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 38355

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

Always under construction

Contents

R. Hiptmair

rev 38355, November 5, 2011

About this course

Focus

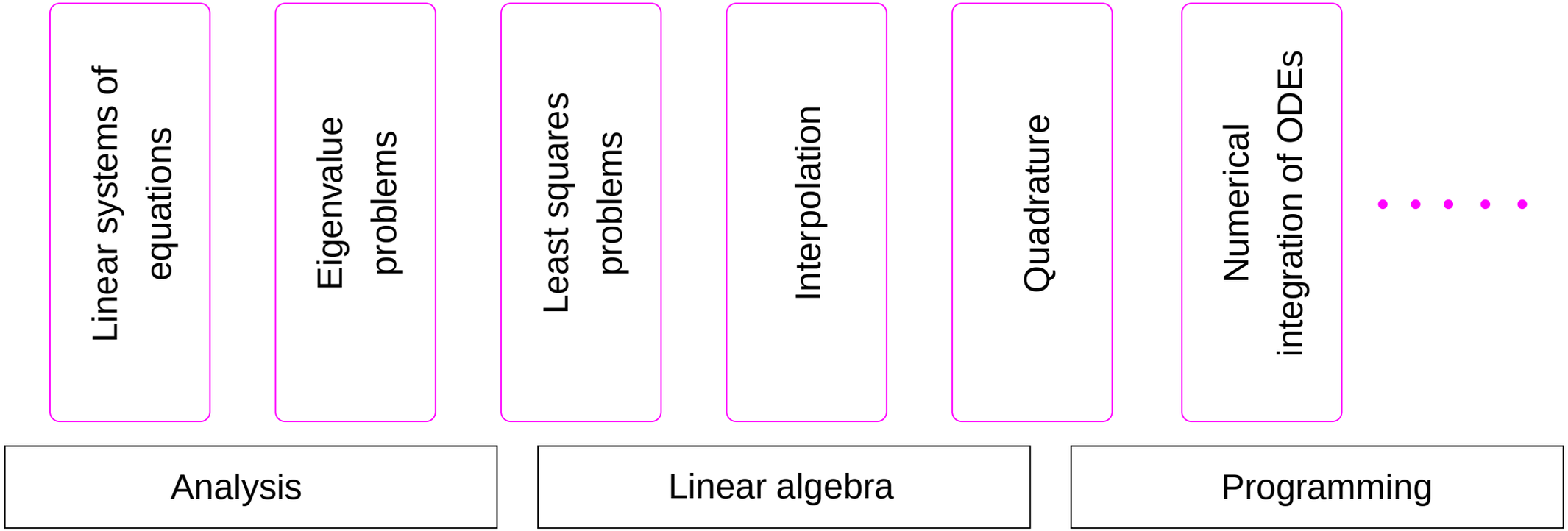
- ▷ on algorithms (principles, scope, and limitations),
- ▷ on (efficient, stable) implementation in MATLAB,
- ▷ on numerical experiments (design and interpretation).

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)

R. Hiptmair

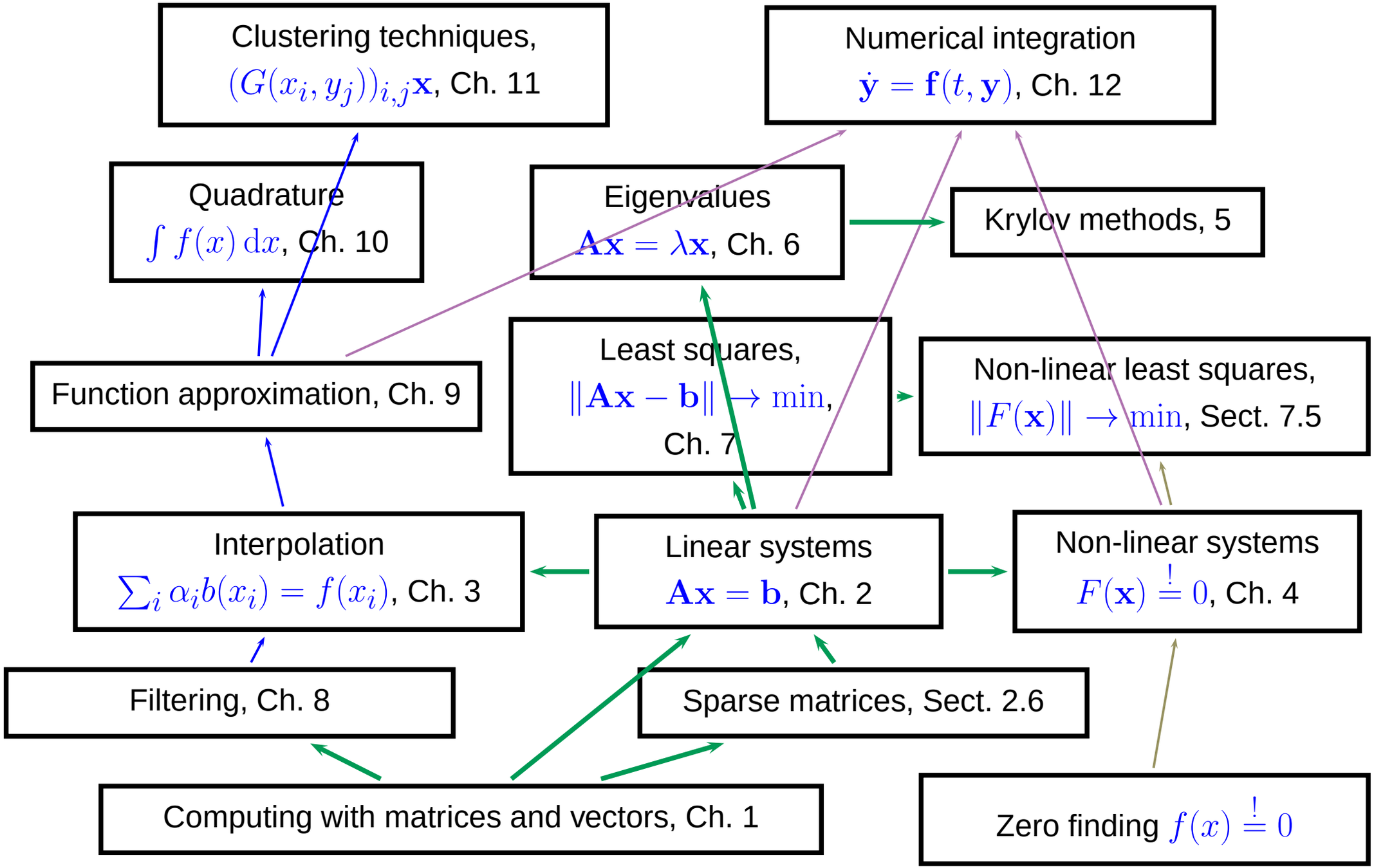
rev 38355, October 21, 2011

Contents



R. Hiptmair

rev 38355, October 21, 2011



R. Hiptmair

rev 38355, October 21, 2011

Relevance

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

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

interpolation
least squares
} ▶ Computer graphics

eigensolvers
sparse linear systems
} ▶ Graph theoretic algorithms

numerical integration } ▶ Computer animation

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

R. Hiptmair

rev 38355, October 21, 2011

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 examnination after the end of the course.

Perseverance will be rewarded!

Books

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

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)

To reach the desks in front of HG J 54, please take an elevator near Polyterrasse.

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`

Online homework submission (optional)

You can use the online homework submission tool:
`http://www.math.ethz.ch/~grsam/submit/?VL=02`

Examination

- 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 !

`http://elbanet.ethz.ch/wikifarm/rhiptmair/index.php?n=Main.NCSECourse`

(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`, Subject: 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)

10: My prior knowledge of MATLAB was insufficient for the course (HS 10: 31-29-13-22-5, 2.4)

11: More formal proofs would be desirable (HS 10: 56-19-15-7-4, 1.9)

12: The explanations on the blackboard promote understanding (HS 10: 13-36-25-20-5, 2.7)

13: The codes included in the lecture material convey useful information (HS 10: 5-20-18-25-29, 3.5)

14: The master solutions for exercise problems offer too little guidance. (HS 10: 5-13-22-15-13, 3.2)

15: 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.

Evaluation results: Fall semester 2009
Fall semester 2010

MATLAB

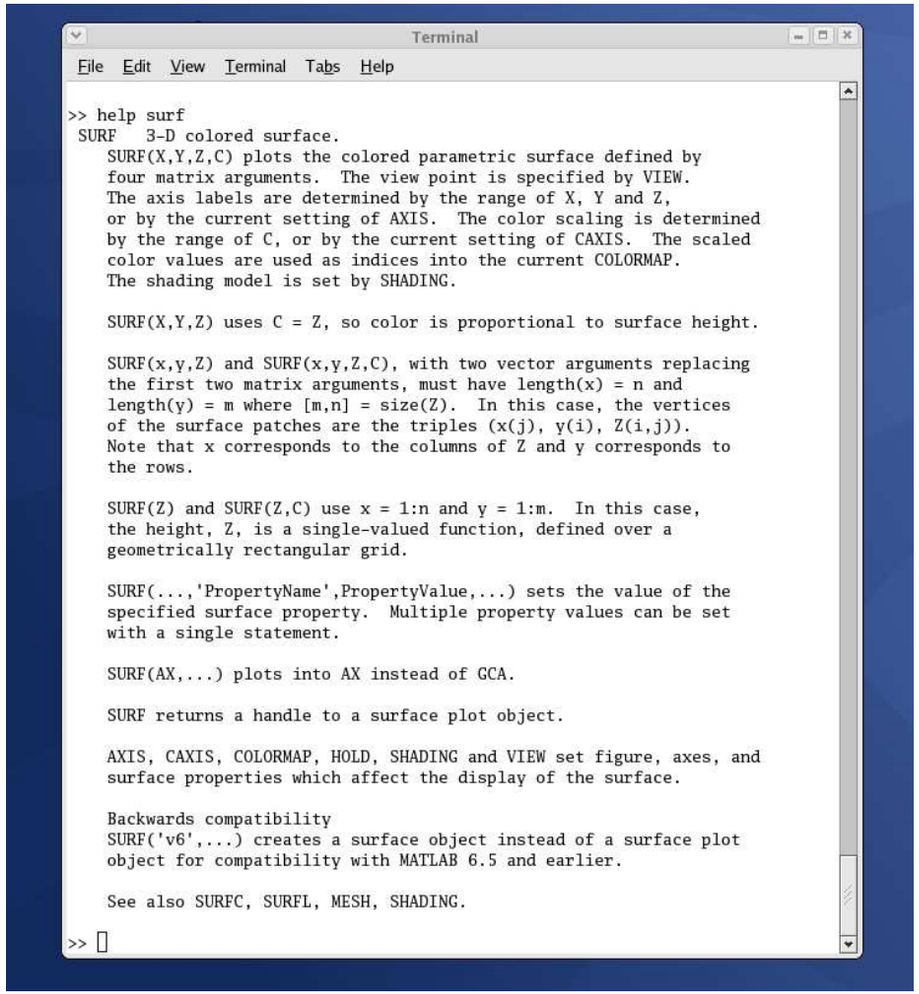
MATLAB ("“Matrix Laboratory”):

- 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)



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 Gilbe
- Numerical algorithms
- NUMAWWW

1 Computing with Matrices and Vectors [48, Sect. 2.1&2.2], [27, Sect. 2.1&2.2]

1.1 Notations

1.1.1 Vectors

1.1.2 Matrices

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

R. Hiptmair

rev 38355, February 24, 2009

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

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

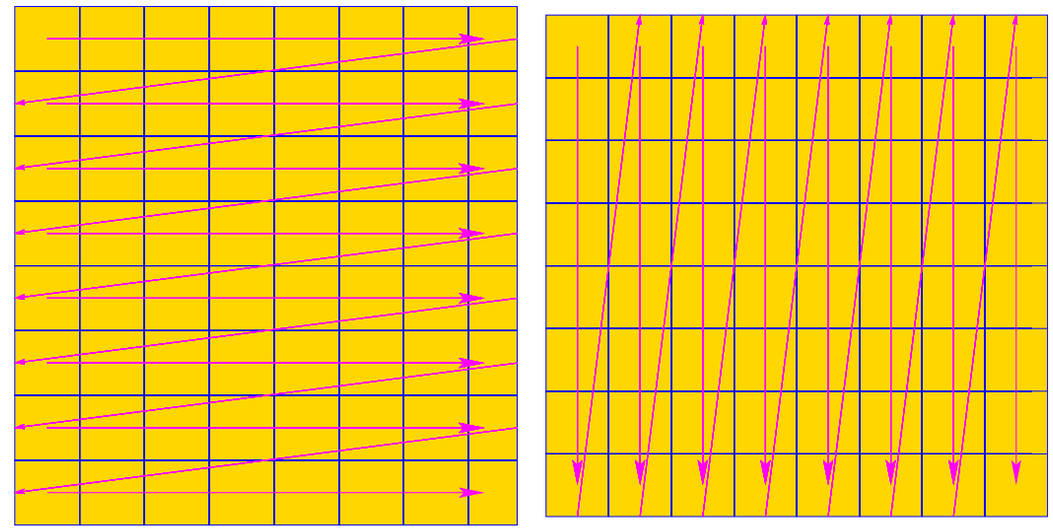
Access to entry a_{ij} of $\mathbf{A} \in \mathbb{K}^{n,m}$, $i = 1, \dots, n$, $j = 1, \dots, m$:

row major:

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

column major:

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



row major column major

△

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

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

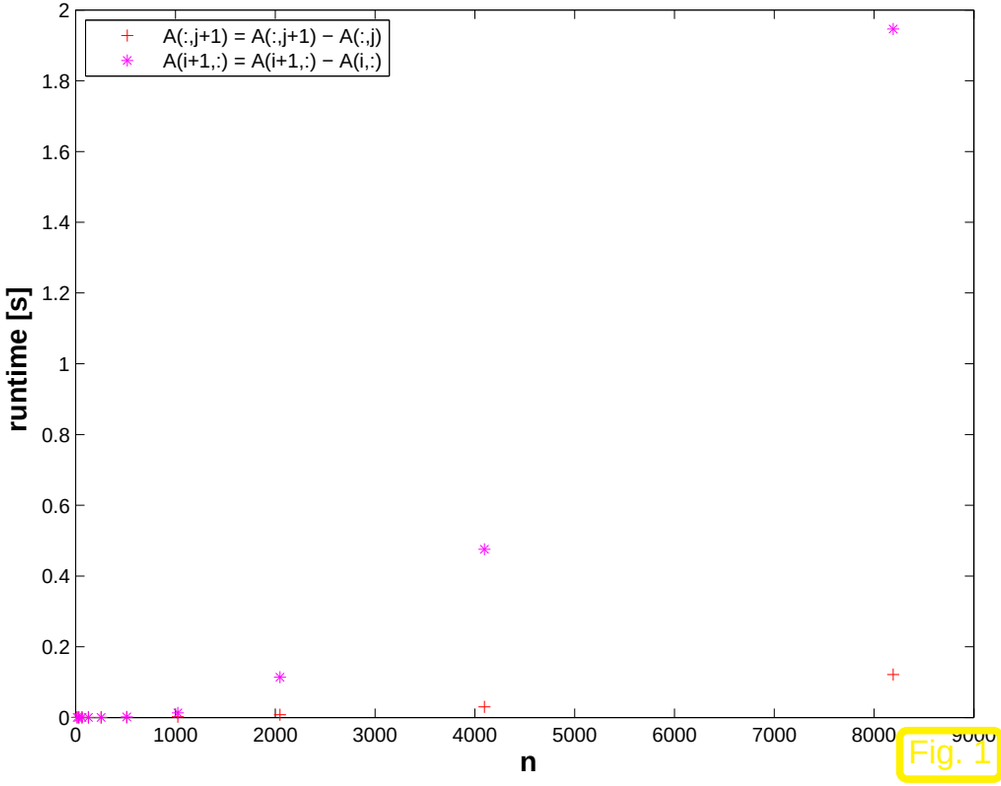
Code 1.1.3: timing for row and column oriented matrix access in MATLAB

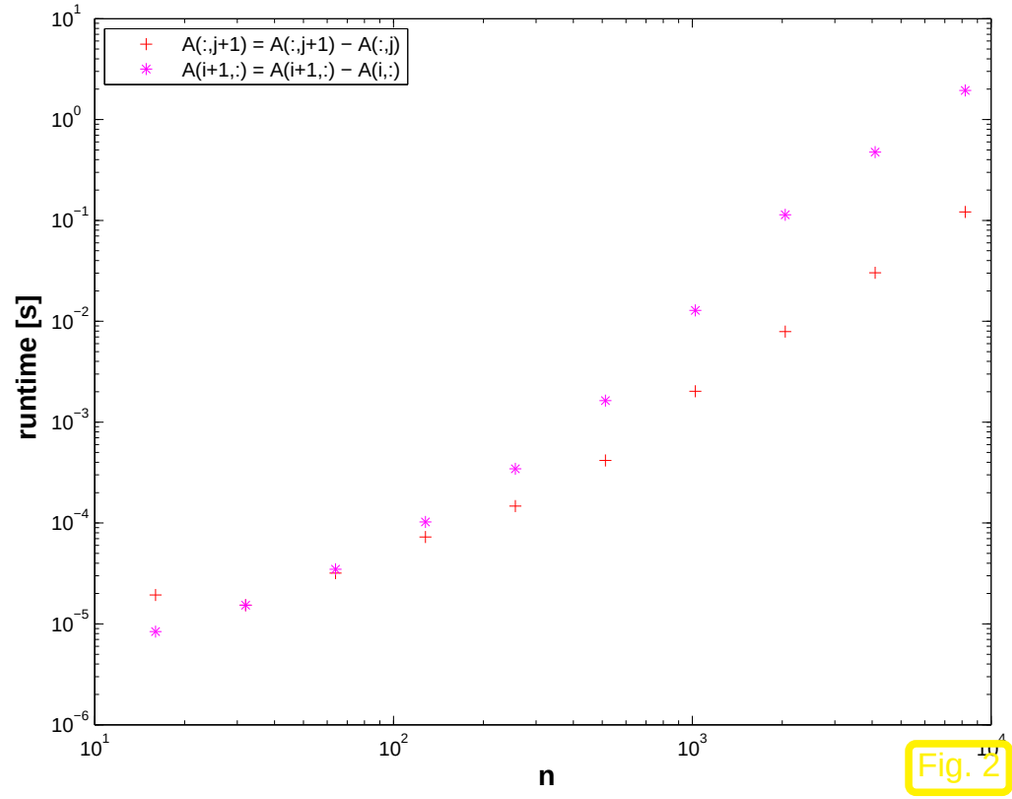
```
% Timing for row/column operations on matrices
% We conduct K runs in order to reduce the risk of skewed maesurements
% due to OS activity during MATLAB run.
K = 3; res = [];
for n=2.^(4:13)
  A = randn(n,n);

  t1 = realmax;
  for k=1:K,  tic;
    for j = 1:n-1,  A(:,j+1) = A(:,j+1) - A(:,j); end;
    t1 = min(toc,t1);
  end
  t2 = realmax;
  for k=1:K,  tic;
    for i = 1:n-1,  A(i+1,:) = A(i+1,:) - A(i,:); end;
    t2 = min(toc,t2);
  end
  res = [res; n, t1 , t2];
end

% Plot runtimes versus matrix sizes
figure; plot(res(:,1),res(:,2),'r+', res(:,1),res(:,3),'m*');
xlabel('{\bf n}','fontsize',14);
```


R. Hiptmair

rev 38355, February 24, 2009

```
ylabel('{\bf runtime [s]}','fontsize',14);
legend('A(:,j+1) = A(:,j+1) - A(:,j)','A(i+1,:) = A(i+1,:) -
  A(i,:)',...
       'location','northwest');
print -depsc2 '../PICTURES/accessrtlin.eps';

figure; loglog(res(:,1),res(:,2),'r+', res(:,1),res(:,3),'m*');
xlabel('{\bf n}','fontsize',14);
ylabel('{\bf runtime [s]}','fontsize',14);
legend('A(:,j+1) = A(:,j+1) - A(:,j)','A(i+1,:) = A(i+1,:) -
  A(i,:)',...
       'location','northwest');
print -depsc2 '../PICTURES/accessrtlog.eps';
```





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

◇

1.2 Elementary operations

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

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

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

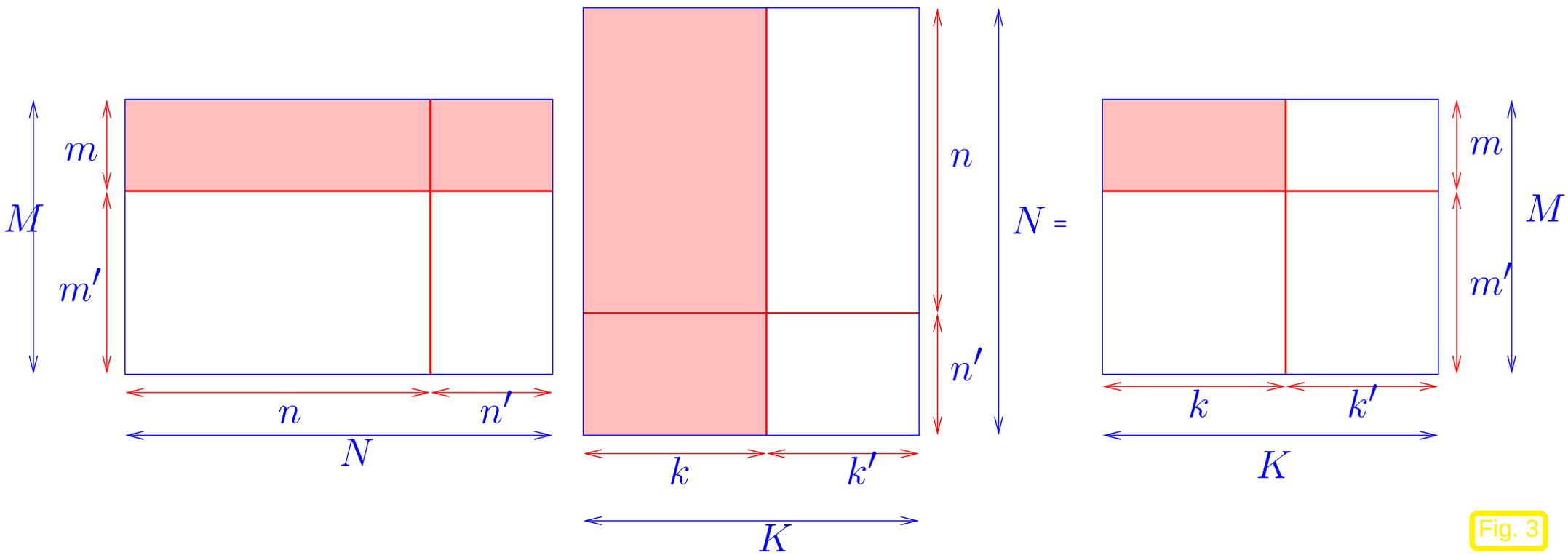


Fig. 3

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

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

Example 1.3.3 (Efficient associative matrix multiplication).

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

$$\mathbf{y} = \left(\mathbf{a}\mathbf{b}^\top\right)\mathbf{x} \,. \tag{1.3.4}$$

```
T = a*b'; y = T*x;
```

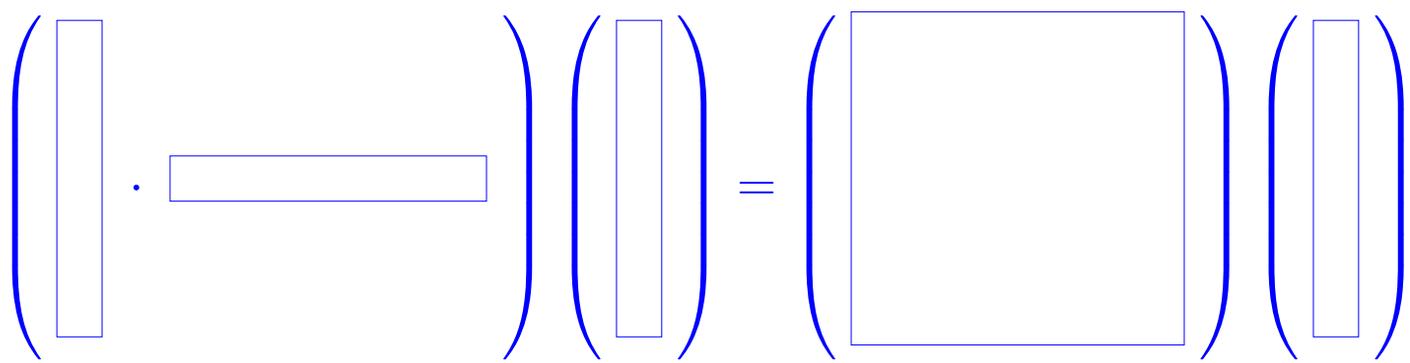
➤ complexity $O(mn)$

$$\mathbf{y} = \mathbf{a}\left(\mathbf{b}^\top\mathbf{x}\right) \,. \tag{1.3.5}$$

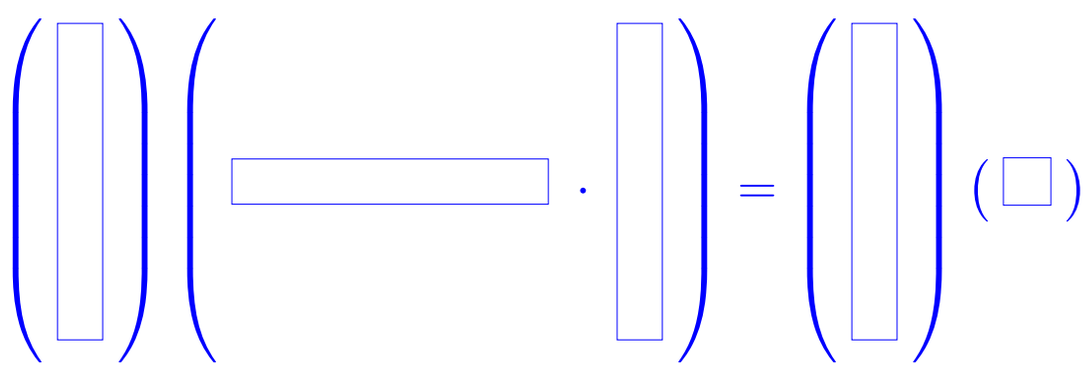
```
t = b'*x; y = a*t;
```

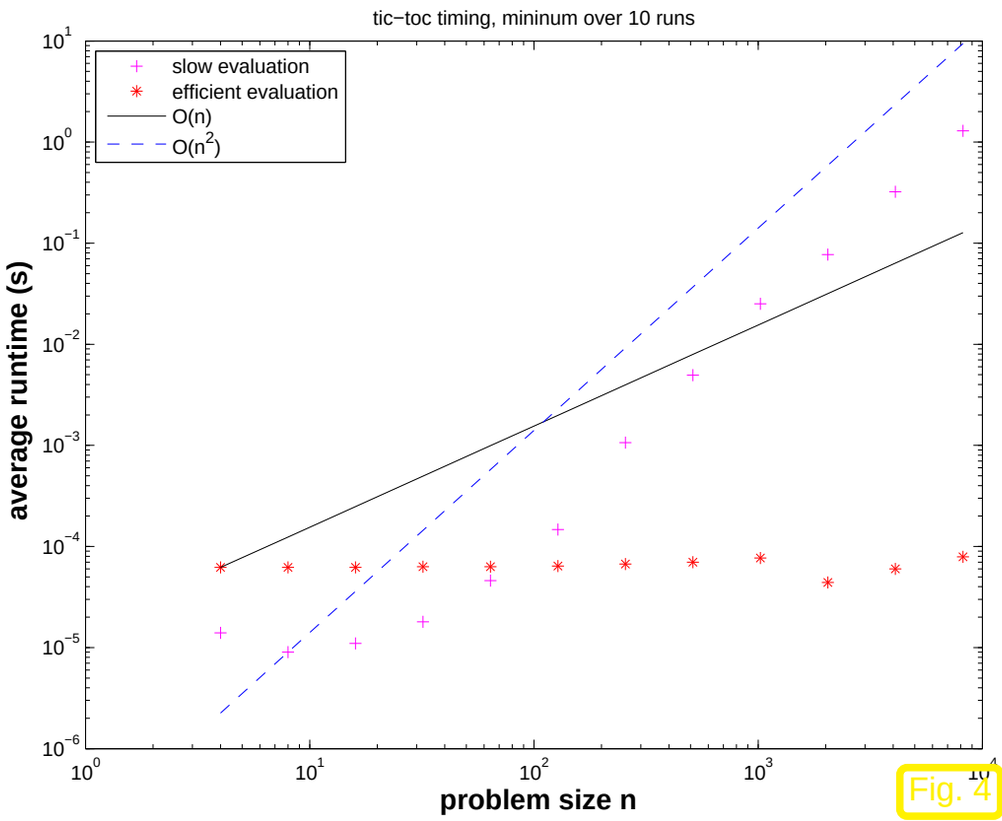
➤ complexity $O(n+m)$ ("linear complexity")

Visualization of evaluation according to (1.3.4):



Visualization of evaluation according to (1.3.4):





◁ 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

◇

1.4 BLAS

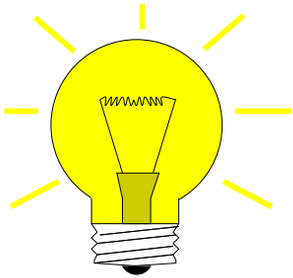
2 Direct Methods for Linear Systems of Equations

2.1 Gaussian Elimination

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

☞ "exact" solution computable with finitely many elementary operations

Algorithm: Gaussian elimination ($\rightarrow$ secondary school, linear algebra,)



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

R. Hiptmair

rev 38355, October 30, 2011

Algorithm 2.1.3.

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

Code 2.1.4: Solving LSE $\mathbf{Ax} = \mathbf{b}$ with Gaussian elimination

```
function x = gausselimsolve(A,b)
% Gauss elimination without pivoting, x = A\b
% A must be an n×n-matrix, b an n-vector
n = size(A,1); A = [A,b]; %
% Forward elimination (cf.  step ① in Ex. 2.1.1)
for i=1:n-1, pivot = A(i,i);
  for k=i+1:n, fac = A(k,i)/pivot;
      A(k,i+1:n+1) = A(k,i+1:n+1) -
        fac*A(i,i+1:n+1); %
  end
end
% Back substitution (cf.  step ② in Ex. 2.1.1)
A(n,n+1) = A(n,n+1) /A(n,n);
for i=n-1:-1:1
    for l=i+1:n
        A(i,n+1) = A(i,n+1) -
          A(l,n+1)*A(i,l);
    end
    A(i,n+1) = A(i,n+1)/A(i,i);
end
x = A(:,n+1); %
```

R. Hiptmair

rev 38355, October 30, 2011

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

$$\begin{aligned} \text{elimination}: &\quad \sum_{i=1}^{n-1}(n-i)(2(n-i)+3) = n(n-1)(\tfrac{2}{3}n+\tfrac{7}{6}) \text{ Ops.},\\ \text{back substitution}: &\quad \sum_{i=1}^{n} 2(n-i)+1 = n^2 \text{ Ops.}. \end{aligned} \tag{2.1.5}$$

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).

MATLAB \

▷ based on LAPACK

▷ based on BLAS ($\rightarrow$ Sect. 1.4)

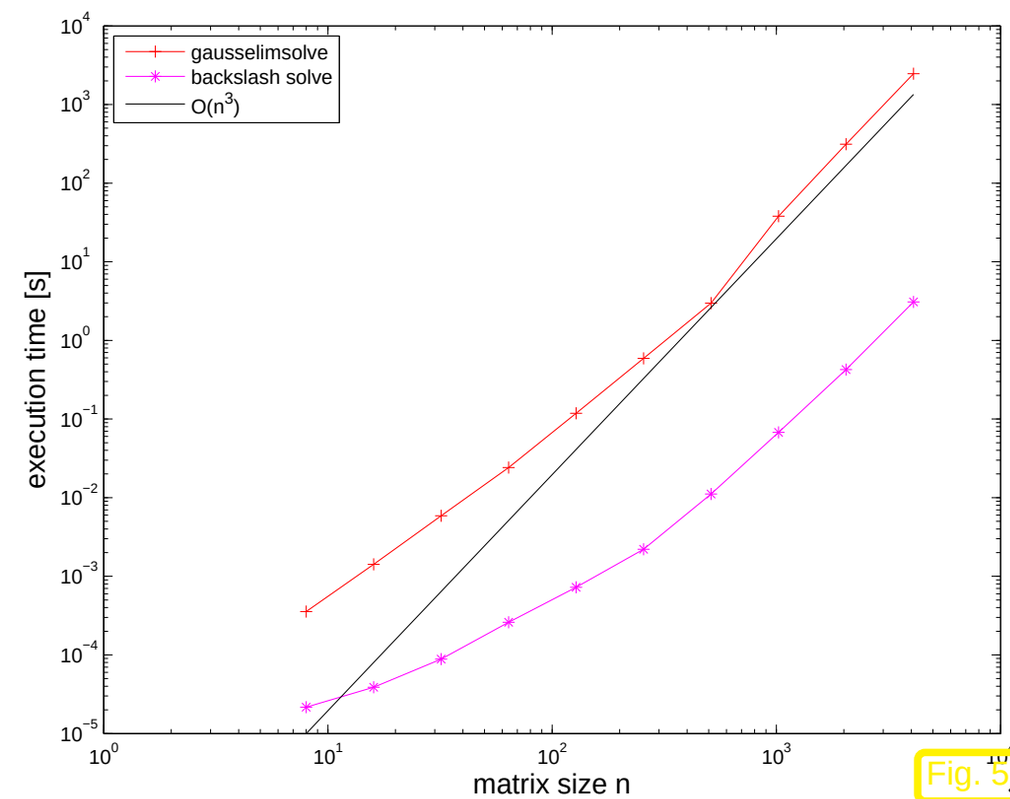


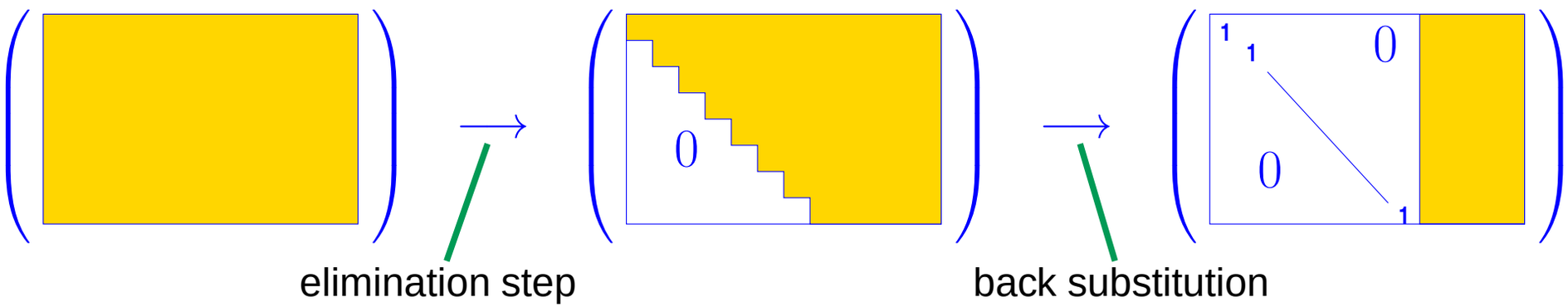
Fig. 5 ◇

Never implement Gaussian elimination yourself **!**

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

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

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



R. Hiptmair

rev 38355, October 30, 2011

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 $\mathbf{X} \in \mathbb{K}^{n,k}$

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

MATLAB:

```
X = A\B;
```

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

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

```
function X = gausselimsolvemult(A,B)
% Gauss elimination without pivoting, X = A\B
n = size(A,1); m = n + size(B,2); A = [A,B];
for i=1:n-1, pivot = A(i,i);
  for k=i+1:n, fac = A(k,i)/pivot;
      A(k,i+1:m) = A(k,i+1:m) - fac*A(i,i+1:m);
  end
end
A(n,n+1:m) = A(n,n+1:m) /A(n,n);
for i=n-1:-1:1
    for l=i+1:n
        A(i,n+1:m) = A(i,n+1:m) - A(l,n+1:m)*A(i,l);
    end
    A(i,n+1:m) = A(i,n+1:m)/A(i,i);
end
X = A(:,n+1:m);
```

△

R. Hiptmair

rev 38355, September 26, 2011

2.2 LU-Decomposition/LU-Factorization

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:

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

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

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

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

$\diamond$

R. Hiptmair

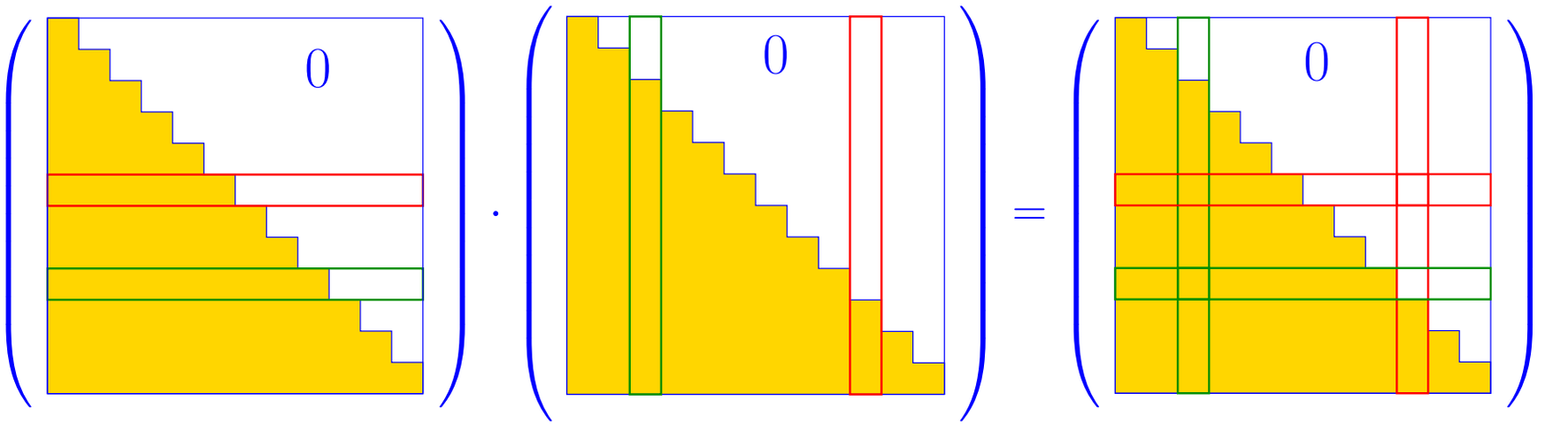
rev 38355, September 26, 2011

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

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

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

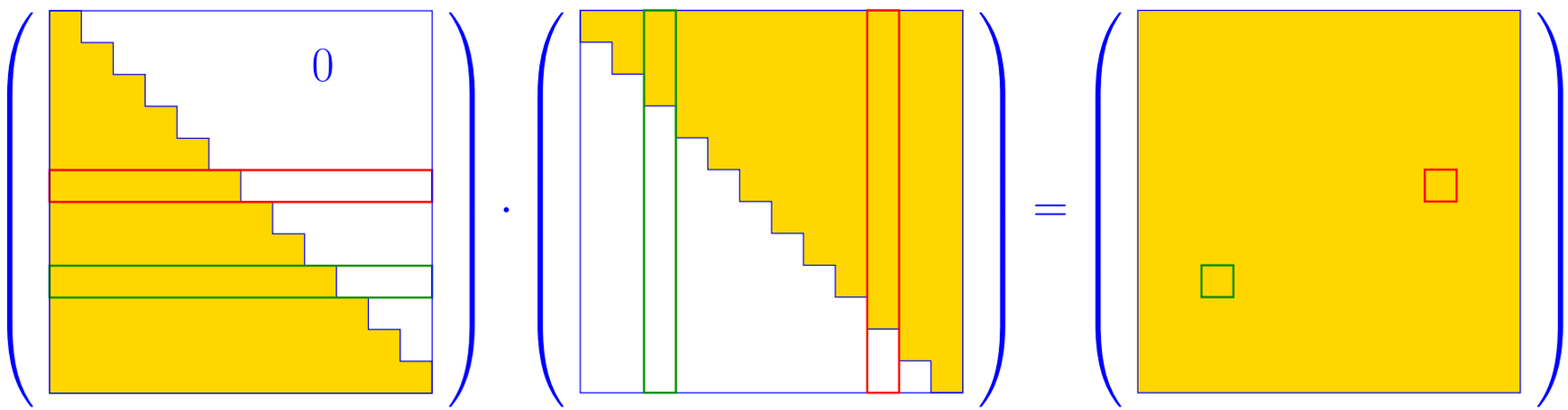
"Proof by visualization" $\rightarrow$ Rem. 1.2.4



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].

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.

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



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

➤ • row by row computation of $\mathbf{U}$

• column by column computation of $\mathbf{L}$

Entries of $\mathbf{A}$ can be replaced with those of $\mathbf{L}$, $\mathbf{U}$!
(so-called in situ/in place computation)

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

$\hat{=}$ rows of $\mathbf{U}$

$\hat{=}$ columns of $\mathbf{L}$

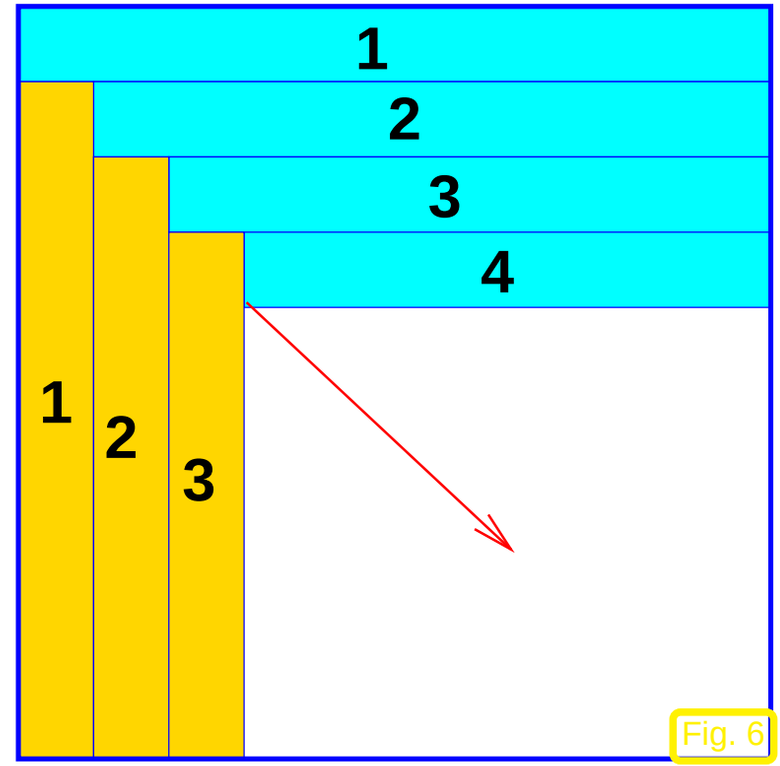
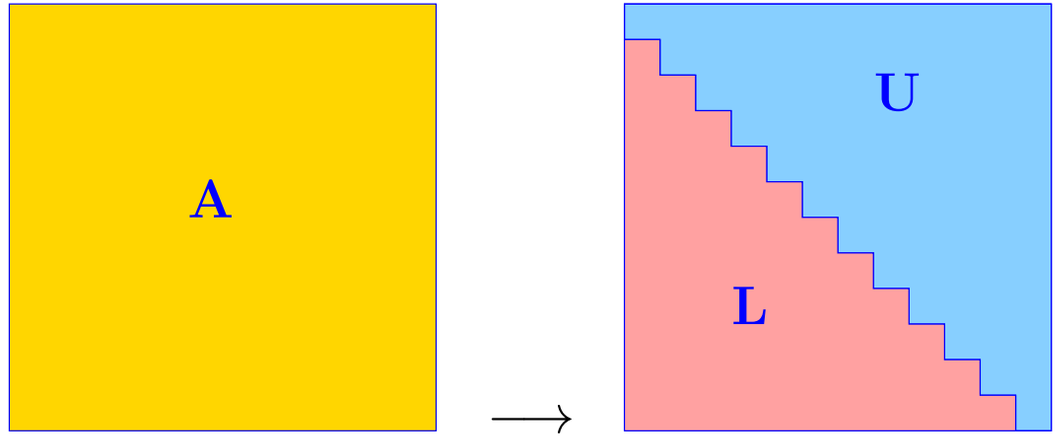


Fig. 6

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

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

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 !

```
% Setting:  N ≫ 1, large matrix A
for j=1:N
    x = A\b;
    b = some_function(x);
end
```

computational effort $O(Nn^3)$

smart !

```
% Setting:  N ≫ 1, large matrix A
[L,U] = lu(A);
for j=1:N
    x = U\(L\b);
    b = some_function(x);
end
```

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

△

Remark 2.2.16 (Block LU-factorization).

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

Schur complement

$$\underbrace{\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & 0 \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{S} \end{pmatrix}}_{\text{block LU-factorization}} , \quad \boxed{\mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}} . \tag{2.2.17}$$

$\rightarrow$ block Gaussian elimination, see Rem. 2.1.12.

△

2.3 Pivoting

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

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

breakdown of Gaussian elimination pivot element $= 0$ — Gaussian elimination feasible

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

Example 2.3.1 (Pivoting and numerical stability). $\rightarrow$ [13, Bsp. 3.2.3]

R. Hiptmair

rev 38355, October 10, 2011

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

Ouput of MATLAB run:

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

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

What is wrong with MATLAB? Needed: insight into roundoff errors $\rightarrow$ Sect. 2.4

$\diamond$

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

Algorithm 2.3.5 (Gaussian elimination with partial pivoting).

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

```
function x = gepiv(A,b)
% Solving an LSE Ax = b by Gaussian elimination with partial pivoting
n = size(A,1); A = [A,b]; %
% Forward elimination by rank-1 modification, see Rem. 2.1.10
for k=1:n-1
   [p,j] = max(abs(A(k:n,k))./max(abs(A(k:n,k:n))')') %
   if (p < eps*norm(A(k:n,k:n),1)), %
     disp('A nearly singular'); end
   A([k,j+k-1],k:n+1) = A([j+k-1,k],k:n+1);                    %
   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); %
end
% Back substitution (same as in Code 2.1.6)
A(n,n+1) = A(n,n+1) /A(n,n);
for i=n-1:-1:1
  A(i,n+1) = (A(i,n+1) - A(i,i+1:n)*A(i+1:n,n+1))/A(i,i);
end
x = A(:,n+1); %
```


R. Hiptmair

rev 38355, October 10, 2011

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

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

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

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

```
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

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* $\mathbf{PA} = \mathbf{LU}$.

Example 2.3.14 (Ex. 2.3.4 cnt'd).

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 2 \\ 2 & -3 & 2 \\ 1 & 24 & 0 \end{pmatrix} \overset{❶}{\rightarrow} \begin{pmatrix} 2 & -3 & 2 \\ 1 & 2 & 2 \\ 1 & 24 & 0 \end{pmatrix} \overset{❷}{\rightarrow} \begin{pmatrix} 2 & -3 & 2 \\ 0 & 3.5 & 1 \\ 0 & 25.5 & -1 \end{pmatrix} \overset{❸}{\rightarrow} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 25.5 & -1 \\ 0 & 3.5 & 1 \end{pmatrix} \overset{❹}{\rightarrow} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 25.5 & -1 \\ 0 & 0 & 1.373 \end{pmatrix}$$

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

$\diamond$

MATLAB function: `[L,U,P] = lu(A)` (P = permutation matrix)

Remark 2.3.15 (Row swapping commutes with forward elimination).

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.

$\triangle$

2.4 Supplement: Machine Arithmetic

Computer = finite automaton $\rightarrow$ can handle only *finitely many* numbers, not $\mathbb{R}$

machine numbers, set $\mathbb{M}$

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

Example 2.4.6 (Input errors and roundoff errors).

Code 2.4.7: input errors and roundoff errors

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

R. Hiptmair

rev 38355, October 14, 2011

◇

Notation: floating point realization of $\star \in \{+, -, \cdot, /\}$: $\widetilde{\star}$

correct rounding:

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

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

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

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

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

$$\star \in \{+, -, \cdot, /\}: \qquad x \widetilde{\star} y := \operatorname{rd}(x \star y) \tag{2.4.9}$$

Assumption 2.4.10 ("Axiom" of roundoff analysis)**.**
There is a small positive number $\mathtt{eps}$*, the machine precision, such that for the elementary arithmetic operations* $\star \in \{+, -, \cdot, /\}$ *and "hard-wired" functions** $f \in \{\exp, \sin, \cos, \log, \ldots\}$ *holds*

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

with $|\delta| < \mathtt{eps}$*.*

► 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

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

◇

Remark 2.4.13 (Adding eps to 1).

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

Code 2.4.14: $1 + \epsilon$ in MATLAB

```
>> fprintf('%30.25f\n',1+0.5*eps)
   1.0000000000000000000000000
>> fprintf('%30.25f\n',1-0.5*eps)
   0.9999999999999998889776975
>>
  fprintf('%30.25f\n',(1+2/eps)-2/eps);
   0.0000000000000000000000000
```

△

Do we have to worry about these tiny roundoff errors **?**

! YES (→ Sect. 2.3):

- accumulation of roundoff errors
- amplification of roundoff errors

▷ back to Gaussian elimination/LU-factorization with pivoting

2.5 Stability of Gaussian Elimination

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

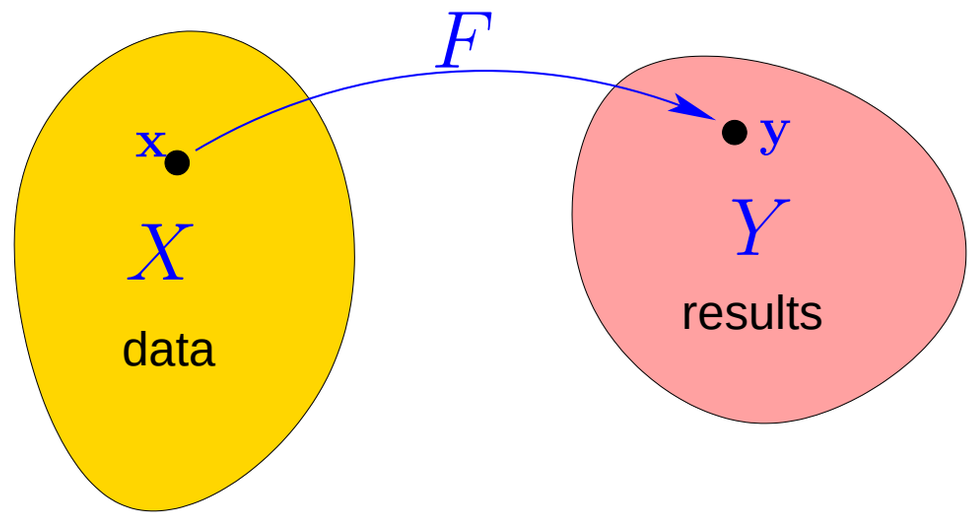
2.5.1 Vector norms and matrix norms [13, Sect. 2.1.2], [35, Sect. 1.2], [51, Sect. 1.11]

2.5.2 Numerical Stability [13, Sect. 2.3]

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 $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A} \in \mathbb{K}^{n,n}$, $\mathbf{b} \in \mathbb{K}^n$:

"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 $\boxed{(\mathbf{A}, \mathbf{b}) \mapsto F(\mathbf{A}, \mathbf{b}) := \mathbf{A}^{-1}\mathbf{b}}$, (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)

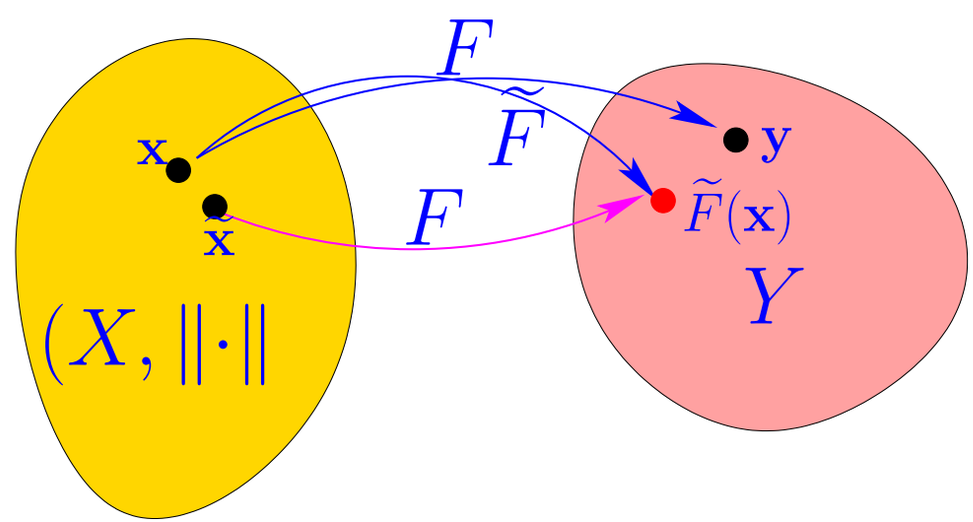
Below: X, Y = normed vector spaces, e.g., $X = \mathbb{R}^n$, $Y = \mathbb{R}^m$

Definition 2.5.11 (Stable algorithm)**.**
An algorithm $\widetilde{F}$ for solving a problem $F : X \mapsto Y$ is numerically stable, if for all $x \in X$ its result $\widetilde{F}(\mathbf{x})$ (affected by roundoff) is the exact result for "slightly perturbed" data:

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

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 $\mathbf{A}\mathbf{x} = \mathbf{b}$ *(*$\mathbf{A} \in \mathbb{K}^{n,n}$*,* $\mathbf{b} \in \mathbb{K}^n$*) are* numerically equivalent*:*

❶ *Gaussian elimination (forward elimination and back substitution) without pivoting, see Algorithm 2.1.3.*

❷ *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

Theorem 2.5.13 (Stability of Gaussian elimination with partial pivoting)**.**
Let $\mathbf{A} \in \mathbb{R}^{n,n}$ *be regular and* $\mathbf{A}^{(k)} \in \mathbb{R}^{n,n}$, $k = 1, \ldots, n-1$, *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 $\widetilde{\mathbf{x}} \in \mathbb{R}^n$ *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* $\mathtt{eps}$, $\rightarrow$ *Ass. 2.4.10) there is* $\Delta\mathbf{A} \in \mathbb{R}^{n,n}$ *with*

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

$$\textit{such that} \qquad (\mathbf{A} + \Delta\mathbf{A})\widetilde{\mathbf{x}} = \mathbf{b} \;.$$

R. Hiptmair

rev 38355, October 14, 2011

ρ "small" ➥ Gaussian elimination with partial pivoting is stable ($\rightarrow$ Def. 2.5.11)

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

Example 2.5.14 (Wilkinson's counterexample).

$$a_{ij} = \begin{cases} 1 & \text{, if } i = j \vee j = n \text{ ,} \\ -1 & \text{, if } i > j \text{ ,} \\ 0 & \text{else.} \end{cases} \quad , \qquad \text{n=10:} \quad \mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

Partial pivoting does not trigger row permutations !

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

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

◇

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

Example 2.5.15 (Stability by small random perturbations).

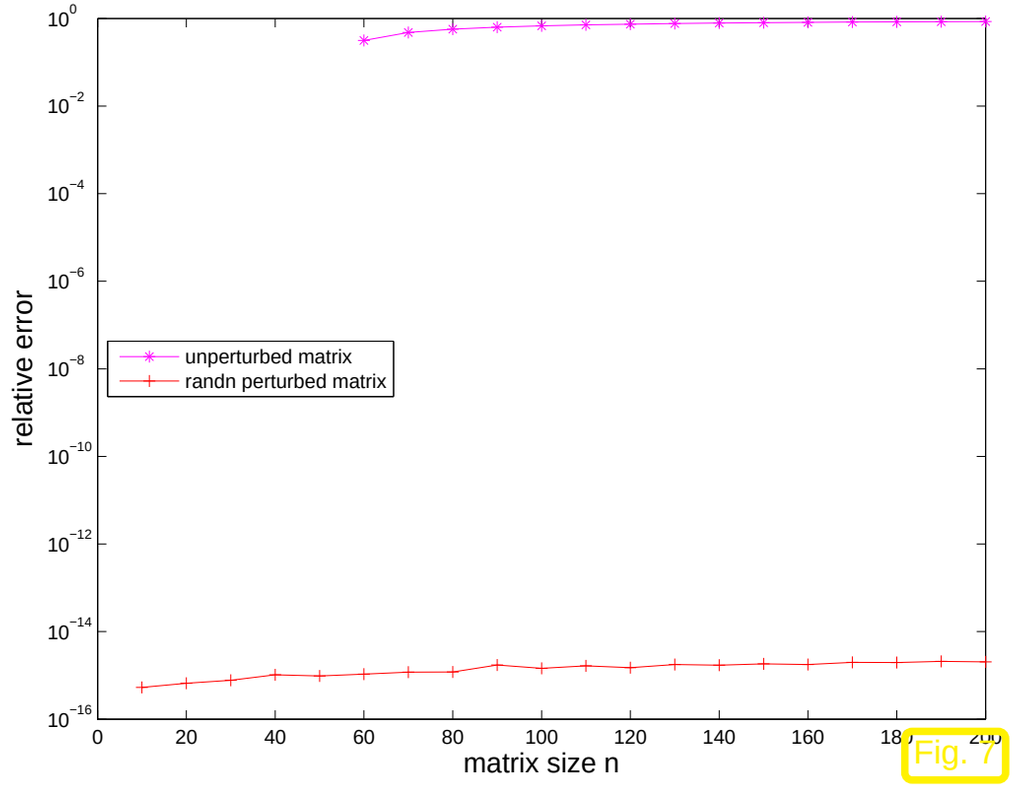
R. Hiptmair

rev 38355, October 14, 2011

```
% Curing Wilkinson's counterexample by random perturbation
% Theory:  Spielman and Teng
res = [];
for n=10:10:200
  % Build Wilkinson matrix
  A = [tril(-ones(n,n-1))+2*[eye(n-1);
       zeros(1,n-1)],ones(n,1)];
  % imposed solution
  x = ((-1).^(1:n))';
  relerr = norm(A\(A*x)-x)/norm(x);
  % Randomly perturbed Wilkinson matrix by matrix with iid
  % N(0,eps) distributed entries
  Ap = A + eps*randn(size(A));
  relerrp = norm(Ap\(A*x)-x)/norm(x);
  res = [res; n relerr relerrp];
end
semilogy(res(:,1),res(:,2),'m-*',res(:,1),res(:,3),'r-+');
xlabel('matrix size n','fontsize',14);
ylabel('relative error','fontsize',14);
legend('unperturbed matrix','randn perturbed
  matrix','location','west');

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


R. Hiptmair

rev 38355, October 14, 2011



◇

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 $\widetilde{\mathbf{x}} \in \mathbb{K}^n$ *of the LSE* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *(*$\mathbf{A} \in \mathbb{K}^{n,n}$*,* $\mathbf{b} \in \mathbb{K}^n$*), its residual is the vector*

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

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

R. Hiptmair

rev 38355, October 14, 2011

Numerical experiment with *nearly singular matrix*

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

singular rank-1 matrix

with

$$\mathbf{u} = \tfrac{1}{3}(1,2,3,\dots,10)^T \,,$$
$$\mathbf{v} = (-1,\tfrac{1}{2},-\tfrac{1}{3},\tfrac{1}{4},\dots,\tfrac{1}{10})^T$$

Code 2.5.18: small residuals for GE

```
n = 10; u = (1:n)'/3; v =
  (1./u).*(-1).^((1:n)');
x = ones(10,1); nx = norm(x,'inf');

result = [];
for epsilon = 10.^(-5:-0.5:-14)
    A = u*v' + epsilon*eye(n);
    b = A*x; nb = norm(b,'inf');
    xt = A\b; % Gaussian elimination
    r = b - A*xt; % residual
    result = [result; epsilon,
      norm(x-xt,'inf')/nx,
      norm(r,'inf')/nb, cond(A,'inf')];
end
```

R. Hiptmair

rev 38355, October 14, 2011

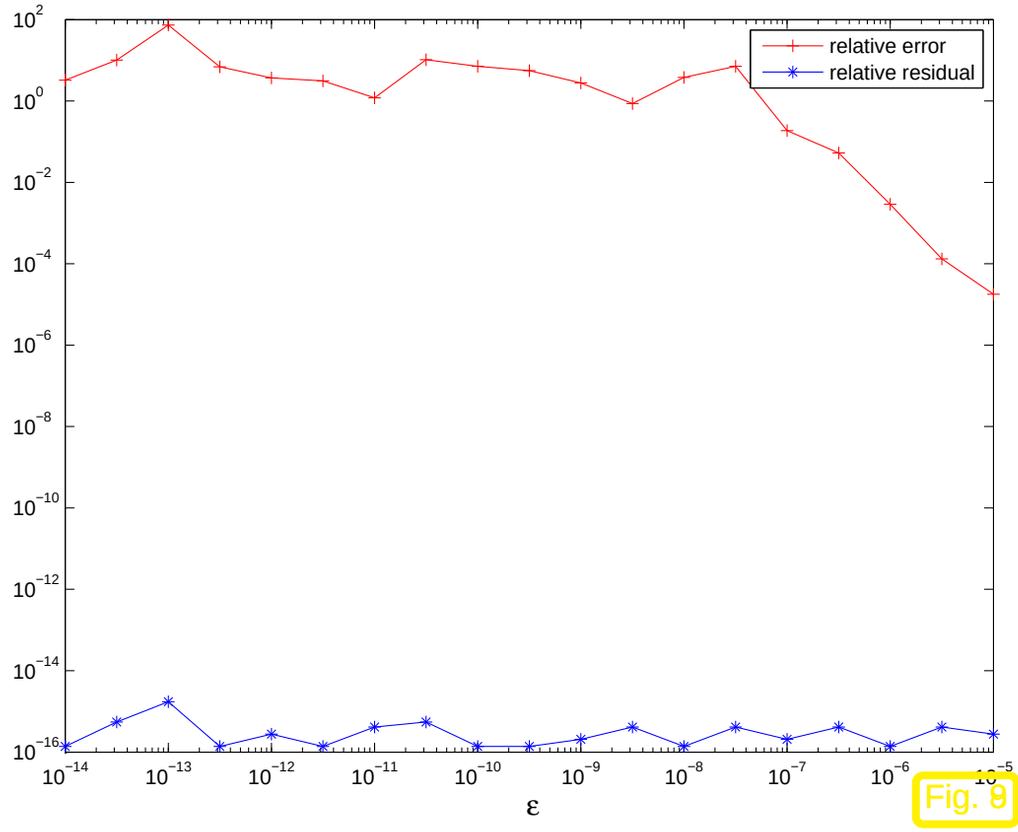


Fig. 9

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

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

◇

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

```
n = 10; u = (1:n)'/3; v = (1./u).*(-1).^u;
x = ones(10,1); nx = norm(x,'inf');

result = [];
for epsilon = 10.^(-5:-0.5:-14)
    A = u*v' + epsilon*rand(n,n);
    b = A*x; nb = norm(b,'inf');
    xt = A\b;        % Gaussian elimination
    r = b - A*xt;    % residualB
    B = inv(A); xi = B*b;
    ri = b - A*xi; % residual
    R = eye(n) - A*B; % residual
    result = [result; epsilon, norm(r,'inf')/nb, norm(ri,'inf')/nb, norm(R,'inf')/norm(B,'inf') ];
end
```

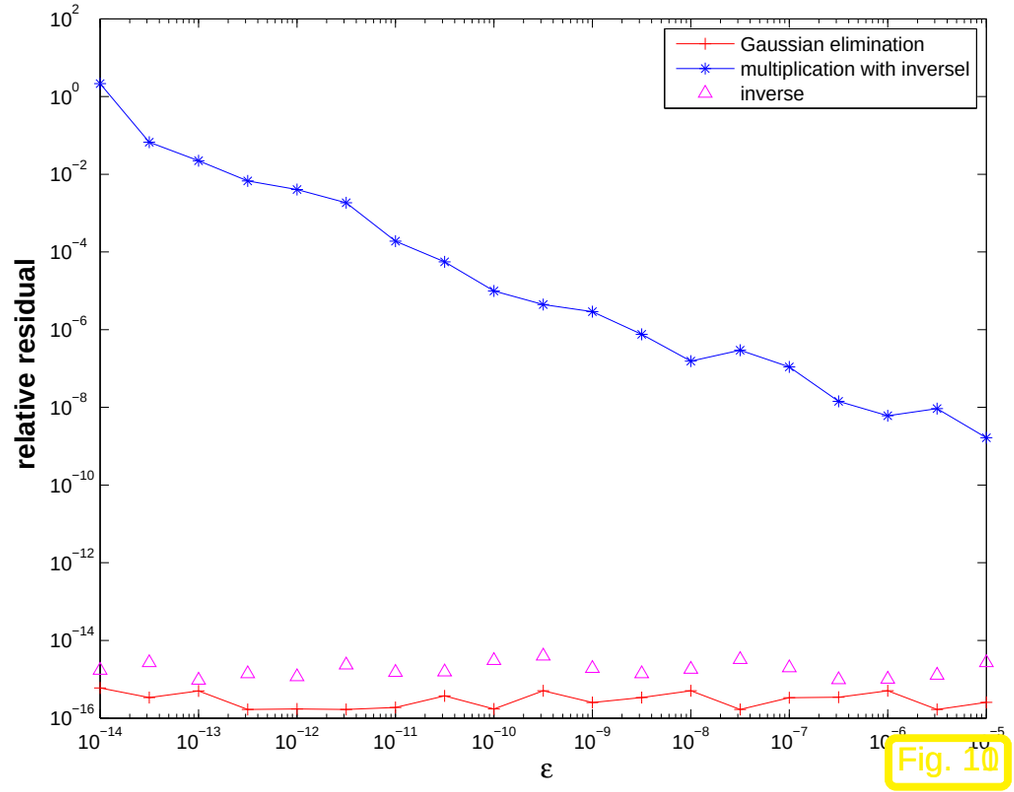


Fig. 10

2.5.4 Conditioning

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

(normwise) relative error: $\epsilon_r := \dfrac{\|\mathbf{x} - \widetilde{\mathbf{x}}\|}{\|\mathbf{x}\|}$.

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

Perturbed linear system:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \leftrightarrow \quad (\mathbf{A} + \Delta\mathbf{A})\widetilde{\mathbf{x}} = \mathbf{b} + \Delta\mathbf{b} \quad \blacktriangleright \quad (\mathbf{A} + \Delta\mathbf{A})(\widetilde{\mathbf{x}} - \mathbf{x}) = \Delta\mathbf{b} - \Delta\mathbf{A}\mathbf{x} \,. \tag{2.5.23}$$

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

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

$$\underbrace{\frac{\|\mathbf{x}-\widetilde{\mathbf{x}}\|}{\|\mathbf{x}\|}}_{\text{relative error}} \leq \frac{\left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\|}{1-\left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\| \|\Delta\mathbf{A}\| / \|\mathbf{A}\|} \underbrace{\left(\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}\right)}_{\text{relative perturbations}} .$$

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

Condition (number) of a matrix $\mathbf{A} \in \mathbb{R}^{n,n}$*:* $\qquad \operatorname{cond}(\mathbf{A}) := \left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\|$

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

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

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

(2.5.27) ➢

- If $\mathrm{cond}(\mathbf{A}) \gg 1$, *small perturbations* in $\mathbf{A}$ can lead to *large relative errors* in the solution of the LSE.
- If $\mathrm{cond}(\mathbf{A}) \gg 1$, 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

$$\mathbf{A} = \mathbf{u}\mathbf{v}^T + \epsilon\mathbf{I} \;,$$
$$\mathbf{u} = \tfrac{1}{3}(1,2,3,\dots,10)^T \;,$$
$$\mathbf{v} = (-1,\tfrac{1}{2},-\tfrac{1}{3},\tfrac{1}{4},\dots,\tfrac{1}{10})^T$$

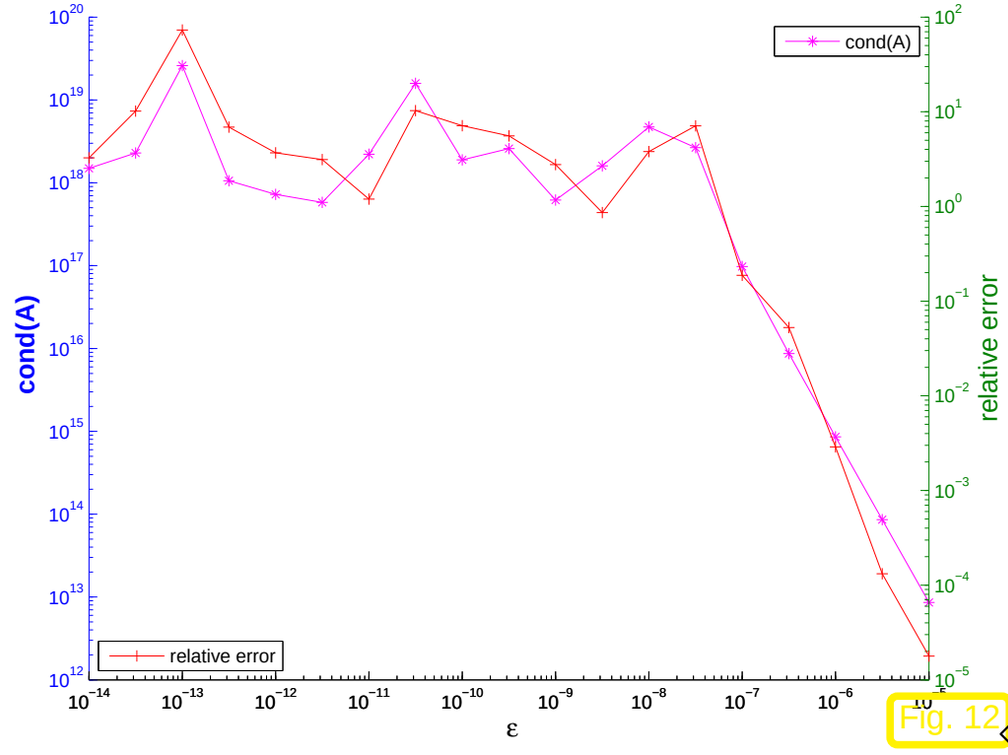


Fig. 12 ◇

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

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

$\updownarrow (*)$

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

▷ Instability of Gaussian elimination !

Code 2.5.30: GE for “Wilkinson system”

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

R. Hiptmair

rev 38355, October 14, 2011

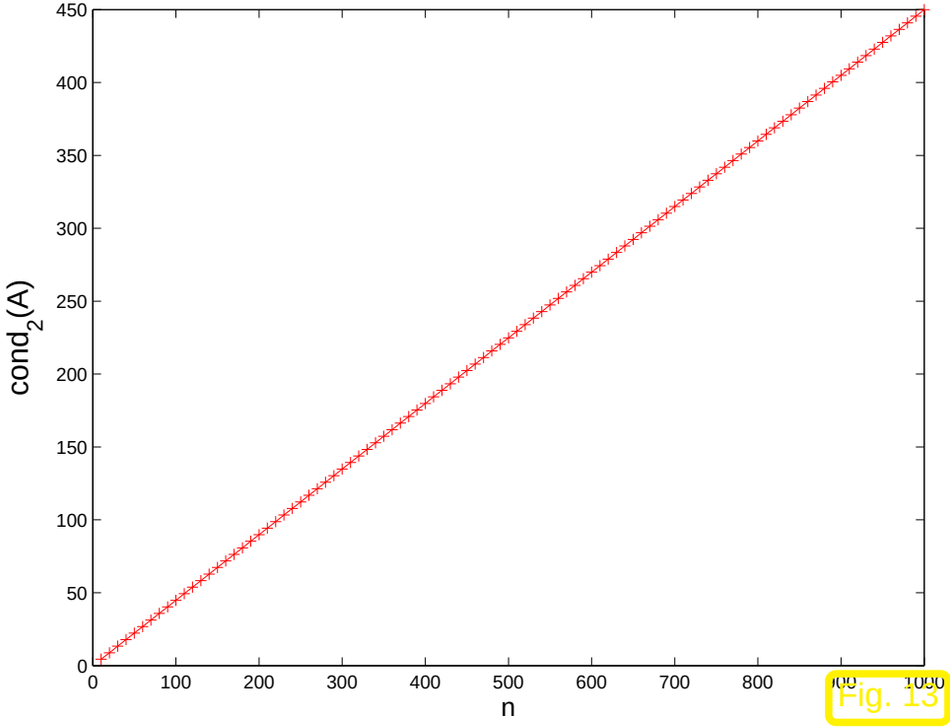

cond$_2$(A)
n
0
50
100
150
200
250
300
350
400
450
0
100
200
300
400
500
600
700
800
1000

Fig. 13

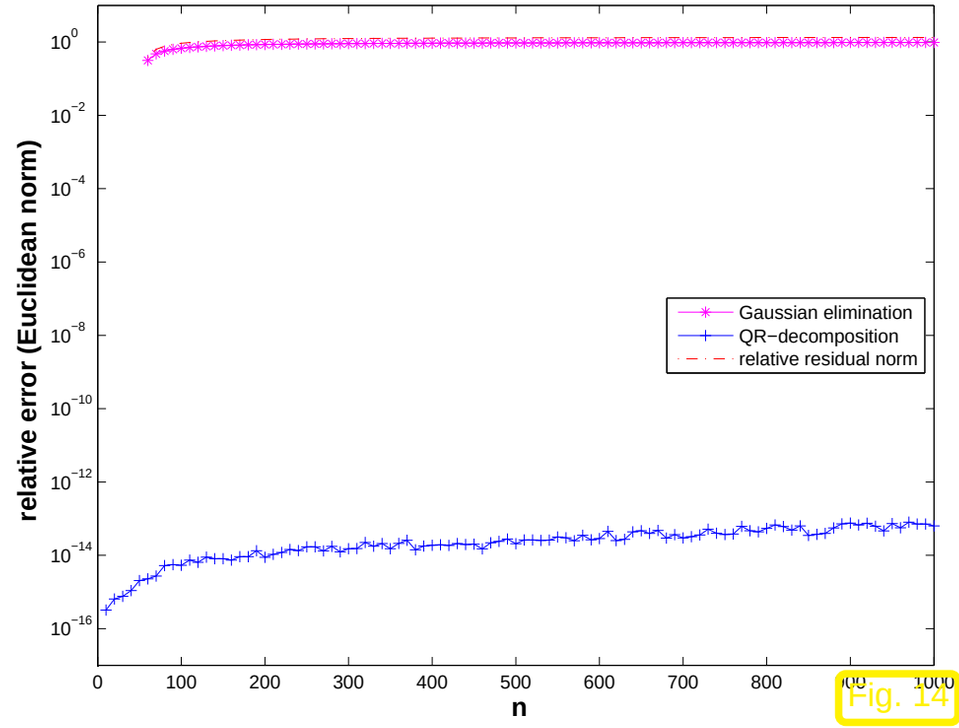

relative error (Euclidean norm)
n
Gaussian elimination
QR−decomposition
relative residual norm
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
0
100
200
300
400
500
600
700
800
1000

Fig. 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 $\|\cdot\|$

$$\begin{array}{c}\mathbf{A}\mathbf{x} = \mathbf{b} \\ (\mathbf{A}+\Delta\mathbf{A})\widetilde{\mathbf{x}} = \mathbf{b}+\Delta\mathbf{b}\end{array} \quad \Rightarrow \quad \frac{\|\mathbf{x}-\widetilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\operatorname{cond}(\mathbf{A})}{1-\operatorname{cond}(\mathbf{A})\,\|\Delta\mathbf{A}\|\,/\,\|\mathbf{A}\|}\left(\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}+\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}\right) . \tag{2.5.31}$$

▶ $\operatorname{cond}(\mathbf{A}) \gg 1$ ➢ <u>small</u> relative changes of data $\mathbf{A}, \mathbf{b}$ *may* effect <u>huge</u> relative changes in solution.

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

▶ $\operatorname{cond}(\mathbf{A})$ 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 $\mathbf{A}, \mathbf{b}$).

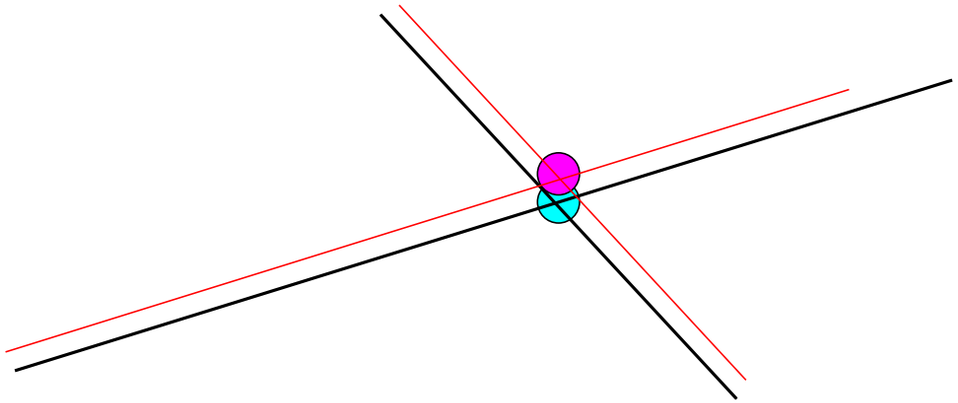
Terminology:

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

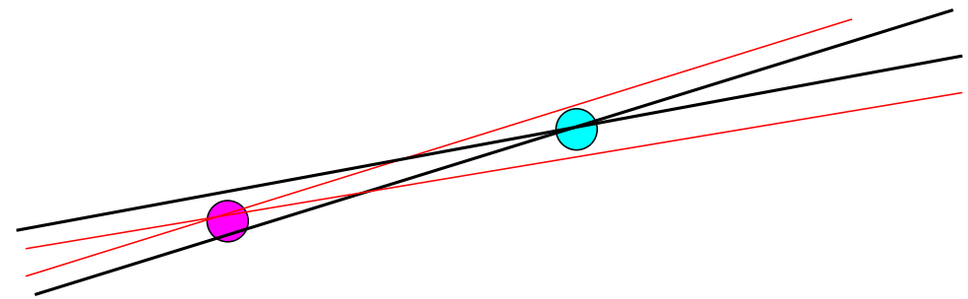
Note: sensitivity gauge depends on the chosen norm **!**

Example 2.5.32 (Intersection of lines in 2D).

In distance metric:



nearly orthogonal intersection: well-conditioned



glancing intersection: ill-conditioned

R. Hiptmair

rev 38355, October 14, 2011

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

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

▶ 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 2×2 matrices

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

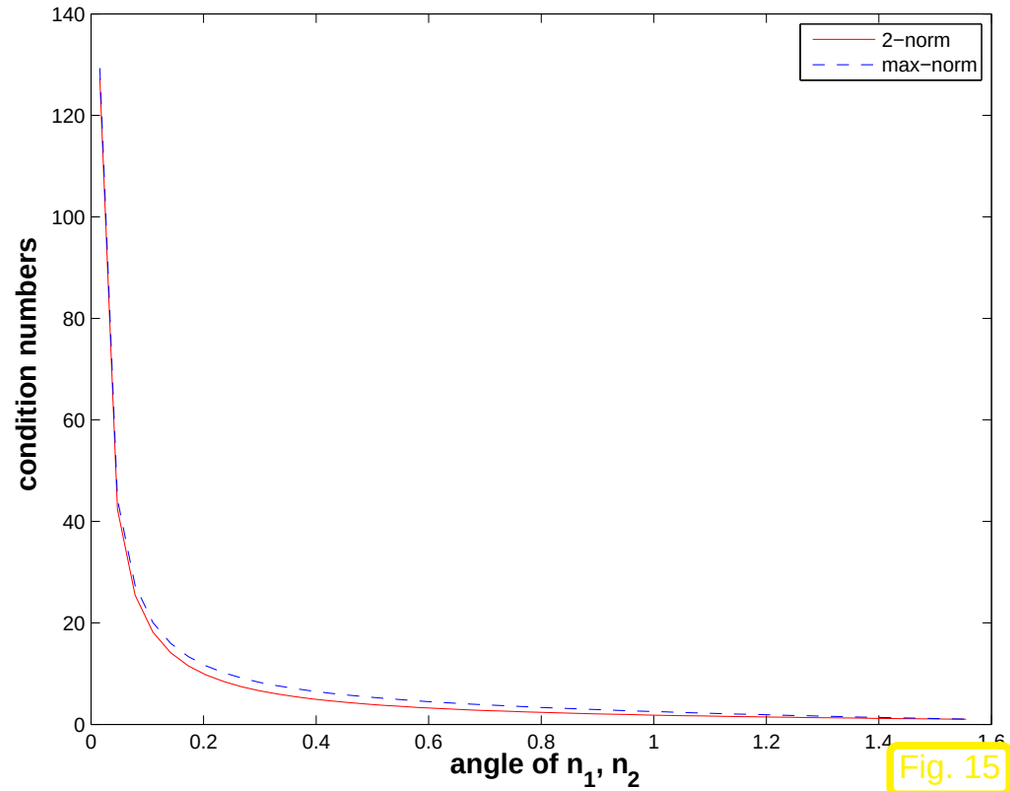


Fig. 15

◇

Heuristics:

$$\operatorname{cond}(\mathbf{A}) \gg 1 \quad \leftrightarrow \quad \text{columns/rows of } \mathbf{A} \text{ “almost linearly dependent”}$$

R. Hiptmair

rev 38355, October 7, 2011

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*

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

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

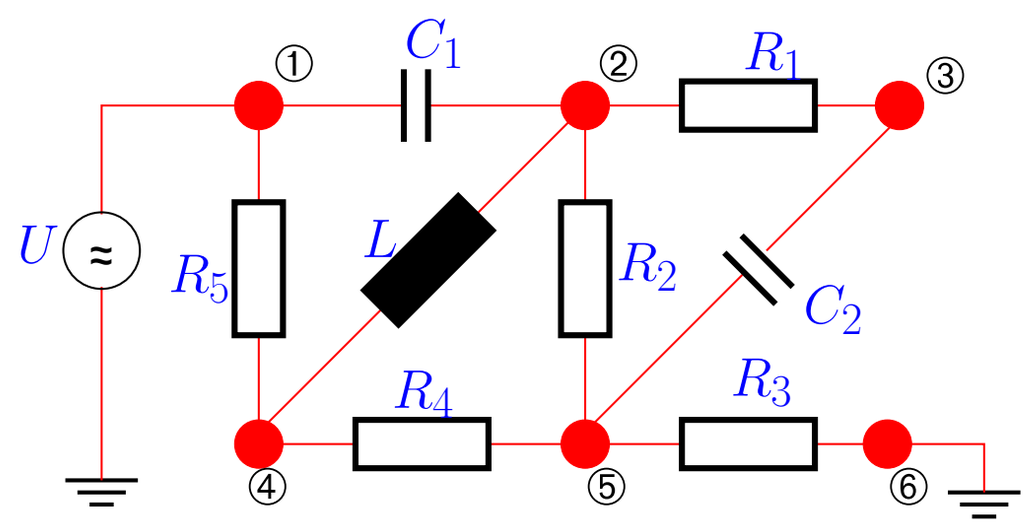
Node (*ger.:* Knoten) $\hat{=}$ junction of wires

☞ *number* nodes $1, \dots, n$

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

Kirchhoff current law (KCL) : sum of node currents $= 0$:

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



R. Hiptmair

rev 38355, October 7, 2011

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: $I = \dfrac{U}{R}$, $[R] = 1\mathrm{VA}^{-1}$
- capacitor: $I = \imath\omega C U$, capacitance $[C] = 1\mathrm{AsV}^{-1}$
- coil/inductor : $I = \dfrac{U}{\imath\omega L}$, inductance $[L] = 1\mathrm{Vs}A^{-1}$

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

Constitutive relations + (2.6.4) ▶ linear system of equations:

$$\begin{aligned}
② : \quad & \imath\omega C_1(U_2 - U_1) + R_1^{-1}(U_2 - U_3) - \imath\omega^{-1}L^{-1}(U_2 - U_4) + R_2^{-1}(U_2 - U_5) &= 0 \ , \\
③ : \quad & R_1^{-1}(U_3 - U_2) + \imath\omega C_2(U_3 - U_5) &= 0 \ , \\
④ : \quad & R_5^{-1}(U_4 - U_1) - \imath\omega^{-1}L^{-1}(U_4 - U_2) + R_4^{-1}(U_4 - U_5) &= 0 \ , \\
⑤ : \quad & R_2^{-1}(U_5 - U_2) + \imath\omega C_2(U_5 - U_3) + R_4^{-1}(U_5 - U_4) + R_3^{-1}(U_5 - U_6) &= 0 \ , \\
& U_1 = U \quad , \quad U_6 = 0 \ .
\end{aligned}$$

▼

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

◇

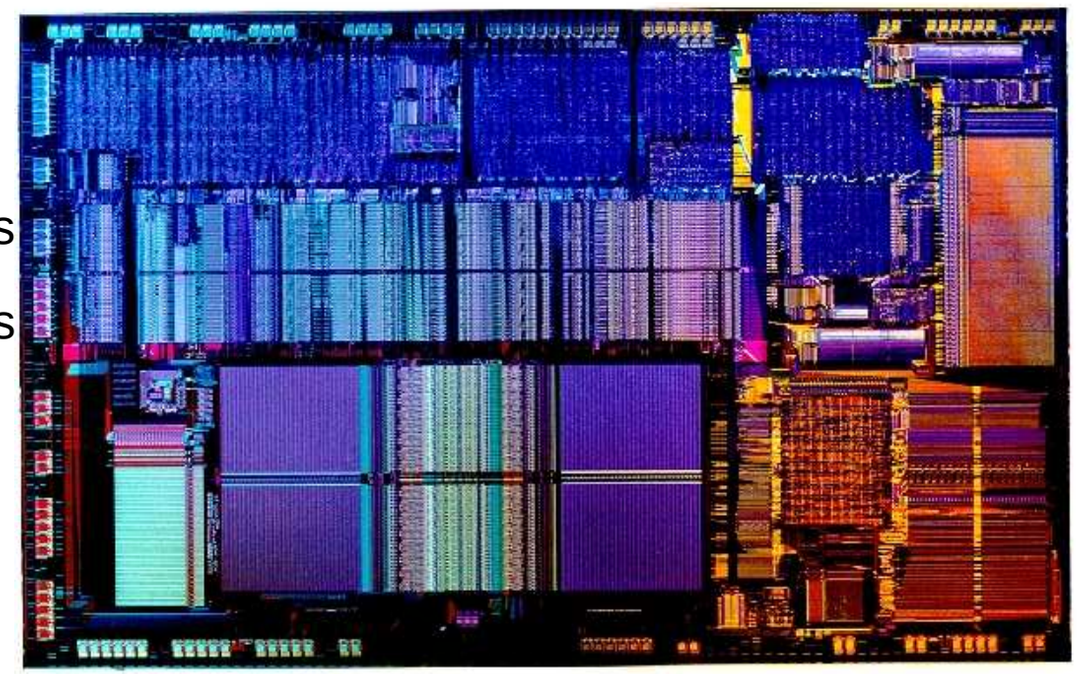
Example 2.6.7 (Sparse LSE in circuit modelling).

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



◇

R. Hiptmair

rev 38355, October 7, 2011

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) $\rightarrow$ 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).

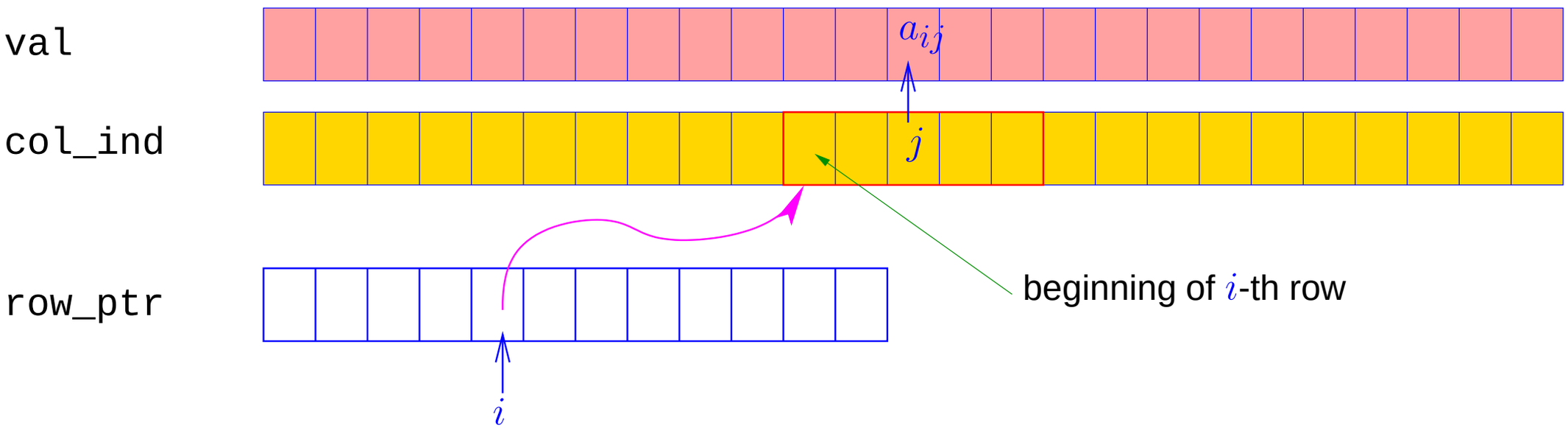
Data for matrix $\mathbf{A} = (a_{ij}) \in \mathbb{K}^{n,n}$ kept in three arrays

`double * val`	size $\mathrm{nnz}(\mathbf{A}) := \#\{(i,j) \in \{1,\dots,n\}^2, a_{ij} \neq 0\}$
`unsigned int * col_ind`	size $\mathrm{nnz}(\mathbf{A})$
`unsigned int * row_ptr`	size $n+1$ & `row_ptr`$[n+1] = \mathrm{nnz}(\mathbf{A}) + 1$

$\mathrm{nnz}(\mathbf{A}) \hat{=}$ (**n**umber of **n**on**z**eros) of $\mathbf{A}$

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

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



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

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

◇

R. Hiptmair

rev 38355, October 7, 2011

2.6.2 Sparse matrices in MATLAB

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).

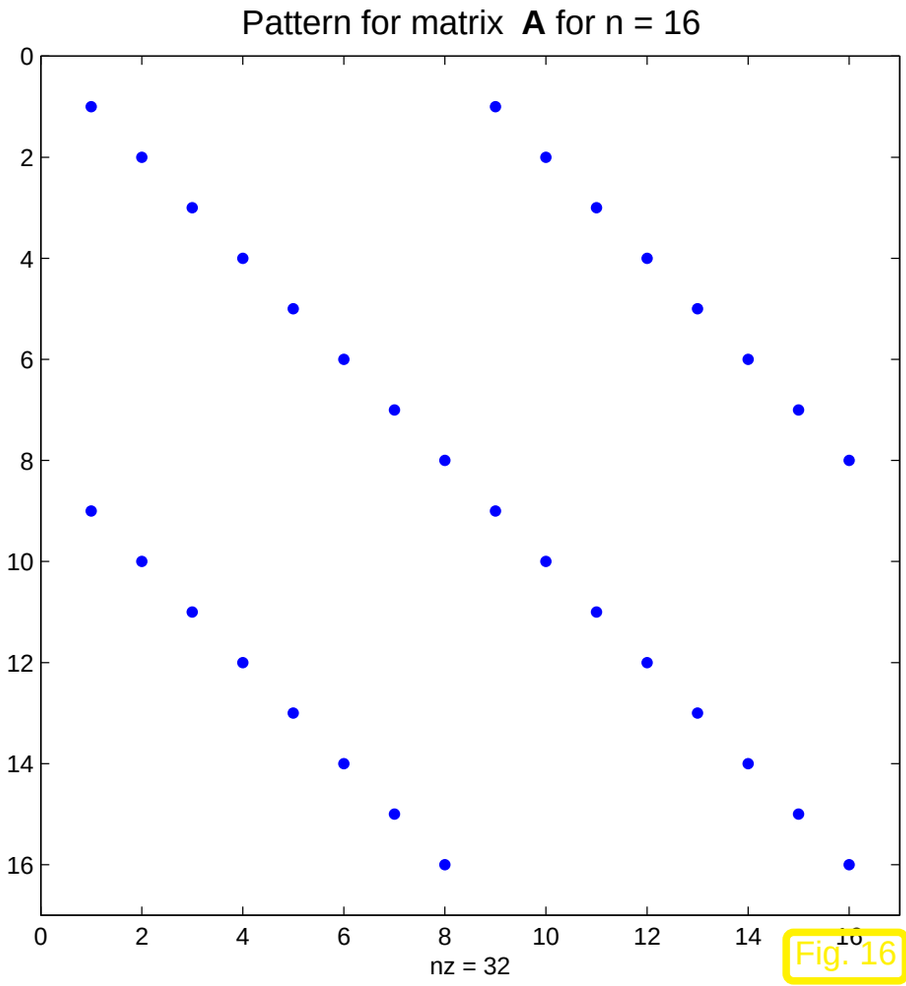


Fig. 16

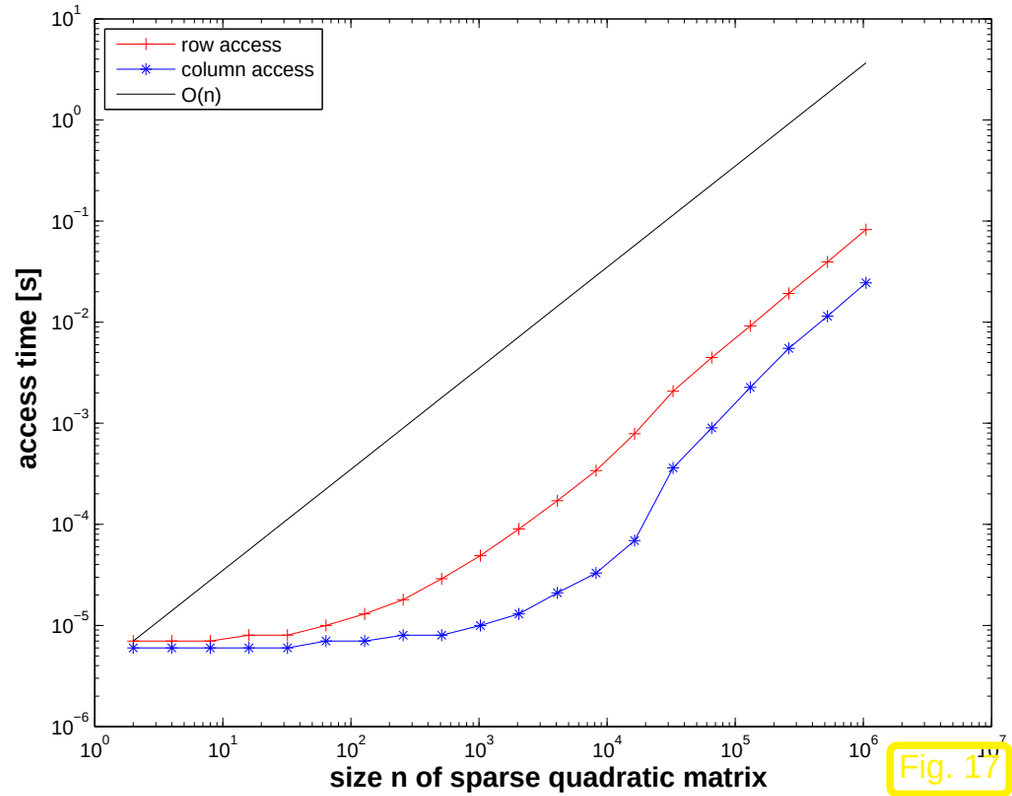


Fig. 17

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

```
figure; spy(spdiags(repmat([-1 2 5],16,1),[-8,0,8],16,16)); %
title('Pattern for matrix {\bf A} for n = 16','fontsize',14);
print -depsc2 '../PICTURES/spdiagsmatspy.eps';

t = [];
for i=1:20
  n = 2^i; m = n/2;
```

```
  A = spdiags(repmat([-1 2 5],n,1),[-n/2,0,n/2],n,n); %

  t1 = inf; for j=1:5, tic; v = A(m,:)+j; t1 = min(t1,toc); end
  t2 = inf; for j=1:5, tic; v = A(:,m)+j; t2 = min(t2,toc); end
  t = [t; size(A,1), nnz(A), t1, t2 ];
end

figure;
loglog(t(:,1),t(:,3),'r+-', t(:,1),t(:,4),'b*-',...
       t(:,1),t(1,3)*t(:,1)/t(1,1),'k-');
xlabel('{\bf size n of sparse quadratic matrix}','fontsize',14);
ylabel('{\bf access time [s]}','fontsize',14);
legend('row access','column
  access','O(n)','location','northwest');

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

◇

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

Code 2.6.12: Initialization of sparse matrices: version I

```
A1 = sparse(n,n);
```

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

Code 2.6.13: Initialization of sparse matrices: version II

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

Code 2.6.14: Initialization of sparse matrices: version III

```
dat = zeros(6*n,3); k = 0;
  for i=1:n
    for j=1:n
        if (abs(i-j) == 1), k=k+1; dat(k,:) = [i,j,1.0];
        end;
        if (abs(i-j) == round(n/3))
            k=k+1; dat(k,:) = [i,j,-1.0];
        end;
```

```
  end; end;
  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

```
% Driver routine for initialization of sparse matrices
K = 3; r = [];
for n=2.^(8:14)
    t1= 1000; for k=1:K,  fprintf('sparse1, %d, %d\n',n,k); tic;
      sparse1; t1 = min(t1,toc); end
     t2= 1000; for k=1:K,  fprintf('sparse2, %d, %d\n',n,k); tic;
       sparse2; t2 = min(t2,toc); end
    t3= 1000; for k=1:K,  fprintf('sparse3, %d, %d\n',n,k); tic;
      sparse3; t3 = min(t3,toc); end
    r = [r; n, t1 , t2, t3];
end

loglog(r(:,1),r(:,2),'r*',r(:,1),r(:,3),'m+',r(:,1),r(:,4),'b^');
xlabel('{\bf matrix size n}','fontsize',14);
ylabel('{\bf time [s]}','fontsize',14);
legend('Initialization I','Initialization II','Initialization
  III',...
    'location','northwest');
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)

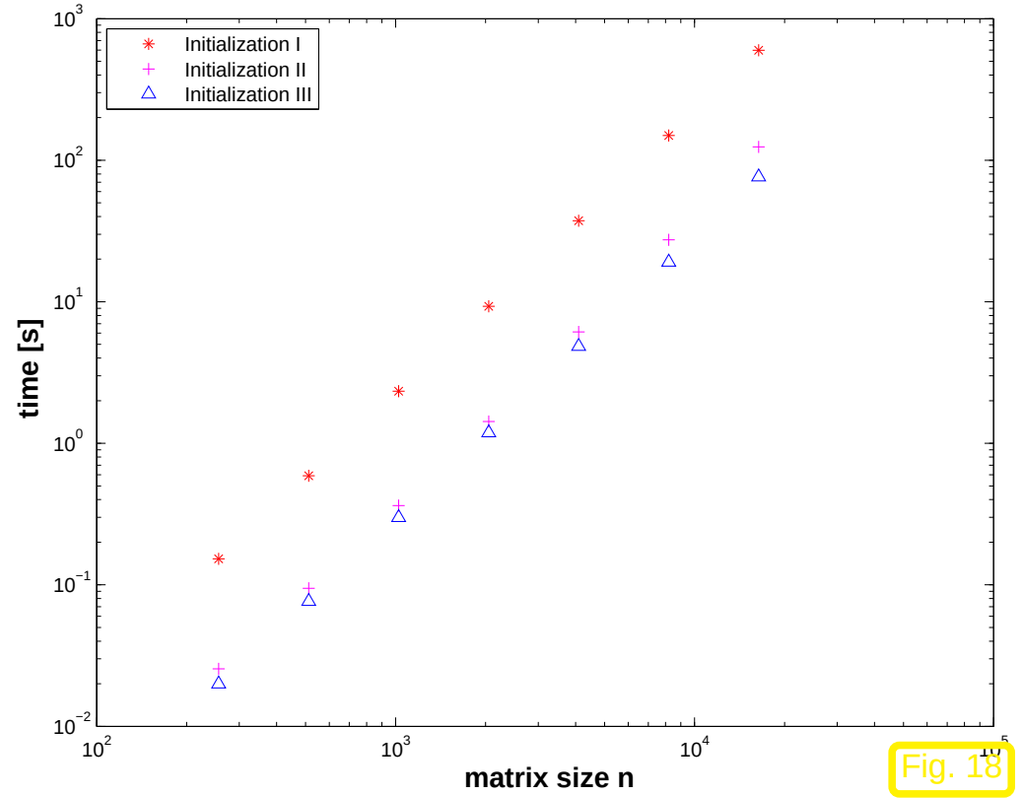


Fig. 18

◇

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);
```

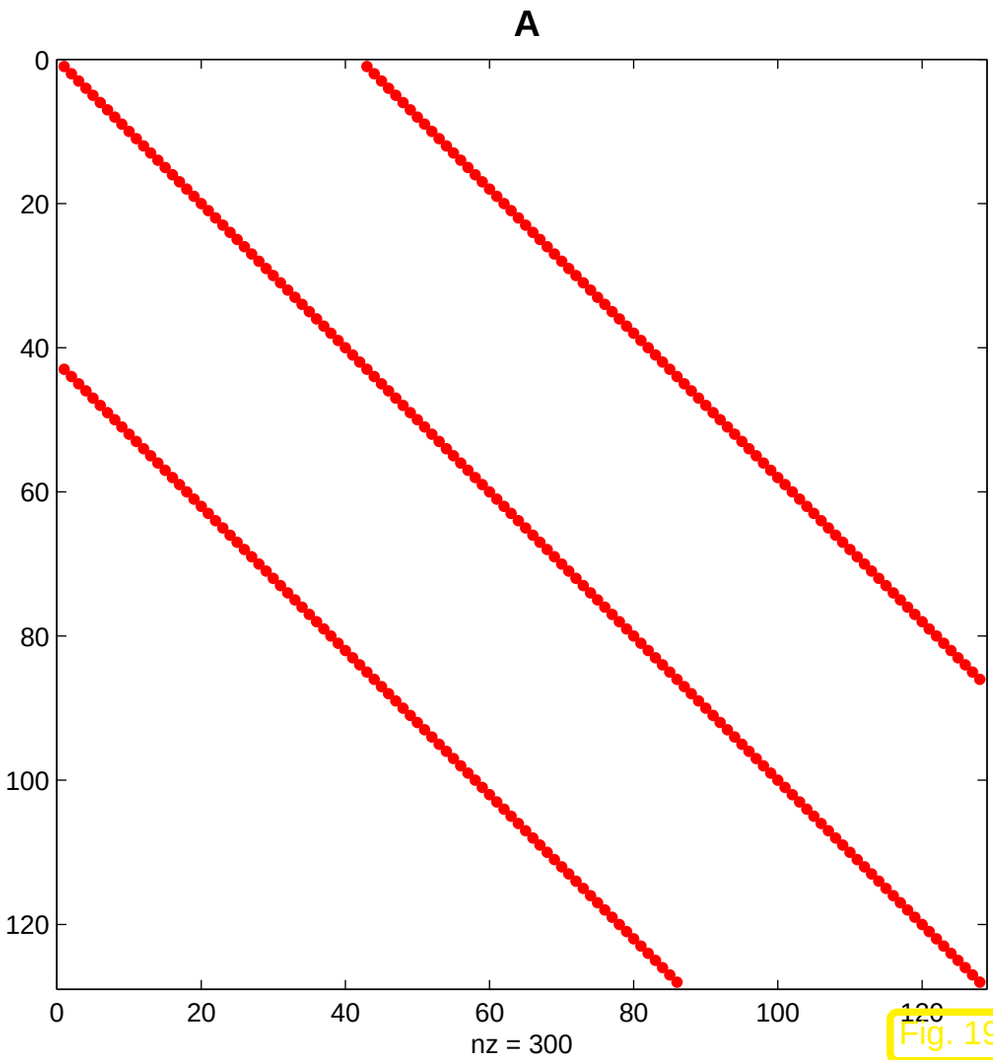


Fig. 19

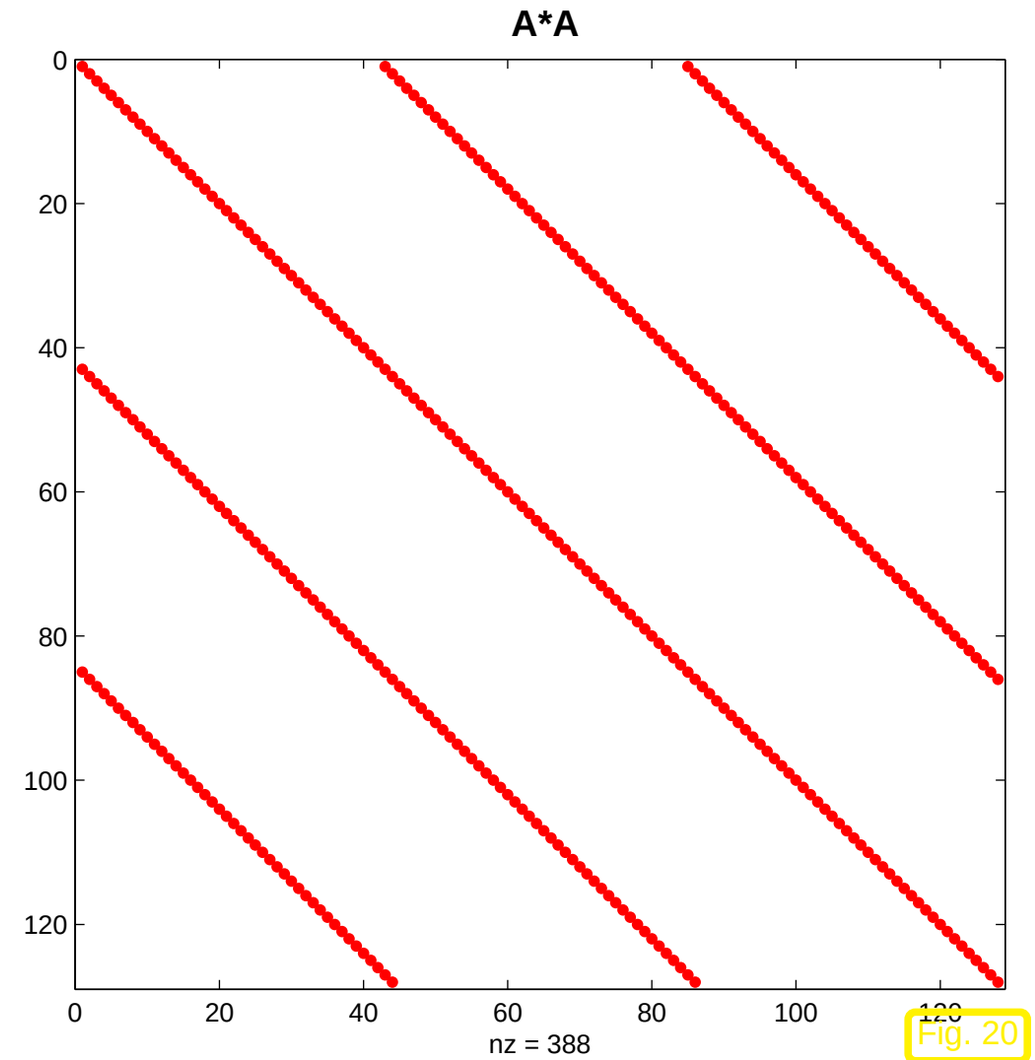
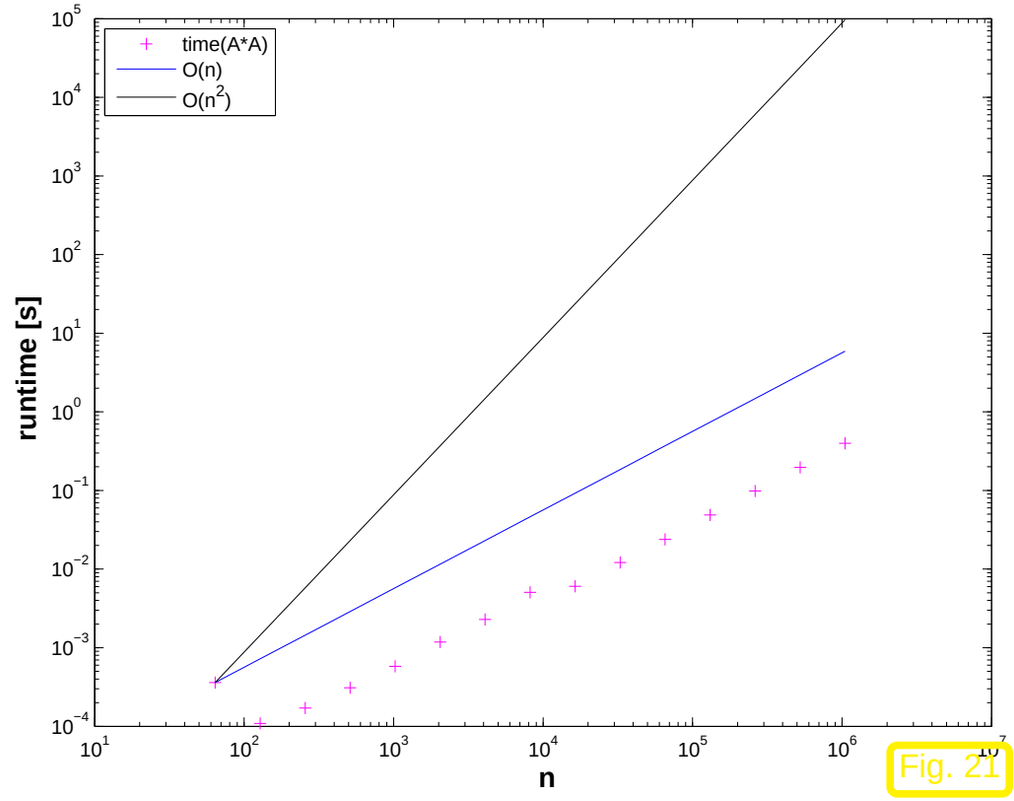


Fig. 20

➢ $\mathbf{A}^2$ is still a sparse matrix ($\rightarrow$ Notion 2.6.1)

runtimes for matrix multiplication `A*A` in MATLAB (`tic`/`toc` timing) ▷

(same platform as in Ex. 2.6.11)



▶ $O(n)$ asymptotic complexity: "optimal" implementation ! ◇

Remark 2.6.17 (Silly MATLAB).

A strange behavior of MATLAB from version R2010a:

```
% MATLAB script demonstrating the awkward effects of treating entries of
% sparse matrices as sparse matrices themselves.
A = spdiags(ones(100,3),-1:1,100,100);
b = A(1,1),
```


R. Hiptmair

rev 38355, October 7, 2011

```
c = full(b),
whos('b','c');
sum=0; tic; for i=1:1e6, sum = sum + b; end, toc
sum=0; tic; for i=1:1e6, sum = sum + c; end, toc
```

△

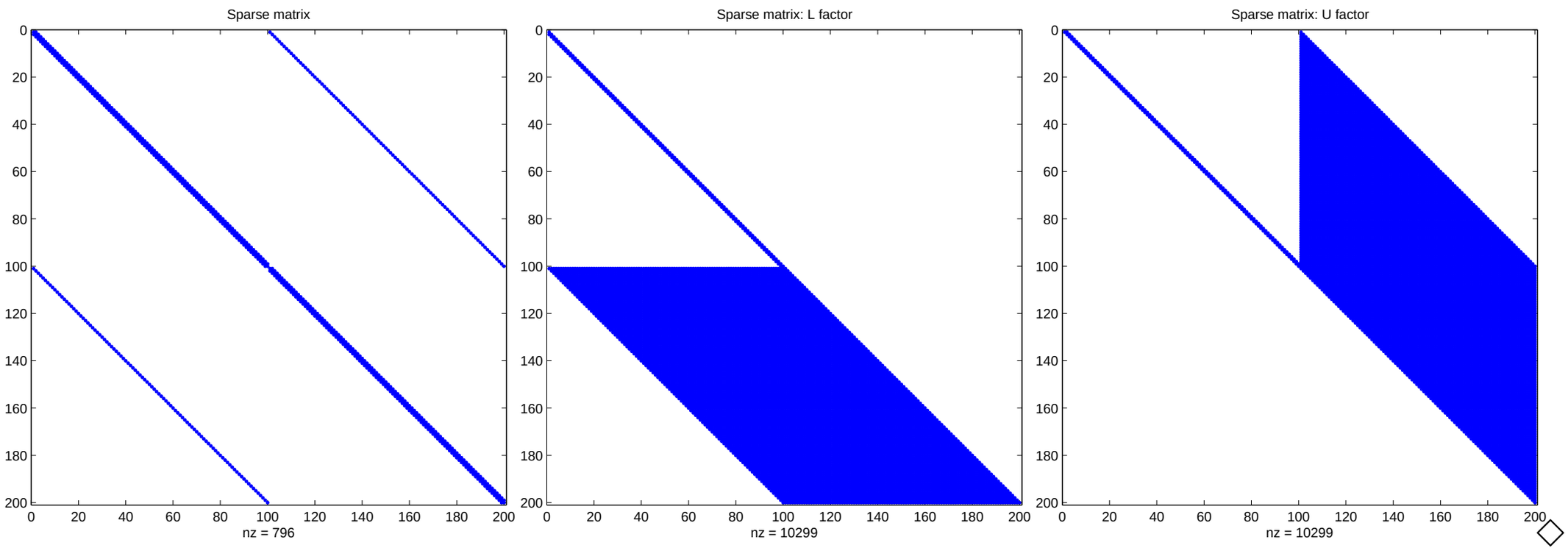
2.6.3 LU-factorization of sparse matrices

Example 2.6.18 (LU-factorization of sparse matrices)*.*

$$\mathbf{A} = \left(\begin{array}{cccc|cccc} 3 & -1 & & & -1 & & & \\ -1 & \ddots & \ddots & & & \ddots & & \\ & \ddots & \ddots & -1 & & & \ddots & \\ & & -1 & 3 & & & & -1 \\ \hline -1 & & & & 3 & -1 & & \\ & \ddots & & & -1 & \ddots & \ddots & \\ & & \ddots & & & \ddots & \ddots & -1 \\ & & & -1 & & & -1 & 3 \end{array}\right) \in \mathbb{R}^{n,n}, n \in \mathbb{N}$$

Code 2.6.19: LU-factorization of sparse matrix

```
% Demonstration of fill-in for LU-factorization of sparse matrices
n = 100;
A = [gallery('tridiag',n,-1,3,-1), speye(n); speye(n) ,
  gallery('tridiag',n,-1,3,-1)]; [L,U,P] = lu(A);
figure; spy(A); title('Sparse matrix'); print -depsc2
  '../PICTURES/sparseA.eps';
figure; spy(L); title('Sparse matrix: L factor'); print -depsc2
  '../PICTURES/sparseL.eps';
figure; spy(U); title('Sparse matrix: U factor'); print -depsc2
  '../PICTURES/sparseU.eps';
```



R. Hiptmair

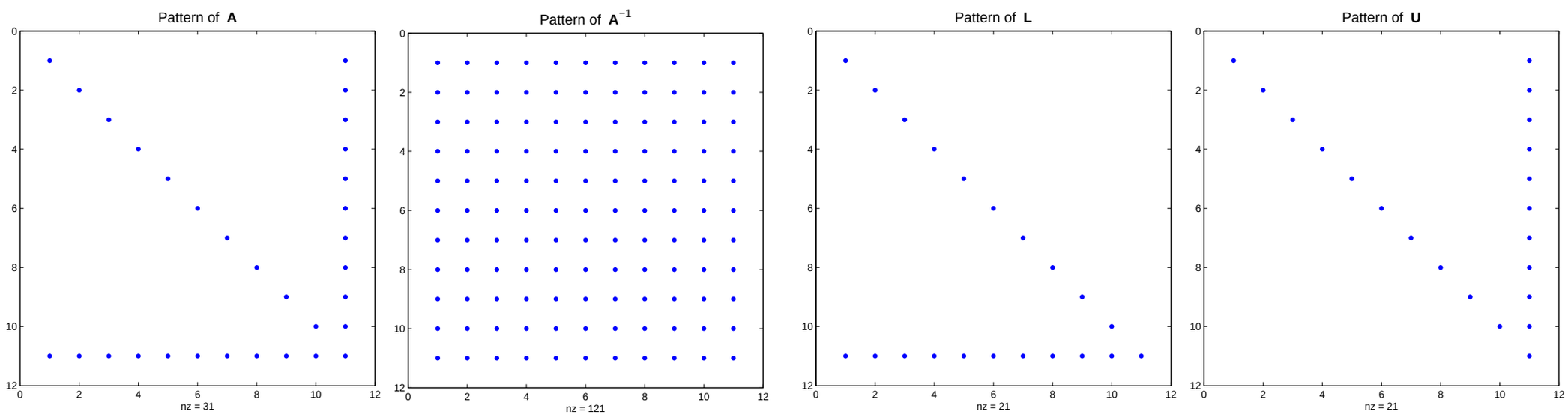
rev 38355, October 7, 2011

$\mathbf{A}$ sparse $\not\Rightarrow$ 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)*.*

```
% Simple example for dense inverse despite sparse LU-factors
A = [diag(1:10),ones(10,1);ones(1,10),2];
[L,U] = lu(A); spy(A); spy(L); spy(U); spy(inv(A));
```



$$\mathbf{L}, \mathbf{U} \text{ sparse} \quad \not\Longrightarrow \quad \mathbf{A}^{-1} \text{ sparse !}$$

◇

Example 2.6.22 ("arrow matrix").

$$\mathbf{A} = \left(\begin{array}{c|c} \alpha & \mathbf{b}^\top \\ \hline \mathbf{c} & \mathbf{D} \end{array}\right) , \quad \begin{array}{l} \alpha \in \mathbb{R} , \\ \mathbf{b}, \mathbf{c} \in \mathbb{R}^{n-1} , \\ \mathbf{D} \in \mathbb{R}^{n-1,n-1} \text{ regular diagonal matrix, } \rightarrow \text{Def. 2.2.3} \end{array} \tag{2.6.23}$$

Run algorithm 2.3.8 (Gaussian elimination without pivoting):

- factor matrices with $O(n^2)$ non-zero entries.
- computational costs: $O(n^3)$

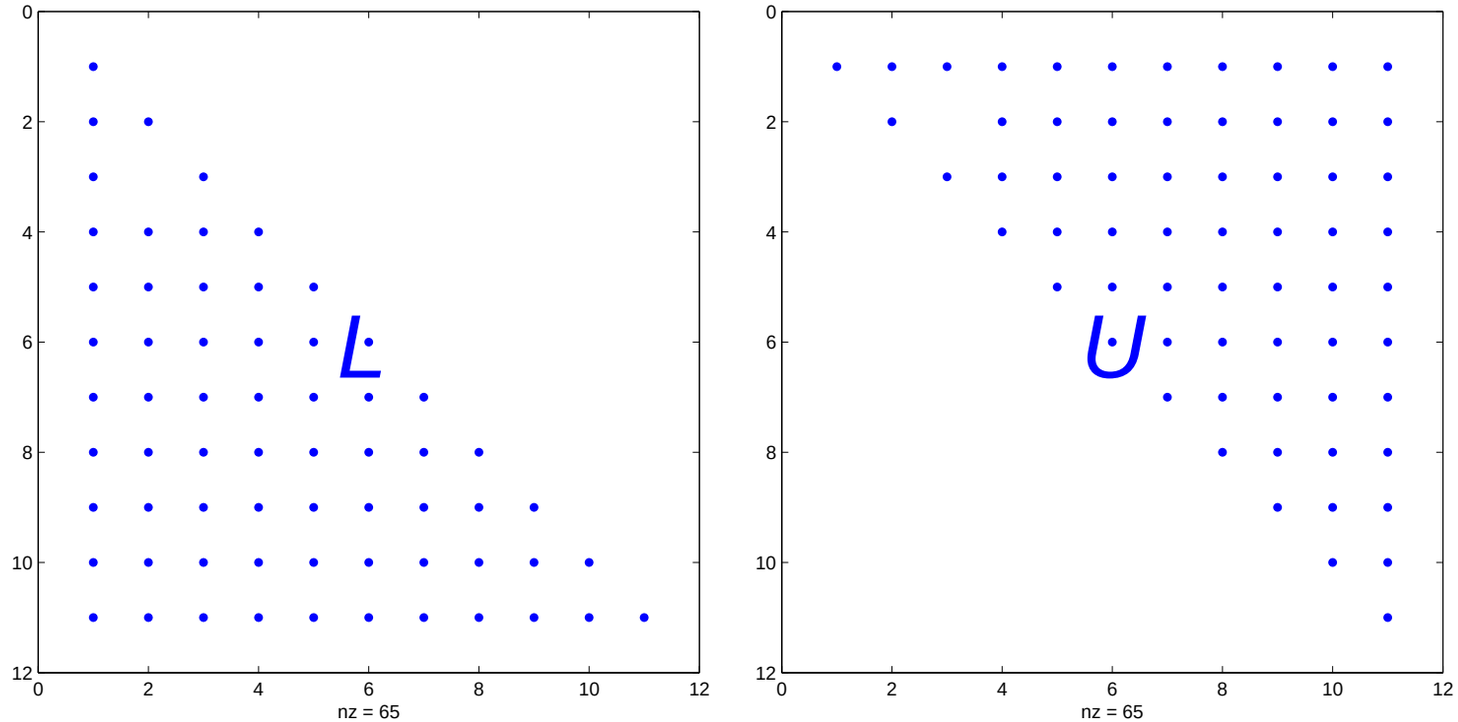
Code 2.6.24: LU-factorization of arrow matrix

```
n = 10; A = [ n+1, (n:-1:1) ;
  ones(n,1), eye(n,n)];
[L,U,P] = lu(A); spy(L); spy(U);
```

R. Hiptmair

rev 38355, October 7, 2011

Obvious fill-in ($\rightarrow$ Def. 2.6.20)



R. Hiptmair

rev 38355, October 7, 2011

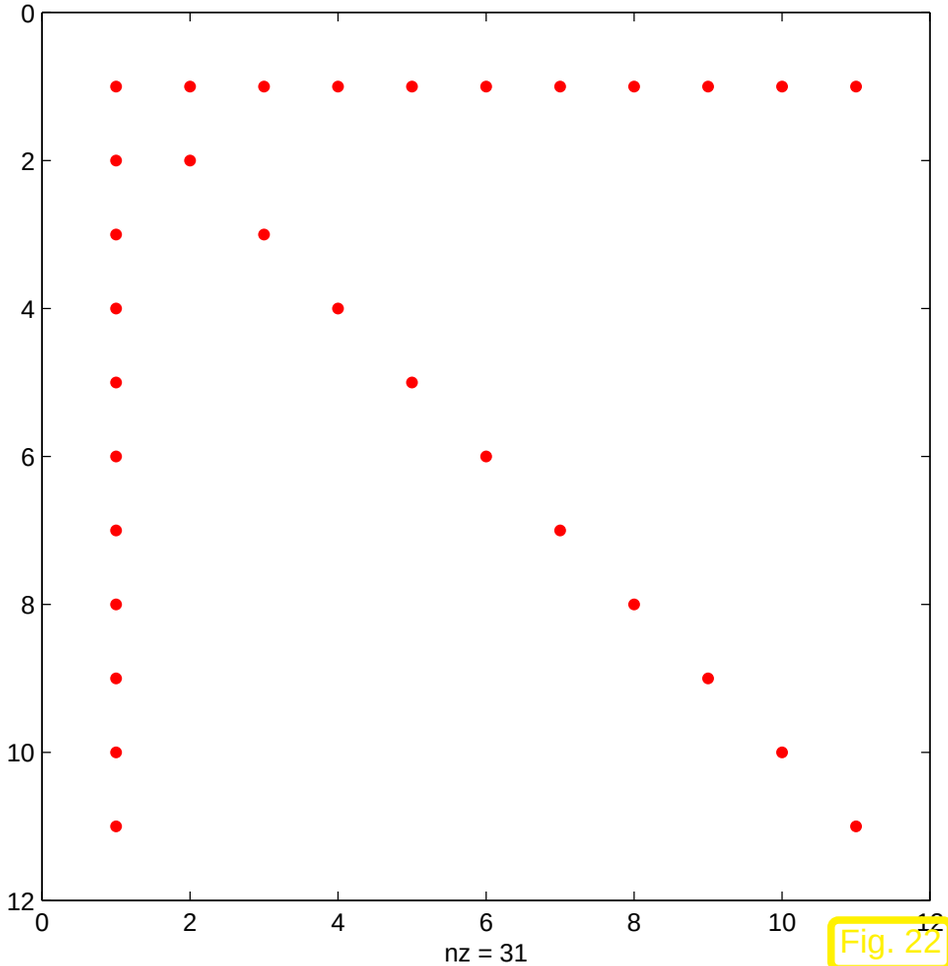


Fig. 22

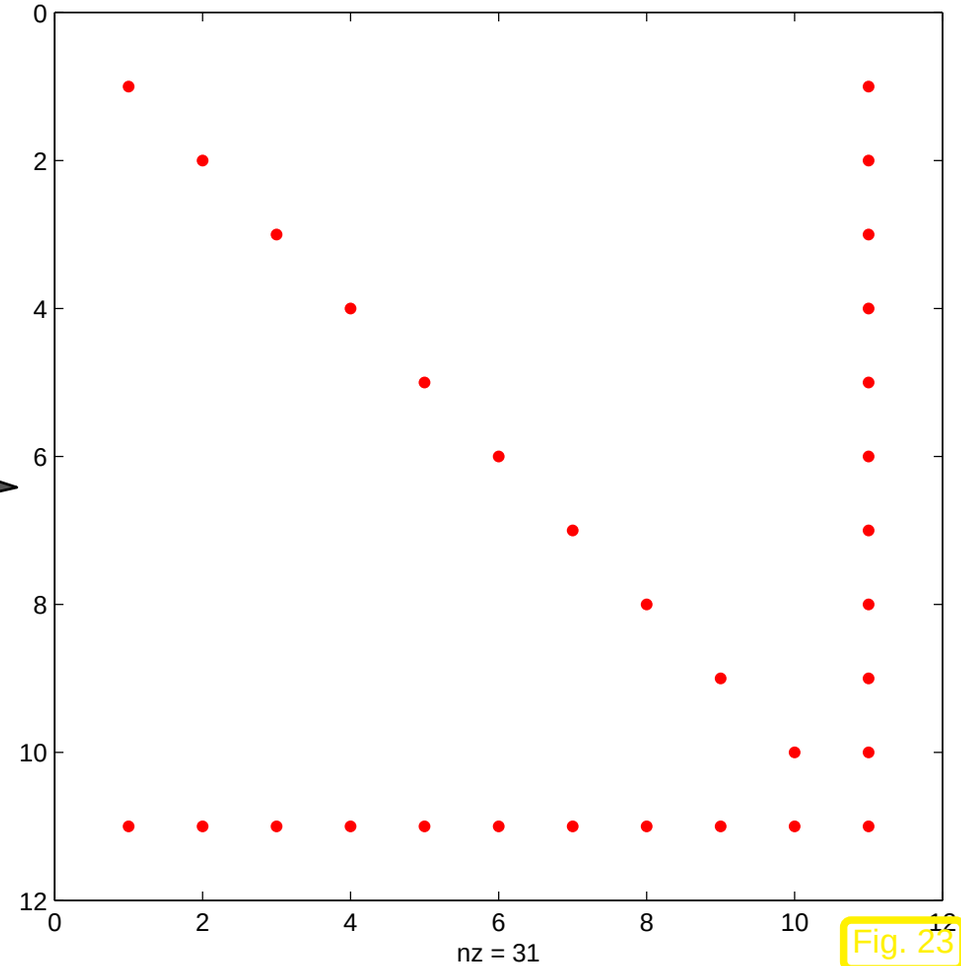


Fig. 23

Code 2.6.25: Permuting arrow matrix, see Figs. 29, 30

```
n = 10; A = [ n+1, (n:-1:1) ; ones(n,1), eye(n,n)];
% Permutation matrix (→ Def. 2.3.12) encoding cyclic permutation
P = [zeros(n,1),eye(n);1,zeros(1,n)];

figure('name','A');
spy(A,'r.'); print -depsc  '../PICTURES/InvArrowSpy.eps';
figure('name','PAPT');
```

```
spy(P*A*P','r.'); print -depsc '../PICTURES/ArrowSpy.eps';
```


Solving LSE $\mathbf{Ax} = \mathbf{y}$ with $\mathbf{A}$ from 2.6.23: two MATLAB codes

"naive" implementation via "\":

Code 2.6.28: LSE with arrow matrix, implementation I

```
function x =
  sa1(alpha,b,c,d,y)
A = [alpha, b'; c, diag(d)];
x = A\y;
```

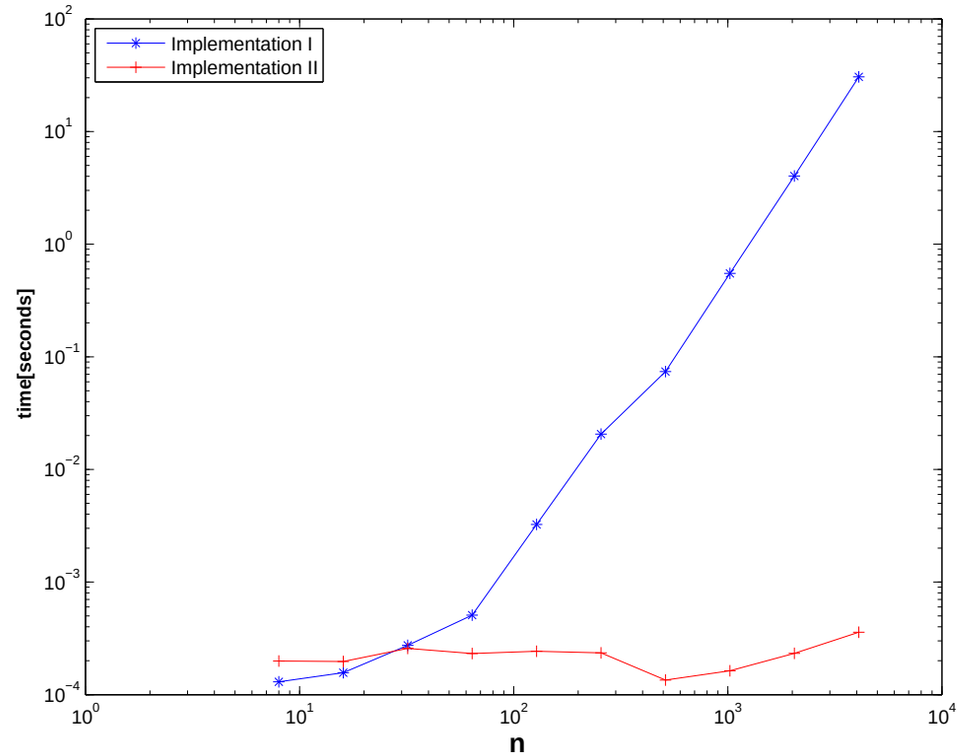
"structure aware" implementation:

Code 2.6.30: LSE with arrow matrix, implementation II

```
function x =
  sa2(alpha,b,c,d,y)
z = b./d;
xi = (y(1) -
  dot(z,y(2:end)))...
      /(alpha-dot(z,c));
x = [xi; (y(2:end)-xi*c)./d];
```

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-+');
```



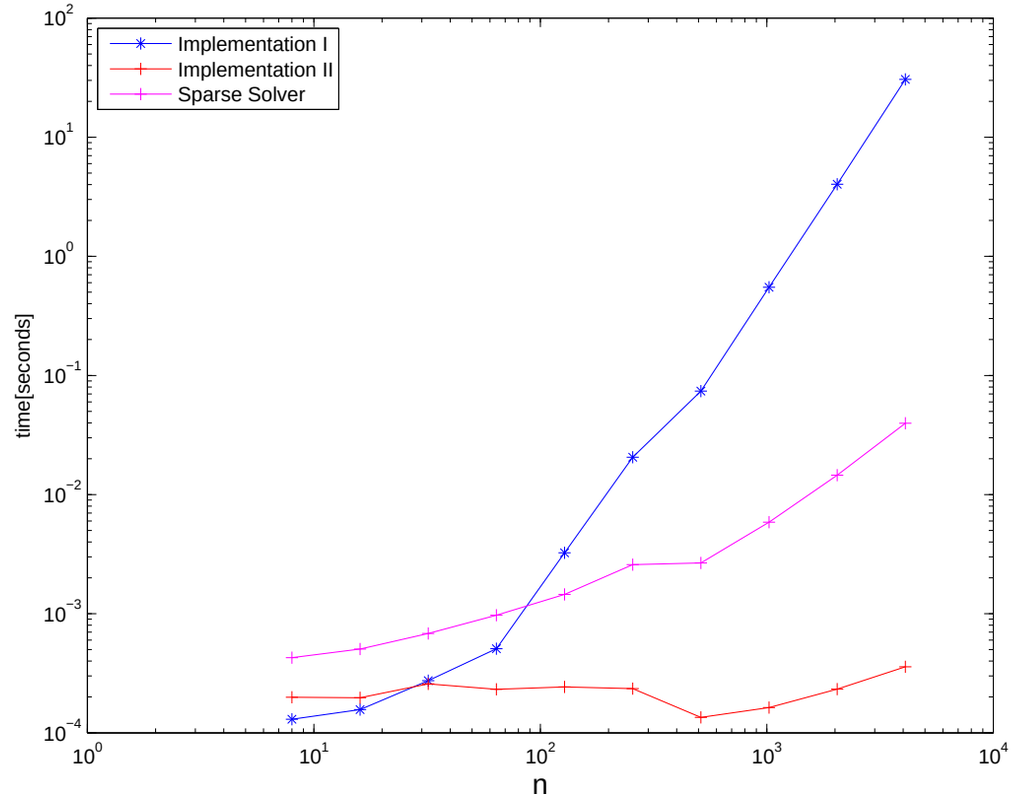
Platform as in Ex. 2.6.11

MATLAB can do much better **!**

Use sparse matrix format:

Code 2.6.31: sparse solver for arrow matrix

```
function x =
  sa3(alpha,b,c,d,y)
n = length(d);
A = [alpha, b';...
      c,spdiags(d,0,n,n)];
x = A\y;
```



◇

R. Hiptmair

rev 38355, October 7, 2011

Exploit structure of (sparse) linear systems of equations **!**

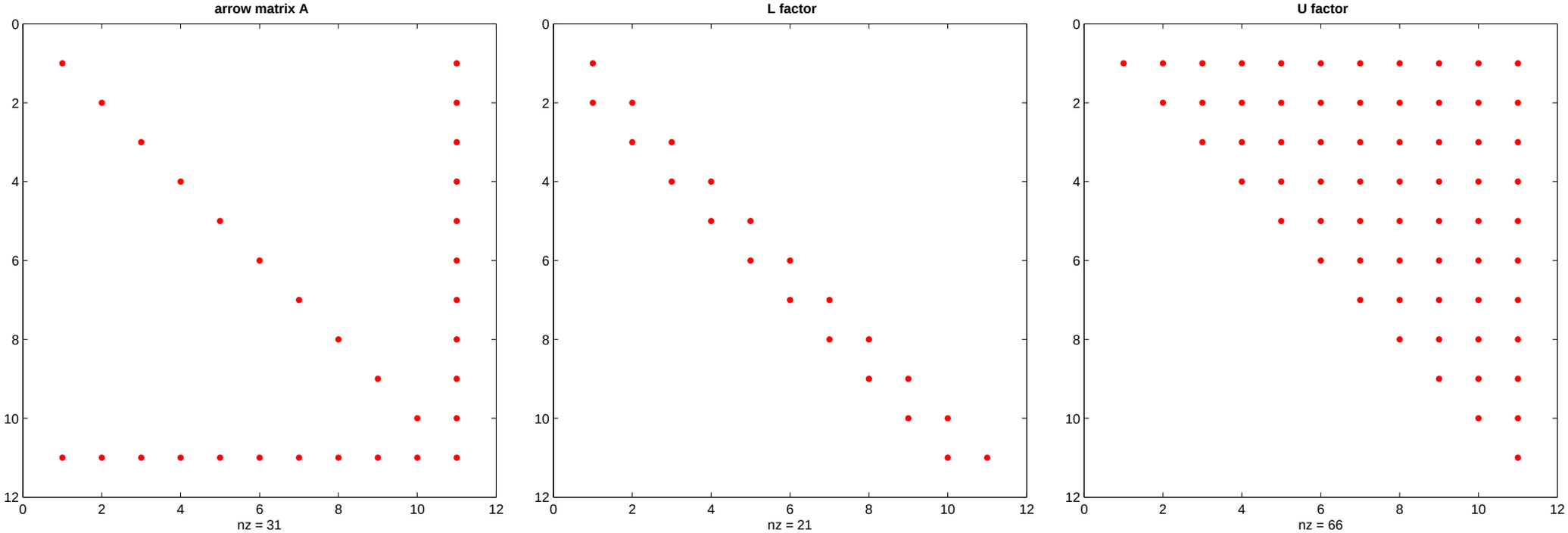
Caution: stability at risk

Example 2.6.32 (Pivoting destroys sparsity).

Code 2.6.33: fill-in due to pivoting

```
% Study of fill-in with LU-factorization due to pivoting
n = 10; D = diag(1./(1:n));
A = [ D , 2*ones(n,1); 2*ones(1,n), 2];
[L,U,P] = lu(A);
figure; spy(A,'r'); title('{\bf arrow matrix A}');
print -depsc2 '../PICTURES/fillinpivotA.eps';
figure; spy(L,'r'); title('{\bf L factor}');
print -depsc2 '../PICTURES/fillinpivotL.eps';
figure; spy(U,'r'); title('{\bf U factor}');
print -depsc2 '../PICTURES/fillinpivotU.eps';
```

$$\mathbf{A} = \begin{pmatrix} 1 & & & & 2 \\ & \frac{1}{2} & & & 2 \\ & & \ddots & & \vdots \\ & & & \frac{1}{10} & 2 \\ 2 & & \dots & & 2 \end{pmatrix} \quad \rightarrow \quad \text{arrow matrix, Ex. 2.6.21}$$



A **L** **U**

◇

R. Hiptmair

rev 38355, October 7, 2011

2.6.4 Banded matrices [13, Sect. 3.7]

Definition 2.6.34 (bandwidth)**.**
For $\mathbf{A} = (a_{ij})_{i,j} \in \mathbb{K}^{m,n}$ *we call*

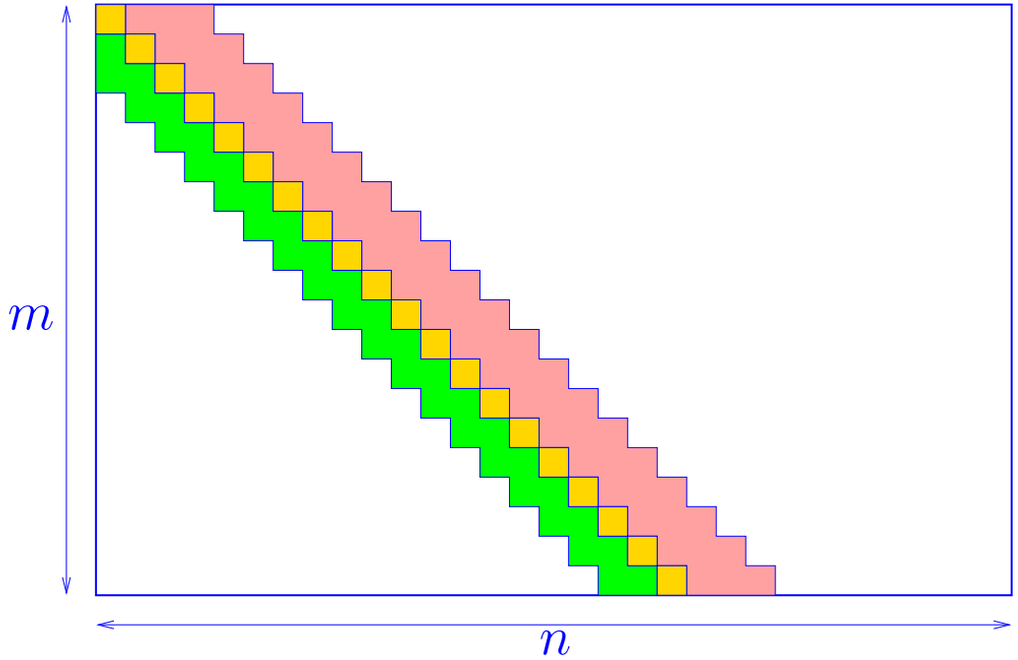
$$\overline{m}(\mathbf{A}) := \min\{k \in \mathbb{N}\colon j - i > k \Rightarrow a_{ij} = 0\} \text{ upper bandwidth},$$
$$\underline{m}(\mathbf{A}) := \min\{k \in \mathbb{N}\colon i - j > k \Rightarrow a_{ij} = 0\} \text{ lower bandwidth}.$$

$m(\mathbf{A}) := \overline{m}(\mathbf{A}) + \underline{m}(\mathbf{A}) + 1$ = *bandwidth von* $\mathbf{A}$ *(*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 $m(\mathbf{A}) \ll n \mathrel{\hat{=}}$ banded matrix

R. Hiptmair

rev 38355, October 7, 2011



: diagonal

: super-diagonals

: sub-diagonals

◁ $\overline{m}(\mathbf{A}) = 3$, $\underline{m}(\mathbf{A}) = 2$

▶ for banded matrix $\mathbf{A} \in \mathbb{K}^{m,n}$: $\operatorname{nnz}(\mathbf{A}) \leq \min\{m,n\} m(\mathbf{A})$

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 !)

Definition 2.6.35 (Matrix envelope (*ger.:* Hülle))**.**

For $\mathbf{A} \in \mathbb{K}^{n,n}$ *define*

row bandwidth $m_i^R(\mathbf{A}) := \max\{0, i-j : a_{ij} \neq 0, 1 \leq j \leq n\}, i \in \{1, ..., n\}$

column bandwidth $m_j^C(\mathbf{A}) := \max\{0, j-i : a_{ij} \neq 0, 1 \leq i \leq n\}, j \in \{1, ..., n\}$

envelope $\operatorname{env}(\mathbf{A}) := \left\{ (i,j) \in \{1,\dots,n\}^2 : \begin{array}{c} i - m_i^R(\mathbf{A}) \leq j \leq i \ , \\ j - m_j^C(\mathbf{A}) \leq i \leq j \end{array} \right\}$

Example 2.6.36 (Envelope of a matrix)*.*

$$\mathbf{A} = \begin{pmatrix} * & 0 & * & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & * & 0 & 0 \\ * & 0 & * & 0 & 0 & 0 & * \\ 0 & 0 & 0 & * & * & 0 & * \\ 0 & * & 0 & * & * & * & 0 \\ 0 & 0 & 0 & 0 & * & * & 0 \\ 0 & 0 & * & * & 0 & 0 & * \end{pmatrix} \quad \begin{array}{l} m_1^R(A) = 0 \\ m_2^R(A) = 0 \\ m_3^R(A) = 2 \\ m_4^R(A) = 0 \\ m_5^R(A) = 3 \\ m_6^R(A) = 1 \\ m_7^R(A) = 4 \end{array}$$

$\operatorname{env}(A)$ = red elements

$*$ ≙ non-zero matrix entry $a_{ij} \neq 0$

R. Hiptmair

rev 38355, October 7, 2011

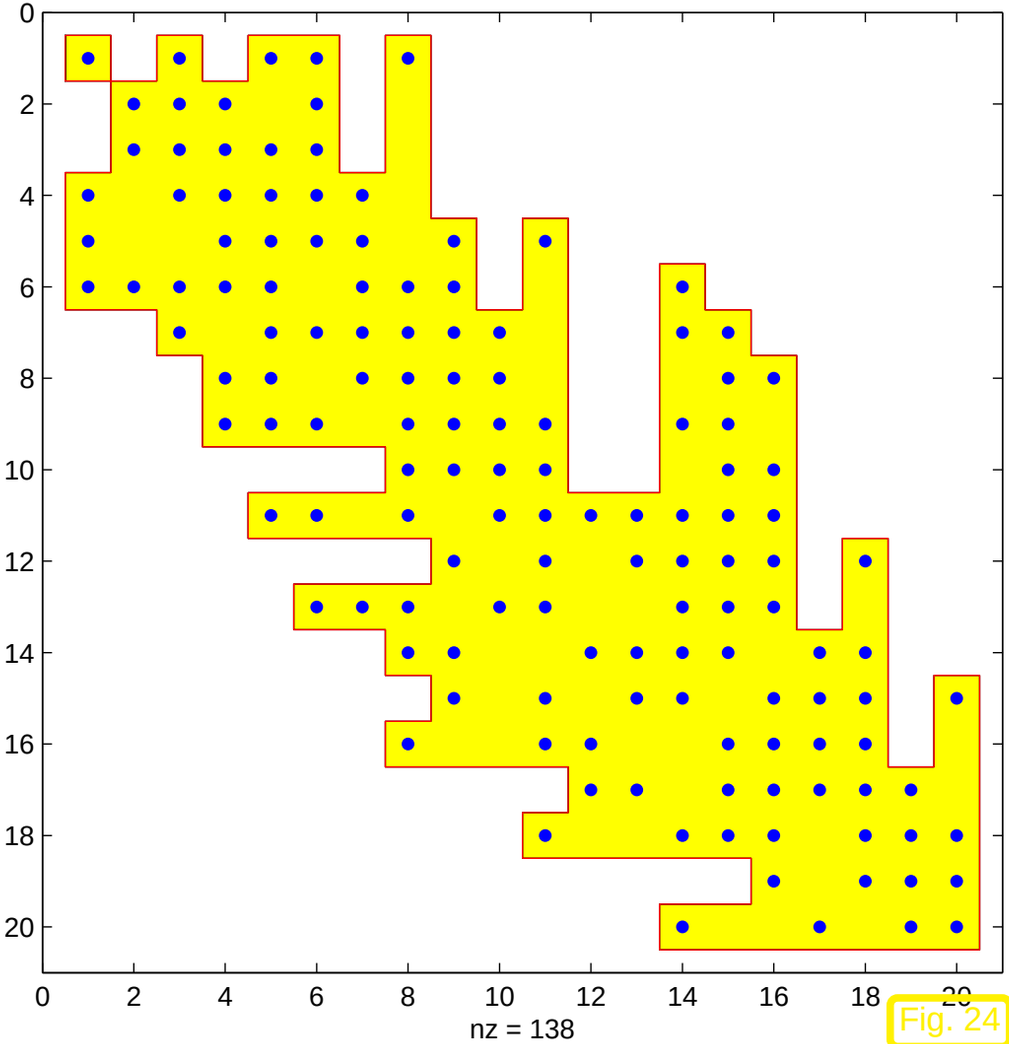


Fig. 24

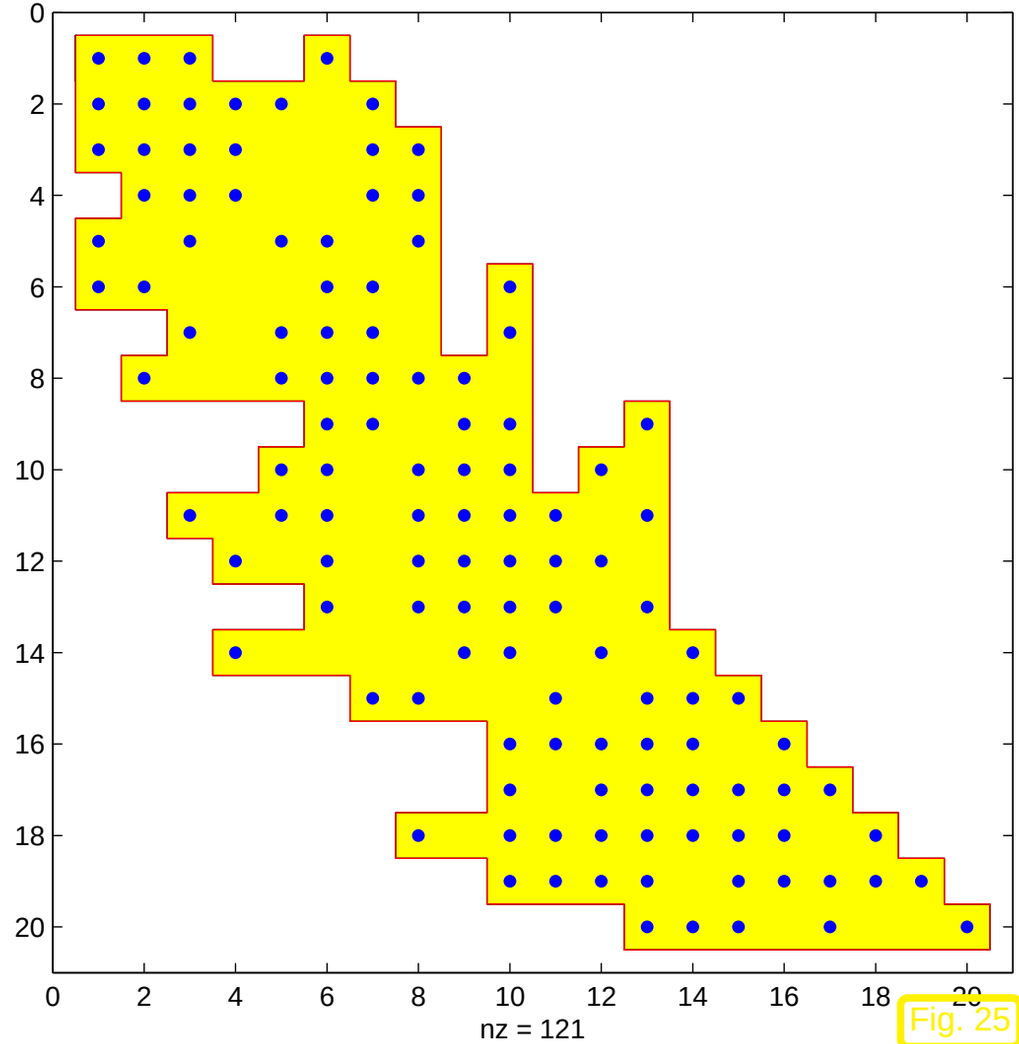


Fig. 25

◇

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* $\mathrm{env}(\mathbf{A})$*.*

Gaussian elimination *without pivoting*

Envelope-aware LU-factorization:

Minimizing bandwidth/envelope:

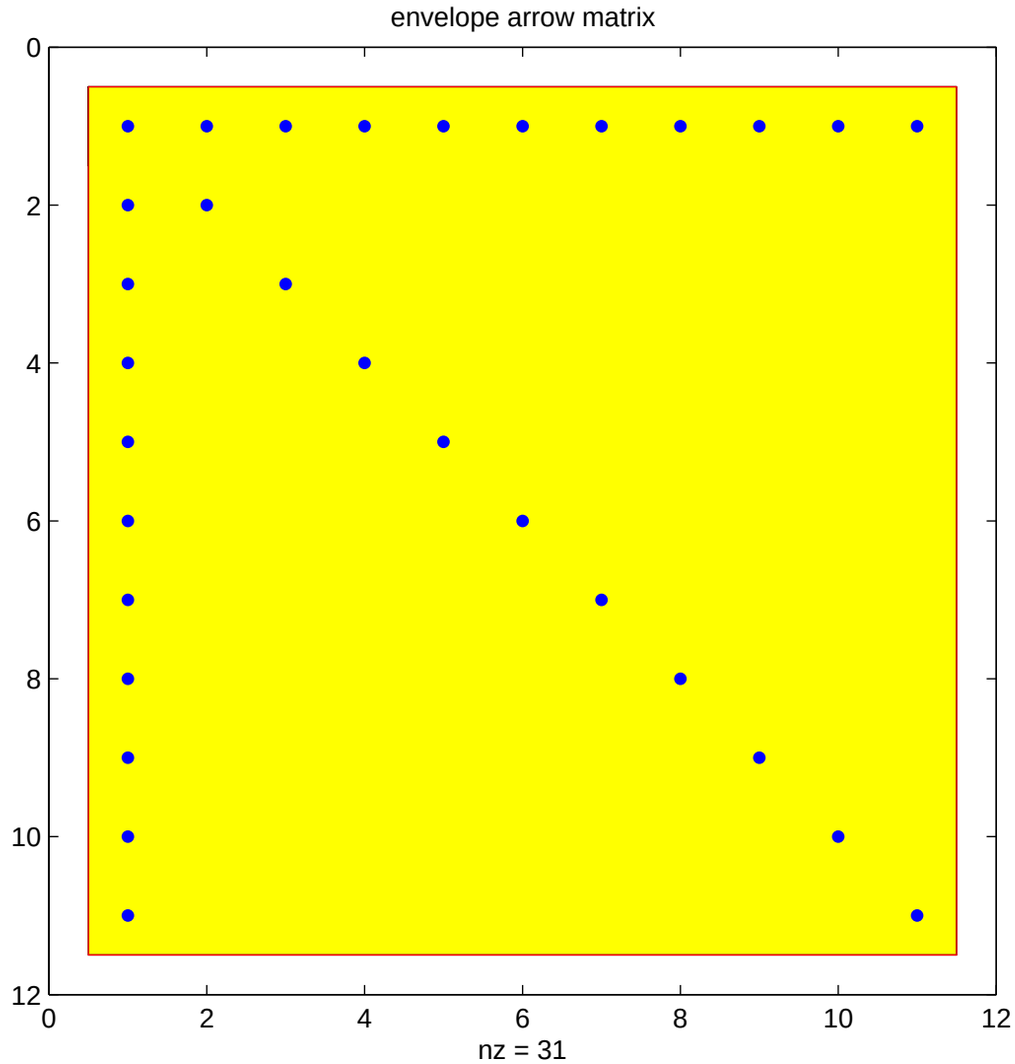
Goal: Minimize $m_i(\mathbf{A})$,$\mathbf{A} = (a_{ij}) \in \mathbb{R}^{N,N}$, by permuting rows/columns of $\mathbf{A}$

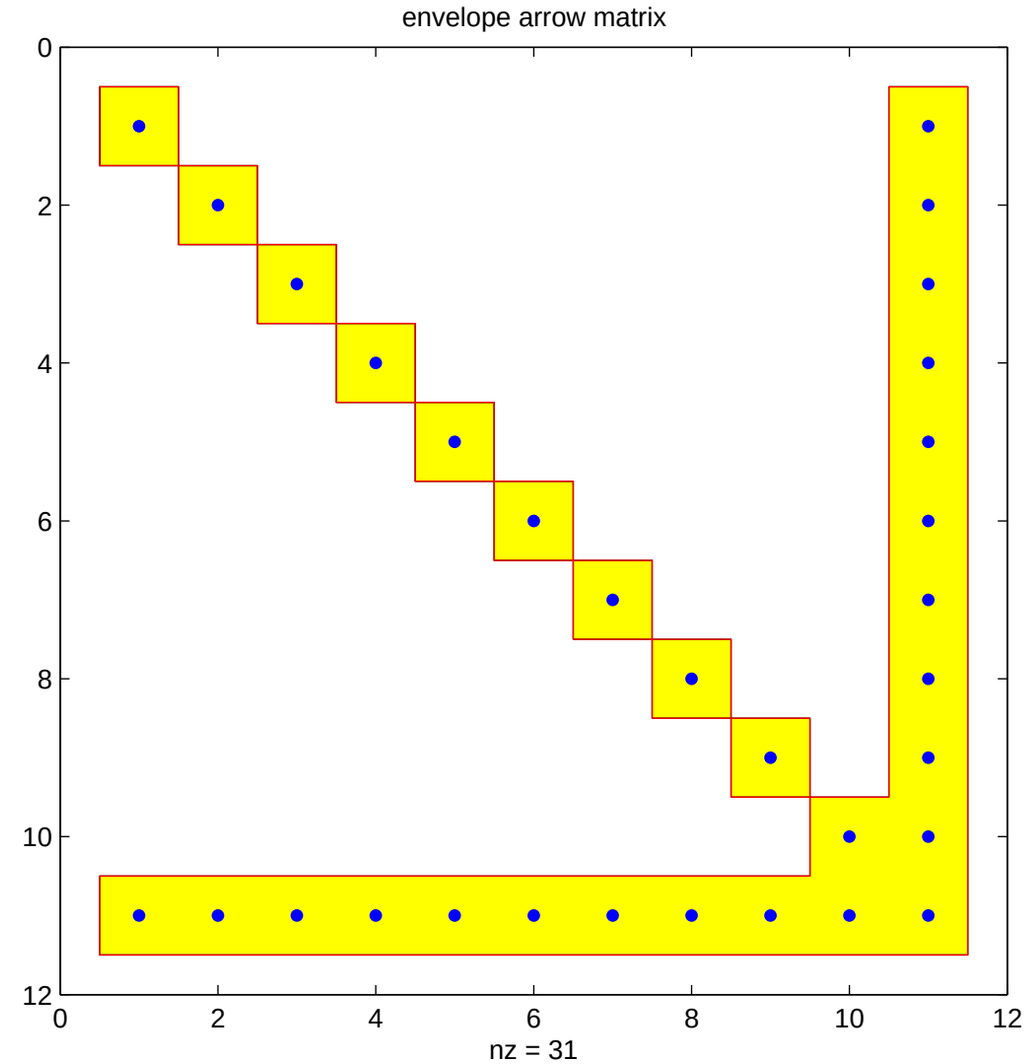
Example 2.6.47 (Reducing bandwidth by row/column permutations)*.*

Recall: cyclic permutation of rows/columns of arrow matrix $\rightarrow$ Ex. 2.6.22

R. Hiptmair

rev 38355, October 7, 2011





Another example:Reflection at cross diagonal ➤ reduction of $\#\,\mathrm{env}(\mathbf{A})$

$$\begin{pmatrix} * & 0 & 0 & * & * & * \\ 0 & * & 0 & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & 0 \\ * & 0 & 0 & * & * & * \\ * & 0 & 0 & * & * & * \\ * & 0 & 0 & * & * & * \end{pmatrix} \longrightarrow \begin{pmatrix} * & * & * & 0 & 0 & * \\ * & * & * & 0 & 0 & * \\ * & * & * & 0 & 0 & * \\ 0 & 0 & 0 & * & 0 & 0 \\ 0 & 0 & 0 & 0 & * & 0 \\ * & * & * & 0 & 0 & * \end{pmatrix}$$

R. Hiptmair

rev 38355, October 7, 2011

$$i \leftarrow N+1-i$$

$$\#\,\mathrm{env}(\mathbf{A}) = 30 \qquad\qquad \#\,\mathrm{env}(\mathbf{A}) = 22$$

◇

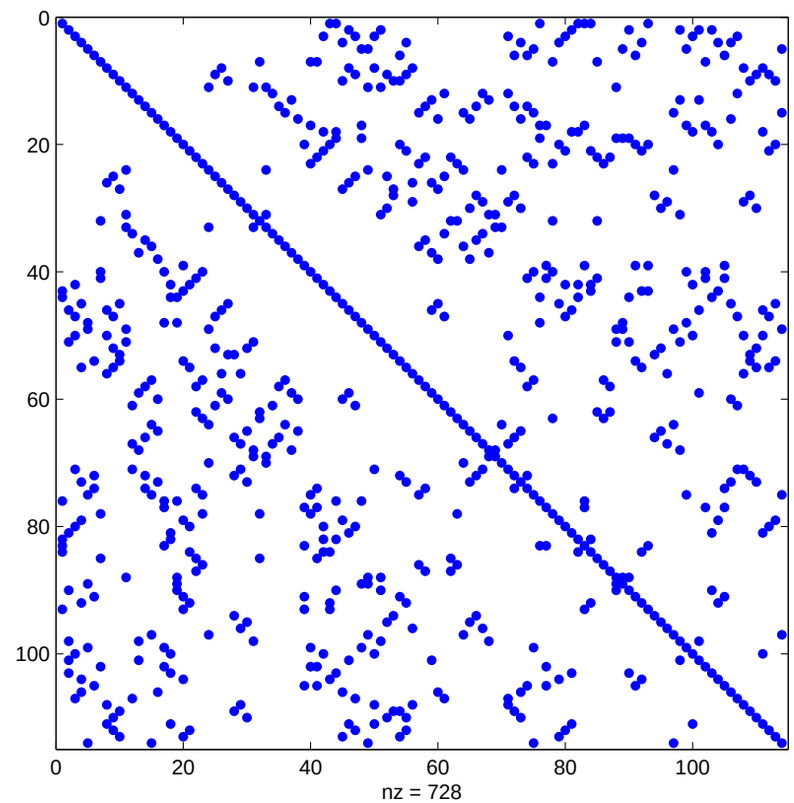
Example 2.6.48 (Reducing fill-in by reordering).

$\mathbf{M}$: 114×114 symmetric matrix (from computational PDEs)

Code 2.6.49: preordering in MATLAB

```
spy(M);
[L,U] = lu(M); spy(U);
r = symrcm(M);
[L,U] = lu(M(r,r)); spy(U);
m = symamd(M);
[L,U] = lu(M(m,m)); spy(U);
```

Pattern of $\mathbf{M}$ ➥

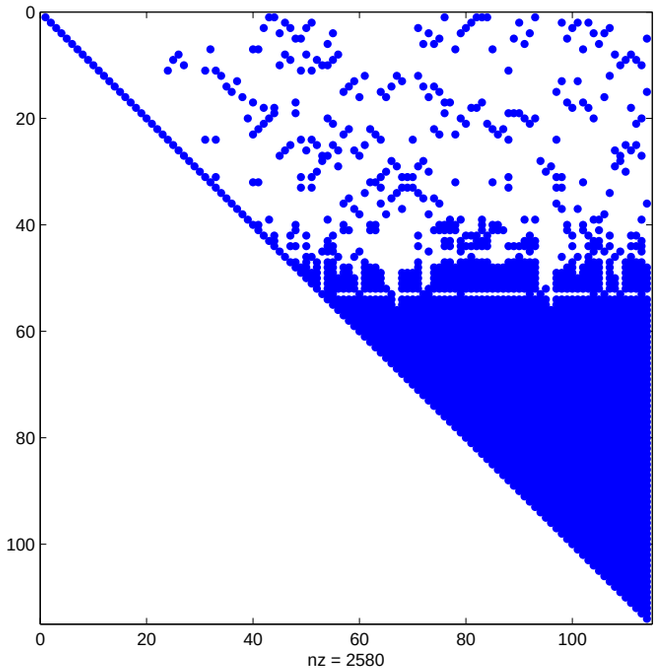


(Here: no row swaps from pivoting !)

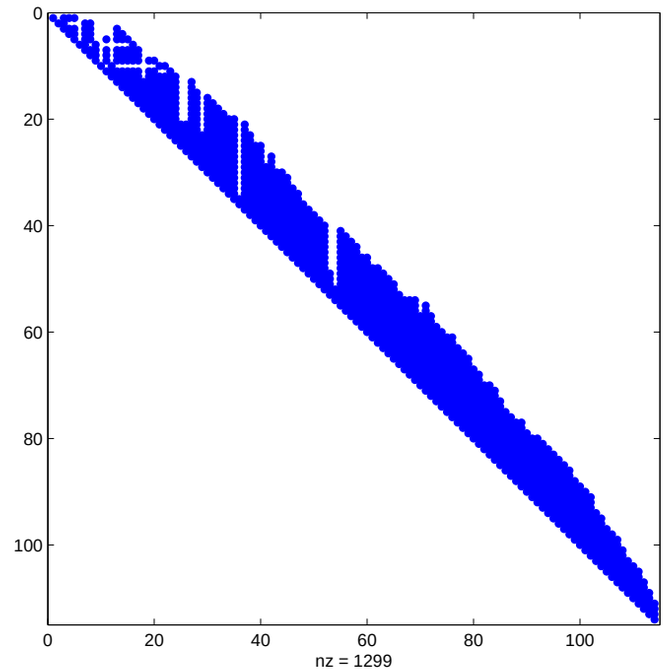
Examine patterns of LU-factors (→ Sect. 2.2) after reordering:

R. Hiptmair

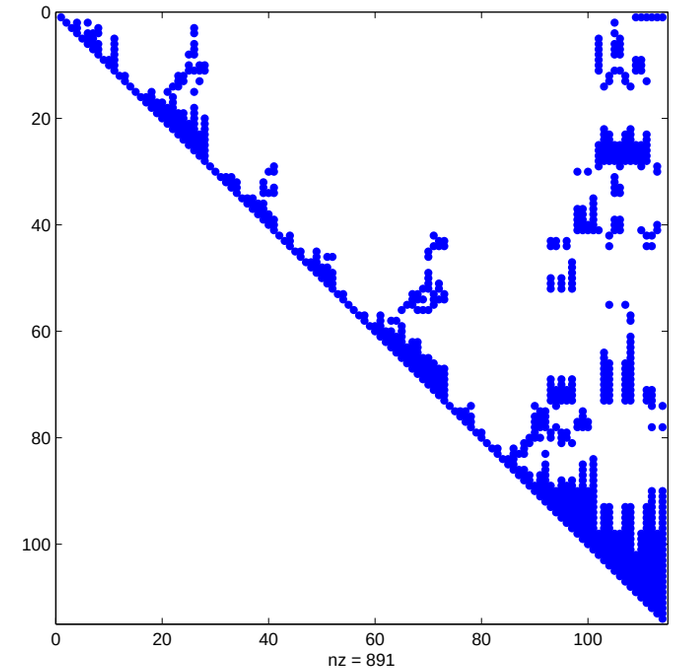
rev 38355, October 7, 2011



no reordering



reverse Cuthill-McKee



approximate minimum degree

◇

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/`)

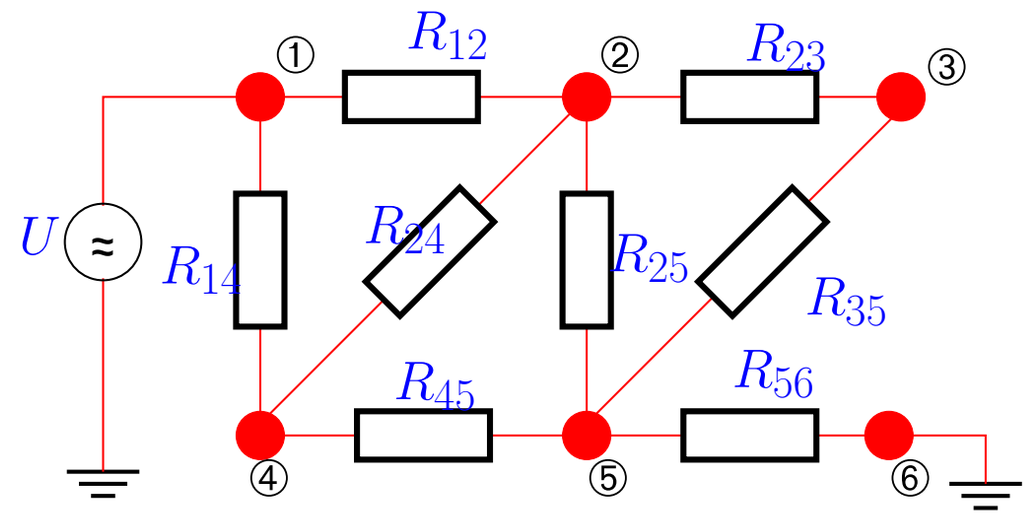
→ Matlab-\ (on sparse storage formats)

2.7 Stable Gaussian elimination without pivoting

Example 2.7.1 (Diagonally dominant matrices from nodal analysis). $\rightarrow$ Ex. 2.6.3

Consider:

electrical circuit entirely composed of Ohmic resistors.



Circuit equations from nodal analysis, see Ex. 2.6.3:

$$
\begin{aligned}
② : \quad & R_{12}^{-1}(U_2-U_1)+R_{23}^{-1}(U_2-U_3)-R_{24}^{-1}(U_2-U_4)+R_{25}^{-1}(U_2-U_5) &= 0\,,\\
③ : \quad & R_{23}^{-1}(U_3-U_2)+R_{35}^{-1}(U_3-U_5) &= 0\,,\\
④ : \quad & R_{14}^{-1}(U_4-U_1)-R_{24}^{-1}(U_4-U_2)+R_{45}^{-1}(U_4-U_5) &= 0\,,\\
⑤ : \quad & R_{25}^{-1}(U_5-U_2)+R_{35}^{-1}(U_5-U_3)+R_{45}^{-1}(U_5-U_4)+R_{56}(U_5-U_6) &= 0\,,\\
& U_1 = U \quad , \quad U_6 = 0\,.
\end{aligned}
$$

R. Hiptmair

rev 38355, October 7, 2011

$$\begin{pmatrix} \frac{1}{R_{12}}+\frac{1}{R_{23}}+\frac{1}{R_{24}}+\frac{1}{R_{25}} & -\frac{1}{R_{23}} & -\frac{1}{R_{24}} & -\frac{1}{R_{25}} \\ -\frac{1}{R_{23}} & \frac{1}{R_{23}}+\frac{1}{R_{35}} & 0 & -\frac{1}{R_{35}} \\ -\frac{1}{R_{24}} & 0 & \frac{1}{R_{24}}+\frac{1}{R_{45}} & -\frac{1}{R_{45}} \\ -\frac{1}{R_{25}} & -\frac{1}{R_{35}} & -\frac{1}{R_{45}} & \frac{1}{R_{22}}+\frac{1}{R_{35}}+\frac{1}{R_{45}}+\frac{1}{R_{56}} \end{pmatrix} \begin{pmatrix} U_2 \\ U_3 \\ U_4 \\ U_5 \end{pmatrix} = \begin{pmatrix} \frac{1}{R_{12}} \\ 0 \\ \frac{1}{R_{14}} \\ 0 \end{pmatrix} U$$

$\diamond$

Definition 2.7.7 (Diagonally dominant matrix)**.** $\rightarrow$ *[51, Def. 1.24]*

$\mathbf{A} \in \mathbb{K}^{n,n}$ *is diagonally dominant, if*

$$\forall k \in \{1,\dots,n\}: \quad \sum_{j \neq k} |a_{kj}| \leq |a_{kk}| \ .$$

The matrix $\mathbf{A}$ *is called strictly diagonally dominant, if*

$$\forall k \in \{1,\dots,n\}: \quad \sum_{j \neq k} |a_{kj}| < |a_{kk}| \ .$$

R. Hiptmair

rev 38355, October 7, 2011

Lemma 2.7.8 (LU-factorization of diagonally dominant matrices)**.**

$$\mathbf{A} \quad \begin{array}{c}\textit{regular, diagonally dominant}\\ \textit{with positive diagonal}\end{array} \quad \Leftrightarrow \quad \left\{ \begin{array}{c} \mathbf{A} \quad \textit{has LU-factorization} \\ \Updownarrow \\ \textit{Gaussian elimination feasible without pivoting}^{(*)} \end{array} \right.$$

Proof. (2.1.11) $\rightarrow$ induction w.r.t. n

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*

$$\mathbf{M} = \mathbf{M}^{\mathsf{H}} \quad \wedge \quad \mathbf{x}^{\mathsf{H}}\mathbf{M}\mathbf{x} > 0 \quad \Leftrightarrow \quad \mathbf{x} \neq 0 \; .$$

If $\mathbf{x}^{\mathsf{H}}\mathbf{M}\mathbf{x} \geq 0$ *for all* $\mathbf{x} \in \mathbb{K}^n$ $\quad \triangleright \quad$ $\mathbf{M}$ *positive semi-definite.*

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.*

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).

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* $r_{ii} > 0$, $i = 1, \ldots, n$, *such that* $\mathbf{A} = \mathbf{R}^{\mathsf{H}}\mathbf{R}$ *(Cholesky decomposition).*

MATLAB function:

```
R = chol(A)
```

Gaussian elimination *without pivoting* is a *numerically stable* way to solve LSEs with s.p.d. system 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)

$$F : \begin{cases} \mathbb{K}^n & \mapsto \mathbb{K}^n \\ \mathbf{x} & \mapsto \mathbf{A}\mathbf{x} \end{cases}$$

for *given* regular $\mathbf{A} \in \mathbb{K}^{n,n}$ $\succ$ $\mathbf{x} \hat{=}$ "data"

$$\Rightarrow \quad \frac{\|\Delta\mathbf{y}\|}{\|\mathbf{y}\|} \leq \frac{\|\mathbf{A}\|\,\|\Delta\mathbf{x}\|}{\left\|\mathbf{A}^{-1}\right\|^{-1}\|\mathbf{x}\|} = \mathrm{cond}(\mathbf{A})\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \,. \tag{2.8.2}$$

relative perturbation in result

relative perturbation in data

$\succ$ Condition number $\mathrm{cond}(\mathbf{A})$ ($\rightarrow$ Def. 2.5.26) bounds amplification of relative error in argument vector in matrix$\times$vector-multiplication $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$.

$\triangle$

Example 2.8.3 (Conditioning of row transformations).

2×2 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 $\mathbf{T}(\mu)$ ▷

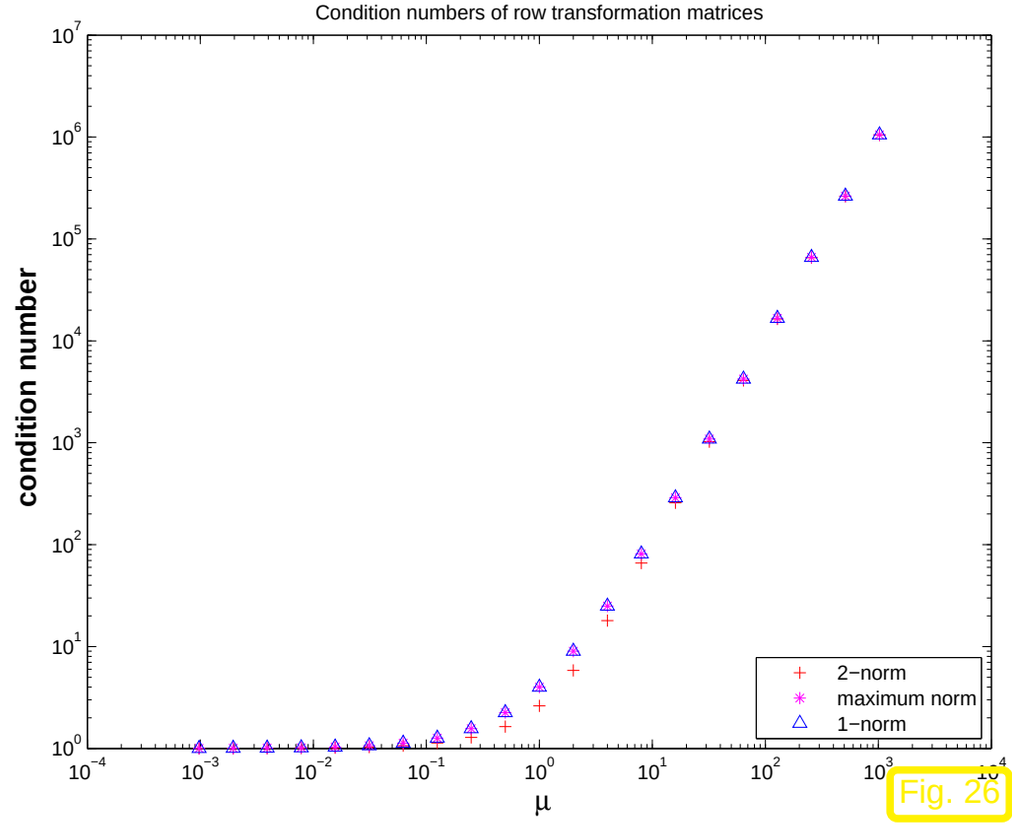


Fig. 26

◇

Goal: find unitary row transformation rendering certain matrix elements zero.

$$\mathbf{Q}\left(\begin{array}{c}\blacksquare\\ \blacksquare\end{array}\right) = \left(\begin{array}{c}\blacksquare\\ 0\ \ \blacksquare\end{array}\right) \quad \text{with} \quad \mathbf{Q}^H = \mathbf{Q}^{-1}\,.$$

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:

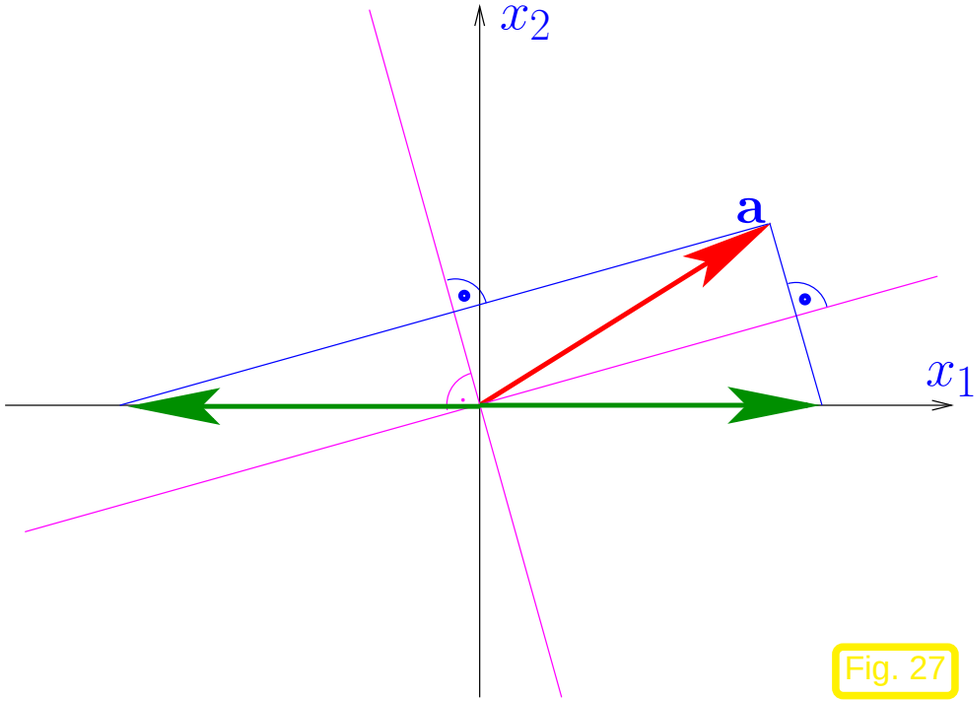


Fig. 27

reflection at angle bisector,

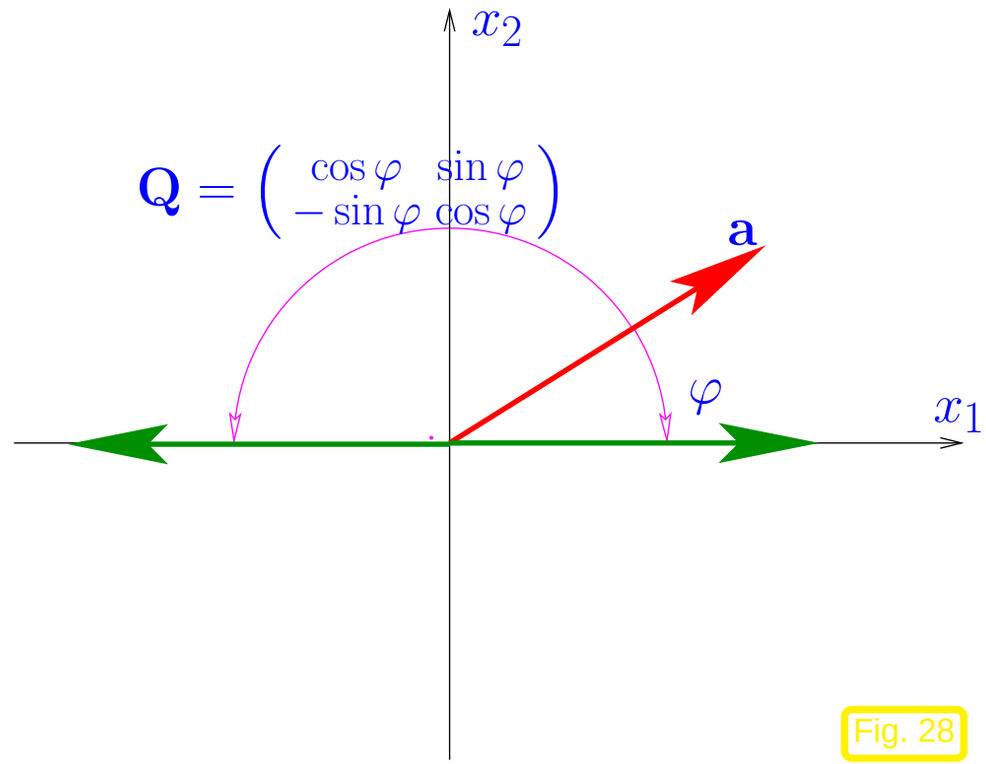


Fig. 28

rotation turning $\mathbf{a}$ onto x_1-axis.

➢ Note: two possible reflections/rotations ◇

In nD: given $\mathbf{a} \in \mathbb{R}^n$ find orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n,n}$ such that $\mathbf{Qa} = \|\mathbf{a}\|_2\, \mathbf{e}_1$, $\mathbf{e}_1 \hat{=}$ 1st unit vector.

Choice 1: Householder reflections

$$\mathbf{Q} = \mathbf{H}(\mathbf{v}) := \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^H}{\mathbf{v}^H\mathbf{v}} \quad \text{with} \quad \mathbf{v} = \tfrac{1}{2}(\mathbf{a} \pm \|\mathbf{a}\|_2\,\mathbf{e}_1)\ . \tag{2.8.8}$$

Choice 2: successive Givens rotations [35, Sect. 14] ($\rightarrow$ 2D case)

$$\mathbf{G}_{1k}(a_1,a_k)\mathbf{a} := \begin{pmatrix} \bar{\gamma} & \cdots & \bar{\sigma} & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots \\ -\sigma & \cdots & \gamma & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_k \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_1^{(1)} \\ \vdots \\ 0 \\ \vdots \\ a_n \end{pmatrix} \quad \text{, if} \quad \begin{aligned} \gamma &= \frac{a_1}{\sqrt{|a_1|^2+|a_k|^2}}\ , \\ \sigma &= \frac{a_k}{\sqrt{|a_1|^2+|a_k|^2}}\ . \end{aligned} \tag{2.8.10}$$

MATLAB-Function: `[G,x] = planerot(a);`

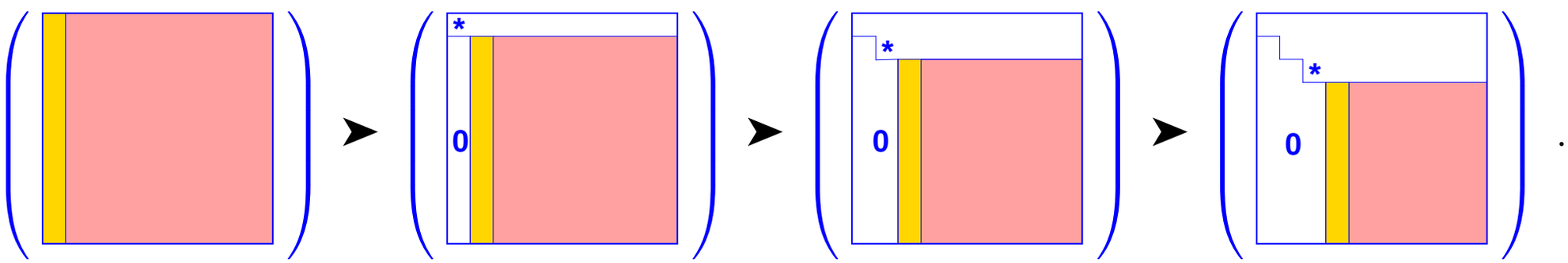
Code 2.8.11: (plane) Givens rotation

