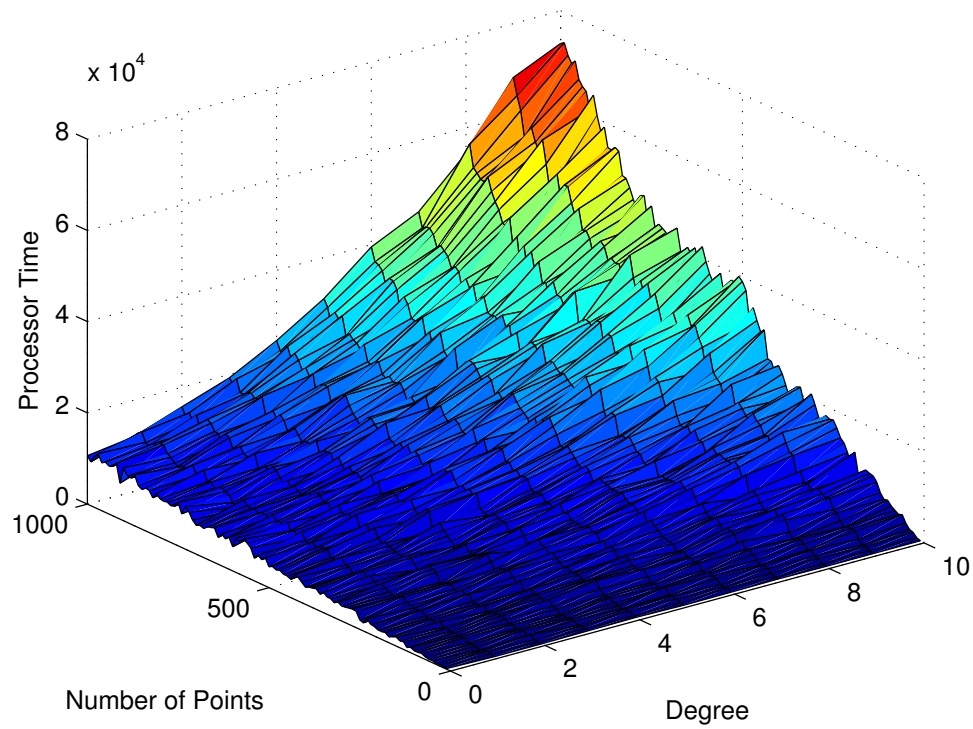
- a) put the admissibility condition in `admissible_ClusterPair`
- b) put kernel function in `kernel_function`
- c) put **NONE** as return values in `get_iOperation`
  - 1) Run `main.cpp`, input as follows  
Choice - 2, NumberofPts - 10 500 10,  
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1  
Degree - 0 0 1, Admissibility Coefficient - **0.5 0.5 1**  
Output File from C++: `Count.txt`  
OutputFile from MATLAB : `MaxCluster_0.5.eps`
  - 2) Run `count_indiv.m` from Matlab, arguments:

$x = \text{sqrt}(0 : 1/n : (1-1/n)) ; \eta = 0.5$   
In file changeable.c

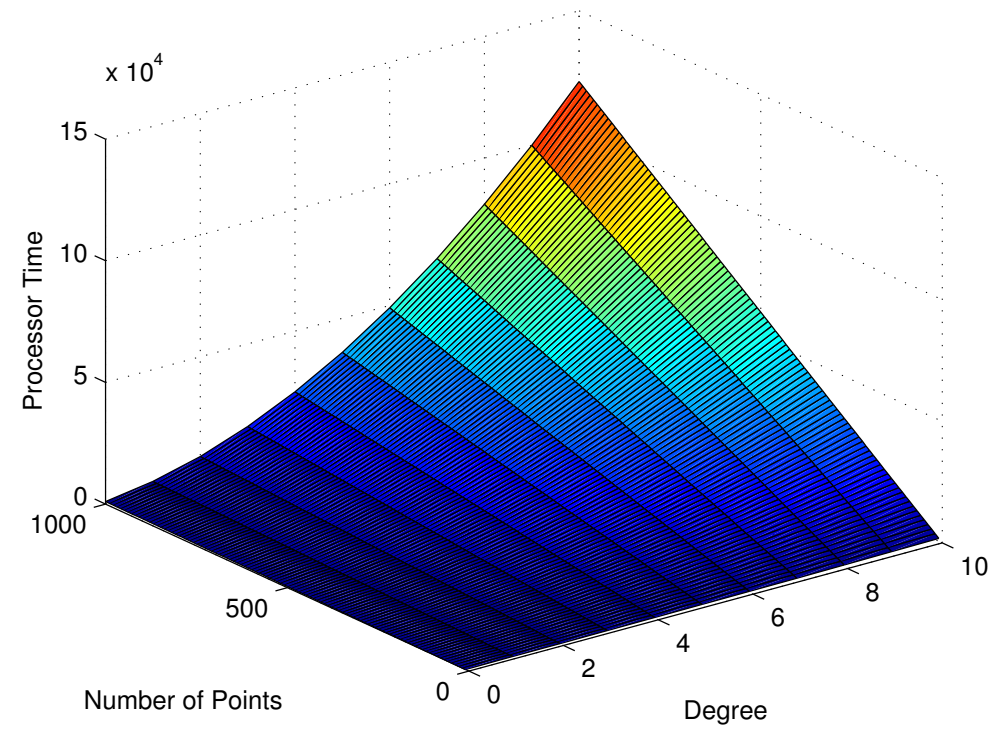
- a) put the admissibility condition in `admissible_ClusterPair`
- b) put kernel function in `kernel_function`
- c) put **SQRT** as return values in `get_iOperation`
  - 1) Run `main.cpp`, input as follows  
Choice - 2, NumberofPts - 10 500 10,  
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1  
Degree - 0 0 1, Admissibility Coefficient - **0.5 0.5 1**  
Output File from C++: `Count.txt`  
OutputFile from MATLAB : `MaxCluster_0.5.eps`
  - 2) Run `count_indiv.m` from Matlab, arguments:

Computational Effort for step ②:  $O(n \log_2 n (d + 1)^2)$ , Memory Required  $O(n \log_2 n (d + 1)^2)$

Computational Time – Processor Time Chebyshev Nodes



Computational Time – Processor Time – Theoretical Chebyshev Nodes





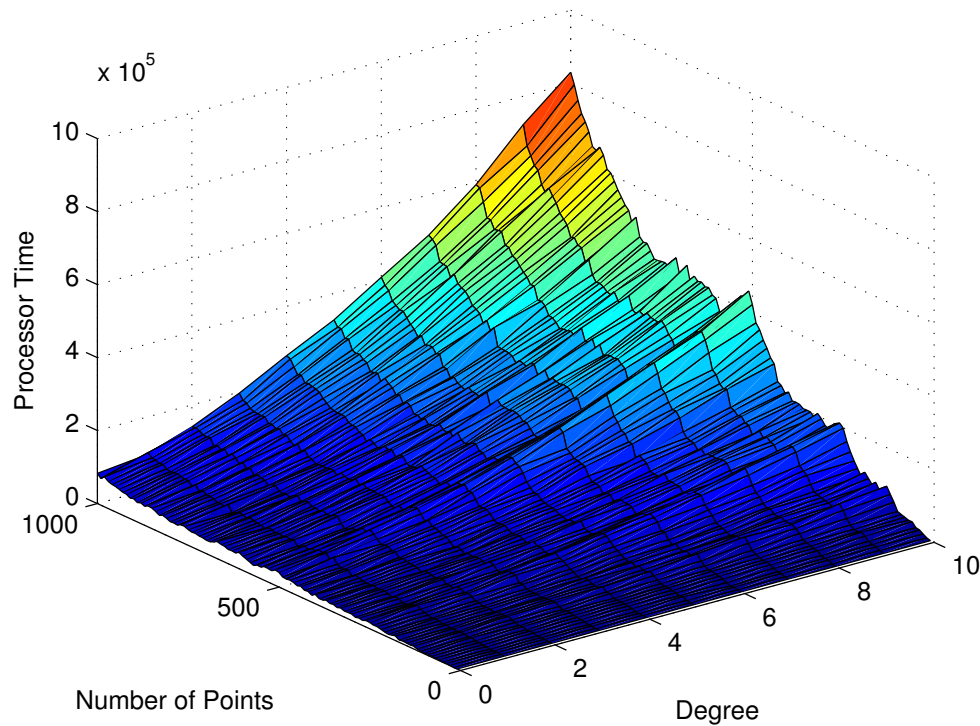
③: Far field analysis  $\hat{=}$  Sum

$$\sum_{\substack{\sigma \in T_x \\ x_i \in \sigma}} \sum_{\substack{\mu \in T_y \\ (\sigma, \mu) \in P^{\text{far}}}} \mathbf{V}_\sigma \mathbf{X}_{\sigma, \mu} \mathbf{w}_\mu$$

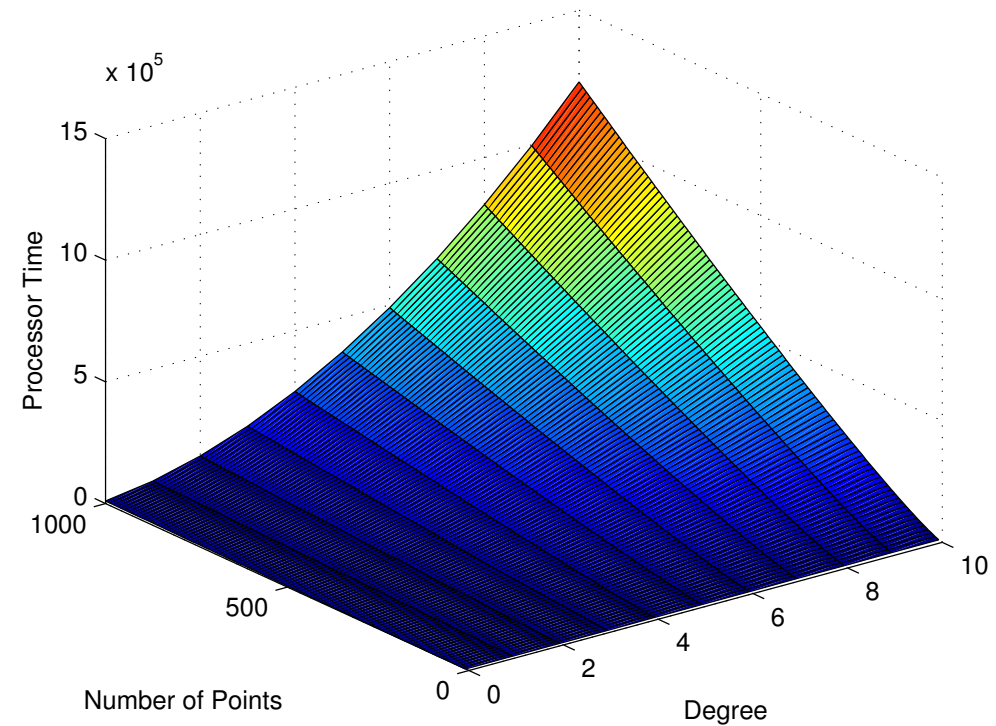
for each  $\sigma \in T_x$  { Compute  $\mathbf{V}_\sigma$ ;  $\mathbf{s} := 0$   
 for each  $\mu \in T_y, (\sigma, \mu) \in P^{\text{far}}$  {  $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{X}_{\sigma, \mu} \mathbf{w}_\mu$  }  
 $\mathbf{f}_{|\sigma} \leftarrow \mathbf{f}_{|\sigma} + \mathbf{V}_\sigma \mathbf{s}$  }

Computational Effort  $O(n \log_2 n (d + 1)^2)$ , Memory Required  $O(n(d + 1))$

Computational Time – Processor Time Far Field



Computational Time – Processor Time – Theoretical Far Field

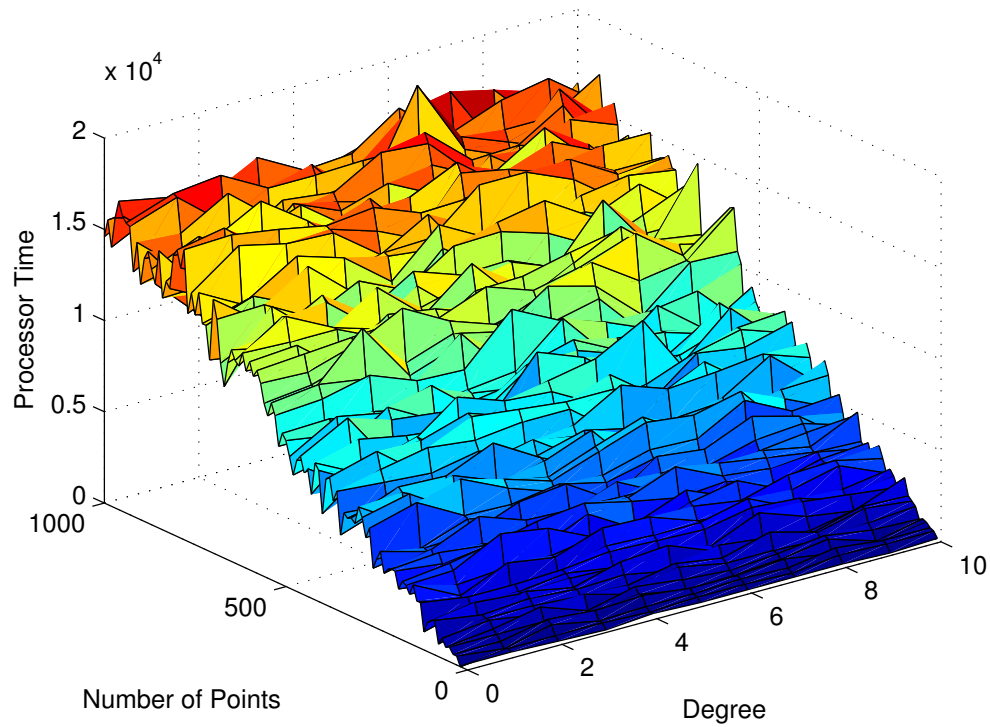


④: Near field Computation

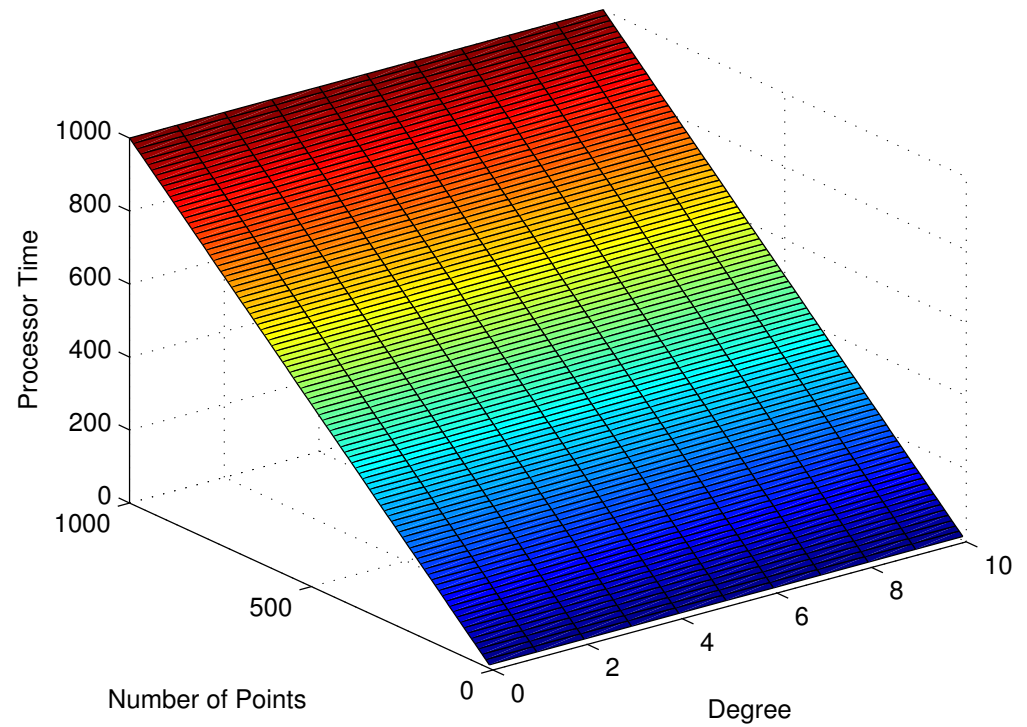
for each  $(\sigma, \mu) \in P^{\text{near}}$  { for each  $i: x_i \in \sigma$  {  $\mathbf{f}_i \leftarrow \mathbf{f}_i + \sum_{j: y_j \in \mu} G(x_i, y_j)c_j$  } }

Computational Effort  $O(n)$

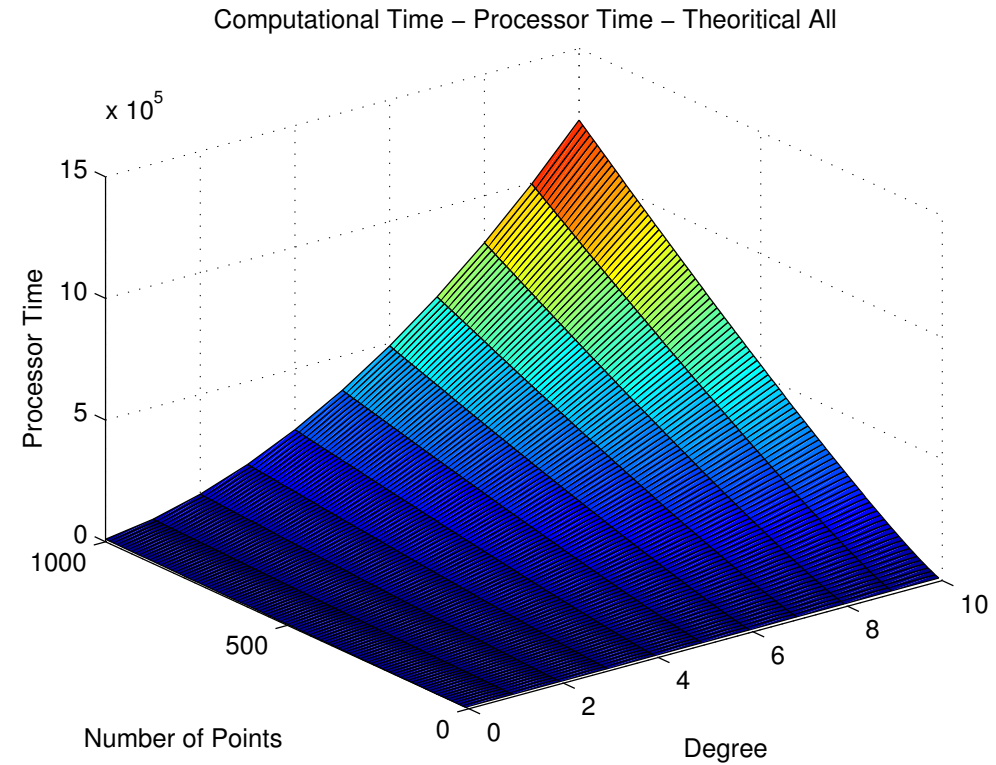
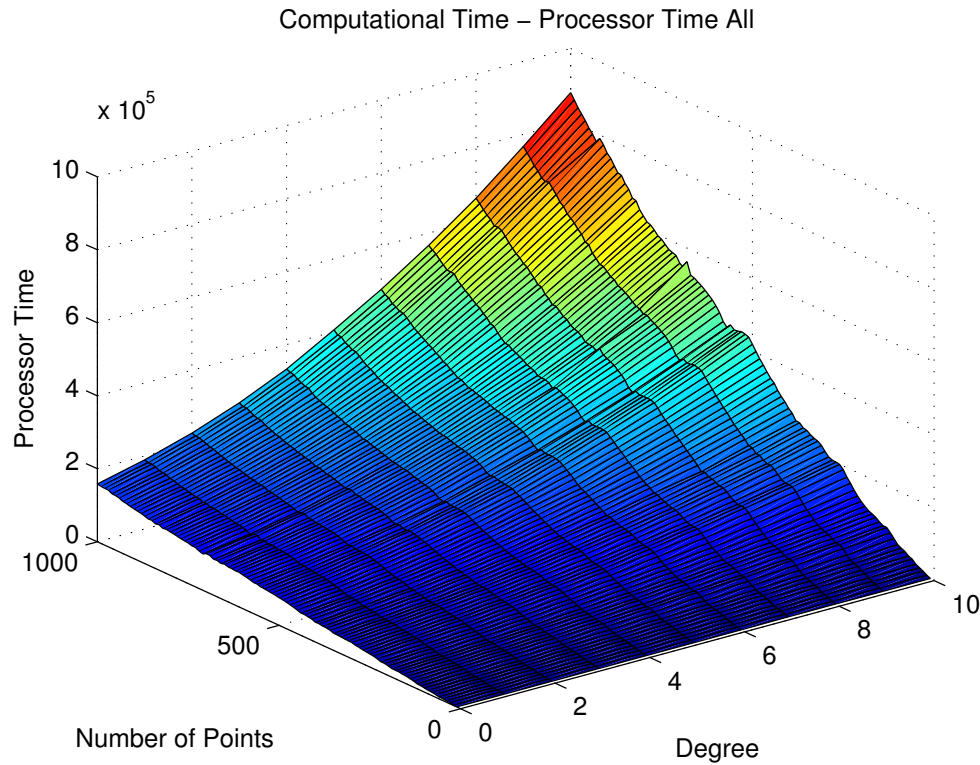
Computational Time – Processor Time Near Field



Computational Time – Processor Time – Theoretical Near Field



Total Computational Effort/Total Memory Required  $O((d + 1)^2 n \log_2 n)$



*Remark 11.4.5* (Convergence of cluster approximation).

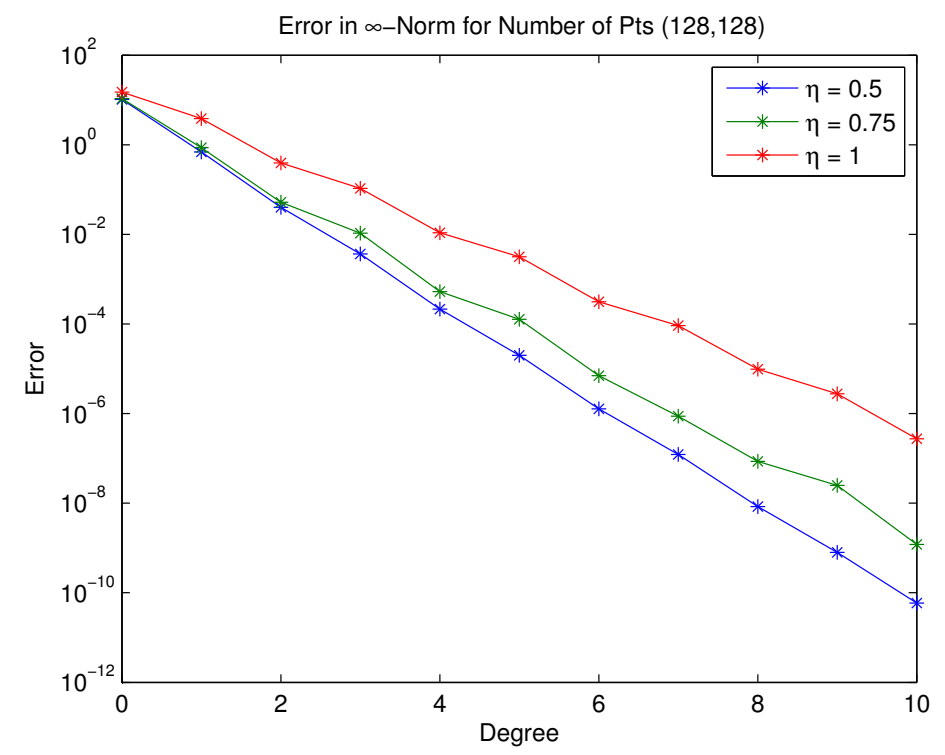
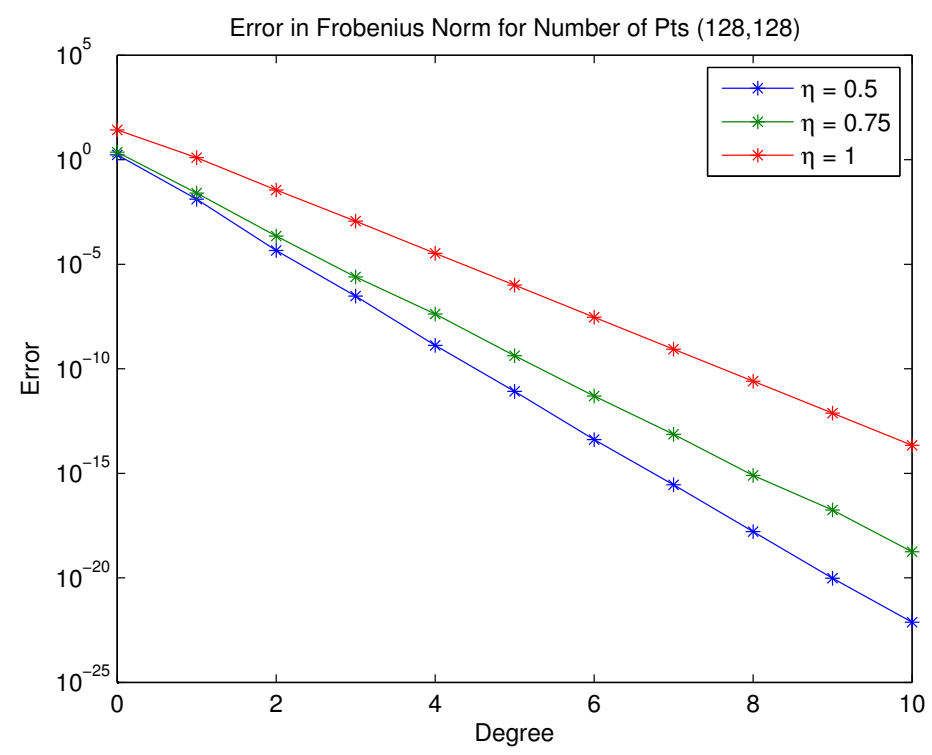
if  $G(x, y)$  analytic ( $\rightarrow$  Def. 9.2.20) in  $\{(x, y): x \neq y\}$  then

Cluster approximation inherits **exponential convergence** from Chebyshev interpolation



# Example 11.4.6 (Convergence of clustering approximation with collocation matrix).

- $x = 0:1/128:1;$ ,  $G(x, y) = |x - y|^{-1}$  for  $x \neq y$ ,  $G(x, x) = 0$ .



In file changeable.c

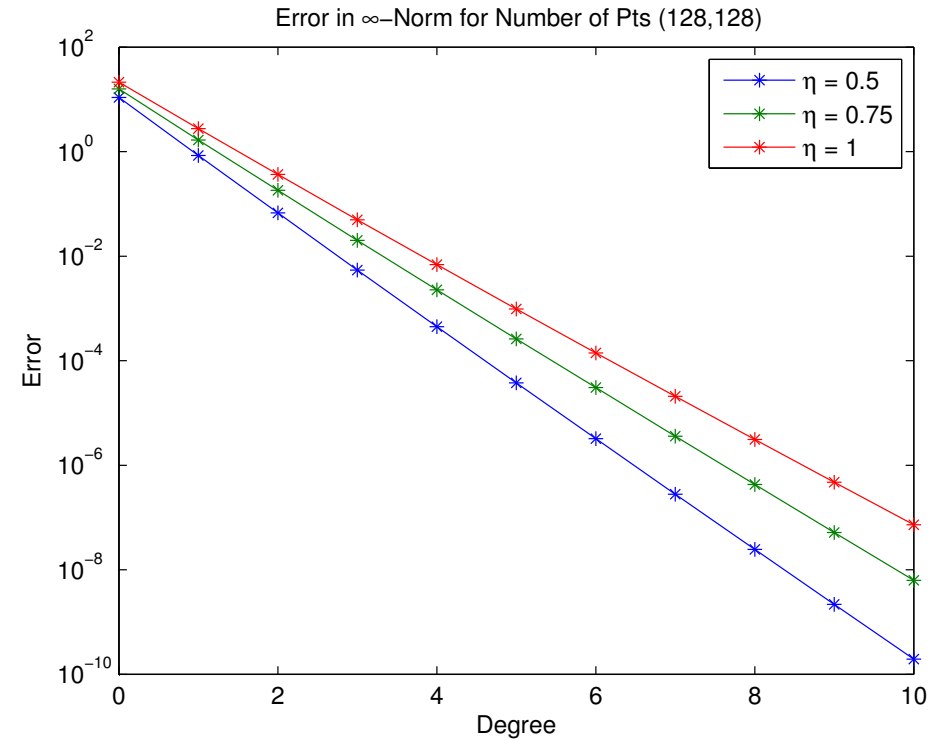
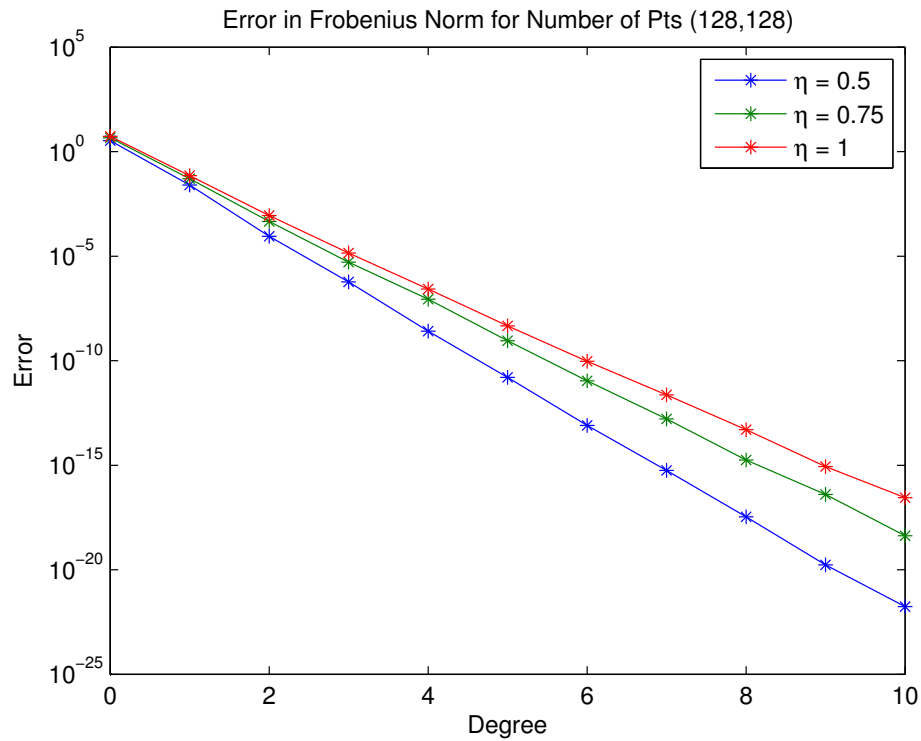
- put the admissibility condition in `admissible_ClusterPair`
- put kernel function in `kernel_function`
- put **NONE** as return values in `get_iOperation`
  - Run `main.cpp`, input as follows  
 Choice - 2, NumberofPts - 128 128 1,  
 Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1  
 Degree - 0 10 1, Admissibility Coefficient - **0.5 1.0 0.25**

Output File from C++: `Error.txt`  
 OutputFile from MATLAB : `Error_in_Frobenius_Norm_128_128.eps`

- Run `error_plot.m` from Matlab, arguments:  
 0 (argument here is for specifying the index of Number of Pts.)



•  $x = \text{sqrt}(0:1/128:1)$ ; ,  $G(x, y) = |x - y|^{-1}$  for  $x \neq y$ ,  $G(x, x) = 0$ .



In file changeable.c

a) put the admissibility condition in `admissible_ClusterPair`

b) put kernel function in `kernel_function`

c) put **SQRT** as return values in `get_iOperation`

1) Run `main.cpp`, input as follows

Choice - 2, NumberofPts - 128 128 1,

Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1

Degree - 0 10 1, Admissibility Coefficient - **0.5 1.0 0.25**

Output File from C++: `Error.txt`

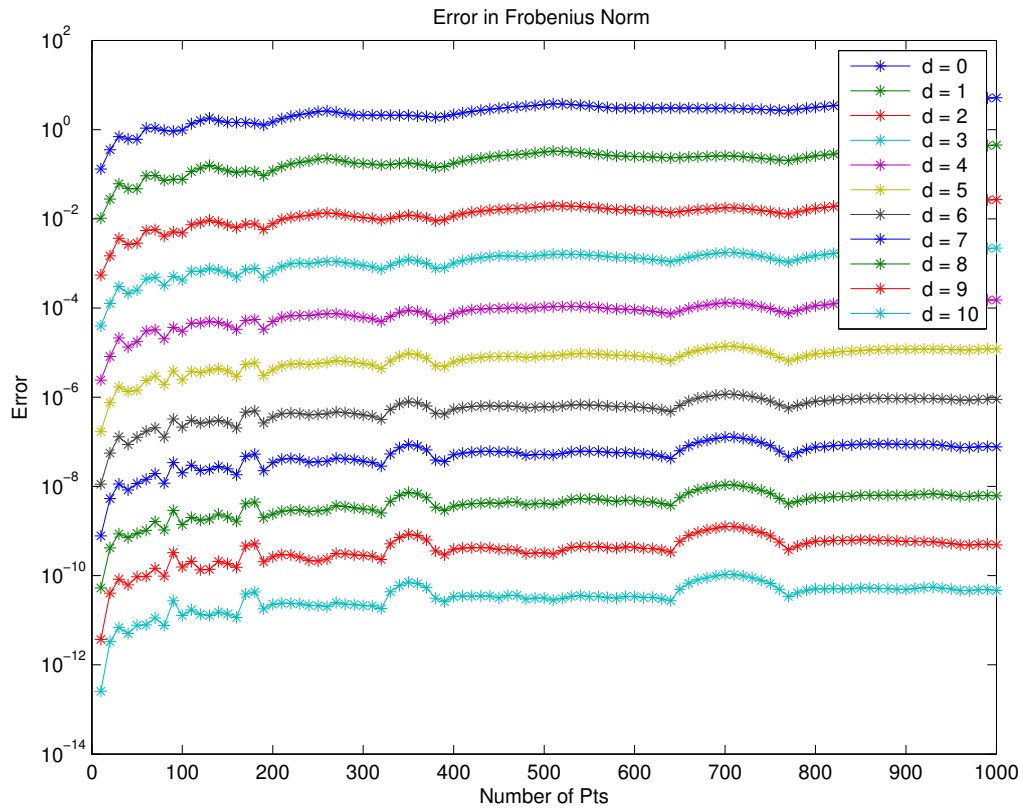
OutputFile from MATLAB : `Error_in_Frobenius_Norm_128_128.eps`

2) Run `error_plot.m` from Matlab, arguments:

0 (argument here is for specifying the index of Number of Pts.)



# ● Scaled Frobenius Error v/s Number of Points



- In file changeable.c
- a) put the admissibility condition in `admissible_ClusterPair`
  - b) put kernel function in `kernel_function`
  - c) put **NONE** as return values in `get_iOperation`
    - 1) Run `main.cpp`, input as follows  
Choice - 2, NumberofPts - 10 1000 10,  
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1  
Degree - 0 10 1, Admissibility Coefficient - 0.5 0.5 1.0  
Output File from C++: `Error.txt`
    - 2) Run `error_plot_Degree.m` from Matlab
- OutputFile from MATLAB : `Error_in_Frobenius_Norm_Degree.eps`

**Example 11.4.7 (Analysis of trigonometric polynomials).** [19]

Given:  $\{t_0, \dots, t_{n-1}\} \subset [0, 1[$ ,  $\alpha_{-m+1}, \dots, \alpha_m \in \mathbb{C}$ ,  $n = 2m$ ,  $m \in \mathbb{N}$ , compute

$$c_k := p(t_k), \quad j = 0, \dots, n-1 \quad \text{for} \quad p(t) := \underbrace{\sum_{j=-m+1}^m \alpha_j e^{-2\pi i j t}}_{\text{Trigonometric Polynomial}}.$$

Discrete Fourier transformation (DFT)  $\rightarrow$  Section 8.2:

$$f_l := \sum_{j=-m+1}^m \alpha_j e^{-\frac{2\pi i}{n} j l} \quad \stackrel{\text{Lemma 8.2.10}}{\Leftrightarrow} \quad \alpha_j = \frac{1}{n} \sum_{l=-m+1}^m f_l e^{\frac{2\pi i}{n} l j}$$

FFT:  $f_l, l = -m+1, \dots, m$ , calculated with computational effort  $O(n \log_2 n)$

$$\begin{aligned} c_k &= \sum_{j=-m+1}^m \alpha_j e^{2\pi i j t_k} = \sum_{j=-m+1}^m \frac{1}{n} \left( \sum_{l=-m+1}^m f_l e^{\frac{2\pi i}{n} l j} \right) e^{-2\pi i j t_k} \\ &= \frac{1}{n} \sum_{l=-m+1}^m f_l \sum_{j=-m+1}^m e^{-2\pi i j (t_k - l/n)} \\ &= \frac{1}{n} \sum_{l=-m+1}^m f_l e^{-2\pi i (t_k - l/n)(-m+1)} \frac{1 - e^{-2\pi i n (t_k - l/n)}}{1 - e^{-2\pi i (t_k - l/n)}} \end{aligned}$$

$$= \frac{1}{n} \sum_{l=-m+1}^m f_l e^{-\pi i t_k} \sin(\pi n t_k) \frac{1}{\sin(\pi(t_k - l/n))} (-1)^l e^{-\pi i l/n} .$$

$$\mathbf{c} = \text{diag} \left( \frac{\sin(\pi n t_k)}{e^{\pi i t_k}} \right)_{k=-m+1}^m \mathbf{M} \text{diag} \left( (-1)^l e^{-\pi i l/n} \right)_{l=-m+1, \dots, m} \mathbf{f} .$$

in accordance with collocation matrix (11.1.1)

$$\mathbf{M} := \left( \frac{1}{\sin(\pi(t_k - l/n))} \right)_{\substack{k=-m+1, \dots, m \\ l=-m+1, \dots, m}} \in \mathbb{R}^{2n, 2n} .$$

Approximative analysis of Clustering algorithms ! → USFFT



*Remark 11.4.8.* Clustering approximation example for **fast approximative** implementation of algorithms of numerical linear algebra (→ Chapter ??) → Trend in numerical linear algebra ?

Problem	Exact/direct Method	Approximate/iterative Method
Linear equation systems	Gaussian Elimination	CG-like iterative solvers → Sect. ??
Eigenvalue problem	Transformation methods	Krylov-Subspace method → Sect. ??
Collocations matrix × Vector	BLAS (SAXPY)	Clustering techniques



With Clustering approximation related, or generalizations thereof:

- $\mathcal{H}$ -Matrix-Techniques [6]
- $\mathcal{H}^2$ -Matrix-Techniques [30]
- Multipol-Methods [54, 26]  $\rightarrow$  further “Millennium algorithm”  
[http://orion.math.iastate.edu/burkardt/misc/algorithms\\_dongarra.htm](http://orion.math.iastate.edu/burkardt/misc/algorithms_dongarra.htm)

# 12

## Single Step Methods

### 12.1 Initial value problems (IVP) for ODEs

Acronym:

**ODE** = **ordinary differential equation**

R. Hiptmair  
rev 38286,  
May 3,  
2011

Some grasp of the meaning and theory of ordinary differential equations (ODEs) is indispensable for understanding the construction and properties of numerical methods. Relevant information can be found in [63, Sect. 5.6, 5.7, 6.5].



## Supplementary and further reading:

Books dedicated to numerical methods for ordinary differential equations:


- [16] excellent textbook, but geared to the needs of students of mathematics.
- [32] and [33] : *the* standard reference.
- [31]: wonderful book conveying deep insight, with emphasis on mathematical concepts.

### 12.1.1 Examples

*Example* 12.1.1 (Growth with limited resources). [2, Sect. 1.1], [35, Ch. 60]

$y : [0, T] \mapsto \mathbb{R}$ : bacterial population density as a function of time

$$\dot{y} = f(y) := (\alpha - \beta y) y \quad (12.1.2)$$

 Notation (Newton): dot  $\dot{\hat{\phantom{y}}}$  (total) derivative with respect to time  $t$

- $y \hat{=}$  population density,  $[y] = \frac{1}{\text{m}^2}$
- growth rate  $\alpha - \beta y$  with growth coefficients  $\alpha, \beta > 0$ ,  $[\alpha] = \frac{1}{\text{s}}$ ,  $[\beta] = \frac{\text{m}^2}{\text{s}}$ : decreases due to more fierce competition as population density increases.

Note: we can only compute a solution of (12.1.2), when provided with an **initial value**  $y(0)$ .



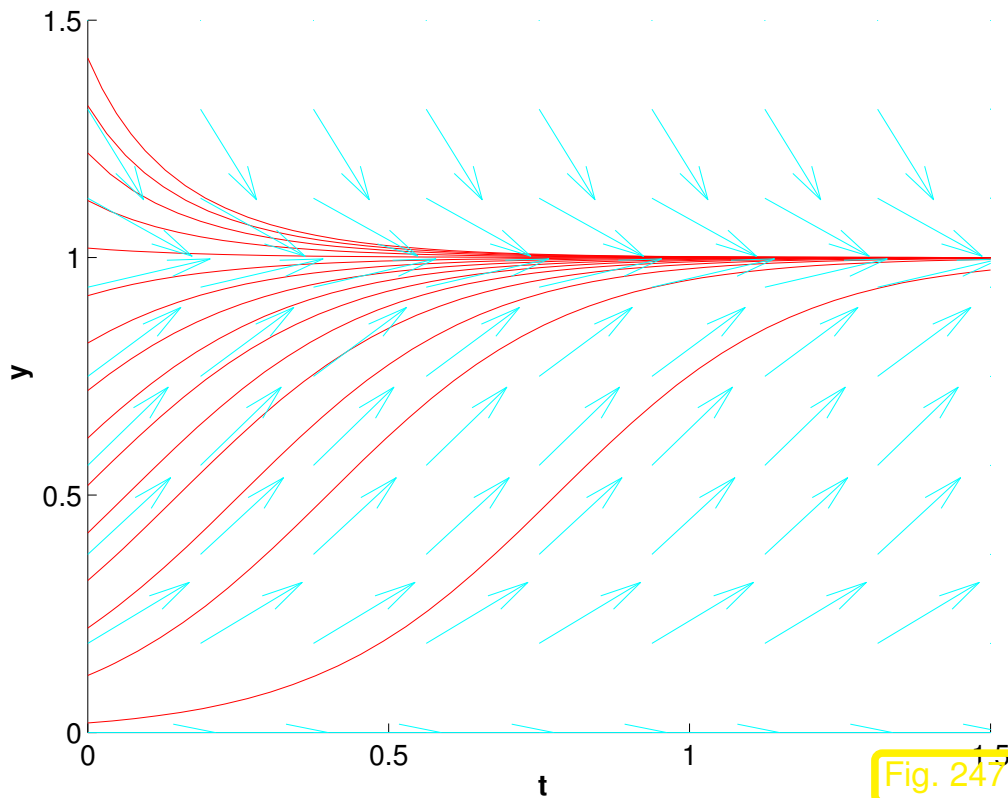


Fig. 247

Solution for different  $y(0)$  ( $\alpha, \beta = 5$ )

By separation of variables

→ solution of (12.1.2)

for  $y(0) = y_0 > 0$

$$y(t) = \frac{\alpha y_0}{\beta y_0 + (\alpha - \beta y_0) \exp(-\alpha t)}, \quad (12.1.3)$$

for all  $t \in \mathbb{R}$

$f'(y^*) = 0$  for  $y^* \in \{0, \alpha/\beta\}$ , which are the **stationary points** for the ODE (12.1.2). If  $y(0) = y^*$  the solution will be constant in time.



Terminology:

(12.1.2) = **autonomous** ODE

An ODE is called autonomous, if the right hand side does not depend on the independent variable (usually denoted by  $t$ ), but only on the state (usually denoted by  $y/\mathbf{y}$ )

*Example 12.1.4* (Predator-prey model). [2, Sect. 1.1], [31, Sect. 1.1.1], [35, Ch. 60], [13, Ex. 11.3]

Predators and prey coexist in an ecosystem. Without predators the population of prey would be governed by a simple exponential growth law. However, the growth rate of prey will decrease with increasing numbers of predators and, eventually, become negative. Similar considerations apply to the predator population and lead to an ODE model.

Model: autonomous **Lotka-Volterra ODE**:

$$\begin{aligned} \dot{u} &= (\alpha - \beta v)u \\ \dot{v} &= (\delta u - \gamma)v \end{aligned} \Leftrightarrow \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \text{with} \quad \mathbf{y} = \begin{pmatrix} u \\ v \end{pmatrix}, \quad \mathbf{f}(\mathbf{y}) = \begin{pmatrix} (\alpha - \beta v)u \\ (\delta u - \gamma)v \end{pmatrix}. \quad (12.1.5)$$

population sizes:

$u(t) \rightarrow$  no. of prey at time  $t$ ,

$v(t) \rightarrow$  no. of predators at time  $t$

vector field  $\mathbf{f}$  for Lotka-Volterra ODE ▷

Solution curves are trajectories of particles carried along by velocity field  $\mathbf{f}$ .

