Numerical Methods

for Computational Science and Engineering

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

Prof. Ralf Hiptmair and Dr. Vasile Gradinaru

Draft version February 19, 2011, Subversion 30401

http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE10.pdf

Always under construction

Contents

R. Hiptmair

rev 30401, February 19, 2011

About this course

Focus

▷ on algorithms (principles, scope, and limitations)

▷ on implementation (efficiency, stability)

▷ 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)

Contents

R. Hiptmair

rev 30401, September 23, 2010

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

Indispensable:	Learning by doing (➔ exercises)

Reading instructions

This course materials are neither a textbook nor 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.

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.

- W. DAHMEN AND A. REUSKEN, *Numerik für Ingenieure und Naturwissenschaftler*, Springer, Heidelberg, 2006.
- M. HANKE-BOURGEOIS, *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*, Mathematische Leitfäden, B.G. Teubner, Stuttgart, 2002.
- C. MOLER, *Numerical Computing with MATLAB*, SIAM, Philadelphia, PA, 2004.
- P. DEUFLHARD AND A. HOHMANN, *Numerische Mathematik. Eine algorithmisch orientierte Einführung*, DeGruyter, Berlin, 1 ed., 1991.

Essential prerequisite for this course is a solid knowledge in linear algebra. Familiarity with the topics covered in the first semester courses (see the textbook [39] and the lecture notes [23]) is taken for granted.

General information

Lecturer:	Prof. Ralf Hiptmair,	HG G 58.2,	☎ 044 632 3404,	hiptmair@sam.math.ethz.ch
Assistants:	Andrea Moiola	, HG J 45,	☎ 044 632 3449,	andrea.moiola@sam.math.ethz.ch
	Robert Carnecky	, CAB G 66.2,	☎ 044 632 5527,	crobi@inf.ethz.ch
	Robert Gantner	,		gantnerr@student.ethz.ch
	Andreas Noever	,		anoever@student.ethz.ch
	Jonas Sukis	,		sukysj@student.ethz.ch

Classes:	Mon, 8.15-10.00 (HG G3), Thu, 10.15-11.55 (HG G5)
Tutorials:	Mon, 10.15-11.55 (HG E 21)
	Mon, 13.15-15 (ETF E 1, LFO C 13)
	Wed, 8.15-10.00 (HG E 1.2)
	Thu, 8.15-10.00 (HG G 26.3, HG G 26.5)

Consulting hours:	Tue 12-13	Robert Carnecky
	Wed 12-13	Andrea Moiola
	Thu 12-13	Robert Gantner / Andreas Noever / Jonas Sukis

They will take place in the corridor in front of HG J 54.

Assignments

- The assignment sheets will be uploaded on the class webpage 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
  - take part in the tutorial classes. The attendance is **mandatory**: at most two tutorial classes can be missed during the semester.
  - contribute actively during the tutorial class. Every student will be asked to present the solution of a few exercises at the **blackboard**.
  - make a serious effort to solve the exercises and hand in the attempted solutions. In order to obtain the “Testat” at least **33%** of the homework problems have to be solved and handed in.

Self-grading

For each exercise sheet the students have to state which (sub)problems they have been 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.

Website: `http://www.math.ethz.ch/education/bachelor/lectures/hs2010/math/nummath_cs`

Examination

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

  Thu, Feb 10, 2011, 09:00 – 12:00

- Dry-run for computer based examination:

  Mon, Jan ???, 2011, 09:00, registration via course website

- Pre-exam question session:

  Mon, Jan 31, 2011, 8:15-10:00, HG G 5

- Subject of examination:
  - Alll topics of Chapters 1 through 12, which have been addressed in class or in a homework problem.

- Homework problems on sheet 1 through 13.

- Lecture documents will be available as PDF during the examination. The corresponding final version of the lecture documents will be made available on Jan 31, 2011.
- The exam questions will be asked both in German and in English.

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,

R. Hiptmair

rev 30401, January 4, 2011

- 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: 15.11.2010

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)

R. Hiptmair

rev 30401, January 4, 2011

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
Andrea Moiola	AMO
Robert Carnecky	ROC
Robert Gantner	ROG
Andreas Noever	ANO
Jonas Sukis	JOS

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”):

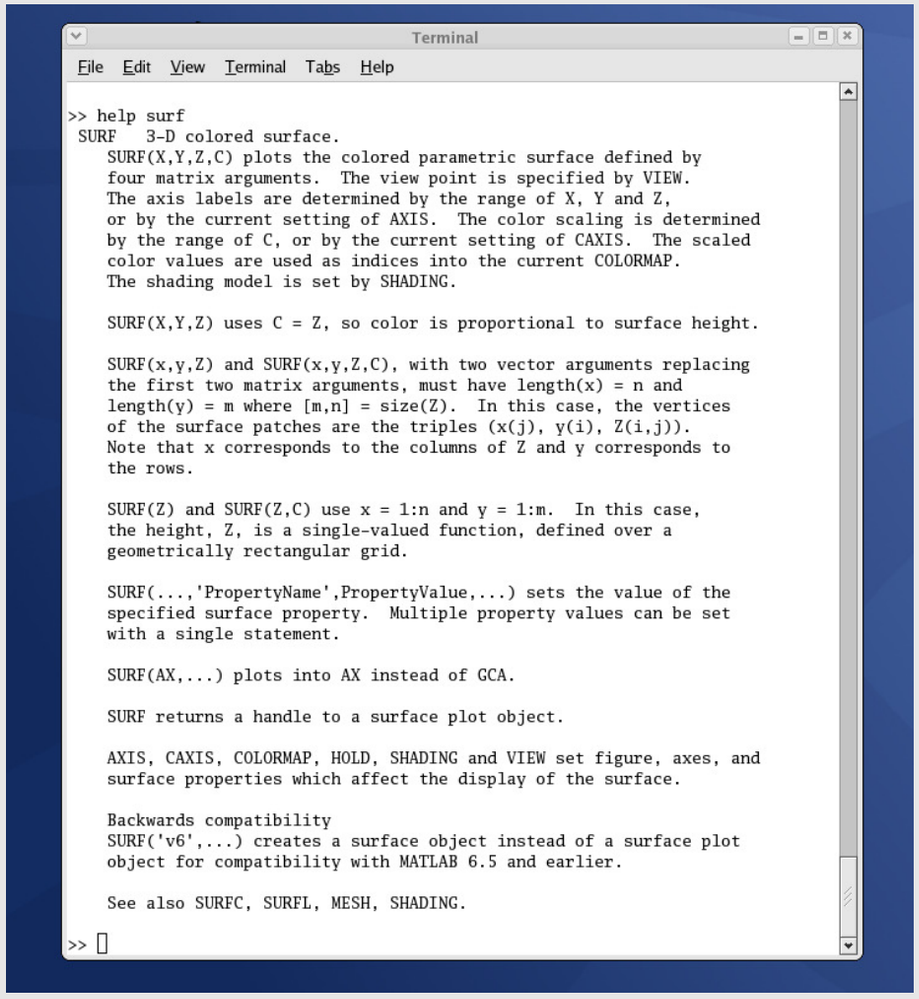
- full fledged high level *programming language* (for numerical algorithms)
- integrated *programming environment*
- versatile collection of *numerical libraries*

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

R. Hiptmair

rev 30401, December 16, 2010

- Numerical algorithms
- NUMAWWW

1 Computing with Matrices and Vectors [39, Sect. 2.1&2.2], [23, Sect. 2.1&2.2]

The implementation of most numerical algorithms relies on array type data structures modelling concepts from linear algebra (matrices and vectors). Related information can be found in [20, Ch. 1].

R. Hiptmair

rev 30401, February 24, 2009

Learning outcomes for this chapter:

- Understanding of the concepts of computational effort/cost and asymptotic complexity.
- Ability to determine the (asymptotic) computational effort for a concrete algorithm.
- Ability to manipulate simple expressions involving matrices and vectors in order to reduce the computational cost for their evaluation.
- Knowledge about BLAS and the benefit of using BLAS routines.

1.1 Notations

1.1.1 Vectors

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

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

- Default in this lecture: vectors $=$ column vectors

$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{K}^n$	$(x_1 \cdots x_n)$
column vector	row vector

  vector space of column vectors with n components

  ✎ notation for column vectors: **bold** small roman letters, e.g. $\mathbf{x}, \mathbf{y}, \mathbf{z}$

- Initialization of vectors in MATLAB:

column vectors `x = [1;2;3];`

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

- Transposing: $\begin{cases} \text{column vector} \mapsto \text{row vector} \\ \text{row vector} \mapsto \text{column vector} \end{cases}$.

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

Transposing in MATLAB: `x_T = x';`

- Adressing vector components:

✎ notation: $\mathbf{x} = (x_1 \ldots x_n)^T \rightarrow x_i, i = 1, \ldots, n$

$\mathbf{x} \in \mathbb{K}^n \rightarrow (\mathbf{x})_i, i = 1, \ldots, n$

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

- Selecting sub-vectors:

✎ notation: $\mathbf{x} = (x_1 \ldots x_n)^T \succ (\mathbf{x})_{k:l} = (x_k, \ldots, x_l)^T, 1 \leq k \leq l \leq n$

MATLAB:

```
x(4:7)      <--> [x(4);x(5);x(6);x(7)],
x(7:-1:4)   <--> [x(7);x(6);x(5);x(4)],
x(3:2:10)   <--> [x(3);x(5);x(7);x(9)]
```

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

R. Hiptmair

rev 30401, February 24, 2009

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

1.1.2 Matrices

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

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

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

✎ notation: **bold** capital roman letters, e.g., $\mathbf{A}, \mathbf{S}, \mathbf{Y}$

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

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

R. Hiptmair

rev 30401, February 24, 2009

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

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

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

$\rightarrow$ entry $(\mathbf{A})_{i,j} = a_{ij}$, $1 \le i \le n, 1 \le j \le m$,
$\rightarrow$ i-th row, $1 \le i \le n$: $a_{i,:} = (\mathbf{A})_{i,:}$,
$\rightarrow$ j-th column, $1 \le j \le m$: $a_{i,:} = (\mathbf{A})_{:,j}$,
$\rightarrow$ matrix block (sub-matrix) $(a_{ij})_{\substack{i=k,\dots,l \\ j=r,\dots,s}} = (\mathbf{A})_{k:l,r:s}$, $1 \le k \le l \le n$, $1 \le r \le s \le m$.

MATLAB:

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

- Transposed matrix:

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

R. Hiptmair
rev 30401, February 24, 2009

- Adjoint matrix (Hermitian transposed):

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

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

- Special matrices:

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

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

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

R. Hiptmair

rev 30401, February 24, 2009

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

$\mathbf{A} \in \mathbb{K}^{m,n}$ ◣ linear array (size mn) + index computations

(Note: leading dimension (*row major, column major*))

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

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

A_arr	1	2	3	4	5	6	7	8	9

Column major (Fortran, MATLAB, OpenGL):

A_arr	1	4	7	2	5	8	3	6	9

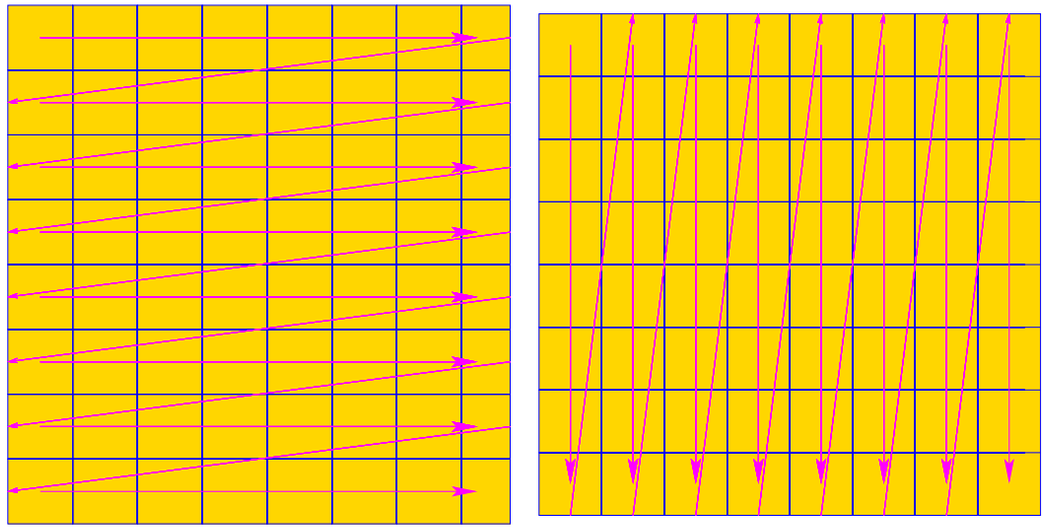
Access to entry a_{ij} of $\mathbf{A} \in \mathbb{K}^{n,m}$, $i = 1, \ldots, n$, $j = 1, \ldots, m$:

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).

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

R. Hiptmair

rev 30401, February 24, 2009

column oriented

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

access

$n = 3000 \rightsquigarrow 0.1s$

row oriented

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

access

$n = 3000 \rightsquigarrow 0.3s$

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

```
% Timing for row/column operations on matrices
K = 3; res = [];
for n=2.^(4:13)
  A = randn(n,n);

  t1 = 1000;
  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 = 1000;
  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
```

R. Hiptmair

rev 30401, February 24, 2009

```
18
19 % Plot runtimes versus matrix sizes
20 figure; plot(res(:,1),res(:,2),'r+', res(:,1),res(:,3),'m*');
21 xlabel('{\bf n}','fontsize',14);
22 ylabel('{\bf runtime [s]}','fontsize',14);
23 legend('A(:,j+1) = A(:,j+1) - A(:,j)','A(i+1,:) = A(i+1,:) -
     A(i,:)',...
24       'location','northwest');
25 print -depsc2 '../PICTURES/accessrtlin.eps';
26
27 figure; loglog(res(:,1),res(:,2),'r+', res(:,1),res(:,3),'m*');
28 xlabel('{\bf n}','fontsize',14);
29 ylabel('{\bf runtime [s]}','fontsize',14);
30 legend('A(:,j+1) = A(:,j+1) - A(:,j)','A(i+1,:) = A(i+1,:) -
     A(i,:)',...
31       'location','northwest');
32 print -depsc2 '../PICTURES/accessrtlog.eps';
```

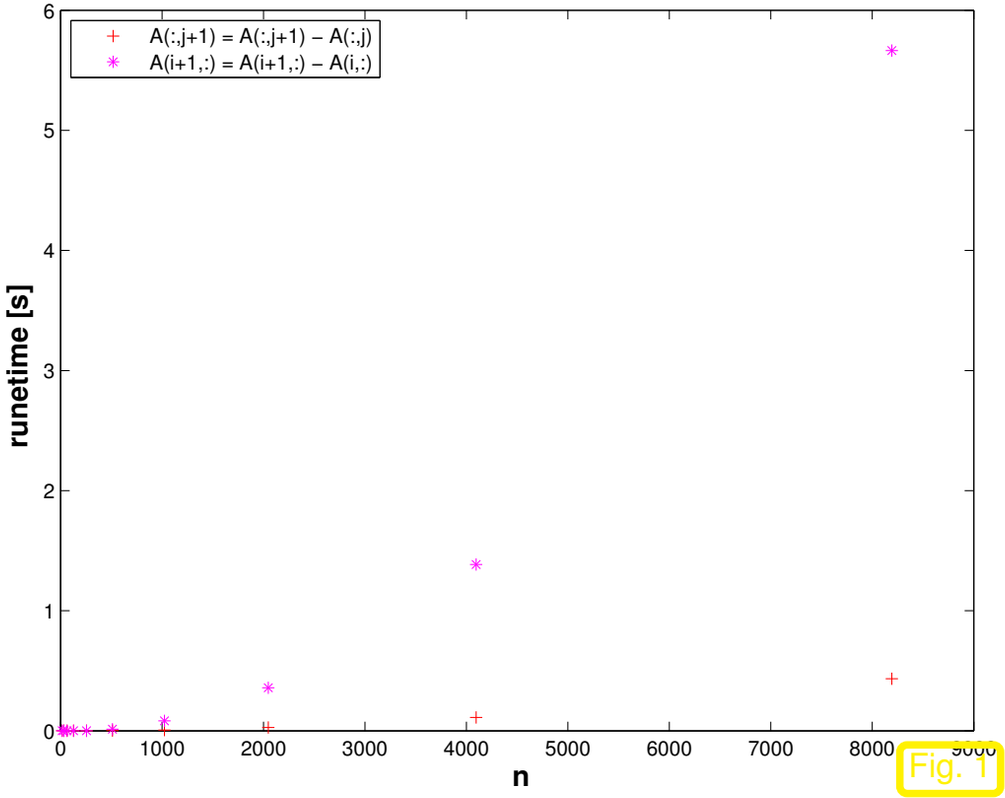


Fig. 1

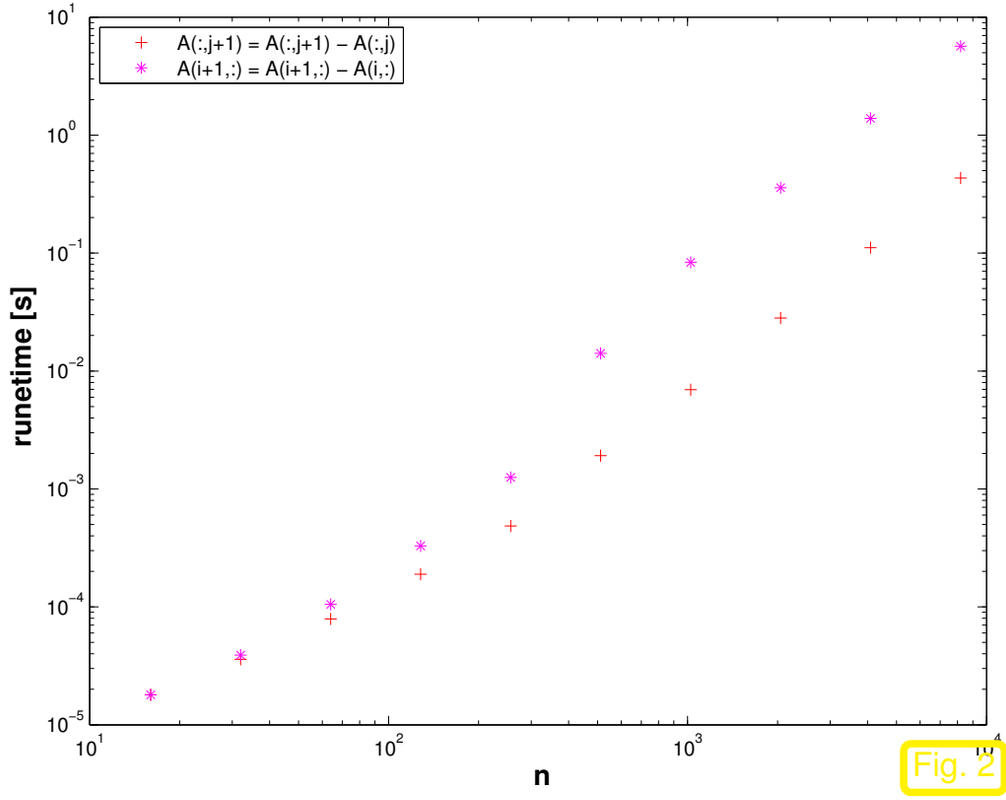


Fig. 2

R. Hiptmair

rev 30401, September 10, 2010

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

◇

1.2 Elementary operations

What you should know from linear algebra:

- vector space operations in $\mathbb{K}^{m,n}$ (addition, multiplication with scalars)
- dot product: $\mathbf{x},\mathbf{y} \in \mathbb{K}^n$, $n \in \mathbb{N}$: $\mathbf{x}\cdot\mathbf{y} := \mathbf{x}^H\mathbf{y} = \sum_{i=1}^{n} \bar{x}_i y_i \in \mathbb{K}$

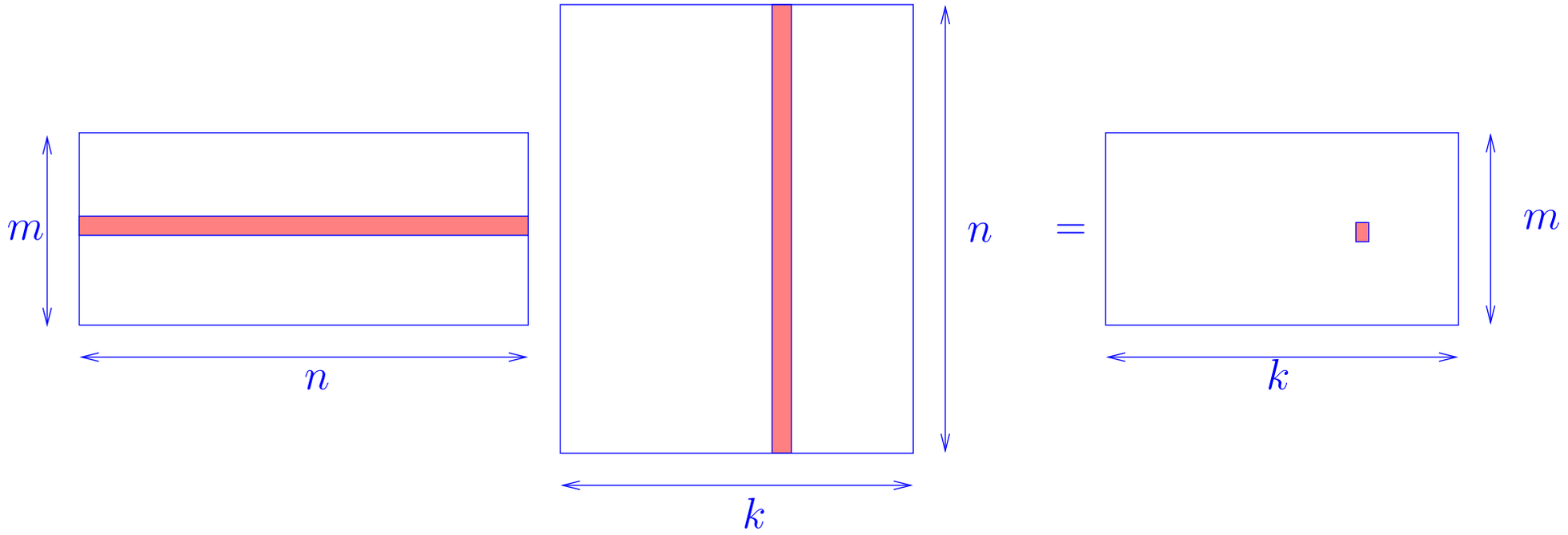
  (in MATLAB: `dot(x,y)`)
- tensor product: $\mathbf{x} \in \mathbb{K}^m, \mathbf{y} \in \mathbb{K}^n$, $n \in \mathbb{N}$: $\mathbf{x}\mathbf{y}^H = \left(x_i \bar{y}_j\right)_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}} \in \mathbb{K}^{m,n}$
- All are special cases of the matrix product:

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

“Visualization” of matrix product:

R. Hiptmair

rev 30401, September 10, 2010



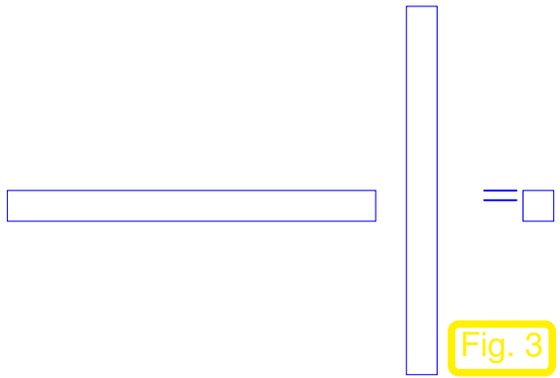


Fig. 3

dot product

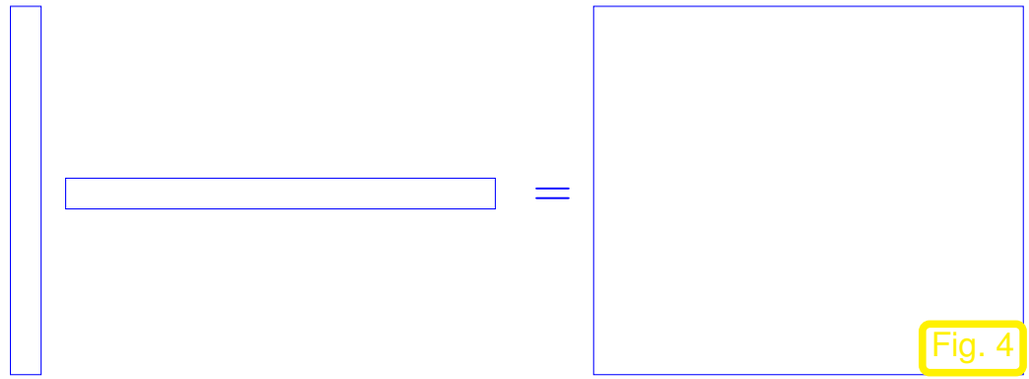


Fig. 4

tensor product

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

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

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

$\downarrow$ matrix assembled from columns

$\downarrow$ matrix assembled from rows

△

Remark 1.2.4 (Understanding the structure of product matrices).

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

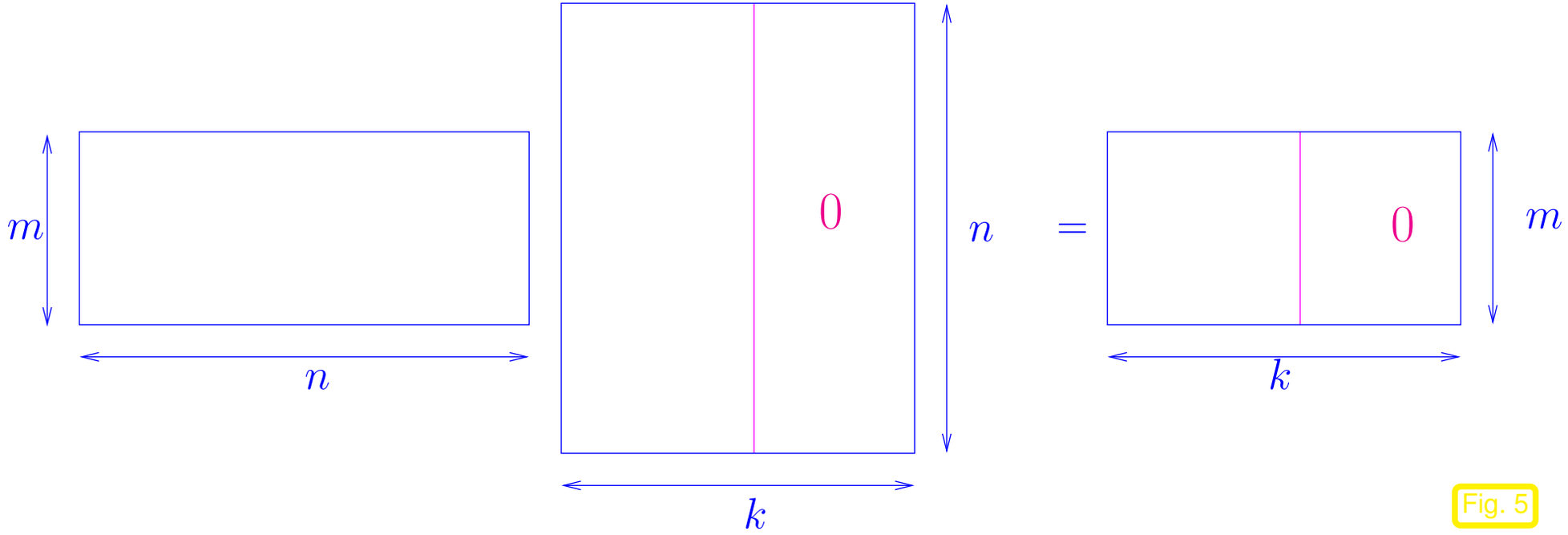


Fig. 5

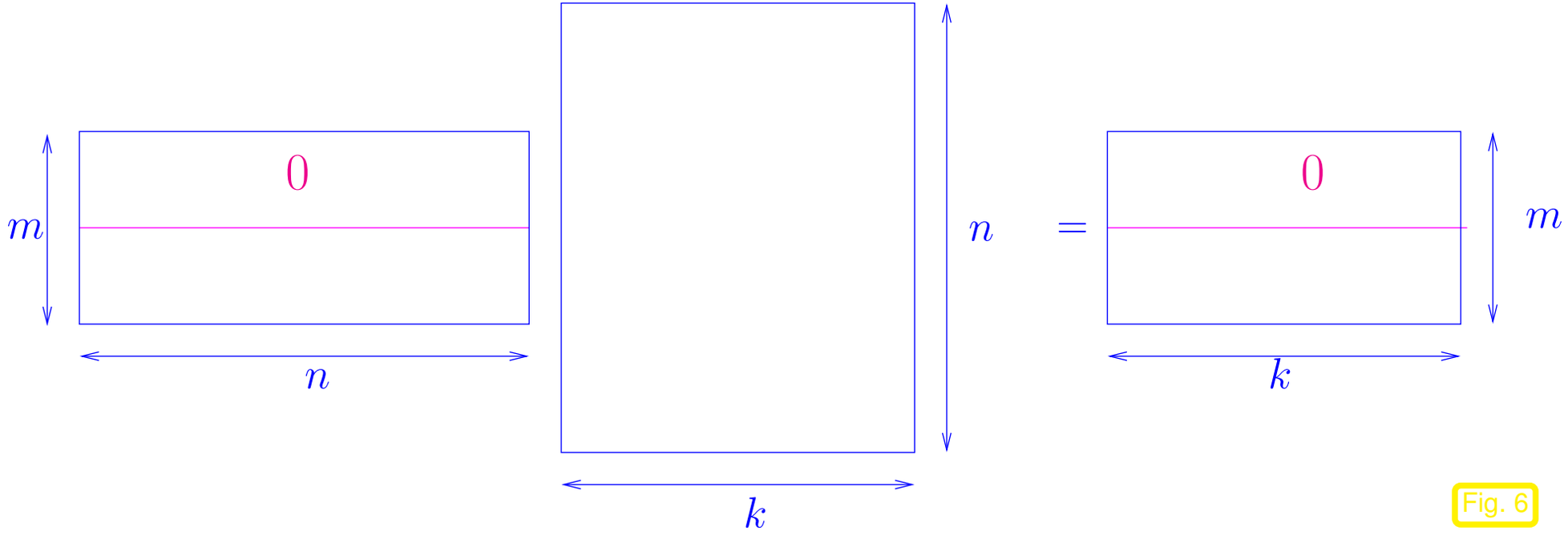


Fig. 6

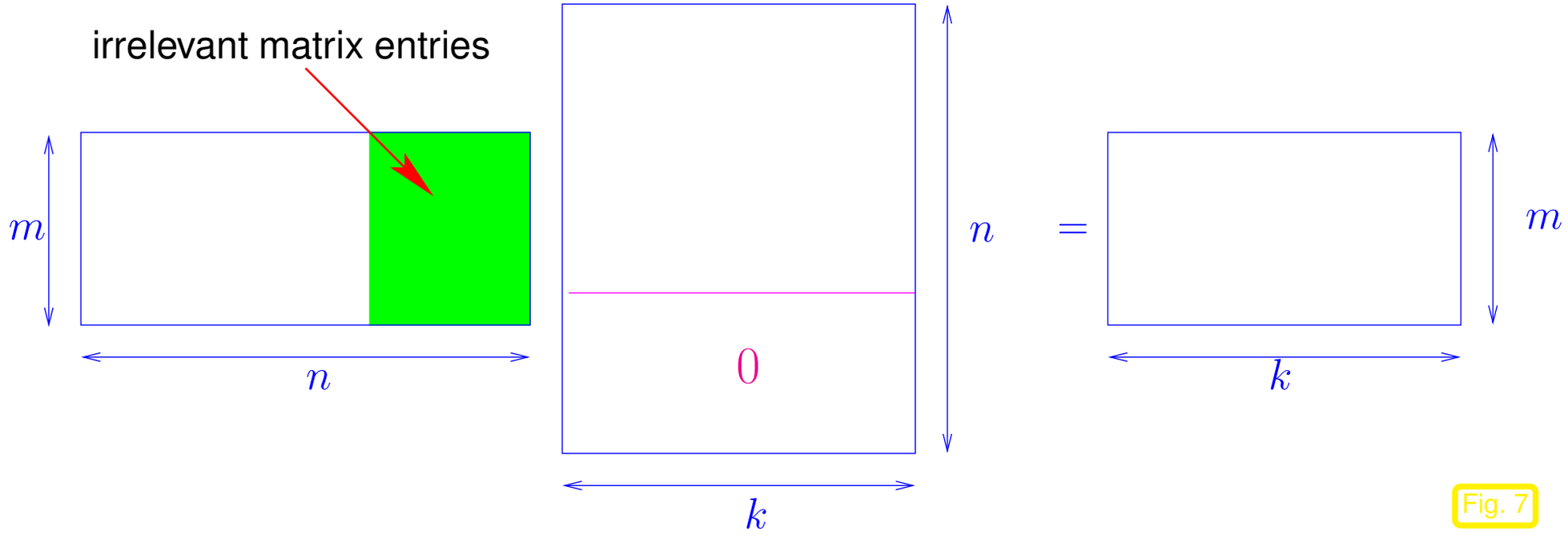
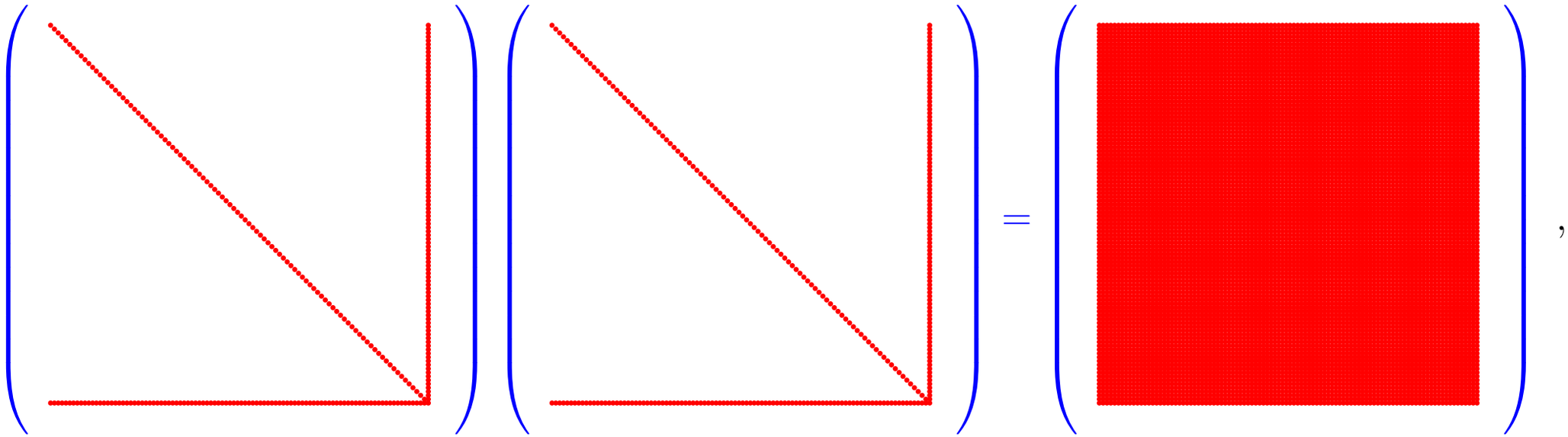
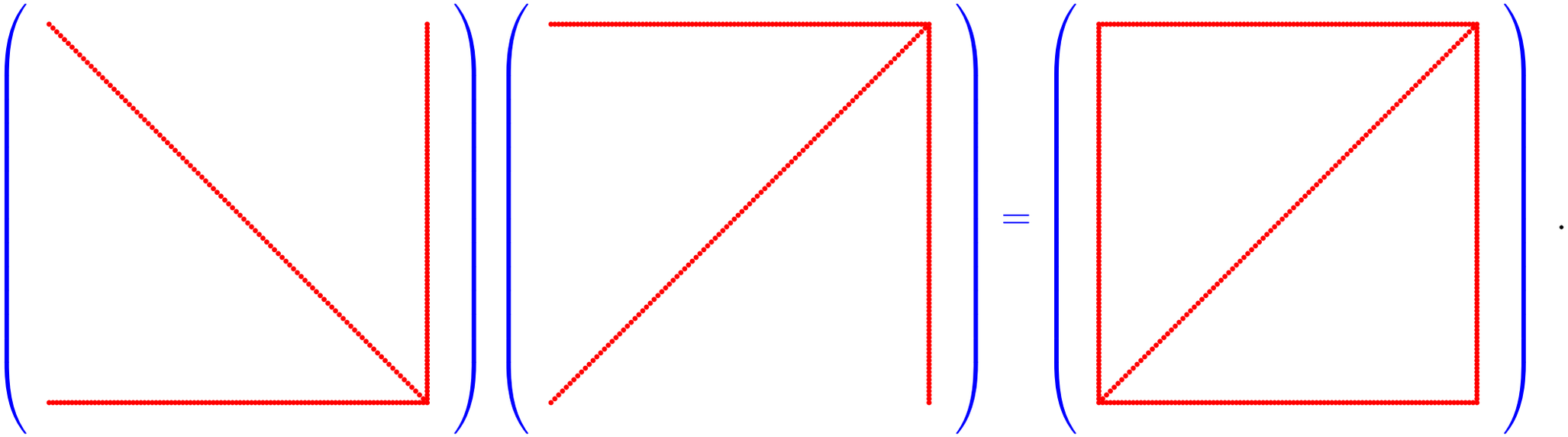


Fig. 7



R. Hiptmair

rev 30401, September 10, 2010



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

Code 1.2.5: visualizing structure of matrices

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

Remark 1.2.6 (Scalings).

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

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

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

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

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & & a_{2m} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \begin{pmatrix} d_1 & 0 & & 0 \\ 0 & d_2 & & 0 \\ & & \ddots & \\ 0 & 0 & & d_m \end{pmatrix} = \begin{pmatrix} d_1 a_{11} & d_2 a_{12} & \dots & d_m a_{1m} \\ d_1 a_{21} & d_2 a_{22} & \dots & d_m a_{2m} \\ \vdots & & & \vdots \\ d_1 a_{n1} & d_2 a_{n2} & \dots & d_m a_{nm} \end{pmatrix}$$

$$= \begin{pmatrix} d_1 (\mathbf{A})_{:,1} & \dots & d_m (\mathbf{A})_{:,m} \end{pmatrix} .$$

R. Hiptmair

rev 30401, September 10, 2010

Example 1.2.7 (Row and column transformations).

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

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

left-multiplication right-multiplication	with transformation matrices	➙ row transformations column transformations

◇

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

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

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

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

Remark 1.2.8 (Matrix algebra).

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

△

Remark 1.2.9 (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

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

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{pmatrix} . \tag{1.2.10}$$

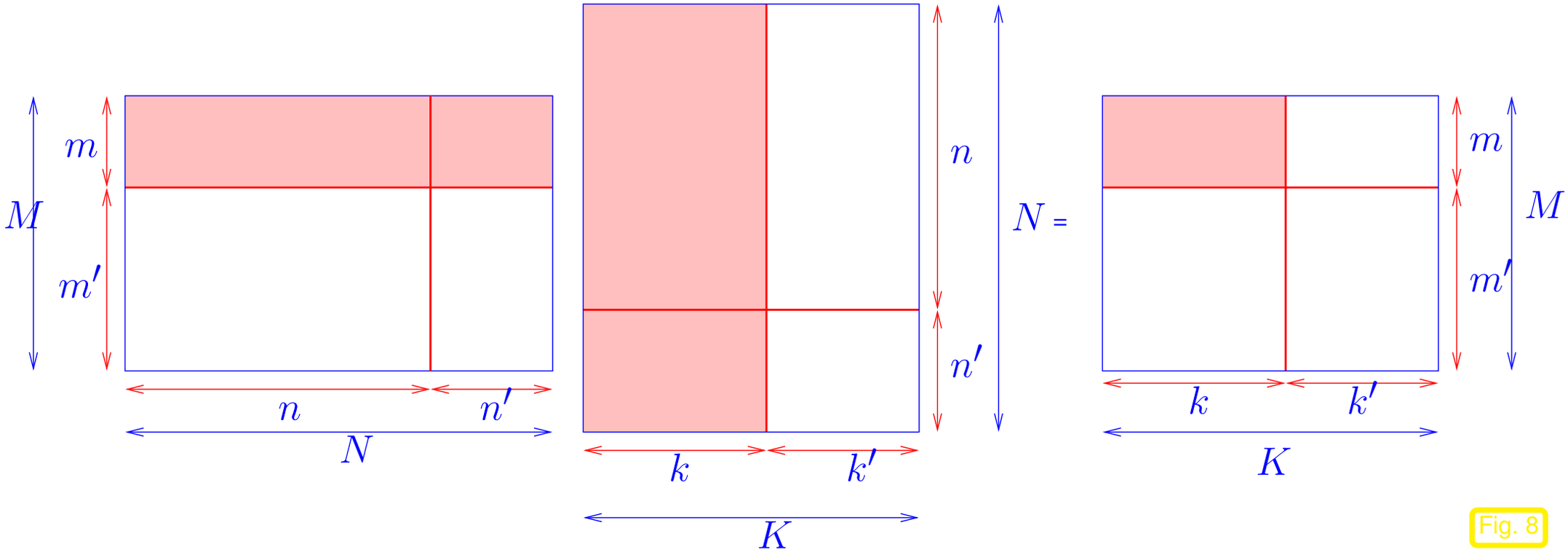


Fig. 8

R. Hiptmair

rev 30401, September 10, 2010

Bottom line: one can compute with block-structured matrices in *almost* ($*$) the same ways as with matrices with real/complex entries.

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

△

1.3 Complexity/computational effort

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

additions/multiplications

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

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

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

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

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

Remark 1.3.1 ("Fast" matrix multiplication).

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

For $m = n = k$ there are (sophisticated) variants with better asymptotic complexity, e.g., the divide-and-conquer Strassen algorithm [51] with asymptotic complexity $O(n^{\log_2 7})$:

Start from $\mathbf{A}, \mathbf{B} \in \mathbb{K}^{n,n}$ with $n = 2\ell$, $\ell \in \mathbb{N}$. The idea relies on the block matrix product (1.2.10) with $\mathbf{A}_{ij}, \mathbf{B}_{ij} \in \mathbb{K}^{\ell,\ell}$, $i, j \in \{1, 2\}$. Let $\mathbf{C} := \mathbf{AB}$ be partitioned accordingly: $\mathbf{C} = \begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{22} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{pmatrix}$. Then tedious elementary computations reveal

$$\begin{aligned} \mathbf{C}_{11} &= \mathbf{Q}_0 + \mathbf{Q}_3 - \mathbf{Q}_4 + \mathbf{Q}_6 \,, \\ \mathbf{C}_{21} &= \mathbf{Q}_1 + \mathbf{Q}_3 \,, \\ \mathbf{C}_{12} &= \mathbf{Q}_2 + \mathbf{Q}_4 \,, \\ \mathbf{C}_{22} &= \mathbf{Q}_0 + \mathbf{Q}_2 - \mathbf{Q}_1 + \mathbf{Q}_5 \,, \end{aligned}$$

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

$$\begin{aligned} \mathbf{Q}_0 &= (\mathbf{A}_{11} + \mathbf{A}_{22}) * (\mathbf{B}_{11} + \mathbf{B}_{22}) \,, \\ \mathbf{Q}_1 &= (\mathbf{A}_{21} + \mathbf{A}_{22}) * \mathbf{B}_{11} \,, \\ \mathbf{Q}_2 &= \mathbf{A}_{11} * (\mathbf{B}_{12} - \mathbf{B}_{22}) \,, \\ \mathbf{Q}_3 &= \mathbf{A}_{22} * (-\mathbf{B}_{11} + \mathbf{B}_{21}) \,, \\ \mathbf{Q}_4 &= (\mathbf{A}_{11} + \mathbf{A}_{12}) * \mathbf{B}_{22} \,, \\ \mathbf{Q}_5 &= (-\mathbf{A}_{11} + \mathbf{A}_{21}) * (\mathbf{B}_{11} + \mathbf{B}_{12}) \,, \\ \mathbf{Q}_6 &= (\mathbf{A}_{12} - \mathbf{A}_{22}) * (\mathbf{B}_{21} + \mathbf{B}_{22}) \,. \end{aligned}$$

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

A refined algorithm of this type can achieve complexity $O(n^{2.36})$, see [9].

Example 1.3.2 (Efficient associative matrix multiplication).

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

$$\mathbf{y} = \left(\mathbf{a}\mathbf{b}^T\right)\mathbf{x} \,. \qquad (1.3.3)$$

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

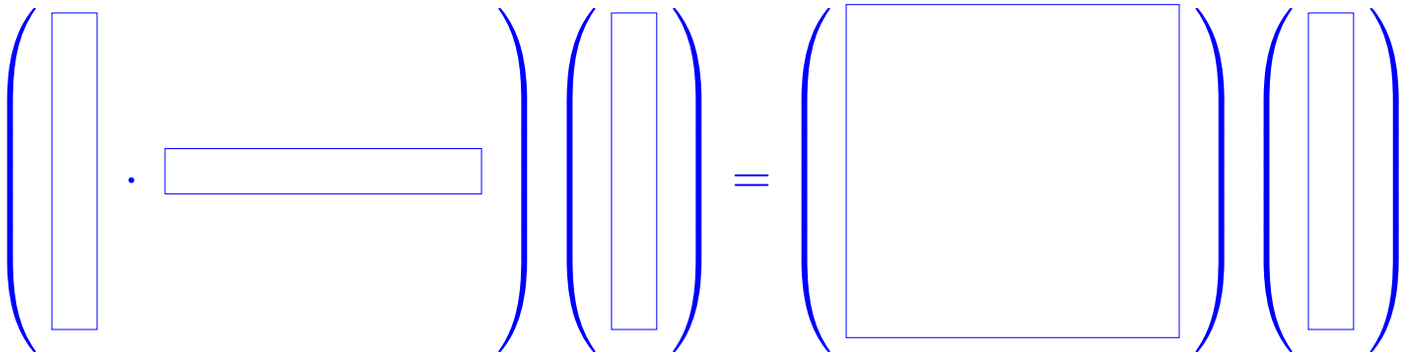
➤ complexity $O(mn)$

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

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

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

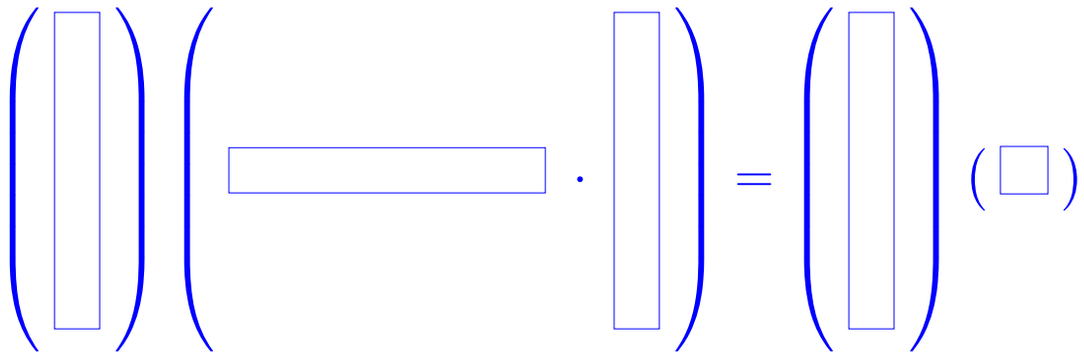
Visualization of evaluation according to (1.3.3):



R. Hiptmair

rev 30401, September 10, 2010

Visualization of evaluation according to (1.3.3):



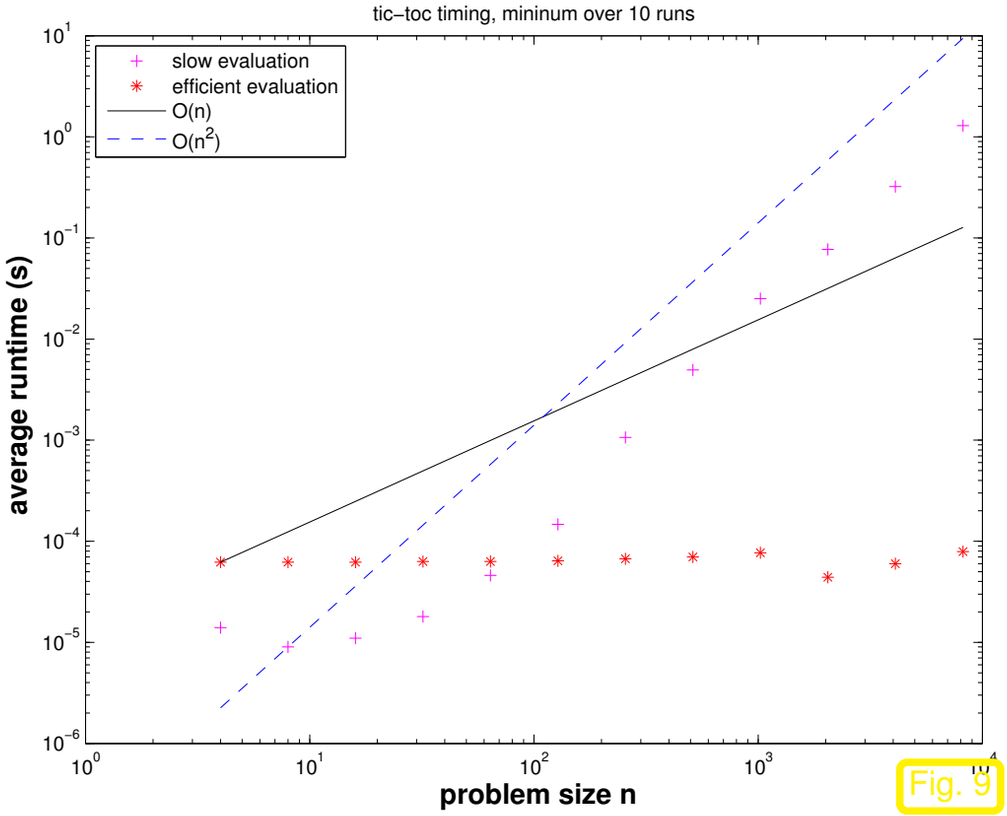


Fig. 9

◁ 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

R. Hiptmair

rev 30401, September 10, 2010

Code 1.3.5: MATLAB code for Ex. 1.3.2

```
function dottenstiming(N,nruns)
% This function compares the runtimes for the
% multiplication of a vector with a rank-1 matrix ab^T, a,b ∈ R^n
% using different associative evaluations

if (nargin < 1), N = 2.^(2:13); end
if (nargin < 2), nruns = 10; end

times = []; % matrix for storing recorded runtimes
for n=N
  % Initialize dense vectors a, b, x (column vectors!)
  a = (1:n)'; b = (n:-1:1)'; x = rand(n,1);

  % Measuring times using MATLAB tic-toc commands
  tfool = 1000; for i=1:nruns, tic; y = (a*b')*x; tfool =
    min(tfool,toc); end;
  tsmart = 1000; for i=1:nruns, tic; y = a*dot(b',x); tsmart =
    min(tsmart,toc); end;
  times = [times;n, tfool, tsmart];
end

% log-scale plot for investigation of asymptotic complexity
```

R. Hiptmair

rev 30401, September 10, 2010

```
figure('name','dottenstiming');
loglog(times(:,1),times(:,2),'m+',...
       times(:,1),times(:,3),'r*',...
       times(:,1),times(:,1)*times(1,3)/times(1,1),'k-',...
       times(:,1),(times(:,1).^2)*times(2,2)/(times(2,1)^2),'b--');
xlabel('{\bf problem size n}','fontsize',14);
ylabel('{\bf average runtime (s)}','fontsize',14);
title('tic-toc timing, mininum over 10 runs');
legend('slow evaluation','efficient evaluation',...
       'O(n)','O(n^2)','location','northwest');
print -depsc2 '../PICTURES/dottenstiming.eps';
```


◇

Remark 1.3.6 (Reading off complexity).

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

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

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

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

➢ quick "visual test" of conjectured asymptotic complexity

More rigorous: Perform linear regression on $(\log n_i, \log t_i)$, $i = 1, \ldots, N$ (→ Ch. 7)

△

Remark 1.3.7 (Relevance of asymptotic complexity).

Runtimes in Ex. 1.3.2 illustrate that the

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

because on modern computer architectures with multi-level memory hierarchies the *memory access pattern* may be more important for efficiency than the mere number of floating point operations, see [32].

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

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

△

1.4 BLAS

BLAS = basic linear algebra subroutines

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

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

Example 1.4.1 (Different implementations of matrix multiplication in MATLAB).

Code 1.4.2: Timing different implementations of matrix multiplication

```
% MATLAB script for timing different implementations of matrix multiplications
nruns = 3; times = [];
for n=2.^(2:10) % n = size of matrices
  fprintf('matrix size n = %d\n',n);
  A = rand(n,n); B = rand(n,n); C = zeros(n,n);
  t1 = realmax;
  % loop based implementation (no BLAS)
  for l=1:nruns
    tic;
    for i=1:n, for j=1:n
        for k=1:n, C(i,j) = C(i,j) + A(i,k)*B(k,j); end
      end, end
    t1 = min(t1,toc);
    end
  t2 = realmax;
```

```
16    % dot product based implementation (BLAS level 1)
17    for l=1:nruns
18      tic;
19      for i=1:n
20        for j=1:n, C(i,j) = dot(A(i,:),B(:,j));  end
21      end
22      t2 = min(t2,toc);
23    end
24    t3 = realmax;
25    % matrix-vector based implementation (BLAS level 2)
26    for l=1:nruns
27      tic;
28      for j=1:n, C(:,j) = A*B(:,j); end
29      t3 = min(t3,toc);
30    end
31    t4 = realmax;
32    % BLAS level 3 matrix multiplication
33    for l=1:nruns
34      tic; C = A*B; t4 = min(t4,toc);
35    end
36    times = [ times; n t1 t2 t3 t4];
37  end
38
39  figure('name','mmtiming');
```

```
40 loglog(times(:,1),times(:,2),'r+-',...
41       times(:,1),times(:,3),'m*-',...
42       times(:,1),times(:,4),'b^-',...
43       times(:,1),times(:,5),'kp-');
44 title('Timings: Different implementations of matrix
    multiplication');
45 xlabel('matrix size n','fontsize',14);
46 ylabel('time [s]','fontsize',14);
47 legend('loop implementation','dot product implementation',...
48        'matrix-vector implementation','BLAS gemm (MATLAB *)',...
49        'location','northwest');
50
51 print -depsc2 '../PICTURES/mvtiming.eps';
```


R. Hiptmair

rev 30401, September 10, 2010

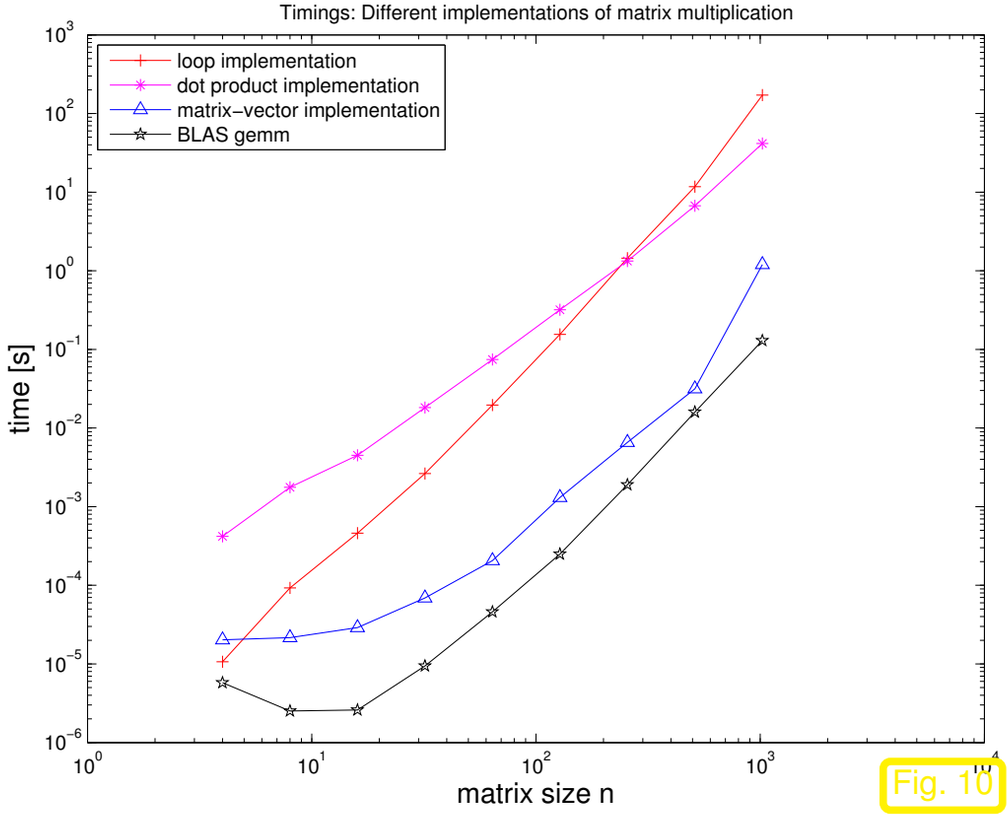


Fig. 10

Platform:

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

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

◇

R. Hiptmair

rev 30401, September 10, 2010

Grouping of BLAS routines (“levels”) according to asymptotic complexity, see [20, Sect. 1.1.12]:

- **Level 1**: vector operations such as scalar products and vector norms.
  asymptotic complexity $O(n)$, (with $n \hat{=}$ vector length),
  e.g.: dot product: $\rho = \mathbf{x}^T\mathbf{y}$

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

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

R. Hiptmair

rev 30401, September 10, 2010

- **BLAS LEVEL 1**: vector operations, asymptotic complexity $O(n)$, $n \hat{=}$ vector length
  - dot product $\rho = \mathbf{x}^T\mathbf{y}$

```
xDOT(N,X,INCX,Y,INCY)
```

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

– INCX $\hat{=}$ stride for traversing vector X

– $Y \hat{=}$ vector $\mathbf{y}$: array of type x

– INCY $\hat{=}$ stride for traversing vector Y

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

```
xAXPY(N,ALPHA,X,INCX,Y,INCY)
```

– x $\in \{\mathrm{S}, \mathrm{D}, \mathrm{C}, \mathrm{Z}\}$, S $\hat{=}$ type `float`, D $\hat{=}$ type `double`, C $\hat{=}$ type `complex`

– $N \hat{=}$ length of vector (modulo stride INCX)

– ALPHA $\hat{=}$ scalar α

– $X \hat{=}$ vector $\mathbf{x}$: array of type x

– INCX $\hat{=}$ stride for traversing vector X

– $Y \hat{=}$ vector $\mathbf{y}$: array of type x

– INCY $\hat{=}$ stride for traversing vector Y

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

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

```
xGEMV(TRANS,M,N,ALPHA,A,LDA,X,
INCX,BETA,Y,INCY)
```

R. Hiptmair

rev 30401, September 10, 2010

– $\mathtt{x} \in \{\mathrm{S}, \mathrm{D}, \mathrm{C}, \mathrm{Z}\}$, scalar type: S $\hat{=}$ type `float`, D $\hat{=}$ type `double`, C $\hat{=}$ type `complex`

– $M, N \mathrel{\hat{=}}$ size of matrix $\mathbf{A}$

– `ALPHA` $\hat{=}$ scalar parameter α

– `A` $\hat{=}$ matrix $\mathbf{A}$ stored in *linear array* of length $M \cdot N$ (column major arrangement)

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

– `LDA` $\hat{=}$ "leading dimension" of $\mathbf{A} \in \mathbb{K}^{n,m}$, that is, the number n of rows.

– $X \mathrel{\hat{=}}$ vector $\mathbf{x}$: array of type x

– `INCX` $\hat{=}$ stride for traversing vector X

– `BETA` $\hat{=}$ scalar paramter β

– $Y \mathrel{\hat{=}}$ vector $\mathbf{y}$: array of type x

– `INCY` $\hat{=}$ stride for traversing vector Y

- **BLAS LEVEL 3**: matrix-matrix operations, asymptotic complexity $O(mnk)$, $(m, n, k) \mathrel{\hat{=}}$ matrix sizes

  – matrix×matrix multiplication $\mathbf{C} = \alpha\mathbf{A}\mathbf{B} + \beta\mathbf{C}$

`x`GEMM(TRANSA, TRANSB, M, N, K,

```
ALPHA,A,LDA,X,B,LDB,
BETA,C,LDC)
```

(☞ meaning of arguments as above)

Remark 1.4.3 (BLAS calling conventions).

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

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

△

Example 1.4.4 (Gaining efficiency through use of BLAS). → Ex. 1.4.1

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

Code 1.4.5: ColumnMajor Matrix Class in C++

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

R. Hiptmair

rev 30401, September 10, 2010

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

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

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

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

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


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

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

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

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

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

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

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

R. Hiptmair

rev 30401, September 10, 2010

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

```
}
```

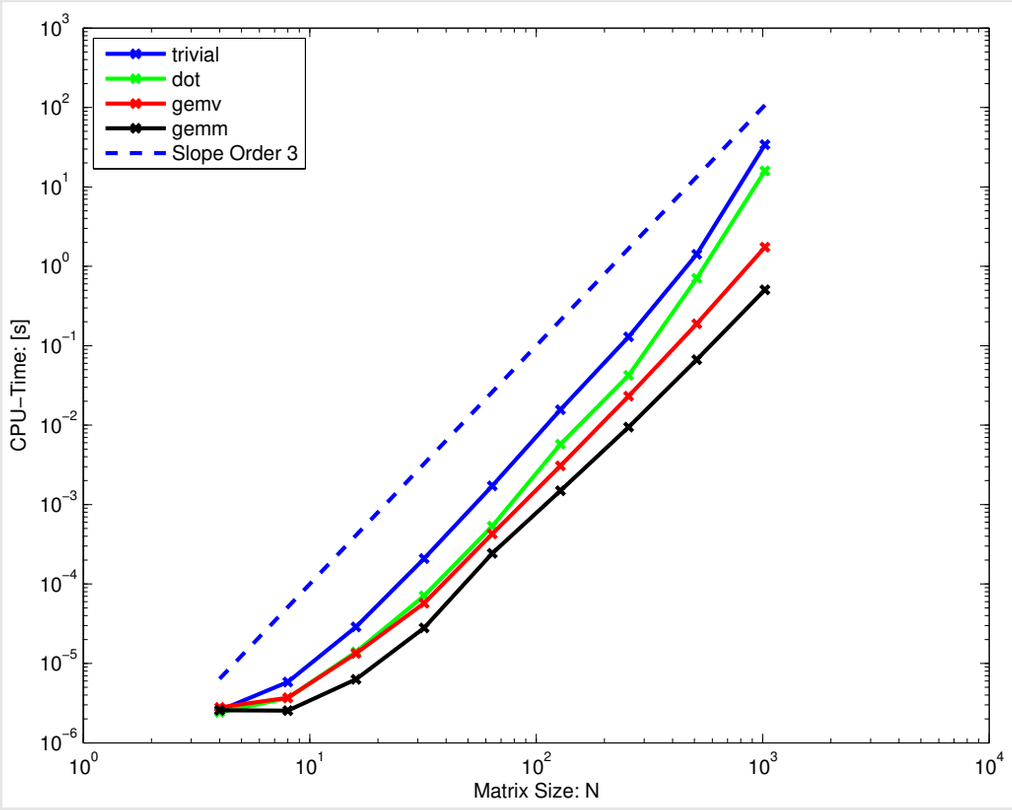
Code 1.4.13: Timings of different Matrix Multiplications in C++

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cassert>
#include "simpleTimer.h"
#include "unixTimer.h"
#include "ColumnMajorMatrix.h"
/* Main Routine for the timing of different
 * Matrix Matrix Multiplication implementations */
int main (int argc, char * const argv[]) {

    double T0(1e20),T1(1e20),T2(1e20),T3(1e20);
    simpleTimer watch;
    int rep(1),n(5);
    if (argc>1) n=atoi(argv[1]);
    if (argc>2) rep=atoi(argv[2]);
    //Declare Input Data
    ColumnMajorMatrix A(n,n);
    A.initRand();//A.initGrow();
    ColumnMajorMatrix B(A);
    //The Results:
```

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

R. Hiptmair

rev 30401, September 10, 2010



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

OS: Mac OS X

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

Compiler: intel v.11.0 (-O3 option)

◇

Available BLAS implementations:

Below a list of the most common BLAS implementations:

- Reference implementations in C and Fortran (open-source): http://www.netlib.org/blas/

- ATLAS: Automatically tuned linear algebra software (open-source):
  http://math-atlas.sourceforge.net/
- uBLAS: generic C++ template library (part of Boost)
  www.boost.org
- Intel MKL: vendor-specific implementation

Installation:

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

Time Measurement:

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

- the *wall time* or *real time* denotes the time an observer would measure between the program start and end. (c.f. wall clock)
- the *user time* denotes the cpu-time spent in executing the user's code.
- the *system time* denotes the cpu-time that the system spent on behalf of the user's code (e.g. memory allocation, i/o handling etc.)

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

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

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


R. Hiptmair

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


R. Hiptmair

rev 30401, September 10, 2010

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

```
/*
 * unixTimer Class using times()-command from the unixbased
   times.h
 * this class will report the user, system and real time.
 */
#include <sys/param.h>
#include <sys/times.h>
#include <sys/types.h>
class unixtimer {
public:
     unixtimer():utime(0),stime(0),rtime(0),bStarted(false)
     {}

  void start() {rt0=times(&t0);    bStarted=true; }

     double stop() {
          tms t1;
          long rt1;
          assert(bStarted);
          rt1=times(&t1);
          utime=((double)(t1.tms_utime-t0.tms_utime))/
            CLOCKS_PER_SEC*10000;
          stime=((double)(t1.tms_stime-t0.tms_stime))/
```

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


R. Hiptmair

rev 30401, September 10, 2010

2 Direct Methods for Linear Systems of Equations

The fundamental task:

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

Sought : solution vector $\mathbf{x} \in \mathbb{K}^n$: $\boxed{\mathbf{Ax} = \mathbf{b}}$ $\leftarrow$ (square) linear system of equations (LSE)

(*ger.:* lineares Gleichungssystem)

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

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

- with discrete linear models in network theory (see Ex. 2.0.1), control, statistics;

R. Hiptmair

rev 30401, January 28, 2011

- in the case of *discretized* boundary value problems for ordinary and partial differential equations ($\rightarrow$ course “Numerical methods for partial differential equations”, 4th semester);
- as a result of linearization (e.g, “Newton’s method” $\rightarrow$ Sect. 4.4).

Example 2.0.1 (Nodal analysis of (linear) electric circuit).

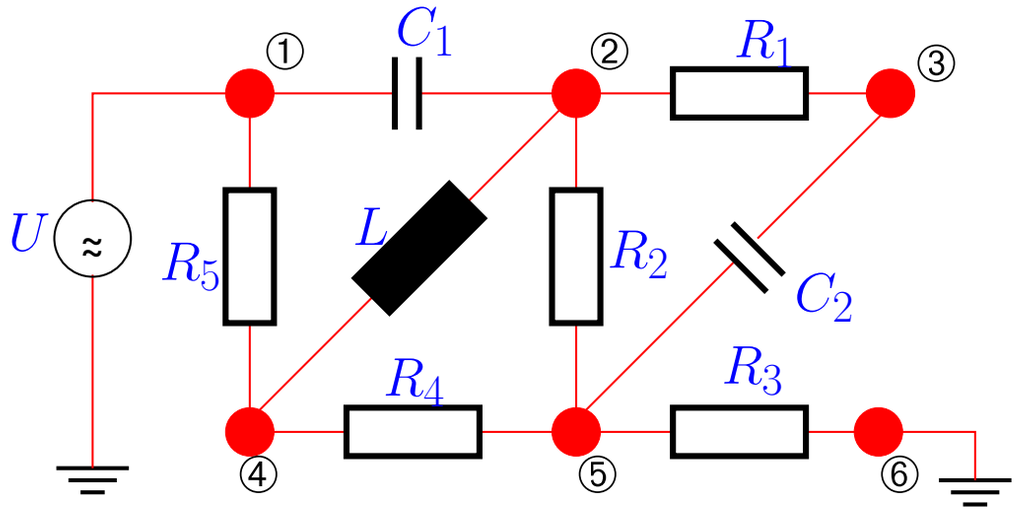
Node (*ger.:* Knoten) $\hat{=}$ junction of wires

☞ *number* nodes $1, \ldots, n$

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

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

$$\forall k \in \{1, \ldots, n\}: \quad \sum_{j=1}^{n} I_{kj} = 0 \,. \qquad \text{(2.0.2)}$$



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

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

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

R. Hiptmair

rev 30401, January 28, 2011

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

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

These constitutive relations are derived by assuming a harmonic time-dependence of all quantities:

$$\text{voltage:} \quad u(t) = \mathrm{Re}\{U\exp(i\omega t)\} \quad , \quad \text{current:} \quad i(t) = \mathrm{Re}\{I\exp(\omega t)\} \ . \tag{2.0.3}$$

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

$$\dot{u}(t) = \mathrm{Re}\{i\omega U\exp(i\omega t)\} \quad , \quad \dot{i}(t) = \mathrm{Re}\{i\omega I\exp(i\omega t)\} \ . \tag{2.0.4}$$

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

$$q(t) = Cu(t) \quad \overset{i(t)=\dot{q}(t)}{\Rightarrow} \quad i(t) = C\dot{u}(t) \ .$$

For a coil the voltage is proportional to the rate of change of current: $u(t) = L\dot{i}(t)$. Combined with (2.0.3) and (2.0.4) this leads to the above constitutive relations.

R. Hiptmair

rev 30401, January 28, 2011

Constitutive relations + (2.0.2) ▶ linear system of equations:

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

▼

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

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

◇

Theory

Known from linear algebra:

Definition 2.0.5 (Invertible matrix). $\rightarrow$ *[39, Sect. 2.3]*

$$\mathbf{A} \in \mathbb{K}^{n,n} \quad \begin{matrix} \textit{invertible} / \\ \textit{regular} \end{matrix} \quad :\Leftrightarrow \quad \exists_1 \mathbf{B} \in \mathbb{K}^{n,n}: \quad \mathbf{AB} = \mathbf{BA} = \mathbf{I} \; .$$

$\mathbf{B} \mathrel{\hat{=}}$ *inverse of* $\mathbf{A}$, (✎ *notation* $\mathbf{B} = \mathbf{A}^{-1}$)

Definition 2.0.6 (Rank of a matrix). $\rightarrow$ *[39, Sect. 2.4]*

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

R. Hiptmair

rev 30401, January 28, 2011

Theorem 2.0.7 (Criteria for invertibility of matrix). → *[39, Sect. 2.3 & Cor. 3.8]*

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

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

Formal way to denote solution of LSE:

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

matrix inverse

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

R. Hiptmair

rev 30401, January 28, 2011

Always avoid computing the inverse of a matrix (which can almost always be avoided)! In particular, never ever even contemplate using `x = inv(A)*b` to solve the linear system of equations $\mathbf{Ax} = \mathbf{b}$. The next sections present a sound way to do this.

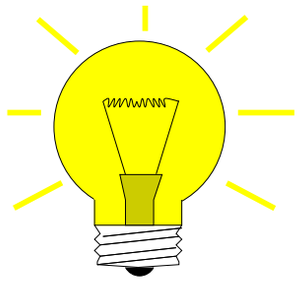
2.1 Gaussian Elimination

Exceptional feature of linear systems of equations (LSE):

☞ "exact" solution computable with finitely many elementary operations

Algorithm: Gaussian elimination ($\rightarrow$ secondary school, linear algebra [23, Ch. 1], [39, Ch. 1])

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



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

Ex. 1.2.7: row transformations $\leftrightarrow$ left-multiplication with transformation matrix

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

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

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

R. Hiptmair

rev 30401, September 27, 2010

Example 2.1.1 (Gaussian elimination).

① (Forward) elimination:

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

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

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

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

▻ transformation of LSE to upper triangular form

② Solve by back substitution: back substitution = Rücksubstitution

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

More detailed examples: [23, Sect. 1.1], [39, Sect. 1.1].

More general:

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

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

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

with

$$\begin{aligned} a_{ij}^{(1)} &= a_{ij} - a_{1j}\,l_{i1}, \qquad i,j = 2,\ldots,n\,, \\ b_i^{(1)} &= b_i - b_1\,l_{i1}, \qquad i = 2,\ldots,n\,. \end{aligned}$$

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

R. Hiptmair

rev 30401, September 27, 2010

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1\\
a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)}\\
a_{33}^{(2)}x_3 + \cdots + a_{3n}^{(2)}x_n &= b_3^{(2)}\\
\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots &\\
a_{n3}^{(2)}x_3 + \cdots + a_{nn}^{(2)}x_n &= b_n^{(2)}
\end{aligned}$$

After $n-1$ steps: linear systems of equations in upper triangular form

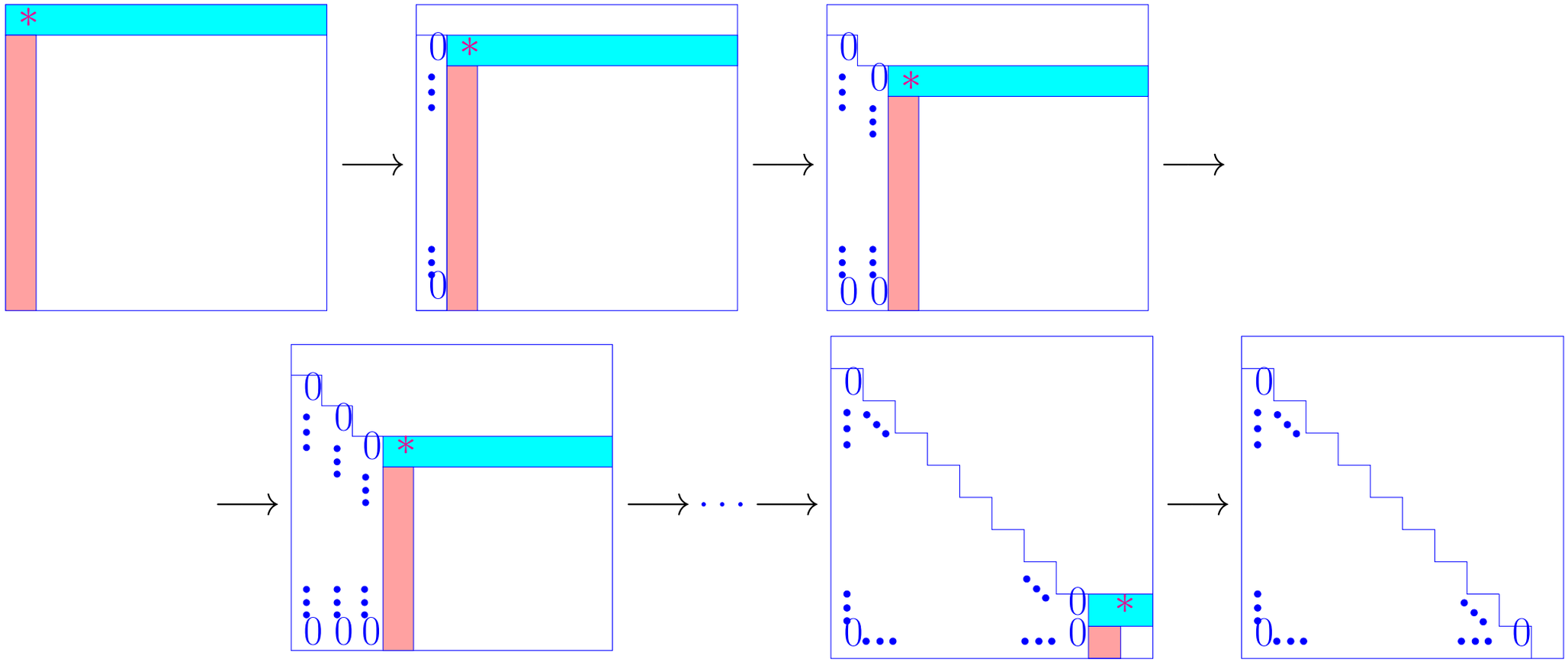
$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1\\
a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)}\\
a_{33}^{(2)}x_3 + \cdots + a_{3n}^{(2)}x_n &= b_3^{(2)}\\
\ddots \quad \ddots \quad \ddots \quad \vdots \quad \vdots \quad \vdots &\\
\ddots \quad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

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

Graphical depiction:

R. Hiptmair

rev 30401, September 27, 2010



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

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

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

with

R. Hiptmair

rev 30401, September 27, 2010

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

multipliers l_{ik}

R. Hiptmair

rev 30401, September 27, 2010

Algorithm 2.1.3.

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

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

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

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

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


R. Hiptmair

rev 30401, September 27, 2010

computational costs ($\leftrightarrow$ number of elementary operations) of Gaussian elimination:

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

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

Example 2.1.6 (Runtime of Gaussian elimination).

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

```
% MATLAB script for timing numerical solution of linear systems
nruns = 3; times = [];
for n = 2.^(3:12)
  fprintf('Matrix size n = %d\n',n);
  % Initialized random matrix and right hand side
  A = rand(n,n) + n*eye(n); b = rand(n,1);
  t1 = realmax; t2 = realmax;
  for j=1:nruns
    tic; x1 = gausselimsolve(A,b); t1 = min(t1,toc);
    tic; x2 = A\b; t2 = min(t2,toc);
      norm(x1-x2),
  end
```

```
13    times = [times; n t1 t2];
14  end
15
16  figure('name','gausstiming');
17  loglog(times(:,1),times(:,2),'r-+',times(:,1),times(:,3),'m-*',...
18         times(:,1),times(:,1).^3*(1E-5/(times(1,1)^3)),'k-');
19  xlabel('matrix size n','fontsize',14);
20  ylabel('execution time [s]','fontsize',14);
21  legend('gausselimsolve','backslash
    solve','O(n^3)','location','northwest');
22
23  print -depsc2 '../PICTURES/gausstiming.eps';
```

MATLAB \

▷ based on LAPACK

▷ based on BLAS.

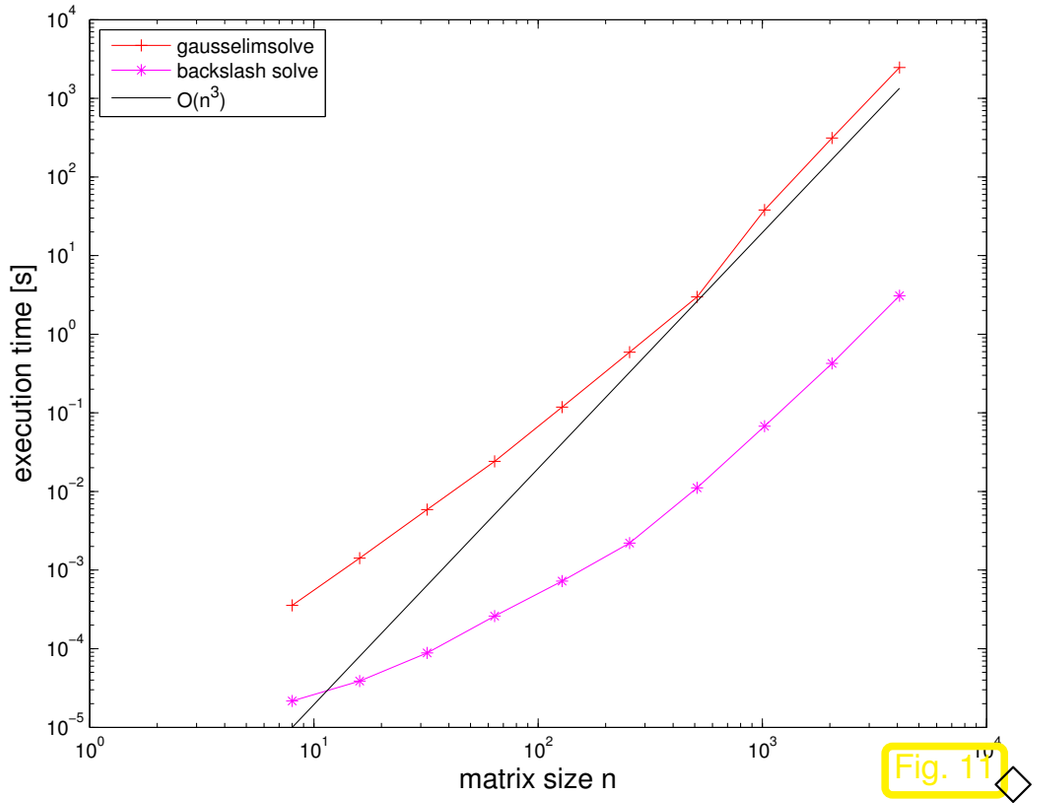


Fig. 11

\ about two orders of magnitude faster than a diret implementation

Never implement Gaussian elimination yourself **!**

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

Remark 2.1.8 (Gaussian elimination via rank-1 modifications).

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

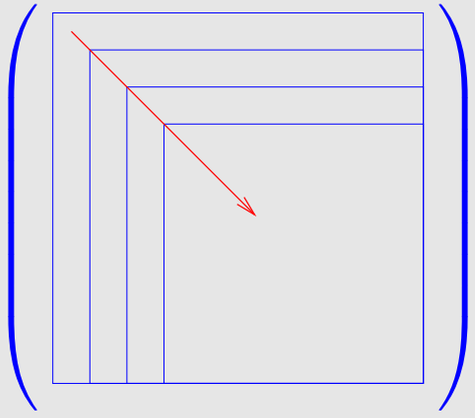
R. Hiptmair

$$\mathbf{A} := \left(\begin{array}{c|c} \alpha & \mathbf{c}^T \\ \hline \mathbf{d} & \mathbf{C} \end{array}\right) \rightarrow \mathbf{A}' := \left(\begin{array}{c|c} \alpha & \mathbf{c}^T \\ \hline \mathbf{0} & \mathbf{C}' := \mathbf{C} - \frac{\mathbf{d}\mathbf{c}^T}{\alpha} \end{array}\right) . \qquad (2.1.9)$$

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

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

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



.

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


R. Hiptmair

rev 30401, September 27, 2010

In this code the Gaussian elimination is carried out in situ: the matrix `A` is replaced with the transformed matrices during elimination. If the matrix is not needed later this offers maximum efficiency.

△

Remark 2.1.10 (Block Gaussian elimination).

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

$$\left(\begin{array}{cc|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{b}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{b}_2 \end{array}\right) \xrightarrow{\text{❶}} \left(\begin{array}{cc|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{b}_1 \\ 0 & \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} & \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1 \end{array}\right)$$

$$\xrightarrow{\text{❷}} \left(\begin{array}{cc|c} \mathbf{I} & 0 & \mathbf{A}_{11}^{-1}(\mathbf{b}_1 - \mathbf{A}_{12}\mathbf{S}^{-1}\mathbf{b}_S) \\ 0 & \mathbf{I} & \mathbf{S}^{-1}\mathbf{b}_S \end{array}\right) ,$$

where $\mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$ (Schur complement, see Rem. 2.2.14), $\mathbf{b}_S := \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1$

❶: elimination step, ❷: backsubstitution step

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

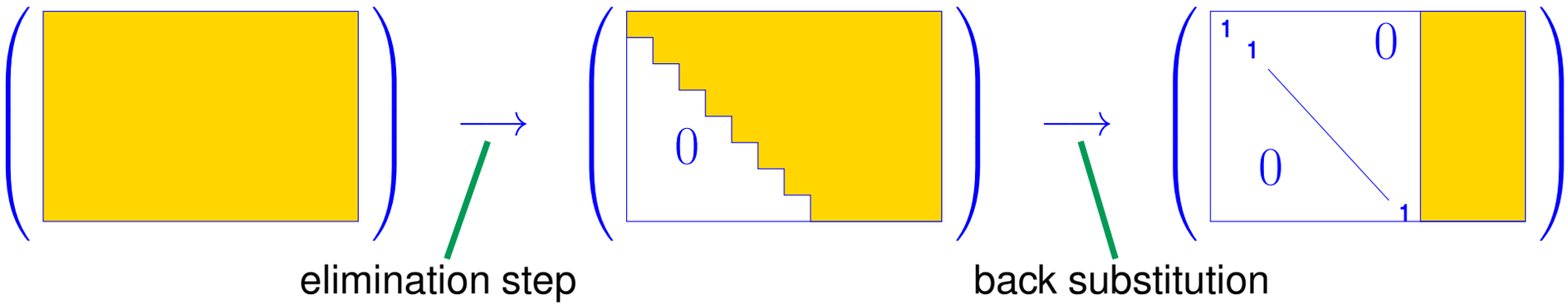
△

R. Hiptmair

rev 30401, September 27, 2010

Remark 2.1.11 (Gaussian elimination for non-square matrices).

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



R. Hiptmair

rev 30401, September 27, 2010

Simultaneous solving of
LSE with multiple right hand sides

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

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

MATLAB:

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

asymptotic complexity: $O(n^2 m)$

Code 2.1.12: Gaussian elimination with multiple r.h.s.

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

R. Hiptmair

rev 30401, September 28, 2010

2.2 LU-Decomposition/LU-Factorization

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

Mathematical issue: existence & uniqueness
Numerical issue: algorithm for computing factor matrices

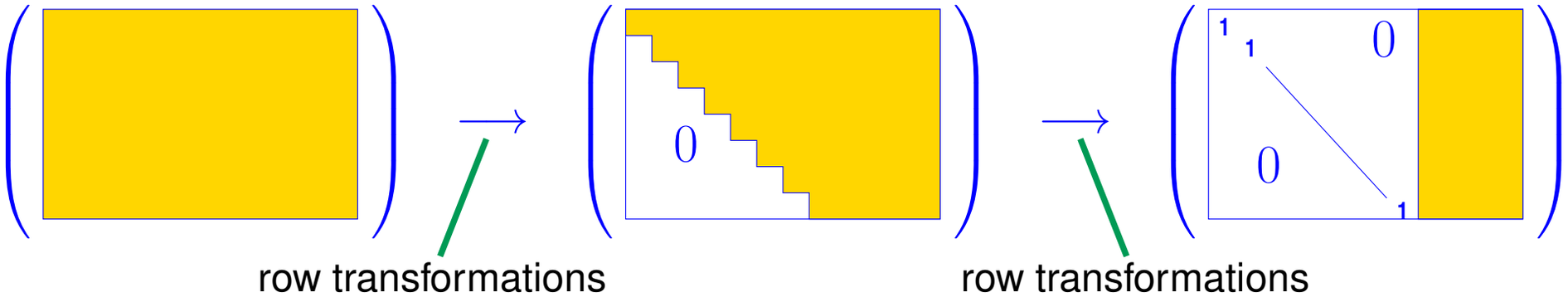
Matrix factorizations

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

The gist of Gaussian elimination:

R. Hiptmair

rev 30401, September 28, 2010



Here: row transformation = adding a multiple of a matrix row to another row, or
multiplying a row with a non-zero scalar (number)
(more special than row transformations discussed in Ex. 1.2.7)

Note: these row transformations preserve regularity of a matrix (why ?)
▷ suitable for transforming linear systems of equations

Example 2.2.1 (Gaussian elimination and LU-factorization). → [39, Sect. 2.4], [29, II.4], [23, Sect. 3.1]

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

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

R. Hiptmair

rev 30401, September 28, 2010

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

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

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

◇

Perspective: link Gaussian elimination to matrix factorization → Ex. 2.2.1

(row transformation = multiplication with elimination matrix)

$$a_1 \neq 0 \quad \blacktriangleright \quad \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ -\frac{a_2}{a_1} & 1 & & & 0 \\ -\frac{a_3}{a_1} & & \ddots & & \\ \vdots & & & & \\ -\frac{a_n}{a_1} & 0 & & & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} .$$

R. Hiptmair

rev 30401, September 28, 2010

$n-1$ steps of Gaussian elimination: ➤ matrix factorization ($\rightarrow$ Ex. 2.1.1)

(non-zero pivot elements assumed)

$$\mathbf{A} = \mathbf{L}_1 \cdot \cdots \cdot \mathbf{L}_{n-1}\mathbf{U} \quad \text{with} \quad \begin{array}{l}\text{elimination matrices } \mathbf{L}_i,\ i = 1,\ldots,n-1\ , \\ \text{upper triangular matrix } \mathbf{U} \in \mathbb{R}^{n,n}\ .\end{array}$$

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

$\mathbf{L}_1 \cdot \cdots \cdot \mathbf{L}_{n-1}$ are normalized lower triangular matrices

(entries = multipliers $-\frac{a_{ik}}{a_{kk}}$ from (2.1.2) $\rightarrow$ Ex. 2.1.1)

R. Hiptmair

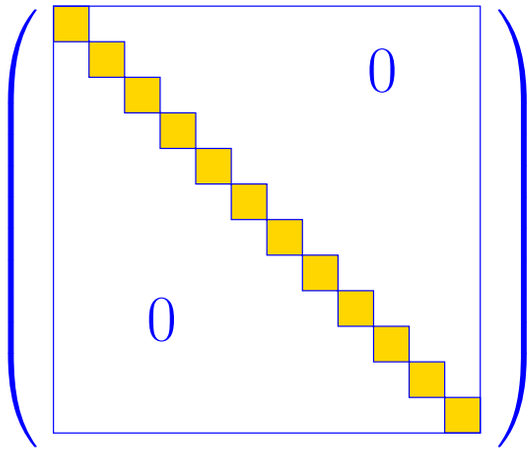
rev 30401, September 28, 2010

Definition 2.2.2 (Types of matrices)**.**

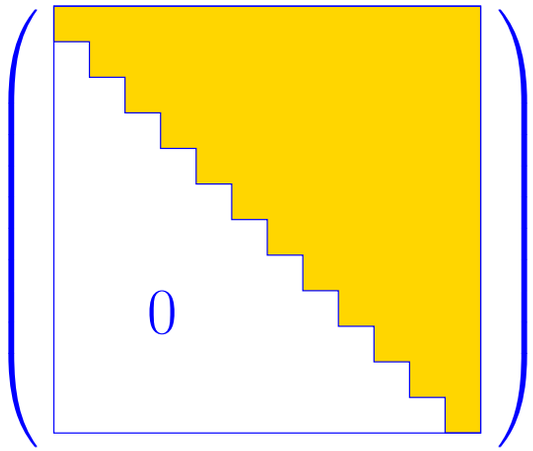
A matrix $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m,n}$ *is*

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

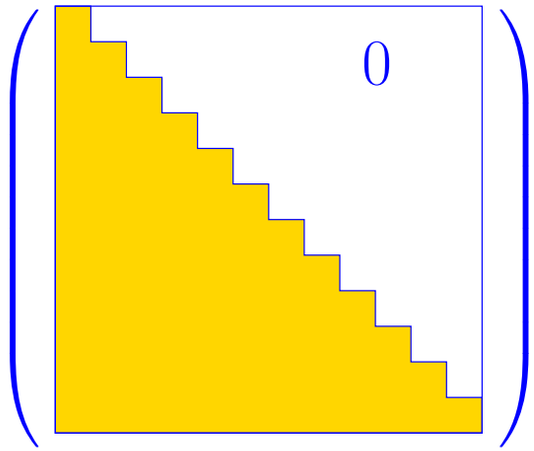
A triangular matrix is normalized, *if* $a_{ii} = 1$, $i = 1, \ldots, \min\{m, n\}$.



diagonal matrix



upper triangular

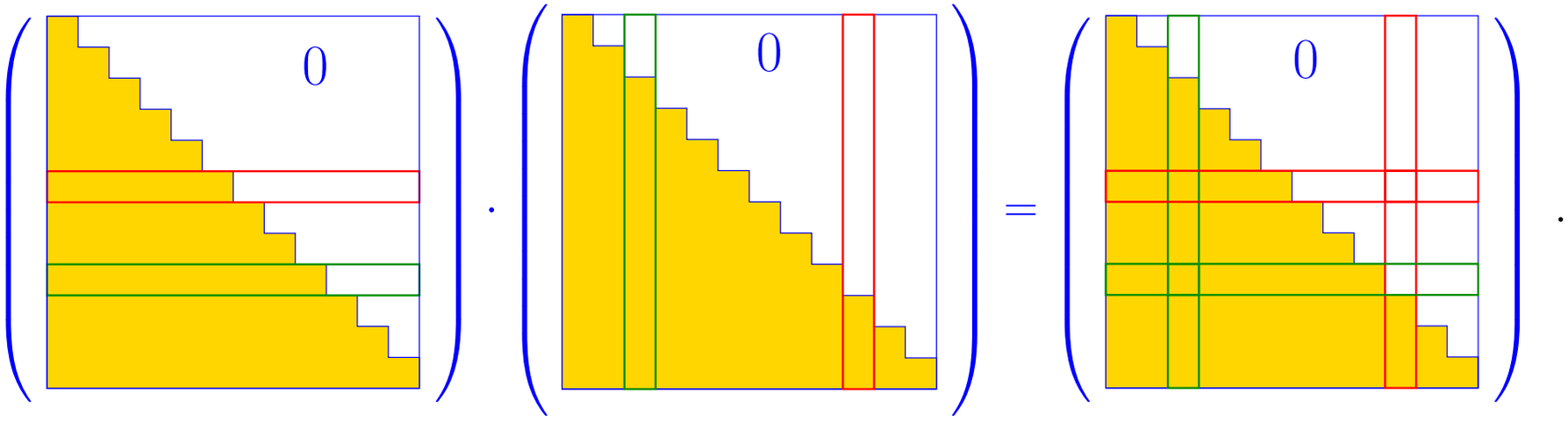


lower triangular

Lemma 2.2.3 (Group of regular diagonal/triangular matrices)**.**

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

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



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 $\mathbf{U}$, [10, Thm. 3.2.1], [39, Thm. 2.10], [23, Sect. 3.1].

Lemma 2.2.4 (Existence of LU-decomposition)**.**

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

Proof. by induction w.r.t. n, which establishes existence of normalized lower triangular matrix $\widetilde{\mathbf{L}}$ and regular upper triangular matrix $\widetilde{\mathbf{U}}$ such that

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

Then solve

❶ $\widetilde{\mathbf{L}}\mathbf{y} = \mathbf{b}$ $\rightarrow$ provides $\mathbf{y} \in \mathbb{K}^n$,

❷ $\mathbf{x}^T\widetilde{\mathbf{U}} = \mathbf{a}^T$ $\rightarrow$ provides $\mathbf{x} \in \mathbb{K}^n$,

❸ $\mathbf{x}^T\mathbf{y} + \xi = \alpha$ $\rightarrow$ provides $\xi \in \mathbb{K}$.

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

R. Hiptmair

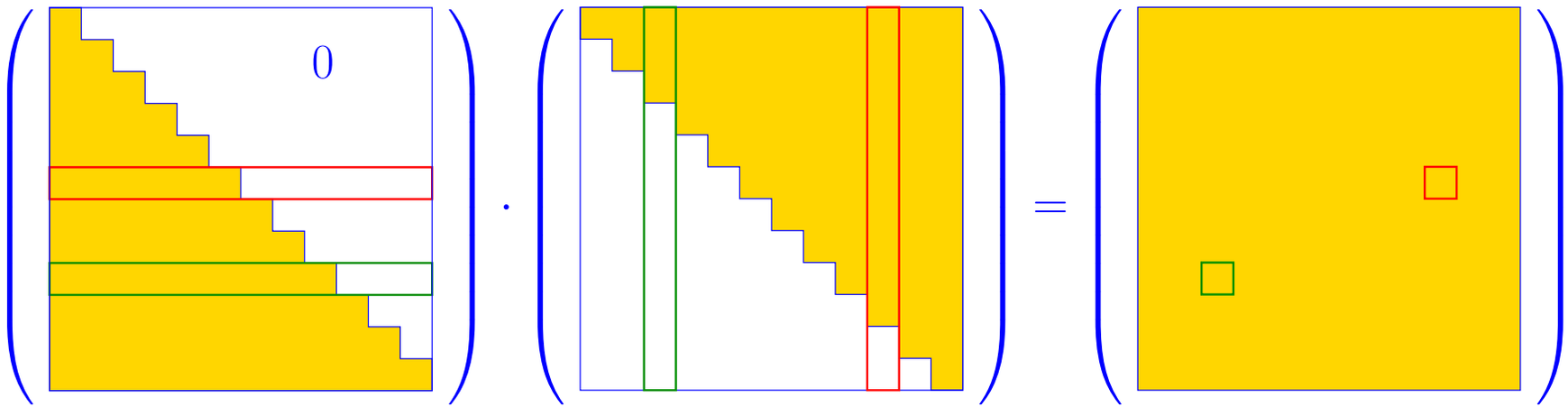
rev 30401, September 28, 2010

Remark: *Uniqueness* of LU-decomposition:

Regular upper triangular matrices and normalized lower triangular matrices form *matrix groups* ($\rightarrow$ Lemma 2.2.3). Their only common element is the identity matrix.

$$\mathbf{L}_1\mathbf{U}_1 = \mathbf{L}_2\mathbf{U}_2 \quad \Rightarrow \quad \mathbf{L}_2^{-1}\mathbf{L}_1 = \mathbf{U}_2\mathbf{U}_1^{-1} = \mathbf{I}\,.$$

A direct way to LU-decomposition [23, Sect. 3.1]:



$$\mathbf{LU} = \mathbf{A} \quad \Rightarrow \quad a_{ik} = \sum_{j=1}^{\min\{i,k\}} l_{ij}u_{jk} = \begin{cases} \sum_{j=1}^{i-1} l_{ij}u_{jk} + 1 \cdot u_{ik} & \text{, if } i \leq k \text{ ,} \\ \sum_{j=1}^{k-1} l_{ij}u_{jk} + l_{ik}u_{kk} & \text{, if } i > k \text{ .} \end{cases} \tag{2.2.5}$$

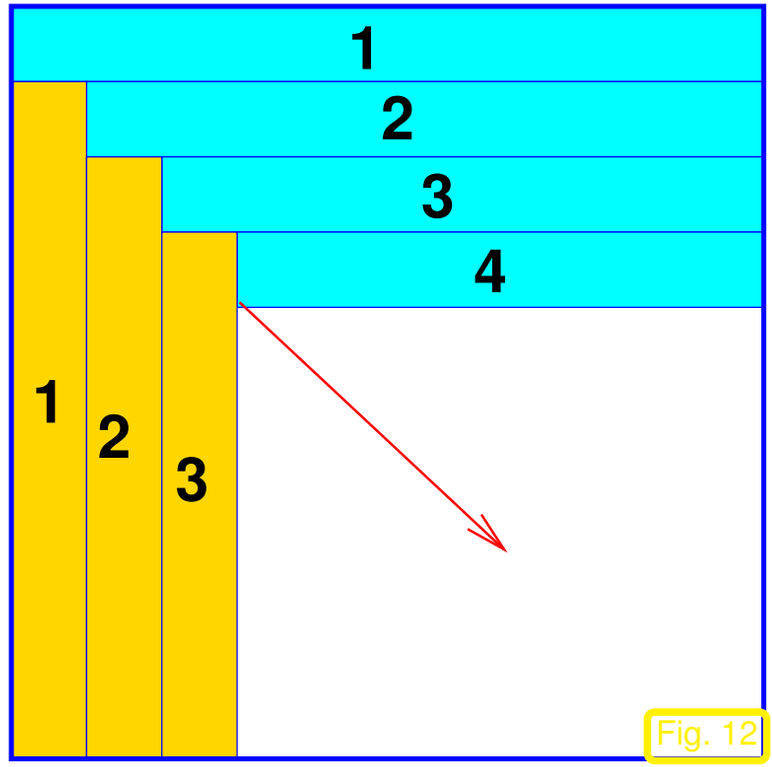


Fig. 12

➤
- 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, [23, Alg. 3.1])

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

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

LU-factorization = "inversion" of matrix multiplication:

Code 2.2.6: LU-factorization

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

Code 2.2.7: matrix multiplication $\mathbf{L} \cdot \mathbf{U}$

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

Observe: Solving for entries `L(i,k)` of $\mathbf{L}$ and `U(k,j)` of $\mathbf{U}$ in the multiplication of an upper trian-

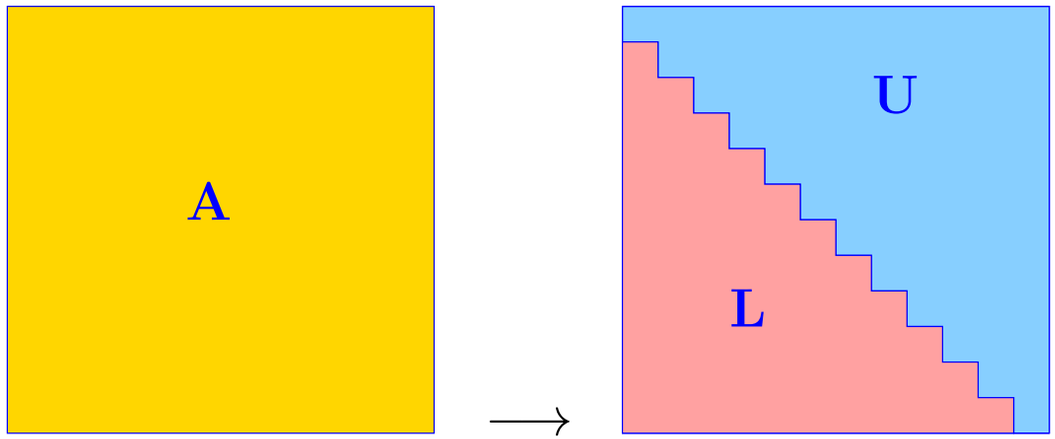
R. Hiptmair

rev 30401, September 28, 2010

gular and normalized lower triangular matrix ($\rightarrow$ Code 2.2.6) yields the algorithm for LU-factorization ($\rightarrow$ Code 2.2.5).

asymptotic complexity of LU-factorization of $\mathbf{A} \in \mathbb{R}^{n,n}$ = $\frac{1}{3}n^3 + O(n^2)$ (2.2.8)

Remark 2.2.9 (In-situ LU-decomposition).



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

△

Remark 2.2.10 (Recursive LU-factorization).

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

$\mathbf{L}$,$\mathbf{U}$ stored in place of $\mathbf{A}$:

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

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

△

Solving a linear system of equations by LU-factorization:

Algorithm 2.2.11 (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 $\frac{1}{2}n(n+1)$

 (in leading order) the same as for Gaussian elimination

Remark 2.2.12 (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 $O(n^3 + Nn^2)$

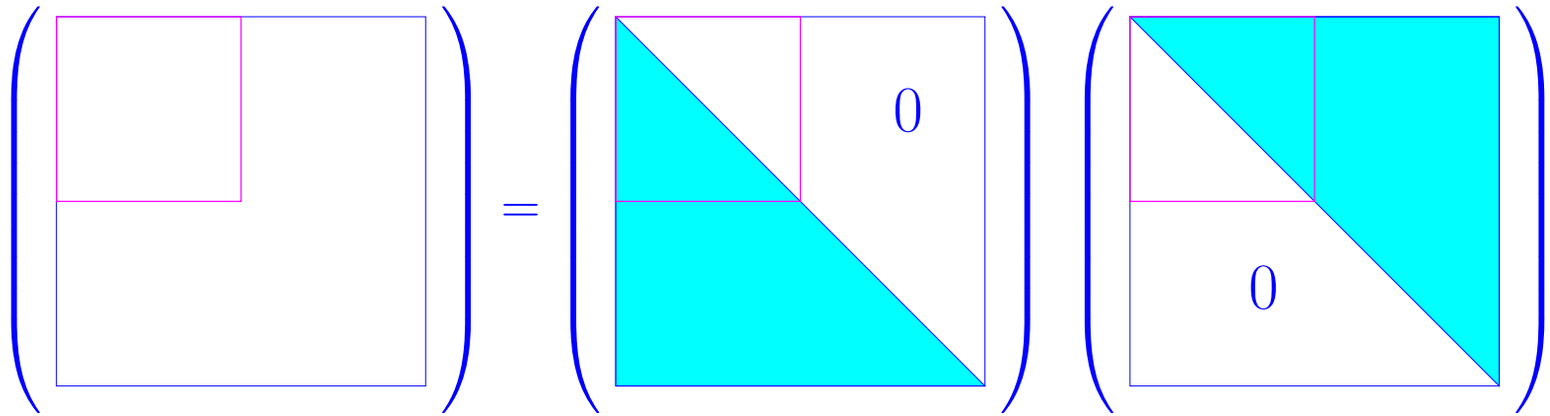
R. Hiptmair

rev 30401, September 28, 2010

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

△

Remark 2.2.13 ("'Partial LU-decompositions" of principal minors).



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

△

Remark 2.2.14. Block matrix multiplication (1.2.10) $\cong$ block LU-decomposition:

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

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & 0 \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{S} \end{pmatrix} \quad , \quad \text{Schur complement } \mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \tag{2.2.15}$$

$\rightarrow$ block Gaussian elimination, see Rem. 2.1.10. △

2.3 Pivoting

Known from linear algebra [39, Sect. 1.1]:

$$\underbrace{\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}}_{\text{breakdown of Gaussian elimination}\atop\text{pivot element } = 0} \qquad \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_2 \\ b_1 \end{pmatrix}}_{\text{Gaussian elimination feasible}}$$

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

Example 2.3.1 (Pivoting and numerical stability). → [10, Bsp. 3.2.3]

```
% Example: numerical instability without pivoting
A = [5.0E-17 , 1; 1 , 1];
b = [1;2];
x1 = A\b,
x2 =gausselim(A,b), % see Code 2.1.11
[L,U] = lufak(A); % see Code 2.2.5
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 → Sect. 2.4

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

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

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

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

LU-factorization after swapping rows:

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

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

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

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

◇

Suitable pivoting essential for controlling impact of roundoff errors
on Gaussian elimination ($\rightarrow$ Sect. 2.5.2, [39, Sect. 2.5])

R. Hiptmair

rev 30401, September 20, 2010

Example 2.3.4 (Gaussian elimination with pivoting for 3×3-matrix).

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 2 \\ 2 & -7 & 2 \\ 1 & 24 & 0 \end{pmatrix} \overset{❶}{\to} \begin{pmatrix} 2 & -7 & 2 \\ 1 & 2 & 2 \\ 1 & 24 & 0 \end{pmatrix} \overset{❷}{\to} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 5.5 & 1 \\ 0 & 27.5 & -1 \end{pmatrix} \overset{❸}{\to} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 27.5 & -1 \\ 0 & 5.5 & 1 \end{pmatrix} \overset{❹}{\to} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 27.5 & -1 \\ 0 & 0 & 1.2 \end{pmatrix}$$

❶: swap rows 1 & 2.

❷: elimination with top row as pivot row

❸: swap rows 2 & 3

❹: elimination with 2nd row as pivot row

◇

Algorithm 2.3.5.

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

Code 2.3.6: recursive Gaussian elimination with row pivoting

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

```
8       C = gsrecpiv(A(2:end,2:end)-fac*A(1,2:end)); %
9       A = [A(1,:) ; -fac, C ]; %
10  end
```

choice of pivot row index j, relatively largest pivot:

$$j \in \{i, \ldots, n\} \text{ such that } \frac{|a_{ki}|}{\sum_{l=i}^{n} |a_{kl}|} \to \max \tag{2.3.7}$$

for $k = j$, $k \in \{i, \ldots, n\}$. (relatively largest pivot element p)

Explanations to Code 2.3.5:

Line 4: Find the *relatively largest* pivot element p and the index j of the corresponding row of the matrix, see (2.3.9)

Line 5: If the pivot element is still very small relative to the norm of the matrix, then we have encountered an entire column that is close to zero. The matrix is (close to) singular and LU-factorization does not exist.

Line 6: Swap the first and the j-th row of the matrix.

Line 7: Initialize the vector of multiplier.

Line 8: Call the routine for the upper right $(n-1) \times (n-1)$-block of the matrix after subtracting suitable multiples of the first row from the other rows, *cf.* Rem. 2.1.8 and Rem. 2.2.10.

Line 9: Reassemble the parts of the LU-factors. The vector of multipliers yields a column of $\mathbf{L}$, see Ex. 2.2.1.

Algorithm 2.3.8.

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

Row permutations recorded in vector `p` !

Usual choice of pivot: $j \in \{i, \ldots, n\}$ such that

$$\frac{|a_{ki}|}{\sum_{l=i}^{n} |a_{kl}|} \to \max \qquad (2.3.9)$$

for $k = j$, $k \in \{i, \ldots, n\}$.

(relatively largest pivot element)

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

Example 2.3.10 (Rationale for partial pivoting policy (2.3.9)).

R. Hiptmair

rev 30401, September 20, 2010

Why *relatively* largest pivot element in (2.3.9)? scaling invariance desirable

Scale linear system of equations from Ex. 2.3.1:

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

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

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

```
1 % Example: importance of scale-invariant
    pivoting
2 epsilon = 5.0E-17;
3 A = [epsilon , 1; 1 , 1]; b = [1;2];
4 D = [1/epsilon, 0; 0 ,1];
5 A = D*A; b = D*b;
6 x1 = A\b, % MATLAB internal Gaussian
    elimination
7 x2 =gausselim(A,b), % see Code 2.1.11
8 [L,U] = lufak(A); % see Code 2.2.5
9 z = L\b; x3 = U\z,
```

Ouput of MATLAB run:

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

Theoretical foundation of algorithm 2.3.5:

Definition 2.3.11 (Permutation matrix)**.**

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

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

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

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

Lemma 2.3.12 (Existence of LU-factorization with pivoting). $\rightarrow$ *[10, Thm. 3.25], [29, Thm. 4.4]*

For any regular $\mathbf{A} \in \mathbb{K}^{n,n}$ there is a permutation matrix $\mathbf{P} \in \mathbb{K}^{n,n}$, 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. 2.2.2), such that $\mathbf{PA} = \mathbf{LU}$.

Proof. (by induction)

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

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

$$\begin{pmatrix} \mathbf{P}' & 0 \\ 0 & 1 \end{pmatrix} \mathbf{PA} = \begin{pmatrix} \mathbf{P}' & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A}' & \mathbf{x} \\ \mathbf{y}^T & \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{L}'\mathbf{U}' & \mathbf{x} \\ \mathbf{y}^T & \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{L}' & 0 \\ \mathbf{c}^T & 1 \end{pmatrix} \begin{pmatrix} \mathbf{U} & \mathbf{d} \\ 0 & \alpha \end{pmatrix} ,$$

if we choose

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

which is always possible. □

Example 2.3.13 (Ex. 2.3.4 cnt'd).

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 2 \\ 2 & -7 & 2 \\ 1 & 24 & 0 \end{pmatrix} \overset{❶}{\to} \begin{pmatrix} 2 & -7 & 2 \\ 1 & 2 & 2 \\ 1 & 24 & 0 \end{pmatrix} \overset{❷}{\to} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 5.5 & 1 \\ 0 & 27.5 & -1 \end{pmatrix} \overset{❸}{\to} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 27.5 & -1 \\ 0 & 5.5 & 1 \end{pmatrix} \overset{❹}{\to} \begin{pmatrix} 2 & -7 & 2 \\ 0 & 27.5 & -1 \\ 0 & 0 & 1.2 \end{pmatrix}$$

$$\mathbf{U} = \begin{pmatrix} 2 & -7 & 2 \\ 0 & 27.5 & -1 \\ 0 & 0 & 1.2 \end{pmatrix} , \quad \mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0.2 & 1 \end{pmatrix} , \quad \mathbf{P} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} .$$

Two permutations: in step ❶ swap rows #1 and #2, in step ❸ swap rows #2 and #3. Apply these swaps to the identity matrix and you will recover $\mathbf{P}$.

◇

R. Hiptmair

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

Remark 2.3.14 (Row swapping commutes with forward elimination).

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

The LU-factorization of $\mathbf{A} \in \mathbb{K}^{n,n}$ with partial pivoting by Alg. 2.3.5 is *numerically equivalent* to the LU-factorization of $\mathbf{PA}$ without pivoting ($\rightarrow$ Code 2.2.5), when $\mathbf{P}$ is a permutation matrix gathering the row swaps entailed by partial pivoting.

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

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

2.4 Supplement: Machine Arithmetic

→ [10, Sect. 2.2].

Supplementary and further reading:

A very detailed exposition and in-depth discussion of the all the material in this section can be found in [30]:

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

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

machine numbers, set $\mathbb{M}$

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

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

Computers cannot compute “properly” !

▼

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

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

"Computers use floating point numbers (scientific notation)"

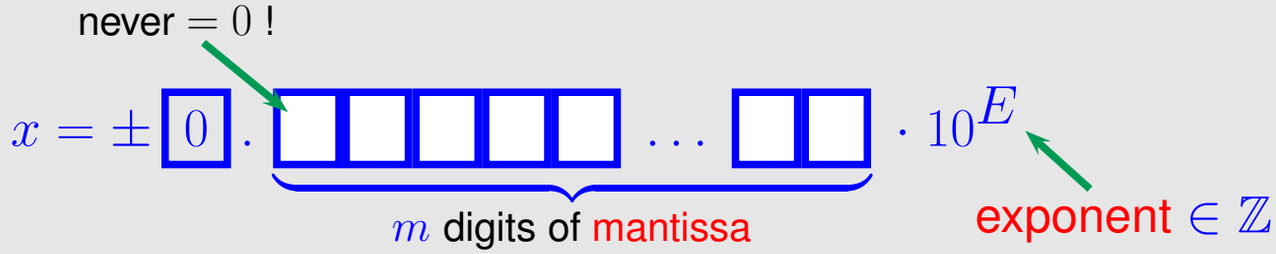
Example 2.4.1 (Decimal floating point numbers).

3-digit normalized decimal floating point numbers:

valid: $0.723 \cdot 10^2$, $0.100 \cdot 10^{-20}$, $-0.801 \cdot 10^5$

invalid: $0.033 \cdot 10^2$, $1.333 \cdot 10^{-4}$, $-0.002 \cdot 10^3$

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



$\diamond$

R. Hiptmair

rev 30401, October 14, 2010

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

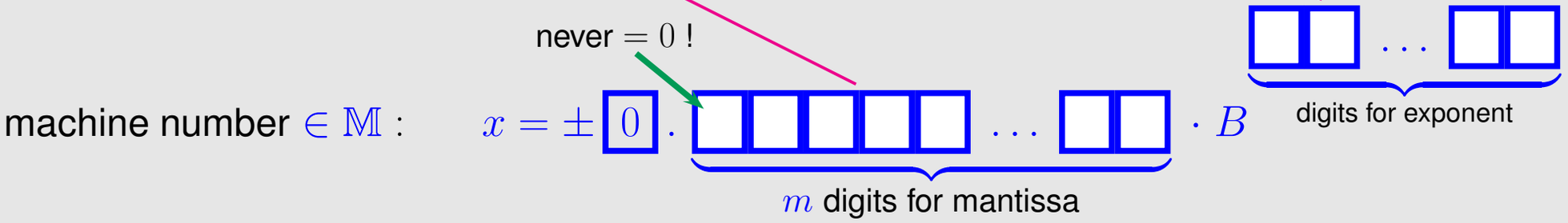
Definition 2.4.2 (Machine numbers/floating point numbers)**.**

Given

- ☞ *basis* $B \in \mathbb{N} \setminus \{1\}$,
- ☞ *exponent range* $\{e_{\min}, \ldots, e_{\max}\}$, $e_{\min}, e_{\max} \in \mathbb{Z}$, $e_{\min} < e_{\max}$,
- ☞ *number* $m \in \mathbb{N}$ *of digits (for mantissa),*

the corresponding set of machine numbers is

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



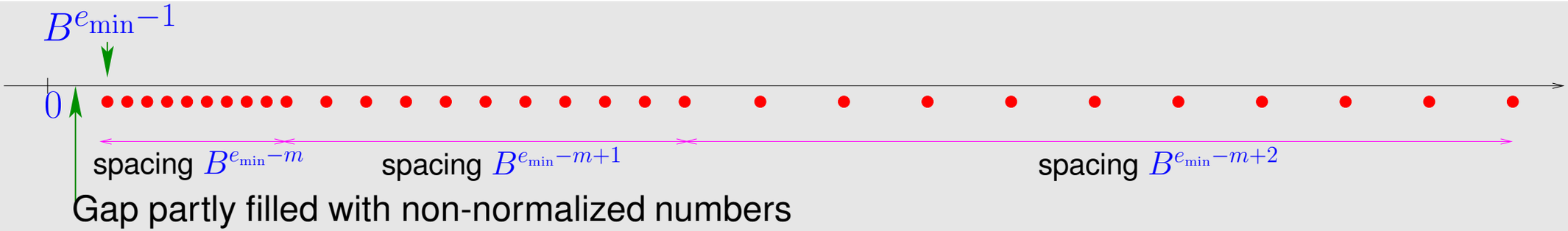
Largest machine number (in modulus) : $x_{\max} = (1 - B^{-m}) \cdot B^{e_{\max}}$

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

Distribution of machine numbers:

R. Hiptmair

rev 30401, October 14, 2010



Example 2.4.3 (IEEE standard 754 for machine numbers). $\rightarrow$ [40], $\rightarrow$ link

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

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

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

$*$: including bit indicating sign

$\diamond$

Remark 2.4.4 (Special cases in IEEE standard).

R. Hiptmair

rev 30401, October 14, 2010

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

$E = e_{\max}$, $M \neq 0 \mathrel{\hat{=}}$ NaN = *Not a number* $\rightarrow$ exception
$E = e_{\max}$, $M = 0 \mathrel{\hat{=}}$ Inf = *Infinity* $\rightarrow$ overflow
$E = 0 \mathrel{\hat{=}}$ Non-normalized numbers $\rightarrow$ underflow
$E = 0$, $M = 0 \mathrel{\hat{=}}$ number 0

△

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

☞ MATLAB *always* uses double precision

```
>> format hex; realmin, format long; realmin
ans = 0010000000000000
ans = 2.225073858507201e-308
```

```
>> format hex; realmax, format long; realmax
ans = 7fefffffffffffff
ans = 1.797693134862316e+308
```


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
```


◇

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

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

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

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

Assumption 2.4.10 ("Axiom" of roundoff analysis)**.**

There is a small positive number 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 $|\delta| < \mathtt{eps}$*.*

R. Hiptmair

rev 30401, October 14, 2010

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

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

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

Code 2.4.12: Finding out eps in MATLAB

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

◇

Remark 2.4.13 (Adding eps to 1).

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

Code 2.4.14: $1+\epsilon$ in MATLAB

```
1 >> fprintf('%30.25f\n',1+0.5*eps)
2    1.0000000000000000000000000
3 >> fprintf('%30.25f\n',1-0.5*eps)
4    0.9999999999999998889776975
5 >>
  fprintf('%30.25f\n',(1+2/eps)-2/eps);
6    0.0000000000000000000000000
```

In fact $1\widetilde{+}\mathtt{eps} = 1$ would comply with the "axiom" of roundoff error analysis, Ass. 2.4.10:

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

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

△

R. Hiptmair

rev 30401, October 14, 2010

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 [10, Sect. 2.1.2], [29, Sect. 1.2]

Norms provide tools for measuring errors. Recall from linear algebra and calculus:

Definition 2.5.1 (Norm).

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

(i) $\forall \mathbf{x} \in X\colon \quad \mathbf{x} \neq 0 \quad \Leftrightarrow \quad \|\mathbf{x}\| > 0$ *(definite),*

(ii) $\|\lambda \mathbf{x}\| = |\lambda| \|\mathbf{x}\| \quad \forall \mathbf{x} \in X, \lambda \in \mathbb{K}$ *(homogeneous),*

(iii) $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in X$ *(triangle inequality).*

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

name	:	definition	MATLAB function
Euclidean norm	:	$\|\mathbf{x}\|_2 := \sqrt{\|x_1\|^2 + \cdots + \|x_n\|^2}$	`norm(x)`
1-norm	:	$\|\mathbf{x}\|_1 := \|x_1\| + \cdots + \|x_n\|$	`norm(x,1)`
∞-norm, max norm	:	$\|\mathbf{x}\|_\infty := \max\{\|x_1\|, \ldots, \|x_n\|\}$	`norm(x,inf)`

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

$$\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1 \leq \sqrt{n}\,\|\mathbf{x}\|_2 \;, \tag{2.5.2}$$

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{n}\,\|\mathbf{x}\|_\infty \;, \tag{2.5.3}$$

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_1 \leq n\,\|\mathbf{x}\|_\infty \;. \tag{2.5.4}$$

Definition 2.5.5 (Matrix norm)**.**

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

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

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

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

Example 2.5.6 (Matrix norm associated with ∞-norm and 1-norm).

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

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

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

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

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

◇

Special formulas for Euclidean matrix norm [20, Sect. 2.3.3]:

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

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

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

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

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

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

On the other hand, for the same reason:

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

Corollary 2.5.10 (2-Matrixnorm and eigenvalues)**.**
For $\mathbf{A} \in \mathbb{K}^{m,n}$ the 2-Matrixnorm $\|\mathbf{A}\|_2$ is the square root of the largest (in modulus) eigenvalue of $\mathbf{A}^H \mathbf{A}$.

2.5.2 Numerical Stability [10, Sect. 2.3]

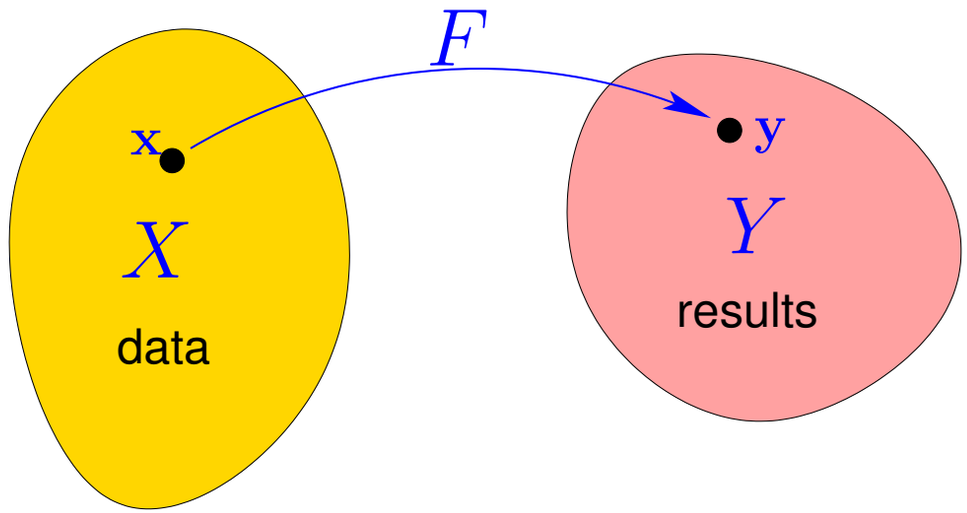
Supplementary and further reading:

Also see the article [5] for a simplified but powerful presentation of stability analysis of numerical algorithms.

Abstract point of view:

Our notion of "problem":

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



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

"The problem:"

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

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

- Judgement about the stability of an algorithm depends on the chosen norms !
- What is meant by $C \approx 1$ in practice ?

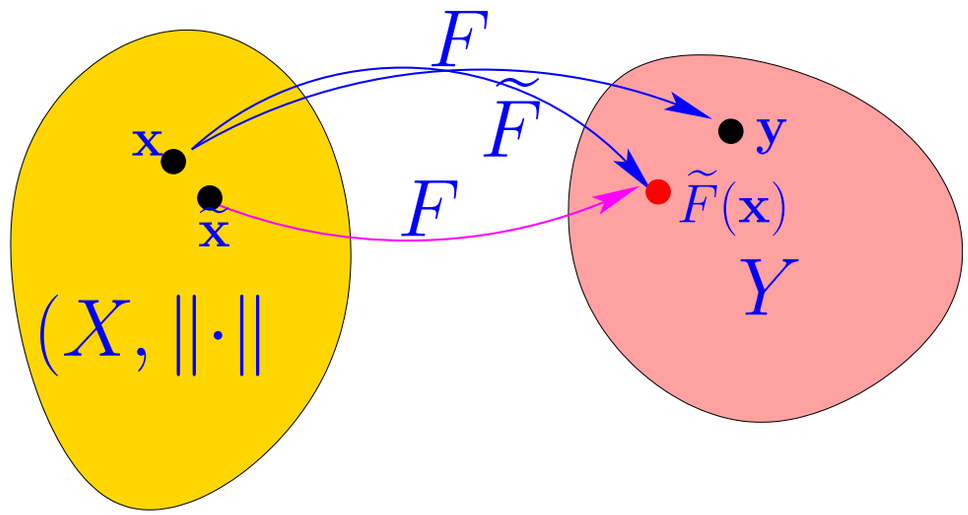
  $C \approx 1 \leftrightarrow C \approx$ no. of elementary operations for computing $\widetilde{F}(\mathbf{x})$: The longer a computation takes the more accumulation of roundoff errors we are willing to tolerate.
- The use of computer arithmetic involves inevitable relative input errors ($\rightarrow$ Ex. 2.4.6) of the size of eps. Moreover, in most applications the input data are also affected by other (e.g, measurement) errors. Hence stability means that

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

➢ for stable algorithms roundoff errors are “harmless”.

Terminology:

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



2.5.3 Roundoff analysis of Gaussian elimination

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

Lemma 2.5.12 (Equivalence of Gaussian elimination and LU-factorization)**.**
The following algorithms for solving the LSE $\mathbf{Ax} = \mathbf{b}$ *(*$\mathbf{A} \in \mathbb{K}^{n,n}$, $\mathbf{b} \in \mathbb{K}^n$*) are* numerically equivalent*:*

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

❷ *LU-factorization of* $\mathbf{A}$ *(*$\rightarrow$ *Code 2.2.5) followed by forward and backward substitution, see Algorithm 2.2.11.*

Rem. 2.3.14 ➢ sufficient to consider LU-factorization without pivoting

A profound roundoff analysis of Gaussian eliminatin/LU-factorization can be found in [20, Sect. 3.3 & 3.5] and [30, Sect. 9.3]. A less rigorous, but more lucid discussion is given in [54, Lecture 22].

Here we only quote a result due to Wilkinson, [30, Thm. 9.5]:

Theorem 2.5.13 (Stability of Gaussian elimination with partial pivoting)**.**
Let $\mathbf{A} \in \mathbb{R}^{n,n}$ be regular and $\mathbf{A}^{(k)} \in \mathbb{R}^{n,n}$, $k = 1, \ldots, n-1$, denote the intermediate matrix arising in the k-th step of Algorithm 2.3.5 (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 2.3.5 (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

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

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

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

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

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

Example 2.5.14 (Wilkinson's counterexample).

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

R. Hiptmair

rev 30401, October 14, 2010

Partial pivoting does not trigger row permutations !

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

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

◇

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

Discussion in [54, Lecture 22]: growth factors larger than the order$O(\sqrt{n})$ are exponentially rare in certain relevant classes of *random matrices*.

Example 2.5.15 (Stability by small random perturbations).

```
% 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);
```


R. Hiptmair

rev 30401, October 14, 2010

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


R. Hiptmair

rev 30401, October 14, 2010

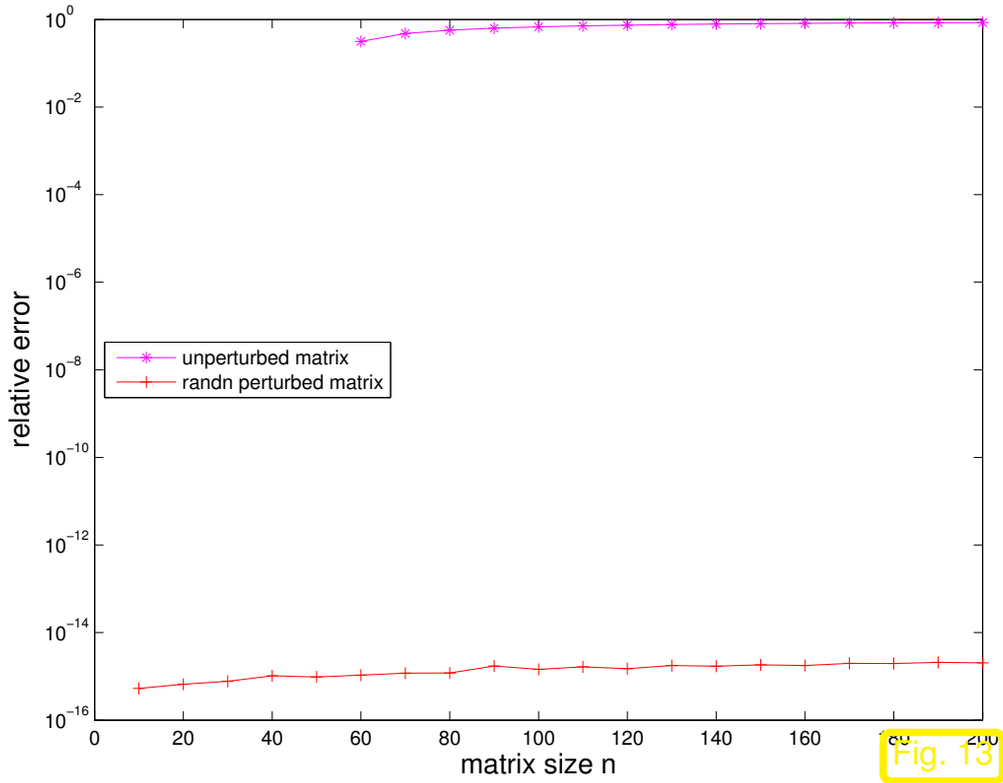


Fig. 13

Recall statement in Sect. 2.5.3 about "improbability" of matrices for which Gaussian elimination with partial pivoting is unstable. This is now matched by the observation that a *tiny random* perturbation of the matrix (almost certainly) cures the problem. This is investigated by the brand-new field of smoothed analysis of numerical algorithms, see [46].

◇

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*

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

Simple consideration:

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

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

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

R. Hiptmair

rev 30401, October 14, 2010

Numerical experiment with *nearly singular matrix*

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

$\mathbf{u}\mathbf{v}^T$: singular rank-1 matrix

with

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

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 30401, October 14, 2010

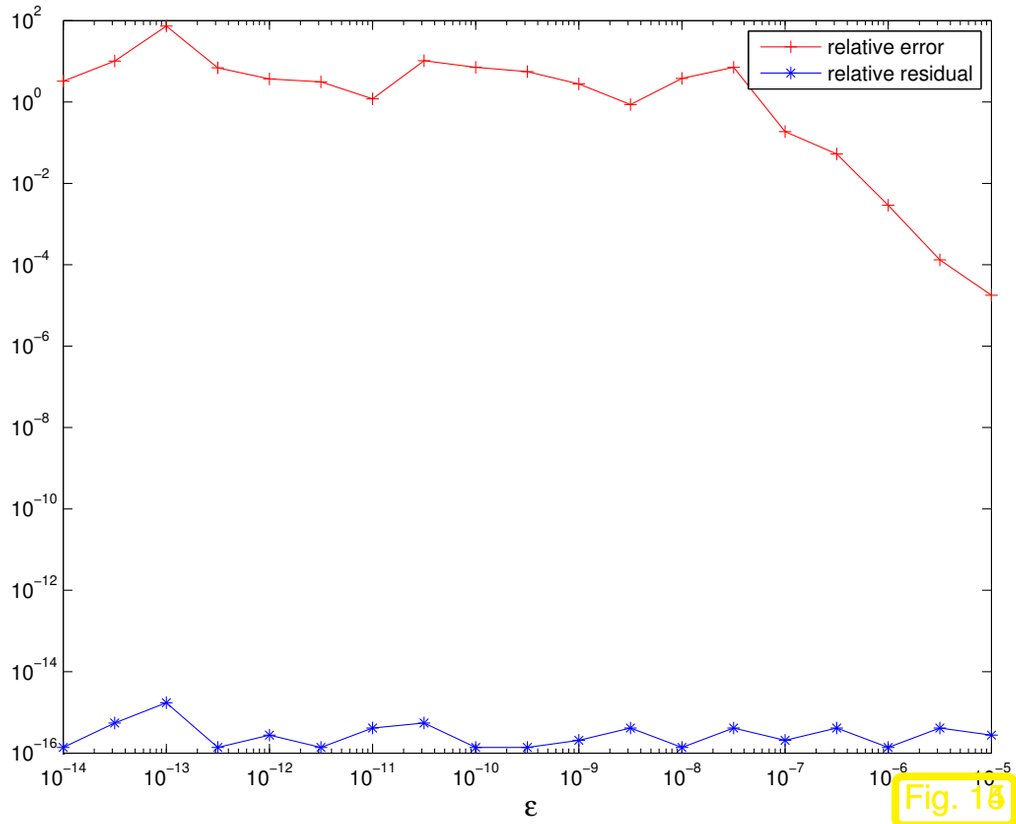


Fig. 15

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

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

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

$$\left\{ \begin{array}{lll} \mathbf{A}(\mathbf{x}-\widetilde{\mathbf{x}}) = \mathbf{r} & \Rightarrow & \|\mathbf{x}-\widetilde{\mathbf{x}}\| \leq \left\|\mathbf{A}^{-1}\right\| \|\mathbf{r}\| \\ \mathbf{A}\mathbf{x} = \mathbf{b} & \Rightarrow & \|\mathbf{b}\| \leq \|\mathbf{A}\| \|\mathbf{x}\| \end{array} \right. \quad \Rightarrow \quad \frac{\|\mathbf{x}-\widetilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}\| \left\|\mathbf{A}^{-1}\right\| \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \,. \tag{2.5.19}$$

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

R. Hiptmair

rev 30401, October 14, 2010

Example 2.5.20 (Instability of multiplication with inverse).

Nearly singular matrix from Ex. 2.5.17

Code 2.5.22: instability of multiplication with inverse

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

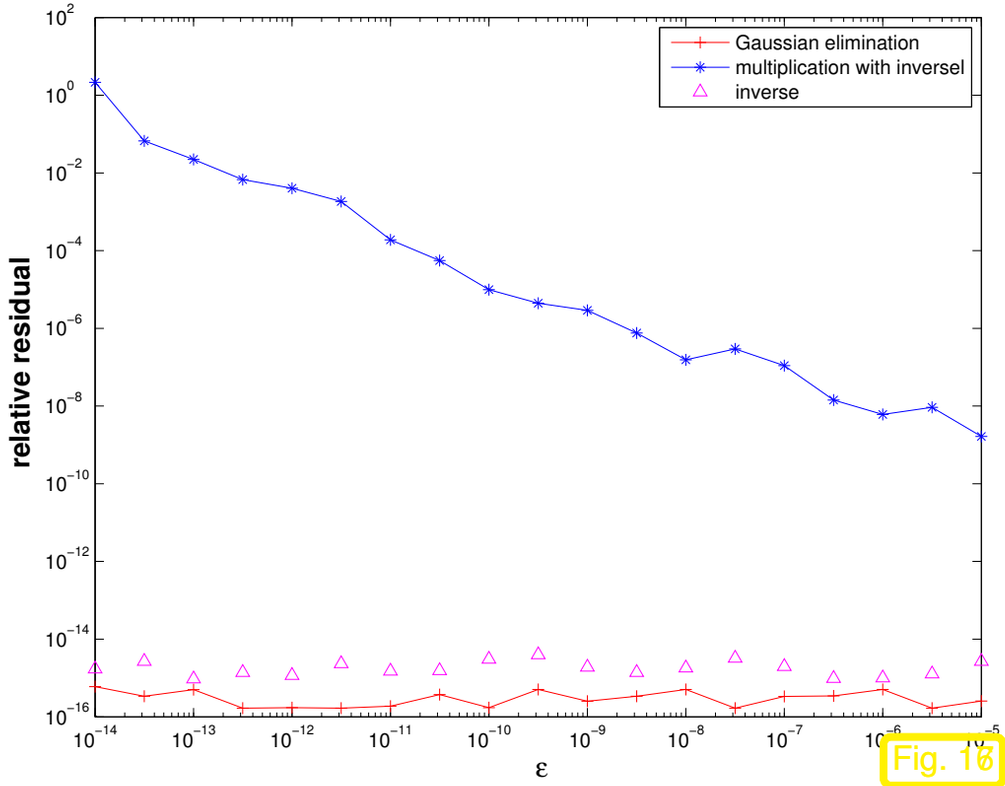


Fig. 16

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

R. Hiptmair

rev 30401, October 14, 2010

2.5.4 Conditioning

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

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

$$\text{(normwise) relative error:} \qquad \epsilon_r := \frac{\|\mathbf{x} - \widetilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \,.$$

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

Perturbed linear system:

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

Theorem 2.5.24 (Conditioning of LSEs)**.**

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

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

relative error — *relative perturbations*

The proof is based on the following fundamental result:

Lemma 2.5.25 (Perturbation lemma)**.**

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

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

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

R. Hiptmair

rev 30401, October 14, 2010

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

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

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

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

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

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

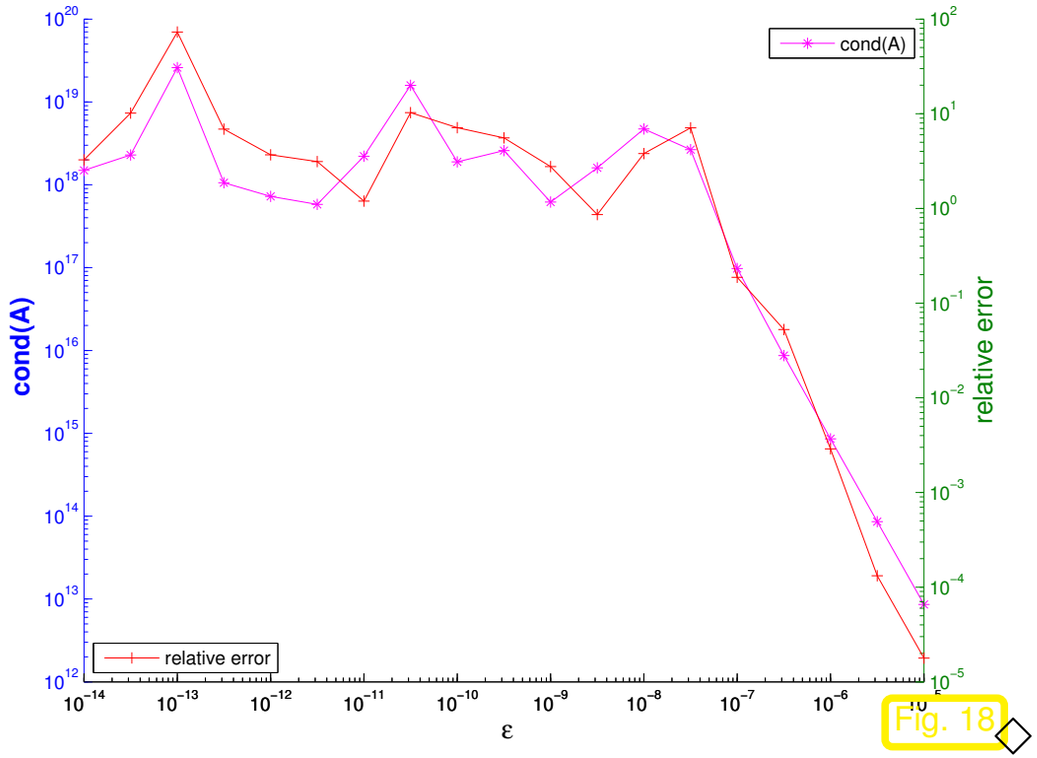


Fig. 18

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

R. Hiptmair

rev 30401, October 14, 2010

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-*');
```

(*) If $\mathrm{cond}_2(\mathbf{A})$ was huge, then big errors in the solution of a linear system can be caused by small perturbations of either the system matrix or the right hand side vector, see (2.5.19) and the message of Thm. 2.5.24, (2.5.27). In this case, a stable algorithm can obviously produce a grossly "wrong" solution, as was already explained after (2.5.27).
Hence, lack of stability of Gaussian elimination will only become apparent for linear systems with well-conditioned system matrices.

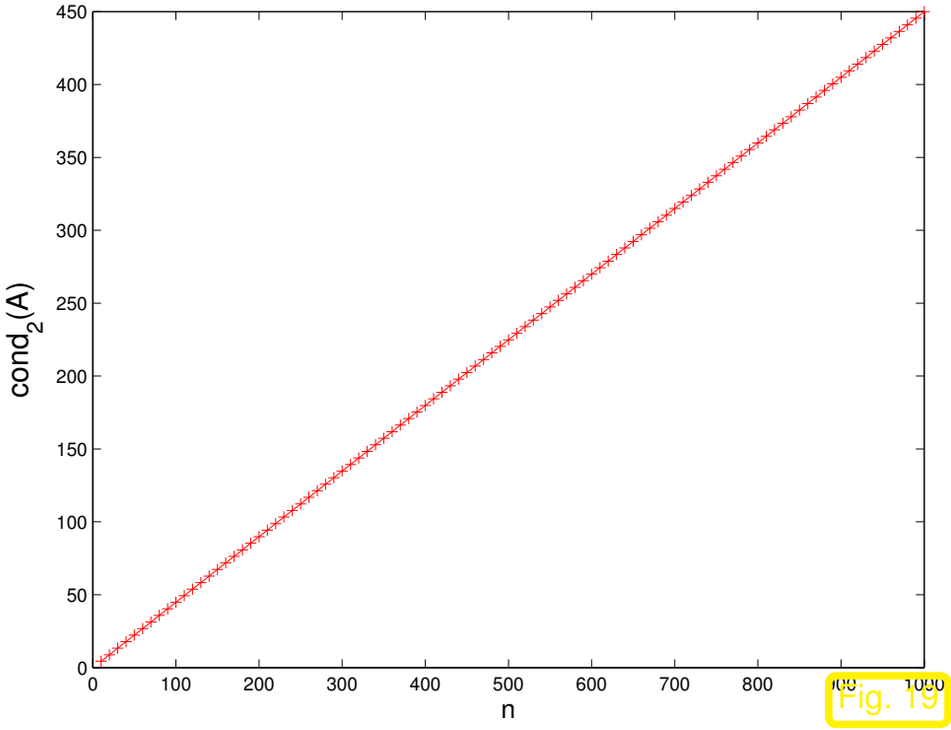


Fig. 19

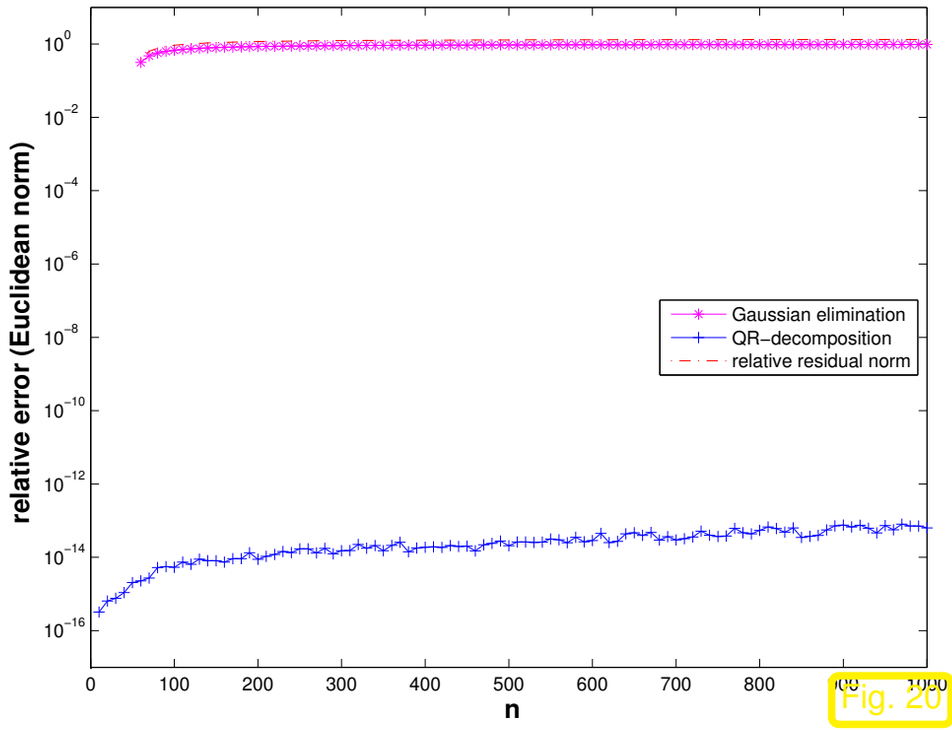


Fig. 20

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

$\diamond$

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 $\|\cdot\|$

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

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

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

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

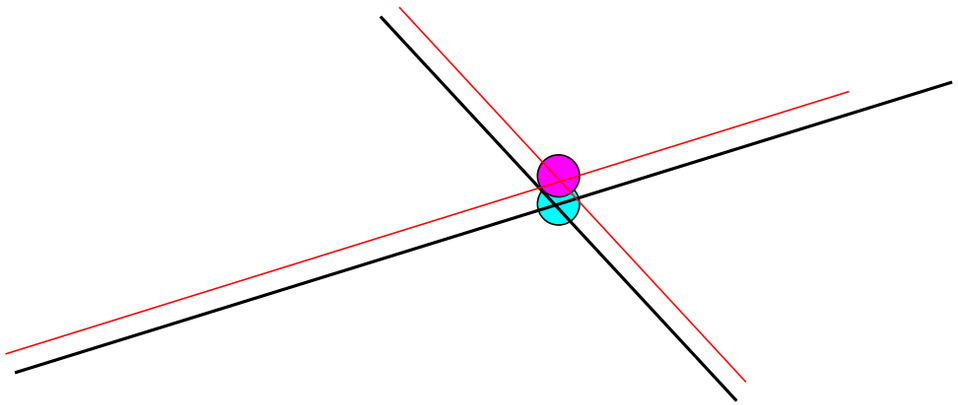
Terminology:

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

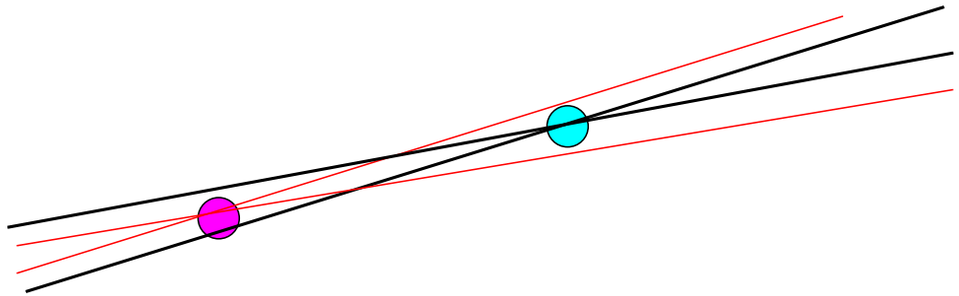
Note: sensitivity gauge depends on the chosen norm **!**

Example 2.5.32 (Intersection of lines in 2D).

In distance metric:



nearly orthogonal intersection: well-conditioned



glancing intersection: ill-conditioned

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

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

R. Hiptmair

rev 30401, October 14, 2010

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';
```

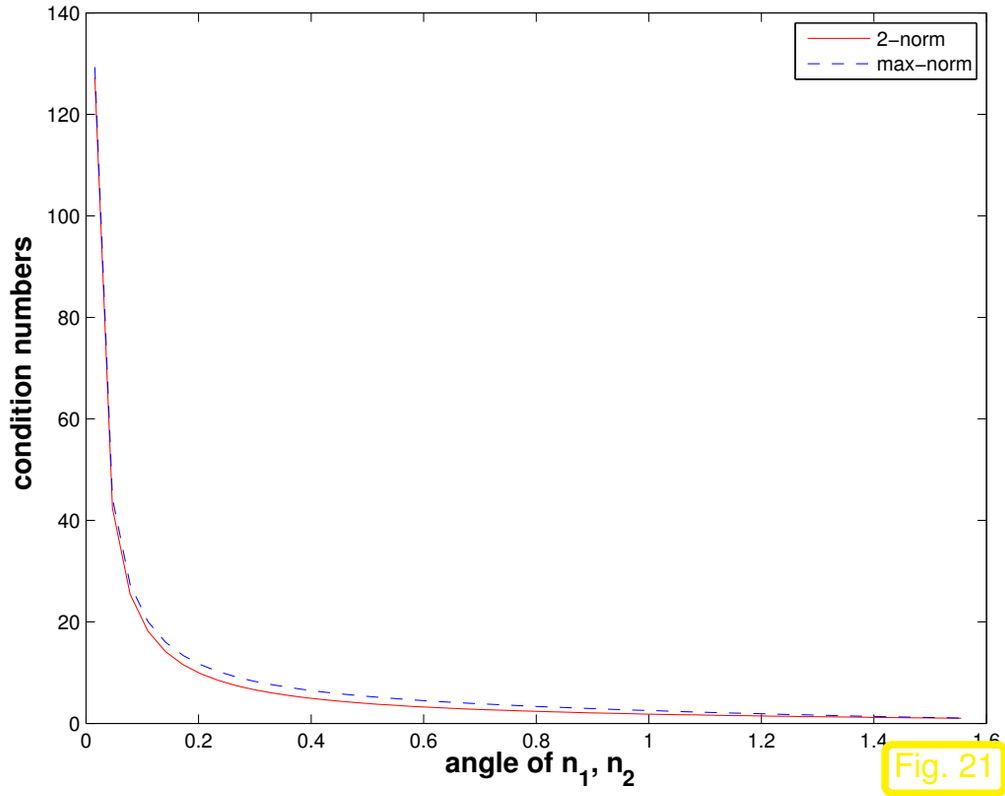


Fig. 21

◇

Heuristics:

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

R. Hiptmair

rev 30401, January 28, 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*

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

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

R. Hiptmair

rev 30401, January 28, 2011

A more rigorous "mathematical" definition:

Definition 2.6.2 (Sparse matrices)**.**
Given a strictly increasing sequences $m : \mathbb{N} \mapsto \mathbb{N}$, $n : \mathbb{N} \mapsto \mathbb{N}$, a family $(\mathbf{A}^{(l)})_{l \in \mathbb{N}}$ of matrices with $\mathbf{A}^{(l)} \in \mathbb{K}^{m_l,n_l}$ is sparse (opposite: dense), if

$$\operatorname{nnz}(\mathbf{A}^{(l)}) := \#\{(i,j) \in \{1,\ldots,m_l\} \times \{1,\ldots,n_l\} \colon a_{ij}^{(l)} \neq 0\} = O(n_l + m_l) \quad \text{for} \quad l \to \infty \ .$$

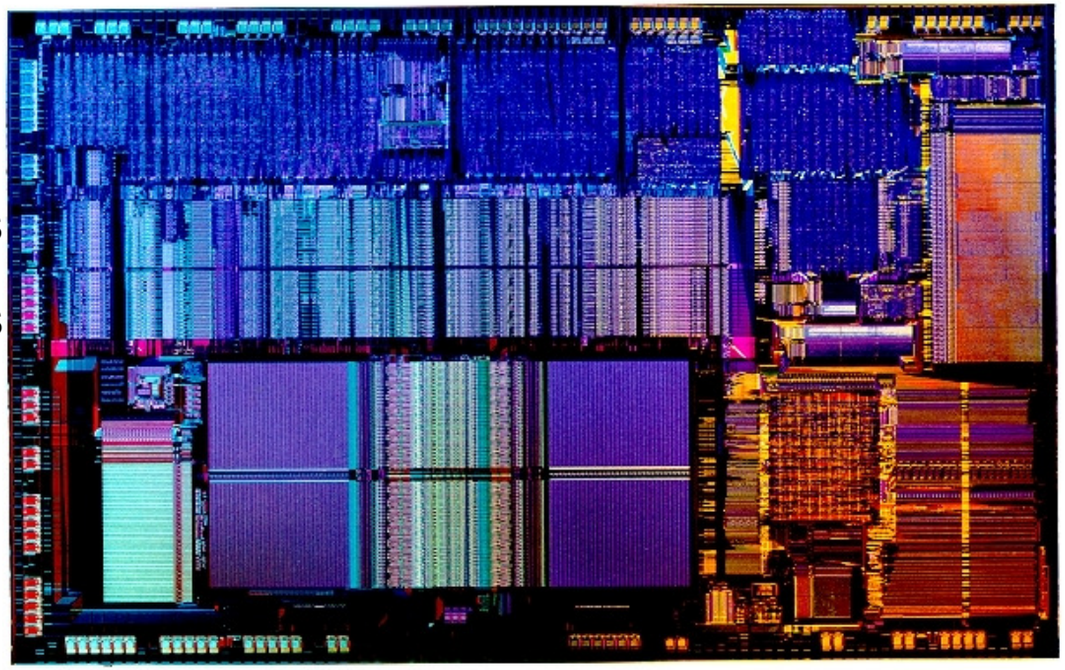
Example 2.6.3 (Sparse LSE in circuit modelling).

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

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

[In the case of a linear circuit]

nodal analysis ➤ sparse circuit matrix



◇

Another important context in which sparse matrices usually arise:

☛ discretization of boundary value problems for partial differential equations ($\rightarrow$ 4th semester course “Numerical treatment of PDEs”

2.6.1 Sparse matrix storage formats

Special sparse matrix storage formats store *only* non-zero entries:
(➢ usually $O(n+m)$ storage required for sparse $n \times m$-matrix)

- Compressed Row Storage (CRS)
- Compressed Column Storage (CCS) $\rightarrow$ 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 2.6.4 (Compressed row-storage (CRS) format).

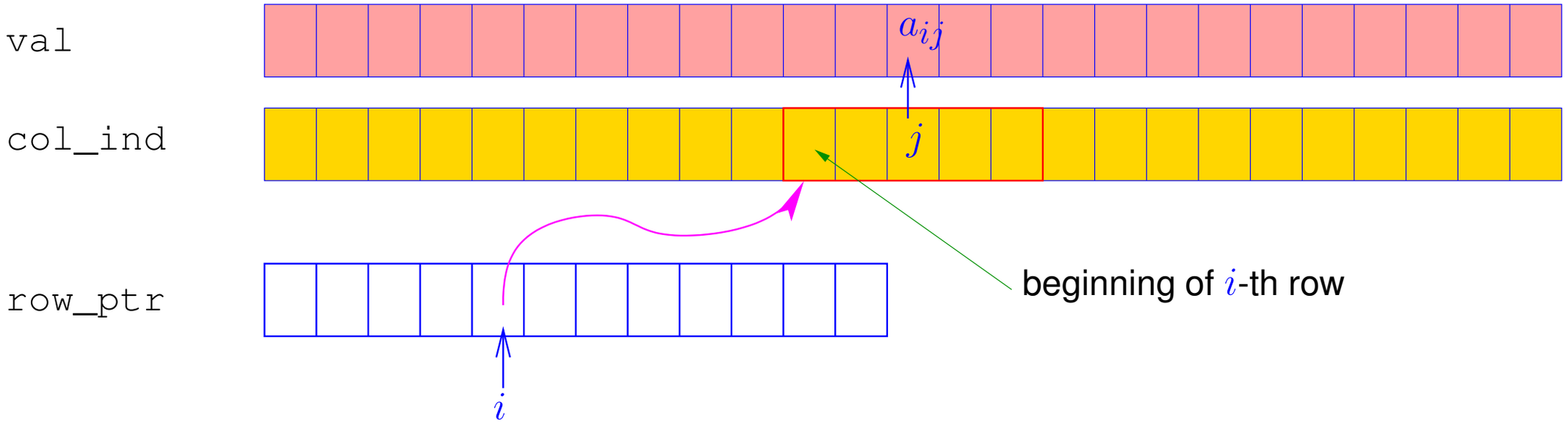
Data for matrix $\mathbf{A} = (a_{ij}) \in \mathbb{K}^{n,n}$ kept in three arrays

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

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

◇

2.6.2 Sparse matrices in MATLAB

Supplementary and further reading:

A detailed discussion of sparse matrix formats and how to work with them efficiently is given in [19].

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 2.6.5 (Accessing rows and columns of sparse matrices).

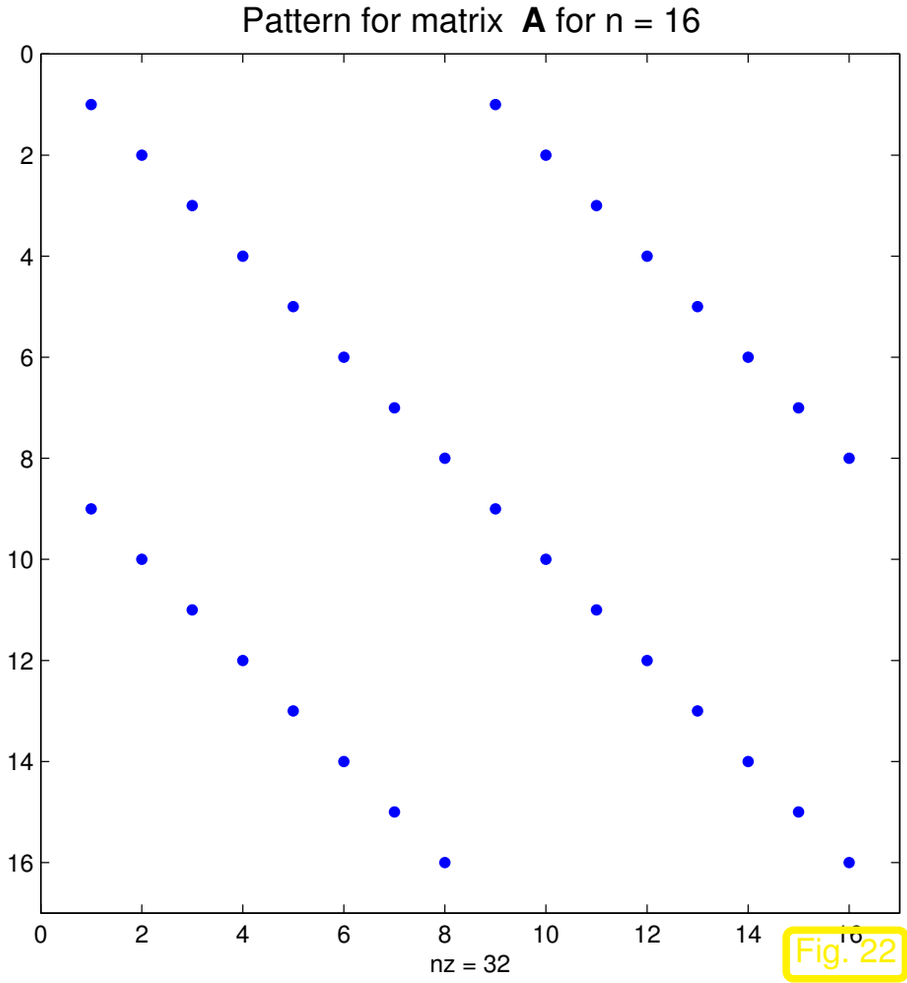


Fig. 22

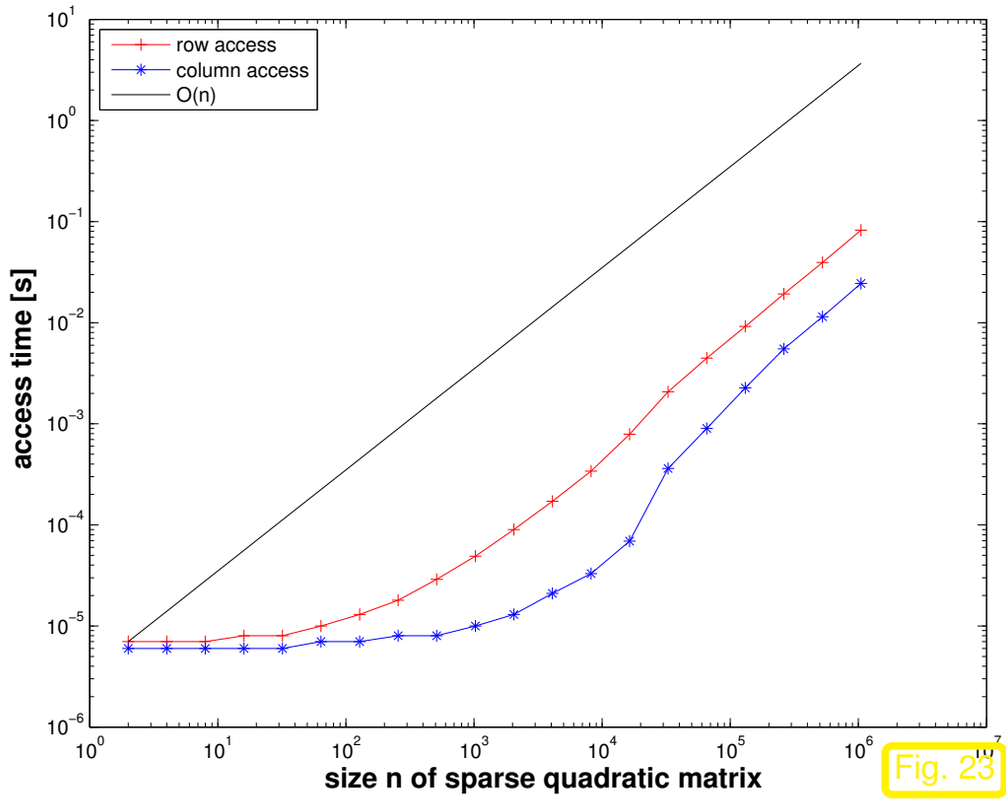


Fig. 23

Code 2.6.6: timing access to rows/columns of a sparse matrix

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

R. Hiptmair

rev 30401, January 28, 2011

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


R. Hiptmair

rev 30401, January 28, 2011

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

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

◇

Example 2.6.7 (Efficient Initialization of sparse matrices in MATLAB).

Code 2.6.8: 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 2.6.9: Initialization of sparse matrices: version II

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


R. Hiptmair

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

Code 2.6.10: Initialization of sparse matrices: version III

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

Code 2.6.11: Initialization of sparse matrices: driver script

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


R. Hiptmair

rev 30401, January 28, 2011

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

Timings: ▷

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

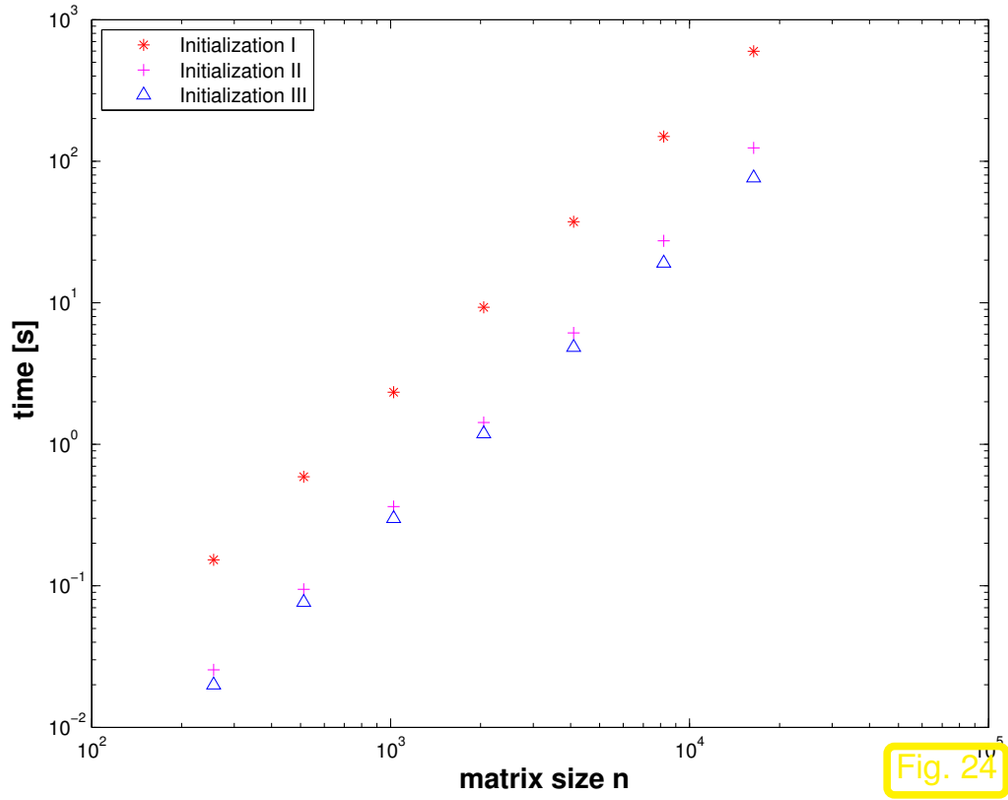


Fig. 24

☛ It is grossly inefficient to initialize a matrix in CCS format (→ Ex. 2.6.4) by setting individual entries one after another, because this usually entails moving large chunks of memory to create space for new non-zero entries.

Instead calls like

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

R. Hiptmair

rev 30401, January 28, 2011

where

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

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

◇

Example 2.6.12 (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);
```


R. Hiptmair

rev 30401, January 28, 2011

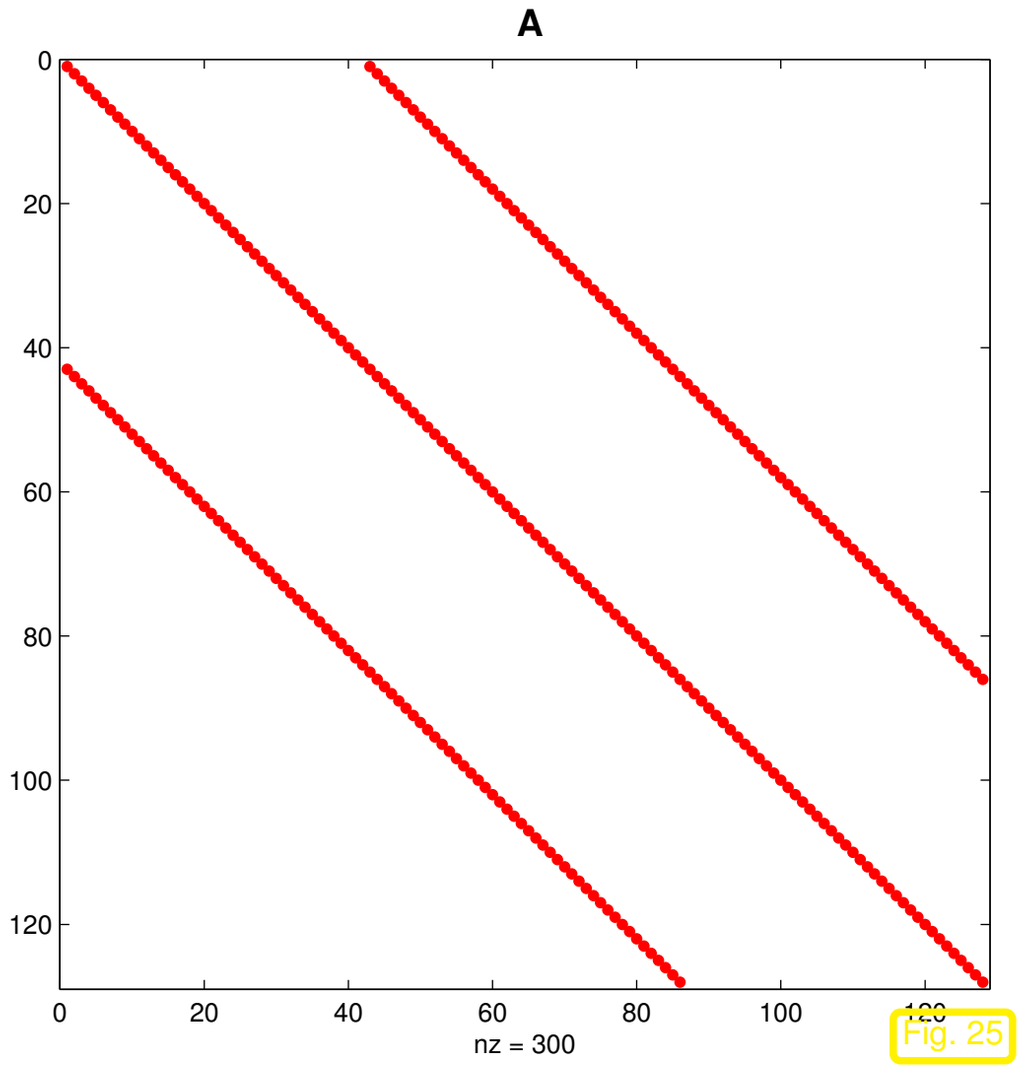


Fig. 25

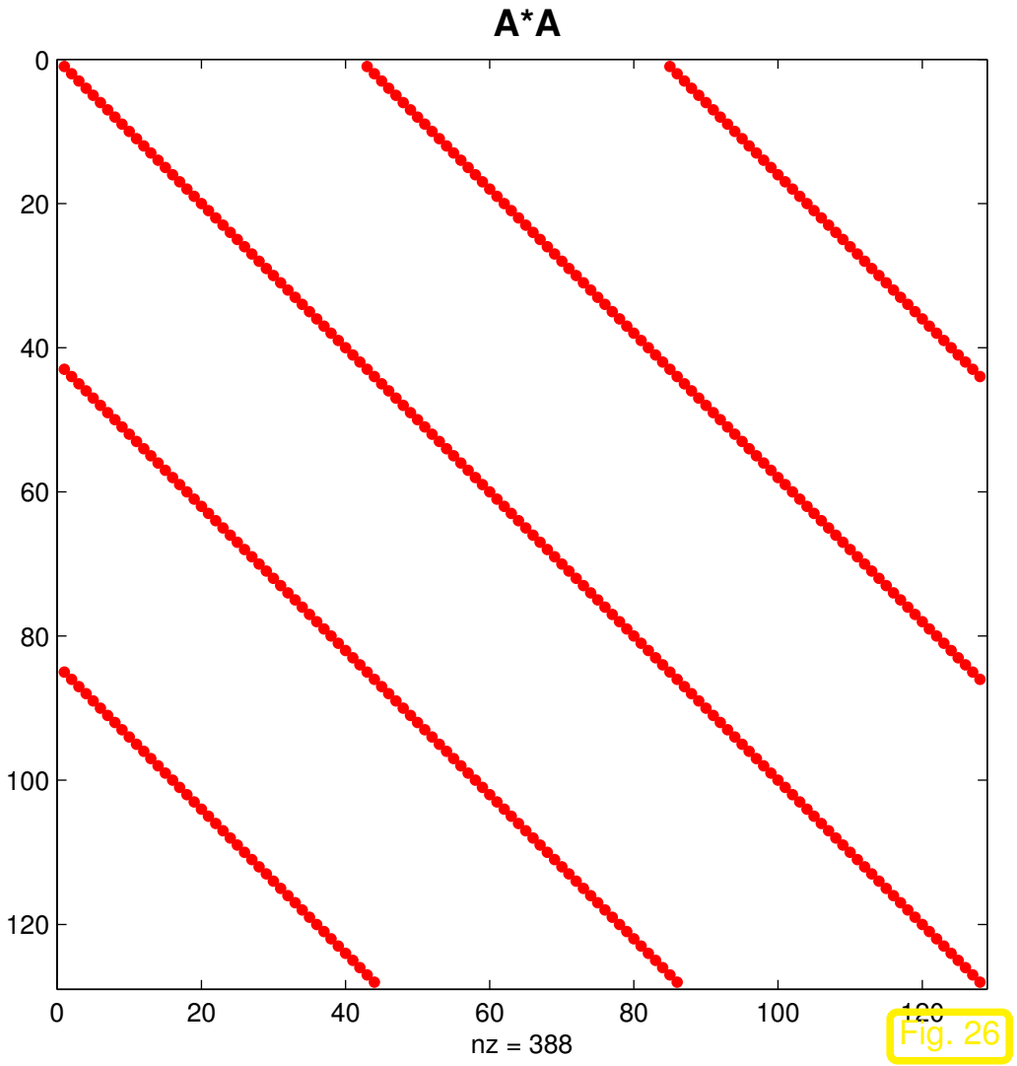


Fig. 26

➢ $\mathbf{A}^2$ 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. 2.6.7)

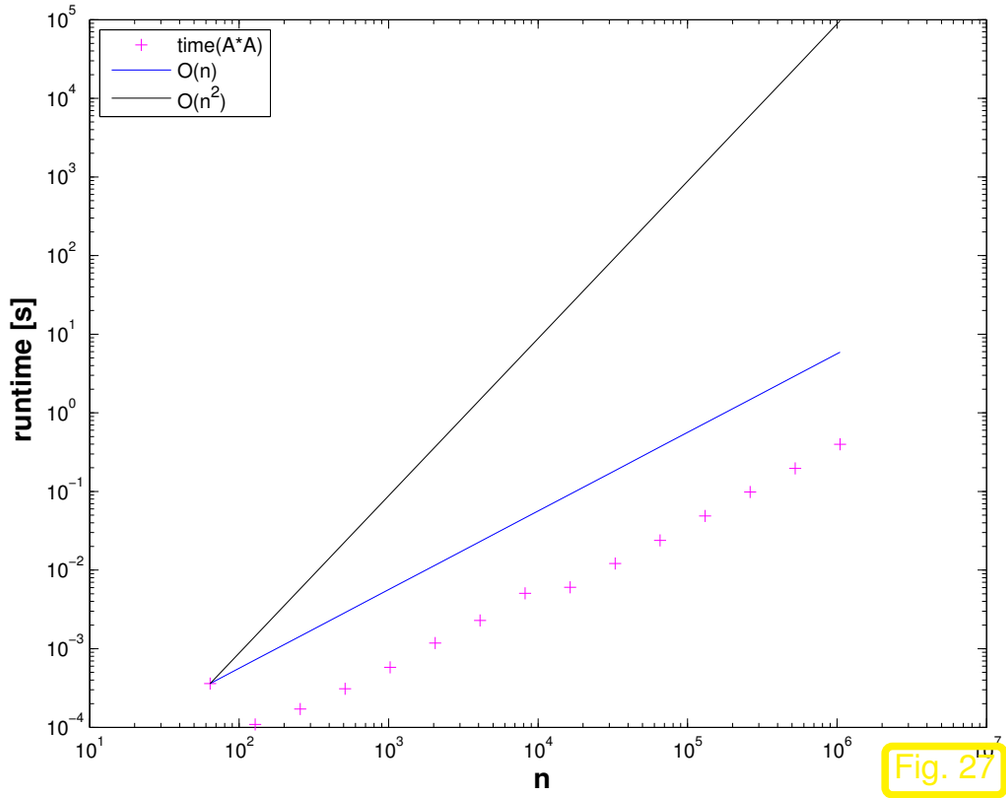


Fig. 27

R. Hiptmair

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

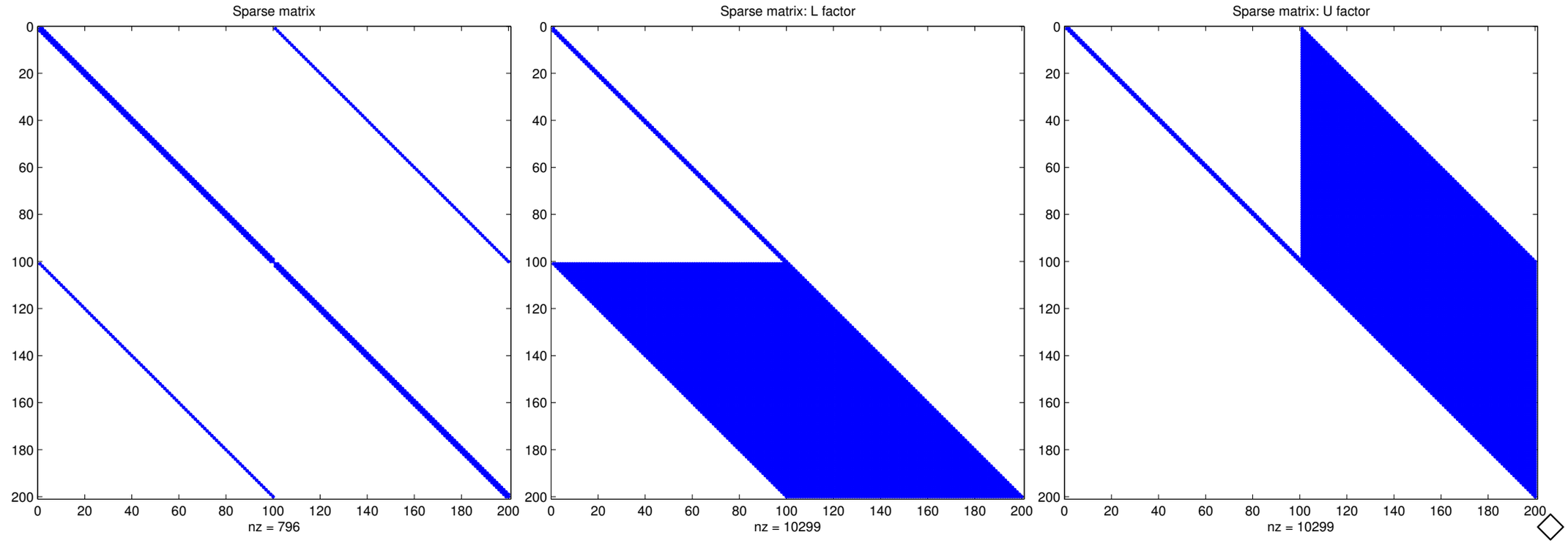
2.6.3 LU-factorization of sparse matrices

Example 2.6.13 (LU-factorization of sparse matrices).

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

Code 2.6.14: 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';
```



$\mathbf{A}$ sparse $\not\Rightarrow$ LU-factors sparse

Definition 2.6.15 (Fill-in)**.**

Let $\mathbf{A} = \mathbf{LU}$ be an LU-factorization (→ 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).

R. Hiptmair

rev 30401, January 28, 2011

Example 2.6.16 (Sparse LU-factors).

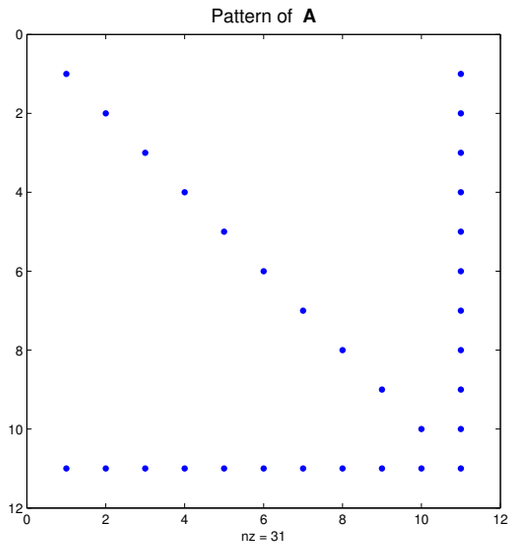
Ex. 2.6.13 ➢ massive fill-in can occur for sparse matrices

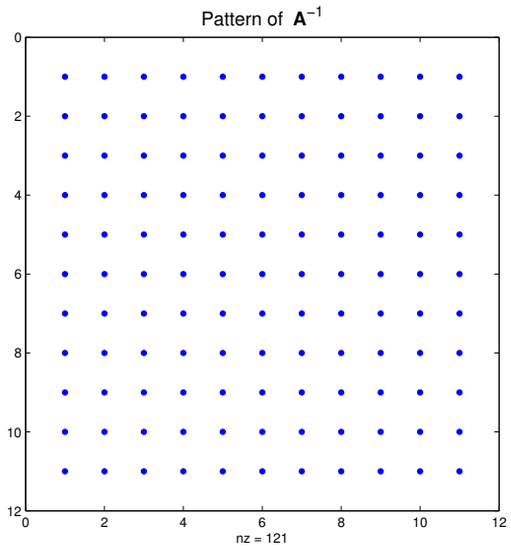
This example demonstrates that fill-in can be largely avoided, if the matrix has favorable structure. In this case a LSE with this particular system matrix $\mathbf{A}$ can be solved efficiently, that is, with a computational effort $O(\mathrm{nnz}(\mathbf{A}))$ by Gaussian elimination.

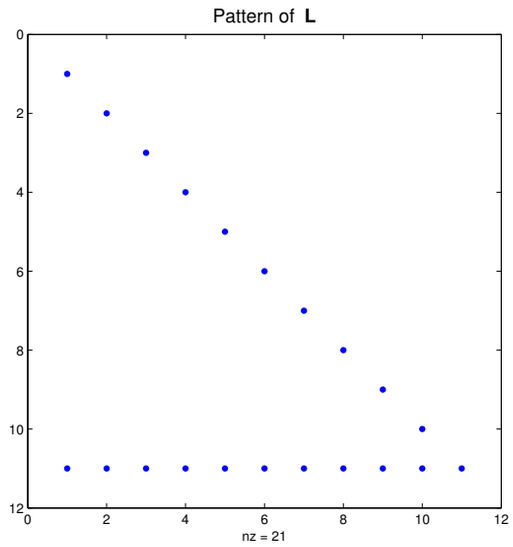
```
A = [diag(1:10),ones(10,1);ones(1,10),2];
[L,U] = lu(A); spy(A); spy(L); spy(U); spy(inv(A));
```

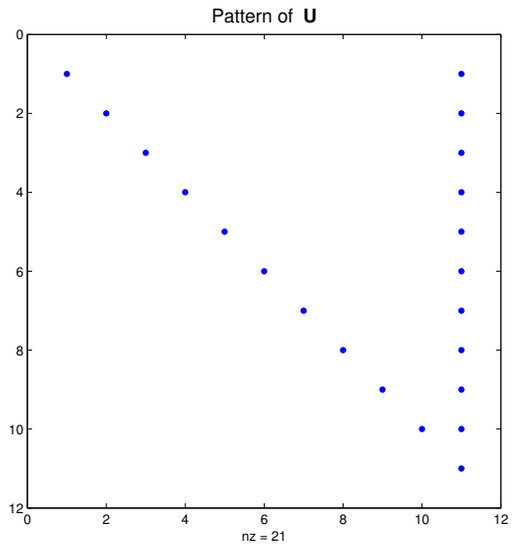

$\mathbf{A}$ is called an “arrow matrix”, see the pattern of non-zero entries below.

Recalling Rem. 2.2.13 it is easy to see that the LU-factors of $\mathbf{A}$ will be sparse and that their sparsity patterns will be as depicted below.









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

Besides stability issues, see Ex. 2.5.17, this is another reason why using `x = inv(A)*y` instead of `y = A\b` is usually a major blunder.

$\diamond$

Example 2.6.17 (“arrow matrix”).

$$\mathbf{A} = \left(\begin{array}{c|c} \alpha & \mathbf{b}^T \\ \hline \mathbf{c} & \mathbf{D} \end{array}\right) , \quad \begin{array}{l} \alpha \in \mathbb{R} , \\ \mathbf{b}, \mathbf{c} \in \mathbb{R}^{n-1} , \\ \mathbf{D} \in \mathbb{R}^{n-1,n-1} \text{ regular diagonal matrix, } \rightarrow \text{ Def. 2.2.2} \end{array} \tag{2.6.18}$$

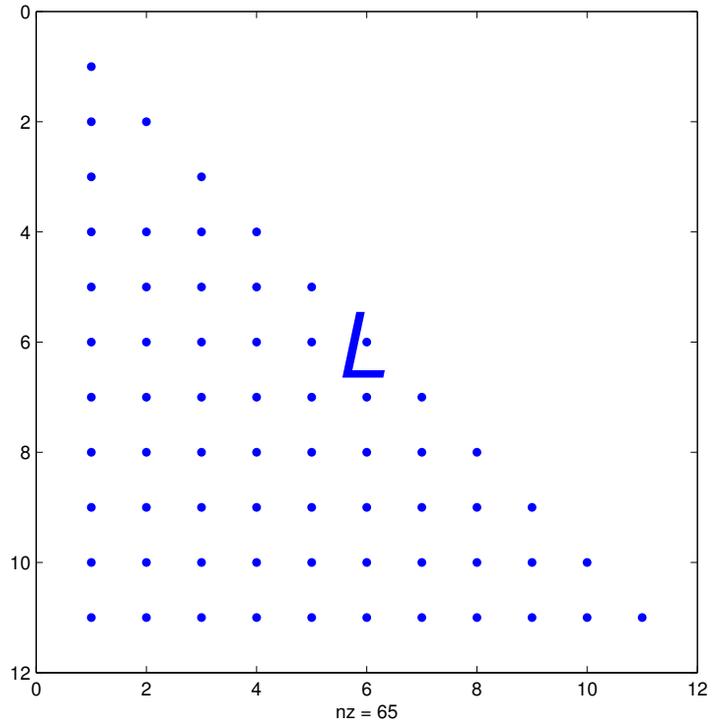
Run algorithm 2.3.5 (Gaussian elimination without pivoting):

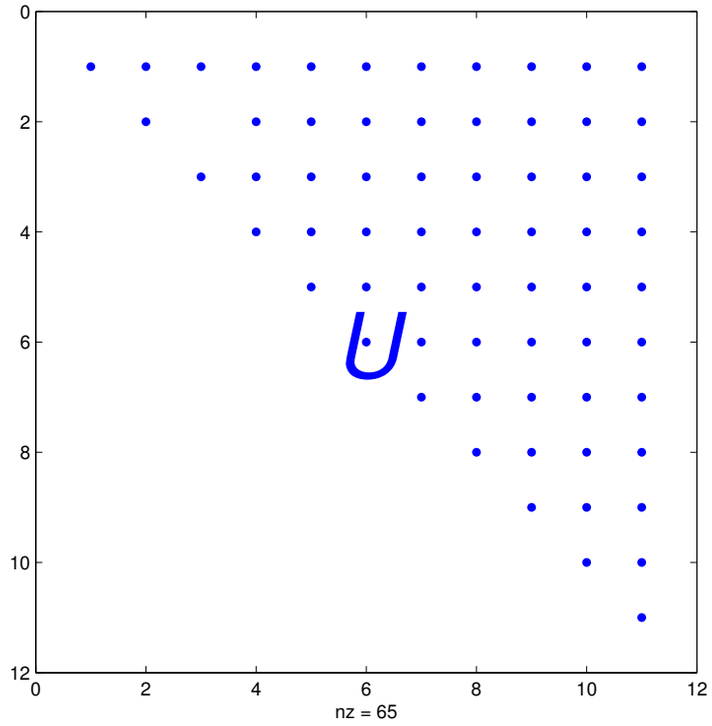
- factor matrices with $O(n^2)$ non-zero entries.
- computational costs: $O(n^3)$

Code 2.6.19: LU-factorization of arrow matrix

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

Obvious fill-in ($\rightarrow$ Def. 2.6.15)





Cyclic permutation of rows/columns:

- 1st row/column $\rightarrow$ n-th row/column
- i-th row/column $\rightarrow$ $i-1$-th row/column, $i = 2, \ldots, n$

$$\mathbf{A} = \left(\begin{array}{c|c} \mathbf{D} & \mathbf{c} \\ \hline \mathbf{b}^T & \alpha \end{array} \right) \tag{2.6.20}$$

$\succ$ LU-factorization requires $O(n)$ operations, see Ex. 2.6.16.

R. Hiptmair

rev 30401, January 28, 2011

After permuting rows of $\mathbf{A}$ from (2.6.20) , *cf.* (2.2.15):

$$\mathbf{L} = \left(\begin{array}{c|c} \mathbf{I} & 0 \\ \hline \mathbf{b}^T\mathbf{D}^{-1} & 1 \end{array}\right) \quad , \quad \mathbf{U} = \left(\begin{array}{c|c} \mathbf{D} & \mathbf{c} \\ \hline 0 & \sigma \end{array}\right) \quad , \quad \sigma := \alpha - \mathbf{b}^T\mathbf{D}^{-1}\mathbf{c} \, .$$

➢ No more fill-in, costs merely $O(n)$ **!**

Solving LSE $\mathbf{Ax} = \mathbf{y}$ with $\mathbf{A}$ from 2.6.18: two MATLAB codes

R. Hiptmair

rev 30401, January 28, 2011

“naive” implementation via “\”:

Code 2.6.22: LSE with arrow matrix, implementation I

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

“structure aware” implementation:

Code 2.6.24: LSE with arrow matrix, implementation II

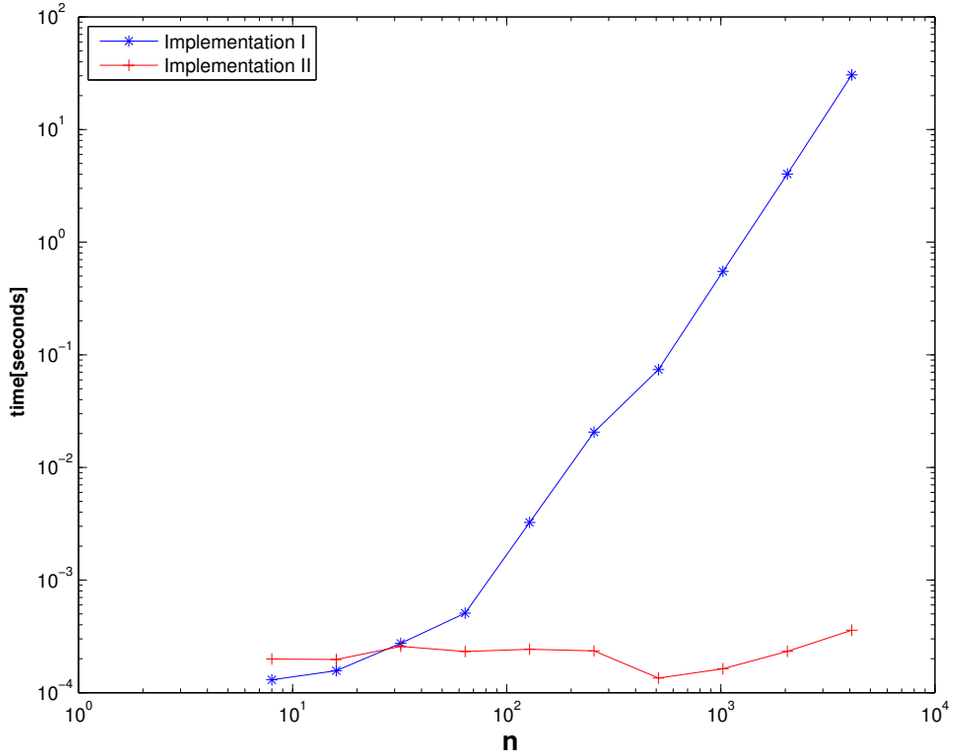
```
1 function x =
    sa2(alpha,b,c,d,y)
2 z = b./d;
3 xi = (y(1) -
    dot(z,y(2:end)))...
4         /(alpha-dot(z,c));
5 x = [xi; (y(2:end)-xi*c)./d];
```


R. Hiptmair

rev 30401, January 28, 2011

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. 2.6.7

MATLAB can do much better !

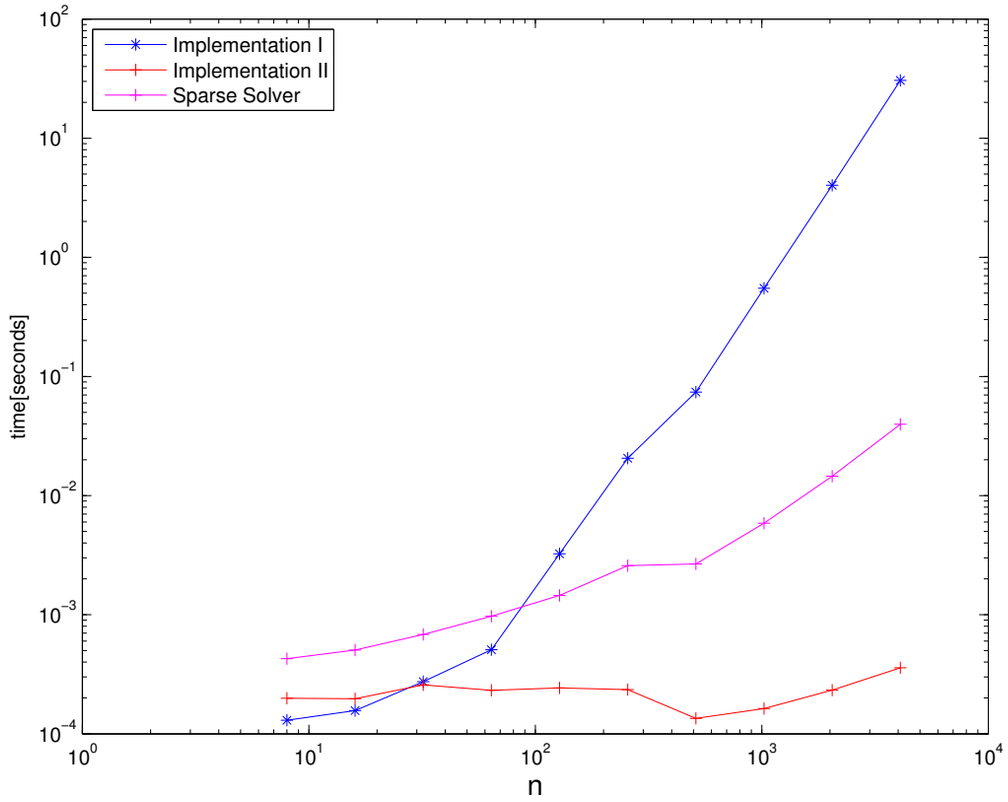
R. Hiptmair

rev 30401, January 28, 2011

Use sparse matrix format:

Code 2.6.25: sparse solver for arrow matrix

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



◇

Exploit structure of (sparse) linear systems of equations !

Caution: stability at risk

Example 2.6.26 (Pivoting destroys sparsity).

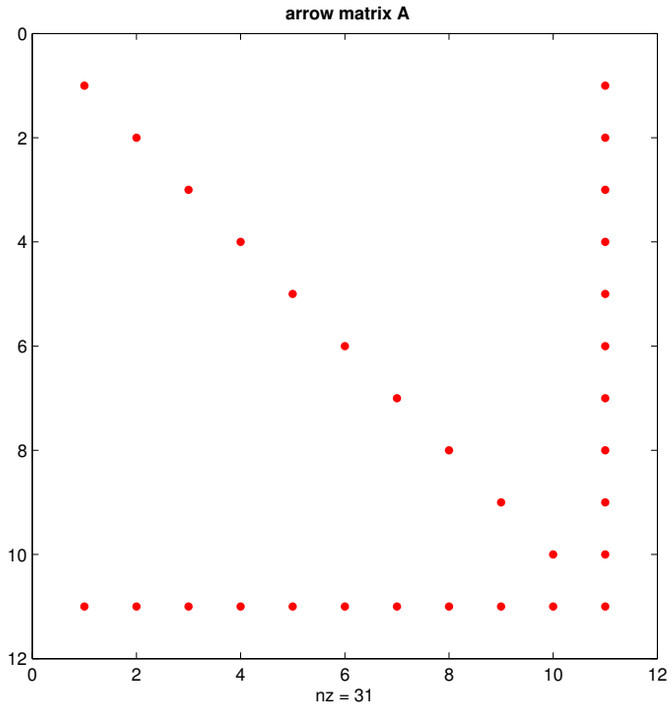
Code 2.6.27: fill-in due to pivoting

```
% 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';
```

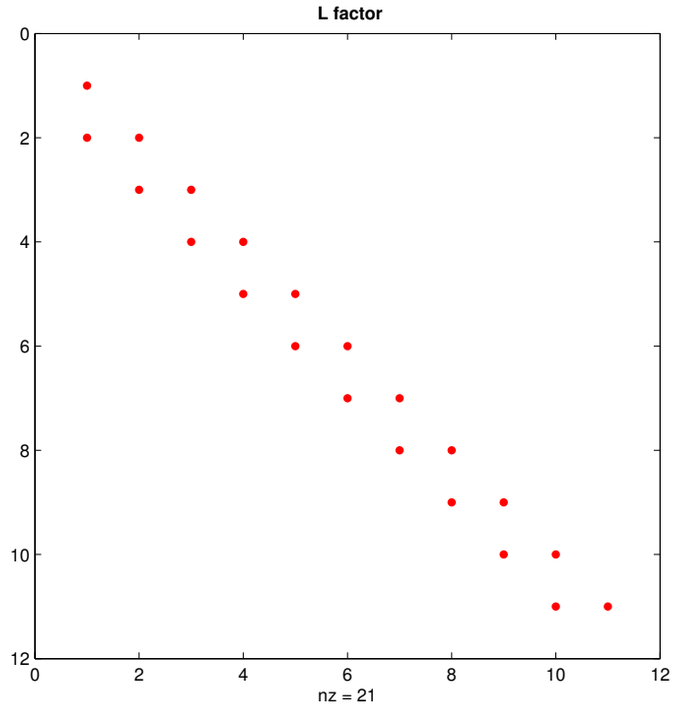
$$\mathbf{A} = \begin{pmatrix} 1 & & & 2 \\ & \frac{1}{2} & & 2 \\ & & \ddots & \vdots \\ & & \frac{1}{10} & 2 \\ 2 & \dots & & 2 \end{pmatrix} \quad \rightarrow \quad \text{arrow matrix, Ex. 2.6.16}$$

R. Hiptmair

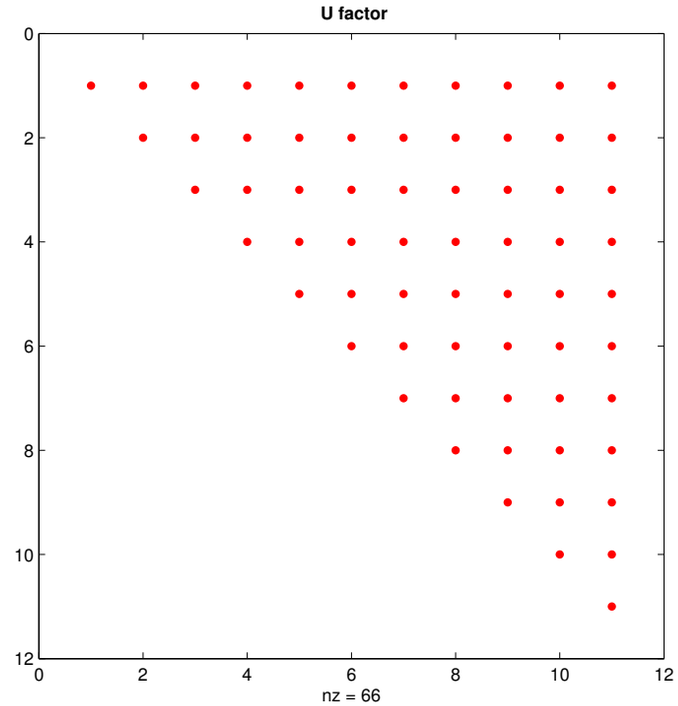
rev 30401, January 28, 2011



$\mathbf{A}$



$\mathbf{L}$



$\mathbf{U}$

In this case the solution of a LSE with system matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ of the above type by means of Gaussian elimination with partial pivoting would incur costs of $O(n^3)$.

$\diamond$

2.6.4 Banded matrices

☛ a special class of sparse matrices with extra structure:

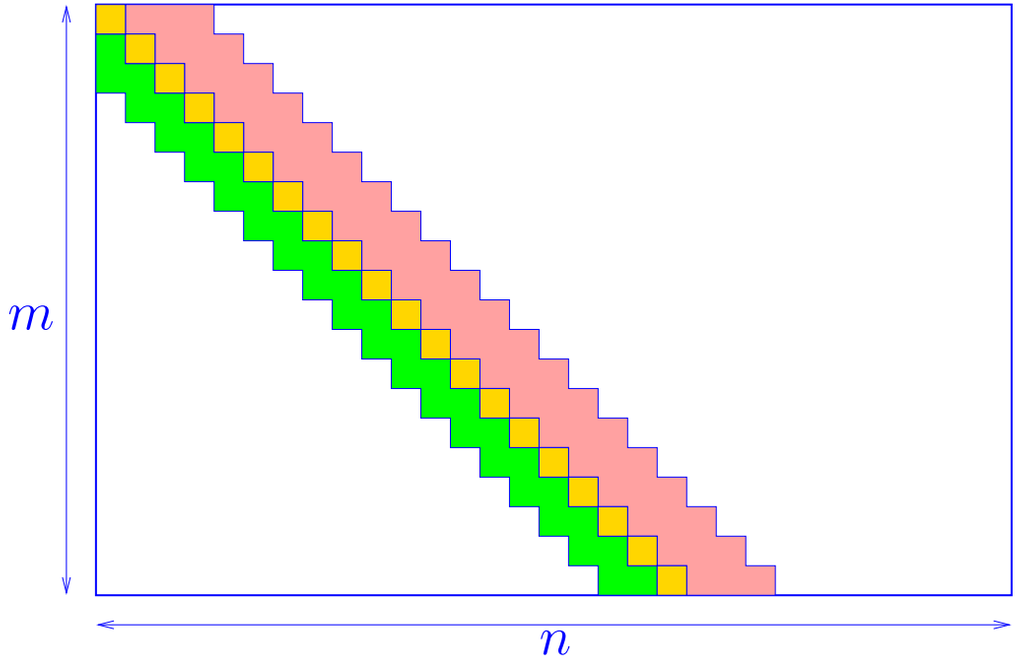
Definition 2.6.28 (bandwidth)**.**
For $\mathbf{A} = (a_{ij})_{i,j} \in \mathbb{K}^{m,n}$ *we call*

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

$m(\mathbf{A}) := \overline{m}(\mathbf{A}) + \underline{m}(\mathbf{A}) + 1$ **=** *bandwidth von* $\mathbf{A}$ *(*ger.: *Bandbreite)*

- $m(\mathbf{A}) = 1 \quad \triangleright \quad \mathbf{A}$ diagonal matrix, → Def. 2.2.2
- $\overline{m}(\mathbf{A}) = \underline{m}(\mathbf{A}) = 1 \quad \triangleright \quad \mathbf{A}$ tridiagonal matrix
- More general: $\mathbf{A} \in \mathbb{R}^{n,n}$ with $m(\mathbf{A}) \ll n \mathrel{\hat{=}}$ banded matrix

R. Hiptmair
rev 30401, January 28, 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 !)

R. Hiptmair

rev 30401, January 28, 2011

We now examine a generalization of the concept of a banded matrix that is particularly useful in the context of Gaussian elimination:

Definition 2.6.29 (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
$$\mathrm{env}(\mathbf{A}) := \left\{ (i,j) \in \{1,\ldots,n\}^2 : \begin{array}{c} i - m_i^R(\mathbf{A}) \leq j \leq i \ , \\ j - m_j^C(\mathbf{A}) \leq i \leq j \end{array} \right\}$$

Example 2.6.30 (Envelope of a matrix)*.*

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

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

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

R. Hiptmair

rev 30401, January 28, 2011

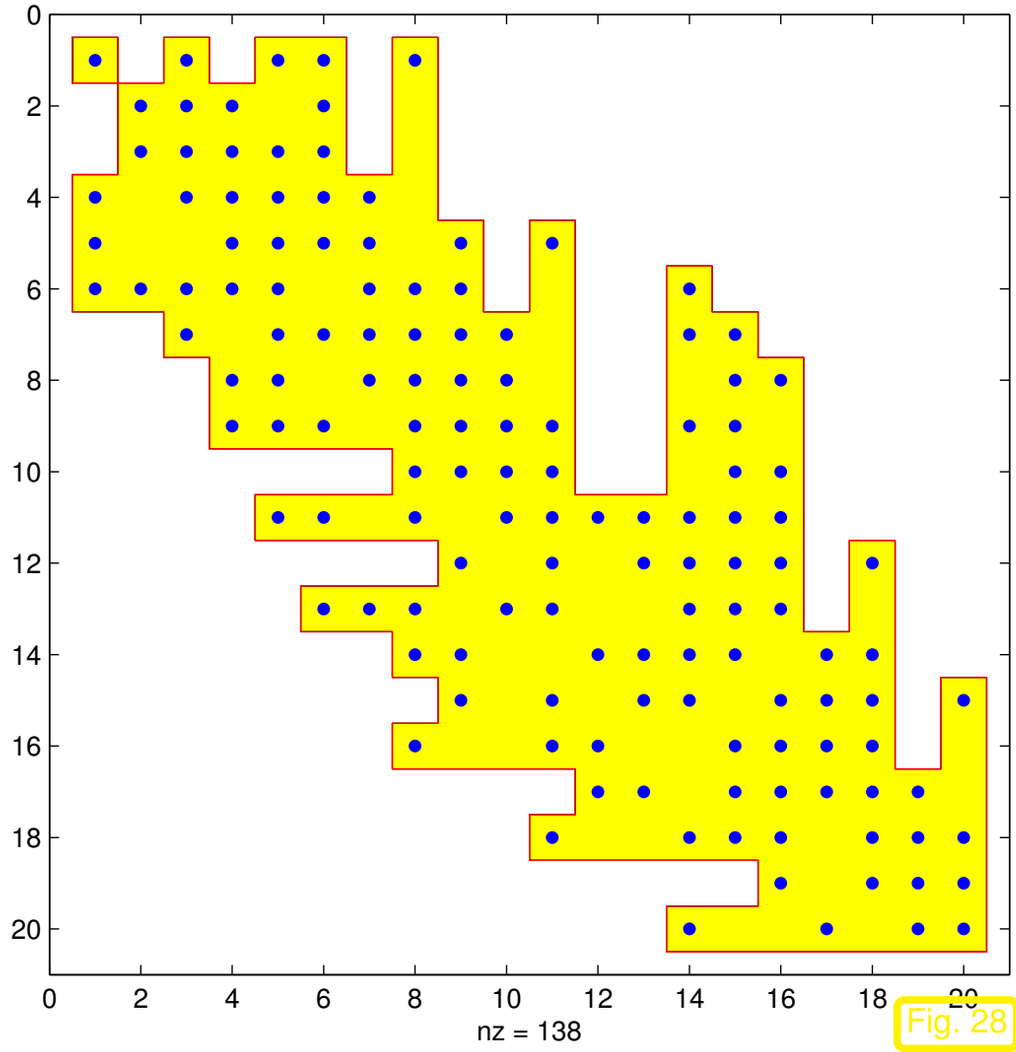


Fig. 28

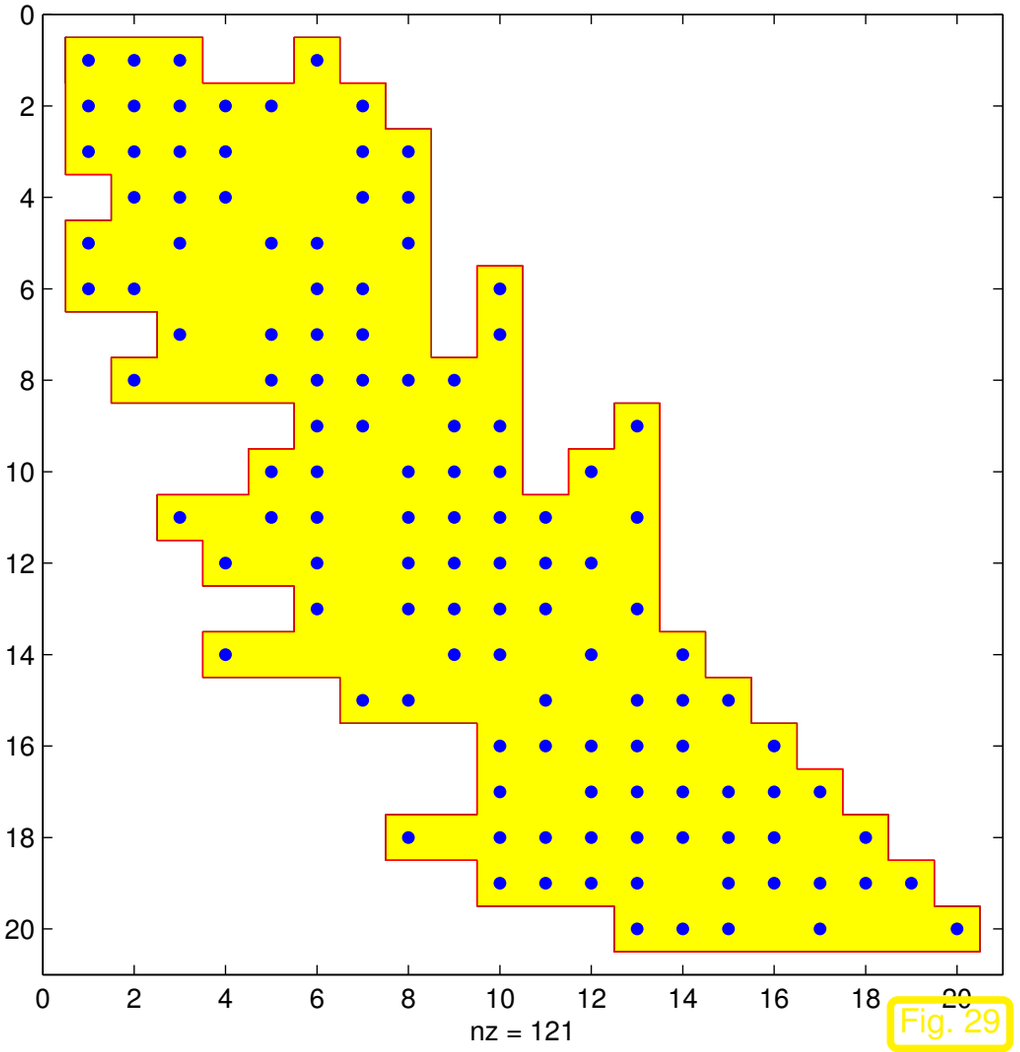


Fig. 29

Note: the envelope of the arrow matrix from Ex. 2.6.16 is just the set of index pairs of its non-zero entries. Hence, the following theorem provides another reason for the sparsity of the LU-factors in that example.

Theorem 2.6.31 (Envelope and fill-in)**.**
If $\mathbf{A} \in \mathbb{K}^{n,n}$ *is regular with* LU*-decomposition* $\mathbf{A} = \mathbf{LU}$*, then fill-in (→ Def. 2.6.15) is confined to* $\operatorname{env}(\mathbf{A})$*.*

Proof. (by induction, version I) Examine first step of Gaussian elimination without pivoting, $a_{11} \neq 0$

$$\mathbf{A} = \begin{pmatrix} a_{11} & \mathbf{b}^T \\ \mathbf{c} & \tilde{\mathbf{A}} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 \\ -\frac{\mathbf{c}}{a_{11}} & \mathbf{I} \end{pmatrix}}_{\mathbf{L}^{(1)}} \underbrace{\begin{pmatrix} a_{11} & \mathbf{b}^T \\ 0 & \tilde{\mathbf{A}} - \frac{\mathbf{c}\mathbf{b}^T}{a_{11}} \end{pmatrix}}_{\mathbf{U}^{(1)}}$$

$$\text{If } (i,j) \notin \operatorname{env}(\mathbf{A}) \quad \Rightarrow \quad \begin{cases} c_{i-1} = 0 & \text{, if } i > j \text{ ,} \\ b_{j-1} = 0 & \text{, if } i < j \text{ .} \end{cases}$$

$$\Rightarrow \quad \operatorname{env}(\mathbf{L}^{(1)}) \subset \operatorname{env}(\mathbf{A}), \quad \operatorname{env}(\mathbf{U}^{(1)}) \subset \operatorname{env}(\mathbf{A}) \text{ .}$$

Moreover, $\operatorname{env}(\tilde{\mathbf{A}} - \frac{\mathbf{c}\mathbf{b}^T}{a_{11}}) = \operatorname{env}(\mathbf{A}(2:n, 2:n))$ □

Proof. (by induction, version II) Use block-LU-factorization, *cf.* Rem. 2.2.14 and proof of Lemma 2.2.4:

$$\left(\begin{array}{c|c} \widetilde{\mathbf{A}} & \mathbf{b} \\ \hline \mathbf{c}^T & \alpha \end{array}\right) = \left(\begin{array}{c|c} \widetilde{\mathbf{L}} & 0 \\ \hline \mathbf{l}^T & 1 \end{array}\right) \left(\begin{array}{c|c} \widetilde{\mathbf{U}} & \mathbf{u} \\ \hline 0 & \xi \end{array}\right) \quad \Rightarrow \quad \begin{array}{l} \widetilde{\mathbf{U}}^T \mathbf{l} = \mathbf{c} \,, \\ \widetilde{\mathbf{L}}\mathbf{u} = \mathbf{b} \,. \end{array} \tag{2.6.32}$$

From Def. 2.6.29:

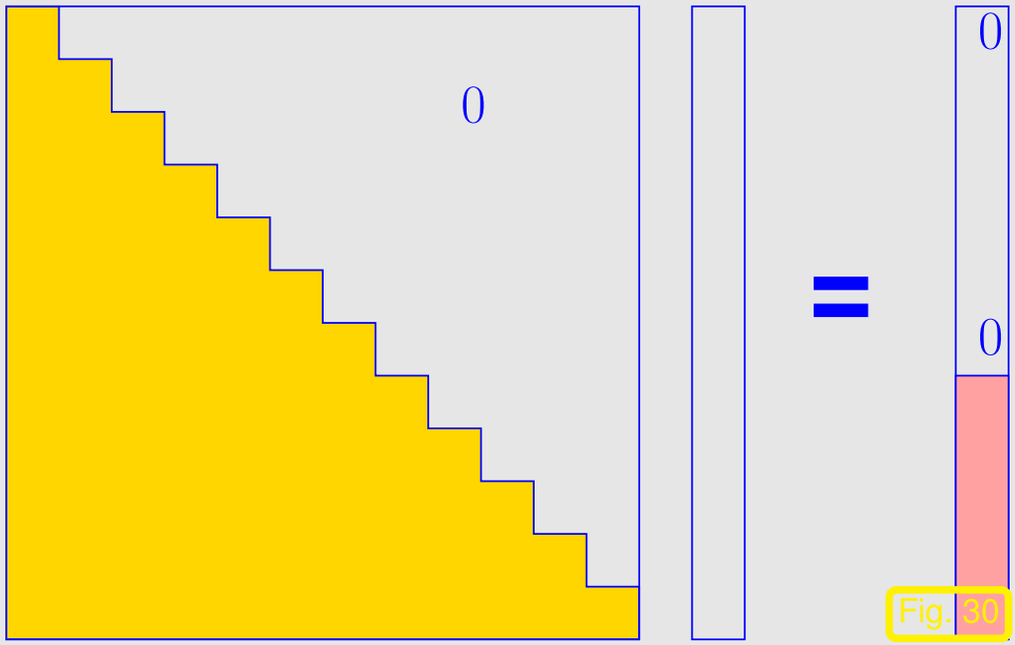


Fig. 30

If $m_n^R(\mathbf{A}) = m$, then $c_1, \ldots, c_{n-m} = 0$ (entries of $\mathbf{c}$ from (2.6.32))

If $m_n^C(\mathbf{A}) = m$, then $b_1, \ldots, b_{n-m} = 0$ (entries of $\mathbf{b}$ from (2.6.32))

$\lhd$ for lower triagular LSE:

If $c_1, \ldots, c_k = 0$ then $l_1, \ldots, l_k = 0$

If $b_1, \ldots, b_k = 0$, then $u_1, \ldots, u_k = 0$

$\Downarrow$

assertion of the theorem $\square$

R. Hiptmair

rev 30401, January 28, 2011

Thm. 2.6.31 immediately suggests a policy for saving cmputational effort when solving linear system whose system matrix $\mathbf{A} \in \mathbb{K}^{n,n}$ is sparse due to *small envelope*:

$$\sharp \operatorname{env}(\mathbf{A}) \ll n^2 :$$

Policy: Confine elimination to envelope!

Details will be given now:

Envelope-aware LU-factorization:

Code 2.6.33: computing row bandwidths, → Def. 2.6.29

```
1 function mr = rowbandwidth(A)
2 % computes row bandwidth numbers m_i^R(A) of A
3 n = size(A,1); mr = zeros(n,1);
4 for i=1:n, mr(i) = max(0,i-min(find(A(i,:) ~= 0))); end
```

Code 2.6.34: envelope aware forward substitution

```
function y = substenv(L,y,mc)
% evelope aware forward substitution for Lx = y
% (L = lower triangular matrix)
% argument mc:  column bandwidth vector
n = size(L,1); y(1) = y(1)/L(1,1);
for i=2:n
  if (mr(i) > 0)
    zeta =
      L(i,i-mr(i):i-1)*y(i-mr(i):i-1);
    y(i) = (y(i) - zeta)/L(i,i);
  else y(i) = y(i)/L(i,i); end
end
```

Asymptotic complexity of envelope aware forward substitution, *cf.* Alg. 2.2.11, for $\mathbf{Lx} = \mathbf{y}$, $\mathbf{L} \in \mathbb{K}^{n,n}$ regular lower triangular matrix is

$$O(\#\operatorname{env}(\mathbf{L}))\ !$$

R. Hiptmair

rev 30401, January 28, 2011

By block LU-factorization $\rightarrow$ Rem. 2.2.14:

$$\left(\begin{array}{c|c} (\mathbf{A})_{1:n-1,1:n-1} & (\mathbf{A})_{1:n-1,n} \\ \hline (\mathbf{A})_{n,1:n-1} & (\mathbf{A})_{n,n} \end{array}\right) = \left(\begin{array}{c|c} \mathbf{L}_1 & 0 \\ \hline \mathbf{l}^T & 1 \end{array}\right) \left(\begin{array}{c|c} \mathbf{U}_1 & \mathbf{u} \\ \hline 0 & \gamma \end{array}\right) , \tag{2.6.35}$$

$$\Rightarrow \quad (\mathbf{A})_{1:n-1,1:n-1} = \mathbf{L}_1\mathbf{U}_1 , \quad \mathbf{L}_1\mathbf{u} = (\mathbf{A})_{1:n-1,n} , \quad \mathbf{U}_1^T\mathbf{l} = (\mathbf{A})_{n,1:n-1}^T , \quad \mathbf{l}^T\mathbf{u} + \gamma = (\mathbf{A})_{n,n} . \tag{2.6.36}$$

Code 2.6.37: envelope aware recursive LU-factorization

```
function [L,U] = luenv(A)
% envelope aware recursive LU-factorization
% of structurally symmetric matrix
n = size(A,1);
if (size(A,2) ~= n),
  error('A must be square'); end
if (n == 1), L = eye(1); U = A;
else
  mr = rowbandwidth(A);
  [L1,U1] = luenv(A(1:n-1,1:n-1));
  u = substenv(L1,A(1:n-1,n),mr);
  l = substenv(U1',A(n,1:n-1)',mr);
  if (mr(n) > 0)
    gamma = A(n,n) -
      l(n-mr(n):n-1)'*u(n-mr(n):n-1);
  else gamma = A(n,n); end
  L = [L1,zeros(n-1,1); l' , 1];
  U = [U1,u;zeros(1,n-1) , gamma];
end
```

◁ recursive implementation of envelope aware recursive LU-factorization (no pivoting !)

Assumption:

$$\mathbf{A} \in \mathbb{K}^{n,n} \text{ is}$$

structurally symmetric

Asymptotic complexity ($\mathbf{A} \in \mathbb{K}^{n,n}$)

$$O(n \cdot \#\mathrm{env}(\mathbf{A})) \ .$$

R. Hiptmair

rev 30401, January 28, 2011

Definition 2.6.38 (Structurally symmetric matrix).

$\mathbf{A} \in \mathbb{K}^{n,n}$ *is structurally symmetric, if*

$$(\mathbf{A})_{i,j} \neq 0 \quad \Leftrightarrow \quad (\mathbf{A})_{j,i} \neq 0 \quad \forall i,j \in \{1,\ldots,n\} \, .$$

- Store only a_{ij}, $(i,j) \in \mathrm{env}(\mathbf{A})$ when computing (in situ) LU-factorization of *structurally symmetric* $\mathbf{A} \in \mathbb{K}^{n,n}$
  - Storage required: $n + 2\sum_{i=1}^{n} m_i(\mathbf{A})$ floating point numbers
  - envelope oriented matrix storage

Example 2.6.39 (Envelope oriented matrix storage).

Linear envelope oriented matrix storage of *symmetric* $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$:

Two arrays:

`double * val` size P,
`unsigned int * dptr` size n

$$P := n + \sum_{i=1}^{n} m_i(A) \; .$$

(2.6.40)

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

Indexing rule:

$$\texttt{dptr}[j] = k \quad \Updownarrow \quad \texttt{val}[k] = a_{jj}$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`val`		a_{11}	a_{22}	a_{31}	a_{32}	a_{33}	a_{44}	a_{52}	a_{53}	a_{54}	a_{55}	a_{65}	a_{66}	a_{73}	a_{74}	a_{75}	a_{76}	a_{77}
`dptr`	0	1	2	5	6	10	12	17										

◇ R. Hiptmair

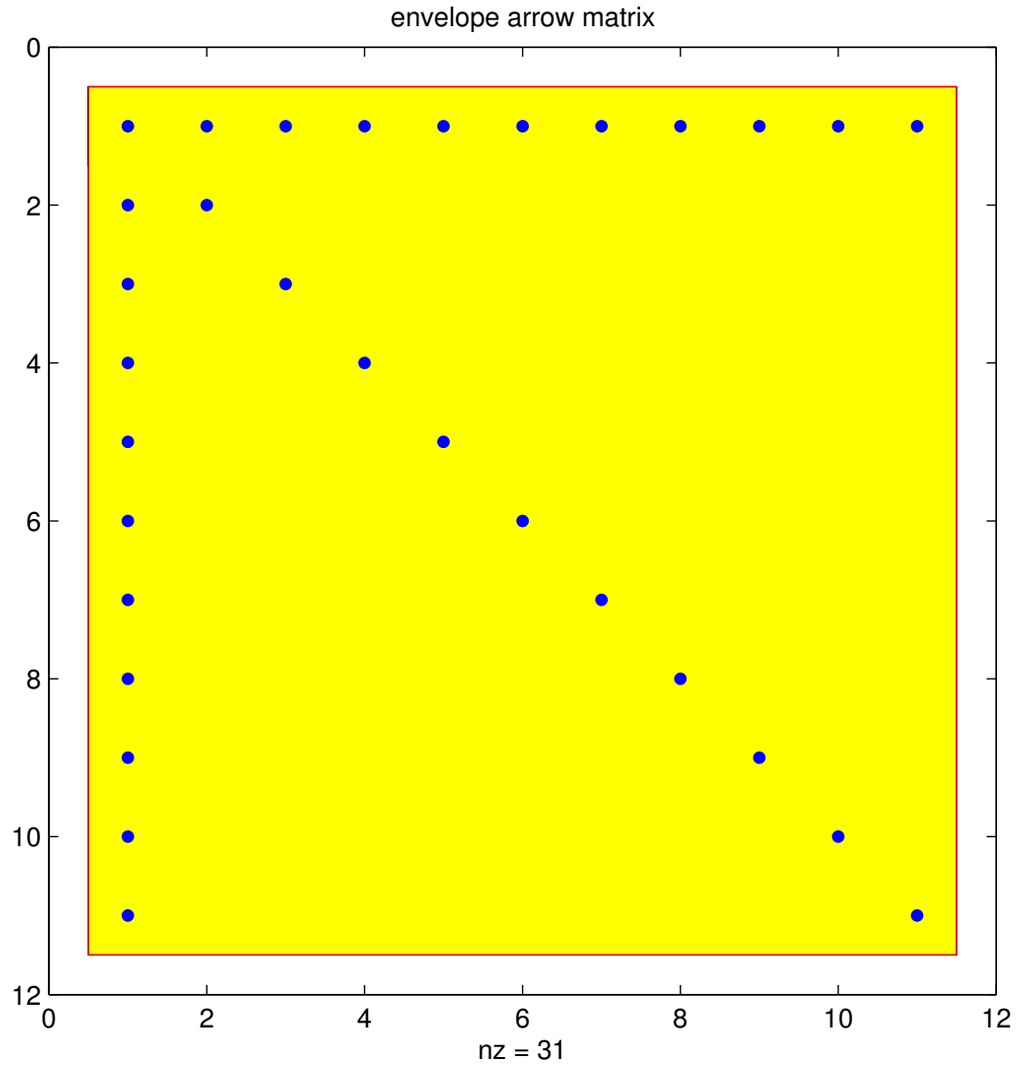
rev 30401, January 28, 2011

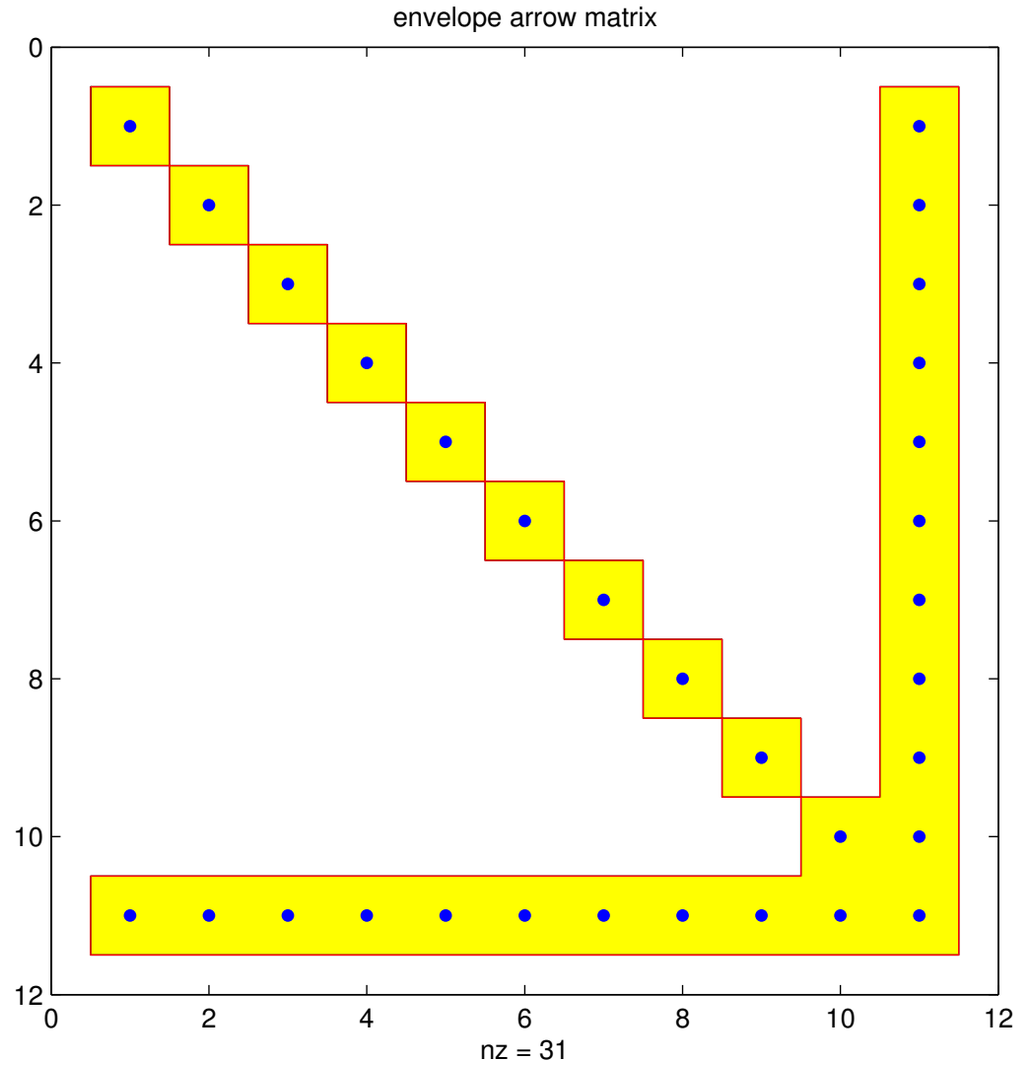
Minimizing bandwidth/envelope:

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

Example 2.6.41 (Reducing bandwidth by row/column permutations).

Recall: cyclic permutation of rows/columns of arrow matrix → Ex. 2.6.17





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 30401, January 28, 2011

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

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

◇

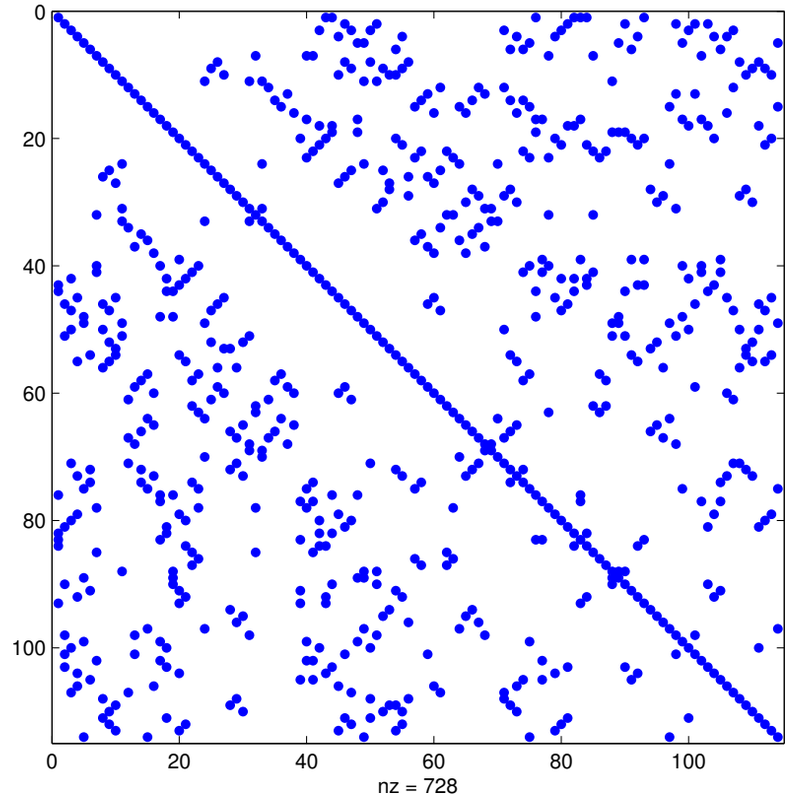
Example 2.6.42 (Reducing fill-in by reordering).

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

Code 2.6.43: preordering in MATLAB