```
function [G,x] = planerot(a)
% plane Givens rotation.
if (a(2) ~= 0)
  r = norm(a);  G = [a'; -a(2) a(1)]/r; x = [r; 0];
else, G = eye(2); end
```

Mapping $\mathbf{a} \in \mathbb{K}^n$ to a multiple of $\mathbf{e}_1$ by $n-1$ *successive* Givens rotations:

$$\begin{pmatrix} a_1 \\ \vdots \\ \vdots \\ \vdots \\ a_n \end{pmatrix} \xrightarrow{\mathbf{G}_{12}(a_1,a_2)} \begin{pmatrix} a_1^{(1)} \\ 0 \\ a_3 \\ \vdots \\ a_n \end{pmatrix} \xrightarrow{\mathbf{G}_{13}(a_1^{(1)},a_3)} \begin{pmatrix} a_1^{(2)} \\ 0 \\ 0 \\ a_4 \\ \vdots \\ a_n \end{pmatrix} \xrightarrow{\mathbf{G}_{14}(a_1^{(2)},a_4)} \cdots \xrightarrow{\mathbf{G}_{1n}(a_1^{(n-2)},a_n)} \begin{pmatrix} a_1^{(n-1)} \\ 0 \\ \vdots \\ \\ \vdots \\ 0 \end{pmatrix} \tag{2.8.12}$$

Transformation to *upper triangular form* ($\rightarrow$ Def. 2.2.3) by successive unitary transformations:



= "target column $\mathbf{a}$" (determines unitary transformation),

= modified in course of transformations.

R. Hiptmair

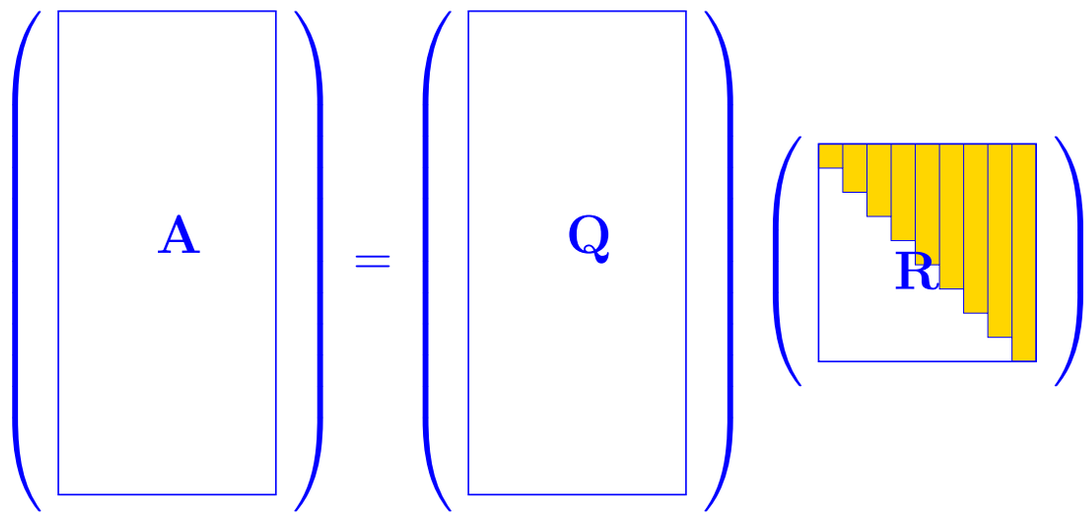
rev 38355, September 7, 2011

$$\mathbf{Q}_{n-1}\mathbf{Q}_{n-2}\cdot\cdots\cdot\mathbf{Q}_1\mathbf{A}=\mathbf{R}\,,$$

QR-factorization (QR-decomposition) of $\mathbf{A}\in\mathbb{C}^{n,n}$: $\mathbf{A}=\mathbf{Q}\mathbf{R}$, $\mathbf{Q}:=\mathbf{Q}_1^H\cdot\cdots\cdot\mathbf{Q}_{n-1}^H$ unitary matrix , $\mathbf{R}$ upper triangular matrix .

Generalization to $\mathbf{A}\in\mathbb{K}^{m,n}$:

$m>n$:



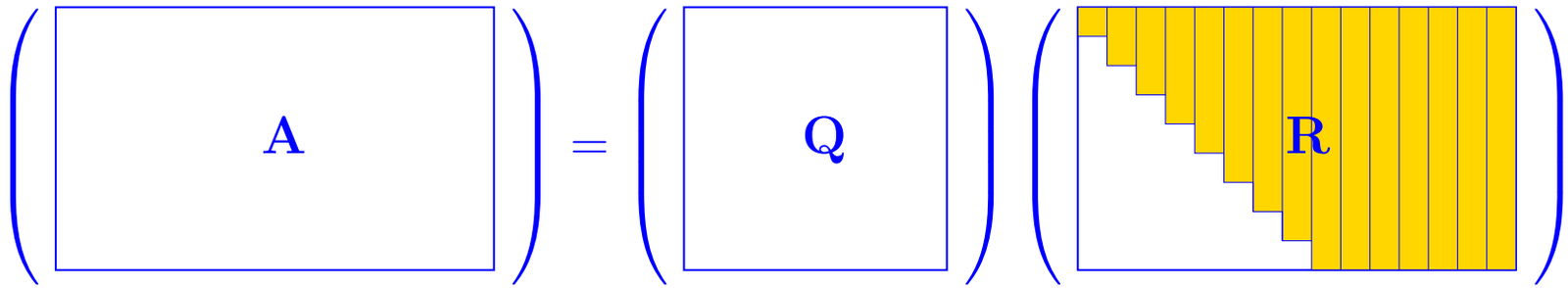
, $\mathbf{A}=\mathbf{Q}\mathbf{R}$, $\mathbf{Q}\in\mathbb{K}^{m,n}$, $\mathbf{R}\in\mathbb{K}^{n,n}$,

(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* $\operatorname{rank}(\mathbf{A})=n$ *is unique, if we demand* $r_{ii}>0$.

$m < n$:



$$\mathbf{A} = \mathbf{Q}\mathbf{R} \quad , \quad \mathbf{Q} \in \mathbb{K}^{m,m} \; , \quad \mathbf{R} \in \mathbb{K}^{m,n} \; ,$$

where $\mathbf{Q}$ unitary, $\mathbf{R}$ upper triangular matrix.

MATLAB functions:

`[Q,R] = qr(A)` $\mathbf{Q} \in \mathbb{K}^{m,m}$, $\mathbf{R} \in \mathbb{K}^{m,n}$ for $\mathbf{A} \in \mathbb{K}^{m,n}$

`[Q,R] = qr(A,0)` $\mathbf{Q} \in \mathbb{K}^{m,n}$, $\mathbf{R} \in \mathbb{K}^{n,n}$ 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] = qr(A)` ➔ Costs: $O(m^2 n)$

`[Q,R] = qr(A,0)` ➔ Costs: $O(mn^2)$

Example 2.8.18 (Complexity of Householder QR-factorization).

Code 2.8.19: timing MATLAB QR-factorizations

```
% Timing QR factorizations

K = 3; r = [];
for n=2.^(2:6)
  m = n*n;

  A = (1:m)'*(1:n) + [eye(n);ones(m-n,n)];
  t1 = 1000; for k=1:K, tic; [Q,R] = qr(A); t1 = min(t1,toc);
    clear Q,R; end
  t2 = 1000; for k=1:K, tic; [Q,R] = qr(A,0); t2 = min(t2,toc);
    clear Q,R; end
  t3 = 1000; for k=1:K, tic; R = qr(A); t3 = min(t3,toc); clear
    R; end
  r = [r; n , m , t1 , t2 , t3];
end
```


R. Hiptmair

rev 38355, September 7, 2011

`tic-toc`-timing of different variants of QR-factorization in MATLAB ▷

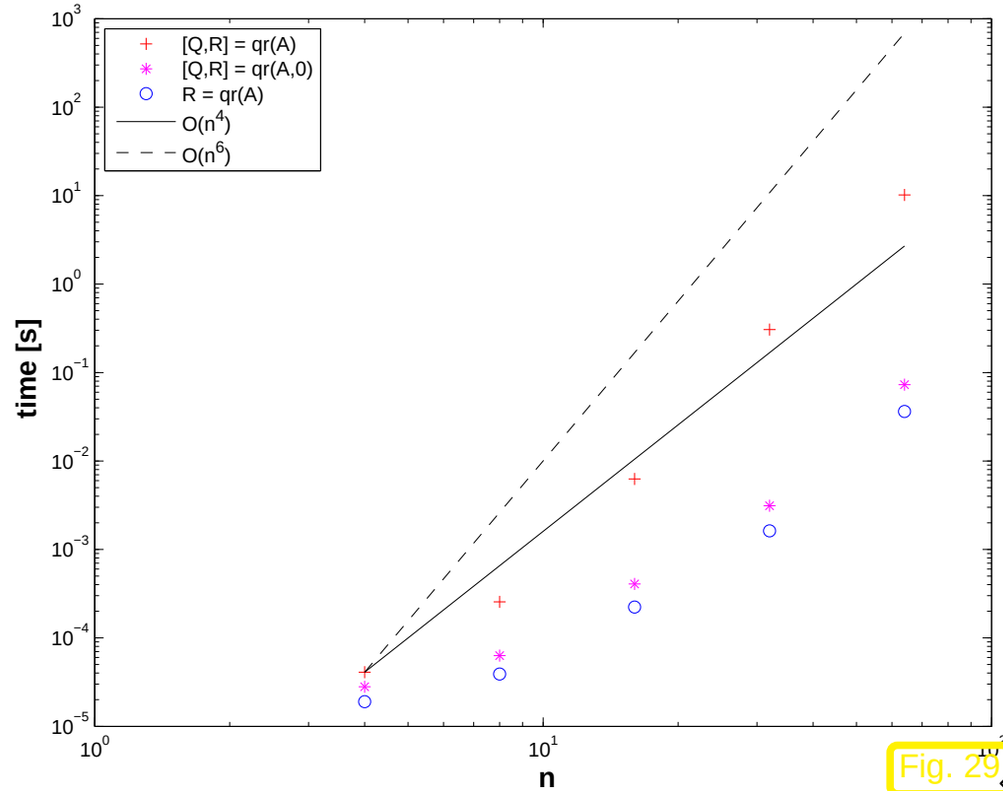
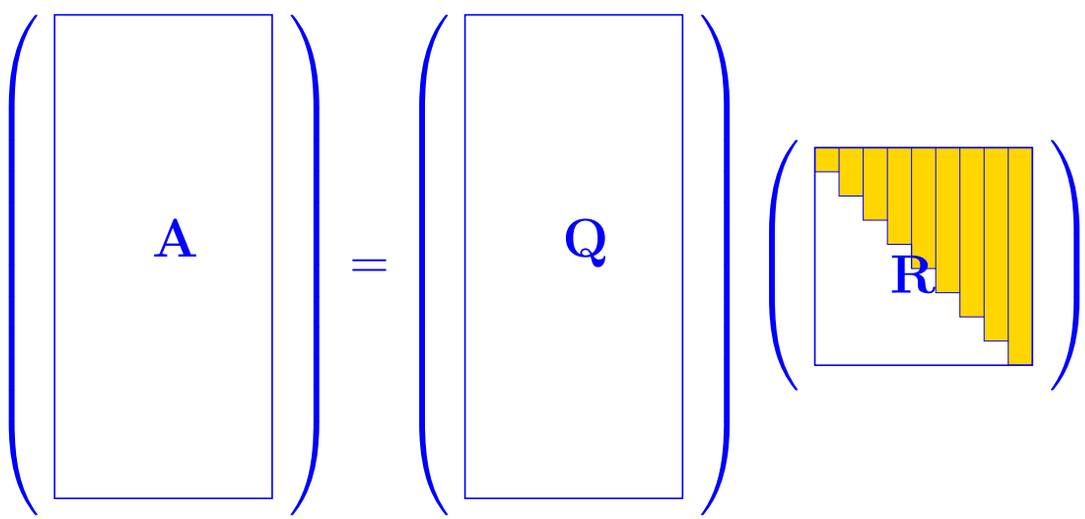


Fig. 29 ◇

▶ Use `[Q,R] = qr(A,0)`, if output sufficient !

Remark 2.8.20 (QR-orthogonalization).



$$, \quad \mathbf{A}, \mathbf{Q} \in \mathbb{K}^{m,n}, \mathbf{R} \in \mathbb{K}^{n,n} \; .$$

R. Hiptmair

rev 38355, September 7, 2011

If $m > n$, $\mathrm{rank}(\mathbf{R}) = \mathrm{rank}(\mathbf{A}) = n$ (full rank)

➤ $\{\mathbf{q}_{\cdot,1}, \dots, \mathbf{q}_{\cdot,n}\}$ is orthonormal basis of $\mathrm{Im}(\mathbf{A})$ with

$$\mathrm{Span}\left\{\mathbf{q}_{\cdot,1}, \dots, \mathbf{q}_{\cdot,k}\right\} = \mathrm{Span}\left\{\mathbf{a}_{\cdot,1}, \dots, \mathbf{a}_{\cdot,k}\right\} \ ,1 \le k \le n \ .$$

△

Algorithm 2.8.22 (Solving linear system of equations by means of QR-decomposition).

$\mathbf{Ax} = \mathbf{b}$:

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)

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 $\mathbf{A} = \mathbf{QR}$ by means of Householder reflections or Givens rotations is (numerically stable) for any $\mathbf{A} \in \mathbb{C}^{m,n}$.

✌ 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). $\longrightarrow$ Ex. 2.5.14

Code 2.8.24: QR-fac. $\leftrightarrow$ Gaussian elimination

```
res = [];
for n=10:10:1000
  A=[tril(-ones(n,n-1))...
       +2*[eye(n-1);...
       zeros(1,n-1)],ones(n,1)];
  x=((-1).^(1:n))';
  b=A*x;
  [Q,R]=qr(A);
  errlu=norm(A\b-x)/norm(x);
  errqr=norm(R\(Q'*b)-x)/norm(x);
  res=[res; n,errlu,errqr];
end
semilogy(res(:,1),res(:,2),'m-*',...
         res(:,1),res(:,3),'b-+');
```

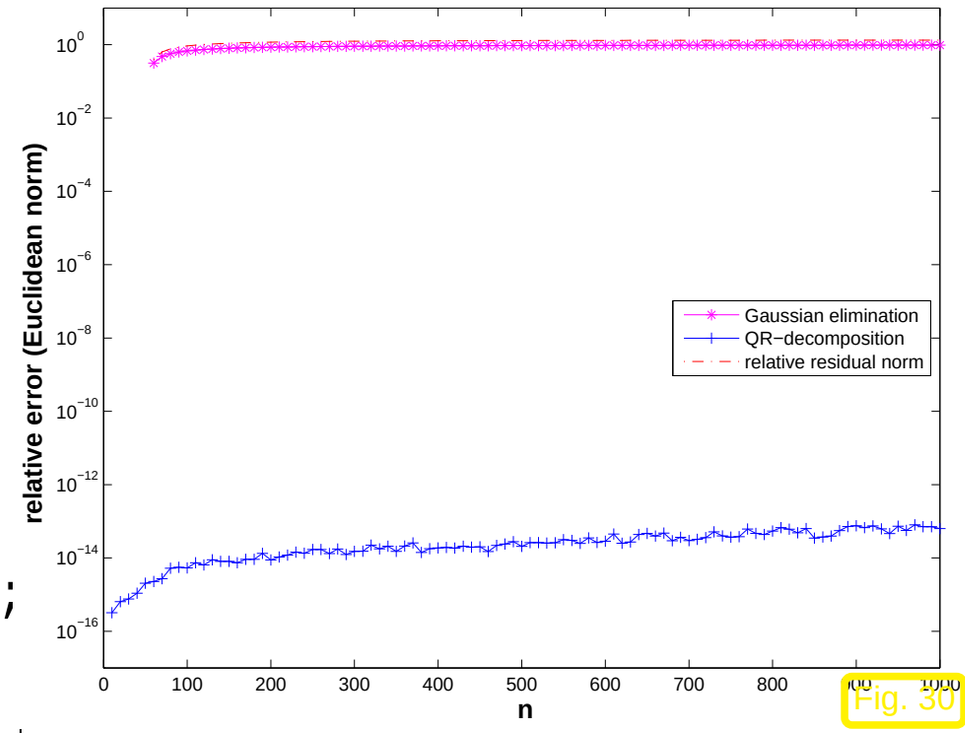


Fig. 30

$\triangleleft$ superior stability of QR-decomposition !

$\diamond$

Fill-in for QR-decomposition **?**

bandwidth

$\mathbf{A} \in \mathbb{C}^{n,n}$ with QR-decomposition $\mathbf{A} = \mathbf{QR}$ $\Rightarrow$ $m(\mathbf{R}) \leq m(\mathbf{A})$ ($\longrightarrow$ Def. 2.6.34)

Example 2.8.25 (QR-based solution of tridiagonal LSE).

Elimination of Sub-diagonals by $n-1$ successive Givens rotations:

$$\begin{pmatrix} * & * & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 & 0 & 0 \\ 0 & * & * & * & 0 & 0 & 0 & 0 \\ 0 & 0 & * & * & * & 0 & 0 & 0 \\ 0 & 0 & 0 & * & * & * & 0 & 0 \\ 0 & 0 & 0 & 0 & * & * & * & 0 \\ 0 & 0 & 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{12}} \begin{pmatrix} * & * & * & 0 & 0 & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 & 0 & 0 & 0 \\ 0 & * & * & * & 0 & 0 & 0 & 0 \\ 0 & 0 & * & * & * & 0 & 0 & 0 \\ 0 & 0 & 0 & * & * & * & 0 & 0 \\ 0 & 0 & 0 & 0 & * & * & * & 0 \\ 0 & 0 & 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{23}} \cdots \xrightarrow{\mathbf{G}_{n-1,n}} \begin{pmatrix} * & * & * & 0 & 0 & 0 & 0 & 0 \\ 0 & * & * & * & 0 & 0 & 0 & 0 \\ 0 & 0 & * & * & * & 0 & 0 & 0 \\ 0 & 0 & 0 & * & * & * & 0 & 0 \\ 0 & 0 & 0 & 0 & * & * & * & 0 \\ 0 & 0 & 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & * \end{pmatrix}$$

MATLAB code (c, d, e, b = column vectors of length n, $n \in \mathbb{N}$, `e(n)`, `c(n)` not used):

$$\mathbf{A} = \begin{pmatrix} d_1 & c_1 & 0 & & \dots & 0 \\ e_1 & d_2 & c_2 & & & \vdots \\ 0 & e_2 & d_3 & c_3 & & \\ \vdots & & \ddots & \ddots & \ddots & c_{n-1} \\ 0 & & \dots & 0 & e_{n-1} & d_n \end{pmatrix} \leftrightarrow \texttt{spdiags([e,d,c],[-1 0 1],n,n)}$$

Code 2.8.26: solving a tridiagonal system by means of QR-decomposition

```
function  y = tridiagqr(c,d,e,b)
n = length(d); t = norm(d)+norm(e)+norm(c);
for k=1:n-1
```

R. Hiptmair

rev 38355, September 7, 2011

```
  [R,z] = planerot([d(k);e(k)]);
  if (abs(z(1))/t < eps), error('Matrix singular'); end;
  d(k) = z(1); b(k:k+1) = R*b(k:k+1);
  Z = R*[c(k), 0;d(k+1), c(k+1)];
  c(k) = Z(1,1); d(k+1) = Z(2,1);
  e(k) = Z(1,2); c(k+1) = Z(2,2);
end
A = spdiags([d,[0;c(1:end-1)],[0;0;e(1:end-2)]],[0 1 2],n,n);
y = A\b;
```

Asymptotic complexity $O(n)$

◇

2.9 ModificationTechniques

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)

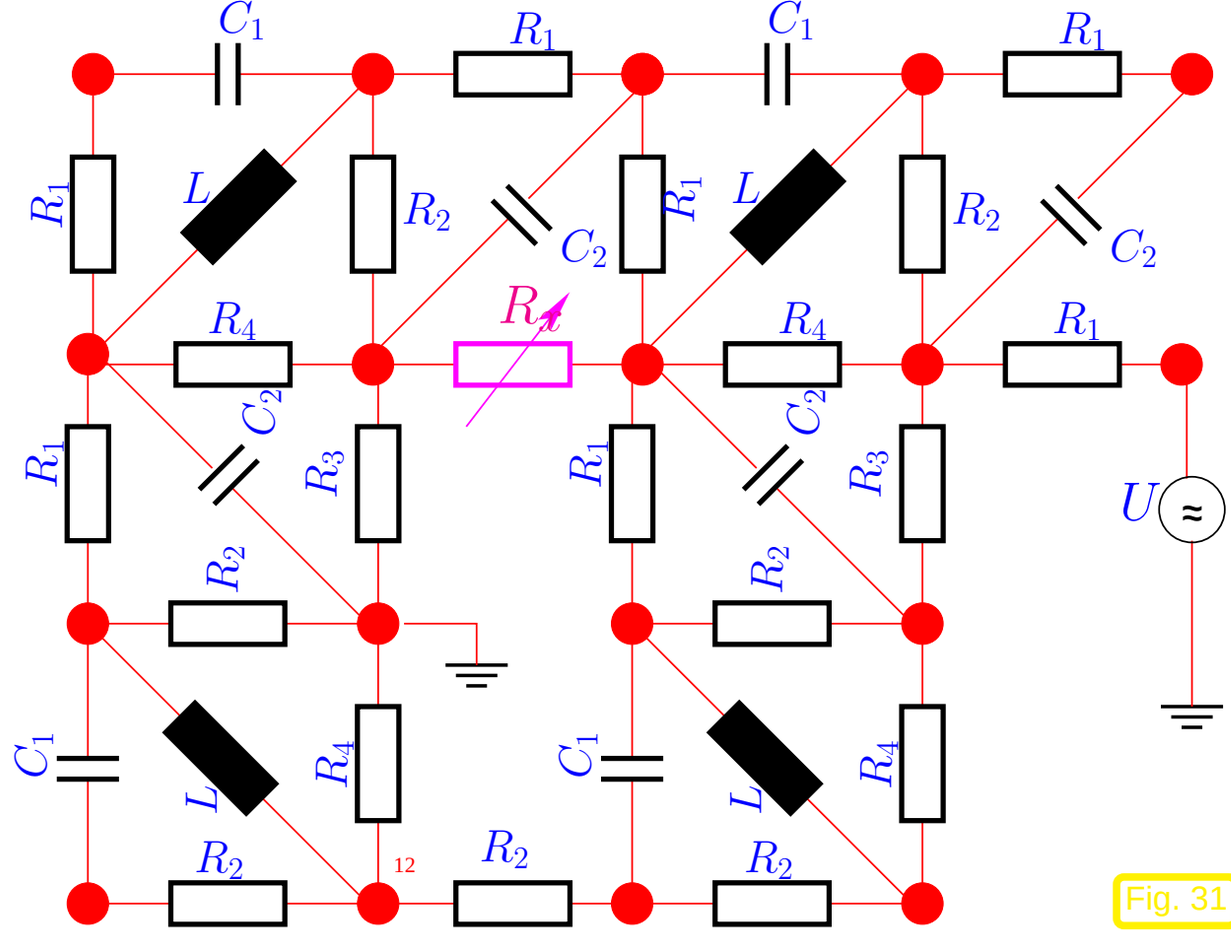


Fig. 31

◇

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 $x \in \mathbb{K}$

$$\mathbf{A}, \widetilde{\mathbf{A}} \in \mathbb{K}^{n,n}: \quad \widetilde{a}_{ij} = \begin{cases} a_{ij} & \text{, if } (i,j) \neq (i^*,j^*) \text{ ,} \\ x + a_{ij} & \text{, if } (i,j) = (i^*,j^*) \text{ ,} \end{cases} \quad , \quad i^*, j^* \in \{1,\dots,n\} \text{ .} \tag{2.9.3}$$

$$\blacktriangleright \quad \boxed{\widetilde{\mathbf{A}} = \mathbf{A} + x \cdot \mathbf{e}_{i^*}\mathbf{e}_{j^*}^T} \text{ .} \tag{2.9.4}$$

Changing a single row: given $\mathbf{x} \in \mathbb{K}^n$

$$\mathbf{A}, \widetilde{\mathbf{A}} \in \mathbb{K}^{n,n}: \quad \widetilde{a}_{ij} = \begin{cases} a_{ij} & \text{, if } i \neq i^* \text{ ,} \\ x_j + a_{ij} & \text{, if } i = i^* \text{ ,} \end{cases} \quad , \quad i^*, j^* \in \{1,\dots,n\} \text{ .}$$

$$\blacktriangleright \quad \boxed{\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{e}_{i^*}\mathbf{x}^T} \text{ .} \tag{2.9.5}$$

$\diamond$

$$\mathbf{A} \in \mathbb{K}^{n,n} \quad \mapsto \quad \widetilde{\mathbf{A}} := \mathbf{A} + \boxed{\mathbf{u}\mathbf{v}^H} \; , \quad \mathbf{u}, \mathbf{v} \in \mathbb{K}^n \; . \tag{2.9.6}$$

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.*

Task: Solve $\widetilde{\mathbf{A}}\mathbf{x} = \mathbf{b}$, 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

```
function x = smw(L,U,u,v,b)
t = L\b; z = U\t;
t = L\u; w = U\t;
alpha = 1+dot(v,w);
if (abs(alpha) <
  eps*norm(U,1)), error('Nearly
  singular matrix'); end;
x = z - w*dot(v,z)/alpha;
```

△

Task: Efficient computation of QR-factorization (→ Sect. 2.8) $\widetilde{\mathbf{A}} = \widetilde{\mathbf{Q}}\widetilde{\mathbf{R}}$ of $\widetilde{\mathbf{A}}$ from (2.9.6), when QR-factorization $\mathbf{A} = \mathbf{Q}\mathbf{R}$ already known

R. Hiptmair

rev 38355, September 7, 2011

① With $\mathbf{w} := \mathbf{Q}^H\mathbf{u}$: $\mathbf{A} + \mathbf{u}\mathbf{v}^H = \mathbf{Q}(\mathbf{R} + \mathbf{w}\mathbf{v}^H)$

➡ Asymptotic complexity $O(n^2)$ (depends on how $\mathbf{Q}$ is stored → Rem. 2.8.21)

② Objective: $\mathbf{w} \to \|\mathbf{w}\|\,\mathbf{e}_1$ ➤ via $n-1$ Givens rotations, see (2.8.10).

$$\mathbf{w} = \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ * \\ * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-1,n}} \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ * \\ 0 \end{pmatrix} \xrightarrow{\mathbf{G}_{n-2,n-1}} \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ 0 \\ 0 \end{pmatrix} \xrightarrow{\mathbf{G}_{n-3,n-2}} \cdots \xrightarrow{\mathbf{G}_{1,2}} \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.9.10}$$

Note: rotations affect $\mathbf{R}$!

$$\mathbf{R} = \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-1,n}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ \vdots & \ddots & & & & \vdots & \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-2,n-1}}$$

R. Hiptmair

rev 38355, September 7, 2011

$$\longrightarrow \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-3,n-2}} \cdots \xrightarrow{\mathbf{G}_{1,2}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ * & * & \cdots & * & * & * & * \\ & & \ddots & & & & \vdots \\ 0 & \cdots & * & * & * & * & * \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} =: \mathbf{R}_1$$

upper Hessenberg matrix: Entry $(i,j) = 0$, if $i > j+1$.

▶ $\mathbf{A} + \mathbf{u}\mathbf{v}^H = \mathbf{Q}\mathbf{Q}_1^H(\underbrace{\mathbf{R}_1 + \|\mathbf{w}\|_2\,\mathbf{e}_1\mathbf{v}^H}_{\text{upper Hessenberg matrix}})$ with unitary $\mathbf{Q}_1 := \mathbf{G}_{12} \cdot \cdots \cdot \mathbf{G}_{n-1,n}$.

➡ Asymptotic complexity $O(n^2)$

③ Successive Givens rotations: $\mathbf{R}_1 + \|\mathbf{w}\|_2\,\mathbf{e}_1\mathbf{v}^H \mapsto$ upper triangular form

$$\mathbf{R}_1 + \|\mathbf{w}\|_2\,\mathbf{e}_1\mathbf{v}^H = \begin{pmatrix} * & * & \cdots & * & * & * & * \\ * & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & * & * & * & * & * \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{12}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & * & * & * & * & * \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{23}} \cdots$$

$$\xrightarrow{\mathbf{G}_{n-2,n-1}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-1,n}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \end{pmatrix} =: \widetilde{\mathbf{R}} \,. \qquad (2.9.11)$$

➡ Asymptotic complexity $O(n^2)$

$$\mathbf{A} + \mathbf{u}\mathbf{v}^H = \widetilde{\mathbf{Q}}\widetilde{\mathbf{R}} \quad \text{mit } \widetilde{\mathbf{Q}} = \mathbf{Q}\mathbf{Q}_1^H \mathbf{G}_{n-1,n}^H \cdot \cdots \cdot \mathbf{G}_{12}^H \,.$$

MATLAB-function: `[Q1,R1] = qrupdate(Q,R,u,v);`

Special case: rank-1-modifications preserving symmetry & *positivity* ($\rightarrow$ Def. 2.7.9):

$$\mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n} \quad \mapsto \quad \widetilde{\mathbf{A}} := \mathbf{A} + \alpha \mathbf{v}\mathbf{v}^H \,, \quad \mathbf{v} \in \mathbb{K}^n \,, \alpha > 0 \,. \qquad (2.9.12)$$

Task: Efficient computation of Cholesky factorization $\widetilde{\mathbf{A}} = \widetilde{\mathbf{R}}^H \widetilde{\mathbf{R}}$ ($\rightarrow$ Lemma 2.7.14) of $\widetilde{\mathbf{A}}$ from (2.9.12), when Cholesky factorization $\mathbf{A} = \mathbf{R}^H \mathbf{R}$ of $\mathbf{A}$ already known

① With $\mathbf{w} := \mathbf{R}^{-H}\mathbf{v}$: $\quad \mathbf{A} + \alpha\mathbf{v}\mathbf{v}^H = \mathbf{R}^H(\mathbf{I} + \alpha\mathbf{w}\mathbf{w}^H)\mathbf{R}$.

➡ Asymptotic complexity $O(n^2)$ (backward substitution !)

② Idea: formal Gaussian elimination: with $\widetilde{\mathbf{w}} = (w_2, \dots, w_n)^T$ $\rightarrow$ see (2.1.11)

$$\mathbf{I} + \alpha\mathbf{w}\mathbf{w}^H = \left(\begin{array}{c|c} 1+\alpha w_1^2 & \alpha w_1 \widetilde{\mathbf{w}}^H \\ \hline \alpha w_1 \widetilde{\mathbf{w}} & \mathbf{I} + \alpha\widetilde{\mathbf{w}}\widetilde{\mathbf{w}}^H \end{array}\right) \rightarrow \left(\begin{array}{c|c} 1+\alpha w_1^2 & \alpha w_1 \widetilde{\mathbf{w}}^H \\ \hline 0 & \mathbf{I} + \alpha^{(1)}\widetilde{\mathbf{w}}\widetilde{\mathbf{w}}^H \end{array}\right) \tag{2.9.13}$$

where $\alpha^{(1)} := \alpha - \dfrac{\alpha^2 w_1^2}{1+\alpha w_1^2}$.

same structure ➢ recursion

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 $\mathtt{d}, \mathtt{s}$ computed $\rightarrow$ (2.9.16))

Code 2.9.15: Cholesky factorization of rank-1-modified identity matrix

```
function [d,s] = roid(alpha,w)
n = length(w);
d = []; s = [];
for i=1:n
  t = alpha*w(i);
  d = [d; sqrt(1+t*w(i))];
  s = [s; t/d(i)];
  alpha = alpha - s(i)^2;
end
```

③ Special structure of $\mathbf{R}_1$:

$$\mathbf{R}_1 = \begin{pmatrix} d_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & d_n \end{pmatrix} + \begin{pmatrix} s_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & s_n \end{pmatrix} \begin{pmatrix} 0 & w_2 & w_3 & \cdots & \cdots & w_n \\ 0 & 0 & w_3 & \cdots & \cdots & w_n \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & 0 & w_{n-1} & w_n \\ 0 & \cdots & & \cdots & 0 & w_n \\ 0 & \cdots & & & \cdots & 0 \end{pmatrix} \tag{2.9.16}$$

R. Hiptmair

rev 38355, September 7, 2011

$$\mathbf{R}_1 = \begin{pmatrix} d_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & d_n \end{pmatrix} + \begin{pmatrix} s_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & s_n \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & \cdots & \cdots & 1 \\ 0 & 0 & 1 & \cdots & \cdots & 1 \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & 0 & 1 & 1 \\ 0 & \cdots & & \cdots & 0 & 1 \\ 0 & \cdots & & & \cdots & 0 \end{pmatrix} \begin{pmatrix} w_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & w_n \end{pmatrix}$$

- Smart multiplication

$$\widetilde{\mathbf{R}} := \mathbf{R}_1 \mathbf{R}$$

➜ Complexity $O(n^2)$

$$\mathbf{A} + \alpha \mathbf{v}\mathbf{v}^H = \widetilde{\mathbf{R}}^H \widetilde{\mathbf{R}} .$$

Code 2.9.18: Update of Cholesky factorization in the case of s.p.d. preserving rank-1-modification

```
function Rt = roudchol(R,alpha,v)
w = R'\v;
[d,s] = roid(alpha,w);
T = zeros(1,n);
for j=n-1:-1:1
  T = [w(j+1)*R(j+1,:)+T(1,:);T];
end
Rt = spdiags(d,0,n,n)*R+spdiags(s,0,n,n)*T;
```

MATLAB-function: `R = cholupdate(R,v);`

2.9.0.2 Adding a column

$$\mathbf{A} \in \mathbb{K}^{m,n} \quad \mapsto \quad \widetilde{\mathbf{A}} = \left[(\mathbf{A}_{:,1}, \dots, (\mathbf{A})_{:,k-1}, \mathbf{v}, (\mathbf{A})_{:,k}, \dots, (\mathbf{A})_{:,n}\right] \;, \quad \mathbf{v} \in \mathbb{K}^m \;. \tag{2.9.19}$$

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 $\widetilde{\mathbf{A}} = \widetilde{\mathbf{Q}}\widetilde{\mathbf{R}}$ of $\widetilde{\mathbf{A}}$ from (2.9.19) , $\widetilde{\mathbf{Q}} \in \mathbb{K}^{m,m}$ unitary, $\widetilde{\mathbf{R}} \in \mathbb{K}^{m,n+1}$ upper triangular

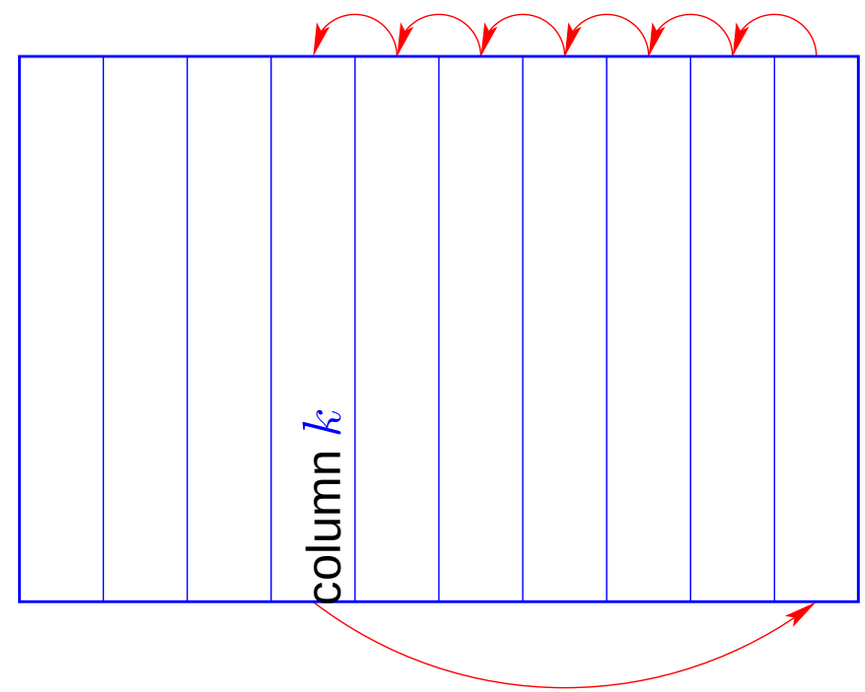
Idea: Easy, if $k = n+1$ (adding last column)

▶ $\exists$ column permutation

$$k \mapsto n+1 \;,\; i \mapsto i-1 \;,\; i = k+1, \dots, n+1$$

$\sim$ permutation matrix

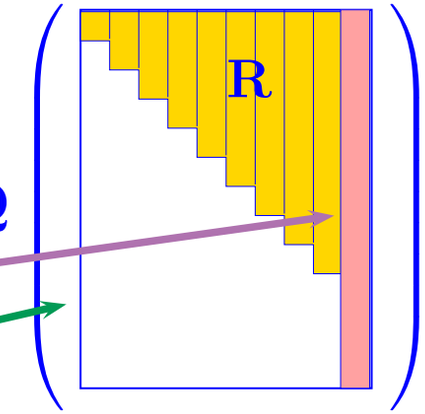
$$\mathbf{P} = \begin{pmatrix} 1 & 0 & \cdots & & & \cdots & 0 \\ 0 & \ddots & & & & & \\ & & 1 & 0 & & & \\ \vdots & & & 0 & 1 & & \\ \vdots & & & & & \ddots & \\ & & & & & & 1 \\ 0 & \cdots & & 1 & & 0 & \end{pmatrix} \in \mathbb{R}^{n+1,n+1}$$



R. Hiptmair

rev 38355, September 7, 2011

$$\widetilde{\mathbf{A}} \quad \longrightarrow \quad \mathbf{A}_1 = \widetilde{\mathbf{A}}\mathbf{P} = [\mathbf{a}_{\cdot 1}, \dots, \mathbf{a}_{\cdot n}, \mathbf{v}] = \mathbf{Q}\,[\mathbf{R} \quad \mathbf{Q}^H\mathbf{v}] = \mathbf{Q}$$



column $\mathbf{Q}^H\mathbf{v}$

case $m > n+1$

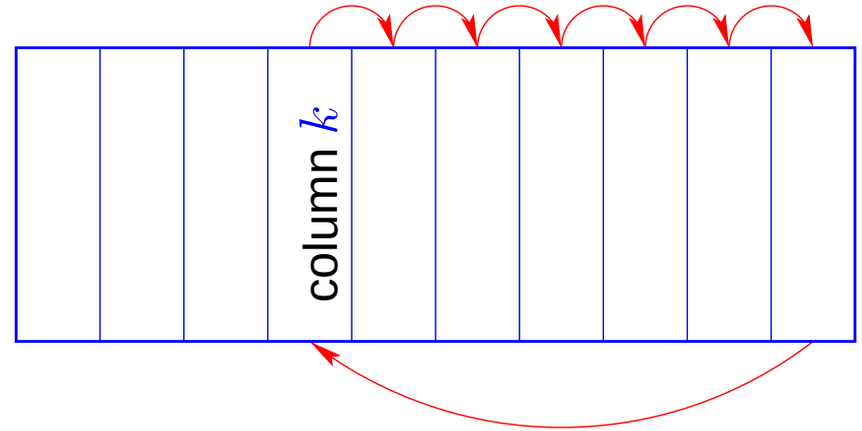
① If $m > n+1$: $\exists$ orthogonal transformation $\mathbf{Q}_1 \in \mathbb{K}^{m,m}$ (Householder reflection) with

$$\mathbf{Q}_1\mathbf{Q}^H\mathbf{v} = \begin{pmatrix} * \\ \vdots \\ * \\ * \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}*\\ \vdots\\ *\end{matrix}}\right\} n+1 \\ \left.\vphantom{\begin{matrix}0\\ \vdots\\ 0\end{matrix}}\right\} m-n-1 \end{matrix} \quad \blacktriangleright \quad \mathbf{Q}_1\mathbf{Q}^H\mathbf{A}_1 = \begin{pmatrix} * & \cdots & & \cdots & * \\ 0 & * & & & * \\ \vdots & & \ddots & & \vdots \\ & & & * & * \\ \vdots & & & & * \\ 0 & \cdots & & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & & \cdots & 0 \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}*\\ \vdots\\ *\end{matrix}}\right\} n+1 \\ \left.\vphantom{\begin{matrix}0\\ \vdots\\ 0\end{matrix}}\right\} m-n-1 \end{matrix}$$

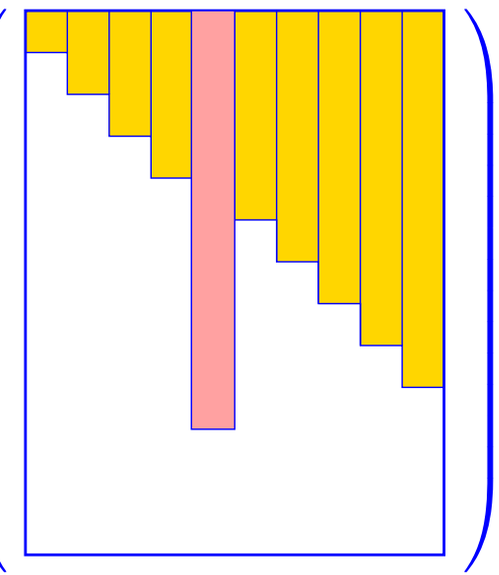
➡ Computational effort $O(m-n)$ (a single reflection)

inverse permutation:

$\sim$ right multiplication with $\mathbf{P}^H$

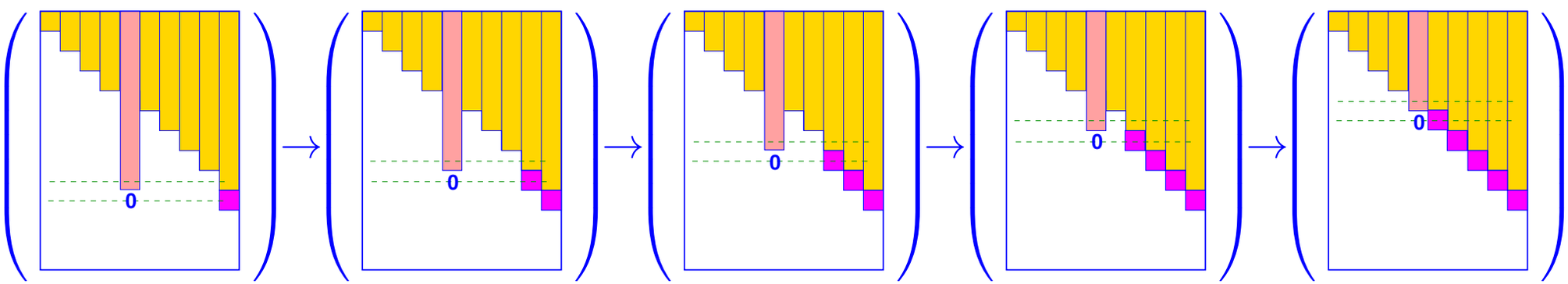


$$\mathbf{Q}_1\mathbf{Q}^H\widetilde{\mathbf{A}} = \mathbf{Q}_1\mathbf{Q}^H\mathbf{A}_1\mathbf{P}^H = \begin{pmatrix} * & \cdots & & * & & \cdots & * \\ 0 & * & & * & & & \vdots \\ \vdots & & \ddots & \vdots & & & \\ & & & \vdots & \ddots & & \vdots \\ & & & * & & * & * \\ \vdots & & & * & & & * \\ 0 & \cdots & & 0 & & \cdots & 0 \\ \vdots & & & \vdots & & & \vdots \\ 0 & \cdots & & 0 & & \cdots & 0 \end{pmatrix} =$$



② $n+1-k$ successive Givens rotations $\Rightarrow$ upper triangular matrix $\widetilde{\mathbf{R}}$

$$\mathbf{Q}_1\mathbf{Q}^H\mathbf{A}_1 = \begin{pmatrix} * & \cdots & & * & & \cdots & * \\ 0 & * & & * & & & \vdots \\ \vdots & & \ddots & \vdots & & & \\ & & & \vdots & \ddots & & \vdots \\ & & & * & & * & * \\ & & & * & & & * \\ \vdots & & & * & & & 0 \\ 0 & \cdots & & 0 & & \cdots & \vdots \\ \vdots & & & \vdots & & & \vdots \\ 0 & \cdots & & 0 & & \cdots & 0 \end{pmatrix} \xrightarrow{\mathbf{G}_{n,n+1}} \cdots \xrightarrow{\mathbf{G}_{k,k+1}} \begin{pmatrix} * & \cdots & & * & & \cdots & * \\ 0 & * & & * & & & \vdots \\ \vdots & & \ddots & \vdots & & & \\ & & & * & & & \\ & & & 0 & \ddots & & \\ & & & 0 & & * & \vdots \\ \vdots & & & 0 & & & * \\ 0 & \cdots & & 0 & & \cdots & 0 \\ \vdots & & & \vdots & & & \\ 0 & \cdots & & 0 & & \cdots & 0 \end{pmatrix}$$



- - - - $\hat{=}$ rows targeted by plane rotations, ■ $\hat{=}$ new entries $\neq 0$

➡ Asymptotic complexity $O\left((n-k)^2\right)$

MATLAB-function: `[Q1,R1] = qrinsert(Q,R,j,x)`

2.9.0.3 Adding a row

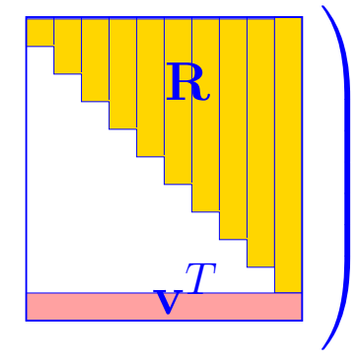
$$\mathbf{A} \in \mathbb{K}^{m,n} \quad \mapsto \quad \widetilde{\mathbf{A}} = \begin{bmatrix} (\mathbf{A})_{1,:} \\ \vdots \\ (\mathbf{A})_{k-1,:} \\ \mathbf{v}^T \\ (\mathbf{A})_{k,:} \\ \vdots \\ (\mathbf{A})_{m,:} \end{bmatrix} , \quad \mathbf{v} \in \mathbb{K}^n . \tag{2.9.20}$$

Given: QR-factorization $\mathbf{A} = \mathbf{QR}$, $\mathbf{Q} \in \mathbb{K}^{m+1,m+1}$ unitary, $\mathbf{R} \in \mathbb{K}^{m,n}$ upper triangular matrix.

Task: efficient computation of QR-factorization $\widetilde{\mathbf{A}} = \widetilde{\mathbf{Q}}\widetilde{\mathbf{R}}$ of $\widetilde{\mathbf{A}}$ from (2.9.20) , $\widetilde{\mathbf{Q}} \in \mathbb{K}^{m+1,m+1}$ unitary, $\widetilde{\mathbf{R}} \in \mathbb{K}^{m+1,n+1}$ 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) $\mathbf{P} \in \{0,1\}^{m+1,m+1}$

$$\mathbf{P}\widetilde{\mathbf{A}} = \begin{pmatrix} \mathbf{A} \\ \mathbf{v}^T \end{pmatrix} \quad \blacktriangleright \quad \begin{pmatrix} \mathbf{Q}^H & 0 \\ 0 & 1 \end{pmatrix} \mathbf{P}\widetilde{\mathbf{A}} = \begin{pmatrix} \mathbf{R} \\ \mathbf{v}^T \end{pmatrix} \quad =$$



case $m = n$

② Transform into upper triangular form by m successive Givens rotations:

$$\begin{pmatrix} * & \cdots & & & \cdots & * \\ 0 & * & & & & \vdots \\ \vdots & 0 & \ddots & & & \\ \vdots & \vdots & & * & & \vdots \\ 0 & 0 & & 0 & * & * \\ 0 & 0 & & 0 & 0 & * \\ * & \cdots & \cdots & * & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{1,m+1}} \begin{pmatrix} * & \cdots & & & \cdots & * \\ 0 & * & & & & \vdots \\ \vdots & 0 & \ddots & & & \\ \vdots & \vdots & & * & & \vdots \\ 0 & 0 & & 0 & * & * \\ 0 & 0 & & 0 & 0 & * \\ 0 & * & \cdots & * & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{2,m+1}} \cdots$$

R. Hiptmair

rev 38355, September 7, 2011

$$\cdots \xrightarrow{\mathbf{G}_{m-1,m+1}} \begin{pmatrix} * & \cdots & & & \cdots & * \\ 0 & * & & & & \vdots \\ \vdots & 0 & \ddots & & & \\ \vdots & \vdots & & * & & \vdots \\ 0 & 0 & & 0 & * & * \\ 0 & 0 & & 0 & 0 & * \\ 0 & \cdots & & \cdots & 0 & * \end{pmatrix} \xrightarrow{\mathbf{G}_{m,m+1}} \begin{pmatrix} * & \cdots & & & \cdots & * \\ 0 & * & & & & \vdots \\ \vdots & 0 & \ddots & & & \\ \vdots & \vdots & & * & & \vdots \\ 0 & 0 & & 0 & * & * \\ 0 & 0 & & 0 & 0 & * \\ 0 & \cdots & & \cdots & 0 & 0 \end{pmatrix} := \widetilde{\mathbf{R}} \quad (2.9.21)$$

③ With $\mathbf{Q}_1 = \mathbf{G}_{m,m+1} \cdot \cdots \cdot \mathbf{G}_{1,m+1}$

$$\widetilde{\mathbf{A}} = \mathbf{P}^T \begin{pmatrix} \mathbf{Q} & 0 \\ 0 & 1 \end{pmatrix} \mathbf{Q}_1^H \widetilde{\mathbf{R}} = \widetilde{\mathbf{Q}}\widetilde{\mathbf{R}} \quad \text{with unitary} \quad \widetilde{\mathbf{Q}} \in \mathbb{K}^{m+1,m+1} .$$

3 Data Interpolation in 1D

3.1 Abstract interpolation

Example 3.1.2 (Constitutive relations (*ger.* Kennlinien) from measurements).

Known: several *accurate* measurements

$$(t_i, y_i)\ , \quad i = 1, \dots, m$$

Examples:

t	y
voltage U	current I
pressure p	density ρ
magnetic field H	magnetic flux B
...	...

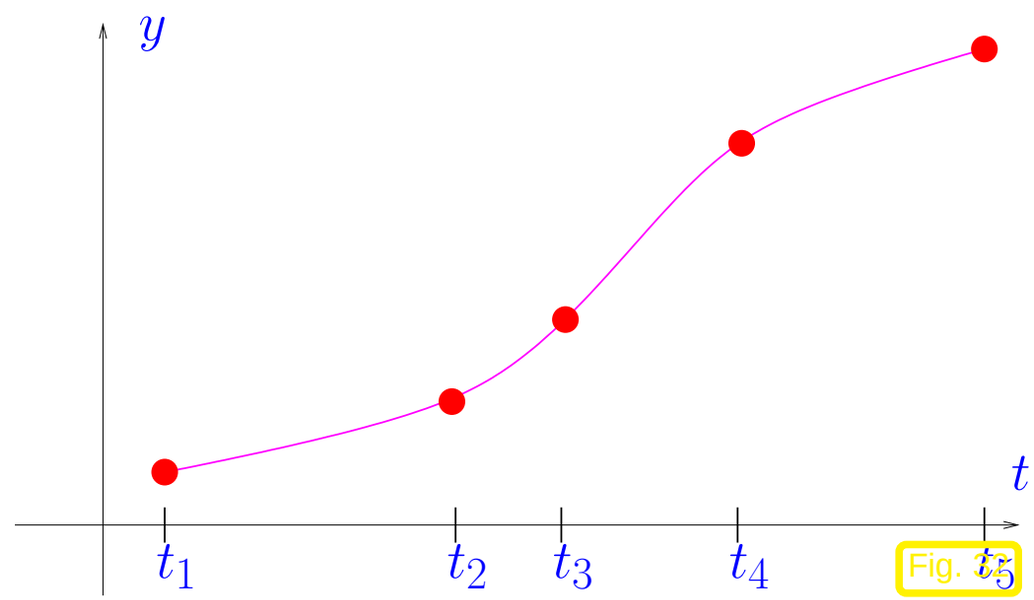


Fig. 32

R. Hiptmair

rev 38355, September 7, 2011

Remark 3.1.3 (Function representation).

```
class Function {
  private:
    // various internal data describing f
  public:
    // Constructor:  acccepts information necessary for specifying the function
    Function(/* .... */);
    // Evaluation operator
    double operator () (double t) const;
};
```

△

Example 3.1.5 (Piecewise linear interpolation). See also Sect. 3.6.1

Piecewise linear interpolant of data ▷

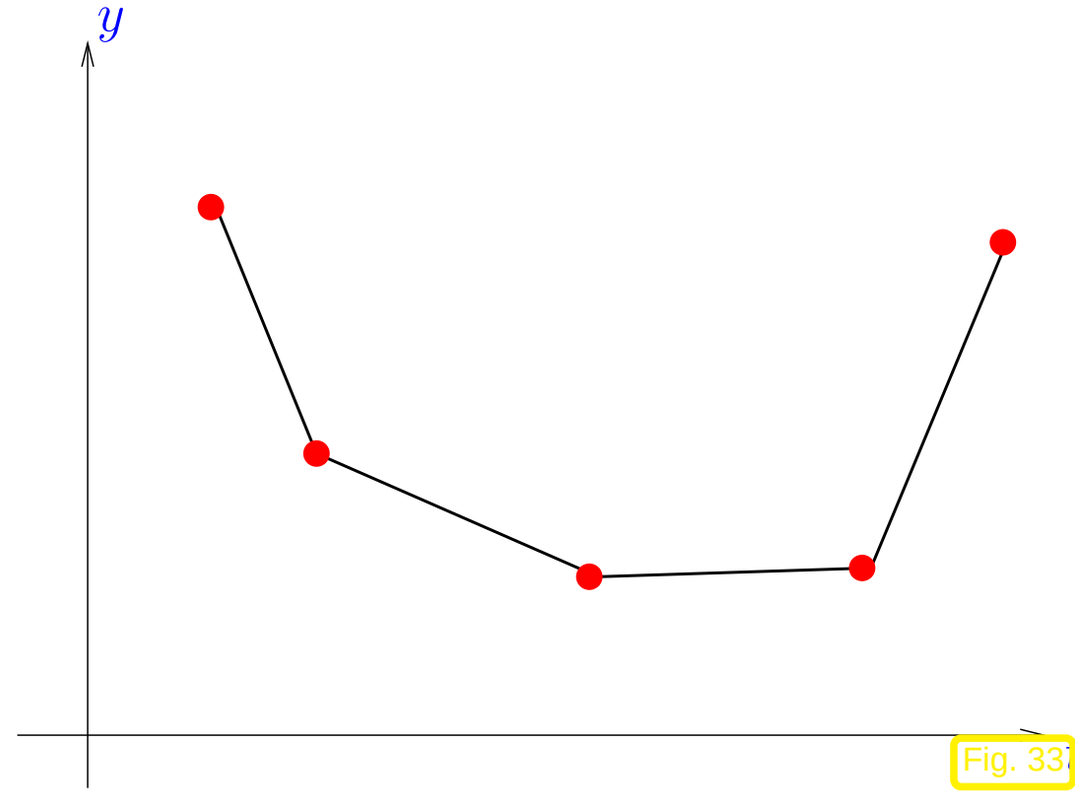


Fig. 33

“Tent function” (“hat function”) basis:

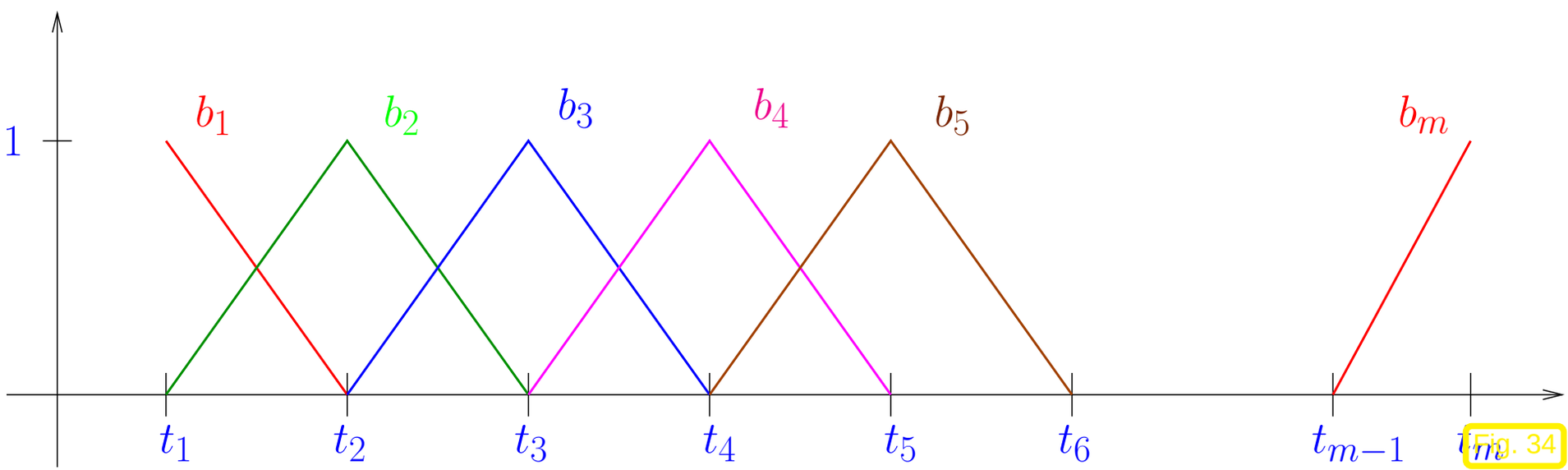


Fig. 34

R. Hiptmair

rev 38355, September 7, 2011

Remark 3.1.8 (Interpolation as linear mapping).

Remark 3.1.11 ("Software solution" of interpolation problem).

```
class Interpolant {
  private:
  // various internal data describing f
  public:
    // Constructor:  computation of coefficients c_j of representation (3.1.4)
    Interpolant(const vector<double> &t,const vector<double> &y);
    // Evaluation operator for interpolant f
    double operator () (double t) const;
};
```

3.2 Polynomials

Notation: Vector space of the polynomials of degree $\leq k$, $k \in \mathbb{N}$:

$$\mathcal{P}_k := \{t \mapsto \alpha_k t^k + \alpha_{k-1} t^{k-1} + \cdots + \alpha_1 t + \alpha_0 \,,\, \alpha_j \in \mathbb{K}\} \,. \tag{3.2.1}$$

leading coefficient

Theorem 3.2.2 (Dimension of space of polynomials)**.**

$$\dim \mathcal{P}_k = k+1 \quad \textit{and} \quad \mathcal{P}_k \subset C^\infty(\mathbb{R}).$$

Remark 3.2.3 (Polynomials in Matlab)*.*

MATLAB: $\alpha_k t^k + \alpha_{k-1} t^{k-1} + \cdots + \alpha_0$ ➔ Vector $(\alpha_k, \alpha_{k-1}, \ldots, \alpha_0)$ (ordered!). △

Remark 3.2.4 (Horner scheme)*.* $\rightarrow$ [13, Bem. 8.11]

Evaluation of a polynomial in monomial representation: Horner scheme

$$p(t) = t(\cdots t(t(\alpha_n t + \alpha_{n-1}) + \alpha_{n-2}) + \cdots + \alpha_1) + \alpha_0 \,. \tag{3.2.5}$$

Code 3.2.6: Horner scheme (polynomial in MATLAB format, see Rem. 3.2.3)

```
function y = polyval(p,x)
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

Interpolation problem (→ 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

$$p(t_j) = y_j \quad \text{for} \quad j = 0, \dots, n \,. \tag{3.3.1}$$

R. Hiptmair

rev 38355, October 26, 2011

For nodes $t_0 < t_1 < \cdots < t_n$ ($\rightarrow$ Lagrange interpolation) consider

$$\text{Lagrange polynomials} \quad L_i(t) := \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t-t_j}{t_i-t_j}\,, \quad i=0,\dots,n\,. \tag{3.3.2}$$

➔ $$L_i \in \mathcal{P}_n \quad \text{and} \quad \boxed{L_i(t_j) = \delta_{ij}}$$

Example 3.3.3. Lagrange polynomials for uniformly spaced nodes

$$\mathcal{T} := \left\{ t_j = -1 + \frac{2}{n}\,j \right\}\,, \quad j=0,\dots,n\,.$$

Plot $n=10,\ j=0,2,5$ ➔

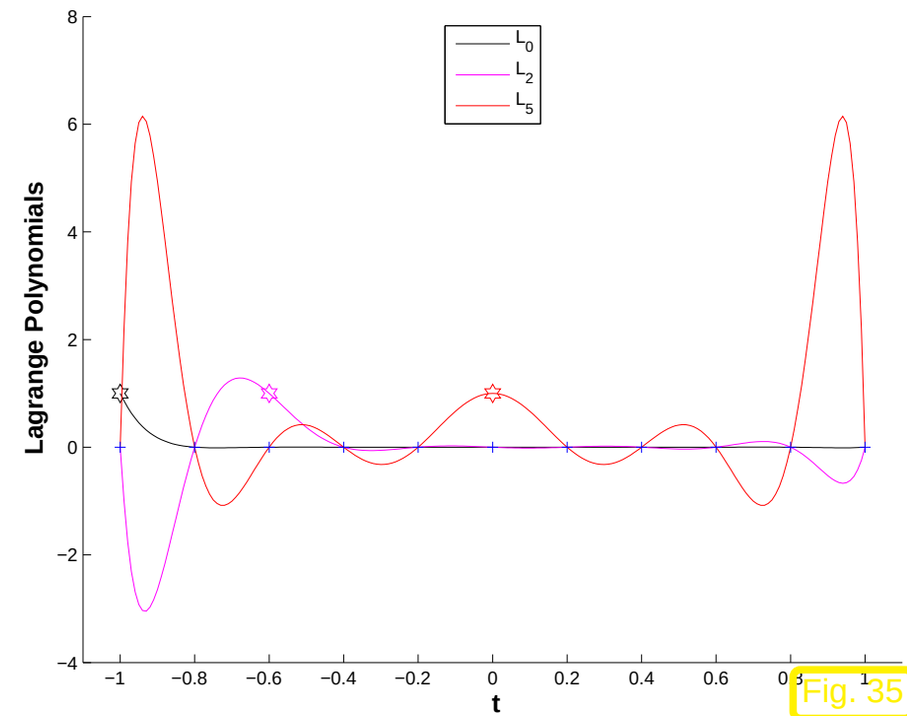


Fig. 35

◇

R. Hiptmair

rev 38355, October 26, 2011

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* $p \in \mathcal{P}_n$.

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*

$$\mathsf{I}_{\mathcal{T}} : \begin{cases} \mathbb{K}^{n+1} & \rightarrow \mathcal{P}_n \ , \\ (y_0, \dots, y_n)^T & \mapsto \textit{interpolating polynomial } p \ . \end{cases} \tag{3.3.8}$$

R. Hiptmair

rev 38355, October 26, 2011

3.4 Polynomial Interpolation: Algorithms

3.4.1 Multiple evaluations

```
class PolyInterp {
  private:
  // various internal data describing p
  public:
    // Constructor taking node vector (t_0,...,t_n) as argument
    PolyInterp(const vector<double> &t);
    // Evaluation operator at x for data (y_0,...,y_n)
    double eval(double x,const vector<double> &y) const;
};
```

Barycentric interpolation formula

$$p(t) = \frac{\displaystyle\sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}\, y_i}{\displaystyle\sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}} \,. \tag{3.4.1}$$

with $\lambda_i = \dfrac{1}{(t_i-t_0)\cdots(t_i-t_{i-1})(t_i-t_{i+1})\cdots(t_i-t_n)}$, $i = 0,\dots,n$ $\rightarrow$ precompute !

Computational effort:
- computation of λ_i: $O(n^2)$ (only once),
- every subsequent evaluation of p: $O(n)$,

$\Rightarrow$ total effort $O(Nn) + O(n^2)$

Code 3.4.2: Evaluation of the interpolation polynomials with barycentric formula

```
function p = intpolyval(t,y,x)
% t:  row vector of nodes t_0,...,t_n
% y:  row vector of data y_0,...,y_n
% x:  row vector of evaluation points x_1,...,x_N
n = length(t); % number of interpolation nodes = degree of polynomial -1
N = length(x); % Number of evaluation points stored in x
% Precompute the weights λ_i with effort O(n^2)
for k = 1:n
  lambda(k) = 1 / prod(t(k) - t([1:k-1,k+1:n])); end;
for i = 1:N
% Compute quotient of weighted sums of λ_i/(t-t_i), effort O(n)
  z = (x(i)-t); j = find(z == 0);
  if (~isempty(j)), p(i) = y(j); % avoid division by zero
  else
    mu = lambda./z; p(i) = sum(mu.*y)/sum(mu);
  end
end
```


R. Hiptmair

rev 38355, October 26, 2011

3.4.2 Single evaluation [13, Sect. 8.2.2]

For $\{i_0,\dots,i_m\} \subset \{0,\dots,n\}$, $0 \le m \le n$:

$p_{i_0,\dots,i_m}$ = interpolation polynomial of degree m through $(t_{i_0}, y_{i_0}),\dots,(t_{i_m}, y_{i_m})$,

recursive definition:

$$\begin{aligned} p_i(t) &\equiv y_i \;, \qquad\qquad i = 0,\dots,n \;, \\ p_{i_0,\dots,i_m}(t) &= \frac{(t-t_{i_0})p_{i_1,\dots,i_m}(t) - (t-t_{i_m})p_{i_0,\dots,i_{m-1}}(t)}{t_{i_m} - t_{i_0}} \\ &= TODO! \;. \end{aligned} \tag{3.4.3}$$

Aitken-Neville algorithm:

	$n =$ 0	1	2	3
t_0	$y_0 =: p_0(x)$	$\to p_{01}(x)$	$\to p_{012}(x)$	$\to p_{0123}(x)$
t_1	$y_1 =: p_1(x)$	$\to p_{12}(x)$	$\to p_{123}(x)$	
t_2	$y_2 =: p_2(x)$	$\to p_{23}(x)$		
t_3	$y_3 =: p_3(x)$			

Code 3.4.4: Aitken-Neville algorithm

```
function v = ANipoleval(t,y,x)
for i=1:length(y)
  for k=i-1:-1:1
    y(k) =
      y(k+1)+(y(k+1)-y(k))*...
           (x-t(i))/(t(i)-t(k));
  end
end
v = y(1);
```

R. Hiptmair

rev 38355, October 26, 2011

```
class PolyEval {
   private:
     // evaluation point and various internal data describing the polynomials
   public:
      // Constructor taking the evaluation point as argument
      PolyEval(double x);
      // Add another data point and update internal information
      void addPoint(t,y);
      // Value of current interpolating polynomial at x
      double eval(void) const;
};
```

Example 3.4.5 (Timing polynomial evaluations).

R. Hiptmair

rev 38355, October 26, 2011

Comparison of the computational time needed for polynomial interpolation of

$$\{t_i = i\}_{i=1,\dots,n}, \qquad \{y_i = \sqrt{i}\}_{i=1,\dots,n},$$

$n = 3,\dots,200$, and evaluation in a point $x \in [0,n]$.

Minimum `tic-toc`-computational time over 100 runs ➔

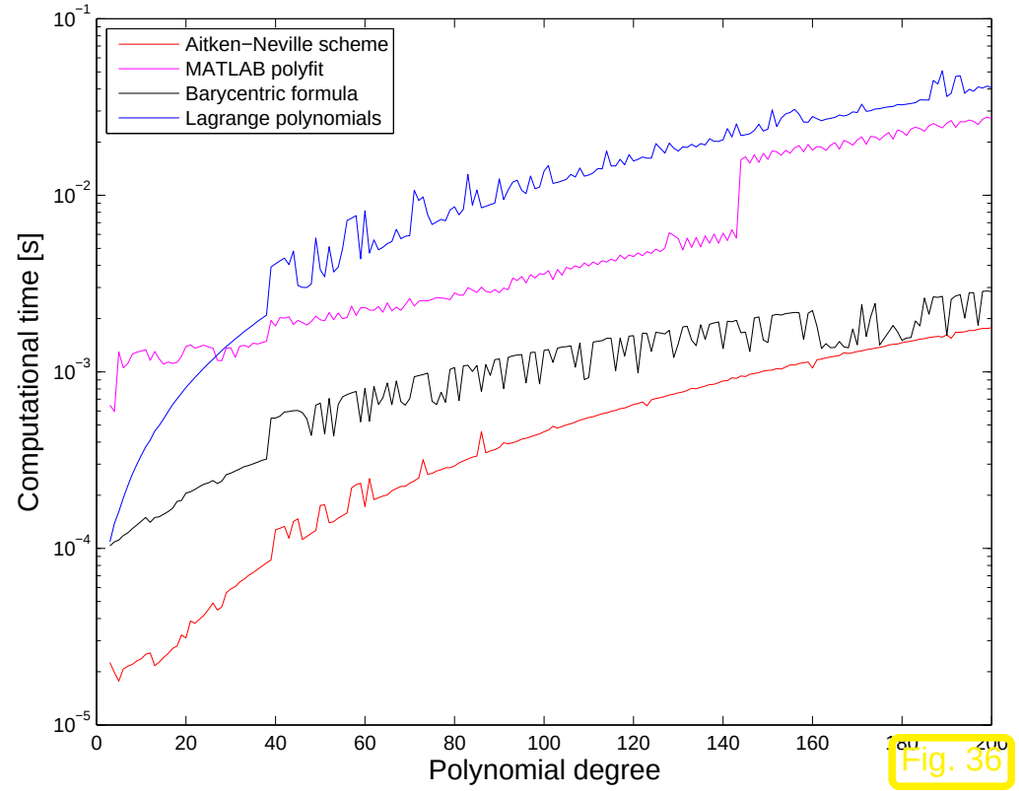


Fig. 36

Code 3.4.6: MATLAB polynomial evaluation using built-in function `polyfit`

```
function v=ipoleval(t,y,x)
  p = polyfit(t,y,length(y)-1);
  v=polyval(p,x);
```

R. Hiptmair

rev 38355, October 26, 2011

Code 3.4.7: Lagrange polynomial interpolation and evaluation

```
function p = intpolyval_lag(t,y,x)
p=zeros(size(x));
for k=1:length(t); p=p + y(k)*lagrangepoly(x, k-1, t); end

function L=lagrangepoly(x, index, nodes)
L=1;
for j=[0:index-1, index+1:length(nodes)-1];
    L = L .* (x-nodes(j+1)) ./ ( nodes(index+1)-nodes(j+1) );
end
```

◇

3.4.3 Extrapolation to zero

Problem: compute $\lim_{t\to 0} f(t)$ with prescribed precision, when the evaluation of the function `y=f(t)` is numerically unstable ($\rightarrow$ Sect. 2.5.2) for $|t| \ll 1$.

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 3.4.8 (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

$$f(x) \approx \frac{f(x+h) - f(x-h)}{h} \,. \tag{3.4.9}$$

▶ 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

Code 3.4.10: Numerical differentiation by extrapolation to zero

```
function d = diffex(f,x,h0,tol)
% f:  handle of to a function defined in a neighborhood of x ∈ ℝ,
% x:  point at which approximate derivative is desired,
% h0:  initial distance from x,
% tol:  relative target tolerance
h = h0;
% Aitken-Neville scheme, see Code 3.4.3 (x = 0!)
y(1) = (f(x+h0)-f(x-h0))/(2*h0);
for i=2:10
  h(i) = h(i-1)/2;
  y(i) = f(x+h(i))-f(x-h(i)))/h(i-1);
  for k=i-1:-1:1
```

```
    y(k) = y(k+1)-(y(k+1)-y(k))*h(i)/(h(i)-h(k));
  end
  % termination of extrapolation, when desired tolerance is achieved
  if (abs(y(2)-y(1)) < tol*abs(y(1))), break; end %
end
d = y(1);
```

`diffex2(@atan,1.1,0.5)`

Degree	Relative error
0	0.04262829970946
1	0.02044767428982
2	0.00051308519253
3	0.00004087236665
4	0.00000048930018
5	0.00000000746031
6	0.00000000001224

`diffex2(@sqrt,1.1,0.5)`

Degree	Relative error
0	0.02849215135713
1	0.01527790811946
2	0.00061205284652
3	0.00004936258481
4	0.00000067201034
5	0.00000001253250
6	0.00000000004816
7	0.00000000000021

`diffex2(@exp,1.1,0.5)`

Degree	Relative error
0	0.04219061098749
1	0.02129207652215
2	0.00011487434095
3	0.00000825582406
4	0.00000000589624
5	0.00000000009546
6	0.00000000000002

◇

3.4.4 Newton basis and divided differences [13, Sect. 8.2.4], [51, Sect. 8.2]

```
class PolyEval {
  private:
  // evaluation point and various internal data describing the polynomials
  public:
    // Idle Constructor
    PolyEval();
    // Add another data point and update internal information
    void addPoint(t,y);
    // Evaluation of current interpolating polynomial at x
    double operator ()  (double x) const;
};
```


Tool: "update friendly" representation: Newton basis for $\mathcal{P}_n$

$$N_0(t) := 1 \,, \quad N_1(t) := (t - t_0) \,, \quad \dots \quad , \quad N_n(t) := \prod_{i=0}^{n-1} (t - t_i) \,. \tag{3.4.13}$$

➢ LSE for polynomial interpolation problem in Newton basis:

$$a_j \in \mathbb{R}: \quad a_0 N_0(t_j) + a_1 N_1(t_j) + \cdots + a_n N_n(t_j) = y_j, \quad j = 0, \dots, n \,.$$

$\Leftrightarrow$ triangular linear system

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & (t_1 - t_0) & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & (t_n - t_0) & \cdots & \prod_{i=0}^{n-1}(t_n - t_i) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} .$$

Simpler and more efficient algorithm using divided differences:

$$\begin{aligned} y[t_i] &= y_i \\ y[t_i, \dots, t_{i+k}] &= \frac{y[t_{i+1}, \dots, t_{i+k}] - y[t_i, \dots, t_{i+k-1}]}{t_{i+k} - t_i} \qquad \text{(recursion)} \end{aligned} \tag{3.4.14}$$

Recursive calculation by divided differences scheme, *cf.* Aitken-Neville scheme, Code 3.4.3:

$$\begin{array}{c|ccccc} t_0 & y[t_0] & & & & \\ & & > & y[t_0,t_1] & & \\ t_1 & y[t_1] & & & > & y[t_0,t_1,t_2] \\ & & > & y[t_1,t_2] & & & > & y[t_0,t_1,t_2,t_3], \\ t_2 & y[t_2] & & & > & y[t_1,t_2,t_3] \\ & & > & y[t_2,t_3] & & \\ t_3 & y[t_3] & & & & \end{array} \tag{3.4.15}$$

R. Hiptmair

rev 38355, October 26, 2011

Code 3.4.16: Divided differences, recursive implementation, in situ computation

```
function y = divdiff(t,y)
n = length(y)-1;
if (n > 0)
  y(1:n) = divdiff(t(1:n),y(1:n));
  for j=0:n-1
    y(n+1) = (y(n+1)-y(j+1))/(t(n+1)-t(j+1));
  end
end
```

Code 3.4.18: Divided differences evaluation by modified Horner scheme

```
function p = evaldivdiff(t,y,x)
dd=divdiff(t,y); % Compute divided differences, see Code 3.4.16
n = length(y)-1;
p=dd(n+1);
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 3.4.19 (Class `PolyEval`).

Code 3.4.20: “Update friendly” polynomial interpolant

```
class PolyEval {
private:
  std::vector<double> t;  // Interpolation nodes
  std::vector<double> y;  // Coefficients in Newton representation
public:
  PolyEval(); // Idle constructor
  void addPoint(double t, double y); // Add another data point
  // evaluate value of current interpolating polynomial at x,
  double operator() (double x) const;
private:
  // Update internal representation, called by addPoint()
  void divdiff();
};

PolyEval::PolyEval() {}

void PolyEval::addPoint(double td, double yd)
{ t.push_back(td); y.push_back(yd); divdiff(); }

// Update coefficients in Newton basis representation, cf.  Code 3.4.16
void PolyEval::divdiff() {
  int n = t.size();
  for(int j=0; j<n-1; j++) y[n-1] =
```


R. Hiptmair

rev 38355, October 26, 2011

```
      ((y[n-1]-y[j])/(t[n-1]-t[j]));
  }

  double PolyEval::operator() (double x) const {
    double s = y.back();
    for(int i = y.size()-2; i>= 0; --i) s = s*(x-t[i])+y[i];
    return s;
  }
```

◇

3.5 Polynomial Interpolation: Sensitivity [51, Sect. 8.1.3]

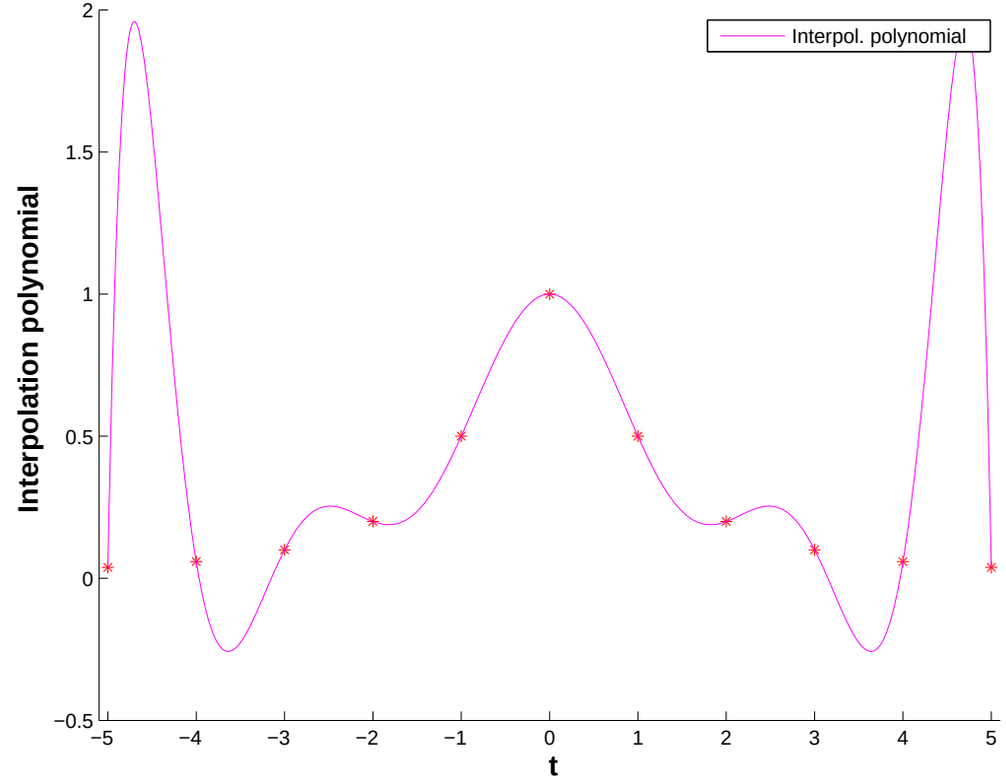
Example 3.5.1 (Oscillating interpolation polynomial: Runge's counterexample). → [13, Sect. 8.3], [51, Ex. 8.1]

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 $n = 10$ ➜



See example 9.1.5.

possible strong oscillations of interpolating polynomials of *high degree* on *uniformly spaced* nodes!

◇

R. Hiptmair

rev 38355, October 26, 2011

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

$$\|\mathsf{I}_{\mathcal{T}}\|_{\infty\to\infty} := \sup_{\mathbf{y}\in\mathbb{K}^{n+1}\setminus\{0\}} \frac{\|\mathsf{I}_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)}}{\|\mathbf{y}\|_\infty} = \left\|\sum_{i=0}^n |L_i|\right\|_{L^\infty(I)} , \tag{3.5.6}$$

$$\|\mathsf{I}_{\mathcal{T}}\|_{2\to 2} := \sup_{\mathbf{y}\in\mathbb{K}^{n+1}\setminus\{0\}} \frac{\|\mathsf{I}_{\mathcal{T}}(\mathbf{y})\|_{L^2(I)}}{\|\mathbf{y}\|_2} \leq \left(\sum_{i=0}^n \|L_i\|^2_{L^2(I)}\right)^{\frac{1}{2}} . \tag{3.5.7}$$

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)*.*

$$I = [-1,1], \quad \mathcal{T} = \{-1 + \tfrac{2k}{n}\}_{k=0}^n \quad \text{(uniformly spaced nodes)}$$

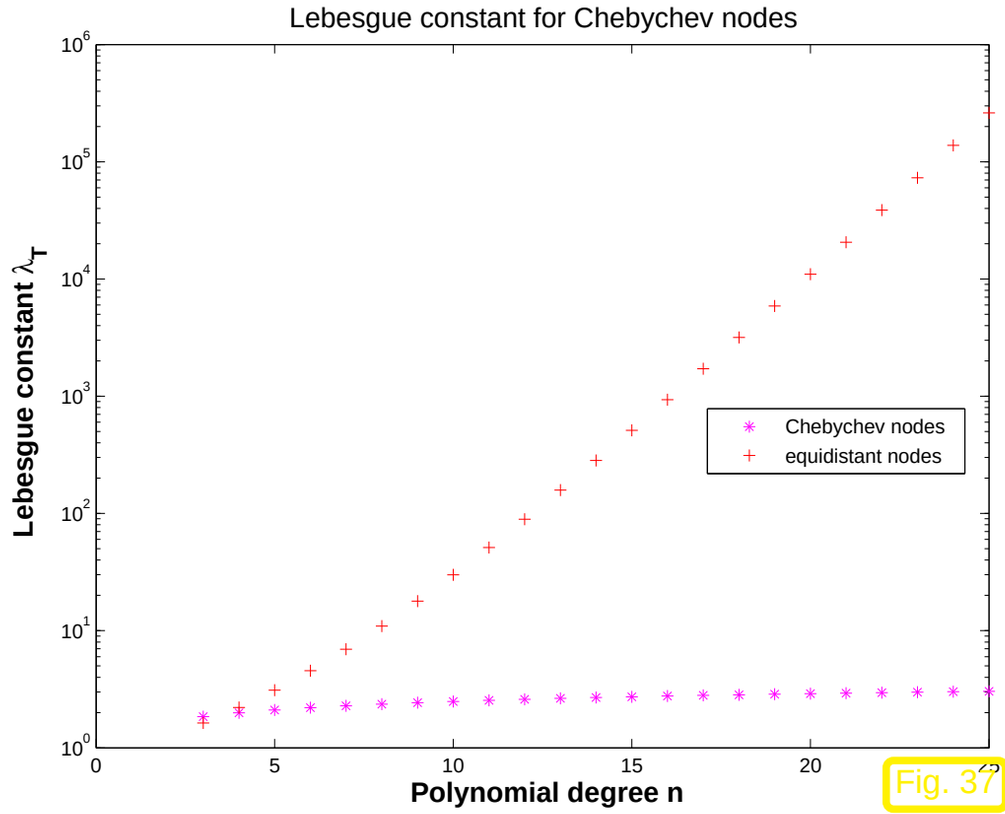


Fig. 37

Theory [11]: for uniformly spaced nodes

$$\boxed{\lambda_{\mathcal{T}} \geq Ce^{n/2}}$$

for $C > 0$ independent of n.

△

R. Hiptmair

rev 38355, October 26, 2011

3.6 Shape preserving interpolation

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

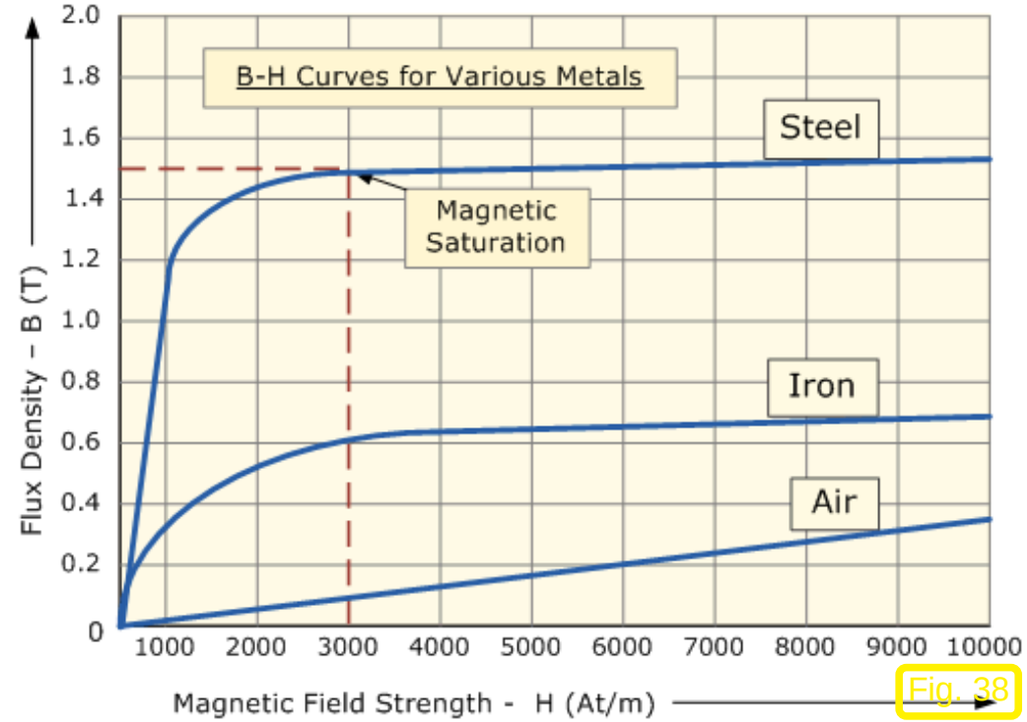


Fig. 38

◇

R. Hiptmair

rev 38355, October 26, 2011

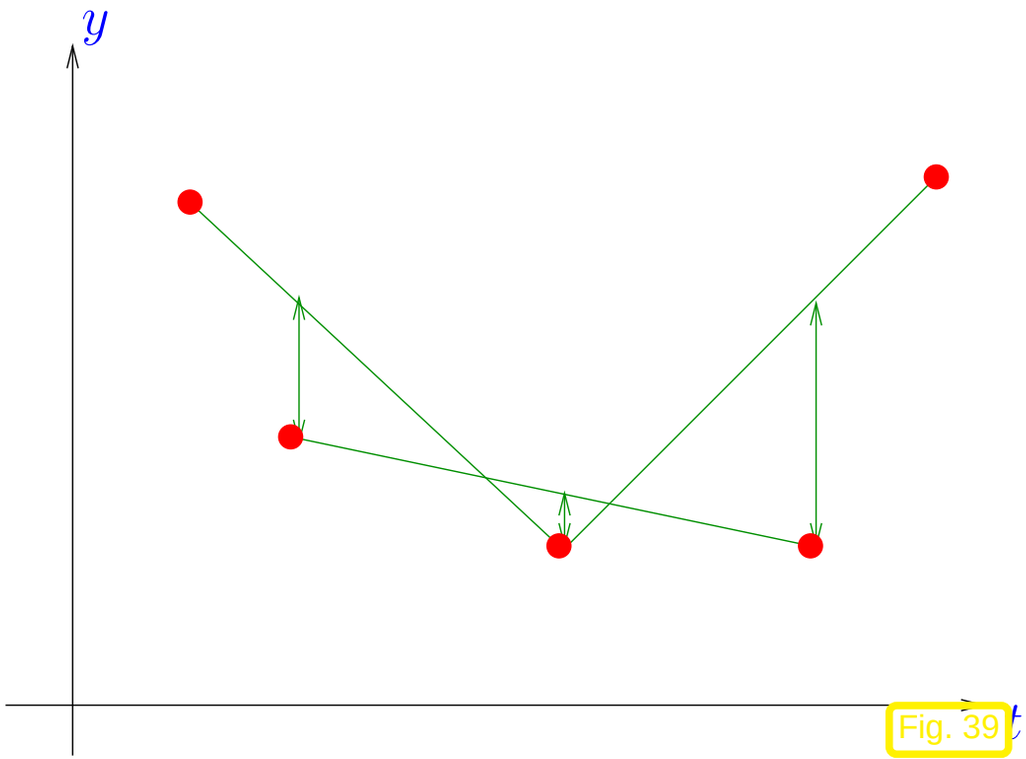
Definition 3.6.2 (monotonic data)**.**

The data (t_i, y_i) are called monotonic when $y_i \geq y_{i-1}$ or $y_i \leq y_{i-1}$, $i = 1, \ldots, n$.

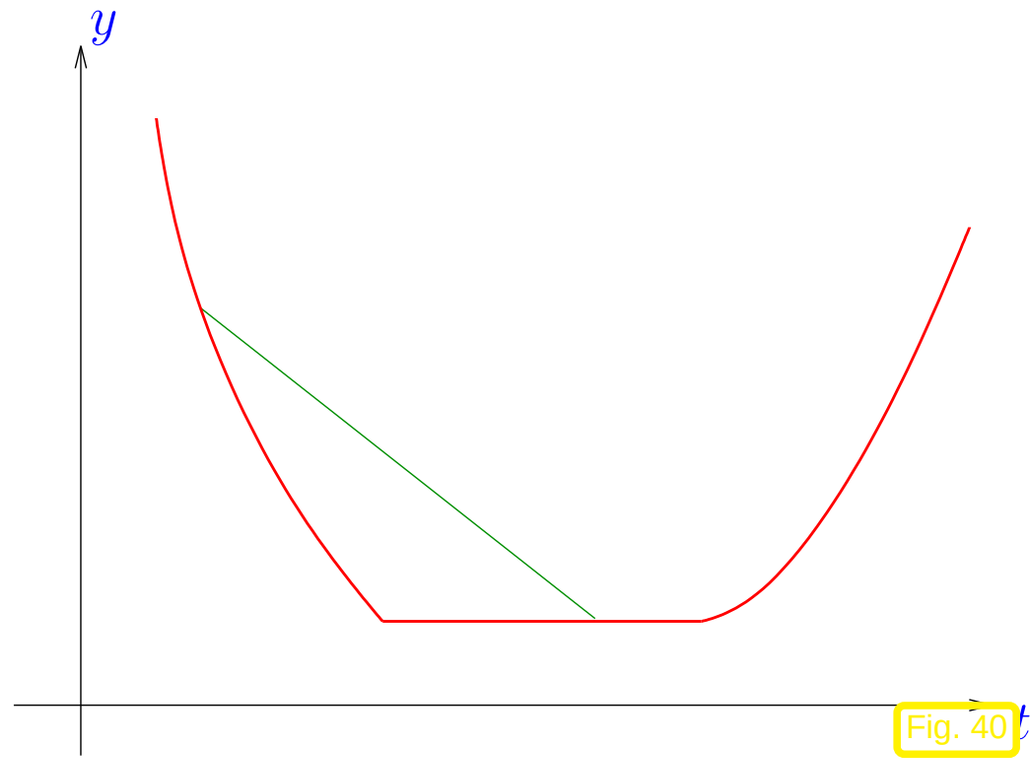
Definition 3.6.3 (Convex/concave data)**.**

The data $\{(t_i, y_i)\}_{i=0}^{n}$ *are called* convex (concave) *if*

$$\Delta_j \overset{(\geq)}{\leq} \Delta_{j+1}\,, \quad j = 1, \dots, n-1 \quad , \quad \Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}}\,, \quad j = 1, \dots, n\,.$$



Convex data



Convex function

Now consider interpolation: data $(t_i, y_i),\ i = 0, \dots, n$ $\longrightarrow$ interpolant f

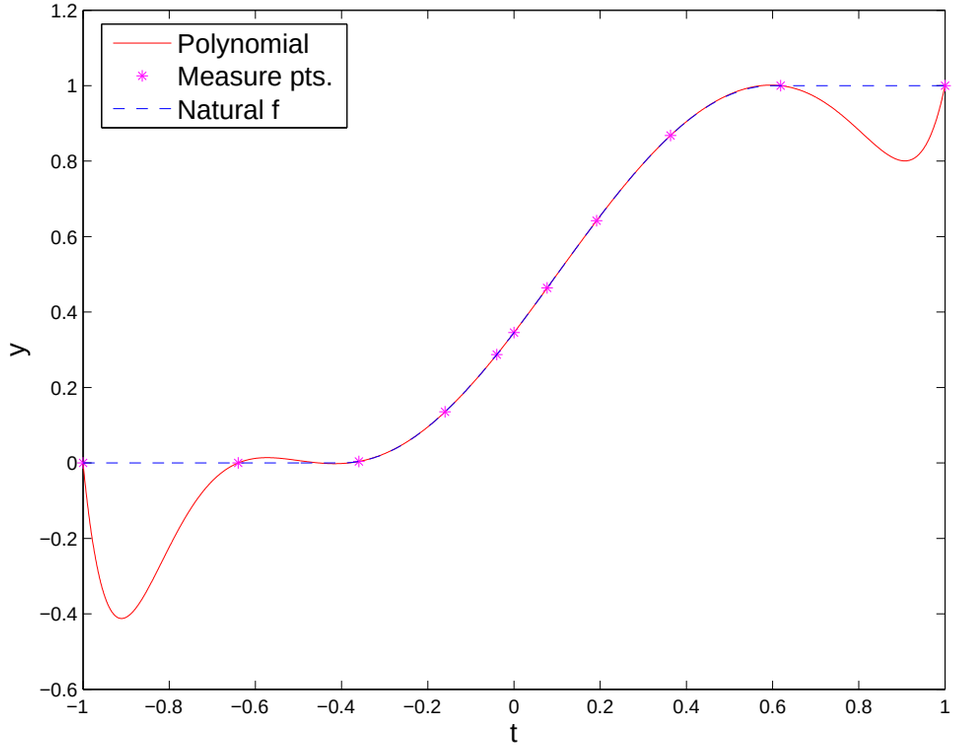
Goal: shape preserving interpolation:

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).



← Interpolating polynomial, degree = 10

◇

R. Hiptmair

rev 38355, October 26, 2011

3.6.1 Piecewise linear interpolation

Data: $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \dots, n$, $n \in \mathbb{N}$, $t_0 < t_1 < \cdots < t_n$.

Piecewise linear interpolant, *cf.* Ex. 3.1.5:

$$s(x) = \frac{(t_{i+1} - t)y_i + (t - t_i)y_{i+1}}{t_{i+1} - t_i} \qquad t \in [t_i, t_{i+1}].$$

Piecewise linear interpolant of data in Fig. 49:

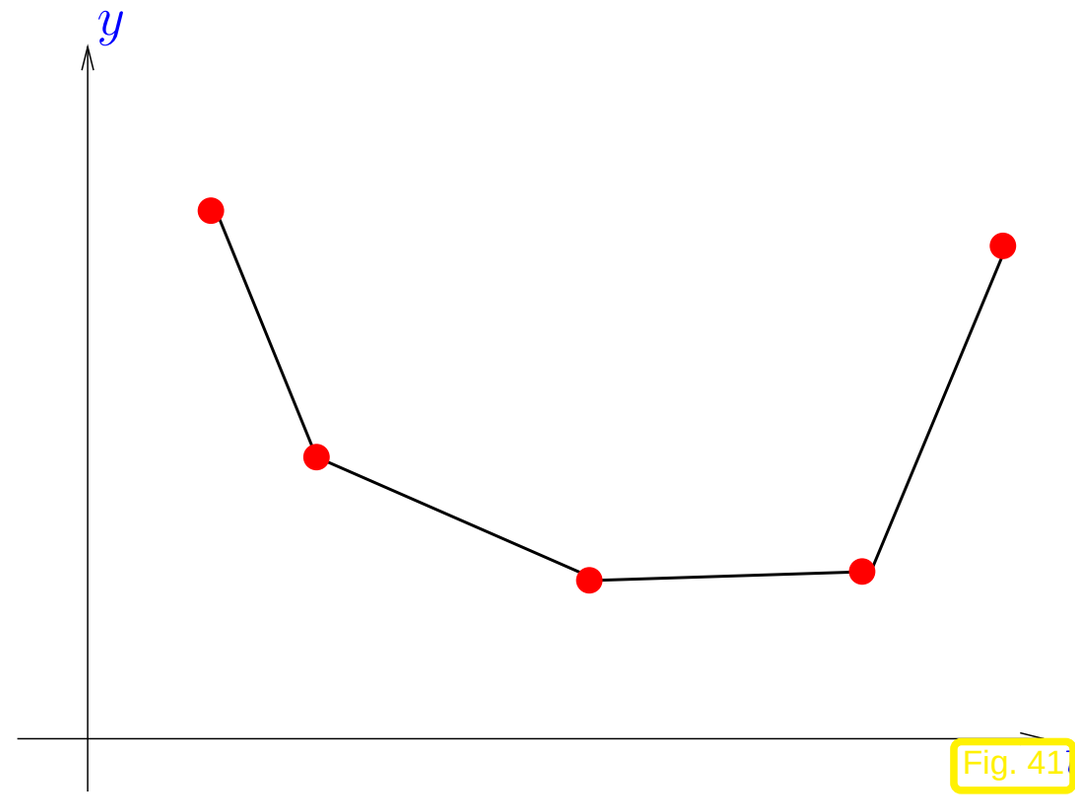


Fig. 41

Obvious: linear interpolation is linear (as mapping $\mathbf{y} \mapsto s$) and local:

$$y_j = \delta_{ij} \;, \quad i,j = 0,\dots,n \quad \Rightarrow \quad \mathrm{supp}(s) \subset [t_{i-1}, t_{i+1}] \;.$$

R. Hiptmair

rev 38355, October 26, 2011

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.*

Example 3.6.7 (Piecewise quadratic interpolation)*.*

Nodes as in Ex. 3.6.5

Piecewise linear/quadratic interpolation ▷

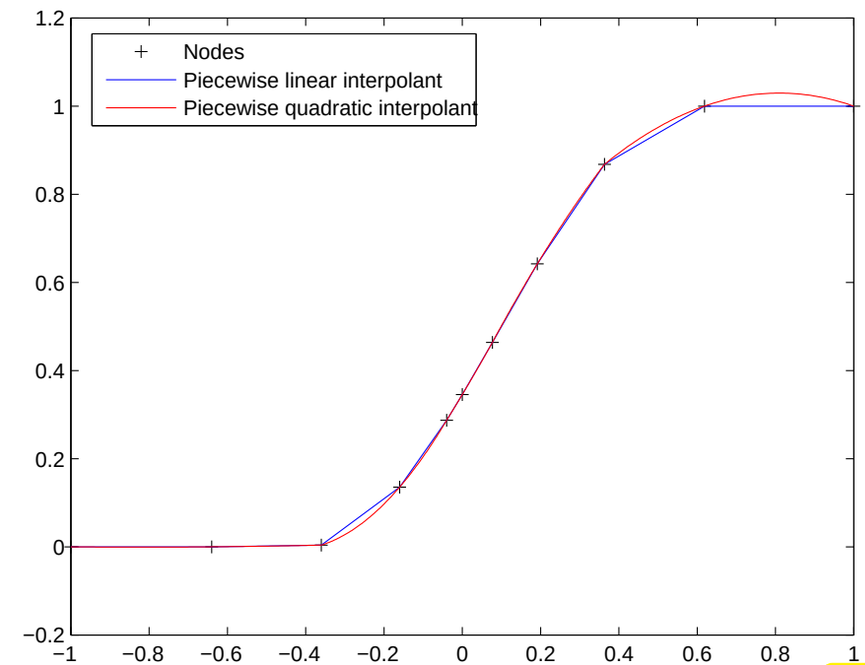


Fig. 42

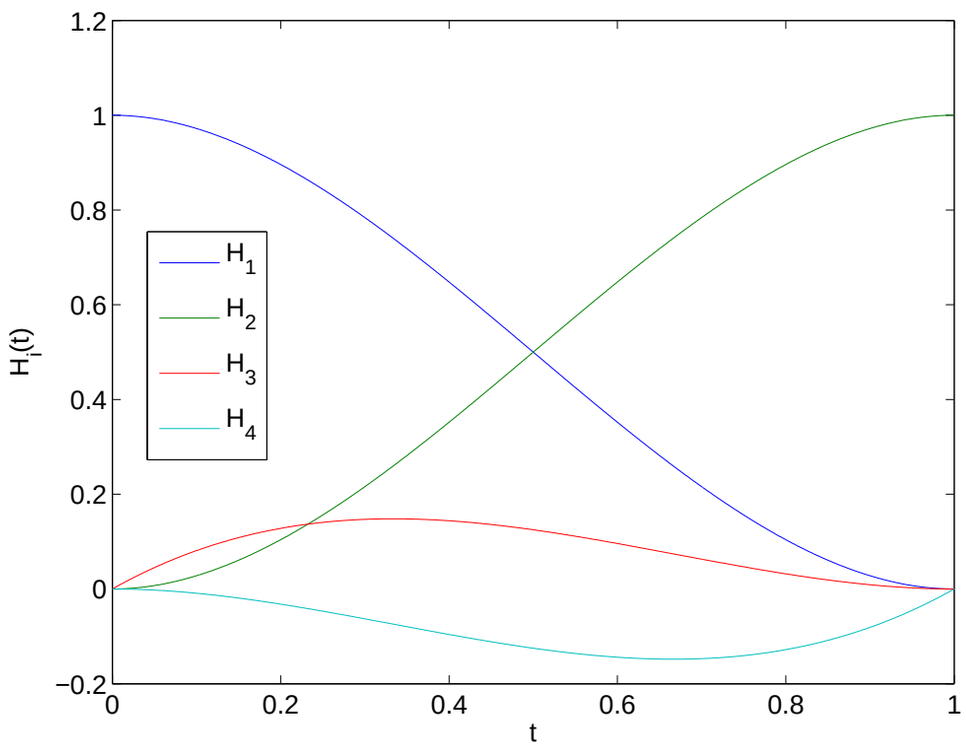
◇

3.7 Cubic Hermite Interpolation

3.7.1 Definition and algorithms

Given: mesh points $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \dots, n$, $\quad t_0 < t_1 < \cdots < t_n$

Goal: function $f \in C^1([t_0, t_n])$, $f(t_i) = y_i$, $i = 0, \dots, n$



$$
\begin{aligned}
H_1(t) &:= \phi(\tfrac{t_i - t}{h_i}) \;, \quad H_2(t) := \phi(\tfrac{t - t_{i-1}}{h_i}) \;, \\
H_3(t) &:= -h_i \psi(\tfrac{t_i - t}{h_i}) \;, \quad H_4(t) := h_i \psi(\tfrac{t - t_{i-1}}{h_i}) \;, \\
h_i &:= t_i - t_{i-1} \quad , \\
\phi(\tau) &:= 3\tau^2 - 2\tau^3 \quad , \\
\psi(\tau) &:= \tau^3 - \tau^2 \;.
\end{aligned}
\tag{3.7.2}
$$

◁ Local basis polynomial on $[0, 1]$

R. Hiptmair

rev 38355, September 26, 2011

	$H(t_{i-1})$	$H(t_i)$	$H'(t_{i-1})$	$H'(t_i)$
H_1	1	0	0	0
H_2	0	1	0	0
H_3	0	0	1	0
H_4	0	0	0	1

How to choose the slopes c_i **?**

Natural attempt: (weighted) average of local slopes:

$$c_i = \begin{cases} \Delta_1 & \text{, for } i = 0 \text{ ,} \\ \Delta_n & \text{, for } i = n \text{ ,} \\ \frac{t_{i+1}-t_i}{t_{i+1}-t_{i-1}}\Delta_i + \frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}}\Delta_{i+1} & \text{, if } 1 \leq i < n \text{ .} \end{cases} \quad , \quad \Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}} \text{ ,} j = 1, \dots, n \text{ .} \tag{3.7.4}$$

► Linear local interpolation operator

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

$$f(x) := \sin(5x)\, e^x \ .$$

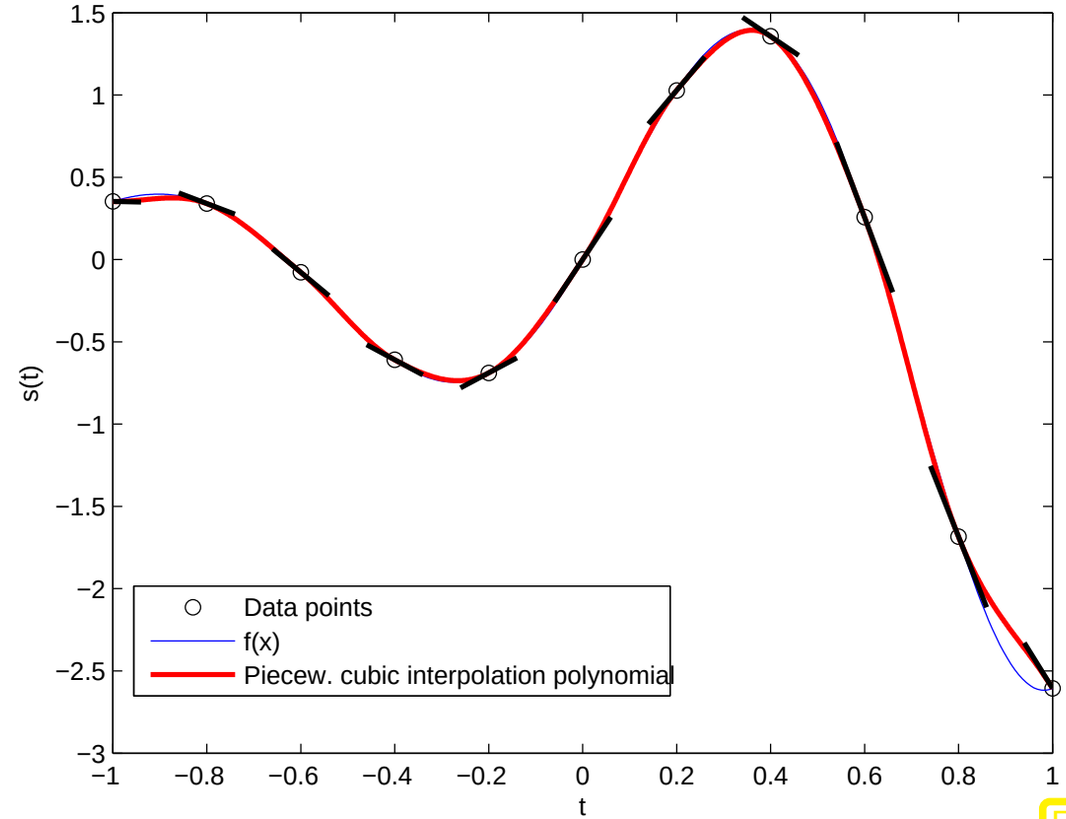


Fig. 43

Use of weighted averages of slopes as in (3.7.4).

See Code 3.7.5.

No preservation of monotonicity!

◇

3.7.2 Shape preserving Hermite interpolation

$$c_i = \begin{cases} 0 & \text{, if } \operatorname{sgn}(\Delta_i) \neq \operatorname{sgn}(\Delta_{i+1}) \text{ ,} \\ \text{weighted average of } \Delta_i, \Delta_{i+1} & \text{otherwise} \end{cases} \quad , \quad i = 1, \dots, n-1 \text{ .} \tag{3.7.7}$$

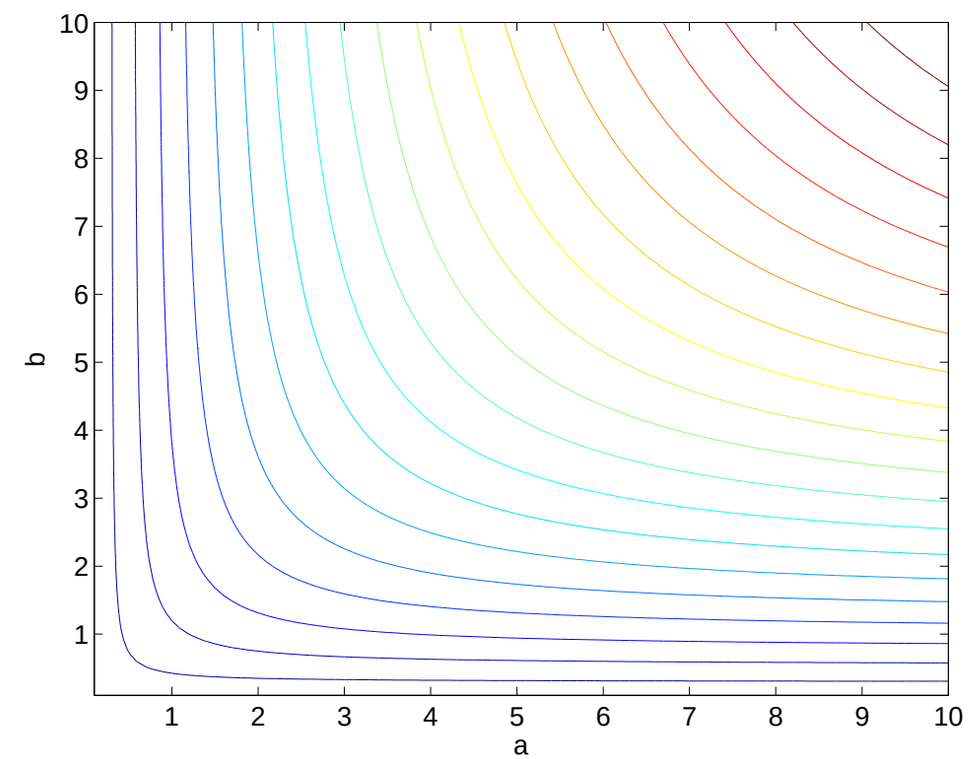
Which kind of average **?**

$$c_i = \frac{1}{\frac{w_a}{\Delta_i} + \frac{w_b}{\Delta_{i+1}}} \tag{3.7.8}$$

= 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 ➔ ($w_a = w_b = 1/2$).



Concrete choice of the weights:

$$w_a = \frac{2h_{i+1} + h_i}{3(h_{i+1} + h_i)}, \qquad w_b = \frac{h_{i+1} + 2h_i}{3(h_{i+1} + h_i)},$$

$$\overset{\mathrm{sgn}(\Delta_1)=\mathrm{sgn}(\Delta_2)}{\longrightarrow} \qquad c_i = \begin{cases} \Delta_1 & \text{, if } i = 0 \text{ ,} \\ \dfrac{3(h_{i+1}+h_i)}{\frac{2h_{i+1}+h_i}{\Delta_i} + \frac{2h_i+h_{i+1}}{\Delta_{i+1}}} & \text{, for } i \in \{1, \dots, n-1\} \text{ ,} \\ \Delta_n & \text{, if } i = n \text{ ,} \end{cases} \qquad h_i := t_i - t_{i-1} \text{ .} \tag{3.7.9}$$

Example 3.7.10 (Monotonicity preserving piecewise cubic polynomial interpolation).

R. Hiptmair

rev 38355, September 26, 2011

Data from ex. 3.6.5

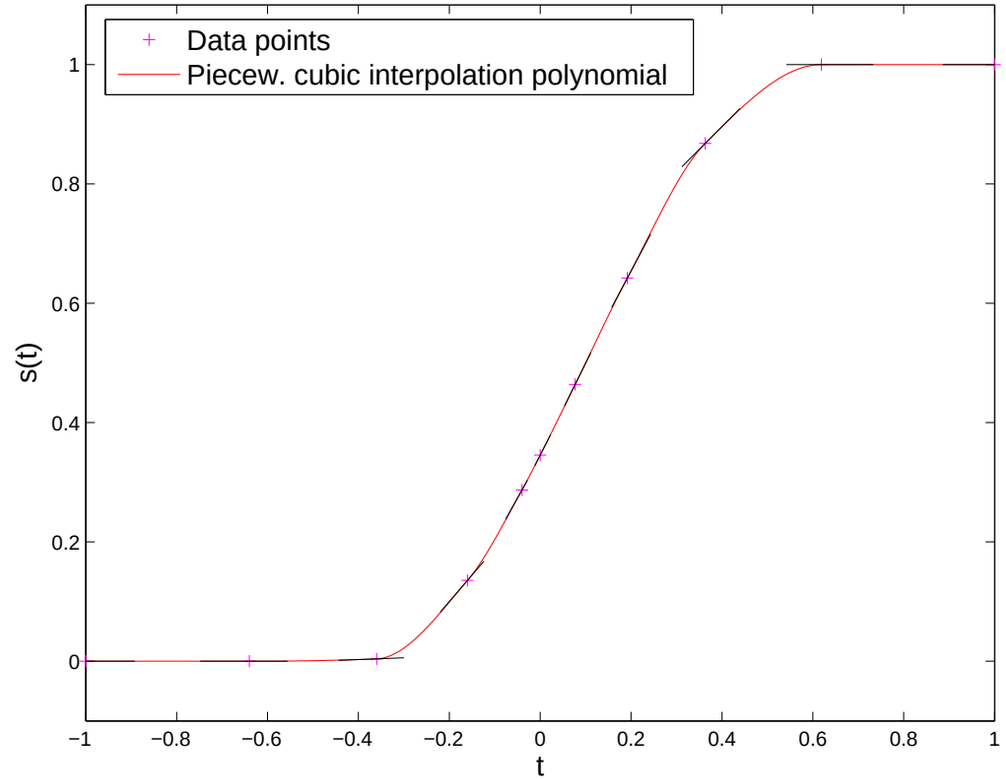
MATLAB-function:

```
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.*

R. Hiptmair

rev 38355, October 30, 2011

3.8 Splines [13, Ch. 9]

Definition 3.8.1 (Spline space). $\rightarrow$ *[51, Def. 8.1]*

Given an interval $I := [a,b] \subset \mathbb{R}$ *and a knot set/mesh* $\mathcal{M} := \{a = t_0 < t_1 < \ldots < t_{n-1} < t_n = b\}$, *the vector space* $\mathcal{S}_{d,\mathcal{M}}$ *of the spline functions of degree* d *(or order* $d+1$*) is defined by*

$$\mathcal{S}_{d,\mathcal{M}} := \{s \in C^{d-1}(I) \colon s_j := s_{|[t_{j-1},t_j]} \in \mathcal{P}_d \; \forall j = 1,\ldots,n\} \; .$$

$d-1$*-times continuously differentiable* locally *polynomial of degree* d

R. Hiptmair

rev 38355, October 30, 2011

- $d = 0$: $\mathcal{M}$-piecewise constant *discontinuous* functions
- $d = 1$: $\mathcal{M}$-piecewise linear *continuous* functions
- $d = 2$: *continuously differentiable* $\mathcal{M}$-piecewise quadratic functions

3.8.1 Cubic spline interpolation [35, XIII, 46], [51, Sect. 8.6.1]

Another special case: cubic spline interpolation, $d = 3$ (related to Hermite interpolation, Sect. 3.7)

(3.7.1)

$$\begin{aligned} s_{|[t_{j-1},t_j]}(t) = \; & s(t_{j-1}) & \cdot \; & (1 - 3\tau^2 + 2\tau^3) & + \\ & s(t_j) & \cdot \; & (3\tau^2 - 2\tau^3) & + \\ & h_j s'(t_{j-1}) & \cdot \; & (\tau - 2\tau^2 + \tau^3) & + \\ & h_j s'(t_j) & \cdot \; & (-\tau^2 + \tau^3) \; , & \end{aligned} \tag{3.8.3}$$

with $h_j := t_j - t_{j-1}$, $\tau := (t - t_{j-1})/h_j$.

$s \in C^2([t_0, t_n]) \quad \Rightarrow \quad n-1$ continuity constraints for $s''(t)$ at the internal nodes

$$s''_{|[t_{j-1},t_j]}(t_j) = s''_{|[t_j,t_{j+1}]}(t_j) \; , \quad j = 1, \dots, n-1 \; . \tag{3.8.4}$$

(3.8.4) ➔ $n-1$ linear equations for n slopes $c_j := s'(t_j)$

$$\frac{1}{h_j} c_{j-1} + \left(\frac{2}{h_j} + \frac{2}{h_{j+1}} \right) c_j + \frac{1}{h_{j+1}} c_{j+1} = 3 \left(\frac{y_j - y_{j-1}}{h_j^2} + \frac{y_{j+1} - y_j}{h_{j+1}^2} \right) \; , \tag{3.8.6}$$

for $j = 1, \dots, n-1$.

(3.8.6) $\Leftrightarrow$ *underdetermined* $(n-1) \times (n+1)$ linear system of equations

$$\begin{pmatrix} b_0 & a_1 & b_1 & 0 & \cdots & & \cdots & 0 \\ 0 & b_1 & a_2 & b_2 & & & & \\ & 0 & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & a_{n-2} & b_{n-2} & 0 \\ 0 & \cdots & & \cdots & 0 & b_{n-2} & a_{n-1} & b_{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} 3\left(\frac{y_1-y_0}{h_1^2} + \frac{y_2-y_1}{h_2^2}\right) \\ \vdots \\ 3\left(\frac{y_{n-1}-y_{n-2}}{h_{n-1}^2} + \frac{y_n-y_{n-1}}{h_n^2}\right) \end{pmatrix} . \quad (3.8.7)$$

with

$$\begin{aligned} b_i &:= \frac{1}{h_{i+1}}, \quad i = 0, 1, \dots, n-1 \,, \\ a_i &:= \frac{2}{h_i} + \frac{2}{h_{i+1}}, \quad i = 0, 1, \dots, n-1 \,. \\ & [\quad b_i, a_i > 0 \quad , \quad a_i = 2(b_i + b_{i-1}) \quad] \end{aligned}$$

R. Hiptmair

rev 38355, October 30, 2011

➜ 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: $s''(t_0) = s''(t_n) = 0$

$$\frac{2}{h_1}c_0 + \frac{1}{h_1}c_1 = 3\,\frac{y_1 - y_0}{h_1^2} \quad , \quad \frac{1}{h_n}c_{n-1} + \frac{2}{h_n}c_n = 3\,\frac{y_n - y_{n-1}}{h_n^2} \; .$$

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

$$\mathbf{A} := \begin{pmatrix} a_1 & b_1 & 0 & \cdots & 0 & b_0 \\ b_1 & a_2 & b_2 & & & 0 \\ 0 & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & & & \ddots & a_{n-1} & b_{n-1} \\ b_0 & 0 & \cdots & 0 & b_{n-1} & a_0 \end{pmatrix} \quad , \quad \begin{aligned} b_i &:= \frac{1}{h_{i+1}}, \quad i = 0,1,\dots,n-1 \; , \\ a_i &:= \frac{2}{h_i} + \frac{2}{h_{i+1}}, \quad i = 0,1,\dots,n-1 \; . \end{aligned}$$

MATLAB-function: `v = spline(t,y,x)`: natural / complete spline interpolation
(see spline-toolbox in MATLAB)

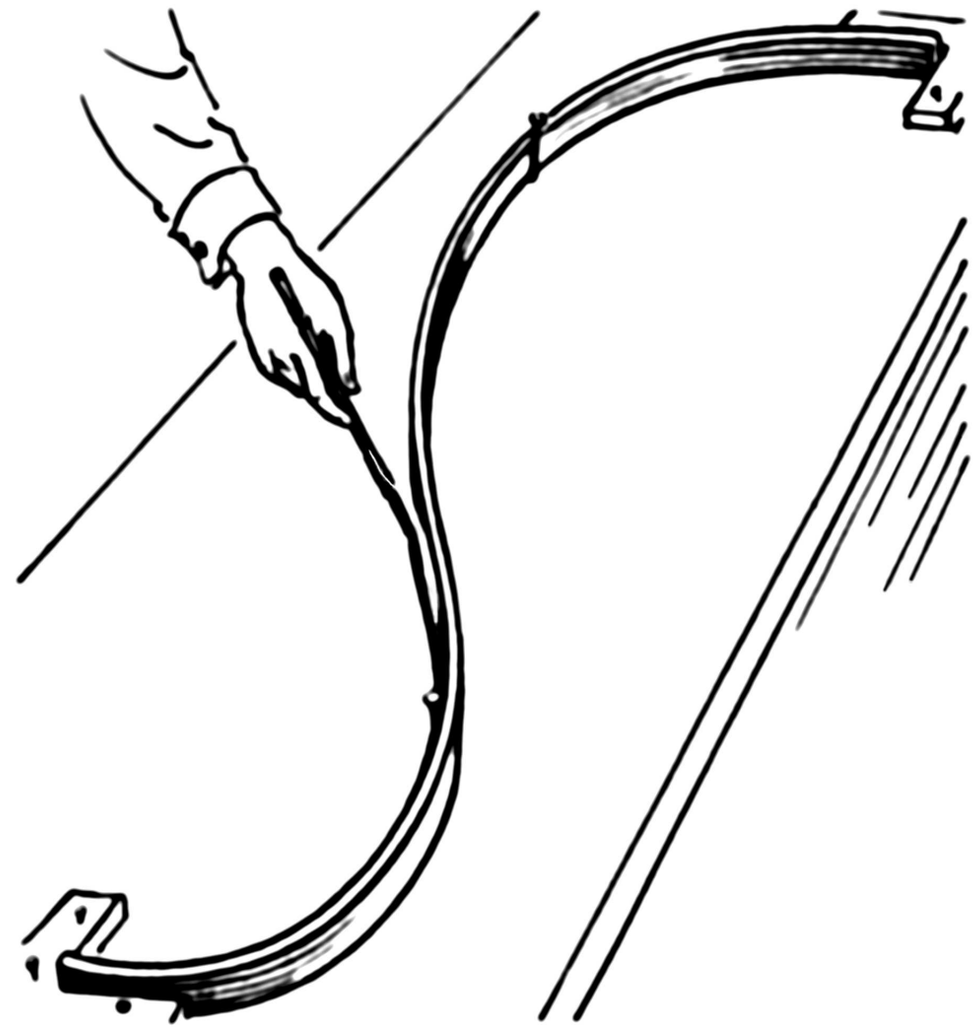
Structural properties of cubic spline interpolants

Remark 3.8.9 (Extremal properties of natural cubic spline interpolants). → [51, Sect. 8.6.1, Property 8.2]

Remark 3.8.12 (Origin of the term “Spline”).

Cubic spline interpolation

before MATLAB

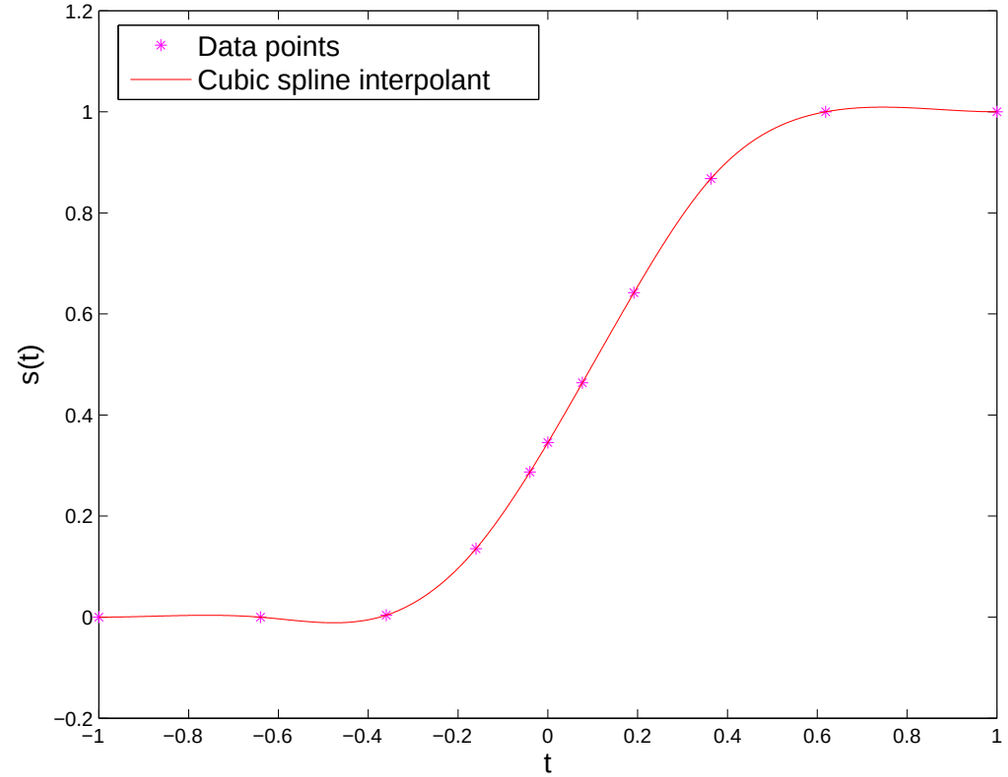


Remark 3.8.13 (Shape preservation).

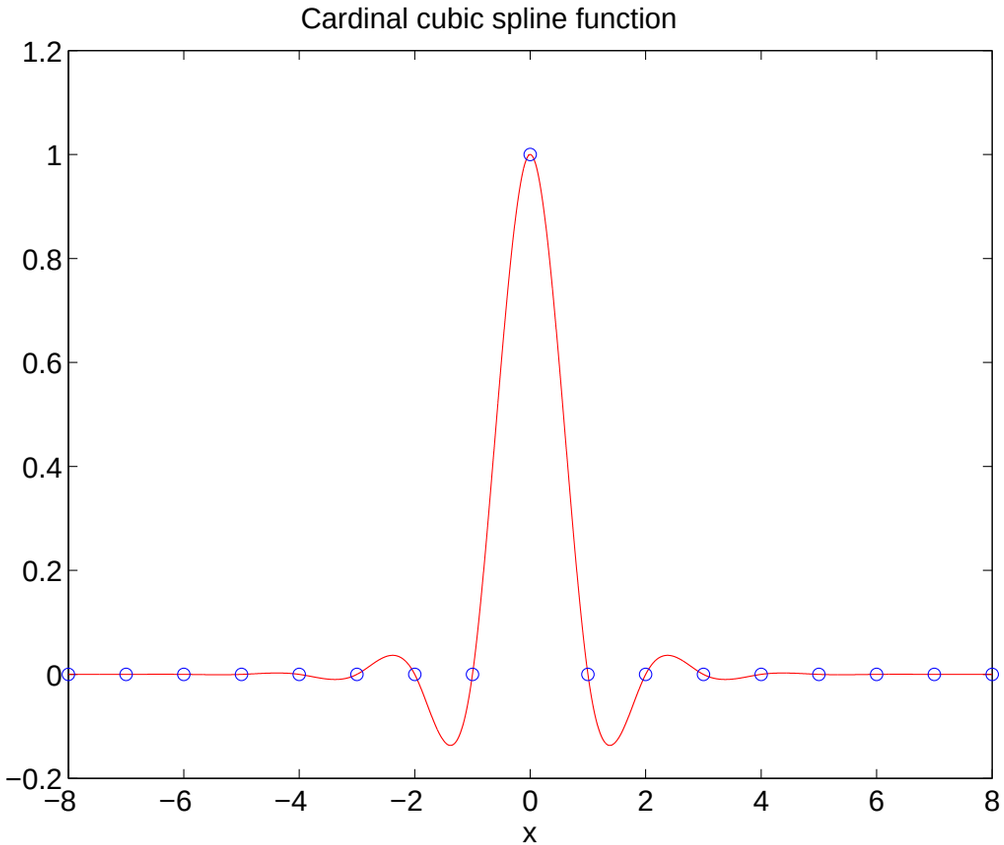
Data $s(t_j) = y_j$ from Ex. 3.6.5 and

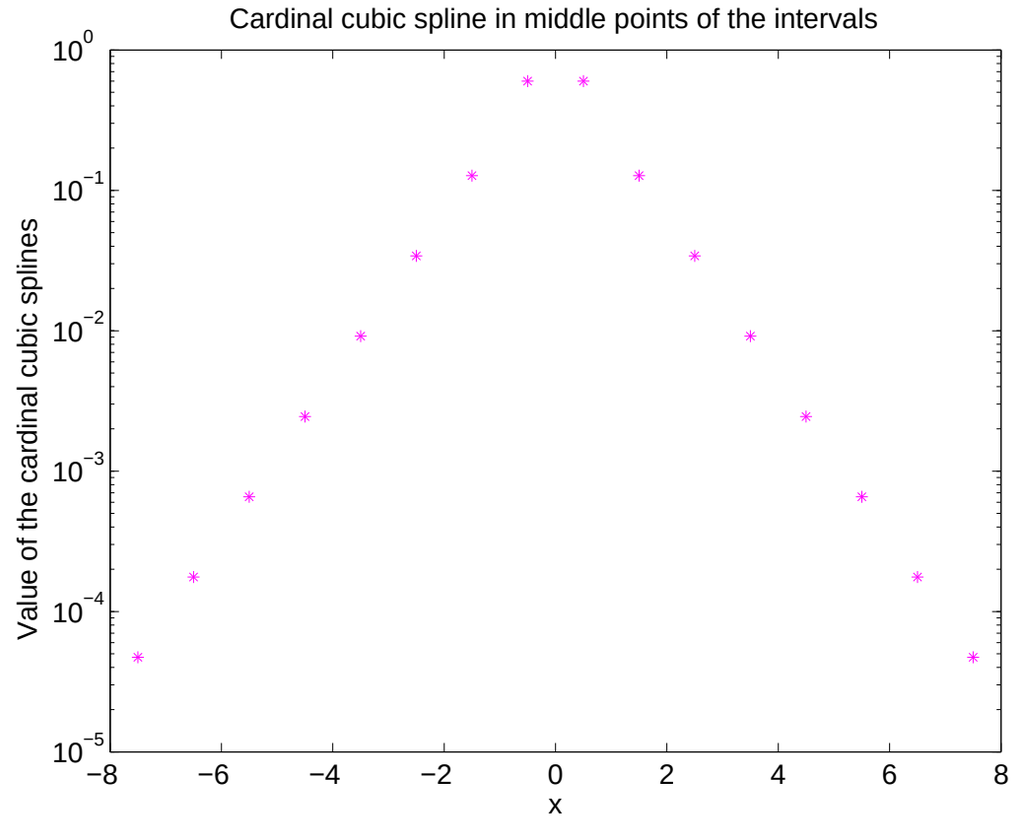
$$c_0 := \frac{y_1 - y_0}{t_1 - t_0} \,,$$
$$c_n := \frac{y_n - y_{n-1}}{t_n - t_{n-1}} \,.$$

△



Example 3.8.14 (Locality of the natural cubic spline interpolation).





◇

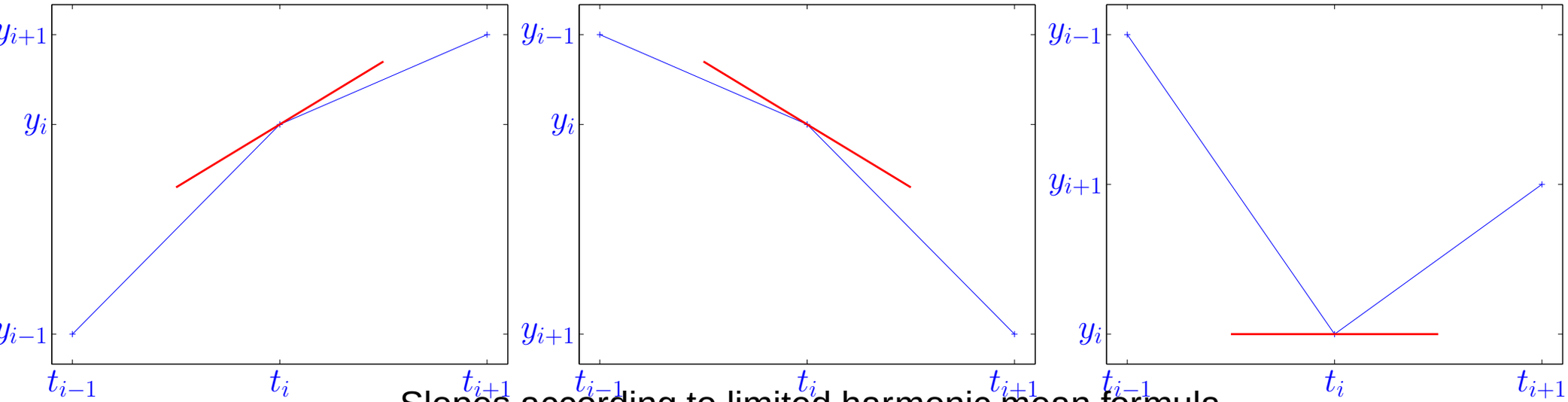
3.8.2 Shape Preserving Spline Interpolation

Given: data points $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \dots, n$, assume ordering $t_0 < t_1 < \cdots < t_n$.

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)

① Shape preserving choice of slopes c_i, $i = 0, \dots, n$ [43, 50], analogous to Sect. 3.7.2



Slopes according to limited harmonic mean formula

② Choice of "extra knots" $p_i \in]t_{i-1}, t_i]$, $i = 1, \dots, n$:

These points will be used to build the knot set for the final quadratic spline:

$$\mathcal{M} = \{t_0 < p_1 \leq t_1 < p_2 \leq \cdots < p_n \leq t_n\} \ .$$

```
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
```

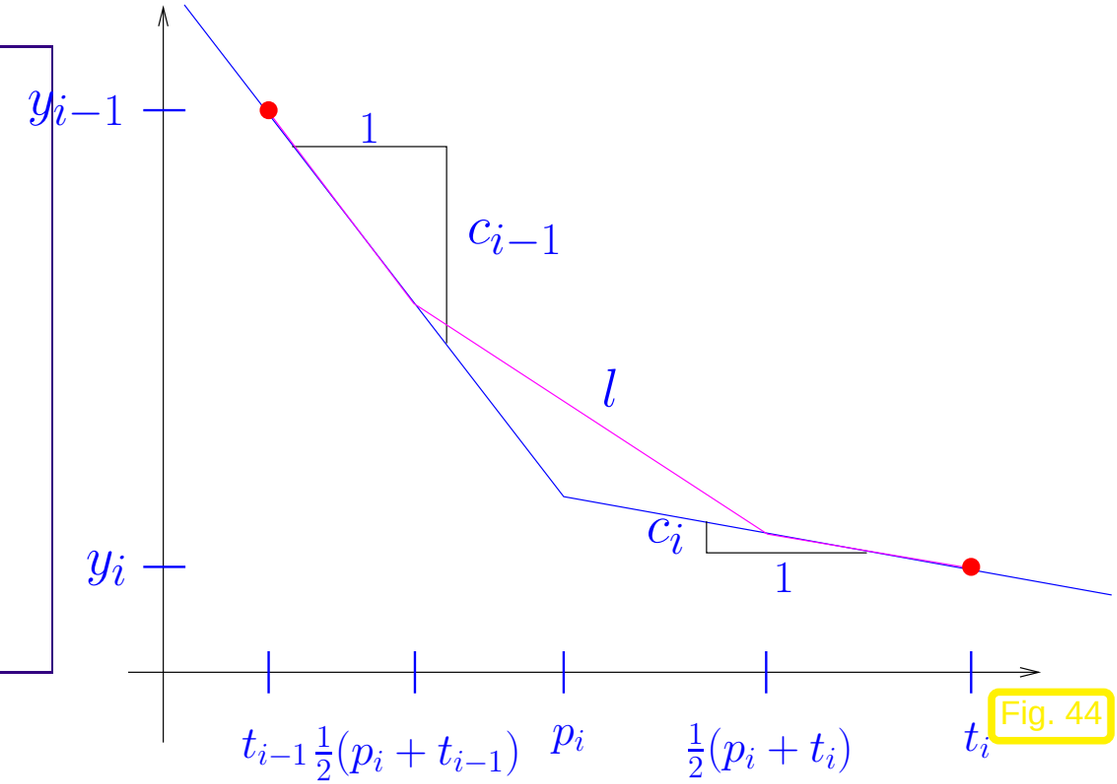


Fig. 44

R. Hiptmair

rev 38355, October 1, 2011

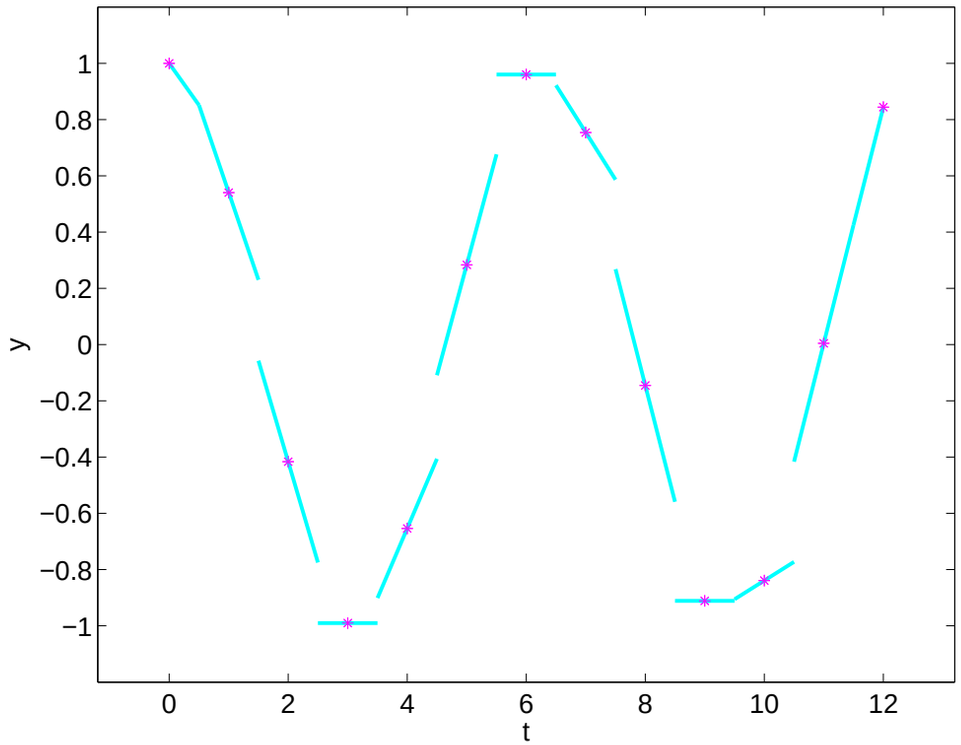
③ Set l = linear spline (polygon) on the knot set $\mathcal{M}'$ (middle points of $\mathcal{M}$)

$$\mathcal{M}' = \{t_0 < \tfrac{1}{2}(t_0 + p_1) < \tfrac{1}{2}(p_1 + t_1) < \tfrac{1}{2}(t_1 + p_2) < \cdots < \tfrac{1}{2}(t_{n-1} + p_n) < \tfrac{1}{2}(p_n + t_n) < t_n\} ;$$

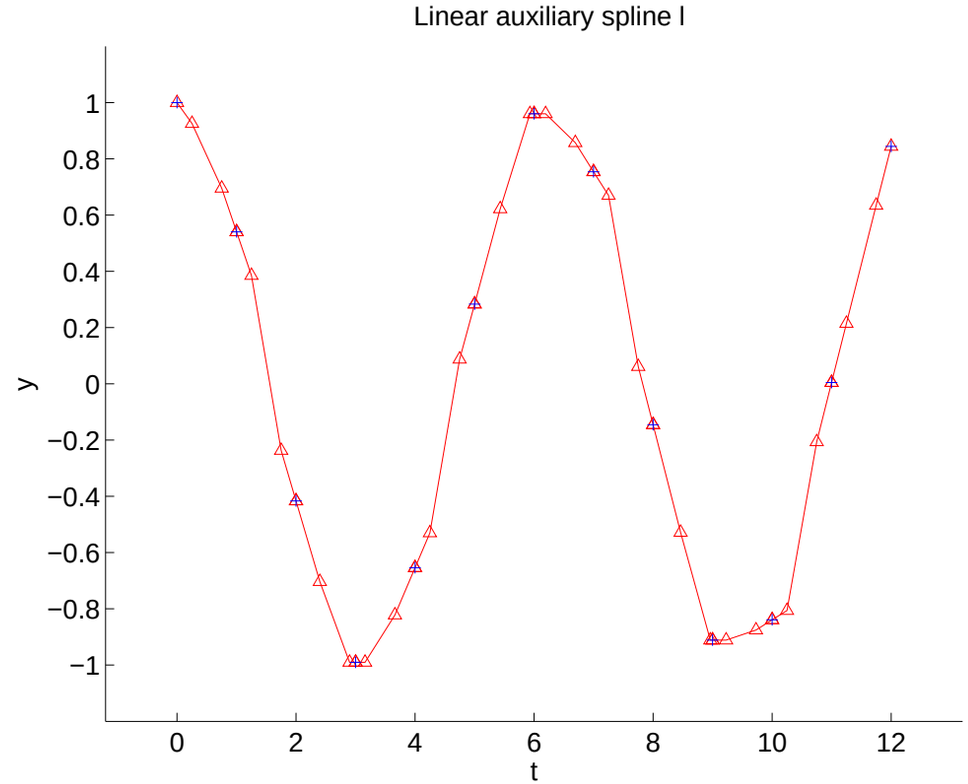
with $l(t_i) = y_i$, $l'(t_i) = c_i$.

Example 3.8.16 (Auxiliary construction for shape preserving quadratic spline interpolation).

Data points: `t=(0:12); y = cos(t);`



Local slopes c_i, $i = 0, \ldots, n$



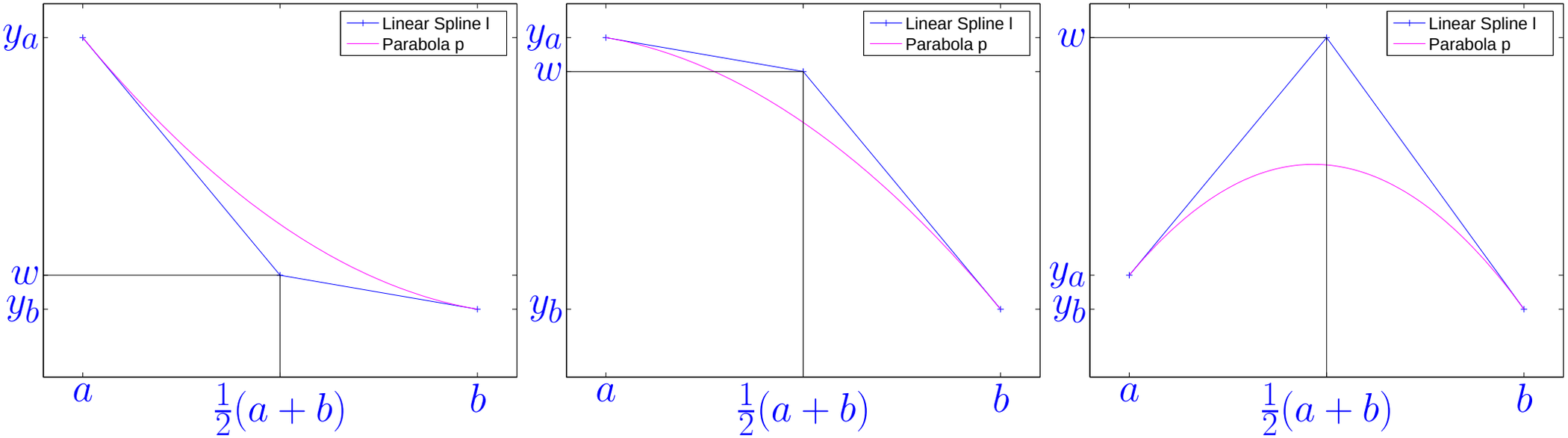
Linear auxiliary spline l

◇

R. Hiptmair

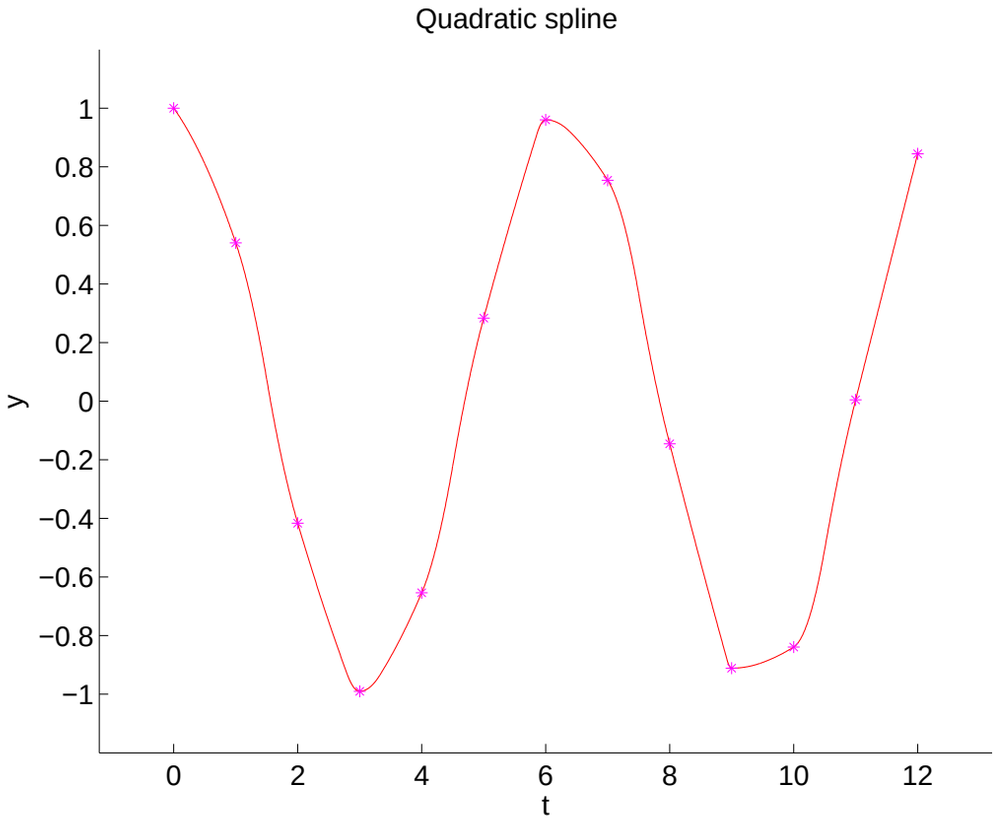
rev 38355, October 1, 2011

④ Local quadratic approximation / interpolation of l:



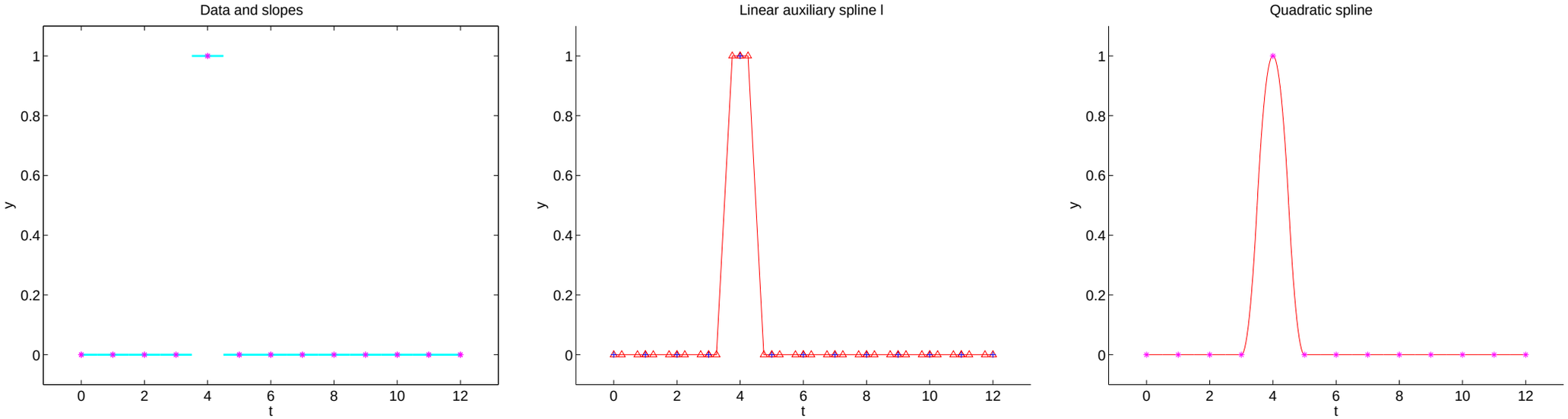
Continuation of Ex. 3.8.16:

Interpolating quadratic spline →



R. Hiptmair

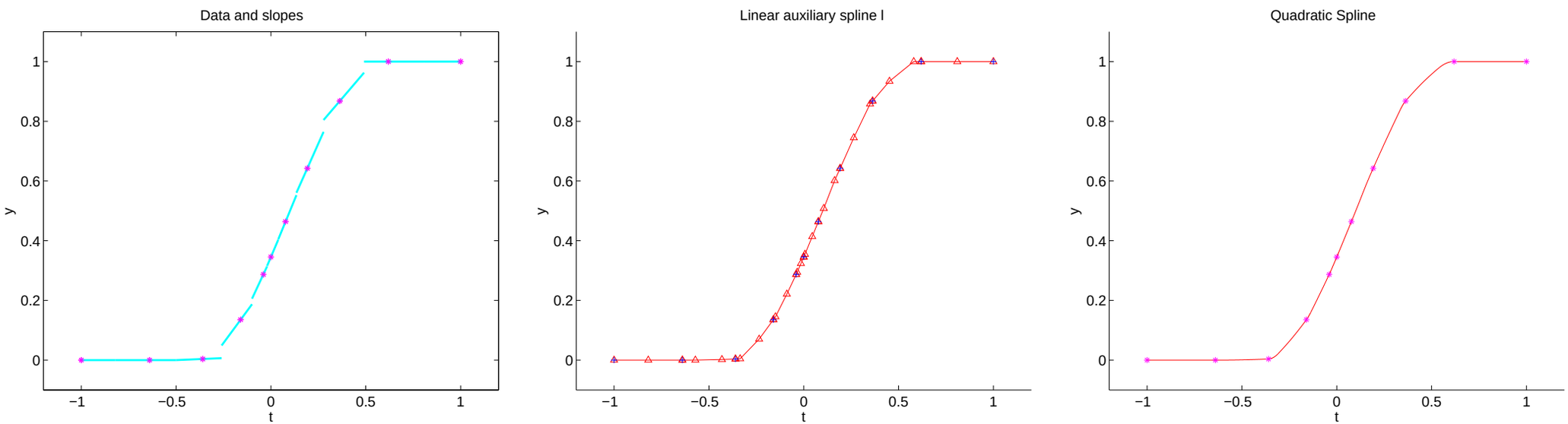
rev 38355, October 1, 2011



Shape preserving quadratic spline interpolation = local **but** not linear

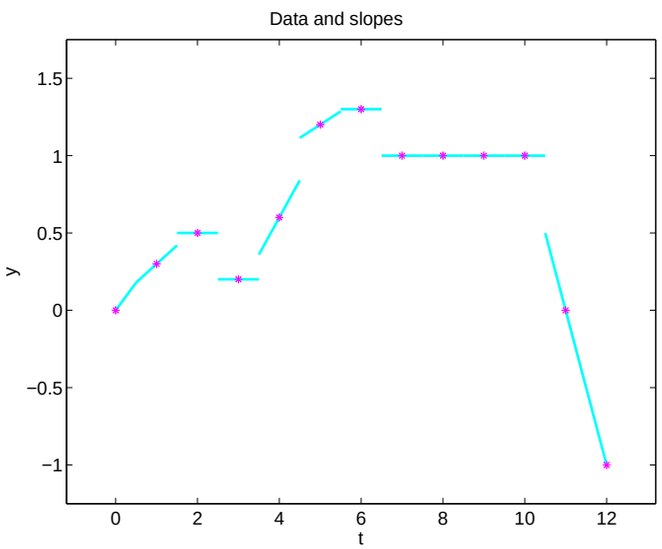
Example 3.8.17 (Shape preserving quadratic spline interpolation).

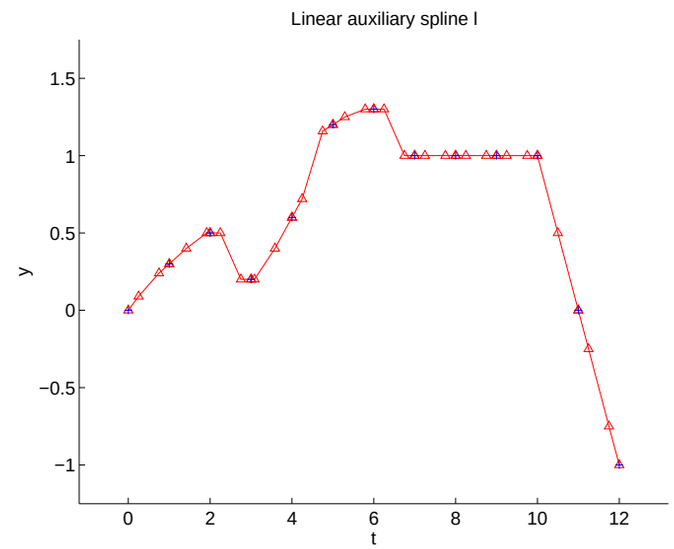
Data from Ex. 3.6.5:

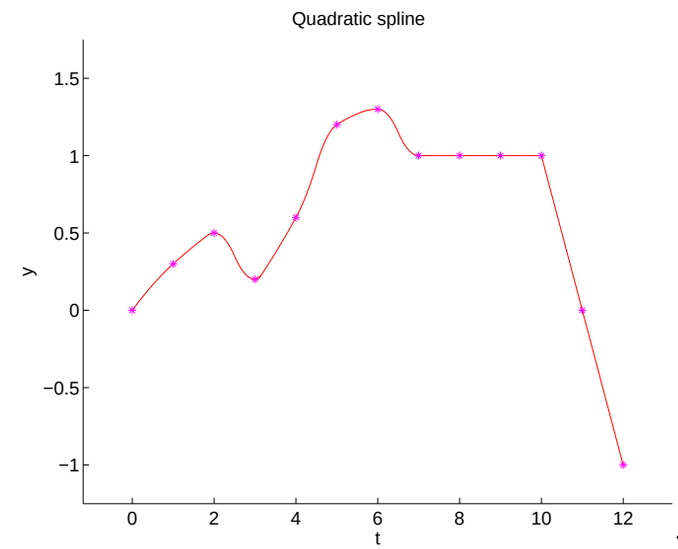


Data from [43]:

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







◇

R. Hiptmair

rev 38355, October 1, 2011

4 Iterative Methods for Non-Linear Systems of Equations

Example 4.0.1 (Non-linear electric circuit).

Schmitt trigger circuit ▷

NPN bipolar junction transistor:

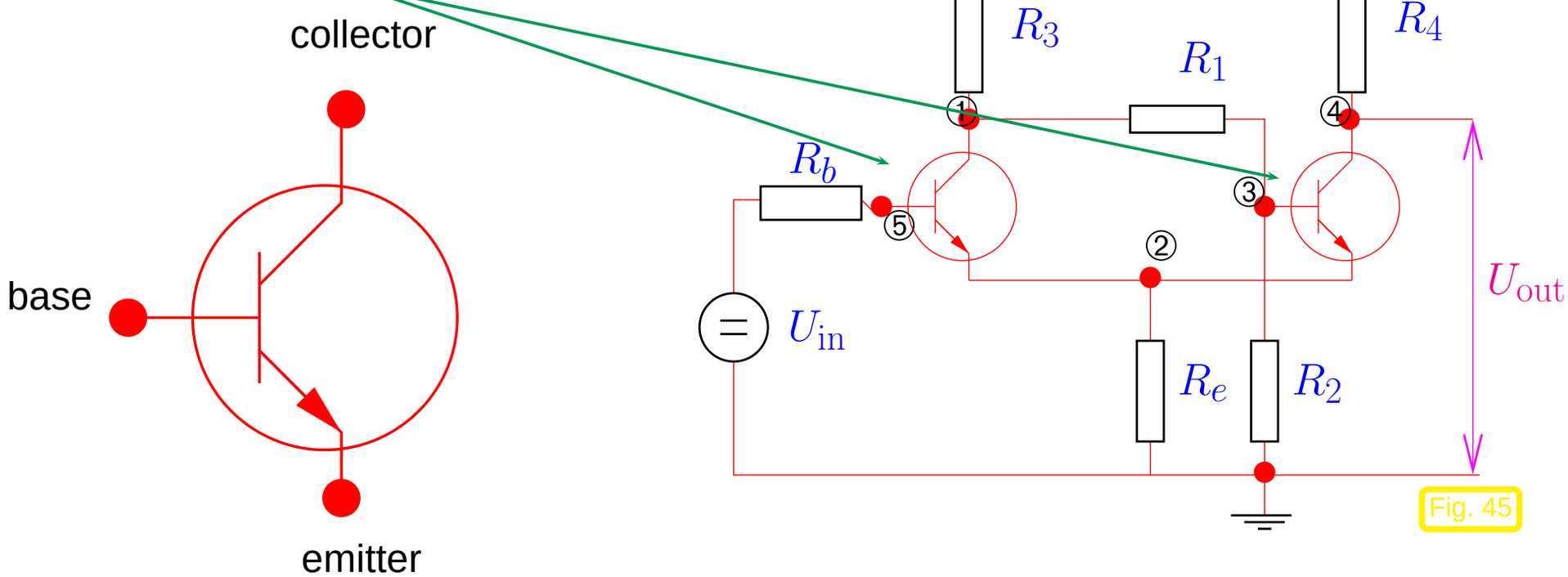


Fig. 45

R. Hiptmair

rev 38355, October 21, 2010

Ebers-Moll model (large signal approximation):

$$
\begin{aligned}
I_{\mathsf{C}} &= I_{\mathsf{S}}\left(e^{\frac{U_{\mathsf{BE}}}{U_{\mathsf{T}}}} - e^{\frac{U_{\mathsf{BC}}}{U_{\mathsf{T}}}}\right) - \frac{I_{\mathsf{S}}}{\beta_R}\left(e^{\frac{U_{\mathsf{BC}}}{U_{\mathsf{T}}}} - 1\right) = I_{\mathsf{C}}(U_{\mathsf{BE}}, U_{\mathsf{BC}}) \;, \\
I_{\mathsf{B}} &= \frac{I_{\mathsf{S}}}{\beta_F}\left(e^{\frac{U_{\mathsf{BE}}}{U_{\mathsf{T}}}} - 1\right) + \frac{I_{\mathsf{S}}}{\beta_R}\left(e^{\frac{U_{\mathsf{BC}}}{U_{\mathsf{T}}}} - 1\right) = I_{\mathsf{B}}(U_{\mathsf{BE}}, U_{\mathsf{BC}}) \;, \\
I_{\mathsf{E}} &= I_{\mathsf{S}}\left(e^{\frac{U_{\mathsf{BE}}}{U_{\mathsf{T}}}} - e^{\frac{U_{\mathsf{BC}}}{U_{\mathsf{T}}}}\right) + \frac{I_{\mathsf{S}}}{\beta_F}\left(e^{\frac{U_{\mathsf{BE}}}{U_{\mathsf{T}}}} - 1\right) = I_{\mathsf{E}}(U_{\mathsf{BE}}, U_{\mathsf{BC}}) \;.
\end{aligned}
\tag{4.0.2}
$$

I_{C}, I_{B}, I_{E}: current in collector/base/emitter,
U_{BE}, U_{BC}: potential drop between base-emitter, base-collector.

R. Hiptmair

rev 38355, October 21, 2010

Non-linear system of equations from nodal analysis (→ Ex. 2.6.3):

$$
\begin{aligned}
① &: \quad R_3(U_1 - U_+) + R_1(U_1 - U_3) + I_{\mathsf{B}}(U_5 - U_1, U_5 - U_2) = 0 \;, \\
② &: \quad R_e U_2 + I_{\mathsf{E}}(U_5 - U_1, U_5 - U_2) + I_{\mathsf{E}}(U_3 - U_4, U_3 - U_2) = 0 \;, \\
③ &: \quad R_1(U_3 - U_1) + I_{\mathsf{B}}(U_3 - U_4, U_3 - U_2) = 0 \;, \\
④ &: \quad R_4(U_4 - U_+) + I_{\mathsf{C}}(U_3 - U_4, U_3 - U_2) = 0 \;, \\
⑤ &: \quad R_b(U_5 - U_{\mathrm{in}}) + I_{\mathsf{B}}(U_5 - U_1, U_5 - U_2) = 0 \;.
\end{aligned}
\tag{4.0.3}
$$

5 equations $\leftrightarrow$ 5 unknowns U_1, U_2, U_3, U_4, U_5

Formally: (4.0.3) $\longleftrightarrow$ $F(\mathbf{u}) = 0$

Given: function $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$, $n \in \mathbb{N}$

$$\Updownarrow$$

Possible meaning: Availability of a procedure `function y=F(x)` evaluating F

Sought: solution of non-linear equation $F(\mathbf{x}) = 0$

In general no existence & uniqueness of solutions

4.1 Iterative methods

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.

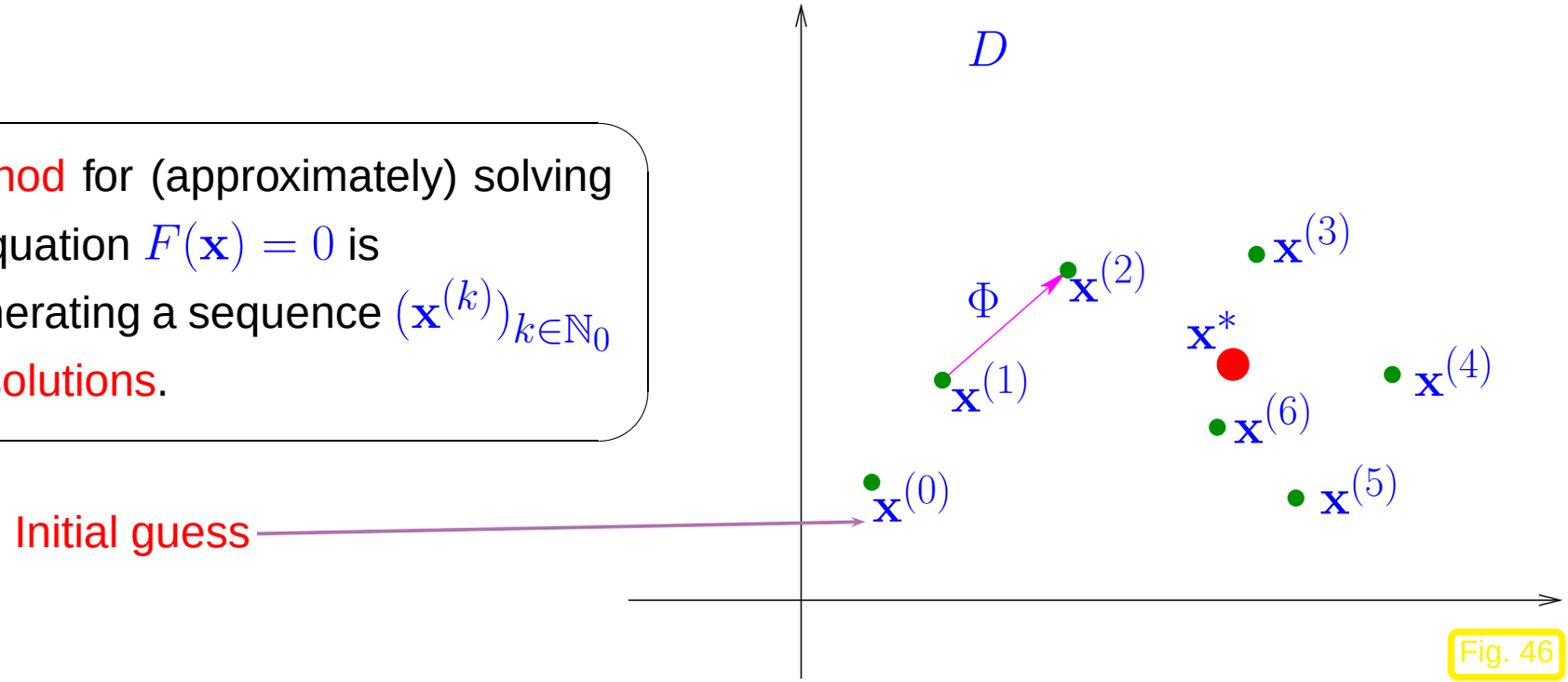


Fig. 46

Fundamental concepts: convergence ➡ speed of convergence

consistency

R. Hiptmair

rev 38355, February 24, 2009

local convergence ▷

(Only initial guesses "sufficiently close" to $\mathbf{x}^*$ guarantee convergence.)

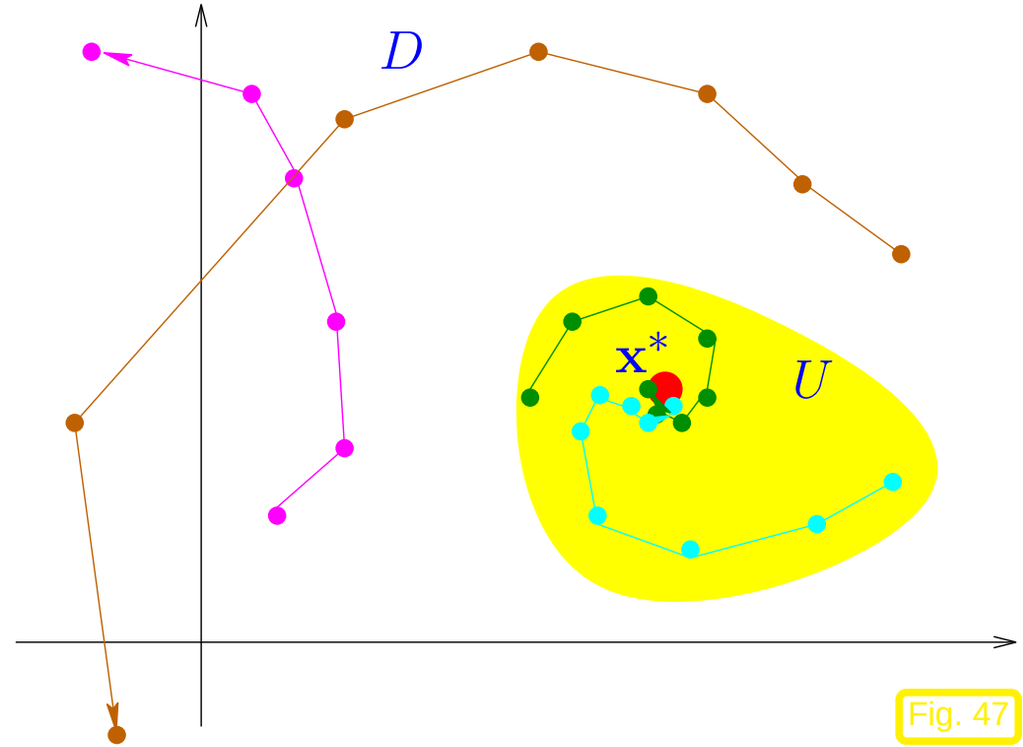


Fig. 47

Goal: Find iterative methods that converge (locally) to a solution of $F(\mathbf{x}) = 0$.

4.1.1 Speed of convergence

"*Speed of convergence*" $\leftrightarrow$ decrease of norm (see Def. 2.5.1) of iteration error

Definition 4.1.6 (Linear convergence)**.**
A sequence $\mathbf{x}^{(k)}$, $k = 0, 1, 2, \ldots$, *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)*.*

Fact of convergence of iteration is	independent	of choice of norm
Fact of linear convergence	depends	on choice of norm
Rate of linear convergence	depends	on choice of norm

△

Example 4.1.12 (Linearly convergent iteration)*.*

Iteration ($n = 1$):

$$x^{(k+1)} = x^{(k)} + \frac{\cos x^{(k)} + 1}{\sin x^{(k)}} .$$

Code 4.1.13: simple fixed point iteration

```
y  = [ ];
for i = 1:15
  x = x + (cos(x)+1)/sin(x);
  y = [y,x];
end
err = y - x;
rate = err(2:15)./err(1:14);
```

Note: $x^{(15)}$ replaces the exact solution x^* in the computation of the rate of convergence.

k	$x^{(0)} = 0.4$		$x^{(0)} = 0.6$		$x^{(0)} = 1$	
	$x^{(k)}$	$\frac{\lvert x^{(k)}-x^{(15)}\rvert}{\lvert x^{(k-1)}-x^{(15)}\rvert}$	$x^{(k)}$	$\frac{\lvert x^{(k)}-x^{(15)}\rvert}{\lvert x^{(k-1)}-x^{(15)}\rvert}$	$x^{(k)}$	$\frac{\lvert x^{(k)}-x^{(15)}\rvert}{\lvert x^{(k-1)}-x^{(15)}\rvert}$
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

Rate of convergence ≈ 0.5

Linear convergence as in Def. 4.1.6

$\Updownarrow$

error graphs $=$ straight lines in lin-log scale

$\rightarrow$ Rem. 4.1.10

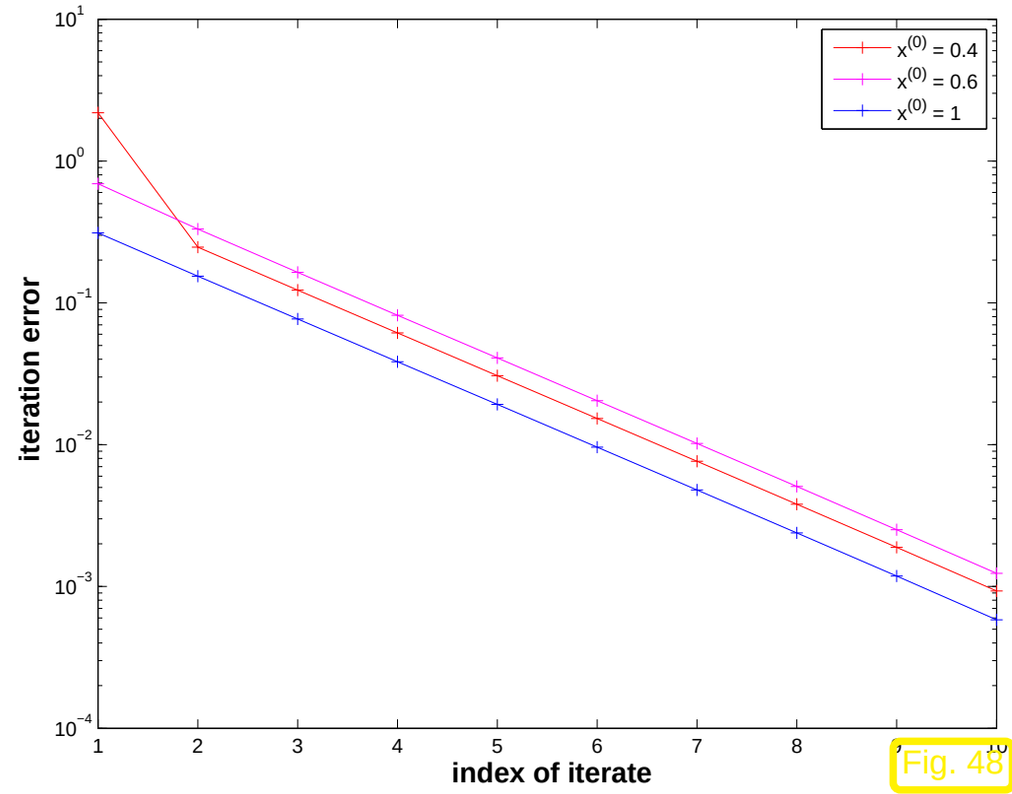


Fig. 48

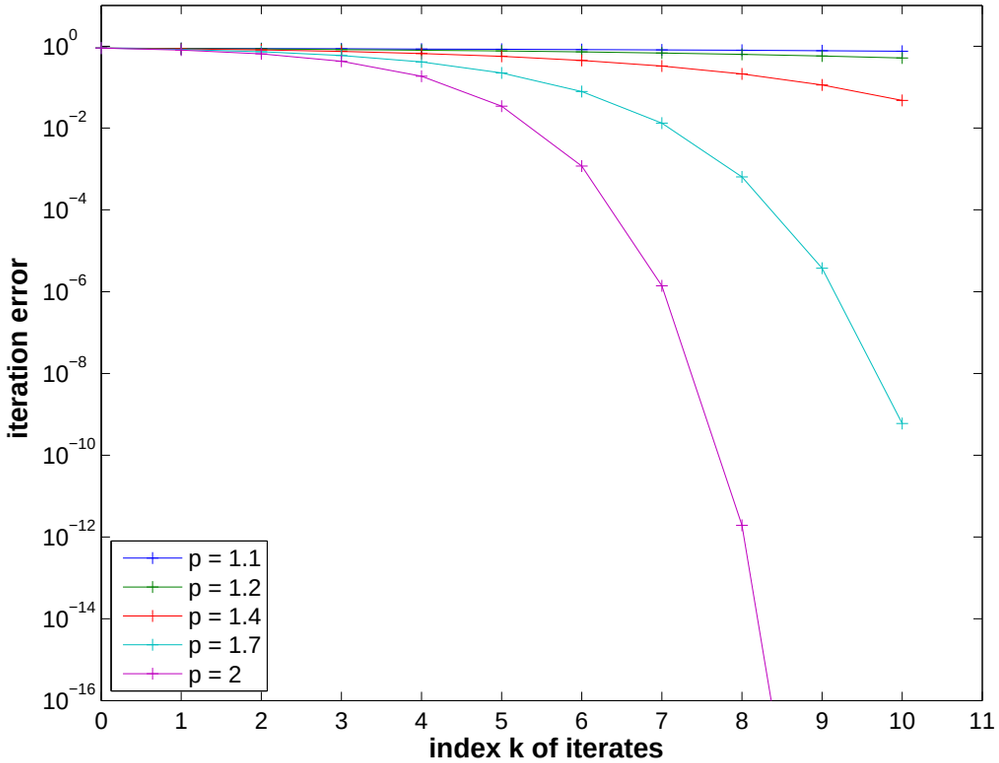
$\diamond$

Definition 4.1.14 (Order of convergence). $\rightarrow$ *[35, Sect. 17.2], [13, Def. 5.14], [51, Def. 6.1]*

*A **convergent** sequence $\mathbf{x}^{(k)}$, $k = 0, 1, 2, \ldots$, 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 $\rightarrow$ Def. 4.1.6).



◁ Qualitative error graphs for convergence of order p
(lin-log scale)

Example 4.1.16 (quadratic convergence). (= convergence of order 2)

Iteration for computing $\sqrt{a}$, $a > 0$:

$$x^{(k+1)} = \frac{1}{2}(x^{(k)} + \frac{a}{x^{(k)}}) \quad \Rightarrow \quad |x^{(k+1)} - \sqrt{a}| = \frac{1}{2x^{(k)}}|x^{(k)} - \sqrt{a}|^2 \,. \tag{4.1.17}$$

Numerical experiment: iterates for $a = 2$:

k	$x^{(k)}$	$e^{(k)} := x^{(k)} - \sqrt{2}$	$\log \frac{\lvert e^{(k)}\rvert}{\lvert e^{(k-1)}\rvert} : \log \frac{\lvert e^{(k-1)}\rvert}{\lvert e^{(k-2)}\rvert}$
0	2.00000000000000000	0.58578643762690485	
1	1.50000000000000000	0.08578643762690485	
2	1.41666666666666652	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 !]

◇

R. Hiptmair

rev 38355, February 24, 2009

4.1.2 Termination criteria

④ Residual based termination: STOP convergent iteration $\{\mathbf{x}^{(k)}\}_{k\in\mathbb{N}_0}$, when

$$\left\| F(\mathbf{x}^{(k)}) \right\| \leq \tau \ , \qquad \tau \hat{=} \text{ prescribed tolerance } > 0 \ .$$

☹ no guaranteed accuracy

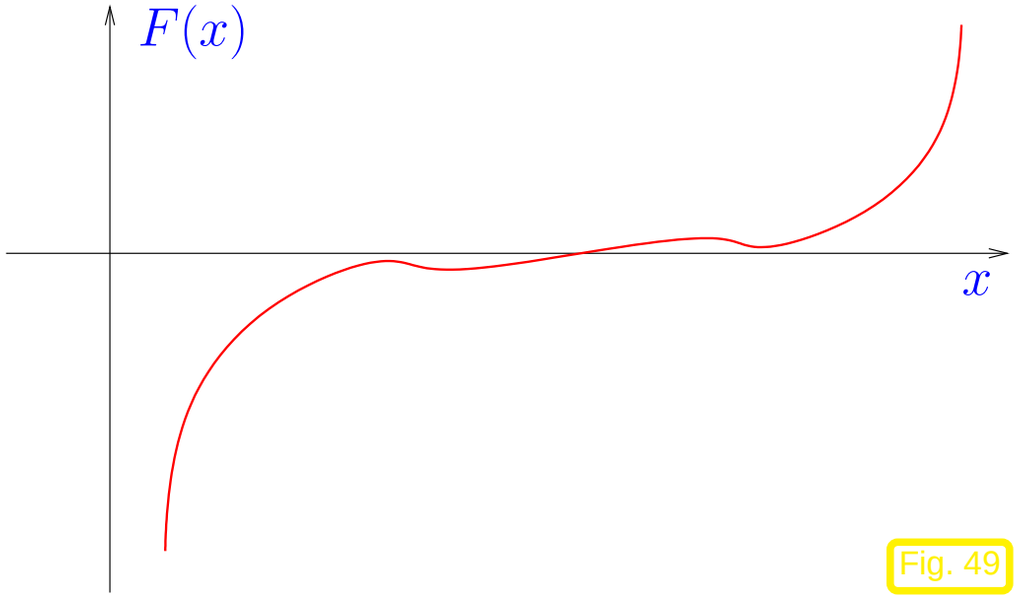


Fig. 49

$\left\| F(\mathbf{x}^{(k)}) \right\|$ small $\not\Rightarrow$ $|x - x^*|$ small

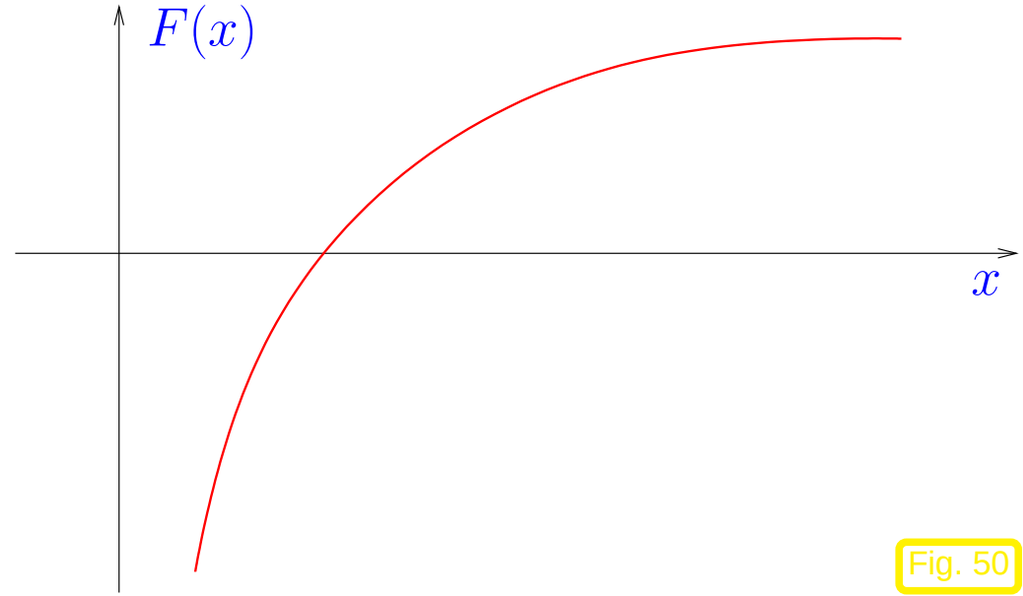


Fig. 50

$\left\| F(\mathbf{x}^{(k)}) \right\|$ small $\Rightarrow$ $|x - x^*|$ small

⑤

Correction based termination: STOP convergent iteration $\{\mathbf{x}^{(k)}\}_{k\in\mathbb{N}_0}$, when

$$\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\| \leq \begin{cases} \tau_{\text{abs}} \\ \textbf{or} \\ \tau_{\text{rel}} \left\| \mathbf{x}^* \right\| \end{cases} , \qquad \begin{matrix} \tau_{\text{abs}} \\ \tau_{\text{rel}} \end{matrix} \text{ prescribed } \begin{matrix} \text{absolute} \\ \text{relative} \end{matrix} \text{ tolerances } > 0 \,.$$

Remark 4.1.23 (A posteriori termination criterion for linearly convergent iterations). $\rightarrow$ [13, Lemma 5.