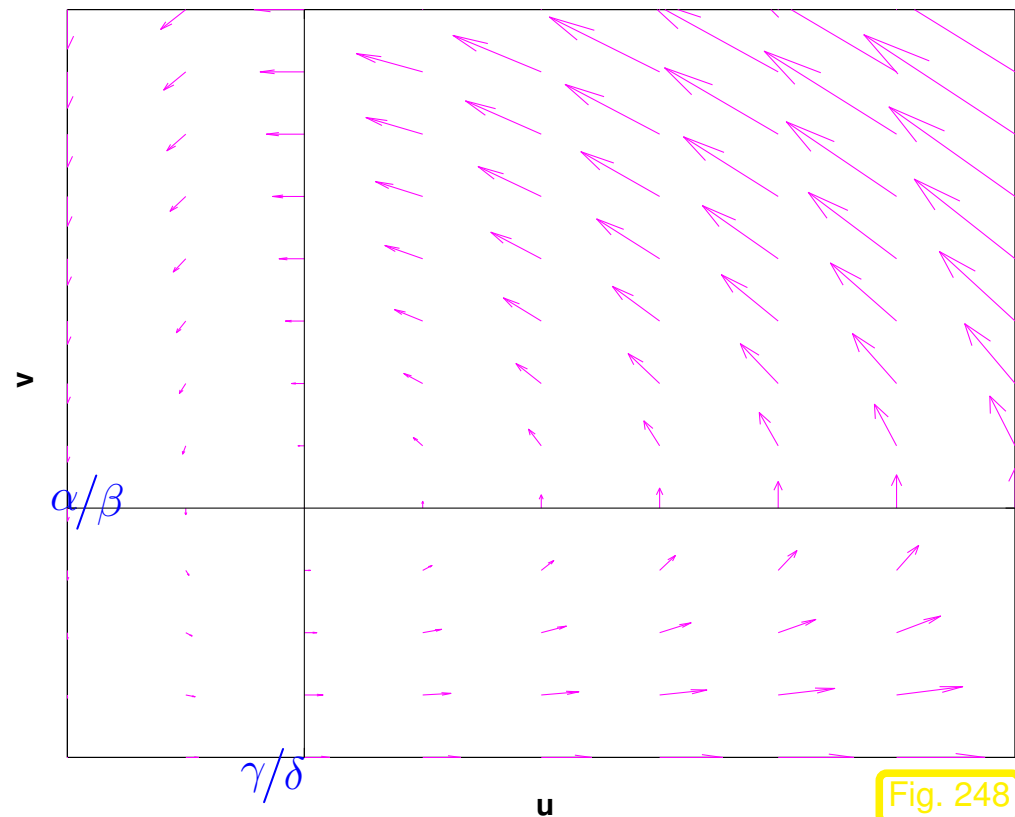


Fig. 248

Parameter values for Fig. 248:  $\alpha = 2, \beta = 1, \delta = 1, \gamma = 1$

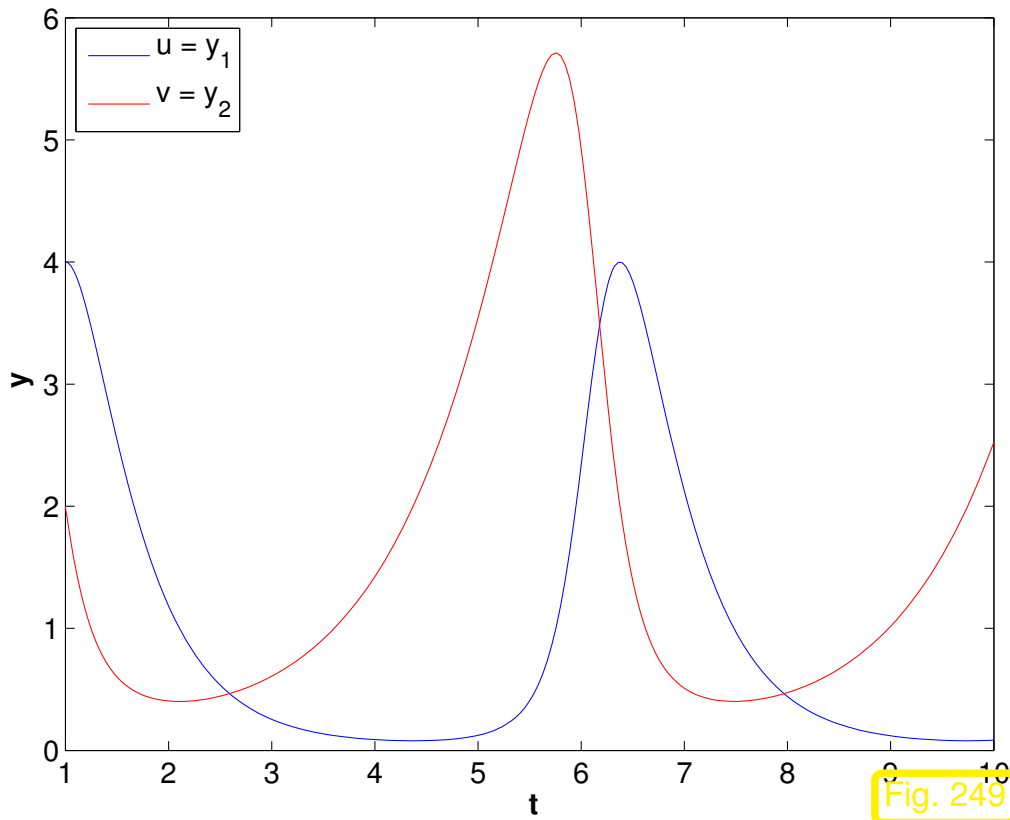


Fig. 249

Solution  $\begin{pmatrix} u(t) \\ v(t) \end{pmatrix}$  for  $\mathbf{y}_0 := \begin{pmatrix} u(0) \\ v(0) \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$

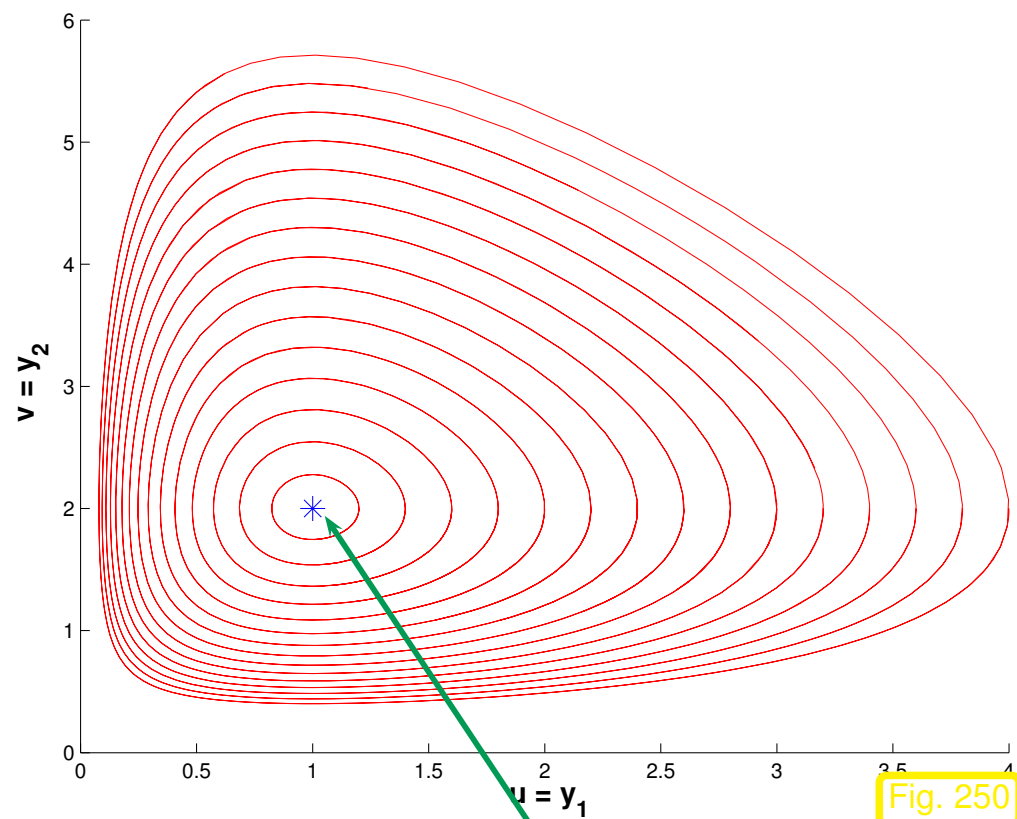


Fig. 250

Solution curves for (12.1.5)

stationary point

Parameter values for Figs. 250, 249:  $\alpha = 1, \beta = 1, \delta = 1, \gamma = 2$



Example 12.1.6 (Heartbeat model). → [14, p. 655]

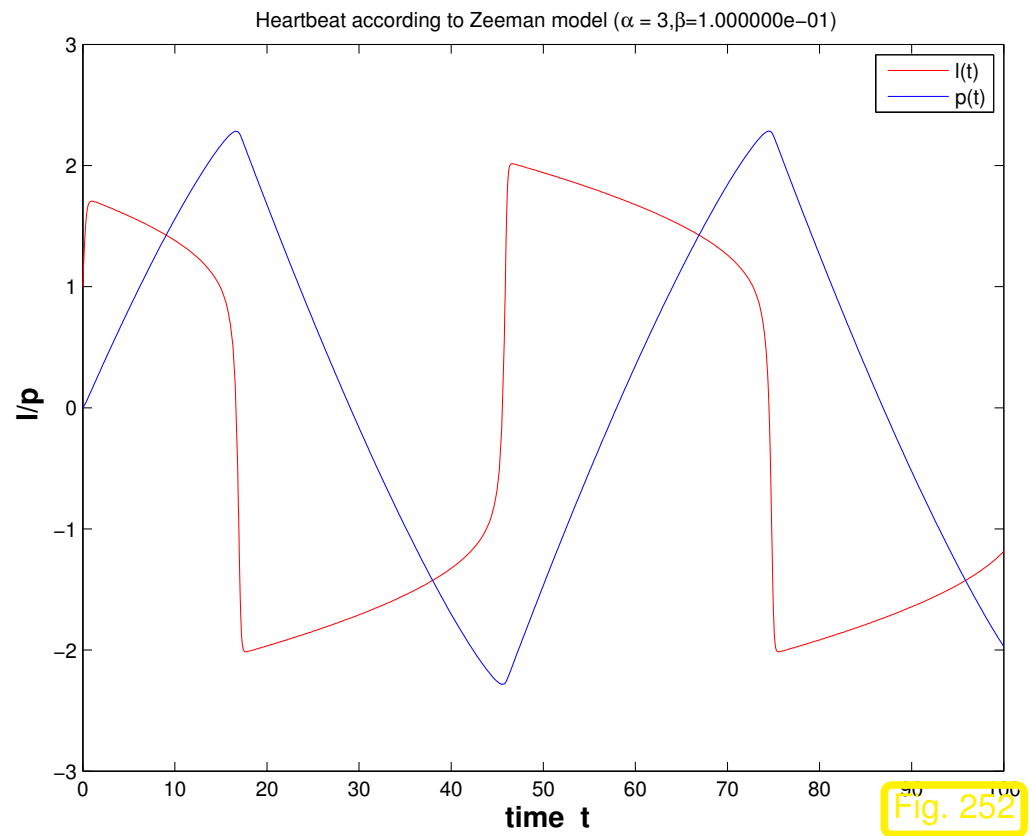
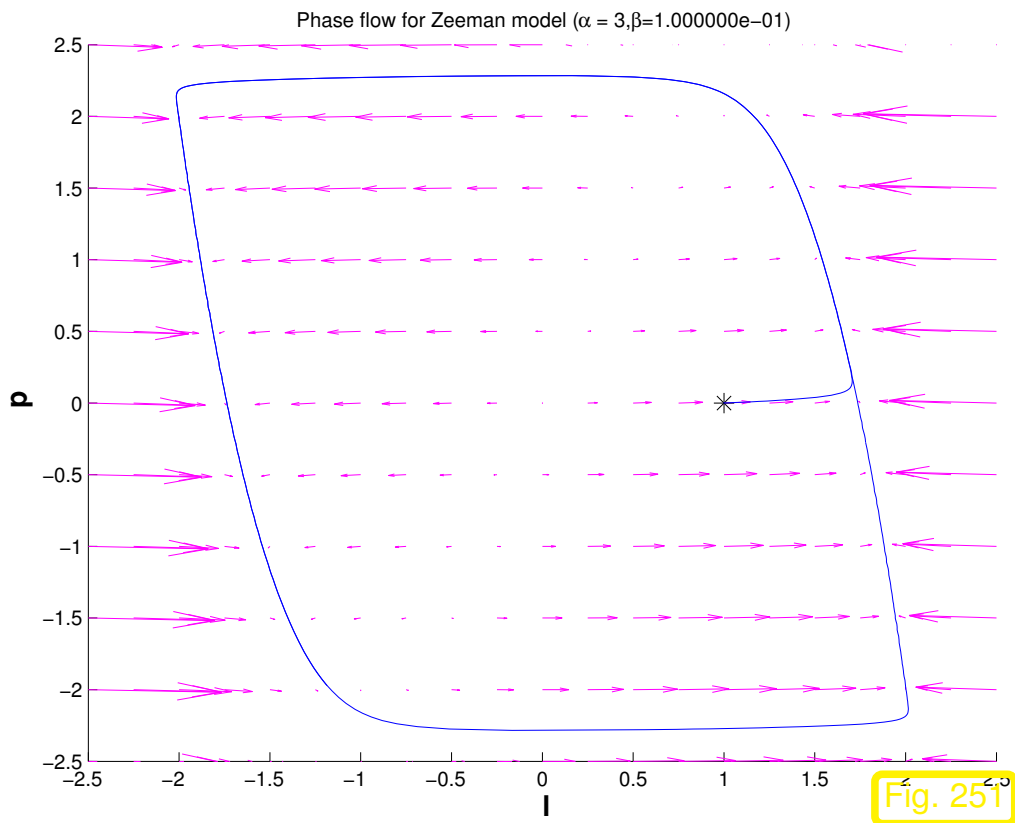
State of heart described by quantities:  $l = l(t) \hat{=} \text{length of muscle fiber}$   
 $p = p(t) \hat{=} \text{electro-chemical potential}$

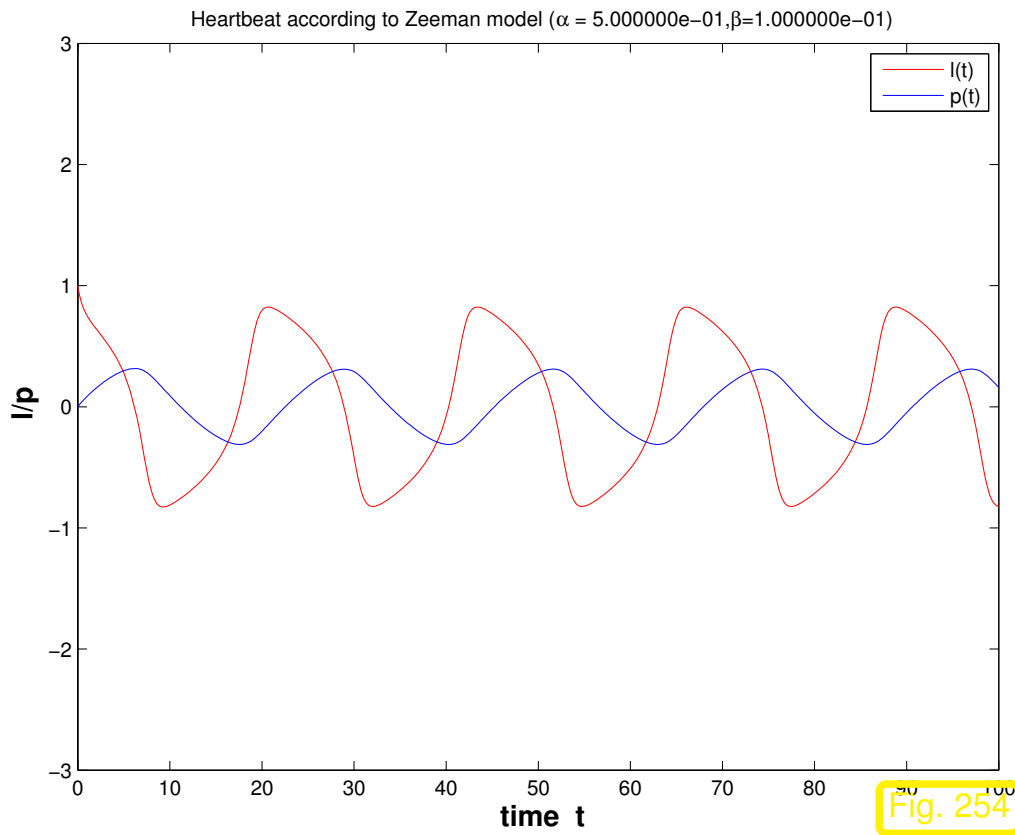
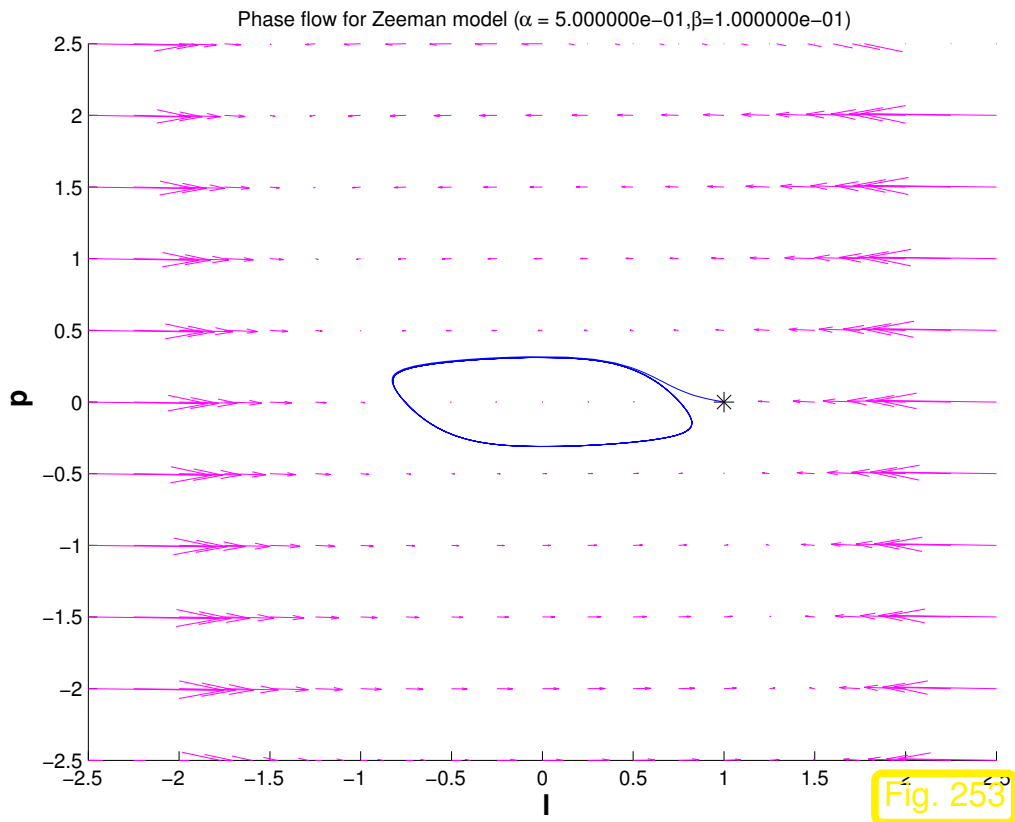
Phenomenological model: 
$$\begin{aligned} \dot{l} &= -(l^3 - \alpha l + p), \\ \dot{p} &= \beta l, \end{aligned} \quad (12.1.7)$$

with parameters:  $\alpha \hat{=} \text{pre-tension of muscle fiber}$   
 $\beta \hat{=} \text{(phenomenological) feedback parameter}$

This is the so-called Zeeman model: it is a phenomenological model entirely based on macroscopic observations without relying on knowledge about the underlying molecular mechanisms.

Vector fields and solutions for different choices of parameters:





Observation:  $\alpha \ll 1 \blacktriangleright$  atrial fibrillation



Example 12.1.8 (Transient circuit simulation). [35, Ch. 64]

Transient nodal analysis, cf. Ex. 2.6.3:

Kirchhoff current law

$$i_R(t) - i_L(t) - i_C(t) = 0. \quad (12.1.9)$$

Transient constitutive relations:

$$i_R(t) = R^{-1}u_R(t), \quad (12.1.10)$$

$$i_C(t) = C \frac{du_C}{dt}(t), \quad (12.1.11)$$

$$u_L(t) = L \frac{di_L}{dt}(t). \quad (12.1.12)$$

Given: source voltage  $U_s(t)$

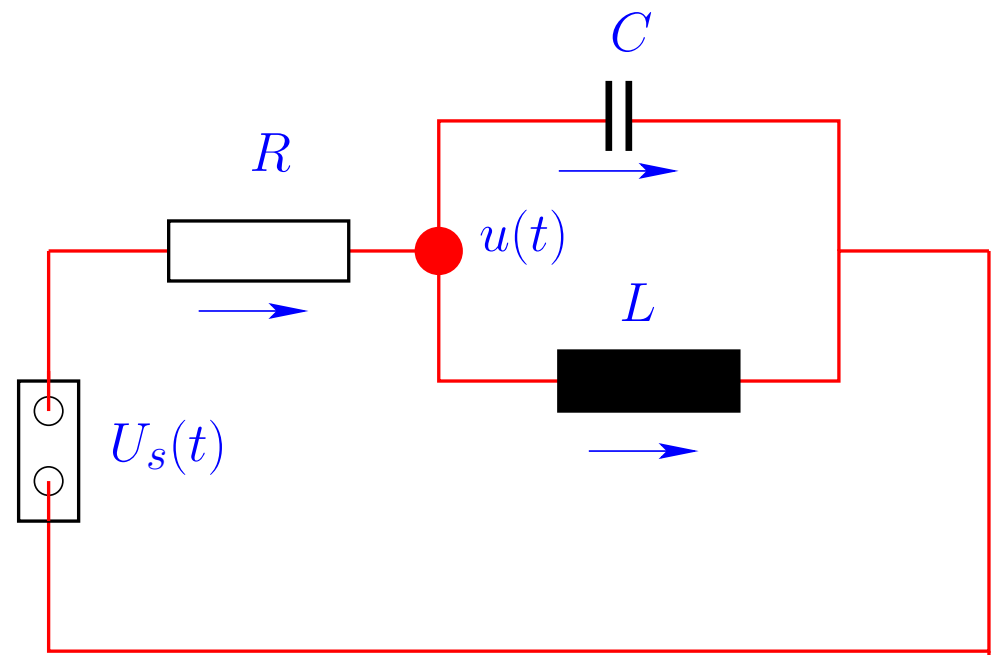


Fig. 255

Differentiate (12.1.9) w.r.t.  $t$  and plug in constitutive relations for circuit elements:

$$R^{-1} \frac{du_R}{dt}(t) - L^{-1}u_L(t) - C \frac{d^2u_C}{dt^2}(t) = 0.$$

We follow the policy of nodal analysis and express all voltages by potential differences between nodes of the circuit. For this simple circuit there is only one node with unknown potential, see Fig. 255. Its time-dependent potential will be denoted by  $u(t)$ .

$$R^{-1}(\dot{U}_s(t) - \dot{u}(t)) - L^{-1}u(t) - C \frac{d^2u}{dt^2}(t) = 0.$$



▶ autonomous **2nd-order** ordinary differential equation:

$$C\ddot{u} + R^{-1}\dot{u} + L^{-1}u = R^{-1}\dot{U}_s .$$



## 12.1.2 Theory [51, Sect. 11.1], [13, Sect. 11.3]

Abstract mathematical description

**Initial value problem (IVP) for first-order ordinary differential equation (ODE):**

( $\rightarrow$  [63, Sect. 5.6], [13, Sect. 11.1])

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \quad . \quad (12.1.13)$$

- $\mathbf{f} : I \times D \mapsto \mathbb{R}^d \hat{=} \text{right hand side (r.h.s.)}$  ( $d \in \mathbb{N}$ ), given in procedural form

function  $v = f(t, y)$ .

- $I \subset \mathbb{R} \hat{=} \text{(time)interval}$   $\leftrightarrow$  “time variable”  $t$
- $D \subset \mathbb{R}^d \hat{=} \text{state space/phase space}$   $\leftrightarrow$  “state variable”  $\mathbf{y}$  (*ger.:* Zustandsraum)
- $\Omega := I \times D \hat{=} \text{extended state space}$  (of tuples  $(t, \mathbf{y})$ )
- $t_0 \hat{=} \text{initial time}$ ,  $\mathbf{y}_0 \hat{=} \text{initial state}$   $\triangleright$  **initial conditions**

For  $d > 1$   $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$  can be viewed as a **system of ordinary differential equations:**

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad \iff \quad \begin{pmatrix} \dot{y}_1 \\ \vdots \\ \dot{y}_d \end{pmatrix} = \begin{pmatrix} f_1(t, y_1, \dots, y_d) \\ \vdots \\ f_d(t, y_1, \dots, y_d) \end{pmatrix} .$$

Recall: autonomous ODE  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$  ( $\mathbf{f}$  does not depend on time  $t$ )

For autonomous ODEs:

- $I = \mathbb{R}$  and r.h.s.  $\mathbf{y} \mapsto \mathbf{f}(\mathbf{y})$  can be regarded as stationary vector field (velocity field)
- if  $t \mapsto \mathbf{y}(t)$  is solution  $\Rightarrow$  for any  $\tau \in \mathbb{R}$   $t \mapsto \mathbf{y}(t + \tau)$  is solution, too.
- initial time irrelevant: canonical choice  $t_0 = 0$

Note: autonomous ODEs naturally arise when modelling time-invariant systems/phenomena. All examples from Sect. 12.1.1 above led to autonomous ODEs.

*Remark 12.1.14* (Conversion into autonomous ODE).

Idea: include time as an extra  $d + 1$ -st component of an extended state vector.

This solution component has to grow linearly  $\Leftrightarrow$  temporal derivative = 1

$$\mathbf{z}(t) := \begin{pmatrix} \mathbf{y}(t) \\ t \end{pmatrix} = \begin{pmatrix} \mathbf{z}' \\ z_{d+1} \end{pmatrix} : \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \Leftrightarrow \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}), \quad \mathbf{g}(\mathbf{z}) := \begin{pmatrix} \mathbf{f}(z_{d+1}, \mathbf{z}') \\ 1 \end{pmatrix} .$$

➤ focus on autonomous ODEs in the remainder of this chapter



Remark 12.1.15 (From higher order ODEs to first order systems). [13, Sect. 11.2]

Ordinary differential equation of order  $n \in \mathbb{N}$ :

$$\mathbf{y}^{(n)} = \mathbf{f}(t, \mathbf{y}, \dot{\mathbf{y}}, \dots, \mathbf{y}^{(n-1)}) \quad (12.1.16)$$

✎ Notation: superscript  $(n) \hat{=}$   $n$ -th temporal derivative  $t$

➤ Conversion into 1st-order ODE (system of size  $nd$ )

$$\mathbf{z}(t) := \begin{pmatrix} \mathbf{y}(t) \\ \mathbf{y}^{(1)}(t) \\ \vdots \\ \mathbf{y}^{(n-1)}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_n \end{pmatrix} \in \mathbb{R}^{dn}: \quad (12.1.16) \leftrightarrow \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}), \quad \mathbf{g}(\mathbf{z}) := \begin{pmatrix} \mathbf{z}_2 \\ \mathbf{z}_3 \\ \vdots \\ \mathbf{z}_n \\ \mathbf{f}(t, \mathbf{z}_1, \dots, \mathbf{z}_n) \end{pmatrix} \quad (12.1.17)$$

Note:  $n$  initial values  $\mathbf{y}(t_0), \dot{\mathbf{y}}(t_0), \dots, \mathbf{y}^{(n-1)}(t_0)$  required!



Basic assumption: right hand side  $\mathbf{f} : I \times D \mapsto \mathbb{R}^d$  locally Lipschitz continuous in  $\mathbf{y}$

**Definition 12.1.18** (Lipschitz continuous function). ( $\rightarrow$  [63, Def. 4.1.4])

$\mathbf{f} : \Omega \mapsto \mathbb{R}^d$  is *Lipschitz continuous* (in the second argument), if

$$\exists L > 0: \quad \|\mathbf{f}(t, \mathbf{w}) - \mathbf{f}(t, \mathbf{z})\| \leq L \|\mathbf{w} - \mathbf{z}\| \quad \forall (t, \mathbf{w}), (y, \mathbf{z}) \in \Omega .$$

**Definition 12.1.19** (Local Lipschitz continuity). ( $\rightarrow$  [63, Def. 4.1.5])

$\mathbf{f} : \Omega \mapsto \mathbb{R}^d$  is *locally Lipschitz continuous*, if

$$\forall (t, \mathbf{y}) \in \Omega: \exists \delta > 0, L > 0:$$

$$\|\mathbf{f}(\tau, \mathbf{z}) - \mathbf{f}(\tau, \mathbf{w})\| \leq L \|\mathbf{z} - \mathbf{w}\|$$

$$\forall \mathbf{z}, \mathbf{w} \in D: \|\mathbf{z} - \mathbf{y}\| < \delta, \|\mathbf{w} - \mathbf{y}\| < \delta, \forall \tau \in I: |t - \tau| < \delta .$$



Notation:  $D_{\mathbf{y}}\mathbf{f} \hat{=}$  derivative of  $\mathbf{f}$  w.r.t. state variable (= Jacobian  $\in \mathbb{R}^{d,d}$  !)

A simple criterion for local Lipschitz continuity:

**Lemma 12.1.20** (Criterion for local Lipschitz continuity).

If  $\mathbf{f}$  and  $D_{\mathbf{y}}\mathbf{f}$  are continuous on the extended state space  $\Omega$ , then  $\mathbf{f}$  is locally Lipschitz continuous ( $\rightarrow$  Def. 12.1.19).

**Theorem 12.1.21** (Theorem of Peano & Picard-Lindelöf). [2, Satz II(7.6)], [63, Satz 6.5.1], [13, Thm. 11.10], [35, Thm. 73.1]

If  $\mathbf{f} : \hat{\Omega} \mapsto \mathbb{R}^d$  is locally Lipschitz continuous ( $\rightarrow$  Def. 12.1.19) then for all initial conditions  $(t_0, \mathbf{y}_0) \in \hat{\Omega}$  the IVP (12.1.13) has a solution  $\mathbf{y} \in C^1(J(t_0, \mathbf{y}_0), \mathbb{R}^d)$  with **maximal** (temporal) domain of definition  $J(t_0, \mathbf{y}_0) \subset \mathbb{R}$ .

*Remark 12.1.22* (Domain of definition of solutions of IVPs).

Solutions of an IVP have an intrinsic maximal domain of definition

! domain of definition/domain of existence  $J(t_0, \mathbf{y}_0)$  usually depends on  $(t_0, \mathbf{y}_0)$  !

Terminology: if  $J(t_0, \mathbf{y}_0) = I \rightarrow$  solution  $\mathbf{y} : I \mapsto \mathbb{R}^d$  is **global**.



Notation: for autonomous ODE we always have  $t_0 = 0$ , therefore write  $J(\mathbf{y}_0) := J(0, \mathbf{y}_0)$ .

In light of Rem. 12.1.14 and Thm. 12.1.21: we consider only

$$\text{autonomous IVP: } \boxed{\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad , \quad \mathbf{y}(0) = \mathbf{y}_0} \quad , \quad (12.1.23)$$

with locally Lipschitz continuous ( $\rightarrow$  Def. 12.1.19) right hand side  $\mathbf{f}$ .

**Assumption 12.1.24** (Global solutions).

*All solutions of (12.1.23) are global:  $J(\mathbf{y}_0) = \mathbb{R}$  for all  $\mathbf{y}_0 \in D$ .*

Change of perspective: fix “time of interest”  $t \in \mathbb{R} \setminus \{0\}$

$\triangleright$  mapping  $\Phi^t : \begin{cases} D \mapsto D \\ \mathbf{y}_0 \mapsto \mathbf{y}(t) \end{cases}$  ,  $t \mapsto \mathbf{y}(t)$  solution of IVP (12.1.23) ,

is well-defined mapping of the state space into itself, by Thm. 12.1.21 and Ass. 12.1.24

Now, we may also let  $t$  vary, which spawns a *family* of mappings  $\{\Phi^t\}$  of the state space into itself. However, it can also be viewed as a mapping with two arguments, a time  $t$  and an initial state value  $\mathbf{y}_0$ !



**Definition 12.1.25** (Evolution operator).

Under Assumption 12.1.24 the mapping

$$\Phi : \begin{cases} \mathbb{R} \times D \mapsto D \\ (t, \mathbf{y}_0) \mapsto \Phi^t \mathbf{y}_0 := \mathbf{y}(t) \end{cases},$$

where  $t \mapsto \mathbf{y}(t) \in C^1(\mathbb{R}, \mathbb{R}^d)$  is the unique (global) solution of the IVP  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ ,  $\mathbf{y}(0) = \mathbf{y}_0$ , is the **evolution operator** for the autonomous ODE  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ .

Note:  $t \mapsto \Phi^t \mathbf{y}_0$  describes the solution of  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$  for  $\mathbf{y}(0) = \mathbf{y}_0$  (a trajectory)

*Remark 12.1.26* (Group property of autonomous evolutions).

Under Assumption 12.1.24 the evolution operator gives rise to a **group** of mappings  $D \mapsto D$ :

$$\Phi^s \circ \Phi^t = \Phi^{s+t}, \quad \Phi^{-t} \circ \Phi^t = Id \quad \forall t \in \mathbb{R}. \quad (12.1.27)$$

This is a consequence of the uniqueness theorem Thm. 12.1.21. It is also intuitive: following an evolution up to time  $t$  and then for some more time  $s$  leads us to the same final state as observing it for the whole time  $s + t$ .

Targeted: initial value problem (12.1.13)

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \quad . \quad (12.1.13)$$

Sought: *approximate* solution of (12.1.13) on  $[t_0, T]$  up to **final time**  $T \neq t_0$

However, the solution of an initial value problem is a *function*  $J(t_0, \mathbf{y}_0) \mapsto \mathbb{R}^d$  and requires a suitable approximate representation. We postpone this issue here and first study a geometric approach to numerical integration.

R. Hiptmair  
rev 38286,  
May 3,  
2011

**numerical integration** = approximate solution of initial value problems for ODEs

(Please distinguish from “numerical quadrature”, see Ch. 10.)

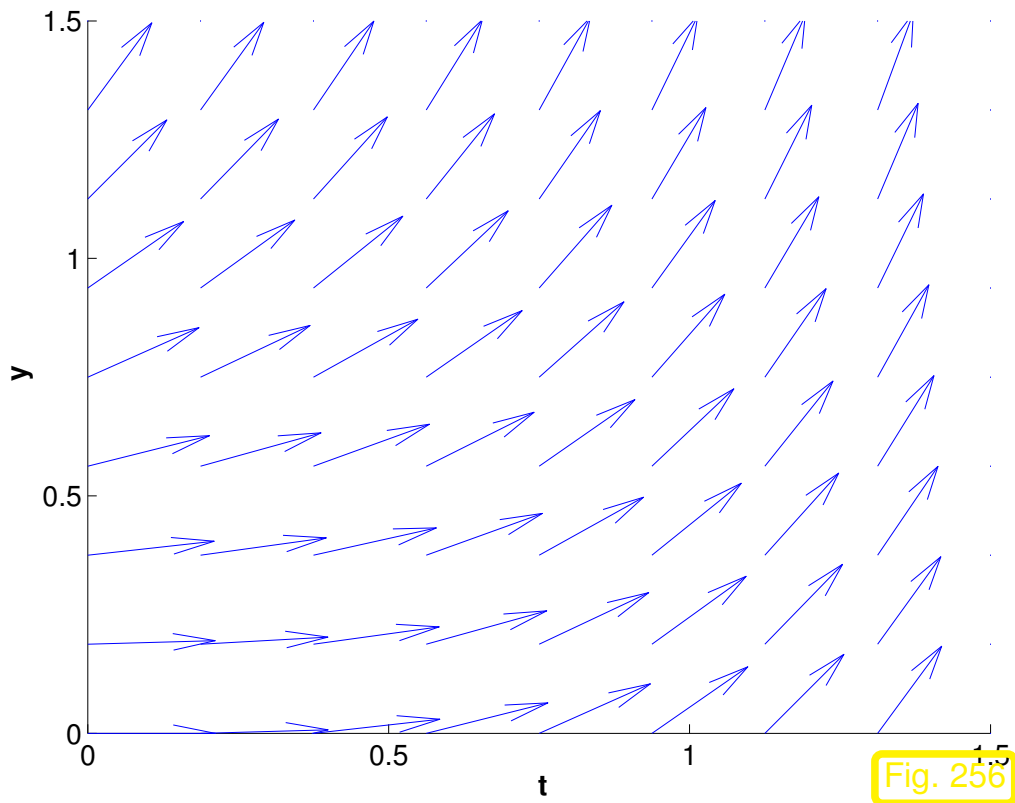
# 12.2.1 Explicit Euler method

Example 12.2.1 (Tangent field and solution curves).

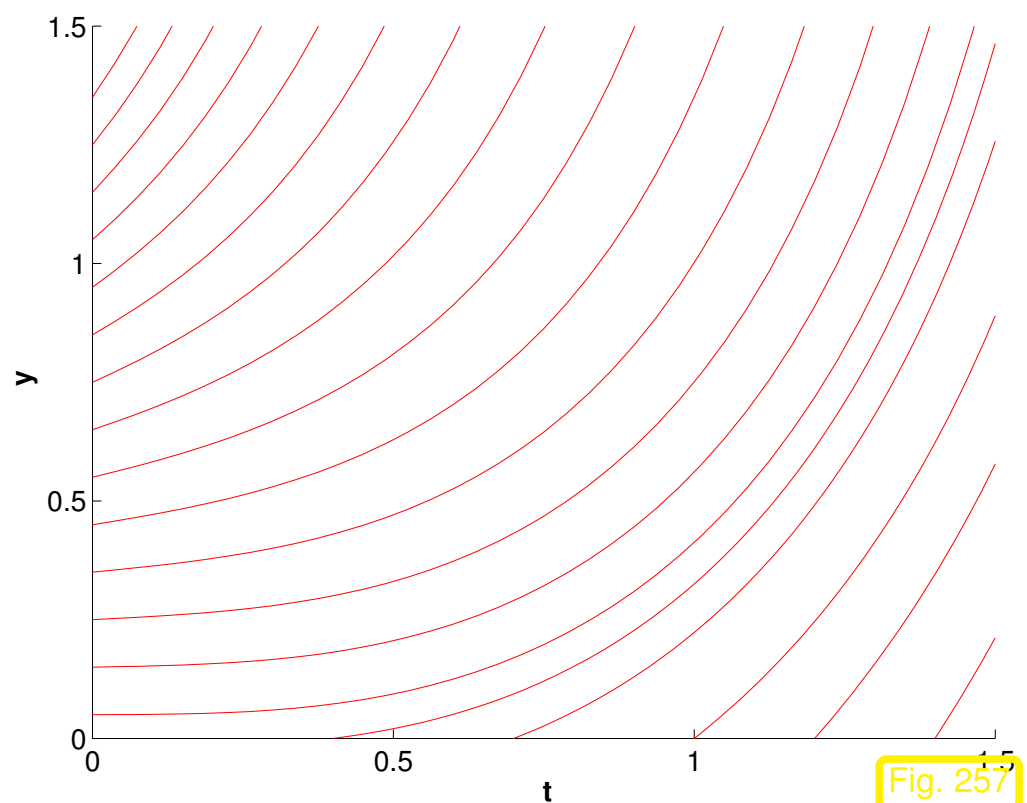
Riccati differential equation

$$\dot{y} = y^2 + t^2 \quad \blacktriangleright \quad d = 1, \quad I, D = \mathbb{R}^+ . \quad (12.2.2)$$

scalar ODE



tangent field



solution curves

solution curves run tangentially to the tangent field in each point of the extended state space.



- Idea:
- ❶ **timestepping**: successive approximation of evolution on *small* intervals  $[t_{k-1}, t_k]$ ,  $k = 1, \dots, N$ ,  $t_N := T$ ,
  - ❷ approximation of solution on  $[t_{k-1}, t_k]$  by **tangent** curve to current initial condition.

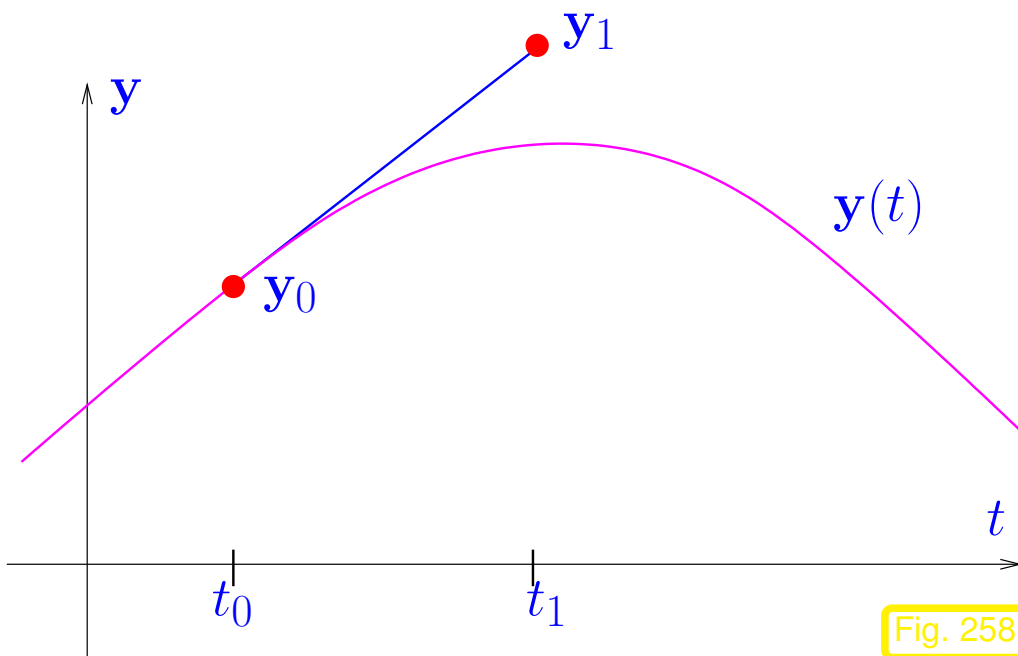
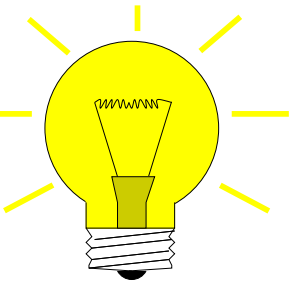


Fig. 258

**explicit Euler method** (Euler 1768)

◁ First step of explicit Euler method ( $d = 1$ ):

Slope of tangent =  $f(t_0, y_0)$

$y_1$  serves as initial value for next step!

see also [35, Ch. 74], [13, Alg. 11.4]

Example 12.2.3 (Visualization of explicit Euler method).

IVP for Riccati differential equation, see Ex. 12.2.1

$$\dot{y} = y^2 + t^2. \quad (12.2.2)$$

Here:  $y_0 = \frac{1}{2}, t_0 = 0, T = 1,$

—  $\hat{=}$  “Euler polygon” for uniform timestep  $h = 0.2$

$\rightarrow$   $\hat{=}$  tangent field of Riccati ODE

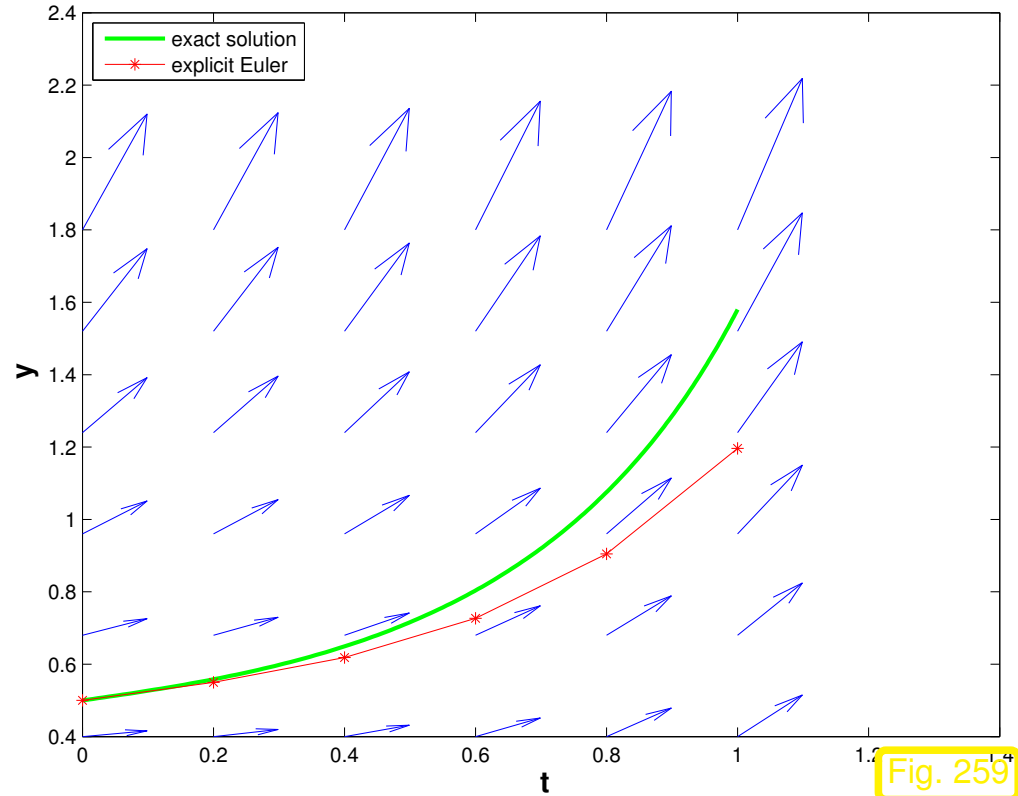


Fig. 259

Formula: explicit Euler method generates a *sequence*  $(\mathbf{y}_k)_{k=0}^N$  by the recursion

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_k, \mathbf{y}_k), \quad k = 0, \dots, N - 1, \quad (12.2.4)$$

with local (size of) **timestep** (stepsize)  $h_k := t_{k+1} - t_k$ .

*Remark 12.2.5* (Explicit Euler method as **difference scheme**).

(12.2.4) by approximating derivative  $\frac{d}{dt}$  by **forward difference quotient** on a (temporal) **mesh**  $\mathcal{M} := \{t_0, t_1, \dots, t_N\}$ :

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \longleftrightarrow \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f(t_k, \mathbf{y}_h(t_k)), \quad k = 0, \dots, N-1. \quad (12.2.6)$$

**Difference schemes** follow a simple policy for the *discretization* of differential equations: replace all derivatives by difference quotients connecting solution values on a set of discrete points (the mesh).



## 12.2.2 Implicit Euler method

Why forward difference quotient and not backward difference quotient? Let's try!

On (temporal) **mesh**  $\mathcal{M} := \{t_0, t_1, \dots, t_N\}$  we obtain

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \iff \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f(t_{k+1}, \mathbf{y}_h(t_{k+1})), \quad k = 0, \dots, N-1. \quad (12.2.7)$$

backward difference quotient

This leads to another simple timestepping scheme analogous to (12.2.4):

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}), \quad k = 0, \dots, N-1, \quad (12.2.8)$$

with local **timestep** (stepsize)  $h_k := t_{k+1} - t_k$ .

(12.2.8) = **implicit Euler method**

Note: (12.2.8) requires solving of a (possibly non-linear) system of equations to obtain  $\mathbf{y}_{k+1}$  !  
 (► Terminology “implicit”)

*Remark 12.2.9* (Feasibility of implicit Euler timestepping).

Issue: Is (12.2.8) well defined, that is, can we solve it for  $\mathbf{y}_{k+1}$  and is this solution unique?

Intuition: for small timesteps  $h > 0$  the right hand side of (12.2.8) is a “small perturbation of the identity”.

R. Hiptmair  
rev 38286,  
May 3,  
2011

Formal: consider autonomous ODE and assume continuously differentiable right hand side:  $\mathbf{f} \in C^1(D, \mathbb{R}^d)$ .

(12.2.8)  $\Leftrightarrow$   $h$ -dependent non-linear system of equations:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) \Leftrightarrow G(h, \mathbf{y}_{k+1}) = 0 \quad \text{with} \quad G(h, \mathbf{z}) := \mathbf{z} - h\mathbf{f}(\mathbf{z}) - \mathbf{y}_k .$$



Partial derivative:

$$\frac{dG}{dz}(0, \mathbf{z}) = \mathbf{I}$$

Implicit function theorem [63, Thm. 7.8.1]: for *sufficiently small*  $|h|$  the equation  $G(h, \mathbf{z}) = 0$  defines a continuous function  $\mathbf{z} = \mathbf{z}(h)$ .

➤ for *sufficiently small*  $h > 0$  the equation (12.2.8) can be uniquely solved for  $\mathbf{y}_{k+1}$ .

How to interpret the sequence  $(\mathbf{y}_k)_{k=0}^N$  from (12.2.4)?

By “geometric insight” we expect:

$$\mathbf{y}_k \approx \mathbf{y}(t_k)$$

(Throughout, we use the notation  $\mathbf{y}(t)$  for the exact solution of an IVP.)

If we are merely interested in the final state  $\mathbf{y}(T)$ , then the explicit Euler method will give us the answer  $\mathbf{y}_N$ .

If we are interested in an approximate solution  $\mathbf{y}_h(t) \approx \mathbf{y}(t)$  as a function  $[t_0, T] \mapsto \mathbb{R}^d$ , we have to do

**post-processing** = reconstruction of a function from  $\mathbf{y}_k, k = 0, \dots, N$

Technique: *interpolation*, see Ch. 3

Simplest option: piecewise linear interpolation ( $\rightarrow$  Sect. 3.6.1)  $\rightarrow$  **Euler polygon**, see Fig. 259.

### 12.2.3 Abstract single step methods

Recall Euler methods for autonomous ODE  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ :

$$\text{explicit Euler: } \mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k),$$

implicit Euler:  $\mathbf{y}_{k+1}$ :  $\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_{k+1})$ .

Both formulas provide a mapping

$$(\mathbf{y}_k, h_k) \mapsto \Psi(h, \mathbf{y}_k) := \mathbf{y}_{k+1}. \quad (12.2.10)$$

Recall the interpretation of the  $\mathbf{y}_k$  as approximations of  $\mathbf{y}(t_k)$ :

$$\Psi(h, \mathbf{y}) \approx \Phi^h \mathbf{y}, \quad (12.2.11)$$

where  $\Phi$  is the evolution operator ( $\rightarrow$  Def. 12.1.25) for  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ .

The Euler methods provide approximations for evolution operator for ODEs

This is what every single step method does: it tries to approximate the evolution operator  $\Phi$  for an ODE by a mapping of the type (12.2.10).

$\rightarrow$  mapping  $\Psi$  from (12.2.10) is called **discrete evolution**.

Vice versa: a mapping  $\Psi$  as in (12.2.10) defines a single step method.

**Definition 12.2.12** (Single step method (for autonomous ODE)).  $\rightarrow$  [51, Def. 11.2]

Given a discrete evolution  $\Psi : \Omega \subset \mathbb{R} \times D \mapsto \mathbb{R}^d$ , an initial state  $\mathbf{y}_0$ , and a temporal mesh  $\mathcal{M} := \{t_0 < t_1 < \dots < t_N = T\}$  the recursion

$$\mathbf{y}_{k+1} := \Psi(t_{k+1} - t_k, \mathbf{y}_k), \quad k = 0, \dots, N - 1, \quad (12.2.13)$$

defines a **single step method** (SSM, ger.: *Einschrittverfahren*) for the autonomous IVP  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ ,  $\mathbf{y}(0) = \mathbf{y}_0$ .