```
1 spy(M);
2 [L,U] = lu(M); spy(U);
3 r = symrcm(M);
4 [L,U] = lu(M(r,r)); spy(U);
5 m = symamd(M);
6 [L,U] = lu(M(m,m)); spy(U);
```

Pattern of $\mathbf{M}$ ➥

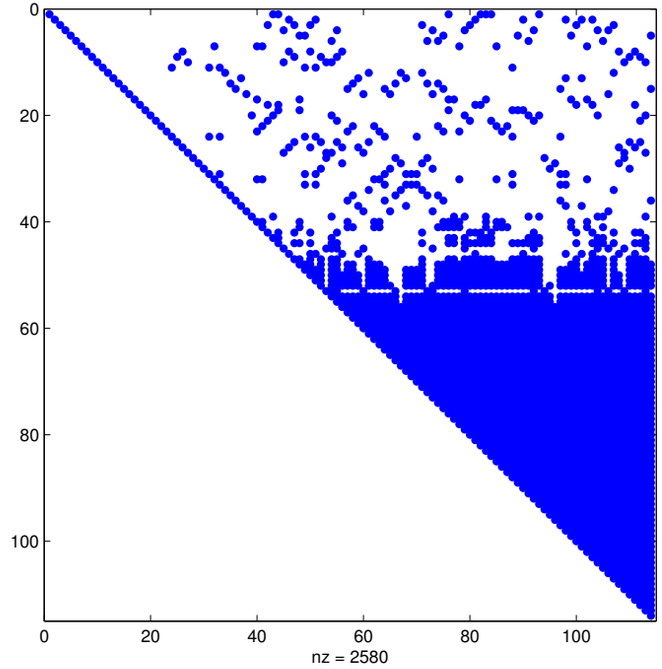


(Here: no row swaps from pivoting !)

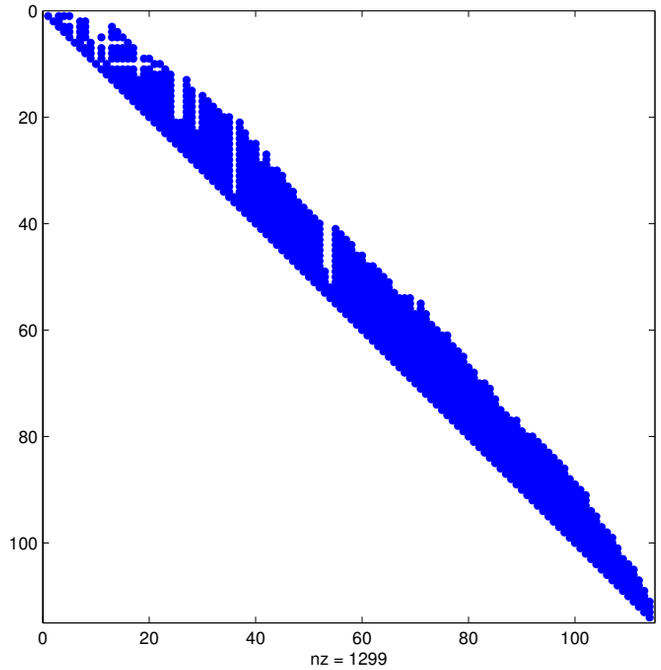
Examine patterns of LU-factors (→ Sect. 2.2) after reordering:

R. Hiptmair

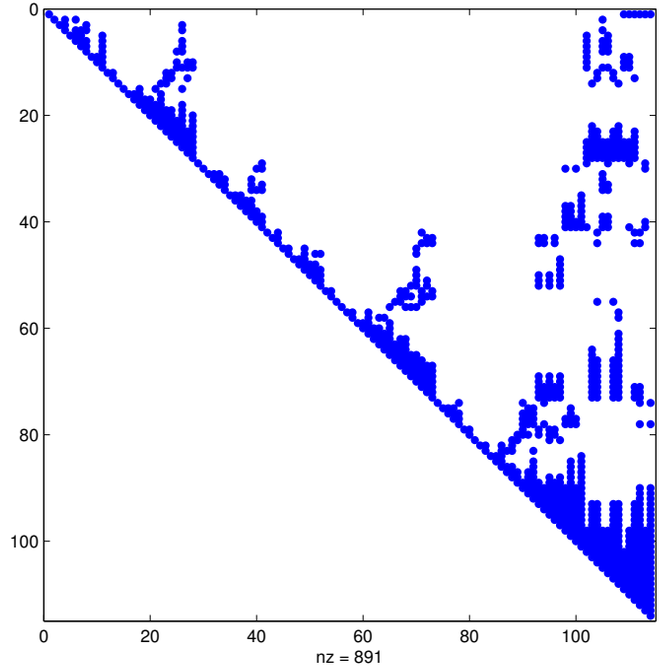
rev 30401, January 28, 2011



no reordering



reverse Cuthill-McKee



approximate minimum degree ◇

R. Hiptmair

rev 30401, October 7, 2010

Advice: Use numerical libraries for solving LSE with sparse system matrices **!**

→ SuperLU (`http://www.cs.berkeley.edu/~demmel/SuperLU.html`)

→ UMFPACK (`http://www.cise.ufl.edu/research/sparse/umfpack/`)

→ Pardiso (`http://www.pardiso-project.org/`)

→ Matlab-\ (on sparse storage formats)

2.7 Stable Gaussian elimination without pivoting

Thm. 2.6.31 ➢ special structure of the matrix helps avoid fill-in in Gaussian elimination/LU-factorization *without* pivoting.

Ex. 2.6.26 ➢ pivoting can trigger huge fill-in that would not occur without it.

Ex. 2.6.42 ➢ fill-in reducing effect of reordering can be thwarted by later row swapping in the course of pivoting.

Sect. 2.5.3: pivoting essential for stability of Gaussian elimination/LU-factorization

R. Hiptmair

rev 30401, October 7, 2010

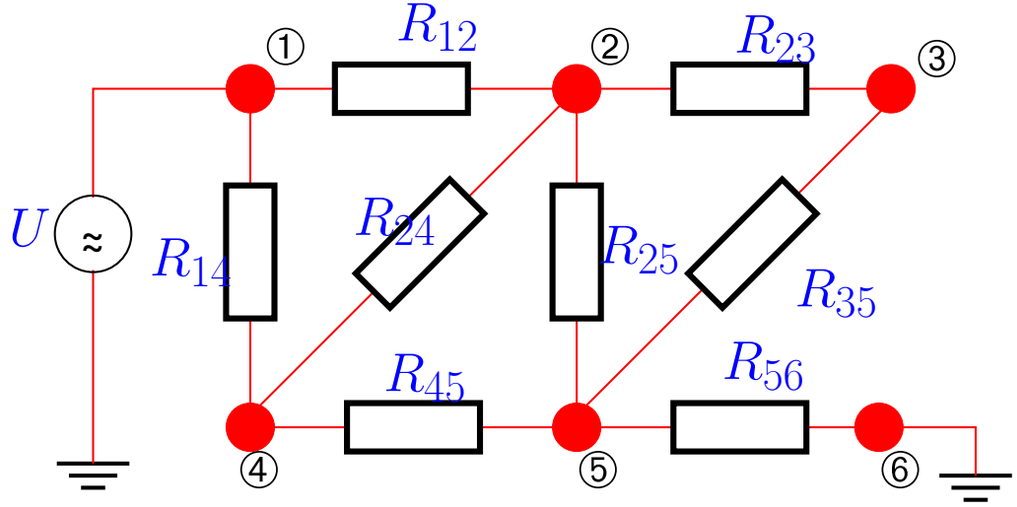
Very desirable: a priori criteria, when Gaussian elimination/LU-factorization remains stable even without pivoting. This can help avoid the extra work for partial pivoting and makes it possible to exploit structure without worrying about stability.

This section will introduce classes of matrices that allow Gaussian elimination without pivoting. Fortunately, linear systems of equations featuring system matrices from these classes are very common in applications.

Example 2.7.1 (Diagonally dominant matrices from nodal analysis). → Ex. 2.0.1

Consider:

electrical circuit entirely composed of Ohmic resistors.



Circuit equations from nodal analysis, see Ex. 2.0.1:

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

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

R. Hiptmair

rev 30401, October 7, 2010

➢ Matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ arising from nodal analysis satisfies

- $\mathbf{A} = \mathbf{A}^T \quad , \quad a_{kk} > 0 \quad , \quad a_{kj} \leq 0$ for $k \neq j$, (2.7.2)
- $\sum_{j=1}^{n} a_{kj} \geq 0$, $k = 1, \ldots, n$, (2.7.3)
- $\mathbf{A}$ is regular. (2.7.4)

All these properties are obvious except for the fact that $\mathbf{A}$ is regular.

Proof of (2.7.4): By Thm. 2.0.7 it suffices to show that the nullspace of $\mathbf{A}$ is trivial: $\mathbf{A}\mathbf{x} = 0 \quad \Rightarrow \quad \mathbf{x} = 0$.

Pick $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{A}\mathbf{x} = 0$, and $i \in \{1, \ldots, n\}$ so that

$$|x_i| = \max\{|x_j|,\ j = 1, \ldots, n\} \ .$$

$\mathbf{x}$

$$\mathbf{Ax} = 0 \quad \Rightarrow \quad x_i = \sum_{j \neq i} \frac{a_{ij}}{a_{ii}} x_j \quad \Rightarrow \quad |x_i| \leq \sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} |x_j| \; . \tag{2.7.5}$$

By (2.7.3) and the sign condition from (2.7.2) we conclude

$$\sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} \leq 1 \; . \tag{2.7.6}$$

Hence, (2.7.6) combined with the above estimate (2.7.5) that tells us that the maximum is smaller equal than a mean implies $|x_j| = |x_i|$ for all $j = 1, \ldots, n$. Finally, the sign condition $a_{kj} \leq 0$ for $k \neq j$ enforces the same sign of all x_i. Thus, we conclude, w.l.o.g., $x_1 = x_2 = \cdots = x_n$. As

$$\exists i \in \{1, \ldots, n\}: \quad \sum_{j=1}^{n} a_{ij} > 0 \quad \text{(strict inequality)} \; ,$$

$\mathbf{Ax} = 0$ is only possible for $\mathbf{x} = 0$.

$\diamond$

R. Hiptmair

rev 30401, October 7, 2010

Definition 2.7.7 (Diagonally dominant matrix)**.**

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

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

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

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

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

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

$(*)$: partial pivoting & diagonally dominant matrices $\succ$ no row permutations **!**

R. Hiptmair

rev 30401, October 7, 2010

Proof. (2.1.9) $\rightarrow$ induction w.r.t. n:

Clear: partial pivoting in the first step selects a_{11} as pivot element, *cf.* (2.3.9).

After 1st step of elimination:

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} \;, \quad i,j = 2,\ldots,n \quad \Rightarrow \quad a_{ii}^{(1)} > 0 \;.$$

$$\begin{aligned} |a_{ii}^{(1)}| - \sum_{\substack{j=2 \\ j\neq i}}^{n} |a_{ij}^{(1)}| &= \left| a_{ii} - \frac{a_{i1}}{a_{11}} a_{1i} \right| - \sum_{\substack{j=2 \\ j\neq i}}^{n} \left| a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} \right| \\ &\geq a_{ii} - \frac{|a_{i1}||a_{1i}|}{a_{11}} - \sum_{\substack{j=2 \\ j\neq i}}^{n} |a_{ij}| - \frac{|a_{i1}|}{a_{11}} \sum_{\substack{j=2 \\ j\neq i}}^{n} |a_{1j}| \\ &\geq a_{ii} - \frac{|a_{i1}||a_{1i}|}{a_{11}} - \sum_{\substack{j=2 \\ j\neq i}}^{n} |a_{ij}| - |a_{i1}| \frac{a_{11} - |a_{1i}|}{a_{11}} \geq a_{ii} - \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \geq 0 \;. \end{aligned}$$

$\mathbf{A}$ regular, diagonally dominant $\Rightarrow$ partial pivoting according to (2.3.9) selects i-th row in i-th step.

R. Hiptmair

rev 30401, October 7, 2010

There is another important class of matrices permitting stable Gaussian elimintation without pivoting:

Definition 2.7.9 (Symmetric positive definite (s.p.d.) matrices)**.**

$\mathbf{M} \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$, *is symmetric (Hermitian) positive definite (s.p.d.), if*

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

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

Lemma 2.7.10 (Necessary conditions for s.p.d.)**.**

For a symmetric/Hermitian positive definite matrix $\mathbf{M} = \mathbf{M}^H \in \mathbb{K}^{n,n}$ *holds true:*

1. $m_{ii} > 0$, $i = 1, \ldots, n$,
2. $m_{ii} m_{jj} - |m_{ij}|^2 > 0 \quad \forall 1 \leq i < j \leq n$,
3. *all eigenvalues of* $\mathbf{M}$ *are positive.* *(*$\leftarrow$ *also sufficient for symmetric/Hermitian* $\mathbf{M}$*)*

Remark 2.7.11 (S.p.d. Hessians).

Recall from analysis: in a local minimum x^* of a C^2-function $f : \mathbb{R}^n \mapsto \mathbb{R}$ ➤ Hessian $D^2 f(x^*)$ s.p.d.

To compute the minimum of a C^2-function iteratively by means of Newton's method ($\rightarrow$ Sect. 4.4) a linear system of equations with the s.p.d. Hessian as system matrix has to be solved in each step.

The solutions of many equations in science and engineering boils down to finding the minimum of some (energy, entropy, etc.) function, which accounts for the prominent role of s.p.d. linear systems in applications.

△

Lemma 2.7.12. *A diagonally dominant Hermitian/symmetric matrix with non-negative diagonal entries is positive* semi*-definite.*
A strictly *diagonally dominant Hermitian/symmetric matrix with positive diagonal entries is positive definite.*

Proof. For $\mathbf{A} = \mathbf{A}^H$ diagonally dominant, use inequality between arithmetic and geometric mean (AGM) $ab \leq \frac{1}{2}(a^2 + b^2)$:

$$\begin{aligned}
\mathbf{x}^H \mathbf{A}\mathbf{x} &= \sum_{i=1}^{n} a_{ii}|x_i|^2 + \sum_{i\neq j} a_{ij}\bar{x}_i x_j \geq \sum_{i=1}^{n} a_{ii}|x_i|^2 - \sum_{i\neq j} |a_{ij}||x_i||x_j| \\
&\geq \sum_{i=1}^{n} a_{ii}|x_i|^2 - \tfrac{1}{2}\sum_{i\neq j} |a_{ij}|(|x_i|^2 + |x_j|^2) \\
&\geq \tfrac{1}{2}\Big(\sum_{i=1}^{n}\{a_{ii}|x_i|^2 - \sum_{j\neq i} |a_{ij}||x_i|^2\}\Big) + \tfrac{1}{2}\Big(\sum_{j=1}^{n}\{a_{ii}|x_j|^2 - \sum_{i\neq j} |a_{ij}||x_j|^2\}\Big) \\
&\geq \sum_{i=1}^{n} |x_i|^2\Big(a_{ii} - \sum_{j\neq i} |a_{ij}|\Big) \geq 0 \; .
\end{aligned}$$

Theorem 2.7.13 (Gaussian elimination for s.p.d. matrices)**.**
Every symmetric/Hermitian positive definite matrix (→ Def. 2.7.9) possesses an LU-decomposition (→ Sect. 2.2).

Equivalent to assertion of theorem: Gaussian elimination feasible *without pivoting*

In fact, this theorem is a corollary of Lemma 2.2.4, because all principal minors of an s.p.d. matrix are s.p.d. themselves.

Sketch of alternative self-contained proof.

Proof by induction: consider first step of elimination

$$\mathbf{A} = \begin{pmatrix} a_{11} & \mathbf{b}^T \\ \mathbf{b} & \widetilde{\mathbf{A}} \end{pmatrix} \xrightarrow[\text{Gaussian elimination}]{\text{1. step}} \begin{pmatrix} a_{11} & \mathbf{b}^T \\ 0 & \widetilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^T}{a_{11}} \end{pmatrix} .$$

R. Hiptmair

rev 30401, October 7, 2010

➢ to show: $\widetilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^T}{a_{11}}$ s.p.d. ($\rightarrow$ step of induction argument)

- Evident: symmetry of $\widetilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^T}{a_{11}} \in \mathbb{R}^{n-1,n-1}$
- As $\mathbf{A}$ s.p.d. ($\rightarrow$ Def. 2.7.9), for every $\boldsymbol{y} \in \mathbb{R}^{n-1} \setminus \{0\}$

$$0 < \begin{pmatrix} -\frac{\mathbf{b}^T\mathbf{y}}{a_{11}} \\ \mathbf{y} \end{pmatrix}^T \begin{pmatrix} a_{11} & \mathbf{b}^T \\ \mathbf{b} & \widetilde{\mathbf{A}} \end{pmatrix} \begin{pmatrix} -\frac{\mathbf{b}^T\mathbf{y}}{a_{11}} \\ \mathbf{y} \end{pmatrix} = \boldsymbol{y}^T(\widetilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^T}{a_{11}})\boldsymbol{y} \; .$$

$\Rightarrow$ $\widetilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^T}{a_{11}}$ positive definite. □

The proof can also be based on the identities

$$\left(\begin{array}{c|c} (\mathbf{A})_{1:n-1,1:n-1} & (\mathbf{A})_{1:n-1,n} \\ \hline (\mathbf{A})_{n,1:n-1} & (\mathbf{A})_{n,n} \end{array}\right) = \left(\begin{array}{c|c} \mathbf{L}_1 & 0 \\ \hline \mathbf{l}^T & 1 \end{array}\right) \left(\begin{array}{c|c} \mathbf{U}_1 & \mathbf{u} \\ \hline 0 & \gamma \end{array}\right) \; , \tag{2.6.35}$$

$$\Rightarrow \quad (\mathbf{A})_{1:n-1,1:n-1} = \mathbf{L}_1\mathbf{U}_1 \; , \quad \mathbf{L}_1\mathbf{u} = (\mathbf{A})_{1:n-1,n} \; , \quad \mathbf{U}_1^T\mathbf{l} = (\mathbf{A})_{n,1:n-1}^T \; , \quad \mathbf{l}^T\mathbf{u} + \gamma = (\mathbf{A})_{n,n} \; ,$$

noticing that the principal minor $(\mathbf{A})_{1:n-1,1:n-1}$ is also s.p.d. This allows a simple induction argument.

Note: no pivoting required ($\rightarrow$ Sect. 2.3)

(partial pivoting always picks current pivot row)

R. Hiptmair

rev 30401, October 7, 2010

Lemma 2.7.14 (Cholesky decomposition for s.p.d. matrices). → *[23, Sect. 3.4], [29, Sect. II.5]*

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* $\mathbf{A} = \mathbf{R}^H \mathbf{R}$ *(Cholesky decomposition).*

Thm. 2.7.13 $\Rightarrow$ $\mathbf{A} = \mathbf{LU}$ (unique LU-decomposition of $\mathbf{A}$, Lemma 2.2.4)

$$\mathbf{A} = \mathbf{L}\mathbf{D}\widetilde{\mathbf{U}} \quad , \quad \begin{array}{l} \mathbf{D} \mathrel{\hat{=}} \text{diagonal of } \mathbf{U} \ , \\ \widetilde{\mathbf{U}} \mathrel{\hat{=}} \textit{normalized} \text{ upper triangular matrix} \rightarrow \text{Def. 2.2.2} \end{array}$$

Due to uniqueness of LU-decomposition

$$\mathbf{A} = \mathbf{A}^T \quad \Rightarrow \quad \mathbf{U} = \mathbf{D}\mathbf{L}^T \quad \Rightarrow \quad \boxed{\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T} \ ,$$

with unique $\mathbf{L}$, $\mathbf{D}$ (diagonal matrix)

$$\mathbf{x}^T \mathbf{A}\mathbf{x} > 0 \quad \forall \mathbf{x} \neq 0 \quad \Rightarrow \quad \mathbf{y}^T \mathbf{D}\mathbf{y} > 0 \quad \forall \mathbf{y} \neq 0 \ .$$

➤ D has positive diagonal ➡ $\mathbf{R} = \sqrt{\mathbf{D}}\mathbf{L}^T$. □

R. Hiptmair

rev 30401, October 7, 2010

In analogy to (2.2.5)

$$\mathbf{R}^H\mathbf{R} = \mathbf{A} \quad \Rightarrow \quad a_{ik} = \sum_{j=1}^{\min\{i,k\}} \overline{r}_{ji} r_{jk} = \begin{cases} \sum_{j=1}^{i-1} \overline{r}_{ji} r_{jk} + \overline{r}_{ii} r_{ik} & \text{, if } i < k \text{ ,} \\ \sum_{j=1}^{i-1} |r_{ji}|^2 + r_{ii}^2 & \text{, if } i = k \text{ .} \end{cases} \tag{2.7.15}$$

Code 2.7.16: simple Cholesky factorization

```
1  function R = cholfac(A)
2  % simple Cholesky factorization
3  n = size(A,1);
4  for k = 1:n
5    for j=k+1:n
6      A(j,j:n) = A(j,j:n) -
         A(k,j:n)*A(k,j)/A(k,k);
7    end
8    A(k,k:n) =
       A(k,k:n)/sqrt(A(k,k));
9  end
10 R = triu(A);
```

Computational costs (# elementary arithmetic operations) of Cholesky decomposition: $\frac{1}{6}n^3 + O(n^2)$

($\succ$ half the costs of LU-factorization, Code. 2.2.5)

MATLAB function: `R = chol(A)`

Solving LSE with s.p.d. system matrix via

Cholesky decomposition + forward & backward substitution

is numerically stable ($\rightarrow$ Def. 2.5.11)

Recall Thm. 2.5.13: Numerical instability of Gaussian elimination (with any kind of pivoting) manifests itself in massive growth of the entries of intermediate elimination matrices $\mathbf{A}^{(k)}$.

Use the relationship between LU-factorization and Cholesky decomposition, which tells us that we only have to monitor the growth of entries of intermediate upper triangular "Cholesky factorization matrices" $\mathbf{A} = (\mathbf{R}^{(k)})^H \mathbf{R}^{(k)}$.

Consider: Euclidean vector norm/matrix norm ($\rightarrow$ Def. 2.5.5) $\|\cdot\|_2$

$$\mathbf{A} = \mathbf{R}^H \mathbf{R} \quad \Rightarrow \quad \|\mathbf{A}\|_2 = \sup_{\|\mathbf{x}\|_2=1} \mathbf{x}^H \mathbf{R}^H \mathbf{R}\mathbf{x} = \sup_{\|\mathbf{x}\|_2=1} (\mathbf{R}\mathbf{x})^H (\mathbf{R}\mathbf{x}) = \|\mathbf{R}\|_2^2 \ .$$

➤ For all intermediate Cholesky factorization matrices holds: $\left\| (\mathbf{R}^{(k)})^H \right\|_2 = \left\| \mathbf{R}^{(k)} \right\|_2 = \|\mathbf{A}\|_2^{1/2}$

This rules out a blowup of entries of the $\mathbf{R}^{(k)}$.

R. Hiptmair

rev 30401, October 7, 2010

Remark 2.7.17 (Telling MATLAB about matrix properties).

MATLAB-\ assumes generic matrix, cannot detect special properties of (fully populated) matrix (e.g. symmetrc, s.p.d., triangular).

➤ Use `y = linsolve(A,b,opts)`

$$\texttt{opts} \in \left\{ \begin{array}{lll} \texttt{LT} & \leftrightarrow & \texttt{A} \text{ lower triangular matrix} \\ \texttt{UT} & \leftrightarrow & \texttt{A} \text{ upper triangular matrix} \\ \texttt{UHESS} & \leftrightarrow & \texttt{A} \text{ upper Hessenberg matrix} \\ \texttt{SYM} & \leftrightarrow & \texttt{A} \text{ Hermitian matrix} \\ \texttt{POSDEF} & \leftrightarrow & \texttt{A} \text{ positive definite matrix} \end{array} \right\}$$

△

2.8 QR-factorization/decomposition [29, Sect. 13], [23, 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"

Goal: Estimate relative perturbations in $F(\mathbf{x})$ due to relative perturbations in $\mathbf{x}$.
(*cf.* the same investigations for linear systems of equations in Sect. 2.5.5 and Thm. 2.5.24)

We assume that $\mathbb{K}^n$ is equipped with *some* vector norm ($\rightarrow$ Def. 2.5.1) and we use the induced matrix norm ($\rightarrow$ Def. 2.5.5) on $\mathbb{K}^{n,n}$.

Using linearity and the elementary estimate $\|\mathbf{M}\mathbf{x}\| \leq \|\mathbf{M}\| \|\mathbf{x}\|$, which is a direct consequence of the definition of an induced matrix norm, we obtain

$$\begin{aligned} \mathbf{A}\mathbf{x} = \mathbf{y} \quad &\Rightarrow \quad \|\mathbf{x}\| \leq \left\|\mathbf{A}^{-1}\right\| \|\mathbf{y}\| \\ \mathbf{A}(\mathbf{x}+\Delta\mathbf{x}) = \mathbf{y}+\Delta\mathbf{y} \quad &\Rightarrow \quad \mathbf{A}\Delta\mathbf{x} = \Delta\mathbf{y} \quad \Rightarrow \quad \|\Delta\mathbf{y}\| \leq \|\mathbf{A}\| \|\Delta\mathbf{x}\| \end{aligned}$$

R. Hiptmair

rev 30401, January 28, 2011

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

relative perturbation in result — relative perturbation in data

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

△

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 $\mathbf{T}(\mu)$ ▷

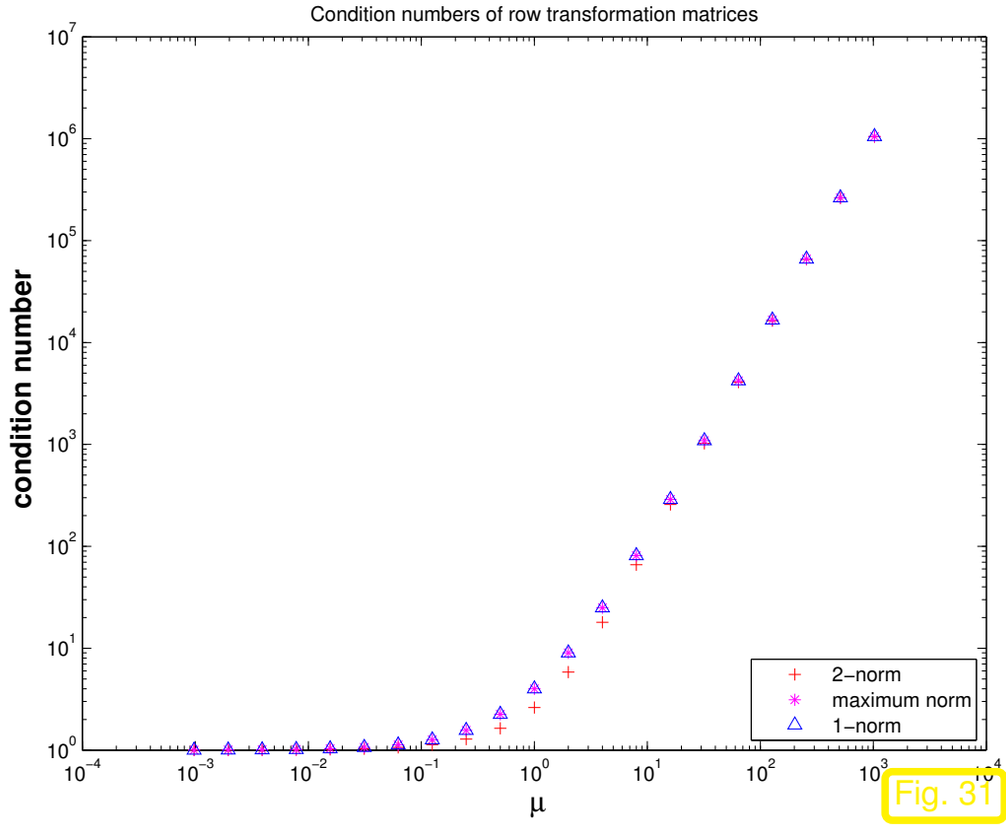


Fig. 31

R. Hiptmair

rev 30401, January 28, 2011

Code 2.8.4: computing different condition numbers of row transformation matrices

```
T = eye(2); res = [];
for mult = 2.^(-10:10)
    T(1,2) = mult;
    res = [res; mult, cond(T,2), cond(T,'inf'), cond(T,1)];
end
figure;
loglog(res(:,1),res(:,2),'r+', res(:,1),res(:,3),'m*',
  res(:,1),res(:,4),'b^');
xlabel('{\bf \mu}','fontsize',14);
ylabel('{\bf condition number}','fontsize',14);
title('Condition numbers of row transformation matrices');
legend('2-norm', 'maximum norm', '1-norm', 'location',
  'southeast');
print -depsc2 '../PICTURES/rowtrfcond.eps';
```

R. Hiptmair

rev 30401, January 28, 2011

Observation: $\mathrm{cond}(\mathbf{T}(\mu))$ large for large μ

As explained in Sect. 2.2, Gaussian (forward) elimination can be viewed as successive multiplication

with elimination matrices. If an elimination matrix has a large condition number, then small relative errors in the entries of intermediate matrices caused by earlier roundoff errors can experience massive amplification and, thus, spoil all further steps (➤ loss of numerical stability, Def. 2.5.11).

Therefore, the entries of elimination matrices should be kept small, and this is the main rationale behind (partial) pivoting ($\rightarrow$ Sect. 2.3), which ensures that multipliers have modulus ≤ 1 throughout forward elimination.

$\diamond$

There is a class of transformation that avoids any amplification of relative errors (measured in Euclidean vector norm)!

Recall from linear algebra [23, Sect. 2.8]:

Definition 2.8.5 (Unitary and orthogonal matrices)**.**

- $\mathbf{Q} \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$, *is unitary, if* $\mathbf{Q}^{-1} = \mathbf{Q}^H$.
- $\mathbf{Q} \in \mathbb{R}^{n,n}$, $n \in \mathbb{N}$, *is orthogonal, if* $\mathbf{Q}^{-1} = \mathbf{Q}^T$.

Theorem 2.8.6 (Criteria for Unitarity)**.**

$$\mathbf{Q} \in \mathbb{C}^{n,n} \quad \textit{unitary} \quad \Leftrightarrow \quad \|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2 \quad \forall \mathbf{x} \in \mathbb{K}^n \ .$$

$\mathbf{Q}$ unitary $\Rightarrow$ $\operatorname{cond}(\mathbf{Q}) = 1$ $\overset{(2.8.2)}{\blacktriangleright}$ unitary transformations enhance (numerical) stability

If $\mathbf{Q} \in \mathbb{K}^{n,n}$ unitary, then

- all rows/columns (regarded as vectors $\in \mathbb{K}^n$) have Euclidean norm $= 1$,
- all rows/columns are pairwise orthogonal (w.r.t. Euclidean inner product),
- $|\det \mathbf{Q}| = 1$, and all eigenvalues $\in \{z \in \mathbb{C}\colon |z| = 1\}$.
- $\|\mathbf{Q}\mathbf{A}\|_2 = \|\mathbf{A}\|_2$ for any matrix $\mathbf{A} \in \mathbb{K}^{n,m}$

R. Hiptmair

rev 30401, January 28, 2011

Drawbacks of LU-factorization:

☹ often pivoting required (→ destroys structure, Ex. 2.6.26, leads to fill-in)

☹ Possible (theoretical) instability of partial pivoting $\rightarrow$ Ex. 2.5.14

Stability problems of Gaussian elimination without pivoting are due to the fact that row transformations can convert well-conditioned matrices to ill-conditioned matrices, *cf.* Ex. 2.5.14

Which bijective row transformations preserve the Euclidean condition number of a matrix ?

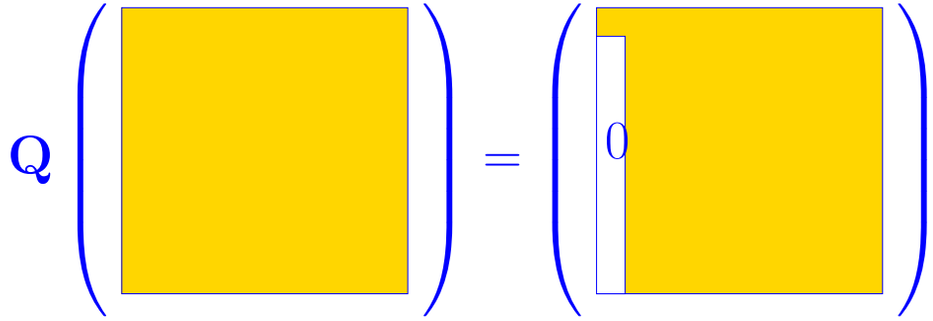
R. Hiptmair

rev 30401, January 28, 2011

➢ transformations hat preserve the Euclidean norm of a vector **!**

Investigate algorithms that use orthogonal/unitary row transformations to convert a matrix to upper triangular form.

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



with $\mathbf{Q}^H = \mathbf{Q}^{-1}$.

This "annihilation of column entries" is the key operation in Gaussian forward elimination, where it is achieved by means of non-unitary row transformations, see Sect. 2.2. Now we want to find a counterpart of Gaussian elimination based on unitary row transformations on behalf of numerical stability.

Example 2.8.7 ("Annihilating" orthogonal transformations in 2D).

In 2D: two possible orthogonal transformations make 2nd component of $\mathbf{a} \in \mathbb{R}^2$ vanish:

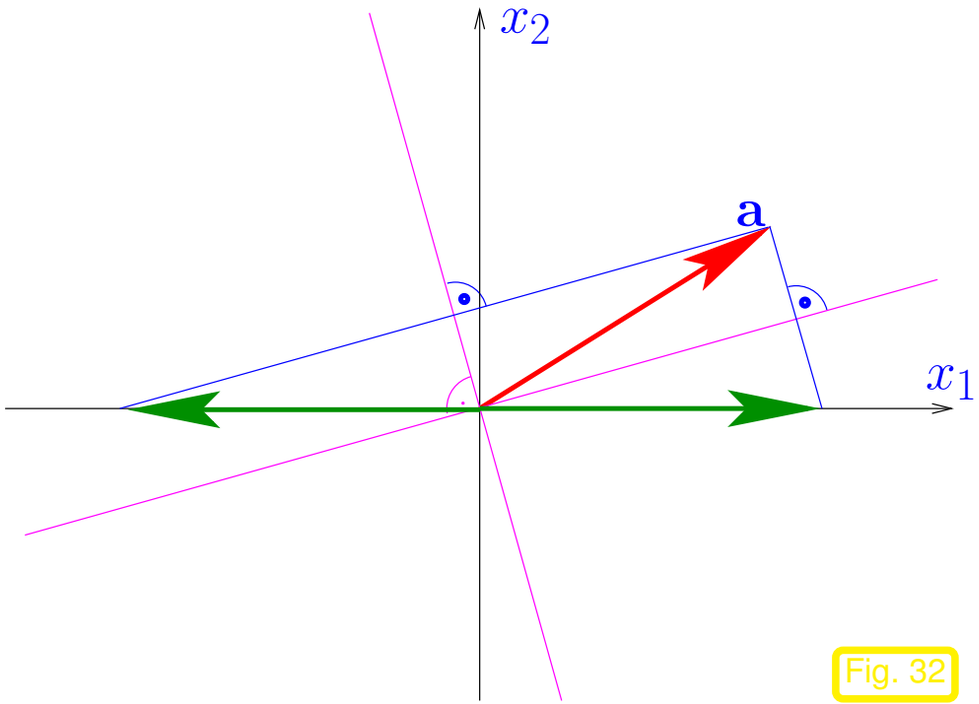


Fig. 32

reflection at angle bisector,

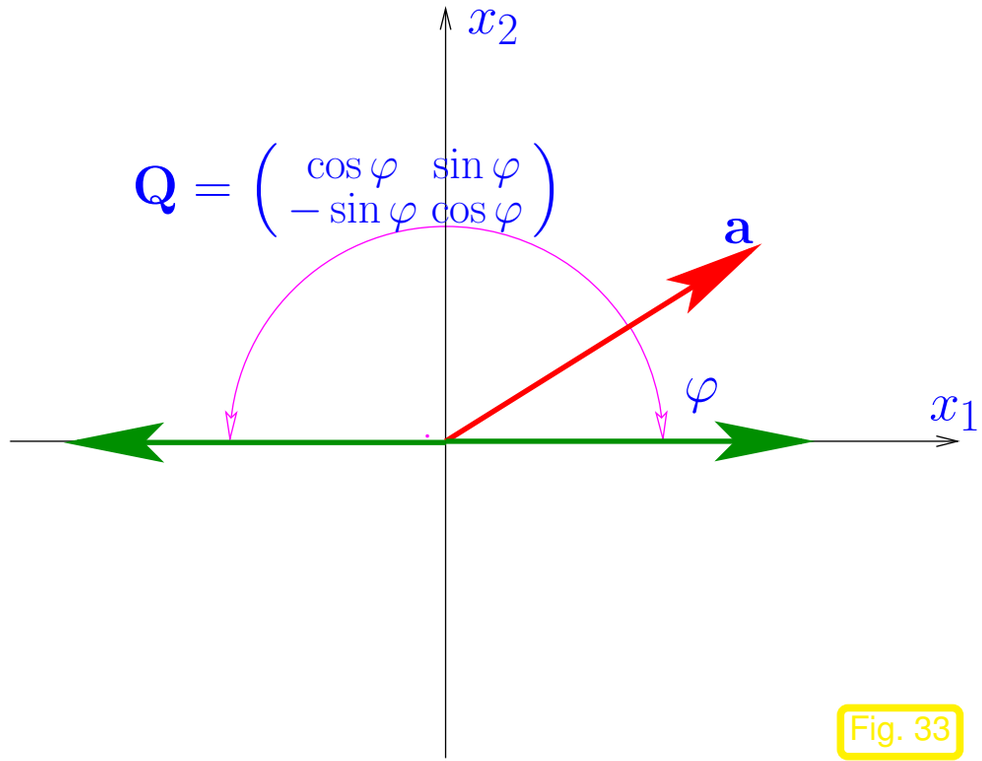


Fig. 33

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

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

"Geometric derivation" of Householder reflection, see Figure 32

Given $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ with $\|\mathbf{a}\| = \|\mathbf{b}\|$, the difference vector $\mathbf{v} = \mathbf{b} - \mathbf{a}$ is orthogonal to the bisector.

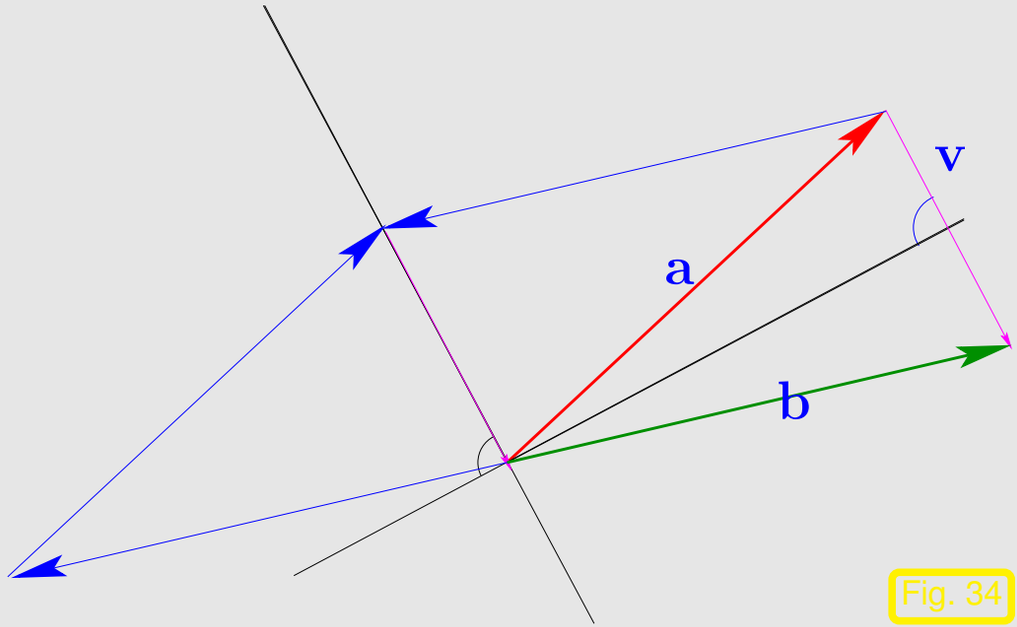


Fig. 34

$$\mathbf{b} = \mathbf{a} - (\mathbf{a} - \mathbf{b}) = \mathbf{a} - \mathbf{v}\frac{\mathbf{v}^T\mathbf{v}}{\mathbf{v}^T\mathbf{v}}$$
$$= \mathbf{a} - 2\mathbf{v}\frac{\mathbf{v}^T\mathbf{a}}{\mathbf{v}^T\mathbf{v}} = \mathbf{a} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\mathbf{a} = \mathbf{H}(\mathbf{v})\mathbf{a} ,$$

because, due to orthogonality $(\mathbf{a} - \mathbf{b}) \perp (\mathbf{a} + \mathbf{b})$

$$(\mathbf{a} - \mathbf{b})^T(\mathbf{a} - \mathbf{b}) = (\mathbf{a} - \mathbf{b})^T(\mathbf{a} - \mathbf{b} + \mathbf{a} + \mathbf{b}) = 2(\mathbf{a} - \mathbf{b})^T\mathbf{a} .$$

Remark 2.8.9 (Details of Householder reflections).

R. Hiptmair

rev 30401, January 28, 2011

- Practice: for the sake of numerical stability (in order to avoid so-called *cancellation*) choose

$$\mathbf{v} = \begin{cases} \frac{1}{2}(\mathbf{a} + \|\mathbf{a}\|_2 \mathbf{e}_1) & \text{, if } a_1 > 0 \text{ ,} \\ \frac{1}{2}(\mathbf{a} - \|\mathbf{a}\|_2 \mathbf{e}_1) & \text{, if } a_1 \leq 0 \text{ .} \end{cases}$$

  However, this is not really needed [30, Sect. 19.1] !

- If $\mathbb{K} = \mathbb{C}$ and $a_1 = |a_1| \exp(i\varphi)$, $\varphi \in [0, 2\pi]$, then choose

$$\mathbf{v} = \frac{1}{2}(\mathbf{a} \pm \|\mathbf{a}\|_2 \mathbf{e}_1 \exp(-i\varphi)) \quad \text{in} \quad (2.8.8).$$

- efficient storage of Householder matrices $\rightarrow$ [3]

Choice 2: successive Givens rotations [29, Sect. 14] ($\rightarrow$ 2D case)

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

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
```

Again, two options for annihilating rotation, see Ex. 2.8.7: either can be chosen and guarantees numerical stability.

So far, we know how to annihilate a single component of a vector by means of a Givens rotation that targets that component and some other (the first in (2.8.10)).

However, we aim to map *all* components to zero except for the first.

☞ This can be achieved by $n-1$ *successive* Givens rotations.

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

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

✎ Notation: $\mathbf{G}_{ij}(a_1, a_2) \hat{=}$ Givens rotation aimed at rows i and j of the matrix.

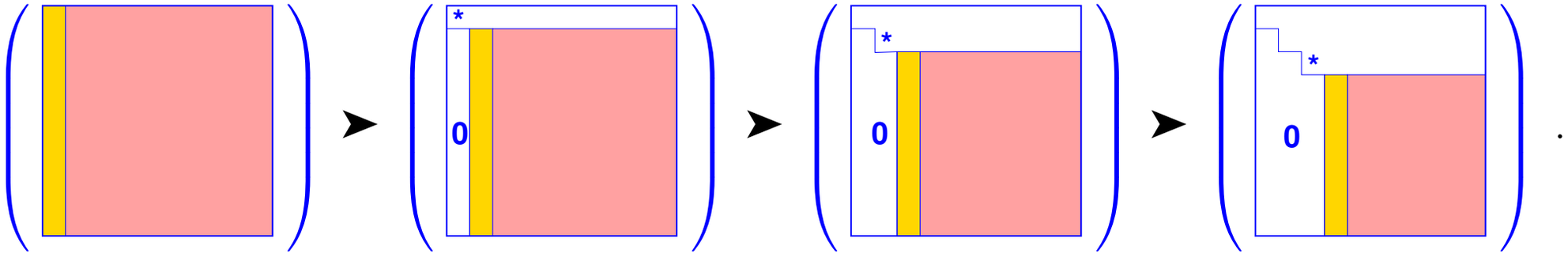
Code 2.8.13: Transformation of a vector according to (2.8.12)

```
function [a,Q] = givenscoltrf(a)
% Orthogonal transformation of a (column) vector into a multiple of the
% first unit vector by successive Givens transformations
n=length(a);
Q=eye(n); % Assemble rotations in matrix, alternative see Rem. 2.8.21
for j=2:n
  G=planerot([a(1);a(j)]); % see Code 2.8.10
  a([1,j])=G*a([1,j]);
  Q(:,[1,j])=Q(:,[1,j])*G';
end
```


Transformation to *upper triangular form* ($\rightarrow$ Def. 2.2.2) by successive unitary transformations:

We may use either Householder reflections or successive Givens rotations as explained above.

Visualization of annihilation of lower triangular part of a matrix by suitable successive Householder reflections (2.8.8):



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

= modified in course of transformations.

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

QR-factorization
(QR-decomposition)

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

The same can be achieved by suitable successive Givens rotations:

R. Hiptmair

rev 30401, January 28, 2011

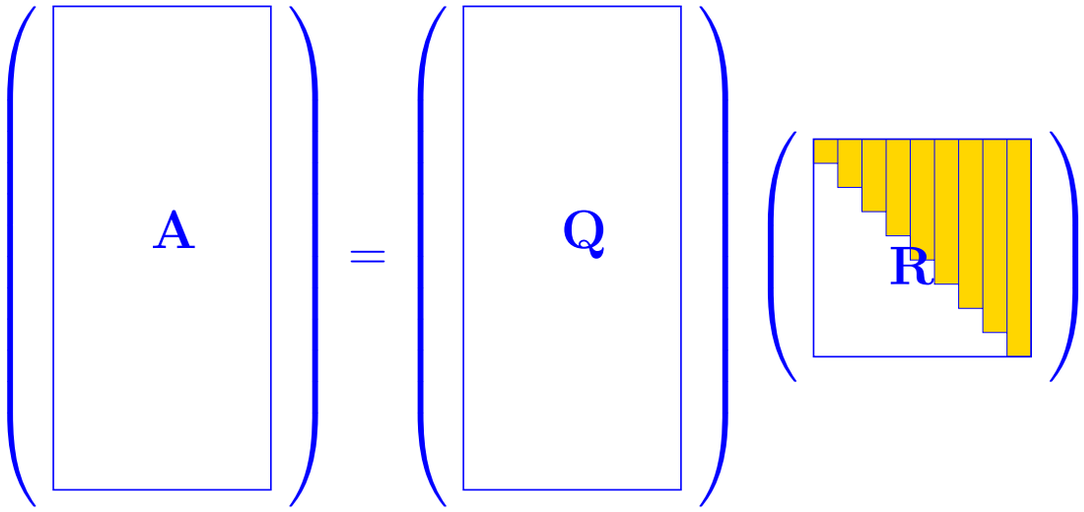
Code 2.8.14: QR decomposition by successive Givens rotations

```
function [Q,A] = qrgivens(A)
% in situ QR decomposition of square matrix A, same semantics as
%  MATLAB built qr() function
n=size(A,1);
Q=eye(n); % Assemble rotations in matrix, alternative see Rem. 2.8.21
for i=1:(n-1)
  for j=n:-1:(i+1)
    G=planerot([A(j-1,i);A(j,i)]); % see Code 2.8.10
    A([j-1,j],:)=G*A([j-1,j],:);
    Q(:,[j-1,j])=Q(:,[j-1,j])*G';
  end
end
```

☞ Obvious asymptotic computational complexity: $O(n^3)$ (for $\mathbf{A} \in \mathbb{R}^{n,n}$)

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

$m > n$:



$$, \mathbf{A} = \mathbf{QR}, \quad \begin{matrix} \mathbf{Q} \in \mathbb{K}^{m,n} \\ \mathbf{R} \in \mathbb{K}^{n,n} \end{matrix} , \tag{2.8.15}$$

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* $r_{ii} > 0$.

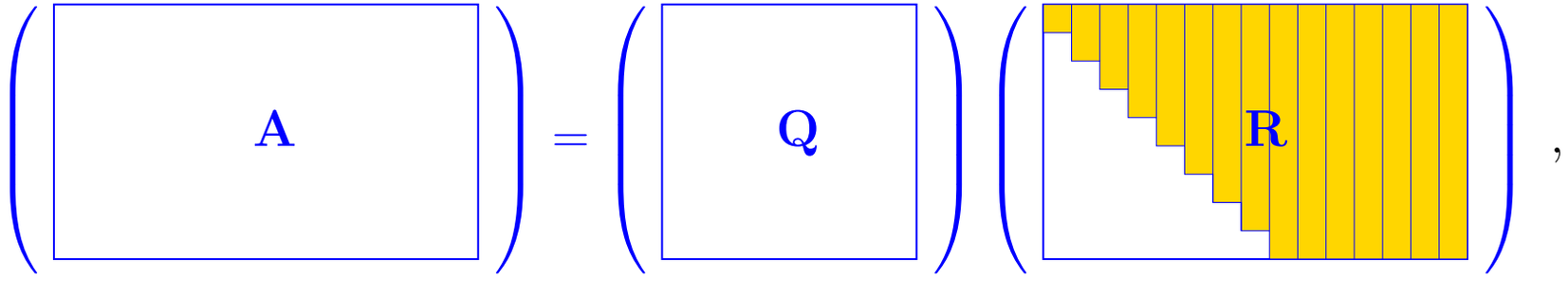
Proof. we observe that $\mathbf{R}$ is regular, if $\mathbf{A}$ has full rank n. Since the regular upper triangular matrices form a group under multiplication:

$$\mathbf{Q}_1\mathbf{R}_1 = \mathbf{Q}_2\mathbf{R}_2 \quad \Rightarrow \quad \mathbf{Q}_1 = \mathbf{Q}_2\mathbf{R} \quad \text{with upper triangular } \mathbf{R} := \mathbf{R}_2\mathbf{R}_1^{-1} .$$

$$\mathbf{I} = \mathbf{Q}_1^H \mathbf{Q}_1 = \mathbf{R}^H \underbrace{\mathbf{Q}_2^H \mathbf{Q}_2}_{=\mathbf{I}} \mathbf{R} = \mathbf{R}^H \mathbf{R} \,.$$

The assertion follows by uniqueness of Cholesky decomposition, Lemma 2.7.14. □

$m < n$:



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

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

Remark 2.8.17 (Choice of unitary/orthogonal transformation).

When to use which unitary/orthogonal transformation for QR-factorization **?**

Householder reflections advantageous for fully populated target columns (dense matrices).

Givens rotations more efficient ($\leftarrow$ more selective), if target column sparsely populated. △

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

```
1 % Timing QR factorizations
2
3 K = 3; r = [];
```

R. Hiptmair

rev 30401, January 28, 2011

```
4  for n=2.^(2:6)
5    m = n*n;
6
7    A = (1:m)'*(1:n) + [eye(n);ones(m-n,n)];
8    t1 = 1000; for k=1:K, tic; [Q,R] = qr(A); t1 = min(t1,toc);
       clear Q,R; end
9    t2 = 1000; for k=1:K, tic; [Q,R] = qr(A,0); t2 = min(t2,toc);
       clear Q,R; end
10   t3 = 1000; for k=1:K, tic; R = qr(A); t3 = min(t3,toc); clear
       R; end
11   r = [r; n , m , t1 , t2 , t3];
12 end
```


R. Hiptmair

rev 30401, January 28, 2011

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

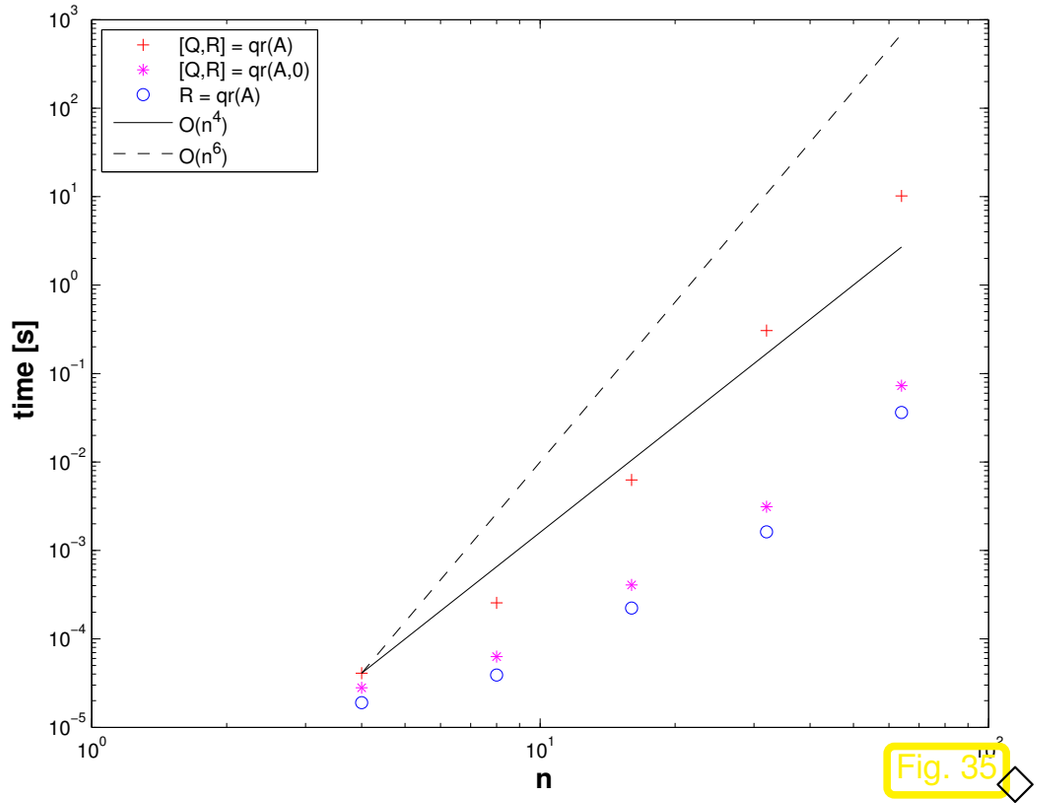


Fig. 35 ◇

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

Remark 2.8.20 (QR-orthogonalization).

$$\left(\begin{array}{|c|}\hline \\ \mathbf{A} \\ \\ \hline \end{array}\right) = \left(\begin{array}{|c|}\hline \\ \mathbf{Q} \\ \\ \hline \end{array}\right) \left(\begin{array}{|c|}\hline \mathbf{R} \\ \hline \end{array}\right) \quad , \quad \mathbf{A}, \mathbf{Q} \in \mathbb{K}^{m,n}, \mathbf{R} \in \mathbb{K}^{n,n} \; .$$

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

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

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

△

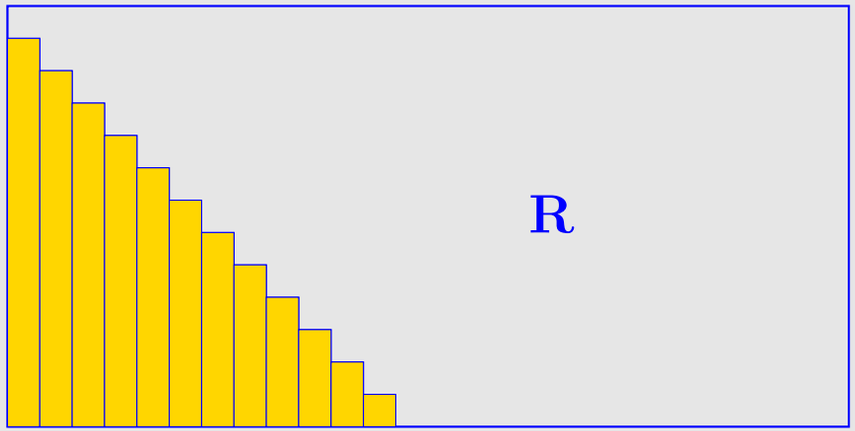
Remark 2.8.21 (Keeping track of unitary transformations).

$$\text{How to store } \begin{matrix}\mathbf{G}_{i_1 j_1}(a_1, b_1) \cdot \cdots \cdot \mathbf{G}_{i_k j_k}(a_k, b_k) \\ \mathbf{H}(\mathbf{v}_1) \cdot \cdots \cdot \mathbf{H}(\mathbf{v}_k)\end{matrix} , \quad ?$$

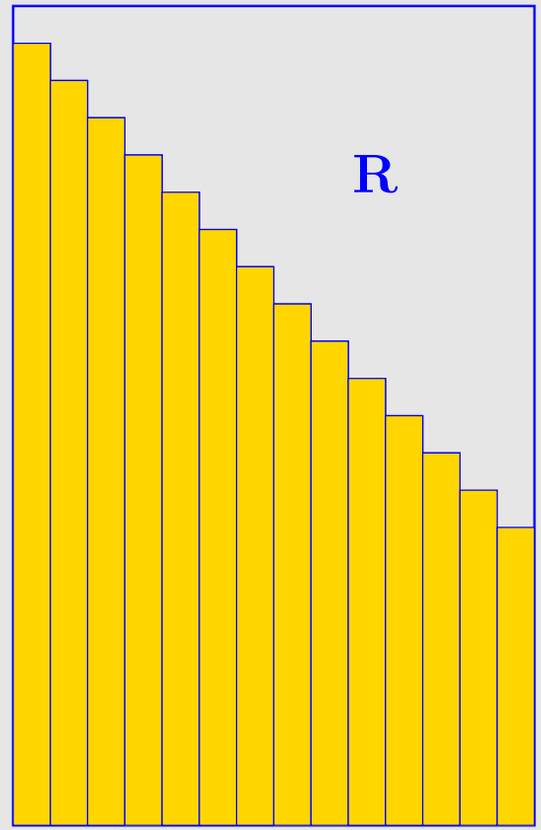
☛ For Householder reflections

$$\mathbf{H}(\mathbf{v}_1) \cdot \cdots \cdot \mathbf{H}(\mathbf{v}_k): \quad \text{store } \mathbf{v}_1, \ldots, \mathbf{v}_k$$

For in place QR-factorization of $\mathbf{A} \in \mathbb{K}^{m,n}$: store "“Householder vectors” $\mathbf{v}_j$ (decreasing size !) in lower triangle of $\mathbf{A}$



↑ Case $m < n$



= Householder vectors

Case $m > n \rightarrow$

☛ Convention for Givens rotations ($\mathbb{K} = \mathbb{R}$)

$$\mathbf{G} = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \quad \Rightarrow \quad \text{store} \quad \rho := \begin{cases} 1 & \text{, if } \gamma = 0 \text{ ,} \\ \frac{1}{2}\operatorname{sign}(\gamma)\sigma & \text{, if } |\sigma| < |\gamma| \text{ ,} \\ 2\operatorname{sign}(\sigma)/\gamma & \text{, if } |\sigma| \geq |\gamma| \text{ .} \end{cases}$$

$$\blacktriangleright \quad \begin{cases} \rho = 1 & \Rightarrow \quad \gamma = 0 \text{ ,} \quad \sigma = 1 \\ |\rho| < 1 & \Rightarrow \quad \sigma = 2\rho \text{ ,} \quad \gamma = \sqrt{1-\sigma^2} \\ |\rho| > 1 & \Rightarrow \quad \gamma = 2/\rho \text{ ,} \quad \sigma = \sqrt{1-\gamma^2} \text{ .} \end{cases}$$

R. Hiptmair

rev 30401, January 28, 2011

Then store $\mathbf{G}_{ij}(a,b)$ as triple (i,j,ρ). The parameter ρ forgets the sign of the matrix $\mathbf{G}_{ij}$, so the signs of the corresponding rows in the transformed matrix $\mathbf{R}$ have to be changed accordingly.

The rationale for this convention is to curb the impact of roundoff errors.

> Storing orthogonal transformation matrices is usually inefficient **!**

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

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

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). → Ex. 2.5.14

Code 2.8.24: QR-fac. ↔ 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-+');
```

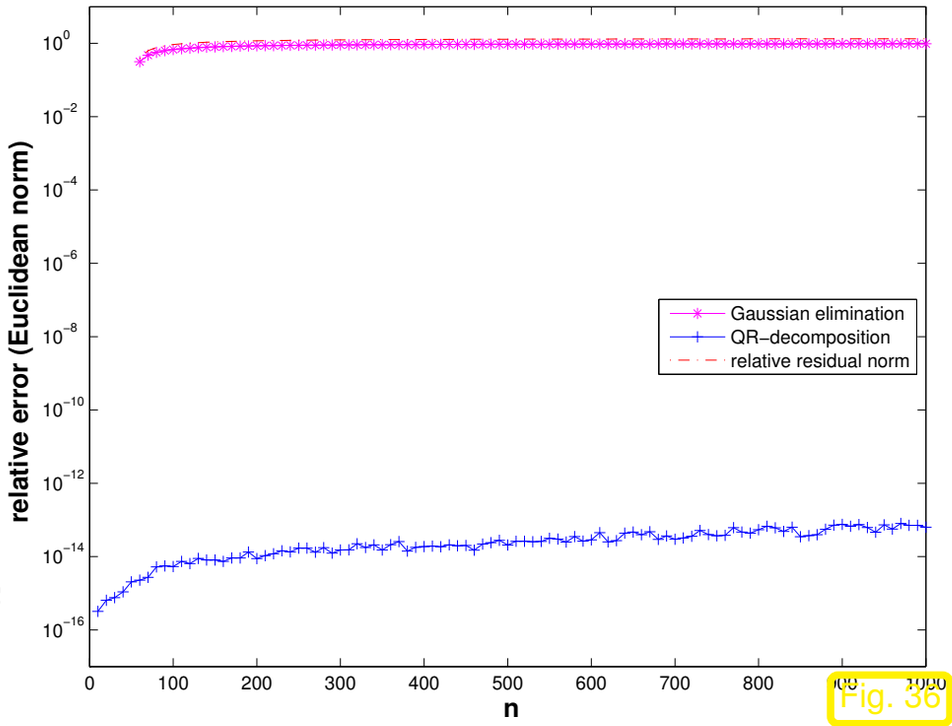


Fig. 36

◁ superior stability of QR-decomposition !

◇

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

bandwidth

$\mathbf{A} \in \mathbb{C}^{n,n}$ with QR-decomposition $\mathbf{A} = \mathbf{QR}$ $\Rightarrow$ $m(\mathbf{R}) \leq m(\mathbf{A})$ ($\rightarrow$ Def. 2.6.28)

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):

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

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,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 $O(n)$

◇

R. Hiptmair

rev 30401, January 28, 2011

Remark 2.8.27 (Storing the $\mathbf{Q}$-factor).

The previous example (Code 2.8.25) showed that assembly of the $\mathbf{Q}$-factor in the QR-factorization of $\mathbf{A}$ is *not needed*, when the linear system of equations $\mathbf{Ax} = \mathbf{b}$ is to be solved by means of QR-factorization: the orthogonal transformations can simply be applied to the right hand side(s) whenever they are applied to the columns of $\mathbf{A}$.

Discussion of Rem. 2.2.12 for QR-factorization:

Inefficient (!) code ▷

```
% Setting:  N ≫ 1,
% large tridiagonal matrix A ∈ ℝ^{n,n}
[Q,R] = qr(A);
for j=1:N
    x = R\(Q'*b);
    b = some_function(x);
end
```

- $\mathbf{Q} \in \mathbb{R}^{n,n}$ dense matrix
- $\mathbf{R} \in \mathbb{R}^{n,n}$ dense matrix

➢ $O(n^2)$ computational effort for executing loop body

Remedies:

- Store $\mathbf{R}$ in sparse matrix format, see Code 2.8.25, Sect. 2.6.2.

R. Hiptmair

rev 30401, January 28, 2011

- Store Givens rotations contained in $\mathbf{Q}$ as array of triplets: $\mathbf{G}_{lk}$ is coded as

  `[l,k,rho]` ,

  where `rho` ($\hat{=}\ \rho$) is chosen as in Rem. 2.8.21.

△

Remark 2.8.28 (Testing for near singularity of a matrix).

Very small (w.r.t. matrix norm) element r_{ii} in QR-factor $\mathbf{R}$ $\quad\leftrightarrow\quad$ $\mathbf{A}$ "nearly singular"

△

2.9 ModificationTechniques

Example 2.9.1 (Resistance to currents map).

Large (linear) electric circuit (modelling → Ex. 2.0.1) ▷

Sought:

Dependence of (certain) branch currents on "continuously varying" resistance R_x

(➢ currents for many different values of R_x)

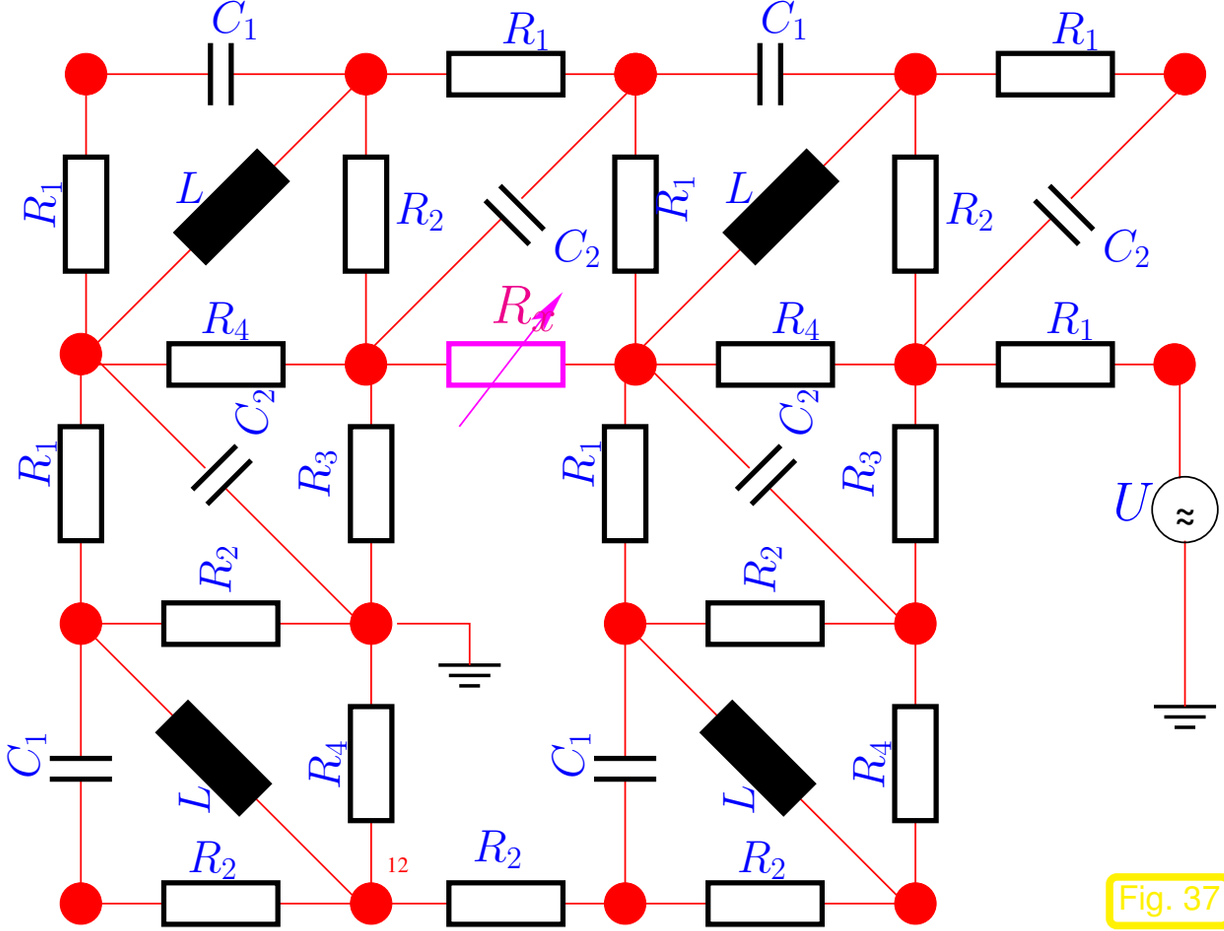


Fig. 37

R. Hiptmair

rev 30401, November 9, 2010

➢ Only *a few* entries of the nodal analysis matrix $\mathbf{A}$ (→ Ex. 2.0.1) are affected by variation of R_x!
(If R_x connects nodes i & j ⇒ only entries $a_{ii}, a_{jj}, a_{ij}, a_{ji}$ of $\mathbf{A}$ depend on R_x)

➢ Repeating Gaussian elimination/LU-factorization for each value of R_x from scratch seems wasteful.

Idea:
- compute (sparse) LU-factorization of $\mathbf{A}$ once
- Repeat: update LU-factors for modified $\mathbf{A}$

+

(partial) forward and backward substitution

◇

Problem: Efficient *update* of matrix factorizations in the case of 'slight" changes of the matrix [20, Sect. 12.6], [50, 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 $x \in \mathbb{K}$

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

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

Recall: $\mathbf{e}_i \hat{=} i$-th unit vector

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

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

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

Both matrix modifications (2.9.3) and (2.9.5) are specimens of a rank-1-modifications.

$\diamond$

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

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.*

Proof. Straightforward algbra:

$$\left(\mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I}+\mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^H\mathbf{A}^{-1}\right)(\mathbf{A}+\mathbf{U}\mathbf{V}^H) =$$

$$\mathbf{I} - \mathbf{A}^{-1}\mathbf{U}\underbrace{(\mathbf{I}+\mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})^{-1}(\mathbf{I}+\mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})}_{=\mathbf{I}}\mathbf{V}^H + \mathbf{A}^{-1}\mathbf{U}\mathbf{V}^H = \mathbf{I} .$$

Uniqueness of the inverse settles the case. □

R. Hiptmair

rev 30401, November 9, 2010

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

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

△

The approach of Rem. 2.9.7 is certainly efficient, but may *suffer from instability* similar to Gaussian elimination without pivoting, *cf.* Ex. 2.3.1.

This can be avoided by using QR-factorization ($\rightarrow$ Sect. 2.8) and corresponding update techniques. This was a key rationale for studying QR-factorization for the solution of linear system of equations.

Other important applications of QR-factorization will be discussed later in Chapter 7.

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

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

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

② Objective: $\mathbf{w} \rightarrow \|\mathbf{w}\|\,\mathbf{e}_1$ ➤ 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 the difference between this arrangement of successive Givens rotations to turn $\mathbf{w}$ into a multiple of the first unit vector $\mathbf{e}_1$, and the different sequence of Givens rotations discussed in Sect. 2.8. Both serve the same purpose, but we shall see in a moment that the smart selection of Givens rotations is crucial in the current context.

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

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

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

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

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

➡ Asymptotic complexity $O(n^2)$

R. Hiptmair

rev 30401, November 9, 2010

Imagine that in (2.9.10) we had chosen to annihilate the components $2, \ldots, n$ of $\mathbf{w}$ by the product of Givens rotations $\mathbf{G}_{12}\mathbf{G}_{13}\cdots\mathbf{G}_{1,n-1}$. This would have resulted in a fully populated matrix $\mathbf{R}_1$!

In this case, the next step could be carried out with an effort $O(n^3)$ only.

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

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

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

R. Hiptmair

rev 30401, November 9, 2010

➡ 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* (→ Def. 2.7.9):

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

If the modified matrix is known to be s.p.d. ➢ Cholesky factorization will be stable. Thus, efficient modification of the Cholesky factor is of practical relevance.

Task: Efficient computation of Cholesky factorization $\widetilde{\mathbf{A}} = \widetilde{\mathbf{R}}^H\widetilde{\mathbf{R}}$ (→ 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 $\mathbf{w} := \mathbf{R}^{-H}\mathbf{v}$: $\mathbf{A} + \alpha\mathbf{v}\mathbf{v}^H = \mathbf{R}^H(\mathbf{I} + \alpha\mathbf{w}\mathbf{w}^H)\mathbf{R}$.

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

② Idea: formal Gaussian elimination: with $\widetilde{\mathbf{w}} = (w_2, \ldots, w_n)^T$ → see (2.1.9)

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

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

same structure ➢ recursion

Proceeding in this way, we obtain the $\mathbf{U}$ matrix of the LU-decomposition $\mathbf{I} + \alpha\mathbf{w}\mathbf{w}^H = \mathbf{L}\mathbf{U}$. In order to obtain the Cholesky decomposition we divide the lines of $\mathbf{U}$ by the square root of its diagonal terms, i.e., $\mathbf{I} + \alpha\mathbf{w}\mathbf{w}^H = \mathbf{L}\mathbf{U} = (\mathbf{L}\mathbf{D}^{1/2})(\mathbf{D}^{-1/2}\mathbf{U}) = \mathbf{R}^H\mathbf{R}$, where

$$\mathbf{D} = \operatorname{diag}\big(\operatorname{diag}(\mathbf{R})\big) = \operatorname{diag}(1+\alpha w_1^2, 1+\alpha^{(1)} w_2^2, \ldots).$$

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 `d`, `s` computed → (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 $\mathbf{R}_1$:

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

R. Hiptmair

rev 30401, November 9, 2010

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

- 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

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

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

R. Hiptmair

rev 30401, November 9, 2010

2.9.0.2 Adding a column

Let us adopt an academic point of view: Before we have seen how to update a QR-factorization in the case of rank-1-modification of a *square* matrix.

However, the QR-factorization makes sense for an arbitrary *rectangular* matrix. A possible modification of rectangular matrices is achieved by adding a row or a column. How can QR-factors updated efficiently for these kinds of modifications.

An application of these modification techniques will be given in Chapter 7.

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

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

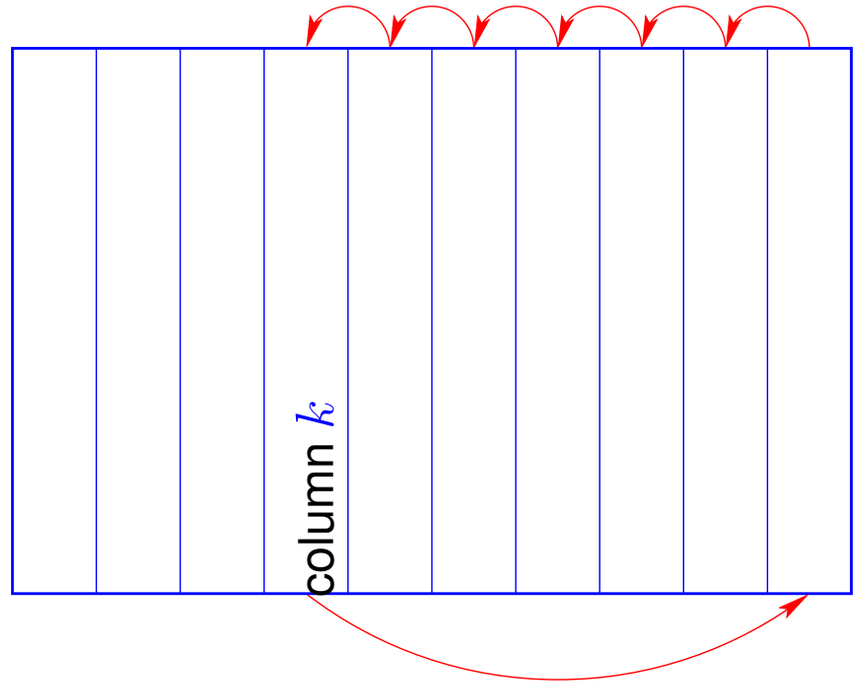
Idea: Easy, if $k = n+1$ (adding last column)

$\exists$ column permutation

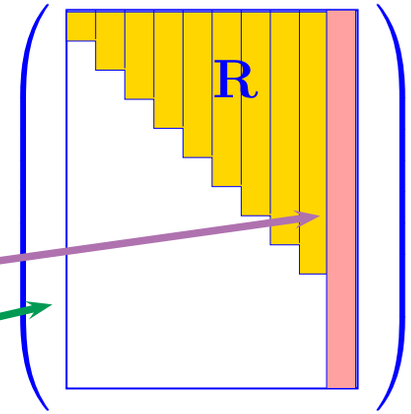
$$k \mapsto n+1 \;,\; i \mapsto i-1 \;,\; i = k+1, \ldots, n+1$$

$\sim$ permutation matrix

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



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



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

R. Hiptmair

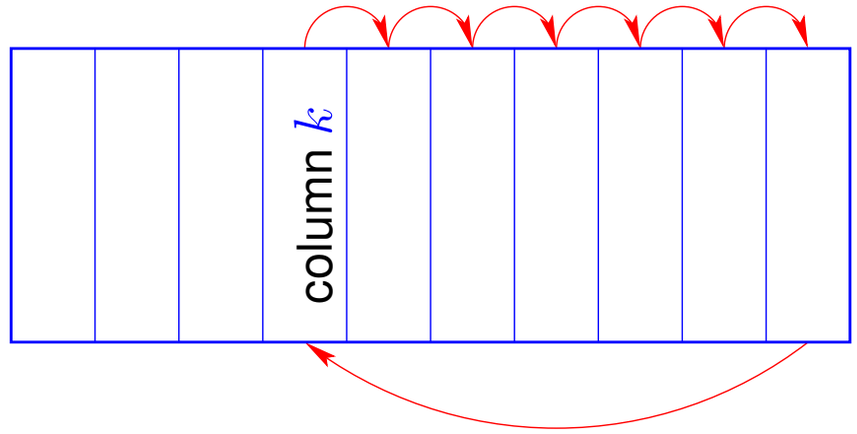
rev 30401, November 9, 2010

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

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

inverse permutation:

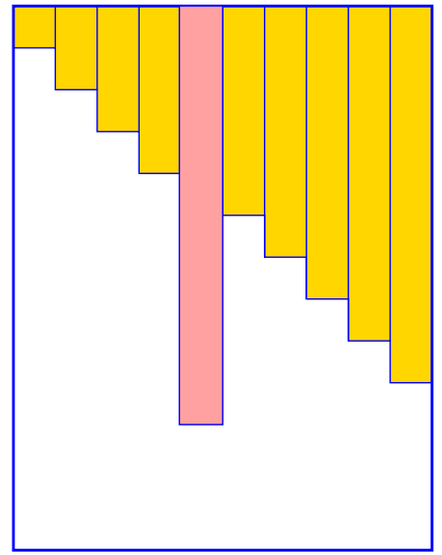
$\sim$ right multiplication with $\mathbf{P}^H$



R. Hiptmair

rev 30401, November 9, 2010

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

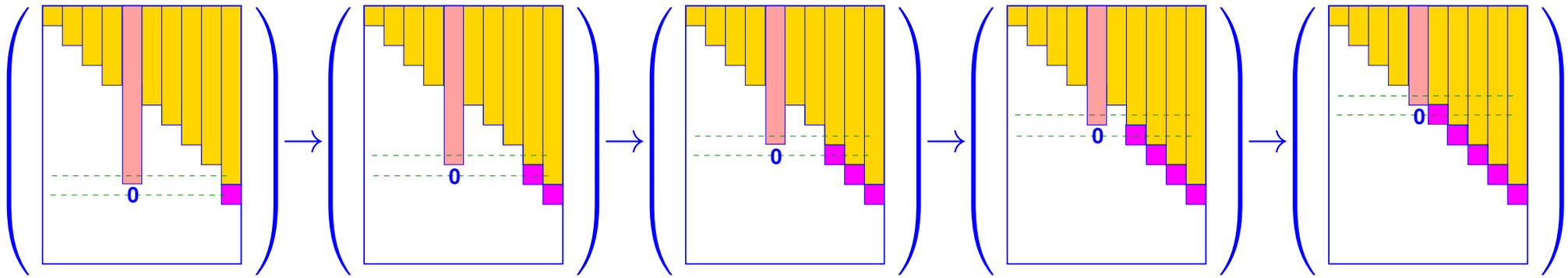


② $n+1-k$ successive Givens rotations $\Rightarrow$ upper triangular matrix $\widetilde{\mathbf{R}}$

R. Hiptmair

rev 30401, November 9, 2010

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



- - - - $\hat{=}$ 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)
```

R. Hiptmair

rev 30401, November 9, 2010

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

Task: efficient computation of QR-factorization $\widetilde{\mathbf{A}} = \widetilde{\mathbf{Q}}\widetilde{\mathbf{R}}$ 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 $m+1 \leftarrow k$, $i \leftarrow i+1$, $i = k, \ldots, m$:
→ unitary permutation matrix (→ Def. 2.3.11) $\mathbf{P} \in \{0,1\}^{m+1,m+1}$

$$\mathbf{P}\widetilde{\mathbf{A}} = \begin{pmatrix} \mathbf{A} \\ \mathbf{v}^T \end{pmatrix} \quad \blacktriangleright \quad \begin{pmatrix} \mathbf{Q}^H & 0 \\ 0 & 1 \end{pmatrix} \mathbf{P}\widetilde{\mathbf{A}} = \begin{pmatrix} \mathbf{R} \\ \mathbf{v}^T \end{pmatrix} = \left(\begin{array}{c} \mathbf{R} \\ \mathbf{v}^T \end{array} \right) .$$

case $m = n$

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

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

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

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

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

☞ Similar update algorithms exist for modifications arising from dropping one row or column of a matrix.

R. Hiptmair

rev 30401, November 9, 2010

3 Data Interpolation in 1D

3.1 Abstract interpolation

Problem: (multidimensional) **data interpolation** (point interpolation):

Given: data points $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \ldots, m$, $\mathbf{x}_i \in D \subset \mathbb{R}^n$, $\mathbf{y}_i \in \mathbb{R}^d$

Goal: *reconstruction of a (continuous) function* $\mathbf{f} : D \mapsto \mathbb{R}^d$ satisfying interpolation conditions

$$\boxed{\mathbf{f}(\mathbf{x}_i) = \mathbf{y}_i, \quad i = 1, \ldots, m}$$

Additional requirements:

- smoothness of $\mathbf{f}$, e.g. $\mathbf{f} \in C^1$, etc.
- shape of $\mathbf{f}$ (positivity, monotonicity, convexity $\rightarrow$ Sect. 3.5)

Focus in this chapter: $n = 1 \quad \leftrightarrow \quad$ interpolation of data depending on *one* parameter ($t \in \mathbb{R}$):

$$\text{interpolation conditions:} \quad \mathbf{f}(t_i) = \mathbf{y}_i \;, \quad i = 1, \ldots, m \;. \tag{3.1.1}$$

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

In this context: t, $y \mathrel{\hat{=}}$ two state variables of a physical system, a functional dependence $y = y(t)$ is assumed.

Known: several *accurate* measurements

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

Examples:

t	y
voltage U	current I
pressure p	density ρ
magnetic field H	magnetic flux B
$\ldots$	$\ldots$

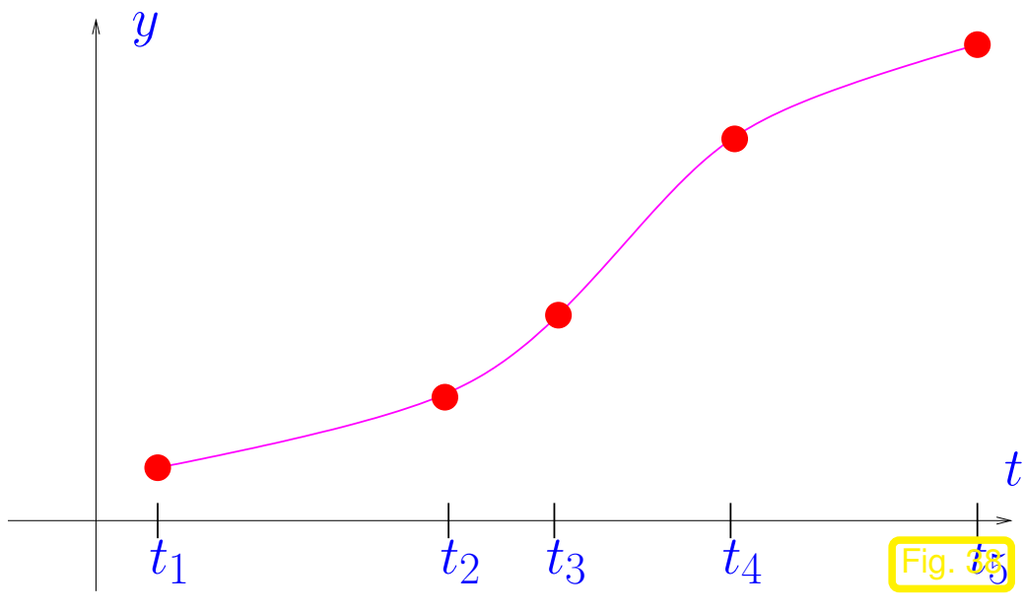


Fig. 35

Meaning of attribute “accurate”: justification for interpolation. If measured values y_i were affected by considerable errors, one would not impose the interpolation conditions (3.1.1), but opt for data fitting.

◇

Remark 3.1.3 (Function representation).

! General function $f : D \subset \mathbb{R} \mapsto \mathbb{K}^d$, D interval, contains an “infinite amount of information”.

? How to represent f on a computer?

→ Idea: parametrization, a finite number of parameters $c_1, \ldots, c_n$, $n \in \mathbb{N}$, characterizes f.

Special case: Representation with *finite linear combination* of basis functions

$b_j : D \subset \mathbb{R} \mapsto \mathbb{K}$, $j = 1, \ldots, n$:

$$f = \sum_{j=1}^{n} c_j b_j \quad , \quad c_j \in \mathbb{K}^d \, . \tag{3.1.4}$$

➙ $f \in$ finite dimensional function space $V_n := \operatorname{Span}\{b_1, \ldots, b_n\}$.

☞ Focus in this chapter will be on this special case.

△

Remark 3.1.5 (Interpolation as linear mapping).

Setting: interpolation based on finite linear combinations of basis functions, see Rem. 3.1.3:

$$\text{(3.1.1) \& (3.1.4)} \quad \Rightarrow \quad f(t_i) = \sum_{j=1}^{n} c_j b_j(t_i) \, , \quad i = 1, \ldots, m \, , \tag{3.1.6}$$

$$\Updownarrow$$

$$\mathbf{A}\mathbf{c} := \begin{pmatrix} b_1(t_1) & \ldots & b_n(t_1) \\ \vdots & & \vdots \\ b_1(t_m) & \ldots & b_n(t_m) \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} =: \mathbf{y} \, . \tag{3.1.7}$$

= $m \times n$ linear system of equations !

Necessary condition for unique solvability of interpolation problem (3.1.6) : $m = n$

If $m = n$ and $\mathbf{A}$ from (3.1.7) regular ($\rightarrow$ Def. 2.0.5),then

For *fixed* t_i the interpolation problem (3.1.6) defines linear mapping

$$\mathsf{I} : \begin{cases} \mathbb{K}^n & \mapsto & V_n \\ \mathbf{y} & \mapsto & f \end{cases}$$

data space

function space

More precisely:

An interpolation operator $\mathsf{I} : \mathbb{R}^{n+1} \mapsto C^0([t_0, t_n])$ for the given nodes $t_0 < t_1 < \cdots < t_n$ is called *linear*, if

$$\mathsf{I}(c\mathbf{y} + \beta\mathbf{z}) = c\mathsf{I}(\mathbf{y}) + \beta\mathsf{I}(\mathbf{z}) \quad \forall \mathbf{y}, \mathbf{z} \in \mathbb{R}^{n+1},\ c, \beta \in \mathbb{R}\ .$$

R. Hiptmair

rev 30401, November 9, 2010

Remark 3.1.8 ("Software solution" of interpolation problem).

```
1 class Interpolant {
2   private:
3   // various internal data describing f
4   public:
5     // Constructor:  computation of coefficients c_j of representation (3.1.4)
6     Interpolant(const vector<double> &t,const vector<double> &y);
7     // Evaluation operator for interpolant f
8     double operator () (double t) const;
9 };
```


Practical object oriented implementation of interpolation operator:

- Constructor: "setup phase", e.g. building and solving linear system of equations (3.1.7)
- Evaluation operator, e.g., implemented as evaluation of linear combination (3.1.4)

Crucial issue: computational effort for evaluation of interpolant at single point: $O(1)$ or $O(n)$ (or in between) ?

3.2 Polynomials

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

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

leading coefficient

Terminology: the functions $t \mapsto t^k$, $k \in \mathbb{N}_0$, are called monomials

$t \mapsto \alpha_k t^k + \alpha_{k-1} t^{k-1} + \cdots + \alpha_0$ = monomial representation of a polynomial.

Obvious: $\mathcal{P}_k$ is a vector space. What is its dimension?

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

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

Proof. Dimension formula by linear independence of monomials.

Why are polynomials important in computational mathematics ?

- ➙ Easy to compute, integrate and differentiate
- ➙ Vector space & algebra
- ➙ Analysis: Taylor polynomials & power series

Remark 3.2.3 (Polynomials in Matlab)*.*

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

△

R. Hiptmair

rev 30401, November 9, 2010

Remark 3.2.4 (Horner scheme). → [10, 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)

```
1 function y = polyval(p,x)
2 y = p(1); for i=2:length(p), y = x*y+p(i); end
```

Asymptotic complexity: $O(n)$

Use: MATLAB "built-in"-function `polyval(p,x);`.

△

3.3 Polynomial Interpolation: Theory [10, Sect. 8.2.1]

Interpolation problem (→ Sect. 3.1): (re-)construction of a polynomial through points (t_i, y_i) .

R. Hiptmair

rev 30401, February 19, 2011

Lagrange polynomial interpolation problem

Given the simple nodes $t_0, \ldots, t_n$, $n \in \mathbb{N}$, $-\infty < t_0 < t_1 < \cdots < t_n < \infty$ and the values $y_0, \ldots, y_n \in \mathbb{K}$ compute $p \in \mathcal{P}_n$ such that

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

General polynomial interpolation problem

Given the (possibly multiple) nodes $t_0, \ldots, t_n$, $n \in \mathbb{N}$, $-\infty < t_0 \le t_1 \le \cdots \le t_n < \infty$ and the values $y_0, \ldots, y_n \in \mathbb{K}$ compute $p \in \mathcal{P}_n$ such that

$$\frac{\mathrm{d}^k}{\mathrm{d}t^k} p(t_j) = y_j \quad \text{for} \quad k = 0, \ldots, l_j \quad \text{and} \quad j = 0, \ldots, n \,, \tag{3.3.2}$$

where $l_j := \max\{i - i' : t_j = t_i = t_{i'},\, i, i' = 0, \ldots, n\}$ is the multiplicity of the nodes t_j.

When all the multiplicities are equal to 2: Hermite interpolation (or osculatory interpolation)

$t_0 = t_1 < t_2 = t_3 < \cdots < t_{n-1} = t_n$ ➙ $p(t_{2j}) = y_{2j}$, $p'(t_{2j}) = y_{2j+1}$, (double nodes).

R. Hiptmair

rev 30401, February 19, 2011

3.3.1 Lagrange polynomials

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

$$\text{Lagrange polynomials} \quad L_i(t) := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - t_j}{t_i - t_j}, \quad i = 0, \ldots, n . \tag{3.3.3}$$

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

Recall the Kronecker symbol $\delta_{ij} = \begin{cases} 1 & \text{if } i = j , \\ 0 & \text{else.} \end{cases}$

Example 3.3.4. Lagrange polynomials for uniformly spaced nodes

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

Plot $n = 10,\ j = 0, 2, 5$ ➙

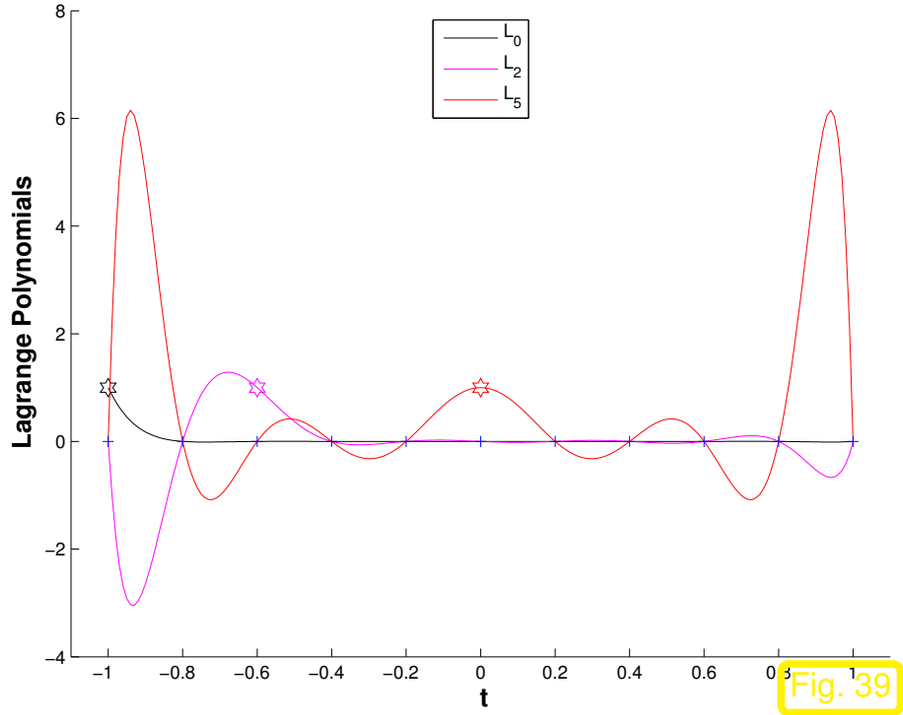


Fig. 39

◇

The Lagrange interpolation polynomial p for data $(t_i, y_i)_{i=0}^n$ has the representation:

$$p(t) = \sum_{i=0}^{n} y_i L_i(t) , \qquad \Rightarrow \qquad p \in \mathcal{P}_n \quad \text{and} \quad p(t_i) = y_i . \tag{3.3.5}$$

R. Hiptmair

rev 30401, February 19, 2011

Theorem 3.3.6 (Existence & uniqueness of Lagrange interpolation polynomial)**.**
The general polynomial interpolation problem (3.3.1) *admits a unique solution* $p \in \mathcal{P}_n$.

Proof. Consider the *linear* evaluation operator

$$\mathrm{eval}_{\mathcal{T}} : \begin{cases} \mathcal{P}_n & \mapsto \mathbb{R}^{n+1} \\ p & \mapsto (p(t_i))_{i=0}^{n} \end{cases},$$

which maps between finite-dimensional vector spaces of the same dimension, see Thm. 3.2.2.

Representation (3.3.5) $\Rightarrow$ existence of interpolating polynomial
$\Rightarrow$ $\mathrm{eval}_{\mathcal{T}}$ is surjective ("onto")

Known from linear algebra: for a linear mapping $T : V \mapsto W$ between finite-dimensional vector spaces with $\dim V = \dim W$ holds the equivalence

$$T \text{ surjective} \quad \Leftrightarrow \quad T \text{ bijective} \quad \Leftrightarrow \quad T \text{ injective.}$$

Applying this equivalence to $\mathrm{eval}_{\mathcal{T}}$ yields the assertion of the theorem $\square$

R. Hiptmair
rev 30401, February 19, 2011

Lagrangian polynomial interpolation leads to linear systems of equations: with monomial basis representation

$$p(t_j) = y_j \quad \Longleftrightarrow \quad \sum_{i=0}^{n} a_i t_j^i = y_j,\ j = 0,\dots,n$$

$\Longleftrightarrow$ solution of $(n+1) \times (n+1)$ linear system $\mathbf{Va} = \mathbf{y}$ with matrix

$$\mathbf{V} = \begin{pmatrix} 1 & t_0 & t_0^2 & \cdots & t_0^n \\ 1 & t_1 & t_1^2 & \cdots & t_1^n \\ 1 & t_2 & t_2^2 & \cdots & t_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^n \end{pmatrix}. \tag{3.3.7}$$

A matrix in the form of $\mathbf{V}$ is called Vandermonde matrix.

Given a column vector `t`, the corresponding Vandermonde matrix can be generated by

- `for j = 1:length(t); V(j,:) = t(j).^[0:length(t-1)]; end;`
- `for j = 1:length(t); V(:,j) = t.^(j-1); end;`
- `fliplr(vander(t))`

Theorem 3.3.8 (Lagrange interpolation as linear mapping). → *Rem. 3.1.5*

The polynomial interpolation in the nodes $\mathcal{T} := \{t_j\}_{j=0}^n$ *defines a linear operator*

$$\mathsf{I}_{\mathcal{T}} : \begin{cases} \mathbb{K}^{n+1} & \to \mathcal{P}_n \ , \\ (y_0, \ldots, y_n)^T & \mapsto \textit{interpolating polynomial } p \ . \end{cases} \tag{3.3.9}$$

Remark 3.3.10 (Matrix representation of interpolation operator).

In the case of Lagrange interpolation:

- if Lagrange polynomials are chosen as basis for $\mathcal{P}_n$, → $\mathsf{I}_{\mathcal{T}}$ is represented by the identity matrix;

R. Hiptmair

rev 30401, February 19, 2011

- if monomials are chosen as basis for $\mathcal{P}_n$, $\rightarrow$ $\mathsf{I}_{\mathcal{T}}$ is represented by the inverse of the Vandermonde matrix $\mathbf{V}$, see (3.3.7).

△

Definition 3.3.11 (Generalized Lagrange polynomials)**.**
The generalized Lagrange polynomials on the nodes $\mathcal{T} = \{t_j\}_{j=0}^{n} \subset \mathbb{R}$ *are* $L_i := \mathsf{I}_{\mathcal{T}}(\mathbf{e}_{i+1})$, $i = 0, \ldots, n$, *where* $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)^T \in \mathbb{R}^{n+1}$ *are the unit vectors.*

Theorem 3.3.12 (Existence & uniqueness of generalized Lagrange interpolation polynomials)**.**
The general polynomial interpolation problem (3.3.2) *admits a unique solution* $p \in \mathcal{P}_n$.

Example 3.3.13. (Generalized Lagrange polynomials for Hermite Interpolation)

double nodes

$t_0 = 0, t_1 = 0, t_2 = 1, t_3 = 1 \quad \Rightarrow \quad n = 3$

(cubic Hermite interpolation).

Explicit formulas for the polynomials $\rightarrow$ see (3.6.2).

$\diamond$

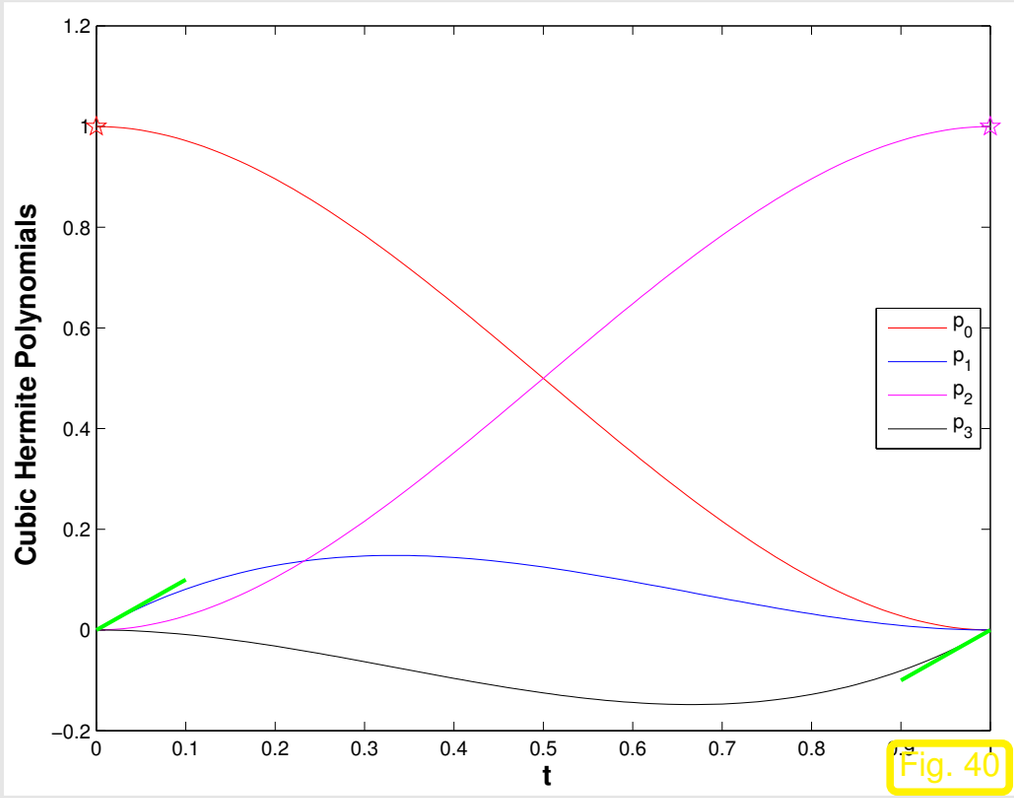


Fig. 40

3.3.2 Conditioning of polynomial interpolation

Sect. 2.5.4 discussed the conditioning/sensitivity of linear systems of equations: This concepts tries to capture how perturbations in the data affect the output of a problem map.

Here: data $\leftrightarrow$ values y_i
problem map $\leftrightarrow$ polynomial interpolation mapping $\mathsf{I}_{\mathcal{T}}$, see Thm. 3.3.8

"The (pointwise) conditioning of interpolation tells us to what extent perturbations in the y-data will affect the values of the interpolating function somewhere else. In the case of ill-conditioning small perturbations in the data can cause big variations in some function values, which is clearly undesirable."

Necessary for studying the conditioning: norms ($\rightarrow$ Def. 2.5.1) on vector space of continuous functions $C(I)$, $I \subset \mathbb{R}$

Note relationship with vector norms $\rightarrow$ Sect. 2.5.1.

supremum norm $\|f\|_{L^\infty(I)} := \sup\{|f(t)|: t \in I\}$, (3.3.14)

L^2-norm $$\|f\|_{L^2(I)}^2 := \int_I |f(t)|^2 \,\mathrm{d}t \;, \tag{3.3.15}$$

L^1-norm $$\|f\|_{L^1(I)} := \int_I |f(t)| \,\mathrm{d}t \;. \tag{3.3.16}$$

Lemma 3.3.17 (Absolute conditioning of polynomial interpolation)**.**
Given a mesh $\mathcal{T} \subset \mathbb{R}$ with generalized Lagrange polynomials L_i, $i = 0, \ldots, n$, and fixed $I \subset \mathbb{R}$, the norm of the interpolation operator satisfies

$$\|\mathrm{I}_{\mathcal{T}}\|_{\infty\to\infty} := \sup_{\mathbf{y}\in\mathbb{K}^{n+1}\setminus\{0\}} \frac{\|\mathrm{I}_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)}}{\|\mathbf{y}\|_\infty} = \left\|\sum_{i=0}^n |L_i|\right\|_{L^\infty(I)} \;, \tag{3.3.18}$$

$$\|\mathrm{I}_{\mathcal{T}}\|_{2\to 2} := \sup_{\mathbf{y}\in\mathbb{K}^{n+1}\setminus\{0\}} \frac{\|\mathrm{I}_{\mathcal{T}}(\mathbf{y})\|_{L^2(I)}}{\|\mathbf{y}\|_2} \le \left(\sum_{i=0}^n \|L_i\|_{L^2(I)}^2\right)^{\frac{1}{2}} \;. \tag{3.3.19}$$

Proof. (for the L^∞-Norm) By $\triangle$-inequality

$$\|\mathrm{I}_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)} = \left\|\sum_{j=0}^n y_j L_j\right\|_{L^\infty(I)} \le \sup_{t\in I} \sum_{j=0}^n |y_j||L_j(t)| \le \|\mathbf{y}\|_\infty \left\|\sum_{i=0}^n |L_i|\right\|_{L^\infty(I)} \;,$$

R. Hiptmair

rev 30401, February 19, 2011

equality in (3.3.18) for $\mathbf{y} := (\operatorname{sgn}(L_j(t^*)))_{j=0}^n$, $t^* := \operatorname{argmax}_{t\in I} \sum_{i=0}^n |L_i(t)|$. □

Proof. (for the L^2-Norm) By $\triangle$-inequality and Cauchy-Schwarz inequality

$$\|\mathrm{I}_{\mathcal{T}}(\mathbf{y})\|_{L^2(I)} \le \sum_{j=0}^n |y_j| \left\|L_j\right\|_{L^2(I)} \le \left(\sum_{j=0}^n |y_j|^2\right)^{\frac{1}{2}} \left(\sum_{j=0}^n \left\|L_j\right\|_{L^2(I)}^2\right)^{\frac{1}{2}} . \quad \square$$

Terminology: Lebesgue constant of $\mathcal{T}$: $\lambda_{\mathcal{T}} := \left\|\sum_{i=0}^n |L_i|\right\|_{L^\infty(I)}$

Remark 3.3.20 (Lebesgue constant for equidistant nodes).

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

Asymptotic estimate (with (3.3.3) and Stirling formula): for $n = 2m$

$$|L_m(1-\tfrac{1}{n})| = \frac{\frac{1}{n}\cdot\frac{1}{n}\cdot\frac{3}{n}\cdot\ldots\cdot\frac{n-3}{n}\cdot\frac{n+1}{n}\cdot\ldots\cdot\frac{2n-1}{n}}{\left(\frac{2}{n}\cdot\frac{4}{n}\cdot\ldots\cdot\frac{n-2}{n}\cdot 1\right)^2} = \frac{(2n)!}{(n-1)2^{2n}((n/2)!)^2 n!} \sim \frac{2^{n+3/2}}{\pi\,(n-1)\,n}$$

Theory [8]: for uniformly spaced nodes $\boxed{\lambda_{\mathcal{T}} \geq Ce^{n/2}}$ for $C > 0$ independent of n. △

Example 3.3.21 (Oscillating interpolation polynomial: Runge's counterexample).

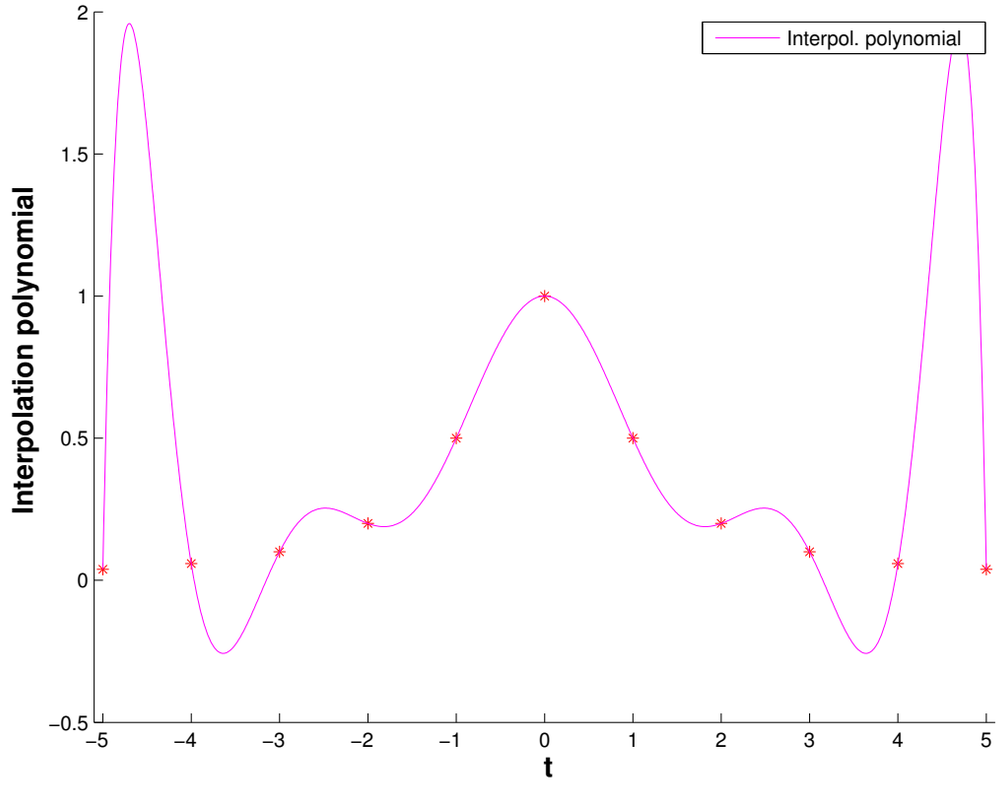
The potentially vary bad conditioning of polynomial interpolation in the case of equidistant nodes allows tremendous oscillations of the interpolating polynomial between the nodes:

Interpolation polynomial with uniformly spaced nodes:

$$\mathcal{T} := \left\{ -5 + \frac{10}{n}\, j \right\}_{j=0}^{n} ,$$

$$y_j = \frac{1}{1+t_j^2} , \quad j = 0, \ldots\ n.$$

Plot $n = 10$ ➙



See example 9.1.5.

R. Hiptmair

rev 30401, February 19, 2011

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

◇

3.4 Polynomial Interpolation: Algorithms

Given: nodes $\mathcal{T} := \{-\infty < t_0 < t_1 < \ldots < t_n < \infty\}$,
values $\boldsymbol{y} := \{y_0, y_1, \ldots, y_n\}$,

define: $p := \mathsf{I}_{\mathcal{T}}(\boldsymbol{y})$ as the unique Lagrange interpolation polynomial given by Theorem 3.3.6.

3.4.1 Multiple evaluations

Task: For *fixed* set $\{t_0, \ldots, t_n\}$ of nodes
effcient evaluation of $p(x_k)$ for $p \in \mathcal{P}_n$ interpolating in (t_i, y_i^k), $i = 0, \ldots, n$, $k = 1, \ldots, N$, $N \gg 1$.

```
1 class PolyInterp {
2   private:
3   // various internal data describing p
4   public:
5     // Constructor taking node vector (t_0,...,t_n) as argument
6     PolyInterp(const vector<double> &t);
7     // Evaluation operator at x for data (y_0,...,y_n)
8     double eval(double x,const vector<double> &y) const;
9 };
```

- Interpolation with Lagrange polynomials (3.3.3) , (3.3.5) is not efficient: $O(n^2)$ operations for every invocation of `eval(x,y)`.

- More efficient formula:

Simple manipulations:

$$p(t) = \sum_{i=0}^{n} L_i(t)\, y_i = \sum_{i=0}^{n} \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t-t_j}{t_i-t_j}\, y_i = \sum_{i=0}^{n} \lambda_i \prod_{\substack{j=0 \\ j\neq i}}^{n} (t-t_j)\, y_i = \prod_{j=0}^{n}(t-t_j) \cdot \sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}\, y_i \,.$$

with $\lambda_i = \dfrac{1}{(t_i-t_0)\cdots(t_i-t_{i-1})(t_i-t_{i+1})\cdots(t_i-t_n)}$, $i = 0,\ldots,n$.

From above formula, with $p(t) \equiv 1$, $y_i = 1$:

$$1 = \prod_{j=0}^{n}(t-t_j) \sum_{i=0}^{n} \frac{\lambda_i}{t-t_i} \quad \Rightarrow \quad \prod_{j=0}^{n}(t-t_j) = \frac{1}{\sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}}$$

R. Hiptmair

rev 30401, February 19, 2011

Barycentric interpolation formula
$$p(t) = \frac{\sum_{i=0}^{n} \frac{\lambda_i}{t-t_i} y_i}{\sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}} \,. \qquad (3.4.1)$$

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

R. Hiptmair

rev 30401, February 19, 2011

```
11 % Compute quotient of weighted sums of λ_i/(t−t_i), effort O(n)
12   z = (x(i)-t); j = find(z == 0);
13   if (~isempty(j)), p(i) = y(j); % avoid division by zero
14   else
15     mu = lambda./z; p(i) = sum(mu.*y)/sum(mu);
16   end
17 end
```

tic-toc-computational time, Matlab polyval vs. barycentric formula → Ex. 3.4.5.

3.4.2 Single evaluation [10, Sect. 8.2.2]

Task: evaluation of p in few points, **Aitken-Neville scheme**

Given: nodes $\mathcal{T} := \{t_j\}_{j=0}^n \subset \mathbb{R}$, pairwise different, $t_i \neq t_j$ for $i \neq j$,
values $y_0, \ldots, y_n$,
one evaluation point $t \in \mathbb{R}$.

For $\{i_0, \ldots, i_m\} \subset \{0, \ldots, n\}$, $0 \leq m \leq n$:

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

recursive definition:

$$p_i(t) \equiv y_i \ , \qquad i = 0, \ldots, n \ ,$$
$$p_{i_0,\ldots,i_m}(t) = \frac{(t - t_{i_0})p_{i_1,\ldots,i_m}(t) - (t - t_{i_m})p_{i_0,\ldots,i_{m-1}}(t)}{t_{i_m} - t_{i_0}} \ . \tag{3.4.3}$$

R. Hiptmair

rev 30401, February 19, 2011

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

```
1 function v = ANipoleval(t,y,x)
2 for i=1:length(y)
3   for k=i-1:-1:1
4     y(k) =
        y(k+1)+(y(k+1)-y(k))*...
5             (x-t(i))/(t(i)-t(k));
6   end
7 end
8 v = y(1);
```

```
1  class PolyEval {
2    private:
3      // evaluation point and various internal data describing the polynomials
4    public:
5      // Constructor taking the evaluation point as argument
6      PolyEval(double x);
7      // Add another data point and update internal information
8      void addPoint(t,y);
9      // Value of current interpolating polynomial at x
10     double eval(void) const;
11 };
```

Example 3.4.5 (Timing polynomial evaluations).

R. Hiptmair

rev 30401, February 19, 2011

Comparison of the computational time needed for polynomial interpolation of

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

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

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

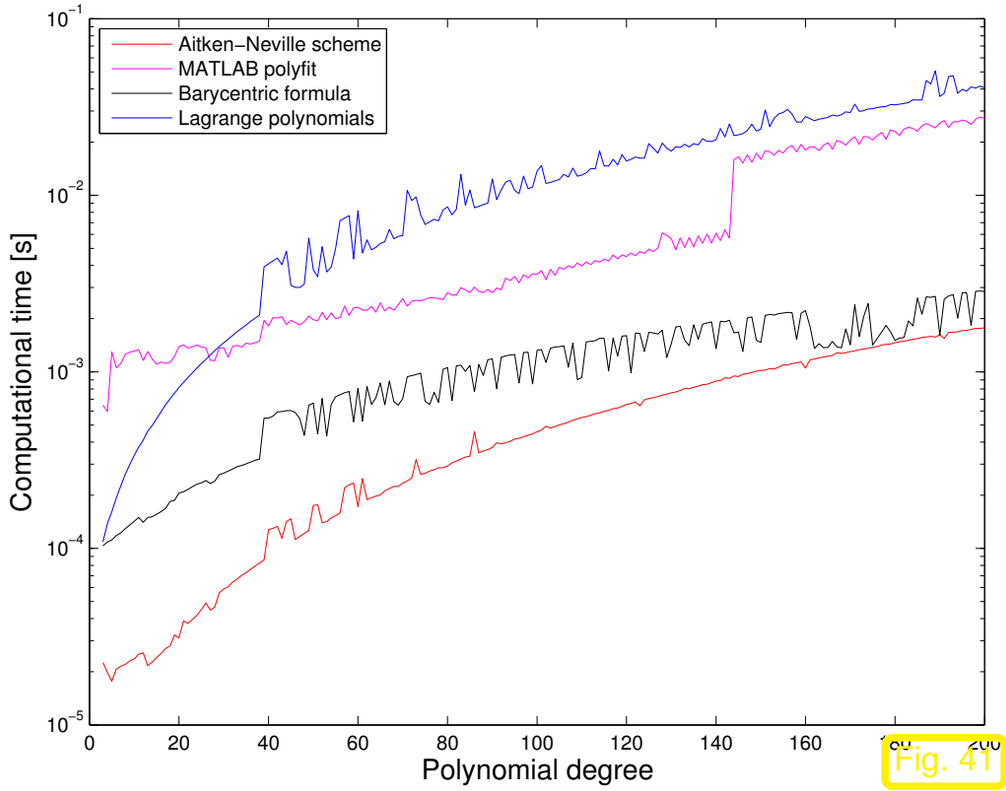


Fig. 41

R. Hiptmair

rev 30401, February 19, 2011

Code 3.4.6: Timing polynomial evaluations

```
time=zeros(1,4);
f=@(x) sqrt(x);                    % function to interpolate:
for k=1:100
  res = [];
  for n=3:1:200      % n = increasing polynomial degree
    t = (1:n);      y = f(t);       x=n*rand;
    tic;   v1 = ANipoleval(t,y,x);          time(1) = toc;
    tic;   v2 = ipoleval(t,y,x);            time(2) = toc;
    tic;   v3 = intpolyval(t,y,x);          time(3) = toc;
    tic;   v4 = intpolyval_lag(t,y,x);      time(4) = toc;
    res = [res; n,time];
  end
  if (k == 1), finres = res;
  else, finres = min(finres,res); end
end
figure
semilogy(finres(:,1),finres(:,2),'r-',finres(:,1),finres(:,3),'m-',
  finres(:,1),finres(:,4),'k-', finres(:,1),finres(:,5),'b-');
xlabel('Polynomial degree','Fontsize',14);
ylabel('Computational time [s]','Fontsize',14);
legend('Aitken-Neville scheme','MATLAB polyfit', 'Barycentric formula',
  'Lagrange polynomials',2);
```

This uses functions given in Code 3.4.1, Code 3.4.3 and the MATLAB function `polyfit` (with a clearly greater computational effort !)

Code 3.4.7: MATLAB polynomial evaluation using built-in function `polyfit`

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

Code 3.4.8: 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

Extrapolation is the same as interpolation but the evaluation point t is outside the interval $[\inf_{j=0,\ldots,n} t_j, \sup_{j=0,\ldots,n} t_j]$. W.l.o.g. assume $t = 0$, $t_i > 0$.

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 $|t| \ll 1$.

The extrapolation technique introduced below works well, if

- f is an *even function* of its argument: $f(t) = f(-t)$,
- f behaves "nicely" around $t = 0$.

Rigorous: existence of an asymptotic expansion in h^2

$$f(h) = f(0) + A_1 h^2 + A_2 h^4 + \cdots + A_n h^{2n} + R(h) \quad , \quad A_k \in \mathbb{K} \, ,$$

with remainder estimate $|R(h)| = O(h^{2n+2})$ for $h \to 0$.

R. Hiptmair

rev 30401, February 19, 2011

Idea:

① evaluation of $f(t_i)$ for different t_i, $i = 0, \ldots, n$, $|t_i| > 0$.

② $f(0) \approx p(0)$ with interpolation polynomial $p \in \mathcal{P}_n$, $p(t_i) = f(t_i)$.

Example 3.4.9 (Numeric differentiation through extrapolation).

Given: smooth function $f : I \subset \mathbb{R} \mapsto \mathbb{R}$ in procedural form: `function y = f(x)`

Sought: (approximation of) $f'(x)$, $x \in I$.

Natural idea: approximation of derivative by (symmetric) difference quotient

$$f(x) \approx \frac{f(x+h) - f(x-h)}{h} \,. \tag{3.4.10}$$

 straightforward implementation:

MATLAB-CODE : Numeric differentiation through finite differences & relative errors.

```
x=1.1; h=2.^[-1:-5:-36];
atanerr = abs(dirnumdiff(atan,x,h)-1/(1+x^2))*(1+x^2);
sqrterr = abs(dirnumdiff(sqrt,x,h)-1/(2*sqrt(x)))*(2*sqrt(x));
experr = abs(dirnumdiff(exp,x,h)-exp(x))/exp(x);

function[df]=dirnumdiff(f,x,h)
df=(f(x+h)-f(x))./h;
end
```


R. Hiptmair

rev 30401, February 19, 2011

$f(x) = \arctan(x)$

h	Relative error
2^{-1}	0.20786640808609
2^{-6}	0.00773341103991
2^{-11}	0.00024299312415
2^{-16}	0.00000759482296
2^{-21}	0.00000023712637
2^{-26}	0.00000001020730
2^{-31}	0.00000005960464
2^{-36}	0.00000679016113

$f(x) = \sqrt{x}$

h	Relative error
2^{-1}	0.09340033543136
2^{-6}	0.00352613693103
2^{-11}	0.00011094838842
2^{-16}	0.00000346787667
2^{-21}	0.00000010812198
2^{-26}	0.00000001923506
2^{-31}	0.00000001202188
2^{-36}	0.00000198842224

$f(x) = \exp(x)$

h	Relative error
2^{-1}	0.29744254140026
2^{-6}	0.00785334954789
2^{-11}	0.00024418036620
2^{-16}	0.00000762943394
2^{-21}	0.00000023835113
2^{-26}	0.00000000429331
2^{-31}	0.00000012467100
2^{-36}	0.00000495453865

Effect of roundoff errors: $h \to 0$ does not achieve arbitrarily high accuracy. Rather, fewer correct digits for very small h! (This is due to a an effect called cancellation) that haunts the denomimator of the difference quotient. More detailed discussion in [10, Sect. 8.2.6].

Extrapolation offers a numerically stable ($\to$ Sect. 2.5.2) alternative, because for a $2(n+1)$-times continuously differentiable function $f : D \subset \mathbb{R} \mapsto \mathbb{R}$, $x \in D$ (Taylor sum in x with Lagrange residual)

$$T(h) := \frac{f(x+h) - f(x-h)}{2h} \sim f'(x) + \sum_{k=1}^{n} \frac{1}{(2k)!} \frac{\mathrm{d}^{2k} f}{\mathrm{d}x^{2k}}(x)\, h^{2k} + \frac{1}{(2n+2)!} f^{(2n+2)}(\xi(x)) \; .$$

R. Hiptmair

rev 30401, February 19, 2011

Code 3.4.11: 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

advantage: guaranteed accuracy ➙ efficiency

◇

R. Hiptmair

Additional explanations concerning the problems encountered in the numerical evaluation of difference quotients:

Example 3.4.12 (Roundoff errors and difference quotients). Ex. 3.4.9

Approximate derivative by difference quotient: $f'(x) \approx \frac{f(x+h)-f(x)}{h}$.

Calculus: better approximation for smaller $h > 0$, isn't it **?**

MATLAB-CODE: Numerical differentiation of exp(x)

```
h = 0.1; x = 0.0;
for i = 1:16
    df = (exp(x+h)-exp(x))/h;
    fprintf('%d %16.14f\n',i,df-1);
    h = h*0.1;
end
```

Recorded relative error, $f(x) = e^x$, $x = 0$ ▷

$\log_{10}(h)$	relative error
-1	0.05170918075648
-2	0.00501670841679
-3	0.00050016670838
-4	0.00005000166714
-5	0.00000500000696
-6	0.00000049996218
-7	0.00000004943368
-8	-0.00000000607747
-9	0.00000008274037
-10	0.00000008274037
-11	0.00000008274037
-12	0.00008890058234
-13	-0.00079927783736
-14	-0.00079927783736
-15	0.11022302462516
-16	-1.00000000000000

Note: An analysis based on expressions for remainder terms of Taylor expansions shows that the approximation error cannot be blamed for the loss of accuracy as $h \to 0$ (as expected).

Explanation relying on roundoff error analysis, see Sect. 2.4:

MATLAB-CODE: Numerical differentiation of exp(x)

```
h = 0.1; x = 0.0;
for i = 1:16
    df = (exp(x+h)-exp(x))/h;
    fprintf('%d %16.14f\n',i,df-1);
    h = h*0.1;
end
```

Obvious cancellation $\longrightarrow$ error amplification

$$\left.\begin{array}{l} f'(x) - \dfrac{f(x+h)-f(x)}{h} \to 0 \\ \text{Impact of roundoff} \to \infty \end{array}\right\} \quad \text{for } h \to 0 \,.$$

$\log_{10}(h)$	relative error
-1	0.05170918075648
-2	0.00501670841679
-3	0.00050016670838
-4	0.00005000166714
-5	0.00000500000696
-6	0.00000049996218
-7	0.00000004943368
-8	-0.00000000607747
-9	0.00000008274037
-10	0.00000008274037
-11	0.00000008274037
-12	0.00008890058234
-13	-0.00079927783736
-14	-0.00079927783736
-15	0.11022302462516
-16	-1.00000000000000

Analysis for $f(x) = \exp(x)$:

R. Hiptmair

rev 30401, February 19, 2011

$$\texttt{df} = \frac{e^{x+h}\boxed{(1+\delta_1)} - e^x\boxed{(1+\delta_2)}}{h}$$

correction factors take into account roundoff:
($\rightarrow$ "“axiom of roundoff analysis”, Ass. 2.4.10)

$$= e^x\left(\frac{e^h-1}{h} + \frac{\delta_1 e^h - \delta_2}{h}\right) \qquad |\delta_1|, |\delta_2| \le \texttt{eps}\ .$$

$1+O(h)$ $\qquad O(h^{-1})$ für $h \to 0$

$$\Rightarrow \quad |\texttt{df}| \le e^x\left(\frac{e^h-1}{h} + \texttt{eps}\,\frac{1+e^h}{h}\right)$$

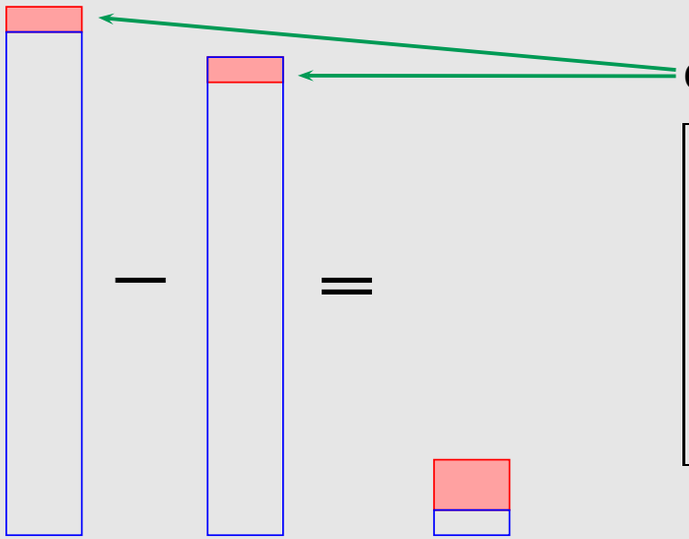
relative error: $\left|\frac{e^x - \texttt{df}}{e^x}\right| \approx h + \frac{2\texttt{eps}}{h} \to \min$ for $h = \sqrt{2\,\texttt{eps}}$.

In double precision: $\sqrt{2\texttt{eps}} = 2.107342425544702 \cdot 10^{-8}$

R. Hiptmair

rev 30401, February 19, 2011

What is this mysterious cancellation (*ger.:* Auslöschung) **?**



Cancellation

$\hat{=}$

Subtraction of almost equal numbers

(➤ extreme amplification of relative errors)

Example 3.4.13 (cancellation in decimal floating point arithmetic).

x, y afflicted with relative errors $\approx 10^{-7}$:

$$\begin{array}{rcll} x &=& 0.123467* & \leftarrow \text{7th digit perturbed} \\ y &=& 0.123456* & \leftarrow \text{7th digit perturbed} \\ \hline x-y &=& 0.000011* = 0.11*000 \cdot 10^{-4} & \leftarrow \text{3rd digit perturbed} \end{array}$$

padded zeroes

◇

R. Hiptmair

rev 30401, February 19, 2011

3.4.4 Newton basis and divided differences [10, Sect. 8.2.4]

Drawback of the Lagrange basis: adding another data point affects *all* basis polynomials!

```
1  class PolyEval {
2    private:
3    // evaluation point and various internal data describing the polynomials
4    public:
5      // Idle Constructor
6      PolyEval();
7      // Add another data point and update internal information
8      void addPoint(t,y);
9      // Evaluation of current interpolating polynomial at x
10     double operator ()  (double x) const;
11 };
```

The challenge: Both `addPoint()` and the evaluation operator may be called many times and the implementation has to remain efficient under these circumstances.

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

$$N_0(t) := 1\,, \quad N_1(t) := (t - t_0)\,, \quad \ldots \quad , \quad N_n(t) := \prod_{i=0}^{n-1} (t - t_i)\,. \tag{3.4.14}$$

Note: $N_n \in \mathcal{P}_n$ with *leading coefficient* 1.

$\succ$ LSE for polynomial interpolation problem in Newton basis:

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

$\Leftrightarrow$ triangular linear system

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

Solution of the system with forward substitution:

$$
\begin{aligned}
a_0 &= y_0 \ , \\
a_1 &= \frac{y_1 - a_0}{t_1 - t_0} = \frac{y_1 - y_0}{t_1 - t_0} \ , \\
a_2 &= \frac{y_2 - a_0 - (t_2 - t_0)a_1}{(t_2 - t_0)(t_2 - t_1)} = \frac{y_2 - y_0 - (t_2 - t_0)\frac{y_1 - y_0}{t_1 - t_0}}{(t_2 - t_0)(t_2 - t_1)} = \frac{\frac{y_2 - y_0}{t_2 - t_0} - \frac{y_1 - y_0}{t_1 - t_0}}{t_2 - t_1} \ , \\
&\vdots
\end{aligned}
$$

Observation: same quantities computed again and again !

R. Hiptmair

rev 30401, February 19, 2011

In order to find a better algorithm, we turn to a new interpretation of the coefficients a_j of the interpolating polynomials in Newton basis.

Newton basis polynomial $N_j(t)$: degree j and leading coefficient 1

$\Rightarrow$ a_j is the leading coefficient of the interpolating polynomial $p_{0,\ldots,j}$

(notation $p_{0,\ldots,j}$ introduced in Sect. 3.4.2, see (3.4.3))

➢ Recursion (3.4.3) implies recursion for leading coefficients $a_{\ell,\ldots,m}$ of interpolating polynomials $p_{\ell,\ldots,m}$, $0 \le \ell \le m \le n$:

$$a_{\ell,\ldots,m} = \frac{a_{\ell+1,\ldots,m} - a_{\ell,\ldots,m-1}}{t_m - t_\ell} \,.$$

Simpler and more efficient algorithm using divided differences:

$$\begin{aligned} y[t_i] &= y_i \\ y[t_i,\ldots,t_{i+k}] &= \frac{y[t_{i+1},\ldots,t_{i+k}] - y[t_i,\ldots,t_{i+k-1}]}{t_{i+k} - t_i} \end{aligned} \quad \text{(recursion)} \qquad (3.4.15)$$

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

$$\begin{array}{c|ccccc} t_0 & y[t_0] & & & & \\ & & > & y[t_0,t_1] & & \\ t_1 & y[t_1] & & > & y[t_0,t_1,t_2] & \\ & & > & y[t_1,t_2] & & > \; y[t_0,t_1,t_2,t_3], \\ t_2 & y[t_2] & & > & y[t_1,t_2,t_3] & \\ & & > & y[t_2,t_3] & & \\ t_3 & y[t_3] & & & & \end{array} \qquad (3.4.16)$$

R. Hiptmair

rev 30401, February 19, 2011

the elements are computed from left to right, every "$>$" means recursion (3.4.15).

If a new datum (t_{n+1}, y_{n+1}) is added, it is enough to compute $n+2$ new terms

$$y[t_{n+1}],\ y[t_n, t_{n+1}],\ \ldots, y[t_0, \ldots, t_{n+1}].$$

Code 3.4.17: 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
```

By derivation: computed finite differences are the coefficients of interpolating polynomials in Newton basis:

$$p(t) = a_0 + a_1(t-t_0) + a_2(t-t_0)(t-t_1) + \cdots + a_n \prod_{j=0}^{n-1}(t-t_j) \tag{3.4.18}$$

R. Hiptmair

rev 30401, February 19, 2011

$$a_0 = y[t_0],\ a_1 = y[t_0,t_1],\ a_2 = y[t_0,t_1,t_2],\ \ldots.$$

"Backward evaluation" of $p(t)$ in the spirit of Horner's scheme (→ Rem. 3.2.4, [10, Alg. 8.20]):

$$p \leftarrow a_n, \quad p \leftarrow (t - t_{n-1})p + a_{n-1}, \quad p \leftarrow (t - t_{n-2})p + a_{n-2}, \quad \ldots.$$

Code 3.4.19: Divided differences evaluation by modified Horner scheme

```
1 function p = evaldivdiff(t,y,x)
2 dd=divdiff(t,y); % Compute divided differences, see Code 3.4.16
3 n = length(y)-1;
4 p=dd(n+1);
5 for j=n:-1:1, p = (x-t(j)).*p+dd(j); end
```


Computational effort:

- $O(n^2)$ for computation of divided differences,
- $O(n)$ for every single evaluation of $p(t)$.

Example 3.4.20 (Class `PolyEval`).

Implementation of a C++ class supporting the efficient update and evaluation of an interpolating polynomial making use of

- presentation in Newton basis (3.4.14),
- computation of representation coefficients through divided difference scheme (3.4.16), see Code 3.4.16,
- evaluation by means of Horner scheme, see Code 3.4.18.

Code 3.4.21: “Update friendly” polynomial interpolant

```
1  #ifndef NUMCSE_POLYEVAL_HPP
2  #define NUMCSE_POLYEVAL_HPP
3  #include <vector>
4  #include <iostream>
5  #include <cassert>
6  namespace numcse
7  {
8    class PolyEval {
9    private:
10     std::vector<double> t;   // Interpolation nodes
11     std::vector<double> y;   // Coefficients in Newton representation
12   public:
```

```
13    PolyEval(); // Idle constructor
14    void addPoint(double t, double y); // Add another data point
15    // evaluate value of current interpolating polynomial at x,
16    double operator() (double x) const;
17  private:
18    // Update internal representation, called by addPoint()
19    void divdiff();
20  };
21
22  PolyEval::PolyEval() {}
23
24  void PolyEval::addPoint(double td, double yd)
25  { t.push_back(td); y.push_back(yd); divdiff(); }
26
27  void PolyEval::divdiff() {
28    int n = t.size();
29    for(int j=0; j<n-1; j++) y[n-1] =
        ((y[n-1]-y[j])/(t[n-1]-t[j]));
30  }
31
32  double PolyEval::operator() (double x) const {
33    double s = y.back();
34    for(int i = y.size()-2; i>= 0; --i) s = s*(x-t[i])+y[i];
35    return s;
```

```
36      }
37  }
38  #endif
```


◇

Remark 3.4.22 (Divided differences and derivatives).

If $y_0, \ldots, y_n$ are the values of a smooth function f in the points $t_0, \ldots, t_n$, that is, $y_j := f(t_j)$, then

$$y[t_i, \ldots, t_{i+k}] = \frac{f^{(k)}(\xi)}{k!}$$

for a certain $\xi \in [t_i, t_{i+k}]$, see [10, Thm. 8.21].

△

3.5 Shape preserving interpolation

When reconstructing a quantitative dependence of quantities from measurements, first principles from physics often stipulate qualitative constraints, which translate into *shape properties* of the function f, e.g., when modelling the material law for a gas:

$$t_i \text{ pressure values, } y_i \text{ densities} \quad \succ \quad f \text{ positive \& monotone.}$$

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

R. Hiptmair

rev 30401, February 19, 2011

Example 3.5.1 (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

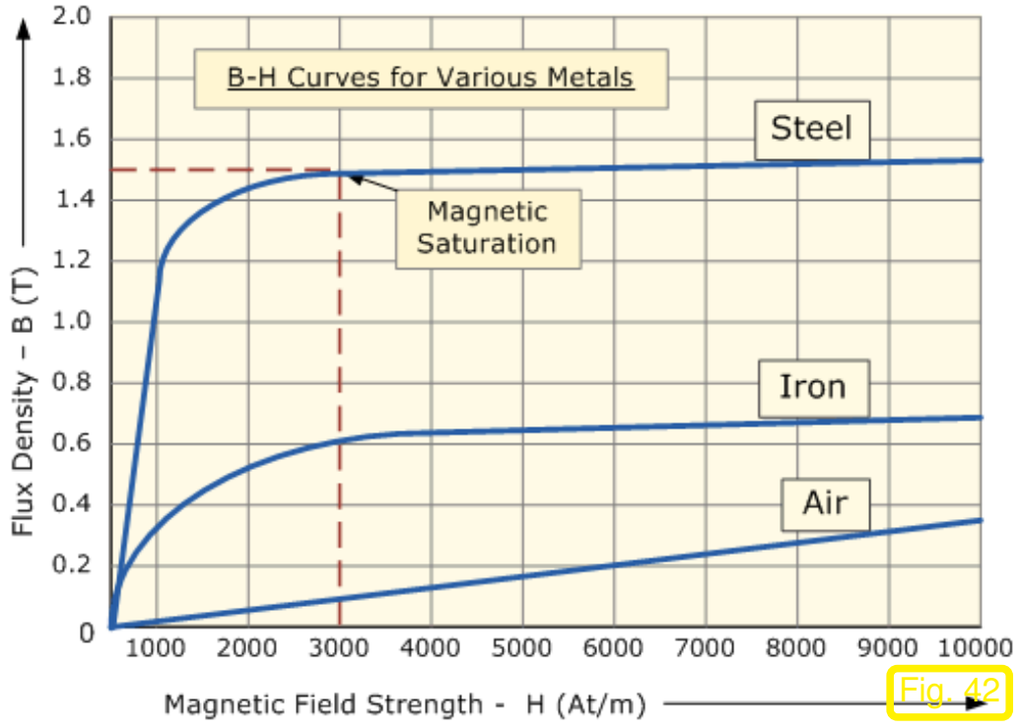


Fig. 42

◇

R. Hiptmair

rev 30401, February 19, 2011

Definition 3.5.2 (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 3.5.3 (Convex/concave data)**.**

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

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

Mathematical characterization of convex data:

$$y_i \leq \frac{(t_{i+1} - t_i)y_{i-1} + (t_i - t_{i-1})y_{i+1}}{t_{i+1} - t_{i-1}} \qquad \forall\, i = 1, \ldots, n-1,$$

i.e., each data point lies below the line segment connecting the other data, *cf.* definition of convexity of a function [52, Def. 5.5.2].

R. Hiptmair

rev 30401, February 19, 2011

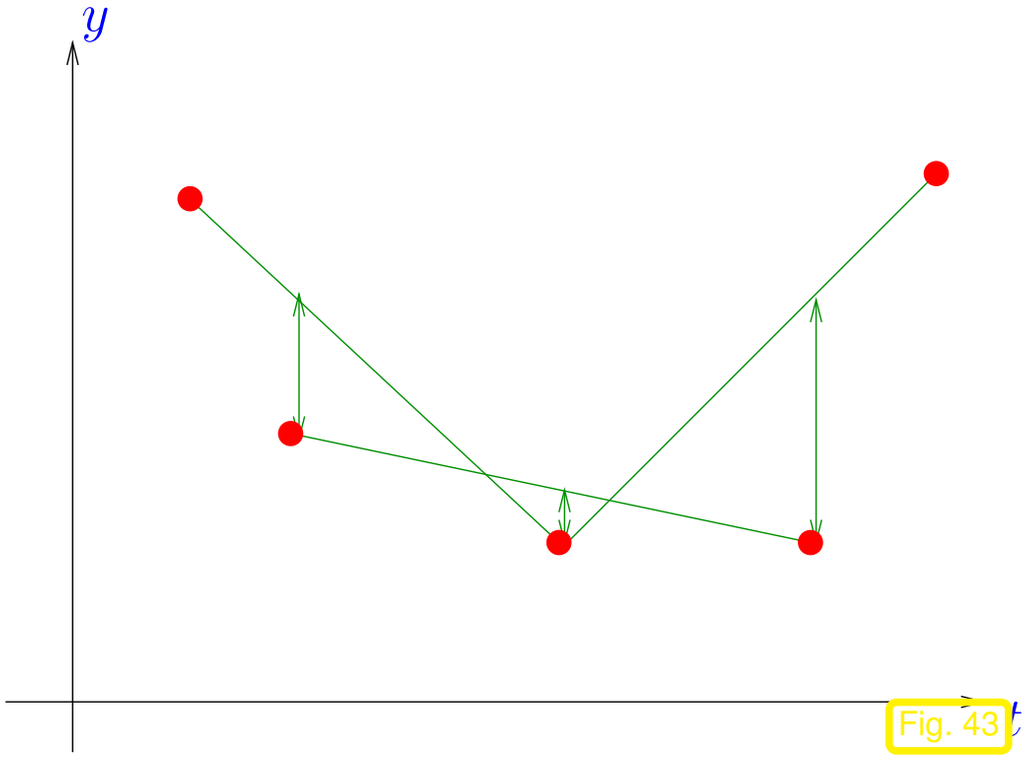


Fig. 43

Convex data

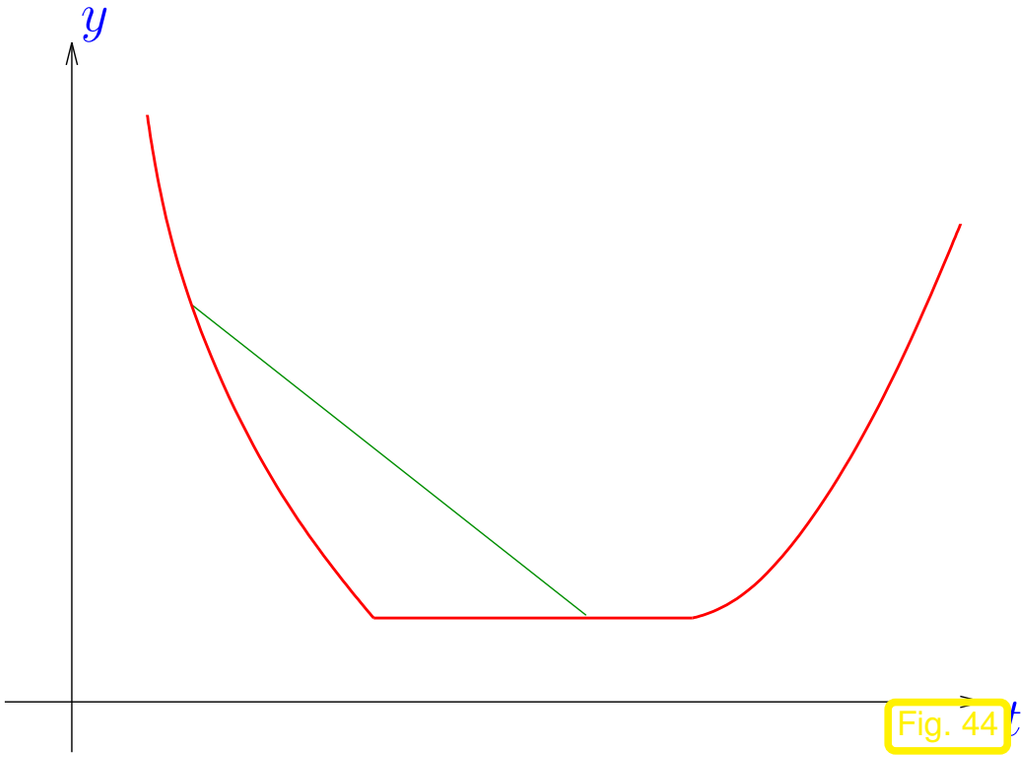


Fig. 44

Convex function

Definition 3.5.4 (Convex/concave function)**.** *[52, Def. 5.5.2]* →

$$f : I \subset \mathbb{R} \mapsto \mathbb{R} \quad \begin{matrix} \textit{convex} \\ \textit{concave} \end{matrix} \quad :\Leftrightarrow \quad \begin{matrix} f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y) \\ f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda) f(y) \end{matrix} \quad \begin{matrix} \forall\, 0 \leq \lambda \leq 1 \\ \forall\, x, y \in I \,. \end{matrix},$$

R. Hiptmair

rev 30401, February 19, 2011

Now consider interpolation: data $(t_i, y_i),\ i = 0, \ldots, n$ $\longrightarrow$ 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 3.5.5 (Bad behavior of global polynomial interpolants).

Positive and monotonic data:

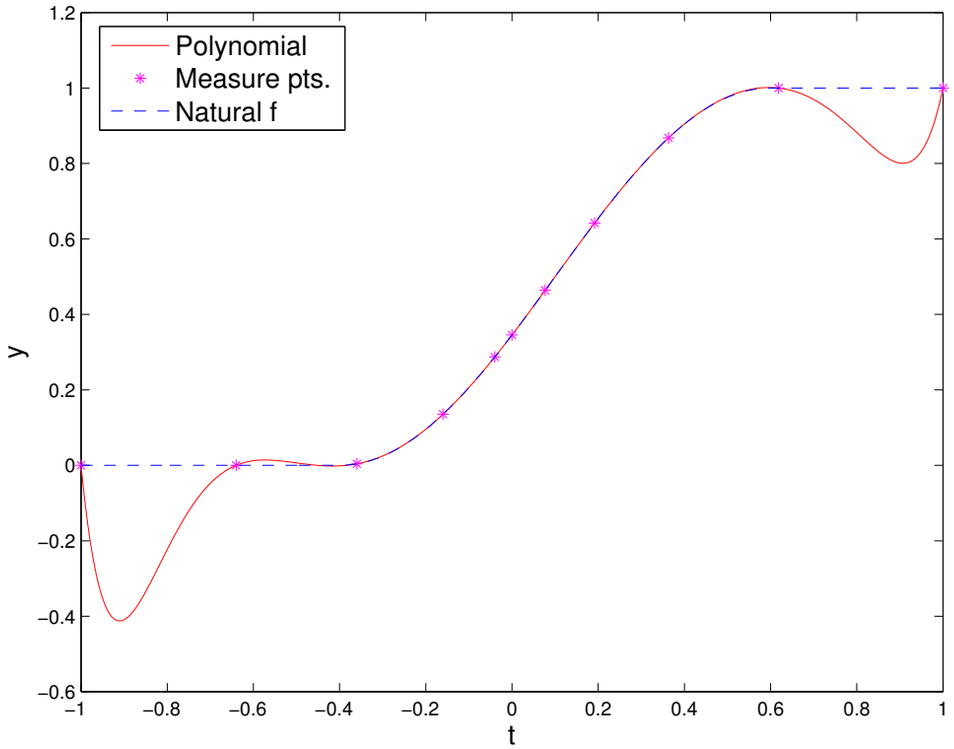
t_i	-1.0000	-0.6400	-0.3600	-0.1600	-0.0400	0.0000	0.0770	0.1918	0.3631	0.6187	1.0000
y_i	0.0000	0.0000	0.0039	0.1355	0.2871	0.3455	0.4639	0.6422	0.8678	1.0000	1.0000

R. Hiptmair

rev 30401, February 19, 2011

created by taking points on the graph of

$$f(t) = \begin{cases} 0 & \text{if } t < -\frac{2}{5}\,, \\ \frac{1}{2}(1+\cos(\pi(t-\frac{3}{5}))) & \text{if } -\frac{2}{5} < t < \frac{3}{5}\,, \\ 1 & \text{otherwise.} \end{cases}$$



← Interpolating polynomial, degree = 10

Oscillations at the endpoints of the interval (see Ex. 9.1.5)

- No locality
- No positivity
- No monotonicity
- No local conservation of the curvature

◇

R. Hiptmair

rev 30401, February 19, 2011

3.5.1 Piecewise linear interpolation

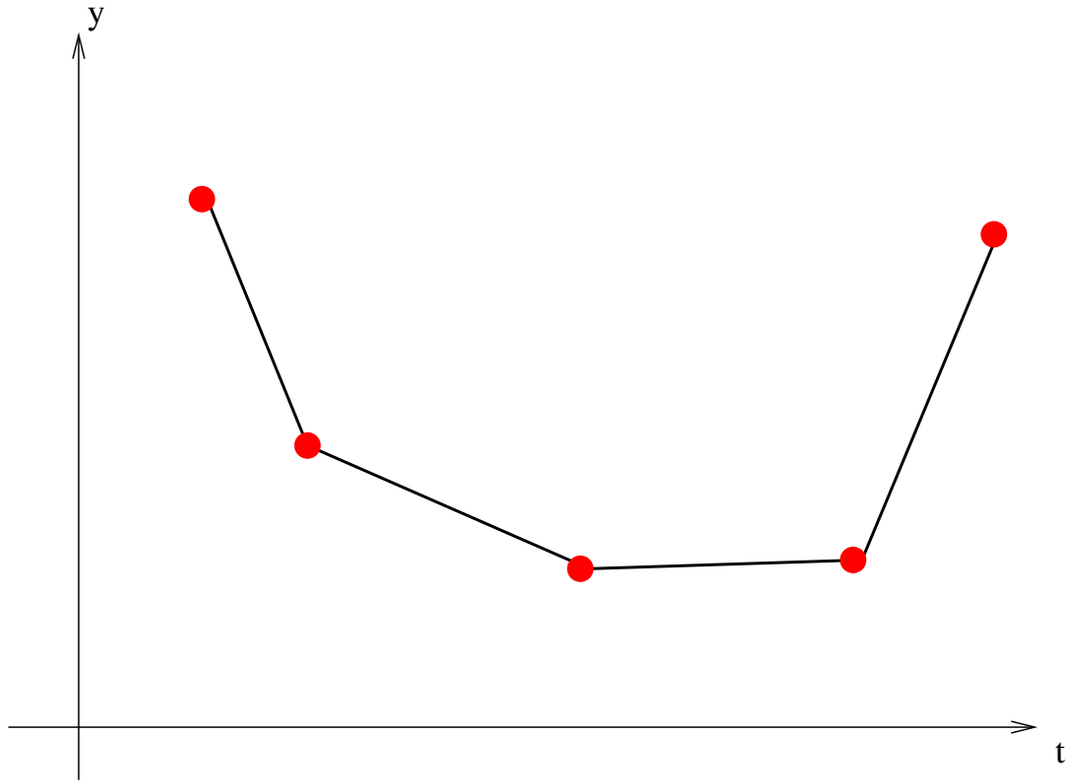
There is a very simple method of achieving perfect shape preservation by means of a linear (→ Rem. 3.1.5) interpolation operator into the space of continuous functions:

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

Piecewise linear interpolant:

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

Piecewise linear interpolant of data in Fig. 43:



Piecewise linear interpolation means simply “connect the data points in $\mathbb{R}^2$ using straight lines”.

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

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

Theorem 3.5.6 (Local shape preservation by piecewise linear interpolation)**.**
Let $s \in C([t_0, t_n])$ be the piecewise linear interpolant of $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \ldots, n$, for every subinterval $I = [t_j, t_k] \subset [t_0, t_n]$:

if $(t_i, y_i)\big\|_I$ are positive/negative	$\Rightarrow$	*$s\big\|_I$ is positive/negative,*
if $(t_i, y_i)\big\|_I$ are monotonic (increasing/decreasing)	$\Rightarrow$	*$s\big\|_I$ is monotonic (increasing/decreasing),*
if $(t_i, y_i)\big\|_I$ are convex/concave	$\Rightarrow$	*$s\big\|_I$ is convex/concave.*

Local shape preservation **=** perfect shape preservation!

Bad news: none of this properties carries over to local polynomial interpolation of higher polynomial degree $d > 1$.

Example 3.5.7 (Piecewise quadratic interpolation)*.*

Known (Thm. 3.3.6): Parabola (polynomial of degree 2) uniquely determined by 3 data points ➢ form groups of three adjacent data points.

Assume: $n = 2m$ even

piecewise quadratic interpolant $q : [\min\{t_i\}, \max\{t_i\}] \mapsto \mathbb{R}$ defined by

$$q_j := q_{|[t_{2j-2}, t_{2j}]} \in \mathcal{P}_2 \quad , \quad q_j(t_i) = y_i \ , \quad i = 2j-2, 2j-1, 2j \ , \quad j = 1, \ldots, m \ . \qquad (3.5.8)$$

Nodes as in Ex. 3.5.5

Piecewise linear/quadratic interpolation ▷

No shape preservation for piecewise quadratic interpolant

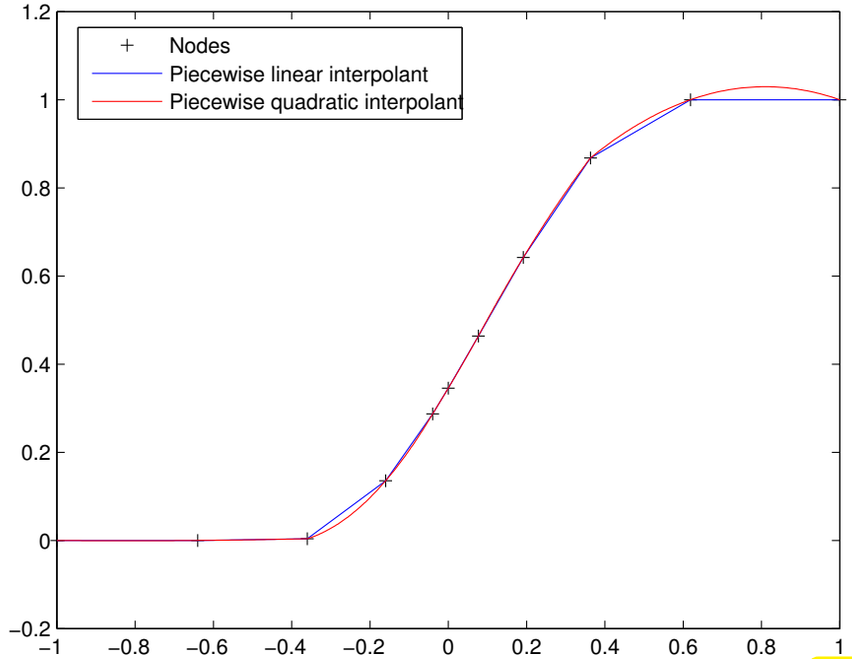


Fig. 45

◇

The “only” drawback of piecewise linear interpolation:

interpolant is only C^0 but not C^1 (no continuous derivative).

R. Hiptmair

rev 30401, October 21, 2010

3.6 Cubic Hermite Interpolation

Aim: construct local shape-preserving ($\rightarrow$ Sect. 3.5) (linear ?) interpolation operator that fixes shortcoming of piecewise linear interpolation by ensuring C^1-smoothness of the interpolant.

✎ notation: $C^1([a,b]) \hat{=}$ space of *continuously differentiable* functions $[a,b] \mapsto \mathbb{R}$.

3.6.1 Definition and algorithms

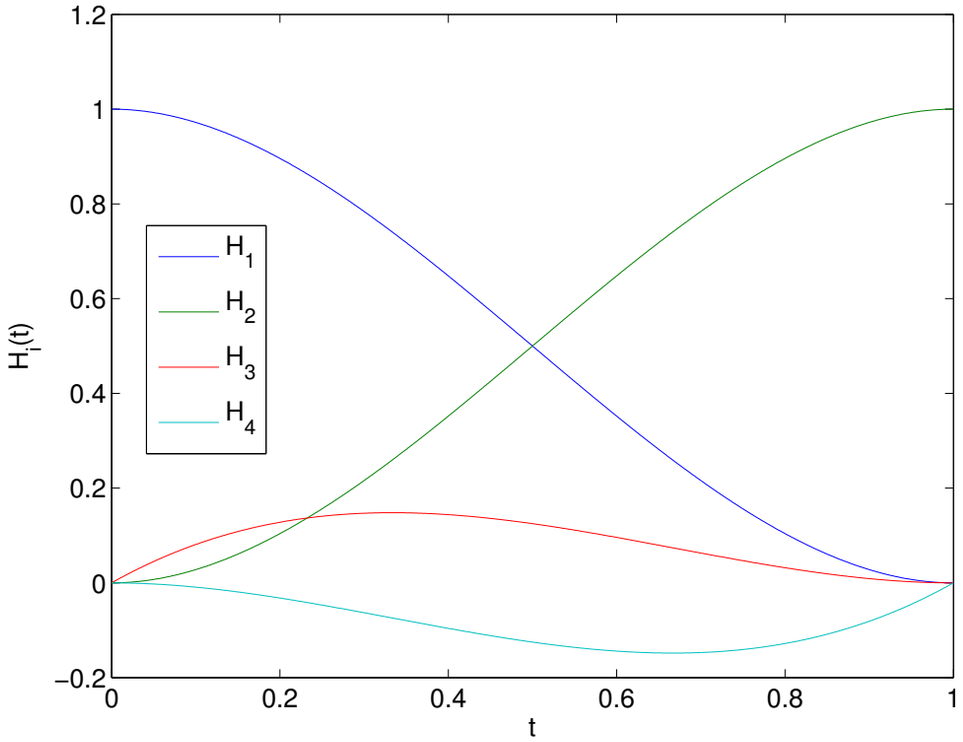
Given: mesh points $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \ldots, n$, $t_0 < t_1 < \cdots < t_n$

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

➙ Piecewise cubic Hermite interpolation polynomial $s \in C^1([t_0,t_n])$:
with given slopes $c_i \in \mathbb{R}$, $i = 0,\ldots,n$

$$s_{|[t_{i-1},t_i]} \in \mathcal{P}_3\,, \quad i = 1,\ldots,n \quad , \quad \boxed{s(t_i) = y_i}\,, \quad , \quad \boxed{s'(t_i) = c_i}\,, \quad i = 0,\ldots,n\,.$$

◥ $$s(t) = y_{i-1}H_1(t) + y_iH_2(t) + c_{i-1}H_3(t) + c_iH_4(t) \quad , \quad t \in [t_{i-1},t_i]\,, \tag{3.6.1}$$



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

(3.6.2)

◁ Local basis polynomial on $[0,1]$

R. Hiptmair

rev 30401, October 21, 2010

Piecewise cubic polynomial s on t_1, t_2 with $s(t_1) = y_1$, $s(t_2) = y_2$, $s'(t_1) = c_1$, $s'(t_2) = c_2$:

efficient local evaluation ➙

Code 3.6.3: Local evaluation of cubic Hermite polynomial

```
function
  s=hermloceval(t,t1,t2,y1,y2,c1,c2)
% y1, y2:  data values, c1, c2:  slopes
h = t2-t1; t = (t-t1)/h;
a1 = y2-y1; a2 = a1-h*c1;
a3 = h*c2-a1-a2;
s = y1+(a1+(a2+a3*t).*(t-1)).*t;
```

How to choose the slopes c_i **?**

Natural attempt: average of local slopes:

$$c_i = \begin{cases} \Delta_1 & \text{, for } i = 0 \text{ ,} \\ \Delta_n & \text{, for } i = n \text{ ,} \\ \frac{t_{i+1}-t_i}{t_{i+1}-t_{i-1}}\Delta_i + \frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}}\Delta_{i+1} & \text{, if } 1 \le i < n \text{ .} \end{cases} \quad , \quad \Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}} \text{ ,} j = 1, \ldots, n \text{ .} \tag{3.6.4}$$

➙ Linear local interpolation operator

R. Hiptmair

rev 30401, October 21, 2010

Example 3.6.5 (Piecewise cubic Hermite interpolation).

Data points:

- 11 equispaced nodes

  $t_j = -1 + 0.2\,j, \quad j = 0, \ldots, 10.$

  in the interval $I = [-1, 1]$,
- $y_i = f(t_i)$ with

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

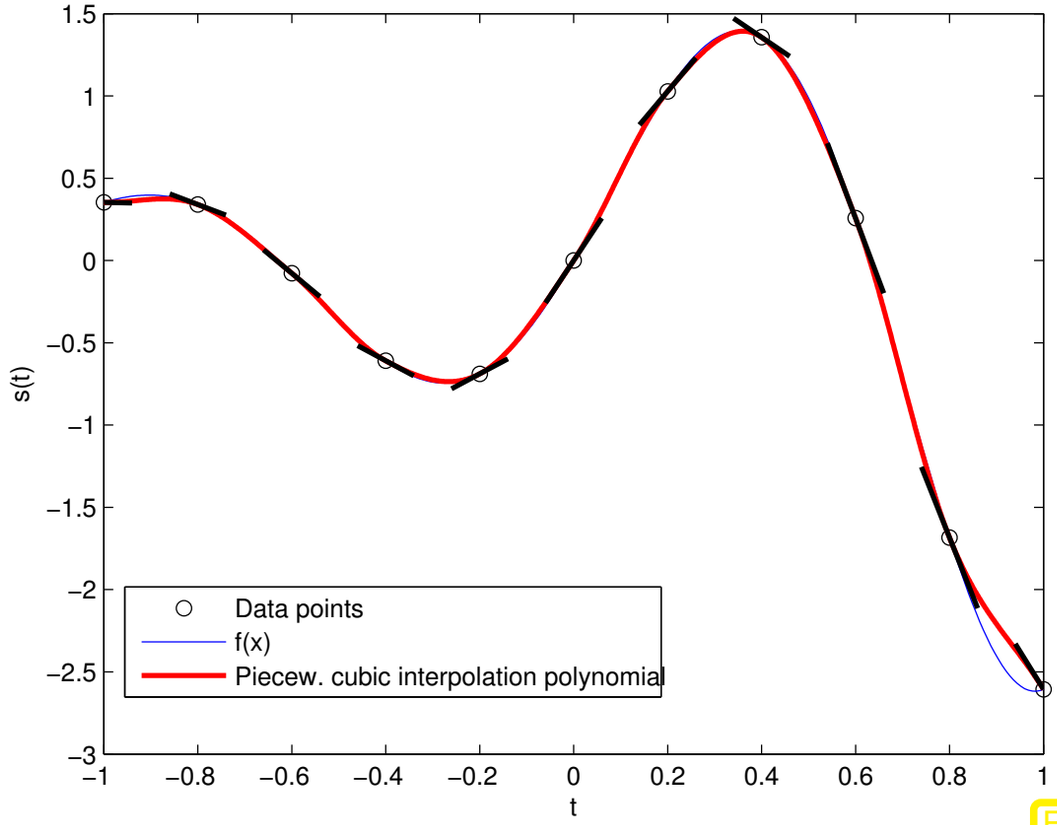


Fig. 46

Use of weighted averages of slopes as in (3.6.4).

See Code 3.6.5.

No preservation of monotonicity!

R. Hiptmair

rev 30401, October 21, 2010

Code 3.6.6: Piecewise cubic Hermite interpolation

```
function hermintp1(f,t)
% compute and plot the cubic Hermite interpolant of the function f in the nodes t
% using weighted averages according to (3.6.4) as local slopes
n = length(t); h = diff(t); % computes lengths of intervals between nodes: h_i := t_{i+1} - t_i
y = f(t); % f must support collective evaluation for row vector argument
delta = diff(y)./h; % slopes of piecewise linear interpolant
c = [delta(1),...
     ((h(2:end).*delta(1:end-1)+h(1:end-1).*delta(2:end))...
        ./(t(3:end) - t(1:end-2)) ),...
     delta(end)]; % slopes from weighted average, see (3.6.4)

figure('Name','Hermite Interpolation');
plot(t,y,'ko'); hold on;  % plot data points
fplot(f,[t(1), t(n)]);
for j=1:n-1  % compute and plot the Hermite interpolant with slopes c
  vx = linspace(t(j),t(j+1), 100);
  plot(vx,hermloceval(vx,t(j),t(j+1),y(j),y(j+1),c(j),c(j+1)),'r-',
    'LineWidth',2);
end
for j=2:n-1  % plot segments indicating the slopes c_i
```

R. Hiptmair

rev 30401, October 21, 2010

```
20    plot([t(j)-0.3*h(j-1),t(j)+0.3*h(j)],...
21         [y(j)-0.3*h(j-1)*c(j),y(j)+0.3*h(j)*c(j)],'k-','LineWidth',2)
22  end
23  plot([t(1),t(1)+0.3*h(1)],[y(1),y(1)+0.3*h(1)*c(1)],'k-',
    'LineWidth',2);
24  plot([t(end)-0.3*h(end),t(end)],[y(end)-0.3*h(end)*c(end),y(end)],'k
    'LineWidth',2);
25  xlabel('t'); ylabel('s(t)');
26  legend('Data points','f(x)','Piecew. cubic interpolation
    polynomial ');
27  hold off;
```

Invocation: `hermintp1( @(x)sin(5*x).*exp(x), [-1:0.2:1]);`

3.6.2 Shape preserving Hermite interpolation

Slopes according to (3.6.4) ➢ Hermite interpolation does not preserve monotonicity.

Remedy: choice of the slopes c_i via "limiter" $\rightarrow$ conservation of monotonicity.

$$c_i = \begin{cases} 0 & \text{, if } \operatorname{sgn}(\Delta_i) \neq \operatorname{sgn}(\Delta_{i+1}) \text{ ,} \\ \text{weighted average of } \Delta_i, \Delta_{i+1} & \text{otherwise} \end{cases} \quad , \quad i = 1, \ldots, n-1 \text{ .} \tag{3.6.7}$$

Which kind of average **?**

$$c_i = \frac{1}{\frac{w_a}{\Delta_i} + \frac{w_b}{\Delta_{i+1}}} \tag{3.6.8}$$

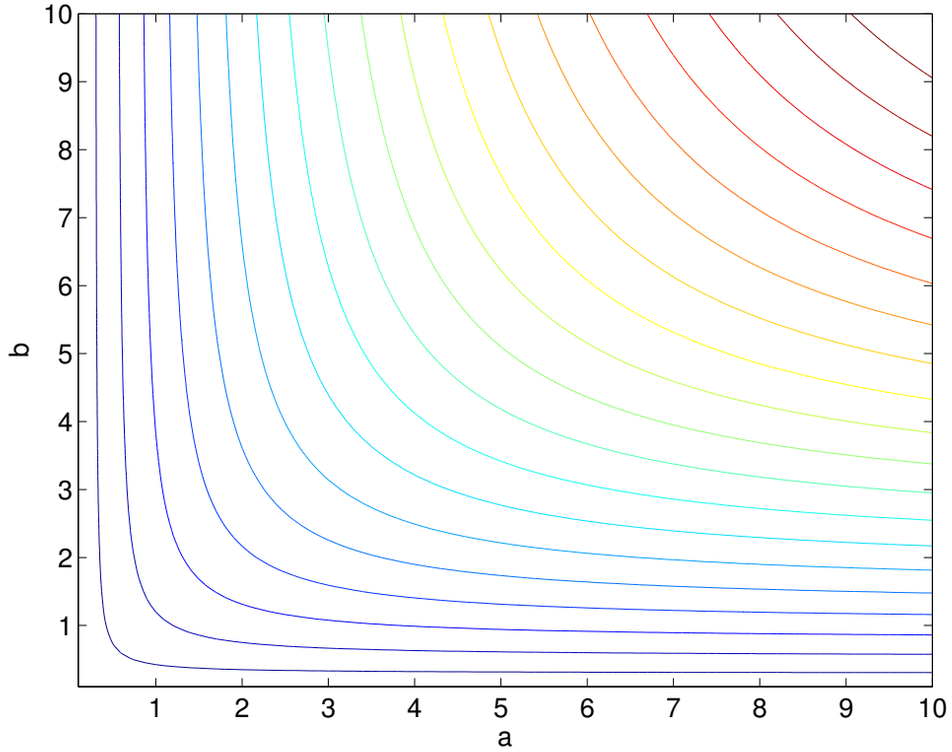
= weighted harmonic mean of the slopes with weights w_a, w_b, $(w_a + w_b = 1)$.

R. Hiptmair

rev 30401, October 21, 2010

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:

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

$$\operatorname{sgn}(\Delta_1) = \operatorname{sgn}(\Delta_2) \;\Rightarrow\; c_i = \begin{cases} \Delta_1 & \text{, if } i = 0 \text{ ,} \\ \dfrac{3(h_{i+1}+h_i)}{\frac{2h_{i+1}+h_i}{\Delta_i} + \frac{2h_i+h_{i+1}}{\Delta_{i+1}}} & \text{, for } i \in \{1, \ldots, n-1\} \text{ ,} \\ \Delta_n & \text{, if } i = n \text{ ,} \end{cases} \quad h_i := t_i - t_{i-1} \text{ .} \tag{3.6.9}$$

Example 3.6.10 (Monotonicity preserving piecewise cubic polynomial interpolation).

Data from ex. 3.5.5

MATLAB-function:

```
v = pchip(t,y,x);
```

`t`: Sampling points

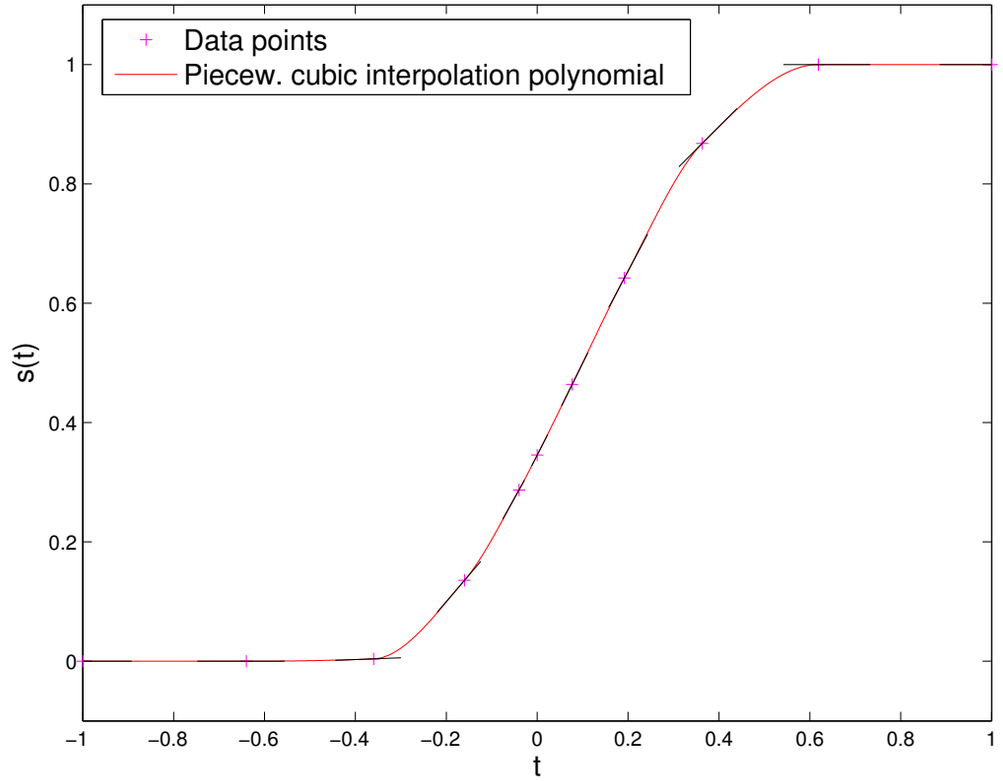
`y`: Sampling values

`x`: Evaluation points

`v`: Vector $s(x_i)$

Local interpolation operator

! Non linear interpolation operator



Theorem 3.6.11 (Monotonicity preservation of limited cubic Hermite interpolation)**.**
The cubic Hermite interpolation polynomial with slopes as in (3.6.9) *provides a* local monotonicity-preserving C^1*-interpolant.*

Proof. See F. FRITSCH UND R. CARLSON, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal., 17 (1980), S. 238–246. □

Remark 3.6.12 (Non-linear interpolation).

The monotonicity preserving cubic Hermite interpolation is non-linear !

Obviously, (3.1.5) is not satisfied, because linearity is clearly violated by the slope formulas (3.6.7), (3.6.9).

△

Calculation of the c_i in `pchip` (details in [17]):

Code 3.6.13: Monotonicity preserving slopes in pchip

```
1 function c = pchipslopes(t,y)
2 % Calculation of local slopes c_i for shape preserving cubic Hermite
    interpolation, see (3.6.7), (3.6.9)
3 % t, y are row vectors passing the data points
```

```
n = length(t); h = diff(t); delta = diff(y)./h; % linear slopes
c = zeros(size(h));
k = find(sign(delta(1:n-2)).*sign(delta(2:n-1))>0)+1;
w1 = 2*h(k)+h(k-1); w2 = h(k)+2*h(k-1);
c(k) = (w1+w2)./(w1./delta(k-1) + w2./delta(k));
% Special slopes at endpoints, beyond (3.6.9)
c(1) = pchipend(h(1),h(2),delta(1),delta(2));
c(n) = pchipend(h(n-1),h(n-2),delta(n-1),delta(n-2));

function d = pchipend(h1,h2,del1,del2)
% Noncentered, shape-preserving, three-point formula.
d = ((2*h1+h2)*del1 - h1*del2)/(h1+h2);
if sign(d) ~= sign(del1), d = 0;
elseif (sign(del1)~=sign(del2))(abs(d)>abs(3*del1)), d = 3*del1;
  end
```


R. Hiptmair

rev 30401, October 27, 2010

3.7 Splines [10, Ch. 9]

Definition 3.7.1 (Spline space).

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

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

$C^{d-1}(I)$: $d-1$-times continuosly differentiable

$\mathcal{P}_d$: locally *polynomial of degree* d

Spline spaces mapped onto each other by differentiation & integration:

$$s \in \mathcal{S}_{d,\mathcal{M}} \quad \Rightarrow \quad s' \in \mathcal{S}_{d-1,\mathcal{M}} \quad \wedge \quad \int_a^t s(\tau)\,\mathrm{d}\tau \in \mathcal{S}_{d+1,\mathcal{M}} \; .$$

R. Hiptmair

rev 30401, October 27, 2010

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

Dimension of spline space by counting argument (heuristic):

$$\dim \mathcal{S}_{d,\mathcal{M}} = n \cdot \dim \mathcal{P}_d - \#\{C^{d-1} \text{ continuity constraints}\} = n \cdot (d+1) - (n-1) \cdot d = n + d \ .$$

Note special case: interpolation in $\mathcal{S}_{1,\mathcal{M}}$ = piecewise linear interpolation.

3.7.1 Cubic spline interpolation [29, XIII, 46]

R. Hiptmair

rev 30401, October 27, 2010

Cognitive psychology: C^2-functions are perceived as "smooth".

➙ C^2-spline interpolants $\leftrightarrow$ $d = 3$ received special attention in CAD.

Another special case: cubic spline interpolation, $d = 3$ (related to Hermite interpolation, Sect. 3.6)

Task: Given mesh $\mathcal{M} := \{t_0 < t_1 < \cdots < t_n\}$, $n \in \mathbb{N}$, "find" cubic spline $s \in \mathcal{S}_{3,\mathcal{M}}$ such that

$$s(t_j) = y_j \quad , \quad j = 0, \ldots, n \; . \tag{3.7.2}$$

$\hat{=}$ interpolation at knots !

From dimensional considerations it is clear that the interpolation conditions will fail to fix the interpolating cubic spline uniquely:

$$\dim \mathcal{S}_{3,\mathcal{M}} - \#\{\text{interpolation conditions}\} = (n+3) - (n+1) = 2 \text{ free d.o.f.}$$

R. Hiptmair

rev 30401, October 27, 2010

“two conditions are missing”

s

Reuse representation through cubic Hermite basis polynomials from (3.6.2):

(3.6.1)

$$s_{|[t_{j-1},t_j]}(t) = \begin{array}{ll} s(t_{j-1}) & \cdot(1-3\tau^2+2\tau^3)+ \\ s(t_j) & \cdot(3\tau^2-2\tau^3)+ \\ h_j s'(t_{j-1}) & \cdot(\tau-2\tau^2+\tau^3)+ \\ h_j s'(t_j) & \cdot(-\tau^2+\tau^3)\ , \end{array} \tag{3.7.3}$$

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

➢ Task of cubic spline interpolation boils down to finding slopes $s'(t_j)$ in nodes of the mesh.

Once these slopes are known, the efficient local evaluation of a cubic spline function can be done as for a cubic Hermite interpolant, see Sect. 3.6.1, Code 3.6.2.

Note: if $s(t_j)$, $s'(t_j)$, $j = 0, \ldots, n$, are fixed, then the representation (3.7.3) already guarantees $s \in C^1([t_0, t_n])$, *cf.* the discussion for cubic Hermite interpolation, Sect. 3.6.

➢ only continuity of s''
- has to be enforced by choice of $s'(t_j)$
- ⇕
- will yield extra conditions to fix the $s'(t_j)$

However, do the

- interpolation conditions (3.7.2) $s(t_j) = y_j, \quad j = 0, \ldots, n$, and the
- regularity constraint $s \in C^2([t_0, t_n])$

uniquely determine the unknown slopes $c_j := s'(t_j)$ **?**

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

$$s''_{|[t_{j-1},t_j]}(t_j) = s''_{|[t_j,t_{j+1}]}(t_j) \, , \quad j = 1, \ldots, n-1 \, . \tag{3.7.4}$$

Based on (3.7.3), we express (3.7.4) in concrete terms, using

$$s''_{|[t_{j-1},t_j]}(t) = s(t_{j-1})h_j^{-2}6(-1+2\tau) + s(t_j)h_j^{-2}6(1-2\tau) + h_j^{-1}s'(t_{j-1})(-4+6\tau) + h_j^{-1}s'(t_j)(-2+6\tau) \;, \tag{3.7.5}$$

which can be obtained by the chain rule and from $\frac{d\tau}{dt} = h_j^{-1}$.

$$\overset{(3.7.5)}{\Rightarrow} \quad s''_{|[t_{j-1},t_j]}(t_{j-1}) = -6\cdot s(t_{j-1})h_j^{-2} + 6\cdot s(t_j)h_j^{-2} - 4\cdot h_j^{-1}s'(t_{j-1}) - 2\cdot h_j^{-1}s'(t_j) \;,$$
$$s''_{|[t_{j-1},t_j]}(t_j) = 6\cdot s(t_{j-1})h_j^{-2} + -6\cdot s(t_j)h_j^{-2} + 2\cdot h_j^{-1}s'(t_{j-1}) + 4\cdot h_j^{-1}s'(t_j) \;.$$

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

$$\frac{1}{h_j}c_{j-1} + \left(\frac{2}{h_j} + \frac{2}{h_{j+1}}\right)c_j + \frac{1}{h_{j+1}}c_{j+1} = 3\left(\frac{y_j - y_{j-1}}{h_j^2} + \frac{y_{j+1}-y_j}{h_{j+1}^2}\right) \;, \tag{3.7.6}$$

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

R. Hiptmair

rev 30401, October 27, 2010

(3.7.6) $\Leftrightarrow$ *undetermined* $(n-1)\times(n+1)$ linear system of equations

$$\begin{pmatrix} b_0 & a_1 & b_1 & 0 & \cdots & & \cdots & 0 \\ 0 & b_1 & a_2 & b_2 & & & & \\ & 0 & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & a_{n-2} & b_{n-2} & 0 \\ 0 & \cdots & & \cdots & 0 & b_{n-2} & a_{n-1} & b_{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} 3\left(\frac{y_1-y_0}{h_1^2}+\frac{y_2-y_1}{h_2^2}\right) \\ \vdots \\ 3\left(\frac{y_{n-1}-y_{n-2}}{h_{n-1}^2}+\frac{y_n-y_{n-1}}{h_n^2}\right) \end{pmatrix} . \quad (3.7.7)$$

➙ 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: $s''(t_0)=s''(t_n)=0$

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

R. Hiptmair

rev 30401, October 27, 2010

➤ Linear system of equations with tridiagonal s.p.d. ($\rightarrow$ Def. 2.7.9, Lemma 2.7.12) coefficient matrix $\rightarrow$ $c_0, \ldots, c_n$

Thm. 2.6.31 $\Rightarrow$ computational effort for the solution $= O(n)$

③ Periodic cubic spline interpolation: $s'(t_0) = s'(t_n)$, $s''(t_0) = s''(t_n)$

$n \times n$-linear system with s.p.d. coefficient matrix

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

Solved with rank-1-modifications technique (see Section 2.9.0.1, Lemma 2.9.8) + tridiagonal elimination, computational effort $O(n)$

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

Notice analogies and differences:

		extra degrees of freedom fixed by:
1. piecewise polynomial interpolant, $d = 3$,	piecewise cubic polynomials that match the data (t_i, y_i)	1. intermediate nodes,
2. Hermite interpolant,		2. slopes,
3. cubic spline,		3. C^2-constraint, complete/natural/periodic constraint.

Structural properties of cubic spline interpolants

Remark 3.7.8 (Extremal properties of natural cubic spline interpolants).

For $f : [a,b] \mapsto \mathbb{R}$, $f \in C^2([a,b])$: $\frac{1}{2}\int_a^b |f''(t)|^2 \,\mathrm{d}t$ = elastic bending energy.

On the grid $\mathcal{M} := \{a = t_0 < t_1 < \cdots < t_n = b\}$: $s \in \mathcal{S}_{3,\mathcal{M}}$ = natural cubic spline interpolant of $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \ldots, n$.

For every $k \in C^2([t_0,t_n])$ satisfying $k(t_i) = 0$, $i = 0, \ldots, n$:

$$h(\lambda) := \tfrac{1}{2}\int_a^b |s'' + \lambda k''|^2 \,\mathrm{d}t \quad \Rightarrow \quad \frac{\mathrm{d}h}{\mathrm{d}\lambda}\Big|_{\lambda=0} = \int_a^b s''(t)k''(t)\,\mathrm{d}t = \sum_{j=1}^{n}\int_{t_{j-1}}^{t_j} s''(t)k''(t)\,\mathrm{d}t$$

Two times integration by parts, $s^{(4)} \equiv 0$:

$$\frac{\mathrm{d}h}{\mathrm{d}\lambda}\Big|_{\lambda=0} = -\sum_{j=1}^{n}\Big(s'''(t_j^-)\underbrace{k(t_j)}_{=0} - s'''(t_{j-1}^+)\underbrace{k(t_{j-1})}_{=0}\Big) + \underbrace{s''(t_n)}_{=0}k'(t_n) - \underbrace{s''(t_0)}_{=0}k'(t_0) = 0\,.$$

R. Hiptmair

rev 30401, October 27, 2010

Theorem 3.7.9 (Optimality of natural cubic spline interpolant).
The natural cubic spline interpolant minimizes the elastic curvature energy among all interpolating functions in $C^2([a,b])$.

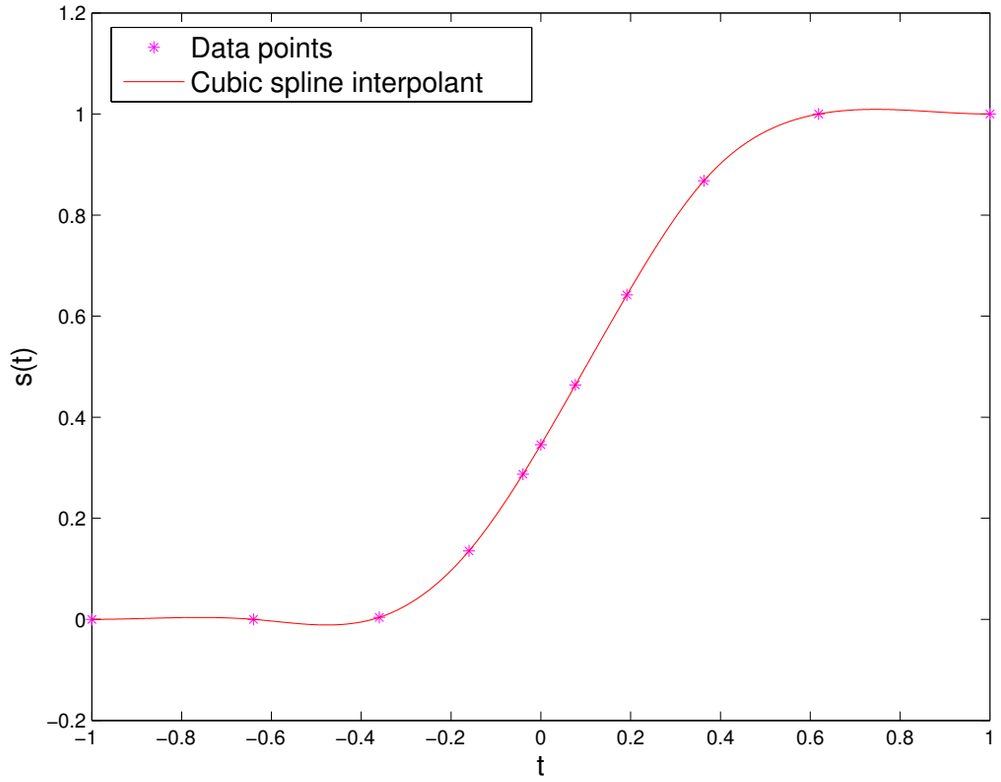
△

Remark 3.7.10 (Shape preservation).
Data $s(t_j) = y_j$ from Ex. 3.5.5 and

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

The cubic spline interpolant is not monotonicity- or curvature-preserving
(cubic spline interpolation is linear!)

△



R. Hiptmair

rev 30401, October 27, 2010

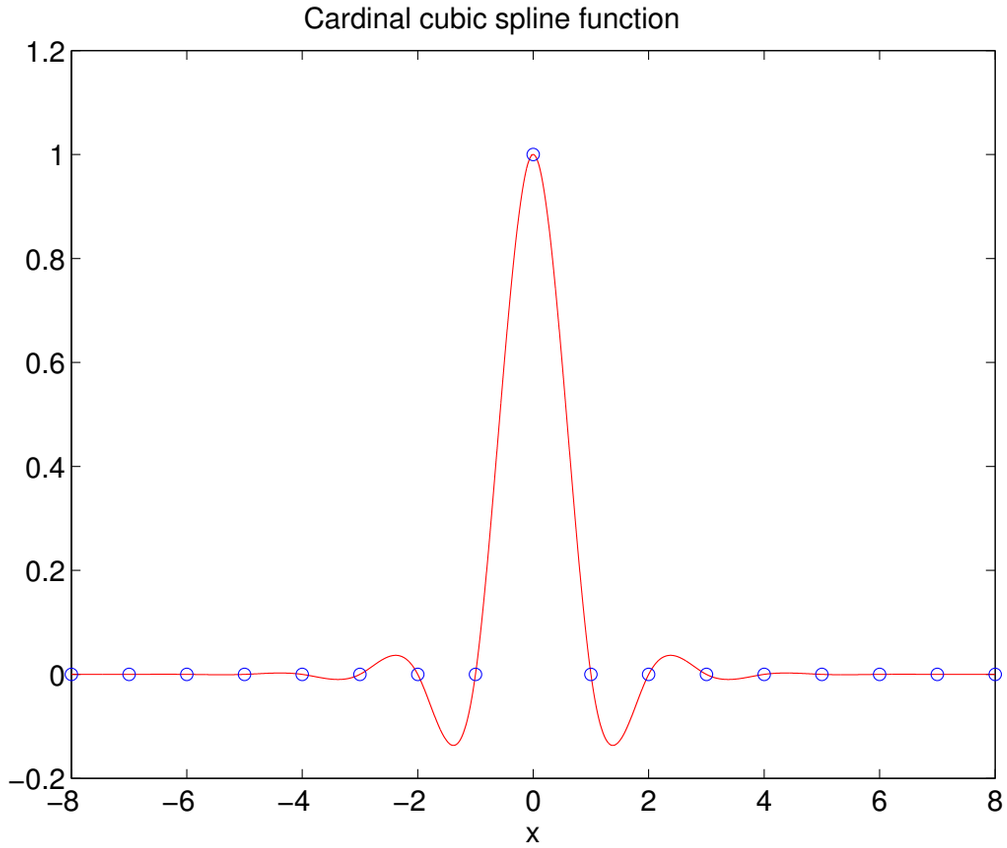
Example 3.7.11 (Locality of the natural cubic spline interpolation).

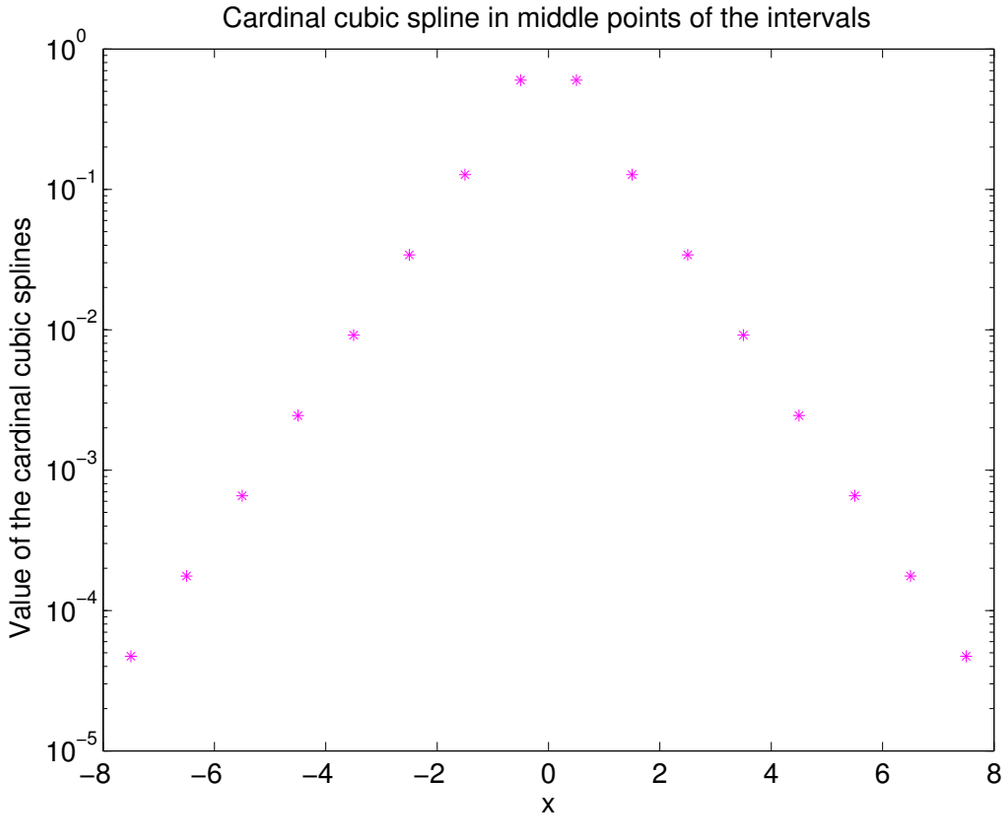
Given a grid $\mathcal{M} := \{t_0 < t_1 < \cdots < t_n\}$ the ith natural cardinal spline is defined as

$$L_i \in \mathcal{S}_{3,\mathcal{M}}, \quad L_i(t_j) = \delta_{ij} \ , \quad L_i''(t_0) = L_i''(t_n) = 0 \ .$$

$$\text{Natural spline interpolant:} \quad s(t) = \sum_{j=0}^{n} y_j L_j(t) \ .$$

Decay of L_i $\leftrightarrow$ locality of the cubic spline interpolation.





Exponential decay of the cardinal splines → cubic spline interpolation is “almost local”

◇

3.7.2 Shape Preserving Spline Interpolation

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

Sought:
- knot set $\mathcal{M} \subset [t_0, t_n]$ ($\rightarrow$ Def 3.7.1),
- an interpolating quadratic spline function $s \in \mathcal{S}_{2,\mathcal{M}}$, $s(t_i) = y_i$, $i = 0, \ldots, n$

that preserves the "shape" of the data ($\rightarrow$ Sect. 3.5)

Notice that $\mathcal{M} \neq \{t_j\}_{j=0}^n$: s interpolates the data in the points t_i but is piecewise polynomial on $\mathcal{M}$!

We proceed in four steps:

① "Shape-faithful" choice of slopes c_i, $i = 0, \ldots, n$ [35, 41], analogous to Sect. 3.6.2

Recall (3.6.7) and (3.6.9): we fix the slopes c_i in the nodes using the harmonic mean of data slopes Δ_j, the final interpolant will be tangent to these segments in the points (t_i, y_i). If (t_i, y_i) is a local maximum or minimum of the data, c_j is set to zero.

$$\text{Limiter} \quad c_i := \begin{cases} \dfrac{2}{\Delta_i^{-1}+\Delta_{i+1}^{-1}} & \text{if } \operatorname{sign}(\Delta_i) = \operatorname{sign}(\Delta_{i+1}) \text{ ,} \\ 0 & \text{otherwise,} \end{cases} \qquad i = 1, \ldots, n-1 \text{ .}$$

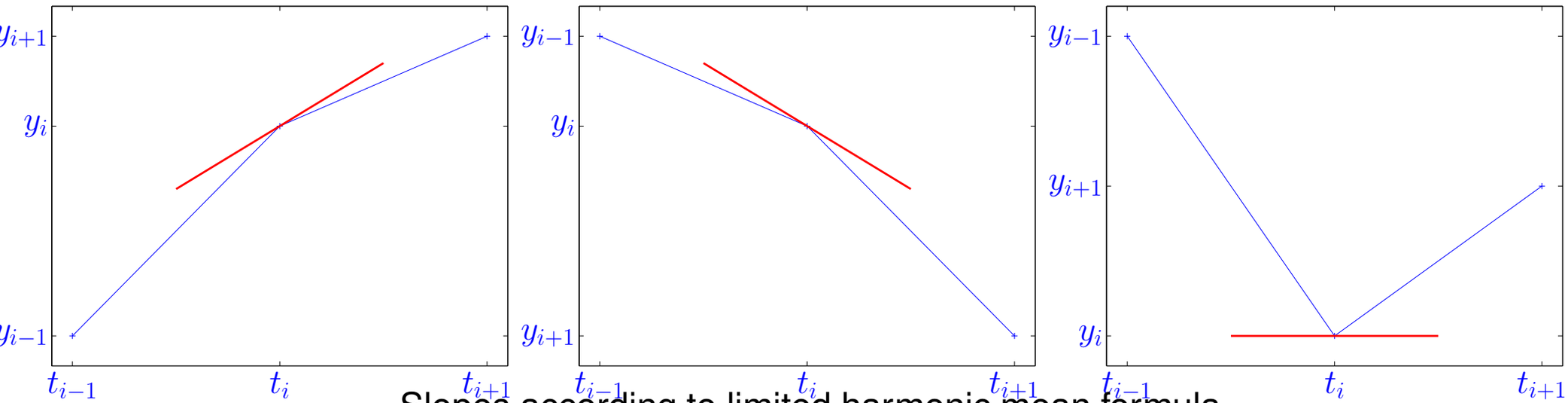
$$c_0 := 2\Delta_1 - c_1 \text{ ,} \qquad c_n := 2\Delta_n - c_{n-1} \text{ ,}$$

where $\Delta_j = \frac{y_j - y_{j-1}}{t_j - t_{j-1}}$. If signs of slopes agree, then choose

$$c_i = \frac{2}{\Delta_i^{-1} + \Delta_{i+1}^{-1}}.$$

R. Hiptmair

rev 30401, October 3, 2010



Slopes according to limited harmonic mean formula

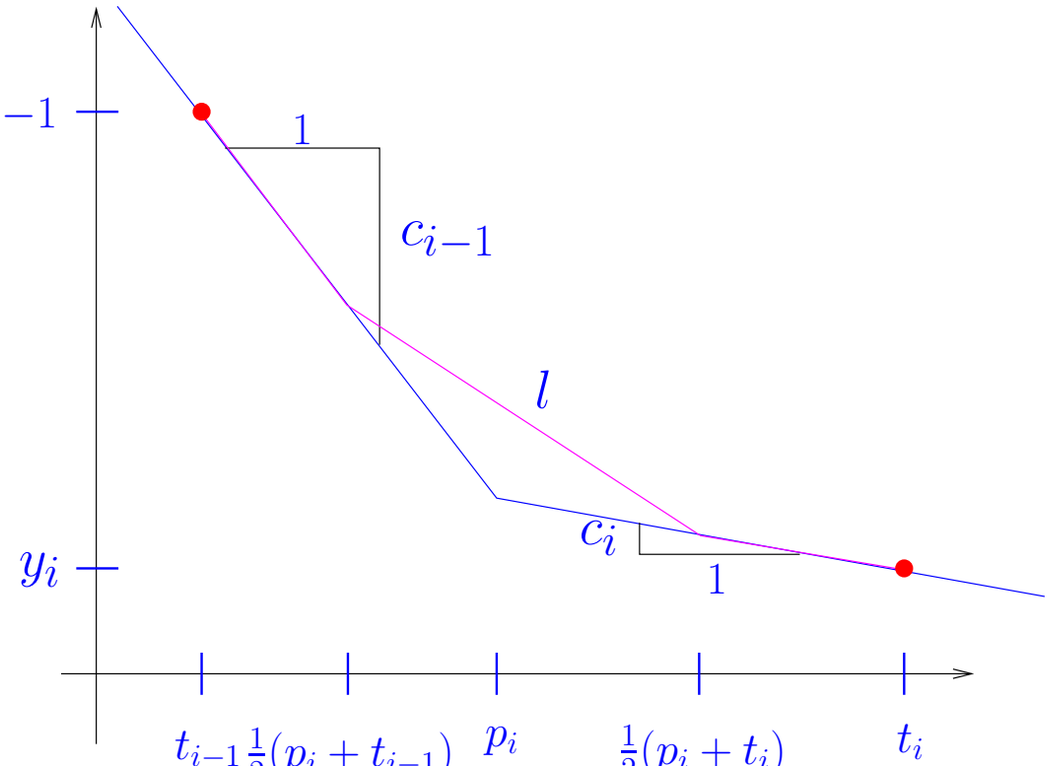
② Choice of "extra knots" $p_i \in]t_{i-1}, t_i]$, $i = 1, \ldots, n$:

$$p_i = \begin{cases} \text{intersection of the two straight lines} & \\ \text{resp. through points} (t_{i-1}, y_{i-1}), (t_i, y_i) & \text{if the intersection point belongs to } (t_{i-1}, t_i] \text{ ,} \\ \text{with slopes } c_{i-1}, c_i \text{, resp.} & \\ \frac{1}{2}(t_{i-1} + t_i) & \text{otherwise .} \end{cases}$$

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

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

```
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
```



R. Hiptmair

rev 30401, October 3, 2010

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

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

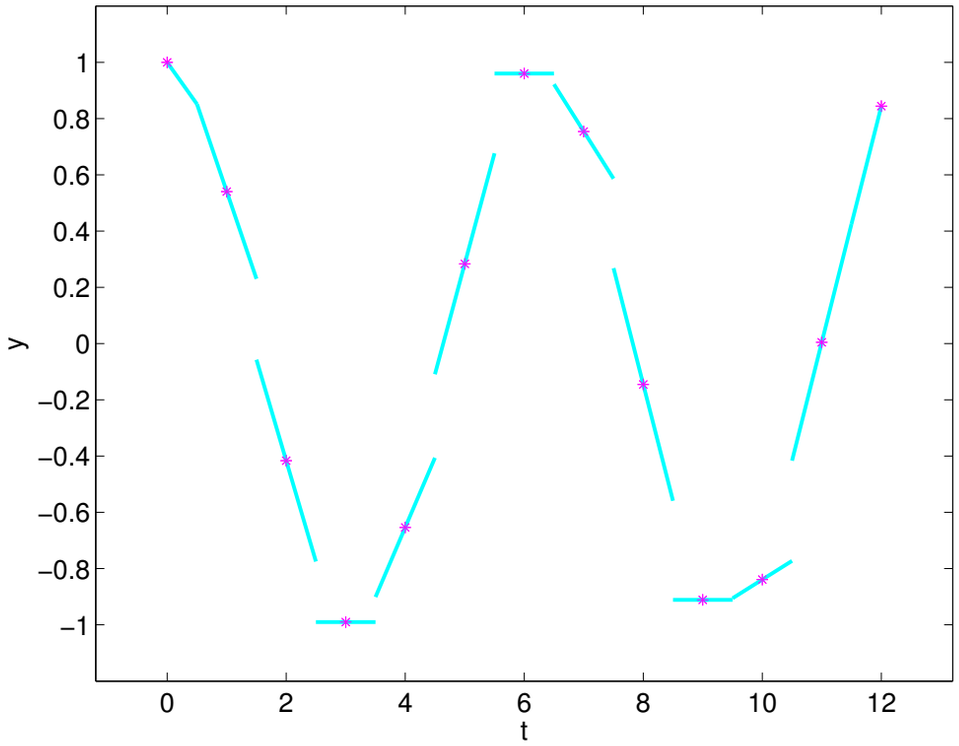
with $l(t_i) = y_i$, $l'(t_i) = c_i$.

- In each interval $(\frac{1}{2}(p_j+t_j), \frac{1}{2}(t_j+p_{j+1}))$ the spline corresponds to the segment of slope c_j passing through the data node (t_j, y_j).
- In each interval $(\frac{1}{2}(t_j+p_{j+1}), \frac{1}{2}(p_{j+1}+t_{j+1}))$ the spline corresponds to the segment connecting the previous ones.

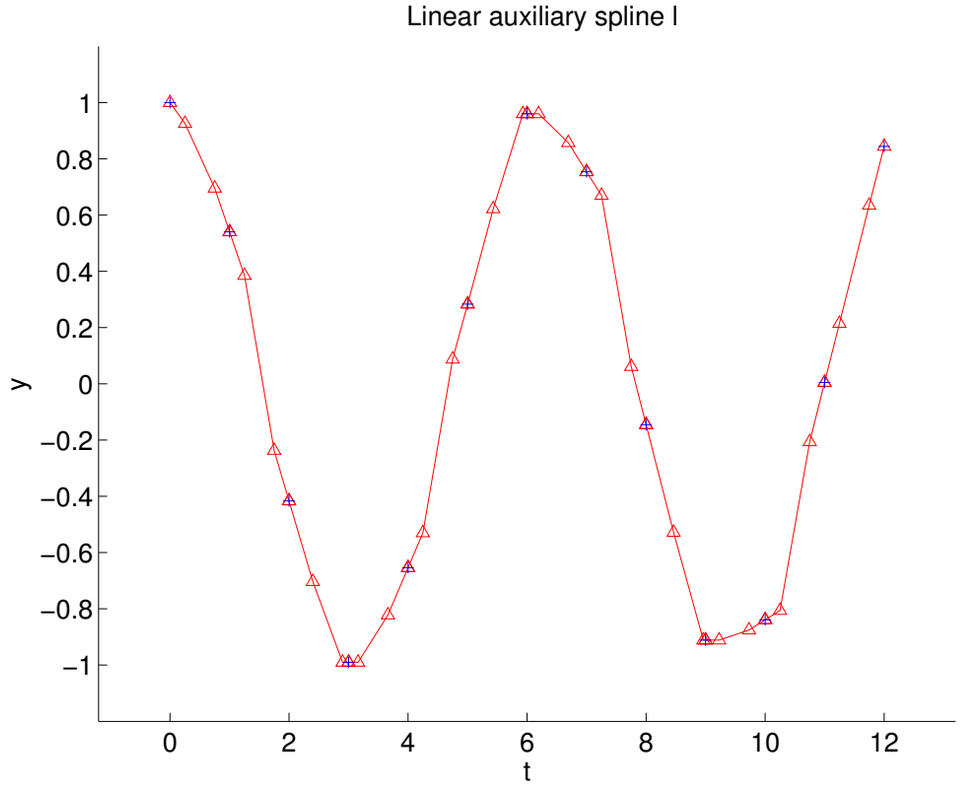
l "inherits" local monotonicity and curvature from the data.

Example 3.7.12 (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

◇

④ Local quadratic approximation / interpolation of l:

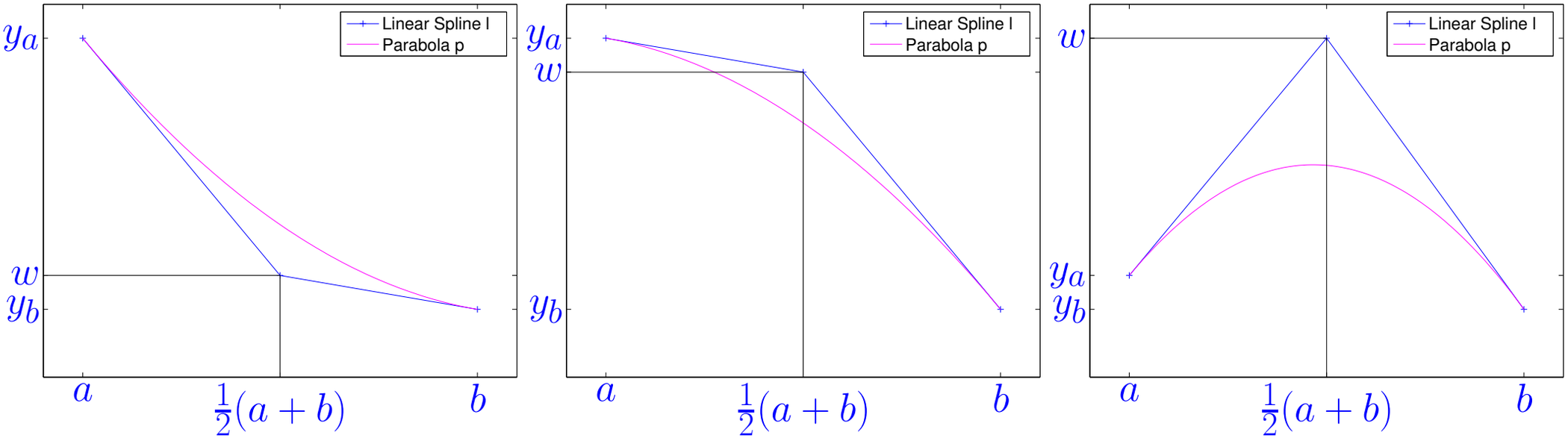
If g is a linear spline through three points (a, y_a), $(\frac{1}{2}(a+b), w)$, (b, y_b), $a < b$, $y_a, y_b, w \in \mathbb{R}$,

the parabola $p(t) := (y_a(b-t)^2 + 2w(t-a)(b-t) + y_b(t-a)^2)/(b-a)^2$, $a \le t \le b$,

satisfies $p(a) = y_a$, $p(b) = y_b$, $p'(a) = g'(a)$, $p'(b) = g'(b)$.

g monotonic increasing / decreasing $\Rightarrow$ p monotonic increasing / decreasing

g convex / concave $\Rightarrow$ p convex / concave



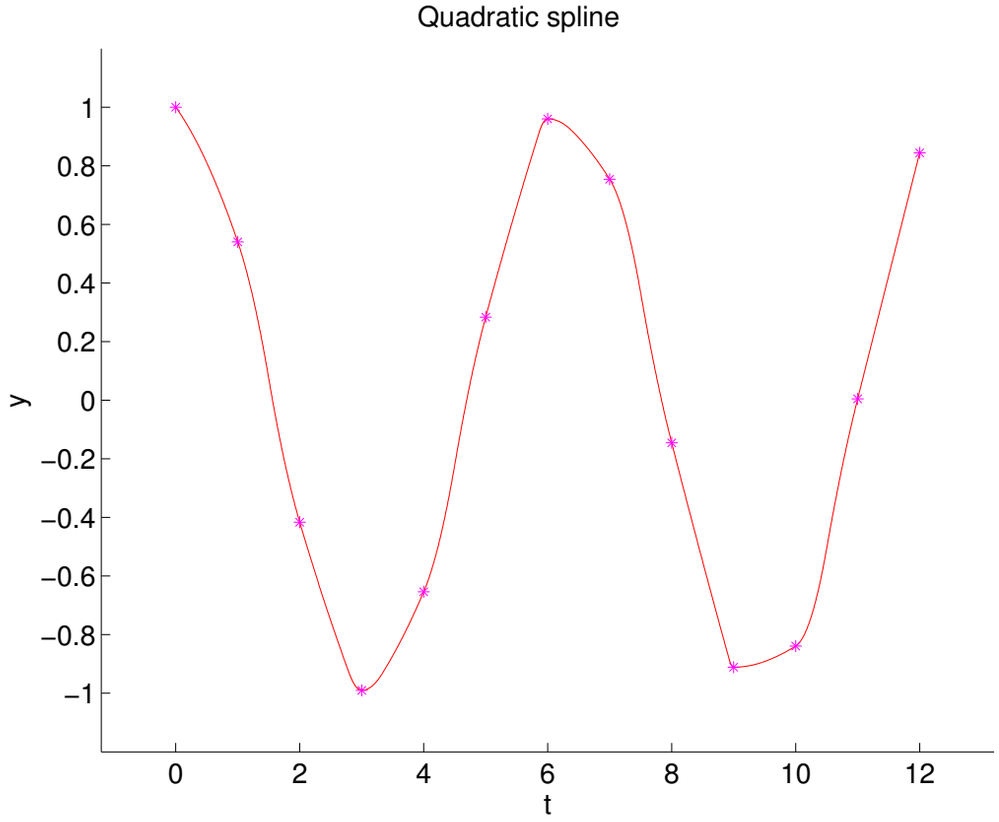
This implies that the final quadratic spline that passes through the points (t_j, y_j) with slopes c_j can be built locally as p using the linear spline l, in place of g.

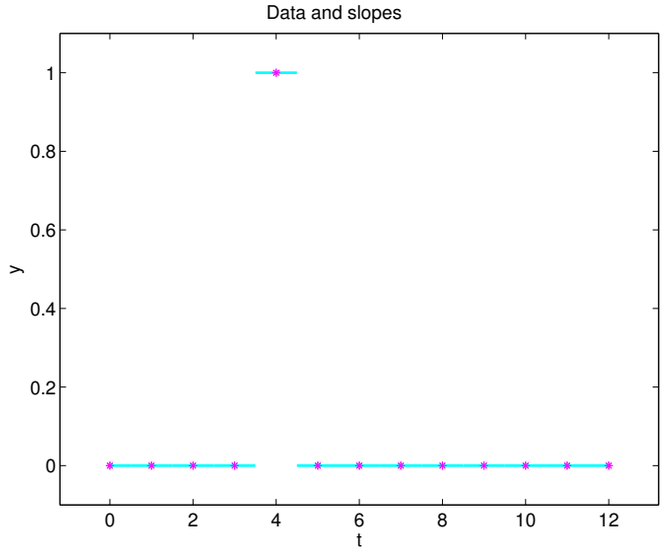
R. Hiptmair

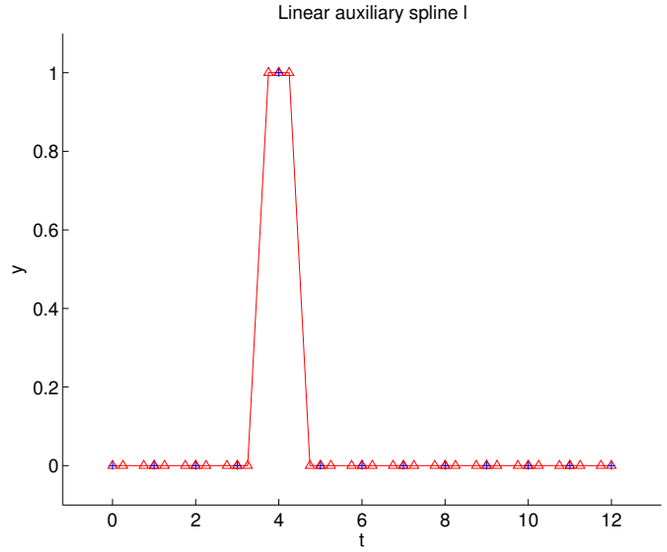
rev 30401, October 3, 2010

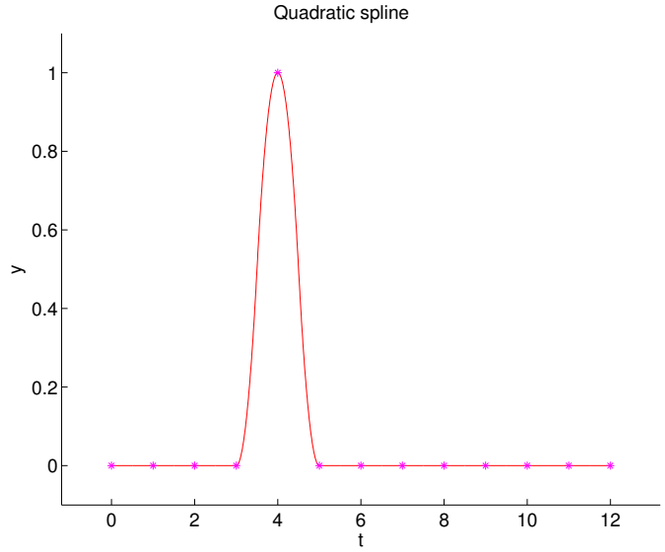
Continuation of Ex. 3.7.12:

Interpolating quadratic spline









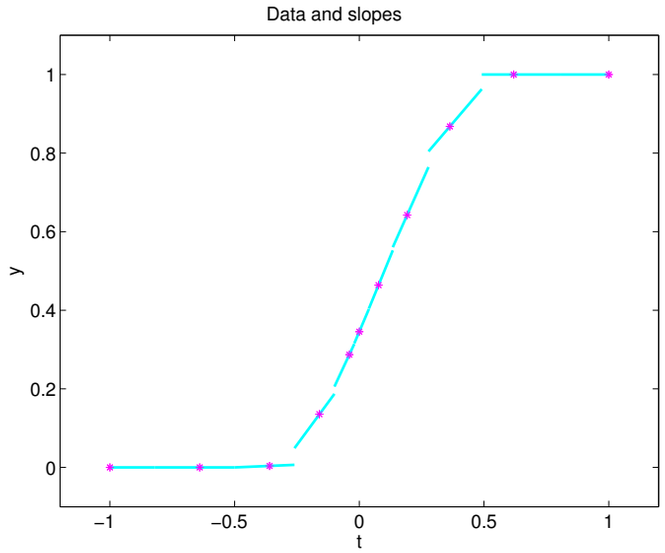
Shape preserving quadratic spline interpolation = local **but** not linear

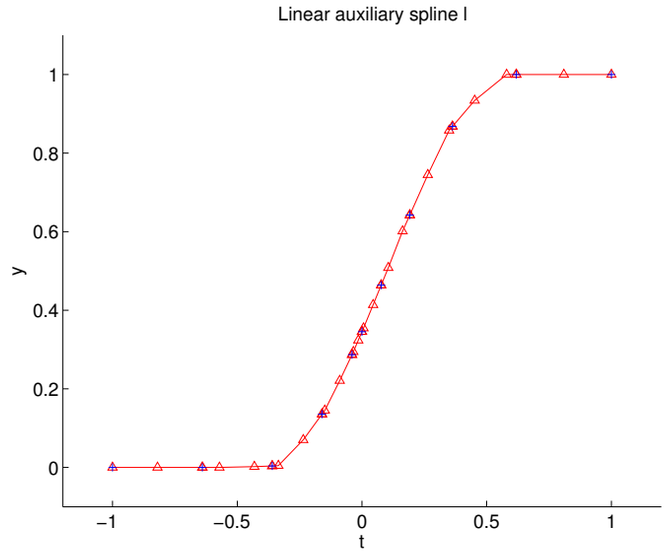
R. Hiptmair

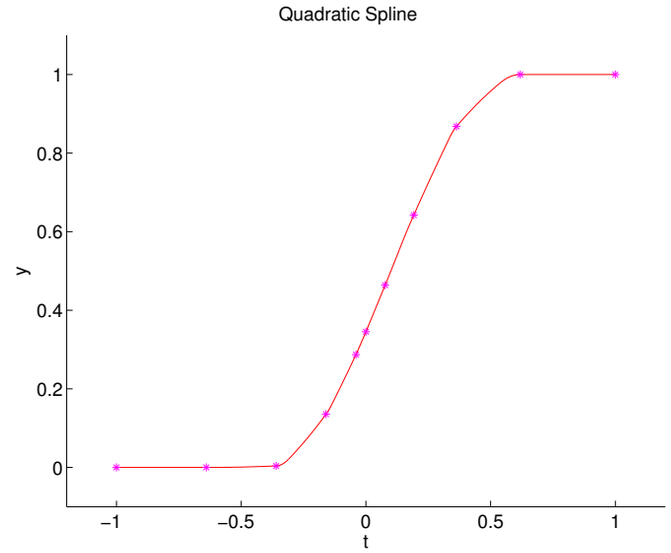
rev 30401, October 3, 2010

Example 3.7.13 (Shape preserving quadratic spline interpolation).

Data from Ex. 3.5.5:





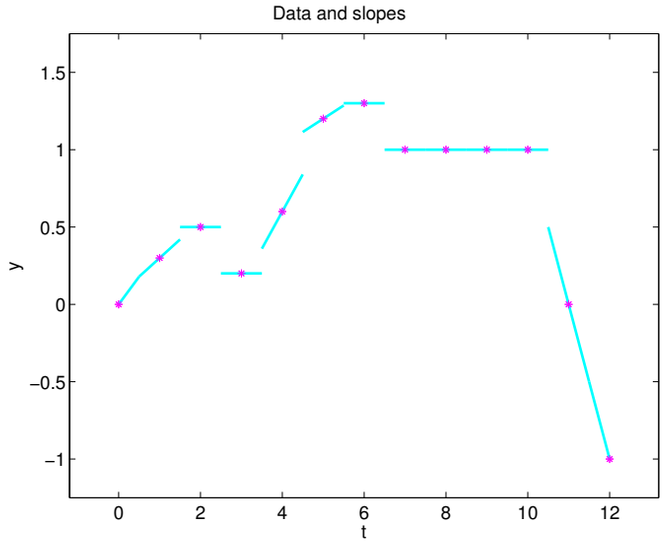


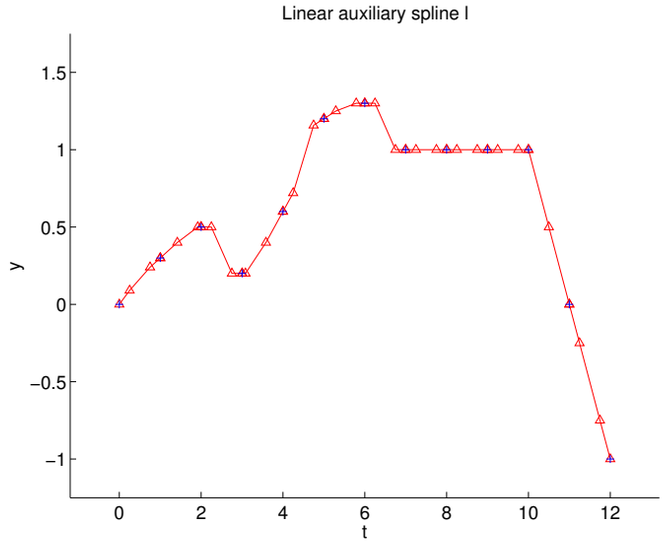
Data from [35]:

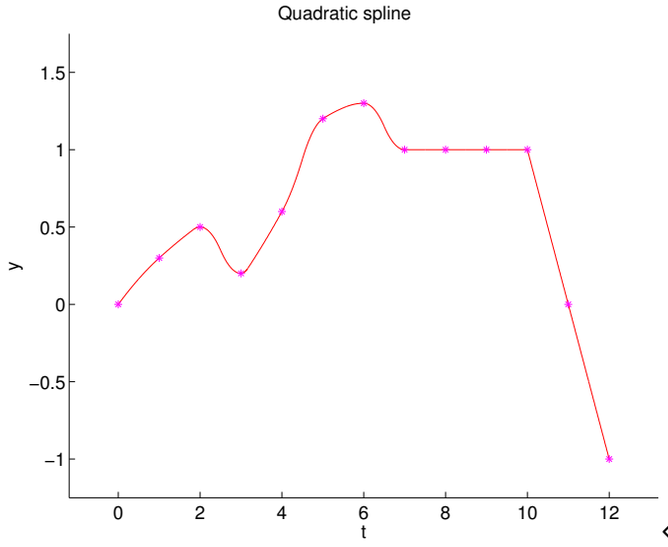
t_i	0	1	2	3	4	5	6	7	8	9	10	11	12
y_i	0	0.3	0.5	0.2	0.6	1.2	1.3	1	1	1	1	0	-1

R. Hiptmair

rev 30401, October 3, 2010







Code 3.7.14: Step by step shape preserving spline interpolation

```
% 22.06.2009 shapepresintp.m
% Shape preserving interpolation through nodes (t,y)
% Build a quadratic spline interpolation without overshooting
% In 4 steps, see the comments in each step
%
% example:
% shapepresintp([1, 1.5,3:10], [ 1 2.5 4 3.8 3 2 1 4 2 1])
% shapepresintp([0:0.05:1], cosf([0:0.05:1]))
% shapepresintp([0:0.05:1], [zeros(1,5),1,zeros(1,15) ] )
% shapepresintp([0:12], [0 0.3 0.5 0.2 0.6 1.2 1.3 1 1 1 1 0 -1])
```

```
1
2  function shapepresintp(t,y)
3
4  n=length(t)-1;
5  % plot the data and prepare the figure
6  figure;
7  hplot(1)=plot(t,y,'k*-');
8  hold on;
9  newaxis=axis;
10 newaxis([3,4])=[newaxis(3)-0.5, newaxis(4)+0.5];
11 axis(newaxis);              % enlarge the vertical size of the plot
12 title('Data points - Press enter to continue')
13 plot(t,ones(1,n+1)* (newaxis(3)+0.25),'k.');
14 set(gca, 'XTick', t)
15 leg={'Data','Slopes','Middle points','Linear spline' ,'Sh. pres.
     spline'};
16 legend(hplot(1),leg{1});
17 pause;
18
19 % ===== Step 1:  choice of slopes =====
20 % shape-faithful slopes (c) in the nodes using harmonic mean of data slopes
21 % the final interpolant will be tangents to these segments
22
```

R. Hiptmair

rev 30401, October 3, 2010

```
disp('STEP 1')
title('Shape-faithful slopes - Press enter to continue')
h=diff(t);
delta = diff(y) ./ h;                    % slopes of data
c=zeros(size(t));
for j=1:n-1
  if (delta(j)*delta(j+1) >0)
    c(j+1) = 2/(1/delta(j) + 1/delta(j+1));
  end
end
c(1)=2*delta(1)-c(2);  c(n+1)=2*delta(n)-c(n);

% plot segments indicating the slopes c(i):
% use (vector) plot handle 'hplot' to reduce the linewidth in step 2
hplots=zeros(1,n+1);
for j=2:n
  hplotsl(j)=plot([t(j)-0.3*h(j-1),t(j)+0.3*h(j)],
    [y(j)-0.3*h(j-1)*c(j),y(j)+0.3*h(j)*c(j)],'-','linewidth',2);
end
hplotsl(1)=plot([t(1),t(1)+0.3*h(1)], [y(1),y(1)+0.3*h(1)*c(1)],
  '-','linewidth',2);
hplotsl(n+1)= plot([t(end)-0.3*h(end),t(end)],
  [y(end)-0.3*h(end)*c(end),y(end)], '-','linewidth',2);
```

```
53 legend([hplot(1), hplotsl(1)], leg{1:2});
54 pause;
55
56 % ===== Step 2:  choice of middle points =====
57 % fix points p(j) in [t(j), t(j+1)], depending on the slopes c(j), c(j+1)
58
59 disp('STEP 2')
60 title('Middle points - Press enter to continue')
61 set(hplotsl,'linewidth',1)
62
63 p = (t(1)-1)*ones(1,length(t)-1);
64 for j=1:n
65   if (c(j) ~= c(j+1))
66     p(j)=(y(j+1)-y(j)+ t(j)*c(j)-t(j+1)*c(j+1)) / (c(j)-c(j+1));
67   end
68   % check and repair if p(j) is outside its interval:
69   if ((p(j)<t(j))||(p(j)>t(j+1)));    p(j) = 0.5*(t(j)+t(j+1));
     end;
70 end
71
72 hplot(2)=plot(p,ones(1,n)*(newaxis(3)+0.25),'go');
73 legend([hplot(1), hplotsl(1), hplot(2)], leg{1:3});
74 pause;
```

```
75
76 % ===== Step 3:  auxiliary linear spline =====
77 % build the linear spline with nodes in:
78 % -t(j)
79 % -the middle points between t(j) and p(j)
80 % -the middle points between p(j) and t(j+1)
81 % -t(j+1)
82 % and with slopes c(j) in t(j), for every j
83
84 disp('STEP 3')
85 title('Auxiliary linear spline - Press enter to continue')
86
87 for j=1:n
88   hplot(3)=plot([t(j) 0.5*(p(j)+t(j)) 0.5*(p(j)+t(j+1)) t(j+1)],
         [y(j) y(j)+0.5*(p(j)-t(j))*c(j)
     y(j+1)+0.5*(p(j)-t(j+1))*c(j+1) y(j+1)],  'm-^');
89   plot ([t(j) 0.5*(p(j)+t(j)) 0.5*(p(j)+t(j+1)) t(j+1)],
     ones(1,4)*(newaxis(3)+0.25)  ,  'm^');
90 end
91 legend([hplot(1), hplotsl(1), hplot(2), hplot(3)], leg{1:4});
92 pause;
93
94 % ===== Step 4:  quadratic spline =====
```

```
% final quadratic shape preserving spline
% quadratic polynomial in the intervals [t(j), p(j)] and [p(j), t(j)]
% tangent in t(j) and p(j) to the linear spline of step 3

disp('STEP 4')
title('Quadratic spline')

% for every interval 2 quadratic interpolations
% a, b, ya, yb = extremes and values in the subinterval
% w = value in middle point that gives the right slope
for j=1:n
    a=t(j);
    b=p(j);
    ya = y(j);
    w = y(j)+0.5*(p(j)-t(j))*c(j);
    yb = ((t(j+1)-p(j))*(y(j)+0.5*(p(j)-t(j))*c(j))+...
      (p(j)-t(j))*(y(j+1)+0.5*(p(j)-t(j+1))*c(j+1)))/(t(j+1)-t(j));
    x=linspace(a,b,100);
    pb = (ya*(b-x).^2 + 2*w*(x-a).*(b-x)+yb*(x-a).^2)/((b-a)^2);
    hplot(4)=plot(x,pb,'r-', 'linewidth',2);

    a = b;
    b = t(j+1);
```

```
18      ya = yb;
19      yb = y(j+1);
20      w = y(j+1)+0.5*(p(j)-t(j+1))*c(j+1);
21      x = (a:(b-a)/100:b);
22      pb = (ya*(b-x).^2 + 2*w*(x-a).*(b-x)+yb*(x-a).^2)/((b-a)^2);
23      plot(x,pb,'r-', 'linewidth',2);
24
25      plot(p(j),ya,'go');
26  end
27
28  % replot initial nodes over the other plots:
29  plot(t,y,'k*');
30  % plot(p,yb,'go')
31  legend([hplot(1), hplotsl(1), hplot(2:4)], leg);
32  title('Shape preserving interpolation')
```


R. Hiptmair

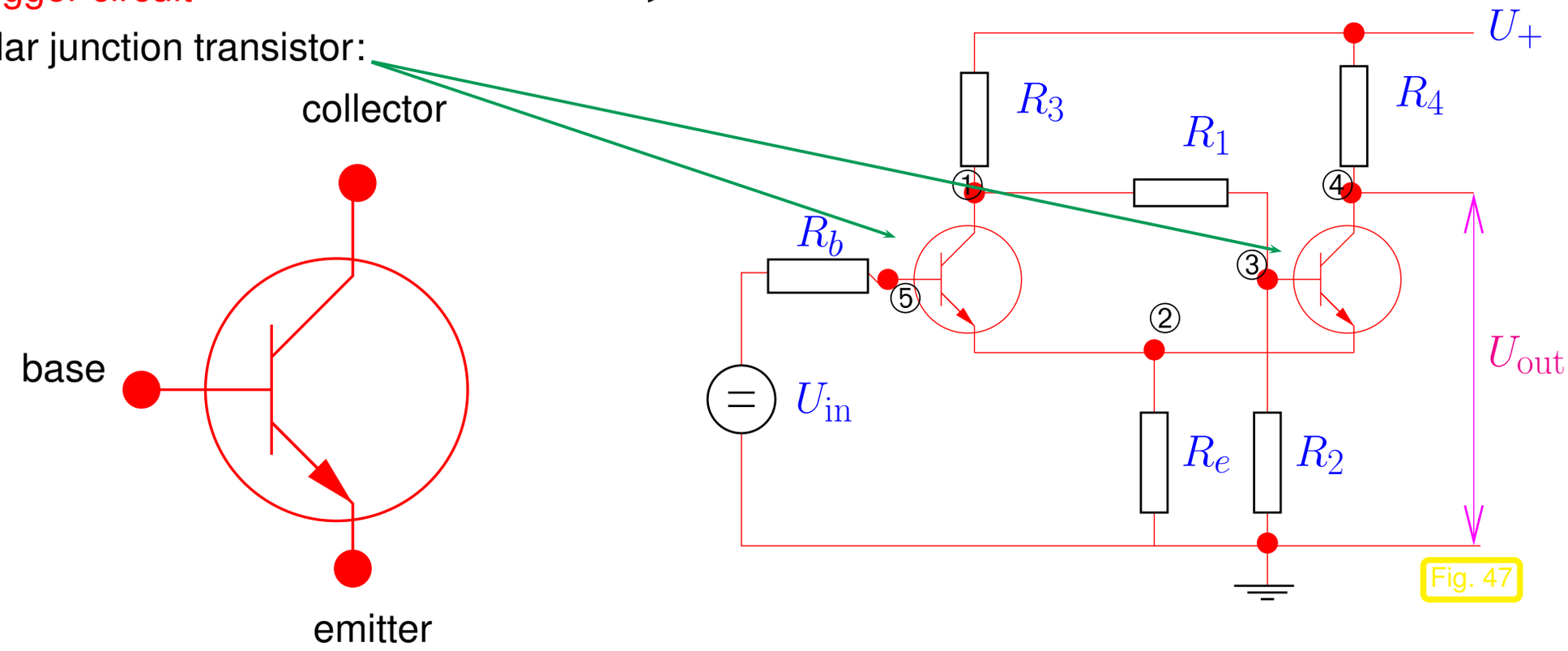
rev 30401, October 3, 2010

4 Iterative Methods for Non-Linear Systems of Equations

Example 4.0.1 (Non-linear electric circuit).

Schmitt trigger circuit ▷

NPN bipolar junction transistor:



R. Hiptmair

rev 30401, October 21, 2010

Ebers-Moll model (large signal approximation):

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

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

(β_F is the forward common emitter current gain (20 to 500), β_R is the reverse common emitter current gain (0 to 20), I_S is the reverse saturation current (on the order of 10^{-15} to 10^{-12} amperes), U_T is the thermal voltage (approximately 26 mV at 300 K).)

R. Hiptmair

rev 30401, October 21, 2010

Non-linear system of equations from nodal analysis (→ Ex. 2.0.1):

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

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

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

A non-linear system of equations is a concept almost *too abstract to be useful*, because it covers an extremely wide variety of problems . Nevertheless in this chapter we will mainly look at "generic" methods for such systems. This means that every method discussed may take a good deal of fine-tuning before it will really perform satisfactorily for a given non-linear system of equations.

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

$\Updownarrow$

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

R. Hiptmair

rev 30401, October 21, 2010

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

Note: $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ $\leftrightarrow$ "same number of equations and unknowns"

In general no existence & uniqueness of solutions

4.1 Iterative methods

R. Hiptmair

rev 30401, February 24, 2009

Remark 4.1.1 (Necessity of iterative approximation).

Gaussian elimination ($\rightarrow$ Sect. 2.1) provides an algorithm that, if carried out in exact arithmetic, computes the solution of a linear system of equations with a *finite* number of elementary operations. However, linear systems of equations represent an exceptional case, because it is hardly ever possible to solve general systems of non-linear equations using only finitely many elementary operations. Certainly this is the case whenever irrational numbers are involved.

△

An iterative method for (approximately) solving the non-linear equation $F(\mathbf{x}) = 0$ is an algorithm generating a sequence $(\mathbf{x}^{(k)})_{k\in\mathbb{N}_0}$ of approximate solutions.

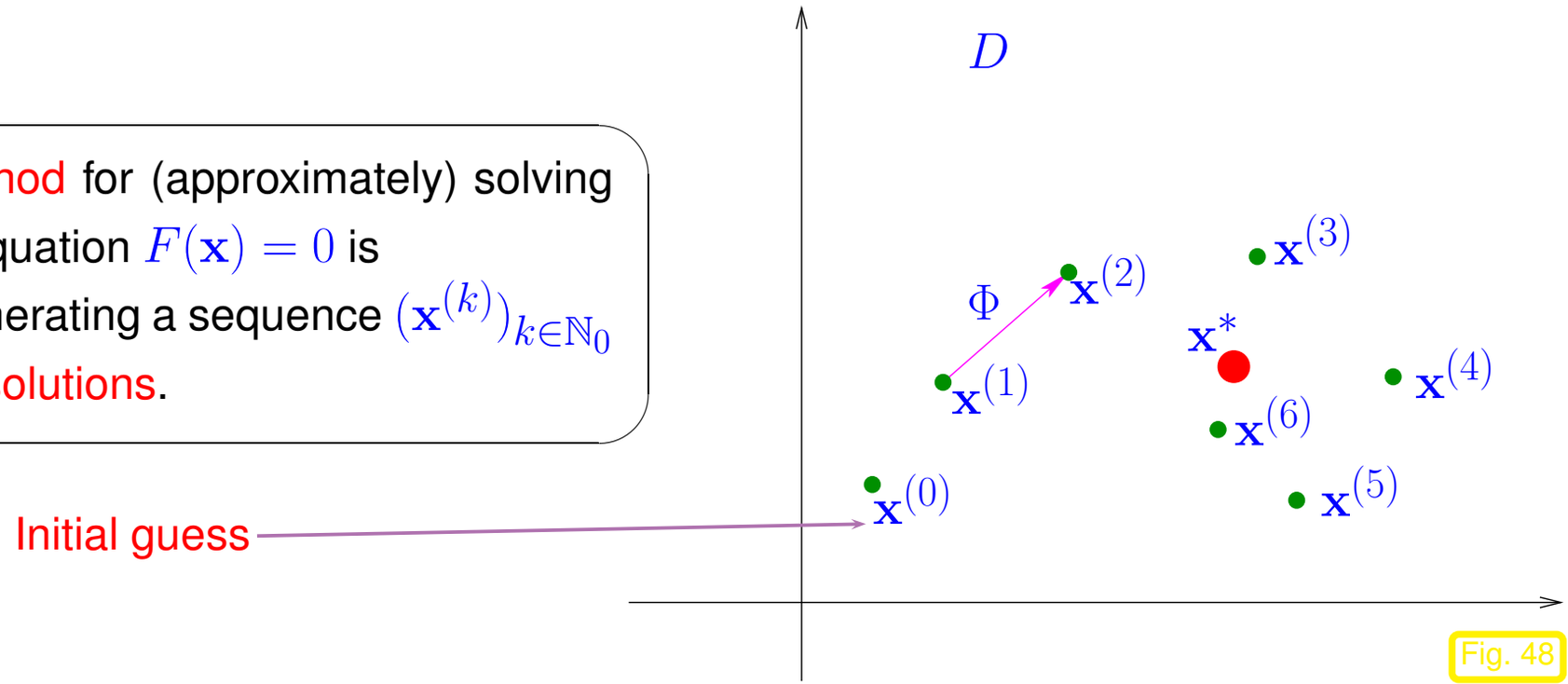


Fig. 48

- iterate $\mathbf{x}^{(k)}$ depends on F and (one or several) $\mathbf{x}^{(n)}$, $n < k$, e.g.,

$$\mathbf{x}^{(k)} = \underbrace{\Phi_F(\mathbf{x}^{(k-1)}, \ldots, \mathbf{x}^{(k-m)})}_{\text{iteration function for } m\text{-point method}} \tag{4.1.2}$$

- $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(m-1)}$ = initial guess(es) (*ger.:* Anfangsnäherung)

Definition 4.1.3 (Convergence of iterative methods)**.**

An iterative method converges (for fixed initial guess(es)) $:\Leftrightarrow$ $\mathbf{x}^{(k)} \to \mathbf{x}^*$ *and* $F(\mathbf{x}^*) = 0$.

Definition 4.1.4 (Consistency of iterative methods)**.**

An iterative method is consistent with $F(\mathbf{x}) = 0$

$$:\Leftrightarrow \quad \Phi_F(\mathbf{x}^*, \ldots, \mathbf{x}^*) = \mathbf{x}^* \quad \Leftrightarrow \quad F(\mathbf{x}^*) = 0$$

R. Hiptmair

rev 30401, February 24, 2009

Terminology: error of iterates $\mathbf{x}^{(k)}$ is defined as: $\mathbf{e}^{(k)} := \mathbf{x}^{(k)} - \mathbf{x}^*$

Definition 4.1.5 (Local and global convergence). $\rightarrow$ *[29, Def. 17.1]*

An iterative method converges locally to $\mathbf{x}^* \in \mathbb{R}^n$, if there is a neighborhood $U \subset D$ of $\mathbf{x}^*$, such that

$$\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(m-1)} \in U \quad \Rightarrow \quad \mathbf{x}^{(k)} \text{ well defined} \quad \wedge \quad \lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$$

for the sequences $(\mathbf{x}^{(k)})_{k\in\mathbb{N}_0}$ of iterates.

If $U = D$, the iterative method is globally convergent.

local convergence ▷

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

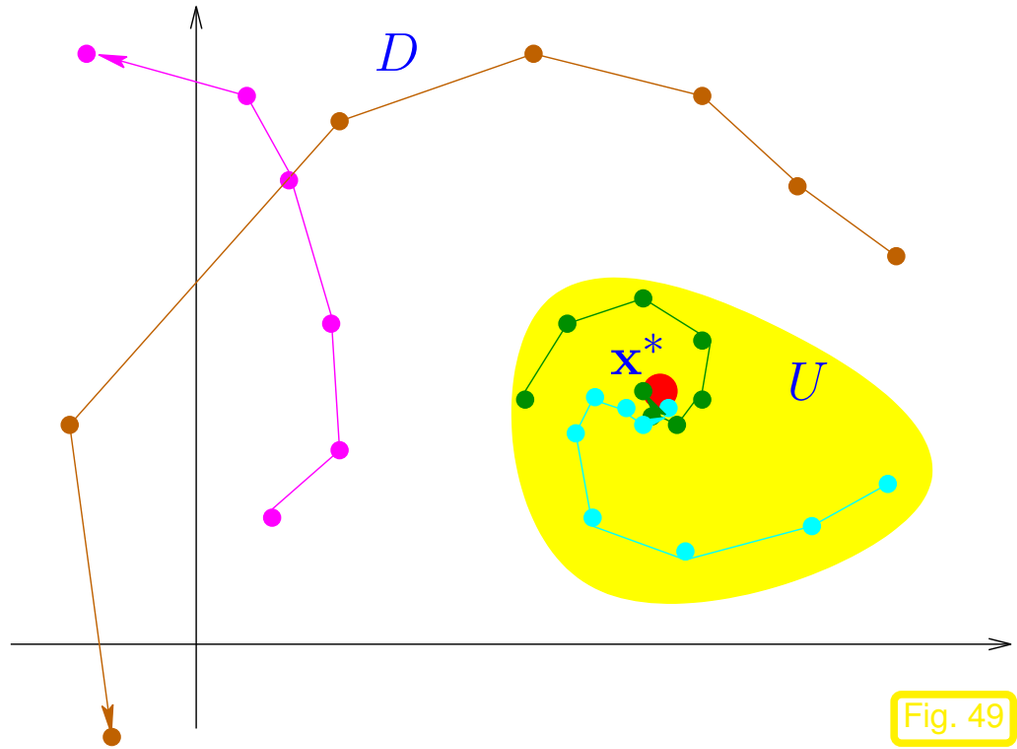


Fig. 49

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

Two general questions: How to measure the speed of convergence?
When to terminate the iteration?

R. Hiptmair

rev 30401, February 24, 2009

4.1.1 Speed of convergence

Here and in the sequel, $\|\cdot\|$ designates a generic vector norm, see Def. 2.5.1. Any occurring matrix norm is induced by this vector norm, see Def. 2.5.5.

It is important to be aware which statements depend on the choice of norm and which do not!

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

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

Recall: equivalence of all norms on finite dimensional vector space $\mathbb{K}^n$:

Definition 4.1.8 (Equivalence of norms).
Two norms $\|\cdot\|_1$ and $\|\cdot\|_2$ on a vector space V are equivalent if

$$\exists \underline{C}, \overline{C} > 0: \quad \underline{C}\,\|v\|_1 \leq \|v\|_2 \leq \overline{C}\,\|v\|_1 \quad \forall v \in V \ .$$

Theorem 4.1.9 (Equivalence of all norms on finite dimensional vector spaces).
If $\dim V < \infty$ all norms ($\rightarrow$ Def. 2.5.1) on V are equivalent ($\rightarrow$ Def. 4.1.8).

R. Hiptmair

rev 30401, February 24, 2009

Remark 4.1.10 (Detecting linear convergence).

norms of iteration errors

$\updownarrow$

$\sim$ straight line in lin-log plot

$$\left\|\mathbf{e}^{(k)}\right\| \leq L^k \left\|\mathbf{e}^{(0)}\right\| \ ,$$
$$\log(\left\|\mathbf{e}^{(k)}\right\|) \leq k\log(L) + \log(\left\|\mathbf{e}^{(0)}\right\|) \ .$$

(●: Any "faster" convergence also qualifies as linear !)

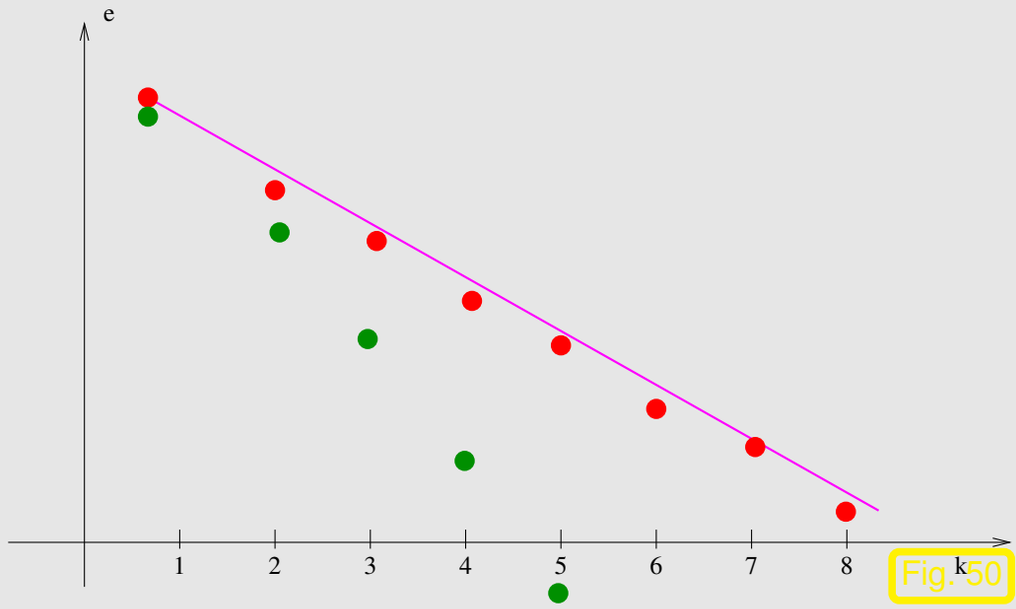


Fig. 50

Let us abbreviate the error norm in step k by $\epsilon_k := \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|$. In the case of linear convergence (see Def. 4.1.6) assume (with $0 < L < 1$)

$$\epsilon_{k+1} \approx L\epsilon_k \quad \Rightarrow \quad \log \epsilon_{k+1} \approx \log L + \log \epsilon_k \quad \Rightarrow \quad \log \epsilon_k \approx k \log L + \log \epsilon_0 \ . \qquad (4.1.11)$$

We conclude that $\log L < 0$ describes slope of graph in lin-log error chart.

△

Example 4.1.12 (Linearly convergent iteration).

Iteration ($n = 1$):

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

x has to be initialized with the different values for x_0.

Code 4.1.13: simple fixed point iteration

```
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

Linear convergence as in Def. 4.1.6

$\Updownarrow$

error graphs = straight lines in lin-log scale

$\rightarrow$ Rem. 4.1.10

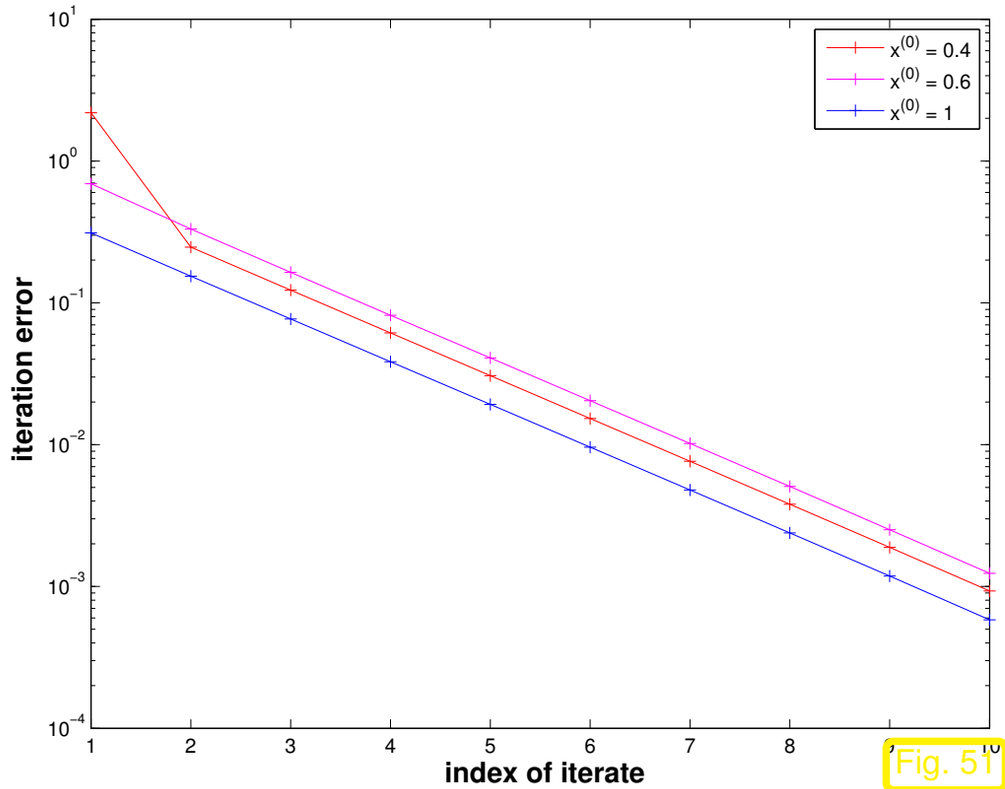


Fig. 51

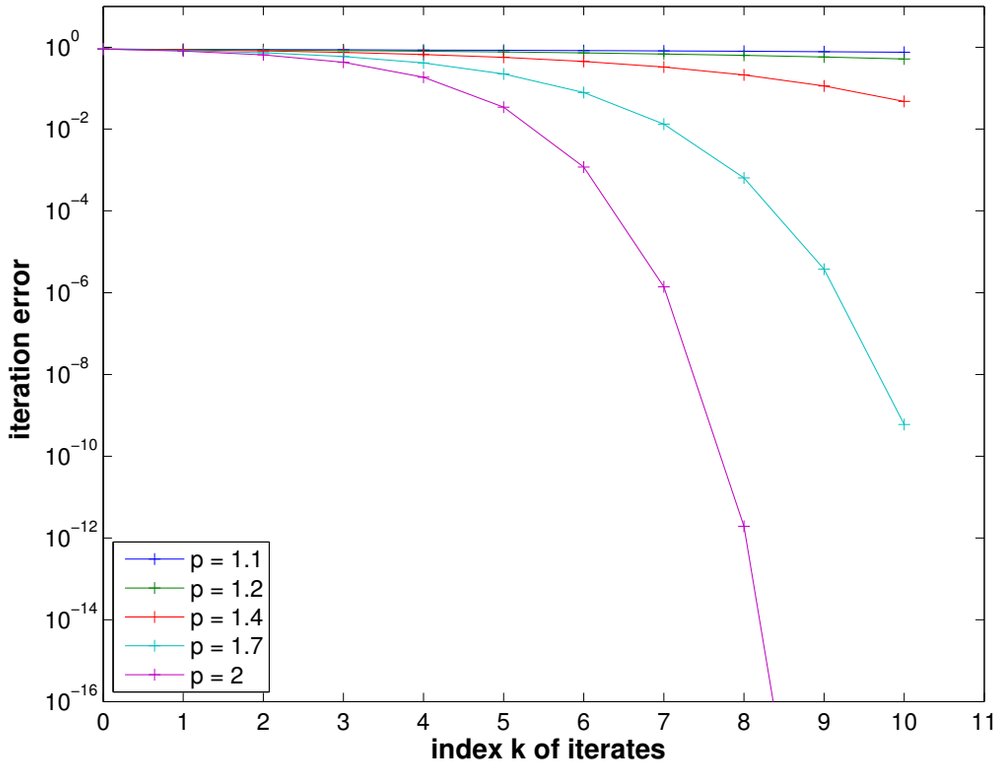
$\diamond$

Definition 4.1.14 (Order of convergence). $\rightarrow$ *[29, Sect. 17.2], [10, Def. 5.14]*

*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)

In the case of convergence of order p ($p > 1$) (see Def. 4.1.14):

$$\epsilon_{k+1} \approx C\epsilon_k^p \quad \Rightarrow \quad \log \epsilon_{k+1} = \log C + p \log \epsilon_k \quad \Rightarrow \quad \log \epsilon_{k+1} = \log C \sum_{l=0}^{k} p^l + p^{k+1} \log \epsilon_0$$

$$\Rightarrow \quad \log \epsilon_{k+1} = -\frac{\log C}{p-1} + \left(\frac{\log C}{p-1} + \log \epsilon_0 \right) p^{k+1} \, .$$

In this case, the error graph is a concave power curve (for sufficiently small ϵ_0 !)