Known: iteration linearly convergent with rate of convergence $0 < L < 1$:

R. Hiptmair

rev 38355, February 24, 2009

Iterates satisfy: $$\boxed{\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \leq \frac{L}{1-L}\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\|} \quad . \tag{4.1.24}$$

△

Example 4.1.25 (A posteriori error bound for linearly convergent iteration).

Iteration of Example 4.1.12:

$$x^{(k+1)} = x^{(k)} + \frac{\cos x^{(k)} + 1}{\sin x^{(k)}} \quad \Rightarrow x^{(k)} \to \pi \quad \text{for } x^{(0)} \text{ close to } \pi \ .$$

Observed rate of convergence: $L = 1/2$

Error and error bound for $x^{(0)} = 0.4$:

k	$\lvert x^{(k)} - \pi \rvert$	$\frac{L}{1-L}\lvert x^{(k)} - x^{(k-1)} \rvert$	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!

◇

R. Hiptmair

rev 38355, October 3, 2011

4.2 Fixed Point Iterations [13, Sect. 5.3], [51, Sect. 6.3]

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 $\mathbf{x}^{(0)} \in U$

$$\rightarrow \underbrace{\boxed{\text{iterates } (\mathbf{x}^{(k)})_{k\in\mathbb{N}_0}: \quad \mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})}}_{\rightarrow \text{ 1-point method, } cf. \text{ (4.1.2)}} .$$

4.2.1 Consistent fixed point iterations

R. Hiptmair

rev 38355, October 3, 2011

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*

$$F(\mathbf{x}) = 0 \quad \textit{and} \quad \mathbf{x} \in U \cap D \quad \Leftrightarrow \quad \Phi(\mathbf{x}) = \mathbf{x} .$$

General construction of fixed point iterations that is consistent with $F(\mathbf{x}) = 0$:
rewrite $F(\mathbf{x}) = 0 \quad \Leftrightarrow \quad \Phi(\mathbf{x}) = \mathbf{x}$ and then

use the fixed point iteration $\quad \mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})$. (4.2.2)

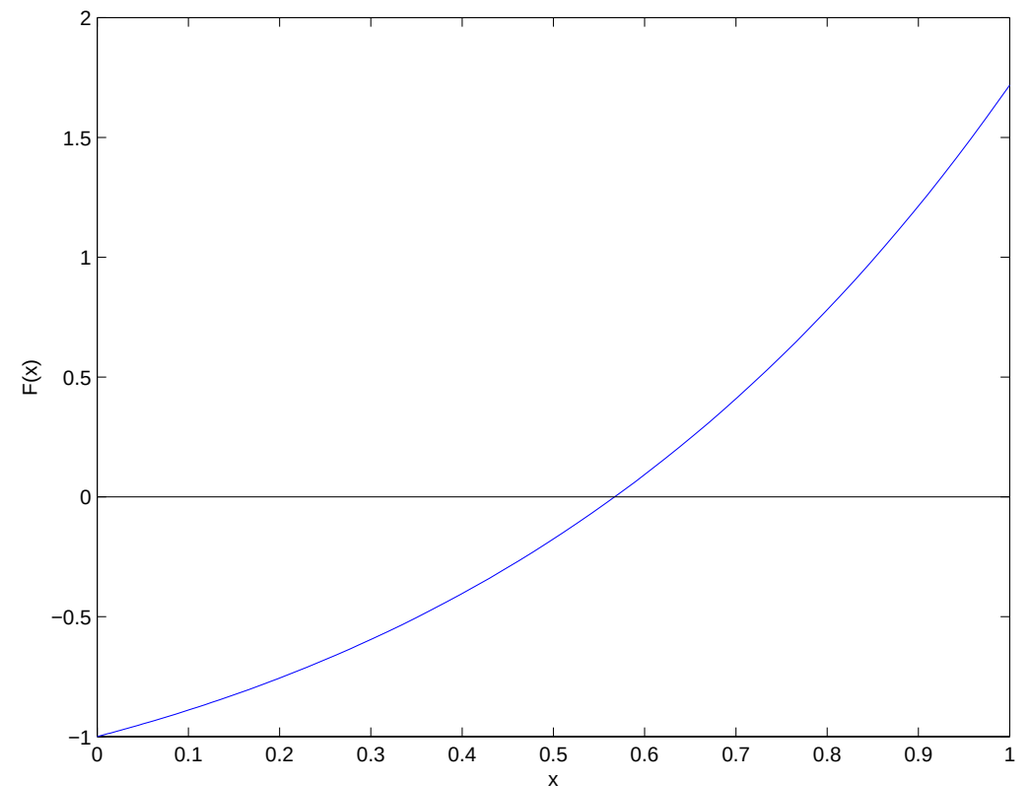
Note: there are *many ways* to transform $F(\mathbf{x}) = 0$ into a fixed point form **!**

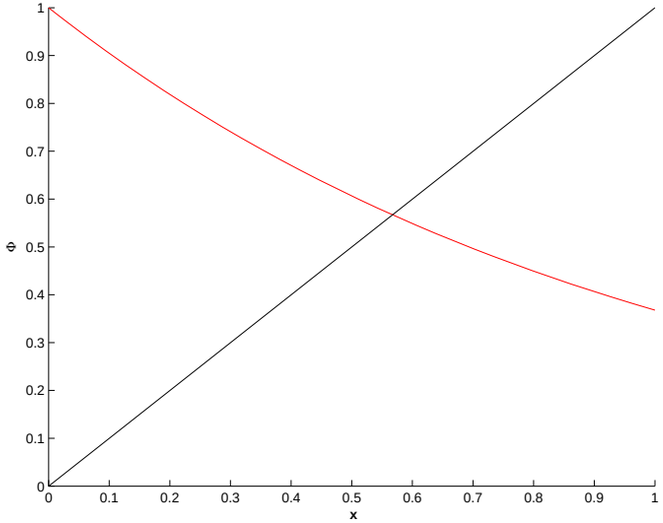
Example 4.2.3 (Options for fixed point iterations).

$$F(x) = xe^x - 1 \ , \quad x \in [0,1] \ .$$

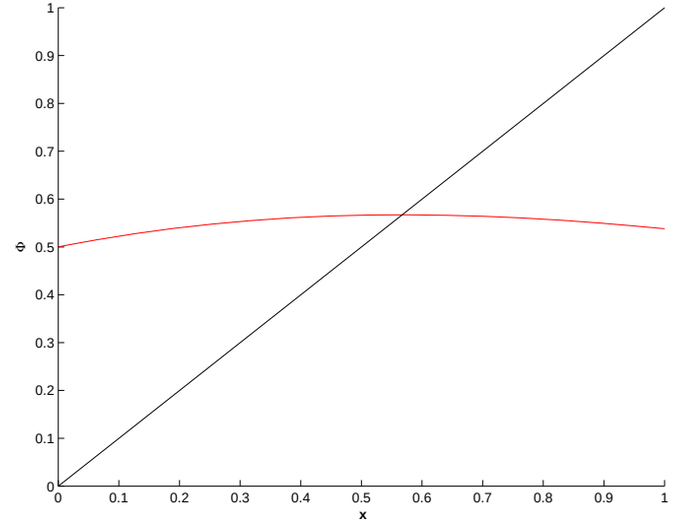
Different fixed point forms:

$$\begin{aligned} \Phi_1(x) &= e^{-x} \ , \\ \Phi_2(x) &= \frac{1+x}{1+e^x} \ , \\ \Phi_3(x) &= x + 1 - xe^x \ . \end{aligned}$$

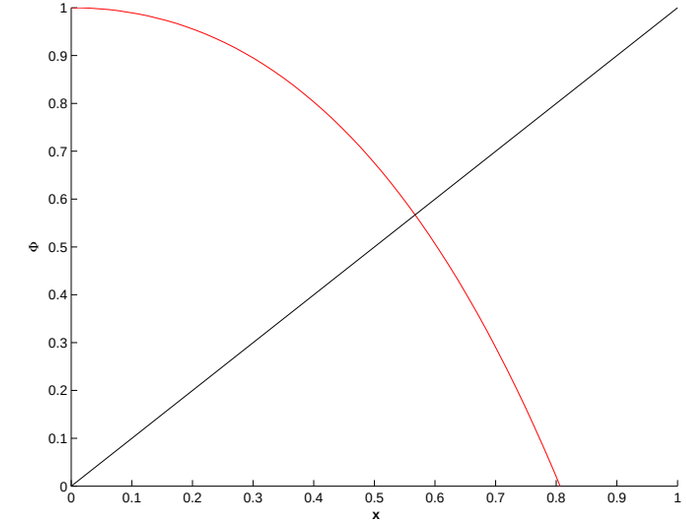




function Φ_1



function Φ_2



function Φ_3

k	$x^{(k+1)} := \Phi_1(x^{(k)})$	$x^{(k+1)} := \Phi_2(x^{(k)})$	$x^{(k+1)} := \Phi_3(x^{(k)})$
0	0.500000000000000	0.500000000000000	0.500000000000000
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

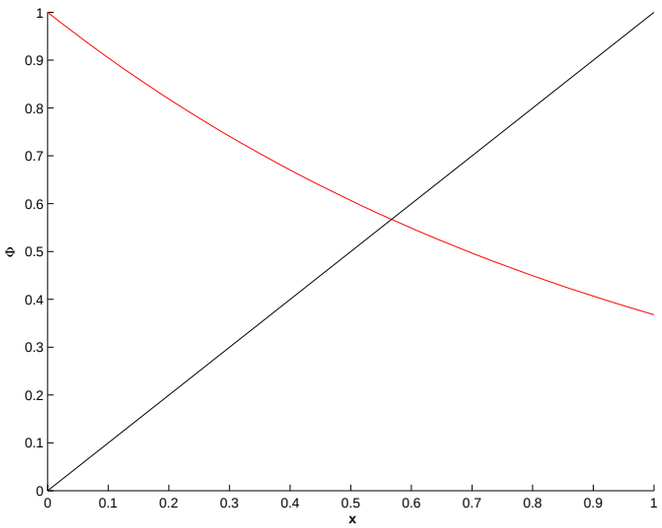
k	$\lvert x_1^{(k+1)} - x^* \rvert$	$\lvert x_2^{(k+1)} - x^* \rvert$	$\lvert x_3^{(k+1)} - x^* \rvert$
0	0.067143290409784	0.067143290409784	0.067143290409784
1	0.039387369302849	0.000832287212566	0.108496074240152
2	0.021904078517179	0.000000125374922	0.219330611898582
3	0.012559804468284	0.000000000000003	0.288178118764323
4	0.007078662470882	0.000000000000000	0.723649245792953
5	0.004028858567431	0.000000000000000	0.410183132337935
6	0.002280343429460	0.000000000000000	1.186907542305364
7	0.001294757160282	0.000000000000000	0.146569797006362
8	0.000733837662863	0.000000000000000	0.310516641279937
9	0.000416343852458	0.000000000000000	0.357777386500765
10	0.000236077474313	0.000000000000000	0.974565695952037

Observed: linear convergence of $x_1^{(k)}$, quadratic convergence of $x_2^{(k)}$, no convergence (erratic behavior of $x_3^{(k)}$), $x_i^{(0)} = 0.5$. $\diamond$

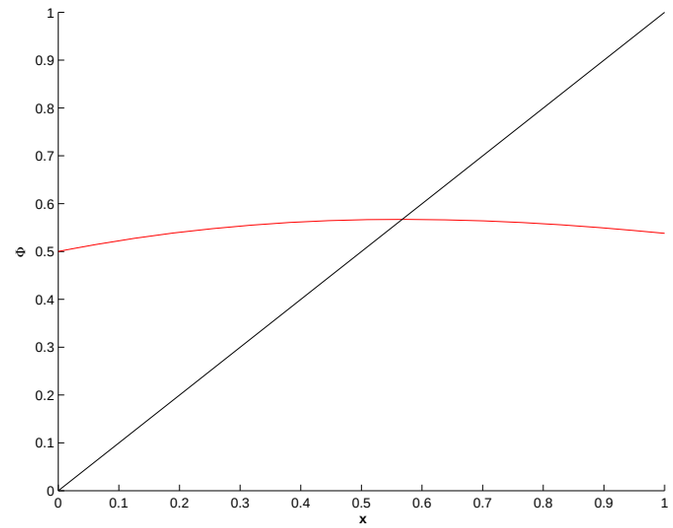
Question: can we explain/forecast the behaviour of the iteration?

4.2.2 Convergence of fixed point iterations

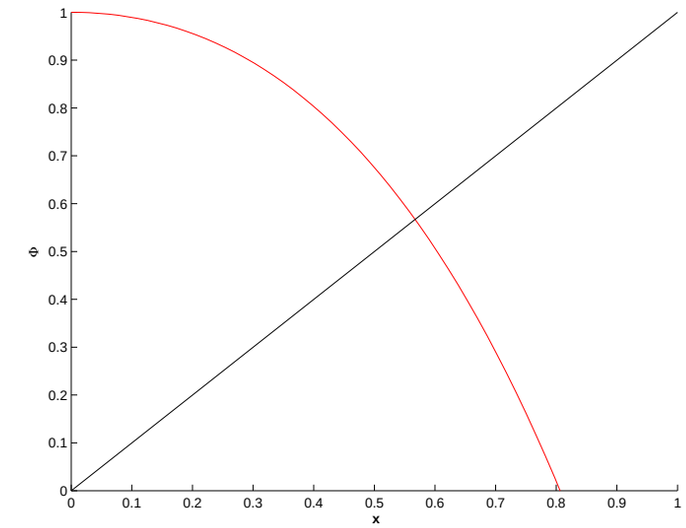
Ex. 4.2.3 revisited: vastly different behavior of different fixed point iterations for $n = 1$:



Φ_1: linear convergence ?



Φ_2: quadratic convergence ?



function Φ_3: no convergence

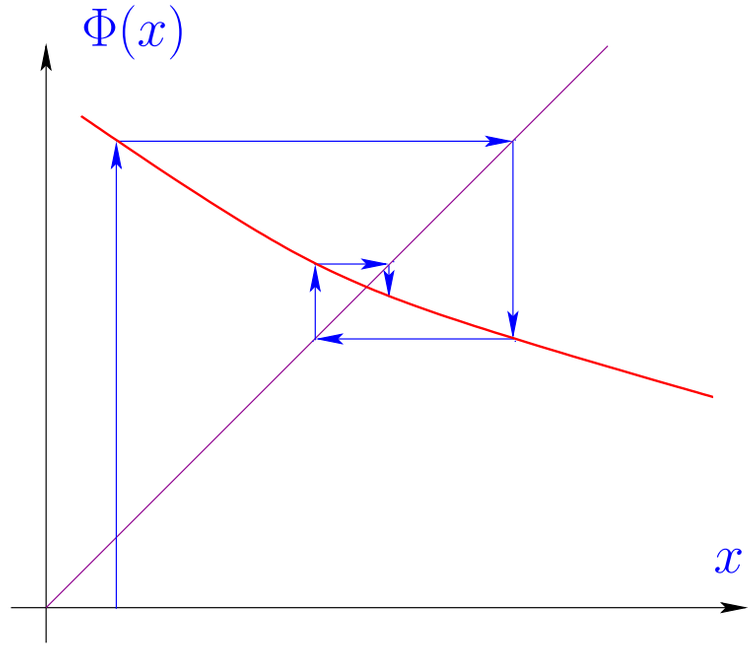
Example 4.2.4 (Fixed point iteration in 1D).

1D setting ($n = 1$): $\Phi : \mathbb{R} \mapsto \mathbb{R}$ continuously differentiable, $\Phi(x^*) = x^*$

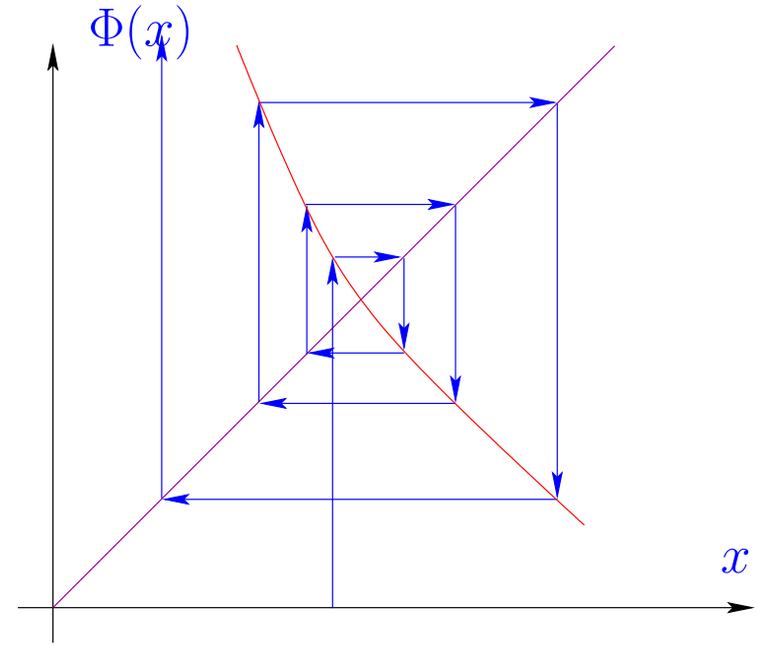
fixed point iteration: $x^{(k+1)} = \Phi(x^{(k)})$

R. Hiptmair

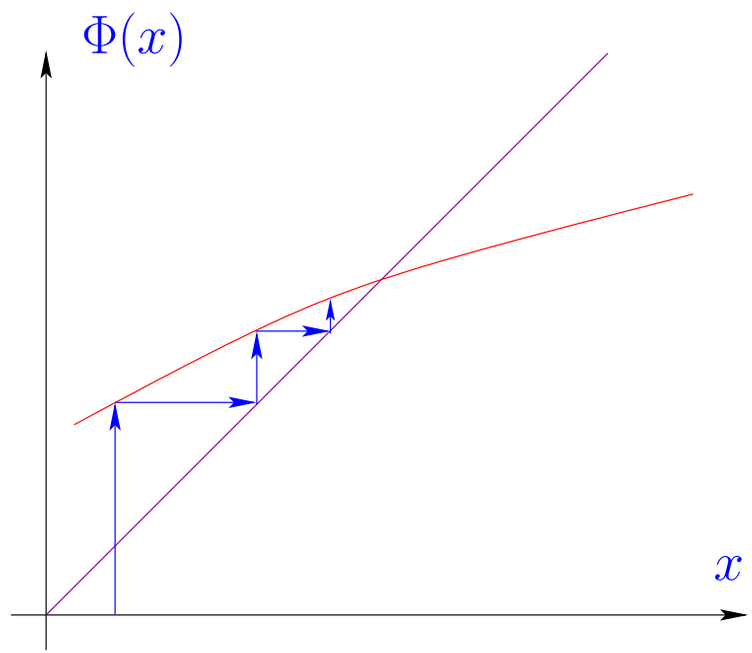
rev 38355, October 3, 2011



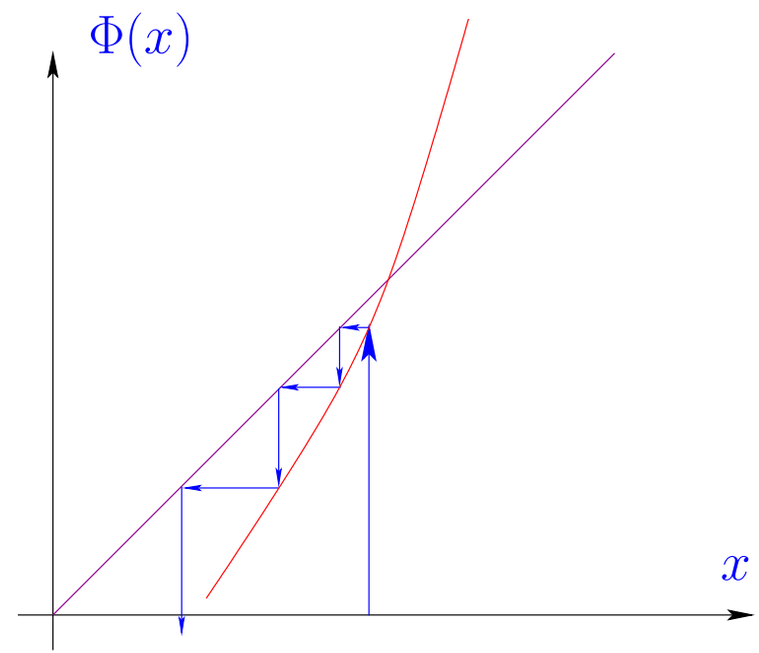
$-1 < \Phi'(x^*) \leq 0$ ➢ convergence



$\Phi'(x^*) < -1$ ➢ divergence



$0 \leq \Phi'(x^*) < 1$ ➢ convergence



$1 < \Phi'(x^*)$ ➢ divergence

Lemma 4.2.8 (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 !

✎ notation: $D\Phi(\mathbf{x}) \hat{=}$ Jacobian (*ger.:* Jacobi-Matrix) of Φ at $\mathbf{x} \in D$
$\rightarrow$ [63, Sect. 7.6]

Remark 4.2.10 (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 $L = \|D\Phi(x^*)\|$

△

Example 4.2.11 (Multidimensional fixed point iteration).

System of equations in fixed point form:

$$\begin{cases} x_1 - c(\cos x_1 - \sin x_2) &= 0 \\ (x_1 - x_2) - c\sin x_2 &= 0 \end{cases} \quad \Rightarrow \quad \begin{cases} c(\cos x_1 - \sin x_2) &= x_1 \\ c(\cos x_1 - 2\sin x_2) &= x_2 \end{cases} .$$

Define: $$\Phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = c\begin{pmatrix} \cos x_1 - \sin x_2 \\ \cos x_1 - 2\sin x_2 \end{pmatrix} \quad \Rightarrow \quad D\Phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = -c\begin{pmatrix} \sin x_1 & \cos x_2 \\ \sin x_1 & 2\cos x_2 \end{pmatrix} .$$

Choose *appropriate* norm: $\|\cdot\|$ = ∞-norm $\|\cdot\|_\infty$ ($\rightarrow$ Example 2.5.6) ;

$$\text{if } c < \frac{1}{3} \quad \Rightarrow \quad \|D\Phi(\mathbf{x})\|_\infty < 1 \quad \forall \mathbf{x} \in \mathbb{R}^2 ,$$

➢ (at least) linear convergence of the fixed point iteration.

◇

R. Hiptmair

rev 38355, October 3, 2011

What about higher order convergence ($\rightarrow$ Def. 4.1.14, *cf.* Φ_2 in Ex. 4.2.3) ?

Refined convergence analysis for $n = 1$ (scalar case, $\Phi : \mathrm{dom}(\Phi) \subset \mathbb{R} \mapsto \mathbb{R}$):

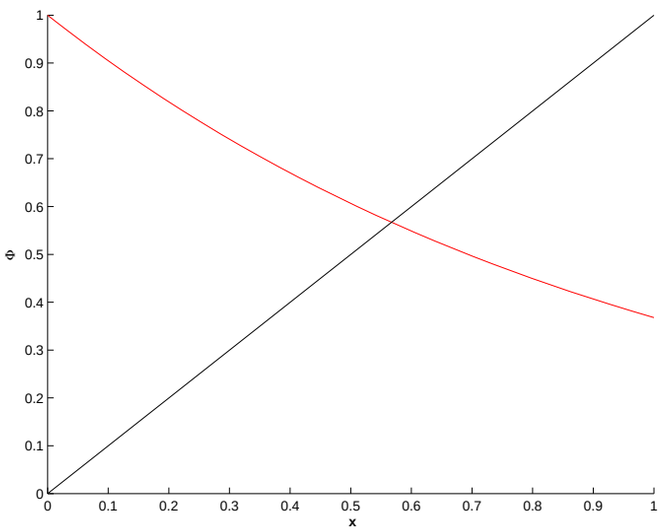
Theorem 4.2.12 (Taylor's formula). $\rightarrow$ *[63, Sect. 5.5]*

If $\Phi : U \subset \mathbb{R} \mapsto \mathbb{R}$, U *interval, is* $m+1$ *times continuously differentiable,* $x \in U$

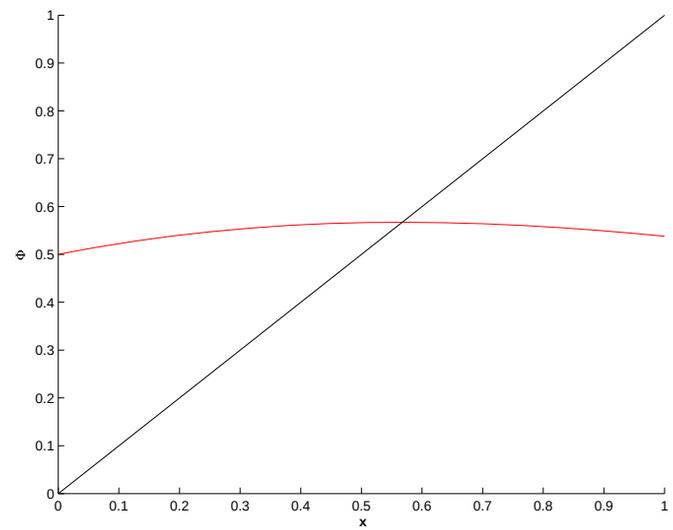
$$\Phi(y) - \Phi(x) = \sum_{k=1}^{m} \frac{1}{k!}\Phi^{(k)}(x)(y-x)^k + O(|y-x|^{m+1}) \quad \forall y \in U . \tag{4.2.13}$$

Lemma 4.2.15 (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).*

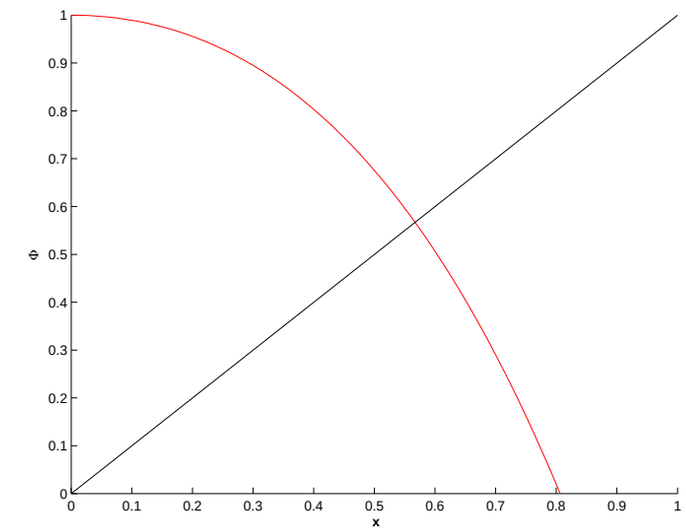
Example 4.2.3 continued:



function Φ_1



function Φ_2



function Φ_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 4.2.16 (Termination criterion for contractive fixed point iteration).

Recap of Rem. 4.1.23:

$$\left\| \mathbf{x}^* - \mathbf{x}^{(k)} \right\| \leq \frac{L^{k-l}}{1-L} \left\| \mathbf{x}^{(l+1)} - \mathbf{x}^{(l)} \right\| \ . \tag{4.2.17}$$

Set $l = 0$ in (4.2.18)	Set $l = k-1$ in (4.2.18)
a priori termination criterion	a posteriori termination criterion
$\left\| \mathbf{x}^* - \mathbf{x}^{(k)} \right\| \leq \frac{L^k}{1-L} \left\| \mathbf{x}^{(1)} - \mathbf{x}^{(0)} \right\|$ (4.2.18)	$\left\| \mathbf{x}^* - \mathbf{x}^{(k)} \right\| \leq \frac{L}{1-L} \left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \right\|$ (4.2.19)

△

4.3 Zero Finding

Now, focus on scalar case $n = 1$: $F : I \subset \mathbb{R} \mapsto \mathbb{R}$ **continuous**, I interval

Sought: $x^* \in I$: $F(x^*) = 0$

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 !)

$\Rightarrow$ $\exists\, x^* \in]\min\{a,b\}, \max\{a,b\}[$:
$F(x^*) = 0$.

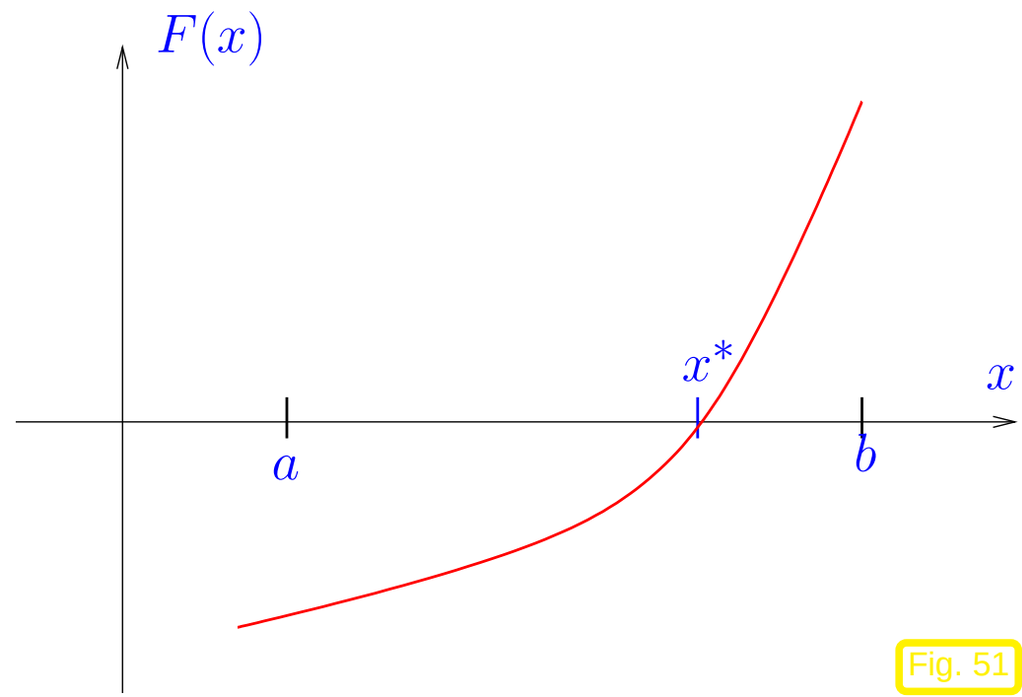


Fig. 51

R. Hiptmair

rev 38355, October 20, 2011

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
```

`tol` $\hat{=}$ absolute tolerance

Handle to MATLAB function providing F.

Avoid infinite loop, if `tol` $<$ resolution of $\mathbb{M}$ at zero x^* ("$\mathbb{M}$-based termination criterion")).

Advantages:

- "foolproof"
- requires only F evaluations

Drawbacks:

Merely "linear-type"onvergence: $|x^{(k)} - x^*| \leq 2^{-k}|b-a|$

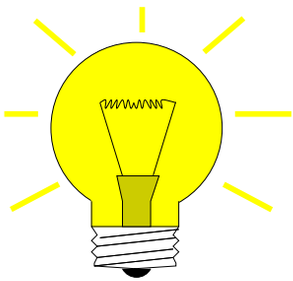
$\blacktriangleright$ $\log_2\left(\frac{|b-a|}{\texttt{tol}}\right)$ steps necessary

R. Hiptmair

rev 38355, October 20, 2011

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 $\widetilde{F}$
(using function values/derivative values in $x^{(k)}, x^{(k-1)}, \dots, x^{(k-m)}$)

❷ $x^{(k+1)} :=$ zero of $\widetilde{F}$
(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 $x^{(k)}$:

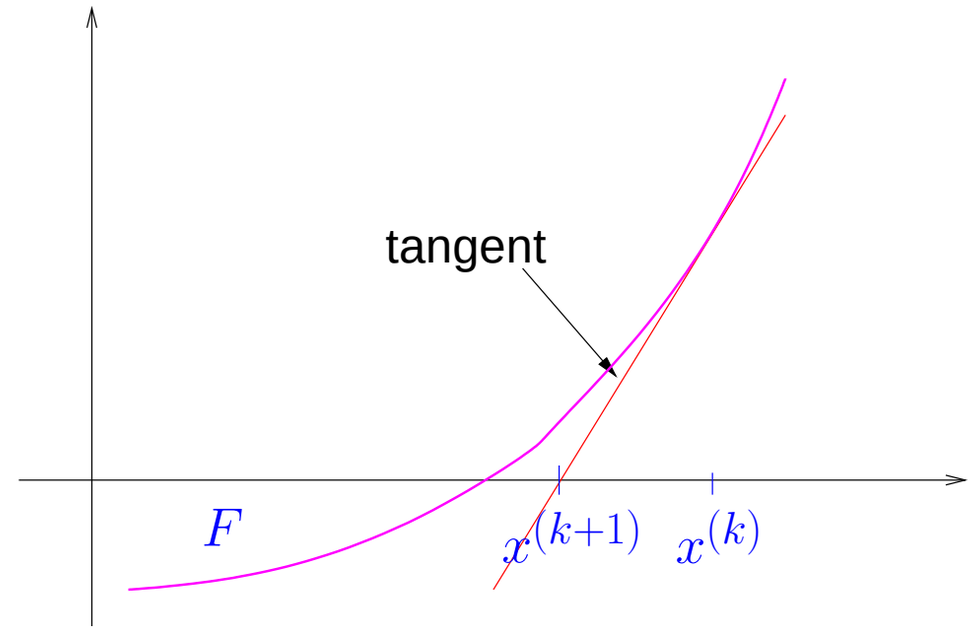
$$\widetilde{F}(x) := F(x^{(k)}) + F'(x^{(k)})(x - x^{(k)})$$

take $x^{(k+1)}$ **:=** zero of tangent

We obtain Newton iteration

$$\boxed{x^{(k+1)} := x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})}} \quad , \qquad (4.3.3)$$

that requires $F'(x^{(k)}) \neq 0$.



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

Example 4.3.5 (Halley's iteration). $\rightarrow$ [35, Sect. 18.3]

Given $x^{(k)} \in I$, next iterate **:=** zero of model function: $h(x^{(k+1)}) = 0$, where

$$h(x) := \frac{a}{x+b} + c \quad \text{(rational function) such that } F^{(j)}(x^{(k)}) = h^{(j)}(x^{(k)}) \ , \quad j = 0, 1, 2 \ .$$

$$\frac{a}{x^{(k)}+b}+c=F(x^{(k)})\;,\;-\frac{a}{(x^{(k)}+b)^2}=F'(x^{(k)})\;,\;\frac{2a}{(x^{(k)}+b)^3}=F''(x^{(k)})\;.$$

$$x^{(k+1)}=x^{(k)}-\frac{F(x^{(k)})}{F'(x^{(k)})}\cdot\frac{1}{1-\frac{1}{2}\frac{F(x^{(k)})F''(x^{(k)})}{F'(x^{(k)})^2}}\;.$$

Halley's iteration for $F(x)=\frac{1}{(x+1)^2}+\frac{1}{(x+0.1)^2}-1\;,\quad x>0:$ 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 (→ Sect. 4.3.3) ◇

Example 4.3.6 (Adapted Newton method).

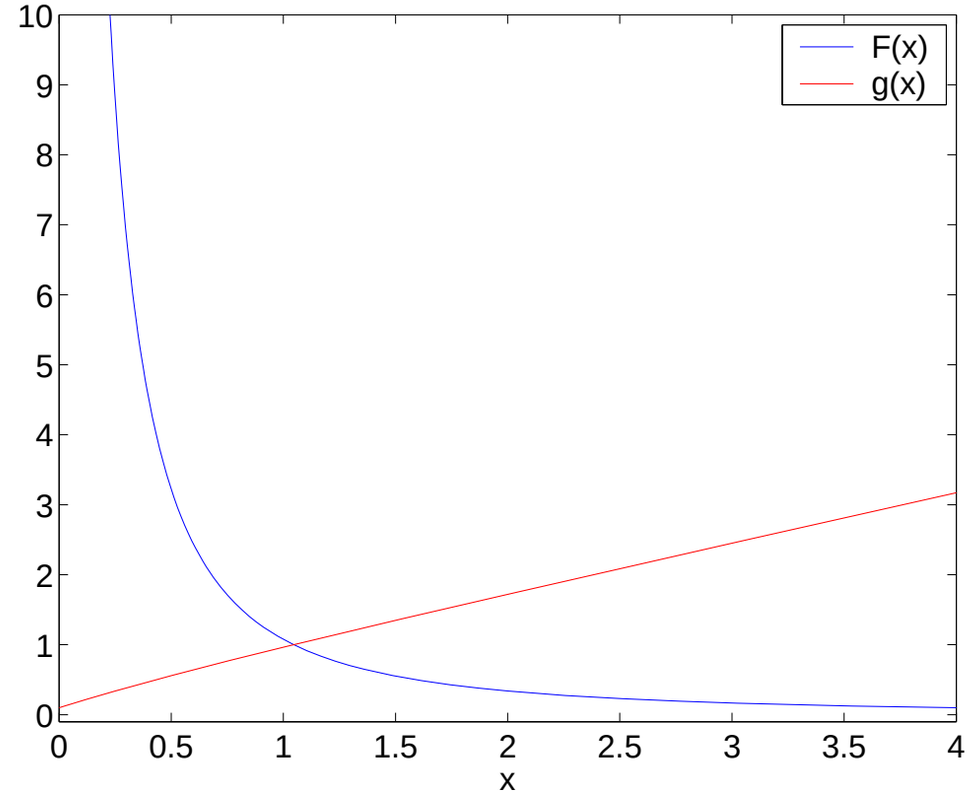
As in Ex. 4.3.5:

$$F(x) = \frac{1}{(x+1)^2} + \frac{1}{(x+0.1)^2} - 1 \,, \quad x > 0 :$$

Observation:

$$F(x) + 1 \approx 2x^{-2} \text{ for } x \gg 1$$

and so $g(x) := \dfrac{1}{\sqrt{F(x)+1}}$ "almost" linear for $x \gg 1$



Idea: instead of $F(x) \overset{!}{=} 0$ tackle $g(x) \overset{!}{=} 1$ with Newton's method (4.3.3).

$$x^{(k+1)} = x^{(k)} - \frac{g(x^{(k)}) - 1}{g'(x^{(k)})} = x^{(k)} + \left(\frac{1}{\sqrt{F(x^{(k)}) + 1}} - 1 \right) \frac{2(F(x^{(k)}) + 1)^{3/2}}{F'(x^{(k)})}$$

$$= x^{(k)} + \frac{2(F(x^{(k)}) + 1)(1 - \sqrt{F(x^{(k)}) + 1})}{F'(x^{(k)})} .$$

Convergence recorded for $x^{(0)} = 0$:

R. Hiptmair

rev 38355, October 20, 2011

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

◇

4.3.2.3 Multi-point methods

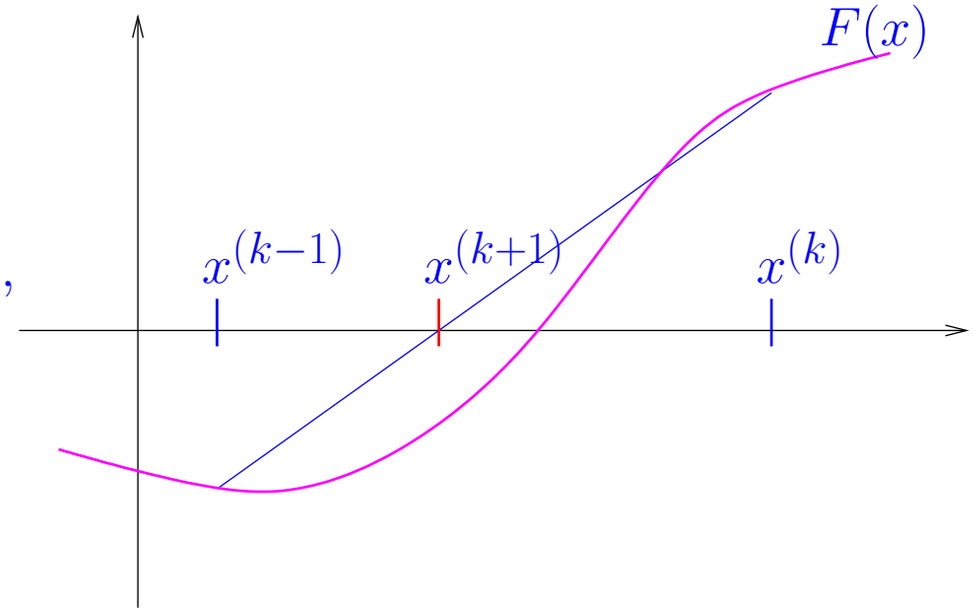
Idea: Replace F with interpolating polynomial

producing interpolatory model function methods

Simplest representative: secant method $\rightarrow$ [35, Sect. 18.2], [13, Sect, 5.5.3]

$x^{(k+1)}$ = zero of secant

$$s(x) = F(x^{(k)}) + \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}(x - x^{(k)}) \,, \tag{4.3.11}$$



$$\blacktriangleright \quad x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})(x^{(k)} - x^{(k-1)})}{F(x^{(k)}) - F(x^{(k-1)})} \,. \tag{4.3.12}$$

secant method

(MATLAB implementation)

- Only one function evaluation per step
- no derivatives required !

Code 4.3.13: secant method

```
function x = secant(x0,x1,F,tol)
fo = F(x0);
for i=1:MAXIT
  fn = F(x1);
  s = fn*(x1-x0)/(fn-fo);
  x0 = x1; x1 = x1-s;
  if (abs(s) < tol), x = x1;
    return; end
  fo = fn;
end
```

Example 4.3.14 (secant method). $F(x) = xe^x - 1\,, \quad x^{(0)} = 0\,, x^{(1)} = 5\,.$

k	$x^{(k)}$	$F(x^{(k)})$	$e^{(k)} := x^{(k)} - x^*$	$\frac{\log\lvert e^{(k+1)}\rvert - \log\lvert e^{(k)}\rvert}{\log\lvert e^{(k)}\rvert - \log\lvert e^{(k-1)}\rvert}$
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	-0.00000000000001	-0.00000000000000	

◇

R. Hiptmair

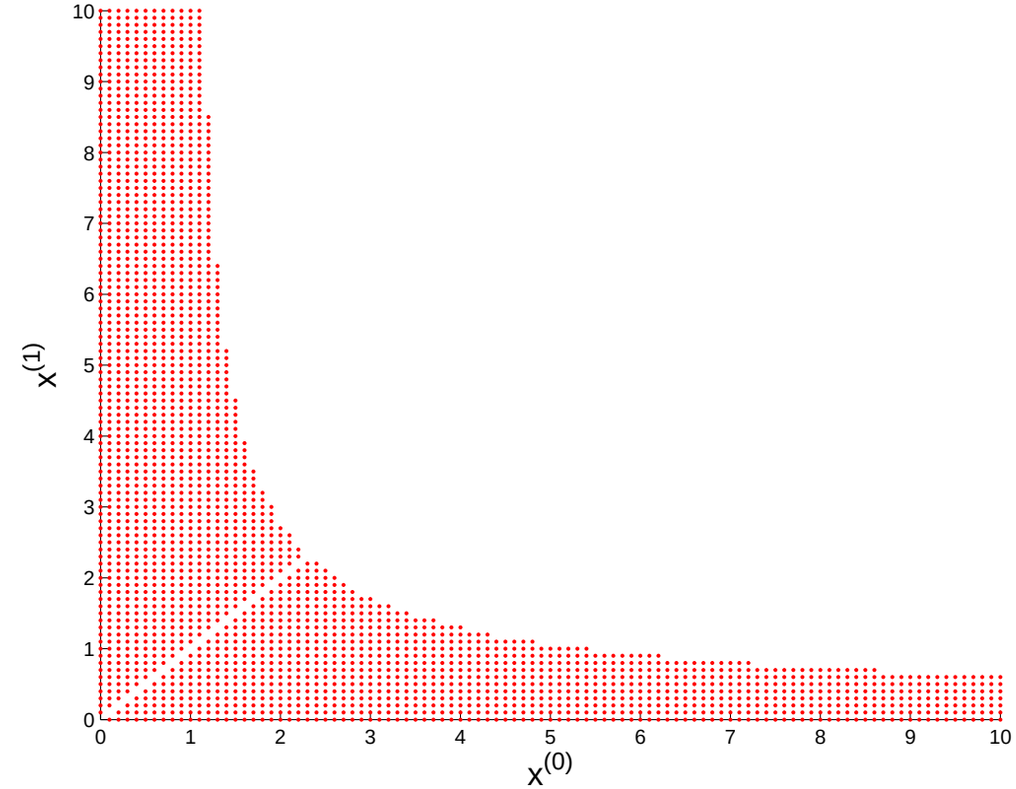
rev 38355, October 20, 2011

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: $F : I \subset \mathbb{R} \mapsto \mathbb{R}$ one-to-one

$$F(x^*) = 0 \quad \Rightarrow \quad F^{-1}(0) = x^* \ .$$

- 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 $x^{(k+1)} := p(0)$

$$F(x^*) = 0 \quad \Leftrightarrow \quad F^{-1}(0) = x^*$$

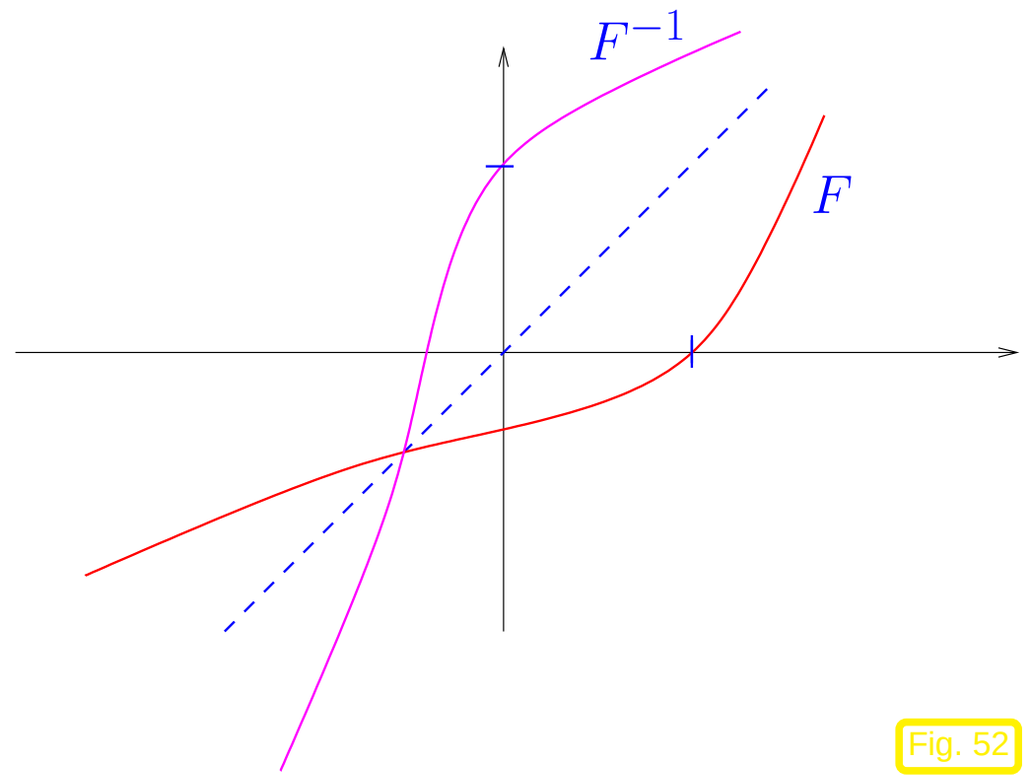


Fig. 52

Case $d = 1$ ➢ secant method

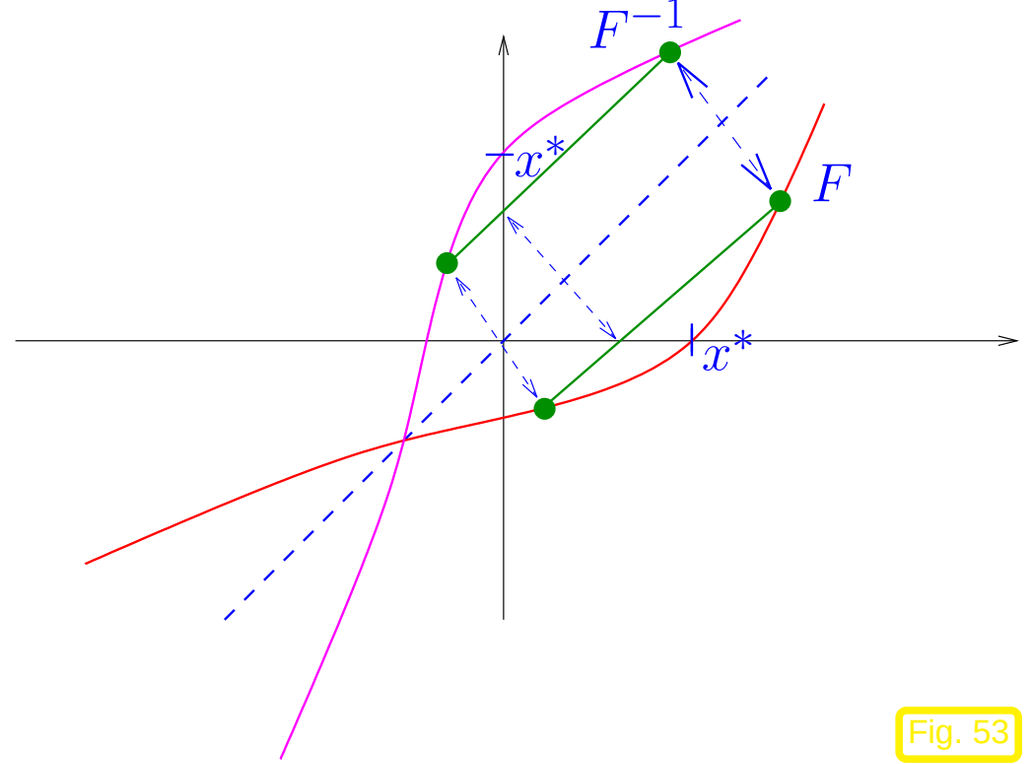


Fig. 53

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);
```

$$\blacktriangleright \quad x^{(k+1)} = \frac{F_0^2(F_1x_2 - F_2x_1) + F_1^2(F_2x_0 - F_0x_2) + F_2^2(F_0x_1 - F_1x_0)}{F_0^2(F_1 - F_2) + F_1^2(F_2 - F_0) + F_2^2(F_0 - F_1)} \,.$$

$$\left(F_0 := F(x^{(k-2)}), F_1 := F(x^{(k-1)}), F_2 := F(x^{(k)}), x_0 := x^{(k-2)}, x_1 := x^{(k-1)}, x_2 := x^{(k)} \right)$$

Example 4.3.19 (quadratic inverse interpolation). $F(x) = xe^x - 1$, $x^{(0)} = 0$, $x^{(1)} = 2.5$,$x^{(2)} = 5$.

k	$x^{(k)}$	$F(x^{(k)})$	$e^{(k)} := x^{(k)} - x^*$	$\frac{\log\lvert e^{(k+1)}\rvert - \log\lvert e^{(k)}\rvert}{\log\lvert e^{(k)}\rvert - \log\lvert e^{(k-1)}\rvert}$
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	0.00000000000003	0.00000000000001	

◇

4.3.3 Note on Efficiency

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

$$\text{(e.g,} \quad W \approx \frac{\#\{\text{evaluations of } F\}}{\text{step}} + n \cdot \frac{\#\{\text{evaluations of } F'\}}{\text{step}} + \cdots \text{)}$$

Crucial: number of steps $k = k(\rho)$ to achieve *relative reduction of error*

$$\left\| e^{(k)} \right\| \leq \rho \left\| e^{(0)} \right\| \ , \quad \rho > 0 \text{ prescribed ?} \tag{4.3.20}$$

Notice: $|\log \rho| \quad \leftrightarrow \quad$ No. of significant digits of $x^{(k)}$

Measure for efficiency:

$$\boxed{\text{Efficiency} := \frac{\text{no. of digits gained}}{\text{total work required}} = \frac{|\log \rho|}{k(\rho) \cdot W}} \tag{4.3.23}$$

asymptotic efficiency w.r.t. $\rho \to 0$ ➜ $|\log \rho| \to \infty$):

$$\text{Efficiency}_{|\rho \to 0} = \begin{cases} -\dfrac{\log C}{W} & \text{, if } \quad p = 1 \ , \\ \dfrac{\log p \cdot |\log \rho|}{W \log(|\log \rho|)} & \text{, if } \quad p > 1 \ . \end{cases} \tag{4.3.24}$$

R. Hiptmair

rev 38355, October 20, 2011

Example 4.3.25 (Efficiency of iterative methods).

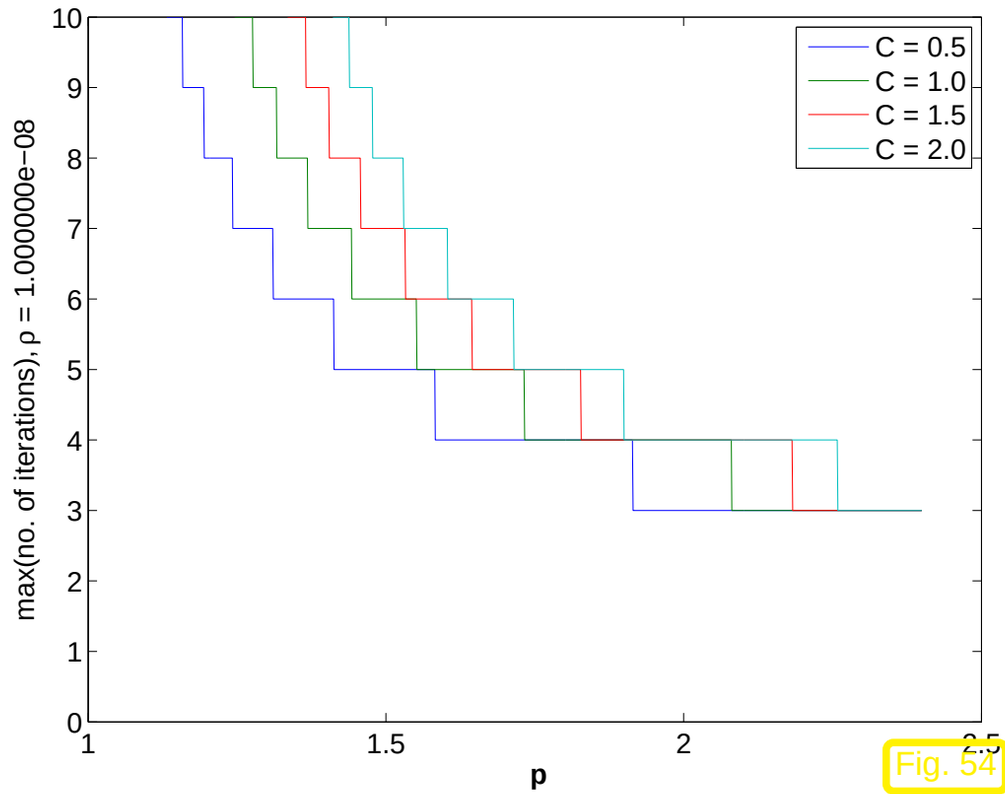


Fig. 54

Evaluation (4.3.22) for $\left\|e^{(0)}\right\| = 0.1$, $\rho = 10^{-8}$

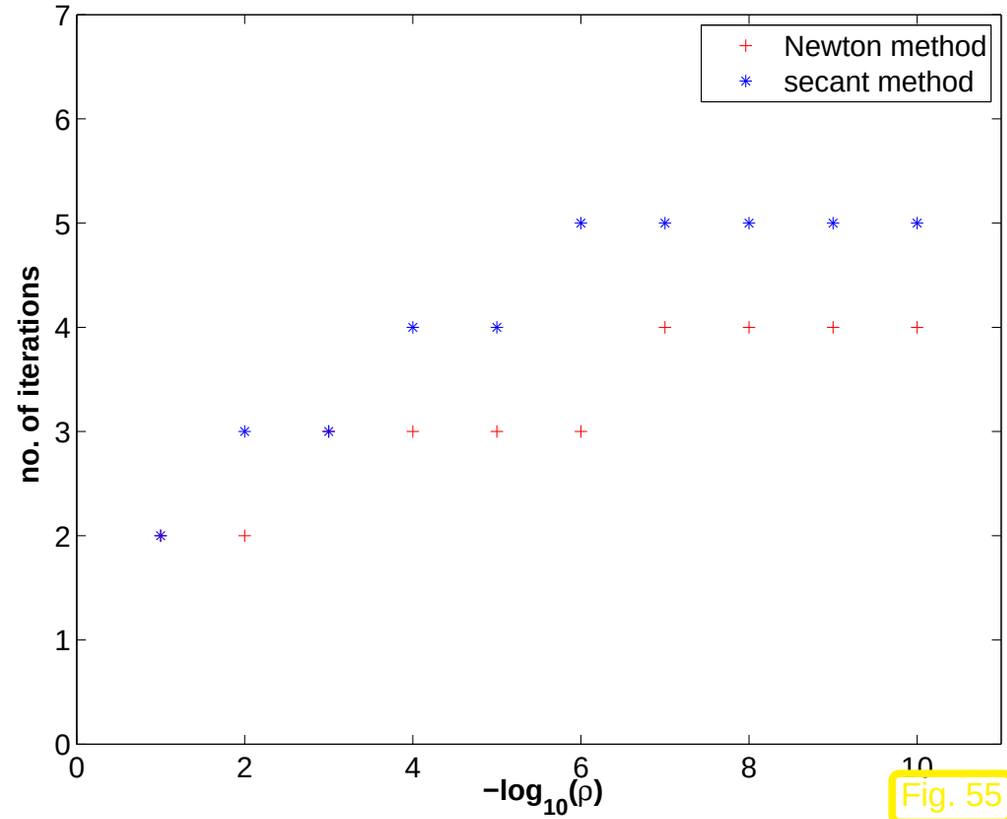


Fig. 55

Newton's method $\leftrightarrow$ secant method, $C = 1$, initial error $\left\|e^{(0)}\right\| = 0.1$

$$W_{\text{Newton}} = 2W_{\text{secant}} \ , \quad p_{\text{Newton}} = 2,\ p_{\text{secant}} = 1.62 \quad \Rightarrow \quad \frac{\log p_{\text{Newton}}}{W_{\text{Newton}}} : \frac{\log p_{\text{secant}}}{W_{\text{secant}}} = 0.71 \ .$$

➢ 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 $\mathbf{x}^{(k)} \in D$ ➢ $\mathbf{x}^{(k+1)}$ as zero of affine linear model function

$$F(\mathbf{x}) \approx \widetilde{F}(\mathbf{x}) := F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) \ ,$$

$DF(\mathbf{x}) \in \mathbb{R}^{n,n}$ = 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 $n = 1$)

$$\boxed{\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})} \quad , \quad [\text{ if } DF(\mathbf{x}^{(k)}) \text{ regular }] \qquad (4.4.1)$$

Terminology: $-DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$ = Newton correction

MATLAB template for Newton method:

Solve linear system:

A\b = $\mathbf{A}^{-1}b$ $\rightarrow$ 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
```

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 $\mathbf{A}$!

Remark 4.4.6 (Simplified Newton method). [13, Sect. 5.6.2]

Simplified Newton Method:

use the same $DF(\mathbf{x}^{(k)})$ for all/several steps

Code 4.4.7: simplified Newton method

```
function x = simpnewton(x,F,DF,rtol,atol)
% MATLAB template for simplified Newton method
[L,U] = lu(DF(x)); % one LU-decomposition
s = U\(L\F(x)); x = x-s;
% termination based on relative and absolute tolerance
ns = norm(s); nx = norm(x);
while ((ns > rtol*nx) && (ns > atol))
  s = U\(L\F(x)); x = x-s;
end
```

➢ (usually) merely linear convergence instead of quadratic convergence

△

R. Hiptmair

rev 38355, October 26, 2011

4.4.2 Convergence of Newton's method

Example 4.4.9 (Convergence of Newton's method in 2D).

$$F(\mathbf{x}) = \begin{pmatrix} x_1^2 - x_2^4 \\ x_1 - x_2^3 \end{pmatrix} \;, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2 \quad \text{with solution} \quad F\begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0 \;.$$

k	$\mathbf{x}^{(k)}$	$\epsilon_k := \|\mathbf{x}^* - \mathbf{x}^{(k)}\|_2$	$\dfrac{\log \epsilon_{k+1} - \log \epsilon_k}{\log \epsilon_k - \log \epsilon_{k-1}}$
0	$(0.7,\ 0.7)^T$	4.24e-01	
1	$(0.87850000000000,\ 1.06428571428571 4)^T$	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	$(0.999999999999998,\ 1.000000000000000)^T$	2.11e-15	
6	$(1,\ 1)^T$		

$\diamond$

4.4.3 Termination of Newton iteration

Practical a-posteriori termination criterion for Newton's method:

$$\boxed{DF(\mathbf{x}^{(k-1)}) \approx DF(\mathbf{x}^{(k)}) \text{: quit as soon as } \left\| DF(\mathbf{x}^{(k-1)})^{-1} F(\mathbf{x}^{(k)}) \right\| < \tau_{\mathrm{rel}} \left\| \mathbf{x}^{(k)} \right\|}$$

affine invariant termination criterion

Terminology: $\Delta\bar{\mathbf{x}}^{(k)} := DF(\mathbf{x}^{(k-1)})^{-1} F(\mathbf{x}^{(k)}) \hat{=}$ 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

4.4.4 Damped Newton method [13, pp. 200]

Example 4.4.14 (Local convergence of Newton's method).

$$F(x) = xe^x - 1 \quad \Rightarrow \quad F'(-1) = 0$$

$$x^{(0)} < -1 \;\Rightarrow\; x^{(k)} \to -\infty$$
$$x^{(0)} > -1 \;\Rightarrow\; x^{(k)} \to x^*$$

◇

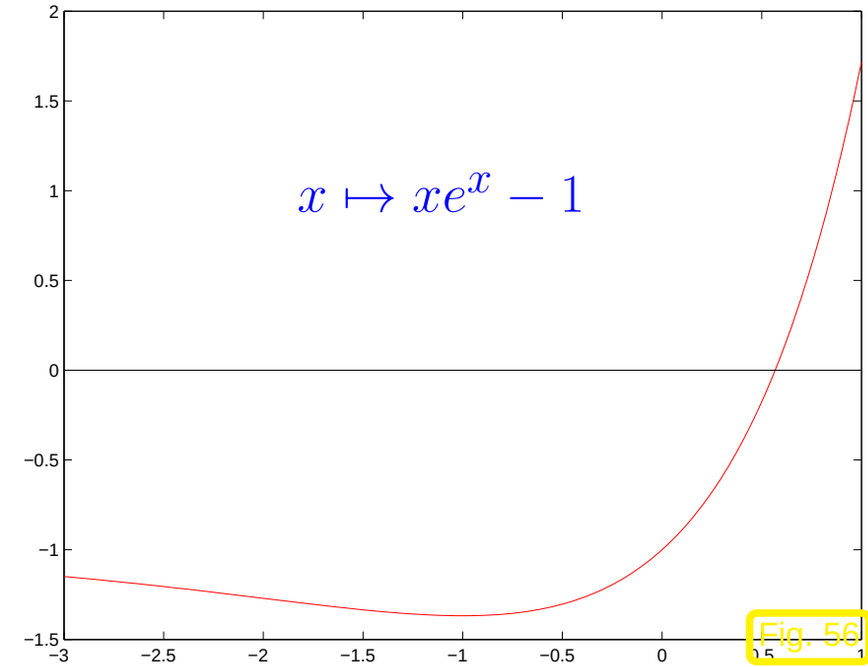


Fig. 56

Example 4.4.15 (Region of convergence of Newton method).

$$F(x) = \arctan(ax)\;, \quad a > 0, x \in \mathbb{R}$$

with zero $x^* = 0$.

◇

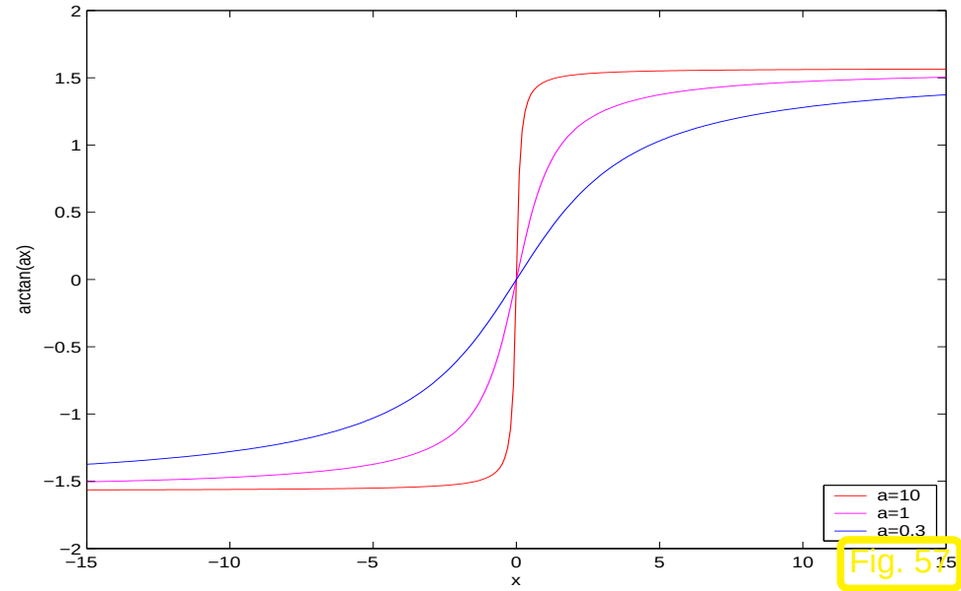
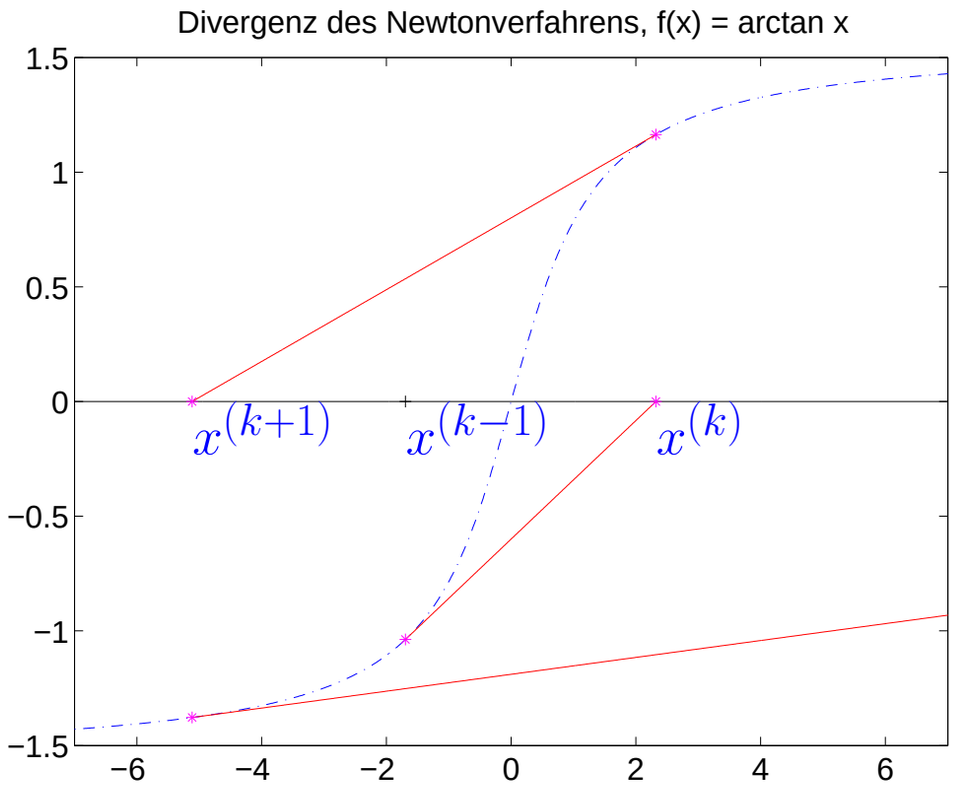
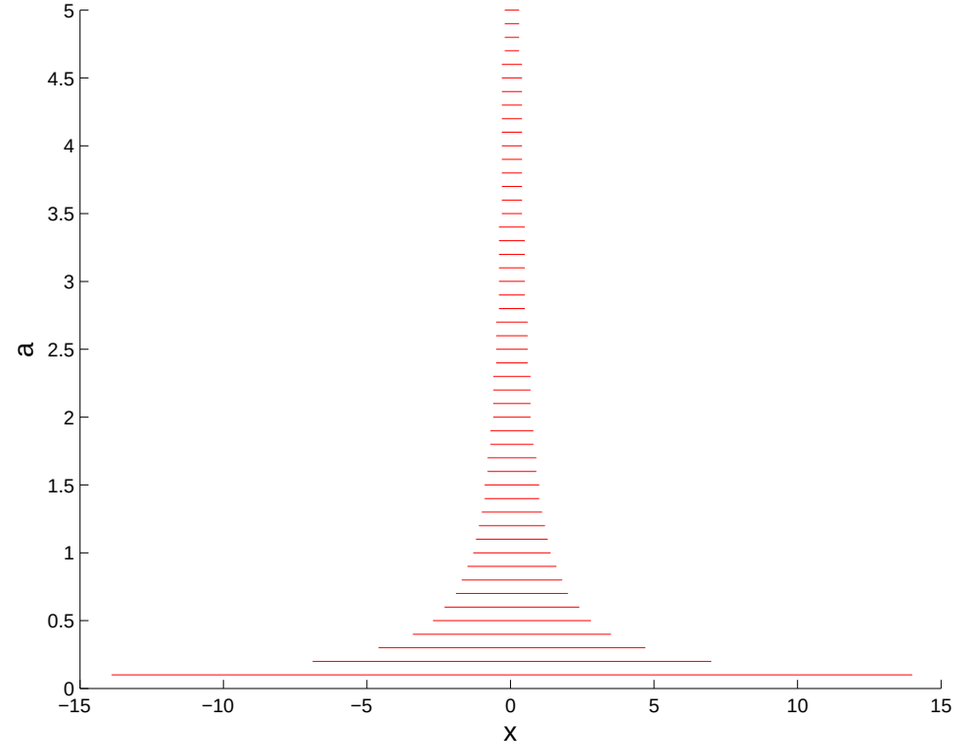


Fig. 57

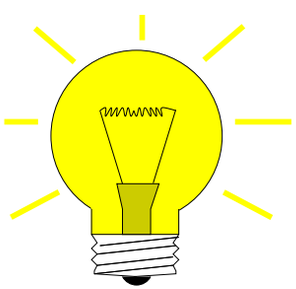




red zone = $\{x^{(0)} \in \mathbb{R}, x^{(k)} \to 0\}$

R. Hiptmair

rev 38355, October 26, 2011



- we observe “overshooting” of Newton correction

Idea: damping of Newton correction:

With $\lambda^{(k)} > 0$: $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \lambda^{(k)} DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$. (4.4.16)

Terminology: $\lambda^{(k)}$ = damping factor

Choice of damping factor: affine invariant natural monotonicity test [15, Ch. 3]

$$\text{“maximal”} \quad 0 < \lambda^{(k)} \leq 1: \qquad \left\|\Delta\overline{\mathbf{x}}(\lambda^{(k)})\right\| \leq (1 - \frac{\lambda^{(k)}}{2}) \left\|\Delta\mathbf{x}^{(k)}\right\|_2 \tag{4.4.17}$$

where

$$\Delta\mathbf{x}^{(k)} := DF(\mathbf{x}^{(k)})^{-1}F(\mathbf{x}^{(k)}) \qquad \rightarrow \text{current Newton correction ,}$$

$$\Delta\overline{\mathbf{x}}(\lambda^{(k)}) := DF(\mathbf{x}^{(k)})^{-1}F(\mathbf{x}^{(k)} + \lambda^{(k)}\Delta\mathbf{x}^{(k)}) \qquad \rightarrow \text{tentative simplified Newton correction .}$$

Code 4.4.18: Damped Newton method

```
function [x,cvg] =
  dampnewton(x,F,DF,rtol,atol)
[L,U] = lu(DF(x)); s = U\(L\F(x));
xn = x-s; lambda = 1; cvg = 0;
f = F(xn); st = U\(L\f); stn = norm(st);
while ((stn>rtol*norm(xn)) && (stn > atol))
  while (norm(st) > (1-lambda/2)*norm(s))
    lambda = lambda/2;
    if (lambda < LMIN), cvg = -1; return;
      end
    xn = x-lambda*s; f = F(xn);
    st =  U\(L\f);
  end
  x = xn; [L,U] = lu(DF(x)); s = U\(L\f);
  lambda = min(2*lambda,1);
  xn = x-lambda*s; f = F(xn); st =
    U\(L\f);
end
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 λ

Example 4.4.19 (Damped Newton method). (→ 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	-0.00000000000001	-0.00000000000001

◇

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 $x^{(0)} = -1.5$

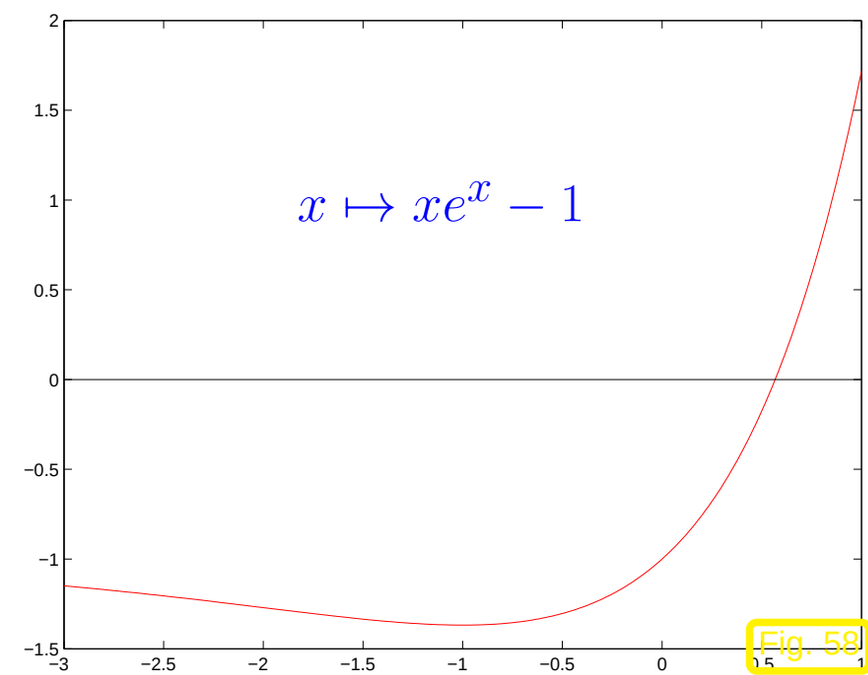


Fig. 58

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 < LMIN`!			

◇

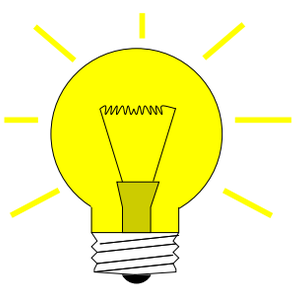
4.4.5 Quasi-Newton Method [51, Sect. 7.1.4]

R. Hiptmair

rev 38355, October 26, 2011

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



$$F'(x^{(k)}) \approx \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \quad \text{"difference quotient"} \qquad (4.4.21)$$

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:

$$\mathbf{J}_k(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}) = F(\mathbf{x}^{(k)}) - F(\mathbf{x}^{(k-1)}) \; . \tag{4.4.22}$$

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: $\mathbf{J}_k\mathbf{z} = \mathbf{J}_{k-1}\mathbf{z} \quad \forall \mathbf{z}\colon \mathbf{z} \perp (\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})$. (4.4.23)

i.e.:

$$\boxed{\mathbf{J}_k := \mathbf{J}_{k-1} + \frac{F(\mathbf{x}^{(k)})(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})^T}{\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\|_2^2}} \tag{4.4.24}$$

R. Hiptmair

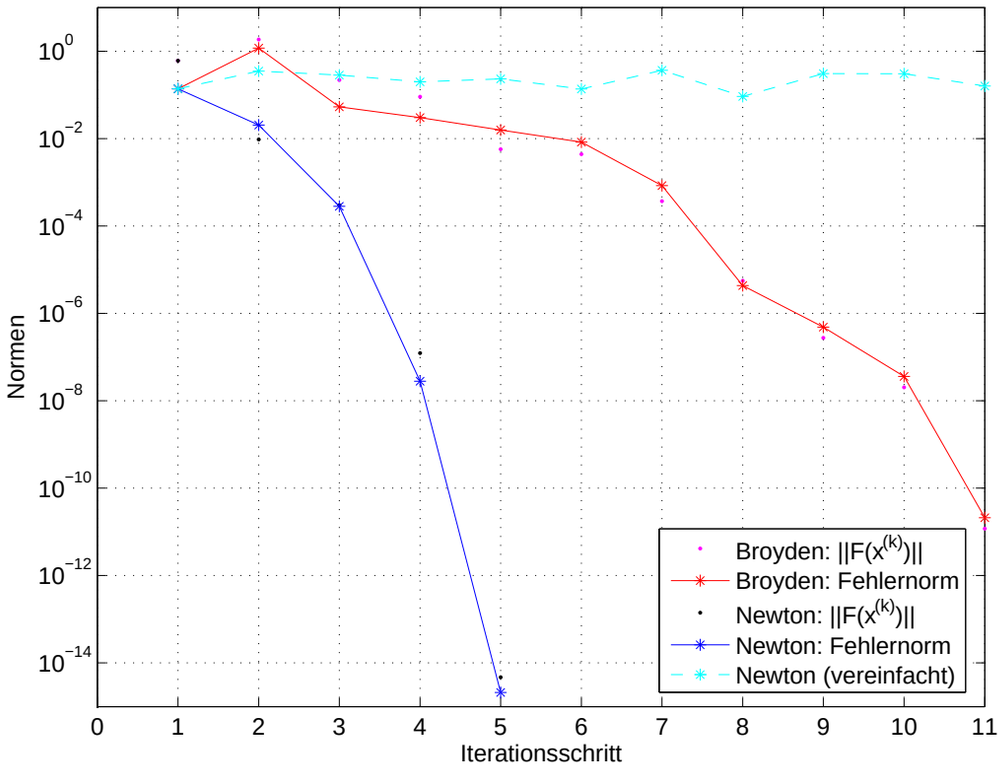
rev 38355, October 26, 2011

Broydens Quasi-Newton Method for solving $F(\mathbf{x}) = 0$:

$$\begin{aligned} &\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}, \; \Delta\mathbf{x}^{(k)} := -\mathbf{J}_k^{-1} F(\mathbf{x}^{(k)}) \; , \\ &\mathbf{J}_{k+1} := \mathbf{J}_k + \frac{F(\mathbf{x}^{(k+1)})(\Delta\mathbf{x}^{(k)})^T}{\left\|\Delta\mathbf{x}^{(k)}\right\|_2^2} \; . \end{aligned} \tag{4.4.25}$$

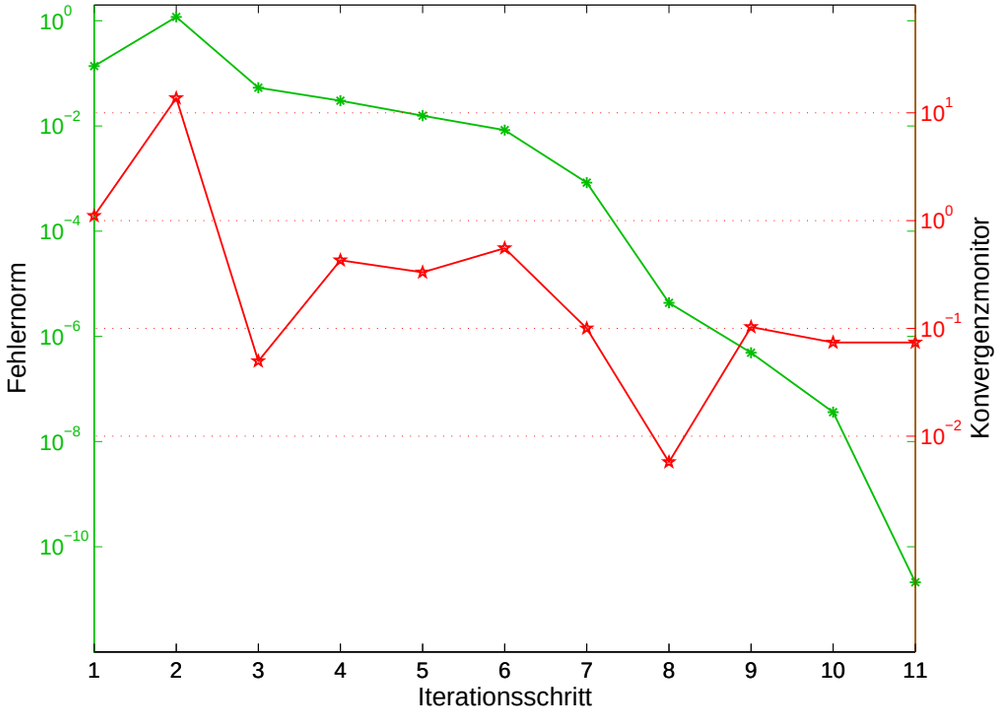
Example 4.4.27 (Broydens Quasi-Newton Method: Convergence).

- In the non-linear system of the example 4.4.9, $n = 2$ take $\mathbf{x}^{(0)} = (0.7.0.7)^T$ and $\mathbf{J}_0 = DF(\mathbf{x}^{(0)})$



R. Hiptmair

rev 38355, October 26, 2011



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 $\mu > 1$

◇

MATLAB-CODE: Broyden method (4.4.25)

```
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 $\Delta\mathbf{x}^{(k)}$, $\left\|\Delta\mathbf{x}^{(k)}\right\|_2^2$ (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

$$\frac{\left\|\mathbf{J}_{k-1}^{-1}F(\mathbf{x}^{(k)})\right\|}{\left\|\Delta\mathbf{x}^{(k-1)}\right\|} < 1 \text{ ?}$$

correction $\mathbf{s} = \mathbf{J}_k^{-1}F(\mathbf{x}^{(k)})$

Computational cost : N steps

- $O(N^2 \cdot n)$ operations with vectors, (Level I)
- 1 LU-decomposition of J, $N\times$ solutions of SLEs, see section 2.2
- N evalutations of F !

Memory cost : N steps

- LU-factors of J + auxiliary vectors $\in \mathbb{R}^n$
- N vectors $\mathbf{x}^{(k)} \in \mathbb{R}^n$

Example 4.4.30 (Broyden method for a large non-linear system).

$$F(\mathbf{x}) = \begin{cases} \mathbb{R}^n & \mapsto \mathbb{R}^n \\ \mathbf{x} & \mapsto \operatorname{diag}(\mathbf{x})\mathbf{A}\mathbf{x} - \mathbf{b} \end{cases},$$

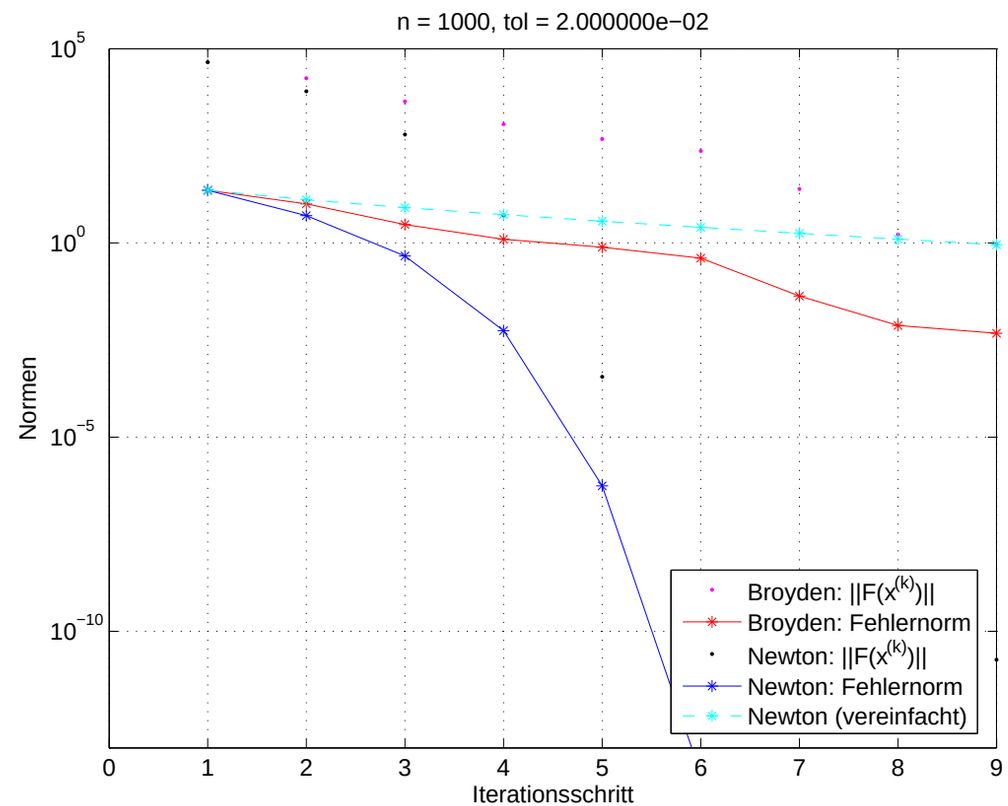
$$\mathbf{b} = (1, 2, \ldots, n) \in \mathbb{R}^n ,$$

$$\mathbf{A} = \mathbf{I} + \mathbf{a}\mathbf{a}^T \in \mathbb{R}^{n,n} ,$$

$$\mathbf{a} = \frac{1}{\sqrt{\mathbf{1} \cdot \mathbf{b} - \mathbf{1}}}(\mathbf{b} - \mathbf{1}) .$$

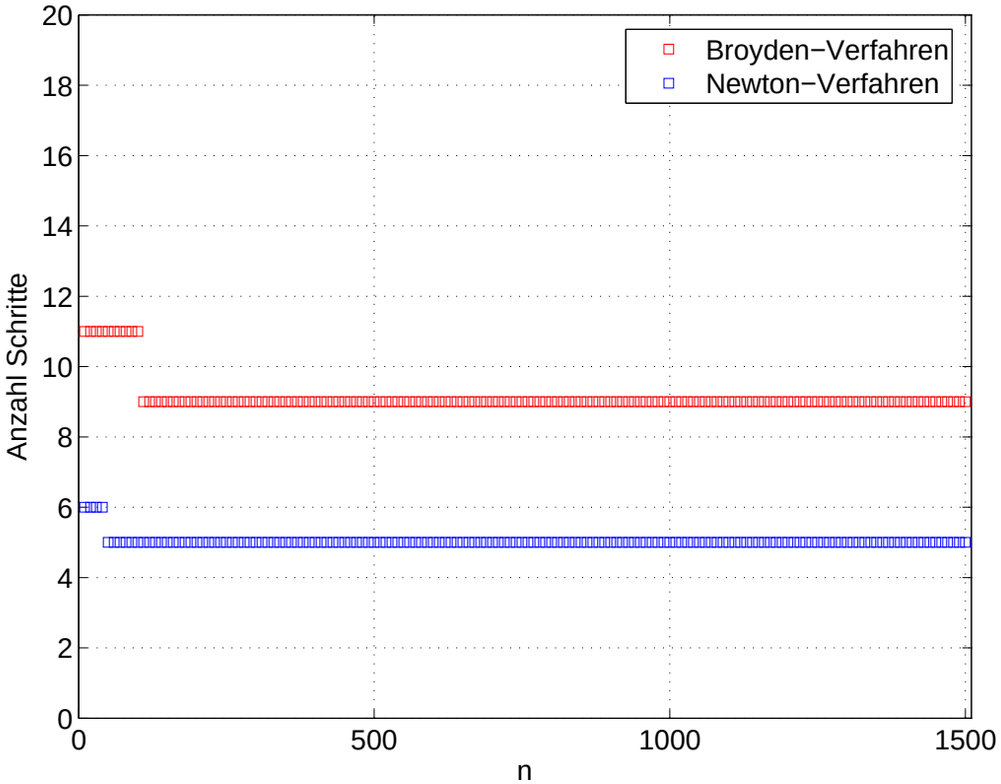
The interpretation of the results resemble the example 4.4.27 ▷

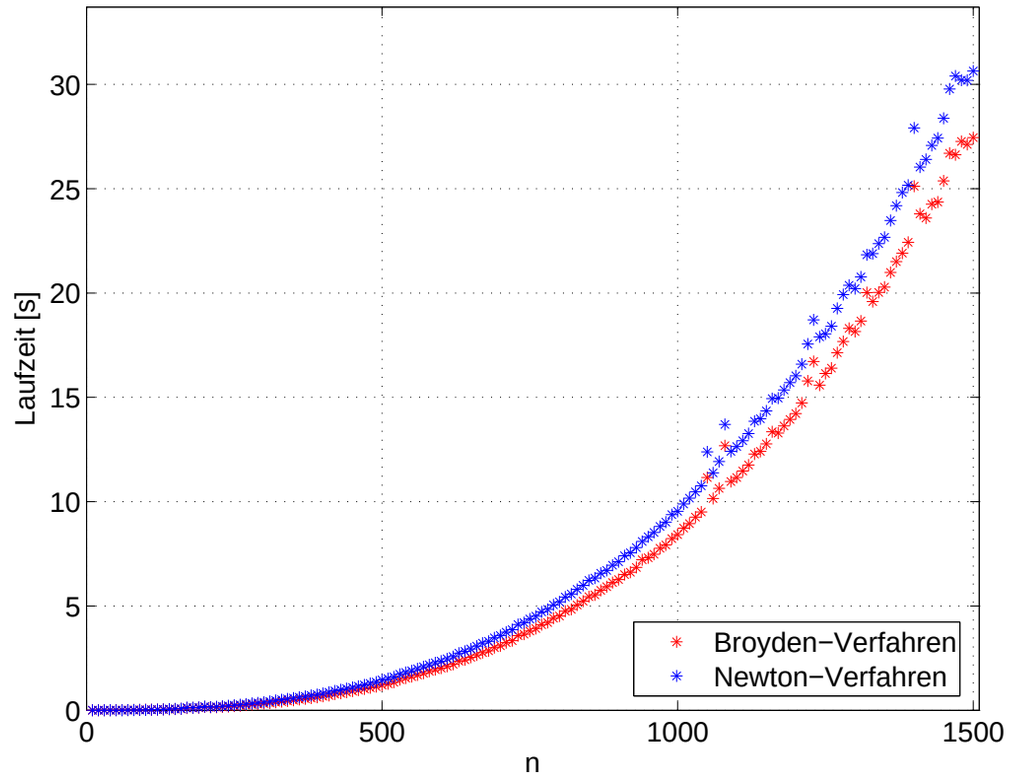
```
h = 2/n; x0 = (2:h:4-h)';
```



Efficiency comparison: Broyden method $\longleftrightarrow$ Newton method:

(in case of dimension n use tolerance `tol` $= 2n \cdot 10^{-5}$, `h = 2/n; x0 = (2:h:4-h)';`)





◇

5 Krylov Methods for Linear Systems of Equations

5.1 Descent Methods [51, Sect. 4.3.3]

Focus: Linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{n,n}$, $\mathbf{b} \in \mathbb{R}^n$, $n \in \mathbb{N}$ given,

with symmetric positive definite (s.p.d., $\rightarrow$ Def. 2.7.9) system matrix $\mathbf{A}$

➨ $\mathbf{A}$-inner product $(\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x}^\top \mathbf{A}\mathbf{y}$ $\Rightarrow$ "$\mathbf{A}$-geometry"

Definition 5.1.1 (Energy norm). $\rightarrow$ *[35, Def. 9.1]*

A s.p.d. matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ induces an energy norm

$$\|\mathbf{x}\|_A := (\mathbf{x}^\top \mathbf{A}\mathbf{x})^{1/2}, \quad \mathbf{x} \in \mathbb{R}^n .$$

R. Hiptmair

rev 38355, November 4, 2011

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:*

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad \mathbf{x} = \arg\min_{\mathbf{y}\in\mathbb{R}^n} J(\mathbf{y}) \,, \quad J(\mathbf{y}) := \tfrac{1}{2}\mathbf{y}^\top \mathbf{A}\mathbf{y} - \mathbf{b}^\top \mathbf{y} \,. \tag{5.1.4}$$

A quadratic functional

Example 5.1.6 (Quadratic functional in 2D)*.*

Plot of J from (5.1.4) for $\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

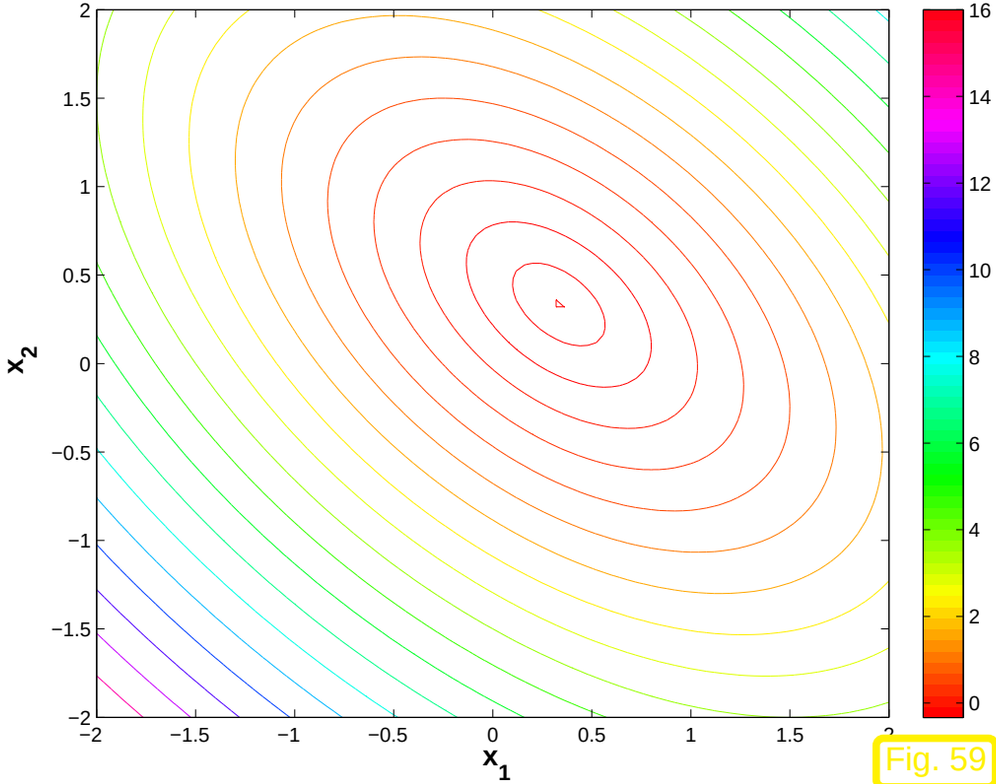


Fig. 59

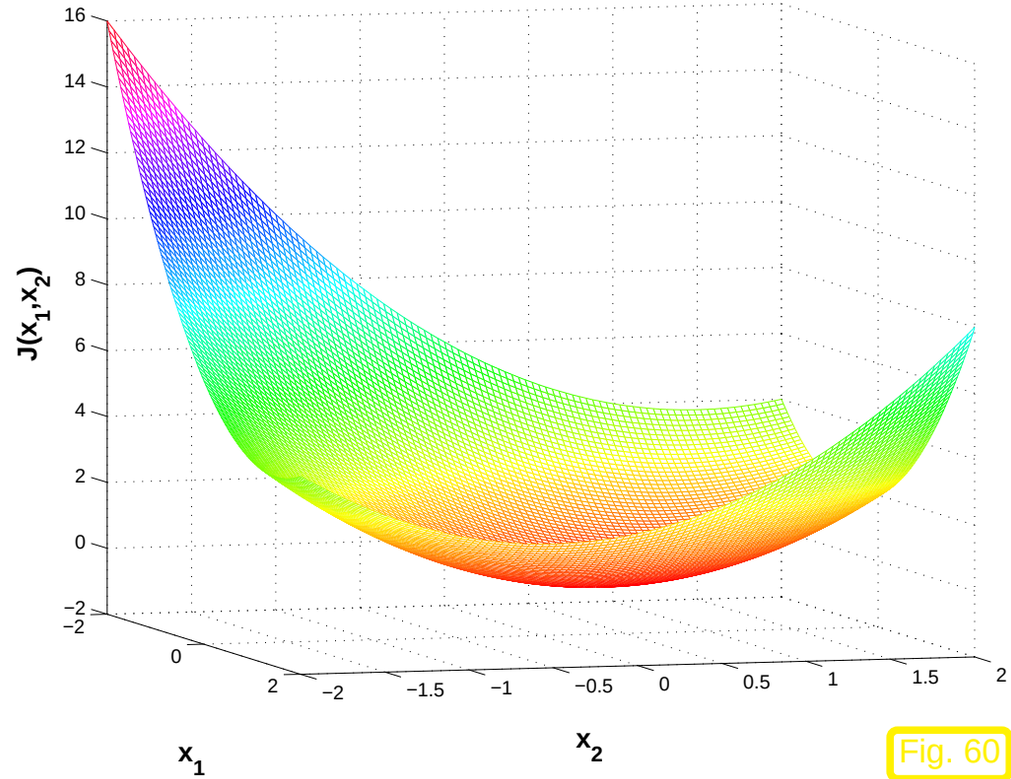
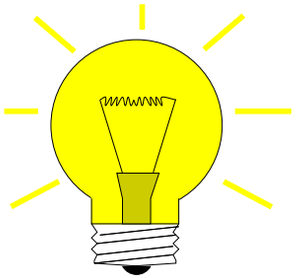


Fig. 60

R. Hiptmair

rev 38355, November 4, 2011

◇



Algorithmic idea: (Lemma 5.1.3 ➢) 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 $\mathbf{x}^* \in D$: $\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}\in D} F(\mathbf{x})$

The most natural iteration:

Algorithm 5.1.7 (Steepest descent). (*ger.:* steilster Abstieg)

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 $\left(\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\| \leq \tau_{\text{rel}} \left\|\mathbf{x}^{(k)}\right\| \right.$ **or** $\left. \left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\| \leq \tau_{\text{abs}} \right)$

- $\mathbf{d}_k \hat{=}$ *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])

$$\mathbf{grad}\, F(\mathbf{x}) = \begin{pmatrix} \frac{\partial F}{\partial x_i} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n \qquad (5.1.8)$$

provides the direction of local steepest ascent/descent of F

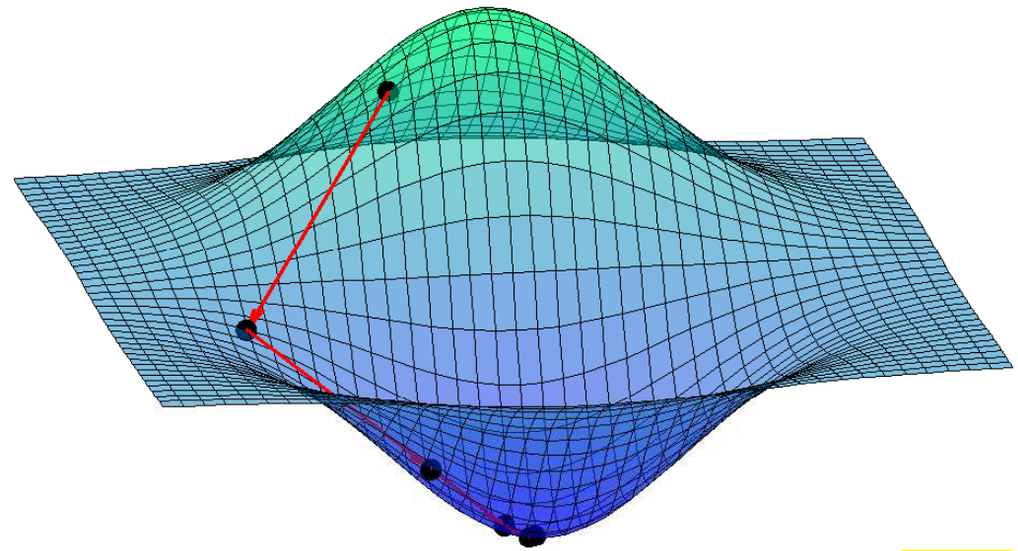


Fig. 61

5.1.3 Gradient method for s.p.d. linear system of equations

Steepest descent iteration = gradient method for LSE $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{n,n}$ 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 $\left(\left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \right\| \leq \tau_{\text{rel}} \left\| \mathbf{x}^{(k)} \right\| \text{ or } \left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \right\| \leq \tau_{\text{abs}} \right)$

Code 5.1.12: gradient method for $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A}$ s.p.d.

```
function x =
  gradit(A,b,x,rtol,atol,maxit)
r = b-A*x; % residual → Def. 2.5.16
for k=1:maxit
  p = A*r;
  ts = (r'*r)/(r'*p); % cf.  (5.1.10)
  x = x + ts*r;
  cn = (abs(ts)*norm(r); % norm of
    correction
  if ((cn < tol*norm(x)) ||
      (cn < atol))
    return; end
  r = r - ts*p; %
end
```

One step of gradient method involves

- $\boxed{\textit{A single} \text{ matrix}\times\text{vector product with } \mathbf{A}}$,
- 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) + $O(n)$

R. Hiptmair

rev 38355, November 4, 2011

5.1.4 Convergence of the gradient method

Example 5.1.14 (Gradient method in 2D).

S.p.d. matrices $\in \mathbb{R}^{2,2}$:

$$\mathbf{A}_1 = \begin{pmatrix} 1.9412 & -0.2353 \\ -0.2353 & 1.0588 \end{pmatrix} \quad , \quad \mathbf{A}_2 = \begin{pmatrix} 7.5353 & -1.8588 \\ -1.8588 & 0.5647 \end{pmatrix}$$

Eigenvalues: $\sigma(\mathbf{A}_1) = \{1, 2\}$, $\sigma(\mathbf{A}_2) = \{0.1, 8\}$

✎ notation: spectrum of a matrix $\in \mathbb{K}^{n,n}$ $\sigma(\mathbf{M}) := \{\lambda \in \mathbb{C}\colon \lambda$ is eigenvalue of $\mathbf{M}\}$

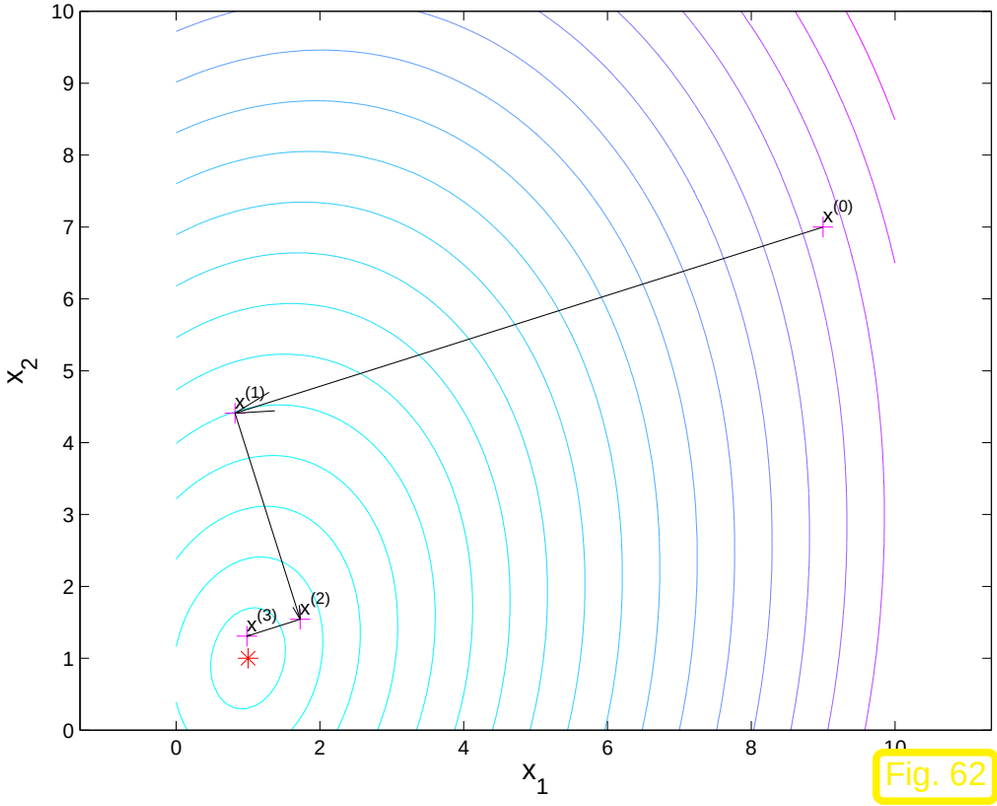


Fig. 62

iterates of Alg. 5.1.11 for $\mathbf{A}_1$

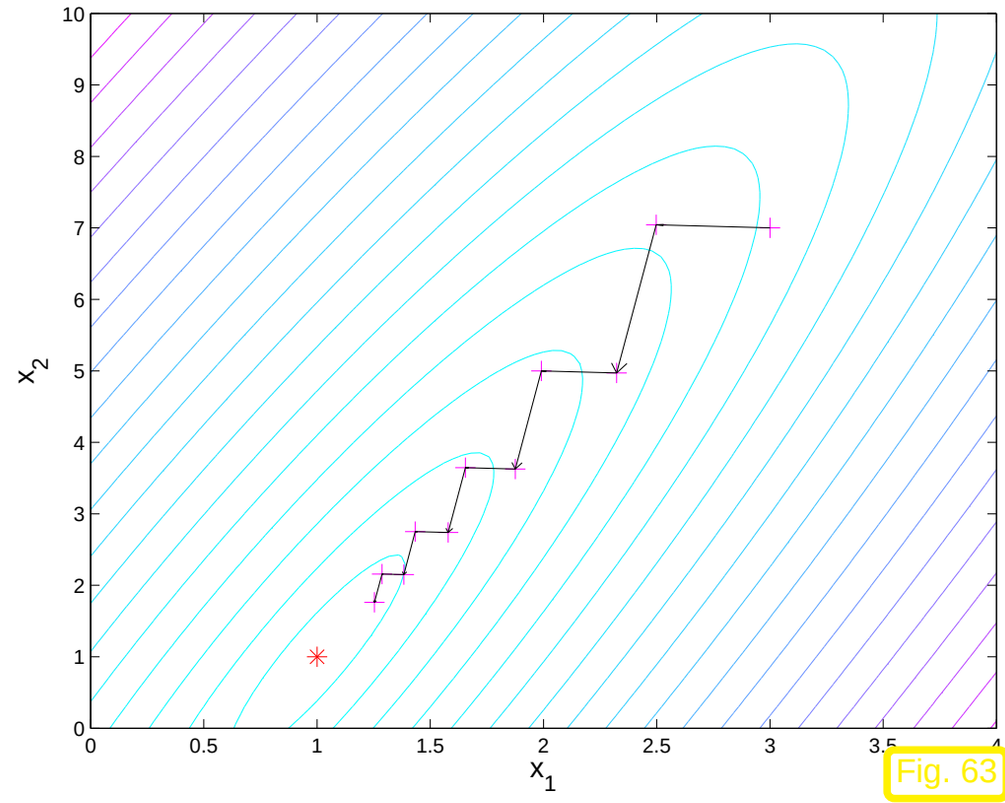


Fig. 63

iterates of Alg. 5.1.11 for $\mathbf{A}_2$

◇

Example 5.1.17 (Convergence of gradient method).

Convergence of gradient method for diagonal matrices, $\mathbf{x}^* = (1,\dots,1)^\top$, $\mathbf{x}^{(0)} = 0$:

```
d = 1:0.01:2;  A1 = diag(d);
d = 1:0.1:11;  A2 = diag(d);
d = 1:1:101;   A3 = diag(d);
```

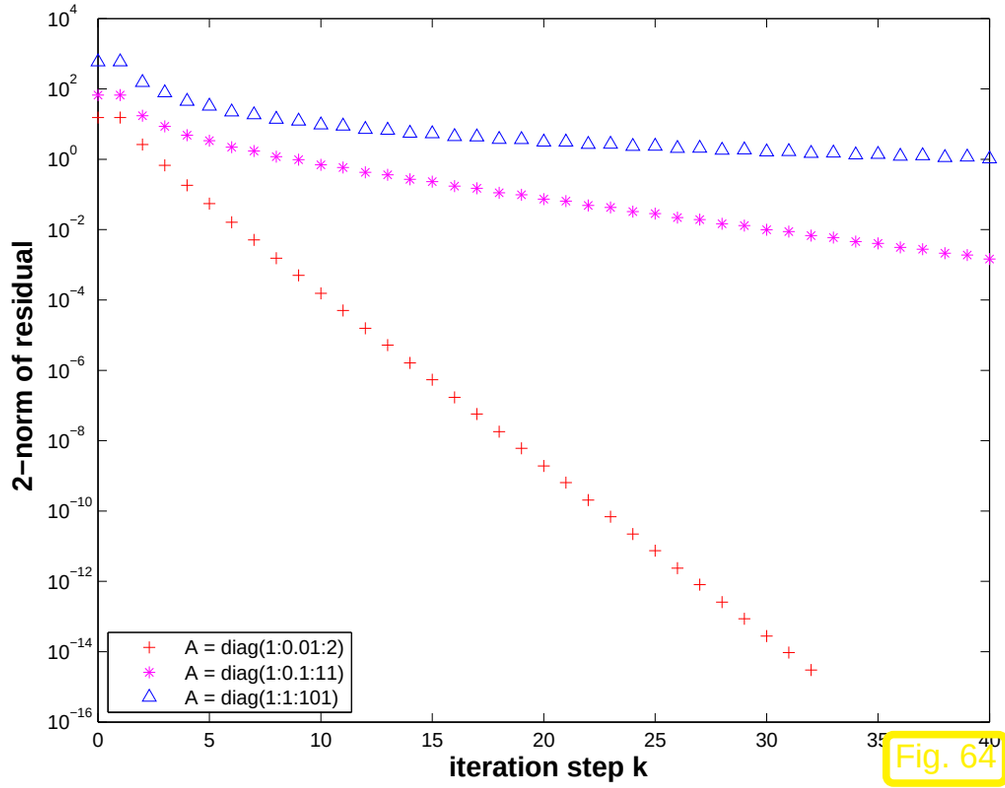


Fig. 64

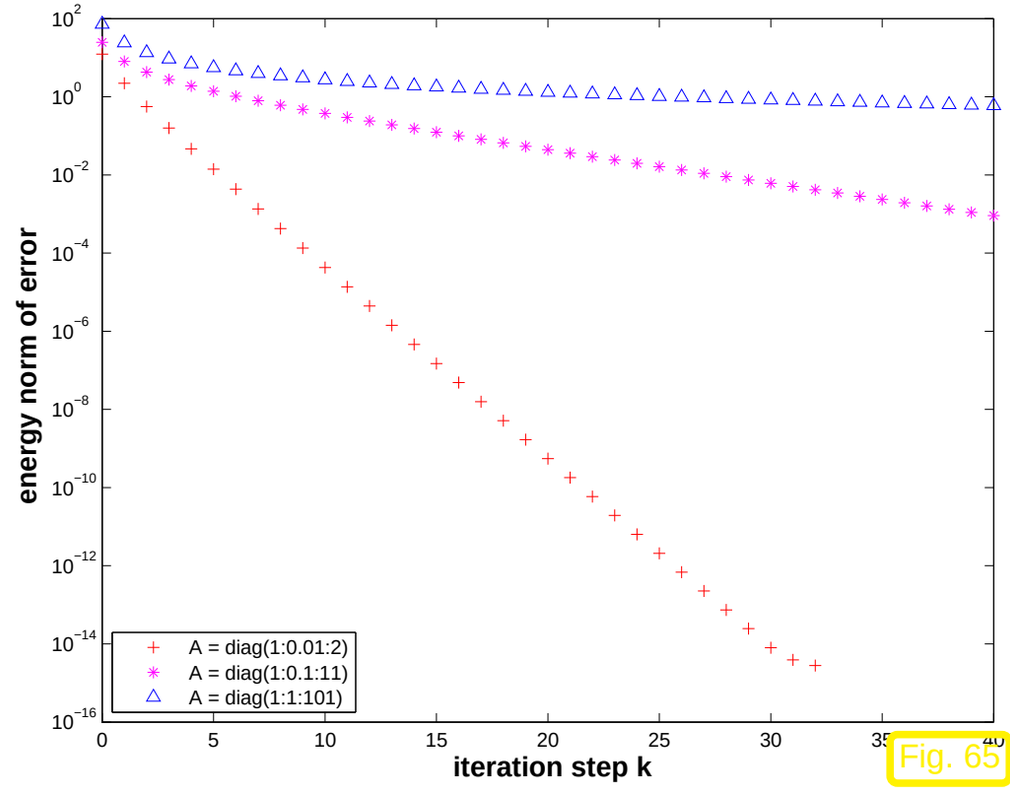


Fig. 65

R. Hiptmair

rev 38355, November 4, 2011

Impact of distribution of diagonal entries ($\leftrightarrow$ eigenvalues) of (diagonal matrix) $\mathbf{A}$

($\mathbf{b} = \mathbf{x}^* = 0$, `x0 = cos((1:n)');`)

Test matrix #1: `A=diag(d); d = (1:100);`

Test matrix #2: `A=diag(d); d = [1+(0:97)/97 , 50 , 100];`

Test matrix #3: `A=diag(d); d = [1+(0:49)*0.05, 100-(0:49)*0.05];`

Test matrix #4: eigenvalues exponentially dense at 1

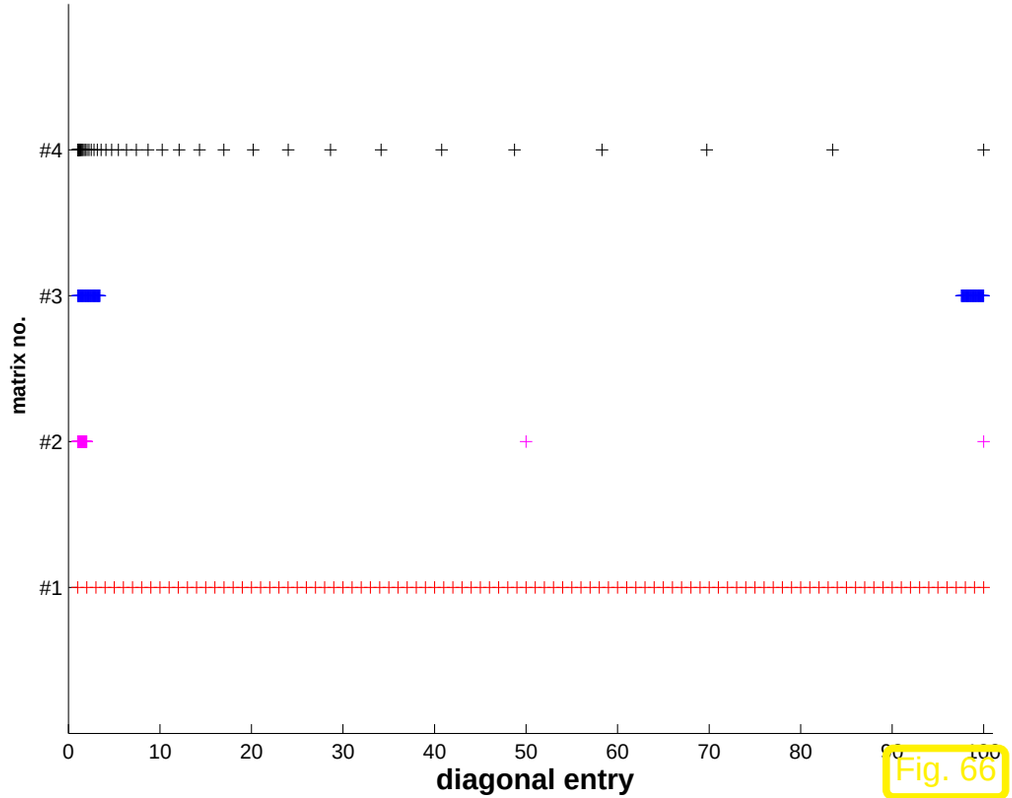
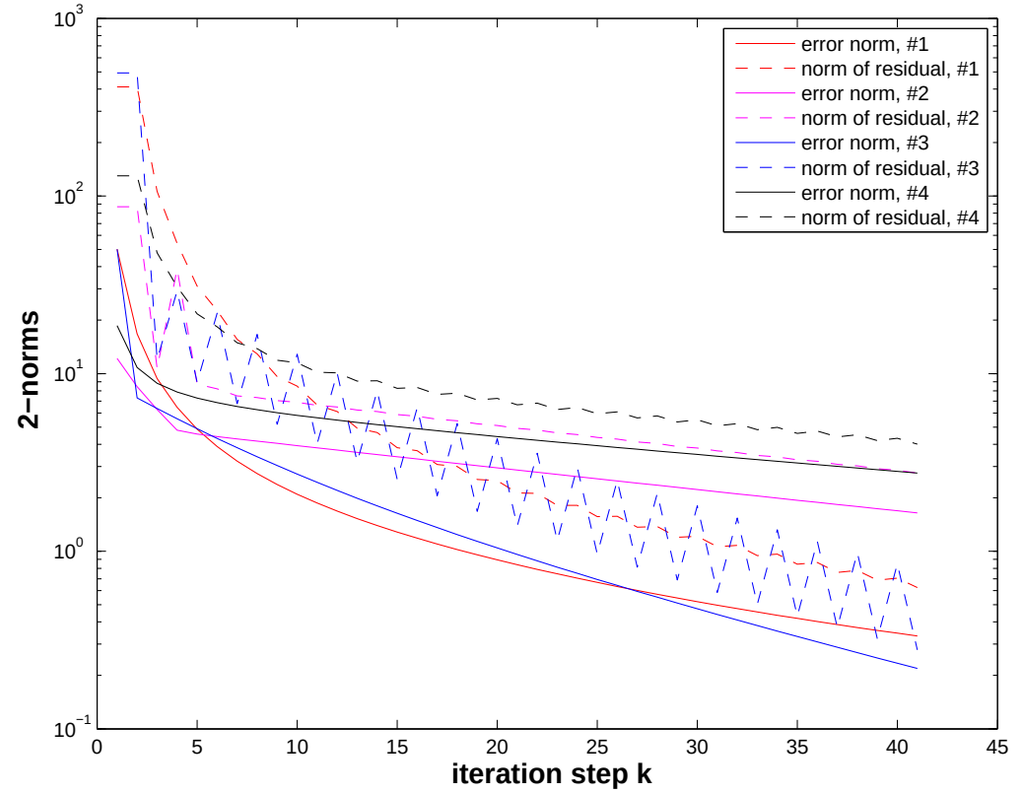


Fig. 66



◇

R. Hiptmair

rev 38355, November 4, 2011

Theorem 5.1.18 (Convergence of gradient method/steepest descent)**.**

The iterates of the gradient method of Alg. 5.1.11 satisfy

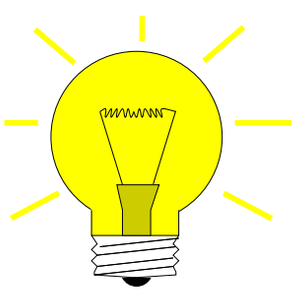
$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\|_A \leq L \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|_A \;, \quad L := \frac{\mathrm{cond}_2(\mathbf{A}) - 1}{\mathrm{cond}_2(\mathbf{A}) + 1} \;,$$

that is, the iteration converges at least linearly (→ Def. 4.1.6) w.r.t. energy norm (→ Def. 5.1.1).

✎ notation: $\mathrm{cond}_2(\mathbf{A}) \hat{=}$ condition number ($\rightarrow$ Def. 2.5.26) of $\mathbf{A}$ induced by 2-norm

5.2 Conjugate gradient method (CG) [35, Ch. 9], [13, Sect. 13.4], [51, Sect. 4.3.4]

Liability of gradient method of Sect. 5.1.3: NO MEMORY



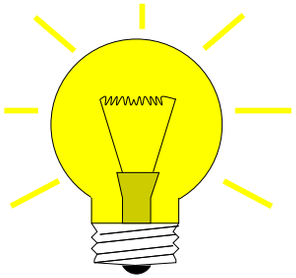
Idea:

Replace linear search with subspace correction

Given:
- initial guess $\mathbf{x}^{(0)}$
- nested subspaces $U_1 \subset U_2 \subset U_3 \subset \cdots \subset U_n = \mathbb{R}^n$, $\dim U_k = k$

$$\mathbf{x}^{(k)} := \underset{\mathbf{x} \in U_k + \mathbf{x}^{(0)}}{\operatorname{argmin}} \; J(x) \,, \tag{5.2.1}$$

quadratic functional from (5.1.4)



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)

$$\boxed{U_{k+1} = \mathrm{Span}\,\{U_k, \mathbf{r}_k\}}\ , \quad \mathbf{x}^{(k)} \text{ from (5.2.1).} \tag{5.2.2}$$

(5.2.1) and (5.2.2) define the conjugate gradient method (CG) for the iterative solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$

5.2.1 Krylov spaces

Definition 5.2.6 (Krylov space)**.**
For $\mathbf{A} \in \mathbb{R}^{n,n}$, $\mathbf{z} \in \mathbb{R}^n$, $\mathbf{z} \neq 0$, *the* l*-th Krylov space is defined as*

$$\mathcal{K}_l(\mathbf{A}, \mathbf{z}) := \operatorname{Span}\left\{\mathbf{z}, \mathbf{A}\mathbf{z}, \dots, \mathbf{A}^{l-1}\mathbf{z}\right\} \ .$$

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*

$$U_k = \operatorname{Span}\left\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \dots, \mathbf{A}^{k-1}\mathbf{r}_0\right\} = \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0) \ ,$$

where $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$ *is the initial residual.*

R. Hiptmair

rev 38355, November 4, 2011

5.2.2 Implementation of CG

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 $\mathbf{Ax} = \mathbf{b}$

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)};$$

$$\textbf{for } j = 1 \textbf{ to } l \textbf{ do } \{ \quad \mathbf{x}^{(j)} := \mathbf{x}^{(j-1)} + \frac{\mathbf{p}_j^\top \mathbf{r}_0}{\mathbf{p}_j^\top \mathbf{A}\mathbf{p}_j} \mathbf{p}_j \quad \} \tag{5.2.9}$$

R. Hiptmair

rev 38355, November 4, 2011

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*.

$\mathbf{A}$-orthogonalities/orthogonalities ➤ short recursions

Algorithm 5.2.17 (CG method for solving $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}$ s.p.d.). $\rightarrow$ [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**;

$\beta := \frac{\mathbf{r}^T \mathbf{r}}{\beta}$;

$\mathbf{p} := \mathbf{r} + \beta \mathbf{p}$;

}

R. Hiptmair

rev 38355, November 4, 2011

1 matrix$\times$vector product, 3 dot products, 3 AXPY-operations per step:

If $\mathbf{A}$ sparse, $\mathrm{nnz}(\mathbf{A}) \sim n$ ➤ computational effort $O(n)$ per step

Code 5.2.18: basic CG iteration for solving $\mathbf{Ax} = \mathbf{b}$, Alg. 5.2.17

```
function x = cg(evalA,b,x,tol,maxit)
% x supplies initial guess, maxit maximal number of CG steps
% evalA must pass a handle to a MATLAB function realizing A*x
r = b - evalA(x); rho = 1; n0 = norm(r);
for i = 1 : maxit
   rho1 = rho; rho = r' * r;
   if (i == 1), p = r;
   else  beta = rho/rho1; p = r + beta * p; end
   q = evalA(p); alpha = rho/(p' * q);
   x = x + alpha * p;     % update of approximate solution
   if (norm(b-A*x) <= tol*n0) return; end % termination, see Rem. 5.2.19
   r = r - alpha * q;       % update of residual
end
```

MATLAB-function:

`x=pcg(A,b,tol,maxit,[],[],x0)` : Solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ with at most `maxit` CG steps: stop, when $\|\mathbf{r}_l\| : \|\mathbf{r}_0\| < \mathrm{tol}$.

`x=pcg(Afun,b,tol,maxit,[],[],x0)`: `Afun` = handle to function for computing $\mathbf{A}\times$vector.

`[x,flag,relr,it,resv] = pcg(...)` : diagnostic information about iteration

5.2.3 Convergence of CG

Example 5.2.21 (Impact of roundoff errors on CG). $\rightarrow$ [51, Rem. 4.3]

Numerical experiment: `A=hilb(20)`, $\mathbf{x}^{(0)} = 0$, $\mathbf{b} = (1,\dots,1)^\top$

Hilbert-Matrix: extremely ill-conditioned

residual norms during CG iteration

$\mathbf{R} = \left[\mathbf{r}_0,\dots,\mathbf{r}^{(10)}\right]$

$\triangleright$:

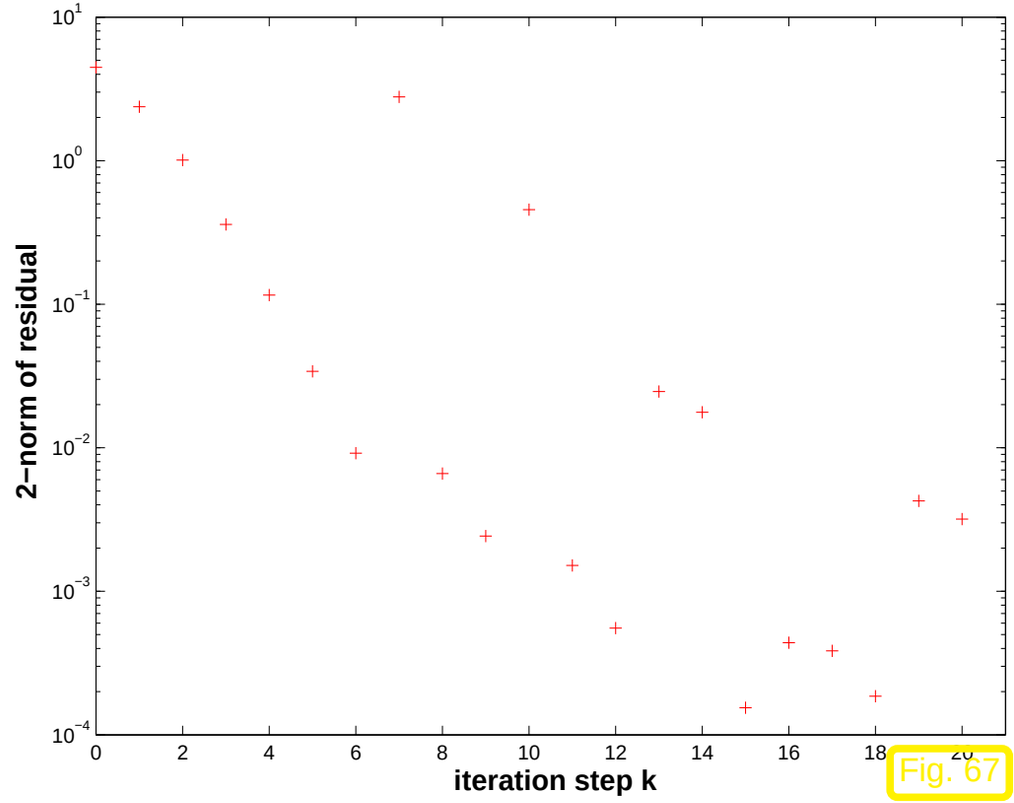


Fig. 67

$\mathbf{R}^\top\mathbf{R} =$

R. Hiptmair

rev 38355, November 4, 2011

$$\begin{pmatrix} 1.000000 & -0.000000 & 0.000000 & -0.000000 & 0.000000 & -0.000000 & 0.016019 & -0.795816 & -0.430569 & 0.348133 \\ -0.000000 & 1.000000 & -0.000000 & 0.000000 & -0.000000 & 0.000000 & -0.012075 & 0.600068 & -0.520610 & 0.420903 \\ 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 & -0.000000 & 0.001582 & -0.078664 & 0.384453 & -0.310577 \\ -0.000000 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 & -0.000024 & 0.001218 & -0.024115 & 0.019394 \\ 0.000000 & -0.000000 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 & -0.000002 & 0.000151 & -0.000118 \\ -0.000000 & 0.000000 & -0.000000 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 & -0.000000 & 0.000000 \\ 0.016019 & -0.012075 & 0.001582 & -0.000024 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & -0.000000 & 0.000000 \\ -0.795816 & 0.600068 & -0.078664 & 0.001218 & -0.000002 & 0.000000 & -0.000000 & 1.000000 & 0.000000 & -0.000000 \\ -0.430569 & -0.520610 & 0.384453 & -0.024115 & 0.000151 & -0.000000 & -0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 0.348133 & 0.420903 & -0.310577 & 0.019394 & -0.000118 & 0.000000 & 0.000000 & -0.000000 & 0.000000 & 1.000000 \end{pmatrix}$$

◇

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);
```

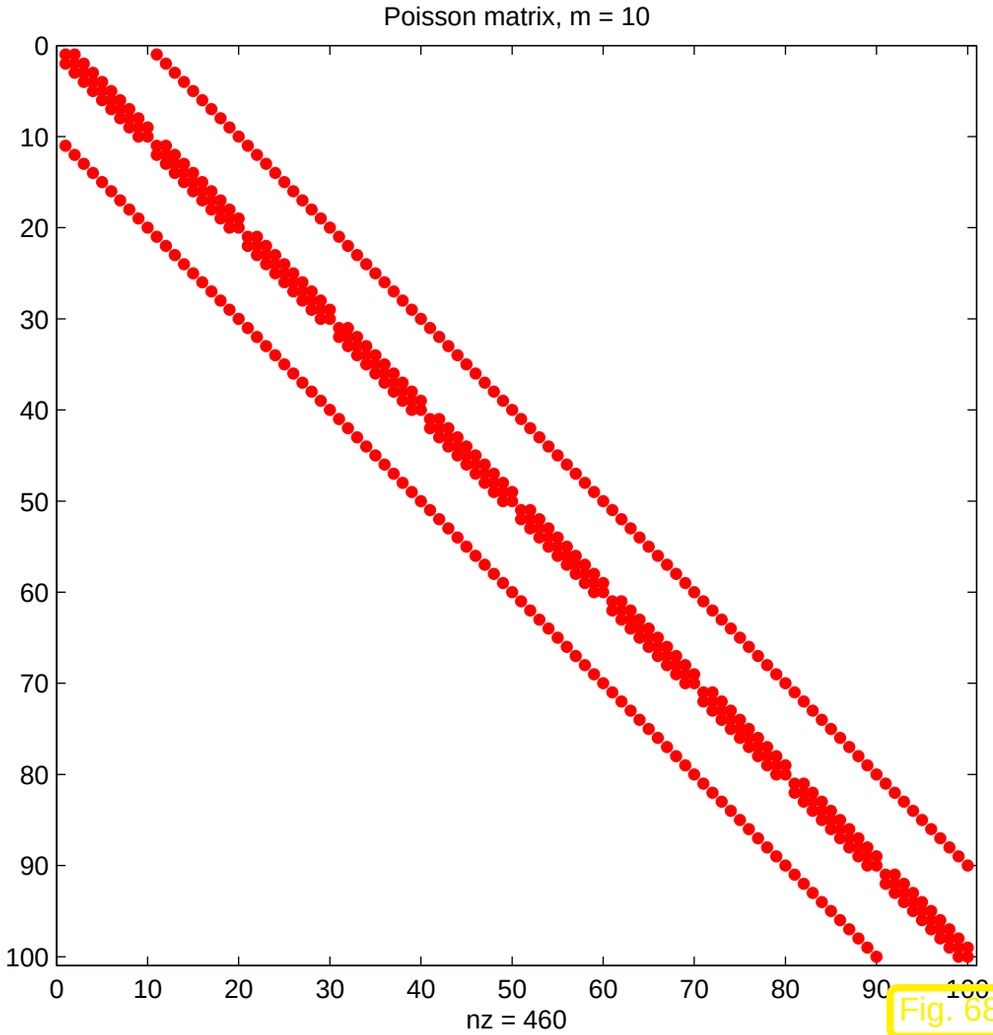

Poisson matrix, m = 10
0
10
20
30
40
50
60
70
80
90
100
0
10
20
30
40
50
60
70
80
90
100
nz = 460

Fig. 68

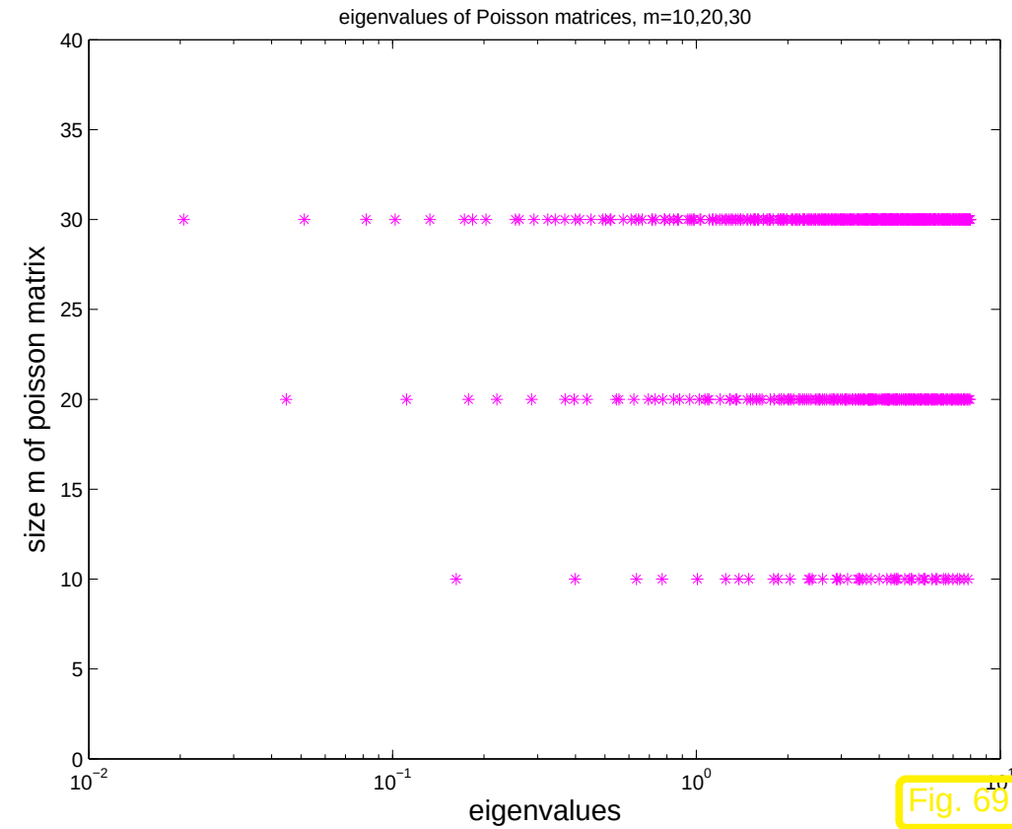

eigenvalues of Poisson matrices, m=10,20,30
size m of poisson matrix
eigenvalues
0
5
10
15
20
25
30
35
40
10^{-2}
10^{-1}
10^{0}

Fig. 69

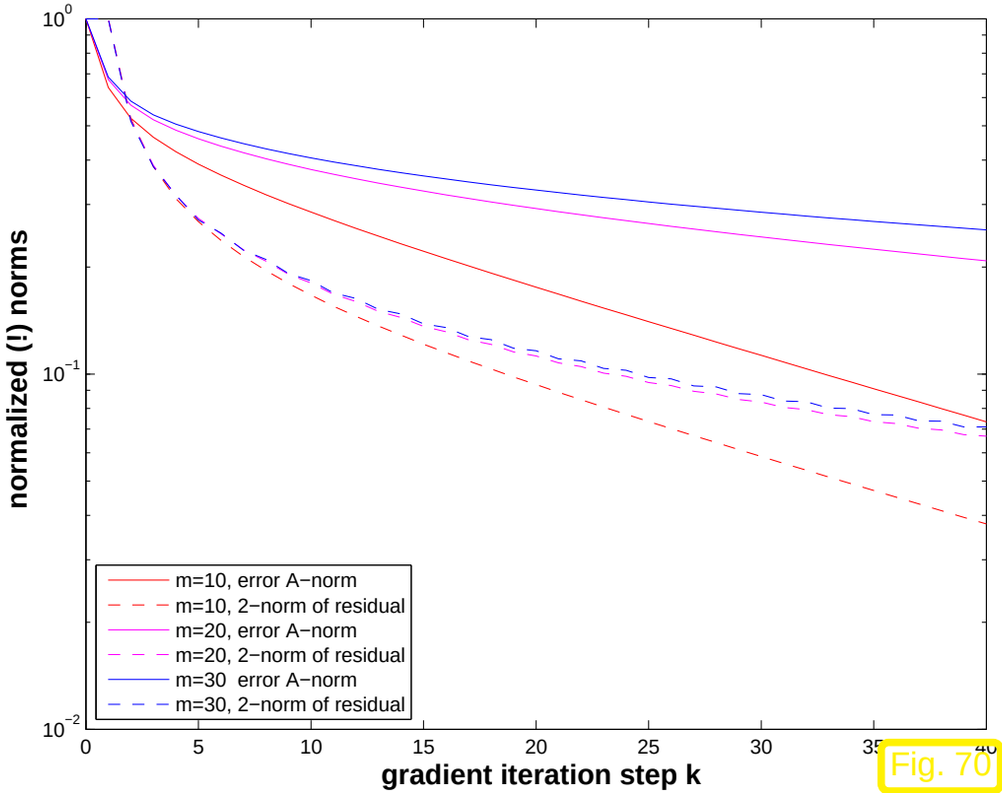


Fig. 70

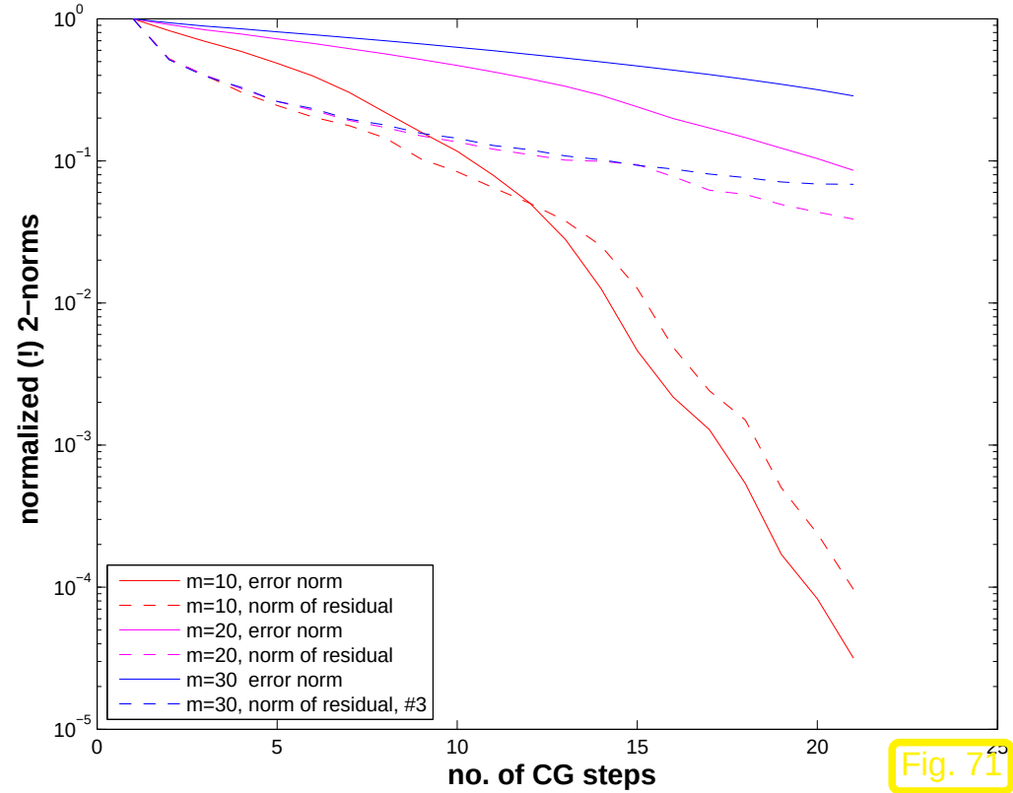


Fig. 71

◇

R. Hiptmair

rev 38355, November 4, 2011

Theorem 5.2.25 (Convergence of CG method)**.**
The iterates of the CG method for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ *(see Code 5.2.17) with* $\mathbf{A} = \mathbf{A}^\top$ *s.p.d. satisfy*

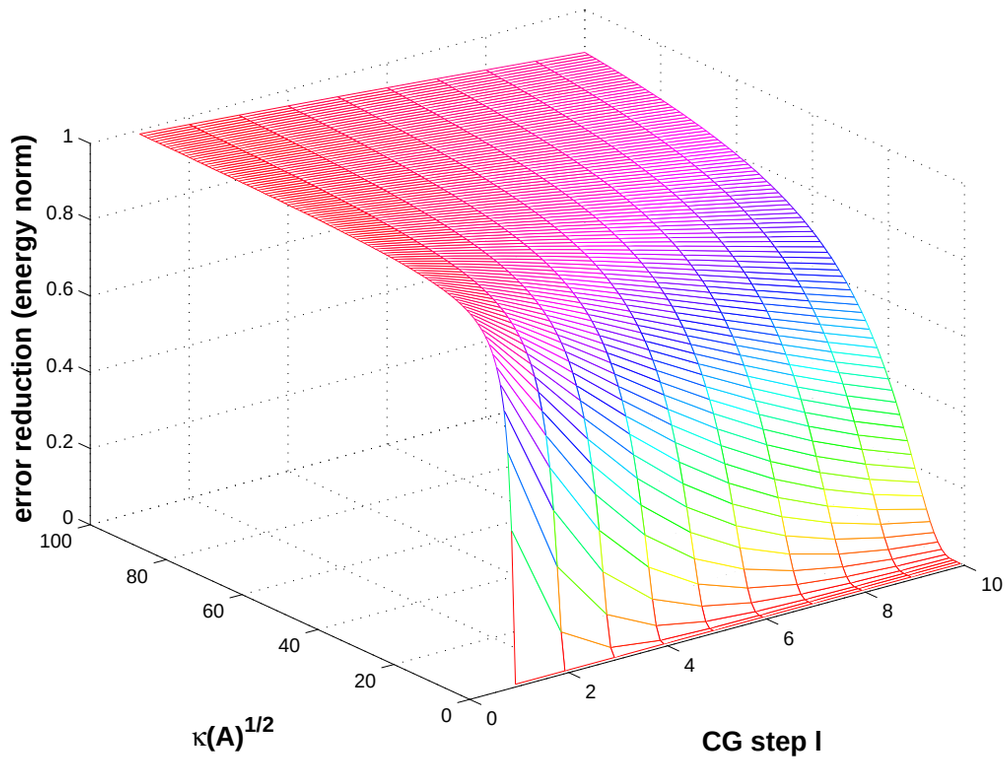
$$\left\|\mathbf{x} - \mathbf{x}^{(l)}\right\|_A \leq \frac{2\left(1 - \frac{1}{\sqrt{\kappa(\mathbf{A})}}\right)^l}{\left(1 + \frac{1}{\sqrt{\kappa(\mathbf{A})}}\right)^{2l} + \left(1 - \frac{1}{\sqrt{\kappa(\mathbf{A})}}\right)^{2l}} \left\|\mathbf{x} - \mathbf{x}^{(0)}\right\|_A$$
$$\leq 2\left(\frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1}\right)^l \left\|\mathbf{x} - \mathbf{x}^{(0)}\right\|_A .$$

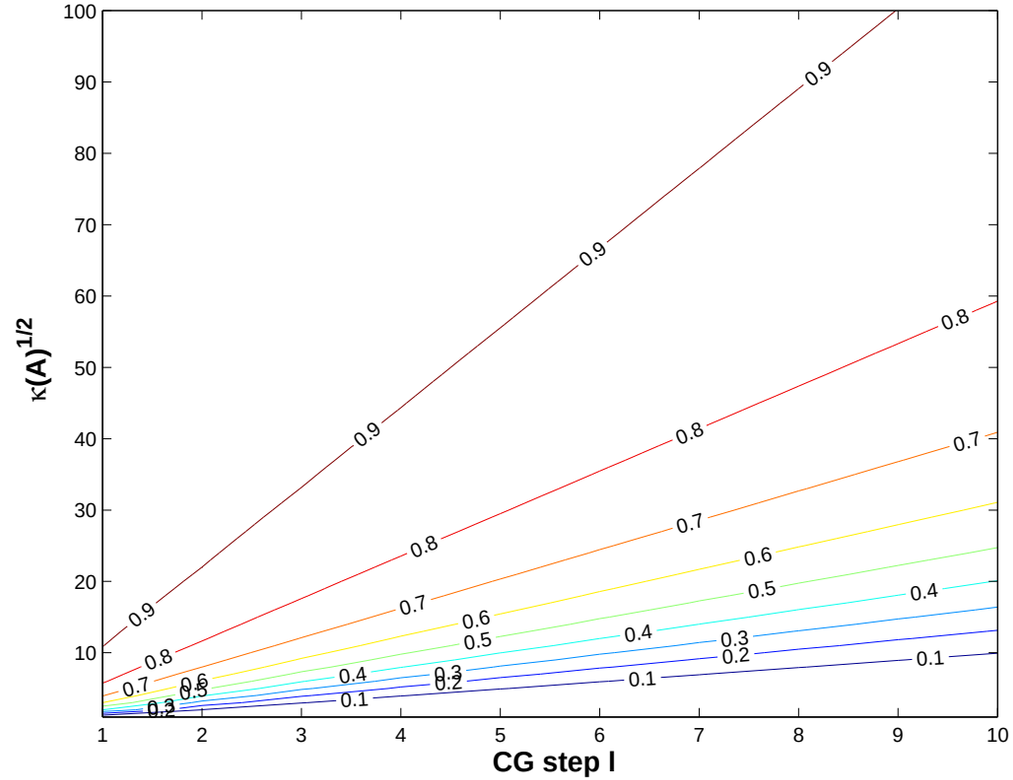
(recall: $\kappa(\mathbf{A})$ *= spectral condition number of* $\mathbf{A}$*,* $\kappa(\mathbf{A}) = \mathrm{cond}_2(\mathbf{A})$*)*

R. Hiptmair

rev 38355, November 4, 2011

Plots of bounds for error reduction (in energy norm) during CG iteration from Thm. 5.2.25:





Example 5.2.27 (Convergence rates for CG method).

Code 5.2.28: CG for Poisson matrix

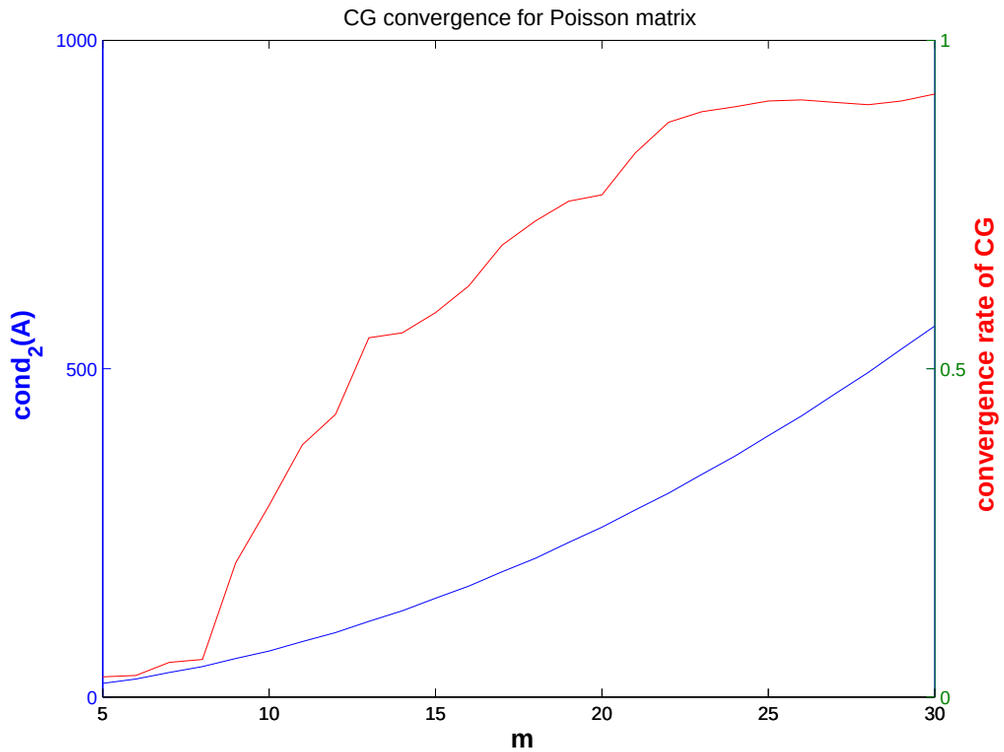
```
A = gallery('poisson',m); n = size(A,1);
x0 = (1:n)'; b = ones(n,1); maxit = 30;
  tol =0;
[x,flag,relres,iter,resvec] =
  pcg(A,b,tol,maxit,[],[],x0);
```

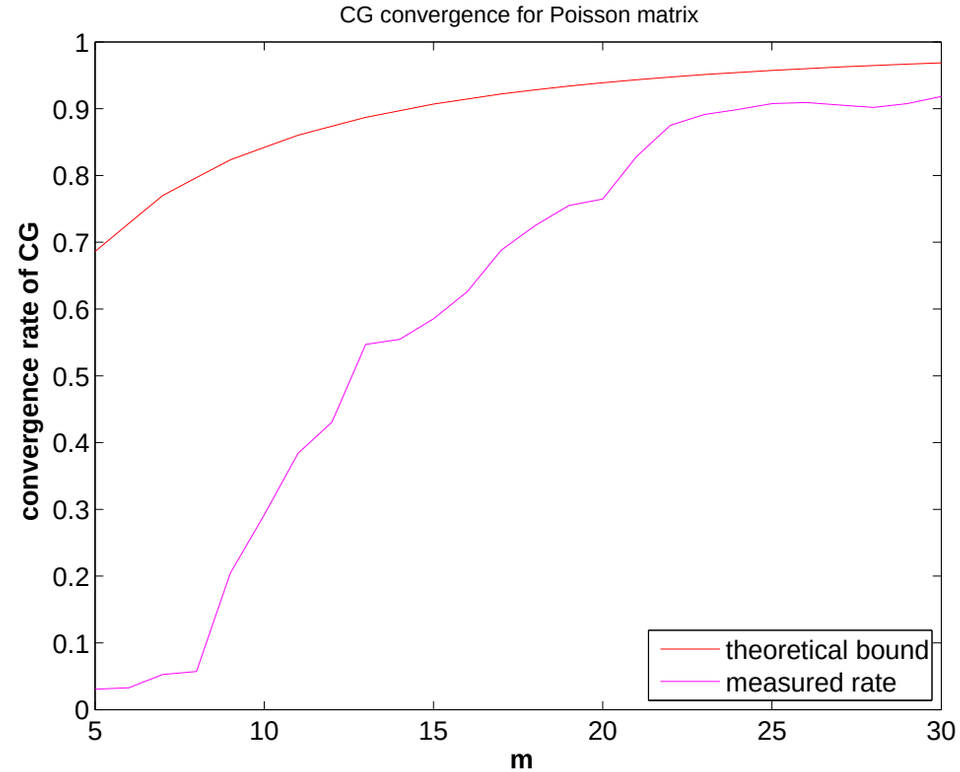
Measurement of rate of (linear) convergence:

$$\text{rate} \approx \sqrt[10]{\frac{\|\mathbf{r}_{30}\|_2}{\|\mathbf{r}_{20}\|_2}} \,. \qquad (5.2.29)$$

R. Hiptmair

rev 38355, November 4, 2011





◇

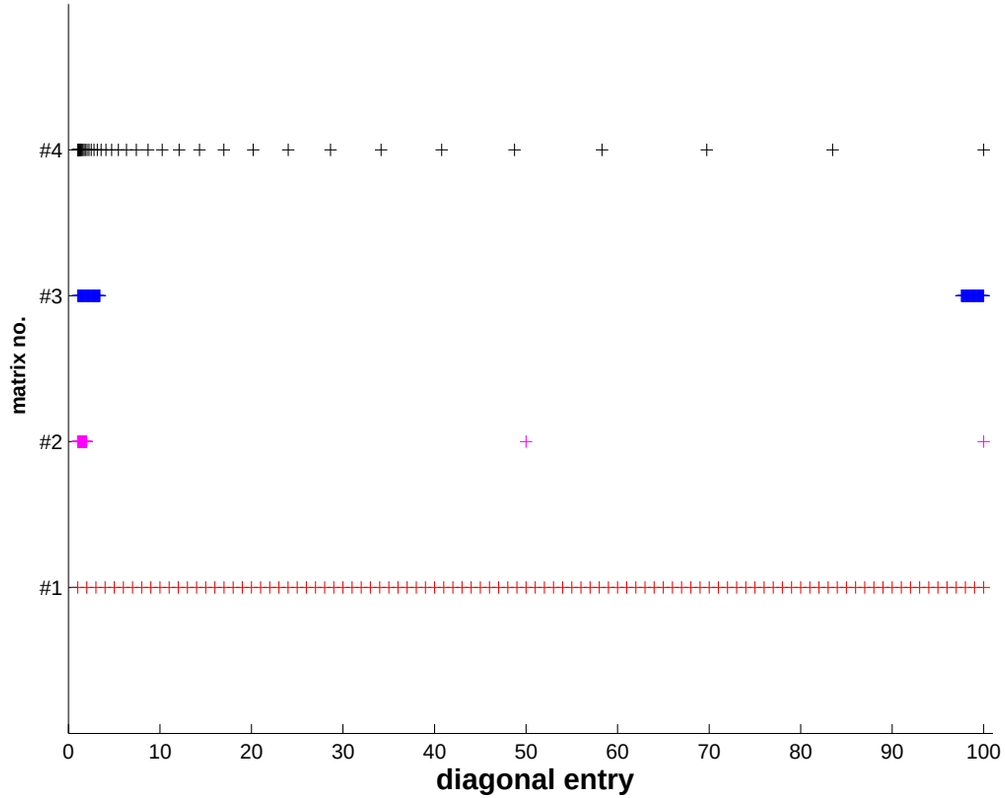
R. Hiptmair

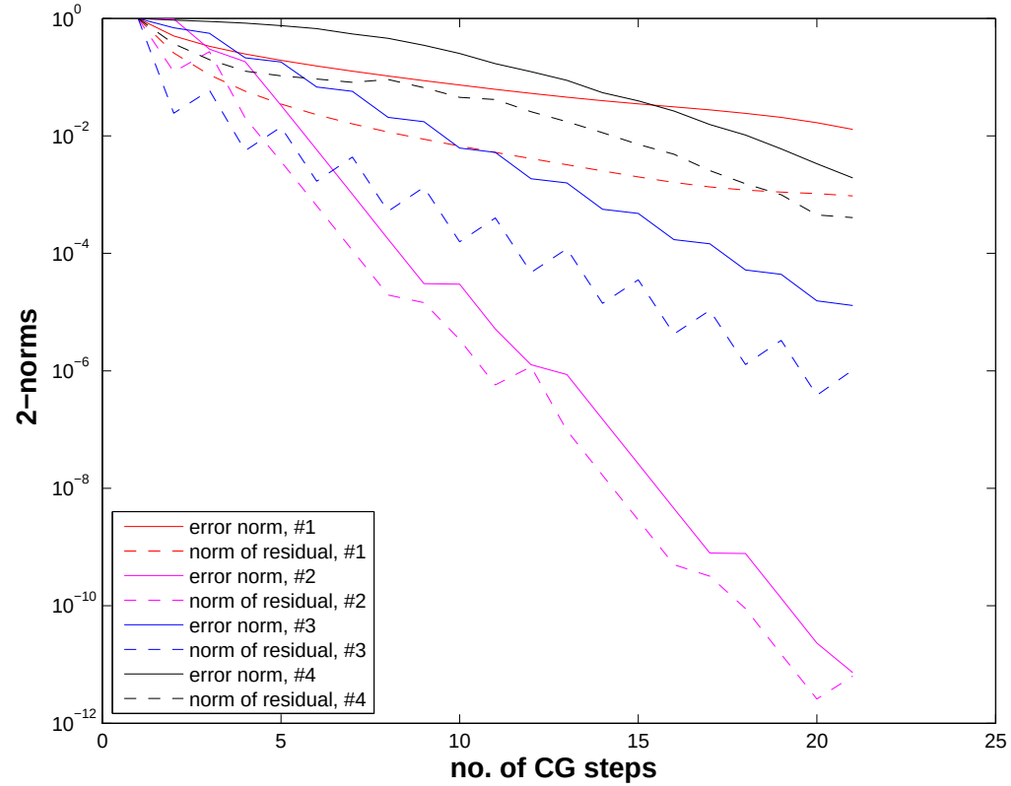
rev 38355, November 4, 2011

Example 5.2.30 (CG convergence and spectrum). $\longrightarrow$ Ex. 5.1.17

Test matrix #1: `A=diag(d); d = (1:100);`
Test matrix #2: `A=diag(d); d = [1+(0:97)/97 , 50 , 100];`
Test matrix #3: `A=diag(d); d = [1+(0:49)*0.05, 100-(0:49)*0.05];`
Test matrix #4: eigenvalues exponentially dense at 1

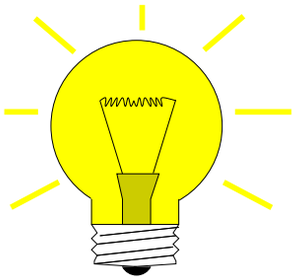
```
x0 = cos((1:n)'); b = zeros(n,1);
```


#4
#3
#2
#1
matrix no.
0
10
20
30
40
50
60
70
80
90
100
diagonal entry


10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
2−norms
0
5
10
15
20
25
no. of CG steps
error norm, #1
norm of residual, #1
error norm, #2
norm of residual, #2
error norm, #3
norm of residual, #3
error norm, #4
norm of residual, #4

◇

5.3 Preconditioning [13, Sect. 13.5], [35, Ch. 10], [51, Sect. 4.3.5]



Idea: Preconditioning

Apply CG method to transformed linear system

$$\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{b}} \quad , \quad \widetilde{\mathbf{A}} := \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2} \, , \, \widetilde{\mathbf{x}} := \mathbf{B}^{1/2}\mathbf{x} \, , \, \widetilde{\mathbf{b}} := \mathbf{B}^{-1/2}\mathbf{b} \, , \tag{5.3.1}$$

with "small" $\kappa(\widetilde{\mathbf{A}})$, $\mathbf{B} = \mathbf{B}^\top \in \mathbb{R}^{N,N}$ s.p.d. $\hat{=}$ preconditioner.

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$\times$vector multiplication $\mathbf{A}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^n$.*

CG for $\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{b}}$

Input : initial guess $\widetilde{\mathbf{x}}^{(0)} \in \mathbb{R}^n$

Output : approximate solution $\widetilde{\mathbf{x}}^{(l)} \in \mathbb{R}^n$

$\widetilde{\mathbf{p}}_1 := \widetilde{\mathbf{r}}_0 := \widetilde{\mathbf{b}} - \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\widetilde{\mathbf{x}}^{(0)}$;

for $j = 1$ **to** l **do** {

$$\alpha := \frac{\widetilde{\mathbf{p}}_j^T \widetilde{\mathbf{r}}_{j-1}}{\widetilde{\mathbf{p}}_j^T \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j}$$

$\widetilde{\mathbf{x}}^{(j)} := \widetilde{\mathbf{x}}^{(j-1)} + \alpha\,\widetilde{\mathbf{p}}_j$;

$\widetilde{\mathbf{r}}_j = \widetilde{\mathbf{r}}_{j-1} - \alpha\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{1/2}\widetilde{\mathbf{p}}_j$;

$$\widetilde{\mathbf{p}}_{j+1} = \widetilde{\mathbf{r}}_j - \frac{(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j)^T\widetilde{\mathbf{r}}_j}{\widetilde{\mathbf{p}}_j^T \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j}\,\widetilde{\mathbf{p}}_j;$$

}

Equivalent CG with transformed variables

Input : initial guess $\mathbf{x}^{(0)} \in \mathbb{R}^n$

Output : approximate solution $\mathbf{x}^{(l)} \in \mathbb{R}^n$

$\mathbf{B}^{1/2}\widetilde{\mathbf{r}}_0 := \mathbf{B}^{1/2}\widetilde{\mathbf{b}} - \mathbf{A}\mathbf{B}^{-1/2}\widetilde{\mathbf{x}}^{(0)}$;

$\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_1 := \mathbf{B}^{-1}(\mathbf{B}^{1/2}\widetilde{\mathbf{r}}_0)$;

for $j = 1$ **to** l **do** {

$$\alpha := \frac{(\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j)^T\mathbf{B}^{1/2}\widetilde{\mathbf{r}}_{j-1}}{(\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j)^T\mathbf{A}\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j}$$

$\mathbf{B}^{-1/2}\widetilde{\mathbf{x}}^{(j)} := \mathbf{B}^{-1/2}\widetilde{\mathbf{x}}^{(j-1)} + \alpha\,\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j$;

$\mathbf{B}^{1/2}\widetilde{\mathbf{r}}_j = \mathbf{B}^{1/2}\widetilde{\mathbf{r}}_{j-1} - \alpha\mathbf{A}\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j$;

$$\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_{j+1} = \mathbf{B}^{-1}(\mathbf{B}^{-1/2}\widetilde{\mathbf{r}}_j) - \frac{(\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j)^T\mathbf{A}\mathbf{B}^{-1}(\mathbf{B}^{1/2}\widetilde{\mathbf{r}}_j)}{(\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j)^T\mathbf{A}\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j}\,\mathbf{B}^{-1/2}\widetilde{\mathbf{p}}_j;$$

}

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 $\mathbf{x} \hat{=} \mathbf{x}^{(l)}$

$\mathbf{p} := \mathbf{r} := \mathbf{b} - \mathbf{A}\mathbf{x}$; $\mathbf{p} := \mathbf{B}^{-1}\mathbf{r}$; $\mathbf{q} := \mathbf{p}$; $\tau_0 := \mathbf{p}^\top \mathbf{r}$;

for $l = 1$ **to** $l_{\max}$ **do** {

$\beta := \mathbf{r}^\top \mathbf{q}$; $\mathbf{h} := \mathbf{A}\mathbf{p}$; $\alpha := \frac{\beta}{\mathbf{p}^\top \mathbf{h}}$;

$\mathbf{x} := \mathbf{x} + \alpha \mathbf{p}$;

$\mathbf{r} := \mathbf{r} - \alpha \mathbf{h}$; (5.3.6)

$\mathbf{q} := \mathbf{B}^{-1}\mathbf{r}$; $\beta := \frac{\mathbf{r}^\top \mathbf{q}}{\beta}$;

if $|\mathbf{q}^\top \mathbf{r}| \leq \tau \cdot \tau_0$ **then stop**;

$\mathbf{p} := \mathbf{q} + \beta \mathbf{p}$;

}

Computational effort per step: 1 evaluation $\mathbf{A} \times$vector,
1 evaluation $\mathbf{B}^{-1} \times$vector,
3 dot products, 3 AXPY-operations

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

```
A =
  spdiags(repmat([1/n,-1,2+2/n,-1,1/n],n,1),[-n/2,-1,0,1,n/2],n,n);
b = ones(n,1); x0 = ones(n,1); tol = 1.0E-4; maxit = 1000;
evalA = @(x) A*x;

% no preconditioning, see Code 5.3.6
invB = @(x) x; [x,rn] = pcgbase(evalA,b,tol,maxit,invB,x0);

% tridiagonal preconditioning, see Code 5.3.6
B = spdiags(spdiags(A,[-1,0,1]),[-1,0,1],n,n);
invB  = @(x) B\x; [x,rnpc] = pcgbase(evalA,b,tol,maxit,invB,x0); %
```


R. Hiptmair

rev 38355, November 4, 2011

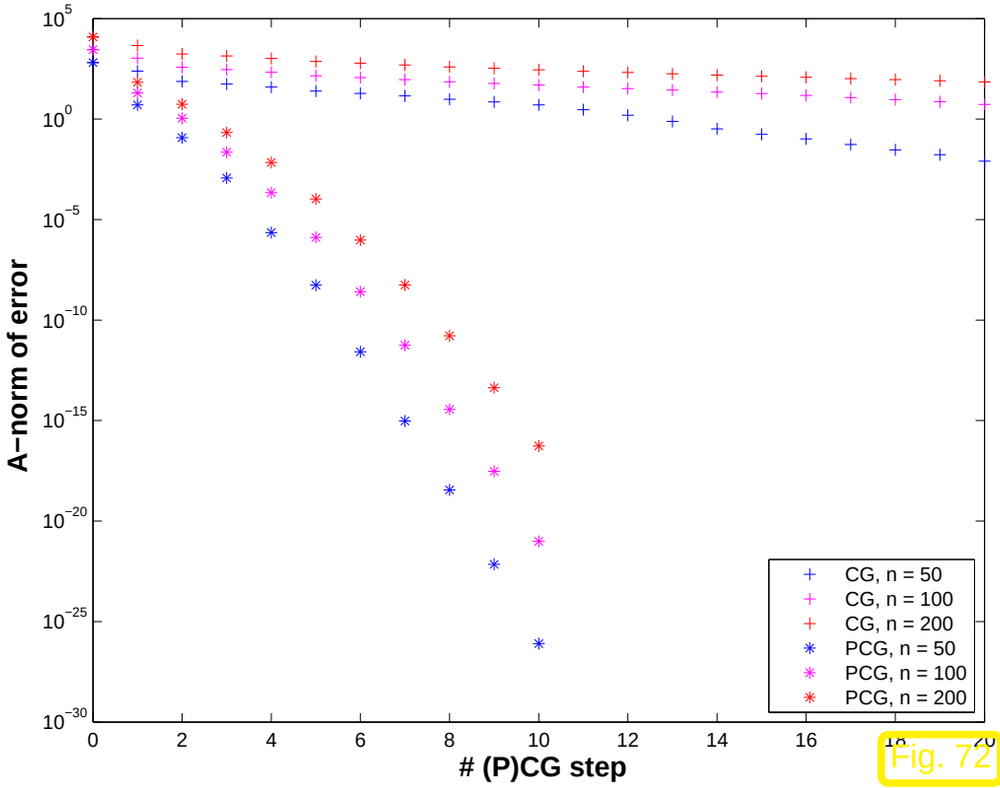

A−norm of error
(P)CG step
CG, n = 50
CG, n = 100
CG, n = 200
PCG, n = 50
PCG, n = 100
PCG, n = 200

Fig. 72

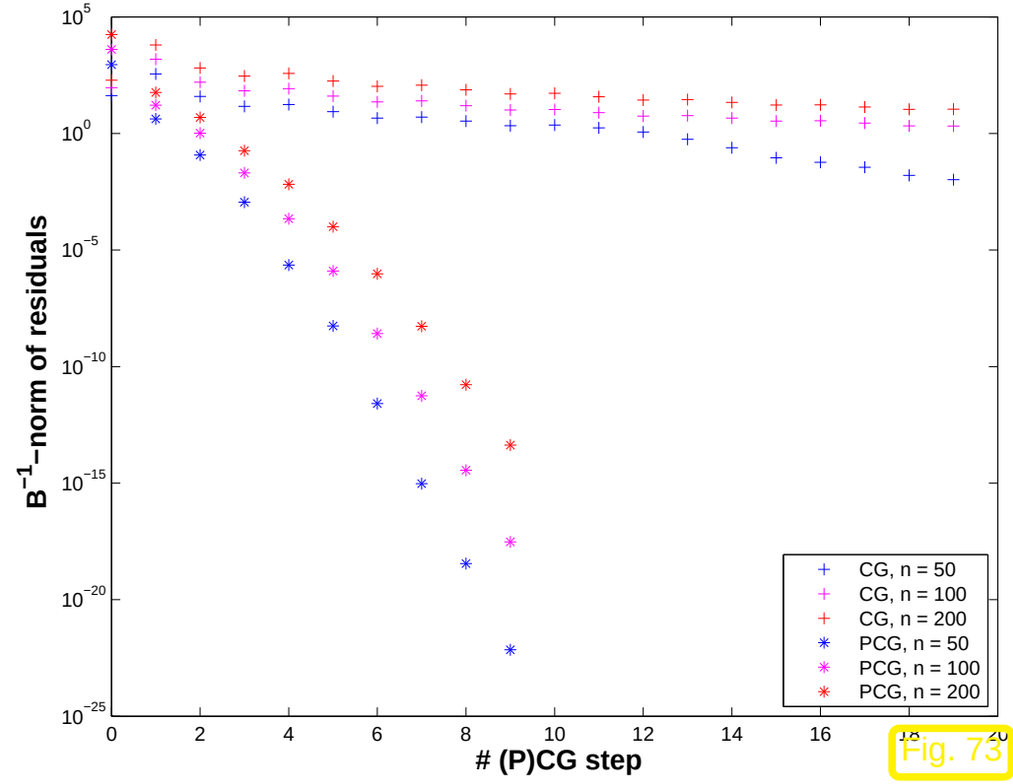

B^−1−norm of residuals
(P)CG step
CG, n = 50
CG, n = 100
CG, n = 200
PCG, n = 50
PCG, n = 100
PCG, n = 200

Fig. 73

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

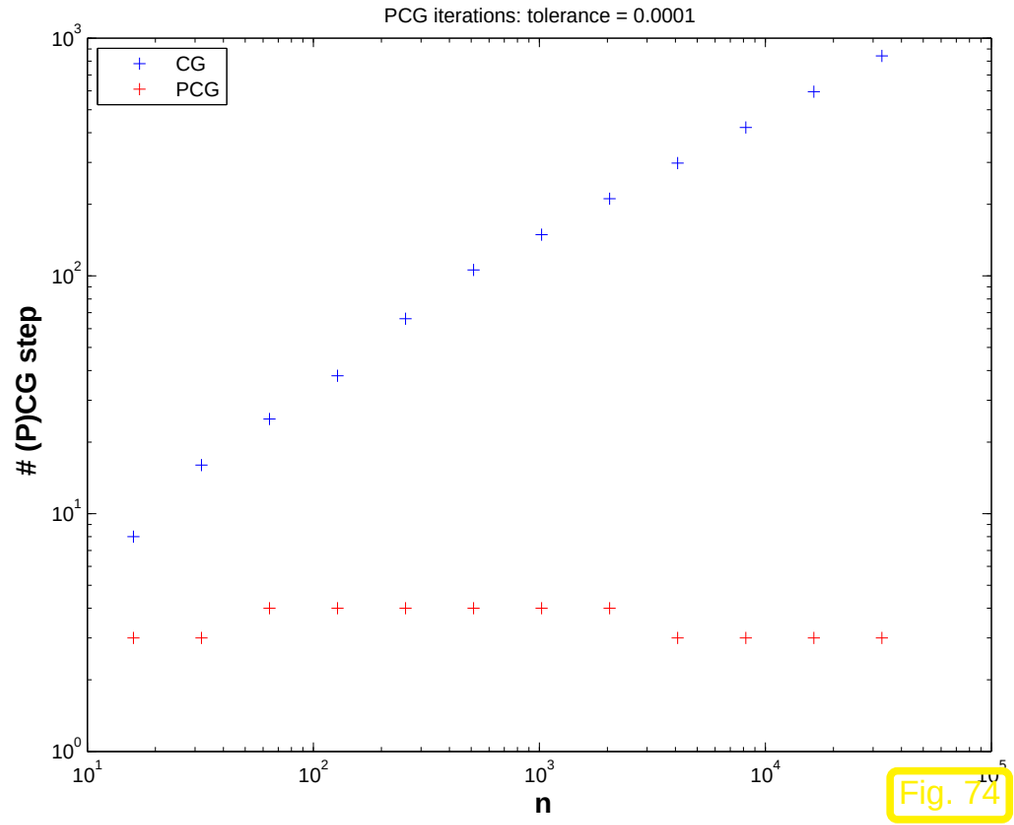


Fig. 74

◇

Remark 5.3.13 (Termination of PCG).

available during PCG iteration (5.3.6)

$$\boxed{\frac{1}{\kappa(\mathbf{B}^{-1}\mathbf{A})}\frac{\left\|\mathbf{e}^{(l)}\right\|_A^2}{\left\|\mathbf{e}^{(0)}\right\|_A^2} \leq \frac{(\mathbf{B}^{-1}\mathbf{r}_l)^\top\mathbf{r}_l}{(\mathbf{B}^{-1}\mathbf{r}_0)^\top\mathbf{r}_0} \leq \kappa(\mathbf{B}^{-1}\mathbf{A})\frac{\left\|\mathbf{e}^{(l)}\right\|_A^2}{\left\|\mathbf{e}^{(0)}\right\|_A^2}} \qquad (5.3.14)$$

MATLAB-function: `[x,flag,relr,it,rv] = pcg(A,b,tol,maxit,B,[],x0);`

(A, B may be handles to functions providing $\mathbf{Ax}$ and $\mathbf{B}^{-1}\mathbf{x}$, resp.)

Remark 5.3.15 (Termination criterion in MATLAB-`pcg`). $\rightarrow$ [51, Sect. 4.6]

Implementation (skeleton) of MATLAB built-in `pcg`:

MATLAB PCG algorithm

```
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)) <= tolb*norm(b)), return; end
  r = r - alpha * q;
end
```

Dubious termination criterion !

△

5.4 Survey of Krylov Subspace Methods

5.4.1 Minimal residual methods

Iterative solver for $\mathbf{A}\mathbf{x} = \mathbf{b}$ 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,Binvfun,[],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 $\mathbf{A} \in \mathbb{R}^{n,n}$:

MATLAB-function:
- `[x,flag,relr,it,rv] = gmres(A,b,rs,tol,maxit,B,[],x0);`
- `[...] = gmres(Afun,b,rs,tol,maxit,Binvfun,[],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

5.4.2 Iterations with short recursions [51, Sect. 4.5]

Iterative methods for *general* regular system matrix $\mathbf{A}$:

MATLAB-function:
- `[x,flag,r,it,rv] = bicgstab(A,b,tol,maxit,B,[],x0)`
- `[...] = bicgstab(Afun,b,tol,maxit,Binvfun,[],x0);`

Computational costs : 2 $\mathbf{A}\times$vector, 2 $\mathbf{B}^{-1}\times$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,Binvfun,[],x0);`

Computational costs : 2 $\mathbf{A}\times$vector, 2 $\mathbf{B}^{-1}\times$vector, 2 dot products, 12 SAXPYs per step
Memory requirements: 10 vectors $\in \mathbb{R}^n$

- little (useful) covergence theory available
- stagnation & “breakdowns” commonly occur

Example 5.4.3 (Failure of Krylov iterative solvers).