Procedural view of discrete evolutions:

$$\Psi^h \mathbf{y} \longleftrightarrow \text{function } y1 = \text{esvstep}(h, y0) . \\ ( \text{function } y1 = \text{esvstep}(@ (y) \text{ rhs}(y), h, y0) )$$

 Notation:  $\Psi^h \mathbf{y} := \Psi(h, \mathbf{y})$

Concept of single step method according to Def. 12.2.12 can be generalized to non-autonomous ODEs, which leads to recursions of the form:

$$\mathbf{y}_{k+1} := \Psi(t_k, t_{k+1}, \mathbf{y}_k), \quad k = 0, \dots, N - 1,$$

for discrete evolution defined on  $I \times I \times D$ .

*Remark 12.2.14* (Notation for single step methods).

Many authors specify a single step method by writing down the first step:

$$\mathbf{y}_1 = \text{expression in } \mathbf{y}_0 \text{ and } \mathbf{f}.$$

Also this course will sometimes adopt this practice.



# 12.3 Convergence of single step methods [13, Sect. 11.5] [51, Sect. 11.3]

Important issue: accuracy of approximation  $\mathbf{y}_k \approx \mathbf{y}(t_k)$  ?

*Remark 12.3.1* (Asymptotic perspective in convergence analysis).

As in the case of composite numerical quadrature, see Sect. 10.3: in general impossible to predict error  $\|\mathbf{y}_N - \mathbf{y}(T)\|$  for particular choice of timesteps.

Tractable: **asymptotic behavior** of error for timestep  $h := \max_k h_k \rightarrow 0$

▶ Will tell us asymptotic gain in accuracy for extra computational effort.  
(computational effort = no. of **f**-evaluations)



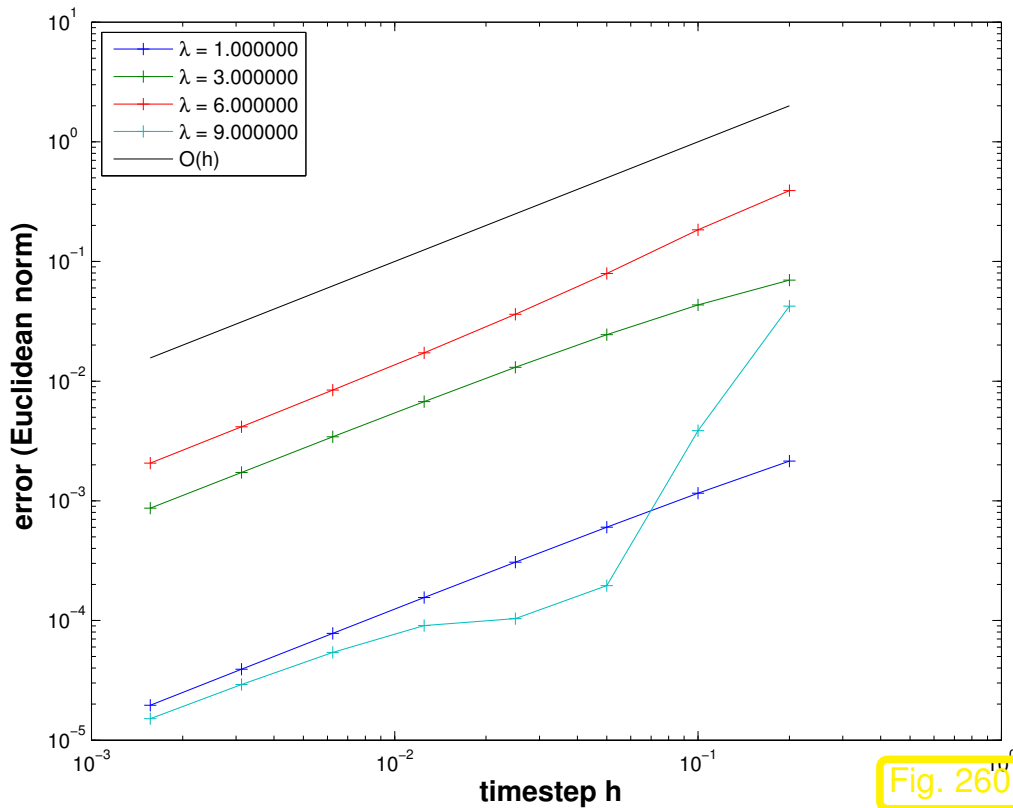
*Example 12.3.2* (Speed of convergence of Euler methods).

• IVP for logistic ODE, see Ex. 12.1.1

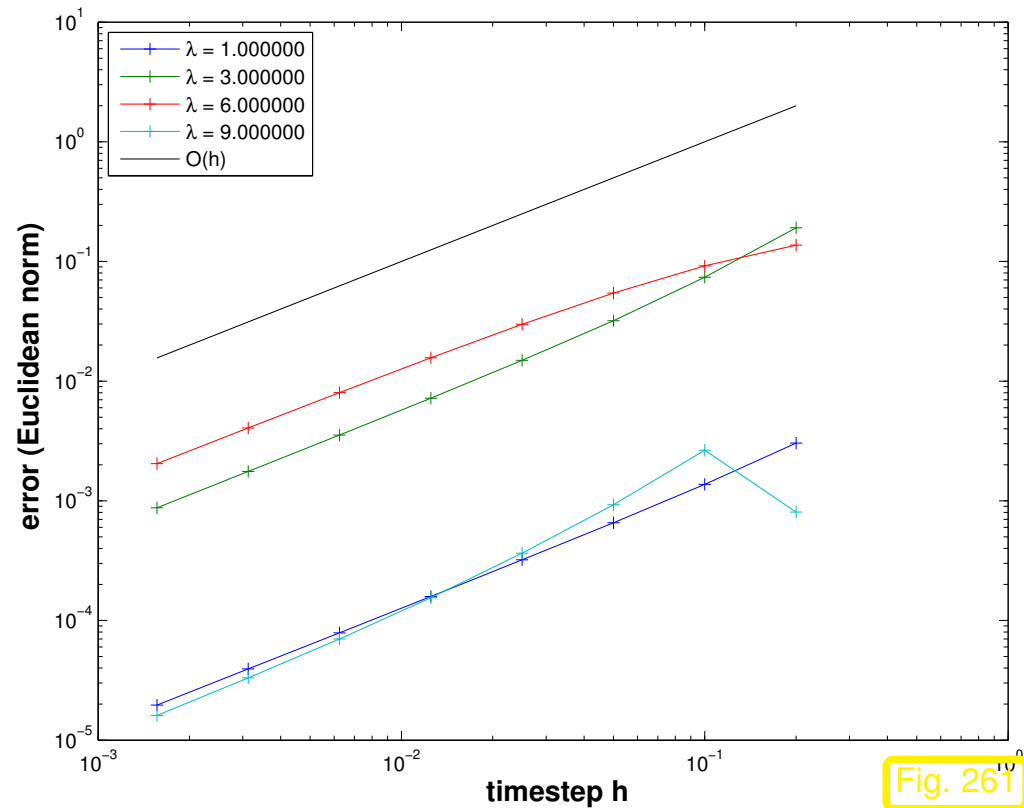
$$\dot{y} = \lambda y(1 - y) \quad , \quad y(0) = 0.01 \quad .$$

• Explicit and implicit Euler methods (12.2.4)/(12.2.8) with uniform timestep  $h = 1/N$ ,  $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$ .

• Monitored: Error at final time  $E(h) := |y(1) - y_N|$



explicit Euler method



implicit Euler method

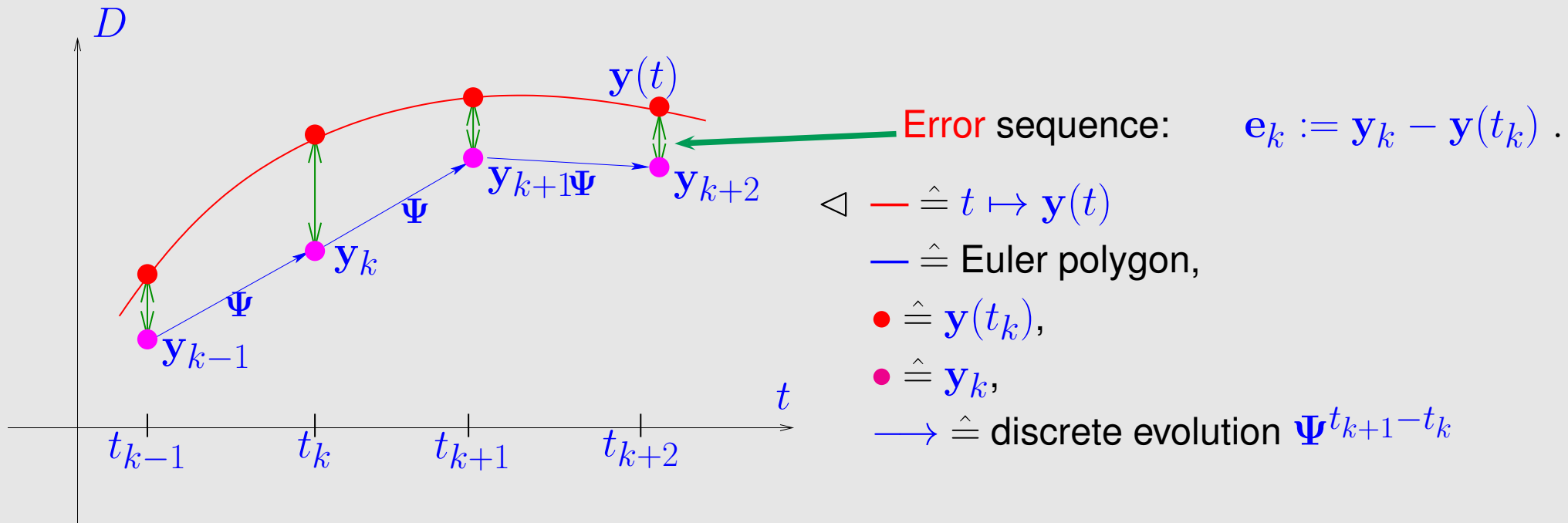
$O(h)$  algebraic convergence in both cases

**Convergence analysis** [35, Ch. 74] for explicit Euler method (12.2.4) for autonomous IVP (12.1.13) with sufficiently smooth and (*globally*) Lipschitz continuous  $\mathbf{f}$ , that is,

$$\exists L > 0: \quad \|\mathbf{f}(t, \mathbf{y}) - \mathbf{f}(t, \mathbf{z})\| \leq L \|\mathbf{y} - \mathbf{z}\| \quad \forall \mathbf{y}, \mathbf{z} \in D. \quad (12.3.3)$$

Recall: recursion for explicit Euler method

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k), \quad k = 1, \dots, N - 1. \quad (12.2.4)$$



① Abstract splitting of error:



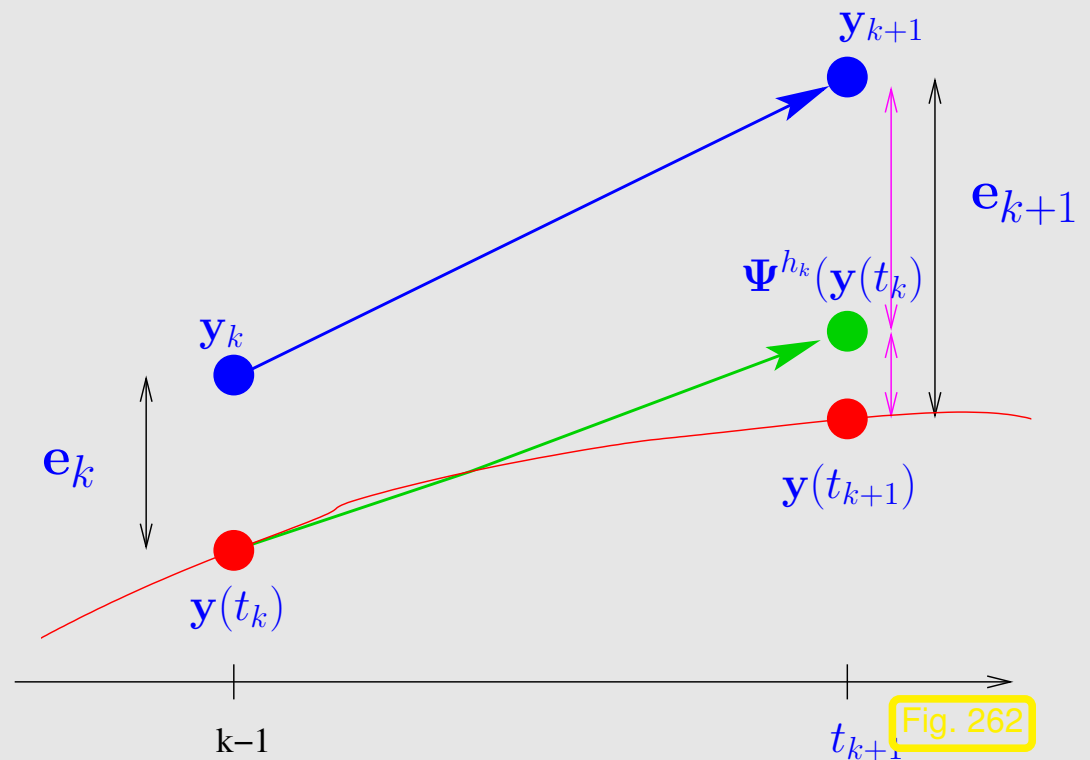
Here and in what follows we rely on the abstract concepts of the evolution operator  $\Phi$  associated with the ODE  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$  ( $\rightarrow$  Def. 12.1.25) and discrete evolution operator  $\Psi$  defining the explicit Euler single step method, see Def. 12.2.12:

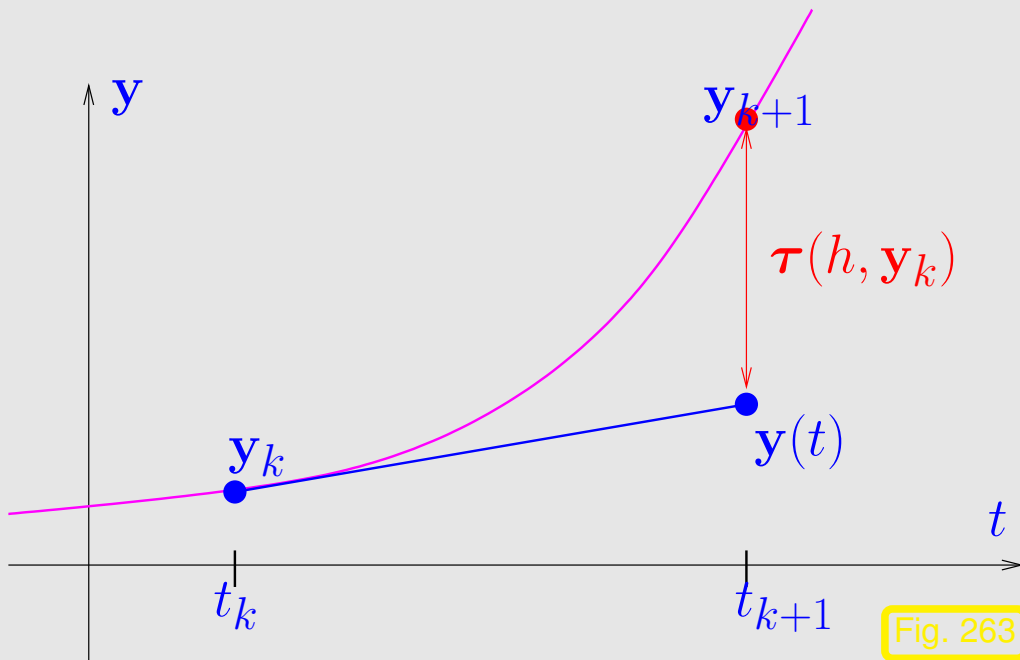
$$(12.2.4) \Rightarrow \Psi^h \mathbf{y} = \mathbf{y} + h\mathbf{f}(\mathbf{y}) . \quad (12.3.4)$$

We argue that in this context the abstraction pays off, because it helps elucidate a general technique for the convergence analysis of single step methods.

Fundamental error splitting

$$\begin{aligned} \mathbf{e}_{k+1} &= \Psi^{h_k} \mathbf{y}_k - \Phi^{h_k} \mathbf{y}(t_k) \\ &= \underbrace{\Psi^{h_k} \mathbf{y}_k - \Psi^{h_k} \mathbf{y}(t_k)}_{\text{propagated error}} \\ &\quad + \underbrace{\Psi^{h_k} \mathbf{y}(t_k) - \Phi^{h_k} \mathbf{y}(t_k)}_{\text{one-step error}} . \end{aligned} \quad (12.3.5)$$





one-step error:

$$\tau(h, \mathbf{y}) := \Psi^h \mathbf{y} - \Phi^h \mathbf{y} . \quad (12.3.6)$$

◁ geometric visualisation of one-step error for explicit Euler method (12.2.4), *cf.* Fig. 258.

📝 notation:  $t \mapsto \mathbf{y}(t) \hat{=}$  (unique) solution of IVP, *cf.* Thm. 12.1.21.

## ② Estimate for one-step error:

Geometric considerations: distance of a smooth curve and its tangent shrinks as the square of the distance to the intersection point (curve locally looks like a parabola in the  $\xi - \eta$  coordinate system, see Fig. 265).

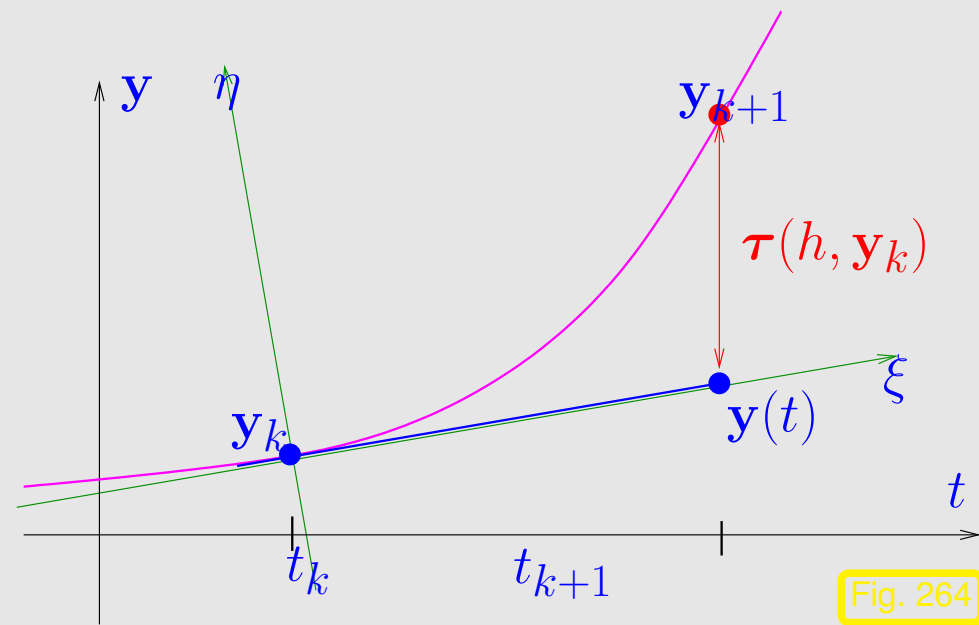


Fig. 264

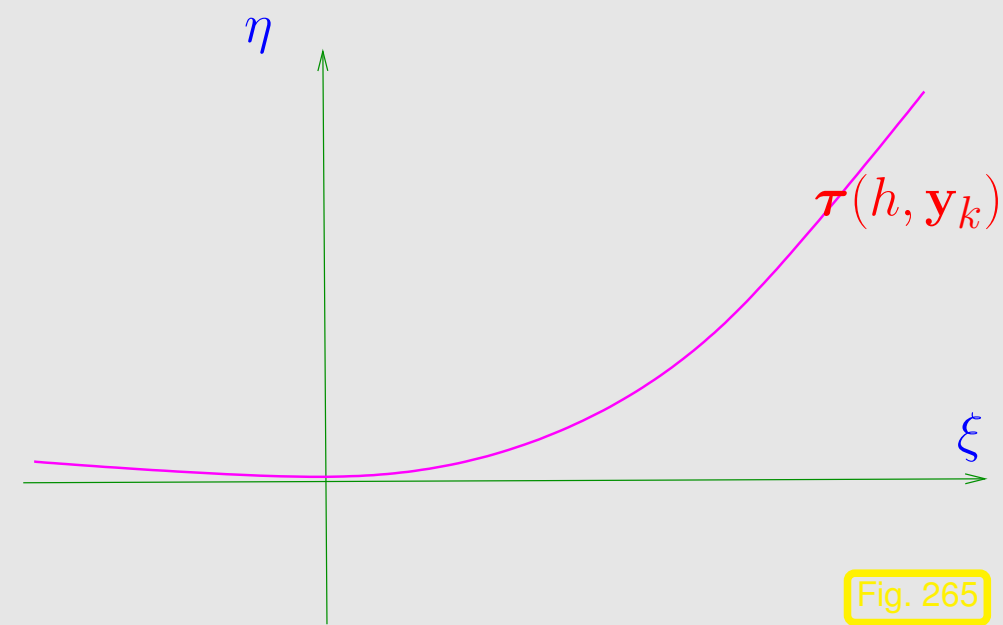


Fig. 265

Analytic considerations: recall Taylor's formula for function  $\mathbf{y} \in C^{K+1}$

$$\mathbf{y}(t+h) - \mathbf{y}(t) = \sum_{j=0}^K \mathbf{y}^{(j)}(t) \frac{h^j}{j!} + \underbrace{\int_t^{t+h} f^{(K+1)}(\tau) \frac{(t+h-\tau)^K}{K!} d\tau}_{= \frac{f^{(K+1)}(\xi)}{K!} h^{K+1}}, \quad (12.3.7)$$

for some  $\xi \in [t, t+h]$

$\Rightarrow$  if  $\mathbf{y} \in C^2([0, T])$ , then

$\blacktriangleright \quad \mathbf{y}(t_{k+1}) - \mathbf{y}(t_k) = \dot{\mathbf{y}}(t_k)h_k + \frac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \quad \text{for some } t_k \leq \xi_k \leq t_{k+1}$

$$= \mathbf{f}(\mathbf{y}(t_k))h_k + \frac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 ,$$

since  $t \mapsto \mathbf{y}(t)$  solves the ODE, which implies  $\dot{\mathbf{y}}(t_k) = \mathbf{f}(\mathbf{y}(t_k))$ . This leads to an expression for the one-step error from (12.3.6)

$$\begin{aligned} \tau(h_k, \mathbf{y}(t_k)) &= \Psi^{h_k} \mathbf{y}(t_k) - \mathbf{y}(t_k + h_k) \\ &\stackrel{(12.3.4)}{=} \mathbf{y}(t_k) + h_k \mathbf{f}(\mathbf{y}(t_k)) - \mathbf{y}(t_k) - \mathbf{f}(\mathbf{y}(t_k))h_k + \frac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \\ &= \frac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 . \end{aligned} \tag{12.3.8}$$

Stoppily speaking, we observe  $\tau(h_k, \mathbf{y}(t_k)) = O(h_k^2)$  uniformly for  $h_k \rightarrow 0$ .

③ **Estimate for the propagated error** from (12.3.5)

$$\begin{aligned} \left\| \Psi^{h_k} \mathbf{y}_k - \Psi^{h_k} \mathbf{y}(t_k) \right\| &= \left\| \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k) - \mathbf{y}(t_k) - h_k \mathbf{f}(\mathbf{y}(t_k)) \right\| \\ &\stackrel{(12.3.3)}{\leq} (1 + Lh_k) \left\| \mathbf{y}_k - \mathbf{y}(t_k) \right\| . \end{aligned} \tag{12.3.9}$$

③ **Recursion** for error norms  $\epsilon_k := \|\mathbf{e}_k\|$  by  $\triangle$ -inequality:

$$\epsilon_{k+1} \leq (1 + h_k L) \epsilon_k + \rho_k , \quad \rho_k := \frac{1}{2} h_k^2 \max_{t_k \leq \tau \leq t_{k+1}} \|\ddot{\mathbf{y}}(\tau)\| . \tag{12.3.10}$$

Taking into account  $\epsilon_0 = 0$  this leads to

$$\epsilon_k \leq \sum_{l=1}^k \prod_{j=1}^{l-1} (1 + Lh_j) \rho_l, \quad k = 1, \dots, N. \quad (12.3.11)$$

Use the elementary estimate  $(1 + Lh_j) \leq \exp(Lh_j)$  (by convexity of exponential function):

$$(12.3.11) \Rightarrow \epsilon_k \leq \sum_{l=1}^k \prod_{j=1}^{l-1} \exp(Lh_j) \cdot \rho_l = \sum_{l=1}^k \exp\left(L \sum_{j=1}^{l-1} h_j\right) \rho_l.$$

Note:  $\sum_{j=1}^{l-1} h_j \leq T$  for final time  $T$

$$\begin{aligned} \blacktriangleright \quad \epsilon_k &\leq \exp(LT) \sum_{l=1}^k \rho_l \leq \exp(LT) \max_k \frac{\rho_k}{h_k} \sum_{l=1}^k h_l \\ &\leq T \exp(LT) \max_{l=1, \dots, k} h_l \cdot \max_{t_0 \leq \tau \leq t_k} \|\ddot{\mathbf{y}}(\tau)\|. \end{aligned}$$

$$\blacktriangleright \quad \|\mathbf{y}_k - \mathbf{y}(t_k)\| \leq T \exp(LT) \max_{l=1, \dots, k} h_l \cdot \max_{t_0 \leq \tau \leq t_k} \|\ddot{\mathbf{y}}(\tau)\|.$$

Total error arises from accumulation of one-step errors!

- error bound =  $O(h)$ ,  $h := \max_l h_l$  (► 1st-order **algebraic convergence**)
- Error bound grows *exponentially* with the length  $T$  of the integration interval.

Most commonly used single step methods display algebraic convergence of integer order with respect to the meshwidth  $h := \max_k h_k$ . This offers a criterion for gauging their quality.

The sequence  $(\mathbf{y}_k)_k$  generated by a

single step method ( $\rightarrow$  Def. 12.2.12) of **order** (of consistency)  $p \in \mathbb{N}$

for  $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$  on a mesh  $\mathcal{M} := \{t_0 < t_1 < \dots < t_N = T\}$  satisfies

$$\max_k \|\mathbf{y}_k - \mathbf{y}(t_k)\| \leq Ch^p \quad \text{for } h := \max_{k=1, \dots, N} |t_k - t_{k-1}| \rightarrow 0, \quad (12.3.12)$$

with  $C > 0$  independent of  $\mathcal{M}$ , provided that  $\mathbf{f}$  is *sufficiently smooth*, see [13, Thm. 11.25].

# 12.4 Runge-Kutta methods [13, Sect. 11.6], [35, Ch. 76], [51, Sect.

So far we only know first order methods, the explicit and implicit Euler method (12.2.4) and (12.2.8), respectively.

Do we need anything else?

*Remark* 12.4.1 (Rationale for high-order single step methods). *cf.* [13, Sect. 11.5.3]

Rem. 12.3.1 ➤ Due to unknown constants “ $C > 0$ ” it is too ambitious to ask how many timesteps are needed so that  $\|\mathbf{y}(T) - \mathbf{y}_N\|$  stays below a prescribed bound, *cf.* discussion in Sect. 10.3 in the context of numerical quadrature.

Question answered by *asymptotic estimates* like (12.3.12)

# What extra computational effort buys a prescribed *reduction of the error*?

(cf. considerations in Sect. 4.3.3)

Usual concept for single step methods ( $\rightarrow$  Def. 12.2.12)

Computational effort  $\sim$  total number of **f**-evaluations,

$\sim$  number of timesteps, if evaluation of discrete evolution  $\Psi^h$  ( $\rightarrow$  Def. 12.2.12) requires fixed number of **f**-evaluations,


$\sim h^{-1}$ , in the case of uniform timestep size  $h > 0$ .

$$\text{Goal: } \frac{\text{err}(h_{\text{new}})}{\text{err}(h_{\text{old}})} \stackrel{!}{=} \rho \quad \text{for } 0 < \rho < 1.$$

$$(12.3.12) \quad \Rightarrow \quad \frac{h_{\text{new}}^p}{h_{\text{old}}^p} \stackrel{!}{=} \rho \quad \Leftrightarrow \quad \boxed{h_{\text{new}} = \rho^{1/p} h_{\text{old}}}.$$

For single step method of **order**  $p \in \mathbb{N}$

increase effort by factor  $\rho^{-1/p}$   $\blacktriangleright$  reduce error by factor  $\rho$

 the larger the order  $p$ , the less effort for a prescribed reduction of the error!





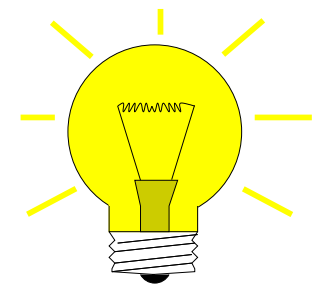
Now we will build a class of methods that achieve orders  $> 1$ . The starting point is a simple *integral equation* satisfied by solutions of initial value problems:

$$\text{IVP: } \begin{array}{l} \dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}(t)) , \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{array} \Rightarrow \mathbf{y}(t_1) = \mathbf{y}_0 + \int_{t_0}^{t_1} \mathbf{f}(\tau, \mathbf{y}(\tau)) \, d\tau$$

Idea: approximate integral by means of  $s$ -point quadrature formula ( $\rightarrow$  Sect. 10.1, defined on reference interval  $[0, 1]$ ) with nodes  $c_1, \dots, c_s$ , weights  $b_1, \dots, b_s$ .

$$\mathbf{y}(t_1) \approx \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{f}(t_0 + c_i h, \mathbf{y}(t_0 + c_i h)) , \quad h := t_1 - t_0 . \quad (12.4.2)$$

Obtain these values by **bootstrapping**



bootstrapping = use the same idea in a simpler version to get  $\mathbf{y}(t_0 + c_i h)$ , noting that these values can be replaced by other approximations obtained by methods already constructed (this approach will be elucidated in the next example).

What error can we afford in the approximation of  $\mathbf{y}(t_0 + c_i h)$  (under the assumption that  $\mathbf{f}$  is Lipschitz continuous)?

Goal: one-step error  $\mathbf{y}(t_1) - \mathbf{y}_1 = O(h^{p+1})$

This goal can already be achieved, if only

$\mathbf{y}(t_0 + c_i h)$  is approximated up to an error  $O(h^p)$ ,

because in (12.4.2) a factor of size  $h$  multiplies  $\mathbf{f}(t_0 + c_i, \mathbf{y}(t_0 + c_i h))$ .

This is accomplished by a less accurate discrete evolution than the one we are bidding for. Thus, we can construct discrete evolutions of higher and higher order, successively.

*Example 12.4.3* (Construction of simple Runge-Kutta methods).

Quadrature formula = trapezoidal rule (10.2.7):

$$Q(f) = \frac{1}{2}(f(0) + f(1)) \leftrightarrow s = 2: \quad c_1 = 0, c_2 = 1, \quad b_1 = b_2 = \frac{1}{2}, \quad (12.4.4)$$

and  $\mathbf{y}(T)$  approximated by explicit Euler step (12.2.4)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0), \quad \mathbf{k}_2 = \mathbf{f}(t_0 + h, \mathbf{y}_0 + h\mathbf{k}_1), \quad \mathbf{y}_1 = \mathbf{y}_0 + \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2). \quad (12.4.5)$$

(12.4.5) = **explicit trapezoidal rule** (for numerical integration of ODEs).

Quadrature formula  $\rightarrow$  simplest Gauss quadrature formula = midpoint rule ( $\rightarrow$  Ex. 10.2.5) &  $\mathbf{y}(\frac{1}{2}(t_1 - t_0))$  approximated by explicit Euler step (12.2.4)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0), \quad \mathbf{k}_2 = \mathbf{f}(t_0 + \frac{h}{2}, \mathbf{y}_0 + \frac{h}{2}\mathbf{k}_1), \quad \mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{k}_2. \quad (12.4.6)$$

(12.4.6) = **explicit midpoint rule** (for numerical integration of ODEs) [13, Alg. 11.18].



*Example 12.4.7* (Convergence of simple Runge-Kutta methods).

- IVP:  $\dot{y} = 10y(1 - y)$  (logistic ODE (12.1.2)),  $y(0) = 0.01$ ,  $T = 1$ ,
- Explicit single step methods, uniform timestep  $h$ .

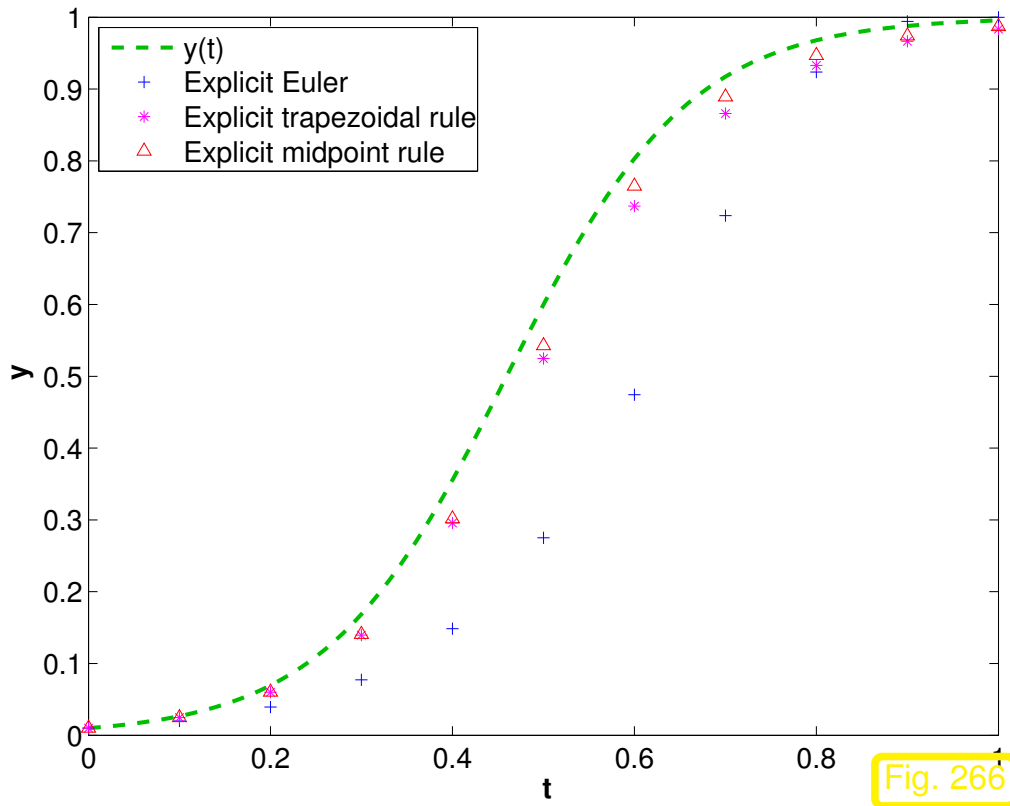


Fig. 266

$y_h(j/10), j = 1, \dots, 10$  for explicit RK-methods

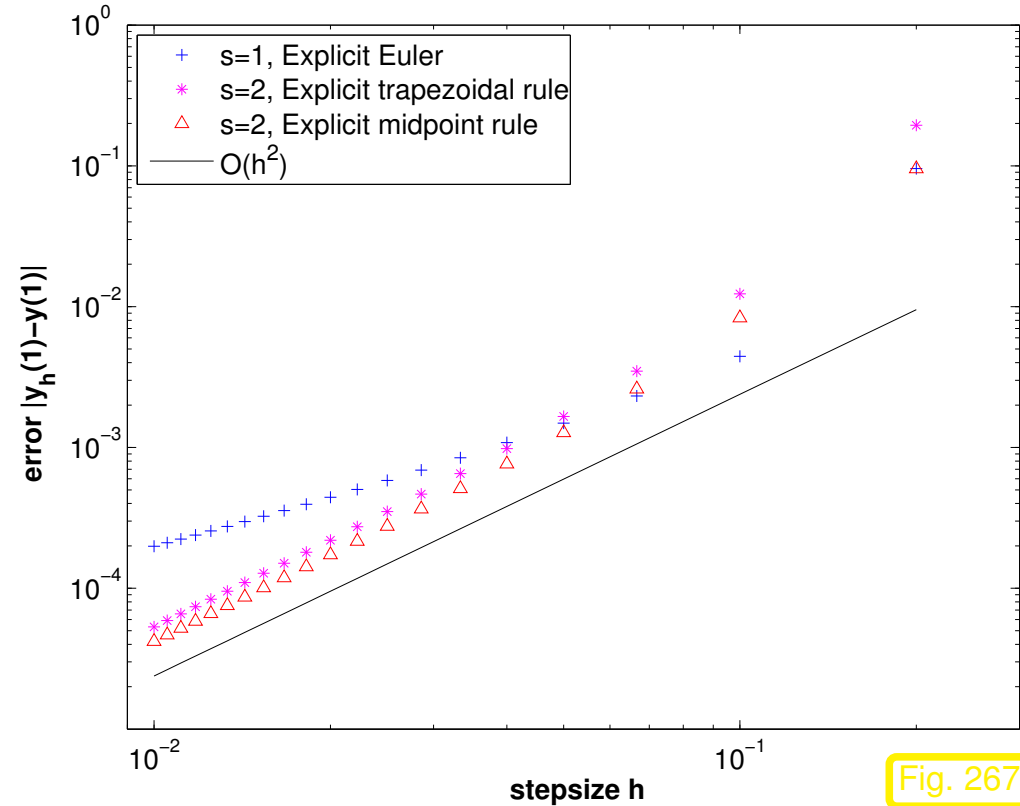


Fig. 267

Errors at final time  $y_h(1) - y(1)$

Observation: obvious algebraic convergence with integer rates/orders

explicit trapezoidal rule (12.4.5) order 2  
 explicit midpoint rule (12.4.6) order 2

The formulas that we have obtained follow a general pattern:

**Definition 12.4.8** (Explicit Runge-Kutta method).

For  $b_i, a_{ij} \in \mathbb{R}$ ,  $c_i := \sum_{j=1}^{i-1} a_{ij}$ ,  $i, j = 1, \dots, s$ ,  $s \in \mathbb{N}$ , an *s-stage explicit Runge-Kutta single step method* (RK-SSM) for the IVP (12.1.13) is defined by

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h \sum_{j=1}^{i-1} a_{ij} \mathbf{k}_j), \quad i = 1, \dots, s, \quad \mathbf{y}_1 := \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i.$$

The  $\mathbf{k}_i \in \mathbb{R}^d$  are called *increments*.

Recall Rem. 12.2.14 to understand how the discrete evolution for an explicit Runge-Kutta method is specified in this definition by giving the formulas for the first step. This is a convention widely adopted in the literature about numerical methods for ODEs. Of course, the increments  $\mathbf{k}_i$  have to be computed anew in each timestep.

The implementation of an  $s$ -stage explicit Runge-Kutta single step method according to Def. 12.4.8 is straightforward: The increments  $\mathbf{k}_i \in \mathbb{R}^d$  are computed successively, starting from  $\mathbf{k}_1 = \mathbf{f}(t_0 + c_1 h, \mathbf{y}_0)$ .

► Only  $s$   $\mathbf{f}$ -evaluations and AXPY operations ( $\rightarrow$  Sect. 1.4) are required.

Shorthand notation for (explicit) Runge-Kutta methods [13, (11.75)]

Butcher scheme

(Note:  $\mathcal{A}$  is strictly lower triangular  $s \times s$ -matrix)

►

$$\begin{array}{c|c} \mathbf{c} & \mathcal{A} \\ \hline & \mathbf{b}^T \end{array}$$

:=

$$\begin{array}{c|ccc} c_1 & 0 & \dots & 0 \\ c_2 & a_{21} & \ddots & \vdots \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \dots & a_{s,s-1} & 0 \\ \hline & b_1 & \dots & b_s \end{array} \quad (12.4.9)$$

Note that in Def. 12.4.8 the coefficients  $b_i$  can be regarded as weights of a quadrature formula on  $[0, 1]$ : apply explicit Runge-Kutta single step method to “ODE”  $\dot{y} = f(t)$ .

►

Necessarily

$$\sum_{i=1}^s b_i = 1$$

*Example 12.4.10* (Butcher scheme for some explicit RK-SSM). [13, Sect. 11.6.1]

- Explicit Euler method (12.2.4):

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$



order = 1

- explicit trapezoidal rule (12.4.5):

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$



order = 2

- explicit midpoint rule (12.4.6):

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$$



order = 2

- Classical 4th-order RK-SSM:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array}$$



order = 4

- Kutta's 3/8-rule:

$$\begin{array}{c|cccc}
 0 & 0 & 0 & 0 & 0 \\
 \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
 \frac{1}{3} & -\frac{1}{3} & 1 & 0 & 0 \\
 1 & 1 & -1 & 1 & 0 \\
 \hline
 & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8}
 \end{array}
 \quad \triangleright \quad \text{order} = 4$$



*Remark 12.4.11* (“Butcher barriers” for explicit RK-SSM).

order $p$	1	2	3	4	5	6	7	8	$\geq 9$
minimal no. $s$ of stages	1	2	3	4	6	7	9	11	$\geq p + 3$

No general formula available so far

Known: order  $p <$  number  $s$  of stages of RK-SSM



**Remark 12.4.12** (Explicit ODE integrator in MATLAB).

Syntax:

$$[t, y] = \text{ode45}(\text{odefun}, \text{tspan}, y_0);$$

odefun : **Handle** to a function of type  $@(t, y) \leftrightarrow$  r.h.s.  $\mathbf{f}(t, \mathbf{y})$   
 tspan : vector  $(t_0, T)^T$ , initial and final time for numerical integration  
 y0 : (vector) passing initial state  $\mathbf{y}_0 \in \mathbb{R}^d$

Return values:

t : temporal mesh  $\{t_0 < t_1 < t_2 < \dots < t_{N-1} = t_N = T\}$   
 y : sequence  $(\mathbf{y}_k)_{k=0}^N$  (column vectors)

Code 12.4.13: code excerpts for MATLAB integrator ode45

```

1 function varargout = ode45(ode,tspan,y0,options,varargin)
2 % Processing of input parameters omitted
3 % :
4 % Initialize method parameters, c.f. Butcher scheme (12.4.9)
5 pow = 1/5;
6 A = [1/5, 3/10, 4/5, 8/9, 1, 1];
7 B = [
8     1/5          3/40      44/45      19372/6561      9017/3168      35/384
9     0            9/40      -56/15     -25360/2187     -355/33        0

```

```
0      0      0      32/9      64448/6561      46732/5247      500/1113
1      0      0      0      -212/729      49/176      125/192
2      0      0      0      0      -5103/18656      -2187/6784
3      0      0      0      0      0      11/84
4      0      0      0      0      0      0
5      ];
6  E = [71/57600; 0; -71/16695; 71/1920; -17253/339200; 22/525; -1/40];
7  % : (choice of stepsize and main loop omitted)
8  % ADVANCING ONE STEP.
9  hA = h * A;
10 hB = h * B;
11 f(:,2) = feval(odeFcn,t+hA(1),y+f*hB(:,1),odeArgs{:});
12 f(:,3) = feval(odeFcn,t+hA(2),y+f*hB(:,2),odeArgs{:});
13 f(:,4) = feval(odeFcn,t+hA(3),y+f*hB(:,3),odeArgs{:});
14 f(:,5) = feval(odeFcn,t+hA(4),y+f*hB(:,4),odeArgs{:});
15 f(:,6) = feval(odeFcn,t+hA(5),y+f*hB(:,5),odeArgs{:});
16
17 tnew = t + hA(6);
18 if done, tnew = tfinal; end % Hit end point exactly.
19 h = tnew - t; % Purify h.
20 ynew = y + f*hB(:,6);
21 % : (stepsize control, see Sect. 12.5 dropped
```



MATLAB-integrator: `ode45()`:MATLAB-CODE: usage of `ode45`

```
fn = @(t,y) 5*y*(1-y);  
[t,y] = ode45(fn, [0 1.5], y0);  
plot(t,y,'r-');
```

- Handle passing r.h.s.
- initial and final time
- initial state  $y_0$

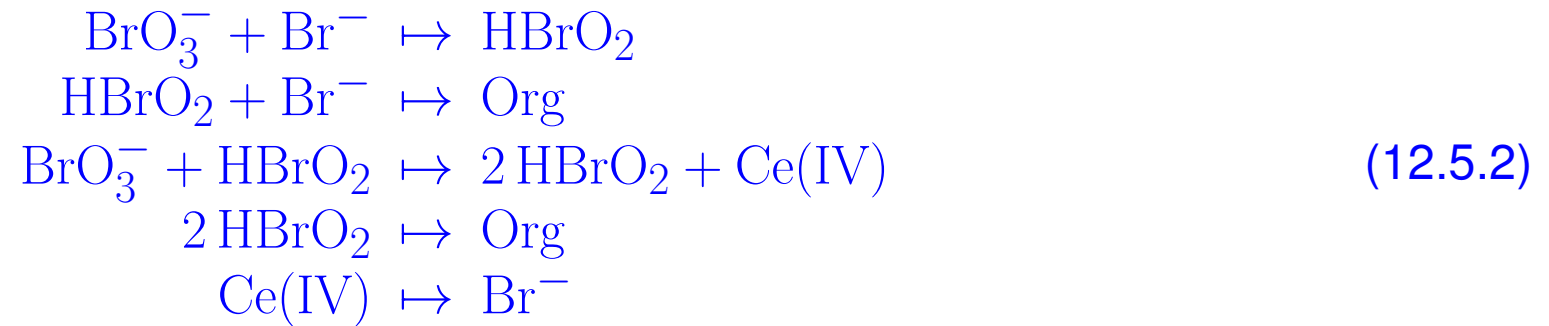


## 12.5 Stepsize control [13, Sect. 11.7], [51, Sect. 11.8.2]

*Example 12.5.1* (Oregonator reaction).

See [35, Ch. 62] for the ODE-based modelling of kinetics of chemical reactions.