R. Hiptmair

rev 30401, February 24, 2009

Remark 4.1.15 (Detecting order of convergence).

How to guess the order of convergence ($\rightarrow$ Def. 4.1.14) from a numerical experiment?
Abbreviate $\epsilon_k := \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\|$ (norm of iteration error):

$$\text{assume} \quad \epsilon_{k+1} \approx C\epsilon_k^p \quad \Rightarrow \quad \log \epsilon_{k+1} \approx \log C + p \log \epsilon_k \quad \Rightarrow \quad \frac{\log \epsilon_{k+1} - \log \epsilon_k}{\log \epsilon_k - \log \epsilon_{k-1}} \approx p \; .$$

➢ monitor the quotients $(\log \epsilon_{k+1} - \log \epsilon_k)/(\log \epsilon_k - \log \epsilon_{k-1})$ over several steps of the iteration.

△

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

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

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

By the arithmetic-geometric mean inequality (AGM) $\sqrt{ab} \leq \frac{1}{2}(a+b)$ we conclude: $x^{(k)} > \sqrt{a}$ for $k \geq 1$.

$\Rightarrow$ sequence from (4.1.17) converges with order 2 to $\sqrt{a}$

Note: $x^{(k+1)} < x^{(k)}$ for all $k \geq 2$ $\succ$ $(x^{(k)})_{k \in \mathbb{N}_0}$ converges as a decreasing sequence that is bounded from below ($\rightarrow$ analysis course)

Numerical experiment: iterates for $a = 2$:

k	$x^{(k)}$	$e^{(k)} := x^{(k)} - \sqrt{2}$	$\log \frac{\lvert e^{(k)} \rvert}{\lvert e^{(k-1)} \rvert} : \log \frac{\lvert e^{(k-1)} \rvert}{\lvert e^{(k-2)} \rvert}$
0	2.0000000000000000	0.5857864376269485	
1	1.5000000000000000	0.0857864376269485	
2	1.4166666666666652	0.0024531042935713 7	1.850
3	1.4142156862745096 6	0.0000021239014145 2	1.984
4	1.4142135623746898 7	0.0000000000015947 2	2.000
5	1.4142135623730949 2	0.0000000000000002 2	0.630

Note the doubling of the number of significant digits in each step ! [impact of roundoff !]

The doubling of the number of significant digits for the iterates holds true for any convergent second-order iteration:

Indeed, denoting the relative error in step k by δ_k, we have:

$$\begin{aligned} x^{(k)} = x^*(1+\delta_k) \quad &\Rightarrow \quad x^{(k)} - x^* = \delta_k x^* \,. \\ \Rightarrow |x^*\delta_{k+1}| = |x^{(k+1)} - x^*| \leq C|x^{(k)} - x^*|^2 &= C|x^*\delta_k|^2 \\ \Rightarrow \qquad |\delta_{k+1}| &\leq C|x^*|\delta_k^2 \,. \end{aligned} \tag{4.1.18}$$

Note: $\delta_k \approx 10^{-\ell}$ means that $\mathbf{x}^{(k)}$ has ℓ significant digits.

Also note that if $C \approx 1$, then $\delta_k = 10^{-\ell}$ and (4.1.16) implies $\delta_{k+1} \approx 10^{-2\ell}$.

◇

4.1.2 Termination criteria

Usually (even without roundoff errors) the iteration will never arrive at an/the exact solution $\mathbf{x}^*$ after finitely many steps. Thus, we can only hope to compute an *approximate* solution by accepting $\mathbf{x}^{(K)}$ as result for some $K \in \mathbb{N}_0$. Termination criteria (*ger.:* Abbruchbedingungen) are used to determine a suitable value for K.

For the sake of efficiency: $\triangleright$ stop iteration when iteration error is just "small enough"

"small enough" depends on concrete setting:

Usual goal: $\left\|\mathbf{x}^{(K)} - \mathbf{x}^*\right\| \leq \tau$, $\tau \;\hat{=}$ prescribed (absolute) tolerance.

Ideal: $$K = \operatorname{argmin}\{k \in \mathbb{N}_0 \colon \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| < \tau\} \,. \tag{4.1.19}$$

① A priori termination: stop iteration after fixed number of steps (possibly depending on $\mathbf{x}^{(0)}$).

Drawback: hardly ever possible !

Alternative: A posteriori termination criteria

use already computed iterates to decide when to stop

②

Reliable termination: stop iteration $\{\mathbf{x}^{(k)}\}_{k\in\mathbb{N}_0}$ with limit $\mathbf{x}^*$, when

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \leq \tau \;, \qquad \tau \hat{=} \text{ prescribed (absolute) tolerance } > 0 \;. \tag{4.1.20}$$

$\mathbf{x}^*$ not known !

Invoking additional properties of either the non-linear system of equations $F(\mathbf{x}) = 0$ or the iteration it is sometimes possible to tell that for sure $\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \leq \tau$ for all $k \geq K$, though this K may be (significantly) larger than the optimal termination index from (4.1.19), see Rem. 4.1.23.

R. Hiptmair

rev 30401, February 24, 2009

③ use that $\mathbb{M}$ is finite! ($\rightarrow$ Sect. 2.4)

➢ possible to wait until (convergent) iteration becomes stationary

possibly grossly inefficient ! (always computes "up to machine precision")

Code 4.1.22: stationary iteration in $\mathbb{M}$, $\rightarrow$ Ex. 4.1.16

```
1 function x = sqrtit(a)
2 x_old = -1; x = a;
3 while (x_old ~= x)
4   x_old = x;
5   x = 0.5*(x+a/x);
6 end
```

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

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

no guaranteed accuracy

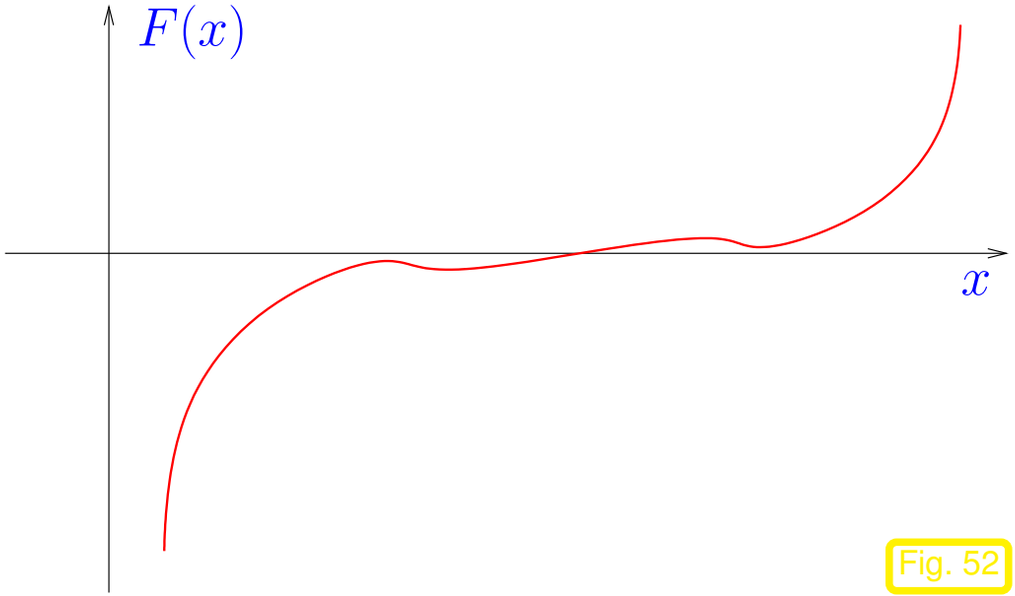


Fig. 52

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

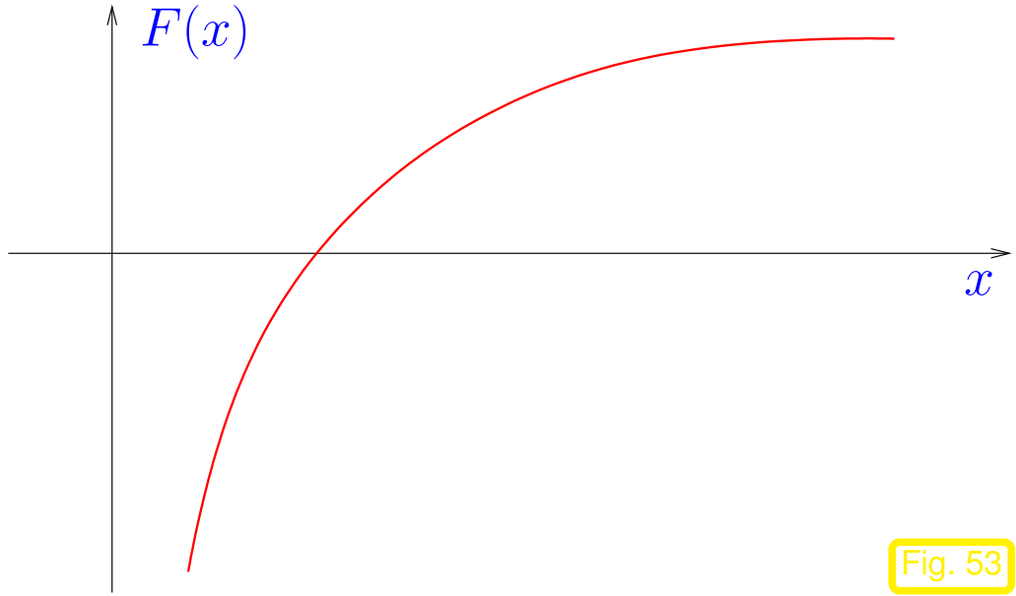


Fig. 53

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

Sometimes extra knowledge about the type/speed of convergence allows to achieve reliable termination in the sense that (4.1.20) can be guaranteed though the number of iterations might be (slightly) too large.

Remark 4.1.23 (A posteriori termination criterion for linearly convergent iterations). $\rightarrow$ [10, Lemma 5.

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

Derivation of a posteriori termination criterion for linearly convergent iterations with rate of convergence $0 < L < 1$:

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \overset{\triangle\text{-inequ.}}{\leq} \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\| + \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \leq \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\| + L\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| .$$

Iterates satisfy:

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

This suggests that we take the right hand side of (4.1.24) as a posteriori error bound.

△

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

Iteration of Example 4.1.12:

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

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	2.19156222199710 1	4.93315487558689 4	2.74159265358979 3
2	0.24713909778107 0	1.94442312421603 1	1.69728402643496 1
3	0.12293673787683 4	0.12420235990423 6	0.00126562202740 1
4	0.06139083520621 7	0.06154590267061 8	0.00015506746440 1
5	0.03068577347226 3	0.03070506173395 4	0.00001928826169 1
6	0.01534168269623 5	0.01534409077602 8	0.00000240807979 2
7	0.00767069088918 5	0.00767099180705 0	0.00000030091786 4
8	0.00383532663866 6	0.00383536425052 0	0.00000003761185 4
9	0.00191766096863 7	0.00191766567002 9	0.00000000470139 2
10	0.00095883019048 9	0.00095883077814 7	0.00000000058765 8
11	0.00047941505854 9	0.00047941513194 1	0.00000000007339 2
12	0.00023970752464 6	0.00023970753390 3	0.00000000000925 7
13	0.00011985376194 9	0.00011985376269 6	0.00000000000074 7
14	0.00005992688130 8	0.00005992688064 1	0.00000000000066 7
15	0.00002996344074 5	0.00002996344056 3	0.00000000000018 1

Hence: the a posteriori error bound is highly accurate in this case!

◇

R. Hiptmair

rev 30401, February 24, 2009

Note: If L not known then using $\widetilde{L} > L$ in error bound is playing safe.

4.2 Fixed Point Iterations

Non-linear system of equations $F(\mathbf{x}) = 0$, $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$,

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

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

Sequence of iterates need not be well defined: $\mathbf{x}^{(k)} \notin U$ possible !

4.2.1 Consistent fixed point iterations

Definition 4.2.1 (Consistency of fixed point iterations, *c.f.* Def. 4.1.4)**.**
A fixed point iteration $\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)})$ *is consistent with* $F(\mathbf{x}) = 0$*, if*

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

Note: iteration function Φ continuous **and** fixed point iteration (locally) convergent to $\mathbf{x}^* \in U$ $\Rightarrow$ $\mathbf{x}^*$ is a fixed point of Φ.

R. Hiptmair

rev 30401, November 10, 2010

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 $\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})$. (4.2.2)

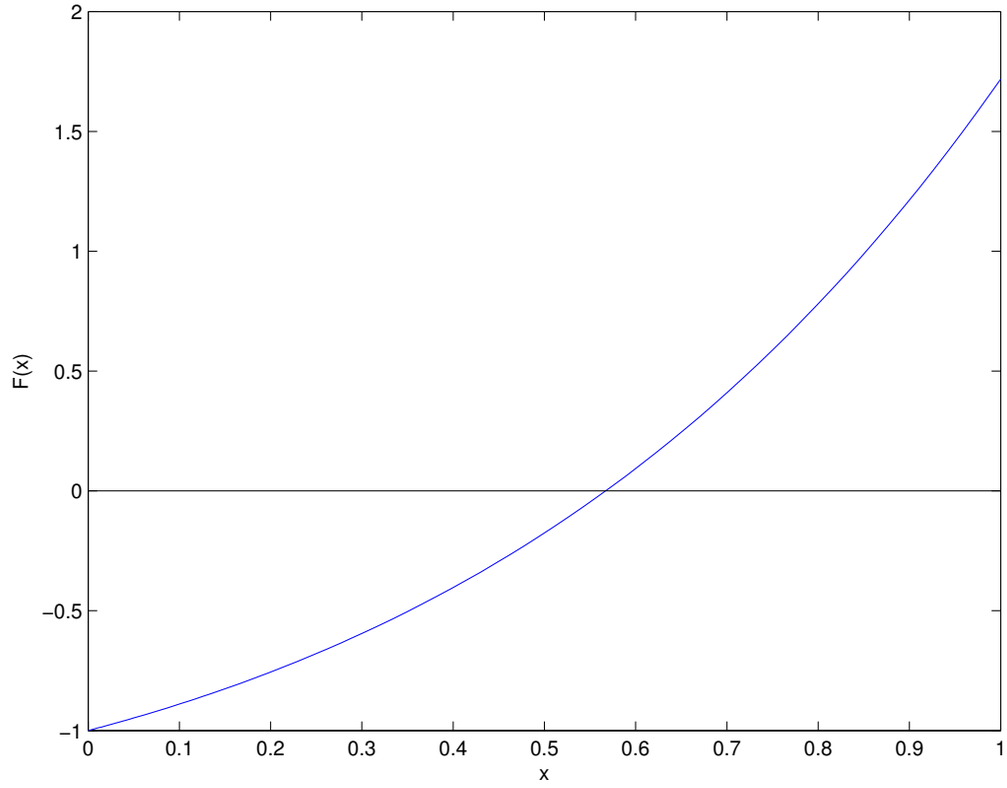
Note: there are *many ways* to transform $F(\mathbf{x}) = 0$ into a fixed point form **!**

Example 4.2.3 (Options for fixed point iterations).

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

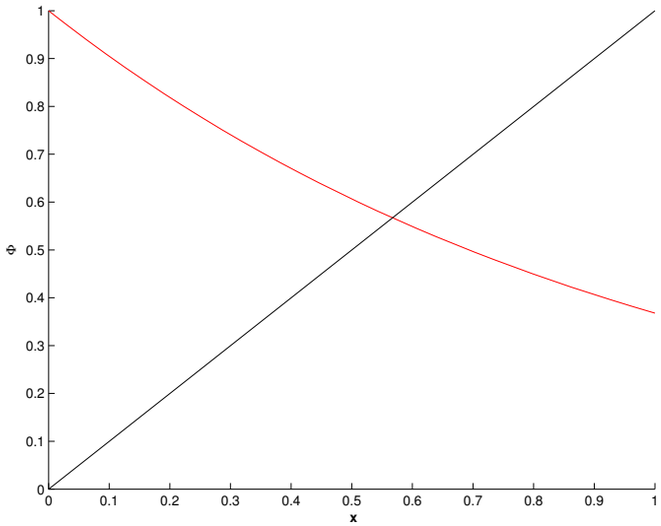
Different fixed point forms:

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

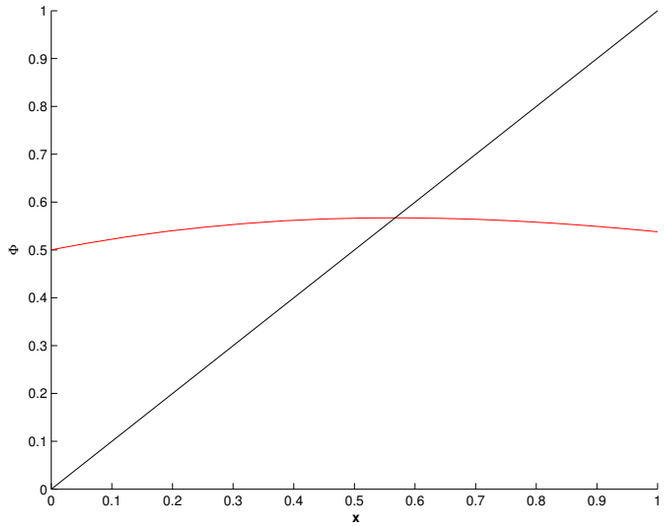


R. Hiptmair

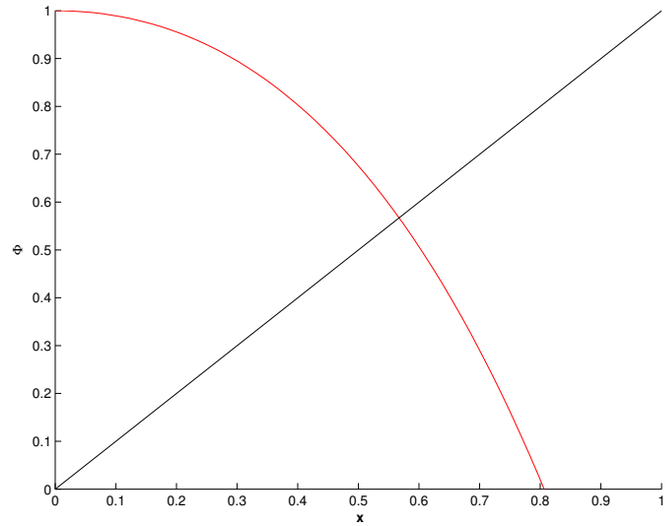
rev 30401, November 10, 2010



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

R. Hiptmair

rev 30401, November 10, 2010

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 $x_3^{(k)}$), $x_i^{(0)} = 0.5$.

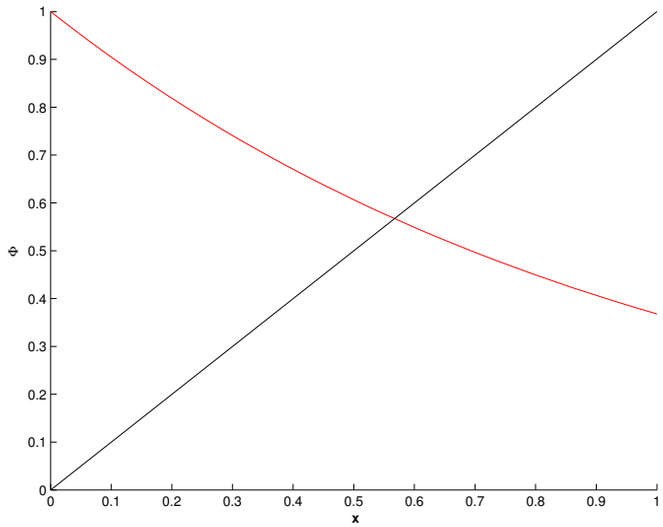
$\diamond$

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

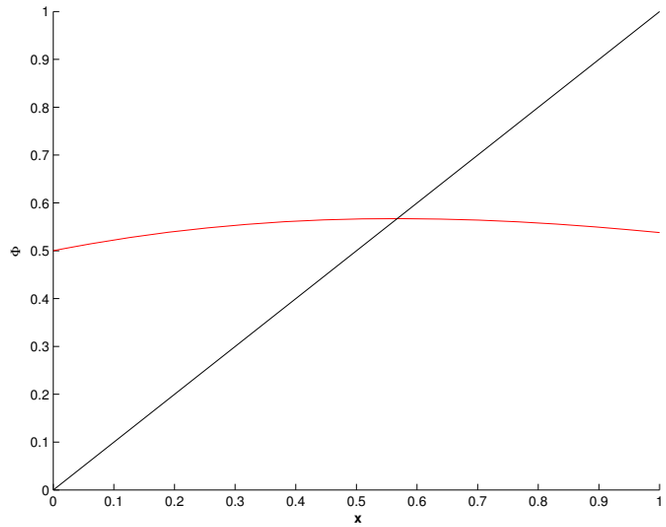
4.2.2 Convergence of fixed point iterations

In this section we will try to find easily verifiable conditions that ensure convergence (of a certain order) of fixed point iterations. It will turn out that these conditions are surprisingly simple and general.

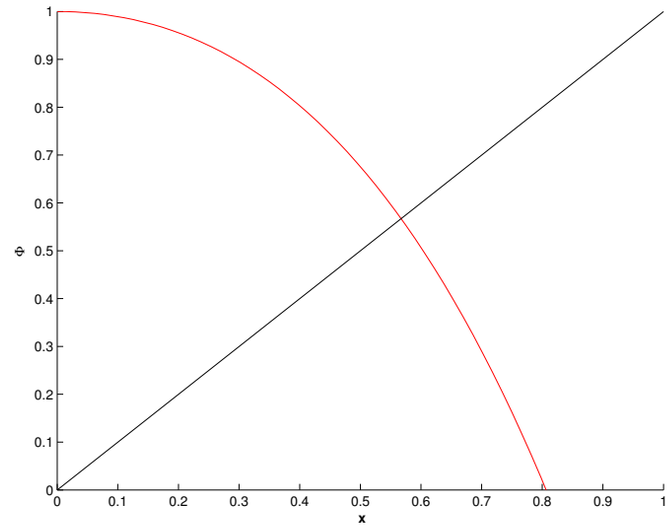
Ex. 4.2.3 revisited: vastly different behavior of different fixed point iterations for $n = 1$:



Φ_1: linear convergence ?



Φ_2: quadratic convergence ?



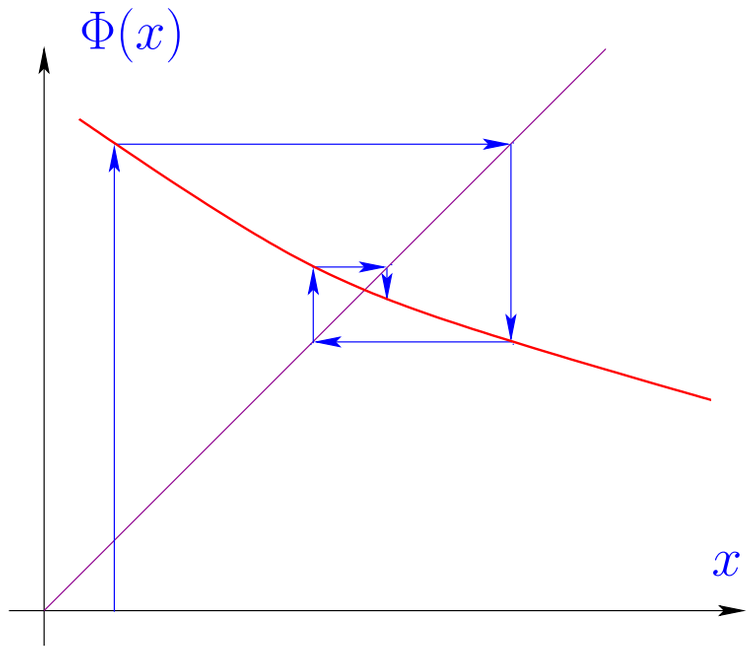
function Φ_3: no convergence

Example 4.2.4 (Fixed point iteration in 1D).

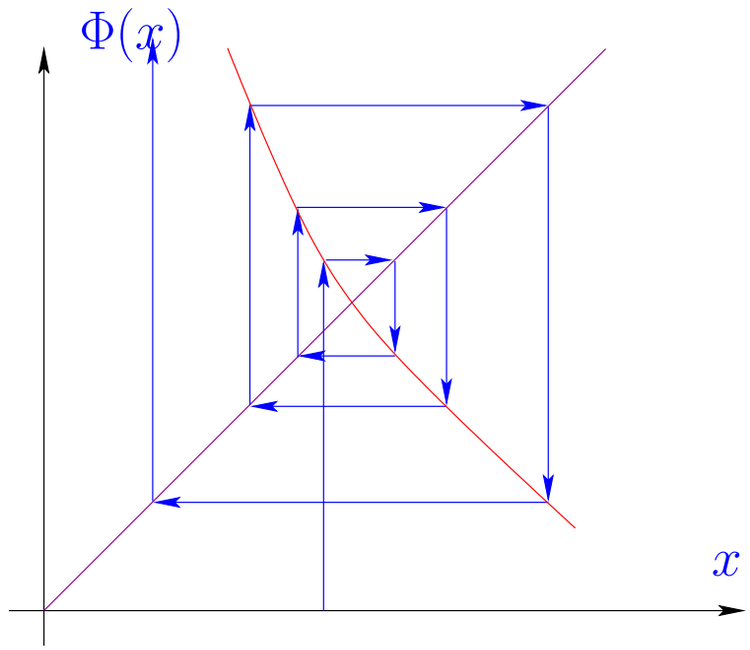
1D setting ($n = 1$): $\Phi : \mathbb{R} \mapsto \mathbb{R}$ continuously differentiable, $\Phi(x^*) = x^*$

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

Visualization of different convergence behavior of fixed point iterations:



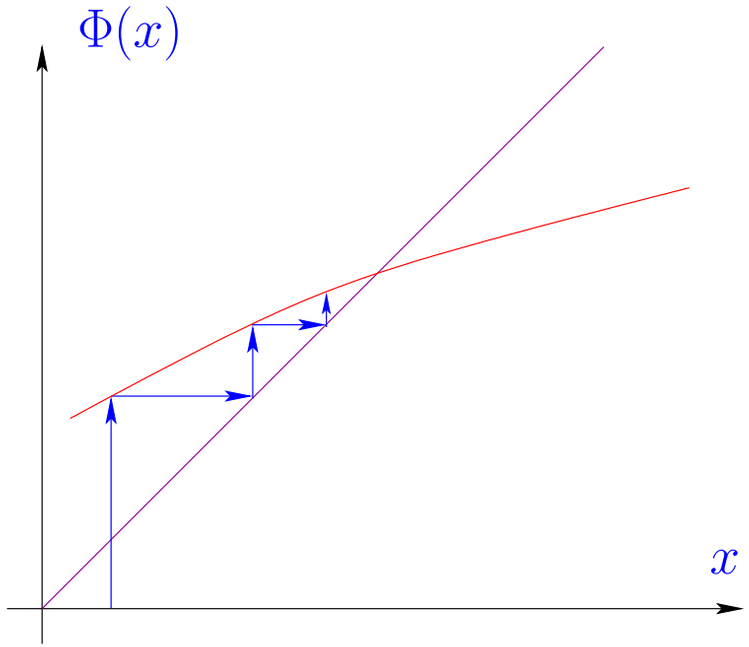
$-1 < \Phi'(x^*) \leq 0$ ➢ convergence



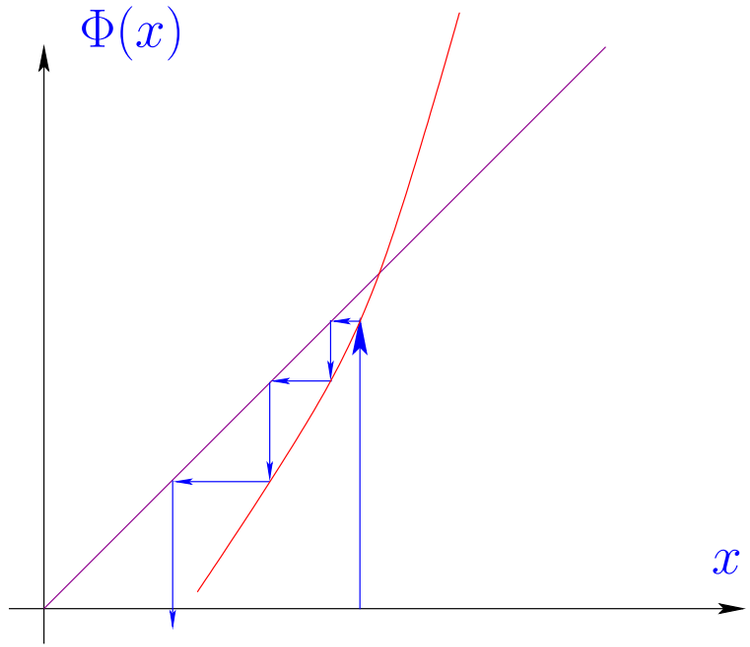
$\Phi'(x^*) < -1$ ➢ divergence

R. Hiptmair

rev 30401, November 10, 2010



$0 \leq \Phi'(x^*) < 1$ ➢ convergence



$1 < \Phi'(x^*)$ ➢ divergence

Numerical examples ➢ Ex. 4.2.3, iteration functions Φ_1 and Φ_3

◇

Definition 4.2.5 (Contractive mapping).

$\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ *is contractive (w.r.t. norm* $\|\cdot\|$ *on* $\mathbb{R}^n$*), if*

$$\exists L < 1: \quad \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in U \ . \tag{4.2.6}$$

A simple consideration: if $\Phi(\mathbf{x}^*) = \mathbf{x}^*$ (fixed point), then a fixed point iteration induced by a contractive mapping Φ satisfies

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| = \left\|\Phi(\mathbf{x}^{(k)}) - \Phi(\mathbf{x}^*)\right\| \overset{(4.2.8)}{\leq} L \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \ ,$$

that is, the iteration converges (at least) linearly ($\rightarrow$ Def. 4.1.6).

Note that

$$\Phi \text{ contractive} \quad \Rightarrow \quad \Phi \text{ has } \textit{at most one} \text{ fixed point.}$$

Remark 4.2.7 (Banach's fixed point theorem). → [52, Satz 6.5.2], [10, Satz 5.8]

A key theorem in calculus (also functional analysis):

Theorem 4.2.8 (Banach's fixed point theorem).
If $D \subset \mathbb{K}^n$ ($\mathbb{K} = \mathbb{R}, \mathbb{C}$) closed and bounded and $\Phi : D \mapsto D$ satisfies

$$\exists L < 1: \quad \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in D \ ,$$

then there is a unique fixed point $\mathbf{x}^* \in D$, $\Phi(\mathbf{x}^*) = \mathbf{x}^*$, which is the limit of the sequence of iterates $\mathbf{x}^{(k+1)} := \Phi(x^{(k)})$ for any $\mathbf{x}^{(0)} \in D$.

Proof. Proof based on 1-point iteration $\mathbf{x}^{(k)} = \Phi(\mathbf{x}^{(k-1)})$, $\mathbf{x}^{(0)} \in D$:

$$\begin{aligned} \left\|\mathbf{x}^{(k+N)} - \mathbf{x}^{(k)}\right\| &\leq \sum_{j=k}^{k+N-1} \left\|\mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}\right\| \leq \sum_{j=k}^{k+N-1} L^j \left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\| \\ &\leq \frac{L^k}{1-L} \left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\| \xrightarrow{k\to\infty} 0 \ . \end{aligned}$$

$(\mathbf{x}^{(k)})_{k\in\mathbb{N}_0}$ Cauchy sequence ➤ convergent $\mathbf{x}^{(k)} \xrightarrow{k\to\infty} \mathbf{x}^*$.

Continuity of Φ ➤ $\Phi(\mathbf{x}^*) = \mathbf{x}^*$. Uniqueness of fixed point is evident. □

A simple criterion for a differentiable Φ to be contractive:

Lemma 4.2.9 (Sufficient condition for local linear convergence of fixed point iteration)**.** → *[29, Thm. 17.2], [10, 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 $\mathbf{x} \in D$

→ [52, Sect. 7.6]

R. Hiptmair

rev 30401, November 10, 2010

"Visualization" of the statement of Lemma 4.2.9 in Ex. 4.2.4: The iteration converges *locally*, if Φ is flat in a neighborhood of x^*, it will diverge, if Φ is steep there.

Proof. (of Lemma 4.2.9) By definition of derivative

$$\|\Phi(\mathbf{y}) - \Phi(\mathbf{x}^*) - D\Phi(\mathbf{x}^*)(\mathbf{y}-\mathbf{x}^*)\| \leq \psi(\|\mathbf{y}-\mathbf{x}^*\|)\,\|\mathbf{y}-\mathbf{x}^*\| \ ,$$

with $\psi : \mathbb{R}_0^+ \mapsto \mathbb{R}_0^+$ satisfying $\lim_{t\to 0}\psi(t) = 0$.

Choose $\delta > 0$ such that

$$L := \psi(t) + \|D\Phi(\mathbf{x}^*)\| \leq \tfrac{1}{2}(1 + \|D\Phi(\mathbf{x}^*)\|) < 1 \quad \forall 0 \leq t < \delta \ .$$

By inverse triangle inequality we obtain for fixed point iteration

$$\begin{aligned} &\|\Phi(\mathbf{x}) - \mathbf{x}^*\| - \|D\Phi(\mathbf{x}^*)(\mathbf{x}-\mathbf{x}^*)\| \leq \psi(\|\mathbf{x}-\mathbf{x}^*\|)\,\|\mathbf{x}-\mathbf{x}^*\| \\ \blacktriangleright\quad &\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \leq (\psi(t) + \|D\Phi(\mathbf{x}^*)\|)\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \leq L\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \ , \end{aligned}$$

if $\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| < \delta$. □

R. Hiptmair

rev 30401, November 10, 2010

Lemma 4.2.10 (Sufficient condition for linear convergence of fixed point iteration)**.**
Let U be convex and $\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ be continuously differentiable with $L := \sup_{\mathbf{x}\in U} \|D\Phi(\mathbf{x})\| < 1$. If $\Phi(\mathbf{x}^*) = \mathbf{x}^*$ for some interior point $\mathbf{x}^* \in U$, then the fixed point iteration $\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)})$ converges to $\mathbf{x}^*$ at least linearly.

Recall: $U \subset \mathbb{R}^n$ convex $:\Leftrightarrow (t\mathbf{x} + (1-t)\mathbf{y}) \in U$ for all $\mathbf{x}, \mathbf{y} \in U$, $0 \le t \le 1$

R. Hiptmair

rev 30401, November 10, 2010

Proof. (of Lemma 4.2.10) By the mean value theorem

$$\Phi(\mathbf{x}) - \Phi(\mathbf{y}) = \int_0^1 D\Phi(\mathbf{x} + \tau(\mathbf{y}-\mathbf{x}))(\mathbf{y}-\mathbf{x})\,\mathrm{d}\tau \quad \forall \mathbf{x}, \mathbf{y} \in \mathrm{dom}(\Phi)\ .$$

$$\Rightarrow \quad \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \le L\,\|\mathbf{y}-\mathbf{x}\|\ ,$$

$$\Rightarrow \quad \left\|(\mathbf{x})^{(k+1)} - \mathbf{x}^*\right\| \le L\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|\ .$$

We find that Φ is contractive on U with unique fixed point $\mathbf{x}^*$, to which $\mathbf{x}^{(k)}$ converges linearly for $k \to \infty$.

Remark 4.2.11 (Bound for asymptotic rate of linear convergence).

If $0 < \|D\Phi(\mathbf{x}^*)\| < 1$, $\mathbf{x}^{(k)} \approx \mathbf{x}^*$ then the (worst) asymptotic rate of linear convergence is $L = \|D\Phi(x^*)\|$

△

Example 4.2.12 (Multidimensional fixed point iteration).

$$\begin{array}{ccc} \text{System of equations} & \text{in} & \text{fixed point form:} \\ \begin{cases} x_1 - c(\cos x_1 - \sin x_2) = 0 \\ (x_1 - x_2) - c\sin x_2 = 0 \end{cases} & \Rightarrow & \begin{cases} c(\cos x_1 - \sin x_2) = x_1 \\ c(\cos x_1 - 2\sin x_2) = x_2 \end{cases} . \end{array}$$

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

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.

The existence of a fixed point is also guaranteed, because Φ maps into the closed set $[-3,3]^2$. Thus, the Banach fixed point theorem, Thm. 4.2.8, can be applied.

◇

R. Hiptmair

rev 30401, November 10, 2010

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 4.2.13 (Taylor's formula). $\rightarrow$ *[52, Sect. 5.5]*

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

$$\Phi(y) - \Phi(x) = \sum_{k=1}^{m} \frac{1}{k!} \Phi^{(k)}(x)(y-x)^k + O(|y-x|^{m+1}) \quad \forall y \in U \; . \tag{4.2.14}$$

Apply Taylor expansion (4.2.14) to iteration function Φ:

If $\Phi(x^*) = x^*$ and $\Phi : \mathrm{dom}(\Phi) \subset \mathbb{R} \mapsto \mathbb{R}$ is "sufficiently smooth"

$$x^{(k+1)} - x^* = \Phi(x^{(k)}) - \Phi(x^*) = \sum_{l=1}^{m} \frac{1}{l!} \Phi^{(l)}(x^*)(x^{(k)} - x^*)^l + O(|x^{(k)} - x^*|^{m+1}) \; . \tag{4.2.15}$$

Lemma 4.2.16 (Higher order local convergence of fixed point iterations)**.**
If $\Phi : U \subset \mathbb{R} \mapsto \mathbb{R}$ is $m+1$ times continuously differentiable, $\Phi(x^*) = x^*$ for some x^* in the interior of U, and $\Phi^{(l)}(x^*) = 0$ for $l = 1, \ldots, m$, $m \geq 1$, then the fixed point iteration (4.2.2) *converges locally to x^* with order $\geq m+1$ ($\rightarrow$ Def. 4.1.14).*

Proof. For neighborhood $\mathcal{U}$ of x^*

$$\text{(4.2.15)} \quad \Rightarrow \quad \exists C > 0: \quad |\Phi(y) - \Phi(x^*)| \leq C\,|y - x^*|^{m+1} \quad \forall y \in \mathcal{U}\ .$$

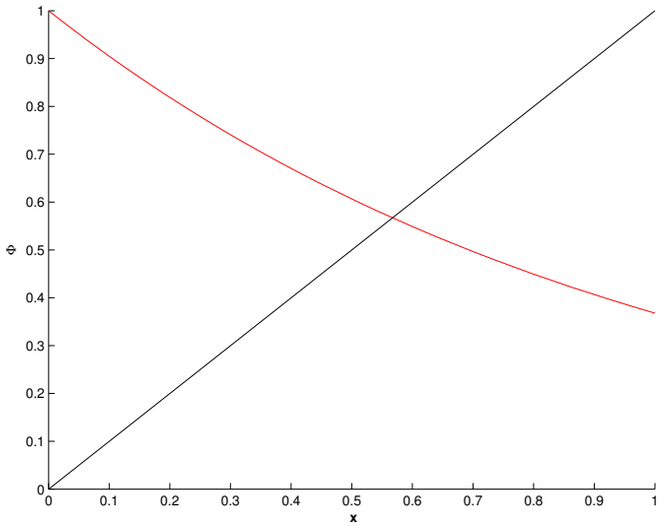
$$\delta^m C < 1/2: \quad |x^{(0)} - x^*| < \delta \quad \Rightarrow \quad |x^{(k)} - x^*| < 2^{-k}\delta \quad \succ \quad \text{local convergence}\ .$$

Then appeal to (4.2.15) □

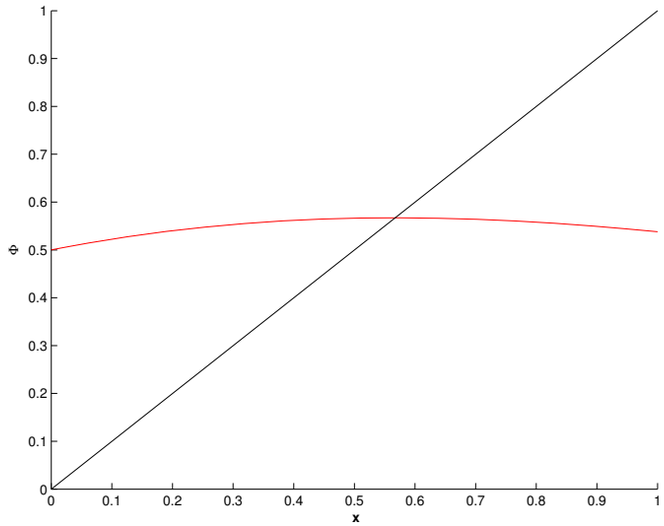
Example 4.2.3 continued:

R. Hiptmair

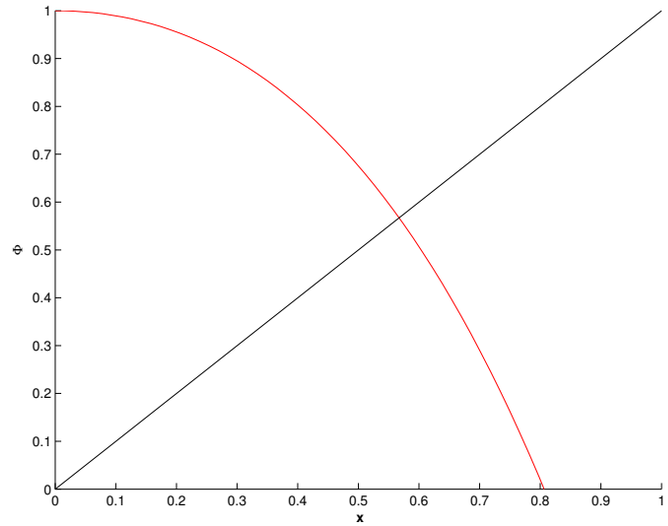
rev 30401, November 10, 2010



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 4.2.17 (Termination criterion for contractive fixed point iteration).

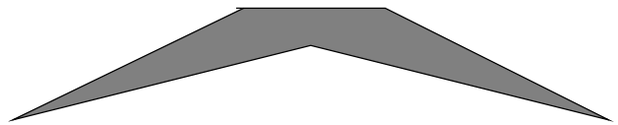
Recap of Rem. 4.1.23:

Termination criterion for contractive fixed point iteration, *c.f.* (4.2.8), with contraction factor $0 \leq L < 1$:

$$\left\|\mathbf{x}^{(k+m)} - \mathbf{x}^{(k)}\right\| \overset{\triangle\text{-ineq.}}{\leq} \sum_{j=k}^{k+m-1} \left\|\mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}\right\| \leq \sum_{j=k}^{k+m-1} L^{j-k} \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\|$$
$$= \frac{1-L^m}{1-L} \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\| \leq \frac{1-L^m}{1-L} L^{k-l} \left\|\mathbf{x}^{(l+1)} - \mathbf{x}^{(l)}\right\| .$$

hence for $m \to \infty$, with $\mathbf{x}^* := \lim_{k\to\infty} \mathbf{x}^{(k)}$:

$$\left\|\mathbf{x}^* - \mathbf{x}^{(k)}\right\| \leq \frac{L^{k-l}}{1-L} \left\|\mathbf{x}^{(l+1)} - \mathbf{x}^{(l)}\right\| . \tag{4.2.18}$$



R. Hiptmair

rev 30401, November 10, 2010

Set $l = 0$ in (4.2.18)

a priori termination criterion

$$\left\|\mathbf{x}^* - \mathbf{x}^{(k)}\right\| \leq \frac{L^k}{1-L}\left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\| \tag{4.2.19}$$

Set $l = k - 1$ in (4.2.18)

a posteriori termination criterion

$$\left\|\mathbf{x}^* - \mathbf{x}^{(k)}\right\| \leq \frac{L}{1-L}\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\| \tag{4.2.20}$$

△

4.3 Zero Finding

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

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

4.3.1 Bisection [10, 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 !)

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

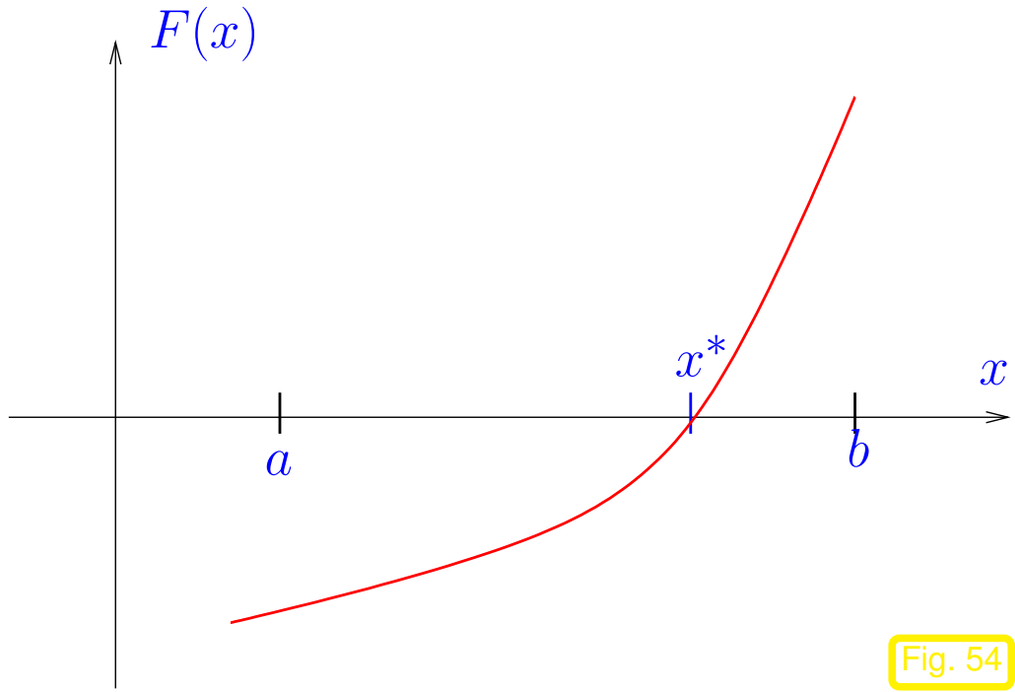


Fig. 54

Algorithm 4.3.1 (Bisection method).

R. Hiptmair

rev 30401, November 9, 2010

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")).

This is an example for an algorithm that (in the case of tol=0) uses the properties of machine arithmetic to define an a posteriori termination criterion, see Sect. 4.1.2. The iteration will terminate, when, e.g., $a \widetilde{+} \frac{1}{2}(b-a) = a$, which, by the Ass. 2.4.10 can only happen, when

$$|\tfrac{1}{2}(b-a)| \leq \texttt{eps} \cdot |a| \ .$$

Since the exact zero is located between a and b, this condition implies a relative error $\leq$ eps of the computed zero.

Advantages:

- “foolproof”
- requires only F evaluations

Drawbacks:

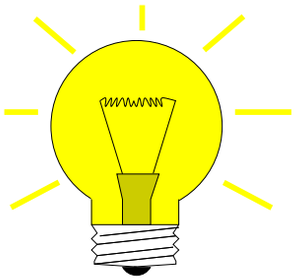
Merely linear convergence: $|x^{(k)} - x^*| \leq 2^{-k}|b-a|$

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

Remark 4.3.2. MATLAB function `fzero` is based on bisection approach. △

4.3.2 Model function methods

$\hat{=}$ class of iterative methods for finding zeroes of F:



Idea: Given: approximate zeroes $x^{(k)}, x^{(k-1)}, \ldots, x^{(k-m)}$

❶ replace F with model function $\widetilde{F}$
(using function values/derivative values in $x^{(k)}, x^{(k-1)}, \ldots, x^{(k-m)}$)

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 : $x^{(k+1)} = \Phi_F(x^{(k)}, x^{(k-1)}, \ldots, x^{(k-m)})$, $k \in \mathbb{N}$, $m = 2, 3, \ldots$.

4.3.2.1 Newton method in scalar case [29, Sect. 18.1], [10, Sect. 5.5.2]

Assume: $F : I \mapsto \mathbb{R}$ continuously differentiable

R. Hiptmair

rev 30401, November 9, 2010

model function:= tangent at F in $x^{(k)}$:

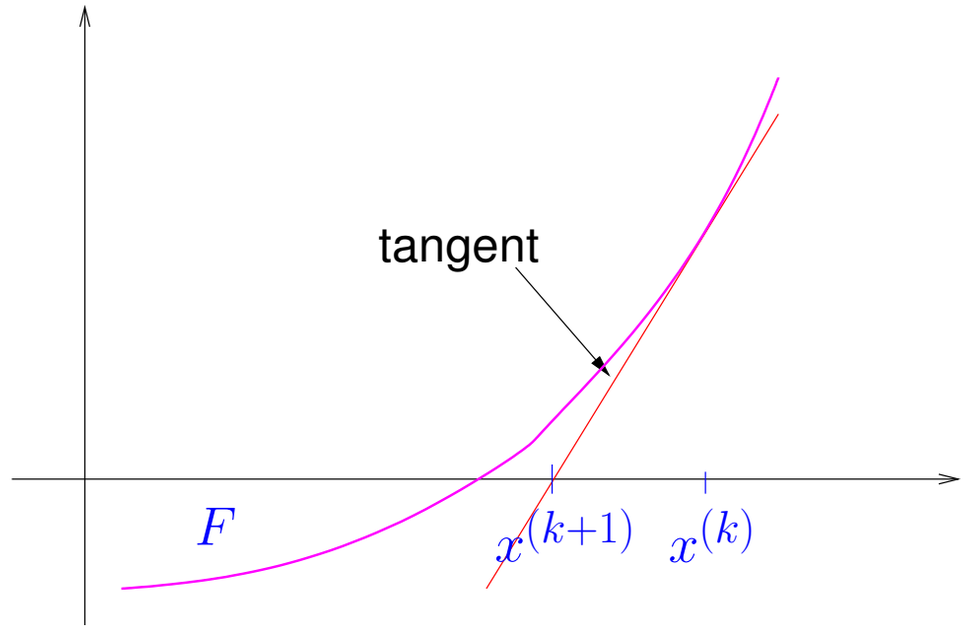
$$\widetilde{F}(x) := F(x^{(k)}) + F'(x^{(k)})(x - x^{(k)})$$

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

We obtain Newton iteration

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

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



Example 4.3.4 (Newton method in 1D). (→ Ex. 4.2.3)

Newton iterations for two different scalar non-linear equation with the same solution sets:

$$F(x) = xe^x - 1 \Rightarrow \quad F'(x) = e^x(1+x) \quad \Rightarrow \quad x^{(k+1)} = x^{(k)} - \frac{x^{(k)}e^{x^{(k)}} - 1}{e^{x^{(k)}}(1+x^{(k)})} = \frac{(x^{(k)})^2 + e^{-x^{(k)}}}{1+x^{(k)}}$$

$$F(x) = x - e^{-x} \quad \Rightarrow \quad F'(x) = 1 + e^{-x} \quad \Rightarrow \quad x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - e^{-x^{(k)}}}{1+e^{-x^{(k)}}} = \frac{1+x^{(k)}}{1+e^{x^{(k)}}}$$

R. Hiptmair

rev 30401, November 9, 2010

Ex. 4.2.3 shows quadratic convergence ! ($\rightarrow$ Def. 4.1.14)

Newton iteration (4.3.3) $\hat{=}$ fixed point iteration ($\rightarrow$ Sect.4.2) with iteration function

$$\Phi(x) = x - \frac{F(x)}{F'(x)} \quad \Rightarrow \quad \Phi'(x) = \frac{F(x)F''(x)}{(F'(x))^2} \quad \Rightarrow \quad \Phi'(x^*) = 0 \ , \quad \text{, if } F(x^*) = 0, \ F'(x^*) \neq 0 \ .$$

From Lemma 4.2.16:

R. Hiptmair

rev 30401, November 9, 2010

Newton method locally quadratically convergent ($\rightarrow$ Def. 4.1.14) to zero x^*, if $F'(x^*) \neq 0$

4.3.2.2 Special one-point methods

Idea underlying other one-point methods: non-linear local approximation

Useful, if *a priori knowledge* about the structure of F (e.g. about F being a rational function, see below) is available. This is often the case, because many problems of 1D zero finding are posed for functions given in analytic form with a few parameters.

Prerequisite: Smoothness of F: $F \in C^m(I)$ for some $m > 1$

Example 4.3.5 (Halley's iteration). $\rightarrow$ [29, Sect. 18.3]

This example demonstrates that non-polynomial model functions can offer excellent approximation of F. In this example the model function is chosen as a quotient of two linear function, that is, from the simplest class of true rational functions.

Of course, that this function provides a good model function is merely "a matter of luck", unless you have some more information about F. Such information might be available from the application context.

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

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

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

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

Halley's iteration for $F(x) = \dfrac{1}{(x+1)^2} + \dfrac{1}{(x+0.1)^2} - 1 \;, \quad x > 0 :$ 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

R. Hiptmair

rev 30401, November 9, 2010

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) ◇

R. Hiptmair

rev 30401, November 9, 2010

In the previous example Newton's method performed rather poorly. Often its convergence can be boosted by converting the non-linear equation to an equivalent one (that is, one with the same solutions) for another function g, which is "closer to a linear function":

Assume $F \approx \widehat{F}$, where $\widehat{F}$ is invertible with an inverse $\widehat{F}^{-1}$ that can be evaluated with little effort.

$$\blacktriangleright \quad g(x) := \widehat{F}^{-1}(F(x)) \approx x \ .$$

Then apply Newton's method to $g(x)$, using the formula for the derivative of the inverse of a function

$$\frac{d}{dy}(\widehat{F}^{-1})(y) = \frac{1}{\widehat{F}'(\widehat{F}^{-1}(y))} \quad \Rightarrow \quad g'(x) = \frac{1}{\widehat{F}'(g(x))} \cdot F'(x) \ .$$

Example 4.3.6 (Adapted Newton method).

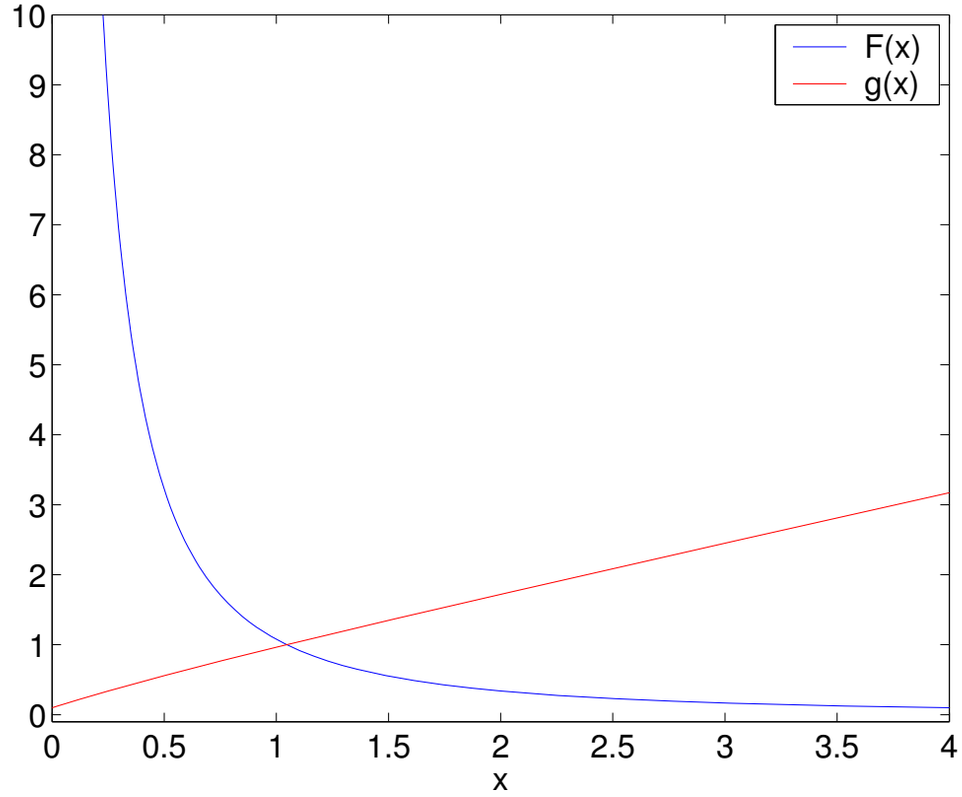
As in Ex. 4.3.5:

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

Observation:

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

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



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

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

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

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

R. Hiptmair

rev 30401, November 9, 2010

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

$\diamond$

For zero finding there is wealth of iterative methods that offer higher order of convergence.

One idea: consistent modification of the Newton-Iteration:

► fixed point iteration : $\Phi(x) = x - \dfrac{F(x)}{F'(x)} H(x)$ with "proper" $H : I \mapsto \mathbb{R}$.

Aim: find H such that the method is of p-th order; tool: Lemma 4.2.16.

Assume: F smooth "enough" and $\exists\, x^* \in I$: $F(x^*) = 0$, $F'(x^*) \neq 0$.

$$\Phi = x - uH \quad , \quad \Phi' = 1 - u'H - uH' \quad , \quad \Phi'' = -u''H - 2u'H - uH'' \,,$$

R. Hiptmair

rev 30401, November 9, 2010

$$\text{with } u = \frac{F}{F'} \quad \Rightarrow \quad u' = 1 - \frac{FF''}{(F')^2} \quad , \quad u'' = -\frac{F''}{F'} + 2\frac{F(F'')^2}{(F')^3} - \frac{FF'''}{(F')^2} \, .$$

$F(x^*) = 0 \quad \blacktriangleright \quad u(x^*) = 0, \, u'(x^*) = 1, \, u''(x^*) = -\frac{F''(x^*)}{F'(x^*)}.$

$$\blacktriangleright \quad \Phi'(x^*) = 1 - H(x^*) \quad , \quad \Phi''(x^*) = \frac{F''(x^*)}{F'(x^*)} H(x^*) - 2H'(x^*) \, . \tag{4.3.7}$$

Lemma 4.2.16 $\succ$ **Necessary** conditions for local convergence of order p:

$p = 2$ (quadratical convergence): $H(x^*) = 1$,

$p = 3$ (cubic convergence): $H(x^*) = 1 \quad \wedge \quad H'(x^*) = \frac{1}{2}\frac{F''(x^*)}{F'(x^*)}$.

In particular: $H(x) = G(1 - u'(x))$ with "proper" G

$$\blacktriangleright \quad \text{fixed point iteration} \quad x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})} G\left(\frac{F(x^{(k)})F''(x^{(k)})}{(F'(x^{(k)}))^2} \right) \, . \tag{4.3.8}$$

Lemma 4.3.9. *If $F \in C^2(I)$, $F(x^*) = 0$, $F'(x^*) \neq 0$, $G \in C^2(U)$ in a neighbourhood U of 0, $G(0) = 1$, $G'(0) = \frac{1}{2}$, then the fixed point iteration* (4.3.8) *converge locally cubically to x^*.*

Proof: Lemma 4.2.16, (4.3.7) and

$$H(x^*) = G(0) \quad , \quad H'(x^*) = -G'(0)u''(x^*) = G'(0)\frac{F''(x^*)}{F'(x^*)} \; .$$

Example 4.3.10 (Example of modified Newton method).

- $G(t) = \dfrac{1}{1-\frac{1}{2}t}$ ➡ Halley's iteration ($\rightarrow$ Ex. 4.3.5)
- $G(t) = \dfrac{2}{1+\sqrt{1-2t}}$ ➡ Euler's iteration
- $G(t) = 1+\frac{1}{2}t$ ➡ quadratic inverse interpolation

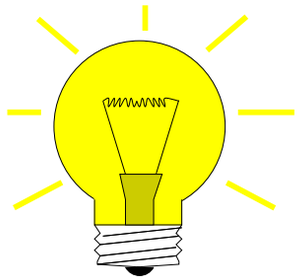
Numerical experiment:

$$F(x) = xe^x - 1 \; ,$$
$$x^{(0)} = 5$$

k	$e^{(k)} := x^{(k)} - x^*$		
	Halley	Euler	Quad. Inv.
1	2.81548211105635	3.57571385244736	2.03843730027891
2	1.37597082614957	2.76924150041340	1.02137913293045
3	0.34002908011728	1.95675490333756	0.28835890388161
4	0.00951600547085	1.25252187565405	0.01497518178983
5	0.00000024995484	0.51609312477451	0.00000315361454

R. Hiptmair

4.3.2.3 Multi-point methods



Idea: Replace F with interpolating polynomial

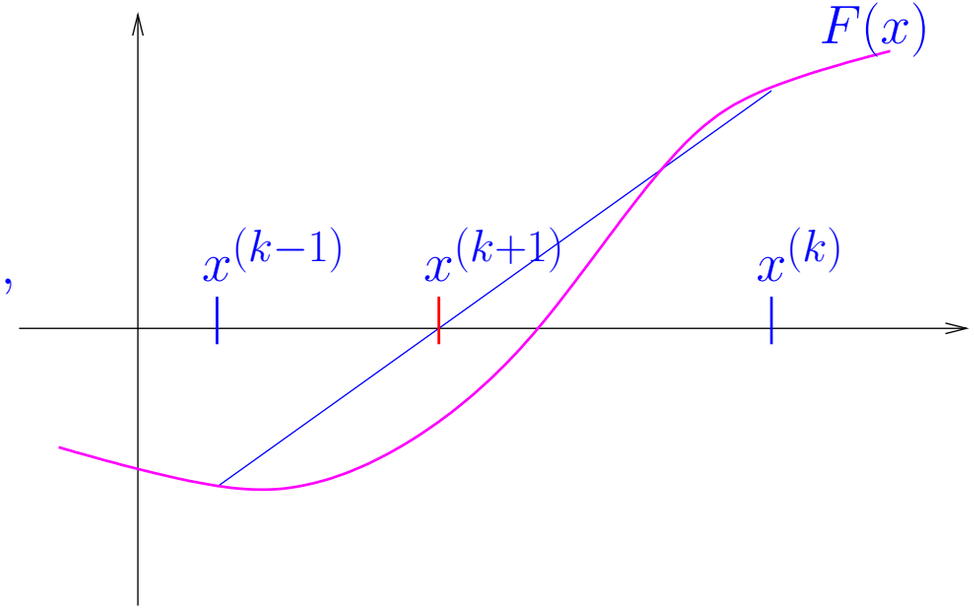
producing interpolatory model function methods

Simplest representative: secant method $\rightarrow$ [29, Sect. 18.2], [10, 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

```
1 function x = secant(x0,x1,F,tol)
2 fo = F(x0);
3 for i=1:MAXIT
4   fn = F(x1);
5   s = fn*(x1-x0)/(fn-fo);
6   x0 = x1; x1 = x1-s;
7   if (abs(s) < tol), x = x1;
      return; end
8   fo = fn;
9 end
```

R. Hiptmair

rev 30401, November 9, 2010

Remember: $F(x)$ may only be available as output of a (complicated) procedure. In this case it is difficult to find a procedure that evaluates $F'(x)$. Thus the significance of methods that do not involve evaluations of derivatives.

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

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

A startling observation: the method seems to have a *fractional* (!) order of convergence, see Def. 4.1.14.

◇

Remark 4.3.15 (Fractional order of convergence of secant method).

Indeed, a fractional order of convergence can be proved for the secant method, see[29, Sect. 18.2]. Here is a crude outline of the reasoning:

Asymptotic convergence of secant method: error $e^{(k)} := x^{(k)} - x^*$

$$e^{(k+1)} = \Phi(x^* + e^{(k)}, x^* + e^{(k-1)}) - x^* \quad \text{, with} \quad \Phi(x,y) := x - \frac{F(x)(x-y)}{F(x)-F(y)} \,. \tag{4.3.16}$$

Use MAPLE to find Taylor expansion (assuming F sufficiently smooth):

```
> Phi := (x,y) -> x-F(x)*(x-y)/(F(x)-F(y));
> F(s) := 0;
> e2 = normal(mtaylor(Phi(s+e1,s+e0)-s,[e0,e1],4));
```

➢ linearized error propagation formula:

$$e^{(k+1)} \doteq \frac{1}{2}\frac{F''(x^*)}{F'(x^*)} e^{(k)} e^{(k-1)} = C e^{(k)} e^{(k-1)} \,. \tag{4.3.17}$$

Try $e^{(k)} = K(e^{(k-1)})^p$ to determine the order of convergence ($\rightarrow$ Def. 4.1.14):

$$\Rightarrow \quad e^{(k+1)} = K^{p+1}(e^{(k-1)})^{p^2}$$

$$\Rightarrow \quad (e^{(k-1)})^{p^2-p-1} = K^{-p}C \quad \Rightarrow \quad p^2 - p - 1 = 0 \quad \Rightarrow \quad p = \tfrac{1}{2}(1 \pm \sqrt{5}) \; .$$

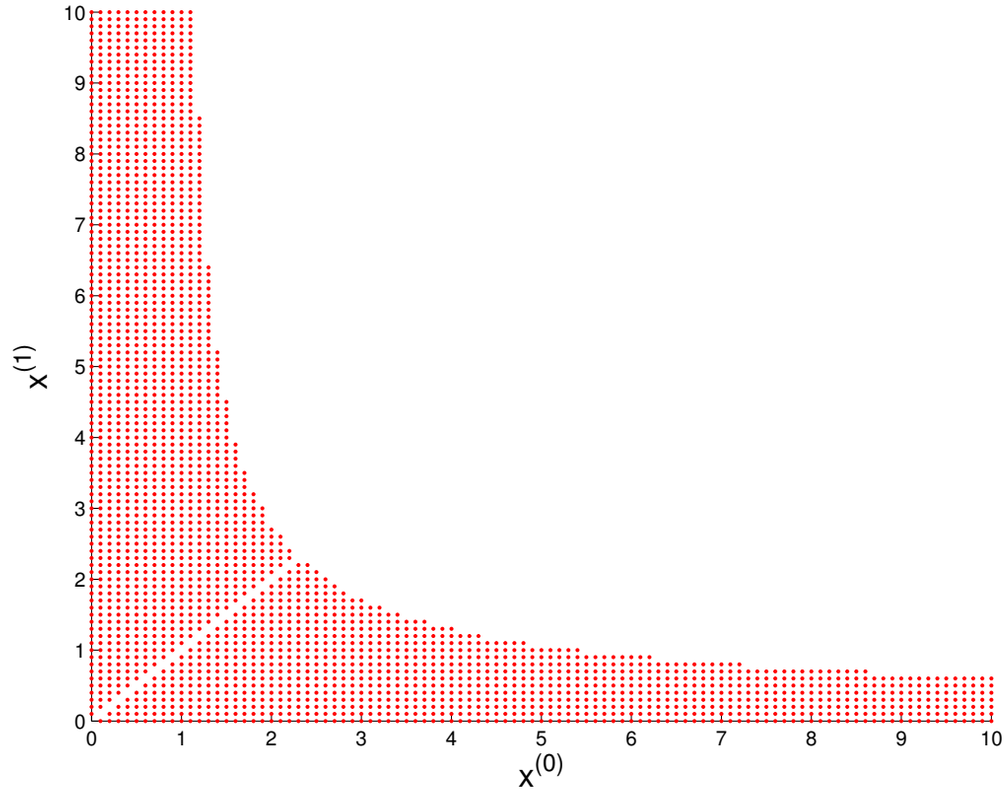
As $e^{(k)} \to 0$ for $k \to \infty$ we get the order of convergence $p = \frac{1}{2}(1+\sqrt{5}) \approx 1.62$ (see Ex. 4.3.14 !)

Example 4.3.18 (local convergence of secant method).

$$F(x) = \arctan(x)$$

- $\hat{=}$ secant method converges for a pair $(x^{(0)}, x^{(1)})$ of initial guesses.

= local convergence $\rightarrow$ Def. 4.1.5



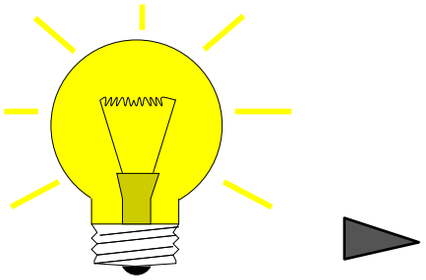
R. Hiptmair

rev 30401, November 9, 2010

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

$$p(F(x^{(k-m)})) = x^{(k-m)} \ , \quad m = 0, \ldots, d \ .$$

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

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

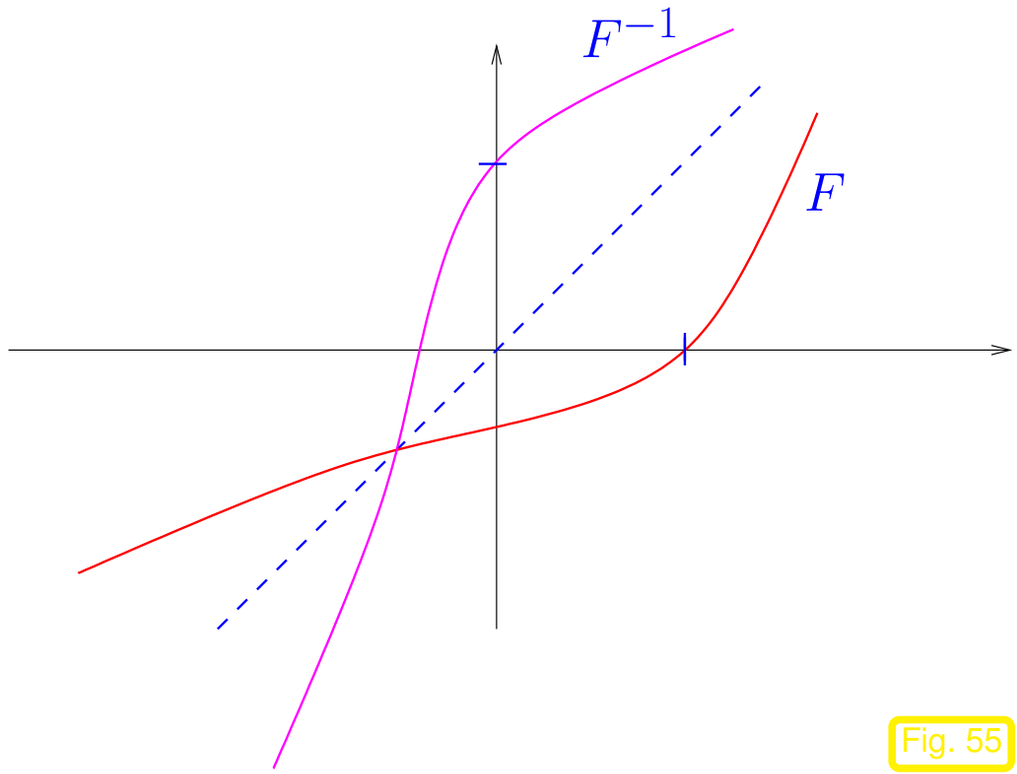


Fig. 55

R. Hiptmair

rev 30401, November 9, 2010

Case $m = 1$ ➢ secant method

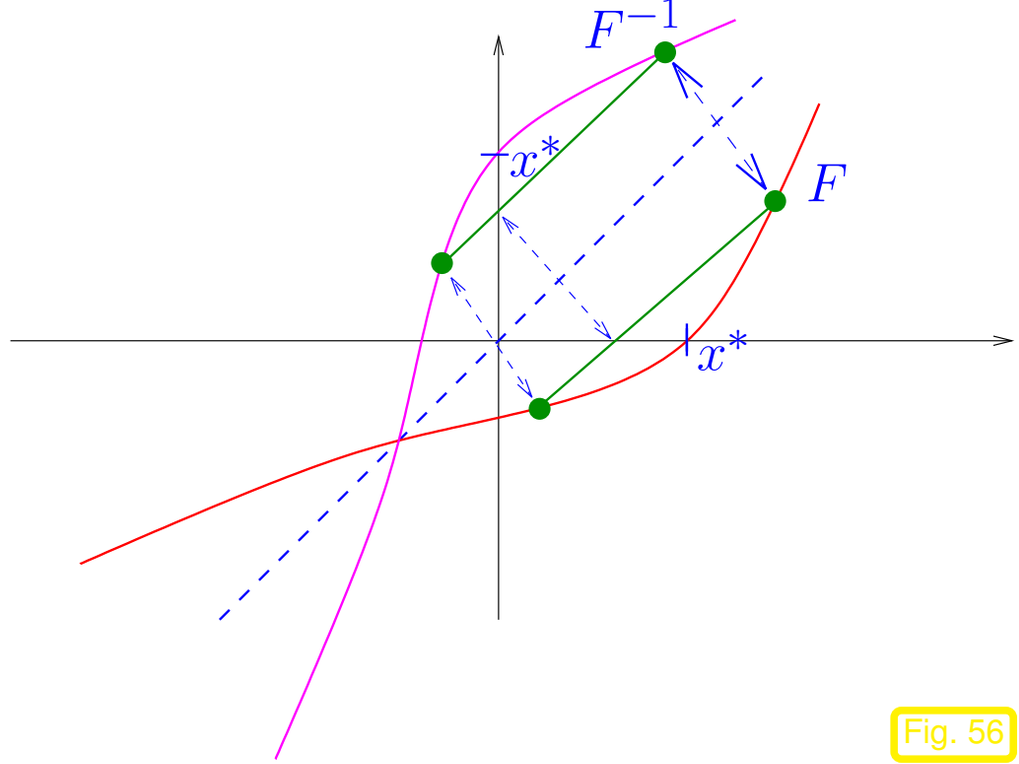


Fig. 56

Case $m = 2$: quadratic inverse interpolation, see [36, 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);
```

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

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

R. Hiptmair

rev 30401, November 9, 2010

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	

Also in this case the numerical experiment hints at a fractional rate of convergence, as in the case of the secant method, see Rem. 4.3.15.

◇

R. Hiptmair

rev 30401, November 9, 2010

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: $W \mathrel{\hat{=}}$ 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*

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

R. Hiptmair

rev 30401, November 9, 2010

Error recursion can be converted into expressions (4.3.21) and (4.3.22) that related the error norm $\left\|\mathbf{e}^{(k)}\right\|$ to $\left\|\mathbf{e}^{(0)}\right\|$ and lead to quantitative bounds for the number of steps to achieve (4.3.20) for iterative method of order p, $p \geq 1$ ($\rightarrow$ Def. 4.1.14):

$$\exists C > 0: \quad \left\|e^{(k)}\right\| \leq C \left\|e^{(k-1)}\right\|^p \quad \forall k \geq 1 \quad (C < 1 \text{ for } p = 1) \ .$$

Assuming $C \left\|e^{(0)}\right\|^{p-1} < 1$ (guarantees convergence):

$$p = 1: \qquad \left\|e^{(k)}\right\| \overset{!}{\leq} C^k \left\|e^{(0)}\right\| \quad \text{requires} \quad k \geq \frac{\log \rho}{\log C} \ , \tag{4.3.21}$$

$$p > 1: \quad \left\|e^{(k)}\right\| \overset{!}{\leq} C^{\frac{p^k-1}{p-1}} \left\|e^{(0)}\right\|^{p^k} \quad \text{requires} \quad p^k \geq 1 + \frac{\log \rho}{\log C/p-1 + \log(\left\|e^{(0)}\right\|)}$$

$$\Rightarrow \quad k \geq \log(1 + \frac{\log \rho}{\log L_0})/\log p \ , \tag{4.3.22}$$

$$L_0 := C^{1/p-1} \left\|e^{(0)}\right\| < 1 \ .$$

If $\rho \ll 1$, then $\log(1 + \frac{\log \rho}{\log L_0}) \approx \log|\log \rho| - \log|\log L_0| \approx \log|\log \rho|$. This simplification will be made in the context of asymptotic considerations $\rho \to 0$ below.

R. Hiptmair

rev 30401, November 9, 2010

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

Measure for efficiency:

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

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

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

Example 4.3.25 (Efficiency of iterative methods).

R. Hiptmair

rev 30401, November 9, 2010

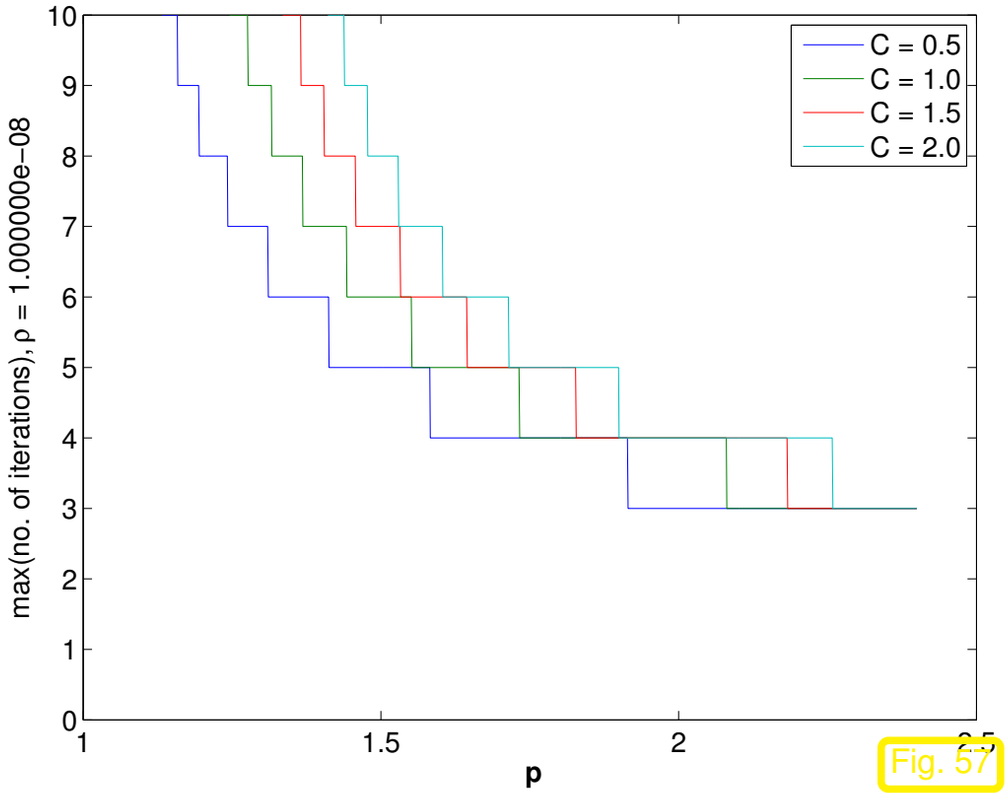


Fig. 57

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

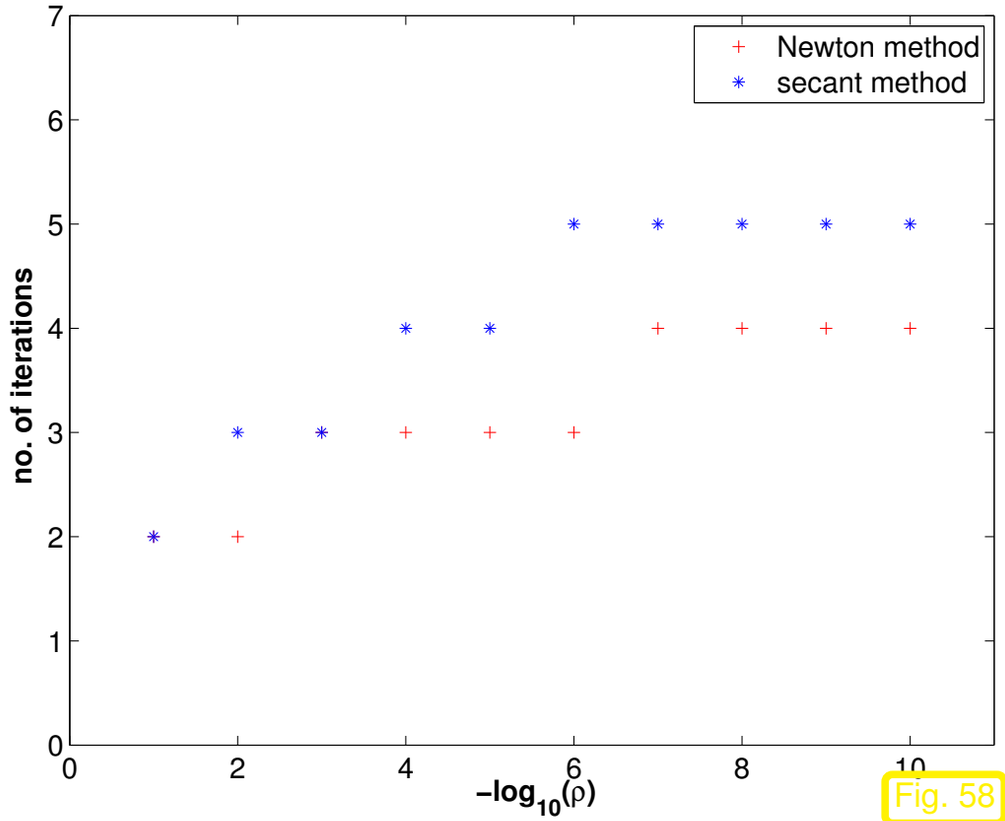


Fig. 58

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! ◇

R. Hiptmair

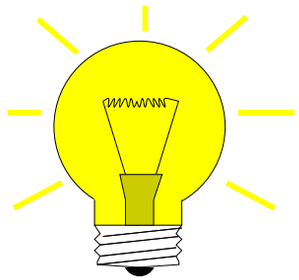
rev 30401, November 9, 2010

4.4 Newton's Method [29, Sect. 19], [10, 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

R. Hiptmair

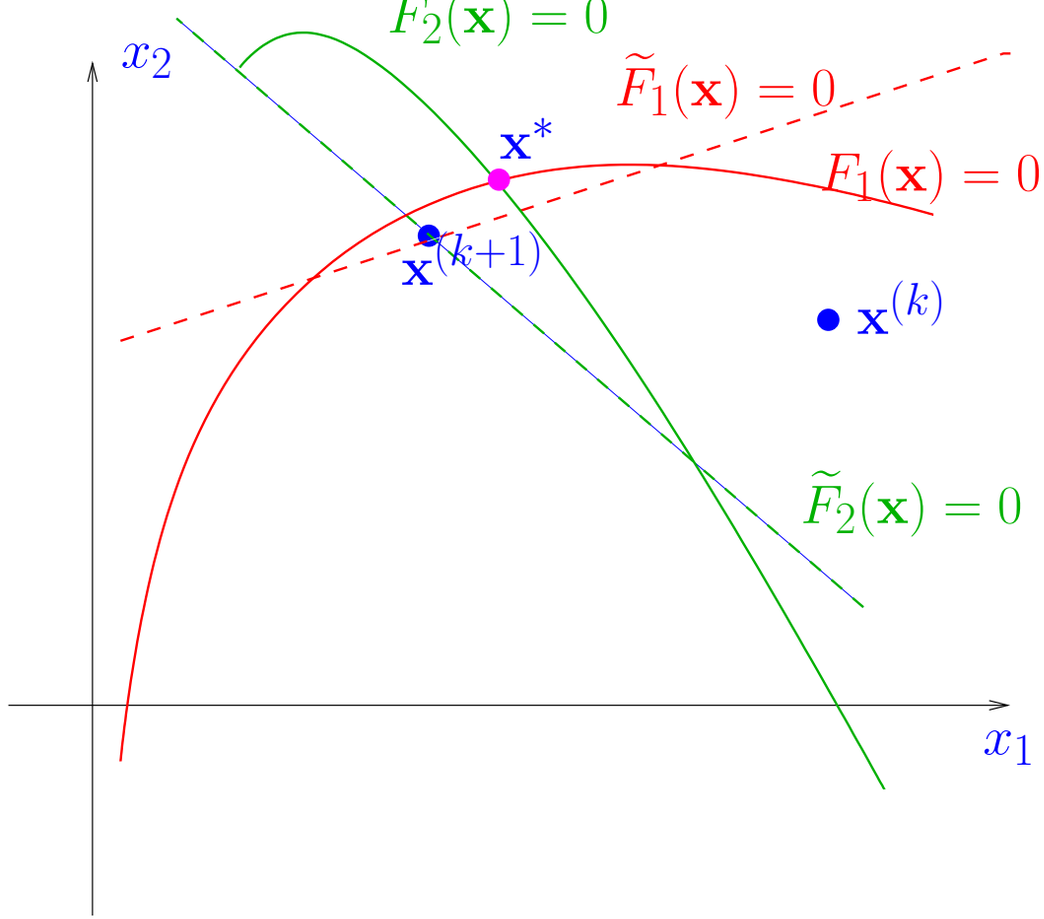
rev 30401, November 9, 2010

Illustration of idea of Newton's method for $n = 2$:

▷

Sought: intersection point $\mathbf{x}^*$ of the curves $F_1(\mathbf{x}) = 0$ and $F_2(\mathbf{x}) = 0$.

Idea: $\mathbf{x}^{(k+1)}$ = the intersection of two straight lines (= zero sets of the components of the model function, *cf.* Ex. 2.5.32) that are approximations of the original curves



R. Hiptmair

rev 30401, November 9, 2010

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,tol)
for i=1:MAXIT
  s = DF(x) \ F(x);
x = x-s;
  if (norm(s) < tol*norm(x))
    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 is eventually costly!

Remark 4.4.2 (Affine invariance of Newton method).

An important property of the Newton iteration (4.4.1): affine invariance $\rightarrow$ [12, 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 $\mathbf{A}$!

R. Hiptmair

rev 30401, November 9, 2010

This is a simple computation:

$$DG(\mathbf{x}) = \mathbf{A}DF(\mathbf{x}) \quad \Rightarrow \quad DG(\mathbf{x})^{-1}G(\mathbf{x}) = DF(\mathbf{x})^{-1}\mathbf{A}^{-1}\mathbf{A}F(\mathbf{x}) = DF(\mathbf{x})^{-1}F(\mathbf{x}) \,.$$

Use affine invariance as guideline for

- convergence theory for Newton's method: assumptions and results should be affine invariant, too.
- modifying and extending Newton's method: resulting schemes should preserve affine invariance.

△

Remark 4.4.3 (Differentiation rules). → Repetition: basic analysis

Statement of the Newton iteration (4.4.1) for $F : \mathbb{R}^n \mapsto \mathbb{R}^n$ given as analytic expression entails computing the Jacobian DF. To avoid cumbersome component-oriented considerations, it is useful to know the *rules of multidimensional differentiation*:

Let V, W be finite dimensional vector spaces, $F : D \subset V \mapsto W$ sufficiently smooth. The differential $DF(\mathbf{x})$ of F in $\mathbf{x} \in V$ is the *unique*

$$\text{linear mapping} \quad DF(\mathbf{x}) : V \mapsto W \ ,$$
$$\text{such that} \quad \|F(\mathbf{x}+\mathbf{h}) - F(\mathbf{x}) - DF(\mathbf{h})\mathbf{h}\| = o(\|\mathbf{h}\|) \quad \forall \mathbf{h} \ , \ \|\mathbf{h}\| < \delta \ .$$

- For $F : V \mapsto W$ linear, i.e. $F(\mathbf{x}) = \mathbf{A}\mathbf{x}$, $\mathbf{A}$ matrix ➤ $DF(\mathbf{x}) = \mathbf{A}$.
- Chain rule: $F : V \mapsto W$, $G : W \mapsto U$ sufficiently smooth

$$D(G \circ F)(\mathbf{x})\mathbf{h} = DG(F(\mathbf{x}))(DF(\mathbf{x}))\mathbf{h} \ , \quad \mathbf{h} \in V \ , \mathbf{x} \in D \ . \tag{4.4.4}$$

- Product rule: $F : D \subset V \mapsto W$, $G : D \subset V \mapsto U$ sufficiently smooth, $b : W \times U \mapsto Z$ bilinear

$$T(\mathbf{x}) = b(F(\mathbf{x}), G(\mathbf{x})) \quad \Rightarrow \quad DT(\mathbf{x})\mathbf{h} = b(DF(\mathbf{x})\mathbf{h}, G(\mathbf{x})) + b(F(\mathbf{x}), DG(\mathbf{x})\mathbf{h}) \ , \quad \mathbf{h} \in V, \mathbf{x} \in D \ . \tag{4.4.5}$$

For $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}$ the gradient $\mathbf{grad}\, F : D \mapsto \mathbb{R}^n$, and the Hessian matrix $HF(\mathbf{x}) : D \mapsto \mathbb{R}^{n,n}$ are defined as

$$\mathbf{grad}\, F(\mathbf{x})^T\mathbf{h} := DF(\mathbf{x})\mathbf{h} \ , \qquad \mathbf{h}_1^T HF(\mathbf{x})\mathbf{h}_2 := D(DF(\mathbf{x})(\mathbf{h}_1))(\mathbf{h}_2) \ , \quad \mathbf{h}, \mathbf{h}_1, \mathbf{h}_2 \in V \ .$$

R. Hiptmair

rev 30401, November 9, 2010

A simple example: $\Psi : \mathbb{R}^n \mapsto \mathbb{R}$, $\Psi(\mathbf{x}) := \mathbf{x}^T\mathbf{A}\mathbf{x}$, with $\mathbf{A} \in \mathbb{R}^{n,n}$

We apply the product rule (4.4.5) with $F, G = Id$, which means $DF(\mathbf{x}) = DG(\mathbf{x}) = \mathbf{I}$, and the *bilinear form* $b(\mathbf{x}, \mathbf{y}) := \mathbf{x}^T\mathbf{A}\mathbf{y}$:

$$\blacktriangleright \quad D\Psi(\mathbf{x})\mathbf{h} = \mathbf{h}^T\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{A}\mathbf{h} = \underbrace{\left(\mathbf{x}^T\mathbf{A}^T + \mathbf{x}^T\mathbf{A}\right)}_{=(\mathbf{grad}\,\Psi(\mathbf{x}))^T}\mathbf{h} \,.$$

Remark 4.4.6 (Simplified Newton method). [10, 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

```
1 function x = simpnewton(x,F,DF,tol)
2 % MATLAB template for simplified Newton method
3 [L,U] = lu(DF(x)); % one LU-decomposition
4 s = U\(L\F(x)); x = x-s;
5 while (norm(s) > tol*norm(x)) % termination
6   s = U\(L\F(x)); x = x-s;
7 end
```

Note: resuse of LU-decomposition, *cf.* Rem. 2.2.12

➢ (usually) merely linear convergence instead of quadratic convergence

△

Remark 4.4.8 (Numerical Differentiation for computation of Jacobian).

If $DF(\mathbf{x})$ is not available (e.g. when $F(\mathbf{x})$ is given only as a procedure):

Numerical Differentiation: $$\frac{\partial F_i}{\partial x_j}(\mathbf{x}) \approx \frac{F_i(\mathbf{x} + h\vec{e}_j) - F_i(\mathbf{x})}{h} .$$

Caution: impact of roundoff errors for small h **!** $\rightarrow$ Ex. 3.4.9

△

4.4.2 Convergence of Newton's method

Newton iteration (4.4.1) $\hat{=}$ fixed point iteration ($\rightarrow$ Sect. 4.2) with

$$\Phi(\mathbf{x}) = \mathbf{x} - DF(\mathbf{x})^{-1}F(\mathbf{x}) .$$

$$\left[\text{“product rule”}: \quad D\Phi(\mathbf{x}) = \mathbf{I} - D(DF(\mathbf{x})^{-1})F(\mathbf{x}) - DF(\mathbf{x})^{-1}DF(\mathbf{x}) \quad \right]$$

$$F(\mathbf{x}^*) = 0 \quad \Rightarrow \quad D\Phi(\mathbf{x}^*) = 0\ .$$

4.2.16

Local quadratic convergence of Newton's method, if $DF(\mathbf{x}^*)$ regular

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

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

Jacobian (analytic computation): $$DF(\mathbf{x}) = \begin{pmatrix} \partial_{x_1}F_1(x) & \partial_{x_2}F_1(x) \\ \partial_{x_1}F_2(x) & \partial_{x_2}F_2(x) \end{pmatrix} = \begin{pmatrix} 2x_1 & -4x_2^3 \\ 1 & -3x_2^2 \end{pmatrix}$$

R. Hiptmair

rev 30401, November 9, 2010

Realization of Newton iteration (4.4.1):

1. Solve LSE

$$\begin{pmatrix} 2x_1 & -4x_2^3 \\ 1 & -3x_2^2 \end{pmatrix} \Delta\mathbf{x}^{(k)} = -\begin{pmatrix} x_1^2 - x_2^4 \\ x_1 - x_2^3 \end{pmatrix},$$

where $\mathbf{x}^{(k)} = (x_1, x_2)^T$.

2. Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}$

Lines 14, 15: estimation of order of convergence, see Rem. 4.1.15.

Code 4.4.10: Newton iteration in 2D

```
1  F=@(x) [x(1)^2-x(2)^4; x(1)-x(2)^3];
2  DF=@(x) [2*x(1),-4*x(2)^3;1,-3*x(2)^2];
3  x=[0.7;0.7]; x_ast=[1;1]; tol=1E-10;
4
5  res=[0,x',norm(x-x_ast)];
6  s = DF(x)\F(x); x = x-s;
7  res = [res; 1,x',norm(x-x_ast)]; k=2;
8  while (norm(s) > tol*norm(x))
9    s = DF(x)\F(x); x = x-s;
10   res = [res; k,x',norm(x-x_ast)];
11   k = k+1;
12 end
13
14 ld = diff(log(res(:,4))); %
15 rates = ld(2:end)./ld(1:end-1); %
```

R. Hiptmair

rev 30401, November 9, 2010

k	$\mathbf{x}^{(k)}$	$\epsilon_k := \|\mathbf{x}^* - \mathbf{x}^{(k)}\|_2$	$\frac{\log \epsilon_{k+1} - \log \epsilon_k}{\log \epsilon_k - \log \epsilon_{k-1}}$
0	$(0.7,\ 0.7)^T$	4.24e-01	
1	$(0.878500000000000,\ 1.064285714285714)^T$	1.37e-01	1.69
2	$(1.018159432741880,\ 1.009148824639360)^T$	2.03e-02	2.23
3	$(1.000233559163000,\ 1.000159139360750)^T$	2.83e-04	2.15
4	$(1.000000005838520,\ 1.000000027265520)^T$	2.79e-08	1.77
5	$(0.999999999999980,\ 1.000000000000000)^T$	2.11e-15	
6	$(1,\ 1)^T$		

☞ (Some) evidence of quadratic convergence, see Rem. 4.1.15.

◇

R. Hiptmair

rev 30401, November 9, 2010

There is a sophisticated theory about the convergence of Newton's method. For example one can find the following theorem in [14, Thm. 4.10], [12, Sect. 2.1]):

Theorem 4.4.11 (Local quadratic convergence of Newton's method)**. If***:*

(A) $D \subset \mathbb{R}^n$ *open and convex,*

(B) $F : D \mapsto \mathbb{R}^n$ *continuously differentiable,*

(C) $DF(\mathbf{x})$ *regular* $\forall \mathbf{x} \in D$*,*

(D) $\exists L \geq 0$: $\left\| DF(\mathbf{x})^{-1}(DF(\mathbf{x}+\mathbf{v}) - DF(\mathbf{x})) \right\|_2 \leq L \left\| \mathbf{v} \right\|_2 \quad \begin{array}{l} \forall \mathbf{v} \in \mathbb{R}^n, \mathbf{v}+\mathbf{x} \in D, \\ \forall \mathbf{x} \in D \end{array}$,

(E) $\exists \mathbf{x}^*$: $F(\mathbf{x}^*) = 0$ (*existence of solution in* D)

(F) initial guess $\mathbf{x}^{(0)} \in D$ *satisfies* $\rho := \left\| \mathbf{x}^* - \mathbf{x}^{(0)} \right\|_2 < \frac{2}{L} \quad \wedge \quad B_\rho(\mathbf{x}^*) \subset D$.

then *the Newton iteration* (4.4.1) *satisfies:*

(i) $\mathbf{x}^{(k)} \in B_\rho(\mathbf{x}^*) := \{\mathbf{y} \in \mathbb{R}^n, \left\| \mathbf{y} - \mathbf{x}^* \right\| < \rho\}$ *for all* $k \in \mathbb{N}$*,*

(ii) $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$*,*

(iii) $\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\|_2 \leq \frac{L}{2} \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\|_2^2$ (*local quadratic convergence*) .

R. Hiptmair

rev 30401, November 9, 2010

✎ notation: ball $B_\rho(\mathbf{z}) := \{\mathbf{x} \in \mathbb{R}^n: \ \|\mathbf{x}-\mathbf{z}\|_2 \leq \rho\}$

Terminology: (D) $\hat{=}$ affine invariant Lipschitz condition

Problem: Usually neither ω nor x^* are known !

► In general: a priori estimates as in Thm. 4.4.11 are of little practical relevance.

4.4.3 Termination of Newton iteration

A first viable idea:

Asymptotically due to quadratic convergence:

$$\left\|\mathbf{x}^{(k+1)}-\mathbf{x}^*\right\| \ll \left\|\mathbf{x}^{(k)}-\mathbf{x}^*\right\| \quad \Rightarrow \quad \left\|\mathbf{x}^{(k)}-\mathbf{x}^*\right\| \approx \left\|\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right\| \ . \tag{4.4.12}$$

➢ quit iterating as soon as $\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\| = \left\|DF(\mathbf{x}^{(k)})^{-1}F(\mathbf{x}^{(k)})\right\| < \tau \left\|\mathbf{x}^{(k)}\right\|$,
with τ = tolerance

$\rightarrow$ uneconomical: one needless update, because $\mathbf{x}^{(k)}$ already accurate enough **!**

Remark 4.4.13. New aspect for $n \gg 1$: computation of Newton correction may be expensive **!** △

Therefore we would like to use an a-posteriori termination criterion that dispenses with computing (and "inverting") another Jacobian $DF(\mathbf{x}^{(k)})$ just to tell us that $\mathbf{x}^{(k)}$ is already accurate enough.

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

$DF(\mathbf{x}^{(k-1)}) \approx DF(\mathbf{x}^{(k)})$: quit as soon as $\left\|DF(\mathbf{x}^{(k-1)})^{-1}F(\mathbf{x}^{(k)})\right\| < \tau \left\|\mathbf{x}^{(k)}\right\|$

affine invariant termination criterion

Justification: we expect $DF(\mathbf{x}^{(k-1)}) \approx DF(\mathbf{x}^{(k)})$, when Newton iteration has converged. Then appeal to (4.4.12).

Rationale: LU-decomposition of $DF(\mathbf{x}^{(k-1)})$ is already available ➤ less effort.

If we used the residual based termination criterion

$$\left\| F(\mathbf{x}^{(k)}) \right\| \leq \tau \ ,$$

then the resulting algorithm would not be affine invariant, because for $F(\mathbf{x}) = 0$ and $\mathbf{A}F(\mathbf{x}) = 0$, $\mathbf{A} \in \mathbb{R}^{n,n}$ regular, the Newton iteration might terminate with different iterates.

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

Reuse of LU-factorization (→ Rem. 2.2.12) of $DF(\mathbf{x}^{(k-1)})$ ➤ $\Delta\bar{\mathbf{x}}^{(k)}$ available with $O(n^2)$ operations

Summary: The Newton Method


converges *asymptotically* very fast: doubling of number of significant digits in each step


often a very small region of convergence, which requires an initial guess rather close to the solution.

4.4.4 Damped Newton method [10, pp. 200]

Potentially big problem: Newton method converges quadratically, but only *locally*, which may render it useless, if convergence is guaranteed only for initial guesses very close to exact solution, see also Ex. 4.3.18.

In this section we study a method to enlarge the region of convergence, at the expense of quadratic convergence, of course.

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

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

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

◇

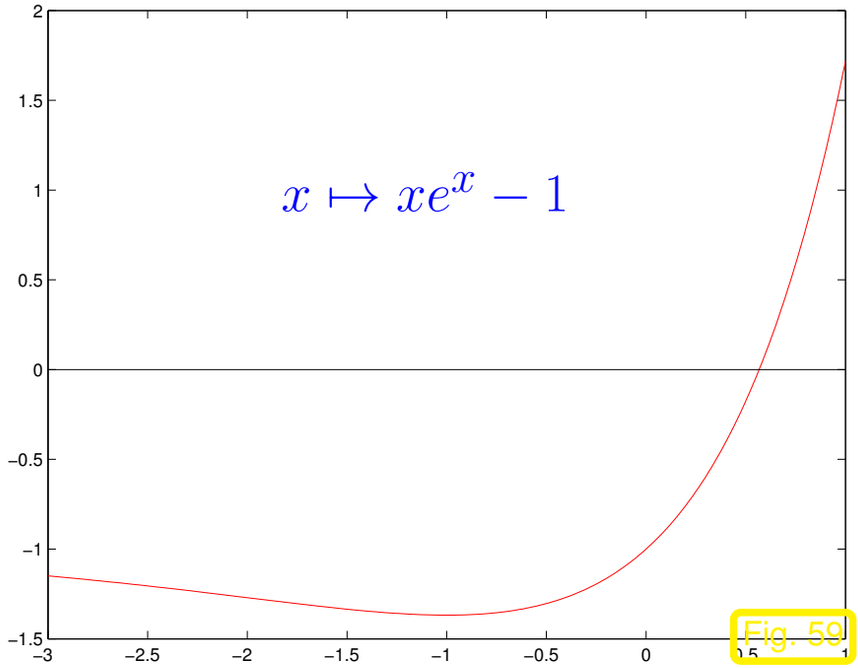


Fig. 59

Example 4.4.15 (Region of convergence of Newton method).

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

with zero $x^* = 0$.

◇

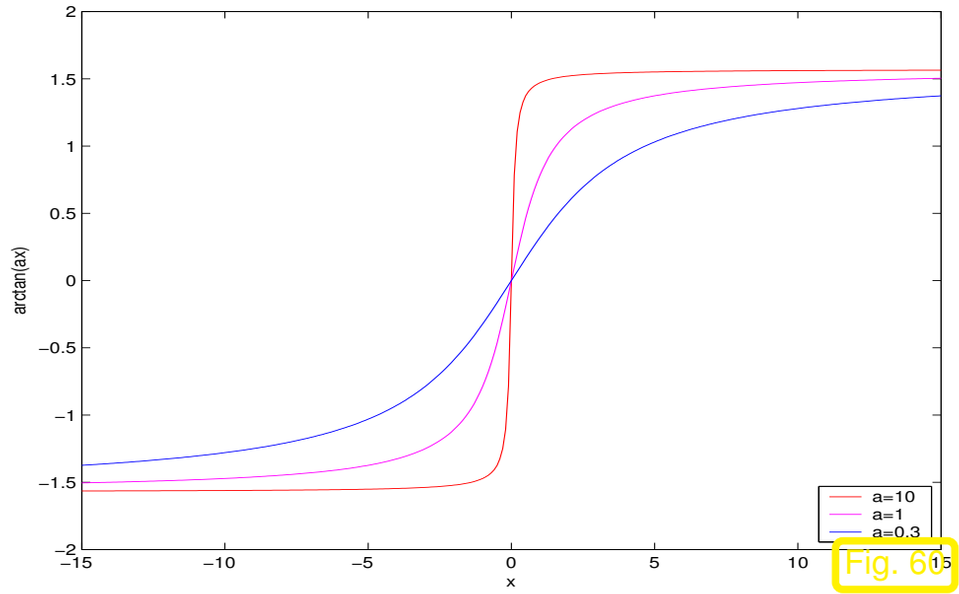
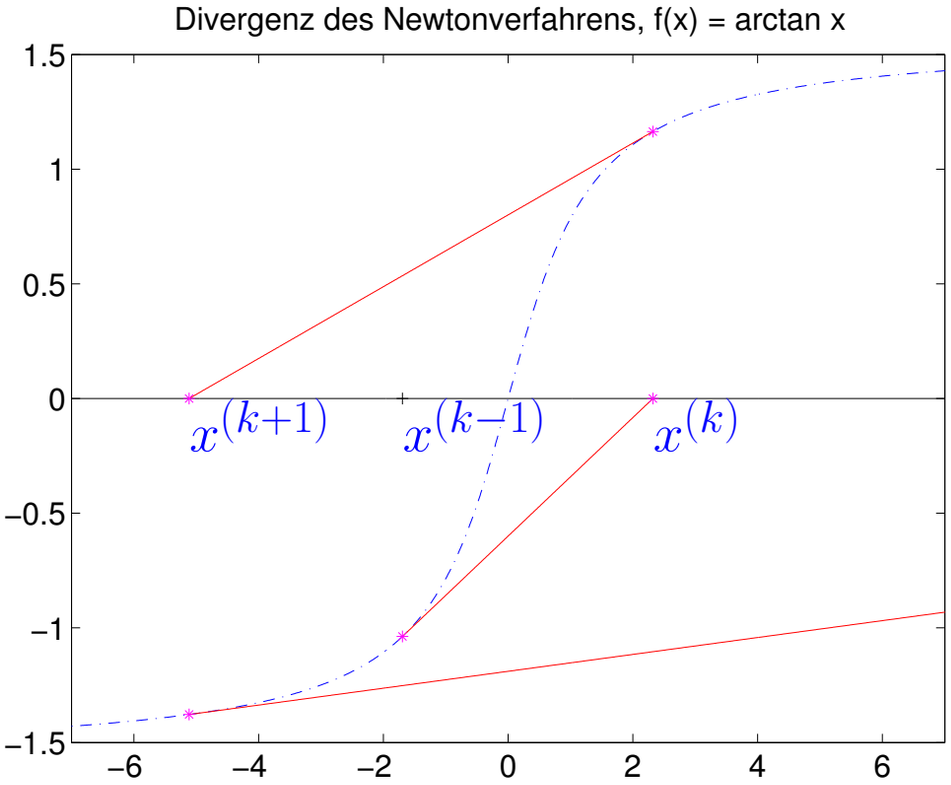
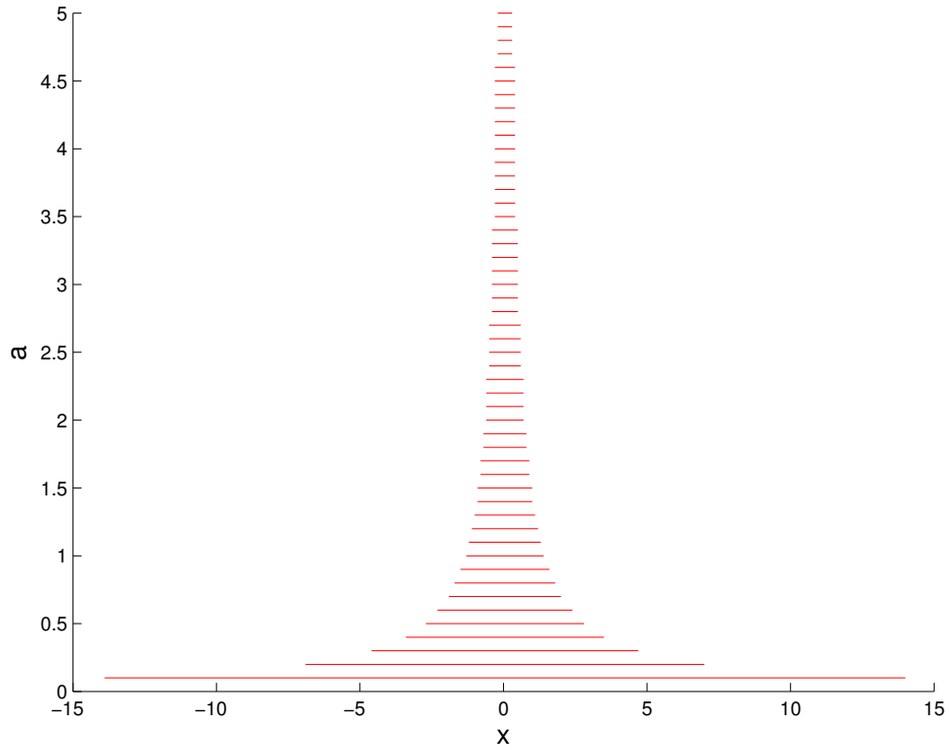


Fig. 60

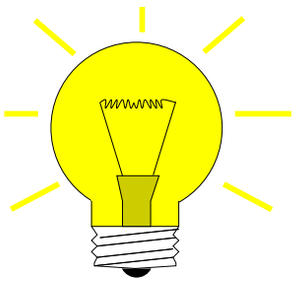




red zone = $\{x^{(0)} \in \mathbb{R}, x^{(k)} \to 0\}$

R. Hiptmair

rev 30401, November 9, 2010



▶ 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 [12, Ch. 3]

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

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

Heuristics: convergence $\Leftrightarrow$ size of Newton correction decreases

Code 4.4.19: Damped Newton method (non-optimal implementation !)

```
function [x,cvg] = dampnewton(x,F,DF,tol)
[L,U] = lu(DF(x)); s = U\(L\F(x));
xn = x-s; lambda = 1; cvg = 0;
f = F(xn); st = U\(L\f);
while (norm(st) > tol*norm(xn))
  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. 2.2.12

a-posteriori termination criterion (based on simplified Newton correction, *cf.* Sect. 4.4.3)

Natural monotonicity test (4.4.17)

Reduce damping factor λ

R. Hiptmair

rev 30401, November 9, 2010

Policy: Reduce damping factor by a factor $q \in]0,1[$ (usually $q = \frac{1}{2}$) until the affine invariant natural monotonicity test (4.4.17) passed.

Example 4.4.20 (Damped Newton method). ($\rightarrow$ Ex. 4.4.15)

$F(x) = \arctan(x)$,

- $x^{(0)} = 20$
- $q = \frac{1}{2}$
- `LMIN = 0.001`

Observation: asymptotic quadratic convergence

k	$\lambda^{(k)}$	$x^{(k)}$	$F(x^{(k)})$
1	0.03125	0.94199967624205	0.7554074974604
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 4.4.21 (Failure of damped Newton method).

- As in Ex. 4.4.14:
  $F(x) = xe^x - 1$,

- Initial guess for damped Newton method $x^{(0)} = -1.5$

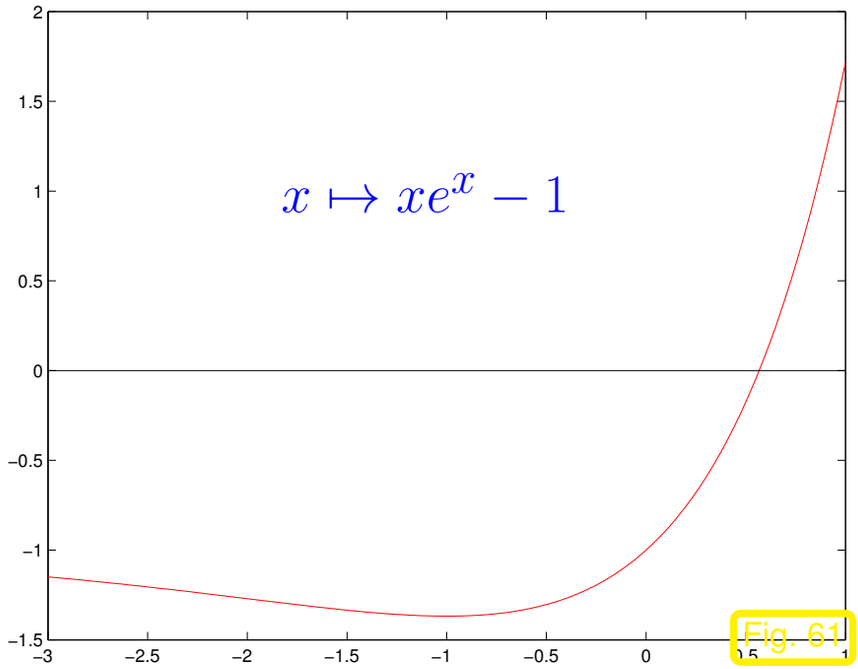


Fig. 61

Observation:

Newton correction pointing in "wrong direction"

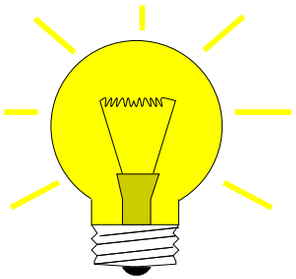
➤ no convergence despite damping

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

What to do when $DF(\mathbf{x})$ is not available and numerical differentiation (see remark 4.4.8) is too expensive**?**

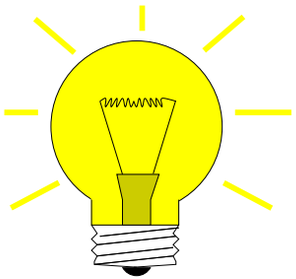


Idea: in one dimension ($n = 1$) apply the secant method (4.3.11) of section 4.3.2.3

$$F'(x^{(k)}) \approx \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \quad \text{"difference quotient"} \qquad (4.4.22)$$

already computed **!** $\rightarrow$ cheap

Generalisation for $n > 1$ **?**

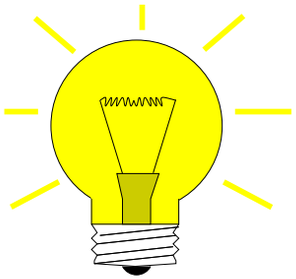


Idea: rewrite (4.4.22) as a secant condition for the approximation $\mathbf{J}_k \approx DF(\mathbf{x}^{(k)})$, $\mathbf{x}^{(k)} \hat{=}$ iterate:

$$\mathbf{J}_k(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}) = F(\mathbf{x}^{(k)}) - F(\mathbf{x}^{(k-1)}) \; . \qquad (4.4.23)$$

BUT: many matrices $\mathbf{J}_k$ fulfill (4.4.23)

Hence: we need more conditions for $\mathbf{J}_k \in \mathbb{R}^{n,n}$



Idea: get $\mathbf{J}_k$ by a modification of $\mathbf{J}_{k-1}$

Broyden conditions: $\mathbf{J}_k\mathbf{z} = \mathbf{J}_{k-1}\mathbf{z} \quad \forall \mathbf{z}: \mathbf{z} \perp (\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})$. (4.4.24)

i.e.: $$\mathbf{J}_k := \mathbf{J}_{k-1} + \frac{F(\mathbf{x}^{(k)})(\mathbf{x}^{(k)}-\mathbf{x}^{(k-1)})^T}{\left\|\mathbf{x}^{(k)}-\mathbf{x}^{(k-1)}\right\|_2^2} \tag{4.4.25}$$

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

$$\begin{aligned} &\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}, \; \Delta\mathbf{x}^{(k)} := -\mathbf{J}_k^{-1}F(\mathbf{x}^{(k)}) , \\ &\mathbf{J}_{k+1} := \mathbf{J}_k + \frac{F(\mathbf{x}^{(k+1)})(\Delta\mathbf{x}^{(k)})^T}{\left\|\Delta\mathbf{x}^{(k)}\right\|_2^2} . \end{aligned} \tag{4.4.26}$$

Start: initialize $\mathbf{J}_0$ e.g. with the exact Jacobi matrix $DF(\mathbf{x}^{(0)})$.

R. Hiptmair

rev 30401, November 9, 2010

Remark 4.4.27 (Minimal property of Broydens rank 1 modification).

Let $\mathbf{J} \in \mathbb{R}^{n,n}$ fulfill (4.4.23) and $\mathbf{J}_k$, $\mathbf{x}^{(k)}$ from (4.4.26) then $(\mathbf{I} - \mathbf{J}_k^{-1}\mathbf{J})(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = -\mathbf{J}_k^{-1}F(\mathbf{x}^{(k+1)})$

and hence

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

$$\leq \left\|\mathbf{I} - \mathbf{J}_k^{-1}\mathbf{J}\right\|_2 \ .$$

In conlcusion,
(4.4.25) gives the $\|\cdot\|_2$-minimal relative correction of $\mathbf{J}_{k-1}$, such that the secant condition (4.4.23) holds.

△

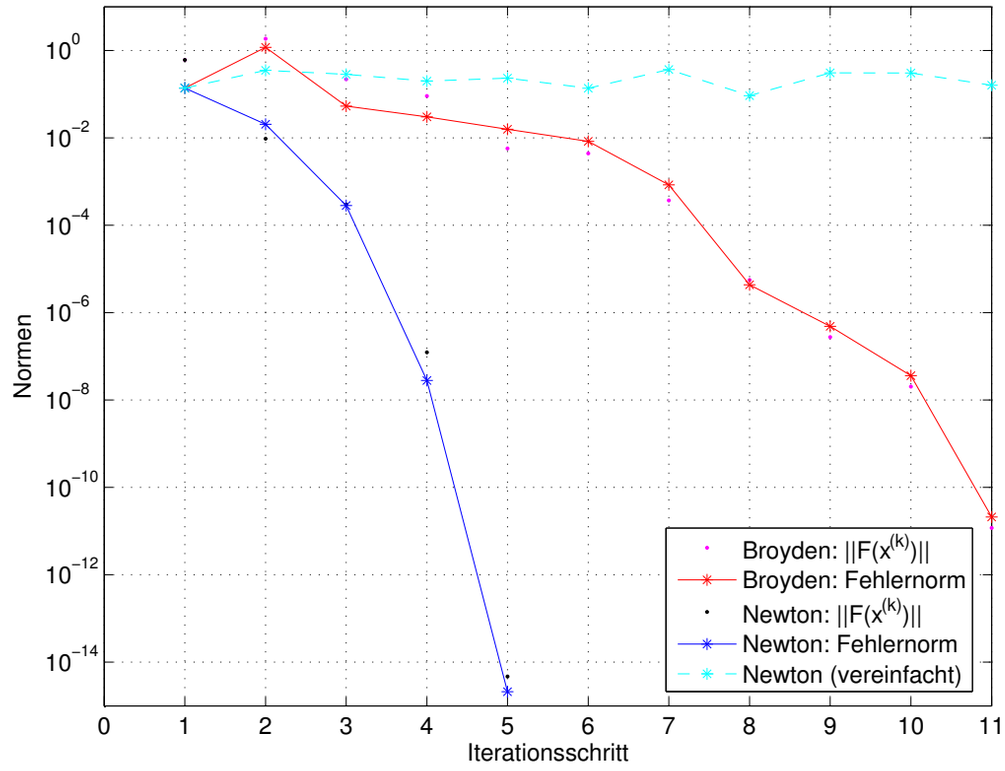
Example 4.4.28 (Broydens Quasi-Newton Method: Convergence).

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

The numerical example shows that the method is:

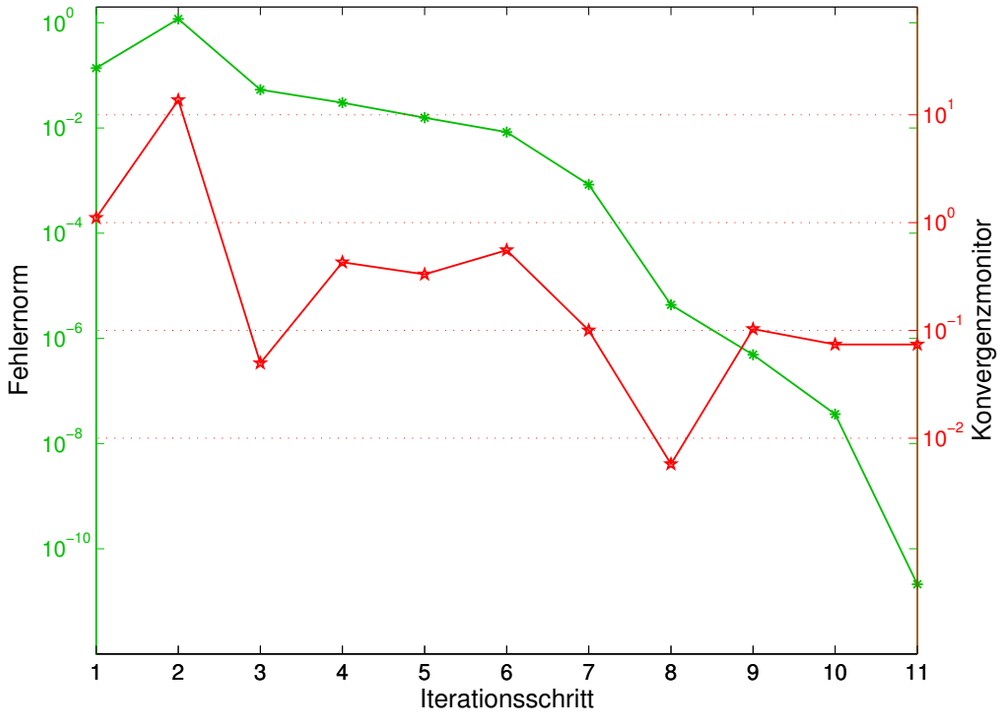
slower than Newton method (4.4.1), but ▷

better than simplified Newton method (see remark. 4.4.6)



Observation: for all iterative methods for non-linear systems of equations convergence can fail (stall/diverge).

➢ algorithms should warn user of impending failure.



convergence monitor

=

quantity that alerts to difficulties in the convergence of an iteration

Here:

$$\mu := \frac{\left\| \mathbf{J}_{k-1}^{-1} F(\mathbf{x}^{(k)}) \right\|}{\left\| \Delta \mathbf{x}^{(k-1)} \right\|}$$

Heuristics: no convergence whenever $\mu > 1$

◇

Remark 4.4.29 (Damped Broyden method).

Option to improve robustness (increase region of local convergence):

damped Broyden method (*cf.* same idea for Newton's method, section 4.4.4)

R. Hiptmair

rev 30401, November 9, 2010

Implementation of (4.4.26):

Observe: (4.4.26) is a rank-1-update $\rightarrow$ Sect. 2.9.0.1

Idea: use Sherman-Morrison-Woodbury update-formula, Lemma 2.9.7

$$\mathbf{J}_{k+1}^{-1} = \left(\mathbf{I} - \frac{\mathbf{J}_k^{-1} F(\mathbf{x}^{(k+1)}) (\Delta \mathbf{x}^{(k)})^T}{\left\| \Delta \mathbf{x}^{(k)} \right\|_2^2 + \Delta \mathbf{x}^{(k)} \cdot \mathbf{J}_k^{-1} F(\mathbf{x}^{(k+1)})} \right) \mathbf{J}_k^{-1} = \left(\mathbf{I} + \frac{\Delta \mathbf{x}^{(k+1)} (\Delta \mathbf{x}^{(k)})^T}{\left\| \Delta \mathbf{x}^{(k)} \right\|_2^2} \right) \mathbf{J}_k^{-1} \tag{4.4.30}$$

that is well defined, if

$$\left\| \mathbf{J}_k^{-1} F(\mathbf{x}^{(k+1)}) \right\|_2 < \left\| \Delta \mathbf{x}^{(k)} \right\|_2$$

"simplified Quasi-Newton correction"

Iterated application of (4.4.30) pays off, if iteration terminates after a few steps.

MATLAB-CODE: Broyden method (4.4.26)

```
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 (4.4.30))

solve two SLEs

Termination, see 4.4.2

construct $\mathbf{w} := \mathbf{J}_k^{-1}F(\mathbf{x}^{(k)})$ ($\rightarrow$ recursion (4.4.30))

convergence monitor

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

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

R. Hiptmair

rev 30401, November 9, 2010

Computational cost : N steps

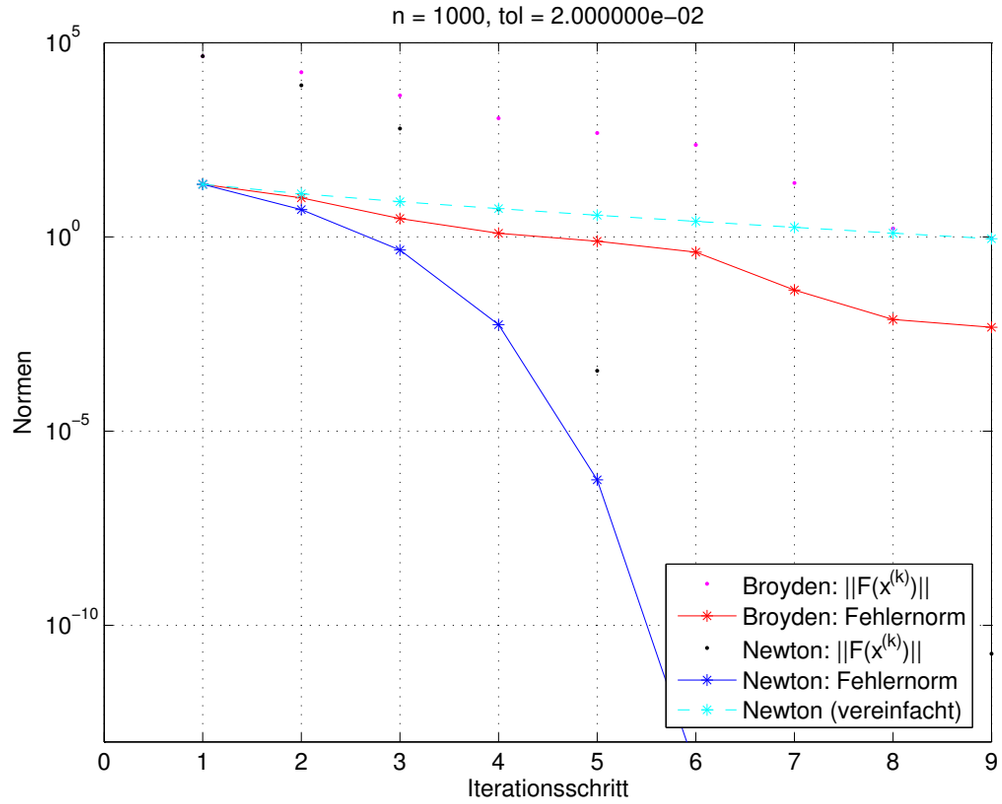
- $O(N^2 \cdot n)$ operations with vectors, (Level I)
- 1 LU-decomposition of $\mathtt{J}$, $N\times$ solutions of SLEs, see section 2.2
- N evalutations of F !

Memory cost : N steps

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

Example 4.4.31 (Broyden method for a large non-linear system).

$$F(\mathbf{x}) = \begin{cases} \mathbb{R}^n \mapsto \mathbb{R}^n \\ \mathbf{x} \mapsto \operatorname{diag}(\mathbf{x})\mathbf{A}\mathbf{x} - \mathbf{b} \end{cases},$$
$$\mathbf{b} = (1, 2, \ldots, n) \in \mathbb{R}^n ,$$
$$\mathbf{A} = \mathbf{I} + \mathbf{a}\mathbf{a}^T \in \mathbb{R}^{n,n} ,$$
$$\mathbf{a} = \frac{1}{\sqrt{\mathbf{1}\cdot\mathbf{b} - \mathbf{1}}}(\mathbf{b} - \mathbf{1}) .$$

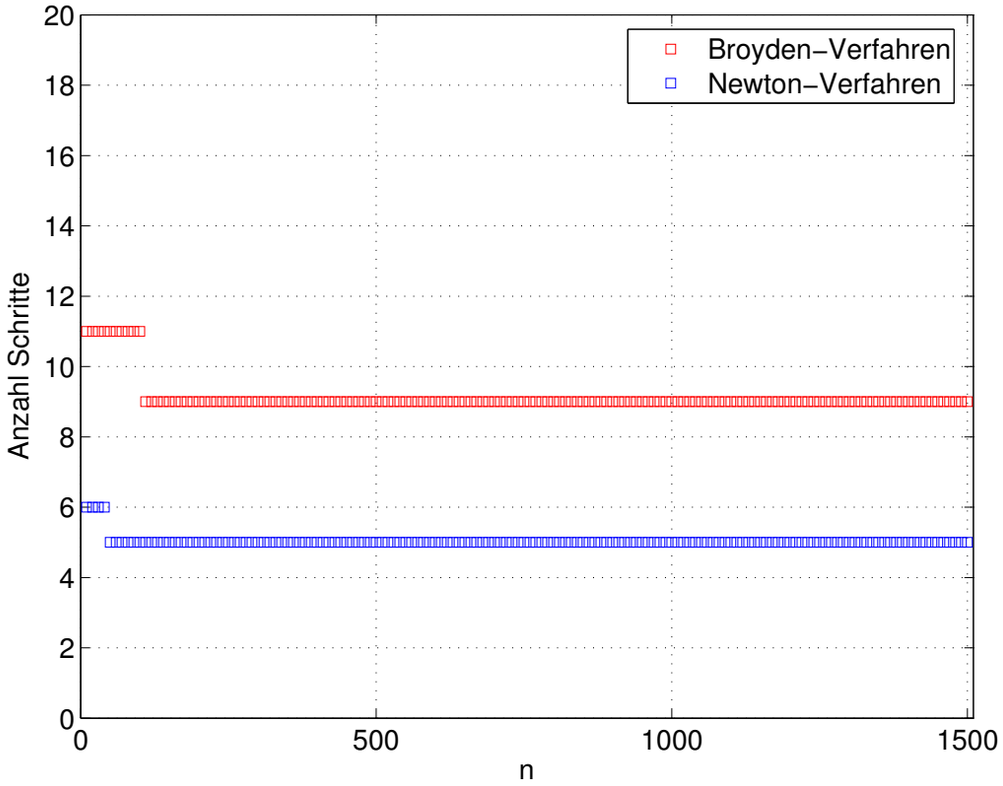


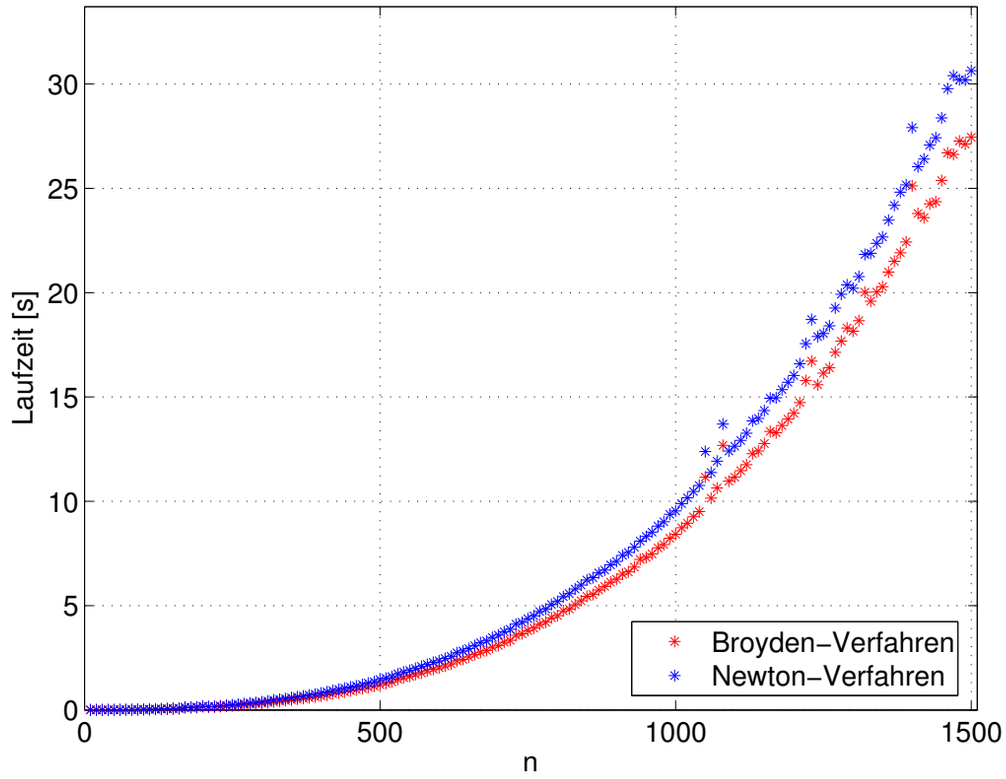
The interpretation of the results resemble the example 4.4.28 ▷

```
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)';`)





☞ In conclusion,
the Broyden method is worthwile for dimensions $n \gg 1$ and low accuracy requirements.

5 Krylov Methods for Linear Systems of Equations

= A class of iterative methods ($\rightarrow$ section 4.1) for approximate solution of large linear systems of equations $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{K}^{n,n}$.

R. Hiptmair

rev 30401, November 15, 2010

BUT, we have reliable *direct* methods (Gauss eliminination $\rightarrow$ Sect. 2.1, LU-factorization $\rightarrow$ Alg. 2.2.11, QR-factorization $\rightarrow$ Alg. 2.8.22) that provide an (apart from roundoff errors) exact solution with a *finite* number of elementary operations!

Alas, direct elimination may not be feasible, or may be grossly inefficient, because

- it may be too expensive (e.g. for $\mathbf{A}$ too large, sparse), $\rightarrow$ (2.2.8),
- inevitable fill-in may exhaust main memory,
- the system matrix may be available only as procedure `y=evalA(x)` $\leftrightarrow \mathbf{y} = \mathbf{Ax}$

5.1 Descent Methods

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

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

➡ $\mathbf{A}$-inner product $(\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x}^T \mathbf{A}\mathbf{y}$ ⇒ "$\mathbf{A}$-geometry"

Definition 5.1.1 (Energy norm). → *[29, Def. 9.1]*

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

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

Remark 5.1.2 (Krylov methods for complex s.p.d. system matrices).

In this chapter, for the sake of simplicity, we restrict ourselves to $\mathbb{K} = \mathbb{R}$.

However, the (conjugate) gradient methods introduced below also work for LSE $\mathbf{Ax} = \mathbf{b}$ with $\mathbf{A} \in \mathbb{C}^{n,n}$, $\mathbf{A} = \mathbf{A}^H$ s.p.d. when ^{T} is replaced with ^{H} (Hermitian transposed). Then, all theoretical statements remain valid unaltered for $\mathbb{K} = \mathbb{C}$.

△

R. Hiptmair

rev 30401, November 15, 2010

5.1.1 Quadratic minimization context

Lemma 5.1.3 (S.p.d. LSE and quadratic minimization problem)**.**

A LSE with $\mathbf{A} \in \mathbb{R}^{n,n}$ *s.p.d. and* $\mathbf{b} \in \mathbb{R}^n$ *is equivalent to a minimization problem:*

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

A quadratic functional

Proof. If $\mathbf{x}^* := \mathbf{A}^{-1}\mathbf{b}$ a straightforward computation using $\mathbf{A} = \mathbf{A}^T$ shows

$$\begin{aligned} J(\mathbf{x}) - J(\mathbf{x}^*) &= \tfrac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{x} - \tfrac{1}{2}(\mathbf{x}^*)^T\mathbf{Ax}^* + \mathbf{b}^T\mathbf{x}^* \\ &\overset{\mathbf{b}=\mathbf{Ax}^*}{=} \tfrac{1}{2}\mathbf{x}^T\mathbf{Ax} - (\mathbf{x}^*)^T\mathbf{Ax} + \tfrac{1}{2}(\mathbf{x}^*)^T\mathbf{Ax}^* \\ &= \tfrac{1}{2}\left\|\mathbf{x} - \mathbf{x}^*\right\|_A^2 \;. \end{aligned} \tag{5.1.5}$$

Then the assertion follows from the properties of the energy norm.

Example 5.1.6 (Quadratic functional in 2D).

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

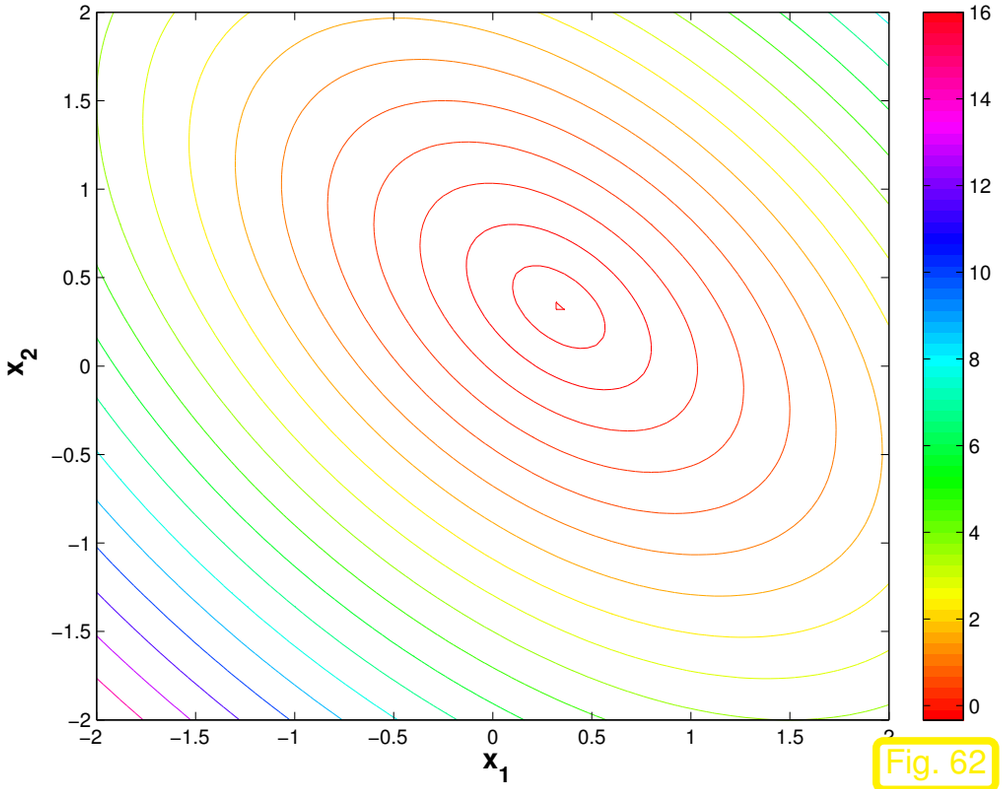


Fig. 62

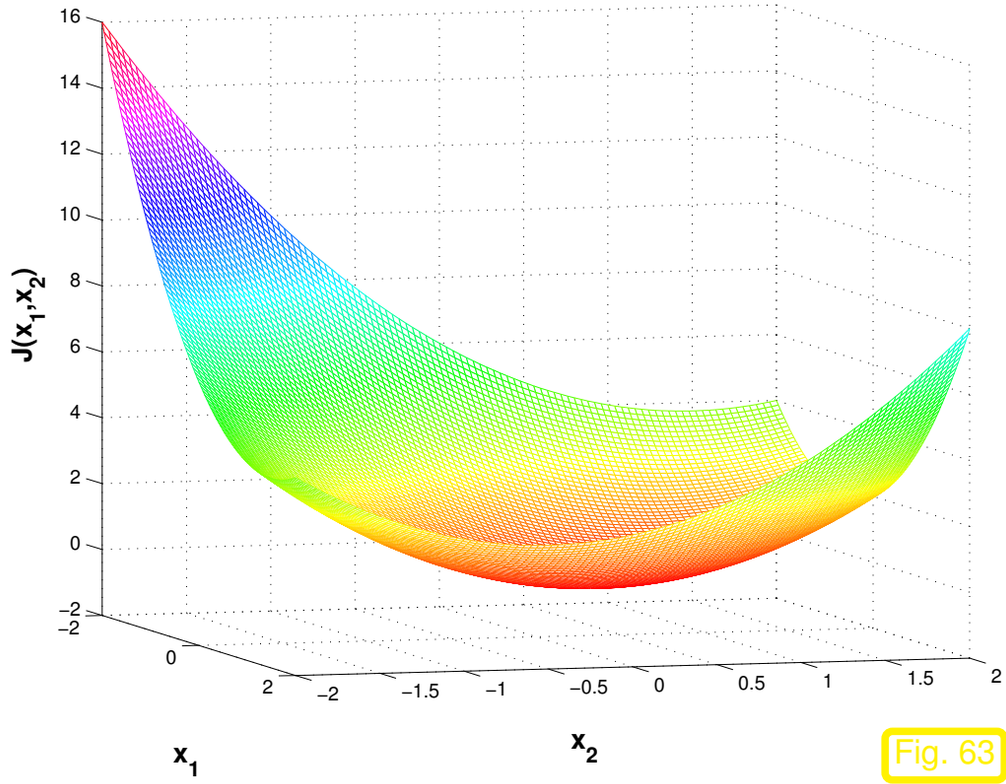
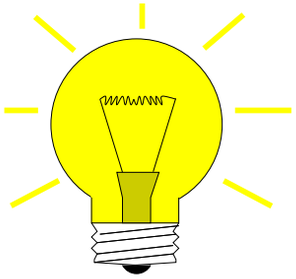


Fig. 63

Level lines of quadratic functionals with s.p.d. $\mathbf{A}$ are (hyper)ellipses

R. Hiptmair

rev 30401, November 15, 2010



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 $\mathbf{x}^* \in D$: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in D} F(\mathbf{x})$

Note that a minimizer need not exist, if F is not bounded from below (e.g., $F(x) = x^3$, $x \in \mathbb{R}$, or $F(x) = \log x$, $x > 0$), or if D is open (e.g., $F(x) = \sqrt{x}$, $x > 0$).

The existence of a minimizer is guaranteed if F is bounded from below and D is closed ($\rightarrow$ Analysis).

The most natural iteration:

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

Initial guess $\mathbf{x}^{(0)} \in D$, $k = 0$

repeat

$\mathbf{d}_k := -\mathbf{grad}\, F(\mathbf{x}^{(k)})$

$t^* := \operatorname{argmin}_{t\in\mathbb{R}} F(\mathbf{x}^{(k)} + t\mathbf{d}_k)$ (line search)

$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + t^*\mathbf{d}_k$

$k := k + 1$

until $\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\| \leq \tau \left\|\mathbf{x}^{(k)}\right\|$

- $\mathbf{d}_k \mathrel{\hat{=}}$ *direction of steepest descent*
- linear search $\mathrel{\hat{=}}$ 1D minimization: use Newton's method ($\rightarrow$ Sect. 4.3.2.1) on derivative
- a posteriori termination criterion, see Sect. 4.1.2 for a discussion.
($\tau \mathrel{\hat{=}}$ prescribed tolerance)

R. Hiptmair

rev 30401, November 15, 2010

The gradient ($\rightarrow$ [52, 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

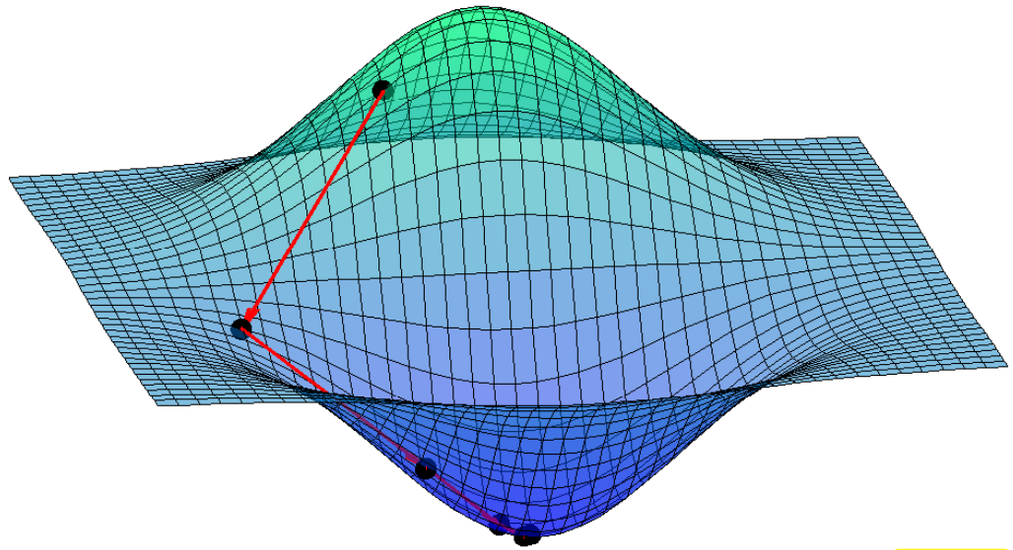


Fig. 64

Of course this very algorithm can encouter plenty of difficulties:

- iteration may get stuck in a *local minimum*,
- iteration may diverge or lead out of D,
- line search may not be feasible.

R. Hiptmair

rev 30401, November 15, 2010

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

However, for the quadratic minimization problem (5.1.4) Alg. 5.1.7 will converge:

("Geometric intuition", see Fig. 62: quadratic functional J with s.p.d. $\mathbf{A}$ has unique global minimum, $\mathbf{grad}\, J \neq 0$ away from minimum, pointing towards it.)

Adaptation: steepest descent algorithm Alg. 5.1.7 for quadratic minimization problem (5.1.4)

$$F(\mathbf{x}) := J(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} \quad \Rightarrow \quad \mathbf{grad}\, J(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}\ . \tag{5.1.9}$$

This follows from $\mathbf{A} = \mathbf{A}^T$, the componentwise expression

$$J(\mathbf{x}) = \tfrac{1}{2}\sum_{i,j=1}^{n} a_{ij}x_i x_j - \sum_{i=1}^{n} b_i x_i$$

and the definition (5.1.8) of the gradient.

➢ For the descent direction in Alg. 5.1.7 applied to the minimization of J from (5.1.4) holds

$$\mathbf{d}_k = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} =: \mathbf{r}_k \quad \text{the residual (→ Def. 2.5.16) for } \mathbf{x}^{(k-1)}.$$

R. Hiptmair

rev 30401, November 15, 2010

Alg. 5.1.7 for $F = J$ from (5.1.4): function to be minimized in line search step:

$$\varphi(t) := J(\mathbf{x}^{(k)} + t\mathbf{d}_k) = J(\mathbf{x}^{(k)}) + t\mathbf{d}_k^T(\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}) + \tfrac{1}{2}t^2\mathbf{d}_k^T\mathbf{A}\mathbf{d}_k \quad \rightarrow \text{ a parabola ! .}$$

$$\frac{d\varphi}{dt}(t^*) = 0 \quad \Leftrightarrow \quad \boxed{t^* = \frac{\mathbf{d}_k^T\mathbf{d}_k}{\mathbf{d}_k^T\mathbf{A}\mathbf{d}_k}} \quad \text{(unique minimizer) .} \tag{5.1.10}$$

Note: $\mathbf{d}_k = 0 \quad \Leftrightarrow \quad \mathbf{A}\mathbf{x}^{(k)} = \mathbf{b}$ (solution found !)

Note: $\mathbf{A}$ s.p.d. ($\rightarrow$ Def. 2.7.9) $\quad \Rightarrow \quad \mathbf{d}_k^T\mathbf{A}\mathbf{d}_k > 0$, if $\mathbf{d}_k \neq 0$

$\varphi(t)$ is a parabola that is bounded from below (upward opening)

R. Hiptmair

rev 30401, November 15, 2010

Based on (5.1.9) and (5.1.10) we obtain the following steepest descent method for the minimization problem (5.1.4):

Steepest descent iteration = gradient method for LSE $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{n,n}$ s.p.d., $\mathbf{b} \in \mathbb{R}^n$:

Algorithm 5.1.11 (Gradient method).

Initial guess $\mathbf{x}^{(0)} \in \mathbb{R}^n$, $k = 0$

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

repeat

$$t^* := \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_k^T \mathbf{A}\mathbf{r}_k}$$

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + t^* \mathbf{r}_k$$

$$\mathbf{r}_{k+1} := \mathbf{r}_k - t^* \mathbf{A}\mathbf{r}_k$$

$$k := k + 1$$

until $\left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \right\| \leq \tau \left\| \mathbf{x}^{(k)} \right\|$

Code 5.1.13: gradient method for $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}$ s.p.d.

```
1  function x =
     gradit(A,b,x,tol,maxit)
2  % Python:
     ../PYTHON/GradientMethod.py, gradit()
3  r = b-A*x;
4  for k=1:maxit
5    p = A*r;
6    ts = (r'*r)/(r'*p);
7    x = x + ts*r;
8    if (abs(ts)*norm(r) <
       tol*norm(x))
9      return; end
10   r = r - ts*p;
11 end
```

Recursion for residuals:

$$\mathbf{r}_{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{A}(\mathbf{x}^{(k)} + t^* \mathbf{r}_k) = \mathbf{r}_k - t^* \mathbf{A}\mathbf{r}_k \,. \tag{5.1.14}$$

One step of gradient method involves

- *A single* matrix×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×vector) + $O(n)$

➢ If $\mathbf{A} \in \mathbb{R}^{n,n}$ is a sparse matrix ($\rightarrow$ Sect. 2.6) with "$O(n)$ nonzero entries", and the data structures allow to perform the matrix×vector product with a computational effort $O(n)$, then a single step of the gradient method costs $O(n)$ elementary operations.

5.1.4 Convergence of gradient method

Example 5.1.15 (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 $\in \mathbb{K}^{n,n}$ $\sigma(\mathbf{M}) := \{\lambda \in \mathbb{C}: \lambda \text{ is eigenvalue of } \mathbf{M}\}$

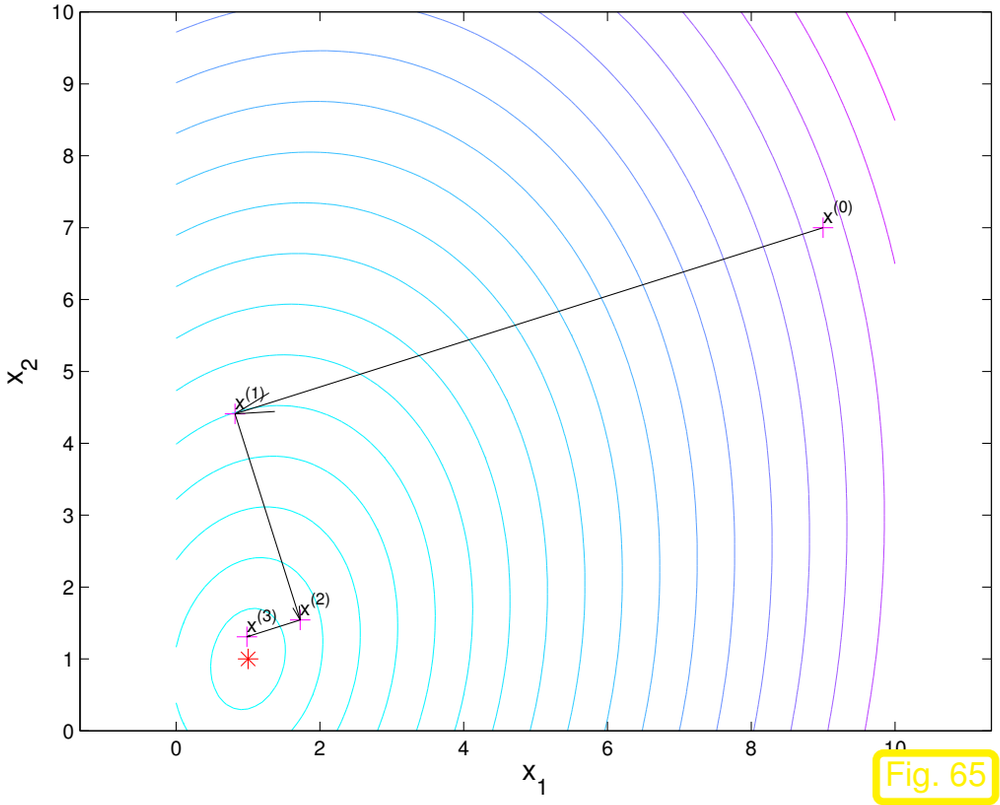


Fig. 65

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

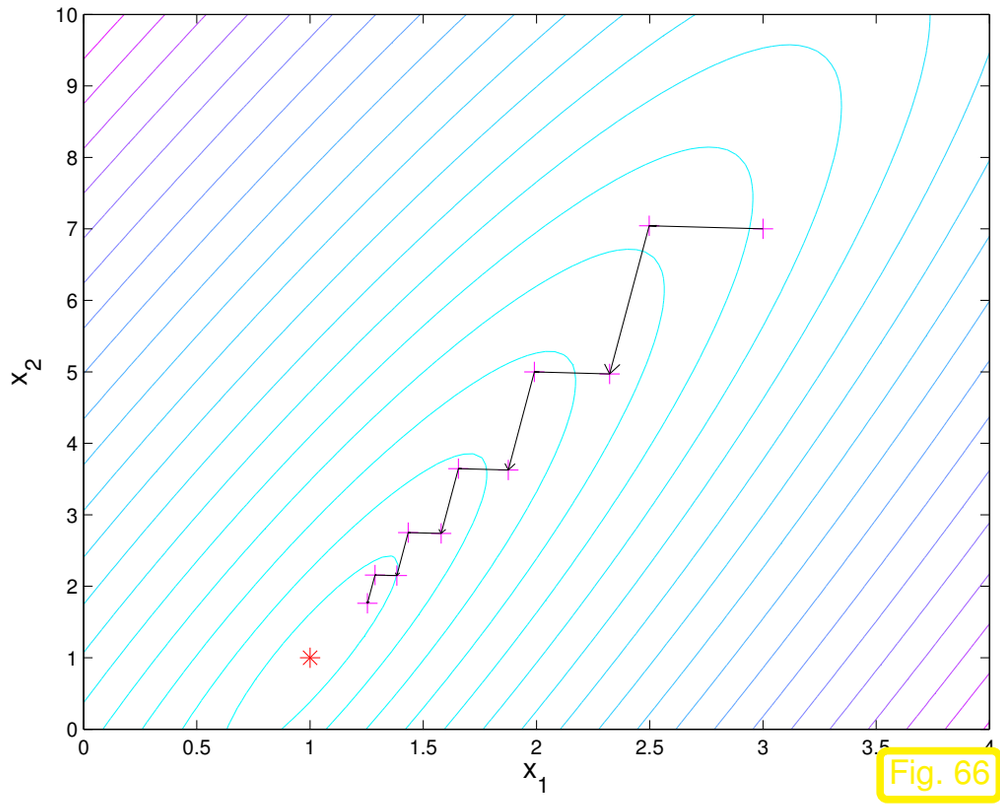


Fig. 66

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

R. Hiptmair

rev 30401, November 15, 2010

Recall theorem on principal axis transformation: every real *symmetric* matrix can be diagonalized by *orthogonal* similarity transformations, see Cor. 6.1.9, [39, Thm. 7.8], [23, Satz 9.15],

$\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n} \quad \Rightarrow \quad \exists \mathbf{Q} \in \mathbb{R}^{n,n}$ orthogonal: $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T \quad , \quad \mathbf{D} = \operatorname{diag}(d_1, \ldots, d_n) \in \mathbb{R}^{n,n}$ di

(5.1.16)

$$J(\mathbf{Q}\widehat{\mathbf{y}}) = \tfrac{1}{2}\widehat{\mathbf{y}}^T\mathbf{D}\widehat{\mathbf{y}} - \underbrace{(\mathbf{Q}^T\mathbf{b})^T}_{=:\widehat{\mathbf{b}}^T}\widehat{\mathbf{y}} = \tfrac{1}{2}\sum_{i=1}^{n} d_i\widehat{y}_i^2 - \widehat{b}_i\widehat{y}_i \; .$$

Hence, a rigid transformation (rotation, reflection) maps the level surfaces of J from (5.1.4) to ellipses with principal axes d_i. As $\mathbf{A}$ s.p.d. $d_i > 0$ is guaranteed.

Observations:

- Larger spread of spectrum leads to more elongated ellipses as level lines ➢ slower convergence of gradient method, see Fig. 66.
- Orthogonality of successive residuals $\mathbf{r}_k$, $\mathbf{r}_{k+1}$.

Clear from definition of Alg. 5.1.11:

$$\mathbf{r}_k^T\mathbf{r}_{k+1} = \mathbf{r}_k^T\mathbf{r}_k - \mathbf{r}_k^T\frac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{r}_k^T\mathbf{A}\mathbf{r}_k}\mathbf{A}\mathbf{r}_k = 0 \; . \tag{5.1.17}$$

Example 5.1.18 (Convergence of gradient method).

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

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

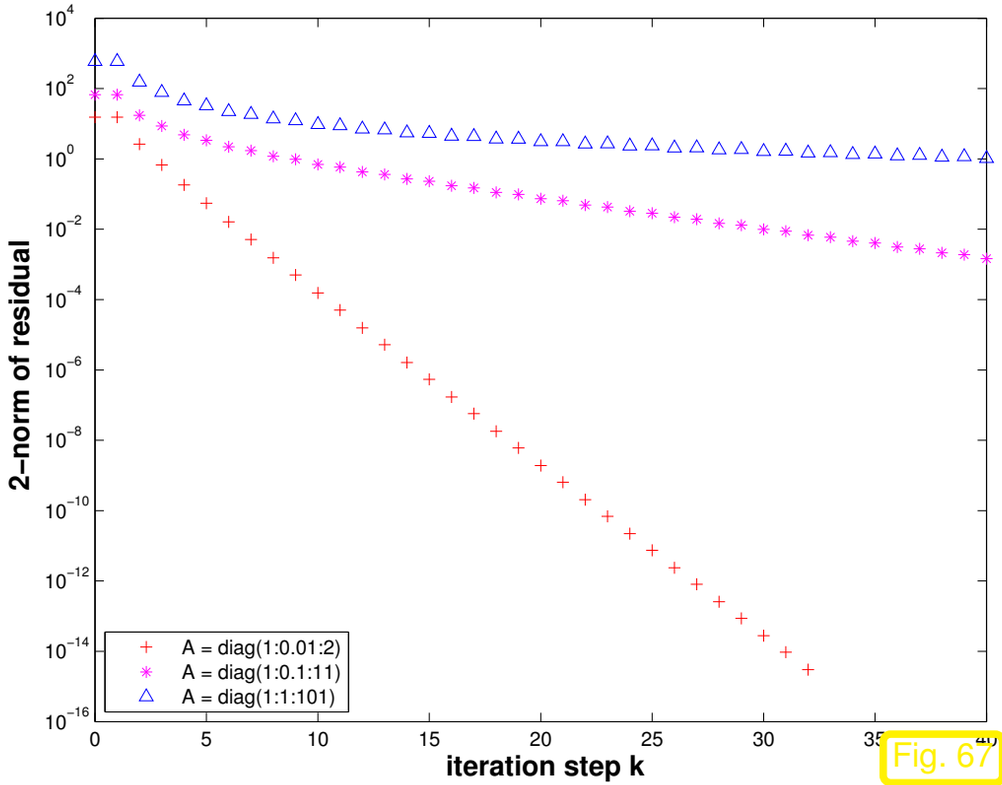


Fig. 67

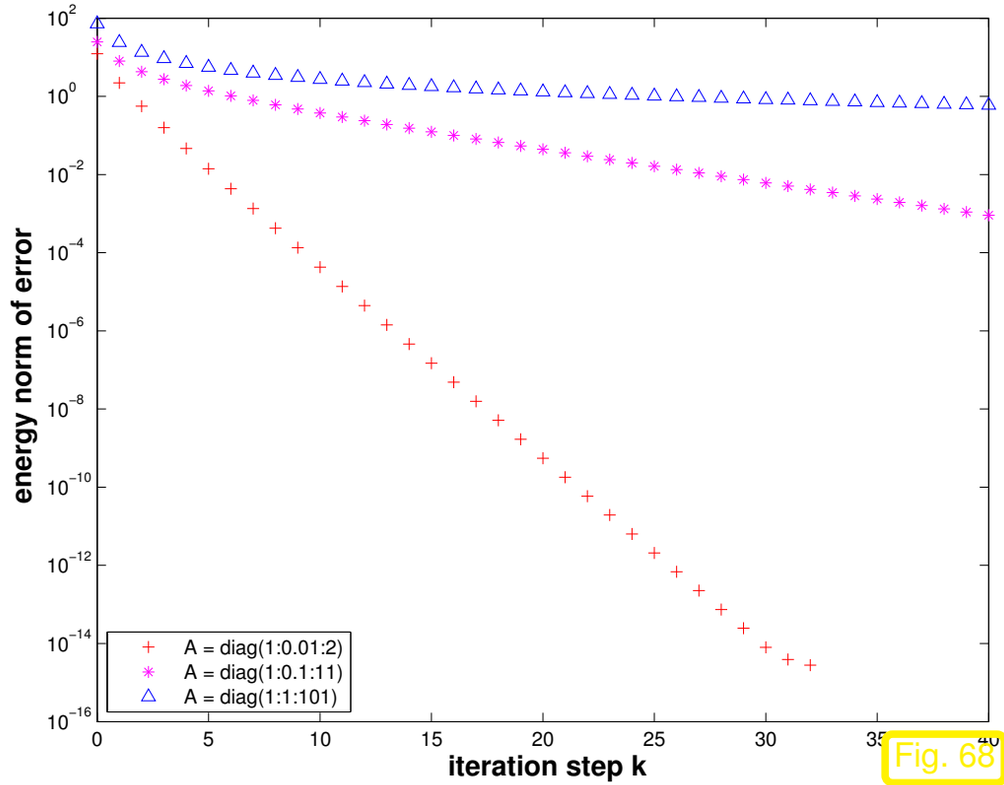


Fig. 68

R. Hiptmair

rev 30401, November 15, 2010

Note: To study convergence it is *sufficient to consider diagonal matrices*, because

1. (5.1.16): for every $\mathbf{A} \in \mathbb{R}^{n,n}$ with $\mathbf{A}^T = \mathbf{A}$ there is an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n.n}$ such that $\mathbf{A} = \mathbf{Q}^T\mathbf{D}\mathbf{Q}$ with a diagonal matrix $\mathbf{D}$ (principal axis transformation), $\rightarrow$ Cor. 6.1.9, [39, Thm. 7.8], [23, Satz 9.15],
2. when applying the gradient method Alg. 5.1.11 to both $\mathbf{A}\mathbf{x} = \mathbf{b}$ and $\mathbf{D}\widetilde{\mathbf{x}} = \widetilde{\mathbf{b}} := \mathbf{Q}\mathbf{b}$, then the iterates $\mathbf{x}^{(k)}$ and $\widetilde{\mathbf{x}}^{(k)}$ are related by $\mathbf{Q}\mathbf{x}^{(k)} = \widetilde{\mathbf{x}}^{(k)}$.

With $\widetilde{\mathbf{r}}_k := \mathbf{Q}\mathbf{r}_k$, $\widetilde{\mathbf{x}}^{(k)} := \mathbf{Q}\mathbf{x}^{(k)}$, using $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$:

Initial guess $\mathbf{x}^{(0)} \in \mathbb{R}^n$, $k = 0$

$\mathbf{r}_0 := \mathbf{b} - \mathbf{Q}^T\mathbf{D}\mathbf{Q}\mathbf{x}^{(0)}$

repeat

$$t^* := \frac{\mathbf{r}_k^T\mathbf{Q}^T\mathbf{Q}\mathbf{r}_k}{\mathbf{r}_k^T\mathbf{Q}^T\mathbf{D}\mathbf{Q}\mathbf{r}_k}$$

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + t^*\mathbf{r}_k$$

$$\mathbf{r}_{k+1} := \mathbf{r}_k - t^*\mathbf{Q}^T\mathbf{D}\mathbf{Q}\mathbf{r}_k$$

$$k := k + 1$$

until $\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\| \leq \tau \left\|\mathbf{x}^{(k)}\right\|$

Initial guess $\widetilde{\mathbf{x}}^{(0)} \in \mathbb{R}^n$, $k = 0$

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

repeat

$$t^* := \frac{\widetilde{\mathbf{r}}_k^T\widetilde{\mathbf{r}}_k}{\widetilde{\mathbf{r}}_k^T\mathbf{D}\widetilde{\mathbf{r}}_k}$$

$$\widetilde{\mathbf{x}}^{(k+1)} := \widetilde{\mathbf{x}}^{(k)} + t^*\widetilde{\mathbf{r}}_k$$

$$\widetilde{\mathbf{r}}_{k+1} := \widetilde{\mathbf{r}}_k - t^*\mathbf{D}\widetilde{\mathbf{r}}_k$$

$$k := k + 1$$

until $\left\|\widetilde{\mathbf{x}}^{(k)} - \widetilde{\mathbf{x}}^{(k-1)}\right\| \leq \tau \left\|\widetilde{\mathbf{x}}^{(k)}\right\|$

R. Hiptmair

rev 30401, November 15, 2010

Observation:
- linear convergence ($\rightarrow$ Def. 4.1.6), see also Rem. 4.1.10
- rate of convergence increases ($\leftrightarrow$ speed of convergence decreases) with spread of spectrum of $\mathbf{A}$

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

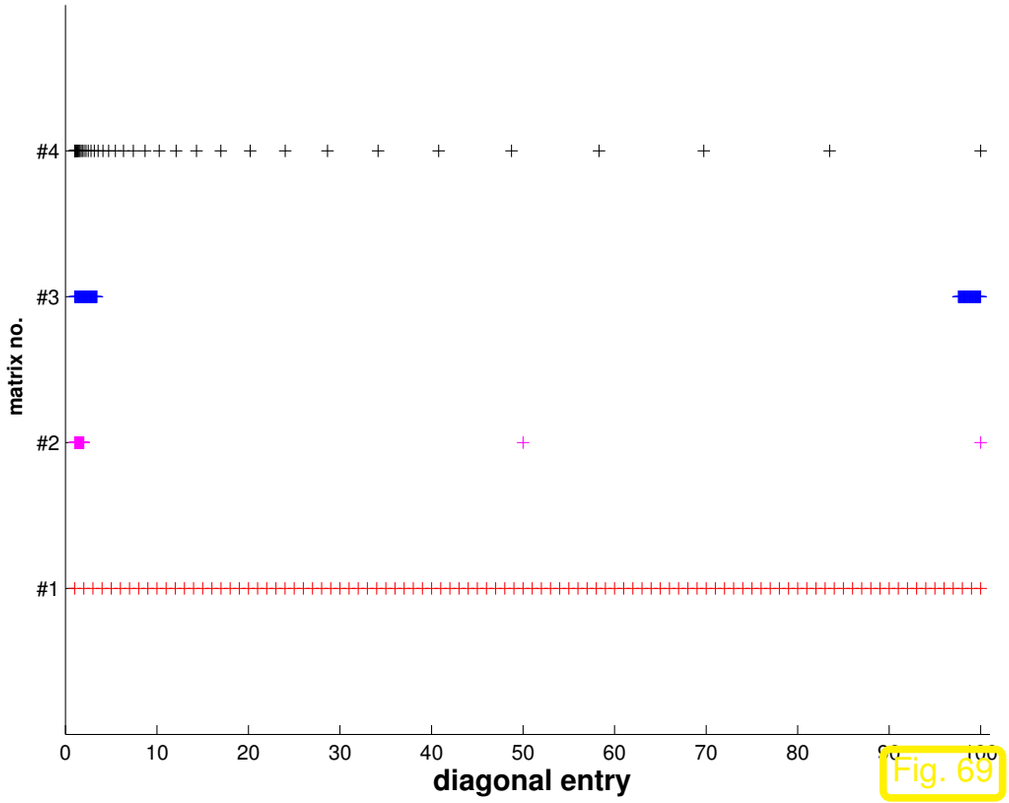
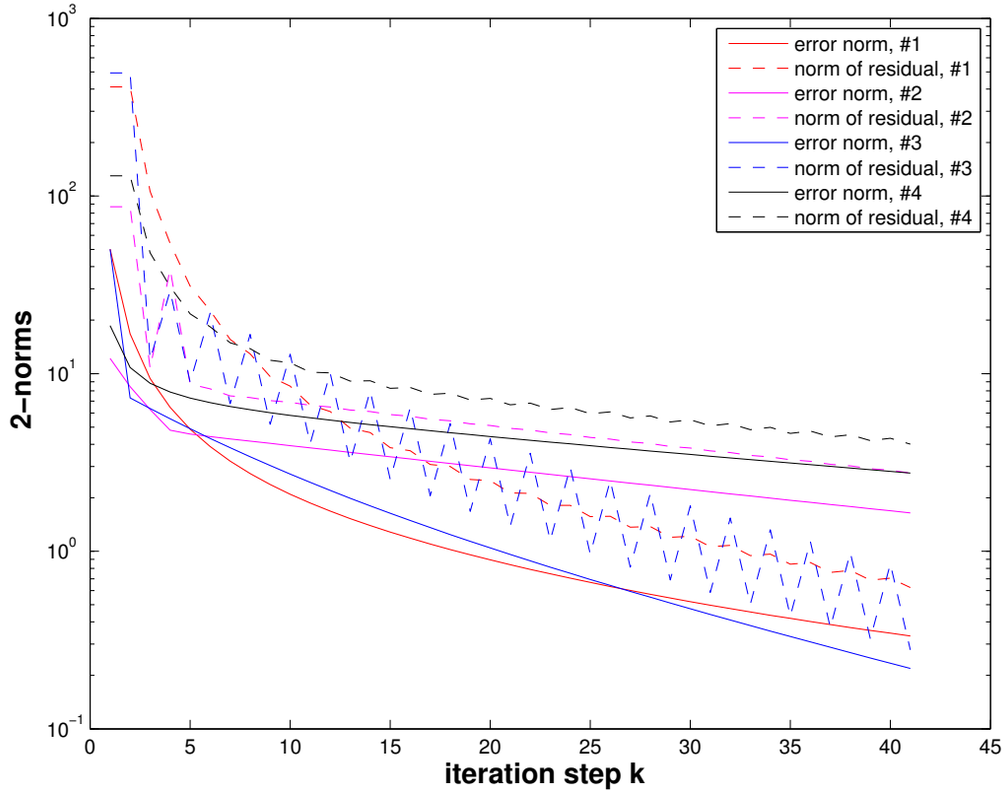


Fig. 69



Observation: Matrices #1, #2 & #4 ➢ little impact of distribution of eigenvalues on *asymptotic* convergence (exception: matrix #2)

◇

Theory [24, Sect. 9.2.2]:

R. Hiptmair

rev 30401, November 15, 2010

Theorem 5.1.19 (Convergence of gradient method/steepest descent)**.**

The iterates of the gradient method of Alg. 5.1.11 satisfy

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

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

✎ notation: $\operatorname{cond}_2(\mathbf{A}) \mathrel{\hat{=}}$ condition number of $\mathbf{A}$ induced by 2-norm

Remark 5.1.20 (2-norm from eigenvalues). → [23, Sect. 10.6], [39, Sect. 7.4]

$$\mathbf{A}=\mathbf{A}^T \;\Rightarrow\; \begin{aligned} &\|\mathbf{A}\|_2 = \max(|\sigma(\mathbf{A})|) \;, \\ &\left\|\mathbf{A}^{-1}\right\|_2 = \min(|\sigma(\mathbf{A})|)^{-1} \end{aligned} \quad \text{, if } \mathbf{A} \text{ regular.} \tag{5.1.21}$$

$$\mathbf{A}=\mathbf{A}^T \;\Rightarrow\; \operatorname{cond}_2(\mathbf{A}) = \frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})} \;, \quad \text{where} \quad \begin{aligned} &\lambda_{\max}(\mathbf{A}) := \max(|\sigma(\mathbf{A})|) \;, \\ &\lambda_{\min}(\mathbf{A}) := \min(|\sigma(\mathbf{A})|) \;. \end{aligned} \tag{5.1.22}$$

✎ other notation $\kappa(\mathbf{A}) := \dfrac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})} \mathrel{\hat{=}}$ spectral condition number of $\mathbf{A}$

(for general $\mathbf{A}$: $\lambda_{\max}(\mathbf{A})/\lambda_{\min}(\mathbf{A})$ largest/smallest eigenvalue *in modulus*)

R. Hiptmair

rev 30401, November 15, 2010

These results are an immediate consequence of the fact that

$$\forall \mathbf{A} \in \mathbb{R}^{n,n}, \quad \mathbf{A}^T = \mathbf{A} \quad \exists \mathbf{U} \in \mathbb{R}^{n,n}, \quad \mathbf{U}^{-1} = \mathbf{U}^T : \quad \mathbf{U}^T \mathbf{A} \mathbf{U} \quad \text{is diagonal,}$$

see (5.1.16), Cor. 6.1.9, [39, Thm. 7.8], [23, Satz 9.15].

Please note that for general regular $\mathbf{M} \in \mathbb{R}^{n,n}$ we *cannot* expect $\mathrm{cond}_2(\mathbf{M}) = \kappa(\mathbf{M})$.

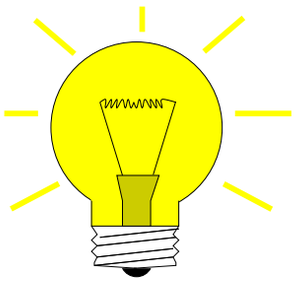
△

5.2 Conjugate gradient method (CG) [29, Ch. 9], [10, Sect. 13.4]

Again we consider a linear system of equations $\mathbf{Ax} = \mathbf{b}$ with s.p.d. ($\rightarrow$ Def. 2.7.9) system matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ and given $\mathbf{b} \in \mathbb{R}^n$.

Liability of gradient method of Sect. 5.1.3: NO MEMORY

1D line search in Alg. 5.1.11 is oblivious of former line searches, which rules out reuse of information gained in previous steps of the iteration. This is a typical drawback of 1-point iterative methods.



Idea: Replace linear search with subspace correction

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

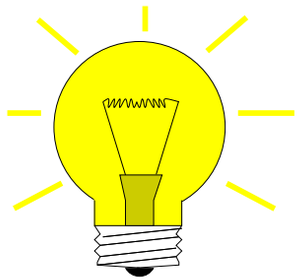
$$\mathbf{x}^{(k)} := \operatorname*{argmin}_{\mathbf{x} \in U_k + \mathbf{x}^{(0)}} J(x)\,, \tag{5.2.1}$$

$J(x)$: quadratic functional from (5.1.4)

Note: Once the subspaces U_k and $\mathbf{x}^{(0)}$ are fixed, the iteration (5.2.1) is well defined, because $J_{|U_k+\mathbf{x}^{(0)}}$ always possess a unique minimizer.

Obvious (from Lemma 5.1.3): $\mathbf{x}^{(n)} = \mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$

Thanks to (5.1.5), definition (5.2.1) ensures: $\boxed{\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\|_A \leq \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|_A}$



How to find suitable subspaces U_k **?**

Idea: $U_{k+1} \leftarrow U_k +$ " local steepest descent direction"

given by $-\mathbf{grad}\, J(\mathbf{x}^{(k)}) = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} = \mathbf{r}_k$ (residual $\rightarrow$ Def. 2.5.16)

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

R. Hiptmair

rev 30401, November 15, 2010

Obvious: $\mathbf{r}_k = 0 \quad \Rightarrow \quad \mathbf{x}^{(k)} = \mathbf{x}^* := \mathbf{A}^{-1}\mathbf{b}$ done ✔

Lemma 5.2.3 ($\mathbf{r}_k \perp U_k$).
With $\mathbf{x}^{(k)}$ according to (5.2.1), *U_k from* (5.2.2) *the residual $\mathbf{r}_k := \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$ satisfies*

$$\mathbf{r}_k^T \mathbf{u} = 0 \quad \forall \mathbf{u} \in U_k \quad (\text{"}\mathbf{r}_k \perp U_k\text{"}).$$

Geometric consideration: since $\mathbf{x}^{(k)}$ is the minimizer of J over the affine space $U_k + \mathbf{x}^{(0)}$, the projection of the steepest descent direction $\mathbf{grad}\, J(\mathbf{x}^{(k)})$ onto U_k has to vanish:

$$\mathbf{x}^{(k)} := \operatorname*{argmin}_{\mathbf{x} \in U_k + \mathbf{x}^{(0)}} J(x) \quad \Rightarrow \quad \mathbf{grad}\, J(\mathbf{x}^{(k)}) \perp U_k \,. \tag{5.2.4}$$

Formal proof. Consider

$$\psi(t) = J(\mathbf{x}^{(k)} + t\mathbf{u}) \,, \quad \mathbf{u} \in U_k, \quad t \in \mathbb{R} \,.$$

By (5.2.1), $t \mapsto \psi(t)$ has a global minimum in $t = 0$, which implies

$$\frac{d\psi}{dt}(0) = \mathbf{grad}\, J(\mathbf{x}^{(k)})^T \mathbf{u} = (\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b})^T \mathbf{u} = 0 \,.$$

Since $\mathbf{u} \in U_k$ was arbitrary, the lemma is proved. □

Corollary 5.2.5. *If* $\mathbf{r}_l \neq 0$ *for* $l = 0, \ldots, k$, $k \leq n$, *then* $\{\mathbf{r}_0, \ldots, \mathbf{r}_k\}$ *is an orthogonal basis of* U_k.

Lemma 5.2.3 also implies that, if $U_0 = \{0\}$, then $\dim U_k = k$ as long as $\mathbf{x}^{(k)} \neq \mathbf{x}^*$, that is, before we have converged to the exact solution.

(5.2.1) and (5.2.2) define the conjugate gradient method (CG) for the iterative solution of

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

(hailed as a "top ten algorithm" of the 20th century, SIAM News, 33(4))

5.2.1 Krylov spaces

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

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

Equivalent definition: $\mathcal{K}_l(\mathbf{A}, \mathbf{z}) = \{p(\mathbf{A})\mathbf{z}\colon\ p \text{ polynomial of degree } \leq l\}$

Lemma 5.2.7. *The subspaces $U_k \subset \mathbb{R}^n$, $k \geq 1$, defined by* (5.2.1) *and* (5.2.2) *satisfy*

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

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

Proof. (by induction) Obviously $\mathbf{A}\mathcal{K}_k(\mathbf{A},\mathbf{r}_0) \subset \mathcal{K}_{k+1}(\mathbf{A},\mathbf{r}_0)$. In addition

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}(\mathbf{x}^{(0)} + \mathbf{z}) \quad \text{for some } \mathbf{z} \in U_k \quad \Rightarrow \quad \mathbf{r}_k = \underbrace{\mathbf{r}_0}_{\in \mathcal{K}_{k+1}(\mathbf{A},\mathbf{r}_0)} - \underbrace{\mathbf{A}\mathbf{z}}_{\in \mathcal{K}_{k+1}(\mathbf{A},\mathbf{r}_0)} .$$

Since $U_{k+1} = \operatorname{Span}\{U_k, \mathbf{r}_k\}$, we obtain $U_{k+1} \subset \mathcal{K}_{k+1}(\mathbf{A},\mathbf{r}_0)$. Dimensional considerations based on Lemma 5.2.3 finish the proof. □

5.2.2 Implementation of CG

Assume: basis $\{\mathbf{p}_1, \ldots, \mathbf{p}_l\}$, $l = 1, \ldots, n$, of $\mathcal{K}_l(\mathbf{A},\mathbf{r})$ available

$$\mathbf{x}^{(l)} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A},\mathbf{r}_0) \quad \succ \quad \text{set} \quad \mathbf{x}^{(l)} = \mathbf{x}^{(0)} + \gamma_1\mathbf{p}_1 + \cdots + \gamma_l\mathbf{p}_l .$$

For $\psi(\gamma_1, \ldots, \gamma_l) := J(\mathbf{x}^{(0)} + \gamma_1\mathbf{p}_1 + \cdots + \gamma_l\mathbf{p}_l)$ holds

$$(5.2.1) \quad \Leftrightarrow \quad \frac{\partial \psi}{\partial \gamma_j} = 0 , \quad j = 1, \ldots, l .$$

This leads to a linear system of equations by which the coefficients γ_j can be computed:

$$\begin{pmatrix} \mathbf{p}_1^T\mathbf{A}\mathbf{p}_1 & \cdots & \mathbf{p}_1^T\mathbf{A}\mathbf{p}_l \\ \vdots & & \vdots \\ \mathbf{p}_l^T\mathbf{A}\mathbf{p}_1 & \cdots & \mathbf{p}_l^T\mathbf{A}\mathbf{p}_l \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_l \end{pmatrix} = \begin{pmatrix} \mathbf{p}_1^T\mathbf{r} \\ \vdots \\ \mathbf{p}_l^T\mathbf{r} \end{pmatrix} . \tag{5.2.8}$$

Great simplification, if $\{\mathbf{p}_1, \ldots, \mathbf{p}_l\}$ $\mathbf{A}$-orthogonal basis of $\mathcal{K}_l(\mathbf{A}, \mathbf{r})$: $\mathbf{p}_j^T\mathbf{A}\mathbf{p}_i = 0$ for $i \neq j$.

Recall: s.p.d. $\mathbf{A}$ induces an inner product ➢ concept of orthogonality [39, Sect. 4.4], [23, Sect. 6.2]

Assume: $\mathbf{A}$-orthogonal basis $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ of $\mathbb{R}^n$ available, such that

$$\boxed{\mathrm{Span}\,\{\mathbf{p}_1, \ldots, \mathbf{p}_l\} = \mathcal{K}_l(\mathbf{A}, \mathbf{r})} .$$

➢ (Efficient) successive computation of $\mathbf{x}^{(l)}$ becomes possible, see [10, Lemma 13.24]

(LSE (5.2.8) becomes diagonal !)

Input: : initial guess $\mathbf{x}^{(0)} \in \mathbb{R}^n$
Given: : $\mathbf{A}$-orthogonal bases $\{\mathbf{p}_1, \ldots, \mathbf{p}_l\}$ of $\mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)$, $l = 1, \ldots, n$
Output: : approximate solution $\mathbf{x}^{(l)} \in \mathbb{R}^n$ of $\mathbf{A}\mathbf{x} = \mathbf{b}$

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

for $j = 1$ **to** l **do** { $\mathbf{x}^{(j)} := \mathbf{x}^{(j-1)} + \dfrac{\mathbf{p}_j^T \mathbf{r}_0}{\mathbf{p}_j^T \mathbf{A}\mathbf{p}_j}\mathbf{p}_j$ } (5.2.9)

Task: Efficient computation of $\mathbf{A}$-orthogonal vectors $\{\mathbf{p}_1, \ldots, \mathbf{p}_l\}$ spanning $\mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)$ *during the CG iteration*.

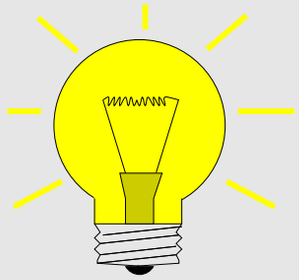
Lemma 5.2.3 implies orthogonality $\mathbf{p}_j \perp \mathbf{r}_m := \mathbf{b} - \mathbf{A}\mathbf{x}^{(m)}$, $1 \le j \le m \le l$. Also by $\mathbf{A}$-orthogonality of the $\mathbf{p}_k$

$$\mathbf{p}_j^T(\mathbf{b} - \mathbf{A}\mathbf{x}^{(m)}) = \mathbf{p}_j^T \left(\underbrace{\mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}}_{=\mathbf{r}_0} - \sum_{k=1}^{m} \frac{\mathbf{p}_k^T \mathbf{r}_0}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \mathbf{A}\mathbf{p}_k \right) = 0 \,. \tag{5.2.10}$$

From linear algebra we already know a way to construct orthogonal basis vectors:

R. Hiptmair
rev 30401, November 15, 2010

(5.2.10) $\Rightarrow$ Idea: Gram-Schmidt orthogonalization [39, Thm. 4.8], [23, Alg. 6.1], of residuals $\mathbf{r}_j := \mathbf{b} - \mathbf{A}\mathbf{x}^{(j)}$ w.r.t. $\mathbf{A}$-inner product:



$$\mathbf{p}_1 := \mathbf{r}_0 \ , \ \mathbf{p}_{j+1} := \underbrace{(\mathbf{b} - \mathbf{A}\mathbf{x}^{(j)})}_{=:\mathbf{r}_j} - \sum_{k=1}^{j} \frac{\mathbf{p}_k^T \mathbf{A}\mathbf{r}_j}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \mathbf{p}_k \ , \quad j = 1, \ldots, l-1 \quad . \tag{5.2.11}$$

Lemma 5.2.12 (Bases for Krylov spaces in CG)**.**
If they do not vanish, the vectors $\mathbf{p}_j$, $1 \le j \le l$, *and* $\mathbf{r}_j := \mathbf{b} - \mathbf{A}\mathbf{x}^{(j)}$, $0 \le j \le l$, *from* (5.2.9), (5.2.11) *satisfy*

(i) $\{\mathbf{p}_1, \ldots, \mathbf{p}_j\}$ *is* $\mathbf{A}$*-orthogonal basis von* $\mathcal{K}_j(\mathbf{A}, \mathbf{r}_0)$,

(ii) $\{\mathbf{r}_0, \ldots, \mathbf{r}_{j-1}\}$ *is orthogonal basis of* $\mathcal{K}_j(\mathbf{A}, \mathbf{r}_0)$, *cf. Cor. 5.2.5*

Proof. $\mathbf{A}$-orthogonality of $\mathbf{p}_j$ by construction, study (5.2.11).

$$\text{(5.2.9) \& (5.2.11)} \quad \Rightarrow \quad \mathbf{p}_{j+1} = \mathbf{r}_0 - \sum_{k=1}^{j} \frac{\mathbf{p}_k^T \mathbf{r}_0}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \mathbf{A}\mathbf{p}_k - \sum_{k=1}^{j} \frac{\mathbf{p}_k^T \mathbf{A}\mathbf{r}_j}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \mathbf{p}_k \ .$$

$$\Rightarrow \quad \mathbf{p}_{j+1} \in \operatorname{Span}\{\mathbf{r}_0, \mathbf{p}_1, \ldots, \mathbf{p}_j, \mathbf{A}\mathbf{p}_1, \ldots, \mathbf{A}\mathbf{p}_j\} \ .$$

A simple induction argument confirms (i)

$$(5.2.11) \quad \Rightarrow \quad \mathbf{r}_j \in \mathrm{Span}\left\{\mathbf{p}_1, \ldots, \mathbf{p}_{j+1}\right\} \quad \& \quad \mathbf{p}_j \in \mathrm{Span}\left\{\mathbf{r}_0, \ldots, \mathbf{r}_{j-1}\right\} . \tag{5.2.13}$$

$$\blacktriangleright \quad \boxed{\mathrm{Span}\left\{\mathbf{p}_1, \ldots, \mathbf{p}_j\right\} = \mathrm{Span}\left\{\mathbf{r}_0, \ldots, \mathbf{r}_{j-1}\right\} = \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)} . \tag{5.2.14}$$

$$(5.2.10) \quad \Rightarrow \quad \mathbf{r}_j \perp \mathrm{Span}\left\{\mathbf{p}_1, \ldots, \mathbf{p}_j\right\} = \mathrm{Span}\left\{\mathbf{r}_0, \ldots, \mathbf{r}_{j-1}\right\} . \tag{5.2.15}$$

□

Orthogonalities from Lemma 5.2.12 ➤ short recursions for $\mathbf{p}_k$, $\mathbf{r}_k$, $\mathbf{x}^{(k)}$!

(5.2.10) $\Rightarrow$ (5.2.11) collapses to $\mathbf{p}_{j+1} := \mathbf{r}_j - \dfrac{\mathbf{p}_j^T \mathbf{A}\mathbf{r}_j}{\mathbf{p}_j^T \mathbf{A}\mathbf{p}_j}\mathbf{p}_j , \quad j = 1, \ldots, l .$

recursion for residuals:

$$(5.2.9) \quad \blacktriangleright \quad \mathbf{r}_j = \mathbf{r}_{j-1} - \frac{\mathbf{p}_j^T \mathbf{r}_0}{\mathbf{p}_j^T \mathbf{A}\mathbf{p}_j}\mathbf{A}\mathbf{p}_j .$$

Lemma 5.2.12, (i) $\blacktriangleright$

$$\mathbf{r}_{j-1}^H \mathbf{p}_j = \left(\mathbf{r}_0 + \sum_{k=1}^{m-1} \frac{\mathbf{r}_0^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k}\mathbf{A}\mathbf{p}_k\right)^T \mathbf{p}_j = \mathbf{r}_0^T \mathbf{p}_j . \tag{5.2.16}$$

The orthogonality (5.2.16) together with (5.2.15) permits us to replace $\mathbf{r}_0$ with $\mathbf{r}_{j-1}$ in the actual implementation.