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & \cdots & & & \cdots & 0 \\ 0 & 0 & 1 & 0 & & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & & \\ & & & & & & \ddots & \vdots \\ \vdots & & & & & \ddots & \ddots & 0 \\ 0 & & & & & & 0 & 1 \\ 1 & 0 & \cdots & & & & \cdots & 0 \end{pmatrix} \quad , \quad \mathbf{b} = \begin{pmatrix} 0 \\ \vdots \\ \\ \\ \vdots \\ 0 \\ 1 \end{pmatrix} \quad \blacktriangleright \quad \mathbf{x} = \mathbf{e}_1 \; .$$

$$\mathbf{x}^{(0)} = 0 \quad \succ \quad \mathbf{r}_0 = \mathbf{e}_n \quad \succ \quad \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0) = \operatorname{Span}\{\mathbf{e}_n, \mathbf{e}_{n-1}, \dots, \mathbf{e}_{n-l+1}\}$$

$$\blacktriangleright \qquad \min\{\|\mathbf{y} - \mathbf{x}\|_2 : \mathbf{y} \in \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\} = \begin{cases} 1 & \text{, if } l \leq n \; , \\ 0 & \text{, for } l = n \; . \end{cases}$$

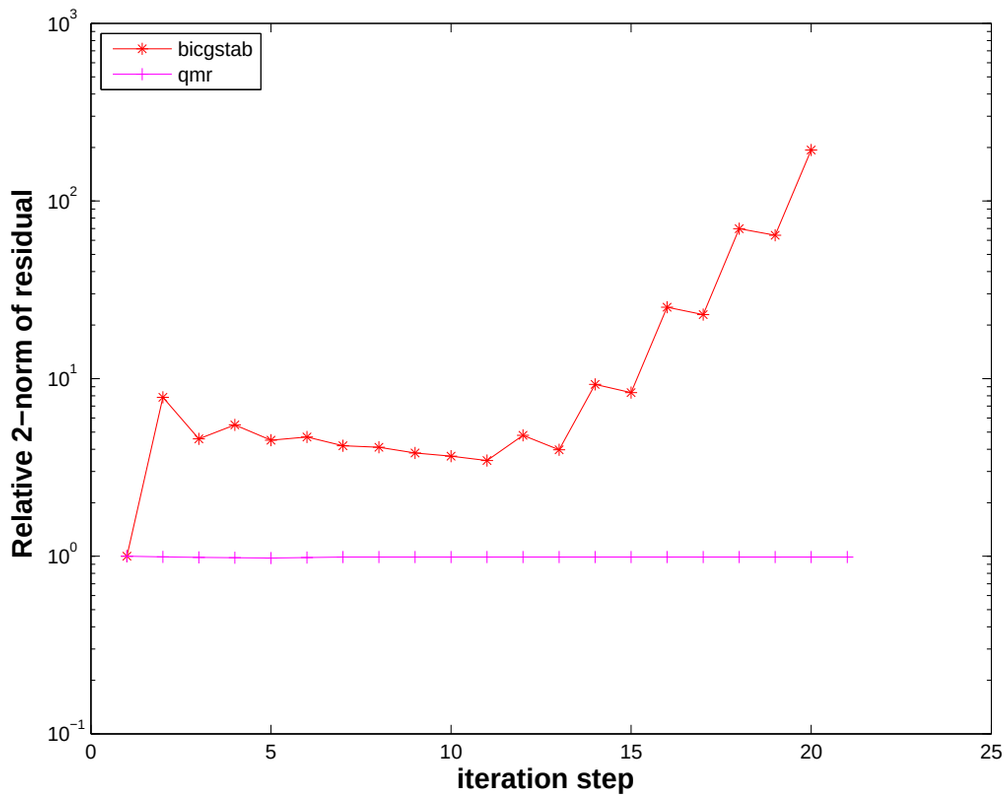
◇

TRY & PRAY

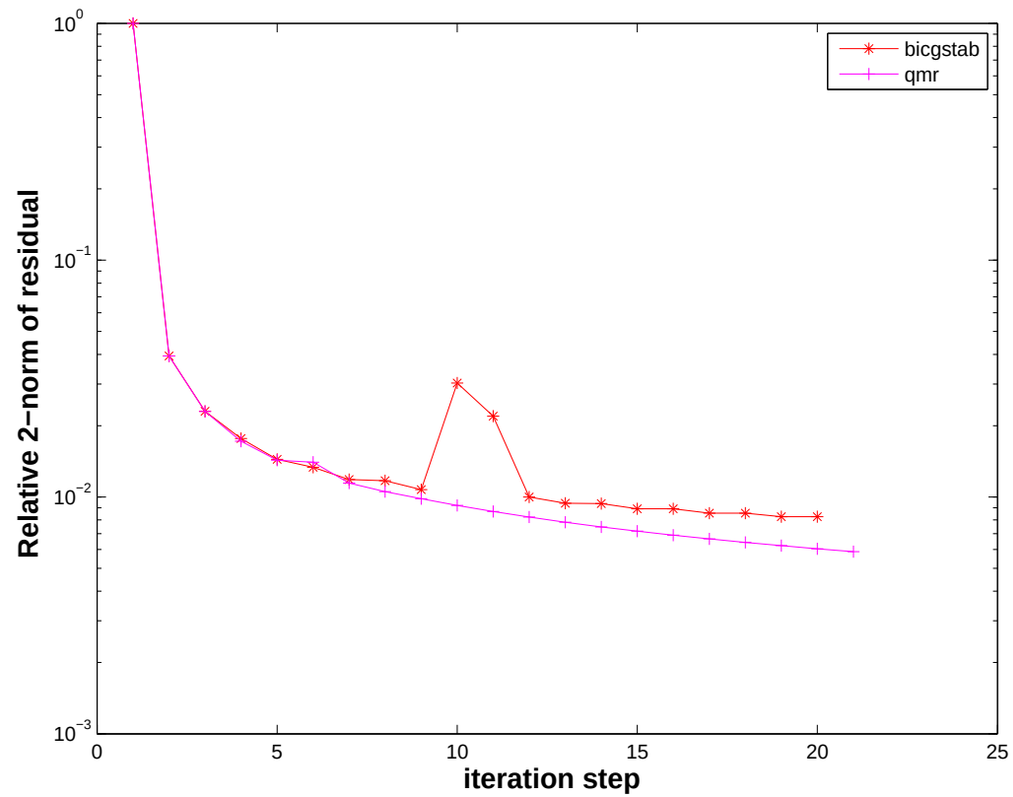
Example 5.4.4 (Convergence of Krylov subspace methods for non-symmetric system matrix).

```
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 $\mathbf{A}$



tridiagonal matrix $\mathbf{B}$

◇

6 Eigenvalues

Example 6.0.1 (Resonances of linear electric circuits).

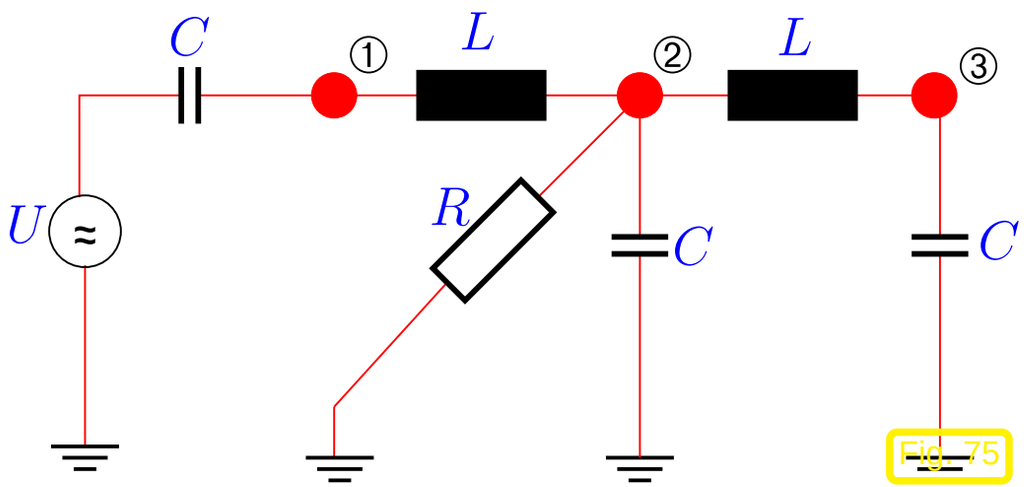


Fig. 75

➢ system matrix from nodal analysis at angular frequency $\omega > 0$:

$$\mathbf{A} = \begin{pmatrix} \imath\omega C + \frac{1}{\imath\omega L} & -\frac{1}{\imath\omega L} & 0 \\ -\frac{1}{\imath\omega L} & \imath\omega C + \frac{1}{R} + \frac{2}{\imath\omega L} & -\frac{1}{\imath\omega L} \\ 0 & -\frac{1}{\imath\omega L} & \imath\omega C + \frac{1}{\imath\omega L} \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{R} & 0 \\ 0 & 0 & 0 \end{pmatrix} + \imath\omega \begin{pmatrix} C & 0 & 0 \\ 0 & C & 0 \\ 0 & 0 & C \end{pmatrix} + {}^{1}\!/\!_{\imath\omega} \begin{pmatrix} \frac{1}{L} & -\frac{1}{L} & 0 \\ -\frac{1}{L} & \frac{2}{L} & -\frac{1}{L} \\ 0 & -\frac{1}{L} & \frac{1}{L} \end{pmatrix} .$$

R. Hiptmair

rev 38355, November 11, 2011

$$\mathbf{A}(\omega) := \mathbf{W} + i\omega\mathbf{C} - i\omega^{-1}\mathbf{S} \quad , \quad \mathbf{W}, \mathbf{C}, \mathbf{S} \in \mathbb{R}^{n,n} \text{ symmetric .} \tag{6.0.2}$$

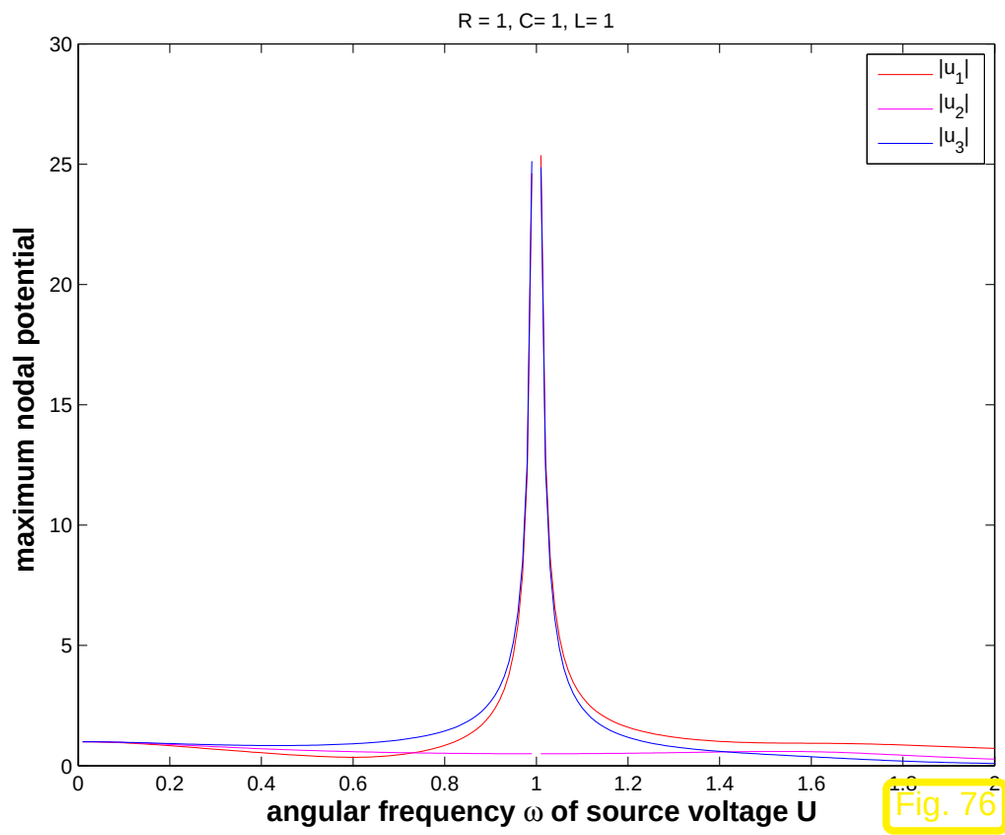


Fig. 76

◁ plot of $|u_i(U)|$, $i = 1, 2, 3$ for $R = L = C = 1$ (scaled model)

Blow-up of some nodal potentials for certain ω **!**

resonant frequencies **=** $\omega \in \{\omega \in \mathbb{R}\colon \mathbf{A}(\omega) \text{ singular}\}$

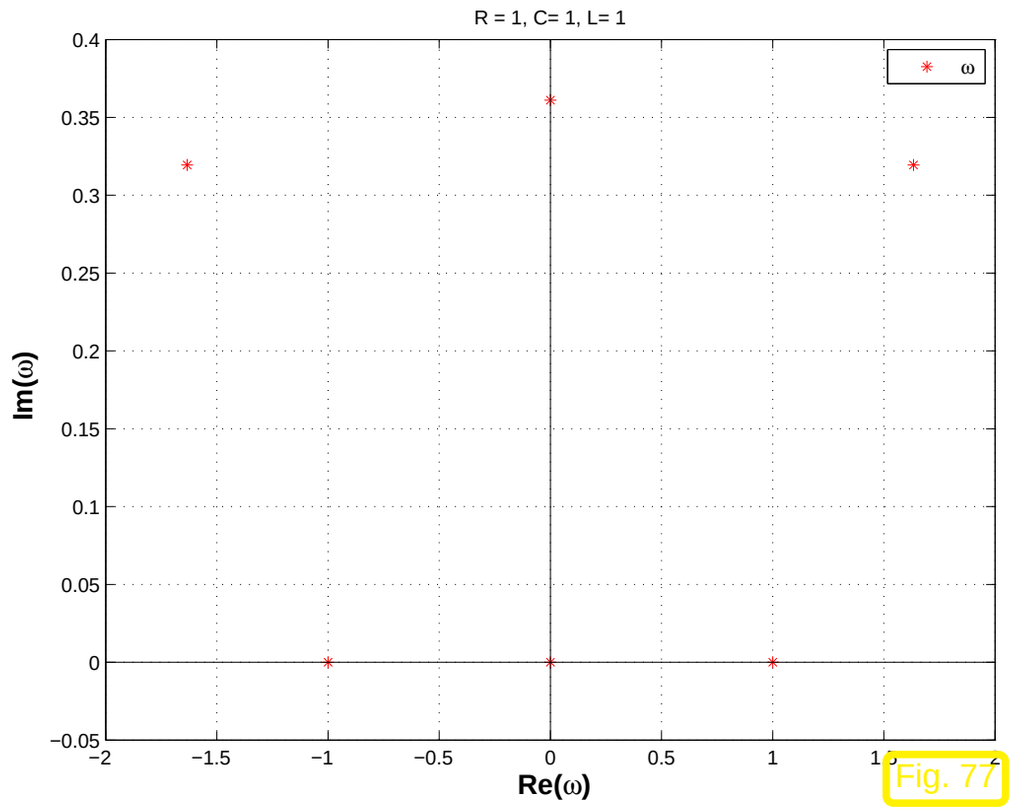


Fig. 77

◁ resonant frequencies for circuit from Fig. 6.0.2 (including decaying modes with $\mathrm{Im}(\omega) > 0$)

◇

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):

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \quad , \quad \mathbf{A} \in \mathbb{C}^{n,n} \; . \tag{6.0.8}$$

R. Hiptmair

rev 38355, November 11, 2011

$$\mathbf{A} = \mathbf{S}\underbrace{\begin{pmatrix}\lambda_1 & & \\ & \ddots & \\ & & \lambda_n\end{pmatrix}}_{=:\mathbf{D}}\mathbf{S}^{-1}\,, \quad \mathbf{S} \in \mathbb{C}^{n,n} \text{ regular} \quad \Longrightarrow \quad \left(\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \quad \overset{\mathbf{z}=\mathbf{S}^{-1}\mathbf{y}}{\longleftrightarrow} \quad \dot{\mathbf{z}} = \mathbf{D}\mathbf{z}\right).$$

$\diamond$

6.1 Theory of eigenvalue problems [48, Ch. 7], [27, Ch. 9], [51, Sect. 1.7]

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 $\mathbf{x} \neq 0$, $\lambda \in \mathbb{C}$

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{B}\mathbf{x} \quad \Leftrightarrow \quad \mathbf{B}^{-1}\mathbf{A}\mathbf{x} = \lambda \mathbf{x} \; . \tag{6.1.10}$$

$\mathbf{x} \hat{=}$ generalized eigenvector, $\lambda \hat{=}$ generalized eigenvalue

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 $\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{D}$

Similar functionality for generalized EVP $\mathbf{A}\mathbf{x} = \lambda \mathbf{B}\mathbf{x}$, $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{n,n}$

`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 $\mathbf{AV} = \mathbf{BVD}$

Remark 6.2.4 (Computational effort for eigenvalue computations).

Computational effort (#elementary operations) for `eig()`:

eigenvalues & eigenvectors of $\mathbf{A} \in \mathbb{K}^{n,n}$	$\sim 25n^3 + O(n^2)$
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 tridiagonal $\mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n}$	$\sim 30n^2 + O(n)$

$\Big\}$ $O(n^3)$!

△

Example 6.2.5 (Runtimes of `eig`).

```
A = rand(500,500); B = A'*A; C = gallery('tridiag',500,1,3,1);
```

➤
- $\mathbf{A}$ generic dense matrix
- $\mathbf{B}$ symmetric (s.p.d. → Def. 2.7.9) matrix
- $\mathbf{C}$ s.p.d. *tridiagonal* matrix

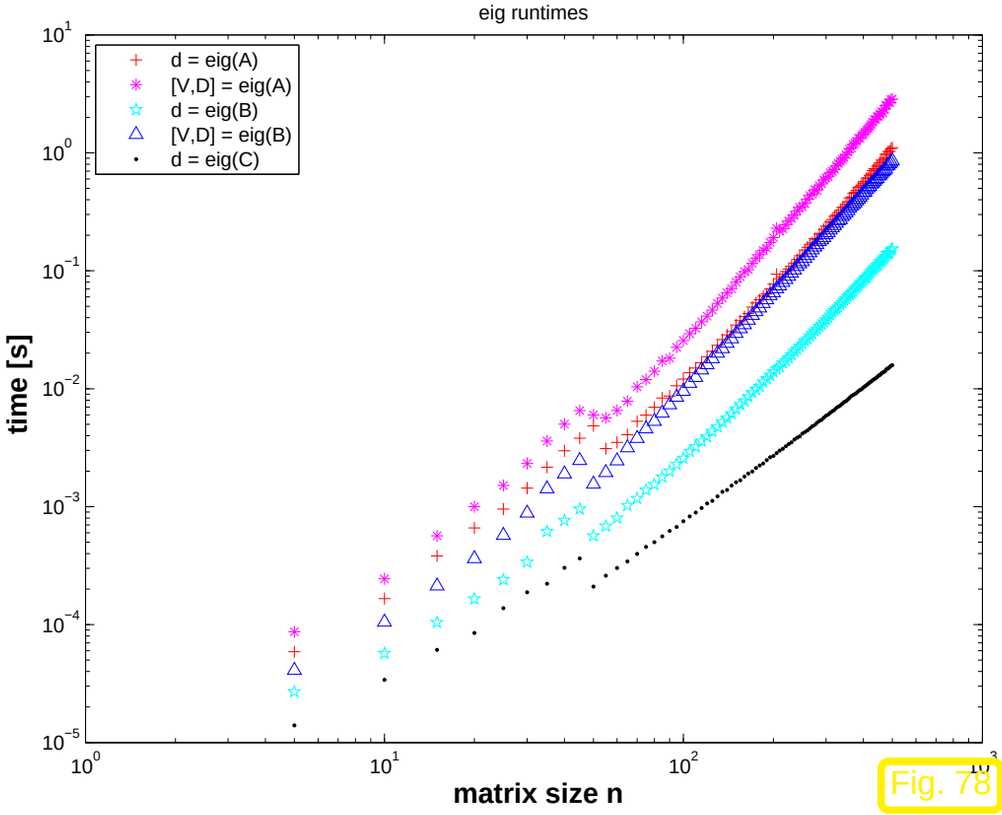


Fig. 78

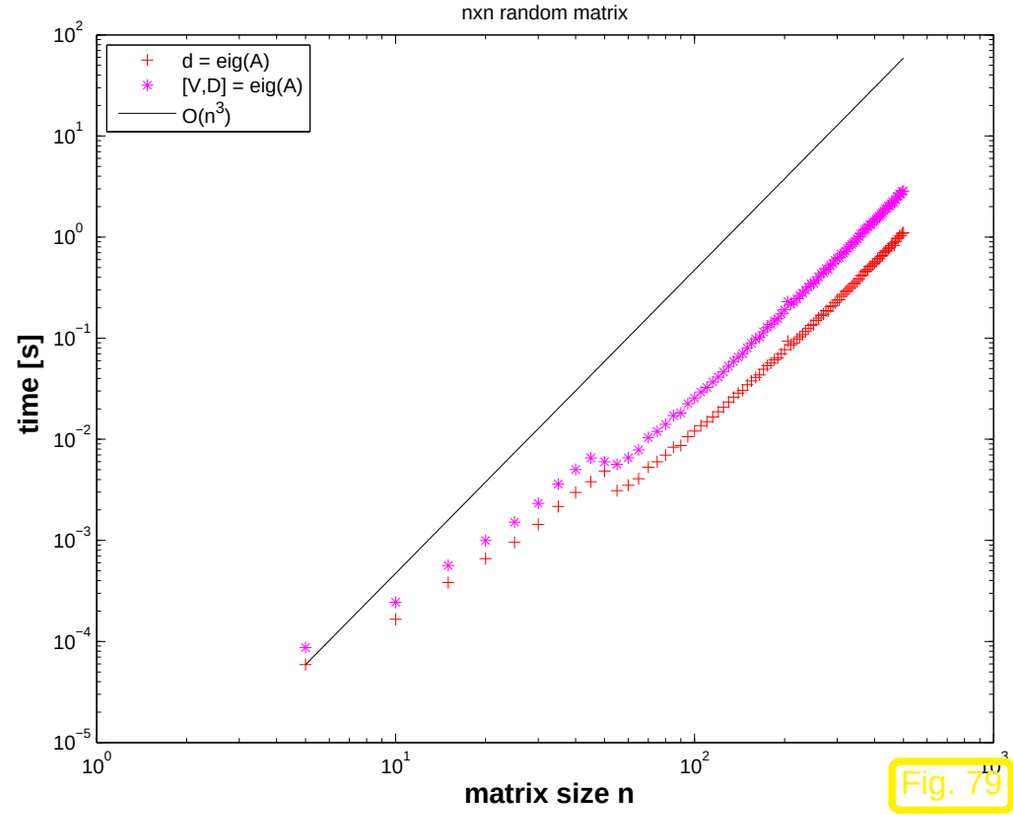


Fig. 79

R. Hiptmair

rev 38355, November 4, 2011

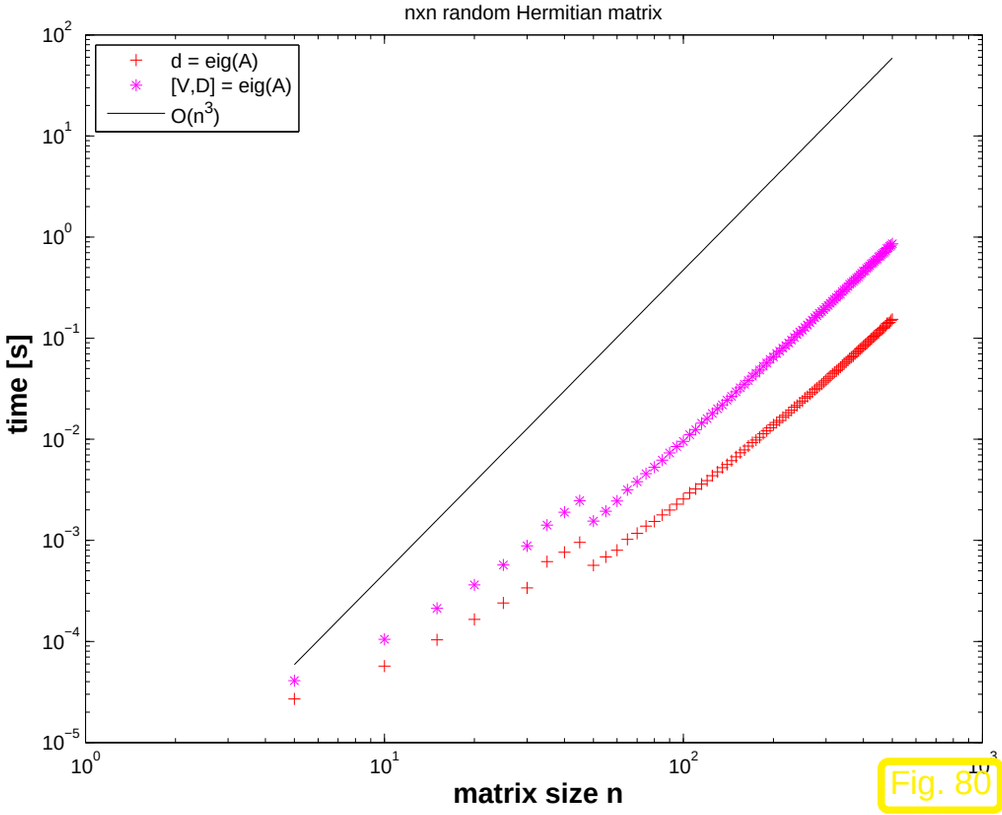


Fig. 80

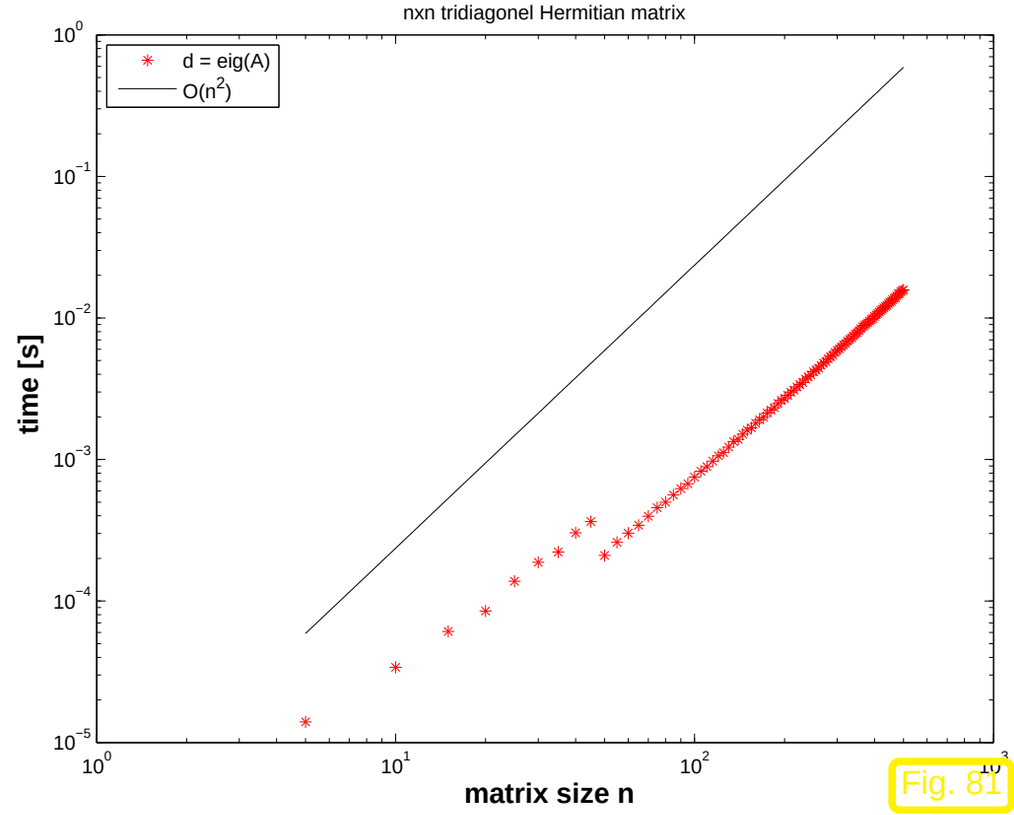


Fig. 81

◇

R. Hiptmair

rev 38355, November 4, 2011

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 Δt, and then

❶ either follows each of ℓ links on a page with probabilty $1/\ell$.

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 $1 - d$.

Question: Fraction of time spent by random surfer on i-th page (= page rank $x_i \in [0,1]$)

Method: Stochastic simulation $\triangleright$

Code 6.3.2: stochastic page rank simulation

```
function prstochsim(Nhops)
% Load web graph data stored in N × N-matrix G
load harvard500.mat;
```

R. Hiptmair

rev 38355, November 4, 2011

```
N = size(G,1); d = 0.15;
count = zeros(1,N); cp = 1;
figure('position',[0 0 1200 1000]); pause;
for n=1:Nhops
  % Find links from current page cp
  idx = find(G(:,cp)); l = size(idx,1); rn = rand(); %
  % If no links, jump to any other pages with equal probability
  if (isempty(idx)),  cp = floor(rn*N)+1;
  % With probabilty d jump to any other page
  elseif (rn < d), cp = floor(rn/d*N)+1;
  % Follow outgoing links with equal probabilty
  else cp = idx(floor((rn-d)/(1-d)*l)+1,1);
  end
  count(cp) = count(cp) + 1;
  plot(1:N,count/n,'r.'); axis([0 N+1 0 0.1]);
  xlabel('{\bf harvard500: no. of page}','fontsize',14);
  ylabel('{\bf page rank}','fontsize',14);
  title(sprintf('{\\bf page rank, harvard500: %d
    hops}',n),'fontsize',14);
  drawnow;
end
```


R. Hiptmair

rev 38355, November 4, 2011

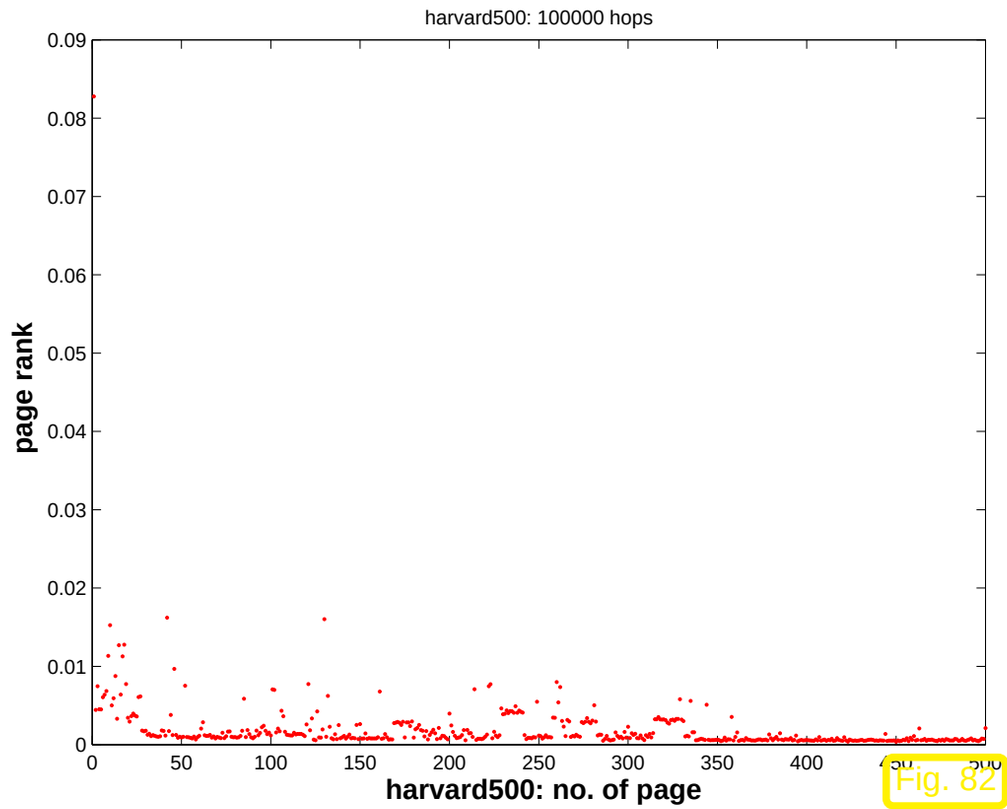


Fig. 82

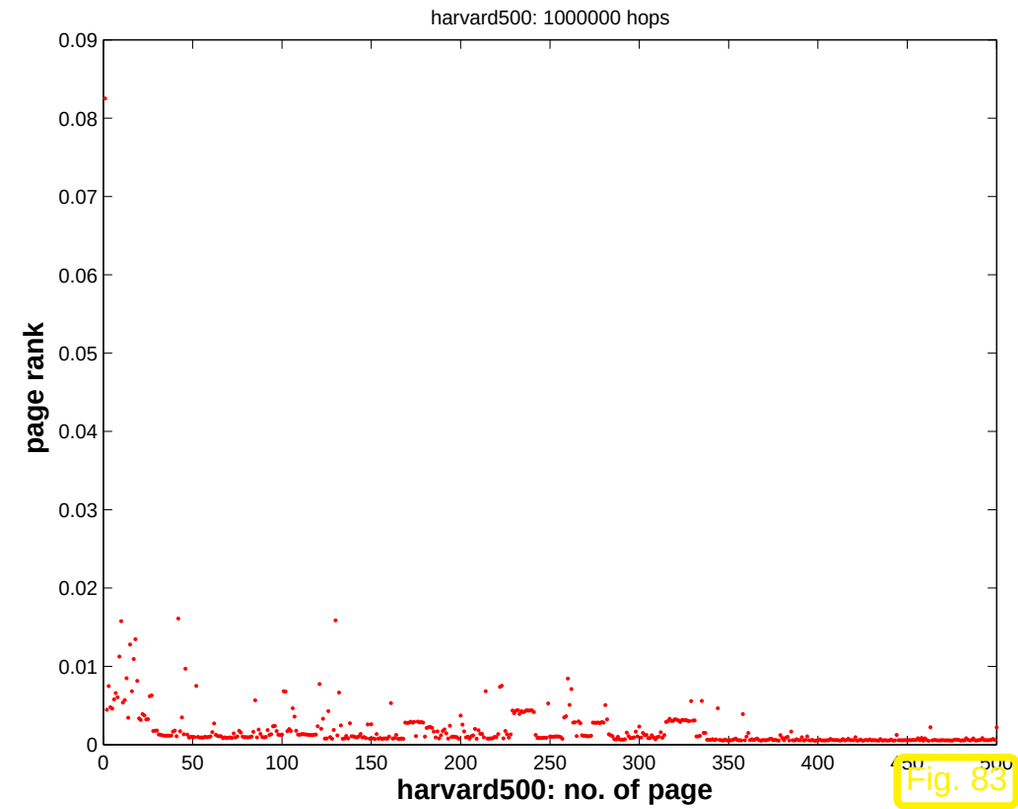


Fig. 83

R. Hiptmair

rev 38355, November 4, 2011

- Numbering of pages $1,\dots,N$, $\ell_i \hat{=}$ number of links from page i
- $N \times N$-matrix of transition probabilities page $j \to$ page i: $\mathbf{A} = (a_{ij})_{i,j=1}^N \in \mathbb{R}^{N,N}$

$$a_{ij} \in [0,1] \quad \hat{=} \text{ probability to jump from page } j \text{ to page } i.$$

$$\Rightarrow \quad \sum_{i=1}^N a_{ij} = 1 \; . \tag{6.3.3}$$

Code 6.3.7: tracking fractions of many surfers

```
function prpowitsim(d,Nsteps)
% MATLAB way of specifying Default arguments
if (nargin < 2), Nsteps = 5; end
if (nargin < 1), d = 0.15; end
% load connectivity matrix and build transition matrix
load harvard500.mat; A = prbuildA(G,d);
N = size(A,1); x = ones(N,1)/N;

figure('position',[0 0 1200 1000]);
plot(1:N,x,'r+'); axis([0 N+1 0 0.1]);
% Plain power iteration for stochastic matrix A
for l=1:Nsteps
  pause; x = A*x; plot(1:N,x,'r+'); axis([0 N+1 0 0.1]);
  title(sprintf('{\\bf step %d}',l),'fontsize',14);
  xlabel('{\bf harvard500: no. of page}','fontsize',14);
  ylabel('{\bf page rank}','fontsize',14); drawnow;
end
```

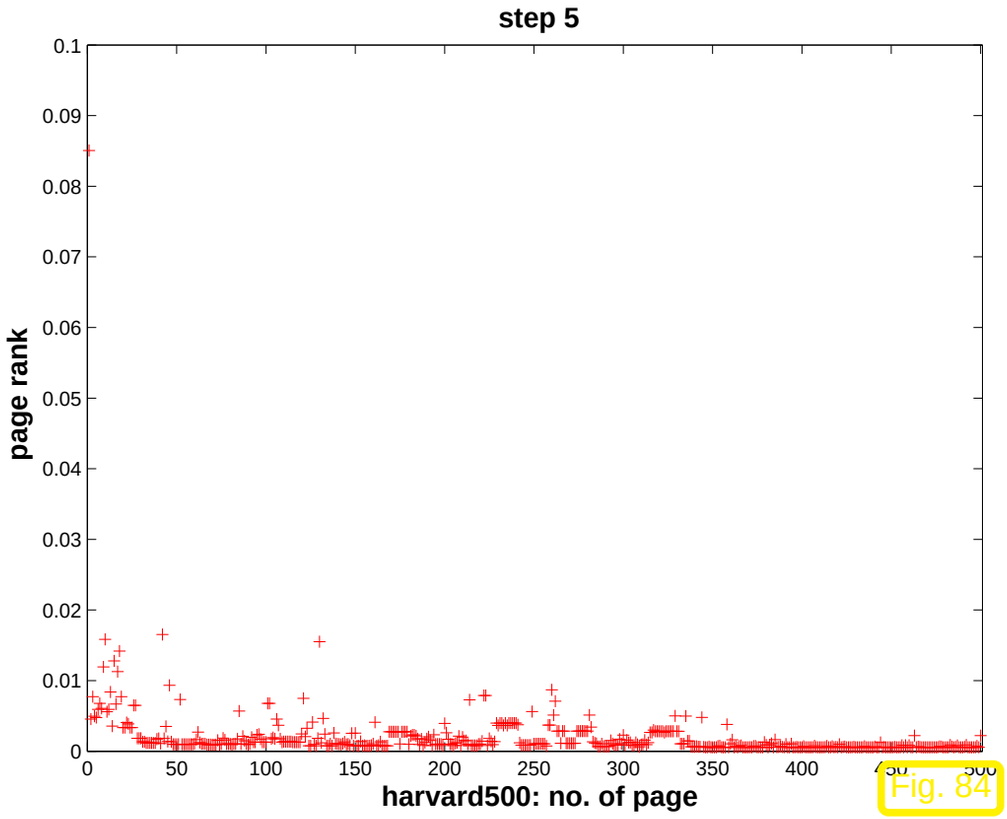


Fig. 84

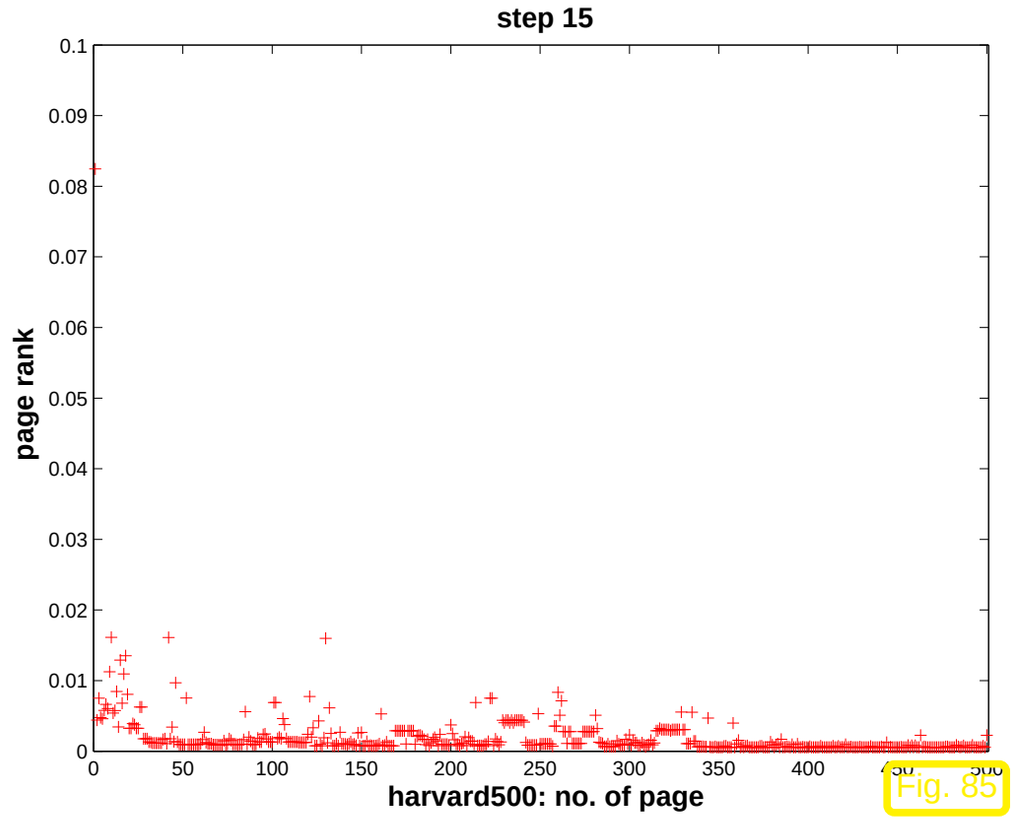


Fig. 85

Comparison:

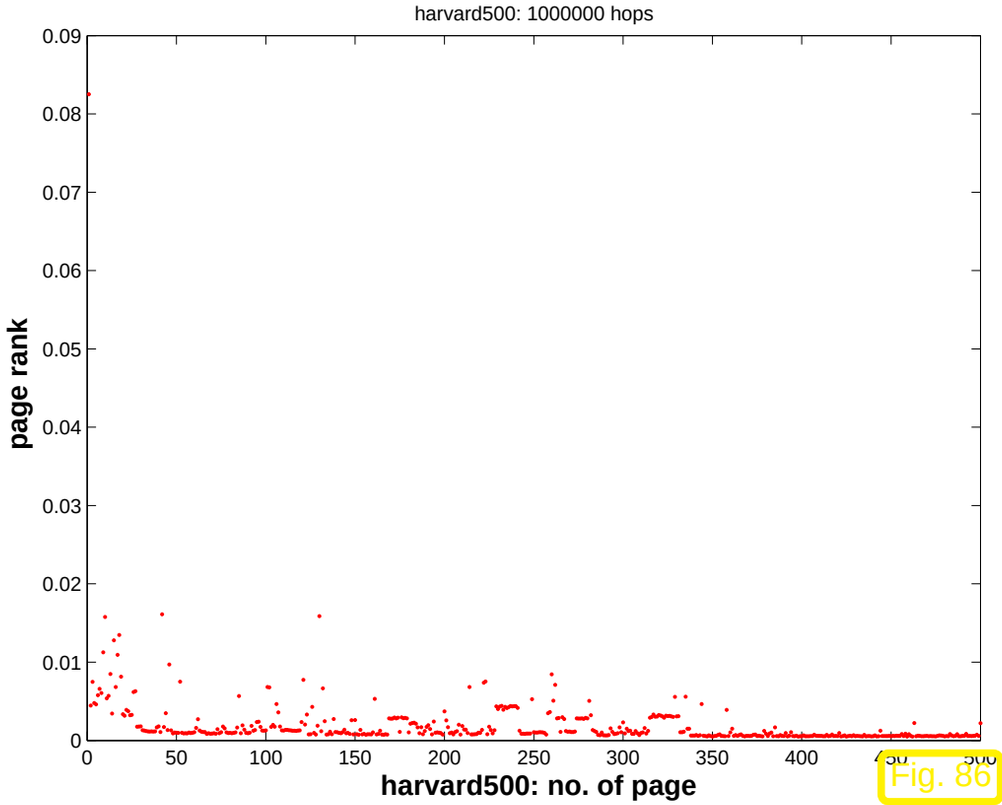


Fig. 86

Single surfer stochastic simulation

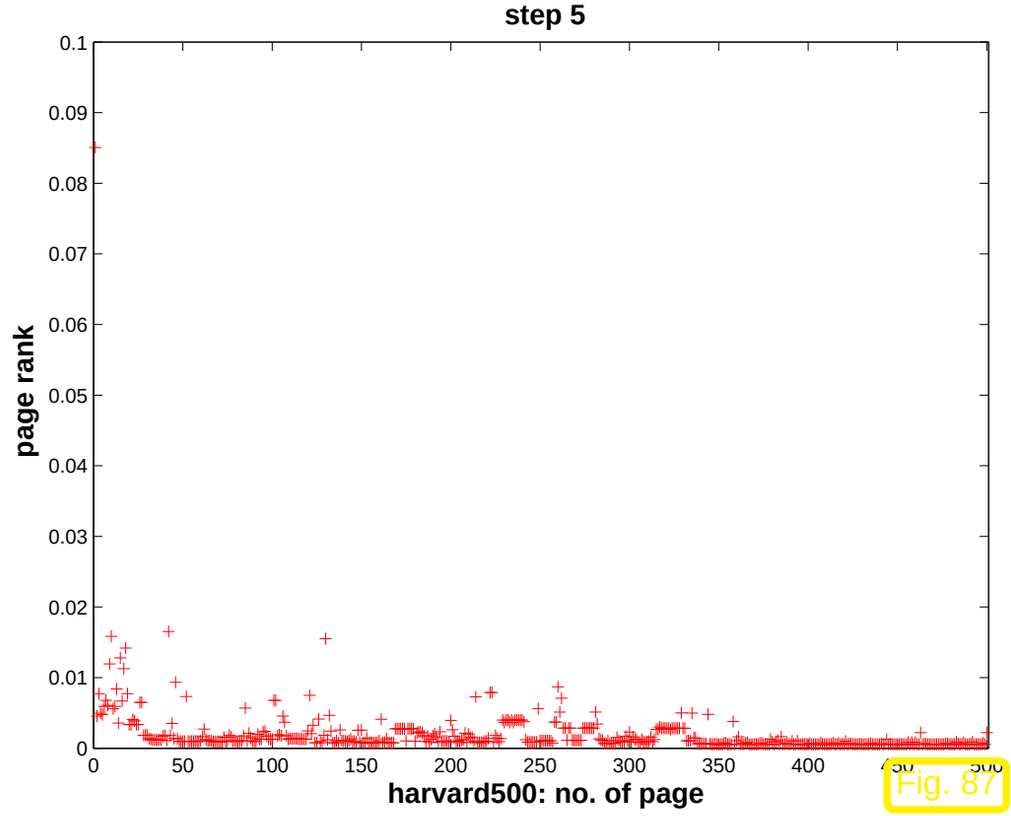


Fig. 87

Power method, Code 6.3.6

R. Hiptmair

rev 38355, November 4, 2011

Code 6.3.9: computing page rank vector $\mathbf{r}$ via `eig`

```
function prevp
load harvard500.mat; d = 0.15;
[V,D] = eig(prbuildA(G,d));

figure; r = V(:,1); N = length(r);
plot(1:N,r/sum(r),'m.'); axis([0 N+1 0 0.1]);
xlabel('{\bf harvard500: no. of page}','fontsize',14);
ylabel('{\bf entry of r-vector}','fontsize',14);
title('harvard 500: Perron-Frobenius vector');
print -depsc2 '../PICTURES/prevp.eps';
```

Plot of entries of *unique* vector $\mathbf{r} \in \mathbb{R}^N$ with

$$0 \leq (\mathbf{r})_i \leq 1 \,, \quad \|\mathbf{r}\|_1 = 1 \,, \quad \boxed{\mathbf{A}\mathbf{r} = \mathbf{r}} \,.$$

Inefficient implementation!

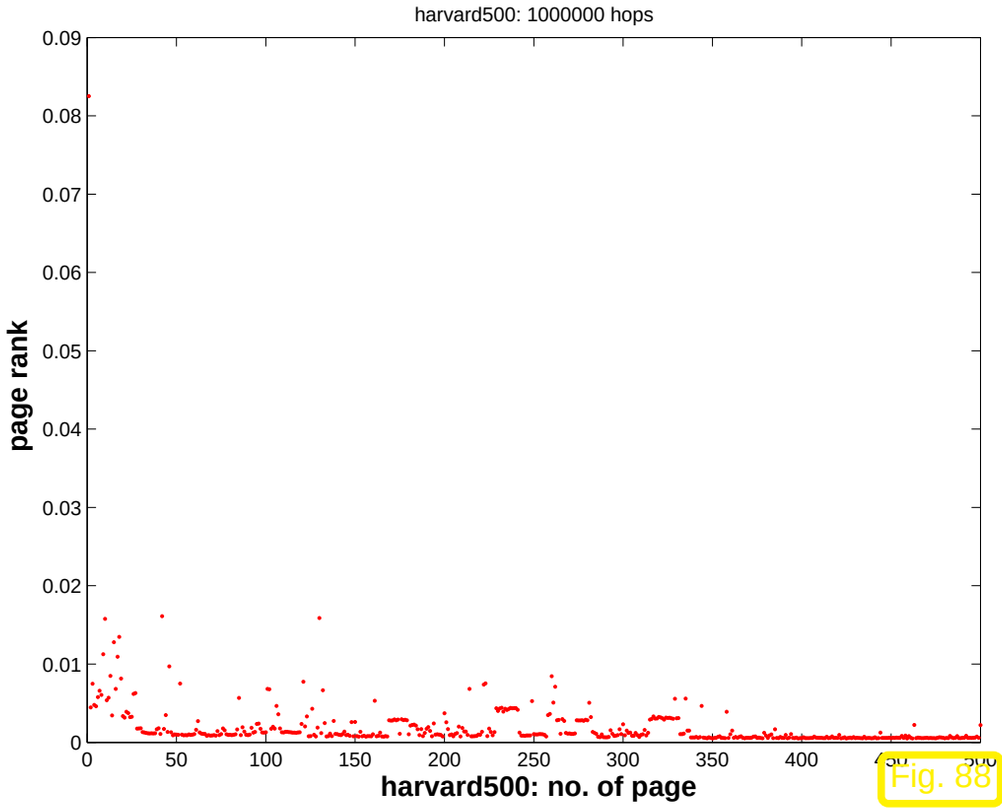


Fig. 88

stochastic simulation

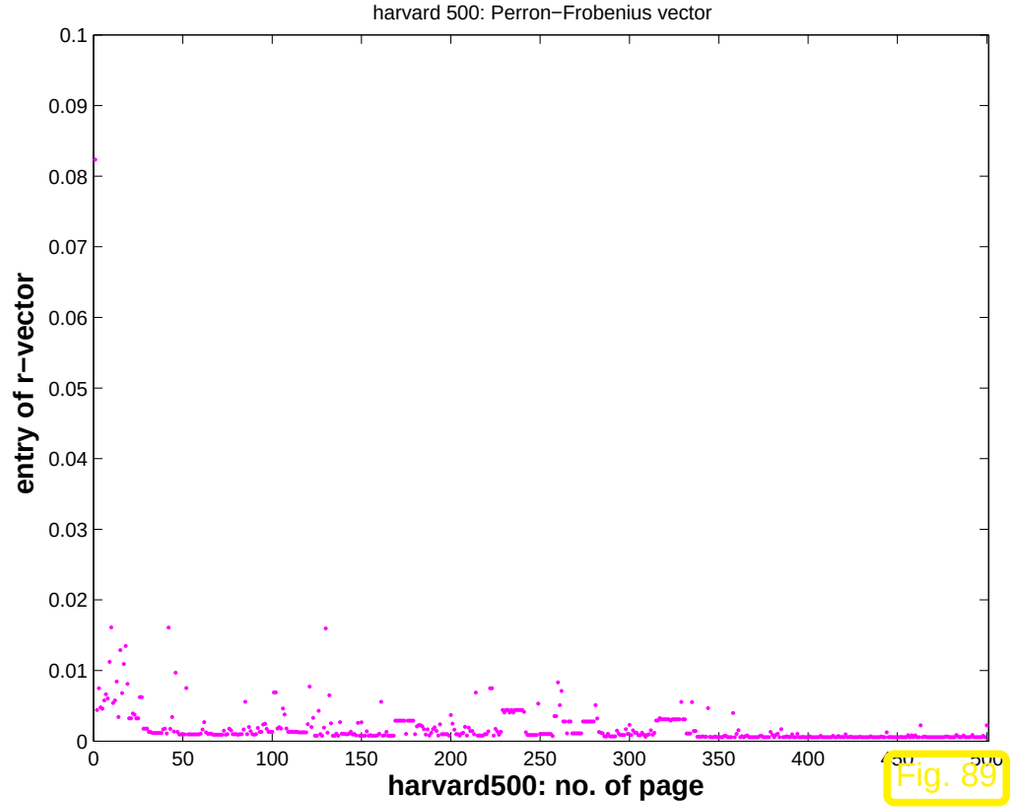


Fig. 89

eigenvector computation

$\mathbf{A} \hat{=}$ page rank transition probability matrix, see Code 6.3.4, $d = 0.15$, harvard500 example.

Errors: ▷

$$\left\| \mathbf{A}^k \mathbf{x}_0 - \mathbf{r} \right\|_1 \quad ,$$

with $\mathbf{x}_0 = \mathbf{1}/N$, $N = 500$.

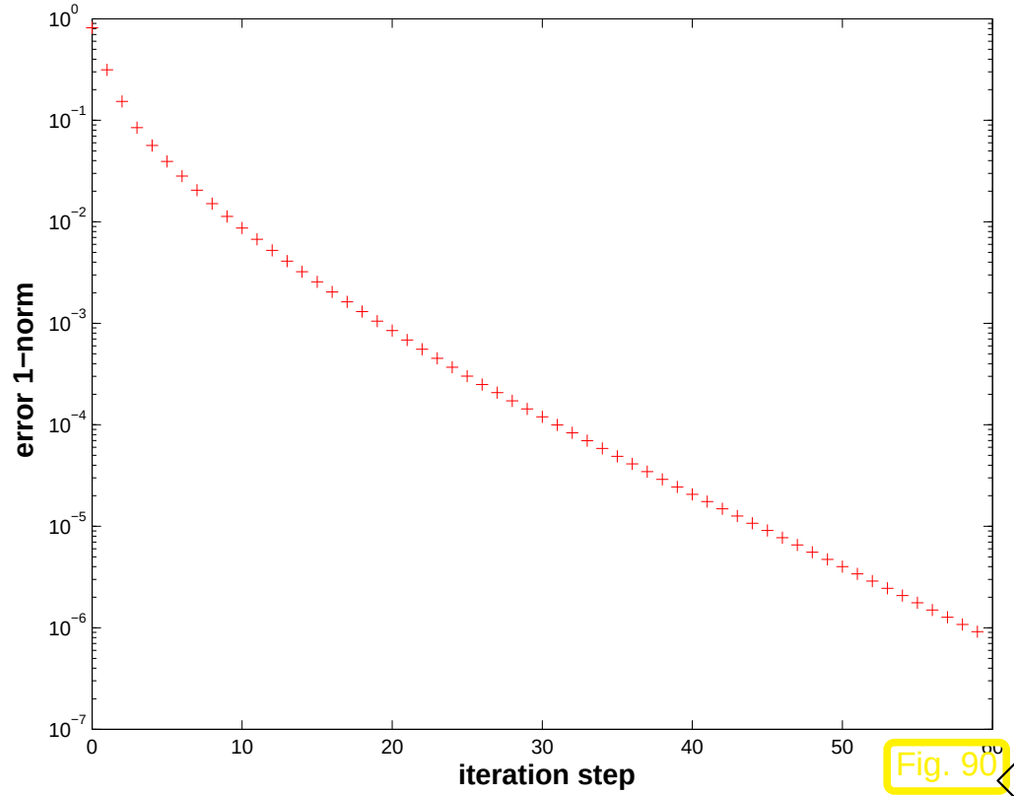
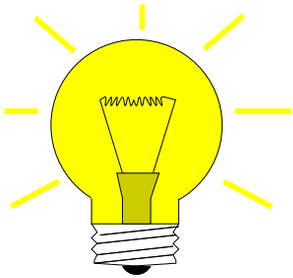


Fig. 90 ◇

R. Hiptmair

rev 38355, November 4, 2011

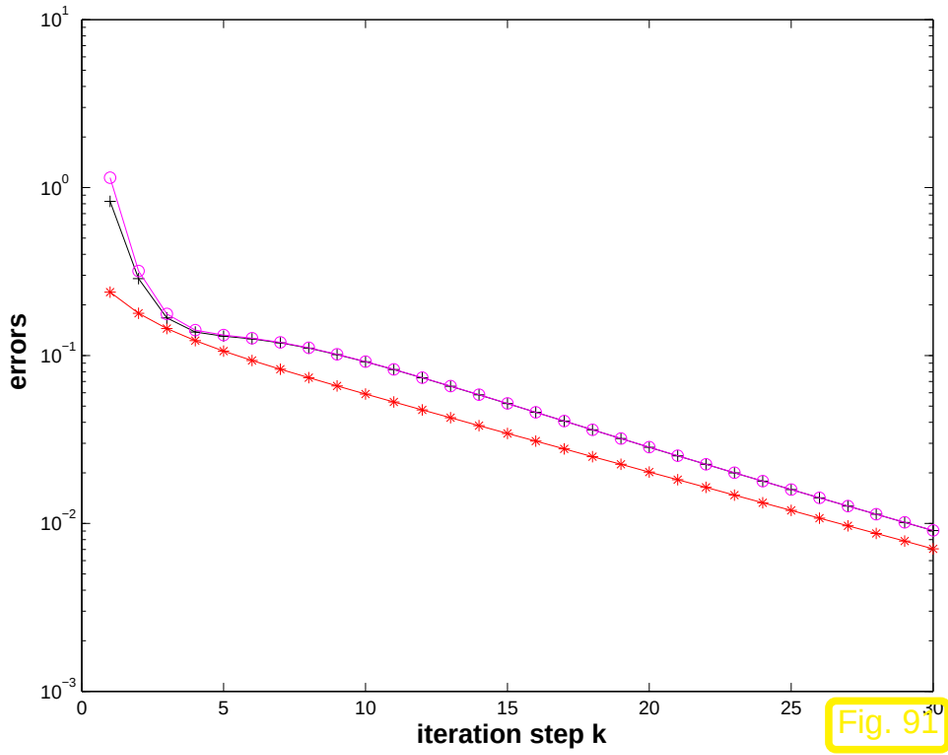
Task: given $\mathbf{A} \in \mathbb{K}^{n,n}$, find largest (in modulus) eigenvalue of $\mathbf{A}$ and (an) associated eigenvector.



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




```
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 $\left\| \frac{\mathbf{z}^{(k)}}{\left\|\mathbf{z}^{(k)}\right\|} - (\mathbf{S})_{:,10} \right\|$

(Note: $(\mathbf{S})_{:,10} \hat{=}$ eigenvector for eigenvalue 10)

$\mathbf{z}^{(0)}$ = random vector

◇

R. Hiptmair

rev 38355, November 4, 2011

Suggests direct power method (*ger.:* Potenzmethode): iterative method ($\rightarrow$ Sect. 4.1)

initial guess: $\mathbf{z}^{(0)}$ "arbitrary" ,

next iterate: $\mathbf{w} := \mathbf{A}\mathbf{z}^{(k-1)}$, $\mathbf{z}^{(k)} := \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$, $k = 1, 2, \ldots$. (6.3.11)

Computational effort: $1\times$ matrix$\times$vector per step ➢ $\boxed{\text{inexpensive for sparse matrices}}$

When (6.3.11) has converged, two common ways to recover $\lambda_{\max}$ $\rightarrow$ [13, Alg. 7.20]

❶ $\mathbf{A}\mathbf{z}^{(k)} \approx \lambda_{\max}\mathbf{z}^{(k)}$ ➢ $|\lambda_n| \approx \dfrac{\left\|\mathbf{A}\mathbf{z}^{(k)}\right\|}{\left\|\mathbf{z}^{(k)}\right\|}$ (modulus only!)

❷ $\lambda_{\max} \approx \operatorname*{argmin}_{\theta\in\mathbb{R}} \left\|\mathbf{A}\mathbf{z}^{(k)} - \theta\mathbf{z}^{(k)}\right\|_2^2$ ➢ $\lambda_{\max} \approx \dfrac{(\mathbf{z}^{(k)})^H\mathbf{A}\mathbf{z}^{(k)}}{\left\|\mathbf{z}^{(k)}\right\|_2^2}$.

Definition 6.3.15. *For* $\mathbf{A} \in \mathbb{K}^{n,n}$, $\mathbf{u} \in \mathbb{K}^n$ *the Rayleigh quotient is defined by*

$$\rho_{\mathbf{A}}(\mathbf{u}) := \frac{\mathbf{u}^H\mathbf{A}\mathbf{u}}{\mathbf{u}^H\mathbf{u}} .$$

Example 6.3.17 (Direct power method). $\rightarrow$ Ex. 6.3.17 cnt'd

R. Hiptmair

rev 38355, November 4, 2011

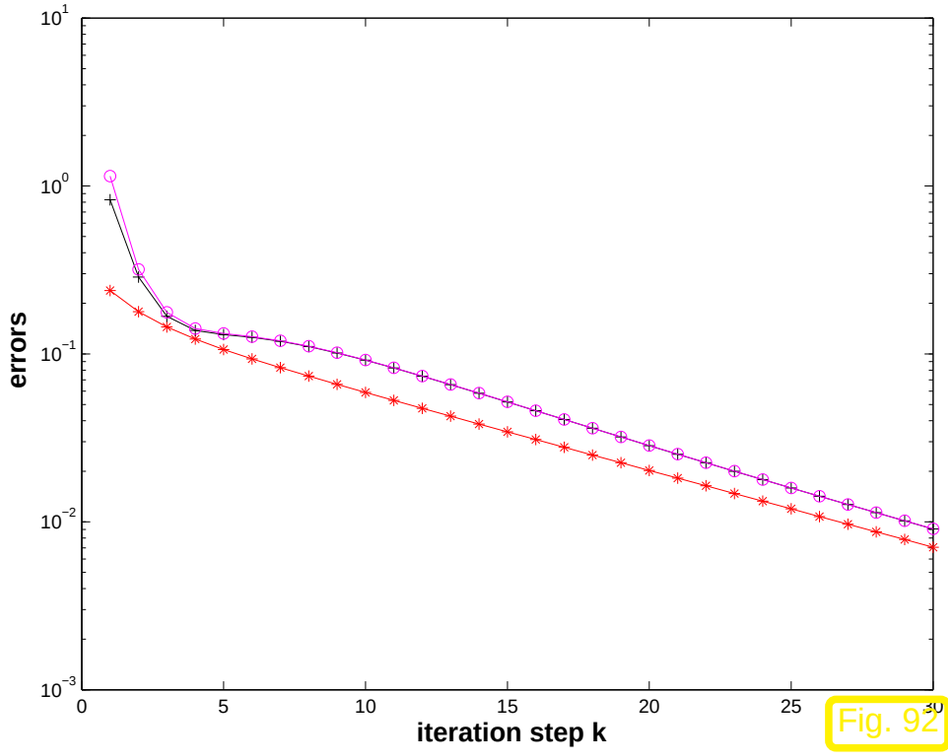


Fig. 92

```
n = length(d);
S = triu(diag(n:-1:1,0)+...
    ones(n,n));
A = S*diag(d,0)*inv(S);
```

◁ `d = (1:10)';`

o : error $|\lambda_n - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})|$

$*$: error norm $\left\|\mathbf{z}^{(k)} - \mathbf{s}_{\cdot,n}\right\|$

$+$: $\left|\lambda_n - \frac{\left\|\mathbf{A}\mathbf{z}^{(k-1)}\right\|_2}{\left\|\mathbf{z}^{(k-1)}\right\|_2}\right|$

$\mathbf{z}^{(0)}$ = random vector

Test matrices:

① `d=(1:10)';` ➢ $|\lambda_{n-1}| : |\lambda_n| = 0.9$

② `d = [ones(9,1); 2];` ➢ $|\lambda_{n-1}| : |\lambda_n| = 0.5$

③ `d = 1-2.^(-(1:0.5:5)');` ➢ $|\lambda_{n-1}| : |\lambda_n| = 0.9866$

$$\rho_{\mathrm{EV}}^{(k)} := \frac{\left\| \mathbf{z}^{(k)} - \mathbf{s}_{\cdot,n} \right\|}{\left\| \mathbf{z}^{(k-1)} - \mathbf{s}_{\cdot,n} \right\|} \,,$$

$$\rho_{\mathrm{EW}}^{(k)} := \frac{|\rho_{\mathbf{A}}(\mathbf{z}^{(k)}) - \lambda_n|}{|\rho_{\mathbf{A}}(\mathbf{z}^{(k-1)}) - \lambda_n|} \,.$$

	①		②		③	
k	$\rho_{\mathrm{EV}}^{(k)}$	$\rho_{\mathrm{EW}}^{(k)}$	$\rho_{\mathrm{EV}}^{(k)}$	$\rho_{\mathrm{EW}}^{(k)}$	$\rho_{\mathrm{EV}}^{(k)}$	$\rho_{\mathrm{EW}}^{(k)}$
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

◇

R. Hiptmair

rev 38355, November 4, 2011

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 λ_n normalized according to $\|\mathbf{y}\|_2 = \|\mathbf{v}\|_2 = 1$. Then there is convergence

$$\left\|\mathbf{A}\mathbf{z}^{(k)}\right\|_2 \rightarrow \lambda_n \quad , \quad \mathbf{z}^{(k)} \rightarrow \pm\mathbf{v} \quad \textit{linearly with rate} \quad \frac{|\lambda_{n-1}|}{|\lambda_n|} \, ,$$

where $\mathbf{z}^{(k)}$ are the iterates of the direct power iteration and $\mathbf{y}^H\mathbf{z}^{(0)} \neq 0$ is assumed.

R. Hiptmair

rev 38355, November 10, 2011

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 $\mathbf{P} \in \{0, \ldots, 255\}^{m,n}$, $m, n \in \mathbb{N}$
($(\mathbf{P})_{ij} \leftrightarrow$ pixel, $0 \hat{=}$ black, $255 \hat{=}$ white)

Loading and displaying images in MATLAB

Code 6.3.23: loading and displaying an image

```
M = imread('eth.pbm');
[m,n] = size(M);
fprintf('%dx%d grey scale pixel image\n',m,n);
figure; image(M); title('ETH view');
col = [0:1/215:1]'*[1,1,1]; colormap(col);
```

(Fuzzy) task: Local segmentation

Find connected patches of image of the same shade/color

Next: Statement of (rigorously defined) problem, *cf.* Sect. 2.5.2:

Preparation: Numbering of pixels $1 \ldots, mn$, e.g, lexicographic numbering:

- pixel set $\mathcal{V} := \{1. \ldots, nm\}$
- indexing: index(pixel$_{i,j}$) $= (i-1)n + j$

✎ notation: $p_k := (\mathbf{P})_{ij}$, if $k =$ index(pixel$_{i,j}$) $= (i-1)n + j$, $k = 1, \ldots, N := mn$

R. Hiptmair

rev 38355, November 10, 2011

Local similarity matrix:

$$\mathbf{W} \in \mathbb{R}^{N,N} \;, \quad N := mn \;, \qquad (6.3.24)$$

$$(\mathbf{W})_{ij} = \begin{cases} 0 & \text{, if pixels } i,\, j \text{ not adjacent,} \\ 0 & \text{, if } i = j \;, \\ \sigma(p_i, p_j) & \text{, if pixels } i,\, j \text{ adjacent.} \end{cases}$$

$\leftrightarrow \hat{=}$ adjacent pixels ▷

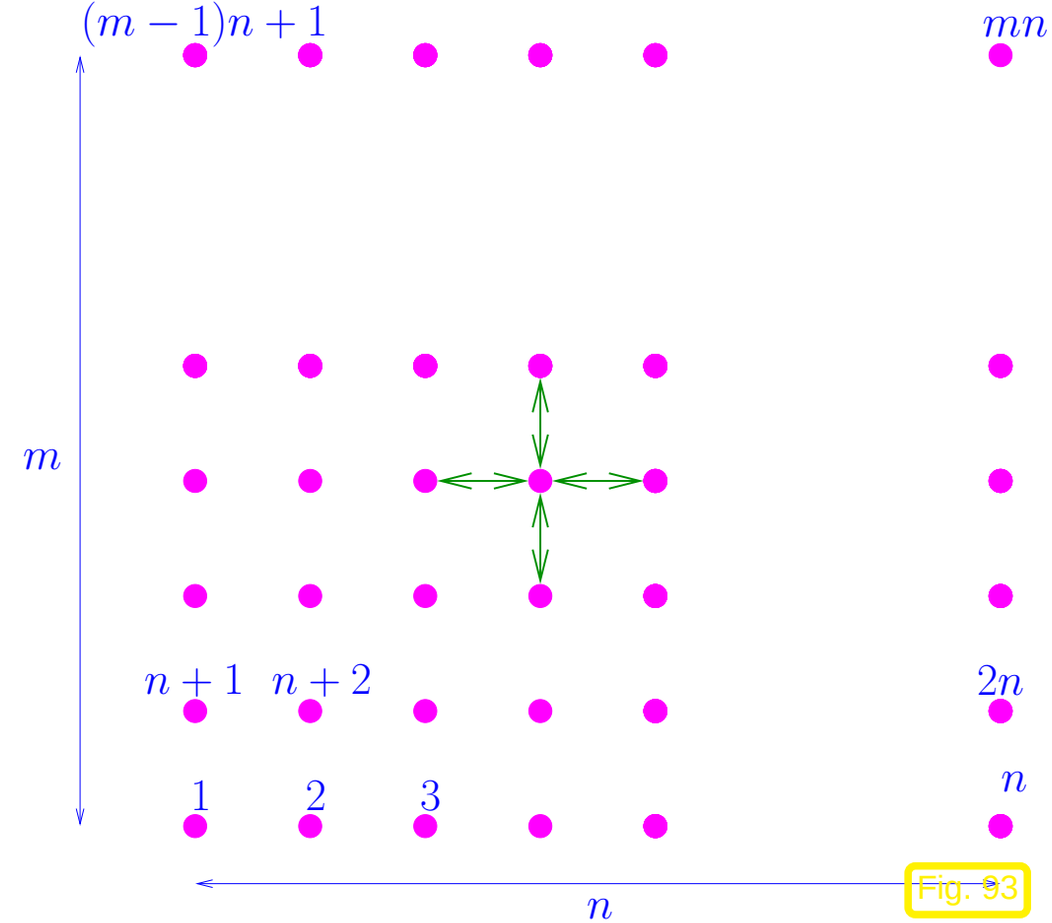


Fig. 93

Similarity function, e.g., with $\alpha > 0$

$$\sigma(x,y) := \exp(-\alpha(x-y)^2) \;, \quad x, y \in \mathbb{R} \;.$$

Lexicographic numbering ▷

Definition 6.3.25 (Normalized cut). *(→ [58, Sect. 2])*

For $\mathcal{X} \subset \mathcal{V}$ define the normalized cut as

$$\mathrm{Ncut}(\mathcal{X}) := \frac{\mathrm{cut}(\mathcal{X})}{\mathrm{weight}(\mathcal{X})} + \frac{\mathrm{cut}(\mathcal{X})}{\mathrm{weight}(\mathcal{V} \setminus \mathcal{X})} \;,$$

with $\quad \mathrm{cut}(\mathcal{X}) := \sum_{i \in \mathcal{X}, j \notin \mathcal{X}} w_{ij} \;, \quad \mathrm{weight}(\mathcal{X}) := \sum_{i \in \mathcal{X}, j \in \mathcal{X}} w_{ij} \;.$

Segmentation problem (rigorous statement):

$$\boxed{\text{find} \quad \mathcal{X}^* \subset \mathcal{V}: \quad \mathcal{X}^* = \operatorname*{argmin}_{\mathcal{X}\subset\mathcal{V}} \mathrm{Ncut}(\mathcal{X})} \quad . \tag{6.3.26}$$

NP-hard combinatorial optimization problem !

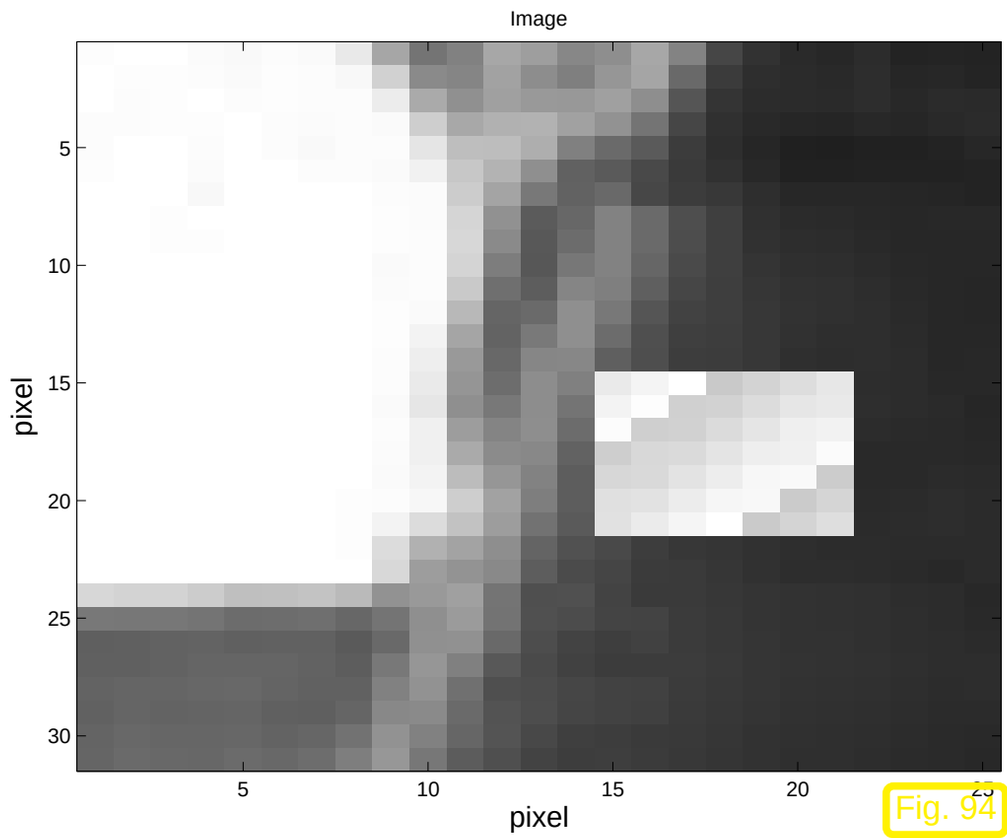


Fig. 94

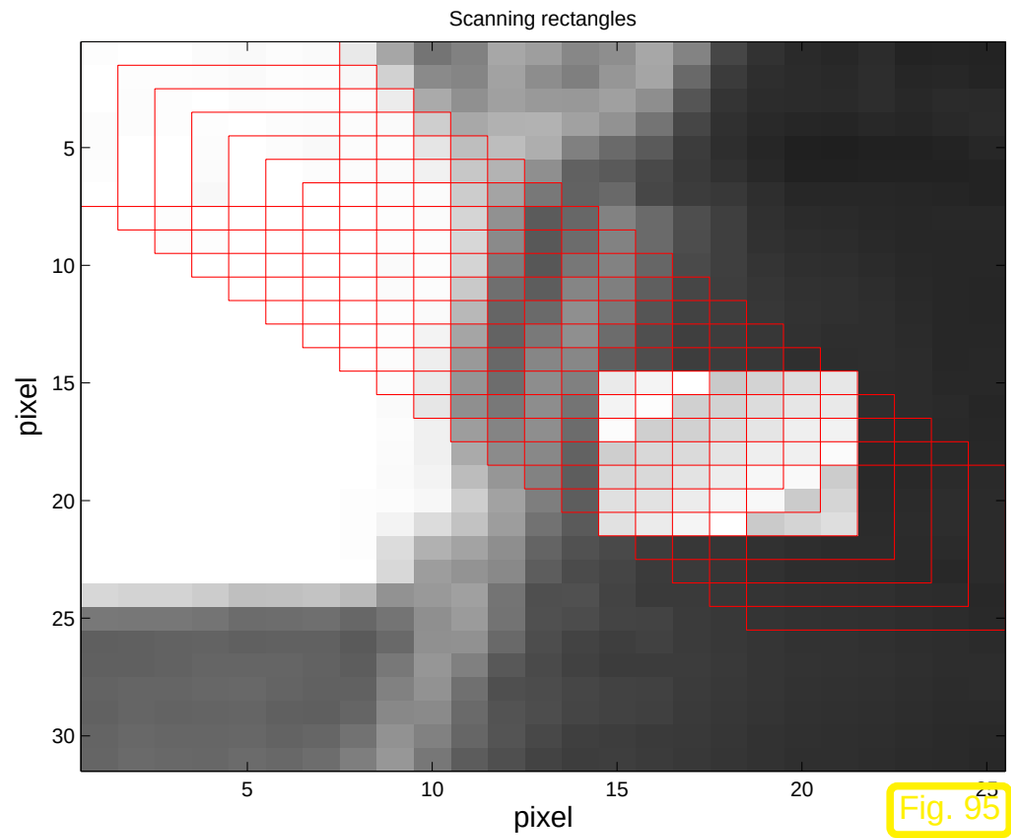


Fig. 95

R. Hiptmair

rev 38355, November 10, 2011

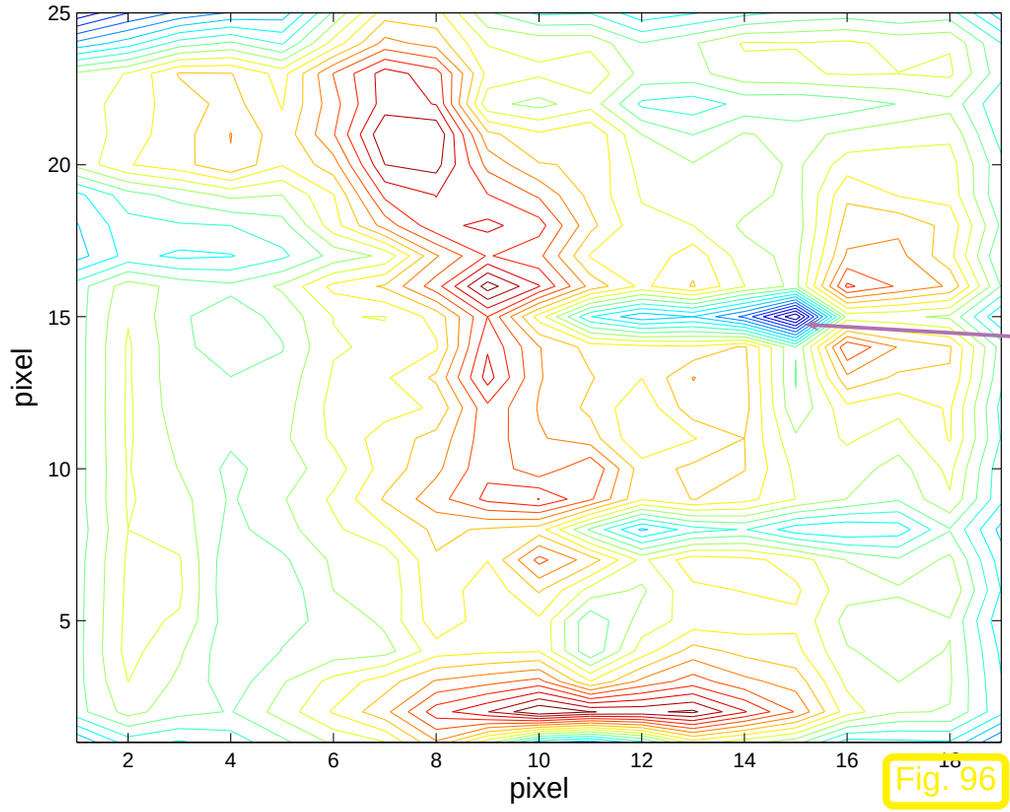


Fig. 96

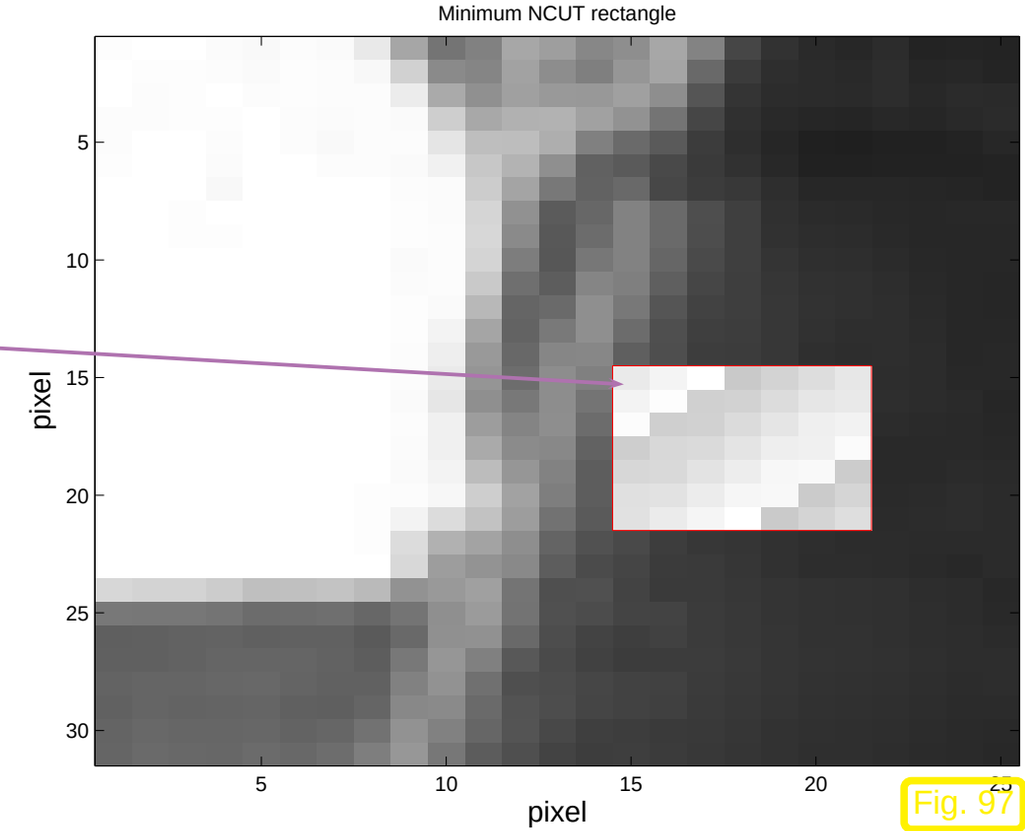


Fig. 97

Equivalent reformulation:

indicator function: $$z : \{1,\dots,N\} \mapsto \{-1,1\}\,, \quad z_i := z(i) = \begin{cases} 1 & \text{, if } i \in \mathcal{X}\,, \\ -1 & \text{, if } i \notin \mathcal{X}\,. \end{cases} \tag{6.3.27}$$

$$\blacktriangleright \quad \mathrm{Ncut}(\mathcal{X}) = \frac{\sum\limits_{z_i>0,z_j<0} -w_{ij}z_iz_j}{\sum\limits_{z_i>0} d_i} + \frac{\sum\limits_{z_i>0,z_j<0} -w_{ij}z_iz_j}{\sum\limits_{z_i<0} d_i}\,, \tag{6.3.28}$$

$$d_i = \sum_{j\in\mathcal{V}} w_{ij} = \mathrm{weight}(\{i\}) \; . \tag{6.3.29}$$

Sparse matrices:

$$\mathbf{D} := \mathrm{diag}(d_1,\dots,d_N) \in \mathbb{R}^{N,N} \; , \quad \mathbf{A} := \mathbf{D} - \mathbf{W} = \mathbf{A}^\top \; . \tag{6.3.30}$$

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*

$$\mathrm{Ncut}(\mathcal{X}) = \frac{\mathbf{y}^\top \mathbf{A}\mathbf{y}}{\mathbf{y}^\top \mathbf{D}\mathbf{y}} \; , \quad \mathbf{y} := (\mathbf{1}+\mathbf{z}) - \beta(\mathbf{1}-\mathbf{z}) \; , \quad \beta := \frac{\sum_{z_i>0} d_i}{\sum_{z_i<0} d_i} \; .$$

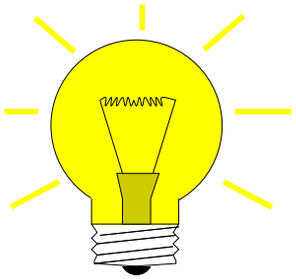
generalized Rayleigh quotient $\rho_{\mathbf{A},\mathbf{D}}(\mathbf{y})$

R. Hiptmair

rev 38355, November 10, 2011

▶ segmentation problem (6.3.26) $\Leftrightarrow$ $\displaystyle\operatorname*{argmin}_{\mathbf{y}\in\{2,-2\beta\}^N,\,\mathbf{1}^\top\mathbf{D}\mathbf{y}=0} \rho_{\mathbf{A},\mathbf{D}}(\mathbf{y}) \; .$ (6.3.35)

still NP-hard



Idea: **Relaxation**

Discrete optimization problem $\rightarrow$ continuous optimization problem

$$(6.3.35) \quad \rightarrow \quad \operatorname*{argmin}_{\mathbf{y}\in\mathbb{R}^N,\,\mathbf{y}\neq 0,\,\mathbf{1}^\top\mathbf{D}\mathbf{y}=0} \rho_{\mathbf{A},\mathbf{D}}(\mathbf{y}) \,. \tag{6.3.36}$$

Theorem 6.3.37 (Rayleigh quotient theorem)*.*

Let $\lambda_1 < \lambda_2 < \cdots < \lambda_m$*,* $m \leq n$*, be the sorted sequence of all (real!) eigenvalues of* $\mathbf{A} = \mathbf{A}^{\mathsf{H}} \in \mathbb{C}^{n,n}$*. Then*

$$\operatorname{Eig}_{\mathbf{A}}(\lambda_1) = \operatorname*{argmin}_{\mathbf{y}\in\mathbb{C}^{n.n}\setminus\{0\}} \rho_{\mathbf{A}}(\mathbf{y}) \quad \textit{and} \quad \operatorname{Eig}_{\mathbf{A}}(\lambda_m) = \operatorname*{argmax}_{\mathbf{y}\in\mathbb{C}^{n.n}\setminus\{0\}} \rho_{\mathbf{A}}(\mathbf{y}) \,.$$

Algorithm 6.3.49 (Binary grayscale image segmentation (outline))*.*

❶ Given similarity function σ 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 $\widehat{\mathbf{A}} := \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} + 2(\mathbf{D}^{1/2}\mathbf{1})(\mathbf{D}^{1/2}\mathbf{1})^\top$.

❸ Set $\mathbf{x}^* = \mathbf{D}^{-1/2}\mathbf{y}^*$ and define the image segment as pixel set

$$\mathcal{X} := \{i \in \{1,\dots,N\} \colon x_i^* > \underbrace{\frac{1}{N}\sum_{i=1}^{N} x_i^*}_{\text{mean value of entries of } \mathbf{x}^*}\} \,. \tag{6.3.50}$$

Code 6.3.51: 1st stage of segmentation of grayscale image

```
% Read image and build matrices, see Code 6.3.31 and (6.3.30)
P = imread('image.pbm'); [m,n] = size(P); [A,D] = imgsegmat(P);
% Build scaling matrics
N = size(A,1); dv = sqrt(spdiags(A,0));
Dm = spdiags(1./dv,[0],N,N); % D^{-1/2}
Dp = spdiags(dv,[0],N,N);    % D^{-1/2}
% Build (densely populated !) matrix Â
c = Dp*ones(N,1); Ah = Dm*A*Dm + 2*c*c';
% Compute and sort eigenvalues; grossly inefficient !
[W,E] = eig(full(Ah)); [ev,idx] = sort(diag(E)); W(:,idx) = W;
% Obtain eigenvector x* belonging to 2nd smallest generalized
% eigenvalue of A and D
x = W(:,1); x = Dm*v;
% Extract segmented image
xs = reshape(x,m,n); Xidx = find(xs>(sum(sum(xs))/(n*m)));
```

1st-stage of segmentation of 31×25 grayscale pixel image (`root.pbm`, red pixels $\hat{=} \mathcal{X}$, $\sigma(x,y) = \exp(-(x-y/10)^2)$)

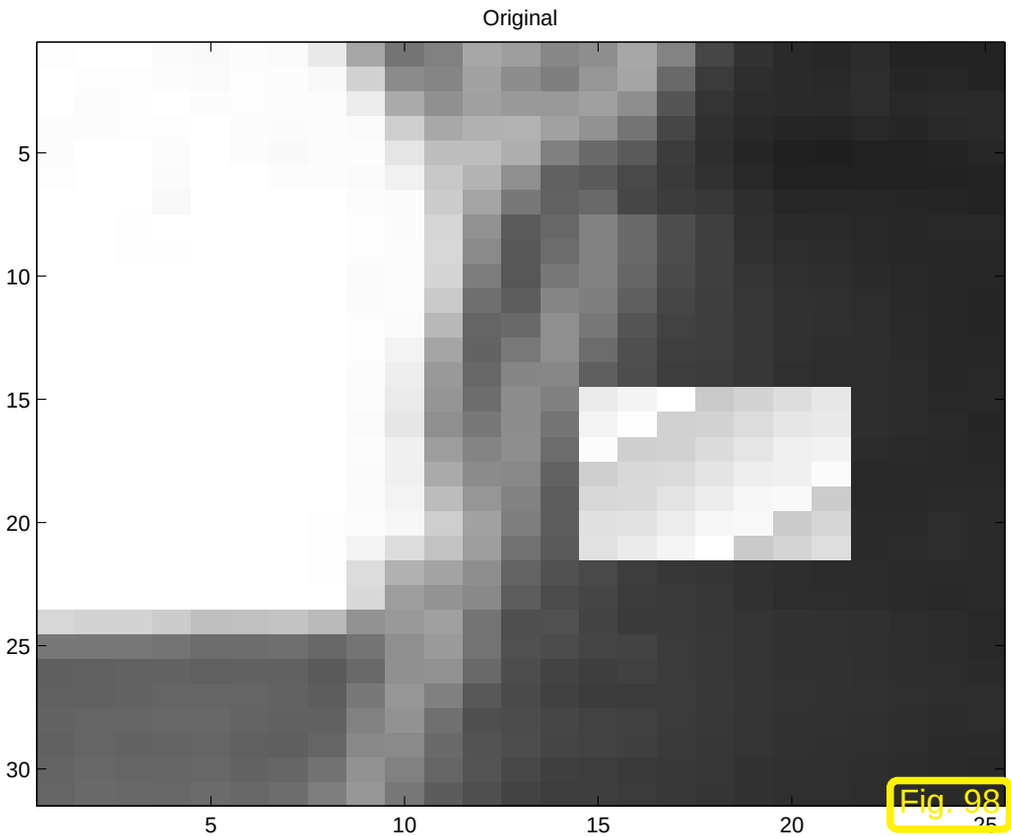


Fig. 98

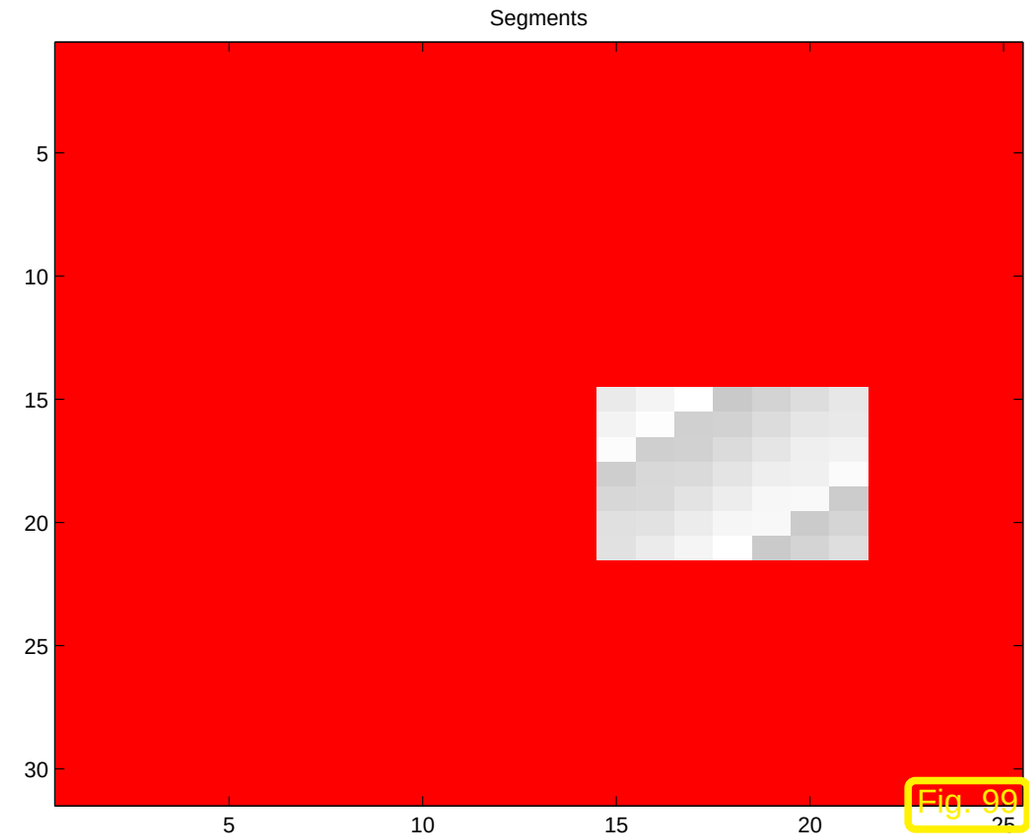


Fig. 99

R. Hiptmair

rev 38355, November 10, 2011

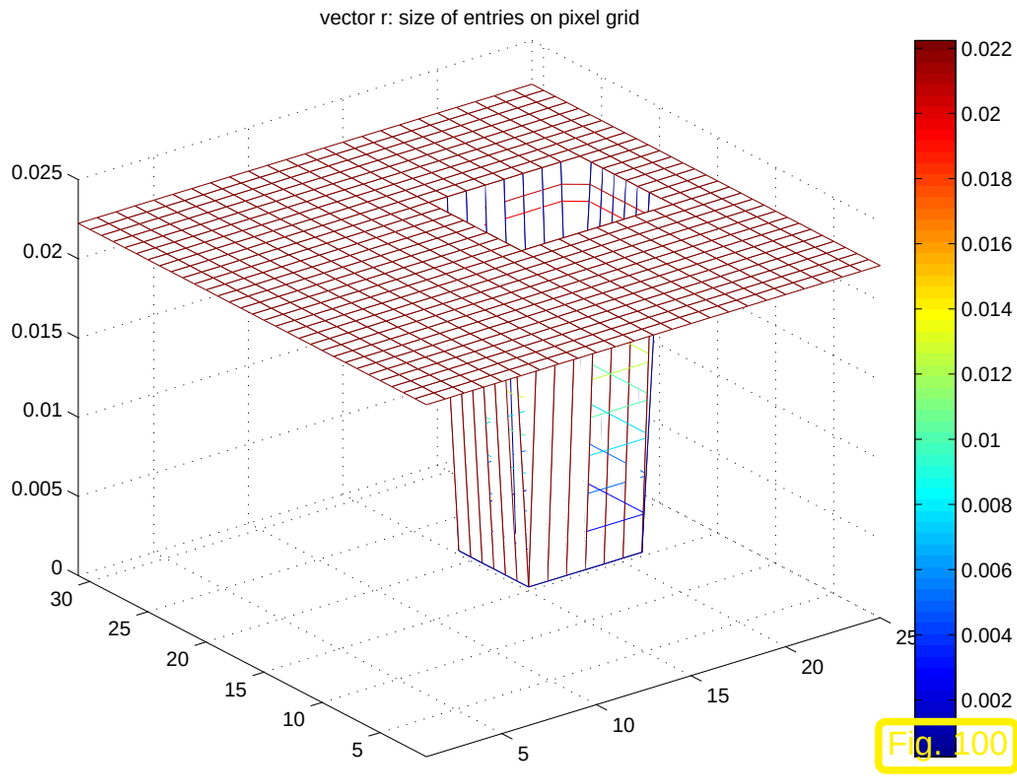


Fig. 100

Image from Fig. 111:

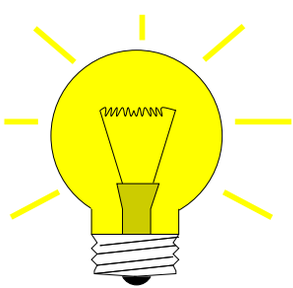
◁ eigenvector $\mathbf{x}^*$ plotted on pixel grid

◇

R. Hiptmair

rev 38355, November 10, 2011

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:

$$\text{Smallest (in modulus) EV of } \mathbf{A} \quad = \quad \left(\text{Largest (in modulus) EV of } \mathbf{A}^{-1}\right)^{-1}$$

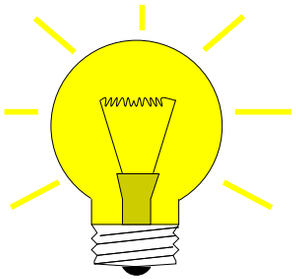
Direct power method (→ Sect. 6.3.1) for $\mathbf{A}^{-1}$ = inverse iteration

Code 6.3.52: inverse iteration for computing $\lambda_{\min}(\mathbf{A})$ and associated eigenvector

```
function [lmin,y] = invit(A,tol)
[L,U] = lu(A); % single intial LU-factorization, see Rem. 2.2.13
n = size(A,1); x = rand(n,1); x = x/norm(x); % random initial guess
y = U\(L\x); lmin = 1/norm(y); y = y*lmin; lold = 0;
while(abs(lmin-lold) > tol*lmin) % termination, if small relative change
  lold = lmin; x = y;
  y = U\(L\x); % core iteration:  y = A^{-1}x,
  lmin = 1/norm(y); % new approxmation of λ_min(A)
  y = y*lmin;  % normalization y := y/||y||_2
end
```

Remark 6.3.53 (Shifted inverse iteration).

△



Idea: A posteriori adaptation of shift

Use $\alpha := \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})$ in k-th step of inverse iteration.

R. Hiptmair

rev 38355, November 10, 2011

Algorithm 6.3.56 (Rayleigh quotient iteration). $\rightarrow$ [35, Alg. 25.2]]

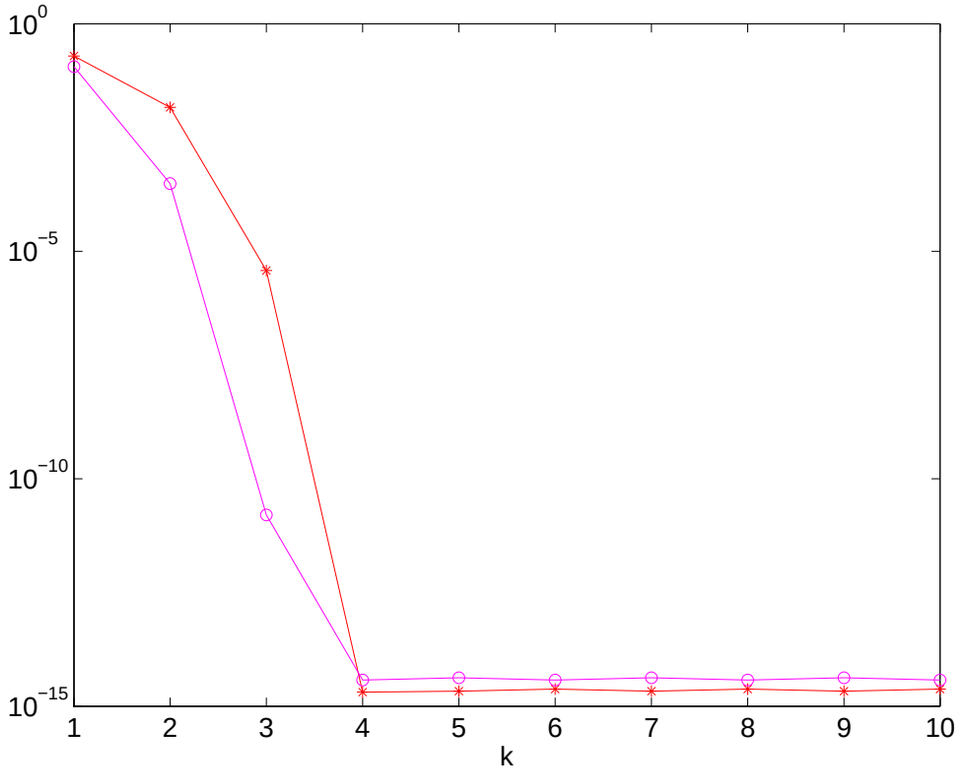
Code 6.3.57: Rayleigh quotient iteration (for *normal* $\mathbf{A} \in \mathbb{R}^{n,n}$)

```
function [z,lmin] = rqui(A,tol,maxit)
alpha = 0; n = size(A,1);
z = rand(size(A,1),1); z = z/norm(z); % z^(0)
for i=1:maxit
  z = (A-alpha*speye(n))\z; % z^(k+1) = (A - ρ_A(z^(k))I)^(-1) x^(k)
  z = z/norm(z); lmin=dot(A*z,z); % Computation of ρ_A(z^(k+1))
  if (abs(alpha-lmin) < tol*lmin) % Desired relative accuracy reached ?
    break; end;
  alpha = lmin;
end
```

Example 6.3.58 (Rayleigh quotient iteration).

R. Hiptmair

rev 38355, November 10, 2011




```
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';
```

o : $|\lambda_{\min} - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})|$

$*$: $\left\|\mathbf{z}^{(k)} - \mathbf{x}_j\right\|$, $\lambda_{\min} = \lambda_j$, $\mathbf{x}_j \in \mathrm{Eig}_{\mathbf{A}}(\lambda_j)$,

: $\left\|\mathbf{x}_j\right\|_2 = 1$

k	$\lvert\lambda_{\min} - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})\rvert$	$\left\|\mathbf{z}^{(k)} - \mathbf{x}_j\right\|$
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. $\rightarrow$ *[35, Thm. 25.4]*
If $\mathbf{A} = \mathbf{A}^H$, *then* $\rho_{\mathbf{A}}(\mathbf{z}^{(k)})$ *converges* locally of order 3 *($\rightarrow$ Def. 4.1.14) to the smallest eigenvalue (in modulus), when* $\mathbf{z}^{(k)}$ *are generated by the Rayleigh quotient iteration* (6.3.56).

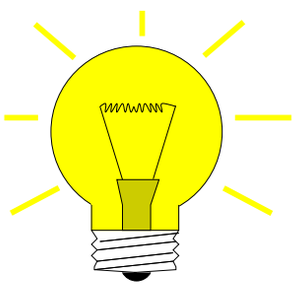
$\diamond$

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 ?



Idea: (for inverse iteration without shift, $\mathbf{A} = \mathbf{A}^H$ s.p.d.)

Instead of solving $\mathbf{Aw} = \mathbf{z}^{(k-1)}$ compute $\mathbf{w} = \mathbf{B}^{-1}\mathbf{z}^{(k-1)}$ with "inexpensive" s.p.d. approximate inverse $\mathbf{B}^{-1} \approx \mathbf{A}^{-1}$

➢ $\mathbf{B} \hat{=}$ 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 $\mathbf{A}^{-1} \to \mathbf{B}^{-1}$ possible only when applied to residual quantity

residual quantity = quantity that $\to 0$ in the case of convergence to exact solution

[Preconditioned inverse iteration (PINVIT) for s.p.d. $\mathbf{A}$]

$\mathbf{z}^{(0)}$ arbitrary,

$$\begin{aligned} \mathbf{w} &= \mathbf{z}^{(k-1)} - \mathbf{B}^{-1}(\mathbf{A}\mathbf{z}^{(k-1)} - \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{z}^{(k-1)}) \;, \\ \mathbf{z}^{(k)} &= \frac{\mathbf{w}}{\|\mathbf{w}\|_2} \;, \end{aligned} \qquad k = 1, 2, \dots \;. \tag{6.3.60}$$

R. Hiptmair

rev 38355, November 10, 2011

Code 6.3.61: preconditioned inverse iteration (6.3.60)

```
function [lmin,z,res] =
  pinvit(evalA,n,invB,tol,maxit)
% invB ≙ handle to function implementing preconditioner B^{-1}
z = (1:n)'; z = z/norm(z); % initial guess
res = []; rho = 0;
for i=1:maxit
  v = evalA(z); rhon = dot(v,z); % Rayleigh quotient
  r = v - rhon*z;     % residual
  z = z - invB(r);    % iteration according to (6.3.60)
  z = z/norm(z);      % normalization
  res = [res; rhon]; % tracking iteration
  if (abs(rho-rhon) < tol*abs(rhon)), break;
  else rho = rhon; end
end
lmin =  dot(evalA(z),z); res = [res; lmin],
```

Computational effort:

1 matrix×vector

1 evaluation of preconditioner

A few AXPY-operations

Example 6.3.62 (Convergence of PINVIT).

S.p.d. matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, tridiagonal preconditioner, see Ex. 5.3.11

```
A = spdiags(repmat([1/n,-1,2*(1+1/n),-1,1/n],n,1),
  [-n/2,-1,0,1,n/2],n,n);
evalA = @(x) A*x;
% inverse iteration
```

```
invB = @(x) A\x;
% tridiagonal preconditioning
B = spdiags(spdiags(A,[-1,0,1]),[-1,0,1],n,n); invB  = @(x) B\x;
```

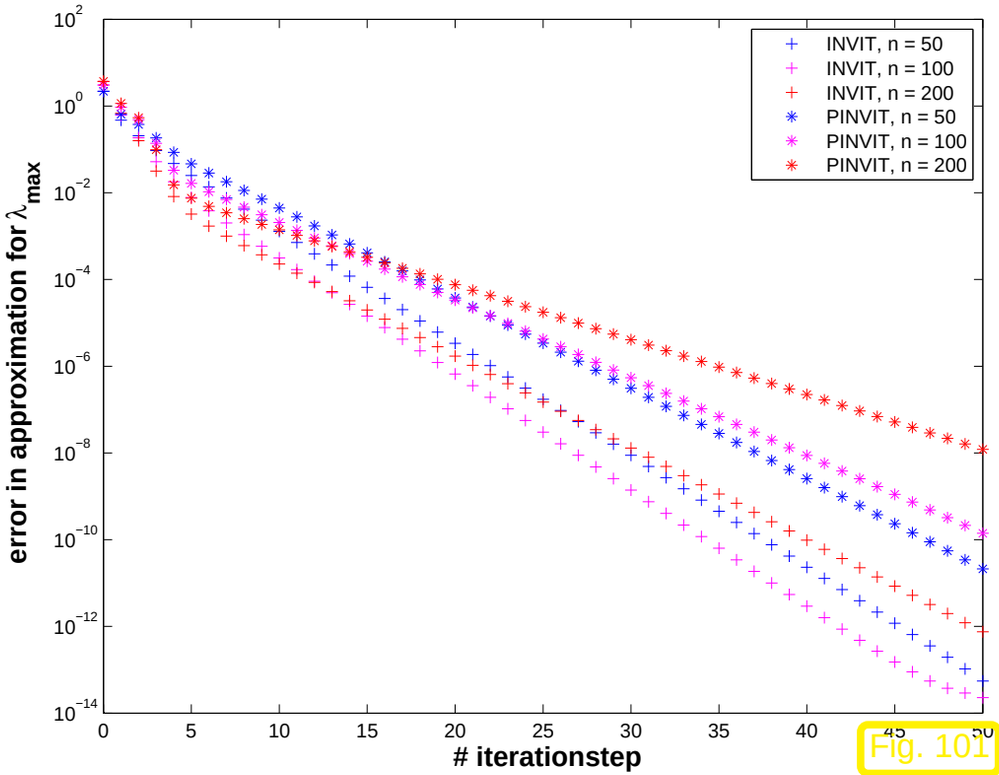


Fig. 101

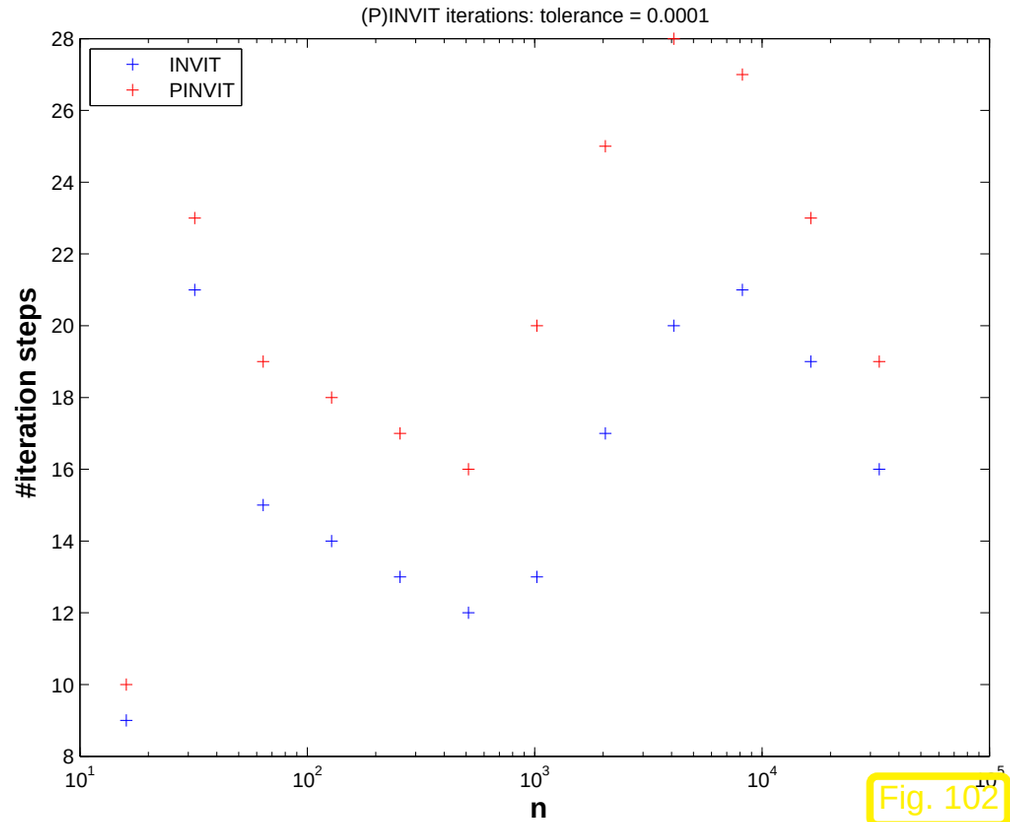


Fig. 102

◇

Theory [47, 46]:
- linear convergence of (6.3.60)
- fast convergence, if spectral condition number $\kappa(\mathbf{B}^{-1}\mathbf{A})$ small

6.3.4 Subspace iterations

Example 6.3.68 (Vibrations of a truss structure). *cf.* [35, Sect. 3], MATLAB's `truss` demo

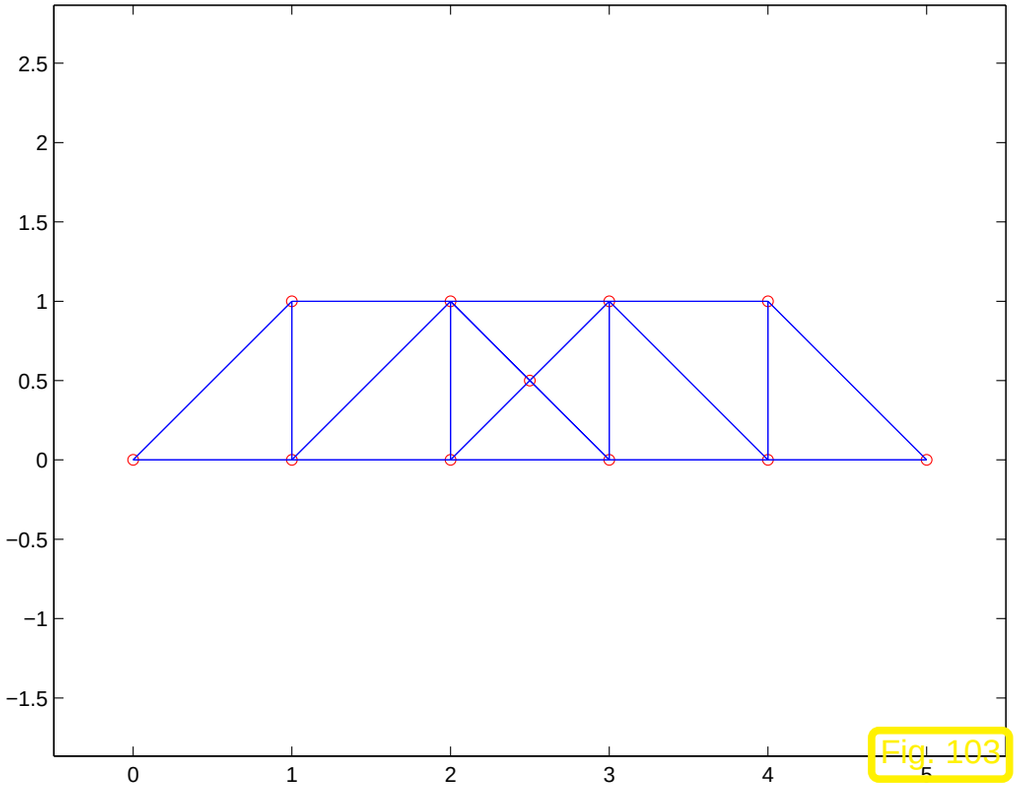


Fig. 103

◁ a "bridge" truss

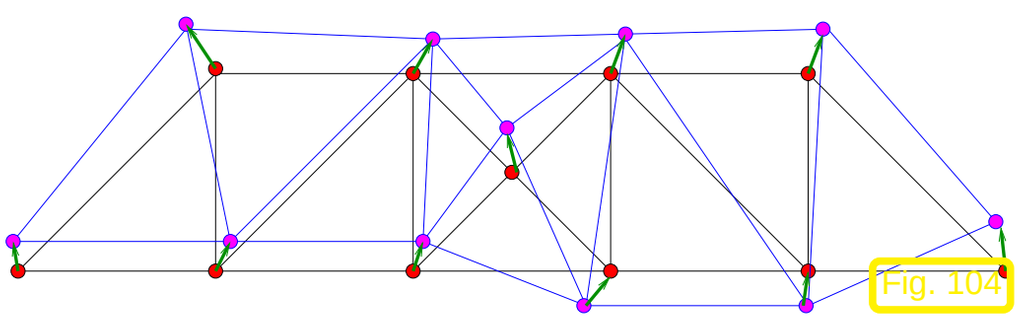


Fig. 104

◁ deformed truss:

- $\hat{=}$ point masses at positions $\mathbf{p}^i$

$\rightarrow$ $\hat{=}$ displacement vectors $\mathbf{u}^i$

- $\hat{=}$ shifted masses at $\mathbf{p}^i + \mathbf{u}^i$

(6.3.78) ▶ $$\boxed{\mathbf{M}\frac{d\mathbf{u}}{dt^2}(t) + \mathbf{A}\mathbf{u}(t) = \mathbf{f}(t)} \quad . \tag{6.3.79}$$

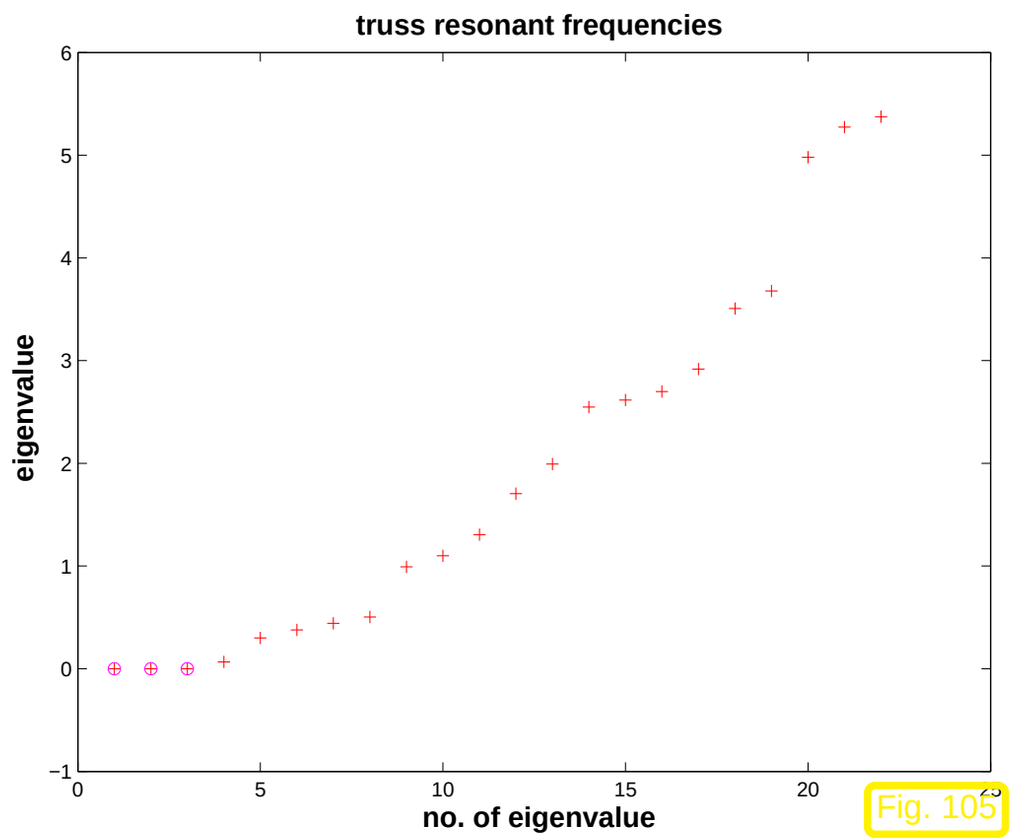


Fig. 105

◁ resonant frequencies of bridge truss from Fig. 116.

R. Hiptmair

rev 38355, November 10, 2011

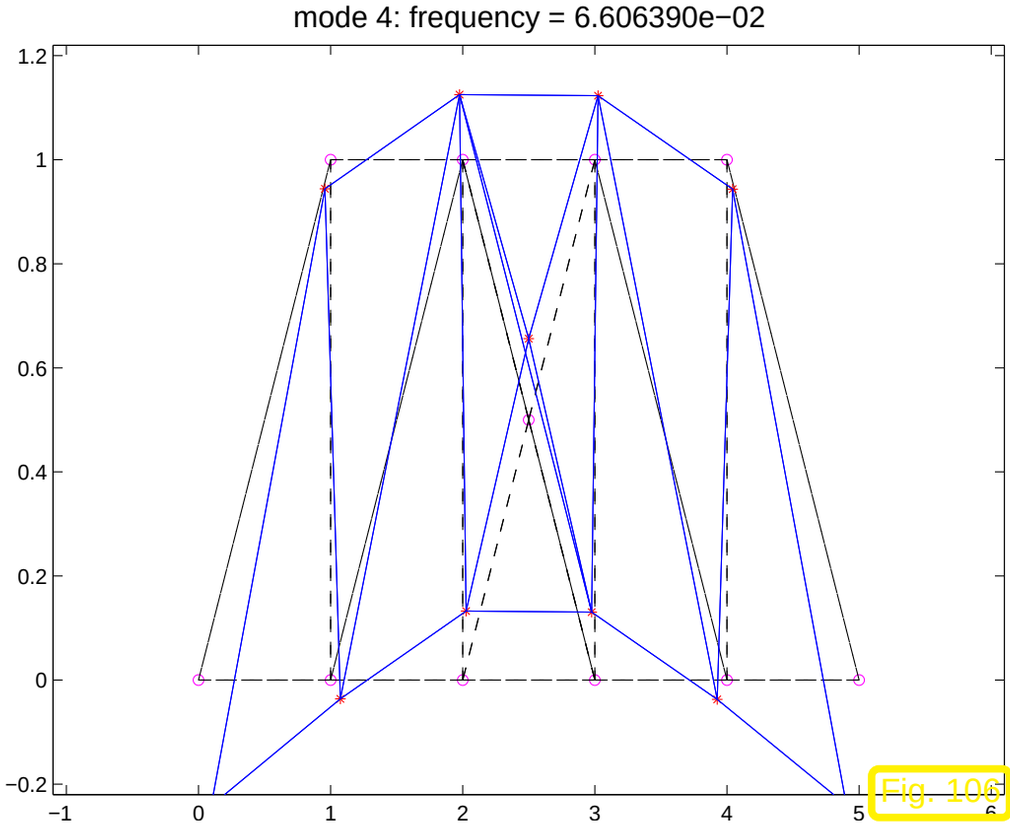


Fig. 106

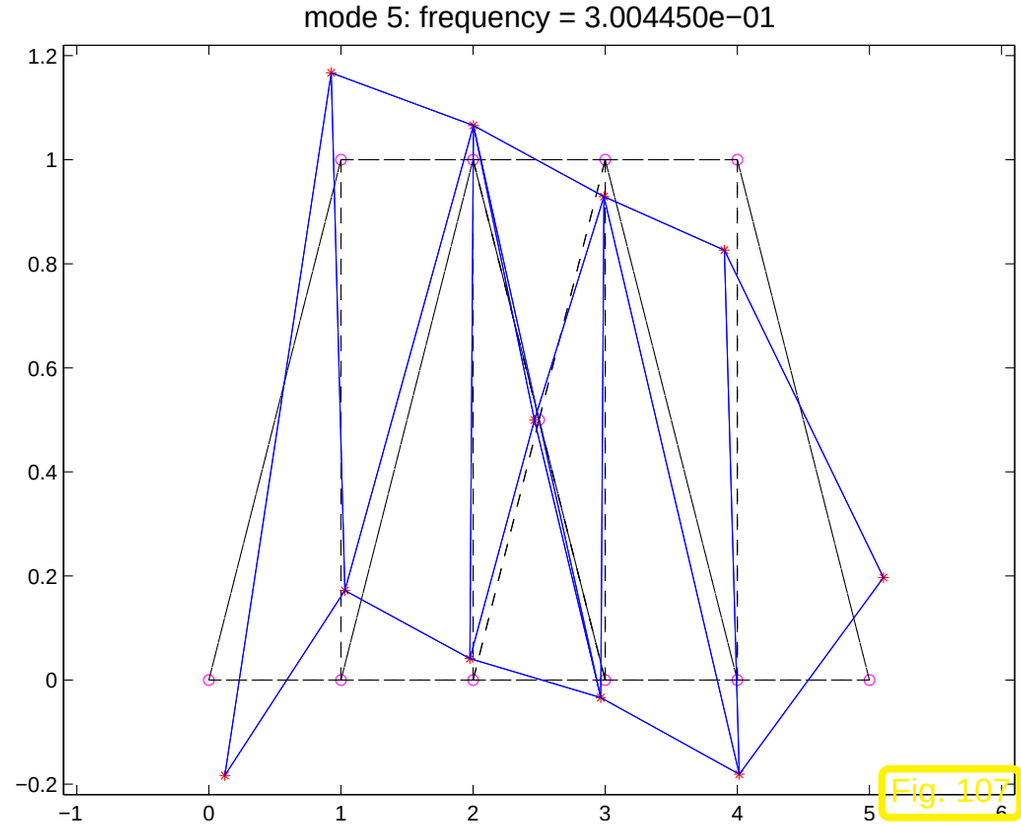


Fig. 107

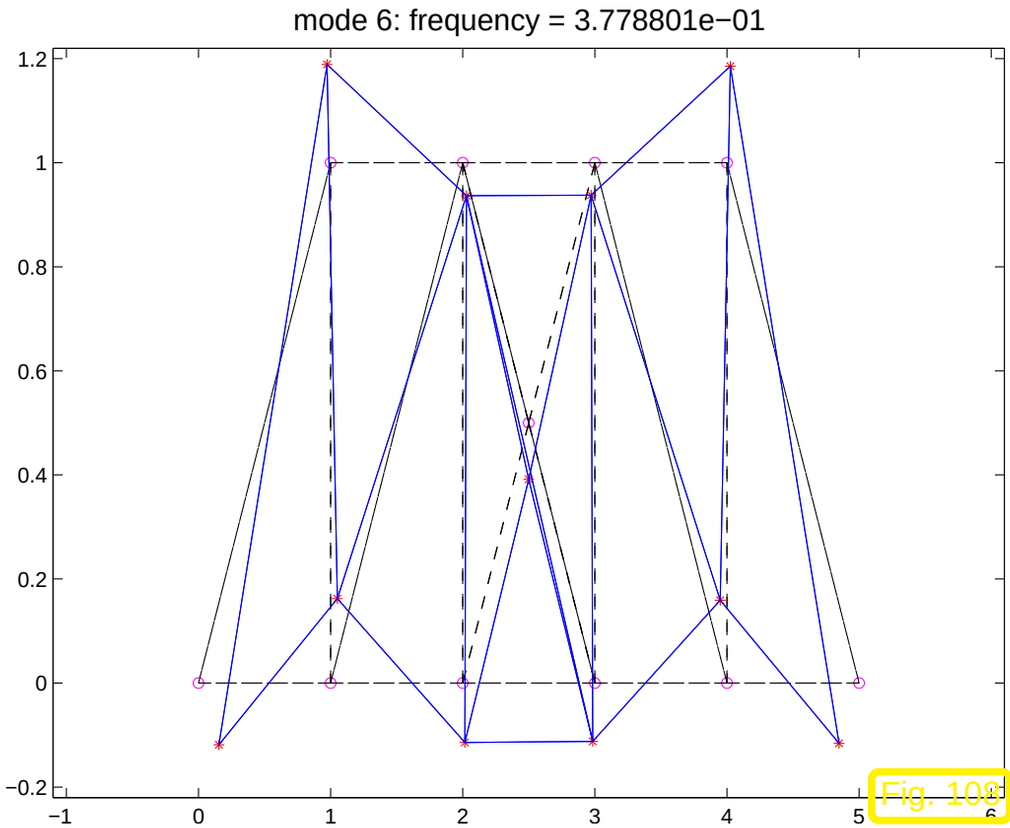


Fig. 108

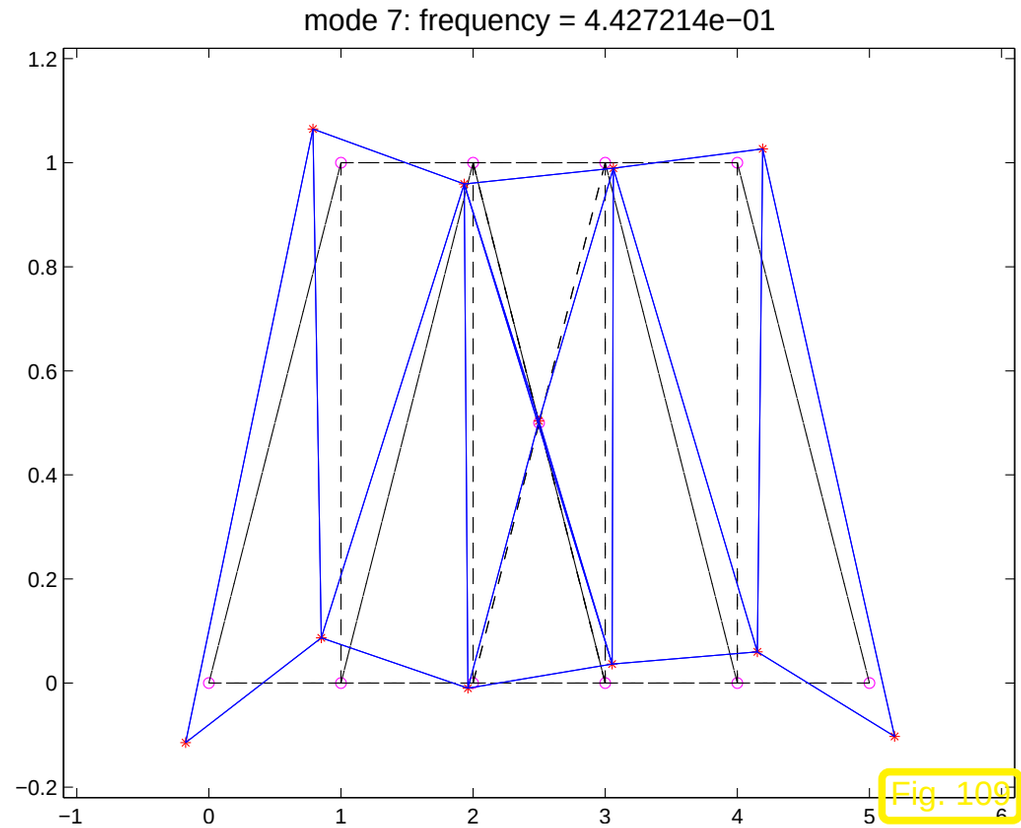


Fig. 109

◇

Task: Compute m, $m \ll n$, of the smallest/largest (in modulus) eigenvalues of $\mathbf{A} = \mathbf{A}^{\mathsf{H}} \in \mathbb{C}^{n,n}$ and associated eigenvectors.

6.3.4.1 Orthogonalization

Code 6.3.82: one step of subspace power iteration, $m = 2$

```
function [v,w] = sspowitstep(A,v,w)
v = A*v; w = A*w; % "power iteration", cf. (6.3.11)
% orthogonalization, cf. Gram-Schmidt orthogonalization (5.2.11)
v = v/norm(v); w = w - dot(v,w)*v; w = w/norm(w); % now w ⊥ v
```

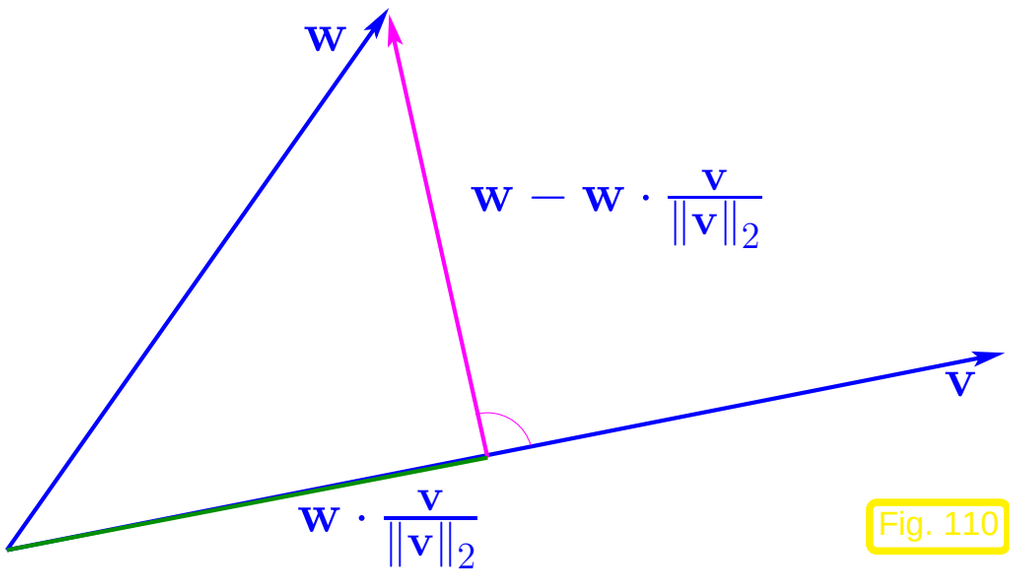


Fig. 110

◁ Orthogonalization of two vectors (see Line 4 of Code 6.3.81)

Example 6.3.83 (Subspace power iteration with orthogonal projection).

☞ construction of matrix $\mathbf{A} = \mathbf{A}^\top$ as in Ex. 6.3.58

$\sigma(\mathbf{A}) = \{1, 2, \dots, 10\}$:

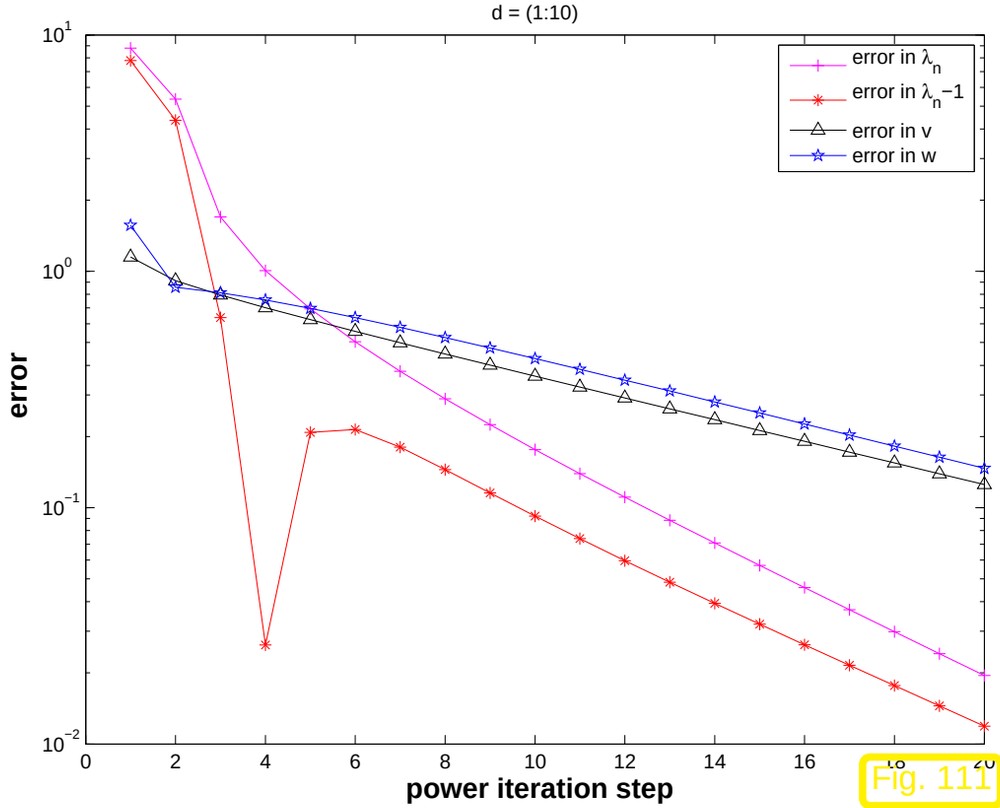


Fig. 111

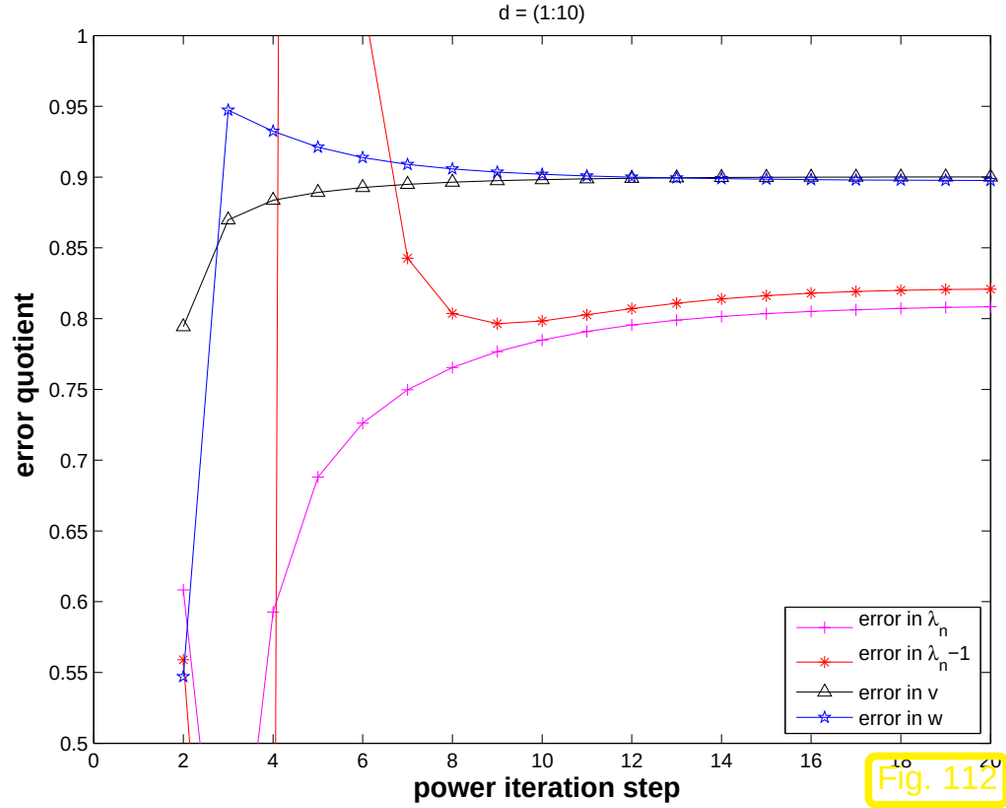


Fig. 112

$\sigma(\mathbf{A}) = \{0.5, 1, \ldots, 4, 9.5, 10\}$:

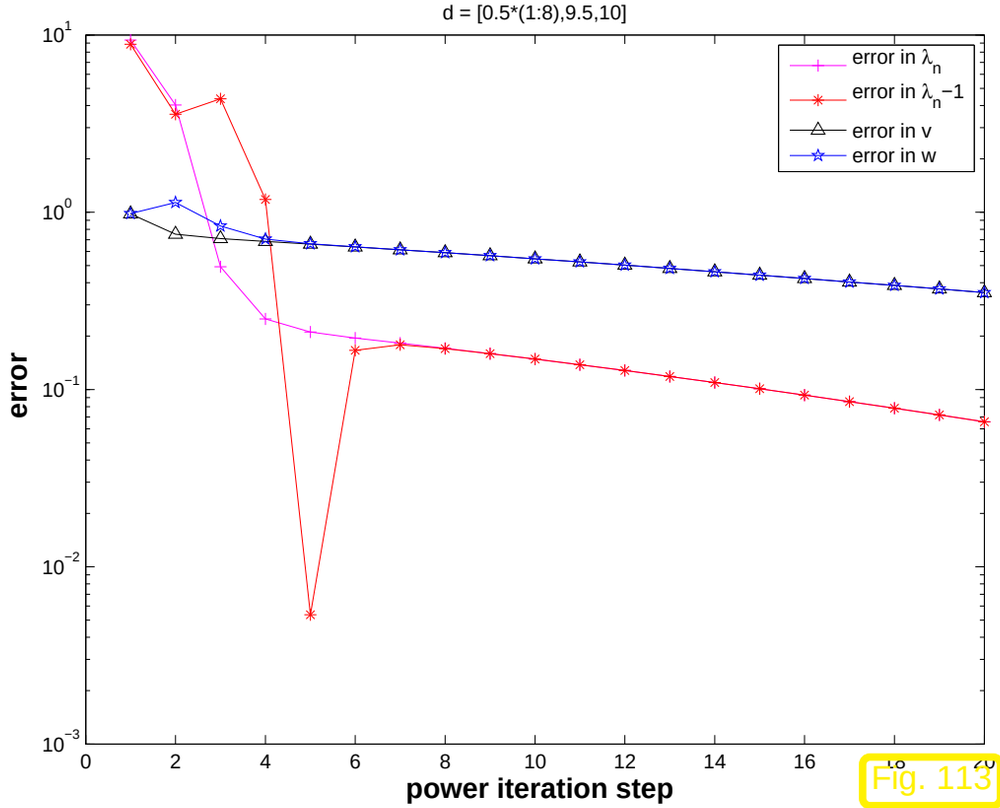


Fig. 113

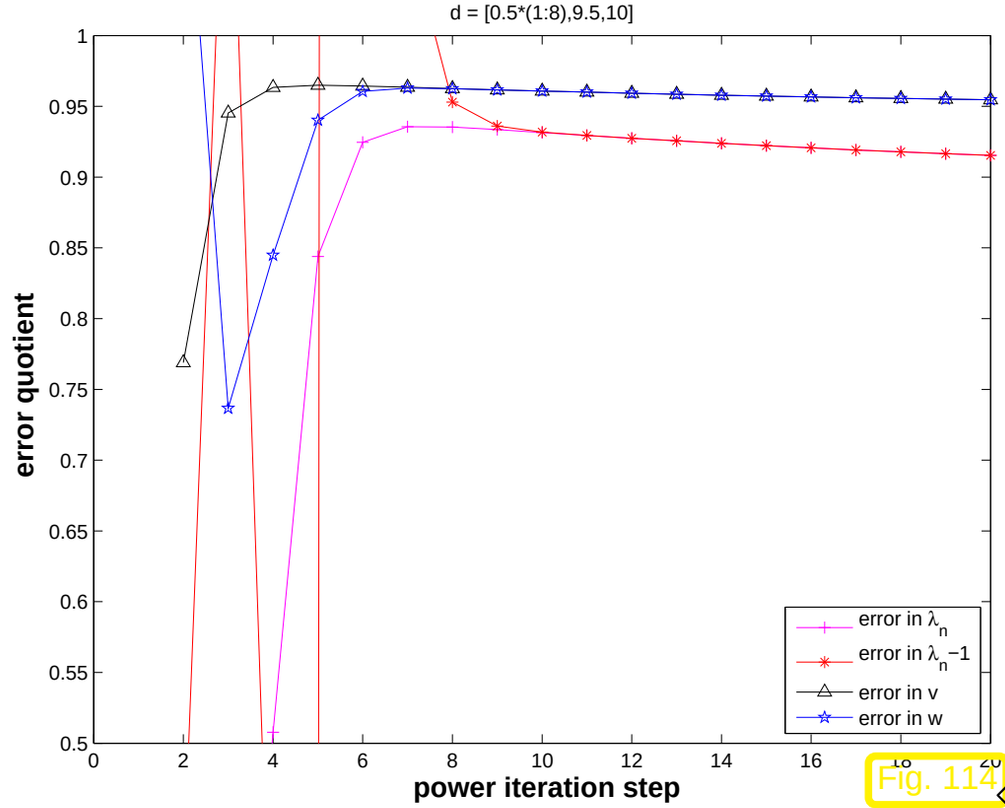


Fig. 114

R. Hiptmair

rev 38355, November 10, 2011

Code 6.3.89: Gram-Schmidt orthonormalization (do not use, unstable algorithm!)

```
function Q = gso(V)
% Gram-Schmidt orthonormalization of the columns of V ∈ ℝ^{n,m}, see
% (6.3.87). The vectors q_1,...,q_m are returned as the columns of
% the orthogonal matrix Q.
m = size(V,2);
Q = V(:,1)/norm(V(:,1)); % normalization
for l=2:m
  q = V(:,l);
  % orthogonalization
```

```
  for k=1:l-1
    q = q - dot(Q(:,k),V(:,l))*Q(:,k);
  end
  Q = [Q,q/norm(q)]; % normalization
end
```

`[Q,R] = qr(V,0)` (Asymptotic computational cost: $O(m^2 n)$)

dummy return value (for our purposes) dummy argument

Code 6.3.91: General subspace power iteration step with `qr` based orthonormalization

```
function V = sspowitstep(A,V)
% power iteration with orthonormalization for A = A^T.
% columns of matrix V span subspace for power iteration.
V = A*V;            % actual power iteration on individual columns
[V,R] = qr(V,0); % Gram-Schmidt orthonormalization (6.3.87)
```


6.3.4.2 Ritz projection

Task: given $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{K}^n$, $k \ll n$, extract (good approximations of) eigenvectors of $\mathbf{A} = \mathbf{A}^{\mathsf{H}} \in \mathbb{K}^{n,n}$ contained in $\operatorname{Span}\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$.

Code 6.3.98: one step of subspace power iteration with Ritz projection, matrix version

```
function V = sspowitsteprp(A,V)
V = A*V;                % power iteration applied to columns of V
[Q,R] = qr(V,0);        % orthonormalization, see Sect. 6.3.4.1
[U,D] = eig(Q'*A*Q);    % Solve Ritz projected m×m eigenvalue problem
V = Q*U;                % recover approximate eigenvectors
ev = diag(D);           % approximate eigenvalues
```

Example 6.3.99 (Power iteration with Ritz projection).

Matrix as in Ex. 6.3.83, $\sigma(\mathbf{A}) = \{0.5, 1, \dots, 4, 9.5, 10\}$:

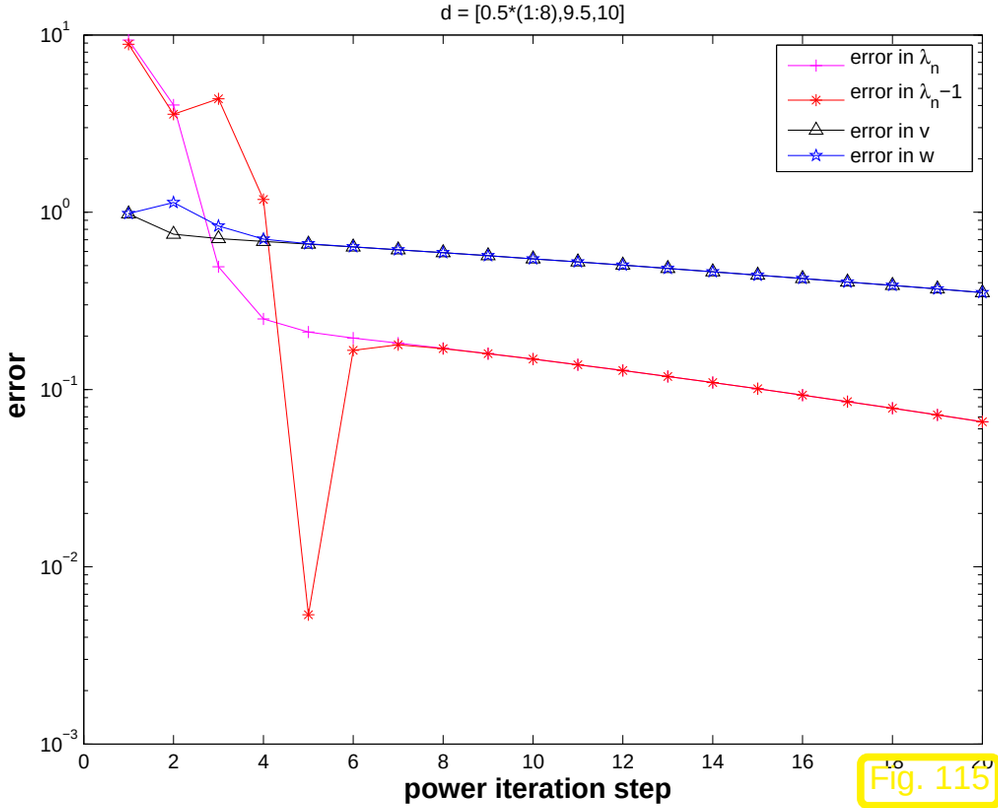


Fig. 115

simple orthonormalization, Ex. 6.3.83

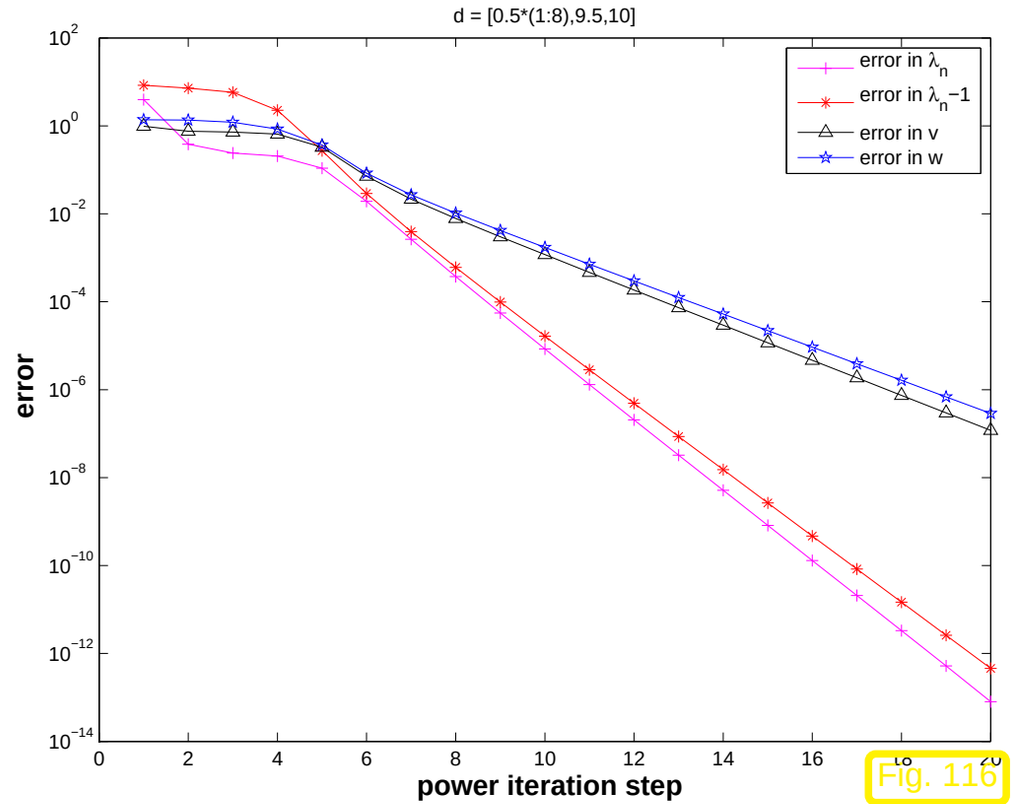
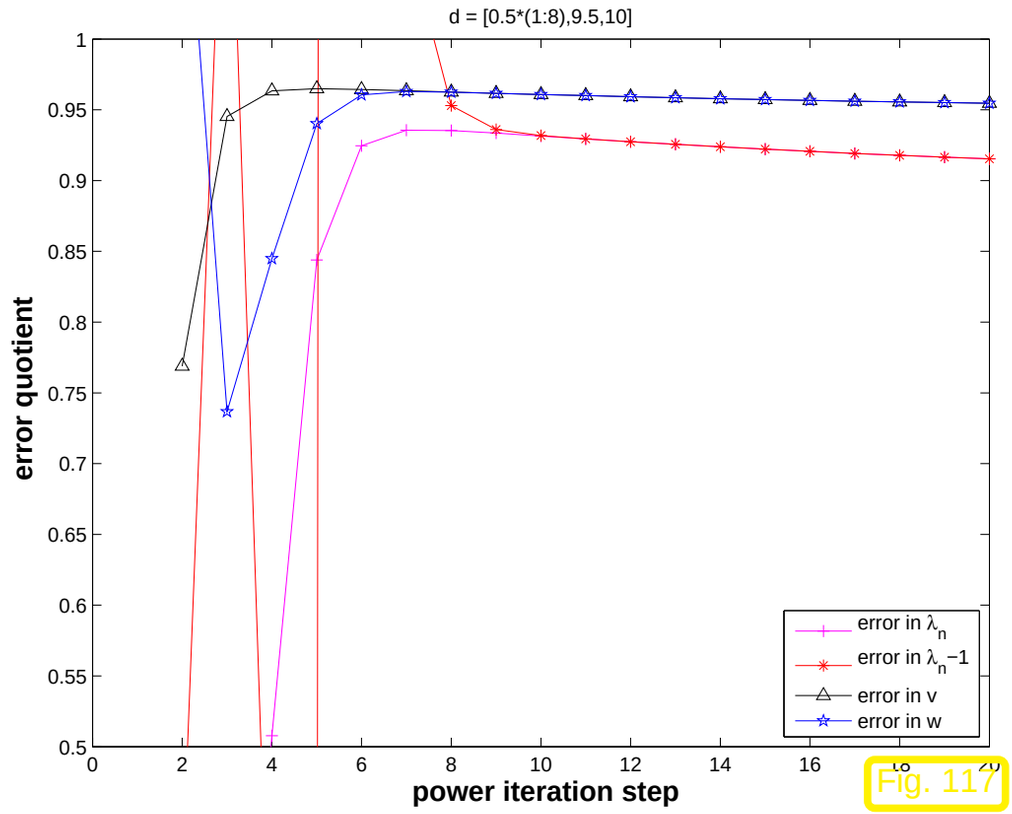


Fig. 116

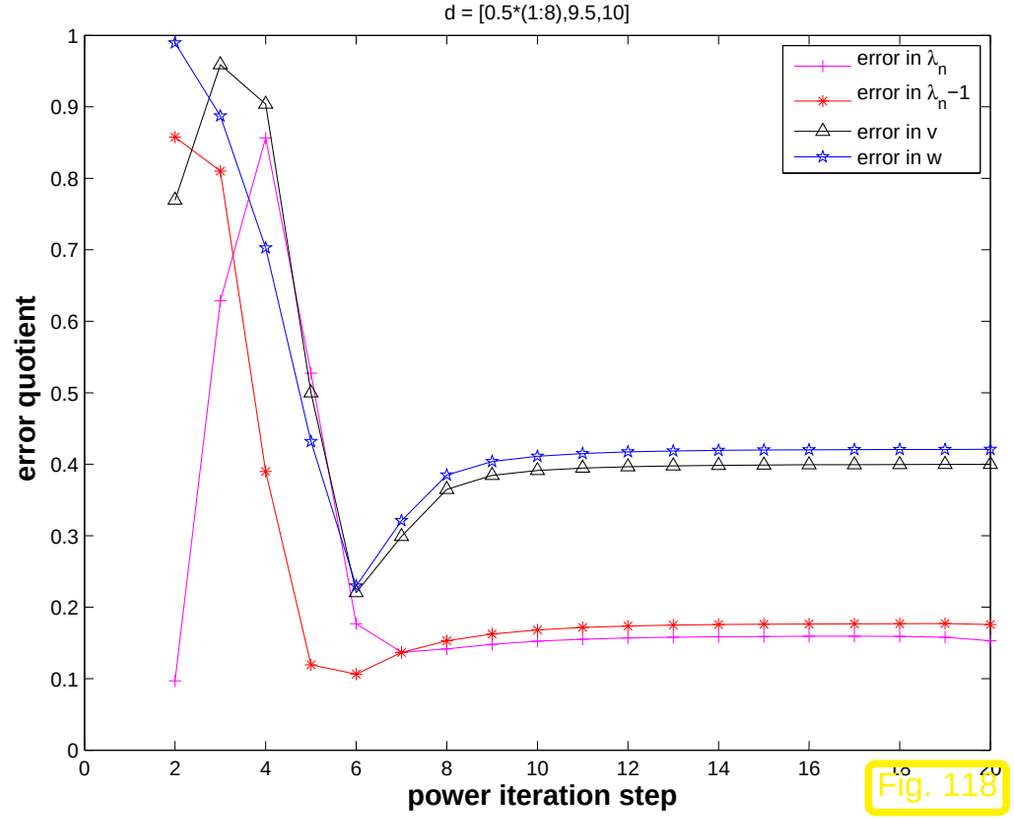
with Ritz projection, Code 6.3.97

R. Hiptmair

rev 38355, November 10, 2011



simple orthonormalization, Ex. 6.3.83



with Ritz projection, Code 6.3.97

R. Hiptmair

rev 38355, November 10, 2011

◇

Example 6.3.102 (Convergence of subspace variant of direct power method).

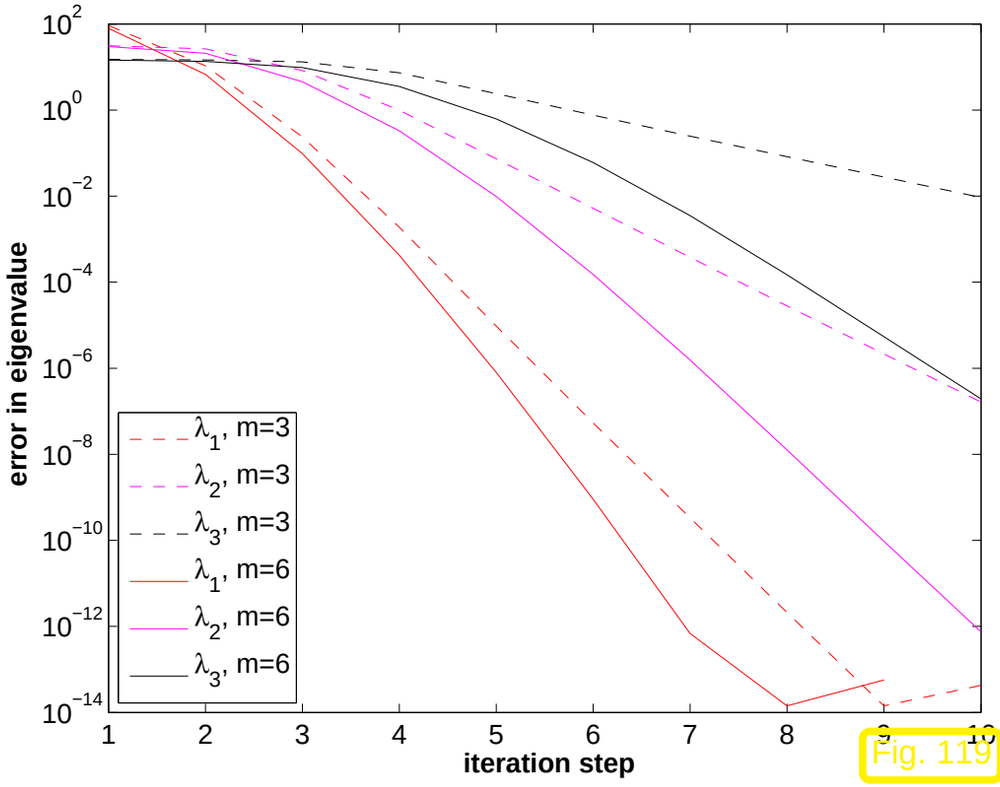


Fig. 119

S.p.d. test matrix: $a_{ij} := \min\{\frac{i}{j}, \frac{j}{i}\}$

```
n=200; A = gallery('lehmer',n);
```

"Initial eigenvector guesses":

```
V = eye(n,m);
```

◇

R. Hiptmair

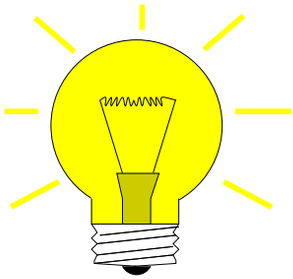
rev 38355, November 7, 2011

Remark 6.3.103 (Subspace power methods).

Analoguous to Alg. 6.3.101: construction of subspace variants of inverse iteration (→ Code 6.3.51), PINVIT (6.3.60), and Rayleigh quotient iteration (6.3.56).

△

6.4 Krylov Subspace Methods [35, Sect. 30]



Idea: Better $\mathbf{z}^{(k)}$ from Ritz projection onto $V := \mathrm{Span}\left\{\mathbf{z}^{(0)},\dots,\mathbf{z}^{(k)}\right\}$
(= space spanned by previous iterates)

For direct power method (6.3.11): $\mathbf{z}^{(k)} || \mathbf{A}^k \mathbf{z}^{(0)}$

$$V = \mathrm{Span}\left\{\mathbf{z}^{(0)}, \mathbf{A}\mathbf{z}^{(0)},\dots,\mathbf{A}^{(k)}\mathbf{z}^{(0)}\right\} = \mathcal{K}_{k+1}(\mathbf{A},\mathbf{z}^{(0)})$$ a Krylov space, $\rightarrow$ Def. 5.2.6 . (6.4.2)

R. Hiptmair

rev 38355, November 7, 2011

Code 6.4.3: Ritz projections onto Krylov space (6.4.2)

```
function [V,D] = kryleig(A,m)
% Ritz projection onto Krylov subspace. An orthonormal basis of K_m(A,1) is assembled into the columns of V.
n = size(A,1); V = (1:n)'; V = V/norm(V);
for l=1:m-1
  V = [V,A*V(:,end)]; [Q,R] = qr(V,0);
  [W,D] = eig(Q'*A*Q); V = Q*W;
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 $\mathbf{A}$!)

Example 6.4.4 (Ritz projections onto Krylov space).