Special case of oscillating Zhabotinski-Belousov reaction [25]:



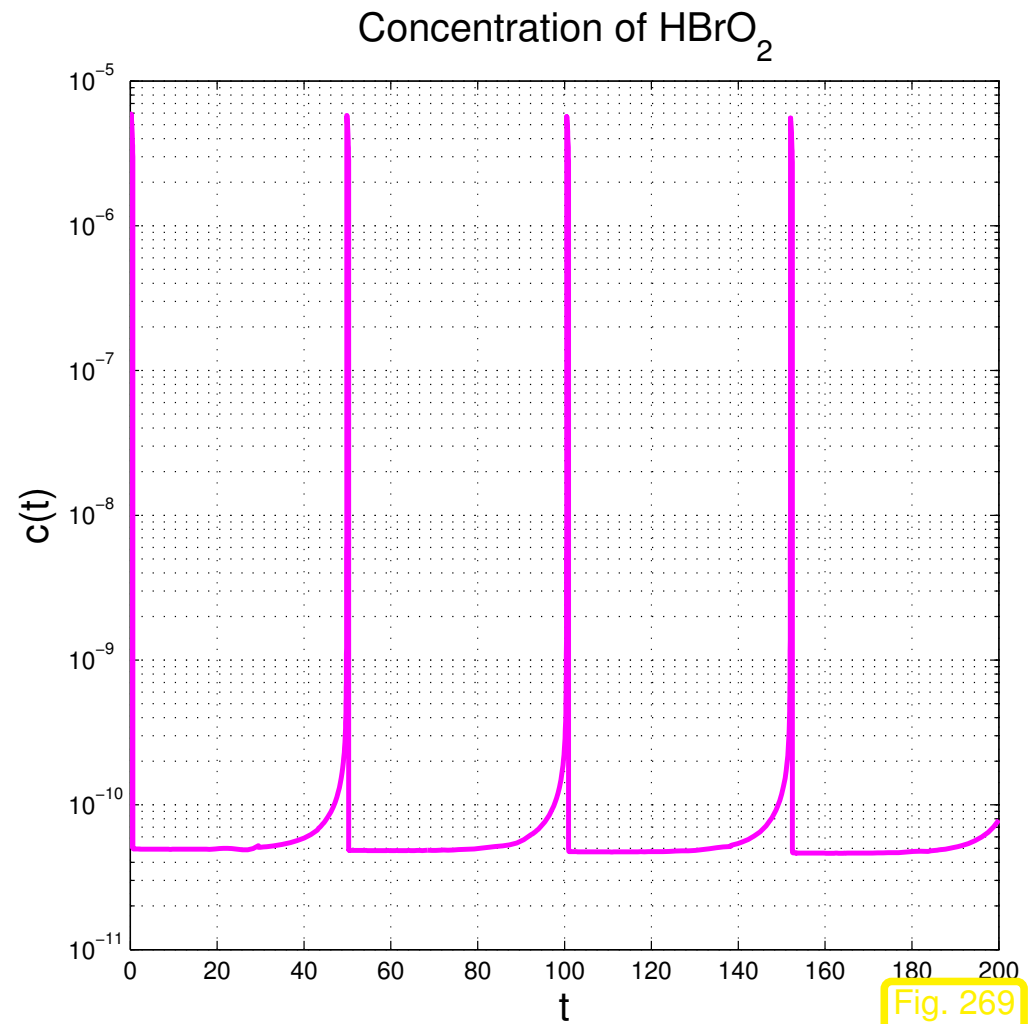
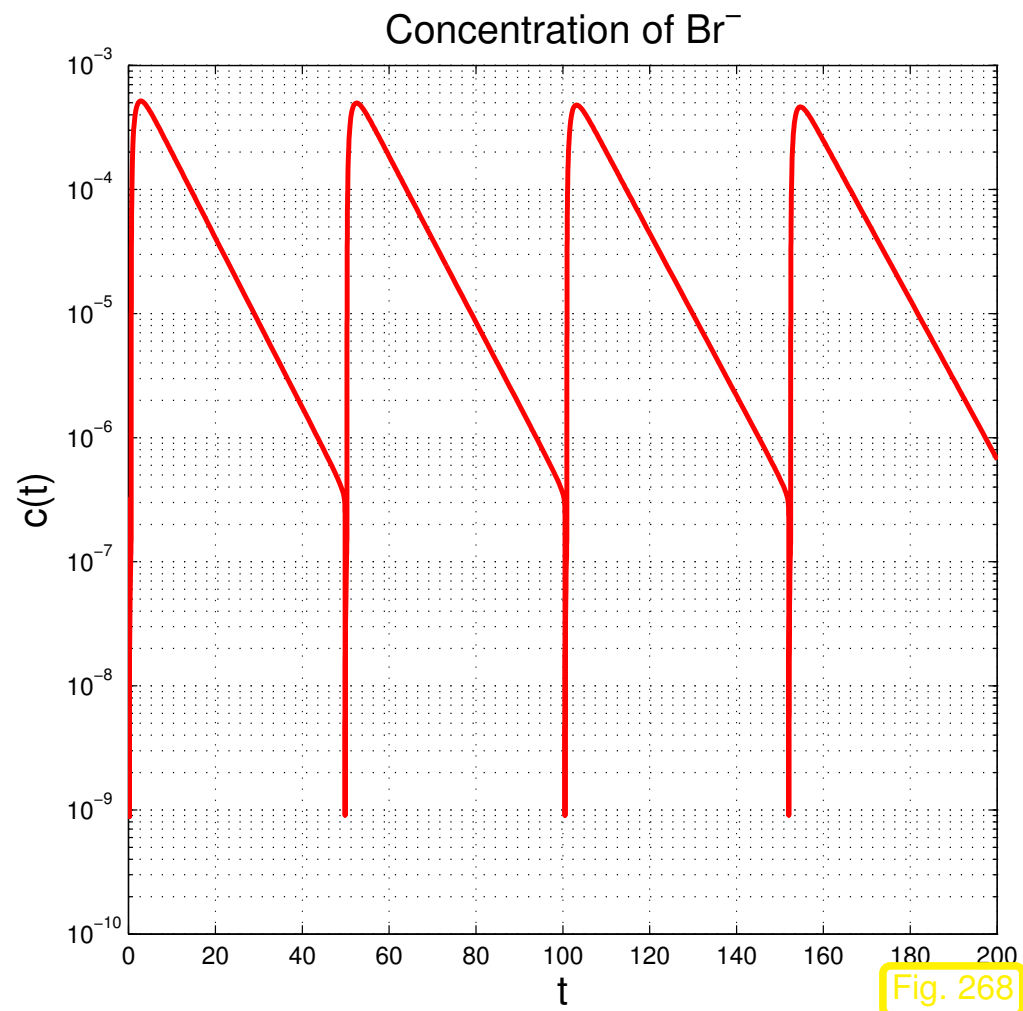
$$\begin{aligned}
 y_1 := c(\text{BrO}_3^-): \quad \dot{y}_1 &= -k_1 y_1 y_2 - k_3 y_1 y_3, \\
 y_2 := c(\text{Br}^-): \quad \dot{y}_2 &= -k_1 y_1 y_2 - k_2 y_2 y_3 + k_5 y_5, \\
 y_3 := c(\text{HBrO}_2): \quad \dot{y}_3 &= k_1 y_1 y_2 - k_2 y_2 y_3 + k_3 y_1 y_3 - 2k_4 y_3^2, \\
 y_4 := c(\text{Org}): \quad \dot{y}_4 &= k_2 y_2 y_3 + k_4 y_3^2, \\
 y_5 := c(\text{Ce(IV)}): \quad \dot{y}_5 &= k_3 y_1 y_3 - k_5 y_5,
 \end{aligned}
 \tag{12.5.3}$$

with (non-dimensionalized) reaction constants:

$$k_1 = 1.34, \quad k_2 = 1.6 \cdot 10^9, \quad k_3 = 8.0 \cdot 10^3, \quad k_4 = 4.0 \cdot 10^7, \quad k_5 = 1.0.$$

**periodic** chemical reaction  Video 1, Video 2

MATLAB simulation with initial state  $y_1(0) = 0.06$ ,  $y_2(0) = 0.33 \cdot 10^{-6}$ ,  $y_3(0) = 0.501 \cdot 10^{-10}$ ,  
 $y_4(0) = 0.03$ ,  $y_5(0) = 0.24 \cdot 10^{-7}$ :



We observe a strongly non-uniform behavior of the solution in time.

This is very common with evolutions arising from practical models (circuit models, chemical reaction models, mechanical systems)

Example 12.5.4 (Blow-up).

Scalar autonomous IVP:

$$\dot{y} = y^2, \quad y(0) = y_0 > 0.$$

$$\blacktriangleright \quad y(t) = \frac{y_0}{1 - y_0 t}, \quad t < 1/y_0.$$

Solution exists only for finite time and then suffers a **Blow-up**, that is,  $\lim_{t \rightarrow 1/y_0} y(t) = \infty$ :  $J(y_0) = ] - \infty, 1/y_0 ]$ !

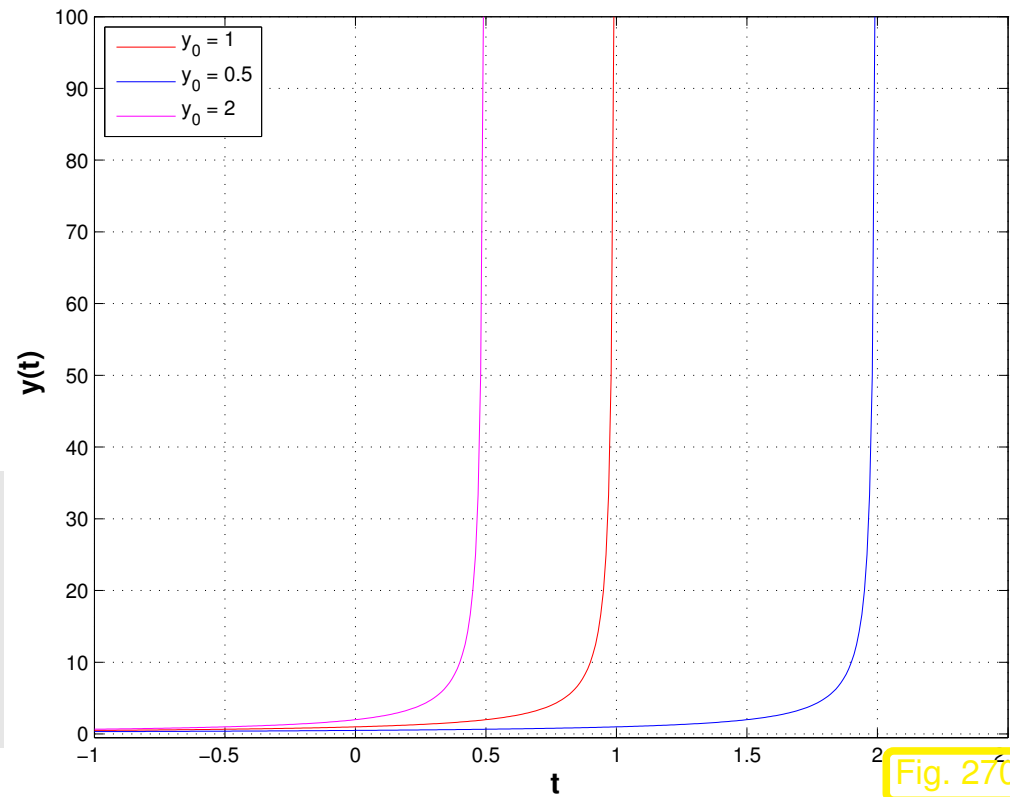
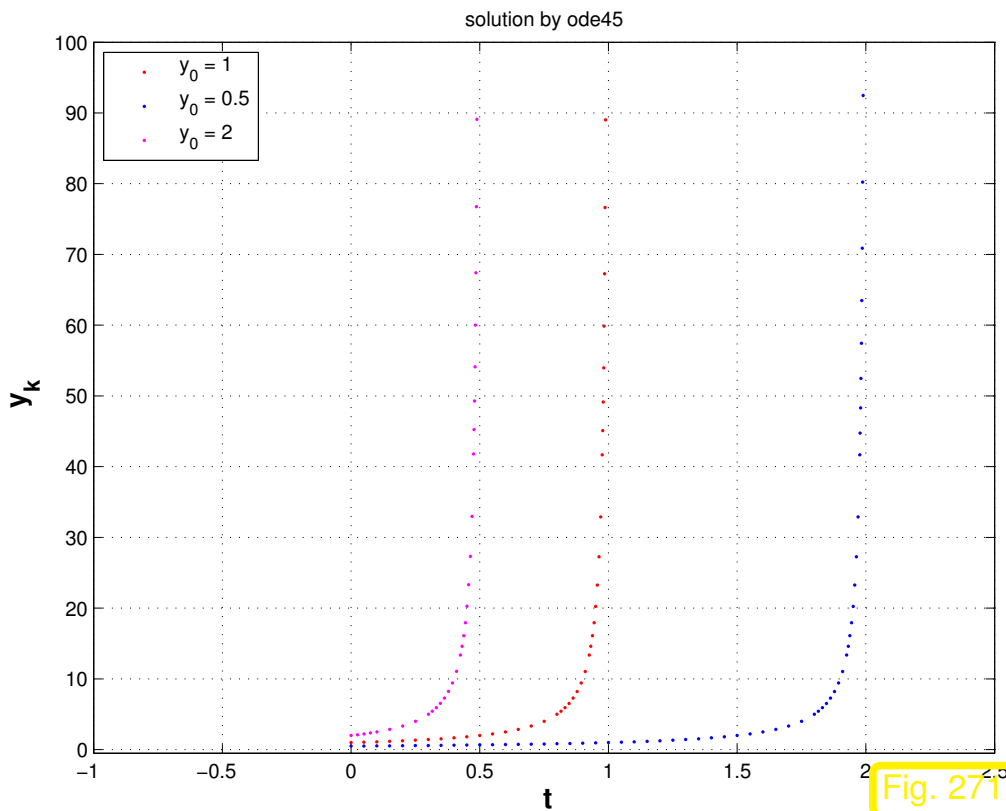


Fig. 270

R. Hiptmair  
rev 38286,  
May 3,  
2011

How to choose temporal mesh  $\{t_0 < t_1 < \dots < t_{N-1} < t_N\}$  for single step method in case  $J(y_0)$  is not known, even worse, if it is not clear a priori that a blow up will happen?

Just imagine: what will result from equidistant explicit Euler integration (12.2.4) applied to the above IVP?



```

1 fun = @(t,y) y.^2;
2 [t1,y1] = ode45(fun,[0 2],1);
3 [t2,y2] = ode45(fun,[0 2],0.5);
4 [t3,y3] = ode45(fun,[0 2],2);

```

## MATLAB warning messages:

```
Warning: Failure at t=9.999694e-01. Unable to meet integration
tolerances without reducing the step size below the smallest
value allowed (1.776357e-15) at time t.
```

```
> In ode45 at 371
   In simpleblowup at 22
```

```
Warning: Failure at t=1.999970e+00. Unable to meet integration
tolerances without reducing the step size below the smallest
value allowed (3.552714e-15) at time t.
```

```
> In ode45 at 371
   In simpleblowup at 23
```

```
Warning: Failure at t=4.999660e-01. Unable to meet integration
```

tolerances without reducing the step size below the smallest value allowed ( $8.881784e-16$ ) at time  $t$ .

```
> In ode45 at 371  
In simpleblowup at 24
```

We observe: `ode45` manages to reduce stepsize more and more as it approaches the singularity of the solution!



Key issue (discussed for autonomous ODEs below):

Choice of *good temporal mesh*  $\{0 = t_0 < t_1 < \dots < t_{N-1} < t_N\}$   
for a given single step method applied to an IVP

What does “good” mean ?



be efficient

be accurate

*Objective:*  $N$  as small as possible &  $\max_{k=1,\dots,N} \|\mathbf{y}(t_k) - \mathbf{y}_k\| < \text{TOL}$ , TOL = tolerance  
 or  $\|\mathbf{y}(T) - \mathbf{y}_N\| < \text{TOL}$

*Policy:* Try to curb/balance **one-step error** by

- adjusting *current* stepsize  $h_k$ ,
- predicting suitable *next* timestep  $h_{k+1}$

} **local-in-time  
stepsize control**

*Tool:* **Local-in-time** one-step error estimator (*a posteriori*, based on  $\mathbf{y}_k, h_{k-1}$ )

Why local-in-time timestep control (based on estimating the one-step error)?

Consideration: If a small time-local error in a single timestep leads to large error  $\|\mathbf{y}_k - \mathbf{y}(t_k)\|$  at later times, then local-in-time timestep control is powerless about it and will not even notice!!

Nevertheless, local-in-time timestep control is used almost exclusively,

- ☞ because we do not want to discard past timesteps, which could amount to tremendous waste of computational resources,
- ☞ because it is inexpensive and it works for many practical problems,
- ☞ because there is no reliable method that can deliver guaranteed accuracy for general IVP.

“Recycle” heuristics already employed for adaptive quadrature, see Sect. 10.5:

Idea:

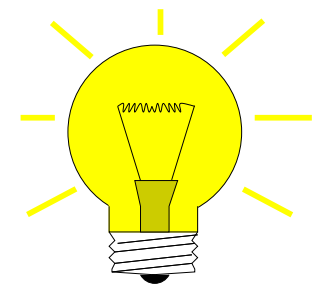
*Estimation of one-step error, cf. Sect. 10.5*

Compare two discrete evolutions  $\Psi^h, \tilde{\Psi}^h$  of different order for *current timestep*  $h$ :

If  $\text{Order}(\tilde{\Psi}) > \text{Order}(\Psi)$

$$\Rightarrow \underbrace{\Phi^h \mathbf{y}(t_k) - \Psi^h \mathbf{y}(t_k)}_{\text{one-step error}} \approx \text{EST}_k := \tilde{\Psi}^h \mathbf{y}(t_k) - \Psi^h \mathbf{y}(t_k) . \quad (12.5.5)$$

**Heuristics** for concrete  $h$



Compare  $EST_k \leftrightarrow ATOL$   $\triangleright$  Reject/accept current step (12.5.6)  
 $EST_k \leftrightarrow RTOL \|\mathbf{y}_k\|$

absolute tolerance relative tolerance

Simple algorithm:

$EST_k < \max\{ATOL, \|\mathbf{y}_k\| RTOL\}$ : Carry out next timestep (stepsize  $h$ )  
 Use larger stepsize (e.g.,  $\alpha h$  with some  $\alpha > 1$ ) for following step  
 (\*)

$EST_k > \max\{ATOL, \|\mathbf{y}_k\| RTOL\}$ : Repeat current step with smaller stepsize  $< h$ , e.g.,  $\frac{1}{2}h$

Rationale for (\*): if the current stepsize guarantees sufficiently small one-step error, then it might be possible to obtain a still acceptable one-step error with a larger timestep, which would enhance efficiency (fewer timesteps for total numerical integration). This should be tried, since timestep control will usually provide a safeguard against undue loss of accuracy.

```

1 function [t,y] =
   odeintadapt(Psilow, Psihigh, T, y0, h0, reltol, abstol, hmin)
2 t = 0; y = y0; h = h0; %
3 while ((t(end) < T) (h > hmin)) %
4   yh = Psihigh(h, y0); % high order discrete evolution  $\tilde{\Psi}^h$ 
5   yH = Psilow(h, y0); % low order discrete evolution  $\Psi^h$ 
6   est = norm(yH-yh); %  $\leftrightarrow$   $EST_k$ 
7
8   if (est < max(reltol*norm(y0), abstol)) %
9     y0 = yh; y = [y, y0]; t = [t, t(end) + min(T-t(end), h)]; %
10    h = 1.1*h; % step accepted, try with increased stepsize
11  else, h = h/2; end % step rejected, try with half the stepsize
12 end

```

## Comments on Code 12.5.6:

- Input arguments:

- `PsiLow`, `PsiHigh`: function handles to discrete evolution operators for autonomous ODE of different order, type  $@(y, h)$ , expecting a state (column) vector as first argument, and a stepsize as second,
  - `T`: final time  $T > 0$ ,
  - `y0`: initial state  $y_0$ ,
  - `h0`: stepsize  $h_0$  for the first timestep
  - `reltol`, `abstol`: relative and absolute tolerances, see (12.5.6),
  - `hmin`: minimal stepsize, timestepping terminates when stepsize control  $h_k < h_{\min}$ , which is relevant for detecting blow-ups or collapse of the solution.
- 
- line 3: check whether final time is reached or timestepping has ground to a halt ( $h_k < h_{\min}$ ).
  - line 4, 5: advance state by low and high order integrator.
  - line 6: compute norm of estimated error, see (12.5.5).
  - line 8: make comparison (12.5.6) to decide whether to accept or reject local step.
  - line 9, 10: step accepted, update state and current time and suggest 1.1 times the current stepsize for next step.
  - line 11 step rejected, try again with half the stepsize.
  - Return values:

- $t$ : temporal mesh  $t_0 < t_1 < t_2 < \dots < t_N < T$ , where  $t_N < T$  indicated premature termination (collapse, blow-up),
- $y$ : sequence  $(\mathbf{y}_k)_{k=0}^N$ .

By the heuristic considerations, see (12.5.5) it seems that  $EST_k$  measures the one-step error for the low-order method  $\Psi$  and that we should use  $\mathbf{y}_{k+1} = \Psi^{h_k} \mathbf{y}_k$ , if the timestep is accepted.

However, it would be foolish not to use the better value  $\mathbf{y}_{k+1} = \tilde{\Psi}^{h_k} \mathbf{y}_k$ , since it is available for free. This is what is done in every implementation of adaptive methods, also in Code 12.5.6, and this choice can be justified by control theoretic arguments [16, Sect. 5.2].

*Example 12.5.8* (Simple adaptive stepsize control).

- IVP for ODE  $\dot{y} = \cos(\alpha y)^2$ ,  $\alpha > 0$ , solution  $y(t) = \arctan(\alpha(t - c))/\alpha$  for  $y(0) \in ]-\pi/2, \pi/2[$
- Simple adaptive timestepping based on explicit Euler (12.2.4) and explicit trapezoidal rule (12.4.5)

## Code 12.5.9: MATLAB function for Ex. 12.5.8

```

1 function odeintadaptdriver(T,a,reltol,abstol)
2 % Simple adaptive timestepping strategy of Code 12.5.6
3 % based on explicit Euler (12.2.4) and explicit trapezoidal
4 % rule (12.4.5)
5
6 % Default arguments
7 if (nargin < 4), abstol = 1E-4; end
8 if (nargin < 3), reltol = 1E-2; end
9 if (nargin < 2), a = 20; end
10 if (nargin < 1), T = 2; end
11
12 % autonomous ODE  $\dot{y} = \cos(ay)$  and its general solution
13 f = @(y) (cos(a*y).^2); sol = @(t) (atan(a*(t-1))/a);
14 % Initial state  $y_0$ 
15 y0 = sol(0);
16
17 % Discrete evolution operators, see Def. 12.2.12
18 Psilow = @(h,y) (y + h*f(y)); % Explicit Euler (12.2.4)
19 % Explicit trapzoidal rule (12.4.5)
20 Psihigh = @(h,y) (y + 0.5*h*(f(y)+f(y+h*f(y))));
21
22 % Heuristic choice of initial timestep and  $h_{\min}$ 
23 h0 = T/(100*(norm(f(y0))+0.1)); hmin = h0/10000;
24 % Main adaptive timestepping loop, see Code 12.5.6
25 [t,y,rej,ee] = odeintadapt_ext(Psilow,Psihigh,T,y0,h0,reltol,abstol,hmin);
26
27 % Plotting the exact the approximate solutions and rejected timesteps
28 figure ('name', 'solutions');

```

```
29 tp = 0:T/1000:T; plot(tp,sol(tp),'g-','linewidth',2); hold on;
30 plot(t,y,'r.');
```

31 plot(rej,0,'m+');

32 title(sprintf('Adaptive timestepping, rtol = %f, atol = %f, a = %f',reltol,abstol,a));

33 xlabel('\bf t','fontsize',14);

34 ylabel('\bf y','fontsize',14);

35 legend('y(t)','y\_k','rejection','location','northwest');

36 print -depsc2 '../PICTURES/odeintadaptsol.eps';

37

38 fprintf('%d timesteps, %d rejected timesteps\n',length(t)-1,length(rej));

39

40 *% Plotting estimated and true errors*

41 figure('name','(estimated) error');

42 plot(t,abs(sol(t) - y),'r+',t,ee,'m\*');

43 xlabel('\bf t','fontsize',14);

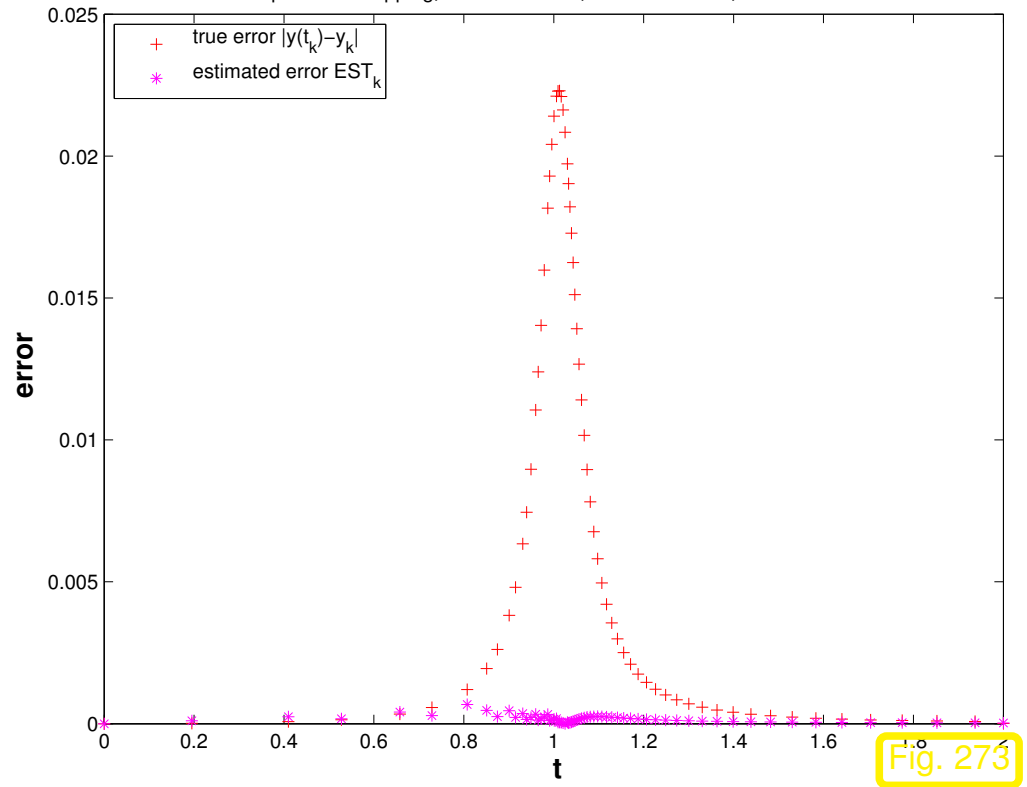
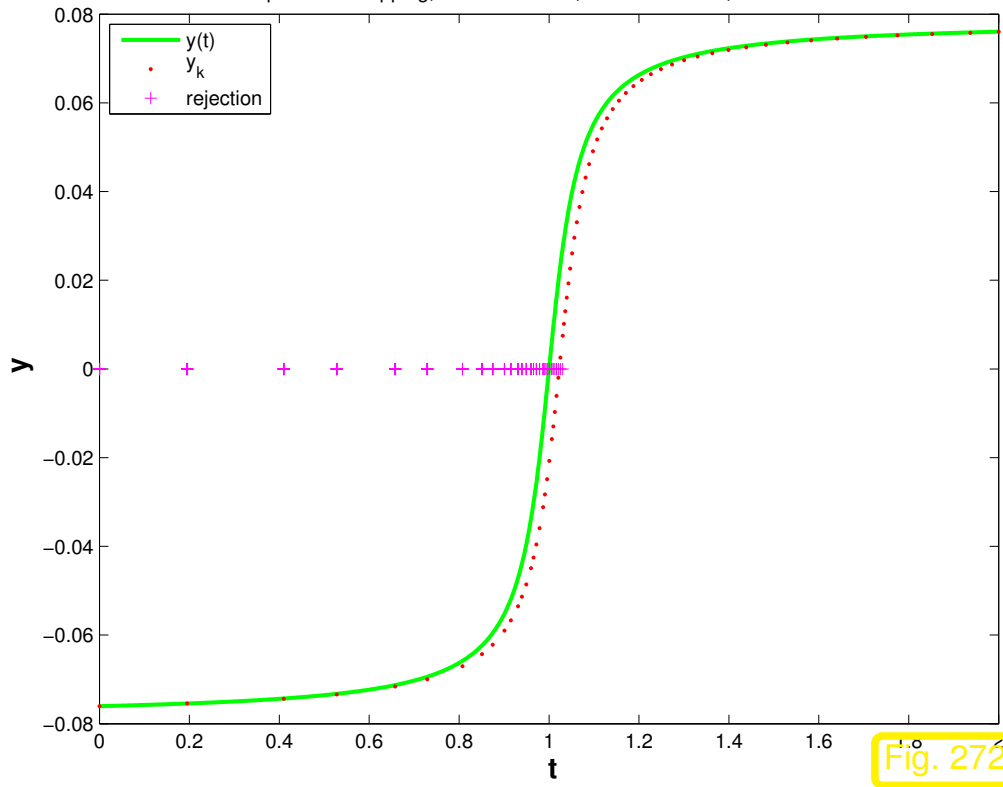
44 ylabel('\bf error','fontsize',14);

45 legend('true error |y(t\_k)-y\_k|','estimated error EST\_k','location','northwest');

46 title(sprintf('Adaptive timestepping, rtol = %f, atol = %f, a = %f',reltol,abstol,a));

47 print -depsc2 '../PICTURES/odeintadapterr.eps';





Statistics: 66 timesteps, 131 rejected timesteps

Observations:

- ☞ Adaptive timestepping well resolves local features of solution  $y(t)$  at  $t = 1$
- ☞ Estimated error (an estimate for the one-step error) and true error are **not** related!

*Example* 12.5.10 (Gain through adaptivity). → Ex. 12.5.8

Simple adaptive timestepping from previous experiment Ex. 12.5.8.

New: initial state  $y(0) = 0!$

Now we study the dependence of the maximal point error on the computational effort, which is proportional to the number of timesteps.

R. Hiptmair  
rev 38286,  
May 3,  
2011

Code 12.5.11: MATLAB function for Ex. 12.5.10

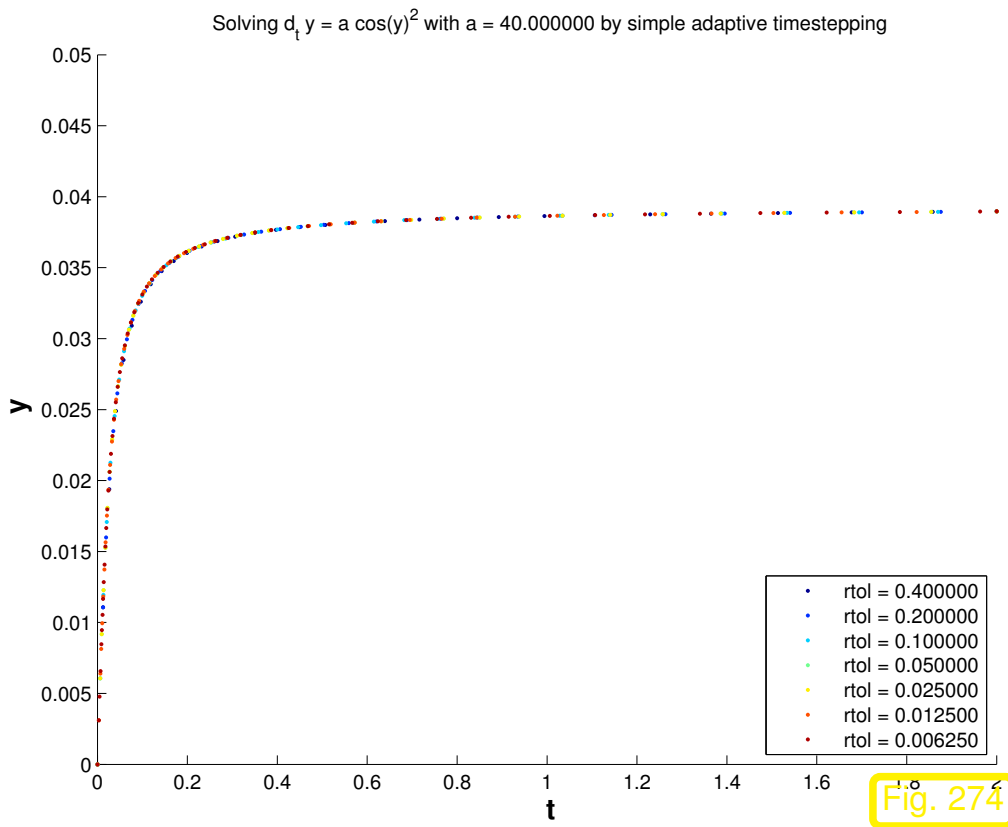
```
1 function adaptgain(T,a,reltol,abstol)
2 % Experimental study of gain through simple adaptive timestepping
3 % strategy of Code 12.5.6 based on explicit Euler
4 % (12.2.4) and explicit trapezoidal
```

12.5  
p. 1204

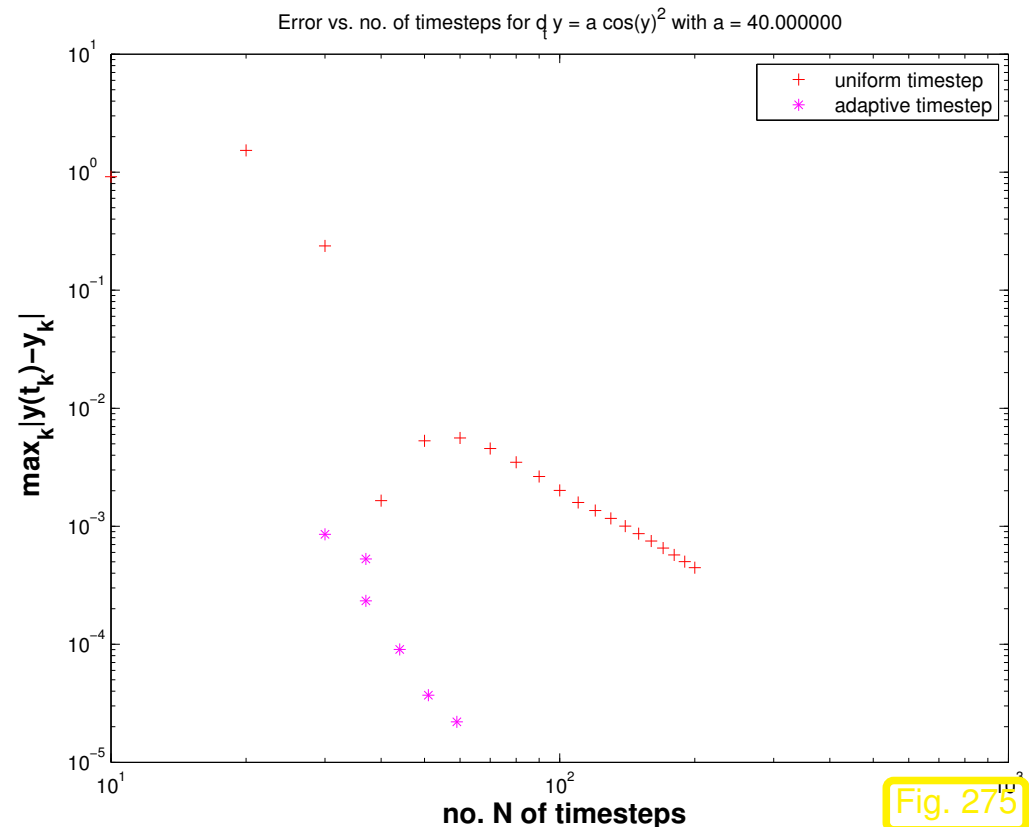
```
5 % rule (12.4.5)
6
7 % Default arguments
8 if (nargin < 4), abstol = 1E-3; end
9 if (nargin < 3), reltol = 1E-1; end
10 if (nargin < 2), a = 40; end
11 if (nargin < 1), T = 2; end
12
13 % autonomous ODE  $\dot{y} = \cos(ay)$  and its general solution
14 f = @(y) (cos(a*y).^2); sol = @(t) (atan(a*(t))/a);
15 % Initial state  $y_0$ 
16 y0 = sol(0);
17
18 % Discrete evolution operators, see Def. 12.2.12
19 PsiLow = @(h,y) (y + h*f(y)); % Explicit Euler (12.2.4)
20 % Explicit trapezoidal rule (12.4.5)
21 PsiHigh = @(h,y) (y + 0.5*h*(f(y)+f(y+h*f(y))));
22
23 % Loop over uniform timesteps of varying length and integrate ODE by explicit
   trapezoidal
24 % rule (12.4.5)
25 erruf = [];
26 for N=10:10:200
27     h = T/N; t = 0; y = y0; err = 0;
```

```
28  for k=1:N
29      y = Psihigh(h,y); t = t+h;
30      err = max(err,abs(sol(t) - y));
31  end
32  erruf = [erruf;N, err];
33  end
34
35  % Run adaptive timestepping with various tolerances, which is the only way
36  % to make it use a different total number of timesteps.
37  % Plot the solution sequences for different values of the relative tolerance.
38  figure('name','adaptive timestepping');
39  axis([0 2 0 0.05]); hold on; col = colormap;
40  errad = []; l = 1;
41  for rtol=reltol*2.^(2:-1:-4)
42      % Crude choice of initial timestep and  $h_{\min}$ 
43      h0 = T/10; hmin = h0/10000;
44      % Main adaptive timestepping loop, see Code 12.5.6
45      [t,y,rej,ee] =
46          odeintadapt_ext(Psilow,Psihigh,T,y0,h0,rtol,0.01*rtol,hmin);
47      errad = [errad; length(t)-1, max(abs(sol(t) - y)), rtol,
48              length(rej)];
49      fprintf('rtol = %d: %d timesteps, %d rejected timesteps\n',
50              rtol,length(t)-1,length(rej));
```

```
48 plot(t,y,'.','color',col(10*(l-1)+1,:));
49 leg{l} = sprintf('rtol = %f',rtol); l = l+1;
50 end
51 xlabel ('\bf t','fontsize',14);
52 ylabel ('\bf y','fontsize',14);
53 legend(leg,'location','southeast');
54 title (sprintf('Solving d_t y = a cos(y)^2 with a = %f by simple
    adaptive timestepping',a));
55 print -depsc2 '../PICTURES/adaptgainsol.eps';
56
57 % Plotting the errors vs. the number of timesteps
58 figure('name','gain by adaptivity');
59 loglog(erruf(:,1), erruf(:,2),'r+',errad(:,1), errad(:,2),'m*');
60 xlabel ('\bf no. N of timesteps','fontsize',14);
61 ylabel ('\bf max_k|y(t_k)-y_k|','fontsize',14);
62 title (sprintf('Error vs. no. of timesteps for d_t y = a cos(y)^2
    with a = %f',a));
63 legend('uniform timestep','adaptive
    timestep','location','northeast');
64 print -depsc2 '../PICTURES/adaptgain.eps';
```



Solutions  $(y_k)_k$  for different values of  $rtol$



Error vs. computational effort

Observations:

☞ Adaptive timestepping achieves much better accuracy for a fixed computational effort.

Example 12.5.12 (“Failure” of adaptive timestepping). → Ex. 12.5.10

Same ODE and simple adaptive timestepping as in previous experiment Ex. 12.5.10. Same evaluations.

Now: initial state  $y(0) = -0.0386$  as in Ex. 12.5.8

Solving  $d_t y = a \cos(y)^2$  with  $a = 40.000000$  by simple adaptive timestepping

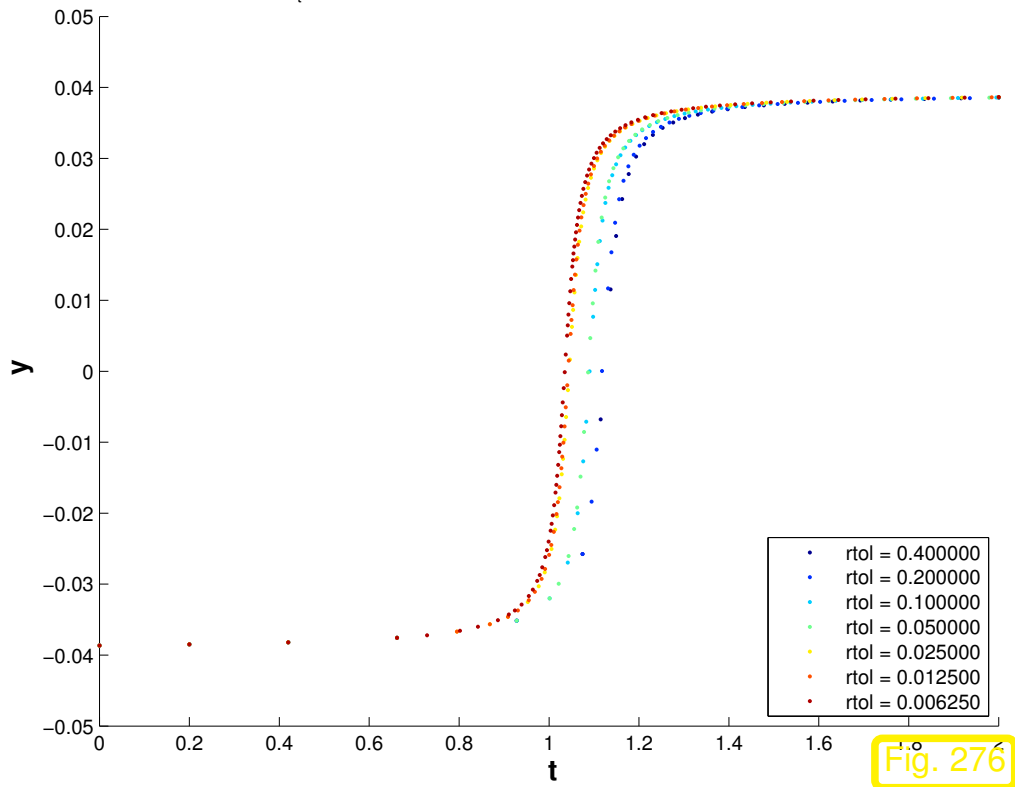


Fig. 276

Solutions  $(y_k)_k$  for different values of `rtol`

Error vs. no. of timesteps for  $d_t y = a \cos(y)^2$  with  $a = 40.000000$

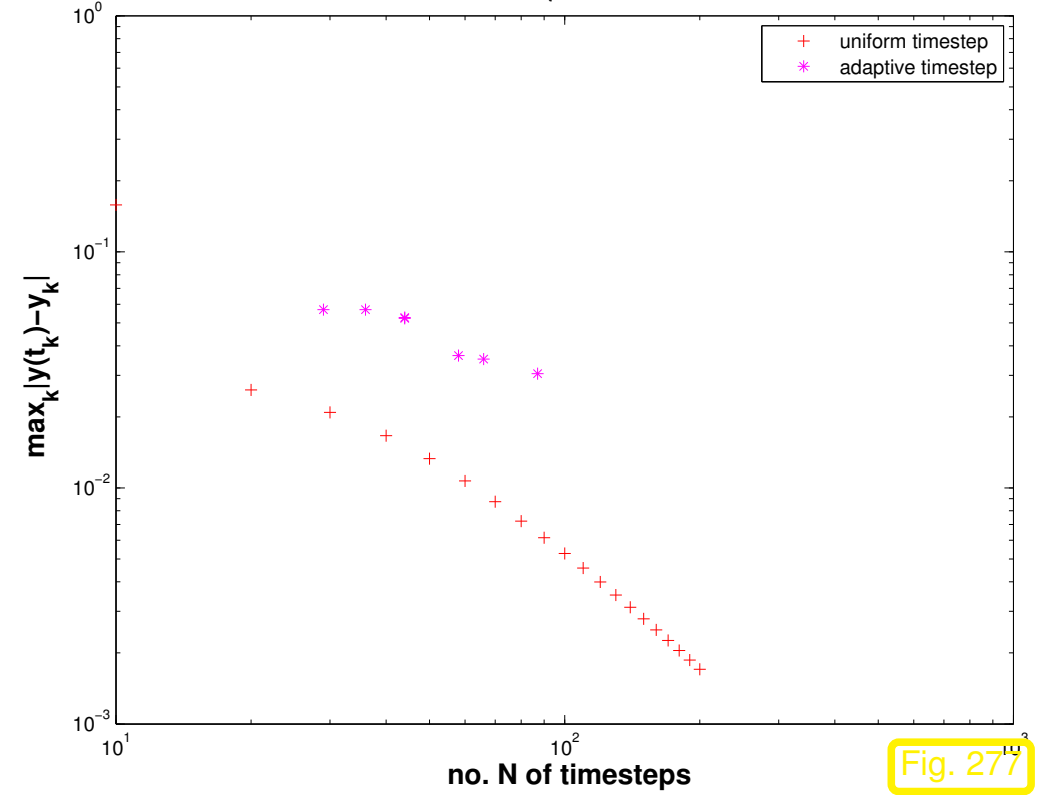


Fig. 277

Error vs. computational effort

## Observations:

- ☞ Adaptive timestepping leads to larger errors at the same computational cost as uniform timestepping.

Explanation: the position of the steep step of the solution has a sensitive dependence on an initial value  $y(0) \approx -0.04$ . Hence, small local errors in the initial timesteps will lead to large errors at around time  $t \approx 1$ . The stepsize control is mistaken in condoning these small one-step errors in the first few steps and, therefore, incurs huge errors later.





*Remark* 12.5.13 (Refined local stepsize control).

The above algorithm (Code 12.5.6) is simple, but the rule for increasing/shrinking of timestep arbitrary “wastes” information contained in  $EST_k : TOL$ :

More ambitious goal !      When  $EST_k > TOL$  : stepsize adjustment    better  $h_k = ?$   
    When  $EST_k < TOL$  : stepsize prediction    good  $h_{k+1} = ?$

Assumption: At our disposal are two discrete evolutions:

- $\Psi$  with  $order(\Psi) = p$  (→ “low order” single step method)
- $\tilde{\Psi}$  with  $order(\tilde{\Psi}) > p$  (→ “higher order” single step method)

These are the same building blocks as for the simple adaptive strategy employed in Code 12.5.6 (, passed as arguments `Psilow`, `Psihigh` there).

Asymptotic expressions for one-step error for  $h \rightarrow 0$ :

$$\begin{aligned} \Psi^{h_k} \mathbf{y}(t_k) - \Phi^{h_k} \mathbf{y}(t_k) &= ch^{p+1} + O(h_k^{p+2}), \\ \tilde{\Psi}^{h_k} \mathbf{y}(t_k) - \Phi^{h_k} \mathbf{y}(t_k) &= O(h^{p+2}), \end{aligned} \tag{12.5.14}$$

with some  $c > 0$ .

Why  $h^{p+1}$ ? Remember estimate (12.3.8) from the error analysis of the explicit Euler method: we also found  $O(h_k^2)$  there for the one-step error of a single step method of order 1.

Heuristics: the timestep  $h$  is small  $\Rightarrow$  “higher order terms”  $O(h^{p+2})$  can be ignored.

$$\blacktriangleright \begin{aligned} \Psi^{h_k} \mathbf{y}(t_k) - \Phi^{h_k} \mathbf{y}(t_k) &\doteq ch_k^{p+1} + O(h_k^{p+2}), \\ \tilde{\Psi}^{h_k} \mathbf{y}(t_k) - \Phi^{h_k} \mathbf{y}(t_k) &\doteq O(h_k^{p+2}). \end{aligned} \Rightarrow \boxed{\text{EST}_k \doteq ch_k^{p+1}}. \quad (12.5.15)$$

 notation:  $\doteq$  equality up to higher order terms in  $h_k$

$$\text{EST}_k \doteq ch_k^{p+1} \Rightarrow c \doteq \frac{\text{EST}_k}{h_k^{p+1}}. \quad (12.5.16)$$

Available in algorithm, see (12.5.5)

For the sake of *accuracy* (stipulates “ $\text{EST}_k < \text{TOL}$ ”) & *efficiency* (favors “ $>$ ”) we aim for

$$\text{EST}_k \stackrel{!}{=} \text{TOL} := \max\{\text{ATOL}, \|\mathbf{y}_k\| \text{RTOL}\}. \quad (12.5.17)$$

What timestep  $h_*$  can actually achieve (12.5.17), if we “believe” in (12.5.15) (and, therefore, in (12.5.16))?

$$(12.5.16) \ \& \ (12.5.17) \ \Rightarrow \ \text{TOL} = \frac{\text{EST}_k}{h_k^{p+1}} h_*^{p+1} .$$



“Optimal timestep”:  
(stepsize prediction)

$$h_* = h^{p+1} \sqrt{\frac{\text{TOL}}{\text{EST}_k}} .$$

(12.5.18)

adjusted stepsize (A)

suggested stepsize  
(B)

(A): In case  $\text{EST}_k > \text{TOL}$   $\triangleright$  repeat step with stepsize  $h_*$ .

(B): If  $\text{EST}_k \leq \text{TOL}$   $\triangleright$  use  $h_*$  as stepsize for next step.

Code 12.5.19: refined local stepsize control for single step methods

```

1 function [t,y] =
   odeintssctrl(Psilow,p,Psihigh,T,y0,h0,reltol,abstol,hmin)
2 t = 0; y = y0; h = h0; %
3 while ((t(end) < T) (h > hmin)) %
4   yh = Psihigh(h,y0); % high order discrete evolution  $\tilde{\Psi}^h$ 
5   yH = Psilow(h,y0); % low order discrete evolution  $\Psi^h$ 
6   est = norm(yH-yh); %  $\leftrightarrow$   $\text{EST}_k$ 
7
8   tol = max(reltol*norm(y(:,end)),abstol); %

```

```
9 | h = h*max(0.5, min(2, (tol/est)^(1/(p+1))))); % Optimal stepsize  
   | according to (12.5.18)  
10 | if (est < tol) %  
11 |     y0 = yh; y = [y, y0]; t = [t, t(end) + min(T-t(end), h)]; % step  
   |     accepted  
12 | end  
13 | end
```

Comments on Code 12.5.18 (see comments on Code 12.5.6 for more explanations):

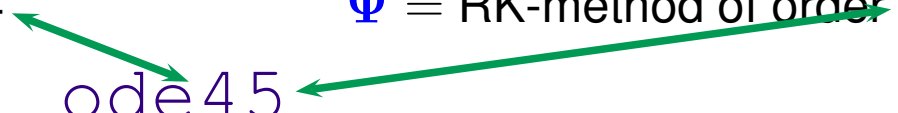
- Input arguments as for Code 12.5.6, except for  $p \hat{=}$  order of lower order discrete evolution.
- line 9: compute presumably better local stepsize according to (12.5.18),
- line 10: decide whether to repeat the step or advance,
- line 11: extend output arrays if current step has not been rejected.

*Remark 12.5.20* (Stepsize control in MATLAB).

$\Psi \hat{=}$  RK-method of order 4

$\tilde{\Psi} \hat{=}$  RK-method of order 5

ode45



Specifying tolerances for MATLAB's integrators:

```
options = odeset('abstol', atol, 'reltol', rtol, 'stats', 'on');
[t, y] = ode45(@(t, x) f(t, x), tspan, y0, options);
(f = function handle, tspan  $\hat{=}$  [t0, T], y0  $\hat{=}$  y0, t  $\hat{=}$  tk, y  $\hat{=}$  yk)
```



Example 12.5.21 (Adaptive timestepping for mechanical problem).