Algorithm 5.2.17 (CG method for solving $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}$ s.p.d.). $\rightarrow$ [10, 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 30401, November 15, 2010

1 matrix×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 $\mathbf{Ax} = \mathbf{b}$

```
function x = cg(A,b,x,tol,maxit)
% x supplies initial guess, maxit maximal number of CG steps
r = b - A * 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 = A * 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:

R. Hiptmair

rev 30401, November 15, 2010

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

`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

Remark 5.2.19 (A posteriori termination criterion for plain CG).

For any vector norm and associated matrix norm ($\rightarrow$ Def. 2.5.5) hold (with residual $\mathbf{r}_l := \mathbf{b} - \mathbf{A}\mathbf{x}^{(l)}$)

$$\frac{1}{\mathrm{cond}(\mathbf{A})}\frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|} \leq \frac{\left\|\mathbf{x}^{(l)}-\mathbf{x}^*\right\|}{\left\|\mathbf{x}^{(0)}-\mathbf{x}^*\right\|} \leq \mathrm{cond}(\mathbf{A})\,\frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|}\,. \tag{5.2.20}$$

relative decrease of iteration error

(5.2.20) can easily be deduced from the error equation $\mathbf{A}(\mathbf{x}^{(k)} - \mathbf{x}^*) = \mathbf{r}_k$, see Def. 2.5.16 and (2.5.19).

△

5.2.3 Convergence of CG

Note: CG is a *direct solver*, because (in exact arithmetic) $\mathbf{x}^{(k)} = \mathbf{x}^*$ for some $k \leq n$

Example 5.2.21 (Impact of roundoff errors on CG).

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

Hilbert-Matrix: extremely ill-conditioned

residual norms during CG iteration $\mathbf{R} = \left[\mathbf{r}_0, \ldots, \mathbf{r}^{(10)}\right]$

$\triangleright$:

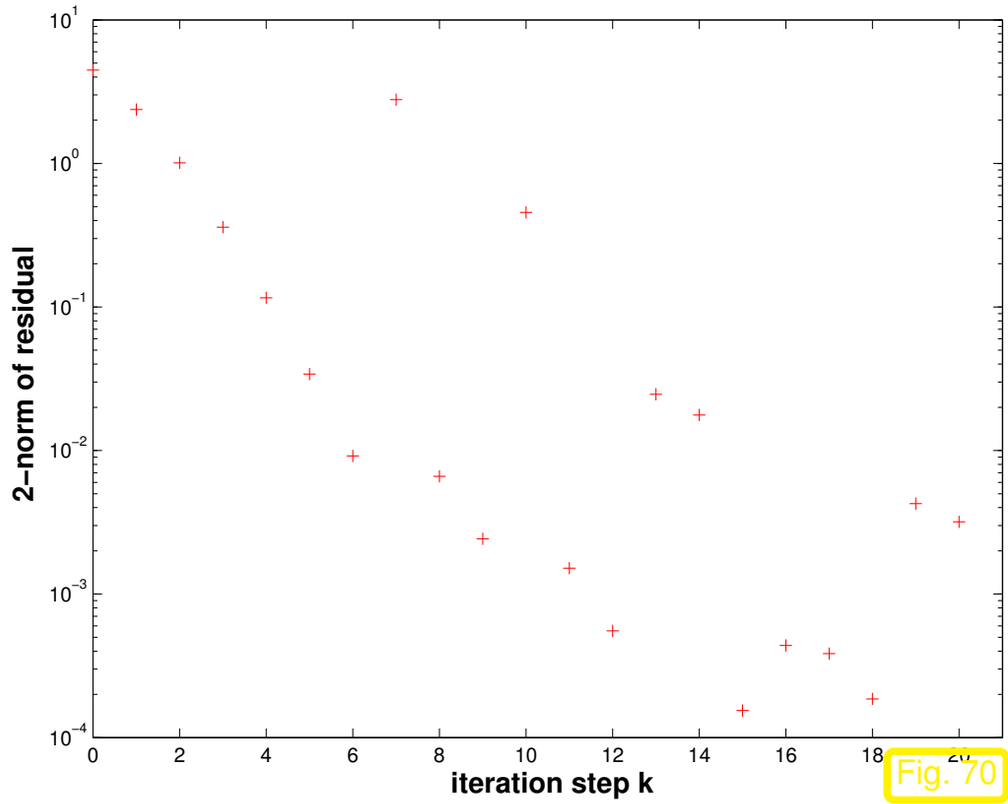


Fig. 70

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

➢ Roundoff
- destroys orthogonality of residuals
- prevents computation of exact solution after n steps.

➤ Numerical instability (→ Def. 2.5.11) ➢ pointless to (try to) use CG as direct solver!

◇

Practice: CG used for large n as *iterative solver*: $\mathbf{x}^{(k)}$ for some $k \ll n$ is expected to provide good approximation for $\mathbf{x}^*$

Example 5.2.22 (Convergence of CG as iterative solver).

CG (Code 5.2.17)& gradient method (Code 5.1.11) for LSE with sparse s.p.d. “Poisson matrix”

```
A = gallery('poisson',m); x0 = (1:n)'; b = zeros(n,1);
```

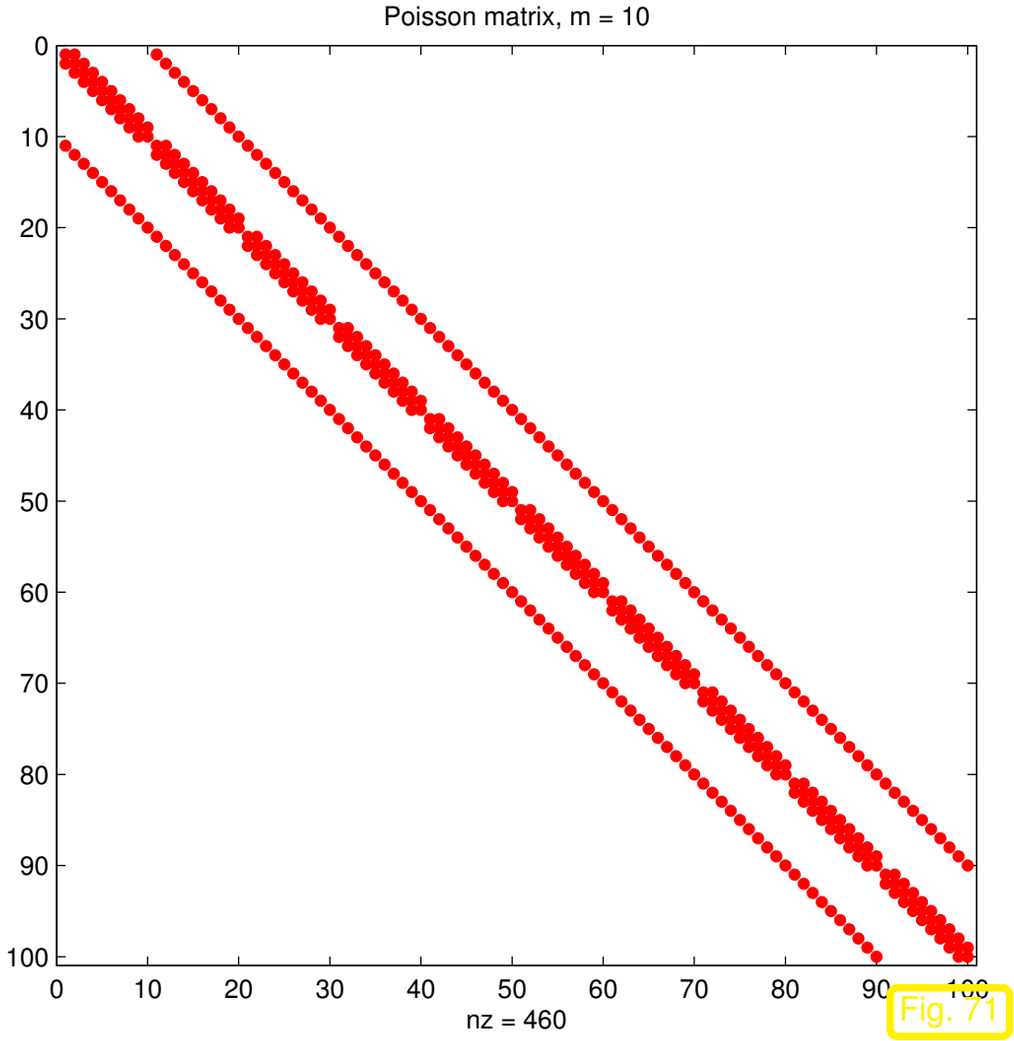


Fig. 71

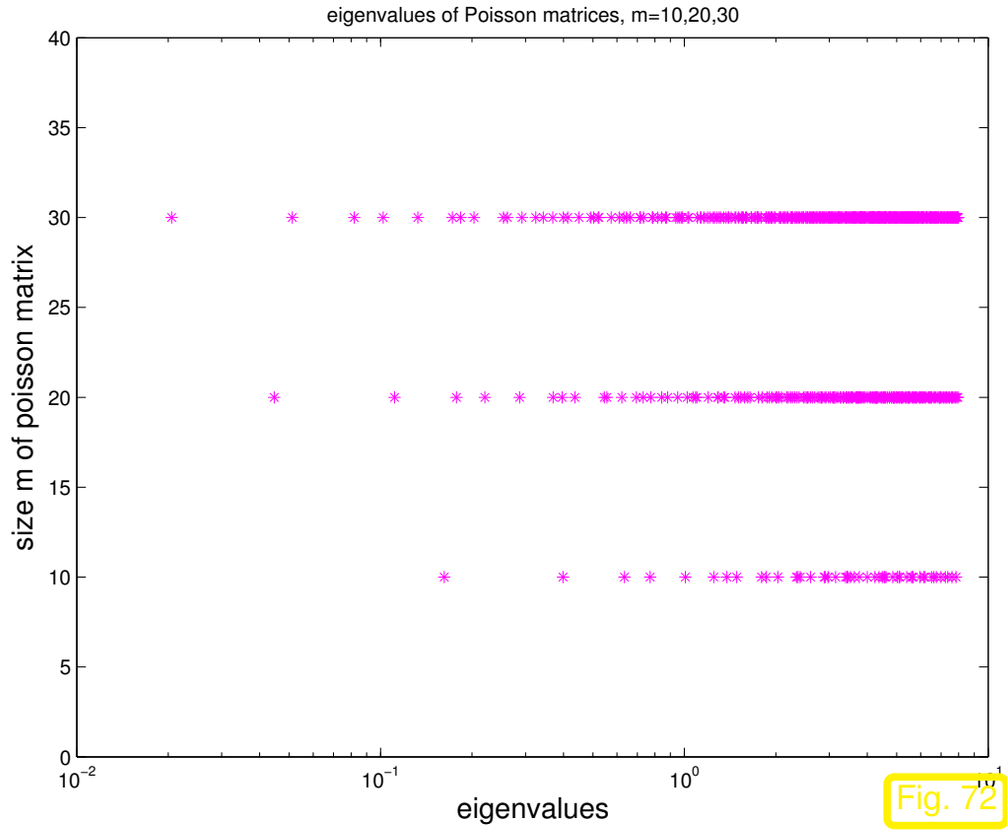


Fig. 72

R. Hiptmair

rev 30401, November 15, 2010

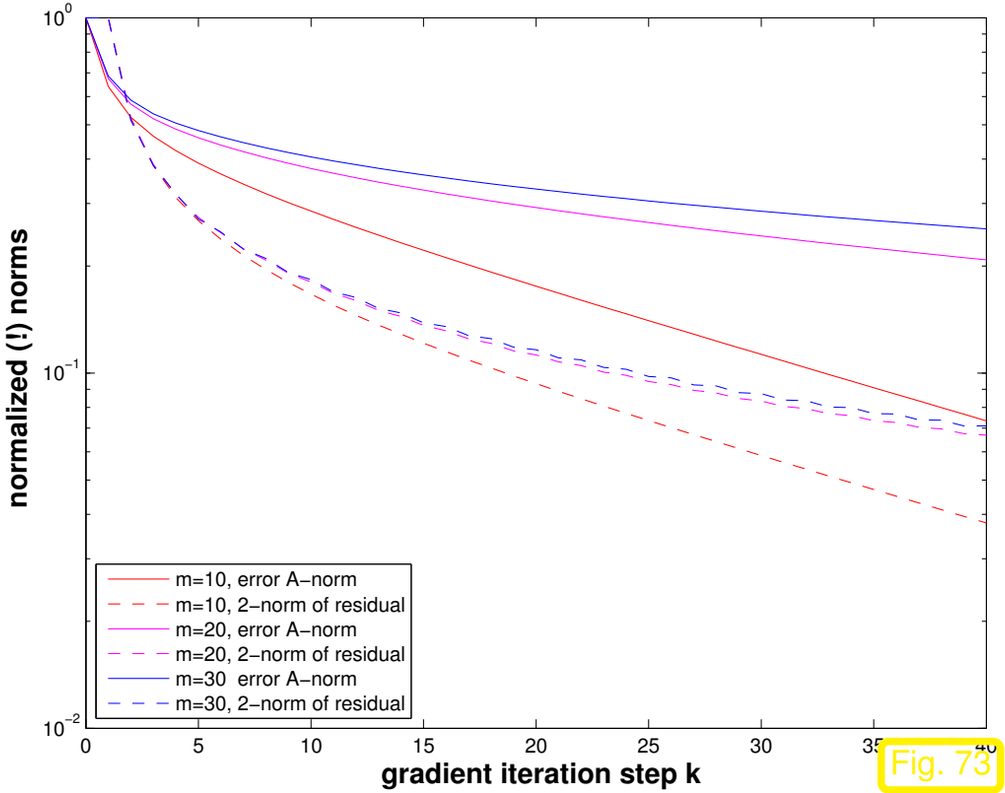


Fig. 73

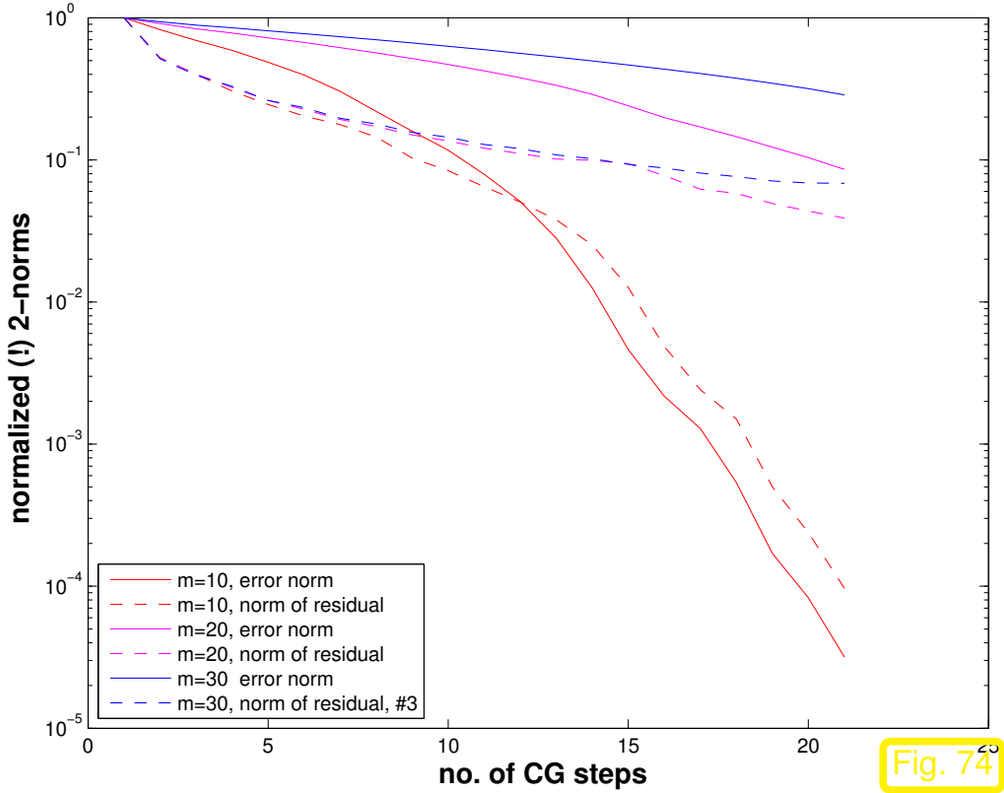


Fig. 74

R. Hiptmair

rev 30401, November 15, 2010

Observations:

- CG much faster than gradient method (as expected, because it has “memory”)
- Both, CG and gradient method converge more slowly for larger sizes of Poisson matrices.

Convergence theory:

A simple consequence of (5.1.5) and (5.2.1):

Corollary 5.2.23 ("Optimality" of CG iterates)**.**
Writing $\mathbf{x}^* \in \mathbb{R}^n$ for the exact solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ the CG iterates satisfy

$$\left\|\mathbf{x}^* - \mathbf{x}^{(l)}\right\|_A = \min\{\|\mathbf{y} - \mathbf{x}^*\|_A : \mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\} \quad , \quad \mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)} \, .$$

This paves the way for a quantitative convergence estimate:

$$\mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}) \quad \Leftrightarrow \quad \mathbf{y} = \mathbf{x}^{(0)} + \mathbf{A}\, p(\mathbf{A})(\mathbf{x} - \mathbf{x}^{(0)}) \, , \quad p \text{ = polynomial of degree } \leq l - 1 \, .$$

$$\blacktriangleright \quad \mathbf{x} - \mathbf{y} = q(\mathbf{A})(\mathbf{x} - \mathbf{x}^{(0)}), \quad q \text{ = polynomial of degree } \leq l \, , \quad q(0) = 1 \, .$$

R. Hiptmair

rev 30401, November 15, 2010

$$\left\|\mathbf{x}-\mathbf{x}^{(l)}\right\|_A \leq \boxed{\min\{\max_{\lambda\in\sigma(\mathbf{A})}|q(\lambda)|\text{: } q \text{ polynomial of degree } \leq l\ ,\quad q(0)=1\}} \cdot \left\|\mathbf{x}-\mathbf{x}^{(0)}\right\|_A\ . \tag{5.2.24}$$

Bound this minimum for $\lambda \in [\lambda_{\min}(\mathbf{A}), \lambda_{\max}(\mathbf{A})]$ by using suitable "polynomial candidates"

Tool: Chebychev polynomials ➢ lead to the following estimate [24, Satz 9.4.2], [10, Satz 13.29]

Theorem 5.2.25 (Convergence of CG method)**.**

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

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

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

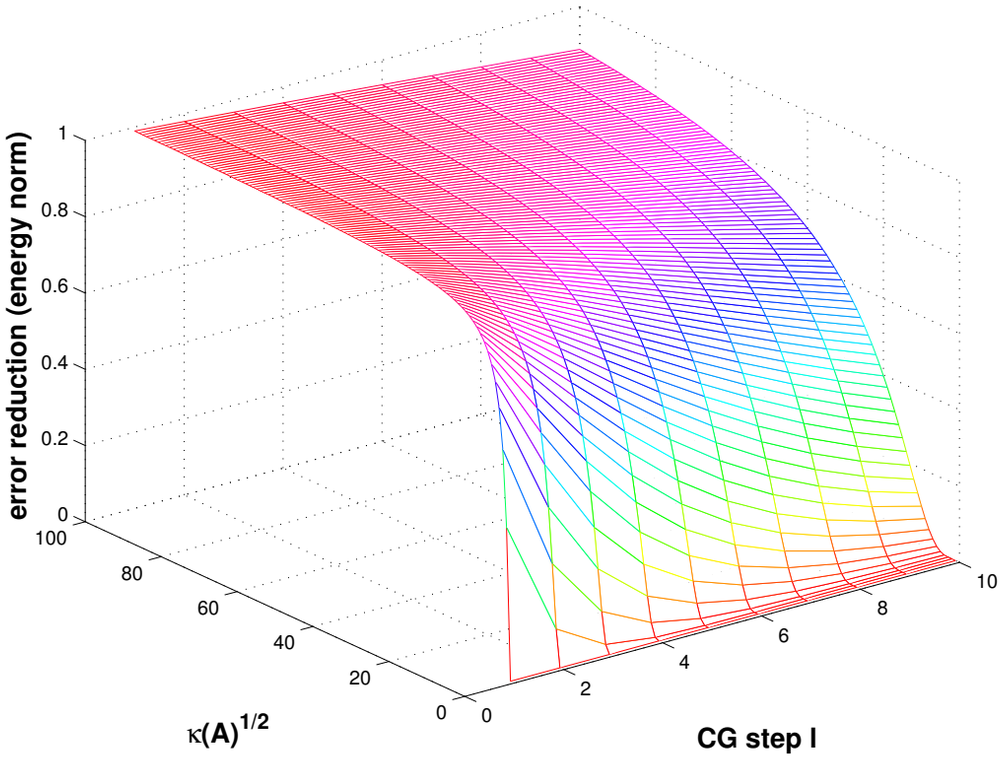
(recall: $\kappa(\mathbf{A})$ *= spectral condition number of* $\mathbf{A}$*,* $\kappa(\mathbf{A}) = \mathrm{cond}_2(\mathbf{A})$*)*

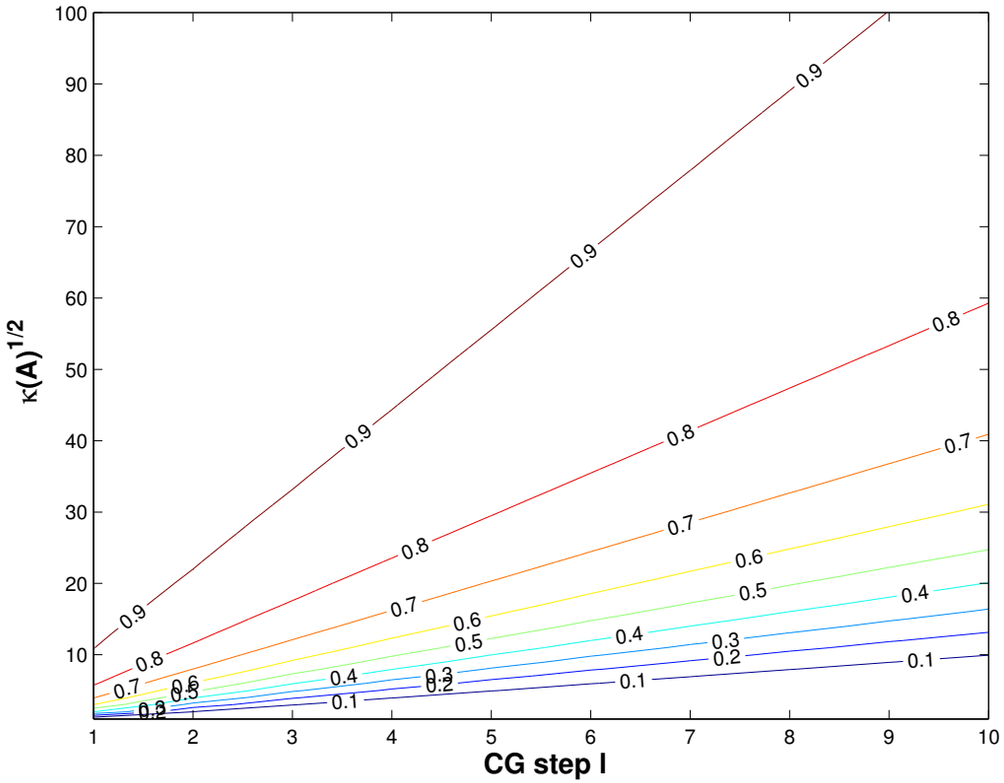
R. Hiptmair

rev 30401, November 15, 2010

The estimate of this theorem confirms *asymptotic linear convergence* of the CG method ($\rightarrow$ Def. 4.1.6) with a rate of $\dfrac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1}$

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





Code 5.2.26: plotting theoretical bounds for CG convergence rate

```
1 function plottheorate
2 % Python:  ../PYTHON/ConjugateGradient.py plottheorate()
3
4 [X,Y] = meshgrid(1:10,1:100); R = zeros(100,10);
5 for I=1:100
6   t = 1/I;
7   for j=1:10
```

```
8       R(I,j) = 2*(1-t)^j/((1+t)^(2*j)+(1-t)^(2*j));
9     end
10  end
11
12  figure; view([-45,28]); mesh(X,Y,R); colormap hsv;
13  xlabel('{\bf CG step l}','Fontsize',14);
14  ylabel('{\bf \kappa(A)^{1/2}}','Fontsize',14);
15  zlabel('{\bf error reduction (energy norm)}','Fontsize',14);
16
17  print -depsc2 '../PICTURES/theorate1.eps';
18
19  figure; [C,h] = contour(X,Y,R); clabel(C,h);
20  xlabel('{\bf CG step l}','Fontsize',14);
21  ylabel('{\bf \kappa(A)^{1/2}}','Fontsize',14);
22
23  print -depsc2 '../PICTURES/theorate2.eps';
```

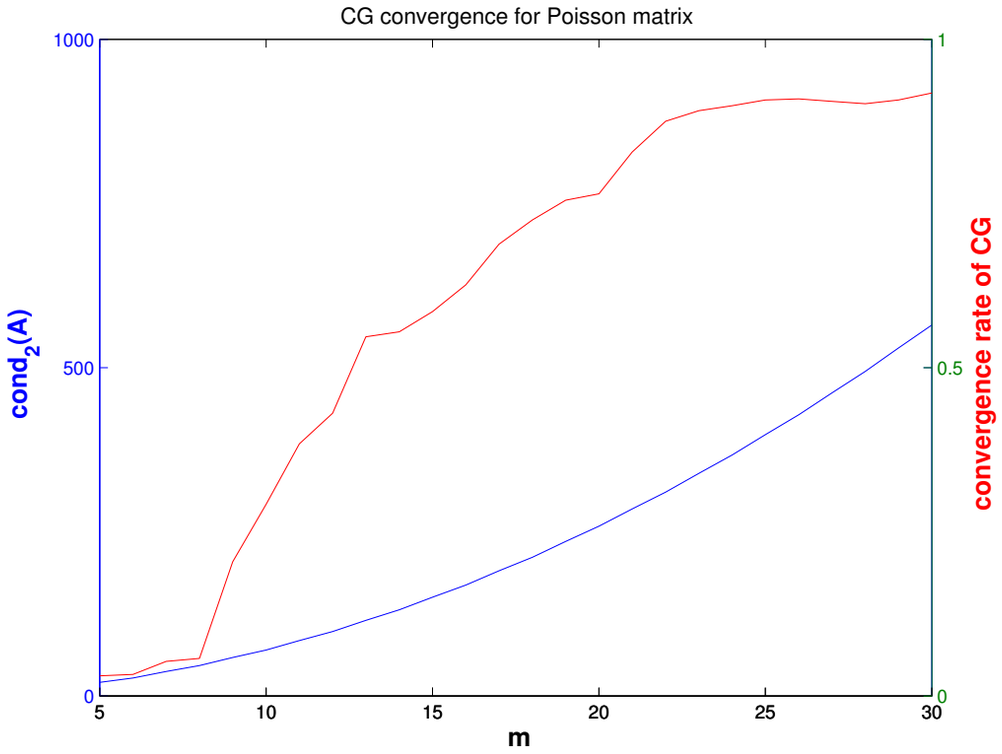
Example 5.2.27 (Convergence rates for CG method).

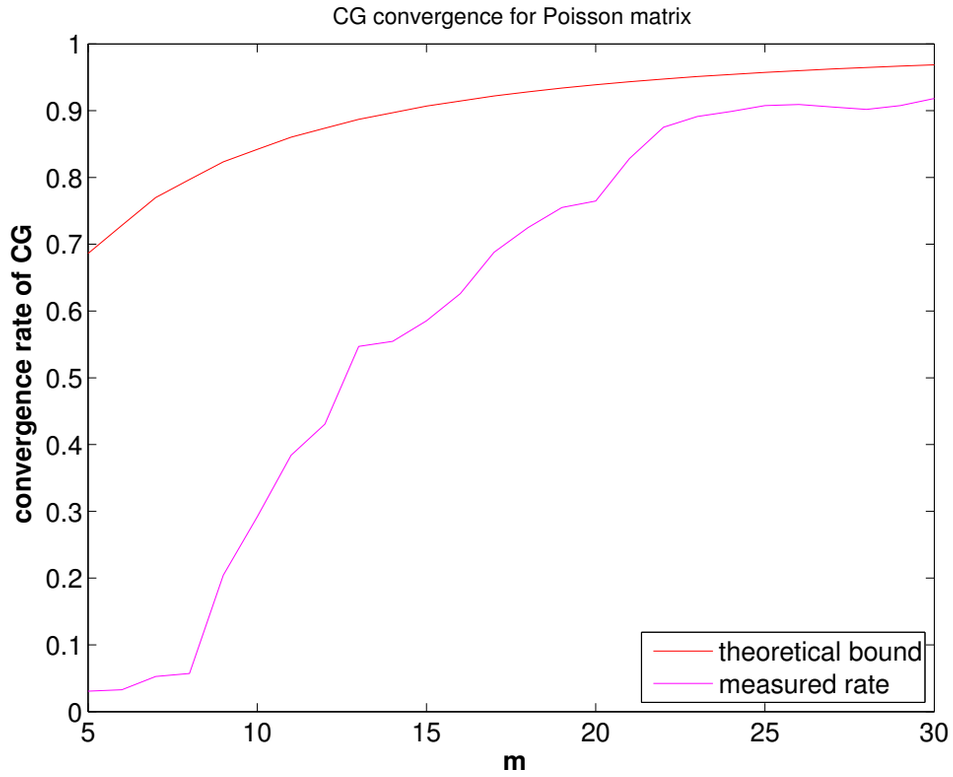
Code 5.2.28: CG for Poisson matrix

```
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:

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





Justification for estimating the rate of linear convergence (→ Def. 4.1.6) of $\|\mathbf{r}_k\|_2 \to 0$:

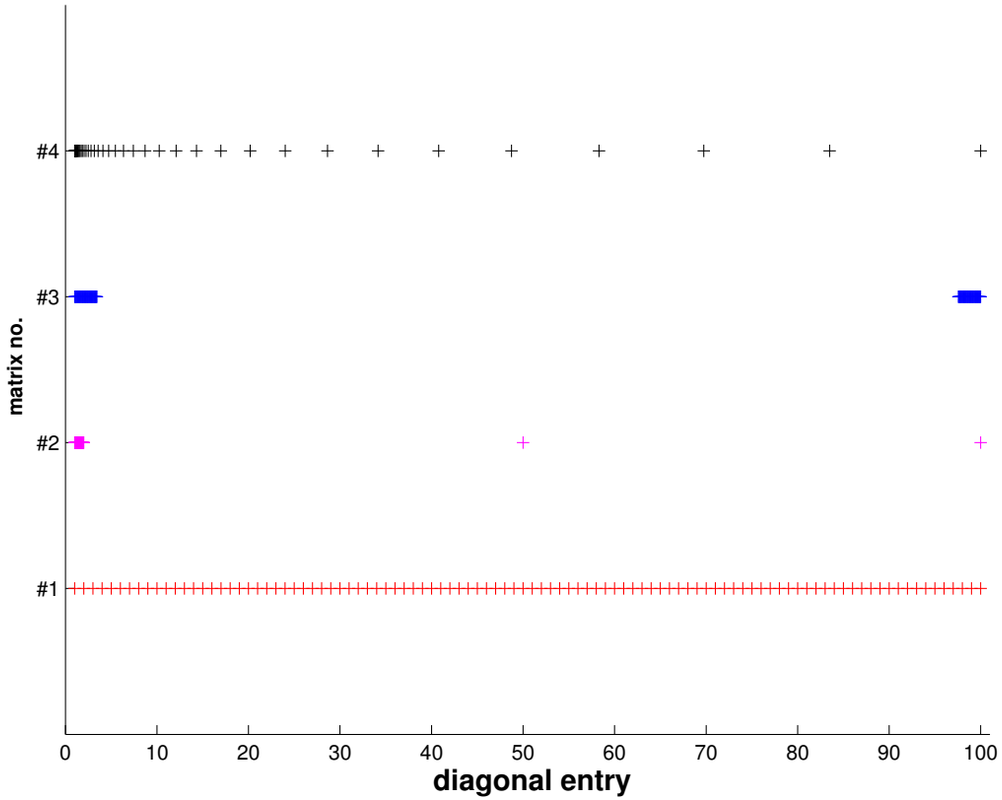
$$\|\mathbf{r}_{k+1}\|_2 \approx L\,\|\mathbf{r}_k\|_2 \quad \Rightarrow \quad \|\mathbf{r}_{k+m}\|_2 \approx L^m\,\|\mathbf{r}_k\|_2 \ .$$

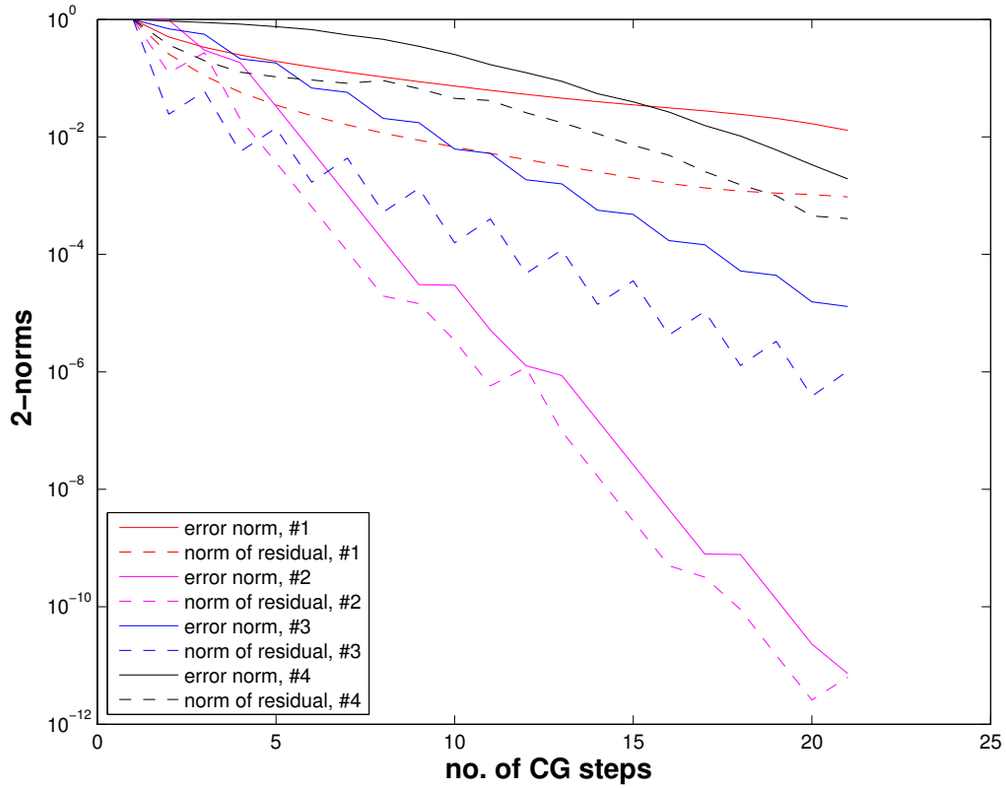
◇

Example 5.2.30 (CG convergence and spectrum). → Ex. 5.1.18

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);
```





Observations: Distribution of eigenvalues has crucial impact on convergence of CG

(This is clear from the convergence theory, because detailed information about the spectrum allows a much better choice of "candidate polynomial" in (5.2.24) than merely using Chebychev polynomials)

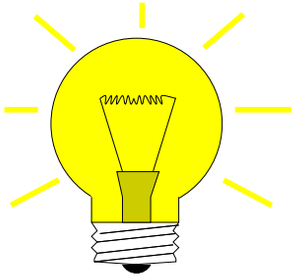
➢ Clustering of eigenvalues leads to faster convergence of CG

(in stark contrast to the behavior of the gradient method, see Ex. 5.1.18)

◇

5.3 Preconditioning [10, Sect. 13.5], [29, Ch. 10]

Thm. 5.2.25 ➢ (Potentially) slow convergence of CG in case $\kappa(\mathbf{A}) \gg 1$.



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

Remark 5.3.2 (Square root of a s.p.d. matrix).

What is meant by the "square root" $\mathbf{B}^{1/2}$ of a s.p.d. matrix $\mathbf{B}$ **?**

Recall (5.1.16) : for every $\mathbf{B} \in \mathbb{R}^{n,n}$ with $\mathbf{B}^T = \mathbf{B}$ there is an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n.n}$ such that $\mathbf{B} = \mathbf{Q}^T\mathbf{D}\mathbf{Q}$ with a diagonal matrix $\mathbf{D}$ ($\rightarrow$ Cor. 6.1.9, [39, Thm. 7.8], [33, Satz 9.15]). If $\mathbf{B}$ is s.p.d. the (diagonal) entries of $\mathbf{D}$ are strictly positive and we can define

$$\mathbf{D} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n), \quad \lambda_i > 0 \quad \Rightarrow \quad \mathbf{D}^{1/2} := \operatorname{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n}) \ .$$

This is generalized to

$$\mathbf{B}^{1/2} := \mathbf{Q}^T\mathbf{D}^{1/2}\mathbf{Q} \ ,$$

and one easily verifies, using $\mathbf{Q}^T = \mathbf{Q}^{-1}$, that $(\mathbf{B}^{1/2})^2 = \mathbf{B}$ and that $\mathbf{B}^{1/2}$ is s.p.d. In fact, these two requirements already determine $\mathbf{B}^{1/2}$ uniquely:

$\mathbf{B}^{1/2}$ is the unique s.p.d. matrix such that $(\mathbf{B}^{1/2})^2 = \mathbf{B}$.

$\triangle$

R. Hiptmair

rev 30401, November 15, 2010

Notion 5.3.3 (Preconditioner).

*A s.p.d. matrix $\mathbf{B} \in \mathbb{R}^{n,n}$ is called a preconditioner (*ger.: *Vorkonditionierer) for the s.p.d. matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, if*

1. *$\kappa(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2})$ is "small" and*
2. *the evaluation of $\mathbf{B}^{-1}\mathbf{x}$ is about as expensive (in terms of elementary operations) as the matrix×vector multiplication $\mathbf{A}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^n$.*

Recall: spectral condition number $\kappa(\mathbf{A}) := \dfrac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})}$, see (5.1.22)

There are several equivalent ways to express that $\kappa(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2})$ is "small":

- $\kappa(\mathbf{B}^{-1}\mathbf{A})$ is "small",

  because spectra agree $\sigma(\mathbf{B}^{-1}\mathbf{A}) = \sigma(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2})$ due to similarity ($\rightarrow$ Lemma 6.1.6)
- $\exists 0 < \gamma < \Gamma$, Γ/γ "'small": $\gamma\,(\mathbf{x}^T\mathbf{B}\mathbf{x}) \leq \mathbf{x}^T\mathbf{A}\mathbf{x} \leq \Gamma\,(\mathbf{x}^T\mathbf{B}\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^n$,

  where equivalence is seen by transforming $\mathbf{y} := \mathbf{B}^{-1/2}\mathbf{x}$ and appealing to the min-max Theorem 6.3.35.

R. Hiptmair

rev 30401, November 15, 2010

"Reader's digest" version of notion 5.3.3:

S.p.d. $\mathbf{B}$ preconditioner $:\Leftrightarrow$ $\mathbf{B}^{-1}$ = cheap approximate inverse of $\mathbf{A}$

Problem: $\mathbf{B}^{1/2}$, which occurs prominently in (5.3.1) is usually not available with acceptable computational costs.

However, if one formally applies Algorithm 5.2.17 to the transformed system

$$\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} := \left(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2}\right)(\mathbf{B}^{1/2}\mathbf{x}) = \widetilde{\mathbf{b}} := \mathbf{B}^{-1/2}\mathbf{b}\ ,$$

from (5.3.1), it becomes apparent that, after suitable transformation of the iteration variables $\mathbf{p}_j$ and $\mathbf{r}_j$, $\mathbf{B}^{1/2}$ and $\mathbf{B}^{-1/2}$ invariably occur in products $\mathbf{B}^{-1/2}\mathbf{B}^{-1/2} = \mathbf{B}^{-1}$ and $\mathbf{B}^{1/2}\mathbf{B}^{-1/2} = \mathbf{I}$. Thus, thanks to this intrinsic transformation square roots of $\mathbf{B}$ are not required for the implementation!

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

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** {

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

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

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

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

}

R. Hiptmair

rev 30401, November 15, 2010

with the transformations:

$$\widetilde{\mathbf{x}}^{(k)} = \mathbf{B}^{1/2}\mathbf{x}^{(k)} \quad , \quad \widetilde{\mathbf{r}}_k = \mathbf{B}^{-1/2}\mathbf{r}_k \quad , \quad \widetilde{\mathbf{p}}_k = \mathbf{B}^{-1/2}\mathbf{r}_k \; . \tag{5.3.4}$$

Algorithm 5.3.5 (Preconditioned CG method (PCG)). [10, Alg. 13.32], [29, 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 $\mathbf{x} \hat{=} \mathbf{x}^{(l)}$

$\mathbf{p} := \mathbf{r} := \mathbf{b} - \mathbf{A}\mathbf{x}$; $\mathbf{p} := \mathbf{B}^{-1}\mathbf{r}$; $\mathbf{q} := \mathbf{p}$; $\tau_0 := \mathbf{p}^T\mathbf{r}$;
for $l = 1$ **to** $l_{\max}$ **do** {
  $\beta := \mathbf{r}^T\mathbf{q}$; $\mathbf{h} := \mathbf{A}\mathbf{p}$; $\alpha := \frac{\beta}{\mathbf{p}^T\mathbf{h}}$;
  $\mathbf{x} := \mathbf{x} + \alpha\mathbf{p}$;
  $\mathbf{r} := \mathbf{r} - \alpha\mathbf{h}$;
  $\mathbf{q} := \mathbf{B}^{-1}\mathbf{r}$; $\beta := \frac{\mathbf{r}^T\mathbf{q}}{\beta}$;
  if $|\mathbf{q}^T\mathbf{r}| \leq \tau \cdot \tau_0$ **then stop**;
  $\mathbf{p} := \mathbf{q} + \beta\mathbf{p}$;
  }

(5.3.6)

R. Hiptmair

rev 30401, November 15, 2010

Code 5.3.7: simple implementation of PCG algorithm Alg. 5.3.5

```
function [x,rn,xk] = pcgbase(evalA,b,tol,maxit,invB,x)
% python:  ../PYTHON/ConjugateGradient.py, pcgbase()
r = b - evalA(x); rho = 1; rn = [];
if (nargout > 2), xk = x; end
for i = 1 : maxit
  y = invB(r);
  rho_old = rho; rho = r' * y; rn = [rn,rho];
  if (i == 1), p = y; rho0 = rho;
  elseif (rho < rho0*tol), return;
  else beta = rho/rho_old; p = y+beta*p; end
  q = evalA(p); alpha = rho /(p' * q);
  x = x + alpha * p;
  r = r - alpha * q;
  if (nargout > 2), xk = [xk,x]; end
end
```

Computational effort per step: 1 evaluation $\mathbf{A}\times$vector,
1 evaluation $\mathbf{B}^{-1}\times$vector,
3 dot products, 3 AXPY-operations

R. Hiptmair

rev 30401, November 15, 2010

Remark 5.3.8 (Convergence theory for PCG).

Assertions of Thm. 5.2.25 remain valid with $\kappa(\mathbf{A})$ replaced with $\kappa(\mathbf{B}^{-1}\mathbf{A})$ and energy norm based on $\widetilde{\mathbf{A}}$ instead of $\mathbf{A}$.

△

Example 5.3.9 (Simple preconditioners).

$\mathbf{B}$ = easily invertible "part" of $\mathbf{A}$

- $\mathbf{B} = \texttt{diag}(\mathbf{A})$: Jacobi preconditioner (diagonal scaling)
- $(\mathbf{B})_{ij} = \begin{cases} (\mathbf{A})_{ij} & \text{, if } |i-j| \le k \text{ ,} \\ 0 & \text{else,} \end{cases}$ for some $k \ll n$.
- Symmetric Gauss-Seidel preconditioner

Idea: Solve $\mathbf{Ax} = \mathbf{b}$ approximately in two stages:

① Approximation $\mathbf{A}^{-1} \approx \mathtt{tril}(\mathbf{A})$ (lower triangular part): $\widetilde{\mathbf{x}} = \mathtt{tril}(\mathbf{A})^{-1}\mathbf{b}$

② Approximation $\mathbf{A}^{-1} \approx \mathtt{triu}(\mathbf{A})$ (upper triangular part) and use this to approximately "solve" the error equation $\mathbf{A}(\mathbf{x} - \widetilde{\mathbf{x}}) = \mathbf{r}$, with residual $\mathbf{r} := \mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}}$:

$$\mathbf{x} = \widetilde{\mathbf{x}} + \mathtt{triu}(\mathbf{A})^{-1}(\mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}}) \ .$$

With $\mathbf{L}_A := \mathtt{tril}(\mathbf{A})$, $\mathbf{U}_A := \mathtt{triu}(\mathbf{A})$ one finds

$$\mathbf{x} = (\mathbf{L}_A^{-1} + \mathbf{U}_A^{-1} - \mathbf{U}_A^{-1}\mathbf{A}\mathbf{L}_A^{-1})\mathbf{b} \quad \blacktriangleright \quad \mathbf{B}^{-1} = \mathbf{L}_A^{-1} + \mathbf{U}_A^{-1} - \mathbf{U}_A^{-1}\mathbf{A}\mathbf{L}_A^{-1} \ . \tag{5.3.10}$$

◇

More complicated preconditioning strategies:

- Incomplete Cholesky factorization, MATLAB-`ichol`, [10, Sect. 13.5]]
- Sparse approximate inverse preconditioner (SPAI)

Example 5.3.11 (Tridiagonal preconditioning).

Efficacy of preconditioning of sparse LSE with tridiagonal part:

Code 5.3.12: LSE for Ex. 5.3.11

```
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 30401, November 15, 2010

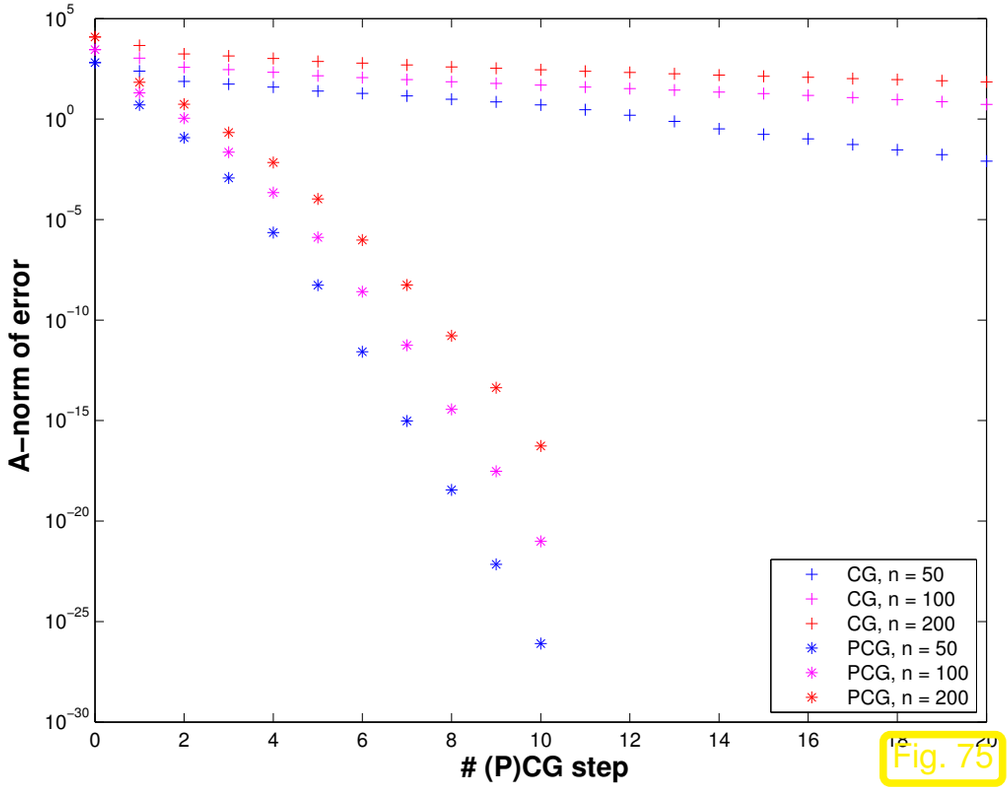

A–norm of error
(P)CG step
CG, n = 50
CG, n = 100
CG, n = 200
PCG, n = 50
PCG, n = 100
PCG, n = 200

Fig. 75

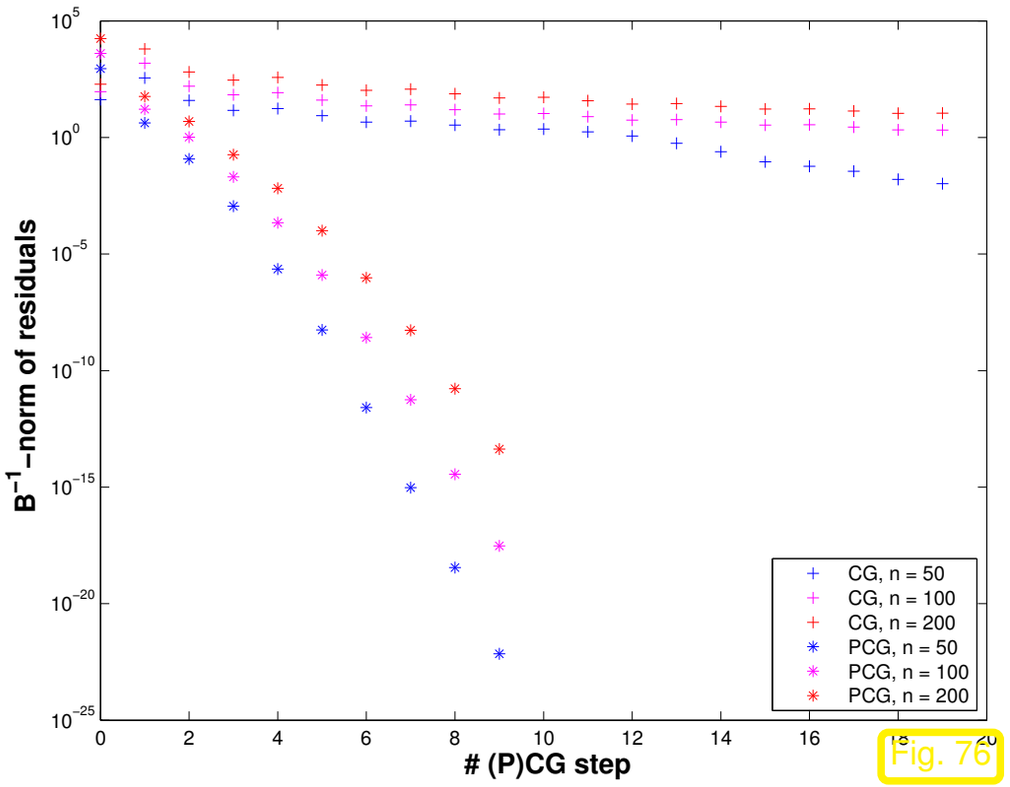

B^{-1}–norm of residuals
(P)CG step
CG, n = 50
CG, n = 100
CG, n = 200
PCG, n = 50
PCG, n = 100
PCG, n = 200

Fig. 76

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

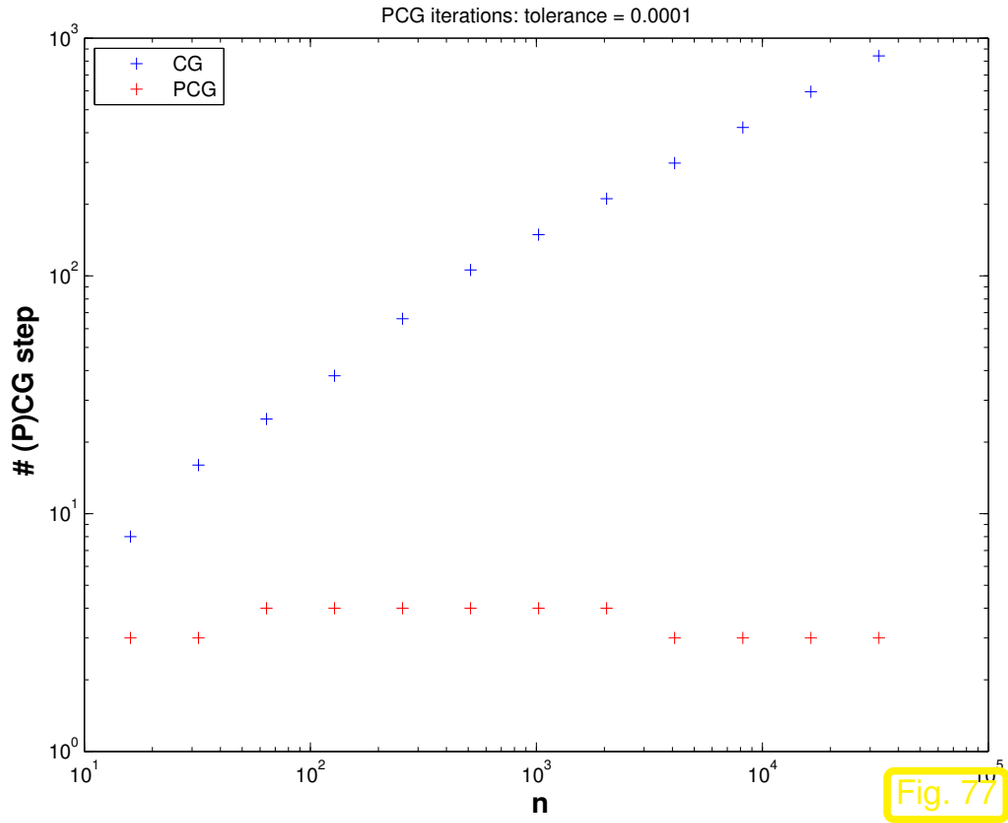


Fig. 77

Clearly in this example the tridiagonal part of the matrix is dominant for large n. In addition, its condition number grows $\sim n^2$ as is revealed by a closer inspection of the spectrum.

Preconditioning with the tridiagonal part manages to suppress this growth of the condition number of $\mathbf{B}^{-1}\mathbf{A}$ and ensures fast convergence of the preconditioned CG method.

◇

Remark 5.3.13 (Termination of PCG).

Recall Rem. 5.2.19, (5.2.20):

$$\frac{1}{\operatorname{cond}(\mathbf{A})}\frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|} \leq \frac{\left\|\mathbf{x}^{(l)}-\mathbf{x}^*\right\|}{\left\|\mathbf{x}^{(0)}-\mathbf{x}^*\right\|} \leq \operatorname{cond}(\mathbf{A})\,\frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|}\,. \tag{5.2.20}$$

$\mathbf{B}$ good preconditioner ➤ $\operatorname{cond}_2(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2})$ small (→ Notion. 5.3.3)

Idea: consider (5.2.20) for
- Euclidean norm $\|\cdot\| = \|\cdot\|_2 \quad \leftrightarrow \quad \operatorname{cond}_2$
- transformed quantities $\widetilde{\mathbf{x}}$, $\widetilde{\mathbf{r}}$, see (5.3.1), (5.3.4)

Monitor 2-norm of transformed residual:

$$\widetilde{\mathbf{r}} = \widetilde{\mathbf{b}} - \widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \mathbf{B}^{-1/2}\mathbf{r} \quad \Rightarrow \quad \|\widetilde{\mathbf{r}}\|_2^2 = \mathbf{r}^T\mathbf{B}^{-1}\mathbf{r}\,.$$

(5.2.20) ➤ estimate for 2-norm of transformed iteration errors: $\left\|\widetilde{\mathbf{e}}^{(l)}\right\|_2^2 = (\mathbf{e}^{(l)})^T\mathbf{B}\mathbf{e}^{(l)}$

Analoguous to (5.2.20), estimates for energy norm (→ Def. 5.1.1) of error $\mathbf{e}^{(l)} := \mathbf{x} - \mathbf{x}^{(l)}$, $\mathbf{x}^* := \mathbf{A}^{-1}\mathbf{b}$:

Use error equation $\mathbf{A}\mathbf{e}^{(l)} = \mathbf{r}_l$:

$$\mathbf{r}_l^T \mathbf{B}^{-1} \mathbf{r}_l = (\mathbf{B}^{-1}\mathbf{A}\mathbf{e}^{(l)})^T \mathbf{A}\mathbf{e}^{(l)} \leq \lambda_{\max}(\mathbf{B}^{-1}\mathbf{A}) \left\| \mathbf{e}^{(l)} \right\|_A^2 ,$$

$$\left\| \mathbf{e}^{(l)} \right\|_A^2 = (\mathbf{A}\mathbf{e}^{(l)})^T \mathbf{e}^{(l)} = \mathbf{r}_l^T \mathbf{A}^{-1} \mathbf{r}_l = \mathbf{B}^{-1}\mathbf{r}^T \mathbf{B}\mathbf{A}^{-1} \mathbf{r}_l \leq \lambda_{\max}(\mathbf{B}\mathbf{A}^{-1}) \, (\mathbf{B}^{-1}\mathbf{r}_l)^T \mathbf{r}_l \, .$$

available during PCG iteration (5.3.6)

$$\frac{1}{\kappa(\mathbf{B}^{-1}\mathbf{A})} \frac{\left\| \mathbf{e}^{(l)} \right\|_A^2}{\left\| \mathbf{e}^{(0)} \right\|_A^2} \leq \frac{(\mathbf{B}^{-1}\mathbf{r}_l)^T \mathbf{r}_l}{(\mathbf{B}^{-1}\mathbf{r}_0)^T \mathbf{r}_0} \leq \kappa(\mathbf{B}^{-1}\mathbf{A}) \frac{\left\| \mathbf{e}^{(l)} \right\|_A^2}{\left\| \mathbf{e}^{(0)} \right\|_A^2} \tag{5.3.14}$$

$\kappa(\mathbf{B}^{-1}\mathbf{A})$ "“small” ➤ $\mathbf{B}^{-1}$-energy norm of residual $\approx$ $\mathbf{A}$-norm of error !
($\mathbf{r}_l \cdot \mathbf{B}^{-1}\mathbf{r}_l = \mathbf{q}^T \mathbf{r}$ in Algorithm (5.3.6))

R. Hiptmair

rev 30401, November 15, 2010

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`).

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

Idea: Replace Euclidean inner product in CG with $\mathbf{A}$-inner product

$$\left\|\mathbf{x}^{(l)} - \mathbf{x}\right\|_A \quad \text{replaced with} \quad \left\|\mathbf{A}(\mathbf{x}^{(l)} - \mathbf{x})\right\|_2 = \|\mathbf{r}_l\|_2$$

► MINRES method [24, Sect. 9.5.2] (for *any* symmetric matrix **!**)

Theorem 5.4.1. *For $\mathbf{A} = \mathbf{A}^H \in \mathbb{R}^{n,n}$ the residuals $\mathbf{r}_l$ generated in the MINRES iteration satisfy*

$$\|\mathbf{r}_l\|_2 = \min\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\}$$

$$\blacktriangleright \quad \|\mathbf{r}_l\|_2 \leq \frac{2\left(1 - \frac{1}{\kappa(\mathbf{A})}\right)^l}{\left(1 + \frac{1}{\kappa(\mathbf{A})}\right)^{2l} + \left(1 - \frac{1}{\kappa(\mathbf{A})}\right)^{2l}} \|\mathbf{r}_0\|_2 \ .$$

Note: similar formula for (linear) rate of convergence as for CG, see Thm. 5.2.25, but with $\sqrt{\kappa(\mathbf{A})}$ replaced with $\kappa(\mathbf{A})$!

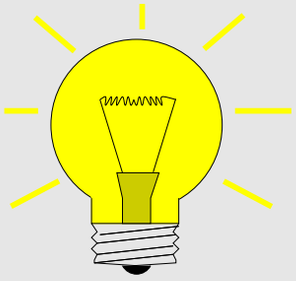
Iterative solver for $\mathbf{Ax} = \mathbf{b}$ with *symmetric* system matrix $\mathbf{A}$:

MATLAB-functions:
- `[x,flg,res,it,resv] = minres(A,b,tol,maxit,B,[],x0);`
- `[...] = minres(Afun,b,tol,maxit,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 $\mathbf{A} \in \mathbb{R}^{n,n}$:

Idea: Solver overdetermined linear system of equations



$$\mathbf{x}^{(l)} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0): \quad \mathbf{A}\mathbf{x}^{(l)} = \mathbf{b}$$

in *least squares sense*, $\rightarrow$ Chapter 7.

$$\mathbf{x}^{(l)} = \operatorname{argmin}\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\} \, .$$

➤ GMRES method for general matrices $\mathbf{A} \in \mathbb{R}^{n,n}$ $\rightarrow$ [29, Ch. 16]

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

Remark 5.4.2 (Restarted GMRES).

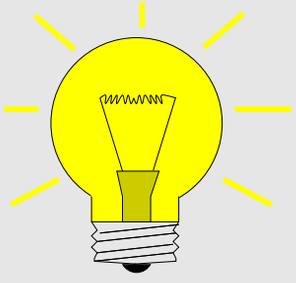
After many steps of GMRES we face considerable computational costs and memory requirements for every further step. Thus, the iteration may be *restarted* with the current iterate $\mathbf{x}^{(l)}$ as initial guess $\rightarrow$

`rs`-parameter triggers restart after every `rs` steps (Danger: failure to converge).

△

5.4.2 Iterations with short recursions

Iterative methods for *general* regular system matrix $\mathbf{A}$:



Idea: Given $\mathbf{x}^{(0)} \in \mathbb{R}^n$ determine (better) approximation $\mathbf{x}^{(l)}$ through Petrov-Galerkin condition

$$\mathbf{x}^{(l)} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0): \quad \mathbf{p}^H(\mathbf{b} - \mathbf{A}\mathbf{x}^{(l)}) = 0 \quad \forall \mathbf{p} \in W_l \ ,$$

with suitable test space W_l, $\dim W_l = l$, e.g. $W_l := \mathcal{K}_l(\mathbf{A}^H, \mathbf{r}_0)$ ($\rightarrow$ bi-conjugate gradients, BiCG)

Zoo of methods with short recursions (i.e. constant effort per step)

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).

R. Hiptmair

rev 30401, October 28, 2010

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & \cdots & & \cdots & 0 \\ 0 & 0 & 1 & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \\ & & & & & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ 0 & & & & & 0 & 1 \\ 1 & 0 & \cdots & & & \cdots & 0 \end{pmatrix} , \quad \mathbf{b} = \begin{pmatrix} 0 \\ \vdots \\ \\ \\ \\ \vdots \\ 0 \\ 1 \end{pmatrix} \quad \blacktriangleright \quad \mathbf{x} = \mathbf{e}_1 .$$

$$\mathbf{x}^{(0)} = 0 \quad \succ \quad \mathbf{r}_0 = \mathbf{e}_n \quad \succ \quad \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0) = \operatorname{Span}\{\mathbf{e}_n, \mathbf{e}_{n-1}, \ldots, \mathbf{e}_{n-l+1}\}$$

$$\blacktriangleright \quad \min\{\|\mathbf{y} - \mathbf{x}\|_2 : \mathbf{y} \in \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\} = \begin{cases} 1 & \text{, if } l \leq n \text{ ,} \\ 0 & \text{, for } l = n \text{ .} \end{cases}$$

$\diamond$

R. Hiptmair

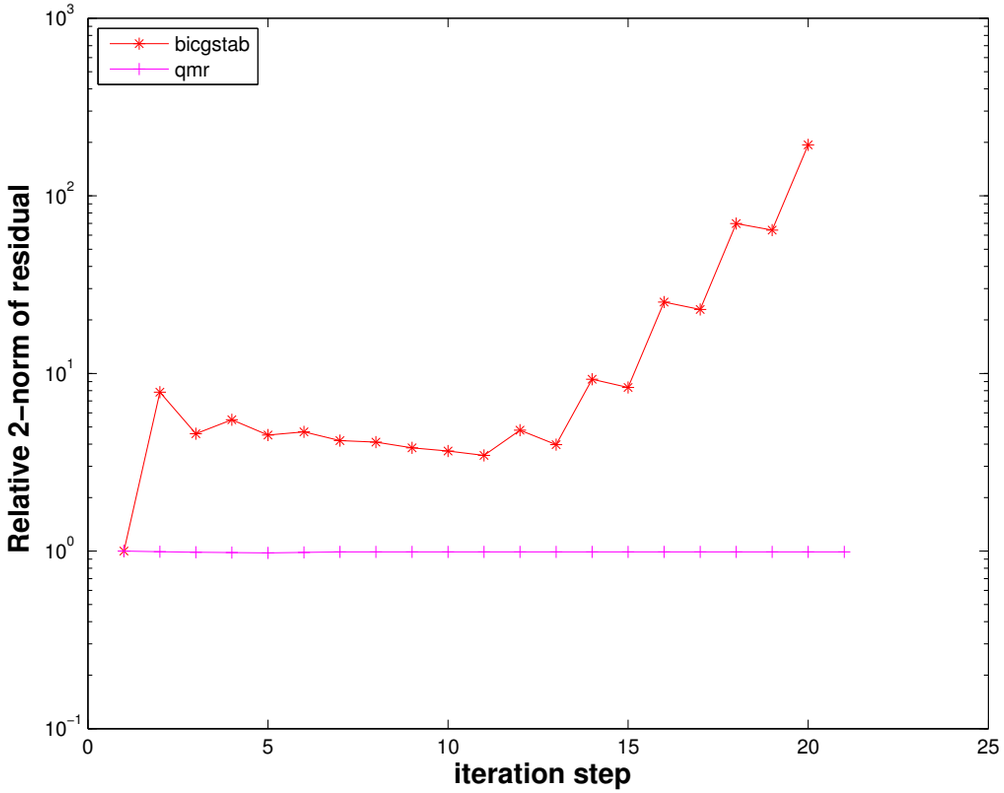
rev 30401, October 28, 2010

TRY & PRAY

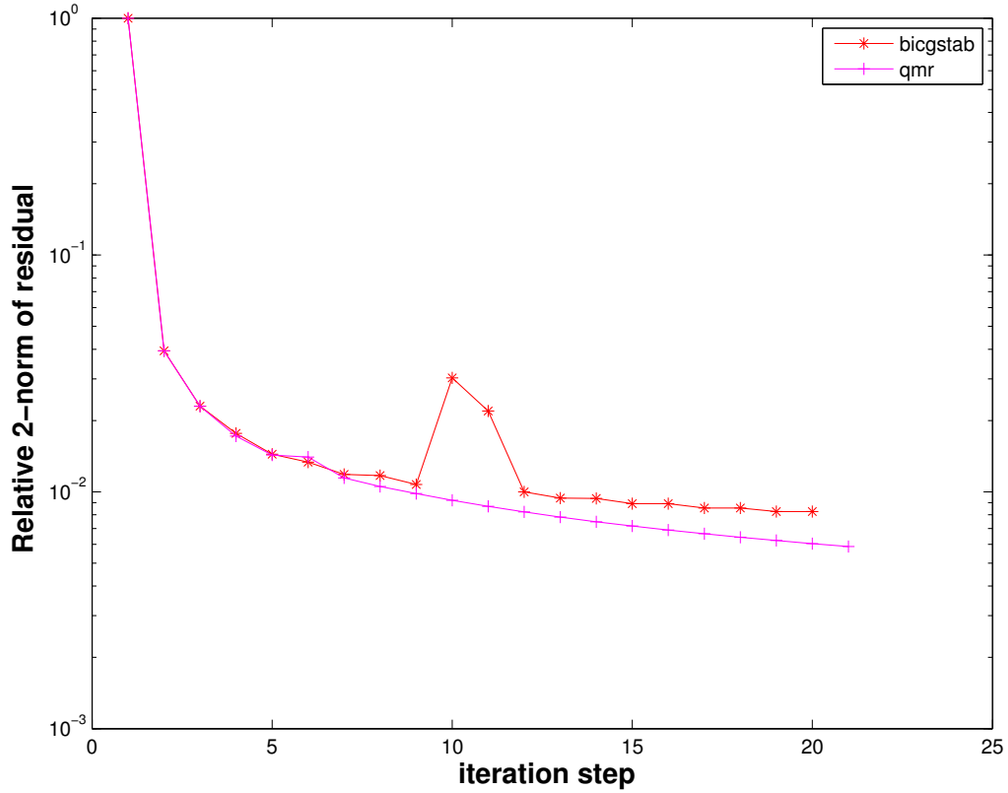
Example 5.4.4 (Convergence of Krylov subspace methods for non-symmetric system matrix).

```
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 $\mathbf{B}$ ◇

Summary:

Advantages of Krylov methods vs. direct elimination (**IF** they converge at all/sufficiently fast).

- They require system matrix $\mathbf{A}$ in procedural form `y=evalA(x)` $\leftrightarrow \mathbf{y} = \mathbf{A}\mathbf{x}$ only.
- They can perfectly exploit sparsity of system matrix.
- They can cash in on low accuracy requirements (**IF** viable termination criterion available).
- They can benefit from a good initial guess.

R. Hiptmair

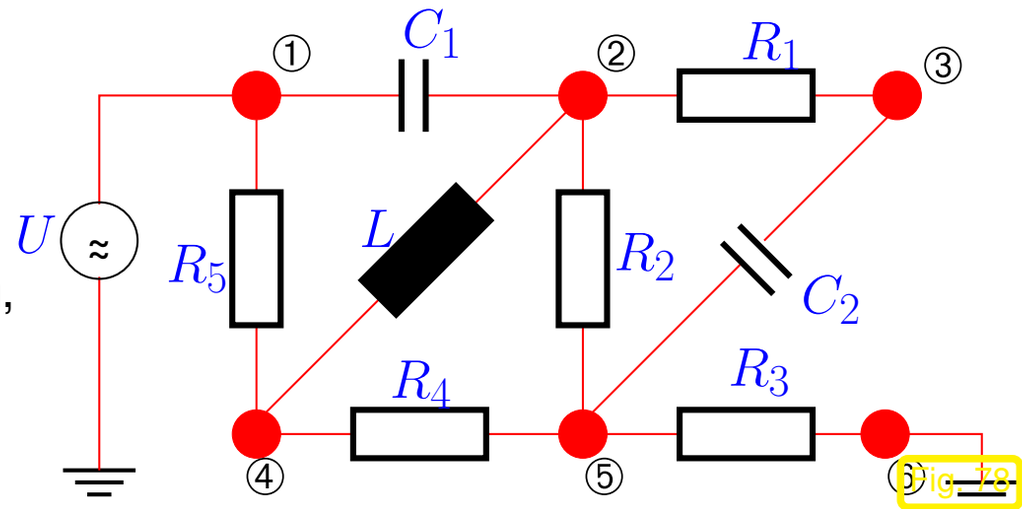
rev 30401, October 28, 2010

6 Eigenvalues

Example 6.0.1 (Resonances of linear electric circuits).

Circuit from Ex. 2.0.1 ▷

(linear components only, time-harmonic excitation, "frequency domain")



Ex. 2.0.1: nodal analysis of linear ($\leftrightarrow$ composed of resistors, inductors, capacitors) electric circuit in frequency domain (at angular frequency $\omega > 0$) , see (2.0.3))

➢ linear system of equations for nodal potentials with complex system matrix $\mathbf{A}$

R. Hiptmair

rev 30401, November 9, 2010

For circuit of Fig. 78 at angular frequency $\omega > 0$:

$$\mathbf{A} = \begin{pmatrix} i\omega C_1 + \frac{1}{R_1} - \frac{i}{\omega L} + \frac{1}{R_2} & -\frac{1}{R_1} & \frac{i}{\omega L} & -\frac{1}{R_2} \\ -\frac{1}{R_1} & \frac{1}{R_1} + i\omega C_2 & 0 & -i\omega C_2 \\ \frac{i}{\omega L} & 0 & \frac{1}{R_5} - \frac{i}{\omega L} + \frac{1}{R_4} & -\frac{1}{R_4} \\ -\frac{1}{R_2} & -i\omega C_2 & -\frac{1}{R_4} & \frac{1}{R_2} + i\omega C_2 + \frac{1}{R_4} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{R_1} + \frac{1}{R_2} & -\frac{1}{R_1} & 0 & -\frac{1}{R_2} \\ -\frac{1}{R_1} & \frac{1}{R_1} & 0 & 0 \\ 0 & 0 & \frac{1}{R_5} + \frac{1}{R_4} & -\frac{1}{R_4} \\ -\frac{1}{R_2} & 0 & -\frac{1}{R_4} & \frac{1}{R_2} + \frac{1}{R_4} \end{pmatrix} + i\omega \begin{pmatrix} C_1 & 0 & 0 & 0 \\ 0 & C_2 & 0 & -C_2 \\ 0 & 0 & 0 & 0 \\ 0 & -C_2 & 0 & C_2 \end{pmatrix} - i/\omega \begin{pmatrix} \frac{1}{L} & 0 & -\frac{1}{L} & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{L} & 0 & \frac{1}{L} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{A}(\omega) := \mathbf{W} + i\omega\mathbf{C} - i\omega^{-1}\mathbf{S} \quad , \quad \mathbf{W}, \mathbf{C}, \mathbf{S} \in \mathbb{R}^{n,n} \text{ symmetric .} \tag{6.0.2}$$

resonant frequencies = $\omega \in \{\omega \in \mathbb{R}: \mathbf{A}(\omega) \text{ singular}\}$

If the circuit is operated at a real resonant frequency, the circuit equations will not possess a solution. Of course, the real circuit will always behave in a well-defined way, but the linear model will break down due to extremely large currents and voltages. In an experiment this breakdown manifests

itself as a rather explosive meltdown of circuits components. Hence, it is vital to determine resonant frequencies of circuits in order to avoid their destruction.

➥ relevance of numerical methods for solving:

$$\text{Find} \quad \omega \in \mathbb{C} \setminus \{0\}: \quad \mathbf{W} + i\omega\mathbf{C} - i\omega^{-1}\mathbf{S} \quad \text{singular .}$$

This is a quadratic eigenvalue problem: find $\mathbf{x} \neq 0$, $\omega \in \mathbb{C} \setminus \{0\}$,

$$\mathbf{A}(\omega)\mathbf{x} = (\mathbf{W} + i\omega\mathbf{C} - i\omega^{-1}\mathbf{S})\mathbf{x} = 0 \; . \tag{6.0.3}$$

Substitution: $\mathbf{y} = -i\omega^{-1}\mathbf{x}$ [53, Sect. 3.4]:

$$(6.0.3) \quad \Leftrightarrow \quad \underbrace{\begin{pmatrix} \mathbf{W} & \mathbf{S} \\ \mathbf{I} & 0 \end{pmatrix}}_{:=\mathbf{M}} \underbrace{\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}}_{:=\mathbf{z}} = \omega \underbrace{\begin{pmatrix} -i\mathbf{C} & 0 \\ 0 & -i\mathbf{I} \end{pmatrix}}_{:=\mathbf{B}} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$$

➢ generalized linear eigenvalue problem of the form: find $\omega \in \mathbb{C}$, $\mathbf{z} \in \mathbb{C}^{2n} \setminus \{0\}$ such that

$$\mathbf{M}\mathbf{z} = \omega\mathbf{B}\mathbf{z} \; . \tag{6.0.4}$$

In this example one is mainly interested in the eigenvalues ω, whereas the eigenvectors $\mathbf{z}$ usually need not be computed.

R. Hiptmair

rev 30401, November 9, 2010

◇

Example 6.0.5 (Analytic solution of homogeneous linear ordinary differential equations). → [52, Remark 5.6.1], [23, Sect. 10.1], [39, Sect. 8.1], [10, Ex. 7.3]

Autonomous homogeneous linear ordinary differential equation (ODE):

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \quad , \quad \mathbf{A} \in \mathbb{C}^{n,n} \, . \tag{6.0.6}$$

$$\mathbf{A} = \mathbf{S} \underbrace{\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}}_{=:\mathbf{D}} \mathbf{S}^{-1} \, , \quad \mathbf{S} \in \mathbb{C}^{n,n} \text{ regular} \quad \Longrightarrow \quad \left(\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \quad \overset{\mathbf{z}=\mathbf{S}^{-1}\mathbf{y}}{\longleftrightarrow} \quad \dot{\mathbf{z}} = \mathbf{D}\mathbf{z} \right) .$$

➢ solution of initial value problem:

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \, , \quad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{C}^n \quad \Rightarrow \quad \mathbf{y}(t) = \mathbf{S}\mathbf{z}(t) \, , \quad \dot{\mathbf{z}} = \mathbf{D}\mathbf{z} \, , \quad \mathbf{z}(0) = \mathbf{S}^{-1}\mathbf{y}_0 \, .$$

The initial value problem for the *decoupled* homogeneous linear ODE $\dot{\mathbf{z}} = \mathbf{D}\mathbf{z}$ has a simple analytic solution

$$\mathbf{z}_i(t) = \exp(\lambda_i t)(\mathbf{z}_0)_i = \exp(\lambda_i t) \left((\mathbf{S}^{-1})^T_{i,:} \mathbf{y}_0 \right) \, .$$

R. Hiptmair

rev 30401, November 9, 2010

In light of Rem. 1.2.2:

$$\mathbf{A} = \mathbf{S} \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \mathbf{S}^{-1} \quad \Leftrightarrow \quad \mathbf{A}\left((\mathbf{S})_{:,i}\right) = \lambda_i \left((\mathbf{S})_{:,i}\right) \quad i = 1, \ldots, n \,. \tag{6.0.7}$$

In order to find the transformation matrix $\mathbf{S}$ all non-zero solution vectors (= eigenvectors) $\mathbf{x} \in \mathbb{C}^n$ of the linear eigenvalue problem

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$$

have to be found.

◇

6.1 Theory of eigenvalue problems [39, Ch. 7], [23, Ch. 9]

Definition 6.1.1 (Eigenvalues and eigenvectors). → *[39, Sects. 7.1,7.2], [23, Sect. 9.1]*

- $\lambda \in \mathbb{C}$ *eigenvalue* (ger.: *Eigenwert*) *of* $\mathbf{A} \in \mathbb{K}^{n,n}$ $:\Leftrightarrow$ $\underbrace{\det(\lambda\mathbf{I} - \mathbf{A})}_{\text{characteristic polynomial } \chi(\lambda)} = 0$
- *spectrum of* $\mathbf{A} \in \mathbb{K}^{n,n}$: $\sigma(\mathbf{A}) := \{\lambda \in \mathbb{C}: \lambda \text{ eigenvalue of } \mathbf{A}\}$
- *eigenspace* (ger.: *Eigenraum*) *associated with eigenvalue* $\lambda \in \sigma(\mathbf{A})$:
$$\mathrm{Eig}_{\mathbf{A}}(\lambda) := \mathrm{Ker}(\lambda\mathbf{I} - \mathbf{A})$$
- $\mathbf{x} \in \mathrm{Eig}_{\mathbf{A}}(\lambda) \setminus \{0\} \quad \Rightarrow \quad \mathbf{x}$ *is eigenvector*
- *Geometric multiplicity* (ger.: *Vielfachheit*) *of an eigenvalue* $\lambda \in \sigma(\mathbf{A})$:
$$m(\lambda) := \dim \mathrm{Eig}_{\mathbf{A}}(\lambda)$$

Two simple facts:

$$\lambda \in \sigma(\mathbf{A}) \quad \Rightarrow \quad \dim \mathrm{Eig}_{\mathbf{A}}(\lambda) > 0\,, \tag{6.1.2}$$

$$\det(\mathbf{A}) = \det(\mathbf{A}^T) \quad \forall \mathbf{A} \in \mathbb{K}^{n,n} \quad \Rightarrow \quad \sigma(\mathbf{A}) = \sigma(\mathbf{A}^T)\,. \tag{6.1.3}$$

✎ notation: $\rho(\mathbf{A}) := \max\{|\lambda|: \lambda \in \sigma(\mathbf{A})\} \hat{=}$ spectral radius of $\mathbf{A} \in \mathbb{K}^{n,n}$

Theorem 6.1.4 (Bound for spectral radius)**.**
For any matrix norm $\|\cdot\|$ induced by a vector norm (→ Def. 2.5.5)

$$\rho(\mathbf{A}) \leq \|\mathbf{A}\| \ .$$

Proof. Let $\mathbf{z} \in \mathbb{C}^n \setminus \{0\}$ be an eigenvector to the largest (in modulus) eigenvalue λ of $\mathbf{A} \in \mathbb{C}^{n,n}$. Then

$$\|\mathbf{A}\| := \sup_{\mathbf{x}\in\mathbb{C}^{n,n}\setminus\{0\}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} \geq \frac{\|\mathbf{A}\mathbf{z}\|}{\|\mathbf{z}\|} = |\lambda| = \rho(\mathbf{A}) \ .$$

Lemma 6.1.5 (Gershgorin circle theorem)**.** → *[10, Thm. 7.13], [29, Thm. 32.1] For any $\mathbf{A} \in \mathbb{K}^{n,n}$ holds true*

$$\sigma(\mathbf{A}) \subset \bigcup_{j=1}^{n} \left\{ z \in \mathbb{C}\colon |z - a_{jj}| \leq \sum\nolimits_{i\neq j} |a_{ji}| \right\} \ .$$

Lemma 6.1.6 (Similarity and spectrum). → *[23, Thm. 9.7], [10, Lemma 7.6], [39, Thm. 7.2]*

The spectrum of a matrix is invariant with respect to similarity transformations:

$$\forall \mathbf{A} \in \mathbb{K}^{n,n}: \quad \sigma(\mathbf{S}^{-1}\mathbf{A}\mathbf{S}) = \sigma(\mathbf{A}) \quad \forall \text{ regular } \mathbf{S} \in \mathbb{K}^{n,n} \ .$$

Lemma 6.1.7. *Existence of a one-dimensional invariant subspace*

$$\forall \mathbf{C} \in \mathbb{C}^{n,n}: \quad \exists \mathbf{u} \in \mathbb{C}^{n}: \quad \mathbf{C}\left(\operatorname{Span}\{\mathbf{u}\}\right) \subset \operatorname{Span}\{\mathbf{u}\} \ .$$

Theorem 6.1.8 (Schur normal form). $\rightarrow$ *[25, Thm .2.8.1]*

$\forall \mathbf{A} \in \mathbb{K}^{n,n}$: $\exists \mathbf{U} \in \mathbb{C}^{n,n}$ *unitary*: $\mathbf{U}^H \mathbf{A} \mathbf{U} = \mathbf{T}$ *with* $\mathbf{T} \in \mathbb{C}^{n,n}$ *upper triangular* .

Corollary 6.1.9 (Principal axis transformation).

$\mathbf{A} \in \mathbb{K}^{n,n}$, $\mathbf{A}\mathbf{A}^H = \mathbf{A}^H\mathbf{A}$: $\exists \mathbf{U} \in \mathbb{C}^{n,n}$ *unitary*: $\mathbf{U}^H \mathbf{A} \mathbf{U} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$, $\lambda_i \in \mathbb{C}$.

A matrix $\mathbf{A} \in \mathbb{K}^{n,n}$ with $\mathbf{A}\mathbf{A}^H = \mathbf{A}^H\mathbf{A}$ is called normal.

Examples of normal matrices are

- Hermitian matrices: $\mathbf{A}^H = \mathbf{A}$ ➤ $\sigma(\mathbf{A}) \subset \mathbb{R}$
- unitary matrices: $\mathbf{A}^H = \mathbf{A}^{-1}$ ➤ $|\sigma(\mathbf{A})| = 1$
- skew-Hermitian matrices: $\mathbf{A} = -\mathbf{A}^H$ ➤ $\sigma(\mathbf{A}) \subset i\mathbb{R}$

R. Hiptmair

rev 30401, November 9, 2010

➤

Normal matrices can be diagonalized by *unitary* similarity transformations

Symmetric real matrices can be diagonalized by *orthogonal* similarity transformations

In Thm. 6.1.9:
- $\lambda_1, \ldots, \lambda_n$ = eigenvalues of $\mathbf{A}$
- Columns of $\mathbf{U}$ = orthonormal basis of eigenvectors of $\mathbf{A}$

Eigenvalue problems: (EVPs)

❶ Given $\mathbf{A} \in \mathbb{K}^{n,n}$ find **all** eigenvalues (= spectrum of $\mathbf{A}$).

❷ Given $\mathbf{A} \in \mathbb{K}^{n,n}$ find $\sigma(\mathbf{A})$ plus **all** eigenvectors.

❸ Given $\mathbf{A} \in \mathbb{K}^{n,n}$ find **a few** eigenvalues and associated eigenvectors

R. Hiptmair

rev 30401, November 9, 2010

(Linear) generalized eigenvalue problem:

Given $\mathbf{A} \in \mathbb{C}^{n,n}$, regular $\mathbf{B} \in \mathbb{C}^{n,n}$, seek $\mathbf{x} \neq 0$, $\lambda \in \mathbb{C}$

$$\mathbf{Ax} = \lambda \mathbf{Bx} \quad \Leftrightarrow \quad \mathbf{B}^{-1}\mathbf{Ax} = \lambda \mathbf{x} \; . \tag{6.1.10}$$

$\mathbf{x} \mathrel{\hat{=}}$ generalized eigenvector, $\lambda \mathrel{\hat{=}}$ generalized eigenvalue

Obviously every generalized eigenvalue problem is equivalent to a standard eigenvalue problem

$$\mathbf{Ax} = \lambda \mathbf{Bx} \quad \Leftrightarrow \quad \mathbf{B}^{-1}\mathbf{A} = \lambda \mathbf{x} \; .$$

However, usually it is not advisable to use this equivalence for numerical purposes!

Remark 6.1.11 (Generalized eigenvalue problems and Cholesky factorization).

If $\mathbf{B} = \mathbf{B}^H$ s.p.d. ($\rightarrow$ Def. 2.7.9) with Cholesky factorization $\mathbf{B} = \mathbf{R}^H\mathbf{R}$

$$\mathbf{Ax} = \lambda \mathbf{Bx} \quad \Leftrightarrow \quad \widetilde{\mathbf{A}}\mathbf{y} = \lambda \mathbf{y} \quad \text{where } \widetilde{\mathbf{A}} := \mathbf{R}^{-H}\mathbf{A}\mathbf{R}^{-1}, \; \mathbf{y} := \mathbf{Rx} \; .$$

→ This transformation can be used for efficient computations.

△

6.2 “Direct” Eigensolvers

Purpose: solution of eigenvalue problems ❶, ❷ for dense matrices “up to machine precision”

MATLAB-function: `eig`

`d = eig(A)` : computes spectrum $\sigma(\mathbf{A}) = \{d_1, \ldots, d_n\}$ 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 $\mathbf{AV} = \mathbf{VD}$

Remark 6.2.1 (QR-Algorithm). → [20, Sect. 7.5], [39, Sect. 10.3], [29, Ch. 26]

R. Hiptmair

rev 30401, November 8, 2010

Note: All “direct” eigensolvers are iterative methods

Idea: Iteration based on successive **unitary** similarity transformations

$$\mathbf{A} = \mathbf{A}^{(0)} \longrightarrow \mathbf{A}^{(1)} \longrightarrow \ldots \longrightarrow \begin{cases} \text{diagonal matrix} & \text{, if } \mathbf{A} = \mathbf{A}^H \text{ ,} \\ \text{upper triangular matrix} & \text{, else.} \\ (\rightarrow \text{Thm. 6.1.8}) & \end{cases}$$

(superior stability of unitary transformations, see Rem. 2.8.1)

QR-algorithm (with shift)

- in general: quadratic convergence
- cubic convergence for normal matrices

(→ [20, Sect. 7.5,8.2])

Code 6.2.2: QR-algorithm with shift

```
function d = eigqr(A,tol)
n = size(A,1);
while (norm(tril(A,-1)) > tol*norm(A))
% shift by eigenvalue of lower right 2×2 block
  closest to (A)_{n,n}
  sc = eig(A(n-1:n,n-1:n));
  [dummy,si] = min(abs(sc-A(n,n)));
  shift = sc(si);
  [Q,R] = qr( A - shift * eye(n));
  A = Q'*A*Q;
end
d = diag(A);
```

Computational cost: $O(n^3)$ operations per step of the QR-algorithm

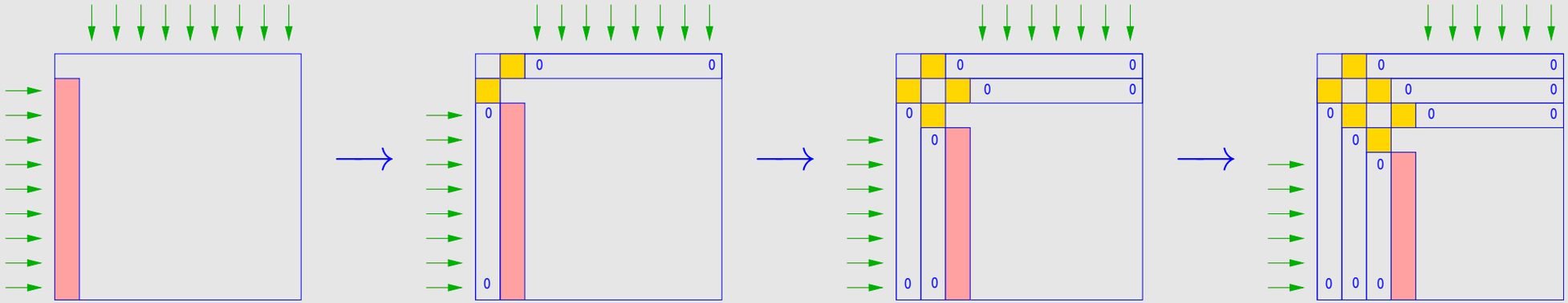
Library implementations of the QR-algorithm provide *numerically stable* eigensolvers (→ Def.2.5.11)

R. Hiptmair

rev 30401, November 8, 2010

Remark 6.2.3 (Unitary similarity transformation to tridiagonal form).

Successive Householder similarity transformations of $\mathbf{A} = \mathbf{A}^H$:
(→ $\hat{=}$ affected rows/columns,  $\hat{=}$ targeted vector)



transformation to tridiagonal form **!** (for general matrices a similar strategy can achieve a similarity transformation to upper Hessenberg form)

this transformation is used as a preprocessing step for QR-algorithm ➢ `eig`.

△

Similar functionality for generalized EVP $\mathbf{Ax} = \lambda\mathbf{Bx}$, $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{n,n}$

R. Hiptmair

rev 30401, November 8, 2010

`d = eig(A,B)` : computes all generalized eigenvalues

`[V,D] = eig(A,B)` : computes $\mathbf{V} \in \mathbb{C}^{n,n}$, *diagonal* $\mathbf{D} \in \mathbb{C}^{n,n}$ such that $\mathbf{AV} = \mathbf{BVD}$

Note: (Generalized) eigenvectors can be recovered as columns of $\mathbf{V}$:

$$\mathbf{AV} = \mathbf{VD} \quad \Leftrightarrow \quad \mathbf{A}(\mathbf{V})_{:,i} = (\mathbf{D})_{i,i}\mathbf{V}_{:,i} \;,$$

if $\mathbf{D} = \operatorname{diag}(d_1, \ldots, d_n)$.

Remark 6.2.4 (Computational effort for eigenvalue computations).

Computational effort (#elementary operations) for `eig()`:

eigenvalues & eigenvectors of $\mathbf{A} \in \mathbb{K}^{n,n}$	$\sim 25n^3 + O(n^2)$
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 $\mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n}$	$\sim 30n^2 + O(n)$

$\Big\}$ $O(n^3)$!

Note: `eig` not available for sparse matrix arguments

Exception: `d=eig(A)` for sparse *Hermitian* matrices

Example 6.2.5 (Runtimes of `eig`).

Code 6.2.6: measuring runtimes of `eig`

```
function eigtiming

A = rand(500,500); B = A'*A;
C = gallery('tridiag',500,1,3,1);
times = [];
for n=5:5:500
  An = A(1:n,1:n); Bn = B(1:n,1:n); Cn = C(1:n,1:n);
  t1 = 1000; for k=1:3, tic; d = eig(An); t1 = min(t1,toc); end
  t2 = 1000; for k=1:3, tic; [V,D] = eig(An); t2 = min(t2,toc);
    end
  t3 = 1000; for k=1:3, tic; d = eig(Bn); t3 = min(t3,toc); end
  t4 = 1000; for k=1:3, tic; [V,D] = eig(Bn); t4 = min(t4,toc);
    end
  t5 = 1000; for k=1:3, tic; d = eig(Cn); t5 = min(t5,toc); end
  times = [times; n t1 t2 t3 t4 t5];
end
```

```
15
16 figure;
17 loglog(times(:,1),times(:,2),'r+', times(:,1),times(:,3),'m*',...
18       times(:,1),times(:,4),'cp', times(:,1),times(:,5),'b^',...
19       times(:,1),times(:,6),'k.');
20 xlabel('{\bf matrix size n}','fontsize',14);
21 ylabel('{\bf time [s]}','fontsize',14);
22 title('eig runtimes');
23 legend('d = eig(A)','[V,D] = eig(A)','d = eig(B)','[V,D] =
   eig(B)','d = eig(C)',...
24       'location','northwest');
25
26 print -depsc2 '../PICTURES/eigtimingall.eps'
27
28 figure;
29 loglog(times(:,1),times(:,2),'r+', times(:,1),times(:,3),'m*',...
30      times(:,1),(times(:,1).^3)/(times(1,1)^3)*times(1,2),'k-');
31 xlabel('{\bf matrix size n}','fontsize',14);
32 ylabel('{\bf time [s]}','fontsize',14);
33 title('nxn random matrix');
34 legend('d = eig(A)','[V,D] =
   eig(A)','O(n^3)','location','northwest');
35
```


R. Hiptmair

rev 30401, November 8, 2010

```
print -depsc2 '../PICTURES/eigtimingA.eps'

figure;
loglog(times(:,1),times(:,4),'r+', times(:,1),times(:,5),'m*',...
      times(:,1),(times(:,1).^3)/(times(1,1)^3)*times(1,2),'k-');
xlabel('{\bf matrix size n}','fontsize',14);
ylabel('{\bf time [s]}','fontsize',14);
title('nxn random Hermitian matrix');
legend('d = eig(A)','[V,D] =
  eig(A)','O(n^3)','location','northwest');

print -depsc2 '../PICTURES/eigtimingB.eps'

figure;
loglog(times(:,1),times(:,6),'r*',...
      times(:,1),(times(:,1).^2)/(times(1,1)^2)*times(1,2),'k-');
xlabel('{\bf matrix size n}','fontsize',14);
ylabel('{\bf time [s]}','fontsize',14);
title('nxn tridiagonel Hermitian matrix');
legend('d = eig(A)','O(n^2)','location','northwest');

print -depsc2 '../PICTURES/eigtimingC.eps'
```

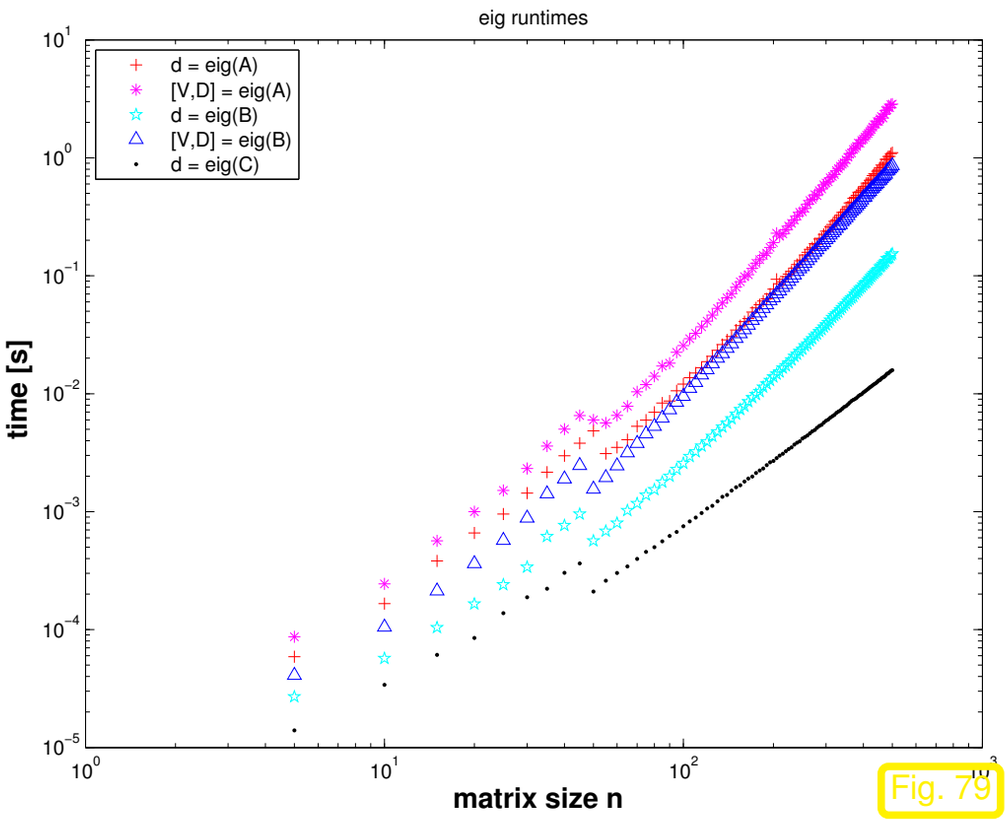


Fig. 79

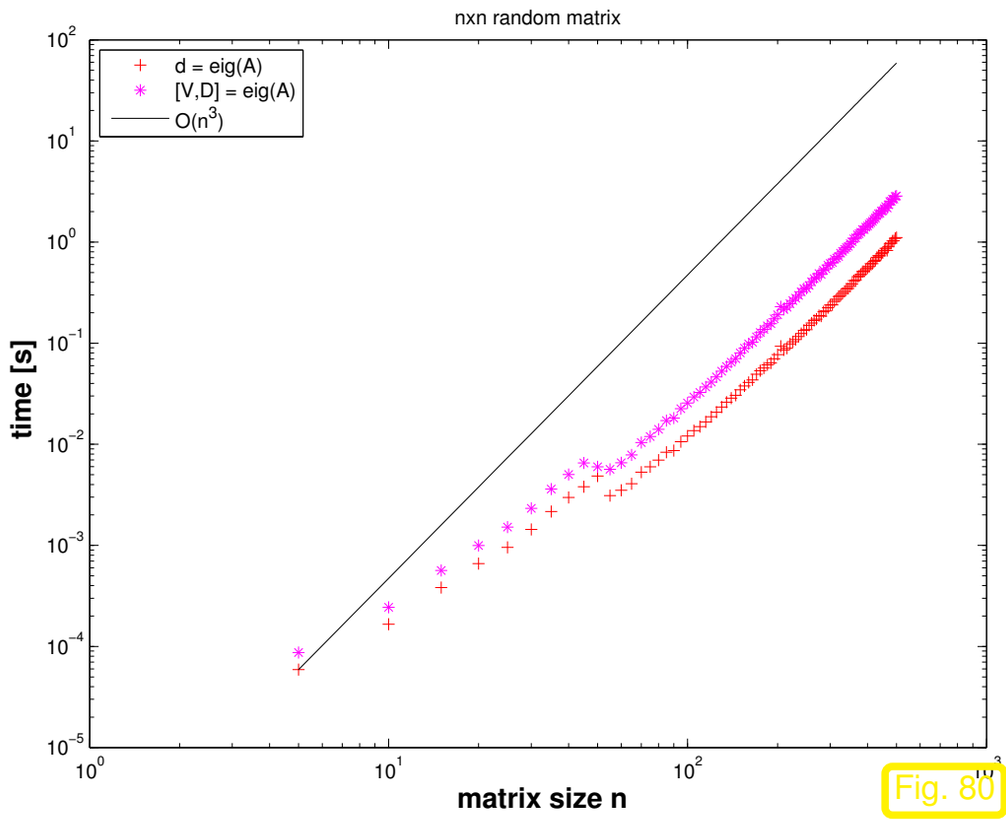


Fig. 80

R. Hiptmair

rev 30401, November 8, 2010

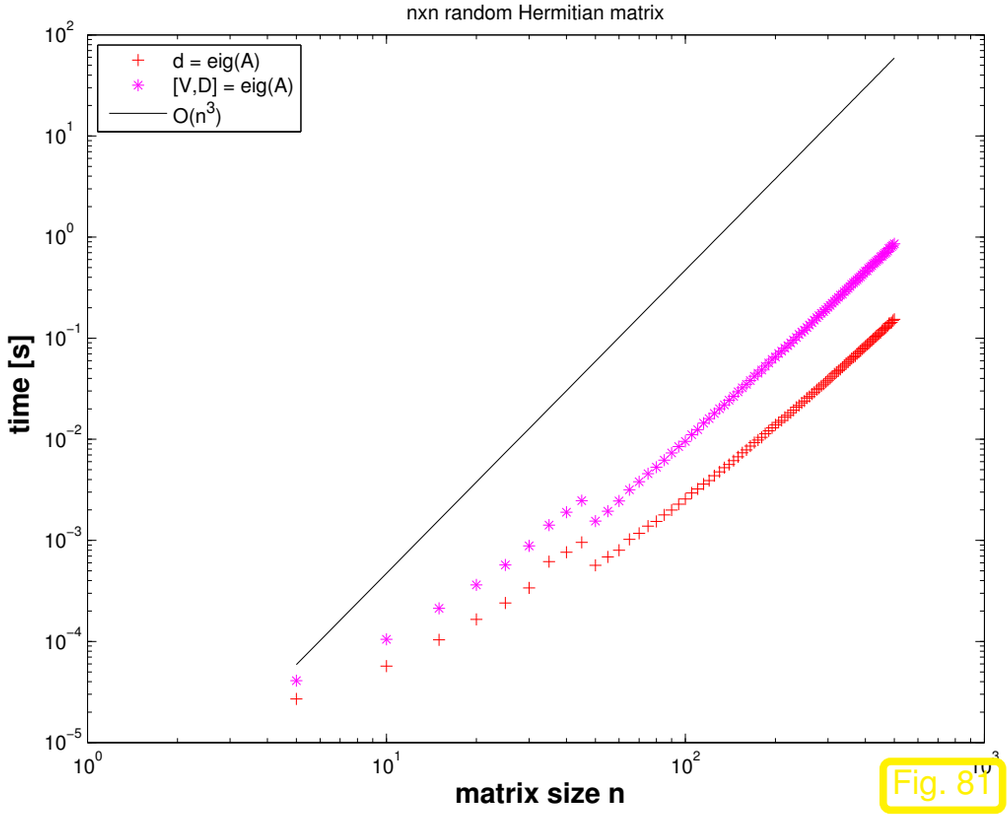


Fig. 81

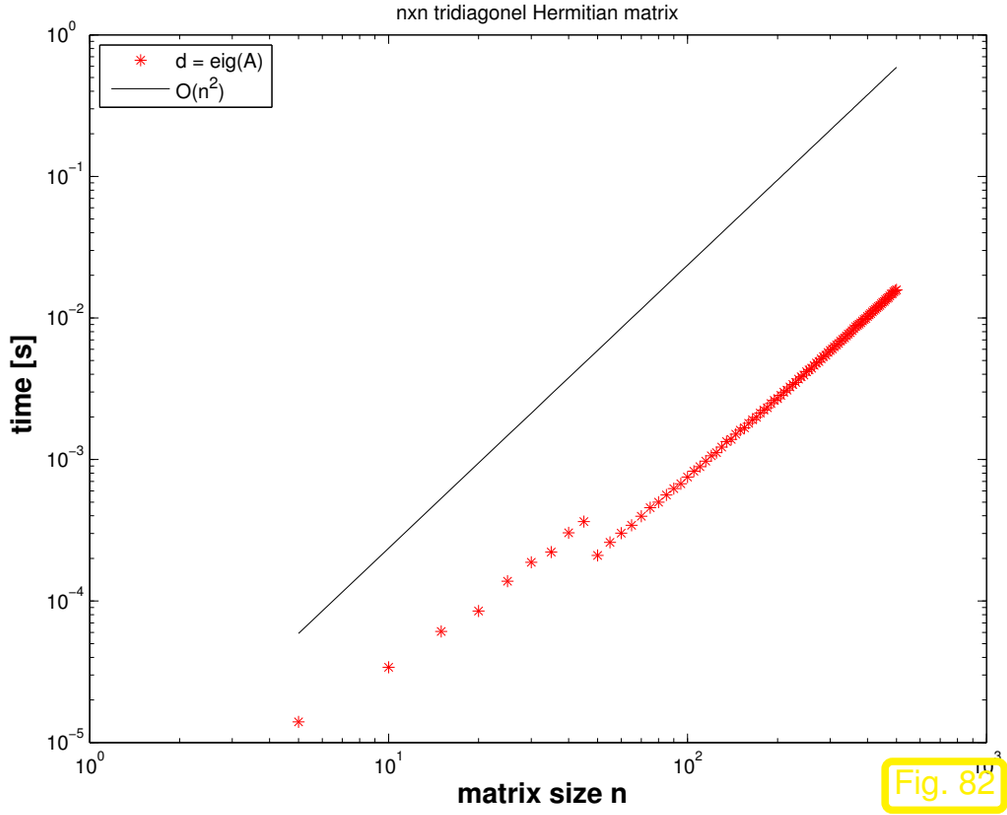


Fig. 82

R. Hiptmair

rev 30401, November 22, 2010

☛ For the sake of efficiency: think which information you really need when computing eigenvalues/eigenvectors of dense matrices

Potentially more efficient methods for *sparse matrices* will be introduced below in Sects. 6.3, 6.4.

6.3 Power Methods

6.3.1 Direct power method [10, Sect. 7.5]

Example 6.3.1 (Page rank algorithm). $\rightarrow$ [34]

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 $1 - d$.

Stationary Markov chain, state space $\hat{=}$ set of all web pages

Question: Fraction of time spent by random surfer on i-th page (= page rank $x_i \in [0, 1]$)

This number $\in]0,1[$ can be used to gauge the "importance" of a web page, which, in turns, offers a way to sort the hits resulting from a keyword query: the GOOGLE idea.

Method: Stochastic simulation ▷

Code 6.3.2: stochastic page rank simulation

```
function prstochsim(Nhops)
% Load web graph data stored in N × N-matrix G
load harvard500.mat;
N = size(G,1); d = 0.15;
count = zeros(1,N); cp = 1;
figure('position',[0 0 1600 1200]); 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;
```

R. Hiptmair

rev 30401, November 22, 2010

```
18    plot(1:N,count/n,'r.'); axis([0 N+1 0 0.1]);
19    xlabel('{\bf harvard500: no. of page}','fontsize',14);
20    ylabel('{\bf page rank}','fontsize',14);
21    title(sprintf('{\\bf page rank, harvard500: %d
        hops}',n),'fontsize',14);
22    drawnow;
23  end
```


Explanations for Code 6.3.1:

- Line 9: `rand` generates uniformly distributed pseudo-random numbers $\in [0,1[$
- Web graph encoded in $\mathbf{G} \in \{0,1\}^{N,N}$:

$$(\mathbf{G})_{ij} = 1 \quad \Rightarrow \quad \text{link } j \to i \ ,$$

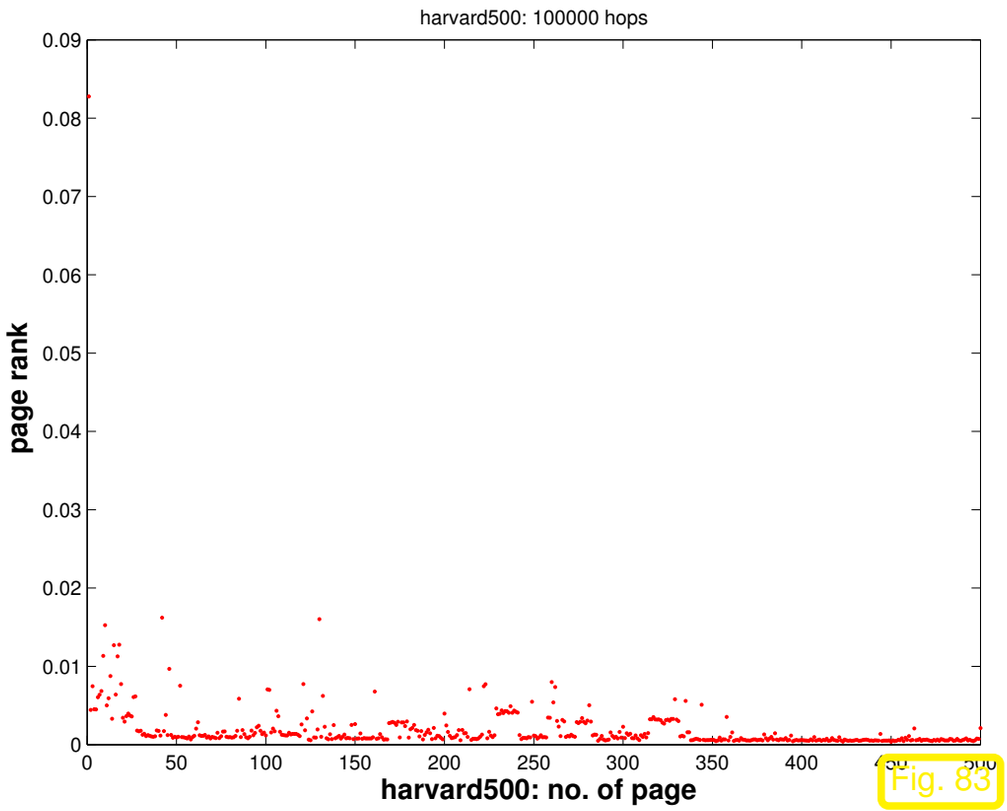


Fig. 83

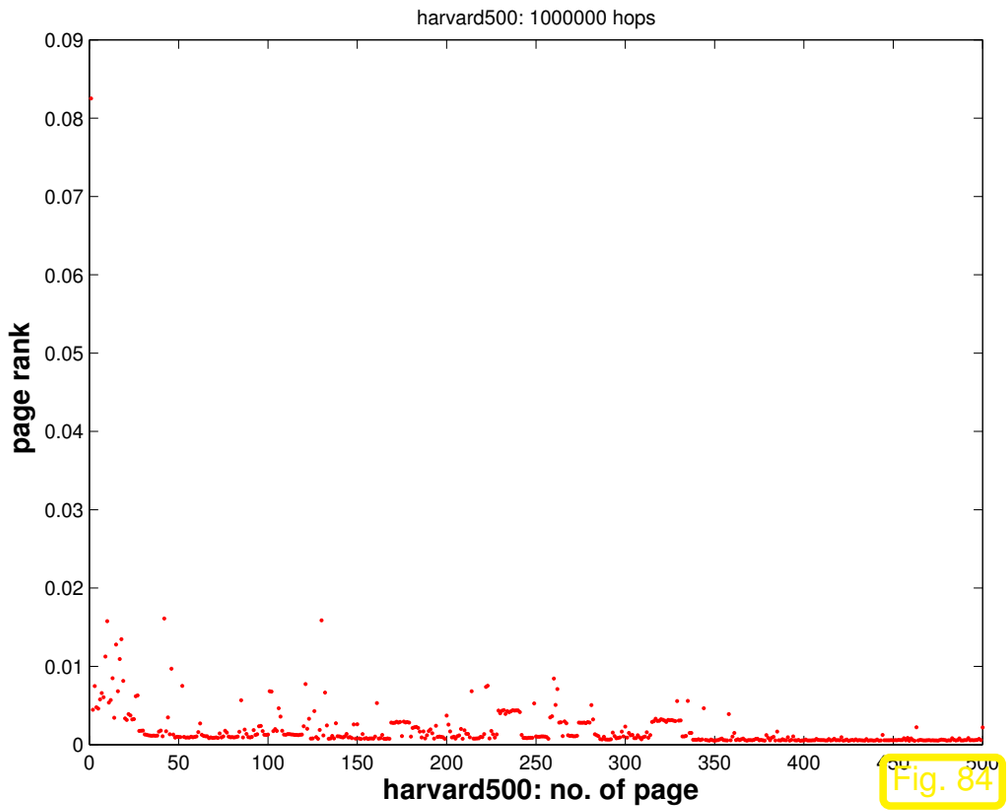


Fig. 84

R. Hiptmair

rev 30401, November 22, 2010

Observation: relative visit times stabilize as the number of hops in the stochastic simulation $\to \infty$.

The limit distribution is called stationary distribution/invariant measure of the Markov chain. This is what we seek.

- Numbering of pages $1, \ldots, N$, $\ell_i \mathrel{\hat{=}}$ number of links from page i

- $N \times N$-matrix of transition probabilities page $j \to$ page i: $\mathbf{A} = (a_{ij})_{i,j=1}^{N} \in \mathbb{R}^{N,N}$

$$a_{ij} \in [0,1] \quad \hat{=} \text{ probability to jump from page } j \text{ to page } i.$$

$$\Rightarrow \quad \sum_{i=1}^{N} a_{ij} = 1 \; . \tag{6.3.3}$$

A matrix $\mathbf{A} \in [0,1]^{N,N}$ with the property (6.3.3) is called a (column) stochastic matrix.

"Meaning" of $\mathbf{A}$: given $\mathbf{x} \in [0,1]^N$, $\|\mathbf{x}\|_1 = 1$, where x_i is the probability of the surfer to visit page i, $i = 1, \ldots, N$, at an instance t in time, $\mathbf{y} = \mathbf{A}\mathbf{x}$ satisfies

$$y_j \geq 0 \quad , \quad \sum_{j=1}^{N} y_j = \sum_{j=1}^{N} \sum_{i=1}^{N} a_{ji} x_i = \sum_{i=1}^{N} x_i \underbrace{\sum_{j=1}^{N} a_{ij}}_{=1} = \sum_{i=1}^{N} x_i = 1 \; .$$

$\blacktriangleright$ $y_j \hat{=}$ probability for visiting page j at time $t + \Delta t$.

R. Hiptmair

rev 30401, November 22, 2010

Transition probability matrix for page rank computation

$$(\mathbf{A})_{ij} = \begin{cases} \frac{1}{N} & \text{, if } (\mathbf{G})_{ij} = 0 \; \forall i = 1, \ldots, N \text{ ,} \\ d/N + (1-d)(\mathbf{G})_{ij}/\ell_j & \text{else.} \end{cases} \tag{6.3.4}$$

Code 6.3.5: transition probability matrix for page rank

```
1 function A = prbuildA(G,d)
2 N = size(G,1);
3 l = full(sum(G)); idx = find(l>0);
4 s = zeros(N,1); s(idx) = 1./l(idx);
5 ds = ones(N,1)/N; ds(idx) = d/N;
6 A = ones(N,1)*ones(1,N)*diag(ds) +
    (1-d)*G*diag(s);
```

Note: special treatment of zero columns!

Stochastic simulation based on a single surfer is *slow*. Alternatives?

Thought experiment: Instead of a single random surfer we may consider $m \in \mathbb{N}$, $m \gg 1$, of them who visit pages independently. The fraction of time $m \cdot T$ they all together spend on page i will obviously be the same for $T \to \infty$ as that for a single random surfer.

Instead of counting the surfers we watch the proportions of them visiting particular web pages at an instance of time. Thus, after the k-th hop we can assign a number $x_i^{(k)} \in [0,1]$ to web page i, which gives the proportion of surfers currently on that page: $x_i^{(k)} := \frac{n_i^{(k)}}{m}$, where $n_i^{(k)} \in \mathbb{N}_0$ designates the number of surfers on page i after the k-th hop.

Now consider $m \to \infty$. The law of law of large numbers suggests that the ("infinitely many") surfers visiting page j will move on to other pages proportional to the transistion probabilities a_{ij}: in terms of proportions, for $m \to \infty$ the stochastic evolution becomes a deterministic discrete dynamical system and we find

$$x_i^{(k+1)} = \sum_{j=1}^{N} a_{ij} x_j^{(k)} \;, \tag{6.3.6}$$

that is, the proportion of surfers ending up on page i equals the sum of the proportions on the "source pages" weighted with the transition probabilities.

Notice that (6.3.6) amounts to matrix×vector. Thus, writing $\mathbf{x}^{(0)} \in [0,1]^N$, $\left\|\mathbf{x}^{(0)}\right\| = 1$ for the initial distribution of the surfers on the net we find

$$\mathbf{x}^{(k)} = \mathbf{A}^k \mathbf{x}^{(0)}$$

R. Hiptmair

rev 30401, November 22, 2010

will be their mass distribution after k hops. If the limit exists, the i-th component of $\mathbf{x}^* := \lim_{k\to\infty} \mathbf{x}^{(k)}$ tells us which fraction of the (infinitely many) surfers will be visiting page i most of the time. Thus, $\mathbf{x}^*$ yields the stationary distribution of the Markov chain.

Code 6.3.7: tracking fractions of many surfers

```
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 1600 1200]);
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
```


R. Hiptmair

rev 30401, November 22, 2010

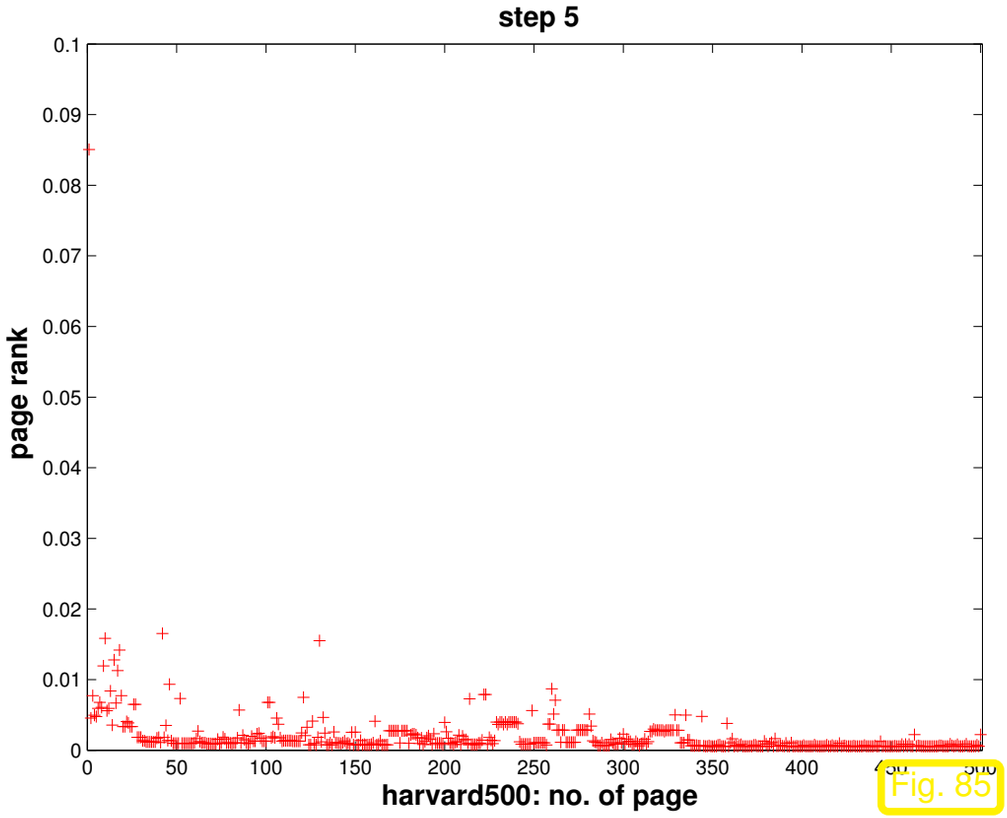


Fig. 85

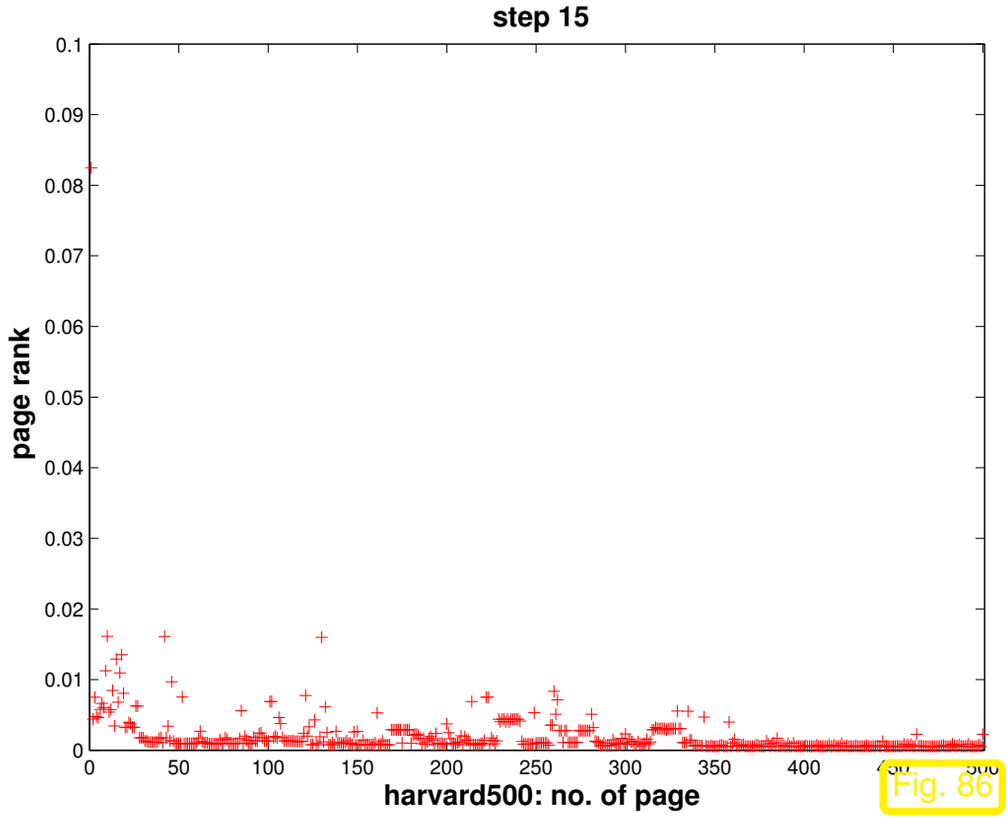


Fig. 86

Comparison:

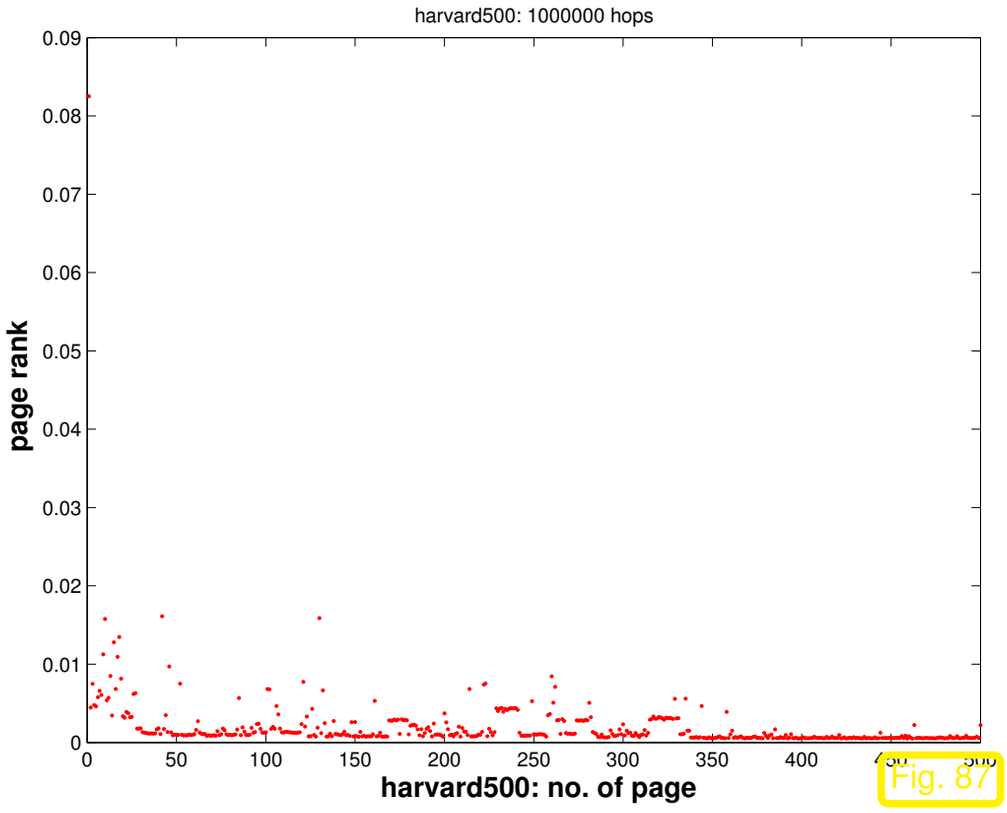


Fig. 87

Single surfer stochastic simulation

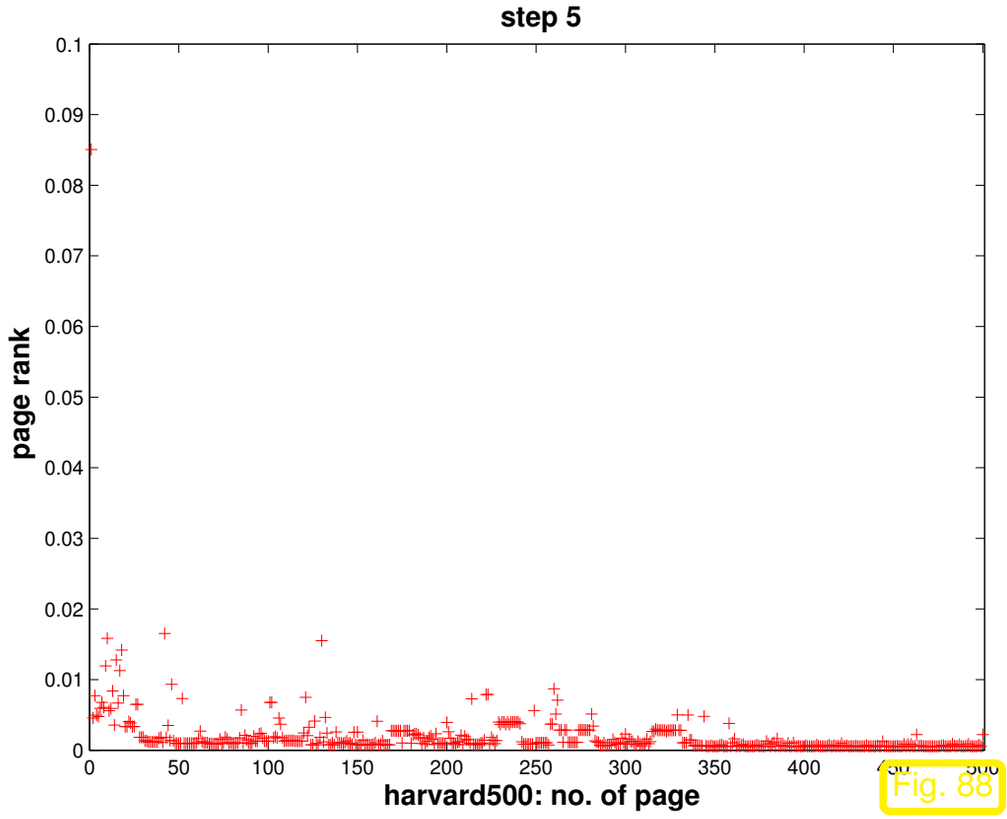


Fig. 88

Power method, Code 6.3.6

Observation: Convergence of the $\mathbf{x}^{(k)} \to \mathbf{x}^*$, and the limit must be a fixed point of the iteration function:

$$\mathbf{A}\mathbf{x}^* = \mathbf{x}^* \quad \Rightarrow \quad \mathbf{x}^* \in \mathrm{Eig}_{\mathbf{A}}(1)\,.$$

Does $\mathbf{A}$ possess an eigenvalue $= 1$? Does the associated eigenvector really provide a probability distribution (after scaling), that is, are all of its entries non-negative? Is this probability distribution unique? To answer these questions we have to study the matrix $\mathbf{A}$:

R. Hiptmair

rev 30401, November 22, 2010

For every stochastic matrix $\mathbf{A}$, by definition (6.3.3)

$$\mathbf{A}^T\mathbf{1} = \mathbf{1} \overset{(6.1.3)}{\Rightarrow} 1 \in \sigma(\mathbf{A}) ,$$

$$(2.5.8) \Rightarrow \|\mathbf{A}\|_1 = 1 \overset{\text{Thm 6.1.4}}{\Rightarrow} \rho(\mathbf{A}) = 1 ,$$

where $\rho(\mathbf{A})$ is the spectral radius of the matrix $\mathbf{A}$, see Sect. 6.1.

For $\mathbf{r} \in \mathrm{Eig}_{\mathbf{A}}(1)$, that is, $\mathbf{Ar} = \mathbf{r}$, denote by $|\mathbf{r}|$ the vector $(|r_i|)_{i=1}^N$. Since all entries of $\mathbf{A}$ are non-negative, we conclude by the triangle inequality that $\|\mathbf{Ar}\|_1 \leq \|\mathbf{A}|\mathbf{r}|\|_1$

$$\Rightarrow \quad 1 = \|\mathbf{A}\|_1 = \sup_{\mathbf{x}\in\mathbb{R}^N} \frac{\|\mathbf{Ax}\|_1}{\|\mathbf{x}\|_1} \geq \frac{\|\mathbf{A}|\mathbf{r}|\|_1}{\||\mathbf{r}|\|_1} \geq \frac{\|\mathbf{Ar}\|_1}{\|\mathbf{r}\|_1} = 1 .$$

$$\Rightarrow \quad \|\mathbf{A}|\mathbf{r}|\|_1 = \|\mathbf{Ar}\|_1 \overset{\text{if } a_{ij}>0}{\Rightarrow} |\mathbf{r}| = \pm\mathbf{r} .$$

Hence, different components of $\mathbf{r}$ cannot have opposite sign, which means, that $\mathbf{r}$ *can be chosen to have non-negative entries*, if the entries of $\mathbf{A}$ are strictly positive, which is the case for $\mathbf{A}$ from (6.3.4). After normalization $\|\mathbf{r}\|_1 = 1$ the eigenvector can be regarded as a probability distribution on $\{1, \ldots, N\}$.

If $\mathbf{Ar} = \mathbf{r}$ and $\mathbf{As} = \mathbf{s}$ with $(\mathbf{r})_i \geq 0$, $(\mathbf{s})_i \geq 0$, $\|\mathbf{r}\|_1 = \|\mathbf{s}\|_1 = 1$, then $\mathbf{A}(\mathbf{r}-\mathbf{s}) = \mathbf{r}-\mathbf{s}$. Hence, by the above considerations, also all the entries of $\mathbf{r}-\mathbf{s}$ are either non-negative or non-positive. By the

R. Hiptmair

rev 30401, November 22, 2010

assumptions on $\mathbf{r}$ and $\mathbf{s}$ this is only possible, if $\mathbf{r} - \mathbf{s} = 0$. We conclude that

$$\mathbf{A} \in]0,1]^{N,N} \text{ stochastic} \quad \Rightarrow \quad \dim \operatorname{Eig}_{\mathbf{A}}(1) = 1 \,. \tag{6.3.8}$$

Sorting the pages according to the size of the corresponding entries in $\mathbf{r}$ yields the famous "page rank".

Code 6.3.9: computing page rank vector $\mathbf{r}$ via `eig`

```
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

$$0 \leq (\mathbf{r})_i \leq 1 \,,$$
$$\|\mathbf{r}\|_1 = 1 \,,$$
$$\boxed{\mathbf{A}\mathbf{r} = \mathbf{r}} \,.$$

Inefficient implementation!

R. Hiptmair

rev 30401, November 22, 2010

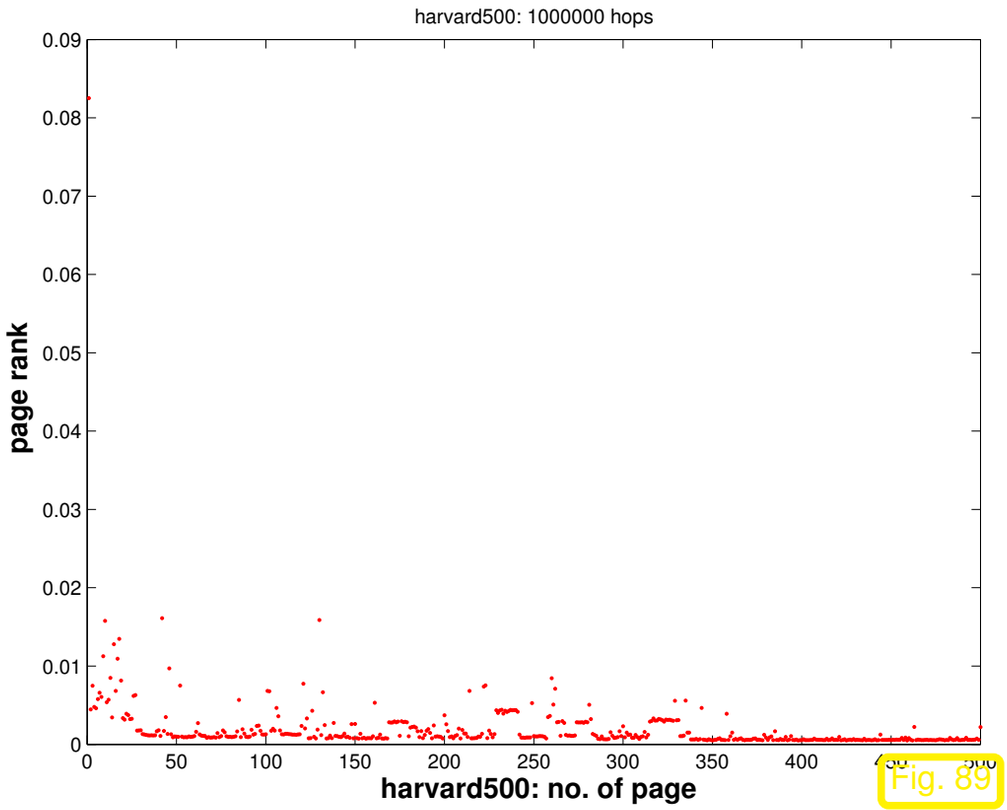


Fig. 89

stochastic simulation

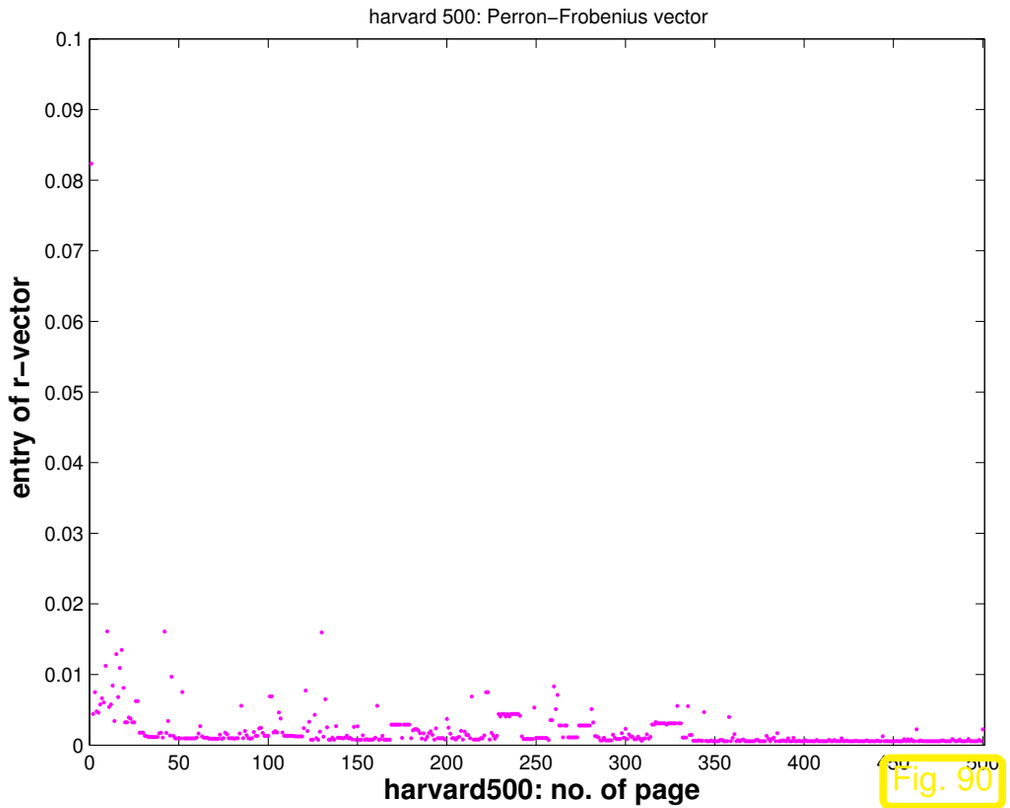


Fig. 90

eigenvector computation

The possibility to compute the stationary probability distribution of a Markov chain through an eigenvector of the transition probability matrix is due to a property of stationary Markov chains called ergodicity.

R. Hiptmair

rev 30401, November 22, 2010

$\mathbf{A} \hat{=}$ page rank transition probability matrix, see Code 6.3.4, $d = 0.15$, harvard500 example.

Errors: ▷

$$\left\| \mathbf{A}^k \mathbf{x}_0 - \mathbf{r} \right\|_1 \ ,$$

with $\mathbf{x}_0 = \mathbf{1}/N$, $N = 500$.

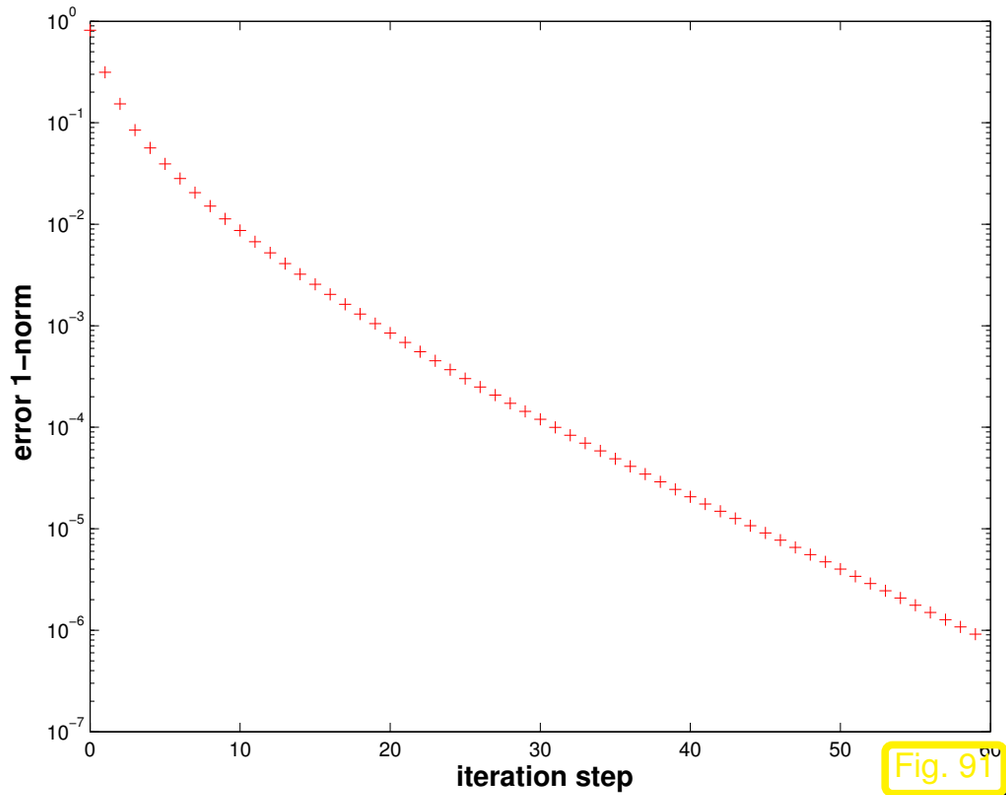


Fig. 91 ◇

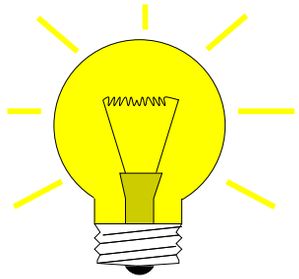
We observe linear convergence! (→ Def. 4.1.6, iteration error vs. iteration count ≈ straight line lin-log plot)

The computation of page rank amounts to finding the eigenvector of the matrix $\mathbf{A}$ of transition probabilities that belongs to its *largest* eigenvalue 1. This is addressed by an important class of practical eigenvalue problems:

R. Hiptmair

rev 30401, November 22, 2010

Task: given $\mathbf{A} \in \mathbb{K}^{n,n}$, find largest (in modulus) eigenvalue of $\mathbf{A}$ and (an) associated eigenvector.



Idea: (suggested by page rank computation, Code 6.3.6)

Iteration: $\mathbf{z}^{(k+1)} = \mathbf{A}\mathbf{z}^{(k)}$, $\mathbf{z}^{(0)}$ arbitrary

Example 6.3.10 (Power iteration). $\rightarrow$ Ex. 6.3.1

Try the above iteration for general 10×10-matrix, largest eigenvalue 10, algebraic multiplicity 1.

R. Hiptmair

rev 30401, November 22, 2010

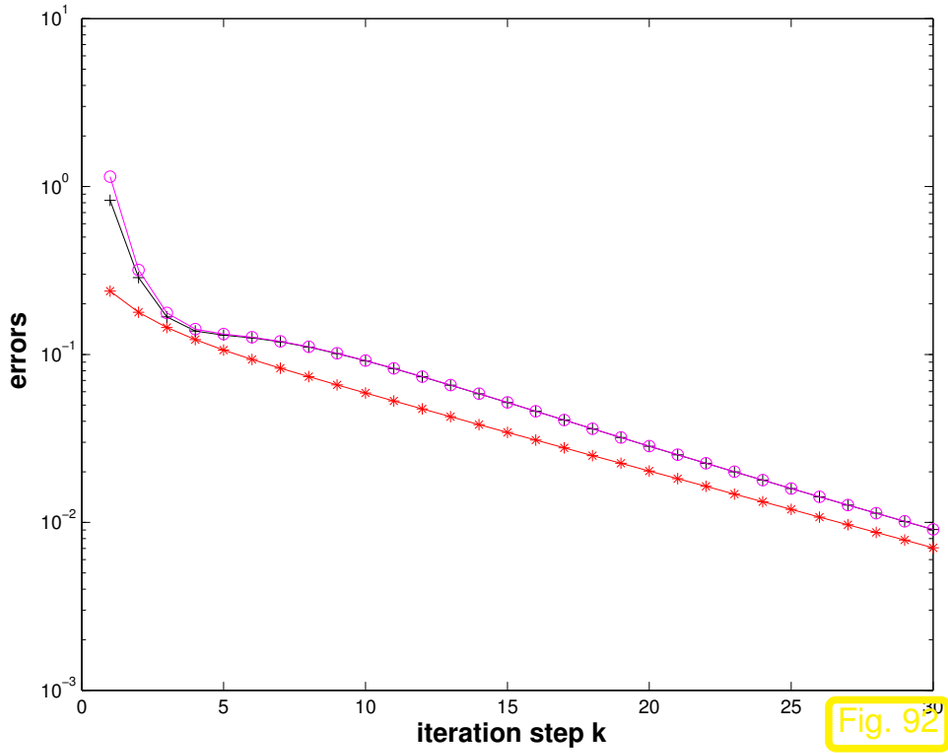


Fig. 92

```
d = (1:10)'; n = length(d);
S = triu(diag(n:-1:1,0)+...
    ones(n,n));
A = S*diag(d,0)*inv(S);
```

◁ error norm $\left\| \frac{\mathbf{z}^{(k)}}{\|\mathbf{z}^{(k)}\|} - (\mathbf{S})_{:,10} \right\|$

(Note: $(\mathbf{S})_{:,10} \hat{=}$ eigenvector for eigenvalue 10)

$\mathbf{z}^{(0)}$ = random vector

Observation: linear convergence of (normalized) eigenvectors!

◇

R. Hiptmair

rev 30401, November 22, 2010

Suggests direct power method (*ger.:* Potenzmethode): iterative method (→ Sect. 4.1)

$$\begin{aligned} &\text{initial guess:} \quad \mathbf{z}^{(0)} \text{ “arbitrary”} , \\ &\text{next iterate:} \quad \mathbf{w} := \mathbf{A}\mathbf{z}^{(k-1)} , \quad \mathbf{z}^{(k)} := \frac{\mathbf{w}}{\|\mathbf{w}\|_2} , \quad k = 1, 2, \ldots . \end{aligned} \tag{6.3.11}$$

Note: the “normalization” of the iterates in (6.3.11) does not change anything (in exact arithmetic) and helps *avoid overflow* in floating point arithmetic.

Computational effort: $1\times$ matrix$\times$vector per step ➢ inexpensive for sparse matrices

A persuasive theoretical justification for the direct power method:

Assume $\mathbf{A} \in \mathbb{K}^{n,n}$ to be diagonalizable:

$$\Leftrightarrow \quad \exists \text{ basis } \{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \text{ of eigenvectors of } \mathbf{A}: \quad \mathbf{A}\mathbf{v}_j = \lambda_j \mathbf{v}_j, \lambda_j \in \mathbb{C}.$$

Assume

$$|\lambda_1| \leq |\lambda_2| \leq \cdots \leq |\lambda_{n-1}| < |\lambda_n| \quad , \quad \|\mathbf{v}_j\|_2 = 1 \, . \tag{6.3.12}$$

Key observations for power iteration (6.3.11)

$$\mathbf{z}^{(k)} = \mathbf{A}^k \mathbf{z}^{(0)} \qquad (\rightarrow \text{name "power method"}) \tag{6.3.13}$$

$$\mathbf{z}^{(0)} = \sum_{j=1}^{n} \zeta_j \mathbf{v}_j \quad \Rightarrow \quad \mathbf{z}^{(k)} = \sum_{j=1}^{n} \zeta_j \lambda_j^k \, \mathbf{v}_j \, . \tag{6.3.14}$$

Due to (6.3.12) for large $k \gg 1$ ($\Rightarrow |\lambda_n^k| \gg |\lambda_j^k|$ for $j \neq n$) the contribution of $\mathbf{v}_n$ (size $\zeta_n \lambda_n^k$) in the *eigenvector expansion* (6.3.14) will be much larger than the contribution (size $\zeta_n \lambda_j^k$) of any other eigenvector (, if $\zeta_n \neq 0$): the eigenvector for λ_n will swamp all other for $k \to \infty$.

Further (6.3.14) nutures expectation: $\mathbf{v}_n$ will become dominant in $\mathbf{z}^{(k)}$ the faster, the better $|\lambda_n|$ is separated from $|\lambda_{n-1}|$, see Thm. 6.3.19 for rigorous statement.

$\mathbf{z}^{(k)} \rightarrow$ eigenvector, but how do we get the associated eigenvalue λ_n ?

When (6.3.11) has converged, two common ways to recover $\lambda_{\max}$ $\rightarrow$ [10, 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 $\lambda_{\max} \approx \operatorname{argmin}_{\theta\in\mathbb{R}} \left\|\mathbf{A}\mathbf{z}^{(k)} - \theta\mathbf{z}^{(k)}\right\|_2^2$ ➤ $\lambda_{\max} \approx \dfrac{(\mathbf{z}^{(k)})^H\mathbf{A}\mathbf{z}^{(k)}}{\left\|\mathbf{z}^{(k)}\right\|_2^2}$.

This latter formula is extremely useful, which has earned it a special name:

Definition 6.3.15. *For $\mathbf{A} \in \mathbb{K}^{n,n}$, $\mathbf{u} \in \mathbb{K}^n$ the Rayleigh quotient is defined by*

$$\rho_{\mathbf{A}}(\mathbf{u}) := \frac{\mathbf{u}^H\mathbf{A}\mathbf{u}}{\mathbf{u}^H\mathbf{u}} .$$

An immediate consequence of the definitions:

$$\lambda \in \sigma(\mathbf{A}) \quad , \quad \mathbf{z} \in \mathrm{Eig}_\lambda(\mathbf{A}) \quad \Rightarrow \quad \rho_{\mathbf{A}}(\mathbf{z}) = \lambda \, . \tag{6.3.16}$$

Example 6.3.17 (Direct power method). → Ex. 6.3.17 cnt'd

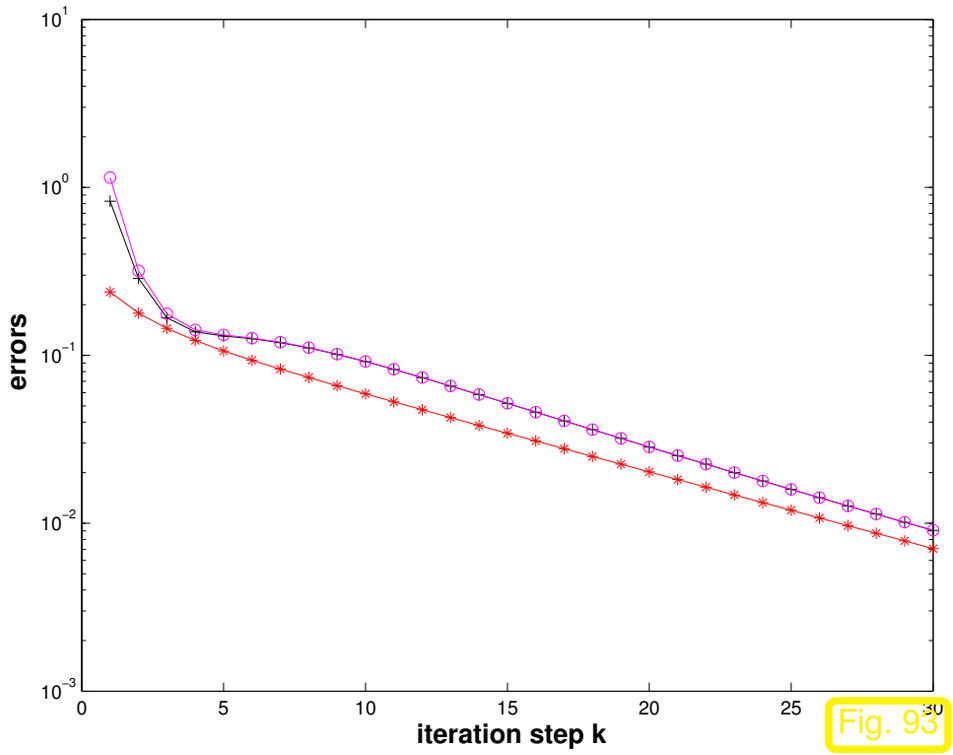


Fig. 93

```
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 $\left\| \mathbf{z}^{(k)} - \mathbf{s}_{\cdot,n} \right\|$

$+$: $\left| \lambda_n - \dfrac{\left\| \mathbf{A}\mathbf{z}^{(k-1)} \right\|_2}{\left\| \mathbf{z}^{(k-1)} \right\|_2} \right|$

$\mathbf{z}^{(0)}$ = random vector

R. Hiptmair

rev 30401, November 22, 2010

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)');` ➢ $|\lambda_{n-1}| : |\lambda_n| = 0.9866$

Code 6.3.18: Investigating convergence of direct power method

```
% Demonstration of direct power method for Ex. 6.3.17
maxit = 30; d = [1:10]'; n = length(d);
% Initialize the matrix A
S = triu(diag(n:-1:1,0)+ones(n,n));
A = S*diag(d,0)*inv(S);
% This calculates the exact eigenvalues (for error calculation)
[V,D] = eig(A);

k = find(d == max(abs(d)));
if (length(k) > 1), error('No single largest EV'); end;
ev = X(:,k(1)); ev = ev/norm(ev); ev
ew = d(k(1)); ew
if (ew < 0), sgn = -1; else sgn = 1; end

z = rand(n,1); z = z/norm(z);
s = 1;
res = [];

% Actual direct power iteration
```


R. Hiptmair

rev 30401, November 22, 2010

```
for i=1:maxit
  w = A*z; l = norm(w); rq = real(dot(w,z)); z = w/l;
  res = [res;i,l,norm(s*z-ev),abs(l-abs(ew)),abs(sgn*rq-ew)];
  s = s * sgn;
end

% Plot the result
semilogy(res(:,1),res(:,3),'r-*',res(:,1),res(:,4),'k-+',res(:,1),res(:,5),'m-o')
xlabel('{\bf iteration step k}','FontSize',14);
ylabel('{\bf errors}','FontSize',14);
print -deps2c '../PICTURES/pm1.eps';
```

$$\rho_{\mathrm{EV}}^{(k)} := \frac{\left\| \mathbf{z}^{(k)} - \mathbf{s}_{\cdot,n} \right\|}{\left\| \mathbf{z}^{(k-1)} - \mathbf{s}_{\cdot,n} \right\|} ,$$

$$\rho_{\mathrm{EW}}^{(k)} := \frac{|\rho_{\mathbf{A}}(\mathbf{z}^{(k)}) - \lambda_n|}{|\rho_{\mathbf{A}}(\mathbf{z}^{(k-1)}) - \lambda_n|} .$$

	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

◇

Theorem 6.3.19 (Convergence of direct power method). → *[10, 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

$$\left\|\mathbf{A}\mathbf{z}^{(k)}\right\|_2 \to \lambda_n \quad , \quad \mathbf{z}^{(k)} \to \pm\mathbf{v} \quad \textit{linearly with rate} \quad \frac{|\lambda_{n-1}|}{|\lambda_n|} \, ,$$

where $\mathbf{z}^{(k)}$ are the iterates of the direct power iteration and $\mathbf{y}^H\mathbf{z}^{(0)} \neq 0$ is assumed.

Remark 6.3.20 (Initial guess for power iteration).

roundoff errors ➤ $\mathbf{y}^H\mathbf{z}^{(0)} \neq 0$ always satisfied in practical computations

Usual (not the best!) choice for $\mathbf{x}^{(0)}$ = random vector △

Remark 6.3.21 (Termination criterion for direct power iteration). (→ Sect. 4.1.2)

Adaptation of a posteriori termination criterion (4.2.20)

"relative change" $\leq$ `tol`:

$$\begin{cases} \left\| \mathbf{z}^{(k)} - \mathbf{z}^{(k-1)} \right\| \leq (1/L - 1)\texttt{tol} \;, \\ \left| \dfrac{\left\| \mathbf{A}\mathbf{z}^{(k)} \right\|}{\left\| \mathbf{z}^{(k)} \right\|} - \dfrac{\left\| \mathbf{A}\mathbf{z}^{(k-1)} \right\|}{\left\| \mathbf{z}^{(k-1)} \right\|} \right| \leq (1/L - 1)\texttt{tol} \quad \text{see (4.1.24)} \;. \end{cases}$$

Estimated rate of convergence

△

R. Hiptmair

rev 30401, January 28, 2011

6.3.2 Inverse Iteration [10, Sect. 7.6]

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

```
1 M = imread('eth.pbm');
2 [m,n] = size(M);
3 fprintf('%dx%d grey scale pixel image\n',m,n);
4 figure; image(M); title('ETH view');
5 col = [0:1/215:1]'*[1,1,1]; colormap(col);
```

(Fuzzy) task: Local segmentation

Find connected patches of image of the same shade/color

More general segmentation problem (non-local): identify parts of the image, not necessarily connected, with the same texture.

Next: Statement of (rigorously defined) problem, *cf.* Sect. 2.5.2:

Preparation: Numbering of pixels $1 \ldots, mn$, e.g, lexicographic numbering:

- pixel set $\mathcal{V} := \{1 \ldots, nm\}$
- indexing: index$(\text{pixel}_{i,j}) = (i-1)n + j$

✎ notation: $p_k := (\mathbf{P})_{ij}$, if $k = \text{index}(\text{pixel}_{i,j}) = (i-1)n + j$, $k = 1, \ldots, N := mn$

Local similarity matrix:

$$\mathbf{W} \in \mathbb{R}^{N,N} \;, \quad N := mn \;, \tag{6.3.24}$$

$$(\mathbf{W})_{ij} = \begin{cases} 0 & \text{, if pixels } i,\, j \text{ not adjacent,} \\ 0 & \text{, if } i = j \;, \\ \sigma(p_i, p_j) & \text{, if pixels } i,\, j \text{ adjacent.} \end{cases}$$

$\leftrightarrow \hat{=}$ adjacent pixels ▷

Similarity function, e.g., with $\alpha > 0$

$$\sigma(x,y) := \exp(-\alpha(x-y)^2) \;, \quad x, y \in \mathbb{R} \;.$$

Lexicographic numbering ▷

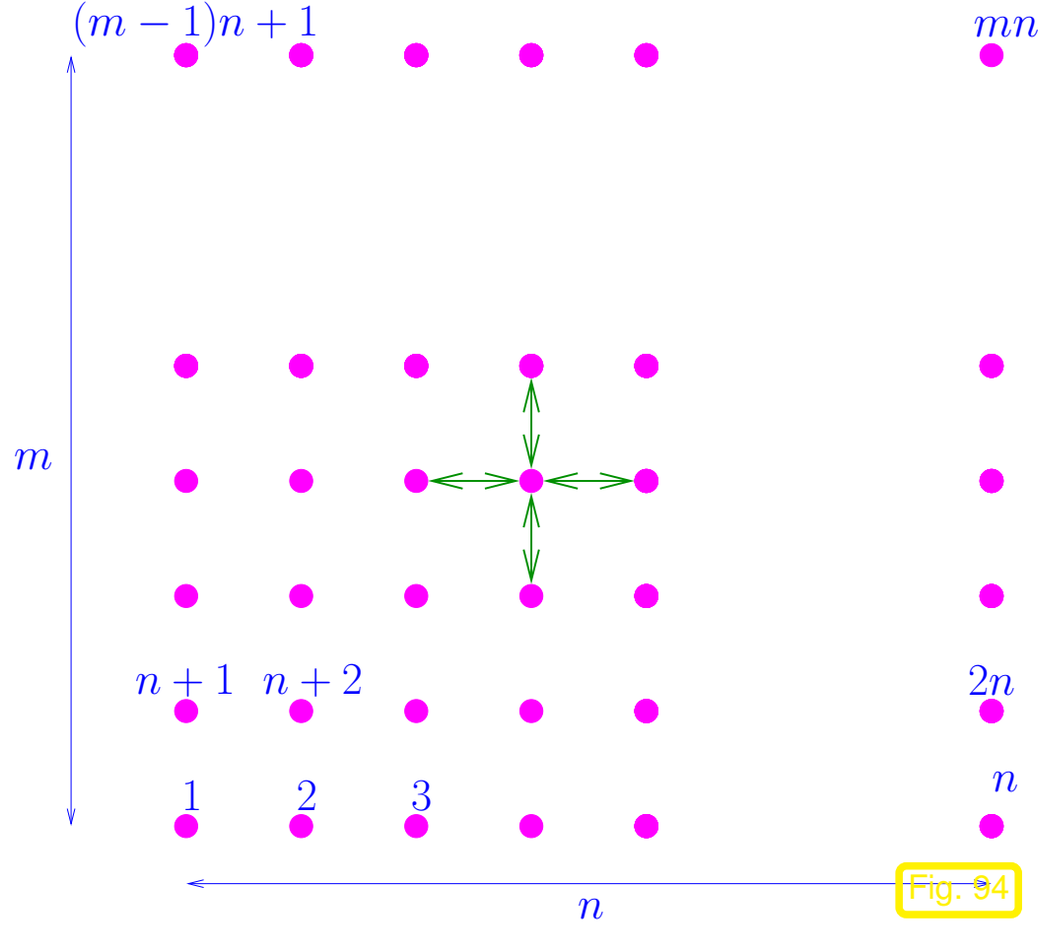


Fig. 94

The entries of the matrix $\mathbf{W}$ measure the "similarity" of neighboring pixels: if $(\mathbf{W})_{ij}$ is large, they encode (almost) the same intensity, if $(\mathbf{W})_{ij}$ is close to zero, then they belong to parts of the picture with very different brightness. In the latter case, the boundary of the segment may separate the two pixels.

R. Hiptmair

rev 30401, January 28, 2011

Definition 6.3.25 (Normalized cut). *(→ [48, 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 $$\mathrm{cut}(\mathcal{X}) := \sum_{i \in \mathcal{X}, j \notin \mathcal{X}} w_{ij} \quad , \quad \mathrm{weight}(\mathcal{X}) := \sum_{i \in \mathcal{X}, j \in \mathcal{V}} w_{ij} .$$

Segmentation problem (rigorous statement):

$$\text{find} \quad \mathcal{X}^* \subset \mathcal{V}: \quad \mathcal{X}^* = \operatorname*{argmin}_{\mathcal{X} \subset \mathcal{V}} \mathrm{Ncut}(\mathcal{X}) \quad . \tag{6.3.26}$$

NP-hard combinatorial optimization problem !

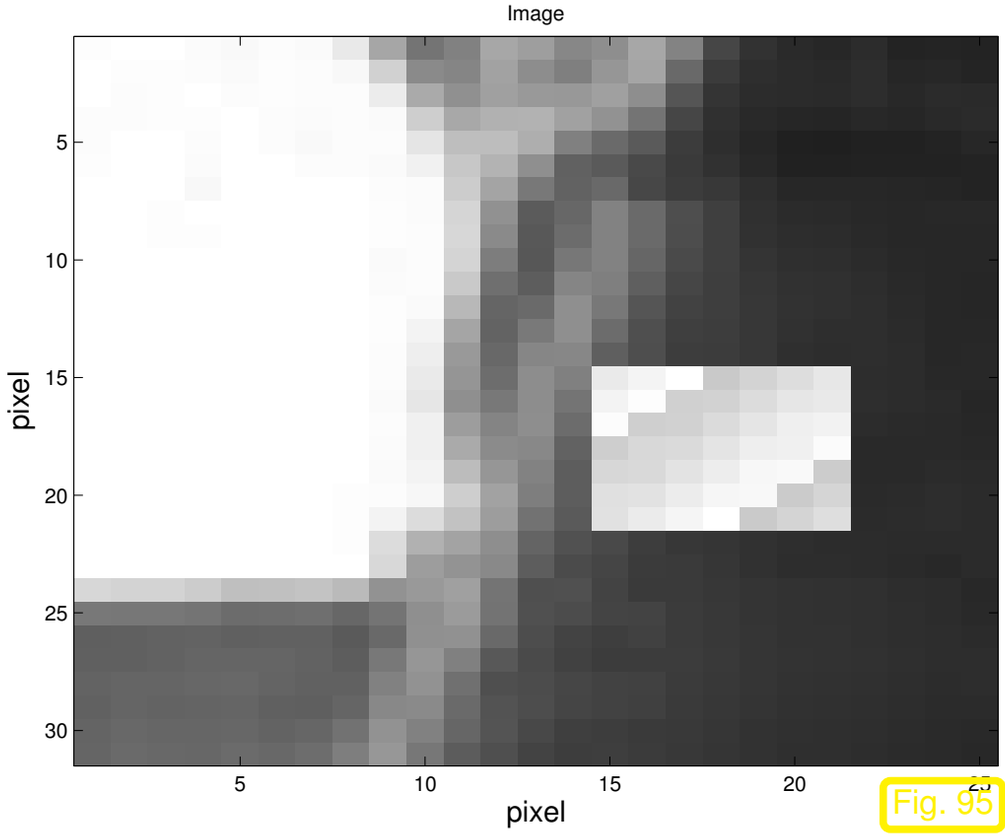


Image
pixel
pixel
5
10
15
20
25
30

Fig. 95

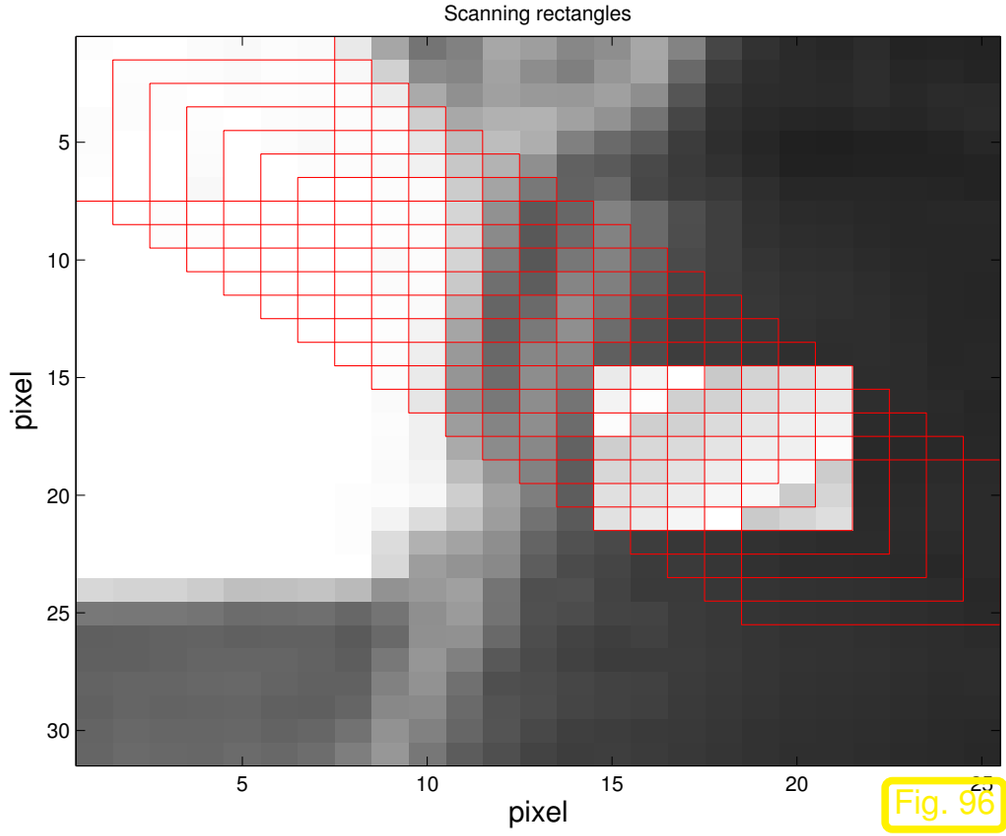

Scanning rectangles
pixel
pixel
5
10
15
20
25
30

Fig. 96

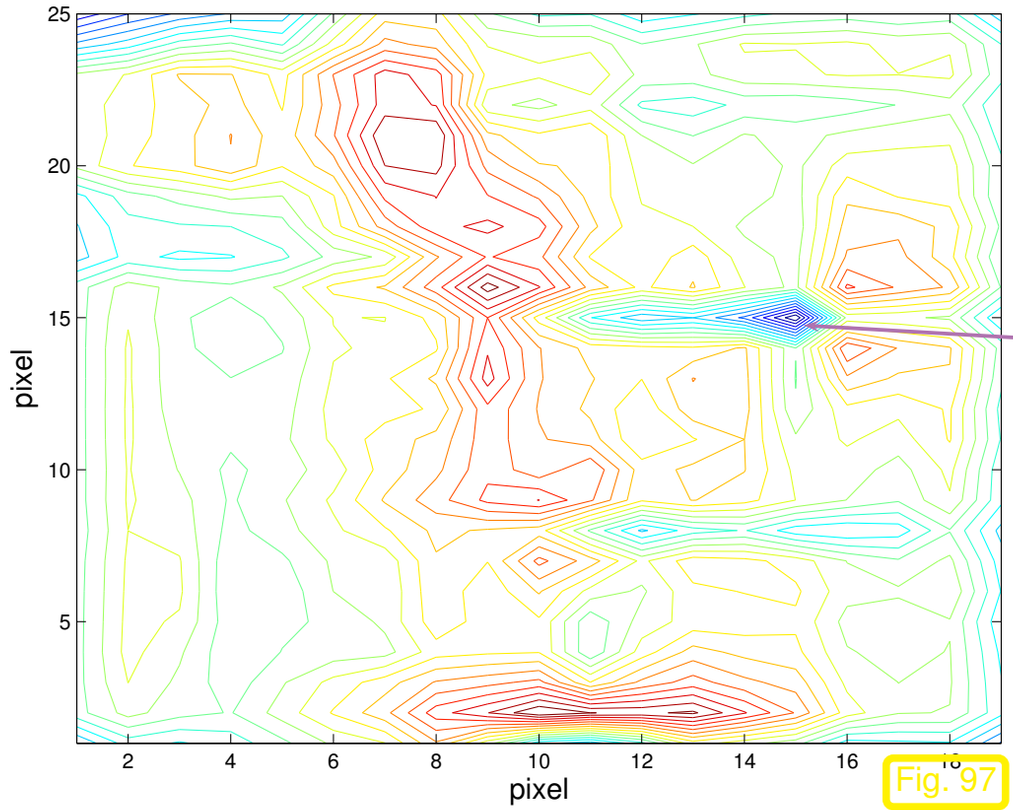


Fig. 97

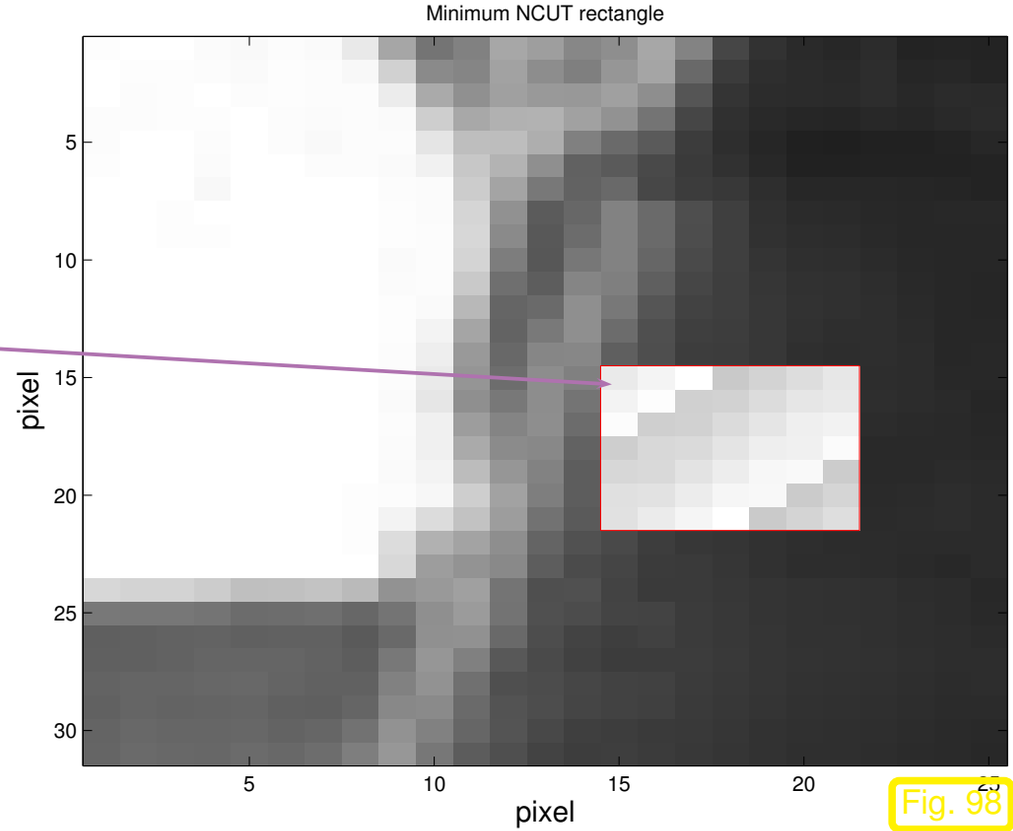


Fig. 98

△ $\mathrm{Ncut}(\mathcal{X})$ for pixel subsets $\mathcal{X}$ defined by sliding rectangles, see Fig. 96.

Equivalent reformulation:

indicator function: $$z : \{1, \ldots, N\} \mapsto \{-1, 1\}\,, \quad z_i := z(i) = \begin{cases} 1 & \text{, if } i \in \mathcal{X}\,, \\ -1 & \text{, if } i \notin \mathcal{X}\,. \end{cases} \tag{6.3.27}$$

R. Hiptmair

rev 30401, January 28, 2011

$$\text{Ncut}(\mathcal{X}) = \frac{\sum\limits_{z_i>0, z_j<0} -w_{ij} z_i z_j}{\sum\limits_{z_i>0} d_i} + \frac{\sum\limits_{z_i>0, z_j<0} -w_{ij} z_i z_j}{\sum\limits_{z_i<0} d_i} , \tag{6.3.28}$$

$$d_i = \sum_{j\in\mathcal{V}} w_{ij} = \text{weight}(\{i\}) . \tag{6.3.29}$$

Sparse matrices:

$$\mathbf{D} := \text{diag}(d_1, \ldots, d_N) \in \mathbb{R}^{N,N} , \quad \mathbf{A} := \mathbf{D} - \mathbf{W} = \mathbf{A}^T . \quad \Rightarrow \quad \mathbf{1}^T \mathbf{A} = \mathbf{A}\mathbf{1} = 0 . \tag{6.3.30}$$

Code 6.3.31: assembly of $\mathbf{A}$, $\mathbf{D}$

```
1 function [A,D] = imgsegmat(P)
2 P = double(P); [n,m] = size(P);
3 spdata = zeros(4*n*m,1); spidx  = zeros(4*n*m,2);
4 k = 1;
5 for ni=1:n
6   for mi=1:m
7     mni = (mi-1)*n+ni;
```

```
8       if (ni-1>0), spidx(k,:) = [mni,mni-1];
9         spdata(k) = Sfun(P(ni,mi),P(ni-1,mi));
10        k = k + 1;
11      end
12      if (ni+1<=n), spidx(k,:) = [mni,mni+1];
13        spdata(k) = Sfun(P(ni,mi),P(ni+1,mi));
14        k = k + 1;
15      end
16      if (mi-1>0), spidx(k,:) = [mni,mni-n];
17        spdata(k) = Sfun(P(ni,mi),P(ni,mi-1));
18        k = k + 1;
19      end
20      if (mi+1<=m), spidx(k,:) = [mni,mni+n];
21        spdata(k) = Sfun(P(ni,mi),P(ni,mi+1));
22        k = k + 1;
23      end
24    end
25  end
26  % Efficient initialization, see Sect. 2.6.2, Ex. 2.6.7
27  W = sparse(spidx(1:k-1,1),spidx(1:k-1,2),spdata(1:k-1),n*m,n*m);
28  D = spdiags(full(sum(W')'),[0],n*m,n*m);
29  A = W - D;
```

Lemma 6.3.32 (Ncut and Rayleigh quotient). *(→ [48, Sect. 2])*

With $\mathbf{z} \in \{-1,1\}^N$ *according to* (6.3.27) *there holds*

$$\mathrm{Ncut}(\mathcal{X}) = \frac{\mathbf{y}^T\mathbf{A}\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}} \quad , \quad \mathbf{y} := (\mathbf{1}+\mathbf{z}) - \beta(\mathbf{1}-\mathbf{z}) \; , \quad \beta := \frac{\sum\limits_{z_i>0} d_i}{\sum\limits_{z_i<0} d_i} \; .$$

generalized Rayleigh quotient $\rho_{\mathbf{A},\mathbf{D}}(\mathbf{y})$

Proof. Note that by (6.3.27) $(\mathbf{1}-\mathbf{z})_i = 0 \;\Leftrightarrow\; i \in \mathcal{X}$, $(\mathbf{1}+\mathbf{z})_i = 0 \;\Leftrightarrow\; i \notin \mathcal{X}$. Hence, since $(\mathbf{1}+\mathbf{z})^T\mathbf{D}(\mathbf{1}-\mathbf{z}) = 0$,

$$4\,\mathrm{Ncut}(\mathcal{X}) = (\mathbf{1}+\mathbf{z})^T\mathbf{A}(\mathbf{1}+\mathbf{z})\left(\frac{1}{\kappa\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{1}{(1-\kappa)\mathbf{1}^T\mathbf{D}\mathbf{1}}\right) = \frac{\mathbf{y}^T\mathbf{A}\mathbf{y}}{\beta\mathbf{1}^T\mathbf{D}\mathbf{1}} \; ,$$

where $\kappa := \sum\limits_{z_i>0} d_i / \sum\limits_i d_i = \frac{\beta}{1+\beta}$. Also observe

$$\mathbf{y}^T\mathbf{D}\mathbf{y} = (\mathbf{1}+\mathbf{z})^T\mathbf{D}(\mathbf{1}+\mathbf{z}) + \beta^2(\mathbf{1}-\mathbf{z})^T\mathbf{D}(\mathbf{1}-\mathbf{z}) =$$
$$4(\sum_{z_i>0} d_i + \beta^2\sum_{z_i<0} d_i) = 4\beta\sum_i d_i = 4\beta\mathbf{1}^T\mathbf{D}\mathbf{1} \; .$$

R. Hiptmair

rev 30401, January 28, 2011

This finishes the proof. □

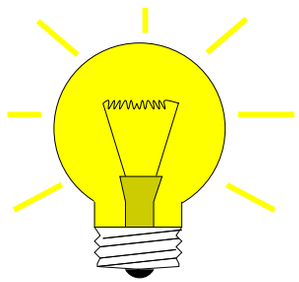
$$\blacktriangleright \quad \text{segmentation problem (6.3.26)} \quad \Leftrightarrow \quad \operatorname{argmin}_{\mathbf{y}\in\{2,-2\beta\}^N,\, \mathbf{1}^T\mathbf{D}\mathbf{y}=0} \rho_{\mathbf{A},\mathbf{D}}(\mathbf{y}) \,. \tag{6.3.33}$$

still NP-hard

➢ Minimizing $\mathrm{Ncut}(\mathcal{X})$ amounts to minimizing a (generalized) Rayleigh quotient (→ Def. 6.3.15) over a *discrete* set of vectors, which is still an NP-hard problem.

R. Hiptmair

rev 30401, January 28, 2011



Idea: Relaxation

Discrete optimization problem → continuous optimization problem

$$(6.3.33) \quad \rightarrow \quad \operatorname{argmin}_{\mathbf{y}\in\mathbb{R}^N,\, \|\mathbf{y}\|_{\mathbf{D}}=1,\, \mathbf{1}^T\mathbf{D}\mathbf{y}=0} \rho_{\mathbf{A},\mathbf{D}}(\mathbf{y}) \,. \tag{6.3.34}$$

The constraints $\|\mathbf{y}\|_{\mathbf{D}} = 1$, $\mathbf{1}^T\mathbf{D}\mathbf{y} = 0$ compound the difficulties of solving (6.3.34). However, the next theorem establishes a link to a generalized eigenvalue problem.

Theorem 6.3.35 (Courant-Fischer min-max theorem). $\rightarrow$ *[20, Thm. 8.1.2]*

Let $\lambda_1 < \lambda_2 < \cdots < \lambda_m$, $m \leq n$, *be the sorted sequence of the (real!) eigenvalues of* $\mathbf{A} = \mathbf{A}^H \in \mathbb{C}^{n,n}$. *Write*

$$U_0 = \{0\}\,, \quad U_\ell := \sum_{j=1}^{\ell} \mathrm{Eig}_{\mathbf{A}}(\lambda_j)\,, \ell = 1,\ldots,m \quad \textit{and} \quad U_\ell^\perp := \{\mathbf{x} \in \mathbb{C}^n \colon \mathbf{u}^T\mathbf{x} = 0\ \forall \mathbf{u} \in U_\ell\}\,.$$

Then

$$\min_{\mathbf{y} \in U_{\ell-1}^\perp \setminus \{0\}} \rho_{\mathbf{A}}(\mathbf{y}) = \lambda_\ell\,, \quad 1 \leq \ell \leq m\,, \qquad \operatorname*{argmin}_{\mathbf{y} \in U_{\ell-1}^\perp \setminus \{0\}} \rho_{\mathbf{A}}(\mathbf{y}) \subset \mathrm{Eig}_{\mathbf{A}}(\lambda_\ell)\,.$$

Proof. For diagonal $\mathbf{A} \in \mathbb{R}^{n,n}$ the assertion of the theorem is obvious. Thus, Cor. 6.1.9 settles everything. □

R. Hiptmair

rev 30401, January 28, 2011

Application: If $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$ with eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, then

$$\lambda_1 = \min_{\mathbf{z}\in\mathbb{R}^n} \rho_{\mathbf{A}}(z) \quad , \quad \lambda_2 = \min_{\mathbf{z}\in\mathbb{R}^n, \mathbf{z}\perp\mathbf{v}_1} \rho_{\mathbf{A}}(z) \, ,$$

where $\mathbf{v}_1 \in \mathrm{Eig}_{\mathbf{A}}(\lambda_1) \setminus \{0\}$.

Well, in Lemma 6.3.32 we encounter a generalized Rayleigh quotient $\rho_{\mathbf{A},\mathbf{D}}(\mathbf{y})$! How can the min-max theorem be applied to it?

◮ Transformation: $\rho_{\mathbf{A},\mathbf{D}}(\mathbf{D}^{-1/2}\mathbf{z}) = \rho_{\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}}(\mathbf{z}) \, , \quad \mathbf{y} \in \mathbb{R}^n \, .$

◮ Apply Thm. 6.3.35 to transformed matrix $\widetilde{\mathbf{A}} := \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. The *elementary* manipulations show

$$\begin{aligned} \text{(6.3.34)} \quad \Leftrightarrow \quad & \operatorname*{argmin}_{\mathbf{1}^T\mathbf{D}\mathbf{y}=0} \rho_{\mathbf{A},\mathbf{D}}(\mathbf{y}) \overset{\mathbf{z}=\mathbf{D}^{1/2}\mathbf{y}}{=} \operatorname*{argmin}_{\mathbf{1}\mathbf{D}^{1/2}\mathbf{z}=0} \rho_{\mathbf{A},\mathbf{D}}(\mathbf{D}^{-1/2}\mathbf{z}) \\ & = \operatorname*{argmin}_{\mathbf{1}\mathbf{D}^{1/2}\mathbf{z}=0} \rho_{\widetilde{\mathbf{A}}}(\mathbf{z}) \quad \text{with} \quad \widetilde{\mathbf{A}} := \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} \, . \end{aligned} \tag{6.3.36}$$

An agreeable surprise:

$\mathbf{A}\mathbf{1} = 0$ and $\mathbf{A}$ is diagonally dominant, see Def. 2.7.7.

R. Hiptmair

rev 30401, January 28, 2011

Lemma 2.7.12 $\Longrightarrow$ $\mathbf{A}$ is *positive semidefinite* ($\rightarrow$ Def. 2.7.9) with smallest eigenvalue 0.

Lemma 6.3.37. $\dim \operatorname{Eig}_{\mathbf{A}}(0) = 1$

Proof. We rely on an smart averaging argument, which confirms that $\mathbf{Ax} = 0$ implies $\mathbf{x} \in \operatorname{Span}\{\mathbf{1}\}$.

Let $\mathbf{Ax} = 0$ and

$$j \in \{1, \ldots, n\}: \quad |x_j| = \max_k |x_k| \; . \tag{6.3.38}$$

By definition of $\mathbf{A}$, see (6.3.30), we know $a_{ii} > 0$, $a_{ij} < 0$ for $i \neq j$, and $\sum_{k=1}^{n} a_{jk} = 0$, so that

$$x_j = \frac{1}{a_{jj}} \Big(\sum_{k \neq j} (-a_{jk}) x_k \Big)$$

is an *average*, which, by (6.3.38), implies

$$a_{jk} \neq 0 \quad \Rightarrow \quad x_k = x_j \; .$$

Since the graph of $\mathbf{A}$ is connected, all components of $\mathbf{x}$ are seen to be equal. □

Lemma 6.3.37 $\Rightarrow$ $\operatorname{Eig}_{\widetilde{\mathbf{A}}}(0) = \operatorname{Span}\left\{\mathbf{D}^{1/2}\mathbf{1}\right\}$

Continue (6.3.36)

$$\operatorname*{argmin}_{\mathbf{1}^T\mathbf{D}\mathbf{y}=0} = \mathbf{D}^{-1/2} \operatorname*{argmin}_{\mathbf{z}\perp\operatorname{Eig}_{\widetilde{\mathbf{A}}}(0)} \rho_{\widetilde{\mathbf{A}}}(\mathbf{z}) \overset{\text{Thm. 6.3.35}}{=} \mathbf{D}^{-1/2}\operatorname{Eig}_{\widetilde{\mathbf{A}}}(\widetilde{\lambda}) \quad \text{for} \quad \widetilde{\lambda} = \min\{\sigma(\widetilde{\mathbf{A}}) \setminus \{0\}\} .$$

Summary:

Thm. 6.3.35 & $\mathbf{A}\mathbf{1} = 0$ $\Rightarrow$ solution of (6.3.34) is eigenvector for smallest non-zero eigenvalue of $\mathbf{A}\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$.

Algorithm 6.3.39 (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 $\mathbf{x}^*$, $\|\mathbf{x}^*\|_2 = 1$ as eigenvector for 2nd smallest eigenvalue of generalized eigenvalue problem $\mathbf{A}\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$.

❸ Define the image segment as pixel set

$$\mathcal{X} := \{i \in \{1,\ldots,N\}: x_i^* > \frac{1}{N}\sum_{i=1}^{N} x_i^*\} \,. \tag{6.3.40}$$

mean value of entries of $\mathbf{x}^*$

Code 6.3.41: 1st stage of segmentation of grayscale image

```
1 P = imread('image.pbm'); [m,n] = size(P); [A,D] = imgsegmat(P);
2 [V,E] = eig(full(A+D),full(D)); % grossly inefficient !
3 xs = reshape(V(:,2),m,n); Xidx = find(xs>(sum(sum(xs))/(n*m)));
```

1st-stage of segmentation of 31×25 grayscale pixel image (`root.pbm`, red pixels $\hat{=} \mathcal{X}$, $\sigma(x,y) = \exp(-(x-y/10)^2)$)

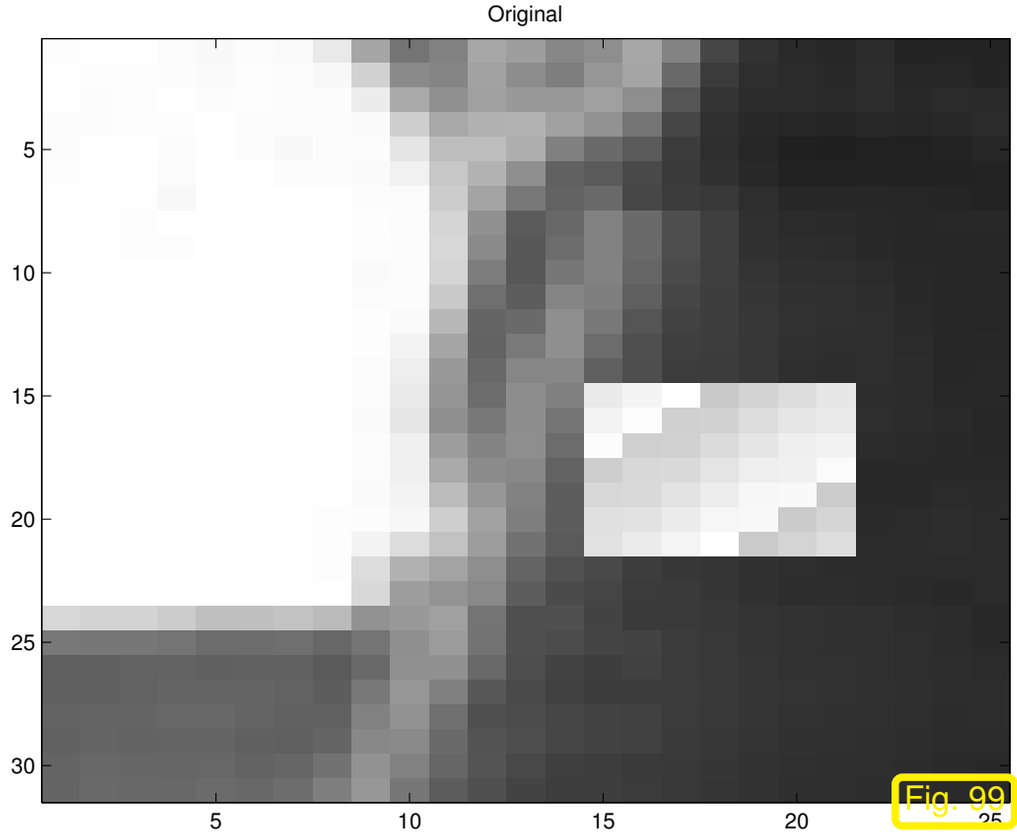


Fig. 99

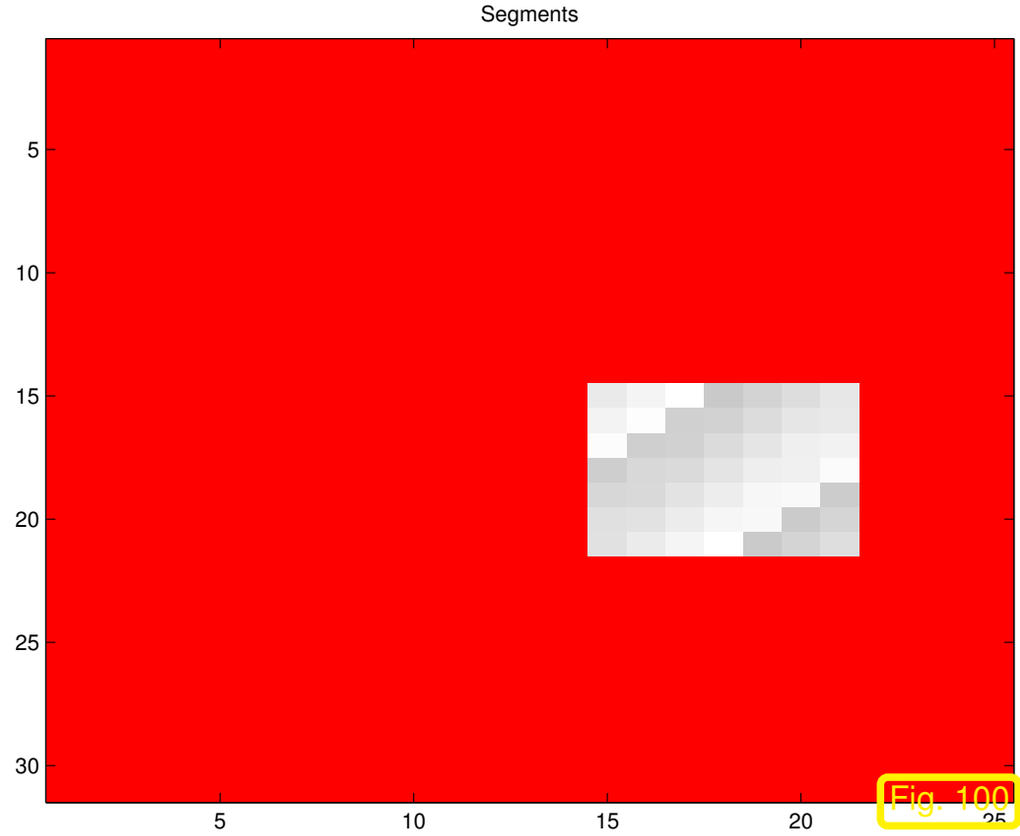


Fig. 100

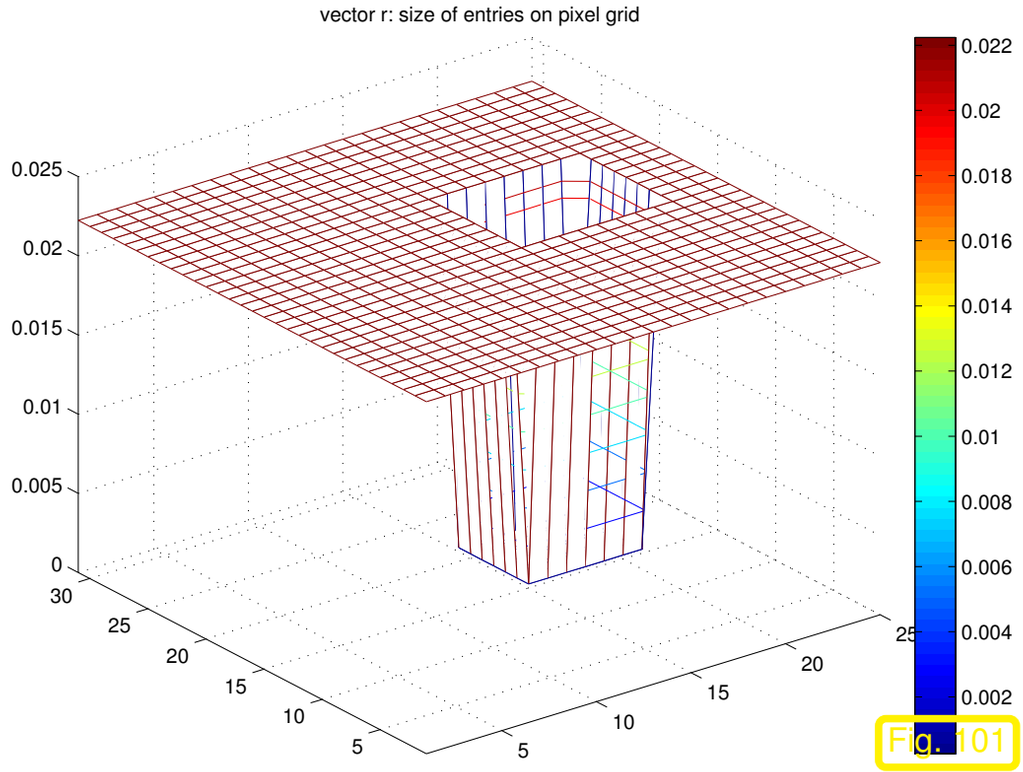


Fig. 101

Image from Fig. 99:

◁ eigenvector $\mathbf{x}^*$ plotted on pixel grid

R. Hiptmair

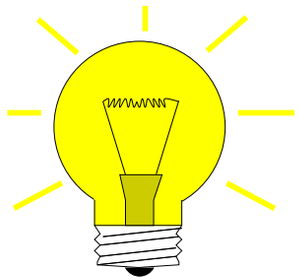
rev 30401, January 28, 2011

To identify more segments, the same algorithm is *recursively applied* to segment parts of the image already determined.

Practical segmentation algorithms rely on many more steps of which the above algorithm is only one, preceeded by substantial preprocessing. Moreover, they dispense with the strictly local perspective adopted above and take into account more distant connections between image parts, often in a randomized fashion [48].

Before discussing how to find the eigenvector belonging *second smallest* eigenvalue, we consider a simpler class of eigenvalue problems.

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 ($\rightarrow$ Sect. 6.3.1) for $\mathbf{A}^{-1}$ = inverse iteration

R. Hiptmair

rev 30401, January 28, 2011

Code 6.3.42: 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. 2.2.12
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
```

Note: reuse of LU-factorization, see Rem. 2.2.12

Remark 6.3.43 (Shifted inverse iteration).

More general task:

For $\alpha \in \mathbb{C}$ find $\lambda \in \sigma(\mathbf{A})$ such that $|\alpha - \lambda| = \min\{|\alpha - \mu|, \mu \in \sigma(\mathbf{A})\}$

R. Hiptmair

rev 30401, January 28, 2011

Shifted inverse iteration: [10, Alg .7.24]

$$\mathbf{z}^{(0)} \text{ arbitrary}, \quad \mathbf{w} = (\mathbf{A} - \alpha\mathbf{I})^{-1}\mathbf{z}^{(k-1)} \quad , \quad \mathbf{z}^{(k)} := \frac{\mathbf{w}}{\|\mathbf{w}\|_2} , \quad k = 1, 2, \ldots , \tag{6.3.44}$$

where: $(\mathbf{A} - \alpha\mathbf{I})^{-1}\mathbf{z}^{(k-1)} \hat{=}$ solve $(\mathbf{A} - \alpha\mathbf{I})\mathbf{w} = \mathbf{z}^{(k-1)}$ based on Gaussian elimination ($\leftrightarrow$ a single LU-factorization of $\mathbf{A} - \alpha\mathbf{I}$ as in Code 6.3.41).

What if "by accident" $\alpha \in \sigma(\mathbf{A})$ ($\Leftrightarrow \mathbf{A} - \alpha\mathbf{I}$ singular) **?**

Stability of Gaussian elimination/LU-factorization ($\rightarrow$ Sect. 2.5.3) will ensure that "$\mathbf{w}$ from (6.3.44) points in the right direction"

In other words, roundoff errors may badly affect the length of the solution $\mathbf{w}$, but not its direction.

Practice: If, in the course of Gaussian elimination/LU-factorization a zero pivot element is really encountered, then we just *replace it with* `eps`, in order to avoid `inf` values!

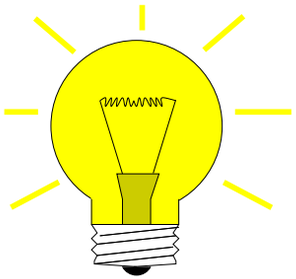
Thm. 6.3.19 ➢ Convergence of shifted inverse iteration for $\mathbf{A}^H = \mathbf{A}$:

Asymptotic linear convergence, Rayleigh quotient $\to \lambda_j$ with rate

$$\frac{|\lambda_j - \alpha|}{\min\{|\lambda_i - \alpha|, i \neq j\}} \quad \text{with} \quad \lambda_j \in \sigma(\mathbf{A}) \,, \quad |\alpha - \lambda_j| \leq |\alpha - \lambda| \quad \forall \lambda \in \sigma(\mathbf{A}) \,.$$

Extremely fast for $\alpha \approx \lambda_j$!

△



Idea: A posteriori adaptation of shift

Use $\alpha := \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})$ in k-th step of inverse iteration.

Algorithm 6.3.45 (Rayleigh quotient iteration). → [29, Alg. 25.2]]

Code 6.3.46: 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)
```

```
4  for i=1:maxit
5    z = (A-alpha*speye(n))\z;  % z^(k+1) = (A - ρ_A(z^(k))I)^-1 x^(k)
6    z = z/norm(z); lmin=dot(A*z,z);  % Computation of ρ_A(z^(k+1))
7    if (abs(alpha-lmin) < tol*lmin)  % Desired relative accuracy reached ?
8      break; end;
9    alpha = lmin;
10 end
```

Line 5: note use of `speye` to preserve spare matrix data format!

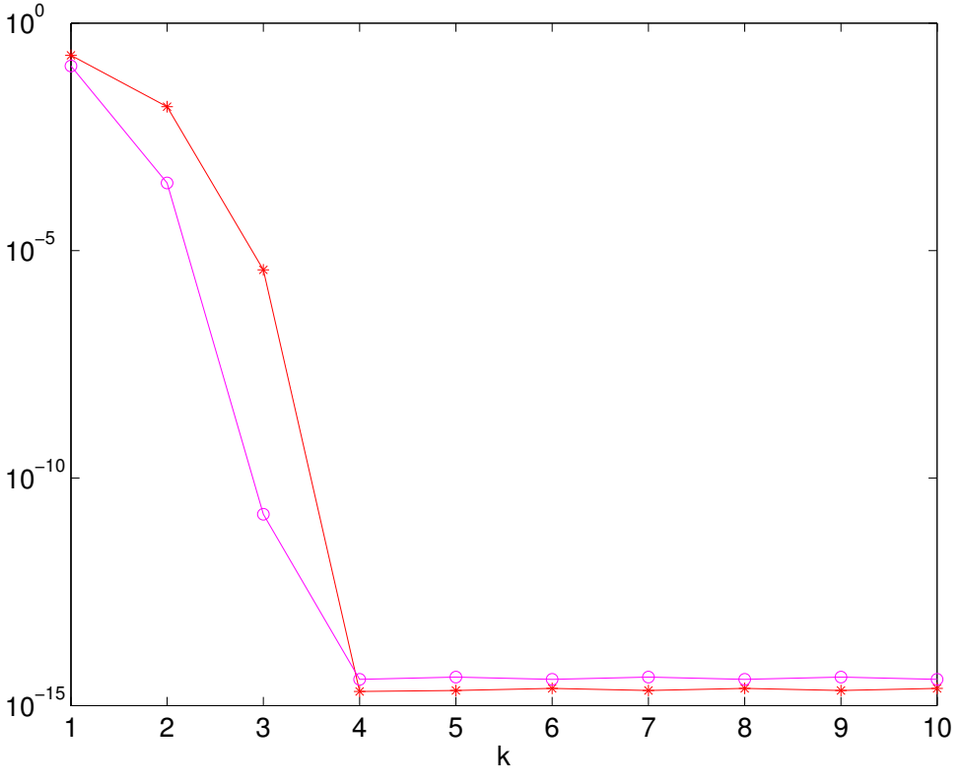
- Drawback compared with Code 6.3.41: reuse of LU-factorization no longer possible.
- Even if LSE nearly singular, stability of Gaussian elimination guarantees correct direction of $\mathbf{z}$, see discussion in Rem. 6.3.43.

Example 6.3.47 (Rayleigh quotient iteration).

R. Hiptmair

rev 30401, January 28, 2011

Monitored: iterates of Rayleigh quotient iteration (6.3.45) for s.p.d. $\mathbf{A} \in \mathbb{R}^{n,n}$




```
d = (1:10)';
n = length(d);
Z = diag(sqrt(1:n),0)+ones(n,n);
[Q,R] = qr(Z);
A = Q*diag(d,0)*Q';
```

o : $|\lambda_{\min} - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})|$

$*$: $\left\|\mathbf{z}^{(k)} - \mathbf{x}_j\right\|$, $\lambda_{\min} = \lambda_j$, $\mathbf{x}_j \in \mathrm{Eig}_{\mathbf{A}}(\lambda_j)$,

: $\left\|\mathbf{x}_j\right\|_2 = 1$

R. Hiptmair

rev 30401, January 28, 2011

k	$\lvert\lambda_{\min} - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})\rvert$	$\left\lVert\mathbf{z}^{(k)} - \mathbf{x}_j\right\rVert$
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 6.3.48. → *[29, Thm. 25.4]*
If $\mathbf{A} = \mathbf{A}^H$, *then* $\rho_{\mathbf{A}}(\mathbf{z}^{(k)})$ *converges* locally of order 3 *(→ Def. 4.1.14) to the smallest eigenvalue (in modulus), when* $\mathbf{z}^{(k)}$ *are generated by the Rayleigh quotient iteration* (6.3.45).

◇

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 (→ Code 6.3.41) and Rayleigh quotient iteration (6.3.45).

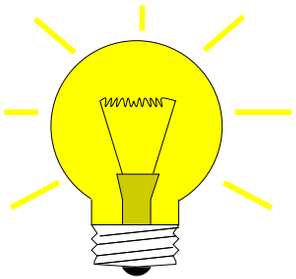
? What if direct solution of $\mathbf{Ax} = \mathbf{b}$ not feasible ?

This can happen, in case

- for large sparse $\mathbf{A}$ the amount of fill-in exhausts memory, despite sparse elimination techniques ($\rightarrow$ Sect. 2.6.3),
- $\mathbf{A}$ is available only through a routine `evalA(x)` providing $\mathbf{A}\times$vector.

We expect that an approximate solution of the linear systems of equations encountered during inverse iteration should be sufficient, because we are dealing with approximate eigenvectors anyway.

Thus, iterative solvers for solving $\mathbf{A}\mathbf{w} = \mathbf{z}^{(k-1)}$ may be considered, see Sect. 5. However, the required accuracy is not clear a priori. Here we examine an approach that completely dispenses with an iterative solver and uses a *preconditioner* ($\rightarrow$ Def. 5.3.3) instead.



Idea: (for inverse iteration without shift, $\mathbf{A} = \mathbf{A}^H$ s.p.d.)

Instead of solving $\mathbf{A}\mathbf{w} = \mathbf{z}^{(k-1)}$ compute $\mathbf{w} = \mathbf{B}^{-1}\mathbf{z}^{(k-1)}$ with "inexpensive" s.p.d. approximate inverse $\mathbf{B}^{-1} \approx \mathbf{A}^{-1}$

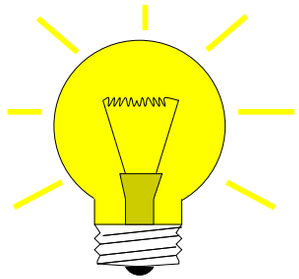
$\succ$ $\mathbf{B} \hat{=}$ Preconditioner for $\mathbf{A}$, see Def. 5.3.3

R. Hiptmair

rev 30401, January 28, 2011

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

Natural residual quantity for eigenvalue problem $\mathbf{Ax} = \lambda\mathbf{x}$:

$$\mathbf{r} := \mathbf{Az} - \rho_{\mathbf{A}}(\mathbf{z})\mathbf{z} \quad , \quad \rho_{\mathbf{A}}(\mathbf{z}) = \text{Rayleigh quotient} \to \text{Def. 6.3.15 .}$$

Note: only *direction* of $\mathbf{A}^{-1}\mathbf{z}$ matters in inverse iteration (6.3.44)

$$(\mathbf{A}^{-1}\mathbf{z}) \parallel (\mathbf{z} - \mathbf{A}^{-1}(\mathbf{Az} - \rho_{\mathbf{A}}(\mathbf{z})\mathbf{z})) \quad \Rightarrow \quad \text{defines same next iterate!}$$

[Preconditioned inverse iteration (PINVIT) for s.p.d. $\mathbf{A}$]

R. Hiptmair

rev 30401, January 28, 2011

$\mathbf{z}^{(0)}$ arbitrary,

$$\mathbf{w} = \mathbf{z}^{(k-1)} - \mathbf{B}^{-1}(\mathbf{A}\mathbf{z}^{(k-1)} - \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{z}^{(k-1)}) \,,$$
$$\mathbf{z}^{(k)} = \frac{\mathbf{w}}{\|\mathbf{w}\|_2} \,, \qquad k = 1, 2, \ldots \,. \quad (6.3.49)$$

Code 6.3.50: preconditioned inverse iteration (6.3.49)

```
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 (6.3.49)
  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 6.3.51 (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;
```

Monitored: error decay during iteration of Code 6.3.49: $|\rho_{\mathbf{A}}(\mathbf{z}^{(k)}) - \lambda_{\min}(\mathbf{A})|$

R. Hiptmair

rev 30401, January 28, 2011

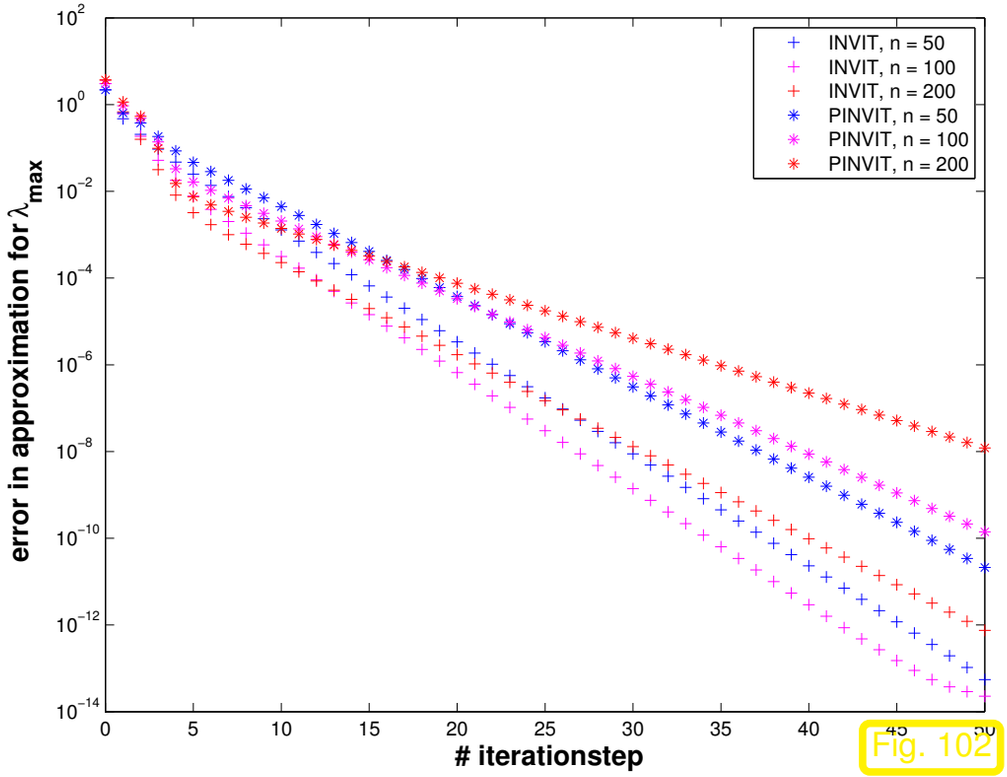


Fig. 102

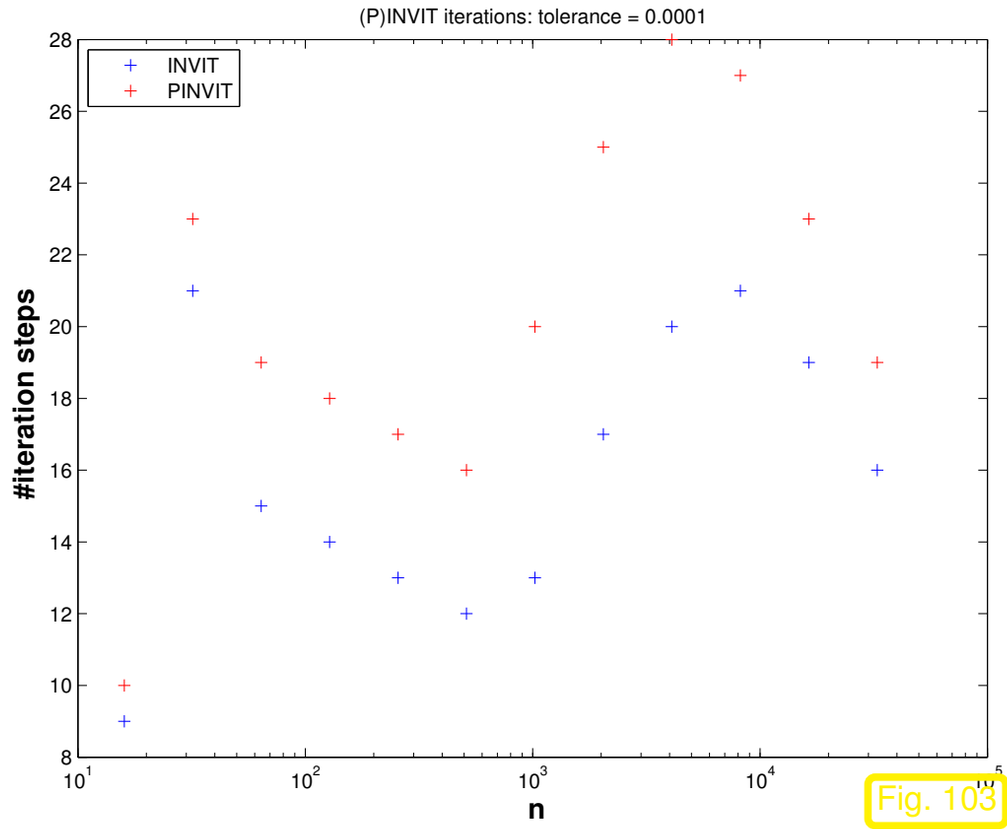


Fig. 103

R. Hiptmair

rev 30401, January 28, 2011

Observation: linear convergence of eigenvectors also for PINVIT.

◇

Theory [38, 37]:
- linear convergence of (6.3.49)
- fast convergence, if spectral condition number $\kappa(\mathbf{B}^{-1}\mathbf{A})$ small

The theory of PINVIT [38, 37] is based on the identity

$$\mathbf{w} = \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{A}^{-1}\mathbf{z}^{(k-1)} + (\mathbf{I} - \mathbf{B}^{-1}\mathbf{A})(\mathbf{z}^{(k-1)} - \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{A}^{-1}\mathbf{z}^{(k-1)}) \; . \tag{6.3.52}$$

For small residual $\mathbf{A}\mathbf{z}^{(k-1)} - \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{z}^{(k-1)}$ PINVIT almost agrees with the regular inverse iteration.

6.3.4 Subspace iterations

Task:	Compute m, $m \ll n$, of the largest/smallest (in modulus) eigenvalues of $\mathbf{A} = \mathbf{A}^H \in \mathbb{C}^{n,n}$ and associated eigenvectors.

Preliminary considerations (in $\mathbb{R}$):

According to Cor. 6.1.9: For $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$ there is a factorization $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^T$ with $\mathbf{D} =$

$\mathrm{diag}(\lambda_1,\ldots,\lambda_n)$ $\lambda_j \in \mathbb{R}$ $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ $\mathbf{U}$ $\mathbf{u}_j := (\mathbf{U})_{:,j}$ $j = 1,\ldots,n$ $\mathbf{A}$

$$0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-2} < \lambda_{n-1} < \lambda_n$$

direct power iteration 6.3.11

5.2.11

Code 6.3.53: one step of subspace power iteration, $m = 2$

```
1 function [v,w] = sspowitstep(A,v,w)
2 v = A*v; w = A*w; % "power iteration", cf. (6.3.11)
3 % orthogonalization, cf. Gram-Schmidt orthogonalization (5.2.11)
4 v = v/norm(v); w = w - dot(v,w)*v; w = w/norm(w); % now w ⊥ v
```


R. Hiptmair

rev 30401, January 28, 2011

Analysis through eigenvector expansions ($\mathbf{v},\mathbf{w}\in\mathbb{R}^n$, $\|\mathbf{v}\|_2=\|\mathbf{w}\|_2=1$)

$$\mathbf{v}=\sum_{i=1}^{n}\alpha_j\mathbf{u}_j \quad , \quad \mathbf{w}=\sum_{i=1}^{n}\beta_j\mathbf{u}_j \;,$$

$$\Rightarrow \quad \mathbf{A}\mathbf{v}=\sum_{i=1}^{n}\lambda_j\alpha_j\mathbf{u}_j \quad , \quad \mathbf{A}\mathbf{w}=\sum_{i=1}^{n}\lambda_j\beta_j\mathbf{u}_j \;,$$

$$\mathbf{v}_0:=\frac{\mathbf{v}}{\|\mathbf{v}\|_2}=\Big(\sum_{i=1}^{n}\lambda_j^2\alpha_j^2\Big)^{-1/2}\sum_{i=1}^{n}\lambda_j\alpha_j\mathbf{u}_j \;,$$

$$\mathbf{A}\mathbf{w}-(\mathbf{v}_0^T\mathbf{A}\mathbf{w})\mathbf{v}_0=\sum_{i=1}^{n}\left(\beta_j-\Big(\sum_{i=1}^{n}\lambda_j^2\alpha_j\beta_j/\sum_{i=1}^{n}\lambda_j^2\alpha_j^2\Big)\alpha_j\right)\lambda_j\mathbf{u}_j \;.$$

We notice that $\mathbf{v}$ is just mapped to the next iterate in the regular direct power iteration (6.3.11). After many steps, it will be very close to $\mathbf{u}_n$, and, therefore, we may now assume $\mathbf{v}=\mathbf{u}_n \Leftrightarrow \alpha_j=\delta_{j,n}$ (Kronecker symbol).

$$\mathbf{z}:=\mathbf{A}\mathbf{w}-(\mathbf{v}_0^T\mathbf{A}\mathbf{w})\mathbf{v}_0=0\cdot\mathbf{u}_n+\sum_{i=1}^{n-1}\lambda_j\beta_j\mathbf{u}_j \;,$$

R. Hiptmair

rev 30401, January 28, 2011

$$\mathbf{w}^{(\mathrm{new})} := \frac{\mathbf{z}}{\|\mathbf{z}\|_2} = \Big(\sum_{i=1}^{n-1} \lambda_j^2 \beta_j^2\Big)^{-1/2} \sum_{i=1}^{n-1} \lambda_j \beta_j \mathbf{u}_j \ .$$

The sequence $\mathbf{w}^{(k)}$ produced by repeated application of the mapping given by Code 6.3.52 asymptotically (that is, when $\mathbf{v}^{(k)}$ has already converged to $\mathbf{u}_n$) agrees with the sequence produced by the direct power method for $\widetilde{\mathbf{A}} := \mathbf{U}\,\mathrm{diag}(\lambda_1, \ldots, \lambda_{n-1}, 0)$. Its convergence will be governed by the relative gap $\lambda_{n-2}/\lambda_{n-1}$, see Thm. 6.3.19.

However: if $\mathbf{v}^{(k)}$ itself converges slowly, this reasoning does not apply.

Example 6.3.54 (Subspace power iteration with orthogonal projection).

☞ construction of matrix $\mathbf{A} = \mathbf{A}^T$ as in Ex. 6.3.47

Code 6.3.55: power iteration with orthogonal projection for two vectors

```
1 function sppowitdriver(d,maxit)
2 % monitor power iteration with orthogonal projection for finding
3 % the two largest (in modulus) eigenvalues and associated eigenvectors
4 % of a symmetric matrix with prescribed eigenvalues passed in d
```

```
if (nargin < 10), maxit = 20; end
if (nargin < 1), d = (1:10)'; end
% Generate matrix
n = length(d);
Z = diag(sqrt(1:n),0) + ones(n,n);
[Q,R] = qr(Z);    % generate orthogonal matrix
A = Q*diag(d,0)*Q'; % A = A^T with spectrum σ(A) = {d_1,...,d_n}
% Compute "exact" eigenvectors and eigenvalues
[V,D] = eig(A); [d,idx] = sort(diag(D)),
v_ex = V(:,idx(n)); w_ex = V(:,idx(n-1));
lv_ex = d(n); lw_ex = d(n-1);

v = ones(n,1); w = (-1).^v; % (Arbitrary) initial guess for eigenvectors
v = v/norm(v); w = w/norm(w);
result = [];
for k=1:maxit
  v_new = A*v; w_new = A*w; % "power iteration", cf. (6.3.11)
  % Rayleigh quotients provide approximate eigenvalues
  lv = dot(v_new,v); lw = dot(w_new,w);
  % orthogonalization, cf. Gram-Schmidt orthogonalization (5.2.11): w ⊥ v
  v = v_new/norm(v_new); w = w_new - dot(v,w_new)*v; w = w/norm(w);
  % Record errors in eigenvalue and eigenvector approximations. Note that the
  % direction of the eigenvectors is not specified.
```

R. Hiptmair

rev 30401, January 28, 2011

```
  result = [result; k, abs(lv-lv_ex), abs(lw-lw_ex), ...
            min(norm(v-v_ex),norm(v+v_ex)),
              min(norm(w-w_ex),norm(w+w_ex))];
end

figure('name','sspowit');
semilogy(result(:,1),result(:,2),'m-+',...
     result(:,1),result(:,3),'r-*',...
     result(:,1),result(:,4),'k-^',...
     result(:,1),result(:,5),'b-p');
title('d = [0.5*(1:8),9.5,10]');
xlabel('{\bf power iteration step}','fontsize',14);
ylabel('{\bf error}','fontsize',14);
legend('error in \lambda_n','error in \lambda_n-1','error in
  v','error in w','location','northeast');
print -depsc2 '../PICTURES/sspowitcvg1.eps';

rates = result(2:end,2:end)./result(1:end-1,2:end);
figure('name','rates');
plot(result(2:end,1),rates(:,1),'m-+',...
     result(2:end,1),rates(:,2),'r-*',...
     result(2:end,1),rates(:,3),'k-^',...
     result(2:end,1),rates(:,4),'b-p');
```

```
axis([0 maxit 0.5 1]);
title('d = [0.5*(1:8),9.5,10]');
xlabel('{\bf power iteration step}','fontsize',14);
ylabel('{\bf error quotient}','fontsize',14);
legend('error in \lambda_n','error in \lambda_n-1','error in
  v','error in w','location','southeast');
print -depsc2 '../PICTURES/sspowitcvgrates1.eps';
```

$\sigma(\mathbf{A}) = \{1, 2, \ldots, 10\}$:

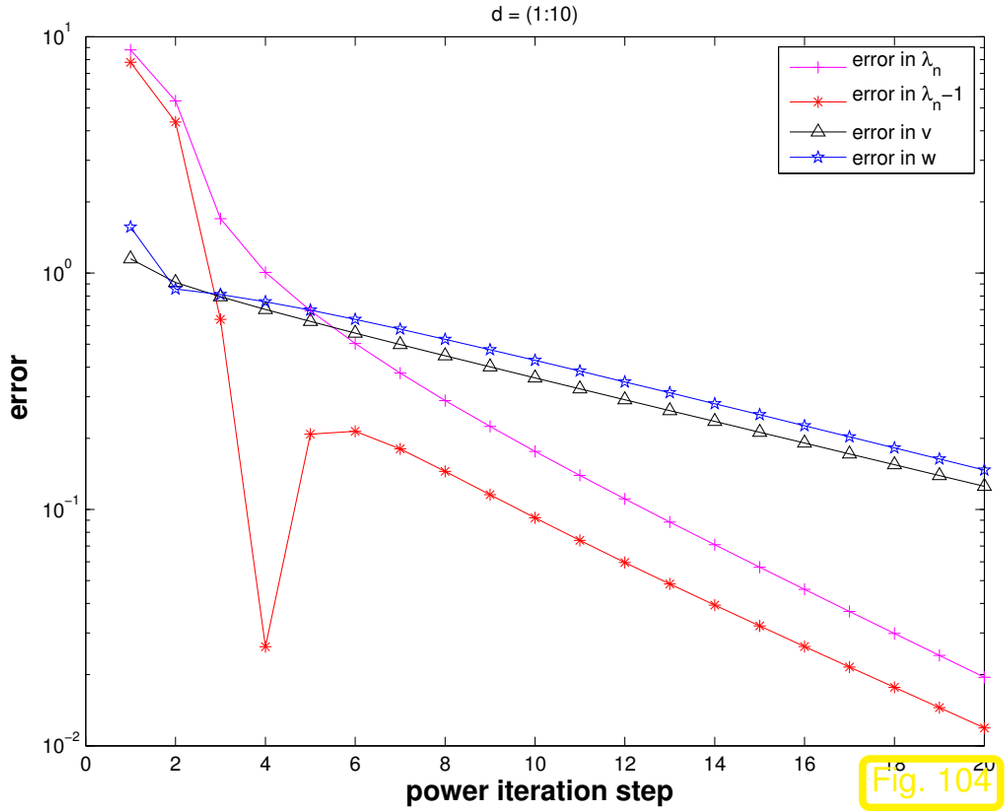


Fig. 104

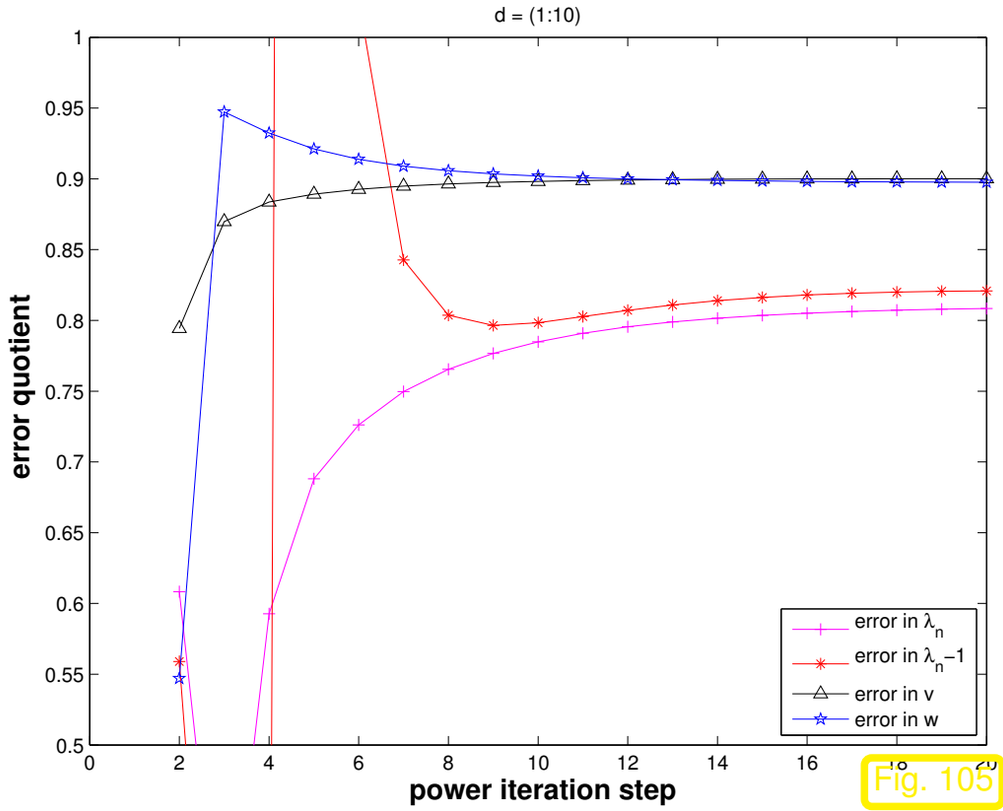


Fig. 105

R. Hiptmair

rev 30401, January 28, 2011

$\sigma(\mathbf{A}) = \{0.5, 1, \ldots, 4, 9.5, 10\}$:

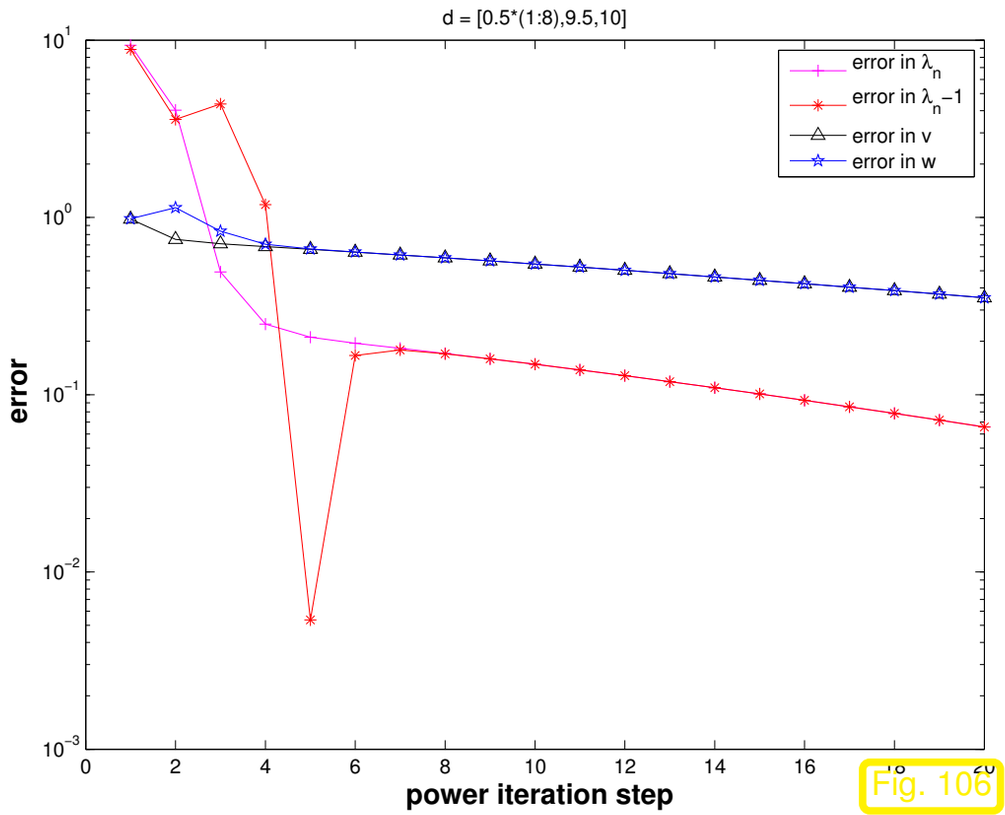


Fig. 106

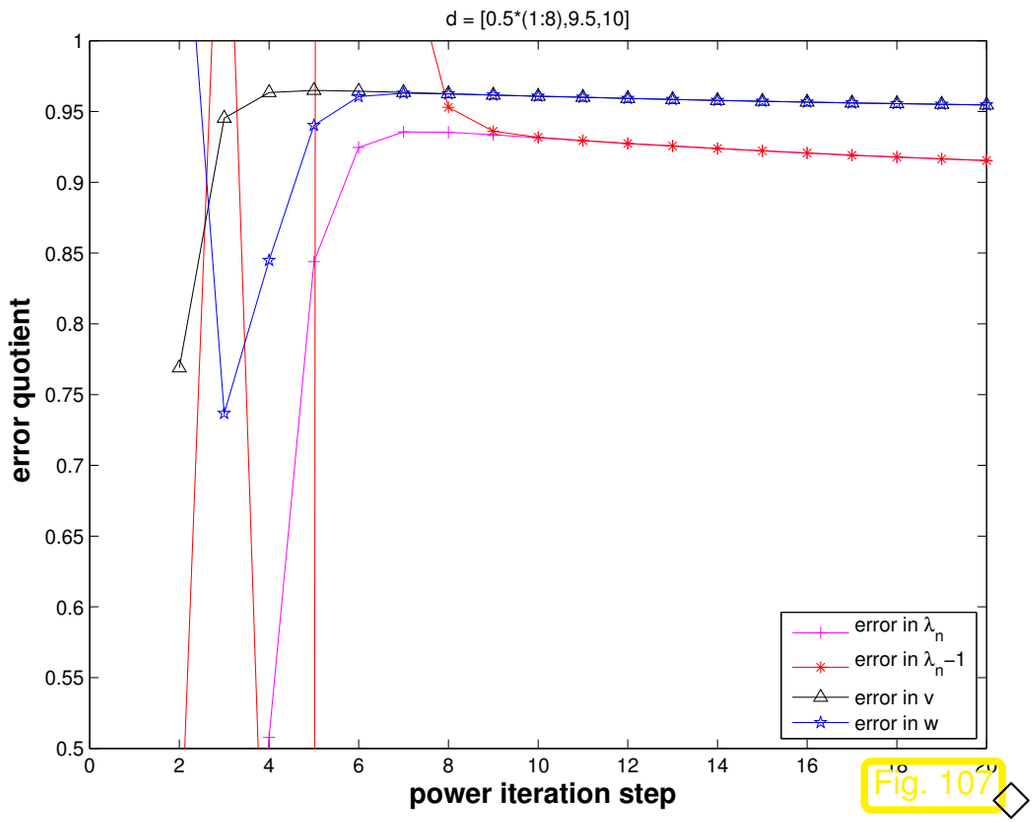


Fig. 107

Remark 6.3.56 (Generalized normalization).

The following two MATLAB code snippets perform the same function, *cf.* Code 6.3.52:

R. Hiptmair

rev 30401, January 28, 2011

```
v = v/norm(v);
w = w - dot(v,w)*v; w = w/norm(w);
```

```
[Q,R] = qr([v,w],0);
v = Q(:,1); w = Q(:,2);
```

Explanation ➢ Rem. 2.8.20

△

How to get more than two eigenvectors (belonging to largest (in modulus) eigenvalues) **?**

☞ Apply power iteration **+** subsequent orthonormalization to m vectors!

Code 6.3.57: General subspace power iteration step with 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 via QR-decomposition, see
  Rem. 2.8.20
```


R. Hiptmair

rev 30401, January 28, 2011

We revisit the above setting, Code 6.3.52. Is it possible to use the "$\mathbf{w}$-sequence" to accelerate the convergence of the "$\mathbf{v}$-sequence"?

Recall that by the min-max theorem Thm. 6.3.35

$$\mathbf{u}_n = \operatorname{argmax}_{\mathbf{x}\in\mathbb{R}^n} \rho_{\mathbf{A}}(\mathbf{x}) \quad , \quad \mathbf{u}_{n-1} = \operatorname{argmax}_{\mathbf{x}\in\mathbb{R}^n,\, \mathbf{x}\perp\mathbf{u}_n} \rho_{\mathbf{A}}(\mathbf{x}) \; . \tag{6.3.58}$$

Idea: maximize Rayleigh quotient over $\operatorname{Span}\{\mathbf{v},\mathbf{w}\}$, where $\mathbf{v}$, $\mathbf{w}$ are output by Code 6.3.52. This leads to the optimization problem

$$(\alpha^*,\beta^*) := \operatorname*{argmax}_{\alpha,\beta\in\mathbb{R},\, \alpha^2+\beta^2=1} \rho_{\mathbf{A}}(\alpha\mathbf{v}+\beta\mathbf{w}) = \operatorname*{argmax}_{\alpha,\beta\in\mathbb{R},\, \alpha^2+\beta^2=1} \rho_{(\mathbf{v},\mathbf{w})^T\mathbf{A}(\mathbf{v},\mathbf{w})}(\begin{pmatrix}\alpha\\ \beta\end{pmatrix}) \; . \tag{6.3.59}$$

Then a better approximation for the eigenvector to the largest eigenvalue is

$$\mathbf{v}^* := \alpha^*\mathbf{v} + \beta^*\mathbf{w} \; .$$

Note that $\|\mathbf{v}^*\|_2 = 1$, if both $\mathbf{v}$ and $\mathbf{w}$ are normalized, which is guaranteed in Code 6.3.52.

Then, orthogonalizing $\mathbf{w}$ w.r.t $\mathbf{v}^*$ will produce a new iterate $\mathbf{w}^*$.

Again the min-max theorem Thm. 6.3.35 tells us that we can find $(\alpha^*,\beta^*)^T$ as eigenvector to the largest eigenvalue of

$$(\mathbf{v},\mathbf{w})^T\mathbf{A}(\mathbf{v},\mathbf{w})\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \lambda\begin{pmatrix}\alpha\\ \beta\end{pmatrix} \; . \tag{6.3.60}$$

Since eigenvectors of symmetric matrices are mutually orthogonal, we find $\mathbf{w}^* = \alpha_2\mathbf{v} + \beta_2\mathbf{w}$, where $(\alpha_2, \beta_2)^T$ is the eigenvector of (6.3.60) belonging to the smallest eigenvalue. This assumes orthonormal vectors $\mathbf{v}$, $\mathbf{w}$.

Summing up the following MATLAB-function performs these computations:

Code 6.3.61: one step of subspace power iteration, $m = 2$, with Ritz projection

```
function [v,w] = sspowitsteprp(A,v,w)
v = A*v; w = A*w; % "Power iteration", cf. (6.3.11)
[Q,R] = qr([v,w],0); % Orthogonalization, cf. Rem. 2.8.20
[U,D] = eig(Q'*A*Q); % Solve Ritz projected eigenvalue problem
[ev,idx] = sort(abs(diag(D)));
w = Q*U(:,idx(1)); v = Q*U(:,idx(2)); % Recover approximate eigenvectors
```

Example 6.3.62 (Power iteration with Ritz projection).

Code 6.3.63: Main loop: power iteration with Ritz projection for two eigenvectors

```
% See Code 6.3.54 for generation of matrix A and output
for k=1:maxit
```


R. Hiptmair

rev 30401, January 28, 2011

```
  v_new = A*v; w_new = A*w; % "power iteration", cf. (6.3.11)
  [Q,R] = qr([v_new,w_new],0); % orthogonalization, cf. Rem. 2.8.20
  [U,D] = eig(Q'*A*Q); % Solve Ritz projected eigenvalue problem
  [ev,idx] = sort(abs(diag(D))), % Sort eigenvalues
  w = Q*U(:,idx(1)); v = Q*U(:,idx(2)); % Recover approximate
    eigenvectors

  % Record errors in eigenvalue and eigenvector approximations. Note that the
  % direction of the eigenvectors is not specified.
  result = [result; k, abs(ev(2)-lv_ex), abs(ev(1)-lw_ex), ...
            min(norm(v-v_ex),norm(v+v_ex)),
              min(norm(w-w_ex),norm(w+w_ex))];
end
```

Matrix as in Ex. 6.3.54, $\sigma(\mathbf{A}) = \{0.5, 1, \ldots, 4, 9.5, 10\}$:

R. Hiptmair

rev 30401, January 28, 2011

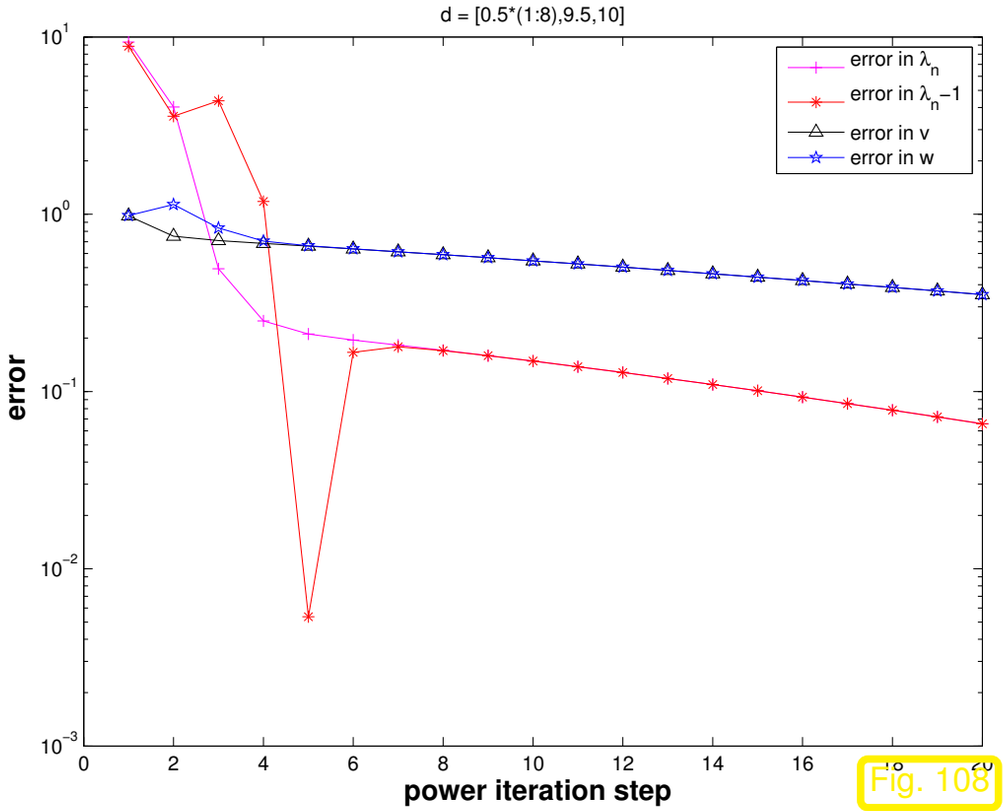


Fig. 108

simple orthonormalization, Ex. 6.3.54

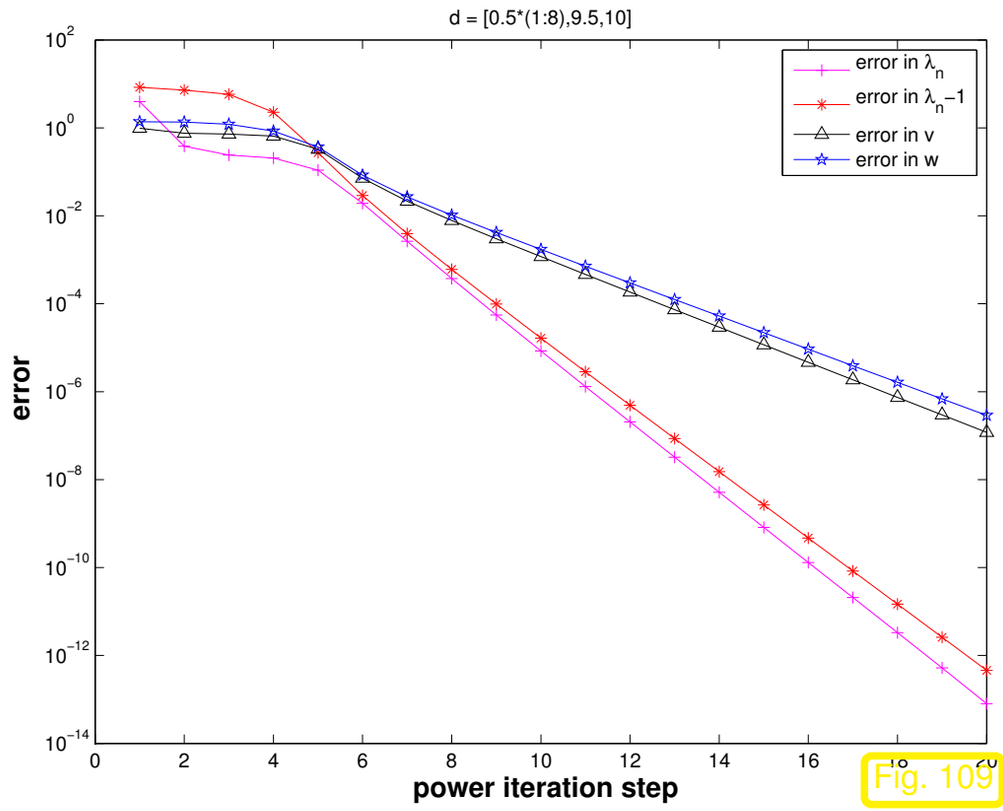


Fig. 109

with Ritz projection, Code 6.3.60

R. Hiptmair

rev 30401, January 28, 2011

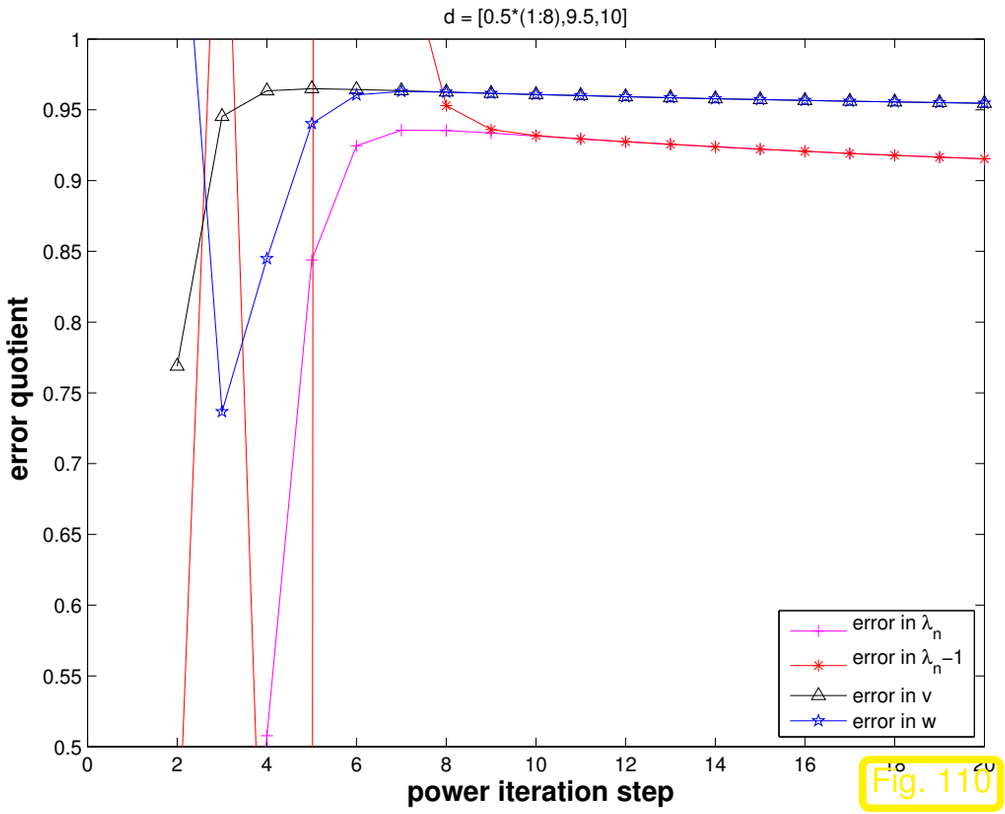


Fig. 110

simple orthonormalization, Ex. 6.3.54

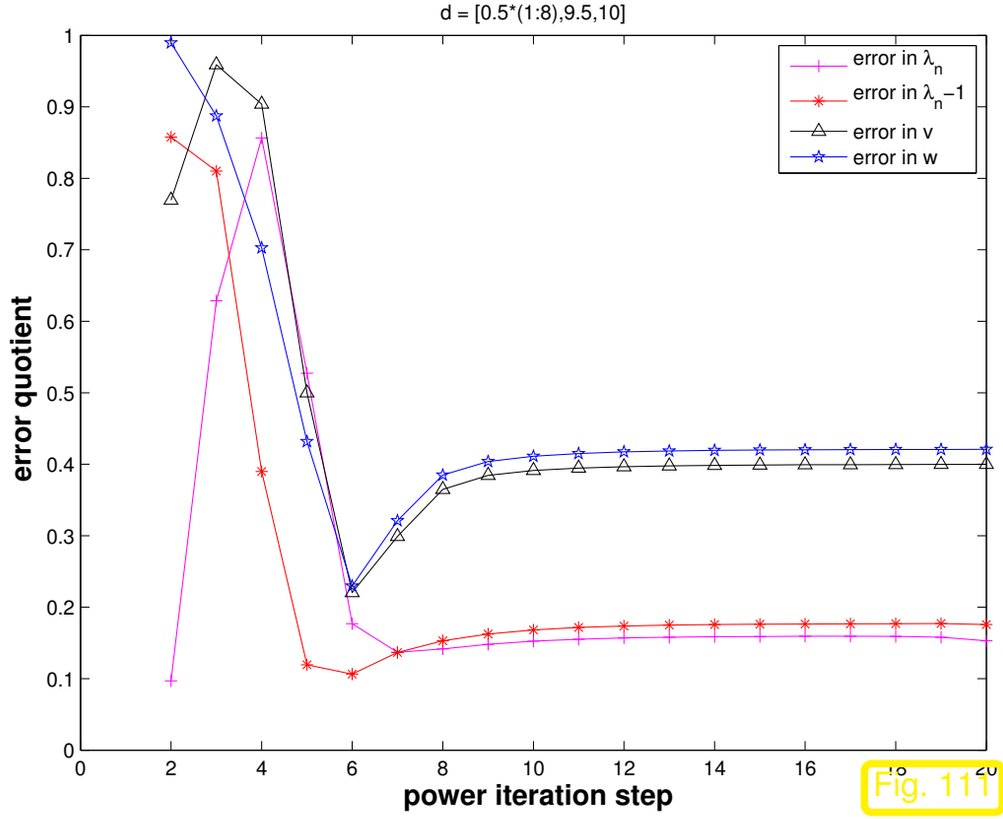


Fig. 111

with Ritz projection, Code 6.3.60

Observation: tremendous acceleration of power iteration through Ritz projection, convergence still linear but with much better rates.

R. Hiptmair

rev 30401, January 28, 2011

General technique: Ritz projection

= "projection of a (symmetric) eigenvalue problem onto a subspace"

Example: Ritz projection of $\mathbf{Ax} = \lambda\mathbf{x}$ onto $\mathrm{Span}\{\mathbf{v}, \mathbf{w}\}$:

$$(\mathbf{v}, \mathbf{w})^T \mathbf{A}(\mathbf{v}, \mathbf{w}) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \lambda (\mathbf{v}, \mathbf{w})^T (\mathbf{v}, \mathbf{w}) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} .$$

More general: Ritz projection of $\mathbf{Ax} = \lambda\mathbf{x}$ onto $\mathrm{Im}(\mathbf{V})$ (subspace spanned by columns of $\mathbf{V}$)

$$\mathbf{V}^H \mathbf{AVw} = \lambda \mathbf{V}^H \mathbf{Vw} . \tag{6.3.64}$$

If $\mathbf{V}$ is unitary, then this generalized eigenvalue problem will become a standard linear eigenvalue problem.

Note that he orthogonalization step in Code 6.3.60 is actually redundant, if exact arithmetic could be employed, because the Ritz projection could also be realized by solving the generalized eigenvalue problem

However, prior orthogonalization is essential for numerical stability ($\rightarrow$ Def. 2.5.11), *cf.* the discussion in Sect. 2.8.

Code 6.3.65: 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 Rem. 2.8.20
[U,D] = eig(Q'*A*Q);  % Solve Ritz projected m × m eigenvalue problem
V = Q*U;                    % recover approximate eigenvectors
ev = diag(D);               % approximate eigenvalues
```

In Code: 6.3.64: diagonal entries of $\mathbf{D}$ provide approximations of eigenvalues. Their (relative) changes can be used as a termination criterion.

Algorithm 6.3.66 (Subspace variant of direct power method with Ritz projection).

Code 6.3.67: Subspace power iteration with Ritz projection

```
function [ev,V] = sspowitrp(A,k,m,tol,maxit)
% Power iteration with Ritz projection for matrix A = A^T ∈ R^{n,n}:
% Subspace of dimension m ≤ n is used to compute the k ≤ m largest
```

```
% eigenvalues of A and associated eigenvectors.
n = size(A,1); V = eye(n,m); d = zeros(m,1); % (Arbitrary) initial
  eigenvectors
% The approximate eigenvectors are stored in the columns of V ∈ ℝ^{n,m}
for i=1:maxit
  [Q,R] = qr(A*V,0); % Power iteration and orthonormalization
  [U,D] = eig(Q'*A*Q); % Small m × m eigenvalue problem for Ritz projection
  [ev,idx] = sort(diag(D)); % eigenvalue approximations in diagonal of D
  V = Q*U; % 2nd part of Ritz projection
  if (abs(ev-d) < tol*max(abs(ev))), break; end
  d = ev;
end
ev = ev(m-k+1:end);
V = V(:,idx(m-k+1:end));
```

Example 6.3.68 (Convergence of subspace variant of direct power method).

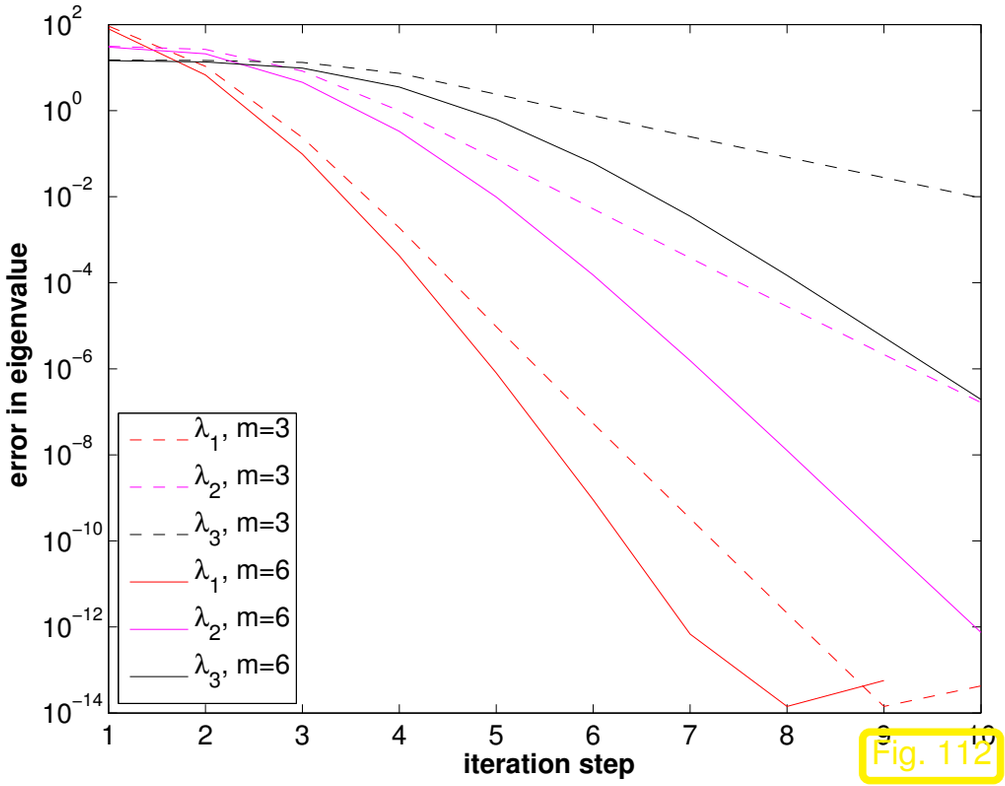


Fig. 112

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);
```

- Observation:
  linear convergence of eigenvalues
- choice $m > k$ boosts convergence of eigenvalues

◇

Remark 6.3.69 (Subspace power methods).

Analoguous to Alg. 6.3.66: construction of subspace variants of inverse iteration ($\rightarrow$ Code 6.3.41), PINVIT (6.3.49), and Rayleigh quotient iteration (6.3.45).

△

R. Hiptmair

rev 30401, November 22, 2010

6.4 Krylov Subspace Methods

All power methods ($\rightarrow$ Sect. 6.3) for the eigenvalue problem (EVP) $\mathbf{Ax} = \lambda\mathbf{x}$ only rely on the last iterate to determine the next one (1-point methods, *cf.* (4.1.2))

➢ NO MEMORY, *cf.* discussion in the beginning of Sect. 5.2.

"Memory for power iterations": pursue same idea that led from the gradient method, Alg. 5.1.11, to the conjugate gradient method, Alg. 5.2.17: use information from previous iterates to achieve efficient minimization over larger and larger subspaces.

Min-max theorem, Thm. 6.3.35 : $\mathbf{A} = \mathbf{A}^H$ $\Rightarrow$ EVPs $\Leftrightarrow$ Finding extrema/stationary points of Rayleigh quotient ($\rightarrow$ Def. 6.3.15)

Setting: EVP $\mathbf{Ax} = \lambda\mathbf{x}$ for real s.p.d. ($\rightarrow$ Def. 2.7.9) matrix $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$

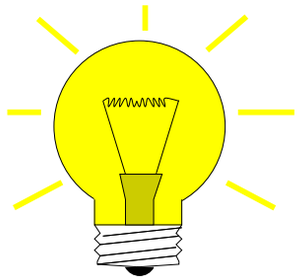
R. Hiptmair

rev 30401, November 22, 2010

notations used below: $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$: eigenvalues of $\mathbf{A}$, counted with multiplicity, see Def. 6.1.1,

$\mathbf{u}_1, \ldots, \mathbf{u}_n \hat{=}$ corresponding orthonormal eigenvectors, *cf.* Cor. 6.1.9.

$$\blacktriangleright \quad \mathbf{AU} = \mathbf{DU} \quad , \quad \mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n) \in \mathbb{R}^{n,n} \; , \quad \mathbf{D} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) \; .$$



Idea: Better $\mathbf{z}^{(k)}$ from Ritz projection onto $V := \operatorname{Span}\left\{\mathbf{z}^{(0)}, \ldots, \mathbf{z}^{(k)}\right\}$ (= space spanned by previous iterates)

R. Hiptmair

rev 30401, November 22, 2010

Recall ($\rightarrow$ Code 6.3.64) Ritz projection of an EVP $\mathbf{Ax} = \lambda\mathbf{x}$ onto a subspace $V := \operatorname{Span}\{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$, $m < n$ ➡ smaller $m \times m$ generalized EVP

$$\underbrace{\mathbf{V}^T\mathbf{AV}}_{:=\mathbf{H}}\mathbf{x} = \lambda\mathbf{V}^T\mathbf{Vx} \quad , \quad \mathbf{V} := (\mathbf{v}_1, \ldots, \mathbf{v}_m) \in \mathbb{R}^{n,m} \; . \tag{6.4.1}$$

From min-max theorem Thm. 6.3.35:

$$\mathbf{u}_n \in V \quad \Rightarrow \quad \text{largest eigenvalue of (6.4.1)} = \lambda_{\max}(\mathbf{A}) \ ,$$
$$\mathbf{u}_1 \in V \quad \Rightarrow \quad \text{smallest eigenvalue of (6.4.1)} = \lambda_{\min}(\mathbf{A}) \ .$$

Intuition: If $\mathbf{u}_n(\mathbf{u}_1)$ "well captured" by V (that is, the angle between the vector and the space V is small), then we can expect that the largest (smallest) eigenvalue of (6.4.1) is a good approximation for $\lambda_{\max}(\mathbf{A})(\lambda_{\min}(\mathbf{A}))$, and that, assuming normalization

$$\mathbf{V}\mathbf{w} \approx \mathbf{u}_1(\mathbf{u}_n) \ ,$$

where $\mathbf{w}$ is the corresponding eigenvector of (6.4.1).