```
n=100;
M=gallery('tridiag',-0.5*ones(n-1,1),2*ones(n,1),-1.5*ones(n-1,1));
[Q,R]=qr(M); A=Q'*diag(1:n)*Q; % synthetic matrix, σ(A) = {1,2,3,...,100}
```

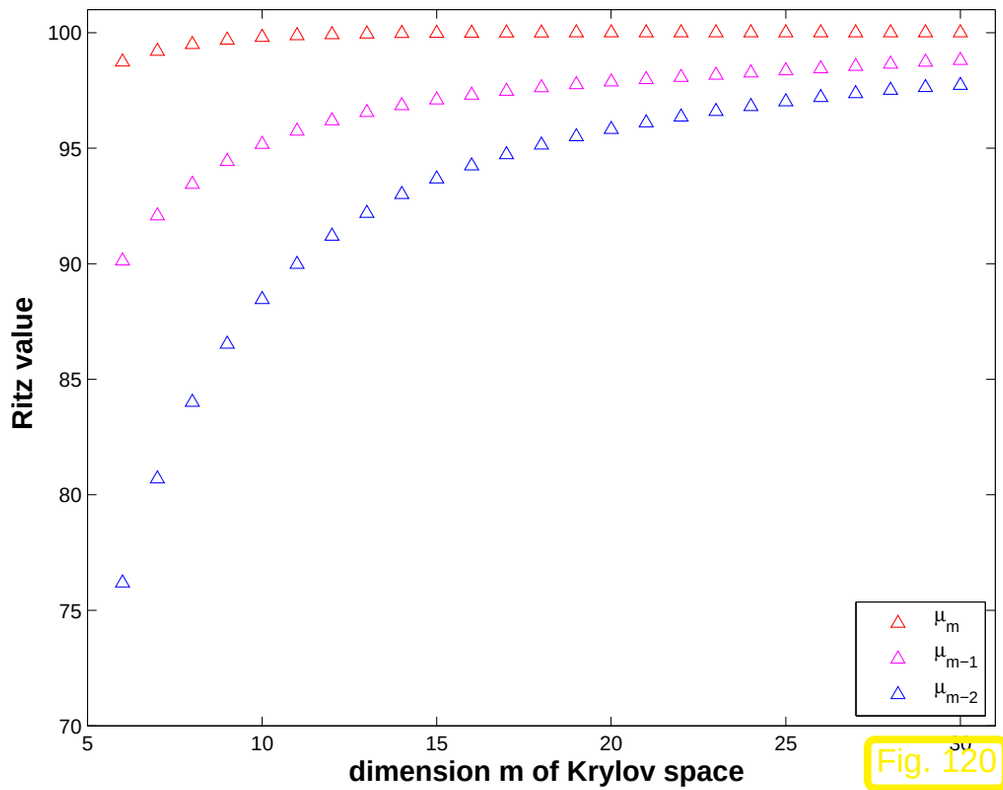


Fig. 120

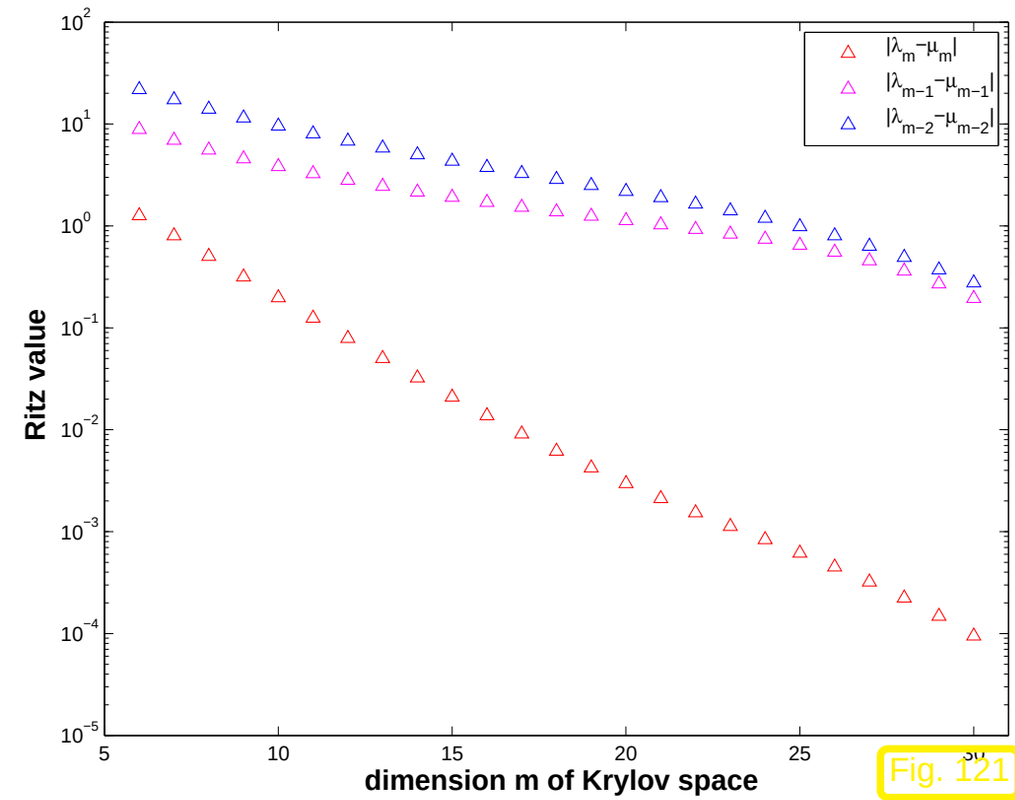


Fig. 121

R. Hiptmair

rev 38355, November 7, 2011

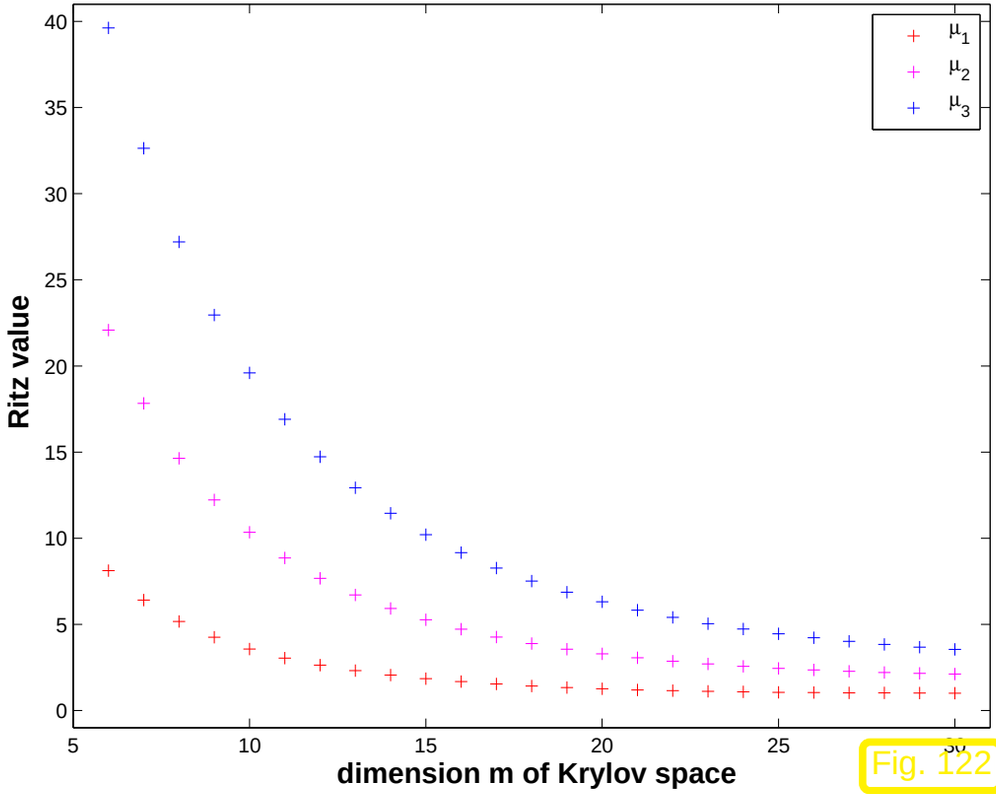


Fig. 122

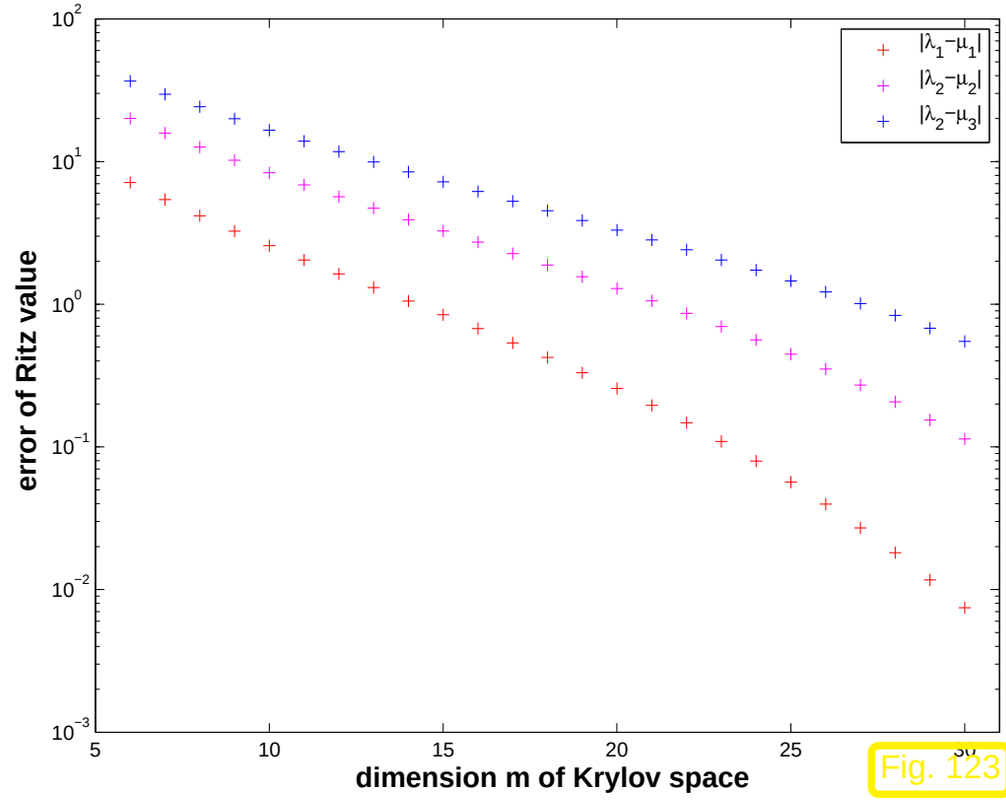


Fig. 123

◇

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 $\mathbf{A}\mathbf{x} = \mathbf{z}$ 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 $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ onto $\mathcal{K}_m(\mathbf{A},\mathbf{z})$: orthogonal matrix

$$\mathbf{V}_m^T\mathbf{A}\mathbf{V}_m\mathbf{x} = \lambda\mathbf{x}\ , \quad \mathbf{V}_m := \left(\frac{\mathbf{r}_0}{\|\mathbf{r}_0\|},\dots,\frac{\mathbf{r}_{m-1}}{\|\mathbf{r}_{m-1}\|}\right) \in \mathbb{R}^{n,m}\ . \tag{6.4.8}$$

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.*

$$\mathbf{V}_l^H\mathbf{A}\mathbf{V}_l = \begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & & \\ & & & & \ddots & \\ & & & \ddots & \ddots & \beta_{k-1} \\ & & & & \beta_{k-1} & \alpha_k \end{pmatrix} =: \mathbf{T}_l \in \mathbb{K}^{k,k} \quad \text{[tridiagonal matrix]} \tag{6.4.10}$$

Algorithm for computing $\mathbf{V}_l$ and $\mathbf{T}_l$:

Lanczos process

Computational effort/step:

$1\times$	$\mathbf{A}\times$vector
2	dot products
2	AXPY-operations
1	division

Code 6.4.11: Lanczos process, *cf.* Code 5.2.17

```
function [V,alph,bet] = lanczos(A,k,z0)
% 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.
V = z0/norm(z0);
% Vectors storing entries of tridiagonal matrix (6.4.10
alph=zeros(k,1); bet = zeros(k,1);
for j=1:k
  q = A*V(:,j); alph(j) = dot(q,V(:,j));
  w = q - alph(j)*V(:,j);
  if (j > 1), w = w - bet(j-1)*V(:,j-1); end
  bet(j) = norm(w); V = [V,w/bet(j)];
end
bet = bet(1:end-1);
```

Example 6.4.12 (Lanczos process for eigenvalue computation).

$\mathbf{A}$ from Ex. 6.4.4

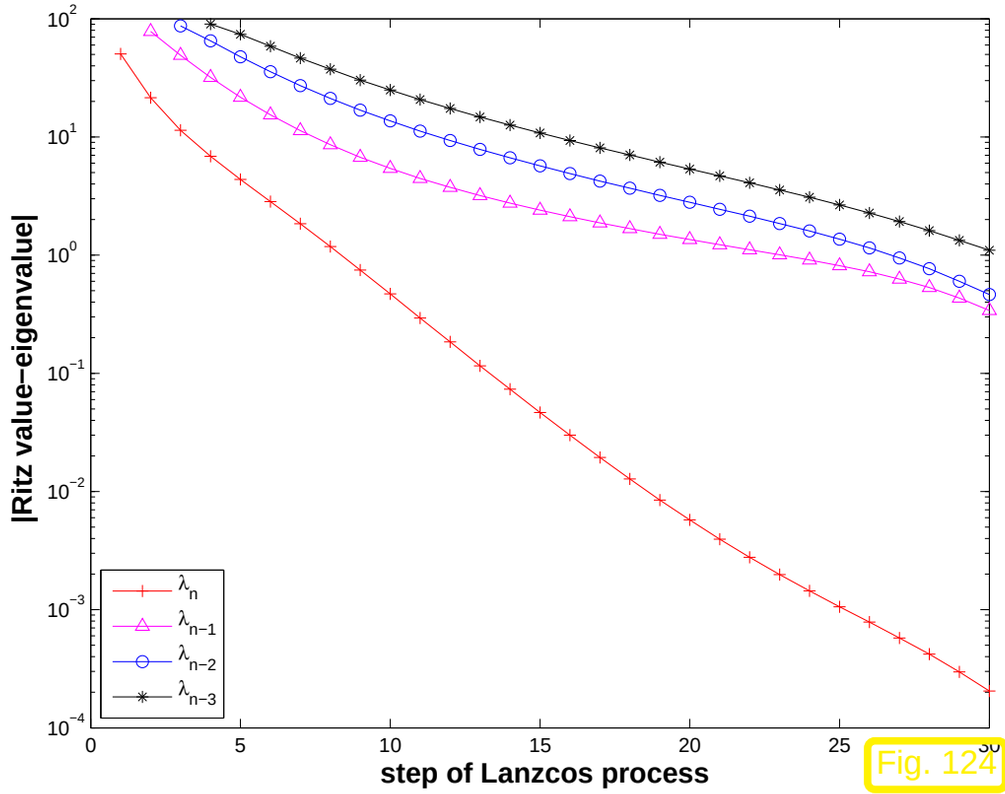


Fig. 124

`A = gallery('minij',100);`

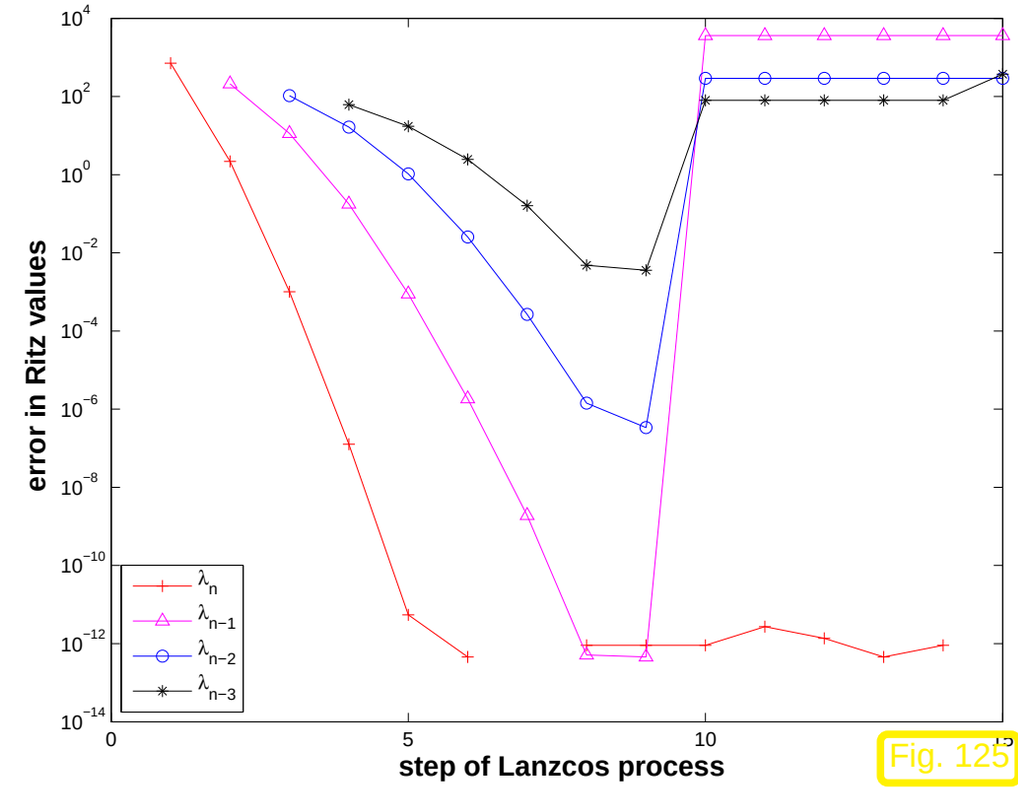


Fig. 125

◇

Example 6.4.13 (Impact of roundoff on Lanczos process).

$$\mathbf{A} \in \mathbb{R}^{10,10} \quad , \quad a_{ij} = \min\{i,j\} \; .$$

`A = gallery('minij',10);`

R. Hiptmair

rev 38355, November 7, 2011

Computed by `[V,alpha,beta] = lanczos(A,n,ones(n,1));`, see Code 6.4.10:

$$\mathbf{T} = \begin{pmatrix} 38.500000 & 14.813845 & & & & & & & \\ 14.813845 & 9.642857 & 2.062955 & & & & & & \\ & 2.062955 & 2.720779 & 0.776284 & & & & & \\ & & 0.776284 & 1.336364 & 0.385013 & & & & \\ & & & 0.385013 & 0.826316 & 0.215431 & & & \\ & & & & 0.215431 & 0.582380 & 0.126781 & & \\ & & & & & 0.126781 & 0.446860 & 0.074650 & \\ & & & & & & 0.074650 & 0.363803 & 0.043121 \\ & & & & & & & 0.043121 & 3.820888 & 11.991094 \\ & & & & & & & & 11.991094 & 41.254286 \end{pmatrix}$$

$$\sigma(\mathbf{A}) = \{0.255680, 0.273787, 0.307979, 0.366209, 0.465233, 0.643104, 1.000000, 1.873023, 5.048917, 44.766069\}$$

$$\sigma(\mathbf{T}) = \{0.263867, 0.303001, 0.365376, 0.465199, 0.643104, 1.000000, 1.873023, 5.048917, 44.765976, 44.766069\}$$

Uncanny cluster of computed eigenvalues of $\mathbf{T}$ ("ghost eigenvalues", [23, Sect. 9.2.5])

$$\mathbf{V}^H\mathbf{V} = \begin{pmatrix}
1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000251 & 0.258801 & 0.883711 \\
0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000106 & 0.109470 & 0.373799 \\
0.000000 & -0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000005 & 0.005373 & 0.018347 \\
0.000000 & 0.000000 & 0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000096 & 0.000328 \\
0.000000 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000001 & 0.000003 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 & 0.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 \\
0.000251 & 0.000106 & 0.000005 & 0.000000 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 \\
0.258801 & 0.109470 & 0.005373 & 0.000096 & 0.000001 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & 0.000000 \\
0.883711 & 0.373799 & 0.018347 & 0.000328 & 0.000003 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000
\end{pmatrix}$$

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

◇

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*.

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 $\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}$,$\mathbf{B}$ s.p.d.

`d = eigs(Afun,n,k)` : `Afun` = handle to function providing matrix$\times$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).

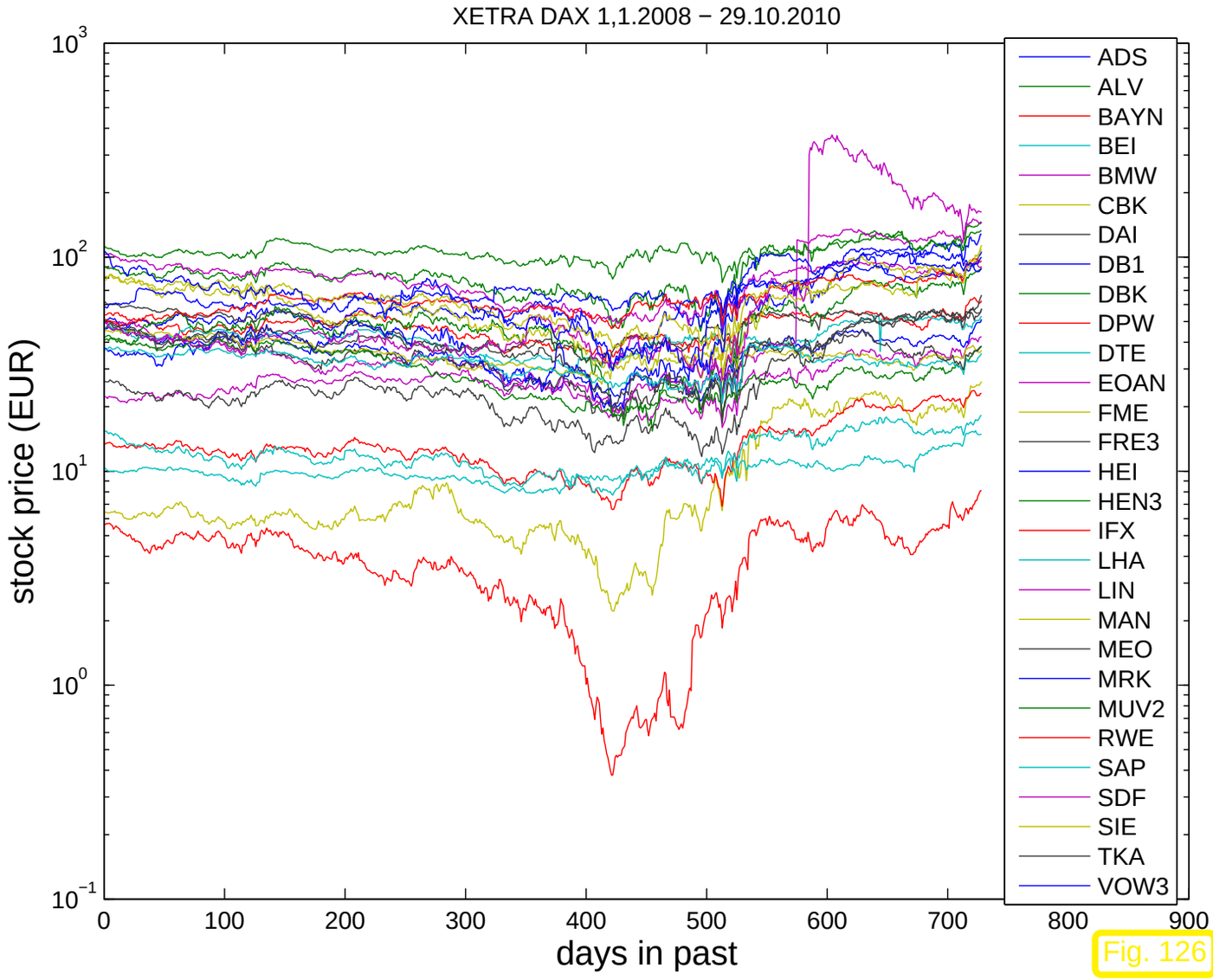


Fig. 126

◁ (end of day) stock prizes

Are there underlying governing trends **?**

◇

Example 6.5.2 (Classification from measured data).

R. Hiptmair

rev 38355, November 29, 2011

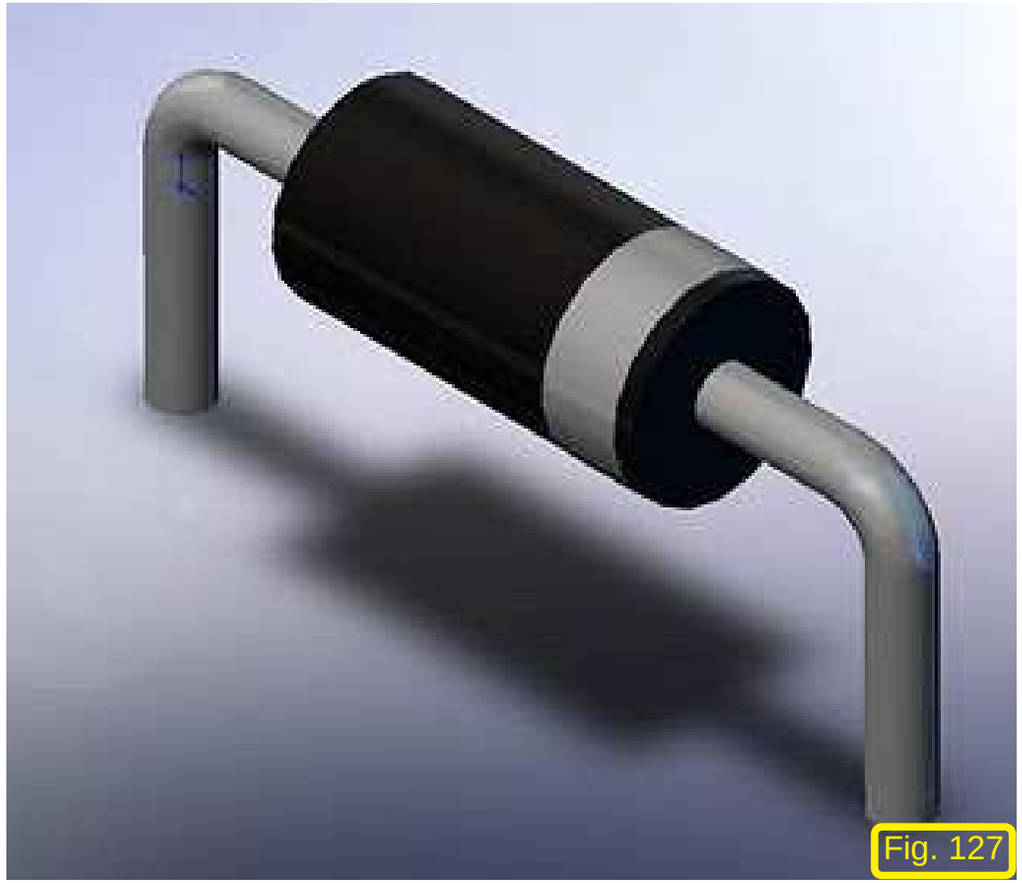


Fig. 127

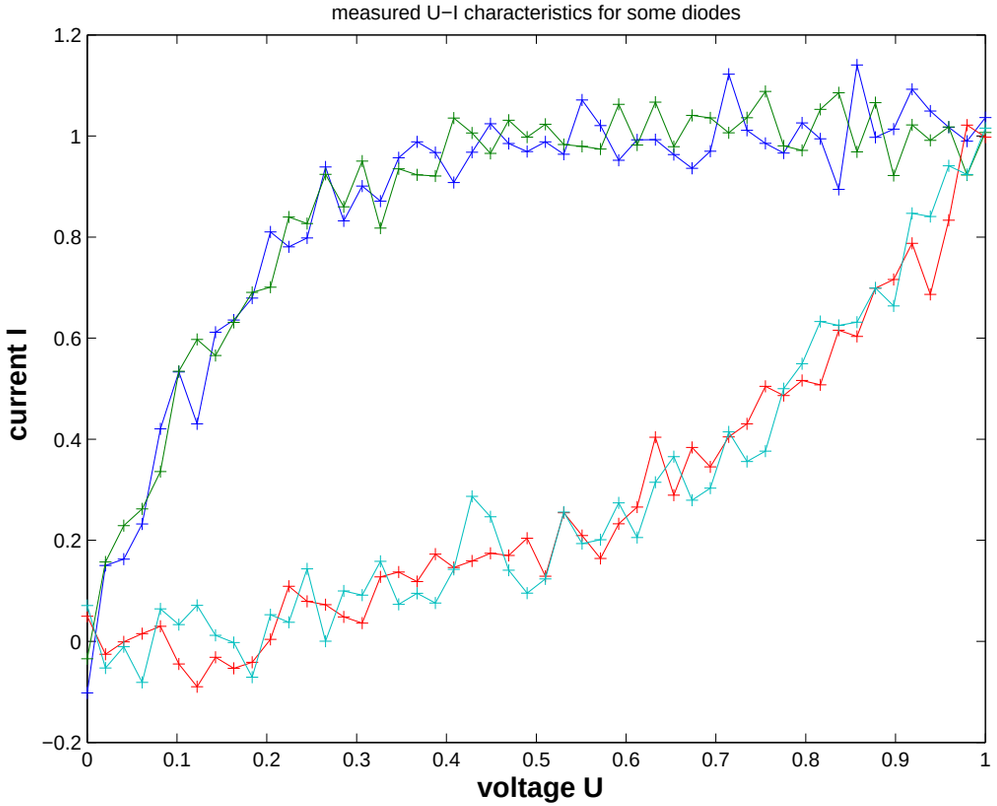
Given: measured U-I characteristics of diodes

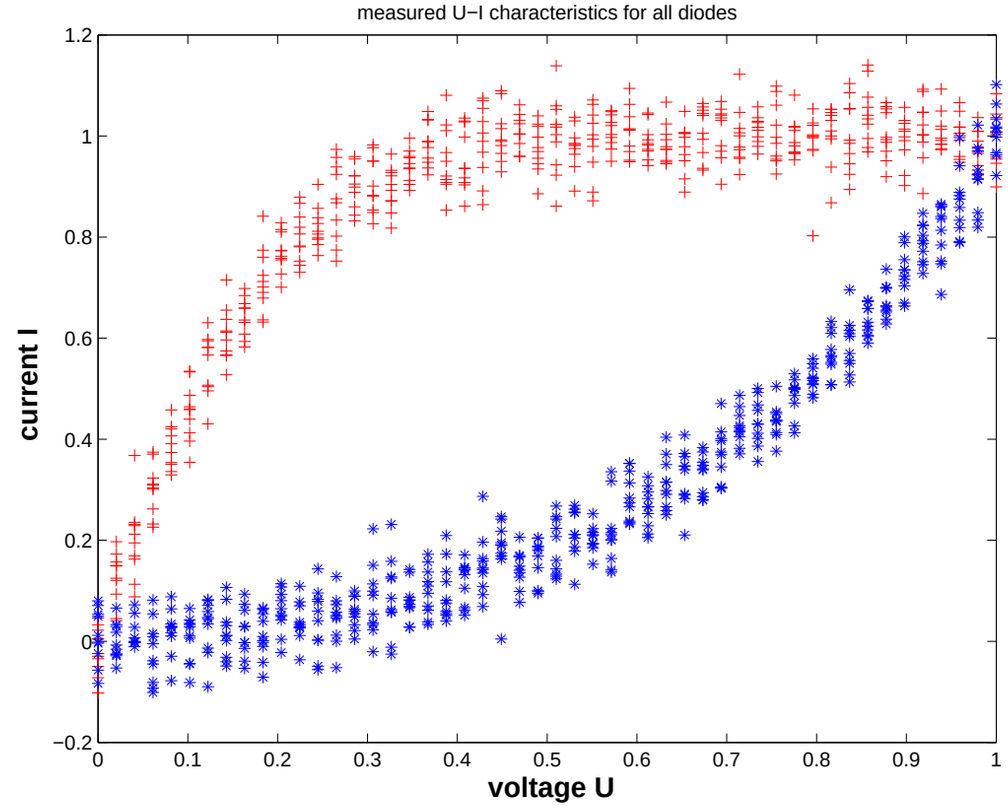
Find out

- how many types,
- the U-I characteristic of each type.

R. Hiptmair

rev 38355, November 29, 2011





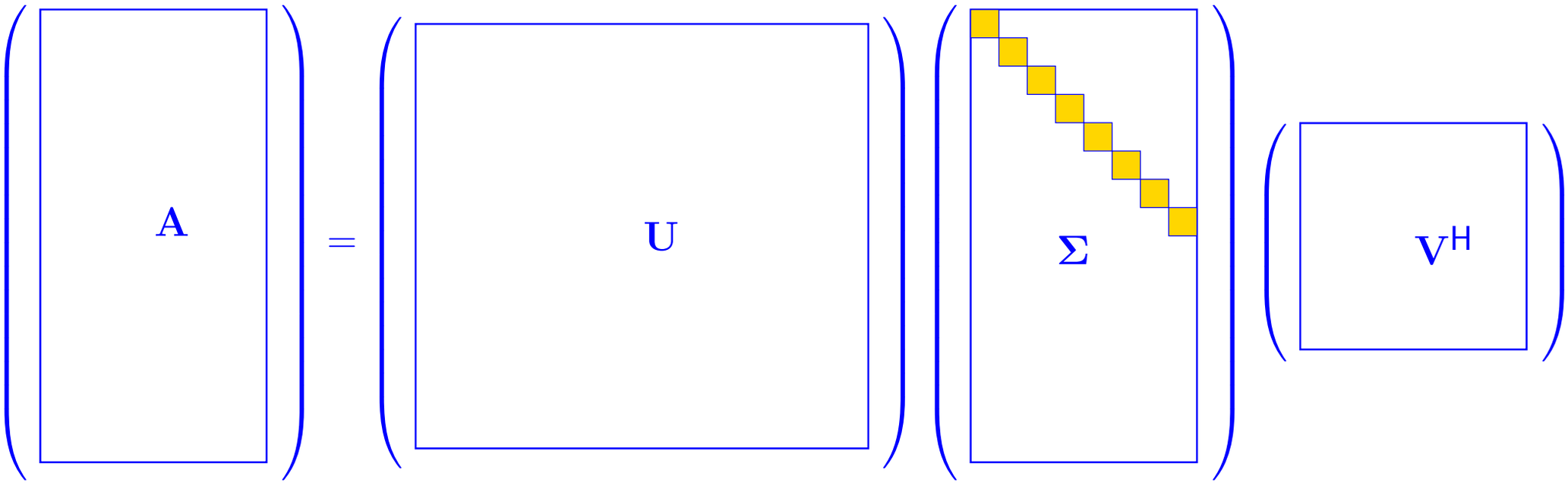
◇

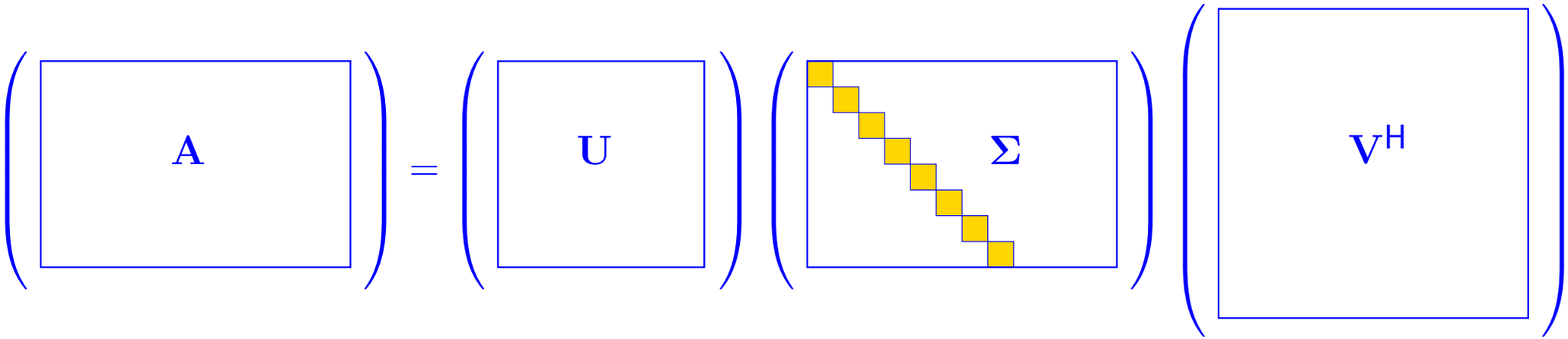
Theorem 6.5.6 (singular value decomposition). $\rightarrow$ *[48, Thm. 9.6], [27, Thm. 11.1]*

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* $^{(*)}$ *matrix* $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m,n}$, $p := \min\{m, n\}$, $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$ *such that*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathsf{H}} .$$

$(*)$: $\mathbf{\Sigma}$ (generalized) diagonal matrix $:\Leftrightarrow$ $(\mathbf{\Sigma})_{i,j} = 0$, if $i \neq j$, $1 \leq i \leq m$, $1 \leq j \leq n$.


A
=
U
Σ
V^H

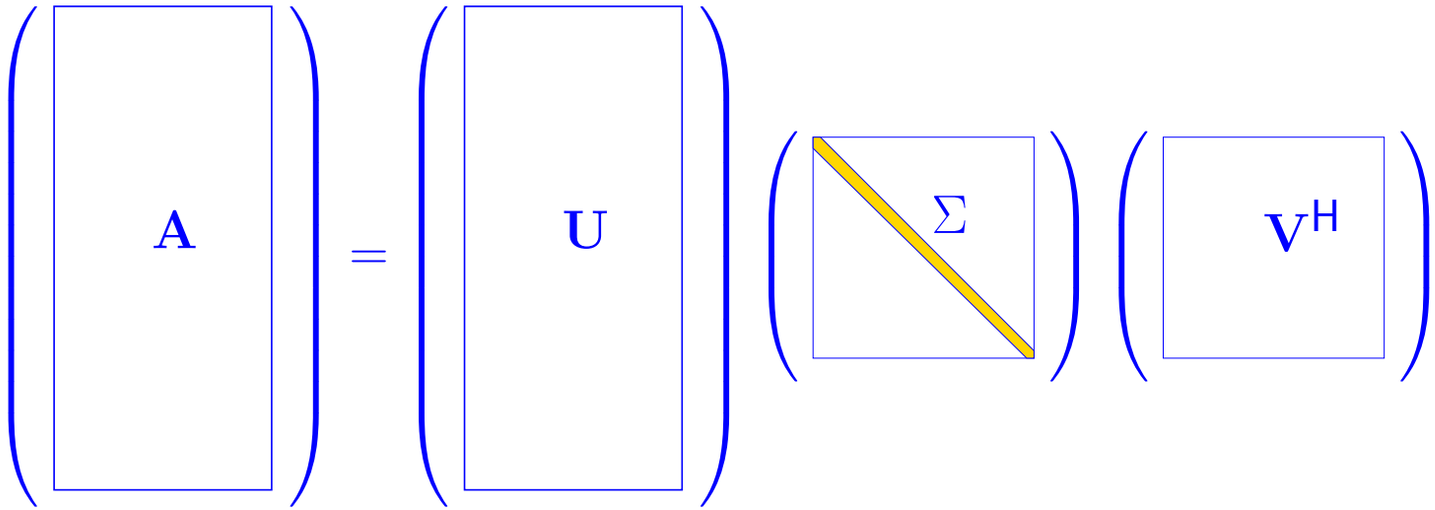

A
=
U
Σ
V^H

Definition 6.5.8 (Singular value decomposition (SVD))**.**
The decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathsf{H}}$ *of Thm. 6.5.6 is called singular value decomposition (SVD) of* $\mathbf{A}$*. The diagonal entries* σ_i *of* $\mathbf{\Sigma}$ *are the singular values of* $\mathbf{A}$*.*

MATLAB-functions (for algorithms see [23, Sect. 8.3]):

`s = svd(A)`	: computes singular values of matrix $\mathbf{A}$
`[U,S,V] = svd(A)`	: computes singular value decomposition according to Thm. 6.5.6
`[U,S,V] = svd(A,0)`	: "economical" singular value decomposition for $m > n$: : $\mathbf{U} \in \mathbb{K}^{m,n}$, $\Sigma \in \mathbb{R}^{n,n}$, $\mathbf{V} \in \mathbb{K}^{n,n}$
`s = svds(A,k)`	: k largest singular values (important for sparse $\mathbf{A} \rightarrow$ Def. 2.6.1)
`[U,S,V] = svds(A,k)`	: partial singular value decomposition: $\mathbf{U} \in \mathbb{K}^{m,k}$, $\mathbf{V} \in \mathbb{K}^{n,k}$, $\Sigma \in \mathbb{R}^{k,k}$ diagonal with k largest singular values of $\mathbf{A}$.

Explanation: “economical” singular value decomposition:



(MATLAB) algorithm for computing SVD is (numerically) stable $\rightarrow$ Def. 2.5.11

Complexity:

$2mn^2 + 2n^3 + O(n^2) + O(mn)$ for `s = svd(A)`,

$4m^2n + 22n^3 + O(mn) + O(n^2)$ for `[U,S,V] = svd(A)`,

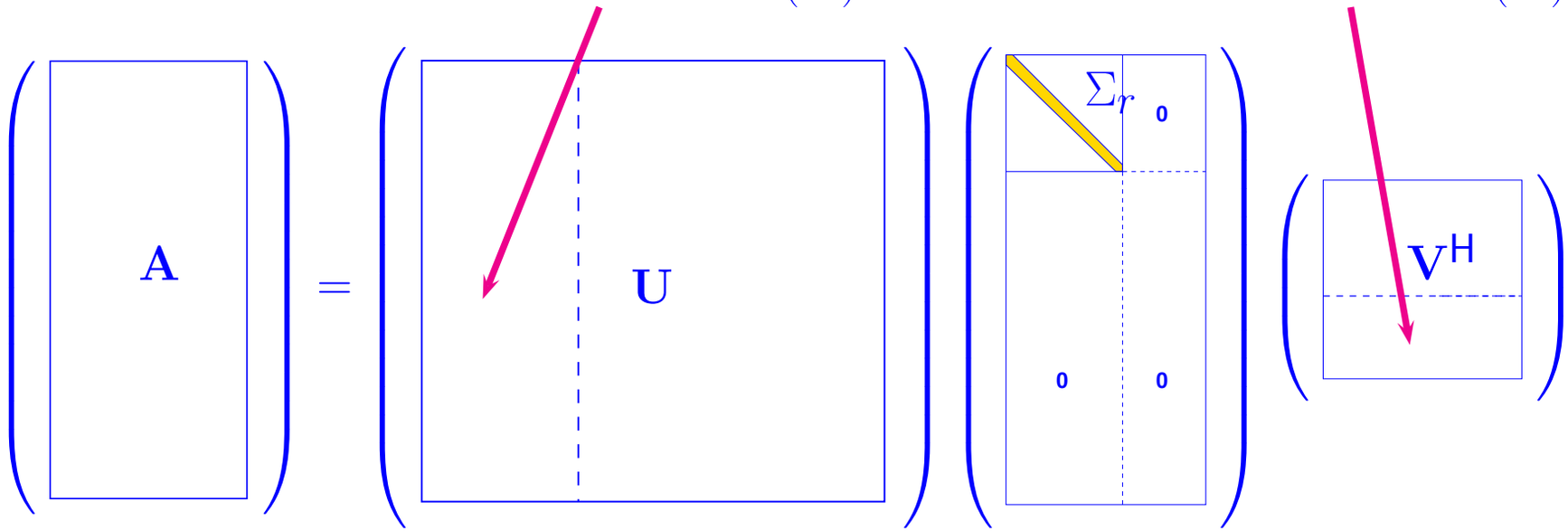
$O(mn^2) + O(n^3)$ for `[U,S,V]=svd(A,0)`, $m \gg n$.

- Application of SVD: computation of rank ($\rightarrow$ Def. 2.0.2), kernel and range of a matrix

R. Hiptmair

rev 38355, November 29, 2011

Illustration: columns = ONB of $\mathrm{Im}(\mathbf{A})$ rows = ONB of $\mathrm{Ker}(\mathbf{A})$

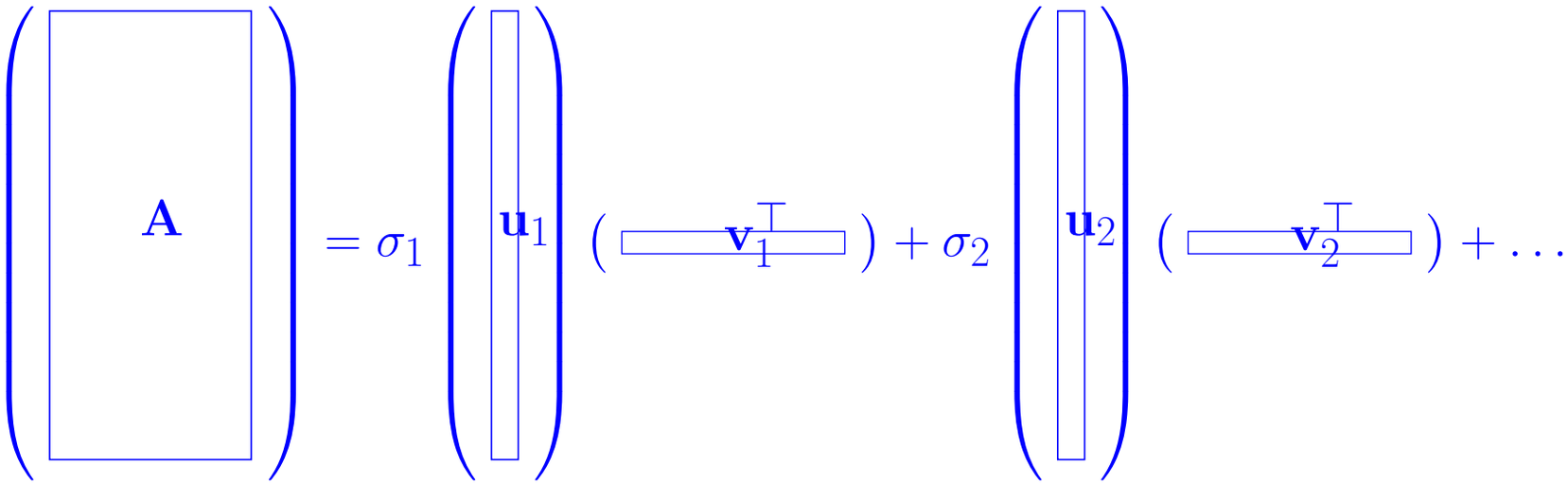


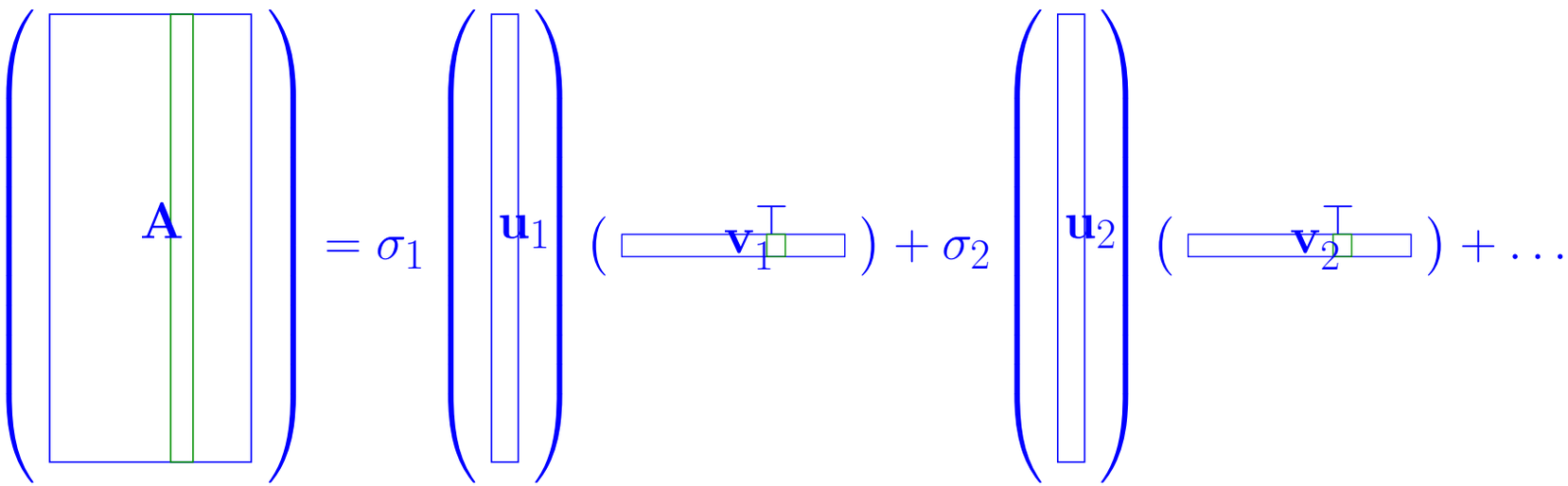
(6.5.15)

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
% kernel selection with relative tolerance tol
if (nargin < 2), tol = eps; end
[U,S,V] = svd(A);   % Singular value decomposition
s = diag(S);        % Extract vector of singular values
% find singular values of relative (w.r.t. σ1) size ≤tol
r = min(find(s/s(1) <= tol)); % "Numerical rank" +1
V = V(:,r:end);               % rightmost columns of V
```

- Application of SVD: PCA by SVD ($\rightarrow$ Rem. 6.5.3)





Example 6.5.18 (Principal component analysis for data classification). $\rightarrow$ Ex. 6.5.2 cnt'd

R. Hiptmair

rev 38355, November 29, 2011

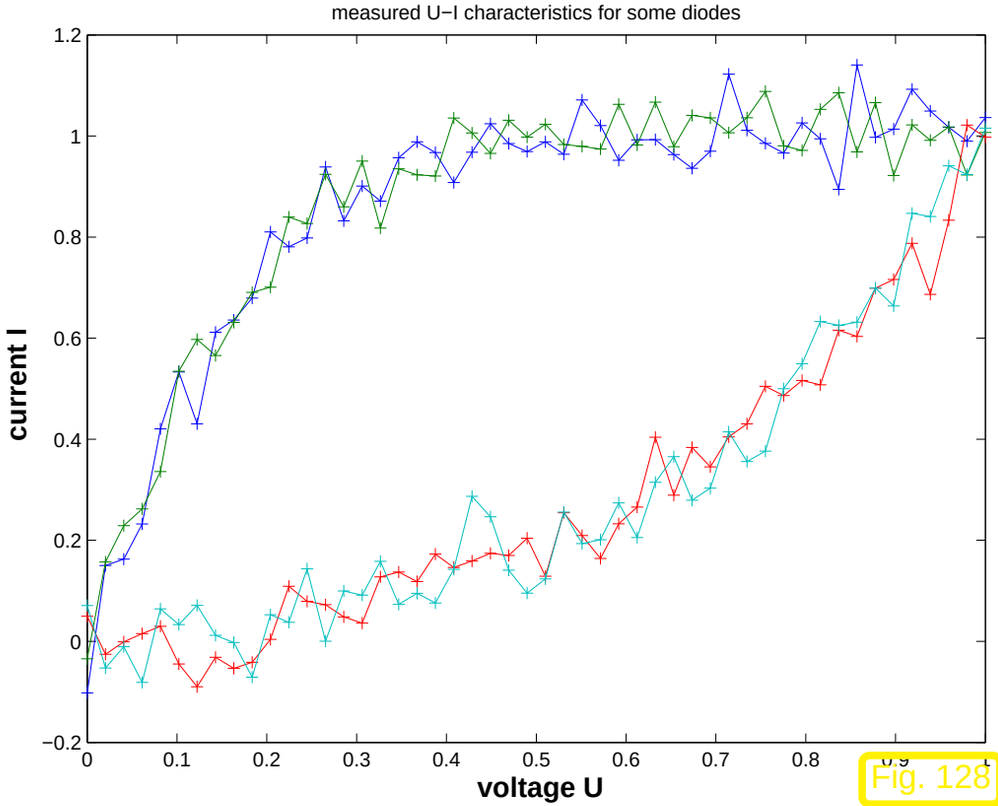


Fig. 128

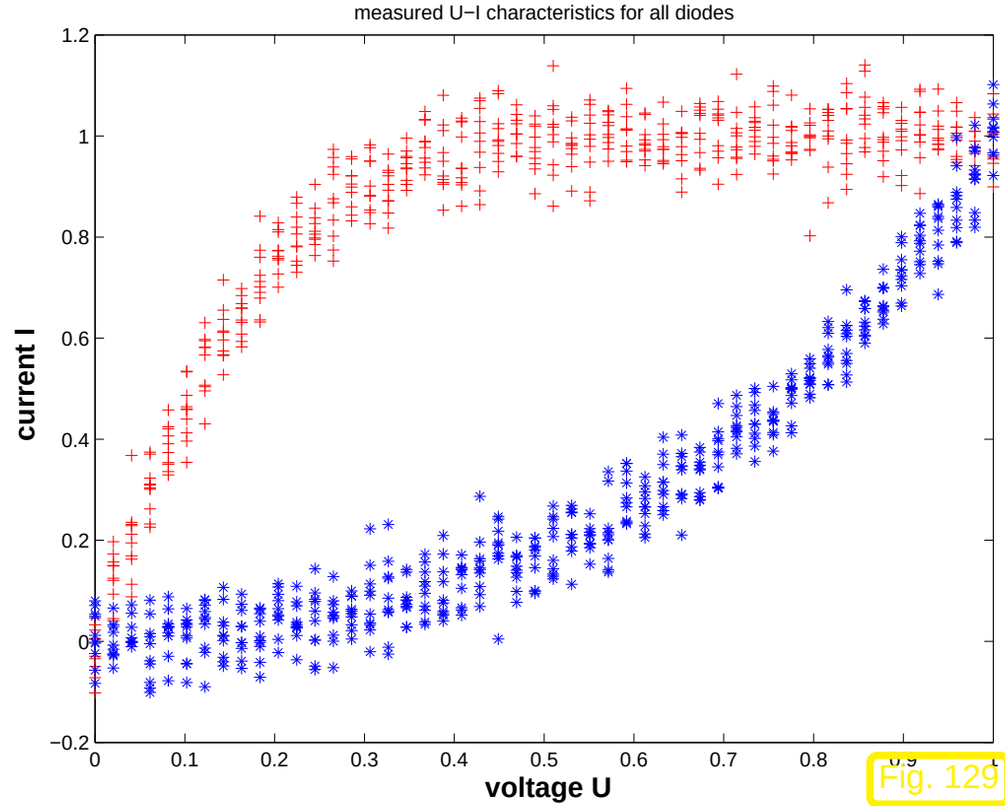


Fig. 129

R. Hiptmair

rev 38355, November 29, 2011

Code 6.5.19: Generation of synthetic perturbed U-I characteristics

```
% Generate synthetic measurement curves with random multiplicative and additive
% perturbations supposed to reflect measurement errors and manufacturing tolerances

% Voltage-current characteristics of both kinds of diodes
i1 = @(u) (2./(1+exp(-10*u)) - 1);
i2 = @(u) ((exp(3*u)-1)/(exp(3)-1));
% Simulated measurements
m = 50; % number of measurements for different input voltages
n = 10; % no.  of diodes of each kind
na = 0.05; % level of additive noise (normally distributed)
```

```
nm = 0.02; % level of multiplicative noise (normally distributed)

uvals = (0:1/(m-1):1);
D1 = (1+nm*randn(n,m)).*(i1(repmat(uvals,n,1)))+na*randn(n,m);
D2 = (1+nm*randn(n,m)).*(i2(repmat(uvals,n,1)))+na*randn(n,m);
A = ([D1;D2])'; A = A(1:size(A,1),randperm(1:size(A,2)));
```

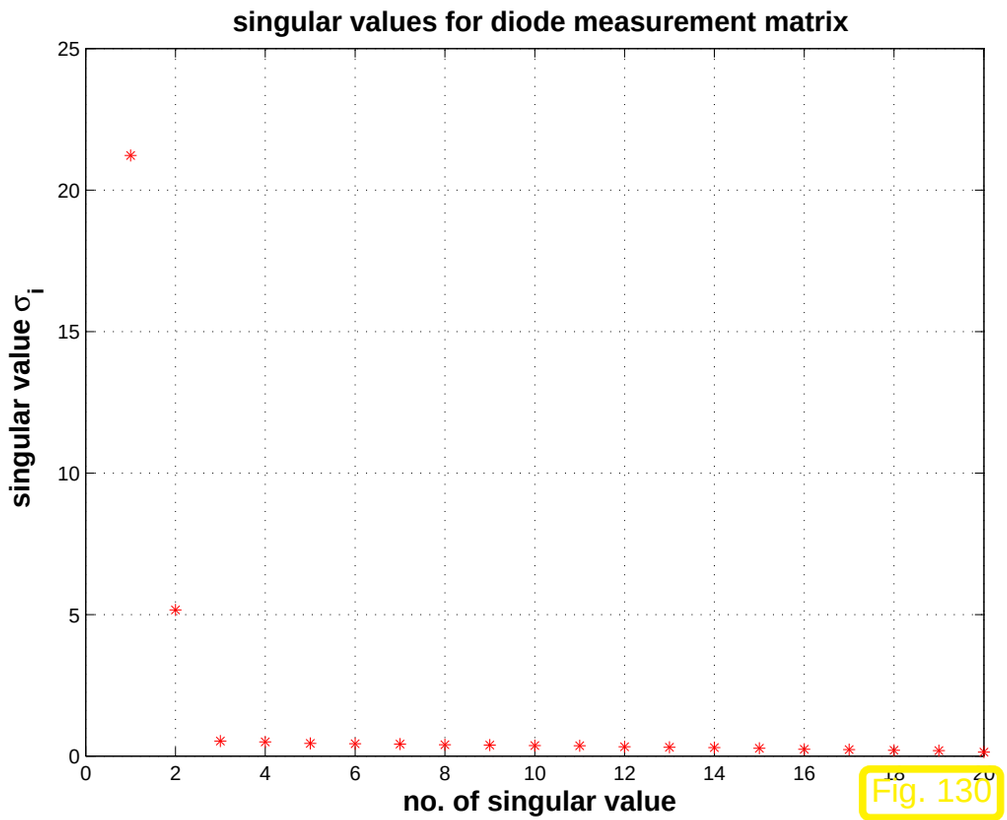


Fig. 130

← distribution of singular values of matrix

two dominant singular values !

measurements display linear correlation with **two** principal components

two types of diodes in batch

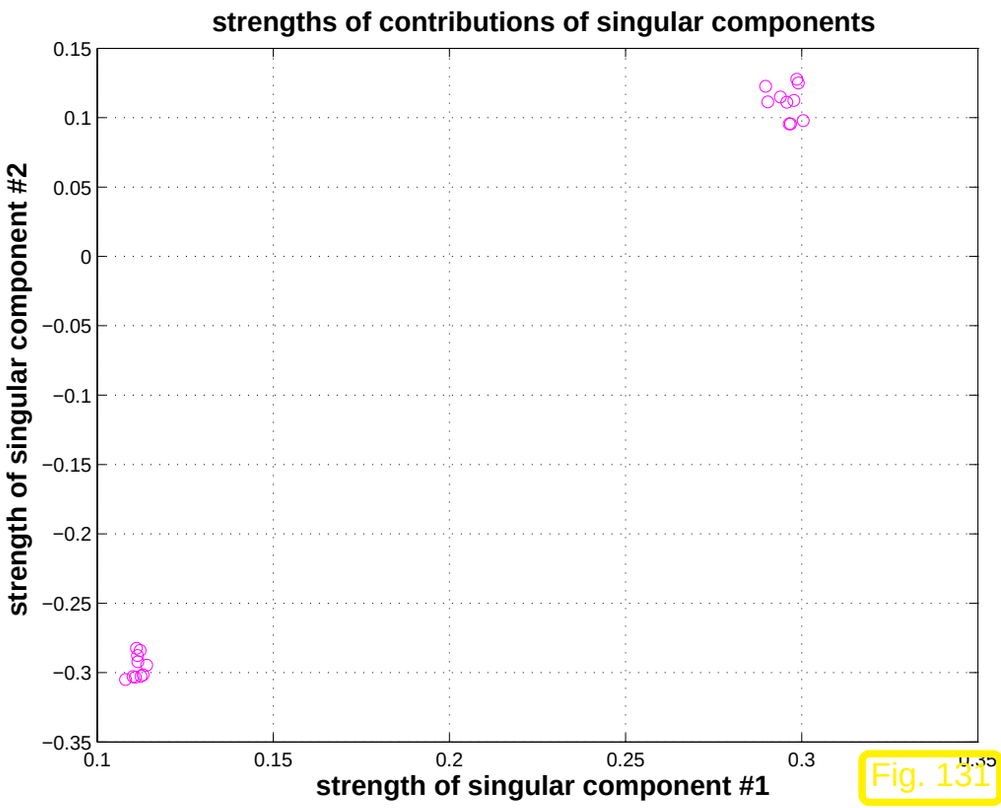


Fig. 131

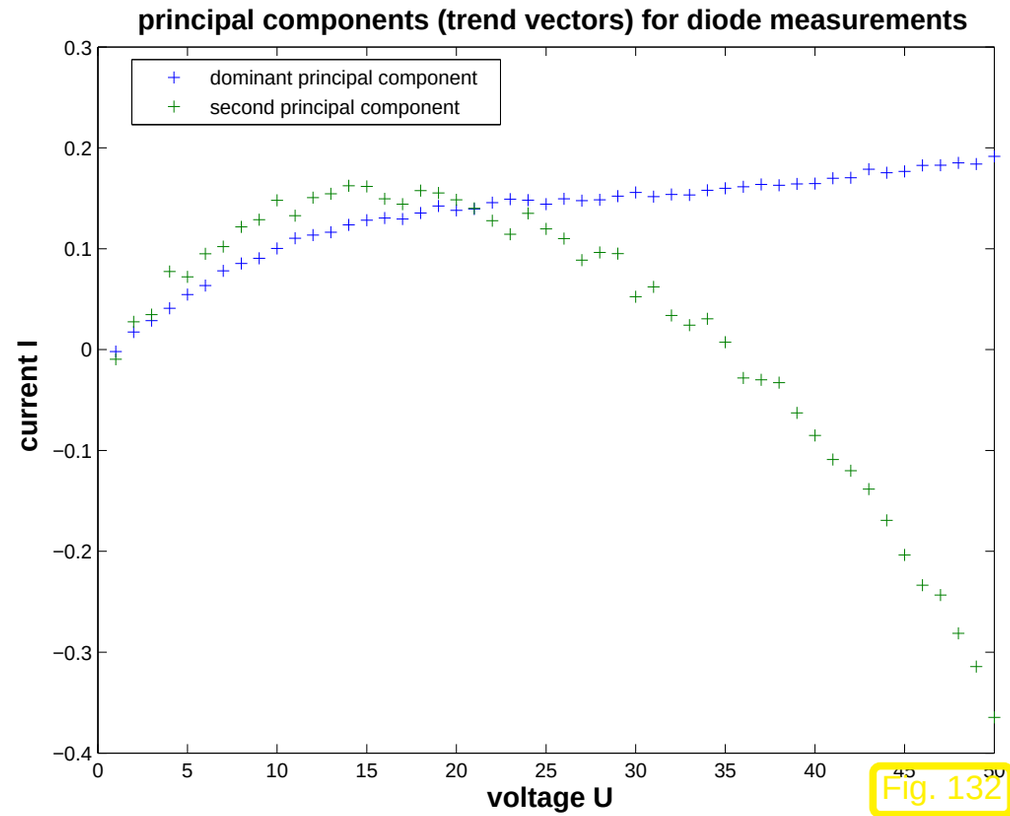


Fig. 132

◇

Example 6.5.29 (PCA of stock prices). $\rightarrow$ Ex. 6.5.1

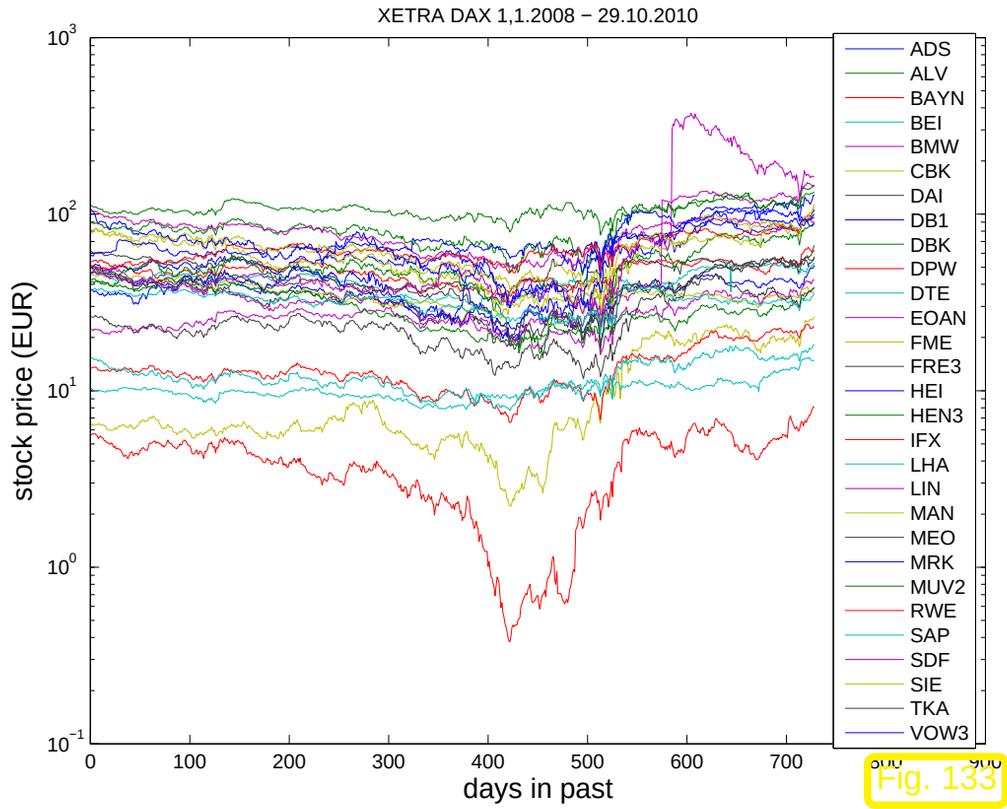


Fig. 133

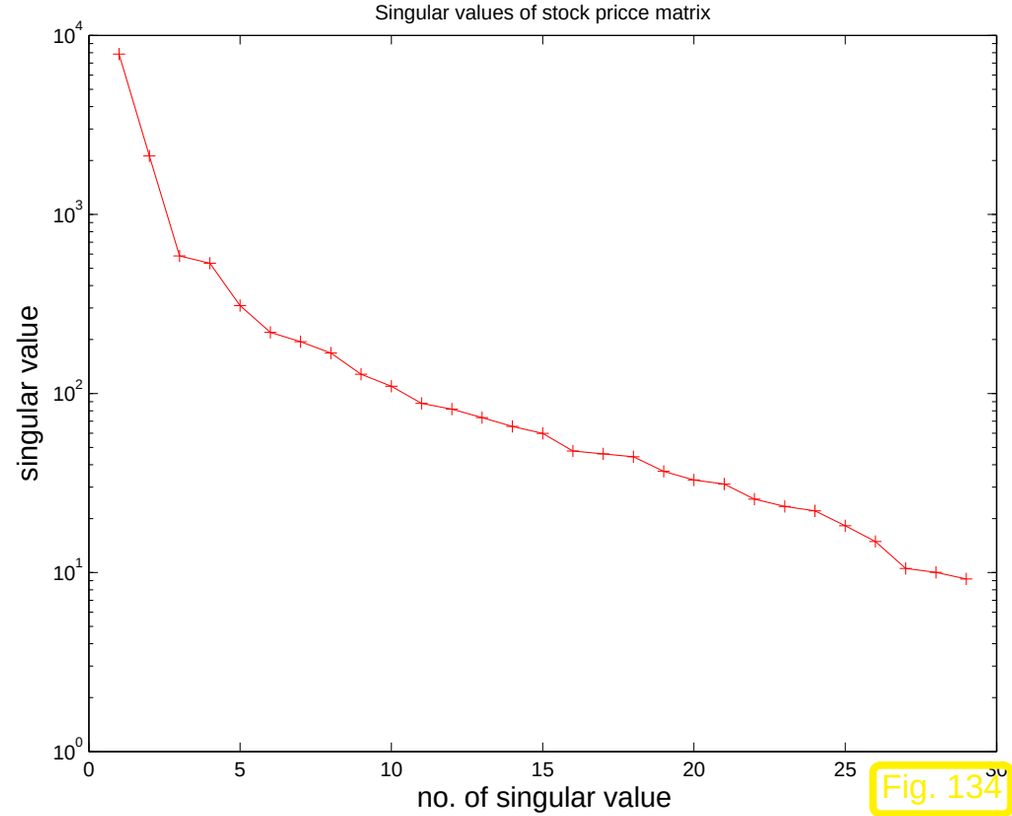


Fig. 134

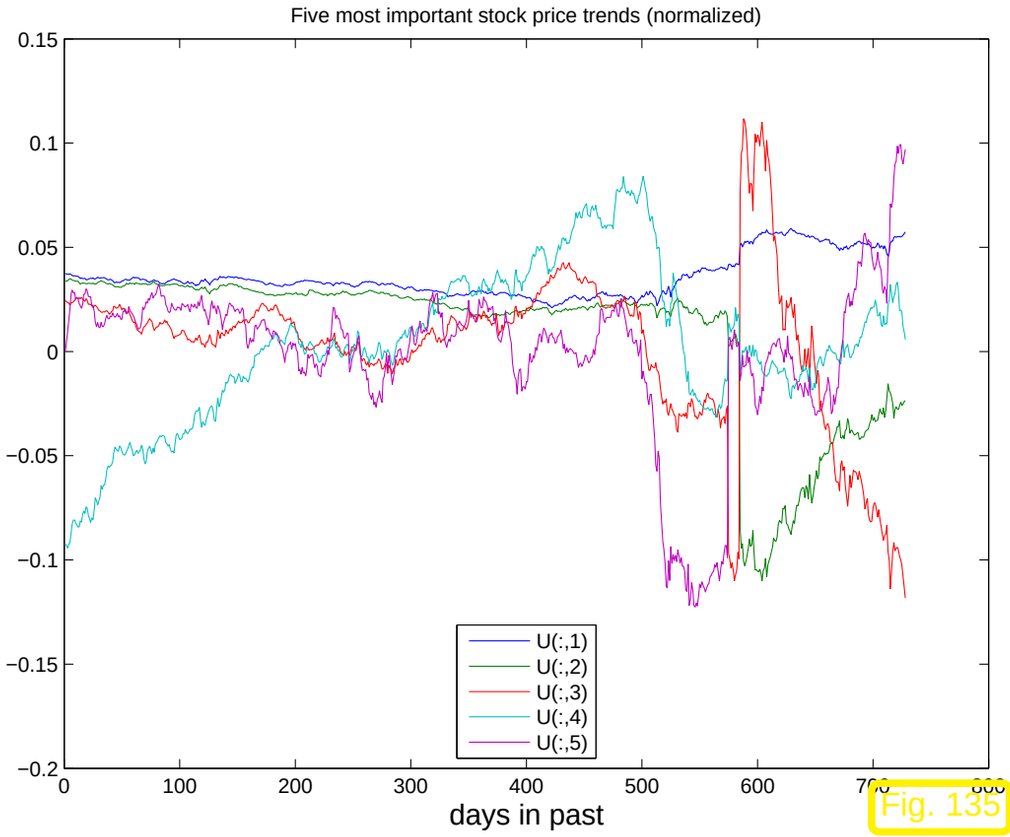


Fig. 135

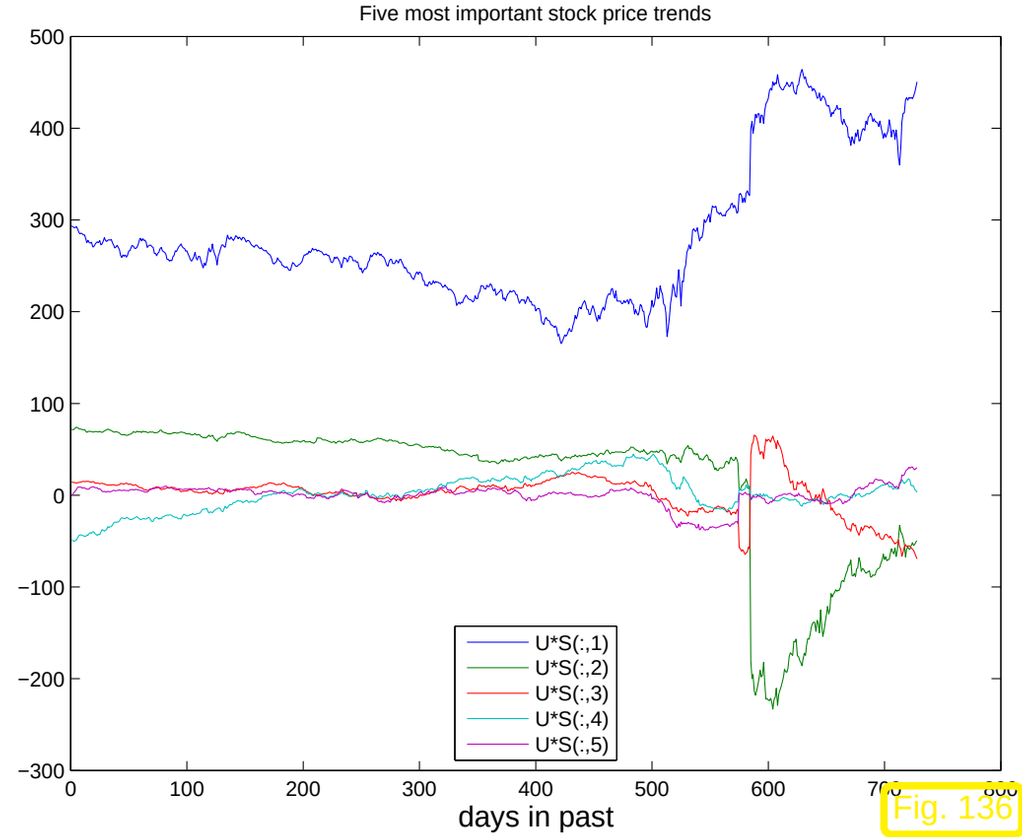


Fig. 136

R. Hiptmair

rev 38355, November 29, 2011

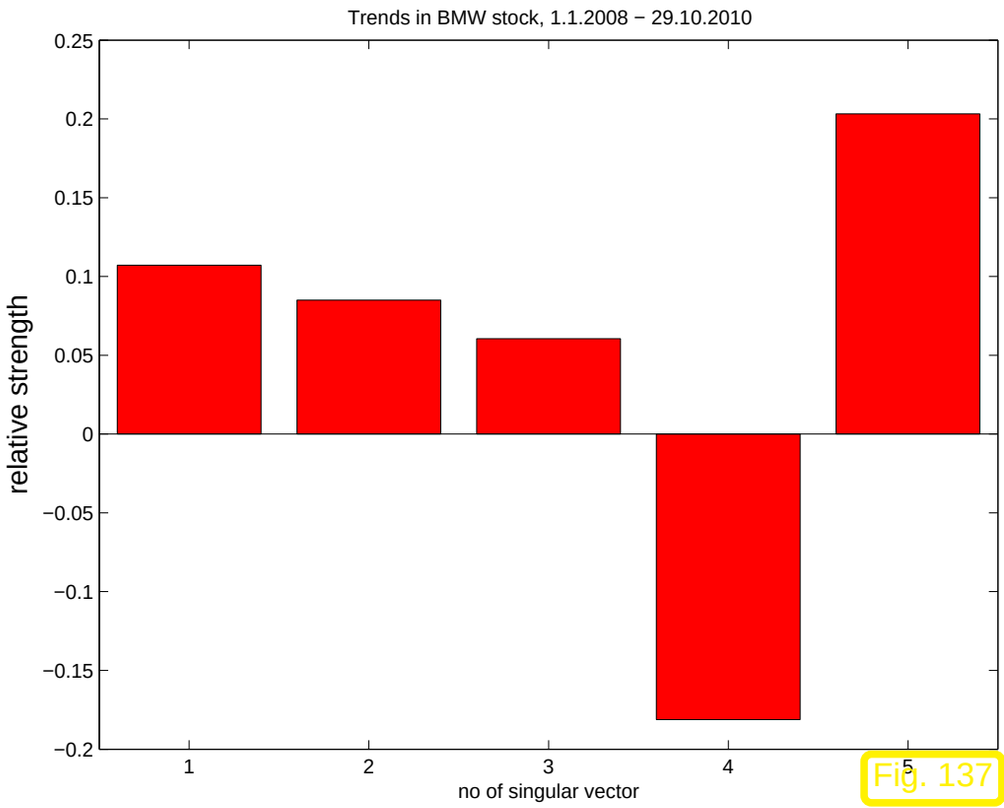


Fig. 137

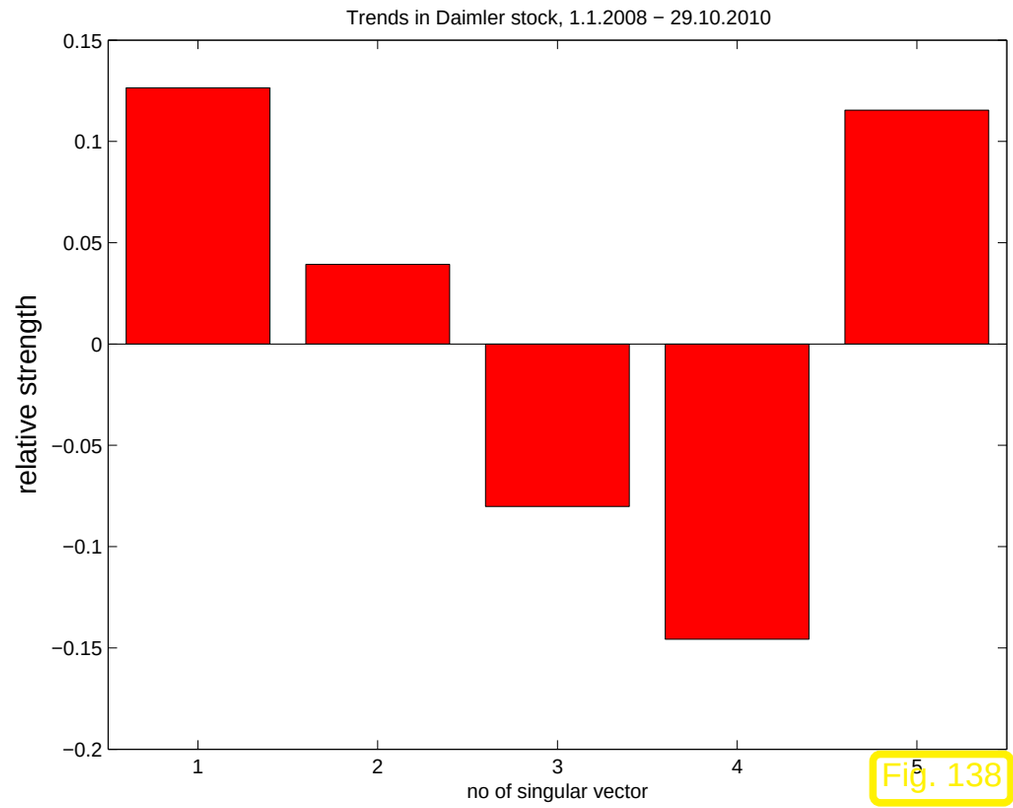


Fig. 138

R. Hiptmair

rev 38355, November 29, 2011

◇

- Application of SVD: extrema of quadratic forms on the unit sphere

- Application of SVD: *best low rank approximation*

Theorem 6.5.36 (best low rank approximation). $\rightarrow$ *[27, Thm. 11.6]*

Let $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathsf{H}}$ *be the SVD of* $\mathbf{A} \in \mathbb{K}^{m.n}$ *(*$\rightarrow$ *Thm. 6.5.6). For* $1 \leq k \leq \operatorname{rank}(\mathbf{A})$ *set* $\mathbf{U}_k := \left[\mathbf{u}_{\cdot,1}, \dots, \mathbf{u}_{\cdot,k}\right] \in \mathbb{K}^{m,k}$, $\mathbf{V}_k := \left[\mathbf{v}_{\cdot,1}, \dots, \mathbf{v}_{\cdot,k}\right] \in \mathbb{K}^{n,k}$, $\mathbf{\Sigma}_k := \operatorname{diag}(\sigma_1, \dots, \sigma_k) \in \mathbb{K}^{k,k}$. *Then, for* $\|\cdot\| = \|\cdot\|_F$ *and* $\|\cdot\| = \|\cdot\|_2$*, holds true*

$$\left\|\mathbf{A} - \mathbf{U}_k\Sigma_k\mathbf{V}_k^{\mathsf{H}}\right\| \leq \|\mathbf{A} - \mathbf{F}\| \quad \forall \mathbf{F} \in \mathcal{R}_k(m,n) \ .$$

Example 6.5.39 (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

R. Hiptmair

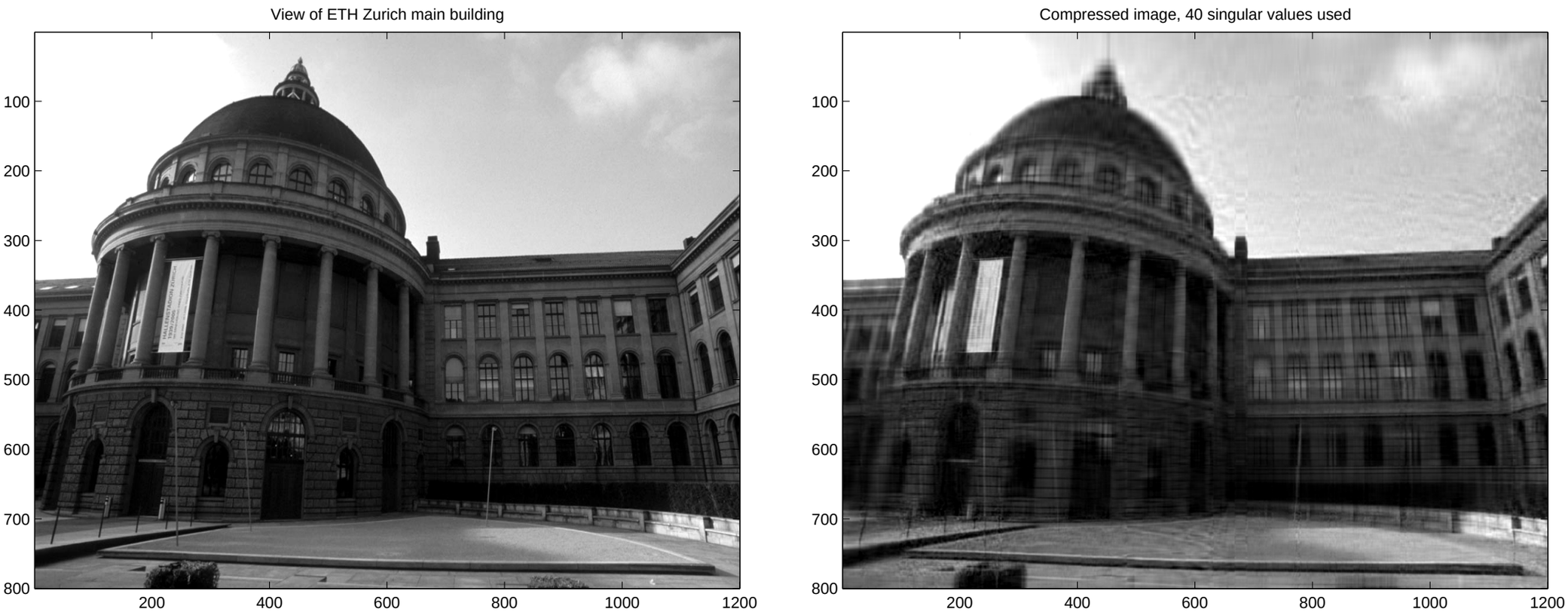
rev 38355, November 29, 2011

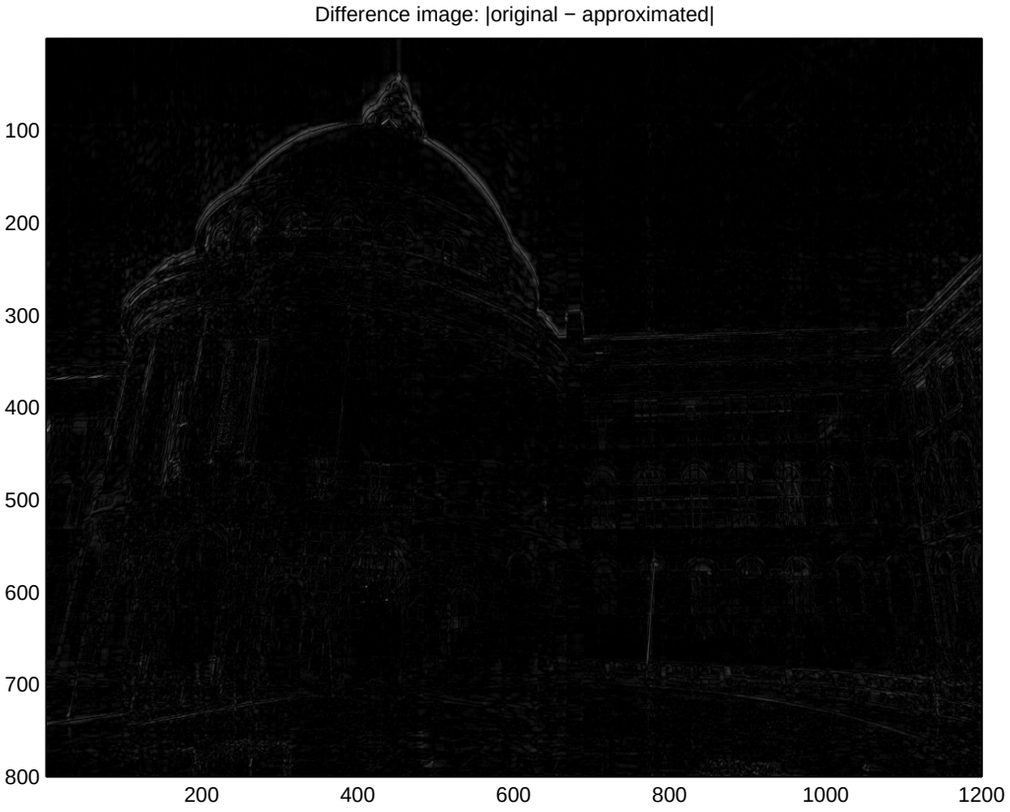
Thm. 6.5.37 $\succ$ best low rank approximation of image: $\widetilde{\mathbf{A}} = \mathbf{U}_k\Sigma_k\mathbf{V}^\top$

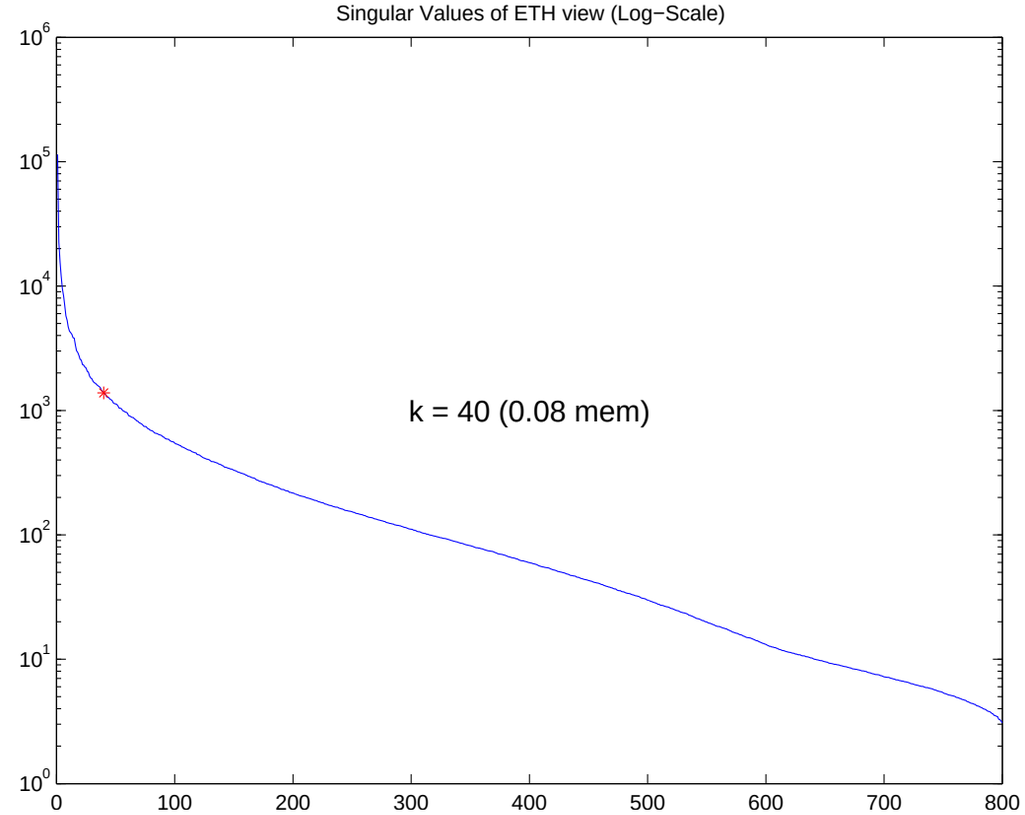
Code 6.5.40: SVD based image compression

```
P = double(imread('eth.pbm'));
[m,n] = size(P); [U,S,V] = svd(P); s = diag(S);
k = 40; S(k+1:end,k+1:end) = 0; PC = U*S*V';

figure('position',[0 0 1600 1200]); col = [0:1/215:1]'*[1,1,1];
subplot(2,2,1); image(P); title('original image'); colormap(col);
```

```
subplot(2,2,2); image(PC); title('compressed (40 S.V.)');
  colormap(col);
subplot(2,2,3); image(abs(P-PC)); title('difference');
  colormap(col);
subplot(2,2,4); cla; semilogy(s); hold on; plot(k,s(k),'ro');
```







◇

R. Hiptmair

rev 38355, November 29, 2011

7 Least Squares

Example 7.0.1 (Least squares data fitting).

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

$$(x_1^*, \dots, x_n^*) = \operatorname*{argmin}_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^{m} |f(x_1, \dots, x_n; t_i) - y_i|^2 \,. \tag{7.0.2}$$

◇

Example 7.0.3 (Linear data fitting). → [13, Sect. 4.1] → [13, Ex. 4.2 & Ex. 4.3]

R. Hiptmair

rev 38355, November 17, 2011

Linear least squares fitting problem:

Given:
- data points (t_i, y_i), $\quad i = 1, \dots, m$
- 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

$$(x_1^*, \dots, x_n^*) = \operatorname*{argmin}_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^{m} \left| \sum_{j=1}^{n} x_j b_j(t_i) - y_i \right|^2 . \tag{7.0.5}$$

Example 7.0.7 (Polybomial interpolation vs. polynomial fitting).

R. Hiptmair

rev 38355, November 17, 2011

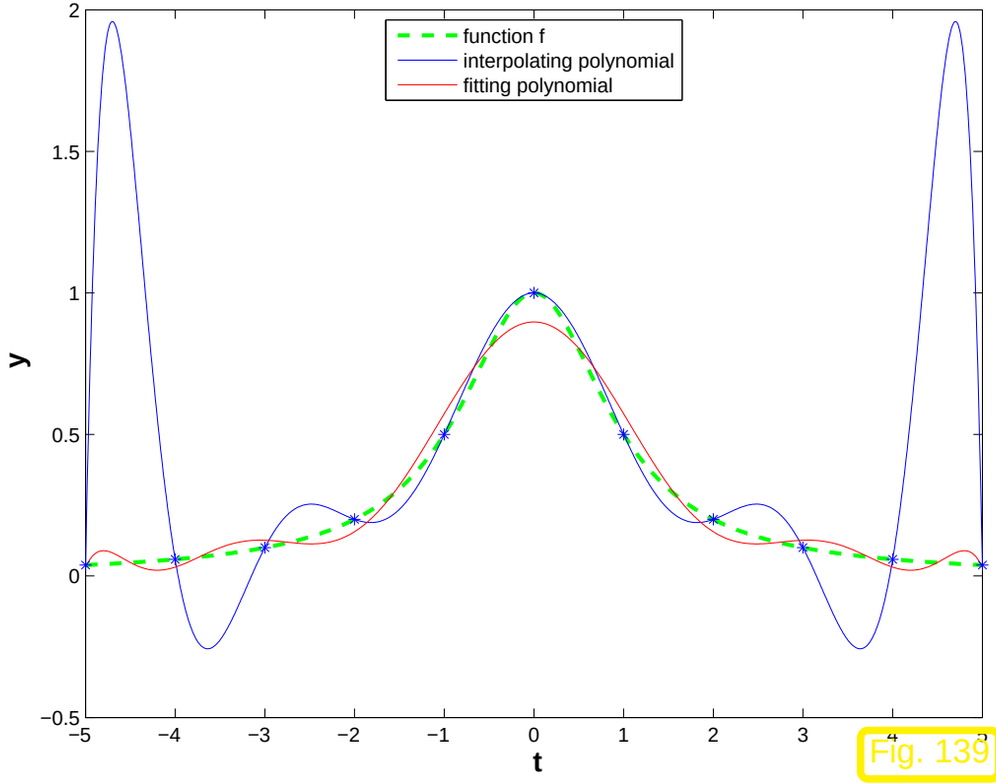


Fig. 139

Data from function $f(t) = \dfrac{1}{1+t^2}$,

- 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 $(t_j, f(t_j))$, $j = 0, \dots, 3d$, $t_j = -5 + j/3$.

◇

R. Hiptmair

rev 38355, November 17, 2011

(Full rank linear) least squares problem: [13, Sect. 4.2]

given: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m, n \in \mathbb{N}$, $m \geq n$, $\operatorname{rank}(\mathbf{A}) = n$, $\mathbf{b} \in \mathbb{R}^m$,

find: $\mathbf{x} \in \mathbb{R}^n$ such that

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \inf\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbb{R}^n\} \tag{7.0.14}$$

$$\Updownarrow$$

$$\mathbf{x} = \operatorname*{argmin}_{\mathbf{y} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2$$

(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

$$\text{(i) } \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \inf\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbb{R}^n\}, \tag{7.0.16}$$

(ii) $\|\mathbf{x}\|_2$ is minimal under the condition (i).

MATLAB "black-box" solver for general linear least squares problems (7.0.16)

`x = A\b` ("backslash") solves (7.0.16) for $\mathbf{A} \in \mathbb{K}^{m,n}$, $m \neq n$.

7.1 Normal Equations [13, Sect. 4.2], [35, Ch. 11]

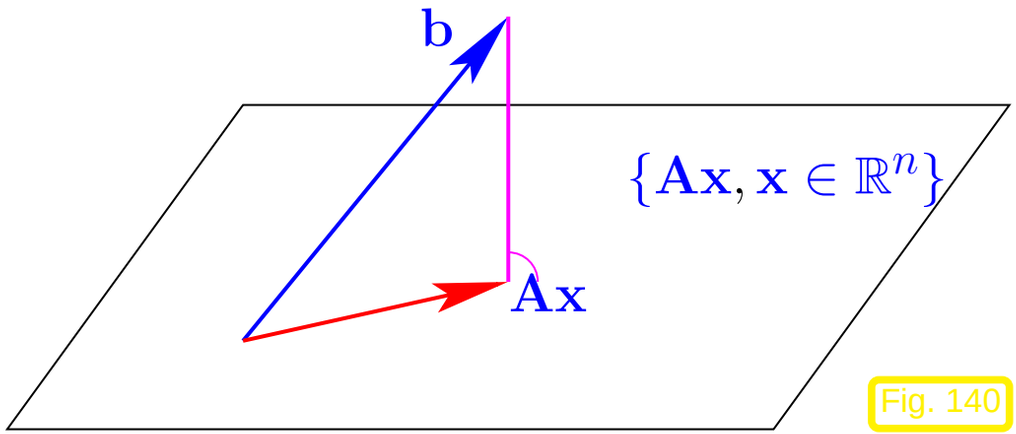


Fig. 140

Geometric interpretation of linear least squares problem (7.0.16):

$\mathbf{x} \hat{=}$ orthogonal projection of $\mathbf{b}$ on the subspace $\mathrm{Im}(\mathbf{A}) := \mathrm{Span}\left\{(\mathbf{A})_{:,1}, \dots, (\mathbf{A})_{:,n}\right\}$.

Example 7.1.4 (Instability of normal equations). $\rightarrow$ [13, Ex. 4.12]

Caution: loss of information in the computation of $\mathbf{A}^{\mathsf{H}}\mathbf{A}$, e.g.

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

$$\Rightarrow \mathbf{A}^{\mathsf{H}}\mathbf{A} = \begin{pmatrix} 1+\delta^2 & 1 \\ 1 & 1+\delta^2 \end{pmatrix}$$

```
>> A = [1 1;...
        sqrt(eps) 0;...
        0     sqrt(eps)];
>> rank(A)
      ans =  2
>> rank(A'*A)
      ans =  1
```

7.2 Orthogonal Transformation Methods [13, Sect. 4.4.2]

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.

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \left\|\mathbf{Q}(\mathbf{R}\mathbf{x} - \mathbf{Q}^{\mathsf{H}}\mathbf{b})\right\|_2 = \left\|\mathbf{R}\mathbf{x} - \widetilde{\mathbf{b}}\right\|_2 \quad , \quad \widetilde{\mathbf{b}} := \mathbf{Q}^{\mathsf{H}}\mathbf{b} \; .$$

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \to \min \quad \Leftrightarrow \quad \left\| \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} - \begin{pmatrix} \widetilde{b}_1 \\ \vdots \\ \\ \\ \vdots \\ \widetilde{b}_m \end{pmatrix} \right\|_2 \to \min \; .$$

R. Hiptmair

rev 38355, November 18, 2011

$$\mathbf{x} = \left(\mathbf{R} \right)^{-1} \begin{pmatrix} \widetilde{b}_1 \\ \vdots \\ \widetilde{b}_n \end{pmatrix} \quad , \quad \text{residuum} \quad \mathbf{r} = \mathbf{Q} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \widetilde{b}_{n+1} \\ \vdots \\ \widetilde{b}_m \end{pmatrix} .$$

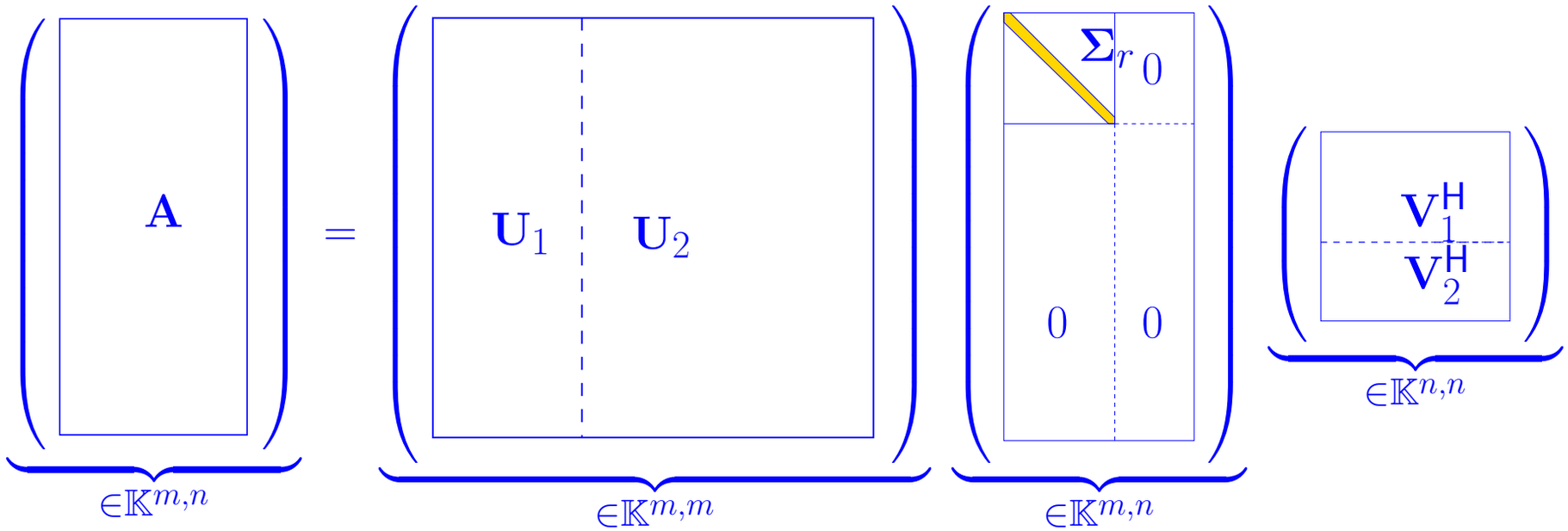
Code 7.2.1: QR-based solver for full rank linear least squares problem (7.0.14)

```
function [x,res] = qrlsqsolve(A,b)
% Solution of linear least squares problem (7.0.14) by means of QR-decomposition
% Note:  A ∈ ℝ^{m,n} with m > n, rank(A) = n is assumed
[m,n] = size(A);
R = triu(qr([A,b],0)), % economical QR-decomposition of extended matrix
x = R(1:n,1:n)\R(1:n,n+1); % x̂ = (R)^{-1}_{1:n,1:n}(Q^T b)_{1:n}
res = R(n+1,n+1); % = ‖Ax̂ − b‖_2 (why ?)
```

Alternative: Solving linear least squares problem (7.0.16) by SVD ($\rightarrow$ Def. 6.5.8)

Most general setting: $\mathbf{A} \in \mathbb{K}^{m,n}$, $\operatorname{rank}(\mathbf{A}) = r \leq \min\{m,n\}$:

SVD: $\mathbf{A} = [\,\mathbf{U}_1 \quad \mathbf{U}_2\,] \begin{pmatrix} \mathbf{\Sigma}_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^{\mathsf{H}} \\ \mathbf{V}_2^{\mathsf{H}} \end{pmatrix}$



(7.2.2)

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \left\| [\mathbf{U}_1\ \mathbf{U}_2] \begin{pmatrix} \mathbf{\Sigma}_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^{\mathsf{H}} \\ \mathbf{V}_2^{\mathsf{H}} \end{pmatrix} \mathbf{x} - \mathbf{b} \right\|_2 = \left\| \begin{pmatrix} \mathbf{\Sigma}_r \mathbf{V}_1^{\mathsf{H}} \mathbf{x} \\ 0 \end{pmatrix} - \begin{pmatrix} \mathbf{U}_1^{\mathsf{H}} \mathbf{b} \\ \mathbf{U}_2^{\mathsf{H}} \mathbf{b} \end{pmatrix} \right\|_2 \tag{7.2.3}$$

▶ solution $\boxed{\mathbf{x} = \mathbf{V}_1 \mathbf{\Sigma}_r^{-1} \mathbf{U}_1^{\mathsf{H}} \mathbf{b}}$, $\|\mathbf{r}\|_2 = \left\| \mathbf{U}_2^{\mathsf{H}} \mathbf{b} \right\|_2$. (7.2.6)

Practical implementation:

"numerical rank" test:

$$r = \max\{i\colon \sigma_i/\sigma_1 > \texttt{tol}\}$$

Code 7.2.7: Solving LSQ problem via SVD

```
function y = lsqsvd(A,b)
[U,S,V] = svd(A,0);
sv = diag(S);
r = max(find(sv/sv(1) > eps));
y = V(:,1:r)*(diag(1./sv(1:r))*...
            (U(:,1:r)'*b));
```

7.3 Total Least Squares

R. Hiptmair

rev 38355, November 18, 2011

Total least squares problem:

Given: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, $\operatorname{rank}(\mathbf{A}) = n$, $\mathbf{b} \in \mathbb{R}^m$,

find: $\widehat{\mathbf{A}} \in \mathbb{R}^{m,n}$, $\widehat{\mathbf{b}} \in \mathbb{R}^m$ with

$$\left\| \underbrace{[\mathbf{A} \quad \mathbf{b}]}_{=:\mathbf{C}} - \underbrace{\left[\widehat{\mathbf{A}} \quad \widehat{\mathbf{b}}\right]}_{=:\widehat{\mathbf{C}}} \right\|_F \to \min \quad , \quad \widehat{\mathbf{b}} \in \operatorname{Im}(\widehat{\mathbf{A}}) \; . \tag{7.3.1}$$

$$\widehat{\mathbf{b}} \in \mathrm{Im}(\widehat{\mathbf{A}}) \quad \Rightarrow \quad \mathrm{rank}(\widehat{\mathbf{C}}) = n \quad \blacktriangleright \quad (7.3.1) \quad \Rightarrow \quad \min_{\mathrm{rank}(\widehat{\mathbf{C}})=n} \left\| \mathbf{C} - \widehat{\mathbf{C}} \right\|_F .$$

Code 7.3.5: Total least squares via SVD

```
function x = lsqtotal(A,b);
% computes only solution x of fitted consistent LSE
[m,n]=size(A);
[U, Sigma, V] = svd([A,b]); % see (7.3.2)
s = V(n+1,n+1);
if s == 0, error('No solution'); end
x = -V(1:n,n+1)/s; % see (7.3.4)
```

R. Hiptmair

rev 38355, December 8, 2010

7.4 Constrained Least Squares

Linear least squares problem with linear constraint:

Given: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, $\operatorname{rank}(\mathbf{A}) = n$, $\mathbf{b} \in \mathbb{R}^m$,
$\mathbf{C} \in \mathbb{R}^{p,n}$, $p < n$, $\operatorname{rank}(\mathbf{C}) = p$, $\mathbf{d} \in \mathbb{R}^p$

Find: $\mathbf{x} \in \mathbb{R}^n$ with

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \to \min \quad , \quad \boxed{\mathbf{C}\mathbf{x} = \mathbf{d}} \; . \tag{7.4.1}$$

Linear constraint

7.4.1 Solution via normal equations

R. Hiptmair

rev 38355, December 8, 2010

Idea: coupling the constraint using the Lagrange multiplier $\mathbf{m} \in \mathbb{R}^p$

$$\mathbf{x} = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \max_{\mathbf{m}\in\mathbb{R}^p} L(\mathbf{x},\mathbf{m}) \; , \tag{7.4.2}$$

$$L(\mathbf{x},\mathbf{m}) := \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \mathbf{m}^H(\mathbf{C}\mathbf{x} - \mathbf{d}) \; . \tag{7.4.3}$$

Solution of min-max problem:

saddle point

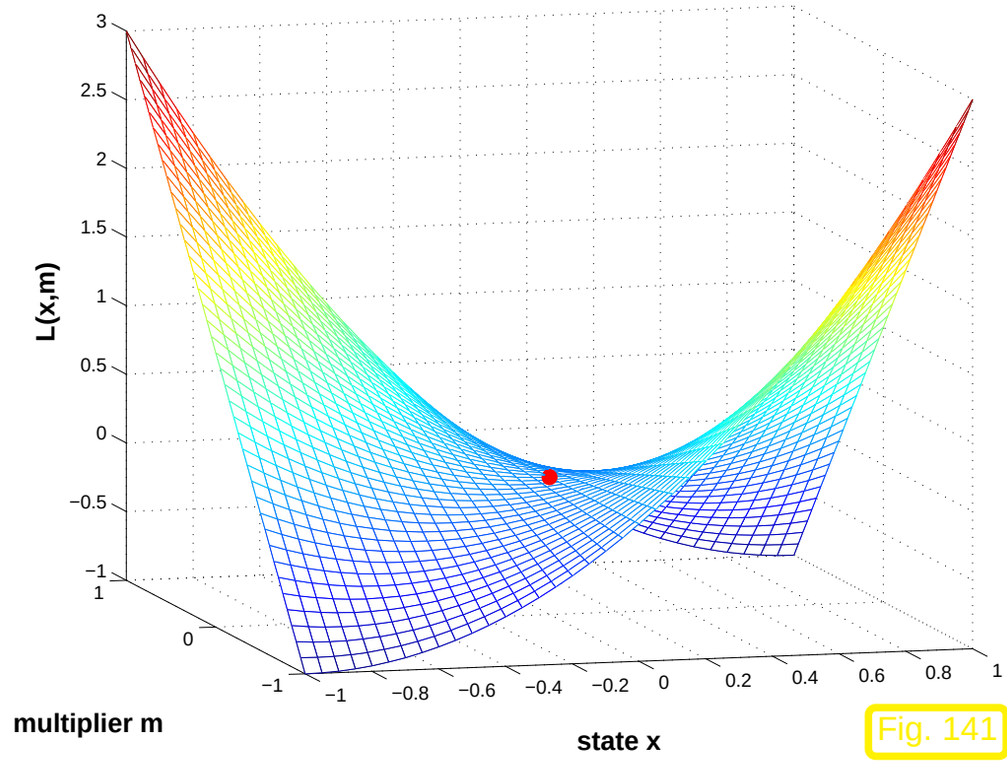


Fig. 141

R. Hiptmair

rev 38355, December 8, 2010

Necessary (and sufficient) condition for the minimizer ($\rightarrow$ Section 7.1)

$$\frac{\partial L}{\partial \mathbf{x}}(\mathbf{x},\mathbf{m}) = \mathbf{A}^H(\mathbf{A}\mathbf{x}-\mathbf{b}) + \mathbf{C}^H\mathbf{m} \stackrel{!}{=} 0 \quad , \quad \frac{\partial L}{\partial \mathbf{m}}(\mathbf{x},\mathbf{m}) = \mathbf{C}\mathbf{x}-\mathbf{d} \stackrel{!}{=} 0. \tag{7.4.4}$$

$$\Downarrow$$

$$\begin{pmatrix} \mathbf{A}^H\mathbf{A} & \mathbf{C}^H \\ \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{m} \end{pmatrix} = \begin{pmatrix} \mathbf{A}^H\mathbf{b} \\ \mathbf{d} \end{pmatrix} \tag{7.4.5}$$

Augmented normal equations
(matrix saddle point problem)

7.4.2 Solution via SVD

① Compute orthonormal basis of $\mathrm{Ker}(\mathbf{C})$ using SVD ($\rightarrow$ Lemma 6.5.14, (7.2.2)):

$$\mathbf{C} = \mathbf{U}\,[\Sigma\ 0]\begin{bmatrix}\mathbf{V}_1^H\\ \mathbf{V}_2^H\end{bmatrix} \quad , \quad \mathbf{U}\in\mathbb{R}^{p,p},\ \Sigma\in\mathbb{R}^{p,p},\ \mathbf{V}_1\in\mathbb{R}^{n,p},\ \mathbf{V}_2\in\mathbb{R}^{n,n-p}$$

$$\blacktriangleright \qquad \mathrm{Ker}(\mathbf{C}) = \mathrm{Im}(\mathbf{V}_2)\ .$$

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): $\qquad \mathbf{x} = \mathbf{x}_0 + \mathbf{V}_2\mathbf{y}, \quad \mathbf{y}\in\mathbb{R}^{n-p}.$

② Insert this representation in (7.4.1) ➤ standard linear least squares

$$\|\mathbf{A}(\mathbf{x}_0+\mathbf{V}_2\mathbf{y})-\mathbf{b}\|_2 \rightarrow \min \quad \Leftrightarrow \quad \|\mathbf{A}\mathbf{V}_2\mathbf{y}-(\mathbf{b}-\mathbf{A}\mathbf{x}_0)\| \rightarrow \min\ .$$

R. Hiptmair

rev 38355, November 17, 2011

7.5 Non-linear Least Squares [13, Ch. 6]

Non-linear least squares problem

Given: $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^m, \quad m, n \in \mathbb{N},\ m > n.$

Find: $\mathbf{x}^* \in D$: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in D} \Phi(\mathbf{x})\ , \quad \Phi(\mathbf{x}) := \frac{1}{2}\|F(\mathbf{x})\|_2^2\ .$ (7.5.3)

Terminology: $D \hat{=}$ parameter space, $x_1, \dots, x_n \hat{=}$ parameter.

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.)

7.5.1 (Damped) Newton method

$$\Phi(\mathbf{x}^*) = \min \quad \Rightarrow \quad \mathbf{grad}\,\Phi(\mathbf{x}) = 0, \quad \mathbf{grad}\,\Phi(\mathbf{x}) := (\frac{\partial\Phi}{\partial x_1}(\mathbf{x}), \dots, \frac{\partial\Phi}{\partial x_n}(\mathbf{x}))^T \in \mathbb{R}^n.$$

Simple idea: use Newton's method ($\rightarrow$ Sect. 4.4) to determine a zero of $\mathbf{grad}\,\Phi : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$.

► For Newton iterate $\mathbf{x}^{(k)}$: Newton correction $\mathbf{s} \in \mathbb{R}^n$ from LSE

$$\underbrace{\left(DF(\mathbf{x}^{(k)})^T DF(\mathbf{x}^{(k)}) + \sum_{j=1}^{m} F_j(\mathbf{x}^{(k)}) D^2 F_j(\mathbf{x}^{(k)}) \right)}_{=H\Phi(\mathbf{x}^{(k)})} \mathbf{s} = - \underbrace{DF(\mathbf{x}^{(k)})^T F(\mathbf{x}^{(k)})}_{=\mathbf{grad}\,\Phi(\mathbf{x}^{(k)})} \,. \qquad (7.5.6)$$

R. Hiptmair

rev 38355, November 17, 2011

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

Gauss-Newton iteration (under assumption (7.5.4))

Initial guess $\mathbf{x}^{(0)} \in D$

$$\mathbf{x}^{(k+1)} := \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \left\| F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) \right\|_2 . \tag{7.5.9}$$

linear least squares problem

MATLAB-\ used to solve linear least squares problem in each step:

for $\mathbf{A} \in \mathbb{R}^{m,n}$

```
x = A\b
```

$\updownarrow$

$\mathbf{x}$ minimizer of $\|\mathbf{Ax} - \mathbf{b}\|_2$ with minimal 2-norm

Code 7.5.10: template for Gauss-Newton method

```
function x = gn(x,F,J,tol)
s = J(x)\F(x);                    %
x = x-s;
while (norm(s) > tol*norm(x))     %
  s = J(x)\F(x);                  %
  x = x-s;
end
```

Example 7.5.11 (Non-linear data fitting (II)). $\rightarrow$ Ex. 7.5.1

Non-linear data fitting problem (7.5.2) for $f(t) = x_1 + x_2 \exp(-x_3 t)$.

$$F(\mathbf{x}) = \begin{pmatrix} x_1 + x_2\exp(-x_3 t_1) - y_1 \\ \vdots \\ x_1 + x_2\exp(-x_3 t_m) - y_m \end{pmatrix} : \mathbb{R}^3 \mapsto \mathbb{R}^m \,,\; DF(\mathbf{x}) = \begin{pmatrix} 1 & e^{-x_3 t_1} & -x_2 t_1 e^{-x_3 t_1} \\ \vdots & \vdots & \vdots \\ 1 & e^{-x_3 t_m} & -x_2 t_m e^{-x_3 t_m} \end{pmatrix}$$

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)

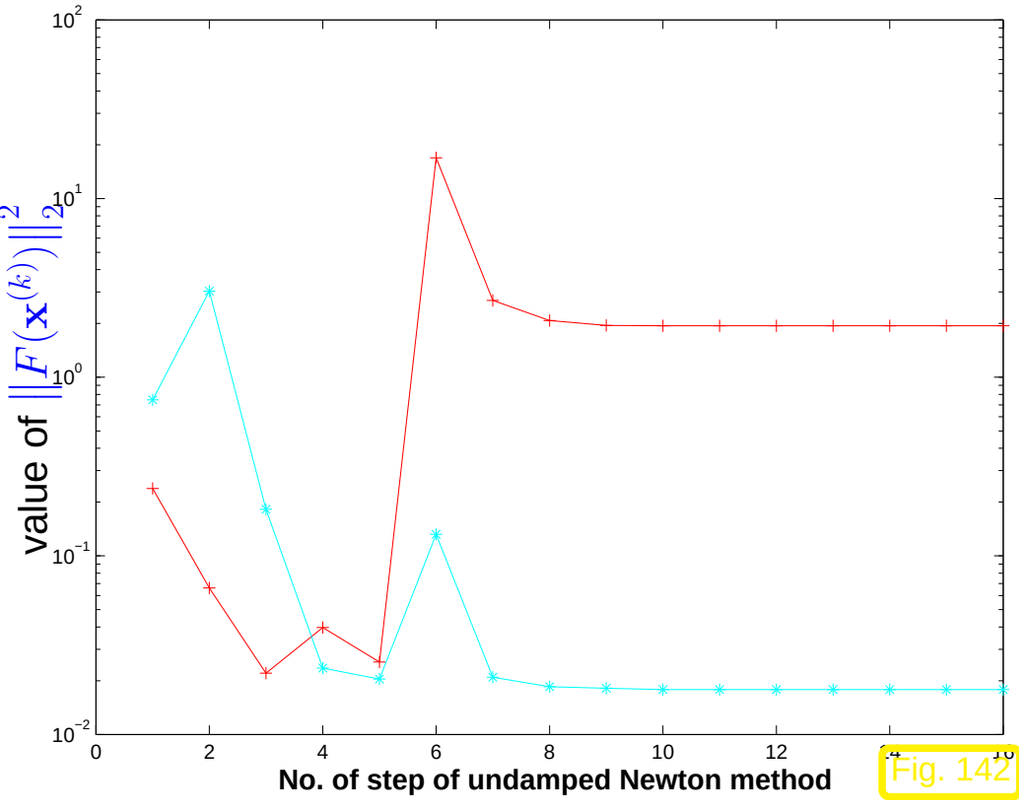


Fig. 142

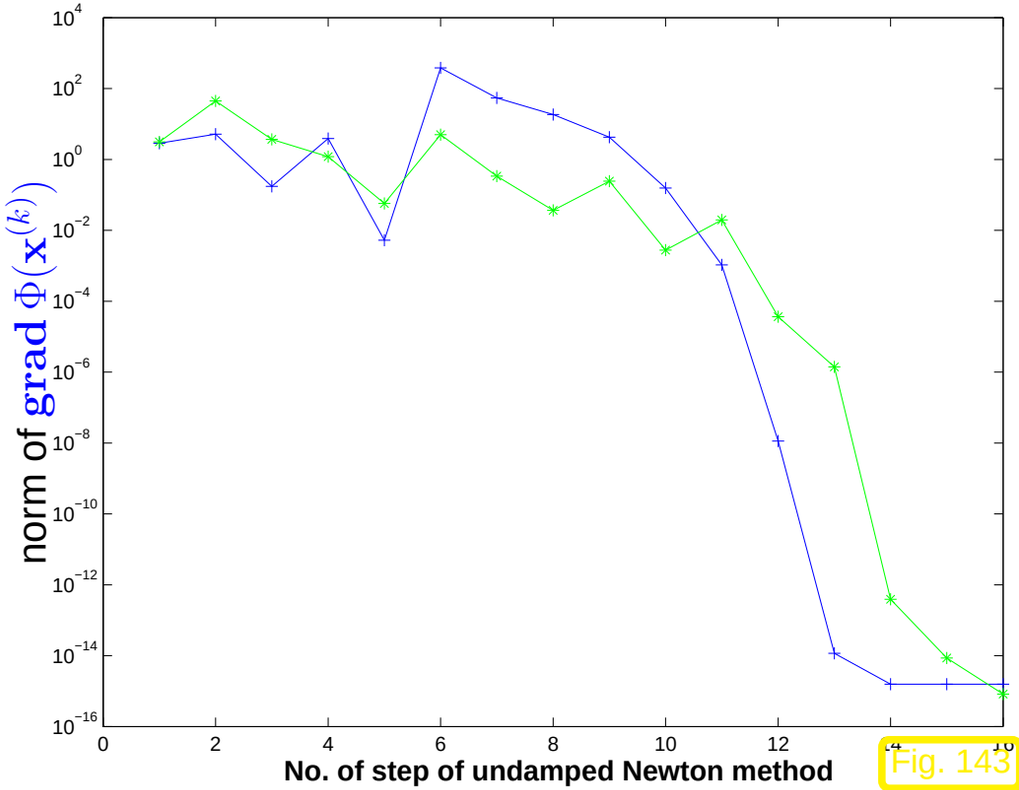


Fig. 143

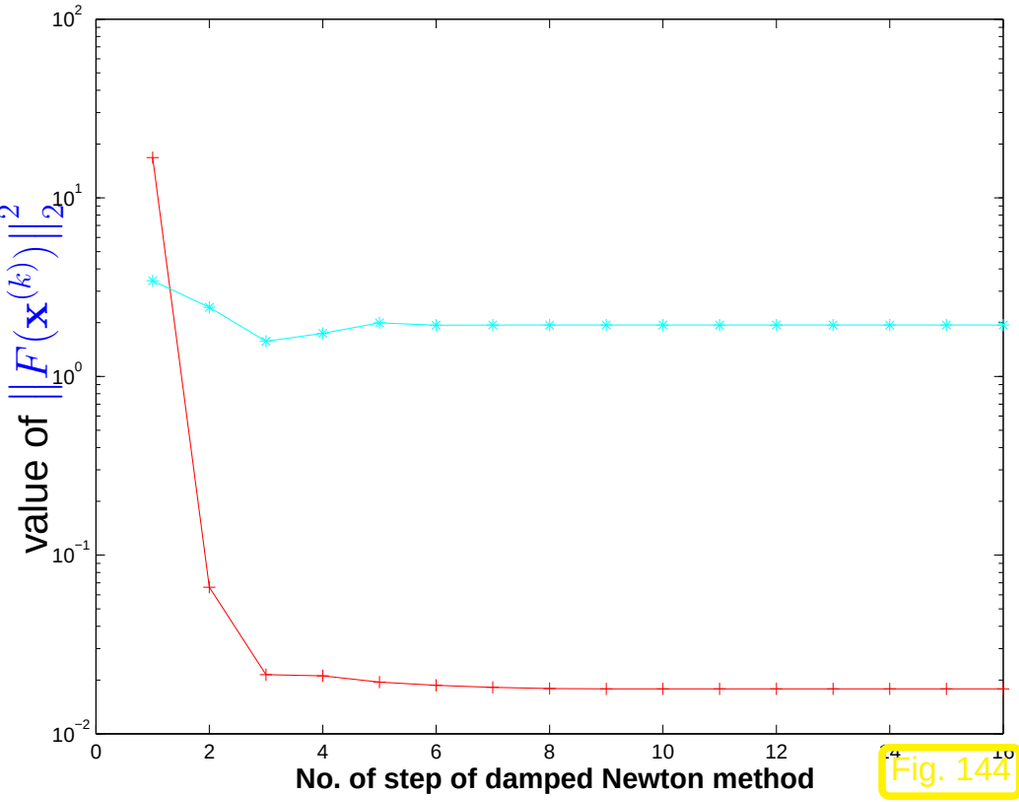


Fig. 144

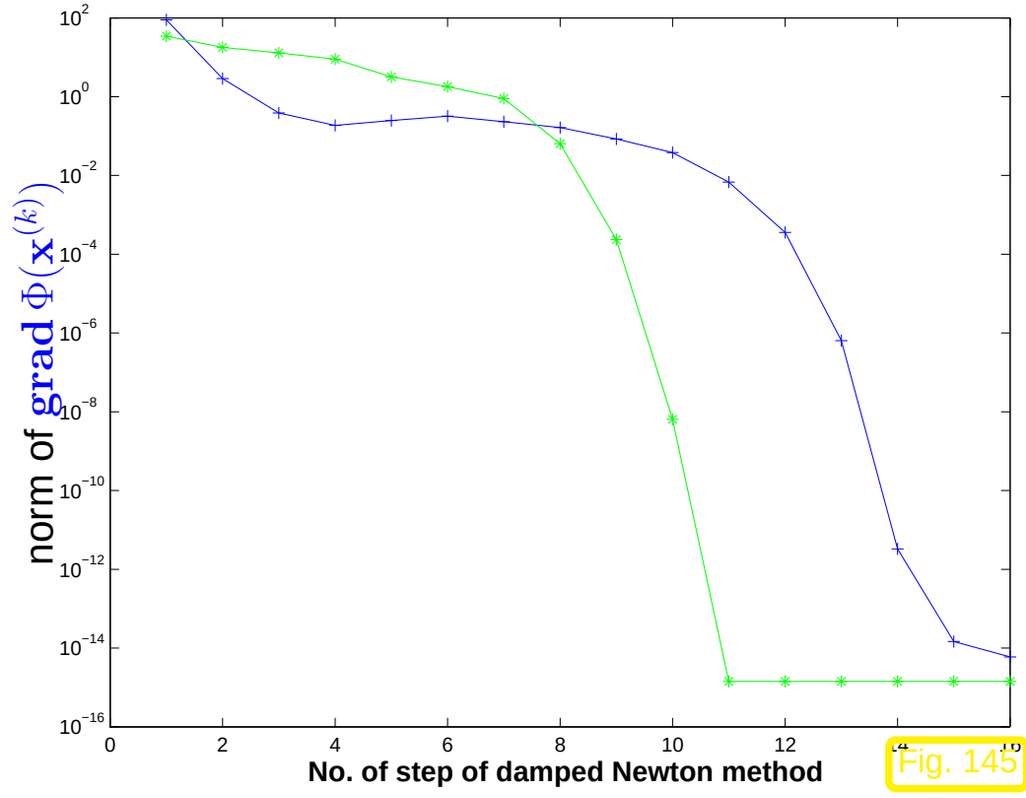


Fig. 145

R. Hiptmair

rev 38355, November 17, 2011

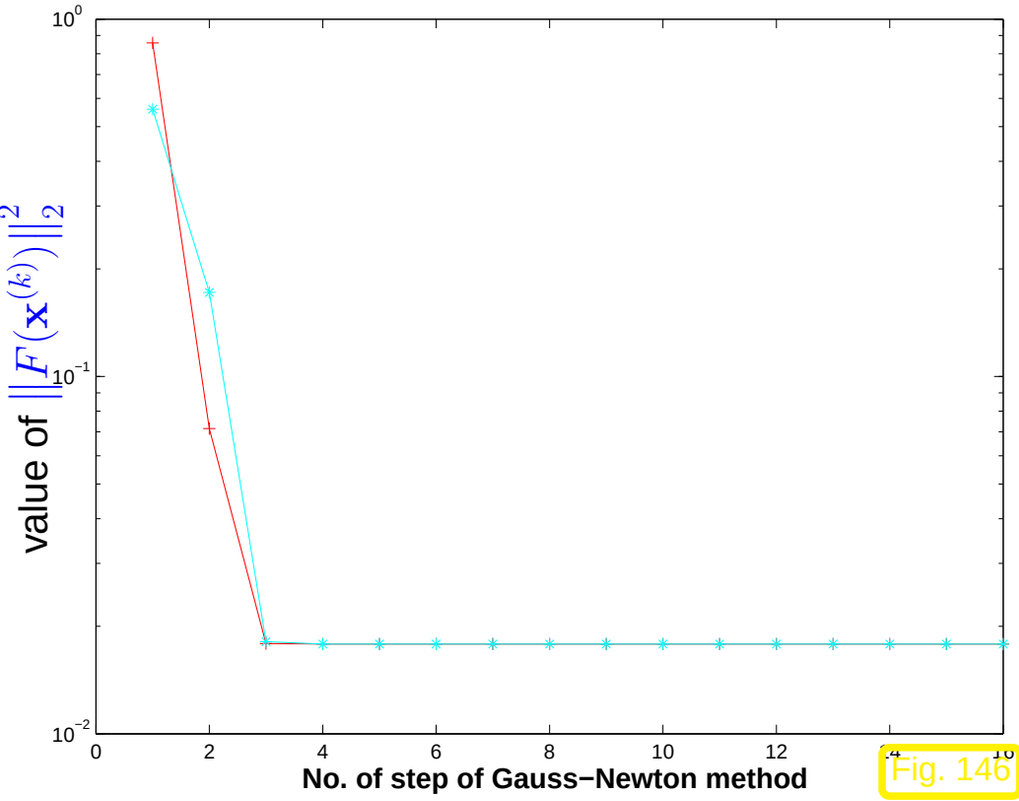


Fig. 146

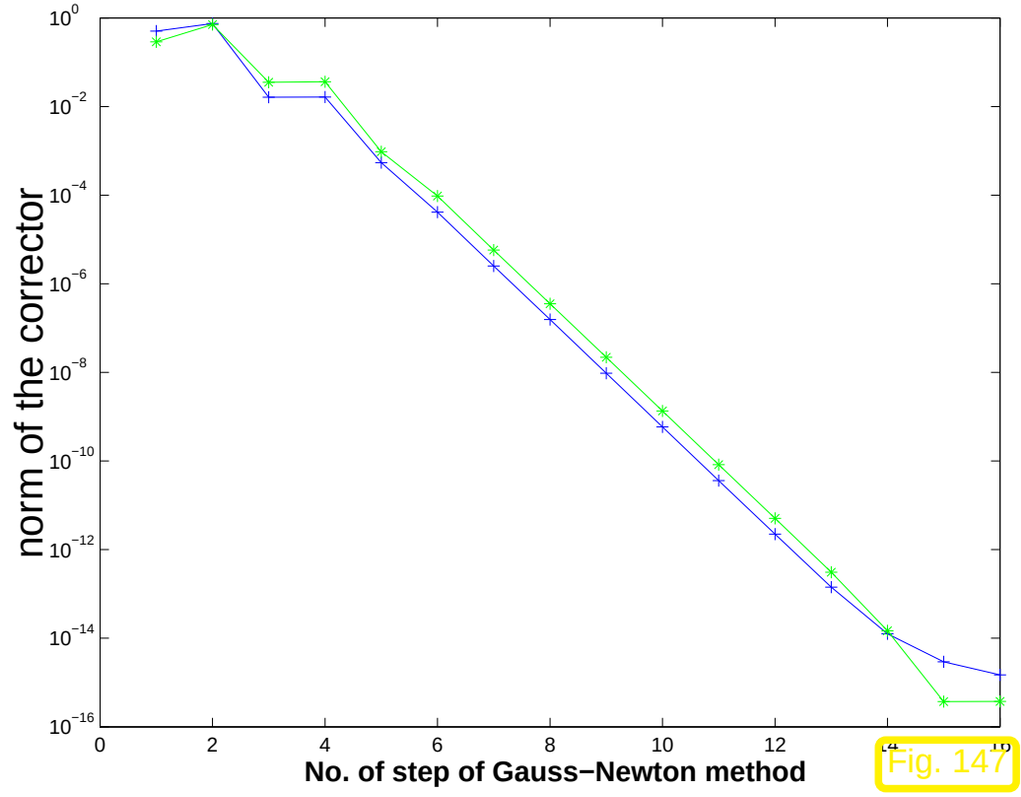


Fig. 147

◇

7.5.3 Trust region method (Levenberg-Marquardt method)

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 $\left\|F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})\mathbf{s}\right\|^2 + \lambda \left\|\mathbf{s}\right\|_2^2$.

$\lambda > 0 \hat{=}$ penalty parameter (how to choose it **?** $\rightarrow$ heuristic)

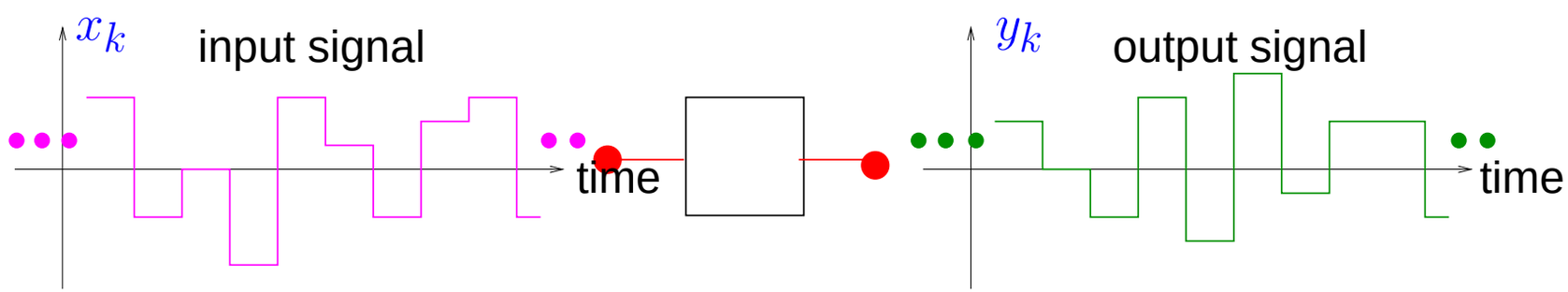
R. Hiptmair

rev 38355, November 17, 2011

8 Filtering Algorithms

8.1 Discrete convolutions

Example 8.1.1 (Discrete finite linear time-invariant causal channel (filter)).



Impulse response of channel (filter): $\mathbf{h} = (h_0, \dots, h_{n-1})^\top$

R. Hiptmair

rev 38355, November 17, 2011

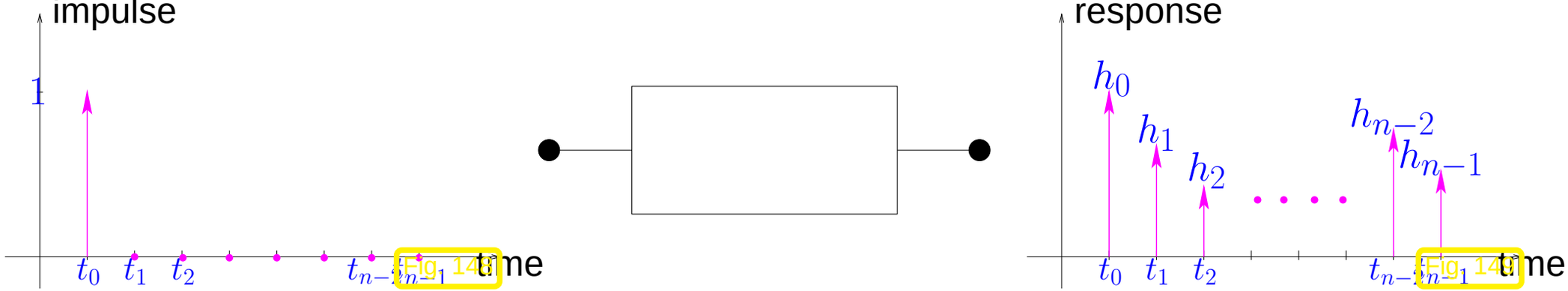


Fig. 148

Fig. 149

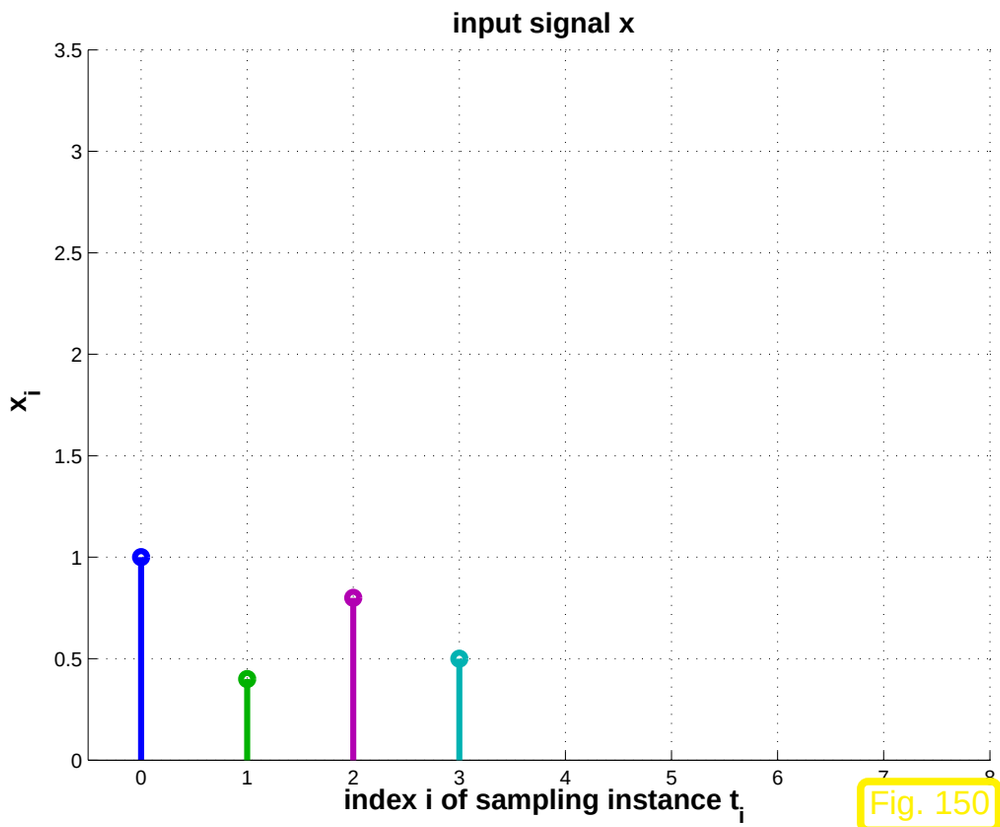


Fig. 150

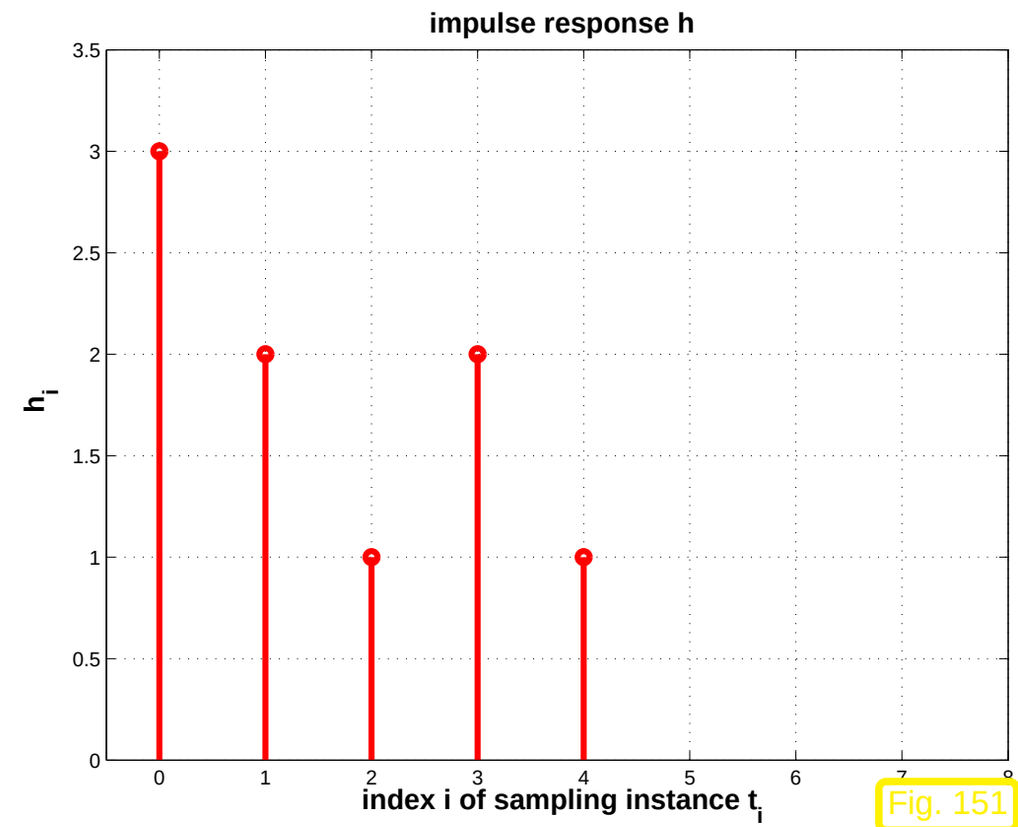


Fig. 151

Output = linear superposition of impulse responses:

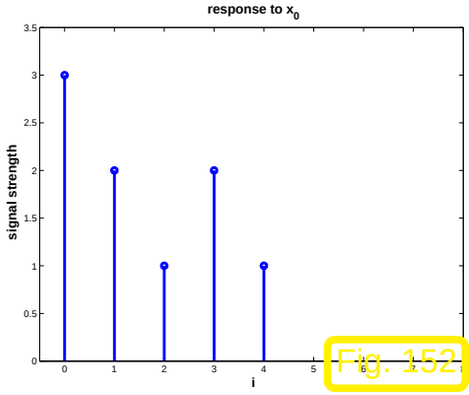


Fig. 152

+

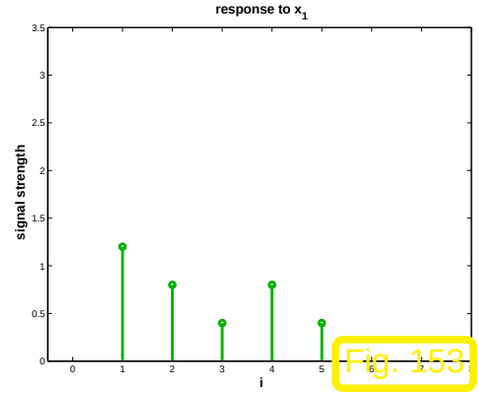


Fig. 153

+

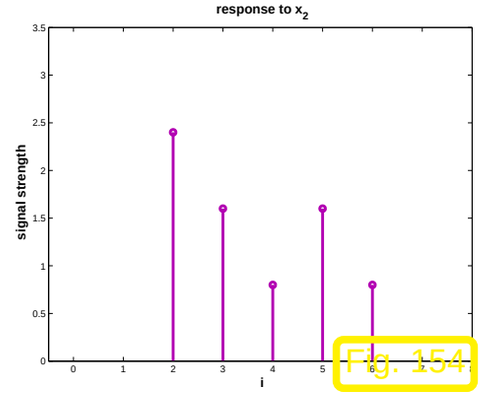


Fig. 154

+

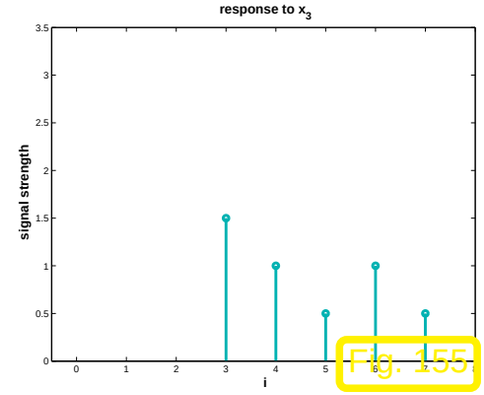


Fig. 155

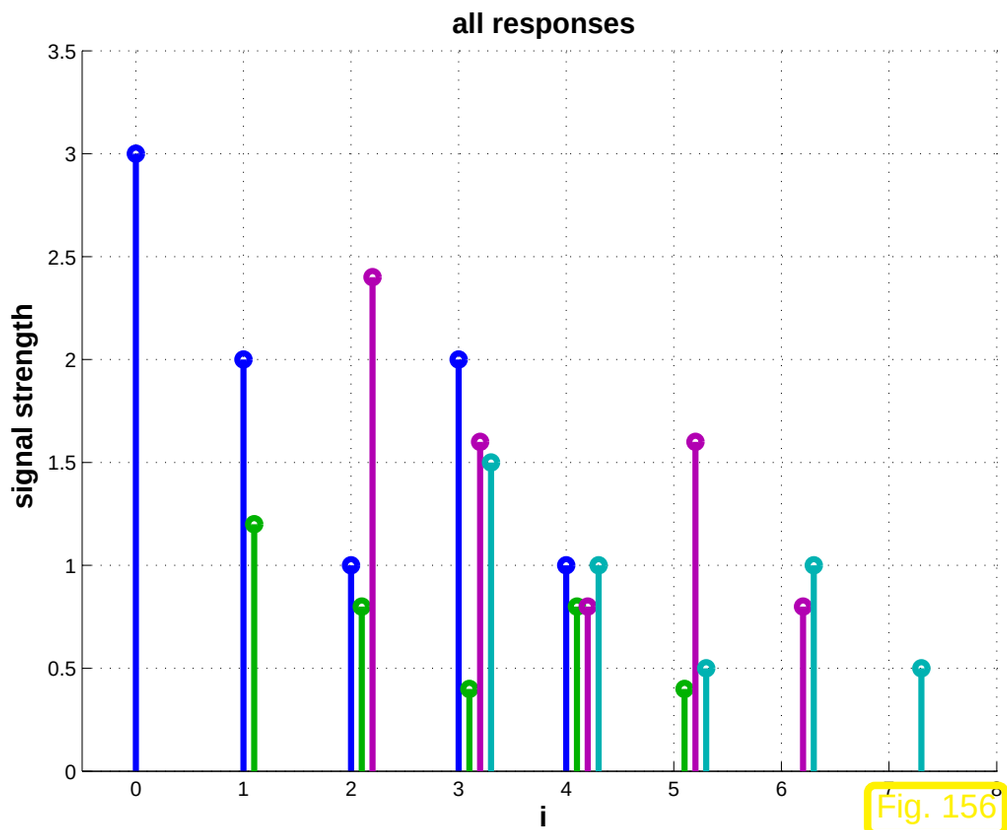


Fig. 156

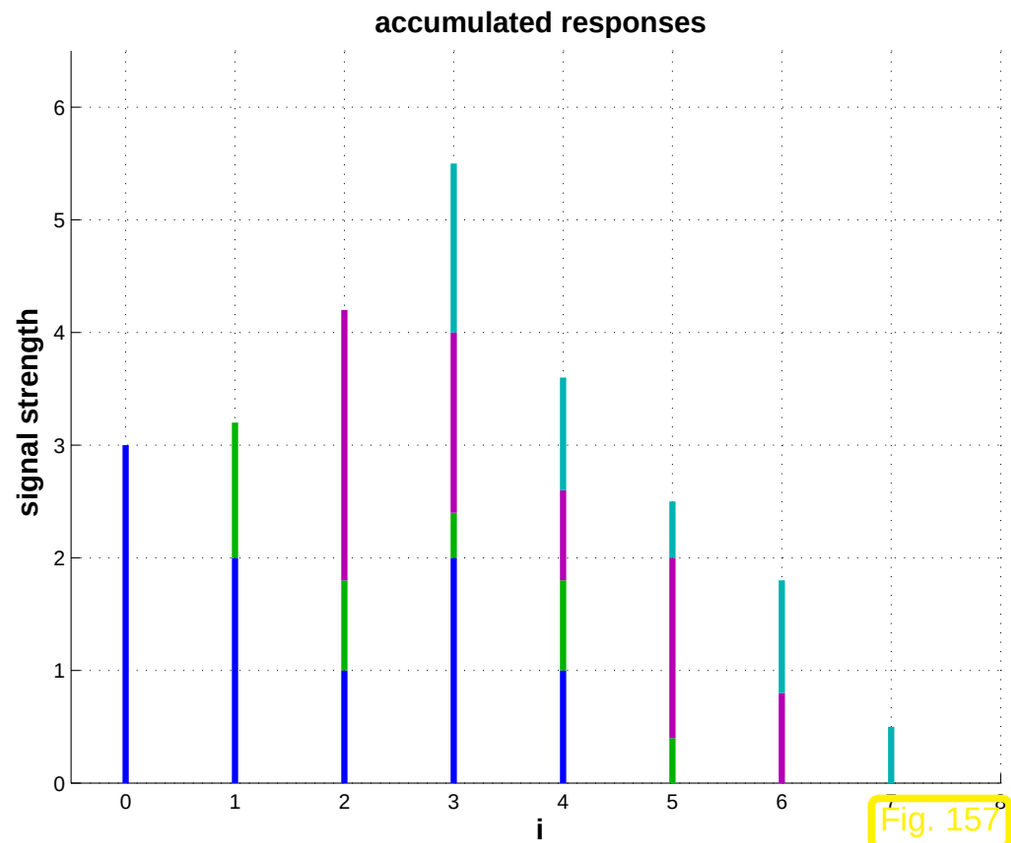


Fig. 157

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \\ \vdots \\ \vdots \\ y_{2n-3} \\ y_{2n-2} \end{pmatrix} = x_0 \begin{pmatrix} h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} + x_1 \begin{pmatrix} 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \cdots + x_{n-1} \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \end{pmatrix} .$$

channel is causal!

$$\boxed{y_k = \sum_{j=0}^{n-1} h_{k-j} x_j} \quad , \quad k = 0, \dots, 2n-2 \quad (h_j := 0 \text{ for } j < 0 \text{ and } j \geq n) . \tag{8.1.3}$$

$\mathbf{x} = (x_0, \dots, x_{n-1})^\top \in \mathbb{R}^n \hat{=}$ input signal $\mapsto$ $\mathbf{y} = (y_0, \dots, y_{2n-2})^\top \in \mathbb{R}^{2n-1} \hat{=}$ output signal.

R. Hiptmair

rev 38355, November 17, 2011

Matrix notation of (8.1.3):

$$\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \\ \vdots \\ y_{2n-2} \end{pmatrix} = \begin{pmatrix} h_0 & 0 & \cdots & 0 \\ h_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ h_{n-1} & \cdots & h_1 & h_0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \\ 0 & \cdots & 0 & h_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ \\ \\ \\ \vdots \\ x_{n-1} \end{pmatrix} . \tag{8.1.4}$$

◇

Example 8.1.5 (Multiplication of polynomials).

◇

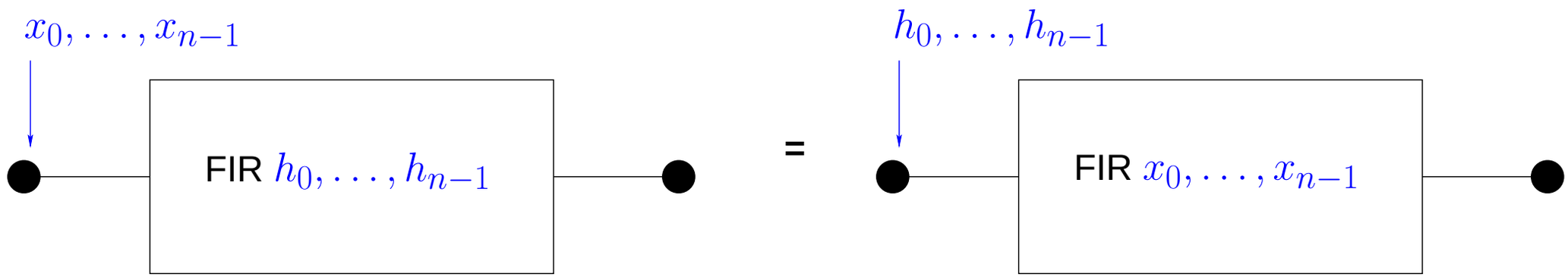
R. Hiptmair

rev 38355, November 17, 2011

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*

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j \ , \quad k = 0, \dots, 2n-2 \quad (h_j := 0 \text{ for } j < 0) \ . \tag{8.1.8}$$

✎ Notation for discrete convolution (8.1.8): $\mathbf{y} = \mathbf{h} * \mathbf{x}$.



Example 8.1.10 (Linear filtering of periodic signals)*.*

n-periodic signal ($n \in \mathbb{N}$) = sequence $(x_j)_{j\in\mathbb{Z}}$ with $\boxed{x_{j+n} = x_j \quad \forall j \in \mathbb{Z}}$

➢ n-periodic signal $(x_j)_{j\in\mathbb{Z}}$ fixed by $x_0, \dots, x_{n-1} \leftrightarrow$ vector $\mathbf{x} = (x_0, \dots, x_{n-1})^\top \in \mathbb{R}^n$.

$$y_k = \sum_{j=0}^{n-1} p_{k-j} x_j \quad \text{for some} \quad p_0, \dots, p_{n-1} \in \mathbb{R}\,, \quad p_k := p_{k-n} \text{ for all } k \in \mathbb{Z}\,. \tag{8.1.11}$$

Matrix notation:

$$\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \vdots \\ y_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} p_0 & p_{n-1} & p_{n-2} & \cdots & & & \cdots & p_1 \\ p_1 & p_0 & p_{n-1} & & & & & \vdots \\ p_2 & p_1 & p_0 & \ddots & & & & \\ \vdots & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & & \ddots & \\ \vdots & & & & & \ddots & \ddots & p_{n-1} \\ p_{n-1} & \cdots & & & & & p_1 & p_0 \end{pmatrix}}_{=:\mathbf{P}} \begin{pmatrix} x_0 \\ \vdots \\ \\ \\ \vdots \\ x_{n-1} \end{pmatrix}\,. \tag{8.1.12}$$

$\diamond$

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

$$(z_k) := (x_k) *_n (y_k) \quad , \quad z_k := \sum_{j=0}^{n-1} x_{k-j} y_j = \sum_{j=0}^{n-1} y_{k-j} x_j \ , \quad k \in \mathbb{Z} \ .$$

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)*

$$:\Leftrightarrow \qquad \exists\, (u_k)_{k\in\mathbb{Z}} \ n\text{-periodic sequence:} \quad c_{ij} = u_{j-i},\ 1 \le i,j \le n.$$

R. Hiptmair

rev 38355, November 17, 2011

Structure of circulant matrix $\triangleright$

$$\begin{pmatrix} u_0 & u_1 & u_2 & \cdots & & \cdots & u_{n-1} \\ u_{n-1} & u_0 & & & & & u_{n-2} \\ u_{n-2} & & & & & & \vdots \\ \vdots & & & & & & \\ \vdots & & & & & & \vdots \\ u_2 & & & & & & u_1 \\ u_1 & u_2 & \cdots & & \cdots & u_{n-1} & u_0 \end{pmatrix}$$

Remark 8.1.19 (Reduction to periodic convolution).

Recall discrete convolution ($\rightarrow$ Def. 8.1.7) of $\mathbf{a} = (a_0, \dots, a_{n-1})^\top \in \mathbb{K}^n$, $\mathbf{b} = (b_0, \dots, b_{n-1})^\top \in \mathbb{K}^n$:

$$(\mathbf{a} * \mathbf{b})_k = \sum_{j=0}^{n-1} a_j b_{k-j} \ , \quad k = 0, \dots, 2n-2 \ .$$

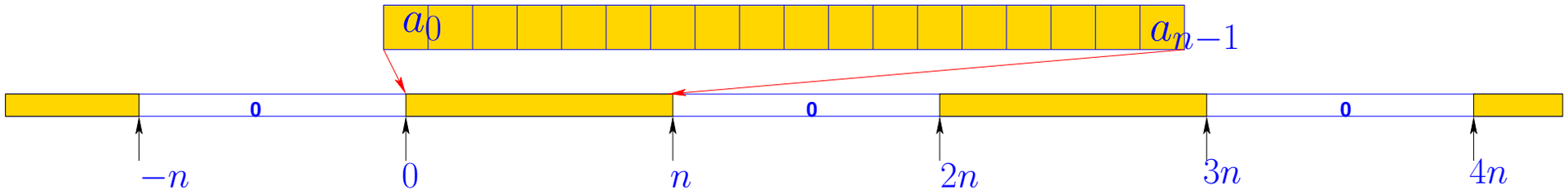
Expand $a_0, \dots, a_{n-1}$ and $b_0, \dots, b_{n-1}$ to $2n-1$-periodic sequences by zero padding:

$$x_k := \begin{cases} a_k & \text{, if } 0 \le k < n \ , \\ 0 & \text{, if } n \le k < 2n-1 \end{cases} \quad , \quad y_k := \begin{cases} b_k & \text{, if } 0 \le k < n \ , \\ 0 & \text{, if } n \le k < 2n-1 \ , \end{cases} \tag{8.1.20}$$

and periodic extension: $x_k = x_{2n-1+k}$, $y_k = y_{2n-1+k}$ for all $k \in \mathbb{Z}$.

R. Hiptmair

rev 38355, November 17, 2011



$$(\mathbf{a} * \mathbf{b})_k = (\mathbf{x} *_{2n-1} \mathbf{y})_k \ , \quad k = 0, \dots, 2n-2 \ . \tag{8.1.21}$$

Matrix view of reduction to periodic convolution, *cf.* (8.1.4)

$$
\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \\ \\ \\ \vdots \\ y_{2n-2} \end{pmatrix}
=
\underbrace{\begin{pmatrix}
h_0 & 0 & \cdots & 0 & h_{n-1} & \cdots & h_1 & h_0 \\
h_1 & \ddots & \ddots & \vdots & 0 & \ddots & \vdots & h_1 \\
\vdots & \ddots & \ddots & 0 & \vdots & \ddots & \ddots & \vdots \\
h_{n-1} & \cdots & h_1 & h_0 & 0 & \cdots & 0 & h_{n-1} \\
0 & \ddots & \ddots & h_1 & h_0 & \ddots & \vdots & 0 \\
\vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \ddots & 0 \\
0 & \cdots & 0 & h_{n-1} & \cdots & h_1 & h_0 & h_0
\end{pmatrix}}_{\text{a } (2n-1)\times(2n-1) \text{ circulant matrix!}}
\begin{pmatrix} x_0 \\ \vdots \\ \\ \\ \\ \\ \\ \vdots \\ x_{n-1} \\ 0 \\ \vdots \\ \\ \\ \\ \\ \vdots \\ 0 \end{pmatrix}
. \qquad (8.1.22)
$$

$\triangle$

R. Hiptmair

rev 38355, November 17, 2011

8.2 Discrete Fourier Transform (DFT)

Example 8.2.1 (Eigenvectors of circulant matrices).

Random 8×8 circulant matrices $\mathbf{C}_1$, $\mathbf{C}_2$ ($\rightarrow$ Def. 8.1.17)

eigenvalues (real part) $\triangleright$

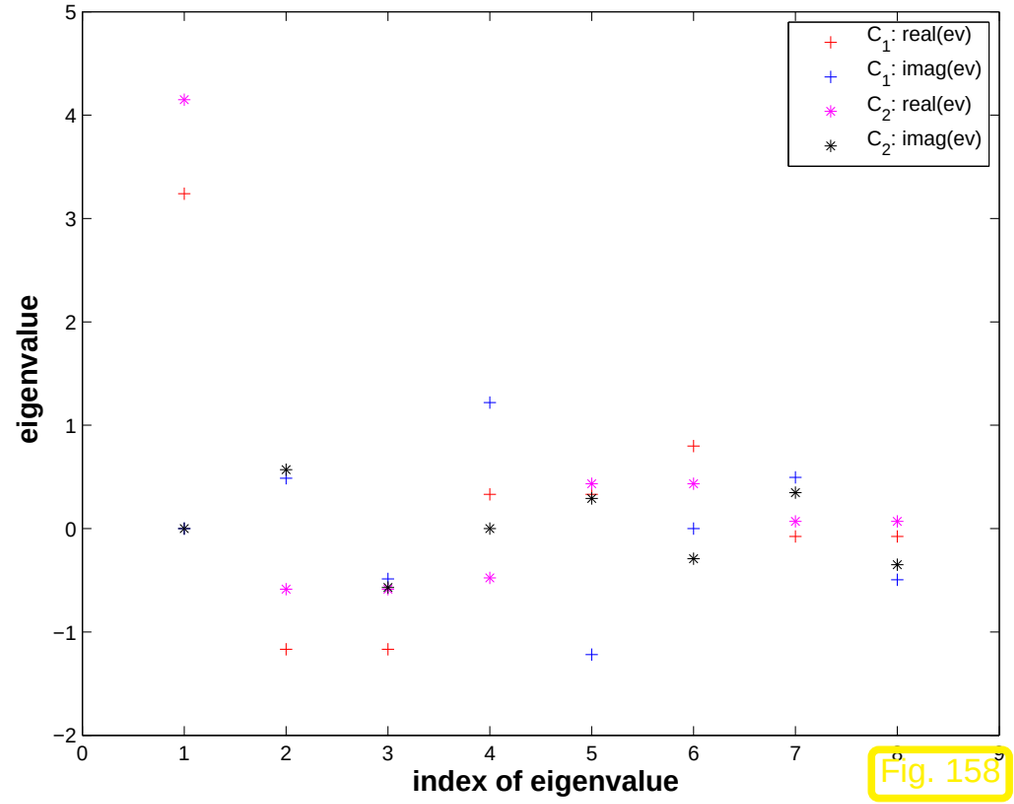
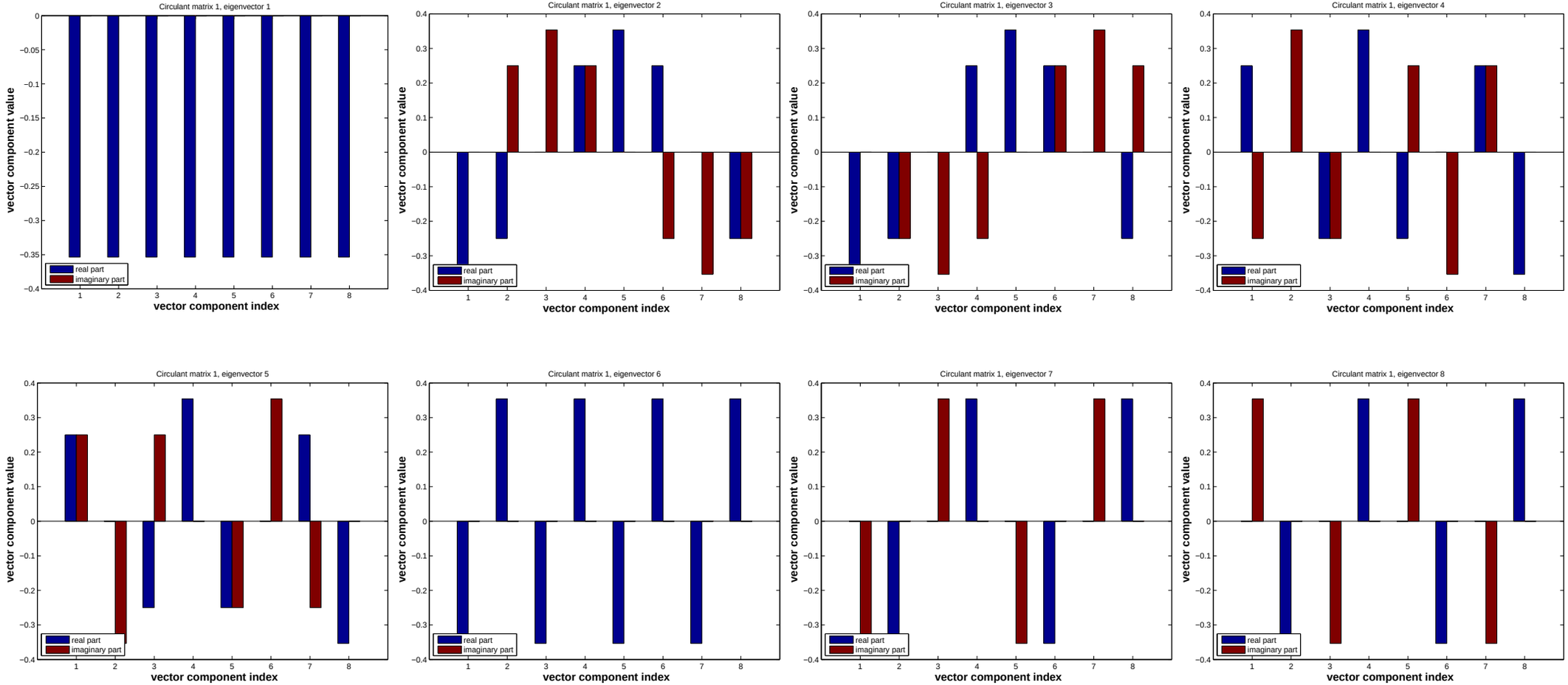


Fig. 158

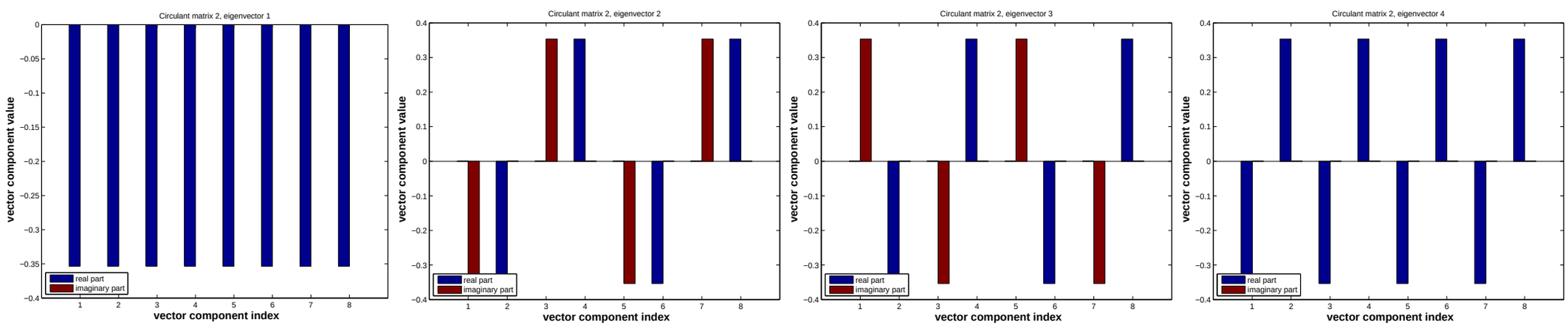
Generated by MATLAB-command:

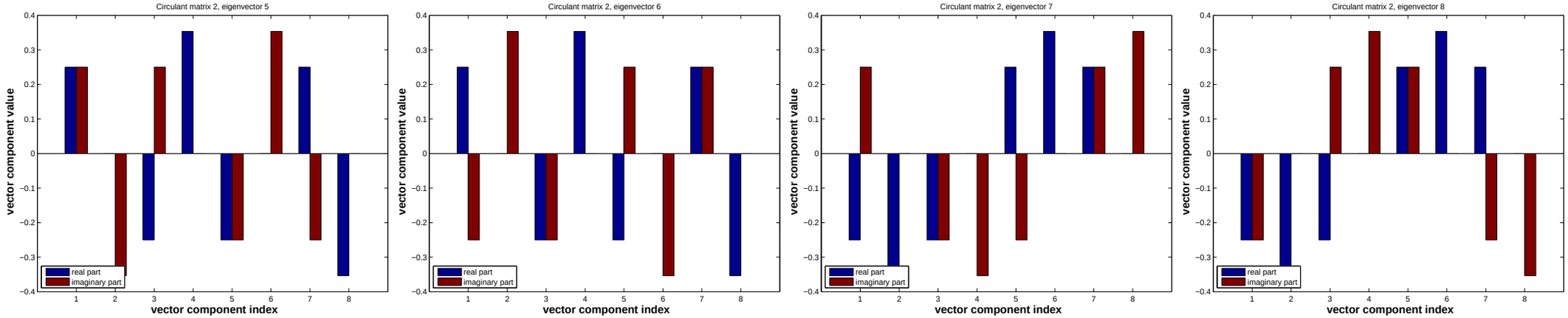
```
C = gallery('circul',rand(n,1));
```

Eigenvectors of matrix $\mathbf{C}_1$:

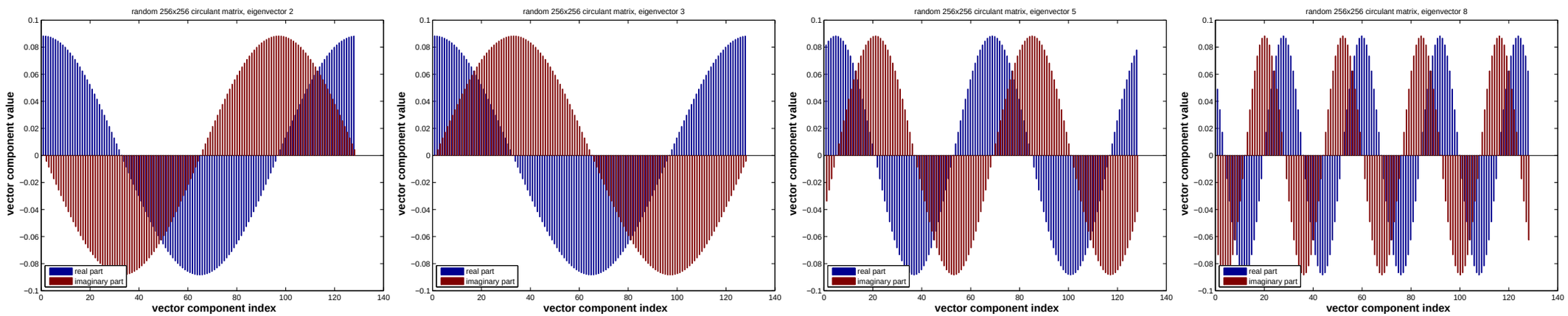


Eigenvectors of matrix $\mathbf{C}_2$





Eigenvectors of C = gallery('circul',(1:128)'); :



◇

✎ notation: nth root of unity $\omega_n := \exp(-2\pi i/n) = \cos(2\pi/n) - i\sin(2\pi/n), \quad n \in \mathbb{N}$

satisfies $\overline{\omega}_n = \omega_n^{-1}$, $\boxed{\omega_n^n = 1}$, $\omega_n^{n/2} = -1$, $\omega_n^k = \omega_n^{k+n} \quad \forall\, k \in \mathbb{Z}$, **(8.2.3)**

$$\sum_{k=0}^{n-1} \omega_n^{kj} = \begin{cases} n & \text{, if } j = 0 \mod n \,, \\ 0 & \text{, if } j \neq 0 \mod n \,. \end{cases} \tag{8.2.4}$$

Orthogonal trigonometric basis of $\mathbb{C}^n$ = *eigenvector basis* for circulant matrices

$$\left\{ \begin{pmatrix} \omega_n^0 \\ \vdots \\ \\ \vdots \\ \omega_n^0 \end{pmatrix} \begin{pmatrix} \omega_n^0 \\ \omega_n^1 \\ \vdots \\ \\ \vdots \\ \omega_n^{n-1} \end{pmatrix} \cdots \begin{pmatrix} \omega_n^0 \\ \omega_n^{n-2} \\ \omega_n^{2(n-2)} \\ \vdots \\ \vdots \\ \omega_n^{(n-1)(n-2)} \end{pmatrix} \begin{pmatrix} \omega_n^0 \\ \omega_n^{n-1} \\ \omega_n^{2(n-1)} \\ \vdots \\ \vdots \\ \omega_n^{(n-1)^2} \end{pmatrix} \right\} .$$

Matrix of change of basis trigonometrical basis $\rightarrow$ standard basis: Fourier-matrix

$$\mathbf{F}_n = \begin{pmatrix} \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \cdots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \cdots & \omega_n^{2n-2} \\ \vdots & \vdots & & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{pmatrix} = \left(\omega_n^{lj}\right)_{l,j=0}^{n-1} \in \mathbb{C}^{n,n} \,. \tag{8.2.8}$$

R. Hiptmair

rev 38355, November 17, 2011

Lemma 8.2.9 (Properties of Fourier matrix)**.**

The scaled Fourier-matrix $\frac{1}{\sqrt{n}}\mathbf{F}_n$ *is unitary (→ Def. 2.8.5):* $\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^{\mathsf{H}} = \frac{1}{n}\overline{\mathbf{F}}_n$.

Lemma 8.2.10 (Diagonalization of circulant matrices (→ 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*

$$\mathbf{C}\overline{\mathbf{F}}_n = \overline{\mathbf{F}}_n \operatorname{diag}(d_1,\dots,d_n) \quad , \quad \mathbf{d} = \mathbf{F}_n(u_0,\dots,u_{n-1})^\top \ .$$

R. Hiptmair

rev 38355, November 17, 2011

Conclusion (from $\overline{\mathbf{F}}_n = n\mathbf{F}_n^{-1}$):

$$\boxed{\mathbf{C} = \mathbf{F}_n^{-1}\operatorname{diag}(d_1,\dots,d_n)\mathbf{F}_n} \ . \tag{8.2.11}$$

Definition 8.2.12 (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* $\mathbf{c} := \mathcal{F}_n(\mathbf{y})$

$$c_k = \sum_{j=0}^{n-1} y_j \, \omega_n^{kj} \quad , \quad k = 0, \dots, n-1 \; . \tag{8.2.13}$$

Terminology: $\mathbf{c} = \mathbf{F}_n\mathbf{y}$ is also called the (discrete) Fourier transform of $\mathbf{y}$

MATLAB-functions for discrete Fourier transform (and its inverse):

DFT: `c=fft(y)` $\leftrightarrow$ inverse DFT: `y=ifft(c);`

8.2.1 Discrete convolution via DFT

Code 8.2.16: discrete periodic convolution: straightforward implementation

```
function z=pconv(u,x)
n = length(x); z = zeros(n,1);
for i=1:n, z(i)=dot(conj(u),
  x([i:-1:1,n:-1:i+1]));
end
```

Code 8.2.18: discrete periodic convolution: DFT implementation

```
function z=pconvfft(u,x)
% Implementation of (8.2.13), cf.
  Lemma 8.2.12
z = ifft(fft(u).*fft(x));
```


Implementation of discrete convolution ($\rightarrow$ Def. 8.1.7) based on periodic discrete convolution $\triangleright$

Built-in MATLAB-function:

```
y = conv(h,x);
```

Code 8.2.19: discrete convolutiuon: DFT implementation

```
function y = myconv(h,x)
n = length(h);
% Zero padding, cf. (8.1.20)
h = [h;zeros(n-1,1)];
x = [x;zeros(n-1,1)];
% Periodic discrete convolution of length 2n-1
y = pconvfft(h,x);
```

8.2.2 Frequency filtering via DFT

Example 8.2.21 (Detecting periodicity in data).

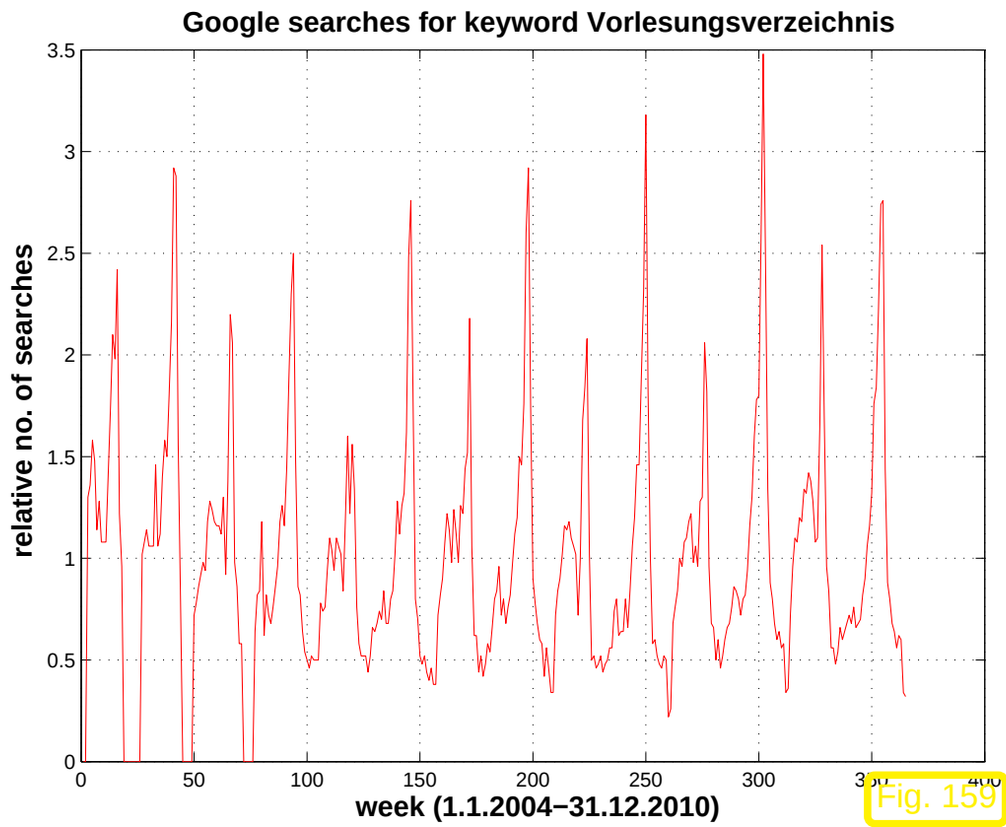


Fig. 159

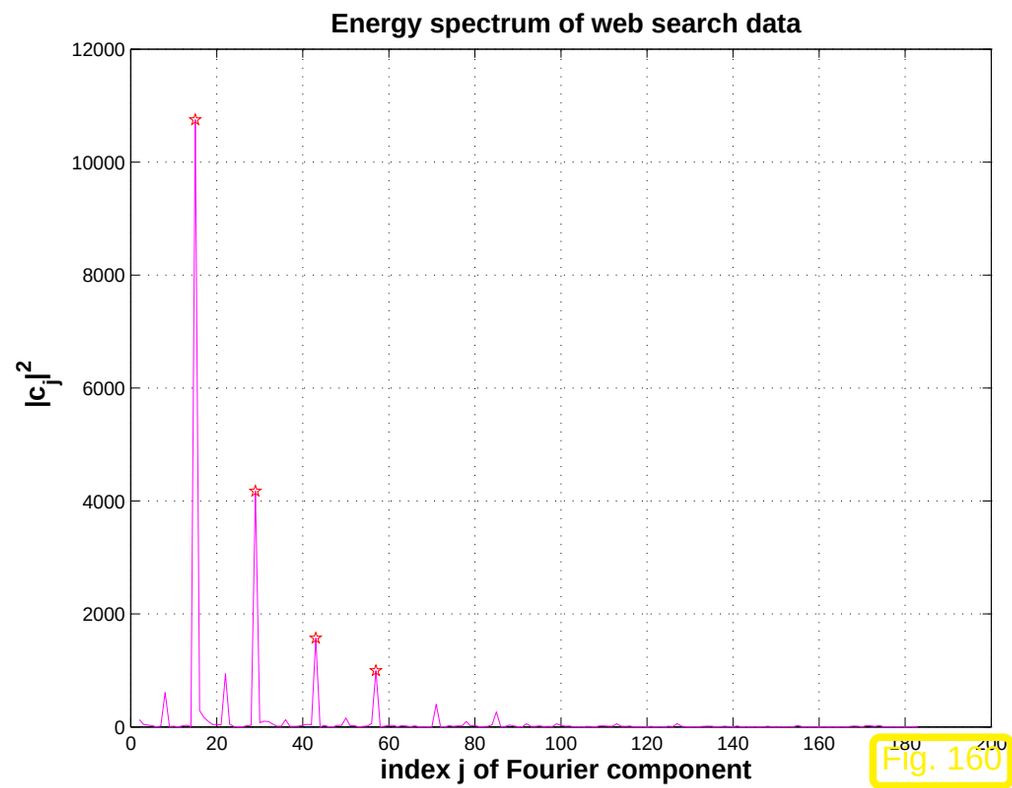
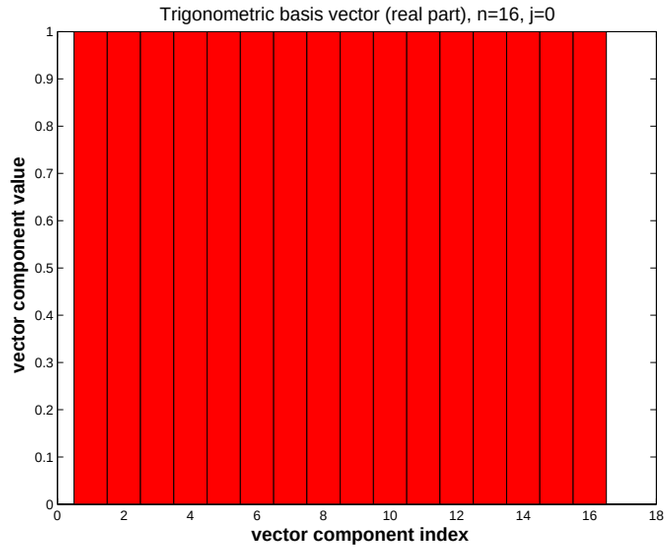


Fig. 160

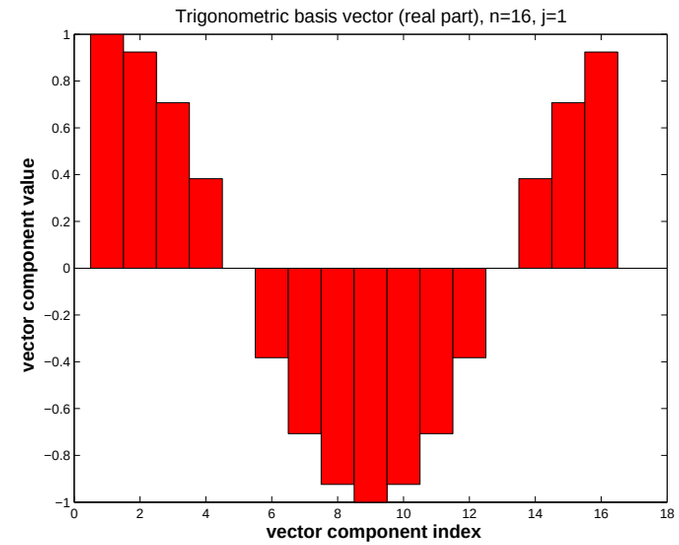
◇

Remark 8.2.23 ("Low" and "high" frequencies).

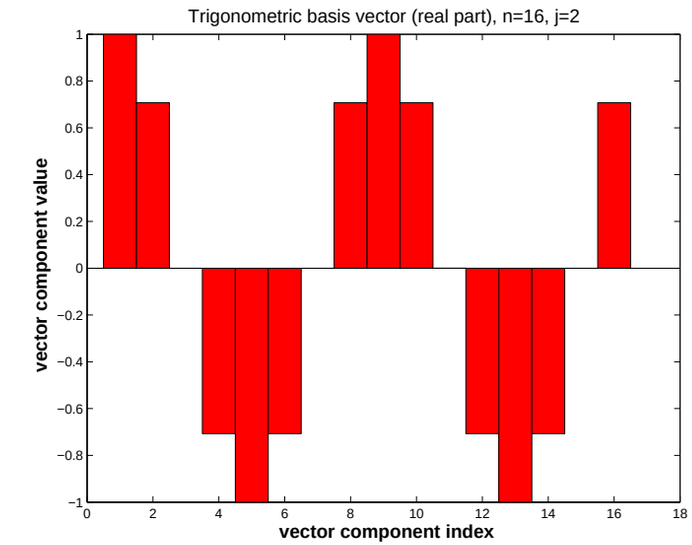
Plots of real parts of trigonometric basis vectors $(\mathbf{F}_n)_{:,j}$ (= columns of Fourier matrix), $n = 16$.



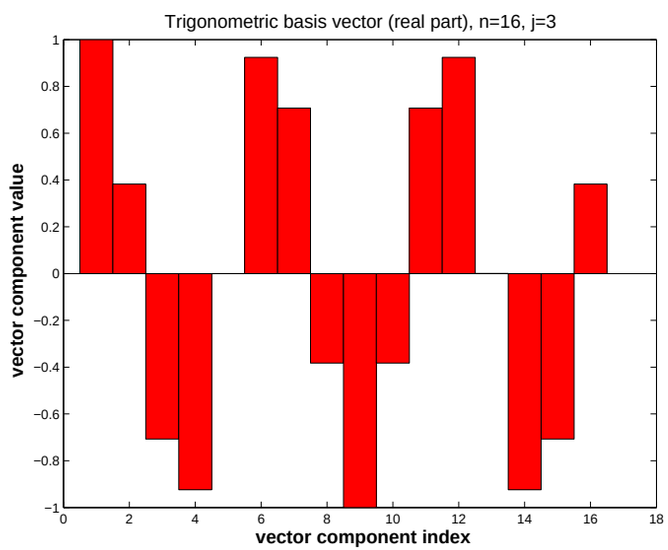
$\mathrm{Re}\left(\mathbf{F}_{16}\right)_{:,0}$



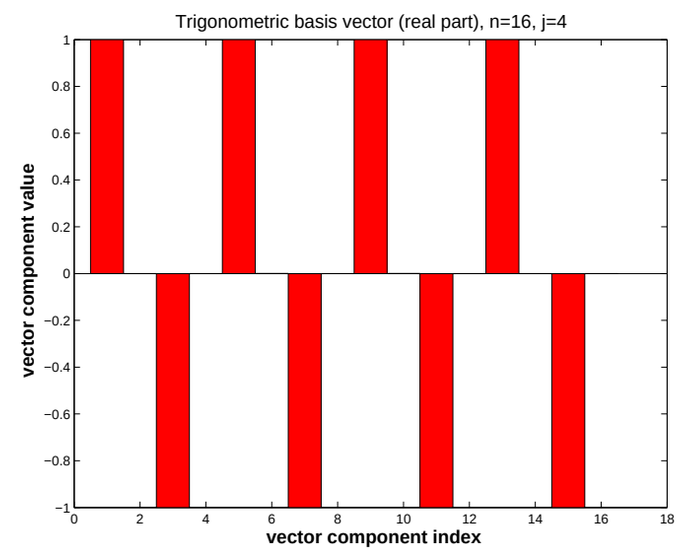
$\mathrm{Re}\left(\mathbf{F}_{16}\right)_{:,1}$



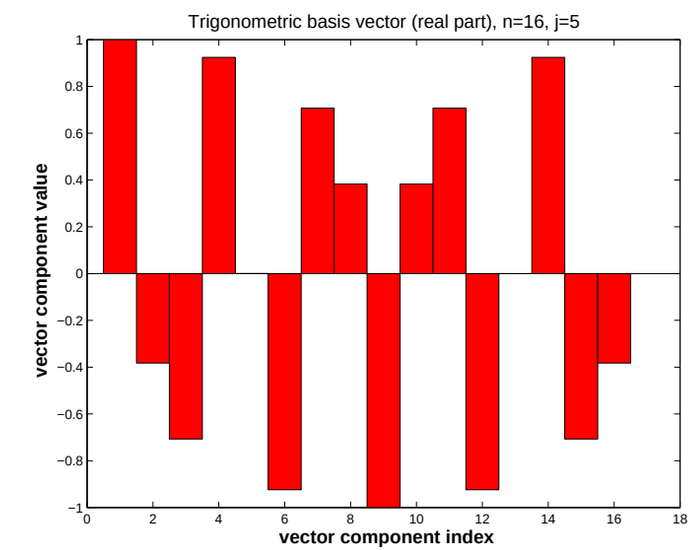
$\mathrm{Re}\left(\mathbf{F}_{16}\right)_{:,2}$



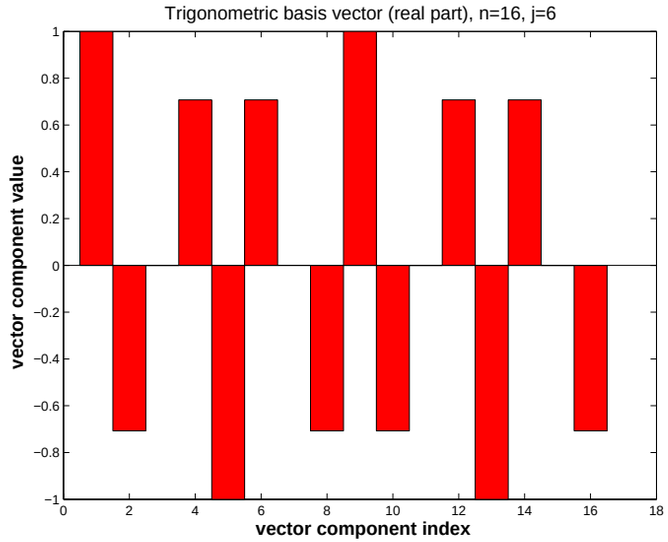
$\mathrm{Re}\left(\mathbf{F}_{16}\right)_{:,3}$



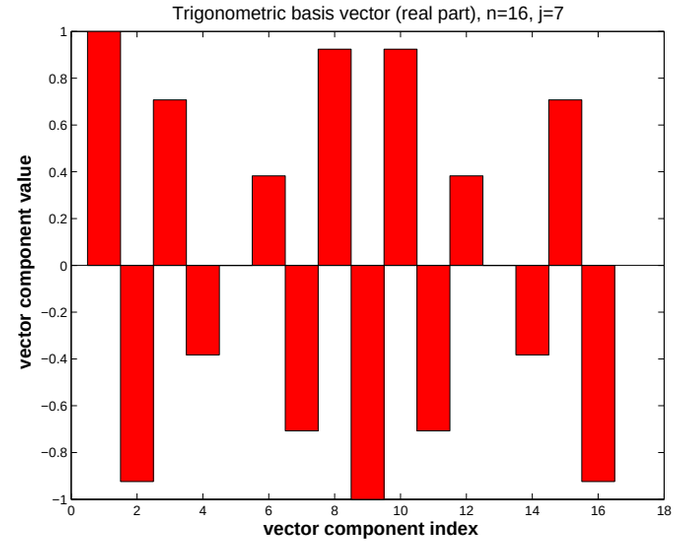
$\mathrm{Re}\left(\mathbf{F}_{16}\right)_{:,4}$



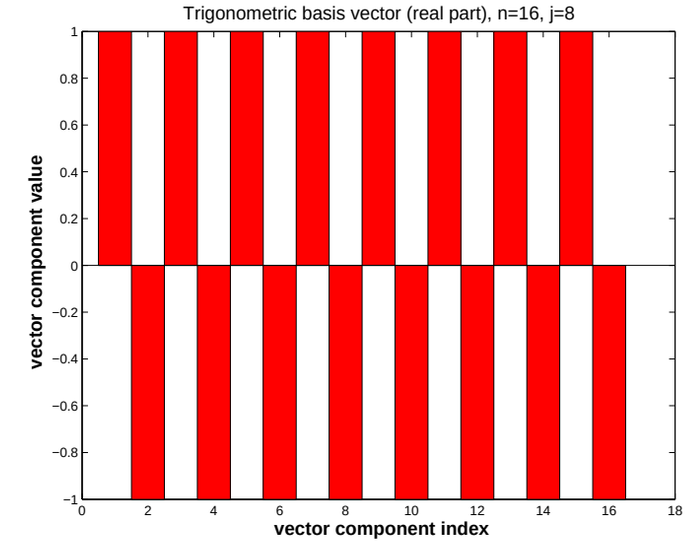
$\mathrm{Re}\left(\mathbf{F}_{16}\right)_{:,5}$



$\mathrm{Re}\left(\mathbf{F}_{16}\right)_{:,6}$



$\mathrm{Re}\left(\mathbf{F}_{16}\right)_{:,7}$



$\mathrm{Re}\left(\mathbf{F}_{16}\right)_{:,8}$

△

R. Hiptmair

rev 38355, November 17, 2011

- Frequency filtering of real discrete periodic signals by suppressing certain "Fourier coefficients".

Code 8.2.24: DFT-based frequency filtering

```
function [low,high] =
  freqfilter(y,k)
m = length(y)/2; c = fft(y);
clow = c; clow(m+1-k:m+1+k) = 0;
chigh = c-clow;
low = real(ifft(clow));
high = real(ifft(chigh));
```

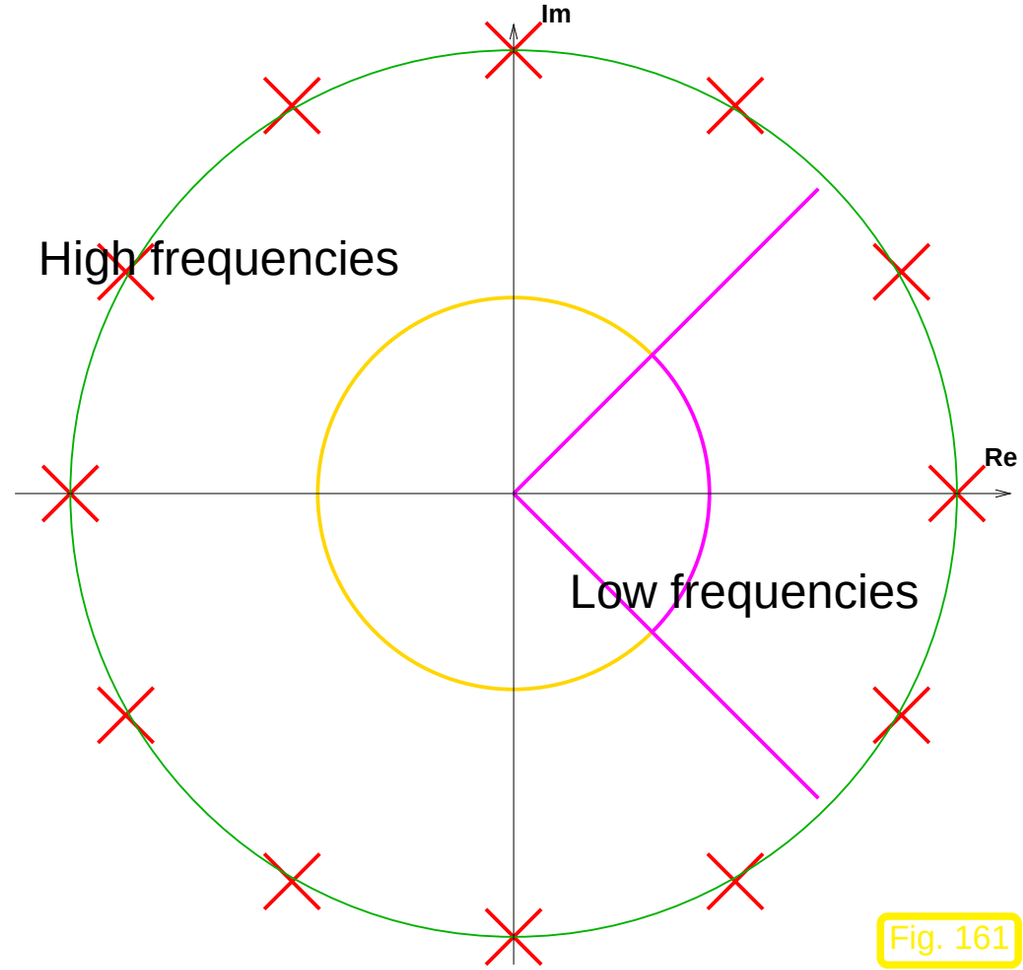
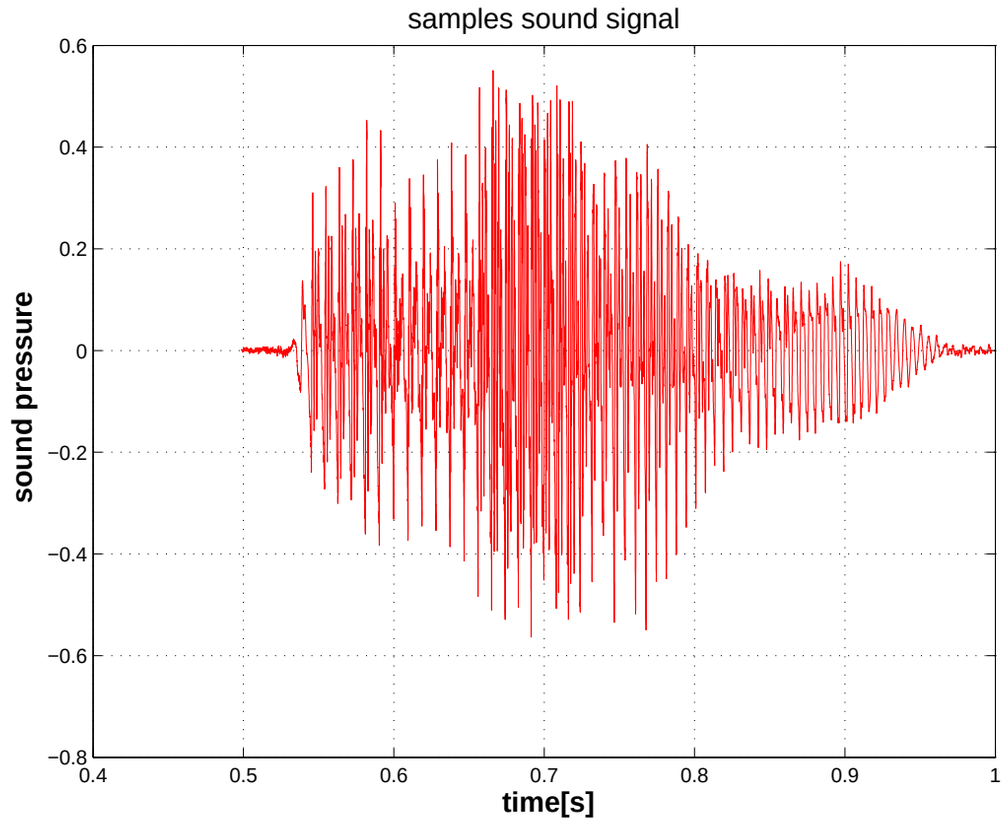


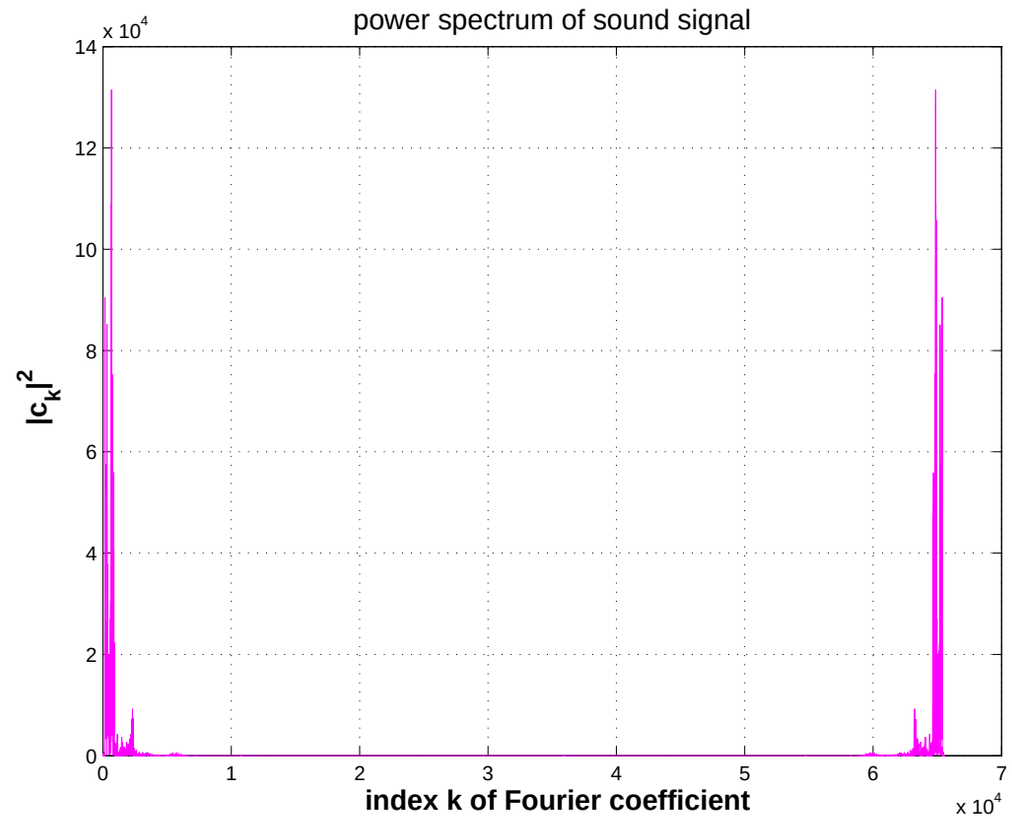
Fig. 161

Example 8.2.26 (Sound filtering by DFT).

DFT based low pass frequency filtering of sound

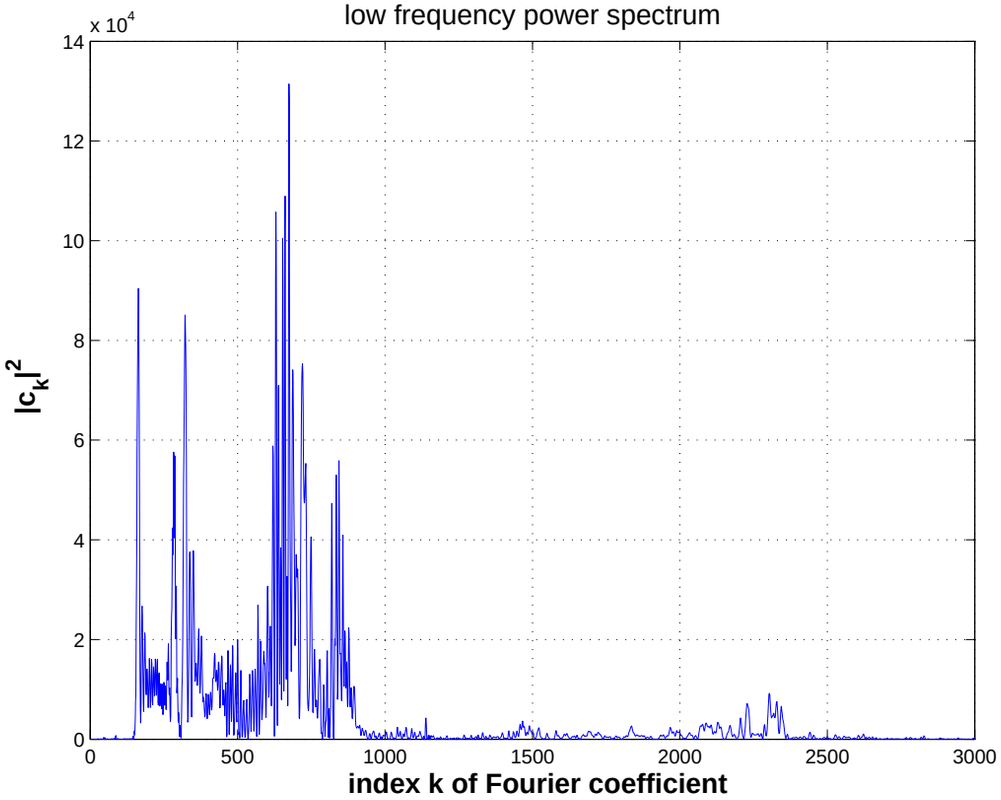
```
[y,sf,nb] = wavread('hello.wav');
c = fft(y); c(m+1:end-m) = 0;
wavwrite(ifft(c),sf,nb,'filtered.wav');
```

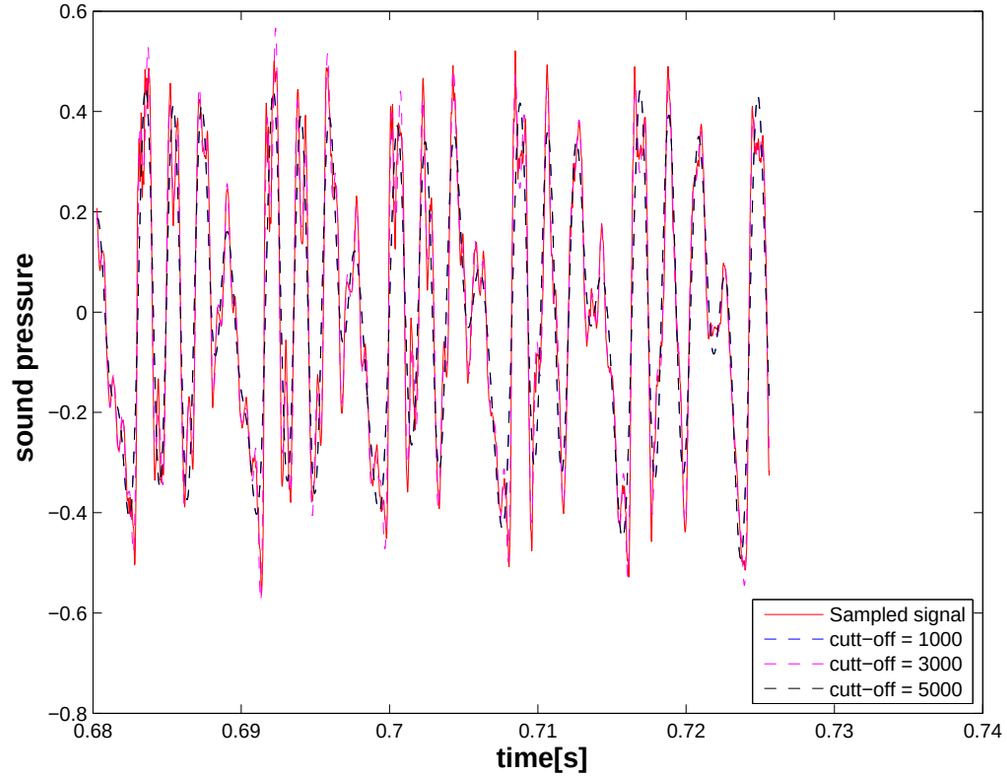




R. Hiptmair

rev 38355, November 17, 2011


low frequency power spectrum
x 10^4
|c_k|^2
index k of Fourier coefficient


sound pressure
time[s]
Sampled signal
cutt−off = 1000
cutt−off = 3000
cutt−off = 5000

◇

8.2.3 Real DFT

$$h_k = \sum_{j=0}^{m-1} (y_{2j} + iy_{2j+1})\,\omega_m^{jk} = \boxed{\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} + i \cdot \boxed{\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}}\,, \tag{8.2.28}$$

$$\overline{h}_{m-k} = \sum_{j=0}^{m-1} \overline{y_{2j} + iy_{2j+1}}\,\overline{\omega}_m^{j(m-k)} = \boxed{\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} - i \cdot \boxed{\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}}\,. \tag{8.2.29}$$

$$\Rightarrow \quad \boxed{\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} = \tfrac{1}{2}(h_k + \overline{h}_{m-k}) \quad , \quad \boxed{\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}} = -\tfrac{1}{2}i(h_k - \overline{h}_{m-k})\,.$$

Use simple identities for roots of unity:

$$c_k = \sum_{j=0}^{n-1} y_j\,\omega_n^{jk} = \boxed{\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} + \omega_n^k \cdot \boxed{\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}}\,. \tag{8.2.30}$$

$$\Rightarrow \quad \begin{cases} c_k = \frac{1}{2}(h_k + \overline{h}_{m-k}) - \frac{1}{2}i\omega_n^k(h_k - \overline{h}_{m-k})\,, \quad k = 0,\dots,m-1\,, \\ c_m = \mathrm{Re}\{h_0\} - \mathrm{Im}\{h_0\}\,, \\ c_k = \overline{c}_{n-k}\,, \quad k = m+1,\dots,n-1\,. \end{cases} \tag{8.2.31}$$

MATLAB-Implementation (by a DFT of length $n/2$):

Code 8.2.32: DFT of real vectors

```
function c = fftreal(y)
n = length(y); m = n/2;
if (mod(n,2) ~= 0), error('n must be even'); end
y = y(1:2:n)+i*y(2:2:n); h = fft(y); h = [h;h(1)];
c = 0.5*(h+conj(h(m+1:-1:1))) - ...
    (0.5*i*exp(-2*pi*i/n).^((0:m)')).*...
    (h-conj(h(m+1:-1:1)));
c = [c;conj(c(m:-1:2))];
```


8.2.4 Two-dimensional DFT

Two-dimensional trigonometric basis of $\mathbb{C}^{m,n}$:

tensor product matrices $$\left\{(\mathbf{F}_m)_{:,j}(\mathbf{F}_n)_{:,\ell}^\top,\ 1 \le j \le m,\ 1 \le \ell \le n\right\} . \tag{8.2.33}$$

Basis transform: for $y_{j_1,j_2} \in \mathbb{C}$, $0 \le j_1 < m$, $0 \le j_2 < n$ compute (nested DFTs !)

$$c_{k_1,k_2} = \sum_{j_1=0}^{m-1}\sum_{j_2=0}^{n-1} y_{j_1,j_2}\,\omega_m^{j_1k_1}\omega_n^{j_2k_2} \quad , \quad 0 \le k_1 < m\,,\, 0 \le k_2 < n\ .$$

MATLAB function: fft2(Y) .