Movement of a point mass in a conservative force field:  $t \mapsto \mathbf{y}(t) \in \mathbb{R}^2 \hat{=}$  trajectory

Newton's law:  $\ddot{\mathbf{y}} = F(\mathbf{y}) := -\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2} .$  (12.5.22)

acceleration  $\rightarrow$   $\ddot{\mathbf{y}}$   $\leftarrow$  force

Equivalent 1st-order ODE, see Rem. 12.1.15: with velocity  $\mathbf{v} := \dot{\mathbf{y}}$

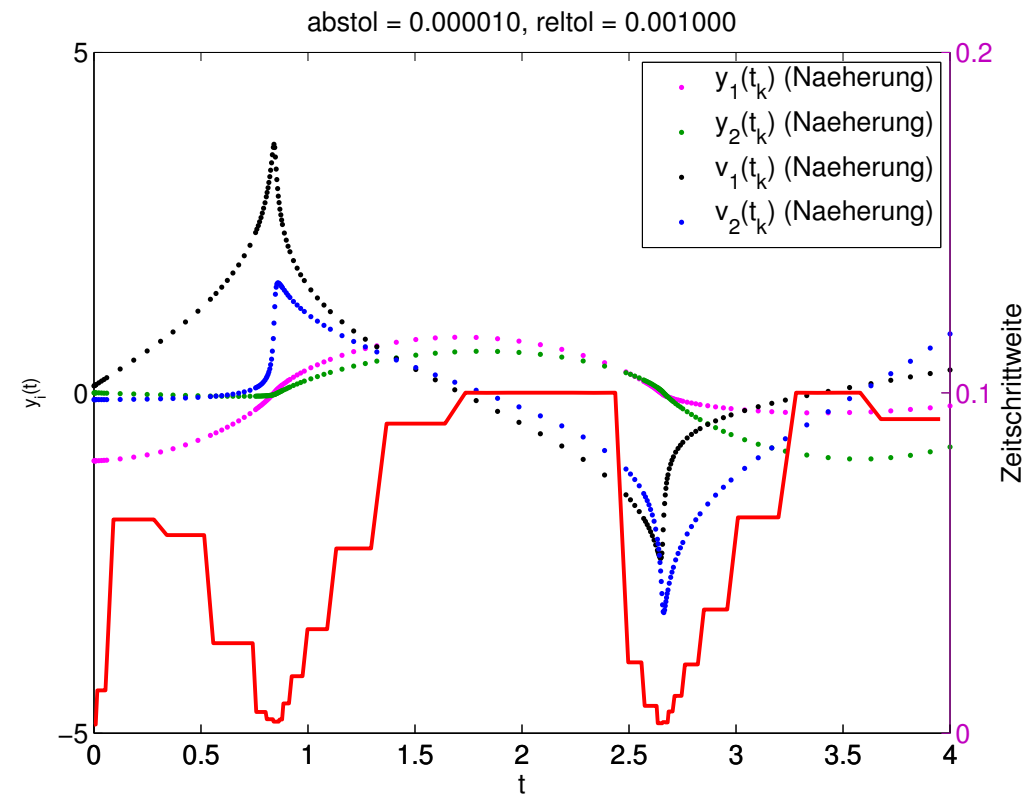
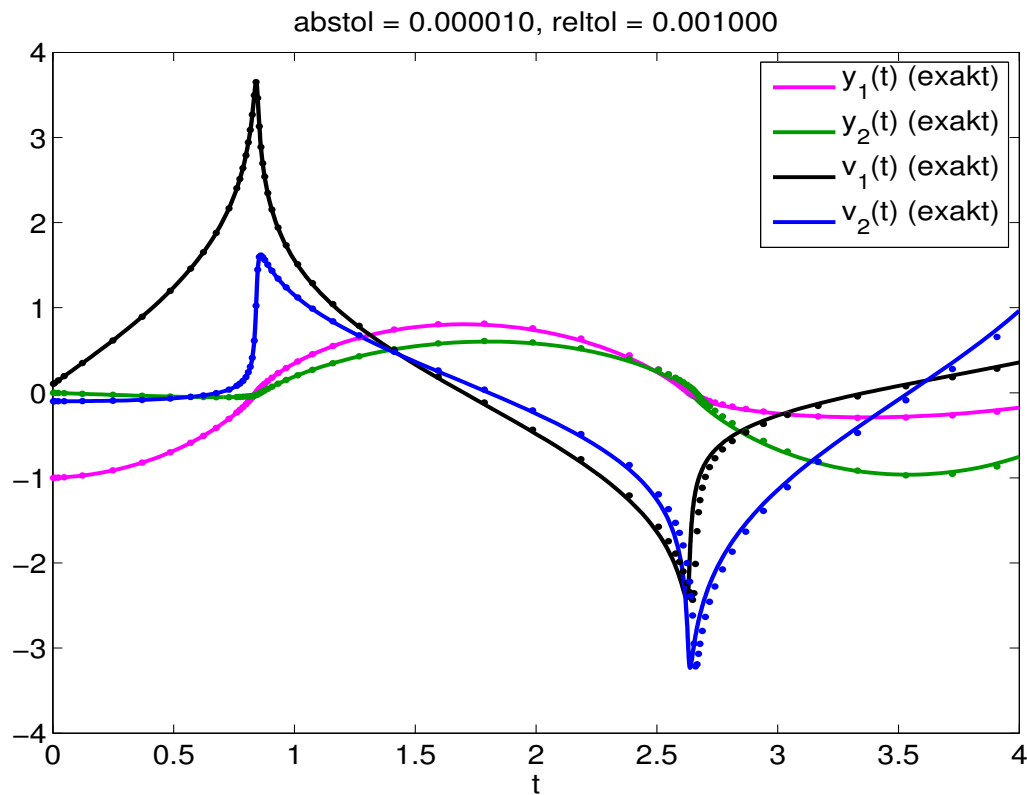
$$\begin{pmatrix} \dot{\mathbf{y}} \\ \dot{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ -\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2} \end{pmatrix} .$$
 (12.5.23)

Initial values used in the experiment:

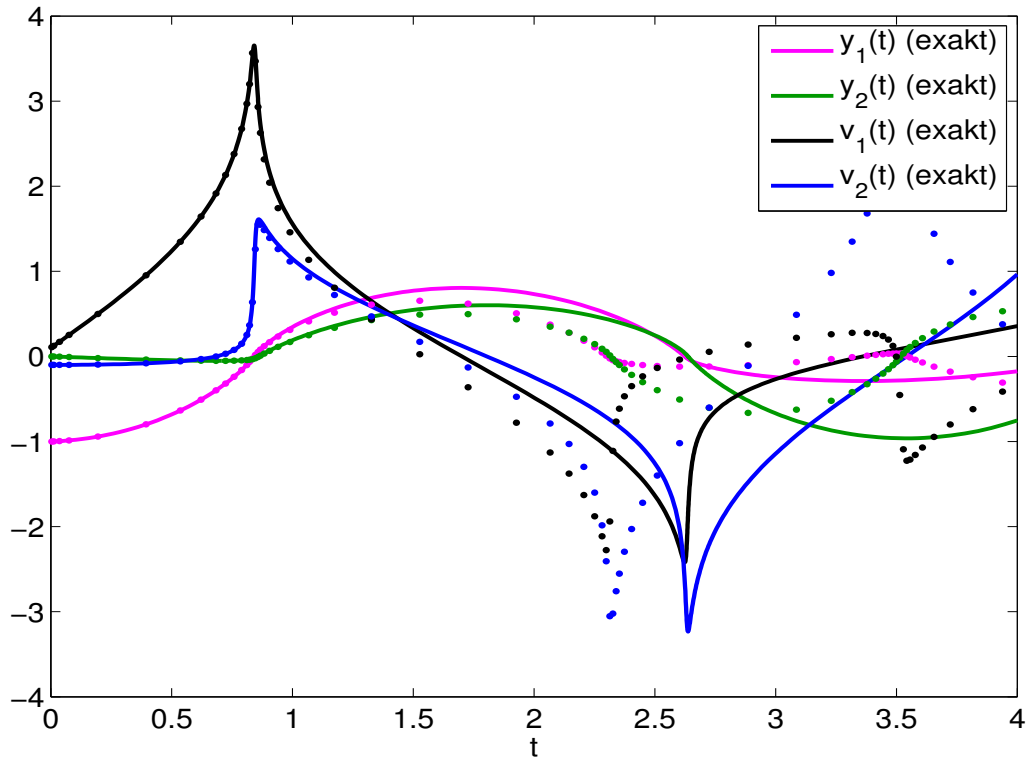
$$\mathbf{y}(0) := \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \quad \mathbf{v}(0) := \begin{pmatrix} 0.1 \\ -0.1 \end{pmatrix}$$

Adaptive integrator: `ode45(@(t,x) f(t,x), [0 4], [-1;0;0.1;-0.1], options):`

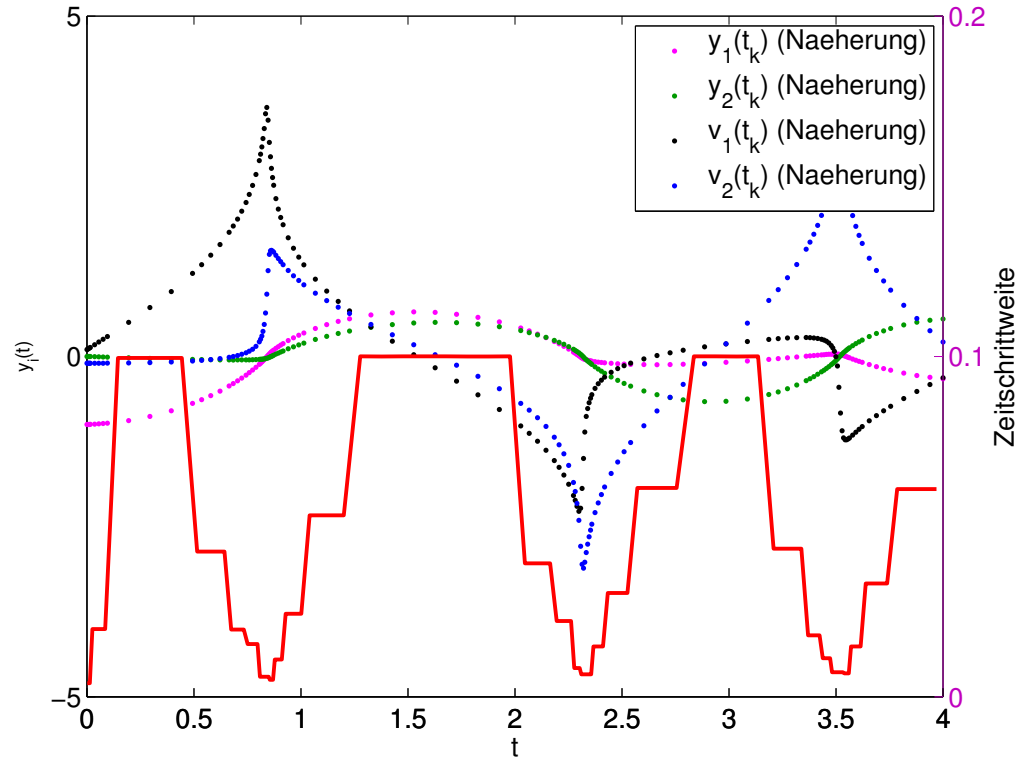
- ❶ `options = odeset('reltol',0.001,'abstol',1e-5);`
- ❷ `options = odeset('reltol',0.01,'abstol',1e-3);`

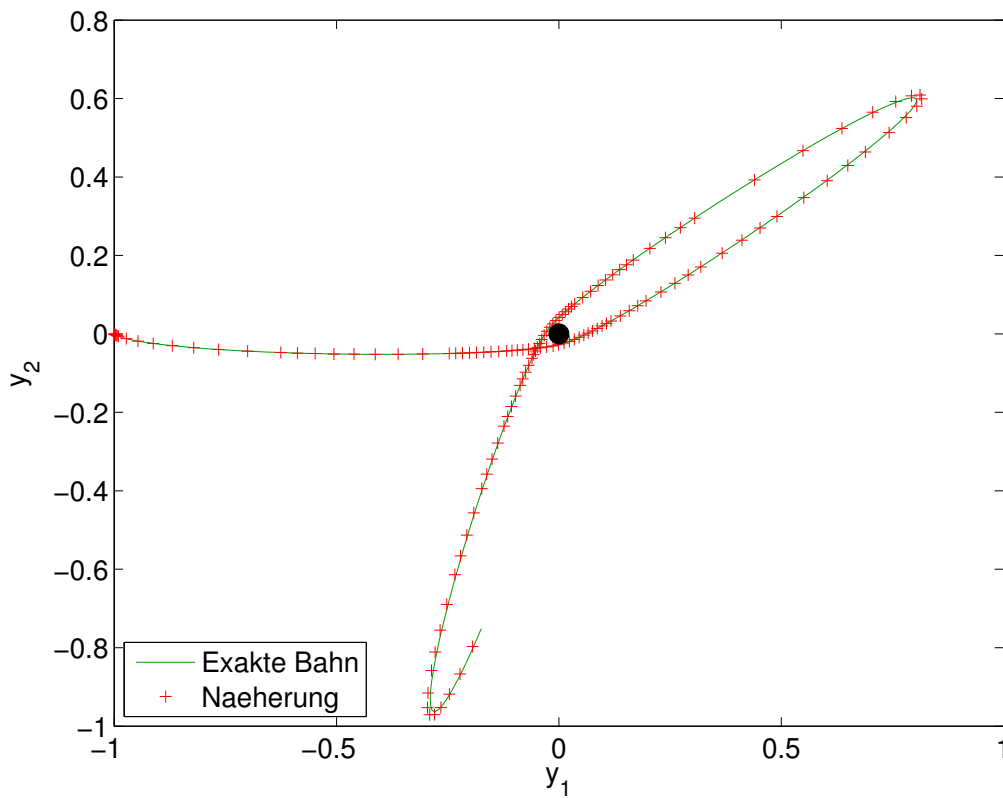


abstol = 0.001000, reltol = 0.010000

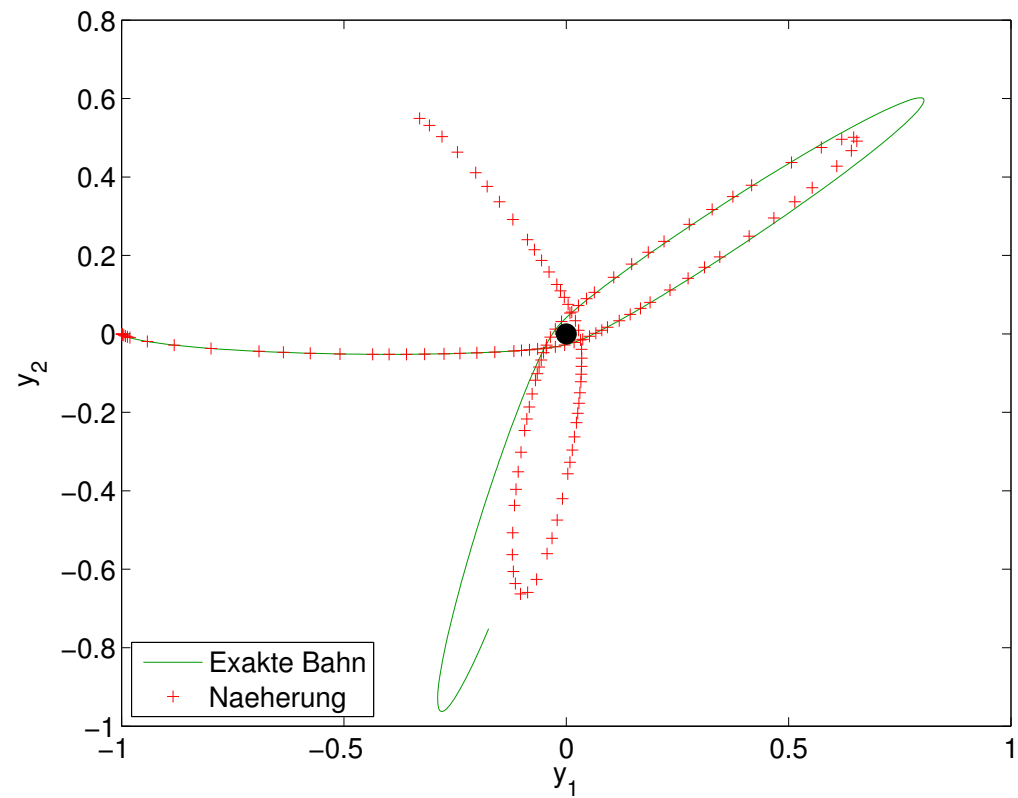


abstol = 0.001000, reltol = 0.010000





reitol=0.001, abstol=1e-5



reitol=0.01, abstol=1e-3

### Observations:

- ☞ Fast changes in solution components captured by adaptive approach through very small timesteps.
- ☞ Completely wrong solution, if tolerance reduced slightly.



An inevitable consequence of time-local error estimation:

Absolute/relative tolerances do *not* allow to predict accuracy of solution!

# 13

## Stiff Integrators [13, Sect. 11.9]

Explicit Runge-Kutta methods with stepsize control ( $\rightarrow$  Sect. 12.5) seem to be able to provide approximate solutions for any IVP with good accuracy provided that tolerances are set appropriately.

Everything settled about numerical integration?

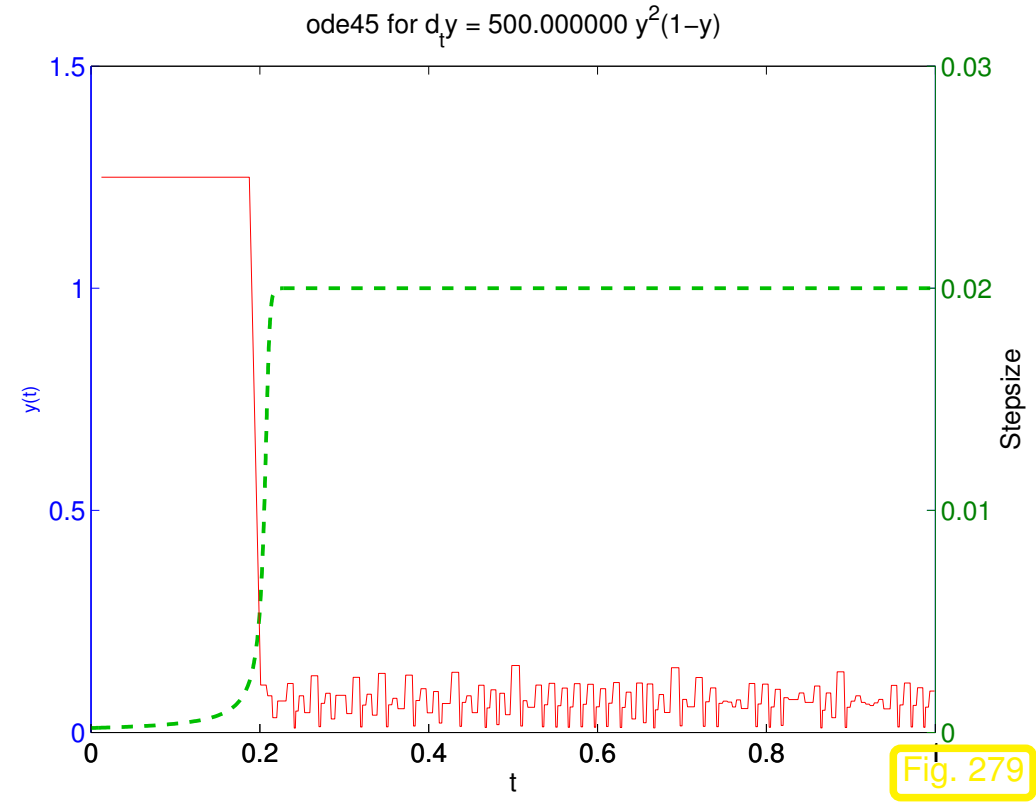
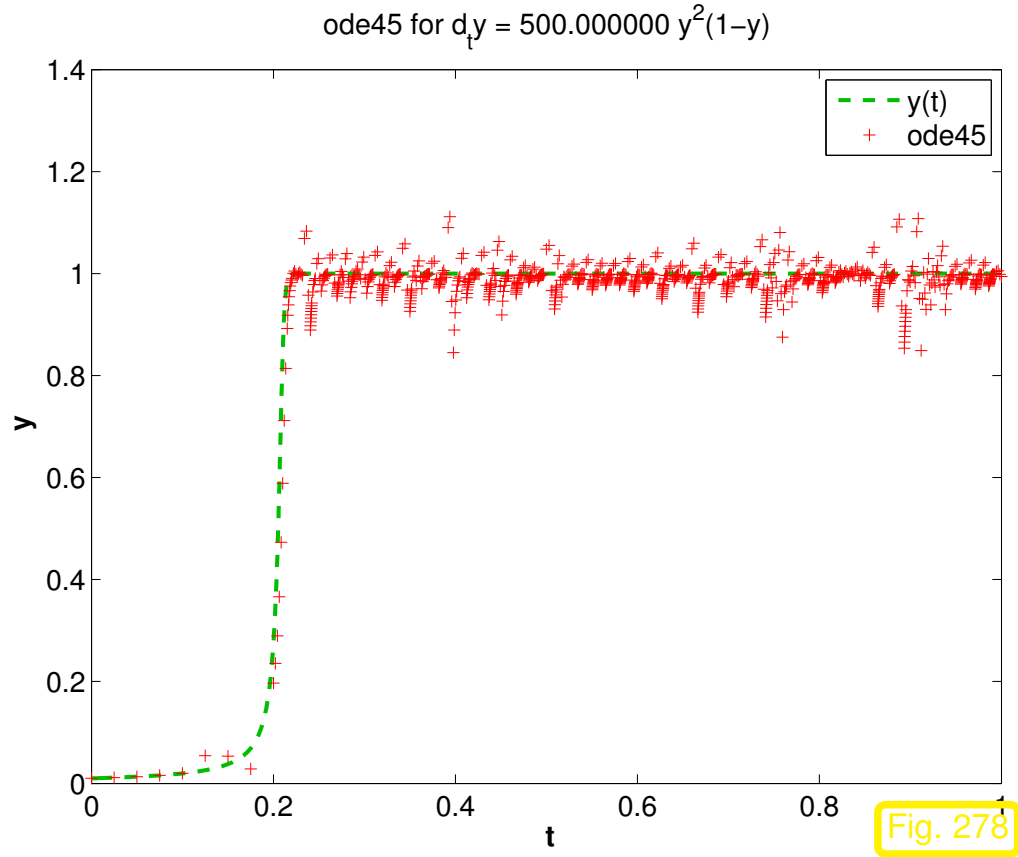
*Example 13.0.1* (ode45 for stiff problem).

$$\text{IVP: } \quad \dot{y} = \lambda y^2(1 - y), \quad \lambda := 500, \quad y(0) = \frac{1}{100}.$$

```
1 fun = @(t,x) 500*x^2*(1-x);  
2 options = odeset('reltol',0.1,'abstol',0.001,'stats','on');  
3 [t,y] = ode45(fun,[0 1],y0,options);
```

The option `stats = 'on'` makes MATLAB print statistics about the run of the integrators.

186 successful steps  
55 failed attempts  
1447 function evaluations



Stepsize control of `ode45` running amok!



The solution is virtually constant from  $t > 0.2$  and, nevertheless, the integrator uses tiny timesteps until the end of the integration interval.



## 13.1 Model problem analysis [35, Ch. 77], [51, Sect. 11.3.3]

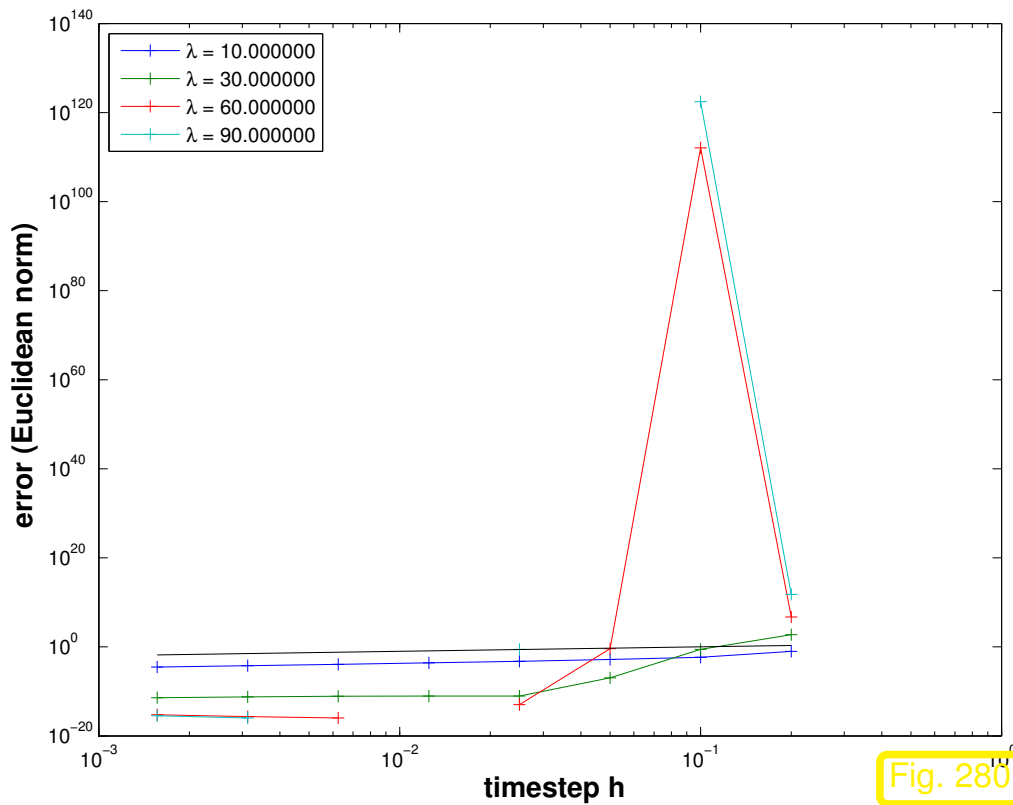
*Example 13.1.1* (Blow-up of explicit Euler method).

As in part II of Ex. 12.3.2:

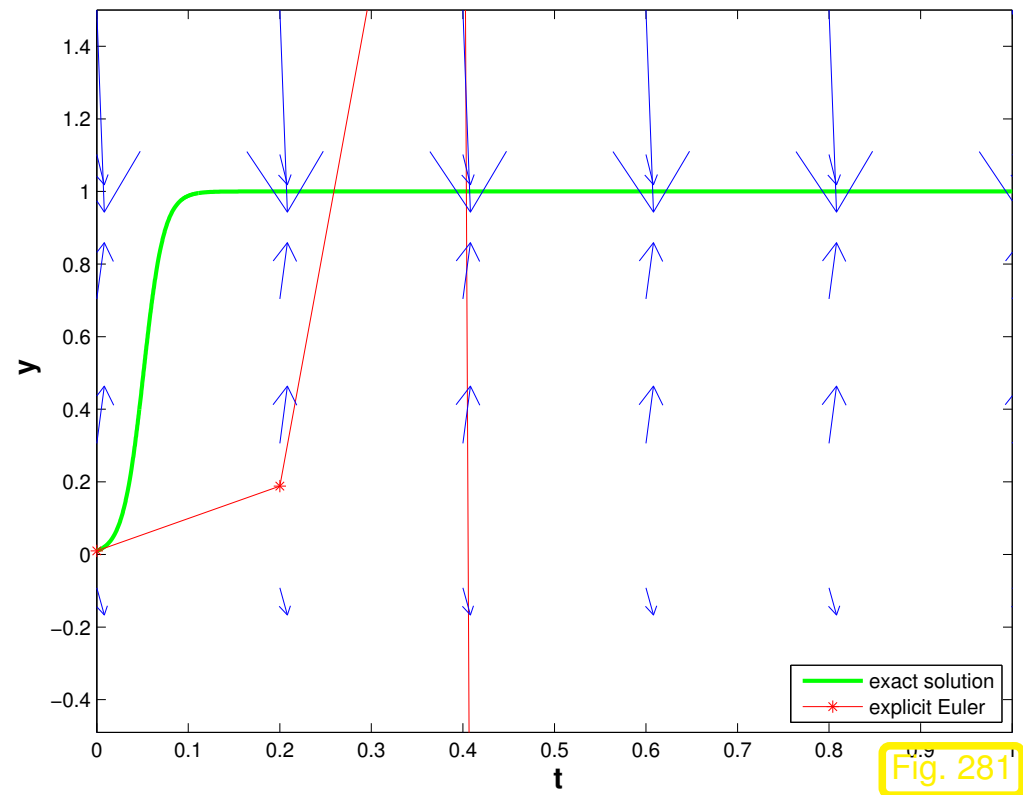
• IVP for logistic ODE, see Ex. 12.1.1

$$\dot{y} = f(y) := \lambda y(1 - y) \quad , \quad y(0) = 0.01 \quad .$$

- Explicit Euler method (12.2.4) with uniform timestep  $h = 1/N$ ,  $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$ .



$\lambda$  large: blow-up of  $y_k$  for large timestep  $h$



$\lambda = 90$ : —  $\hat{=}$   $y(t)$ , —\*  $\hat{=}$  Euler polygon

Explanation:  $y_k$  may miss the stationary point  $y = 1$  (overshooting).

This leads to a sequence  $(y_k)_k$  with exponentially increasing oscillations.

### Deeper analysis:

For  $y \approx 1$ :  $f(y) \approx \lambda(1 - y) \Rightarrow$  If  $y(t_0) \approx 1$ , then the solution of the IVP will behave like the solution of  $\dot{y} = \lambda(1 - y)$ , which is a **linear** ODE. Similarly,  $z(t) := 1 - y(t)$  will behave like the solution of the “decay equation”  $\dot{z} = -\lambda z$ .

Motivated by the considerations in Ex. 13.1.1 we study the explicit Euler method (12.2.4) for the

$$\text{linear model problem: } \dot{y} = \lambda y, \quad y(0) = y_0, \quad \text{with } \lambda \ll 0, \quad (13.1.2)$$

and *exponentially decaying* exact solution

$$y(t) = y_0 \exp(\lambda t) \rightarrow 0 \quad \text{for } t \rightarrow \infty.$$

Recursion of explicit Euler method for (13.1.2):

$$(12.2.4) \text{ for } f(y) = \lambda y: \quad y_{k+1} = y_k(1 + \lambda h) . \quad (13.1.3)$$

$$\blacktriangleright \quad y_k = y_0(1 + \lambda h)^k \Rightarrow |y_k| \rightarrow \begin{cases} 0 & , \text{ if } \lambda h > -2 \quad (\text{qualitatively correct}) , \\ \infty & , \text{ if } \lambda h < -2 \quad (\text{qualitatively wrong}) . \end{cases}$$

**Timestep constraint:** only if  $|\lambda|h < 2$  we obtain decaying solution by explicit Euler method!

Could it be that the timestep control is desperately trying to enforce the qualitatively correct behavior of the numerical solution in Ex. 13.1.1? Let us examine how the simple stepsize control of Code 12.5.6 fares for model problem (13.1.2):

R. Hiptmair  
rev 38286,  
December  
23, 2010

*Example 13.1.4* (Simple adaptive timestepping for fast decay).

- “Linear model problem IVP”:  $\dot{y} = \lambda y, y(0) = 1, \lambda = -100$
- Simple adaptive timestepping method as in Ex. 12.5.8, see Code 12.5.6

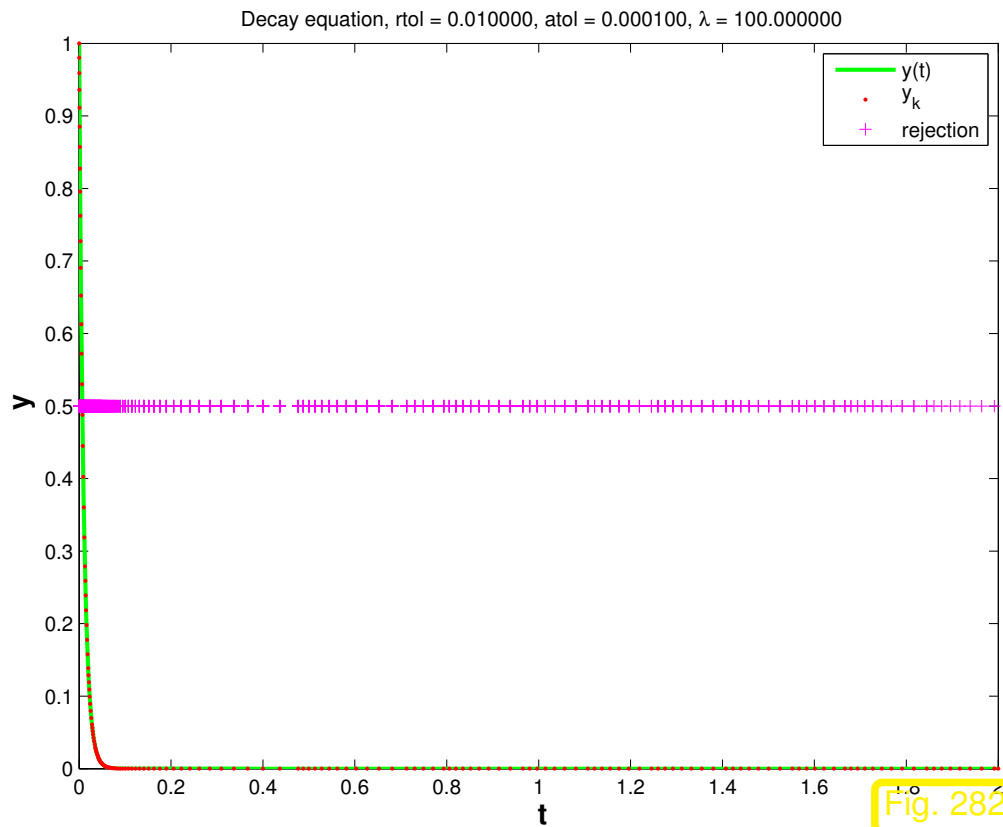


Fig. 282

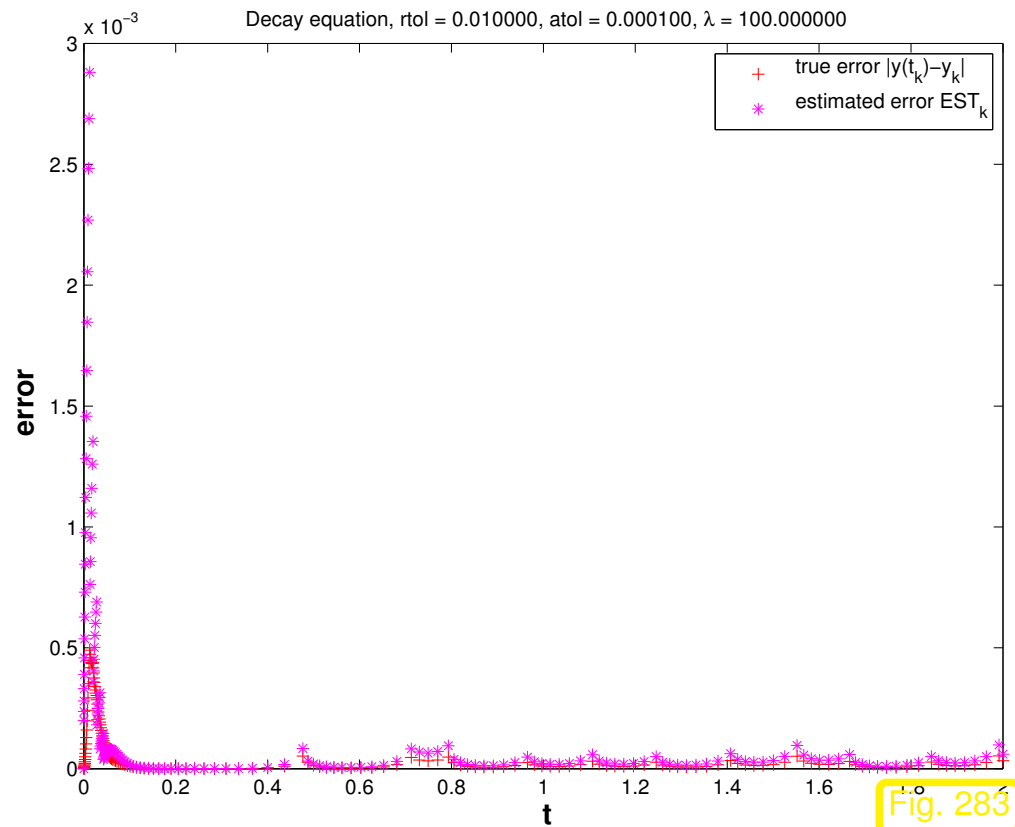


Fig. 283

Observation: in fact, stepsize control enforces small timesteps even if  $y(t) \approx 0$  and persistently triggers rejections of timesteps. This is necessary to prevent overshooting in the Euler method, which contributes to the estimate of the one-step error.





Is this a particular “flaw” of the explicit Euler method? Let us study the behavior of another simple explicit Runge-Kutta method applied to the linear model problem.

*Example 13.1.5* (Explicit trapezoidal rule for decay equation).  $\rightarrow$  [13, Ex. 11.29]

Recall recursion for explicit trapezoidal rule:

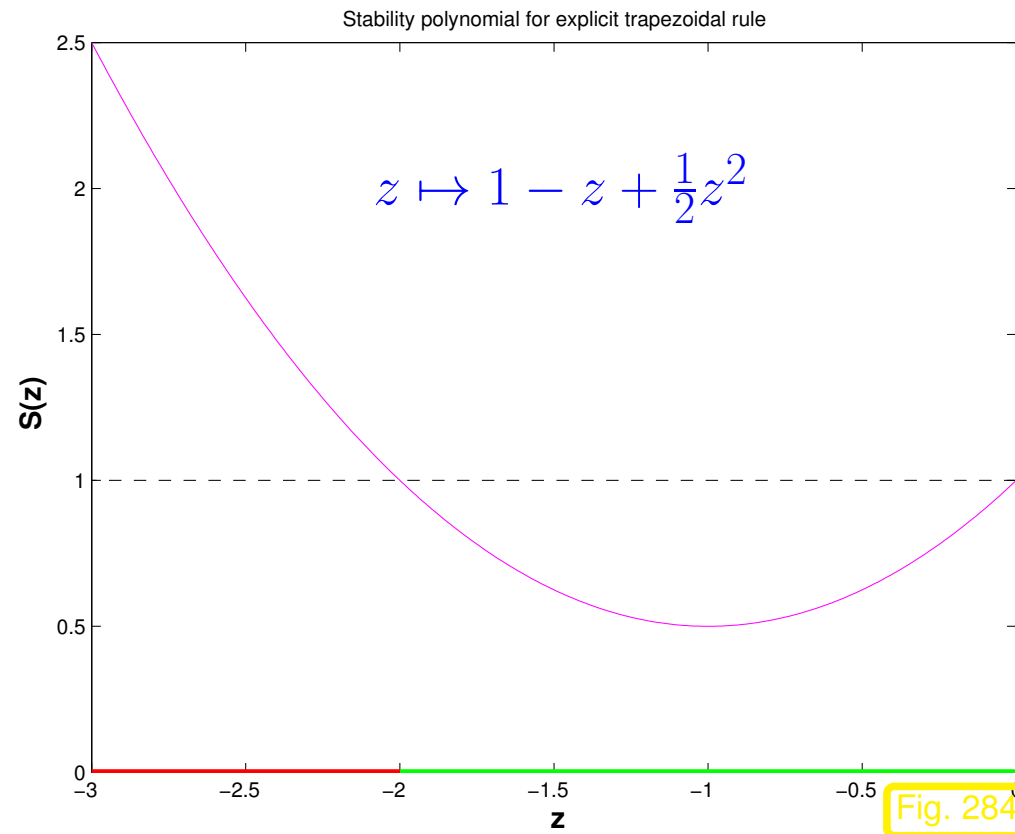
$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0), \quad \mathbf{k}_2 = \mathbf{f}(t_0 + h, \mathbf{y}_0 + h\mathbf{k}_1), \quad \mathbf{y}_1 = \mathbf{y}_0 + \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2). \quad (12.4.5)$$

Apply this to the model problem (13.1.2), that is  $\mathbf{f}(y) = f(y) = \lambda y$ ,  $\lambda < 0$ :

$$\blacktriangleright \quad k_1 = \lambda y_0, \quad k_2 = \lambda(y_0 + hk_1) \quad \Rightarrow \quad y_1 = \underbrace{\left(1 + \lambda h + \frac{1}{2}(\lambda h)^2\right)}_{=:S(h\lambda)} y_0. \quad (13.1.6)$$

$\blacktriangleright$  sequence generated by explicit trapezoidal rule:

$$y_k = S(h\lambda)^k y_0, \quad k = 0, \dots, N. \quad (13.1.7)$$



$$|S(h\lambda)| < 1 \iff -2 < h\lambda < 0.$$

Qualitatively correct decay behavior of  $(y_k)_k$  only under **timestep constraint**

$$h \leq |2/\lambda|. \quad (13.1.8)$$

◇ R. Hiptmair  
rev 38286,  
December  
23, 2010

**Model problem analysis** for general explicit Runge-Kutta method ( $\rightarrow$  Def. 12.4.8): apply Runge-Kutta method encoded by Butcher scheme  $\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$  to (13.1.2)

$$k_i = \lambda \left( y_0 + h \sum_{j=1}^{i-1} a_{ij} k_j \right), \quad \Rightarrow \quad \begin{pmatrix} \mathbf{I} - z\mathfrak{A} & 0 \\ -z\mathbf{b}^T & 1 \end{pmatrix} \begin{pmatrix} \mathbf{k} \\ y_1 \end{pmatrix} = y_0 \begin{pmatrix} \mathbf{1} \\ 1 \end{pmatrix}, \quad (13.1.9)$$

where  $\mathbf{k} \in \mathbb{R}^s \hat{=} (k_1, \dots, k_s)^T / \lambda$  of increments, and  $z := \lambda h$ .

$$\blacktriangleright y_1 = S(z)y_0 \quad \text{with} \quad S(z) := 1 + z\mathbf{b}^T (\mathbf{I} - z\mathfrak{A})^{-1} \mathbf{1} = \det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T). \quad (13.1.10)$$

The first formula for  $S(z)$  immediately follows from (13.1.9), the second is a consequence of Cramer's rule.

Thus we have proved the following theorem.

**Theorem 13.1.11** (Stability function of explicit Runge-Kutta methods).  $\rightarrow [35, \text{Thm. 77.2}], [51, \text{Sect. 11.8.4}]$

The discrete evolution  $\Psi_\lambda^h$  of an explicit  $s$ -stage Runge-Kutta single step method ( $\rightarrow$  Def. 12.4.8) with Butcher scheme  $\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$  (see (12.4.9)) for the ODE  $\dot{y} = \lambda y$  is a multiplication operator according to

$$\Psi_\lambda^h = \underbrace{1 + z\mathbf{b}^T (\mathbf{I} - z\mathfrak{A})^{-1} \mathbf{1}}_{\text{stability function } S(z)} = \det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T), \quad z := \lambda h, \quad \mathbf{1} = (1, \dots, 1)^T \in \mathbb{R}^s.$$

Thm. 13.1.11  $\Rightarrow S \in \mathcal{P}_s$  (polynomial stability function)

Remember from Ex. 13.1.5: for sequence  $(|y_k|)_{k=0}^{\infty}$  produced by explicit Runge-Kutta method applied to IVP (13.1.2) holds  $y_k = S(\lambda h)^k y_0$ .



$$(|y_k|)_{k=0}^{\infty} \text{ non-increasing} \Leftrightarrow |S(\lambda h)| \leq 1, \quad (13.1.12)$$

$$(|y_k|)_{k=0}^{\infty} \text{ exponentially increasing} \Leftrightarrow |S(\lambda h)| > 1.$$

On the other hand:

$$\forall S \in \mathcal{P}_s: \lim_{|z| \rightarrow \infty} |S(z)| = \infty$$

► **timestep constraint:** In order to avoid exponentially increasing (qualitatively wrong for  $\lambda < 0$ ) sequences  $(y_k)_{k=0}^{\infty}$  we must have  $|\lambda h|$  *sufficiently small*.

Small timesteps may have to be used for stability reasons,  
though accuracy may not require them!



Inefficient numerical integration

*Remark* 13.1.13 (Stepsize control detects instability).

Always look at the bright side of life:

Ex. 13.0.1, 13.1.4: Stepsize control guarantees acceptable solutions, with a hefty price tag however.



## 13.2 Stiff problems [51, Sect. 11.10]

Objection: The IVP (13.1.2) may be an oddity rather than a model problem: the weakness of explicit Runge-Kutta methods discussed in the previous section may be just a peculiar response to an unusual situation.

This section will reveal that the behavior observed in Ex. 13.0.1 and Ex. 13.1.1 is typical for a large class of problems and that the model problem (13.1.2) really represents a “generic case”.

*Example 13.2.1* (Transient simulation of RLC-circuit).