For direct power method (6.3.11): $\mathbf{z}^{(k)} || \mathbf{A}^k \mathbf{z}^{(0)}$

$$V = \mathrm{Span}\left\{\mathbf{z}^{(0)}, \mathbf{A}\mathbf{z}^{(0)}, \ldots, \mathbf{A}^{(k)}\mathbf{z}^{(0)}\right\} = \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{z}^{(0)})$$ a Krylov space, → Def. 5.2.6 . (6.4.2)

R. Hiptmair

rev 30401, November 22, 2010

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.* Code 6.3.60.

Note: implementation for demonstration purposes only (inefficient for sparse matrix $\mathbf{A}$!)

Terminology: $\sigma(\mathbf{Q}^T\mathbf{A}\mathbf{Q}) \hat{=}$ Ritz values $\mu_1 \leq \mu_2 \leq \cdots \leq \mu_m$,
eigenvectors of $\mathbf{Q}^T\mathbf{A}\mathbf{Q} \hat{=}$ Ritz vectors

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; % eigenvalues 1,2,3,...,100
```


R. Hiptmair

rev 30401, November 22, 2010

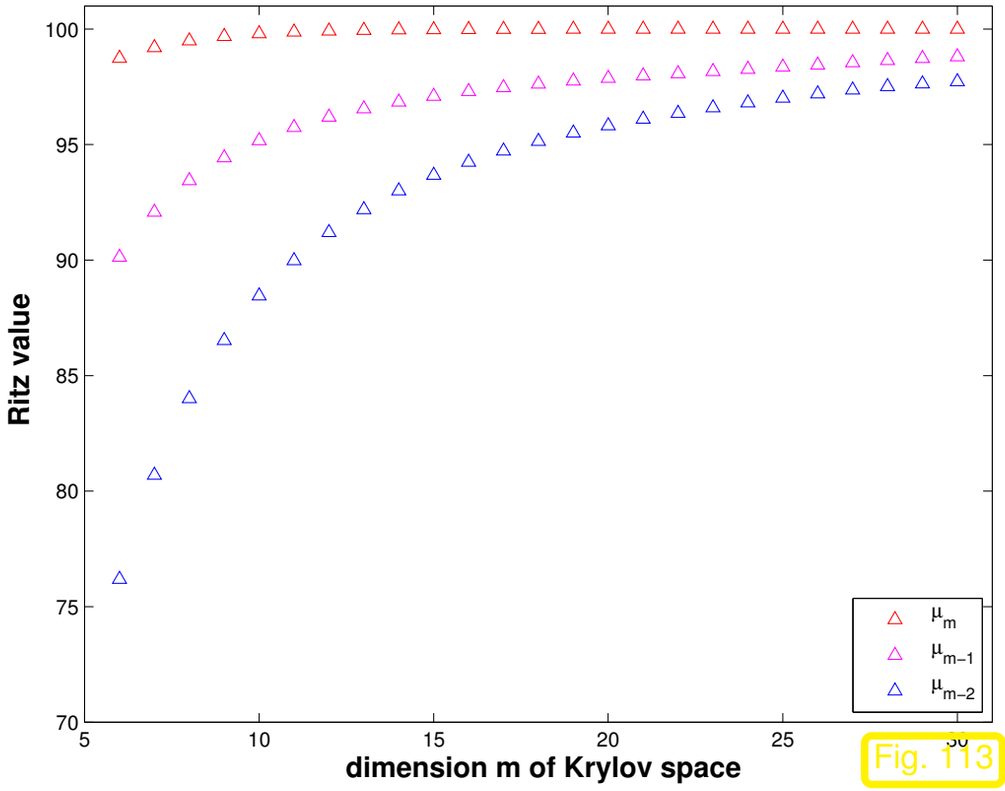


Fig. 113

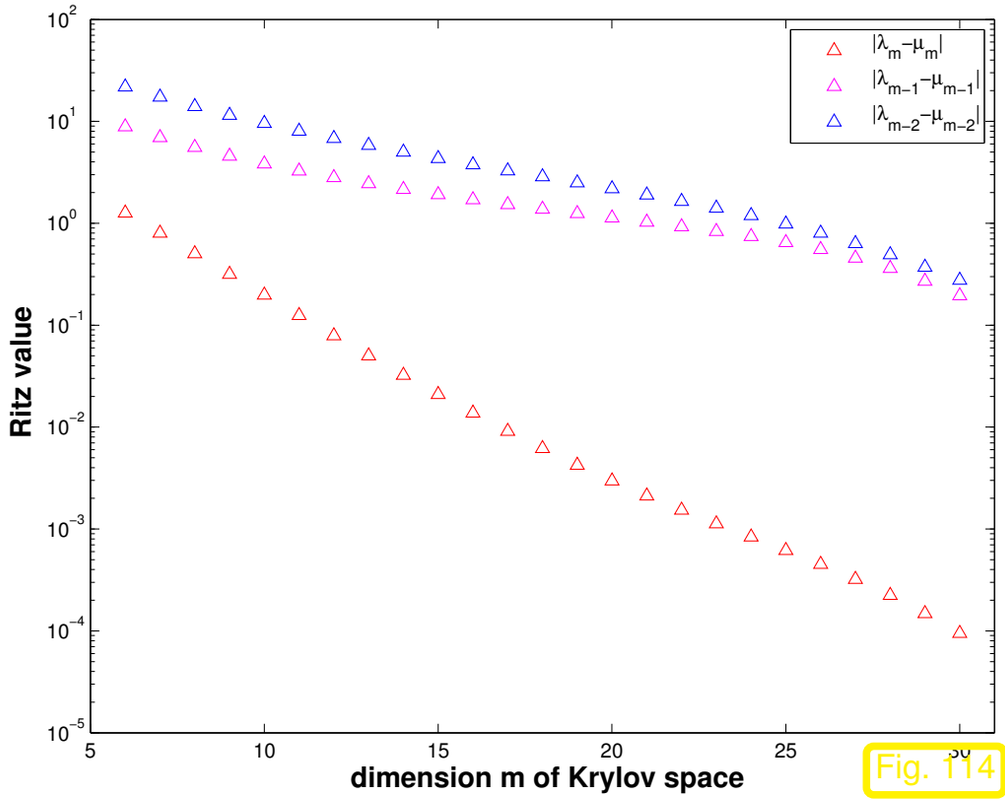


Fig. 114

R. Hiptmair

rev 30401, November 22, 2010

Observation: "vaguely linear" convergence of largest Ritz values to largest eigenvalues. Fastest convergence of largest Ritz value $\rightarrow$ largest eigenvalue of $\mathbf{A}$

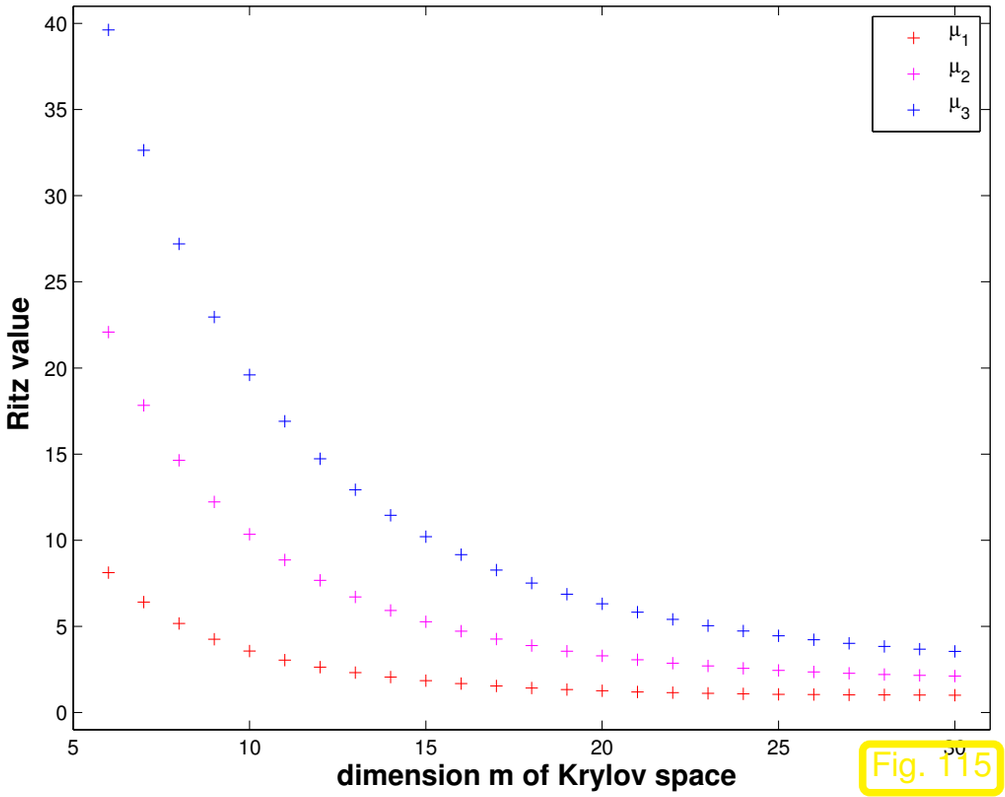


Fig. 115

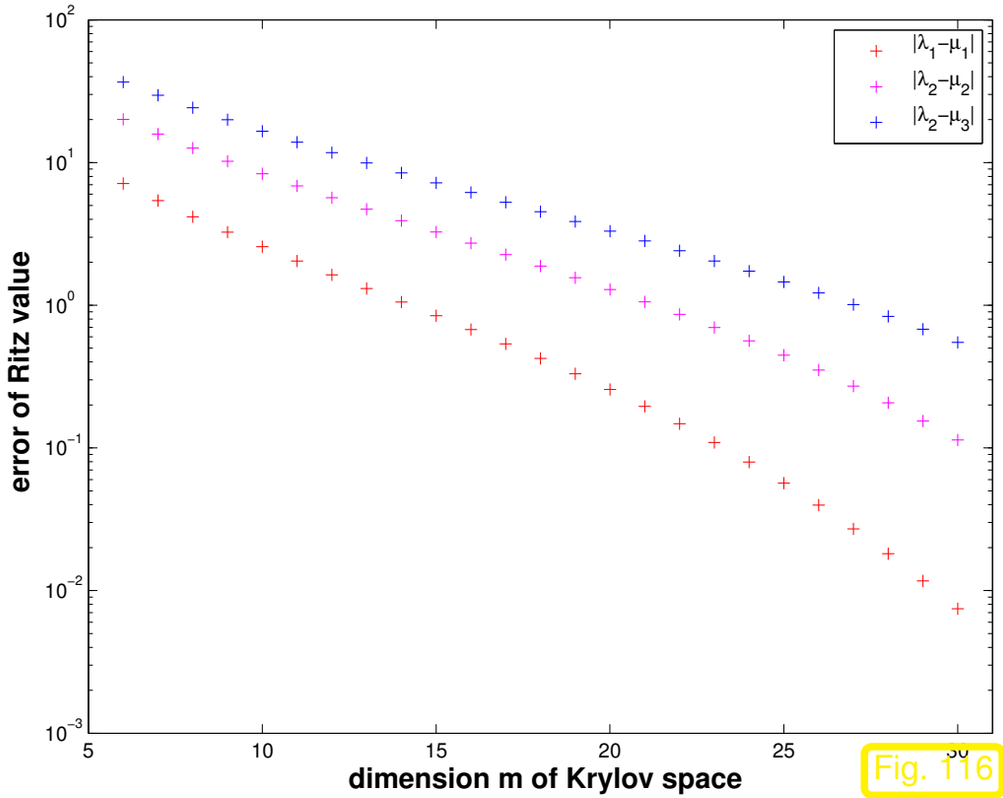


Fig. 116

R. Hiptmair

rev 30401, November 22, 2010

Observation: *Also the smallest Ritz values converge* "vaguely linearly" to the smallest eigenvalues of $\mathbf{A}$. Fastest convergence of smallest Ritz value $\rightarrow$ smallest eigenvalue of $\mathbf{A}$.

$\diamond$

? Why do smallest Ritz values converge to smallest eigenvalues of $\mathbf{A}$?

Consider direct power method (6.3.11) for $\widetilde{\mathbf{A}} := \nu\mathbf{I} - \mathbf{A}$, $\nu > \lambda_{\max}(\mathbf{A})$:

$$\mathbf{z}^{(0)} \text{ arbitrary}, \quad \widetilde{\mathbf{z}}^{(k+1)} = \frac{(\nu\mathbf{I} - \mathbf{A})\widetilde{\mathbf{z}}^{(k)}}{\left\|(\nu\mathbf{I} - \mathbf{A})\widetilde{\mathbf{z}}^{(k)}\right\|_2} \tag{6.4.5}$$

As $\sigma(\nu\mathbf{I} - \mathbf{A}) = \nu - \sigma(\mathbf{A})$ and eigenspaces agree, we infer from Thm. 6.3.19

$$\lambda_1 < \lambda_2 \quad \Rightarrow \quad \mathbf{z}^{(k)} \overset{k\to\infty}{\longrightarrow} \mathbf{u}_1 \quad \& \quad \rho_{\mathbf{A}}(\mathbf{z}^{(k)}) \overset{k\to\infty}{\longrightarrow} \lambda_1 \quad \text{linearly}. \tag{6.4.6}$$

By the binomial theorem (also applies to matrices, if they commute)

$$(\nu\mathbf{I} - \mathbf{A})^k = \sum_{j=0}^{k} \binom{k}{j} \nu^{k-j}\mathbf{A}^j \quad \Rightarrow \quad (\nu\mathbf{I} - \mathbf{A})^k\widetilde{\mathbf{z}}^{(0)} \in \mathcal{K}_k(\mathbf{A}, \mathbf{z}^{(0)}),$$

$$\boxed{\mathcal{K}_k(\nu\mathbf{I} - \mathbf{A}, \mathbf{x}) = \mathcal{K}_k(\mathbf{A}, \mathbf{x})}. \tag{6.4.7}$$

➢ $\mathbf{u}_1$ can also be expected to be "well captured" by $\mathcal{K}_k(\mathbf{A}, \mathbf{x})$ and the smallest Ritz value should provide a good aproximation for $\lambda_{\min}(\mathbf{A})$.

R. Hiptmair

rev 30401, November 22, 2010

Recall from Sect. 5.2.2 , Lemma 5.2.12:

Residuals $\mathbf{r}_0, \ldots, \mathbf{r}_{m-1}$ generated in CG iteration, Alg. 5.2.17 applied to $\mathbf{Ax} = \mathbf{z}$ with $\mathbf{x}^{(0)} = 0$, provide *orthogonal basis* for $\mathcal{K}_m(\mathbf{A}, \mathbf{z})$ (, if $\mathbf{r}_k \neq 0$).

Inexpensive Ritz projection of $\mathbf{Ax} = \lambda\mathbf{x}$ onto $\mathcal{K}_m(\mathbf{A}, \mathbf{z})$: orthogonal matrix

$$\mathbf{V}_m^T \mathbf{A} \mathbf{V}_m \mathbf{x} = \lambda \mathbf{x} \;, \quad \mathbf{V}_m := \left(\frac{\mathbf{r}_0}{\|\mathbf{r}_0\|}, \ldots, \frac{\mathbf{r}_{m-1}}{\|\mathbf{r}_{m-1}\|} \right) \in \mathbb{R}^{n,m} \;. \tag{6.4.8}$$

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.*

Proof. Lemma 5.2.12: $\{\mathbf{r}_0, \ldots, \mathbf{r}_{\ell-1}\}$ is an orthogonal basis of $\mathcal{K}_\ell(\mathbf{A}, \mathbf{r}_0)$, if all the residuals are non-zero. As $\mathbf{A}\mathcal{K}_{\ell-1}(\mathbf{A}, \mathbf{r}_0) \subset \mathcal{K}_\ell(\mathbf{A}, \mathbf{r}_0)$, we conclude the orthogonality $\mathbf{r}_m^T \mathbf{A} \mathbf{r}_j$ for all $j = 0, \ldots, m-2$. Since

$$\left(\mathbf{V}_m^T \mathbf{A} \mathbf{V}_m \right)_{ij} = \mathbf{r}_{i-1}^T \mathbf{A} \mathbf{r}_{j-1} \;, \quad 1 \leq i, j \leq m \;,$$

the assertion of the theorem follows. □

$$\mathbf{V}_l^H \mathbf{A}\mathbf{V}_l = \begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & & \\ & & & & \ddots & \\ & & & \ddots & \ddots & \beta_{k-1} \\ & & & & \beta_{k-1} & \alpha_k \end{pmatrix} =: \mathbf{T}_l \in \mathbb{K}^{k,k} \quad \text{[tridiagonal matrix]} \qquad (6.4.10)$$

Code 6.4.11: Lanczos process, *cf.* Code 5.2.17

```
1  function [V,alph,bet] = lanczos(A,k,z0)
2  % Note:  this implementation of the Lanczos process
     also records the orthonormal CG residuals in the
     columns of the matrix V, which is not needed when
     only eigenvalue approximations are desired.
3  V = z0/norm(z0);
4  % Vectors storing entries of tridiagonal matrix (6.4.10
5  alph=zeros(k,1); bet = zeros(k,1);
6  for j=1:k
7    q = A*V(:,j); alph(j) = dot(q,V(:,j));
8    w = q - alph(j)*V(:,j);
9    if (j > 1), w = w - bet(j-1)*V(:,j-1); end
10   bet(j) = norm(w); V = [V,w/bet(j)];
11 end
12 bet = bet(1:end-1);
```

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

Total computational effort for l steps of Lanczos process, if $\mathbf{A}$ has at most k non-zero entries per row: $O(nkl)$

Note: Code 6.4.10 assumes that no residual vanishes. This could happen, if $\mathbf{z}_0$ exactly belonged to the span of a few eigenvectors. However, in practical computations inevitable round-off errors will always ensure that the iterates do not stay in an invariant subspace of $\mathbf{A}$, *cf.* Rem. 6.3.20.

Convergence (what we expect from the above considerations) → [14, Sect. 8.5])

$$\text{In } l\text{-th step:} \quad \lambda_n \approx \mu_l^{(l)} \,,\, \lambda_{n-1} \approx \mu_{l-1}^{(l)} \,, \ldots, \lambda_1 \approx \mu_1^{(l)} \,,$$

$$\sigma(\mathbf{T}_l) = \{\mu_1^{(l)}, \ldots, \mu_l^{(l)}\}, \quad \mu_1^{(l)} \leq \mu_2^{(l)} \leq \cdots \leq \mu_l^{(l)} \,.$$

Example 6.4.12 (Lanczos process for eigenvalue computation).

$\mathbf{A}$ from Ex. 6.4.4

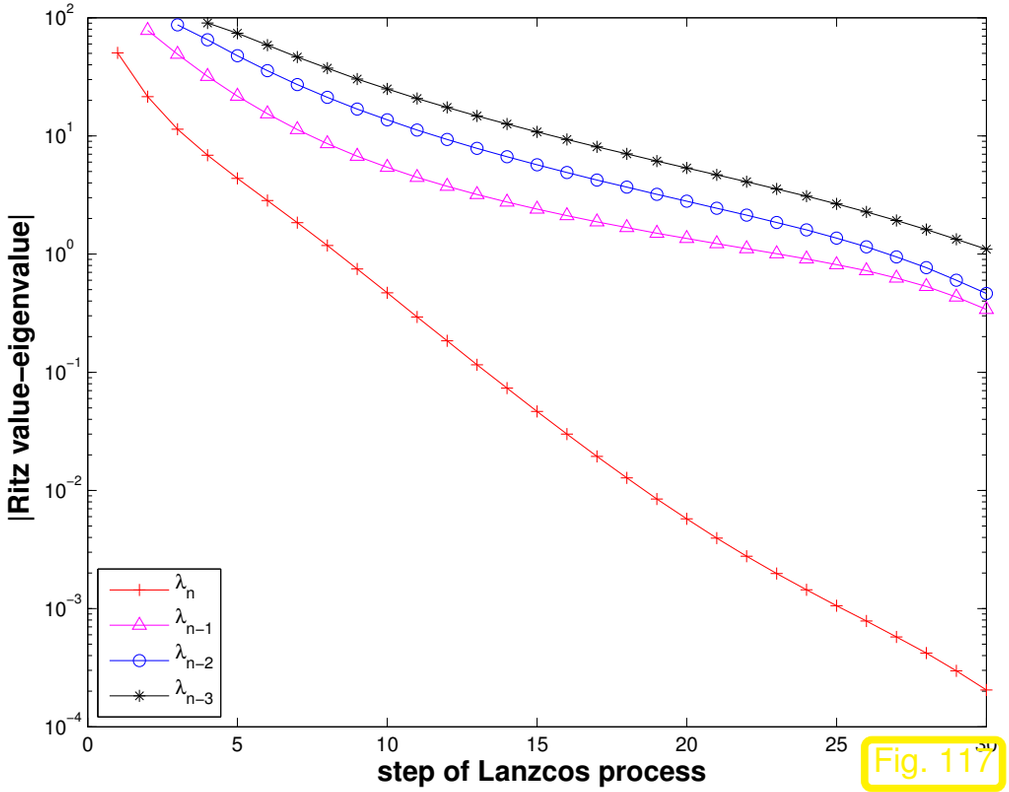


Fig. 117

```
A = gallery('minij',100);
```

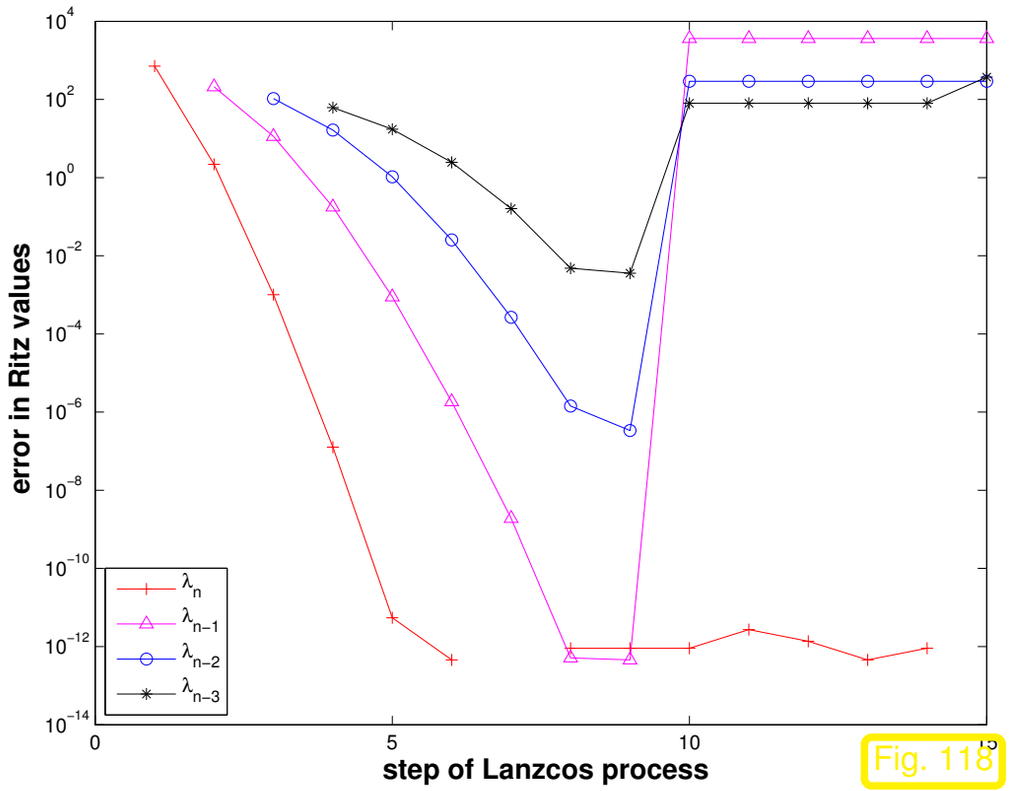


Fig. 118

Observation: same as in Ex. 6.4.4, linear convergence of Ritz values to eigenvalues.

However for $\mathbf{A} \in \mathbb{R}^{10,10}$, $a_{ij} = \min\{i,j\}$ good initial convergence, but sudden "jump" of Ritz values off eigenvalues!

Conjecture: Impact of roundoff errors, *cf.* Ex. 5.2.21

◇

Example 6.4.13 (Impact of roundoff on Lanczos process).

$$\mathbf{A} \in \mathbb{R}^{10,10} \quad , \quad a_{ij} = \min\{i,j\} \, . \qquad \texttt{A = gallery('minij',10);}$$

R. Hiptmair

rev 30401, November 22, 2010

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”, [20, Sect. 9.2.5])

$$\mathbf{V}^H\mathbf{V} = \begin{pmatrix}
1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000251 & 0.258801 & 0.883711 \\
0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000106 & 0.109470 & 0.373799 \\
0.000000 & -0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000005 & 0.005373 & 0.018347 \\
0.000000 & 0.000000 & 0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000096 & 0.000328 \\
0.000000 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000001 & 0.000003 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 & 0.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 & 0.000000 \\
0.000251 & 0.000106 & 0.000005 & 0.000000 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & -0.000000 & 0.000000 \\
0.258801 & 0.109470 & 0.005373 & 0.000096 & 0.000001 & 0.000000 & 0.000000 & -0.000000 & 1.000000 & 0.000000 \\
0.883711 & 0.373799 & 0.018347 & 0.000328 & 0.000003 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000
\end{pmatrix}$$

Loss of orthogonality of residual vectors due to roundoff

(compare: impact of roundoff on CG iteration, Ex. 5.2.21

R. Hiptmair

l	$\sigma(\mathbf{T}_l)$									
1										38.500000
2									3.392123	44.750734
3								1.117692	4.979881	44.766064
4							0.597664	1.788008	5.048259	44.766069
5						0.415715	0.925441	1.870175	5.048916	44.766069
6					0.336507	0.588906	0.995299	1.872997	5.048917	44.766069
7				0.297303	0.431779	0.638542	0.999922	1.873023	5.048917	44.766069
8			0.276160	0.349724	0.462449	0.643016	1.000000	1.873023	5.048917	44.766069
9		0.276035	0.349451	0.462320	0.643006	1.000000	1.873023	3.821426	5.048917	44.766069
10	0.263867	0.303001	0.365376	0.465199	0.643104	1.000000	1.873023	5.048917	44.765976	44.766069

◇

Idea:
- do not rely on orthogonality relations of Lemma 5.2.12
- use explicit Gram-Schmidt orthogonalization [39, Thm. 4.8], [23, Alg .6.1]

Details: inductive approach: given $\{\mathbf{v}_1,\ldots,\mathbf{v}_l\}$ ONB of $\mathcal{K}_l(\mathbf{A},\mathbf{z})$

$$\Rightarrow \quad \widetilde{\mathbf{v}}_{l+1} := \mathbf{A}\mathbf{v}_l - \sum_{j=1}^{l}(\mathbf{v}_j^H\mathbf{A}\mathbf{v}_l)\,\mathbf{v}_j\,, \quad \mathbf{v}_{l+1} := \frac{\widetilde{\mathbf{v}}_{l+1}}{\|\widetilde{\mathbf{v}}_{l+1}\|_2} \quad \Rightarrow \quad \mathbf{v}_{l+1} \perp \mathcal{K}_l(\mathbf{A},\mathbf{z})\,. \tag{6.4.14}$$

(Gram-Schmidt, *cf.* (5.2.11)) orthogonal

Arnoldi process

In step l:

$1\times$	$\mathbf{A}\times$vector
$l+1$	dot products
l	AXPY-operations
n	divisions

➢ Computational cost for l steps, if at most k non-zero entries in each row

Code 6.4.15: Arnoldi process

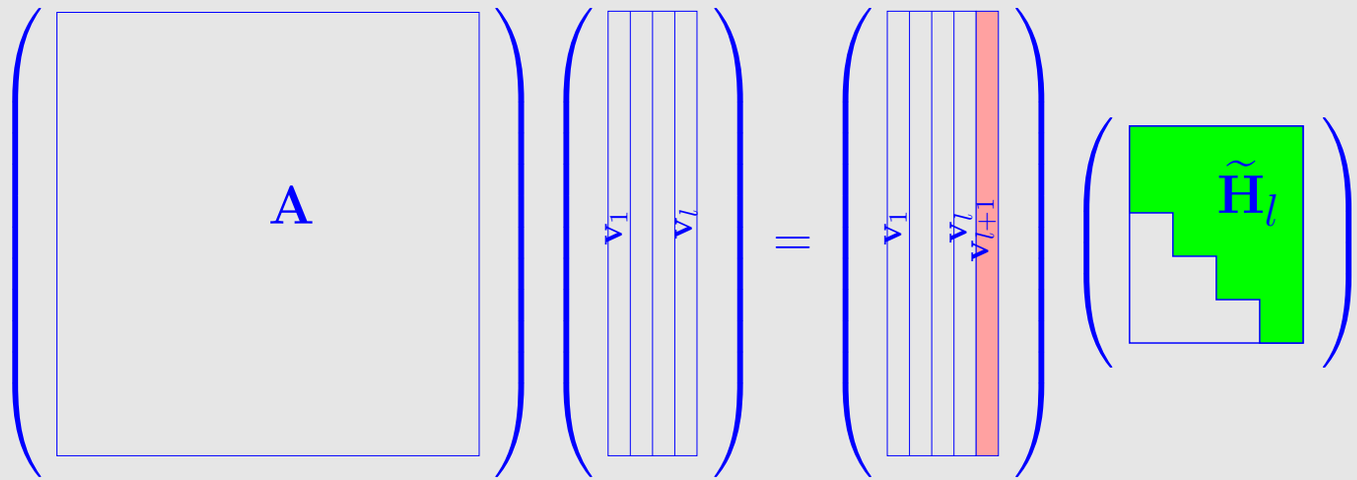
```
1  function [V,H] = arnoldi(A,k,v0)
2  % Columns of V store orthonormal basis of
     Krylov spaces K_l(A,v_0).
3  % H returns Hessenberg matrix, see
     Lemma 6.4.16.
4  V = [v0/norm(v0)];
5  H = zeros(k+1,k);
6  for l=1:k
7    vt = A*V(:,l); % Next basis vector
8    for j=1:l
9      % Gram-Schmidt, see (6.4.14)
10     H(j,l) = dot(V(:,j),vt);
```

If it does not stop prematurely, the Arnoldi process of Code 6.4.14 will yield an *orthonormal basis* (OBN) of $\mathcal{K}_{k+1}(\mathbf{A},\mathbf{v}_0)$ for a **general** $\mathbf{A} \in \mathbb{C}^{n,n}$.

Algebraic view of the Arnoldi process of Code 6.4.14, meaning of output `H`:

$$\mathbf{V}_l = \left[\mathbf{v}_1,\ldots,\mathbf{v}_l\right] : \quad \mathbf{A}\mathbf{V}_l = \mathbf{V}_{l+1}\widetilde{\mathbf{H}}_l \quad , \quad \widetilde{\mathbf{H}}_l \in \mathbb{K}^{l+1,l} \text{ mit } \widetilde{h}_{ij} = \begin{cases} \mathbf{v}_i^H \mathbf{A}\mathbf{v}_j & \text{, if } i \le j \text{ ,} \\ \|\widetilde{\mathbf{v}}_i\|_2 & \text{, if } i = j+1 \text{ ,} \\ 0 & \text{else.} \end{cases}$$

➡ $\widetilde{\mathbf{H}}_l$ = non-square upper Hessenberg matrices



R. Hiptmair

rev 30401, November 22, 2010

Translate Code 6.4.14 to matrix calculus:

Lemma 6.4.16 (Theory of Arnoldi process)**.**
For the matrices $\mathbf{V}_l \in \mathbb{K}^{n,l}$, $\widetilde{\mathbf{H}}_l \in \mathbb{K}^{l+1,l}$ arising in the l-th step, $l \leq n$, of the Arnoldi process holds

(i) $\mathbf{V}_l^H \mathbf{V}_l = \mathbf{I}$ *(unitary matrix),*

(ii) $\mathbf{A}\mathbf{V}_l = \mathbf{V}_{l+1}\widetilde{\mathbf{H}}_l$, $\widetilde{\mathbf{H}}_l$ *is non-square upper Hessenberg matrix,*

(iii) $\mathbf{V}_l^H \mathbf{A}\mathbf{V}_l = \mathbf{H}_l \in \mathbb{K}^{l,l}$, $h_{ij} = \widetilde{h}_{ij}$ *for* $1 \leq i, j \leq l$,

(iv) *If* $\mathbf{A} = \mathbf{A}^H$ *then* $\mathbf{H}_l$ *is tridiagonal (➢ Lanczos process)*

Proof. Direct from Gram-Schmidt orthogonalization and inspection of Code 6.4.14. □

R. Hiptmair

rev 30401, November 22, 2010

Remark 6.4.17 (Arnoldi process and Ritz projection)*.*

Interpretation of Lemma 6.4.16 (iii) & (i):

$\mathbf{H}_l\mathbf{x} = \lambda\mathbf{x}$ is a (generalized) Ritz projection of EVP $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$

△

Eigenvalue approximation for general EVP $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ by Arnoldi process:

$$\text{In } l\text{-th step:} \quad \lambda_n \approx \mu_l^{(l)}\,,\ \lambda_{n-1} \approx \mu_{l-1}^{(l)}\,,\ldots,\ \lambda_1 \approx \mu_1^{(l)}\,,$$

$$\sigma(\mathbf{H}_l) = \{\mu_1^{(l)},\ldots,\mu_l^{(l)}\},\quad |\mu_1^{(l)}| \le |\mu_2^{(l)}| \le \cdots \le |\mu_l^{(l)}|\,.$$

Code 6.4.18: Arnoldi eigenvalue approximation

```
function [dn,V,Ht] = arnoldieig(A,v0,k,tol)
n = size(A,1); V = [v0/norm(v0)];
H = zeros(1,0); dn = zeros(k,1);
for l=1:n
  d = dn;
  Ht = [Ht, zeros(l,1); zeros(1,l)];
  vt = A*V(:,l);
  for j=1:l
    Ht(j,l) = dot(V(:,j),vt);
    vt = vt - Ht(j,l)*V(:,j);
  end
  ev = sort(eig(Ht(1:l,1:l)));
  dn(1:min(l,k)) =
    ev(end:-1:end-min(l,k)+1);
  if (norm(d-dn) < tol*norm(dn)), break;
    end;
  Ht(l+1,l) = norm(vt);
  V = [V, vt/Ht(l+1,l)];
end
```

Arnoldi process for computing the k largest (in modulus) eigenvalues of $\mathbf{A} \in \mathbb{C}^{n,n}$

1 $\mathbf{A}\times$vector per step
($\succ$ attractive for sparse matrices)

However: required storage increases with number of steps, *cf.* situation with GMRES, Sect. 5.4.1.

Heuristic termination criterion

Example 6.4.19 (Stabilty of Arnoldi process).

$$\mathbf{A} \in \mathbb{R}^{100,100} \quad , \quad a_{ij} = \min\{i,j\} \; . \qquad \texttt{A = gallery('minij',100);}$$

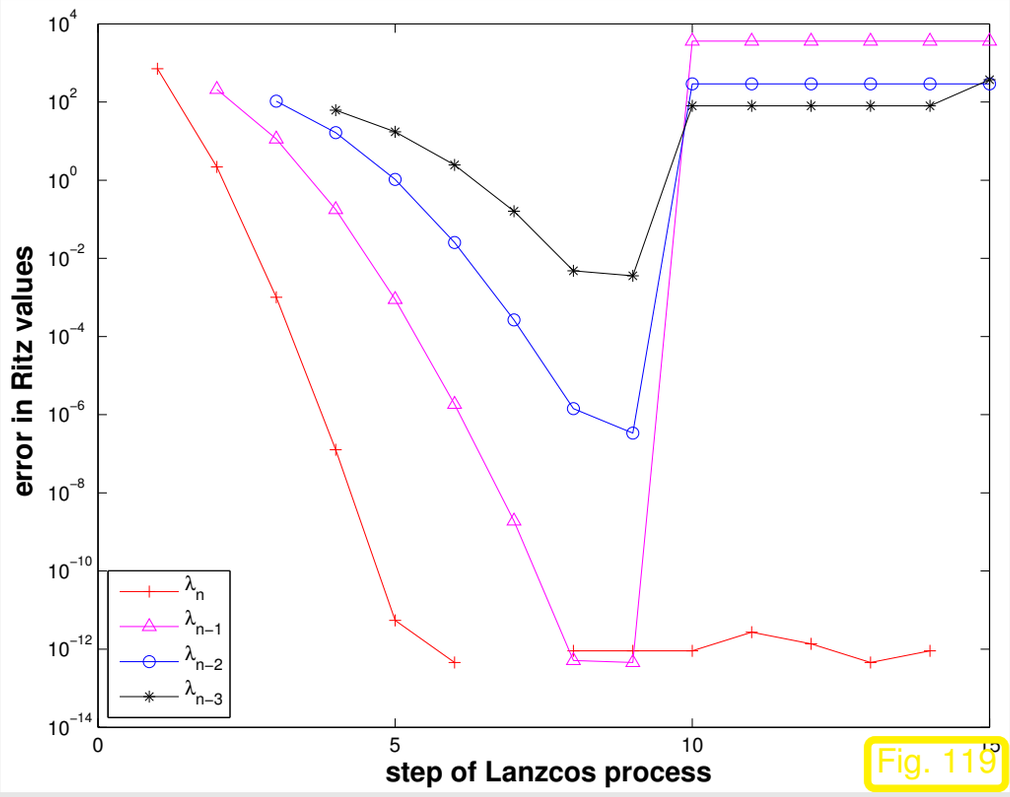


Fig. 119

Lanczos process: Ritz values

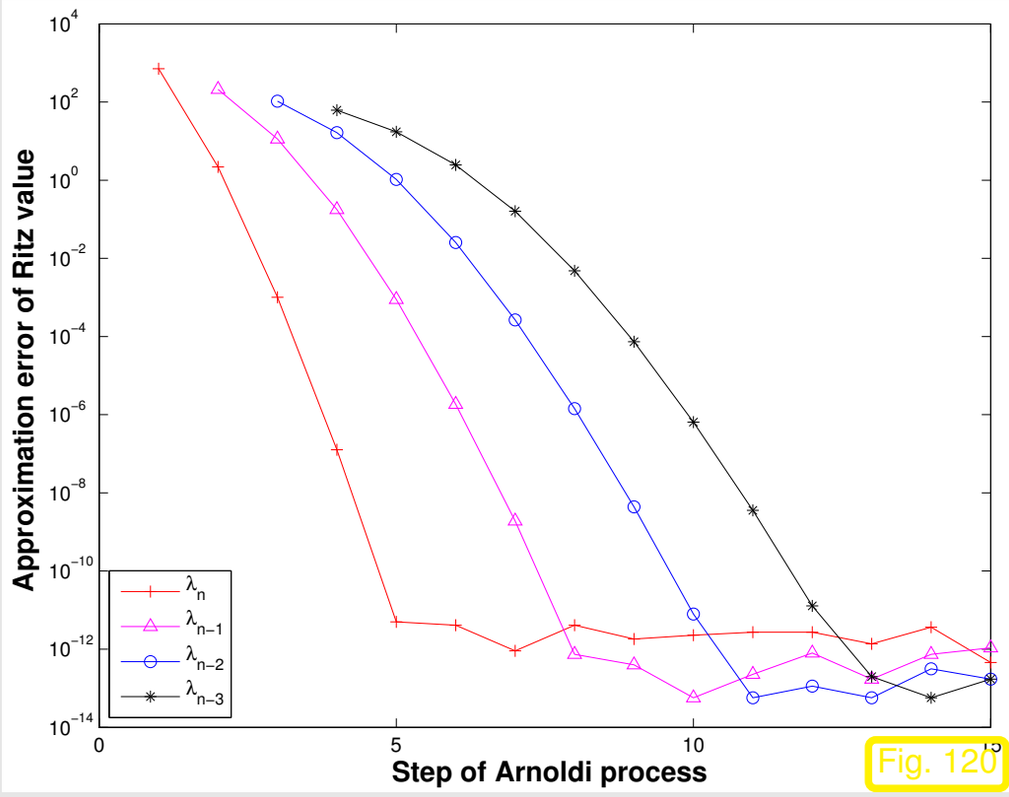


Fig. 120

Arnoldi process: Ritz values

Ritz values during Arnoldi process for `A = gallery('minij',10);` $\leftrightarrow$ Ex. 6.4.12

R. Hiptmair

rev 30401, November 22, 2010

l	$\sigma(\mathbf{H}_l)$									
1										38.500000
2									3.392123	44.750734
3								1.117692	4.979881	44.766064
4							0.597664	1.788008	5.048259	44.766069
5						0.415715	0.925441	1.870175	5.048916	44.766069
6					0.336507	0.588906	0.995299	1.872997	5.048917	44.766069
7				0.297303	0.431779	0.638542	0.999922	1.873023	5.048917	44.766069
8			0.276159	0.349722	0.462449	0.643016	1.000000	1.873023	5.048917	44.766069
9		0.263872	0.303009	0.365379	0.465199	0.643104	1.000000	1.873023	5.048917	44.766069
10	0.255680	0.273787	0.307979	0.366209	0.465233	0.643104	1.000000	1.873023	5.048917	44.766069

Observation: (almost perfect approximation of spectrum of $\mathbf{A}$)

For the above examples both the Arnoldi process and the Lanczos process are *algebraically equivalent*, because they are applied to a symmetric matrix $\mathbf{A} = \mathbf{A}^T$. However, they behave strikingly differently, which indicates that they are *not numerically equivalent*.

The Arnoldi process is much less affected by roundoff than the Lanczos process, because it does not take for granted orthogonality of the "residual vector sequence". Hence, the Arnoldi process enjoys superior numerical stability ($\rightarrow$ Sect. 2.5.2, Def. 2.5.11) compared to the Lanczos process.

$\diamond$

Example 6.4.20 (Eigenvalue computation with Arnoldi process).

Eigenvalue approximation from Arnoldi process for *non-symmetric* $\mathbf{A}$, initial vector `ones(100,1);`

```
1 n=100;
2 M=full(gallery('tridiag',-0.5*ones(n-1,1),2*ones(n,1),-1.5*ones(n-1,
3 A=M*diag(1:n)*inv(M);
```

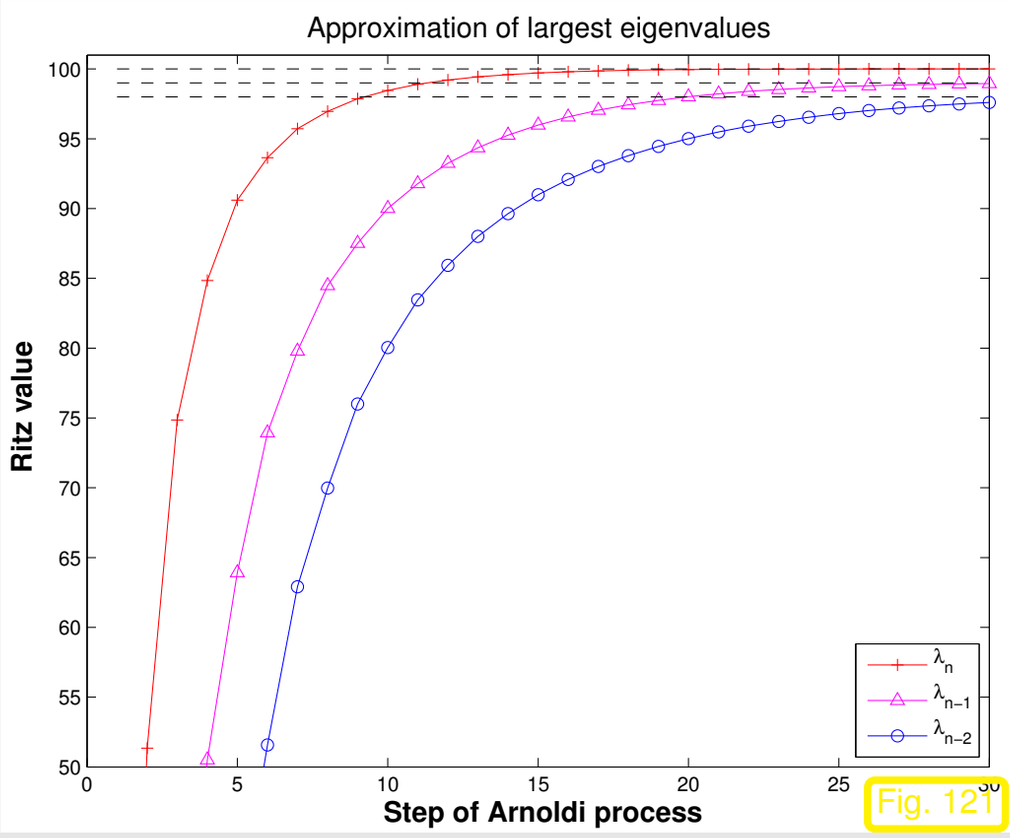


Fig. 121

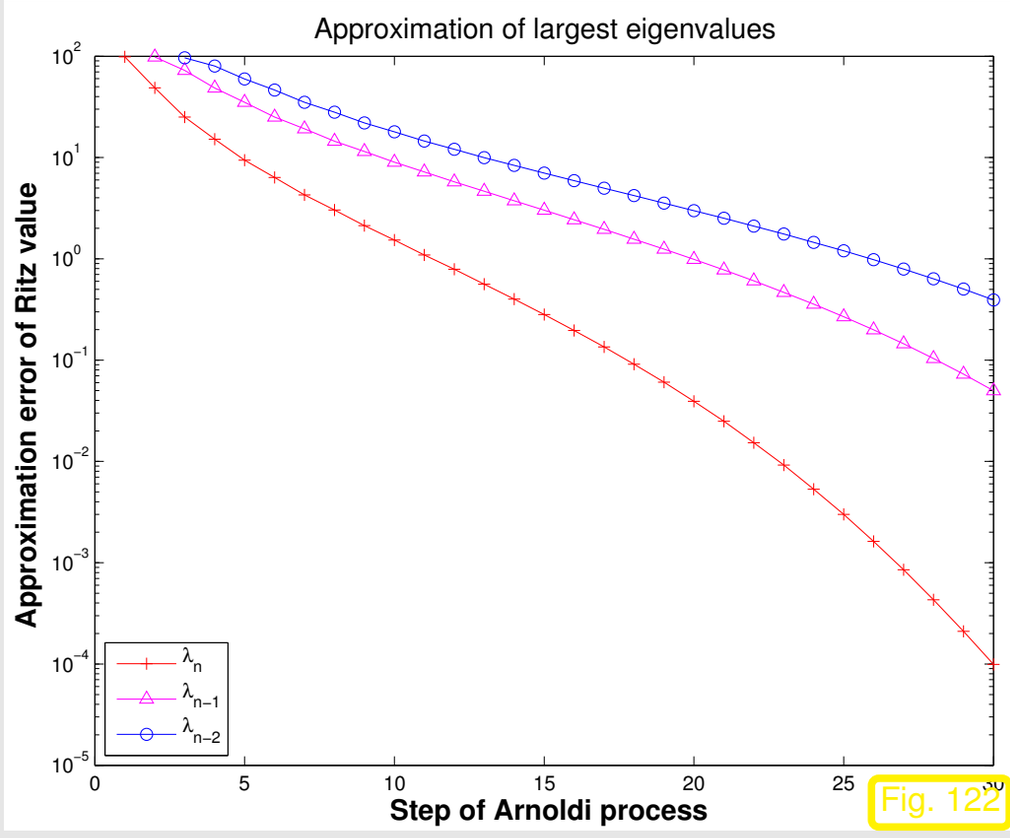


Fig. 122

R. Hiptmair

rev 30401, November 22, 2010

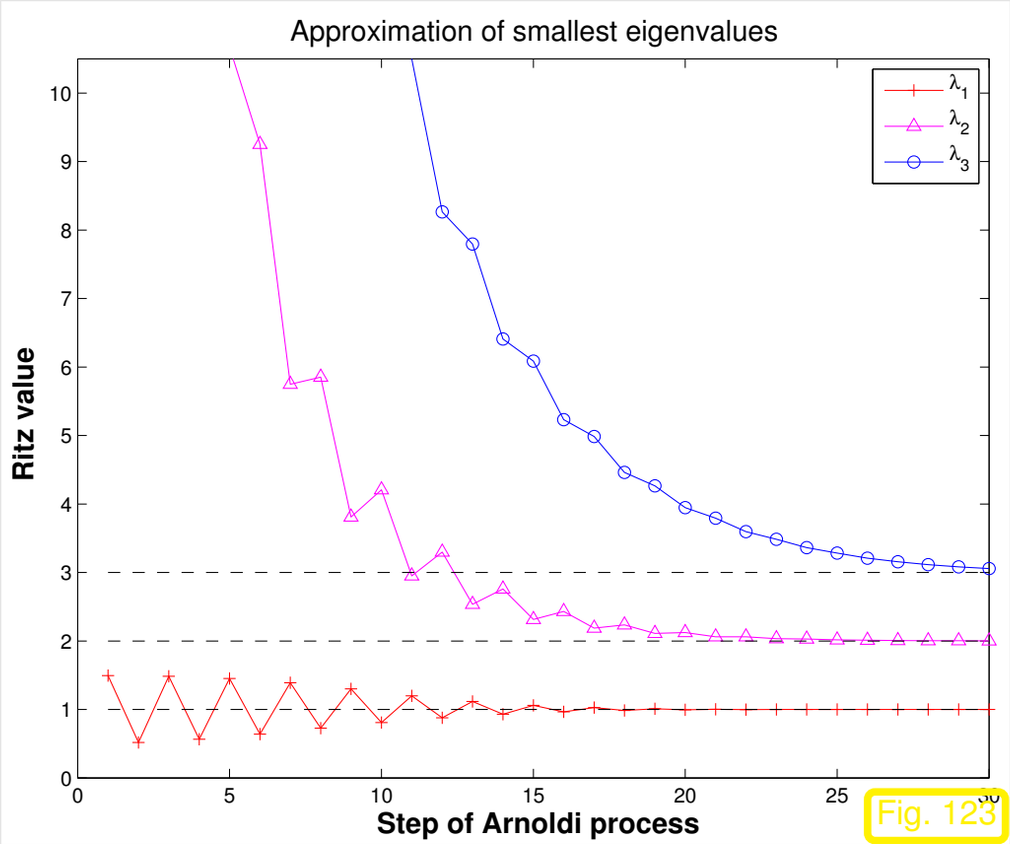


Fig. 123

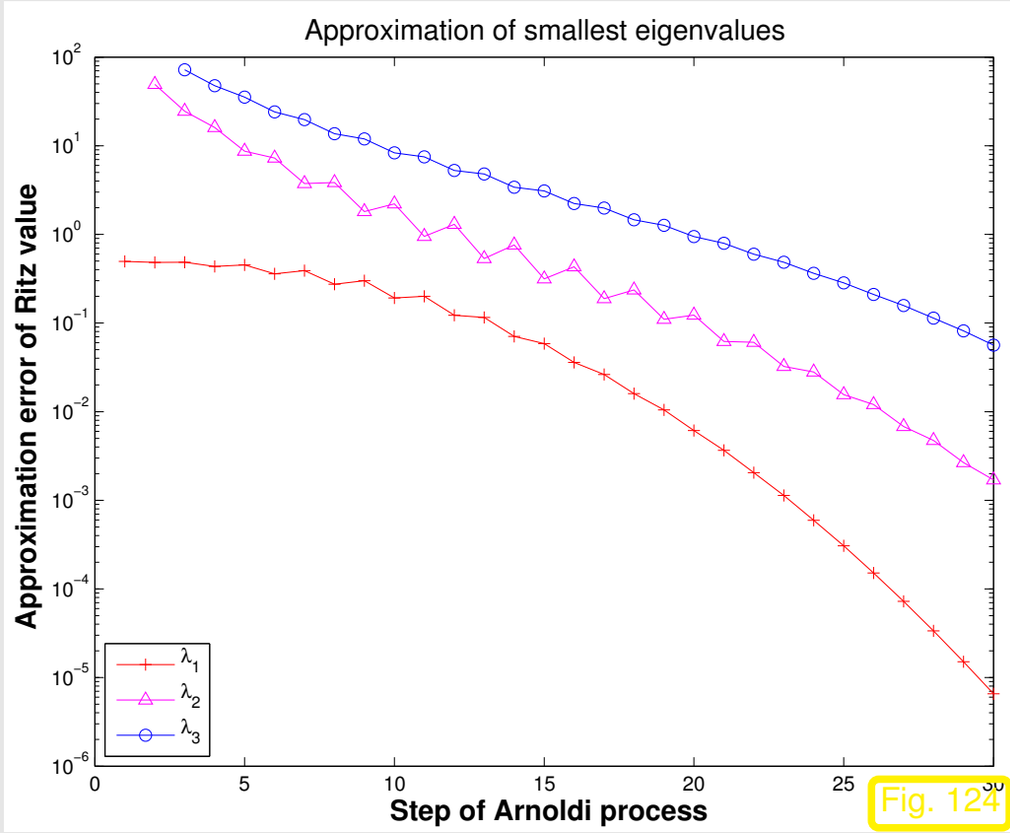


Fig. 124

Observation: "vaguely linear" convergence of largest and smallest eigenvalues, *cf.* Ex. 6.4.4. ◇

Krylov subspace iteration methods (= Arnoldi process, Lanczos process) attractive for computing *a few* of the largest/smallest eigenvalues and associated eigenvectors of *large sparse matrices*.

R. Hiptmair

rev 30401, November 22, 2010

Remark 6.4.21 (Krylov subspace methods for generalized EVP).

Adaptation of Krylov subspace iterative eigensolvers to generalized EVP: $\mathbf{Ax} = \lambda\mathbf{Bx}$, $\mathbf{B}$ s.p.d.: replace Euclidean inner product with "$\mathbf{B}$-inner product" $(\mathbf{x},\mathbf{y}) \mapsto \mathbf{x}^H\mathbf{By}$.

△

MATLAB-functions:

`d = eigs(A,k,sigma)` : k largest/smallest eigenvalues of $\mathbf{A}$

`d = eigs(A,B,k,sigma)` : k largest/smallest eigenvalues for generalized EVP $\mathbf{Ax} = \lambda\mathbf{Bx}$,$\mathbf{B}$ s.p.d.

`d = eigs(Afun,n,k)` : `Afun` = handle to function providing matrix×vector for $\mathbf{A}$/$\mathbf{A}^{-1}$/$\mathbf{A}-\alpha\mathbf{I}$/$(\mathbf{A}-\alpha\mathbf{B})^{-1}$. (Use flags to tell `eigs` about special properties of matrix behind `Afun`.)

`eigs` just calls routines of the open source ARPACK numerical library.

6.5 Singular Value Decomposition

Example 6.5.1 (Trend analysis).

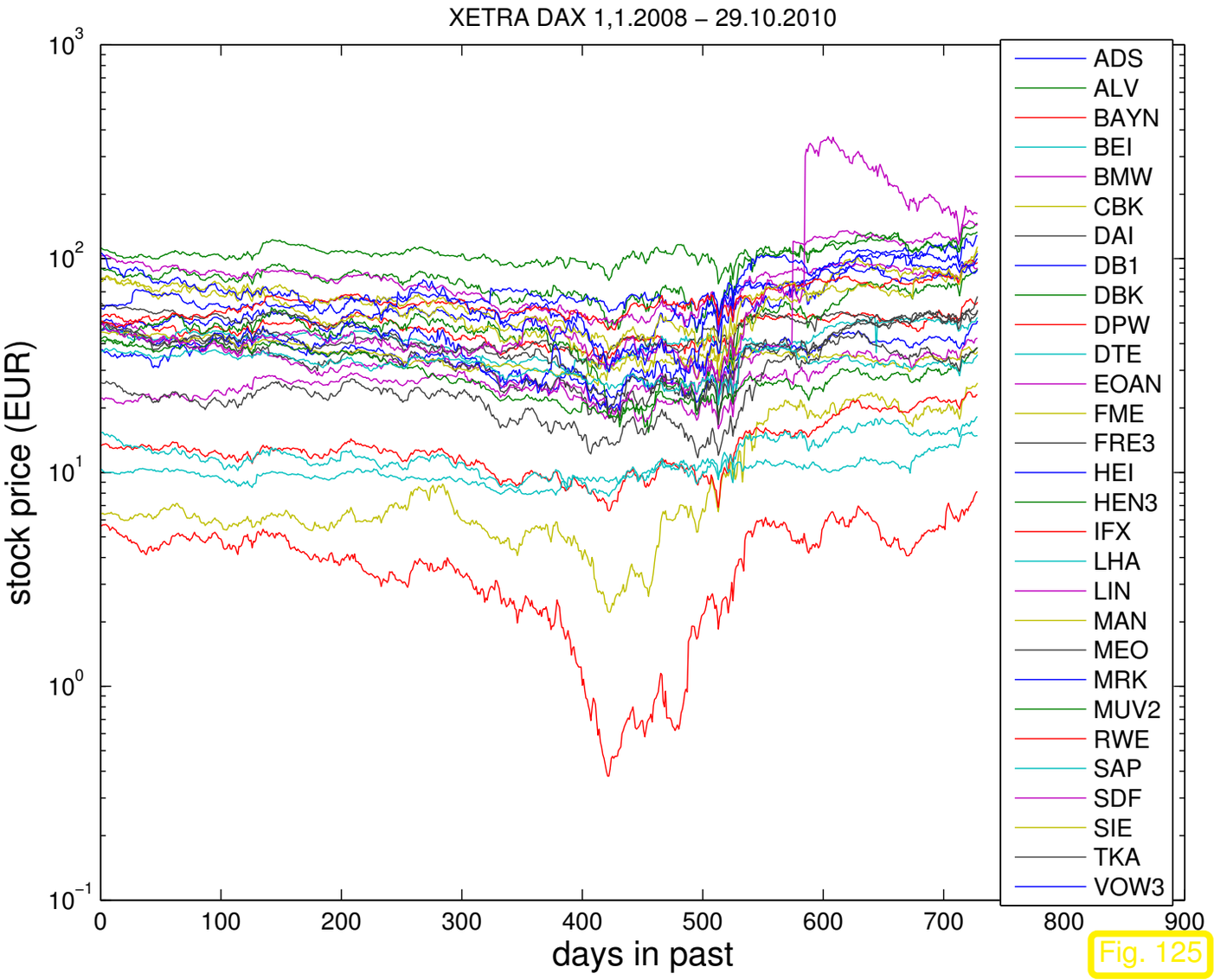


Fig. 125

◁ (end of day) stock prizes

Are there underlying governing trends **?**

◇

R. Hiptmair

rev 30401, February 19, 2011

Remark 6.5.2 (Principal component analysis (PCA)).

Given: n data points $\mathbf{a}_j \in \mathbb{R}^m$, $j = 1, \ldots, n$, in m-dimensional (feature) space
(e.g., $\mathbf{a}_j$ may represent a finite *time series*)

In Ex. 6.5.1: $n \,\hat{=}\,$ number of stocks
$m \,\hat{=}\,$ number of days, for which stock prices are recorded

- Extreme case: all stocks follow exactly *one* trend

$$\leftrightarrow \quad \mathbf{a}_j \in \operatorname{Span}\{\mathbf{u}\} \quad \forall j = 1, \ldots, n \;,$$

for a trend vector $\mathbf{u} \in \mathbb{R}^m$, $\|\mathbf{u}\|_2 = 1$.

- Unlikely case: all stocks prices are governed $p < n$ trends:

$$\leftrightarrow \quad \mathbf{a}_j \in \operatorname{Span}\{\mathbf{u}_1, \ldots, \mathbf{u}_p\} \quad \forall j = 1, \ldots, m \;, \tag{6.5.3}$$

with orthonormal trend vectors $\mathbf{u}_i \in \mathbb{R}^m$, $i = 1, \ldots, p$.

Why orthonormal **?** Trends should be as “independent as possible” (minimally correlated)

R. Hiptmair
rev 30401, February 19, 2011

Perspective of linear algebra:

(6.5.3) $\Leftrightarrow$ $\operatorname{rank}(\mathbf{A}) = p$ for $\mathbf{A} := (\mathbf{a}_1, \ldots, \mathbf{a}_n) \in \mathbb{R}^{m,n}$, $\operatorname{Im}(\mathbf{A}) = \operatorname{Span}\{\mathbf{u}_1, \ldots, \mathbf{u}_p\}$ (6.5.4)

- Realistic: stock prizes *approximately* follow a few trends

$$\mathbf{a}_j \in \operatorname{Span}\{\mathbf{u}_1, \ldots, \mathbf{u}_p\} + \text{“small perturbations”} \quad \forall j = 1, \ldots, m ,$$

with orthonormal trend vectors $\mathbf{u}_i$, $i = 1, \ldots, p$.

Task (PCA): determine (minimal) p and orthonormal trend vectors $\mathbf{u}_i$, $i = 1, \ldots, p$

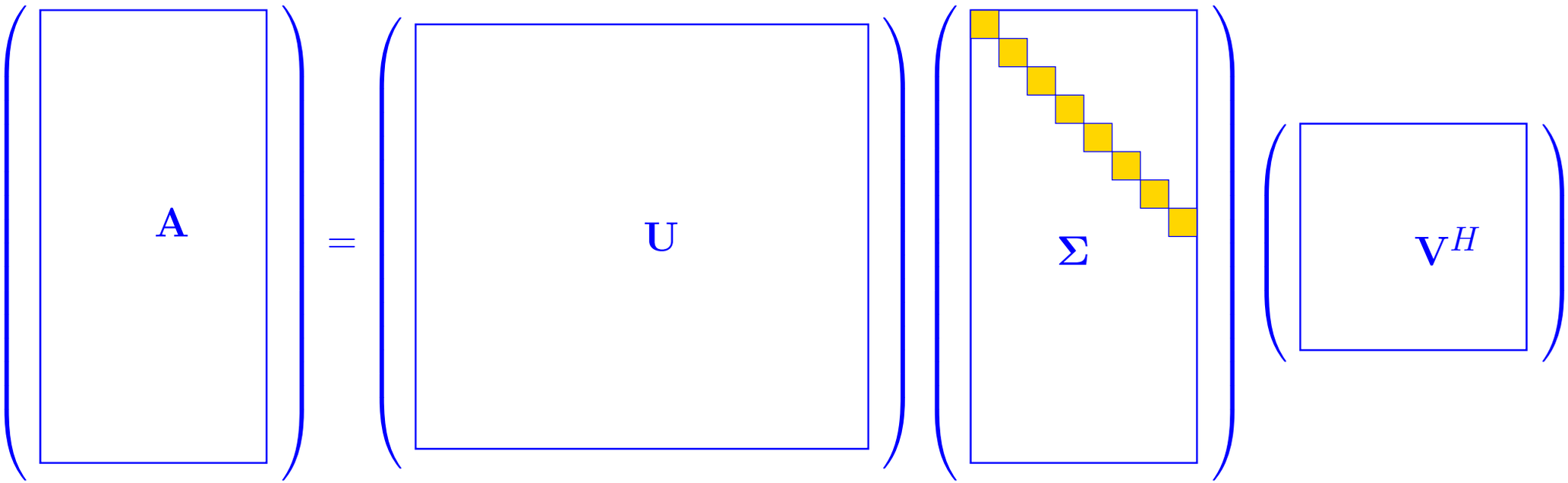
Issue: how to distinguish between trends and perturbations ?

Theorem 6.5.5 (singular value decomposition). → *[39, Thm. 9.6], [23, 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* $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m,n}$, $p := \min\{m, n\}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ *such that*

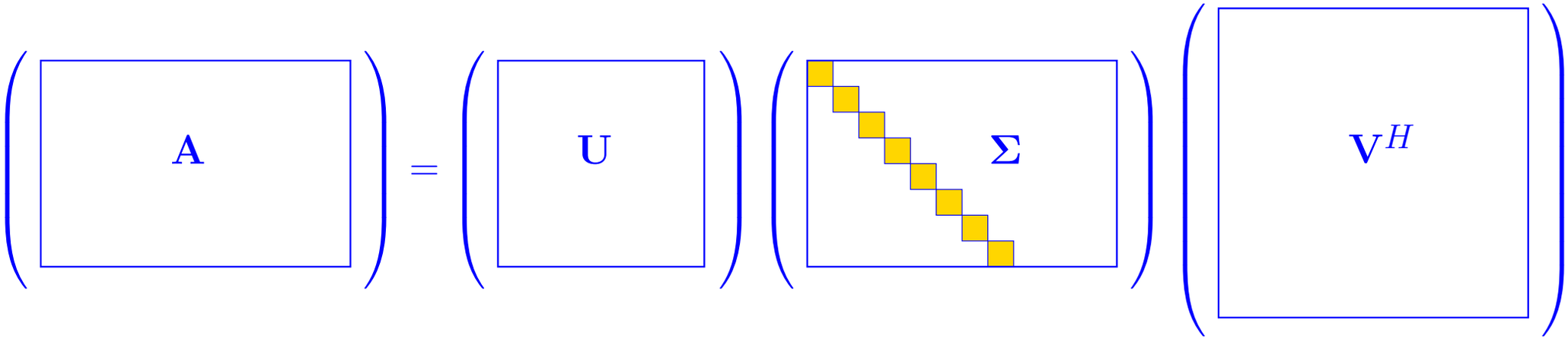
$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H \, .$$

$(*)$: $\mathbf{\Sigma}$ (generalized) diagonal matrix $:\Leftrightarrow$ $(\mathbf{\Sigma})_{i,j} = 0$, if $i \neq j$, $1 \leq i \leq m$, $1 \leq j \leq n$.



R. Hiptmair

rev 30401, February 19, 2011



Proof. (of Thm. 6.5.5, by induction)

[52, Thm. 4.2.3]: Continuous functions attain extremal values on compact sets (here the unit ball $\{\mathbf{x} \in \mathbb{R}^n\colon \|\mathbf{x}\|_2 \leq 1\}$)

$$\blacktriangleright \quad \exists \mathbf{x} \in \mathbb{K}^n, \mathbf{y} \in \mathbb{K}^m\,, \quad \|\mathbf{x}\| = \|\mathbf{y}\|_2 = 1 : \quad \mathbf{A}\mathbf{x} = \sigma\mathbf{y}\,, \quad \sigma = \|\mathbf{A}\|_2\,,$$

where we used the definition of the matrix 2-norm, see Def. 2.5.5. By Gram-Schmidt orthogonalization: $\exists \widetilde{\mathbf{V}} \in \mathbb{K}^{n,n-1}$, $\widetilde{\mathbf{U}} \in \mathbb{K}^{m,m-1}$ such that

$$\mathbf{V} = (\mathbf{x}\ \widetilde{\mathbf{V}}) \in \mathbb{K}^{n,n} \quad , \quad \mathbf{U} = (\mathbf{y}\ \widetilde{\mathbf{U}}) \in \mathbb{K}^{m,m} \quad \text{are unitary.}$$

$$\blacktriangleright \quad \mathbf{U}^H\mathbf{A}\mathbf{V} = (\mathbf{y}\ \widetilde{\mathbf{U}})^H\mathbf{A}(\mathbf{x}\ \widetilde{\mathbf{V}}) = \left(\begin{array}{c|c} \mathbf{y}^H\mathbf{A}\mathbf{x} & \mathbf{y}^H\mathbf{A}\widetilde{\mathbf{V}} \\ \hline \widetilde{\mathbf{U}}^H\mathbf{A}\mathbf{x} & \widetilde{\mathbf{U}}^H\mathbf{A}\widetilde{\mathbf{V}} \end{array}\right) = \left(\begin{array}{c|c} \sigma & \mathbf{w}^H \\ \hline 0 & \mathbf{B} \end{array}\right) =: \mathbf{A}_1 \ .$$

Since

$$\left\|\mathbf{A}_1\begin{pmatrix}\sigma\\ \mathbf{w}\end{pmatrix}\right\|_2^2 = \left\|\begin{pmatrix}\sigma^2+\mathbf{w}^H\mathbf{w}\\ \mathbf{B}\mathbf{w}\end{pmatrix}\right\|_2^2 = (\sigma^2+\mathbf{w}^H\mathbf{w})^2 + \|\mathbf{B}\mathbf{w}\|_2^2 \geq (\sigma^2+\mathbf{w}^H\mathbf{w})^2 \ ,$$

we conclude

$$\|\mathbf{A}_1\|_2^2 = \sup_{0\neq\mathbf{x}\in\mathbb{K}^n} \frac{\|\mathbf{A}_1\mathbf{x}\|_2^2}{\|\mathbf{x}\|_2^2} \geq \frac{\left\|\mathbf{A}_1\begin{pmatrix}\sigma\\ \mathbf{w}\end{pmatrix}\right\|_2^2}{\left\|\begin{pmatrix}\sigma\\ \mathbf{w}\end{pmatrix}\right\|_2^2} \geq \frac{(\sigma^2+\mathbf{w}^H\mathbf{w})^2}{\sigma^2+\mathbf{w}^H\mathbf{w}} = \sigma^2+\mathbf{w}^H\mathbf{w} \ . \tag{6.5.6}$$

$$\sigma^2 = \|\mathbf{A}\|_2^2 = \left\|\mathbf{U}^H\mathbf{A}\mathbf{V}\right\|_2^2 = \|\mathbf{A}_1\|_2^2 \overset{(6.5.6)}{\Longrightarrow} \|\mathbf{A}_1\|_2^2 = \|\mathbf{A}_1\|_2^2 + \|\mathbf{w}\|_2^2 \quad \Rightarrow \quad \mathbf{w} = 0 \ .$$

$$\blacktriangleright \quad \mathbf{A}_1 = \left(\begin{array}{c|c} \sigma & 0 \\ \hline 0 & \mathbf{B} \end{array}\right) \ .$$

Then apply induction argument to $\mathbf{B}$ □.

Definition 6.5.7 (Singular value decomposition (SVD))**.**
The decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$ *of Thm. 6.5.5 is called singular value decomposition (SVD) of* $\mathbf{A}$*. The diagonal entries* σ_i *of* $\mathbf{\Sigma}$ *are the singular values of* $\mathbf{A}$*.*

Lemma 6.5.8. *The squares* σ_i^2 *of the non-zero singular values of* $\mathbf{A}$ *are the non-zero eigenvalues of* $\mathbf{A}^H\mathbf{A}$, $\mathbf{A}\mathbf{A}^H$ *with associated eigenvectors* $(\mathbf{V})_{:,1}, \ldots, (\mathbf{V})_{:,p}$, $(\mathbf{U})_{:,1}, \ldots, (\mathbf{U})_{:,p}$, *respectively.*

Proof. $\mathbf{A}\mathbf{A}^H$ and $\mathbf{A}^H\mathbf{A}$ are similar ($\rightarrow$ Lemma 6.1.6) to diagonal matrices with non-zero diagonal entries σ_i^2 ($\sigma_i \neq 0$), e.g.,

$$\mathbf{A}\mathbf{A}^H = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H\mathbf{V}\mathbf{\Sigma}^H\mathbf{U}^H = \mathbf{U}\underbrace{\mathbf{\Sigma}\mathbf{\Sigma}^H}_{\text{diagonal matrix}}\mathbf{U}^H .$$ □

Remark 6.5.9 (SVD and additive rank-1 decomposition). $\rightarrow$ [23, Cor. 11.2], [39, Thm. 9.8]

Recall from linear algebra: rank-1 matrices are tensor products of vectors

$$\mathbf{A} \in \mathbb{K}^{m,n} \quad \text{and} \quad \operatorname{rank}(\mathbf{A}) = 1 \quad \Leftrightarrow \quad \exists \mathbf{u} \in \mathbb{K}^m, \mathbf{v} \in \mathbb{K}^n: \quad \mathbf{A} = \mathbf{u}\mathbf{v}^H , \tag{6.5.10}$$

R. Hiptmair

rev 30401, February 19, 2011

because $\operatorname{rank}(\mathbf{A}) = 1$ means that $\mathbf{Ax} = \mu(\mathbf{x})\mathbf{u}$ for some $\mathbf{u} \in \mathbb{K}^m$ and linear form $\mathbf{x} \mapsto \mu(\mathbf{x})$. By the Riesz representation theorem the latter can be written as $\mu(\mathbf{x}) = \mathbf{v}^H\mathbf{x}$.

► Singular value decomposition provides additive decomposition into rank-1 matrices:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H = \sum_{j=1}^{p} \sigma_j \, (\mathbf{U})_{:,j}(\mathbf{V})_{:,j}^H \; . \tag{6.5.11}$$

Remark 6.5.12 (Uniqueness of SVD).

SVD of Def. 6.5.7 is not (necessarily) unique, but the singular values are.

Assume that $\mathbf{A}$ has two singular value decompositions

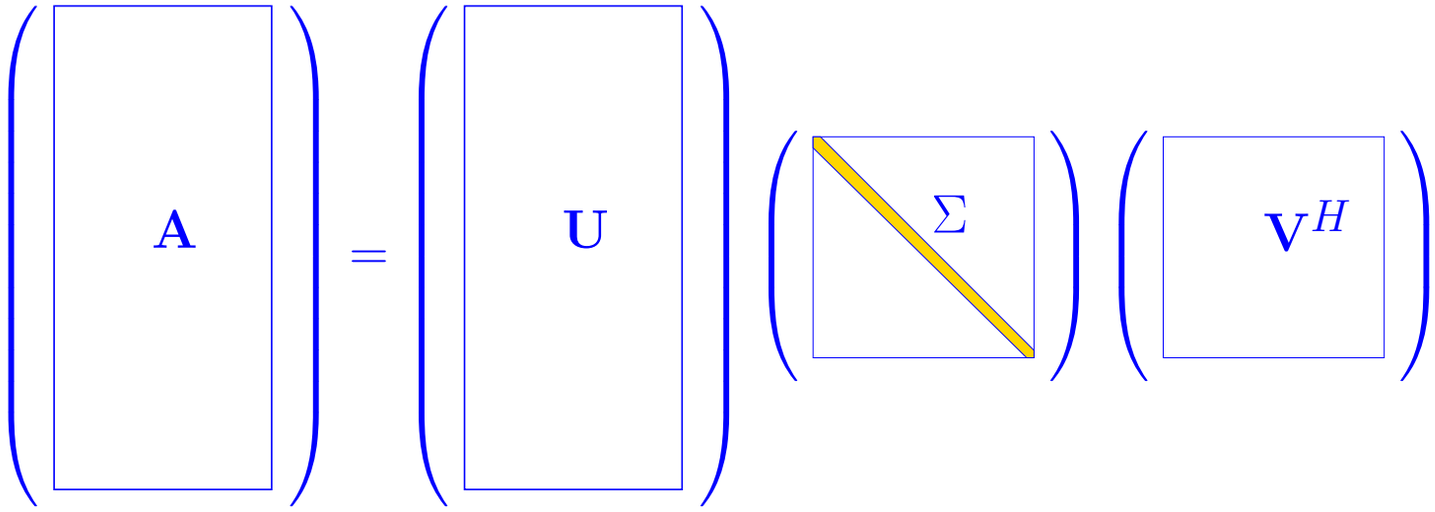
$$\mathbf{A} = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^H = \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^H \quad \Rightarrow \quad \mathbf{U}_1 \underbrace{\mathbf{\Sigma}_1\mathbf{\Sigma}_1^H}_{=\operatorname{diag}(s_1^1,\ldots,s_m^1)} \mathbf{U}_1^H = \mathbf{A}\mathbf{A}^H = \mathbf{U}_2 \underbrace{\mathbf{\Sigma}_2\mathbf{\Sigma}_2^H}_{=\operatorname{diag}(s_1^2,\ldots,s_m^2)} \mathbf{U}_2^H \; .$$

Two similar diagonal matrices with non-increasing diagonal entries are equal ! □

MATLAB-functions (for algorithms see [20, 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. 6.5.5
`[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 $\mathbf{A}$ → 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:



R. Hiptmair

rev 30401, February 19, 2011

(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)`, $m \gg n$.

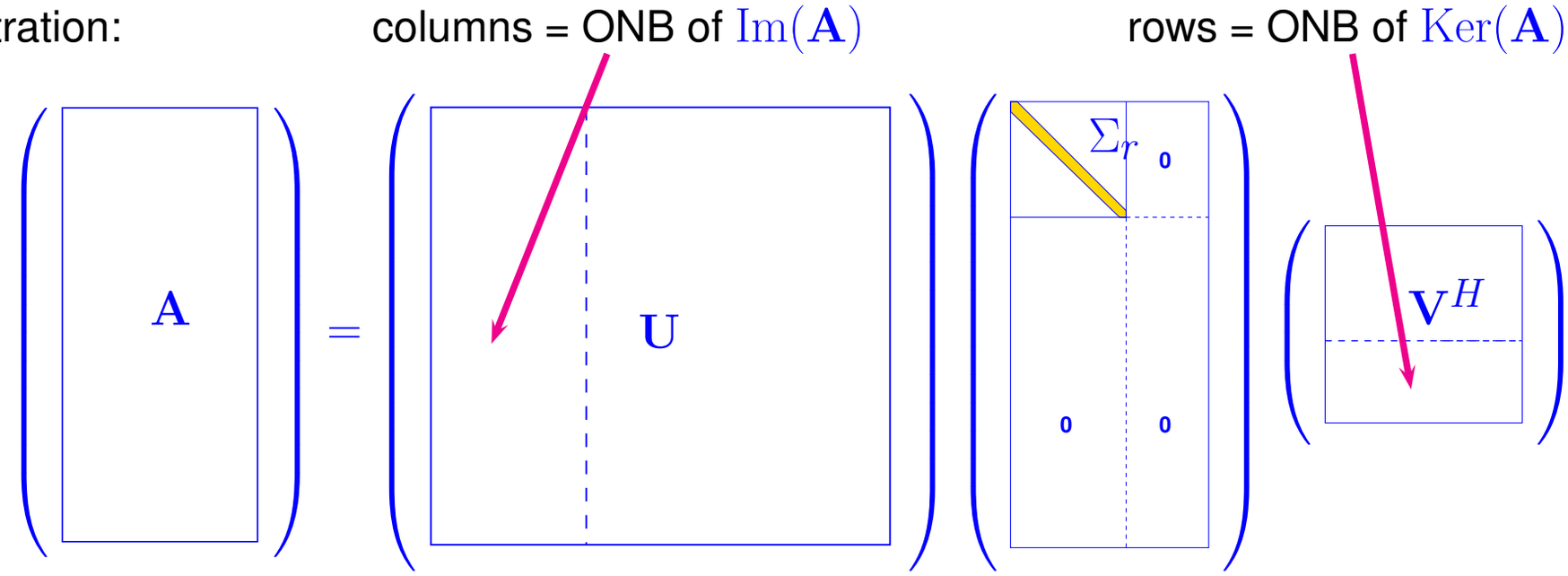
- Application of SVD: computation of rank ($\rightarrow$ Def. 2.0.6), kernel and range of a matrix

Lemma 6.5.13 (SVD and rank of a matrix). $\rightarrow$ *[39, Cor. 9.7],*

If the singular values of $\mathbf{A}$ *satisfy* $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots \sigma_p = 0$, *then*

- $\mathrm{rank}(\mathbf{A}) = r$,
- $\mathrm{Ker}(\mathbf{A}) = \mathrm{Span}\left\{(\mathbf{V})_{:,r+1}, \ldots, (\mathbf{V})_{:,n}\right\}$,
- $\mathrm{Im}(\mathbf{A}) = \mathrm{Span}\left\{(\mathbf{U})_{:,1}, \ldots, (\mathbf{U})_{:,r}\right\}$.

Illustration: columns = ONB of $\mathrm{Im}(\mathbf{A})$ rows = ONB of $\mathrm{Ker}(\mathbf{A})$



(6.5.14)

Code 6.5.15: Computing an ONB of the kernel of a matrix

```
function V = kerncomp(A,tol)
% computes an orthonormal basis of Ker(A).
% kernel selection with relative tolerance tol
if (nargin < 2), tol = eps; end
[U,S,V] = svd(A); s = diag(S);
% find singular values of relative (w.r.t. σ_1) size ≤tol
r = min(find(s/s(1) <= tol));
V = V(:,r:end);
```

R. Hiptmair

rev 30401, February 19, 2011

Remark: MATLAB function r=rank(A) relies on svd(A)

- Application of SVD: PCA by SVD ($\rightarrow$ Rem. 6.5.2)

By Rem. 6.5.9, see (6.5.11), if $\mathbf{u}_j \hat{=}$ columns of $\mathbf{U}$, $\mathbf{v}_j \hat{=}$ columns of $\mathbf{V}$,

$$\mathbf{A} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \ldots$$

R. Hiptmair

rev 30401, February 19, 2011

❶ no perturbations:

$$\text{SVD:}\quad \mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H \quad \text{satisfies} \quad \sigma_1 \geq \sigma_2 \geq \ldots \sigma_p > \sigma_{p+1} = \cdots = \sigma_{\min\{m,n\}} = 0 \;,$$
$$V = \operatorname{Span} \underbrace{\{(\mathbf{U})_{:,1}, \ldots, (\mathbf{U})_{:,p}\}}_{\text{orthonormal trend vectors}} \;.$$

❷ with perturbations:

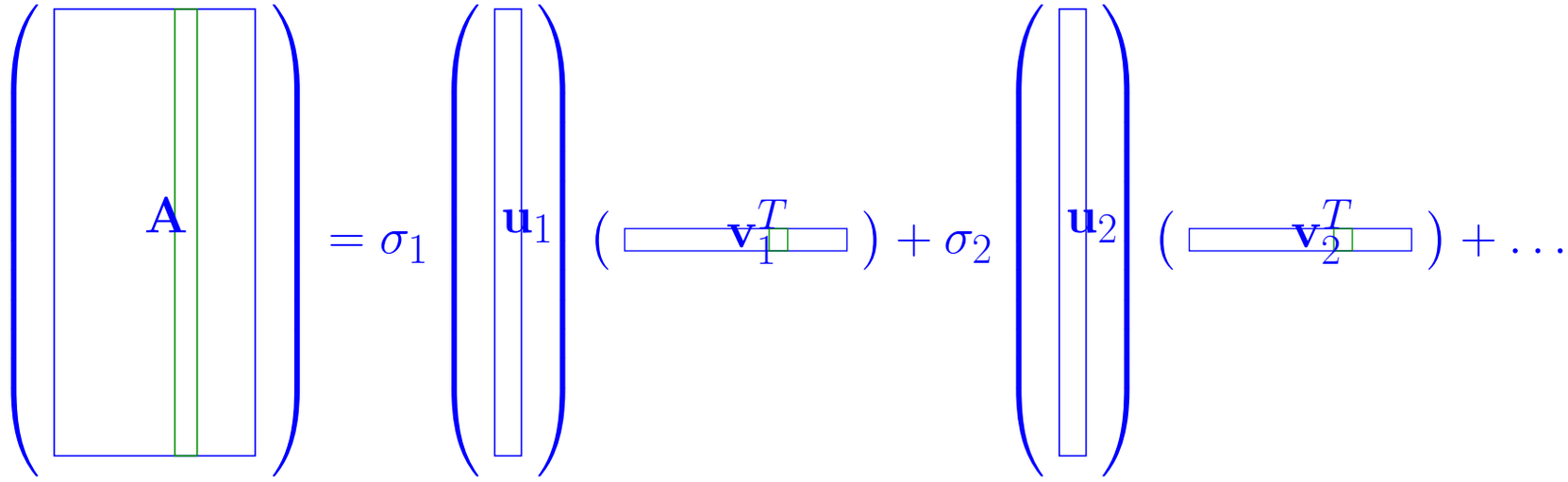
$$\text{SVD:}\quad \mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H \quad \text{satisfies} \quad \sigma_1 \geq \sigma_2 \geq \ldots \sigma_p \gg \sigma_{p+1} \approx \cdots \approx \sigma_{\min\{m,n\}} \approx 0 \;,$$
$$V = \operatorname{Span} \underbrace{\{(\mathbf{U})_{:,1}, \ldots, (\mathbf{U})_{:,p}\}}_{\text{orthonormal trend vectors}} \;.$$

If there is a pronounced gap in distribution of the singular values, which separates p large from $\min\{m,n\} - p$ relatively small singular values, this hints that $\operatorname{Im}(\mathbf{A})$ has essentially dimension p. It depends on the application what one accepts as a “pronounced gap”.

Frequently used criterion:

$$p = \min\left\{ q\colon \sum_{j=1}^{q} \sigma_j^2 \geq (1-\tau) \sum_{j=1}^{\min\{m,n\}} \sigma_j^2 \right\} \quad \text{for} \quad \tau \ll 1 \;.$$

Information carried by $\mathbf{V}$ in PCA context:



j-th data set ($\leftrightarrow$ time series #j) in j-th column of $\mathbf{A}$

$$\text{(6.5.11)} \quad \Rightarrow \quad (\mathbf{A})_{:,j} = \sigma_1 \mathbf{u}_1 (\mathbf{v}_1)_j + \sigma_2 \mathbf{u}_2 (\mathbf{v}_2)_j + \ldots$$

The j-th row of $\mathbf{V}$ (up to the p-th component) gives the weights with which the p identified trends contribute to data set j.

Example 6.5.16 (Data points confined to a subspace).

Code 6.5.17: PCA in three dimensions via SVD

```
% Use of SVD for PCA with perturbations

V = [1 , -1; 0 , 0.5; -1 , 0]; A = V*rand(2,20)+0.1*rand(3,20);
[U,S,V] = svd(A,0);

figure; sv = diag(S(1:3,1:3))

[X,Y] = meshgrid(-2:0.2:0,-1:0.2:1); n = size(X,1); m = size(X,2);
figure; plot3(A(1,:),A(2,:),A(3,:),'r*'); grid on; hold on;
M =
  U(:,1:2)*[reshape(X,1,prod(size(X)));reshape(Y,1,prod(size(Y)))];
mesh(reshape(M(1,:),n,m),reshape(M(2,:),n,m),reshape(M(3,:),n,m));
colormap(cool); view(35,10);

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

R. Hiptmair

rev 30401, February 19, 2011

singular values:

$$\begin{array}{c} 3.1378 \\ 1.8092 \\ \hline 0.1792 \end{array}$$

We observe a gap between the second and third singular value ➢ the data points essentially lie in a 2D subspace.

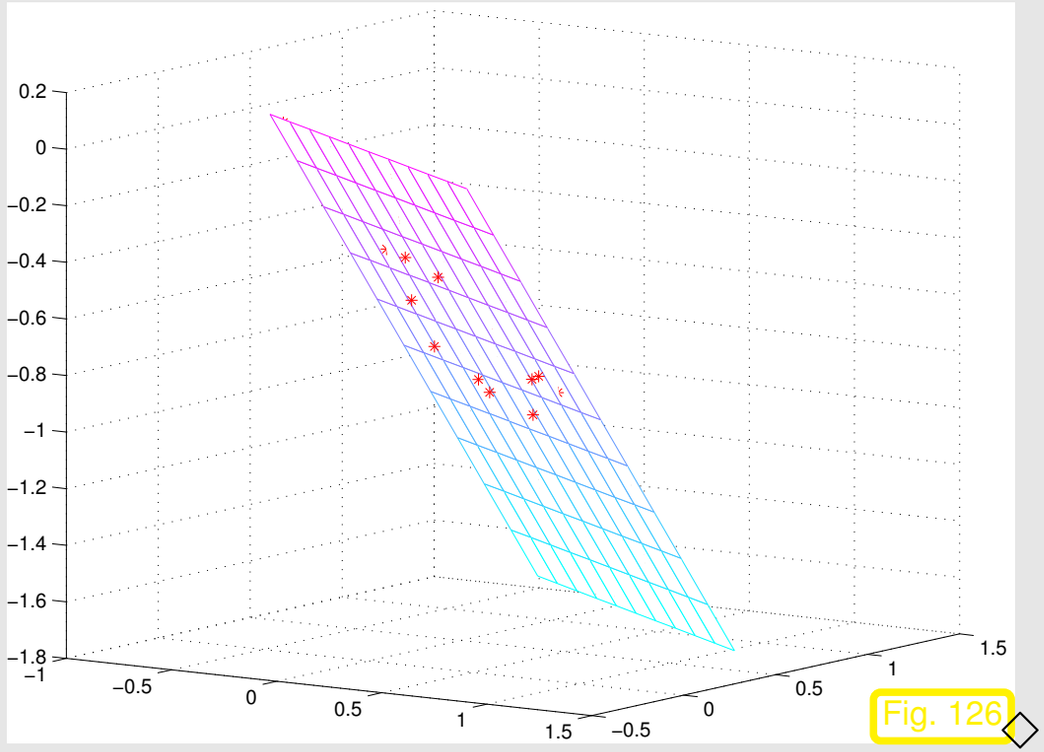


Fig. 126

Example 6.5.18 (Principal component analysis for data analysis). → Ex. 6.5.1 cnt'd

$\mathbf{A} \in \mathbb{R}^{m,n}$, $m \gg n$:

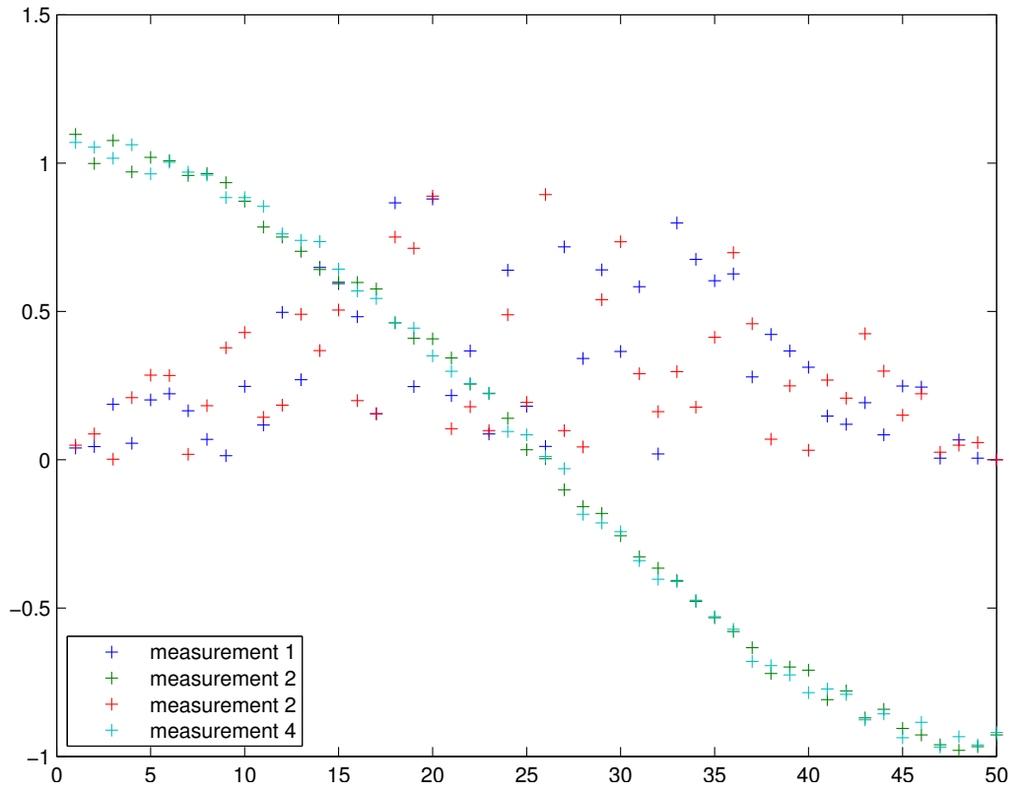
Columns $\mathbf{A}$ → series of measurements at different times/locations etc.
Rows of $\mathbf{A}$ → measured values corresponding to one time/location etc.

Goal: detect linear correlations

Concrete: two quantities measured over one year at 10 different sites

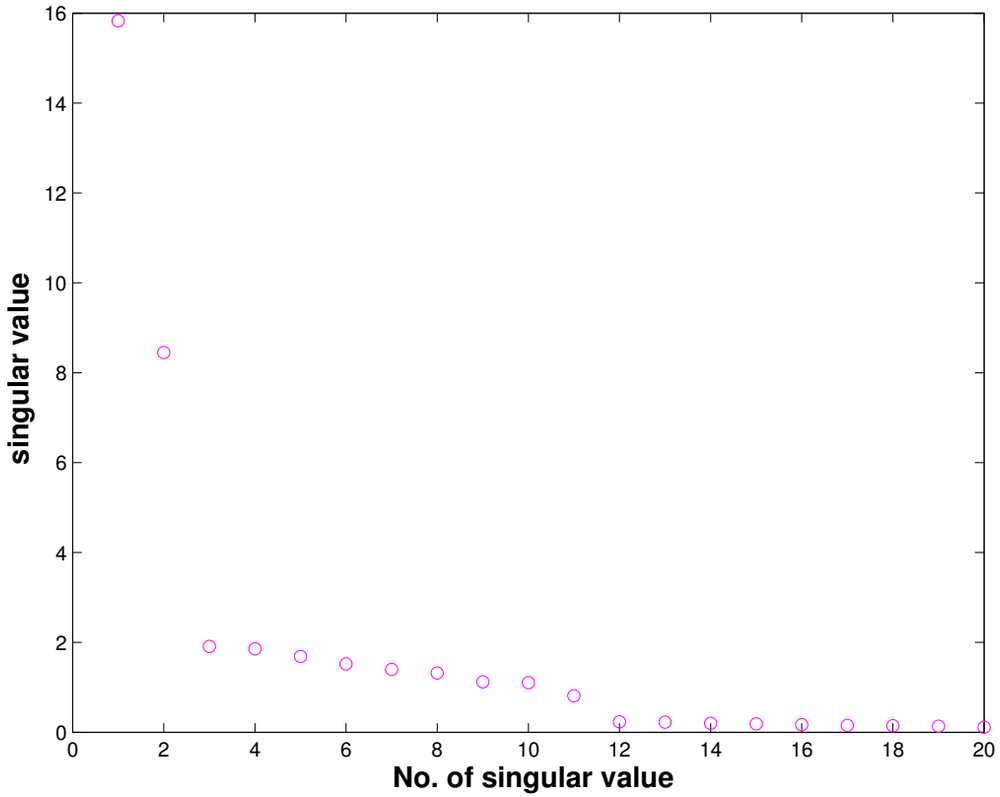
(Of course, measurements affected by errors/fluctuations)

```
n = 10;
m = 50;
x = sin(pi*(1:m)'/m);
y = cos(pi*(1:m)'/m);
A = [];
for i = 1:n
  A = [A, x.*rand(m,1),...
       y+0.1*rand(m,1)];
end
```



R. Hiptmair

rev 30401, February 19, 2011



← distribution of singular values of matrix

two dominant singular values !

measurements display linear correlation with **two** principal components

principal components = $\mathbf{u}_{.,1}$, $\mathbf{u}_{.,2}$ (leftmost columns of $\mathbf{U}$-matrix of SVD)

their relative weights = $\mathbf{v}_{.,1}$, $\mathbf{v}_{.,2}$ (leftmost columns of $\mathbf{V}$-matrix of SVD)

R. Hiptmair

rev 30401, February 19, 2011

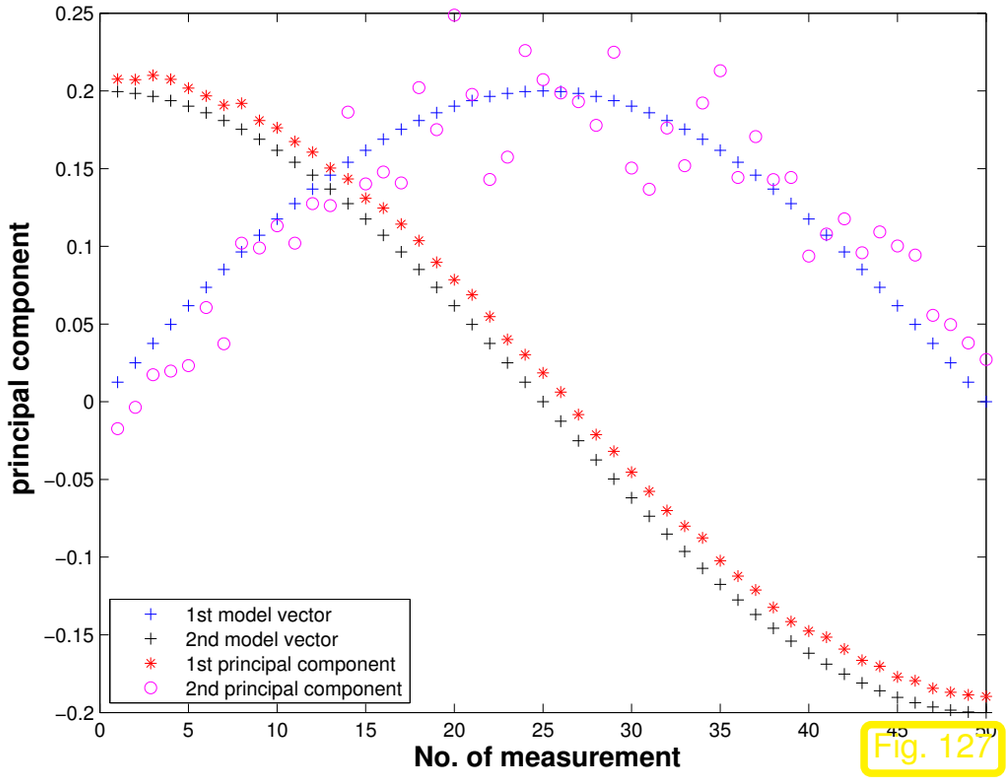


Fig. 127

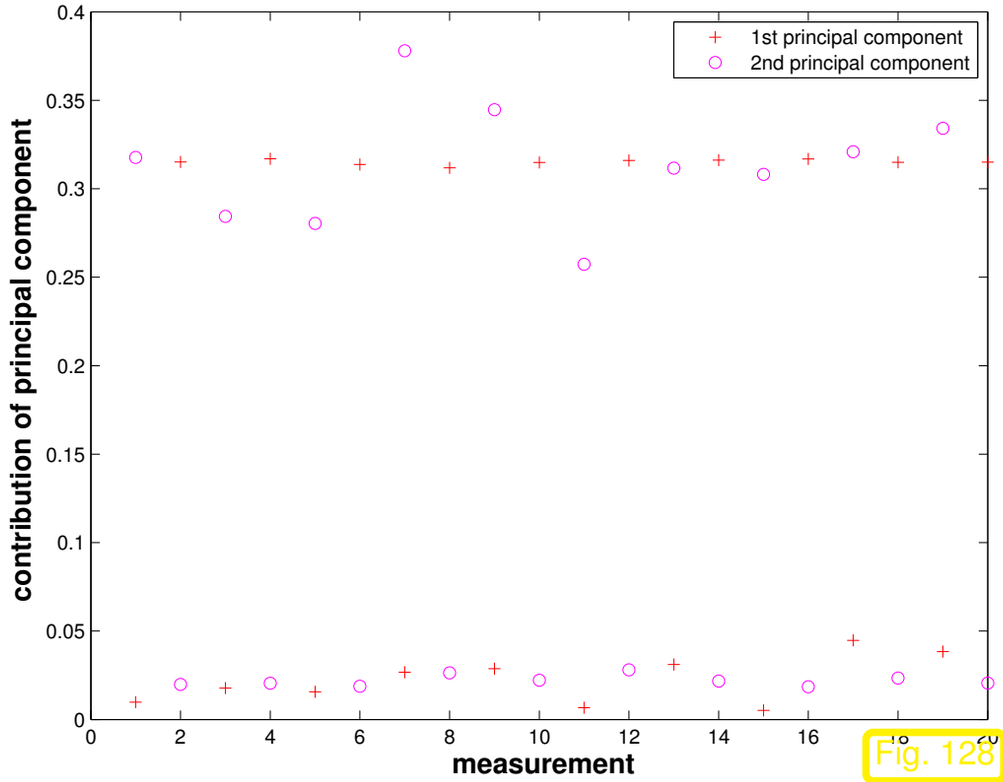


Fig. 128 ◇

Example 6.5.19 (PCA of stock prices). → Ex. 6.5.1

columns of $\mathbf{A}$ → time series of end of day stock prices of individual stocks
rows of $\mathbf{A}$ → closing prices of DAX stocks on a particular day

R. Hiptmair

rev 30401, February 19, 2011

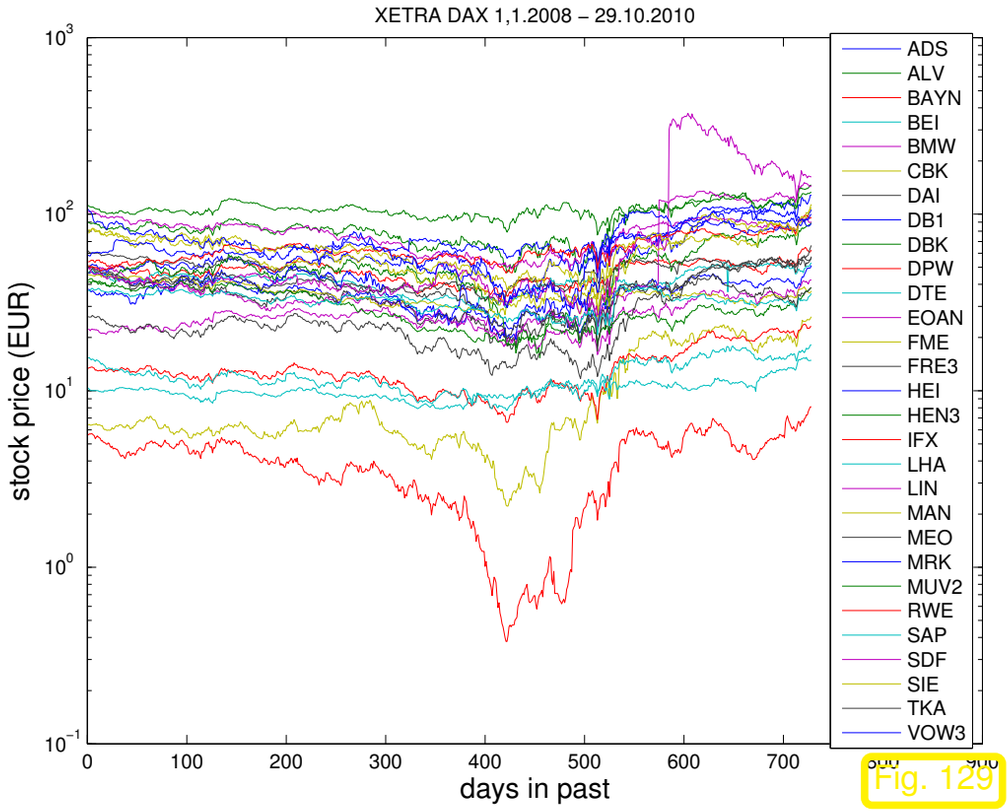


Fig. 129

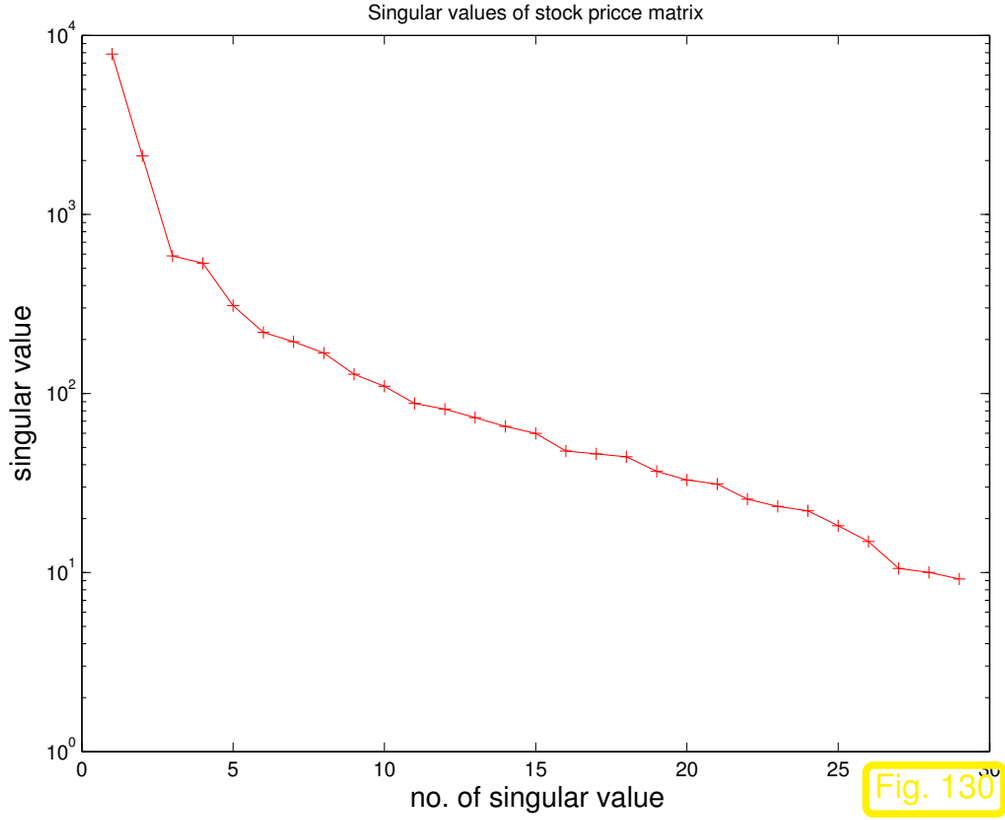


Fig. 130

We observe a pronounced decay of the singular values ($\approx$ exponential decay, logarithmic scale in Fig. 130)

➢ a few trends (corresponding to a few of the largest singular values) govern the time series.

R. Hiptmair

rev 30401, February 19, 2011

Fig. 131

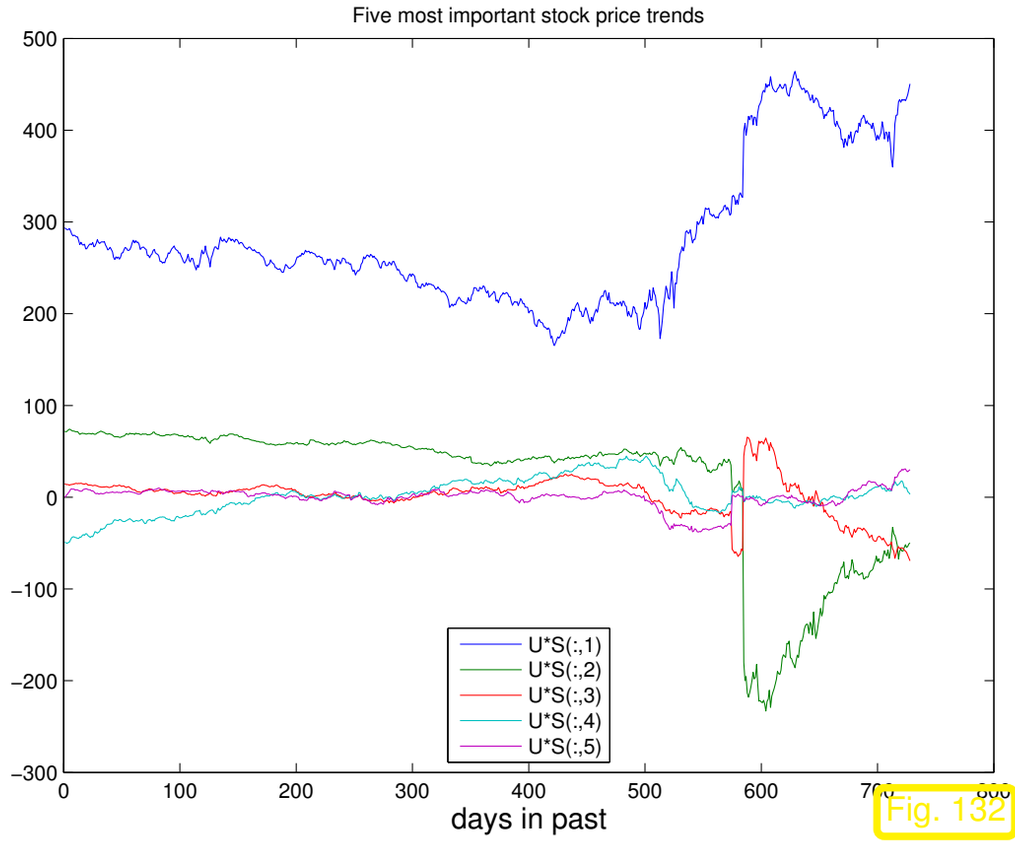


Fig. 132

Columns of $\mathbf{U}$ ($\rightarrow$ Fig. 131) in SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ provide trend vectors, *cf.* Rem. 6.5.2 & Ex. 6.5.18.

When weighted with the corresponding singular value, the importance of a trend contribution emerges, see Fig. 132

R. Hiptmair

rev 30401, February 19, 2011

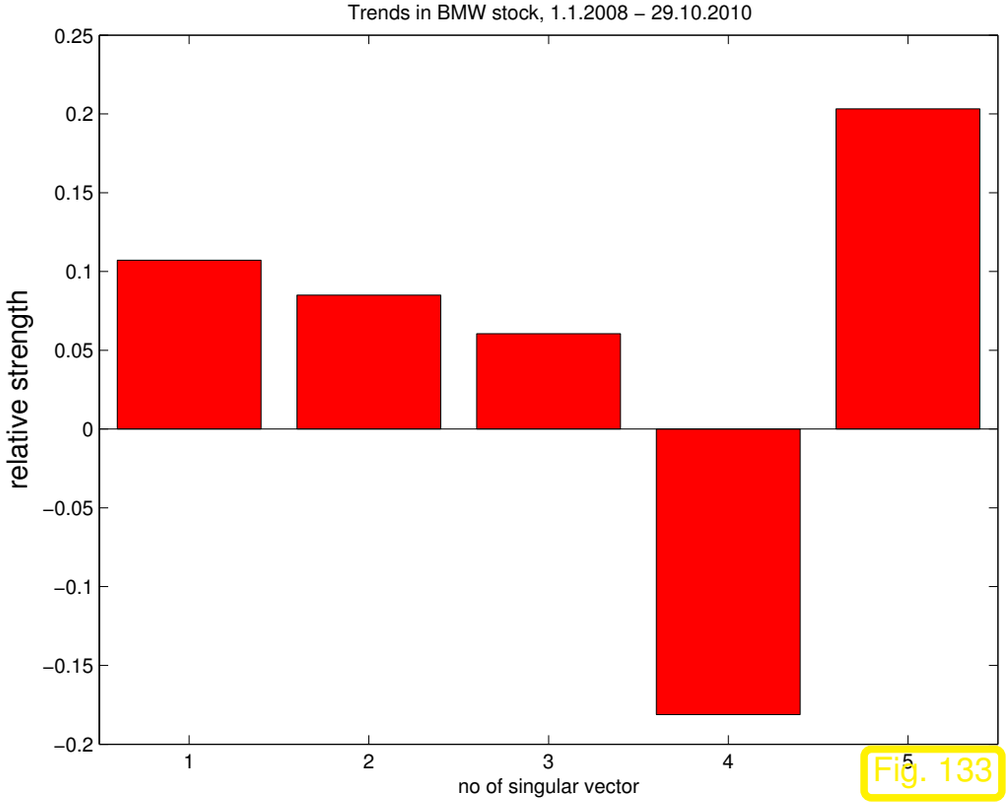


Fig. 133

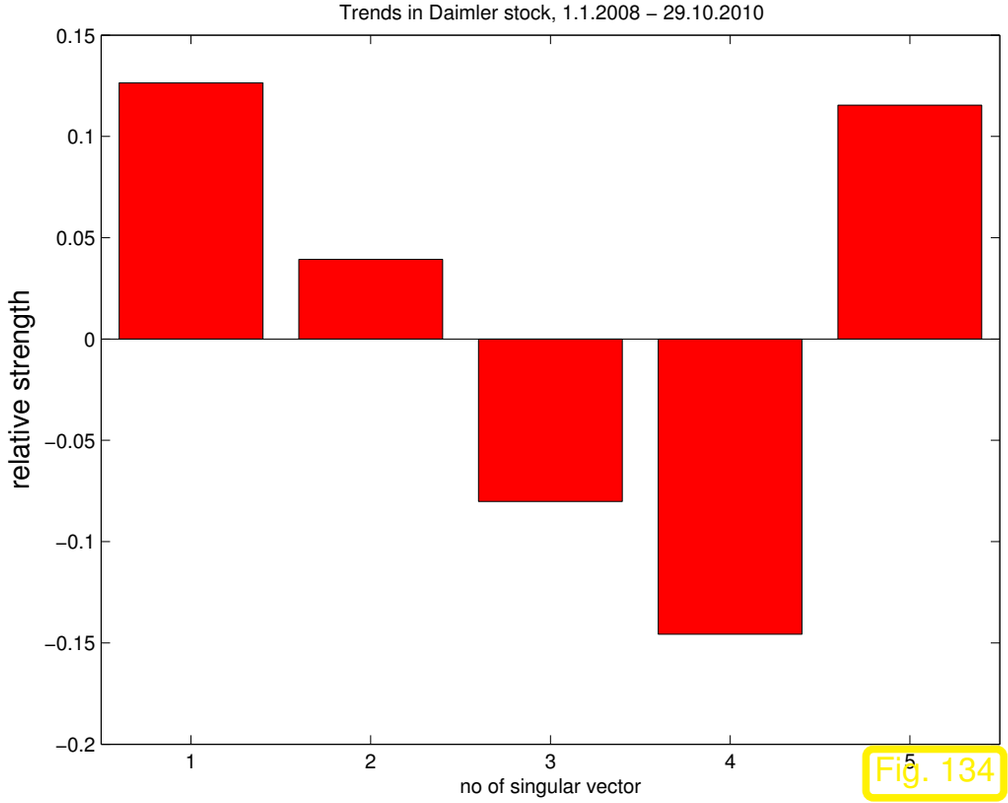


Fig. 134

Stocks of companies from the same sector of the economy should display similar contributions of major trend vectors, because their prices can be expected to be more closely correlated than stock prices in general.

Data obtained from Yahoo Finance:

```
#!/bin/csh
foreach i (ADS ALV BAYN BEI BMW CBK DAI DBK DB1 LHA DPW DTE EOAN FRE3 \
          FME HEI HEN3 IFX SDF LIN MAN MRK MEO MUV2 RWE SAP SIE TKA VOW3)
wget -O "i".csv "http://ichart.finance.yahoo.com/table.csv?s=i.DE&a=00&b=1&
     c=2008&d=09&e=30&f=2010&g=d&ignore=.csv"
sed -i -e 's/-/,/g' "i".csv
end
```

Code 6.5.20: PCA of stock prices in MATLAB

```
% Read end of day XETRA-DAX stock quotes into matrices
k = 1;
M{k} = dlmread('ADS.csv',',',1,0); leg{k} = 'ADS'; k = k + 1;
M{k} = dlmread('ALV.csv',',',1,0); leg{k} = 'ALV'; k = k + 1;
M{k} = dlmread('BAYN.csv',',',1,0); leg{k} = 'BAYN'; k = k + 1;
M{k} = dlmread('BEI.csv',',',1,0); leg{k} = 'BEI'; k = k + 1;
M{k} = dlmread('BMW.csv',',',1,0); leg{k} = 'BMW'; k = k + 1;
M{k} = dlmread('CBK.csv',',',1,0); leg{k} = 'CBK'; k = k + 1;
M{k} = dlmread('DAI.csv',',',1,0); leg{k} = 'DAI'; k = k + 1;
M{k} = dlmread('DB1.csv',',',1,0); leg{k} = 'DB1'; k = k + 1;
M{k} = dlmread('DBK.csv',',',1,0); leg{k} = 'DBK'; k = k + 1;
M{k} = dlmread('DPW.csv',',',1,0); leg{k} = 'DPW'; k = k + 1;
M{k} = dlmread('DTE.csv',',',1,0); leg{k} = 'DTE'; k = k + 1;
M{k} = dlmread('EOAN.csv',',',1,0); leg{k} = 'EOAN'; k = k + 1;
M{k} = dlmread('FME.csv',',',1,0); leg{k} = 'FME'; k = k + 1;
M{k} = dlmread('FRE3.csv',',',1,0); leg{k} = 'FRE3'; k = k + 1;
M{k} = dlmread('HEI.csv',',',1,0); leg{k} = 'HEI'; k = k + 1;
M{k} = dlmread('HEN3.csv',',',1,0); leg{k} = 'HEN3'; k = k + 1;
M{k} = dlmread('IFX.csv',',',1,0); leg{k} = 'IFX'; k = k + 1;
```

R. Hiptmair

rev 30401, February 19, 2011

```
M{k} = dlmread('LHA.csv',',',1,0); leg{k} = 'LHA'; k = k + 1;
M{k} = dlmread('LIN.csv',',',1,0); leg{k} = 'LIN'; k = k + 1;
M{k} = dlmread('MAN.csv',',',1,0); leg{k} = 'MAN'; k = k + 1;
M{k} = dlmread('MEO.csv',',',1,0); leg{k} = 'MEO'; k = k + 1;
M{k} = dlmread('MRK.csv',',',1,0); leg{k} = 'MRK'; k = k + 1;
M{k} = dlmread('MUV2.csv',',',1,0); leg{k} = 'MUV2'; k = k + 1;
M{k} = dlmread('RWE.csv',',',1,0); leg{k} = 'RWE'; k = k + 1;
M{k} = dlmread('SAP.csv',',',1,0); leg{k} = 'SAP'; k = k + 1;
M{k} = dlmread('SDF.csv',',',1,0); leg{k} = 'SDF'; k = k + 1;
M{k} = dlmread('SIE.csv',',',1,0); leg{k} = 'SIE'; k = k + 1;
M{k} = dlmread('TKA.csv',',',1,0); leg{k} = 'TKA'; k = k + 1;
M{k} = dlmread('VOW3.csv',',',1,0); leg{k} = 'VOW3';
% Data format of matrices M:
% M(:,1): year, M(:,2): month, M(:,3): day, M(:,6): end of day stock quote
% Clean data: remove/interpolate days without trading
dv = [];
for j=1:k
  M{j}(:,1) = round(366*M{j}(:,1)+31*M{j}(:,2)+M{j}(:,3));
  dv = [dv; M{j}(:,1)];
end
dv = unique(dv); mxd = max(dv)+1;
dv = mxd - dv; mdays = max(dv);
A = zeros(mdays,k);
for j=1:k, A(mxd - M{j}(:,1),j) = M{j}(:,6); end
idx = find(sum(A') ~= 0); A = A(idx,:);

for j=size(A,1):-1:2
  zidx = find(A(j-1,:) == 0);
  A(j-1,zidx) = A(j,zidx);
```


R. Hiptmair

rev 30401, February 19, 2011

```
end
for j=2:size(A,1)
  zidx = find(A(j,:) == 0);
  A(j,zidx) = A(j-1,zidx);
end

figure('name','DAX');
semilogy(A); set(gca,'xlim',[0 900]);
legend(leg,'location','east');
xlabel('days in past','fontsize',14);
ylabel('stock price (EUR)','fontsize',14);
title('XETRA DAX 1,1.2008 - 29.10.2010');
print -depsc2 '../../PICTURES/stockprizes.eps';

[U,S,V] = svd(A,0);
figure('name','singular values');
semilogy(diag(S),'r-+');
xlabel('no. of singular value','fontsize',14);
ylabel('singular value','fontsize',14);
title('Singular values of stock pricce matrix');
print -depsc2 '../../PICTURES/stocksingval.eps';

figure('name','trend vectors');
plot(U(:,1:5));
xlabel('days in past','fontsize',14);
title('Five most important stock price trends (normalized)');
legend('U(:,1)','U(:,2)','U(:,3)','U(:,4)','U(:,5)','location','south');
print -depsc2 '../../PICTURES/stocktrendsn.eps';

```

```
78 figure('name','trend vectors');
79 plot(U(:,1:5)*S(1:5,1:5));
80 xlabel('days in past','fontsize',14);
81 title('Five most important stock price trends');
82 legend('U*S(:,1)','U*S(:,2)','U*S(:,3)','U*S(:,4)','U*S(:,5)','location','south')
83 print -depsc2 '../../PICTURES/stocktrends.eps';
84
85 figure('name','trend components BMW');
86 trendcomp(V,5,5);
87 title('Trends in BMW stock, 1.1.2008 - 29.10.2010');
88 print -depsc2 '../../PICTURES/bmw.eps';
89
90 figure('name','trend components DAI');
91 trendcomp(V,7,5);
92 title('Trends in Daimler stock, 1.1.2008 - 29.10.2010');
93 print -depsc2 '../../PICTURES/dai.eps';
94
95 figure('name','trend components DBK');
96 trendcomp(V,9,5);
97 title('Trends in Deutsche Bank stock, 1.1.2008 - 29.10.2010');
98 print -depsc2 '../../PICTURES/dbk.eps';
99
00 figure('name','trend components CBK');
01 trendcomp(V,6,5);
02 title('Trends in Commerzbank stock, 1.1.2008 - 29.10.2010');
03 print -depsc2 '../../PICTURES/cbk.eps';
```


R. Hiptmair

rev 30401, February 19, 2011

- Application of SVD: extrema of quadratic forms on the unit sphere

A minimization problem on the Euclidean unit sphere $\{\mathbf{x} \in \mathbb{K}^n\colon \|\mathbf{x}\|_2 = 1\}$:

$$\text{given} \quad \mathbf{A} \in \mathbb{K}^{m,n}\,,\ m > n, \quad \text{find } \mathbf{x} \in \mathbb{K}^n,\ \|\mathbf{x}\|_2 = 1\,, \quad \|\mathbf{A}\mathbf{x}\|_2 \to \min\ . \tag{6.5.21}$$

Use that multiplication with unitary matrices preserves the 2-norm ($\to$ Thm. 2.8.6) and the singular value decomposition $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$ ($\to$ Def. 6.5.7):

$$\begin{aligned}\min_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2^2 &= \min_{\|\mathbf{x}\|_2=1} \left\|\mathbf{U}\Sigma\mathbf{V}^H\mathbf{x}\right\|_2^2 = \min_{\|\mathbf{V}^H\mathbf{x}\|_2=1} \left\|\mathbf{U}\Sigma(\mathbf{V}^H\mathbf{x})\right\|_2^2 \\ &= \min_{\|\mathbf{y}\|_2=1} \|\Sigma\mathbf{y}\|_2^2 = \min_{\|\mathbf{y}\|_2=1} (\sigma_1^2 y_1^2 + \cdots + \sigma_n^2 y_n^2) \geq \sigma_n^2\ .\end{aligned}$$

The minimum σ_n^2 is attained for $\mathbf{y} = \mathbf{e}_n \quad \Rightarrow \quad$ minimizer $\mathbf{x} = \mathbf{V}\mathbf{e}_n = (\mathbf{V})_{:,n}$.

By similar arguments:

$$\sigma_1 = \max_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2 \quad , \quad (\mathbf{V})_{:,1} = \operatorname*{argmax}_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2\ . \tag{6.5.22}$$

R. Hiptmair

rev 30401, February 19, 2011

Recall: 2-norm of the matrix $\mathbf{A}$ ($\rightarrow$ Def. 2.5.5) is defined as the maximum in (6.5.22). Thus we have proved the following theorem:

Lemma 6.5.23 (SVD and Euclidean matrix norm)**.**

- $\forall \mathbf{A} \in \mathbb{K}^{m,n}$: $\|\mathbf{A}\|_2 = \sigma_1(\mathbf{A})$,
- $\forall \mathbf{A} \in \mathbb{K}^{n,n}$ *regular*: $\operatorname{cond}_2(\mathbf{A}) = \sigma_1/\sigma_n$.

Remark: MATLAB functions `norm(A)` and `cond(A)` rely on svd(A)

- Application of SVD: *best low rank approximation*

Definition 6.5.24 (Frobenius norm)**.**

The Frobenius norm of $\mathbf{A} \in \mathbb{K}^{m,n}$ is defined as

$$\|\mathbf{A}\|_F^2 := \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \ .$$

Obvious: $\|\mathbf{A}\|_F$ invariant under unitary transformations of $\mathbf{A}$

Frobenius norm and SVD: $\boxed{\|\mathbf{A}\|_F^2 = \sum_{j=1}^{p} \sigma_j^2}$ (6.5.25)

✎ notation: $\mathcal{R}_k(m,n) := \{\mathbf{A} \in \mathbb{K}^{m,n}\colon \operatorname{rank}(\mathbf{A}) \leq k\}$, $m,n,k \in \mathbb{N}$

Theorem 6.5.26 (best low rank approximation). → *[23, Thm. 11.6]*

Let $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$ be the SVD of $\mathbf{A} \in \mathbb{K}^{m.n}$ (→ Thm. 6.5.5). For $1 \leq k \leq \operatorname{rank}(\mathbf{A})$ set $\mathbf{U}_k := \left[\mathbf{u}_{.,1}, \ldots, \mathbf{u}_{.,k}\right] \in \mathbb{K}^{m,k}$, $\mathbf{V}_k := \left[\mathbf{v}_{.,1}, \ldots, \mathbf{v}_{.,k}\right] \in \mathbb{K}^{n,k}$, $\mathbf{\Sigma}_k := \operatorname{diag}(\sigma_1, \ldots, \sigma_k) \in \mathbb{K}^{k,k}$. Then, for $\|\cdot\| = \|\cdot\|_F$ and $\|\cdot\| = \|\cdot\|_2$, holds true

$$\left\|\mathbf{A} - \mathbf{U}_k\Sigma_k\mathbf{V}_k^H\right\| \leq \|\mathbf{A} - \mathbf{F}\| \quad \forall \mathbf{F} \in \mathcal{R}_k(m,n)\ .$$

Thm 6.5.26: the rank-k-matrix that is *closest* to $\mathbf{A}$ (rank-k best approximation) in either the Euclidean matrix norm or the Frobenniusnorm (→ Def. 6.5.24) can be obtained by truncating the rank-1sum expansion (6.5.11) from the SVD of $\mathbf{A}$ after k terms.

Proof. Write $\mathbf{A}_k = \mathbf{U}_k \Sigma_k \mathbf{V}_k^H$. Obviously, with $r = \operatorname{rank} \mathbf{A}$,

$$\operatorname{rank} \mathbf{A}_k = k \quad \text{and} \quad \|\mathbf{A} - \mathbf{A}_k\| = \|\mathbf{\Sigma} - \mathbf{\Sigma}_k\| = \begin{cases} \sigma_{k+1} & \text{, for } \|\cdot\| = \|\cdot\|_2 \ , \\ \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_r^2} & \text{, for } \|\cdot\| = \|\cdot\|_F \ . \end{cases}$$

❶ Pick $\mathbf{B} \in \mathbb{K}^{n,n}$, $\operatorname{rank} \mathbf{B} = k$.

$$\dim \operatorname{Ker}(\mathbf{B}) = n - k \quad \Rightarrow \quad \operatorname{Ker}(\mathbf{B}) \cap \operatorname{Span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\} \neq \{0\} \ ,$$

where $\mathbf{v}_i$, $i = 1, \ldots, n$ are the columns of $\mathbf{V}$. For $\mathbf{x} \in \operatorname{Ker}(\mathbf{B}) \cap \operatorname{Span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$, $\|\mathbf{x}\|_2 = 1$

$$\|\mathbf{A} - \mathbf{B}\|_2^2 \geq \|(\mathbf{A} - \mathbf{B})\mathbf{x}\|_2^2 = \|\mathbf{A}\mathbf{x}\|_2^2 = \left\| \sum_{j=1}^{k+1} \sigma_j (\mathbf{v}_j^H \mathbf{x}) \mathbf{u}_j \right\|_2^2 = \sum_{j=1}^{k+1} \sigma_j^2 (\mathbf{v}_j^H \mathbf{x})^2 \geq \sigma_{j+1}^2 \ ,$$

because $\sum_{j=1}^{k+1} (\mathbf{v}_j^H \mathbf{x})^2 = 1$.

❷ Find ONB $\{\mathbf{z}_1, \ldots, \mathbf{z}_{n-k}\}$ of $\operatorname{Ker}(\mathbf{B})$ and assemble it into a matrix $\mathbf{Z} = [\mathbf{z}_1 \ldots \mathbf{z}_{n-k}] \in \mathbb{K}^{n,n-k}$

$$\|\mathbf{A} - \mathbf{B}\|_F^2 \geq \|(\mathbf{A} - \mathbf{B})\mathbf{Z}\|_F^2 = \|\mathbf{A}\mathbf{Z}\|_F^2 = \sum_{i=1}^{n-k} \|\mathbf{A}\mathbf{z}_i\|_2^2 = \sum_{i=1}^{n-k} \sum_{j=1}^{r} \sigma_j^2 (\mathbf{v}_j^H \mathbf{z}_i)^2 \qquad \square$$

Since matrix norms $\|\cdot\|_2$ and $\|\cdot\|_F$ are invariant under multiplication with orthogonal (unitary) matrices,

$$\left\|\mathbf{A} - \mathbf{U}_k\Sigma_k\mathbf{V}_k^H\right\|_2 = \sigma_{k+1}\,, \tag{6.5.27}$$

$$\left\|\mathbf{A} - \mathbf{U}_k\Sigma_k\mathbf{V}_k^H\right\|_F^2 = \sum_{j=k+1}^{\min\{m,n\}} \sigma_j^2\,. \tag{6.5.28}$$

This provides precise information about the best approximation error for rank-k-matrices.

Note: information content of a rank-k matrix $\mathbf{M} \in \mathbb{K}^{m,n}$ is equivalent to $k(m+n)$ numbers!

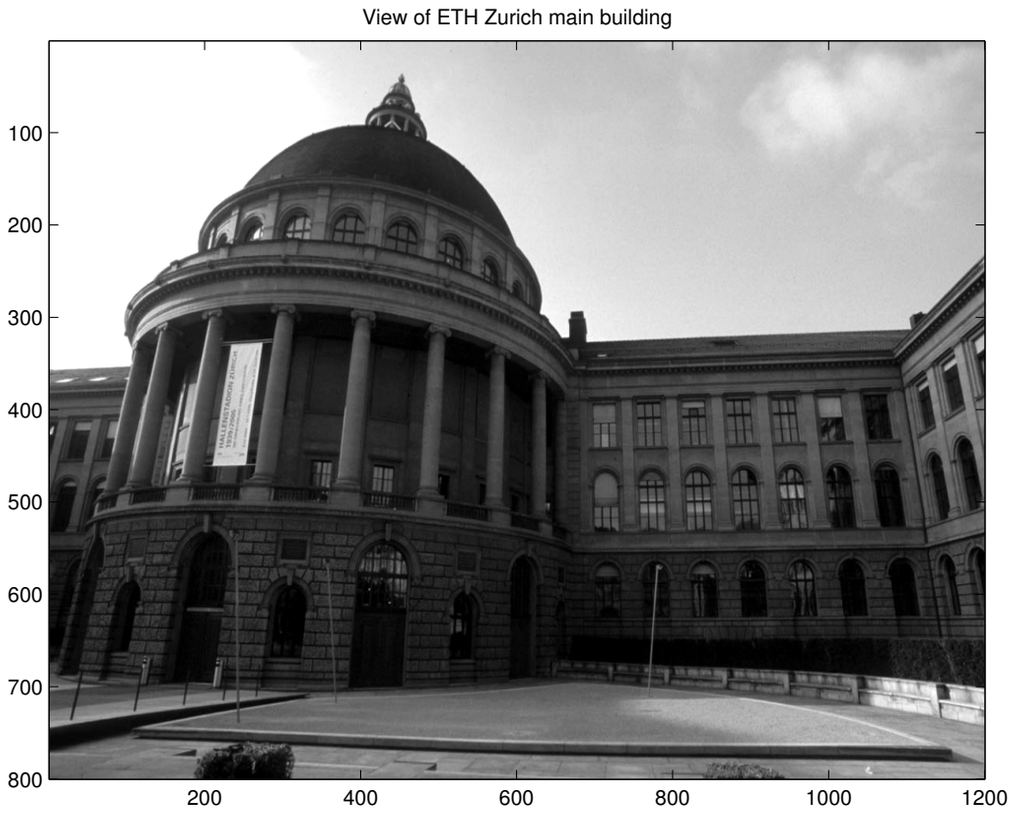
Approximation by low-rank matrices $\leftrightarrow$ matrix compression

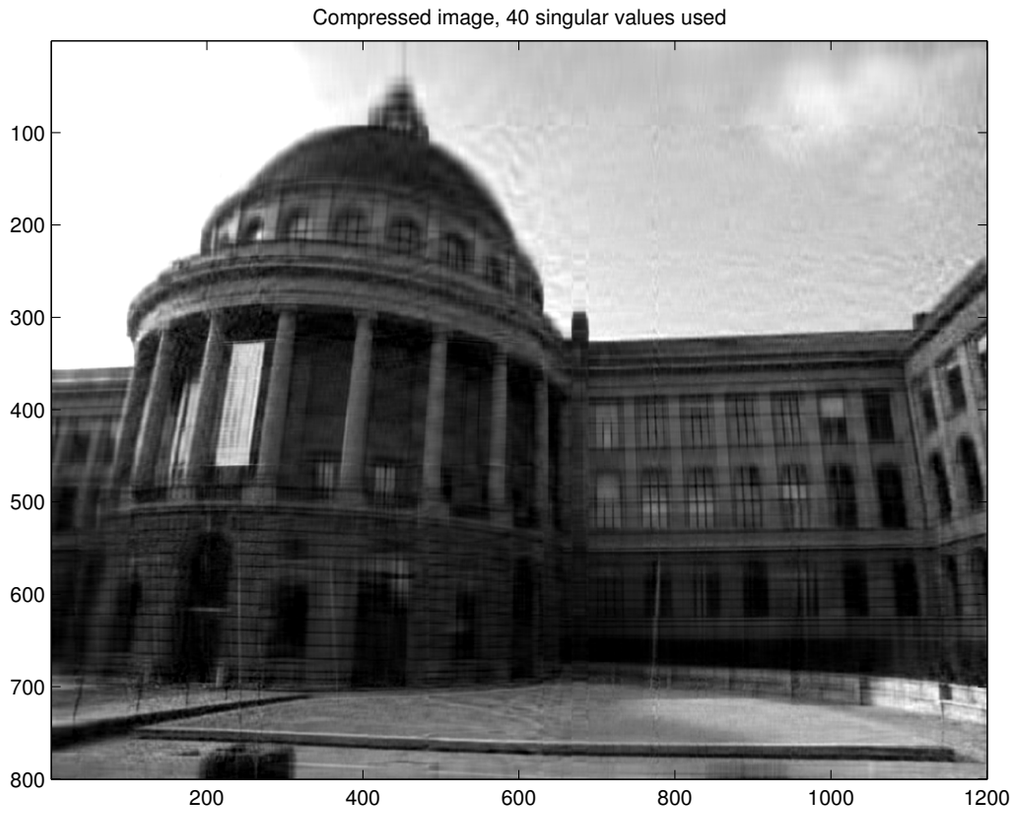
Example 6.5.29 (Image compression).

Image composed of $m \times n$ pixels (greyscale, BMP format) $\leftrightarrow$ matrix $\mathbf{A} \in \mathbb{R}^{m,n}$, $a_{ij} \in \{0,\ldots,255\}$, see Ex. 6.3.22

Code 6.5.30: SVD based image compression

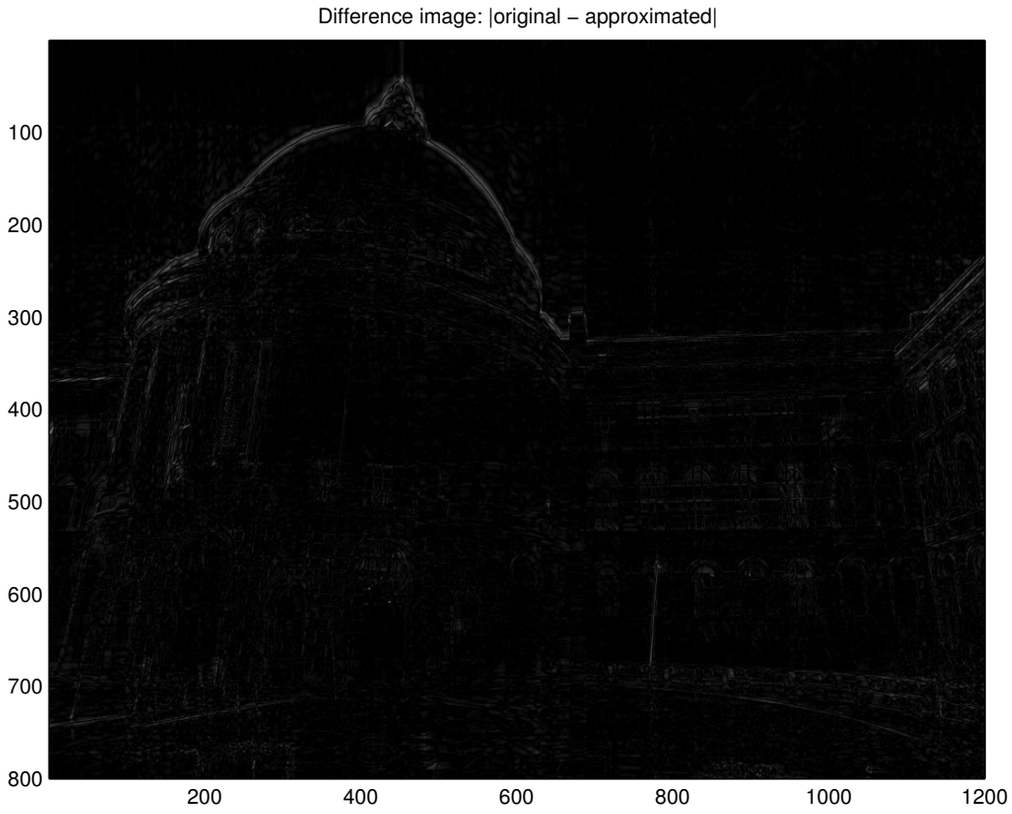
```
P = double(imread('eth.pbm'));
[m,n] = size(P); [U,S,V] = svd(P); s = diag(S);
k = 40; S(k+1:end,k+1:end) = 0; PC = U*S*V';

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');
```

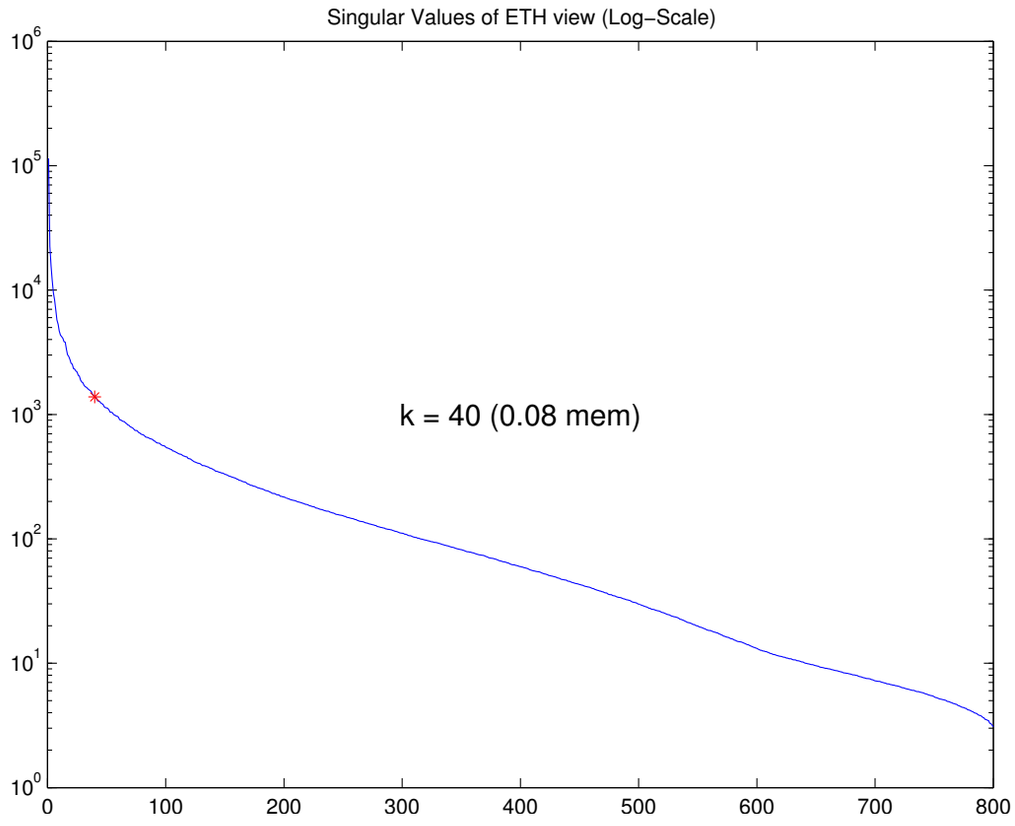




R. Hiptmair

rev 30401, February 19, 2011





R. Hiptmair

rev 30401, February 19, 2011

However: there are better and faster ways to compress images than SVD (JPEG, Wavelets, etc.)

◇

7 Least Squares

Example 7.0.1 (Least squares data fitting).

In Sect. 3.1 we discussed the reconstruction of a *parameterized function* $f(x_1, \ldots, x_n; \cdot) : D \subset \mathbb{R} \mapsto \mathbb{R}$ from data points (t_i, y_i), $i = 1, \ldots, n$, by imposing interpolation conditions (3.1.1). Necessary was that the number n of parameters agreed with number of data points. The interpolation approach is justified in the case of *highly accurate data*.

Frequently encountered: *inaccurate data* (due to *measurement errors*)

➢ interpolation approach dubious (impact of "outliers"!)

Mitigate impact of data uncertainty by

choosing fewer parameters than data points

(measurement errors can “average out”)

Non-linear least squares fitting problem:

Given:
- data points (t_i, y_i), $i = 1, \ldots, m$
- (symbolic formula) for parameterized function $f(x_1, \ldots, x_n; \cdot) : D \subset \mathbb{R} \mapsto \mathbb{R}$, $n < m$

Sought: parameter values $x_1^*, \ldots, x_n^* \in \mathbb{R}$ such that

$$(x_1^*, \ldots, x_n^*) = \operatorname*{argmin}_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^{m} |f(x_1, \ldots, x_n; t_i) - y_i|^2 \,. \tag{7.0.2}$$

◇

Example 7.0.3 (Linear data fitting). → [10, Sect. 4.1]

R. Hiptmair

rev 30401, November 22, 2010

Special case, *cf.* in Sect. 3.1, (3.1.4): Representation of f by *finite linear combination* of basis functions $b_j : D \subset \mathbb{R} \mapsto \mathbb{R}$, $j = 1, \ldots, n$:

$$f(t) = \sum_{j=1}^{n} x_j b_j(t) \quad , \quad x_j \in \mathbb{R}^d \, . \tag{7.0.4}$$

➙ $f \in$ finite dimensional function space $V_n := \operatorname{Span}\{b_1, \ldots, b_n\}$.

Linear least squares fitting problem:

Given:
- data points (t_i, y_i), $i = 1, \ldots, m$
- basis functions $b_j : I \mapsto \mathbb{K}$, $j = 1, \ldots, n$, $n < m$

Sought: coefficients $x_j^* \in \mathbb{R}$, $j = 1, \ldots, n$, such that

$$(x_1^*, \ldots, x_n^*) = \operatorname*{argmin}_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^{m} | \sum_{j=1}^{n} x_j b_j(t_i) - y_i |^2 \, . \tag{7.0.5}$$

Example 7.0.6 (Polynomial fitting).

Special variant of linear data fitting ($\rightarrow$ Ex. 7.0.3): choose f as polynomial of degree $n-1$,

$$V_n = \mathcal{P}_{n-1} \quad , \quad \textit{e.g.} \quad b_j(t) = t^{j-1} \quad \text{(monomial basis, Sect. 3.2)} .$$

➥ MATLAB-function for solving (7.0.2) in this case:

```
p = polyfit(t,y,d);
```

`d` $\hat{=}$ polynomial degree, `t` $\hat{=}$ vector $(t_i)_{i=1}^n$, `y` $\hat{=}$ vector $(y_i)_{i=1}^n$

Note: `p` $\hat{=}$ vector of monomial coefficients in MATLAB convention, see Rem. 3.2.3. ◇

R. Hiptmair

rev 30401, November 22, 2010

Example 7.0.7 (Polybomial interpolation vs. polynomial fitting).

Code 7.0.8: Fitting and interpolating polynomial

```
1 % Comparison of polynomial interpolation and polynomial fitting
```

```
% ("Quick and dirty" MATLAB implementation, see 3.4)
f = @(x) 1./(1+x.^2); % Function providing data points
d = 10;                % Polynomial degree
tip = -5+(0:d)*10/d;          % d+1 nodes for interpolation
tft = -5+(0:3*d)*10/(3*d); % 3d+1 Nodes for polynomial fitting
pip = polyfit(tip,f(tip),d); % Interpolating polynomial (degree d)
pft = polyfit(tft,f(tft),d); % Fitting polynomial (degree d)
x = -5+(0:1000)/100;
h = plot(x,f(x),'g--',...
         x,polyval(pip,x),'b-',...
         x,polyval(pft,x),'r-',...
         tip,f(tip),'b*');
set(h(1),'linewidth',2);
xlabel('{\bf t}','fontsize',14);
ylabel('{\bf y}','fontsize',14);
legend('function f','interpolating polynomial','fitting
  polynomial','location','north');
print -depsc2 '../PICTURES/interpfit.eps';
```


R. Hiptmair

rev 30401, November 22, 2010

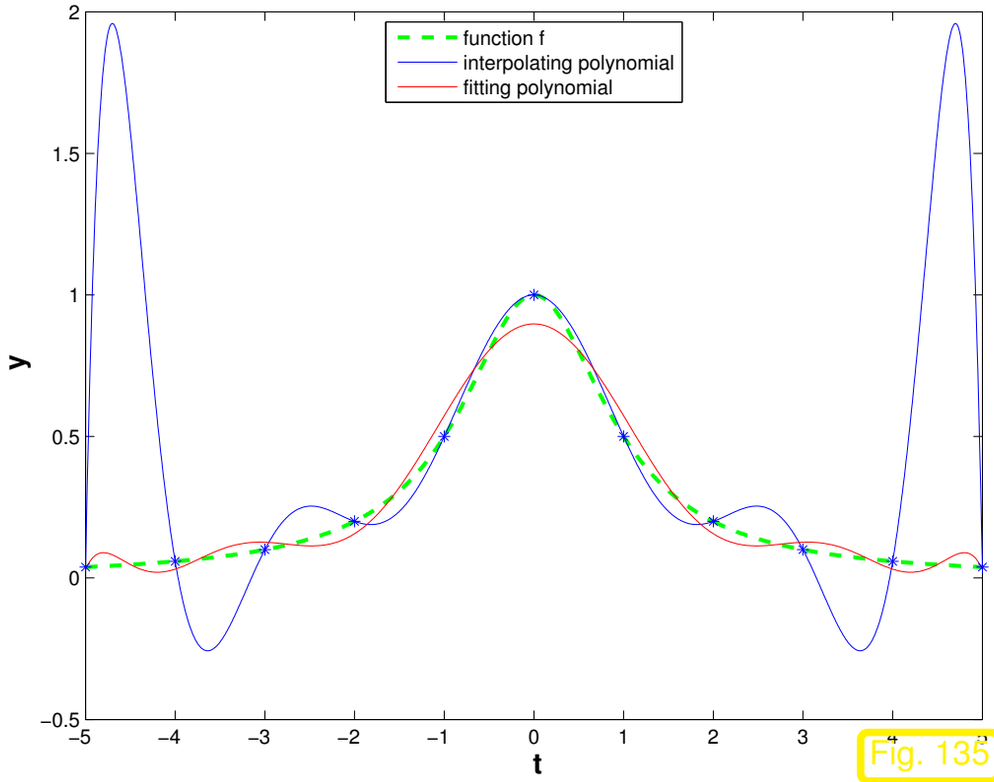


Fig. 135

Data from function $f(t) = \dfrac{1}{1+t^2}$,

- polynomial degree $d = 10$,
- interpolation through data points $(t_j, f(t_j))$, $j = 0, \ldots, d$, $t_j = -5 + j$, see Ex. 3.3.21,
- fitting to data points $(t_j, f(t_j))$, $j = 0, \ldots, 3d$, $t_j = -5 + j/3$.

Fitting helps curb oscillations that haunt polynomial interpolation!

◇

R. Hiptmair

rev 30401, November 22, 2010

Example 7.0.9 (linear regression).

☞ example for multidimensional linear least squares data fitting:

Given: *measured* data points $(\mathbf{x}_i, y_i)$, $\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}$, $i = 1, \ldots, m$, $m \geq n+1$
(y_i, $\mathbf{x}_i$ affected by measurement errors).

Known: without measurement errors data would satisfy
affine linear relationship $y = \mathbf{a}^T\mathbf{x} + c$, $\mathbf{a} \in \mathbb{R}^n$, $c \in \mathbb{R}$.

Goal: *estimate* parameters $\mathbf{a}$, c.

least squares estimate

$$(\mathbf{a}, c) = \operatorname*{argmin}_{\mathbf{p}\in\mathbb{R}^n, q\in\mathbb{R}} \sum_{i=1}^{m} |y_i - \mathbf{p}^T\mathbf{x}_i - q|^2 \tag{7.0.10}$$

linear regression for $n = 2, m = 8$ ▷.

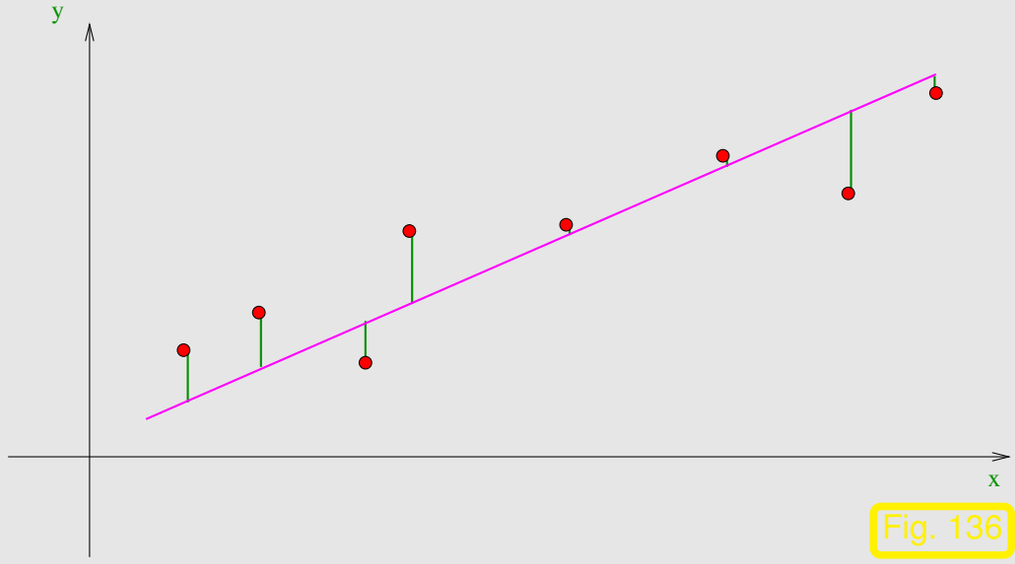


Fig. 136

Remark, see [10, Sect. 4.5]: In statistics we learn that the least squares estimate provides a maximum likelihood estimate, if the measurement errors are uniformly and independently normally

R. Hiptmair
rev 30401, November 22, 2010

distributed.

Remark 7.0.11 (Overdetermined linear systems).

In Ex. 7.0.9 we could try to find $\mathbf{a}$, c by solving the linear system of equations

$$\begin{pmatrix} \mathbf{x}_1^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^T & 1 \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} , \tag{7.0.12}$$

but in case $m > n+1$ we encounter more equations than unknowns.

In Ex. 7.0.3 the same idea leads to the linear system

$$\begin{pmatrix} b_1(t_1) & \dots & b_n(t_1) \\ \vdots & & \vdots \\ b_1(t_m) & \dots & b_n(t_m) \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} , \tag{7.0.13}$$

with the same problem in case $m > n$.

△

R. Hiptmair

rev 30401, November 22, 2010

Now we elaborate the common mathematical structure underlying the linear least squares fitting probem from Ex. 7.0.3 and Ex. 7.0.9.

(Full rank linear) least squares problem: [10, 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 $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \inf\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbb{R}^n\}$ (7.0.14)

Recast as linear least squares problem, *cf.* Rem. 7.0.11:

Ex. 7.0.9: $\mathbf{A} = \begin{pmatrix} \mathbf{x}_1^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^T & 1 \end{pmatrix} \in \mathbb{R}^{m,n+1}$, $\mathbf{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \in \mathbb{R}^n$, $\mathbf{x} = \begin{pmatrix} \mathbf{a} \\ c \end{pmatrix} \in \mathbb{R}^{n+1}$.

Ex. 7.0.3: $\mathbf{A} = \begin{pmatrix} b_1(t_1) & \dots & b_n(t_1) \\ \vdots & & \vdots \\ b_1(t_m) & \dots & b_n(t_m) \end{pmatrix} \in \mathbb{R}^{m,n}$, $\mathbf{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \in \mathbb{R}^m$, $\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$.

In both cases the residual norm $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2$ allows to gauge the quality of the model.

What if $\operatorname{rank}\mathbf{A} < n$ in (7.0.14) **?**

$$\mathbf{A} \in \mathbb{R}^{m,n}\ ,\quad m \geq n\ ,\quad \operatorname{rank}\mathbf{A} < n \overset{\text{[39, Sect. 6.1]}}{\Rightarrow} \exists \mathbf{z} \neq 0\colon\quad \mathbf{A}\mathbf{z} = 0\ . \tag{7.0.15}$$

▶ Solution of (7.0.14) cannot be unique!

➢ We need another condition to single out a unique minimizer: minimum norm condition

(General linear) least squares problem:

given: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m, n \in \mathbb{N}$, $\mathbf{b} \in \mathbb{R}^m$,
find: $\mathbf{x} \in \mathbb{R}^n$ such that

$$\text{(i)}\ \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \inf\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbb{R}^n\}, \tag{7.0.16}$$

$$\text{(ii)}\ \|\mathbf{x}\|_2 \text{ is minimal under the condition (i).}$$

Lemma 7.0.17 (Existence & uniqueness of solutions of the least squares problem)**.**
The least squares problem for $\mathbf{A} \in \mathbb{K}^{m,n}$, $\mathbf{A} \neq 0$, has a unique solution for every $\mathbf{b} \in \mathbb{K}^m$.

Proof. The proof is given by formula (7.2.6) and its derivation, see Sect. 7.2. □

MATLAB "black-box" solver for general linear least squares problems (7.0.16)

`x = A\b` ("backslash") solves (7.0.16) for $\mathbf{A} \in \mathbb{K}^{m,n}$, $m \neq n$.

Reassuring: stable ($\rightarrow$ Def.2.5.11) implementation (for dense matrices).

Remark 7.0.18 (Rank defect in linear least squares problems).

Consider linear least squares fitting problem (7.0.5) with $\mathbf{A} \in \mathbb{R}^{m,n}$ from (7.0.13).

$\operatorname{rank}(\mathbf{A}) < n$ $\Leftrightarrow$ columns of $\mathbf{A}$ are linearly dependent

$\Leftrightarrow$ at least one of the basis functions b_j is redundant, because it can not be distinguished from a linear combination of some others on $\{t_i\}$

➢ too many basis functions $\leftrightarrow$ too few data points

Usually $\mathrm{rank}(\mathbf{A}) < n$ in linear fitting hints at inadequate modelling △

Remark 7.0.19 (Pseudoinverse). → [29, Ch. 12]

By Lemma 7.0.17 the solution operator of the least squares problem (7.0.16) defines a linear mapping $\mathbf{b} \mapsto \mathbf{x}$, which has a matrix representation.

Definition 7.0.20 (Pseudoinverse). *The pseudoinverse* $\mathbf{A}^+ \in \mathbb{K}^{n,m}$ *of* $\mathbf{A} \in \mathbb{K}^{m,n}$ *is the matrix representation of the (linear) solution operator* $\mathbb{R}^m \mapsto \mathbb{R}^n$, $\mathbf{b} \mapsto \mathbf{x}$ *of the least squares problem* (7.0.16) $\|\mathbf{Ax} - \mathbf{b}\| \to \min$, $\|\mathbf{x}\| \to \min$.

MATLAB:

`P = pinv(A)` computes the pseudoinverse.

△

Remark 7.0.21 (Conditioning of the least squares problem). → [10, Sect. 4.3]

Definition 7.0.22 (Generalized condition (number) of a matrix, → Def. 2.5.26).
Let $\sigma_1 \geq \sigma_2 \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_p = 0$, $p := \min\{m,n\}$, *be the singular values (→ Def. 6.5.7) of* $\mathbf{A} \in \mathbb{K}^{m,n}$. *Then*

$$\operatorname{cond}_2(\mathbf{A}) := \frac{\sigma_1}{\sigma_r}$$

is the generalized condition (number) (w.r.t. the 2-norm) of $\mathbf{A}$.

Theorem 7.0.23. *For* $m \geq n$, $\mathbf{A} \in \mathbb{K}^{m,n}$, $\operatorname{rank}(\mathbf{A}) = n$, *let* $\mathbf{x} \in \mathbb{K}^n$ *be the solution of the least squares problem* $\|\mathbf{A}\mathbf{x} - \mathbf{b}\| \to \min$ *and* $\widehat{\mathbf{x}}$ *the solution of the perturbed least squares problem* $\|(\mathbf{A} + \Delta\mathbf{A})\widehat{\mathbf{x}} - \mathbf{b}\| \to \min$. *Then*

$$\frac{\|\mathbf{x} - \widehat{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2} \overset{\cdot}{\leq} \left(2\operatorname{cond}_2(\mathbf{A}) + \operatorname{cond}_2^2(\mathbf{A})\frac{\|\mathbf{r}\|_2}{\|\mathbf{A}\|_2\,\|\mathbf{x}\|_2}\right)\frac{\|\Delta\mathbf{A}\|_2}{\|\mathbf{A}\|_2}$$

holds, where $\mathbf{r} = \mathbf{A}\mathbf{x} - \mathbf{b}$ *is the residual.*

This means: if $\|\mathbf{r}\|_2 \ll 1$ ➤ condition of the least squares problem $\approx \mathrm{cond}_2(\mathbf{A})$
if $\|\mathbf{r}\|_2$ "large" ➤ condition of the least squares problem $\approx \mathrm{cond}_2^2(\mathbf{A})$

7.1 Normal Equations [10, Sect. 4.2], [29, Ch. 11]

Setting (7.0.14): $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, with full rank $\mathrm{rank}(\mathbf{A}) = n$

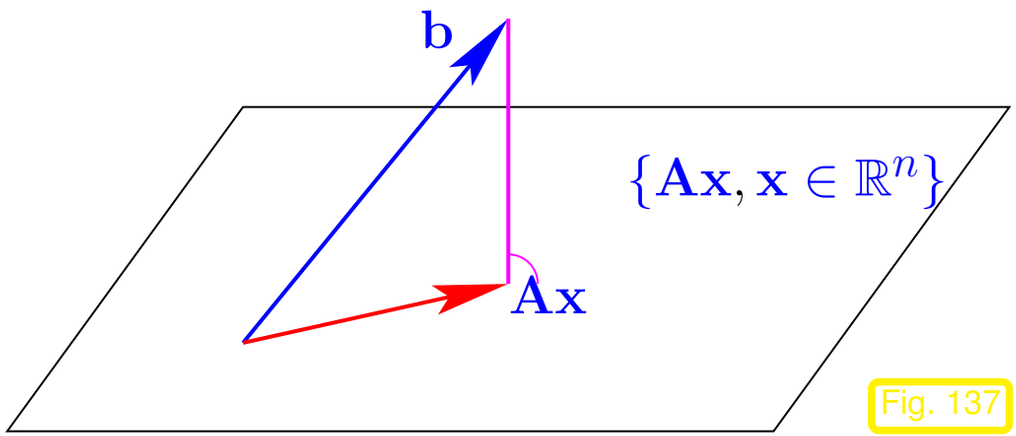


Fig. 137

Geometric interpretation of
linear least squares problem (7.0.16):

$\mathbf{x} \hat{=}$ orthogonal projection of $\mathbf{b}$ on the subspace
$\mathrm{Im}(\mathbf{A}) := \mathrm{Span}\left\{(\mathbf{A})_{:,1}, \ldots, (\mathbf{A})_{:,n}\right\}$.

Geometric interpretation: the least squares problem (7.0.16) amounts to searching the point $\mathbf{p} \in \mathrm{Im}(\mathbf{A})$ nearest (w.r.t. Euclidean distance) to $\mathbf{b} \in \mathbb{R}^m$.

Geometric intuition, see Fig. 137: $\mathbf{p}$ is the orthogonal projection of $\mathbf{b}$ onto $\mathrm{Im}(\mathbf{A})$, that is $\mathbf{b}-\mathbf{p} \perp \mathrm{Im}(\mathbf{A})$. Note the equivalence

$$\mathbf{b}-\mathbf{p} \perp \mathrm{Im}(\mathbf{A}) \quad \Leftrightarrow \quad \mathbf{b}-\mathbf{p} \perp (\mathbf{A})_{:,j}\,, \quad j=1,\ldots,n \quad \Leftrightarrow \quad \mathbf{A}^H(\mathbf{b}-\mathbf{p})=0\,,$$

Representation $\mathbf{p}=\mathbf{A}\mathbf{x}$ leads to normal equations (7.1.2). General discussion in [10, Sect. 4.6].

Solve (7.0.16) for $\mathbf{b} \in \mathbb{R}^m$

$$\mathbf{x} \in \mathbb{R}^n\!: \quad \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2 \to \min \quad \Leftrightarrow \quad f(\mathbf{x}) := \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2 \to \min\ . \tag{7.1.1}$$

A quadratic functional, *cf.* (5.1.4)

$$f(\mathbf{x}) = \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2 = \mathbf{x}^H(\mathbf{A}^H\mathbf{A})\mathbf{x} - 2\mathbf{b}^H\mathbf{A}\mathbf{x} + \mathbf{b}^H\mathbf{b}\ .$$

Minimization problem for f ➢ study gradient, *cf.* (5.1.9)

$$\mathbf{grad}\, f(\mathbf{x}) = 2(\mathbf{A}^H\mathbf{A})\mathbf{x} - 2\mathbf{A}^H\mathbf{b}\ .$$

$$\mathbf{grad}\, f(\mathbf{x}) \stackrel{!}{=} 0: \quad \boxed{\mathbf{A}^H\mathbf{A}\mathbf{x} = \mathbf{A}^H\mathbf{b}} \quad = \text{normal equation of (7.1.1)}$$

Notice: $\operatorname{rank}(\mathbf{A}) = n \;\Rightarrow\; \mathbf{A}^H\mathbf{A} \in \mathbb{R}^{n,n}$ s.p.d. ($\rightarrow$ Def. 2.7.9, [10, Rem. 4.6])

Remark 7.1.3 (Conditioning of normal equations). [10, pp. 128]

Caution: danger of instability, with SVD $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$

$$\operatorname{cond}_2(\mathbf{A}^H\mathbf{A}) = \operatorname{cond}_2(\mathbf{V}\Sigma^H\mathbf{U}^H\mathbf{U}\Sigma\mathbf{V}^H) = \operatorname{cond}_2(\Sigma^H\Sigma) = \frac{\sigma_1^2}{\sigma_n^2} = \operatorname{cond}_2(\mathbf{A})^2 \,.$$

➢ For fairly ill-conditioned $\mathbf{A}$ using the normal equations (7.1.2) to solve the linear least squares problem (7.1.1) numerically may run the risk of huge amplification of roundoff errors incurred during the computation of the right hand side $\mathbf{A}^H\mathbf{b}$: potential instability ($\rightarrow$ Def. 2.5.11) of normal equation approach.

△

R. Hiptmair

rev 30401, January 25, 2011

Example 7.1.4 (Instability of normal equations). → [10, Ex. 4.12]

Caution: loss of information in the computation of $\mathbf{A}^H\mathbf{A}$, e.g.

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

$$\Rightarrow \mathbf{A}^H\mathbf{A} = \begin{pmatrix} 1+\delta^2 & 1 \\ 1 & 1+\delta^2 \end{pmatrix}$$

```
>> A = [1 1;...
       sqrt(eps) 0;...
       0 sqrt(eps)];
>> rank(A)
     ans = 2
>> rank(A'*A)
     ans = 1
```

If $\delta < \sqrt{\texttt{eps}} \Rightarrow 1+\delta^2 = 1$ in $\mathbb{M}$, i.e. $\mathbf{A}^H\mathbf{A}$ "numeric singular", though $\operatorname{rank}(\mathbf{A}) = 2$, see Sect. 2.4, in particular Rem. 2.4.13.

◇

Remark 7.1.5 (Loss of sparsity when forming normal equations).

Another *reason not to compute* $\mathbf{A}^H\mathbf{A}$, when both m, n large:

$$\mathbf{A} \text{ sparse} \quad \not\Rightarrow \quad \mathbf{A}^T\mathbf{A} \text{ sparse}$$

- Potential memory overflow, when computing $\mathbf{A}^T\mathbf{A}$
- Squanders possibility to use efficient sparse direct elimination techniques, see Sect. 2.6.3 △

Remark 7.1.6 (Extended normal equations).

A way to avoid the computation of $\mathbf{A}^H\mathbf{A}$:

Extend normal equations (7.1.2): introduce residual $\mathbf{r} := \mathbf{A}\mathbf{x} - \mathbf{b}$ as new unknown:

$$\mathbf{A}^H\mathbf{A}\mathbf{x} = \mathbf{A}^H\mathbf{b} \quad \Leftrightarrow \quad \mathbf{B}\begin{pmatrix}\mathbf{r}\\ \mathbf{x}\end{pmatrix} := \begin{pmatrix}-\mathbf{I} & \mathbf{A}\\ \mathbf{A}^H & 0\end{pmatrix}\begin{pmatrix}\mathbf{r}\\ \mathbf{x}\end{pmatrix} = \begin{pmatrix}\mathbf{b}\\ 0\end{pmatrix} . \tag{7.1.7}$$

More general substitution $\mathbf{r} := \alpha^{-1}(\mathbf{Ax} - \mathbf{b})$, $\alpha > 0$ to improve the conditioning:

$$\mathbf{A}^H\mathbf{Ax} = \mathbf{A}^H\mathbf{b} \quad \Leftrightarrow \quad \mathbf{B}_\alpha \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \end{pmatrix} := \begin{pmatrix} -\alpha\mathbf{I} & \mathbf{A} \\ \mathbf{A}^H & 0 \end{pmatrix} \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix} . \tag{7.1.8}$$

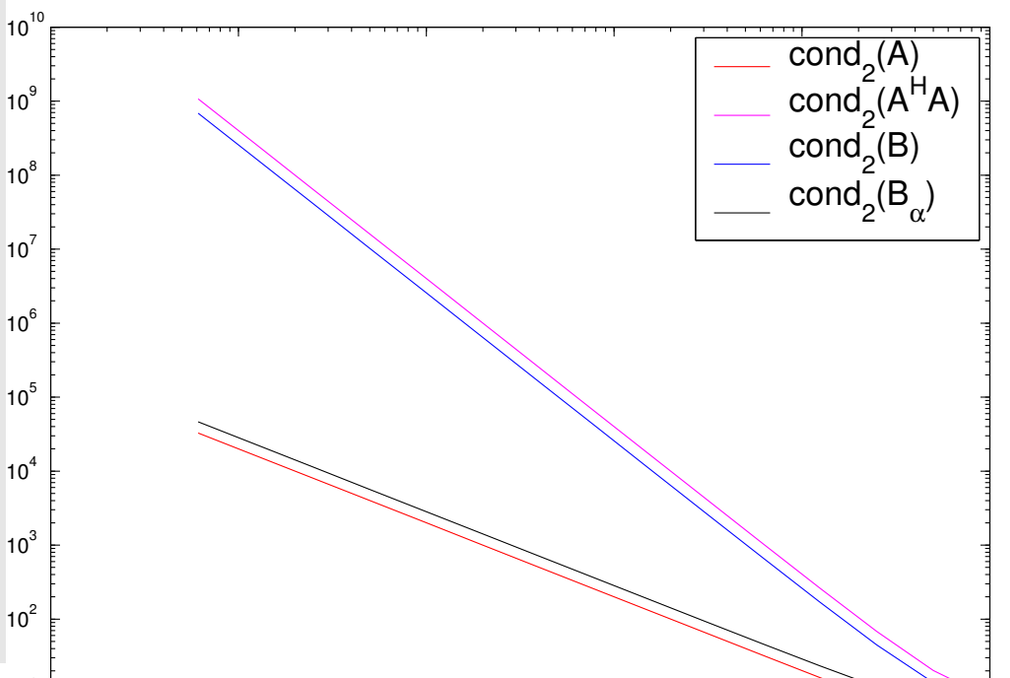
For $m, n \gg 1$, $\mathbf{A}$ sparse, both (7.1.7) and (7.1.8) lead to large sparse linear systems of equations, amenable to sparse direct elimination techniques, see Sect. 2.6.3.

Example 7.1.9 (Conditioning of the extended normal equations).

Consider (7.1.7), (7.1.8) for

$$\mathbf{A} = \begin{pmatrix} 1+\epsilon & 1 \\ 1-\epsilon & 1 \\ \epsilon & \epsilon \end{pmatrix} .$$

Plot of different condition numbers in dependence on ϵ ($\alpha = \epsilon\|\mathbf{A}\|_2 = 4\sqrt{2}$)



7.2 Orthogonal Transformation Methods

Consider the linear least squares problem (7.0.16)

$$\text{given} \quad \mathbf{A} \in \mathbb{R}^{m,n} \text{ , } \mathbf{b} \in \mathbb{R}^m \quad \text{find} \quad \mathbf{x} = \operatorname*{argmin}_{\mathbf{y} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 \ .$$

Assumption: $m \geq n$ and $\mathbf{A}$ has full (maximum) rank: $\operatorname{rank}(\mathbf{A}) = n$.

Recall Thm. 2.8.6: orthogonal (unitary) transformations ($\rightarrow$ Def. 2.8.5) leave 2-norm invariant.

R. Hiptmair

rev 30401, January 25, 2011

Idea: Transformation of $\mathbf{Ax} - \mathbf{b}$ to simpler form by *orthogonal* row transformations:

$$\operatorname*{argmin}_{\mathbf{y}\in\mathbb{R}^n} \|\mathbf{Ay} - \mathbf{b}\|_2 = \operatorname*{argmin}_{\mathbf{y}\in\mathbb{R}^n} \left\|\widetilde{\mathbf{A}}\mathbf{y} - \widetilde{\mathbf{b}}\right\|_2 ,$$

where $\widetilde{\mathbf{A}} = \mathbf{QA}$, $\widetilde{\mathbf{b}} = \mathbf{Qb}$ with orthogonal $\mathbf{Q} \in \mathbb{R}^{m,m}$.

As in the case of LSE ($\rightarrow$ Sect. 2.8): "simpler form" = triangular form.

Concrete realization of this idea by means of QR-decomposition ([10, Sect. 4.4.2], recall Sect. 2.8).

QR-decomposition: $\mathbf{A} = \mathbf{QR}$, $\mathbf{Q} \in \mathbb{K}^{m,m}$ unitary, $\mathbf{R} \in \mathbb{K}^{m,n}$ (regular) upper triangular matrix.

$$\|\mathbf{Ax} - \mathbf{b}\|_2 = \left\|\mathbf{Q}(\mathbf{Rx} - \mathbf{Q}^H\mathbf{b})\right\|_2 = \left\|\mathbf{Rx} - \widetilde{\mathbf{b}}\right\|_2 \quad , \quad \widetilde{\mathbf{b}} := \mathbf{Q}^H\mathbf{b} .$$

R. Hiptmair

rev 30401, January 25, 2011

$$\|\mathbf{Ax}-\mathbf{b}\|_2 \to \min \quad \Leftrightarrow \quad \left\| \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} - \begin{pmatrix} \widetilde{b}_1 \\ \vdots \\ \\ \\ \vdots \\ \widetilde{b}_m \end{pmatrix} \right\|_2 \to \min \ .$$

What can we do to minimize this 2-norm? Obviously, the components $n+1,\ldots,m$ of the vector inside the norm are fixed and do not depend on $\mathbf{x}$. All we can do is to make the first components $1,\ldots,n$ vanish, by choosing a suitable $\mathbf{x}$, see [10, Thm. 4.13].

R. Hiptmair

rev 30401, January 25, 2011

$$\mathbf{x} = \begin{pmatrix} \mathbf{R} \end{pmatrix}^{-1} \begin{pmatrix} \widetilde{b}_1 \\ \vdots \\ \widetilde{b}_n \end{pmatrix} \quad , \quad \text{residuum} \quad \mathbf{r} = \mathbf{Q} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \widetilde{b}_{n+1} \\ \vdots \\ \widetilde{b}_m \end{pmatrix} .$$

Note: by Thm. 2.8.6 residual norm readily available $\|\mathbf{r}\|_2 = \sqrt{\widetilde{b}_{n+1}^2 + \cdots + \widetilde{b}_m^2}$.

Implementation: successive orthogonal row transformations (by means of Householder reflections (2.8.8) for general matrices, and Givens rotations (2.8.10) for banded matrices, see Sect. 2.8 for details) of augmented matrix $(\mathbf{A}, \mathbf{b}) \in \mathbb{R}^{m,n+1}$, which is transformed into $(\mathbf{R}, \widetilde{\mathbf{b}})$

→ Code 7.2.0

$\mathbf{Q}$ need not be stored! (as in the case of linear system solving by means of QR-decomposition, see Alg. 2.8.22)

Code 7.2.1: QR-based solver for full rank linear least squares problem (7.0.14)

```
1 function [x,res] = qrlsqsolve(A,b)
2 % 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 ?)
```

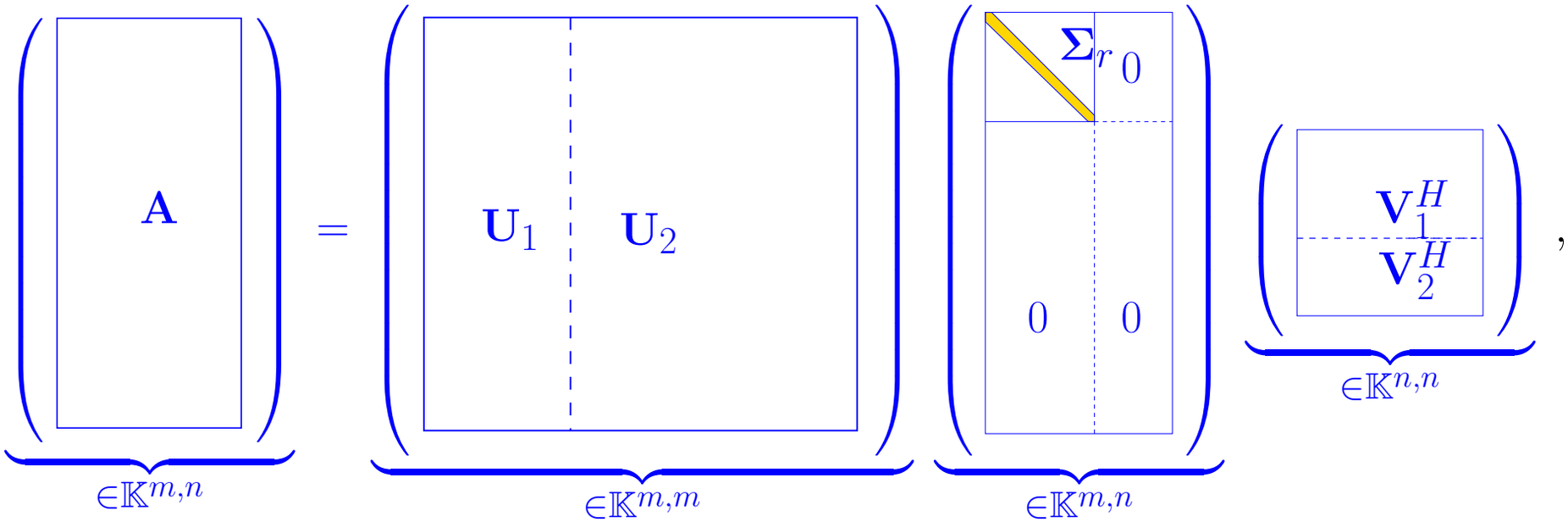
➤ A QR-based algorithm is implemented in the least-squares-solver of the MATLAB-operator "\" (for dense matrices).

Alternative: Solving linear least squares problem (7.0.16) by SVD ($\rightarrow$ Def. 6.5.7)

Most general setting: $\mathbf{A} \in \mathbb{K}^{m,n}$, $\operatorname{rank}(\mathbf{A}) = r \leq \min\{m,n\}$:

Here we drop the assumption of full rank of $\mathbf{A}$. This means that condition (ii) in the definition (7.0.16) of a linear least squares problem may be required for singling out a unique solution.

SVD: $\mathbf{A} = [\,\mathbf{U}_1 \quad \mathbf{U}_2\,] \begin{pmatrix} \mathbf{\Sigma}_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{pmatrix}$



(7.2.2)

with $\mathbf{U}_1 \in \mathbb{K}^{m,r}$, $\mathbf{U}_2 \in \mathbb{K}^{m,m-r}$ with *orthonormal* columns,
$\mathbf{\Sigma}_r = \operatorname{diag}(\sigma_1, \ldots, \sigma_r) \in \mathbb{R}^{r,r}$ (singular values, Def. 6.5.7),
$\mathbf{V}_1 \in \mathbb{K}^{n,r}$, $\mathbf{V}_2 \in \mathbb{K}^{n,n-r}$ with *orthonormal* columns.

Then we use the invariance of the 2-norm of a vector with respect to multiplication with $\mathbf{U} = [\mathbf{U}_1, \mathbf{U}_2]$, see Thm. 2.8.6, together with the fact that $\mathbf{U}$ is unitary, see Def. 2.8.5:

$$[\mathbf{U}_1, \mathbf{U}_2] \cdot \left[\begin{pmatrix} \mathbf{U}_1^H \\ \mathbf{U}_2^H \end{pmatrix} \right] = \mathbf{I} \, .$$

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \left\| [\mathbf{U}_1 \; \mathbf{U}_2] \begin{pmatrix} \mathbf{\Sigma}_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{pmatrix} \mathbf{x} - \mathbf{b} \right\|_2 = \left\| \begin{pmatrix} \mathbf{\Sigma}_r \mathbf{V}_1^H \mathbf{x} \\ 0 \end{pmatrix} - \begin{pmatrix} \mathbf{U}_1^H \mathbf{b} \\ \mathbf{U}_2^H \mathbf{b} \end{pmatrix} \right\|_2 \tag{7.2.3}$$

Logical strategy: choose $\mathbf{x}$ such that the first r components of $\begin{pmatrix} \mathbf{\Sigma}_r \mathbf{V}_1^H \mathbf{x} \\ 0 \end{pmatrix} - \begin{pmatrix} \mathbf{U}_1^H \mathbf{b} \\ \mathbf{U}_2^H \mathbf{b} \end{pmatrix}$ vanish:

$\succ$ (possibly *underdetermined*) $r \times n$ linear system $\quad \mathbf{\Sigma}_r \mathbf{V}_1^H \mathbf{x} = \mathbf{U}_1^H \mathbf{b} \, .$ (7.2.4)

To fix a unique solution in the case $r < n$ we appeal to the minimal norm condition in (7.0.16): solution $\mathbf{x}$ of (7.2.4) is unique up to contributions from

$$\mathrm{Ker}(\mathbf{V}_1^H) = \mathrm{Im}(\mathbf{V}_1)^\perp = \mathrm{Im}(\mathbf{V}_2) \, . \tag{7.2.5}$$

$\mathbf{V}$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathrm{Im}(\mathbf{V}_2)$

$\mathbf{x} \in \mathrm{Im}(\mathbf{V}_1)$ $\qquad\qquad$ 7.2.4

$$\mathbf{\Sigma}_r \underbrace{\mathbf{V}_1^H \mathbf{V}_1}_{=\mathbf{I}} \mathbf{z} = \mathbf{U}_1^H \mathbf{b} \quad \Rightarrow \quad \mathbf{z} = \mathbf{\Sigma}_r^{-1} \mathbf{U}_1^H \mathbf{b} \,.$$

solution $\boxed{\mathbf{x} = \mathbf{V}_1 \mathbf{\Sigma}_r^{-1} \mathbf{U}_1^H \mathbf{b}}$, $\|\mathbf{r}\|_2 = \left\| \mathbf{U}_2^H \mathbf{b} \right\|_2$. (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

```
1 function y = lsqsvd(A,b)
2 [U,S,V] = svd(A,0);
3 sv = diag(S);
4 r = max(find(sv/sv(1) > eps));
5 y = V(:,1:r)*(diag(1./sv(1:r))*...
6             (U(:,1:r)'*b));
```

Remark 7.2.8 (Pseudoinverse and SVD). → Rem. 7.0.19, [29, Ch. 12], [10, Sect. 4.7]

The solution formula (7.2.6) directly yields a representation of the pseudoinverse $\mathbf{A}^+$ (→ Def. 7.0.20) of any matrix $\mathbf{A}$:

Theorem 7.2.9 (Pseudoinverse and SVD).
If $\mathbf{A} \in \mathbb{K}^{m,n}$ *has the SVD decomposition* (7.2.2)*, then* $\mathbf{A}^+ = \mathbf{V}_1 \mathbf{\Sigma}_r^{-1} \mathbf{U}_1^H$ *holds.*

`scipy.linalg.pinv2(A)` and `numpy.linalg.pinv(A)`

△

Remark 7.2.10 (Normal equations vs. orthogonal transformations method).

Superior numerical stability (→ Def. 2.5.11) of orthogonal transformations methods:

► Use orthogonal transformations methods for least squares problems (7.0.16), whenever $\mathbf{A} \in \mathbb{R}^{m,n}$ *dense* and n small.

SVD/QR-factorization cannot exploit sparsity:

▶ Use normal equations in the expanded form (7.1.7)/(7.1.8), when $\mathbf{A} \in \mathbb{R}^{m,n}$ sparse ($\rightarrow$ Def. 2.6.1) and m, n big. △

Example 7.2.11 (Fit of hyperplanes).

This example studies the power and versatility of orthogonal transformations in the context of (generalized) least squares minimization problems.

The Hesse normal form of a hyperplane $\mathcal{H}$ (= affine subspace of dimension $d-1$) in $\mathbb{R}^d$ is:

$$\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^d : c + \mathbf{n}^T\mathbf{x} = 0\} \ , \quad \|\mathbf{n}\|_2 = 1 \ . \tag{7.2.12}$$

▶ Euclidean distance of $\mathbf{y} \in \mathbb{R}^d$ from the plane: $\operatorname{dist}(\mathcal{H}, \mathbf{y}) = |c + \mathbf{n}^T\mathbf{y}| \ .$ (7.2.13)

Goal: given the points $\mathbf{y}_1, \ldots, \mathbf{y}_m$, $m > d$, find $\mathcal{H} \leftrightarrow \{c \in \mathbb{R}, \mathbf{n} \in \mathbb{R}^d, \|\mathbf{n}\|_2 = 1\}$, such that

$$\sum_{j=1}^{m} \operatorname{dist}(\mathcal{H}, \mathbf{y}_j)^2 = \sum_{j=1}^{m} |c + \mathbf{n}^T\mathbf{y}_j|^2 \rightarrow \min \ . \tag{7.2.14}$$

Note: (7.2.14) $\neq$ linear least squares problem due to constraint $\|\mathbf{n}\|_2 = 1$.

$$\text{(7.2.14)} \quad \Leftrightarrow \quad \left\| \underbrace{\begin{pmatrix} 1 & y_{1,1} & \cdots & y_{1,d} \\ 1 & y_{2,1} & \cdots & y_{2,d} \\ \vdots & \vdots & & \vdots \\ 1 & y_{m,1} & \cdots & y_{m,d} \end{pmatrix}}_{=:\mathbf{A}} \begin{pmatrix} c \\ n_1 \\ \vdots \\ n_d \end{pmatrix} \right\|_2 \to \min \quad \text{under constraint} \quad \|\mathbf{n}\|_2 = 1 \; .$$

Step ❶: QR-decomposition ($\to$ Section 2.8)

$$\mathbf{A} := \begin{pmatrix} 1 & y_{1,1} & \cdots & y_{1,d} \\ 1 & y_{2,1} & \cdots & y_{2,d} \\ \vdots & \vdots & & \vdots \\ 1 & y_{m,1} & \cdots & y_{m,d} \end{pmatrix} = \mathbf{QR} \quad , \quad \mathbf{R} := \begin{pmatrix} r_{11} & r_{12} & \cdots & \cdots & r_{1,d+1} \\ 0 & r_{22} & \cdots & \cdots & r_{2,d+1} \\ \vdots & & \ddots & & \vdots \\ 0 & & & & r_{d+1,d+1} \\ 0 & \cdots & & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{m,d+1} \; .$$

$$\|\mathbf{Ax}\|_2 \to \min \quad \Leftrightarrow \quad \|\mathbf{Rx}\|_2 = \left\| \begin{pmatrix} r_{11} & r_{12} & \cdots & \cdots & r_{1,d+1} \\ 0 & r_{22} & \cdots & \cdots & r_{2,d+1} \\ \vdots & & \ddots & & \vdots \\ 0 & & & & r_{d+1,d+1} \\ 0 & \cdots & & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & & \cdots & 0 \end{pmatrix} \begin{pmatrix} c \\ n_1 \\ \vdots \\ \vdots \\ n_d \end{pmatrix} \right\|_2 \to \min \; . \quad \text{(7.2.15)}$$

R. Hiptmair

rev 30401, January 25, 2011

Step ❷ Note that necessarily (why?)

$$c \cdot r_{11} + n_1 \cdot r_{12} + \cdots + r_{1,d+1} \cdot n_d = 0 \, .$$

This insight converts (7.2.15) to

$$\left\| \begin{pmatrix} r_{22} & r_{23} & \cdots & \cdots & r_{2,d+1} \\ 0 & r_{33} & \cdots & \cdots & r_{3,d+1} \\ \vdots & & \ddots & & \vdots \\ 0 & & & & r_{d+1,d+1} \end{pmatrix} \begin{pmatrix} n_1 \\ \vdots \\ \vdots \\ n_d \end{pmatrix} \right\|_2 \to \min \quad , \quad \|\mathbf{n}\|_2 = 1 \, . \tag{7.2.16}$$

(7.2.16) = problem of type (6.5.21), minimization on the Euclidean sphere.

➢ Solve (7.2.16) using SVD !

Note: Since $r_{11} = \left\| (\mathbf{A})_{:,1} \right\|_2 = \sqrt{d+1} \neq 0 \quad \Rightarrow \quad c = -r_{11}^{-1} \sum_{j=1}^{d} r_{1,j+1} n_j$.

Code 7.2.17: (Generalized) distance fitting of a hyperplane: solution of (7.2.18)

```
1 function [c,n] = clsq(A,dim);
2 % Solves constrained linear least squares problem (7.2.18) with dim passing d
3 [m,p] = size(A);
4 if p < dim+1, error ('not enough unknowns'); end;
5 if m < dim, error ('not enough equations'); end;
```

R. Hiptmair

rev 30401, January 25, 2011

```
m = min (m, p);
R = triu (qr (A)); % First step: orthogonal transformation, see Code 7.2.0
[U,S,V] = svd(R(p-dim+1:m,p-dim+1:p)); % Solve (7.2.16)
n = V(:,dim);
c = -R(1:p-dim,1:p-dim)\R(1:p-dim,p-dim+1:p)*n;
```

7.2.18

$$\mathbf{A} \in \mathbb{K}^{m,n} \qquad \mathbf{n} \in \mathbb{R}^d \quad \mathbf{c} \in \mathbb{R}^{n-d}$$

$$\left\| \mathbf{A} \begin{pmatrix} \mathbf{c} \\ \mathbf{n} \end{pmatrix} \right\|_2 \to \min \qquad \text{with constraint } \|\mathbf{n}\|_2 = 1 \ . \tag{7.2.18}$$

7.3 Total Least Squares

R. Hiptmair

rev 30401, November 25, 2010

Given: overdetermined linear system of equations $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$, $m \geq n$.

Known: LSE solvable $\Leftrightarrow$ $\mathbf{b} \in \mathrm{Im}(\mathbf{A})$, *if* $\mathbf{A}$, $\mathbf{b}$ *were not perturbed*,

but $\mathbf{A}$, $\mathbf{b}$ are perturbed (measurement errors).

Sought: *Solvable* overdetermined system of equations $\widehat{\mathbf{A}}\mathbf{x} = \widehat{\mathbf{b}}$, $\widehat{\mathbf{A}} \in \mathbb{R}^{m,n}$, $\widehat{\mathbf{b}} \in \mathbb{R}^m$, "nearest" to $\mathbf{Ax} = \mathbf{b}$.

☞ least squares problem "turned upside down": now we are allowed to tamper with system matrix and right hand side vector!

R. Hiptmair

rev 30401, November 25, 2010

Total least squares problem:

Given: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, $\mathrm{rank}(\mathbf{A}) = n$, $\mathbf{b} \in \mathbb{R}^m$,

find: $\widehat{\mathbf{A}} \in \mathbb{R}^{m,n}$, $\widehat{\mathbf{b}} \in \mathbb{R}^m$ with

$$\left\| \underbrace{[\mathbf{A} \quad \mathbf{b}]}_{=:\mathbf{C}} - \underbrace{\left[\widehat{\mathbf{A}} \quad \widehat{\mathbf{b}}\right]}_{=:\widehat{\mathbf{C}}} \right\|_F \to \min \quad , \quad \widehat{\mathbf{b}} \in \mathrm{Im}(\widehat{\mathbf{A}}) \, . \tag{7.3.1}$$

$\widehat{\mathbf{b}} \in \mathrm{Im}(\widehat{\mathbf{A}}) \;\Rightarrow\; \mathrm{rank}(\widehat{\mathbf{C}}) = n$ ► (7.3.1) $\Rightarrow$ $\min\limits_{\mathrm{rank}(\widehat{\mathbf{C}})=n} \left\| \mathbf{C} - \widehat{\mathbf{C}} \right\|_F$.

☞ $\widehat{\mathbf{C}}$ is the rank-n best approximation of $\mathbf{C}$!

Thm. 6.5.26 ➤ use the SVD decomposition of $\mathbf{C}$ to construct $\widehat{\mathbf{C}}$:

$$\mathbf{C} = \mathbf{U}\Sigma\mathbf{V}^H = \sum_{j=1}^{n+1} \sigma_j (\mathbf{U})_{:,j} (\mathbf{V})_{:,j}^H \overset{\text{Thm. 6.5.26}}{\Longrightarrow} \widehat{\mathbf{C}} = \sum_{j=1}^{n} \sigma_j (\mathbf{U})_{:,j} (\mathbf{V})_{:,j}^H \,. \tag{7.3.2}$$

$$\overset{\mathbf{V}\text{ orthogonal}}{\Longrightarrow} \widehat{\mathbf{C}}(\mathbf{V})_{:,n+1} = 0 \,. \tag{7.3.3}$$

Recall interpretation: $\widehat{\mathbf{A}} = (\widehat{\mathbf{C}})_{1:n,1:n}$, $\widehat{\mathbf{b}} = \widehat{\mathbf{C}}_{1:n,n+1}$

➤ (7.3.3) provides solution

$$\mathbf{x} := \widehat{\mathbf{A}}^{-1}\widehat{\mathbf{b}} = (\mathbf{V})_{:,n+1} \,. \tag{7.3.4}$$

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);
```

R. Hiptmair

rev 30401, November 25, 2010

```
6 if s == 0, error('No solution'); end
7 x = -V(1:n,n+1)/s; % see (7.3.4)
```


7.4 Constrained Least Squares

Sect. 3.1: interpolation ➢ linear system of equations (3.1.7),
Ex. 7.0.3: linear data fitting ➢ linear least squares problem (7.0.5).

What if *some* data points are considered accurate?

- mix interpolation and linear least squares fitting!
- linear least squares problem with linear constraint

Linear least squares problem with linear constraint:

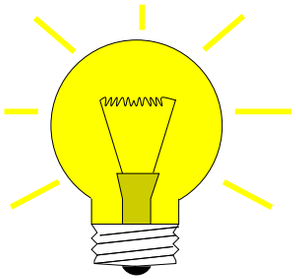
Given: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, $\mathrm{rank}(\mathbf{A}) = n$, $\mathbf{b} \in \mathbb{R}^m$,
$\mathbf{C} \in \mathbb{R}^{p,n}$, $p < n$, $\mathrm{rank}(\mathbf{C}) = p$, $\mathbf{d} \in \mathbb{R}^p$

Find: $\mathbf{x} \in \mathbb{R}^n$ with $\|\mathbf{Ax} - \mathbf{b}\|_2 \to \min$, $\boxed{\mathbf{Cx} = \mathbf{d}}$. (7.4.1)

Linear constraint

7.4.1 Solution via normal equations

Recall important technique from multidimensional calculus: Lagrange multipliers, see [52, Sect. 7.9].



Idea: coupling the constraint using the Lagrange multiplier $\mathbf{m} \in \mathbb{R}^p$

$$\mathbf{x} = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \max_{\mathbf{m}\in\mathbb{R}^p} L(\mathbf{x}, \mathbf{m}) \,, \tag{7.4.2}$$

$$L(\mathbf{x}, \mathbf{m}) := \frac{1}{2}\|\mathbf{Ax} - \mathbf{b}\|^2 + \mathbf{m}^H(\mathbf{Cx} - \mathbf{d}) \,. \tag{7.4.3}$$

R. Hiptmair

rev 30401, December 8, 2010

The simple heuristics behind Lagrange multipliers:

$$\max_{\mathbf{m}\in\mathbb{R}^p} L(\mathbf{x},\mathbf{m}) = \infty, \quad \text{in case } \mathbf{Cx} \neq \mathbf{d}!$$

➥ A minimum in (7.4.2) can only attained, if the constraint is satisfied!

(7.4.2) is called a saddle point problem.

Solution of min-max problem:

saddle point

$$F(x,m) = x^2 - 2xm \quad \triangleright$$

Note that the function is “flat” in the saddle point •

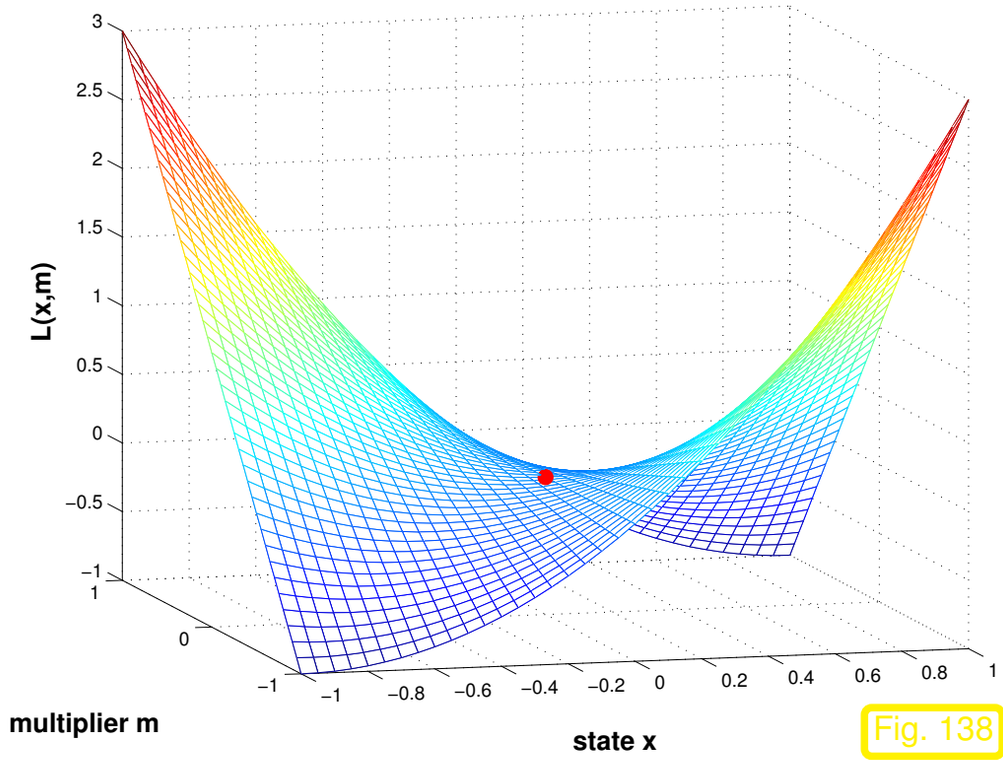


Fig. 138

Necessary (and sufficient) condition for the minimizer (→ Section 7.1)

$$\frac{\partial L}{\partial \mathbf{x}}(\mathbf{x},\mathbf{m}) = \mathbf{A}^H(\mathbf{A}\mathbf{x}-\mathbf{b}) + \mathbf{C}^H\mathbf{m} \stackrel{!}{=} 0 \quad , \quad \frac{\partial L}{\partial \mathbf{m}}(\mathbf{x},\mathbf{m}) = \mathbf{C}\mathbf{x}-\mathbf{d} \stackrel{!}{=} 0. \tag{7.4.4}$$

$$\begin{pmatrix} \mathbf{A}^H\mathbf{A} & \mathbf{C}^H \\ \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{m} \end{pmatrix} = \begin{pmatrix} \mathbf{A}^H\mathbf{b} \\ \mathbf{d} \end{pmatrix}$$

Augmented normal equations
(matrix saddle point problem) (7.4.5)

R. Hiptmair

rev 30401, December 8, 2010

Algorithm for (7.4.5) (based on block-LU-decomposition):

$$\begin{pmatrix} \mathbf{A}^H\mathbf{A} & \mathbf{C}^H \\ \mathbf{C} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{R}^H & 0 \\ \mathbf{G} & -\mathbf{S}^H \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{G}^H \\ 0 & \mathbf{S} \end{pmatrix} \quad , \quad \begin{array}{l} \mathbf{R},\mathbf{S} \in \mathbb{R}^{n,n} \text{ upper triangular matrix,} \\ \mathbf{G} \in \mathbb{R}^{p,n} \text{ .} \end{array}$$

$\mathbf{R}$ from $\mathbf{R}^H\mathbf{R} = \mathbf{A}^H\mathbf{A}$ $\rightarrow$ Cholesky decomposition $\rightarrow$ Sect. 2.7,
$\mathbf{G}$ from $\mathbf{R}^H\mathbf{G}^H = \mathbf{C}^H$ $\rightarrow$ n forward substitution $\rightarrow$ Sect. 2.2,
$\mathbf{S}$ from $\mathbf{S}^H\mathbf{S} = \mathbf{G}\mathbf{G}^H$ $\rightarrow$ Cholesky decomposition $\rightarrow$ Sect. 2.7.

Caution Sect. 7.1: the computation of $\mathbf{A}^H\mathbf{A}$ can be expensive and problematic**!**

(remedy through introduction of a new unknown $\mathbf{r} = \mathbf{A}\mathbf{x} - \mathbf{b}$, cf. (7.1.7))

$$\begin{pmatrix} -\mathbf{I} & \mathbf{A} & 0 \\ \mathbf{A}^H & 0 & \mathbf{C}^H \\ 0 & \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \\ \mathbf{m} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \\ \mathbf{d} \end{pmatrix} \quad . \tag{7.4.6}$$

R. Hiptmair

rev 30401, December 8, 2010

7.4.2 Solution via SVD

Idea: identify the subspace in which the solution can vary without violating the constraint. Since $\mathbf{C}$ has *full rank*, this subspace agrees with the nullspace/kernel of $\mathbf{C}$.

Lemma 6.5.13 ➢ SVD can be used to compute $\mathrm{Ker}(\mathbf{C})$

① Compute orthonormal basis of $\mathrm{Ker}(\mathbf{C})$ using SVD ($\rightarrow$ Lemma 6.5.13, (7.2.2)):

$$\mathbf{C} = \mathbf{U}\,[\Sigma\ 0]\begin{bmatrix}\mathbf{V}_1^H\\ \mathbf{V}_2^H\end{bmatrix}\quad,\quad \mathbf{U}\in\mathbb{R}^{p,p},\ \Sigma\in\mathbb{R}^{p,p},\ \mathbf{V}_1\in\mathbb{R}^{n,p},\ \mathbf{V}_2\in\mathbb{R}^{n,n-p}$$

$$\blacktriangleright\quad \mathrm{Ker}(\mathbf{C}) = \mathrm{Im}(\mathbf{V}_2)\ .$$

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): $\mathbf{x} = \mathbf{x}_0 + \mathbf{V}_2\mathbf{y},\quad \mathbf{y}\in\mathbb{R}^{n-p}$.

② Insert this representation in (7.4.1) ➤ standard linear least squares

$$\|\mathbf{A}(\mathbf{x}_0+\mathbf{V}_2\mathbf{y})-\mathbf{b}\|_2\to\min\quad\Leftrightarrow\quad\|\mathbf{A}\mathbf{V}_2\mathbf{y}-(\mathbf{b}-\mathbf{A}\mathbf{x}_0)\|\to\min\ .$$

R. Hiptmair

rev 30401, January 28, 2011

7.5 Non-linear Least Squares

Example 7.5.1 (Non-linear data fitting (parametric statistics)). → Ex. 7.0.1 revisited

Given: data points (t_i, y_i), $i = 1, \ldots, m$ with measurement errors.

Known: $y = f(\mathbf{x}, t)$ through a function $f : \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}$ depending non-linearly and smoothly on parameters $\mathbf{x} \in \mathbb{R}^n$.

Example: $$f(t) = x_1 + x_2 \exp(-x_3 t), \quad n = 3.$$

Determine parameters by non-linear least squares data fitting:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{m} |f(\mathbf{x}, t_i) - y_i|^2 = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \tfrac{1}{2} \|F(\mathbf{x})\|_2^2 \ , \tag{7.5.2}$$

$$\text{with} \quad F(\mathbf{x}) = \begin{pmatrix} f(\mathbf{x}, t_1) - y_1 \\ \vdots \\ f(\mathbf{x}, t_m) - y_m \end{pmatrix} .$$

◇

R. Hiptmair

rev 30401, January 28, 2011

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: $\mathbf{x}^* \in D$: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in D} \Phi(\mathbf{x})$, $\Phi(\mathbf{x}) := \frac{1}{2}\|F(\mathbf{x})\|_2^2$. (7.5.3)

Terminology: $D \hat{=}$ parameter space, $x_1, \ldots, x_n \hat{=}$ parameter.

As in the case of linear least squares problems ($\rightarrow$ Rem. 7.0.11): a non-linear least squares problem is related to an overdetermined non-linear system of equations $F(\mathbf{x}) = 0$.

As for non-linear systems of equations ($\rightarrow$ Chapter 4): existence and uniqueness of $\mathbf{x}^*$ in (7.5.3) has to be established in each concrete case!

We require "independence for each parameter": $\rightarrow$ Rem. 7.0.18

$\exists$ neighbourhood $\mathcal{U}(\mathbf{x}^*)$such that $DF(\mathbf{x})$ has full rank n $\forall\, \mathbf{x} \in \mathcal{U}(\mathbf{x}^*)$. (7.5.4)

(It means: the columns of the Jacobi matrix $DF(\mathbf{x})$ are linearly independent.)

If (7.5.4) is not satisfied, then the parameters are redundant in the sense that fewer parameters would be enough to model the same dependence (locally at $\mathbf{x}^*$), *cf.* Rem. 7.0.18.

7.5.1 (Damped) Newton method

$$\Phi(\mathbf{x}^*) = \min \quad \Rightarrow \quad \mathbf{grad}\,\Phi(\mathbf{x}) = 0, \quad \mathbf{grad}\,\Phi(\mathbf{x}) := (\frac{\partial\Phi}{\partial x_1}(\mathbf{x}), \ldots, \frac{\partial\Phi}{\partial x_n}(\mathbf{x}))^T \in \mathbb{R}^n.$$

Simple idea: use Newton's method ($\rightarrow$ Sect. 4.4) to determine a zero of $\mathbf{grad}\,\Phi : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$.

Newton iteration (4.4.1) for non-linear system of equations $\mathbf{grad}\,\Phi(\mathbf{x}) = 0$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - H\Phi(\mathbf{x}^{(k)})^{-1}\,\mathbf{grad}\,\Phi(\mathbf{x}^{(k)}) \;, \quad (H\Phi(\mathbf{x}) = \text{Hessian matrix}) \;. \tag{7.5.5}$$

R. Hiptmair

rev 30401, January 28, 2011

Expressed in terms of $F : \mathbb{R}^n \mapsto \mathbb{R}^n$ from (7.5.3):

chain rule (4.4.4) ➤ $\mathbf{grad}\,\Phi(\mathbf{x}) = DF(\mathbf{x})^T F(\mathbf{x})$,

product rule (4.4.5) ➤ $H\Phi(\mathbf{x}) := D(\mathbf{grad}\,\Phi)(\mathbf{x}) = DF(\mathbf{x})^T DF(\mathbf{x}) + \sum_{j=1}^{m} F_j(\mathbf{x}) D^2 F_j(\mathbf{x})$,

$$\Updownarrow$$

$$(H\Phi(\mathbf{x}))_{i,k} = \sum_{j=1}^{n} \frac{\partial^2 F_j}{\partial x_i \partial x_k}(\mathbf{x}) F_j(\mathbf{x}) + \frac{\partial F_j}{\partial x_k}(\mathbf{x}) \frac{\partial F_j}{\partial x_i}(\mathbf{x}) .$$

This allows to rewrite (7.5.5) in concrete terms:

R. Hiptmair

rev 30401, January 28, 2011

For Newton iterate $\mathbf{x}^{(k)}$: Newton correction $\mathbf{s} \in \mathbb{R}^n$ from LSE

$$\underbrace{\left(DF(\mathbf{x}^{(k)})^T DF(\mathbf{x}^{(k)}) + \sum_{j=1}^{m} F_j(\mathbf{x}^{(k)}) D^2 F_j(\mathbf{x}^{(k)}) \right)}_{= H\Phi(\mathbf{x}^{(k)})} \mathbf{s} = - \underbrace{DF(\mathbf{x}^{(k)})^T F(\mathbf{x}^{(k)})}_{= \mathbf{grad}\,\Phi(\mathbf{x}^{(k)})} . \qquad (7.5.6)$$

Remark 7.5.7 (Newton method and minimization of quadratic functional).

Newton's method (7.5.5) for (7.5.3) can be read as *successive minimization* of a local quadratic approximation of Φ:

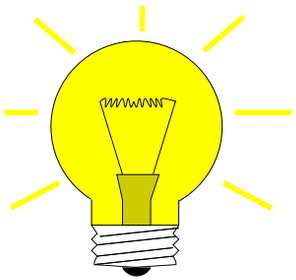
$$\Phi(\mathbf{x}) \approx Q(\mathbf{s}) := \Phi(\mathbf{x}^{(k)}) + \mathbf{grad}\,\Phi(\mathbf{x}^{(k)})^T\mathbf{s} + \frac{1}{2}\mathbf{s}^T H\Phi(\mathbf{x}^{(k)})\mathbf{s}\;, \tag{7.5.8}$$

$$\mathbf{grad}\,Q(\mathbf{s}) = 0 \quad \Leftrightarrow \quad H\Phi(\mathbf{x}^{(k)})\mathbf{s} + \mathbf{grad}\,\Phi(\mathbf{x}^{(k)}) = 0 \quad \Leftrightarrow \quad \text{(7.5.6)}\;.$$

➢ Another model function method (→ Sect. 4.3.2) with quadratic model function for Q.

△

7.5.2 Gauss-Newton method



Idea: local linearization of F: $F(\mathbf{x}) \approx F(\mathbf{y}) + DF(\mathbf{y})(\mathbf{x}-\mathbf{y})$

➢ sequence of *linear* least squares problems

$$\operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \|F(\mathbf{x})\|_2 \text{ approximated by } \underbrace{\operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \|F(\mathbf{x}_0) + DF(\mathbf{x}_0)(\mathbf{x}-\mathbf{x}_0)\|_2}_{(\spadesuit)} ,$$

where $\mathbf{x}_0$ is an approximation of the solution $\mathbf{x}^*$ of (7.5.3).

$$(\spadesuit) \quad \Leftrightarrow \quad \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x}-\mathbf{b}\| \quad \text{with} \quad \mathbf{A} := DF(\mathbf{x}_0) \in \mathbb{R}^{m,n} , \quad \mathbf{b} := -F(\mathbf{x}_0) + DF(\mathbf{x}_0)\mathbf{x}_0 \in \mathbb{R}^m .$$

This is a linear least squares problem of the form (7.0.16).

Note: (7.5.4) $\Rightarrow$ $\mathbf{A}$ has full rank, if $\mathbf{x}_0$ sufficiently close to $\mathbf{x}^*$.

Note: Approach different from local quadratic approximation of Φ underlying Newton's method for (7.5.3), see Sect. 7.5.1, Rem. 7.5.7.

R. Hiptmair

rev 30401, January 28, 2011

Gauss-Newton iteration (under assumption (7.5.4))

Initial guess $\mathbf{x}^{(0)} \in D$

$$\mathbf{x}^{(k+1)} := \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \left\| F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) \right\|_2 . \tag{7.5.9}$$

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

```
1 function x = gn(x,F,J,tol)
2 s = J(x)\F(x);                          %
3 x = x-s;
4 while (norm(s) > tol*norm(x))           %
5   s = J(x)\F(x);                        %
6   x = x-s;
7 end
```

Comments on Code 7.5.9:

☞ Argument `x` passes initial guess $\mathbf{x}^{(0)} \in \mathbb{R}^n$, argument `F` must be a *handle* to a function $F : \mathbb{R}^n \mapsto \mathbb{R}^m$, argument `J` provides the Jacobian of F, namely $DF : \mathbb{R}^n \mapsto \mathbb{R}^{m,n}$, argument `tol` specifies the tolerance for termination

☞ Line 4: iteration terminates if relative norm of correction is below threshold specified in `tol`.

Note: Code 7.5.9 also implements Newton's method (→ Sect. 4.4.1) in the case $m = n$!

Summary:

Advantage of the Gauss-Newton method : second derivative of F not needed.
Drawback of the Gauss-Newton method : no local quadratic convergence.

Example 7.5.11 (Non-linear data fitting (II)). → Ex. 7.5.1

Non-linear data fitting problem (7.5.2) for $f(t) = x_1 + x_2 \exp(-x_3 t)$.

$$F(\mathbf{x}) = \begin{pmatrix} x_1 + x_2 \exp(-x_3 t_1) - y_1 \\ \vdots \\ x_1 + x_2 \exp(-x_3 t_m) - y_m \end{pmatrix} : \mathbb{R}^3 \mapsto \mathbb{R}^m \, , \, DF(\mathbf{x}) = \begin{pmatrix} 1 & e^{-x_3 t_1} & -x_2 t_1 e^{-x_3 t_1} \\ \vdots & \vdots & \vdots \\ 1 & e^{-x_3 t_m} & -x_2 t_m e^{-x_3 t_m} \end{pmatrix}$$

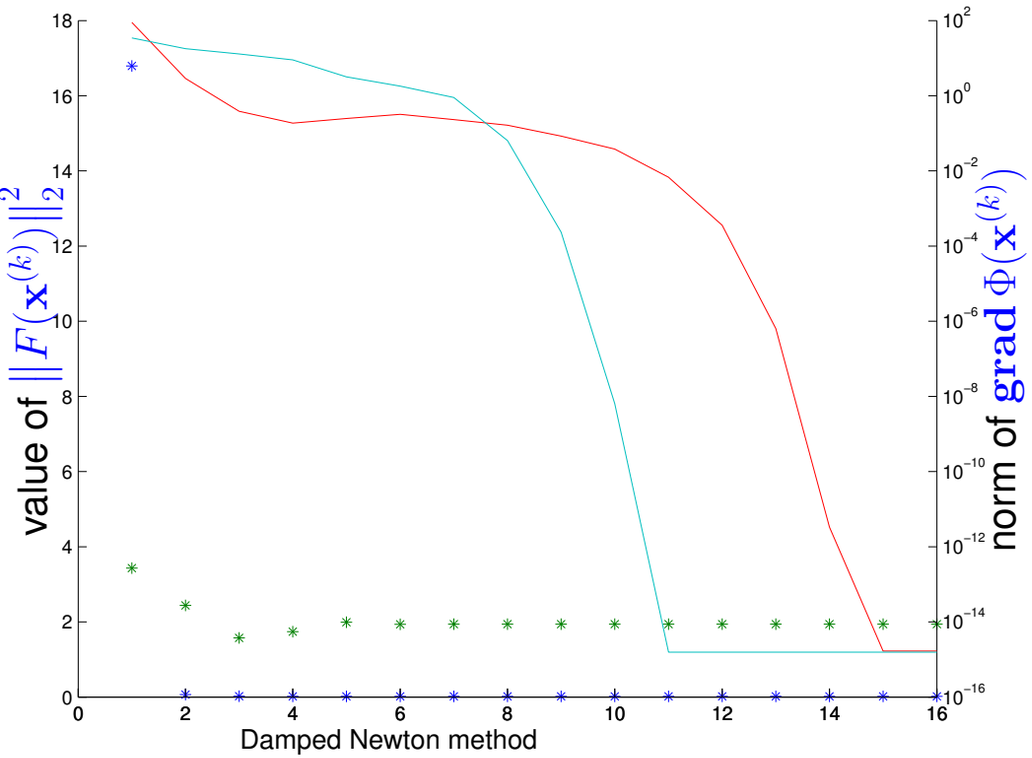
Numerical experiment:

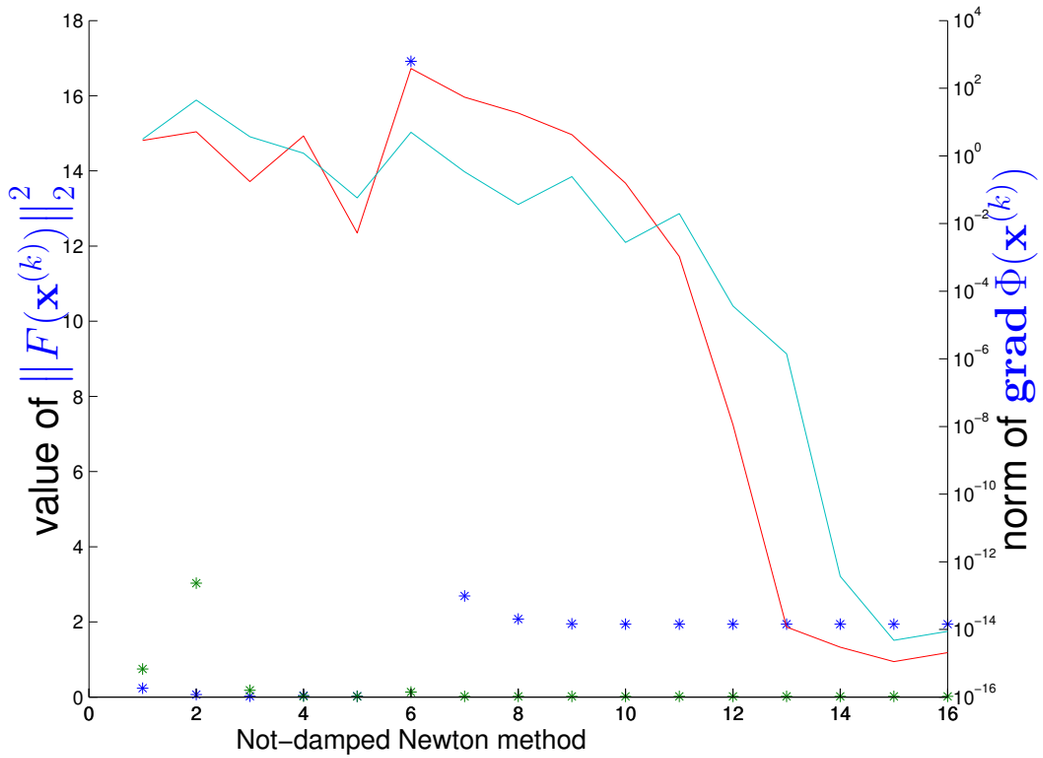
convergence of the Newton method, damped Newton method (→ Section 4.4.4) and Gauss-Newton method for different initial values