Two-dimensional DFT by *nested one-dimensional DFTs* (8.2.15):

```
fft2(Y) = fft(fft(Y).').
```

Example 8.2.34 (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.

$\left(p_{l,k}\right)_{l,k\in\mathbb{Z}} \hat{=}$ periodically extended image:

$$p_{l,j} = (\mathbf{P})_{l+1,j+1} \quad \text{for} \quad 1 \le l \le m,\, 1 \le j \le n\,, \quad p_{l,j} = p_{l+m,j+n} \quad \forall l,k \in \mathbb{Z}\,.$$

Blurring = pixel values get replaced by weighted averages of near-by pixel values
(effect of distortion in optical transmission systems)

$$c_{l,j} = \sum_{k=-L}^{L} \sum_{q=-L}^{L} s_{k,q} p_{l+k,j+q}\,, \quad \begin{array}{l} 0 \le l < m\,, \\ 0 \le j < n\,, \end{array} \quad L \in \{1,\dots,\min\{m,n\}\}\,. \tag{8.2.35}$$

blurred image point spread function

Usually: L small, $s_{k,m} \ge 0$, $\sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q} = 1$ (an averaging)

Used in test calculations: $L = 5$

$$s_{k,q} = \frac{1}{1 + k^2 + q^2} \,.$$

Code 8.2.36: point spread function

```
function S = psf(L)
[X,Y] = meshgrid(-L:1:L,-L:1:L);
S = 1./(1+X.*X+Y.*Y);
S = S/sum(sum(S));
```

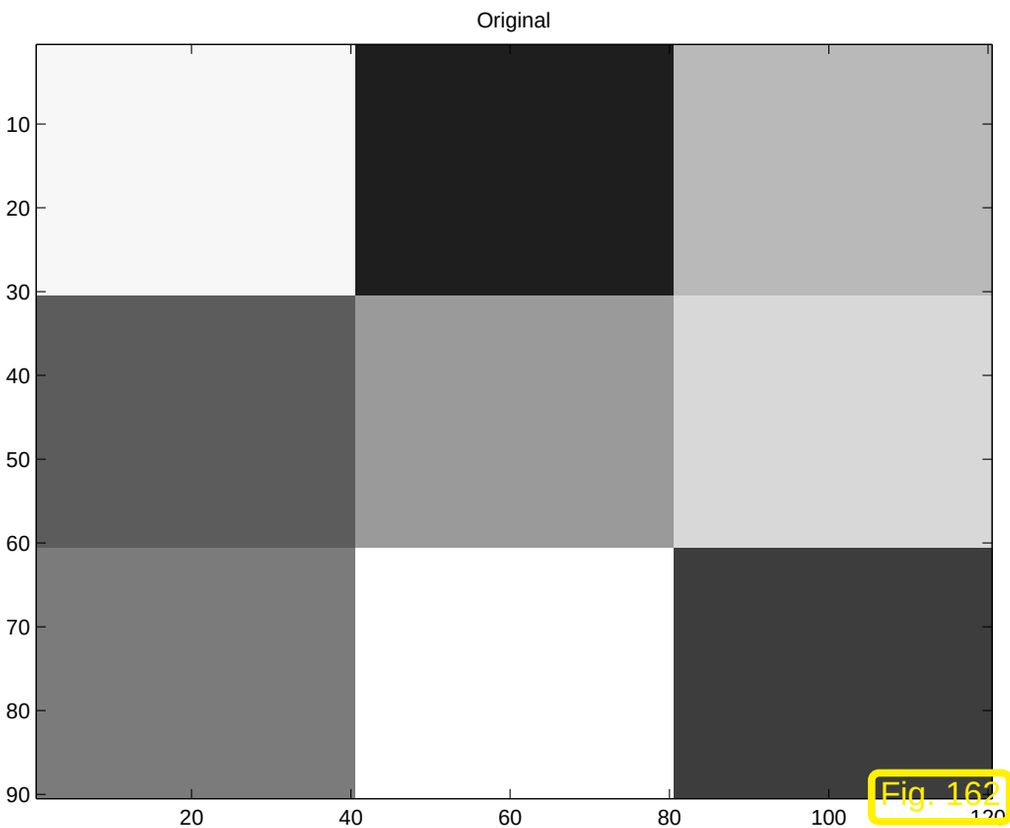


Fig. 162

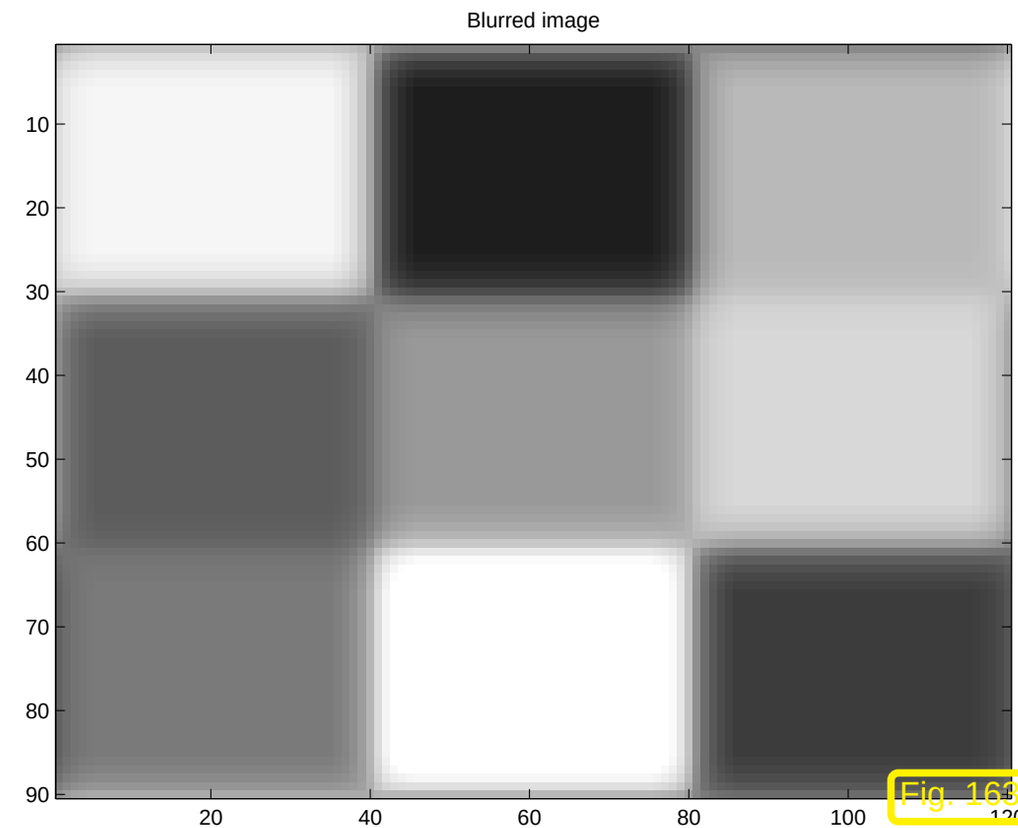


Fig. 163

R. Hiptmair

rev 38355, November 17, 2011

Note:

(8.2.37) defines a linear operator

$\mathcal{B} : \mathbb{R}^{m,n} \mapsto \mathbb{R}^{m,n}$

("blurring operator")

Note: more efficient implementation via MATLAB function conv2(P,S)

Code 8.2.38: blurring operator

```
function C = blur(P,S)
[m,n] = size(P); [M,N] = size(S);
if (M ~= N), error('S not quadratic'); end
L = (M-1)/2; C = zeros(m,n);
for l=1:m, for j=1:n
    s = 0;
    for k=1:(2*L+1), for q=1:(2*L+1)
     kl = l+k-L-1;
     if (kl < 1), kl = kl + m; end
     if (kl > m), kl = kl - m; end
     jm = j+q-L-1;
     if (jm < 1), jm = jm + n; end
     if (jm > n), jm = jm - n; end
     s = s+P(kl,jm)*S(k,q);
      end, end
    C(l,j) = s;
  end, end
```

Inversion of blurring operator

$\Updownarrow$

componentwise scaling in “Fourier domain”

Code 8.2.40: DFT based deblurring

```
function D = deblur(C,S,tol)
[m,n] = size(C); [M,N] = size(S);
if (M ~= N), error('S not quadratic'); end
L = (M-1)/2; Spad = zeros(m,n);
% Zero padding
Spad(1:L+1,1:L+1) = S(L+1:end,L+1:end);
Spad(m-L+1:m,n-L+1:n) = S(1:L,1:L);
Spad(1:L+1,n-L+1:n) = S(L+1:end,1:L);
Spad(m-L+1:m,1:L+1) = S(1:L,L+1:end);
% Inverse of blurring operator
SF = fft2(Spad);
% Test for invertibility
if (nargin < 3), tol = 1E-3; end
if (min(min(abs(SF))) < tol*max(max(abs(SF))))
  error('Deblurring impossible');
end
% DFT based deblurring
D = fft2(ifft2(C)./SF);
```

$\diamond$

8.2.5 Semi-discrete Fourier Transform [51, Sect. 10.11]

$(y_k)_{k\in\mathbb{Z}}$: n-periodic sequence (signal), $n = 2m+1$, $m \in \mathbb{N}$:

$$\text{DFT:} \qquad c_k = \sum_{j=-m}^{m} y_j \exp(-2\pi i \frac{kj}{n}) \;, \quad k = 0,\dots,n-1 \;. \tag{8.2.41}$$

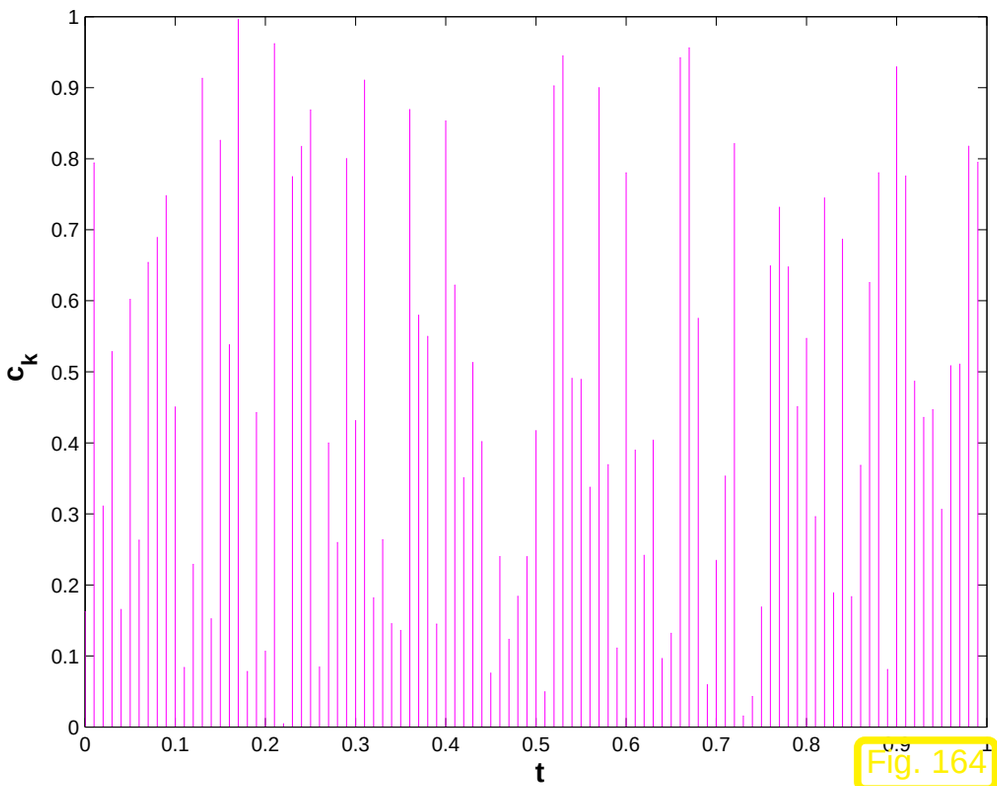


Fig. 164

◁ "Squeezing" a vector $\in \mathbb{R}^n$ into $[0,1[$.

$$c_k \quad \leftrightarrow \quad c(t_k);\,,$$

$$t_k = \frac{k}{n}\,, \quad k = 0,\dots,n-1 \;.$$

Example 8.2.44 ("Squeezed" DFT of a periodically truncated signal).

Bi-infinite discrete signal, “concentrated around 0”: $y_j = \dfrac{1}{1+j^2}, \quad j \in \mathbb{Z}.$

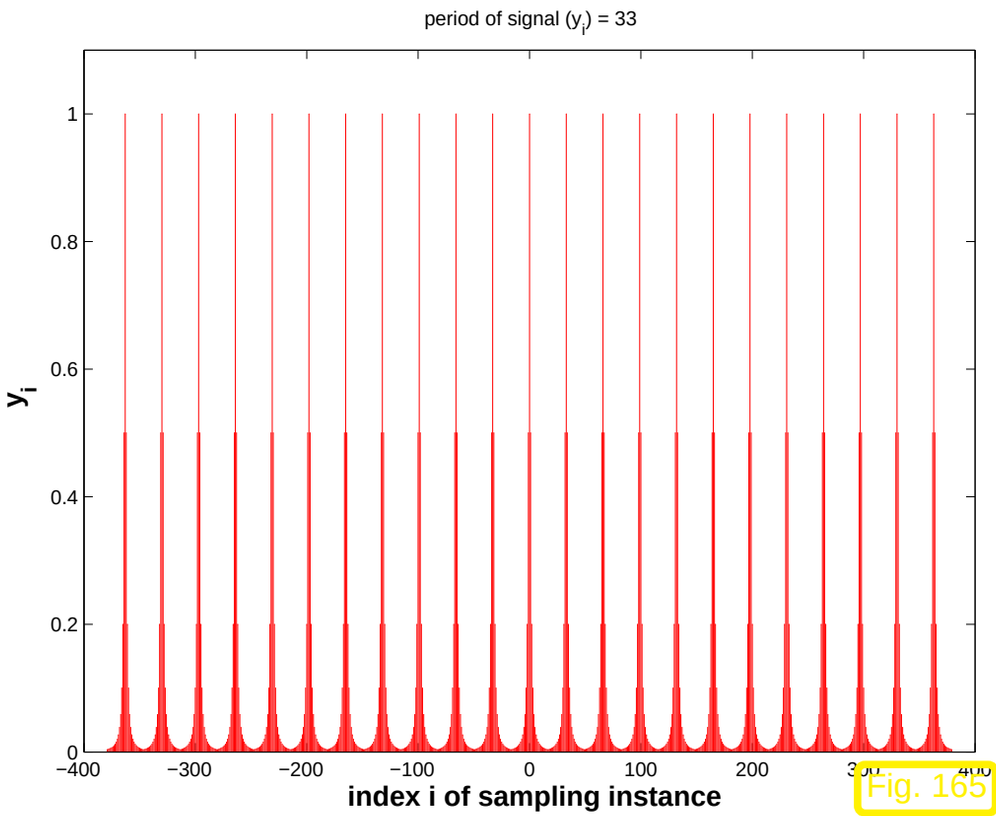


Fig. 165

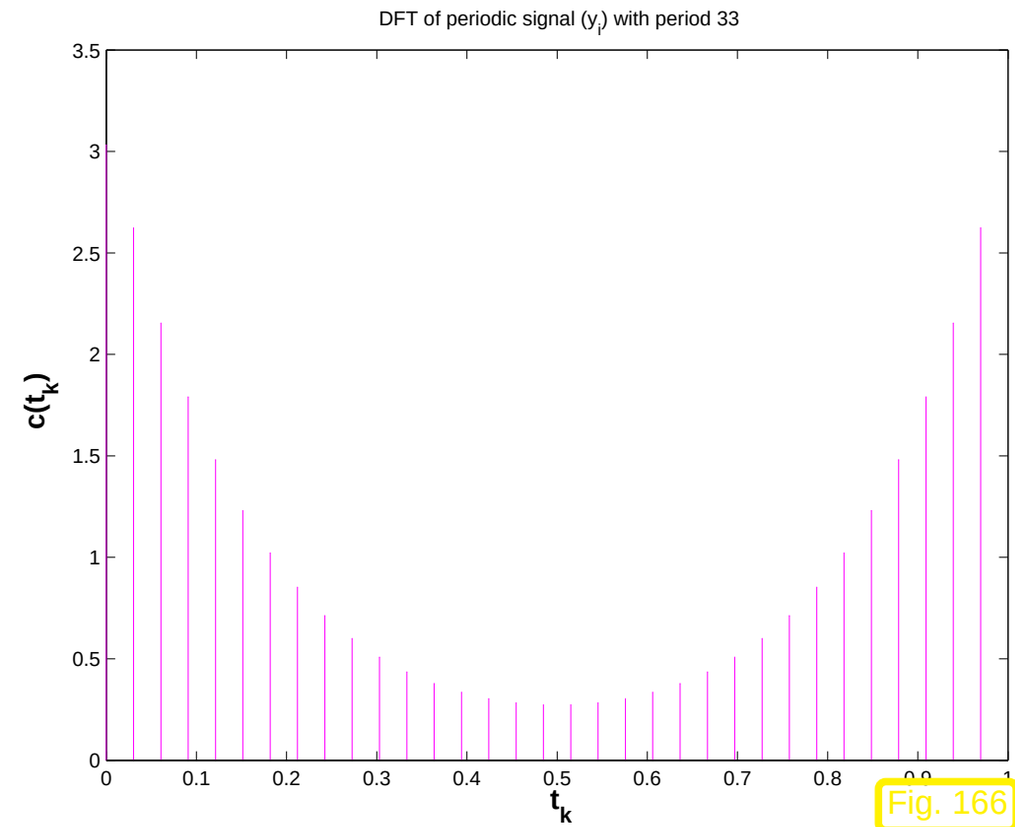


Fig. 166

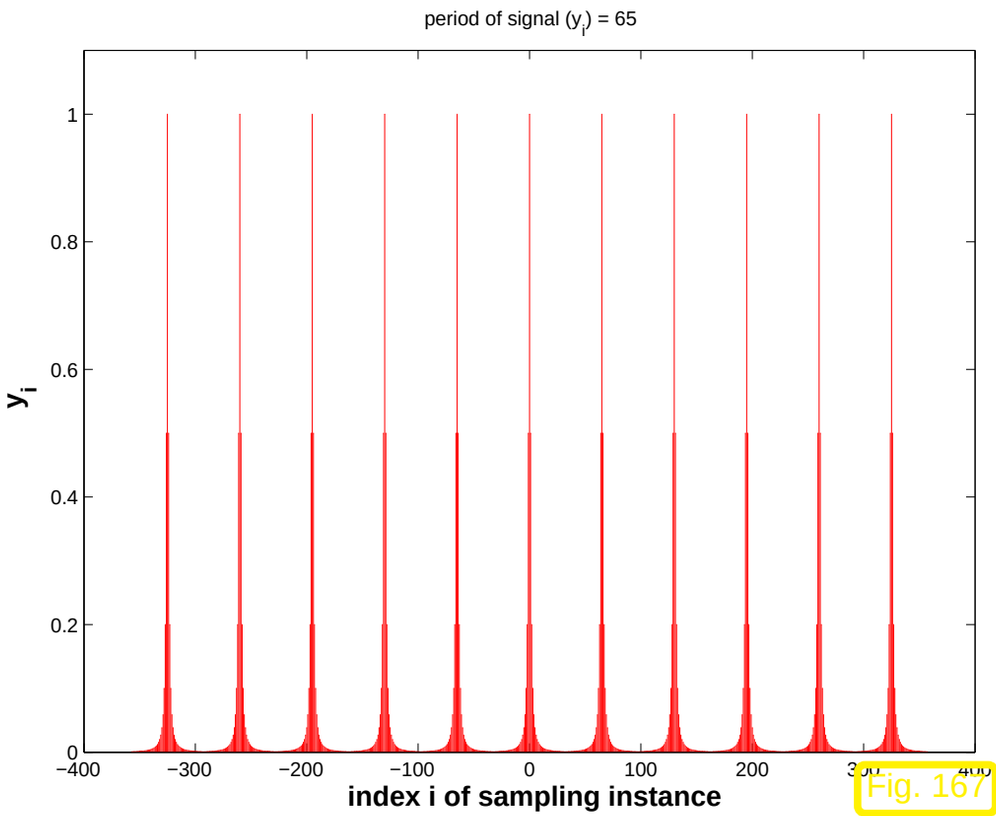


Fig. 167

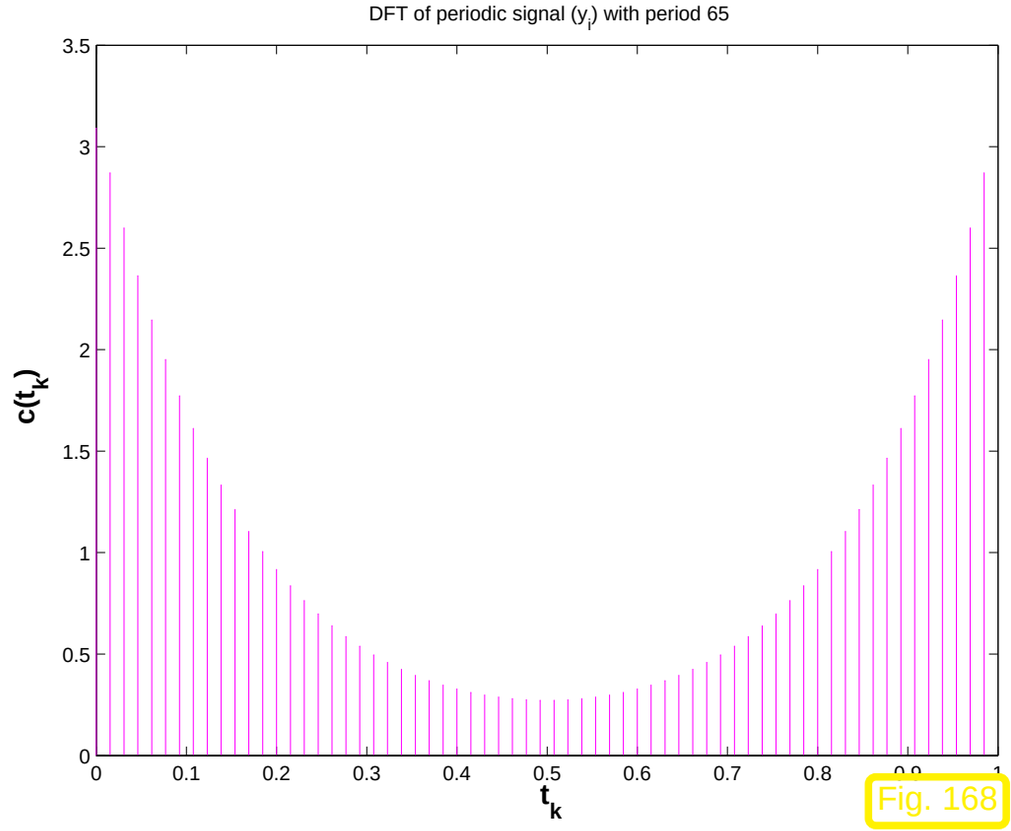


Fig. 168

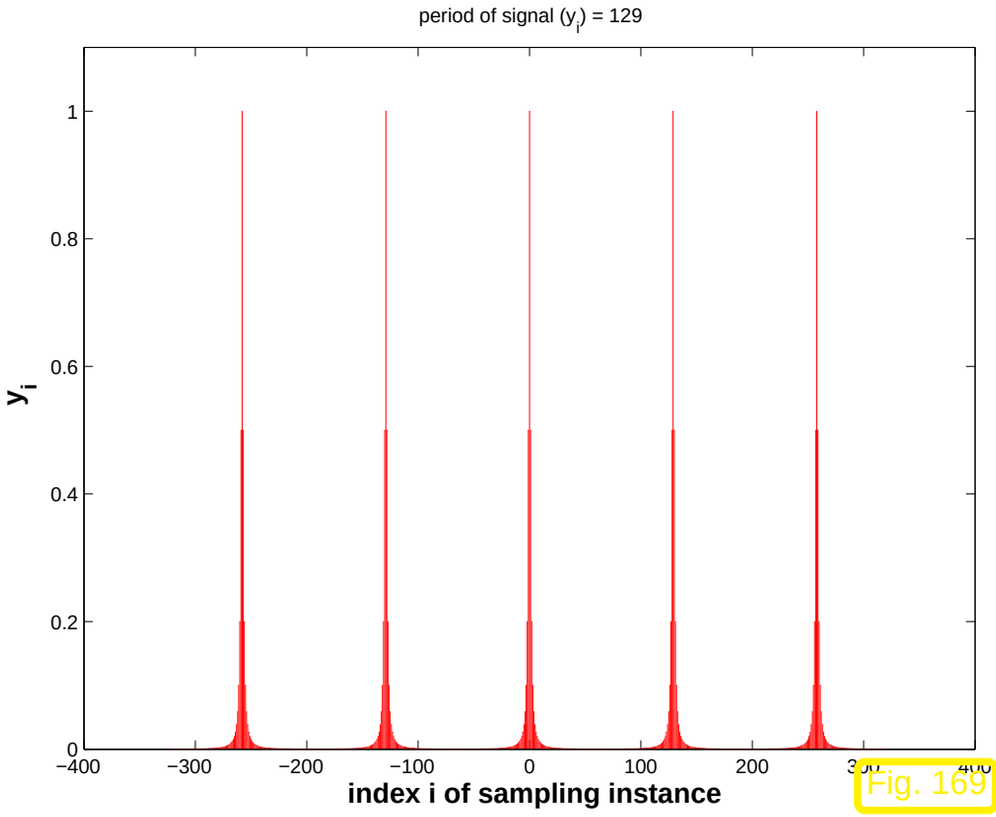


Fig. 169

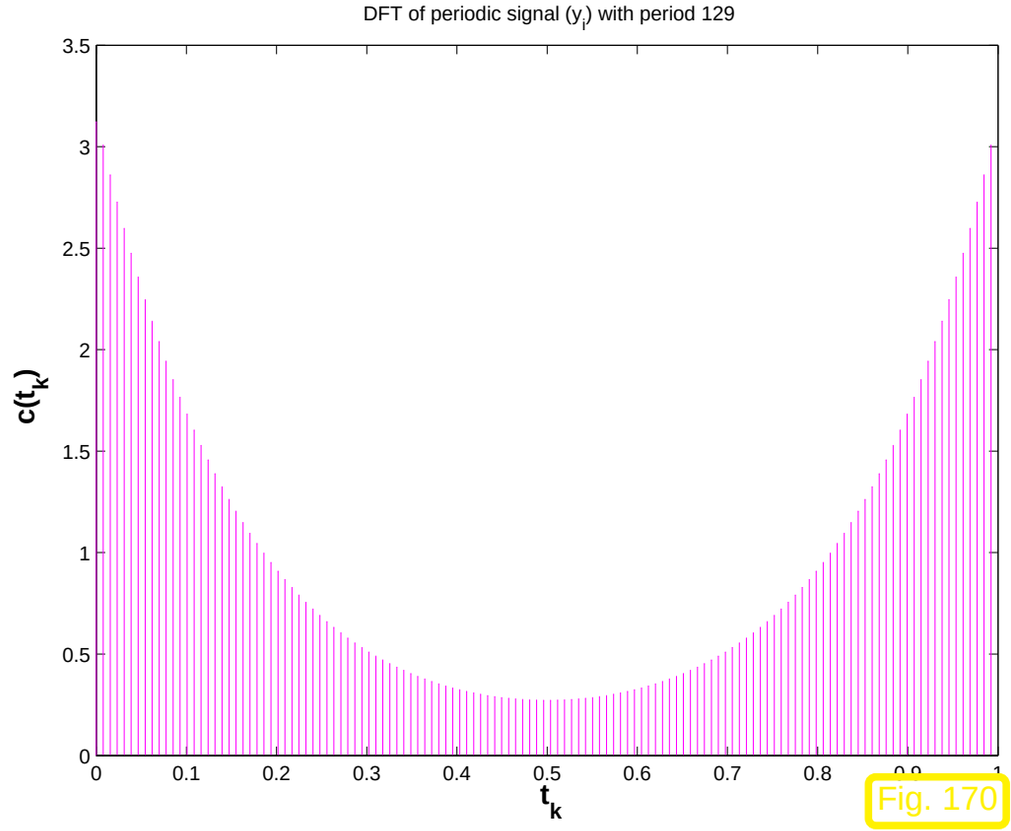


Fig. 170

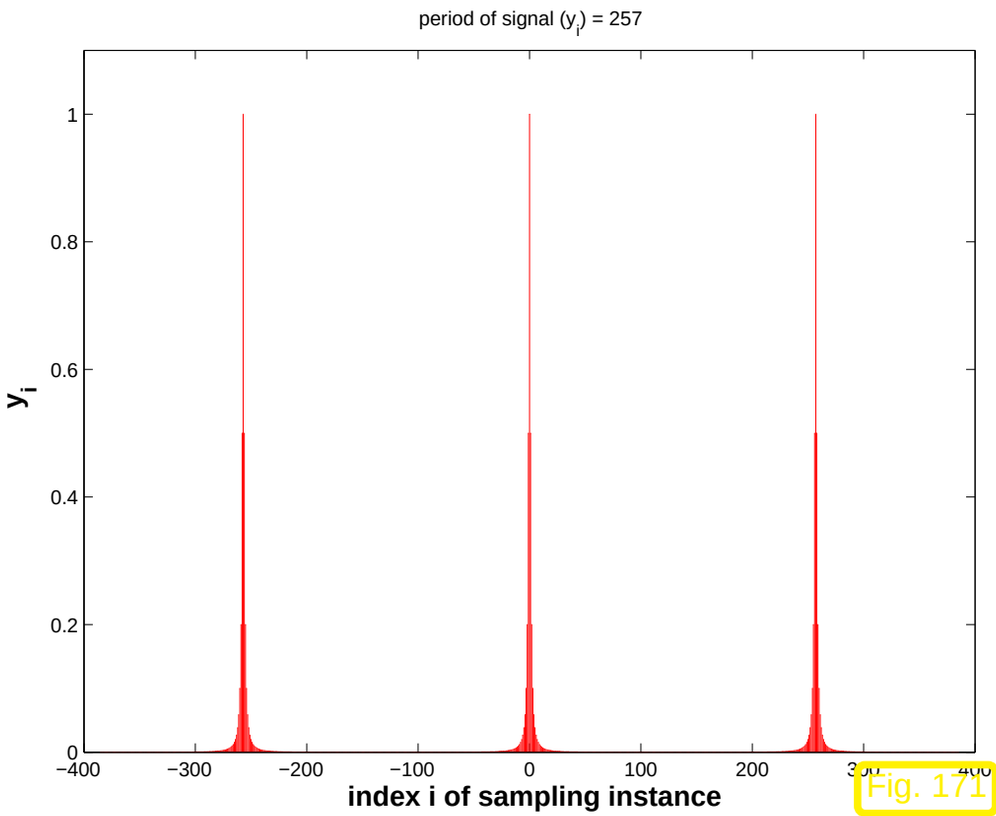


Fig. 171

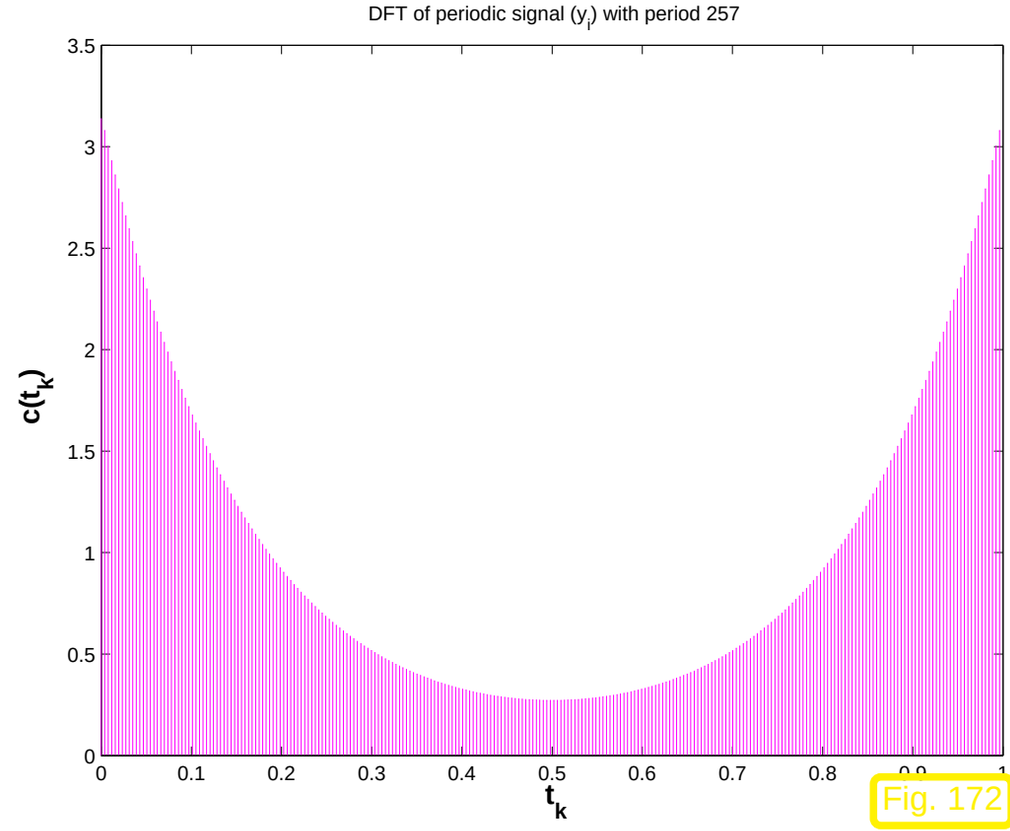


Fig. 172

◇

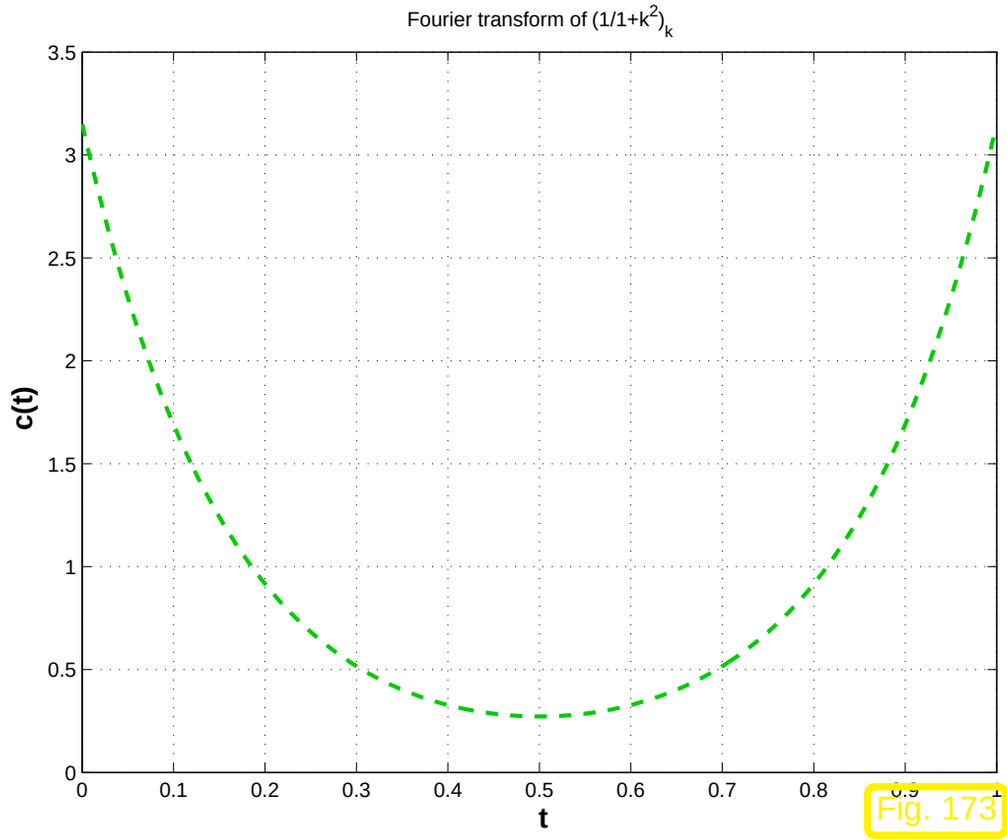


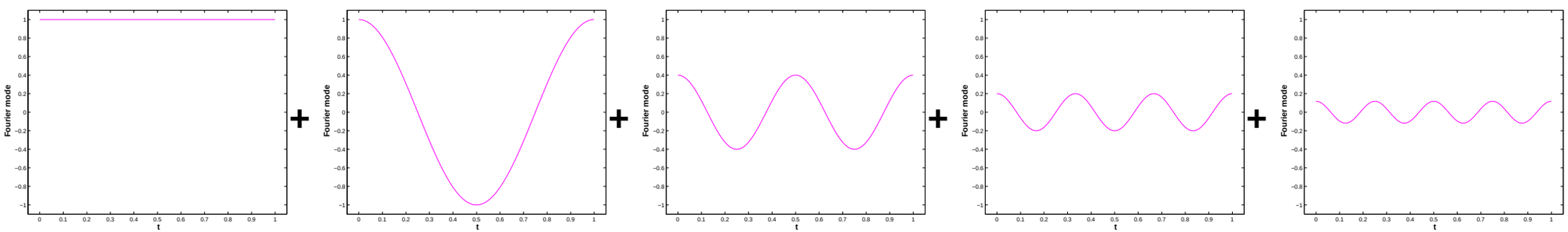
Fig. 173

◁ Fourier transform of $y_k := \frac{1}{1+k^2}$

Fourier transform

=

sum of Fourier modes



Example 8.2.55 (Convergence of Fourier sums).

R. Hiptmair

rev 38355, November 17, 2011

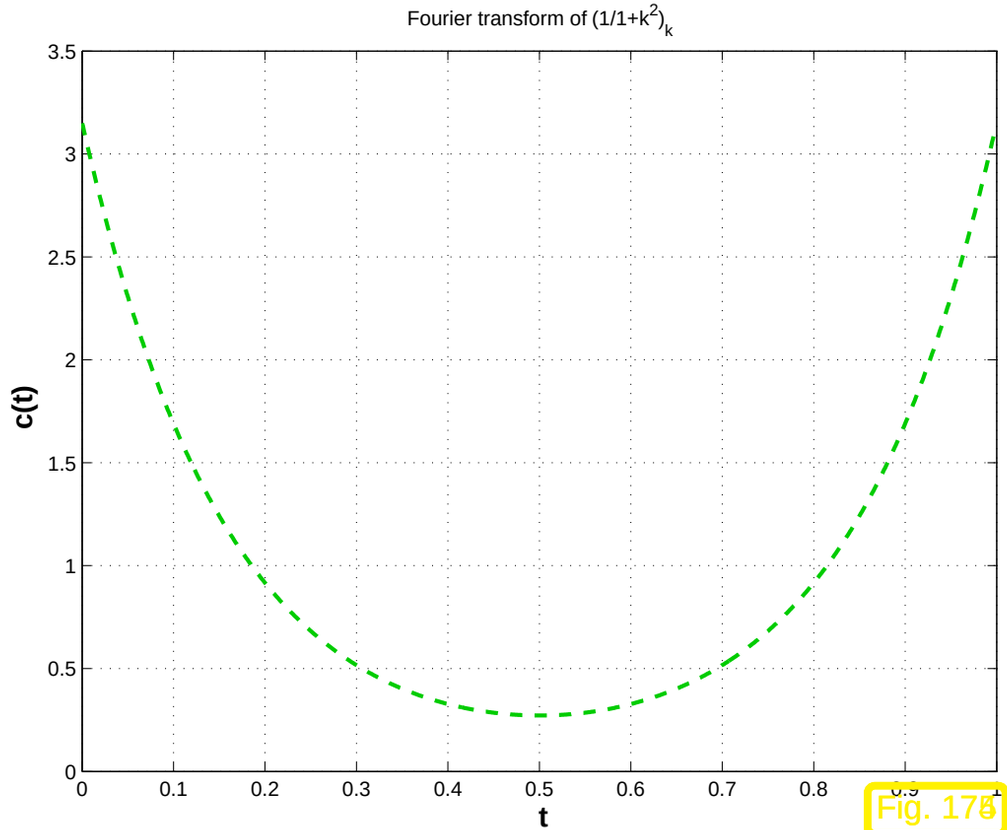


Fig. 175

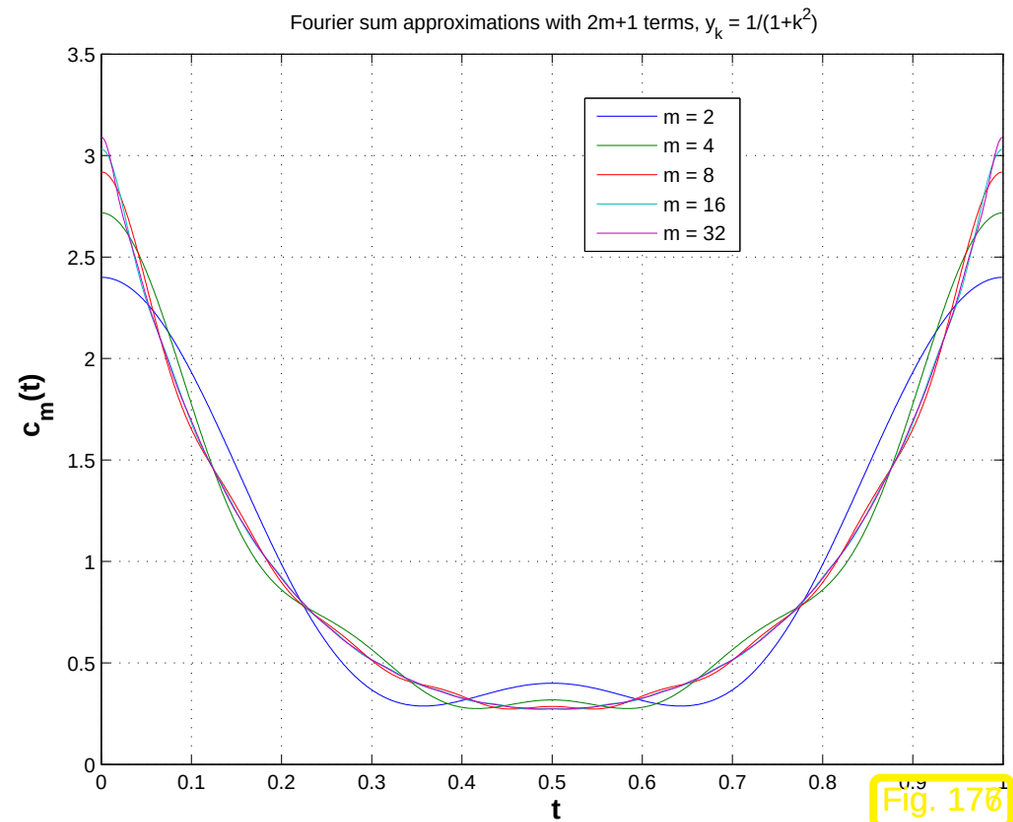


Fig. 176

◇

R. Hiptmair

rev 38355, November 17, 2011

(continuous) function $c : [0,1[\mapsto \mathbb{C}$

$$y_j = \int_0^1 c(t) \exp(2\pi i j t)\,\mathrm{d}t$$

$$\rightleftarrows$$

$$c(t) = \sum_{k\in\mathbb{Z}} y_k \exp(-2\pi i k t)$$

(bi-infinite) sequence $(y_k)_{k\in\mathbb{Z}}$

Fourier transform

Fourier coefficient

Remark 8.2.59 (Filtering in Fourier domain).

△

Theorem 8.2.65 (Isometry property of Fourier transform)**.**
If $\sum_{k\in\mathbb{Z}} |y_j|^2 < \infty$*, then*

$$c(t) = \sum_{k\in\mathbb{Z}} y_k \exp(-2\pi i k t) \quad \Rightarrow \quad \int_0^1 |c(t)|^2\,\mathrm{d}t = \sum_{k\in\mathbb{Z}} |y_j|^2 \,.$$

R. Hiptmair

rev 38355, November 17, 2011

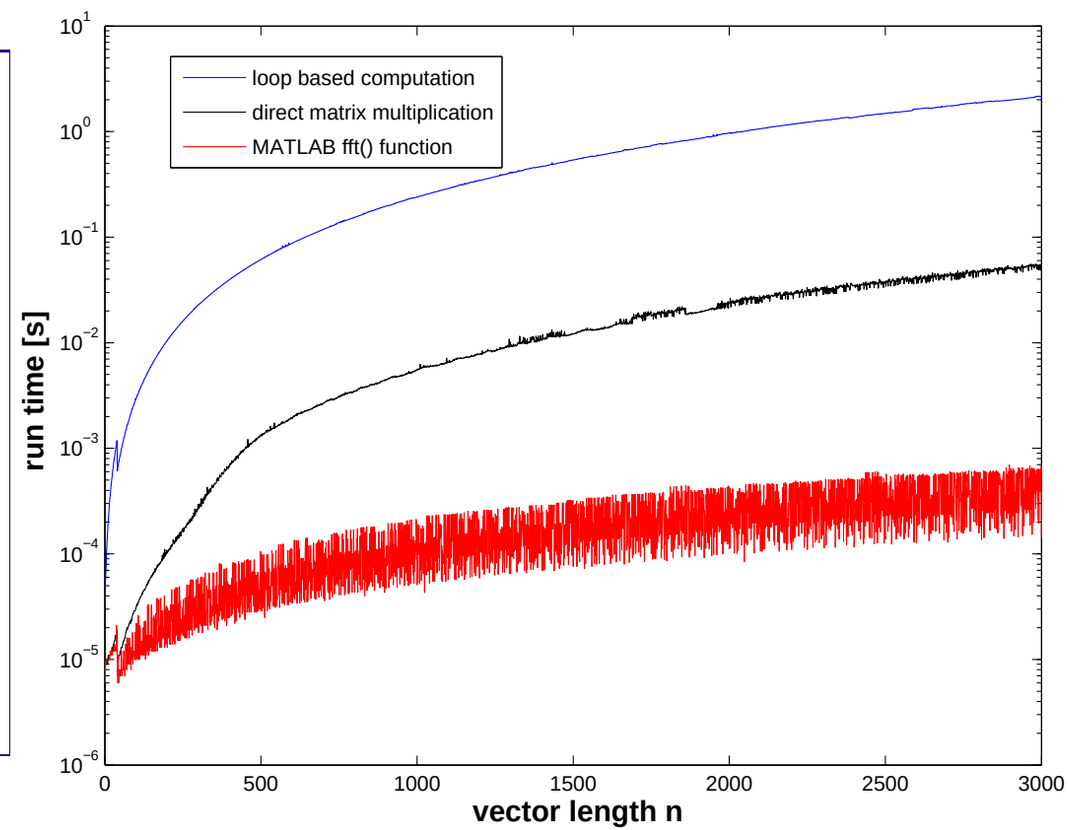
8.3 Fast Fourier Transform (FFT) [13, Sect. 8.7.3], [35, Sect. 53], [51, Sect. 10.9.2]

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)

MATLAB-CODE naive DFT-implementation

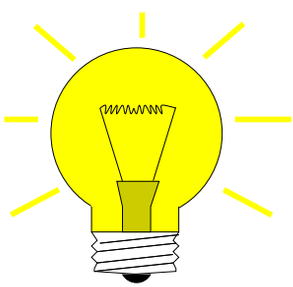
```
c = zeros(n,1);
omega = exp(-2*pi*i/n);
c(1) = sum(y); s = omega;
for j=2:n
  c(j) = y(n);
  for k=n-1:-1:1
    c(j) = c(j)*s+y(k);
  end
  s = s*omega;
end
```



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").



Idea:

divide & conquer recursion

(for DFT of length $n = 2^L$)

FFT-algorithm

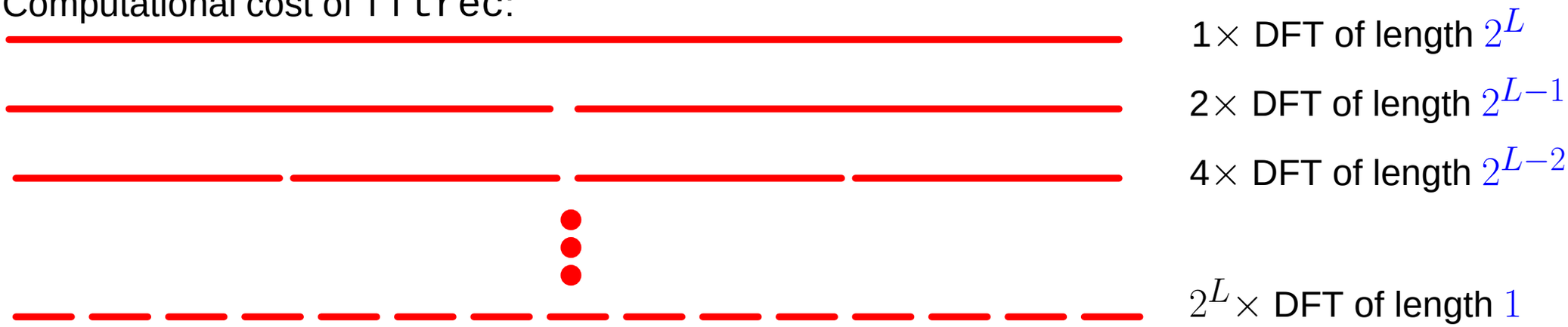
Code 8.3.3: Recursive FFT

```
function c = fftrec(y)
n = length(y);
if (n == 1), c = y; return;
else
  c1 = fftrec(y(1:2:n));
  c2 = fftrec(y(2:2:n));
  c = [c1;c1] +
    (exp(-2*pi*i/n).^((0:n-1)'))
    .*[c2;c2];
end
```

R. Hiptmair

rev 38355, November 17, 2011

Computational cost of `fftrec`:



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 $\approx 5n \log_2 n$).

➢

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 $\rightarrow$ Ex. 8.3.1.

Asymptotic complexity of `c=fft(y)` for $\mathbf{y} \in \mathbb{C}^n$ = $O(n \log n)$.

$\leftarrow$ Sect. 8.2.1

Asymptotic complexity of discrete periodic convolution/multiplication with circulant matrix, see Code 8.2.18:

Cost(`z = 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:

Cost(`z = myconv(h,x)`, $\mathbf{h}, \mathbf{x} \in \mathbb{C}^n$) = $O(n \log n)$.

8.4 Trigonometric transformations [35, Sect. 55]

8.4.1 Sine transform

Another trigonometric basis transform in $\mathbb{R}^{n-1}$, $n \in \mathbb{N}$:

Standard basis of $\mathbb{R}^{n-1}$ "Sine basis"

$$\left\{ \begin{pmatrix} 1 \\ 0 \\ \vdots \\ \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} \cdots \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ \\ \vdots \\ 0 \\ 1 \end{pmatrix} \right\} \longleftarrow \left\{ \begin{pmatrix} \sin(\frac{\pi}{n}) \\ \sin(\frac{2\pi}{n}) \\ \vdots \\ \\ \vdots \\ \sin(\frac{(n-1)\pi}{n}) \end{pmatrix} \begin{pmatrix} \sin(\frac{2\pi}{n}) \\ \sin(\frac{4\pi}{n}) \\ \vdots \\ \\ \vdots \\ \sin(\frac{2(n-1)\pi}{n}) \end{pmatrix} \cdots \begin{pmatrix} \sin(\frac{(n-1)\pi}{n} \\ \\ \sin(\frac{2(n-1)\pi}{n}) \\ \vdots \\ \vdots \\ \sin(\frac{(n-1)^2\pi}{n} \end{pmatrix} \right\}$$

Basis transform matrix (sine basis $\rightarrow$ standard basis): $\mathbf{S}_n := (\sin(jk\pi/n))_{j,k=1}^{n-1} \in \mathbb{R}^{n-1,n-1}$.

R. Hiptmair

rev 38355, November 17, 2011

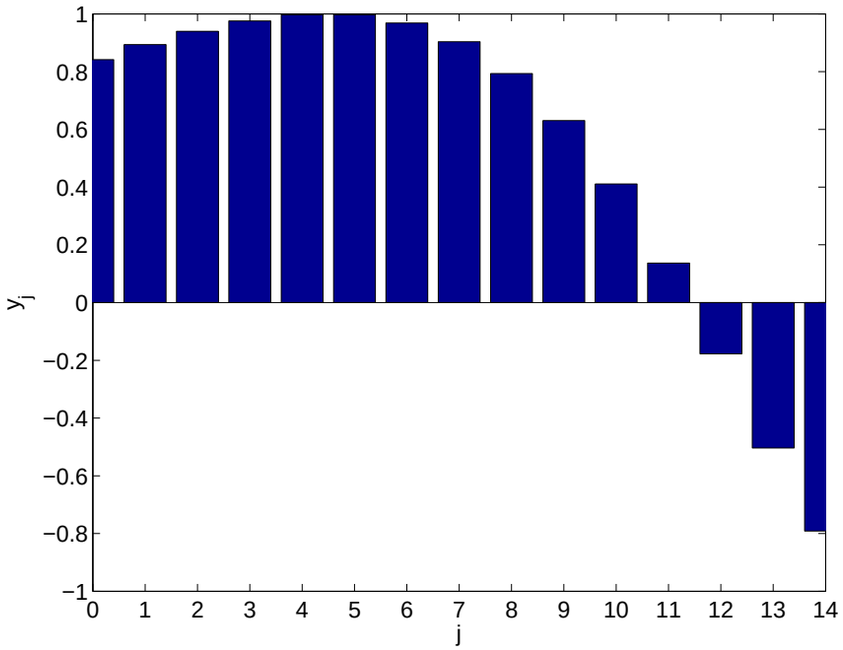
Lemma 8.4.1 (Properties of the sine matrix)**.**

$\sqrt{2/n}\,\mathbf{S}_n$ *is real, symmetric and orthogonal (→ Def. 2.8.5)*

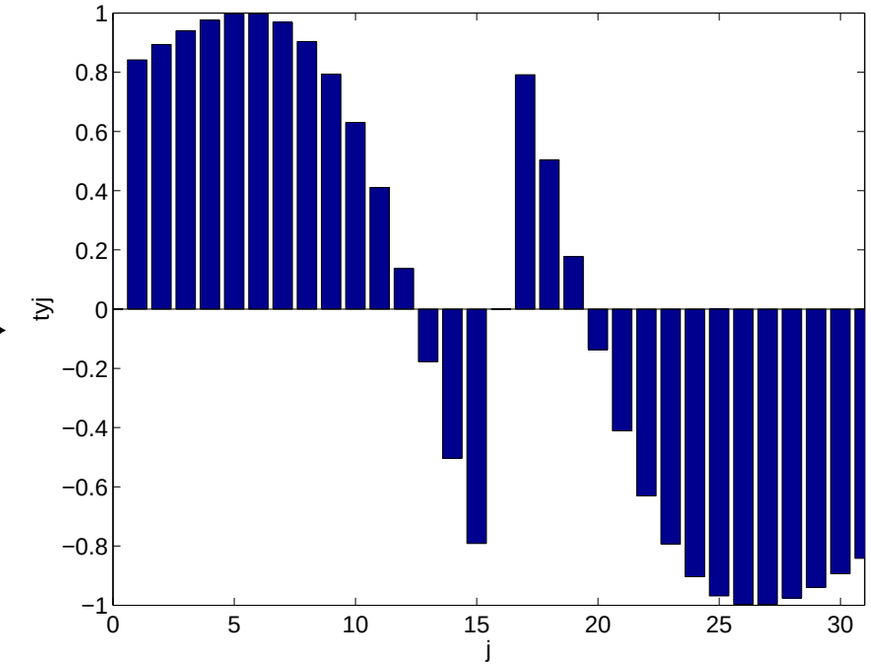
Sine transform: $$\boxed{s_k = \sum_{j=1}^{n-1} y_j \sin(\pi jk/n)} \quad , \quad k = 1, \dots, n-1 \; . \tag{8.4.2}$$

DFT-based algorithm for the sine transform ($\hat{=}$ $\mathbf{S}_n \times$vector):

"wrap around": $$\widetilde{\mathbf{y}} \in \mathbb{R}^{2n} : \widetilde{y}_j = \begin{cases} y_j & \text{, if } j = 1, \dots, n-1 \; , \\ 0 & \text{, if } j = 0, n \; , \\ -y_{2n-j} & \text{, if } j = n+1, \dots, 2n-1 \; . \end{cases} \qquad (\widetilde{\mathbf{y}} \text{ "odd"})$$



$\longrightarrow$



$$
\begin{aligned}
(\mathbf{F}_{2n}\widetilde{\mathbf{y}})_k &\overset{(8.2.15)}{=} \sum_{j=1}^{2n-1} \widetilde{y}_j e^{-\frac{2\pi}{2n}kj} \\
&= \sum_{j=1}^{n-1} y_j e^{-\frac{\pi}{n}kj} - \sum_{j=n+1}^{2n-1} y_{2n-j} e^{-\frac{\pi}{n}kj} \\
&= \sum_{j=1}^{n-1} y_j (e^{-\frac{\pi}{n}kj} - e^{\frac{\pi}{n}kj}) \\
&= -2i\,(\mathbf{S}_n\mathbf{y})_k \quad ,k = 1,\dots,n-1\,.
\end{aligned}
$$

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

R. Hiptmair

rev 38355, November 17, 2011

Remark 8.4.3 (Sine transform via DFT of half length).

Step ①: transform of the coefficients

$$\widetilde{y}_j = \sin(j\pi/n)(y_j + y_{n-j}) + \tfrac{1}{2}(y_j - y_{n-j}) \;, \quad j = 1, \dots, n-1 \quad , \quad \widetilde{y}_0 = 0 \;.$$

Step ②: real DFT (→ Sect. 8.2.3) of $(\widetilde{y}_0, \dots, \widetilde{y}_{n-1}) \in \mathbb{R}^n$: $\qquad c_k := \sum_{j=0}^{n-1} \widetilde{y}_j \, e^{-\frac{2\pi i}{n} jk}$

Hence
$$\begin{aligned}
\mathrm{Re}\{c_k\} &= \sum_{j=0}^{n-1} \widetilde{y}_j \, \cos(-\tfrac{2\pi i}{n} jk) = \sum_{j=1}^{n-1} (y_j + y_{n-j}) \sin(\tfrac{\pi j}{n}) \cos(\tfrac{2\pi i}{n} jk) \\
&= \sum_{j=0}^{n-1} 2 y_j \sin(\tfrac{\pi j}{n}) \cos(\tfrac{2\pi i}{n} jk) = \sum_{j=0}^{n-1} y_j \left(\sin(\frac{2k+1}{n}\pi j) - \sin(\frac{2k-1}{n}\pi j) \right) \\
&= s_{2k+1} - s_{2k-1} \;. \\
\mathrm{Im}\{c_k\} &= \sum_{j=0}^{n-1} \widetilde{y}_j \sin(-\tfrac{2\pi i}{n} jk) = -\sum_{j=1}^{n-1} \tfrac{1}{2}(y_j - y_{n-j}) \sin(\tfrac{2\pi i}{n} jk) = -\sum_{j=1}^{n-1} y_j \sin(\tfrac{2\pi i}{n} jk) \\
&= -s_{2k} \;.
\end{aligned}$$

R. Hiptmair

rev 38355, November 17, 2011

Step ③: extraction of s_k

s_{2k+1} , $k = 0, \dots, \frac{n}{2} - 1$ ➤ from recursion $s_{2k+1} - s_{2k-1} = \mathrm{Re}\{c_k\}$, $s_1 = \sum_{j=1}^{n-1} y_j \sin(\pi j/n)$,

s_{2k}, $k = 1, \dots, \frac{n}{2} - 2$ ➤ $s_{2k} = -\mathrm{Im}\{c_k\}$.

MATLAB-Implementation (via a `fft` of length $n/2$):

MATLAB-CODE Sine transform

```
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
```

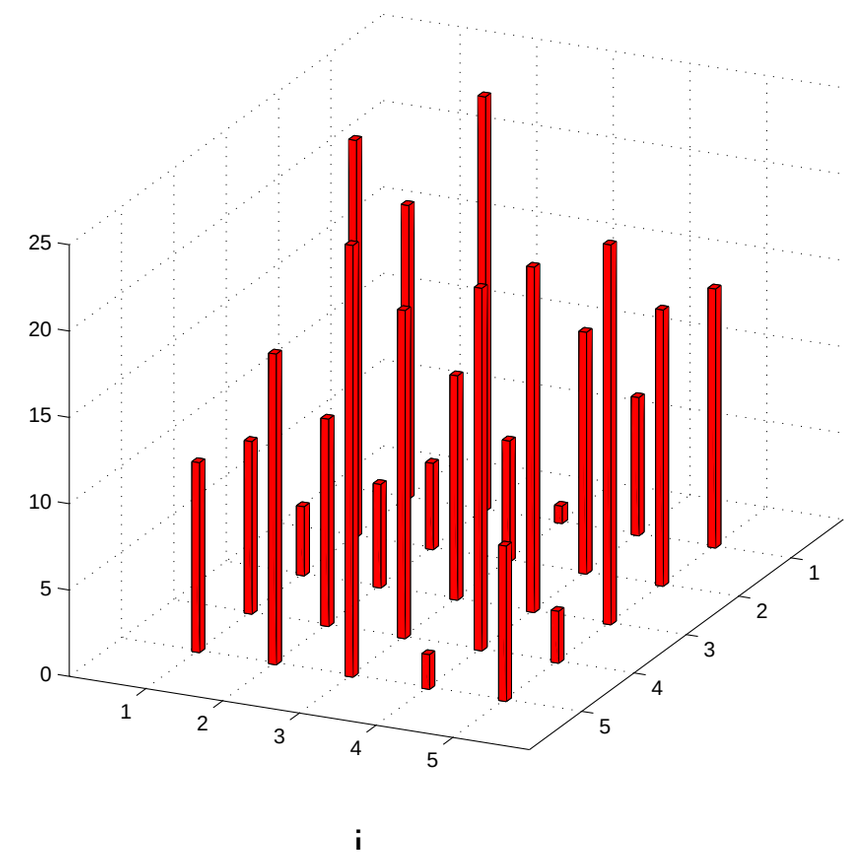
Application: diagonalization of local translation invariant linear operators.

5-points-stencil-operator on $\mathbb{R}^{n,n}$, $n \in \mathbb{N}$, in grid representation:

$$T : \mathbb{R}^{n,n} \mapsto \mathbb{R}^{n,n}, \quad \begin{array}{c} \mathbf{X} \mapsto T(\mathbf{X}) \\ (T(\mathbf{X}))_{ij} := cx_{ij} + c_y x_{i,j+1} + c_y x_{i,j-1} + c_x x_{i+1,j} + c_x x_{i-1,j} \end{array}$$

with $c, c_y, c_x \in \mathbb{R}$, convention: $x_{ij} := 0$ for $(i,j) \notin \{1,\dots,n\}^2$.

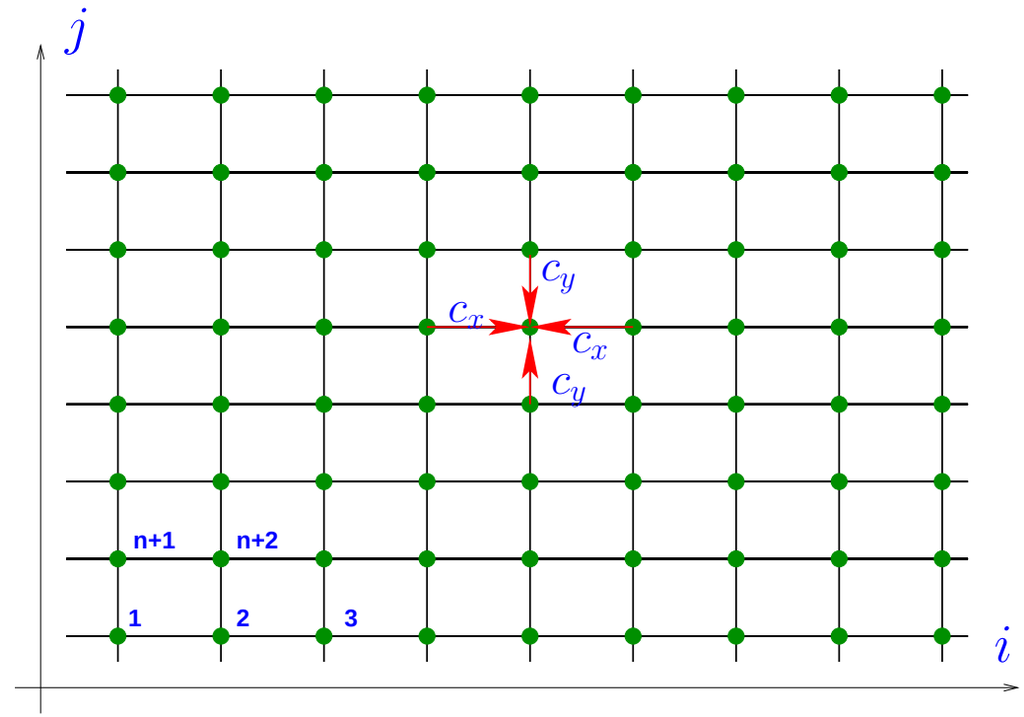
$$\mathbf{X} \in \mathbb{R}^{n,n}$$
$$\updownarrow$$
grid function $\in \{1,\dots,n\}^2 \mapsto \mathbb{R}$



Identification $\mathbb{R}^{n,n} \cong \mathbb{R}^{n^2}$, $x_{ij} \sim \widetilde{x}_{(j-1)n+i}$ gives matrix representation $\mathbf{T} \in \mathbb{R}^{n^2,n^2}$ of T:

$$\mathbf{T} = \begin{pmatrix} \mathbf{C} & c_y\mathbf{I} & 0 & \cdots & \cdots & 0 \\ c_y\mathbf{I} & \mathbf{C} & c_y\mathbf{I} & & & \vdots \\ 0 & \ddots & \ddots & \ddots & & \\ \vdots & & & c_y\mathbf{I} & \mathbf{C} & c_y\mathbf{I} \\ 0 & \cdots & \cdots & 0 & c_y\mathbf{I} & \mathbf{C} \end{pmatrix} \in \mathbb{R}^{n^2,n^2} ,$$

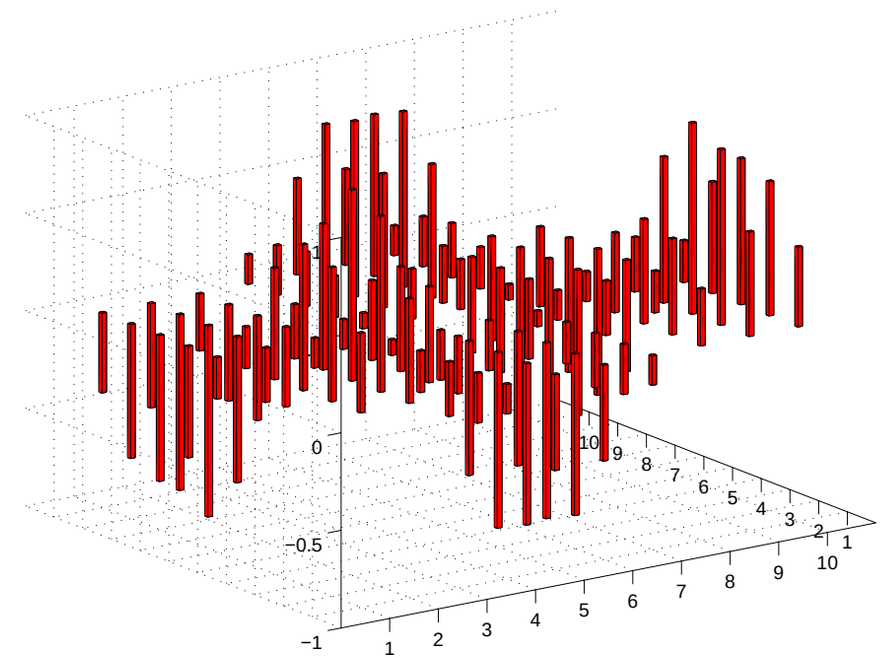
$$\mathbf{C} = \begin{pmatrix} c & c_x & 0 & \cdots & \cdots & 0 \\ c_x & c & c_x & & & \vdots \\ 0 & \ddots & \ddots & \ddots & & \\ \vdots & & & c_x & c & c_x \\ 0 & \cdots & \cdots & 0 & c_x & c \end{pmatrix} \in \mathbb{R}^{n,n} .$$



Sine basis of $\mathbb{R}^{n,n}$:

$$\mathbf{B}^{kl} = \left(\sin(\tfrac{\pi}{n+1}ki)\sin(\tfrac{\pi}{n+1}lj)\right)_{i,j=1}^{n} \quad . \qquad (8.4.4)$$

$n = 10$: grid function $\mathbf{B}^{2,3}$ ➢



$$\begin{aligned}(T(\mathbf{B}^{kl}))_{ij} &= c\sin(\tfrac{\pi}{n}ki)\sin(\tfrac{\pi}{n}lj) + c_y\sin(\tfrac{\pi}{n}ki)\left(\sin(\tfrac{\pi}{n+1}l(j-1)) + \sin(\tfrac{\pi}{n+1}l(j+1))\right) + \\ &\qquad c_x\sin(\tfrac{\pi}{n}lj)\left(\sin(\tfrac{\pi}{n+1}k(i-1)) + \sin(\tfrac{\pi}{n+1}k(i+1))\right) \\ &= \sin(\tfrac{\pi}{n}ki)\sin(\tfrac{\pi}{n}lj)(c + 2c_y\cos(\tfrac{\pi}{n+1}l) + 2c_x\cos(\tfrac{\pi}{n+1}k))\end{aligned}$$

Hence $\mathbf{B}^{kl}$ is eigenvector of $T \leftrightarrow \mathbf{T}$ corresponding to eigenvalue $c + 2c_y\cos(\frac{\pi}{n+1}l) + 2c_x\cos(\frac{\pi}{n+1}k)$.

R. Hiptmair

rev 38355, November 17, 2011

Algorithm for basis transform:

$$\mathbf{X} = \sum_{k=1}^{n}\sum_{l=1}^{n} y_{kl}\mathbf{B}^{kl} \quad \Rightarrow \quad x_{ij} = \sum_{k=1}^{n} \sin(\tfrac{\pi}{n+1}ki) \sum_{l=1}^{n} y_{kl} \sin(\tfrac{\pi}{n+1}lj) \; .$$

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.

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;
```

MATLAB-CODE FFT-based solution of local translation invariant linear operators

```
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));
```

Diagonalization of $\mathbf{T}$

via 2D sine transform

$\Downarrow$

efficient algorithm

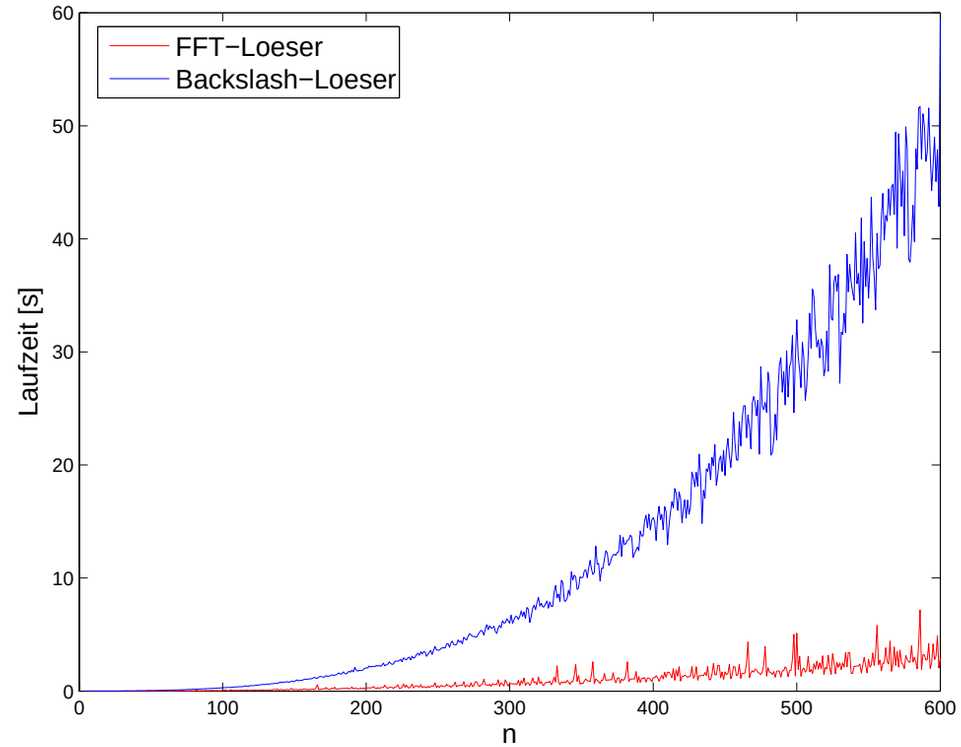
for solving linear system of equations $T(\mathbf{X}) = \mathbf{B}$

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"

$$\left\{ \begin{pmatrix} 1 \\ 0 \\ \vdots \\ \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} \cdots \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ \\ \vdots \\ 0 \\ 1 \end{pmatrix} \right\} \leftarrow \left\{ \begin{pmatrix} 2^{-1/2} \\ \cos(\frac{\pi}{2n}) \\ \cos(\frac{2\pi}{2n}) \\ \vdots \\ \\ \vdots \\ \cos(\frac{(n-1)\pi}{2n}) \end{pmatrix} \begin{pmatrix} 2^{-1/2} \\ \cos(\frac{3\pi}{2n}) \\ \cos(\frac{6\pi}{2n}) \\ \vdots \\ \\ \vdots \\ \cos(\frac{3(n-1)\pi}{2n}) \end{pmatrix} \cdots \begin{pmatrix} 2^{-1/2} \\ \cos(\frac{(2n-1)\pi}{2n}) \\ \cos(\frac{2(2n-1)\pi}{2n}) \\ \vdots \\ \\ \vdots \\ \cos(\frac{(n-1)(2n-1)\pi}{2n}) \end{pmatrix} \right\} .$$

Basis transform matrix (cosine basis $\to$ 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)**.**

$\sqrt{2/n}\,\mathbf{C}_n$ *is real and orthogonal (*$\to$ *Def. 2.8.5).*

R. Hiptmair

rev 38355, November 17, 2011

cosine transform: $$\boxed{c_k = \sum_{j=0}^{n-1} y_j \cos(k\tfrac{2j+1}{2n}\pi)} \quad , \quad k = 1, \dots, n-1 \; , \tag{8.4.7}$$

$$c_0 = \frac{1}{\sqrt{2}} \sum_{j=0}^{n-1} y_j \; .$$

MATLAB-implementation of $\mathbf{C}\mathbf{y}$ ("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 $\mathbf{C}_n^{-1}\mathbf{y}$ ("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);
```

8.5 Toeplitz matrix techniques

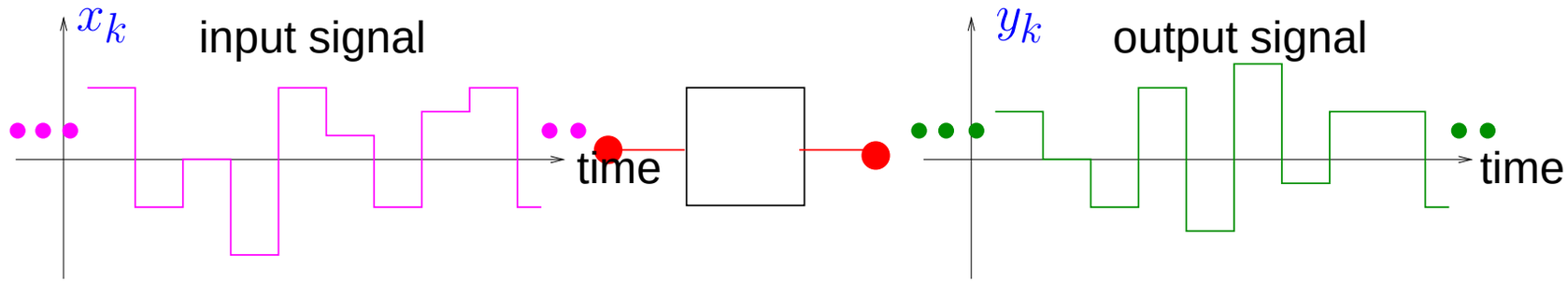
Example 8.5.1 (Parameter identification for linear time-invariant filters).

- $(x_k)_{k\in\mathbb{Z}}$ m-periodic discrete signal = *known* input
- $(y_k)_{k\in\mathbb{Z}}$ m-periodic *measured*$^{(*)}$ output signal of a linear time-invariant filter, see Ex. 8.1.1.

  $(*)$ ➔ measurement errors !
- Known: impulse response of filter has maximal duration $n\Delta t$, $n \in \mathbb{N}$, $n \leq m$

cf. (8.1.3) ➤

$$\exists \mathbf{h} = (h_0,\dots,h_{n-1})^\top \in \mathbb{R}^n, \quad n \leq m : \quad y_k = \sum_{j=0}^{n-1} h_j x_{k-j} \; . \tag{8.5.2}$$



Parameter identification problem: seek $\mathbf{h} = (h_0, \dots, h_{n-1})^\top \in \mathbb{R}^n$ with

$$\|\mathbf{A}\mathbf{h} - \mathbf{y}\|_2 = \left\| \begin{pmatrix} x_0 & x_{-1} & \cdots & & \cdots & x_{1-n} \\ x_1 & x_0 & x_{-1} & & & \vdots \\ \vdots & x_1 & x_0 & \ddots & & \\ & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & x_{-1} \\ x_{n-1} & & & & x_1 & x_0 \\ x_n & x_{n-1} & & & & x_1 \\ \vdots & & & & & \vdots \\ x_{m-1} & \cdots & & & \cdots & x_{m-n} \end{pmatrix} \begin{pmatrix} h_0 \\ \vdots \\ \\ \vdots \\ h_{n-1} \end{pmatrix} - \begin{pmatrix} y_0 \\ \vdots \\ \\ \vdots \\ y_{m-1} \end{pmatrix} \right\|_2 \to \min \ .$$

➢ Linear least squares problem, $\to$ Ch. 7 with Toeplitz matrix $\mathbf{A}$: $(\mathbf{A})_{ij} = x_{i-j}$.

◇

R. Hiptmair

rev 38355, November 17, 2011

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* $t_{ij} = u_{j-i}$, $1 \leq i \leq m$, $1 \leq j \leq n$.

$$\mathbf{T} = \begin{pmatrix} u_0 & u_1 & \cdots & & \cdots & u_{n-1} \\ u_{-1} & u_0 & u_1 & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & u_1 \\ u_{1-m} & \cdots & & \cdots & u_{-1} & u_0 \end{pmatrix}$$

8.5.1 Toeplitz matrix arithmetic

$$\mathbf{C} = \begin{pmatrix} \mathbf{T} & \mathbf{S} \\ \mathbf{S} & \mathbf{T} \end{pmatrix} = \left(\begin{array}{cccccc|cccccc}
u_0 & u_1 & \cdots & & \cdots & u_{n-1} & 0 & u_{1-n} & \cdots & & \cdots & u_{-1} \\
u_{-1} & u_0 & u_1 & & & \vdots & u_{n-1} & 0 & \ddots & & & \vdots \\
\vdots & \ddots & \ddots & \ddots & & \vdots & \vdots & \ddots & \ddots & & & \\
\vdots & & \ddots & \ddots & \ddots & \vdots & & & & & \ddots & \\
\vdots & & & \ddots & \ddots & u_1 & \vdots & & & \ddots & \ddots & u_{1-n} \\
u_{1-n} & \cdots & & \cdots & u_{-1} & u_0 & u_1 & & & & u_{n-1} & 0 \\
\hline
0 & u_{1-n} & \cdots & & \cdots & u_{-1} & u_0 & u_1 & \cdots & & \cdots & u_{n-1} \\
u_{n-1} & 0 & \ddots & & & \vdots & u_{-1} & u_0 & u_1 & & & \vdots \\
\vdots & \ddots & \ddots & & & & \vdots & \ddots & \ddots & \ddots & & \vdots \\
 & & & & \ddots & & \vdots & & \ddots & \ddots & \ddots & \vdots \\
\vdots & & & \ddots & \ddots & u_{1-n} & \vdots & & & \ddots & \ddots & u_1 \\
u_1 & & & & u_{n-1} & 0 & u_{1-n} & \cdots & & \cdots & u_{-1} & u_0
\end{array}\right)$$

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)]);
```

$$\mathbf{C}\begin{pmatrix} \mathbf{x} \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{Tx} \\ \mathbf{Sx} \end{pmatrix}$$

zero padding

Computational effort $O(n \log n)$ for computing $\mathbf{Tx}$ (FFT based, Sect. 8.3)

8.5.2 The Levinson algorithm

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 $O(n^2)$

Code 8.5.9: Levinson algorithm

```
function [x,y] = levinson(u,b)
k = length(u)-1;
if (k == 0), x=b(1); y = u(1);
  return; end
[xk,yk] =
  levinson(u(1:k),b(1:k));
sigma = 1-dot(u(1:k),yk);
t =
  (b(k+1)-dot(u(k:-1:1),xk))/sigma;
x = [ xk-t*yk(k:-1:1);t];
s =
  (u(k+1)-dot(u(k:-1:1),yk))/sigma;
y = [yk-s*yk(k:-1:1); s];
```

9 Approximation of Functions in 1D

9.1 Error estimates for polynomial interpolation

We consider Lagrangian polynomial interpolation on node set

$$\mathcal{T} := \{t_0, \dots, t_n\} \subset I,\ I \subset \mathbb{R}, \text{ interval of length } |I|.$$

R. Hiptmair

rev 38355, October 27, 2011

Goal: estimate of the interpolation error norm $\|f - \mathsf{I}_{\mathcal{T}} f\|$ (for some norm on $C(I)$).

Focus: asymptotic behavior of interpolation error for $n \to \infty$

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 $p := \mathsf{I}_{\mathcal{T}} f \in \mathcal{P}_n$.

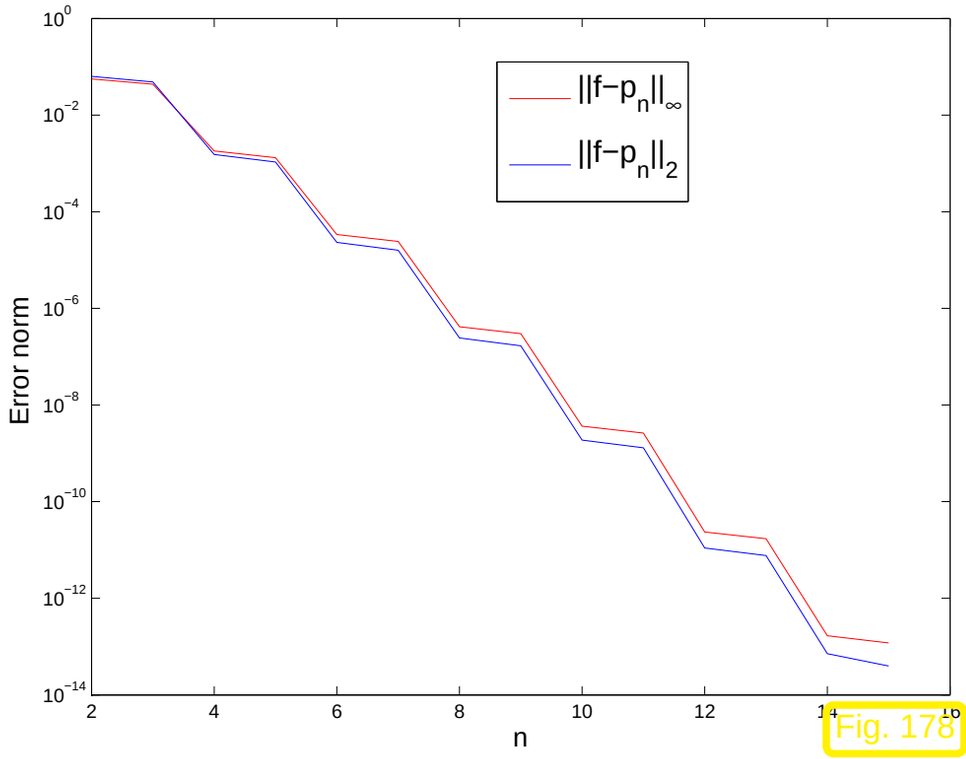


Fig. 178

◇

$$\exists\, C \neq C(n) > 0: \quad \|f - \mathsf{I}_{\mathcal{T}} f\| \leq C\, T(n) \quad \text{for } n \to \infty\ . \tag{9.1.3}$$

R. Hiptmair

rev 38355, October 27, 2011

Classification (best bound for $T(n)$):

$$\begin{array}{lll} \exists\, p > 0: & T(n) \leq n^{-p} & : \text{ algebraic convergence, with rate } p > 0 \,, \\ \exists\, 0 < q < 1: & T(n) \leq q^{n} & : \text{ exponential convergence} \,, \end{array} \quad \forall n \in \mathbb{N} \,.$$

Example 9.1.4 (Runge's example). $\rightarrow$ Ex. 3.5.1

Polynomial interpolation of $f(t) = \frac{1}{1+t^2}$ with equispaced nodes:

$$\mathcal{T} := \left\{ t_j := -5 + \frac{10}{n}\, j \right\}_{j=0}^{n} \,, \quad y_j = \frac{1}{1+t_j^2} \,.j = 0, \ldots, n \,.$$

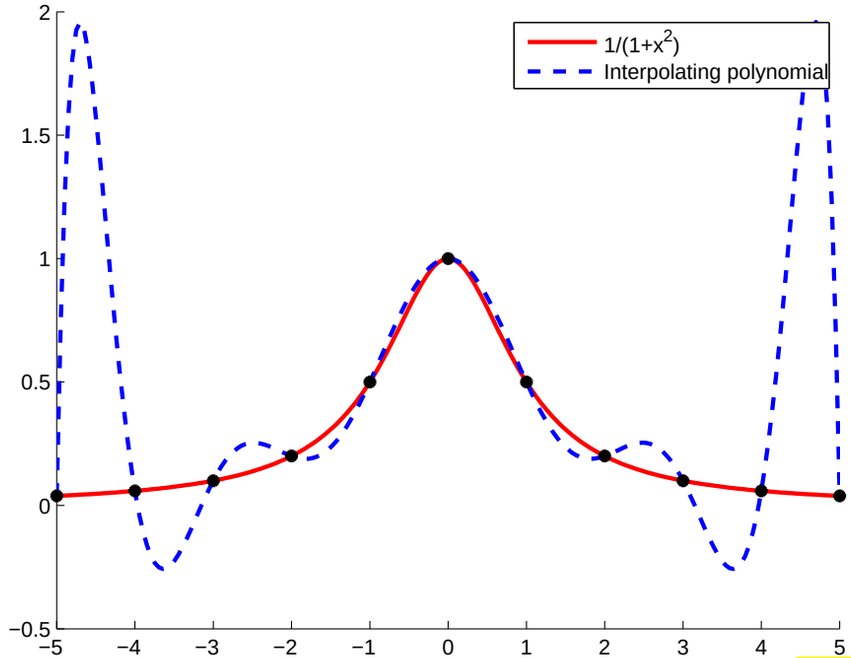


Fig. 179

Interpolating polynomial, $n = 10$

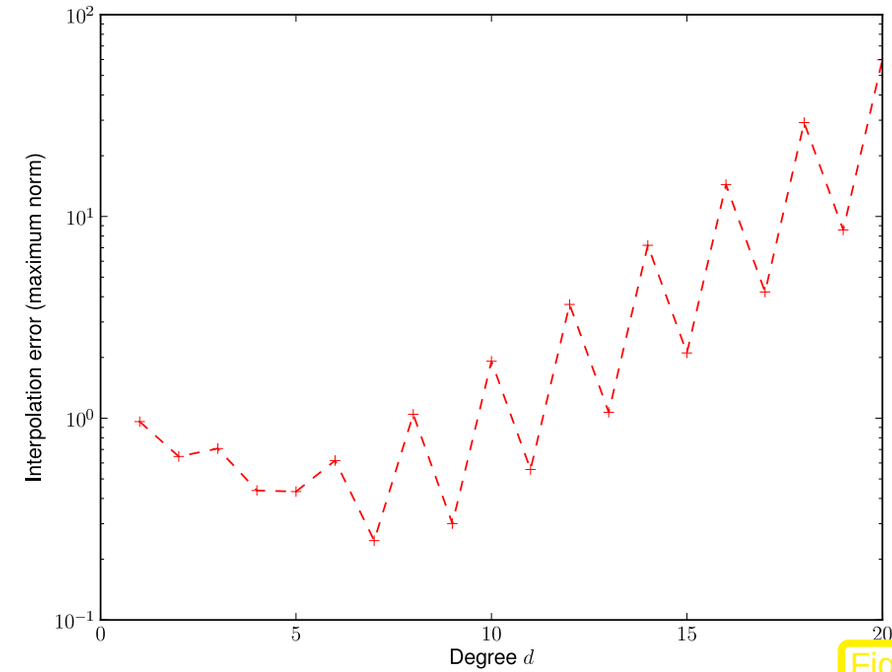


Fig. 180

Approximate $\|f - \mathcal{I}_{\mathcal{T}} f\|_\infty$ on $[-5, 5]$

◇

Theorem 9.1.6 (Representation of interpolation error)**.** *[13, Thm. 8.22], [35, Thm. 37.4]*

$f \in C^{n+1}(I)$: $\forall\, t \in I$: $\exists\, \tau_t \in]\min\{t, t_0, \dots, t_n\}, \max\{t, t_0, \dots, t_n\}[$:

$$f(t) - \mathsf{I}_{\mathcal{T}}(f)(t) = \frac{f^{(n+1)}(\tau_t)}{(n+1)!} \cdot \prod_{j=0}^{n} (t - t_j) \; . \tag{9.1.7}$$

R. Hiptmair

rev 38355, October 27, 2011

Thm. 9.1.7 $\Rightarrow$ $$\|f - \mathsf{I}_{\mathcal{T}} f\|_{L^\infty(I)} \leq \frac{\left\|f^{(n+1)}\right\|_{L^\infty(I)}}{(n+1)!} \max_{t\in I} |(t-t_0)\cdot\cdots\cdot(t-t_n)| \ . \qquad (9.1.11)$$

Interpolation error estimate requires smoothness!

9.2 Chebychev Interpolation

9.2.1 Motivation and definition

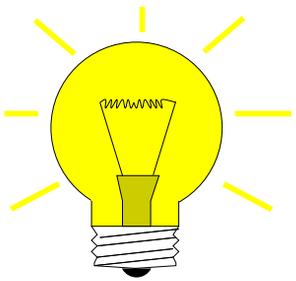
Setting: Mesh of nodes: $\mathcal{T} := \{t_0 < t_1 < \cdots < t_{n-1} < t_n\}$, $n \in \mathbb{N}$,
function $f : I \to \mathbb{R}$ continuous; without loss of generality $I = [-1, 1]$.

Thm. 9.1.7:

$$\|f-p\|_{L^\infty(I)} \leq \frac{1}{(n+1)!}\left\|f^{(n+1)}\right\|_{L^\infty(I)} \|w\|_{L^\infty(I)} \ ,$$

$$w(t) := (t-t_0)\cdot\cdots\cdot(t-t_n) \ .$$

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.

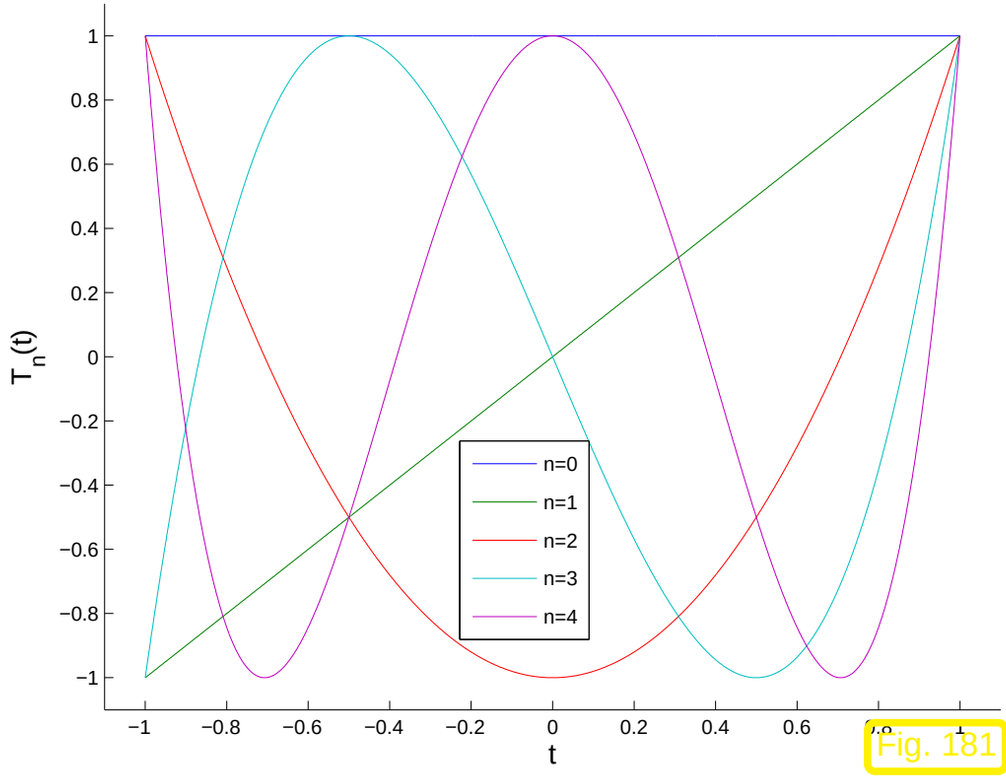
▶ Choice of $t_0,\dots,t_n$ = zeros of q (caution: t_j must belong to I).

Definition 9.2.1 (Chebychev polynomial). $\rightarrow$ *[35, Ch. 32]*

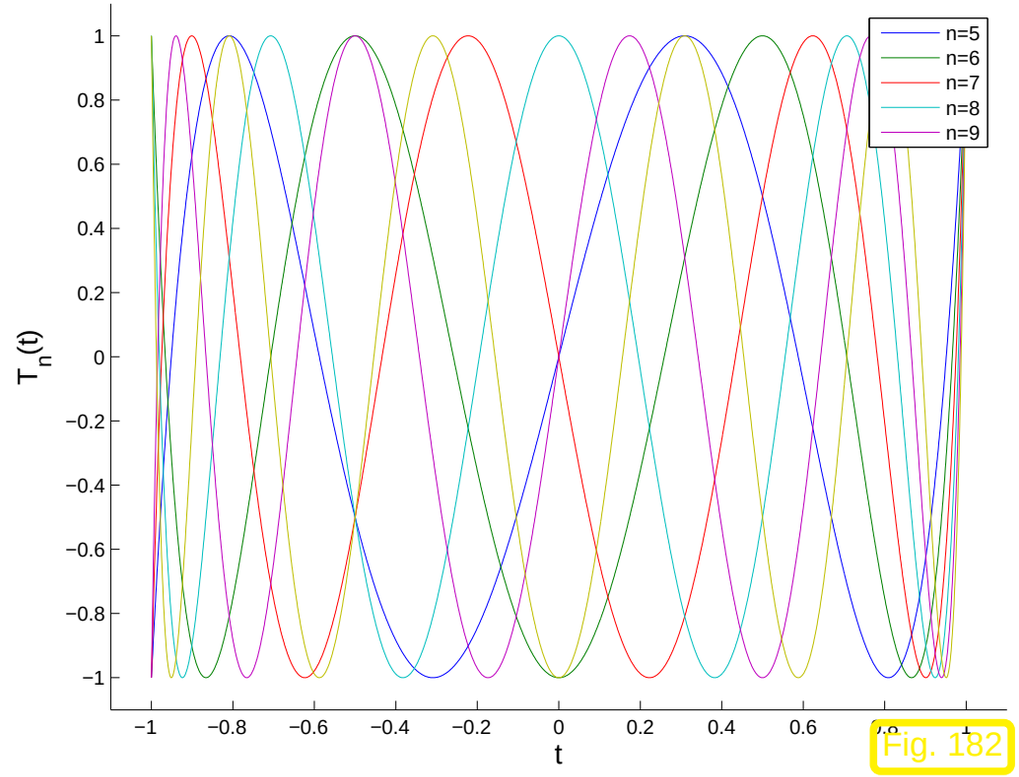
The nth *Chebychev polynomial is* $T_n(t) := \cos(n \arccos t)$, $\quad -1 \leq t \leq 1$.

R. Hiptmair

rev 38355, October 27, 2011



Chebychev polynomials $T_0, \dots, T_4$



Chebychev polynomials $T_5, \dots, T_9$

R. Hiptmair

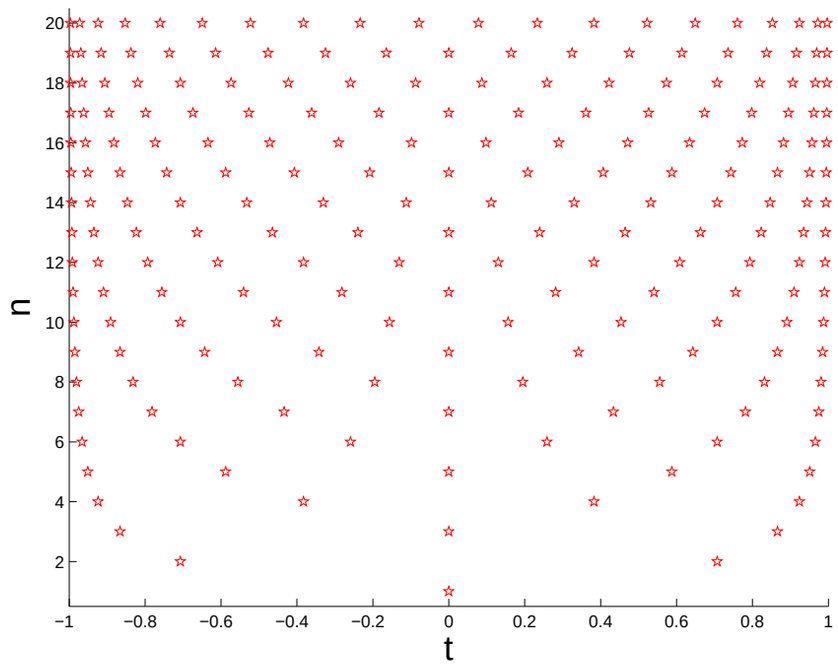
rev 38355, October 27, 2011

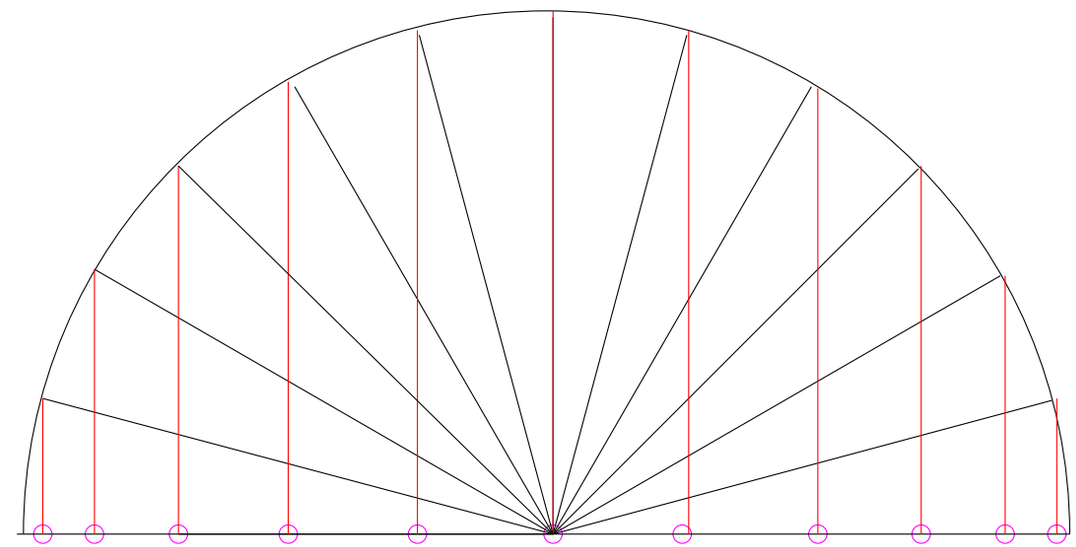
Theorem 9.2.7 (Minimax property of the Chebychev polynomials)**.** *[17, Section 7.1.4.], [35, Thm. 32.2]*

$$\|T_n\|_{L^\infty([-1,1])} = \inf\{\|p\|_{L^\infty([-1,1])} : p \in \mathcal{P}_n, p(t) = 2^{n-1}t^n + \cdots\} \;, \quad \forall n \in \mathbb{N} \;.$$

Zeros of T_n: $$t_k = \cos\left(\frac{2k-1}{2n}\pi\right) \;, \quad k = 1, \dots, n \;. \tag{9.2.8}$$

Location of Chebychev nodes t_k from (9.2.8):

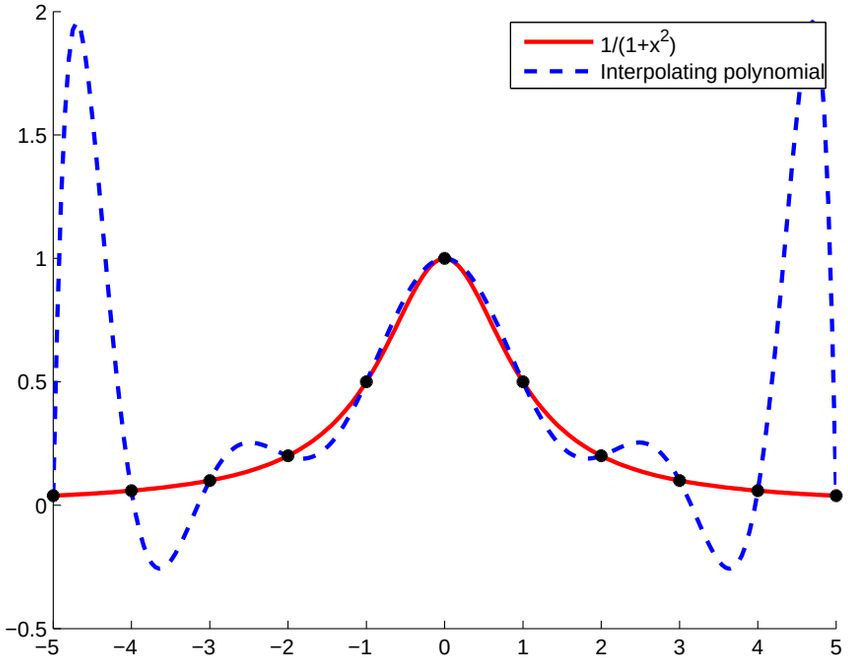


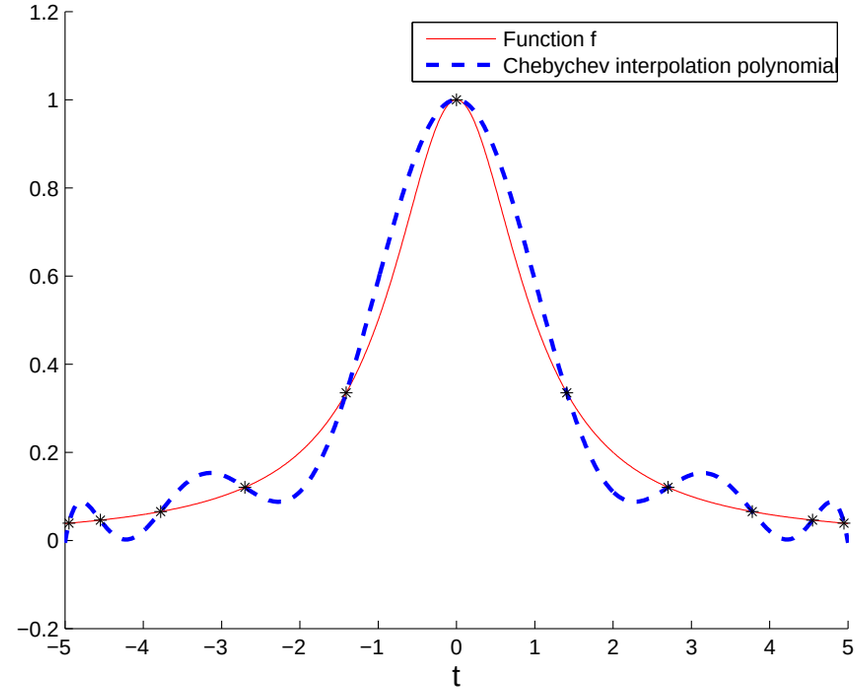


9.2.2 Chebychev interpolation error estimates

Example 9.2.12 (Polynomial interpolation: Chebychev 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:



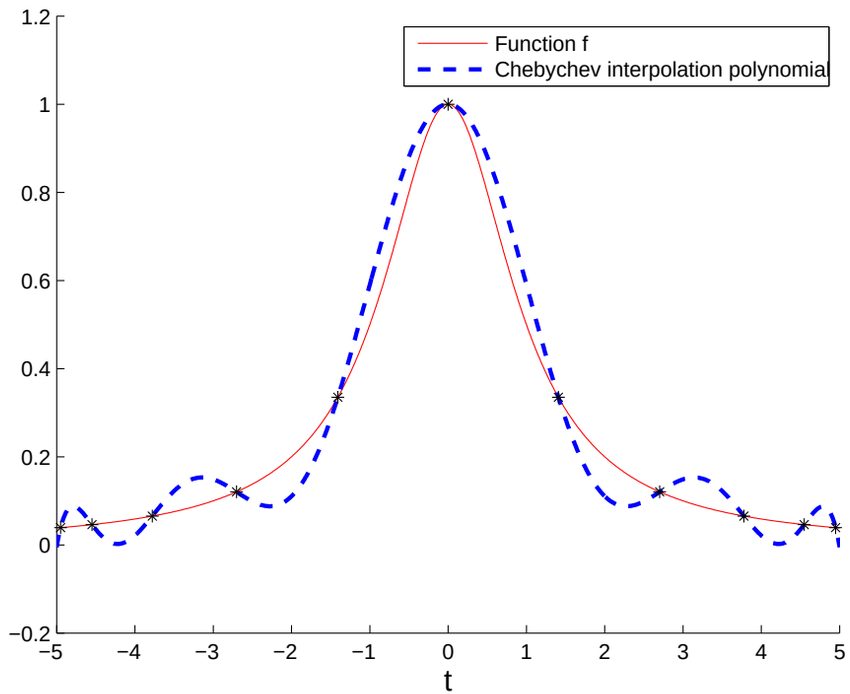


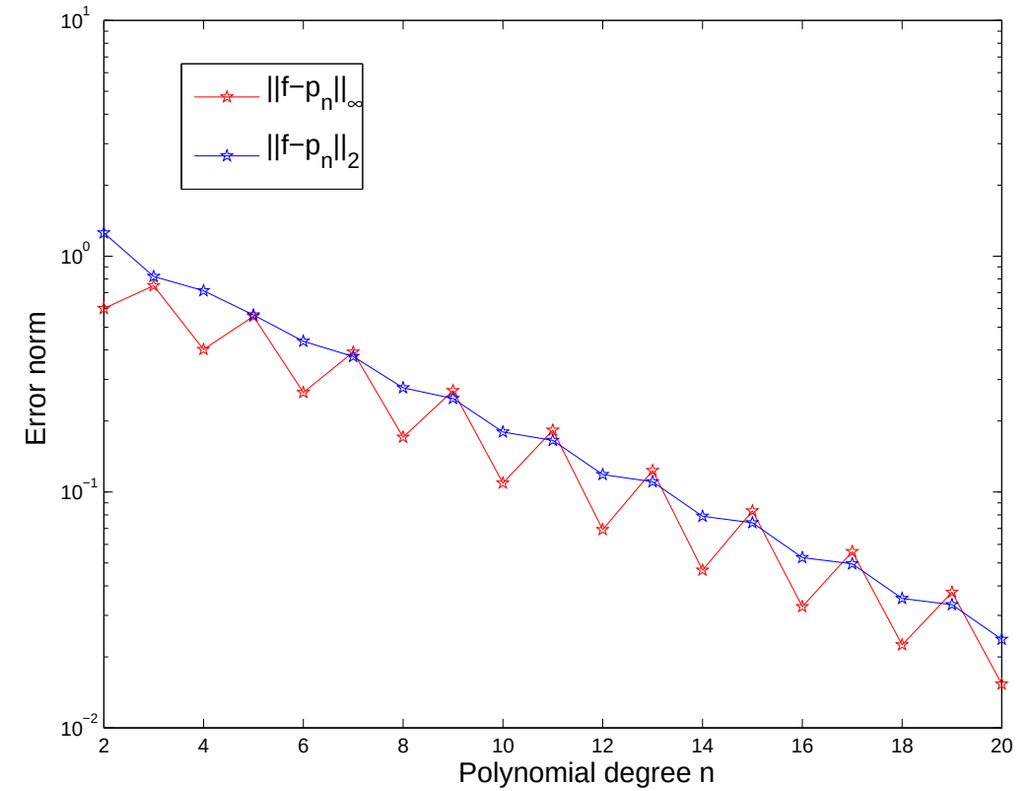
◇

Example 9.2.14 (Chebychev interpolation error).

① $f(t) = (1+t^2)^{-1}$, $I = [-5,5]$ (see Ex. 9.1.5)

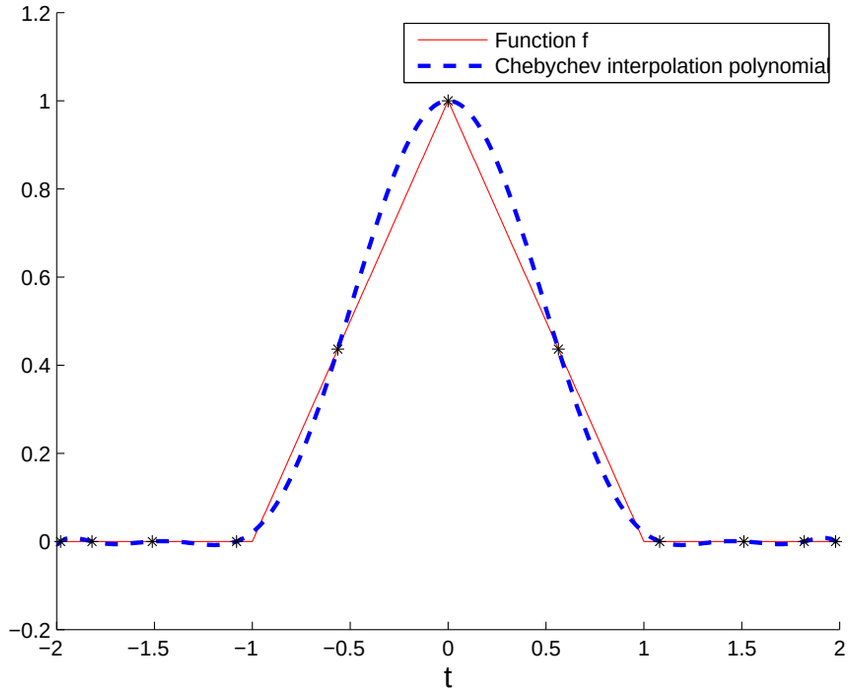
Interpolation with $n = 10$ Chebychev nodes (plot on the left).

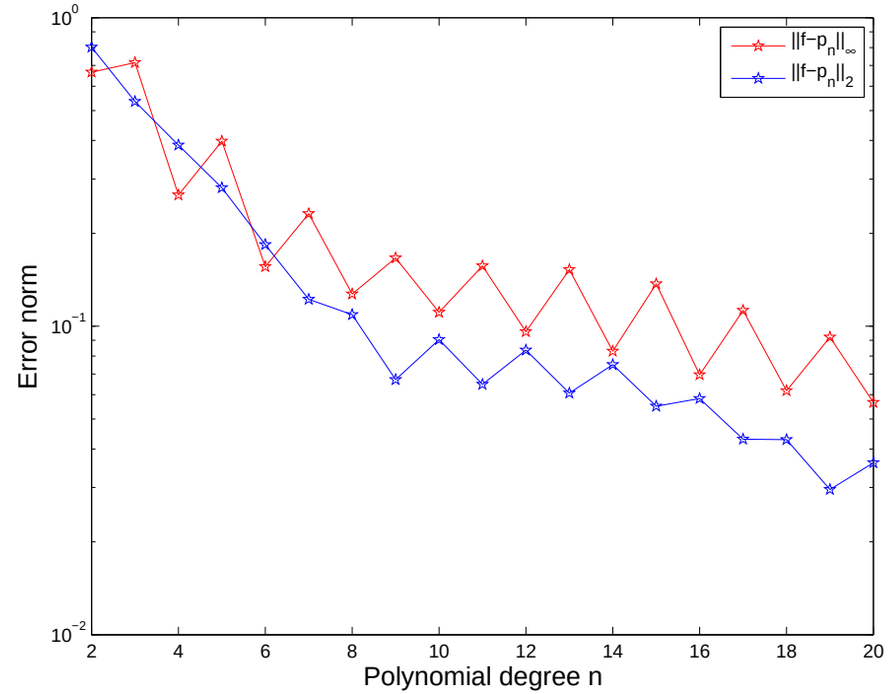


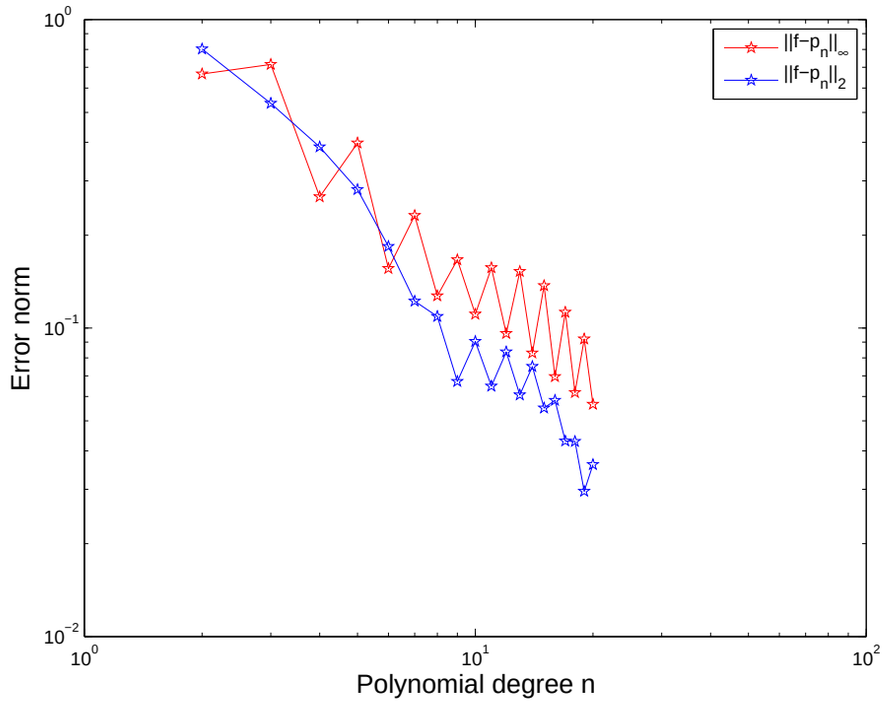


② $f(t) = \max\{1-|t|, 0\}$, $I = [-2,2]$, $n = 10$ nodes (plot on the left).

Now $f \in C^0(I)$ but $f \notin C^1(I)$.



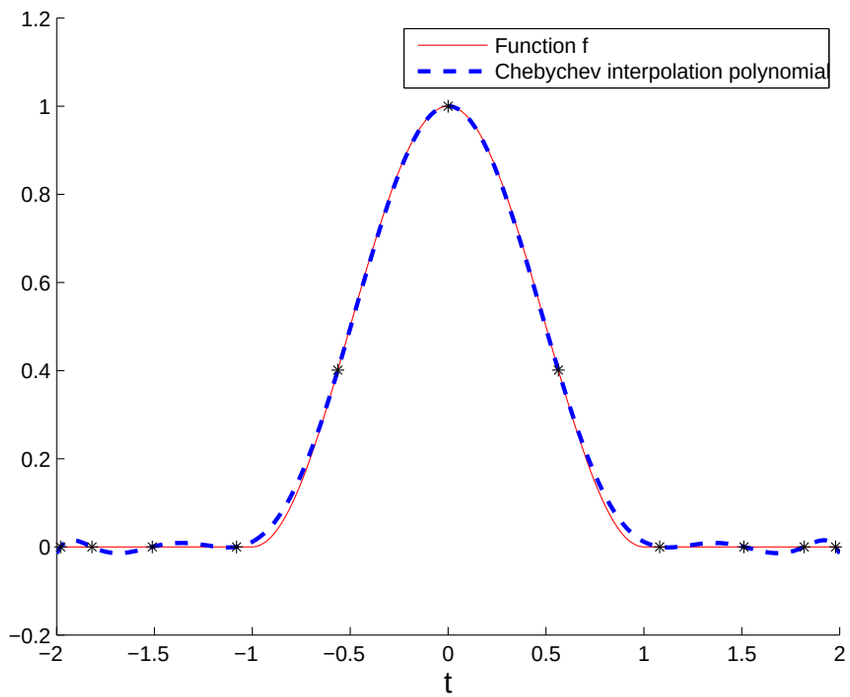


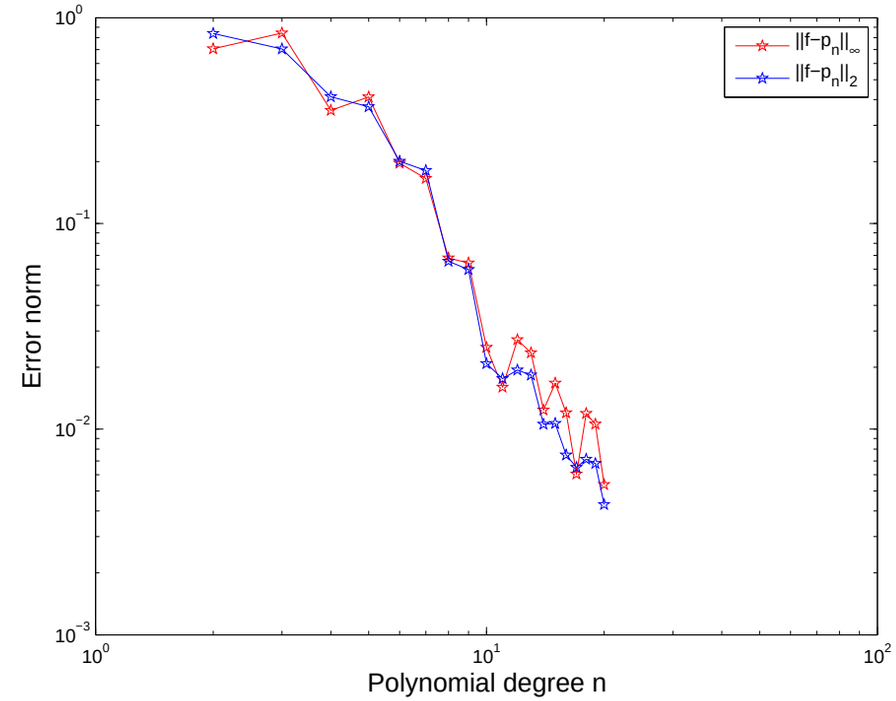


R. Hiptmair

rev 38355, October 27, 2011

③ $f(t) = \begin{cases} \frac{1}{2}(1+\cos \pi t) & |t| < 1 \\ 0 & 1 \leq |t| \leq 2 \end{cases}$ $\quad I = [-2,2], \quad n = 10$ (plot on the left).





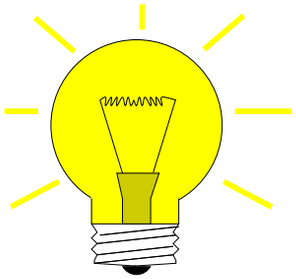
◇

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 $[-1,1]$

```
class ChebInterp {
  private:
  // various internal data describing Chebychev interpolating polynomial p
  public:
    // Constructor taking function f and degree n as arguments
    PolyInterp(const Function &f,unsigned int n);
    // Evaluation operator:  y_j = p(x_j),  j = 1,...,m  (m "large")
    double eval(const vector<double> &x,vector <double> &y) const;
};
```



Trick: expand p into Chebychev polynomials

$$p = \sum_{j=0}^{n} \alpha_j T_j \,, \quad \alpha_j \in \mathbb{R} \,. \tag{9.2.20}$$

Remark 9.2.21 (Fast evaluation of Chebychev expansion). $\rightarrow$ [35, Alg. 32.1]

Code 9.2.23: Clenshaw algorithm for evalation of Chebychev expansion (9.2.22)

```
function y = clenshaw(a,x)
% Clenshaw algorithm for evaluating p = Σ_{j=1}^{n+1} a_j T_{j-1} at points passed in vector x
n = length(a)-1; % degree of polynomial
d = repmat(reshape(a,n+1,1),1,length(x));
for j=n:-1:2
  d(j,:) = d(j,:) + 2*x.*d(j+1,:); % see (9.2.24)
  d(j-1,:) = d(j-1,:) - d(j+1,:);
end
y = d(1,:) + x.*d(2,:);
```

$\triangle$

Code 9.2.29: Efficient computation of Chebychev expansion coefficient of Chebychev interpolant

```
function a = chebexp(y)
% Efficiently compute coefficients α_j in the Chebychev expansion
% p = Σ_{j=0}^{n} α_j T_j of p ∈ P_n based on values y_k,
% k = 0,...,n, in Chebychev nodes t_k, k = 0,...,n.  These values are
```

```
% passed in the row vector y.
n = length(y)-1;    % degree of polynomial
% create vector z by wrapping and componentwise scaling
z = exp(-pi*i*n/(n+1)*(0:2*n+1)).*[y,y(end:-1:1)]; % r.h.s.  vector
c = ifft(z);  % Solve linear system (9.2.31) with effort O(n log n)
b = real(exp(0.5*pi*i/(n+1)*(-n:n+1)).*c); % recover β_j, see (9.2.31)
a = [b(n+1),2*b(n+2:2*n+1)]; % recover α_j, see (9.2.28)
```

9.3 Trigonometric interpolation [13, Sect. 8.5]

Trigonometric interpolation [13, pp. 304]

Given $t_0 < t_1 < \cdots < t_{2n}$, $t_k \in [0,1[$, and $y_k \in \mathbb{R}$, $k = 0,\dots,2n$ find

$$q \in \mathcal{P}_{2n}^T := \operatorname{Span}\{t \mapsto \cos(2\pi jt), t \mapsto \sin(2\pi jt)\}_{j=0}^n \ , \tag{9.3.1}$$

$$\text{with} \quad q(t_k) = y_k \quad \text{for all} \quad k = 0,\dots,2n \ . \tag{9.3.2}$$

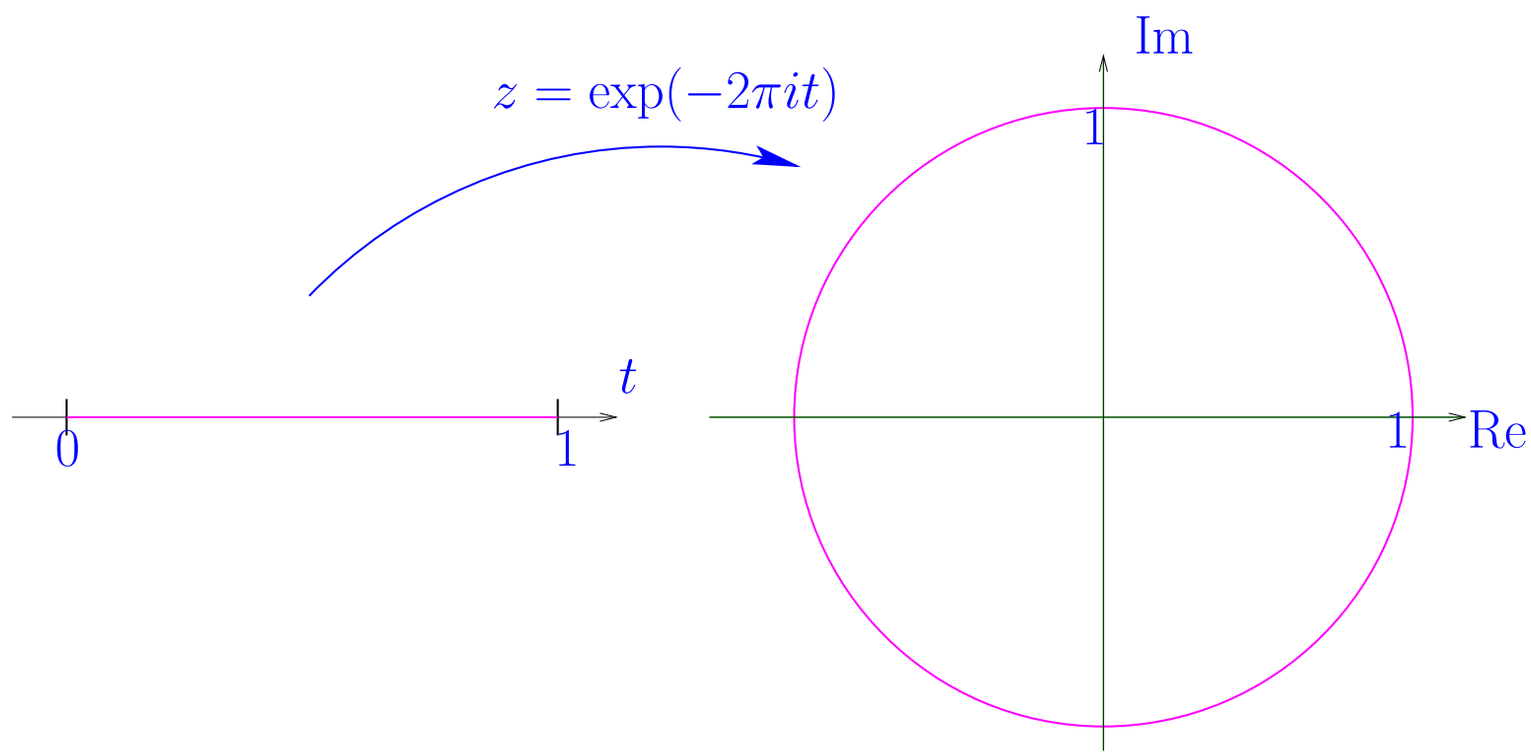
Terminology: $\mathcal{P}_{2n}^T \hat{=}$ space of trigonometric polynomials of degree $2n$.

a polynomial !

$$q \in \mathcal{P}_{2n+1}^T \quad \Rightarrow \quad q(t) = e^{-2\pi int} \cdot p(e^{2\pi it}) \quad \text{with} \quad p(z) = \sum_{j=0}^{2n} \gamma_j z^j \in \mathcal{P}_{2n} \,, \tag{9.3.6}$$

and γ_j from (9.3.5).

$t \to \exp(2\pi int)q(t)$ is a polynomial $p \in \mathcal{P}_{2n}$ restricted to the unit circle $\mathbb{S}^1$ in $\mathbb{C}$.



Trigonometric interpolation through data points (t_k, y_k)	$\Longleftrightarrow$	Polynomial interpolation through data points $(e^{2\pi i t_k}, y_k)$

Code 9.3.7: Evaluation of trigonometric interpolation polynomial in many points

```
function q = trigpolyval(t,y,x)
% Evaluation of trigonometric interpolant at numerous points
% t:  row vector of nodes t_0,...,t_n ∈ [0,1[
% y:  row vector of data y_0,...,y_n
% x:  row vector of evaluation points x_1,...,x_N
N = length(y); if (mod(N,2)~=1), error('#pts odd required'); end
n = (N-1)/2;
tc = exp(2*pi*i*t);      % Interpolation nodes on unit circle
z = exp(2*pi*i*n*t).*y; % Rescaled values, according to q(t) = e^{-2πint} · p(e^{2πit})
% Evaluation of interpolating polynomial on unit circle, see Code 3.4.1
p = intpolyval(tc,z,exp(2*pi*i*x));
q = exp(-2*pi*i*n*x).*p; % Undo the scaling, see (9.3.6)
```

► $(2n+1) \times (2n+1)$ linear system of equations:

$$\sum_{j=0}^{2n} \gamma_j \exp\left(2\pi i \frac{jk}{2n+1}\right) = z_k := \exp\left(2\pi i \frac{nk}{2n+1}\right) y_k \;, \quad k = 0, \dots, 2n \;.$$

$$\Updownarrow$$

$$\overline{\mathbf{F}}_{2n+1}\mathbf{c} = \mathbf{z} \;, \quad \mathbf{c} = (\gamma_0, \dots, \gamma_{2n})^T \quad \overset{\text{Lemma 8.2.10}}{\Rightarrow} \quad \mathbf{c} = \frac{1}{2n+1}\mathbf{F}_{2n}\mathbf{z} \;. \tag{9.3.8}$$

$(2n+1) \times (2n+1)$ (conjugate) Fourier matrix, see (8.2.9)

Code 9.3.10: Efficient computation of coefficient of trigonometric interpolation polynomial (*equidistant nodes*)

```
function [a,b] = trigipequid(y)
% Efficient computation of coefficients in expansion (9.3.3) for a trigonometric
% interpolation polynomial in equidistant points (j/(2n+1), y_j), j = 0,...,2n
% y has to be a row vector of odd length, return values are column vectors
N = length(y); if (mod(N,2)~=1), error('#pts odd!'); end;
n = (N-1)/2;
c = fft(exp(2*pi*i*(n/N)*(0:2*n)).*y)/N; % see (9.3.8)
% From (9.3.5): α_j = 1/2(γ_{n-j} + γ_{n+j}) and β_j = 1/(2i)(γ_{n-j} - γ_{n+j}), j = 1,...,n, α_0 = γ_n
a = transpose([c(n+1),c(n:-1:1)+c(n+2:N)]);
b = transpose(-i*[c(n:-1:1)-c(n+2:N)]);
```

Code 9.3.11: Computation of coefficients of trigonometric interpolation polynomial, general nodes

```
function [a,b] = trigpolycoeff(t,y)
```

```
% Computes expansion coefficients of trigonometric polyonomials (9.3.3)
% t:  row vector of nodes t_0,...,t_n ∈ [0,1[
% y:  row vector of data y_0,...,y_n
% return values are column vectors of expansion coefficients α_j, β_j
N = length(y); if (mod(N,2)~=1), error('#pts odd!'); end
n = (N-1)/2;
M = [cos(2*pi*t'*(0:n)), sin(2*pi*t'*(1:n))];
c = M\y';
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

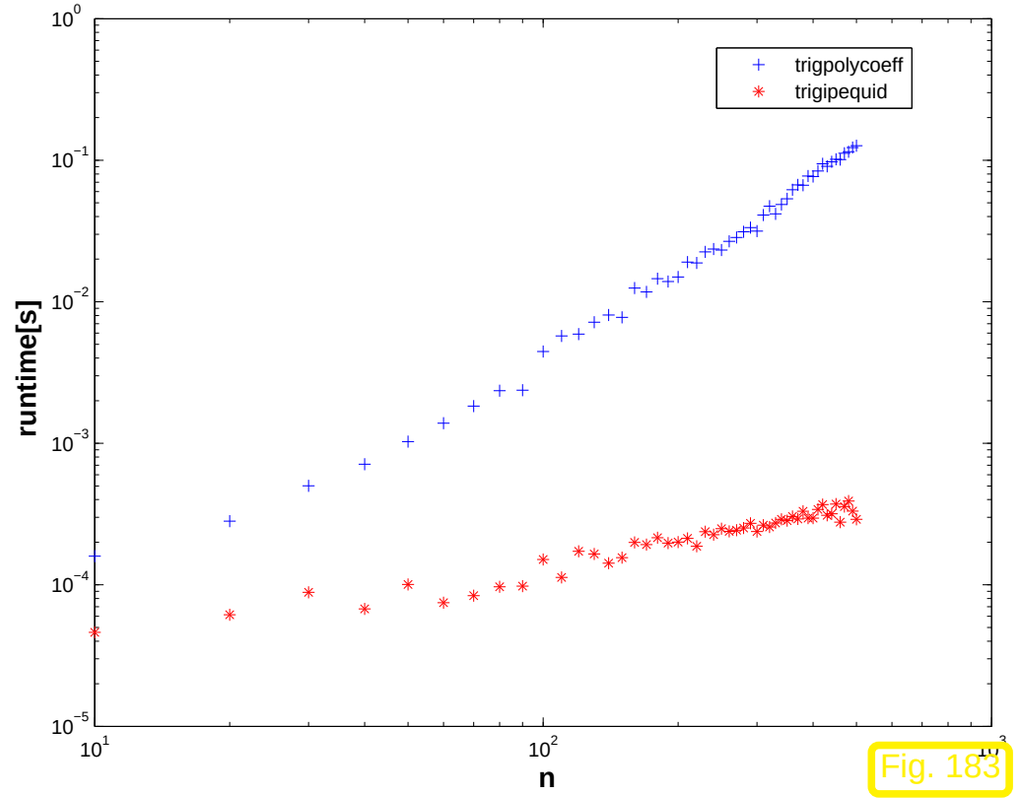


Fig. 183

◇

R. Hiptmair

rev 38355, October 27, 2011

Remark 9.3.14 (Efficient evaluation of trigonometric interpolation polynomials).

Code 9.3.16: Fast evaluation of trigonometric polynomial at *equidistant* points

```
function q = trigipequidcomp(a,b,N)
% Efficient evaluation of trigonometric polynomial at equidistant points
% column vectors a and b pass coefficients α_j, β_j in
% representation (9.3.3)
n = length(a)-1; if (N < (2*n-1)), error('N too small'); end;
gamma = transpose(0.5*[a(end:-1:2)+i*b(end:-1:1);...
```

```
                    2*a(1);a(2:end)-i*b(1:end)]);
ch = [gamma,zeros(1,N-(2*n+1))]; % zero padding
v = conj(fft(conj(ch))); % Multiplication with conjugate Fourier matrix
q = exp(-2*pi*i*n*(0:N-1)/N).*v; % undo rescaling
```

△

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^∞ periodic function: $f(t) = \dfrac{1}{\sqrt{1 + \frac{1}{2}\sin(2\pi t)}}$.

#3 “wedge function”: $f(t) = |t - \frac{1}{2}|$

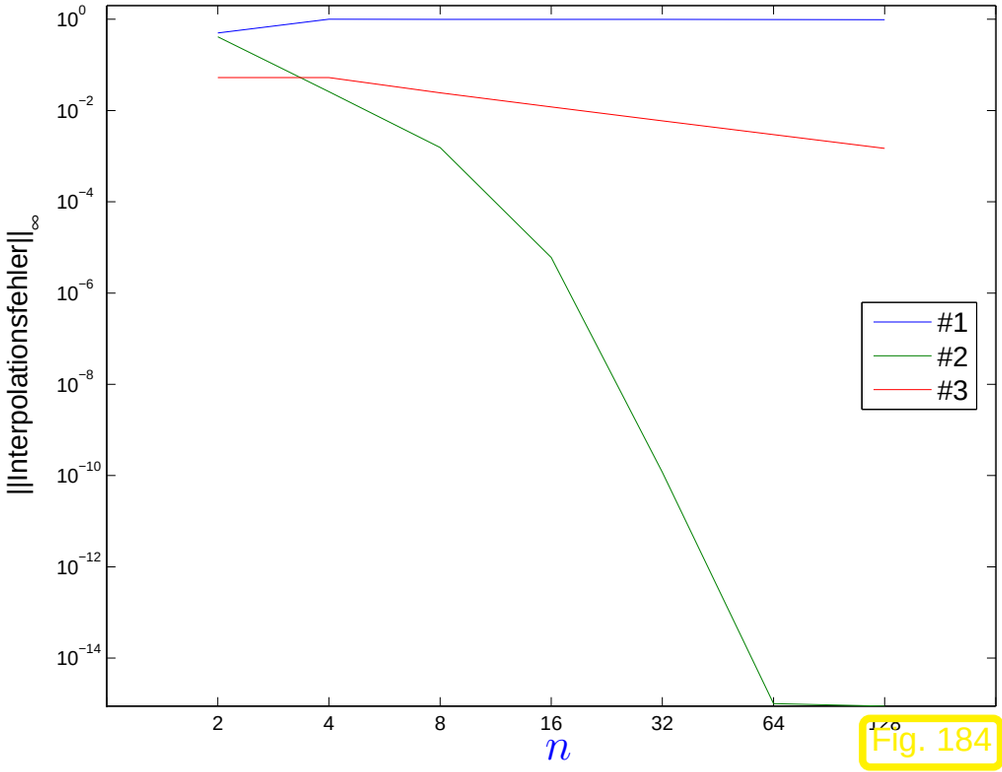


Fig. 184

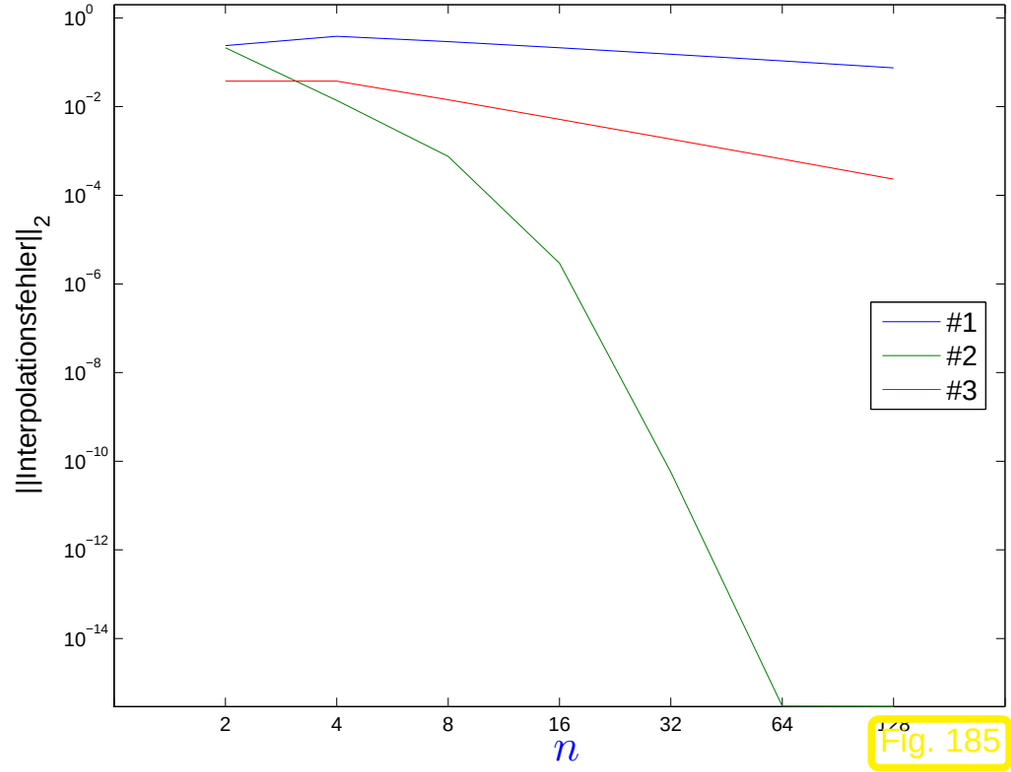
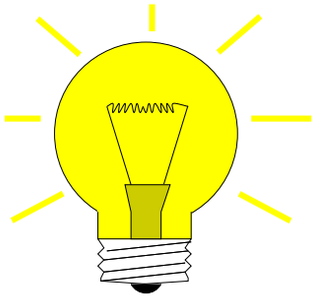


Fig. 185

◇

9.4 Approximation by piecewise polynomials



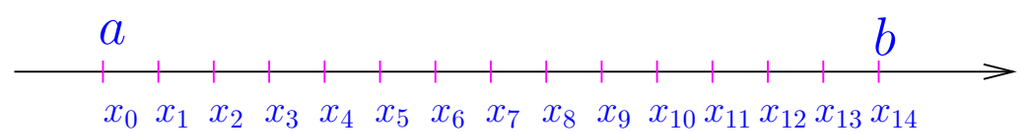
Idea: use piecewise polynomials with respect to a grid/mesh

$$\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_{m-1} < x_m = b\} \tag{9.4.1}$$

to approximate function $f : [a, b] \mapsto \mathbb{R}$, $a < b$.

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

Focus: asymptotic behavior of (some norm of) interpolation error

$$\|f - \mathsf{I}_{\mathcal{M}} f\| \leq CT(N) \quad \text{for} \quad N \to \infty \,, \tag{9.4.4}$$

where $N := \sum_{j=1}^{m} (n_j + 1)$.

Example 9.4.5 (h-convergence of piecewise polynomial interpolation).

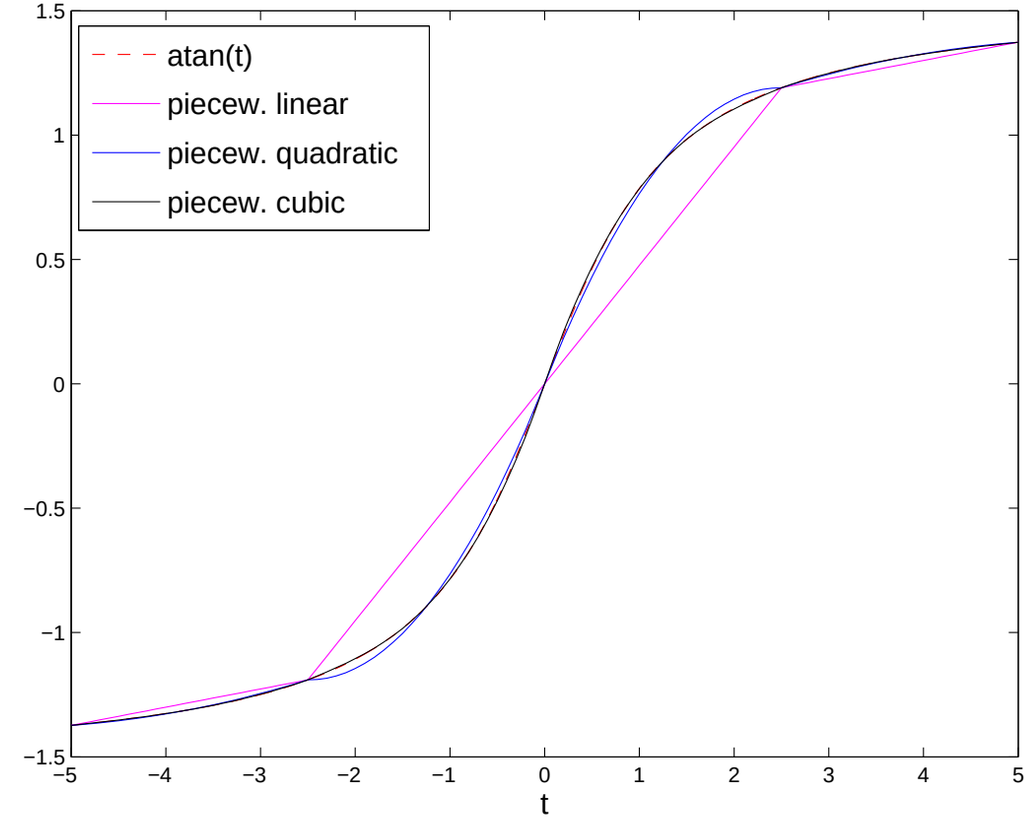
Compare Ex. 3.6.5:

$f(t) = \arctan t$, $I = [-5, 5]$

Grid $\mathcal{M} := \{-5, -\frac{5}{2}, 0, \frac{5}{2}, 5\}$

$\mathcal{T}^j$ equidistant in I_j.

Plots of the piecewise linear, quadratic and cubic polynomial interpolants $\rightarrow$



R. Hiptmair

rev 38355, December 8, 2010

- *Sequence* of (equidistant) meshes: $\mathcal{M}_i := \{-5 + j\,2^{-i}10\}_{j=0}^{2^i}$, $i = 1, \ldots, 6$.
- *Equidistant* local interpolation nodes (endpoints of grid intervals included).

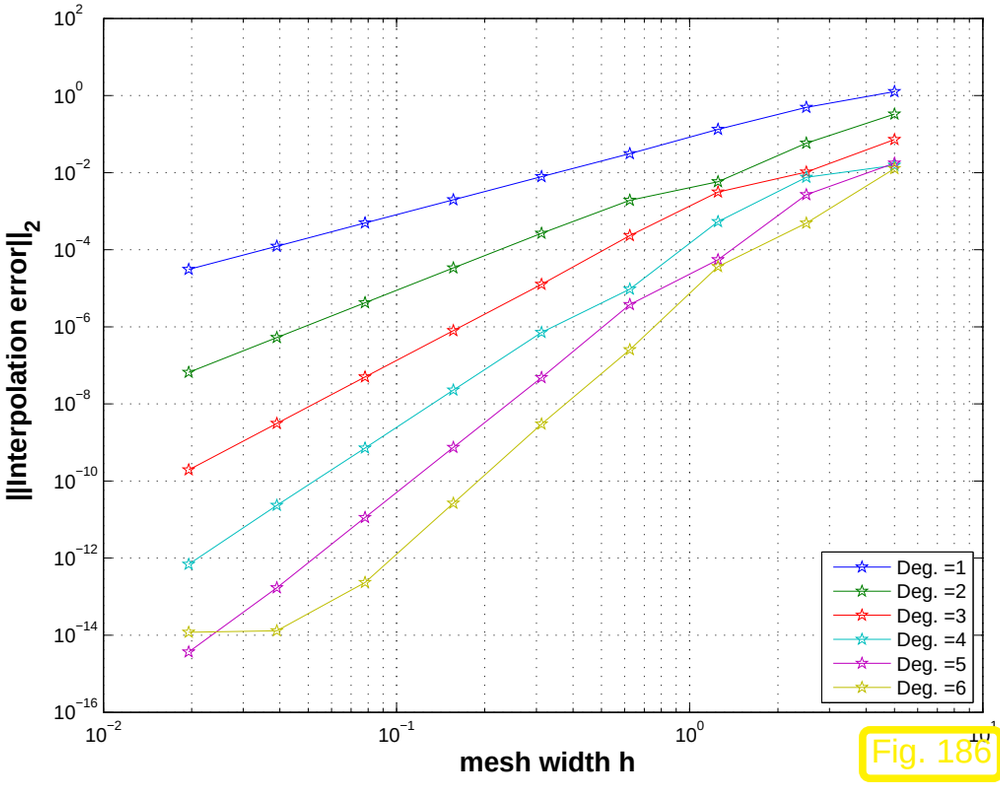


Fig. 186

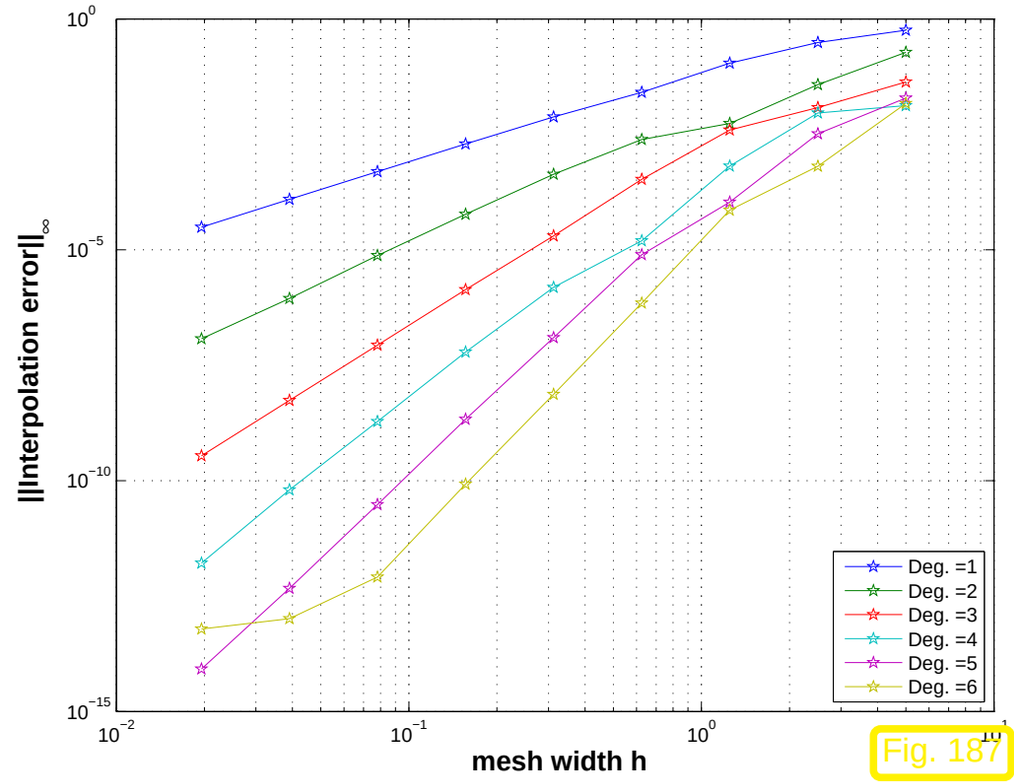


Fig. 187

R. Hiptmair

rev 38355, December 8, 2010

Rates α of algebraic convergence $O(h_{\mathcal{M}}^{\alpha})$ of norms of interpolation error:

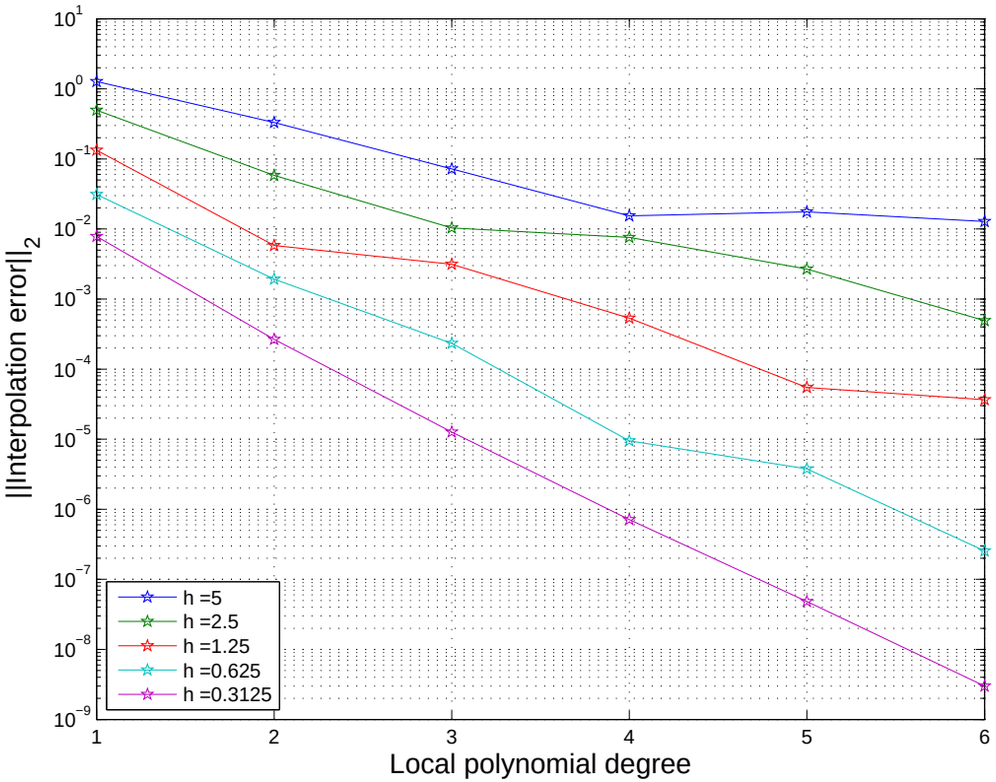
n	1	2	3	4	5	6
w.r.t. L^2-norm	1.9957	2.9747	4.0256	4.8070	6.0013	5.2012
w.r.t. L^∞-norm	1.9529	2.8989	3.9712	4.7057	5.9801	4.9228

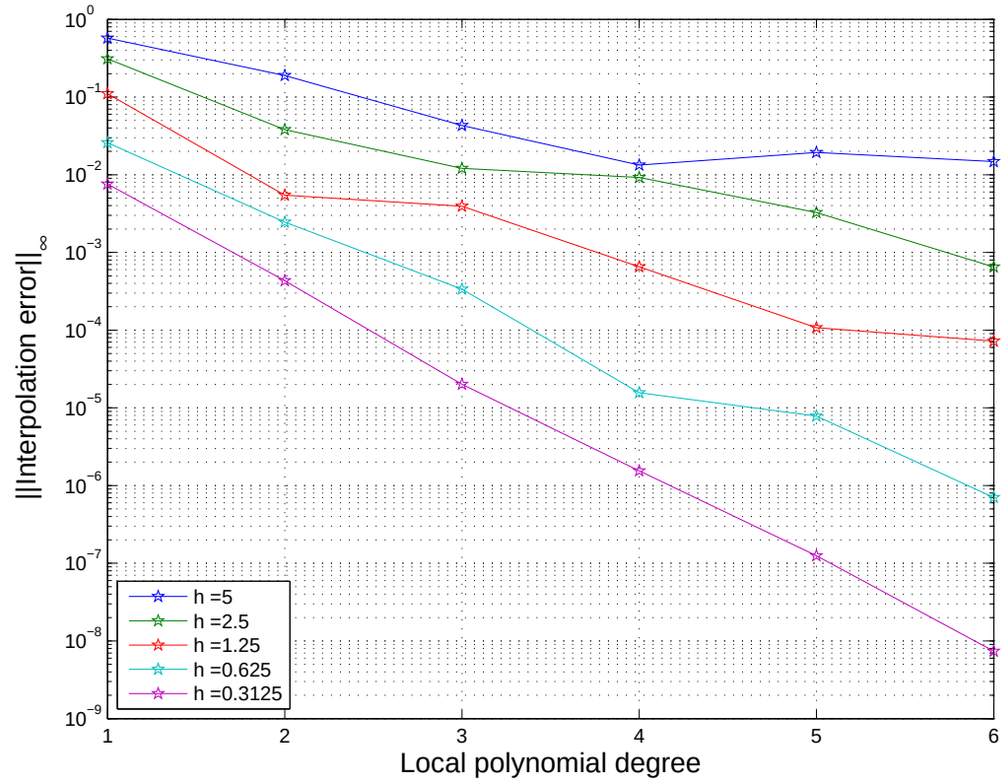
$\diamond$

$$(9.1.12) \quad \Rightarrow \quad \|f - s\|_{L^\infty([x_0,x_m])} \leq \frac{h^{n+1}}{(n+1)!} \left\| f^{(n+1)} \right\|_{L^\infty([x_0,x_m])} \quad , \qquad (9.4.6)$$

with mesh width $h := \max\{|x_j - x_{j-1}| : j = 1, \ldots, m\}$.

Example 9.4.7 (p-convergence of piecewise polynomial interpolation).





◇

9.4.2 Cubic Hermite iterpolation: error estimates

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 $O(h^4)$

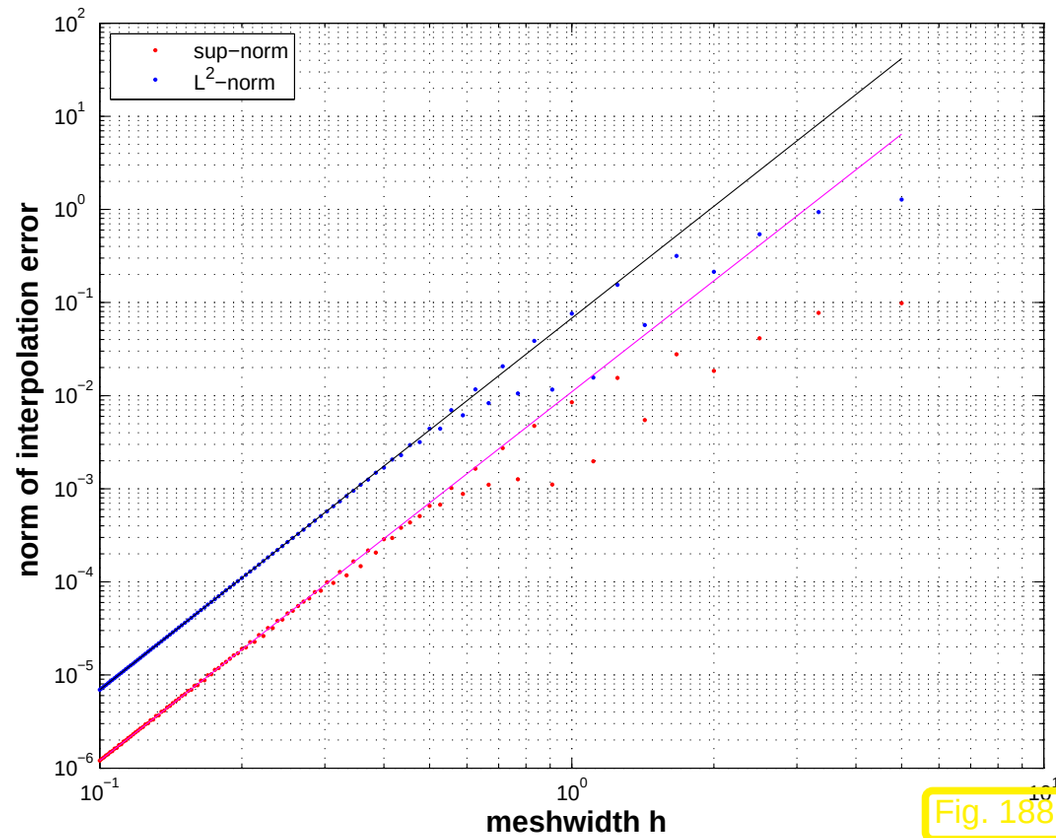


Fig. 188

◇

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)

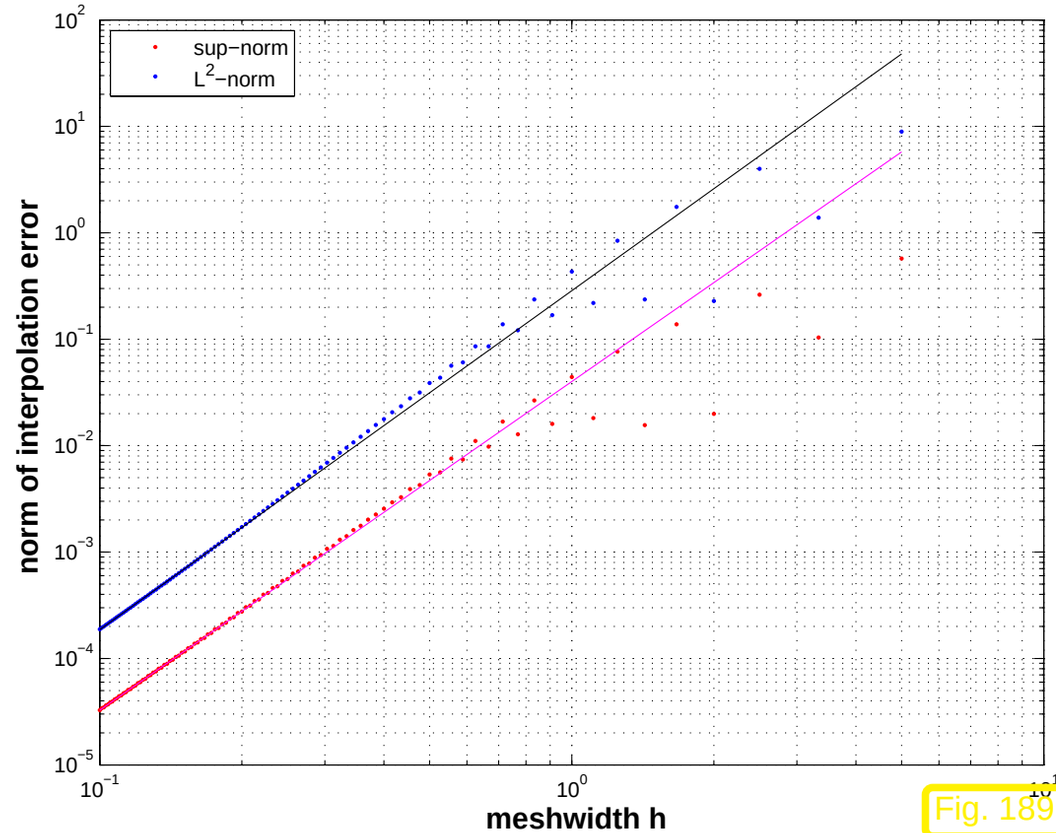


Fig. 189

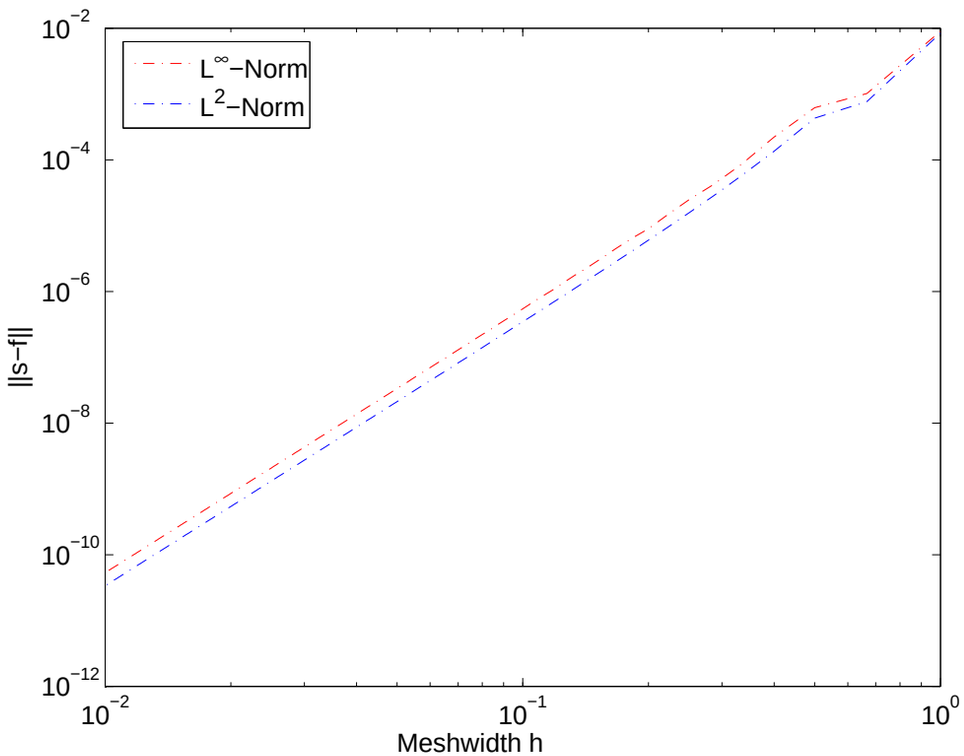
◇

9.4.3 Cubic spline interpolation: error estimates [35, Ch. 47]

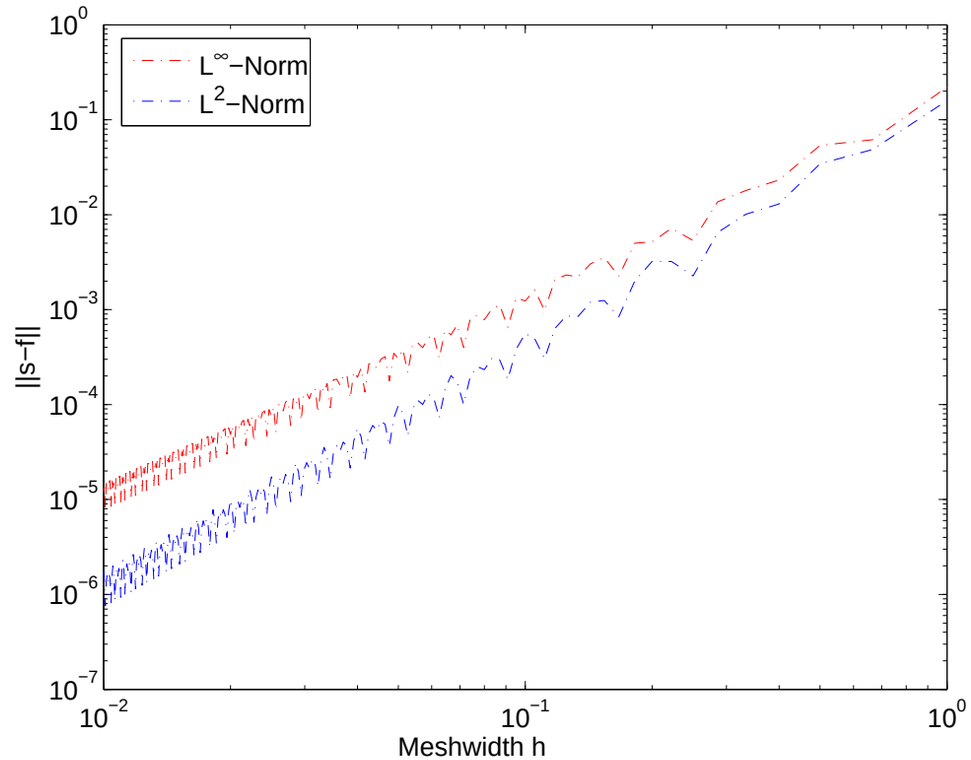
Example 9.4.12 (Approximation by complete cubic spline interpolants).

Grid (knot set) $\mathcal{M} := \{-1 + \frac{2}{n}j\}_{j=0}^{n}$, $n \in \mathbb{N}$ ➔ meshwidth $h = 2/n$, $I = [-1,1]$

$$f_1(t) = \frac{1}{1+e^{-2t}} \in C^\infty(I) \quad , \quad f_2(t) = \begin{cases} 0 & \text{, if } t < -\frac{2}{5} \text{ ,} \\ \frac{1}{2}(1+\cos(\pi(t-\frac{3}{5}))) & \text{, if } -\frac{2}{5} < t < \frac{3}{5} \text{ ,} \\ 1 & \text{otherwise.} \end{cases} \in C^1(I) \text{ .}$$



$$\|f_1 - s\|_{L^\infty([-1,1])} = O(h^4)$$



$$\|f_2 - s\|_{L^\infty([-1,1])} = O(h^2)$$

Algebraic order of convergence in h = min { 1 + regularity of f, 4 }.

Theory [34]: $f \in C^4([t_0,t_n])$ ➔ $\|f - s\|_{L^\infty([t_0,t_n])} \leq \frac{5}{384}h^4 \left\| f^{(4)} \right\|_{L^\infty([t_0,t_n])}$

R. Hiptmair

rev 38355, December 8, 2010

10 Numerical Quadrature [35, VII], [13, Ch. 10]

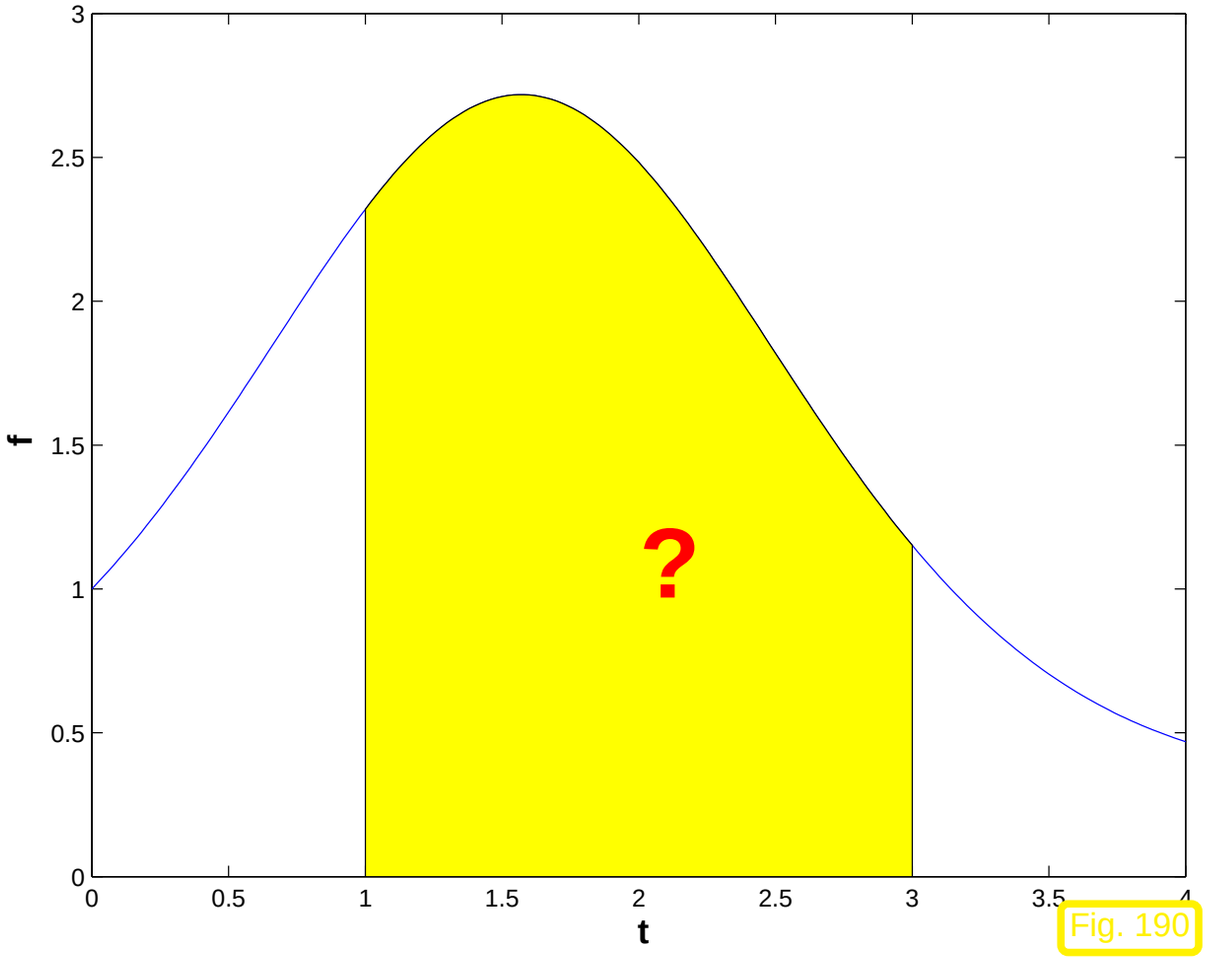


Fig. 190

Numerical quadrature methods

approximate

$$\int_a^b f(t)\,\mathrm{d}t$$

R. Hiptmair

rev 38355, December 9, 2010

Example 10.0.2 (Heating production in electrical circuits).

Time-harmonic excitation:

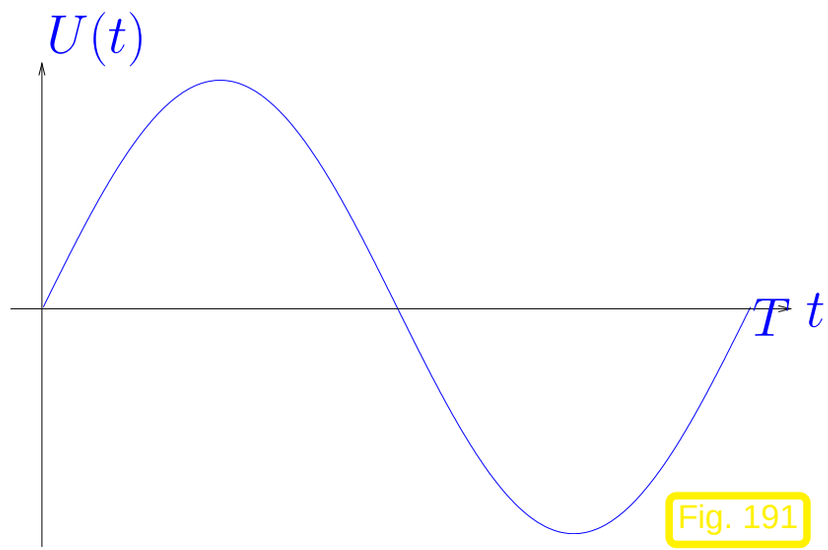


Fig. 191

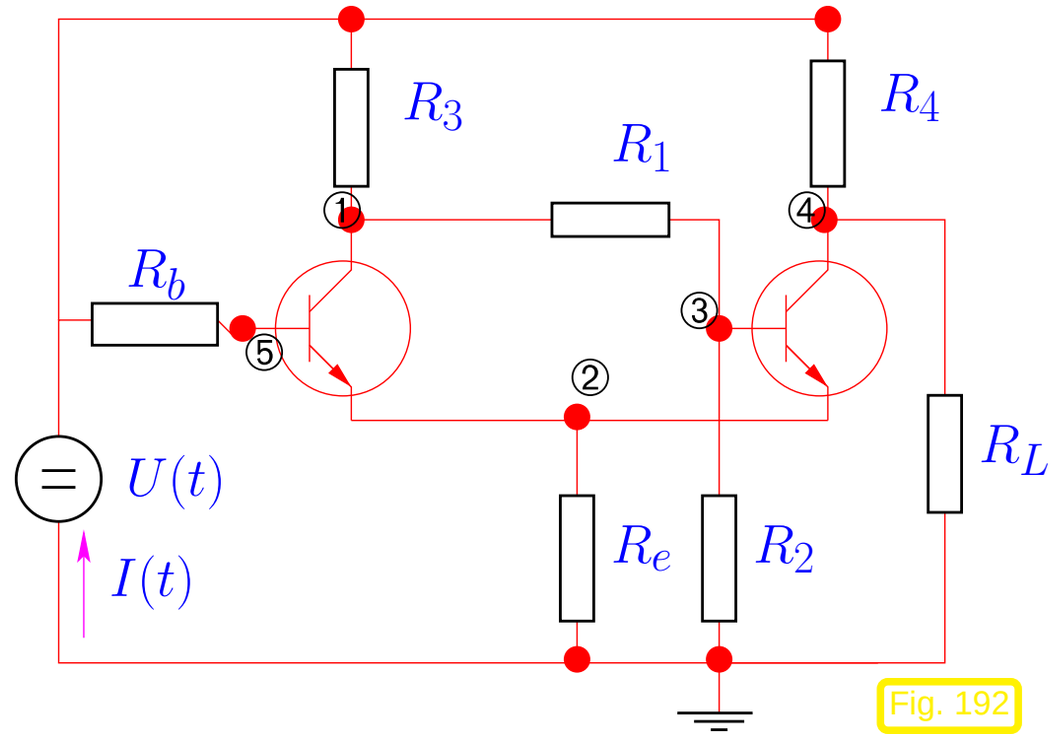


Fig. 192

Integrating power $P = UI$ over period $[0, T]$ yields heat production per period:

$$W_{\text{therm}} = \int_0^T U(t)I(t)\,\mathrm{d}t\,, \quad \text{where} \quad I = I(U)\,.$$

`function I = current(U)` involves solving non-linear system of equations, see Ex. 4.0.1!

◇

R. Hiptmair

rev 38355, December 9, 2010

10.1 Quadrature Formulas

n-point quadrature formula on $[a,b]$: (n-point quadrature rule)

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \sum_{j=1}^{n} w_j^n\, f(c_j^n)\,. \qquad (10.1.1)$$

w_j^n	:	quadrature weights	$\in \mathbb{R}$	(*ger.:* Quadraturgewichte)
c_j^n	:	quadrature nodes	$\in [a,b]$	(*ger.:* Quadraturknoten)

Remark 10.1.3 (Transformation of quadrature rules).

△

R. Hiptmair

rev 38355, December 9, 2010

Our focus (*cf.* interpolation error estimates, Sect. 9.1):

given families of quadrature rules $\{Q_n\}_n$ described by

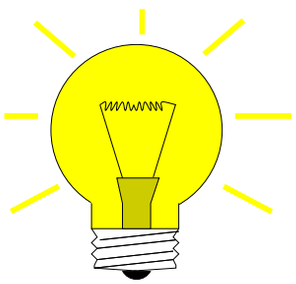
- quadrature weights $\left\{w_j^n,\ j=1,\dots,n\right\}_{n\in\mathbb{N}}$ and
- quadrature nodes $\left\{c_j^n,\ j=1,\dots,n\right\}_{n\in\mathbb{N}}$, we

study the *asymptotic* behavior of the quadrature error $E(n)$ for $n \to \infty$

Qualitative distinction, see (9.1.3):

- ▷ algebraic convergence $E(n) = O(n^{-p})$, $p > 0$
- ▷ exponential convergence $E(n) = O(q^n)$, $0 \leq q < 1$

10.2 Polynomial Quadrature Formulas [13, Sect. 10.2]



Idea: replace integrand f with $p_{n-1} \in \mathcal{P}_{n-1}$ = polynomial interpolant of f for given interpolation nodes $\{t_0, \dots, t_{n-1}\} \subset [a,b]$

$$\blacktriangleright \quad \int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \int_a^b p_{n-1}(t)\,\mathrm{d}t \,. \tag{10.2.3}$$

Lagrange polynomials: $L_i(t) := \prod_{\substack{j=0\\ j\neq i}}^{n-1} \frac{t-t_j}{t_i-t_j}$, $i = 0, \dots, n-1$ $\overset{(3.3.4)}{\blacktriangleright}$ $p_{n-1}(t) = \sum_{i=0}^{n-1} f(t_i) L_i(t)$.

$$\int_a^b p_{n-1}(t)\,\mathrm{d}t = \sum_{i=0}^{n-1} f(t_i) \int_a^b L_i(t)\,\mathrm{d}t \quad \blacktriangleright \quad \begin{array}{ll} \text{nodes} & c_i = t_{i-1}\,, \\ \text{weights} & w_i := \int_a^b L_{i-1}(t)\,\mathrm{d}t\,. \end{array} \tag{10.2.4}$$

Example 10.2.5 (Midpoint rule).