Circuit from Ex. 12.1.8



$$\ddot{u} + \alpha\dot{u} + \beta u = g(t),$$

$$\alpha := (RC)^{-1}, \beta = (LC)^{-1}, g(t) = \alpha\dot{U}_s.$$

Transformation to linear 1st-order ODE, see

Rem. 12.1.15,  $v := \dot{u}$

$$\underbrace{\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix}}_{=:\dot{\mathbf{y}}} = \underbrace{\begin{pmatrix} 0 & 1 \\ -\beta & -\alpha \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} - \begin{pmatrix} 0 \\ g(t) \end{pmatrix}}_{=:\mathbf{f}(t,\mathbf{y})}.$$

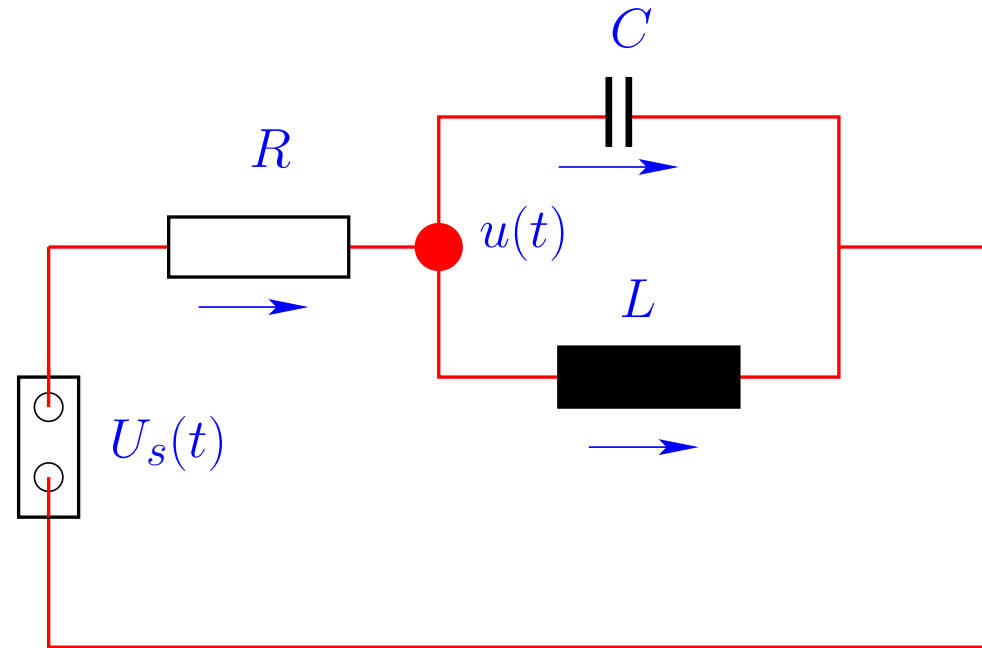
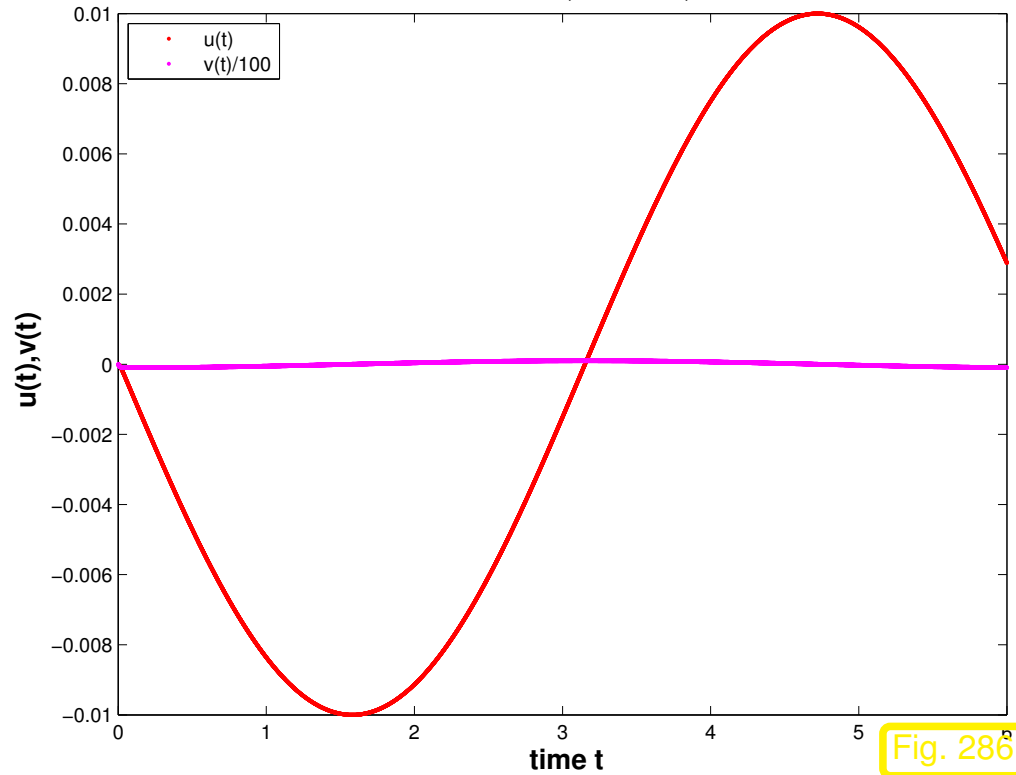


Fig. 285

RCL-circuit: R=100.000000, L=1.000000, C=0.000001



$R = 100\Omega$ ,  $L = 1\text{H}$ ,  $C = 1\mu\text{F}$ ,  $U_s(t) = 1\text{V} \sin(t)$ ,  
 $u(0) = v(0) = 0$  (“switch on”)

ode45 statistics:

17897 successful steps

1090 failed attempts

113923 function evaluations

R. Hiptmair

rev 38286,  
December  
23, 2010

Maybe the time-dependent right hand side due to the time-harmonic excitation severely affects ode45? Let us try a constant exciting voltage:

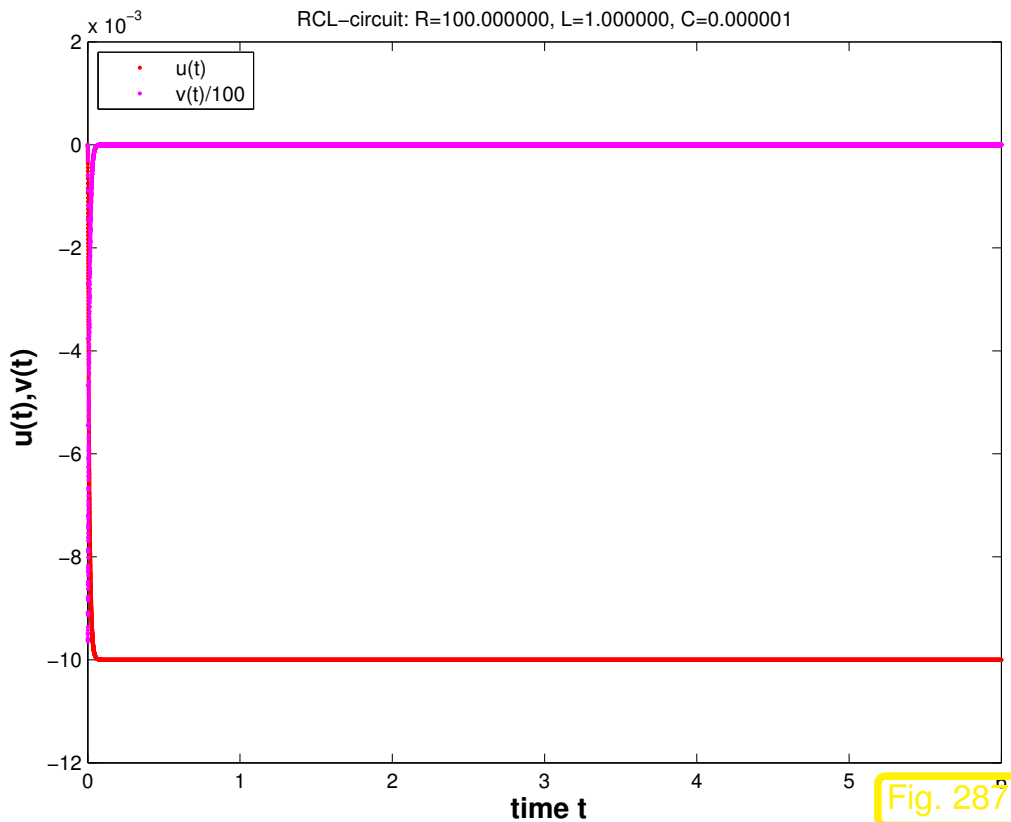


Fig. 287

$R = 100\Omega$ ,  $L = 1\text{H}$ ,  $C = 1\mu\text{F}$ ,  $U_s(t) = 1\text{V}$ ,  
 $u(0) = v(0) = 0$  (“switch on”)

ode45 statistics:

17901 successful steps

1210 failed attempts

114667 function evaluations

### Code 13.2.2: simulation of linear RLC circuit using ode45

```

1 function stiffcircuit(R,L,C,Us,tspan,filename)
2 % Transient simulation of simple linear circuit of Ex. refex:stiffcircuit
3 % R,L,C: paramters for circuits elements (compatible units required)
4 % Us: exciting time-dependent voltage  $U_s = U_s(t)$ , function handle
5 % zero initial values
6
7 % Coefficient for 2nd-order ODE  $\ddot{u} + \alpha\dot{u} + \beta = g(t)$ 

```



```
8 alpha = 1/(R*C); beta = 1/(C*L);
9 % Conversion to 1st-order ODE  $y = My + \begin{pmatrix} 0 \\ g(t) \end{pmatrix}$ . Set up right hand side function.
10 M = [0 , 1; -beta , -alpha]; rhs = @(t,y) (M*y - [ 0 ;
    alpha*Us(t)]);
11 % Set tolerances for MATLAB integrator, see Rem. 12.5.20
12 options = odeset('reltol',0.1,'abstol',0.001,'stats','on');
13 y0 = [0;0]; [t,y] = ode45(rhs,tspan,y0,options);
14
15 % Plot the solution components
16 figure ('name','Transient circuit simulation');
17 plot (t,y(:,1),'r.',t,y(:,2)/100,'m. ');
18 xlabel ('{\bf time t}','fontsize',14);
19 ylabel ('{\bf u(t),v(t)}','fontsize',14);
20 title (sprintf ('RCL-circuit: R=%f, L=%f, C=%f',R,L,C));
21 legend ('u(t)', 'v(t)/100', 'location','northwest');
22
23 print ('-depsc2', sprintf ('../PICTURES/%s.eps',filename));
```

Observation: stepsize control of `ode45` ( $\rightarrow$  Sect. 12.5) enforces extremely small timesteps though solution almost constant except at  $t = 0$ .



Motivated by Ex. 13.2.1 we examine linear homogeneous IVP of the form

$$\dot{\mathbf{y}} = \underbrace{\begin{pmatrix} 0 & 1 \\ -\beta & -\alpha \end{pmatrix}}_{=: \mathbf{M}} \mathbf{y} \quad , \quad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{R}^2 . \quad (13.2.3)$$

In Ex .13.2.1:  $\beta \gg \frac{1}{4}\alpha^2 \gg 1$ .

[63, Sect. 5.6]: general solution of  $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$ ,  $\mathbf{M} \in \mathbb{R}^{2,2}$ , by diagonalization of  $\mathbf{M}$  (if possible):

$$\mathbf{M}\mathbf{V} = \mathbf{M}(\mathbf{v}_1, \mathbf{v}_2) = (\mathbf{v}_1, \mathbf{v}_2) \begin{pmatrix} \lambda_1 & \\ & \lambda_2 \end{pmatrix} . \quad (13.2.4)$$

►  $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^2 \setminus \{0\} \hat{=}$  eigenvectors of  $\mathbf{M}$ ,  $\lambda_1, \lambda_2 \hat{=}$  eigenvalues of  $\mathbf{M}$ , see Def. 6.1.1.

Idea: transform  $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$  into *decoupled* scalar linear ODEs!

$$\dot{\mathbf{y}} = \mathbf{M}\mathbf{y} \quad \Leftrightarrow \quad \mathbf{V}^{-1}\dot{\mathbf{y}} = \mathbf{V}^{-1}\mathbf{M}\mathbf{V}(\mathbf{V}^{-1}\mathbf{y}) \quad \begin{matrix} \mathbf{z}(t) := \mathbf{V}^{-1}\mathbf{y}(t) \\ \Leftrightarrow \end{matrix} \quad \dot{\mathbf{z}} = \begin{pmatrix} \lambda_1 & \\ & \lambda_2 \end{pmatrix} \mathbf{z} . \quad (13.2.5)$$

This yields the general solution of the ODE  $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$

$$\mathbf{y}(t) = A\mathbf{v}_1 \exp(\lambda_1 t) + B\mathbf{v}_2 \exp(\lambda_2 t), \quad A, B \in \mathbb{R}. \quad (13.2.6)$$

Note:  $t \mapsto \exp(\lambda_i t)$  is general solution of the ODE  $\dot{z}_i = \lambda_i z_i$ .

Consider discrete evolution of explicit Euler method (12.2.4) for ODE  $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$

$$\Psi^h \mathbf{y} = \mathbf{y} + h\mathbf{M}\mathbf{y} \quad \Leftrightarrow \quad \mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{M}\mathbf{y}_k.$$

Perform the same transformation as above on the discrete evolution:

$$\mathbf{V}^{-1}\mathbf{y}_{k+1} = \mathbf{V}^{-1}\mathbf{y}_k + h\mathbf{V}^{-1}\mathbf{M}\mathbf{V}(\mathbf{V}^{-1}\mathbf{y}_k) \quad \mathbf{z}_k := \mathbf{V}^{-1}\mathbf{y}_k \quad \Leftrightarrow \quad \underbrace{(\mathbf{z}_{k+1})_i = (\mathbf{z}_k)_i + h\lambda_i (\mathbf{z}_k)_i}_{\hat{=} \text{explicit Euler step for } \dot{z}_i = \lambda_i z_i}. \quad (13.2.7)$$

Crucial insight:

The explicit Euler method generates uniformly bounded solution sequences  $(\mathbf{y}_k)_{k=0}^{\infty}$  for  $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$  with diagonalizable matrix  $\mathbf{M} \in \mathbb{R}^{d,d}$  with eigenvalues  $\lambda_1, \dots, \lambda_d$ , **if and only if** it generates uniformly bounded sequences for **all** the scalar ODEs  $\dot{z} = \lambda_i z, i = 1, \dots, d$ .

An analogous statement is true for all Runge-Kutta methods!

(This is revealed by simple algebraic manipulations of the increment equations.)

So far we conducted the model problem analysis under the premises  $\lambda < 0$ .

However: in Ex. 13.2.1 we have  $\lambda_{1/2} = -\frac{1}{2}\alpha \pm i\sqrt{\beta - \frac{1}{4}\alpha^2}$  (complex eigenvalues!). How will explicit Euler/explicit RK-methods respond to them?

*Example 13.2.8* (Explicit Euler method for damped oscillations).

Consider linear model IVP (13.1.2) for  $\lambda \in \mathbb{C}$ :

$\operatorname{Re} \lambda < 0 \Rightarrow$  exponentially decaying solution  $y(t) = y_0 \exp(\lambda t)$ ,

because  $|\exp(\lambda t)| = \exp(\operatorname{Re} \lambda t)$ .

**Model problem analysis** ( $\rightarrow$  Ex. 13.1.1, Ex. 13.1.5) for explicit Euler method and  $\lambda \in \mathbb{C}$ :

Sequence generated by explicit Euler method (12.2.4) for model problem (13.1.2):

$$y_{k+1} = y_k(1 + h\lambda) . \tag{13.1.3}$$

▶  $\lim_{k \rightarrow \infty} y_k = 0 \iff |1 + h\lambda| < 1 .$

timestep constraint to get decaying (discrete) solution !

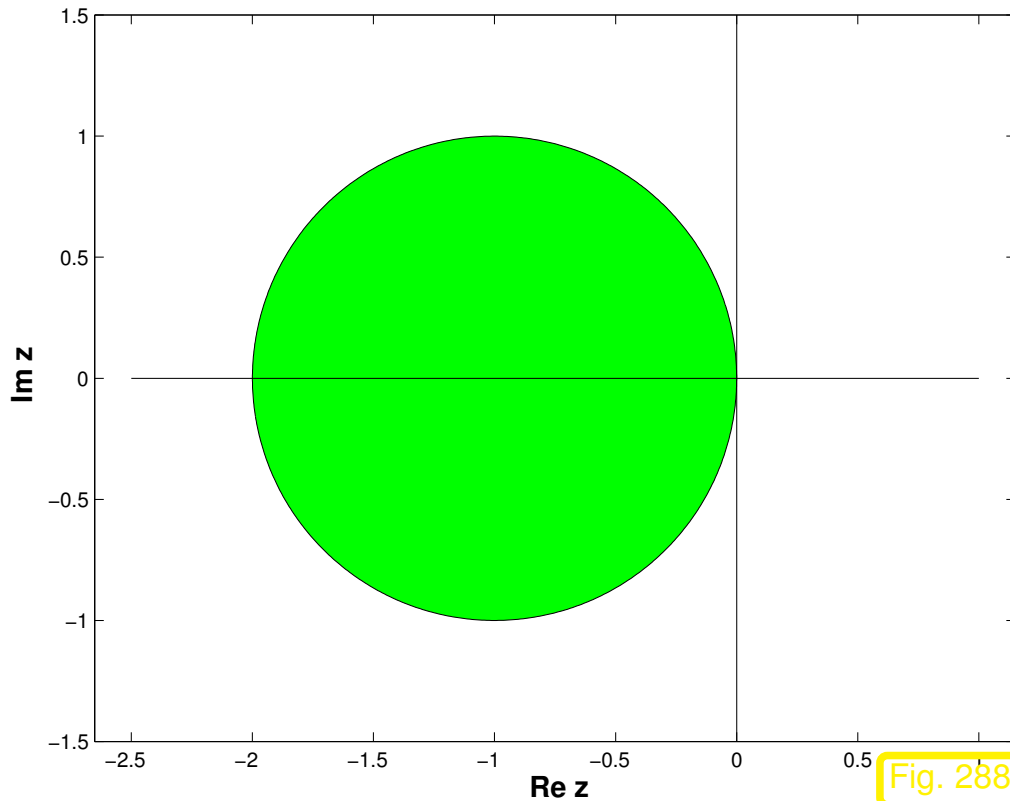


Fig. 288

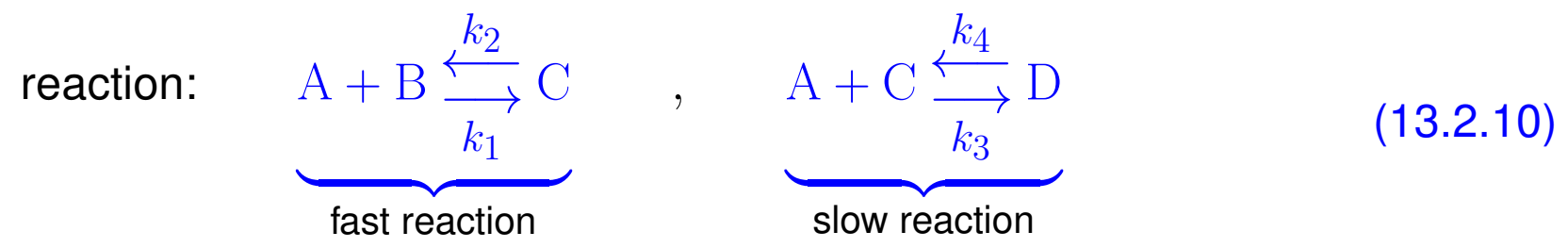
◁  $\{z \in \mathbb{C} : |1 + z| < 1\}$

Now we can conjecture what happens in Ex. 13.2.1: the eigenvalues  $\lambda_{1/2} = -\frac{1}{2}\alpha \pm i\sqrt{\beta - \frac{1}{4}\alpha^2}$  of  $\mathbf{M}$  have a very large (in modulus) negative real part. Since `ode45` can be expected to behave as if it integrates  $\dot{z} = \lambda_2 z$ , it faces a severe timestep constraint, if exponential blow-up is to be avoided, see Ex. 13.1.1. Thus stepsize control must resort to tiny timesteps.



## Can we predict this kind of difficulty ?

*Example* 13.2.9 (Kinetics of chemical reactions). → [35, Ch. 62]



Vastly different reaction constants:

$$k_1, k_2 \gg k_3, k_4$$



If  $c_A(0) > c_B(0)$  ➤ 2nd reaction determines overall long-term reaction dynamics

Mathematical model: ODE involving **concentrations**  $\mathbf{y}(t) = (c_A(t), c_B(t), c_C(t), c_D(t))^T$

$$\dot{\mathbf{y}} := \frac{d}{dt} \begin{pmatrix} c_A \\ c_B \\ c_C \\ c_D \end{pmatrix} = \mathbf{f}(\mathbf{y}) := \begin{pmatrix} -k_1 c_A c_B + k_2 c_C - k_3 c_A c_C + k_4 c_D \\ -k_1 c_A c_B + k_2 c_C \\ k_1 c_A c_B - k_2 c_C - k_3 c_A c_C + k_4 c_D \\ k_3 c_A c_C - k_4 c_D \end{pmatrix}. \quad (13.2.11)$$

MATLAB computation:  $t_0 = 0$ ,  $T = 1$ ,  $k_1 = 10^4$ ,  $k_2 = 10^3$ ,  $k_3 = 10$ ,  $k_4 = 1$

### Code 13.2.12: Simulation of “stiff” chemical reaction

```

1 function chemstiff
2 % Simulation of kinetics of coupled chemical reactions with vastly different
  reaction
3 % rates, see (13.2.11) for the ODE model.
4 % reaction rates  $k_1, k_2, k_3, k_4$ ,  $k_1, k_2 \gg k_3, k_4$ .
5 k1 = 1E4; k2 = 1E3; k3 = 10; k4 = 1;
6 % definition of right hand side function for ODE solver
7 fun = @(t,y) ([-k1*y(1)*y(2) + k2*y(3) - k3*y(1)*y(3) + k4*y(4);
8               -k1*y(1)*y(2) + k2*y(3);
9               k1*y(1)*y(2) - k2*y(3) - k3*y(1)*y(3) + k4*y(4);
0               k3*y(1)*y(3) - k4*y(4)]);
1 tspan = [0 1]; % Integration time interval
2 L = tspan(2)-tspan(1); % Duration of simulation
3 y0 = [1;1;10;0]; % Initial value  $\mathbf{y}_0$ 
4

```

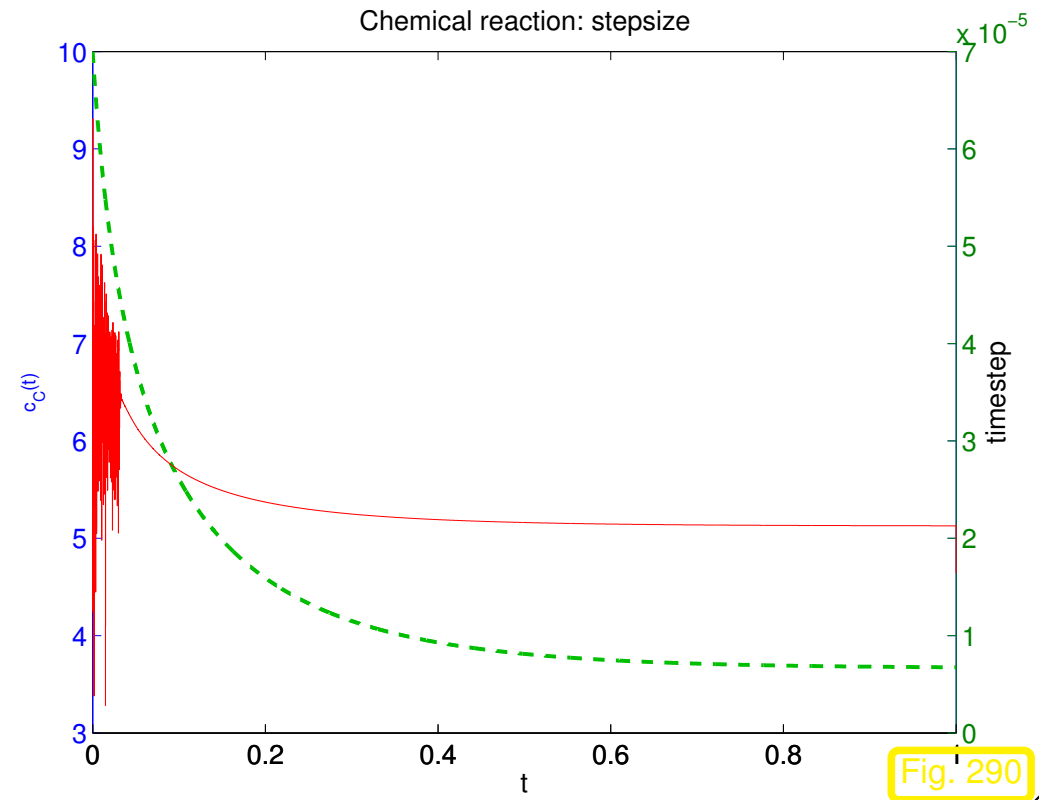
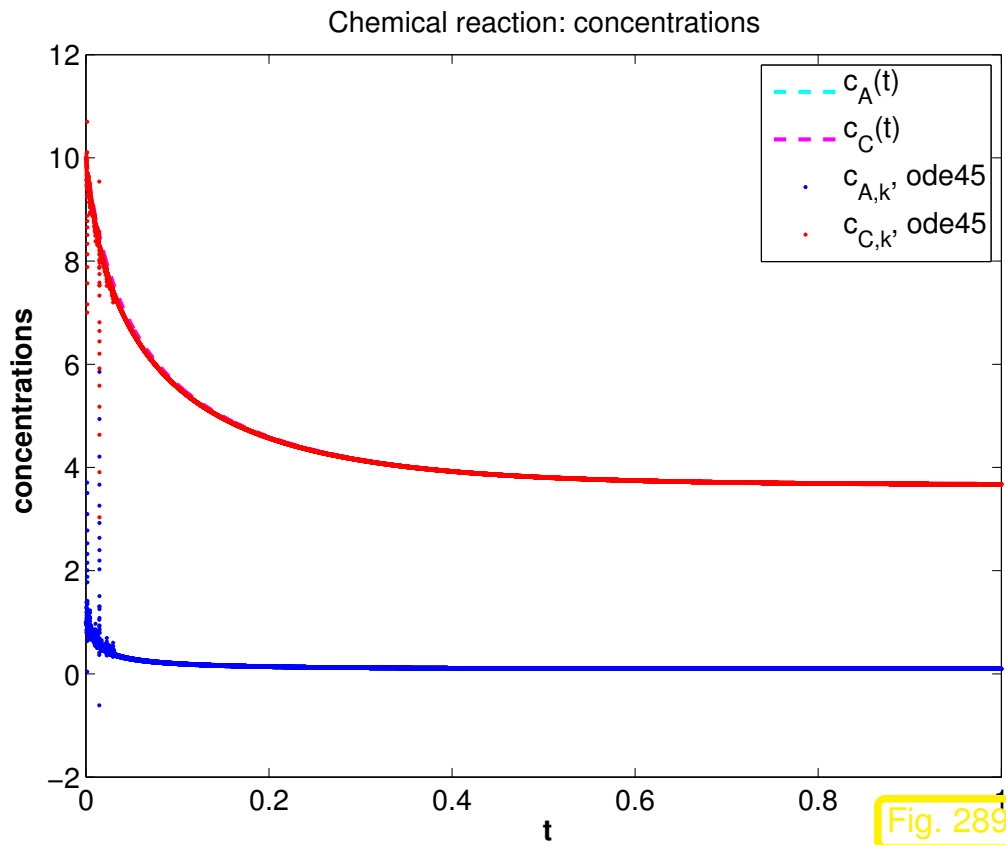
```
5 % compute "exact" solution, using ode113 with tight error tolerances
6 options = odeset('reltol',10*eps,'abstol',eps,'stats','on');
7 % get the 'exact' solution using ode113
8 [tex,yex] = ode113(fun,[0 1],y0,options);
9 % plot 'exact' results
10 figure; h = plot(tex,yex(:,1),'c--',tex,yex(:,3),'m--');
11 set(h(1),'linewidth',2); set(h(2),'linewidth',2);
12 leg{1} = 'c_A(t)'; leg{2} = 'c_C(t)'; hold on;
13
14 % Compute solution with ode45 and moderate tolerances
15 options = odeset('reltol',0.1,'abstol',0.001,'stats','on');
16 [t,y] = ode45(fun,[0 1],y0,options);
17 % plot the solution
18 plot(t,y(:,1),'b.',t,y(:,3),'r. ');
19 leg{3} = 'c_{A,k}, ode45'; leg{4} = 'c_{C,k}, ode45';
20 set(gca,'fontsize',14);
21 xlabel('\bf t');
22 ylabel('\bf concentrations');
23 title('Chemical reaction: concentrations');
24 legend(leg);
25 print -depsc2 '../PICTURES/chemstiff.eps';
26
27 % Plot stepsizes together with "exact" solution
28 figure('Name','Stepsizes');
29 [ax,h1,h2] = plotyy(tex,yex(:,3),0.5*(t(1:end-1)+t(2:end)),diff(t));
30 % set legend, labels, axes, ...
31 set(ax(1),'fontsize',14); set(ax(2),'fontsize',14);
32 set(ax(2),'xcolor',[0 0 0]);
33 xlabel('t');
```



```

14 set (h1,'linestyle','--','color',[0 0.75 0],'linewidth',2);
15 set (h2,'linestyle','-','color','r');
16 set (get(ax(1),'ylabel'),'string','c_C(t)');
17 set (get(ax(2),'ylabel'),'string','timestep','color',[0 0 0],'fontsize',14);
18 title ('Chemical reaction: stepsize');
19 print -depsc2 'chemstiffss.eps';

```



Example 13.2.13 (Strongly attractive limit cycle).

Autonomous ODE  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ 

$$\mathbf{f}(\mathbf{y}) := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \mathbf{y} + \lambda(1 - \|\mathbf{y}\|^2) \mathbf{y}, \quad (13.2.14)$$

on state space  $D = \mathbb{R}^2 \setminus \{0\}$ 

For  $\lambda = 0$ , the initial value problem  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ ,  $\mathbf{y}(0) = \begin{pmatrix} \cos \varphi \\ \sin \varphi \end{pmatrix}$ ,  $\varphi \in \mathbb{R}$  has the solution

$$\mathbf{y}(t) = \begin{pmatrix} \cos(t - \varphi) \\ \sin(t - \varphi) \end{pmatrix}, \quad t \in \mathbb{R}. \quad (13.2.15)$$

For this solution we have  $\|\mathbf{y}(t)\|_2 = 1$  for all times.

► (13.2.15) provides a solution even for  $\lambda \neq 0$ , if  $\|\mathbf{y}(0)\|_2 = 1$ , because in this case the term  $\lambda(1 - \|\mathbf{y}\|^2) \mathbf{y}$  will never become non-zero on the solution trajectory.

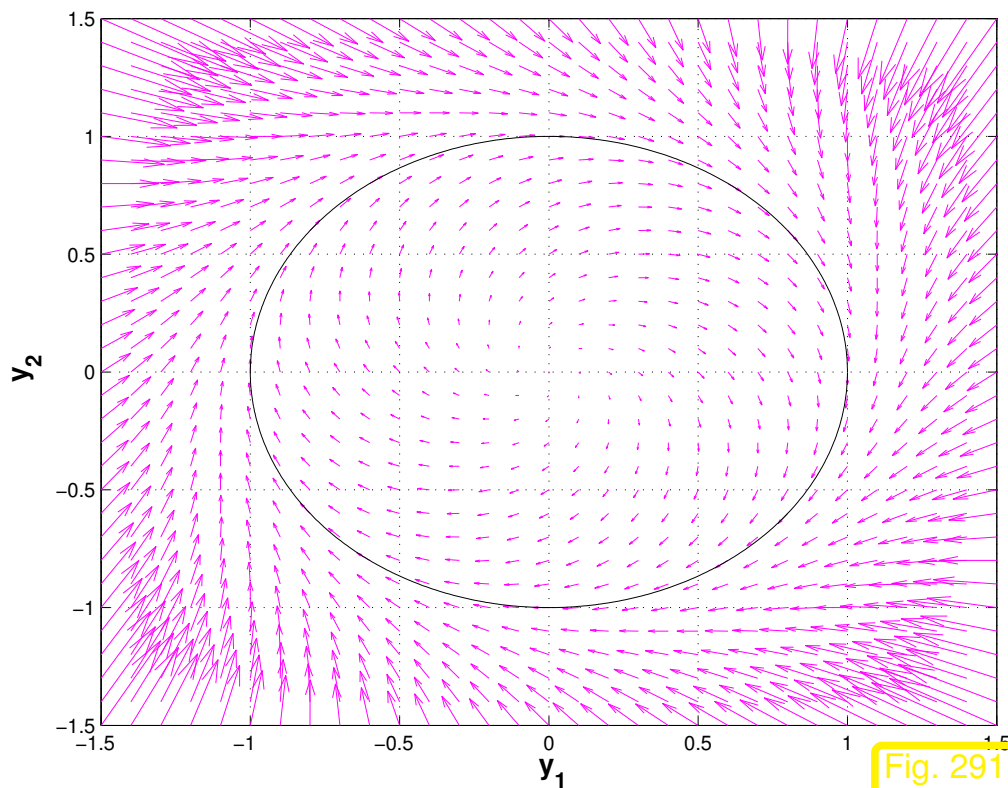


Fig. 291

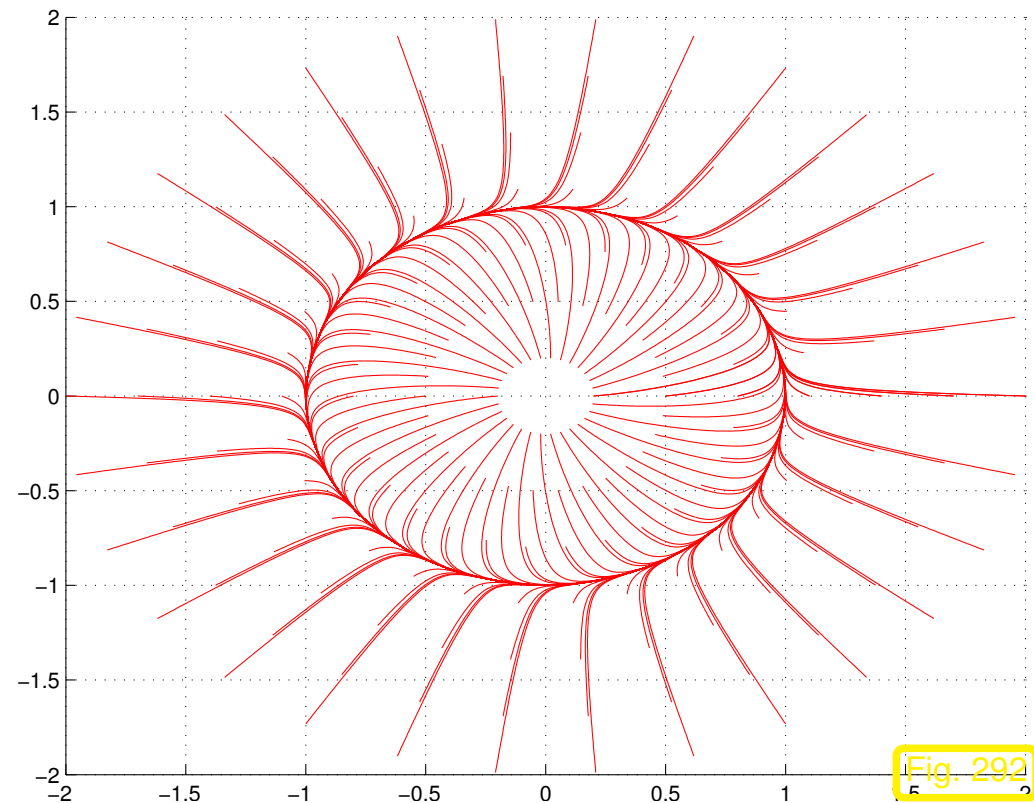
vectorfield  $\mathbf{f}$  ( $\lambda = 1$ )

Fig. 292

solution trajectories ( $\lambda = 10$ )Code 13.2.16: Application of `ode45` for limit cycle problem

```

1 % MATLAB script for solving limit cycle ODE (13.2.14)
2 % define right hand side vectorfield
3 fun = @(t,y) ([-y(2);y(1)] +
    lambda*(1-y(1)\symbol{94}2-y(2)\symbol{94}2)*y);
4 % standard invocation of MATLAB integrator, see Ex. 12.4.14
5 tspan = [0,2*pi]; y0 = [1,0];

```

```
6 opts = odeset('stats','on','reltol',1E-4,'abstol',1E-4);
7 [t45,y45] = ode45(fun,tspan,y0,opts);
```

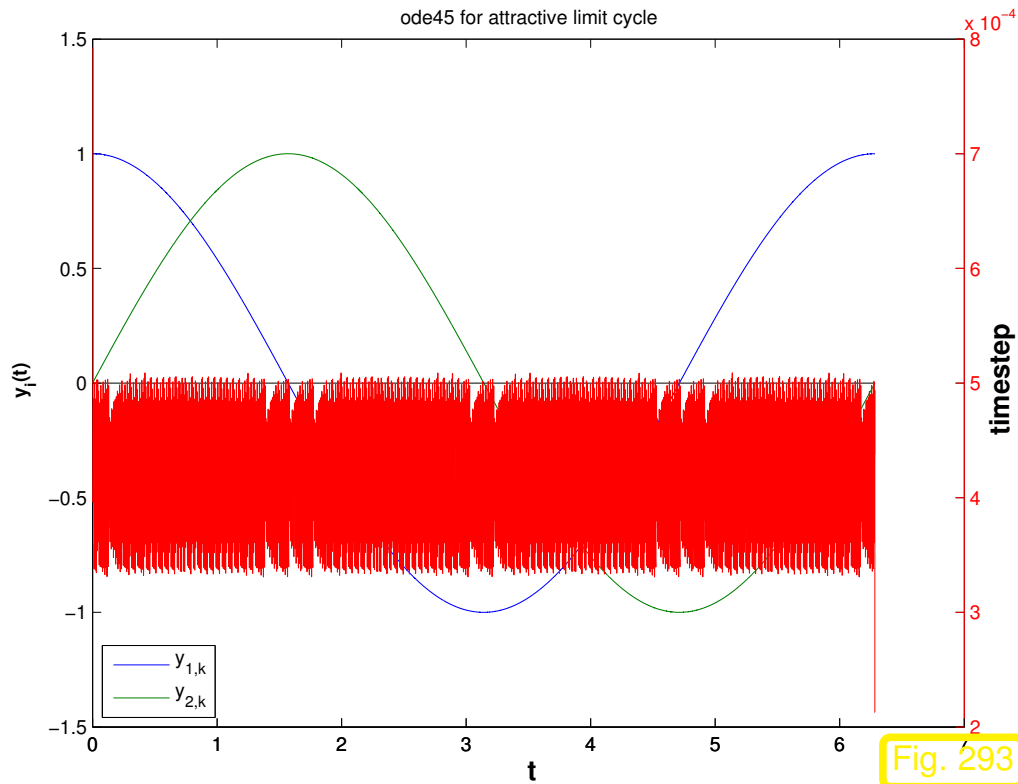
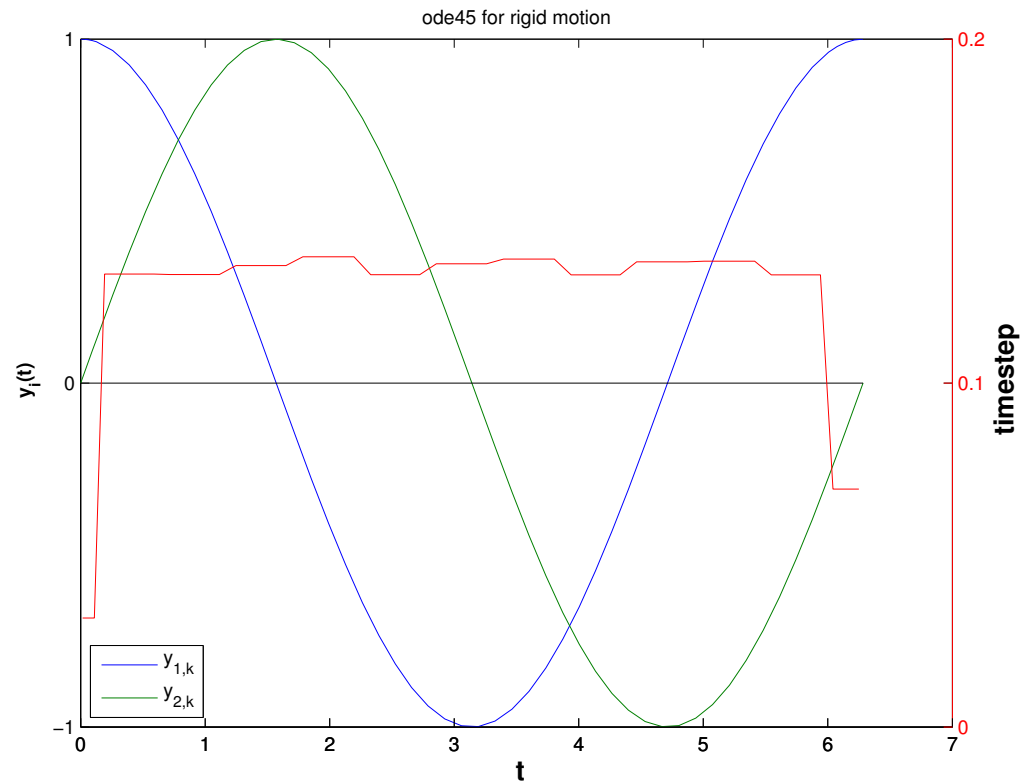


Fig. 293

many (3794) steps ( $\lambda = 1000$ )



accurate solution with few steps ( $\lambda = 0$ )

Confusing observation: we have  $\|y_0\| = 1$ , which implies  $\|y(t)\| = 1 \quad \forall t!$

Thus, the term of the right hand side, which is multiplied by  $\lambda$  will always vanish on the exact solution trajectory, which stays on the unit circle.

Nevertheless, `ode45` is forced to use tiny timesteps by the mere presence of this term.



**Notion 13.2.17** (Stiff IVP).

*An initial value problem is called **stiff**, if stability imposes much tighter timestep constraints on explicit single step methods than the accuracy requirements.*

Typical features of stiff IVPs:

- Presence of **fast transients** in the solution, see Ex. 13.1.1, 13.2.1,
- Occurrence of **strongly attractive** fixed points/limit cycles, see Ex. 13.2.13

*Example 13.3.1* (Implicit Euler timestepping for decay equation).

Again, **model problem analysis**: study implicit Euler method (12.2.8) for IVP (13.1.2)

$$\blacktriangleright \text{ sequence } y_k := \left( \frac{1}{1 - \lambda h} \right)^k y_0 . \quad (13.3.2)$$

$$\Rightarrow \boxed{\operatorname{Re} \lambda < 0 \Rightarrow \lim_{k \rightarrow \infty} y_k = 0 !} \quad (13.3.3)$$

**No** timestep constraint: qualitatively correct behavior of  $(y_k)_k$  for  $\operatorname{Re} \lambda < 0$  and **any**  $h > 0$ !



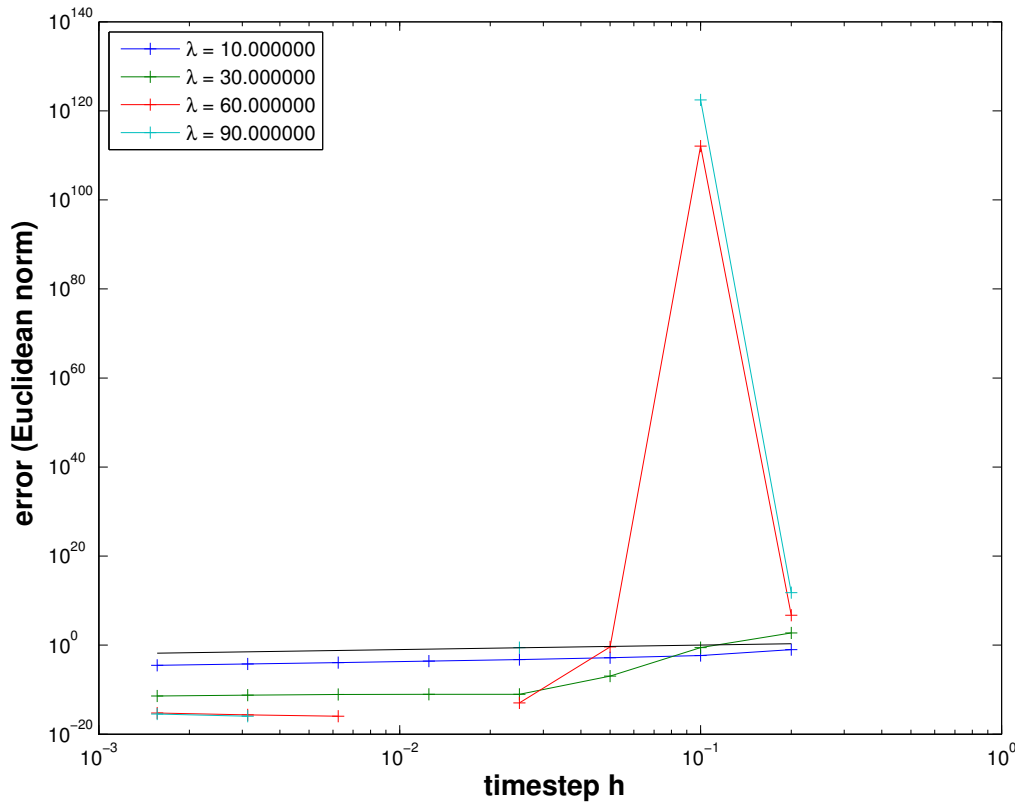
Observe: transformation idea, see (13.2.5), (13.2.7), applies to explicit *and implicit* Euler method alike.

Conjecture: implicit Euler method will not face timestep constraint for stiff problems ( $\rightarrow$  Notion 13.2.17).

*Example* 13.3.4 (Euler methods for stiff logistic IVP).

 Redo Ex. 13.1.1 for implicit Euler method:

### Explicit Euler method (12.2.4)



$\lambda$  large: blow-up of  $y_k$  for large timestep  $h$

### Implicit Euler method (12.2.8)

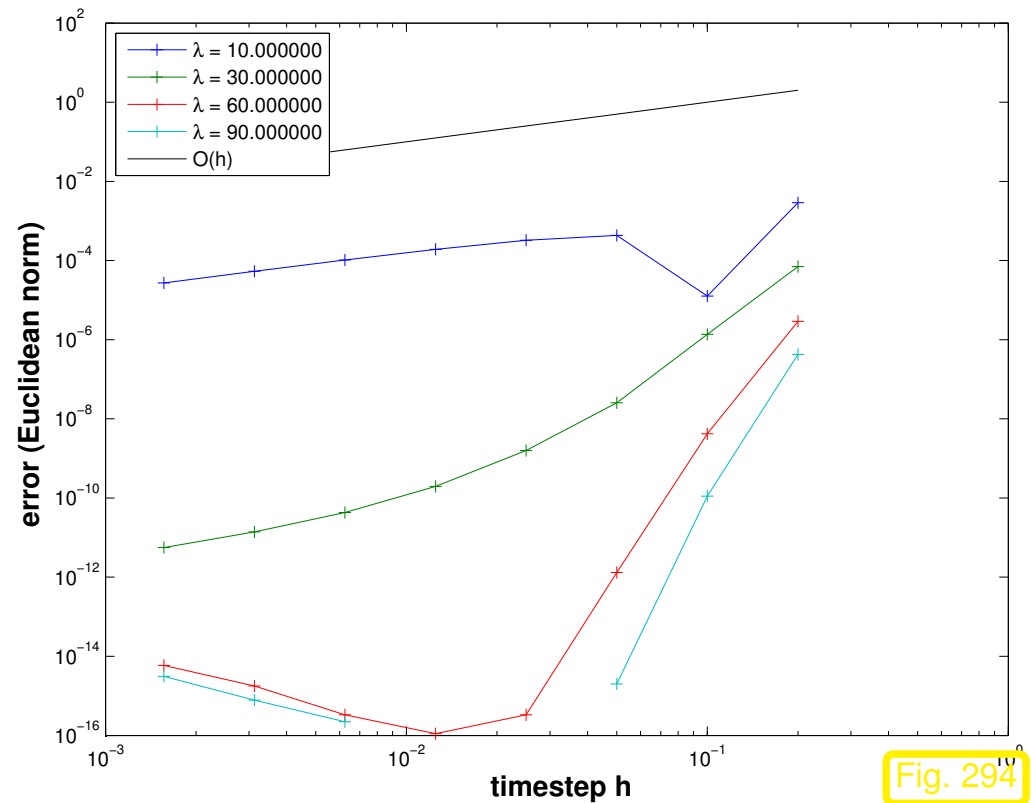


Fig. 294

$\lambda$  large: stable for all timesteps  $h$  !

Well, we see what we expected!





Unfortunately the implicit Euler method is of first order only, see Ex. 12.3.2. Can the Runge-Kutta design principle for integrators also yield higher order methods, which can cope with stiff problems?

**YES !**

**Definition 13.3.5** (General Runge-Kutta method). (cf. Def. 12.4.8)

For  $b_i, a_{ij} \in \mathbb{R}$ ,  $c_i := \sum_{j=1}^s a_{ij}$ ,  $i, j = 1, \dots, s$ ,  $s \in \mathbb{N}$ , an *s-stage Runge-Kutta single step method* (RK-SSM) for the IVP (12.1.13) is defined by

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h \sum_{j=1}^s a_{ij} \mathbf{k}_j), \quad i = 1, \dots, s, \quad \mathbf{y}_1 := \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i.$$

As before, the  $\mathbf{k}_i \in \mathbb{R}^d$  are called *increments*.

Note: computation of increments  $\mathbf{k}_i$  may now require the solution of (*non-linear*) systems of equations of size  $s \cdot d$  ( $\rightarrow$  “implicit” method)

## Shorthand notation for Runge-Kutta methods

## Butcher scheme



$$\frac{\mathbf{c} \mid \mathcal{A}}{\mathbf{b}^T} := \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} . \quad (13.3.6)$$

Note: now  $\mathcal{A}$  can be a general  $s \times s$ -matrix.

$\mathcal{A}$  strict lower triangular matrix

➤ explicit Runge-Kutta method, Def. 12.4.8

$\mathcal{A}$  lower triangular matrix

➤ diagonally-implicit Runge-Kutta method (DIRK)

**Model problem analysis** for general Runge-Kutta single step methods ( $\rightarrow$  Def. 13.3.5): exactly the same as for explicit RK-methods, see (13.1.9), (13.1.10)!

**Theorem 13.3.7** (Stability function of Runge-Kutta methods).

The discrete evolution  $\Psi_\lambda^h$  of an  $s$ -stage Runge-Kutta single step method ( $\rightarrow$  Def. 13.3.5) with Butcher scheme  $\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$  (see (13.3.6)) for the ODE  $\dot{y} = \lambda y$  is a multiplication operator according to

$$\Psi_\lambda^h = \underbrace{1 + z\mathbf{b}^T (\mathbf{I} - z\mathfrak{A})^{-1} \mathbf{1}}_{\text{stability function } S(z)} = \frac{\det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T)}{\det(\mathbf{I} - z\mathfrak{A})}, \quad z := \lambda h, \quad \mathbf{1} = (1, \dots, 1)^T \in \mathbb{R}^s.$$

Note: from the determinant representation of  $S(z)$  we infer that the stability function of an  $s$ -stage Runge-Kutta method is a **rational function** of the form  $S(z) = \frac{P(z)}{Q(z)}$  with  $P \in \mathcal{P}_s, Q \in \mathcal{P}_s$ .

Of course, such rational functions can satisfy  $|S(z)| < 1$  for all  $z < 0$ . For example, the stability function of the implicit Euler method (12.2.8) is

$$\frac{1}{1-z} \xrightarrow{\text{Thm. 13.3.7}} S(z) = \frac{1}{1-z}. \quad (13.3.8)$$

In light of the previous detailed analysis we can now state what we expect from the stability function of a Runge-Kutta method that is suitable for stiff IVP ( $\rightarrow$  Notion 13.2.17):

**Definition 13.3.9** (L-stable Runge-Kutta method).  $\rightarrow$  [35, Ch. 77]

A Runge-Kutta method ( $\rightarrow$  Def. 13.3.5) is *L-stable/asymptotically stable*, if its stability function ( $\rightarrow$  Def. 13.3.7) satisfies

$$(i) \quad \operatorname{Re} z < 0 \quad \Rightarrow \quad |S(z)| < 1, \quad (13.3.10)$$

$$(ii) \quad \lim_{\operatorname{Re} z \rightarrow -\infty} S(z) = 0. \quad (13.3.11)$$

R. Hiptmair

rev 38286,  
December  
23, 2010

*Remark 13.3.12* (Necessary condition for L-stability of Runge-Kutta methods).