```
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);
```


R. Hiptmair

rev 30401, January 28, 2011





Convergence behaviour of the Newton method:

initial value $(1.8, 1.8, 0.1)^T$ (red curve) ➤ Newton method caught in local minimum,

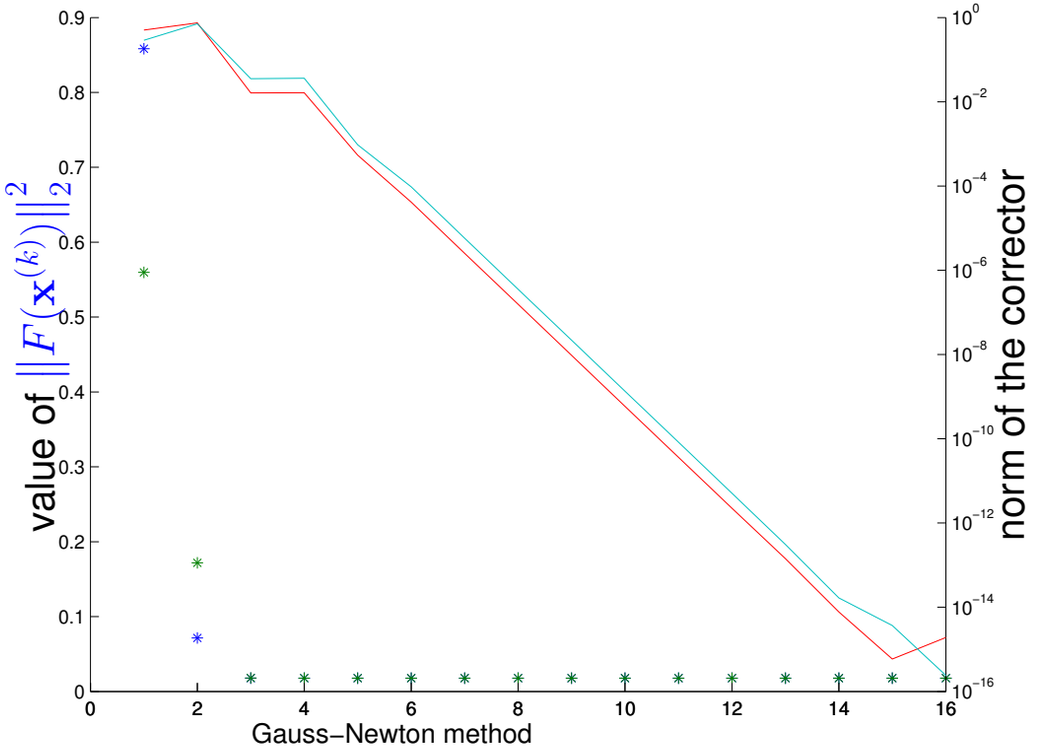
initial value $(1.5, 1.5, 0.1)^T$ (cyan curve) ➤ fast (locally quadratic) convergence.

Gauss-Newton method:

initial value $(1.8, 1.8, 0.1)^T$ (red curve),
initial value $(1.5, 1.5, 0.1)^T$ (cyan curve),

convergence in both cases.

Notice: linear convergence.



◇

7.5.3 Trust region method (Levenberg-Marquardt method)

R. Hiptmair

rev 30401, January 28, 2011

As in the case of Newton's method for non-linear systems of equations, see Sect. 4.4.4: often overshooting of Gauss-Newton corrections occurs.

Remedy as in the case of Newton's method: damping.

Idea: damping of the Gauss-Newton correction in (7.5.9) using a penalty term

instead of $\left\| F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})\mathbf{s} \right\|^2$ minimize $\left\| F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})\mathbf{s} \right\|^2 + \lambda \left\| \mathbf{s} \right\|_2^2$.

$\lambda > 0 \hat{=}$ penalty parameter (how to choose it **?** $\rightarrow$ heuristic)

$$\lambda = \gamma \left\| F(\mathbf{x}^{(k)}) \right\|_2 \quad , \quad \gamma := \begin{cases} 10 & \text{, if } \left\| F(\mathbf{x}^{(k)}) \right\|_2 \geq 10 \text{ ,} \\ 1 & \text{, if } 1 < \left\| F(\mathbf{x}^{(k)}) \right\|_2 < 10 \text{ ,} \\ 0.01 & \text{, if } \left\| F(\mathbf{x}^{(k)}) \right\|_2 \leq 1 \text{ .} \end{cases}$$

Modified (regularized) equation for the corrector $\mathbf{s}$:

$$\left(DF(\mathbf{x}^{(k)})^T DF(\mathbf{x}^{(k)}) + \lambda \mathbf{I} \right) \mathbf{s} = -DF(\mathbf{x}^{(k)}) F(\mathbf{x}^{(k)}) \text{ .} \tag{7.5.12}$$

8 Filtering Algorithms

Perspective of signal processing:

vector $\mathbf{x} \in \mathbb{R}^n$ $\leftrightarrow$ finite discrete (= sampled) signal.

$X = X(t) \hat{=}$ *time-continuous* signal, $0 \le t \le T$,

"sampling": $x_j = X(j\Delta t)$, $j = 0, \ldots, n-1$,
$n \in \mathbb{N}$, $n\Delta t \le T$.

$\Delta t > 0 \hat{=}$ time between samples.

Sampled values arranged in a vector $\mathbf{x} = (x_0, \ldots, x_{n-1})^T \in \mathbb{R}^n$.

Note: vector indices $0, \ldots, n-1$ **!**
("C-style indexing").

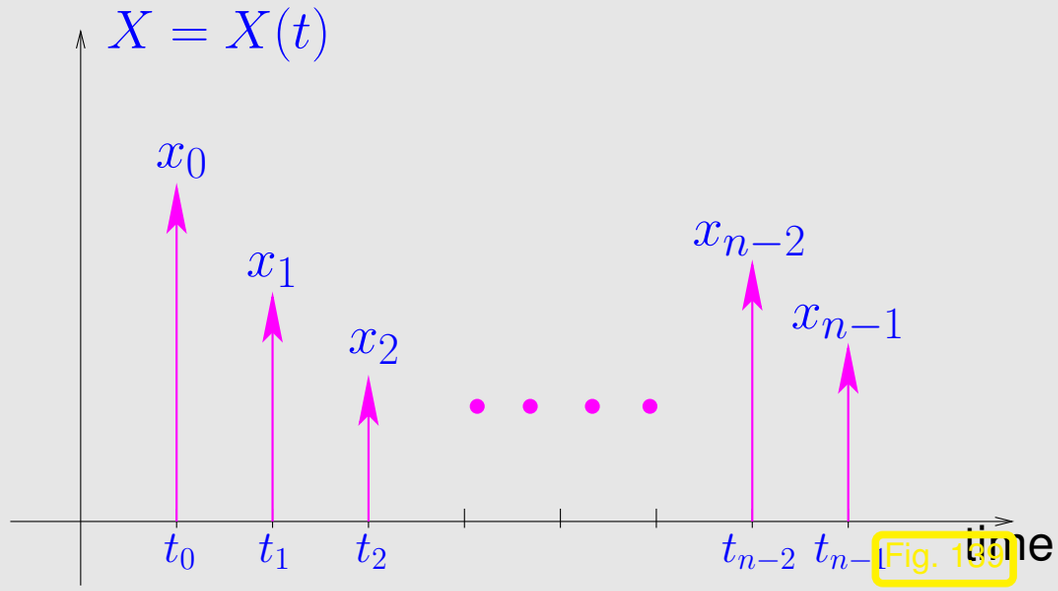
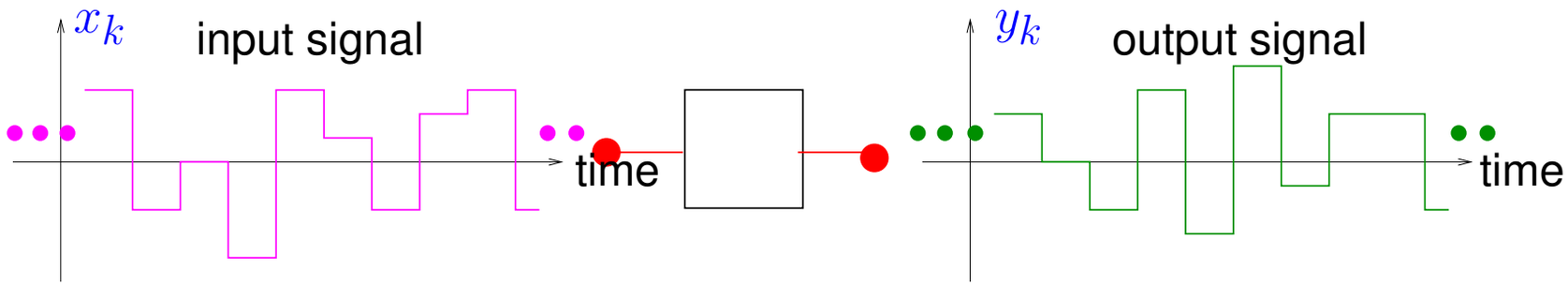


Fig. 199

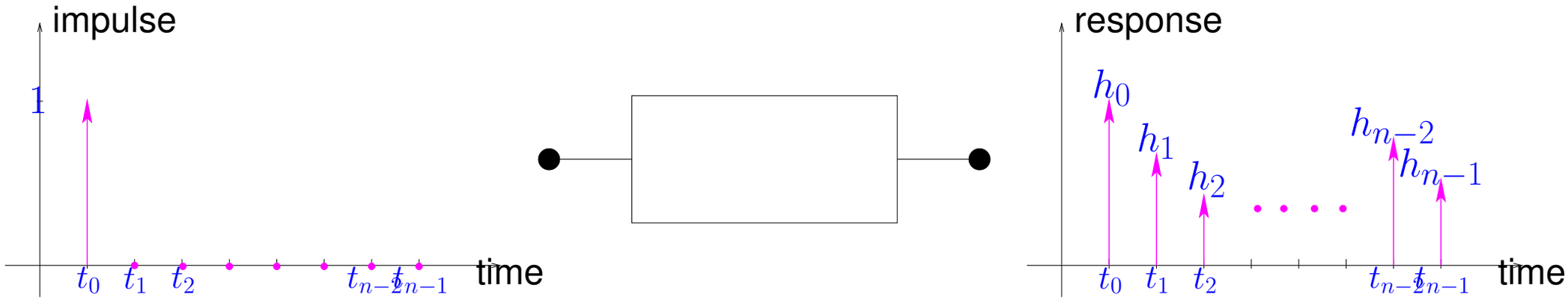
R. Hiptmair
rev 30401, January 28, 2011

8.1 Discrete convolutions

Example 8.1.1 (Discrete finite linear time-invariant causal channel (filter)).



Impulse response of channel (filter): $\mathbf{h} = (h_0, \ldots, h_{n-1})^T$



R. Hiptmair

rev 30401, January 28, 2011

Impulse response = output when filter is fed with a single impulse of strength one, corresponding to input $\mathbf{e}_1$ (first unit vector).

We study a *finite linear time-invariant causal channel (filter)*:
(widely used model for digital communication channels, e.g. in wireless communication theory)

finite: impulse response of finite duration ➢ it can be described by a vector $\mathbf{h}$ of finite length n.

time-invariant: when input is shifted in time, output is shifted by the same amount of time.

linear: input $\mapsto$ output-map is linear

$$\text{output}(\mu \cdot \text{signal 1} + \lambda \cdot \text{signal 2}) = \mu \cdot \text{output}(\text{signal 1}) + \lambda \cdot \text{output}(\text{signal 2}) \ .$$

causal (or physical, or nonanticipative): output depends only on past and present inputs, not on the future.

The output for finite length input $\mathbf{x} = (x_0, \ldots, x_{n-1})^T \in \mathbb{R}^n$ is

a superposition of x_j-weighted $j\Delta t$-shifted impulse responses

R. Hiptmair

rev 30401, January 28, 2011

channel is causal!

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j \;, \quad k = 0, \ldots, 2n-2 \quad (h_j := 0 \text{ for } j < 0 \text{ and } j \geq n) \;. \tag{8.1.2}$$

$\mathbf{x} = (x_0, \ldots, x_{n-1})^T \in \mathbb{R}^n \hat{=}$ input signal $\mapsto$ $\mathbf{y} = (y_0, \ldots, y_{2n-2})^T \in \mathbb{R}^{2n-1} \hat{=}$ output signal.

Matrix notation of (8.1.2):

$$\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \\ \vdots \\ y_{2n-2} \end{pmatrix} = \begin{pmatrix} h_0 & 0 & \cdots & & 0 \\ h_1 & \ddots & & & \\ \vdots & & & & 0 \\ h_{n-1} & & & h_1 & h_0 \\ 0 & \ddots & & & \vdots \\ \vdots & & & & \\ 0 & & \cdots & 0 & h_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ \\ \\ \\ \vdots \\ x_{n-1} \end{pmatrix} \;. \tag{8.1.3}$$

R. Hiptmair

rev 30401, January 28, 2011

Example 8.1.4 (Multiplication of polynomials).

$$p(z) = \sum_{k=0}^{n-1} a_k z^k \,, \quad q(z) = \sum_{k=0}^{n-1} b_k z^k \quad \blacktriangleright \quad (pq)(z) = \sum_{k=0}^{2n-2} \underbrace{\left(\sum_{j=0}^{k} a_j b_{k-j} \right)}_{=:c_k} z^k \tag{8.1.5}$$

➢ coefficients of product polynomial by discrete convolution of coefficients of polynomial factors!

Both in (8.1.2) and (8.1.5) we recognize the same pattern of a particular *bi-linear* combination of

- discrete signals in Ex. 8.1.1,
- polynomial coefficient sequences in Ex. 8.1.4.

Definition 8.1.6 (Discrete convolution)**.**
Given $\mathbf{x} = (x_0, \ldots, x_{n-1})^T \in \mathbb{K}^n$, $\mathbf{h} = (h_0, \ldots, h_{n-1})^T \in \mathbb{K}^n$ *their discrete convolution* (ger.: *diskrete Faltung) is the vector* $\mathbf{y} \in \mathbb{K}^{2n-1}$ *with components*

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j \;, \quad k = 0, \ldots, 2n-2 \quad (h_j := 0 \text{ for } j < 0) \;. \tag{8.1.7}$$

✎ Notation for discrete convolution (8.1.7): $\mathbf{y} = \mathbf{h} * \mathbf{x}$.

R. Hiptmair

rev 30401, January 28, 2011

Defining $x_j := 0$ for $j < 0$, we find that *discrete convolution is commutative*:

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j = \sum_{l=0}^{n-1} h_l x_{k-l} \;, \quad k = 0, \ldots, 2n-2 \;, \qquad \text{(that is, } \quad \mathbf{h} * \mathbf{x} = \mathbf{x} * \mathbf{h} \quad \text{)} \;,$$

obtained by index transformation $l \leftarrow k - j$.

Remark 8.1.8 (Convolution of sequences).

The notion of a discrete convolution of Def. 8.1.6 naturally extends to sequences $\mathbb{N}_0 \mapsto \mathbb{K}$: the (discrete) convolution of two sequences $(x_j)_{j \in \mathbb{N}_0}$, $(y_j)_{j \in \mathbb{N}_0}$ is the sequence $(z_j)_{j \in \mathbb{N}_0}$ defined by

$$z_k := \sum_{j=0}^{k} x_{k-j} y_j = \sum_{j=0}^{k} x_j y_{k-j} \;, \quad k \in \mathbb{N}_0 \;.$$

△

Example 8.1.9 (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 $x_0, \ldots, x_{n-1} \leftrightarrow$ vector $\mathbf{x} = (x_0, \ldots, x_{n-1})^T \in \mathbb{R}^n$.

Whenever the input signal of a time-invariant filter is n-periodic, so will be the output signal. Thus, in the n-periodic setting, a causal *linear* time-invariant filter will give rise to a *linear* mapping $\mathbb{R}^n \mapsto \mathbb{R}^n$ according to

$$y_k = \sum_{j=0}^{n-1} p_{k-j} x_j \quad \text{for some} \quad p_0, \ldots, p_{n-1} \in \mathbb{R} \,. \tag{8.1.10}$$

Note: $p_0, \ldots, p_{n-1}$ does not agree with the impulse response of the filter.

Matrix notation:

$$\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \\ \vdots \\ y_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} p_0 & p_{n-1} & p_{n-2} & \cdots & & \cdots & p_1 \\ p_1 & p_0 & p_{n-1} & & & & \vdots \\ p_2 & p_1 & p_0 & \ddots & & & \\ \vdots & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ \vdots & & & & \ddots & \ddots & p_{n-1} \\ p_{n-1} & \cdots & & & & p_1 & p_0 \end{pmatrix}}_{=:\mathbf{P}} \begin{pmatrix} x_0 \\ \vdots \\ \\ \\ \\ \vdots \\ x_{n-1} \end{pmatrix} . \tag{8.1.11}$$

$(\mathbf{P})_{ij} = p_{i-j}$, $1 \le i,j \le n$, with $p_j := p_{j+n}$ for $1-n \le j < 0$.

Definition 8.1.12 (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

$$(z_k) := (x_k) *_n (y_k) \quad , \quad z_k := \sum_{j=0}^{n-1} x_{k-j} y_j = \sum_{j=0}^{n-1} y_{k-j} x_j \ , \quad k \in \mathbb{Z} \ .$$

✎ notation for discrete periodic convolution: $(x_k) *_n (y_k)$

Since n-periodic sequences can be identified with vectors in $\mathbb{K}^n$ (see above), we can also introduce the discrete periodic convolution of vectors:

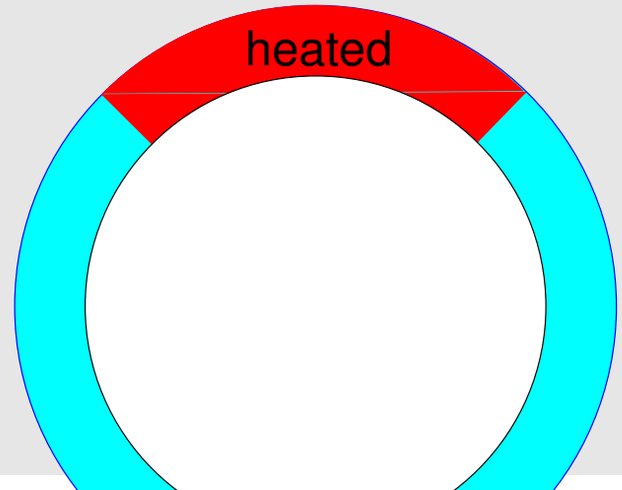
Def. 8.1.12 ➢ discrete periodic convolution of vectors: $\mathbf{z} = \mathbf{x} *_n \mathbf{y} \in \mathbb{K}^n$, $\mathbf{x}, \mathbf{y} \in \mathbb{K}^n$.

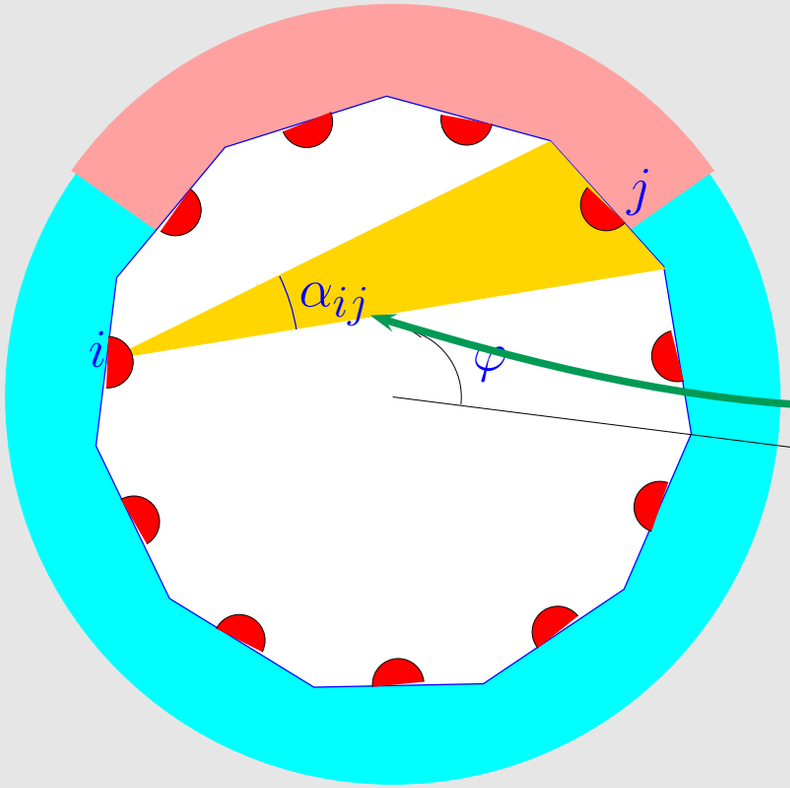
Example 8.1.13 (Radiative heat transfer).

Beyond signal processing discrete periodic convolutions occur in mathematical models:

An engineering problem:

- cylindrical pipe,
- heated on part Γ_H of its perimeter ($\rightarrow$ prescribed heat flux),
- cooled on remaining perimeter Γ_K ($\rightarrow$ constant heat flux).





Modeling (discretization):

- approximation by regular n-polygon, edges Γ_j,
- isotropic radiation of each edge Γ_j (power I_j),

radiative heat flow $\Gamma_j \to \Gamma_i$: $P_{ji} := \frac{\alpha_{ij}}{\pi} I_j$,

opening angle: $\alpha_{ij} = \pi\, \gamma_{|i-j|}$, $1 \le i,j \le n$,

power balance: $\underbrace{\sum_{i=1,i\neq j}^{n} P_{ji}}_{=I_j} - \sum_{i=1,i\neq j}^{n} P_{ij} = Q_j$. (8.1.14)

$Q_j \mathrel{\hat{=}}$ heat flux through Γ_j, satisfies

$$Q_j := \int_{\frac{2\pi}{n}(j-1)}^{\frac{2\pi}{n}j} q(\varphi)\,\mathrm{d}\varphi\,, \quad q(\varphi) := \begin{cases} \text{local heating} & \text{, if } \varphi \in \Gamma_H\,, \\ -\frac{1}{|\Gamma_K|}\int_{\Gamma_H} q(\varphi)\,\mathrm{d}\varphi & \text{(const.), if } \varphi \in \Gamma_K\,. \end{cases}$$

(8.1.14) $\Rightarrow$ LSE: $I_j - \sum_{i=1,i\neq j}^{n} \frac{\alpha_{ij}}{\pi} I_i = Q_j\,, \quad j = 1,\ldots,n\,.$

R. Hiptmair

rev 30401, January 28, 2011

$$n = 8: \quad \begin{pmatrix} 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 \\ -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 \\ -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 \\ -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_2 & -\gamma_4 \\ -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 \\ -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 \\ -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 \\ -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \\ I_8 \end{pmatrix} = \begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \\ Q_7 \\ Q_8 \end{pmatrix} . \tag{8.1.15}$$

This is a linear system of equations with symmetric, singular, and (by Thm. 6.1.5, $\sum \gamma_i \leq 1$) positive semidefinite ($\rightarrow$ Def. 2.7.9) system matrix.

Note: matrices from (8.1.11) and (8.1.15) have the same structure !

Observe: LSE from (8.1.15) can be written by means of the discrete periodic convolution ($\rightarrow$ Def. 8.1.12) of vectors $\mathbf{y} = (1, -\gamma_1, -\gamma_2, -\gamma_3, -\gamma_4, -\gamma_3, -\gamma_2, -\gamma_1)$, $\mathbf{x} = (I_1, \ldots, I_8)$

$$\text{(8.1.15)} \quad \leftrightarrow \quad \mathbf{y} *_8 \mathbf{x} = (Q_1, \ldots, Q_8)^T .$$

Definition 8.1.16 (Circulant matrix)**.**

A matrix $\mathbf{C} = (c_{ij})_{i,j=1}^{n} \in \mathbb{K}^{n,n}$ *is* circulant *(*ger.: *zirkulant)*

$$:\Leftrightarrow \quad \exists\, (u_k)_{k\in\mathbb{Z}} \ n\text{-periodic sequence: } \ c_{ij} = u_{j-i},\ 1 \le i,j \le n.$$

☞ Circulant matrix has constant (main, sub- and super-) diagonals (for which indices $j - i = \text{const.}$).

☞ columns/rows arise by *cyclic permutation* from first column/row.

Similar to the case of banded matrices ($\rightarrow$ Sect. 2.6.4):

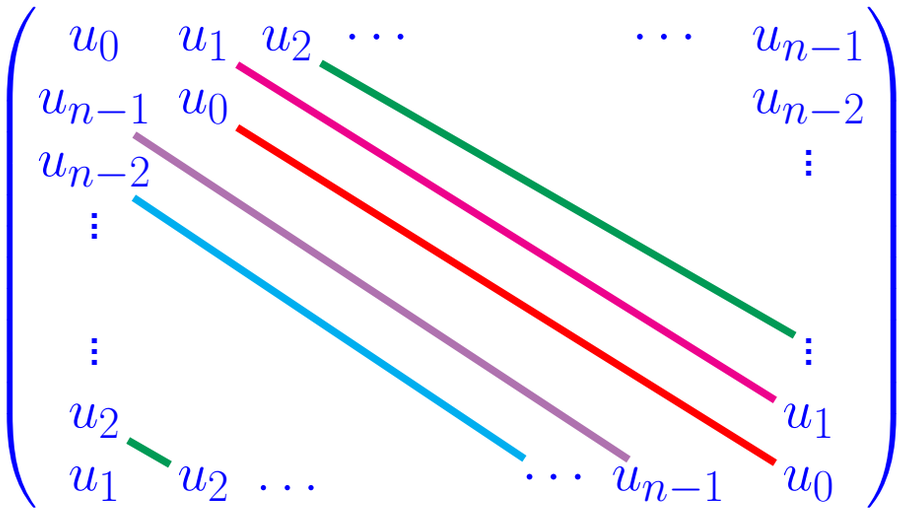
"information content" of circulant matrix $\mathbf{C} \in \mathbb{K}^{n,n}$ = n numbers $\in \mathbb{K}$.

(obviously, one vector $\mathbf{u} \in \mathbb{K}^n$ enough to define circulant matrix $\mathbf{C} \in \mathbb{K}^{n,n}$)

R. Hiptmair

rev 30401, January 28, 2011

Structure of circulant matrix ▷



Remark 8.1.17 (Reduction to periodic convolution).

Recall discrete convolution (→ Def. 8.1.6) of $\mathbf{a} = (a_0, \ldots, a_{n-1})^T \in \mathbb{K}^n$, $\mathbf{b} = (b_0, \ldots, b_{n-1})^T \in \mathbb{K}^n$:

$$(\mathbf{a} * \mathbf{b})_k = \sum_{j=0}^{n-1} a_j b_{k-j} \;, \quad k = 0, \ldots, 2n-2 \;.$$

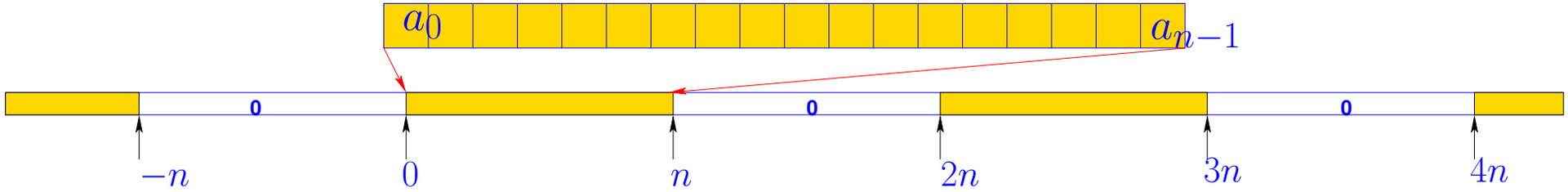
Expand $a_0, \ldots, a_{n-1}$ and $b_0, \ldots, b_{n-1}$ to $2n-1$-periodic sequences by zero padding:

$$x_k := \begin{cases} a_k & \text{, if } 0 \le k < n \;, \\ 0 & \text{, if } n \le k < 2n-1 \end{cases} \;, \quad y_k := \begin{cases} b_k & \text{, if } 0 \le k < n \;, \\ 0 & \text{, if } n \le k < 2n-1 \;, \end{cases} \tag{8.1.18}$$

and periodic extension: $x_k = x_{2n-1+k}$, $y_k = y_{2n-1+k}$ for all $k \in \mathbb{Z}$.

R. Hiptmair

rev 30401, January 28, 2011



$$(\mathbf{a} * \mathbf{b})_k = (\mathbf{x} *_{2n-1} \mathbf{y})_k \ , \quad k = 0, \ldots, 2n-2 \ . \tag{8.1.19}$$

R. Hiptmair

rev 30401, January 28, 2011

Matrix view of reduction to periodic convolution, *cf.* (8.1.3)

$$\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \\ \\ \\ \vdots \\ y_{2n-2} \end{pmatrix} = \underbrace{\begin{pmatrix} h_0 & 0 & \cdots & 0 & h_{n-1} & \cdots & h_1 & h_0 \\ h_1 & & & & 0 & & & \\ \vdots & & & & & & & \\ h_{n-1} & & & & h_0 & 0 & & 0 \\ 0 & & & & h_1 & h_0 & & h_{n-1} \\ & & & & & & & 0 \\ & & & & & & & \vdots \\ 0 & & \cdots & 0 & h_{n-1} & & h_1 & h_0 \end{pmatrix}}_{\text{a } (2n-1)\times(2n-1) \text{ circulant matrix!}} \begin{pmatrix} x_0 \\ \vdots \\ \\ \\ \vdots \\ x_{n-1} \\ 0 \\ \vdots \\ \\ \\ \vdots \\ 0 \end{pmatrix} .$$

△

R. Hiptmair

rev 30401, January 28, 2011

8.2 Discrete Fourier Transform (DFT)

Example 8.2.1 (Eigenvectors of circulant matrices).

Code 8.2.2: Eigenvectors of random circulant matrices

```
function circeig

n = 8;
C = gallery('circul',rand(n,1)); [V1,D1] = eig(C);

for j=1:n
  figure; bar(1:n,[real(V1(:,j)),imag(V1(:,j))],1,'grouped');
  title(sprintf('Circulant matrix 1, eigenvector %d',j));
  xlabel('{\bf vector component index}','fontsize',14);
  ylabel('{\bf vector component value}','fontsize',14);
  legend('real part','imaginary part','location','southwest');
  print('-depsc2',sprintf('../PICTURES/circeig1ev%d.eps',j));
end

C = gallery('circul',rand(n,1)); [V2,D2] = eig(C);
```

```
16
17 for j=1:n
18   figure; bar(1:n,[real(V2(:,j)),imag(V2(:,j))],1,'grouped');
19   title(sprintf('Circulant matrix 2, eigenvector %d',j));
20   xlabel('{\bf vector component index}','fontsize',14);
21   ylabel('{\bf vector component value}','fontsize',14);
22   legend('real part','imaginary part','location','southwest');
23   print('-depsc2',sprintf('../PICTURES/circeig2ev%d.eps',j));
24 end
25
26 figure; plot(1:n,real(diag(D1)),'r+',1:n,imag(diag(D1)),'b+',...
27           1:n,real(diag(D2)),'m*',1:n,imag(diag(D2)),'k*');
28 ax = axis; axis([0 n+1 ax(3) ax(4)]);
29 xlabel('{\bf index of eigenvalue}','fontsize',14);
30 ylabel('{\bf eigenvalue}','fontsize',14);
31 legend('C_1: real(ev)', 'C_1: imag(ev)', 'C_2: real(ev)', 'C_2:
   imag(ev)', 'location','northeast');
32
33 print -depsc2 '../PICTURES/circeigev.eps';
```


R. Hiptmair

rev 30401, January 28, 2011

Random 8×8 circulant matrices $\mathbf{C}_1$, $\mathbf{C}_2$ ($\rightarrow$ Def. 8.1.16)

eigenvalues $\triangleright$

Generated by MATLAB-command:

```
C = gallery('circul',rand(n,1));
```

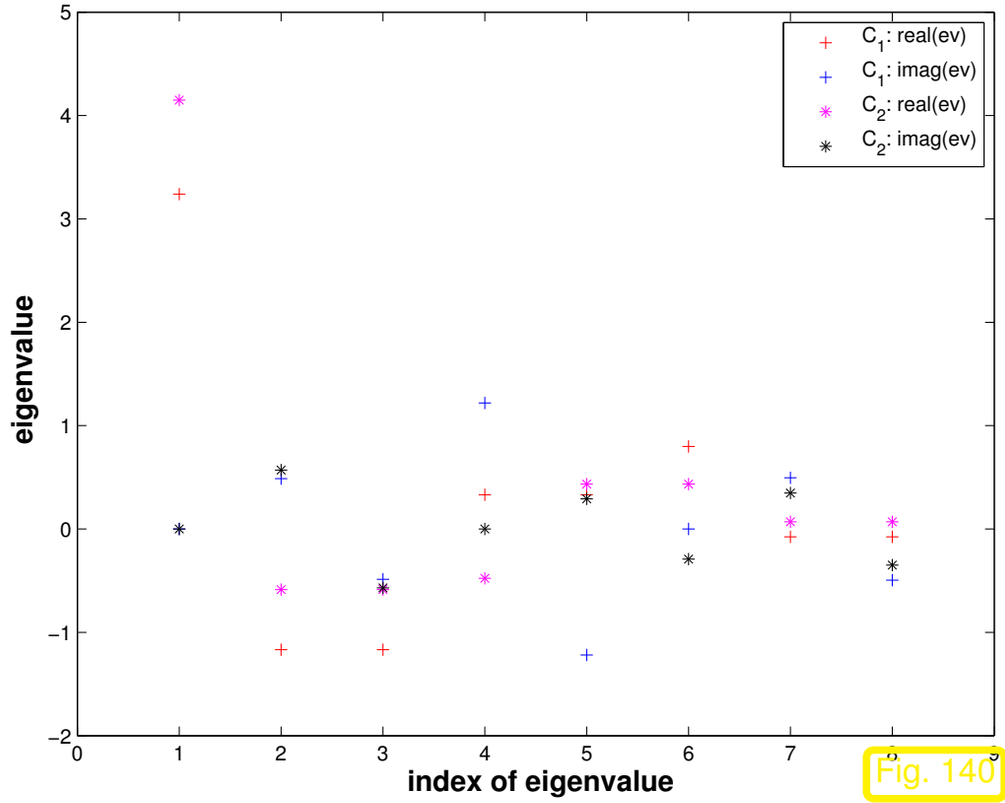


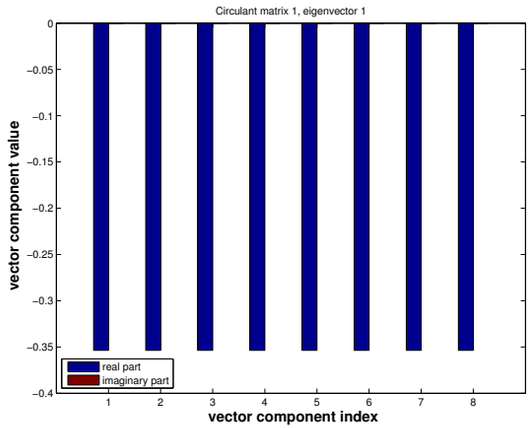
Fig. 140

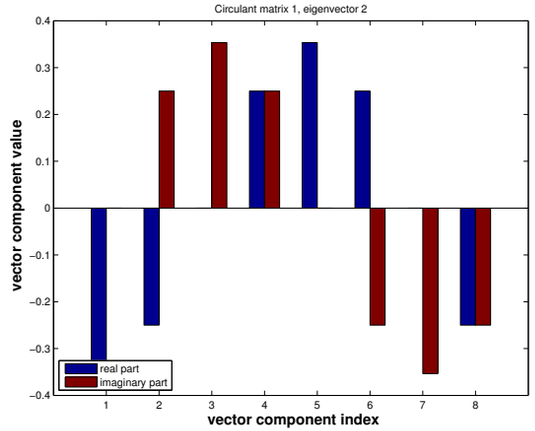
Little relationship between (complex!) eigenvalues can be observed, as can be expected from random matrices with entries $\in [0, 1]$.

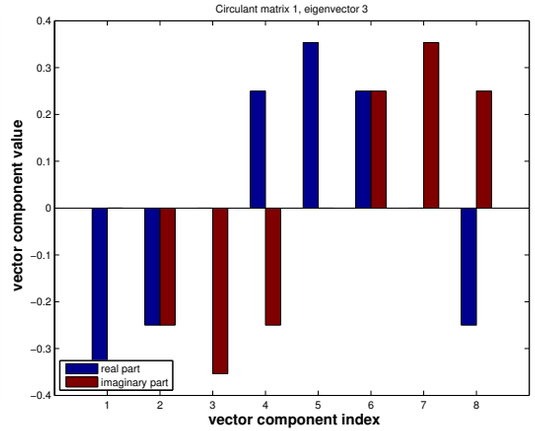
Eigenvectors of matrix $\mathbf{C}_1$:

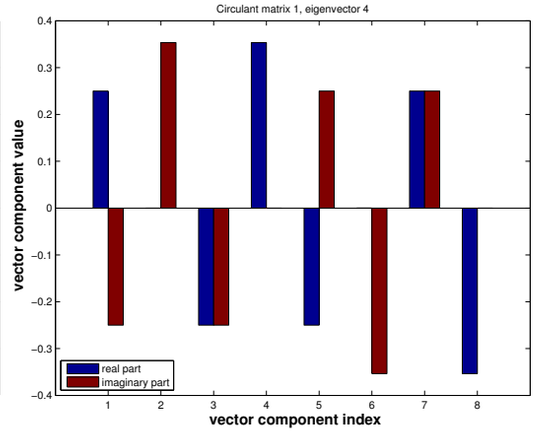
R. Hiptmair

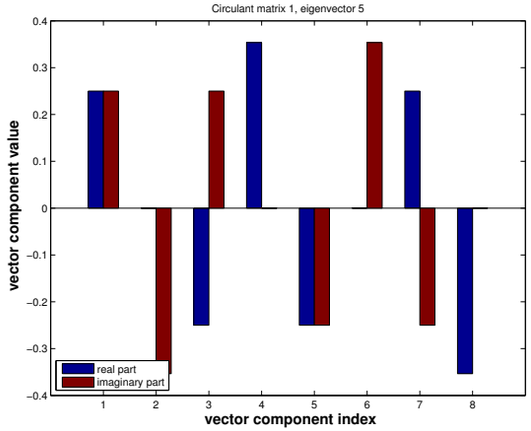
rev 30401, January 28, 2011

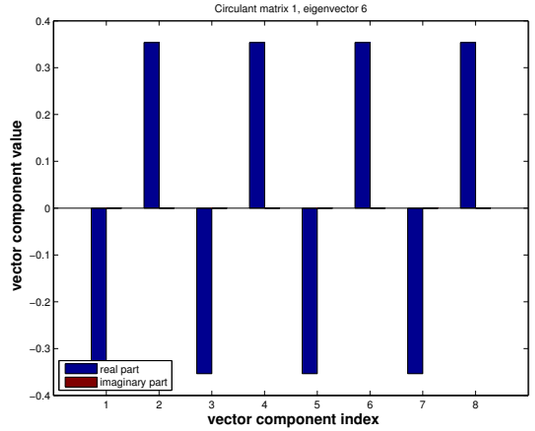


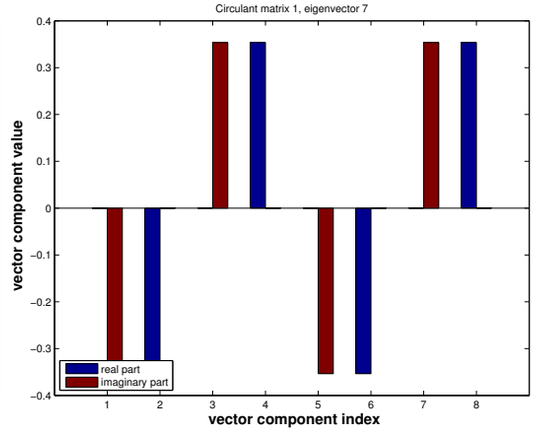


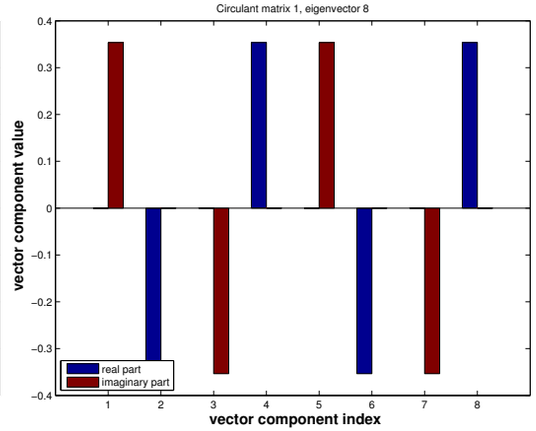








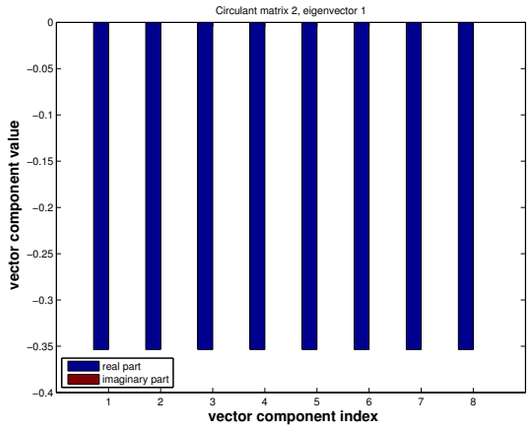


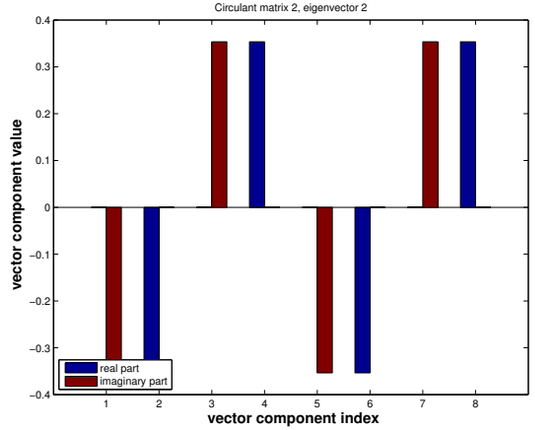


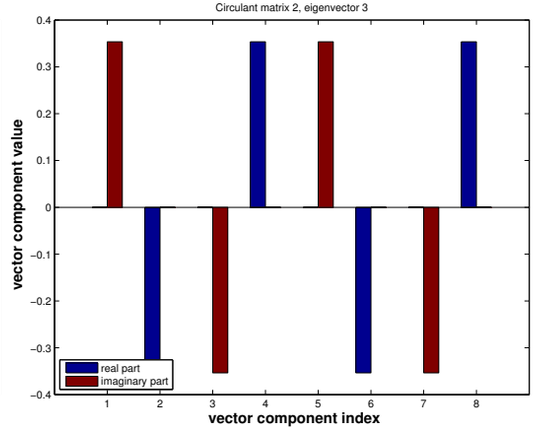
R. Hiptmair

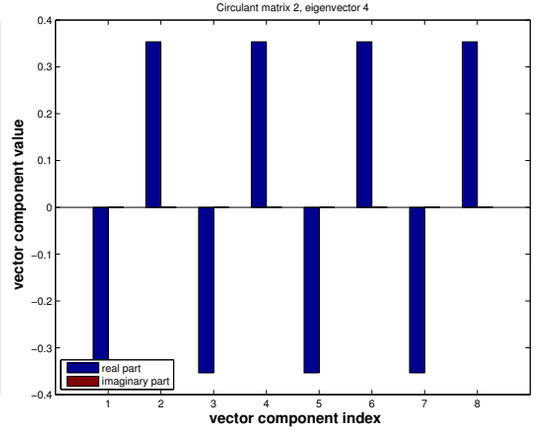
rev 30401, January 28, 2011

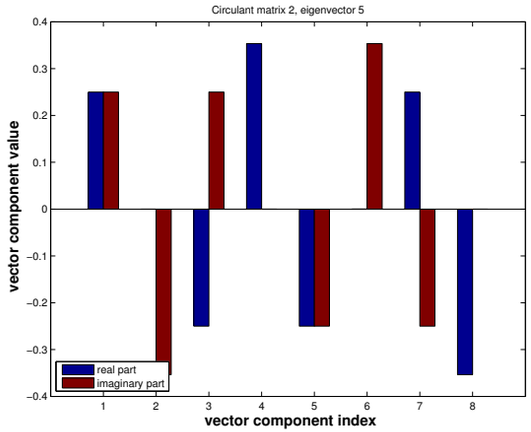
Eigenvectors of matrix $\mathbf{C}_2$

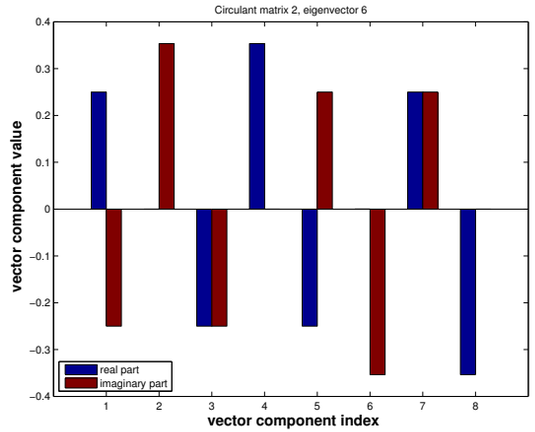


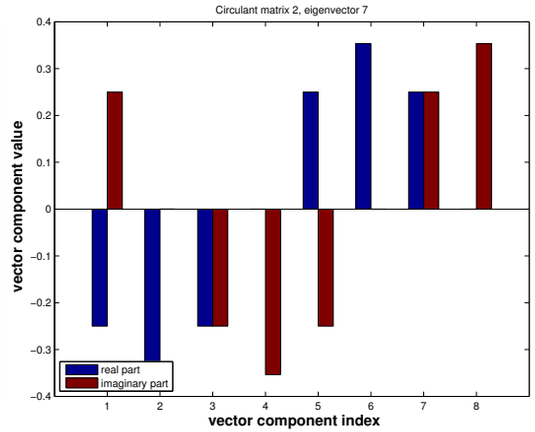


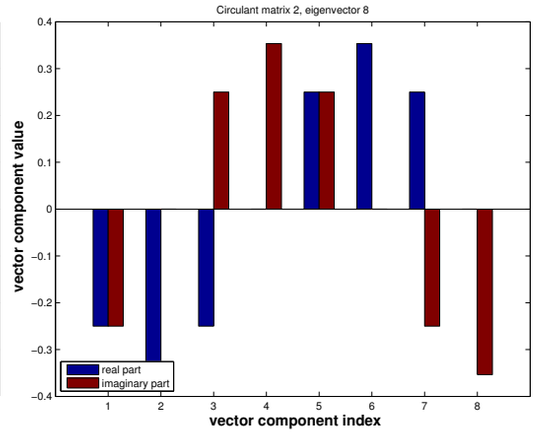






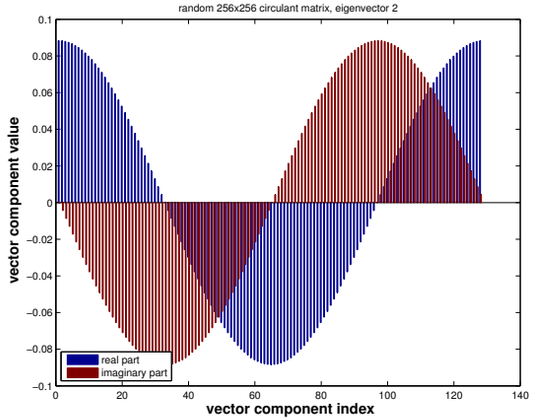


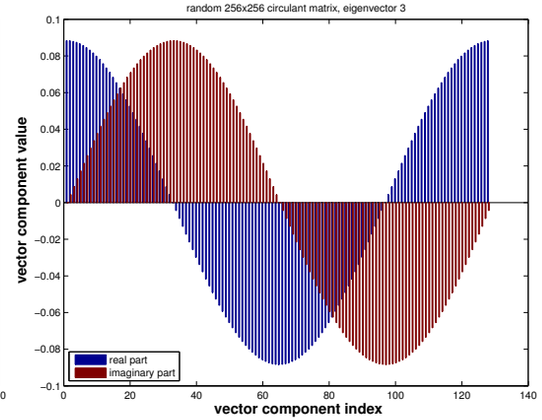


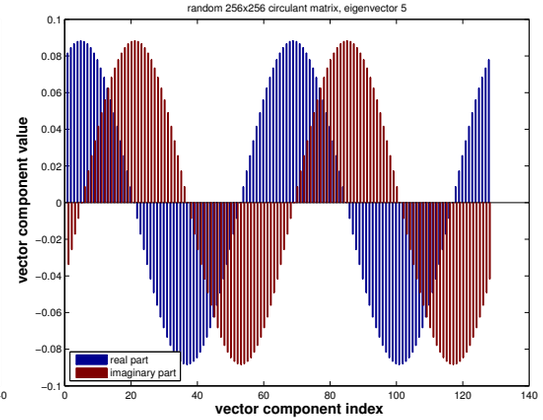


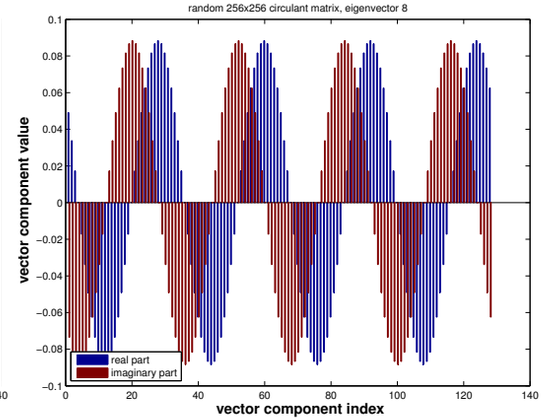
Observation: the different random circulant matrices have the same eigenvectors!

Eigenvectors of C = gallery('circul',(1:128)'); :









R. Hiptmair

rev 30401, January 28, 2011

The eigenvectors remind us of sampled *trigonometric functions* $\cos(k/n)$, $\sin(k/n)$, $k = 0, \ldots, n-1$!

◇

Remark 8.2.3 (Why using $\mathbb{K} = \mathbb{C}$?).

Ex. 8.2.1: *complex* eigenvalues/eigenvectors for general circulant matrices.

Recall from analysis: unified treatment of trigonometric functions via *complex exponential function*

$$\exp(it) = \cos(t) + i\sin(t) \;, \quad t \in \mathbb{R} \;.$$

$\mathbb{C}$! The field of complex numbers $\mathbb{C}$ is the *natural framework* for the analysis of linear, time-invariant filters, and the development of algorithms for circulant matrices.

△

✎ notation: nth root of unity $\omega_n := \exp(-2\pi i/n) = \cos(2\pi/n) - i\sin(2\pi/n)$, $n \in \mathbb{N}$

satisfies $\overline{\omega}_n = \omega_n^{-1}$, $\boxed{\omega_n^n = 1}$, $\omega_n^{n/2} = -1$, $\omega_n^k = \omega_n^{k+n}$ $\forall\, k \in \mathbb{Z}$, (8.2.4)

$$\sum_{k=0}^{n-1} \omega_n^{kj} = \begin{cases} n & \text{, if } j = 0 \mod n \text{ ,} \\ 0 & \text{, if } j \neq 0 \mod n \text{ .} \end{cases} \tag{8.2.5}$$

(8.2.5) is a simple consequence of the geometric sum formula

$$\sum_{k=0}^{n-1} q^k = \frac{1-q^n}{1-q} \quad \forall\, q \in \mathbb{C} \setminus \{1\} \text{ , } \quad n \in \mathbb{N} \text{ .} \tag{8.2.6}$$

$$\Rightarrow \quad \sum_{k=0}^{n-1} \omega_n^{kj} = \frac{1-\omega_n^{nj}}{1-\omega_n^j} = \frac{1-\exp(-2\pi ij)}{1-\exp(-2\pi ij/n)} = 0 \text{ ,}$$

because $\exp(-2\pi ij) = \omega_n^{nj} = (\omega_n^n)^j = 1$ for all $j \in \mathbb{Z}$.

Now we want to confirm the conjecture gleaned from Ex. 8.2.1 that vectors with powers of roots of unity are eigenvectors for any circulant matrix. We do this by simple and straightforward computations:

Consider: $\mathbf{C} \in \mathbb{C}^{n,n}$ circulant matrix ($\rightarrow$ Def. 8.1.16), $c_{ij} = u_{i-j}$, for n-periodic sequence $(u_k)_{k\in\mathbb{Z}}$, $u_k \in \mathbb{C}$

$\mathbf{v}_k \in \mathbb{C}^n$ with $\mathbf{v}_k := (\omega_n^{jk})_{j=0}^{n-1} \in \mathbb{C}^n$, $k \in \{0,\ldots,n-1\}$.

$(u_{j-l}\omega_n^{lk})_{l\in\mathbb{Z}}$ is n-periodic!

$$(\mathbf{C}\mathbf{v}_k)_j = \sum_{l=0}^{n-1} u_{j-l}\omega_n^{lk} = \sum_{l=j-n+1}^{j} u_{j-l}\omega_n^{lk} \tag{8.2.7}$$

$$= \sum_{l=0}^{n-1} u_l \omega_n^{(j-l)k} = \omega_n^{jk} \boxed{\sum_{l=0}^{n-1} u_l \omega_n^{-lk}} = \lambda_k \cdot \omega_n^{jk} = \lambda_k \cdot (\mathbf{v}_k)_j \ .$$

change of summation index

independent of j !

$\mathbf{v}$ is eigenvector of $\mathbf{C}$ to eigenvalue $\lambda_k = \sum_{l=0}^{n-1} u_l \omega_n^{-lk}$.

R. Hiptmair

rev 30401, January 28, 2011

Orthogonal trigonometric basis of $\mathbb{C}^n$ = *eigenvector basis* for circulant matrices

$$\left\{ \begin{pmatrix} \omega_n^0 \\ \vdots \\ \\ \\ \vdots \\ \omega_n^0 \end{pmatrix} \begin{pmatrix} \omega_n^0 \\ \omega_n^1 \\ \vdots \\ \\ \vdots \\ \omega_n^{n-1} \end{pmatrix} \cdots \begin{pmatrix} \omega_n^0 \\ \omega_n^{n-2} \\ \omega_n^{2(n-2)} \\ \vdots \\ \vdots \\ \omega_n^{(n-1)(n-2)} \end{pmatrix} \begin{pmatrix} \omega_n^0 \\ \omega_n^{n-1} \\ \omega_n^{2(n-1)} \\ \vdots \\ \vdots \\ \omega_n^{(n-1)^2} \end{pmatrix} \right\} .$$

(8.2.5) $\Rightarrow$ orthogonality of basis vectors:

$$\mathbf{v}_k := (\omega_n^{jk})_{j=0}^{n-1} \in \mathbb{C}^n: \quad \mathbf{v}_k^H \mathbf{v}_m = \sum_{j=0}^{n-1} \omega_n^{-jk} \omega_n^{jm} = \sum_{j=0}^{n-1} \omega_n^{(m-k)j} \overset{(8.2.5)}{=} 0 \quad \text{, if } k \neq m \text{ .} \qquad (8.2.8)$$

Matrix of change of basis trigonometrical basis $\rightarrow$ standard basis: Fourier-matrix

$$\mathbf{F}_n = \begin{pmatrix} \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \cdots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \cdots & \omega_n^{2n-2} \\ \vdots & \vdots & & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{pmatrix} = \left(\omega_n^{lj}\right)_{l,j=0}^{n-1} \in \mathbb{C}^{n,n} . \qquad (8.2.9)$$

R. Hiptmair

rev 30401, January 28, 2011

Lemma 8.2.10 (Properties of Fourier matrix)**.**

The scaled Fourier-matrix $\frac{1}{\sqrt{n}}\mathbf{F}_n$ *is unitary (→ Def. 2.8.5):* $\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^H = \frac{1}{n}\overline{\mathbf{F}}_n$.

Proof. The lemma is immediate from (8.2.8) and (8.2.5),because

$$\left(\mathbf{F}_n\mathbf{F}_n^H\right)_{l,j} = \sum_{k=0}^{n-1} \omega_n^{(l-1)k}\overline{\omega_n}^{(j-1)k} = \sum_{k=0}^{n-1} \omega_n^{(l-1)k}\omega_n^{-(j-1)k} = \sum_{k=0}^{n-1} \omega_n^{k(l-j)} \;, \quad 1 \le l,j \le n \;.$$

Remark 8.2.11 (Spectrum of Fourier matrix)*.*

$$\frac{1}{n^2}\mathbf{F}_n^4 = I \quad \Rightarrow \quad \sigma(\frac{1}{\sqrt{n}}\mathbf{F}_n) \subset \{1,-1,i,-i\} \;,$$

because, if $\lambda \in \mathbb{C}$ is an eigenvalue of $\mathbf{F}_n$, then there is an eigenvector $\mathbf{x} \in \mathbb{C}^n \setminus \{0\}$ such that $\mathbf{F}_n\mathbf{x} = \lambda\mathbf{x}$, see Def. 6.1.1.

△

R. Hiptmair

Lemma 8.2.12 (Diagonalization of circulant matrices ($\rightarrow$ Def. 8.1.16))**.**
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

$$\mathbf{C}\overline{\mathbf{F}}_n = \overline{\mathbf{F}}_n \operatorname{diag}(d_1, \ldots, d_n) \quad , \quad \mathbf{d} = \mathbf{F}_n(u_0, \ldots, u_{n-1})^T \, .$$

Proof. Straightforward computation, see (8.2.7). □

Conclusion (from $\overline{\mathbf{F}}_n = n\mathbf{F}_n^{-1}$):

$$\boxed{\mathbf{C} = \mathbf{F}_n^{-1} \operatorname{diag}(d_1, \ldots, d_n)\mathbf{F}_n} \, . \tag{8.2.13}$$

Lemma 8.2.12, (8.2.13) ➢ multiplication with Fourier-matrix will be crucial operation in algorithms for circulant matrices and discrete convolutions.

Therefore this operation has been given a special name:

Definition 8.2.14 (Discrete Fourier transform (DFT)).

The linear map $\mathcal{F}_n : \mathbb{C}^n \mapsto \mathbb{C}^n$, $\mathcal{F}_n(\mathbf{y}) := \mathbf{F}_n\mathbf{y}$, $\mathbf{y} \in \mathbb{C}^n$, *is called discrete Fourier transform (DFT), i.e. for* $\mathbf{c} := \mathcal{F}_n(\mathbf{y})$

$$c_k = \sum_{j=0}^{n-1} y_j \, \omega_n^{kj} \quad , \quad k = 0, \ldots, n-1 \, . \tag{8.2.15}$$

Recall the convention also relevant for the discussion of the DFT: vector indexes range from 0 to $n-1$!

Terminology: $\mathbf{c} = \mathbf{F}_n\mathbf{y}$ is also called the (discrete) Fourier transform of $\mathbf{y}$

MATLAB-functions for discrete Fourier transform (and its inverse):

DFT: `c=fft(y)` $\leftrightarrow$ inverse DFT: `y=ifft(c);`

R. Hiptmair

rev 30401, January 28, 2011

8.2.1 Discrete convolution via DFT

Recall discrete periodic convolution $z_k = \sum_{j=0}^{n-1} u_{k-j} x_j$ ($\rightarrow$ Def. 8.1.12), $k = 0, \ldots, n-1$

$$\updownarrow$$

multiplication with circulant matrix ($\rightarrow$ Def. 8.1.16) $\mathbf{z} = \mathbf{C}\mathbf{x}$, $\mathbf{C} := \left(u_{i-j}\right)_{i,j=1}^{n}$.

Idea: (8.2.13) $\succ$ $\mathbf{z} = \mathbf{F}_n^{-1} \operatorname{diag}(\mathbf{F}_n \mathbf{u}) \mathbf{F}_n \mathbf{x}$

Code 8.2.17: discrete periodic convolution: straightforward implementation

```
1 function z=pconv(u,x)
2 n = length(x); z = zeros(n,1);
3 for i=1:n, z(i)=dot(conj(u),
    x([i:-1:1,n:-1:i+1]));
4 end
```

Code 8.2.19: discrete periodic convolution: DFT implementation

```
1 function z=pconvfft(u,x)
2 z = ifft(fft(u).*fft(x));
```

R. Hiptmair

rev 30401, January 28, 2011

Rem. 8.1.17: discrete convolution of n-vectors ($\rightarrow$ Def. 8.1.6) by *periodic* discrete convolution of $2n-1$-vectors (obtained by zero padding):

Implementation of discrete convolution ($\rightarrow$ Def. 8.1.6) based on periodic discrete convolution $\triangleright$

Built-in MATLAB-function:

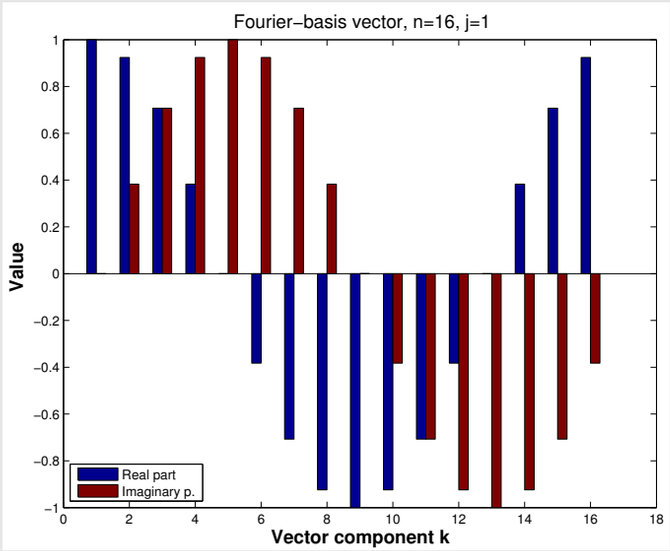
```
y = conv(h,x);
```

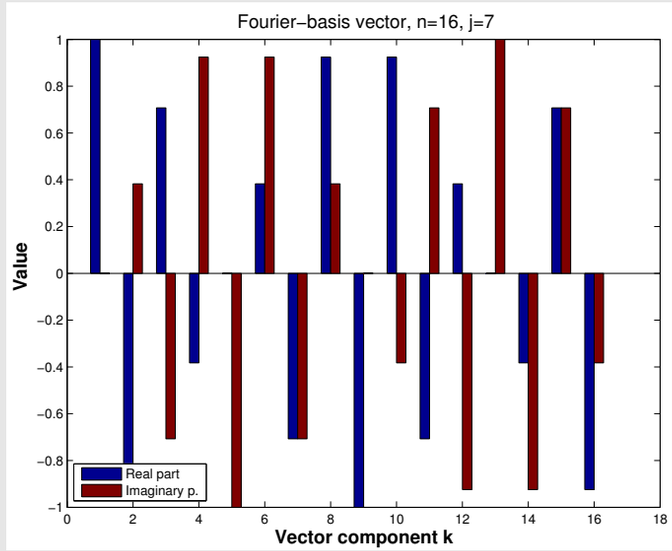
Code 8.2.20: discrete convolutiuon: DFT implementation

```
1 function y = myconv(h,x)
2 n = length(h);
3 % Zero padding
4 h = [h;zeros(n-1,1)]; x =
    [x;zeros(n-1,1)];
5 % Periodic discrete convolution of length 2n-1
6 y = pconvfft(h,x);
```

8.2.2 Frequency filtering via DFT

The trigonometric basis vectors, when interpreted as time-periodic signals, represent harmonic oscillations. This is illustrated when plotting some vectors of the trigonometric basis ($n = 16$):





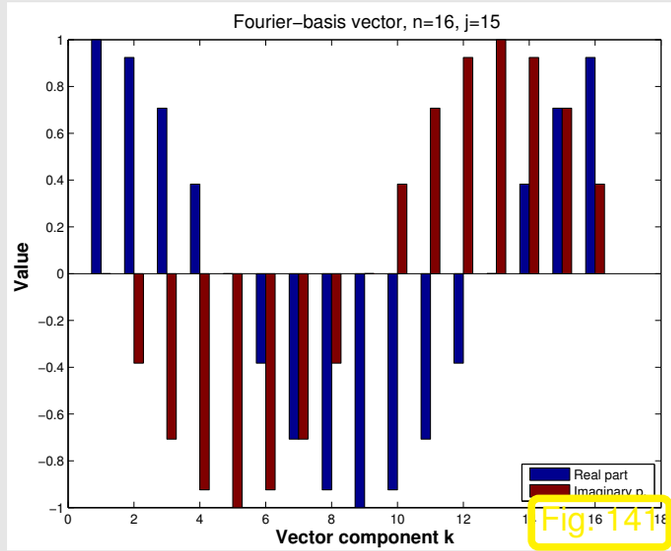


Fig. 141

"slow oscillation/low frequency" "fast oscillation/high frequency" "slow oscillation/low frequency"

R. Hiptmair

rev 30401, January 28, 2011

Dominant coefficients of a signal after transformation to trigonometric basis indicate dominant frequency components.

Terminology: coefficients of a signal w.r.t. trigonometric basis = signal in frequency domain (*ger.*: Frequenzbereich), original signal = time domain (*ger.*: Zeitbereich).

Example 8.2.21 (Frequency identification with DFT).

Extraction of characteristical frequencies from a distorted discrete periodical signal:

```
t = 0:63; x = sin(2*pi*t/64)+sin(7*2*pi*t/64);
y = x + randn(size(t)); %distortion
```

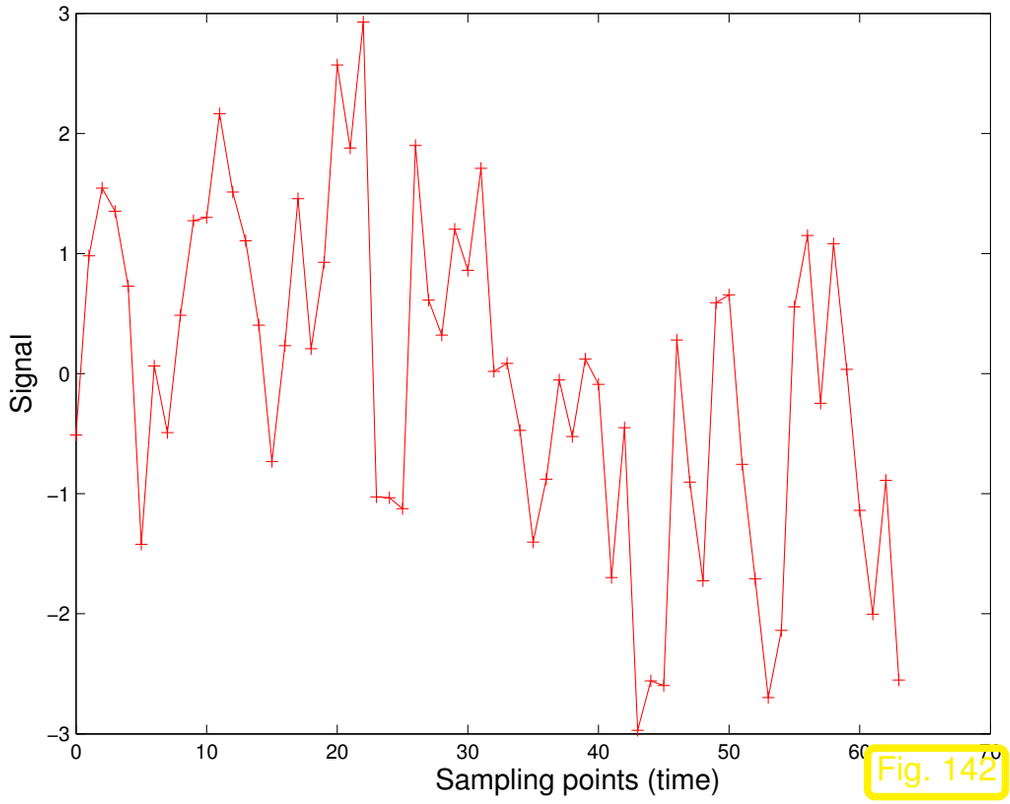


Fig. 142

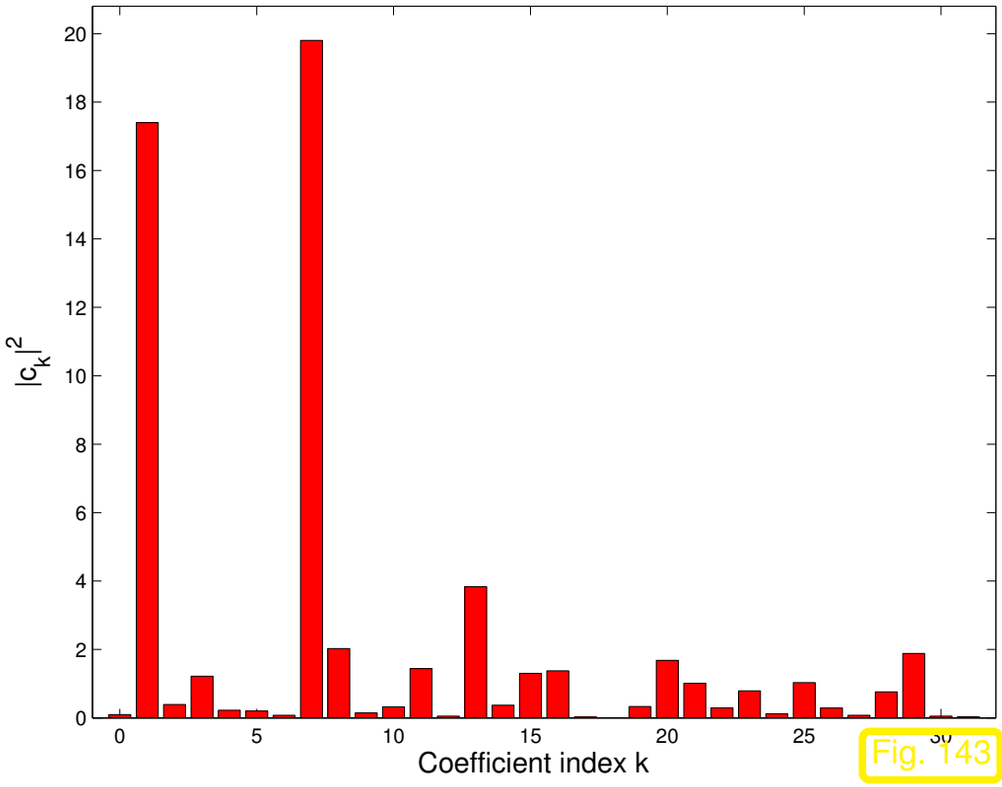


Fig. 143

Generating Fig. 143:

```
c = fft(y); p = (c.*conj(c))/64;

figure('Name','power spectrum');
```

R. Hiptmair

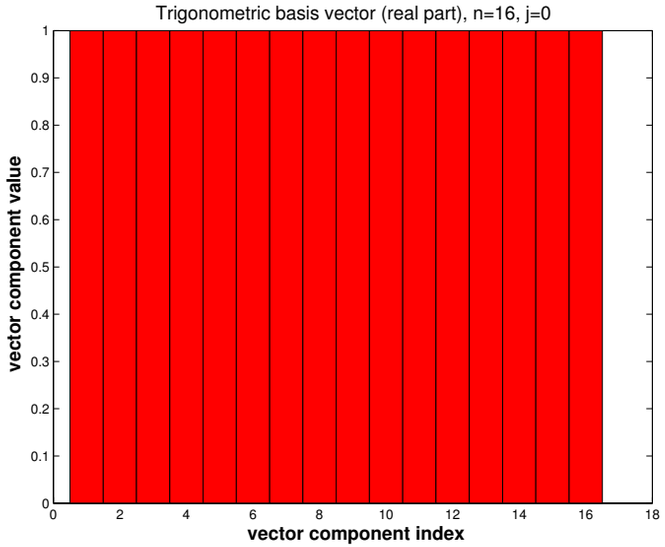
rev 30401, January 28, 2011

```
bar(0:31,p(1:32),'r');
set(gca,'fontsize',14);
axis([-1 32 0 max(p)+1]);
xlabel('{\bf index k of Fourier coefficient}','Fontsize',14);
ylabel('{\bf |c_k|^2}','Fontsize',14);
```

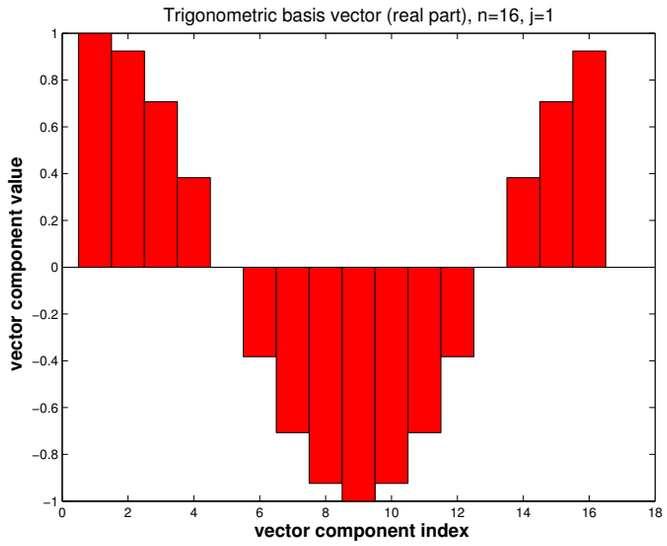

◇

Remark 8.2.22 (“Low” and “high” frequencies).

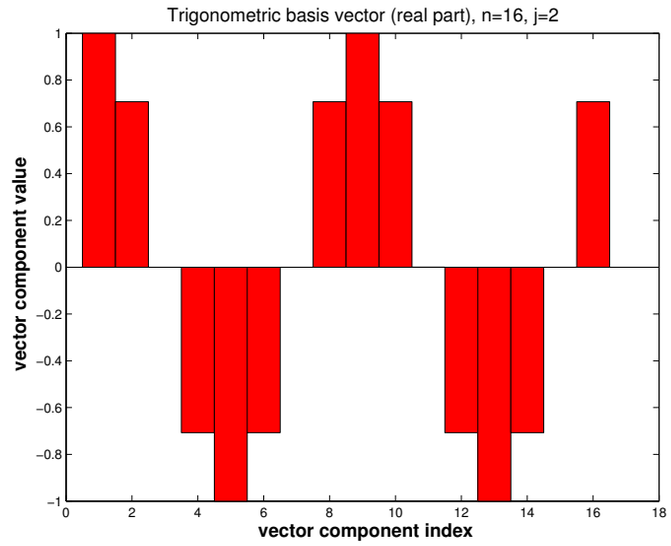
Plots of real parts of trigonometric basis vectors $(\mathbf{F}_n)_{:,j}$ (= columns of Fourier matrix), $n = 16$.



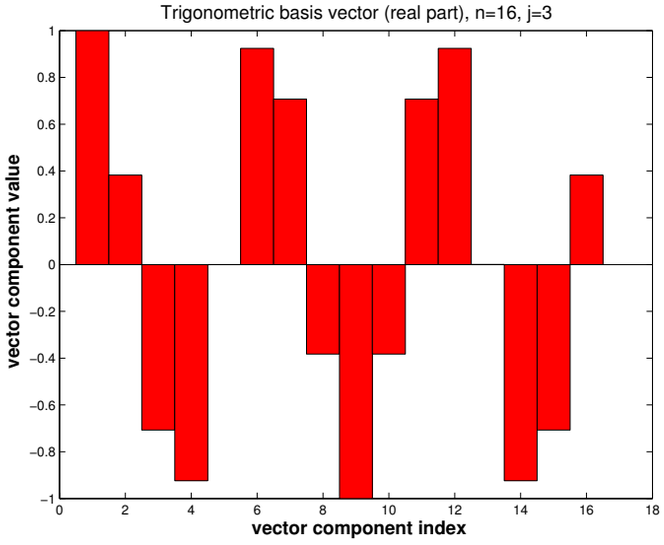
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,0}$



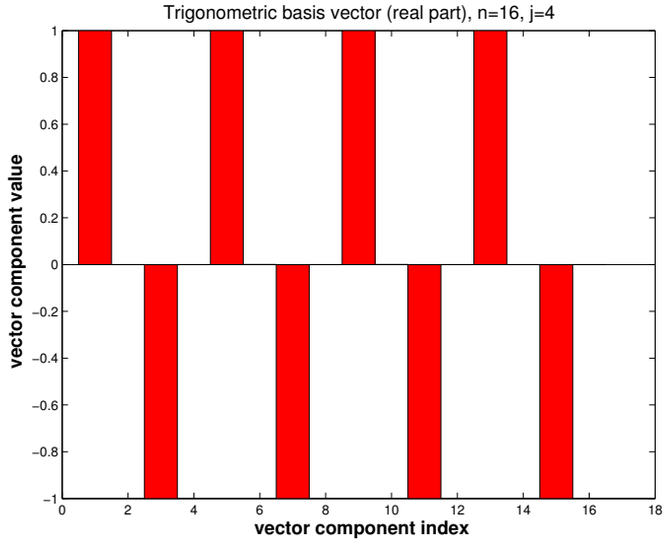
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,1}$



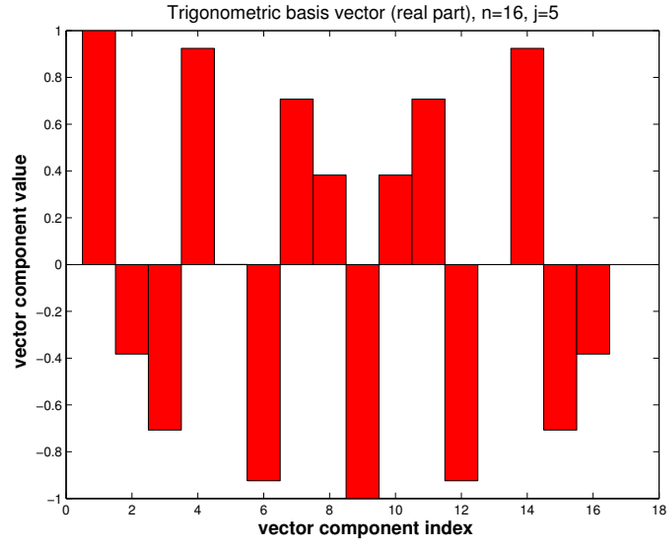
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,2}$



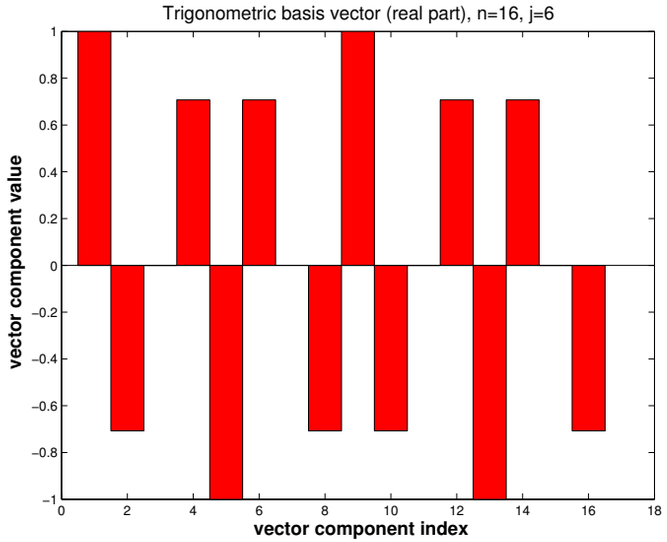
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,3}$



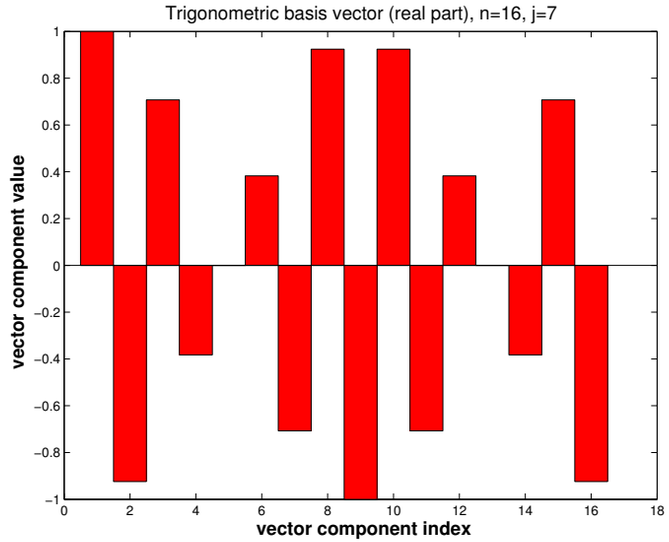
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,4}$



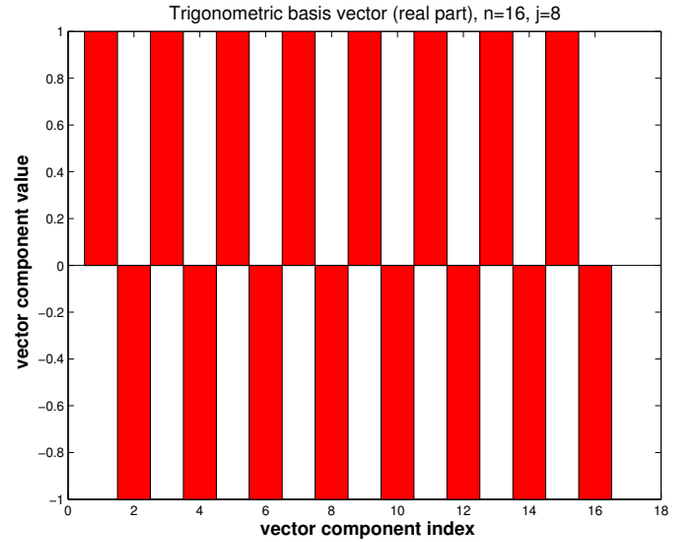
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,5}$



$\mathrm{Re}\,(\mathbf{F}_{16})_{:,6}$



$\mathrm{Re}\,(\mathbf{F}_{16})_{:,7}$



$\mathrm{Re}\,(\mathbf{F}_{16})_{:,8}$

△

By elementary trigonometric identities:

$$\mathrm{Re}\,(\mathbf{F}_n)_{:,j} = \left(\mathrm{Re}\,\omega_n^{(j-1)k}\right)_{k=0}^{n-1} = (\mathrm{Re}\exp(2\pi i(j-1)k/n))_{k=0}^{n-1} = (\cos(2\pi(j-1)x))_{x=0,\frac{1}{n},\ldots,1-\frac{1}{n}} \,.$$

Slow oscillations/low frequencies $\leftrightarrow$ $j \approx 1$ and $j \approx n$.

Fast oscillations/high frequencies $\leftrightarrow$ $j \approx n/2$.

Frequency filtering of real discrete periodic signals by suppressing certain "Fourier coefficients".

Code 8.2.23: DFT-based frequency filtering

```
1 function [low,high] =
    freqfilter(y,k)
2 m = length(y)/2; c = fft(y);
3 clow = c; clow(m+1-k:m+1+k) = 0;
4 chigh = c-clow;
5 low = real(ifft(clow));
6 high = real(ifft(chigh));
```

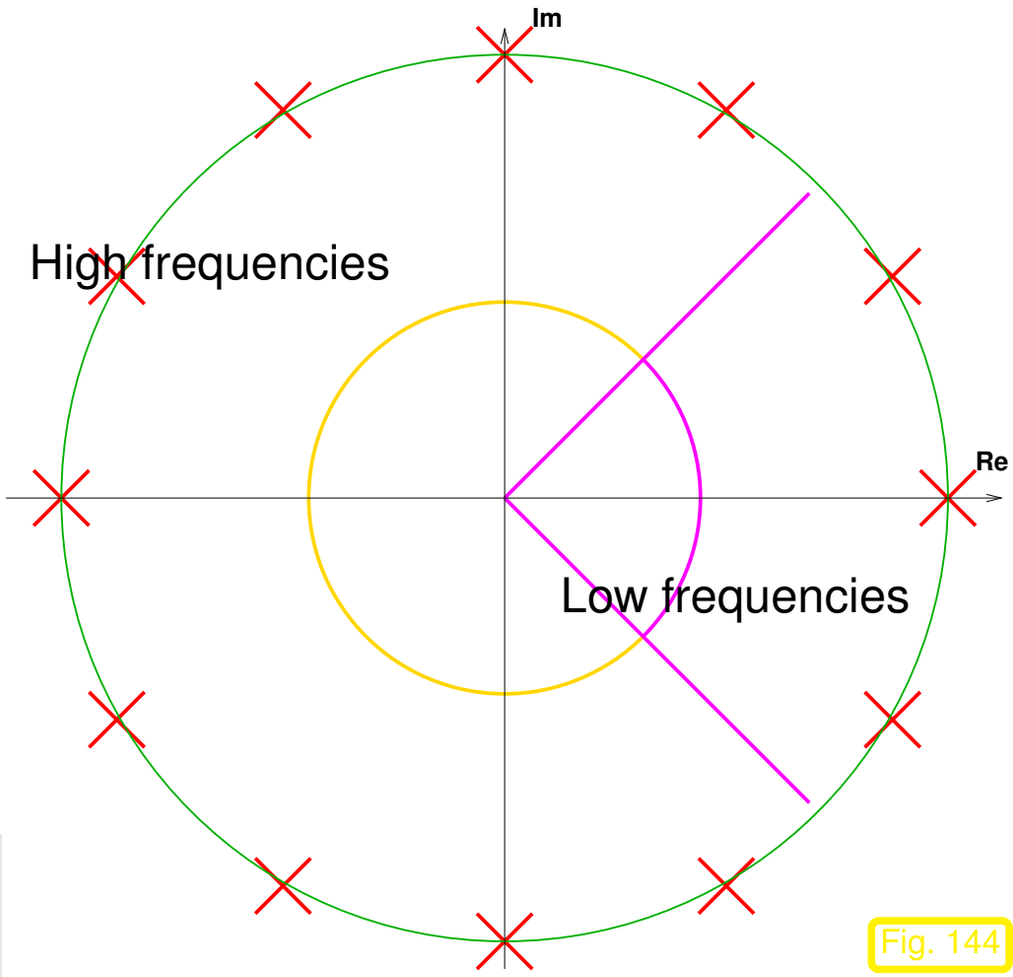


Fig. 144

(can be optimised exploiting $y_j \in \mathbb{R}$ and $c_{n/2-k} = \overline{c}_{n/2+k}$)

Map `y` $\mapsto$ `low` (in Code 8.2.22) $\hat{=}$ low pass filter (*ger.:* Tiefpass).

Map `y` $\mapsto$ `high` (in Code 8.2.22) $\hat{=}$ high pass filter (*ger.:* Hochpass).

R. Hiptmair

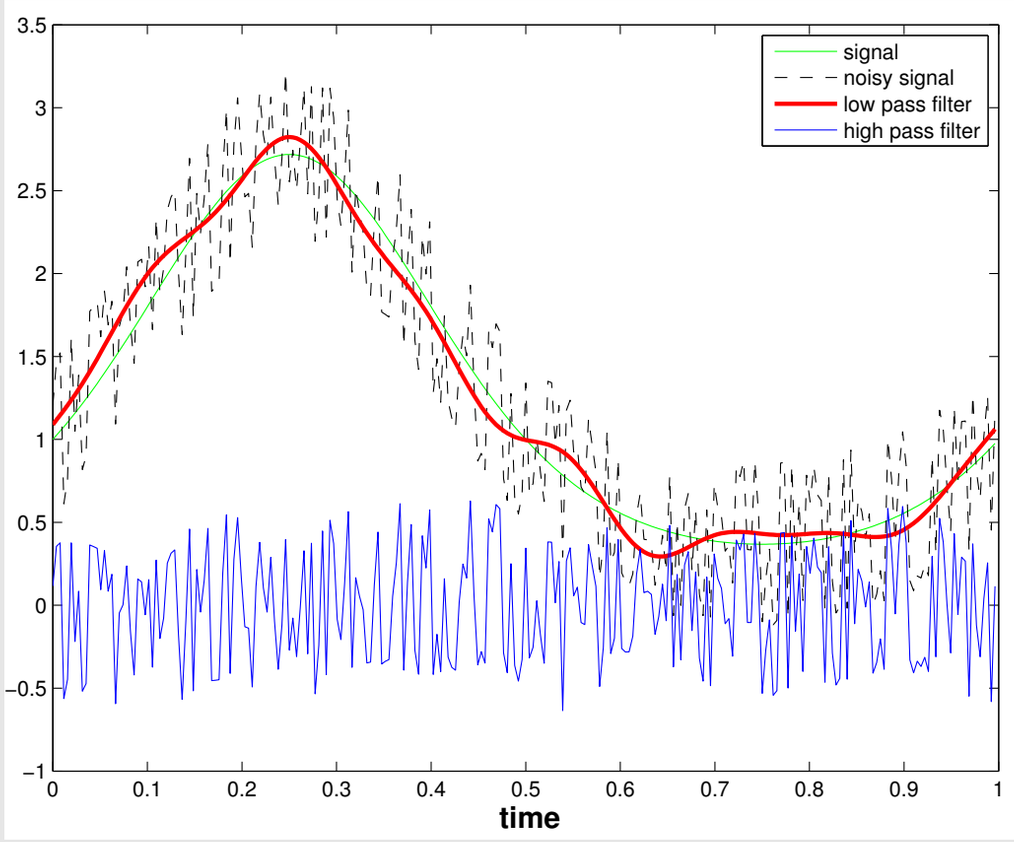
rev 30401, January 28, 2011

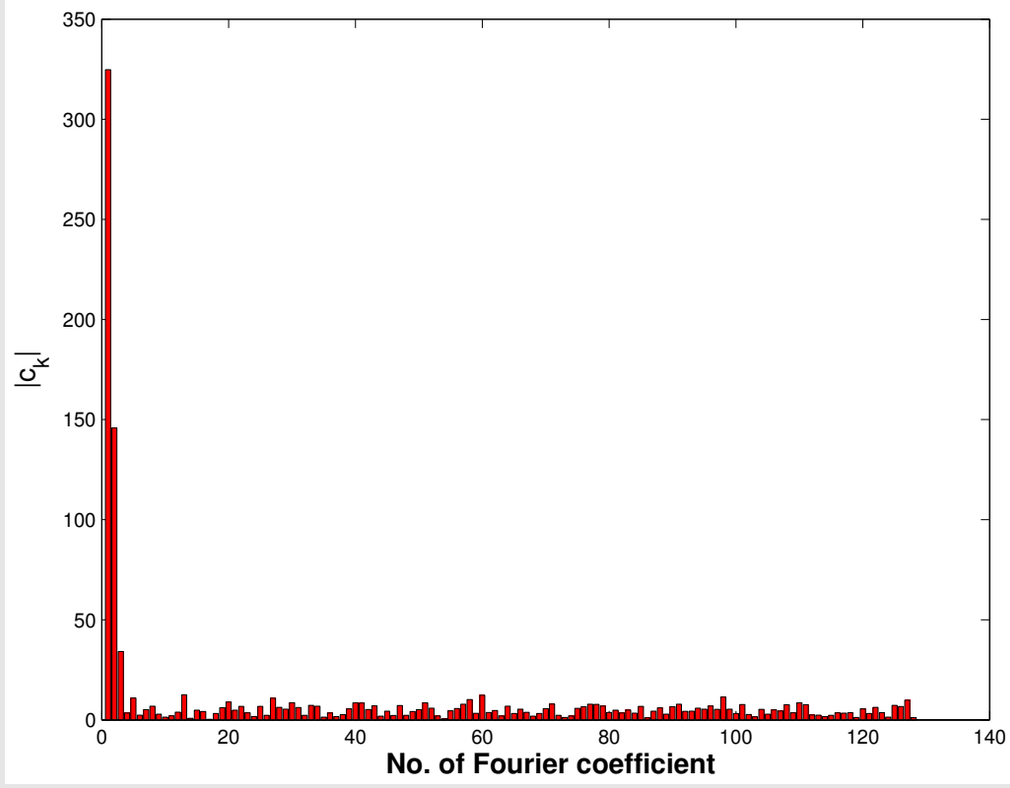
Example 8.2.24 (Frequency filtering by DFT).

Noisy signal:

```
n = 256; y = exp(sin(2*pi*((0:n-1)')/n)) + 0.5*sin(exp(1:n)');
```

Frequency filtering by Code 8.2.22 with $k = 120$.





R. Hiptmair

rev 30401, January 28, 2011

Low pass filtering can be used for *denoising*, that is, the removal of high frequency perturbations of a signal.

◇

Example 8.2.25 (Sound filtering by DFT).

Code 8.2.26: DFT based sound compression

```
% Sound compression by DFT
% Example:  ex:soundfilter

% Read sound data
[y,freq,nbits] = wavread('hello.wav');

n = length(y);
fprintf('Read wav File: %d samples, freq = %d, nbits = %d\n',
  n,freq,nbits);
k = 1; s{k} = y; leg{k} = 'Sampled signal';
```

```
c = fft(y);

figure('name','sound signal');
plot((22000:44000)/freq,s{1}(22000:44000),'r-');
title('samples sound signal','fontsize',14);
xlabel('{\bf time[s]}','fontsize',14);
ylabel('{\bf sound pressure}','fontsize',14);
grid on;

print -depsc2 '../PICTURES/soundsignal.eps';

figure('name','sound frequencies');
plot(1:n,abs(c).^2,'m-');
title('power spectrum of sound signal','fontsize',14);
xlabel('{\bf index k of Fourier coefficient}','fontsize',14);
ylabel('{\bf |c_k|^2}','fontsize',14);
grid on;

print -depsc2 '../PICTURES/soundpower.eps';

figure('name','sound frequencies');
plot(1:3000,abs(c(1:3000)).^2,'b-');
```

```
33 title('low frequency power spectrum','fontsize',14);
34 xlabel('{\bf index k of Fourier coefficient}','fontsize',14);
35 ylabel('{\bf |c_k|^2}','fontsize',14);
36 grid on;
37
38 print -depsc2 '../PICTURES/soundlowpower.eps';
39
40 for m=[1000,3000,5000]
41
42   % Low pass filtering
43   cf = zeros(1,n);
44   cf(1:m) = c(1:m); cf(n-m+1:end) = c(n-m+1:end);
45
46   % Reconstruct filtered signal
47   yf = ifft(cf);
48   wavwrite(yf,freq,nbits,sprintf('hellof%d.wav',m));
49
50   k = k+1;
51   s{k} = real(yf);
52   leg{k} = sprintf('cutt-off = %d',m');
53 end
54
55 % Plot original signal and filtered signals
```

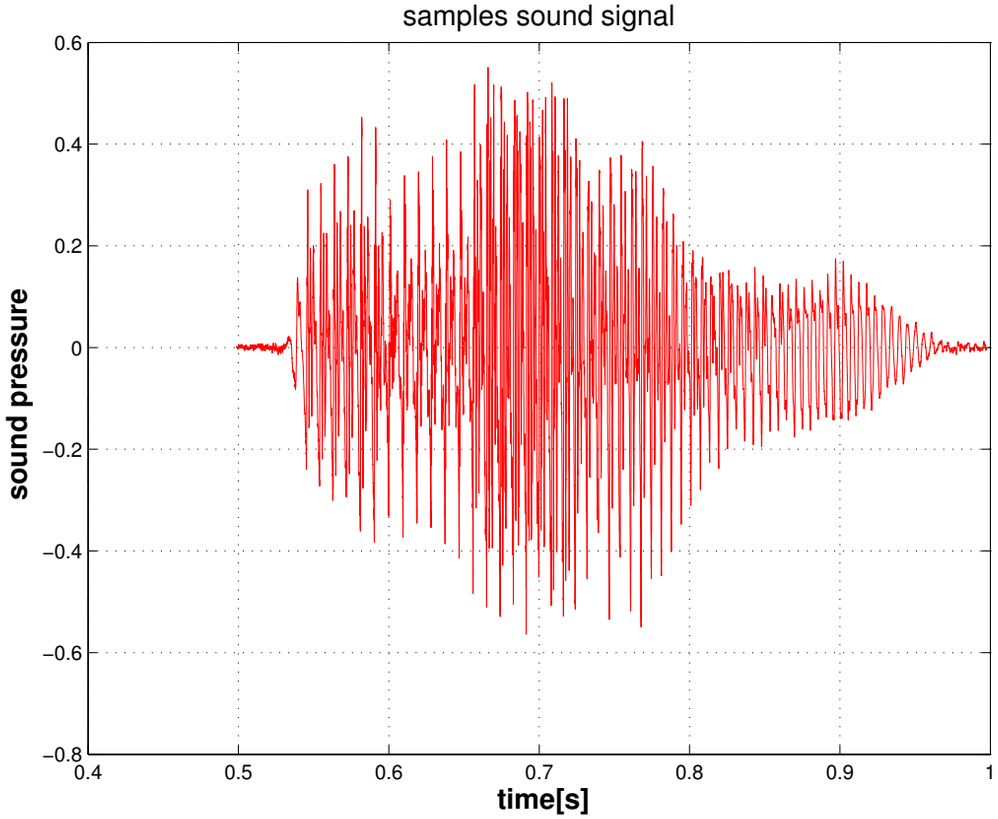

R. Hiptmair

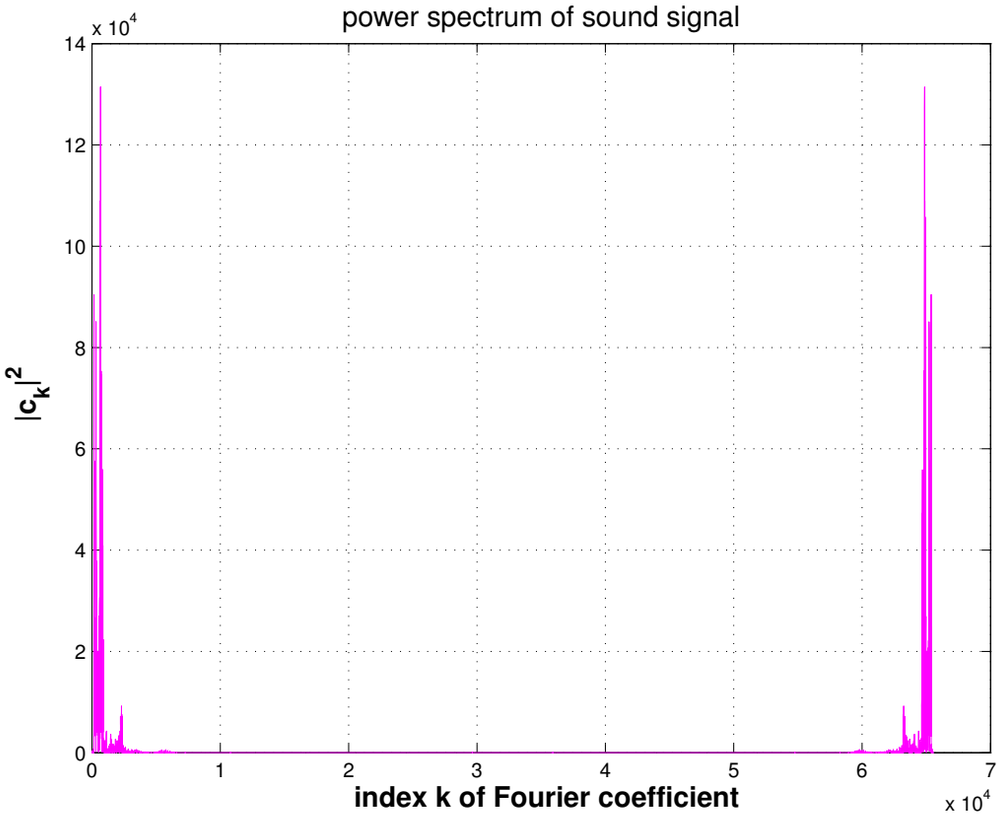
rev 30401, January 28, 2011

```
56 figure('name','sound filtering');
57 plot((30000:32000)/freq,s{1}(30000:32000),'r-',...
58      (30000:32000)/freq,s{2}(30000:32000),'b--',...
59      (30000:32000)/freq,s{3}(30000:32000),'m--',...
60      (30000:32000)/freq,s{2}(30000:32000),'k--');
61 xlabel('{\bf time[s]}','fontsize',14);
62 ylabel('{\bf sound pressure}','fontsize',14);
63 legend(leg,'location','southeast');
64
65 print -depsc2 '../PICTURES/soundfiltered.eps';
```

DFT based low pass frequency filtering of sound

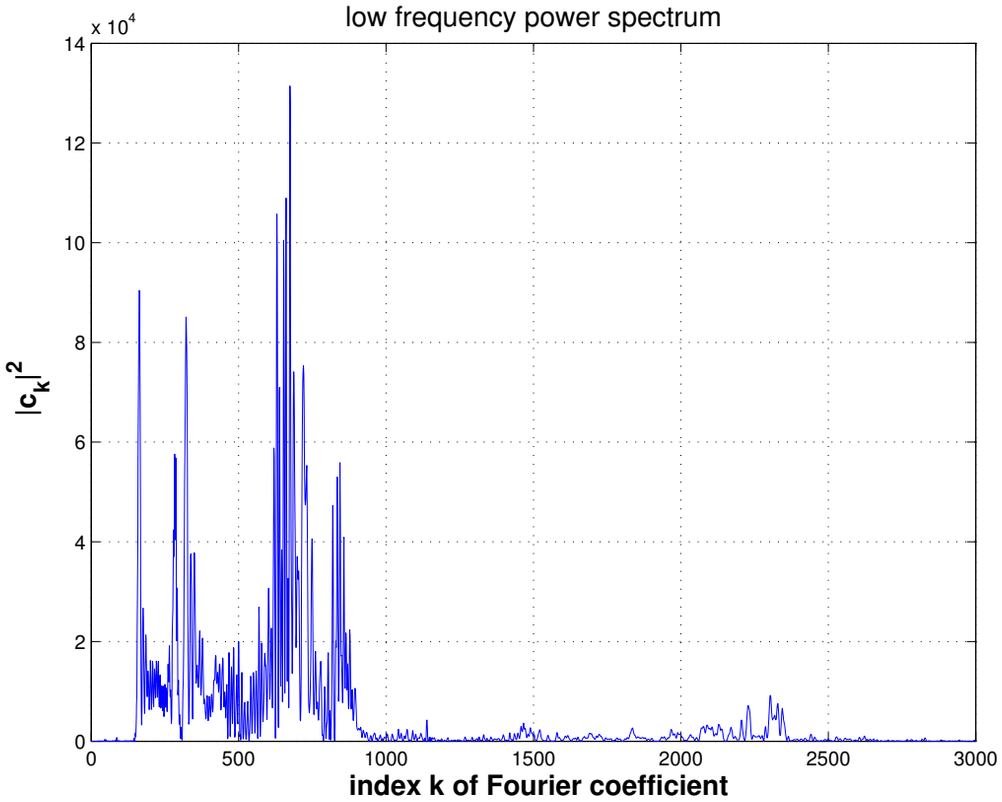
```
[y,sf,nb] = wavread('hello.wav');
c = fft(y); c(m+1:end-m) = 0;
wavwrite(ifft(c),sf,nb,'filtered.wav');
```

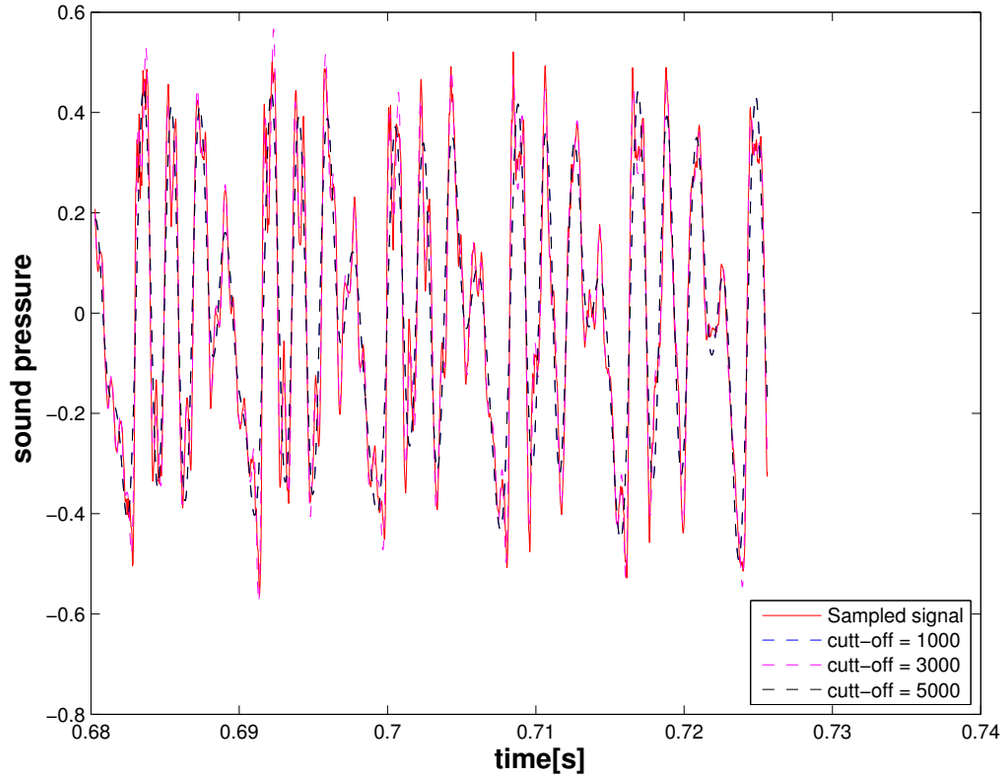




R. Hiptmair

rev 30401, January 28, 2011





R. Hiptmair

rev 30401, January 28, 2011

The power spectrum of a signal $\mathbf{y} \in \mathbb{C}^n$ is the vector $\left(|c_j|^2\right)_{j=0}^{n-1}$, where $\mathbf{c} = \mathbf{F}_n\mathbf{y}$ is the discrete Fourier transform of $\mathbf{y}$.

$\diamond$

8.2.3 Real DFT

Signal obtained from sampling a time-dependent voltage: a real vector.

Aim: efficient DFT (Def. 8.2.14) $(c_0, \ldots, c_{n-1})$ for *real* coefficients $(y_0, \ldots, y_{n-1})^T \in \mathbb{R}^n$, $n = 2m$, $m \in \mathbb{N}$.

If $y_j \in \mathbb{R}$ in DFT formula (8.2.15), we obtain redundant output

$$\omega_n^{(n-k)j} = \overline{\omega}_n^{kj} , \quad k = 0, \ldots, n-1 ,$$

$$\Rightarrow \quad \overline{c}_k = \sum_{j=0}^{n-1} y_j \overline{\omega}_n^{kj} = \sum_{j=0}^{n-1} y_j \omega_n^{(n-k)j} = c_{n-k} , \quad k = 1, \ldots, n-1 .$$

Idea: map $\mathbf{y} \in \mathbb{R}^n$, to $\mathbb{C}^m$ and use DFT of length m.

$$h_k = \sum_{j=0}^{m-1} (y_{2j} + iy_{2j+1})\, \omega_m^{jk} = \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j}\, \omega_m^{jk}} + i \cdot \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j+1}\, \omega_m^{jk}} , \tag{8.2.27}$$

R. Hiptmair

rev 30401, January 28, 2011

$$\overline{h}_{m-k} = \sum_{j=0}^{m-1} \overline{y_{2j} + iy_{2j+1}}\,\overline{\omega}_m^{j(m-k)} = \boxed{\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} - i \cdot \boxed{\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}} \,. \tag{8.2.28}$$

$$\Rightarrow \quad \boxed{\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} = \tfrac{1}{2}(h_k + \overline{h}_{m-k}) \quad , \quad \boxed{\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}} = -\tfrac{1}{2}i(h_k - \overline{h}_{m-k}) \,.$$

Use simple identities for roots of unity:

$$c_k = \sum_{j=0}^{n-1} y_j\,\omega_n^{jk} = \boxed{\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} + \omega_n^k \cdot \boxed{\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}} \,. \tag{8.2.29}$$

$$\Rightarrow \quad \begin{cases} c_k = \frac{1}{2}(h_k + \overline{h}_{m-k}) - \frac{1}{2}i\omega_n^k(h_k - \overline{h}_{m-k}) \,, \quad k = 0, \ldots, m-1 \,, \\ c_m = \mathrm{Re}\{h_0\} - \mathrm{Im}\{h_0\} \,, \\ c_k = \overline{c}_{n-k} \,, \quad k = m+1, \ldots, n-1 \,. \end{cases} \tag{8.2.30}$$

R. Hiptmair

MATLAB-Implementation (by a DFT of length $n/2$):

Code 8.2.31: 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))];
```

(Note: not really optimal MATLAB implementation)

R. Hiptmair

rev 30401, January 28, 2011

8.2.4 Two-dimensional DFT

A natural analogy:

one-dimensional data (“signal”) $\longleftrightarrow$ vector $\mathbf{y} \in \mathbb{C}^n$,

two-dimensional data (“image”) $\longleftrightarrow$ matrix. $\mathbf{Y} \in \mathbb{C}^{m,n}$

Two-dimensional trigonometric basis of $\mathbb{C}^{m,n}$:

tensor product matrices $$\left\{ (\mathbf{F}_m)_{:,j} (\mathbf{F}_n)_{:,\ell}^T,\, 1 \le j \le m,\, 1 \le \ell \le n \right\} . \tag{8.2.32}$$

Basis transform: for $y_{j_1,j_2} \in \mathbb{C}$, $0 \le j_1 < m$, $0 \le j_2 < n$ compute (nested DFTs !)

$$c_{k_1,k_2} = \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} y_{j_1,j_2}\, \omega_m^{j_1 k_1} \omega_n^{j_2 k_2} \quad , \quad 0 \le k_1 < m \,, 0 \le k_2 < n \,.$$

MATLAB function: fft2(Y) .

Two-dimensional DFT by *nested one-dimensional DFTs* (8.2.15):

```
fft2(Y) = fft(fft(Y).').'
```

Here: `.'` simply transposes the matrix (no complex conjugation)

R. Hiptmair

rev 30401, January 28, 2011

Example 8.2.33 (Deblurring by DFT).

Gray-scale pixel image $\mathbf{P} \in \mathbb{R}^{m,n}$, actually $\mathbf{P} \in \{0,\ldots,255\}^{m,n}$, see Ex. 6.3.22.

$\big(p_{l,k}\big)_{l,k\in\mathbb{Z}} \hat{=}$ 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)

$$c_{l,j} = \sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}p_{l+k,j+q}\,, \quad \begin{array}{l} 0 \le l < m\,, \\ 0 \le j < n\,, \end{array} \quad L \in \{1,\ldots,\min\{m,n\}\}\,. \tag{8.2.34}$$

$c_{l,j}$: blurred image; $s_{k,q}$: 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 8.2.35: point spread function

```
1 function S = psf(L)
2 [X,Y] = meshgrid(-L:1:L,-L:1:L);
3 S = 1./(1+X.*X+Y.*Y);
4 S = S/sum(sum(S));
```

R. Hiptmair

rev 30401, January 28, 2011

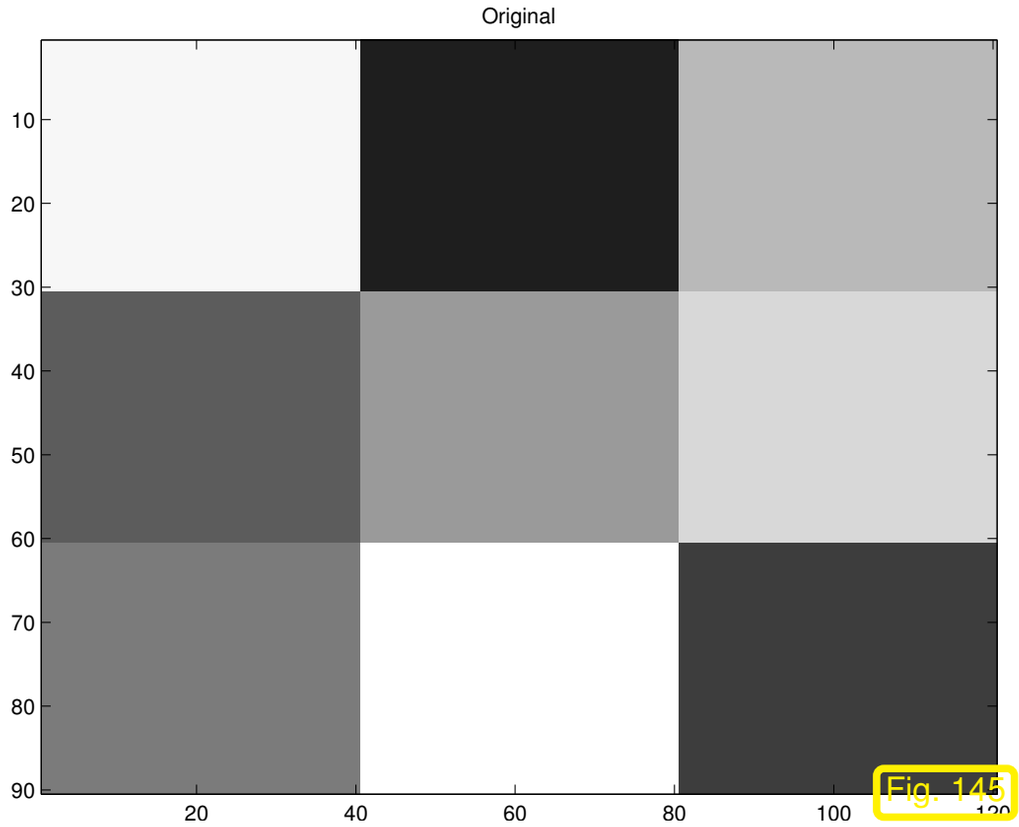


Fig. 145

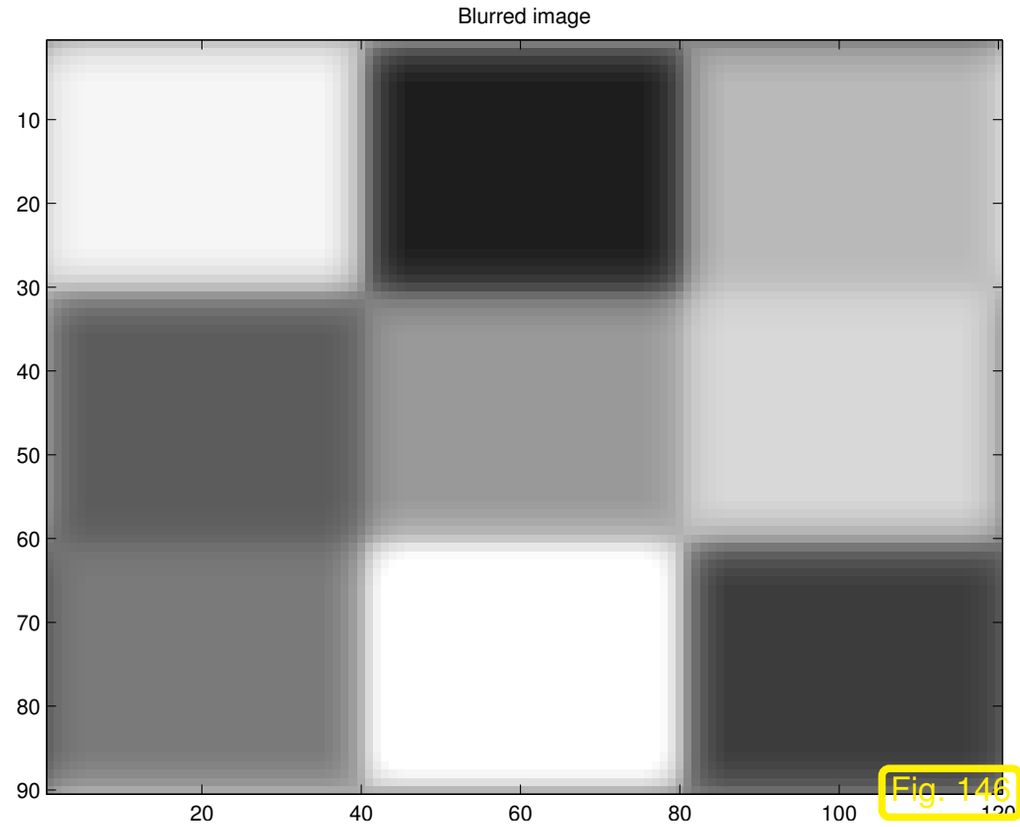


Fig. 146

Code 8.2.36: MATLAB deblurring experiment

```
% Generate artifical "image"
P = kron(magic(3),ones(30,40))*31;
col = [0:1/254:1]'*[1,1,1];
figure; image(P); colormap(col); title('Original');
print -depsc2 '../PICTURES/dborigimage.eps';
% Generate point spread function
```

```
L = 5; S = psf(L);
% Blur image
C = blur(P,S);
figure; image(floor(C)); colormap(col); title('Blurred image');
print -depsc2 '../PICTURES/dbblurredimage.eps';
% Deblur image
D = deblur(C,S);
figure; image(floor(real(D))); colormap(col);
fprintf('Difference of images (Frobenius norm): %f\n',norm(P-D));
```


R. Hiptmair

rev 30401, January 28, 2011

Code 8.2.37: blurring operator

```
1  function C = blur(P,S)
2  [m,n] = size(P); [M,N] = size(S);
3  if (M ~= N), error('S not quadratic'); end
4  L = (M-1)/2; C = zeros(m,n);
5  for l=1:m, for j=1:n
6      s = 0;
7      for k=1:(2*L+1), for q=1:(2*L+1)
8       kl = l+k-L-1;
9       if (kl < 1), kl = kl + m; end
10      if (kl > m), kl = kl - m; end
11      jm = j+q-L-1;
12      if (jm < 1), jm = jm + n; end
13      if (jm > n), jm = jm - n; end
14      s = s+P(kl,jm)*S(k,q);
15       end, end
16     C(l,j) = s;
17   end, end
```

Note:

(8.2.34) 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)`

Recall: derivation of (8.2.7) and Lemma 8.2.12. Try this in 2D!

$$\left(\mathcal{B}(\left(\omega_m^{\nu k}\omega_n^{\mu q}\right)_{k,q\in\mathbb{Z}}\right)_{l,j} = \sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\omega_m^{\nu(l+k)}\omega_n^{\mu(j+q)} = \omega_m^{\nu l}\omega_n^{\mu j}\sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\omega_m^{\nu k}\omega_n^{\mu q}\,.$$

R. Hiptmair

rev 30401, January 28, 2011

$\blacktriangleright$ $\mathbf{V}_{\nu,\mu} := \left(\omega_m^{\nu k}\omega_n^{\mu q}\right)_{k,q\in\mathbb{Z}}$, $0 \le \mu < m$, $0 \le \nu < n$ are the eigenvectors of $\mathcal{B}$:

$$\mathcal{B}\mathbf{V}_{\nu,\mu} = \lambda_{\nu,\mu}\mathbf{V}_{\nu,\mu} \quad , \quad \text{eigenvalue} \quad \lambda_{\nu,\mu} = \underbrace{\sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\omega_m^{\nu k}\omega_n^{\mu q}}_{\text{2-dimensional DFT of point spread function}} \tag{8.2.38}$$

Inversion of blurring operator

$\Updownarrow$

componentwise scaling in “Fourier domain”

Code 8.2.39: DFT based deblurring

```
1  function D = deblur(C,S,tol)
2  [m,n] = size(C); [M,N] = size(S);
3  if (M ~= N), error('S not quadratic'); end
4  L = (M-1)/2; Spad = zeros(m,n);
5  % Zero padding
6  Spad(1:L+1,1:L+1) = S(L+1:end,L+1:end);
7  Spad(m-L+1:m,n-L+1:n) = S(1:L,1:L);
8  Spad(1:L+1,n-L+1:n) = S(L+1:end,1:L);
9  Spad(m-L+1:m,1:L+1) = S(1:L,L+1:end);
10 % Inverse of blurring operator
11 SF = fft2(Spad);
12 % Test for invertibility
13 if (nargin < 3), tol = 1E-3; end
14 if (min(min(abs(SF))) < tol*max(max(abs(SF))))
15   error('Deblurring impossible');
16 end
17 % DFT based deblurring
18 D = fft2(ifft2(C)./SF);
```

$\diamond$

R. Hiptmair

rev 30401, January 28, 2011

8.3 Fast Fourier Transform (FFT)

At first glance (at (8.2.15)): DFT in $\mathbb{C}^n$ seems to require asymptotic computational effort of $O(n^2)$ (matrix×vector multiplication with dense matrix).

Example 8.3.1 (Efficiency of `fft`).

`tic-toc`-timing in MATLAB: compare `fft`, loop based implementation, and direct matrix multiplication
(MATLAB V6.5, Linux, Mobile Intel Pentium 4 - M CPU 2.40GHz, minimum over 5 runs)

Code 8.3.2: timing of different implementations of DFT

```
res = [];
for n=1:1:3000,  y = rand(n,1); c = zeros(n,1);
  t1 = realmax; for k=1:5, tic;
    omega = exp(-2*pi*i/n); c(1) = sum(y); s = omega;
    for j=2:n, c(j) = y(n);
```

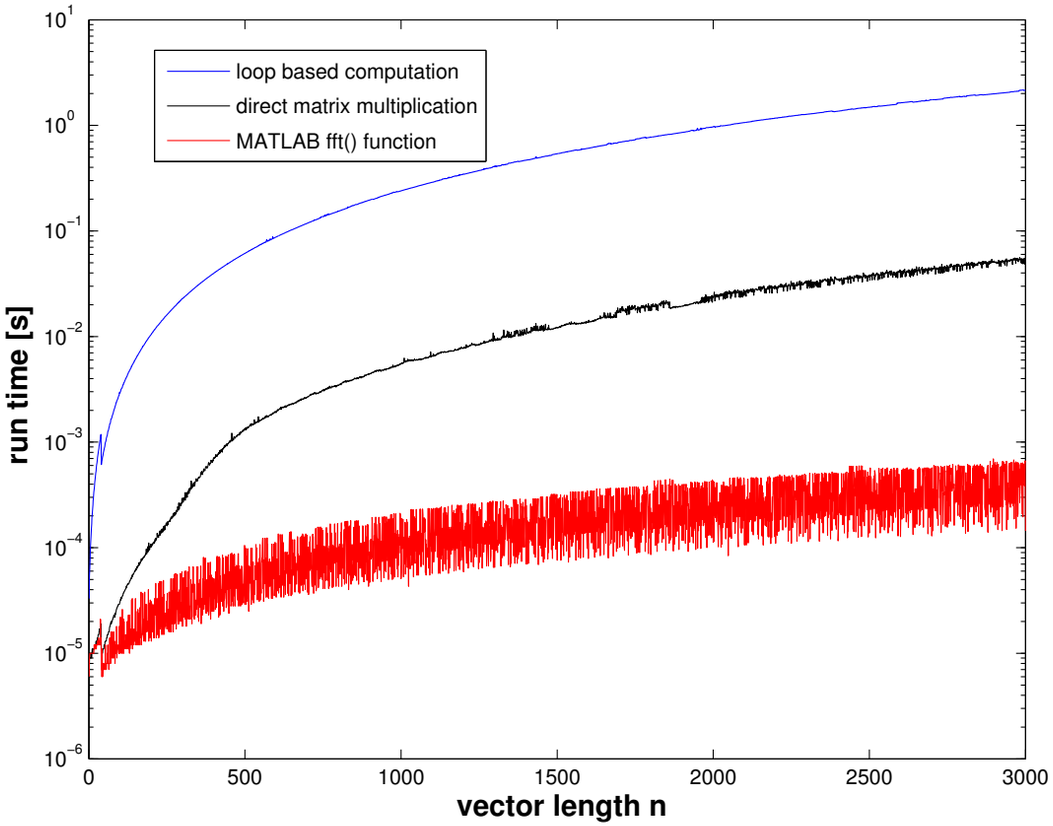
```
      for k=n-1:-1:1, c(j) = c(j)*s+y(k);  end
      s = s*omega;
    end
    t1 = min(t1,toc);
  end
  [I,J] = meshgrid(0:n-1,0:n-1); F = exp(-2*pi*i*I.*J/n);
  t2 = realmax; for k=1:5, tic; c = F*y; t2 = min(t2,toc); end
  t3 = realmax; for k=1:5, tic; d = fft(y); t3 = min(t3,toc); end
  res = [res; n t1 t2 t3];
end

figure('name','FFT timing');
semilogy(res(:,1),res(:,2),'b-',res(:,1),res(:,3),'k-',
  res(:,1),res(:,4),'r-');
ylabel('{\bf run time [s]}','Fontsize',14);
xlabel('{\bf vector length n}','Fontsize',14);
legend('loop based computation','direct matrix
  multiplication','MATLAB fft() function',1);
print -deps2c '../PICTURES/ffttime.eps'
```


R. Hiptmair

rev 30401, January 28, 2011

MATLAB-CODE naive DFT-implementation

```
c = zeros(n,1);
omega = exp(-2*pi*i/n);
c(1) = sum(y); s = omega;
for j=2:n
  c(j) = y(n);
  for k=n-1:-1:1
    c(j) = c(j)*s+y(k);
  end
  s = s*omega;
end
```



Incredible! The MATLAB `fft()`-function clearly beats the $O(n^2)$ asymptotic complexity of the other implementations. Note the logarithmic scale!

◇

R. Hiptmair

rev 30401, January 28, 2011

The secret of MATLAB's `fft()`:

the Fast Fourier Transform algorithm [15]

(discovered by C.F. Gauss in 1805, rediscovered by Cooley & Tuckey in 1965, one of the "top ten algorithms of the century").

An elementary manipulation of (8.2.15) for $n = 2m$, $m \in \mathbb{N}$:

$$\begin{aligned} c_k &= \sum_{j=0}^{n-1} y_j e^{-\frac{2\pi i}{n} jk} \\ &= \sum_{j=0}^{m-1} y_{2j} e^{-\frac{2\pi i}{n} 2jk} + \sum_{j=0}^{m-1} y_{2j+1} e^{-\frac{2\pi i}{n}(2j+1)k} \\ &= \underbrace{\sum_{j=0}^{m-1} y_{2j} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}}}_{=:\widetilde{c}_k^{\mathtt{even}}} + e^{-\frac{2\pi i}{n} k} \cdot \underbrace{\sum_{j=0}^{m-1} y_{2k+1} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}}}_{=:\widetilde{c}_k^{\mathtt{odd}}} . \end{aligned} \tag{8.3.3}$$

Note m-periodicity: $\widetilde{c}_k^{\mathtt{even}} = \widetilde{c}_{k+m}^{\mathtt{even}}$, $\widetilde{c}_k^{\mathtt{odd}} = \widetilde{c}_{k+m}^{\mathtt{odd}}$.

R. Hiptmair

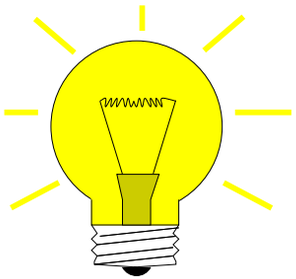
rev 30401, January 28, 2011

Note: $\widetilde{c}_k^{\mathtt{even}}$, $\widetilde{c}_k^{\mathtt{odd}}$ from DFTs of length m!

with $\mathbf{y}_{\mathtt{even}} := (y_0, y_2, \ldots, y_{n-2})^T \in \mathbb{C}^m$: $\left(\widetilde{c}_k^{\mathtt{even}}\right)_{k=0}^{m-1} = \mathbf{F}_m \mathbf{y}_{\mathtt{even}}$,

with $\mathbf{y}_{\mathtt{odd}} := (y_1, y_3, \ldots, y_{n-1})^T \in \mathbb{C}^m$: $\left(\widetilde{c}_k^{\mathtt{odd}}\right)_{k=0}^{m-1} = \mathbf{F}_m \mathbf{y}_{\mathtt{odd}}$.

(8.3.3): DFT of length $2m$ = 2× DFT of length m + 2m additions & multiplications



Idea:

divide & conquer recursion

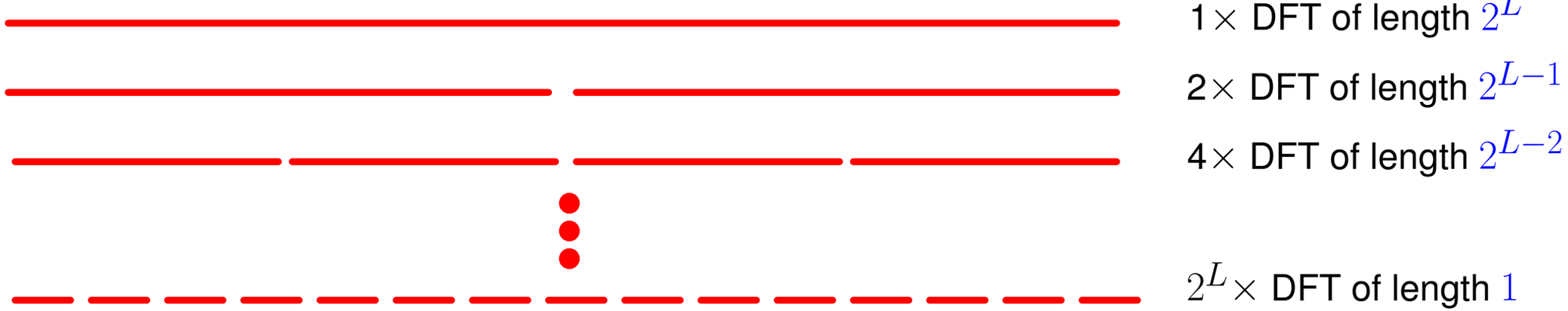
(for DFT of length $n = 2^L$)

FFT-algorithm

Code 8.3.4: Recursive FFT

```
1 function c = fftrec(y)
2 n = length(y);
3 if (n == 1), c = y; return;
4 else
5   c1 = fftrec(y(1:2:n));
6   c2 = fftrec(y(2:2:n));
7   c = [c1;c1] +
      (exp(-2*pi*i/n).^((0:n-1)'))
      .*[c2;c2];
8 end
```

Computational cost of `fftrec`:



Code 8.3.3: each level of the recursion requires $O(2^L)$ elementary operations.

> Asymptotic complexity of FFT algorithm, $n = 2^L$: $O(L2^L) = O(n \log_2 n)$
>
> (MATLAB `fft`-function: cost $\approx 5n \log_2 n$).

➢

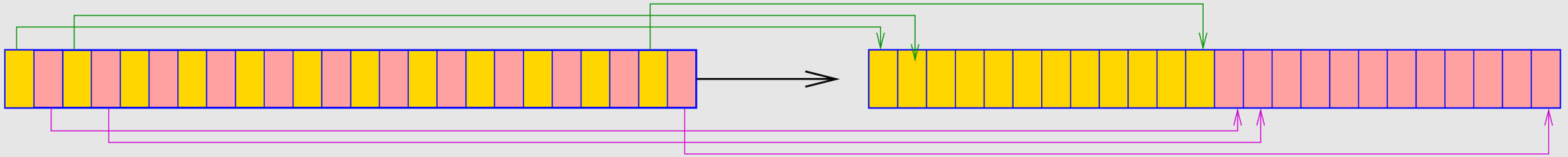
Remark 8.3.5 (FFT algorithm by matrix factorization).

For $n = 2m$, $m \in \mathbb{N}$,

$$\text{permutation} \quad P_m^{\mathrm{OE}}(1, \ldots, n) = (1, 3, \ldots, n-1, 2, 4, \ldots, n) \ .$$

R. Hiptmair

rev 30401, January 28, 2011



$$\omega_n^{2j} = \omega_m^j$$

permutation of rows

$$P_m^{\mathrm{OE}}\mathbf{F}_n = \left(\begin{array}{c|c} \mathbf{F}_m & \mathbf{F}_m \\ \hline \mathbf{F}_m \begin{pmatrix} \omega_n^0 & & & \\ & \omega_n^1 & & \\ & & \ddots & \\ & & & \omega_n^{n/2-1} \end{pmatrix} & \mathbf{F}_m \begin{pmatrix} \omega_n^{n/2} & & & \\ & \omega_n^{n/2+1} & & \\ & & \ddots & \\ & & & \omega_n^{n-1} \end{pmatrix} \end{array}\right)$$

$$\left(\begin{array}{c|c} \mathbf{F}_m & \\ \hline & \mathbf{F}_m \end{array}\right) \left(\begin{array}{c|c} \mathbf{I} & \mathbf{I} \\ \hline \begin{pmatrix} \omega_n^0 & & & \\ & \omega_n^1 & & \\ & & \ddots & \\ & & & \omega_n^{n/2-1} \end{pmatrix} & \begin{pmatrix} -\omega_n^0 & & & \\ & -\omega_n^1 & & \\ & & \ddots & \\ & & & -\omega_n^{n/2-1} \end{pmatrix} \end{array}\right)$$

R. Hiptmair

rev 30401, January 28, 2011

Example: factorization of Fourier matrix for $n = 10$

$$P_5^{\mathrm{OE}}\mathbf{F}_{10} = \left(\begin{array}{ccccc|ccccc} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^8 & \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^8 \\ \omega^0 & \omega^4 & \omega^8 & \omega^2 & \omega^6 & \omega^0 & \omega^4 & \omega^8 & \omega^2 & \omega^6 \\ \omega^0 & \omega^6 & \omega^2 & \omega^8 & \omega^4 & \omega^0 & \omega^6 & \omega^2 & \omega^8 & \omega^4 \\ \omega^0 & \omega^8 & \omega^6 & \omega^4 & \omega^2 & \omega^0 & \omega^8 & \omega^6 & \omega^4 & \omega^2 \\ \hline \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 & \omega^8 & \omega^9 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 & \omega^2 & \omega^5 & \omega^8 & \omega^1 & \omega^4 & \omega^7 \\ \omega^0 & \omega^5 & \omega^0 & \omega^5 & \omega^0 & \omega^5 & \omega^0 & \omega^5 & \omega^0 & \omega^5 \\ \omega^0 & \omega^7 & \omega^4 & \omega^1 & \omega^8 & \omega^5 & \omega^2 & \omega^9 & \omega^6 & \omega^3 \\ \omega^0 & \omega^9 & \omega^8 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{array}\right), \quad \omega := \omega_{10} .$$

△

What if $n \neq 2^L$? Quoted from MATLAB manual:

To compute an n-point DFT when n is composite (that is, when $n = pq$), the FFTW library decomposes the problem using the Cooley-Tukey algorithm, which first computes p transforms of size q,

R. Hiptmair

rev 30401, January 28, 2011

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.

Remark 8.3.6 (FFT based on general factorization).

Fast Fourier transform algorithm for DFT of length $n = pq$, $p, q \in \mathbb{N}$ (Cooley-Tuckey-Algorithm)

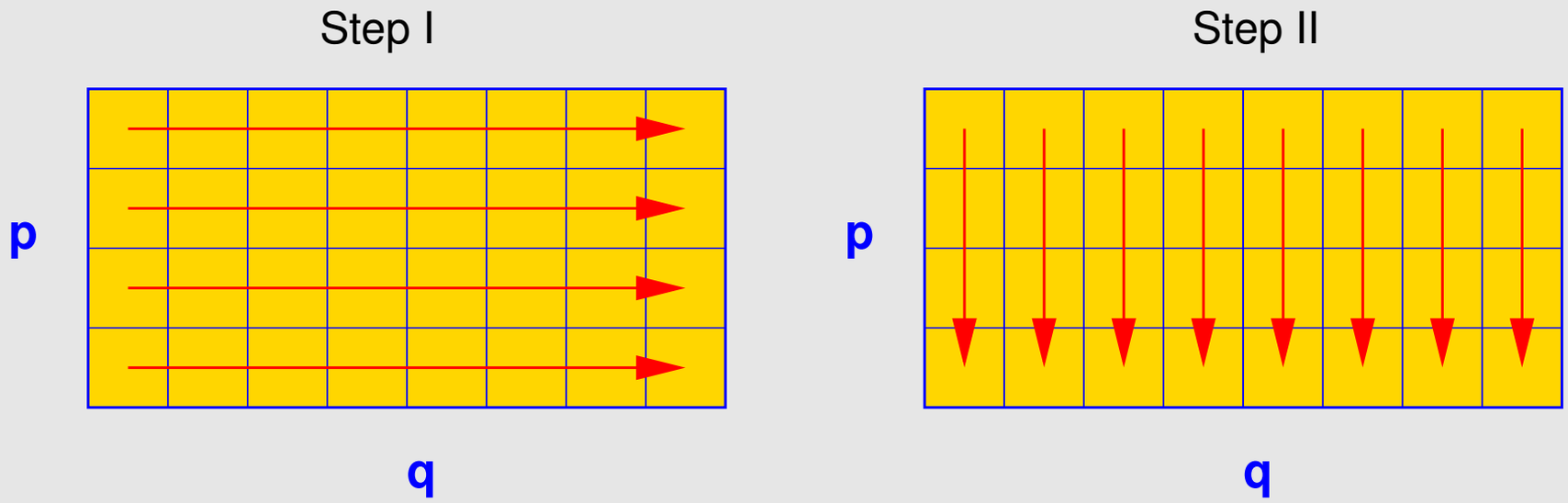
$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{jk} \overset{[j=:lp+m]}{=} \sum_{m=0}^{p-1} \sum_{l=0}^{q-1} y_{lp+m} e^{-\frac{2\pi i}{pq}(lp+m)k} = \sum_{m=0}^{p-1} \omega_n^{mk} \sum_{l=0}^{q-1} y_{lp+m} \, \omega_q^{l(k \mod q)}. \tag{8.3.7}$$

Step I: perform p DFTs of length q $\quad z_{m,k} := \sum_{l=0}^{q-1} y_{lp+m} \, \omega_q^{lk}, \quad 0 \le m < p, 0 \le k < q.$

Step II: for $k =: rq + s, \quad 0 \leq r < p, 0 \leq s < q$

$$c_{rq+s} = \sum_{m=0}^{p-1} e^{-\frac{2\pi i}{pq}(rq+s)m} z_{m,s} = \sum_{m=0}^{p-1} \left(\omega_n^{ms} z_{m,s}\right) \omega_p^{mr}$$

and hence q DFTs of length p give all c_k.



△

R. Hiptmair

rev 30401, January 28, 2011

Remark 8.3.8 (FFT for prime n).

When $n \neq 2^L$, even the Cooley-Tuckey algorithm of Rem. 8.3.6 will eventually lead to a DFT for a vector with prime length.

Quoted from the MATLAB manual:

When n is a prime number, the FFTW library first decomposes an n-point problem into three $(n-1)$-point problems using Rader's algorithm [43]. It then uses the Cooley-Tukey decomposition described above to compute the $(n-1)$-point DFTs.

Details of Rader's algorithm: a theorem from number theory:

$$\forall p \in \mathbb{N} \text{ prime} \quad \exists g \in \{1,\ldots,p-1\}\text{:} \quad \{g^k \mod p\text{: } k = 1,\ldots,p-1\} = \{1,\ldots,p-1\}\,,$$

permutation $P_{p,g} : \{1,\ldots,p-1\} \mapsto \{1,\ldots,p-1\}\,, \quad P_{p,g}(k) = g^k \mod p\,,$
reversing permutation $P_k : \{1,\ldots,k\} \mapsto \{1,\ldots,k\}\,, \quad P_k(i) = k - i + 1\,.$

For Fourier matrix $\mathbf{F} = (f_{ij})_{i,j=1}^{p}$: $P_{p-1}P_{p,g}(f_{ij})_{i,j=2}^{p}P_{p,g}^T$ is circulant.

Example for $p = 13$:
$g = 2$, permutation: $(2\ 4\ 8\ 3\ 6\ 12\ 11\ 9\ 5\ 10\ 7\ 1)$.

$$\mathbf{F}_{13} \longrightarrow \left[\begin{array}{c|cccccccccccc}
\omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \hline
\omega^0 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 \\
\omega^0 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 \\
\omega^0 & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} \\
\omega^0 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 \\
\omega^0 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 \\
\omega^0 & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} \\
\omega^0 & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} \\
\omega^0 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 \\
\omega^0 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 \\
\omega^0 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 \\
\omega^0 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 \\
\omega^0 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2
\end{array}\right]$$

Then apply fast (FFT based!) algorithms for multiplication with circulant matrices to right lower $(n-1) \times (n-1)$ block of permuted Fourier matrix .

△

Asymptotic complexity of `c=fft(y)` for $\mathbf{y} \in \mathbb{C}^n$ = $O(n \log n)$.

R. Hiptmair

rev 30401, January 28, 2011

← Sect. 8.2.1

Asymptotic complexity of discrete periodic convolution/multiplication with circulant matrix, see Code 8.2.17:

$$\text{Cost}(\texttt{z = pconvfft(u,x)}, \mathbf{u}, \mathbf{x} \in \mathbb{C}^n) = O(n \log n).$$

Asymptotic complexity of discrete convolution, see Code 8.2.19:

$$\text{Cost}(\texttt{z = myconv(h,x)}, \mathbf{h}, \mathbf{x} \in \mathbb{C}^n) = O(n \log n).$$

8.4 Trigonometric transformations

Keeping in mind $\exp(2\pi ix) = \cos(2\pi x) + i \sin(2\pi x)$ we may also consider the real/imaginary parts of the Fourier basis vectors $(\mathbf{F}_n)_{:,j}$ as bases of $\mathbb{R}^n$ and define the corresponding basis transformation. They can all be realized by means of `fft` with an asymptotic computational effort of $O(n \log n)$.

Details are given in the sequel.

8.4.1 Sine transform

Another trigonometric basis transform in $\mathbb{R}^{n-1}$, $n \in \mathbb{N}$:

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 $\to$ standard basis): $\mathbf{S}_n := (\sin(jk\pi/n))_{j,k=1}^{n-1} \in \mathbb{R}^{n-1,n-1}$.

Lemma 8.4.1 (Properties of the sine matrix)**.**

$\sqrt{2/n}\,\mathbf{S}_n$ *is real, symmetric and orthogonal (*$\to$ *Def. 2.8.5)*

R. Hiptmair

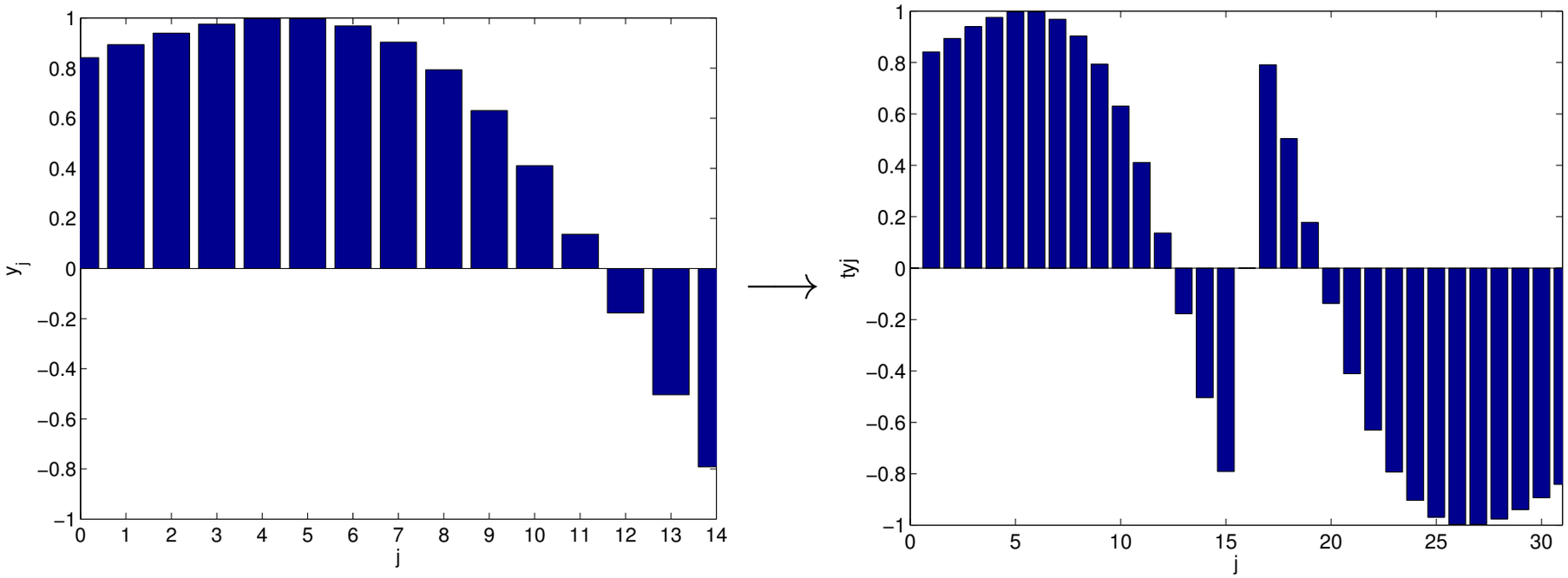
rev 30401, January 28, 2011

Sine transform:

$$\boxed{s_k = \sum_{j=1}^{n-1} y_j \sin(\pi jk/n)} \quad , \quad k = 1, \ldots, n-1 \, . \tag{8.4.2}$$

DFT-based algorithm for the sine transform ($\hat{=}$ $\mathbf{S}_n \times$vector):

"wrap around": $\widetilde{\mathbf{y}} \in \mathbb{R}^{2n}$: $\widetilde{y}_j = \begin{cases} y_j & \text{, if } j = 1, \ldots, n-1 \, , \\ 0 & \text{, if } j = 0, n \, , \\ -y_{2n-j} & \text{, if } j = n+1, \ldots, 2n-1 \, . \end{cases}$ ($\widetilde{\mathbf{y}}$ "odd")



$$
\begin{aligned}
(\mathbf{F}_{2n}\widetilde{\mathbf{y}})_k &\overset{(8.2.15)}{=} \sum_{j=1}^{2n-1} \widetilde{y}_j e^{-\frac{2\pi}{2n}kj} \\
&= \sum_{j=1}^{n-1} y_j e^{-\frac{\pi}{n}kj} - \sum_{j=n+1}^{2n-1} y_{2n-j} e^{-\frac{\pi}{n}kj} \\
&= \sum_{j=1}^{n-1} y_j (e^{-\frac{\pi}{n}kj} - e^{\frac{\pi}{n}kj}) \\
&= -2i\,(\mathbf{S}_n\mathbf{y})_k \quad ,k = 1,\ldots,n-1 \,.
\end{aligned}
$$

Wrap-around implementation

```
function c = sinetrans(y)
n = length(y)+1;
yt = [0,y,0,-y(end:-1:1)];
ct = fft(yt);
c = -ct(2:n)/(2*i);
```

MATLAB-CODE sine transform

Remark 8.4.3 (Sine transform via DFT of half length).

Step ①: transform of the coefficients

$$\widetilde{y}_j = \sin(j\pi/n)(y_j + y_{n-j}) + \tfrac{1}{2}(y_j - y_{n-j}) \,, \quad j = 1,\ldots,n-1 \quad , \quad \widetilde{y}_0 = 0 \,.$$

Step ②: real DFT (→ Sect. 8.2.3) of $(\widetilde{y}_0, \ldots, \widetilde{y}_{n-1}) \in \mathbb{R}^n$: $\quad c_k := \sum_{j=0}^{n-1} \widetilde{y}_j \, e^{-\frac{2\pi i}{n} jk}$

Hence
$$\begin{aligned}
\mathrm{Re}\{c_k\} &= \sum_{j=0}^{n-1} \widetilde{y}_j \, \cos(-\tfrac{2\pi i}{n} jk) = \sum_{j=1}^{n-1} (y_j + y_{n-j}) \sin(\tfrac{\pi j}{n}) \cos(\tfrac{2\pi i}{n} jk) \\
&= \sum_{j=0}^{n-1} 2y_j \sin(\tfrac{\pi j}{n}) \cos(\tfrac{2\pi i}{n} jk) = \sum_{j=0}^{n-1} y_j \left(\sin(\frac{2k+1}{n}\pi j) - \sin(\frac{2k-1}{n}\pi j) \right) \\
&= s_{2k+1} - s_{2k-1} \, . \\
\mathrm{Im}\{c_k\} &= \sum_{j=0}^{n-1} \widetilde{y}_j \sin(-\tfrac{2\pi i}{n} jk) = -\sum_{j=1}^{n-1} \tfrac{1}{2}(y_j - y_{n-j}) \sin(\tfrac{2\pi i}{n} jk) = -\sum_{j=1}^{n-1} y_j \sin(\tfrac{2\pi i}{n} jk) \\
&= -s_{2k} \, .
\end{aligned}$$

Step ③: extraction of s_k

s_{2k+1}, $\quad k = 0, \ldots, \frac{n}{2} - 1$ ➤ from recursion $s_{2k+1} - s_{2k-1} = \mathrm{Re}\{c_k\}$, $\quad s_1 = \sum_{j=1}^{n-1} y_j \, \sin(\pi j/n)$,

s_{2k}, $\quad k = 1, \ldots, \frac{n}{2} - 2$ ➤ $s_{2k} = -\mathrm{Im}\{c_k\}$.

MATLAB-Implementation (via a `fft` of length $n/2$):

R. Hiptmair

rev 30401, January 28, 2011

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
```


△

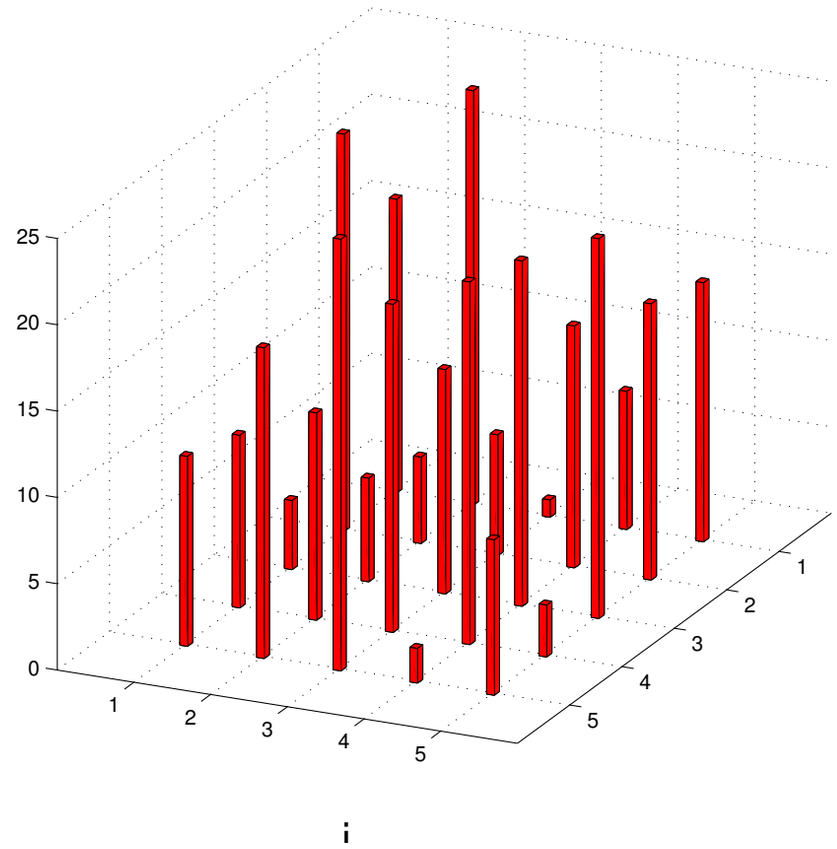
Application: diagonalization of local translation invariant linear operators.

5-points-stencil-operator on $\mathbb{R}^{n,n}$, $n \in \mathbb{N}$, in grid representation:

$$T : \mathbb{R}^{n,n} \mapsto \mathbb{R}^{n,n}, \quad \begin{aligned} \mathbf{X} &\mapsto T(\mathbf{X}) \\ (T(\mathbf{X}))_{ij} &:= cx_{ij} + c_y x_{i,j+1} + c_y x_{i,j-1} + c_x x_{i+1,j} + c_x x_{i-1,j} \end{aligned}$$

with $c, c_y, c_x \in \mathbb{R}$, convention: $x_{ij} := 0$ for $(i,j) \notin \{1,\ldots,n\}^2$.

$$\mathbf{X} \in \mathbb{R}^{n,n}$$
$$\updownarrow$$
grid function $\in \{1,\ldots,n\}^2 \mapsto \mathbb{R}$



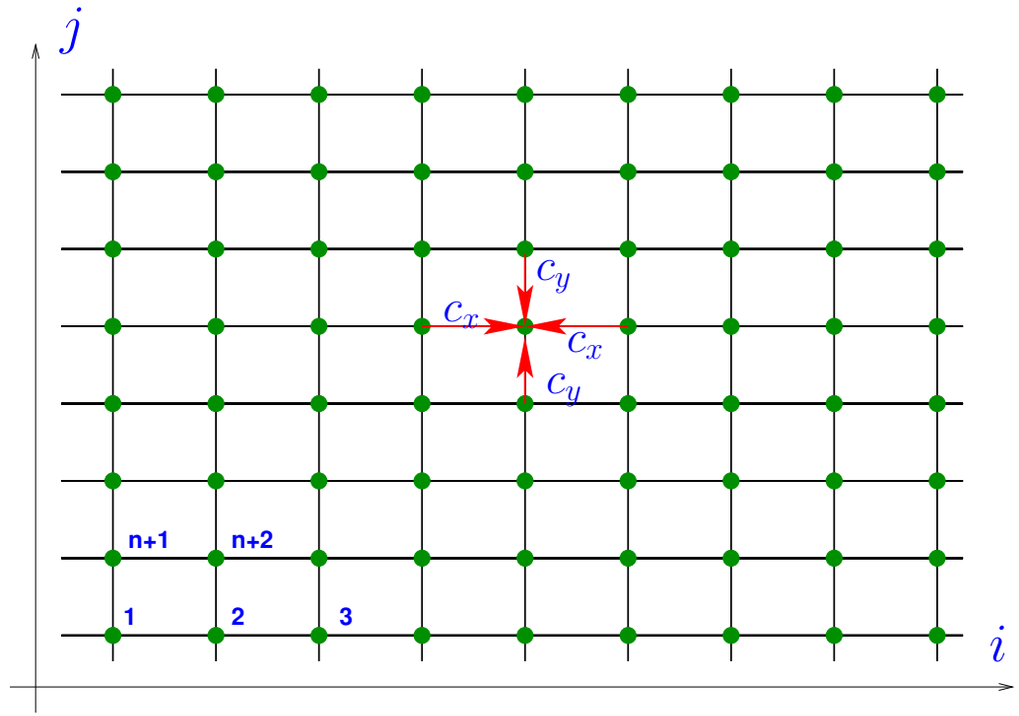
Identification $\mathbb{R}^{n,n} \cong \mathbb{R}^{n^2}$, $x_{ij} \sim \widetilde{x}_{(j-1)n+i}$ gives matrix representation $\mathbf{T} \in \mathbb{R}^{n^2,n^2}$ of T:

R. Hiptmair

rev 30401, January 28, 2011

$$\mathbf{T} = \begin{pmatrix} \mathbf{C} & c_y\mathbf{I} & 0 & \cdots & \cdots & 0 \\ c_y\mathbf{I} & \mathbf{C} & c_y\mathbf{I} & & & \vdots \\ 0 & \ddots & \ddots & \ddots & & \\ \vdots & & & c_y\mathbf{I} & \mathbf{C} & c_y\mathbf{I} \\ 0 & \cdots & \cdots & 0 & c_y\mathbf{I} & \mathbf{C} \end{pmatrix} \in \mathbb{R}^{n^2,n^2} ,$$

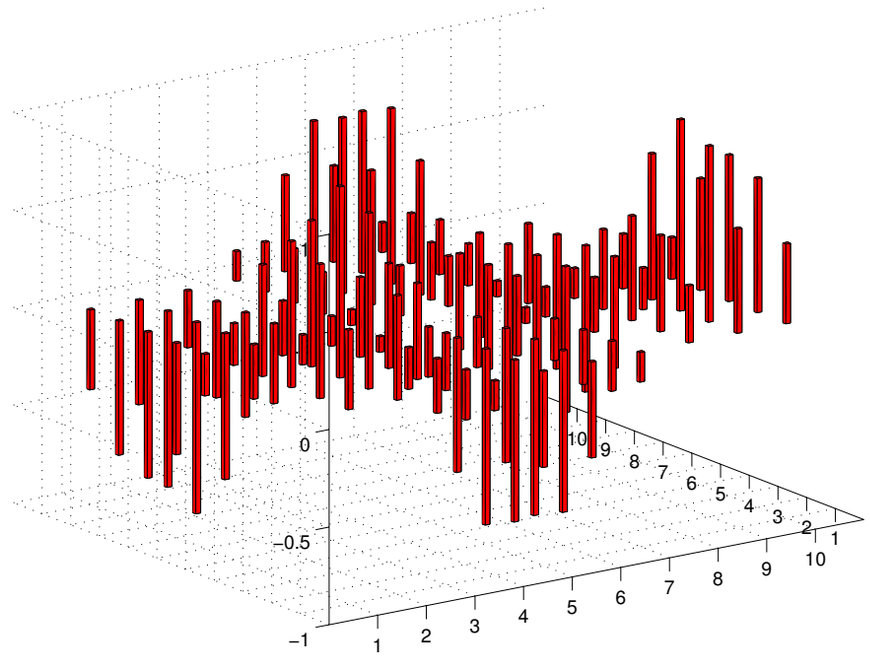
$$\mathbf{C} = \begin{pmatrix} c & c_x & 0 & \cdots & \cdots & 0 \\ c_x & c & c_x & & & \vdots \\ 0 & \ddots & \ddots & \ddots & & \\ \vdots & & & c_x & c & c_x \\ 0 & \cdots & \cdots & 0 & c_x & c \end{pmatrix} \in \mathbb{R}^{n,n} .$$



Sine basis of $\mathbb{R}^{n,n}$:

$$\mathbf{B}^{kl} = \left(\sin(\tfrac{\pi}{n+1}ki)\sin(\tfrac{\pi}{n+1}lj)\right)_{i,j=1}^{n} . \quad (8.4.4)$$

$n = 10$: grid function $\mathbf{B}^{2,3}$ ➢



R. Hiptmair

rev 30401, January 28, 2011

$$
\begin{aligned}
(T(\mathbf{B}^{kl}))_{ij} &= c\sin(\tfrac{\pi}{n}ki)\sin(\tfrac{\pi}{n}lj) + c_y\sin(\tfrac{\pi}{n}ki)\left(\sin(\tfrac{\pi}{n+1}l(j-1)) + \sin(\tfrac{\pi}{n+1}l(j+1))\right) + \\
&\qquad c_x\sin(\tfrac{\pi}{n}lj)\left(\sin(\tfrac{\pi}{n+1}k(i-1)) + \sin(\tfrac{\pi}{n+1}k(i+1))\right) \\
&= \sin(\tfrac{\pi}{n}ki)\sin(\tfrac{\pi}{n}lj)(c + 2c_y\cos(\tfrac{\pi}{n+1}l) + 2c_x\cos(\tfrac{\pi}{n+1}k))
\end{aligned}
$$

Hence $\mathbf{B}^{kl}$ is eigenvector of $T \leftrightarrow \mathbf{T}$ corresponding to eigenvalue $c + 2c_y\cos(\frac{\pi}{n+1}l) + 2c_x\cos(\frac{\pi}{n+1}k)$.

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 (→ 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;
```

R. Hiptmair

rev 30401, January 28, 2011

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

$\Downarrow$

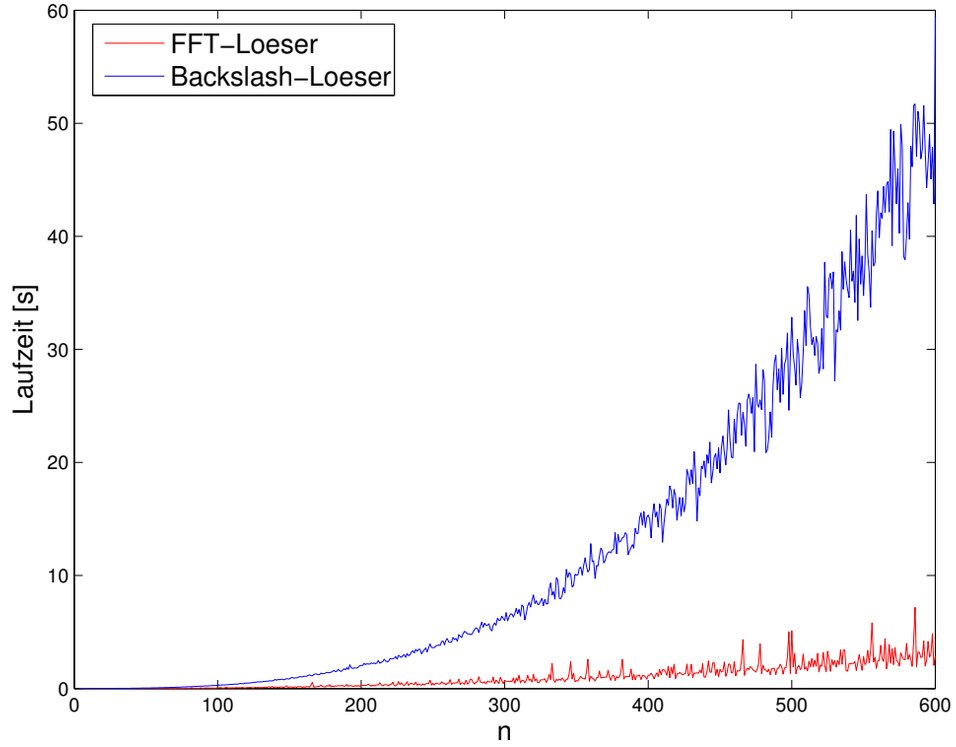
efficient algorithm for solving linear system of equations $T(\mathbf{X}) = \mathbf{B}$

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):

$$\mathbf{C}_n = (c_{ij}) \in \mathbb{R}^{n,n} \quad \text{with} \quad c_{ij} = \begin{cases} 2^{-1/2} & \text{, if } i = 1 \text{ ,} \\ \cos((i-1)\frac{2j-1}{2n}\pi) & \text{, if } i > 1 \text{ .} \end{cases}$$

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 30401, January 28, 2011

Note: $\mathbf{C}_n$ is not symmetric

cosine transform:

$$\boxed{c_k = \sum_{j=0}^{n-1} y_j \cos(k\tfrac{2j+1}{2n}\pi)} \quad , \quad k = 1, \ldots, n-1 \, , \tag{8.4.7}$$

$$c_0 = \frac{1}{\sqrt{2}} \sum_{j=0}^{n-1} y_j \, .$$

MATLAB-implementation of $\mathbf{Cy}$ ("wrapping"-technique):

MATLAB-CODE cosine transform

```
function c = costrans(y)
n = length(y);
z = fft([y,y(end:-1:1)]);
c = real([z(1)/(2*sqrt(2)), ...
      0.5*(exp(-i*pi/(2*n)).^(1:n-1)).*z(2:n)]);
```

MATLAB-implementation of $\mathbf{C}_n^{-1}\mathbf{y}$ ("Wrapping"-technique):

R. Hiptmair

rev 30401, January 28, 2011

MATLAB-CODE : Inverse cosine transform

```
function y=icostrans(c)
n = length(c);
y = [sqrt(2)*c(1),(exp(i*pi/(2*n)).^(1:n-1)).*c(2:end)];
y = ifft([y,0,conj(y(end:-1:2))]);
y = 2*y(1:n);
```

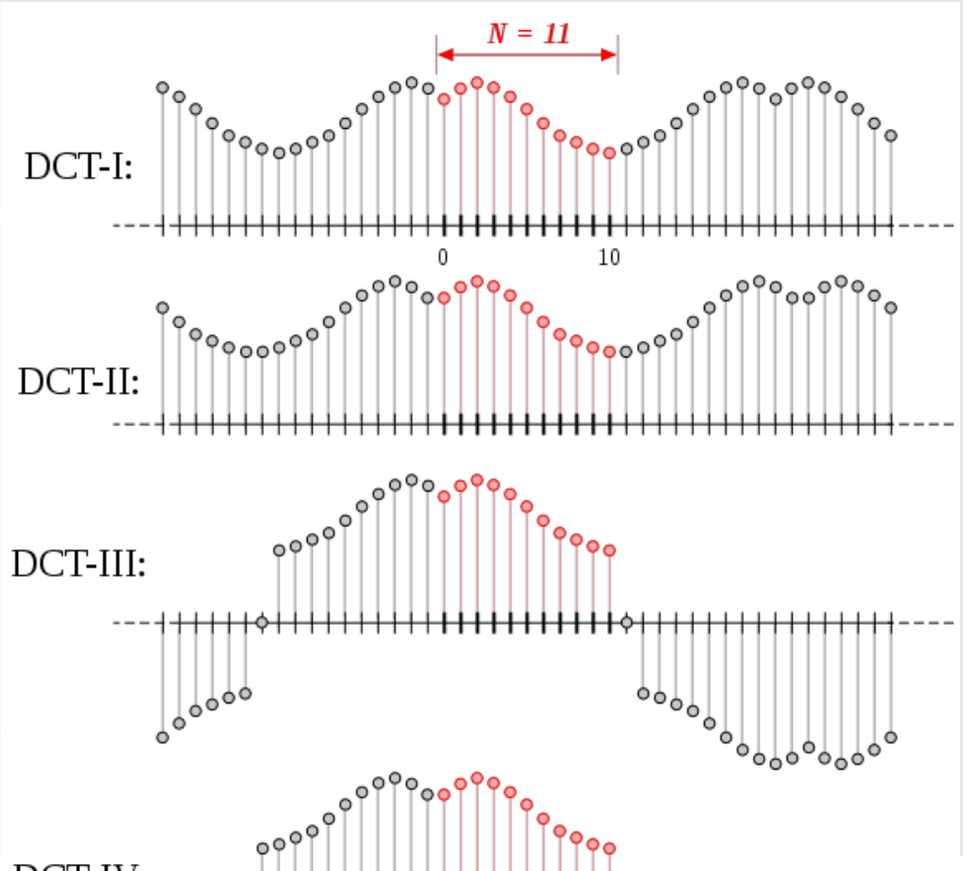
Remark 8.4.8 (Cosine transforms for compression).

The cosine transforms discussed above are named DCT-II and DCT-III.

Various cosine transforms arise by imposing various boundary conditions:

- DCT-II: even around $-1/2$ and $N - 1/2$
- DCT-III: even around 0 and odd around N

DCT-II is used in JPEG-compression while a slightly modified DCT-IV makes the main component of MP3, AAC and WMA formats.



R. Hiptmair

rev 30401, January 28, 2011

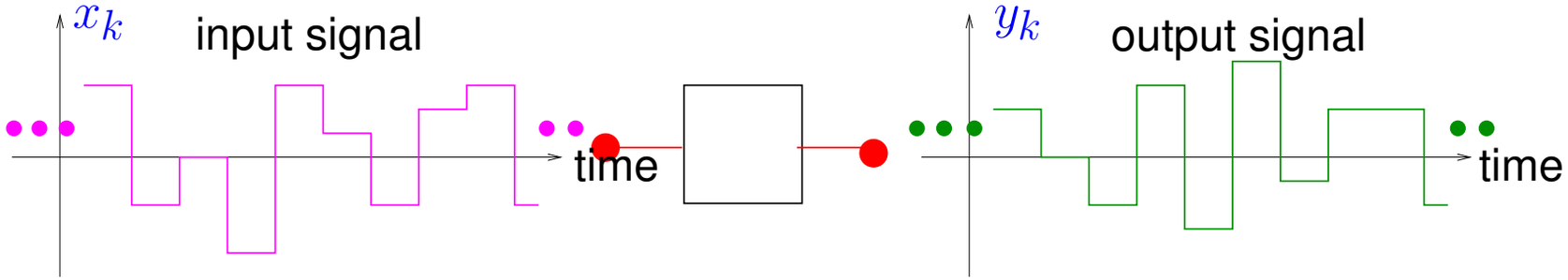
8.5 Toeplitz matrix techniques

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 *known* output signal of a linear time-invariant filter, see Ex. 8.1.1.
- Known: impulse response of filter has maximal duration $n\Delta t$, $n \in \mathbb{N}$, $n \leq m$

cf. (8.1.2)

$$\exists \mathbf{h} = (h_0, \ldots, h_{n-1})^T \in \mathbb{R}^n, \quad n \leq m : \quad y_k = \sum_{j=0}^{n-1} h_j x_{k-j} \; . \tag{8.5.2}$$



Parameter identification problem: seek $\mathbf{h} = (h_0, \ldots, h_{n-1})^T \in \mathbb{R}^n$ with

$$\|\mathbf{Ah} - \mathbf{y}\|_2 = \left\| \begin{pmatrix} x_0 & x_{-1} & \cdots & & \cdots & x_{1-n} \\ x_1 & x_0 & x_{-1} & & & \vdots \\ \vdots & x_1 & x_0 & \ddots & & \\ & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & x_{-1} \\ x_{n-1} & & & & x_1 & x_0 \\ x_n & x_{n-1} & & & & x_1 \\ \vdots & & & & & \vdots \\ x_{m-1} & \cdots & & & \cdots & x_{m-n} \end{pmatrix} \begin{pmatrix} h_0 \\ \vdots \\ \\ \\ \vdots \\ h_{n-1} \end{pmatrix} - \begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \vdots \\ y_{m-1} \end{pmatrix} \right\|_2 \to \min \ .$$

➢ Linear least squares problem, → Ch. 7 with Toeplitz matrix $\mathbf{A}$: $(\mathbf{A})_{ij} = x_{i-j}$.

System matrix of normal equations (→ Sect. 7.1)

$$\mathbf{M} := \mathbf{A}^H \mathbf{A} \quad , \quad (\mathbf{M})_{ij} = \sum_{k=1}^{m} x_{k-i} x_{k-j} = z_{i-j} \quad \text{due to periodicity of } (x_k)_{k \in \mathbb{Z}} \ .$$

➢ $\mathbf{M} \in \mathbb{R}^{n,n}$ is a *matrix with constant diagonals* & s.p.d.

("constant diagonals" $\Leftrightarrow$ $(\mathbf{M})_{i,j}$ depends only on $i - j$)

R. Hiptmair

rev 30401, January 28, 2011

Example 8.5.3 (Linear regression for stationary Markov chains).

Sequence of scalar random variables: $(\mathsf{Y}_k)_{k\in\mathbb{Z}}$ = Markov chain

Assume: stationary (time-independent) correlation

$$\text{Expectation} \qquad \mathcal{E}(\mathsf{Y}_{i-j}\mathsf{Y}_{i-k}) = u_{k-j} \quad \forall i,j,k \in \mathbb{Z}\,, \quad u_i = u_{-i}\,.$$

Model: finite linear relationship

$$\exists \mathbf{x} = (x_1,\ldots,x_n)^T \in \mathbb{R}^n\colon \quad \mathsf{Y}_k = \sum_{j=1}^{n} x_j \mathsf{Y}_{k-j} \quad \forall k \in \mathbb{Z}\,.$$

with *unknown* parameters x_j, $j = 1,\ldots,n$: for fixed $i \in \mathbb{Z}$

$$\text{Estimator} \qquad \mathbf{x} = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} E\Big|\mathsf{Y}_i - \sum_{j=1}^{n} x_j \mathsf{Y}_{i-j}\Big|^2$$

$$\blacktriangleright \qquad E|\mathsf{Y}_i|^2 - 2\sum_{j=1}^{n} x_j u_k + \sum_{k,j=1}^{n} x_k x_j u_{k-j} \to \min\,.$$

$$\blacktriangleright \qquad \mathbf{x}^T\mathbf{A}\mathbf{x} - 2\mathbf{b}^T\mathbf{x} \to \min \quad \text{with} \quad \mathbf{b} = (u_k)_{k=1}^n\,, \quad \mathbf{A} = (u_{i-j})_{i,j=1}^n\,.$$

Lemma 5.1.3 $\Rightarrow$ $\mathbf{x}$ solves $\mathbf{Ax} = \mathbf{b}$ (Yule-Walker-equation, see below)

$\mathbf{A} \;\hat{=}\;$ Covariance matrix: s.p.d. matrix with constant diagonals. $\diamond$

Matrices with constant diagonals occur frequently in mathematical models. They generalize circulant matrices $\rightarrow$ Def. 8.1.16.

Note: "Information content" of a matrix $\mathbf{M} \in \mathbb{K}^{m,n}$ with constant diagonals, that is, $(\mathbf{M})_{i,j} = m_{i-j}$, is $m + n - 1$ numbers $\in \mathbb{K}$.

Definition 8.5.4 (Toeplitz matrix).
$\mathbf{T} = (t_{ij})_{i,j=1}^{n} \in \mathbb{K}^{m,n}$ *is a Toeplitz matrix, if there is a vector* $\mathbf{u} = (u_{-m+1}, \ldots, u_{n-1}) \in \mathbb{K}^{m+n-1}$ *such that* $t_{ij} = u_{j-i}$, $1 \le i \le m$, $1 \le j \le n$.

$$\mathbf{T} = \begin{pmatrix} u_0 & u_1 & \cdots & & \cdots & u_{n-1} \\ u_{-1} & u_0 & u_1 & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & u_1 \\ u_{1-m} & \cdots & & \cdots & u_{-1} & u_0 \end{pmatrix}$$

R. Hiptmair

rev 30401, January 28, 2011

8.5.1 Toeplitz matrix arithmetic

$\mathbf{T} = (u_{j-i}) \in \mathbb{K}^{m,n}$ = Toeplitz matrix with generating vector $\mathbf{u} = (u_{-m+1}, \ldots, u_{n-1}) \in \mathbb{C}^{m+n-1}$

Task: efficient evaluation of matrix×vector product $\mathbf{Tx}$, $\mathbf{x} \in \mathbb{K}^n$

Note: this extended matrix is circulant (→ Def. 8.1.16)

$$\mathbf{C} = \begin{pmatrix} \mathbf{T} & \mathbf{S} \\ \mathbf{S} & \mathbf{T} \end{pmatrix} = \left(\begin{array}{ccccccc|ccccccc} u_0 & u_1 & \cdots & & \cdots & u_{n-1} & & 0 & u_{1-n} & \cdots & & \cdots & u_{-1} & \\ u_{-1} & u_0 & u_1 & & & \vdots & & u_{n-1} & 0 & \ddots & & & \vdots & \\ \vdots & \ddots & \ddots & \ddots & & \vdots & & \vdots & \ddots & \ddots & & & & \\ \vdots & & \ddots & \ddots & \ddots & \vdots & & & & & & \ddots & & \\ \vdots & & & \ddots & \ddots & u_1 & & \vdots & & & \ddots & \ddots & u_{1-n} & \\ u_{1-n} & \cdots & & \cdots & u_{-1} & u_0 & & u_1 & & & & u_{n-1} & 0 & \\ \hline 0 & u_{1-n} & \cdots & & \cdots & u_{-1} & & u_0 & u_1 & \cdots & & \cdots & u_{n-1} & \\ u_{n-1} & 0 & \ddots & & & \vdots & & u_{-1} & u_0 & u_1 & & & \vdots & \\ \vdots & \ddots & \ddots & & & & & \vdots & \ddots & \ddots & \ddots & & \vdots & \\ & & & & \ddots & & & \vdots & & \ddots & \ddots & \ddots & \vdots & \\ \vdots & & & \ddots & \ddots & u_{1-n} & & \vdots & & & \ddots & \ddots & u_1 & \\ u_1 & & & & u_{n-1} & 0 & & u_{1-n} & \cdots & & \cdots & u_{-1} & u_0 & \end{array}\right)$$

This example demonstrates the case $m = n$

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)]);
```

$$\mathbf{C}\begin{pmatrix}\mathbf{x}\\0\end{pmatrix} = \begin{pmatrix}\mathbf{Tx}\\\mathbf{Sx}\end{pmatrix}$$

Computational effort $O(n \log n)$ for computing $\mathbf{Tx}$ (FFT based, Sect. 8.3)

8.5.2 The Levinson algorithm

Given: S.p.d. Toeplitz matrix $\mathbf{T} = (u_{j-i})_{i,j=1}^{n}$, generating vector $\mathbf{u} = (u_{-n+1}, \ldots, u_{n-1}) \in \mathbb{C}^{2n-1}$
(Symmetry $\leftrightarrow u_{-k} = u_k$, w.l.o.g $u_0 = 1$)

Task: efficient solution algorithm for LSE $\mathbf{Tx} = \mathbf{b}$, $\mathbf{b} \in \mathbb{C}^n$
(Yule-Walker problem)

Recursive (inductive) solution strategy:

Define:
- $\mathbf{T}_k := (u_{j-i})_{i,j=1}^k \in \mathbb{K}^{k,k}$ (left upper block of $\mathbf{T}$) ➢ $\mathbf{T}_k$ is s.p.d. Toeplitz matrix,
- $\mathbf{x}^k \in \mathbb{K}^k$: $\mathbf{T}_k\mathbf{x}^k = (b_1,\ldots,b_k)^T \Leftrightarrow \mathbf{x}^k = \mathbf{T}_k^{-1}\mathbf{b}^k$,
- $\mathbf{u}^k := (u_1,\ldots,u_k)^T$

Block-partitioned LSE, *cf.* Rem. 2.1.10, Rem. 2.2.14

$$\mathbf{T}_{k+1}\mathbf{x}^{k+1} = \left(\begin{array}{c|c} \mathbf{T}_k & \begin{matrix} u_k \\ \vdots \\ u_1 \end{matrix} \\ \hline \begin{matrix} u_k & \cdots & u_1 \end{matrix} & 1 \end{array}\right) \left(\begin{array}{c} \widetilde{\mathbf{x}}^{k+1} \\ \hline x_{k+1}^{k+1} \end{array}\right) = \left(\begin{array}{c} \begin{matrix} b_1 \\ \vdots \\ b_k \end{matrix} \\ \hline b_{k+1} \end{array}\right) = \begin{pmatrix} \widetilde{\mathbf{b}}^{k+1} \\ b_{k+1} \end{pmatrix} \tag{8.5.5}$$

Reversing permutation: $P_k : \{1,\ldots,k\} \mapsto \{1,\ldots,k\}$, $P_k(i) := k-i+1$

$$\begin{aligned} \widetilde{\mathbf{x}}_{k+1} &= \mathbf{T}_k^{-1}(\widetilde{\mathbf{b}}^{k+1} - x_{k+1}^{k+1}P_k\mathbf{u}^k) = \mathbf{x}^k - x_{k+1}^{k+1}\mathbf{T}_k^{-1}P_k\mathbf{u}^k\,, \\ x_{k+1}^{k+1} &= b_{k+1} - P_k\mathbf{u}^k\cdot\widetilde{\mathbf{x}}^{k+1} = b_{k+1} - P_k\cdot\mathbf{x}^k + x_{k+1}^{k+1}P_k\cdot\mathbf{T}_k^{-1}P_k\mathbf{u}^k\,. \end{aligned} \tag{8.5.6}$$

Efficient algorithm by using *auxiliary vectors*: $\mathbf{y}^k := \mathbf{T}_k^{-1}P_k\mathbf{u}^k$

$$\mathbf{x}^{k+1} = \begin{pmatrix} \widetilde{\mathbf{x}}^{k+1} \\ x_{k+1}^{k+1} \end{pmatrix} \quad \text{with} \quad \begin{aligned} x_{k+1}^{k+1} &= (b_{k+1} - P_k\mathbf{u}^k)/\sigma_k \\ \widetilde{\mathbf{x}}^{k+1} &= \mathbf{x}^k - x_{k+1}^{k+1}\mathbf{y}^k \end{aligned}\,, \quad \sigma_k := 1 - P_k\mathbf{u}^k\cdot\mathbf{y}^k\,. \tag{8.5.7}$$

R. Hiptmair

rev 30401, January 28, 2011

Levinson algorithm ▷

(recursive, u_{n+1} not used!)

Linear recursion:

Computational cost $\sim (n-k)$ on level k, $k = 0, \ldots, n-1$

➢ Asymptotic complexity $O(n^2)$

MATLAB-CODE 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];
```

Remark 8.5.8 (Fast Toeplitz solvers).

FFT-based algorithms for solving $\mathbf{Tx} = \mathbf{b}$ with asymptotic complexity $O(n \log^3 n)$ [49] **!** △

Supplementary and further reading:

R. Hiptmair

rev 30401, January 28, 2011

[10, Sect. 8.5]: Very detailed and elementary presentation, but the discrete Fourier transform through trigonometric interpolation, which is not covered in this chapter. Hardly addresses discrete convolution.

[29, Ch. IX] presents the topic from a mathematical point of few stressing approximation and trigonometric interpolation. Good reference for algorithms for circulant and Toeplitz matrices.

[47, Ch. 10] also discusses the discrete Fourier transform with emphasis on interpolation and (least squares) approximation. The presentation of signal processing differs from that of the course.

There is a vast number of books and survey papers dedicated to discrete Fourier transforms, see, for instance, [15, 6]. Issues and technical details way beyond the scope of the course are treated there.

9 Approximation of Functions in 1D

Problem of **function approximation**:

Given: function $\mathbf{f} : D \subset \mathbb{R}^n \mapsto \mathbb{R}^d$ (often in procedural form `y=feval(x)`)

Goal: Find a "simple"$^{(*)}$ function $\widetilde{\mathbf{f}} : D \mapsto \mathbb{R}^d$ such that the difference $\mathbf{f} - \widetilde{\mathbf{f}}$ is "small"$^{(\clubsuit)}$

$^{(*)}$: "simple" $\sim$ described by small amount of information, easy to evaluate (e.g, polynomial or piecewise polynomial $\widetilde{\mathbf{f}}$)

$^{(\clubsuit)}$ "small" $\sim$ $\left\|\mathbf{f} - \widetilde{\mathbf{f}}\right\|$ small for some norm $\|\cdot\|$ on space $C(D)$ of continous functions, e.g.,

- L^2-norm $\|\mathbf{g}\|_2^2 := \|\mathbf{g}\|_{L^2(D)}^2 = \int_D |\mathbf{g}(x)|^2 dx$, see (3.3.15),
- supremum norm $\|\mathbf{g}\|_\infty := \|\mathbf{g}\|_{L^\infty(D)} := \max_{x \in D} |\mathbf{g}(x)|$, see (3.3.14)

Below we only consider the case $n = d = 1$: approximation of scalar valued functions defined on an interval. The techniques are applied componentwise in order to cope with $d > 1$.

R. Hiptmair

rev 30401, November 29, 2010

Example 9.0.1 (Taylor approximation). $\rightarrow$ [52, Sect. 5.5]

For $f \in C^k(I)$, $k \in \mathbb{N}$, $I \subset \mathbb{R}$ an interval,

$$f(t) \approx \underbrace{\sum_{j=0}^{k} \frac{f^{(j)}(t_0)}{j!}(t-t_0)^j}_{=:T_k(t)} \quad , \quad t_0 \in I \ .$$

The Taylor polynomial T_k of degree k approximates f in a neighbourhood $J \subset I$ of t_0 (J can be small!). The Taylor approximation is easy and direct but inefficient: a polynomial of lower degree often gives the same accuracy.

$\diamond$

Another technique: Approximation by interpolation

$$\mathbf{f} \xrightarrow{\text{sampling}} (t_i, \mathbf{y}_i := f(t_i))_{i=1}^m \xrightarrow{\text{interpolation}} \widetilde{\mathbf{f}}: \quad \widetilde{\mathbf{f}}(t_i) = \mathbf{y}_i \ .$$

9

$\widetilde{\mathbf{f}}$

Remark 9.0.2 (Interpolation and approximation: enabling technologies).

Approximation and interpolation ($\rightarrow$ Ch. 9) are key components of many numerical methods, like for integration, differentiation and computation of the solutions of differential equations, as well as for computer graphics and generation of smooth curves and surfaces.

this is a “foundations” part of the course

△

9.1 Error estimates for polynomial interpolation

Fcous: approximation of a function by global polynomial interpolation ($\rightarrow$ Sect. 3.3)

Remark 9.1.1 (Approximation by polynomials).

? Is it always possible to approximate a continuous function by polynomials?

✓ Yes! Recall the Weierstrass theorem:
A continuous function f on the interval $[a,b] \subset \mathbb{R}$ can be uniformly approximated by polynomials.

! But not by the interpolation on a fixed mesh [42, pag. 331]:
Given a sequence of meshes of increasing size $\{\mathcal{T}_j\}_{j=1}^{\infty}$, $\mathcal{T}_j = \{x_1^{(j)}, \ldots, x_j^{(j)}\} \subset [a,b]$, $a \leq x_1^{(j)} < x_2^{(j)} < \cdots < x_j^{(j)} \leq b$, there exists a continuous function f such that the sequence interpolating polynomials of f on $\mathcal{T}_j$ does not converge uniformly to f as $j \to \infty$.

R. Hiptmair

rev 30401, November 29, 2010

We consider Lagrangian polynomial interpolation on node set

$$\mathcal{T} := \{t_0, \ldots, t_n\} \subset I,\ I \subset \mathbb{R}, \text{ interval of length } |I|.$$

Notation: For a continuous function $f : I \mapsto \mathbb{K}$ we define the polynomial interpolation operator, see Thm. 3.3.8

$$\mathsf{I}_{\mathcal{T}}(f) := \mathsf{I}_{\mathcal{T}}(\mathbf{y}) \in \mathcal{P}_n \quad \text{with} \quad \mathbf{y} := (f(t_0), \ldots, f(t_n))^T \in \mathbb{K}^{n+1}.$$

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 $p := \mathrm{I}_{\mathcal{T}} f \in \mathcal{P}_n$.

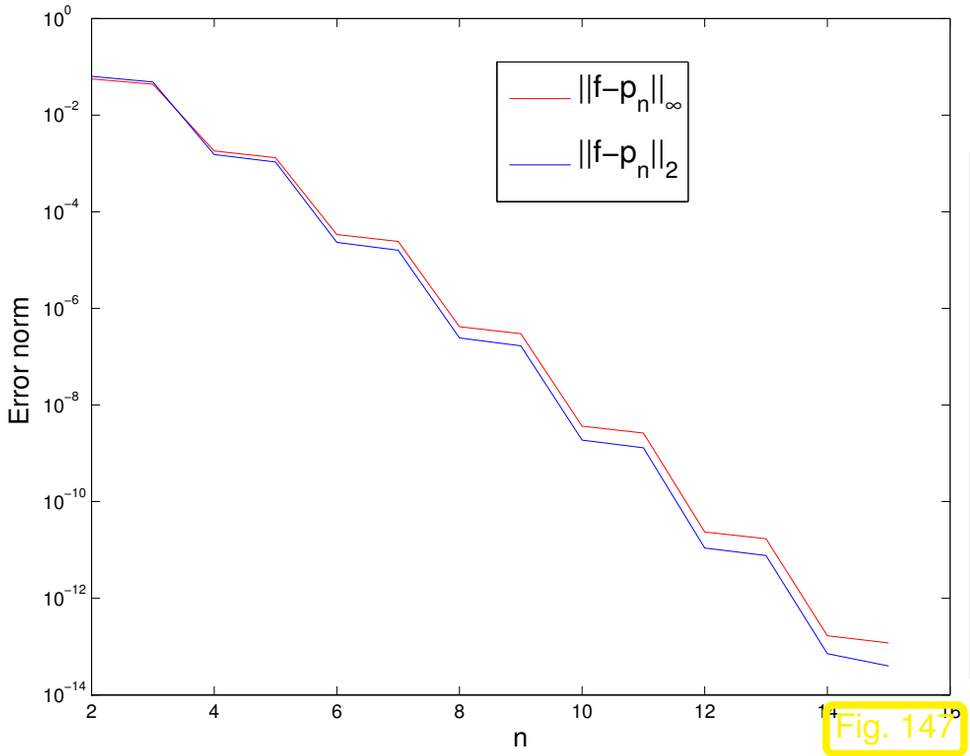


Fig. 147

MATLAB-experiment: computation of the norms.

- L^∞-norm: sampling on a grid of meshsize $\pi/1000$.
- L^2-norm: numerical quadrature ($\rightarrow$ Chapter 10) with trapezoidal rule (10.3.3) on a grid of meshsize $\pi/1000$.

$\diamond$

R. Hiptmair

rev 30401, November 29, 2010

In the previous experiment we observed a clearly visible qualitative behavior of $\|f - \mathrm{I}_{\mathcal{T}} f\|$ as we increased the polynomial degree n. The prediction of the decay law for $\|f - \mathrm{I}_{\mathcal{T}} f\|$ is one goal in the study of interpolation errors.

Often this goal can be achieved, even if a rigorous quantitative bound for a norm of the interpolation error remains elusive.

Important terminology for the qualitative description of $\|f - \mathrm{I}_{\mathcal{T}} f\|$ as a function of the polynomial degree n:

$$\exists\, C \neq C(n) > 0\text{:} \quad \|f - \mathrm{I}_{\mathcal{T}} f\| \leq C\, T(n) \quad \text{for } n \to \infty\ . \tag{9.1.3}$$

Classification (best bound for $T(n)$):

$\exists\, p > 0$: $T(n) \leq n^{-p}$: algebraic convergence, with rate $p > 0$,

$\exists\, 0 < q < 1$: $T(n) \leq q^n$: exponential convergence ,

$\forall n \in \mathbb{N}$.

Convergence behavior of interpolation error is often expressed by means of the Landau-O-notation:

Algebraic convergence: $\|f - \mathrm{I}_{\mathcal{T}} f\| = O(n^{-p})$
Exponential convergence: $\|f - \mathrm{I}_{\mathcal{T}} f\| = O(q^{n})$

for $n \to \infty$ (“asymptotic!”)

Remark 9.1.4 (Exploring convergence).

Given: pairs (n_i, ϵ_i), $i = 1, 2, 3, \ldots$, $n_i \mathrel{\hat{=}}$ polynomial degrees, $\epsilon_i \mathrel{\hat{=}}$ norms of interpolation error

❶ Conjectured: algebraic convergence: $\epsilon_i \approx C n^{-p}$

$$\log(\epsilon_i) \approx \log(C) - p \log n_i$$ (affine linear in log-log scale).

Apply linear regression (MATLAB `polyfit`) to points $(\log n_i, \log \epsilon_i)$ ➢ estimate for rate p.

❶ Conjectured: exponential convergence: $\epsilon_i \approx C \exp(-\beta n_i)$

$$\log \epsilon_i \approx \log(C) - \beta n_i$$ (affine linear in lin-log scale). .

Apply linear regression (Ex. 7.0.9, MATLAB `polyfit`) to points $(n_i, \log \epsilon_i)$ ➢ estimate for $q := \exp(-\beta)$.

147 exponential convergence 9.1.2

Beware: same concept $\leftrightarrow$ different meanings:

- convergence of a sequence (e.g. of iterates $\boldsymbol{x}^{(k)}$ $\rightarrow$ Sect. 4.1)
- convergence of an approximation (dependent on an approximation parameter, e.g. n)

Example 9.1.5 (Runge's example). $\rightarrow$ Ex. 3.3.21

Polynomial interpolation of $f(t) = \frac{1}{1+t^2}$ with equispaced nodes:

$$\mathcal{T} := \left\{ t_j := -5 + \frac{10}{n} j \right\}_{j=0}^{n} , \quad y_j = \frac{1}{1+t_j^2} \, . j = 0, \ldots, n \, .$$

R. Hiptmair

rev 30401, November 29, 2010

Code 9.1.6: Computing the interpolation error for Runge's example

```
% interpolation error plot for Runge's example
% ("Quick and dirty" MATLAB implementation, see 3.4)
f = @(x) (1./(1+x.^2));
x = -5:0.01:5;   % sampling point for approximate maximum norm
fv = f(x);
err = [];
for d=1:20
  t = -5+(0:d)*10/d;
  p = polyfit(t,f(t),d);
  y = polyval(p,x);
  err = [err; d, max(abs(y-fv))];
end

figure; semilogy(err(:,1),err(:,2),'r--+');
xlabel('{\bf degree d}','fontsize',14);
ylabel('{\bf interpolation error (maximum norm)}','fontsize',14);
print -depsc2 '../PICTURES/rungeerrmax.eps';
```


R. Hiptmair

rev 30401, November 29, 2010

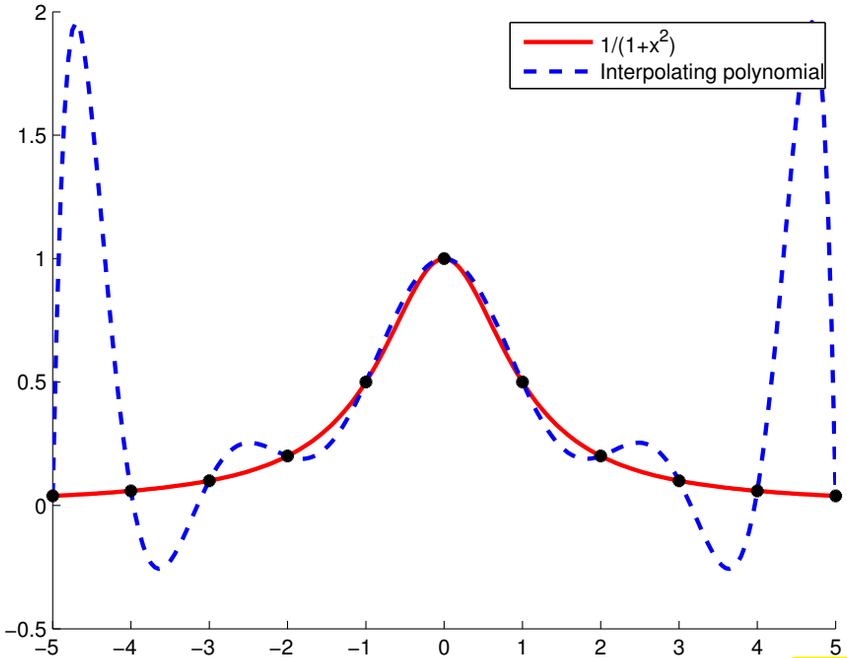


Fig. 148

Interpolating polynomial, $n = 10$

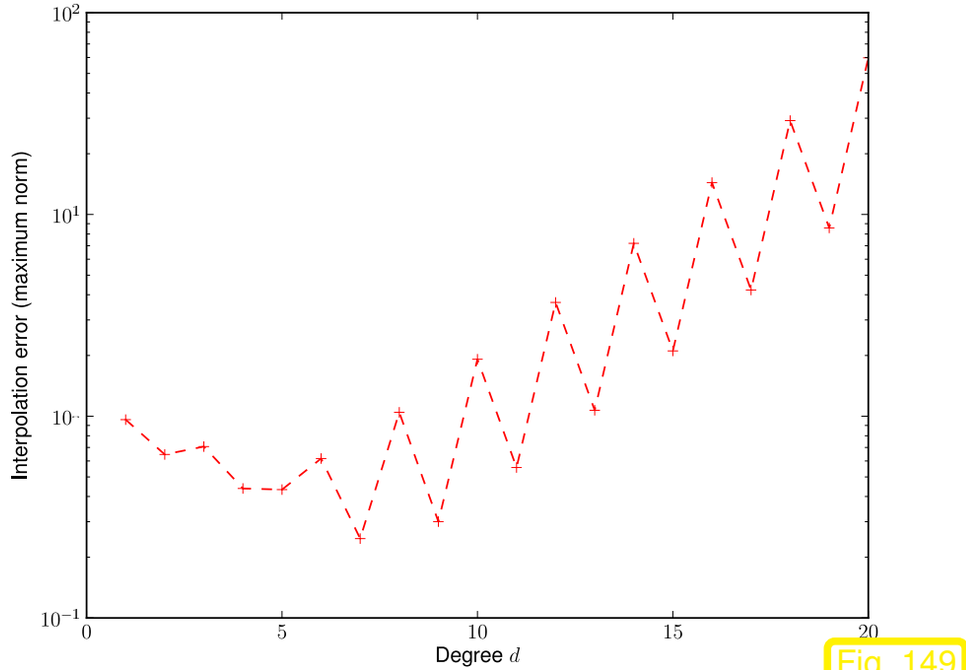


Fig. 149

Approximate $\|f - \mathcal{I}_{\mathcal{T}} f\|_\infty$ on $[-5, 5]$

R. Hiptmair

rev 30401, November 29, 2010

Note: approximation of $\|f - \mathcal{I}_{\mathcal{T}} f\|_\infty$ by *sampling* in 1000 equidistant points.

Observation: Strong oscillations of $\mathsf{I}_{\mathcal{T}} f$ near the endpoints of the interval:

$$\|f - \mathsf{I}_{\mathcal{T}} f\|_{L^\infty(]-5,5[)} \xrightarrow{n\to\infty} \infty \ .$$

Theorem 9.1.7 (Representation of interpolation error). *[10, Thm. 8.22], [29, Thm. 37.4]*

$f \in C^{n+1}(I)$: $\forall t \in I$: $\exists\, \tau_t \in]\min\{t, t_0, \ldots, t_n\}, \max\{t, t_0, \ldots, t_n\}[$:

$$f(t) - \mathsf{I}_{\mathcal{T}}(f)(t) = \frac{f^{(n+1)}(\tau_t)}{(n+1)!} \cdot \prod_{j=0}^{n} (t - t_j) \,. \tag{9.1.8}$$

Proof. Write $q(t) := \prod_{j=0}^{n} (t - t_j) \in \mathcal{P}_{n+1}$ and fix $t \in I$.

$$t \neq t_j \quad \Rightarrow \quad q(t) \neq 0 \quad \Rightarrow \quad \exists c(t) \in \mathbb{R}\colon \quad f(t) - \mathsf{I}_{\mathcal{T}}(f)(t) = cq(t)$$

$\varphi(x) := f(x) - \mathsf{I}_{\mathcal{T}}(f)(x) - cq(x)$ has $n+2$ *distinct* zeros $t_0, \ldots, t_n, t$. By iterated application of the mean value theorem [52, Thm .5.2.1], we conclude

$$\varphi^{(m)} \text{ has } n + 2 - m \text{ distinct zeros in } I.$$

$$\Rightarrow \quad \exists \tau_t \in I\colon \quad \varphi^{(n+1)}(x) = f^{(n+1)}(\tau_t) - c(n+1)! = 0 \ .$$

This fixes the value of $c = \frac{f^{(n+1)}(\tau_t)}{(n+1)!}$. □

The theorem can also be proved using the following lemma.

Lemma 9.1.9 (Error of the polynomial interpolation)**.** *For $f \in C^{n+1}(I)$: $\forall\, t \in I$:*

$$f(t) - \mathsf{I}_{\mathcal{T}}(f)(t) = \int_0^1 \int_0^{\tau_1} \cdots \int_0^{\tau_{n-1}} \int_0^{\tau_n} f^{(n+1)}(t_0 + \tau_1(t_1 - t_0) + \cdots + \tau_n(t_n - t_{n-1}) + \tau(t - t_n))\, \mathrm{d}\tau \mathrm{d}\tau_n \cdots \mathrm{d}\tau_1 \cdot \prod_{j=0}^{n} (t - t_j) \ .$$

Proof. By induction on n, use (3.4.3) and the fundamental theorem of calculus [44, Sect. 3.1]:

Remark 9.1.10. Lemma 9.1.9 holds also for general polynomial interpolation with multiple nodes, see (3.3.2).

△

In the *equation* (9.1.8) we can

- first bound the right hand side via $f^{(n+1)}(\tau_t) \leq \left\| f^{(n+1)} \right\|_{L^\infty(I)}$,
- then increase the right hand side further by switching to the maximum (in modulus) w.r.t. t (the resulting bound does no longer depend on t!),
- and, finally, take the maximum w.r.t. t on the left of $\leq$.

This yields the following interpolation error estimate:

Thm. 9.1.7 $\Rightarrow$

$$\|f - \mathsf{I}_{\mathcal{T}} f\|_{L^\infty(I)} \leq \frac{\left\| f^{(n+1)} \right\|_{L^\infty(I)}}{(n+1)!} \max_{t \in I} |(t - t_0) \cdot \cdots \cdot (t - t_n)| \ . \quad (9.1.11)$$

Interpolation error estimate requires smoothness!

Example 9.1.12 (Error of polynomial interpolation). Ex. 9.1.2 cnt'd

Theoretical explanation for exponential convergence observed for polynomial interpolation of $f(t) = \sin(t)$ on equidistant nodes: by Thm. 9.1.9 and (9.1.11)

$$\left\| f^{(k)} \right\|_{L^\infty(I)} \leq 1 \;, \quad \forall k \in \mathbb{N}_0 \quad \Rightarrow \quad \begin{aligned} \|f - p\|_{L^\infty(I)} &\leq \frac{1}{(1+n)!} \max_{t \in I} \left| (t-0)(t-\tfrac{\pi}{n})(t-\tfrac{2\pi}{n}) \cdot \cdots \cdot (t-\pi) \right| \\ &\leq \frac{1}{n+1} \left(\frac{\pi}{n} \right)^{n+1} . \end{aligned}$$

➙ Uniform asymptotic exponential convergence of the interpolation polynomials (independently of the set of nodes $\mathcal{T}$. In fact, $\|f - p\|_{L^\infty(I)}$ decays even faster than exponential!)

◇

Example 9.1.13 (Runge's example). Ex. 9.1.5 cnt'd

How can the blow-up of the interpolation error observed in Ex. 9.1.5 be reconciled with Thm. 9.1.9 ?

Here $f(t) = \frac{1}{1+t^2}$ allows only to conclude $|f^{(n)}(t)| = 2^n n! \cdot O(|t|^{-2-n})$ for $n \to \infty$.

➙ Possible blow-up of error bound from Thm. 9.1.7 $\to \infty$ for $n \to \infty$.

Remark 9.1.14 (L^2-error estimates for polynomial interpolation).

Thm. 9.1.7 gives error estimates for the L^∞-Norm. And the other norms?

From Lemma. 9.1.9 using Cauchy-Schwarz inequality:

$$\|f - \mathsf{I}_{\mathcal{T}}(f)\|^2_{L^2(I)} = \int_I \left| \int_0^1 \int_0^{\tau_1} \cdots \int_0^{\tau_{n-1}} \int_0^{\tau_n} f^{(n+1)}(\ldots)\, \mathrm{d}\tau \mathrm{d}\tau_n \cdots \mathrm{d}\tau_1 \cdot \underbrace{\prod_{j=0}^{n}(t-t_j)}_{|t-t_j|\le|I|} \right|^2 \mathrm{d}t$$

$$\le \int_I |I|^{2n+2} \underbrace{\mathrm{vol}_{(n+1)}(S_{n+1})}_{=1/(n+1)!} \int_{S_{n+1}} |f^{(n+1)}(\ldots)|^2 \, \mathrm{d}\boldsymbol{\tau}\, \mathrm{d}t$$

$$= \int_I \frac{|I|^{2n+2}}{(n+1)!} \int_I \underbrace{\mathrm{vol}_{(n)}(C_{t,\tau})}_{\leq 2^{(n-1)/2}/n!} |f^{(n+1)}(\tau)|^2 \,\mathrm{d}\tau\mathrm{d}t \;,$$

$$S_{n+1} := \{\mathbf{x} \in \mathbb{R}^{n+1} \colon 0 \leq x_n \leq x_{n-1} \leq \cdots \leq x_1 \leq 1\} \quad \text{(unit simplex)} \;,$$
$$C_{t,\tau} := \{\mathbf{x} \in S_{n+1} \colon t_0 + x_1(t_1 - t_0) + \cdots + x_n(t_n - t_{n-1}) + x_{n+1}(t - t_n) = \tau\} \;.$$

This gives the bound for the L^2-norm of the error:

$$\Rightarrow \qquad \|f - \mathsf{I}_{\mathcal{T}}(f)\|_{L^2(I)} \leq \frac{2^{(n-1)/4}|I|^{n+1}}{\sqrt{(n+1)!n!}} \left(\int_I |f^{(n+1)}(\tau)|^2 \,\mathrm{d}\tau \right)^{1/2} . \tag{9.1.15}$$

Notice: $f \mapsto \left\| f^{(n)} \right\|_{L^2(I)}$ defines a seminorm on $C^{n+1}(I)$
(Sobolev-seminorm, measure of the smoothness of a function).

Estimates like (9.1.15) play a key role in the analysis of numerical methods for solving partial differential equations ($\rightarrow$ course “Numerical methods for partial differential equations”).

R. Hiptmair

rev 30401, November 29, 2010

9.2 Chebychev Interpolation

Perspective: function approximation by polynomial interpolation

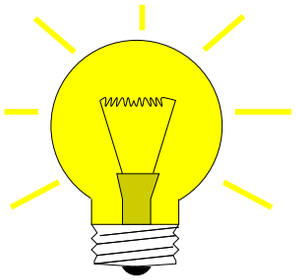
➢ Freedom to choose interpolation nodes judiciously

9.2.1 Motivation and definition

Setting: Mesh of nodes: $\mathcal{T} := \{t_0 < t_1 < \cdots < t_{n-1} < t_n\}$, $n \in \mathbb{N}$,
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 $t_0, \ldots, t_n$ 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.

$\blacktriangleright$ Choice of $t_0, \ldots, t_n$ = zeros of q (caution: t_j must belong to I).

Heuristic:
- t^* extremal point of q $\rightarrow$ $|q(t^*)| = \|q\|_{L^\infty(I)}$,
- q has $n+1$ zeros in I,
- $|q(-1)| = |q(1)| = \|q\|_{L^\infty(I)}$.

Definition 9.2.1 (Chebychev polynomial). $\rightarrow$ *[29, Ch. 32]*

The n^{th} *Chebychev polynomial is* $T_n(t) := \cos(n \arccos t)$, $-1 \leq t \leq 1$.

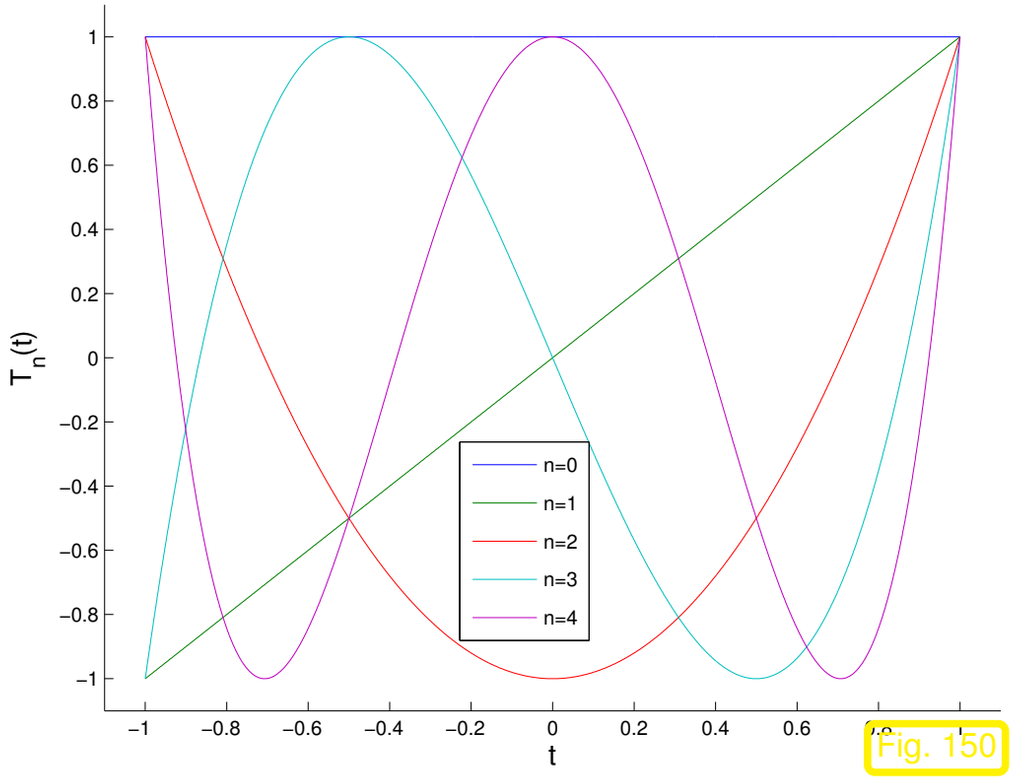


Fig. 150

Chebychev polynomials $T_0, \ldots, T_4$

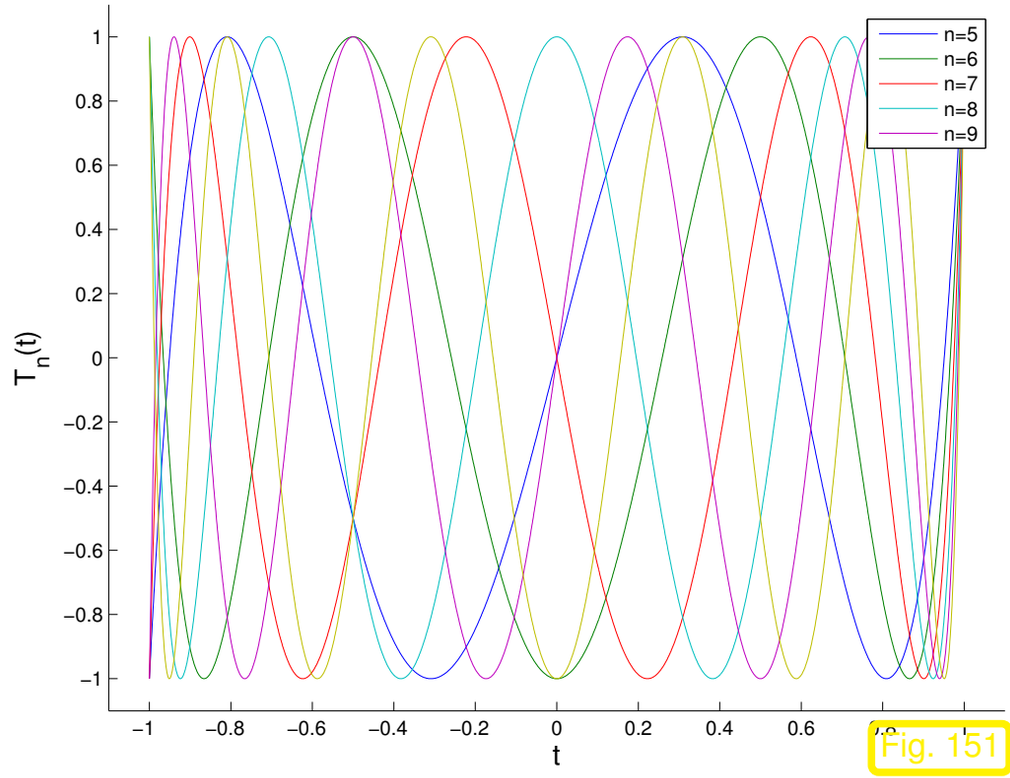


Fig. 151

Chebychev polynomials $T_5, \ldots, T_9$

R. Hiptmair

rev 30401, November 29, 2010

Code 9.2.2: Efficient evaluation of Chebychev polynomials up to a certain degree

```
function V = chebpolmult(d,x)
% Computes the values of the Chebychev polynomials T_0,...,T_d, d ≥ 2
% at points passed in x using the 3-term recursion (9.2.7).
% The values T_k(x_j), are returned in row k+1 of V.
V = ones(1,length(x)); % T_0 ≡ 1
V = [V; x];            % T_1(x) = x
```

```
for k=1:d-1
  V = [V; 2*x.*V(end,:)-V(end-1,:)]; % 3-term recursion
end
```

Code 9.2.3: Plotting Chebychev polynomials, see Fig. 150, 151

```
function chebpolplot(nmax)
% plots Chebychev polybomials up to degree nmax on [-1,1]
N = 1000; x = -1:2/N:1;
V = chebpolmult(nmax,x); % compute values of Chebychev polynomials
for j=0:nmax, leg{j} = sprintf('n=%i',j); end
plot(x,V','-');
legend(leg);
xlabel('{\bf t}','Fontsize',14);
ylabel('{\bf T_n(t)}','Fontsize',14);
axis([-1.1 1.1 -1.1 1.1]);
```

Zeros of T_n:
$$t_k = \cos\left(\frac{2k-1}{2n}\pi\right) \;, \quad k = 1,\ldots,n \;. \tag{9.2.4}$$

R. Hiptmair

rev 30401, November 29, 2010

Too see this, notice

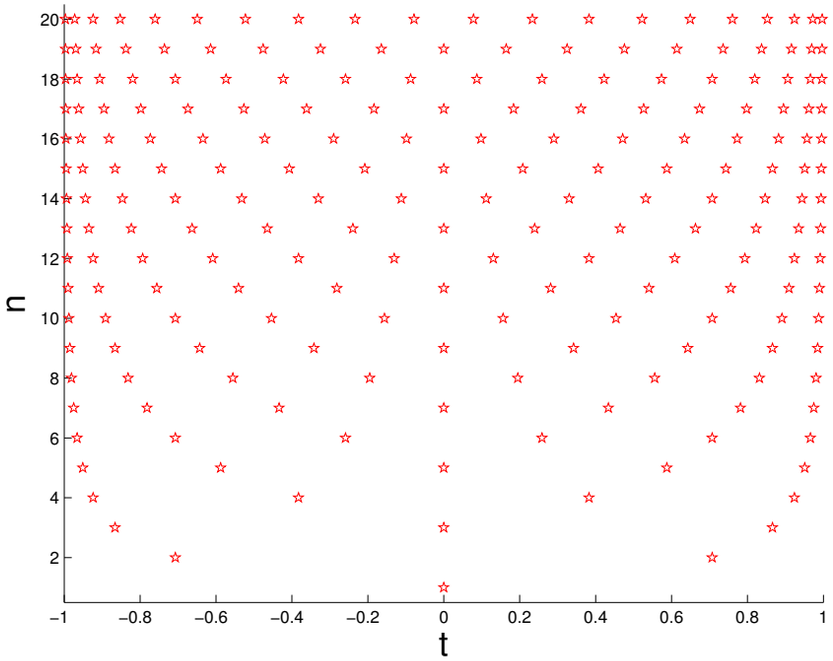
$$T_n(t) = 0 \quad \overset{\text{zeros of } \cos}{\Leftrightarrow} \quad n \arccos t \in (2\mathbb{Z}+1)\frac{\pi}{2}$$

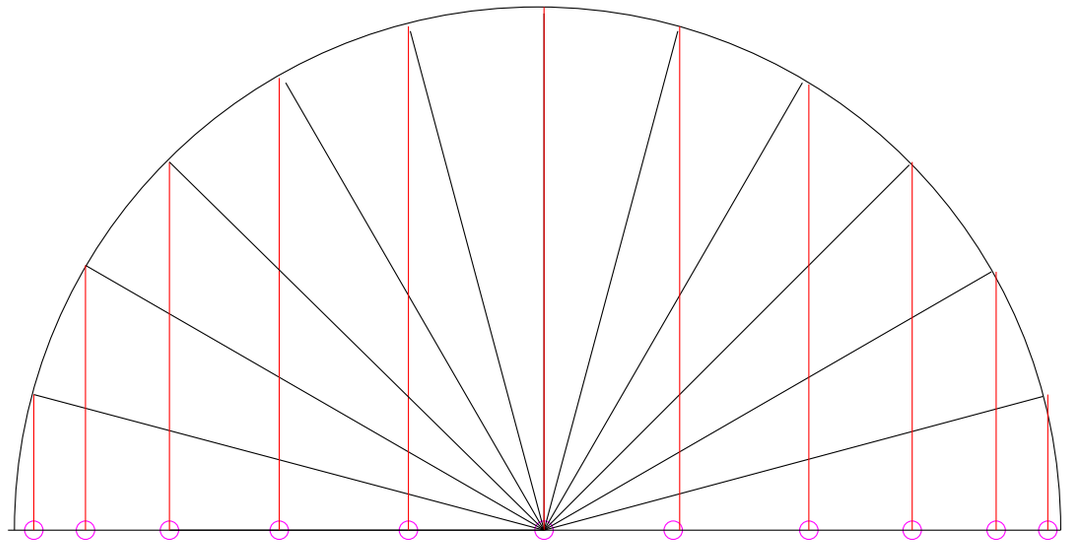
$$\overset{\arccos \in [0,\pi]}{\Leftrightarrow} t \in \left\{ \cos\left(\frac{2k+1}{n}\frac{\pi}{2}\right),\ k = 0,\ldots,n-1 \right\} .$$

Extrema (with value ± 1, alternating signs) of T_n:

$$|T_n(\bar{t}_k)| = 1 \Leftrightarrow \exists\, k = 0,\ldots,n\colon \quad \bar{t}_k = \cos\frac{k\pi}{n} , \qquad \|T_n\|_{L^\infty([-1,1])} = 1 . \tag{9.2.5}$$

Location of Chebychev nodes t_k from (9.2.4):





R. Hiptmair

rev 30401, November 29, 2010

Remark 9.2.6 (3-term recursion for Chebychev polynomial). → [29, (32.2)]

3-term recursion by $\cos(n+1)x = 2\cos nx \cos x - \cos(n-1)x$ with $\cos x = t$:

$$T_{n+1}(t) = 2t\,T_n(t) - T_{n-1}(t) \quad , \quad T_0 \equiv 1 \, , \quad T_1(t) = t \, , \quad n \in \mathbb{N} \, . \tag{9.2.7}$$

This implies:
- $T_n \in \mathcal{P}_n$,
- leading coefficients equal to 2^{n-1},
- T_n linearly independent,
- T_n basis of $\mathcal{P}_n = \operatorname{Span}\{T_0, \ldots, T_n\}$, $n \in \mathbb{N}_0$.

See Code 9.2.1 for algorithmic use of 3-term recursion (9.2.7).

△

Theorem 9.2.8 (Minimax property of the Chebychev polynomials). *[14, Section 7.1.4.], [29, Thm. 32.2]*

$$\|T_n\|_{L^\infty([-1,1])} = \inf\{\|p\|_{L^\infty([-1,1])} : p \in \mathcal{P}_n, p(t) = 2^{n-1}t^n + \cdots\} \, , \quad \forall n \in \mathbb{N} \, .$$

Proof. (indirect) Assume

$$\exists q \in \mathcal{P}_n \text{ , leading coefficient } = 2^{n-1}: \quad \|q\|_{L^\infty([-1,1])} < \|T_n\|_{L^\infty([-1,1])} \ .$$

$(T_n - q)(x) > 0$ in local maxima of T_n
$(T_n - q)(x) < 0$ in local minima of T_n

From knowledge of local extrema of T_n, see (9.2.5):

$$T_n - q \text{ changes sign at least } n+1 \text{ times}$$

$$\Rightarrow \quad T_n - q \text{ has at least } n \text{ zeros}$$

$T_n - q \equiv 0$, because $T_n - q \in \mathcal{P}_{n-1}$ (same leading coefficient!)

Application to approximation by polynomial interpolation: Chebychev interpolation

For $I = [-1,1]$ • "optimal" interpolation nodes $\mathcal{T} = \left\{ \cos\left(\frac{2k+1}{2(n+1)}\pi\right) \ , k = 0, \ldots, n \right\}$,

R. Hiptmair

rev 30401, November 29, 2010

- $w(t) = (t - t_0) \cdots (t - t_{n+1}) = 2^{-n} T_{n+1}(t) \quad , \quad \|w\|_{L^\infty(I)} = 2^{-n}$,

  with leading coefficient 1.

Then, by Thm. 9.1.7,

$$\|f - \mathsf{I}_{\mathcal{T}}(f)\|_{L^\infty([-1,1])} \leq \frac{2^{-n}}{(n+1)!} \left\| f^{(n+1)} \right\|_{L^\infty([-1,1])} . \tag{9.2.9}$$

Remark 9.2.10 (Chebychev polynomials on arbitrary interval).

How to use Chebychev polynomial interpolation on an arbitrary interval?

Scaling argument: interval transformation requires the transport of the functions

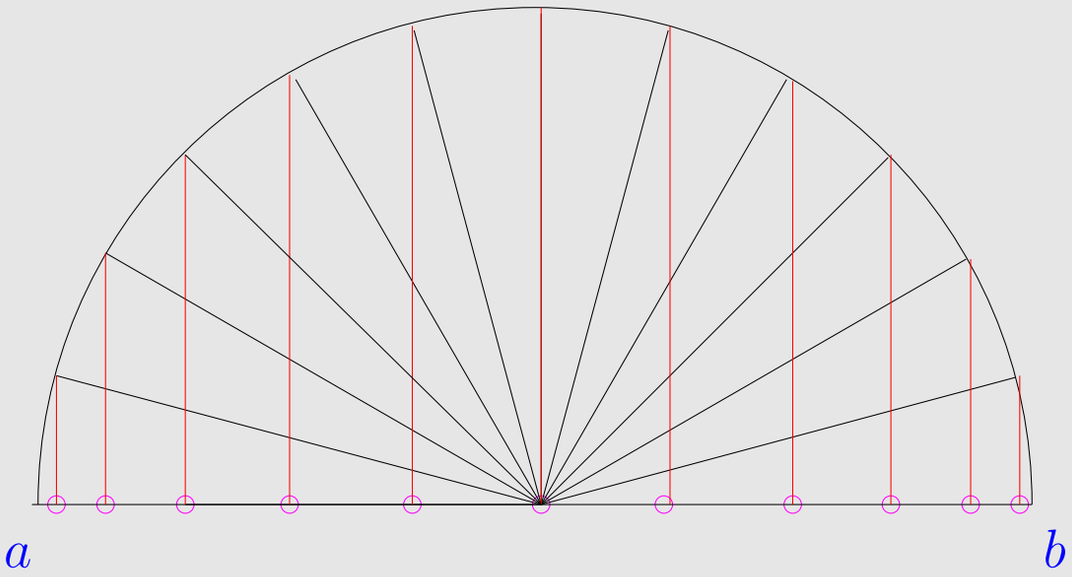
$$[-1,1] \xrightarrow{\widehat{t} \mapsto t := a + \frac{1}{2}(\widehat{t}+1)(b-a)} [a,b] \quad \leftrightarrow \quad \widehat{f}(\widehat{t}) := f(t) .$$

$$p \in \mathcal{P}_n \quad \wedge \quad p(t_j) = f(t_j) \quad \Leftrightarrow \quad \widehat{p} \in \mathcal{P}_n \quad \wedge \quad \widehat{p}(\widehat{t}_j) = \widehat{f}(\widehat{t}_j) .$$

With transformation formula for the integrals & $\frac{\mathrm{d}^n \widehat{f}}{\mathrm{d}\widehat{t}^n}(\widehat{t}) = (\frac{1}{2}|I|)^n \frac{\mathrm{d}^n f}{\mathrm{d}t^n}(t)$:

$$\|f - \mathsf{I}_{\mathcal{T}}(f)\|_{L^\infty(I)} = \left\|\widehat{f} - \mathsf{I}_{\widehat{\mathcal{T}}}(\widehat{f})\right\|_{L^\infty([-1,1])} \leq \frac{2^{-n}}{(n+1)!}\left\|\frac{\mathrm{d}^{n+1}\widehat{f}}{\mathrm{d}\widehat{t}^{n+1}}\right\|_{L^\infty([-1,1])}$$

$$\leq \frac{2^{-2n-1}}{(n+1)!}|I|^{n+1}\left\|f^{(n+1)}\right\|_{L^\infty(I)} \; . \tag{9.2.11}$$



The Chebychev nodes in the interval $I = [a,b]$ are

$$t_k := a + \tfrac{1}{2}(b-a)\left(\cos\left(\frac{2k+1}{2(n+1)}\pi\right) + 1\right) \; , \tag{9.2.12}$$

$$k = 0, \ldots, n \; .$$

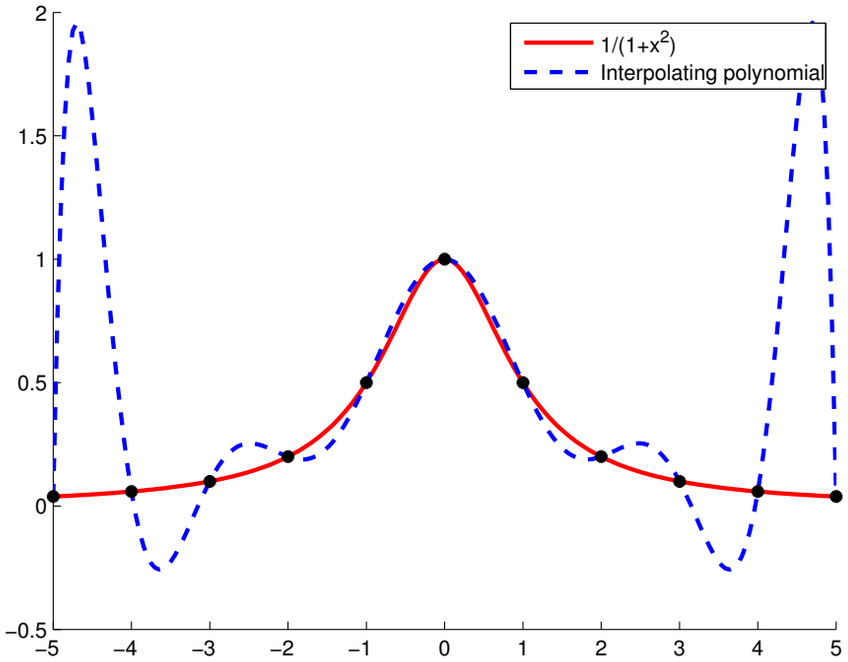
R. Hiptmair

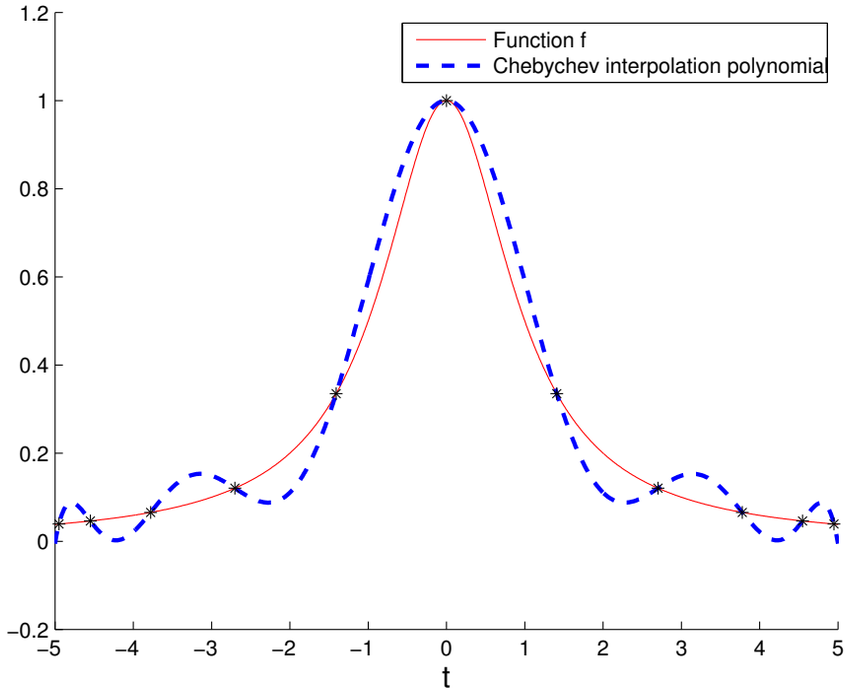
rev 30401, November 29, 2010

9.2.2 Chebychev interpolation error estimates

Example 9.2.13 (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:





◇

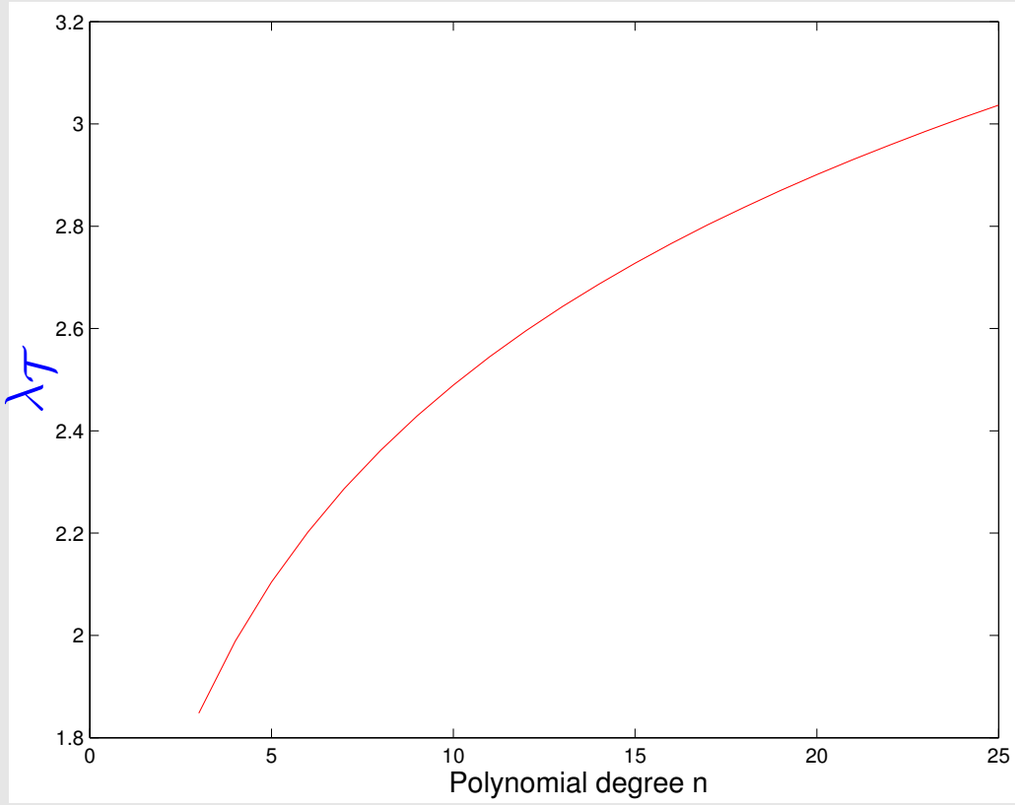
Remark 9.2.14 (Lebesgue Constant for Chebychev nodes). → Sect. 3.3.2

Theory [7, 56, 55]:

$$\lambda_{\mathcal{T}} \sim \frac{2}{\pi}\log(1+n) + o(1) \;,$$

$$\boxed{\lambda_{\mathcal{T}} \leq \tfrac{2}{\pi}\log(1+n) + 1} \;. \qquad (9.2.15)$$

Compare with Lebesgue constant for equidistant nodes, see Rem. 3.3.20.



△

Example 9.2.16 (Chebychev interpolation error).

R. Hiptmair

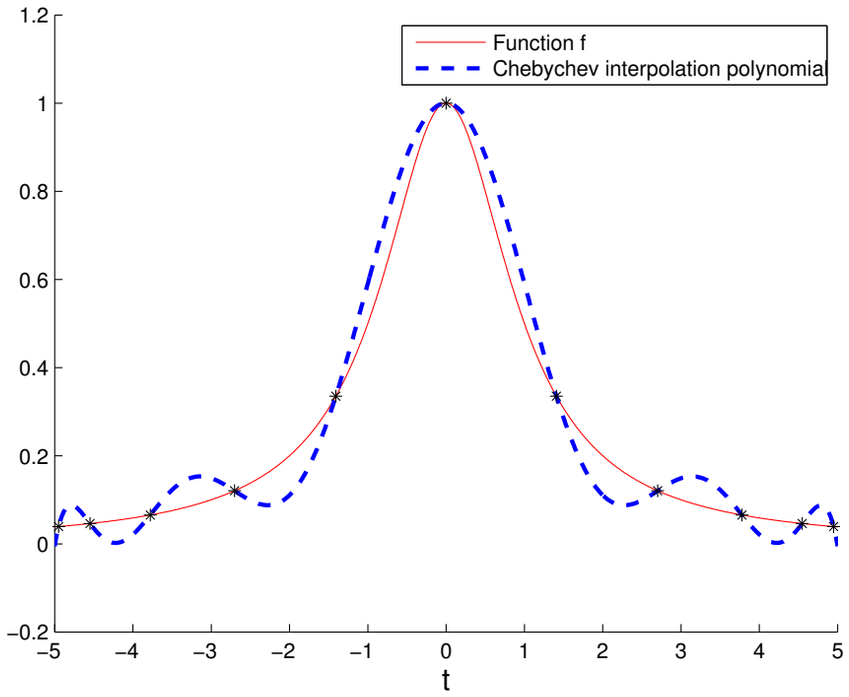
rev 30401, November 29, 2010

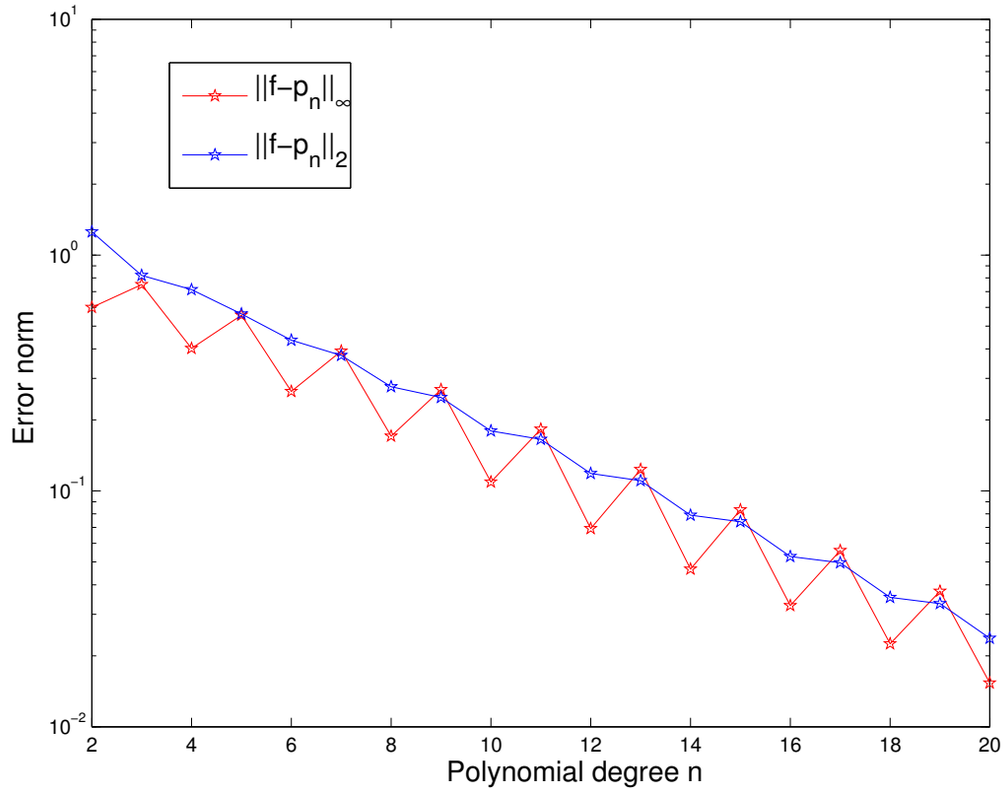
For $I = [a,b]$ let $x_l := a + \frac{b-a}{N} l,\ l = 0,...,N,\ N = 1000$ we approximate the norms of the error

$$||f - p||_\infty \approx \max_{0 \le l \le N} |f(x_l) - p(x_l)| \tag{9.2.17}$$

$$||f - p||_2^2 \approx \frac{b-a}{2N} \sum_{0 \le l < N} \left(|f(x_l) - p(x_l)|^2 + |f(x_{l+1}) - p(x_{l+1})|^2 \right) \tag{9.2.18}$$

① $f(t) = (1+t^2)^{-1}$, $I = [-5,5]$ (see Ex. 9.1.5)
Interpolation with $n = 10$ Chebychev nodes (plot on the left).





R. Hiptmair

rev 30401, November 29, 2010

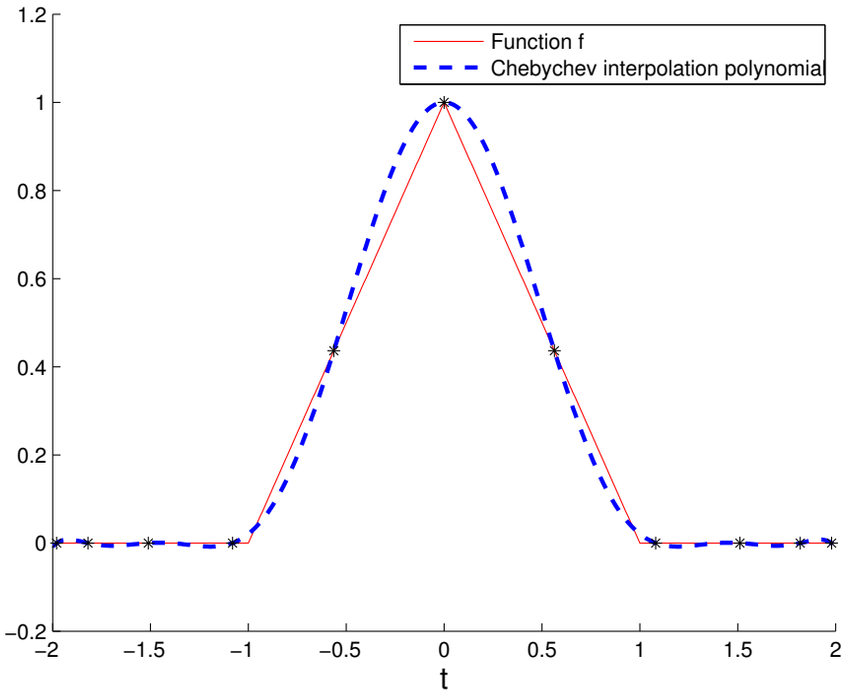
Notice: exponential convergence of the Chebychev interpolation:

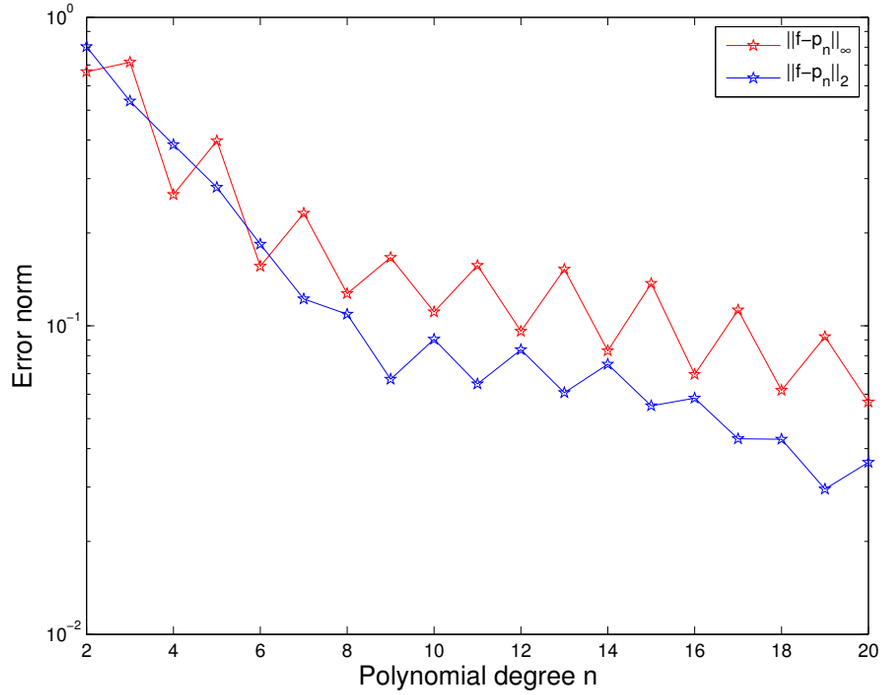
$$p_n \to f, \quad \|f - \mathsf{I}_n f\|_{L^\infty([-5,5])} \approx 0.8^n$$

.

② $f(t) = \max\{1 - |t|, 0\}$, $I = [-2, 2]$, $n = 10$ nodes (plot on the left).

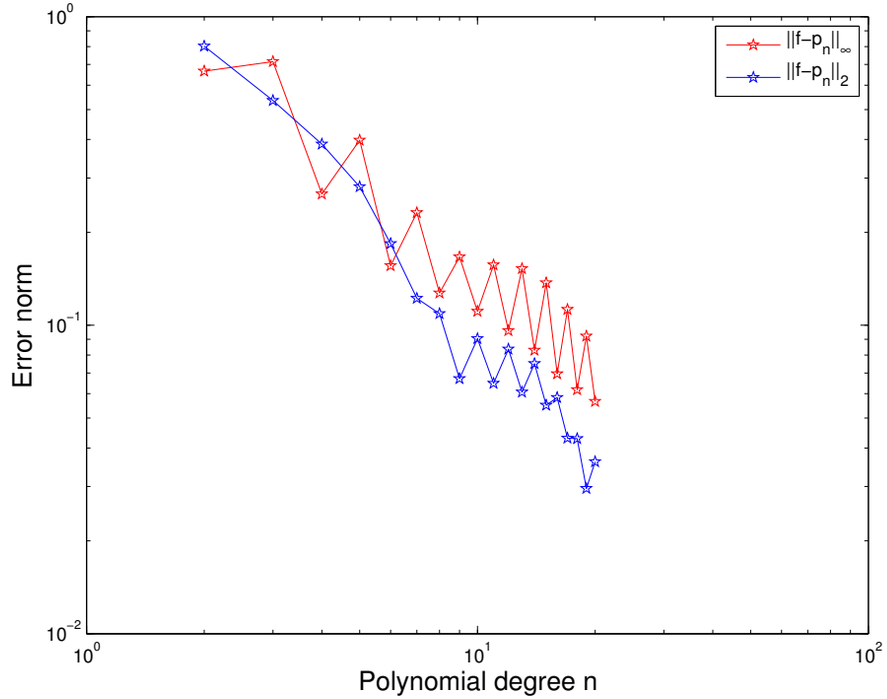
Now $f \in C^0(I)$ but $f \notin C^1(I)$.