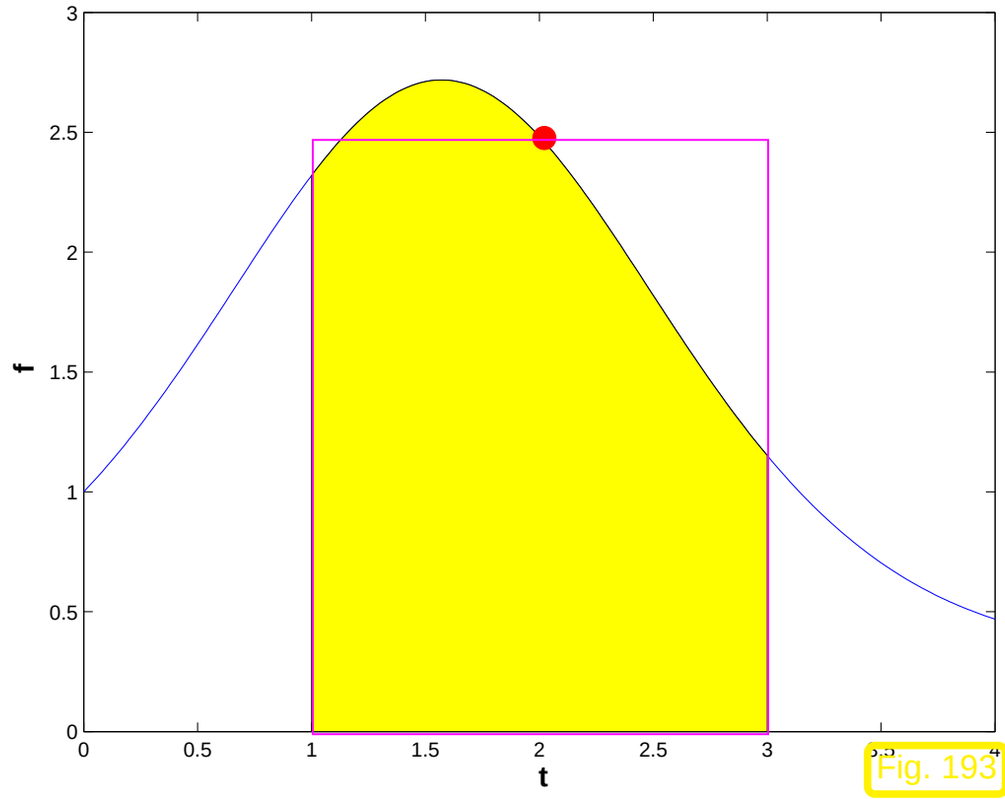


Fig. 193

1-point quadrature formula:

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_{\mathrm{mp}}(f) = (b-a)f(\tfrac{1}{2}(a+b))\ .$$

"midpoint"

◇

Example 10.2.6 (Newton-Cotes formulas). [35, Ch. 38]

Equidistant quadrature nodes $t_j := a + hj, \quad h := \dfrac{b-a}{n}, \quad j = 0,\dots,n$:

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);
```

$$\widehat{Q}_{\mathrm{trp}}(f) := \frac{1}{2}\left(f(0) + f(1)\right) \qquad (10.2.7)$$

$$\left(\int_a^b f(t)\,\mathrm{d}t \approx \frac{b-a}{2}(f(a) + f(b)) \right)$$

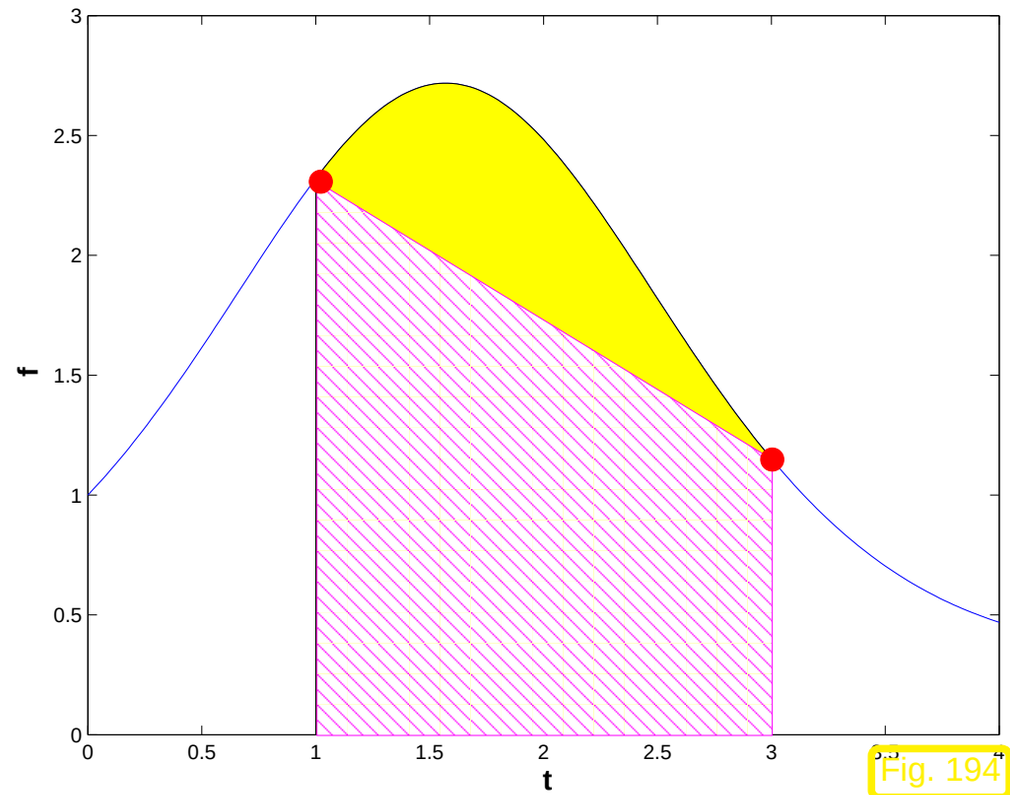


Fig. 194

- $n = 2$: Simpson rule

```
> simpson := newtoncotes(2);
```

R. Hiptmair

rev 38355, October 31, 2011

$$\frac{h}{6}\left(f(0)+4\,f(\tfrac{1}{2})+f(1)\right) \quad \left(\int_a^b f(t)\,\mathrm{d}t \approx \frac{b-a}{6}\left(f(a)+4\,f\left(\frac{a+b}{2}\right)+f(b)\right) \right) \qquad \text{(10.2.8)}$$

- $n \geq 8$: quadrature formulas with *negative* weights

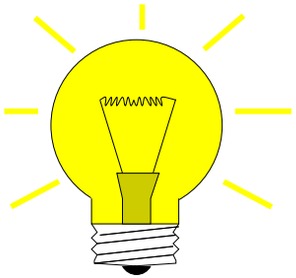
```
> newtoncotes(8);
```

$$\frac{1}{28350}\,h\,(989\,f(0)+5888\,f(\tfrac{1}{8})-928\,f(\tfrac{1}{4})+10496\,f(\tfrac{3}{8})$$
$$-4540\,f(\tfrac{1}{2})+10496\,f(\tfrac{5}{8})-928\,f(\tfrac{3}{4})+5888\,f(\tfrac{7}{8})+989\,f(1))$$

◇

Negative weights compromise numerical stability (→ Def. 2.5.11) **!**

10.3 Composite Quadrature



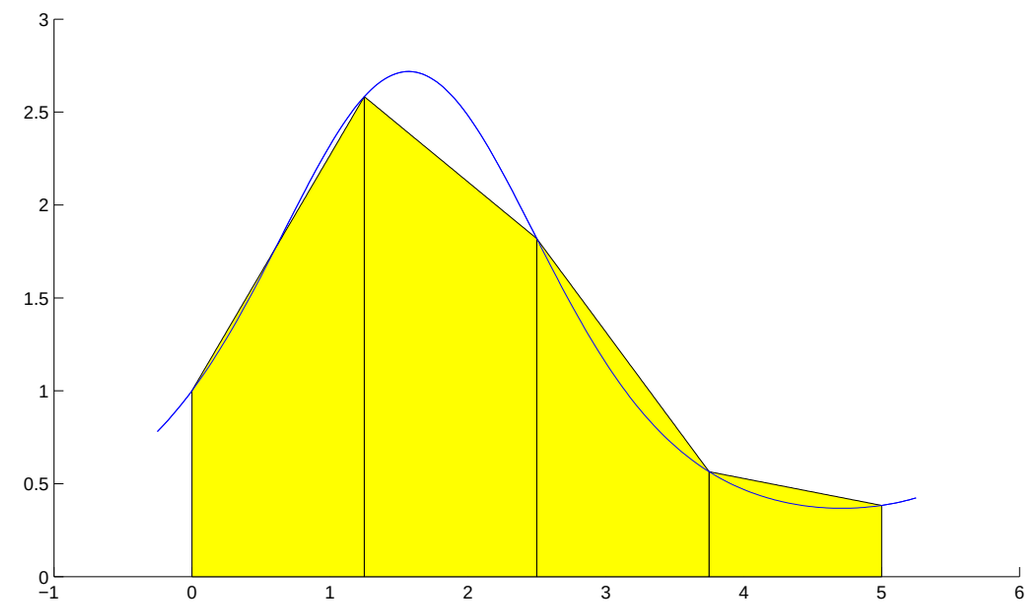
Idea:
- Partition integration domain $[a,b]$ by mesh (grid, $\rightarrow$ Sect.9.4)
$$\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_m = b\}$$
- Apply quadrature formulas from Sects. 10.2, 10.4 locally on mesh intervals $I_j := [x_{j-1}, x_j]$, $j = 1, \ldots, m$, and sum up.

composite quadrature rule

Example 10.3.2 (Simple composite polynomial quadrature rules).

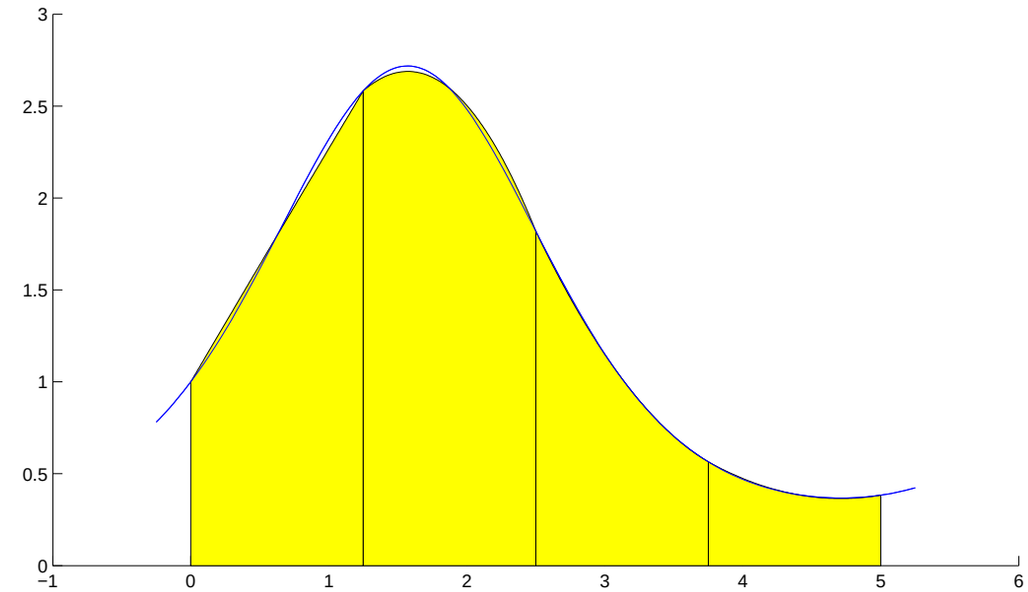
Composite trapezoidal rule, *cf.* (10.2.7)

$$\int_a^b f(t)\mathrm{d}t = \frac{1}{2}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \frac{1}{2}(x_{j+1} - x_{j-1})f(x_j) + \frac{1}{2}(x_m - x_{m-1})f(b) \; . \tag{10.3.3}$$



Composite Simpson rule, *cf.* (10.2.8)

$$\int_a^b f(t)\mathrm{d}t = \frac{1}{6}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \frac{1}{6}(x_{j+1} - x_{j-1})f(x_j) + \sum_{j=1}^{m} \frac{2}{3}(x_j - x_{j-1})f(\tfrac{1}{2}(x_j + x_{j-1})) + \frac{1}{6}(x_m - x_{m-1})f(b) \ . \tag{10.3.4}$$



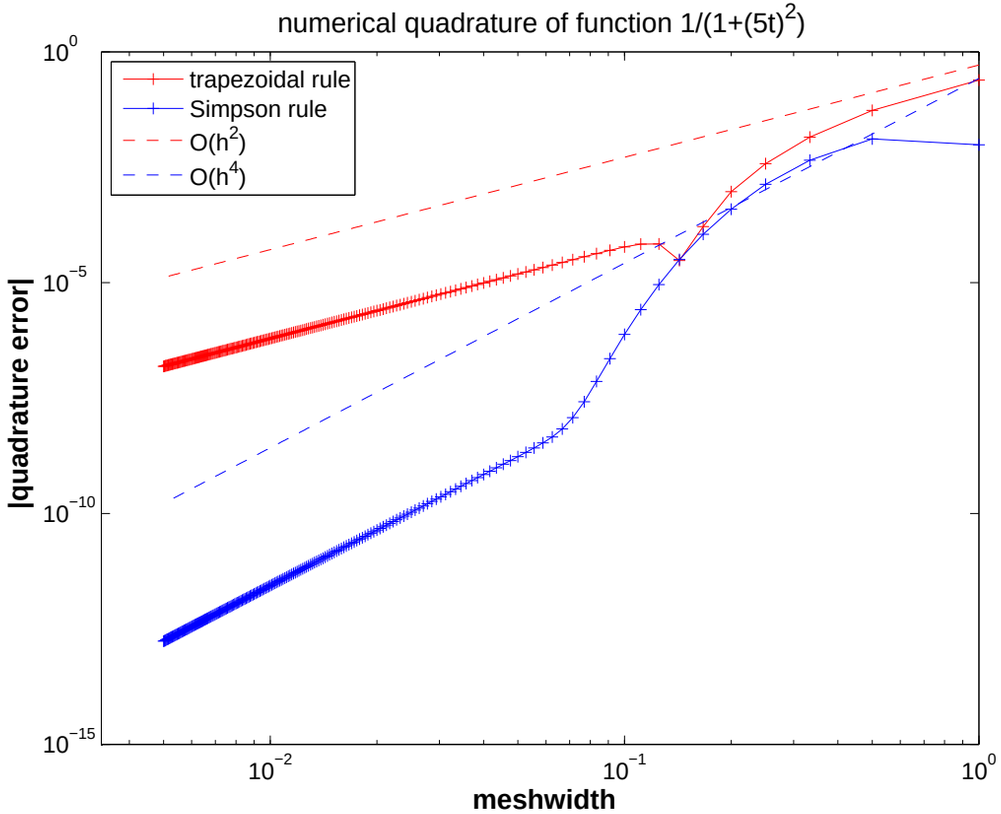
◇

Example 10.3.5 (Quadrature errors for composite quadrature rules).

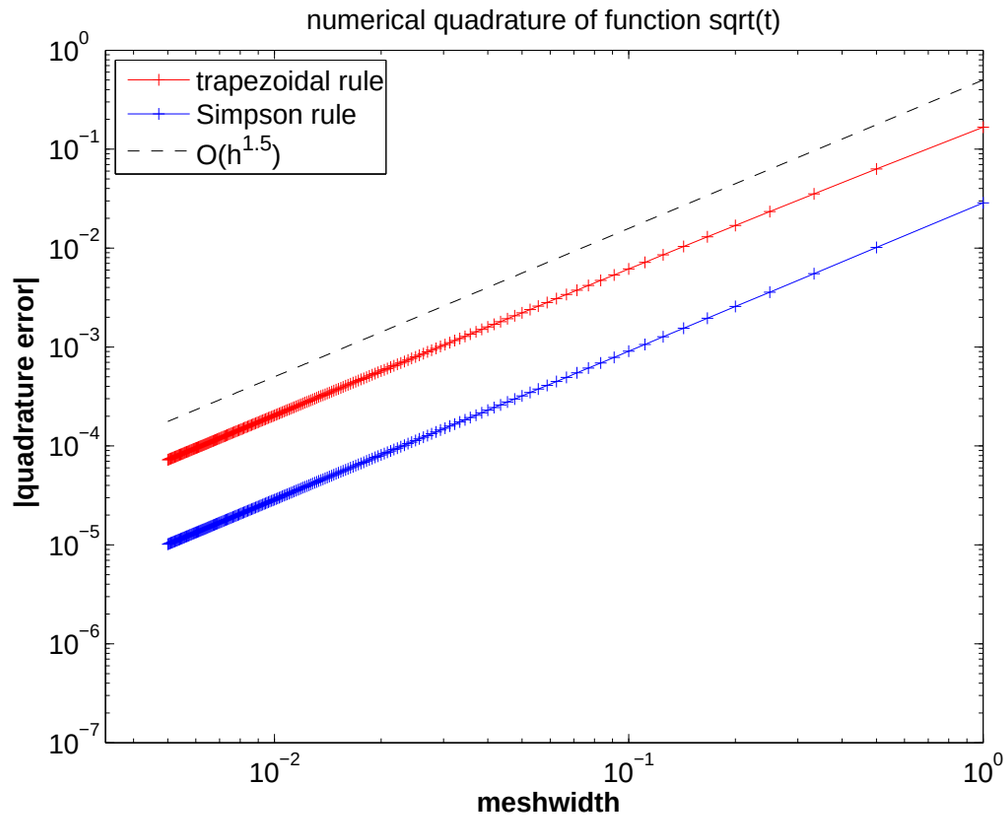
Composite quadrature rules based on

- trapezoidal rule (10.2.7) ➢ local order 2 (exact for linear functions),
- Simpson rule (10.2.8) ➢ local order 3 (exact for quadratic polynomials)

on equidistant mesh $\mathcal{M} := \{jh\}_{j=0}^{n}$, $h = 1/n$, $n \in \mathbb{N}$.



quadrature error, $f_1(t) := \frac{1}{1+(5t)^2}$ on $[0,1]$



quadrature error, $f_2(t) := \sqrt{t}$ on $[0,1]$

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 **!**

◇

Gauge for "quality" of a quadrature formula Q_n:

$$\text{Order}(Q_n) := \max\{n \in \mathbb{N}_0 : \quad Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \quad \forall p \in \mathcal{P}_n\} + 1$$

Focus: *asymptotic* behavior of quadrature error for

mesh width $\quad h := \max_{j=1,\dots,m} |x_j - x_{j-1}| \to 0$

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

$$\exists C > 0: \quad \left| \int_I f(t)\,\mathrm{d}t - Q(f) \right| \leq C h^p \left\| f^{(p)} \right\|_{L^\infty(I)} \quad \forall f \in C^p(I), \forall \mathcal{M} \ .$$

Remark 10.3.11 (Removing a singularity by transformation).

△

Example 10.3.13 (Convergence of equidistant trapezoidal rule).

Equidistant trapezoidal rule (order 2), see (10.3.3)

$$\int_a^b f(t)\,\mathrm{d}t \approx T_m(f) := h\left(\tfrac{1}{2}f(a) + \sum_{k=1}^{m-1} f(kh) + \tfrac{1}{2}f(b)\right)\,, \quad h := \frac{b-a}{m}\,. \tag{10.3.14}$$

1-periodic smooth (analytic) integrand

$$f(t) = \frac{1}{\sqrt{1 - a\sin(2\pi t - 1)}}\,, \quad 0 < a < 1\,.$$

R. Hiptmair

rev 38355, November 8, 2011

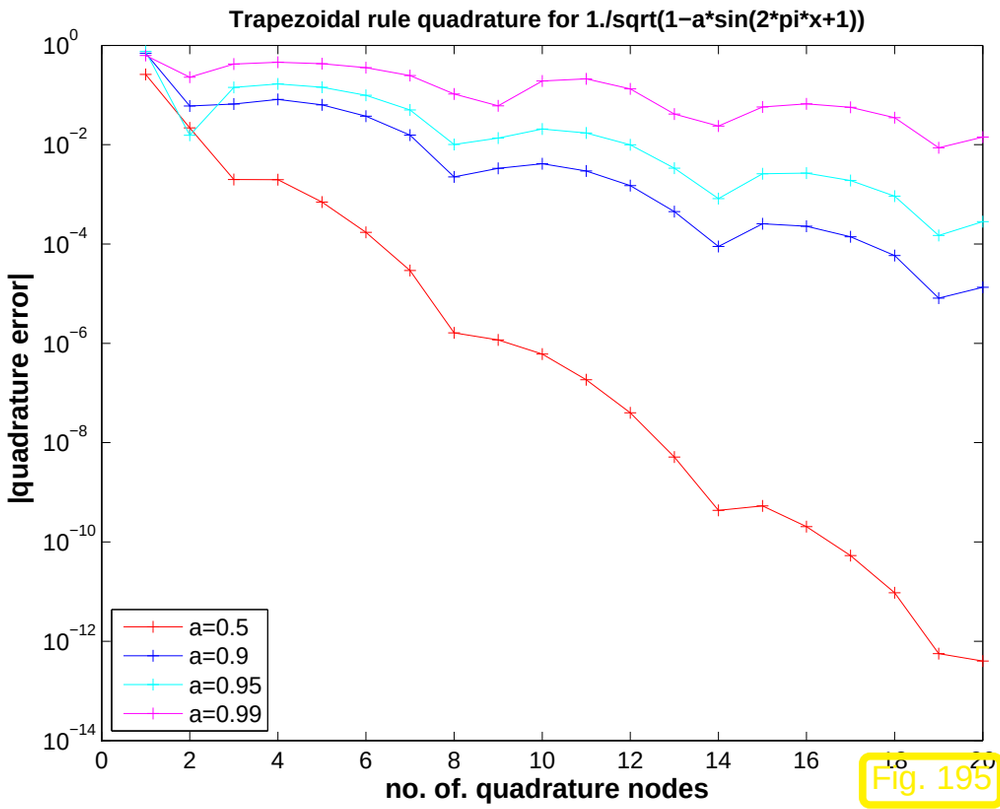


Fig. 195

quadrature error for $T_n(f)$ on $[0,1]$

exponential convergence !!

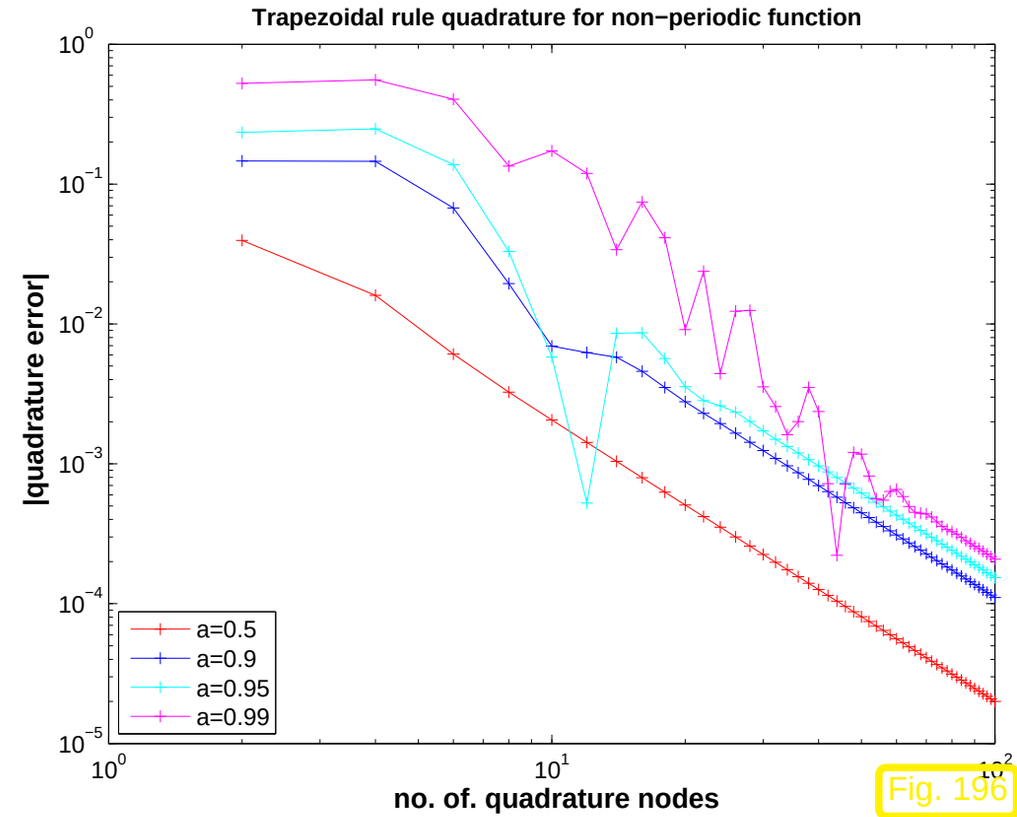


Fig. 196

quadrature error for $T_n(f)$ on $[0,\frac{1}{2}]$

merely algebraic convergence

◇

Remark 10.3.15 (Approximate computation of Fourier coefficients).

Code 10.3.16: DFT-based approximate computation of Fourier coefficients

```
function y = fourcoeffcomp(c,m,ovsmpl)
% Compute the Fourier coefficients y_{-m},...,y_m of the function
```

```
% c : [0,1[ ↦ ℂ using an oversampling factor ovsmpl.
% c must be a handle to a function @(t), which accepts row
% vector arguments
if (nargin < 3), ovsmpl = 2; else ovsmpl = ceil(ovsmpl); end
N = (2*m+1)*ovsmpl; h = 1/N; % Number of quadrature points
% (Inverse) discrete Fourier transform
y = ifft(c(0:h:1-h));
% Undo oversampling and wrapping of Fourier coefficient array
y = [y(N-m+1:N),y(1:m+1)];
```

△

Remark 10.3.17 (Choice of (local) quadrature weights).

△

Lemma 10.3.20 (Bound for order of quadrature formula)**.**

There is no n-point quadrature formula of order $2n+1$

10.4 Gauss Quadrature [35, Ch. 40-41], [13, Sect.10.3]

Example 10.4.1 (2-point quadrature rule of order 4).

Necessary & sufficient conditions for order 4 , *cf.* (10.3.21):

$$Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \;\forall p \in \mathcal{P}_3 \quad \Leftrightarrow \quad Q_n(t^q) = \frac{1}{q+1}(b^{q+1} - a^{q+1}) \;, \quad q = 0,1,2,3 \;.$$

4 equations for weights ω_j and nodes ξ_j, $j = 1,2$ ($a = -1, b = 1$), *cf.* Rem. 10.3.19

$$\begin{aligned} \int_{-1}^{1} 1\,\mathrm{d}t = 2 = 1\omega_1 + 1\omega_2 \quad &, \quad \int_{-1}^{1} t\,\mathrm{d}t = 0 = \xi_1\omega_1 + \xi_2\omega_2 \\ \int_{-1}^{1} t^2\,\mathrm{d}t = \frac{2}{3} = \xi_1^2\omega_1 + \xi_2^2\omega_2 \quad &, \quad \int_{-1}^{1} t^3\,\mathrm{d}t = 0 = \xi_1^3\omega_1 + \xi_2^3\omega_2 \;. \end{aligned} \tag{10.4.2}$$

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);
```

R. Hiptmair

rev 38355, December 13, 2010

➢ weights & nodes: $\left\{\omega_2 = 1, \omega_1 = 1, \xi_1 = 1/3\sqrt{3}, \xi_2 = -1/3\sqrt{3}\right\}$

▶ quadrature formula: $$\int_{-1}^{1} f(x)\,dx \approx f\left(\frac{1}{\sqrt{3}}\right) + f\left(-\frac{1}{\sqrt{3}}\right) \tag{10.4.3}$$

◇

Theorem 10.4.8 (Existence of n-point quadrature formulas of order $2n$)**.**
Let $\left\{\bar{P}_n\right\}_{n\in\mathbb{N}_0}$ be a family of non-zero polynomials that satisfies

- $\bar{P}_n \in \mathcal{P}_n$,
- $\int_{-1}^{1} q(t)\bar{P}_n(t)\,\mathrm{d}t = 0$ *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 $[-1,1]$.

Definition 10.4.12 (Legendre polynomials)**.**
The n-th Legendre polynomial P_n is defined by

- $P_n \in \mathcal{P}_n$,
- $\int_{-1}^{1} P_n(t)q(t)\,\mathrm{d}t = 0 \;\forall q \in \mathcal{P}_{n-1}$,
- $P_n(1) = 1$.

Legendre polynomials $P_0, \dots, P_5$ ➢

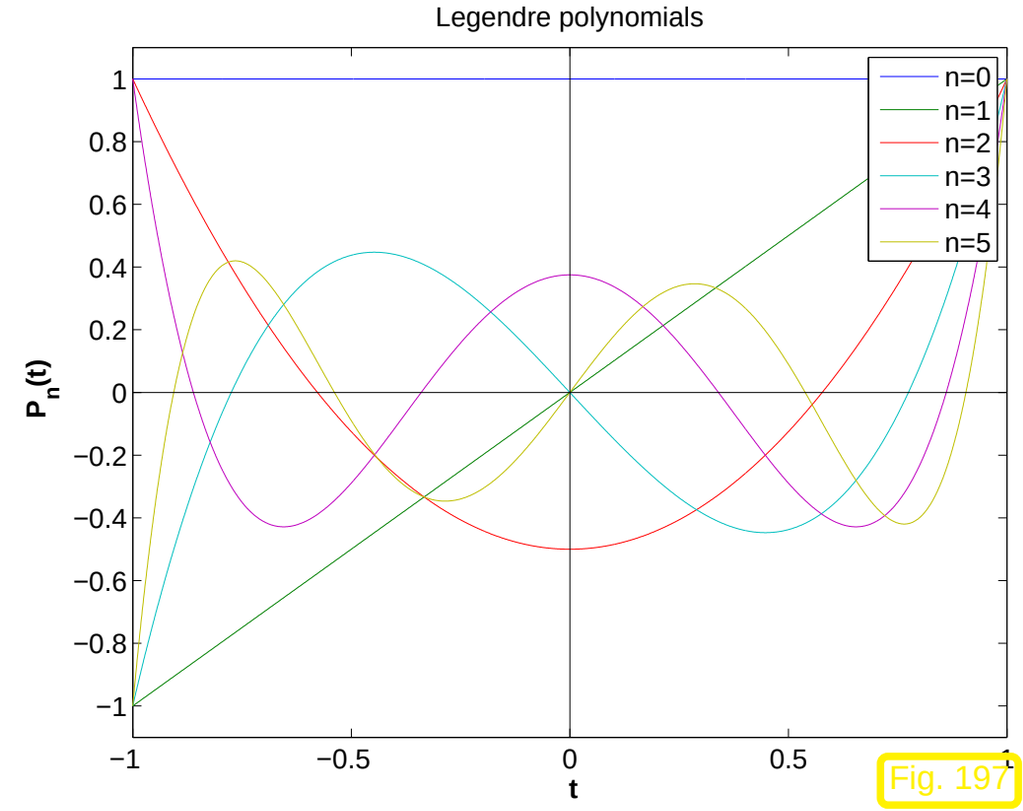


Fig. 197

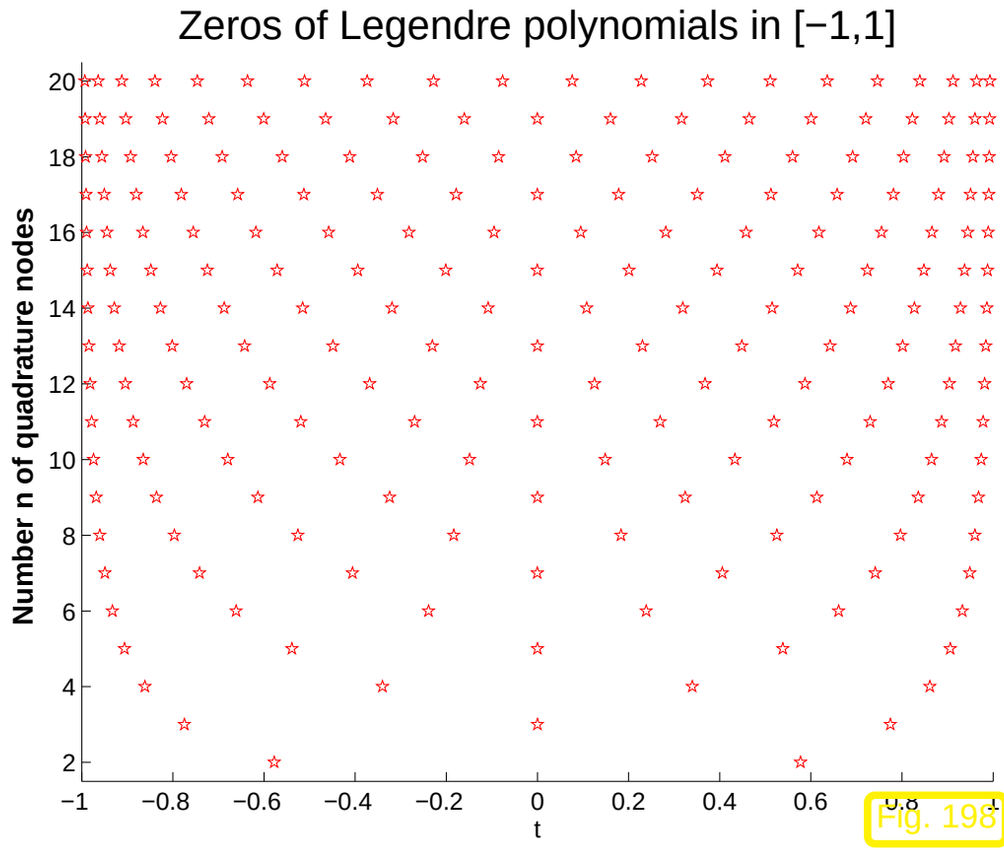


Fig. 198

◁ Obviously:

Lemma 10.4.13 (Zeros of Legendre polynomials)**.**

P_n has n distinct zeros in $]-1,1[$.

Zeros of Legendre polynomials **=** Gauss points

Quadrature formula from Thm. 10.4.8: Gauss-Legendre quadrature
(nodes ξ_j^n **=** Gauss points)

Obviously ▷

Lemma 10.4.14. *(Positivity of Gauss-Legendre quadrature weights)*

The weights of Gauss-Legendre quadrature formulas are positive.

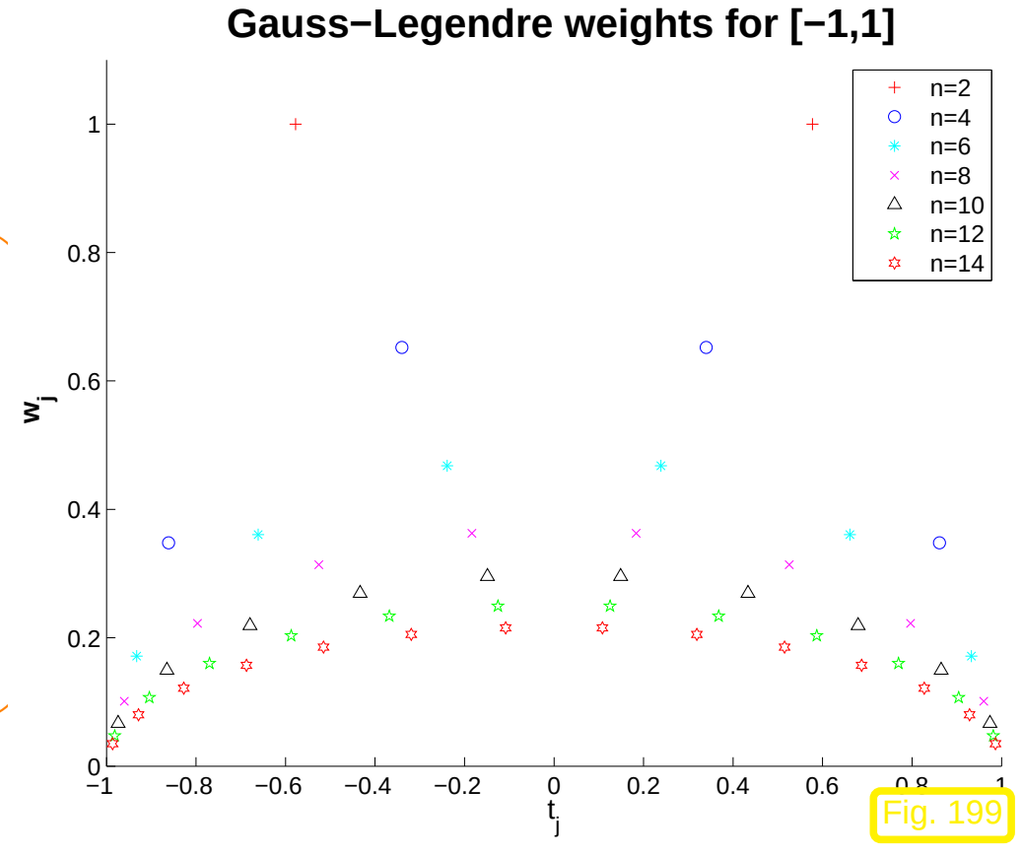
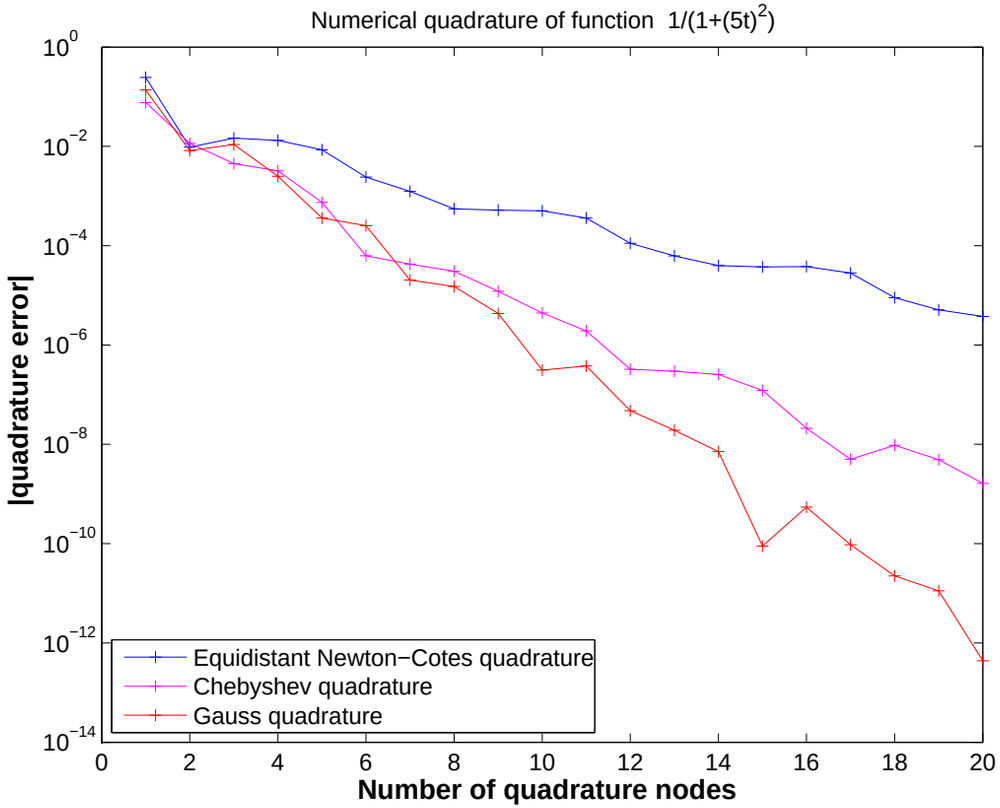
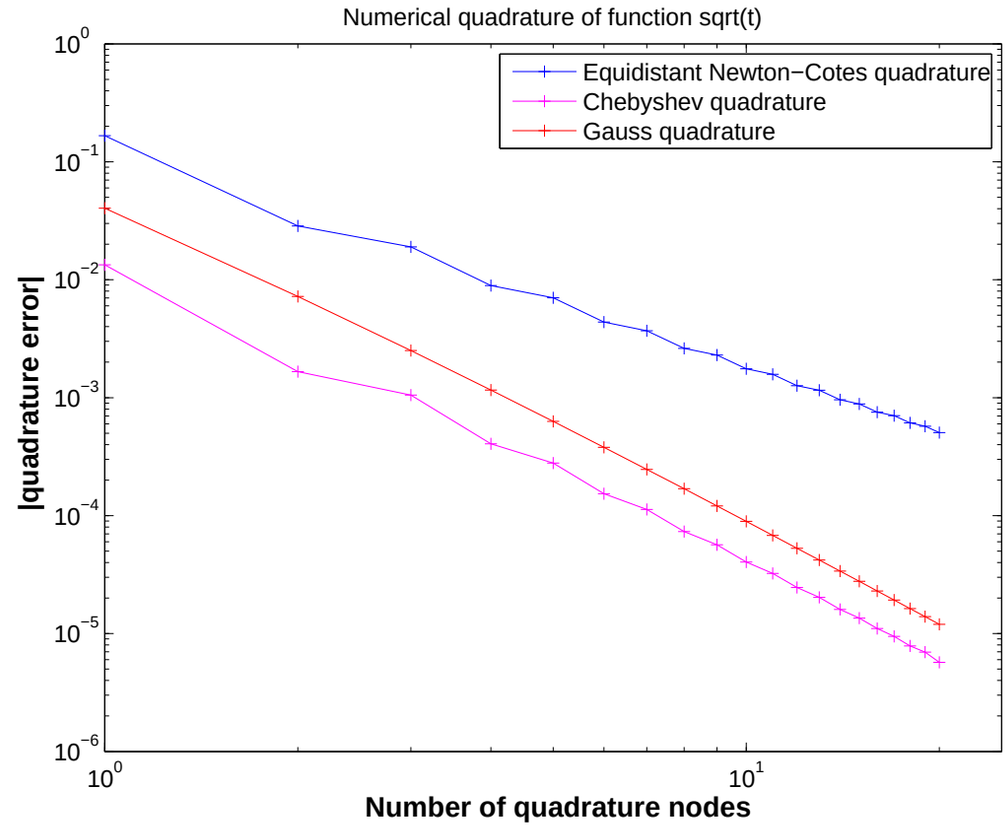


Fig. 199

Example 10.4.22 (Error of (non-composite) quadratures).



quadrature error, $f_1(t) := \frac{1}{1+(5t)^2}$ on $[0,1]$



quadrature error, $f_2(t) := \sqrt{t}$ on $[0,1]$

◇

R. Hiptmair

rev 38355, October 30, 2011

10.5 Adaptive Quadrature

Example 10.5.1 (Rationale for adaptive quadrature).

Consider composite trapezoidal rule (10.3.3) on mesh $\mathcal{M} := \{a = x_0 < x_1 < \cdots < x_m = b\}$:

Local quadrature error (for $f \in C^2([a,b])$):

$$\int_{x_{k-1}}^{x_k} f(t)\,\mathrm{d}t - \frac{1}{2}(f(x_{k-1}) + f(x_k)) \leq (x_k - x_{k-1})^3 \left\|f''\right\|_{L^\infty([x_{k-1},x_k])} .$$

➢ Do not use equidistant mesh !

Refine $\mathcal{M}$, where $|f''|$ large !

Makes sense, e.g., for "spike function" ▷

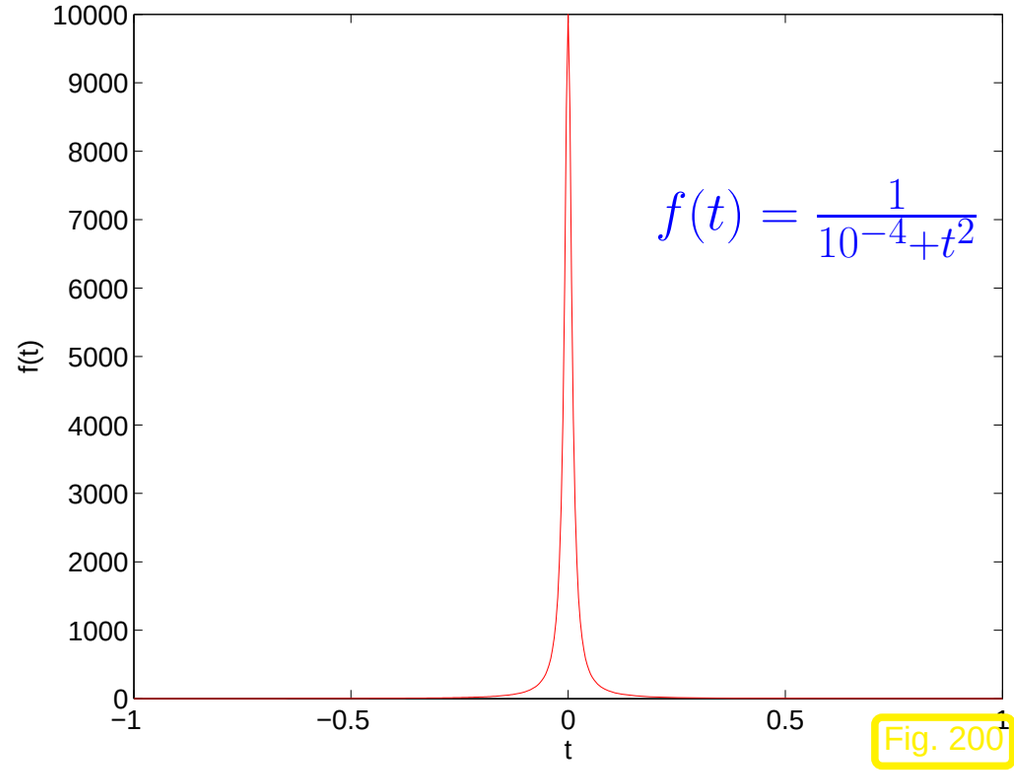
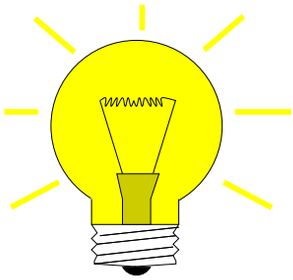


Fig. 200

◇

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 $\rightarrow$ local error estimates

heuristics: error$(Q_2) \ll$ error(Q_1) $\Rightarrow$ error$(Q_1) \approx Q_2(f) - Q_1(f)$.

Now: Q_1 = trapezoidal rule (order 2) $\leftrightarrow$ Q_2 = Simpson rule (order 4)

Given: mesh $\mathcal{M} := \{a = x_0 < x_1 < \cdots < x_m = b\}$

❶ (error estimation)

For $I_k = [x_{k-1}, x_k]$, $k = 1, \ldots, m$ (midpoints $p_k := \frac{1}{2}(x_{k-1} + x_k)$)

$$\mathtt{EST}_k := \Bigg| \underbrace{\frac{h_k}{6}(f(x_{k-1}) + 4f(p_k) + f(x_k))}_{\text{Simpson rule}} - \underbrace{\frac{h_k}{4}(f(x_{k-1}) + 2f(p_k) + f(x_k))}_{\text{trapezoidal rule on split mesh interval}} \Bigg| . \qquad (10.5.2)$$

❷ (Termination)

Simpson rule on $\mathcal{M}$ $\Rightarrow$ preliminary result I

If $\displaystyle\sum_{k=1}^{m} \mathtt{EST}_k \leq \mathtt{RTOL} \cdot I$ ($RTOL$:= prescribed tolerance) $\Rightarrow$ **STOP** (10.5.3)

❸ (local mesh refinement)

$$\mathcal{S} := \{k \in \{1,\dots,m\}\colon \mathtt{EST}_k \geq \eta \cdot \frac{1}{m}\sum_{j=1}^{m} \mathtt{EST}_j\} \;, \quad \eta \approx 0.9 \;. \tag{10.5.4}$$

► new mesh: $\mathcal{M}^* := \mathcal{M} \cup \{p_k\colon k \in \mathcal{S}\}$.

Then continue with step ❶ and mesh $\mathcal{M} \leftarrow \mathcal{M}^*$.

Non-optimal recursive MATLAB implementation:

Code 10.5.5: h-adaptive numerical quadrature

```
function I = adaptquad(f,M,rtol,abstol)
h = diff(M);                          %
mp = 0.5*(M(1:end-1)+M(2:end));       %
fx = f(M); fm = f(mp);                %
trp_loc  = h.*(fx(1:end-1)+2*fm+fx(2:end))/4; %
simp_loc = h.*(fx(1:end-1)+4*fm+fx(2:end))/6; %
I = sum(simp_loc);                    %
est_loc  = abs(simp_loc -trp_loc);    %
err_tot  = sum(est_loc);              %
%
if ((err_tot > rtol*abs(I)) and (err_tot > abstol))
```

```
  refcells = find(est_loc > 0.9*sum(est_loc)/length(h));
  I = adaptquad(f,sort([M,mp(refcells)]),rtol,abstol); %
end
```

Example 10.5.6 (h-adaptive numerical quadrature).

- approximate $\int_0^1 \exp(6\sin(2\pi t))\,\mathrm{d}t$, initial mesh $\mathcal{M}_0 = \{j/10\}_{j=0}^{10}$

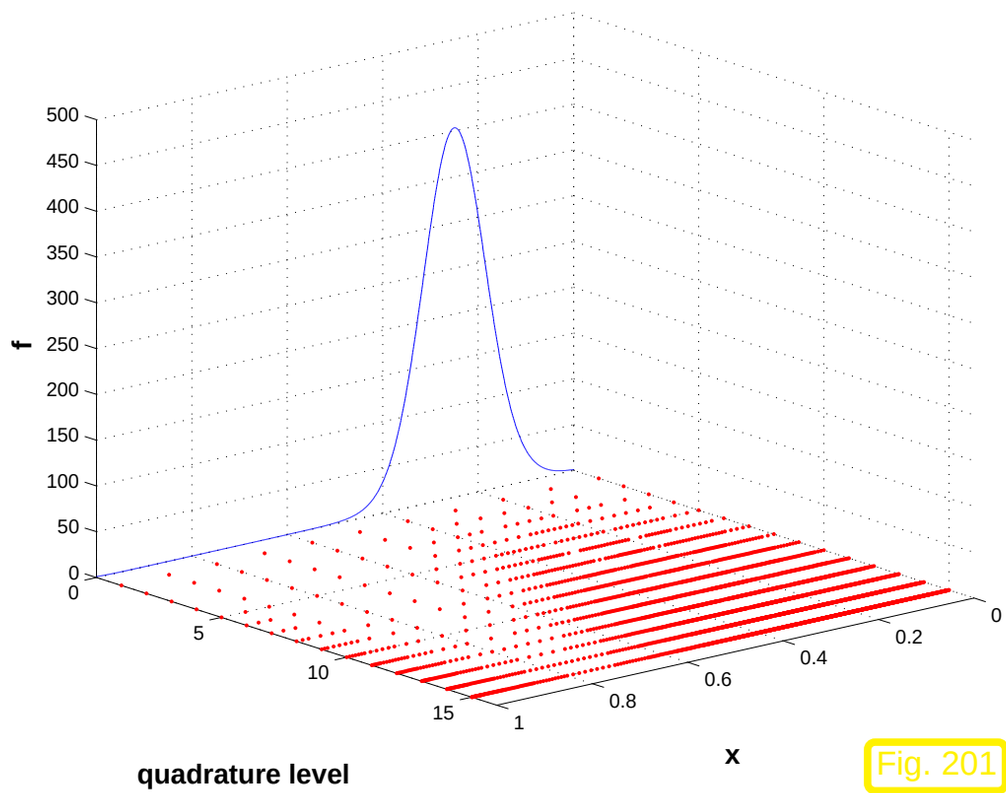


Fig. 201

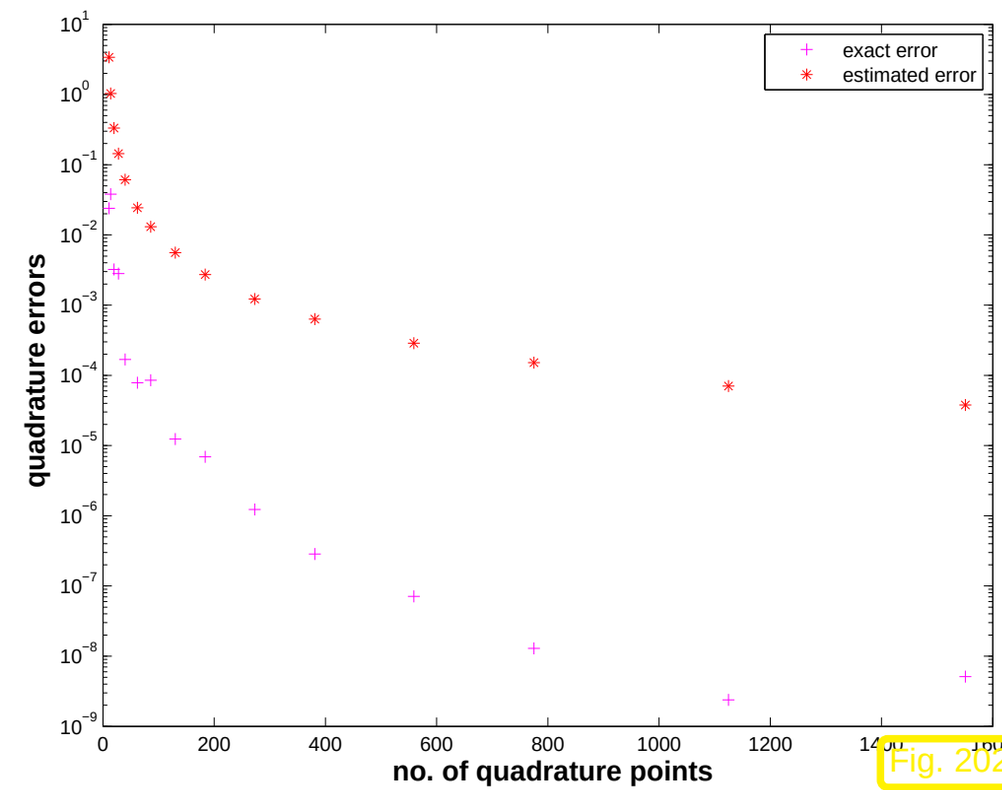


Fig. 202

- approximate $\int_0^1 \min\{\exp(6\sin(2\pi t)), 100\}\,\mathrm{d}t$, initial mesh as above

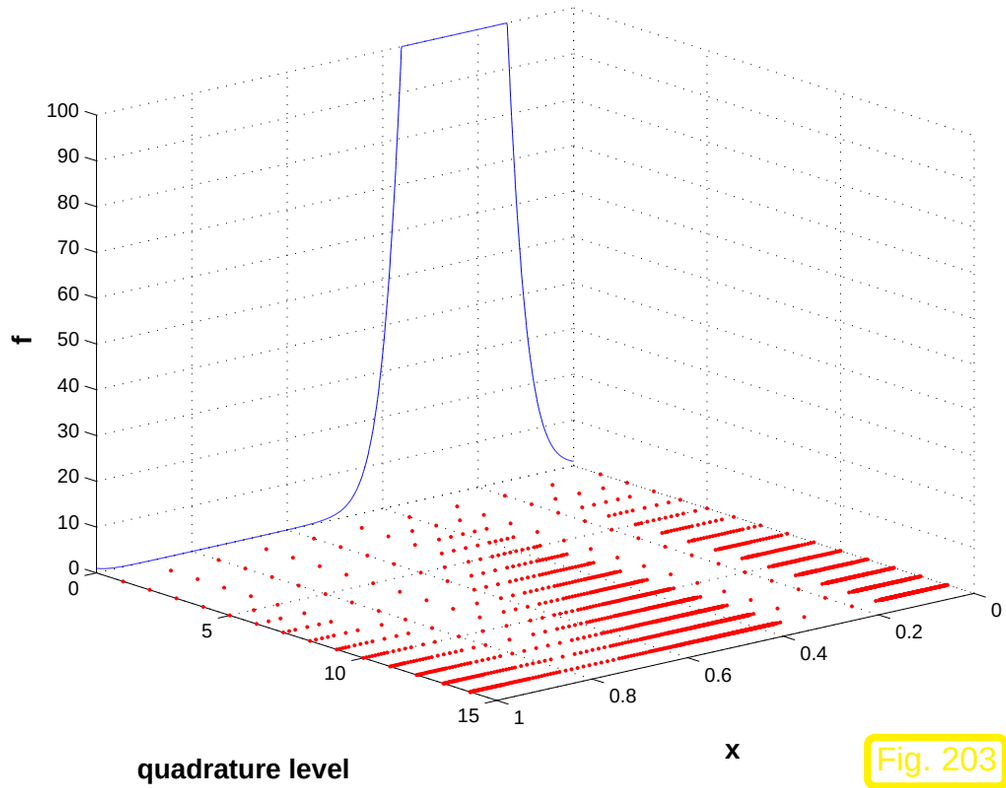


Fig. 203

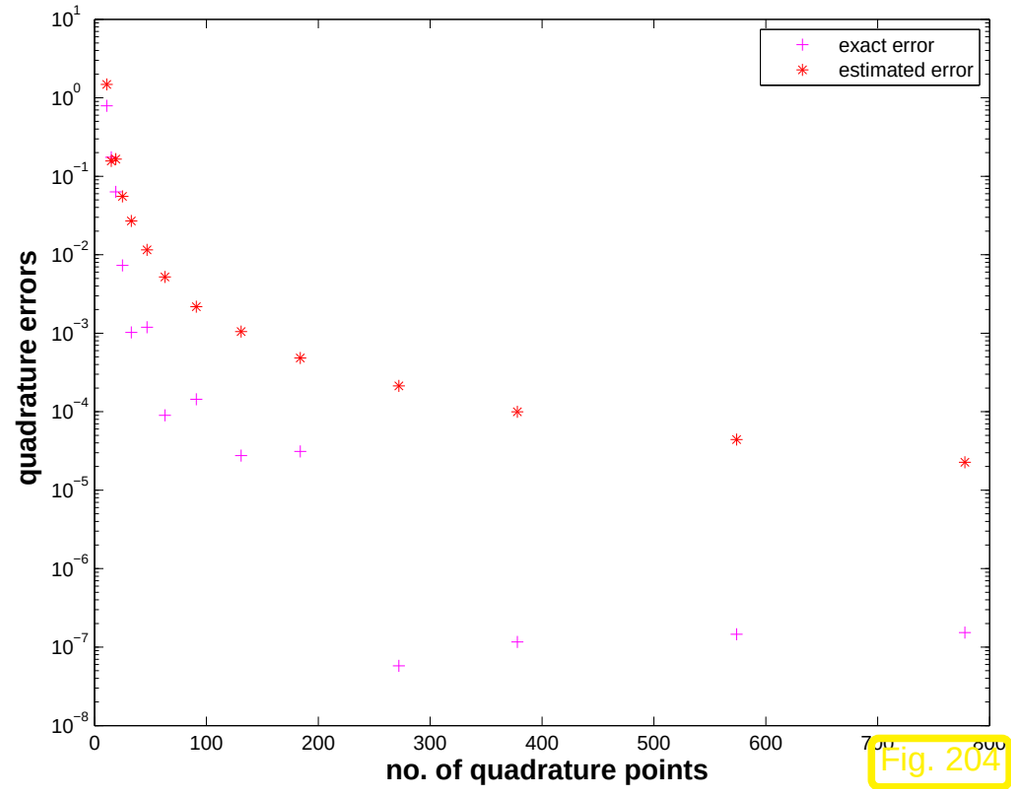


Fig. 204

◇

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

△

R. Hiptmair

rev 38355, October 30, 2011

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 $x_1 < x_2 < \cdots < x_n$, $x_j \in I$, $y_1 < y_2 < \cdots < y_m$, $y_j \in J$

Collocation Matrix: $\mathbf{M} \in \mathbb{C}^{n,m} \quad :\Leftrightarrow \quad (\mathbf{M})_{ij} := G(x_i, y_j)\ , \quad 1 \leq i \leq n, 1 \leq j \leq m\ .$ (11.1.1)

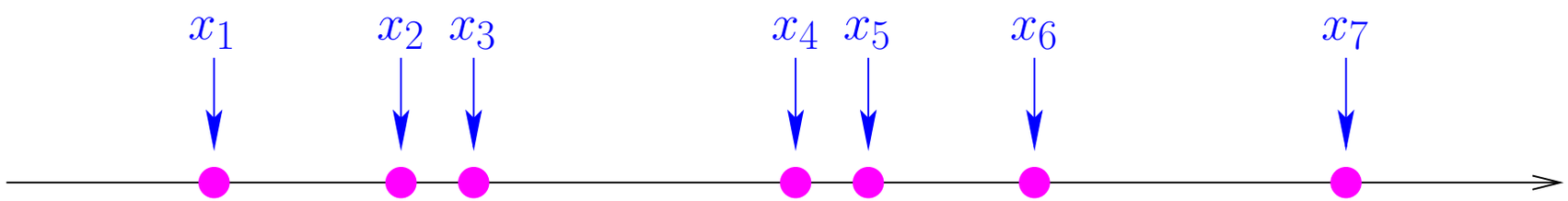
We have to find: *Efficient* algorithms for approximate evaluation of $\mathbf{M} \times$Vector

(Note: Computational Effort $O(mn)$!)

R. Hiptmair

rev 38355, May 28, 2009

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: $\quad f_j = \sum_{\substack{k=1 \\ k\neq j}}^{n} \frac{1}{|x_j - x_k|} c_k c_j \ , \quad j = 1, \dots, n \ .$

- Force on every wire is given by vector $\quad \mathbf{f} = \operatorname{diag}(c_1, \dots, c_n)\mathbf{M} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$,

  where $\quad \mathbf{M} = (m_{ij})_{i,j=1}^{n} \ , \quad m_{ij} = \begin{cases} \frac{1}{|x_j - x_i|} & \text{for } i \neq j \ , \\ 0 & \text{for } i = j \ . \end{cases}$

Collocation matrix $\mathbf{M}$ will be formed using kernel function $\quad G(x, y) = \frac{1}{|x-y|}$

$\diamond$

R. Hiptmair

rev 38355, May 28, 2009

Example 11.1.3 (Gravitational forces in galaxy).

Number of stars in galaxy n ($\approx 10^9$) with position $\boldsymbol{x}_i \in \mathbb{R}^3$ and mass m_i, $i = 1, \ldots, n$.

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 :

$$f_j = \frac{G}{4\pi} \sum_{i \neq j} \frac{1}{\|\mathbf{x}_i - \mathbf{y}_j\|} m_i m_j \ , \quad j \in \{1, \ldots, n\} \ .$$

- Gravitational force on every star is given by

$$\mathbf{f} = \operatorname{diag}(m_1, \ldots, m_n)\mathbf{M} \begin{pmatrix} m_1 \\ \vdots \\ m_n \end{pmatrix} \ , \quad m_{ij} := \begin{cases} \frac{G}{4\pi} \frac{1}{\|\mathbf{x}_i - \mathbf{y}_j\|} & \text{for } i \neq j \ , \\ 0 & \text{for } i = j \ , \end{cases} \quad 1 \leq i, j \leq n \ .$$

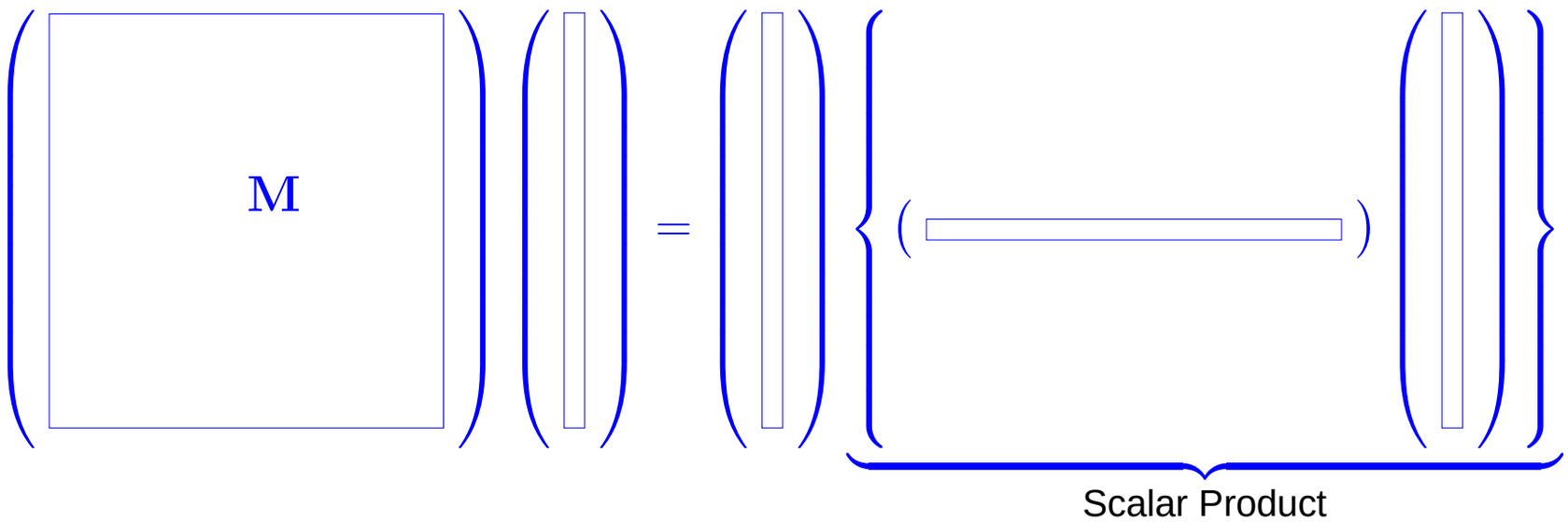
The above example is a 3D generalization of our orignal task. ◇

11.2 Local Separable Approximation

If kernel function is separable i.e. : $G(x,y) = g(x)h(y)$, $g : I \mapsto \mathbb{C}$, $h : J \mapsto \mathbb{C}$

$$\mathbf{M} = \left(g(x_j)\right)_{j=1}^{n} \cdot \left(\left(h(y_j)\right)_{j=1}^{m}\right)^T \qquad \operatorname{rank}(\mathbf{M}) = 1 \ .$$

Computational Effort ($\mathbf{M}\times$Vector) = $m+n$



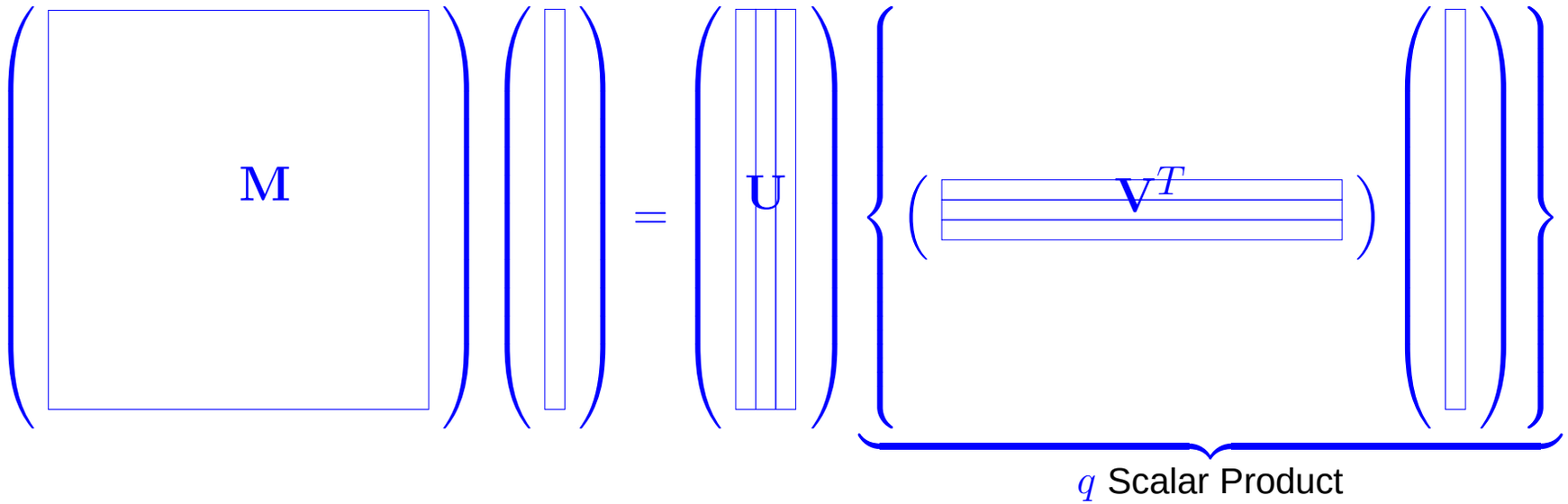
Generalization: $G(x,y) = \sum_{j=1}^{q} g_j(x)h_j(y)$, $g_j : I \mapsto \mathbb{C}$, $h_j : J \mapsto \mathbb{C}$, $q \in \mathbb{N}$

$$\mathbf{M} = \mathbf{U}\mathbf{V}^T \ , \quad \begin{array}{lll} \mathbf{U} \in \mathbb{R}^{n,q} \ , & u_{ij} = g_j(x_i) \ , & j \in \{1,\dots,q\}, i \in \{1,\dots,n\} \ , \\ \mathbf{V} \in \mathbb{R}^{q,m} \ , & v_{ij} = h_i(y_j) \ , & i \in \{1,\dots,q\}, j \in \{1,\dots,m\} \ . \end{array}$$

R. Hiptmair

rev 38355, August 18, 2011

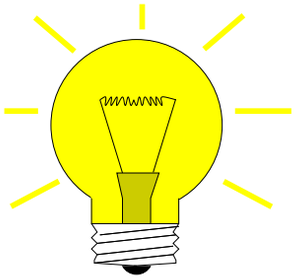
If $\mathrm{rank}(\mathbf{M}) = q$ then Computational Effort ($\mathbf{M}\times$ Vector) $= q(m+n)$



R. Hiptmair

rev 38355, August 18, 2011

Idea: Global approximation of G through sum of separated kernel function:



$$G(x,y) \approx \widetilde{G}(x,y) := \sum_{l=0}^{d}\sum_{k=0}^{d} \kappa_{l,k} g_l(x) h_k(y)\,, \quad \kappa_{l,j} \in \mathbb{C}\,, \quad (x,y) \in I \times J\,. \tag{11.2.1}$$

As G is approximated by $\widetilde{G}$ therfore $\mathbf{M}$ is approximated by $\widetilde{\mathbf{M}}$, where $\widetilde{m}_{ij} = \widetilde{G}(x_i, y_j)$

Remark 11.2.2 (Quality measure for kernel approximation).

$$\left\|\mathbf{M}-\widetilde{\mathbf{M}}\right\|_2^2 \leq \left\|\mathbf{M}-\widetilde{\mathbf{M}}\right\|_F^2 = \sum_{i,j}(m_{ij}-\widetilde{m}_{ij})^2 = \boxed{\sum_{i,j}(G(x_i,y_j)-\widetilde{G}(x_i,y_j))^2} \; .$$

Normalizing quality measure: $$\frac{1}{mn}\sum_{i,j}(G(x_i,y_j)-\widetilde{G}(x_i,y_j))^2 \; . \tag{11.2.3}$$

△

Now, our aim is to find seperable kernel approximation

R. Hiptmair

rev 38355, August 18, 2011

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. Continous* $f : I \times J \mapsto \mathbb{C}$ *defined by*

$$(\mathsf{I}_{\mathcal{X}\times\mathcal{Y}} f)(x,y) := \sum_{j=0}^{n}\sum_{k=0}^{m} f(x_j,y_k)L_j^x(x)L_k^y(y) \; , \quad x,y \in \mathbb{R} \; ,$$

This is tensor product interpolation polynomial.

$$G(x,y) \approx \widetilde{G}(x,y) := \sum_{j=0}^{d}\sum_{k=0}^{d} G(t_j^x, t_k^y) L_j^x(x) L_k^y(y) \ , \tag{11.2.5}$$

$t_0^x, \dots, t_d^x$/$t_0^y, \dots, t_d^y$ 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 $G(x,y) = \dfrac{1}{1+|x-y|^2}$, collocation points $x_i = y_i = \frac{i-1}{n}$, $i = 1,\dots,n$.
- $\widetilde{G}$ approximated by tensor product Chebyshev interpolation polynomial (11.2.10) of degree $d \in \mathbb{N}$.

$$\frac{1}{n^2}\sum_{i,j}(G(x_i,y_j) - \widetilde{G}(x_i,y_j))^2$$

as function of d for $n \in \{100, 200, 400\}$

$\triangleright$

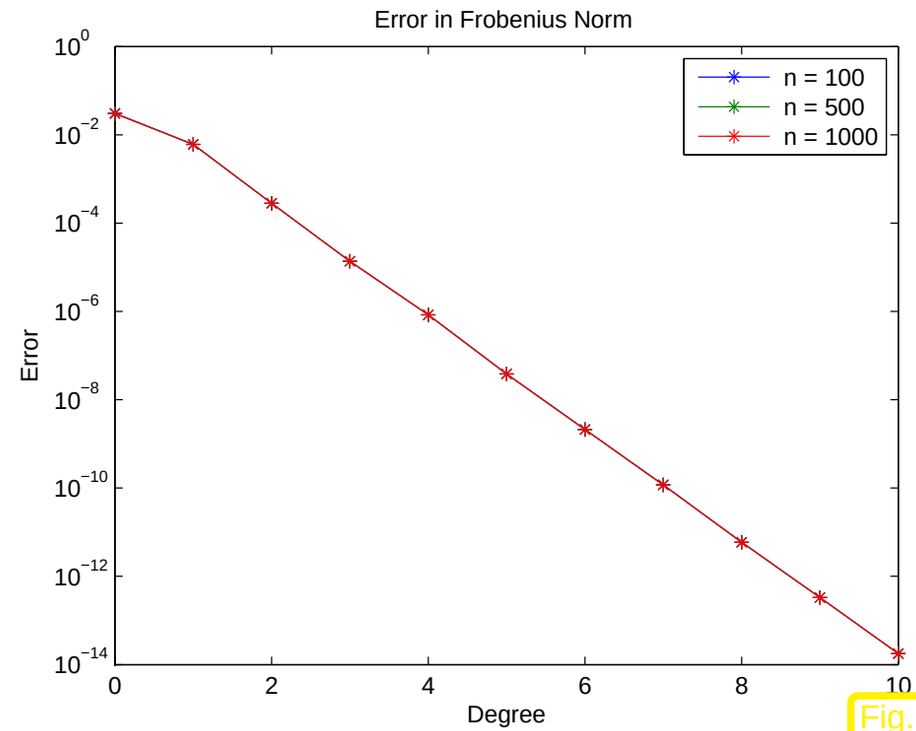


Fig. 205

In file changeable.c
a) put the admissibility condition as return true in admissible_ClusterPair
b) put kernel function as 1.0/(1.0 + pow(fabs(dX - dY),2)) 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: Exponential Connvergence $\left\|\mathbf{M} - \widetilde{\mathbf{M}}\right\|_F \to 0$ 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

$$G(x,y) = \begin{cases} \frac{1}{|x-y|} & \text{if } x \neq y \text{ ,} \\ 0 & \text{otherwise.} \end{cases} \qquad (11.2.8)$$

from ex. 11.1.2.

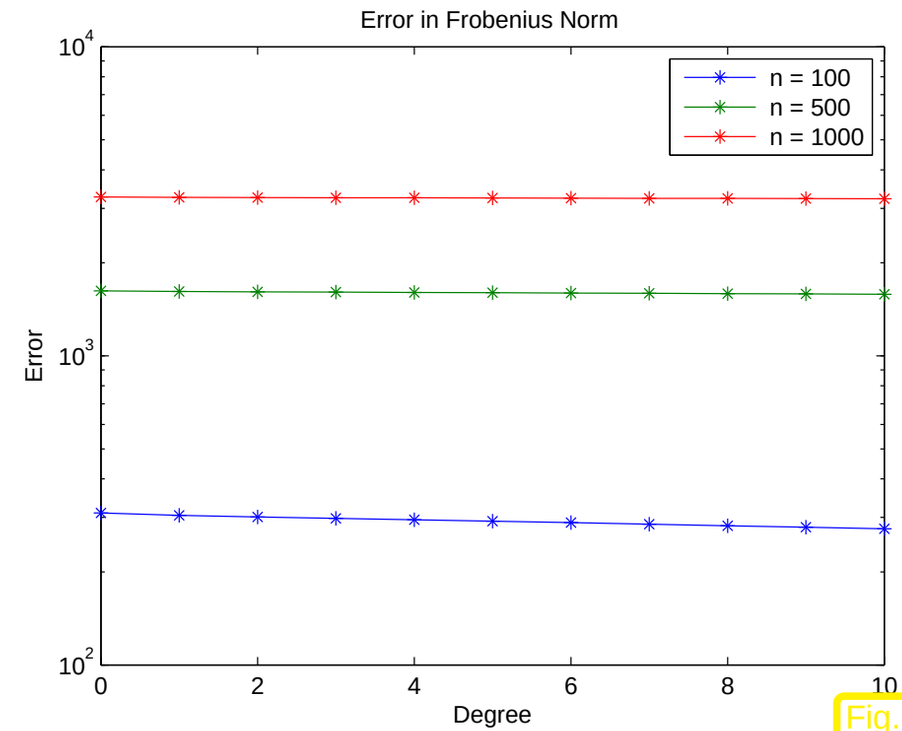


Fig. 206

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

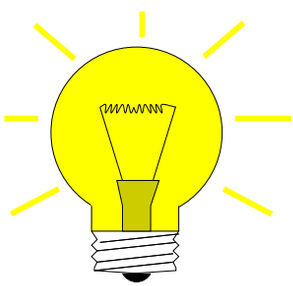
Note: (Virtually) No Convergence $\left\|\mathbf{M} - \widetilde{\mathbf{M}}\right\|_F \to 0$ for $d \to \infty$ ◇

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 thorugh sum of seperated kernel function



$$G(x,y) \approx \sum_{l=0}^{d} \sum_{k=0}^{d} \kappa_{l,k} g_l(x) h_k(y) \;,\; \kappa_{l,j} \in \mathbb{C} \;, \qquad (11.2.9)$$
$$(x,y) \in \widetilde{I} \times \widetilde{Y}, \widetilde{I} \subset I, \widetilde{J} \subset J \;,$$

with $\widetilde{I} \times \widetilde{J} \cap \{(x,y) : x = y\} = \emptyset$.

Actually:

Local approxmation of G on rectangle $\widetilde{I} \times \widetilde{J}$ 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)

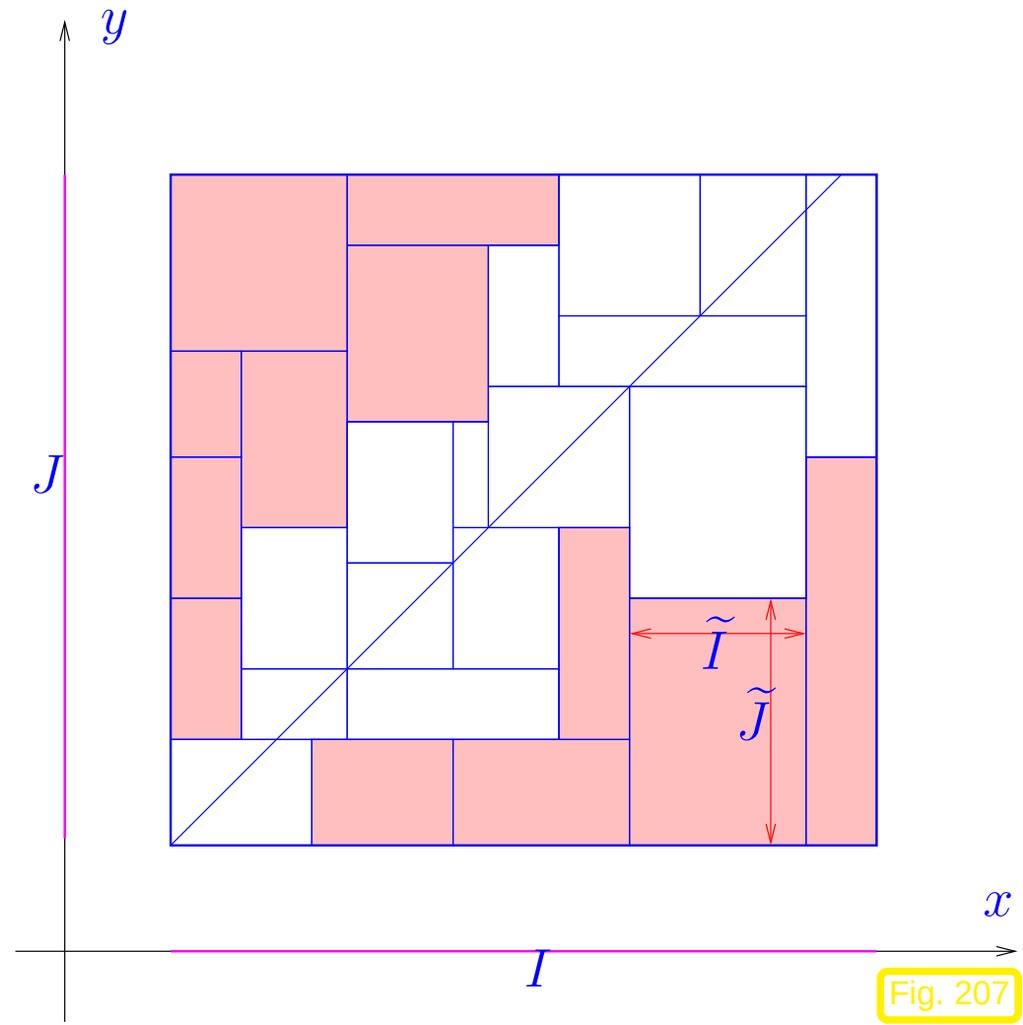


Fig. 207

R. Hiptmair

rev 38355, August 18, 2011

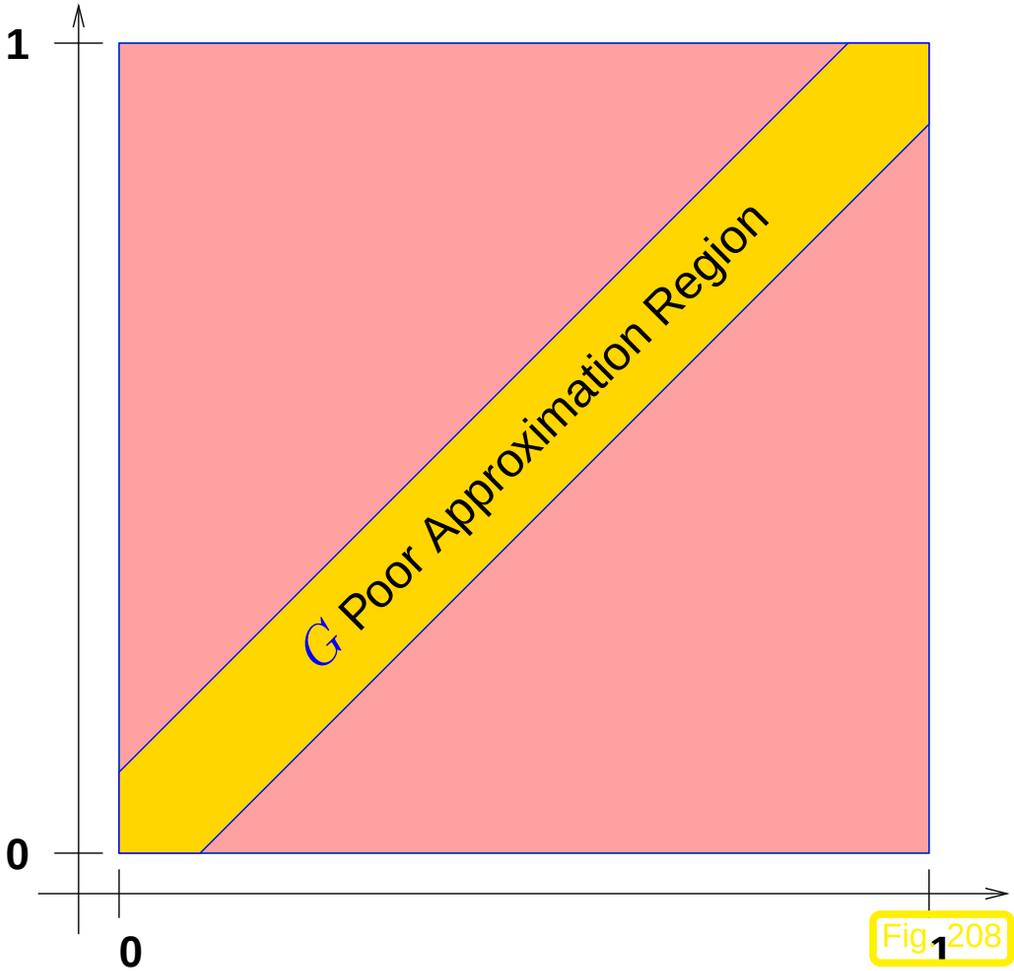


Fig. 208

Terminology:

Near Field: G Poor approximation

Far Field: G Good approximation

R. Hiptmair

rev 38355, August 18, 2011

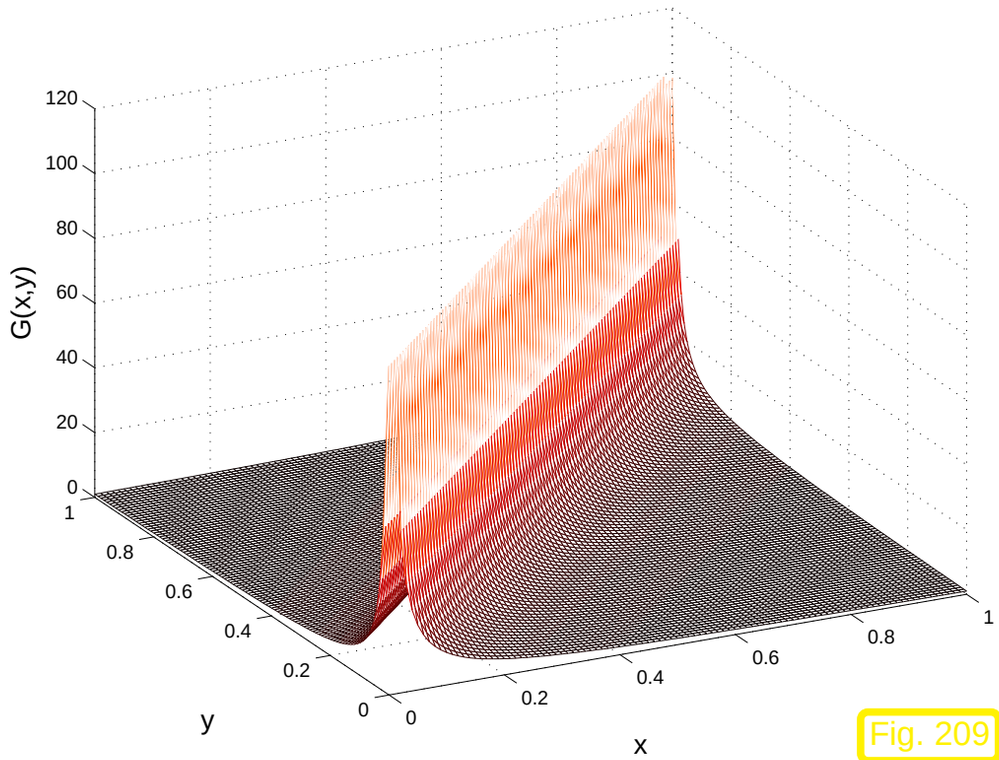


Fig. 209

Kernel function (11.2.8) for $[0,1]^2$

$$G(x,y) = \frac{1}{|x-y|}$$

- Singular for $x = y$
- Analytic far from $\{(x,y) \colon x = y\}$, analyticity ($\rightarrow$ Chapter **??**) increases with $|x-y|$

Similar kernel function: $G(x,y) = \log|x-y|$, $G(x,y) = \dfrac{\partial}{\partial x}\dfrac{1}{|x-y|}$, etc.

R. Hiptmair

rev 38355, August 18, 2011

"“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

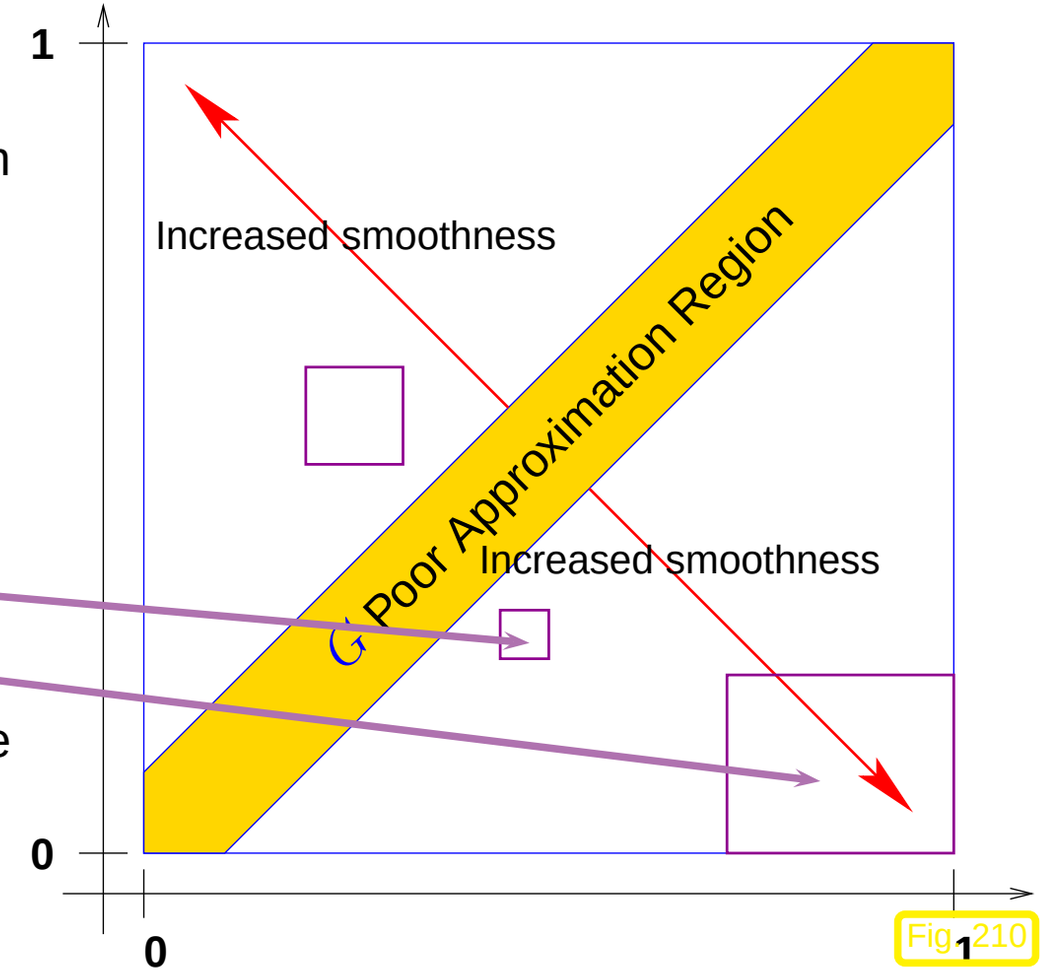


Fig. 210

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 Chebychev interpolation polynomial of G on rectangles of constant size, but with growing distance from $\{(x,y) \colon x = y\}$

$$\widetilde{I}_k = [0.55 + k \cdot 0.05, 0.75 + k \cdot 0.05] \quad , \quad \widetilde{J}_k = [0.25 - k \cdot 0.05, 0.45 - k \cdot 0.05] \; , \quad k \in \{0, \dots, 5\} \; .$$

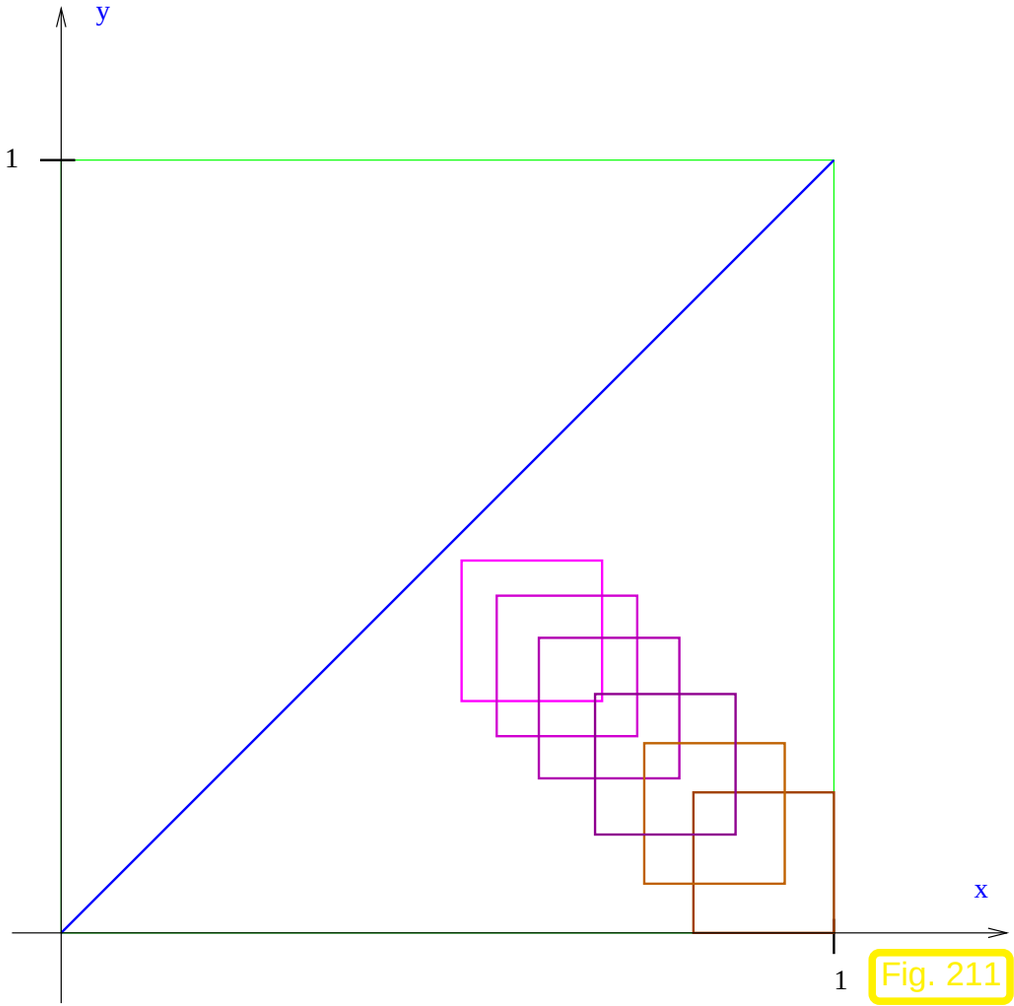


Fig. 211

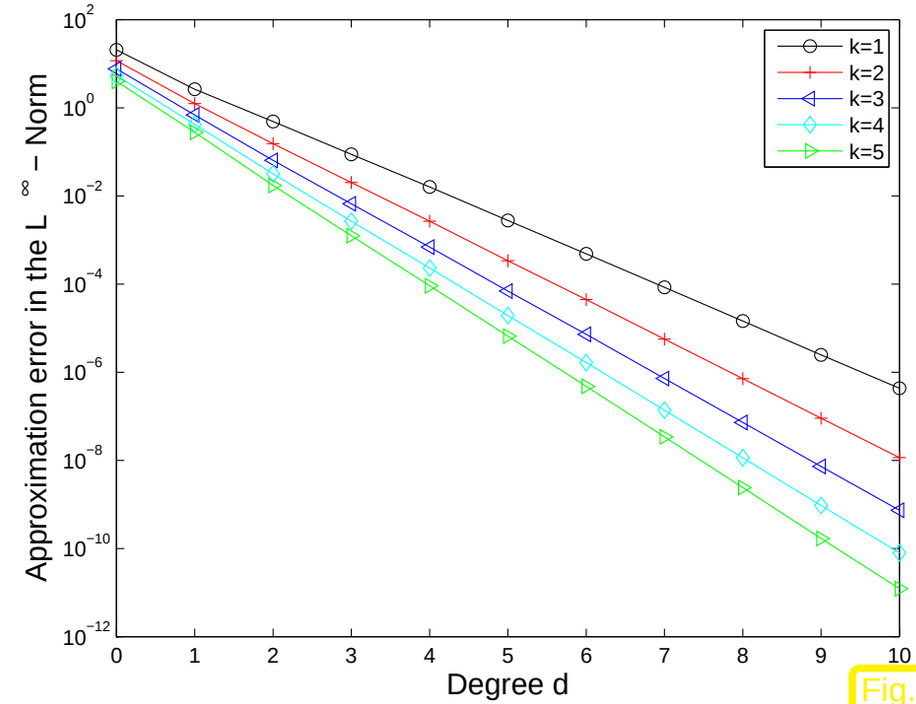


Fig. 212

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

$$[\tfrac{1}{2}(\sqrt{2}-1)\xi + \tfrac{1}{2}, \tfrac{1}{2}(\sqrt{2}+1)\xi + \tfrac{1}{2}] \times [-\tfrac{1}{2}(\sqrt{2}-1)\xi + \tfrac{1}{2}, -\tfrac{1}{2}(\sqrt{2}+1)\xi + \tfrac{1}{2}] \;, \quad 0.05 \le \xi \le \frac{1}{1+\sqrt{2}} \;.$$

that is, Size of rectangles $\sim$ Distance from diagonal

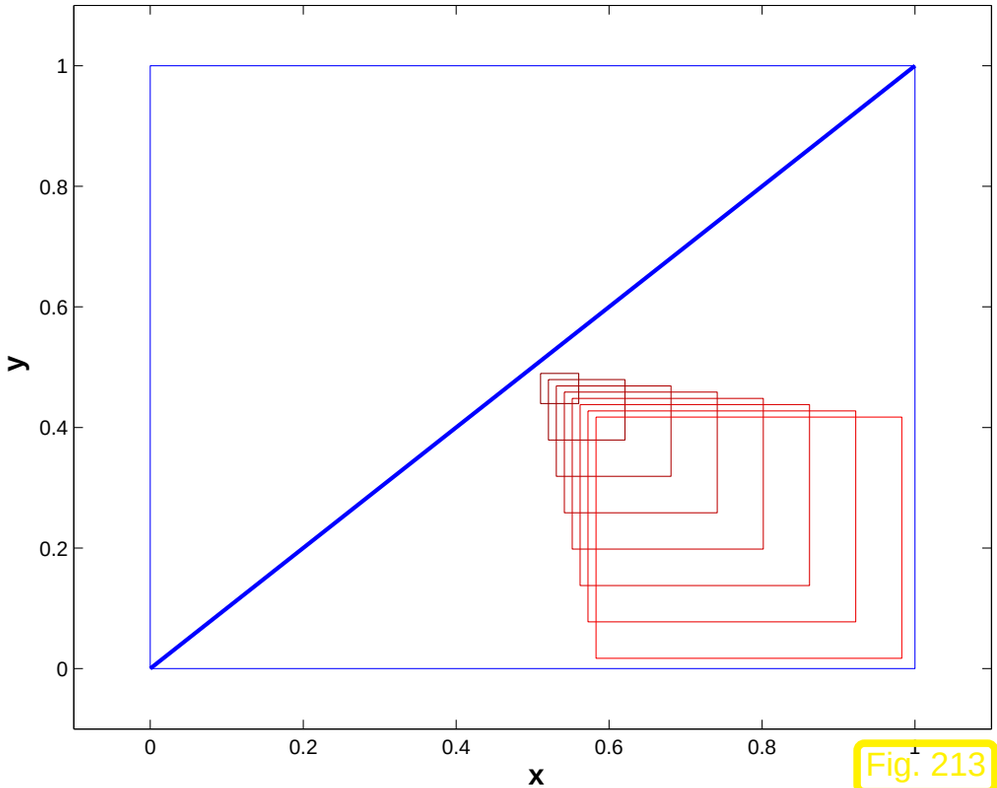


Fig. 213

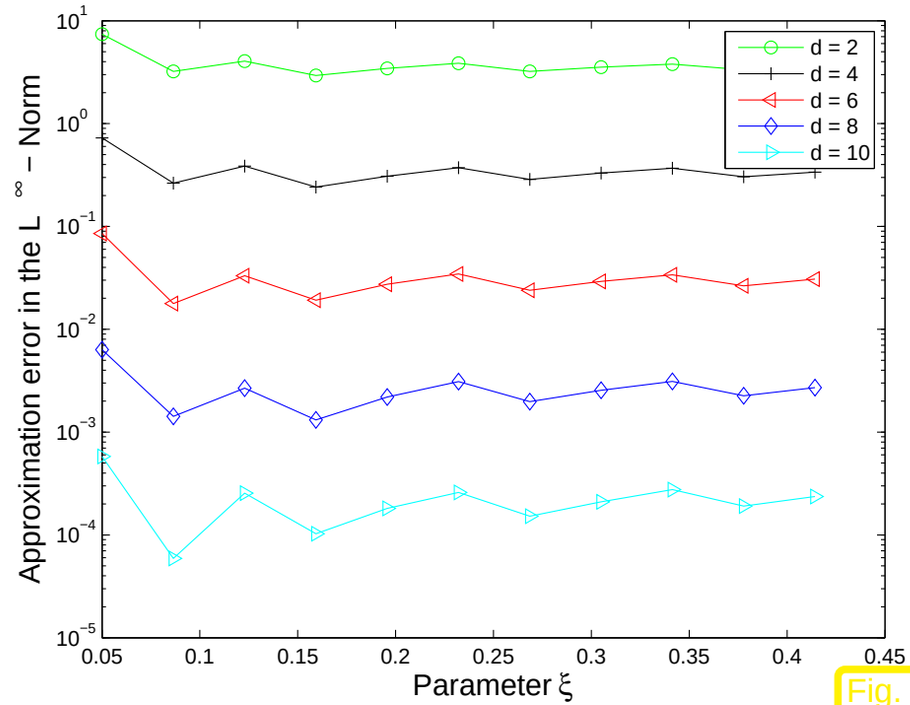


Fig. 214

NOTE: Controlled decrease of interpolation errors with increase in size of rectangles

◇

Acceptability Criteria

$[a,b] \times [c,d]$ is called *η- admissible*, $\eta > 0$, if

distance from diagonal

Size of the rectangle

$$\eta \operatorname{dist}([a,b],[c,d]) \geq \max\{b-a, d-c\} \; . \tag{11.2.12}$$

$(\operatorname{dist}([a,b],[c,d]) := \min\{|x-y| \colon x \in [a,b], y \in [c,d]\})$

11.3 Cluster Trees

TASK: Given $\eta > 0$, find (efficient algorithms) partitioning of $I \times J$ "'far from diagonal" which are η-admissible rectangles.

Remark 11.3.1. Perspective:

$$\begin{array}{c}\text{Partitioning}\\ \text{of } I \times J\end{array} \Longrightarrow \begin{array}{c}\text{Partitioning}\\ \text{of } \{x_i\}_{i=1}^n \times \{y_j\}_{j=1}^m\end{array} \Leftrightarrow \begin{array}{c}\text{Partitioning of}\\ \text{index mesh } \{1,\dots,n\} \times \{1,\dots,m\}\end{array}$$

△

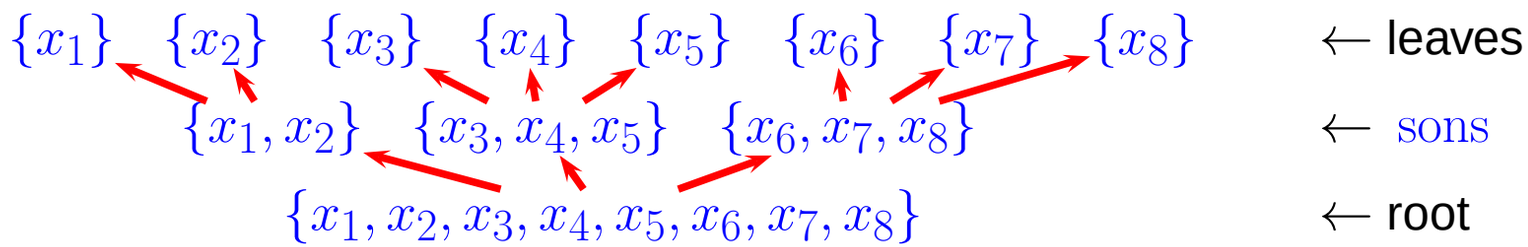
R. Hiptmair

rev 38355, November 27, 2011

Definition 11.3.2 (Cluster Tree)**.** *A Tree (→ Computer Science,Graph theory) T is called Cluster Tree on $\mathbb{P} := \{x_1,\dots,x_n\} \subset \mathbb{R}$:⇔*

- *The nodes of the Tree T (= Cluster) are subset of $\mathbb{P}$.*
- $\mathrm{root}(T) = \mathbb{P}$.
- *For every node σ of T: $\{\sigma' \colon \sigma' \in \mathrm{sons}(\sigma)\}$ is Partitioning of σ.*

Bounding Box of Cluster $\sigma \in T$: $\Gamma(\sigma) := [\min\{x\}_{x\in\sigma}, \max\{x\}_{x\in\sigma}]$



Terminology ($\rightarrow$ Graph theory): Cluster Tree T: $\sigma \in T$: $\mathrm{sons}(\sigma) = \emptyset$ $:\Leftrightarrow$ σ leaf

Cluster Tree T:

Recursive Construction

MATLAB-Data Structure:

$N \times 6$-Matrix, $N = \sharp T$,

Lines $\hat{=}$ Cluster

```
T = [T;i,j,xl,xr,s1,s2]
```

i, j : $\sigma = \{x_i, \ldots, x_j\}$

xl : $\mathrm{xl} = \min_{i \le k \le j} x_k$

xr : $\mathrm{xr} = \max_{i \le k \le j} x_k$

s1, s2: The line indices of son

NOTE : $\sharp T \le 2n$

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

```
void clsClusteringApproximation::build_ClusterTree_Linear(int iOffset, int
   iNumberofPtsInClusterLinear, int* piDimension){
   int iTempLeftChild, iTempRightChild, iStart, iEnd;
   int iStartLeft, iPtsInLeft, iStartRight, iPtsInRight;

   iStart = iOffset;
   iEnd = iOffset + iNumberofPtsInClusterLinear - 1;
   if (iNumberofPtsInClusterLinear == 1){
      add_ClusterLinear(*piDimension, iStart, iEnd, -1, -1);
   }
   else{
      iStartLeft = iOffset;
      iPtsInLeft =  iNumberofPtsInClusterLinear - (iNumberofPtsInClusterLinear/2);
      build_ClusterTree_Linear(iStartLeft, iPtsInLeft, piDimension);
      iTempLeftChild = vec2pclsClusterLinear[*piDimension].size() - 1;

      iStartRight = iOffset + iPtsInLeft;
      iPtsInRight = iNumberofPtsInClusterLinear/2;
      build_ClusterTree_Linear(iStartRight, iPtsInRight, piDimension);
      iTempRightChild = vec2pclsClusterLinear[*piDimension].size() - 1;

      add_ClusterLinear(*piDimension, iStart, iEnd, iTempLeftChild, iTempRightChild);
   }
}
```

Example 11.3.3 (Cluster Tree).

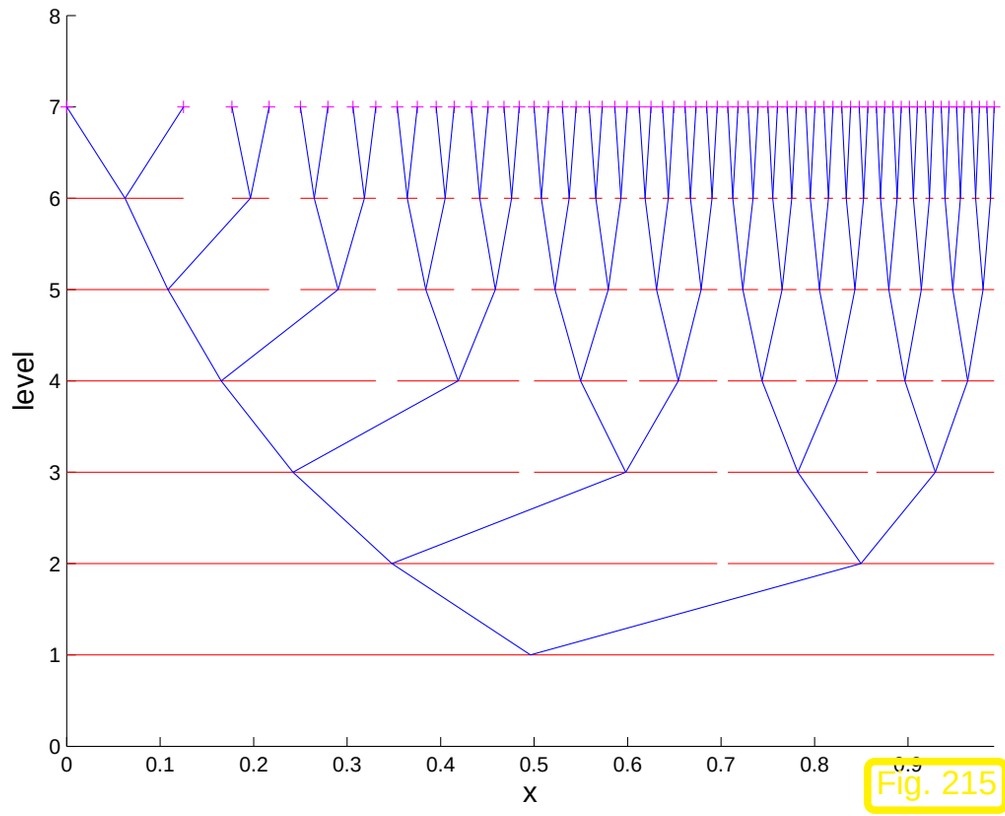


Fig. 215

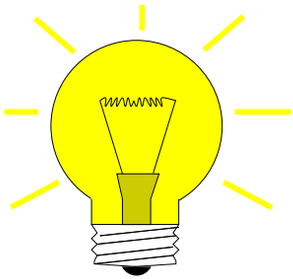
← Binary cluster tree:

```
x = sqrt(0:1/64:(1-1/64));
```

- 2^{l-1} Cluster on Level l with 2^{7-l} points.

(balacnced Cluster tree)

◇



Idea: Choose approximation rectangles = Tensor product of Bounding Box of Cluster $\in T$

R. Hiptmair

rev 38355, November 27, 2011

Cluster σ, μ admissible $:\Leftrightarrow$ $\Gamma(\sigma), \Gamma(\mu)$ admissible

Recursive construction for admissible Partitioning:

`Pnear` = Near field:

The area near the diagonal ($\rightarrow$ No approximation)

`Pfar` = Far field:

Partitioning with η-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!

R. Hiptmair

rev 38355, November 27, 2011

C++ code: Construction for Admissible Partitioning

```
void clsClusteringApproximation::build_ClusterPair (int iIndex1,
  int iIndex2){

    clsClusterLinear *pclsClusterLinear1, *pclsClusterLinear2;

    pclsClusterLinear1 = vec2pclsClusterLinear[0][iIndex1];
    pclsClusterLinear2 = vec2pclsClusterLinear[1][iIndex2];
    if (leaf(0,iIndex1) || leaf(1,iIndex2)){
        add_ClusterPairNear(pclsClusterLinear1,
          pclsClusterLinear2);
        (pclsClusterLinear1->get_ptr_AppearsIn())->push_back(-vecp
        (pclsClusterLinear2->get_ptr_AppearsIn())->push_back(-vecp
    }
    else if (admissible_ClusterPair(iIndex1, iIndex2)){
        add_ClusterPairFar(pclsClusterLinear1,
          pclsClusterLinear2);
        (pclsClusterLinear1->get_ptr_AppearsIn())->push_back(vecpo
        (pclsClusterLinear2->get_ptr_AppearsIn())->push_back(vecpo
    }
    else{
        tree_traverse(pclsClusterLinear1, pclsClusterLinear2);
    }
}
```

C++ code: Preprocessing Code

```
void clsClusteringApproximation::preprocess(){
     int iLoopVari, iLoopVarj;
     for (iLoopVari = 0; iLoopVari < iDimension; iLoopVari++){
          build_ClusterTree_Linear(0, veciNumberofPts[iLoopVari],
            &iLoopVari);
     }
     build_ClusterPair(vec2pclsClusterLinear[0].size() - 1,
       vec2pclsClusterLinear[1].size() - 1);
}
```

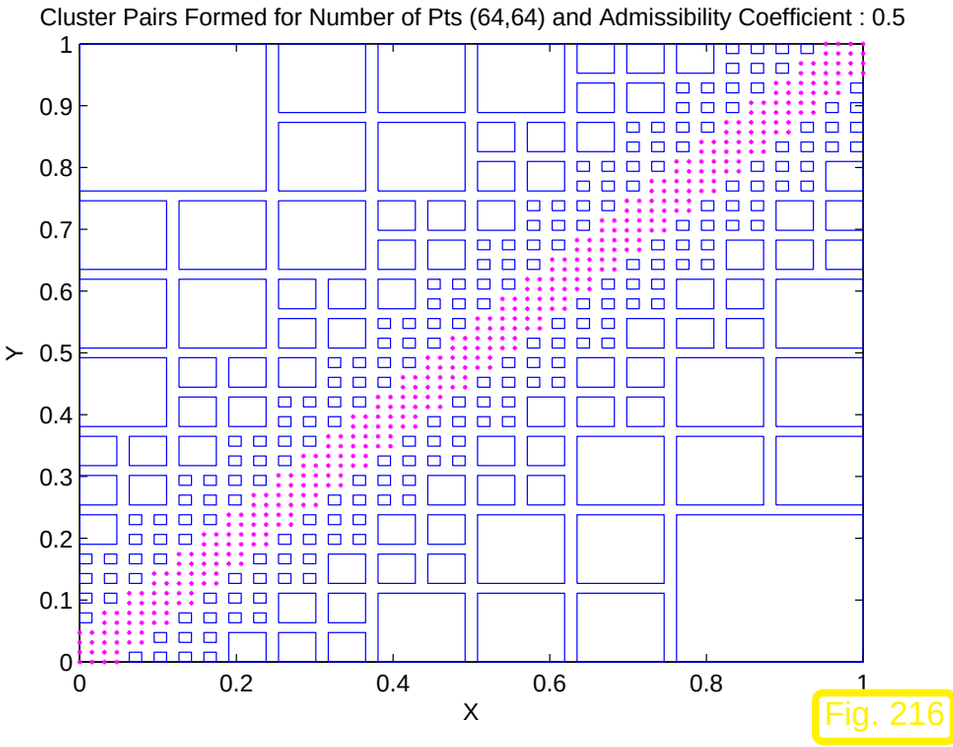


Fig. 216

```
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)

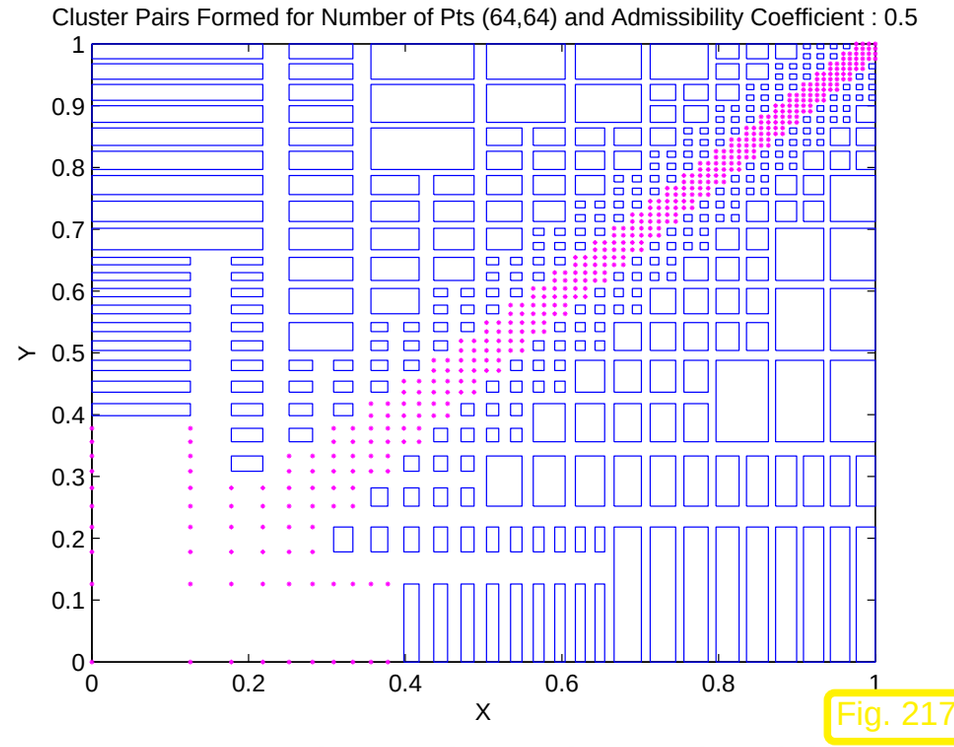


Fig. 217

```
x = 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 , $\square$ = Partition rectangles (Far field)

NOTE : Axis sections of the partition rectangles $\in \{\Gamma(\sigma) \colon \sigma \in T\}$, $T \hat{=}$ Cluster Tree ($\rightarrow$ Def. 11.3.2)

11.4 Algorithm

Local kernel approximation of Far field-Rectangles through Tensor product Chebyshev interpolation polynomial	$\lhd\rhd$	Low rank approximation of $\mathbf{M}$ on *blocks* as (11.1.1)

see Figs. 245,246

Notation:

$$\begin{array}{l} \sigma := \{i_1, \dots, i_2\} \subset \{1, \dots, n\} \\ \mu := \{j_1, \dots, j_2\} \subset \{1, \dots, m\} \end{array} \quad \Rightarrow \quad \mathbf{M}_{|\sigma \times \mu} := (m_{ij})_{\substack{i=i_1,\dots,i_2 \\ j=j_1,\dots,j_2}} \; .$$

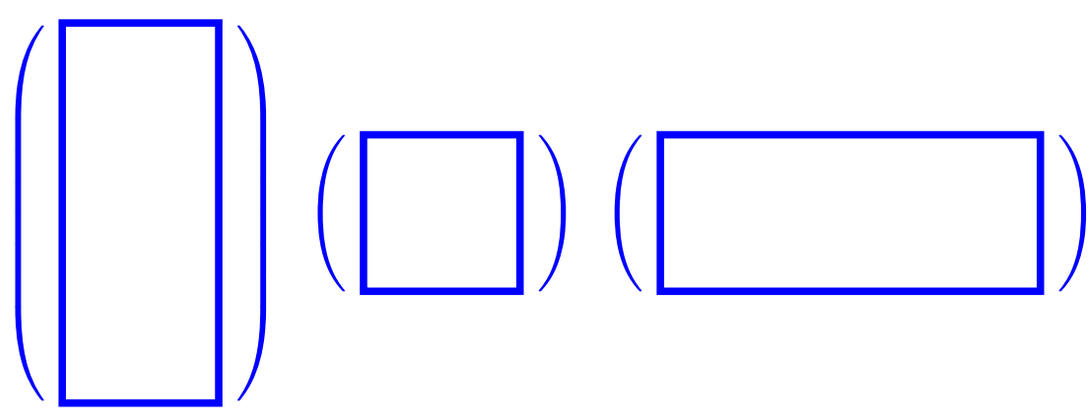
Consider kernel approximation of $[x_{i_1}, x_{i_2}] \times [y_{j_1}, y_{j_2}]$ through Tensor product Chebyshev interpolation polynomial (see (11.2.10)):

$$G(x,y) \approx \widetilde{G}(x,y) := \sum_{j=0}^{d}\sum_{k=0}^{d} G(t_j^x, t_k^y) L_j^x(x) L_k^y(y) \ ,$$

- $t_0^x, \dots, t_d^x / t_0^y, \dots, t_d^y \hat{=}$ Chebyshev nodes for $[x_{i_1}, x_{i_2}]/[y_{j_1}, y_{j_2}]$,
- L_j^x, $L_k^y \hat{=}$ associated Lagrange polynomial.

$$\widetilde{G}(x,y) = \sum_{j=0}^{d}\sum_{k=0}^{d} G(t_j^x, t_k^y) L_j^x(x) L_k^y(y) \ , \quad x \in [x_{i_1}, x_{i_2}],\, y \in [y_{j_1}, y_{j_2}] \ .$$

$$\widetilde{\mathbf{M}}_{|\sigma\times\mu} = \underbrace{\underbrace{\left(L_j^x(x_i)\right)_{\substack{i=i_1,\dots,i_2\\ j=0,\dots,d}}}_{\in\mathbb{R}^{\sharp\sigma,d+1}} \underbrace{\left(G(t_j^x, t_k^y)\right)_{j,k=0,\dots,d}}_{\in\mathbb{R}^{d+1,d+1}} \underbrace{\left(L_k^y(y_j)\right)_{\substack{k=0,\dots,d\\ j=j_1,\dots,j_2}}}_{\in\mathbb{R}^{d+1,\sharp\mu}}}_{\text{rank}\le d+1} \ .$$

$\widetilde{\mathbf{M}}_{|\sigma\times\mu} =$ 

R. Hiptmair

rev 38355, October 28, 2009

Application of partitioning

$$\{x_1,\dots,x_n\}\times\{y_1,\dots,y_m\} = \left(\bigcup_{(\sigma,\mu)\in P^{\mathrm{near}}} \sigma\times\mu\right) \cup \left(\bigcup_{(\sigma,\mu)\in P^{\mathrm{far}}} \sigma\times\mu\right) .$$

Algorithm: Analysis of $\mathbf{f}=\mathbf{M}\mathbf{c}$, $\mathbf{M}$ as (11.1.1)

$$f_i = \sum_{j=1}^{m} G(x_i,y_j)c_j = \underbrace{\boxed{\sum_{j\in P^{\mathrm{near}}(i)} G(x_i,y_j)c_j}}_{\text{Near field contribution}} + \underbrace{\boxed{\sum_{\substack{\sigma\in T_x\\ x_i\in\sigma}} \sum_{\substack{\mu\in T_y\\ (\sigma,\mu)\in P^{\mathrm{far}}}} \sum_{\substack{1\le j\le m\\ y_j\in\mu}} G(x_i,y_j)c_j}}_{\text{Far field contribution}} .$$

Near field coupling indices $P^{\mathrm{near}}(i) := \{j\in\{1,\dots,m\}\colon (\{x_i\}\times\{y_j\})\in P^{\mathrm{near}}\}$

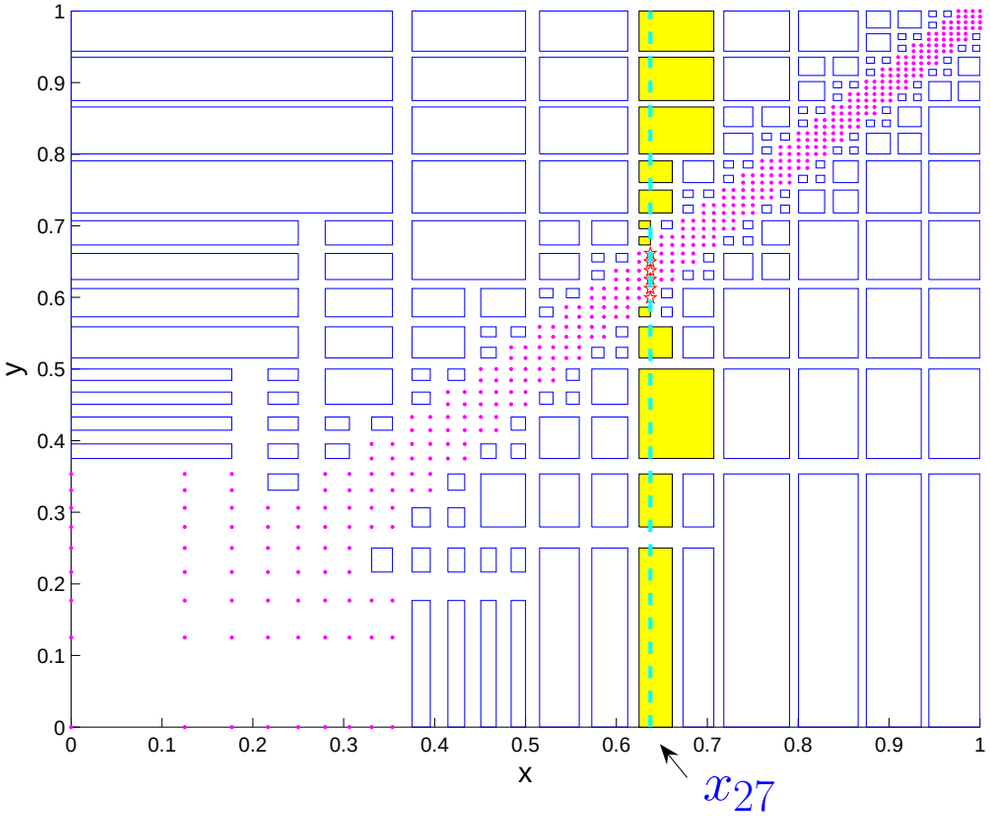


Illustration:

`x = 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

$$\approx \sum_{j \in P^{\text{near}}(i)} G(x_i, y_j) c_j + \sum_{\substack{\sigma \in T_x \\ x_i \in \sigma}} \sum_{\substack{\mu \in T_y \\ (\sigma,\mu) \in P^{\text{far}}}} \sum_{\substack{1 \le j \le m \\ y_j \in \mu}} \sum_{l=0}^{d} \sum_{k=0}^{d} G(t_l^\sigma, t_k^\mu) L_l^\sigma(x_i) L_k^\mu(y_j) c_j$$

$$\approx \sum_{j \in P^{\text{near}}(i)} G(x_i, y_j) c_j + \sum_{\substack{\sigma \in T_x \\ x_i \in \sigma}} \sum_{\substack{\mu \in T_y \\ (\sigma,\mu) \in P^{\text{far}}}} (\mathbf{V}_\sigma \mathbf{X}_{\sigma,\mu} \mathbf{V}_\mu^T \mathbf{c}_{|\mu})_i \ ,$$

R. Hiptmair

rev 38355, October 28, 2009

with

$$t_l^\sigma, t_k^\mu := \text{Chebyshev nodes (9.2.12) in } \Gamma(\sigma), \Gamma(\mu) \ , \quad l = 0, \dots, d \ ,$$

$$\mathbf{X}_{\sigma,\mu} := \left(G(t_l^\sigma, t_k^\mu)\right)_{l,k=0}^{d} \in \mathbb{R}^{d+1,d+1} \ , \tag{11.4.1}$$

$$\mathbf{V}_\sigma := \left(L_l^\sigma(x_i)\right)_{\substack{i:\, x_i \in \sigma \\ l=0,\dots,d}} \in \mathbb{R}^{\sharp\sigma,d+1} \ , \tag{11.4.2}$$

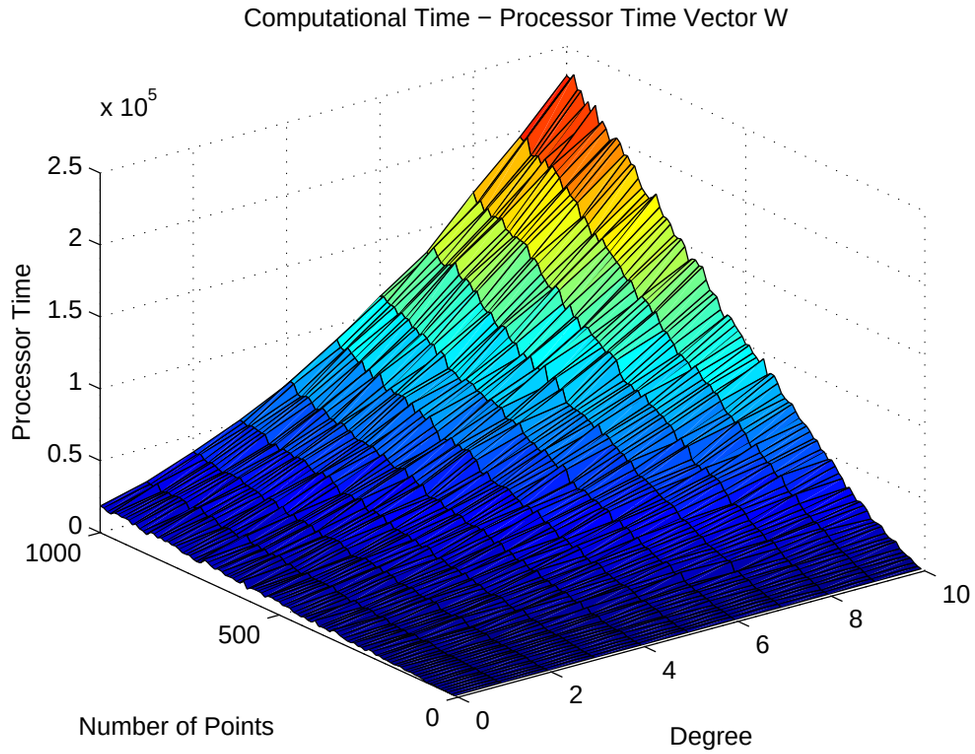
$$\mathbf{V}_\mu := \left(L_k^\mu(y_j)\right)_{\substack{j:\, y_j \in \mu \\ k=0,\dots,d}} \in \mathbb{R}^{\sharp\mu,d+1} \ , \tag{11.4.3}$$

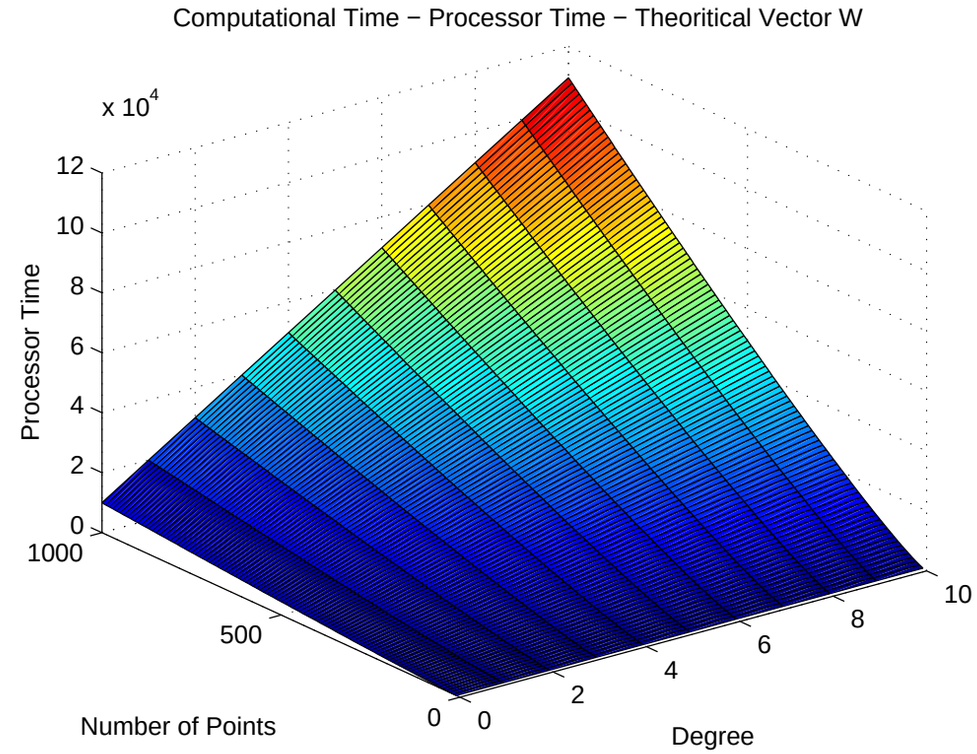
Analysis of Complexity (under the assumption $n \approx m$):

①: Trick: Calculate on above assumption

Compute $\mathbf{w}_\mu := \mathbf{V}_\mu^T \mathbf{c}_{|\mu} \in \mathbb{R}^{d+1}$, $\mu \in T_y$ ➞ $O(n)$ Operations on every level of T_y

Computational Effort $O((d+1)n\log_2 n)$, Memory Required $O((d+1)n\log_2 n)$





R. Hiptmair

rev 38355, October 28, 2009

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++: Time.txt

OutputFile from MATLAB : Time_Vector_W.eps, Time_Theoritical_Vector_W.eps, Time_Chebyshev.eps, Time_Theoritical_Chebyshev.eps, Time_Far.eps, Time_Theoritical_Far.eps, Time_Near.eps, Time_Theoritical_Near.eps, Time_All.eps, Time_Theoritical_All.eps

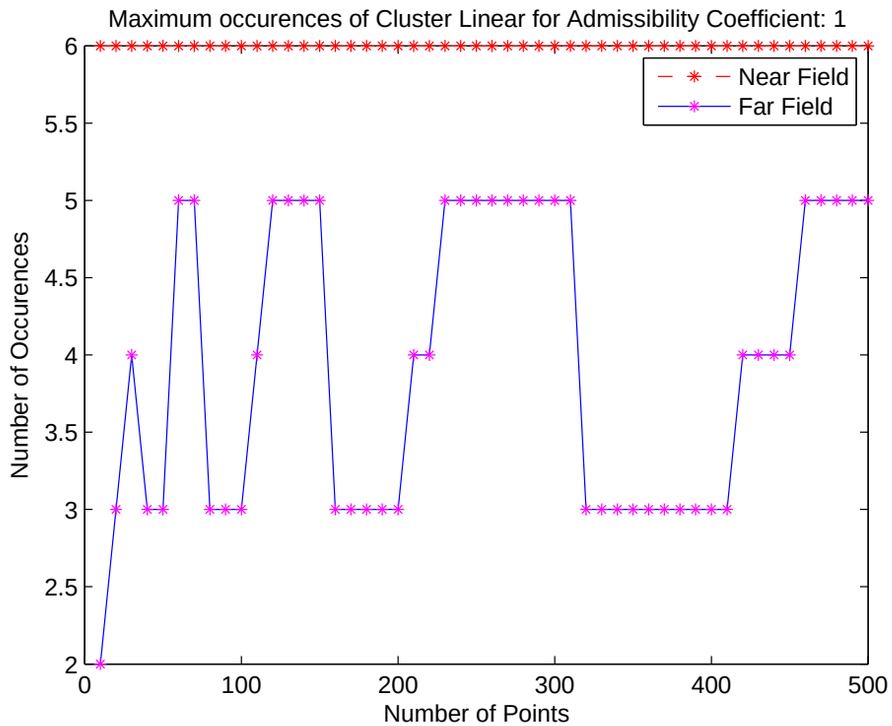
1) Run main.cpp, input as follows : Choice - 2, NumberofPts - 10 1000 10, Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1 Degree - 0 10 1, Admissibility Coefficient - 0.5 0.5 1.0

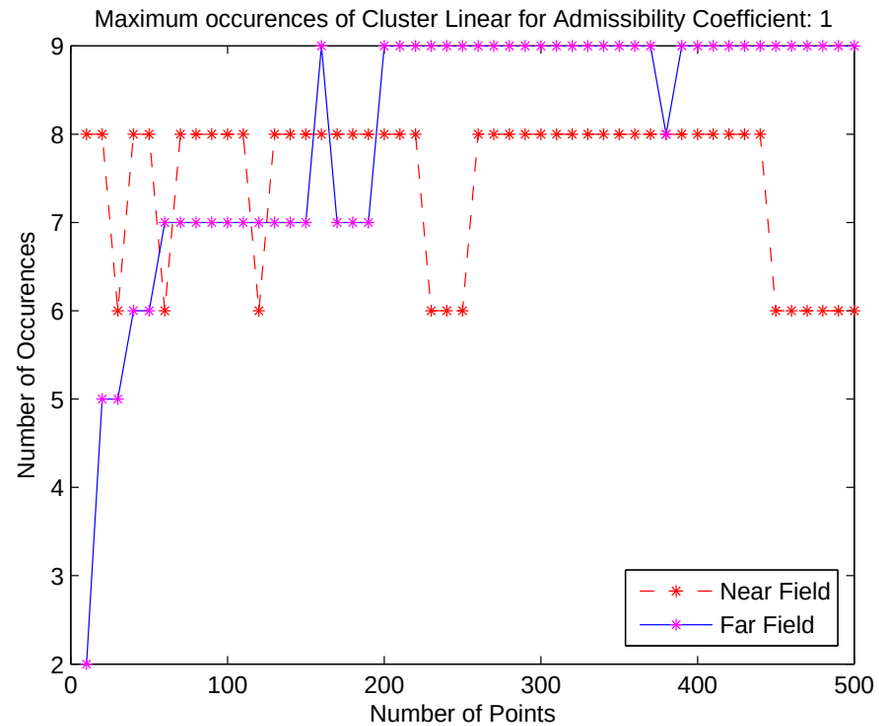
2) Run time_mesh.m and time_mesh_theoritical from Matlab

②: Compute $\mathbf{X}_{\sigma,\mu}$ as in (11.4.1), $\sigma \in T_x$, $\mu \in T_y$, $(\sigma,\mu) \in P^{\text{far}}$.

Uniform distribution of points $\max\{\sharp\{\mu \in T_y : (\sigma,\mu) \in P^{\text{far}}\}, \sigma \in T_x\} = O(1)$

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:
0 (argument here is for specifing the index of Admissibility Coefficient.)

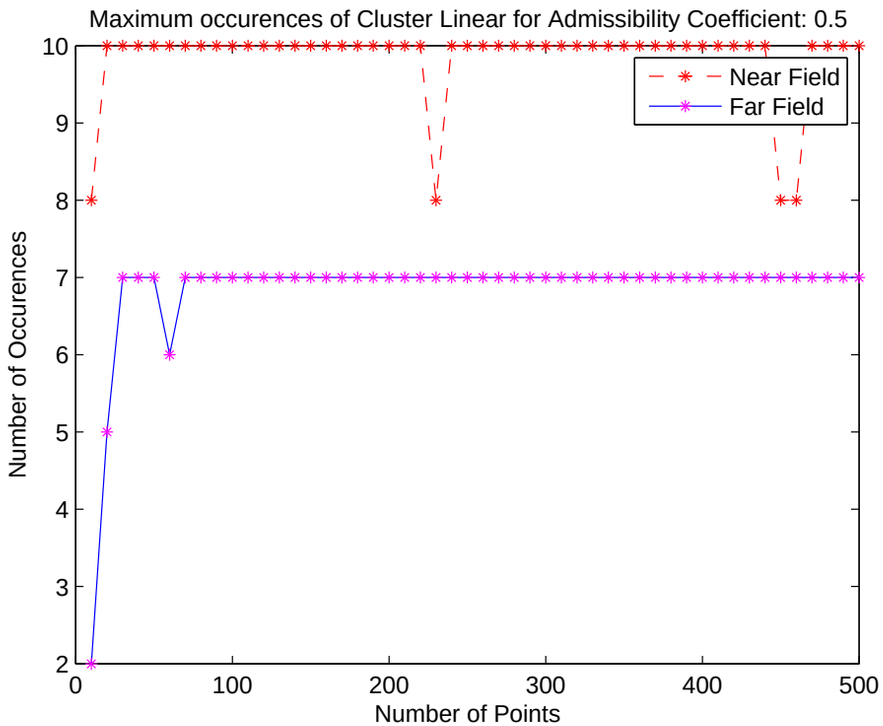
`x = sqrt(0:1/n:(1-1/n))`, $\eta = 1$

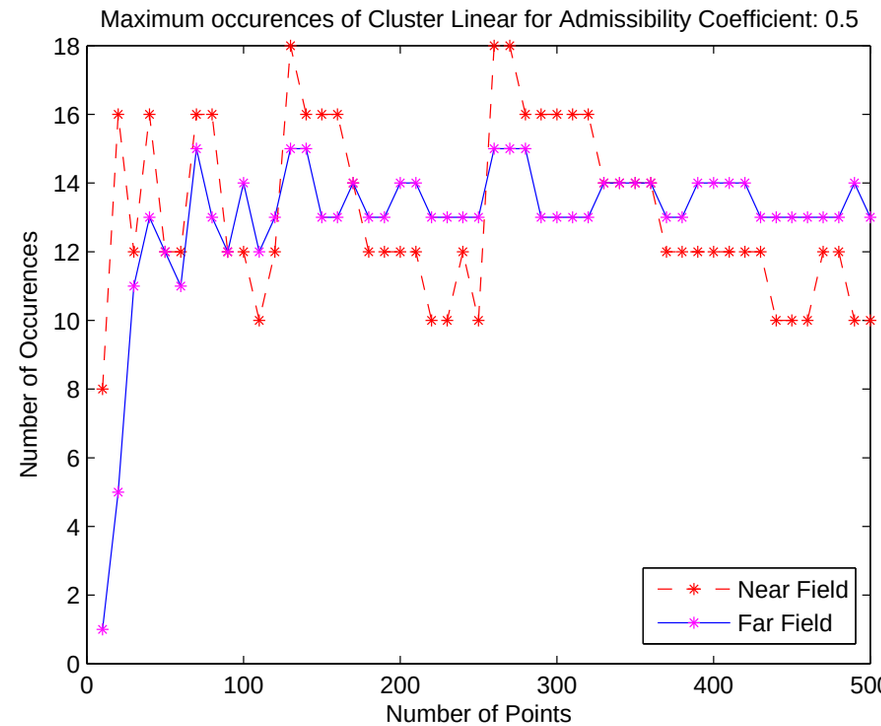
In file changeable.c
a) put the admissibility condition in admissible_ClusterPair
b) put kernel function in kernel_function
c) put SQRT as return values in get_iOperation
1) Run main.cpp, input as follows
Choice - 2, NumberofPts - 10 500 10,
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1
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:
0 (argument here is for specifing the index of Admissibility Coefficient.)

Uniform distribution of x_i, y_j: for every $\sigma \in T_x$

$$\sharp\{\mu \in T_y\colon (\sigma,\mu) \in \texttt{Pfar}\} = O(1)\ ,$$

each cluster contributes only a small number of partitions of rectangle





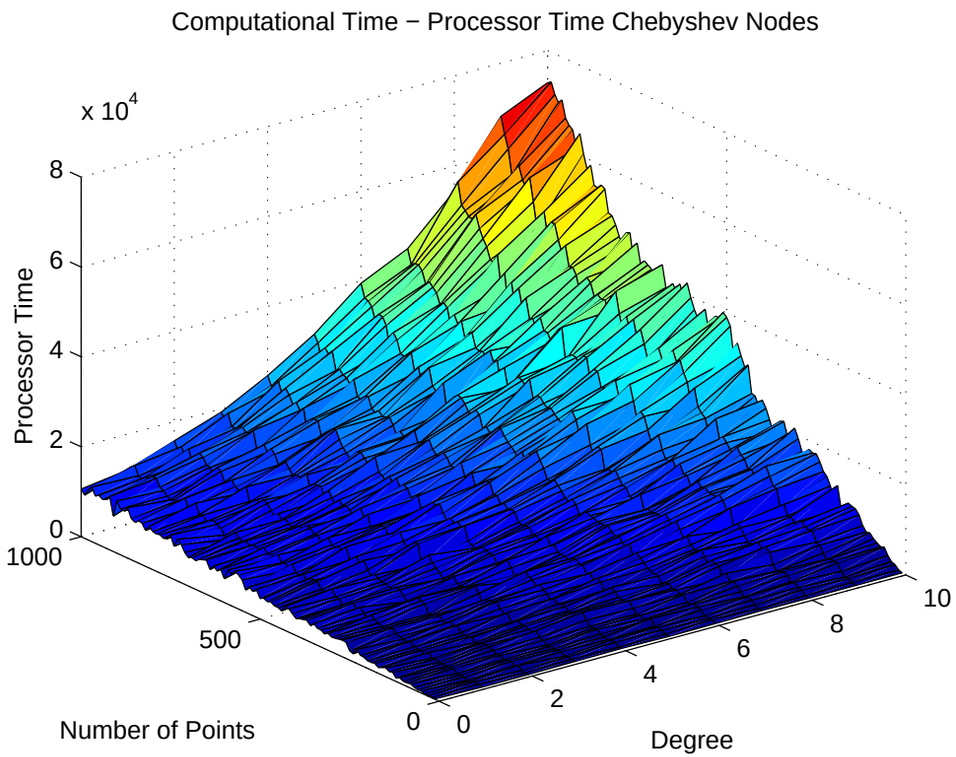
`x = (0:1/n:(1-1/n));`, $\eta = 0.5$

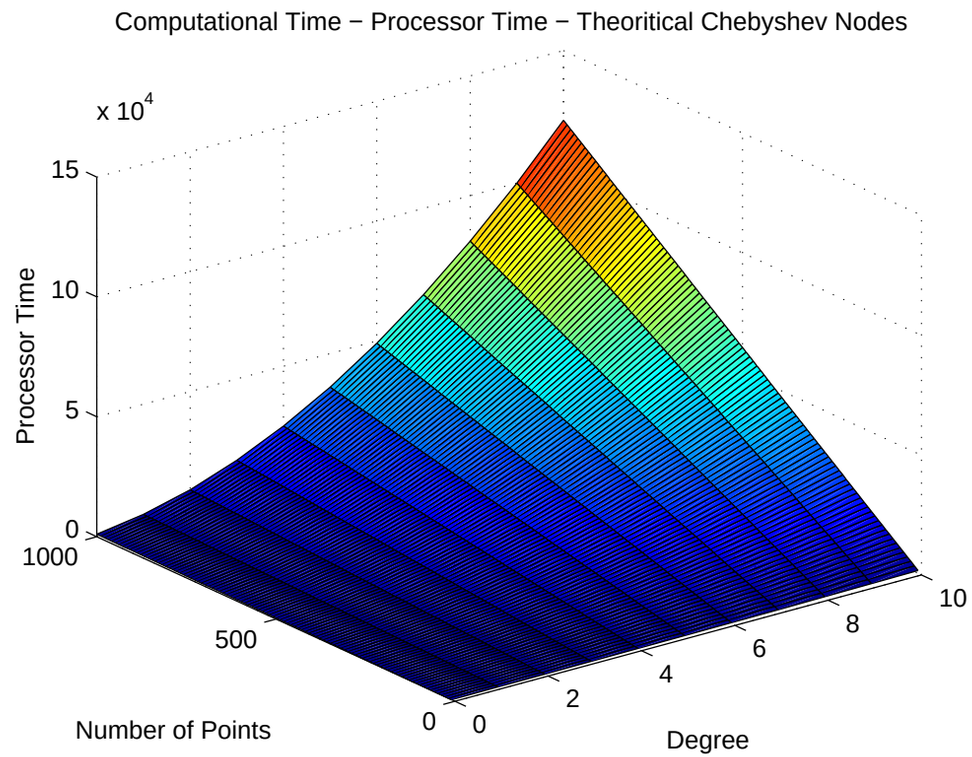
In file changeable.c
a) put the admissibility condition in admissible_ClusterPair
b) put kernel function in kernel_function
c) put NONE as return values in get_iOperation
1) Run main.cpp, input as follows
Choice - 2, NumberofPts - 10 500 10,
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1
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:
0 (argument here is for specifing the index of Admissibility Coefficient.)

`x = 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:
0 (argument here is for specifing the index of Admissibility Coefficient.)

Computational Effort for step ②: $O(n \log_2 n(d+1)^2)$, Memory Required $O(n \log_2 n(d+1)^2)$





R. Hiptmair

rev 38355, October 28, 2009

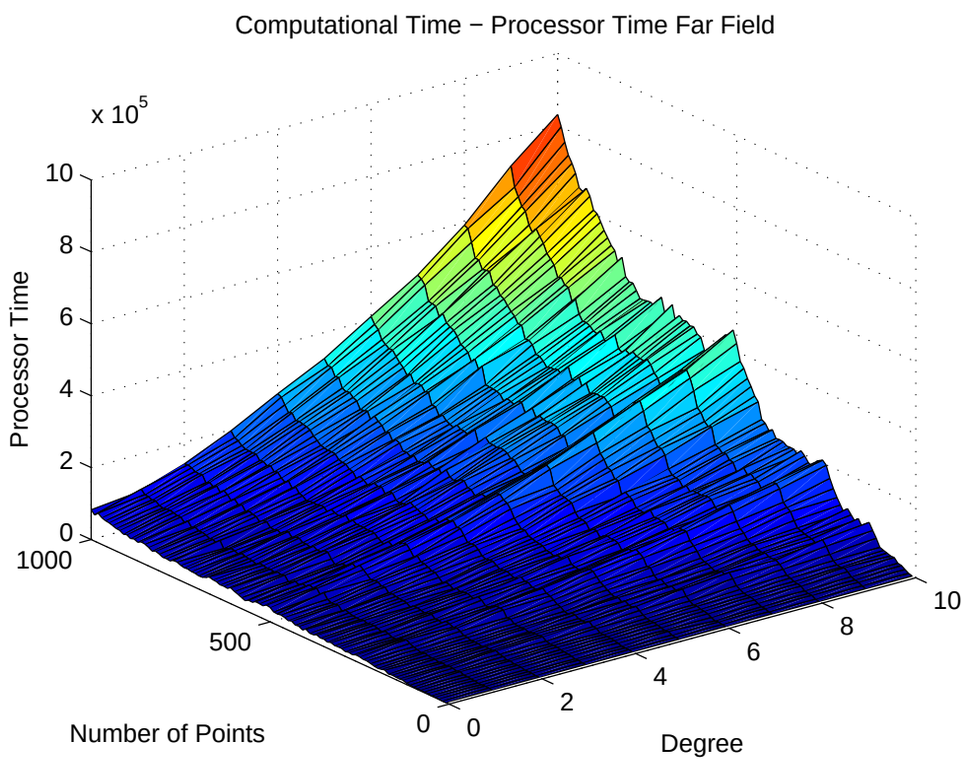
③: Far field analysis $\hat{=}$ Sum $\displaystyle\sum_{\substack{\sigma\in T_x\\ x_i\in\sigma}}\ \sum_{\substack{\mu\in T_y\\ (\sigma,\mu)\in P^{\mathrm{far}}}} \mathbf{V}_\sigma \mathbf{X}_{\sigma,\mu}\mathbf{w}_\mu$

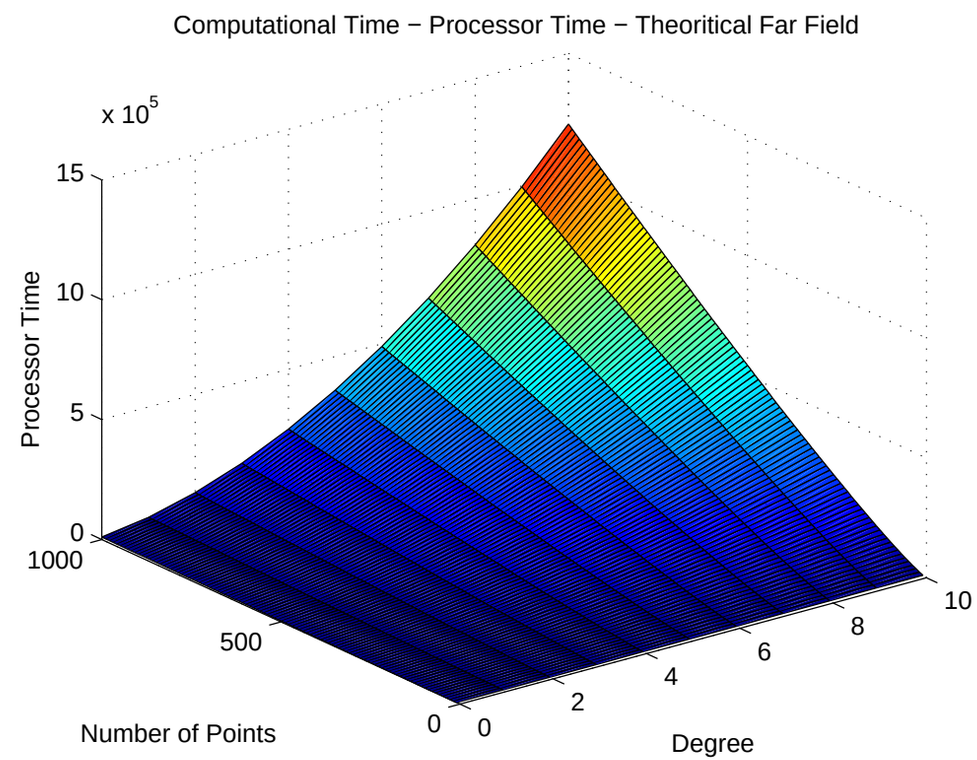
for each $\sigma \in T_x$ { Compute $\mathbf{V}_\sigma$; $\mathbf{s} := 0$

for each $\mu \in T_y$, $(\sigma,\mu) \in P^{\mathrm{far}}$ { $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{X}_{\sigma,\mu}\mathbf{w}_\mu$ }

$\mathbf{f}_{|\sigma} \leftarrow \mathbf{f}_{|\sigma} + \mathbf{V}_\sigma \mathbf{s}$ }

Computational Effort $O(n\log_2 n(d+1)^2)$, Memory Required $O(n(d+1))$

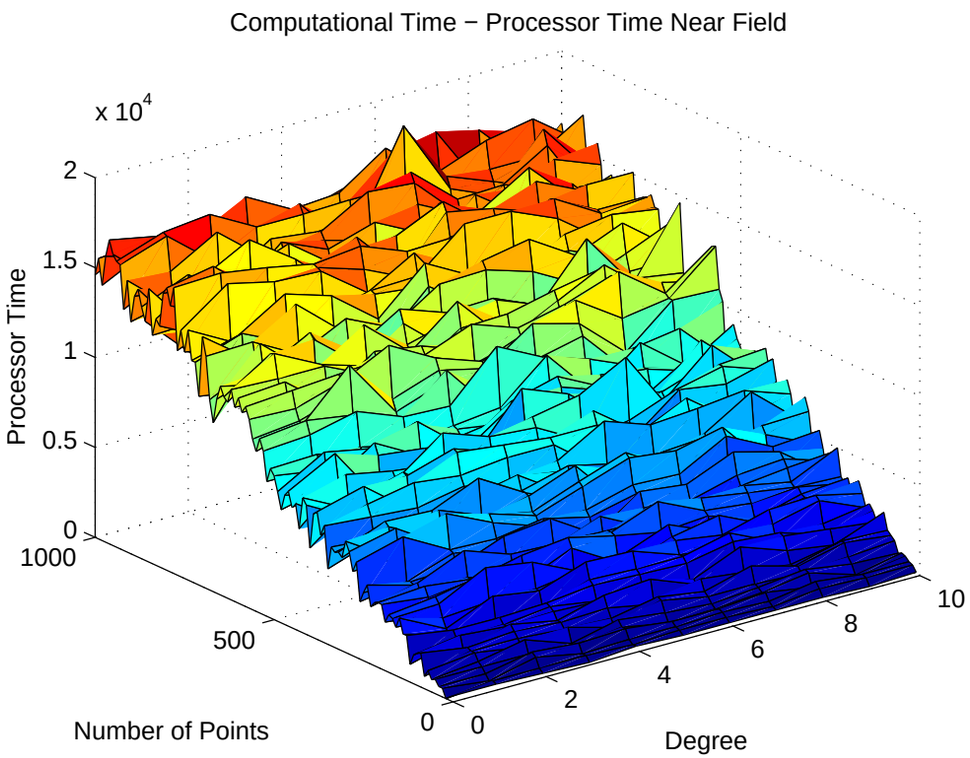


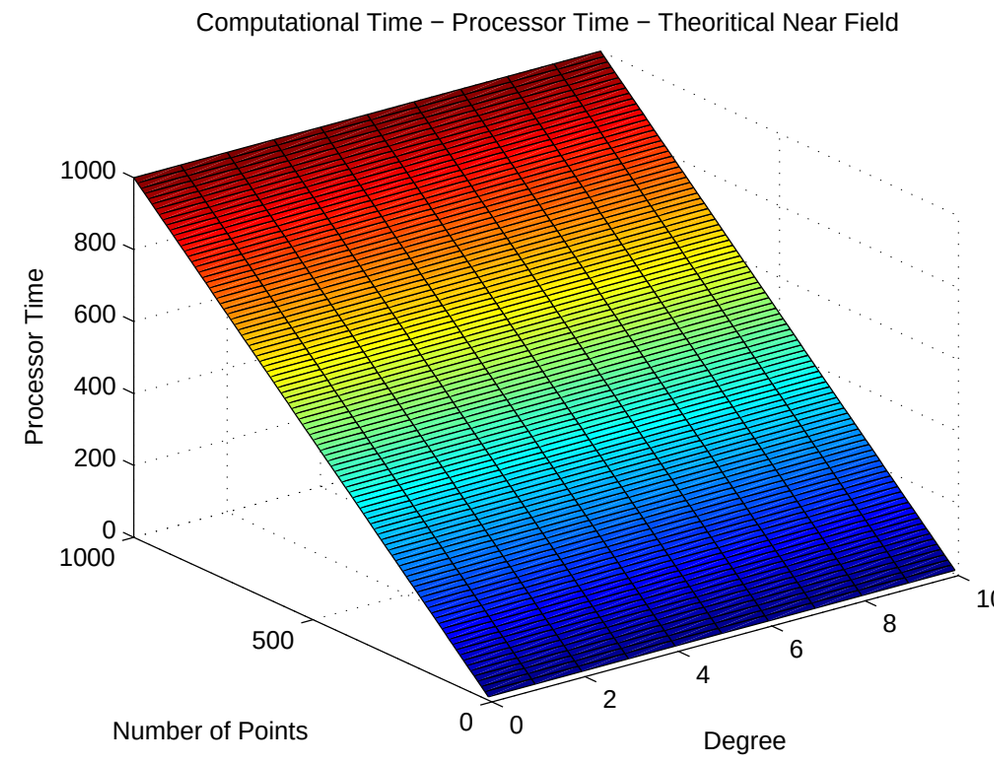


④: Near field Computation

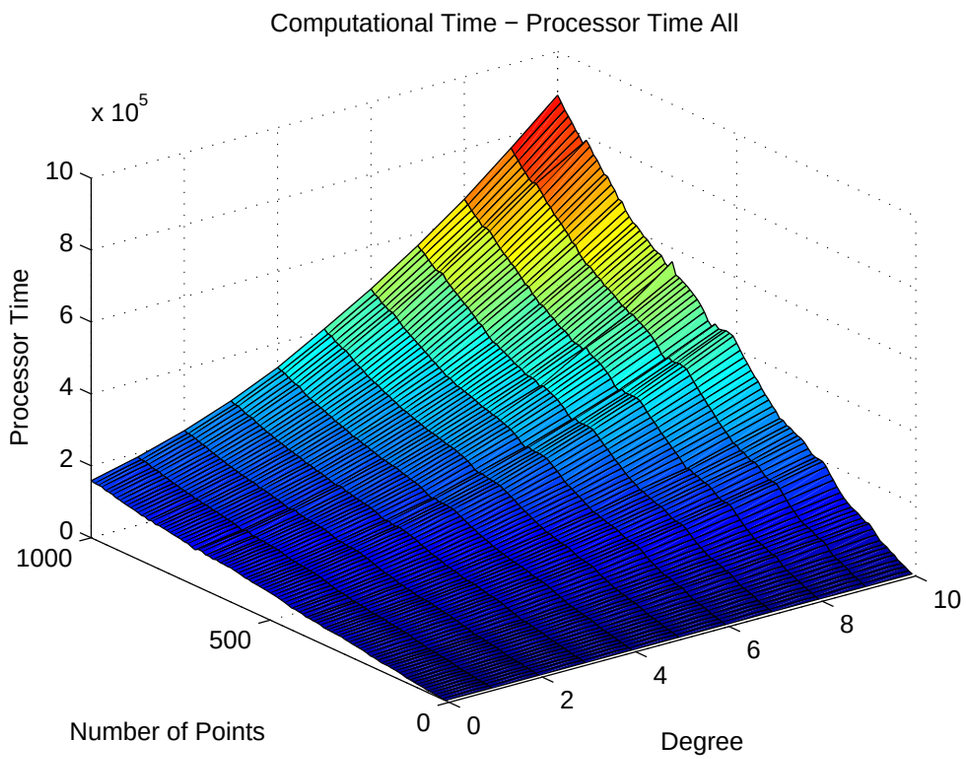
for each $(\sigma,\mu) \in P^{\text{near}}$ { **for each** i: $x_i \in \sigma$ { $\mathbf{f}_i \leftarrow \mathbf{f}_i + \sum_{j:\, y_j \in \mu} G(x_i, y_j) c_j$ } }

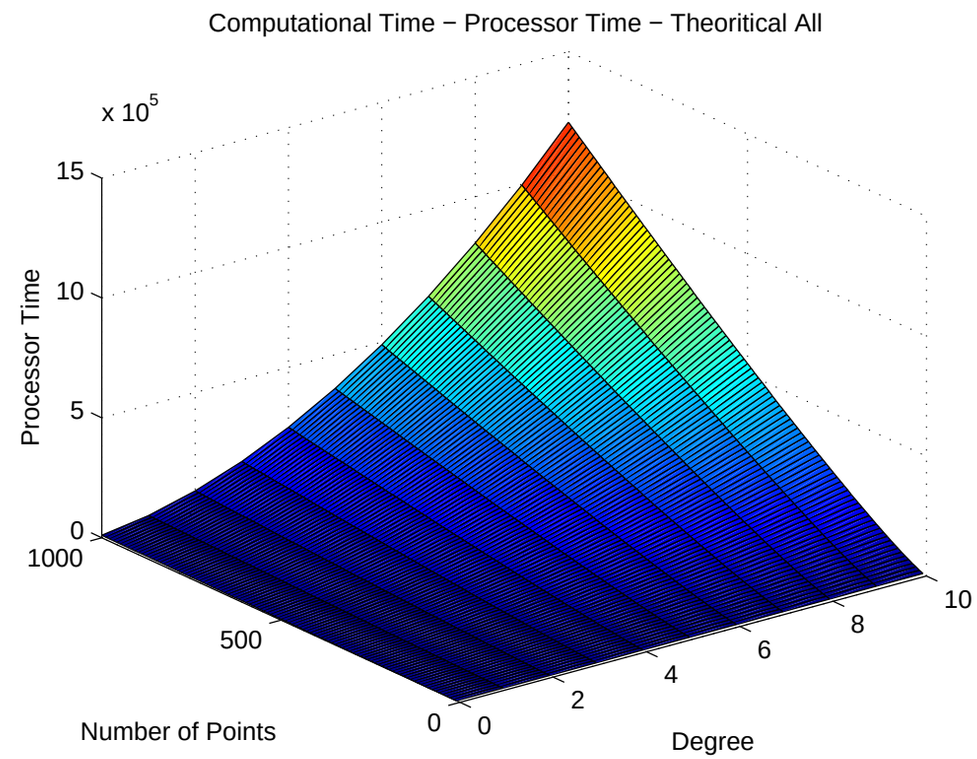
Computational Effort $O(n)$





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 $\{(x,y)\colon x \neq y\}$ then

Cluster approximation inherits exponential convergence from Chebyshev interpolation

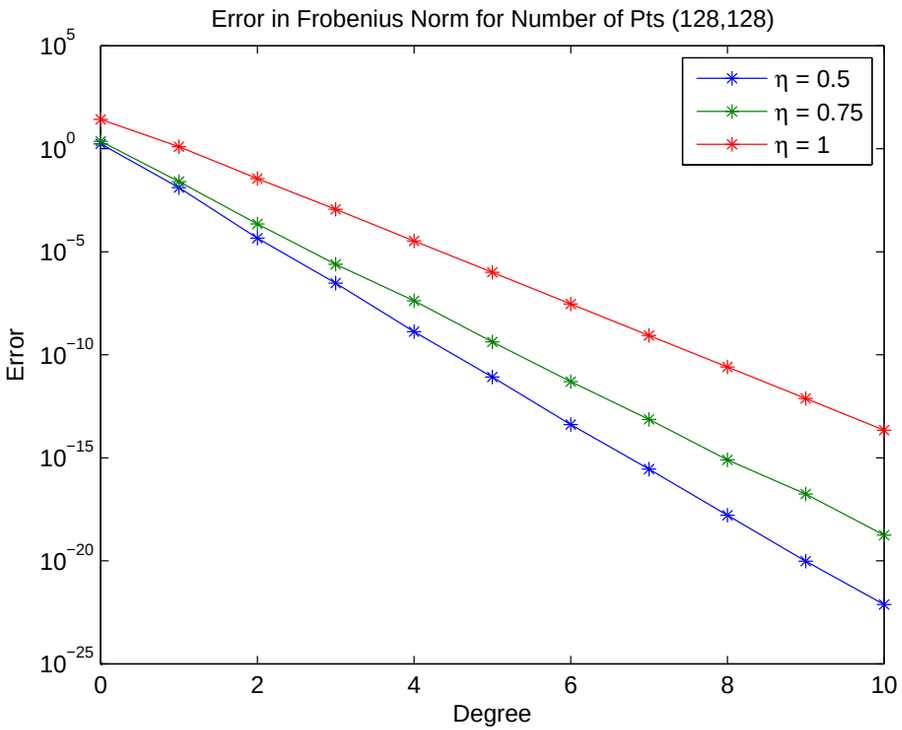
$\triangle$

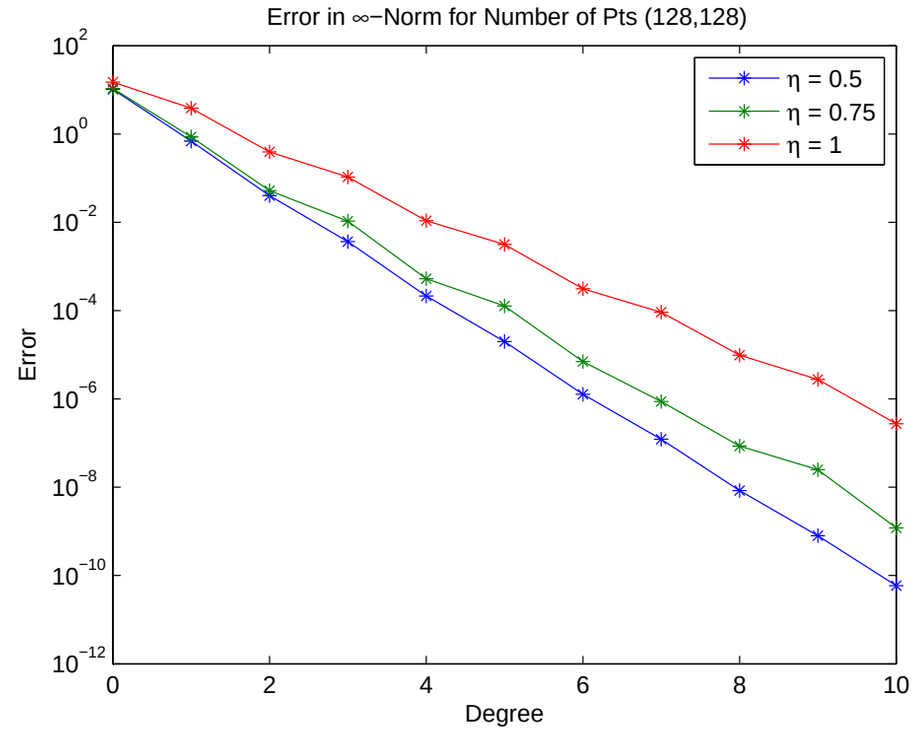
R. Hiptmair

rev 38355, October 28, 2009

Example 11.4.6 (Convergence of clustering approximation with collocation matrix).

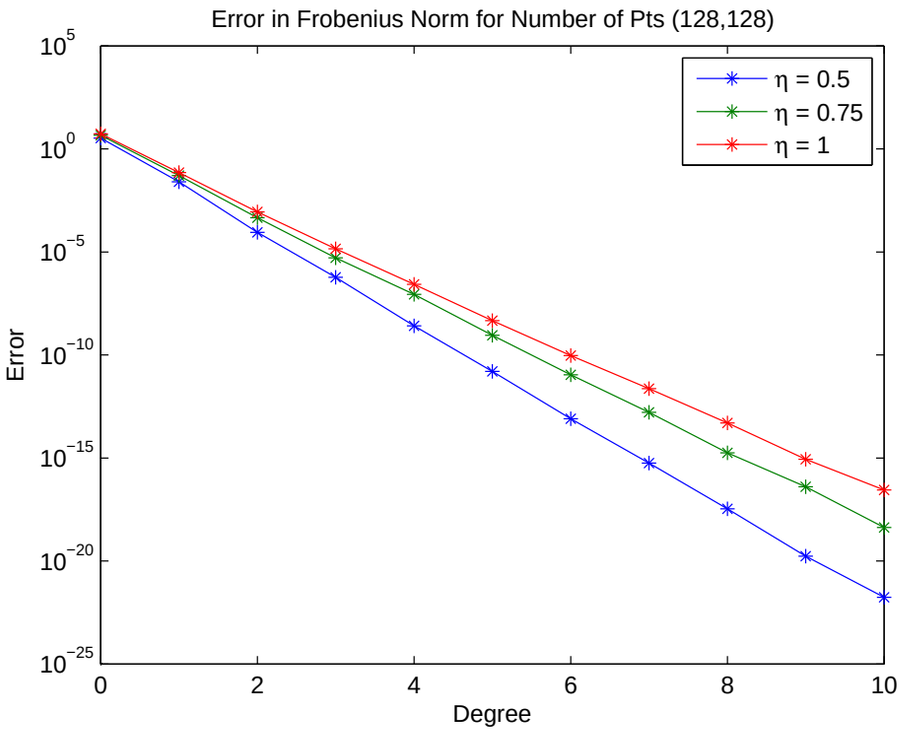
- `x = 0:1/128:1;`, $G(x,y) = |x-y|^{-1}$ for $x \neq y$, $G(x,x) = 0$.

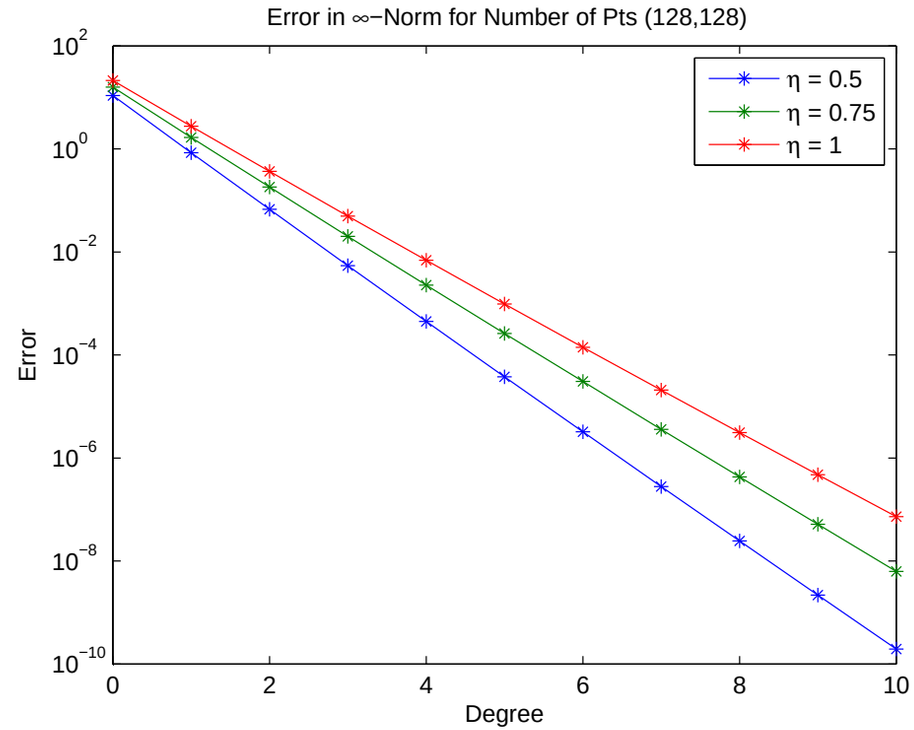




In file changeable.c
a) put the admissibility condition in admissible_ClusterPair
b) put kernel function in kernel_function
c) put NONE as return values in get_iOperation
1) Run main.cpp, input as follows
Choice - 2, NumberofPts - 128 128 1,
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1
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 specifing the index of Number of Pts.)

- `x = sqrt(0:1/128:1);`, $G(x,y) = |x-y|^{-1}$ for $x \neq y$, $G(x,x) = 0$.