Consider: Runge-Kutta method ( $\rightarrow$  Def. 13.3.5) with Butcher scheme  $\frac{\mathbf{c} \mid \mathfrak{A}}{\mathbf{b}^T}$

Assume:  $\mathfrak{A} \in \mathbb{R}^{s,s}$  is regular

For a rational function  $S(z) = \frac{P(z)}{Q(z)}$  the limit for  $|z| \rightarrow \infty$  exists and can easily be expressed by the leading coefficients of the polynomials  $P$  and  $Q$ :

Thm. 13.3.7  $\Rightarrow S(-\infty) = 1 - \mathbf{b}^T \mathfrak{A}^{-1} \mathbf{1}$ . (13.3.13)

► If  $\mathbf{b}^T = (\mathfrak{A})_{:,j}^T$  (row of  $\mathfrak{A}$ )  $\Rightarrow S(-\infty) = 0$ . (13.3.14)

Butcher scheme (13.3.6) for L-stable RK-methods, see Def. 13.3.9

► 
$$\frac{\mathbf{c} \mid \mathfrak{A}}{\mathbf{b}^T} := \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_{s-1} & a_{s-1,1} & \cdots & a_{s-1,s} \\ \hline 1 & b_1 & \cdots & b_s \\ \hline & b_1 & \cdots & b_s \end{array}$$

Example 13.3.15 (L-stable implicit Runge-Kutta methods).

$$\frac{1 \mid 1}{1}$$

Implicit Euler method

$$\frac{\frac{1}{3} \mid \frac{5}{12} \quad -\frac{1}{12}}{1 \mid \frac{3}{4} \quad \frac{1}{4}} \\ \hline \frac{3}{4} \quad \frac{1}{4}$$

Radau RK-SSM, order 3

$$\frac{\frac{4-\sqrt{6}}{10} \mid \frac{88-7\sqrt{6}}{360} \quad \frac{296-169\sqrt{6}}{1800} \quad \frac{-2+3\sqrt{6}}{225}}{\frac{4+\sqrt{6}}{10} \mid \frac{296+169\sqrt{6}}{1800} \quad \frac{88+7\sqrt{6}}{360} \quad \frac{-2-3\sqrt{6}}{225}}{1 \mid \frac{16-\sqrt{6}}{36} \quad \frac{16+\sqrt{6}}{36} \quad \frac{1}{9}} \\ \hline \frac{16-\sqrt{6}}{36} \quad \frac{16+\sqrt{6}}{36} \quad \frac{1}{9}}$$

Radau RK-SSM, order 5

Further information about Radau-Runge-Kutta single step methods can be found in [35, Ch. 79].



## 13.4 Semi-implicit Runge-Kutta methods [35, Ch. 80]



Equations fixing increments  $\mathbf{k}_i \in \mathbb{R}^d$ ,  $i = 1, \dots, s$ , for  $s$ -stage implicit RK-method  
 =  
 (Non-)linear system of equations with  $s \cdot d$  unknowns

R. Hiptmair  
rev 38286,  
December  
23, 2010

On the other hand, we are computing approximate solutions anyway, and the increments *are weighted with stepsize*  $h \ll 1$ , see Def. 13.3.5. So there is no point in determining them with high accuracy.

➤ use a fixed *small* number of Newton steps to solve for  $\mathbf{k}_i$ ,  $i = 1, \dots, s$ .

Extreme case: use only a single Newton step! Let's try.

Example 13.4.1 (Linearization of increment equations).

- Initial value problem for logistic ODE, see Ex. 12.1.1

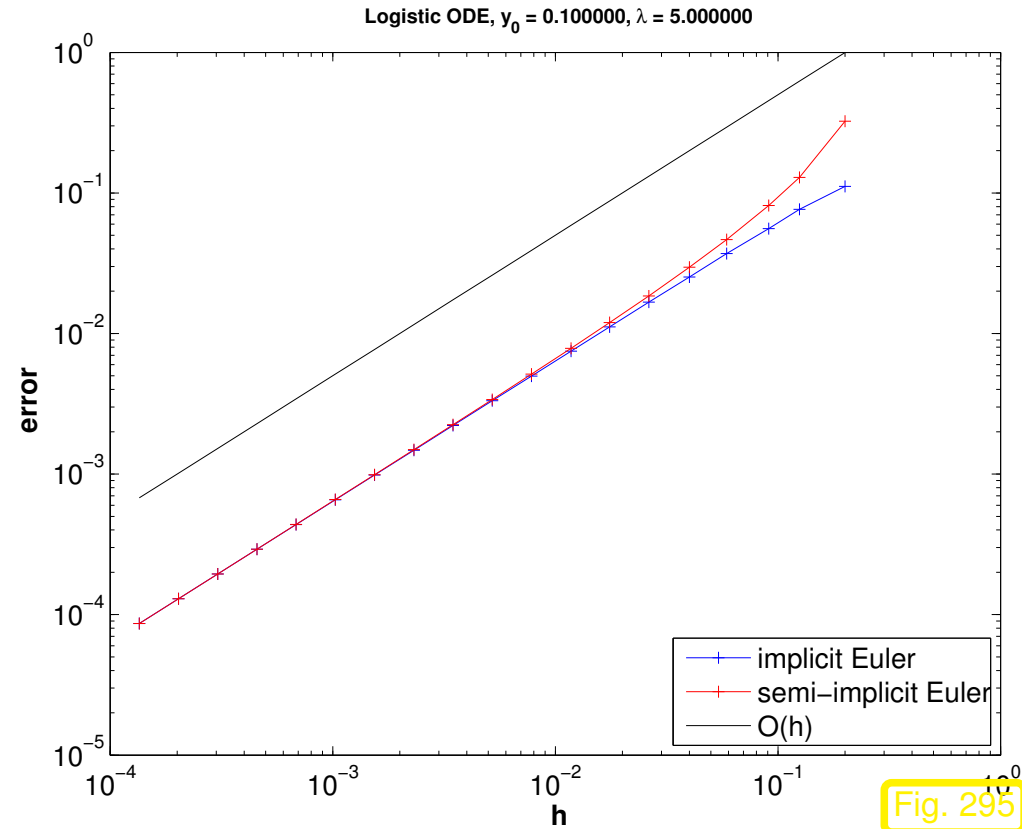
$$\dot{y} = \lambda y(1 - y) \quad , \quad y(0) = 0.1 \quad , \quad \lambda = 5 .$$

- Implicit Euler method (12.2.8) with uniform timestep  $h = 1/n$ ,  
 $n \in \{5, 8, 11, 17, 25, 38, 57, 85, 128, 192, 288, 432, 649, 973, 1460, 2189, 3284, 4926, 7389\}$ .

& approximate computation of  $y_{k+1}$  by  
**1 Newton step** with initial guess  $y_k$

= semi-implicit Euler method

- Measured error  $\text{err} = \max_{j=1, \dots, n} |y_j - y(t_j)|$



From (12.2.8) with timestep  $h > 0$

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}(\mathbf{y}_{k+1}) \quad \Leftrightarrow \quad F(\mathbf{y}_{k+1}) := \mathbf{y}_{k+1} - h\mathbf{f}(\mathbf{y}_{k+1}) - \mathbf{y}_k = 0 .$$

One Newton step applied to  $F(\mathbf{y}) = 0$  with initial guess  $\mathbf{y}_k$  yields

$$\mathbf{y}_{k+1} = \mathbf{y}_k - D\mathbf{f}(\mathbf{y}_k)^{-1}F(\mathbf{y}_k) = \mathbf{y}_k + (\mathbf{I} - hD\mathbf{f}(\mathbf{y}_k))^{-1}h\mathbf{f}(\mathbf{y}_k).$$

Note: for linear ODE with  $\mathbf{f}(\mathbf{y}) = \mathbf{A}\mathbf{y}$ ,  $\mathbf{A} \in \mathbb{R}^{d,d}$ , we recover the original implicit Euler method!

Observation: Approximate evaluation of defining equation for  $\mathbf{y}_{k+1}$  preserves 1st order convergence.

◇ R. Hiptmair  
rev 38286,  
December  
23, 2010

Idea: Use **linearized increment equations** for implicit RK-SSM

$$\mathbf{k}_i = \mathbf{f}(\mathbf{y}_0) + hD\mathbf{f}(\mathbf{y}_0) \left( \sum_{j=1}^s a_{ij} \mathbf{k}_j \right), \quad i = 1, \dots, s. \quad (13.4.2)$$

Linearization does nothing for linear ODEs ➤ stability function (→ Thm. 13.3.7) not affected!



► Class of **semi-implicit (linearly implicit) Runge-Kutta methods** (Rosenbrock-Wanner (ROW) methods):

$$(\mathbf{I} - ha_{ii}\mathbf{J})\mathbf{k}_i = \mathbf{f}(\mathbf{y}_0 + h \sum_{j=1}^{i-1} (a_{ij} + d_{ij})\mathbf{k}_j) - h\mathbf{J} \sum_{j=1}^{i-1} d_{ij}\mathbf{k}_j, \quad (13.4.3)$$

$$\mathbf{J} := D\mathbf{f}(\mathbf{y}_0 + h \sum_{j=1}^{i-1} (a_{ij} + d_{ij})\mathbf{k}_j), \quad (13.4.4)$$

$$\mathbf{y}_1 := \mathbf{y}_0 + \sum_{j=1}^s b_j \mathbf{k}_j. \quad (13.4.5)$$

*Remark* 13.4.6 (Adaptive integrator for stiff problems in MATLAB).

Handle of type  $@(t, y) \ J(t, y)$  to Jacobian  $D\mathbf{f} : I \times D \mapsto \mathbb{R}^{d,d}$

```
opts = odeset('abstol', atol, 'reltol', rtol, 'Jacobian', J)
[t, y] = ode23s(odefun, tspan, y0, opts);
```

Stepsize control according to policy of Sect. 12.5:

$\Psi \hat{=}$  RK-method of order 2

$\tilde{\Psi} \hat{=}$  RK-method of order 3

ode23s

integrator for **stiff** IVP



# 14

## Structure Preservation [31]

### 14.1 Dissipative Evolutions

### 14.2 Quadratic Invariants

### 14.3 Reversible Integrators

### 14.4 Symplectic Integrators

## Course 401-0674-00: Numerical Methods for Partial Differential Equations

Many fundamental models in science & engineering boil down to

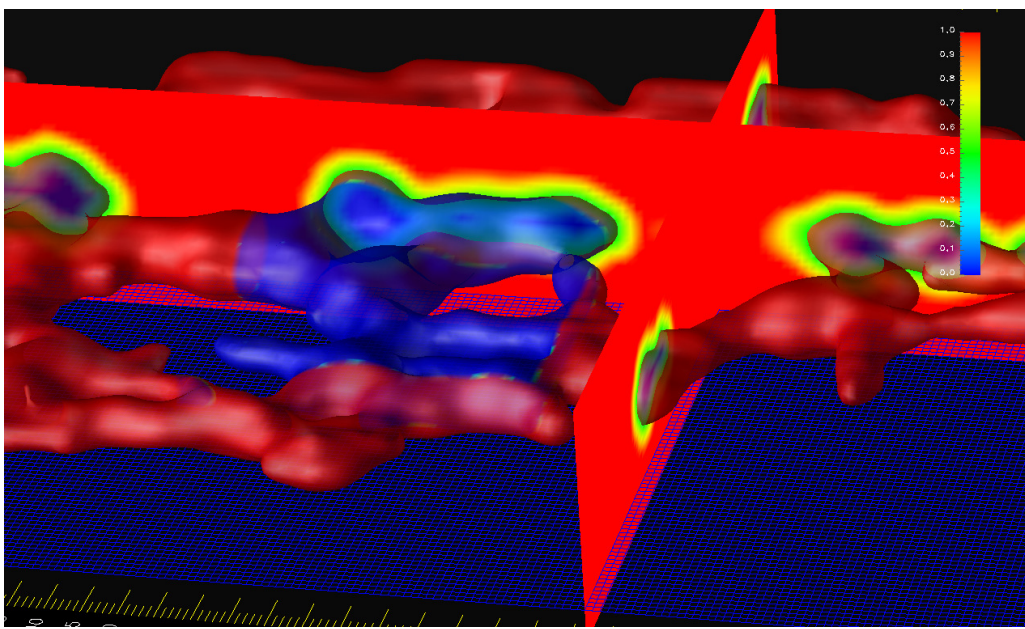
(initial) boundary value problems for **partial differential equations** (PDEs)

► Key role of numerical techniques for PDEs:

- Issue: Appropriate spatial (and temporal) **discretization** of PDE and boundary conditions
- Issue: **fast solution methods** for resulting *large* (non-)linear systems of equations

(initial) boundary value problems and techniques covered in the course:

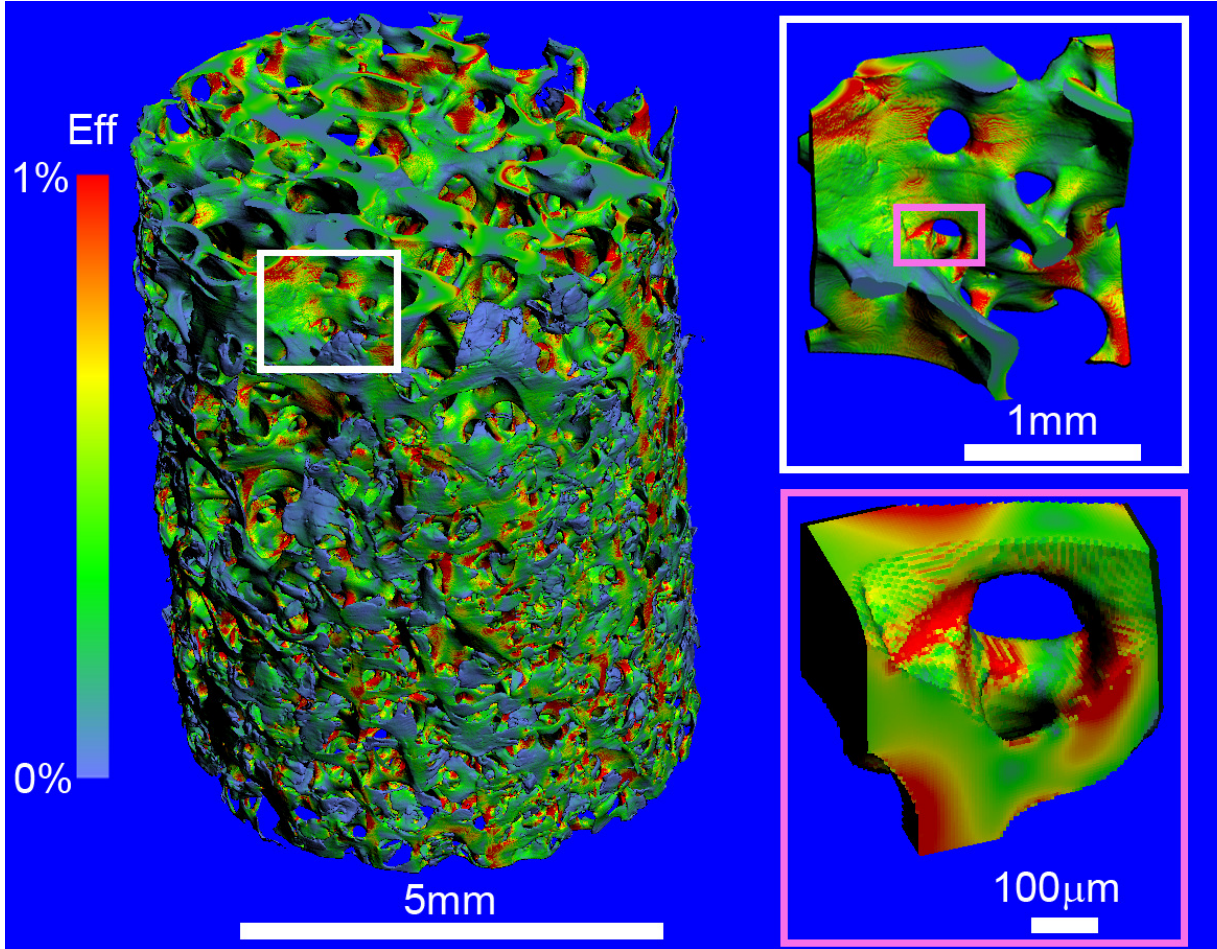
- 1 **Stationary 2nd-order scalar elliptic boundary value problems**



Diffusion boundary value problem:

$$\begin{aligned}
 -\operatorname{div}(\mathbf{A}(\mathbf{x}) \operatorname{grad} u(\mathbf{x})) &= f(\mathbf{x}) \quad \text{in } \Omega \subset \mathbb{R}^d, \\
 u &= g \quad \text{on } \partial\Omega.
 \end{aligned}$$

◁ diffusion on the surface (membrane) of the endoplasmic reticulum (I. Sbalzarini, D-INFK, ETH Zürich)



◁ Elastic deformation of human bone  
(P. Arbenz, D-INFK, ETH Zürich)

## ② Singularly perturbed elliptic boundary value problems

Stationary pollutant transport in water: find concentration  $u = u(\boldsymbol{x})$  such that

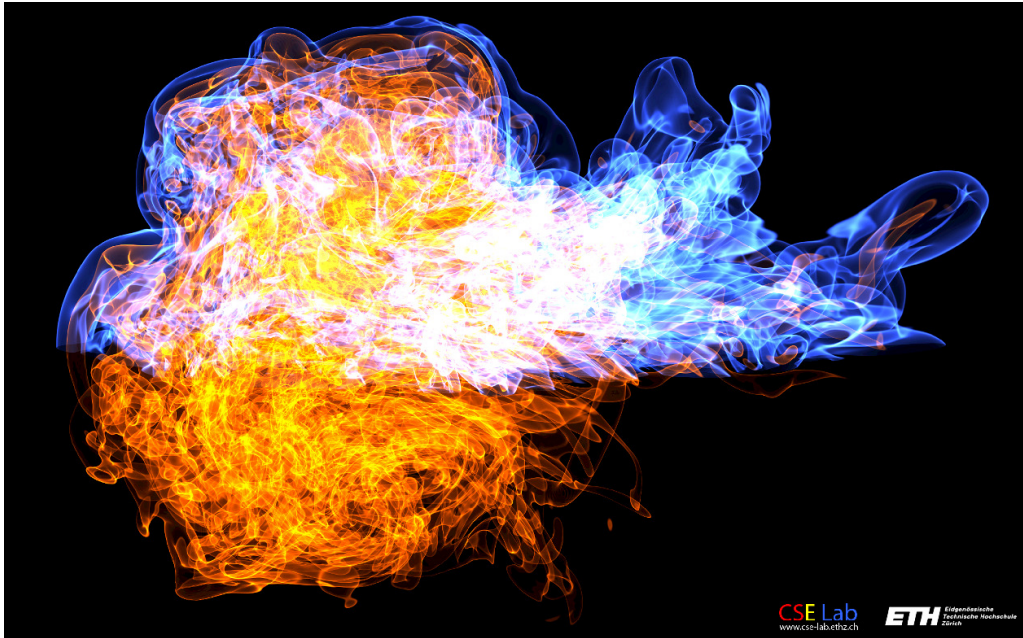
$$-\epsilon \Delta u + \mathbf{v}(\boldsymbol{x}) \cdot \mathbf{grad} u = 0 \quad \text{in } \Omega \quad , \quad u = g \quad \text{on } \partial\Omega .$$

## ③ 2nd-order parabolic evolution problems

Heat conduction: find temperature  $u = u(\mathbf{x}, t)$

$$\frac{\partial}{\partial t} u(\mathbf{x}, t) - \operatorname{div}(\mathbf{A}(\mathbf{x}) \operatorname{grad} u(\mathbf{x}, t)) = 0 \quad \text{in } \Omega \times [0, T] \quad , \quad \begin{aligned} u(\cdot, t) &= g(t) \quad \text{on } \partial\Omega \quad , \\ u(\cdot, 0) &= u_0 \quad \text{in } \Omega \quad . \end{aligned}$$

#### ④ Viscous fluid flow problems



Stokes equations:

$$\begin{aligned} -\Delta \mathbf{u} + \operatorname{grad} p &= \mathbf{f} \quad \text{in } \Omega \quad , \\ \operatorname{div} \mathbf{u} &= 0 \quad \text{in } \Omega \quad , \\ \mathbf{u} &= 0 \quad \text{on } \partial\Omega \quad . \end{aligned}$$

◁ Vortex ring in flow at  $\operatorname{Re} = 7500$ , (P. Koumoutsakos, D-INFK, ETH Zürich)

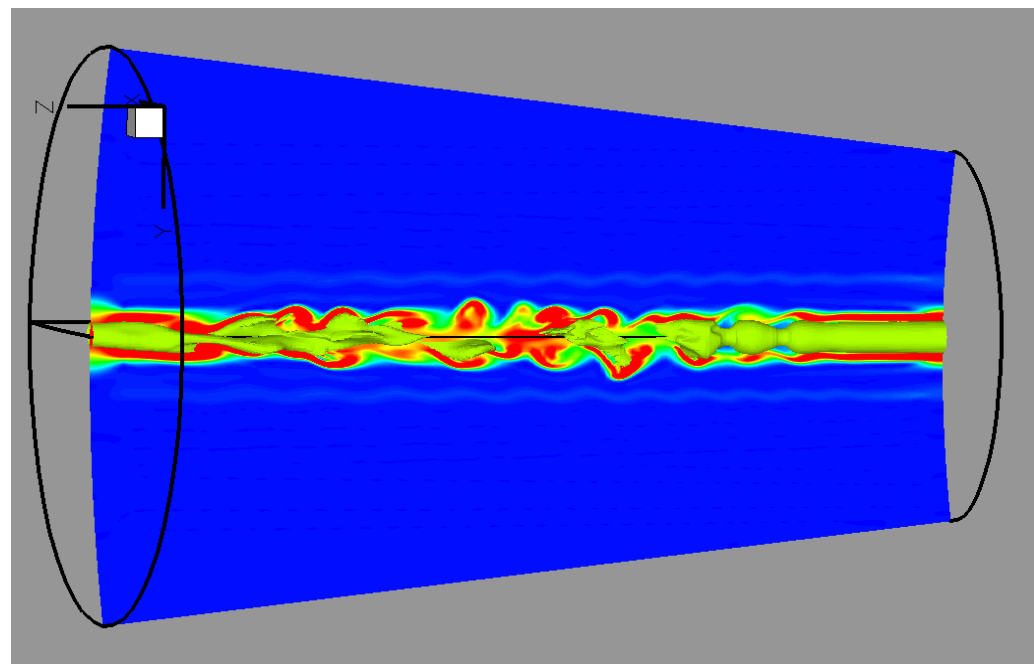
#### ⑤ Conservation laws



1D scalar conservation law with flux  $f$ :

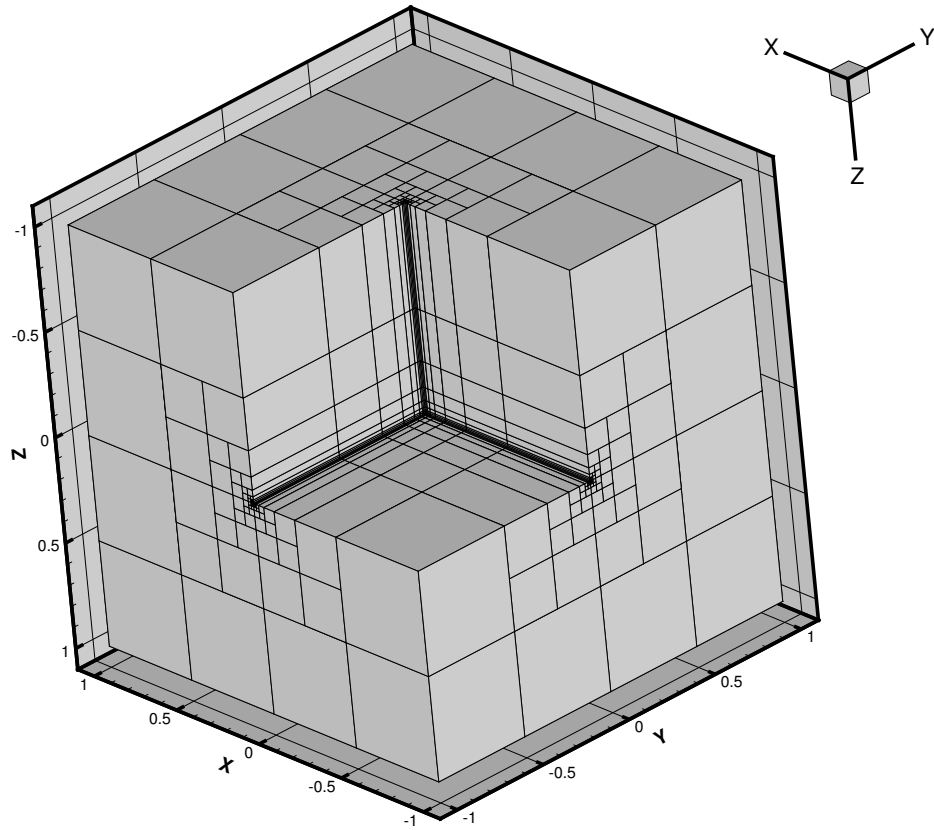
$$\begin{aligned} \frac{\partial}{\partial t} u(x, t) + \frac{\partial}{\partial x} (f(u)) &= 0 && \text{in } \mathbb{R} \times \mathbb{R}^+, \\ u(x, 0) &= u_0(x) && \text{for } x \in \mathbb{R}. \end{aligned}$$

Inviscid fluid flow in 3D (SAM, D-MATH, ETH  
Zürich) ▷



## ⑥ Adaptive finite element methods





◁ Adaptive FEM for diffusion problem:

Geometrically graded mesh at re-entrant corner (SAM, D-MATH, ETH Zürich)

## 7 Multilevel preconditioning

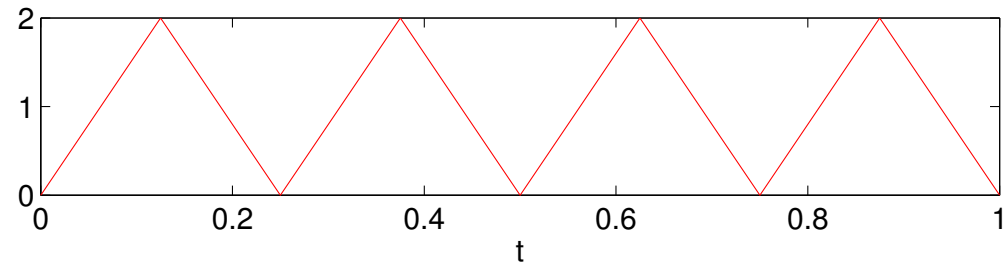
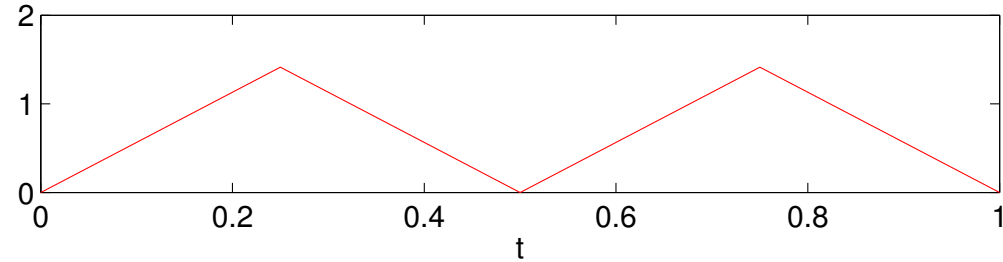
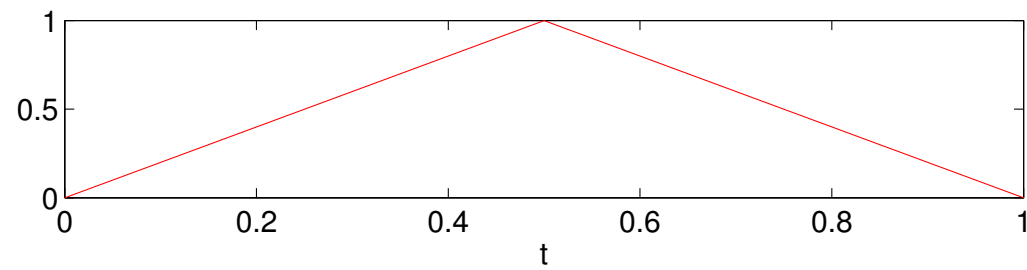
FEM, FD, FV



Huge sparse systems of equations

Efficient preconditioners required

1D hierarchical basis  $\triangleright$



In FS12: Lecturers: Prof. Ralf Hiptmair (SAM, D-MATH)

Classes: Mon 15-17, HG E 7 and Fri 8-10, HG E 3

Tutorials: Mon 8-10, 13-15, Thu 13-15

[LINK to lecture notes](#)

This course will start in autumn term 2012 and deals with methods and techniques for numerical simulation on high performance (parallel) computers.

In HS12: Lecturers: Prof. Petros Koumoutsakos (D-MAVT)  
Prof. Matthias Troyer (D-PHYS)

- Contents:
- Programming Models and Languages
    1. OpenCL
    2. CUDA
    3. Open MP
    4. MPI
  - Computers and Methods
    1. Hardware and Architectures
    2. Libraries
    3. Particles: N-Body solvers (→ numerical integration)
    4. Fields: PDEs

# Bibliography

- [1] C. J. ALPERT, A. B. KAHNG, AND S.-Z. YAO, *Spectral partitioning with multiple eigenvectors*, Discrete Applied Mathematics, 90 (1999), pp. 3 – 26.
- [2] H. AMANN, *Gewöhnliche Differentialgleichungen*, Walter de Gruyter, Berlin, 1st ed., 1983.
- [3] S. AMAT, C. BERMUDEZ, S. BUSQUIER, AND S. PLAZA, *On a third-order Newton-type method free of bilinear operators*, Numerical Lin. Alg., (2010). DOI: 10.1002/nla.654.
- [4] Z.-J. BAI, J. DEMMEL, J. DONGARRA, A. RUHE, AND H. VAN DER VORST, *Templates for the Solution of Algebraic Eigenvalue Problems*, SIAM, Philadelphia, PA, 2000.
- [5] C. BISCHOF AND C. VAN LOAN, *The WY representation of Householder matrices*, SIAM J. Sci. Stat. Comput., 8 (1987).
- [6] S. BÖRM, L. GRASEDYCK, AND W. HACKBUSCH, *Introduction to hierarchical matrices with applications*, Engineering Analysis with Boundary Elements, 27 (2003), pp. 405–422.

- [7] F. BORNEMANN, *A model for understanding numerical stability*, IMA J. Numer. Anal., 27 (2006), pp. 219–231.
- [8] A. BRANDT AND A. LUBRECHT, *Multilevel matrix multiplication and fast solution of integral equations*, J. Comp. Phys., 90 (1990), pp. 348–370.
- [9] E. BRIGHAM, *The Fast Fourier Transform and Its Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
- [10] Q. CHEN AND I. BABUSKA, *Approximate optimal points for polynomial interpolation of real functions in an interval and in a triangle*, Comp. Meth. Appl. Mech. Engr., 128 (1995), pp. 405–417.
- [11] D. COPPERSMITH AND T. RIVLIN, *The growth of polynomials bounded at equally spaced points*, SIAM J. Math. Anal., 23 (1992), pp. 970–983.
- [12] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progression*, J. Symbolic Computing, 9 (1990), pp. 251–280.
- [13] W. DAHMEN AND A. REUSKEN, *Numerik für Ingenieure und Naturwissenschaftler*, Springer, Heidelberg, 2006.
- [14] M. DEAKIN, *Applied catastrophe theory in the social and biological sciences*, Bulletin of Mathematical Biology, 42 (1980), pp. 647–679.
- [15] P. DEUFLHARD, *Newton Methods for Nonlinear Problems*, vol. 35 of Springer Series in Computational Mathematics, Springer, Berlin, 2004.
- [16] P. DEUFLHARD AND F. BORNEMANN, *Scientific Computing with Ordinary Differential Equations*, vol. 42 of Texts in Applied Mathematics, Springer, New York, 2 ed., 2002.

- [17] P. DEUFLHARD AND A. HOHMANN, *Numerische Mathematik I*, DeGruyter, Berlin, 3 ed., 2002.
- [18] P. DUHAMEL AND M. VETTERLI, *Fast fourier transforms: a tutorial review and a state of the art*, *Signal Processing*, 19 (1990), pp. 259–299.
- [19] A. DUTT AND V. ROKHLIN, *Fast Fourier transforms for non-equispaced data II*, *Appl. Comput. Harmon. Anal.*, 2 (1995), pp. 85–100.
- [20] F. FRITSCH AND R. CARLSON, *Monotone piecewise cubic interpolation*, *SIAM J. Numer. Anal.*, 17 (1980), pp. 238–246.
- [21] M. GANDER, W. GANDER, G. GOLUB, AND D. GRUNTZ, *Scientific Computing: An introduction using MATLAB*, Springer, 2005. In Vorbereitung.
- [22] J. GILBERT, C. MOLER, AND R. SCHREIBER, *Sparse matrices in MATLAB: Design and implementation*, *SIAM Journal on Matrix Analysis and Applications*, 13 (1992), pp. 333–356.
- [23] G. GOLUB AND C. VAN LOAN, *Matrix computations*, John Hopkins University Press, Baltimore, London, 2nd ed., 1989.
- [24] C. GOTSMAN AND S. TOLEDO, *On the computation of null spaces of sparse rectangular matrices*, *SIAM J. Matrix Anal. Appl.*, 30 (2008), pp. 445–463.
- [25] C. GRAY, *An analysis of the Belousov-Zhabotinski reaction*, *Rose-Hulman Undergraduate Math Journal*, 3 (2002). <http://www.rose-hulman.edu/mathjournal/archives/2002/vol3-n1/paper1/v3n1-1pd.pdf>.
- [26] L. GREENGARD AND V. ROKHLIN, *A new version of the fast multipole method for the Laplace equation in three dimensions*, *Acta Numerica*, (1997), pp. 229–269.

- [27] M. GUTKNECHT, *Lineare Algebra*, lecture notes, SAM, ETH Zürich, 2009.  
<http://www.sam.math.ethz.ch/~mhg/unt/LA/HS07/>.
- [28] W. HACKBUSCH, *Iterative Lösung großer linearer Gleichungssysteme*, B.G. Teubner–Verlag, Stuttgart, 1991.
- [29] W. HACKBUSCH, *Iterative solution of large sparse systems of equations*, vol. 95 of Applied Mathematical Sciences, Springer-Verlag, New York, 1994. Translated and revised from the 1991 German original.
- [30] W. HACKBUSCH AND S. BÖRM, *Data-sparse approximation by adaptive  $\mathcal{H}^2$ -matrices*, Computing, 69 (2002), pp. 1–35.
- [31] E. HAIRER, C. LUBICH, AND G. WANNER, *Geometric numerical integration*, vol. 31 of Springer Series in Computational Mathematics, Springer, Heidelberg, 2 ed., 2006.
- [32] E. HAIRER, S. NORSETT, AND G. WANNER, *Solving Ordinary Differential Equations I. Nonstiff Problems*, Springer-Verlag, Berlin, Heidelberg, New York, 2 ed., 1993.
- [33] E. HAIRER AND G. WANNER, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, vol. 14 of Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2011.
- [34] C. HALL AND W. MEYER, *Optimal error bounds for cubic spline interpolation*, J. Approx. Theory, 16 (1976), pp. 105–122.
- [35] M. HANKE-BOURGEOIS, *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*, Mathematische Leitfäden, B.G. Teubner, Stuttgart, 2002.

- [36] N. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 2 ed., 2002.
- [37] I. IPSEN AND C. MEYER, *The idea behind Krylov methods*, Technical Report 97-3, Math. Dep., North Carolina State University, Raleigh, NC, January 1997.
- [38] S. JOHNSON, *Notes on the convergence of trapezoidal-rule quadrature*. MIT online course notes, <http://math.mit.edu/~stevenj/trapezoidal.pdf>, 2008.
- [39] D. KALMAN, *A singularly valuable decomposition: The SVD of a matrix*, The College Mathematics Journal, 27 (1996), pp. 2–23.
- [40] M. KOWARSCHIK AND W. C, *An overview of cache optimization techniques and cache-aware numerical algorithms*, in Algorithms for Memory Hierarchies, vol. 2625 of Lecture Notes in Computer Science, Springer, Heidelberg, 2003, pp. 213–232.
- [41] A. LALIENA AND F.-J. SAYAS, *Theoretical aspects of the application of convolution quadrature to scattering of acoustic waves*, Numer. Math., 112 (2009), pp. 637–678.
- [42] A. LENGVILLE AND C. MEYER, *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, Princeton, NJ, 2006.
- [43] D. MCALLISTER AND J. ROULIER, *An algorithm for computing a shape-preserving osculatory quadratic spline*, ACM Trans. Math. Software, 7 (1981), pp. 331–347.
- [44] M. MESSNER, M. SCHANZ, AND E. DARVE, *Fast directional multilevel summation for oscillatory kernels based on chebyshev interpolation*, Journal of Computational Physics, (2011), pp. –.
- [45] C. MOLER, *Numerical Computing with MATLAB*, SIAM, Philadelphia, PA, 2004.



- [46] K. NEYMEYR, *A geometric theory for preconditioned inverse iteration applied to a subspace*, Tech. Rep. 130, SFB 382, Universität Tübingen, Tübingen, Germany, November 1999. Submitted to Math. Comp.
- [47] —, *A geometric theory for preconditioned inverse iteration: III. Sharp convergence estimates*, Tech. Rep. 130, SFB 382, Universität Tübingen, Tübingen, Germany, November 1999.
- [48] K. NIPP AND D. STOFFER, *Lineare Algebra*, vdf Hochschulverlag, Zürich, 5 ed., 2002.
- [49] M. OVERTON, *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, Philadelphia, PA, 2001.
- [50] A. D. H.-D. QI, L.-Q. QI, AND H.-X. YIN, *Convergence of Newton's method for convex best interpolation*, Numer. Math., 87 (2001), pp. 435–456.
- [51] A. QUARTERONI, R. SACCO, AND F. SALERI, *Numerical mathematics*, vol. 37 of Texts in Applied Mathematics, Springer, New York, 2000.
- [52] C. RADER, *Discrete Fourier transforms when the number of data samples is prime*, Proceedings of the IEEE, 56 (1968), pp. 1107–1108.
- [53] R. RANNACHER, *Einführung in die numerische mathematik*. Vorlesungsskriptum Universität Heidelberg, 2000. <http://gaia.iwr.uni-heidelberg.de/>.
- [54] V. ROKHLIN, *Rapid solution of integral equations of classical potential theory*, J. Comp. Phys., 60 (1985), pp. 187–207.
- [55] Y. SAAD, *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, second ed., 2003.

- [56] A. SANKAR, D. SPIELMAN, AND S.-H. TENG, *Smoothed analysis of the condition numbers and growth factors of matrices*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 446–476.
- [57] T. SAUER, *Numerical analysis*, Addison Wesley, Boston, 2006.
- [58] J.-B. SHI AND J. MALIK, *Normalized cuts and image segmentation*, IEEE Trans. Pattern Analysis and Machine Intelligence, 22 (2000), pp. 888–905.
- [59] D. SPIELMAN AND S.-H. TENG, *Spectral partitioning works: planar graphs and finite element meshes*, in Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on, oct 1996, pp. 96 –105.
- [60] M. STEWART, *A superfast toeplitz solver with improved numerical stability*, SIAM J. Matrix Analysis Appl., 25 (2003), pp. 669–693.
- [61] J. STOER, *Einführung in die Numerische Mathematik*, Heidelberger Taschenbücher, Springer, 4 ed., 1983.
- [62] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.
- [63] M. STRUWE, *Analysis für Informatiker*. Lecture notes, ETH Zürich, 2009. <https://moodle-app1.net.ethz.ch/lms/mod/resource/index.php?id=145>.
- [64] F. TISSEUR AND K. MEERBERGEN, *The quadratic eigenvalue problem*, SIAM Review, 43 (2001), pp. 235–286.
- [65] L. TREFETHEN AND D. BAU, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

- [66] P. VERTESI, *On the optimal lebesgue constants for polynomial interpolation*, Acta Math. Hungaria, 47 (1986), pp. 165–178.
- [67] —, *Optimal lebesgue constant for lagrange interpolation*, SIAM J. Numer. Anal., 27 (1990), pp. 1322–1331.

# Index

*LU*-decomposition

existence, 135

$L^2$ -inner product, 1066

*h*-convergence, 1006

MATLAB `arnoldi`, 710

MATLAB `bicgstab`, 570

MATLAB `broyden`, 498

MATLAB `cg`, 535

MATLAB `cholupdate`, 280

MATLAB `chol`, 237

MATLAB `costrans`, 927

MATLAB `ct_rect`, 1110

MATLAB `divide`, 1113

MATLAB `eigs`, 718

MATLAB `fftsolve`, 925

MATLAB `fft`, 857

MATLAB `fzero`, 446

MATLAB `gallery`, 693, 705, 714

MATLAB `gaussquad`, 1077

MATLAB `gmres`, 568

MATLAB `gn`, 816

MATLAB `icostrans`, 928

MATLAB `ifft`, 857

MATLAB `ipoleval`, 319

MATLAB `legendre`, 1076

MATLAB `lsqsvd`, 796

MATLAB `lsqtotal`, 803

MATLAB `lurec`, 119

MATLAB `lu`, 136

MATLAB `minres`, 567

MATLAB newton, 472  
MATLAB ode15s, 1259  
MATLAB ode23s, 1259  
MATLAB ode23, 1215  
MATLAB ode45, 1215  
MATLAB odeset, 1215, 1259  
MATLAB partition, 1113  
MATLAB pcg, 535, 564  
MATLAB pchip, 364  
MATLAB polyfit, 767  
MATLAB polyval, 319  
MATLAB qmr, 570  
MATLAB qrupdate, 277  
MATLAB qr, 257  
MATLAB quad1, 1092  
MATLAB quad, 1092  
MATLAB roudchol, 280  
MATLAB rqui, 650  
MATLAB secant, 459  
MATLAB sinetrans, 921  
MATLAB sinft2d, 924  
MATLAB smw, 273  
MATLAB spline, 375  
MATLAB spowitrp, 693  
MATLAB svds, 728  
MATLAB svd, 728  
MATLAB tic, toc, 925  
MATLAB toeplitz, 935  
MATLAB Adding eps to 1, 146  
MATLAB Characteristic parameters of IEEE floating point numbers, 144  
MATLAB Finding out eps in MATLAB, 146  
MATLAB GE for “Wilkinson system”, 170  
MATLAB Givens rotation, 250  
MATLAB LU-factorization to solve LSE, 120  
MATLAB Numerical differentiation, 325, 326  
MATLAB gallery('circul', v), 847  
MATLAB newtoncotes, 1033  
MATLAB ode45, 1187  
MATLAB odeset, 1216  
MATLAB pconvfft, 858  
MATLAB pconv, 858  
MATLAB polyfit, 948  
MATLAB qrinsert, 285

MATLABrand, 601  
MATLABrank, 731  
MATLABsinetrans, 919  
MATLABeig, 589  
MATLABinput errors and rounding errors, 144  
MATLABlinsolve, 239  
MATLABqrlsqsolve, 791  
MATLABrecursive Gaussian elimination, 104  
MATLABrecursive LU-decomposition, 119  
MATLABbisect , 445  
MATLABblockgs, 104  
MATLABchol, 221  
MATLABdampnewton, 489  
MATLABgausselim, 102  
MATLABpevaltime, 318  
MATLABplanerot, 250  
MATLABpolyfit, 319  
MATLABsa1, 204  
MATLABsa2, 204  
MATLABsa3, 206  
MATLABsparse, 184  
MATLABspdiags, 184  
MATLABspeye, 184  
MATLABspones, 184  
MATLABspy, 198, 221  
MATLABsymamd, 221  
MATLABsymrcm, 221  
3-term recursion  
    for Chebychev polynomials, 959  
    for Legendre polynomials, 1074  
5-points-star-operator, 921  
a posteriori error bound, 423  
a posteriori termination, 419  
a priori termination, 419  
A-inner product, 503  
A-orthogonal, 528  
absolute tolerance, 419, 472, 1197  
Acceptability Criteria, 1108  
adaptive multigrid quadrature, 1086  
adaptive quadrature, 1085  
AGM, 416  
Aitken-Neville scheme, 315  
algebra, 52

algebraic convergence, 946  
algebraic dependence, 61  
analytic function, 974  
Approximation  
    Low rank, 1118  
arrow matrix, 198, 200  
asymptotic complexity, 54  
asymptotic error behavior, 944  
asymptotic rate of linear convergence, 438  
Axiom of roundoff analysis, 145  
XPY operation, 534  
axpy operation, 68  
back substitution, 92  
backward error analysis, 154  
backward substitution, 120  
Banach's fixed point theorem, 434  
Bandbreite  
    Zeilen-, 211  
banded matrix, 209  
bandwidth, 209  
    lower, 209  
    minimizing, 219  
    upper, 209  
barycentric interpolation formula, 313  
basic linear algebra subroutines, 63  
basis  
    cosine, 926  
    orthonormal, 260, 587  
    sine, 917  
    trigonometric, 852  
Belousov-Zhabotinsky reaction, 1189  
bending energy, 377  
Besetzungsmuster, 221  
best low rank approximation, 757  
bicg, 569  
BiCGStab, 570  
bisection, 444  
BLAS, 63  
    axpy, 68  
block LU-decomposition, 122  
block matrix multiplication, 52  
blow-up, 1192  
blurring operator, 882

Bounding Box, 1109

Broyden

- Quasi-Newton Method, 493

Broyden-Verfahren

- convergence monitor, 496

Butcher scheme, 1184, 1252

cache miss, 40

cache thrashing, 40

cancellation, 326, 328

- for Householder transformation, 249

capacitance, 177

capacitor, 177

cardinal

- spline, 383

cardinal basis, 296

cardinal basis function, 382

cardinal interpolant, 382

Cauchy product

- of power series, 835

causal filter, 826

cell

- of a mesh, 1003

CG

- convergence, 548
- preconditioned, 555
- termination criterion, 535

CG = conjugate gradient method, 522

CG algorithm, 532

chain rule, 474

characteristic polynomial, 583

Chebyshev expansion, 977

Chebyshev nodes, 963, 966, 968

Chebyshev polynomials, 543

- 3-term recursion, 959

Chebyshev-interpolation, 957

chemical reaction kinetics, 1240

Cholesky decomposition, 235

- costs, 236

Cholesky factorization, 277

circuit simulation

- transient, 1145

circulant matrix, 841

Classical Runge-Kutta method



Butcher scheme, 1185, 1186  
Clenshaw algorithm, 978  
cluster analysis, 742  
Clustering Approximation  
  Task, 1093  
coil, 177  
Collocation Matrix, 1093, 1094  
column major matrix format, 35  
column sum norm, 150  
column transformation, 51  
complexity, 54  
  asymptotic, 54  
  linear, 58  
  of SVD, 729  
composite quadrature formulas, 1037  
compressed row storage, 182  
computational cost  
  Gaussian elimination, 97  
computational costs  
  LU-decomposition, 118  
  QR-decomposition, 262  
computational effort, 54, 466  
eigenvalue computation, 593  
condition number  
  of a matrix, 168  
  spectral, 521  
conjugate gradient method, 522  
consistency  
  of iterative methods, 406  
  fixed point iteration, 426  
constant  
  Lebesgue, 968  
constitutive relations, 177, 290  
constrained least squares, 804  
convergence  
  algebraic, 946  
  asymptotic, 461  
  exponential, 946, 954, 970  
  global, 407  
  iterative method, 406  
  linear, 409  
  linear in Gauss-Newton method, 820  
  local, 407  
  quadratic, 416

rate, 409

convergence monitor

- of Broyden method, 496

convolution

- discrete, 826, 834
- discrete periodic, 837
- of sequences, 835

cosine

- basis, 926
- transform, 926

cosine matrix, 926

cosine transform, 926

costs

- Cholesky decomposition, 236

Crout's algorithm, 116

CRS, 182

CRS format

- diagonal, 183

cubic Hermite interpolation, 310, 359

cubic spline interpolation

- error estimates, 1019

cyclic permutation, 203

damped Newton method, 485

damping factor, 487

data fitting, 764

- linear, 765
- polynomial, 767

data interpolation, 290

deblurring, 879

deblurring = Entrauschen, 879

definite, 148

dense matrix, 175

descent methods, 503

DFT, 845, 856

- two-dimensional, 877

Diagonal dominance, 227

diagonal matrix, 112

diagonalization

- of a matrix, 587

diagonalization of local translation invariant linear operators, 921

difference quotient, 325

- backward, 1161
- forward, 1160

difference scheme, 1160  
 differential in non-linear least squares, 474  
 direct power method, 615  
 discrete convolution, 826, 834  
 discrete Fourier transform, 845, 856  
 discrete periodic convolution, 837  
 divided differences, 332  
 dot product, 41  
 double precision, 142  
 economical singular value decomposition, 729  
 efficiency, 466  
 eigenspace, 583  
 eigenvalue, 583  
     generalized, 588  
 eigenvalue problem  
     generalized, 588  
 eigenvector, 583  
     generalized, 588  
 electric circuit, 176, 401  
     resonant frequencies, 574  
 elementary arithmetic operations, 139, 145  
 elimination matrix, 110  
 energy norm, 503  
 envelope  
     matrix, 211  
 Equation  
     non-linear, 404  
 equidistant mesh, 1003  
 ergodicity, 611  
 error behavior  
     asymptotic, 944  
 error estimator  
     a posteriori, 423  
 Euler method  
     explicit, 1158  
     implicit, 1161  
     semi implicit, 1257  
 Euler polygon, 1159  
 Euler's iteration, 457  
 expansion  
     asymptotic, 320  
 explicit Euler method, 1158  
     Butcher scheme, 1185

explicit midpoint rule  
     Butcher scheme, 1185  
     for ODEs, 1181  
 explicit Runge-Kutta method, 1183  
 explicit trapezoidal rule  
     Butcher scheme, 1185  
 exponential convergence, 970  
 extended normal equations, 782  
 extended state space  
     of an ODE, 1148  
 extrapolation, 320  
 Far field, 1104  
 fast Fourier transform, 906  
 FFT, 906  
 fill-in, 198  
 filter  
     high pass, 867  
     low pass, 867  
 finite filter, 826  
 fixed point, 426  
 fixed point form, 427  
 fixed point iteration, 425  
 fixed point iteration  
     consistency, 426  
 floating point number, 141  
 floating point numbers, 140  
 forward elimination, 91  
 forward substitution, 120  
 Fourier  
     matrix, 853  
 Fourier coefficient, 899  
 Fourier series, 893  
 Fourier transform, 893  
     discrete, 845, 856  
     isometry property, 903  
 fractional order of convergence, 460  
 frequency filtering, 859  
 function representation, 292  
 Funktion  
     shandles, 472  
 fzero, 446  
 Gauss Quadrature, 1063