R. Hiptmair

rev 30401, November 29, 2010



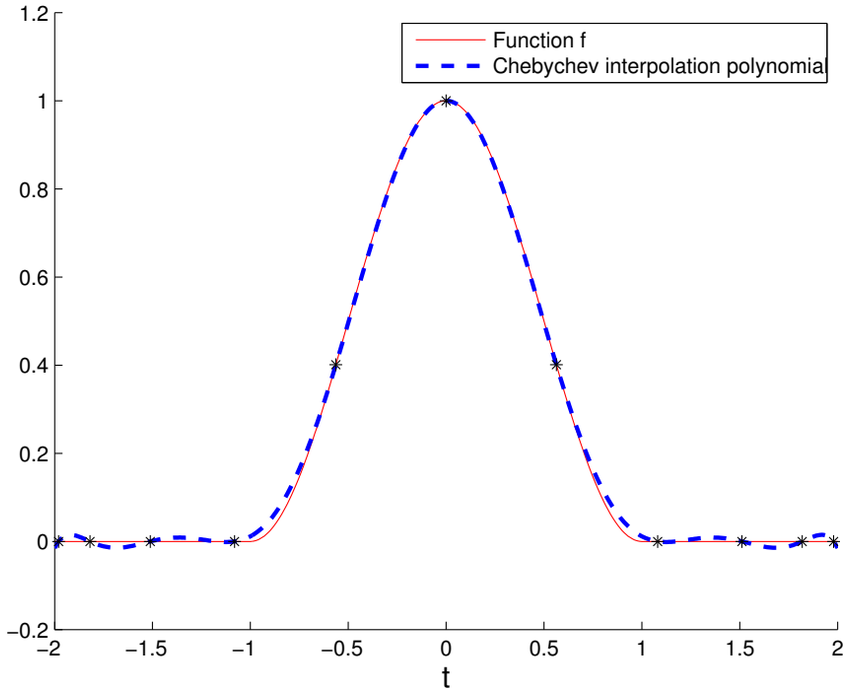
From the doubly logarithmic plot we conclude ➙

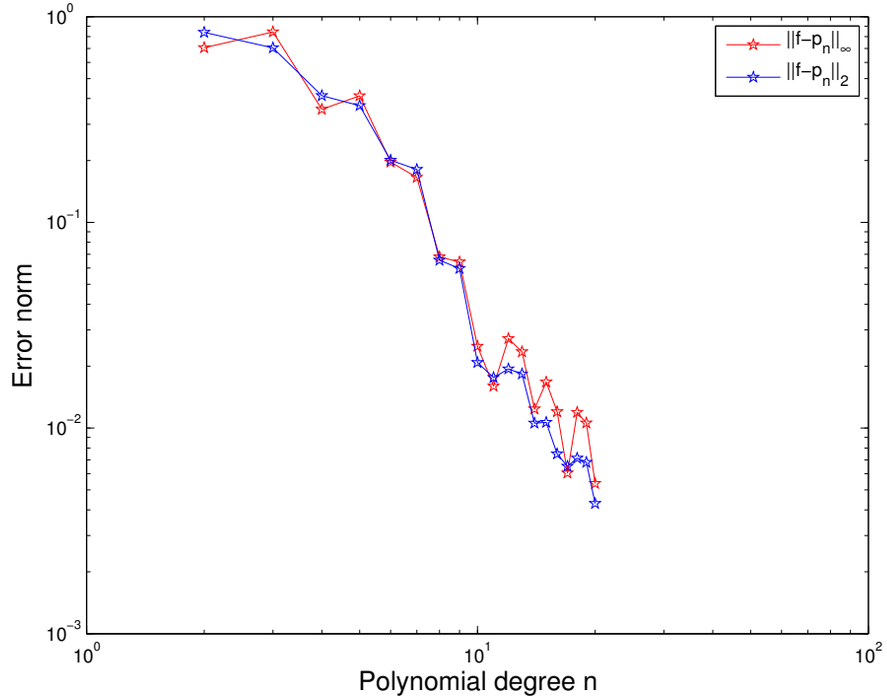
- no exponential convergence
- algebraic convergence (**?**)

③ $f(t) = \begin{cases} \frac{1}{2}(1+\cos \pi t) & |t| < 1 \\ 0 & 1 \leq |t| \leq 2 \end{cases}$ $I = [-2,2]$, $n = 10$ (plot on the left).

R. Hiptmair

rev 30401, November 29, 2010





Notice: only (vaguely) algebraic convergence.

◇

R. Hiptmair

Summary of observations, *cf.* Rem. 9.1.4:

- Essential role of smoothness of f: slow convergence of approximation error of the Cheychev interpolant if f enjoys little smoothness, *cf.* also (9.1.11),
- for smooth $f \in C^\infty$ approximation error of the Cheychev interpolant seems to decay to zero exponentially in the polynomial degree n.

Remark 9.2.19 (Chebychev interpolation of analytic functions). $\rightarrow$ complex analysis

Change of perspective:

Consider the interpoland $f : I \subset \mathbb{R} \mapsto \mathbb{R}$ as the restriction to I of a complex valued function $f : D \subset \mathbb{C} \mapsto \mathbb{C}$, where D is an open neighborhood of I in $\mathbb{C}$.

Example: $f(t) = \dfrac{1}{1+x^2}$ on $I \subset \mathbb{R}$

$$\updownarrow$$

$$f : \mathbb{C} \setminus \{+i, -i\} \mapsto \mathbb{C} \text{ with } f(z) = \frac{1}{1+z^2}$$

Definition 9.2.20 (Analytic function).
A function $f : D \subset \mathbb{C} \mapsto \mathbb{C}$ is called analytic ($\leftrightarrow$ holomorphic) (in D), if (in the sense of uniform convergence)

$$\forall z \in D\colon \quad \exists r > 0, (\alpha_k)_{k=0}^{\infty} \in \mathbb{C}^{\mathbb{N}_0}\colon \quad |w - z| < r \quad \Rightarrow \quad f(w) = \sum_{k=0}^{\infty} \alpha_k (w-z)^k \ .$$

Analyticity $\leftrightarrow$ locally representable by convergent power series

Examples for functions that are analytic in $\mathbb{C}$: $\exp$, $\sin$, $\cos$, $\sinh$, $\cosh$

Rational functions ($\hat{=}$ quotients of polynomials) are analytic everywhere away from their poles ($\hat{=}$ zeros of denominator).

R. Hiptmair

rev 30401, November 29, 2010

Result. Chebychev interpolants of $f : I \mapsto \mathbb{C}$, $I \subset \mathbb{R}$ interval, converge exponentially in the degree, if f is analytic in an open $\mathbb{C}$-neighborhood of I. The rate of exponential convergence increases with the size of this neighborhood relative to the length of I.

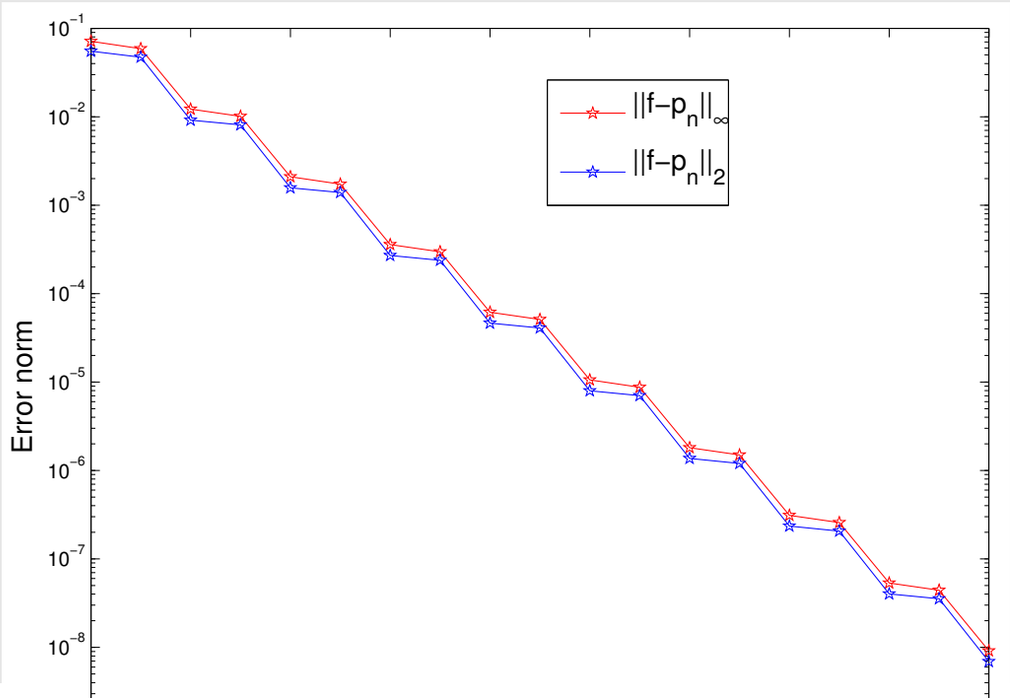
△

Example 9.2.21 (Chebychev interpolation of analytic function). → Ex. 9.2.16 cnt'd

Modification: the same function $f(t) = (1+t^2)^{-1}$ on a smaller interval $I = [-1, 1]$.

(Faster) exponential convergence than on $I =]-5, 5[$:

$$\|f - \mathsf{I}_n f\|_{L^2([-1,1])} \approx 0.42^n.$$



Explanation, *cf.* Rem. 9.2.19: for $I = [-1, 1]$ the poles $\pm i$ of f are farther away relative to the size of the interval than for $I = [-5, 5]$.

◇

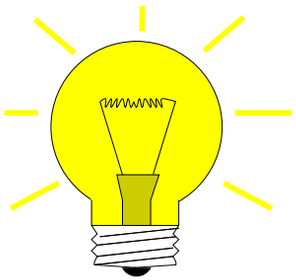
9.2.3 Chebychev interpolation: computational aspects

Task: Given: given degree $n \in \mathbb{N}$, continuous function $f : [-1, 1] \mapsto \mathbb{R}$

Sought: efficient representation/evaluation of interpolating polynomial $p \in \mathcal{P}_n$ in Chebychev nodes (9.2.12) on $[-1, 1]$

```
1 class ChebInterp {
2   private:
3   // various internal data describing Chebychev interpolating polynomial p
4   public:
5     // Constructor taking function f and degree n as arguments
6     PolyInterp(const Function &f,unsigned int n);
```

```
7       // Evaluation operator:  y_j = p(x_j), j = 1,...,m (m "large")
8       double eval(const vector<double> &x,vector <double> &y) const;
9   };
```



Trick: expand p into Chebychev polynomials

$$p = \sum_{j=0}^{n} \alpha_j T_j \ , \quad \alpha_j \in \mathbb{R} \ . \tag{9.2.22}$$

The representation (9.2.22) is always possible, because $\{T_0, \ldots, T_n\}$ is a *basis* of $\mathcal{P}_n$

Remark 9.2.23 (Fast evaluation of Chebychev expansion). → [29, Alg. 32.1]

Use the 3-term recurrence (9.2.7)

$$T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x) \ , \quad j = 2, 3, \ldots \ ,$$

R. Hiptmair

rev 30401, November 29, 2010

to rewrite (9.2.22) as

$$p(x) = \sum_{j=0}^{n-1} \widetilde{\alpha}_j T_j \quad \text{with} \quad \widetilde{\alpha}_j = \begin{cases} \alpha_j + 2x\alpha_{j+1} & \text{, if } j = n-1 \text{ ,} \\ \alpha_j - \alpha_{j+2} & \text{, if } j = n-2 \text{ ,} \\ \alpha_j & \text{else.} \end{cases} \tag{9.2.24}$$

recursive algorithm, see Code 9.2.24.

Code 9.2.25: Recursive evaluation of Chebychev expansion (9.2.22)

```
function y = recclenshaw(a,x)
% Recursive evaluation of a polynomial p = sum_{j=1}^{n+1} a_j T_{j-1} at point x, see (9.2.24)
% The coefficients a_j have to be passed in a row vector.
n = length(a)-1;
if (n<2), y = a(1)+x*a(2);
else
  y = recclenshaw([a(1:n-2), a(n-1)-a(n+1), a(n)+2*x*a(n+1)],x);
end
```

Non-recursive version: Clenshaw algorithm

Code 9.2.26: Clenshaw algorithm for evalation of Chebychev expansion (9.2.22)

```
function y = clenshaw(a,x)
% Clenshaw algorithm for evaluating p = sum_{j=1}^{n+1} a_j T_{j-1} at points passed in vector x
n = length(a)-1; % degree of polynomial
```

R. Hiptmair

rev 30401, November 29, 2010

```
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,:);
```


Computational effort : $O(nm)$ for evaluation at m points

△

Issue: How to compute the Chebychev expansion coefficients α_j in (9.2.22) efficiently from the interpolation conditions

$$p(t_k) = f(t_k)\,, \quad k = 0, \ldots, n\,, \quad \text{for} \quad t_k := \cos\left(\frac{2k+1}{2(n+1)}\pi\right)\,. \tag{9.2.27}$$

Chebychev nodes

Trick: transformation of p into a 1-periodic function:

$$q(s) := p(\cos 2\pi s) = \sum_{j=0}^{n} \alpha_j T_j(\cos 2\pi s) \exp(-2\pi i n s) \overset{\text{Def. 9.2.1}}{=} \sum_{j=0}^{n} \alpha_j \cos(2\pi j s)$$

$$= \sum_{j=0}^{n} \tfrac{1}{2}\alpha_j \big(\exp(2\pi i j s) + \exp(-2\pi i j s)\big) \tag{9.2.28}$$

$$= \sum_{j=-n}^{n+1} \beta_j \exp(-2\pi i j s)\ , \quad \beta_j := \begin{cases} 0 & \text{, for } j = n+1\ , \\ \frac{1}{2}\alpha_j & \text{, for } j = 1,\ldots,n\ , \\ \alpha_0 & \text{, for } j = 0\ , \\ \frac{1}{2}\alpha_{n-j} & \text{, for } j = -n,\ldots,-1\ . \end{cases}$$

Transformed interpolation conditions (9.2.27):

$$t = cos(2\pi s) \overset{(9.2.27)}{\Longrightarrow} q\left(\frac{2k+1}{4(n+1)}\right) = f(t_k)\ , \quad k = 0,\ldots,n\ . \tag{9.2.29}$$

R. Hiptmair

rev 30401, November 29, 2010

Observe symmetry

$$q(s) = q(1-s)$$
$$\Downarrow\leftarrow \text{(9.2.29)}$$
$$q(1 - \frac{2k+1}{4(n+1)}) = y_k \ , \quad k = 0, \ldots, n \ .$$

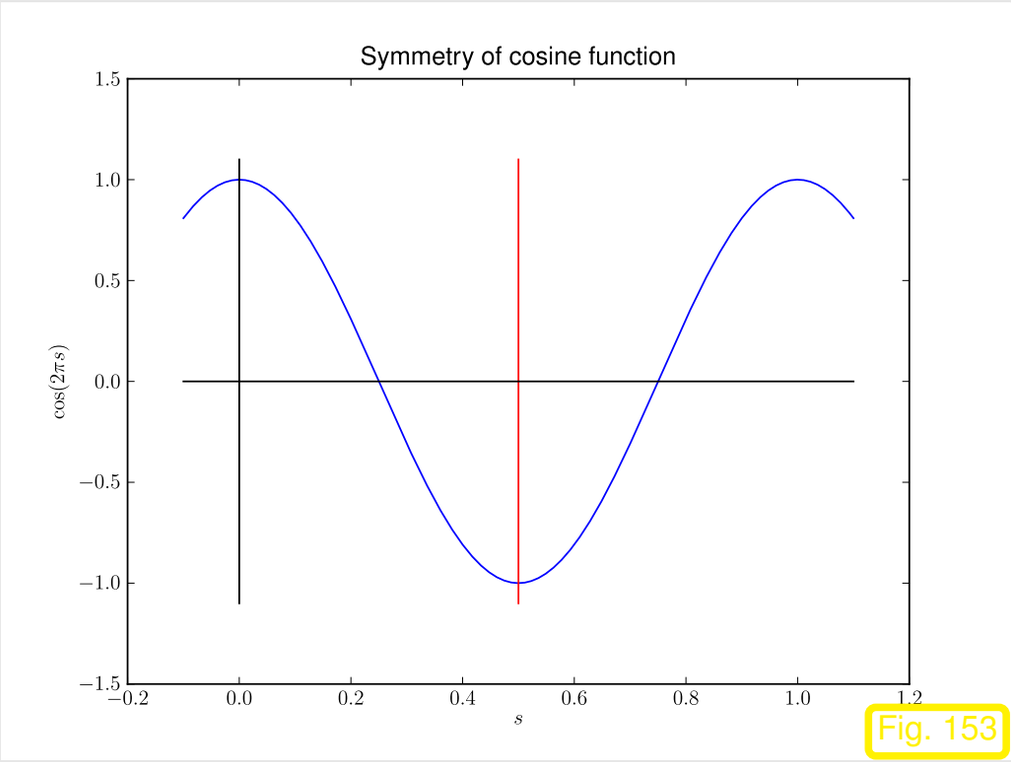


Fig. 153

Extend interpolation conditions (9.2.29) by symmetry, see Fig. 154

$$q\left(\frac{k}{2(n+1)} + \frac{1}{4(n+1)}\right) = z_k := \begin{cases} y_k & \text{, for } k = 0, \ldots, n \ , \\ y_{2n+1-k} & \text{, for } k = n+1, \ldots, 2n+1 \ . \end{cases} \tag{9.2.30}$$

R. Hiptmair

rev 30401, November 29, 2010

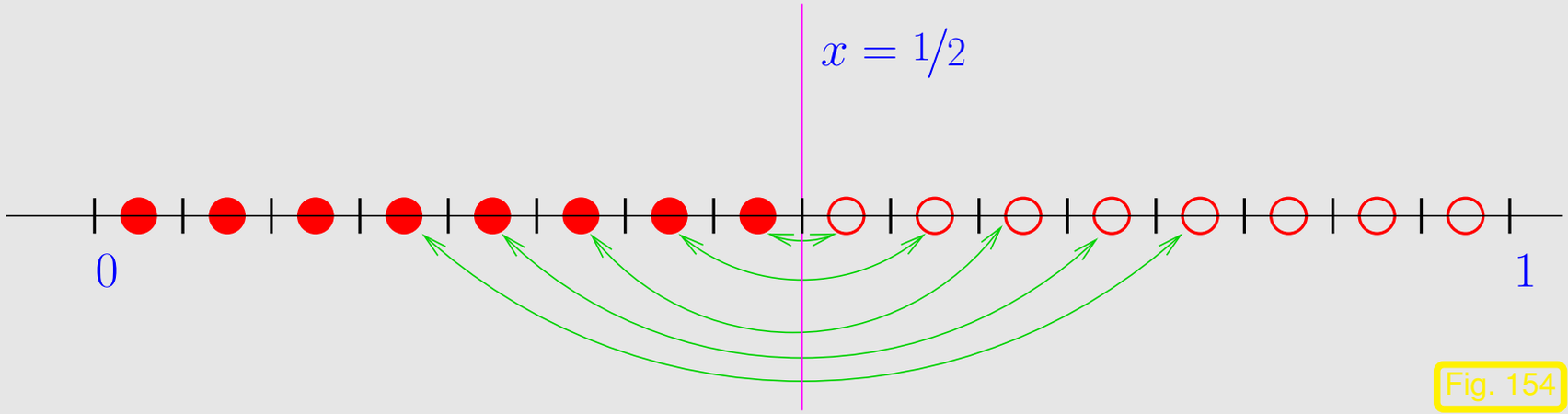


Fig. 154

$\Longleftrightarrow$ linear system of equations (at last):

$$q\left(\frac{k}{2(n+1)}+\frac{1}{4(n+1)}\right) = \sum_{j=-n}^{n+1}\left(\beta_j \exp(-\frac{2\pi ij}{4(n+1)})\right)\exp\left(-\frac{2\pi i}{2n+1}kj\right) = z_k\ .$$

$$\Updownarrow$$

$$\sum_{j=0}^{2n+1}\left(\beta_j \exp(-\frac{2\pi i(j-n)}{4(n+1)})\right)\underbrace{\exp\left(-\frac{2\pi i}{2n+1}kj\right)}_{\omega_{2(n+1)}^{kj}} = \exp(-\pi i\frac{nk}{n+1})z_k\ , \quad k=0,\ldots,2n+1\ .$$

$$\Updownarrow$$

$$\mathbf{F}_{2(n+1)}\mathbf{c} = \mathbf{z} \quad \text{with} \quad \mathbf{c} = \left(\beta_j \exp(-\frac{2\pi ij}{4(n+1)})\right)_{j=0}^{2n+1}\ . \tag{9.2.31}$$

$(2n+2)\times(2n+2)$ Fourier matrix, see (8.2.9)

R. Hiptmair

rev 30401, November 29, 2010

solve (9.2.31) with inverse discrete Fourier transform, see 8.2:

asymptotic complexity $O(n \log n)$ ($\rightarrow$ Sect. 8.3)

Note: by symmetry of $\mathbf{z}$ ➥ $\beta_{2n+1} = 0$, *cf.* (9.2.28)!

Code 9.2.32: Efficient computation of Chebychev expansion coefficient of Chebychev interpolant

```
function a = chebexp(y)
% Efficiently compute coefficients α_j in the Chebychev expansion
% p = sum_{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)
```

R. Hiptmair

rev 30401, November 29, 2010

Remark 9.2.33 (Chebychev representation of built-in functions).

Computers use approximation by sums of Chebychev polynomials in the computation of functions like $\log, \exp, \sin, \cos, \ldots$. The evaluation by means of Clenshaw algorithm according to Code 9.2.25 is more efficient and stable than for approximation by Taylor polynomials.

△

9.3 Trigonometric interpolation [10, Sect. 8.5]

Task: approximation of continuous 1-periodic function

$$f \in C^0(\mathbb{R}) \quad , \quad f(t+1) = f(t) \quad \forall t \in \mathbb{R} \, .$$

Dubious: approximation by global polynomials, because they are not 1-periodic ➢ global polynomial interpolant will lack essential structural property.

Idea: approximation by interpolation based on spaces of periodic functions

Natural candidates: trigonometric functions (sines and cosines)

Trigonometric interpolation [10, pp. 304]

Given $t_0 < t_1 < \cdots < t_{2n}$, $t_k \in [0,1[$, and $y_k \in \mathbb{R}$, $k = 0, \ldots, 2n$ find

$$q \in \mathcal{P}_{2n}^T := \operatorname{Span}\{t \mapsto \cos(2\pi jt), t \mapsto \sin(2\pi jt)\}_{j=0}^{n} \ , \tag{9.3.1}$$

$$\text{with} \quad q(t_k) = y_k \quad \text{for all} \quad k = 0, \ldots, 2n \ . \tag{9.3.2}$$

Terminology: $\mathcal{P}_{2n}^T \hat{=}$ space of trigonometric polynomials of degree $2n$.

Trigonometric interpolation = linear interpolation mapping ($\rightarrow$ Rem. 3.1.5) into function space $\mathcal{P}_{2n}^T$

- $\dim \mathcal{P}_{2n}^T = 2n+1$, because functions are linearly independent, sine dropped for $j = 0$
- functions in (9.3.1) provide basis functions for interpolation, *cf.* (3.1.4)

R. Hiptmair

rev 30401, November 29, 2010

Coefficient representation of $q \in \mathcal{P}_{2n}^T$

$$q(t) = \alpha_0 + \sum_{j=1}^{} \alpha_j \cos(2\pi jt) + \beta_j \sin(2\pi jt) \;, \quad \alpha_j, \beta_j \in \mathbb{R} \;. \tag{9.3.3}$$

Why is $\mathcal{P}_{2n}^T$ called a space of trigonometric *polynomials* **?**

Recall expressions for trigonometric functions via complex exponentials:

$$e^{it} = \cos t + i \sin t \quad \Rightarrow \quad \begin{cases} \cos t = \frac{1}{2}(e^{it} + e^{-it}) \\ \sin t = \frac{1}{2i}(e^{it} - e^{-it}) \;. \end{cases} \tag{9.3.4}$$

R. Hiptmair

rev 30401, November 29, 2010

▶ representation of trigonometric polynomials by means of complex exponentials:

$$\begin{aligned} q(t) &= \alpha_0 + \sum_{j=1}^{n} \alpha_j \cos(2\pi jt) + \beta_j \sin(2\pi jt) \\ &= \alpha_0 + \frac{1}{2}\Big\{ \sum_{j=1}^{n} (\alpha_j - i\beta_j) e^{2\pi ijt} + (\alpha_j + i\beta_j) e^{-2\pi ijt} \Big\} \end{aligned}$$

$$= \alpha_0 + \tfrac{1}{2}\sum_{j=-n}^{-1}(\alpha_{-j} + i\beta_{-j})e^{2\pi ijt} + \tfrac{1}{2}\sum_{j=1}^{n}(\alpha_j - i\beta_j)e^{2\pi ijt}$$

$$= e^{-2\pi int}\sum_{j=0}^{2n}\gamma_j e^{2\pi ijt}\ , \quad \text{with} \quad \gamma_j = \begin{cases} \frac{1}{2}(\alpha_{n-j} + i\beta_{n-j}) & \text{for } j = 0,\ldots,n-1\ , \\ \alpha_0 & \text{for } j = n\ , \\ \frac{1}{2}(\alpha_{j-n} - i\beta_{j-n}) & \text{for } j = n+1,\ldots,2n\ . \end{cases} \tag{9.3.5}$$

Note: $\gamma_j \in \mathbb{C}$ ➢ work in $\mathbb{C}$ in the context of trigonometric interpolation! Admit $y_k \in \mathbb{C}$.

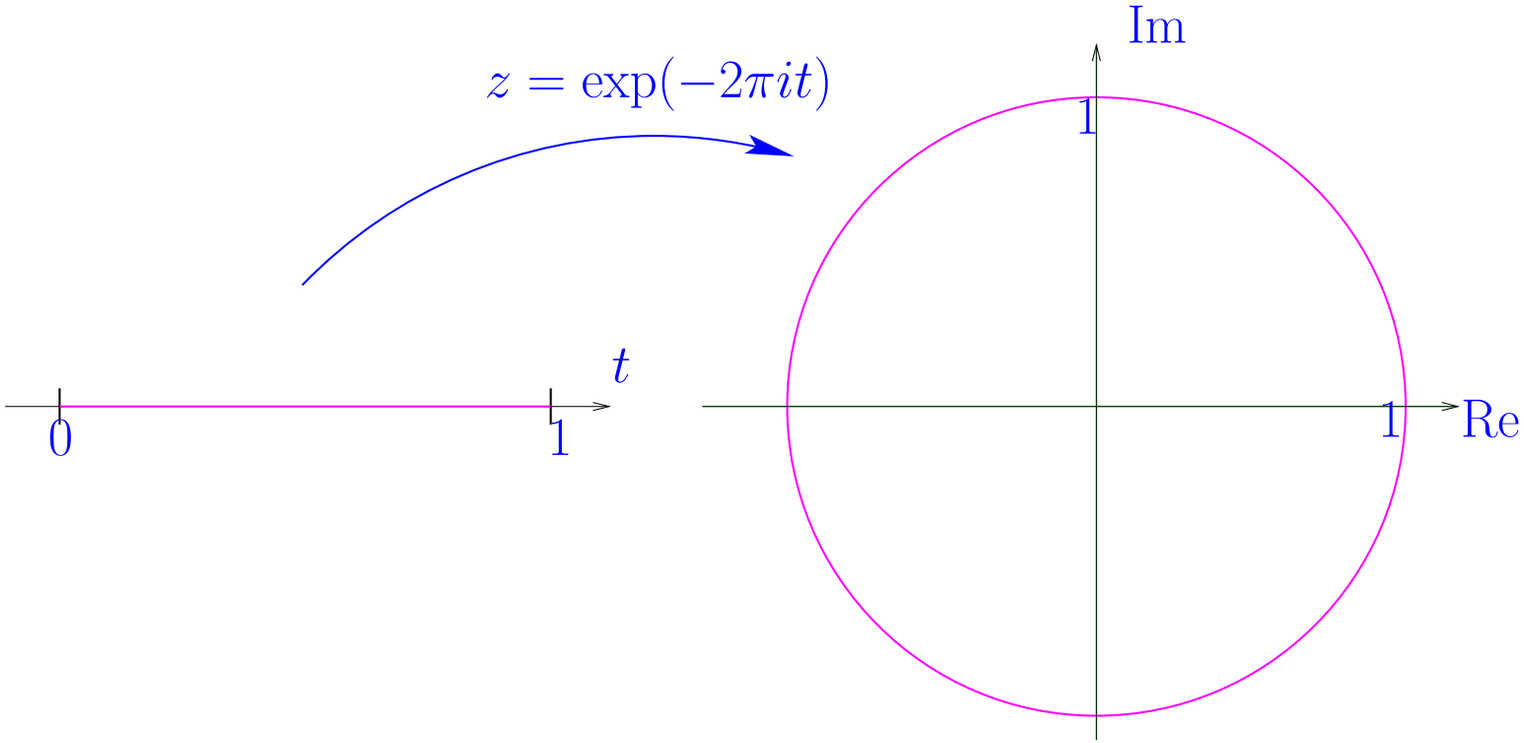
$$q \in \mathcal{P}_{2n+1}^T \quad \Rightarrow \quad q(t) = e^{-2\pi int}\cdot p(e^{2\pi it}) \quad \text{with} \quad p(z) = \sum_{j=0}^{2n}\gamma_j z^j \in \mathcal{P}_{2n}\ , \tag{9.3.6}$$

a polynomial !

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 $\mathbb{C}$.

✎ notation: $\mathbb{S}^1 := \{z \in \mathbb{C}: |z| = 1\}$



Trigonometric interpolation through data points (t_k, y_k)	$\Longleftrightarrow$	Polynomial interpolation through data points $(e^{2\pi i t_k}, y_k)$

R. Hiptmair

rev 30401, November 29, 2010

Trigonometric interpolation = polynomial interpolation on $\mathbb{S}^1$

All theoretical results and algorithms from polynomial interpolation carry over to trigonometric interpolation

- Existence and uniqueness of trigonometric interpolation polynomial, see Thm. 3.3.6,
- Concept of Lagrange polynomials, see (3.3.3),
- the algorithms and representations discussed in Sect. 3.4.

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
```

```
11 p = intpolyval(tc,z,exp(2*pi*i*x));
12 q = exp(-2*pi*i*n*x).*p; % Undo the scaling, see (9.3.6)
```


Back to **Approximation of 1-periodic functions by trigonometric interpolant**

Remember:additional freedom to choose interpolation nodes in the setting of function approximation

Reasonable choice for generic 1-periodic f: uniformly distributed nodes $t_k = \frac{k}{2n+1}$, $k = 0, \ldots, 2n$

Justification: the circle is utterly “isotropic”!

$(2n+1) \times (2n+1)$ linear system of equations:

$$\sum_{j=0}^{2n} \gamma_j \exp\big(2\pi i \frac{jk}{2n+1}\big) = z_k := \exp\big(2\pi i \frac{nk}{2n+1}\big) y_k \;, \quad k = 0, \ldots, 2n \;.$$

$$\Updownarrow$$

$$\overline{\mathbf{F}}_{2n+1}\mathbf{c} = \mathbf{z} \;, \quad \mathbf{c} = (\gamma_0, \ldots, \gamma_{2n})^T \quad \overset{\text{Lemma 8.2.10}}{\Rightarrow} \quad \mathbf{c} = \frac{1}{2n+1}\mathbf{F}_{2n}\mathbf{z} \;. \tag{9.3.8}$$

$(2n+1) \times (2n+1)$ (conjugate) Fourier matrix, see (8.2.9)

Fast solution by means of FFT: $O(n \log n)$ asymptotic complexity, see Sect. 8.3

Code 9.3.10: Efficient computation of coefficient of trigonometric interpolation polynomial (*equidistant nodes*)

```
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): alpha_j = 1/2(gamma_{n-j} + gamma_{n+j}) and beta_j = 1/2i(gamma_{n-j} - gamma_{n+j}), j = 1,...,n, alpha_0 = gamma_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

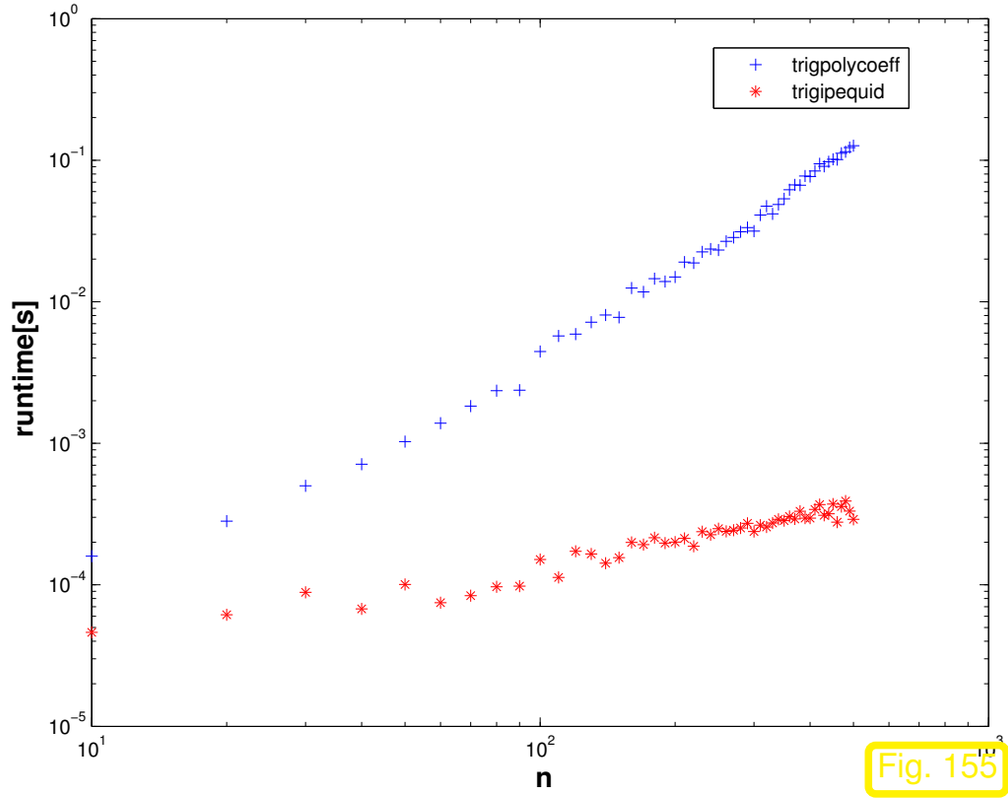


Fig. 155

R. Hiptmair

rev 30401, November 29, 2010

Code 9.3.13: Runtime comparison

```
function trigipequidtiming
% Runtime comparison between efficient (→ Code 9.3.8) and direct computation
% (→ Code 9.3.10 of coefficients of trigonoetric interpolation polynomial in
% equidistant points.
Nruns = 3; times = [];
for n = 10:10:500
  disp(n)
  N = 2*n+1; t = 0:1/N:(1-1/N); y = exp(cos(2*pi*t));
  t1 = realmax; t2 = realmax;
```

```
  for k=1:Nruns
    tic; [a,b] = trigpolycoeff(t,y); t1 = min(t1,toc);
    tic;  [a,b] = trigipequid(y); t2 = min(t2,toc);
  end
  times = [times; n , t1 , t2];
end

figure; loglog(times(:,1),times(:,2),'b+',...
               times(:,1),times(:,3),'r*');
xlabel('{\bf n}','fontsize',14);
ylabel('{\bf runtime[s]}','fontsize',14);
legend('trigpolycoeff','trigipequid','location','best');

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


Same observation as in Ex. 8.3.1: massive gain in efficiency through relying on FFT.

◇

Remark 9.3.14 (Efficient evaluation of trigonometric interpolation polynomials).

Task: evaluation of trigonometric polynomial (9.3.3) at *equidistant* points $\frac{k}{N}$, $N > 2n$. $k = 0, \ldots, N-1$.

$$\text{(9.3.5)} \quad \blacktriangleright \quad q(k/N) = e^{-2\pi ik/N} \sum_{j=0}^{2n} \gamma_j \exp(2\pi i \frac{kj}{N}) \;, \quad k = 0, \ldots, N-1 \;.$$

$$\blacktriangleright \quad q(k/N) = e^{-2\pi ikn/N} v_j \quad \text{with} \quad \mathbf{v} = \overline{\mathbf{F}}_N \widetilde{\mathbf{c}} \;, \tag{9.3.15}$$

Fourier matrix, see (8.2.9).

where $\widetilde{\mathbf{c}} \in \mathbb{C}^N$ is obtained by zero padding of $\mathbf{c} := (\gamma_0, \ldots, \gamma_{2n})^T$:

$$(\widetilde{\mathbf{c}})_k = \begin{cases} \gamma_j & \text{, for } k = 0, \ldots, 2n \;, \\ 0 & \text{, for } k = 2n+1, \ldots, N-1 \;. \end{cases}$$

Code 9.3.16: Fast evaluation of trigonometric polynomial at *equidistant* points

```
1 function q = trigipequidcomp(a,b,N)
2 % Efficient evaluation of trigonometric polynomial at equidistant points
3 % column vectors a and b pass coefficients α_j, β_j in
4 % representation (9.3.3)
5 n = length(a)-1; if (N < (2*n-1)), error('N too small'); end;
6 gamma = transpose(0.5*[a(end:-1:2)+i*b(end:-1:1);...
7                       2*a(1);a(2:end)-i*b(1:end)]);
8 ch = [gamma,zeros(1,N-(2*n+1))]; % zero padding
```

```
v = conj(fft(conj(ch))); % Multiplication with conjugate Fourier matrix
q = exp(-2*pi*i*n*(0:N-1)/N).*v; % undo rescaling
```

Code 9.3.17: *Equidistant* points: fast on the fly evaluation of trigonometric interpolation polynomial

```
function q = trigpolyvalequid(y,N)
% Evaluation of trigonometric interpolation polynomial through (j/(2n+1), y_j),
  j = 0,...,2n
% in equidistant points k/N, k = 0, N-1
% y has to be a row vector of odd length, values returned as rwo vector, too.
N = length(y); if (mod(N,2)~=1), error('#pts odd!'); end;
n = (N-1)/2;
% Compute coefficients γ_j in (9.3.5), see (9.3.8)
c = fft(exp(2*pi*i*(n/N)*(0:2*n)).*y)/N;
ch = [gamma, zeros(1,N-(2*n+1))]; % zero padding
v = conj(fft(conj(ch))); % Evaluate multiplication with conjugate Fourier
  matrix
q = exp(-2*pi*i*n*(0:N-1)/N).*v; % undo rescaling
```


R. Hiptmair

rev 30401, November 29, 2010

Now we study the asymptotic behavior of the error of trigonometric interpolation as $n \to \infty$.

Example 9.3.18 (Interpolation error: trigonometric interpolation).

#1 Step function: $f(t) = 0$ for $|t - \frac{1}{2}| > \frac{1}{4}$, $f(t) = 1$ for $|t - \frac{1}{2}| \leq \frac{1}{4}$

#2 C^∞ periodic function: $f(t) = \dfrac{1}{\sqrt{1 + \frac{1}{2}\sin(2\pi t)}}$.

#3 "wedge function": $f(t) = |t - \frac{1}{2}|$

Approximate computation of norms of interpolation errors on equidistant grid with 4096 points.

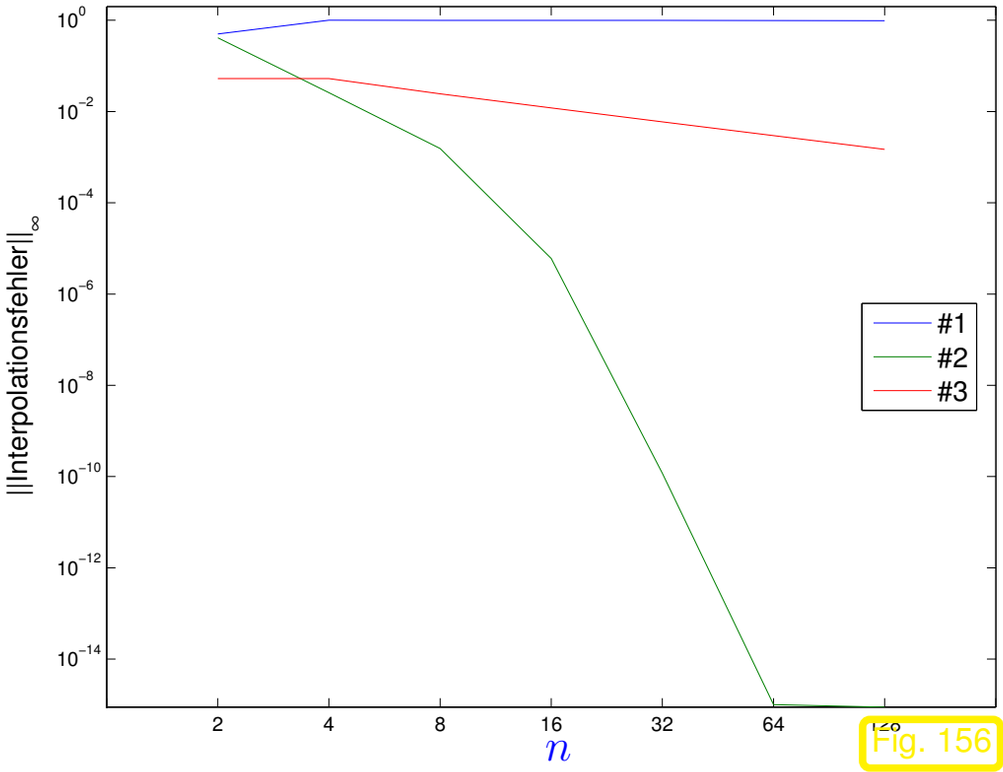


Fig. 156

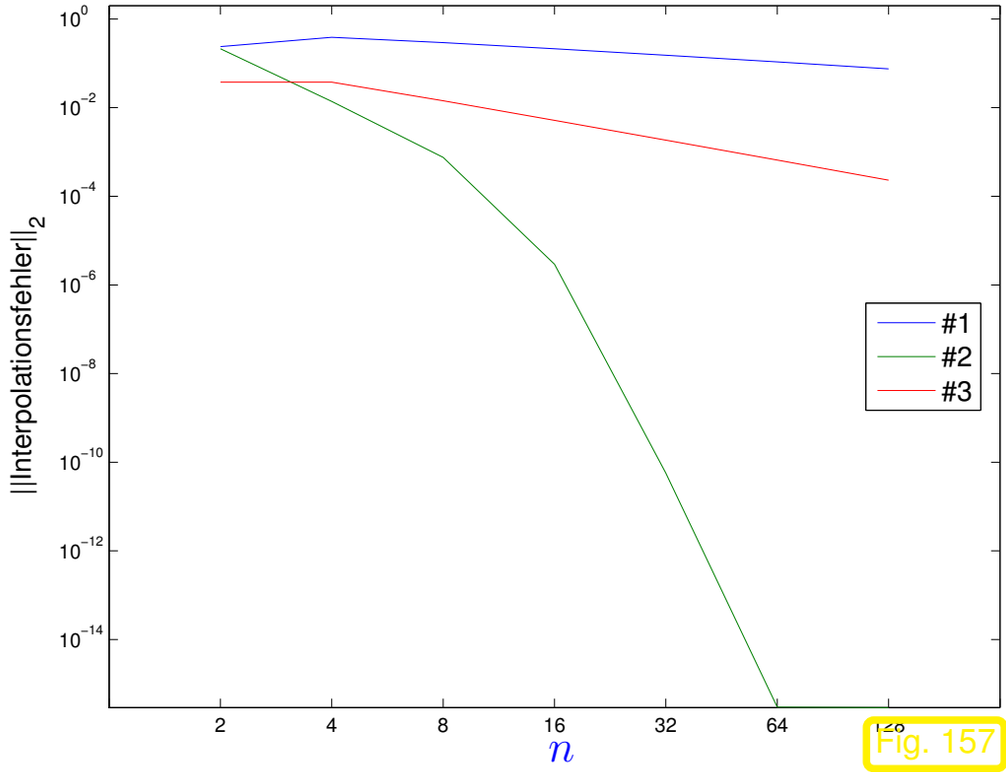


Fig. 157

Observation: Function #1: no convergence in L^{∞}-norm, algebraic convergence

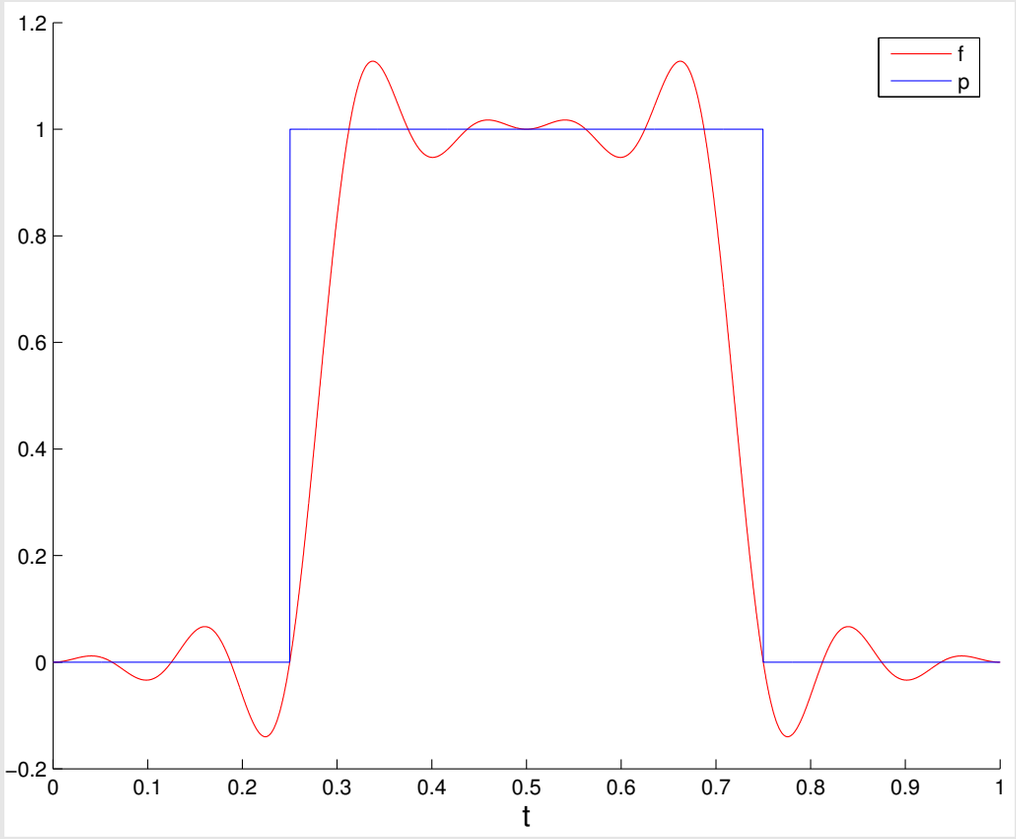
Function #3: algebraic convergence in both norms

Function #2: exponential convergence in both norms

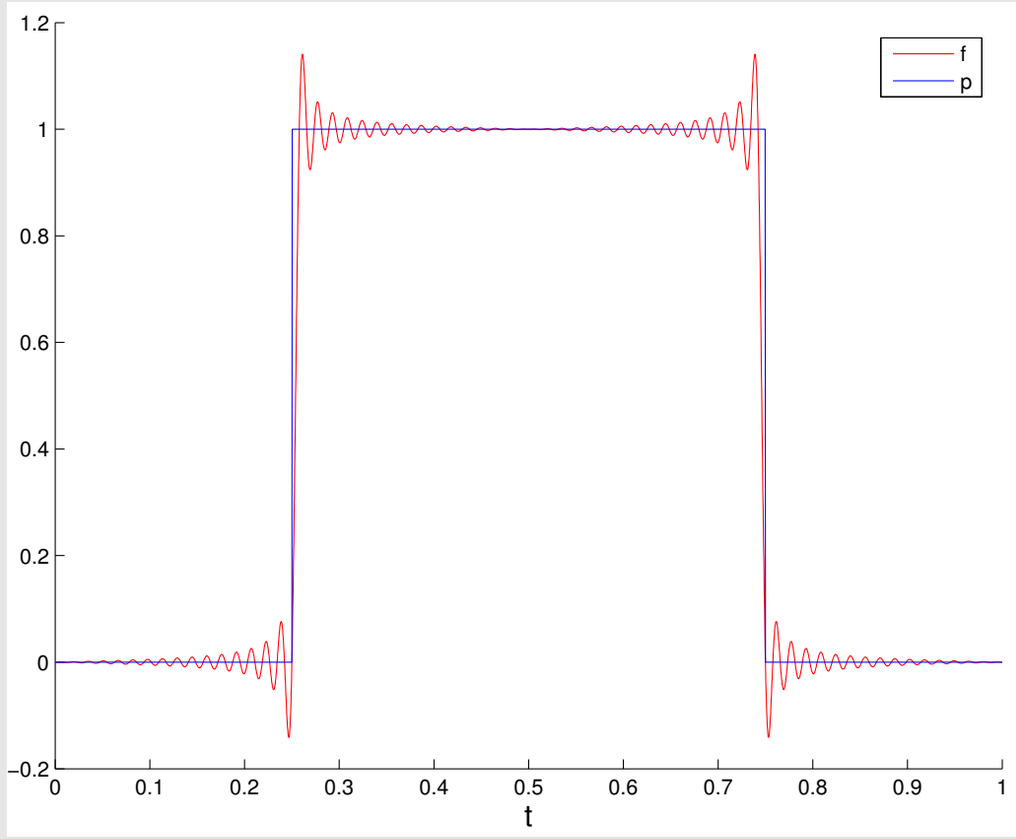
$\diamond$

Example 9.3.19 (Gibbs phenomenon).

Trigonometric interpolants of step function fail to converge in L^∞-norm in Ex. 9.3.18. A closer inspection of the interpolants shows why:



$n = 16$



$n = 128$

Observation: overshooting in neighborhood of discontinuity: Gibbs phenomenon

◇

Example 9.3.20 (Trigonometric interpolation of analytic functions).

$$f(t) = \frac{1}{\sqrt{1 - \alpha \sin(2\pi t)}} \quad \text{auf } I = [0,1] \,. \tag{9.3.21}$$

Approximative computations of error norms by "oversampling" in 4096 points.

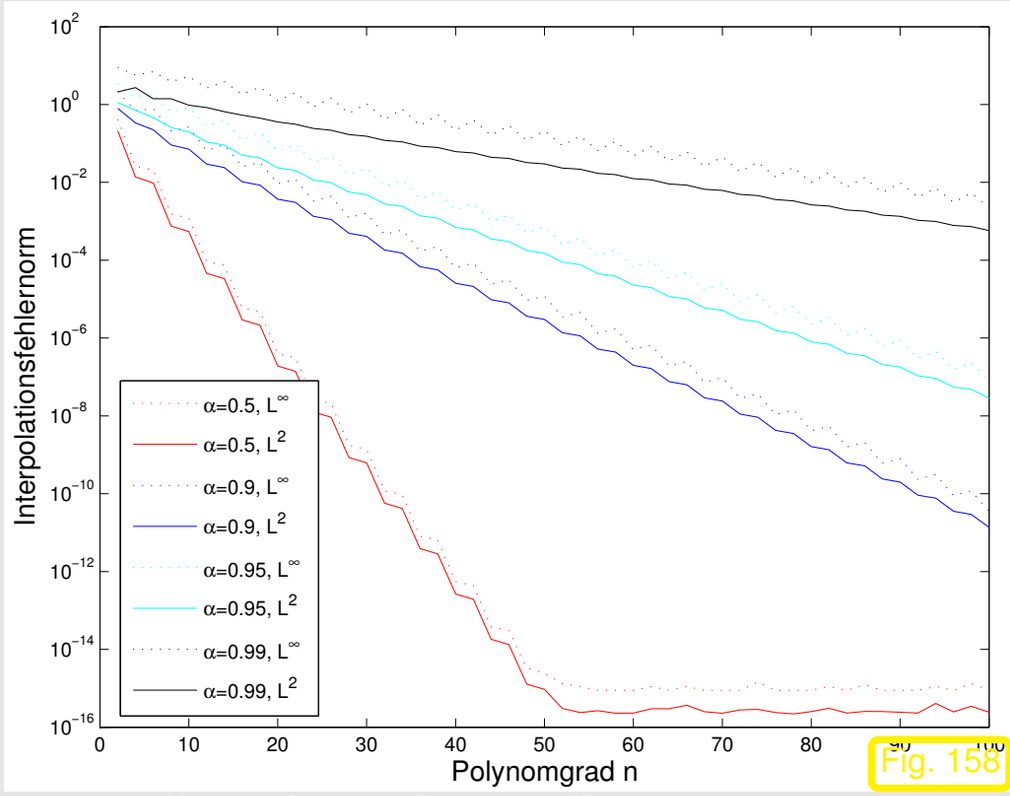


Fig. 158

➢ Observation: exponential convergence in n, faster for smaller α

Explanation, *cf.* Rem. 9.2.19:

Similar to Chebychev interpolants, also trigonometric interpolants converge exponentially fast, if the interpoland f is 1-periodic *analytic* ($\rightarrow$ Def. 9.2.20) in a strip around the real axis in $\mathbb{C}$. The speed of convergence depends on the width of the strip.

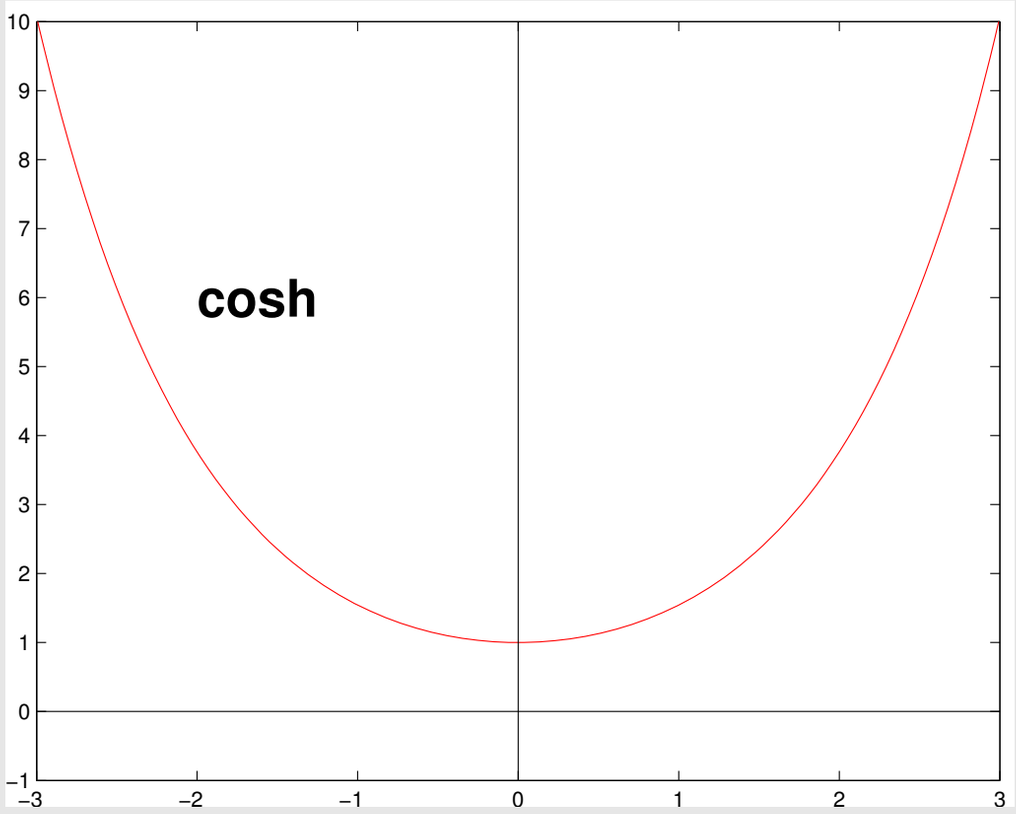
Note that f from (9.3.21) is holomorphic, where

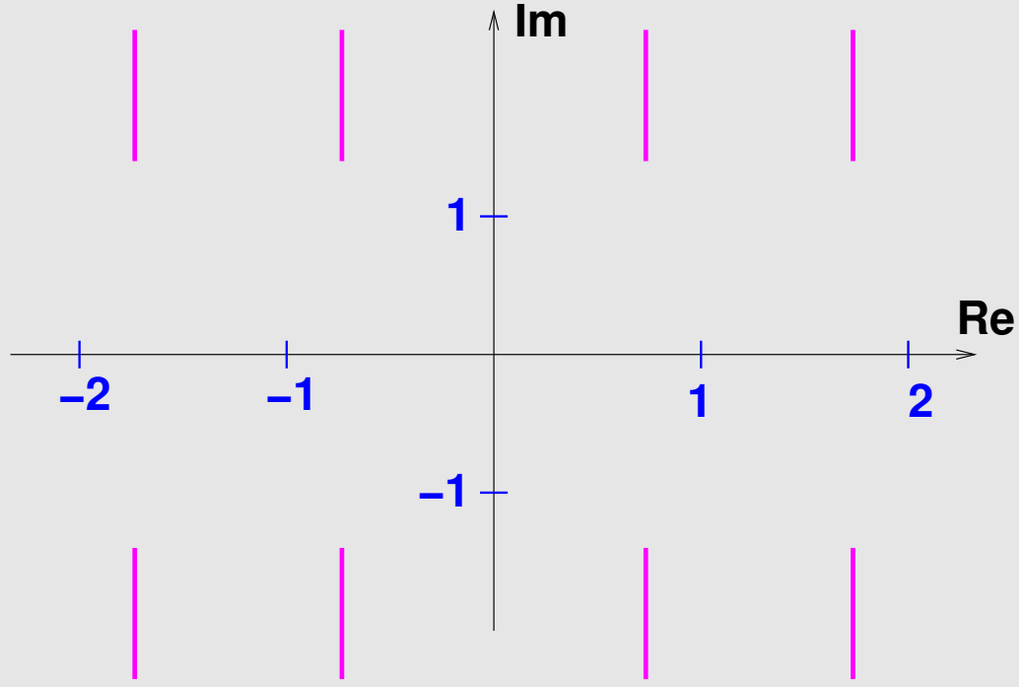
$$1+\alpha\sin(2\pi z) \notin \mathbb{R}_0^- \Leftrightarrow \sin(2\pi z) = \sin(2\pi x)\cosh(2\pi y) + i\cos(2\pi x)\sinh(2\pi y) \notin]-\infty, -1-\tfrac{1}{\alpha}] \ ,$$

because the square root is holomorphic in $\mathbb{C}\setminus\mathbb{R}_0^-$.

Domain of analyticity of f:

$$\mathbb{C}\setminus\bigcup_{k\in\mathbb{Z}}(\tfrac{k}{2}+\tfrac{1}{4}+i(\mathbb{R}\setminus]-\zeta,\zeta[)) \ , \quad \zeta\in\mathbb{R}^+ \ , \ \cosh(2\pi\zeta) = 1+\tfrac{1}{\alpha} \ .$$





➢ f analytic in strip $S := \{z\in\mathbb{C}: : -\zeta < \mathrm{Im}(z) < \zeta\}$.

R. Hiptmair

rev 30401, November 29, 2010

➢ As α decreases the strip of analyticity becomes wider, since $x \to \cosh(x)$ is increasing for $x > 0$.

9.4 Approximation by piecewise polynomials

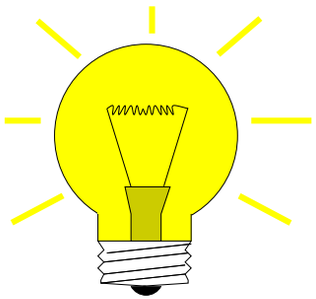
Alternatives to interpolation by global polynomials discussed in Ch. 9:

- piecewise linear/quadratic interpolation $\to$ Sect. 3.5.1, Ex. 3.5.7,
- cubic Hermite interpolation $\to$ Sect. 3.6,
- (cubic) spline interpolation $\to$ 3.7.1.

☞ All these interpolation schemes rely on piecewise polynomials (of different global smoothness)

Focus in this section: function approximation by *piecewise polynomial* interpolants

"piecewise" ➢ refer to partitioning of interval on which we aim to approximate:



Idea: use piecewise polynomials with respect to a grid/mesh

$$\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_{m-1} < x_m = b\} \tag{9.4.1}$$

to approximate function $f : [a, b] \mapsto \mathbb{R}$, $a < b$.

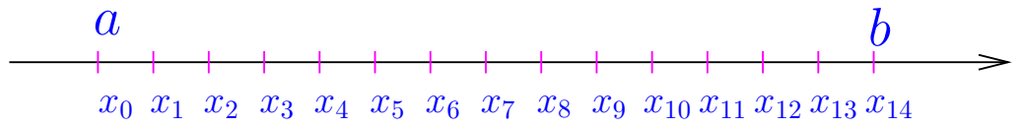
Spline terminology, *cf.* Def. 3.7.1: mesh is also called knot set.

R. Hiptmair

rev 30401, December 8, 2010

Terminology:

- $x_j \mathrel{\hat{=}}$ nodes of the mesh $\mathcal{M}$,
- $[x_{j-1}, x_j[\mathrel{\hat{=}}$ intervals/cells of the mesh,
- $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}| \mathrel{\hat{=}}$ mesh width,
- If $x_j = a + jh \mathrel{\hat{=}}$ equidistant (uniform) mesh with meshwidth $h > 0$



9.4.1 Piecewise Lagrange interpolation

Recall theory of polynomial interpolation → Sect. 3.3: $n+1$ data points needed to fix interpolating polynomial, see Thm. 3.3.6.

► Approach to *local* Lagrange interpolation (→ (3.3.1)) of $f \in C(I)$ on mesh $\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_{m-1} < x_m = b\}$:

❶ choose set of *local* interpolation points

$$\mathcal{T}^j := \{t_0^j, \ldots, t_{n_j}^j\} \subset I_j = [x_{j-1}, x_j] \,, \quad j = 1, \ldots, m \,,$$

for each grid interval, and prescribed local degree $n_j \in \mathbb{N}_0$.

❷ define piecewise interpolating polynomial $s : [x_0, x_m] \to \mathbb{K}$:

$$s_j := s_{|I_j} \in \mathcal{P}_{n_j} \quad \text{and} \quad s_j(t_i^j) = f(t_i^j) \quad i = 0, \ldots, n_j \;, \quad j = 1, \ldots, m \;. \tag{9.4.2}$$

- The mapping $f \mapsto s$ defines a linear operator $\mathsf{I}_{\mathcal{M}} : C^0([a,b]) \mapsto C^0_{\mathcal{M},\mathrm{pw}}([a,b])$.
- $\mathsf{I}_{\mathcal{M}}$ depends on $\mathcal{M}$, the local degrees n_j, and the sets $\mathcal{T}_j$ of local interpolation points (the latter two are suppressed in notation).

✎ notation: $C^0_{\mathcal{M},\mathrm{pw}}(I) \hat{=}$ space of piecewise continuous functions on interval I

Obvious: sufficient condition for global continuity of piecewise polynomial interpolant:

$$t_{n_j}^j = t_0^{j+1} \quad \forall j = 1, \ldots, m-1 \quad \Rightarrow \quad s \in C^0([a,b]) \;. \tag{9.4.3}$$

R. Hiptmair

rev 30401, December 8, 2010

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 $N := \sum_{j=1}^{m} (n_j + 1)$.

The decays of the bound $T(N)$ will characterize the type of convergence: algebraic convergence, exponential convergence, see Sect. 9.1.

But why do we choose this strange number N **?**

Because it agrees with the dimension of the space

$$\{q \in C^0_{\mathcal{M},\mathrm{pw}} \colon q_{|I_j} \in \mathcal{P}_{n_j} \; \forall j = 1, \ldots, m\} \quad !$$

this dimension tells us the number of reals we need to describe the interpolant s, that is, the "information cost" of s.

N is proportional to the number of interpolation conditions **=** number of f-evaluations needed to compute s (why only proportional in general?).

R. Hiptmair

rev 30401, December 8, 2010

Special case: uniform polynomial degree $n_j = n$ for all $j = 1, \ldots, m$.

Study estimates $\|f - \mathsf{I}_{\mathcal{M}} f\| \leq CT(h_{\mathcal{M}})$ for $h_{\mathcal{M}} \to 0$

in terms of meshwidth $h_{\mathcal{M}}$.

Terminology: study of h-convergence

Example 9.4.5 (h-convergence of piecewise polynomial interpolation).

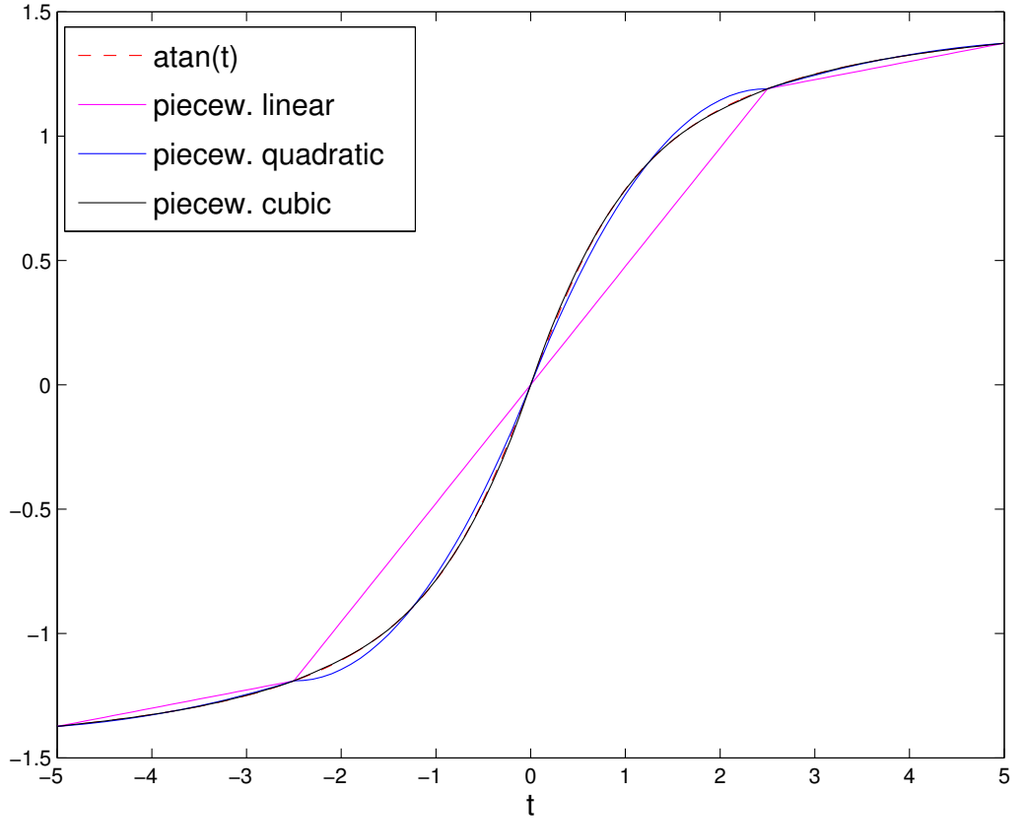
Compare Ex. 3.5.5:

$f(t) = \arctan t$, $I = [-5, 5]$

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 $\rightarrow$



- *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).

Monitored: interpolation error in (approximate) L^∞- and L^2-norms, see (9.2.18), (9.2.17)

$$\|g\|_{L^\infty([-5,5])} \approx \max_{j=0,\ldots,1000} |g(-5 + j/100)| \,,$$

R. Hiptmair

rev 30401, December 8, 2010

$$\|g\|_{L^2([-5,5])} \approx \sqrt{\frac{1}{1000}} \cdot \left(\frac{1}{2} g(-5)^2 + \sum_{j=1}^{999} |g(-5+j/100)|^2 + \frac{1}{2} g(5)^2 \right)^{1/2} .$$

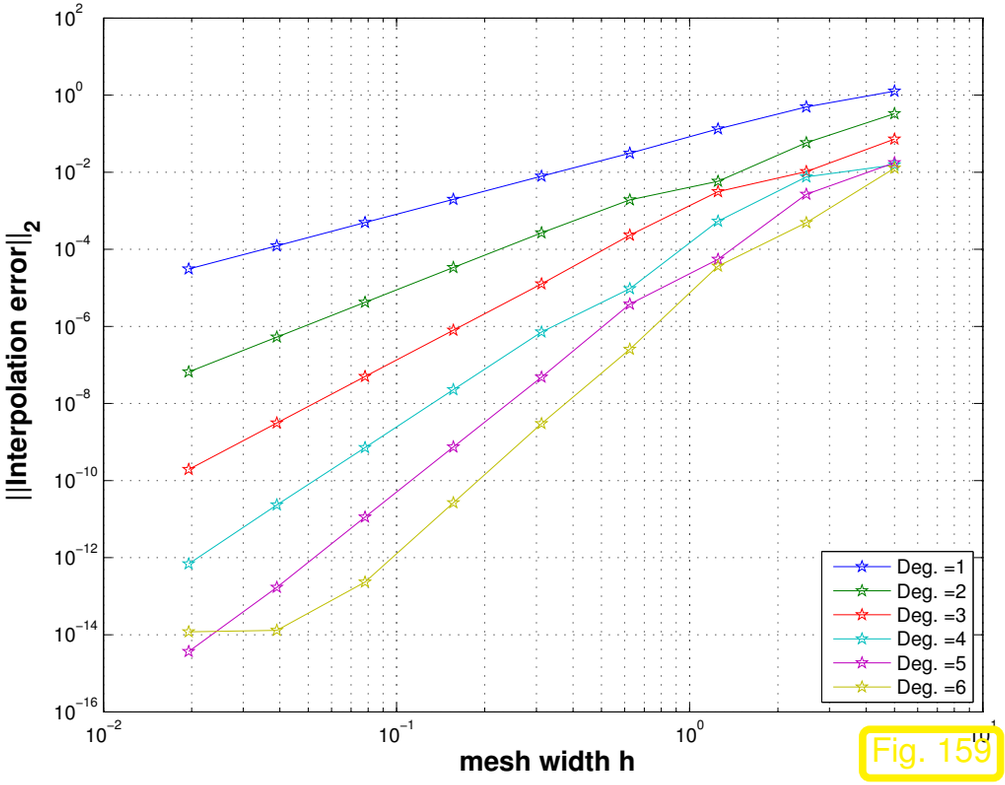


Fig. 159

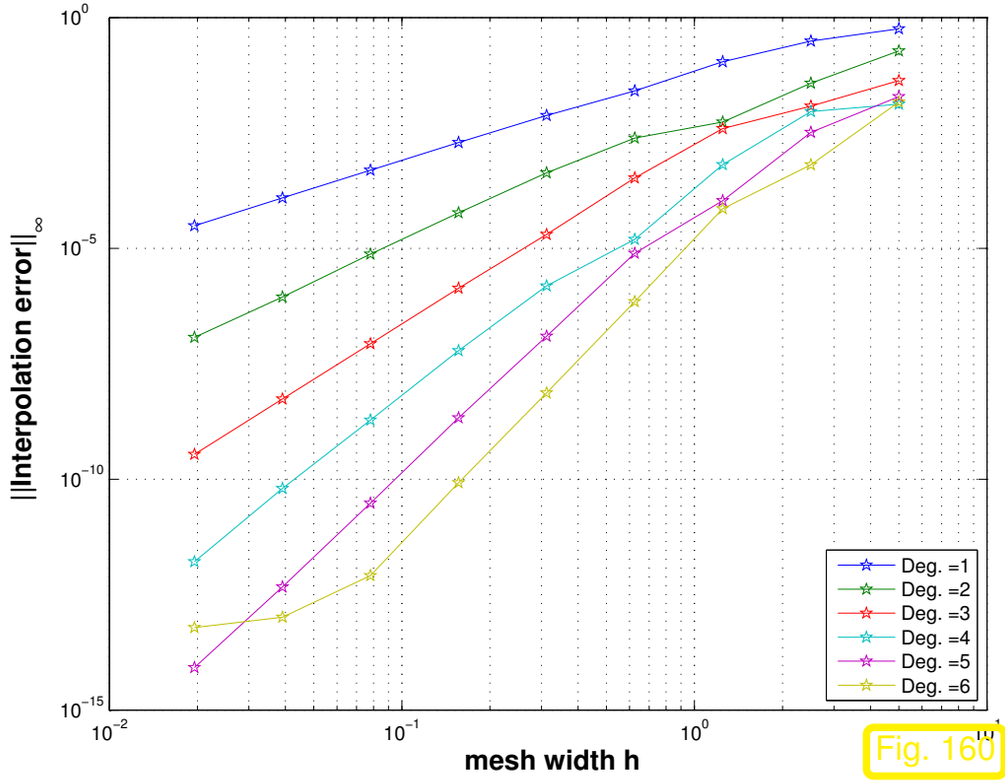


Fig. 160

Observation: Algebraic convergence in meshwidth

(nearly linear error norm graphs in doubly logarithmic scale, see Rem. 9.1.4)

R. Hiptmair

rev 30401, December 8, 2010

Observation: rate of algebraic convergence increases with polynomial degree n

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

➢ Higher polynomial degree provides faster algebraic decrease of interpolation error norms.

Here: rates estimated by linear regression ($\rightarrow$ Ex. 7.0.9) based on MATLAB's `polyfit` and the interpolation errors for meshwidth $h \leq 10 \cdot 2^{-5}$. This was done in order to avoid erratic "preasymptotic", that is, for large meshwidth h, behavior of the error.

The bad rates for $n = 6$ are due to the impact of roundoff, because the norms of the interpolation error had dropped below machine precision, see Figs. 159, 160.

These observations are easily explained by applying the polynomial interpolation error estimates of Sect. 9.1 *locally* on the mesh intervals: for constant polynomial degree $n = n_j$, $j = 1, \ldots, m$

$$\text{(9.1.11)} \quad \Rightarrow \quad \|f - s\|_{L^\infty([x_0,x_m])} \leq \frac{h^{n+1}}{(n+1)!} \left\| f^{(n+1)} \right\|_{L^\infty([x_0,x_m])} \quad , \qquad \text{(9.4.6)}$$

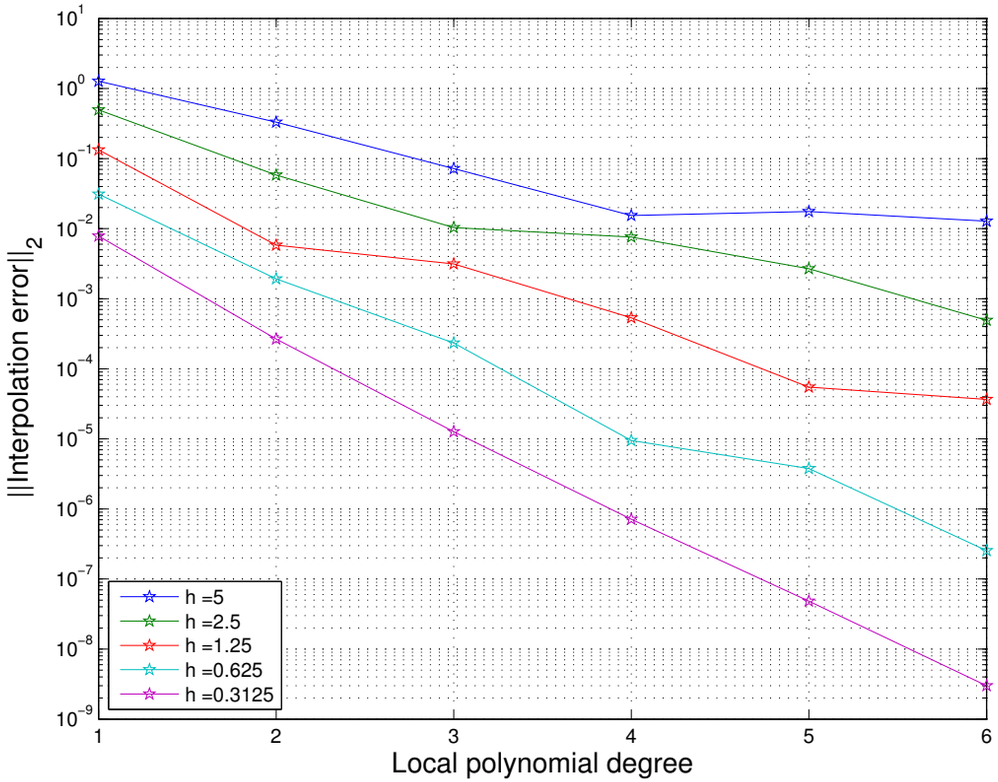
with mesh width $h := \max\{|x_j - x_{j-1}| : j = 1, \ldots, m\}$.

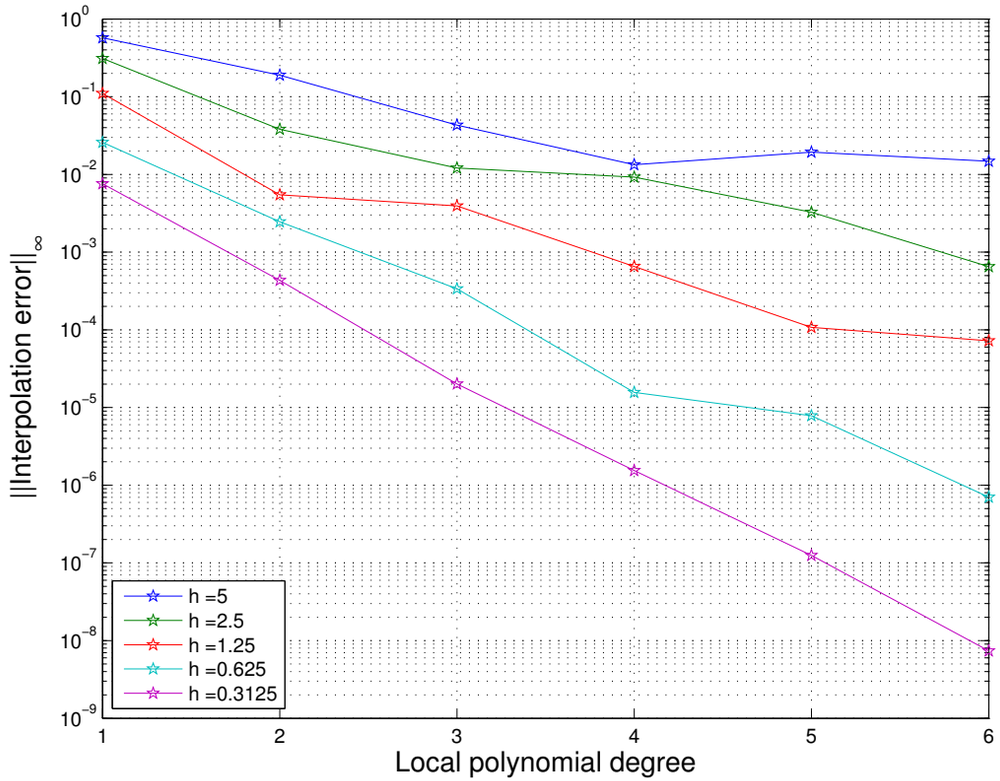
Another special case: fixed mesh $\mathcal{M}$, uniform polynomial degree n

Study estimates $\|f - \mathsf{I}_{\mathcal{M}} f\| \leq CT(n)$ for $n \to \infty$.

Terminology: investigation of p-convergence

Example 9.4.7 (p-convergence of piecewise polynomial interpolation).





Observation: (apparent) exponential convergence in polynomial degree

R. Hiptmair

rev 30401, December 8, 2010

Note: in the case of p-convergence the situation is the same as for standard polynomial interpolation, see 9.1.

In this example we deal with an analytic function, see Rem. 9.2.19. Though equidistant local interpolation nodes are used *cf.* Ex. 9.1.5, the mesh intervals seems to be small enough that even in this case exponential convergence prevails.

◇

9.4.2 Cubic Hermite iterpolation: error estimates

See Sect. 3.6 for definition and algorithms for cubic Hermite interpolation of data points.

Example 9.4.8 (Convergence of Hermite interpolation with exact slopes)*.*

Piecewise cubic Hermite interpolation of

$$f(x) = \arctan(x) \ .$$

- domain: $I = (-5, 5)$
- mesh $\mathcal{T} = \{-5 + hj\}_{j=0}^{n} \subset I$, $h = \frac{10}{n}$,
- exact slopes $c_i = f'(t_i)$, $i = 0, \ldots, n$

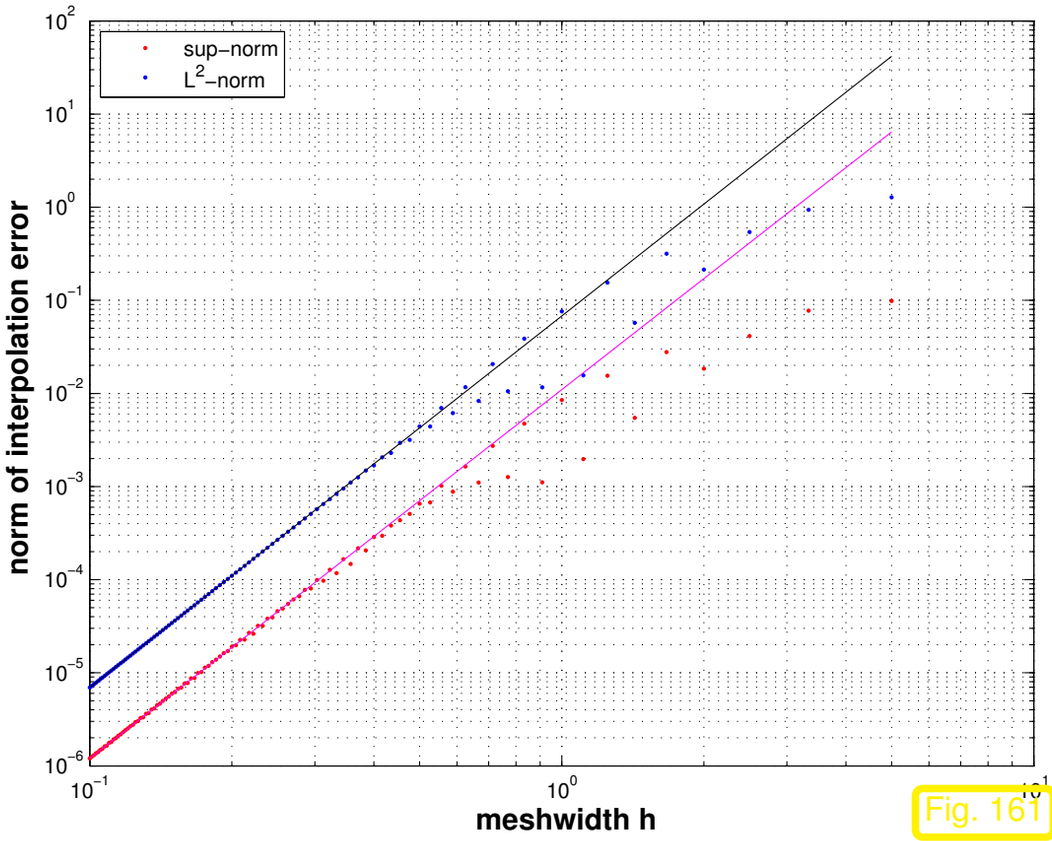


Fig. 161

► algebraic convergence $O(h^4)$

Approximate computation of error norms analoguous to Ex. 9.4.5.

Code 9.4.9: Hermite approximation and orders of convergence with exact slopes

```
1 function hermiteapprox1(f,df,a,b,N)
```

R. Hiptmair

rev 30401, December 8, 2010

```
% Investigation of interpolation error norms for cubic Hermite interpolation of
  f (handle f)
% on [a,b] with slopes given by f' (function handle df).
% N gives the maximum number of mesh intervals
err = [];
for j=2:N
  xx=a; % xx is the fine mesh on which the error norms are computed
  val=f(a); % function values of mesh xx

  t = a:(b-a)/j:b; % mesh nodes
  y = f(t); c=df(t); % function values and slopes provide coefficients

  for k=1:j-1
    vx = linspace(t(k),t(k+1), 100);
    % See Code 3.6.2 for local evaluation of Hermite interpolant
    locval=hermloceval(vx,t(k),t(k+1),y(k),y(k+1),c(k),c(k+1));
    xx=[xx, vx(2:100)];
    val=[val,locval(2:100)];
  end
  d = abs(feval(f,xx)-val); % pointwise error on mesh xx
  h = (b-a)/j;
  % compute L^2 norm of the error using trapezoidal rule
  l2 = sqrt(h*(sum(d(2:end-1).^2)+(d(1)^2+d(end)^2)/2) );
  % columns of err = meshwidth, sup-norm error, L^2 error:
  err = [err; h, max(d),l2];
end

figure('Name','Hermite approximation');
loglog(err(:,1),err(:,2),'r.',err(:,1),err(:,3),'b.');
```


R. Hiptmair

rev 30401, December 8, 2010

```
30 grid on;
31 xlabel('{\bf meshwidth h}','fontsize',14);
32 ylabel('{\bf norm of interpolation error}','fontsize',14);
33 legend('sup-norm','L^2-norm','location','northwest');
34
35 % compute estimates for algebraic orders of convergence
36 % using linear regression on half of the data points
37 pI = polyfit(log(err(ceil(N/2):N-2,1)),log(err(ceil(N/2):N-2,2)),1);
     exp_rate_Linf=pI(1)
38 pL2 = polyfit(log(err(ceil(N/2):N-2,1)),log(err(ceil(N/2):N-2,3)),1);
     exp_rate_L2=pL2(1)
39 hold on;
40 plot([err(1,1),err(N-1,1)], [err(1,1),err(N-1,1)].^pI(1)*exp(pI(2)),'m')
41 plot([err(1,1),err(N-1,1)], [err(1,1),err(N-1,1)].^pL2(1)*exp(pL2(2)),'k')
42
43 print -depsc2 '../PICTURES/hermiperrslopes.eps';
```

Try: `hermiteapprox1(@atan, @(x) 1./(1+x.^2), -5,5,100);`

◇

Example 9.4.10 (Convergence of Hermite interpolation with averaged slopes).

Piecewise cubic Hermite interpolation of

$$f(x) = \arctan(x) \ .$$

- domain: $I = (-5, 5)$
- equidistant mesh $\mathcal{T}$ in I, see Ex. 9.4.8,
- averaged local slopes, see (3.6.4)

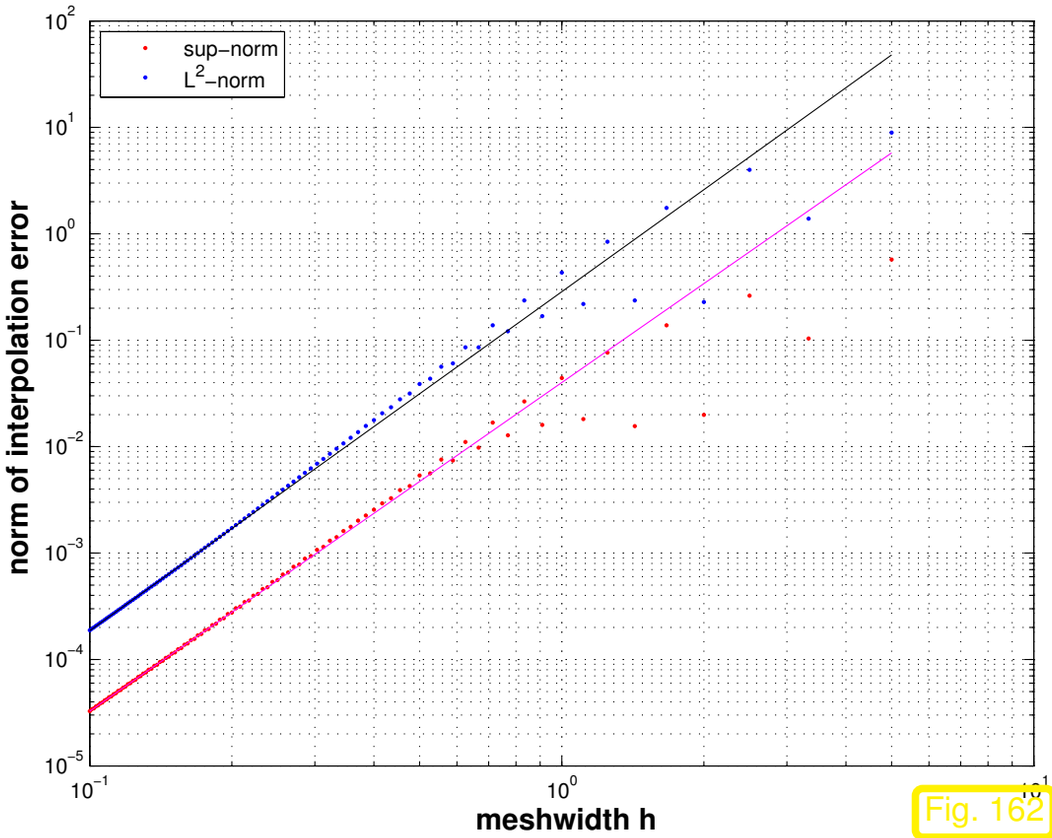


Fig. 162

▶ algebraic convergence $O(h^3)$ in meshwidth

(see Code 9.4.10)

Lower order of convergence due to the choice of the slopes (3.6.4):
from the plot L^∞-norm $\sim$ L^2-norm $\sim O(h^3)$

◇

Code 9.4.11: Hermite approximation and orders of convergence

```
% 16.11.2009 hermiteapprox.m
% Plot convergence of approximation error of cubic hermite interpolation
% with respect to the meshwidth
% print the algebraic order of convergence in sup and L^2 norms
% Slopes:  weighted average of local slopes
%
% inputs:  f function to be interpolated
% a, b left and right extremes of the interval
% N maximum number of subdomains

function hermiteapprox(f,a,b,N)
% Investigation of interpolation error norms for cubic Hermite interpolation of
  f (handle f)
% on [a,b] with linearly averaged slopes according to (3.6.4).
% N gives the maximum number of mesh intervals
err = [];
for j=2:N
  xx=a;      % xx is the fine mesh on which the error norms are computed
  val=f(a); % function values on xx

  t = a:(b-a)/j:b;      % mesh nodes
  y = f(t); c=slopes1(t,y); % coefficients for Hermit polynomial representation

  for k=1:j-1
    vx = linspace(t(k),t(k+1), 100);
    locval=hermloceval(vx,t(k),t(k+1),y(k),y(k+1),c(k),c(k+1));
    xx=[xx, vx(2:100)];
    val=[val,locval(2:100)];
```

R. Hiptmair

rev 30401, December 8, 2010

```
  end
  d = abs(feval(f,xx)- val);
  h = (b-a)/j;
  % compute L^2 norm of the error using trapezoidal rule
  l2 = sqrt(h*(sum(d(2:end-1).^2)+(d(1)^2+d(end)^2)/2) );
  % columns of err = meshwidth, sup-norm error, L^2 error:
  err = [err; h,max(d),l2];
end

figure('Name','Hermite approximation');
loglog(err(:,1),err(:,2),'r.',err(:,1),err(:,3),'b.');
grid on;
xlabel('{\bf meshwidth h}','fontsize',14);
ylabel('{\bf norm of interpolation error}','fontsize',14);
legend('sup-norm','L^2-norm','location','northwest');

% compute estimates for algebraic orders of convergence
% using linear regression on half of the data points
pI = polyfit(log(err(ceil(N/2):N-2,1)),log(err(ceil(N/2):N-2,2)),1);
  exp_rate_Linf=pI(1)
pL2 = polyfit(log(err(ceil(N/2):N-2,1)),log(err(ceil(N/2):N-2,3)),1);
  exp_rate_L2=pL2(1)
hold on;
plot([err(1,1),err(N-1,1)], [err(1,1),err(N-1,1)].^pI(1)*exp(pI(2)),'m')
plot([err(1,1),err(N-1,1)], [err(1,1),err(N-1,1)].^pL2(1)*exp(pL2(2)),'k')

print -depsc2 '../PICTURES/hermiperravgsl.eps';

%------------------------------
```


R. Hiptmair

rev 30401, December 8, 2010

```
55 function     c=slopes1(t,y)
56 h = diff(t);                                    % increments in t
57 delta = diff(y)./h;                             % slopes of piecewise linear
     interpolant
58 c = [delta(1),...
59      ((h(2:end).*delta(1:end-1)+h(1:end-1).*delta(2:end))...
60         ./(t(3:end) - t(1:end-2)) ),...
61      delta(end)];
```

Try: `hermiteapprox(@atan, -5,5,100);`

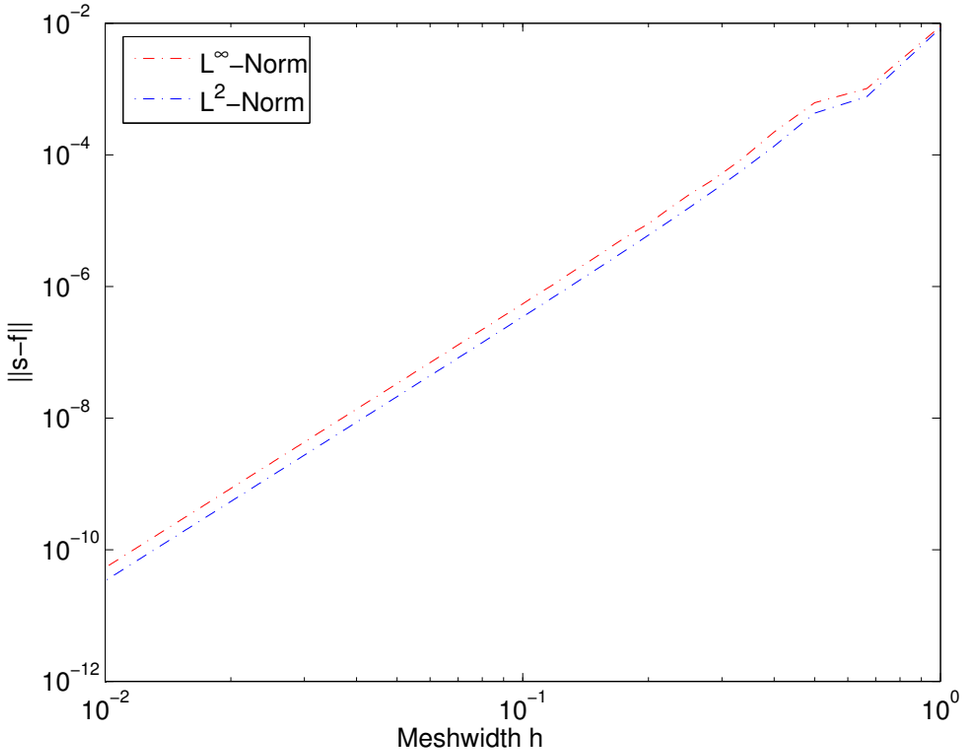
9.4.3 Cubic spline interpolation: error estimates [29, Ch. 47]

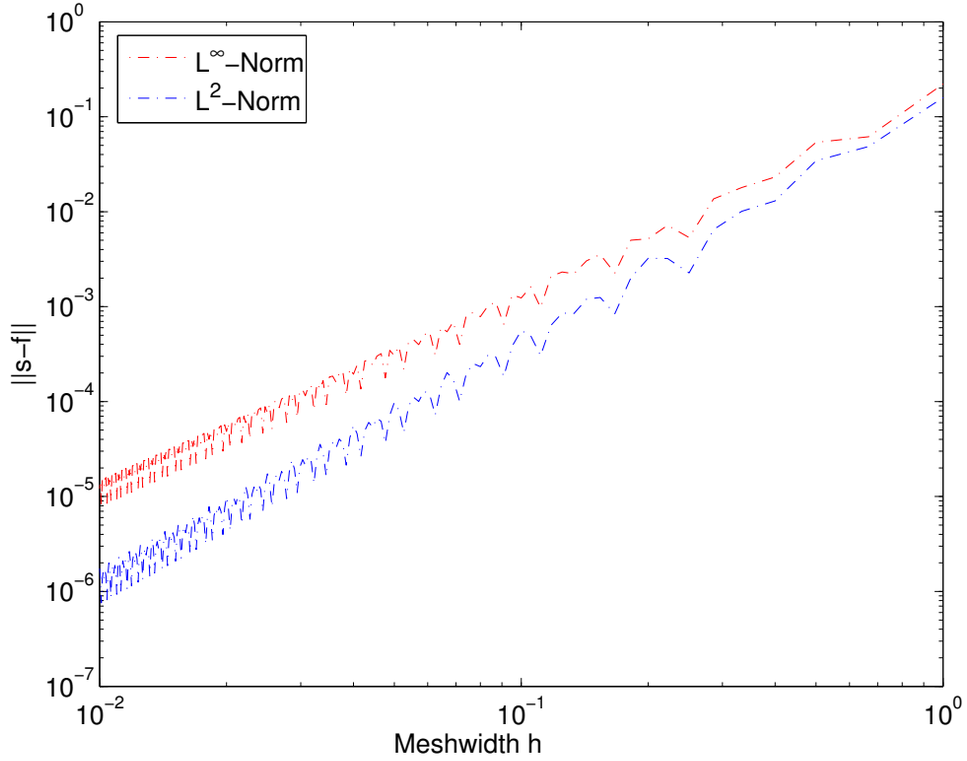
Recall: concept and algorithms for cubic spline interpolation from Sect. 3.7.1.

Example 9.4.12 (Approximation by complete cubic spline interpolants).

Grid (knot set) $\mathcal{M} := \{-1 + \frac{2}{n}j\}_{j=0}^{n}$, $n \in \mathbb{N}$ ➙ meshwidth $h = 2/n$, $I = [-1,1]$

$$f_1(t) = \frac{1}{1+e^{-2t}} \in C^\infty(I) \quad , \quad f_2(t) = \begin{cases} 0 & \text{, if } t < -\frac{2}{5} \text{ ,} \\ \frac{1}{2}(1+\cos(\pi(t-\frac{3}{5}))) & \text{, if } -\frac{2}{5} < t < \frac{3}{5} \text{ , } \in C^1(I) \text{ .} \\ 1 & \text{otherwise.} \end{cases}$$





$\|f_1 - s\|_{L^\infty([-1,1])} = O(h^4)$ $\|f_2 - s\|_{L^\infty([-1,1])} = O(h^2)$

Algebraic order of convergence in h = min { 1 + regularity of f, 4 }.

Theory [28]: $f \in C^4([t_0,t_n])$ ➙ $\|f - s\|_{L^\infty([t_0,t_n])} \leq \frac{5}{384} h^4 \left\| f^{(4)} \right\|_{L^\infty([t_0,t_n])}$

R. Hiptmair

rev 30401, December 8, 2010

See also [10, Rem. 9.2],

◇

Code 9.4.13: Spline approximation error

```
function splineapprox(f,df,a,b,N)
% Study of interpolation error norms for complete cubic spline interpolation of f
% on equidistant knot set.
x = a:0.00025:b;  fv = feval(f,x); % fine mesh for norm evaluation
dfa = feval(df,a); dfb = feval(df,b); % Slopes at endpoints
err = [];
for j=2:N
  t = a:(b-a)/j:b;           % spline knots
  y = [dfa,feval(f,t),dfb];
  % compute complete spline imposing exact first derivative at the endpoints
  % Please refer to MATLAB documentation of spline
  v = spline(t,y,x);
  d = abs(fv-v);
  h = x(2:end)-x(1:end-1);
```

```
  % compute L^2 norm of the error using trapezoidal rule on mesh x
  l2 = sqrt(0.5*dot(h,(d(1:end-1).^2+d(2:end).^2)));
  % columns of err = meshwidth, L^infty error, L^2 error:
  err = [err; (b-a)/j,max(d),l2];
end

figure('Name','Spline interpolation');
plot(t,y(2:end-1),'m*',x,fv,'b-',x,v,'r-');
xlabel('{\bf t}','fontsize',14);  ylabel('{\bf s(t)}','fontsize',14);
legend('Data points','f','Cubic spline interpolant','location','best');

figure('Name','Spline approximation error');
loglog(err(:,1),err(:,2),'r.-',err(:,1),err(:,3),'b.-');
grid on;  xlabel('Meshwidth h');     ylabel('||s-f||');
legend('sup-norm','L^2-norm', 'Location','NorthWest');
%compute algebraic orders of convergence using polynomial fit
p = polyfit(log(err(:,1)),log(err(:,2)),1); exp_rate_Linf=p(1)
p = polyfit(log(err(:,1)),log(err(:,3)),1); exp_rate_L2=p(1)
```

R. Hiptmair

rev 30401, December 8, 2010

Try:

```
splineapprox(@atan, @(x) 1./(1+x.^2), -5,5,100);
splineapprox(@(x) 1./(1+exp(-2*x)),...
             @(x) 2*exp(-2*x)./(1+exp(-2*x)).^2, -1,1,100);
```

10 Numerical Quadrature [29, VII], [10, Ch. 10]

Numerical quadrature

= Approximate numerical evaluation of $\int_\Omega f(\mathbf{x})\,\mathrm{d}\mathbf{x}$, integration domain $\Omega \subset \mathbb{R}^d$

Setting: continuous function $f : \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}$ only available as

```
function y = f(x)
```

(point evaluation, available as *possibly expensive* "black-box" function)

Here: focus on special case $d = 1$: $\Omega = [a, b]$ (interval)

R. Hiptmair

rev 30401, December 9, 2010

Remark 10.0.1 (Importance of numerical quadrature).

☞ Numerical quadrature methods are key building blocks for methods for the numerical treatment of partial differential equations (➢ Course “Numerical treatment of partial differential equations”)

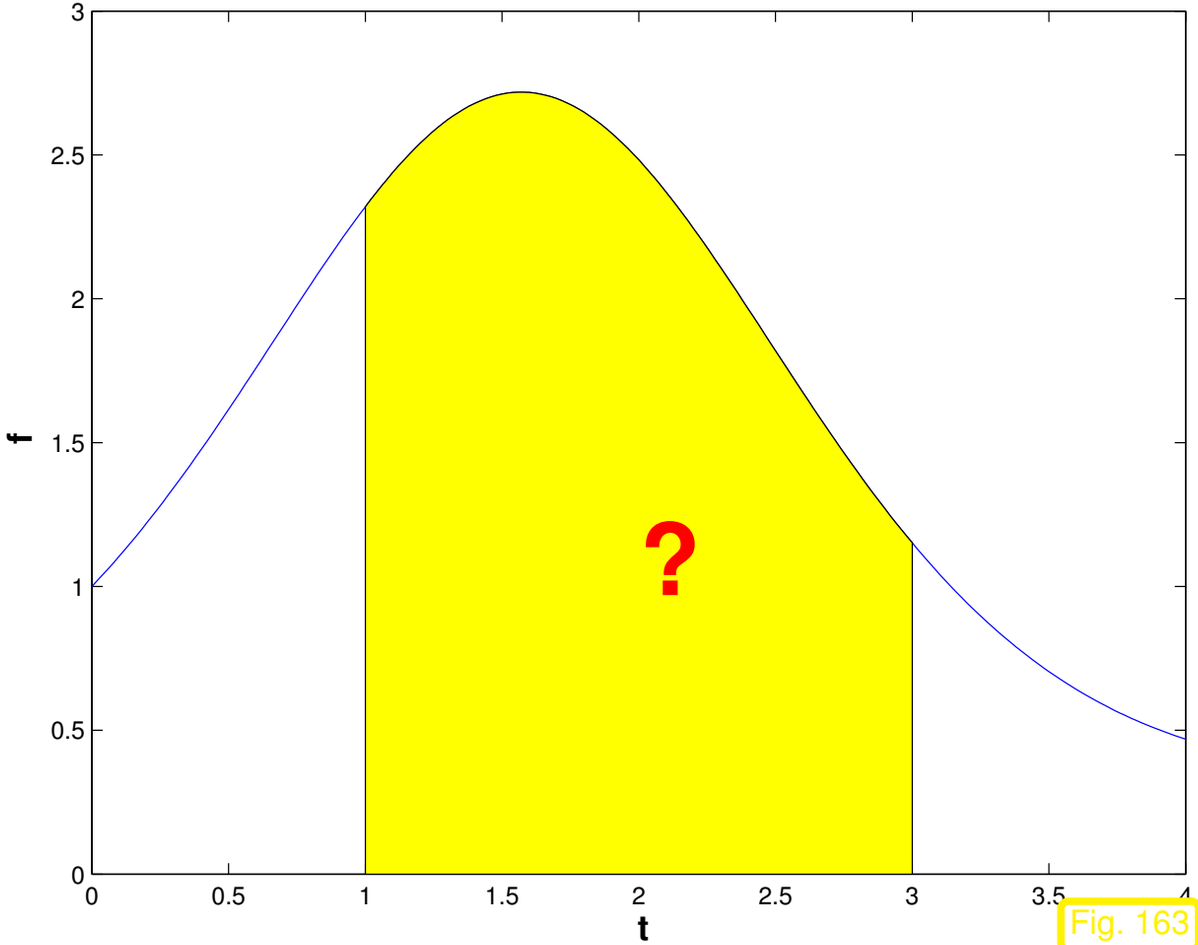


Fig. 163

Numerical quadrature methods

approximate

$$\int_a^b f(t)\,\mathrm{d}t$$

R. Hiptmair

rev 30401, December 9, 2010

Example 10.0.2 (Heating production in electrical circuits).

Time-harmonic excitation:

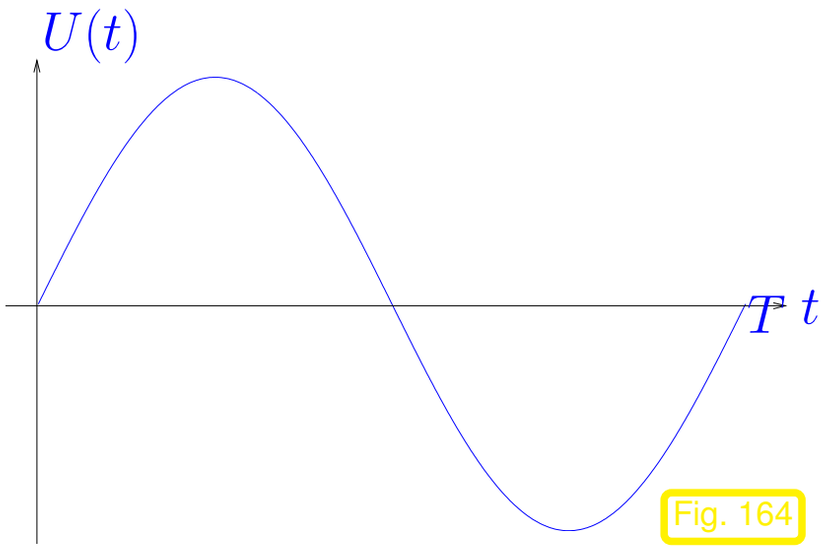


Fig. 164

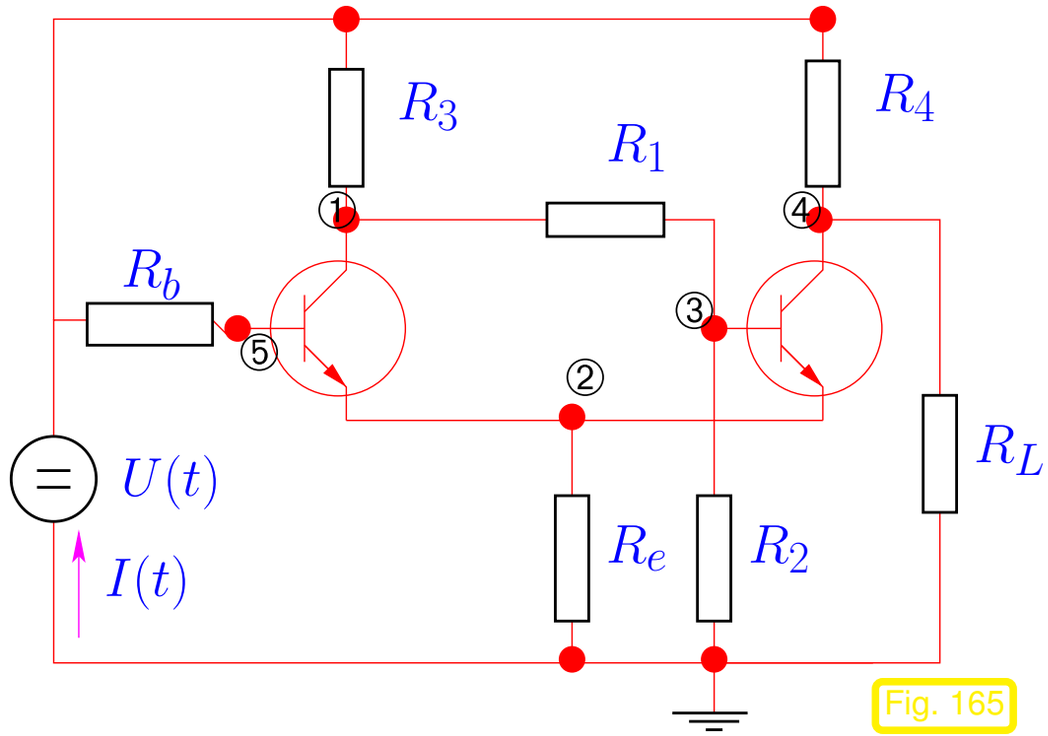


Fig. 165

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 30401, 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)

Note: (10.1.1) compatible with integrand f given in procedural form as `function y = f(x)`.

Code 10.1.2: MATLAB template implementing generic quadrature formula

```
function I = quadformula(f,c,w)
% Generic numerical quadrature routine implementing (10.1.1):
% f is a handle to a function of type f = @(x) ....
% c, w pass quadrature nodes c_j ∈ [a,b], and weights w_j > 0
n = length(c); If (length(w) ~= n), error('#weights != #nodes');
  end
I = 0; for j=1:n, I = I + w(j)*f(c(j)); end
```

R. Hiptmair

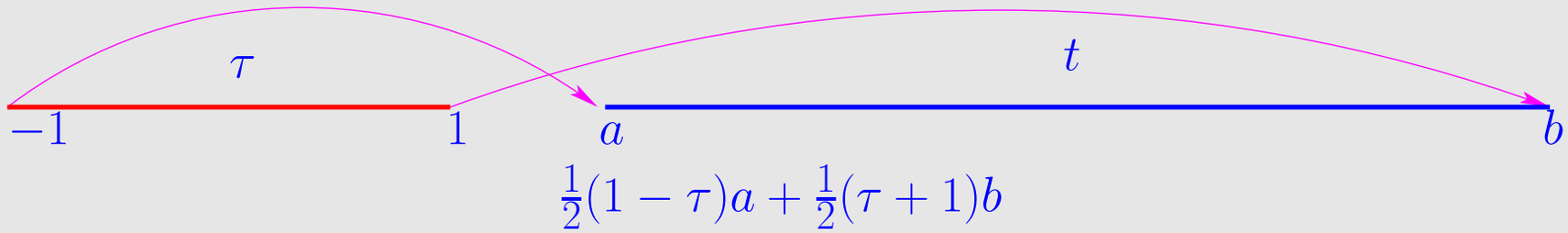
rev 30401, December 9, 2010

Remark 10.1.3 (Transformation of quadrature rules).

Given: quadrature formula $(\widehat{c}_j, \widehat{w}_j)_{j=1}^n$ on reference interval $[-1,1]$

Idea: transformation formula for integrals

$$\int_a^b f(t)\,\mathrm{d}t = \tfrac{1}{2}(b-a)\int_{-1}^1 \widehat{f}(\tau)\,\mathrm{d}\tau\;, \quad \widehat{f}(\tau) := f(\tfrac{1}{2}(1-\tau)a + \tfrac{1}{2}(\tau+1)b)\;. \tag{10.1.4}$$



quadrature formula for general interval $[a,b]$, $a,b \in \mathbb{R}$:

$$\int_a^b f(t)\,\mathrm{d}t \approx \tfrac{1}{2}(b-a)\sum_{j=1}^n \widehat{w}_j \widehat{f}(\widehat{c}_j) = \sum_{j=1}^n w_j f(c_j) \quad \text{with} \quad \begin{aligned} c_j &= \tfrac{1}{2}(1-\widehat{c}_j)a + \tfrac{1}{2}(1+\widehat{c}_j)b\;, \\ w_j &= \tfrac{1}{2}(b-a)\widehat{w}_j\;. \end{aligned}$$

R. Hiptmair

rev 30401, December 9, 2010

► A 1D quadrature formula on arbitrary intervals can be specified by providing its weights $\widehat{w}_j$ /nodes $\widehat{c}_j$ for integration domain $[-1,1]$. Then the above transformation is assumed.

Other common choice of reference interval: $[0,1]$

△

In general the quadrature formula (10.1.1) will only provide an approximate value for the integral.

➢ Inevitable for generic integrand:

quadrature error $$E(n) := \left| \int_a^b f(t)\,\mathrm{d}t - Q_n(f) \right|$$

Our focus (*cf.* interpolation error estimates, Sect. 9.1):

given families of quadrature rules $\{Q_n\}_n$ described by

- quadrature weights $\left\{w_j^n,\ j=1,\ldots,n\right\}_{n\in\mathbb{N}}$ and
- quadrature nodes $\left\{c_j^n,\ j=1,\ldots,n\right\}_{n\in\mathbb{N}}$, 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 $E(n)=O(q^n)$, $0\le q<1$

Note that the number n of nodes agrees with the number of f-evaluations required for evaluation of the quadrature formula. This is usually used as a *measure for the cost* of computing $Q_n(f)$.

Therefore we consider the quadrature error as a function of n.

10.2 Polynomial Quadrature Formulas [10, Sect. 10.2]

Remark 10.2.1 (Approximation and quadrature).

There is a close relationship

$$\text{function approximation} \quad \longrightarrow \quad \text{numerical quadrature}$$

because, once we know a good approximation of the integrand as a linear combination of basis functions,

$$f(t) \approx \sum_{j=1}^{m} \gamma_j b_j(t) \;, \quad b_j(t) \mathrel{\hat{=}} \text{basis functions, see (3.1.4),}$$

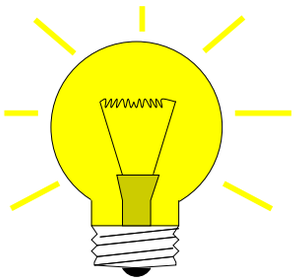
we can expect that

$$\sum_{j=1}^{m} \gamma_j \int_a^b b_j(t)\,\mathrm{d}t \approx \int_a^b f(t)\,\mathrm{d}t \tag{10.2.2}$$

provides a good approximation for the integral.

If the coefficients γ_j are computed relying on point evaluations, then (10.2.2) boils down to a quadrature formula.

We elaborate this idea for function approximation by Lagrangian interpolation polybnomials, see Sect. 3.3 and Sect. 9.1.



Idea: replace integrand f with $p_{n-1} \in \mathcal{P}_{n-1}$ = polynomial interpolant of f for given interpolation nodes $\{t_0, \ldots, t_{n-1}\} \subset [a,b]$

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \int_a^b p_{n-1}(t)\,\mathrm{d}t\,. \tag{10.2.3}$$

Lagrange polynomials: $L_i(t) := \prod_{\substack{j=0 \\ j\neq i}}^{n-1} \frac{t-t_j}{t_i-t_j}\,, \quad i=0,\ldots,n-1$ (3.3.5) $p_{n-1}(t) = \sum_{i=0}^{n-1} f(t_i)L_i(t)\,.$

R. Hiptmair

rev 30401, December 8, 2010

$$\int_a^b p_{n-1}(t)\,\mathrm{d}t = \sum_{i=0}^{n-1} f(t_i) \int_a^b L_i(t)\,\mathrm{d}t \quad \blacktriangleright \quad \begin{aligned} &\text{nodes} \quad c_i = t_{i-1}\,, \\ &\text{weights} \quad w_i := \int_a^b L_{i-1}(t)\,\mathrm{d}t\,. \end{aligned} \tag{10.2.4}$$

Example 10.2.5 (Midpoint rule).

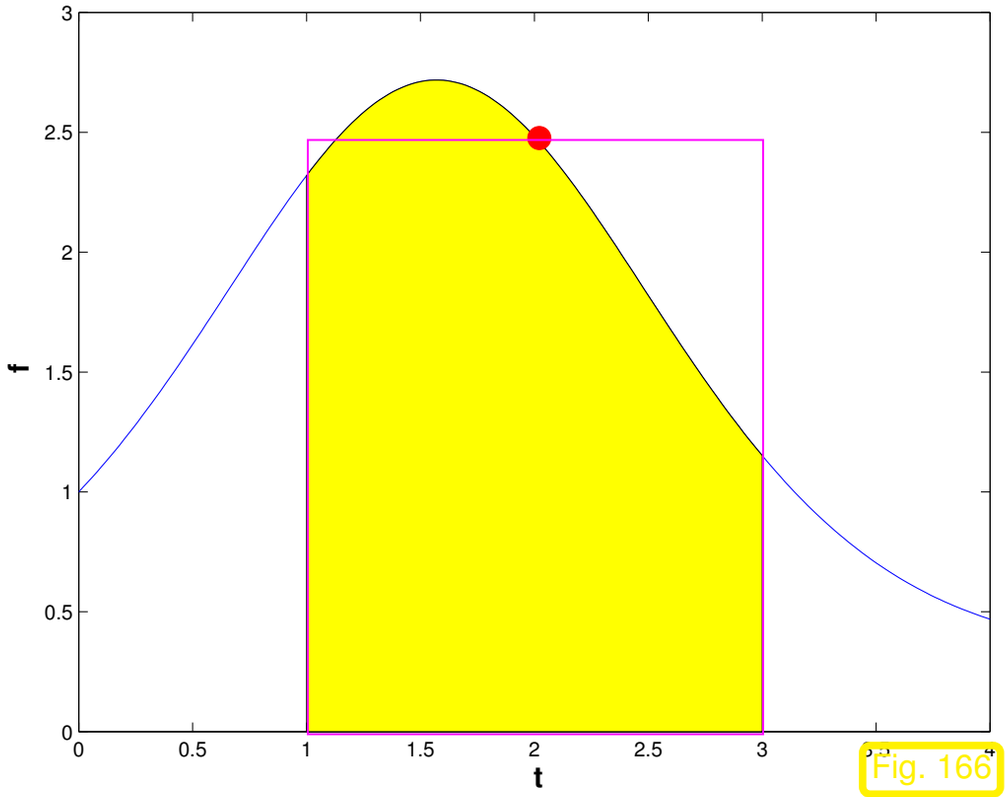


Fig. 166

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). [29, Ch. 38]

R. Hiptmair

rev 30401, December 8, 2010

Equidistant quadrature nodes $t_j := a + hj, \quad h := \frac{b-a}{n}, \quad j = 0, \ldots, n$:

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);
```

$$\widehat{Q}_{\mathrm{trp}}(f) := \frac{1}{2}(f(0) + f(1)) \qquad \text{(10.2.7)}$$

$$\left(\int_a^b f(t)\,\mathrm{d}t \approx \frac{b-a}{2}(f(a) + f(b)) \right)$$

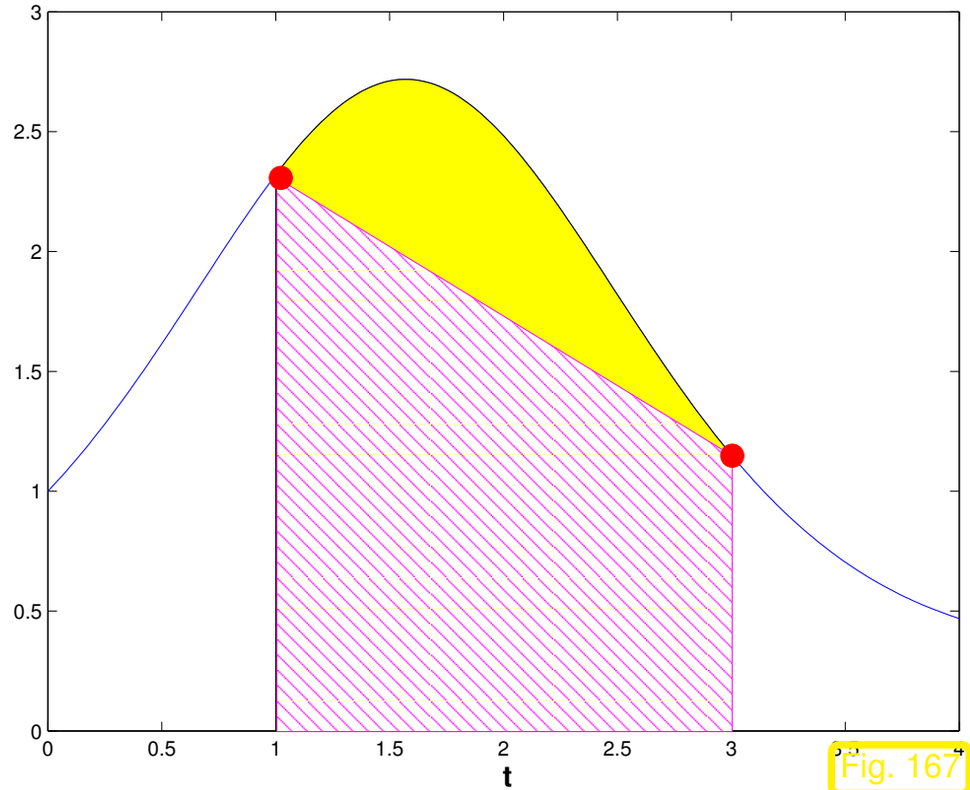


Fig. 167

R. Hiptmair

rev 30401, December 8, 2010

- $n = 2$: Simpson rule

```
> simpson := newtoncotes(2);
```

$$\frac{h}{6}\left(f(0)+4\,f(\tfrac{1}{2})+f(1)\right) \quad \left(\int_a^b f(t)\,\mathrm{d}t \approx \frac{b-a}{6}\left(f(a)+4\,f\left(\frac{a+b}{2}\right)+f(b)\right)\right) \qquad \text{(10.2.8)}$$

- $n = 4$: Milne rule

```
> milne := newtoncotes(4);
```

$$\frac{1}{90}\,h\left(7\,f(0)+32\,f(\tfrac{1}{4})+12\,f(\tfrac{1}{2})+32\,f(\tfrac{3}{4})+7\,f(1)\right)$$

$$\left(\frac{b-a}{90}\,(7f(a)+32f(a+(b-a)/4)+12f(a+(b-a)/2)+32f(a+3(b-a)/4)+7f(b))\right)$$

- $n = 6$: Weddle rule

```
> weddle := newtoncotes(6);
```

$$\frac{1}{840}\,h\,(41\,f(0)+216\,f(\tfrac{1}{6})+27\,f(\tfrac{1}{3})+272\,f(\tfrac{1}{2}) \\ +27\,f(\tfrac{2}{3})+216\,f(\tfrac{5}{6})+41\,f(1))$$

- $n \geq 8$: 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 ($\rightarrow$ Def. 2.5.11) **!**

Alternative: If t_j = Chebychev nodes (9.2.4) ➤ Clenshaw-Curtis rule

Remark 10.2.9 (Error estimates for polynomial quadrature).

Quadrature error estimates directly from L^∞-interpolation error estimates for Lagrangian interpolation with polynomial of degree $n-1$, see Thm. 9.1.7:

$$f \in C^n([a,b]) \quad \Rightarrow \quad \left| \int_a^b f(t)\,\mathrm{d}t - Q_n(f) \right| \leq \frac{1}{n!}(b-a)^{n+1} \left\| f^{(n)} \right\|_{L^\infty([a,b])} \,. \tag{10.2.10}$$

(More refined estimates for Clenshaw-Curtis rules, see (9.2.9), and, in particular, for analytic integrands, see Rem. 9.2.19.)

△

10.3 Composite Quadrature

Sect. 9.4.1: approximation by piecewise polynomial interpolants

Recall Rem. 10.2.1 ➢ piecewise polynomial approximation also gives rise to quadrature formulas: composite quadrature formulas.

Recall: piecewise polynomial interpolation is based on a grid/mesh

$$\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_{m-1} < x_m = b\} \qquad ((9.4.1))$$

➢ composite quadrature formulas also rely on a mesh of the integration interval.

A motivation for composite quadrature, similar to the motivation for piecewise Lagrangian interpolation:

With $a = x_0 < x_1 < \cdots < x_{m-1} < x_m = b$

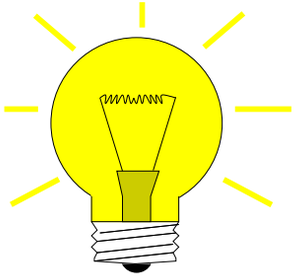
$$\int_a^b f(t)\,\mathrm{d}t = \sum_{j=1}^{m} \int_{x_{j-1}}^{x_j} f(t)\,\mathrm{d}t\ . \tag{10.3.1}$$

Recall (10.2.10): for polynomial quadrature rule (10.2.3) and $f \in C^n([a,b])$ quadrature error shrinks with $n+1$st power of length of integration interval.

Reduction of quadrature error can be achieved by

- splitting of the integration interval according to (10.3.1),
- using the intended quadrature formula on each sub-interval $[x_{j-1}, x_j]$.

Note: Increase in total no. of f-evaluations incurred, which has to be balanced with the gain in accuracy to achieve optimal efficiency, *cf.* Sect. 4.3.3 and Sect. 10.5 for algorithmic realization.



Idea:

- Partition integration domain $[a,b]$ by mesh (grid, $\rightarrow$ Sect.9.4)
$$\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_m = b\}$$
- 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

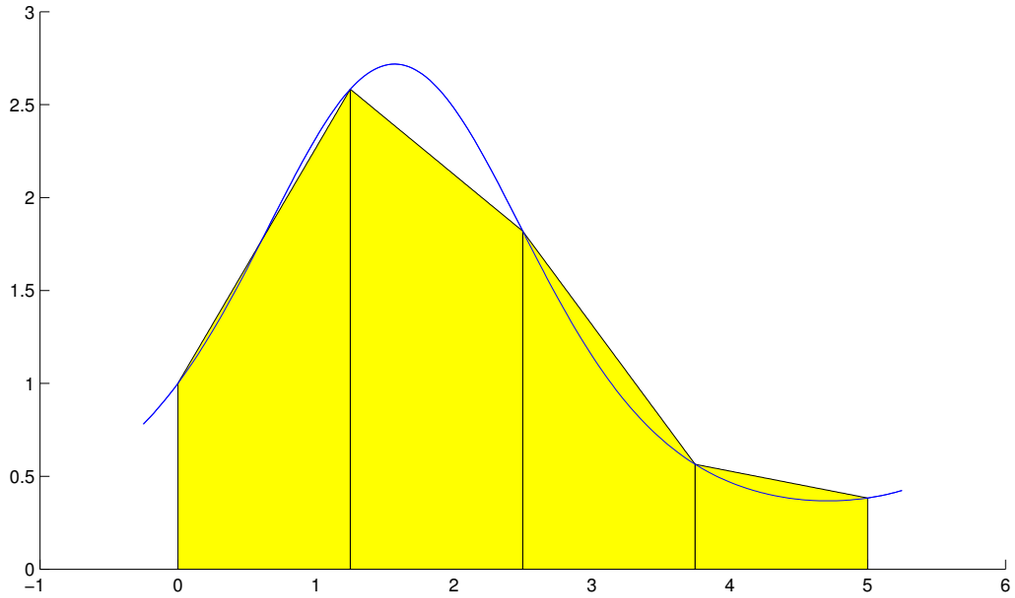
Analogy: global polynomial interpolation $\longleftrightarrow$ piecewise polynomial interpolation
($\rightarrow$ Sect. 9.4)

Note: Here we only consider one and the same quadrature formula (local quadrature formula) applied on all sub-intervals.

Example 10.3.2 (Simple composite polynomial quadrature rules).

Composite trapezoidal rule, *cf.* (10.2.7)

$$\int_a^b f(t)\mathrm{d}t = \tfrac{1}{2}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \tfrac{1}{2}(x_{j+1} - x_{j-1})f(x_j) + \tfrac{1}{2}(x_m - x_{m-1})f(b)\ . \tag{10.3.3}$$



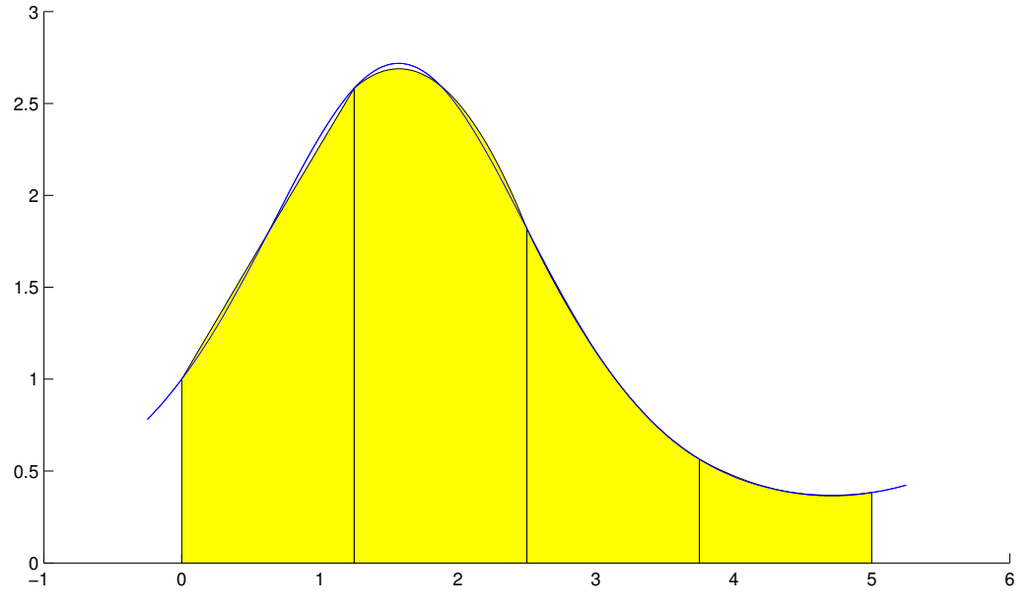
R. Hiptmair

rev 30401, December 13, 2010

➢ arising from piecewise linear interpolation of f.

Composite Simpson rule, *cf.* (10.2.8)

$$\int_a^b f(t)\mathrm{d}t = \frac{1}{6}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \frac{1}{6}(x_{j+1} - x_{j-1})f(x_j) + \sum_{j=1}^{m} \frac{2}{3}(x_j - x_{j-1})f(\tfrac{1}{2}(x_j + x_{j-1})) + \frac{1}{6}(x_m - x_{m-1})f(b)\ . \tag{10.3.4}$$



➢ related to piecewise quadratic Lagrangian interpolation.

Formulas (10.3.3), (10.3.4) directly suggest efficient implementation with minimal number of f-evaluations.

R. Hiptmair

rev 30401, December 13, 2010

How to rate the “quality” of a composite quadrature formula **?**

Clear: It is impossible to predict the quadrature error, unless the integrand is known.

Possible: *asymptotic perspective*

Predict decay of quadrature error as $m \to \infty$ (asymptotic perspective) for certain classes of integrands and “uniform” meshes.

This asymptotic perspective will be adopted in the following numerical experiments, which study the decay of the quadrature error for

- fixed integrand f,
- fixed local (polynomial) quadrature rule,
- for a sequence of ever finer equidistant grids.

Example 10.3.5 (Quadrature errors for composite quadrature rules).

Composite quadrature rules based on

- trapezoidal rule (10.2.7) ➢ local order 2 (exact for linear functions),
- Simpson rule (10.2.8) ➢ local order 3 (exact for quadratic polynomials)

on equidistant mesh $\mathcal{M} := \{jh\}_{j=0}^{n}$, $h = 1/n$, $n \in \mathbb{N}$.

Code 10.3.6: composite trapezoidal rule (10.3.3)

```
function res = trapezoidal(fnct,a,b,N)
% Numerical quadrature based on trapezoidal rule
% fnct handle to y = f(x)
% a,b bounds of integration interval
% N+1 = number of equidistant integration points (can be a vector)
res = [];
for n = N
  h = (b-a)/n;  x = (a:h:b); w = [0.5 ones(1,n-1) 0.5];
  res = [res; h, h*dot(w,feval(fnct,x))];
end
```

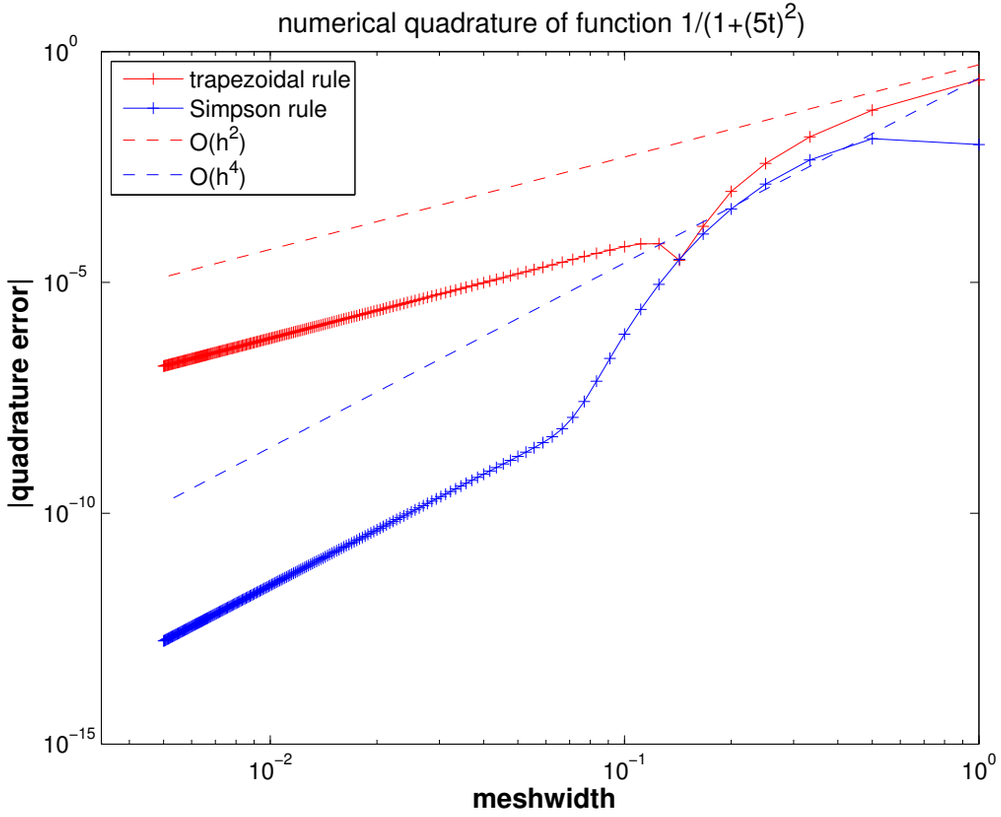
Code 10.3.7: composite Simpson rule (10.3.4)

```
function res = simpson(fnct,a,b,N)
% Numerical quadrature based on Simpson rule
% fnct handle to y = f(x)
% a,b bounds of integration interval
% N+1 = number of equidistant integration points (can be a vector)

res = [];
for n = N
  h = (b-a)/n;
  x = (a:h/2:b);
  fv = feval(fnct,x);
  val = sum(h*(fv(1:2:end-2)+4*fv(2:2:end-1)+fv(3:2:end)))/6;
  res = [res; h, val];
end
```

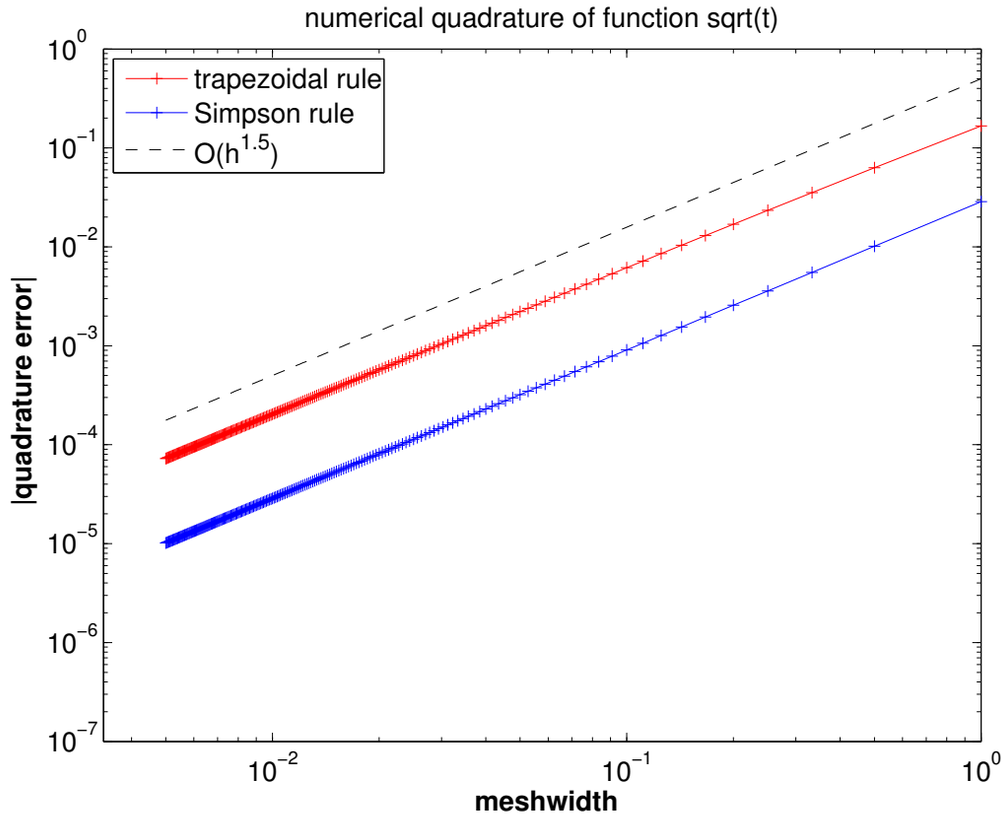
Note: `fnct` is supposed to accept vector arguments and return the function value for each vector component!

R. Hiptmair

rev 30401, December 13, 2010



quadrature error, $f_1(t) := \frac{1}{1+(5t)^2}$ on $[0,1]$



quadrature error, $f_2(t) := \sqrt{t}$ on $[0,1]$

Asymptotic behavior of quadrature error $E(n) := \left| \int_0^1 f(t)\,\mathrm{d}t - Q_n(f) \right|$ for meshwidth "$h \to 0$"

☛ algebraic convergence $E(n) = O(h^\alpha)$ of order $\alpha > 0$, $n = h^{-1}$

➢ Sufficiently smooth integrand f_1: trapezoidal rule $\to \alpha = 2$, Simpson rule $\to \alpha = 4$!?

R. Hiptmair

rev 30401, December 13, 2010

➢ singular integrand f_2: $\alpha = 3/2$ for trapezoidal rule **&** Simpson rule **!**

(lack of) smoothness of integrand limits convergence !

Composite simpson rule: rate = 4 **?** investigate *local* quadrature error on $[0, h]$ with MAPLE

```
> rule := 1/3*h*(f(2*h)+4*f(h)+f(0))
> err := taylor(rule - int(f(x),x=0..2*h),h=0,6);
```

$$err := \left(\frac{1}{90}\left(D^{(4)}\right)(f)(0)\,h^5 + O\left(h^6\right), h, 6\right)$$

➢ Composite Simpson rule converges with rate 4, indeed **!**

Code 10.3.8: errors of composite trapezoidal and Simpson rule

```
1 function comruleerrs()
2 % Numerical quadrature on [0,1]
3
4 figure('Name','1/(1+(5t)^2)');
```

R. Hiptmair

rev 30401, December 13, 2010

```
exact = atan(5)/5;
trres = trapezoidal(inline('1./(1+(5*x).^2)'),0,1,1:200);
smres = simpson(inline('1./(1+(5*x).^2)'),0,1,1:200);
loglog(trres(:,1),abs(trres(:,2)-exact),'r+-',...
       smres(:,1),abs(smres(:,2)-exact),'b+-',...
       trres(:,1),trres(:,1).^2*(trres(1,2)/trres(1,1)^2),'r--',...
       smres(:,1),smres(:,1).^4*(smres(1,2)/smres(1,1)^2),'b--');
set(gca,'fontsize',12);
title('numerical quadrature of function
  1/(1+(5t)^2)','fontsize',14);
xlabel('{\bf meshwidth}','fontsize',14);
ylabel('{\bf |quadrature error|}','fontsize',14);
legend('trapezoidal rule','Simpson rule','O(h^2)','O(h^4)',2);
axis([1/300 1 10^(-15) 1]);
trp1 =
  polyfit(log(trres(end-100:end,1)),log(abs(trres(end-100:end,2)-exa
smp1 =
  polyfit(log(smres(end-100:end,1)),log(abs(smres(end-100:end,2)-exa
print -dpsc2 '../PICTURES/compruleerr1.eps';

figure('Name','sqrt(t)');
exact = 2/3;
trres = trapezoidal(inline('sqrt(x)'),0,1,1:200);
```

R. Hiptmair

rev 30401, December 13, 2010

```
smres = simpson(inline('sqrt(x)'),0,1,1:200);
loglog(trres(:,1),abs(trres(:,2)-exact),'r+-',...
       smres(:,1),abs(smres(:,2)-exact),'b+-',...
       trres(:,1),trres(:,1).^(1.5)*(trres(1,2)/trres(1,1)^2),'k--')
set(gca,'fontsize',14);
title('numerical quadrature of function sqrt(t)','fontsize',14);
xlabel('{\bf meshwidth}','fontsize',14);
ylabel('{\bf |quadrature error|}','fontsize',14);
legend('trapezoidal rule','Simpson rule','O(h^{1.5})',2);
axis([1/300 1 10^(-7) 1]);
trp2 =
  polyfit(log(trres(end-100:end,1)),log(abs(trres(end-100:end,2)-exa
smp2 =
  polyfit(log(smres(end-100:end,1)),log(abs(smres(end-100:end,2)-exa
print -dpsc2 '../PICTURES/compruleerr2.eps';
```


◇

Composite quadrature formulas based on a grid $\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_m = b\}$ and a single local quadrature formula (usually) display *algebraic convergence* $O(h^p)$ of the quadrature error in terms of the meshwidth $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}|$.

Is it possible to predict the rate of algebraic convergence (for *sufficiently smooth*) integrands?

YES, and a simple criterion for the local quadrature rule will do the job!

R. Hiptmair

rev 30401, December 13, 2010

Gauge for "quality" of a quadrature formula Q_n:

$$\text{Order}(Q_n) := \max\{n \in \mathbb{N}_0: \quad Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \quad \forall p \in \mathcal{P}_n\} + 1$$

By construction: polynomial quadrature formulas (10.2.3) exact for $f \in \mathcal{P}_{n-1}$

$\Rightarrow$ n-point polynomial quadrature formula has at least order n

Remark 10.3.9 (Orders of simple polynomial quadrature formulas).

n		Order
0	midpoint rule	2
1	trapezoidal rule (10.2.7)	2
2	Simpson rule (10.2.8)	4
3	$\frac{3}{8}$-rule	4
4	Milne rule	6

△

Focus: *asymptotic* behavior of quadrature error for

mesh width $h := \max_{j=1,\ldots,m} |x_j - x_{j-1}| \to 0$

For *fixed* local n-point quadrature rule: $O(mn)$ f-evaluations for composite quadrature ("total cost")

➢ If mesh equidistant ($|x_j - x_{j-1}| = h$ for all j), then total cost for composite numerical quadrature $= O(h^{-1})$.

Theorem 10.3.10 (Convergence of composite quadrature formulas)**.**
For a composite quadrature formula Q based on a local quadrature formula of order $p \in \mathbb{N}$ holds

$$\exists C > 0: \quad \left| \int_I f(t)\,\mathrm{d}t - Q(f) \right| \leq C h^p \left\| f^{(p)} \right\|_{L^\infty(I)} \quad \forall f \in C^p(I), \forall \mathcal{M} \ .$$

Proof. Let s be the *piecewise polynomial interpolant* of local degree $p-1$ of f on the mesh $\mathcal{M}$, which relies on the local quadrature nodes (p for each interval of the mesh) as interpolation points, Sect. 9.4.1 and (9.4.2) for details.

$Q(f) = q(s)$, because s and f agree in all quadrature nodes,

Order p of local quadrature formulas ➢ local quadrature formulas will yield exact evaluation of $\int_{x_{j-1}}^{x_j} s(t)\,\mathrm{d}t$!

$$\Rightarrow \quad Q(s) = \int_a^b s(t)\,\mathrm{d}t$$

$$\Rightarrow \quad \left| \int_a^b f(t)\,\mathrm{d}t - Q(f) \right| = \left| \int_a^b f(t)\,\mathrm{d}t - \int_a^b s(t)\,\mathrm{d}t + \underbrace{Q(s) - Q(f)}_{=0} \right| \leq (b-a)\,\|f-s\|_{L^\infty([a,b])} \,.$$

Then apply the interpolation error estimate (9.4.6) for piecewise polynomial interpolation. □

Again essential, *cf.* Sect. 9.1: sufficient smoothness of integrand required to achieve algebraic concvergence in h as predicted by order of local quadrature rule.

Remark 10.3.11 (Removing a singularity by transformation).

R. Hiptmair

rev 30401, December 13, 2010

Ex. 10.3.5 ➢ lack of smoothness of integrand limits rate of algebraic convergence of composite quadrature rule for meshwidth $h \to 0$.

Idea: recover integral with smooth integrand by "analytic preprocessing"

Here is an example:

For $f \in C^\infty([0,b])$ compute $\int_0^b \sqrt{t} f(t)\,\mathrm{d}t$ via quadrature rule ($\to$ Ex. 10.3.5)

substitution $s = \sqrt{t}$: $$\int_0^b \sqrt{t} f(t)\,\mathrm{d}t = \int_0^{\sqrt{b}} \boxed{2s^2 f(s^2)}\,\mathrm{d}s\,. \tag{10.3.12}$$

Then: Apply quadrature rule to smooth integrand

△

Example 10.3.13 (Convergence of equidistant trapezoidal rule).

R. Hiptmair

rev 30401, December 13, 2010

Sometimes there are surprises: convergence of a composite quadrature rule is much better than predicted by the order of the local quadrature formula, see [31] for an explanation.

Equidistant trapezoidal rule (order 2), see (10.3.3)

$$\int_a^b f(t)\,\mathrm{d}t \approx T_m(f) := h\left(\tfrac{1}{2}f(a) + \sum_{k=1}^{m-1} f(kh) + \tfrac{1}{2}f(b)\right), \quad h := \frac{b-a}{m}. \tag{10.3.14}$$

Code 10.3.15: equidistant trapezoidal quadrature formula

```
1 function res = trapezoidal(fnct,a,b,N)
2 % Numerical quadrature based on trapezoidal rule
3 % fnct handle to y = f(x)
4 % a,b bounds of integration interval
5 % N+1 = number of equidistant integration points (can be a vector)
6 res = [];
7 for n = N
8   h = (b-a)/n;   x = (a:h:b); w = [0.5 ones(1,n-1) 0.5];
```

```
9     res = [res; h, h*dot(w,feval(fnct,x))];
0   end
```


1-periodic smooth (analytic) integrand

$$f(t) = \frac{1}{\sqrt{1 - a\sin(2\pi t - 1)}}\,, \quad 0 < a < 1\,.$$

(“exact value of integral”: use T_{500})

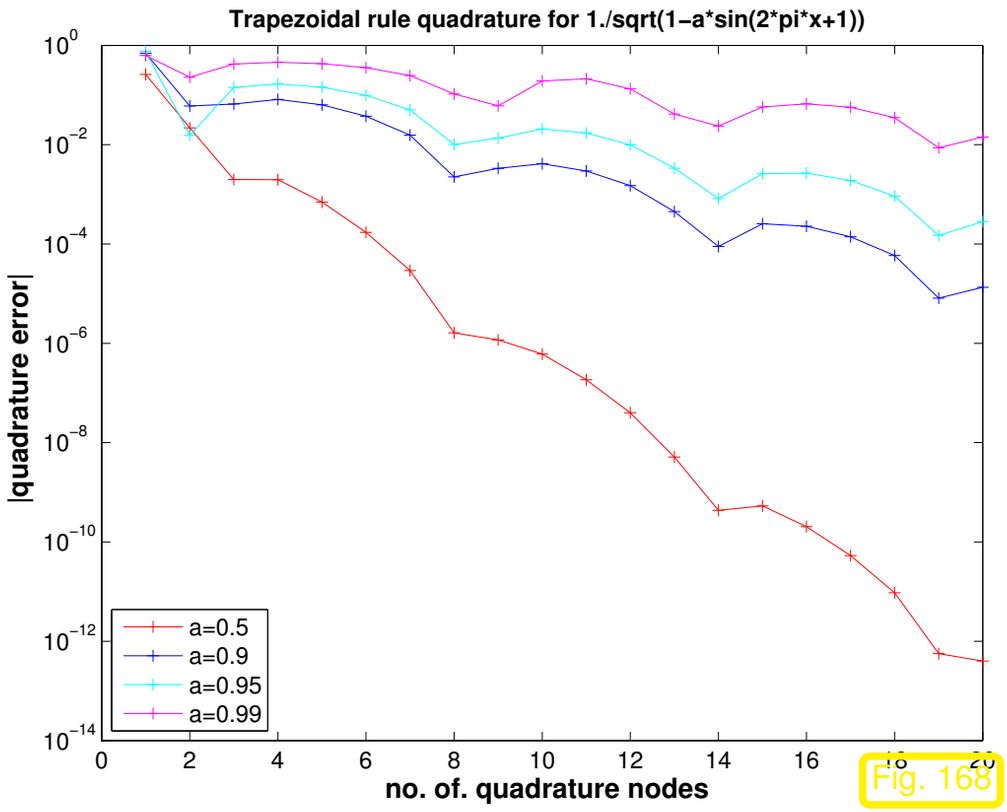


Fig. 168

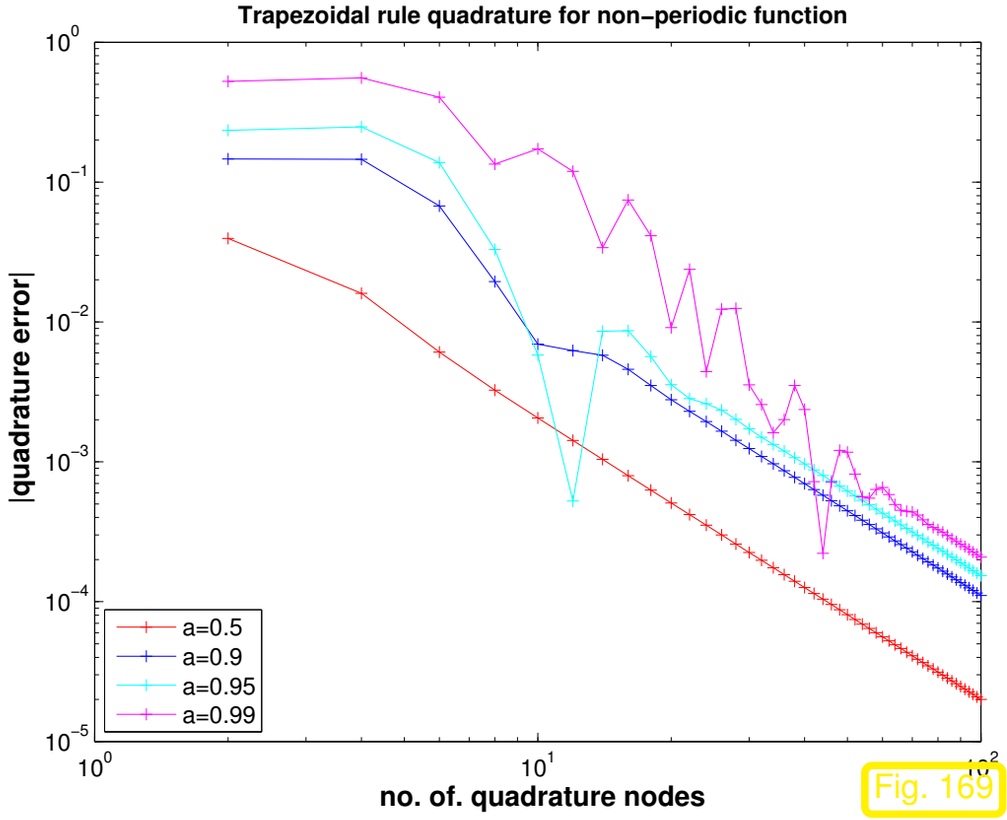


Fig. 169

quadrature error for $T_n(f)$ on $[0,1]$

exponential convergence !!

quadrature error for $T_n(f)$ on $[0,\frac{1}{2}]$

merely algebraic convergence

Code 10.3.16: tracking error of equidistant trapezoidal quadrature formula

```
1 function traperr()
2
3 clear a;
```

R. Hiptmair

rev 30401, December 13, 2010

```
global a;
l = 0; r = 0.5; % integration interval
N = 50;
a = 0.5; res05 = trapezoidal(@issin,l,r,1:N);
ex05 = trapezoidal(@issin,l,r,500); ex05 = ex05(1,2);
a = 0.9; res09 = trapezoidal(@issin,l,r,1:N);
ex09 = trapezoidal(@issin,l,r,500); ex09 = ex09(1,2);
a = 0.95; res95 = trapezoidal(@issin,l,r,1:N);
ex95 = trapezoidal(@issin,l,r,500); ex95 = ex95(1,2);
a = 0.99; res99 = trapezoidal(@issin,l,r,1:N);
ex99 = trapezoidal(@issin,l,r,500); ex99 = ex99(1,2);
figure('name','trapezoidal rule for non-periodic function');
loglog(1./res05(:,1),abs(res05(:,2)-ex05),'r+-',...
     1./res09(:,1),abs(res09(:,2)-ex09),'b+-',...
     1./res95(:,1),abs(res95(:,2)-ex95),'c+-',...
     1./res99(:,1),abs(res99(:,2)-ex99),'m+-');
set(gca,'fontsize',12);
legend('a=0.5','a=0.9','a=0.95','a=0.99',3);
xlabel('{\bf no. of. quadrature nodes}','fontsize',14);
ylabel('{\bf |quadrature error|}','fontsize',14);
title('{\bf Trapezoidal rule quadrature for non-periodic
  function}','fontsize',12);

```


R. Hiptmair

rev 30401, December 13, 2010

```
print -depsc2 '../PICTURES/traperr2.eps';

clear a;
global a;
l = 0; r = 1; % integration interval
N = 20;
a = 0.5; res05 = trapezoidal(@issin,l,r,1:N);
ex05 = trapezoidal(@issin,l,r,500); ex05 = ex05(1,2);
a = 0.9; res09 = trapezoidal(@issin,l,r,1:N);
ex09 = trapezoidal(@issin,l,r,500); ex09 = ex09(1,2);
a = 0.95; res95 = trapezoidal(@issin,l,r,1:N);
ex95 = trapezoidal(@issin,l,r,500); ex95 = ex95(1,2);
a = 0.99; res99 = trapezoidal(@issin,l,r,1:N);
ex99 = trapezoidal(@issin,l,r,500); ex99 = ex99(1,2);
figure('name','trapezoidal rule for periodic function');
semilogy(1./res05(:,1),abs(res05(:,2)-ex05),'r+-',...
      1./res09(:,1),abs(res09(:,2)-ex09),'b+-',...
      1./res95(:,1),abs(res95(:,2)-ex95),'c+-',...
      1./res99(:,1),abs(res99(:,2)-ex99),'m+-');
set(gca,'fontsize',12);
legend('a=0.5','a=0.9','a=0.95','a=0.99',3);
xlabel('{\bf no. of. quadrature nodes}','fontsize',14);
ylabel('{\bf |quadrature error|}','fontsize',14);
```

```
49 title('{\bf Trapezoidal rule quadrature for
     1./sqrt(1-a*sin(2*pi*x+1))}','fontsize',12);
50
51 print -depsc2 '../PICTURES/traperr1.eps';
```


Explanation:

$$f(t) = e^{2\pi ikt} \quad \blacktriangleright \quad \begin{cases} \int_0^1 f(t)\,\mathrm{d}t = \begin{cases} 0 & \text{, if } k \neq 0\,, \\ 1 & \text{, if } k = 0\,. \end{cases} \\ T_m(f) = \frac{1}{m}\sum_{l=0}^{m-1} e^{\frac{2\pi i}{m}lk} \overset{(8.2.5)}{=} \begin{cases} 0 & \text{, if } k \notin m\mathbb{Z}\,, \\ 1 & \text{, if } k \in m\mathbb{Z}\,. \end{cases} \end{cases}$$

$\blacktriangleright$

Equidistant trapezoidal rule T_m is exact for trigonometric polynomials of degree $< 2m$!

It takes sophisticated tools from complex analysis to conclude exponential convergence for analytic integrands from the above observation.

R. Hiptmair

Remark 10.3.17 (Choice of (local) quadrature weights).

So far we have constructed quadrature formulas from (piecewise) polynomial interpolation ➥ (local) Newton-Cotes formulas from Ex. 10.2.6. The degree of the Lagrange interpolation polynomials determined the order of the quadrature formula.

Question: given arbitrary quadrature nodes, how can be achieve a certain order by choosing the weights appropriately?

Given: arbitrary nodes $\xi_1, \ldots, \xi_n$ for n-point (local) quadrature formula on $[a, b]$

Take cue from polynomial quadrature formulas: choice of weights ω_j according to (10.2.4) ensures order $\geq n$.

There is a more direct way without detour via Lagrange polynomials:

If $p_0, \ldots, p_{n-1}$ is a basis of $\mathcal{P}_n$, then, thanks to the linearity of the integral and quadrature formulas,

$$Q_n(p_j) = \int_a^b p_j(t)\,\mathrm{d}t \quad \forall j = 0, \ldots, n-1 \quad \Leftrightarrow \quad Q_n \text{ has order } \geq n\,. \tag{10.3.18}$$

➢ $n \times n$ linear system of equations, see (10.4.2) for an example:

$$\begin{pmatrix} p_0(\xi_1) & \cdots & p_0(\xi_n) \\ \vdots & & \vdots \\ p_{n-1}(\xi_n) & \cdots & p_{n-1}(\xi_n) \end{pmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_n \end{pmatrix} = \begin{pmatrix} \int_a^b p_0(t)\,\mathrm{d}t \\ \vdots \\ \int_a^b p_{n-1}(t)\,\mathrm{d}t \end{pmatrix}\,. \tag{10.3.19}$$

For instance, for the computation of quadrature weights, one may choose the monomial basis $p_j(t) = t^j$.

△

Natural question: What is the maximal order for an n-point quadrature formula **?**

Lemma 10.3.20 (Bound for order of quadrature formula)**.**

There is no n-point quadrature formula of order $2n+1$

Proof. (indirect) Assume there was an n-point quadrature formula with nodes $a \leq \xi_1 < \xi_2 < \ldots < \xi_n \leq b$ of order $2n+1$.

➤ Construct polynomial $p(t) := \prod_{j=1}^{n}(t-\xi_j)^2 \in \mathcal{P}_{2n}$

$$Q_n(p) = 0 \quad \text{but} \quad \int_a^b p(t)\,\mathrm{d}t > 0\ .$$

Thus, the assumption leads to a contradiction. □

10.4 Gauss Quadrature [29, Ch. 40-41], [10, Sect.10.3]

R. Hiptmair

rev 30401, December 13, 2010

Natural question: Are there n-point quadrature formulas of maximal order $2n$ **?**

Heuristics: A quadrature formula has order $m \in \mathbb{N}$ already, if it is exact for m polynomials $\in \mathcal{P}_{m-1}$ that form a basis of $\mathcal{P}_{m-1}$ (recall Thm. 3.2.2).

$$\Updownarrow$$

An n-point quadrature formula has $2n$ "degrees of freedom" (n node positions, n weights).

"No. of equations = No. of unknowns"

Example 10.4.1 (2-point quadrature rule of order 4).

Necessary & sufficient conditions for order 4 , *cf.* (10.3.19):

$$Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \;\forall p \in \mathcal{P}_3 \quad \Leftrightarrow \quad Q_n(t^q) = \frac{1}{q+1}(b^{q+1} - a^{q+1}) \;, \quad q = 0,1,2,3 \;.$$

4 equations for weights ω_j and nodes ξ_j, $j = 1, 2$ ($a = -1, b = 1$), *cf.* Rem. 10.3.17

$$\begin{aligned} \int_{-1}^{1} 1 \,\mathrm{d}t = 2 = 1\omega_1 + 1\omega_2 \quad , \quad \int_{-1}^{1} t \,\mathrm{d}t = 0 = \xi_1\omega_1 + \xi_2\omega_2 \\ \int_{-1}^{1} t^2 \,\mathrm{d}t = \frac{2}{3} = \xi_1^2\omega_1 + \xi_2^2\omega_2 \quad , \quad \int_{-1}^{1} t^3 \,\mathrm{d}t = 0 = \xi_1^3\omega_1 + \xi_2^3\omega_2 \,. \end{aligned} \tag{10.4.2}$$

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);
```

➢ weights & nodes: $\left\{\omega_2 = 1, \omega_1 = 1, \xi_1 = 1/3\sqrt{3}, \xi_2 = -1/3\sqrt{3}\right\}$

◥ quadrature formula:
$$\int_{-1}^{1} f(x)\,dx \approx f\left(\frac{1}{\sqrt{3}}\right) + f\left(-\frac{1}{\sqrt{3}}\right) \tag{10.4.3}$$

◇

R. Hiptmair

rev 30401, December 13, 2010

Optimist's **assumption**: $\exists$ family of n-point quadrature formulas

$$Q_n(f) := \sum_{j=1}^{n} \omega_j^n f(\xi_j^n) \approx \int_{-1}^{1} f(t)\,\mathrm{d}t\ , \quad n \in \mathbb{N}\ ,$$

of order $2n$ $\Leftrightarrow$ exact for polynomials $\in \mathcal{P}_{2n-1}$. (10.4.4)

Define $\bar{P}_n(t) := (t-\xi_1^n)\cdot\cdots\cdot(t-\xi_n^n)\ , \quad t \in \mathbb{R} \ \Rightarrow\ \bar{P}_n \in \mathcal{P}_n\ .$

Note: $\bar{P}_n$ has leading coefficient $= 1$.

By assumption on the order of Q_n: for any $q \in \mathcal{P}_{n-1}$

$$\int_{-1}^{1} \underbrace{q(t)\bar{P}_n(t)}_{\in\mathcal{P}_{2n-1}}\,\mathrm{d}t \overset{(10.4.4)}{=} \sum_{j=1}^{n} \omega_j^n q(\xi_j^n) \underbrace{\bar{P}_n(\xi_j^n)}_{=0} = 0\ .$$

$$\Rightarrow \quad \text{orthogonality} \quad \int_{-1}^{1} q(t)\bar{P}_n(t)\,\mathrm{d}t = 0 \quad \forall q \in \mathcal{P}_{n-1}\ . \tag{10.4.5}$$

$L^2(]-1,1[)$-inner product of q and $\bar{P}_n$

Recall: $(f,g) \mapsto \int_a^b f(t)g(t)\,\mathrm{d}t$ is an inner product on $C^0([a,b])$, the L^2-inner product, see [39, Sect. 4.4, Ex. 2], [23, Ex. 6.5]

➢ Treat space of polynomials $\mathcal{P}_n$ as a vector space equipped with an inner product.

➢ Abstract techniques for vector spaces with inner product can be applied to polynomials, for instance Gram-Schmidt orthogonalization, *cf.* (5.2.11), [39, Thm. 4.8], [23, Alg. 6.1].

Abstract Gram-Schmidt orthogonalization: in a vector space with inner product $\cdot$ orthogonal vectors $q_0, q_1, \ldots$ spanning the same subspaces as the linearly independent vectors $v_0, v_1, \ldots$ are constructed recursively via

$$q_{n+1} := v_{n+1} - \sum_{k=0}^{n} \frac{v_{n+1} \cdot q_k}{q_k \cdot q_k} q_k \quad , \quad q_0 := v_0 \, . \tag{10.4.6}$$

➢ Construction of $\bar{P}_n$ by Gram-Schmidt orthogonalization of monomial basis $\{1, t, t^2, \ldots, t^{n-1}\}$ of $\mathcal{P}_{n-1}$ w.r.t. $L^2(]-1,1[)$-inner product:

$$\bar{P}_0(t) := 1 \, , \quad \bar{P}_{n+1}(t) = t^n - \sum_{k=0}^{n} \frac{\int_{-1}^{1} t^n \bar{P}_k(t)\,\mathrm{d}t}{\int_{-1}^{1} \bar{P}_k^2(t)\,\mathrm{d}t} \cdot \bar{P}_k(t) \tag{10.4.7}$$

R. Hiptmair

rev 30401, December 13, 2010

The considerations so far only reveal constraints on the nodes of an n-point quadrature rule of order $2n$.

They do by no means confirm the existence of such rules, but offer a clear hint on how to construct them:

Theorem 10.4.8 (Existence of n-point quadrature formulas of order $2n$)**.**
Let $\{\bar{P}_n\}_{n\in\mathbb{N}_0}$ be a family of non-zero polynomials that satisfies

- $\bar{P}_n \in \mathcal{P}_n$,
- $\int_{-1}^{1} q(t)\bar{P}_n(t)\,\mathrm{d}t = 0$ *for all* $q \in \mathcal{P}_{n-1}$ ($L^2(]-1,1[)$-*orthogonality*),
- *The set* $\{\xi_j^n\}_{j=1}^m$, $m \le n$, *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. 10.3.17 provides a quadrature formula of order $2n$ on $[-1,1]$.

R. Hiptmair

Proof. Conclude from the orthogonality of the $\bar{P}_n$ that $\{\bar{P}_k\}_{k=0}^n$ is a basis of $\mathcal{P}_n$ and

$$\int_{-1}^{1} h(t)\bar{P}_n(t)\,\mathrm{d}t = 0 \quad \forall h \in \mathcal{P}_{n-1}\ . \tag{10.4.9}$$

Recall division of polynomials with remainder (Euclid's algorithm $\rightarrow$ Course "Diskrete Mathematik"): for any $p \in \mathcal{P}_{2n-1}$

$$p(t) = h(t)\bar{P}_n(t) + r(t)\ , \quad \text{for some} \quad h \in \mathcal{P}_{n-1}, r \in \mathcal{P}_{n-1}\ . \tag{10.4.10}$$

Apply this representation to the integral:

$$\int_{-1}^{1} p(t)\,\mathrm{d}t = \underbrace{\int_{-1}^{1} h(t)\bar{P}_n(t)\,\mathrm{d}t}_{=0 \text{ by (10.4.9)}} + \int_{-1}^{1} r(t)\,\mathrm{d}t \overset{(*)}{=} \sum_{j=1}^{m} \omega_j^n r(\xi_j^n)\ , \tag{10.4.11}$$

$(*)$: by choice of weights according to Rem. 10.3.17 Q_n is exact for polynomials of degree $\leq n-1$!

By choice of nodes as zeros of $\bar{P}_n$ using (10.4.9):

$$\sum_{j=1}^{m} \omega_j^n p(\xi_j^n) \overset{(10.4.10)}{=} \sum_{j=1}^{m} \omega_j^n h(\xi_j^n) \underbrace{\bar{P}_n(\xi_j^n)}_{=0} + \sum_{j=1}^{m} \omega_j^n r(\xi_j^n) \overset{(10.4.11)}{=} \int_{-1}^{1} p(t)\,\mathrm{d}t\ . \qquad \square$$

The family of polynomials $\{\bar{P}_n\}_{n\in\mathbb{N}_0}$ are so-called orthogonal polynomials w.r.t. the $L^2(]-1,1[)$-inner product. They play a key role in analysis.

Definition 10.4.12 (Legendre polynomials)**.**
The n-th Legendre polynomial P_n is defined by

- $P_n \in \mathcal{P}_n$,
- $\int_{-1}^{1} P_n(t)q(t)\,\mathrm{d}t = 0 \;\forall q \in \mathcal{P}_{n-1}$,
- $P_n(1) = 1$.

Legendre polynomials $P_0, \ldots, P_5$ ➢

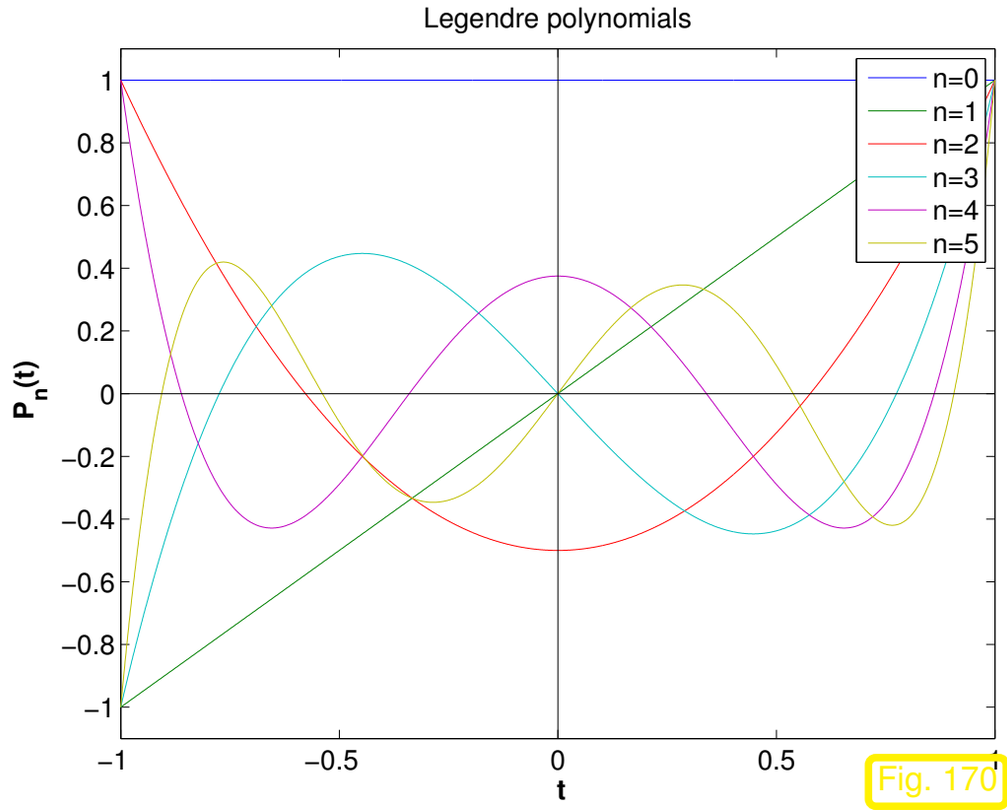


Fig. 170

Notice: the polynomials $\bar{P}_n$ defined by (10.4.7) and the Legendre polynomials P_n of Def. 10.4.12 (merely) *differ* by a constant factor!

R. Hiptmair

rev 30401, December 13, 2010

Note: the above considerations, recall (10.4.5), show that the nodes of an n-point quadrature formula of order $2n$ on $[-1,1]$ must agree with the zeros of $L^2(]-1,1[)$-orthogonal polynomials.

n-point quadrature formulas of order $2n$ are unique

This is not surprising in light of "$2n$ equations for $2n$ degrees of freedom".

We are not done yet: the zeros of $\bar{P}_n$ from (10.4.7) may lie outside $[-1,1]$.
In principle $\bar{P}_n$ could also have less than n real zeros.

The next lemma shows that all this cannot happen.

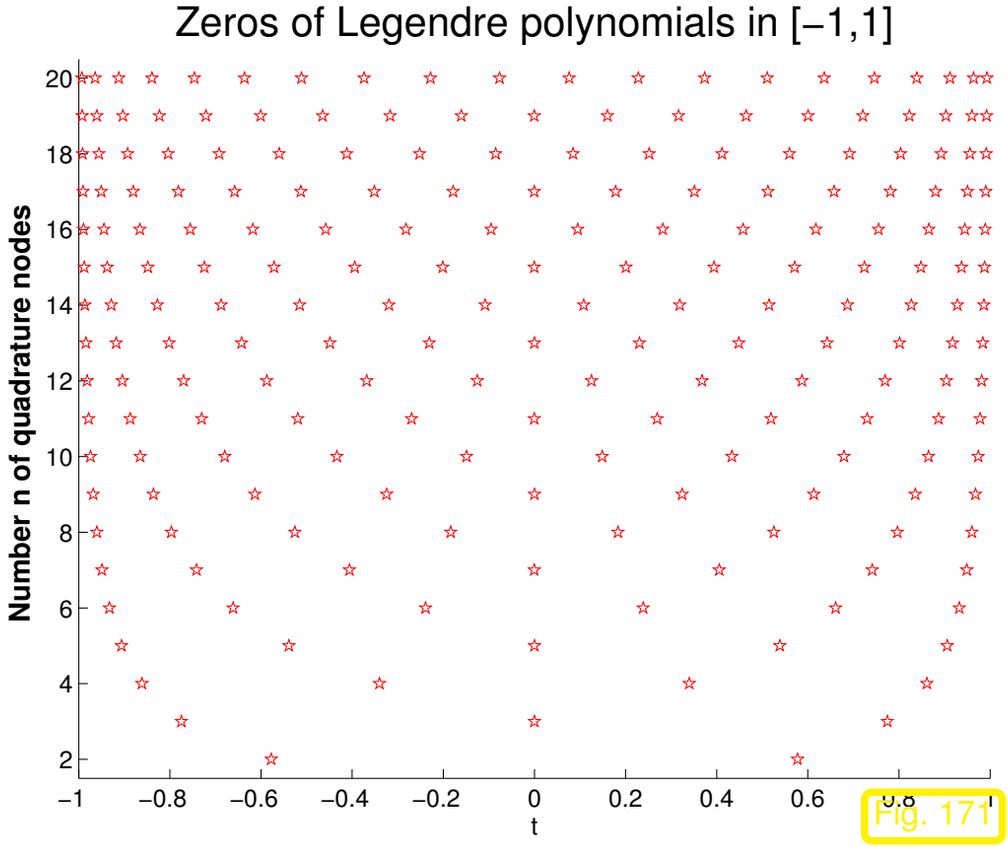


Fig. 171

◁ Obviously:

Lemma 10.4.13 (Zeros of Legendre polynomials)**.**

P_n *has* n *distinct zeros in* $]-1,1[$.

Zeros of Legendre polynomials = Gauss points

R. Hiptmair

rev 30401, December 13, 2010

Proof. (indirect) Assume that P_n has only $m < n$ zeros $\zeta_1, \ldots, \zeta_m$ in $]-1,1[$ *at which it changes sign*. Define

$$q(t) := \prod_{j=1}^{m} (t - \zeta_j) \quad \Rightarrow \quad qP_n \geq 0 \quad \text{or} \quad qP_n \leq 0 \, .$$

$$\Rightarrow \quad \int_{-1}^{1} q(t) P_n(t) \, \mathrm{d}t \neq 0 \, .$$

As $q \in \mathcal{P}_{n-1}$, this contradicts (10.4.9). □

Quadrature formula from Thm. 10.4.8: Gauss-Legendre quadrature
(nodes 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.

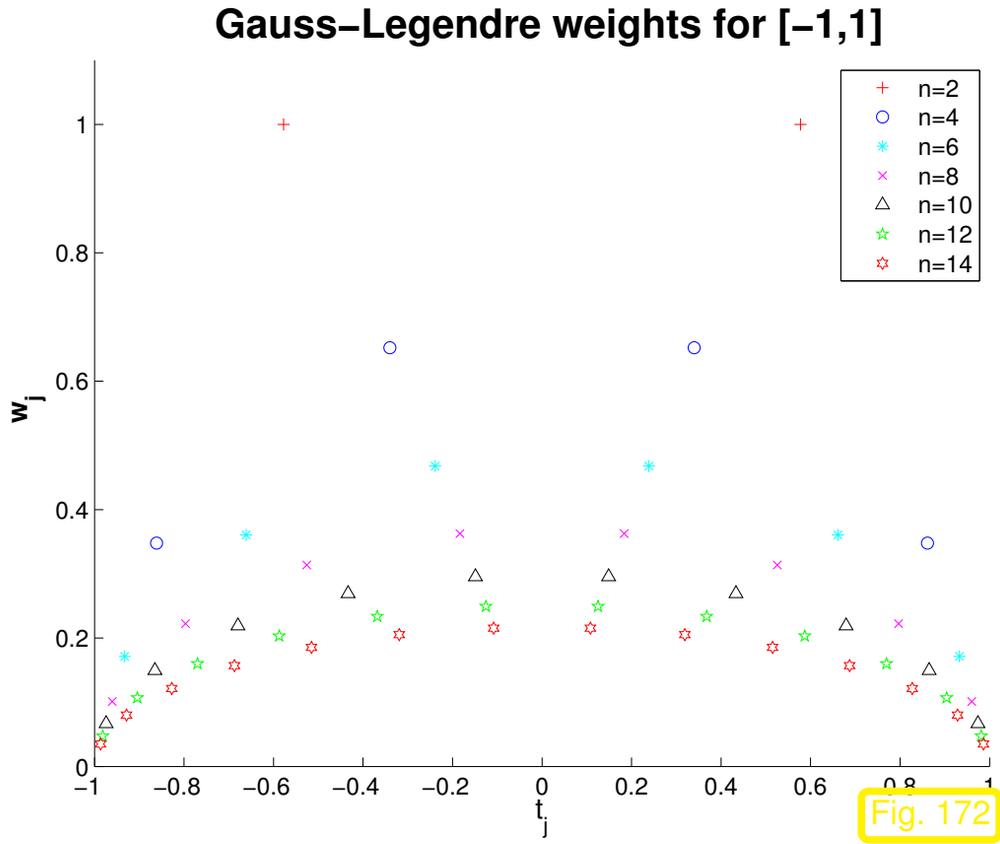


Fig. 172

R. Hiptmair

rev 30401, December 13, 2010

Proof. Writing ξ_j^n, $j = 1, \ldots, n$, for the nodes (Gauss points) of the n-point Gauss-Legendre quadrature formula, $n \in \mathbb{N}$, we define

$$q_k(t) = \prod_{\substack{j=1 \\ j \neq k}}^{n} (t - \xi_j^n)^2 \quad \Rightarrow \quad q_k \in \mathcal{P}_{2n-2} \, .$$

This polynomial is integrated exactly by the quadrature rule: since $q_k(\xi_j^n) = 0$ for $j \neq k$

$$0 < \int_{-1}^{1} q(t) \, \mathrm{d}t = \omega_k^n \underbrace{q(\xi_k^n)}_{>0} \, ,$$

where ω_j^n are the quadrature weights. □

R. Hiptmair

rev 30401, December 13, 2010

Remark 10.4.15 (3-Term recursion for Legendre polynomials).

Note: polynomials $\bar{P}_n$ from (10.4.7) are uniquely characterized by the two properties (try a proof!)

- $\bar{P}_n \in \mathcal{P}_n$ with leading coefficient 1: $\bar{P}(t) = t^n + \ldots$,

- $\int_{-1}^{1} \bar{P}_k(t)\bar{P}_j(t)\,\mathrm{d}t = 0$, if $j \neq k$ ($L^2(]-1,1[)$-orthogonality).

➢ same polynomials $\bar{P}_n$ by another Gram-Schmidt orthogonalization procedure, *cf.* (10.4.6),

$$\bar{P}_{n+1}(t) = t\bar{P}_n(t) - \sum_{k=0}^{n} \frac{\int_{-1}^{1} \tau\bar{P}_n(\tau)\bar{P}_k(\tau)\,\mathrm{d}\tau}{\int_{-1}^{1} \bar{P}_k^2(\tau)\,\mathrm{d}\tau} \cdot \bar{P}_k(t)$$

By orthogonality (10.4.9) the sum collapses, since $\int_{-1}^{1} \tau\bar{P}_n(\tau)\bar{P}k(\tau)\,\mathrm{d}\tau = \int_{-1}^{1} \bar{P}_n(\tau)\underbrace{\left(\tau\bar{P}k(\tau)\right)}_{\in\mathcal{P}_{k+1}}\mathrm{d}\tau = 0$, if $k+1 < n$:

$$\bar{P}_{n+1}(t) = t\bar{P}_n(t) - \frac{\int_{-1}^{1} \tau\bar{P}_n(\tau)\bar{P}_n(\tau)\,\mathrm{d}\tau}{\int_{-1}^{1} \bar{P}_n^2(\tau)\,\mathrm{d}\tau} \cdot \bar{P}_n(t) - \frac{\int_{-1}^{1} \tau\bar{P}_n(\tau)\bar{P}_{n-1}(\tau)\,\mathrm{d}\tau}{\int_{-1}^{1} \bar{P}_{n-1}^2(\tau)\,\mathrm{d}\tau} \cdot \bar{P}_{n-1}(t)\ . \tag{10.4.16}$$

After rescaling (tedious!): 3-term recursion for Legendre polynomials

$$P_{n+1}(t) := \frac{2n+1}{n+1}\,tP_n(t) - \frac{n}{n+1}P_{n-1}(t) \quad , \quad P_0 := 1\ , \quad P_1(t) := t\ . \tag{10.4.17}$$

Reminder ($\rightarrow$ Sect. 9.2.1): similar 3-term recursion (9.2.7) for Chebychev polynomials. Coincidence? Of course not, nothing in mathematics holds "by accident". 3-term recursions are a distinguishing feature of so-called families of orthogonal polynomials, to which the Chebychev polynomials belong

as well, spawned by Gram-Schmidt orthogonalization with respect to a weighted L^2-inner product, however, see [29, VI].

➤ Efficient and *stable* evaluation of Legendre polynomials by means of 3-term recursion (10.4.17), *cf.* the analoguous algorithm for Chebychev polynomials given in Code 9.2.1.

Code 10.4.18: computing Legendre polynomials

```
1 function V= legendre(n,x)
2 V = ones(size(x)); V = [V; x];
3 for j=1:n-1
4   V = [V; ((2*j+1)/(j+1)).*x.*V(end,:) - j/(j+1)*V(end-1,:)]; end
```


Comments on Code 10.4.17:

- ☛ return value: matrix $\mathbf{V}$ with $(\mathbf{V})_{ij} = P_i(x_j)$
- ☛ line 2: takes into account initialization of Legendre 3-term recursion (10.4.17)

Remark 10.4.19 (Computing Gauss nodes and weights).

Compute nodes/weights of Gaussian quadrature by solving an eigenvalue problem! (Golub-Welsch algorithm [18, Sect. 3.5.4])

In codes: ξ_j, ω_j from tables!

Code 10.4.20: Golub-Welsch algorithm

```
function [x,w]=gaussquad(n)
b = zeros(n-1,1);
for i=1:(n-1), b(i)=i/sqrt(4*i*i-1); end
J=diag(b,-1)+diag(b,1); [ev,ew]=eig(J);
x=diag(ew); w=(2*(ev(1,:).*ev(1,:)))';
```

Justification: rewrite 3-term recurrence (10.4.17) for scaled Legendre polynomials $\widetilde{P}_n = \frac{1}{\sqrt{n+1/2}} P_n$

$$t\widetilde{P}_n(t) = \underbrace{\frac{n}{\sqrt{4n^2-1}}}_{=:\beta_n}\widetilde{P}_{n-1}(t) + \underbrace{\frac{n+1}{\sqrt{4(n+1)^2-1}}}_{=:\beta_{n+1}}\widetilde{P}_{n+1}(t)\,. \tag{10.4.21}$$

For fixed $t \in \mathbb{R}$ (10.4.21) can be expressed as

$$t\begin{pmatrix} \widetilde{P}_0(t) \\ \widetilde{P}_1(t) \\ \vdots \\ \widetilde{P}_{n-1}(t) \end{pmatrix} = \begin{pmatrix} 0 & \beta_1 & & & \\ \beta_1 & 0 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots \\ & & & 0 & \beta_{n-1} \end{pmatrix} \begin{pmatrix} \widetilde{P}_0(t) \\ \widetilde{P}_1(t) \\ \vdots \\ \widetilde{P}_{n-1}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \beta_n \widetilde{P}_n(t) \end{pmatrix}$$

R. Hiptmair

rev 30401, December 13, 2010

$$\widetilde{P}_n(\xi) = 0 \quad \Leftrightarrow \quad \xi \mathbf{p}(\xi) = \mathbf{J}_n \mathbf{p}(\xi) \, .$$

The zeros of P_n can be obtained as the n real eigenvalues of the symmetric tridiagonal matrix $\mathbf{J}_n \in \mathbb{R}^{n,n}$!

This matrix $\mathbf{J}_n$ is initialized in line 4 of Code 10.4.19. The computation of the weights in line 6 of Code 10.4.19 is explained in [18, Sect. 3.5.4].

The initial rationale for introducing Gauss quadrature was to obtain a high-order local quadrature rule as building block for composite quadrature rules ($\rightarrow$ Sect. 10.3).

However, once we have the family of Gauss quadrature rules, we may also employ them as global quadrature rules on the entire inegration interval $[a, b]$ and achieve high accuracy by using more and more quadrature nodes.

This kind of use of Gaussian quadrature is explored in the next numerical experiments.

Example 10.4.22 (Error of (non-composite) quadratures).

Code 10.4.23: important polynomial quadrature rules

```
function res = numquad(f,a,b,N,mode)
% Numerical quadrature on [a,b] by polynomial quadrature formula
% f -> function to be integrated (handle), must support vector arguments
% a,b -> integration interval [a,b] (endpoints included)
% N -> Maximal degree of polynomial
% mode (equidistant, Chebychev = Clenshaw-Curtis, Gauss) selects quadrature
  rule
if (nargin < 5), mode = 'equidistant'; end
res = [];

if strcmp(mode,'Gauss')
  for deg=1:N
    [gx,w] = gaussQuad(deg);
    % Gauss points for [a,b]
    x = 0.5*(b-a)*gx+0.5*(a+b);
    y = feval(f,x);
    res = [res; deg, 0.5*(b-a)*dot(w,y)];
  end
else
```

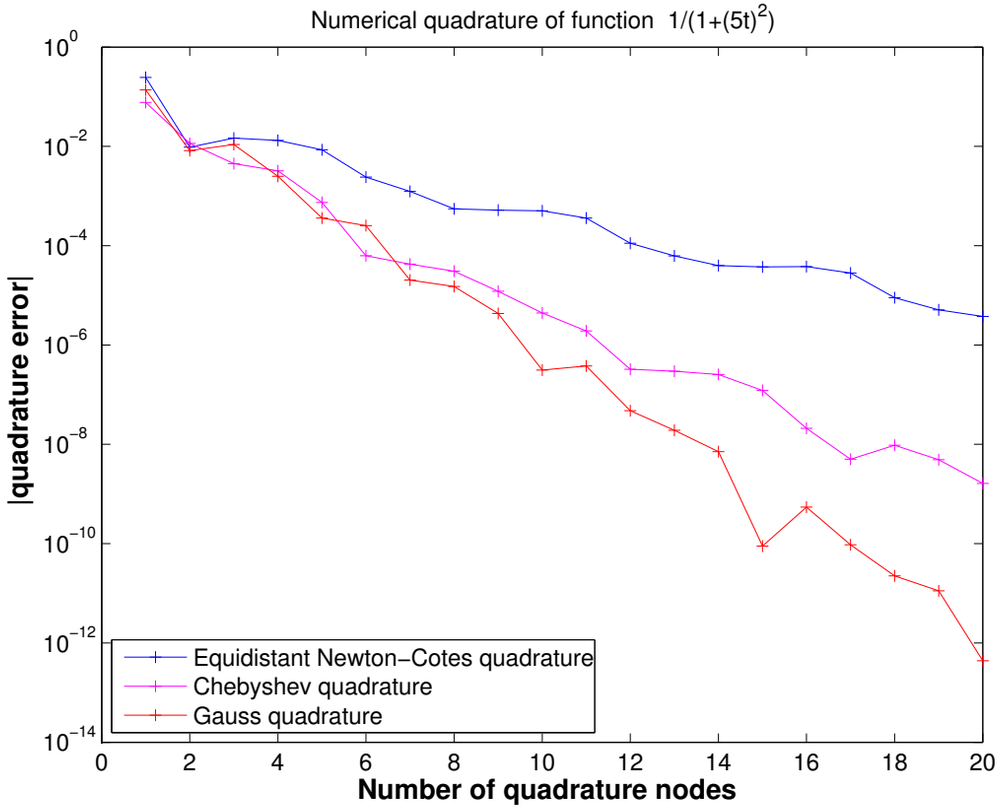
R. Hiptmair

rev 30401, December 13, 2010

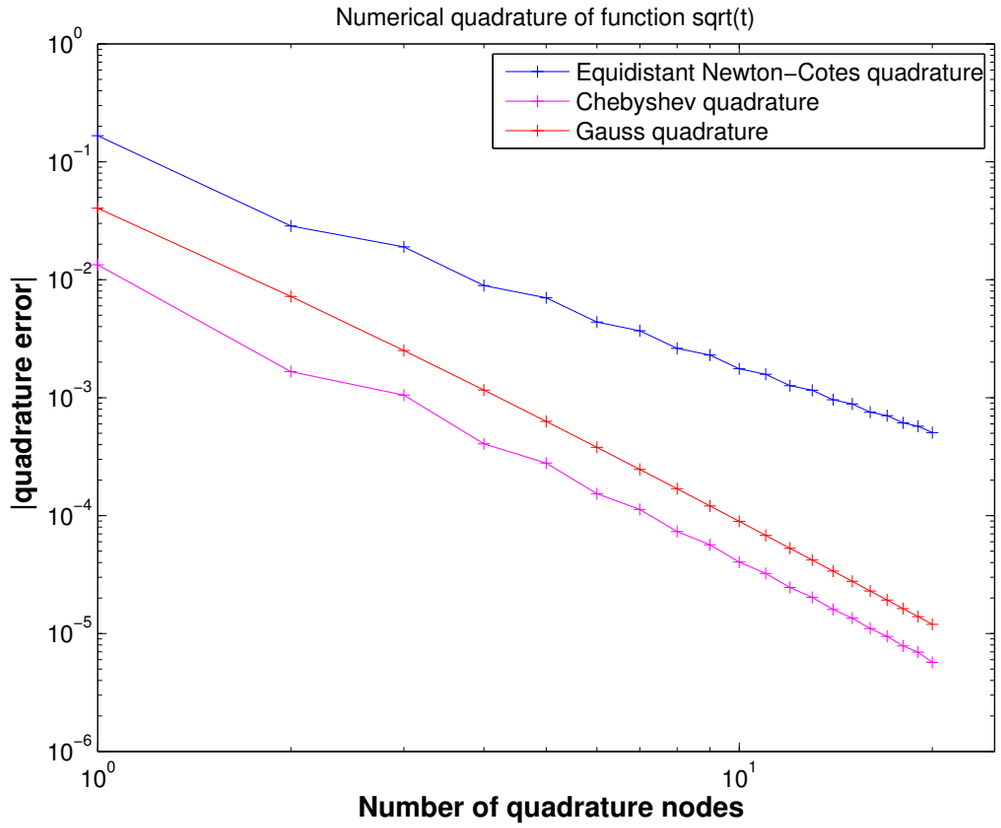
```
19    p = (N+1:-1:1);
20    w = (b.^p - a.^p)./p;
21    for deg=1:N
22      if strcmp(mode,'Chebychev')
23        % Chebychev nodes on [a,b], see (9.2.12)
24        x = 0.5*(b-a)*cos((2*(0:deg)+1)/(2*deg+2)*pi)+0.5*(a+b);
25      else
26        x = (a:(b-a)/deg:b);
27      end
28      % "Quick and dirty" implementation through polynomial interpolation
29      y = feval(f,x);
30      poly = polyfit(x,y,deg);
31      res = [res; deg, dot(w(N+1-deg:N+1),poly)];
32    end
33  end
```


R. Hiptmair

rev 30401, December 13, 2010



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 30401, December 13, 2010

Asymptotic behavior of quadrature error $\epsilon_n := \left| \int_0^1 f(t)\,\mathrm{d}t - Q_n(f) \right|$ for "'$n \to \infty$":

➢ exponential convergence $\epsilon_n \approx O(q^n)$, $0 < q < 1$, for C^∞-integrand f_1 $\rightsquigarrow$: Newton-Cotes quadrature : $q \approx 0.61$, Clenshaw-Curtis quadrature : $q \approx 0.40$, Gauss-Legendre quadrature : $q \approx 0.27$

➢ algebraic convergence $\epsilon_n \approx O(n^{-\alpha})$, $\alpha > 0$, for integrand f_2 with singularity at $t = 0$ $\rightsquigarrow$ Newton-Cotes quadrature : $\alpha \approx 1.8$, Clenshaw-Curtis quadrature : $\alpha \approx 2.5$, Gauss-Legendre quadrature : $\alpha \approx 2.7$

Code 10.4.24: tracking errors of quadrature rules

```
function numquaderrs()
% Numerical quadrature on [0,1]
N = 20;

figure('Name','1/(1+(5t)^2)');
exact = atan(5)/5;
eqdres = numquad(inline('1./(1+(5*x).^2)'),0,1,N,'equidistant');
chbres = numquad(inline('1./(1+(5*x).^2)'),0,1,N,'Chebychev');
gaures = numquad(inline('1./(1+(5*x).^2)'),0,1,N,'Gauss');
semilogy(eqdres(:,1),abs(eqdres(:,2)-exact),'b+-',...
     chbres(:,1),abs(chbres(:,2)-exact),'m+-',...
     gaures(:,1),abs(gaures(:,2)-exact),'r+-');
set(gca,'fontsize',12);
title('Numerical quadrature of function  1/(1+(5t)^2)');
xlabel('{\bf Number of quadrature nodes}','fontsize',14);
ylabel('{\bf |quadrature error|}','fontsize',14);
legend('Equidistant Newton-Cotes quadrature',...
```

```
18          'Clenshaw-Curtis quadrature',...
19          'Gauss quadrature',3);
20 eqdp1 = polyfit(eqdres(:,1),log(abs(eqdres(:,2)-exact)),1)
21 chbp1 = polyfit(chbres(:,1),log(abs(chbres(:,2)-exact)),1)
22 gaup1 = polyfit(gaures(:,1),log(abs(gaures(:,2)-exact)),1)
23 print -depsc2 '../PICTURES/numquaderr1.eps';
24
25 figure('Name','sqrt(t)');
26 exact = 2/3;
27 eqdres = numquad(inline('sqrt(x)'),0,1,N,'equidistant');
28 chbres = numquad(inline('sqrt(x)'),0,1,N,'Chebychev');
29 gaures = numquad(inline('sqrt(x)'),0,1,N,'Gauss');
30 loglog(eqdres(:,1),abs(eqdres(:,2)-exact),'b+-',...
31      chbres(:,1),abs(chbres(:,2)-exact),'m+-',...
32      gaures(:,1),abs(gaures(:,2)-exact),'r+-');
33 set(gca,'fontsize',12);
34 axis([1 25 0.000001 1]);
35 title('Numerical quadrature of function sqrt(t)');
36 xlabel('{\bf Number of quadrature nodes}','fontsize',14);
37 ylabel('{\bf |quadrature error|}','fontsize',14);
38 legend('Equidistant Newton-Cotes quadrature',...
39          'Clenshaw-Curtis quadrature',...
40          'Gauss quadrature',1);
```


R. Hiptmair

rev 30401, December 13, 2010

```
eqdp2 = polyfit(log(eqdres(:,1)),log(abs(eqdres(:,2)-exact)),1)
chbp2 = polyfit(log(chbres(:,1)),log(abs(chbres(:,2)-exact)),1)
gaup2 = polyfit(log(gaures(:,1)),log(abs(gaures(:,2)-exact)),1)
print -depsc2 '../PICTURES/numquaderr2.eps';
```


◇

10.5 Adaptive Quadrature

Example 10.5.1 (Rationale for adaptive quadrature).

Consider composite trapezoidal rule (10.3.3) on mesh $\mathcal{M} := \{a = x_0 < x_1 < \cdots < x_m = b\}$:

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" ▷

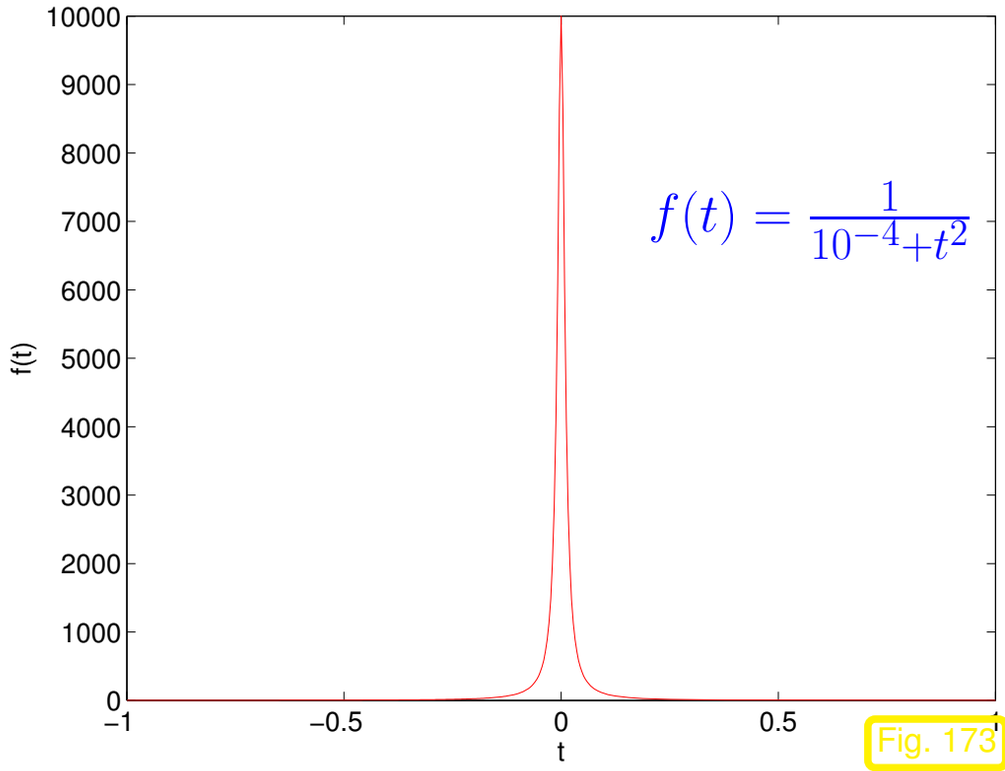


Fig. 173

◇

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* → [14, Sect. 9.7]

R. Hiptmair

rev 30401, September 20, 2010

Idea: local error estimation by comparing local results of two quadrature formulas Q_1, Q_2 of *different* order $\rightarrow$ local error estimates

heuristics: $\text{error}(Q_2) \ll \text{error}(Q_1) \quad \Rightarrow \quad \text{error}(Q_1) \approx Q_2(f) - Q_1(f)$.

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 $p_k := \frac{1}{2}(x_{k-1} + x_k)$)

$$\texttt{EST}_k := \left| \underbrace{\frac{h_k}{6}(f(x_{k-1}) + 4f(p_k) + f(x_k))}_{\text{Simpson rule}} - \underbrace{\frac{h_k}{4}(f(x_{k-1}) + 2f(p_k) + f(x_k))}_{\text{trapezoidal rule on split mesh interval}} \right| . \tag{10.5.2}$$

❷ (Termination)

Simpson rule on $\mathcal{M}$ $\Rightarrow$ preliminary result I

$$\text{If} \quad \sum_{k=1}^{m} \texttt{EST}_k \leq \texttt{RTOL} \cdot I \quad (RTOL := \text{prescribed tolerance}) \quad \Rightarrow \quad \textbf{STOP} \tag{10.5.3}$$

R. Hiptmair

rev 30401, September 20, 2010

❸ (local mesh refinement)

$$\mathcal{S} := \{k \in \{1,\ldots,m\}: \mathtt{EST}_k \geq \eta \cdot \frac{1}{m}\sum_{j=1}^{m} \mathtt{EST}_j\} \;, \quad \eta \approx 0.9 \;. \tag{10.5.4}$$

► new mesh: $\mathcal{M}^* := \mathcal{M} \cup \{p_k: k \in \mathcal{S}\}$.

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))
```

```
2   refcells = find(est_loc > 0.9*sum(est_loc)/length(h));
3   I = adaptquad(f,sort([M,mp(refcells)]),rtol,abstol); %
4 end
```


Comments on Code 10.5.4:

- Arguments: `f` $\hat{=}$ *handle* to function f, `M` $\hat{=}$ initial mesh, `rtol` $\hat{=}$ relative tolerance for termination, `abstol` $\hat{=}$ absolute tolerance for termination, necessary in case the exact integral value $= 0$, which renders a relative tolerance meaningless.
- line 2: compute lengths of mesh-intervals $[x_{j-1}, x_j]$,
- line 3: store positions of midpoints p_j,
- line 4: evaluate function (vector arguments!),
- line 5: local composite trapezoidal rule (10.3.3),
- line 6: local simpson rule (10.2.8),
- line 7: value obtained from composite simpson rule is used as intermediate approximation for integral value,

- line 8: difference of values obtained from local composite trapezoidal rule ($\sim Q_1$) and ocal simpson rule ($\sim Q_2$) is used as an estimate for the local quadrature error.
- line 9: estimate for global error by summing up moduli of local error contributions,
- line 10: terminate, once the estimated total error is below the relative or absolute error threshold,
- line 13 otherwise, add midpoints of mesh intervals with large error contributions according to (10.5.4) to the mesh and continue.

Example 10.5.6 (h-adaptive numerical quadrature).

- approximate $\int_0^1 \exp(6\sin(2\pi t))\,\mathrm{d}t$, initial mesh $\mathcal{M}_0 = \{j/10\}_{j=0}^{10}$

Algorithm: adaptive quadrature, Code 10.5.4

Tolerances: `rtol` $= 10^{-6}$, `abstol` $= 10^{-10}$

We monitor the distribution of quadrature points during the adaptive quadrature and the true and estimated quadrature errors. The "exact" value for the integral is computed by composite Simpson rule on an equidistant mesh with 10^7 intervals.

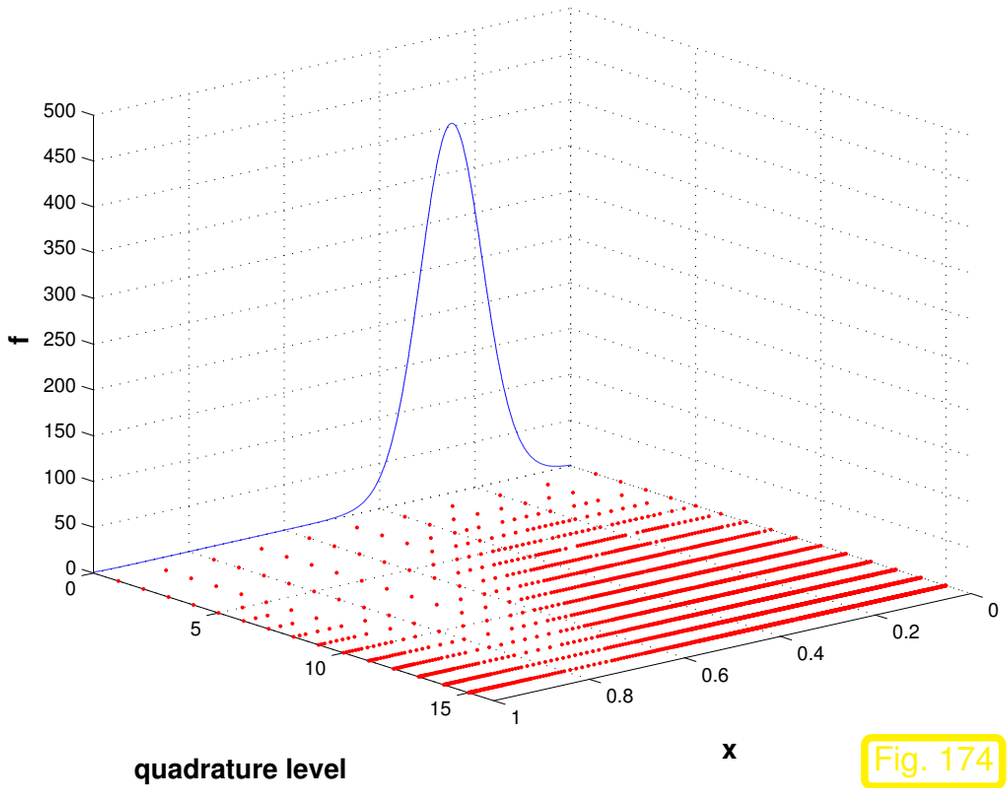


Fig. 174

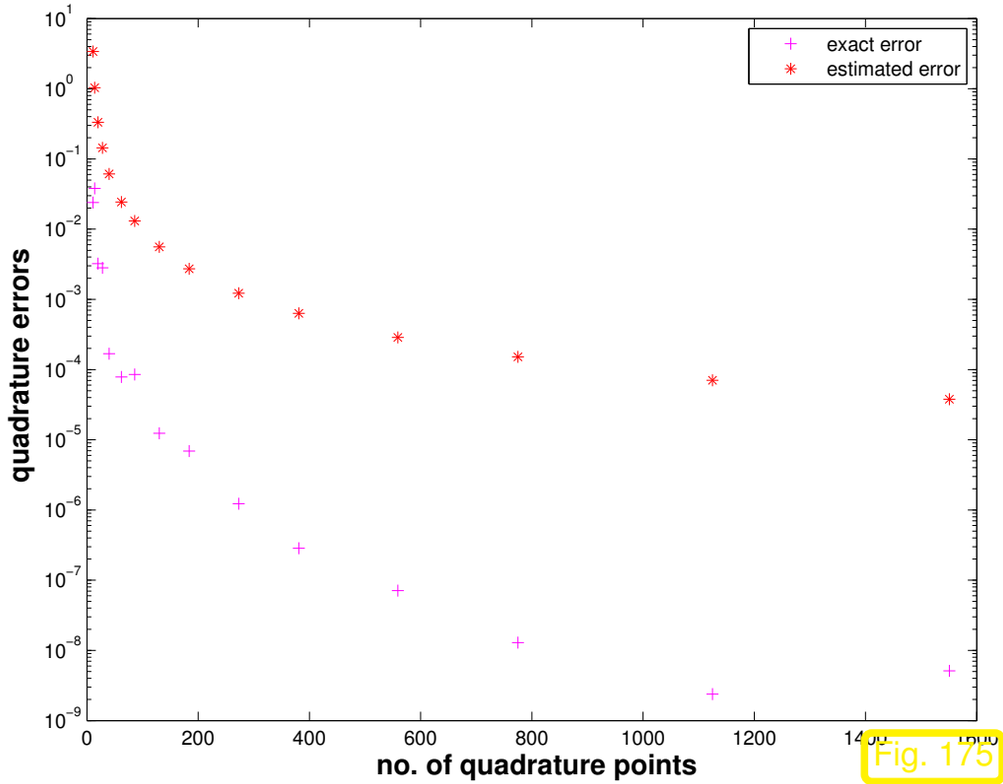


Fig. 175

- approximate $\int_0^1 \min\{\exp(6\sin(2\pi t)), 100\}\,\mathrm{d}t$, initial mesh as above

R. Hiptmair

rev 30401, September 20, 2010

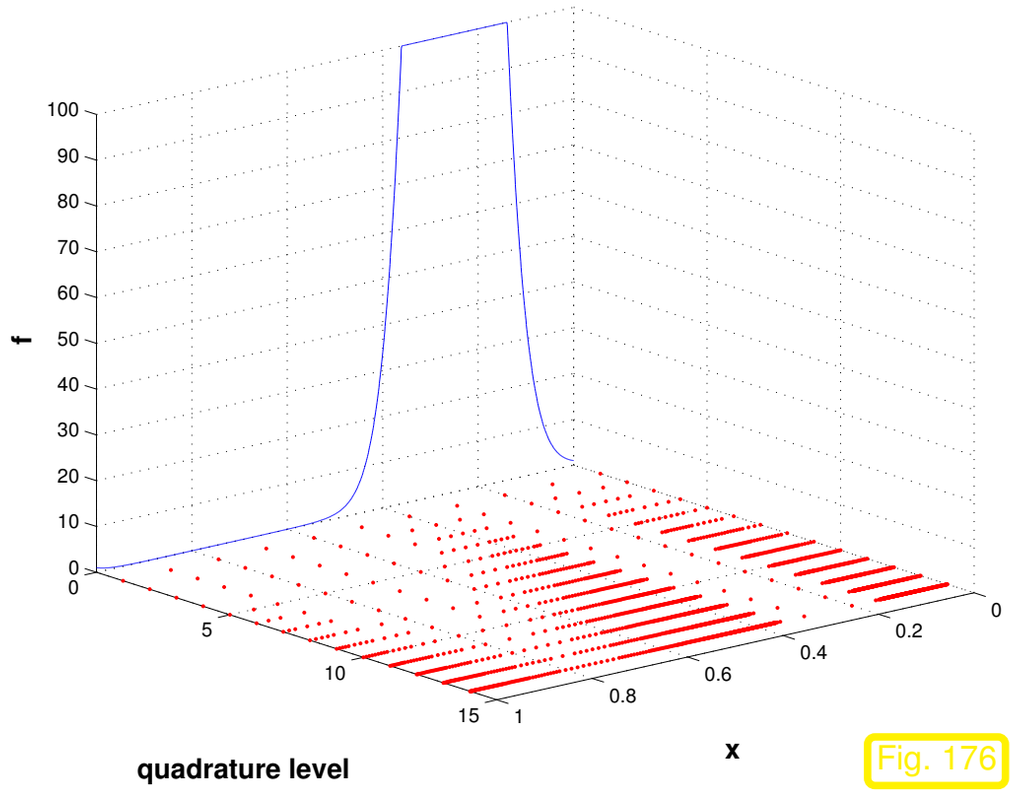


Fig. 176

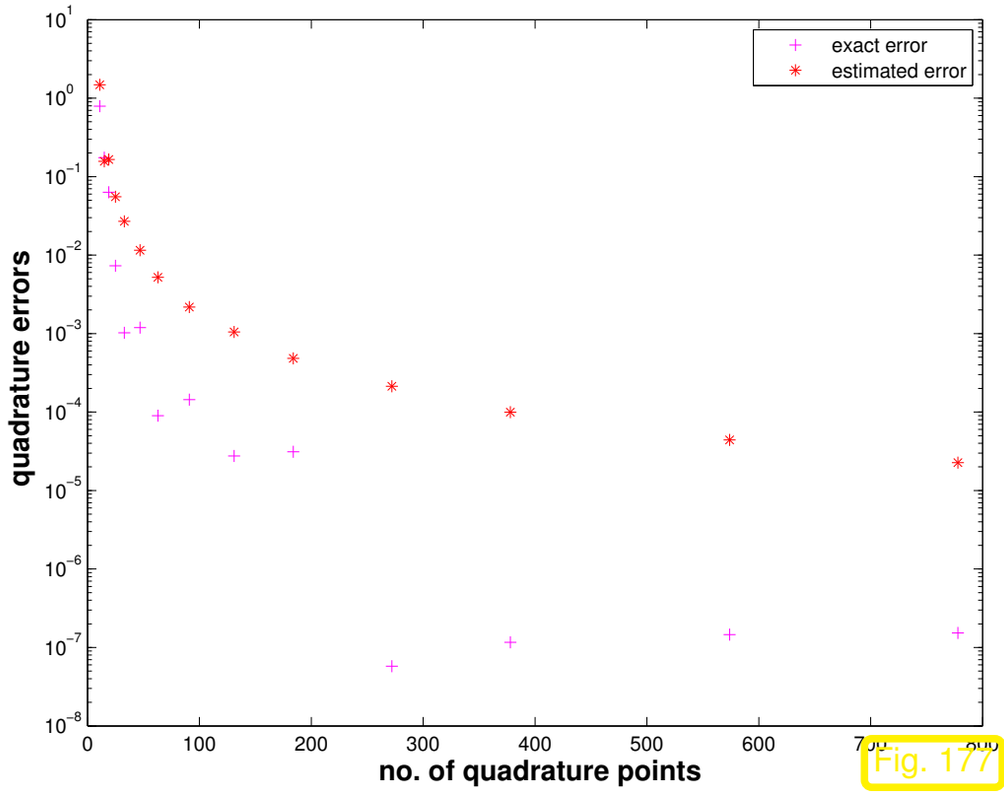


Fig. 177

Observation:

- Adaptive quadrature locally decreases meshwidth where integrand features variations or kinks.
- Trend for estimated error mirrors behavior of true error.
- Overestimation may be due to taking the modulus in (10.5.2)

However, the important information we want to glean from EST_k is about the *distribution* of the quadrature error.

R. Hiptmair

rev 30401, September 20, 2010

Remark 10.5.7 (Adaptive quadrature in MATLAB).

`q = quad(fun,a,b,tol)`: adaptive multigrid quadrature
(local low order quadrature formulas)

`q = quadl(fun,a,b,tol)`: adaptive Gauss-Lobatto quadrature

△

11 Clustering Techniques

11.1 Kernel Matrices

TASK:

Given: Kernel Function $G : I \times J \mapsto \mathbb{C}$, $I, J \subset \mathbb{R}$ Interval: $G(x, y)$ smooth for $x \neq y$

Collocation Points $x_1 < x_2 < \cdots < x_n$, $x_j \in I$, $y_1 < y_2 < \cdots < y_m$, $y_j \in J$

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)

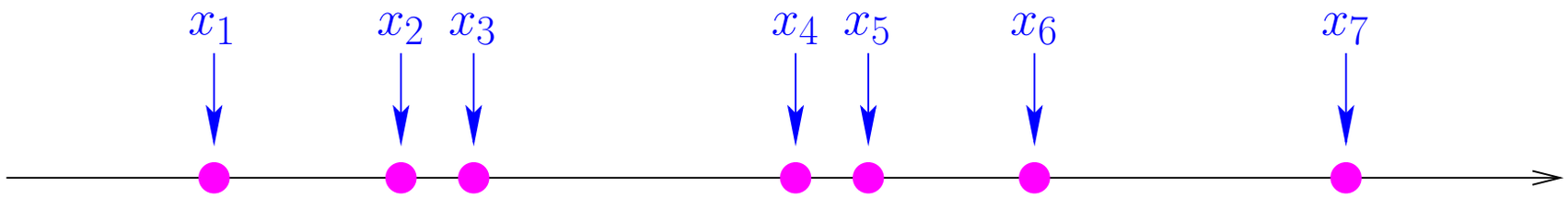
We have to find: *Efficient* algorithms for approximate evaluation of $\mathbf{M} \times$ Vector

(Note: Computational Effort $O(mn)$!)

R. Hiptmair

rev 30401, 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: $f_j = \sum_{\substack{k=1 \\ k\neq j}}^{n} \frac{1}{|x_j - x_k|} c_k c_j \;, \quad j = 1, \ldots, n \;.$
- Force on every wire is given by vector $\mathbf{f} = \operatorname{diag}(c_1, \ldots, c_n)\mathbf{M} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix},$

  where $\mathbf{M} = (m_{ij})_{i,j=1}^{n} \;, \quad m_{ij} = \begin{cases} \frac{1}{|x_j - x_i|} & \text{for } i \neq j \;, \\ 0 & \text{for } i = j \;. \end{cases}$

Collocation matrix $\mathbf{M}$ will be formed using kernel function $G(x, y) = \frac{1}{|x-y|}$

R. Hiptmair

rev 30401, 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. ◇

R. Hiptmair

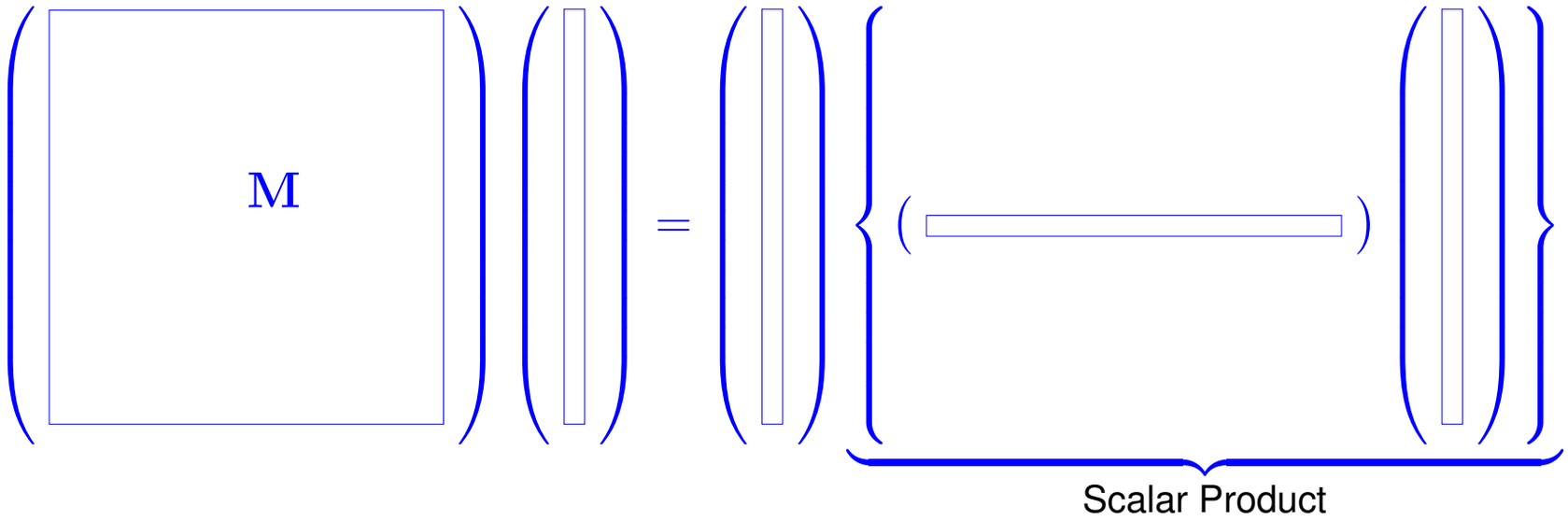
rev 30401, November 15, 2009

11.2 Local Separable Approximation

If kernel function is separable i.e. : $G(x,y) = g(x)h(y), \quad g : I \mapsto \mathbb{C}, h : J \mapsto \mathbb{C}$

$$\mathbf{M} = \left(g(x_j)\right)_{j=1}^{n} \cdot \left(\left(h(y_j)\right)_{j=1}^{m}\right)^T \qquad \operatorname{rank}(\mathbf{M}) = 1 \ .$$

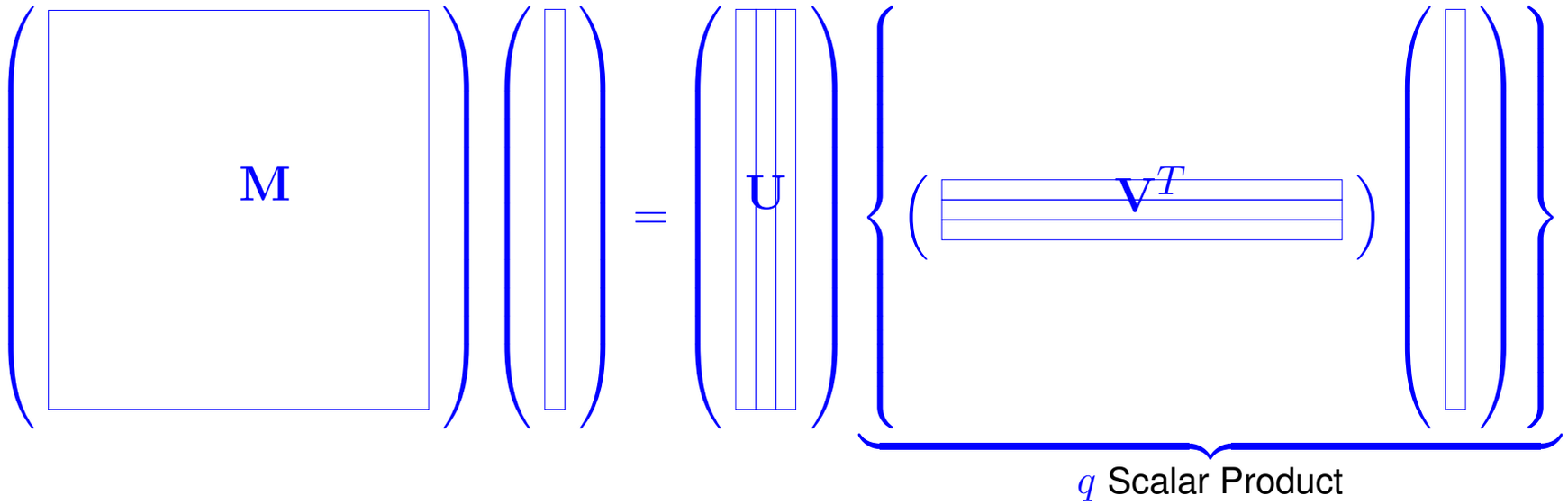
Computational Effort ($\mathbf{M}\times$Vector) = $m+n$



Generalization: $G(x,y) = \sum_{j=1}^{q} g_j(x)h_j(y), \quad g_j : I \mapsto \mathbb{C}, h_j : J \mapsto \mathbb{C}, q \in \mathbb{N}$

$$\mathbf{M} = \mathbf{U}\mathbf{V}^T \ , \quad \begin{array}{l} \mathbf{U} \in \mathbb{R}^{n,q} \ , \quad u_{ij} = g_j(x_i) \ , \quad j \in \{1,\ldots,q\}, i \in \{1,\ldots,n\} \ , \\ \mathbf{V} \in \mathbb{R}^{q,m} \ , \quad v_{ij} = h_i(y_j) \ , \quad i \in \{1,\ldots,q\}, j \in \{1,\ldots,m\} \ . \end{array}$$

If $\mathrm{rank}(\mathbf{M}) = q$ then Computational Effort ($\mathbf{M}\times$ Vector) $= q(m+n)$



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 $\widetilde{m}_{ij} = \widetilde{G}(x_i, y_j)$

R. Hiptmair

rev 30401, November 15, 2009

Remark 11.2.2 (Quality measure for kernel approximation).

$$\left\|\mathbf{M}-\widetilde{\mathbf{M}}\right\|_2^2 \leq \left\|\mathbf{M}-\widetilde{\mathbf{M}}\right\|_F^2 = \sum_{i,j}(m_{ij}-\tilde{m}_{ij})^2 = \boxed{\sum_{i,j}(G(x_i,y_j)-\widetilde{G}(x_i,y_j))^2} \ .$$

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

Definition 11.2.4 (Tensor product interpolation polynomial). $L_j^x \in \mathcal{P}_n$ ($L_k^y \in \mathcal{P}_m$), $j=0,\ldots,n$ ($k=0,\ldots,m$), *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.

R. Hiptmair

rev 30401, November 15, 2009

$$G(x,y) \approx \widetilde{G}(x,y) := \sum_{j=0}^{d} \sum_{k=0}^{d} G(t_j^x, t_k^y) L_j^x(x) L_k^y(y) \;, \tag{11.2.5}$$

$t_0^x, \ldots, t_d^x$/$t_0^y, \ldots, t_d^y$ Chebyshev-nodes in I/J, and L_j^x, L_k^y subordinate Lagrange-polynomial.

Example 11.2.6 (Global separable approximation by smooth kernel function).

- Smooth (even analytic) kernel function $G(x,y) = \frac{1}{1+|x-y|^2}$, collocation points $x_i = y_i = \frac{i-1}{n}$, $i = 1,\ldots,n$.
- $\widetilde{G}$ 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 $n \in \{100, 200, 400\}$

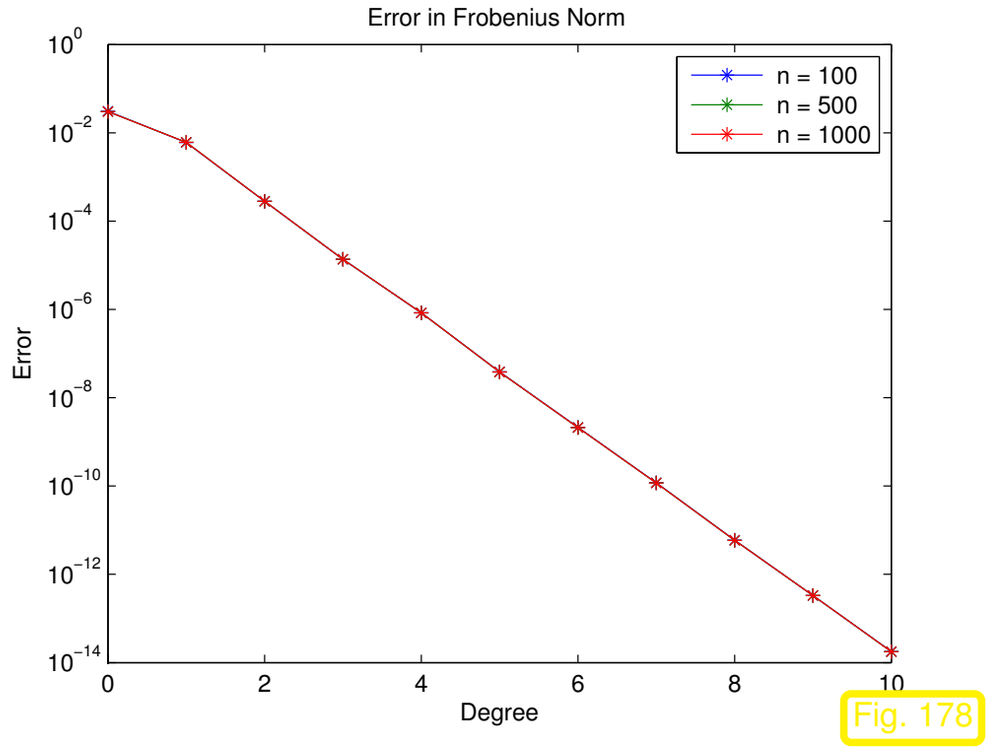


Fig. 178

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

R. Hiptmair

rev 30401, November 15, 2009

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.

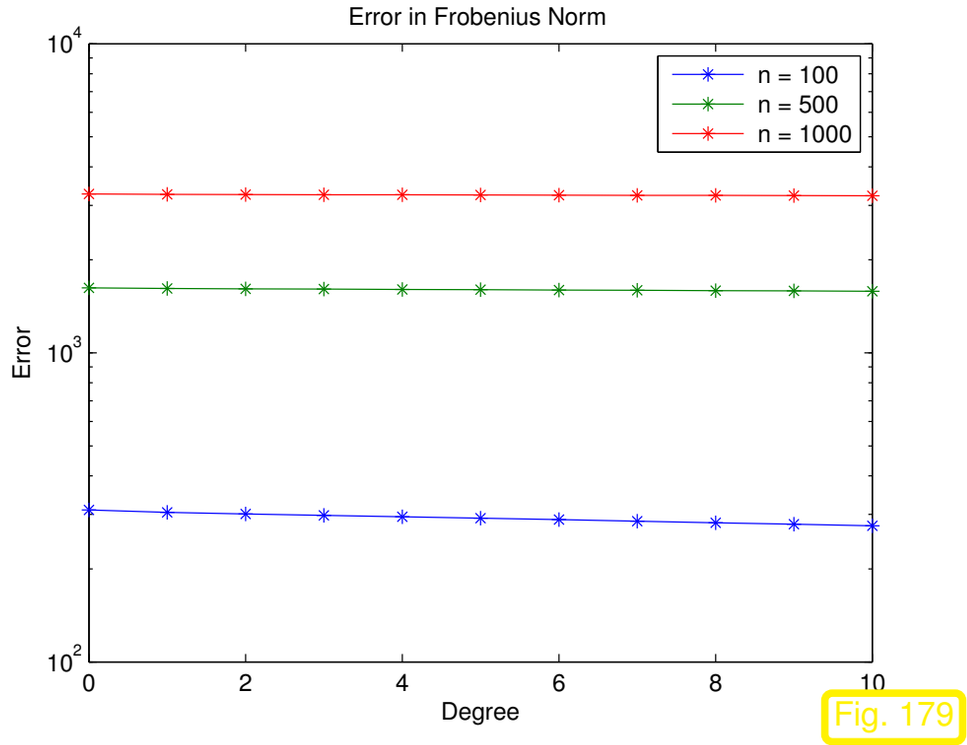


Fig. 179

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

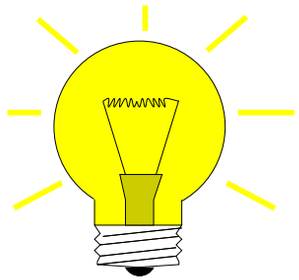
Note: (Virtually) No Convergence $\left\|\mathbf{M}-\widetilde{\mathbf{M}}\right\|_F \to 0$ for $d \to \infty$ ◇

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 $\widetilde{I} \times \widetilde{J} \cap \{(x,y) : x = y\} = \emptyset$.