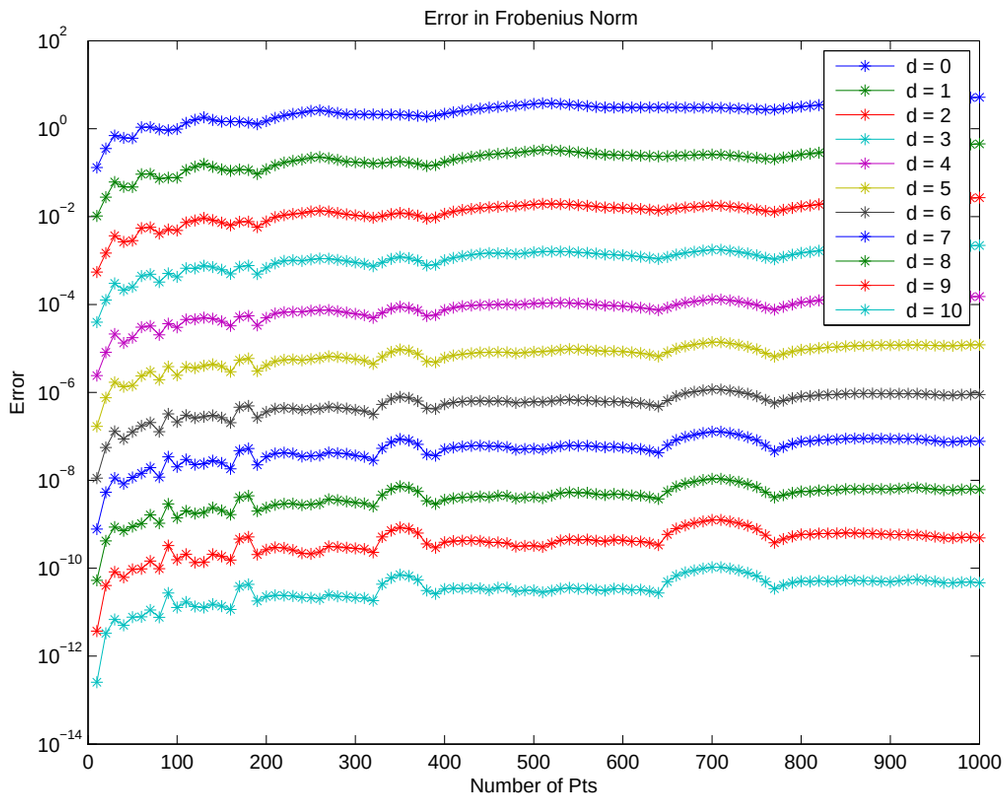




In file changeable.c
a) put the admissibility condition in admissible_ClusterPair
b) put kernel function in kernel_function
c) put SQRT as return values in get_iOperation
1) Run main.cpp, input as follows
Choice - 2, NumberofPts - 128 128 1,
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1
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 specifing 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
OutputFile from MATLAB : Error_in_Frobenius_Norm_Degree.eps
2) Run error_plot_Degree.m from Matlab

R. Hiptmair

rev 38355, October 28, 2009

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

$$c_k := p(t_k)\ , \quad j=0,\dots,n-1 \quad \text{for} \quad \underbrace{p(t) := \sum_{j=-m+1}^{m} \alpha_j e^{-2\pi i j t}}_{\text{Trigonometric Polynomial}} .$$

Discrete Fourier transformation (DFT) $\rightarrow$ Section 8.2:

$$f_l := \sum_{j=-m+1}^{m} \alpha_j e^{-\frac{2\pi i}{n} jl} \quad \overset{\text{Lemma 8.2.10}}{\Leftrightarrow} \quad \alpha_j = \frac{1}{n}\sum_{l=-m+1}^{m} f_l e^{\frac{2\pi i}{n} lj}$$

FFT: f_l, $l=-m+1,\dots,m$, calculated with computational effort $O(n\log_2 n)$

$$\begin{aligned} c_k &= \sum_{j=-m+1}^{m} \alpha_j e^{2\pi i j t_k} = \sum_{j=-m+1}^{m} \frac{1}{n}\Big(\sum_{l=-m+1}^{m} f_l e^{\frac{2\pi i}{n} lj}\Big) e^{-2\pi i j t_k} \\ &= \frac{1}{n}\sum_{l=-m+1}^{m} f_l \sum_{j=-m+1}^{m} e^{-2\pi i j (t_k - l/n)} \\ &= \frac{1}{n}\sum_{l=-m+1}^{m} f_l e^{-2\pi i (t_k-l/n)(-m+1)} \frac{1-e^{-2\pi i n (t_k - l/n)}}{1-e^{-2\pi i (t_k-l/n)}} \end{aligned}$$

R. Hiptmair

rev 38355, October 28, 2009

$$= \frac{1}{n} \sum_{l=-m+1}^{m} f_l e^{-\pi i t_k} \sin(\pi n t_k) \frac{1}{\sin(\pi(t_k - l/n))} (-1)^l e^{-\pi i l/n} \,.$$

$$\mathbf{c} = \operatorname{diag}\left(\frac{\sin(\pi n t_k)}{e^{\pi i t_k}}\right)_{k=-m+1}^{m} \mathbf{M} \operatorname{diag}\left((-1)^l e^{-\pi i l/n}\right)_{l=-m+1,\dots,m} \mathbf{f} \,.$$

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 **!** $\rightarrow$ USFFT

$\diamond$

Remark 11.4.8. Clustering approximation example for fast approximative implementation of algorithms of numerical linear algebra ($\rightarrow$ Chapter **??**) ➜ Trend in numerical linear algebra **?**

Problem	Exact/direct Method	Approximate/iterative Method
Linear equation systems	Gaussian Elimination	CG-like iterative solvers $\rightarrow$ Sect. **??**
Eigenvalue problem	Transformation methods	Krylov-Subspace method $\rightarrow$ Sect. **??**
Collocations matrix$\times$Vector	BLAS (SAXPY)	Clustering techniques

$\triangle$

12 Single Step Methods

12.1 Initial value problems (IVP) for ODEs

Acronym: **ODE** = **o**rdinary **d**ifferential **e**quation

R. Hiptmair

rev 38355, May 3, 2011

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

Model: autonomous logistic differential equations [63, Ex. 5.7.2]

$$\dot{y} = f(y) := (\alpha - \beta y)\, y \tag{12.1.2}$$

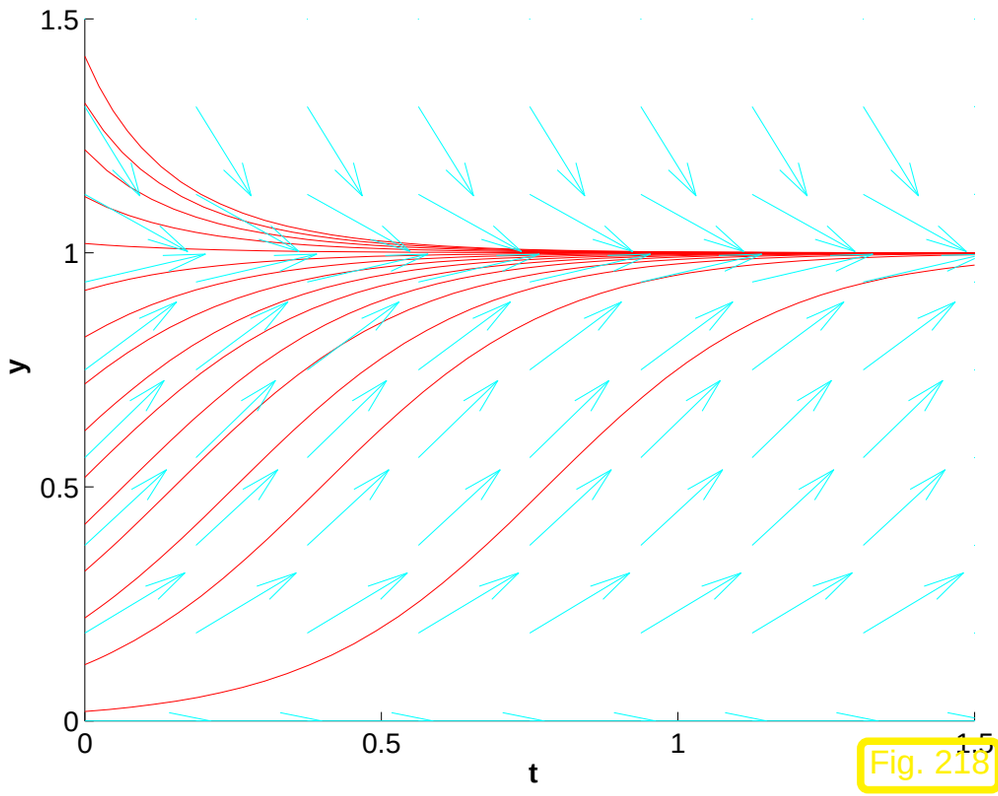


Fig. 218

Solution for different $y(0)$ ($\alpha, \beta = 5$)

By separation of variables

➥ solution of (12.1.2)

for $y(0) = y_0 > 0$

$$y(t) = \frac{\alpha y_0}{\beta y_0 + (\alpha - \beta y_0)\exp(-\alpha t)}\,, \tag{12.1.3}$$

for all $t \in \mathbb{R}$

◇

Example 12.1.4 (Predator-prey model). [2, Sect. 1.1], [31, Sect. 1.1.1], [35, Ch. 60], [13, Ex. 11.3]

Model: autonomous Lotka-Volterra ODE:

$$\begin{aligned} \dot{u} &= (\alpha - \beta v)u \\ \dot{v} &= (\delta u - \gamma)v \end{aligned} \quad \leftrightarrow \quad \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \text{with} \quad \mathbf{y} = \begin{pmatrix} u \\ v \end{pmatrix}, \quad \mathbf{f}(\mathbf{y}) = \begin{pmatrix} (\alpha - \beta v)u \\ (\delta u - \gamma)v \end{pmatrix} . \tag{12.1.5}$$

population sizes:

$u(t) \to$ no. of prey at time t,

$v(t) \to$ no. of predators at time t

---

vector field $\mathbf{f}$ for Lotka-Volterra ODE $\triangleright$

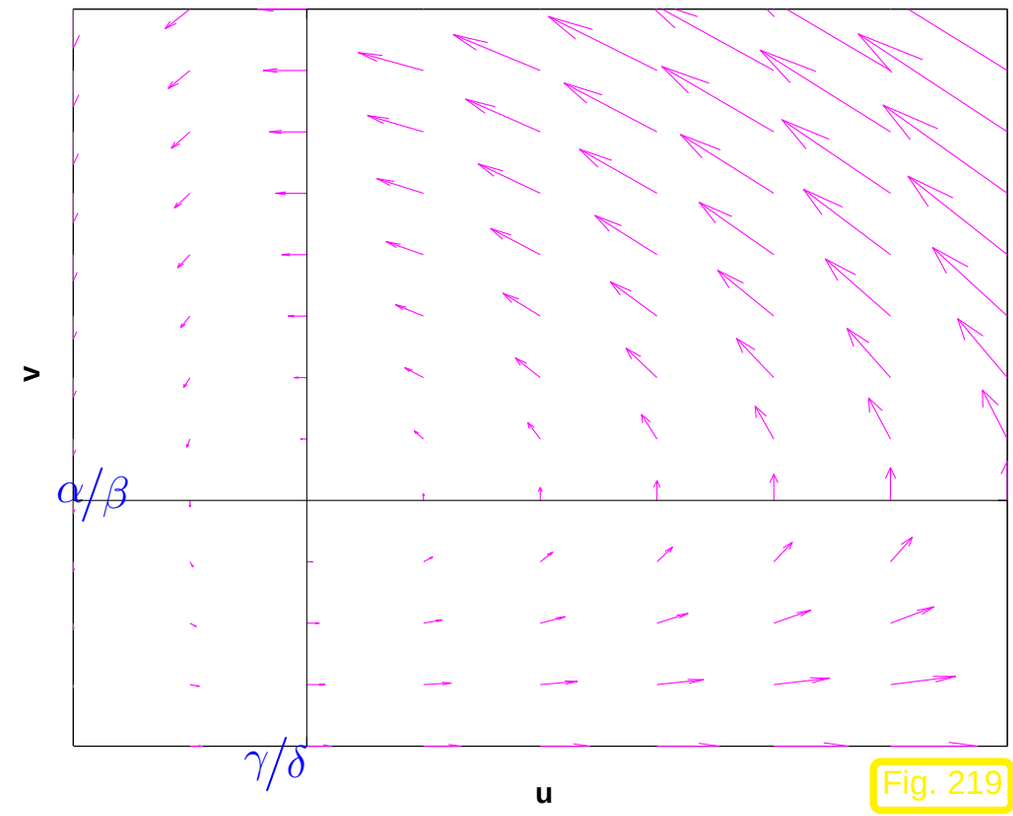


Fig. 219

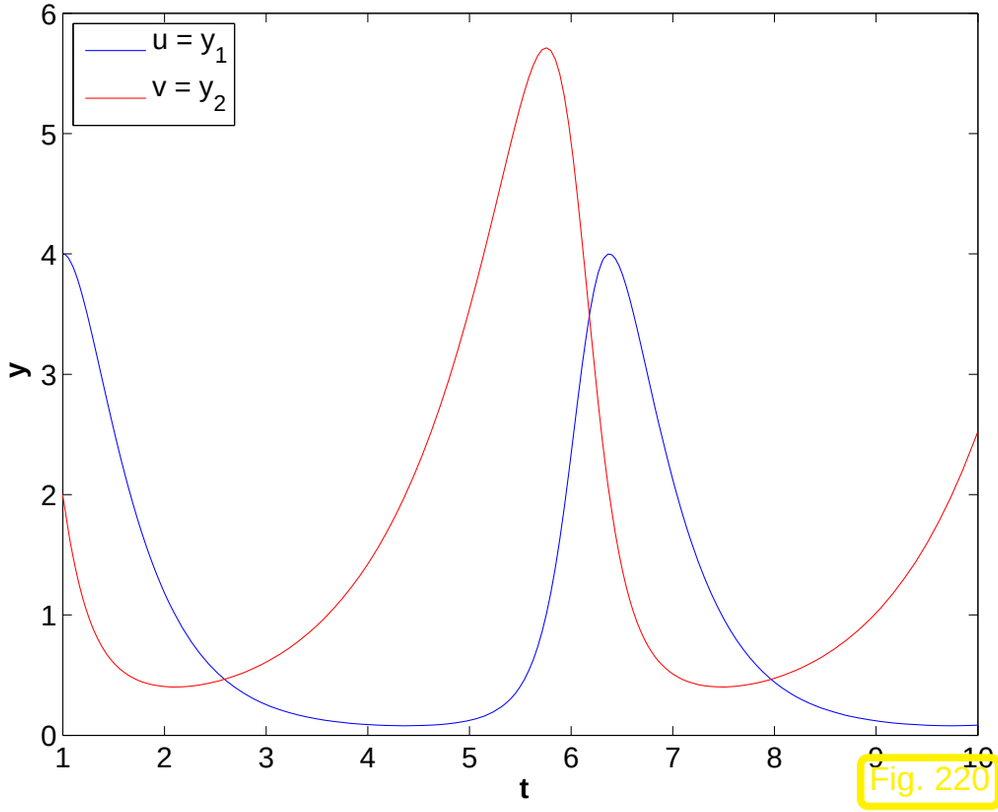


Fig. 220

Solution $\binom{u(t)}{v(t)}$ for $\mathbf{y}_0 := \binom{u(0)}{v(0)} = \binom{4}{2}$

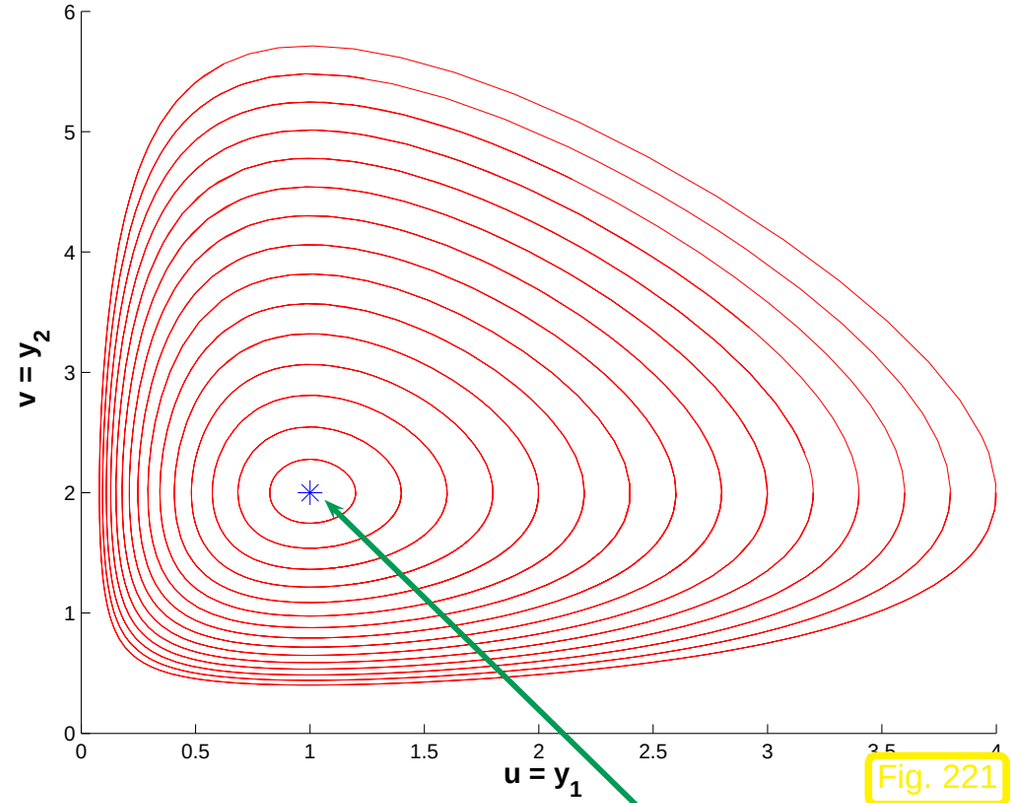


Fig. 221

Solution curves for (12.1.5)

stationary point

◇

Example 12.1.6 (Heartbeat model). $\rightarrow$ [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:

$$\begin{aligned} \dot{l} &= -(l^3 - \alpha l + p)\,, \\ \dot{p} &= \beta l\,, \end{aligned} \tag{12.1.7}$$

with parameters: $\alpha \hat{=}$ pre-tension of muscle fiber

$\beta \hat{=}$ (phenomenological) feedback parameter

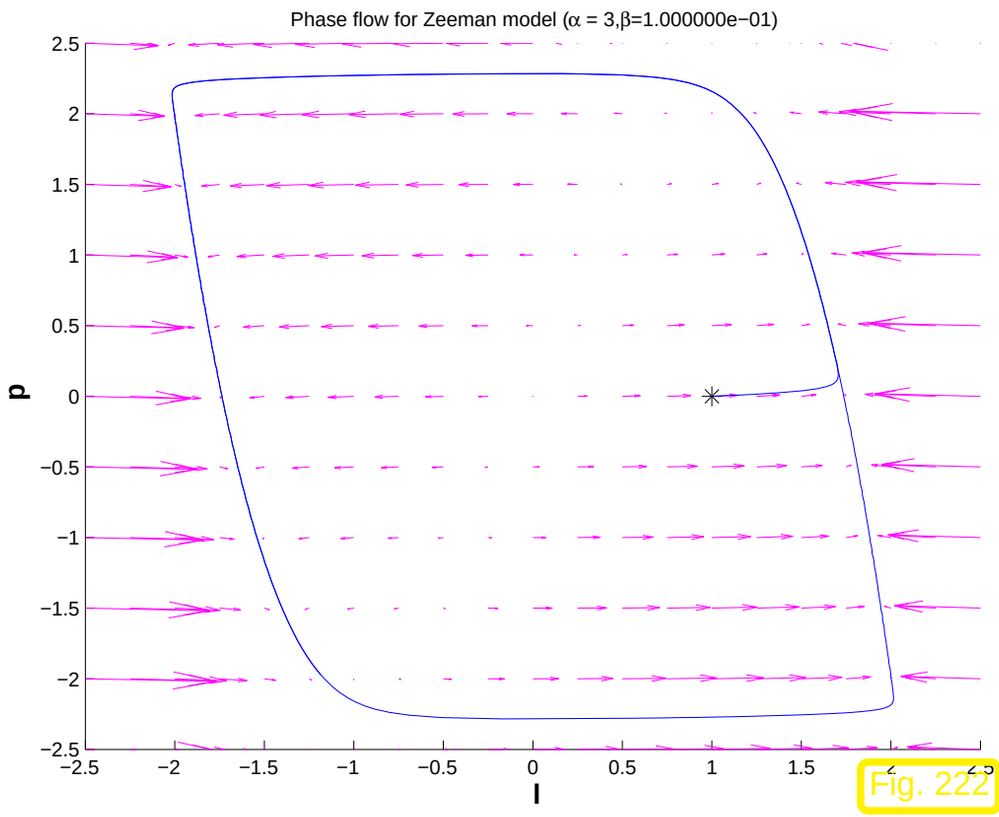


Fig. 222

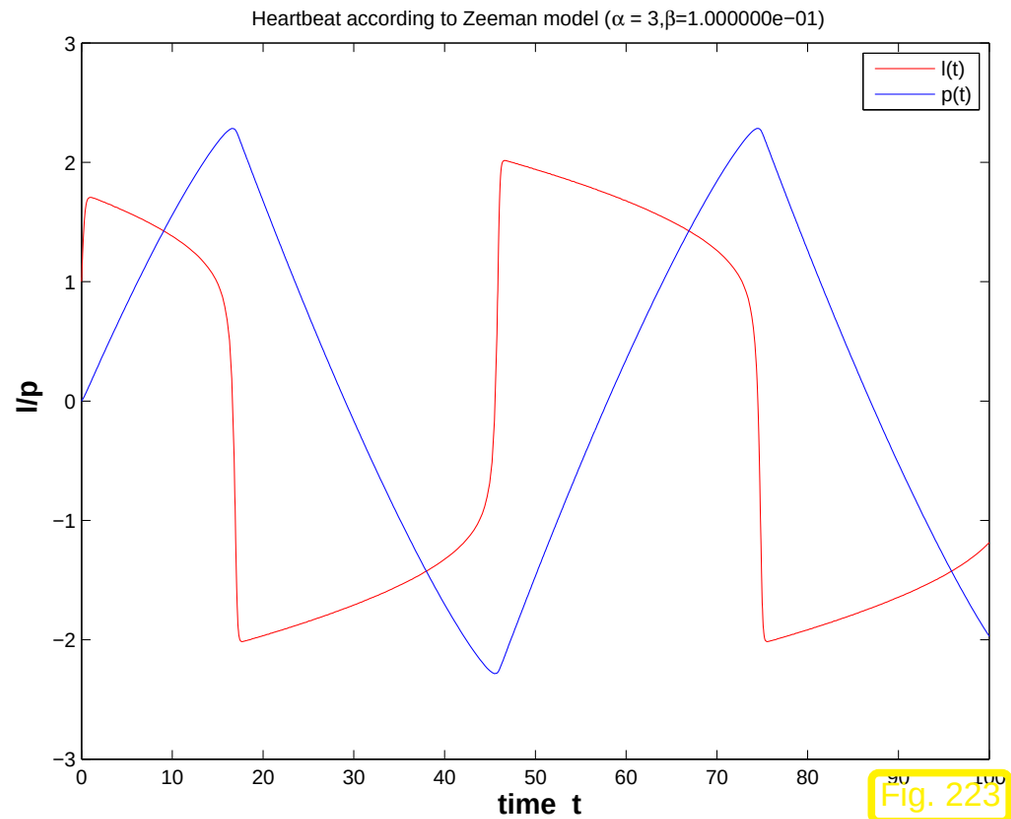


Fig. 223

R. Hiptmair

rev 38355, May 3, 2011

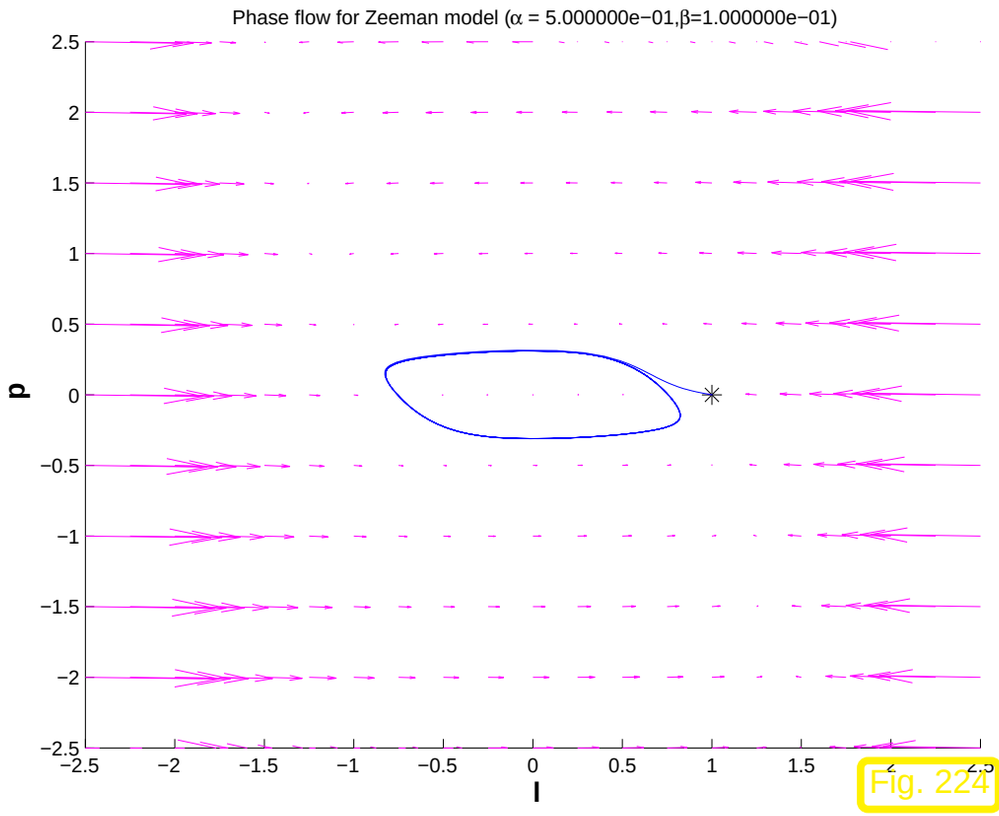


Fig. 224

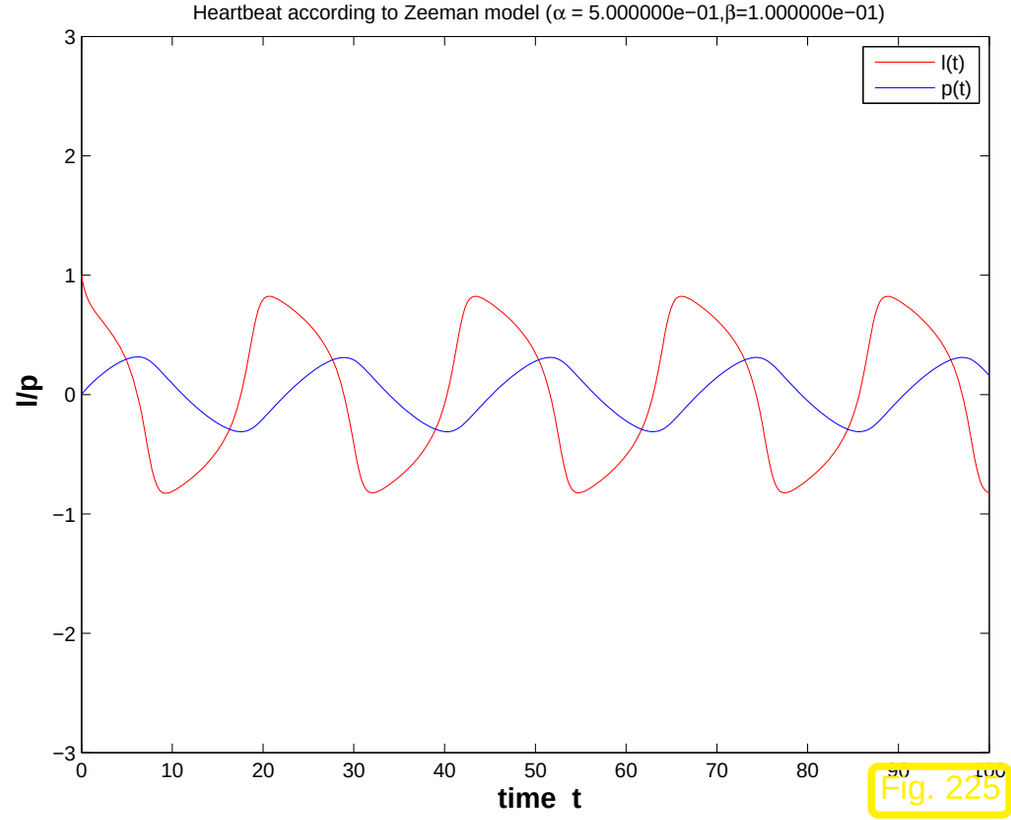


Fig. 225

◇

Example 12.1.8 (Transient circuit simulation). [35, Ch. 64]

Transient nodal analysis, *cf.* Ex. 2.6.3:

Kirchhoff current law

$$i_R(t) - i_L(t) - i_C(t) = 0 \; . \qquad (12.1.9)$$

Transient constitutive relations:

$$i_R(t) = R^{-1} u_R(t) \; , \qquad (12.1.10)$$

$$i_C(t) = C \frac{du_C}{dt}(t) \; , \qquad (12.1.11)$$

$$u_L(t) = L \frac{di_L}{dt}(t) \; . \qquad (12.1.12)$$

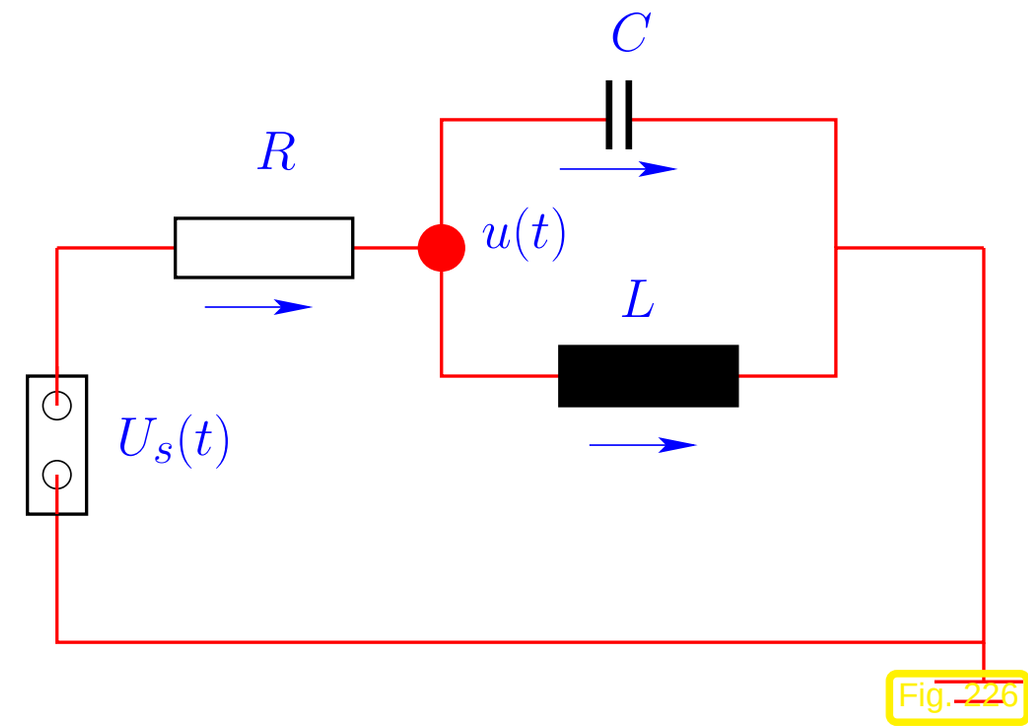


Fig. 226

Given: source voltage $U_s(t)$

► 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])

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \; . \tag{12.1.13}$$

- $\mathbf{f} : I \times D \mapsto \mathbb{R}^d \hat{=}$ right hand side (r.h.s.) ($d \in \mathbb{N}$), given in procedural form

```
function v = f(t,y).
```

- $I \subset \mathbb{R} \hat{=}$ (time)interval $\leftrightarrow$ "time variable" t
- $D \subset \mathbb{R}^d \hat{=}$ state space/phase space $\leftrightarrow$ "state variable" $\mathbf{y}$ (*ger.:* Zustandsraum)
- $\Omega := I \times D \hat{=}$ extended state space (of tupels $(t, \mathbf{y})$)
- $t_0 \hat{=}$ initial time, $\mathbf{y}_0 \hat{=}$ initial state $\succ$ initial conditions

Remark 12.1.14 (Conversion into autonomous ODE).

△

Remark 12.1.15 (From higher order ODEs to first order systems). [13, Sect. 11.2]]

△

Basic assumption: right hand side $\mathbf{f} : I \times D \mapsto \mathbb{R}^d$ locally Lipschitz continuous in $\mathbf{y}$

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 (→ 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* $J(t_0, \mathbf{y}_0) \subset \mathbb{R}$.

In light of Rem. 12.1.14 and Thm. 12.1.21: we consider only

$$\text{autonomous IVP:} \quad \boxed{\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad , \quad \mathbf{y}(0) = \mathbf{y}_0} \, , \tag{12.1.23}$$

with locally Lipschitz continuous (→ 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* $\mathbf{y}_0 \in D$.

Definition 12.1.25 (Evolution operator)**.**
Under Assumption 12.1.24 the mapping

$$\mathbf{\Phi} : \begin{cases} \mathbb{R} \times D & \mapsto & D \\ (t, \mathbf{y}_0) & \mapsto & \mathbf{\Phi}^t \mathbf{y}_0 := \mathbf{y}(t) \end{cases} ,$$

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* $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

12.2 Euler methods

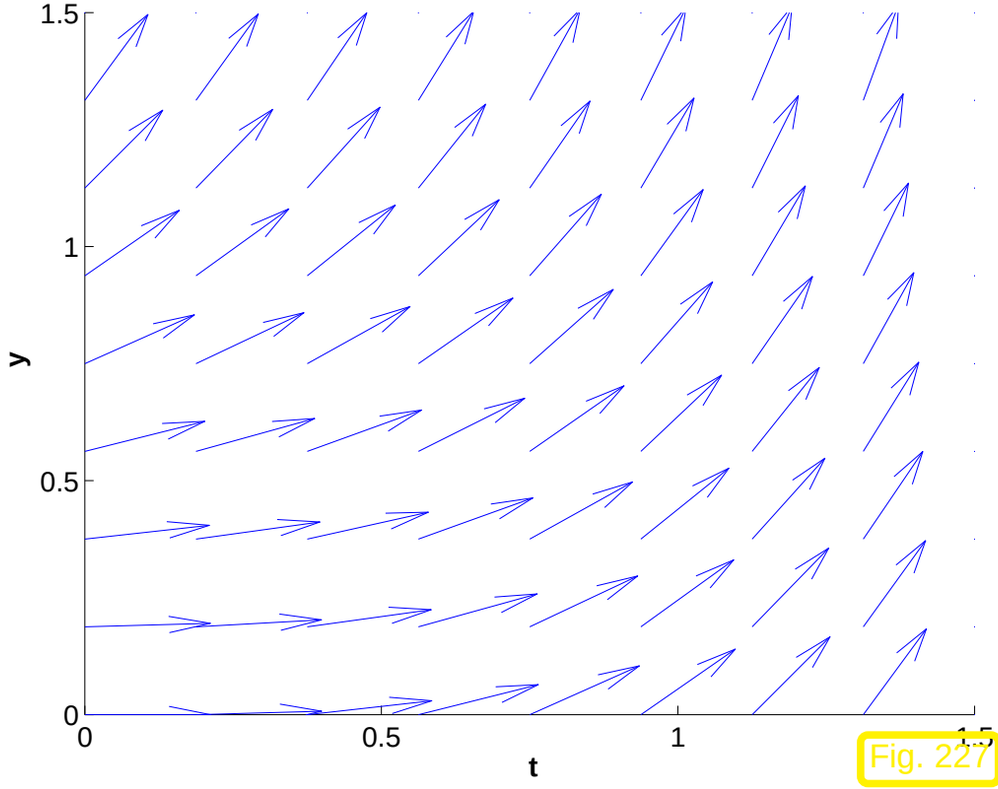
12.2.1 Explicit Euler method

Example 12.2.1 (Tangent field and solution curves)*.*

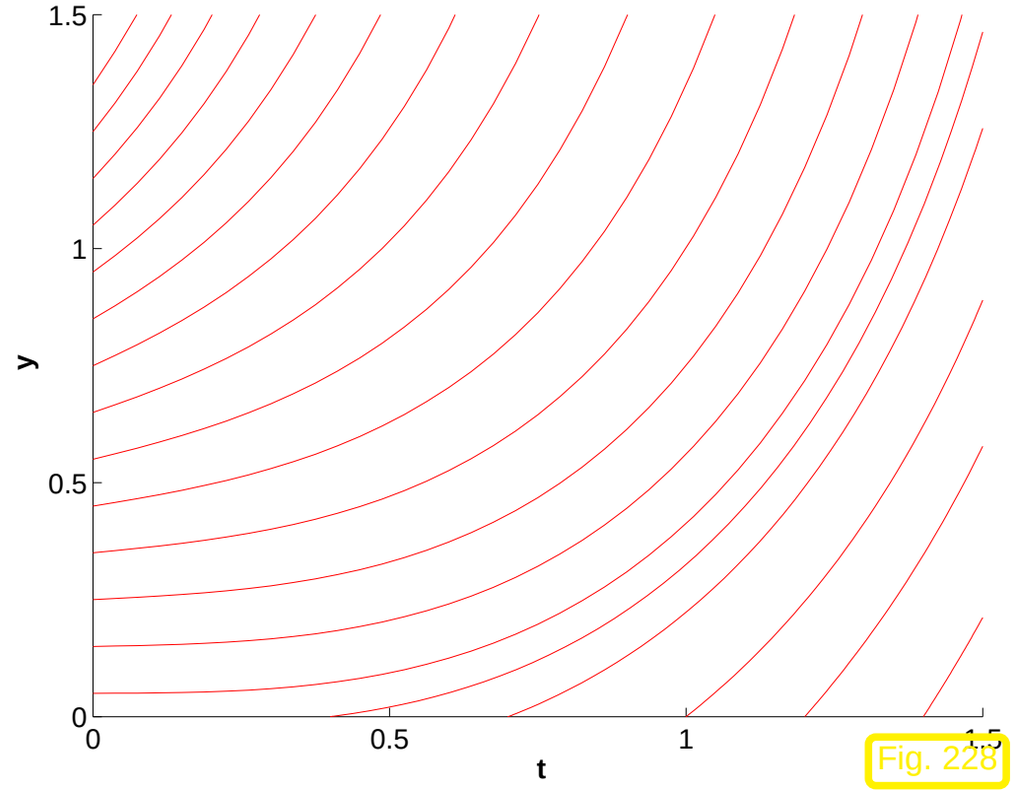
Riccati differential equation

scalar ODE

$$\dot{y} = y^2 + t^2 \quad \blacktriangleright \quad d = 1, \quad I, D = \mathbb{R}^+ . \tag{12.2.2}$$



tangent field

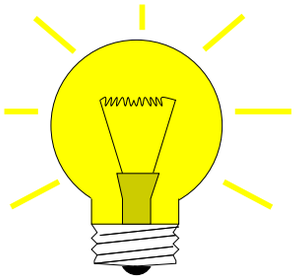


solution curves

◇

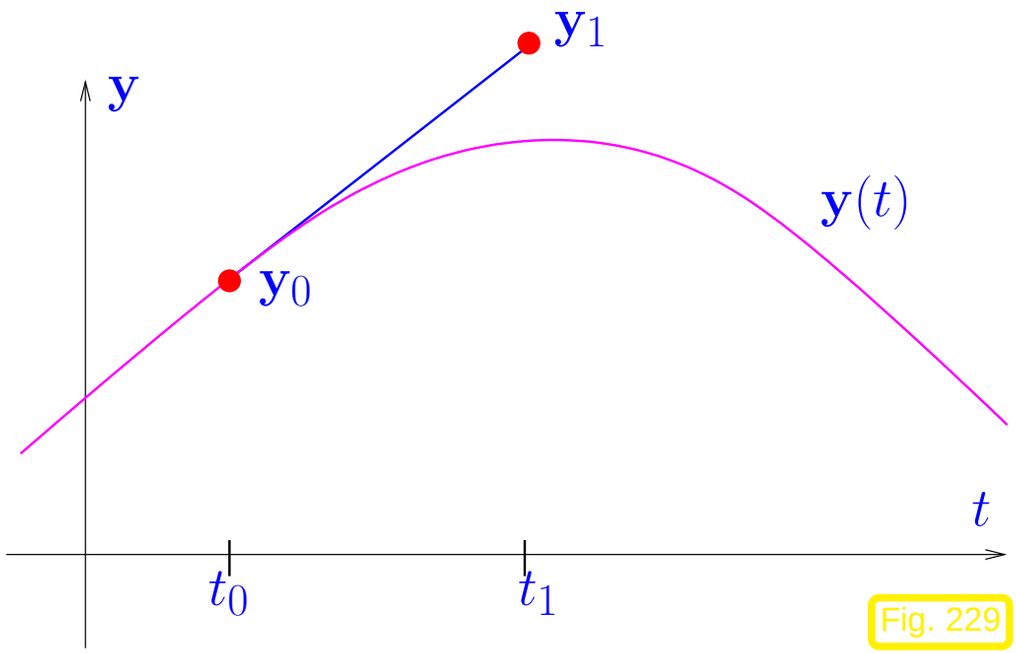
R. Hiptmair

rev 38355, May 3, 2011



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.



Example 12.2.3 (Visualization of explicit Euler method).

explicit Euler method (Euler 1768)

◁ First step of explicit Euler method ($d = 1$):

Slope of tangent $= f(t_0, \mathbf{y}_0)$

$\mathbf{y}_1$ serves as initial value for next step**!**

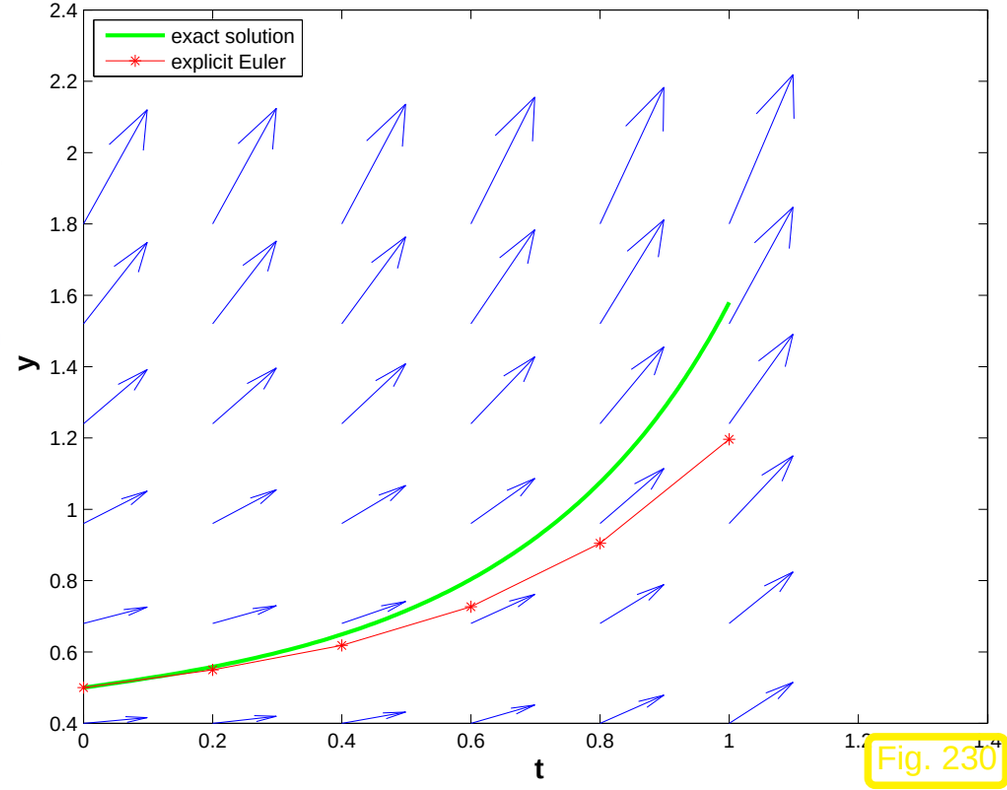
IVP for Riccati differential equation, see Ex. 12.2.1

$$\dot{y} = y^2 + t^2 \ . \qquad (12.2.2)$$

Here: $y_0 = \frac{1}{2}$, $t_0 = 0$, $T = 1$, ▷

— $\hat{=}$ "Euler polygon" for uniform timestep $h = 0.2$

$\mapsto$ $\hat{=}$ tangent field of Riccati ODE



◇

R. Hiptmair

rev 38355, May 3, 2011

Formula: explicit Euler method generates a *sequence* $(\mathbf{y}_k)_{k=0}^N$ by the recursion

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_k, \mathbf{y}_k) \ , \quad k = 0, \dots, N-1 \quad ,} \qquad (12.2.4)$$

with local (size of) timestep (stepsize) $h_k := t_{k+1} - t_k$.

Remark 12.2.5 (Explicit Euler method as difference scheme).

△

12.2.2 Implicit Euler method

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) \;, \quad k = 0, \dots, N-1} \;, \tag{12.2.8}$$

with local timestep (stepsize) $h_k := t_{k+1} - t_k$.

(12.2.8) = implicit Euler method

12.2.3 Abstract single step methods

Definition 12.2.12 (Single step method (for autonomous ODE)). $\rightarrow$ *[51, Def. 11.2]*

Given a discrete evolution $\mathbf{\Psi} : \Omega \subset \mathbb{R} \times D \mapsto \mathbb{R}^d$*, an initial state* $\mathbf{y}_0$*, and a temporal mesh* $\mathcal{M} := \{t_0 < t_1 < \cdots < t_N = T\}$ *the recursion*

$$\mathbf{y}_{k+1} := \mathbf{\Psi}(t_{k+1} - t_k, \mathbf{y}_k) \;, \quad k = 0, \dots, N-1 \;, \tag{12.2.13}$$

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:

$$\mathbf{\Psi}^h\mathbf{y} \quad \longleftrightarrow \quad \texttt{function y1 = esvstep(h,y0)}\,.$$
$$\left(\texttt{function y1 = esvstep(@(y) rhs(y),h,y0)}\right)$$

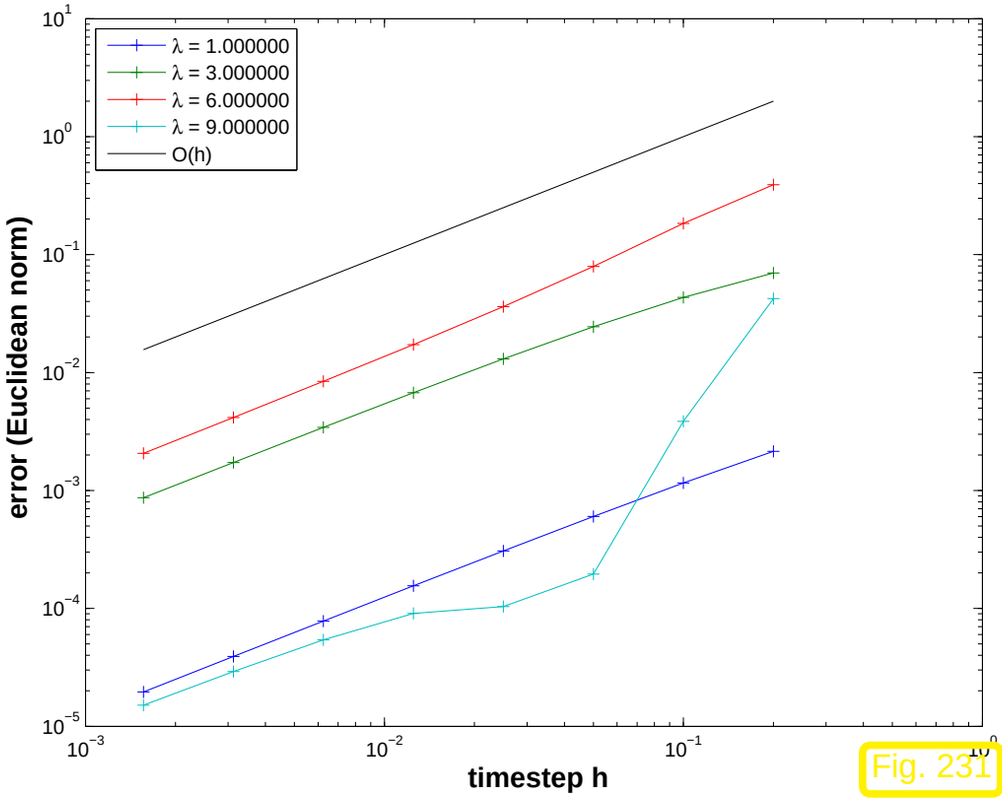
12.3 Convergence of single step methods [13, Sect. 11.5] [51, Sect. 11.3]

Example 12.3.2 (Speed of convergence of Euler methods).

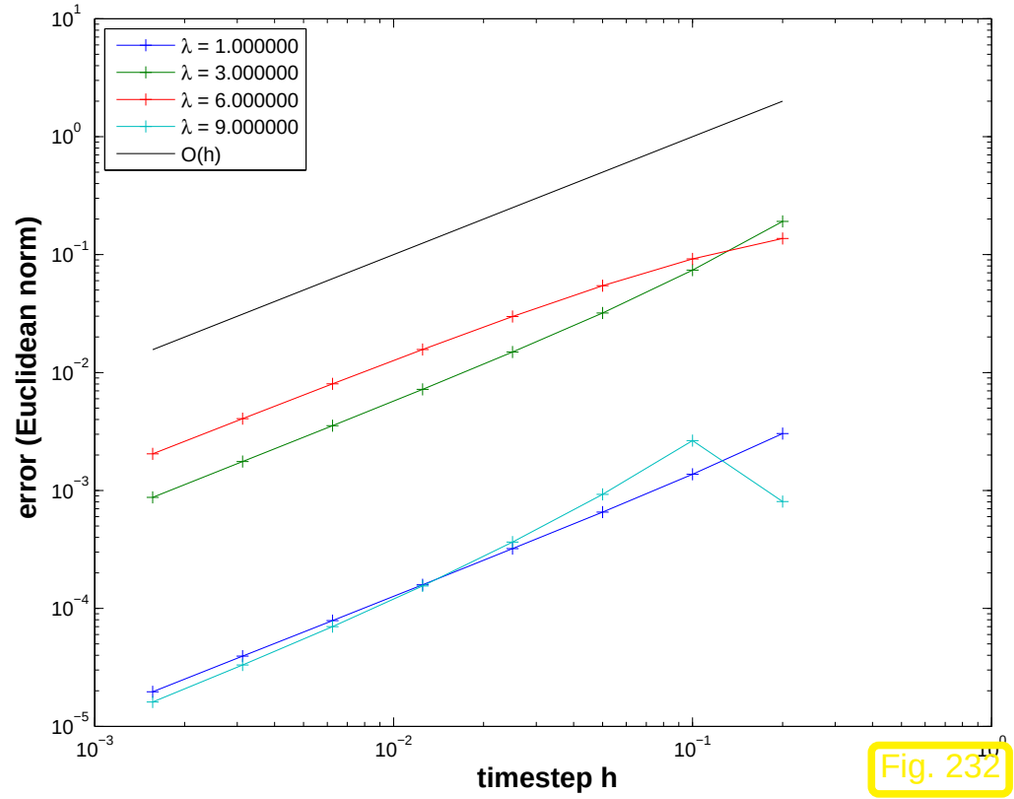
- IVP for logistic ODE, see Ex. 12.1.1

$$\dot{y} = \lambda y(1-y) \quad , \quad y(0) = 0.01\,.$$

- 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

R. Hiptmair

rev 38355, May 3, 2011

$O(h)$ algebraic convergence in both cases

◇

The sequence $(\mathbf{y}_k)_k$ generated by a

single step method ($\rightarrow$ Def. 12.2.12) of order (of consistency) $p \in \mathbb{N}$

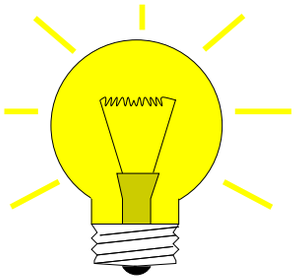
for $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ on a mesh $\mathcal{M} := \{t_0 < t_1 < \cdots < t_N = T\}$ satisfies

$$\max_k \|\mathbf{y}_k - \mathbf{y}(t_k)\| \leq Ch^p \quad \text{for} \quad h := \max_{k=1,\ldots,N} |t_k - t_{k-1}| \to 0 \,, \tag{12.3.16}$$

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.

IVP: $$\begin{aligned} \dot{\mathbf{y}}(t) &= \mathbf{f}(t, \mathbf{y}(t)) \,, \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \quad \Rightarrow \quad \mathbf{y}(t_1) = \mathbf{y}_0 + \int_{t_0}^{t_1} \mathbf{f}(\tau, \mathbf{y}(\tau))\,\mathrm{d}\tau$$



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 $b_1,\dots,b_s$.

$$\mathbf{y}(t_1) \approx \mathbf{y}_1 = \mathbf{y}_0 + h\sum_{i=1}^{s} b_i \mathbf{f}(t_0 + c_i h, \boxed{\mathbf{y}(t_0 + c_i h)}) \;, \quad h := t_1 - t_0 \;. \tag{12.4.2}$$

Obtain these values by bootstrapping

Example 12.4.3 (Construction of simple Runge-Kutta methods).

Quadrature formula = trapezoidal rule (10.2.7):

$$Q(f) = \tfrac{1}{2}(f(0) + f(1)) \quad \leftrightarrow \quad s = 2\text{:} \quad c_1 = 0, c_2 = 1 \;, \quad b_1 = b_2 = \frac{1}{2} \;, \tag{12.4.4}$$

and $\mathbf{y}(T)$ approximated by explicit Euler step (12.2.4)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0) \;, \quad \mathbf{k}_2 = \mathbf{f}(t_0 + h, \mathbf{y}_0 + h\mathbf{k}_1) \;, \quad \mathbf{y}_1 = \mathbf{y}_0 + \tfrac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2) \;. \tag{12.4.5}$$

(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)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0) \;, \quad \mathbf{k}_2 = \mathbf{f}(t_0 + \tfrac{h}{2}, \mathbf{y}_0 + \tfrac{h}{2}\mathbf{k}_1) \;, \quad \mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{k}_2 \;. \tag{12.4.6}$$

(12.4.6) = explicit midpoint rule (for numerical integration of ODEs) [13, Alg. 11.18].

$\diamond$

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.

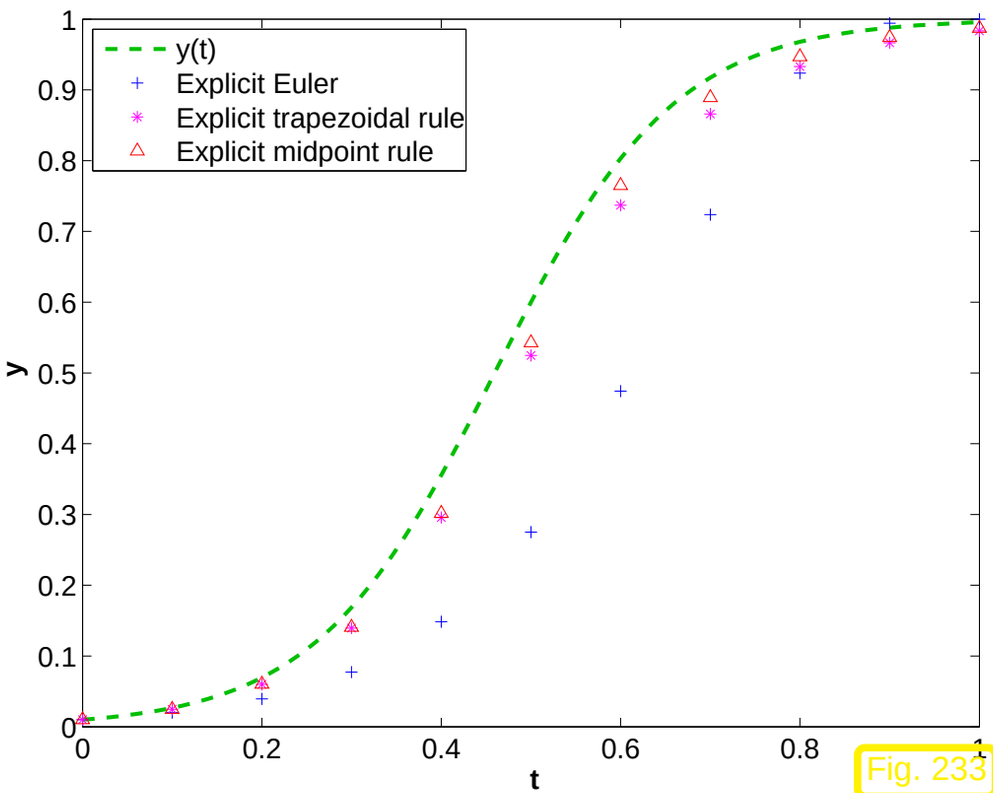


Fig. 233

$y_h(j/10)$, $j = 1, \dots, 10$ for explicit RK-methods

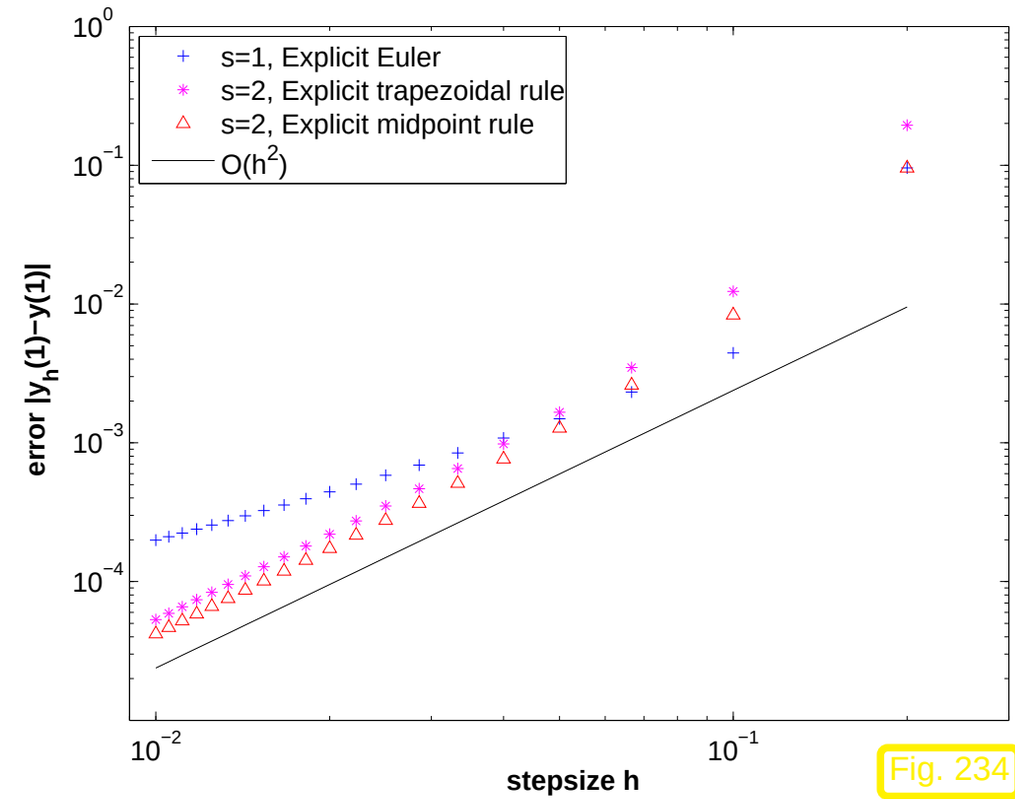


Fig. 234

Errors at final time $y_h(1) - y(1)$

◇

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*

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h\sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j)\ , \quad i = 1,\dots,s \quad , \quad \mathbf{y}_1 := \mathbf{y}_0 + h\sum_{i=1}^{s} b_i \mathbf{k}_i\ .$$

The $\mathbf{k}_i \in \mathbb{R}^d$ *are called increments.*

Shorthand notation for (explicit) Runge-Kutta methods [13, (11.75)]

Butcher scheme

(Note: $\mathfrak{A}$ is strictly lower triangular $s \times s$-matrix)

$$\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} \quad := \quad \begin{array}{c|cccc} c_1 & 0 & & \cdots & 0 \\ c_2 & a_{21} & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{s,s-1} & 0 \\ \hline & b_1 & \cdots & b_s & \end{array} \ . \tag{12.4.9}$$

Example 12.4.10 (Butcher scheme for some explicit RK-SSM). [13, Sect. 11.6.1]

R. Hiptmair

rev 38355, May 3, 2011

- 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

R. Hiptmair

rev 38355, May 3, 2011

- Kutta's 3/8-rule:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{2}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 1 & 1 & -1 & 1 & 0 \\ \hline & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{array} \quad \rightarrow \quad \text{order} = 4$$

◇

Remark 12.4.11 ("Butcher barriers" for explicit RK-SSM).

order p	1	2	3	4	5	6	7	8	≥ 9
minimal no. s of stages	1	2	3	4	6	7	9	11	$\geq p+3$

△

Remark 12.4.12 (Explicit ODE integrator in MATLAB).

Syntax:

```
[t,y] = ode45(odefun,tspan,y0);
```

`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 $\{t_0 < t_1 < t_2 < \cdots < t_{N-1} = t_N = T\}$

`y` : sequence $(\mathbf{y}_k)_{k=0}^N$ (column vectors)

Code 12.4.13: code excerpts for MATLAB integrator `ode45`

```
function varargout = ode45(ode,tspan,y0,options,varargin)
% Processing of input parameters omitted
% ⋮
% Initialize method parameters, c.f.  Butcher scheme (12.4.9)
pow = 1/5;
A = [1/5, 3/10, 4/5, 8/9, 1, 1];
B = [
    1/5         3/40    44/45   19372/6561      9017/3168       35/384
    0           9/40    -56/15  -25360/2187     -355/33         0
    0           0       32/9    64448/6561      46732/5247      500/1113
    0           0       0       -212/729        49/176          125/192
    0           0       0       0               -5103/18656     -2187/6784
    0           0       0       0               0               11/84
    0           0       0       0               0               0
    ];
E = [71/57600; 0; -71/16695; 71/1920; -17253/339200; 22/525; -1/40];
% ⋮ (choice of stepsize and main loop omitted)
```

```
% ADVANCING ONE STEP.
hA = h * A;
hB = h * B;
f(:,2) = feval(odeFcn,t+hA(1),y+f*hB(:,1),odeArgs{:});
f(:,3) = feval(odeFcn,t+hA(2),y+f*hB(:,2),odeArgs{:});
f(:,4) = feval(odeFcn,t+hA(3),y+f*hB(:,3),odeArgs{:});
f(:,5) = feval(odeFcn,t+hA(4),y+f*hB(:,4),odeArgs{:});
f(:,6) = feval(odeFcn,t+hA(5),y+f*hB(:,5),odeArgs{:});

tnew = t + hA(6);
if done, tnew = tfinal; end  % Hit end point exactly.
h = tnew - t;       % Purify h.
ynew = y + f*hB(:,6);
% ⋮ (stepsize control, see Sect. 12.5 dropped
```

△

Example 12.4.14 (Numerical integration of logistic ODE in MATLAB).

MATLAB-CODE: usage of ode45

```
fn = @(t,y) 5*y*(1-y);
[t,y] = ode45(fn,[0 1.5],y0);
plot(t,y,'r-');
```

MATLAB-integrator: `ode45()`:

- Handle passing r.h.s.
- initial and final time
- initial state $\mathbf{y}_0$

◇

12.5 Stepsize control [13, Sect. 11.7], [51, Sect. 11.8.2]

Example 12.5.1 (Oregonator reaction).

$$\begin{aligned}
\mathrm{BrO}_3^- + \mathrm{Br}^- &\mapsto \mathrm{HBrO}_2 \\
\mathrm{HBrO}_2 + \mathrm{Br}^- &\mapsto \mathrm{Org} \\
\mathrm{BrO}_3^- + \mathrm{HBrO}_2 &\mapsto 2\,\mathrm{HBrO}_2 + \mathrm{Ce(IV)} \\
2\,\mathrm{HBrO}_2 &\mapsto \mathrm{Org} \\
\mathrm{Ce(IV)} &\mapsto \mathrm{Br}^-
\end{aligned} \tag{12.5.2}$$

$$\Downarrow$$

$$\begin{aligned}
y_1 := c(\mathrm{BrO}_3^-): &\quad \dot{y}_1 = -k_1 y_1 y_2 - k_3 y_1 y_3 \;, \\
y_2 := c(\mathrm{Br}^-): &\quad \dot{y}_2 = -k_1 y_1 y_2 - k_2 y_2 y_3 + k_5 y_5 \;, \\
y_3 := c(\mathrm{HBrO}_2): &\quad \dot{y}_3 = k_1 y_1 y_2 - k_2 y_2 y_3 + k_3 y_1 y_3 - 2k_4 y_3^2 \;, \\
y_4 := c(\mathrm{Org}): &\quad \dot{y}_4 = k_2 y_2 y_3 + k_4 y_3^2 \;, \\
y_5 := c(\mathrm{Ce(IV)}): &\quad \dot{y}_5 = k_3 y_1 y_3 - k_5 y_5 \;,
\end{aligned} \tag{12.5.3}$$

with (non-dimensionalized) reaction constants:

$$k_1 = 1.34 \;, \quad k_2 = 1.6 \cdot 10^9 \;, \quad k_3 = 8.0 \cdot 10^3 \;, \quad k_4 = 4.0 \cdot 10^7 \;, \quad k_5 = 1.0 \;.$$

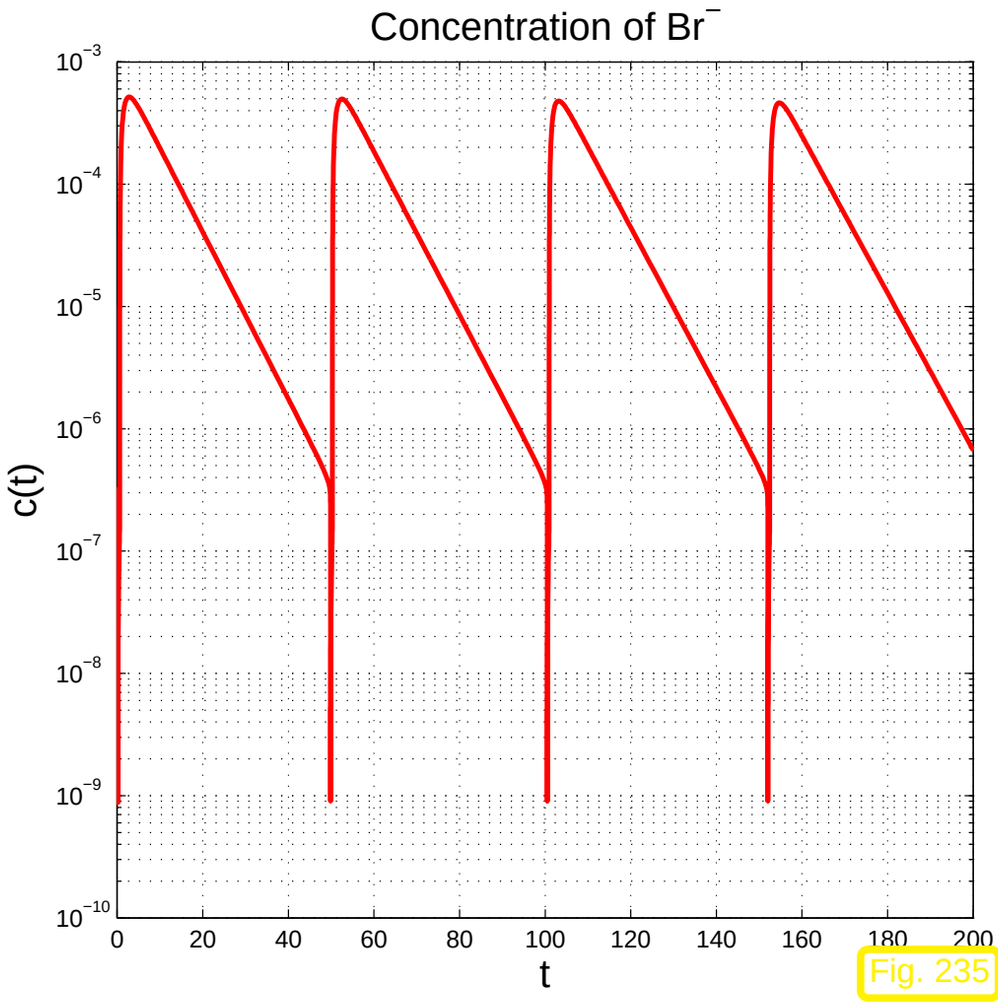


Fig. 235

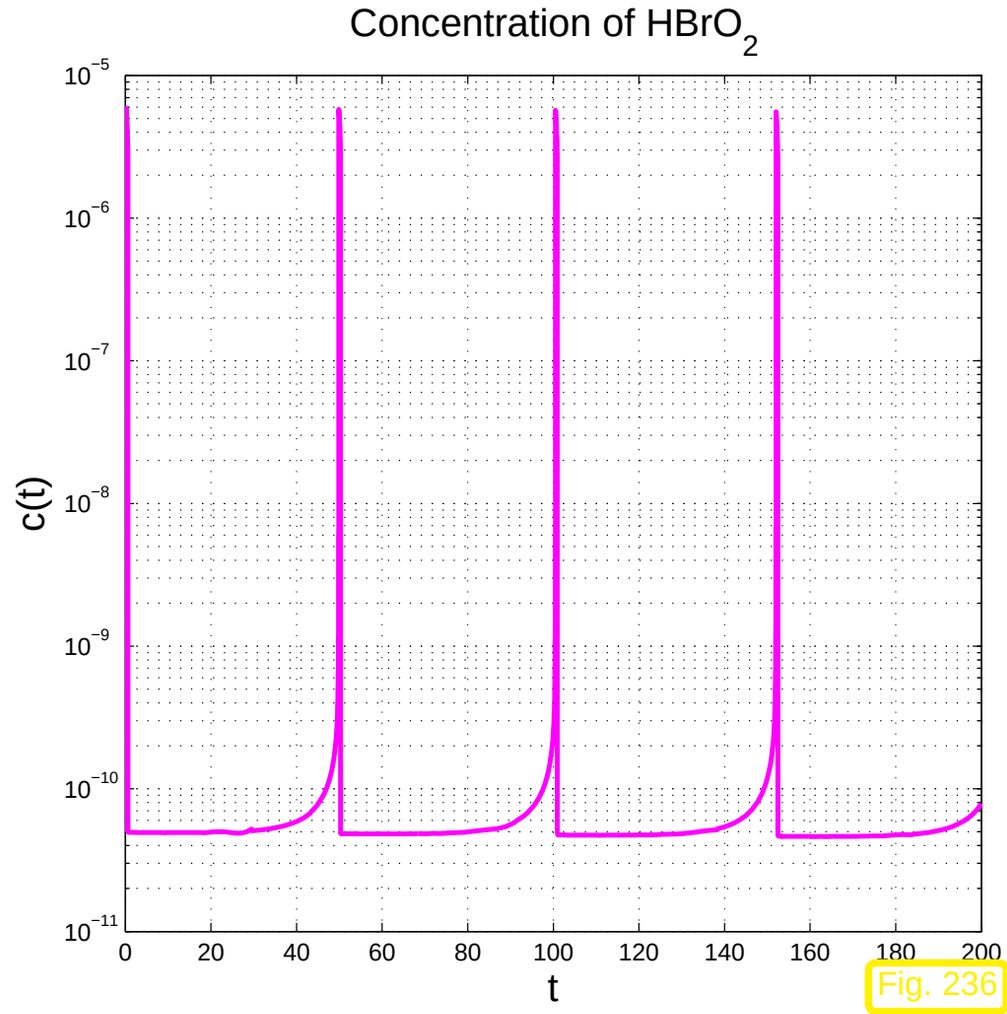


Fig. 236

◇

Example 12.5.4 (Blow-up).

Scalar autonomous IVP:

$$\dot{y} = y^2 \;, \quad y(0) = y_0 > 0 \;.$$

$$\blacktriangleright \quad y(t) = \frac{y_0}{1 - y_0 t} \;, \quad t < {}^{1}\!/_{y_0} \;.$$

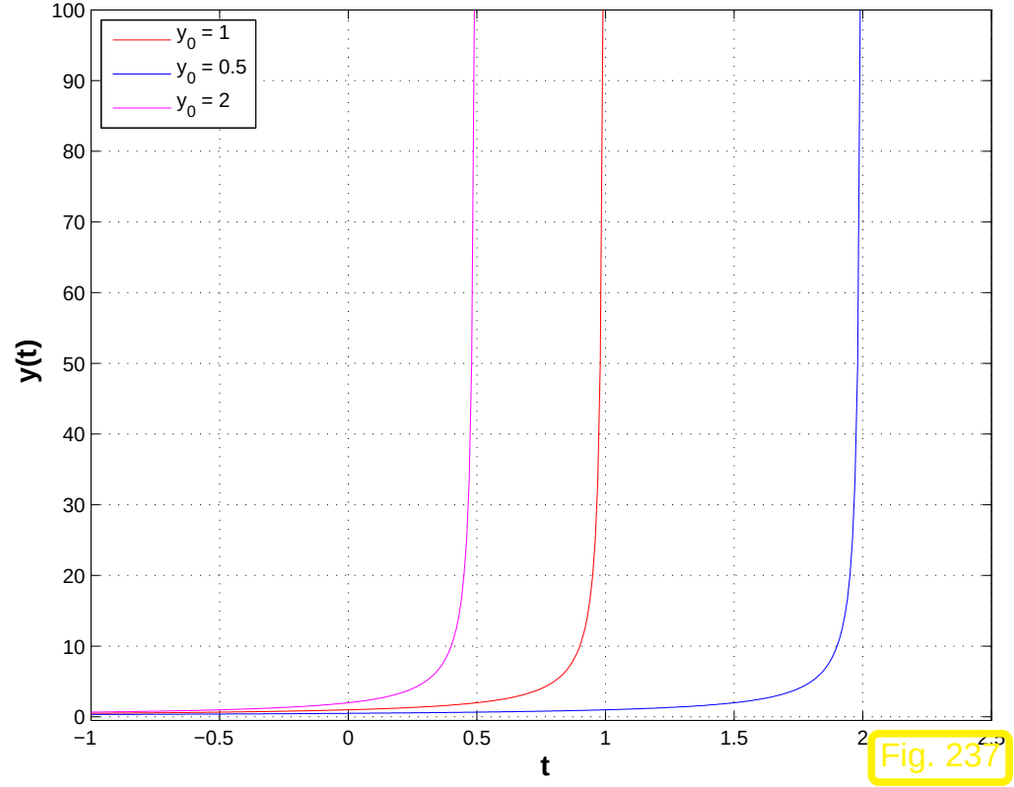


Fig. 237

R. Hiptmair

rev 38355, May 3, 2011

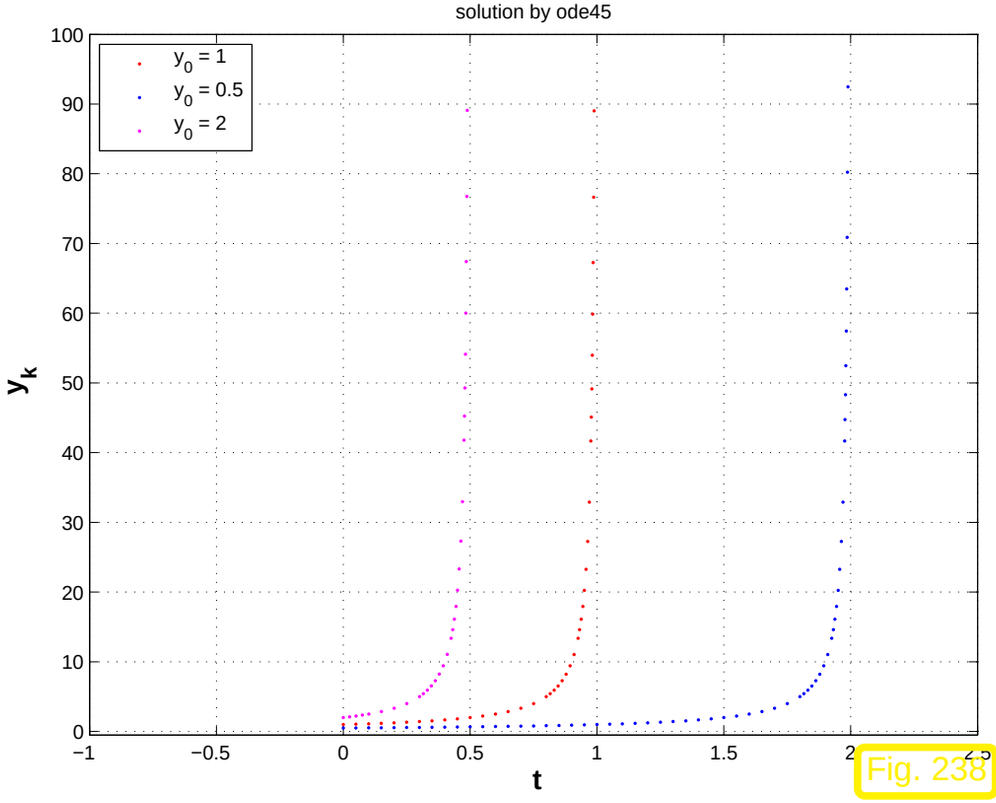


Fig. 238

```
fun = @(t,y) y.^2;
[t1,y1] = ode45(fun,[0 2],1);
[t2,y2] = ode45(fun,[0 2],0.5);
[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
```

◇

be efficient — be accurate

Objective: N as small as possible & $\max_{k=1,\dots,N} \|\mathbf{y}(t_k) - \mathbf{y}_k\| < \texttt{TOL}$ or $\|\mathbf{y}(T) - \mathbf{y}_N\| < \texttt{TOL}$, $\texttt{TOL}$ = tolerance

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 $\mathbf{y}_k, h_{k-1}$)

Idea: *Estimation of one-step error*, *cf.* Sect. 10.5

Compare two discrete evolutions $\mathbf{\Psi}^h$, $\widetilde{\mathbf{\Psi}}^h$ of different order for *current timestep* h:

If Order$(\widetilde{\mathbf{\Psi}}) >$ Order$(\mathbf{\Psi})$

$$\Rightarrow \quad \underbrace{\boldsymbol{\Phi}^h\mathbf{y}(t_k) - \boldsymbol{\Psi}^h\mathbf{y}(t_k)}_{\text{one-step error}} \approx \mathtt{EST}_k := \widetilde{\boldsymbol{\Psi}}^h\mathbf{y}(t_k) - \boldsymbol{\Psi}^h\mathbf{y}(t_k)\,. \tag{12.5.5}$$

Heuristics for concrete h

absolute tolerance

$$\blacktriangleright \quad \text{Compare} \quad \begin{array}{l} \mathtt{EST}_k \leftrightarrow \mathtt{ATOL} \\ \mathtt{EST}_k \leftrightarrow \mathtt{RTOL}\,\|\mathbf{y}_k\| \end{array} \quad \succ \quad \text{Reject/accept current step} \tag{12.5.6}$$

relative tolerance

R. Hiptmair

rev 38355, May 3, 2011

Code 12.5.7: simple local stepsize control for single step methods

```
function [t,y] =
  odeintadapt(Psilow,Psihigh,T,y0,h0,reltol,abstol,hmin)
t = 0; y = y0; h = h0;                    %
while ((t(end) < T)  (h > hmin)) %
  yh = Psihigh(h,y0);  % high order discrete evolution Ψ̃^h
  yH = Psilow(h,y0);   % low order discrete evolution Ψ^h
  est = norm(yH-yh);   % ↔ EST_k

  if (est < max(reltol*norm(y0),abstol))                    %
```

```
    y0 = yh; y = [y,y0]; t = [t,t(end) + min(T-t(end),h)]; %
    h = 1.1*h;  % step accepted, try with increased stepsize
  else, h = h/2; end    % step rejected, try with half the stepsize
end
```

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)

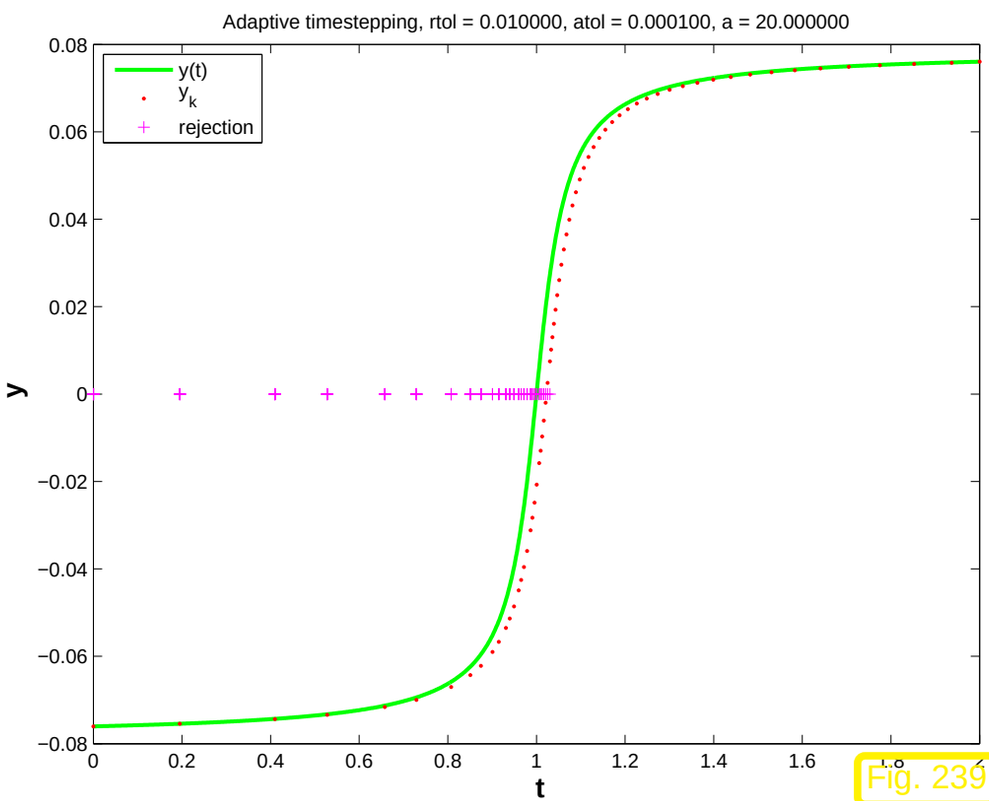


Fig. 239

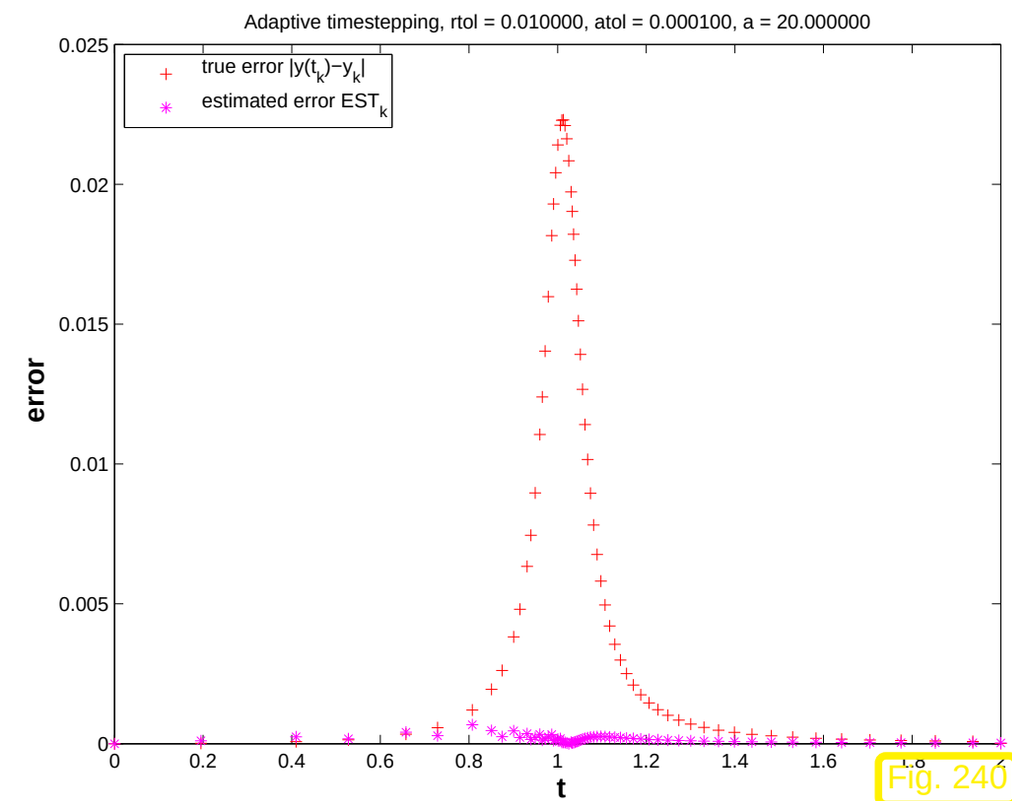
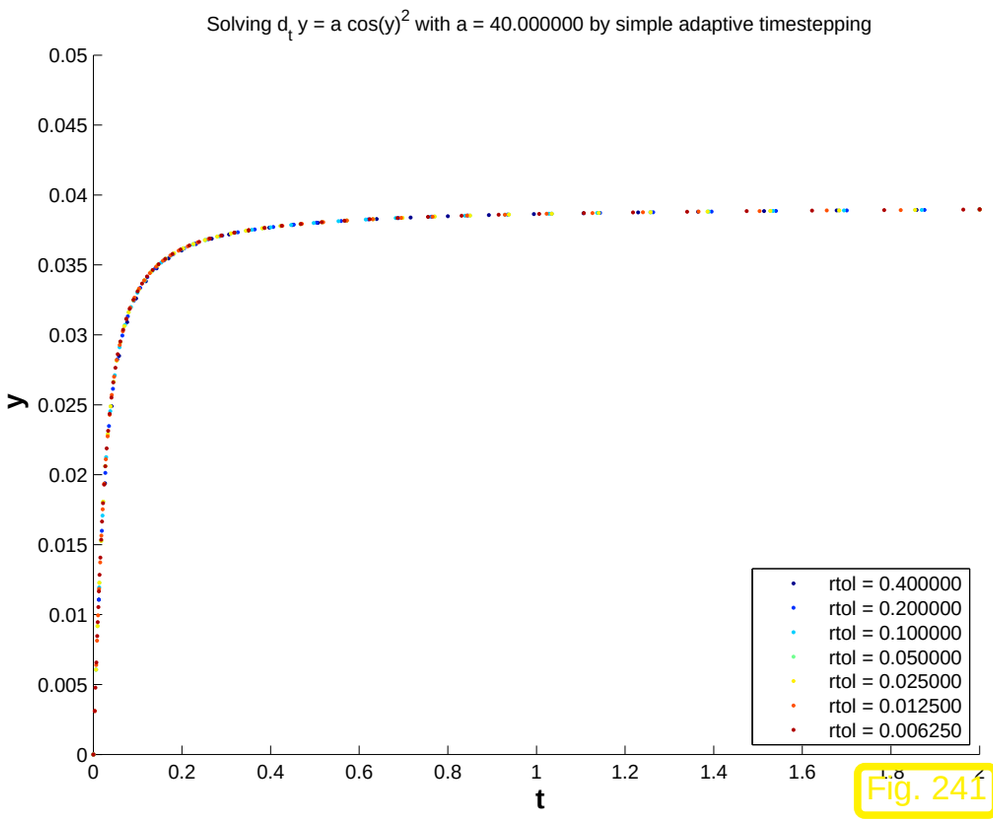


Fig. 240

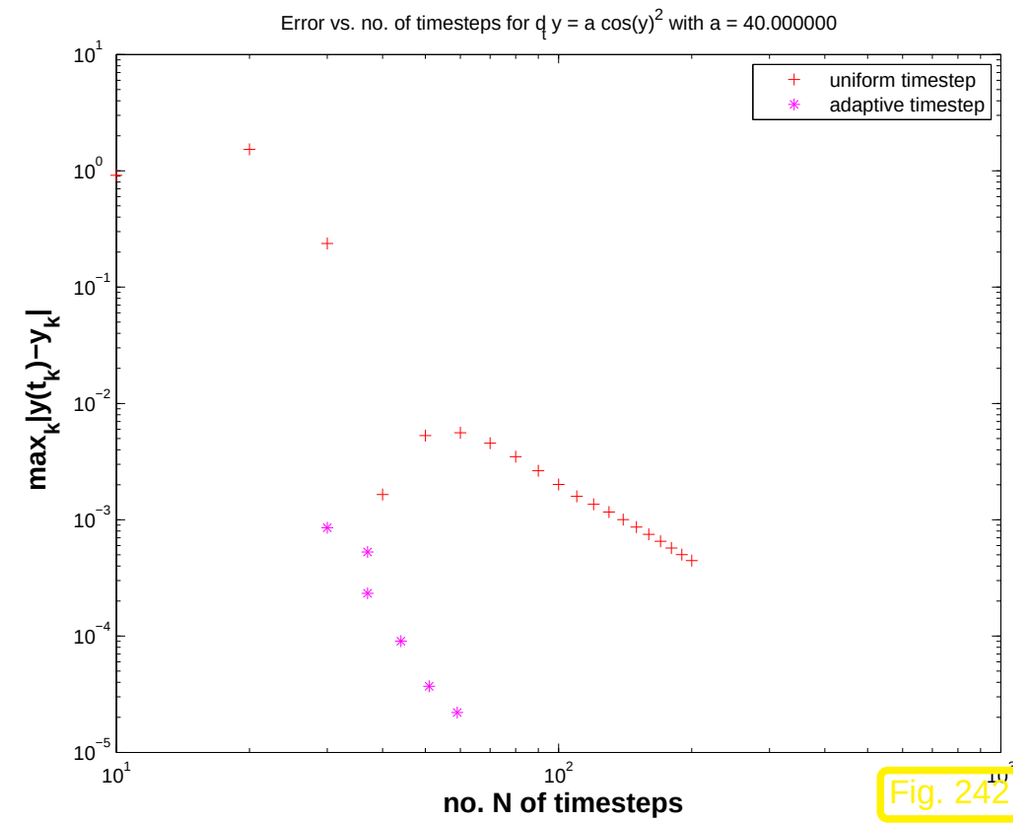
Statistics: `66 timesteps, 131 rejected timesteps`

◇

Example 12.5.10 (Gain through adaptivity). $\longrightarrow$ Ex. 12.5.8



Solutions $(\mathbf{y}_k)_k$ for different values of `rtol`



Error vs. computational effort

◇

Example 12.5.12 ("Failure" of adaptive timestepping). $\rightarrow$ Ex. 12.5.10

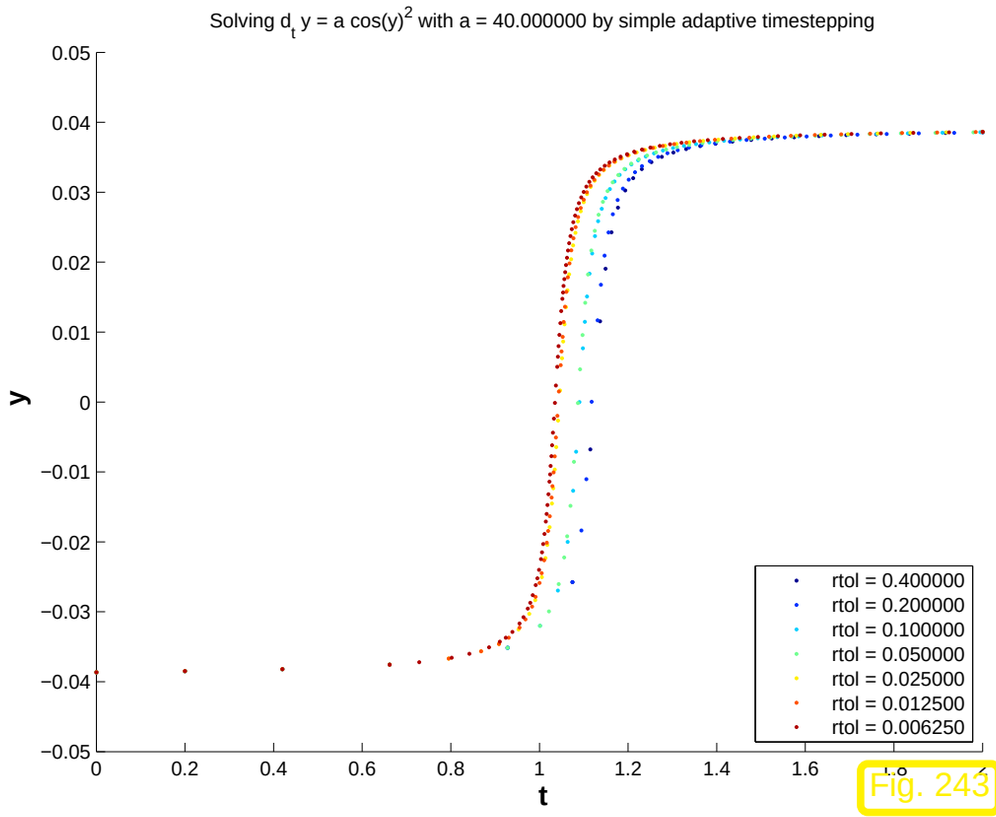


Fig. 243

Solutions $(\mathbf{y}_k)_k$ for different values of `rtol`

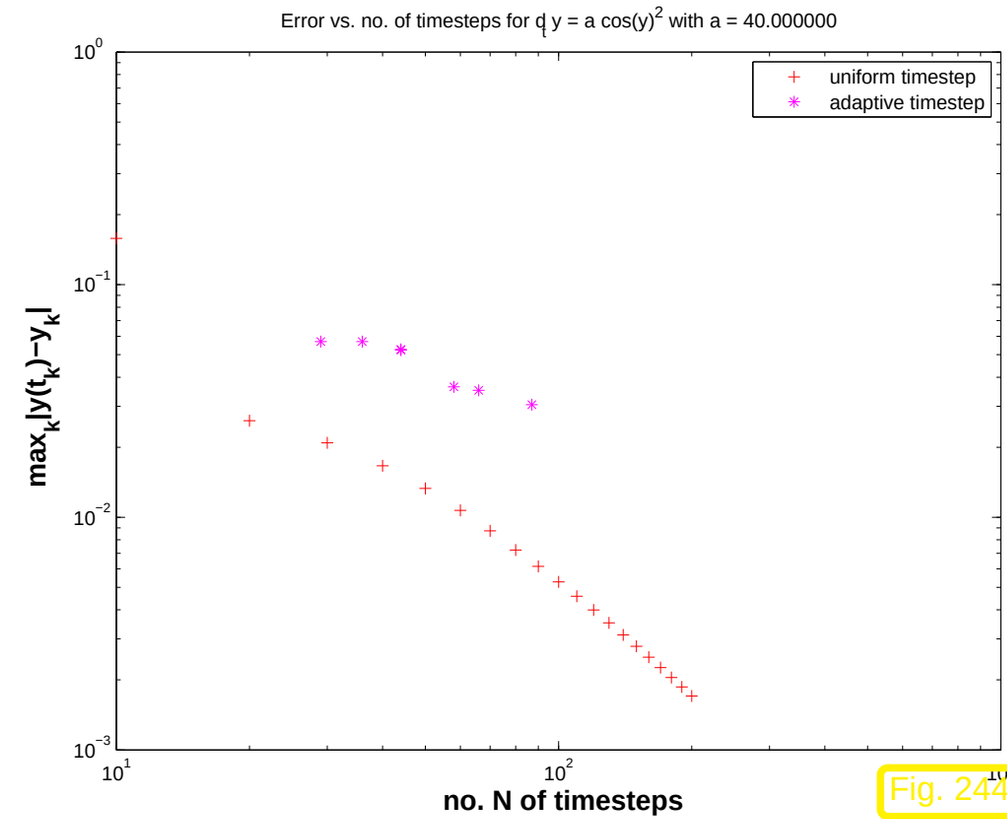


Fig. 244

Error vs. computational effort

◇

Remark 12.5.20 (Stepsize control in MATLAB).

$\Psi \hat{=}$ RK-method of order 4 $\qquad$ $\widetilde{\Psi} \hat{=}$ 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` $\hat{=} [t_0, T]$, `y0` $\hat{=} \mathbf{y}_0$, `t` $\hat{=} t_k$, `y` $\hat{=} \mathbf{y}_k$)

△

R. Hiptmair

rev 38355, May 3, 2011

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: $$\ddot{\mathbf{y}} = F(\mathbf{y}) := -\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2} \,. \tag{12.5.22}$$

acceleration $\qquad$ force

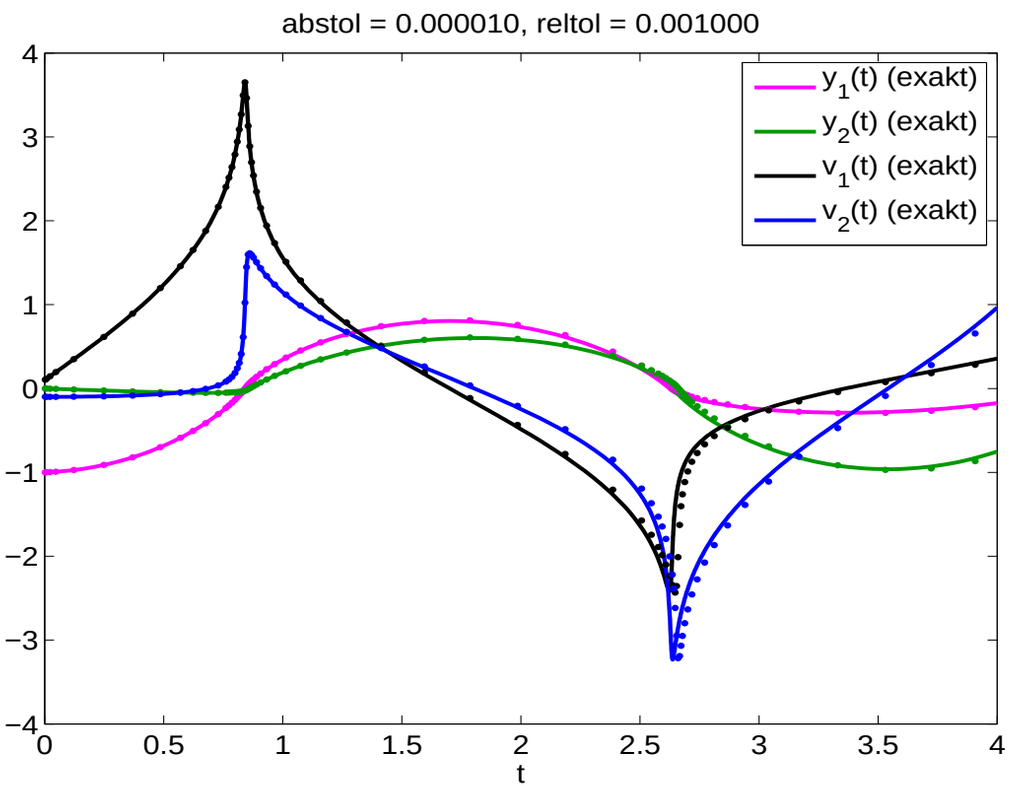
Equivalent 1st-order ODE, see Rem. 12.1.15: with velocity $\mathbf{v} := \dot{\mathbf{y}}$

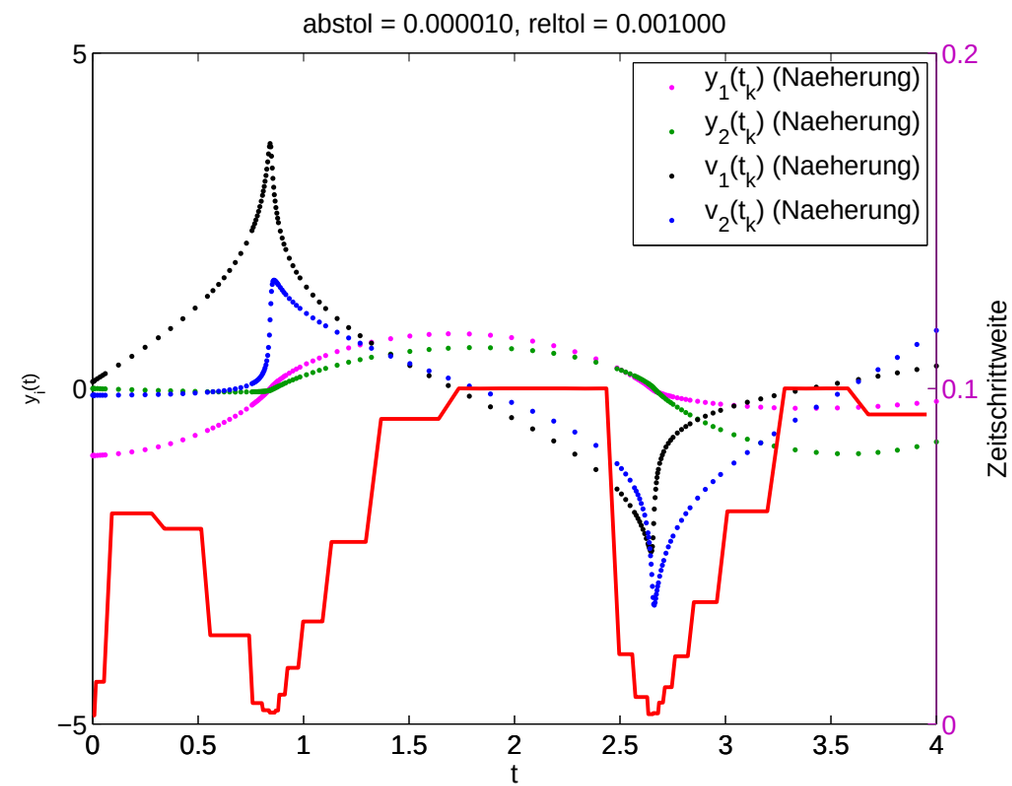
$$\begin{pmatrix} \dot{\mathbf{y}} \\ \dot{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ -\dfrac{2\mathbf{y}}{\|\mathbf{y}\|_2^2} \end{pmatrix} . \tag{12.5.23}$$

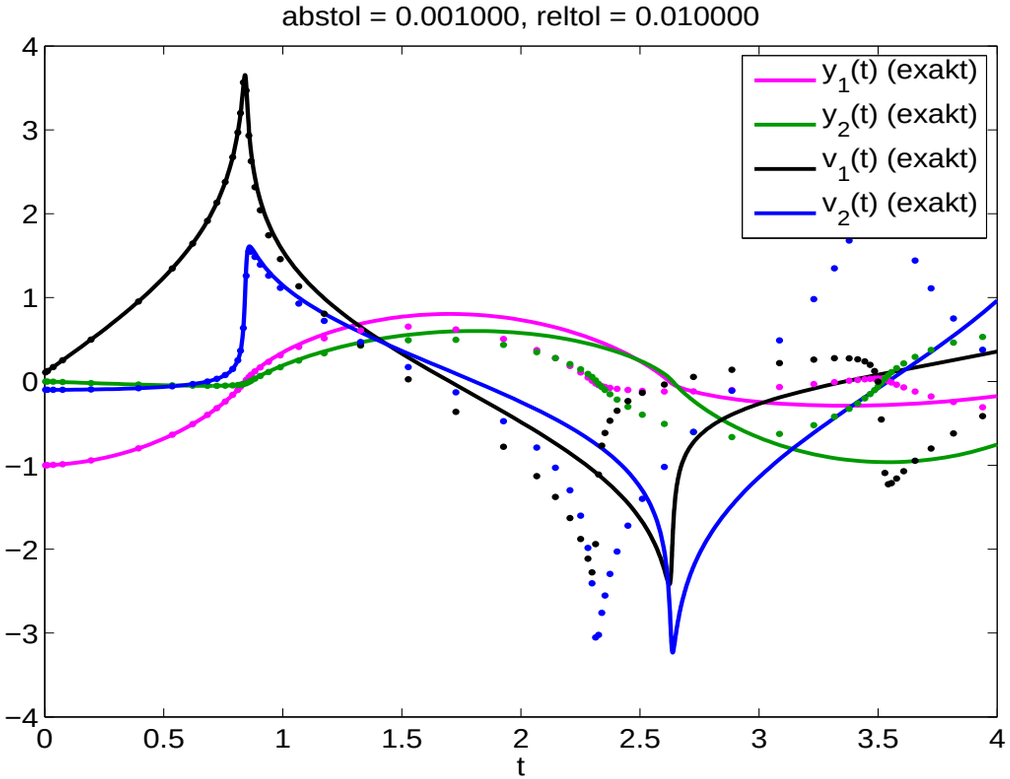
Adaptive integrator: `ode45(@(t,x) f(t,x),[0 4],[-1;0;0.1;-0.1,],options)`:

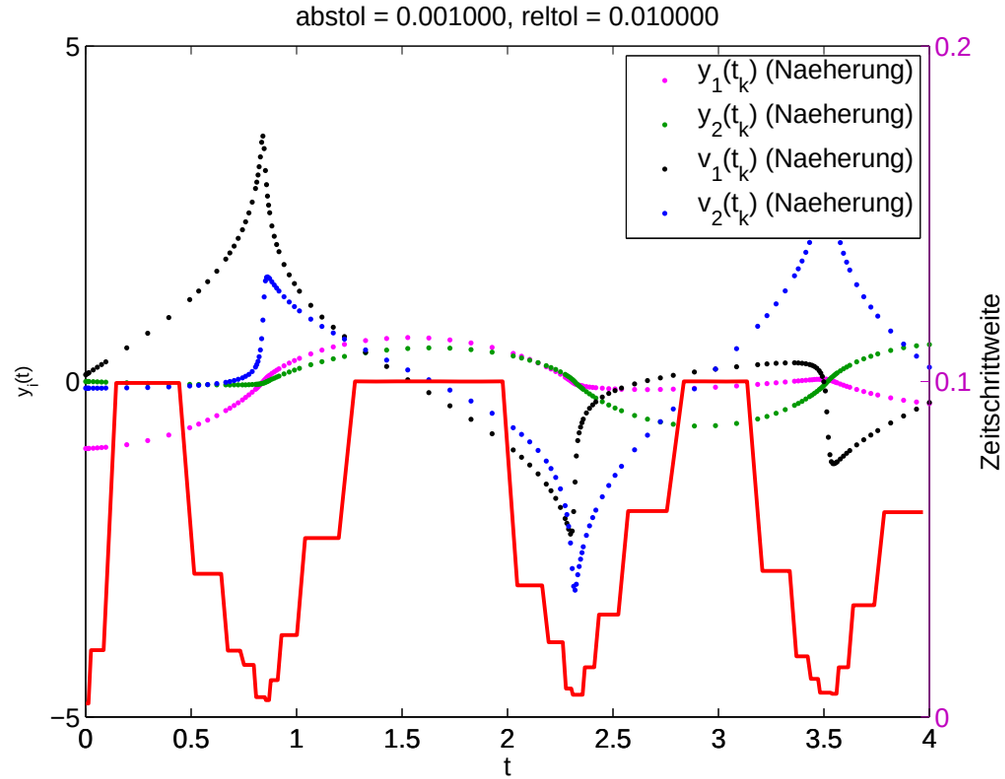
❶ `options = odeset('reltol',0.001,'abstol',1e-5);`

❷ `options = odeset('reltol',0.01,'abstol',1e-3);`



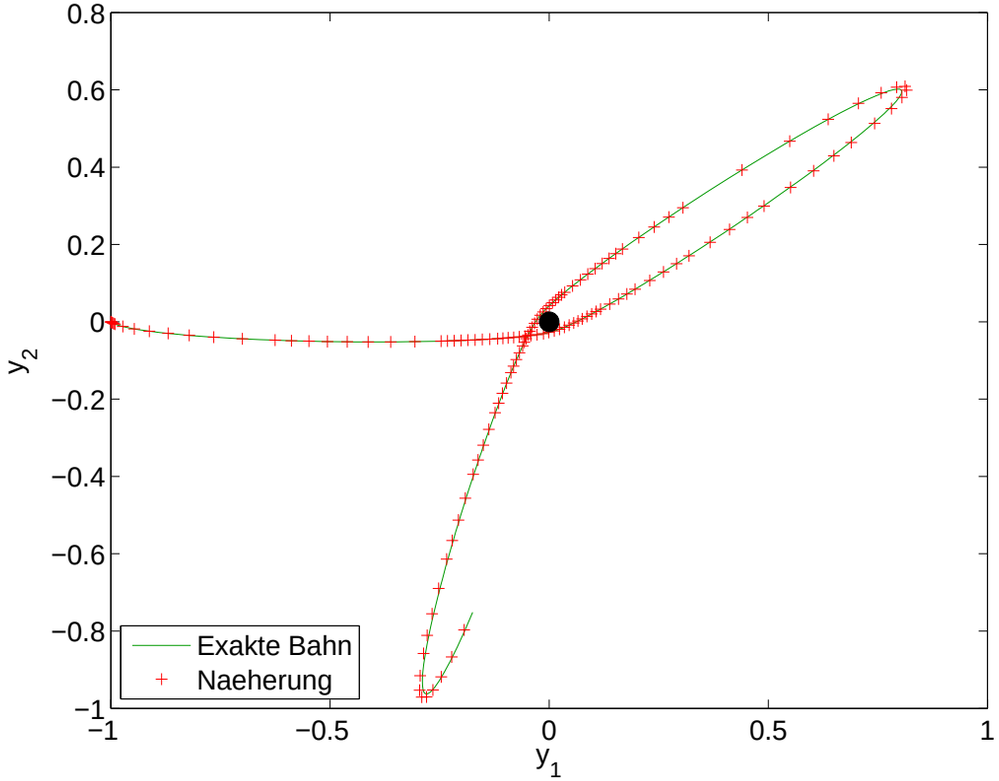




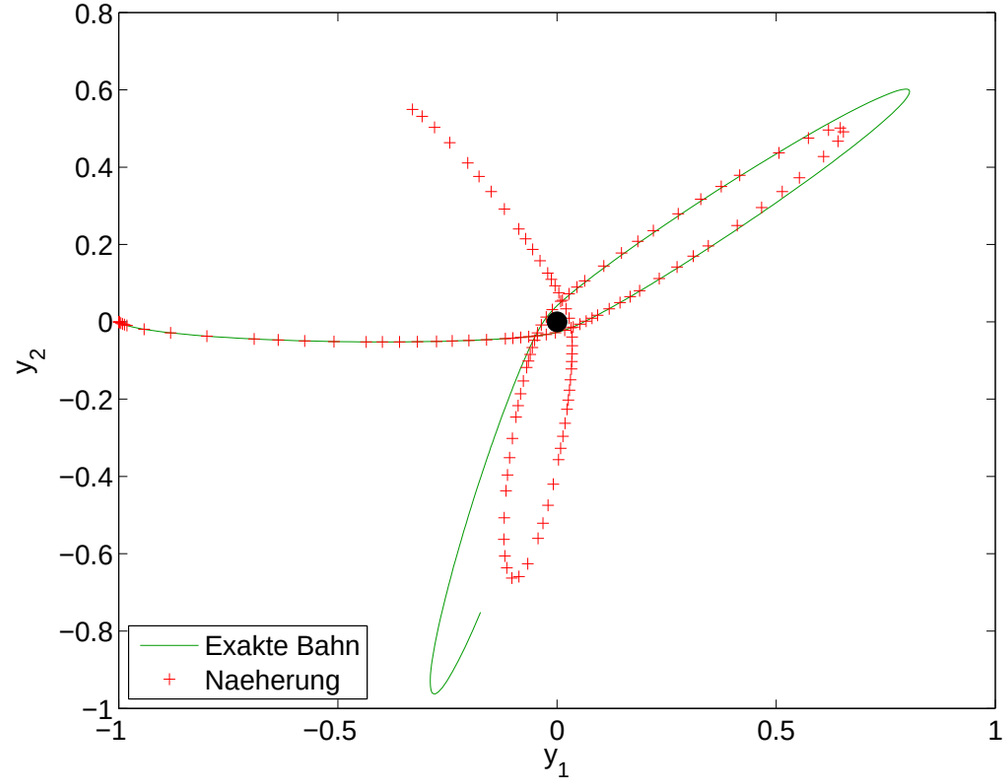


R. Hiptmair

rev 38355, May 3, 2011



reltol=0.001, abstol=1e-5



reltol=0.01, abstol=1e-3

R. Hiptmair

rev 38355, May 3, 2011

◇

Absolute/relative tolerances do *not* allow to predict accuracy of solution!

13 Stiff Integrators [13, Sect. 11.9]

Example 13.0.1 (ode45 for stiff problem).

IVP: $\dot{y} = \lambda y^2(1-y)$, $\lambda := 500$, $y(0) = \frac{1}{100}$.

```
fun = @(t,x) 500*x^2*(1-x);
options = odeset('reltol',0.1,'abstol',0.001,'stats','on');
[t,y] = ode45(fun,[0 1],y0,options);
```

```
186 successful steps
55 failed attempts
1447 function evaluations
```

R. Hiptmair

rev 38355, December 23, 2010

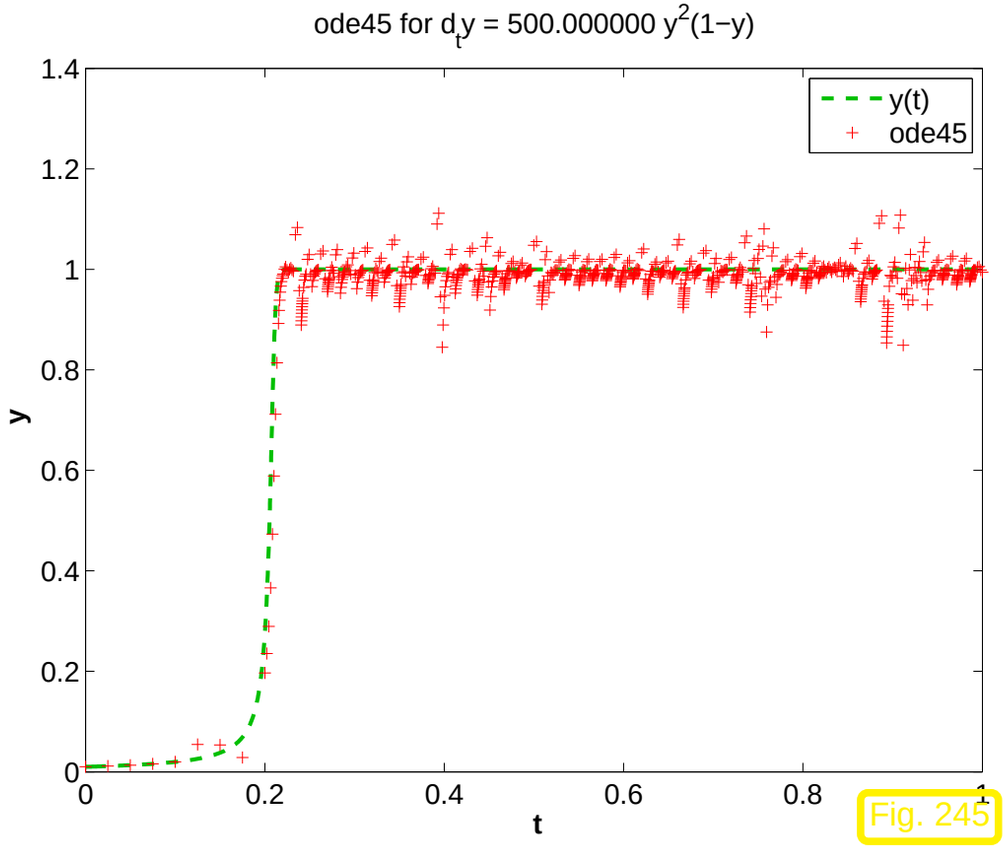


Fig. 245

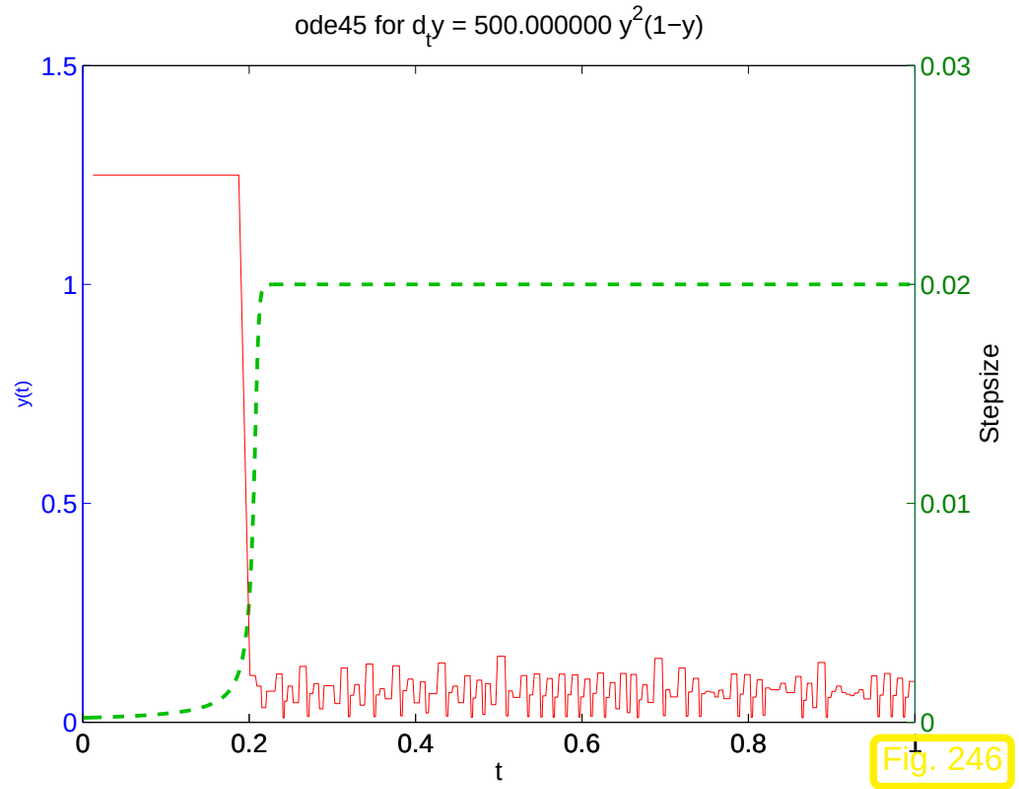


Fig. 246

◇

13.1 Model problem analysis [35, Ch. 77], [51, Sect. 11.3.3]

Example 13.1.1 (Blow-up of explicit Euler method).

- IVP for logistic ODE, see Ex. 12.1.1

$$\dot{y} = f(y) := \lambda y(1-y) \quad , \quad y(0) = 0.01 \; .$$

- Explicit Euler method (12.2.4) with uniform timestep $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.

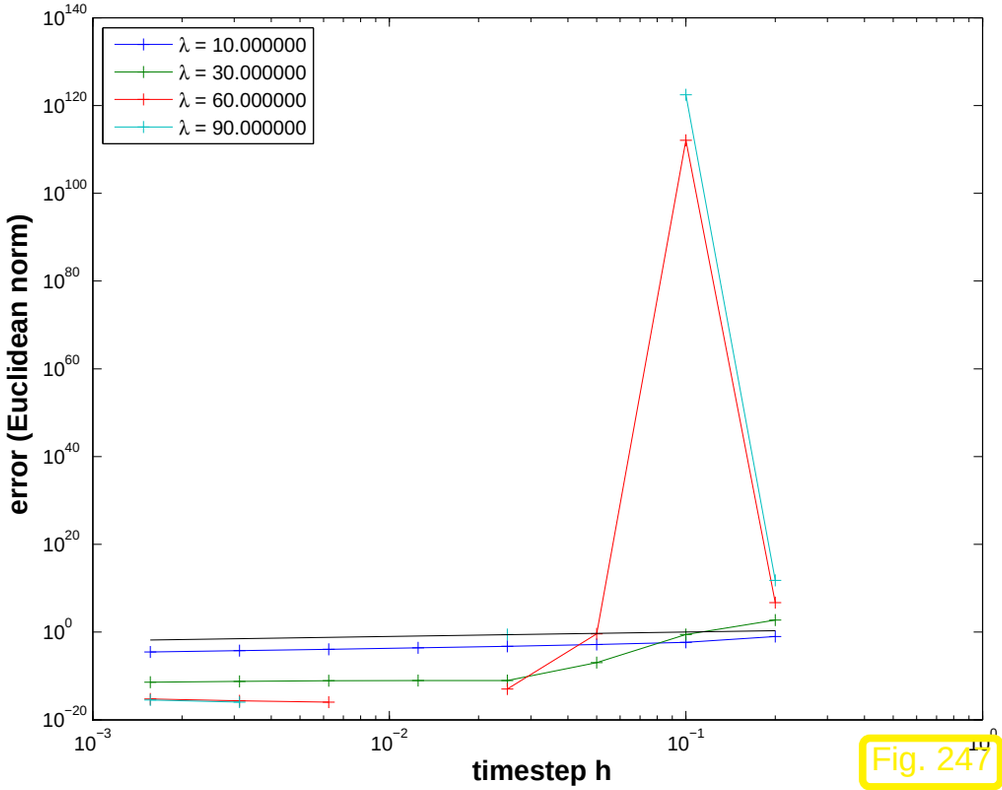


Fig. 247

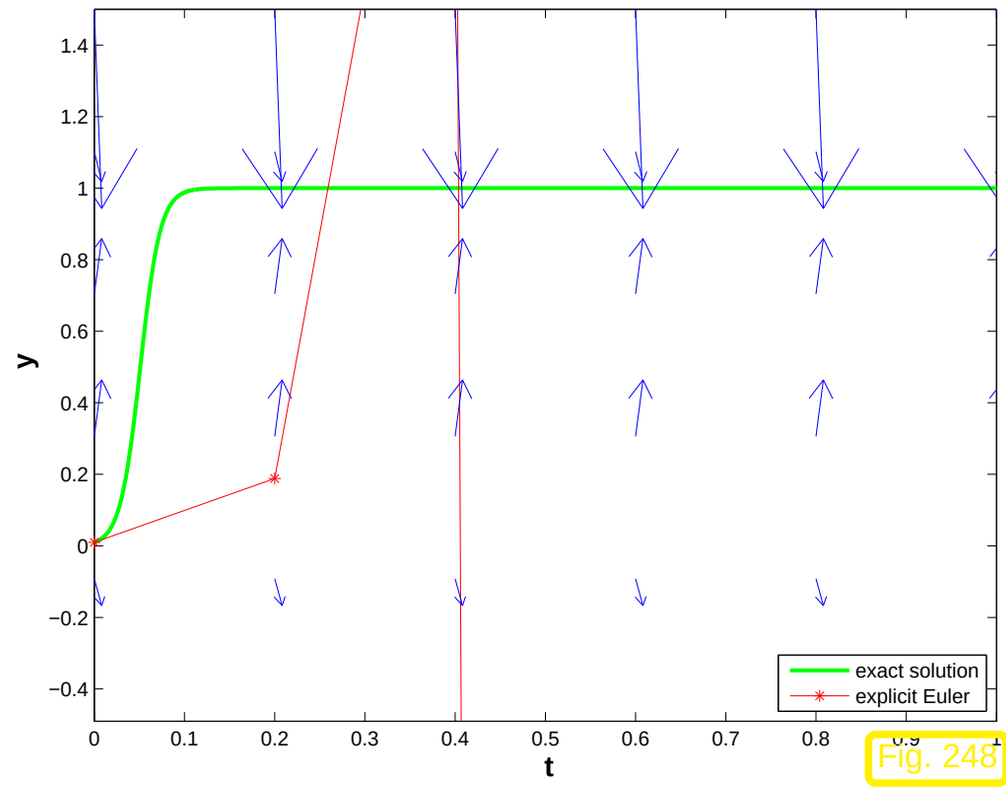


Fig. 248

λ large: blow-up of y_k for large timestep h

$\lambda = 90$: — $\hat{=}$ $y(t)$, — $\hat{=}$ Euler polygon

$\diamond$

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

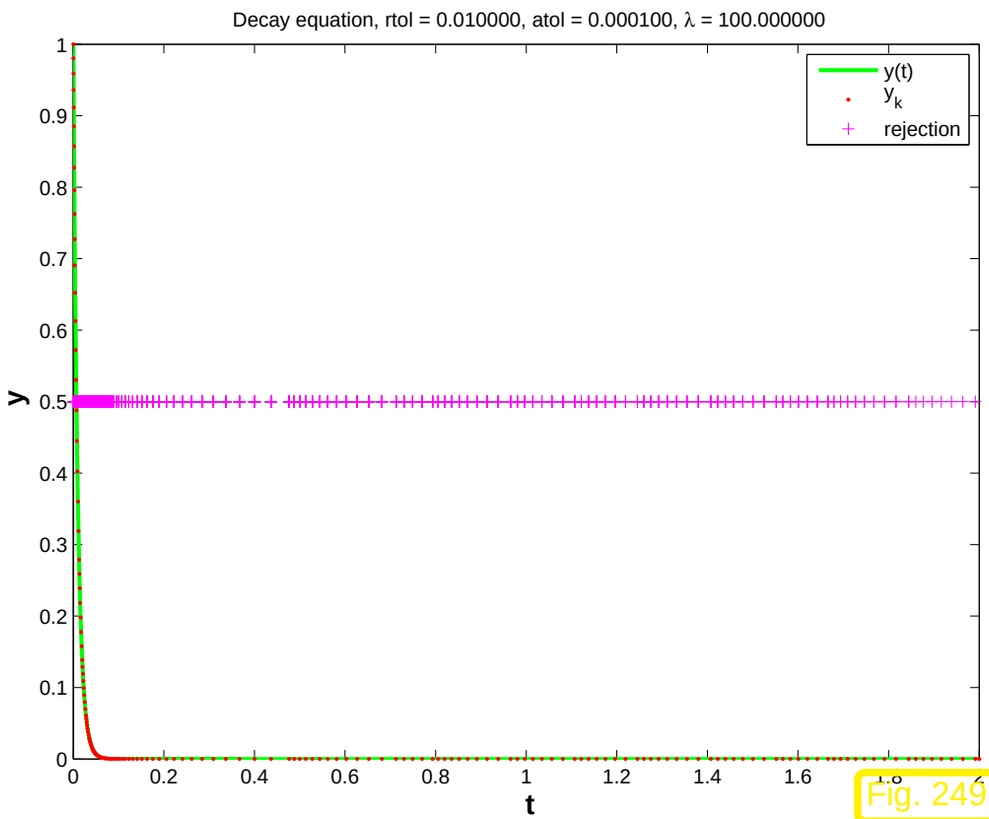


Fig. 249

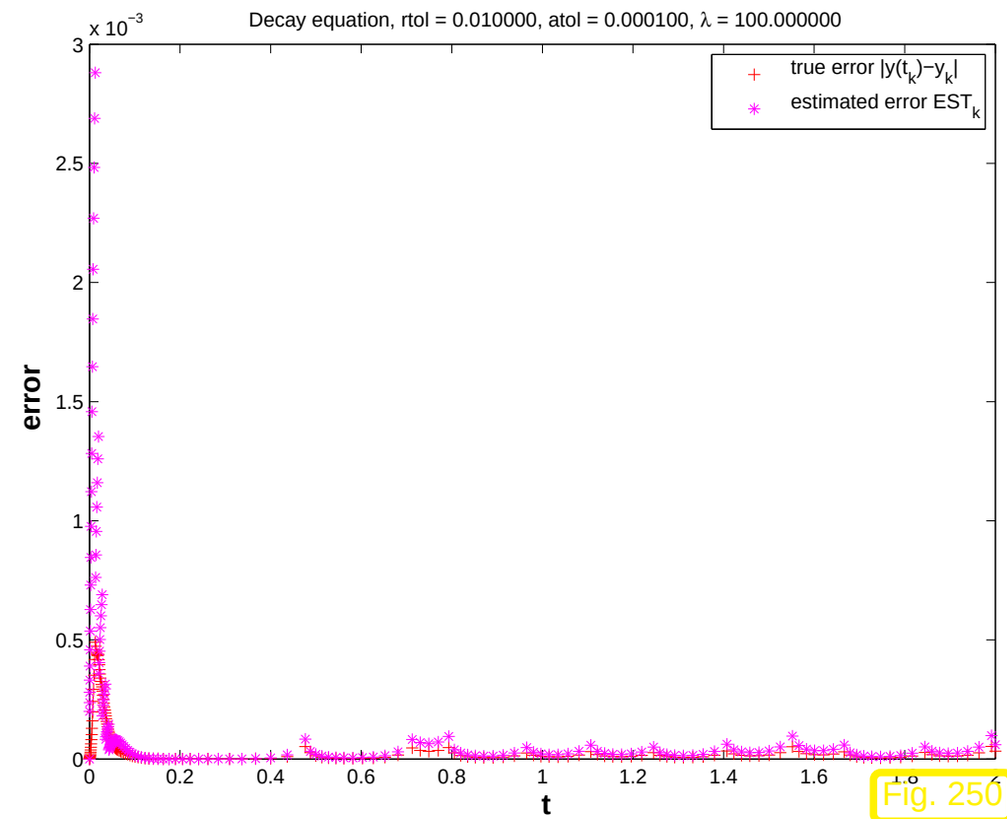


Fig. 250

◇

Example 13.1.5 (Explicit trapzoidal rule for decay equation). $\rightarrow$ [13, Ex. 11.29]

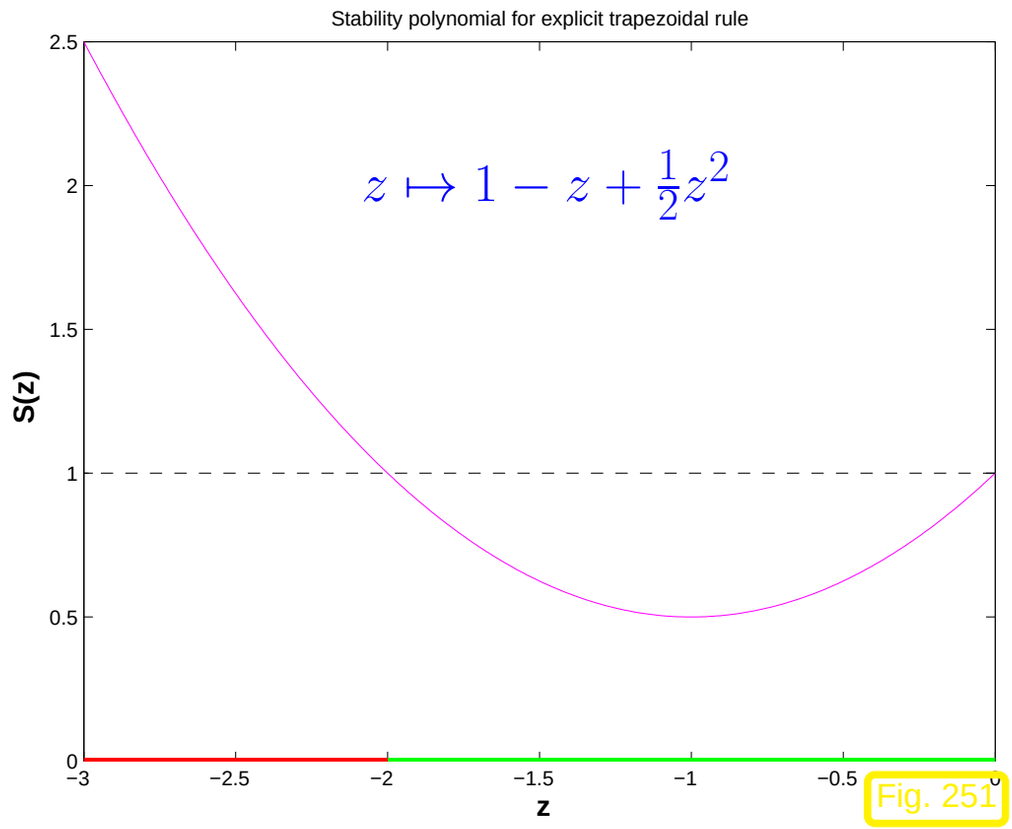


Fig. 251

$\diamond$

R. Hiptmair

rev 38355, December 23, 2010

Theorem 13.1.11 (Stability function of explicit Runge-Kutta methods)**.** *→ [35, Thm. 77.2], [51, Sect. 11.8.4]*

The discrete evolution Ψ_λ^h of an explicit s-stage Runge-Kutta single step method (→ 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*

$$\Psi_\lambda^h = \underbrace{1 + z\mathbf{b}^T\left(\mathbf{I} - z\mathfrak{A}\right)^{-1}\mathbf{1}}_{\text{stability function } S(z)} = \det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T)\,, \quad z := \lambda h\,, \quad \mathbf{1} = (1,\dots,1)^T \in \mathbb{R}^s\,.$$

13.2 Stiff problems [51, Sect. 11.10]

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, $v := \dot{u}$

$$\underbrace{\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix}}_{=:\dot{\mathbf{y}}} = \underbrace{\begin{pmatrix} 0 & 1 \\ -\beta & -\alpha \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} - \begin{pmatrix} 0 \\ g(t) \end{pmatrix}}_{=:\mathbf{f}(t,\mathbf{y})} \ .$$

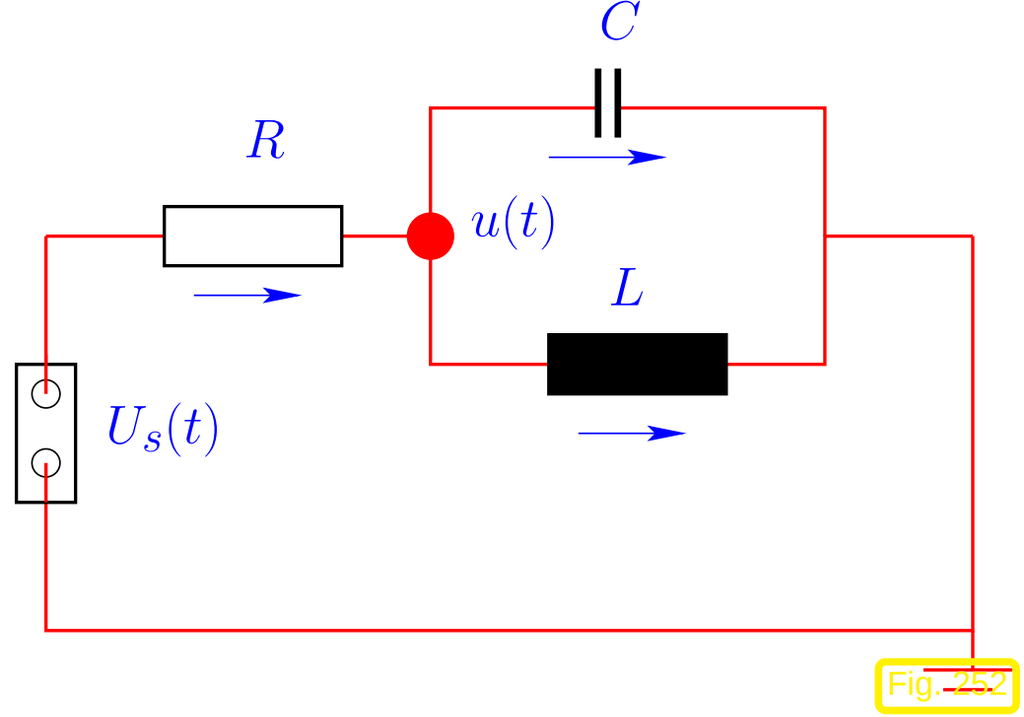


Fig. 252

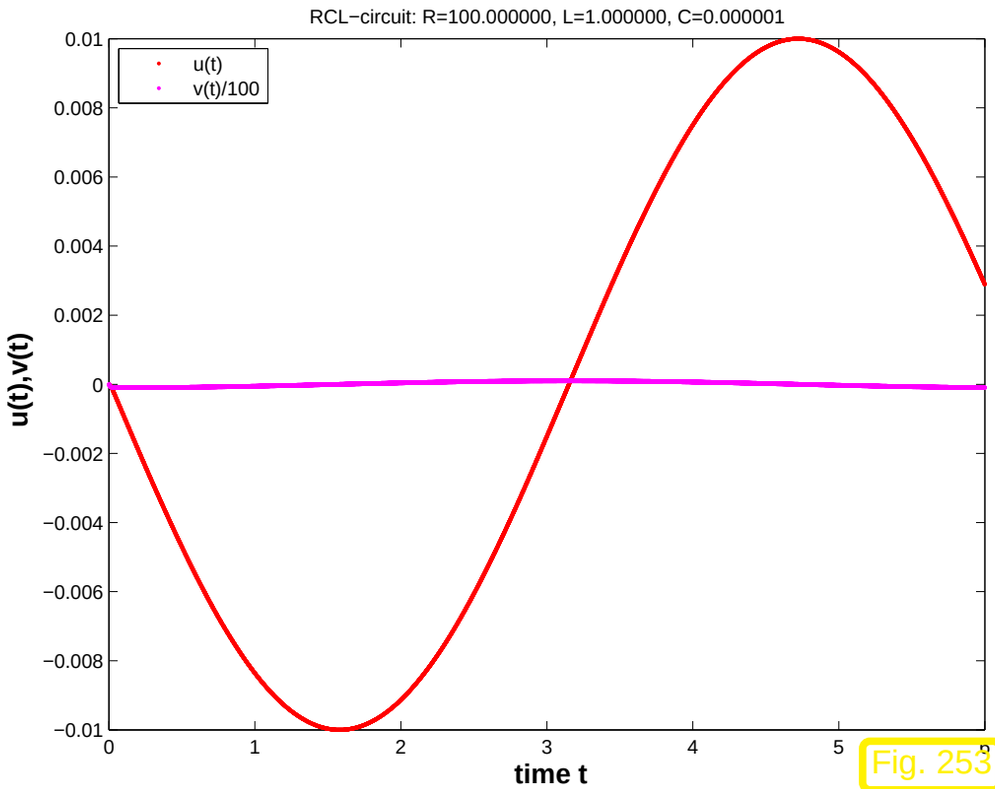


Fig. 253

$R = 100\Omega$, $L = 1\mathrm{H}$, $C = 1\mu\mathrm{F}$, $U_s(t) = 1\mathrm{V}\sin(t)$, $u(0) = v(0) = 0$ (“switch on”)

`ode45` statistics:

```
17897 successful steps
1090 failed attempts
113923 function evaluations
```


R. Hiptmair

rev 38355, December 23, 2010

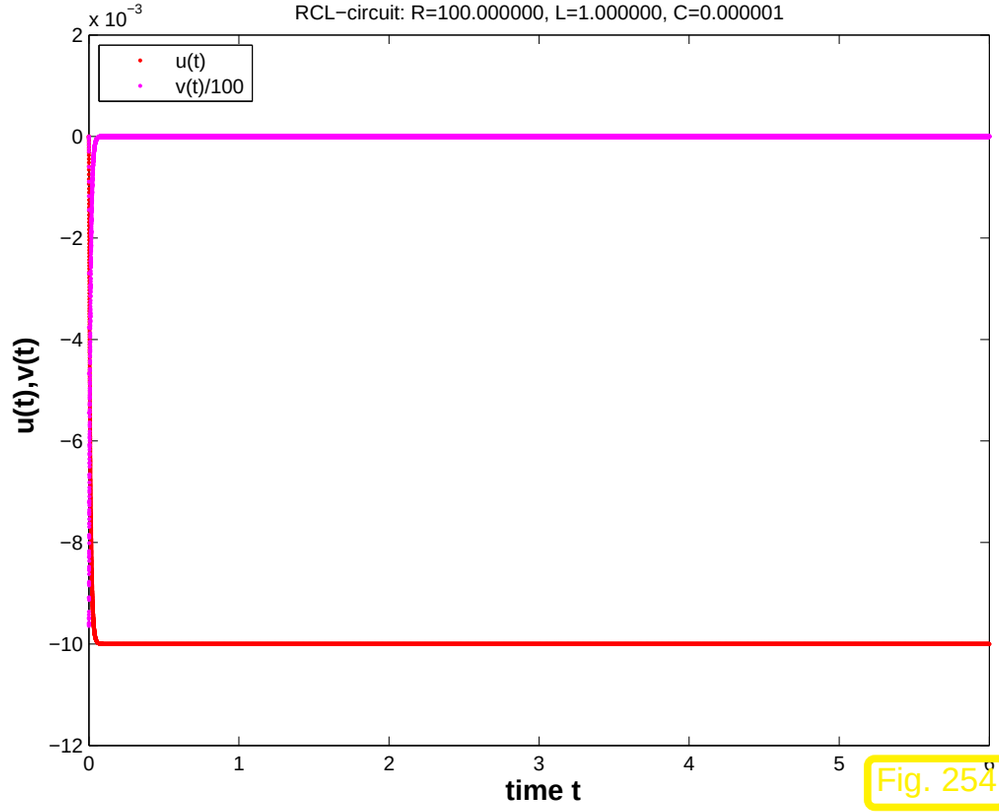


Fig. 254

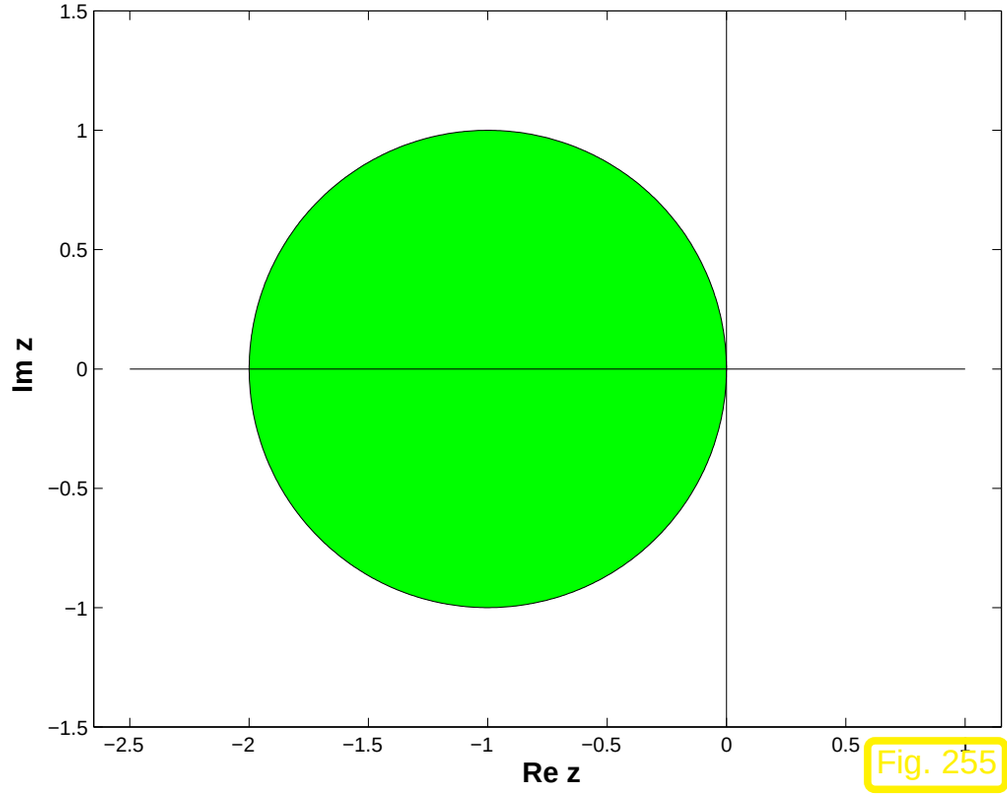
$R = 100\Omega$, $L = 1\mathrm{H}$, $C = 1\mu\mathrm{F}$, $U_s(t) = 1\mathrm{V}$, $u(0) = v(0) = 0$ (“switch on”)

`ode45` statistics:

```
17901 successful steps
1210 failed attempts
114667 function evaluations
```

◇

Example 13.2.7 (Explicit Euler method for damped oscillations).



◁ $\{z \in \mathbb{C}\colon\ |1+z| < 1\}$

◇

R. Hiptmair

rev 38355, December 23, 2010

Can we predict this kind of difficulty ?

Example 13.2.8 (Kinetics of chemical reactions). → [35, Ch. 62]

reaction:
$$\underbrace{\mathrm{A}+\mathrm{B} \underset{k_1}{\overset{k_2}{\rightleftharpoons}} \mathrm{C}}_{\text{fast reaction}} \quad , \quad \underbrace{\mathrm{A}+\mathrm{C} \underset{k_3}{\overset{k_4}{\rightleftharpoons}} \mathrm{D}}_{\text{slow reaction}} \tag{13.2.9}$$

Vastly different reaction constants: $\boxed{k_1, k_2 \gg k_3, k_4}$

► If $c_A(0) > c_B(0)$ ➢ 2nd reaction determines overall long-term reaction dynamics

Mathematical model: ODE involving concentrations $\mathbf{y}(t) = (c_A(t), c_B(t), c_C(t), c_D(t))^T$

$$\dot{\mathbf{y}} := \frac{d}{dt}\begin{pmatrix} c_A \\ c_B \\ c_C \\ c_D \end{pmatrix} = \mathbf{f}(\mathbf{y}) := \begin{pmatrix} -k_1 c_A c_B + k_2 c_C - k_3 c_A c_C + k_4 c_D \\ -k_1 c_A c_B + k_2 c_C \\ k_1 c_A c_B - k_2 c_C - k_3 c_A c_C + k_4 c_D \\ k_3 c_A c_C - k_4 c_D \end{pmatrix} . \tag{13.2.10}$$

R. Hiptmair

rev 38355, December 23, 2010

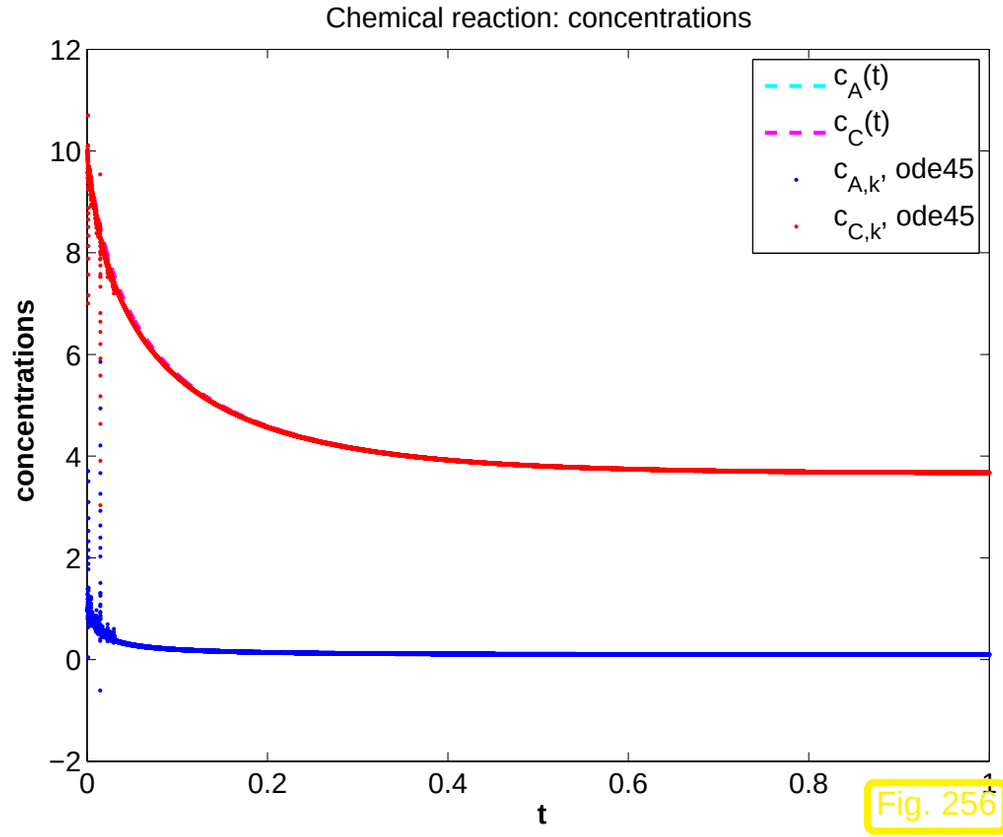


Fig. 256

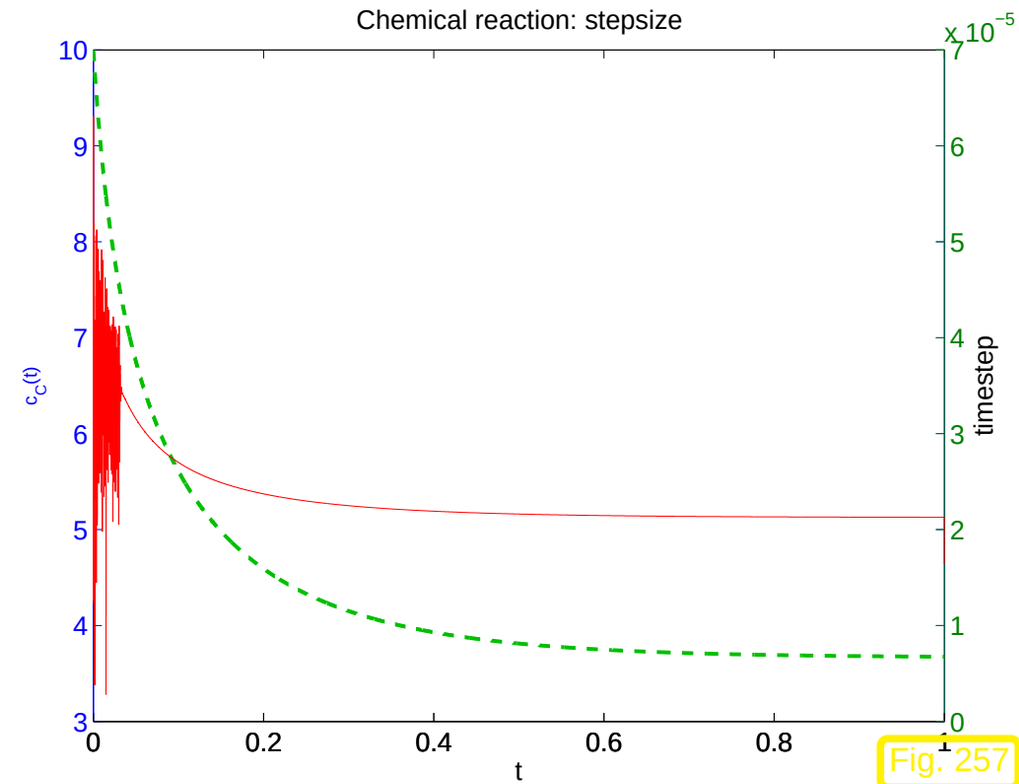


Fig. 257

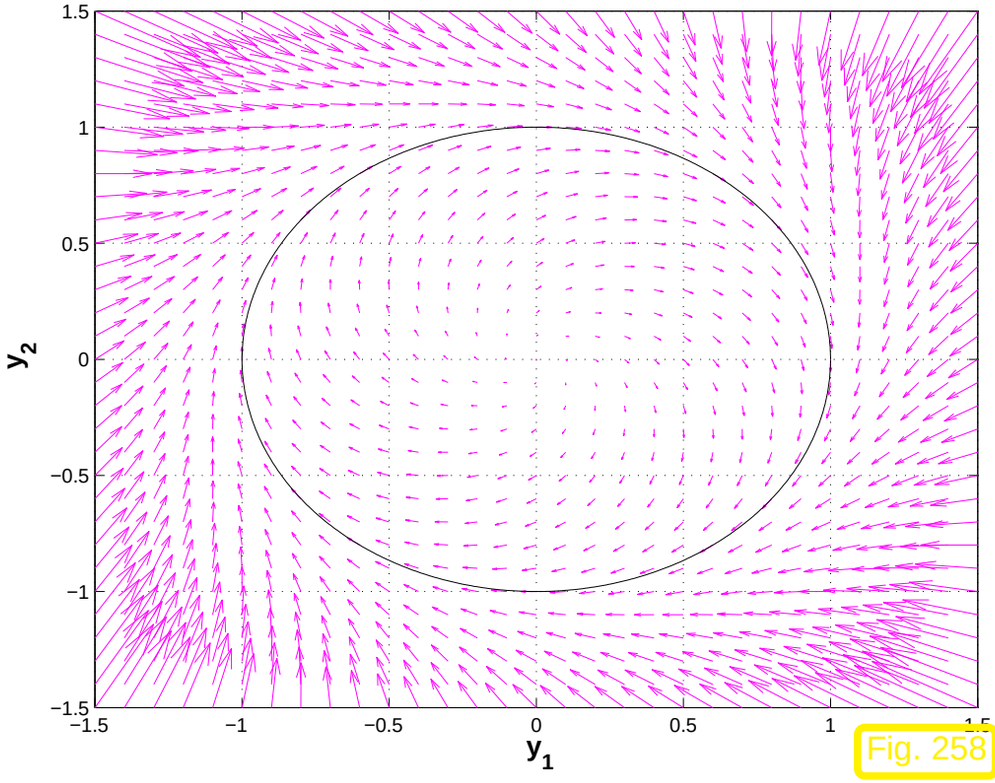
◇

Example 13.2.12 (Strongly attractive limit cycle).

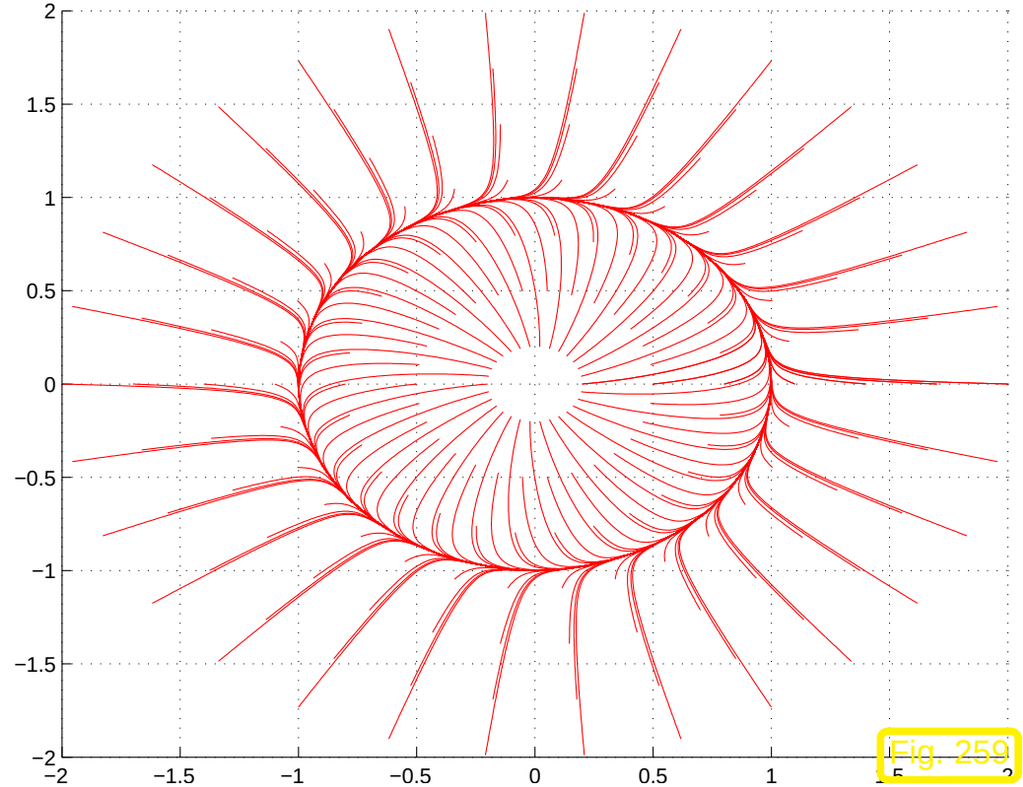
Autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$

$$\mathbf{f}(\mathbf{y}) := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \mathbf{y} + \lambda(1 - \|\mathbf{y}\|^2)\,\mathbf{y}\ , \tag{13.2.13}$$

on state space $D = \mathbb{R}^2 \setminus \{0\}$



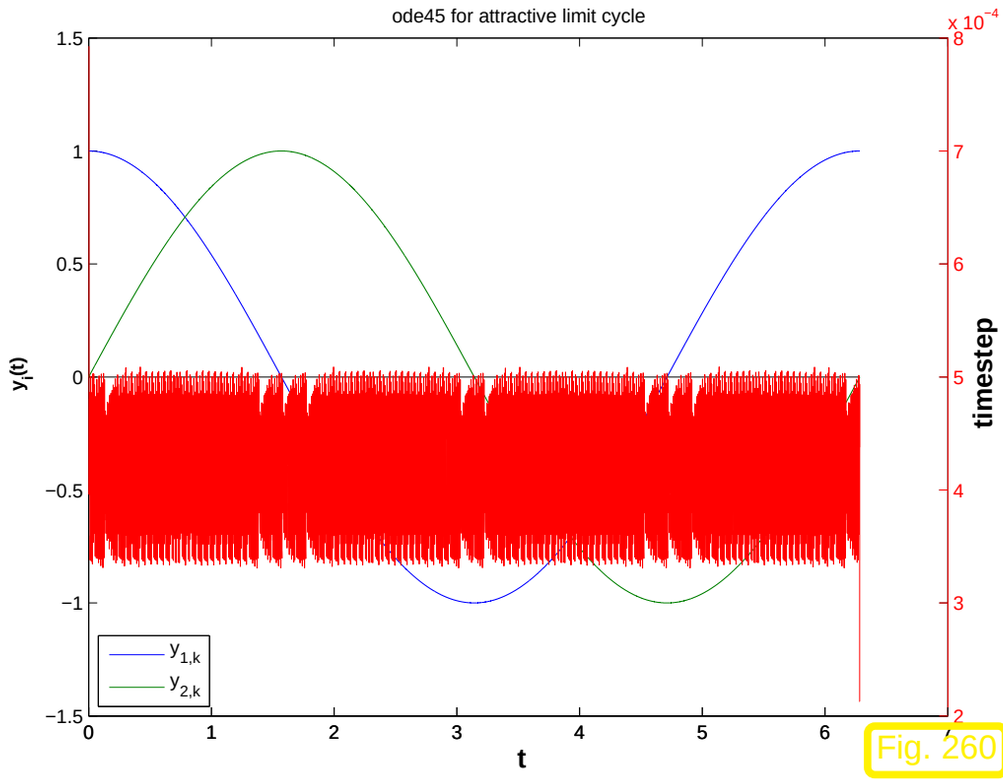
vectorfield $\mathbf{f}$ ($\lambda = 1$)



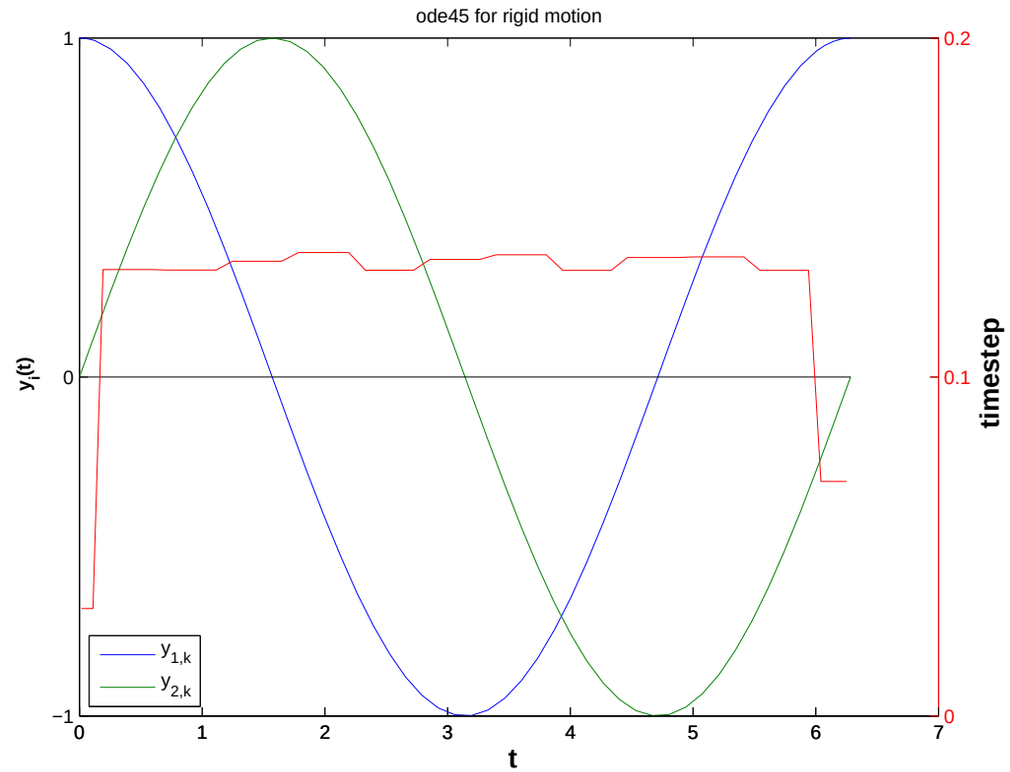
solution trajectories ($\lambda = 10$)

R. Hiptmair

rev 38355, December 23, 2010



many (3794) steps ($\lambda = 1000$)



accurate solution with few steps ($\lambda = 0$)

◇

Notion 13.2.16 (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

13.3 Implicit Runge-Kutta methods [13, Sect. 11.6.2], [51, Sect. 11.

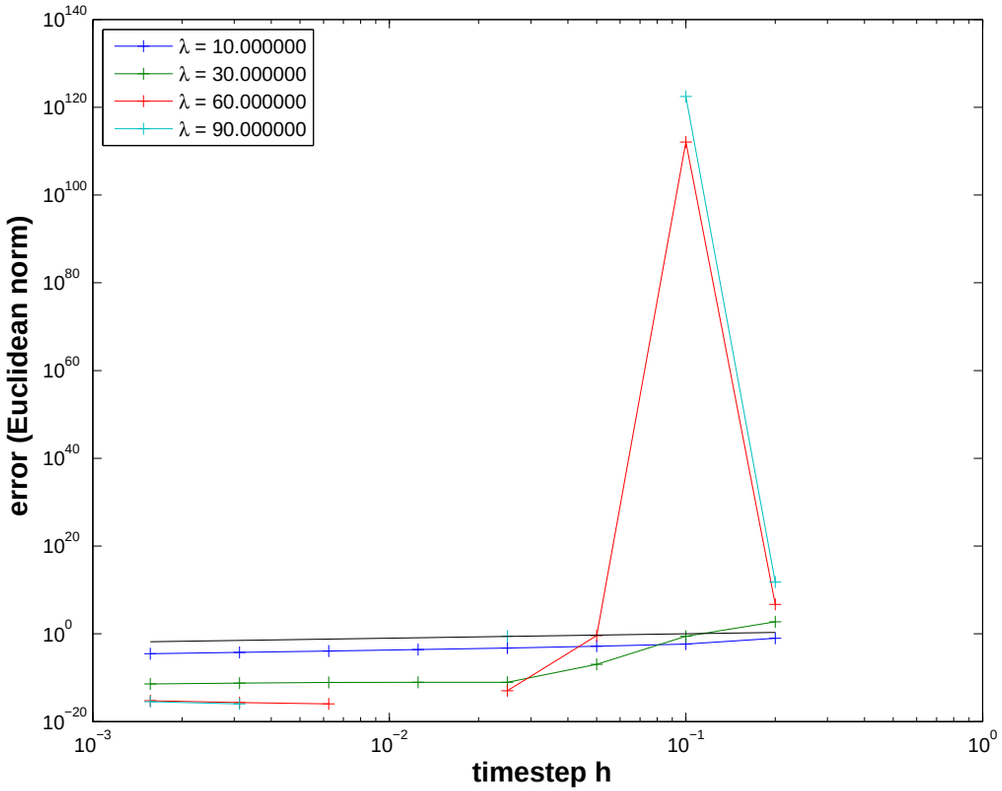
Example 13.3.1 (Implicit Euler timestepping for decay equation).

◇

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)



λ large: blow-up of y_k for large timestep h

Implicit Euler method (12.2.8)

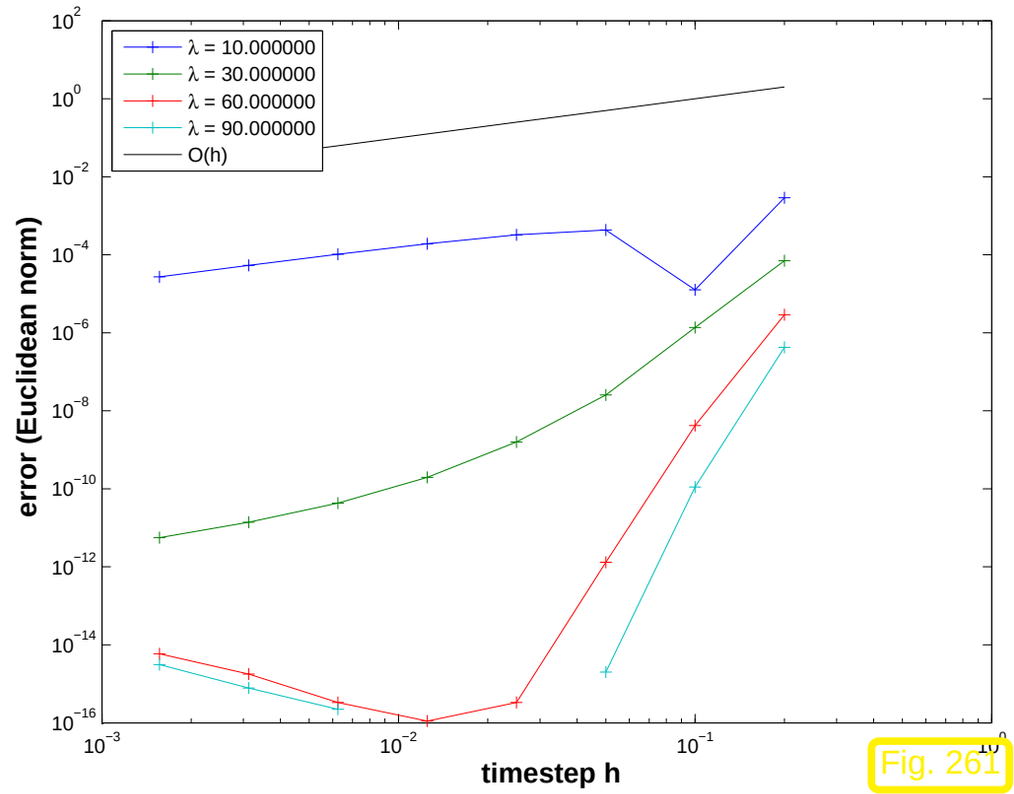


Fig. 261

λ large: stable for all timesteps h **!**

◇

R. Hiptmair

rev 38355, December 23, 2010

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*

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j)\ , \quad i = 1,\dots,s \quad , \quad \mathbf{y}_1 := \mathbf{y}_0 + h\sum_{i=1}^{s} b_i \mathbf{k}_i\ .$$

As before, the $\mathbf{k}_i \in \mathbb{R}^d$ *are called increments.*

Shorthand notation for Runge-Kutta methods

Butcher scheme

$$\triangleright \quad \begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} := \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} . \qquad (13.3.6)$$

R. Hiptmair

rev 38355, December 23, 2010

Theorem 13.3.7 (Stability function of Runge-Kutta methods)**.**
The discrete evolution Ψ_λ^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*

$$\Psi_\lambda^h = \underbrace{1 + z\mathbf{b}^T\left(\mathbf{I} - z\mathfrak{A}\right)^{-1}\mathbf{1}}_{\text{stability function } S(z)} = \frac{\det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T)}{\det(\mathbf{I} - z\mathfrak{A})}\,, \quad z := \lambda h\,, \quad \mathbf{1} = (1,\dots,1)^T \in \mathbb{R}^s\,.$$

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

$$(i) \qquad \operatorname{Re} z < 0 \quad \Rightarrow \quad |S(z)| < 1\,, \tag{13.3.10}$$

$$(ii) \qquad \lim_{\operatorname{Re} z \to -\infty} S(z) = 0\,. \tag{13.3.11}$$

Example 13.3.15 (L-stable implicit Runge-Kutta methods)*.*

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

Implicit Euler method

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$

Radau RK-SSM, order 3

$$\begin{array}{c|ccc} \frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{4+\sqrt{6}}{10} & \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ 1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array}$$

Radau RK-SSM, order 5

$\diamond$

R. Hiptmair

rev 38355, December 23, 2010

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

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 $\mathrm{err} = \max_{j=1,\dots,n} |y_j - y(t_j)|$

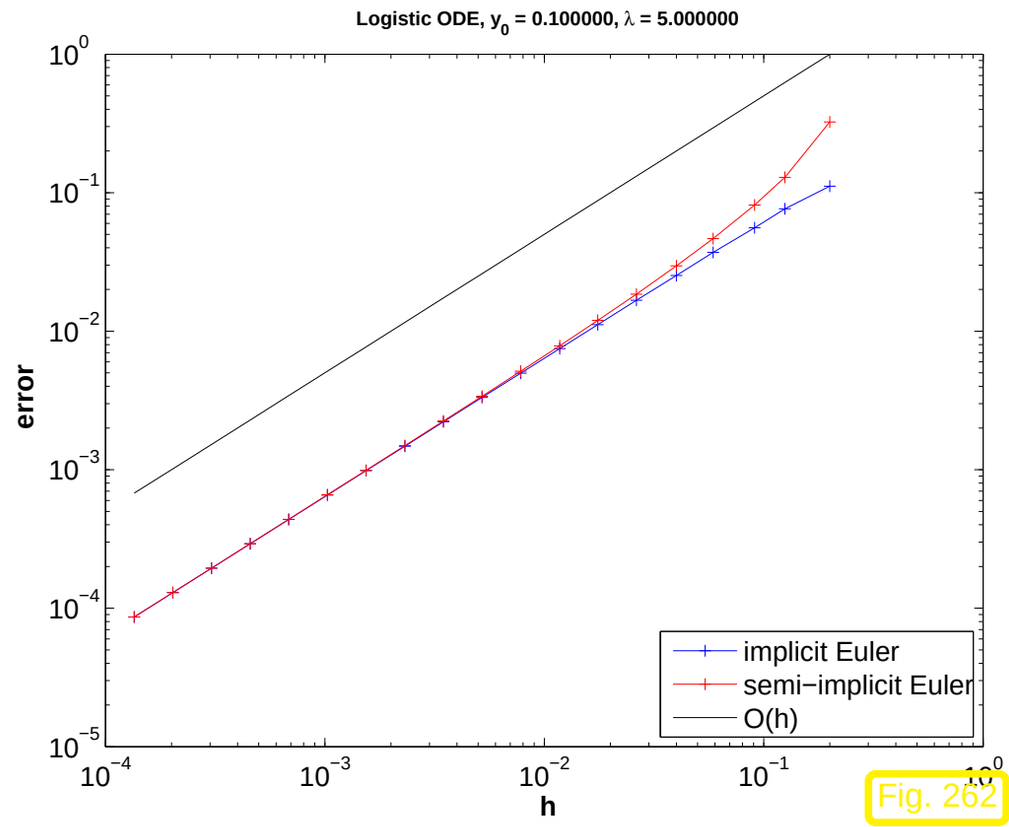
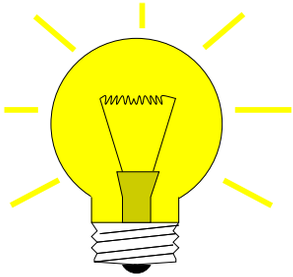


Fig. 262

◇



Idea: Use linearized increment equations for implicit RK-SSM

$$\mathbf{k}_i = \mathbf{f}(\mathbf{y}_0) + hD\mathbf{f}(\mathbf{y}_0)\left(\sum_{j=1}^{s} a_{ij}\mathbf{k}_j\right) \quad , \quad i = 1,\dots,s \; . \tag{13.4.2}$$

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):

$$(\mathbf{I} - ha_{ii}\mathbf{J})\mathbf{k}_i = \mathbf{f}(\mathbf{y}_0 + h\sum_{j=1}^{i-1}(a_{ij}+d_{ij})\mathbf{k}_j) - h\mathbf{J}\sum_{j=1}^{i-1} d_{ij}\mathbf{k}_j \; , \tag{13.4.3}$$

$$\mathbf{J} := D\mathbf{f}\big(\mathbf{y}_0 + h\sum_{j=1}^{i-1}(a_{ij}+d_{ij})\mathbf{k}_j\big) \; , \tag{13.4.4}$$

$$\mathbf{y}_1 := \mathbf{y}_0 + \sum_{j=1}^{s} b_j\mathbf{k}_j \; . \tag{13.4.5}$$

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:

$\Psi \mathrel{\hat{=}}$ RK-method of order 2

$\widetilde{\Psi} \mathrel{\hat{=}}$ RK-method of order 3

ode23s

integrator for **s**tiff IVP

△

R. Hiptmair

rev 38355, December 23, 2010

14 Structure Preservation [31]

14.1 Dissipative Evolutions

14.2 Quadratic Invariants

14.3 Reversible Integrators

14.4 Symplectic Integrators

R. Hiptmair

rev 38355, February 24, 2009

Outlook

Course 401-0674-00: Numerical Methods for Partial Differential Equations

Many fundamental models in science & engineering boil down to

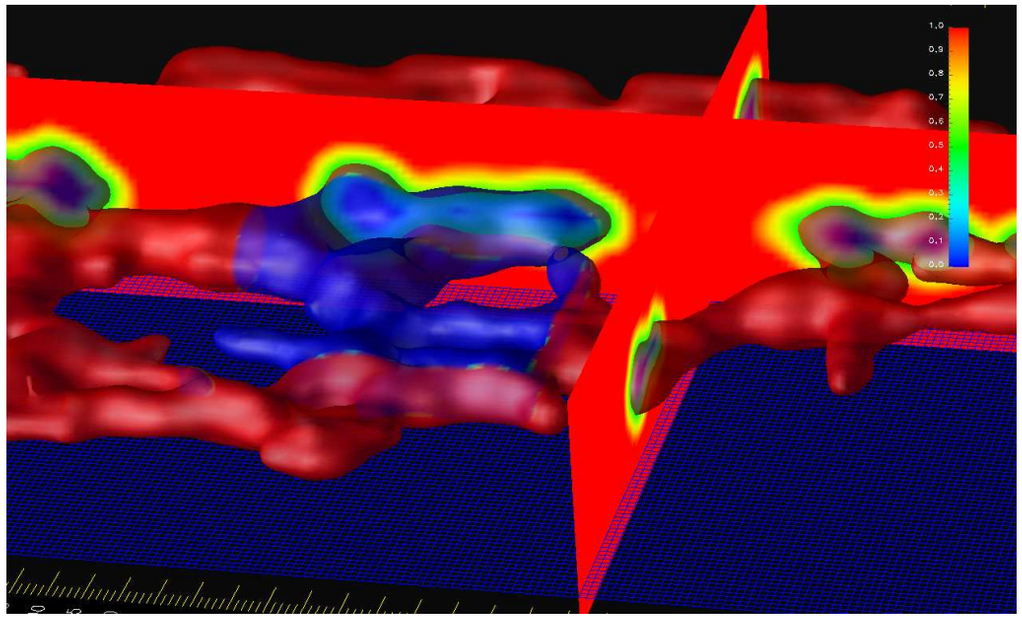
(initial) boundary value problems for partial differential equations (PDEs)

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:

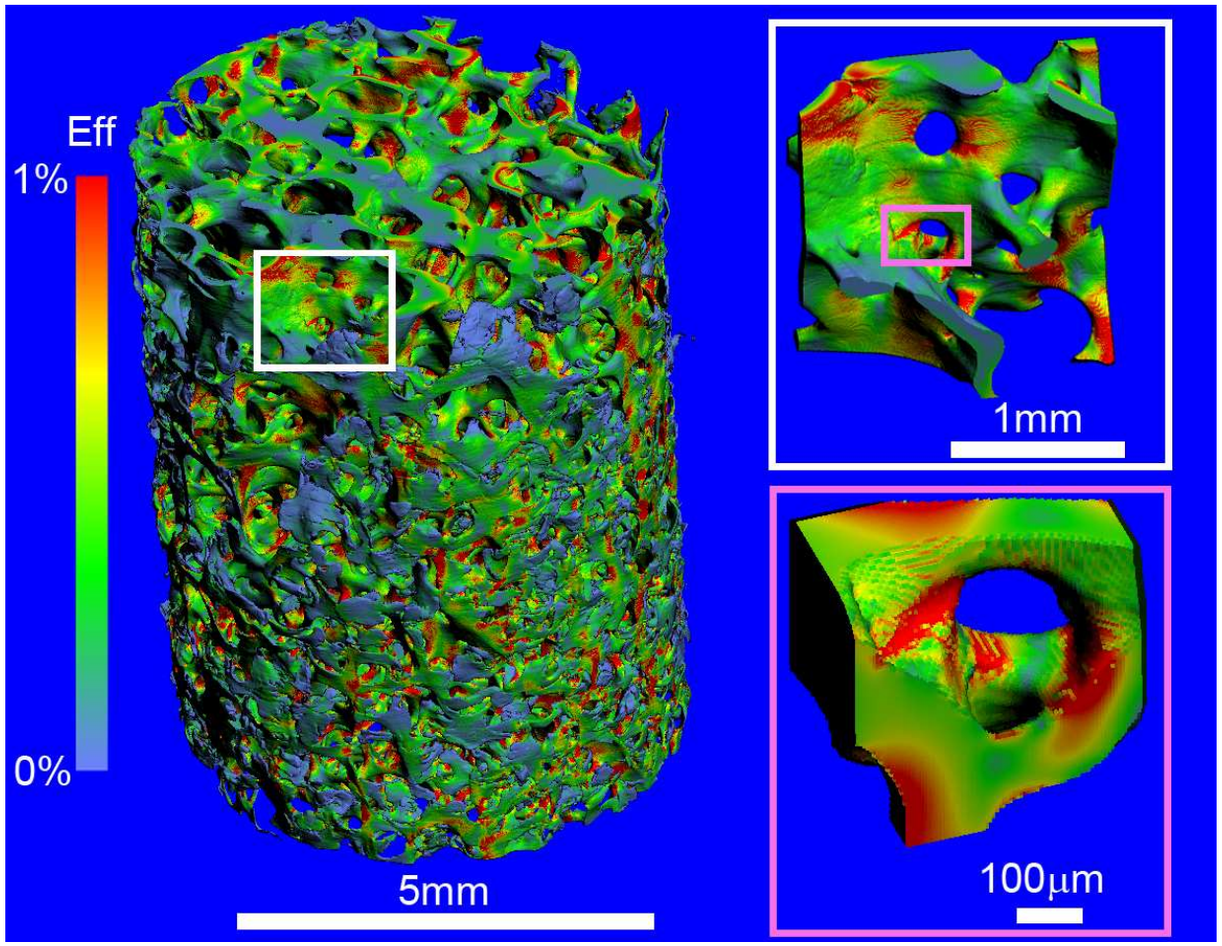
❶ **Stationary 2nd-order scalar elliptic boundary value problems**



Diffusion boundary value problem:

$$\begin{aligned} -\operatorname{div}(\mathbf{A}(\boldsymbol{x})\,\mathbf{grad}\,u(\boldsymbol{x})) &= f(\boldsymbol{x}) \quad \text{in } \Omega \subset \mathbb{R}^d\,, \\ u &= g \qquad \text{on } \partial\Omega\,. \end{aligned}$$

◁ 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)

❷ Singularly perturbed elliptic boundary value problems

Stationary pollutant transport in water: find concentration $u = u(\boldsymbol{x})$ such that

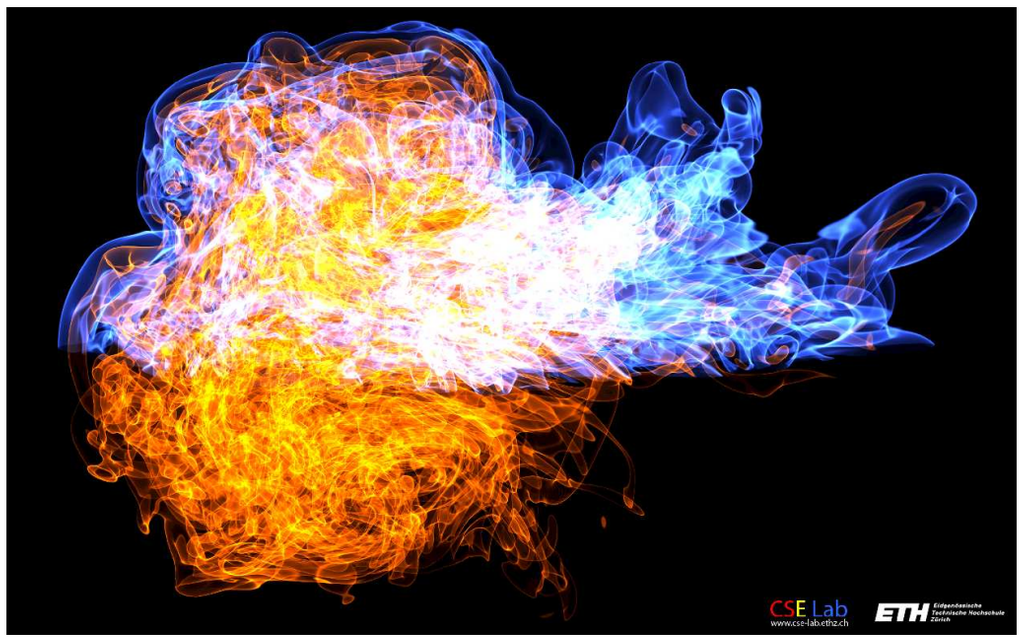
$$-\epsilon\Delta u + \mathbf{v}(\boldsymbol{x}) \cdot \mathbf{grad}\, u = 0 \quad \text{in } \Omega \quad , \quad u = g \quad \text{on } \partial\Omega \; .$$

❸ 2nd-order parabolic evolution problems

Heat conduction: find temperature $u = u(\boldsymbol{x}, t)$

$$\frac{\partial}{\partial t} u(\boldsymbol{x}, t) - \operatorname{div}(\mathbf{A}(\boldsymbol{x})\, \mathbf{grad}\, u(\boldsymbol{x}, t)) = 0 \quad \text{in } \Omega \times [0, T] \quad , \quad \begin{aligned} u(\cdot, t) &= g(t) \quad \text{on } \partial\Omega \ , \\ u(\cdot, 0) &= u_0 \quad \text{in } \Omega \ . \end{aligned}$$

❹ Viscous fluid flow problems



Stokes equations:

$$\begin{aligned} -\Delta \mathbf{u} + \mathbf{grad}\, p &= \mathbf{f} \quad \text{in } \Omega \ , \\ \operatorname{div} \mathbf{u} &= 0 \quad \text{in} \Omega \ , \\ \mathbf{u} &= 0 \quad \text{on } \partial\Omega \ . \end{aligned}$$

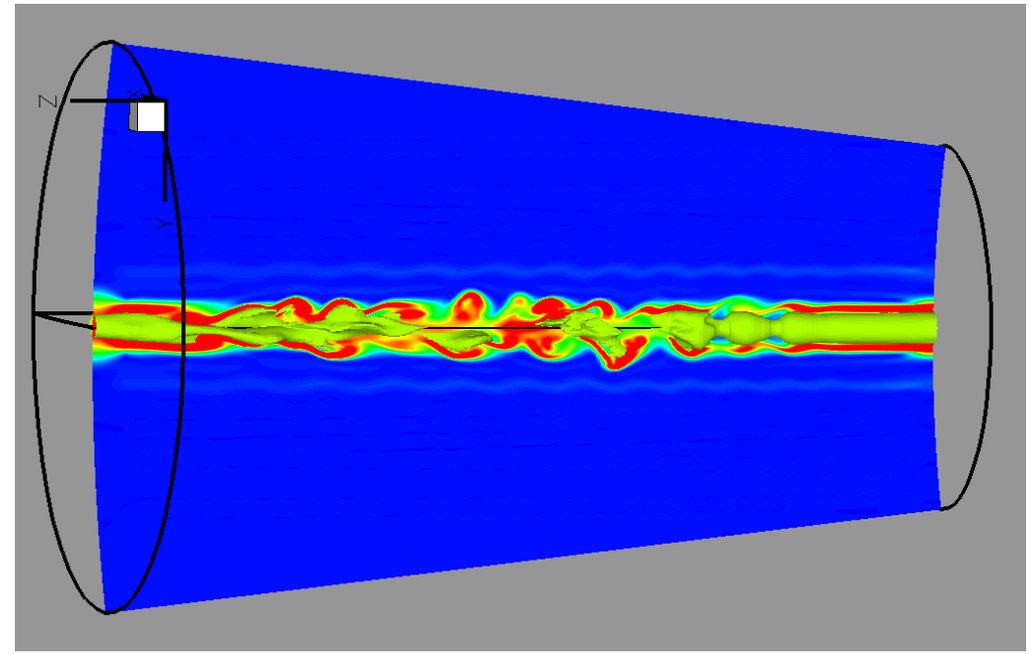
◁ Vortex ring in flow at $\mathtt{Re} = 7500$, (P. Koumoutsakos, D-INFK, ETH Zürich)

❺ Conservation laws

1D scalar conservation law with flux f:

$$\begin{aligned} \frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}(f(u)) &= 0 && \text{in } \mathbb{R}\times\mathbb{R}^+ \,, \\ u(x,0) &= u_0(x) && \text{for } x \in \mathbb{R} \,. \end{aligned}$$

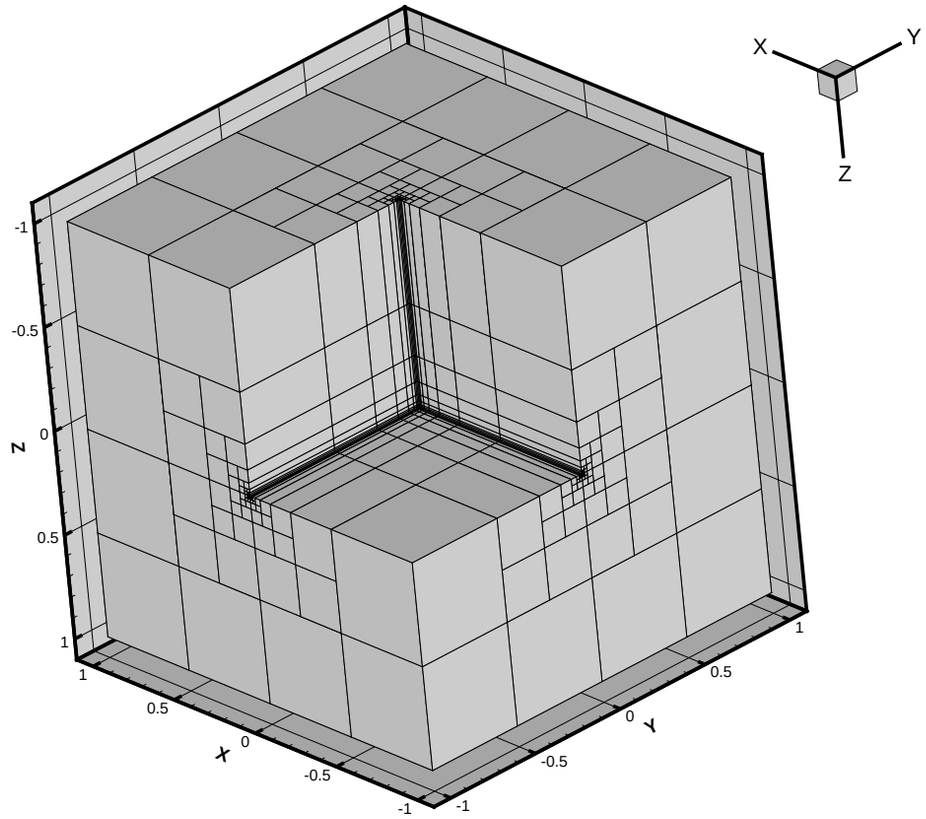
Inviscid fluid flow in 3D (SAM, D-MATH, ETH Zürich) ▷



❻ Adaptive finite element methods

R. Hiptmair

rev 38355, November 10, 2009



◁ Adaptive FEM for diffusion problem:

Geometrically graded mesh at re-entrant corner (SAM, D-MATH, ETH Zürich)

❼ Multilevel preconditioning

R. Hiptmair

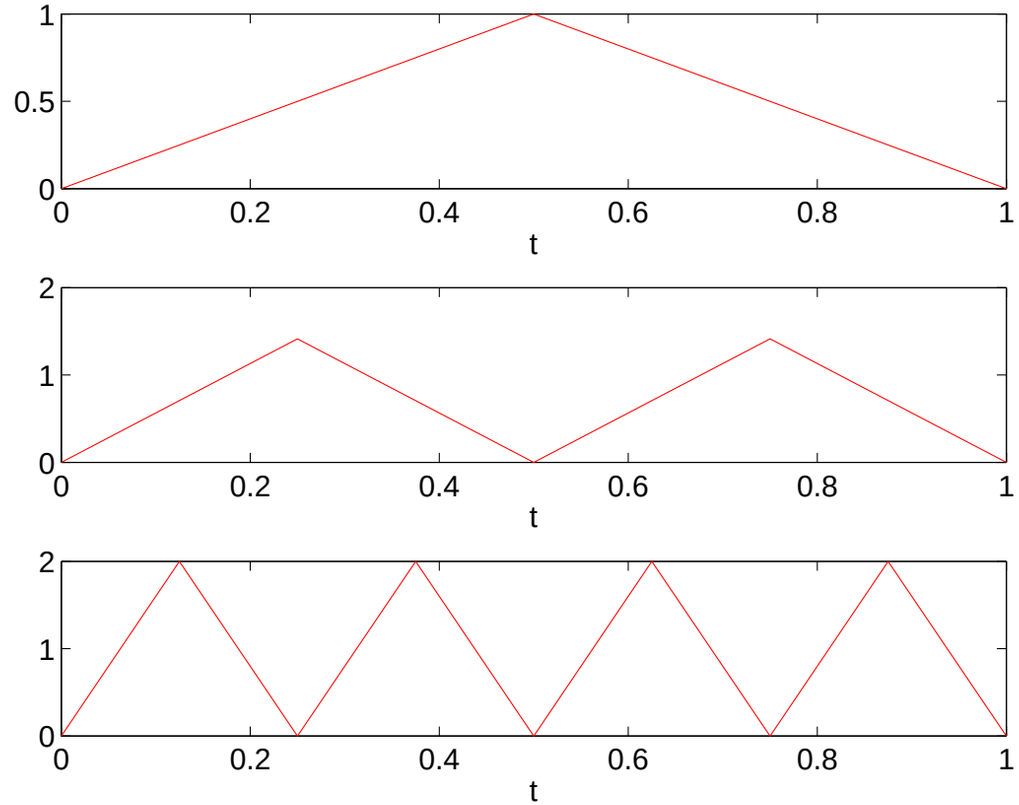
rev 38355, November 10, 2009

FEM, FD, FV

Huge sparse systems of equations

Efficient preconditioners required

1D hierarchical basis ▷



R. Hiptmair

rev 38355, November 10, 2009

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

Course: Parallel Computing for Scientific Simulations

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 ($\rightarrow$ 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. Symbgolic 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. Aanal., 27 (1990), pp. 1322–1331.

Index

R. Hiptmair

rev 38355, November 10, 2009

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

$C^0_{\mathrm{pw}}(I) \hat{=}$ 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

$\mathrm{eps} \hat{=}$ machine precision, 145

$\mathrm{Eig}_{\mathbf{A}}(\lambda) \hat{=}$ eigenspace of $\mathbf{A}$ for eigenvalue λ, 583

$\mathrm{Im}(\mathbf{A}) \hat{=}$ range/column space of matrix $\mathbf{A}$, 730

$\mathrm{Ker}(\mathbf{A}) \hat{=}$ nullspace of matrix $\mathbf{A}$, 730

$\mathcal{K}_l(\mathbf{A},\mathbf{z}) \hat{=}$ Krylov subspace, 526

$\|\mathbf{Ax}-\mathbf{b}\|_2 \to \min \hat{=}$ minimize $\|\mathbf{Ax}-\mathbf{b}\|_2$, 772

$\|\mathbf{A}\|_F^2$, 757

List of Definitions

R. Hiptmair

rev 38355, November 10, 2009

Examples and Remarks

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