Gauss-Newton method, 814  
 Gauss-Seidel preconditioner, 558  
 Gaussian elimination, 89  
     block version, 105  
     by rank-1 modifications, 103  
     for non-square matrices, 100  
     instability, 170  
     with partial pivoting, 127  
 Gerschgorin circle theorem, 585  
 Gibbs phenomenon, 998  
 Givens rotation, 250, 274  
 Givens-Rotation, 264, 284, 287  
 global solution  
     of an IVP, 1153  
 GMRES, 568  
     restarted, 568  
 Golub-Welsch algorithm, 1077  
 gradient, 475, 509  
 Gram-Schmidt  
     Orthonormalisierung, 709  
 Gram-Schmidt orthogonalization, 530, 709  
 Gram-Schmidt orthonormalization, 679, 787  
 graph partitioning, 644  
 grid, 1002  
     grid cell, 1003  
     grid function, 922  
     grid interval, 1003  
 Halley's iteration, 450, 457  
 harmonic mean, 363  
 hat function, 294  
 heartbeat model, 1142  
 Hermite interpolation  
     cubic, 310  
 Hessian, 231  
 Hessian = Hesse-Matrix, 231  
 Hessian matrix, 475  
 high pass filter, 867  
 homogeneous, 148  
 Hooke's law, 661  
 Horner scheme, 301  
 Householder reflection, 247  
 IEEE standard 754, 142  
 ill conditioned, 172

image segmentation, 623  
implicit Euler method, 1161  
impulse response, 827  
    of a filter, 826  
in place, 116, 118  
in situ, 104, 118  
in-situ, 132  
increment equations  
    linearized, 1258  
increments  
    Runge-Kutta, 1183, 1251  
inductance, 177  
inductor, 177  
inf, 143  
infinity, 143  
initial guess, 406, 425  
initial value problem  
    stiff, 1247  
initial value problem (IVP), 1147  
initial value problem (IVP) = Anfangswertproblem, 1147  
inner product

A-, 503  
intermediate value theorem, 444  
interpolation  
    barycentric formula, 313  
    Chebychev, 957  
    complete cubic spline, 374  
    cubic Hermite, 359  
    Hermite, 309  
    Lagrange, 303  
    natural cubic spline, 374  
    periodic cubic spline, 375  
    spline cubic, 369  
    spline cubic, locality, 383  
    spline shape preserving, 385  
    trigonometric, 983  
interpolation operator, 298  
inverse interpolation, 463  
inverse iteration, 646  
    preconditioned, 652  
inverse matrix, 87  
invertible matrix, 87, 88  
iteration

Halley's, 457  
Euler's, 457  
quadratical inverse interpolation, 457  
iteration function, 406, 425  
iterative method  
    convergence, 406  
IVP, 1147  
Jacobi preconditioner, 558  
Jacobian, 435, 470  
Kernel Function, 1093, 1094  
    separable, 1096  
kinetics  
    of chemical reaction, 1240  
Kirchhoff (current) law, 177  
knots  
    spline, 367  
Konvergenz  
    Algebraische, Quadratur, 1082  
Kronecker symbol, 31  
Krylov space, 526  
    for Ritz projection, 697  
L-stable, 1254  
Lagrange multiplier, 805  
Lagrangian multiplier, 806  
Landau-O, 55, 947  
Lapack, 99  
leading coefficient  
    of polynomial, 300  
Least squares  
    with linear constraint, 804  
least squares  
    total, 801  
least squares problem, 772, 773  
    conditioning, 776  
Lebesgue  
    constant, 968  
Lebesgue constant, 341  
Levinson algorithm, 935  
limit cycle, 1243  
limiter, 362  
line search, 508  
linear complexity, 58  
linear correlation, 738

- linear data fitting, 765
- linear electric circuit, 176
- linear filter, 826
- linear operator
  - diagonalization, 921
- linear ordinary differential equation, 581
- linear regression, 62
- linear system of equations, 86
  - multiple right hand sides, 102
- Lloyd-Max algorithm, 742
- local a posteriori error estimation
  - for adaptive quadrature, 1086
- local convergence
  - Newton method, 486
- local linearization, 470
- local mesh refinement
  - for adaptive quadrature, 1086
- locality
  - of interpolation, 381
- logistic differential equation, 1137
- Lotka-Volterra ODE, 1140
- low pass filter, 867
- Low rank approximation, 1118
- lower triangular matrix, 112
- LU-decomposition
  - blocked, 122
  - computational costs, 118
  - envelope aware, 215
  - existence, 114
  - in place, 118
- LU-factorization
  - envelope aware, 215
  - of sparse matrices, 195
  - with pivoting, 130
- machine number, 141
  - exponent, 141
- machine numbers, 142
  - distribution, 141
- machine precision, 145
- mantissa, 141
- Markov chain, 599, 931
  - stationary distribution, 602
- mass matrix, 665



Matlab, 26

MATLAB \-operator, 99

Matrix

adjoint, 34

Collocation-, 1093

Hermitian, 587

Hermitian transposed, 34

normal, 587

obere Dreiecks, 281

obere Hessenberg, 275

skew-Hermitian, 587

transposed, 34

unitary, 587

matrix

banded, 209

condition number, 168

dense, 175

diagonal, 112

envelope, 211

Fourier, 853

Hermitian, 230

Hessian, 475

lower triangular, 112

normalized, 112

orthogonal, 243

positive definite, 230

positive semi-definite, 230

rank, 87

sine, 917

sparse, 175

storage formats, 35

structurally symmetric, 218

symmetric, 230

tridiagonal, 209

unitary, 243

upper triangular, 112

matrix algebra, 52

matrix block, 33

matrix factorization, 106, 110

matrix norm, 149

column sums, 150

row sums, 150

matrix product, 41

matrix storage

envelope oriented, 218  
Matrixnorm, 149  
  Submultiplikativität, 150  
mesh, 1002  
  equidistant, 1003  
  in time, 1166  
mesh width, 1003  
Method  
  Quasi-Newton, 492  
midpoint rule, 1032, 1181  
Milleniums Algorithms, 1135  
Milne rule, 1034  
min-max theorem, 635  
minimal residual methods, 566  
model function, 447  
Modellfunktionsverfahren, 446  
modification technique  
  QR-factorization, 274  
monomial representation  
  of a polynomial, 300  
monomials, 300  
multi-point methods, 447, 458  
multiplicity  
  geometric, 583  
  of an interpolation node, 308  
NaN, 143  
Ncut, 626  
Near field, 1104  
nested  
  subspaces, 523  
Newton  
  basis, 330  
  damping, 487  
  damping factor, 487  
  monotonicity test, 488  
  simplified method, 476  
Newton correction, 470  
  simplified, 484  
Newton iteration, 470  
  numerical Differentiation, 477  
  termination criterion, 482  
Newton method  
  1D, 447

damped, 485  
local convergence, 486  
local quadratic convergence, 477  
region of convergence, 486  
Newton's law of motion, 664  
nodal analysis, 176, 401  
nodal potentials, 177  
node  
  double, 309  
  for interpolation, 303  
  in electric circuit, 176  
  multiple, 308  
  multiplicity, 308  
  of a mesh, 1003  
  quadrature, 1026  
nodes, 303  
  Chebychev, 966  
  Chebychev nodes, 968  
  for interpolation, 303  
non-linear data fitting, 810  
non-normalized numbers, 143  
norm, 148  
 $L^1$ , 340  
 $L^2$ , 340  
 $\infty$ -, 149  
1-, 149  
energy-, 503  
Euclidean, 149  
of matrix, 149  
Sobolev semi-, 956  
supremum, 339  
normal equations, 779  
  extended, 782  
  with constraint, 807  
normalization, 615  
normalized lower triangular matrix, 111  
normalized triangular matrix, 112  
not a number, 143  
Nullstellenbestimmung  
  Modellfunktionsverfahren, 446  
numerical algorithm, 153  
Numerical differentiation  
  roundoff, 326

numerical Differentiation  
    Newton iteration, 477  
numerical differentiation, 321, 325  
numerical quadrature, 1023  
numerical rank, 796  
Obere Hessenbergmatrix, 275  
ODE, 1147  
    scalar, 1157  
Ohmic resistor, 177  
one-point methods, 447  
order  
    of quadrature formula, 1048  
order of convergence, 413  
    fractional, 460  
ordinary differential equation  
    linear, 581  
ordinary differential equation (ODE), 1147  
oregonator, 1189  
orthogonal matrix, 243  
orthogonal polynomials, 1069  
orthonormal basis, 260, 587

overflow, 143  
page rank, 599  
    stochastic simulation, 600  
partial pivoting, 130, 132  
    choice of pivot, 132  
pattern  
    of a matrix, 46  
PCA, 721  
PCG, 555  
Peano  
    Theorem of, 1153  
penalization, 638  
penalty parameter, 639  
periodic sequence, 835  
permutation, 134  
permutation matrix, 134  
Permutationsmatrix, 282  
perturbation lemma, 167  
Petrov-Galerkin condition, 569  
phase space  
    of an ODE, 1148

Picard-Lindelöf

Theorem of, 1153

piecewise quadratic interpolation, 353

PINVIT, 652

Pivot

choice of, 128

pivot, 92, 94, 95

pivot row, 92, 95

pivoting, 124

point spread function, 879

polynomial

characteristic, 583

Lagrange, 303

polynomial fitting, 767

polynomial interpolation

existence and uniqueness, 305

generalized, 308

polynomial space, 300

positive definite

criteria, 230

matrix, 230

potentials

nodal, 177

power spectrum

of a signal, 869

preconditioned CG method, 555

preconditioned inverse iteration, 652

preconditioner, 552

preconditioning, 550

predator-prey model, 1140

principal axis, 742

principal axis transformation, 515, 587

principal component, 738

principal component analysis, 721

principal minor, 122

problem

ill conditioned, 172

sensitivity, 172

well conditioned, 172

product rule, 475

pseudoinverse, 796

Punkt

stationär, 1142

pwer method

direct, 615  
 QR algorithm, 590  
 QR-algorithm with shift, 590  
 QR-decomposition  
   computational costs, 262  
 QR-factorization, QR-decomposition, 253  
 quadratic convergence, 441  
 quadratic eigenvalue problem, 574  
 quadratic functional, 504  
 quadratic interpolation  
   piecewise, 353  
 quadratic inverse interpolation, 464  
 quadratical inverse interpolation, 457  
 quadrature  
   adaptive, 1085  
   polynomial formulas, 1030  
 quadrature formula, 1026  
   local, 1039  
   order, 1048  
 quadrature node, 1026  
 quadrature numerical, 1023  
 quadrature weight, 1026  
 Quasi-Newton Method, 493  
 Quasi-Newton method, 492  
 Radau RK-method  
   order 3, 1255  
   order 5, 1255  
 radiative heat transfer, 838  
 rank  
   column rank, 88  
   computation, 729  
   numerical, 796  
   of a matrix, 87  
   row rank, 88  
 rank-1 modification, 103  
 rank-1-modifications, 270  
 rate  
   of algebraic convergence, 946  
   of convergence, 409  
 Rayleigh quotient, 617, 635  
 Rayleigh quotient iteration, 649  
 regular matrix, 87

relative tolerance, 419, 472, 1197  
rem:Fspec, 854  
rem:polyerrrep, 952  
residual quantity, 653  
Riccati differential equation, 1157, 1159  
right hand side  
    of an ODE, 1148  
Riemann sum, 899  
right hand side vector, 86, 153  
rigid body mode, 668  
Ritz projection, 686, 696  
Ritz value, 686  
Ritz vector, 686  
root of unity, 851  
rounding, 145  
rounding up, 145  
roundoff  
    for numerical differentiation, 326  
row major matrix format, 35  
row permutation, 287  
row sum norm, 150  
row transformation, 51, 90, 108, 109

Runge-Kutta  
    increments, 1183, 1251  
Runge-Kutta method, 1183, 1251  
    L-stable, 1254  
Runge-Kutta methods  
    semi-implicit, 1256  
    stability function, 1229, 1253  
saddle point problem, 806  
    matrix form, 807  
scalar ODE, 1157  
scaling  
    of a matrix, 48  
scheme  
    Aitken-Neville, 315  
    Horner, 301  
Schur  
    Komplement, 123  
Schur complement, 123  
Schur's lemma, 586  
scientific notation, 140  
secant condition, 492

secant method, 458, 492

segmentation  
  of an image, 623

semi-implicit Euler method, 1257

seminorm, 956

sensitivity  
  of a problem, 172

shape  
  preservation, 353  
  preserving spline interpolation, 385

Sherman-Morrison-Woodbury formula, 272

shifted inverse iteration, 647

similarity  
  of matrices, 585

similarity function  
  for image segmentation, 625

similarity transformations, 585

similarity transformation  
  unitary, 590

Simpson rule, 1034

sine  
  basis, 917

  matrix, 917

  transform, 918

Sine transform, 917

single precision, 142

single step method, 1166

singular value decomposition, 719, 726

sparse matrix, 175  
  initialization, 187  
  LU-factorization, 195  
  multiplication, 191

sparse matrix storage formats, 181

spectral condition number, 521

spectral partitioning, 644

spectral radius, 584

spectrum, 583  
  of a matrix, 514

spline, 367  
  cardinal, 383  
  complete cubic, 374  
  cubic, 369  
  cubic, locality, 383  
  knots, 367



- natural cubic, 374
- periodic cubic, 375
- physical, 379
- shape preserving interpolation, 385
- square root
  - of a matrix, 551
- stability function
  - of explicit Runge-Kutta methods, 1229
  - of Runge-Kutta methods, 1253
- Stable
  - algorithm, 153
- stable
  - numerically, 153
- state space
  - of an ODE, 1148
- stationary distribution, 602
- steepest descent, 508
- stiff IVP, 1247
- stiffness matrix, 665
- stochastic matrix, 603
- stochastic simulation of page rank, 600
- Strassen's algorithm, 56
- structurally symmetric matrix, 218
- sub-matrix, 33
- sub-multiplicative, 150
- subspace correction, 523
- subspace iteration
  - for direct power method, 692
- subspaces
  - nested, 523
- SVD, 719, 726
- symmetry
  - structural, 218
- system matrix, 86, 153
- system of equations
  - linear, 86
- tangent field, 1157
- Taylor expansion, 439
- Taylor formula, 941
- Taylor's formula, 439
- Tensor product
  - Chebyshev interpolation polynomial, 1099, 1119, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258, 1259, 1260, 1261, 1262, 1263, 1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283, 1284, 1285, 1286, 1287, 1288, 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337, 1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1351, 1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 1392, 1393, 1394, 1395, 1396, 1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460, 1461, 1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 1474, 1475, 1476, 1477, 1478, 1479, 1480, 1481, 1482, 1483, 1484, 1485, 1486, 1487, 1488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 1496, 1497, 1498, 1499, 1500, 1501, 1502, 1503, 1504, 1505, 1506, 1507, 1508, 1509, 1510, 1511, 1512, 1513, 1514, 1515, 1516, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542, 1543, 1544, 1545, 1546, 1547, 1548, 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 1570, 1571, 1572, 1573, 1574, 1575, 1576, 1577, 1578, 1579, 1580, 1581, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1619, 1620, 1621, 1622, 1623, 1624, 1625, 1626, 1627, 1628, 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636, 1637, 1638, 1639, 1640, 1641, 1642, 1643, 1644, 1645, 1646, 1647, 1648, 1649, 1650, 1651, 1652, 1653, 1654, 1655, 1656, 1657, 1658, 1659, 1660, 1661, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1672, 1673, 1674, 1675, 1676, 1677, 1678, 1679, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1690, 1691, 1692, 1693, 1694, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1712, 1713, 1714, 1715, 1716, 1717, 1718, 1719, 1720, 1721, 1722, 1723, 1724, 1725, 1726, 1727, 1728, 1729, 1730, 1731, 1732, 1733, 1734, 1735, 1736, 1737, 1738, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747, 1748, 1749, 1750, 1751, 1752, 1753, 1754, 1755, 1756, 1757, 1758, 1759, 1760, 1761, 1762, 1763, 1764, 1765, 1766, 1767, 1768, 1769, 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 1783, 1784, 1785, 1786, 1787, 1788, 1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1798, 1799, 1800, 1801, 1802, 1803, 1804, 1805, 1806, 1807, 1808, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1840, 1841, 1842, 1843, 1844, 1845, 1846, 1847, 1848, 1849, 1850, 1851, 1852, 1853, 1854, 1855, 1856, 1857, 1858, 1859, 1860, 1861, 1862, 1863, 1864, 1865, 1866, 1867, 1868, 1869, 1870, 1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010
  - interpolation polynomial, 1098

tensor product, 41  
tent function, 294  
Toeplitz matrices, 929  
termination criterion, 418  
    Newton iteration, 482  
    reliable, 420  
    residual based, 421  
time-invariant filter, 826  
timestep (size), 1159  
timestep constraint, 1230  
timestepping, 1158  
Toeplitz solvers  
    fast algorithms, 938  
tolerance  
    absolute, 445  
tolerance, 421  
    absolute, 1197  
    absoute, 419, 472  
    for adaptive timestepping for ODEs, 1195  
    for termination, 419  
    realtive, 1197  
    relative, 419, 472

total least squares, 801  
trajectory, 1141  
transform  
    cosine, 926  
    fast Fourier, 906  
    sine, 918  
trapezoidal rule, 1033, 1180  
    for ODEs, 1181  
trend, 721  
triangle inequality, 148  
tridiagonal matrix, 209  
trigonometric basis, 852  
trigonometric function, 984  
trigonometric interpolation, 983  
trigonometric polynomial, 901  
trigonometric polynomials, 984  
trigonometric transformations, 916  
truss structure  
    vibrations, 660  
trust region method, 822  
underflow, 143

Uniform convergence

of Fourier series, 894

unit vector, 31

unitary matrix, 243

unitary similiary transformation, 590

upper Hessenberg matrix, 711

upper triangular matrix, 92, 111, 112

Vandermonde matrix, 306

variational calculus, 377

Weddle rule, 1034

weight

quadrature, 1026

well conditioned, 172

Young's modulus, 662

Zerlegung

LU, 120

QR, 281

zero padding, 843, 935, 994

# List of Symbols

$(\mathbf{A})_{i,j} \hat{=}$  reference to entry  $a_{ij}$  of matrix  $\mathbf{A}$ , 33  
 $(\mathbf{A})_{k:l,r:s} \hat{=}$  reference to submatrix of  $\mathbf{A}$  spanning rows  $k, \dots, l$  and columns  $r, \dots, s$ , 33  
 $(\mathbf{x})_i \hat{=}$   $i$ -th component of vector  $\mathbf{x}$ , 31  
 $(x_k) * _n (y_k) \hat{=}$  discrete periodic convolution, 838  
 $C_{\text{pw}}^0(I) \hat{=}$  space of piecewise continuous functions on interval  $I$ , 1004  
 $C^1([a, b]) \hat{=}$  space of continuously differentiable functions  $[a, b] \mapsto \mathbb{R}$ , 355  
 $D\Phi \hat{=}$  **Jacobian** of  $\Phi : D \mapsto \mathbb{R}^n$  at  $\mathbf{x} \in D$ , 435  
 $D_{\mathbf{y}}\mathbf{f} \hat{=}$  Derivative of  $\mathbf{f}$  w.r.t..  $\mathbf{y}$  (Jacobian), 1152  
 $J(t_0, \mathbf{y}_0) \hat{=}$  maximal domain of definition of a solution of an IVP, 1153  
 $O \hat{=}$  zero matrix, 35

$O(n)$ , 55  
 $\mathcal{E} \hat{=}$  expected value of a random variable, 932  
 $\mathcal{P}_n^T \hat{=}$  space of **trigonometric polynomials** of degree  $n$ , 984  
 $\mathcal{R}_k(m, n)$ , 758  
 $\text{eps} \hat{=}$  machine precision, 145  
 $\text{Eig}_{\mathbf{A}}(\lambda) \hat{=}$  eigenspace of  $\mathbf{A}$  for eigenvalue  $\lambda$ , 583  
 $\text{Im}(\mathbf{A}) \hat{=}$  range/column space of matrix  $\mathbf{A}$ , 730  
 $\text{Ker}(\mathbf{A}) \hat{=}$  nullspace of matrix  $\mathbf{A}$ , 730  
 $\mathcal{K}_l(\mathbf{A}, \mathbf{z}) \hat{=}$  Krylov subspace, 526  
 $\|\mathbf{Ax} - \mathbf{b}\|_2 \rightarrow \min \hat{=}$  minimize  $\|\mathbf{Ax} - \mathbf{b}\|_2$ , 772  
 $\|\mathbf{A}\|_F^2$ , 757

$\|\mathbf{x}\|_A \hat{=}$  energy norm induced by s.p.d. matrix  $\mathbf{A}$ , 503  
 $\|f\|_{L^\infty(I)}$ , 339  
 $\|f\|_{L^1(I)}$ , 340  
 $\|f\|_{L^2(I)}^2$ , 340  
 $\mathcal{P}_k$ , 300  
 $\Psi^h \mathbf{y} \hat{=}$  discrete evolution for autonomous ODE, 1166  
 $\mathcal{S}_{d,\mathcal{M}}$ , 367  
 $\mathbf{A}^+$ , 776  
 $\mathbf{A}^\top \hat{=}$  transposed matrix, 34  
 $\mathbf{I} \hat{=}$  identity matrix, 35  
 $\mathbf{h} * \mathbf{x} \hat{=}$  discrete convolution of two vectors, 834  
 $\mathbf{x} *_n \mathbf{y} \hat{=}$  discrete periodic convolution of vectors, 838  
 $\bar{z} \hat{=}$  complex conjugation, 34  
 $\mathbb{M} \hat{=}$  set of machine numbers, 138  
 $\mathbb{S}^1 \hat{=}$  unit circle in  $\mathbb{C}$ , 987  
 $\delta_{ij} \hat{=}$  Kronecker symbol, 31, 303  
 $\mathbf{i} \hat{=}$  imaginary unit, “ $\mathbf{i} := \sqrt{-1}$ ”, 178  
 $\kappa(\mathbf{A}) \hat{=}$  spectral condition number, 521

$\lambda_{\max} \hat{=}$  largest eigenvalue (in modulus), 521  
 $\lambda_{\min} \hat{=}$  smallest eigenvalue (in modulus), 521  
 $\mathbf{1} = (1, \dots, 1)^T$ , 1229, 1253  
 $\text{Ncut}(\mathcal{X}) \hat{=}$  normalized cut of subset of weighted graph, 626  
 $\text{argmin} \hat{=}$  (global) minimizer of a functional, 507  
 $\text{cond}(\mathbf{A})$ , 168  
 $\text{cut}(\mathcal{X}) \hat{=}$  cut of subset of weighted graph, 626  
 $\text{env}(\mathbf{A})$ , 211  
 $\text{nnz}$ , 175  
 $\text{rank}(\mathbf{A}) \hat{=}$  rank of matrix  $\mathbf{A}$ , 87  
 $\text{rd}$ , 145  
 $\text{weight}(\mathcal{X}) \hat{=}$  connectivity of subset of weighted graph, 626  
 $\overline{m}(\mathbf{A})$ , 209  
 $\rho(\mathbf{A}) \hat{=}$  spectral radius of  $\mathbf{A} \in \mathbb{K}^{n,n}$ , 584  
 $\rho_{\mathbf{A}}(\mathbf{u}) \hat{=}$  Rayleigh quotient, 617  
 $\mathbf{f} \hat{=}$  right hand side of an ODE, 1148  
 $\sigma(\mathbf{A}) \hat{=}$  spectrum of matrix  $\mathbf{A}$ , 583  
 $\sigma(\mathbf{M}) \hat{=}$  spectrum of matrix  $\mathbf{M}$ , 514  
 $\tilde{\star}$ , 144  
 $\underline{m}(\mathbf{A})$ , 209

$m(\mathbf{A})$ , 209

$y[t_i, \dots, t_{i+k}] \hat{=} \text{divided difference}$ , 332

$\| \mathbf{x} \|_1$ , 149

$\| \mathbf{x} \|_2$ , 149

$\| \mathbf{x} \|_\infty$ , 149

$\dot{\cdot} \hat{=} \text{Derivative w.r.t. time } t$ , 1138

TOL tolerance, 1195

# List of Definitions

- Analytic function, 974
- Arrow matrix, 200
- Chebyshev polynomial, 958
- circulant matrix, 841
- Cluster
  - Tree, 1109
- concave
  - data, 347
  - function, 348
- Condition (number) of a matrix, 168
- Consistency of fixed point iterations, 426
- Consistency of iterative methods, 406
- Contractive mapping, 433
- Convergence, 406
  - global, 407
  - local, 407
- convex
  - data, 347
  - function, 348
- data
  - concave, 347
  - convex, 347
- Diagonally dominant matrix, 227
- Discrete convolution, 834
- discrete Fourier transform, 856
- discrete periodic convolution, 837
- eigenvalues and eigenvectors, 583
- energy norm, 503

equivalence of norms, 410  
 Evolution operator, 1155  
 Explicit Runge-Kutta method, 1183  
 Fill-In, 198  
 Frobenius norm, 757  
 function  
     concave, 348  
     convex, 348  
 Hessian matrix, 475  
 Inverse of a matrix, 87  
 Krylov space, 526  
 L-stable Runge-Kutta method, 1254  
 Lebesgue constant, 341  
 Legendre polynomials, 1070  
 Linear convergence, 409  
 Lipschitz continuous function, 1151, 1152  
 machine numbers, 141  
 matrix  
     generalized condition number, 777  
     s.p.d, 230  
     symmetric positive definite, 230  
 Matrix envelope, 211  
 matrix norm, 149  
 monotonic data, 346  
 norm, 148  
     Frobenius norm, 757  
 Normalized cut, 626  
 numerical algorithm, 153  
 Order of convergence, 413  
 orthogonal matrix, 243  
 Permutation matrix, 134  
 polynomial  
     Chebychev, 958  
     generalized Lagrange, 309  
 Polynomial  
     Interpolation tensor product, 1098  
 pseudoinverse, 776  
 Rank of a matrix, 87  
 Rayleigh quotient, 617



residual, 161

Runge-Kutta method, 1251

Single step method, 1166

singular value decomposition (SVD), 726

sparse matrices, 175

sparse matrix, 175

splines, 367

stable algorithm, 153

Structurally symmetric matrix, 218

Tensor product

    interpolation polynomial, 1098

Toeplitz matrix, 933

Types of matrices, 112

unitary matrix, 243

# Examples and Remarks

- $LU$ -decomposition of sparse matrices, 196
- $L^2$ -error estimates for polynomial interpolation, 955
- $h$ -adaptive numerical quadrature, 1089
- $p$ -convergence of piecewise polynomial interpolation, 1011
- (Nearly) singular LSE in shifted inverse iteration, 648
- [Bad behavior of global polynomial interpolants, 349
- [Conditioning of row transformations, 241
- [From higher order ODEs to first order systems, 1150
- [Stable solution of LSE by means of QR-decomposition, 263
- `ode45` for stiff problem, 1220
- “Annihilating” orthogonal transformations in 2D, 246
- “Butcher barriers” for explicit RK-SSM, 1186
- “Failure” of adaptive timesteppin, 1209
- “Low” and “high” frequencies, 865
- “Software solution” of interpolation problem, 298
- “Squeezed” DFT of a periodically truncated signal, 887
- $\mathbf{B} = \mathbf{B}^H$  s.p.d. mit Cholesky-Zerlegung, 589
- L-stable implicit Runge-Kutta methods, 1255
- `fft` Efficiency, 904
- $q_1$ -norm from eigenvalues, 520
- 3-Term recursion for Legendre polynomials, 1074

3-term recursion for Chebychev polynomials, 959

A posteriori error bound for linearly convergent iteration, 423

A posteriori termination criterion for linearly convergent iterations, 422

A posteriori termination criterion for plain CG, 535

Accessing rows and columns of sparse matrices, 184

Adapted Newton method, 453

Adaptive integrators for stiff problems in MATLAB, 1259

Adaptive quadrature in MATLAB, 1092

Adaptive timestepping for mechanical problem, 1215

Adding *EPS* to 1, 146

Affine invariance of Newton method, 473

Algorithm for cluster analysis, 742

Analytic solution of homogeneous linear ordinary differential equations, 581

Approximate computation of Fourier coefficients, 1059

Approximation

- convergence Cluster- with collocation matrix, 1130
- approximation uses of, 942

Approximation and quadrature, 1030

Approximation by polynomials, 943

Arnoldi process Ritz projection, 712

Asymptotic perspective in convergence analysis, 1168

auxiliary construction for shape preserving quadrature interpolation, 389

Banach's fixed point theorem, 434

bandwidth, 209

BLAS calling conventions, 70

Block Gaussian elimination, 105

Block LU-factorization, 122

Block matrix product, 52

Blow-up, 1192

Blow-up of explicit Euler method, 1222

Blow-up solutions of vibration equations, 658

Bound for asymptotic rate of linear convergence, 438

Broyden method for a large non-linear system, 499

Broydens Quasi-Newton method: convergence, 494

Butcher scheme for some explicit RK-SSM, 1185

Cancellation in decimal floating point arithmetic, 328

CG convergence and spectrum, 548

Changing entries/rows/columns of a matrix, 270

Characteristic parameters of IEEE floating point numbers, 143

Chebyshev interpolation error, 968

Chebyshev interpolation of analytic function, 975

Chebyshev interpolation of analytic functions, 973

Chebyshev polynomials on arbitrary interval, 965

Chebyshev representation of built-in functions, 983

Chebyshev vs uniform nodes, 966

Choice of quadrature weights, 1060

Choice of unitary/orthogonal transformation, 256

Class `PolyEval`, 334

Classification from measured data, 720

Cluster Tree, 1112

Complexity of Householder QR-factorization, 257

Computational effort for eigenvalue computations, 593

Computing Gauss nodes and weights, 1077

condition  
 extended system, 784

Conditioning and relative error, 169

Conditioning of normal equations, 780

Conditioning of the extended normal equations, 783

Conditioning of the least squares problem, 776

Constitutive relations from measurements, 290

Construction of simple Runge-Kutta methods, 118

Convergence der cluster approximation, 1129

Convergence of CG as iterative solver, 539

Convergence of clustering approximation with collocation matrix, 1130

Convergence of equidistant trapezoidal rule, 1052

- Convergence of Fourier sums, 897
- Convergence of gradient method, 516
- Convergence of Hermite interpolation, 1015
- Convergence of Hermite interpolation with exact slopes, 1012
- Convergence of Krylov subspace methods for non-symmetric system matrix, 571
- Convergence of Newton's method in 2D, 478
- Convergence of PINVIT, 655
- Convergence of simple Runge-Kutta methods, 1181
- Convergence of subspace variant of direct power method, 693
- Convergence rates for CG method, 546
- Convergence theory for PCG, 557
- Conversion into autonomous ODE, 1149
- Convolution of sequences, 835
- Cosine transforms for compression, 928
- CRS format, 182
- cubic Hermite interpolation, 310
- Damped Newton method, 490
- Data points confined to a subspace, 734
- Deblurring by DFT, 879
- Decay conditions for bi-infinite signals, 894
- Decimal floating point numbers, 140
- Details of Householder reflections, 248
- Detecting linear convergence, 411
- Detecting order of convergence, 415
- Detecting periodicity in data, 862
- Different implementations of matrix multiplication in MATLAB, 64
- Differentiation repetition, 474
- Direct power method, 618
- Divided differences and derivatives, 337
- Domain of definition of solutions of IVPs, 1153
- Efficiency of iterative methods, 468
- Efficiency of `fft`, 904
- Efficient associative matrix multiplication, 58
- Efficient evaluation of trigonometric interpolation polynomials, 993
- Efficient Initialization of sparse matrices in MATLAB, 187

- Eigenvalue computation with Arnoldi process, 716
- Eigenvectors of circulant matrices, 845
- Envelope of a matrix, 211
- Envelope oriented matrix storage, 218
- Error estimates for polynomial quadrature, 1036
- Error of Gauss quadrature, 1079
- Error of polynomial interpolation, 954
- Euler methods for stiff logistic IVP, 1249
- Explicit Euler method as difference scheme, 1160
- Explicit Euler method for damped oscillations, 1238
- Explicit integrator in MATLAB, 1187
- Explicit representation of error of polynomial interpolation, 952
- Explicit trapezoidal rule for decay equation, 1227
- Exploring convergence, 947
- Extended normal equations, 782
- Extremal properties of natural cubic spline interpolant, 376
- Failure of damped Newton method, 490
- Failure of Krylov iterative solvers, 570
- Fast evaluation of Chebychev expansion, 977
- Fast matrix multiplication, 56
- Fast Toeplitz solvers, 938
- Feasibility of implicit Euler timestepping, 1162
- FFT algorithm by matrix factorization, 909
- FFT based on general factorization, 912
- FFT for prime vector length, 914
- Filtering in Fourier domain, 900
- Finite linear time-invariant causal channel, 826
- Fit of hyperplanes, 797
- Fixed points in 1D, 431
- Fractional order of convergence of secant method, 461
- Frequency filtering by DFT, 868
- Frequency identification with DFT, 861
- Function representation, 292
- Gain through adaptivity, 1204
- Gaining efficiency through usage of BLAS, 71
- Gaussian elimination, 91
- Gaussian elimination and LU-factorization, 109
- Gaussian elimination for non-square matrices, 100

- Gaussian elimination via rank-1 modifications, 103
- Gaussian elimination with pivoting for  $3 \times 3$ -matrix, 127
- Generalized eigenvalue problems and Cholesky factorization, 589
- Generalized polynomial interpolation, 308
- Gibbs phenomenon, 998
- Global separable approximation by non-smooth kernel function, 1101
- Global separable approximation by smooth kernel function, 1100
- Gradient method in 2D, 513
- Gravitational forces in galaxy, 1095
- Group property of autonomous evolutions, 1155
- Growth with limited resources, 1137
- Halley's iteration, 450
- Heartbeat model, 1142
- Heating generation in electrical circuits, 1025
- Hermite interpolation
  - theorem, 953
- Horner scheme, 301
- IEEE standard
  - special cases, 142
- IEEE standard 754 for machine numbers, 142
- Image compression, 761
- Image segmentation, 623
- Impact of choice of norm, 410
- Impact of data access patterns on runtime, 36
- Impact of roundoff errors on CG, 537
- Impact of roundoff on Lanczos process, 705
- Implicit Euler timestepping for decay equation, 1248
- Importance of numerical quadrature, 1024
- In-situ LU-decomposition, 118
- Initial guess for power iteration, 621
- Input errors and rounding errors, 144
- Instability of multiplication with inverse, 164
- Instability of normal equations, 781
- Interaction calculations for many body systems, 1094
- interpolation
  - cubic spline- locality, 382
  - piecewise cubic monotonicity preserving, 364

- shape preserving quadratic spline, 393
- Interpolation as linear mapping, 296
- interpolation error, 945
- Interpolation error: trigonometric interpolation, 996
- Intersection of lines in 2D, 173
- Justification of Ritz projection by min-max theorem, 686
- Keeping track of unitary transformations, 260
- Kinetics of chemical reactions, 1240
- Krylov methods for complex s.p.d. system matrices, 504
- Krylov subspace methods for generalized EVP, 718
- Lanczos process for eigenvalue computation, 704
- Least squares data fitting, 764
- Lebesgue constant for equidistant nodes, 341
- Linear data fitting, 765
- Linear filtering of periodic signals, 835
- linear regression, 770
- Linear regression for stationary Markov chains, 931
- Lineare zeitinvariante Systeme, 929
- Linearization of increment equations, 1257
- Linearly convergent iteration, 412
- Local convergence of Newton's method, 486
- local convergence of secant method, 462
- Loss of sparsity when forming normal equations, 782
- Machine precision for MATLAB, 146
- Magnetization curves, 346
- Many sequential solutions of LSE, 121
- Matrix algebra, 52
- Matrix storage formats, 35
- Meaningful " $O$ -bounds" for complexity, 55
- Midpoint rule, 1032
- Min-max theorem, 635
- Multidimensional fixed point iteration, 438
- Multiplication of polynomials, 832
- Multiplication of sparse matrices, 191
- Necessary condition for L-stability, 1254



- Necessity of iterative approximation, 405
- Newton method and minimization of quadratic functional, 814
- Newton method in 1D, 448
- Newton method, modified , 457
- Newton-Cotes formulas, 1032
- Nodal analysis of (linear) electric circuit, 176
- Non-linear data fitting, 810
- Non-linear data fitting (II), 817
- Non-linear electric circuit, 401
- Non-linear interpolation, 365
- Normal equations vs. orthogonal transformations method, 797
- Notation for single step methods, 1167
- numerical differentiation through extrapolation, 321
- Numerical integration of logistic ODE in MATLAB, 1188
- Numerical summation of Fourier series, 895
- Occurrence of clusters in partition rectangles, 1124
- Options for fixed point iterations, 427
- Orders of simple polynomials quadrature formulas, 1049
- Oregonator reaction, 1189
- Origin of the term “Spline”, 379
- oscillating interpolation polynomial, 338
- Overdetermined linear systems, 771
- Page rank algorithm, 599
- PCA for data classification, 735
- PCA of stock prices, 748
- piecewise cubic Hermite interpolation, 359
- Piecewise cubic interpolation schemes, 376
- Piecewise linear interpolation, 293
- Piecewise polynomial interpolation, 1006
- Piecewise quadratic interpolation, 353
- Pivoting and numerical stability, 124
- Pivoting destroys sparsity, 207
- Polybomial interpolation vs. polynomial fitting, 767
- Polynomial
  - trigonometric analysis, 1133
- Polynomial evaluation: timing, 316
- Polynomial fitting, 767
- Polynomials in Matlab, 301

- Power iteration, 613
- Power iteration with Ritz projection, 688
- Predator-prey model, 1140
- Principal component analysis, 721
- Pseudoinverse, 776
- Pseudoinverse and SVD, 796
- QR
  - Orthogonalisierung, 259
  - qr based orthogonalization, 682
  - QR-Algorithm, 590
  - QR-based solution of tridiagonal LSE, 264
- Quadratic convergence], 416
- Quadratic functional in 2D, 505
- quadratic inverse interpolation, 465
- Quadratur
  - Gauss-Legendre Ordnung 4, 1064
- Quadrature errors for composite quadrature rules, 1042
- Quality measure for kernel approximation, 1098
- Radiative heat transfer, 838
- Rank defect in linear least squares problems, 775
- Rationale for adaptive quadrature, 1085
- Rationale for high-order single step methods, 1177
- Rationale for partial pivoting policy, 132
- Rayleigh quotient iteration, 650
- Reading off polynomials complexity, 61
- Recursive LU-factorization, 119
- Reducing fill-in by reordering, 221
- reduction to periodic convolution, 842
- Refined local stepsize control, 1211
- Region of convergence of Newton method, 486
- Relevance of asymptotic complexity, 62
- Removing a singularity by transformation, 1051
- Resistance to currents map, 268
- Resonances of linear electrical circuits, 574
- Restarted GMRES, 568
- Ritz projections onto Krylov space, 697
- Roundoff errors and difference quotients, 325
- Row and column transformations, 51
- Row swapping commutes with forward elimination, 136

- Row-wise & column-wise view of matrix product, 43
- Runge's example, 948, 955
- Runtime comparison for computation of coefficient of trigonometric interpolation polynomials, 991
- Runtime of Gaussian elimination, 97
- Runtimes of `eig`, 593
- S.p.d. Hessians, 231
- S.p.d. matrices from nodal analysis, 224
- Scalings, 48
- secant method, 459
- Sensitivity of linear mappings, 240
- Shape preservation, 381
- Shifted inverse iteration, 647
- Silly MATLAB, 193
- Simple adaptive stepsize control, 1200
- Simple adaptive timestepping for fast decay, 1225
- Simple composite polynomial quadrature rules, 1039
- Simple preconditioners, 557
- Simplified Newton method, 476
- Small residuals by Gaussian elimination, 161
- Solving LSE in the case of rank-1-modification, 272
- Sound filtering by DFT, 869
- Sparse  $LU$ -factors, 198
- Spectrum of Fourier matrix, 854
- Speed of convergence of explicit Euler method, 1168
- spline
  - interpolants, approx. complete cubic, 1019
  - natural cubic, locality, 382
  - shape preserving quadratic interpolation, 393
- Square root of a s.p.d. matrix, 551
- Stability by small random perturbations, 158
- Stability of Arnoldi process, 714
- Stepsize control detects instability, 1231
- Stepsize control in MATLAB, 1214
- Storing the  $Q$ -factor, 266
- Strongly attractive limit cycle, 1243
- Subspace power iteration with orthogonal projection, 674

- Subspace power methods, 694
- SVD and additive rank-1 decomposition, 727
- Tangent field and solution curves, 1157
- Taylor approximation, 941
- Tensor product Chebyshev interpolation for variable rectangle sizes, 1107
- Tensor product Chebyshev interpolation on rectangles, 1106
- Termination criterion for contractive fixed point iteration, 442
- Termination criterion for direct power iteration, 622
- Termination criterion in `pcg`, 564
- Termination of PCG, 562
- Testing for near singularity of a matrix, 267
- Transformation of quadrature rules, 1027
- Transient circuit simulation, 1145
- Transient simulation of RLC-circuit, 1232
- Trend analysis, 719
- Tridiagonal preconditioner, 559
- Trigonometric interpolation of analytic functions, 999
- Understanding the structure of product matrices, 43
- Uniqueness of SVD, 728
- Unitary similarity transformation to tridiagonal form 591
- Vibrations of a truss structure, 660
- Visualization of explicit Euler method, 1159
- Why using  $\mathbb{K} = \mathbb{C}$ ?, 850
- Wilkinson's counterexample, 157, 169
- Zerlegung
  - Teil-LU, 122

# German terms

$L^2$ -inner product =  $L^2$ -Skalarprodukt, 1066

], 92

bandwidth = Bandbreite, 209

cancellation = Auslöschung, 328

capacitance = Kapazität, 177

capacitor = Kondensator, 177

circulant matrix = zirkulante Matrix, 841

coil = Spule, 177

column (of a matrix) = (Matrix)spalte, 32

column transformation = Spaltenumformung, 51

column vector = Spaltenvektor, 30

composite quadrature formulas = zusammengesetzte Quadraturformeln, 1037

computational effort = Rechenaufwand, 54

consistency = Konsistenz, 406

constitutive relation = Kennlinie, 290

constitutive relations = Bauelementgleichungen,  
177

constrained least squares = Ausgleichsproblem  
mit Nebenbedingungen, 804

convergence = Konvergenz, 406

convolution = Faltung, 826

damped Newton method = gedämpftes Newton-  
Verfahren, 485

dense matrix = vollbesetzte Matrix, 175

descent methods = Abstiegsverfahren, 503

divided differences = dividierte Differenzen, 332

dot product = (Euklidisches) Skalarprodukt, 41

- eigenspace = Eigenraum, 583
- eigenvalue = Eigenwert, 583
- electric circuit = elektrischer Schaltkreis/Netzwerk, 176
- energy norm = Energienorm, 503
- envelope = Hülle, 211
- extended normal equations = erweiterte Normalengleichungen, 782
- fill-in = "fill-in", 198
- fixed point iteration = Fixpunktiteration, 425
- floating point number = Gleitpunktzahl, 141
- forward elimination = Vorwärtselimination, 91
- Fourier series = Fourierreihe, 893
- Gaussian elimination = Gausselimination, 89
- high pass filter = Hochpass, 867
- identity matrix = Einheitsmatrix, 34
- image segmentation = Bildsegmentierung, 623
- impulse response = Impulsantwort, 827
- in situ = am Ort, 118
- in situ = an Ort und Stelle, 104
- initial guess = Anfangsnäherung, 406
- inverse iteration = inverse Iteration, 623
- Kirchhoff (current) law = Kirchhoffsche Knotenregel, 177
- knot = Knoten, 367
- Krylov space = Krylovraum, 526
- least squares = Methode der kleinsten Quadrate, 764
- line search = Minimierung in eine Richtung, 508
- linear system of equations = lineares Gleichungssystem, 86
- low pass filter = Tiefpassfilter, 867
- lower triangular matrix = untere Dreiecksmatrix, 112
- LU-factorization = LR-Zerlegung, 106
- machine number = Maschinenzahl, 141
- mass matrix = Massenmatrix, 665
- matrix factorization = Matrixzerlegung, 106
- mesh width = Gitterweite, 1003
- mesh/grid = Gitter, 1002

multiplicity= Vielfachheit, 583

nodal analysis = Knotenanalyse, 176

normal equations = Normalengleichungen, 779

order of convergence = Konvergenzordnung, 413

ordinary differential equation = gewöhnliche Differentialgleichung, 1147

partial pivoting = Spaltenpivotsuche, 130

pattern (of a matrix) = Besetzungsmuster, 48

power method = Potenzmethode, 599

preconditioning = Vorkonditionierung, 550

predator-prey model = Räuber-Beute-Modell, 1140

principal axis transformation = Hauptachsentransformation, 587

principal component analysis = Hauptkomponentenanalyse, 721

quadratic functional = quadratisches Funktional, 504

quadrature node = Quadraturknoten, 1026

quadrature weight = Quadraturgewicht, 1026

resistor = Widerstand, 177

right hand side vector = rechte-Seite-Vektor, 86

rounding = Rundung, 145

row (of a matrix) = (Matrix)zeile, 32

row transformation = Zeilenumformung, 51

row vector = Zeilenvektor, 30

saddle point problem = Sattelpunktproblem, 806

scaling = Skalierung, 48

singular value decomposition = Singulärwertzerlegung, 719

sparse matrix = dünnbesezte Matrix, 175

speed of convergence = Konvergenzgeschwindigkeit, 409

steepest descent = steilster Abstieg, 508

stiffness matrix = Steifigkeitsmatrix, 665

storage format = Speicherformat, 35

subspace correction = Unterraumkorrektur, 523

tensor product = Tensorprodukt, 41

tent/hat function = Hutfunktion, 294

termination criterion = Abbruchbedingung, 418

timestepping = Zeitdiskretisierung, 1158

total least squares = totales Ausgleichsproblem,  
801

transpose = transponieren/Transponierte, 30

truss = Stabwerk, 660

unit vector = Einheitsvektor, 31

upper triangular matrix = obere Dreiecksmatrix,  
112

variational calculus = Variationsrechnung, 377

zero padding = Ergänzen durch Null, 994