R. Hiptmair

rev 30401, November 15, 2009

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)

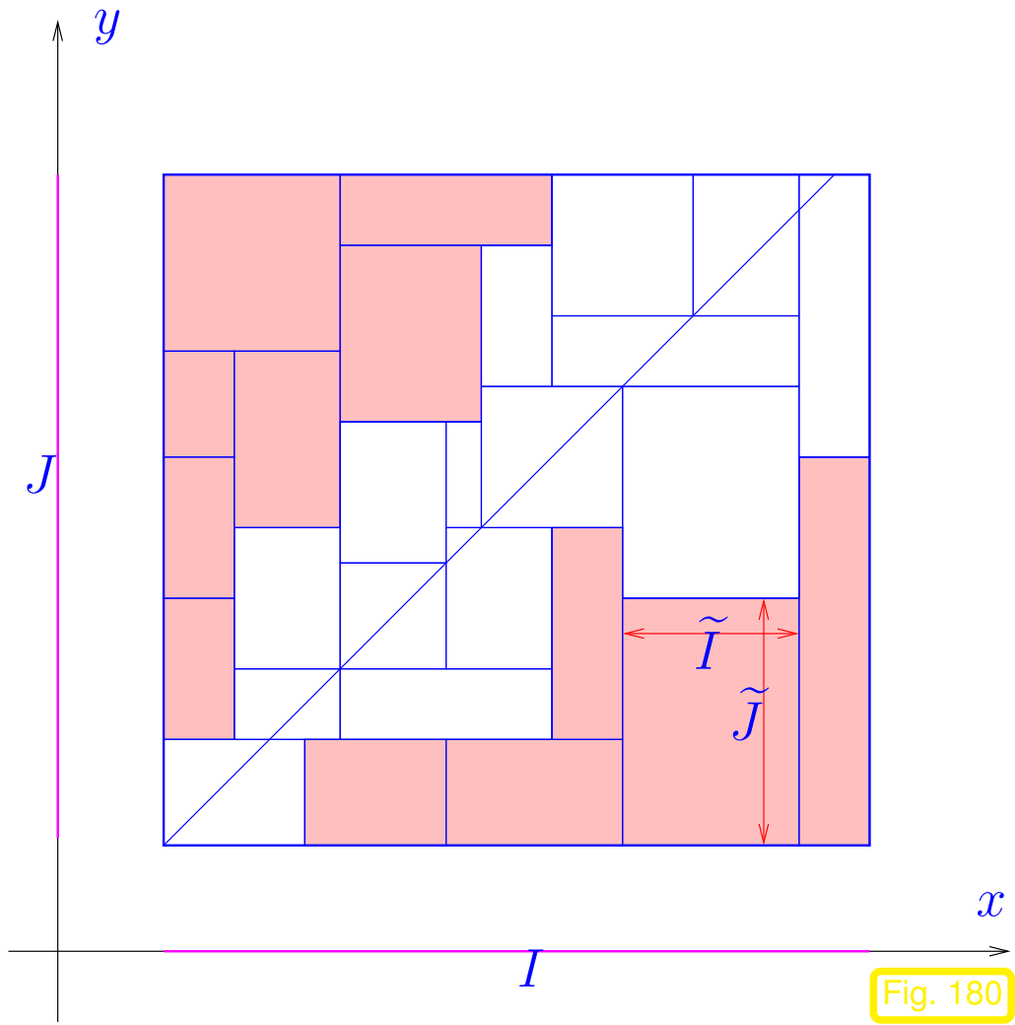


Fig. 180

R. Hiptmair

rev 30401, November 15, 2009

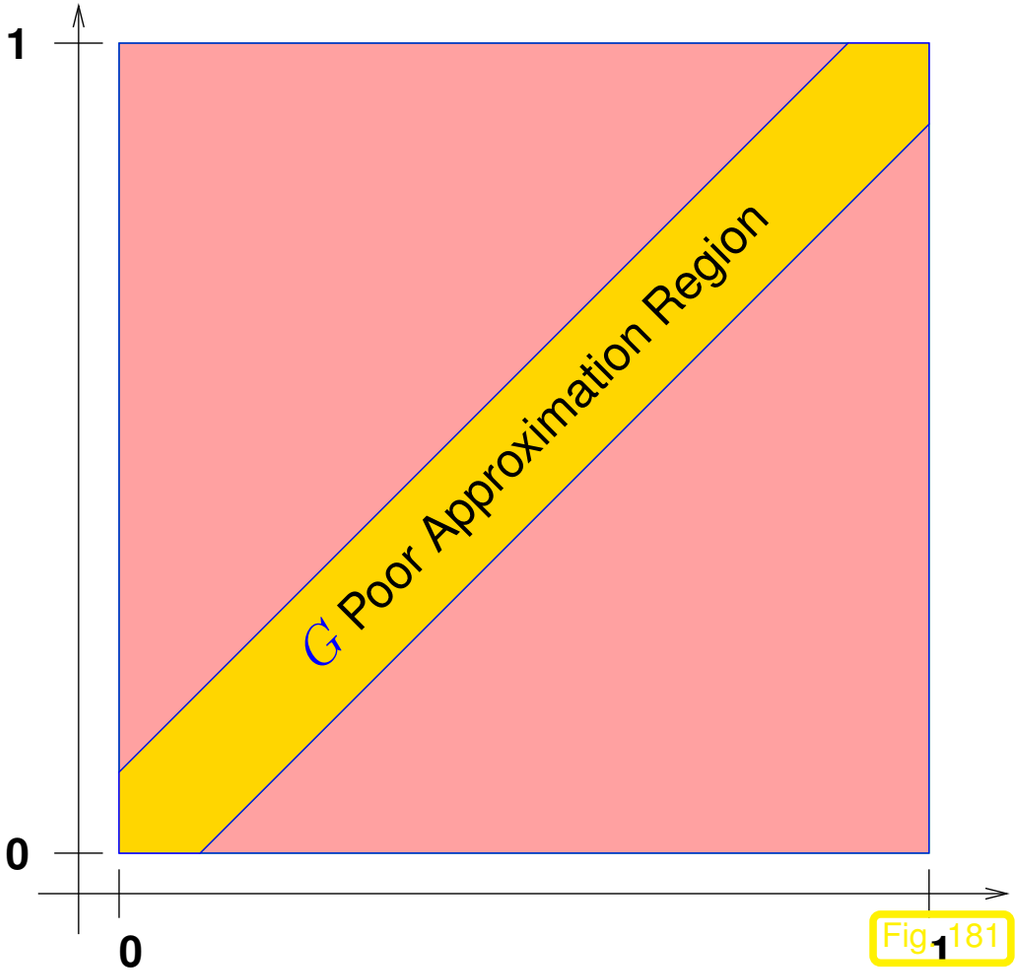


Fig. 181

Terminology:

Near Field: G Poor approximation

Far Field: G Good approximation

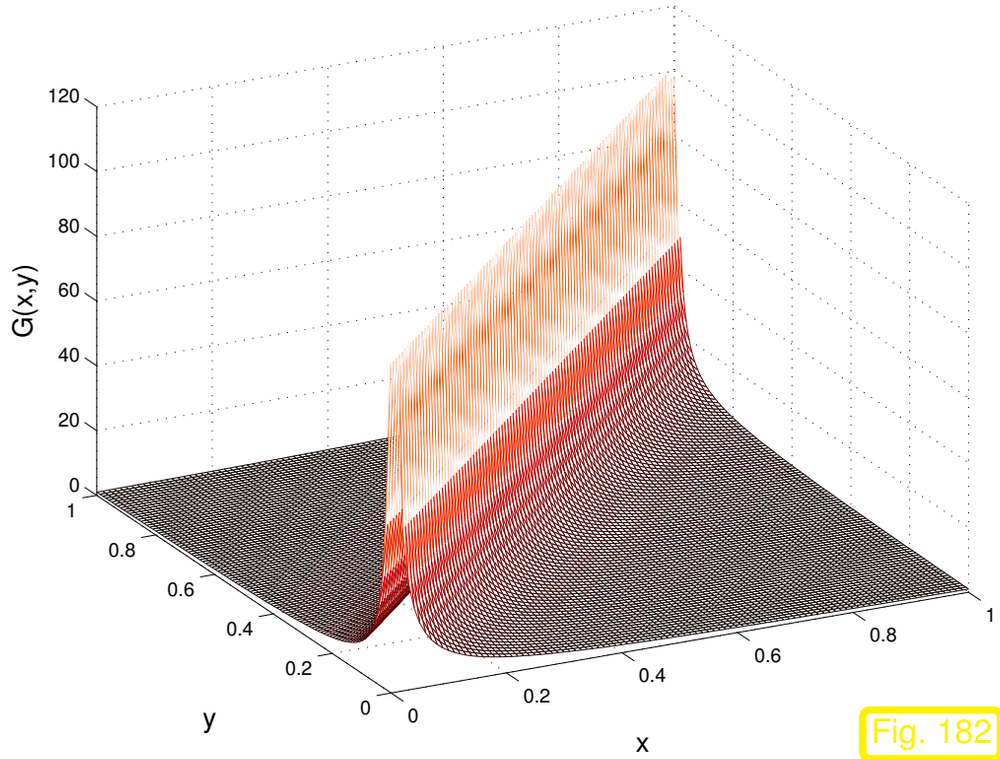


Fig. 182

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 30401, November 15, 2009

"“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

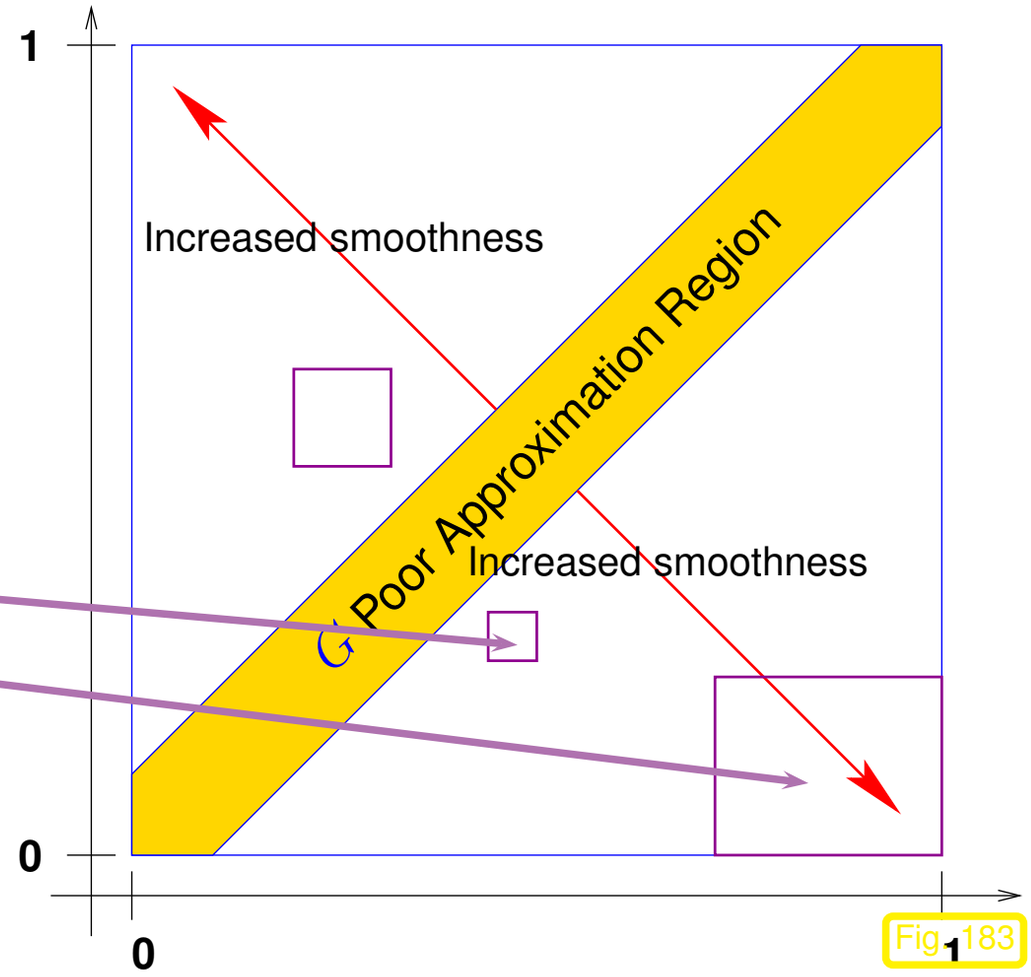


Fig. 183

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 $\{(x,y)\colon x = y\}$

$$\widetilde{I}_k = [0.55 + k \cdot 0.05, 0.75 + k \cdot 0.05] \quad , \quad \widetilde{J}_k = [0.25 - k \cdot 0.05, 0.45 - k \cdot 0.05] \, , \quad k \in \{0, \ldots, 5\} \, .$$

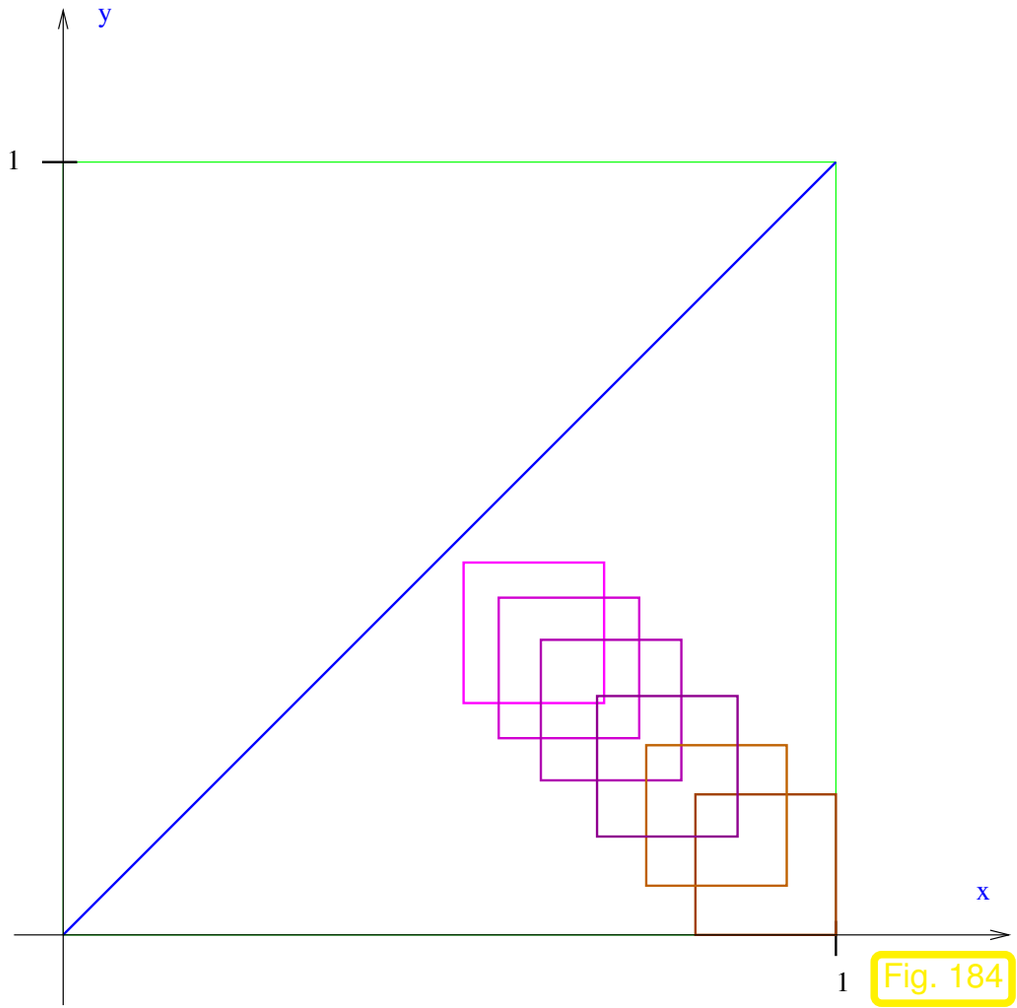


Fig. 184

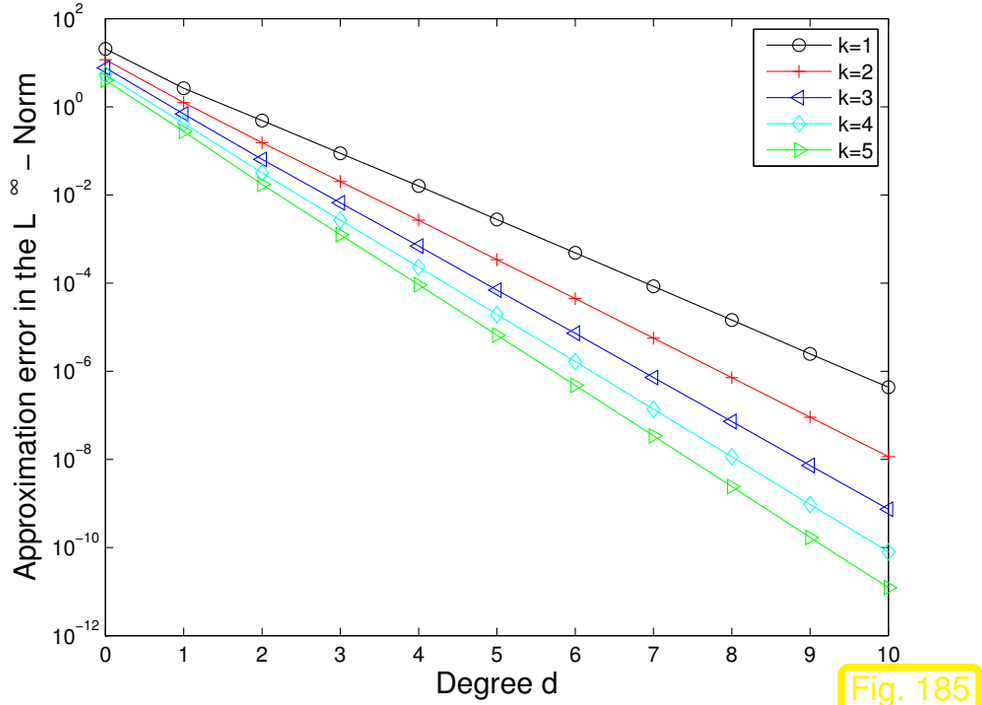


Fig. 185

NOTE: Decreasing interpolation errors with increasing distance from $\{(x,y)\colon x=y\}$

◇

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

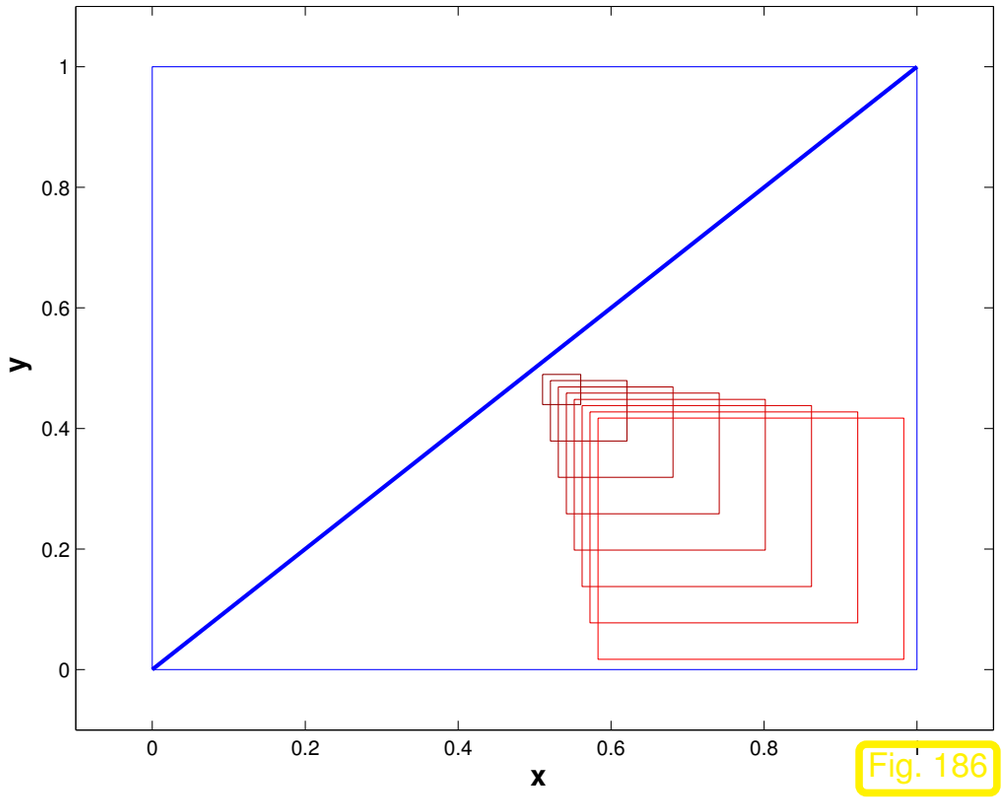


Fig. 186

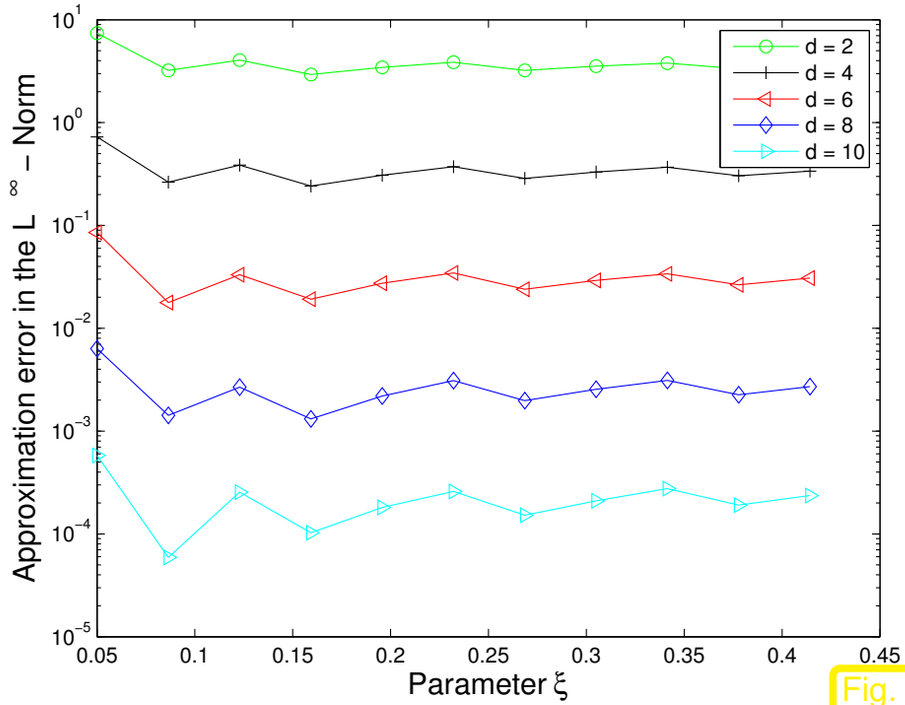


Fig. 187

NOTE: Controlled decrease of interpolation errors with increase in size of rectangles

◇

Acceptability Criteria

distance from diagonal

$[a,b] \times [c,d]$ is called η- *admissible*, $\eta > 0$, if

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:

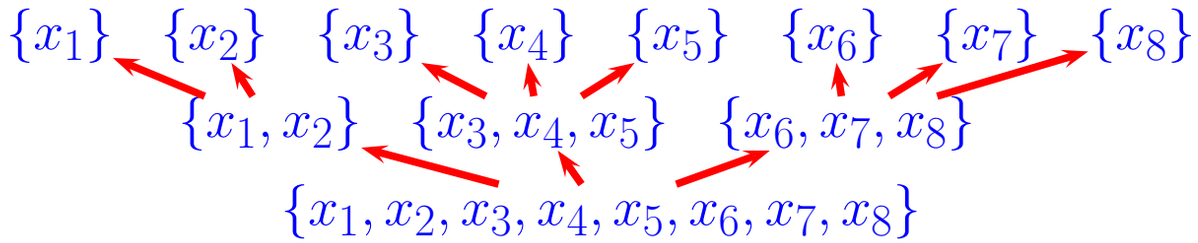
$$\begin{array}{c}\text{Partitioning}\\ \text{of } I \times J\end{array} \Longrightarrow \begin{array}{c}\text{Partitioning}\\ \text{of } \{x_i\}_{i=1}^n \times \{y_j\}_{j=1}^m\end{array} \Leftrightarrow \begin{array}{c}\text{Partitioning of}\\ \text{index mesh } \{1,\ldots,n\} \times \{1,\ldots,m\}\end{array}$$

△

Definition 11.3.2 (Cluster Tree)**.** *A Tree ($\rightarrow$ Computer Science,Graph theory) T is called Cluster Tree on $\mathbb{P} := \{x_1,\ldots,x_n\} \subset \mathbb{R}$:$\Leftrightarrow$*

- *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: $\sigma \in T$: $\mathrm{sons}(\sigma) = \emptyset \quad :\Leftrightarrow \quad \sigma$ 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);
```


R. Hiptmair

rev 30401, December 8, 2010

C++ code: Construction of Cluster Trees

```
1  void clsClusteringApproximation::build_ClusterTree_Linear(int iOffset, int
     iNumberofPtsInClusterLinear, int* piDimension){
2     int iTempLeftChild, iTempRightChild, iStart, iEnd;
3     int iStartLeft, iPtsInLeft, iStartRight, iPtsInRight;
4
5     iStart = iOffset;
6     iEnd = iOffset + iNumberofPtsInClusterLinear - 1;
7     if (iNumberofPtsInClusterLinear == 1){
8        add_ClusterLinear(*piDimension, iStart, iEnd, -1, -1);
9     }
0     else{
1        iStartLeft = iOffset;
2        iPtsInLeft =  iNumberofPtsInClusterLinear - (iNumberofPtsInClusterLinear/2);
3        build_ClusterTree_Linear(iStartLeft, iPtsInLeft, piDimension);
4        iTempLeftChild = vec2pclsClusterLinear[*piDimension].size() - 1;
5
6        iStartRight = iOffset + iPtsInLeft;
7        iPtsInRight = iNumberofPtsInClusterLinear/2;
8        build_ClusterTree_Linear(iStartRight, iPtsInRight, piDimension);
9        iTempRightChild = vec2pclsClusterLinear[*piDimension].size() - 1;
0
1        add_ClusterLinear(*piDimension, iStart, iEnd, iTempLeftChild, iTempRightChild);
2     }
3  }
```

R. Hiptmair

rev 30401, December 8, 2010

Example 11.3.3 (Cluster Tree).

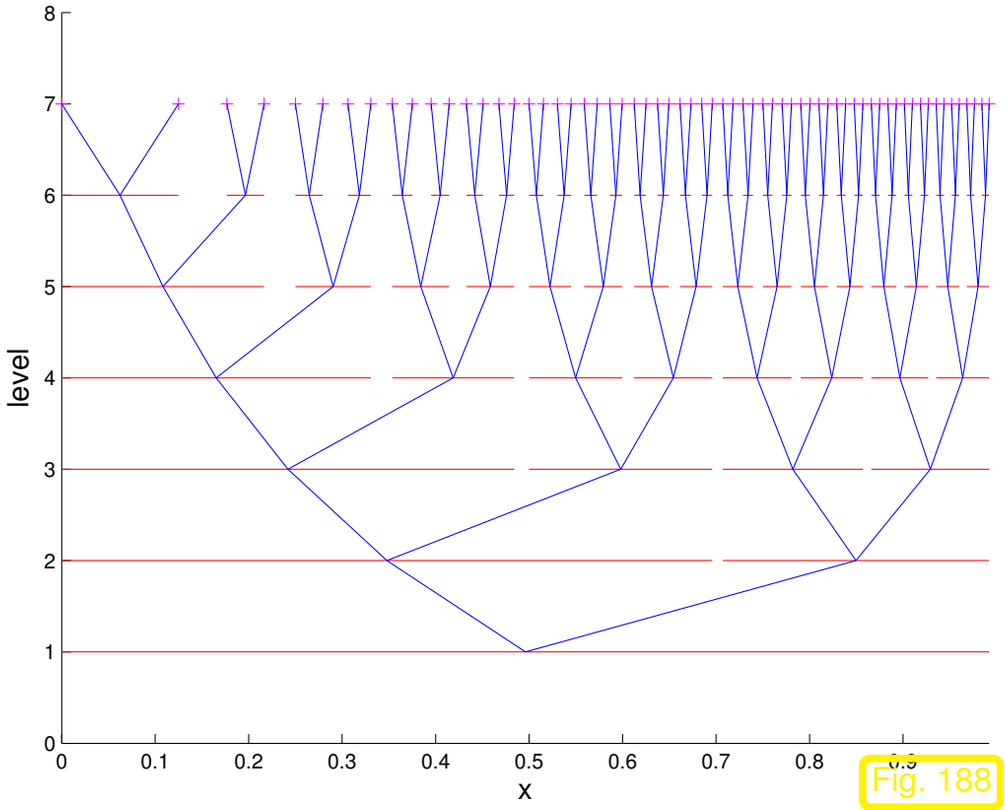


Fig. 188

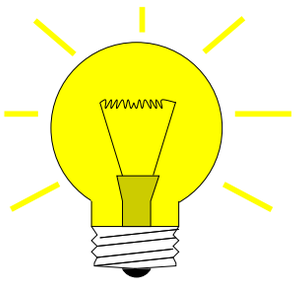
← Binary cluster tree:

```
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 30401, December 8, 2010

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);
```


R. Hiptmair

rev 30401, December 8, 2010

C++ code: Construction for Admissible Partitioning

```
1  void clsClusteringApproximation::build_ClusterPair (int iIndex1,
     int iIndex2){
2
3        clsClusterLinear *pclsClusterLinear1, *pclsClusterLinear2;
4
5        pclsClusterLinear1 = vec2pclsClusterLinear[0][iIndex1];
6        pclsClusterLinear2 = vec2pclsClusterLinear[1][iIndex2];
7        if (leaf(0,iIndex1) || leaf(1,iIndex2)){
8              add_ClusterPairNear(pclsClusterLinear1,
                 pclsClusterLinear2);
9             (pclsClusterLinear1->get_ptr_AppearsIn())->push_back(-vecp
10            (pclsClusterLinear2->get_ptr_AppearsIn())->push_back(-vecp
11       }
12       else if (admissible_ClusterPair(iIndex1, iIndex2)){
13             add_ClusterPairFar(pclsClusterLinear1,
                 pclsClusterLinear2);
14            (pclsClusterLinear1->get_ptr_AppearsIn())->push_back(vecpo
15            (pclsClusterLinear2->get_ptr_AppearsIn())->push_back(vecpo
16       }
17       else{
18             tree_traverse(pclsClusterLinear1, pclsClusterLinear2);
19       }
20 }
```

C++ code: Preprocessing Code

```
1 void clsClusteringApproximation::preprocess(){
2       int iLoopVari, iLoopVarj;
3       for (iLoopVari = 0; iLoopVari < iDimension; iLoopVari++){
4            build_ClusterTree_Linear(0, veciNumberofPts[iLoopVari],
               &iLoopVari);
5       }
6       build_ClusterPair(vec2pclsClusterLinear[0].size() - 1,
          vec2pclsClusterLinear[1].size() - 1);
7 }
```


R. Hiptmair

rev 30401, December 8, 2010

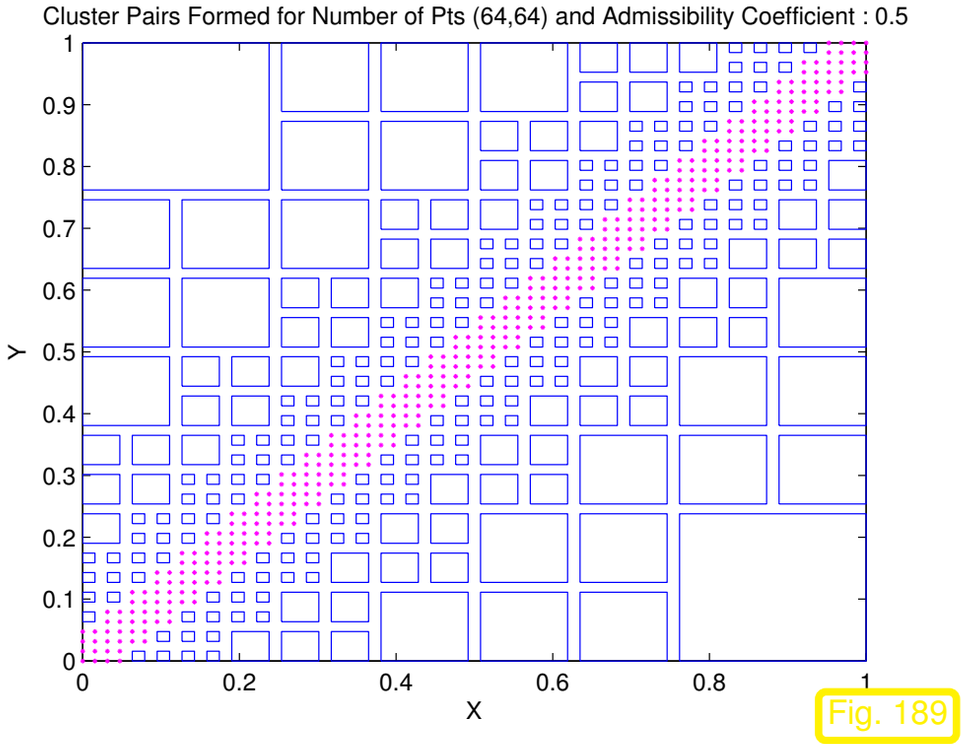


Fig. 189

```
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)

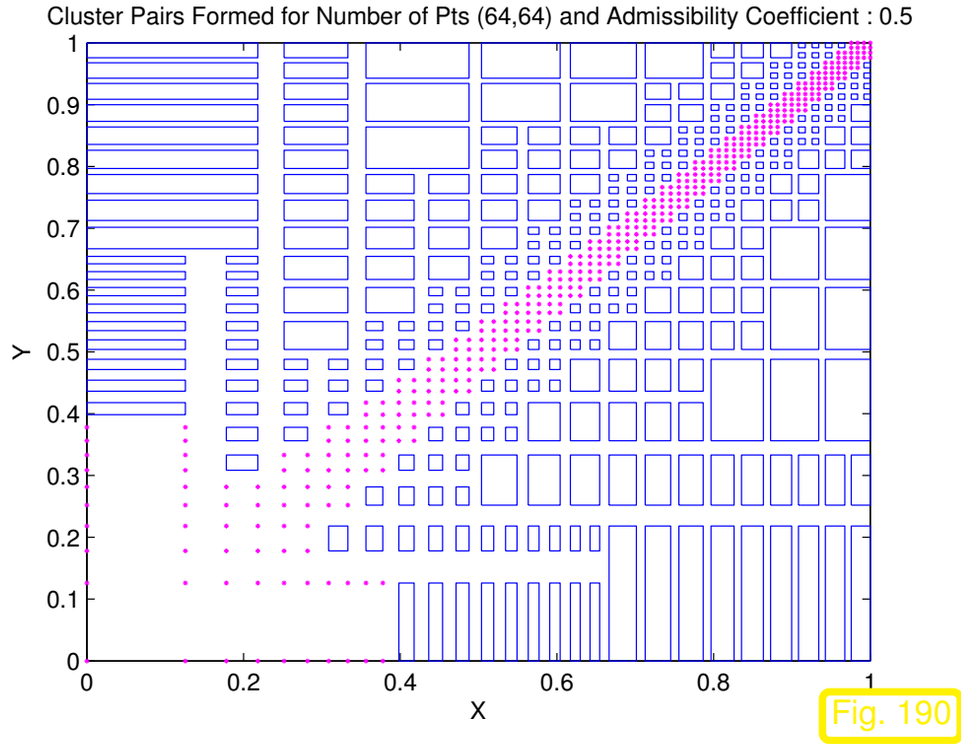


Fig. 190

```
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 $\in \{\Gamma(\sigma)\colon \sigma \in T\}$, $T \mathrel{\hat{=}}$ 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. 189,190

Notation:

$$\begin{array}{l}\sigma := \{i_1,\ldots,i_2\} \subset \{1,\ldots,n\} \\ \mu := \{j_1,\ldots,j_2\} \subset \{1,\ldots,m\}\end{array} \quad \Rightarrow \quad \mathbf{M}_{|\sigma\times\mu} := (m_{ij})_{\substack{i=i_1,\ldots,i_2 \\ j=j_1,\ldots,j_2}} .$$

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)):

$$G(x,y) \approx \widetilde{G}(x,y) := \sum_{j=0}^{d} \sum_{k=0}^{d} G(t_j^x, t_k^y) L_j^x(x) L_k^y(y) \ ,$$

- $t_0^x, \ldots, t_d^x / t_0^y, \ldots, t_d^y \hat{=}$ 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.

$$\widetilde{G}(x,y) = \sum_{j=0}^{d} \sum_{k=0}^{d} G(t_j^x, t_k^y) L_j^x(x) L_k^y(y) \ , \quad x \in [x_{i_1}, x_{i_2}], \, y \in [y_{j_1}, y_{j_2}] \ .$$

$$\widetilde{\mathbf{M}}_{|\sigma \times \mu} = \underbrace{\underbrace{\left(L_j^x(x_i)\right)_{\substack{i=i_1,\ldots,i_2 \\ j=0,\ldots,d}}}_{\in \mathbb{R}^{\sharp\sigma, d+1}} \underbrace{\left(G(t_j^x, t_k^y)\right)_{j,k=0,\ldots,d}}_{\in \mathbb{R}^{d+1,d+1}} \underbrace{\left(L_k^y(y_j)\right)_{\substack{k=0,\ldots,d \\ j=j_1,\ldots,j_2}}}_{\in \mathbb{R}^{d+1,\sharp\mu}}}_{\text{rank} \leq d+1} \ .$$

$$\widetilde{\mathbf{M}}_{|\sigma \times \mu} = \left(\quad \right) \left(\quad \right) \left(\quad \right) \ .$$

R. Hiptmair

rev 30401, October 28, 2009

Application of partitioning

$$\{x_1,\ldots,x_n\}\times\{y_1,\ldots,y_m\} = \left(\bigcup_{(\sigma,\mu)\in P^{\mathrm{near}}} \sigma\times\mu\right) \cup \left(\bigcup_{(\sigma,\mu)\in P^{\mathrm{far}}} \sigma\times\mu\right) .$$

Algorithm: Analysis of $\mathbf{f}=\mathbf{Mc}$, $\mathbf{M}$ as (11.1.1)

$$f_i = \sum_{j=1}^{m} G(x_i,y_j)c_j = \underbrace{\sum_{j\in P^{\mathrm{near}}(i)} G(x_i,y_j)c_j}_{\text{Near field contribution}} + \underbrace{\sum_{\substack{\sigma\in T_x\\ x_i\in\sigma}} \sum_{\substack{\mu\in T_y\\ (\sigma,\mu)\in P^{\mathrm{far}}}} \sum_{\substack{1\le j\le m\\ y_j\in\mu}} G(x_i,y_j)c_j}_{\text{Far field contribution}} .$$

Near field coupling indices $P^{\mathrm{near}}(i) := \{j\in\{1,\ldots,m\}: (\{x_i\}\times\{y_j\})\in P^{\mathrm{near}}\}$

R. Hiptmair

rev 30401, October 28, 2009

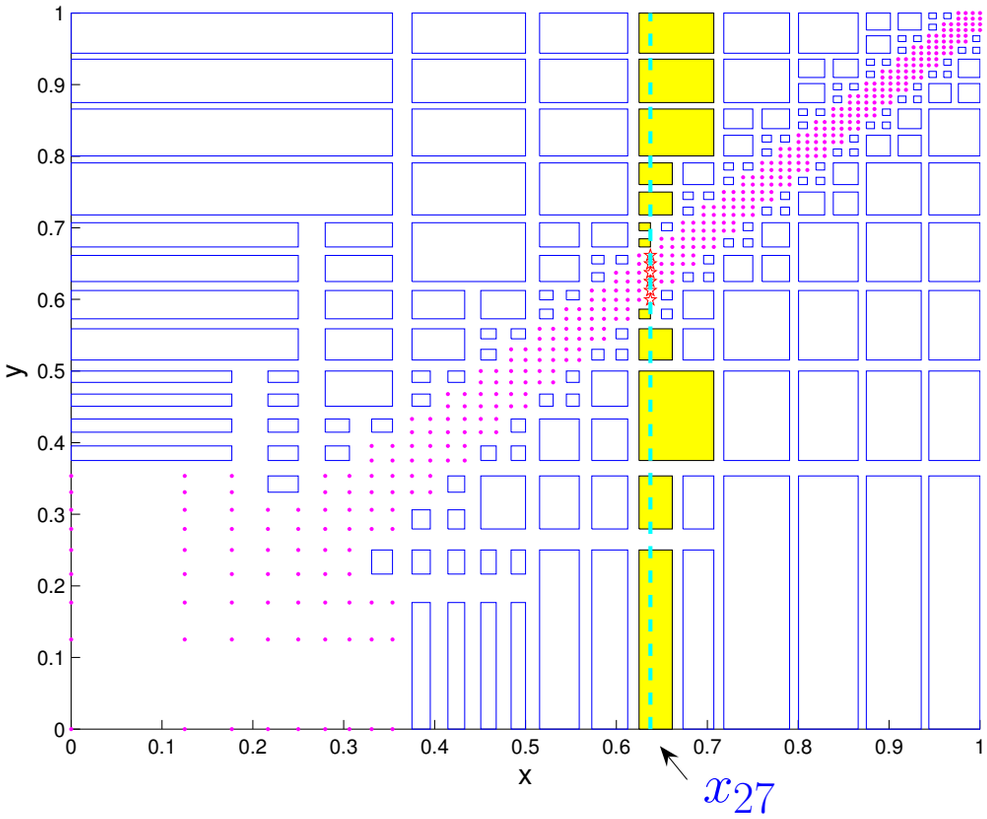


Illustration:

`x = sqrt(0:1/64:1);`, $i = 27$, $\eta = 1$:

$\longrightarrow$ Far field clusters with contribution to f_i

$\star\!\!\longrightarrow$ 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 30401, October 28, 2009

with

$$
\begin{aligned}
t_l^\sigma, t_k^\mu &:= \text{Chebyshev nodes (9.2.12) in } \Gamma(\sigma), \Gamma(\mu)\ , \quad l = 0, \ldots, d\ , \\
\mathbf{X}_{\sigma,\mu} &:= \left(G(t_l^\sigma, t_k^\mu)\right)_{l,k=0}^{d} \in \mathbb{R}^{d+1,d+1}\ , && \text{(11.4.1)} \\
\mathbf{V}_\sigma &:= \left(L_l^\sigma(x_i)\right)_{\substack{i:\, x_i \in \sigma \\ l=0,\ldots,d}} \in \mathbb{R}^{\sharp\sigma,d+1}\ , && \text{(11.4.2)} \\
\mathbf{V}_\mu &:= \left(L_k^\mu(y_j)\right)_{\substack{j:\, y_j \in \mu \\ k=0,\ldots,d}} \in \mathbb{R}^{\sharp\mu,d+1}\ , && \text{(11.4.3)}
\end{aligned}
$$

Analysis of Complexity (under the assumption $n \approx m$):

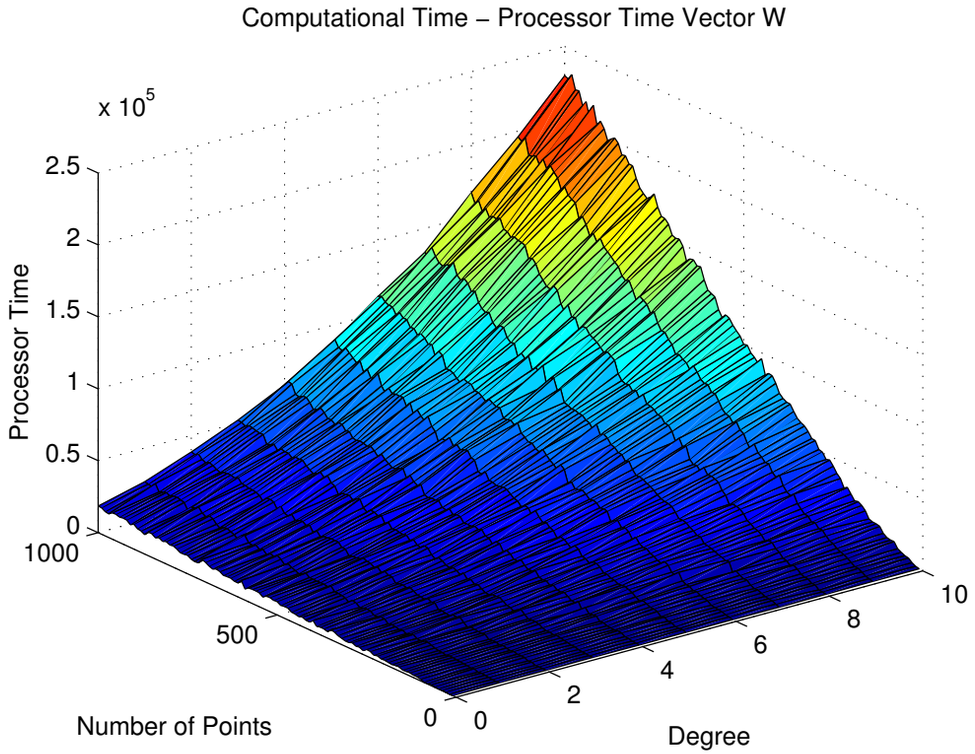
①: Trick: Calculate on above assumption

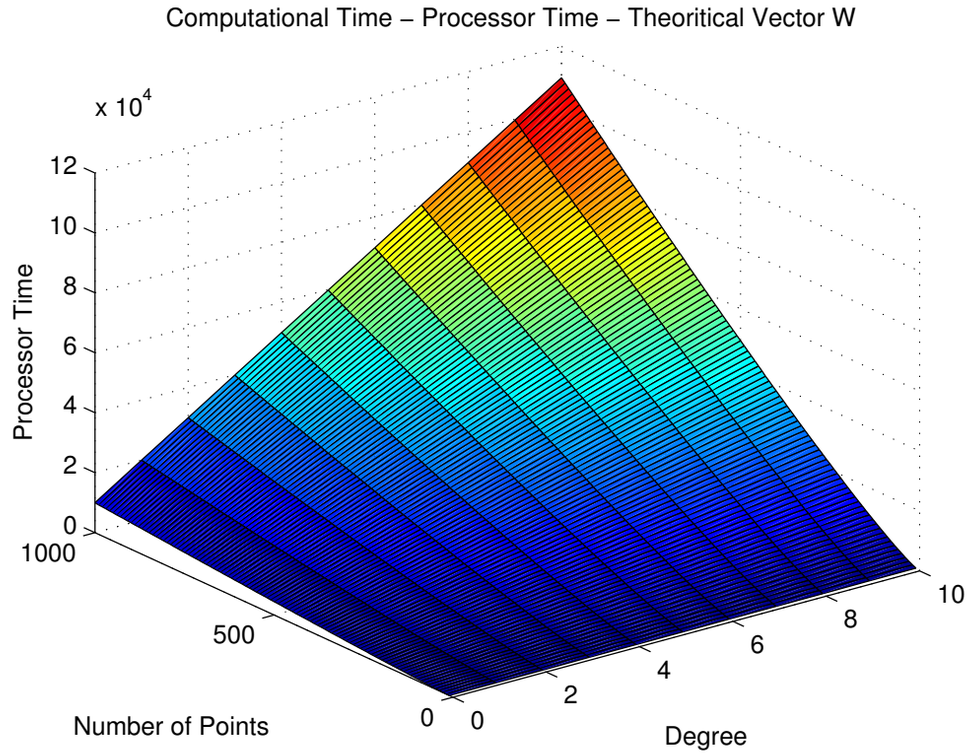
Compute $\mathbf{w}_\mu := \mathbf{V}_\mu^T \mathbf{c}_{|\mu} \in \mathbb{R}^{d+1}$, $\mu \in T_y$ → $O(n)$ Operations on every level of T_y

Computational Effort $O((d+1)n\log_2 n)$, Memory Required $O((d+1)n\log_2 n)$

R. Hiptmair

rev 30401, 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

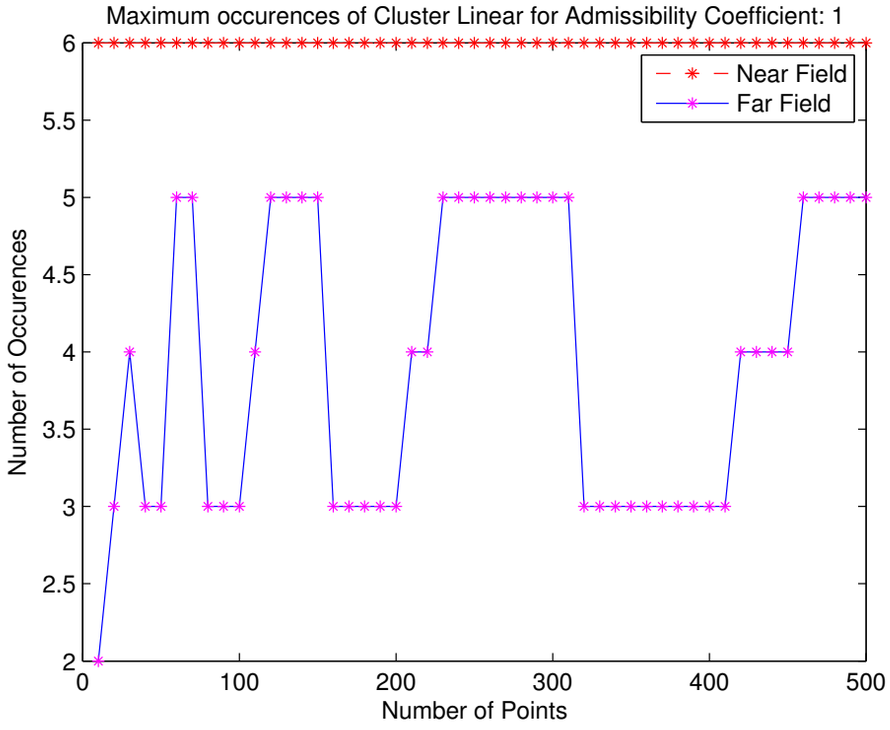
1) Run main.cpp, input as follows : Choice - 2, NumberofPts - 10 1000 10, Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1
Degree - 0 10 1, Admissibility Coefficient - 0.5 0.5 1.0

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

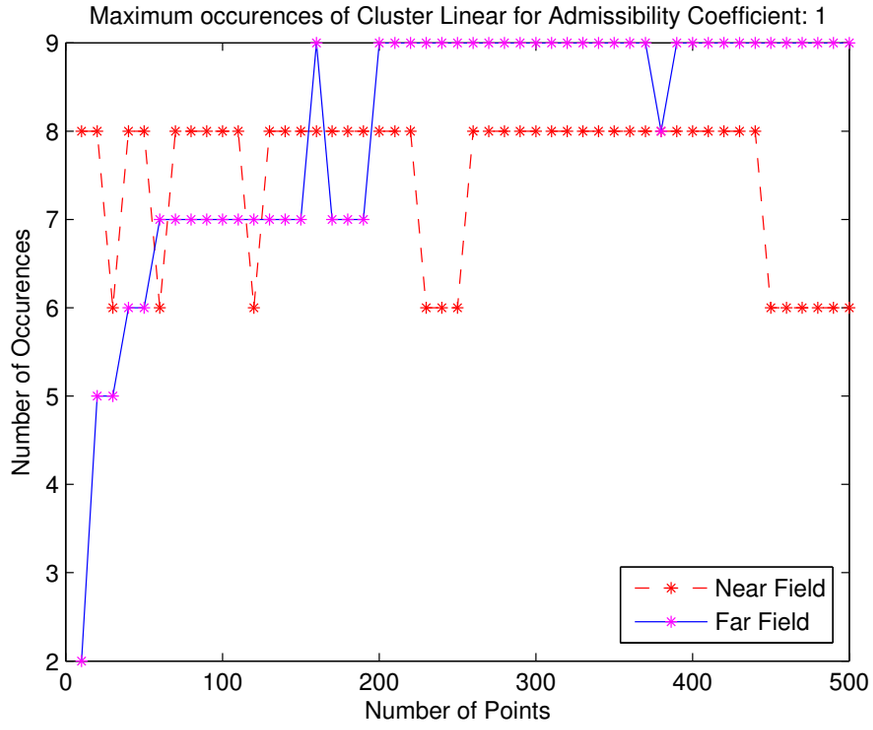
Degree - 0 0 1, Admissibility Coefficient - 1 1 1

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.)

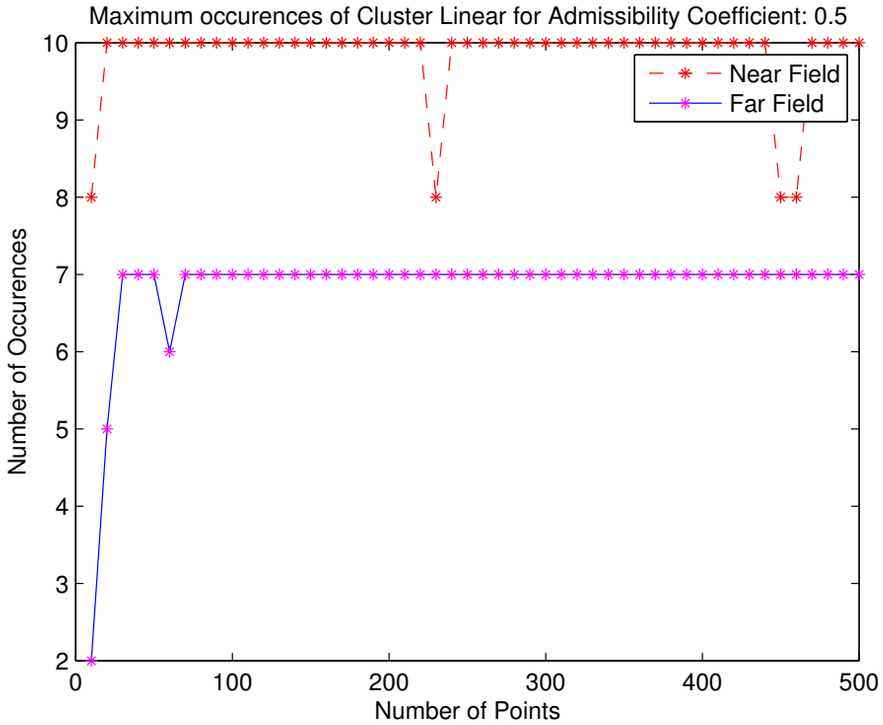
R. Hiptmair

rev 30401, October 28, 2009

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

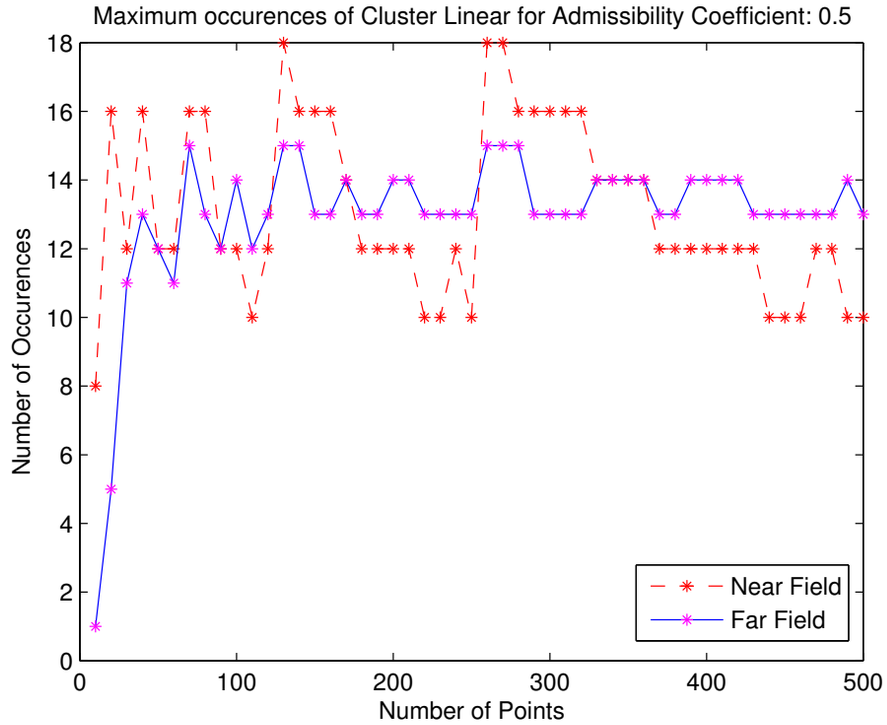
Degree - 0 0 1, Admissibility Coefficient - 0.5 0.5 1

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

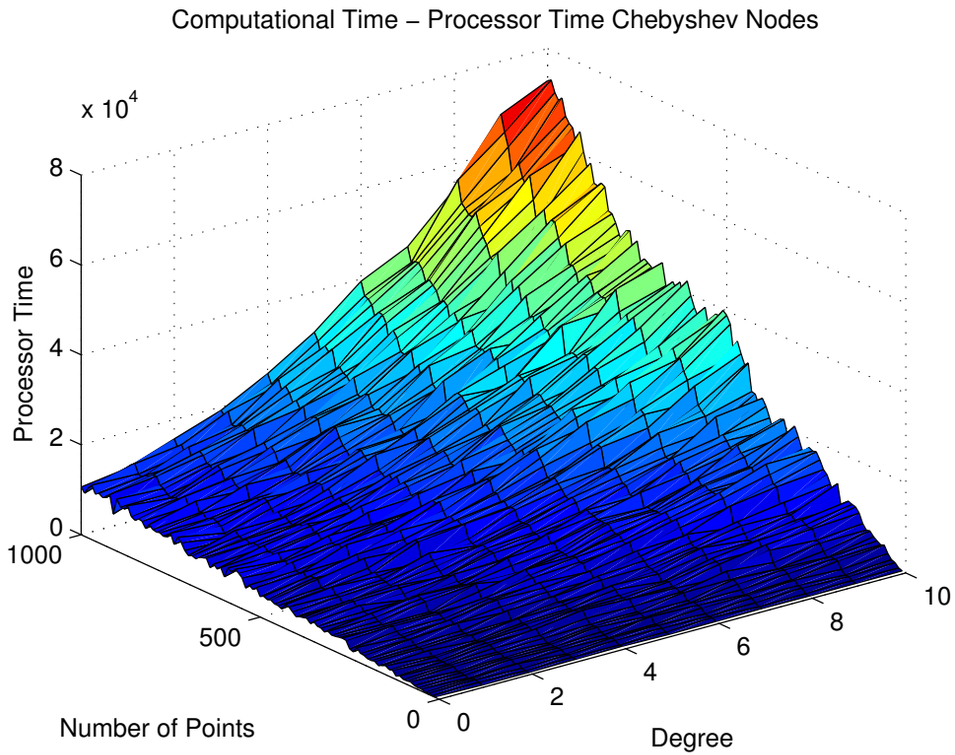
2) Run count_indiv.m from Matlab, arguments:

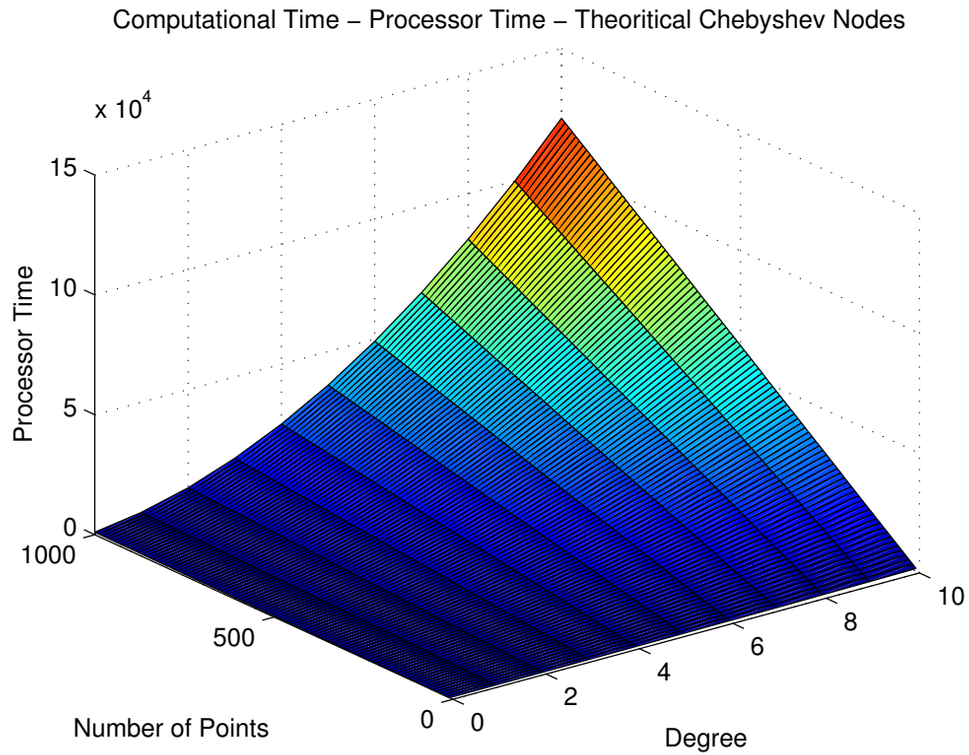
0 (argument here is for specifing the index of Admissibility Coefficient.)

R. Hiptmair

rev 30401, October 28, 2009

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 30401, October 28, 2009

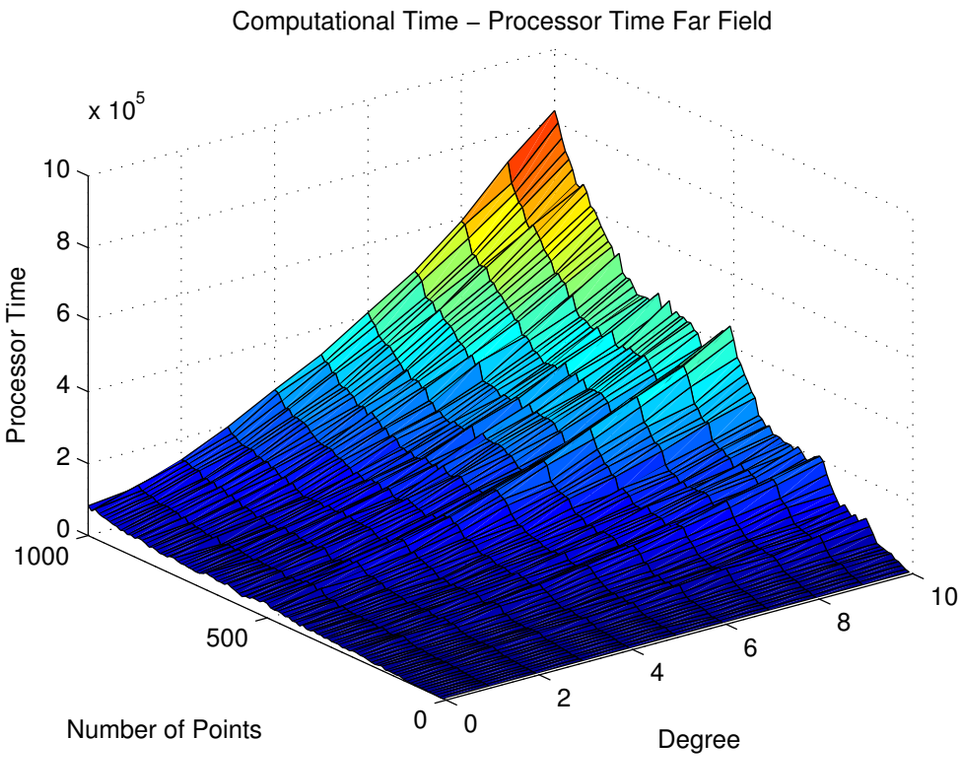
③: Far field analysis $\hat{=}$ Sum $\displaystyle\sum_{\substack{\sigma\in T_x\\ x_i\in\sigma}}\sum_{\substack{\mu\in T_y\\ (\sigma,\mu)\in P^{\rm far}}} \mathbf{V}_\sigma\mathbf{X}_{\sigma,\mu}\mathbf{w}_\mu$

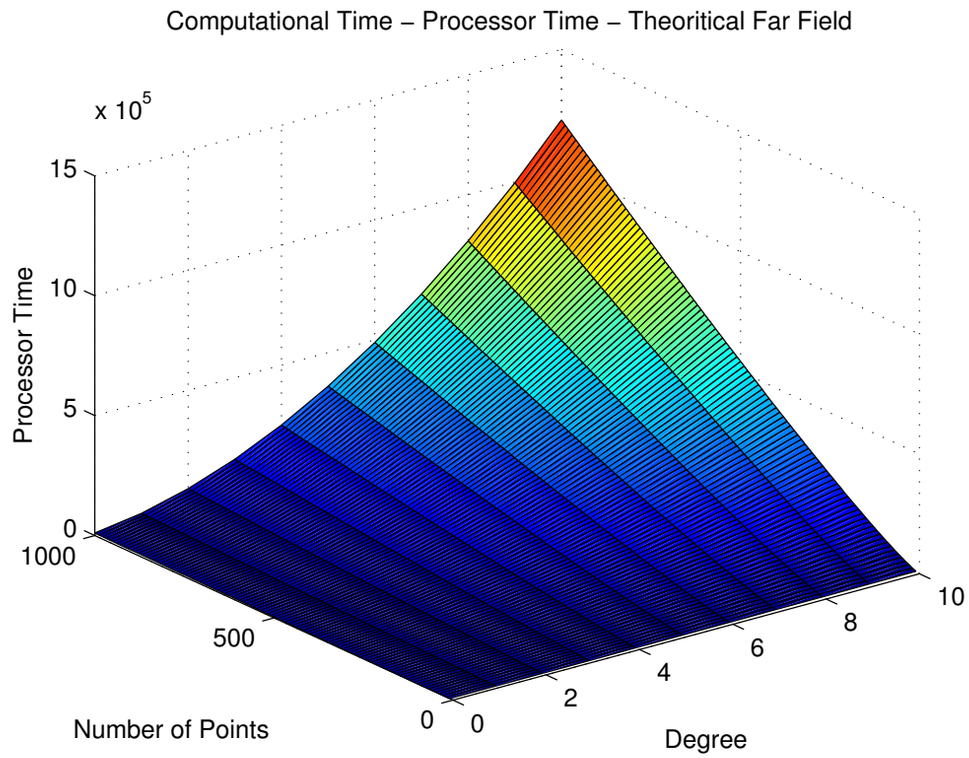
for each $\sigma\in T_x$ { Compute $\mathbf{V}_\sigma$; $\mathbf{s}:=0$

for each $\mu\in T_y$, $(\sigma,\mu)\in P^{\rm far}$ { $\mathbf{s}\leftarrow\mathbf{s}+\mathbf{X}_{\sigma,\mu}\mathbf{w}_\mu$ }

$\mathbf{f}_{|\sigma}\leftarrow\mathbf{f}_{|\sigma}+\mathbf{V}_\sigma\mathbf{s}$ }

Computational Effort $O(n\log_2 n(d+1)^2)$, Memory Required $O(n(d+1))$





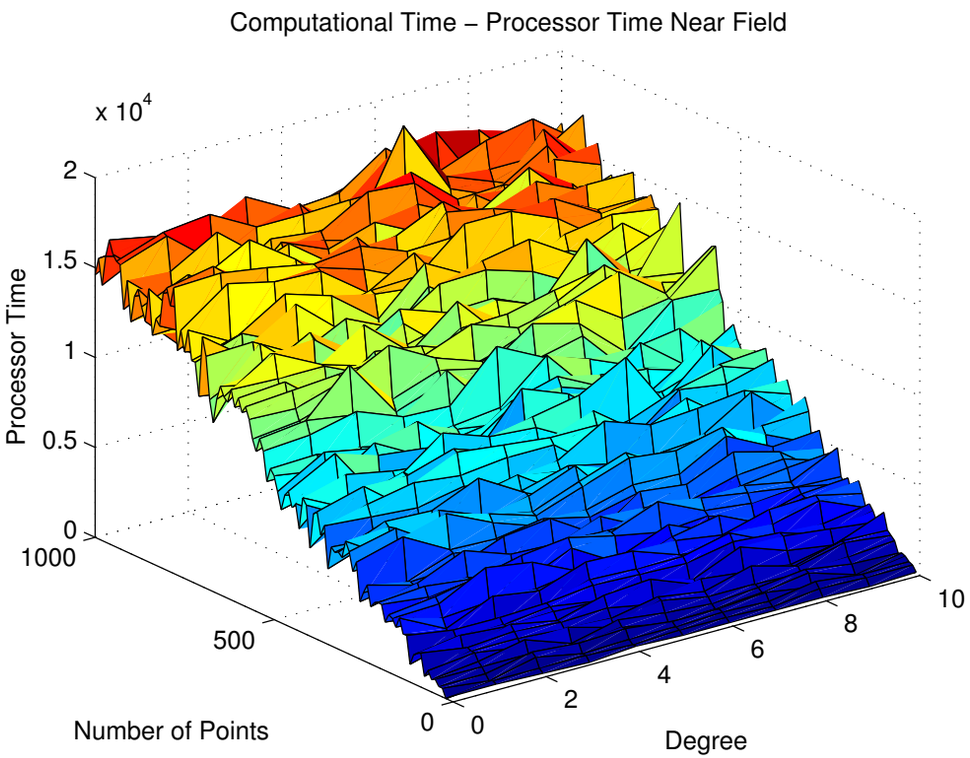
R. Hiptmair

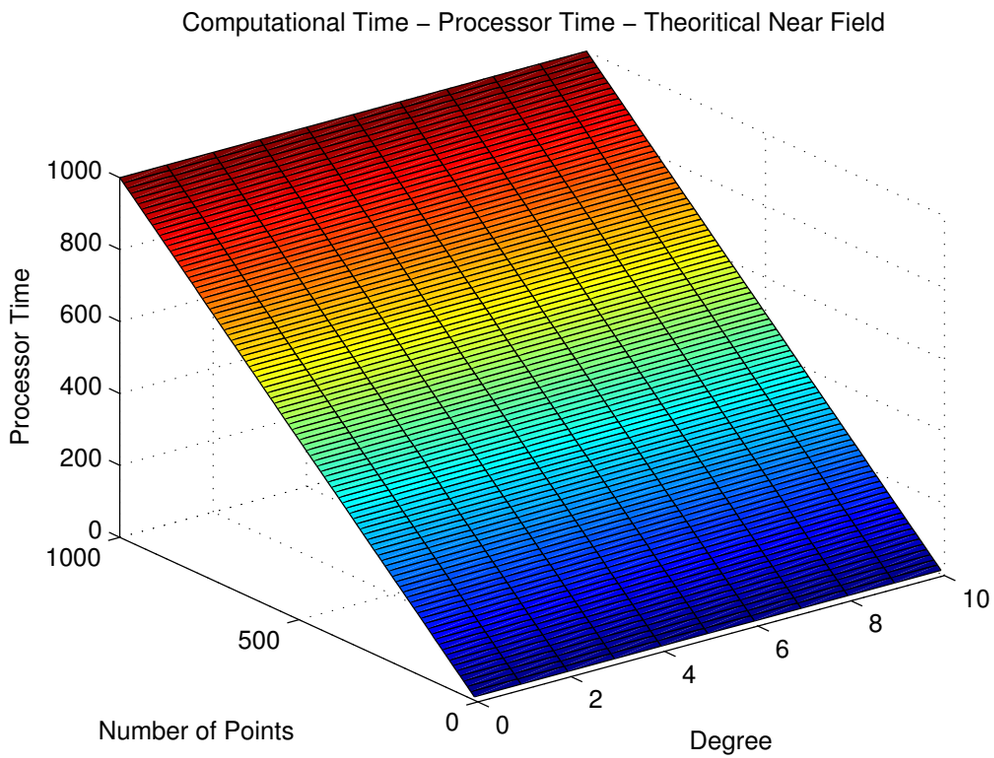
rev 30401, October 28, 2009

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 $O(n)$

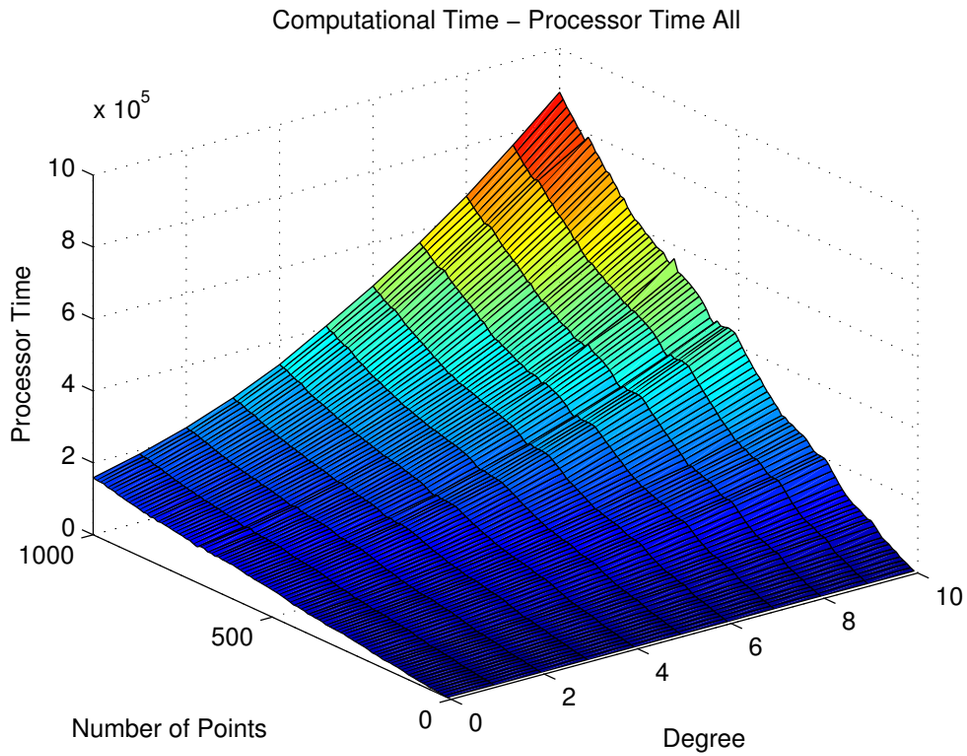


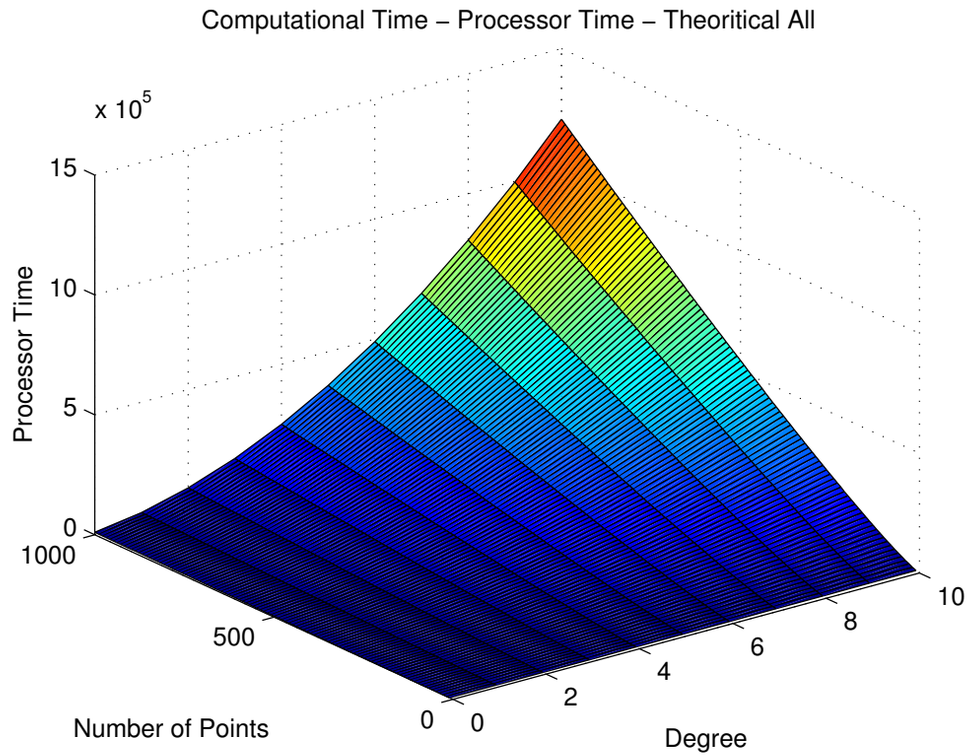


R. Hiptmair

rev 30401, October 28, 2009

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

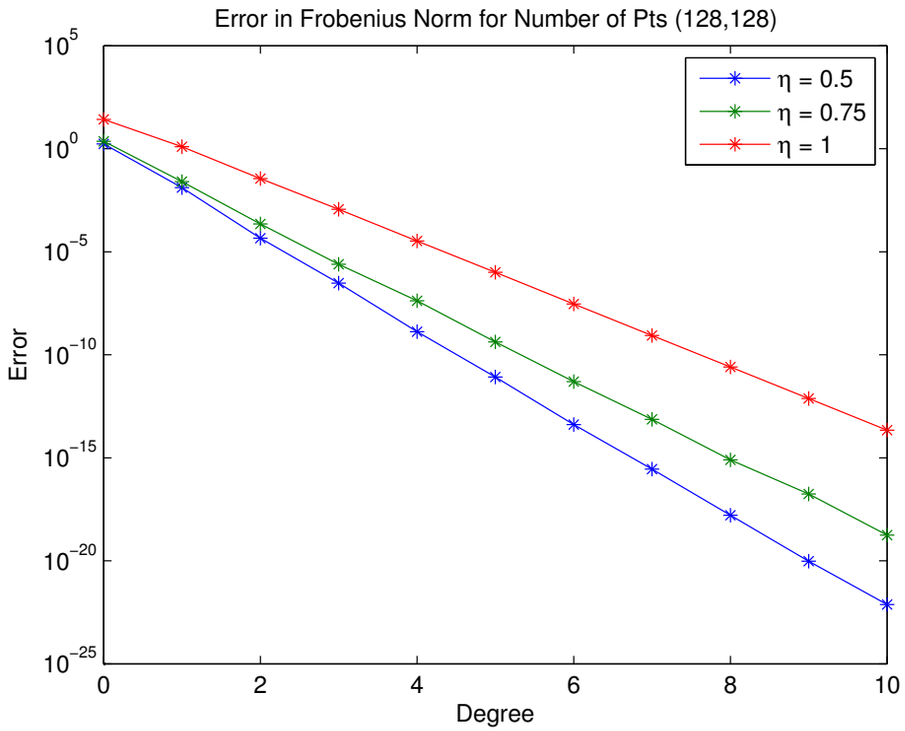
△

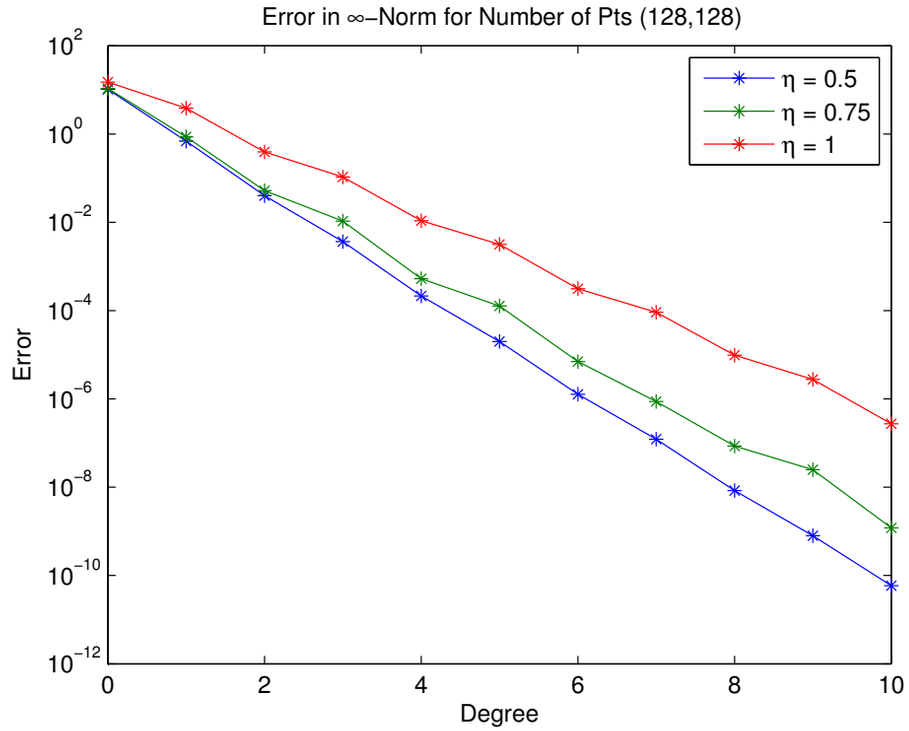
R. Hiptmair

rev 30401, October 28, 2009

Example 11.4.6 (Convergence of clustering approximation with collocation matrix).

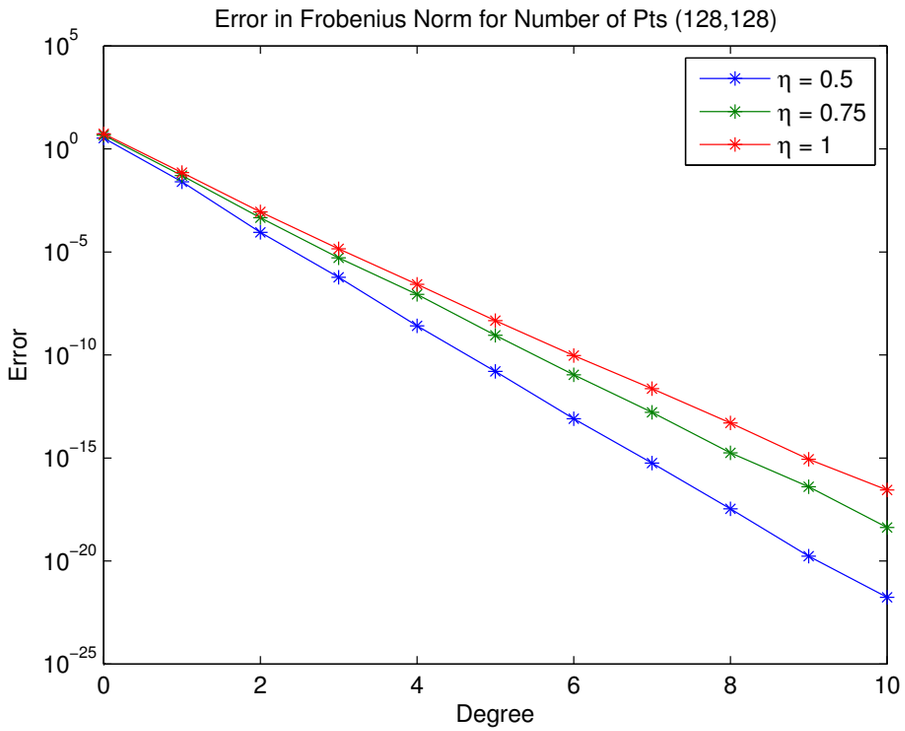
- `x = 0:1/128:1;`, $G(x,y) = |x-y|^{-1}$ for $x \neq y$, $G(x,x) = 0$.

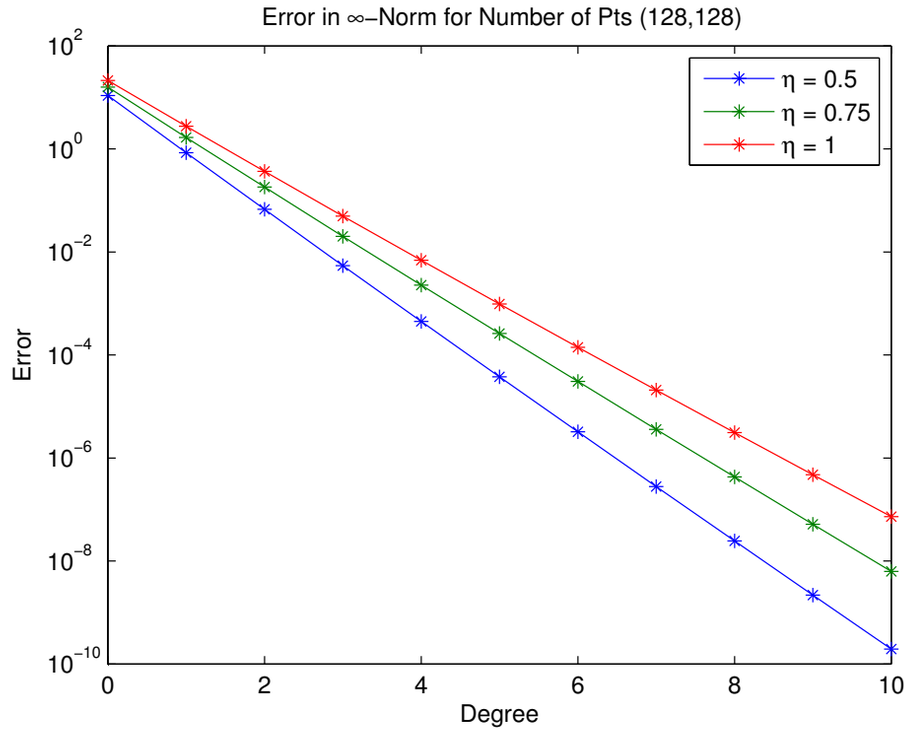




In file changeable.c
a) put the admissibility condition in admissible_ClusterPair
b) put kernel function in kernel_function
c) put NONE as return values in get_iOperation
1) Run main.cpp, input as follows
Choice - 2, NumberofPts - 128 128 1,
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1
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 $x \neq y$, $G(x,x) = 0$.





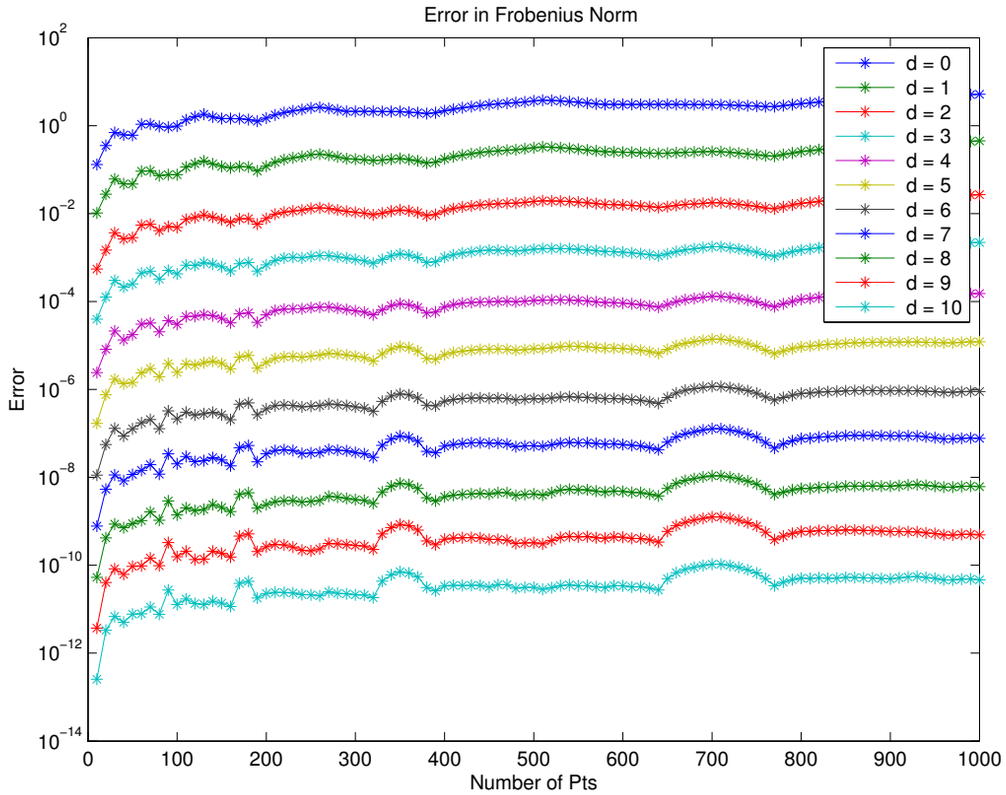
In file changeable.c
a) put the admissibility condition in admissible_ClusterPair
b) put kernel function in kernel_function
c) put SQRT as return values in get_iOperation
1) Run main.cpp, input as follows
Choice - 2, NumberofPts - 128 128 1,
Start of 1 - 0, End of 1 - 1, Start of 2 - 0, End of 2 - 1
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.)

◇

R. Hiptmair

rev 30401, October 28, 2009

- 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 30401, October 28, 2009

Example 11.4.7 (Analysis of trigonometric polynomials). [16]

Given: $\{t_0, \ldots, t_{n-1}\} \subset [0,1[$, $\alpha_{-m+1}, \ldots, \alpha_m \in \mathbb{C}$, $n = 2m$, $m \in \mathbb{N}$, compute

$$c_k := p(t_k)\,, \quad j = 0, \ldots, n-1 \quad \text{for} \quad \underbrace{p(t) := \sum_{j=-m+1}^{m} \alpha_j e^{-2\pi i j t}}_{\text{Trigonometric Polynomial}}\,.$$

Discrete Fourier transformation (DFT) $\rightarrow$ Section 8.2:

$$f_l := \sum_{j=-m+1}^{m} \alpha_j e^{-\frac{2\pi i}{n} j l} \quad \overset{\text{Lemma 8.2.10}}{\Leftrightarrow} \quad \alpha_j = \frac{1}{n} \sum_{l=-m+1}^{m} f_l e^{\frac{2\pi i}{n} l j}$$

FFT: f_l, $l = -m+1, \ldots, m$, calculated with computational effort $O(n \log_2 n)$

$$\begin{aligned} c_k &= \sum_{j=-m+1}^{m} \alpha_j e^{2\pi i j t_k} = \sum_{j=-m+1}^{m} \frac{1}{n} \Big(\sum_{l=-m+1}^{m} f_l e^{\frac{2\pi i}{n} l j} \Big) e^{-2\pi i j t_k} \\ &= \frac{1}{n} \sum_{l=-m+1}^{m} f_l \sum_{j=-m+1}^{m} e^{-2\pi i j (t_k - l/n)} \\ &= \frac{1}{n} \sum_{l=-m+1}^{m} f_l e^{-2\pi i (t_k - l/n)(-m+1)} \frac{1 - e^{-2\pi i n (t_k - l/n)}}{1 - e^{-2\pi i (t_k - l/n)}} \end{aligned}$$

R. Hiptmair

rev 30401, October 28, 2009

$$= \frac{1}{n} \sum_{l=-m+1}^{m} f_l e^{-\pi i t_k} \sin(\pi n t_k) \frac{1}{\sin(\pi(t_k - l/n))} (-1)^l e^{-\pi i l/n} \, .$$

$$\mathbf{c} = \operatorname{diag}\left(\frac{\sin(\pi n t_k)}{e^{\pi i t_k}} \right)_{k=-m+1}^{m} \mathbf{M} \operatorname{diag}\left((-1)^l e^{-\pi i l/n} \right)_{l=-m+1,\ldots,m} \mathbf{f} \, .$$

in accordance with collocation matrix (11.1.1)

$$\mathbf{M} := \left(\frac{1}{\sin(\pi(t_k - l/n))} \right)_{\substack{k=-m+1,\ldots,m \\ l=-m+1,\ldots,m}} \in \mathbb{R}^{2n,2n} \, .$$

Approximative analysis of Clustering algorithms **!** $\rightarrow$ USFFT

◇

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

△

R. Hiptmair

rev 30401, October 28, 2009

With Clusteringapproximation related, or generalizations thereof:

- $\mathcal{H}$-Matrix-Techniques [4]
- $\mathcal{H}^2$-Matrix-Techniques [26]
- Multipol-Methods [45, 22] $\rightarrow$ further "‘ Millennium algorithm” `http://orion.math.iastate.edu/burkardt/misc/algorithms_dongarra.htm`

12 Single Step Methods

12.1 Initial value problems (IVP) for ODEs

Acronym: **ODE** = **o**rdinary **d**ifferential **e**quation

Some grasp of the meaning and theory of ordinary differential equations (ODEs) is indispensable for understanding the construction and properties of numerical methods. Relevant information can be found in [52, Sect. 5.6, 5.7, 6.5].

12.1.1 Examples

Example 12.1.1 (Growth with limited resources). [1, Sect. 1.1], [29, Ch. 60]

$y : [0, T] \mapsto \mathbb{R}$: bacterial population density as a function of time

Model: autonomous logistic differential equations [52, Ex. 5.7.2]

$$\dot{y} = f(y) := (\alpha - \beta y)\, y \tag{12.1.2}$$

✎ Notation (Newton): dot $\dot{}$ $\hat{=}$ (total) derivative with respect to time t

- $y \hat{=}$ population density, $[y] = \frac{1}{\mathrm{m}^2}$
- growth rate $\alpha - \beta y$ with growth coefficients $\alpha, \beta > 0$, $[\alpha] = \frac{1}{\mathrm{s}}$, $[\beta] = \frac{\mathrm{m}^2}{\mathrm{s}}$: decreases due to more fierce competition as population density increases.

Note: we can only compute a solution of (12.1.2), when provided with an initial value $y(0)$.

R. Hiptmair

rev 30401, February 19, 2011

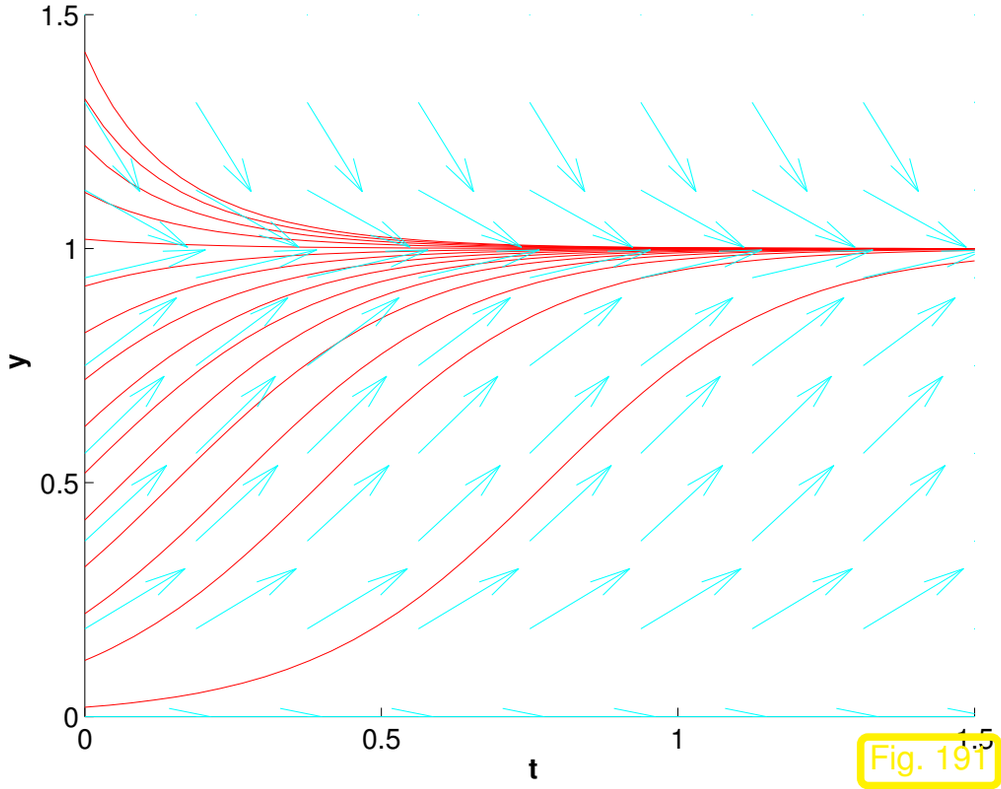


Fig. 191

Solution for different $y(0)$ ($\alpha, \beta = 5$)

By separation of variables

➥ solution of (12.1.2)

for $y(0) = y_0 > 0$

$$y(t) = \frac{\alpha y_0}{\beta y_0 + (\alpha - \beta y_0)\exp(-\alpha t)}\,, \qquad (12.1.3)$$

for all $t \in \mathbb{R}$

$f'(y^*) = 0$ for $y^* \in \{0, \alpha/\beta\}$, which are the stationary points for the ODE (12.1.2). If $y(0) = y^*$ the solution will be constant in time.

◇

Terminology: (12.1.2) = autonomous ODE

An ODE is called autonomous, if the right hand side does not depend on the independent variable (usually denoted by t), but only on the state (usually denoted by y/$\mathbf{y}$)

R. Hiptmair

rev 30401, February 19, 2011

Example 12.1.4 (Predator-prey model). [1, Sect. 1.1], [27, Sect. 1.1.1], [29, Ch. 60], [10, Ex. 11.3]

Predators and prey coexist in an ecosystem. Without predators the population of prey would be governed by a simple exponential growth law. However, the growth rate of prey will decrease with increasing numbers of predators and, eventually, become negative. Similar considerations apply to the predator population and lead to an ODE model.

Model: autonomous Lotka-Volterra ODE:

$$\begin{aligned} \dot{u} &= (\alpha - \beta v)u \\ \dot{v} &= (\delta u - \gamma)v \end{aligned} \quad \leftrightarrow \quad \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \text{with} \quad \mathbf{y} = \begin{pmatrix} u \\ v \end{pmatrix}, \quad \mathbf{f}(\mathbf{y}) = \begin{pmatrix} (\alpha - \beta v)u \\ (\delta u - \gamma)v \end{pmatrix}. \tag{12.1.5}$$

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 ▷

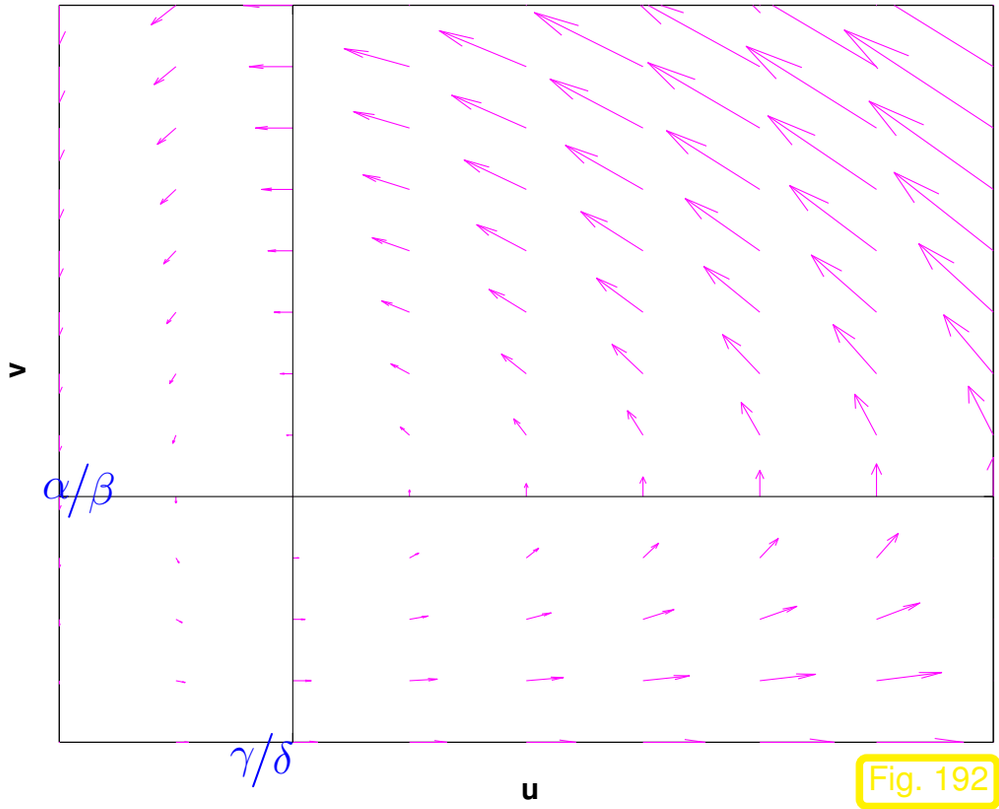


Fig. 192

Solution curves are trajectories of particles carried along by velocity field $\mathbf{f}$.

Parameter values for Fig. 192: $\alpha = 2, \beta = 1, \delta = 1, \gamma = 1$

R. Hiptmair

rev 30401, February 19, 2011

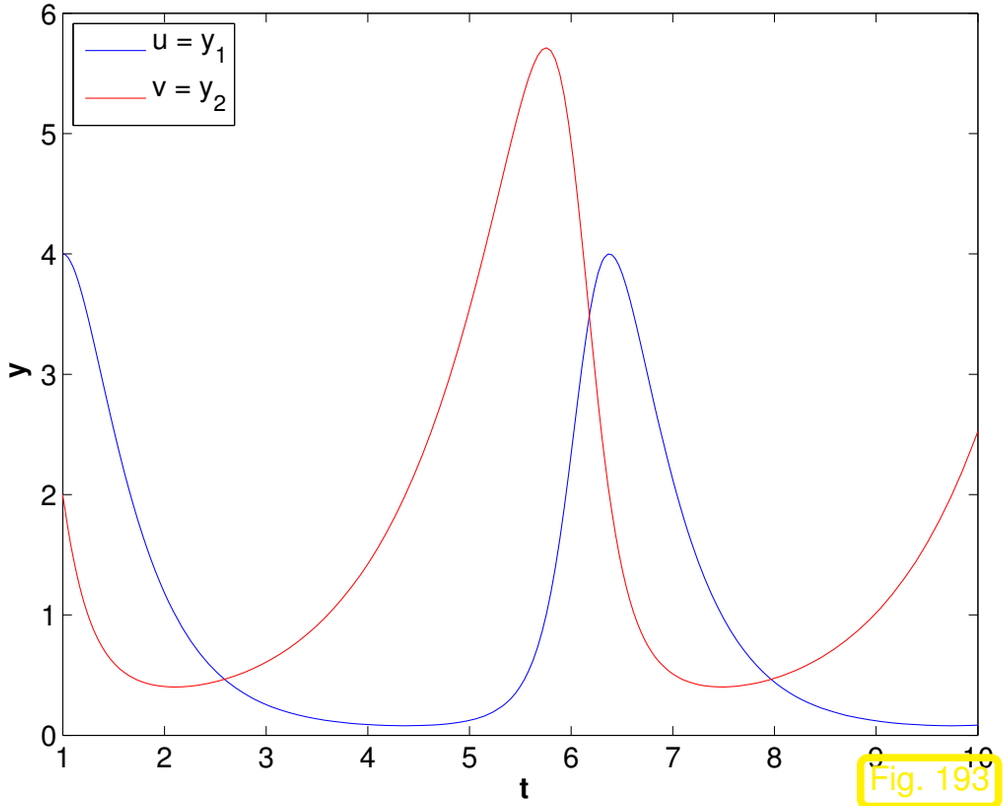


Fig. 193

Solution $\begin{pmatrix} u(t) \\ v(t) \end{pmatrix}$ for $\mathbf{y}_0 := \begin{pmatrix} u(0) \\ v(0) \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$

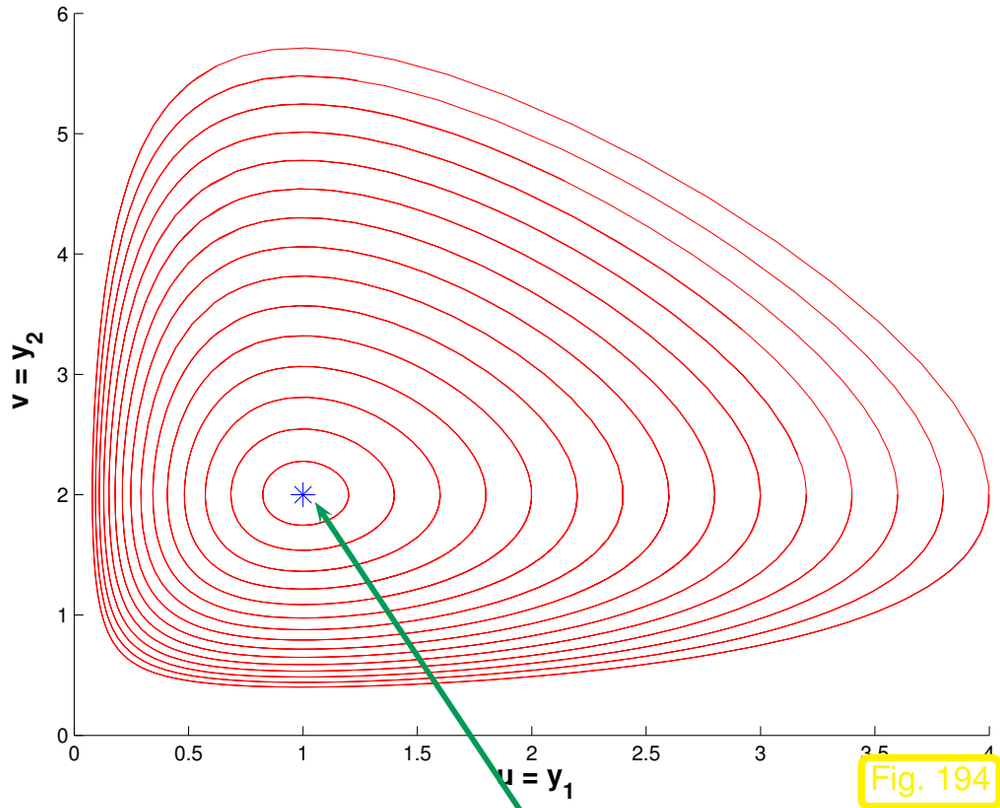


Fig. 194

Solution curves for (12.1.5)

Parameter values for Figs. 194, 193: $\alpha = 1, \beta = 1, \delta = 1, \gamma = 2$

◇

Example 12.1.6 (Heartbeat model). → [11, p. 655]

R. Hiptmair

rev 30401, February 19, 2011

State of heart described by quantities:

$$\begin{aligned} l = l(t) &\;\hat{=}\; \text{length of muscle fiber} \\ p = p(t) &\;\hat{=}\; \text{electro-chemical potential} \end{aligned}$$

Phenomenological model:

$$\begin{aligned} \dot{l} &= -(l^3 - \alpha l + p)\,, \\ \dot{p} &= \beta l\,, \end{aligned} \tag{12.1.7}$$

with parameters:

$$\begin{aligned} \alpha &\;\hat{=}\; \text{pre-tension of muscle fiber} \\ \beta &\;\hat{=}\; \text{(phenomenological) feedback parameter} \end{aligned}$$

This is the so-called Zeeman model: it is a phenomenological model entirely based on macroscopic observations without relying on knowledge about the underlying molecular mechanisms.

Vector fields and solutions for different choices of parameters:

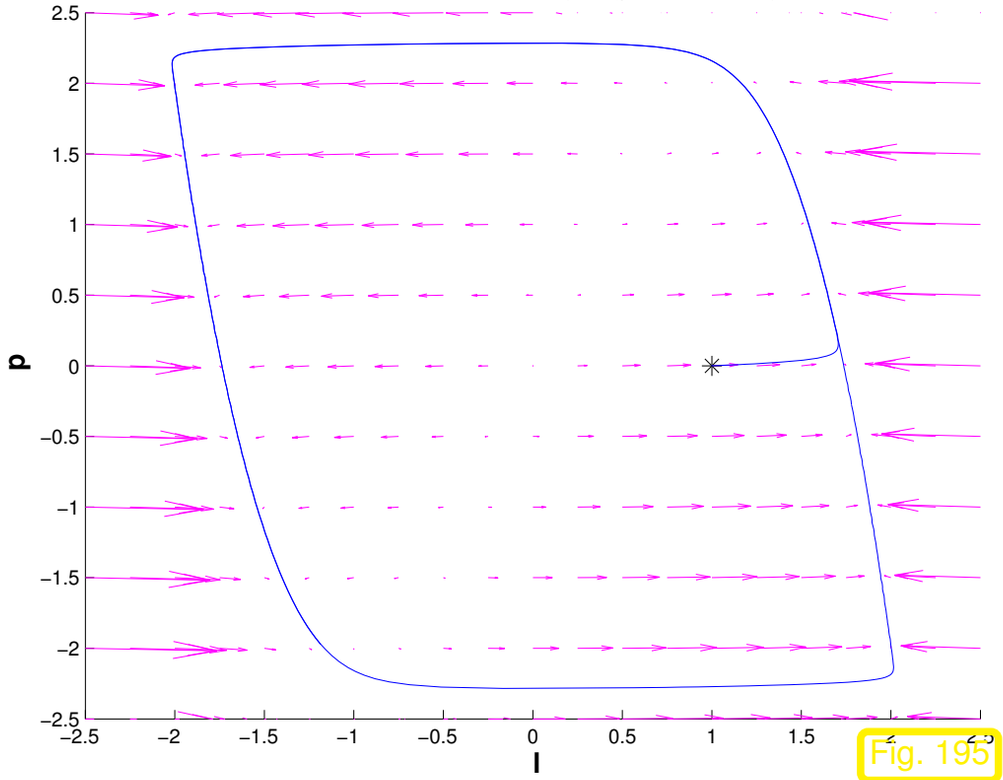


Fig. 195

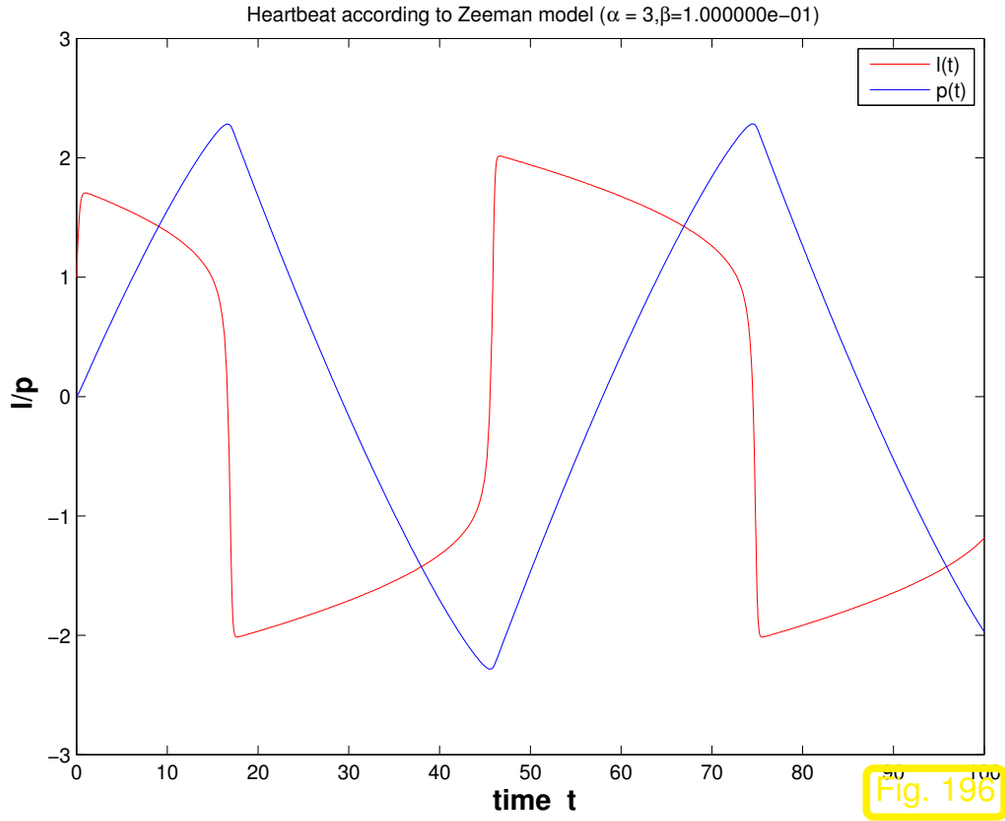


Fig. 196

R. Hiptmair

rev 30401, February 19, 2011

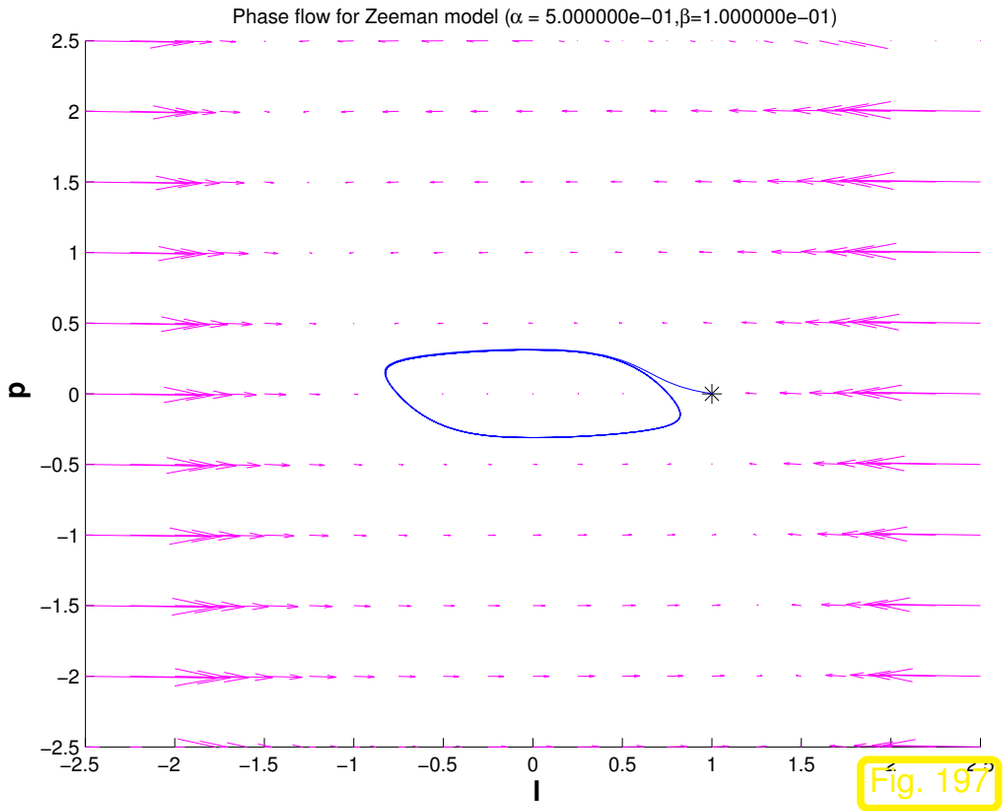


Fig. 197

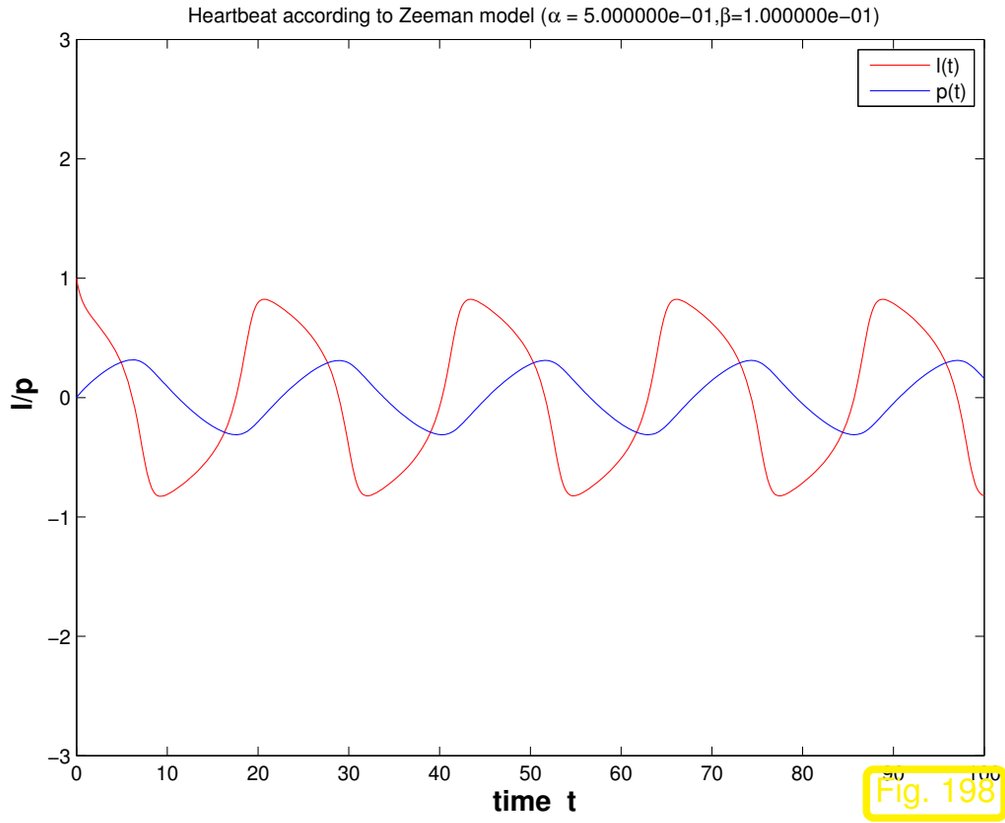


Fig. 198

Observation: $\alpha \ll 1$ ➤ atrial fibrillation

◇

Example 12.1.8 (Transient circuit simulation). [29, Ch. 64]

Transient nodal analysis, *cf.* Ex. 2.0.1:

Kirchhoff current law

$$i_R(t) - i_L(t) - i_C(t) = 0 \,. \qquad (12.1.9)$$

Transient constitutive relations:

$$i_R(t) = R^{-1} u_R(t) \,, \qquad (12.1.10)$$

$$i_C(t) = C \frac{du_C}{dt}(t) \,, \qquad (12.1.11)$$

$$u_L(t) = L \frac{di_L}{dt}(t) \,. \qquad (12.1.12)$$

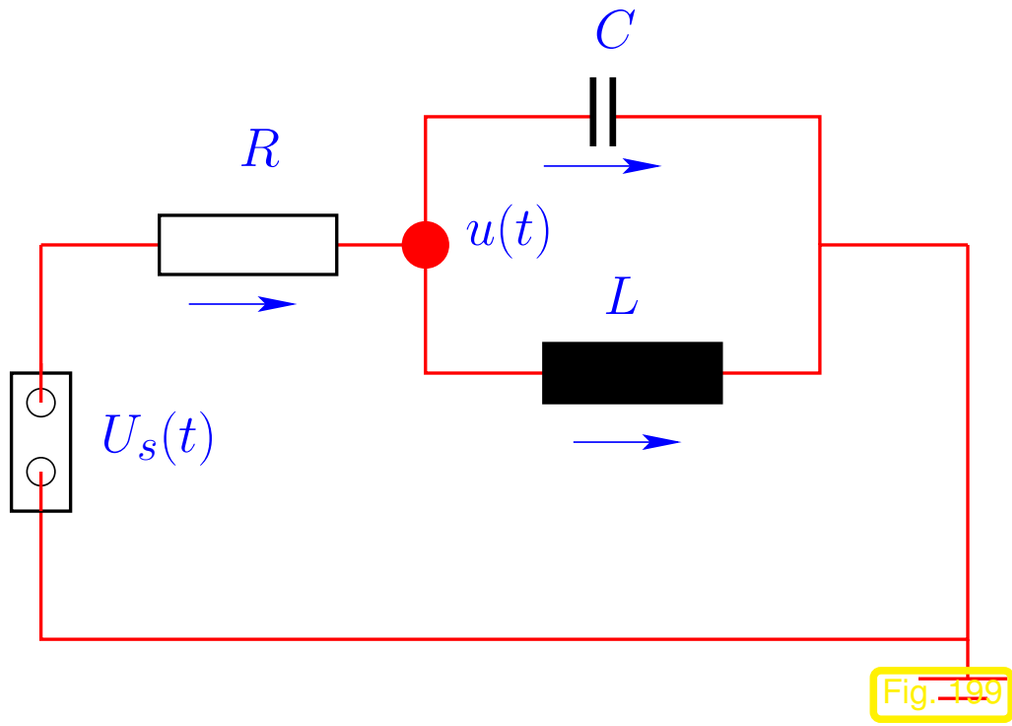


Fig. 199

Given: source voltage $U_s(t)$

Differentiate (12.1.9) w.r.t. t and plug in constitutive relations for circuit elements:

$$R^{-1} \frac{du_R}{dt}(t) - L^{-1} u_L(t) - C \frac{d^2 u_C}{dt^2}(t) = 0 \,.$$

We follow the policy of nodal analysis and express all voltages by potential differences between nodes of the circuit. For this simple circuit there is only one node with unknown potential, see Fig. 199. Its time-dependent potential will be denoted by $u(t)$.

$$R^{-1}(\dot{U}_s(t) - \dot{u}(t)) - L^{-1} u(t) - C \frac{d^2 u}{dt^2}(t) = 0 \,.$$

R. Hiptmair

rev 30401, February 19, 2011

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

Abstract mathematical description

Initial value problem (IVP) for first-order ordinary differential equation (ODE):
(→ [52, Sect. 5.6], [10, 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

For $d > 1$ $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ can be viewed as a system of ordinary differential equations:

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad \Longleftrightarrow \quad \begin{pmatrix} y_1 \\ \vdots \\ y_d \end{pmatrix} = \begin{pmatrix} f_1(t, y_1, \ldots, y_d) \\ \vdots \\ f_d(t, y_1, \ldots, y_d) \end{pmatrix} \, .$$

R. Hiptmair

rev 30401, February 19, 2011

Recall: autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ ($\mathbf{f}$ does not depend on time t)

For autonomous ODEs:

- $I = \mathbb{R}$ and r.h.s. $\mathbf{y} \mapsto \mathbf{f}(\mathbf{y})$ can be regarded as stationary vector field (velocity field)
- if $t \mapsto \mathbf{y}(t)$ is solution $\Rightarrow$ for any $\tau \in \mathbb{R}$ $t \mapsto \mathbf{y}(t+\tau)$ is solution, too.
- initial time irrelevant: canonical choice $t_0 = 0$

Note: autonomous ODEs naturally arise when modelling time-invariant systems/phenomena. All examples from Sect. 12.1.1 above led to autonomous ODEs.

R. Hiptmair

rev 30401, February 19, 2011

Remark 12.1.14 (Conversion into autonomous ODE).

Idea: include time as an extra $d+1$-st component of an extended state vector.

This solution component has to grow linearly $\Leftrightarrow$ temporal derivative $= 1$

$$\mathbf{z}(t) := \begin{pmatrix} \mathbf{y}(t) \\ t \end{pmatrix} = \begin{pmatrix} \mathbf{z}' \\ z_{d+1} \end{pmatrix} : \quad \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad \leftrightarrow \quad \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}) \,, \quad \mathbf{g}(\mathbf{z}) := \begin{pmatrix} \mathbf{f}(z_{d+1}, \mathbf{z}') \\ 1 \end{pmatrix} \,.$$

➢ focus on autonomous ODEs in the remainder of this chapter

△

Remark 12.1.15 (From higher order ODEs to first order systems). [10, Sect. 11.2]]

Ordinary differential equation of order $n \in \mathbb{N}$:

$$\mathbf{y}^{(n)} = \mathbf{f}(t, \mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(n-1)}) \quad . \tag{12.1.16}$$

✎ Notation: superscript $^{(n)} \hat{=}$ n-th temporal derivative t

➤ Conversion into 1st-order ODE (system of size nd)

$$\mathbf{z}(t) := \begin{pmatrix} \mathbf{y}(t) \\ \mathbf{y}^{(1)}(t) \\ \vdots \\ \mathbf{y}^{(n-1)}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_n \end{pmatrix} \in \mathbb{R}^{dn}: \quad (12.1.16) \quad \leftrightarrow \quad \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}) \;, \quad \mathbf{g}(\mathbf{z}) := \begin{pmatrix} \mathbf{z}_2 \\ \mathbf{z}_3 \\ \vdots \\ \mathbf{z}_n \\ \mathbf{f}(t, \mathbf{z}_1, \ldots, \mathbf{z}_n) \end{pmatrix} . \tag{12.1.17}$$

R. Hiptmair

rev 30401, February 19, 2011

Note: n initial values $\mathbf{y}(t_0), \dot{\mathbf{y}}(t_0), \ldots, \mathbf{y}^{(n-1)}(t_0)$ required!

△

Basic assumption: right hand side $\mathbf{f} : I \times D \mapsto \mathbb{R}^d$ locally Lipschitz continuous in $\mathbf{y}$

Definition 12.1.19 (Lipschitz continuous function)**.** *(→ [52, Def. 4.1.4])*

$\mathbf{f} : \Omega \mapsto \mathbb{R}^d$ *is Lipschitz continuous (in the second argument), if*

$$\exists L > 0: \quad \|\mathbf{f}(t, \mathbf{w}) - \mathbf{f}(t, \mathbf{z})\| \leq L \|\mathbf{w} - \mathbf{z}\| \quad \forall (t, \mathbf{w}), (y, \mathbf{z}) \in \Omega \, .$$

Definition 12.1.21 (Local Lipschitz continuity). *(→ [52, Def. 4.1.5])*

$\mathbf{f} : \Omega \mapsto \mathbb{R}^d$ *is locally Lipschitz continuous, if*

$$\forall (t, \mathbf{y}) \in \Omega: \quad \exists \delta > 0,\ L > 0:$$
$$\|\mathbf{f}(\tau, \mathbf{z}) - \mathbf{f}(\tau, \mathbf{w})\| \leq L \|\mathbf{z} - \mathbf{w}\|$$
$$\forall \mathbf{z}, \mathbf{w} \in D:\ \|\mathbf{z} - \mathbf{y}\| < \delta,\ \|\mathbf{w} - \mathbf{y}\| < \delta, \forall \tau \in I: |t - \tau| < \delta\ .$$

✎ Notation: $D_{\mathbf{y}}\mathbf{f} \hat{=}$ derivative of $\mathbf{f}$ w.r.t. state variable (= Jacobian $\in \mathbb{R}^{d,d}$ **!**)

R. Hiptmair

rev 30401, February 19, 2011

A simple criterion for local Lipschitz continuity:

Lemma 12.1.22 (Criterion for local Liptschitz continuity).
If $\mathbf{f}$ and $D_{\mathbf{y}}\mathbf{f}$ are continuous on the extended state space Ω, then $\mathbf{f}$ is locally Lipschitz continuous(→ Def. 12.1.21).

Theorem 12.1.23 (Theorem of Peano & Picard-Lindelöf). *[1, Satz II(7.6)], [52, Satz 6.5.1], [10, Thm. 11.10], [29, Thm. 73.1]*

If $\mathbf{f} : \hat{\Omega} \mapsto \mathbb{R}^d$ *is locally Lipschitz continuous (→ Def. 12.1.21) then for all initial conditions* $(t_0, \mathbf{y}_0) \in \hat{\Omega}$ *the IVP* (12.1.13) *has a solution* $\mathbf{y} \in C^1(J(t_0, \mathbf{y}_0), \mathbb{R}^d)$ *with maximal (temporal) domain of definition* $J(t_0, \mathbf{y}_0) \subset \mathbb{R}$.

Remark 12.1.24 (Domain of definition of solutions of IVPs).

Solutions of an IVP have an intrinsic maximal domain of definition

! domain of definition/domain of existence $J(t_0, \mathbf{y}_0)$ usually depends on $(t_0, \mathbf{y}_0)$!

Terminology: if $J(t_0, \mathbf{y}_0) = I$ ➥ solution $\mathbf{y} : I \mapsto \mathbb{R}^d$ is global.

△

Notation: for autonomous ODE we always have $t_0 = 0$, therefore write $J(\mathbf{y}_0) := J(0, \mathbf{y}_0)$.

In light of Rem. 12.1.14 and Thm. 12.1.23: we consider only

$$\text{autonomous IVP:} \quad \boxed{\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad , \quad \mathbf{y}(0) = \mathbf{y}_0} \ , \tag{12.1.25}$$

with locally Lipschitz continuous ($\rightarrow$ Def. 12.1.21) right hand side $\mathbf{f}$.

Assumption 12.1.26 (Global solutions)**.**

All solutions of (12.1.25) *are global:* $J(\mathbf{y}_0) = \mathbb{R}$ *for all* $\mathbf{y}_0 \in D$.

Change of perspective: fix "time of interest" $t \in \mathbb{R} \setminus \{0\}$

$$\succ \quad \text{mapping} \quad \boldsymbol{\Phi}^t : \begin{cases} D & \mapsto & D \\ \mathbf{y}_0 & \mapsto & \mathbf{y}(t) \end{cases} \quad , \quad t \mapsto \mathbf{y}(t) \text{ solution of IVP (12.1.25)} \ ,$$

is well-defined mapping of the state space into itself, by Thm. 12.1.23 and Ass. 12.1.26

Now, we may also let t vary, which spawns a *family* of mappings $\left\{\boldsymbol{\Phi}^t\right\}$ of the state space into itself. However, it can also be viewed as a mapping with two arguments, a time t and an initial state value $\mathbf{y}_0$!

R. Hiptmair

rev 30401, February 19, 2011

Definition 12.1.27 (Evolution operator)**.**

Under Assumption 12.1.26 the mapping

$$\boldsymbol{\Phi} : \begin{cases} \mathbb{R} \times D \mapsto D \\ (t, \mathbf{y}_0) \mapsto \boldsymbol{\Phi}^t \mathbf{y}_0 := \mathbf{y}(t) \end{cases},$$

where $t \mapsto \mathbf{y}(t) \in C^1(\mathbb{R}, \mathbb{R}^d)$ *is the unique (global) solution of the IVP* $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$, *is the evolution operator for the autonomous ODE* $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

Note: $t \mapsto \boldsymbol{\Phi}^t \mathbf{y}_0$ describes the solution of $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ for $\mathbf{y}(0) = \mathbf{y}_0$ (a trajectory)

Remark 12.1.28 (Group property of autonomous evolutions)*.*

Under Assumption 12.1.26 the evolution operator gives rise to a group of mappings $D \mapsto D$:

$$\boldsymbol{\Phi}^s \circ \boldsymbol{\Phi}^t = \boldsymbol{\Phi}^{s+t} \quad , \quad \boldsymbol{\Phi}^{-t} \circ \boldsymbol{\Phi}^t = Id \quad \forall t \in \mathbb{R} \, . \tag{12.1.29}$$

This is a consequence of the uniqueness theorem Thm. 12.1.23. It is also intuitive: following an evolution up to time t and then for some more time s leads us to the same final state as observing it for the whole time $s + t$.

△

12.2 Euler methods

Targeted: initial value problem (12.1.13)

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \, . \tag{12.1.13}$$

Sought: *approximate* solution of (12.1.13) on $[t_0, T]$ up to final time $T \neq t_0$

However, the solution of an initial value problem is a *function* $J(t_0, \mathbf{y}_0) \mapsto \mathbb{R}^d$ and requires a suitable approximate representation. We postpone this issue here and first study a geometric approach to numerical integration.

numerical integration = approximate solution of initial value problems for ODEs

(Please distinguish from “numerical quadrature”, see Ch. 10.)

12.2.1 Explicit Euler method

Example 12.2.1 (Tangent field and solution curves).

Riccati differential equation

scalar ODE

$$\dot{y} = y^2 + t^2 \quad \blacktriangleright \quad d = 1, \quad I, D = \mathbb{R}^+ \, . \tag{12.2.2}$$

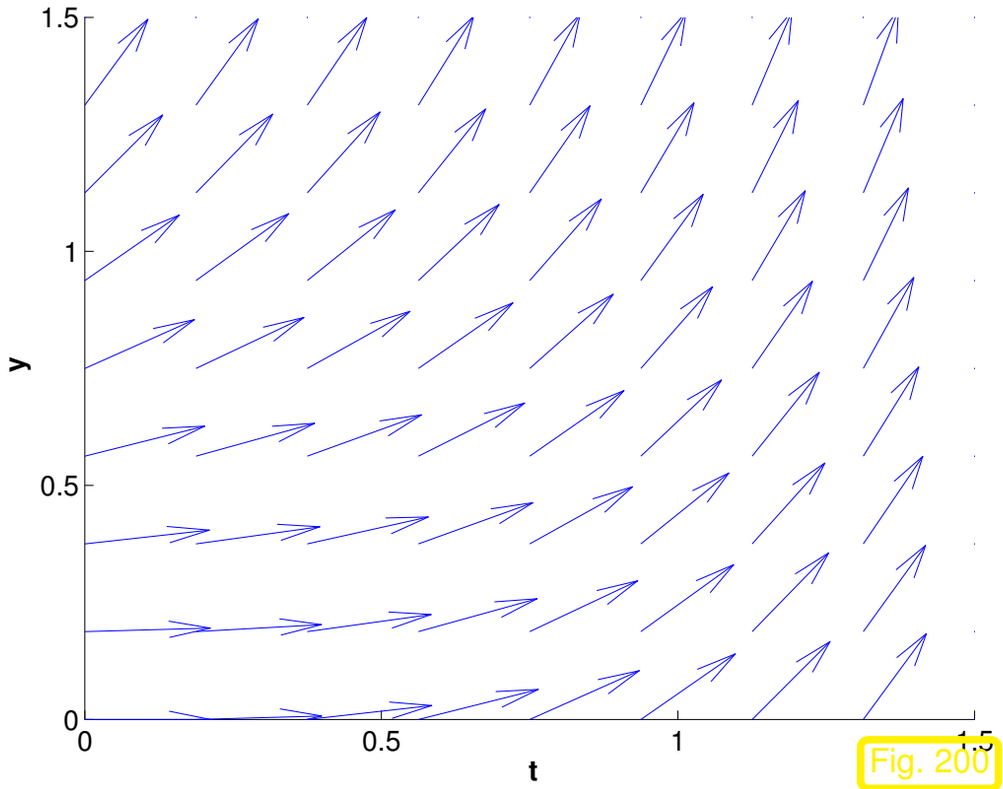


Fig. 200

tangent field

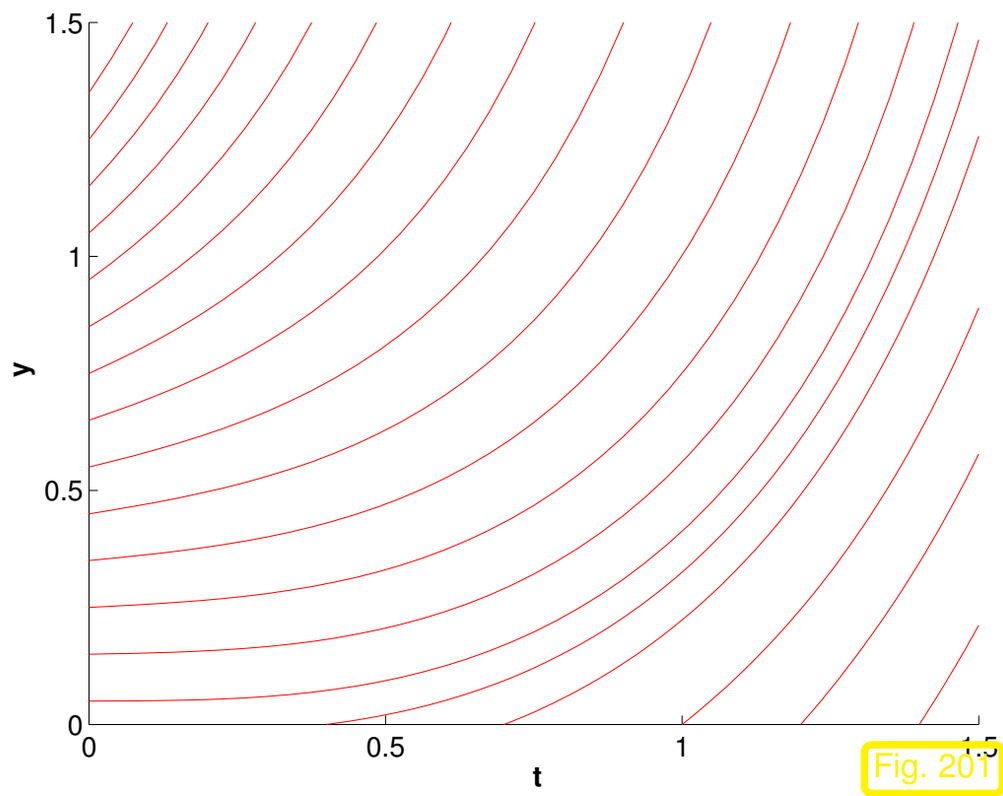
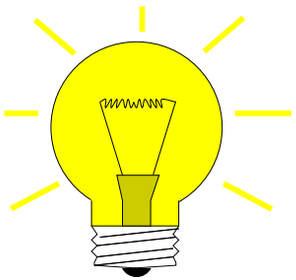


Fig. 201

solution curves

solution curves run tangentially to the tangent field in each point of the extended state space.

◇



Idea: ❶ timestepping: successive approximation of evolution on *small* intervals $[t_{k-1}, t_k]$, $k = 1, \ldots, N$, $t_N := T$,

❷ approximation of solution on $[t_{k-1}, t_k]$ by tangent curve to current initial condition.

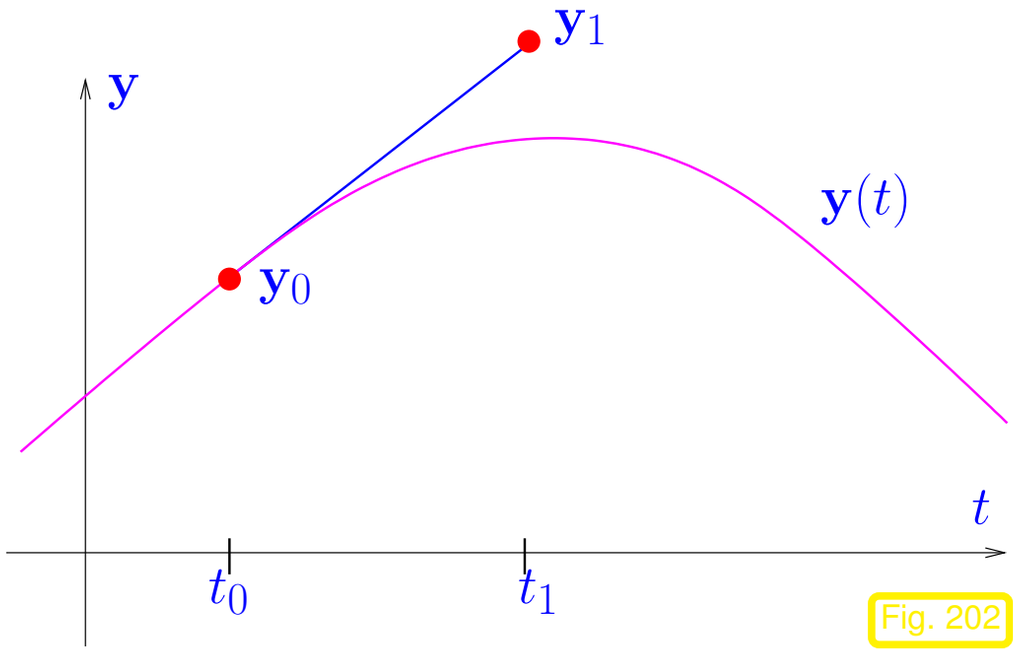


Fig. 202

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!

see also [29, Ch. 74]

R. Hiptmair

rev 30401, February 19, 2011

Example 12.2.3 (Visualization of explicit Euler method).

IVP for Riccati differential equation, see Ex. 12.2.1

$$\dot{y} = y^2 + t^2 \ . \qquad (12.2.2)$$

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

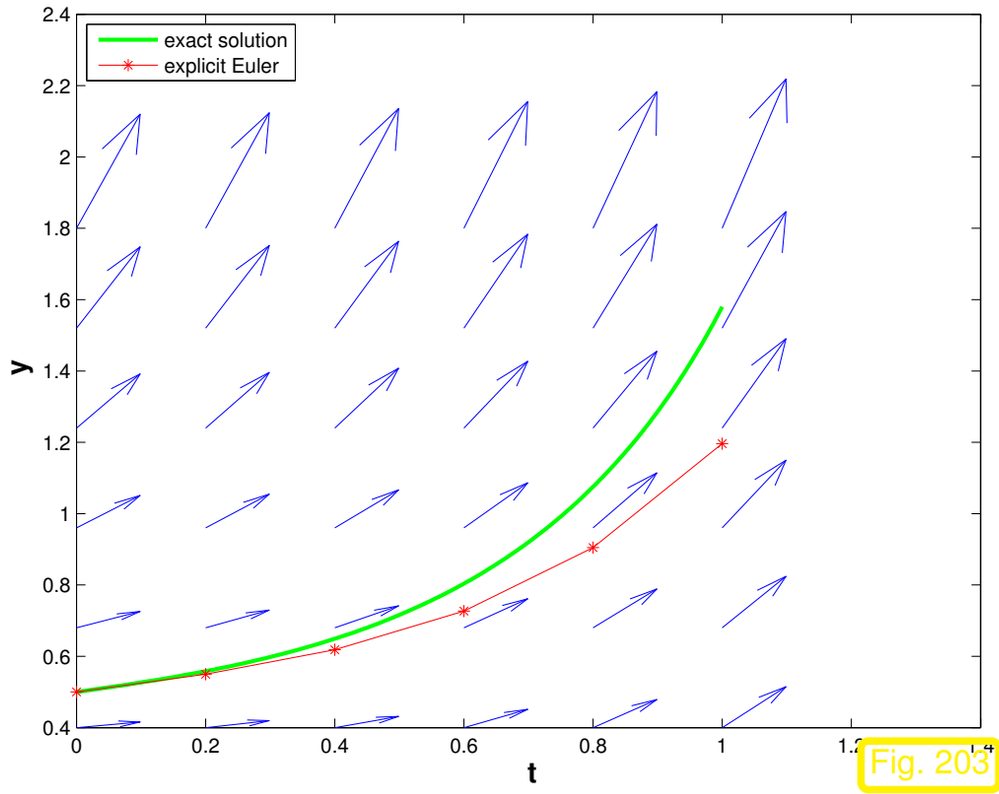


Fig. 203

◇

Formula: explicit Euler method generates a *sequence* $(\mathbf{y}_k)_{k=0}^N$ by the recursion

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_k, \mathbf{y}_k) \ , \quad k = 0, \ldots, N-1 \quad ,} \qquad (12.2.4)$$

with local (size of) timestep (stepsize) $h_k := t_{k+1} - t_k$.

Remark 12.2.5 (Explicit Euler method as difference scheme).

(12.2.4) by approximating derivative $\frac{d}{dt}$ by forward difference quotient on a (temporal) mesh $\mathcal{M} := \{t_0, t_1, \ldots, t_N\}$:

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \longleftrightarrow \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f(t_k, \mathbf{y}_h(t_k)) \;, \quad k = 0, \ldots, N-1 \;. \tag{12.2.7}$$

Difference schemes follow a simple policy for the *discretization* of differential equations: replace all derivatives by difference quotients connecting solution values on a set of discrete points (the mesh).

△

12.2.2 Implicit Euler method

R. Hiptmair

rev 30401, February 19, 2011

Why forward difference quotient and not backward difference quotient? Let's try!

On (temporal) mesh $\mathcal{M} := \{t_0, t_1, \ldots, t_N\}$ we obtain

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \longleftrightarrow \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f(t_{k+1}, \mathbf{y}_h(t_{k+1})) \,, \quad k = 0, \ldots, N-1 \,. \tag{12.2.9}$$

backward difference quotient

This leads to another simple timestepping scheme analoguous to (12.2.4):

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) \,, \quad k = 0, \ldots, N-1} \,, \tag{12.2.10}$$

with local timestep (stepsize) $h_k := t_{k+1} - t_k$.

(12.2.10) = implicit Euler method

R. Hiptmair

rev 30401, February 19, 2011

Note: (12.2.10) requires solving of a (possibly non-linear) system of equations to obtain $\mathbf{y}_{k+1}$ **!**
(➤ Terminology "implicit")

Remark 12.2.12 (Feasibility of implicit Euler timestepping).

Issue: Is (12.2.10) well defined, that is, can we solve it for $\mathbf{y}_{k+1}$ and is this solution unique?

Intuition: for small timesteps $h > 0$ the right hand side of (12.2.10) is a "small perturbation of the identity".

Formal: consider autonomous ODE and assume continuously differentiable right hand side: $\mathbf{f} \in C^1(D, \mathbb{R}^d)$.

(12.2.10) $\leftrightarrow$ h-dependent non-linear system of equations:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) \quad \Leftrightarrow \quad G(h, \mathbf{y}_{k+1}) = 0 \quad \text{with} \quad G(h, \mathbf{z}) := \mathbf{z} - h\mathbf{f}(\mathbf{z}) - \mathbf{y}_k \ .$$

Partial derivative:

$$\frac{dG}{d\mathbf{z}}(0,\mathbf{z}) = \mathbf{I}$$

Implicit function theorem [52, Thm. 7.8.1]: for *sufficiently small* $|h|$ the equation $G(h,\mathbf{z}) = 0$ defines a continuous function $\mathbf{z} = \mathbf{z}(h)$.

➢ for *sufficiently small* $h > 0$ the equation (12.2.10) can be uniquely solved for $\mathbf{y}_{k+1}$.

How to interpret the sequence $(\mathbf{y}_k)_{k=0}^N$ from (12.2.4)?

By "geometric insight" we expect:

$$\boxed{\mathbf{y}_k \approx \mathbf{y}(t_k)}$$

(Throughout, we use the notation $\mathbf{y}(t)$ for the exact solution of an IVP.)

If we are merely interested in the final state $\mathbf{y}(T)$, then the explicit Euler method will give us the answer $\mathbf{y}_N$.

If we are interested in an approximate solution $\mathbf{y}_h(t) \approx \mathbf{y}(t)$ as a function $[t_0, T] \mapsto \mathbb{R}^d$, we have to do

post-processing = reconstruction of a function from $\mathbf{y}_k$, $k = 0, \ldots, N$

Technique: *interpolation*, see Ch. 9

Simplest option: piecewise linear interpolation ($\rightarrow$ Sect. 3.5.1) ➙ Euler polygon, see Fig. 203.

12.2.3 Abstract single step methods

Recall Euler methods for autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$:

explicit Euler: $\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k)$,

implicit Euler: $\mathbf{y}_{k+1}$: $\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_{k+1})$.

Both formulas provide a mapping

$$(\mathbf{y}_k, h_k) \mapsto \boldsymbol{\Psi}(h, \mathbf{y}_k) := \mathbf{y}_{k+1} \ . \tag{12.2.15}$$

Recall the interpretation of the $\mathbf{y}_k$ as approximations of $\mathbf{y}(t_k)$:

$$\boldsymbol{\Psi}(h, \mathbf{y}) \approx \boldsymbol{\Phi}^h \mathbf{y} \ , \tag{12.2.16}$$

where $\boldsymbol{\Phi}$ is the evolution operator ($\rightarrow$ Def. 12.1.27) for $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

The Euler methods provide approximations for evolution operator for ODEs

This is what every single step method does: it tries to approximate the evolution operator $\boldsymbol{\Phi}$ for an ODE by a mapping of the type (12.2.15).

➙ mapping $\boldsymbol{\Psi}$ from (12.2.15) is called discrete evolution.

Vice versa: a mapping $\boldsymbol{\Psi}$ as in (12.2.15) defines a single step method.

Definition 12.2.17 (Single step method (for autonomous ODE))**.**

Given a discrete evolution $\boldsymbol{\Psi} : \Omega \subset \mathbb{R} \times D \mapsto \mathbb{R}^d$*, an initial state* $\mathbf{y}_0$*, and a temporal mesh* $\mathcal{M} := \{t_0 < t_1 < \cdots < t_N = T\}$ *the recursion*

$$\mathbf{y}_{k+1} := \boldsymbol{\Psi}(t_{k+1} - t_k, \mathbf{y}_k) \;, \quad k = 0, \ldots, N-1 \;, \tag{12.2.18}$$

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:

$$\boldsymbol{\Psi}^h\mathbf{y} \quad \longleftrightarrow \quad \begin{array}{l} \texttt{function y1 = esvstep(h,y0)} \;. \\ \big(\,\texttt{function y1 = esvstep(@(y) rhs(y),h,y0)}\,\big) \end{array}$$

✎ Notation: $\boldsymbol{\Psi}^h\mathbf{y} := \boldsymbol{\Psi}(h, \mathbf{y})$

R. Hiptmair

rev 30401, February 19, 2011

Concept of single step method according to Def. 12.2.17 can be generalized to non-autonomous ODEs, which leads to recursions of the form:

$$\mathbf{y}_{k+1} := \boldsymbol{\Psi}(t_k, t_{k+1}, \mathbf{y}_k) \;, \quad k = 0, \ldots, N-1 \;,$$

for discrete evolution defined on $I \times I \times D$.

Remark 12.2.19 (Notation for single step methods).

Many authors specify a single step method by writing down the first step:

$$\mathbf{y}_1 = \text{expression in } \mathbf{y}_0 \text{ and } \mathbf{f} \;.$$

Also this course will sometimes adopt this practice.

△

12.3 Convergence of single step methods [10, Sect. 11.5]

Important issue: accuracy of approximation $\mathbf{y}_k \approx \mathbf{y}(t_k)$?

Remark 12.3.2 (Asymptotic perspective in convergence analysis).

As in the case of composite numerical quadrature, see Sect. 10.3: in general impossible to predict error $\|\mathbf{y}_N - \mathbf{y}(T)\|$ for particular choice of timesteps.

Tractable: asymptotic behavior of error for timestep $h := \max_k h_k \to 0$

Will tell us asymptotic gain in accuracy for extra computational effort.
(computational effort = no. of $\mathbf{f}$-evaluations)

△

Example 12.3.3 (Speed of convergence of Euler methods).

- IVP for Riccati ODE (12.2.2) on $[0, 1]$
- explicit Euler method (12.2.4) with uniform timestep $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.
- Error $\mathrm{err}_h := |y(1) - y_N|$

Observation:

algebraic convergence $\mathrm{err}_h = O(h)$

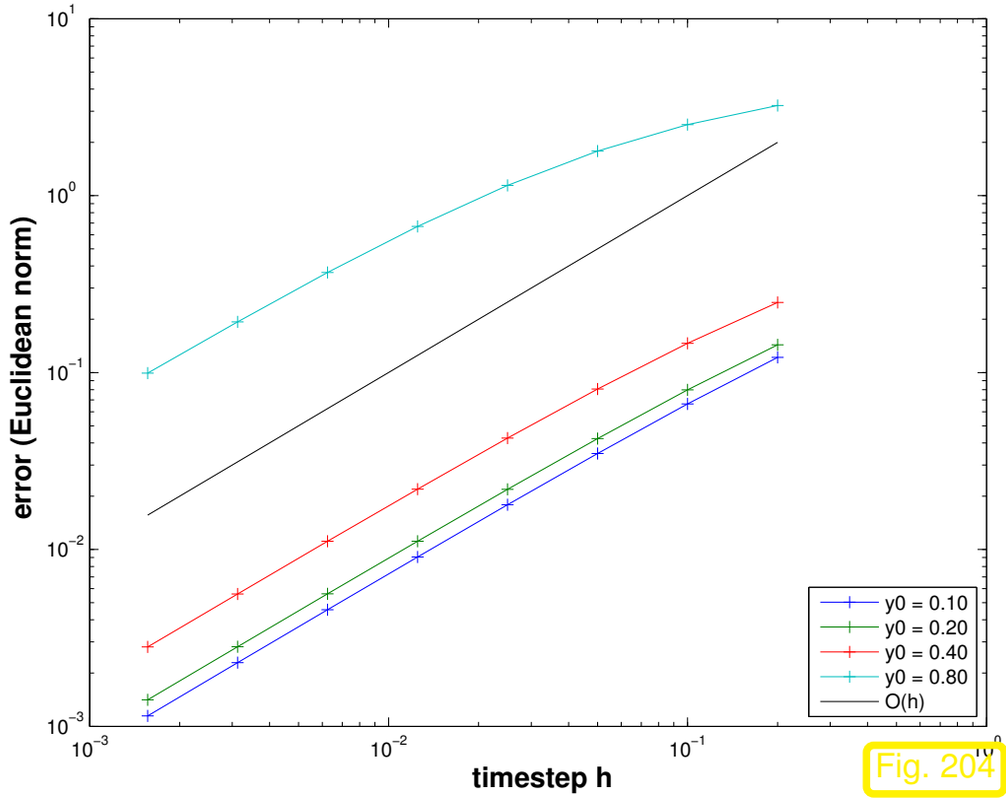


Fig. 204

- 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)/(12.2.10) with uniform timestep $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.
- Monitored: Error at final time $E(h) := |y(1) - y_N|$

R. Hiptmair

rev 30401, February 19, 2011

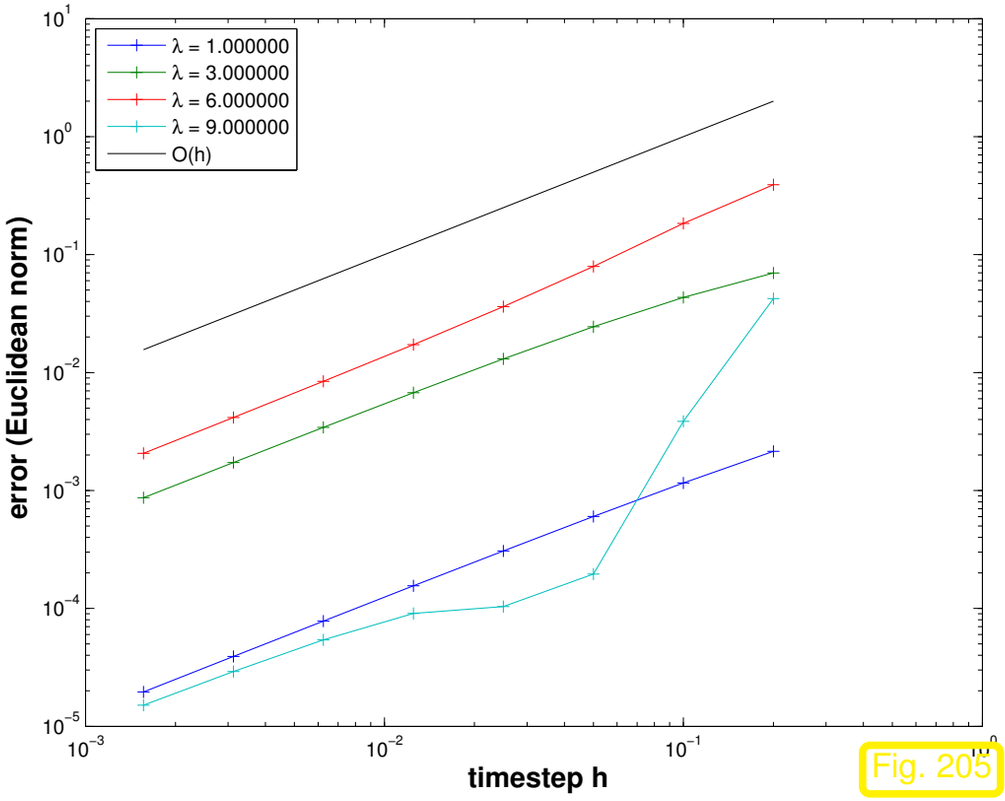


Fig. 205

explicit Euler method

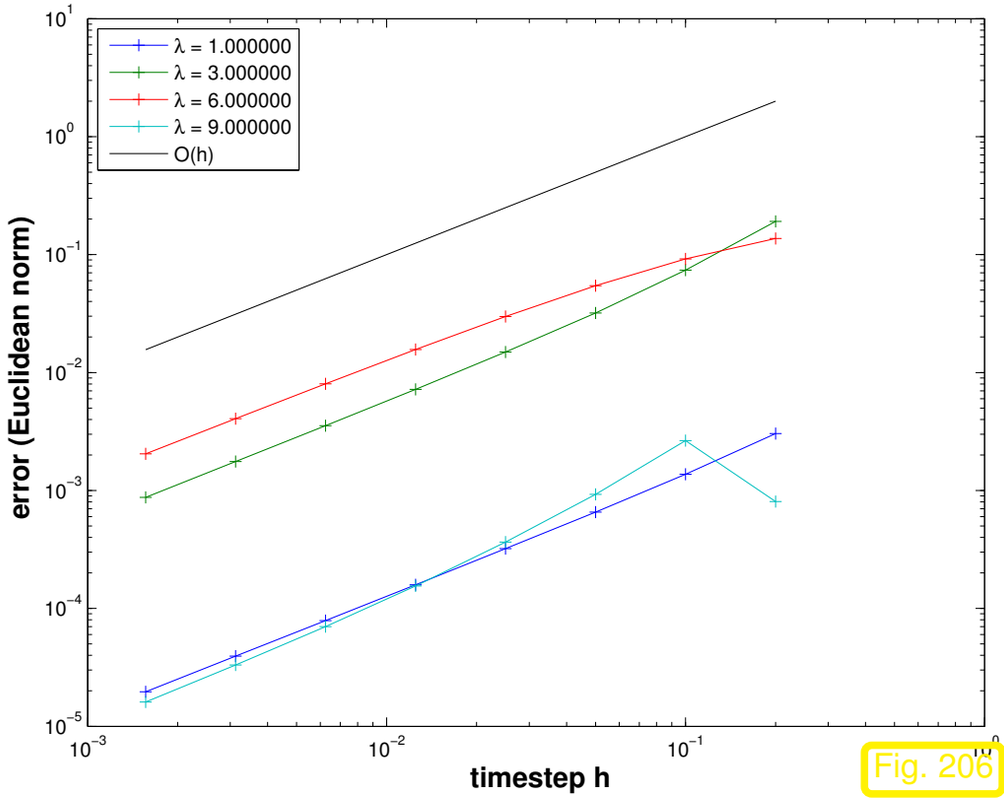


Fig. 206

implicit Euler method

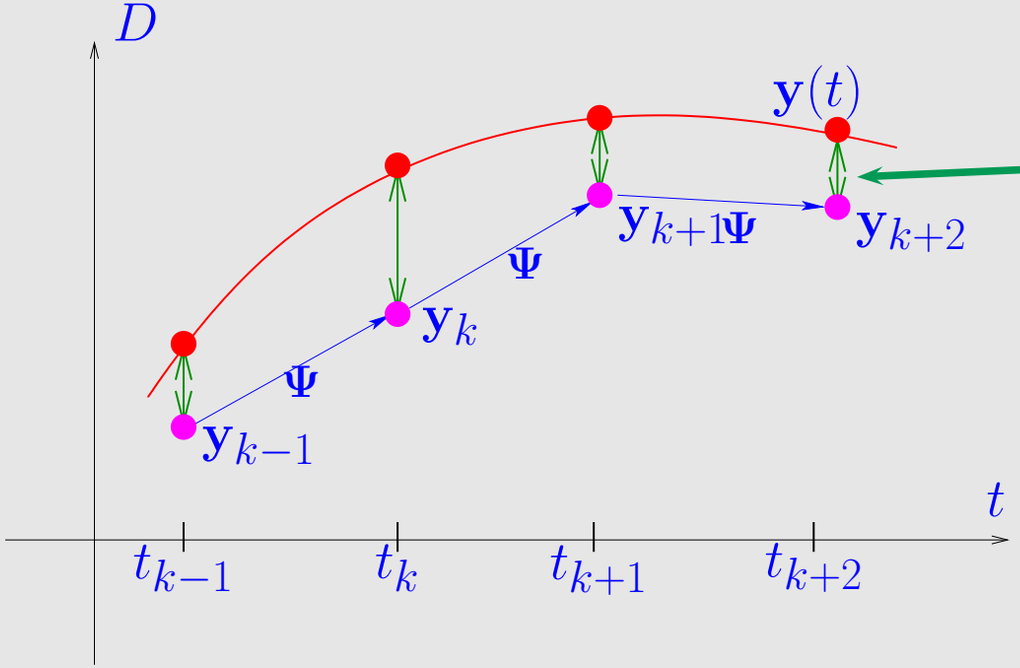
$O(h)$ algebraic convergence in both cases

◇

Convergence analysis [29, Ch. 74] for explicit Euler method (12.2.4) for autonomous IVP (12.1.13) with sufficiently smooth and (*globally*) Lipschitz continuous $\mathbf{f}$, that is,

$$\exists L > 0: \quad \|\mathbf{f}(t,\mathbf{y}) - \mathbf{f}(t,\mathbf{z})\| \leq L \|\mathbf{y} - \mathbf{z}\| \quad \forall \mathbf{y}, \mathbf{z} \in D \; . \tag{12.3.6}$$

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k) \;, \quad k = 1, \ldots, N-1 \;. \tag{12.2.4}$$



Error sequence: $\mathbf{e}_k := \mathbf{y}_k - \mathbf{y}(t_k)$.

— $t \mapsto \mathbf{y}(t)$

—

• $\mathbf{y}(t_k)$

• $\mathbf{y}_k$

$\longrightarrow$ discrete evolution $\boldsymbol{\Psi}^{t_{k+1}-t_k}$

Here and in what follows we rely on the abstract concepts of the evolution operator $\boldsymbol{\Phi}$ associated with the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ ($\rightarrow$ Def. 12.1.27) and discrete evolution operator $\boldsymbol{\Psi}$ defining the explicit Euler single step method, see Def. 12.2.17:

$$(12.2.4) \quad \Rightarrow \quad \boldsymbol{\Psi}^h \mathbf{y} = \mathbf{y} + h\mathbf{f}(\mathbf{y}) \;. \tag{12.3.7}$$

We argue that in this context the abstraction pays off, because it helps elucidate a general technique for the convergence analysis of single step methods.

Fundamental error splitting

$$\begin{aligned}\mathbf{e}_{k+1} &= \boldsymbol{\Psi}^{h_k}\mathbf{y}_k - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) \\ &= \underbrace{\boldsymbol{\Psi}^{h_k}\mathbf{y}_k - \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k)}_{\text{propagated error}} \\ &\quad + \underbrace{\boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k)}_{\text{one-step error}} .\end{aligned} \tag{12.3.22}$$

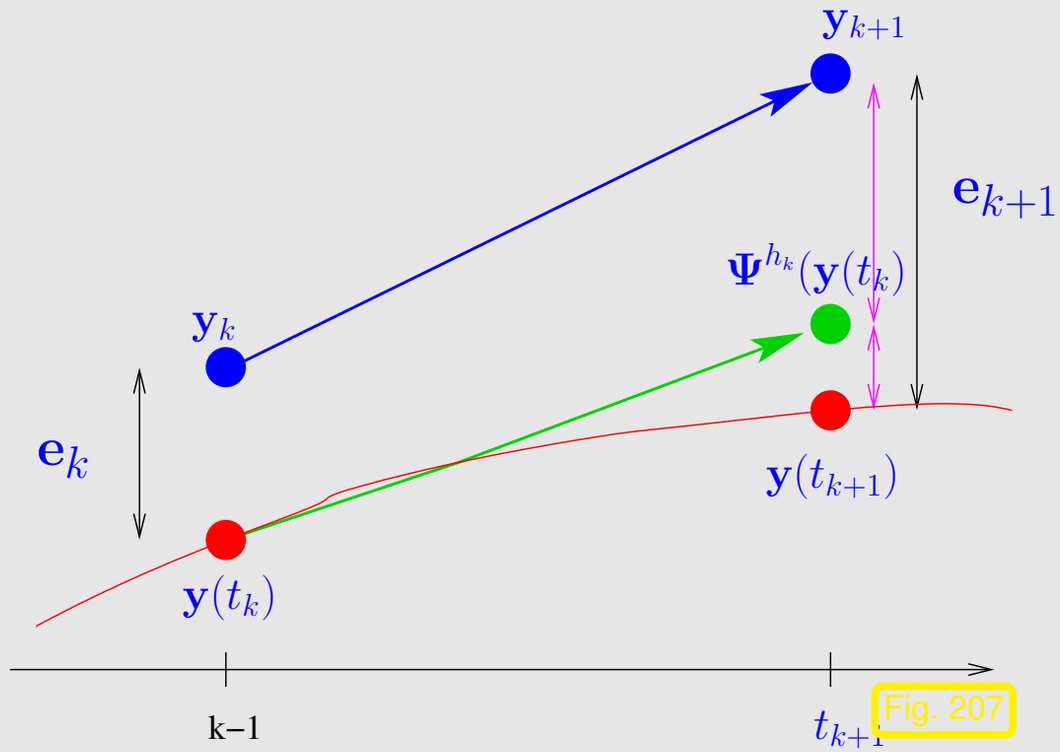


Fig. 207

R. Hiptmair

rev 30401, February 19, 2011

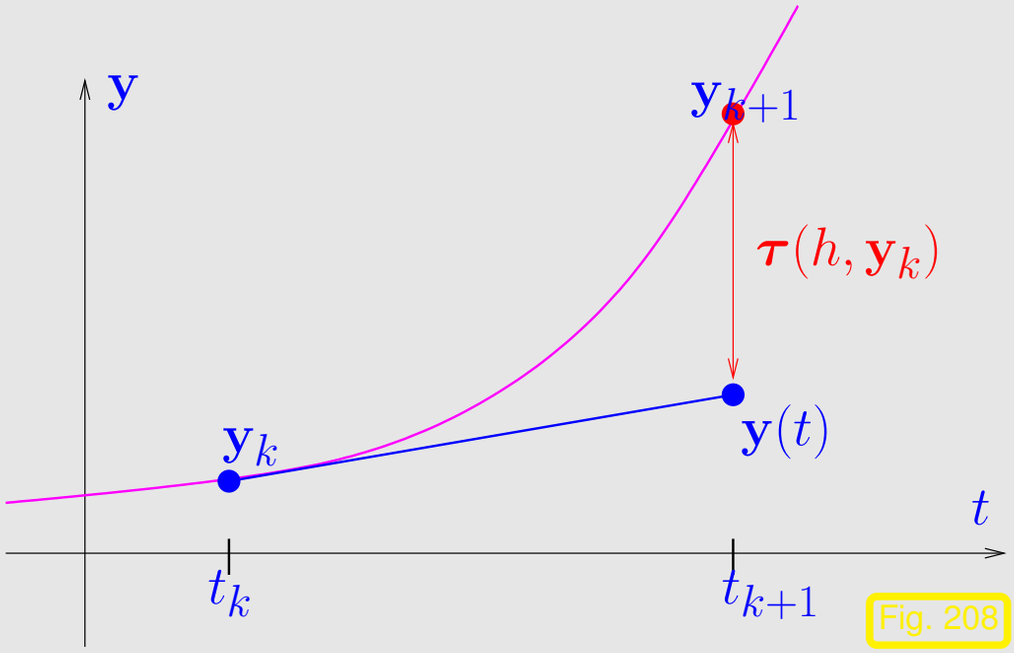


Fig. 208

one-step error:

$$\boldsymbol{\tau}(h,\mathbf{y}) := \boldsymbol{\Psi}^h\mathbf{y} - \boldsymbol{\Phi}^h\mathbf{y} \,. \qquad (12.3.23)$$

◁ geometric visualisation of one-step error for explicit Euler method (12.2.4), *cf.* Fig. 202.

R. Hiptmair

rev 30401, February 19, 2011

✎ notation: $t \mapsto \mathbf{y}(t) \mathrel{\hat{=}}$ (unique) solution of IVP, *cf.* Thm. 12.1.23.

② **Estimate for one-step error**:

Geometric considerations: distance of a smooth curve and its tangent shrinks as the square of the distance to the intersection point (curve locally looks like a parabola in the $\xi - \eta$ coordinate system, see Fig. 210).

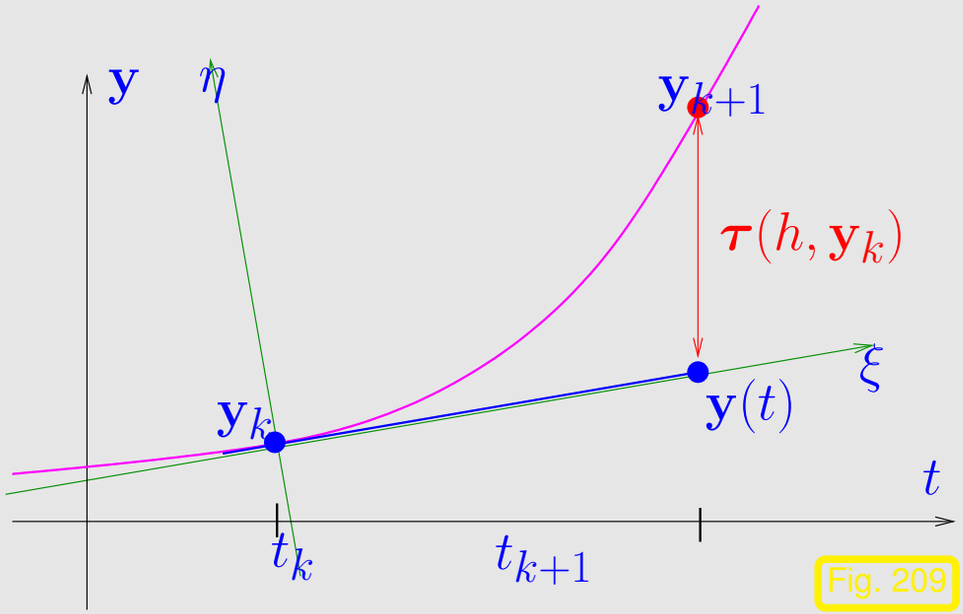


Fig. 209

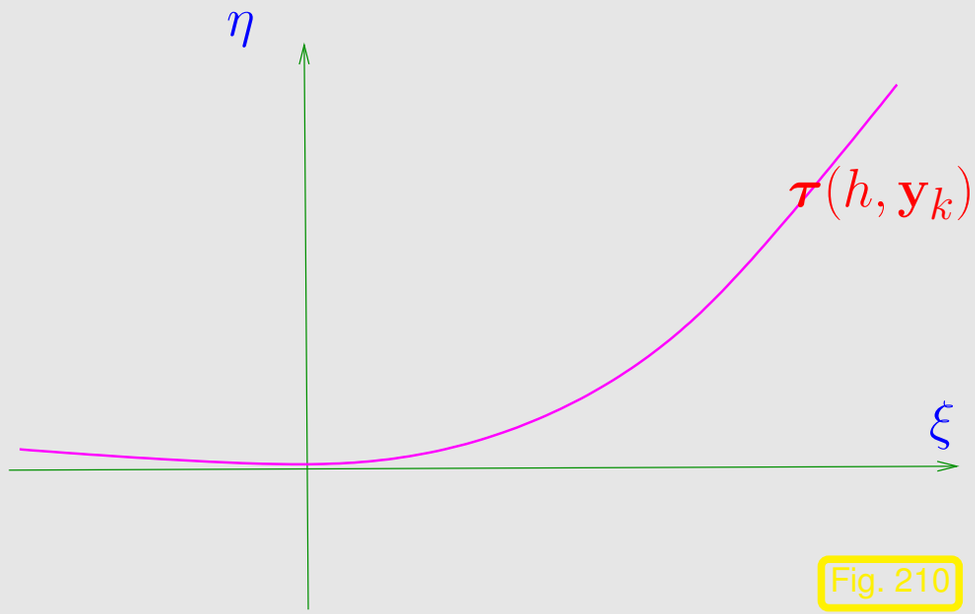


Fig. 210

Analytic considerations: recall Taylor's formula for function $\mathbf{y} \in C^{K+1}$

$$\mathbf{y}(t+h) - \mathbf{y}(t) = \sum_{j=0}^{K} \mathbf{y}^{(j)}(t)\frac{h^j}{j!} + \underbrace{\int_t^{t+h} f^{(K+1)}(\tau)\frac{(t+h-\tau)^K}{K!}\,\mathrm{d}\tau}_{= \dfrac{f^{(K+1)}(\xi)}{K!}h^{K+1}} , \tag{12.3.29}$$

for some $\xi \in [t, t+h]$

$\Rightarrow$ if $\mathbf{y} \in C^2([0,T])$, then

$\blacktriangleright$ $\mathbf{y}(t_{k+1}) - \mathbf{y}(t_k) = \dot{\mathbf{y}}(t_k)h_k + \frac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2$ for some $t_k \le \xi_k \le t_{k+1}$

R. Hiptmair

rev 30401, February 19, 2011

$$= \mathbf{f}(\mathbf{y}(t_k))h_k + \tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \ ,$$

since $t \mapsto \mathbf{y}(t)$ solves the ODE, which implies $\dot{\mathbf{y}}(t_k) = \mathbf{f}(\mathbf{y}(t_k))$. This leads to an expression for the one-step error from (12.3.23)

$$\begin{aligned} \boldsymbol{\tau}(h_k, \mathbf{y}(t_k)) &= \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) - \mathbf{y}(t_k + h_k) \\ &\overset{(12.3.7)}{=} \mathbf{y}(t_k) + h_k\mathbf{f}(\mathbf{y}(t_k)) - \mathbf{y}(t_k) - \mathbf{f}(\mathbf{y}(t_k))h_k + \tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \\ &= \tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \ . \end{aligned} \tag{12.3.30}$$

Sloppily speaking, we observe $\boxed{\boldsymbol{\tau}(h_k, \mathbf{y}(t_k)) = O(h_k^2)}$ uniformly for $h_k \to 0$.

③ **Estimate for the propagated error** from (12.3.22)

$$\begin{aligned} \left\| \boldsymbol{\Psi}^{h_k}\mathbf{y}_k - \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) \right\| &= \|\mathbf{y}_k + h_k\mathbf{f}(\mathbf{y}_k) - \mathbf{y}(t_k) - h_k\mathbf{f}(\mathbf{y}(t_k))\| \\ &\overset{(12.3.6)}{\leq} (1 + Lh_k)\left\|\mathbf{y}_k - \mathbf{y}(t_k)\right\| \ . \end{aligned} \tag{12.3.31}$$

③ *Recursion* for error norms $\epsilon_k := \|\mathbf{e}_k\|$ by $\triangle$-inequality:

$$\epsilon_{k+1} \leq (1 + h_k L)\epsilon_k + \rho_k \ , \quad \rho_k := \tfrac{1}{2}h_k^2 \max_{t_k \leq \tau \leq t_{k+1}} \|\ddot{\mathbf{y}}(\tau)\| \ . \tag{12.3.32}$$

Taking into account $\epsilon_0 = 0$ this leads to

$$\epsilon_k \leq \sum_{l=1}^{k} \prod_{j=1}^{l-1} (1 + Lh_j)\, \rho_l \;, \quad k = 1, \ldots, N \;. \tag{12.3.33}$$

Use the elementary estimate $(1 + Lh_j) \leq \exp(Lh_j)$ (by convexity of exponential function):

$$\text{(12.3.33)} \quad \Rightarrow \quad \epsilon_k \leq \sum_{l=1}^{k} \prod_{j=1}^{l-1} \exp(Lh_j) \cdot \rho_l = \sum_{l=1}^{k} \exp(L \sum_{j=1}^{l-1} h_j) \rho_l \;.$$

Note: $\sum_{j=1}^{l-1} h_j \leq T$ for final time T

$$\blacktriangleright \quad \epsilon_k \leq \exp(LT) \sum_{l=1}^{k} \rho_l \leq \exp(LT) \max_k \frac{\rho_k}{h_k} \sum_{l=1}^{k} h_l$$
$$\leq T \exp(LT) \max_{l=1,\ldots,k} h_l \cdot \max_{t_0 \leq \tau \leq t_k} \|\ddot{\mathbf{y}}(\tau)\| \;.$$

$$\blacktriangleright \quad \|\mathbf{y}_k - \mathbf{y}(t_k)\| \leq T \exp(LT) \max_{l=1,\ldots,k} h_l \cdot \max_{t_0 \leq \tau \leq t_k} \|\ddot{\mathbf{y}}(\tau)\| \;.$$

Total error arises from accumulation of one-step errors!

R. Hiptmair

rev 30401, February 19, 2011

- error bound $= O(h)$, $h := \max_l h_l$ (➤ 1st-order algebraic convergence)
- Error bound grows *exponentially* with the length T of the integration interval.

Most commonly used single step methods display algebraic convergence of integer order with respect to the meshwidth $h := \max_k h_k$. This offers a criterion for gauging their quality.

The sequence $(\mathbf{y}_k)_k$ generated by a

single step method ($\rightarrow$ Def. 12.2.17) 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

$$\max_k \|\mathbf{y}_k - \mathbf{y}(t_k)\| \leq Ch^p \quad \text{for} \quad h := \max_{k=1,\ldots,N} |t_k - t_{k-1}| \to 0 \,, \tag{12.3.34}$$

with $C > 0$ independent of $\mathcal{M}$, provided that $\mathbf{f}$ is *sufficiently smooth*, see [10, Thm. 11.25].

R. Hiptmair

rev 30401, February 19, 2011

12.4 Runge-Kutta methods [10, Sect. 11.6], [29, Ch. 76]

So far we only know first order methods, the explicit and implicit Euler method (12.2.4) and (12.2.10), respectively.

Do we need anything else?

Remark 12.4.2 (Rationale for high-order single step methods). *cf.* [10, Sect. 11.5.3]

Rem. 12.3.2 ➢ Due to unknown constants "$C > 0$" it is too ambitious to ask how many timesteps are needed so that $\|\mathbf{y}(T) - \mathbf{y}_N\|$ stays below a prescribed bound.

Question answered by *asymptotic estimates* like (12.3.34)

What extra computational effort buys a prescribed *reduction of the error*?

(*cf.* considerations in Sect. 4.3.3)

Usual concept for single step methods ($\rightarrow$ Def. 12.2.17)

Computational effort $\sim$ total number of $\mathbf{f}$-evaluations,

$\sim$ number of timesteps, if evaluation of discete evolution $\boldsymbol{\Psi}^h$ ($\rightarrow$ Def. 12.2.17)requires fixed number of $\mathbf{f}$-evaluations,

$\sim h^{-1}$, in the case of uniform timestep size $h > 0$.

$$\text{Goal:} \quad \frac{\mathrm{err}(h_{\mathrm{new}})}{\mathrm{err}(h_{\mathrm{old}})} \stackrel{!}{=} \rho \quad \text{for} \quad 0 < \rho < 1 \; .$$

$$(12.3.34) \quad \Rightarrow \quad \frac{h_{\mathrm{new}}^p}{h_{\mathrm{old}}^p} \stackrel{!}{=} \rho \quad \Leftrightarrow \quad \boxed{h_{\mathrm{new}} = \rho^{1/p} h_{\mathrm{old}}} \; .$$

For single step method of order $p \in \mathbb{N}$

increase effort by factor $\rho^{-1/p}$ ⟹ reduce error by factor ρ

☞ the larger the order p, the less effort for a prescribed reduction of the error!

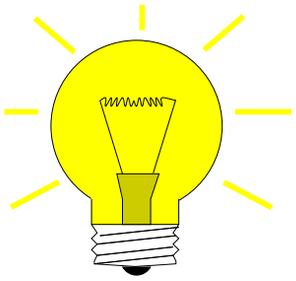
△

R. Hiptmair

rev 30401, February 19, 2011

Now we will build a class of methods that achieve orders > 1. The starting point is a simple *integral equation* satisfied by solutions of initial value problems:

$$\text{IVP:} \quad \begin{aligned} \dot{\mathbf{y}}(t) &= \mathbf{f}(t,\mathbf{y}(t)) \;, \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \quad \Rightarrow \quad \mathbf{y}(t_1) = \mathbf{y}_0 + \int_{t_0}^{t_1} \mathbf{f}(\tau,\mathbf{y}(\tau))\,\mathrm{d}\tau$$



Idea: approximate integral by means of s-point quadrature formula ($\rightarrow$ Sect. 10.1, defined on reference interval $[0,1]$) with nodes $c_1,\ldots,c_s$, weights $b_1,\ldots,b_s$.

$$\mathbf{y}(t_1) \approx \mathbf{y}_1 = \mathbf{y}_0 + h\sum_{i=1}^{s} b_i \mathbf{f}(t_0 + c_i h, \boxed{\mathbf{y}(t_0 + c_i h)}) \;, \quad h := t_1 - t_0 \;. \qquad (12.4.3)$$

Obtain these values by bootstrapping

bootstrapping = use the same idea in a simpler version to get $\mathbf{y}(t_0 + c_i h)$, noting that these values can be replaced by other approximations obtained by methods already constructed (this approach will be elucidated in the next example).

R. Hiptmair

rev 30401, February 19, 2011

What error can we afford in the approximation of $\mathbf{y}(t_0 + c_i h)$ (under the assumption that $\mathbf{f}$ is Lipschitz continuous)?

Goal: one-step error $\mathbf{y}(t_1) - \mathbf{y}_1 = O(h^{p+1})$

This goal can already be achieved, if only

$\mathbf{y}(t_0 + c_i h)$ is approximated up to an error $O(h^p)$,

because in (12.4.3) a factor of size h multiplies $\mathbf{f}(t_0 + c_i, \mathbf{y}(t_0 + c_i h))$.

This is accomplished by a less accurate discrete evolution than the one we are bidding for. Thus, we can construct discrete evolutions of higher and higher order, successively.

Example 12.4.4 (Construction of simple Runge-Kutta methods).

Quadrature formula = trapezoidal rule (10.2.7):

$$Q(f) = \tfrac{1}{2}(f(0) + f(1)) \quad \leftrightarrow \quad s = 2\text{:} \quad c_1 = 0, c_2 = 1\,, \quad b_1 = b_2 = \frac{1}{2}\,, \tag{12.4.5}$$

and $\mathbf{y}(T)$ approximated by explicit Euler step (12.2.4)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0) \;, \quad \mathbf{k}_2 = \mathbf{f}(t_0 + h, \mathbf{y}_0 + h\mathbf{k}_1) \;, \quad \mathbf{y}_1 = \mathbf{y}_0 + \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2) \;. \tag{12.4.6}$$

(12.4.6) = 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)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0) \;, \quad \mathbf{k}_2 = \mathbf{f}(t_0 + \tfrac{h}{2}, \mathbf{y}_0 + \tfrac{h}{2}\mathbf{k}_1) \;, \quad \mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{k}_2 \;. \tag{12.4.7}$$

(12.4.7) = explicit midpoint rule (for numerical integration of ODEs) [10, Alg. 11.18].

◇

Example 12.4.8 (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.

R. Hiptmair

rev 30401, February 19, 2011

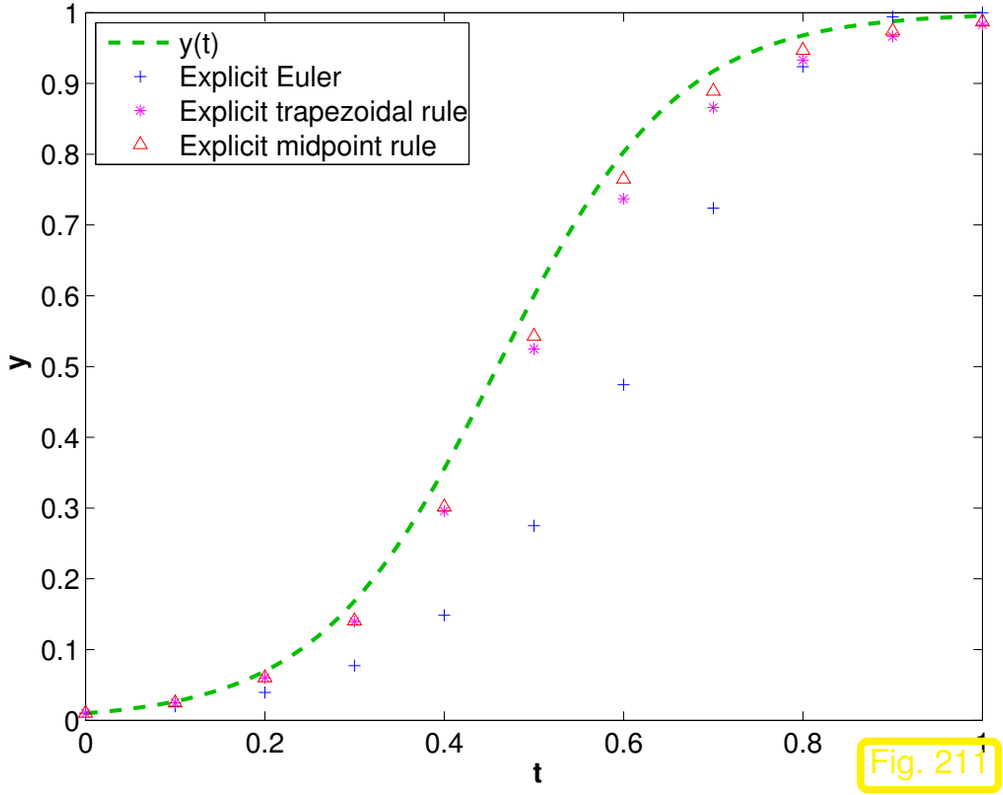


Fig. 211

$y_h(j/10)$, $j = 1, \ldots, 10$ for explicit RK-methods

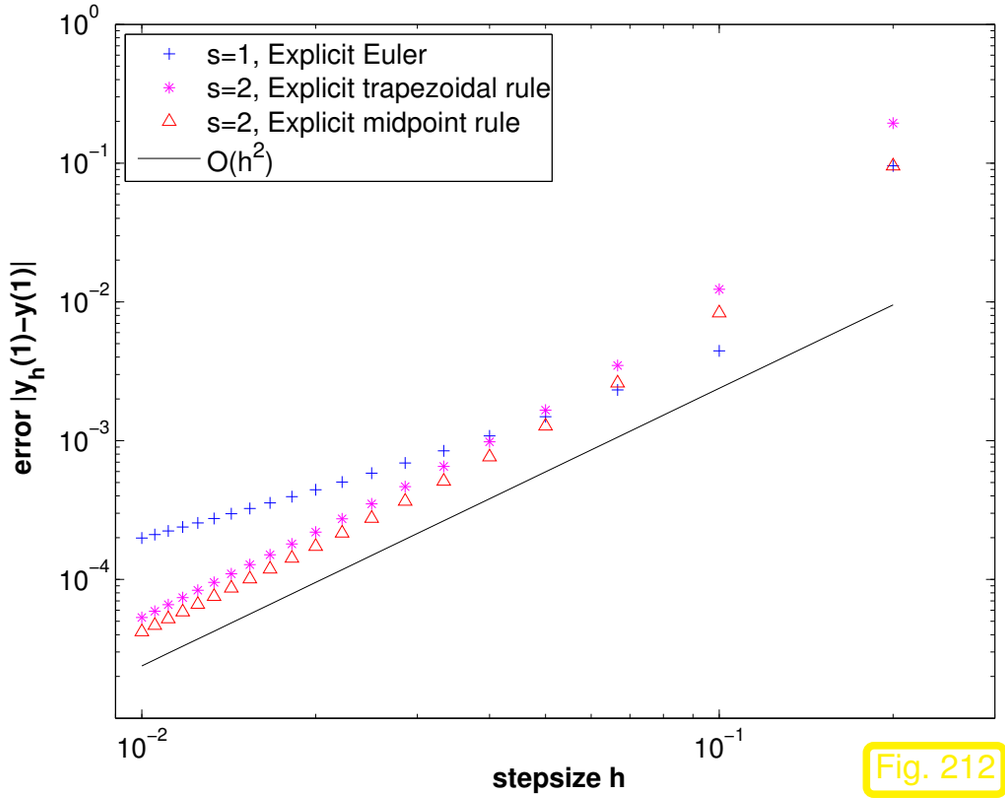


Fig. 212

Errors at final time $y_h(1) - y(1)$

Observation: obvious algebraic convergence with integer rates/orders

explicit trapezoidal rule (12.4.6) order 2
explicit midpoint rule (12.4.7) order 2

R. Hiptmair

rev 30401, February 19, 2011

The formulas that we have obtained follow a general pattern:

Definition 12.4.9 (Explicit Runge-Kutta method)**.**
For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{i-1} a_{ij}$, $i,j = 1,\ldots,s$, $s \in \mathbb{N}$, an s-stage explicit Runge-Kutta single step method (RK-SSM) for the IVP (12.1.13) *is defined by*

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h\sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j)\ , \quad i = 1,\ldots,s \quad , \quad \mathbf{y}_1 := \mathbf{y}_0 + h\sum_{i=1}^{s} b_i \mathbf{k}_i\ .$$

The $\mathbf{k}_i \in \mathbb{R}^d$ are called increments.

Recall Rem. 12.2.19 to understand how the discrete evolution for an explicit Runge-Kutta method is specified in this definition by giving the formulas for the first step. This is a convention widely adopted in the literature about numerical methods for ODEs. Of course, the increments $\mathbf{k}_i$ have to be computed anew in each timestep.

The implementation of an s-stage explicit Runge-Kutta single step method according to Def. 12.4.9 is straightforward: The increments $\mathbf{k}_i \in \mathbb{R}^d$ are computed successively, starting from $\mathbf{k}_1 = \mathbf{f}(t_0 + c_1 h, \mathbf{y}_0)$.

► Only s $\mathbf{f}$-evaluations and AXPY operations are required.

Shorthand notation for (explicit) Runge-Kutta methods [10, (11.75)]

Butcher scheme ▷

$$\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} := \begin{array}{c|ccc} c_1 & 0 & \cdots & 0 \\ c_2 & a_{21} & \ddots & \vdots \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{s,s-1} \; 0 \\ \hline & b_1 & \cdots & b_s \end{array} . \tag{12.4.10}$$

(Note: $\mathfrak{A}$ is strictly lower triangular $s \times s$-matrix)

Note that in Def. 12.4.9 the coefficients b_i can be regarded as weights of a quadrature formula on $[0,1]$: apply explicit Runge-Kutta single step method to "ODE" $\dot{y} = f(t)$.

Necessarily $\sum_{i=1}^{s} b_i = 1$

Example 12.4.11 (Butcher scheme for some explicit RK-SSM). [10, Sect. 11.6.1]

- Explicit Euler method (12.2.4):
$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \quad \rightarrowtail \quad \text{order = 1}$$

- explicit trapezoidal rule (12.4.6):
$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \quad \rightarrowtail \quad \text{order = 2}$$

- explicit midpoint rule (12.4.7):
$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array} \quad \rightarrowtail \quad \text{order = 2}$$

- Classical 4th-order RK-SSM:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array} \quad \succ \quad \text{order = 4}$$

- Kutta's 3/8-rule:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{2}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 1 & 1 & -1 & 1 & 0 \\ \hline & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{array} \quad \succ \quad \text{order = 4}$$

◇

Remark 12.4.12 ("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	$\geq p+3$

No general formula available so far

Known: order p $<$ number s of stages of RK-SSM

△

Remark 12.4.13 (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)

R. Hiptmair

rev 30401, February 19, 2011

Code 12.4.14: code excerpts for MATLAB integrator `ode45`

```
function varargout = ode45(ode,tspan,y0,options,varargin)
% Processing of input parameters omitted
% :
% Initialize method parameters.
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.
```


R. Hiptmair

rev 30401, February 19, 2011

```
30 ynew = y + f*hB(:,6);
31 % : (stepsize control, see Sect. 12.5 dropped
```


△

Example 12.4.15 (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 [10, Sect. 11.7]

Example 12.5.1 (Oregonator reaction).

See [29, Ch. 62] for the ODE-based modelling of kinetics of chemical reactions.

Special case of oscillating Zhabotinski-Belousov reaction [21]:

$$\begin{aligned}
\mathrm{BrO}_3^- + \mathrm{Br}^- &\mapsto \mathrm{HBrO}_2 \\
\mathrm{HBrO}_2 + \mathrm{Br}^- &\mapsto \mathrm{Org} \\
\mathrm{BrO}_3^- + \mathrm{HBrO}_2 &\mapsto 2\,\mathrm{HBrO}_2 + \mathrm{Ce(IV)} \\
2\,\mathrm{HBrO}_2 &\mapsto \mathrm{Org} \\
\mathrm{Ce(IV)} &\mapsto \mathrm{Br}^-
\end{aligned} \tag{12.5.2}$$

$$\begin{aligned}
y_1 := c(\mathrm{BrO}_3^-): &\quad \dot{y}_1 = -k_1 y_1 y_2 - k_3 y_1 y_3 \,, \\
y_2 := c(\mathrm{Br}^-): &\quad \dot{y}_2 = -k_1 y_1 y_2 - k_2 y_2 y_3 + k_5 y_5 \,, \\
y_3 := c(\mathrm{HBrO}_2): &\quad \dot{y}_3 = k_1 y_1 y_2 - k_2 y_2 y_3 + k_3 y_1 y_3 - 2k_4 y_3^2 \,, \\
y_4 := c(\mathrm{Org}): &\quad \dot{y}_4 = k_2 y_2 y_3 + k_4 y_3^2 \,, \\
y_5 := c(\mathrm{Ce(IV)}): &\quad \dot{y}_5 = k_3 y_1 y_3 - k_5 y_5 \,,
\end{aligned} \tag{12.5.3}$$

with (non-dimensionalized) reaction constants:

$$k_1 = 1.34\,, \quad k_2 = 1.6 \cdot 10^9\,, \quad k_3 = 8.0 \cdot 10^3\,, \quad k_4 = 4.0 \cdot 10^7\,, \quad k_5 = 1.0\,.$$

R. Hiptmair

rev 30401, February 19, 2011

MATLAB simulation with initial state $y_1(0) = 0.06$, $y_2(0) = 0.33 \cdot 10^{-6}$, $y_3(0) = 0.501 \cdot 10^{-10}$, $y_4(0) = 0.03$, $y_5(0) = 0.24 \cdot 10^{-7}$:

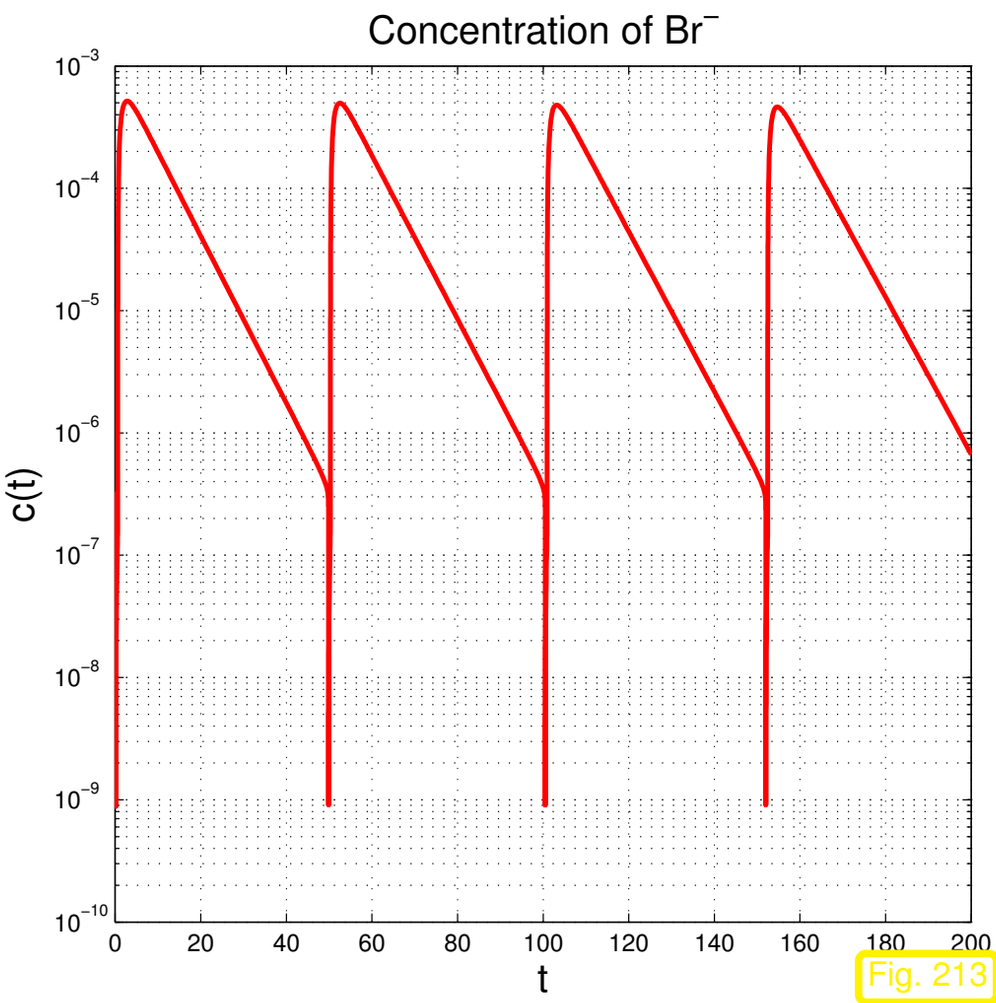


Fig. 213

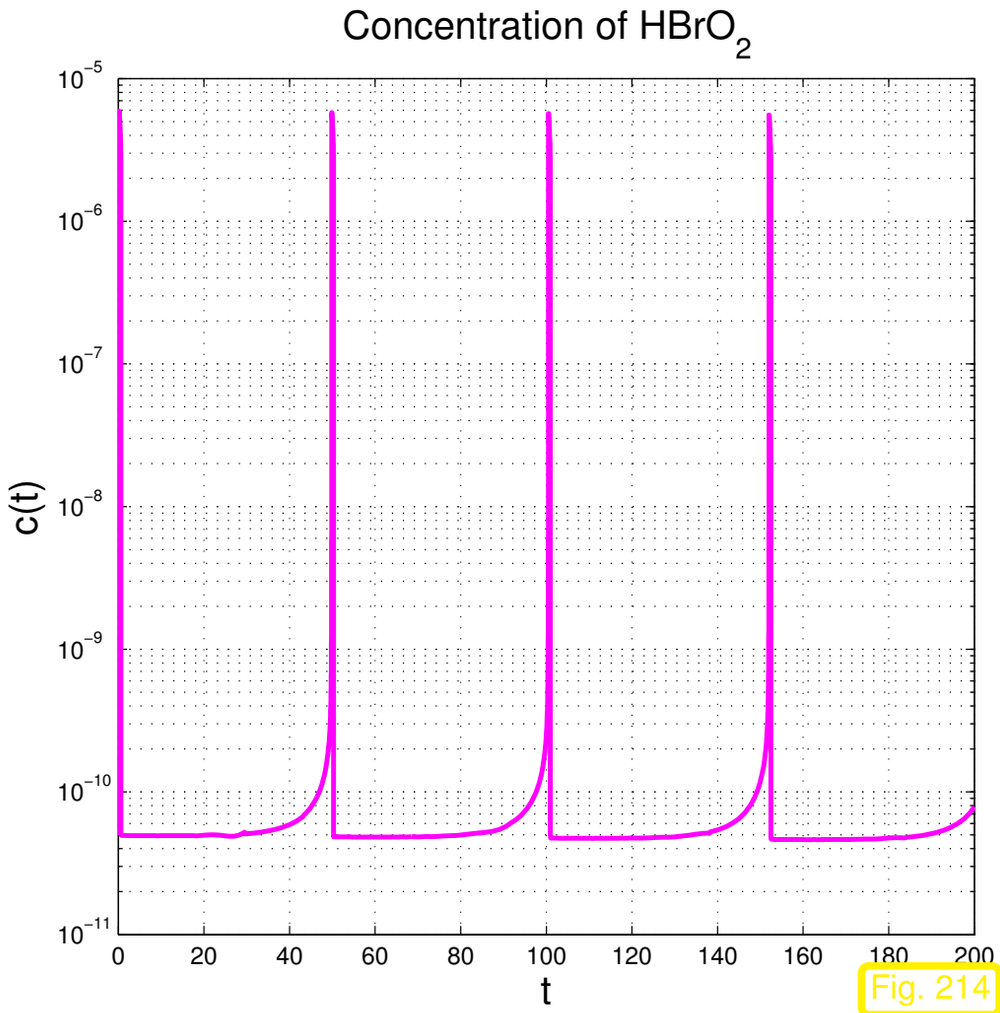


Fig. 214

R. Hiptmair

rev 30401, February 19, 2011

We observe a strongly non-uniform behavior of the solution in time.

This is very common with evolutions arising from practical models (circuit models, chemical reaction models, mechanical systems)

◇

Example 12.5.4 (Blow-up).

Scalar autonomous IVP:

$$\dot{y} = y^2 \;, \quad y(0) = y_0 > 0 \;.$$

$$\blacktriangleright \quad y(t) = \frac{y_0}{1 - y_0 t} \;, \quad t < 1/y_0 \;.$$

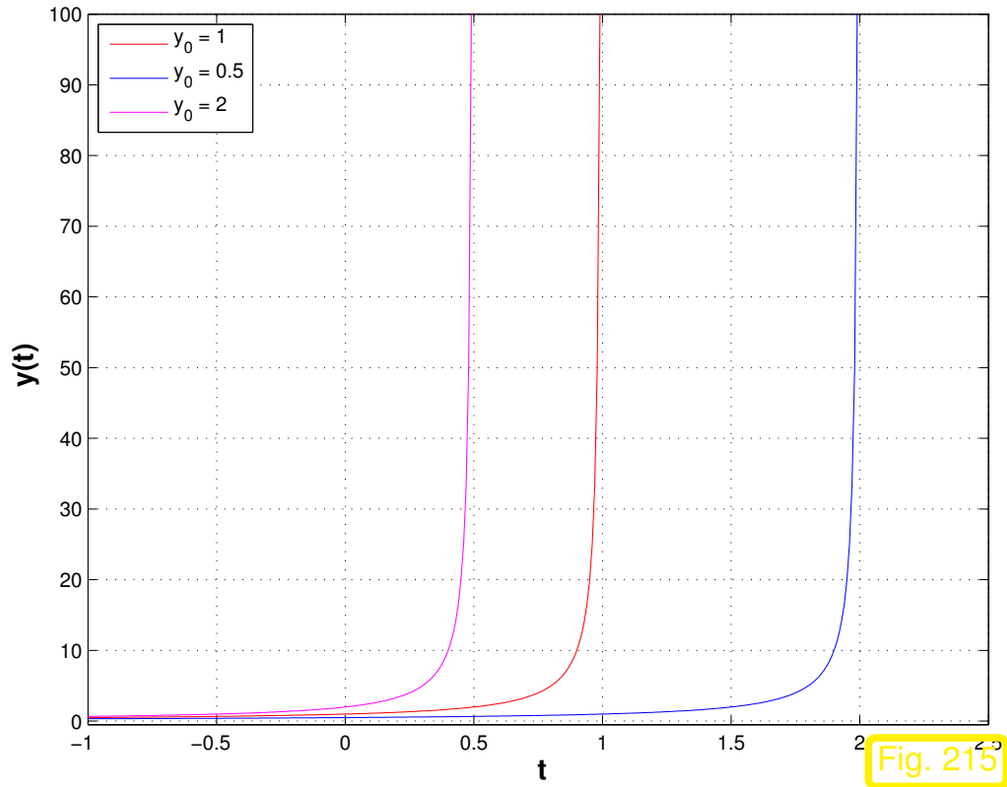


Fig. 215

Solution exists only for finite time and then suffers a Blow-up, that is, $\lim_{t \to 1/y_0} y(t) = \infty$: $J(y_0) =]-\infty, 1/y_0]$!

R. Hiptmair

rev 30401, February 19, 2011

How to choose temporal mesh $\{t_0 < t_1 < \cdots < t_{N-1} < t_N\}$ for single step method in case $J(y_0)$ is not known, even worse, if it is not clear a priori that a blow up will happen?

Just imagine: what will result from equidistant explicit Euler integration (12.2.4) applied to the above IVP?

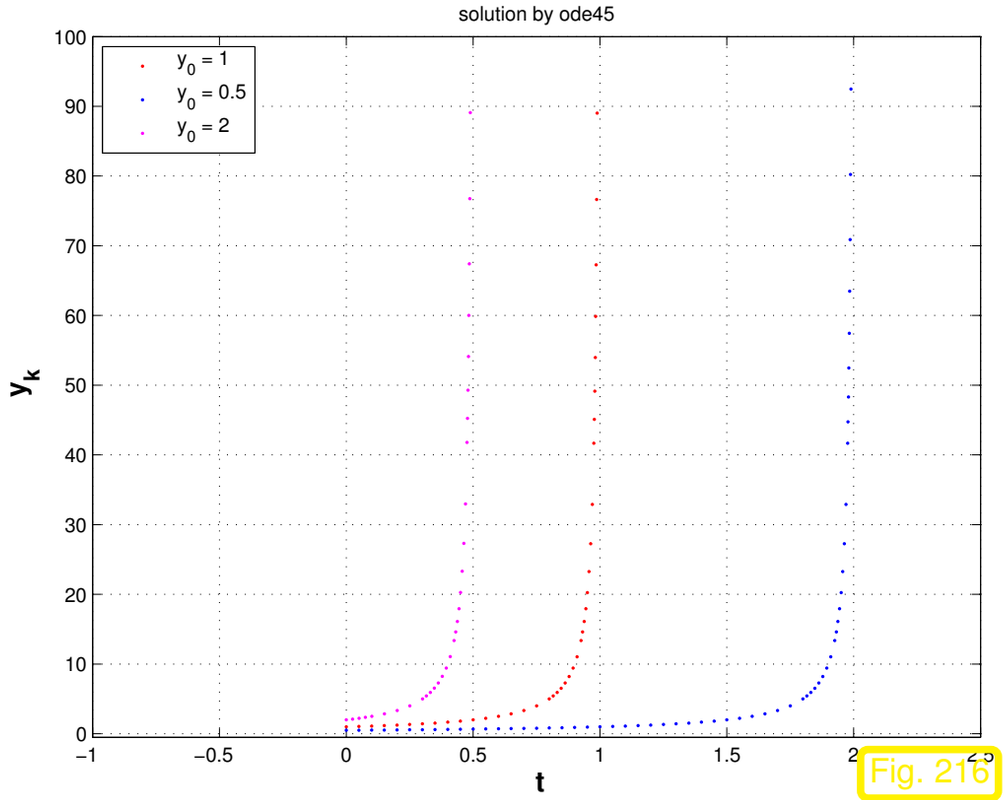


Fig. 216

```
1 fun = @(t,y) y.^2;
2 [t1,y1] = ode45(fun,[0 2],1);
3 [t2,y2] = ode45(fun,[0 2],0.5);
4 [t3,y3] = ode45(fun,[0 2],2);
```

MATLAB warning messages:

R. Hiptmair

rev 30401, February 19, 2011

```
  Warning: Failure at t=9.999694e-01.  Unable to meet integration
  tolerances without reducing the step size below the smallest
  value allowed (1.776357e-15) at time t.
> In ode45 at 371
  In simpleblowup at 22
Warning: Failure at t=1.999970e+00.  Unable to meet integration
  tolerances without reducing the step size below the smallest
  value allowed (3.552714e-15) at time t.
> In ode45 at 371
  In simpleblowup at 23
Warning: Failure at t=4.999660e-01.  Unable to meet integration
  tolerances without reducing the step size below the smallest
  value allowed (8.881784e-16) at time t.
> In ode45 at 371
  In simpleblowup at 24
```

We observe: `ode45` manages to reduce stepsize more and more as it approaches the singularity of the solution!

◇

Key issue (discussed for autonomous ODEs below):

Choice of *good temporal mesh* $\{0 = t_0 < t_1 < \cdots < t_{N-1} < t_N\}$
for a given single step method applied to an IVP

What does “good” mean ?

be efficient — be accurate

Objective: N as small as possible **&** $\max_{k=1,\ldots,N} \|\mathbf{y}(t_k) - \mathbf{y}_k\| < \texttt{TOL}$ or $\|\mathbf{y}(T) - \mathbf{y}_N\| < \texttt{TOL}$, 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 $\mathbf{y}_k, h_{k-1}$)

R. Hiptmair

rev 30401, February 19, 2011

Why local-in-time timestep control (based on estimating the one-step error)**?**

Consideration: If a small time-local error in a single timestep leads to large error $\|\mathbf{y}_k - \mathbf{y}(t_k)\|$ at later times, then local-in-time timestep control is powerless about it and will not even notice!!

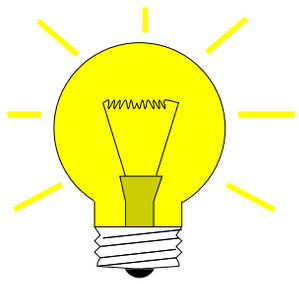
Nevertheless, local-in-time timestep control is used almost exclusively,

- ☞ because we do not want to discard past timesteps, which could amount to tremendous waste of computational resources,
- ☞ because it is inexpensive and it works for many practical problems,
- ☞ because there is no reliable method that can deliver guaranteed accuracy for general IVP.

"Recycle" heuristics already employed for adaptive quadrature, see Sect. 10.5:

R. Hiptmair

rev 30401, February 19, 2011



Idea: *Estimation of one-step error, cf.* Sect. 10.5

Compare two discrete evolutions $\boldsymbol{\Psi}^h$, $\widetilde{\boldsymbol{\Psi}}^h$ of different order for *current timestep* h:

If Order$(\widetilde{\boldsymbol{\Psi}}) >$ Order$(\boldsymbol{\Psi})$

$$\Rightarrow \quad \underbrace{\boldsymbol{\Phi}^h\mathbf{y}(t_k) - \boldsymbol{\Psi}^h\mathbf{y}(t_k)}_{\text{one-step error}} \approx \mathtt{EST}_k := \widetilde{\boldsymbol{\Psi}}^h\mathbf{y}(t_k) - \boldsymbol{\Psi}^h\mathbf{y}(t_k)\ . \tag{12.5.5}$$

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

$\blacktriangleright$ Simple algorithm:

$\mathtt{EST}_k < \max\{\mathtt{ATOL}, \|\mathbf{y}_k\|\,\mathtt{RTOL}\}$: Carry out next timestep (stepsize h)

Use larger stepsize (e.g., αh with some $\alpha > 1$) for following step

R. Hiptmair

rev 30401, February 19, 2011

$\mathtt{EST}_k > \max\{\mathtt{ATOL}, \|\mathbf{y}_k\|\,\mathtt{RTOL}\}$: *Repeat* current step with smaller stepsize $< h$, e.g., $\frac{1}{2}h$

Rationale for $(*)$: if the current stepsize guarantees sufficiently small one-step error, then it might be possible to obtain a still acceptable one-step error with a larger timestep, which would enhance efficiency (fewer timesteps for total numerical integration). This should be tried, since timestep control will usually provide a safeguard against undue loss of accuracy.

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);   %
  yH = Psilow(h,y0);    %
  est = norm(yH-yh);    %

  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;                                                %
  else, h = h/2; end                                        %
end
```

Comments on Code 12.5.6:

- Input arguments:
  - `Psilow`, `Psihigh`: function handles to discrete evolution operators for autonomous ODE of different order, type `@(y,h)`, expecting a state (column) vector as first argument, and a stepsize as second,
  - `T`: final time $T > 0$,
  - `y0`: initial state $\mathbf{y}_0$,
  - `h0`: stepsize h_0 for the first timestep
  - `reltol`, `abstol`: relative and absolute tolerances, see (12.5.6),
  - `hmin`: minimal stepsize, timestepping terminates when stepsize control $h_k < h_{\min}$, which is relevant for detecting blow-ups or collapse of the solution.
- line 3: check whether final time is reached or timestepping has ground to a halt ($h_k < h_{\min}$).
- line 4, 5: advance state by low and high order integrator.
- line 6: compute norm of estimated error, see (12.5.5).
- line 8: make comparison (12.5.6) to decide whether to accept or reject local step.

- line 9, 10: step accepted, update state and current time and suggest 1.1 times the current stepsize for next step.
- line 11 step rejected, try again with half the stepsize.
- Return values:
  - `t`: temporal mesh $t_0 < t_1 < t_2 < \ldots < t_N < T$, where $t_N < T$ indicated premature termination (collapse, blow-up),
  - `y`: sequence $(\mathbf{y}_k)_{k=0}^N$.

! By the heuristic considerations, see (12.5.5) it seems that EST_k measures the one-step error for the low-order method $\boldsymbol{\Psi}$ and that we should use $\mathbf{y}_{k+1} = \boldsymbol{\Psi}^{h_k}\boldsymbol{y}_k$, if the timestep is accepted.

However, it would be foolish not to use the better value $\mathbf{y}_{k+1} = \widetilde{\boldsymbol{\Psi}}^{h_k}\boldsymbol{y}_k$, since it is available for free. This is what is done in every implementation of adaptive methods, also in Code 12.5.6, and this choice can be justified by control theoretic arguments [13, Sect. 5.2].

Example 12.5.8 (Simple adaptive stepsize control).

- IVP for ODE $\dot{y} = \cos(\alpha y)^2$, $\alpha > 0$, solution $y(t) = \arctan(\alpha(t-c))/\alpha$ for $y(0) \in]-\pi/2, \pi/2[$

R. Hiptmair
rev 30401, February 19, 2011

- Simple adaptive timestepping based on explicit Euler (12.2.4) and explicit trapezoidal rule (12.4.6)

Code 12.5.10: MATLAB function for Ex. 12.5.8

```
function odeintadaptdriver(T,a,reltol,abstol)
% Simple adaptive timestepping strategy of Code 12.5.6
% based on explicit Euler (12.2.4) and explicit trapezoidal
% rule (12.4.6)

% Default arguments
if (nargin < 4), abstol = 1E-4; end
if (nargin < 3), reltol = 1E-2; end
if (nargin < 2), a = 20; end
if (nargin < 1), T = 2; end

% autonomous ODE ẏ = cos(ay) and its general solution
f = @(y) (cos(a*y).^2); sol = @(t) (atan(a*(t-1))/a);
% Initial state y_0
y0 = sol(0);

% Discrete evolution operators, see Def. 12.2.17
Psilow = @(h,y) (y + h*f(y)); % Explicit Euler (12.2.4)
% Explicit trapzoidal rule (12.4.6)
Psihigh = @(h,y) (y + 0.5*h*(f(y)+f(y+h*f(y))));

% Heuristic choice of initial timestep and h_min
h0 = T/(100*(norm(f(y0))+0.1)); hmin = h0/10000;
```

R. Hiptmair

rev 30401, February 19, 2011

```
% Main adaptive timestepping loop, see Code 12.5.6
[t,y,rej,ee] = odeintadapt_ext(Psilow,Psihigh,T,y0,h0,reltol,abstol,hmin);

% Plotting the exact the approximate solutions and rejected timesteps
figure('name','solutions');
tp = 0:T/1000:T; plot(tp,sol(tp),'g-','linewidth',2); hold on;
plot(t,y,'r.');
plot(rej,0,'m+');
title(sprintf('Adaptive timestepping, rtol = %f, atol = %f, a =
  %f',reltol,abstol,a));
xlabel('{\bf t}','fontsize',14);
ylabel('{\bf y}','fontsize',14);
legend('y(t)','y_k','rejection','location','northwest');
print -depsc2 '../PICTURES/odeintadaptsol.eps';

fprintf('%d timesteps, %d rejected timesteps\n',length(t)-1,length(rej));

% Plotting estimated and true errors
figure('name','(estimated) error');
plot(t,abs(sol(t) - y),'r+',t,ee,'m*');
xlabel('{\bf t}','fontsize',14);
ylabel('{\bf error}','fontsize',14);
legend('true error |y(t_k)-y_k|','estimated error
  EST_k','location','northwest');
title(sprintf('Adaptive timestepping, rtol = %f, atol = %f, a =
  %f',reltol,abstol,a));
print -depsc2 '../PICTURES/odeintadapterr.eps';
```


R. Hiptmair

rev 30401, February 19, 2011

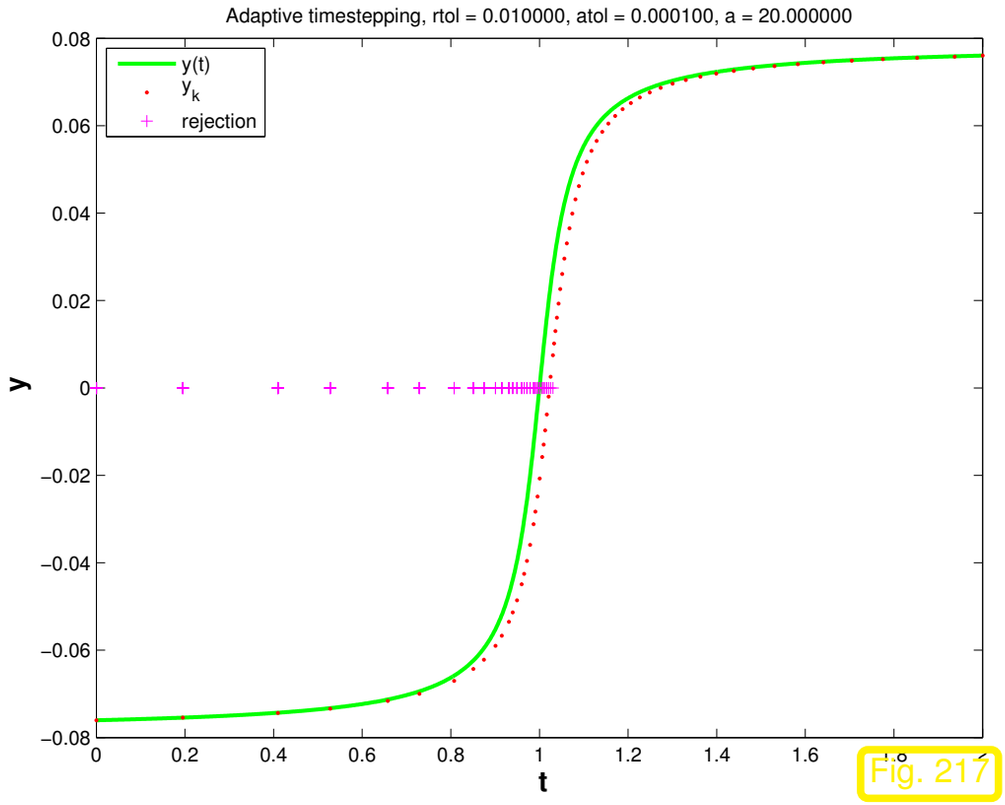


Fig. 217

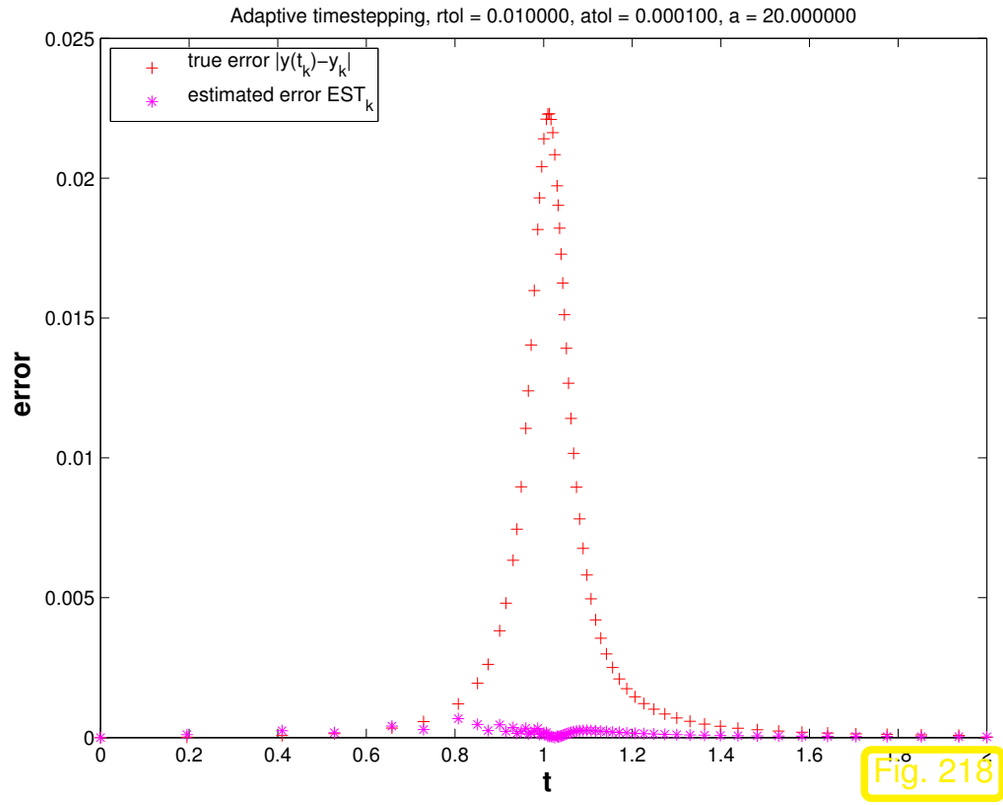


Fig. 218

Statistics: `66 timesteps, 131 rejected timesteps`

Observations:

- ☞ Adaptive timestepping well resolves local features of solution $y(t)$ at $t = 1$
- ☞ Estimated error (an estimate for the one-step error) and true error are **not** related!

R. Hiptmair

rev 30401, February 19, 2011

Example 12.5.11 (Gain through adaptivity). → Ex. 12.5.8

Simple adaptive timestepping from previous experiment Ex. 12.5.8.

New: initial state $y(0) = 0$!

Now we study the dependence of the maximal point error on the computational effort, which is proportional to the number of timesteps.

Code 12.5.13: MATLAB function for Ex. 12.5.11

```
1 function adaptgain(T,a,reltol,abstol)
2 % Experimental study of gasin through simple adaptive timestepping
3 % strategy of Code 12.5.6 based on explicit Euler
4 % (12.2.4) and explicit trapezoidal
```

```
% rule (12.4.6)

% Default arguments
if (nargin < 4), abstol = 1E-3; end
if (nargin < 3), reltol = 1E-1; end
if (nargin < 2), a = 40; end
if (nargin < 1), T = 2; end

% autonomous ODE ẏ = cos(ay) and its general solution
f = @(y) (cos(a*y).^2); sol = @(t) (atan(a*(t))/a);
% Initial state y_0
y0 = sol(0);

% Discrete evolution operators, see Def. 12.2.17
Psilow = @(h,y) (y + h*f(y)); % Explicit Euler (12.2.4)
% Explicit trapzoidal rule (12.4.6)
Psihigh = @(h,y) (y + 0.5*h*(f(y)+f(y+h*f(y))));

% Lop over uniform timesteps of varying length and integrate ODE by explicit
  trapezoidal
% rule (12.4.6)
erruf = [];
for N=10:10:200
  h = T/N; t = 0; y = y0; err = 0;
```

R. Hiptmair

rev 30401, February 19, 2011

```
28    for k=1:N
29      y = Psihigh(h,y); t = t+h;
30      err = max(err,abs(sol(t) - y));
31    end
32    erruf = [erruf;N, err];
33  end
34
35  % Run adaptive timestepping with various tolerances, which is the only way
36  % to make it use a different total number of timesteps.
37  % Plot the solution sequences for different values of the relative tolerance.
38  figure('name','adaptive timestepping');
39  axis([0 2 0 0.05]); hold on; col = colormap;
40  errad = []; l = 1;
41  for rtol=reltol*2.^(2:-1:-4)
42  % Crude choice of initial timestep and h_min
43    h0 = T/10; hmin = h0/10000;
44  % Main adaptive timestepping loop, see Code 12.5.6
45    [t,y,rej,ee] =
        odeintadapt_ext(Psilow,Psihigh,T,y0,h0,rtol,0.01*rtol,hmin);
46    errad = [errad; length(t)-1, max(abs(sol(t) - y)), rtol,
        length(rej)];
47    fprintf('rtol = %d: %d timesteps, %d rejected timesteps\n',
        rtol,length(t)-1,length(rej));
```


R. Hiptmair

rev 30401, February 19, 2011

```
48    plot(t,y,'.','color',col(10*(l-1)+1,:));
49    leg{l} = sprintf('rtol = %f',rtol); l = l+1;
50  end
51  xlabel('{\bf t}','fontsize',14);
52  ylabel('{\bf y}','fontsize',14);
53  legend(leg,'location','southeast');
54  title(sprintf('Solving d_t y = a cos(y)^2 with a = %f by simple
      adaptive timestepping',a));
55  print -depsc2 '../PICTURES/adaptgainsol.eps';
56
57  % Plotting the errors vs.  the number of timesteps
58  figure('name','gain by adaptivity');
59  loglog(erruf(:,1), erruf(:,2),'r+',errad(:,1), errad(:,2),'m*');
60  xlabel('{\bf no. N of timesteps}','fontsize',14);
61  ylabel('{\bf max_k|y(t_k)-y_k|}','fontsize',14);
62  title(sprintf('Error vs. no. of timesteps for d_t y = a cos(y)^2
      with a = %f',a));
63  legend('uniform timestep','adaptive
      timestep','location','northeast');
64  print -depsc2 '../PICTURES/adaptgain.eps';
```

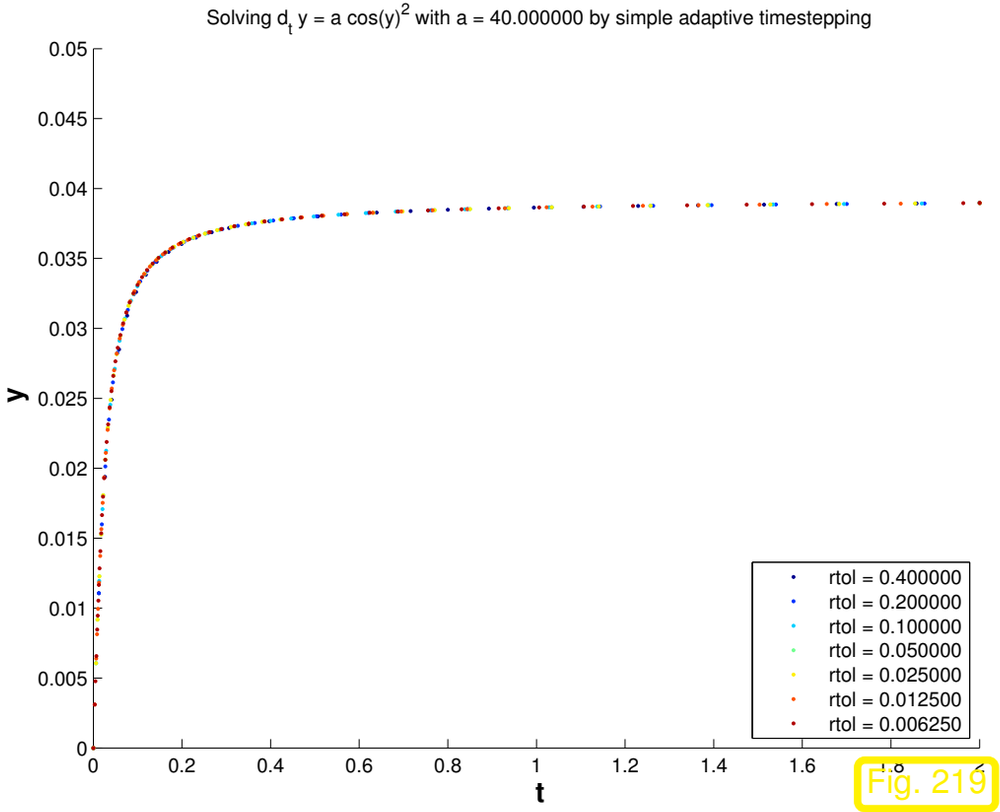


Fig. 219

Solutions $(\mathbf{y}_k)_k$ for different values of `rtol`

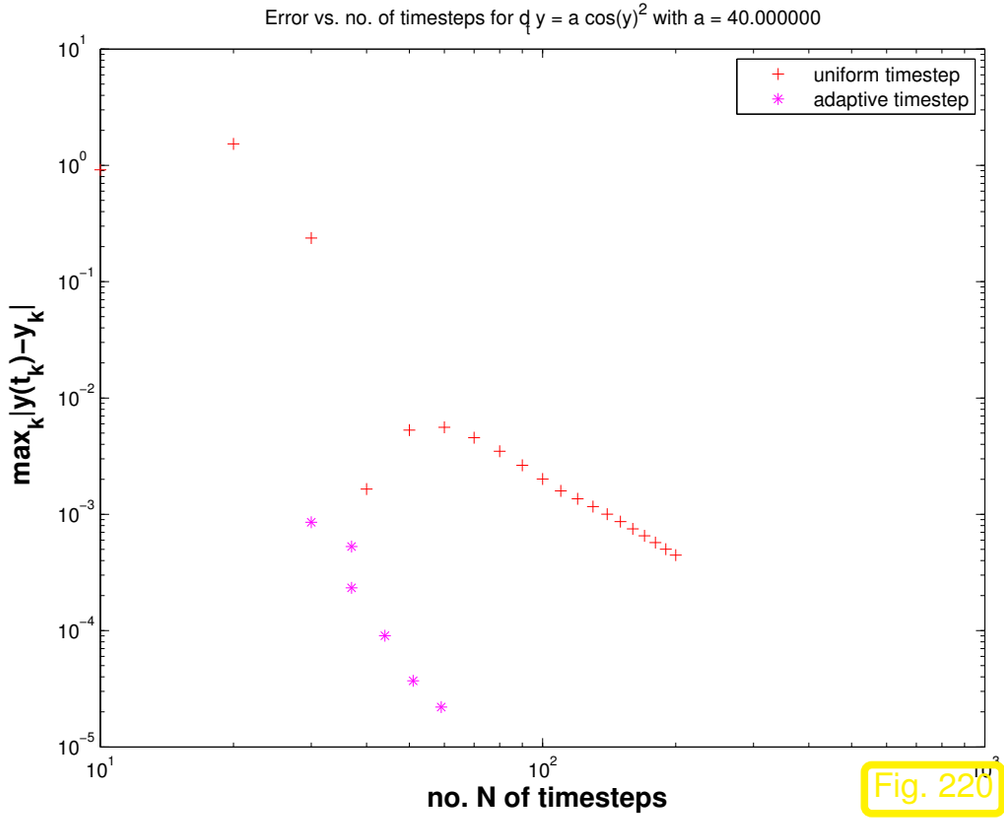


Fig. 220

Error vs. computational effort

Observations:

☞ Adaptive timestepping achieves much better accuracy for a fixed computational effort.

R. Hiptmair

rev 30401, February 19, 2011

Example 12.5.14 (“Failure” of adaptive timestepping). → Ex. 12.5.11

Same ODE and simple adaptive timestepping as in previous experiment Ex. 12.5.11. Same evaluations.

Now: initial state $y(0) = -0.0386$ as in Ex. 12.5.8

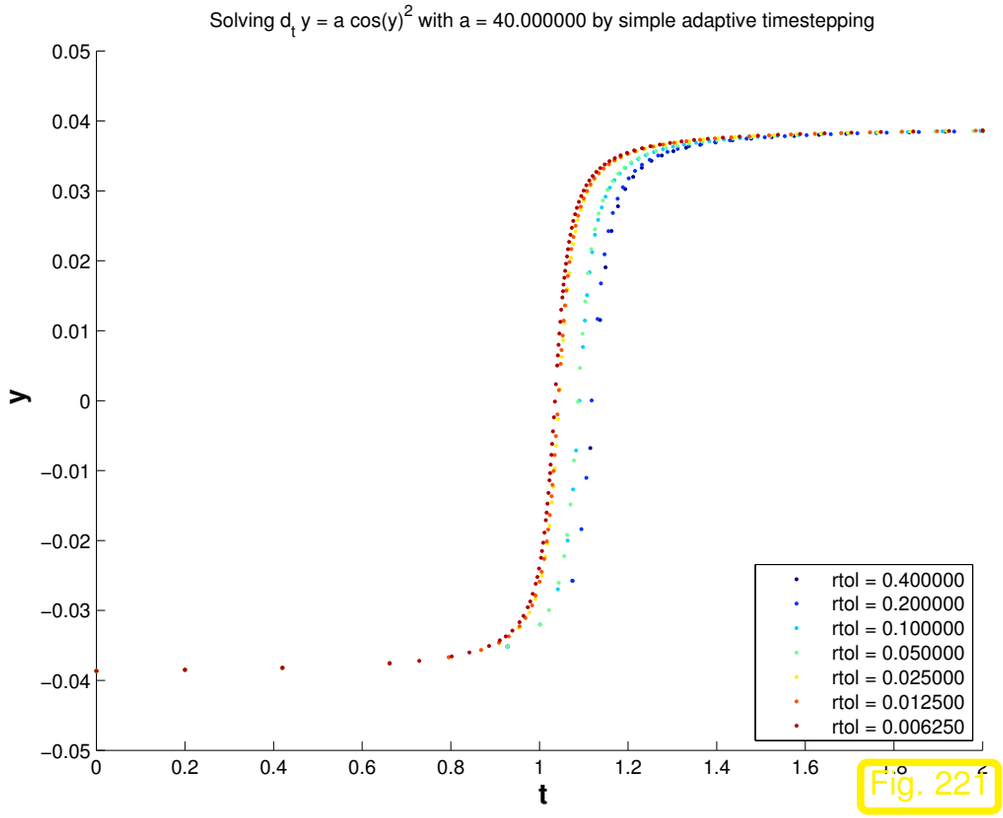


Fig. 221

Solutions $(\mathbf{y}_k)_k$ for different values of `rtol`

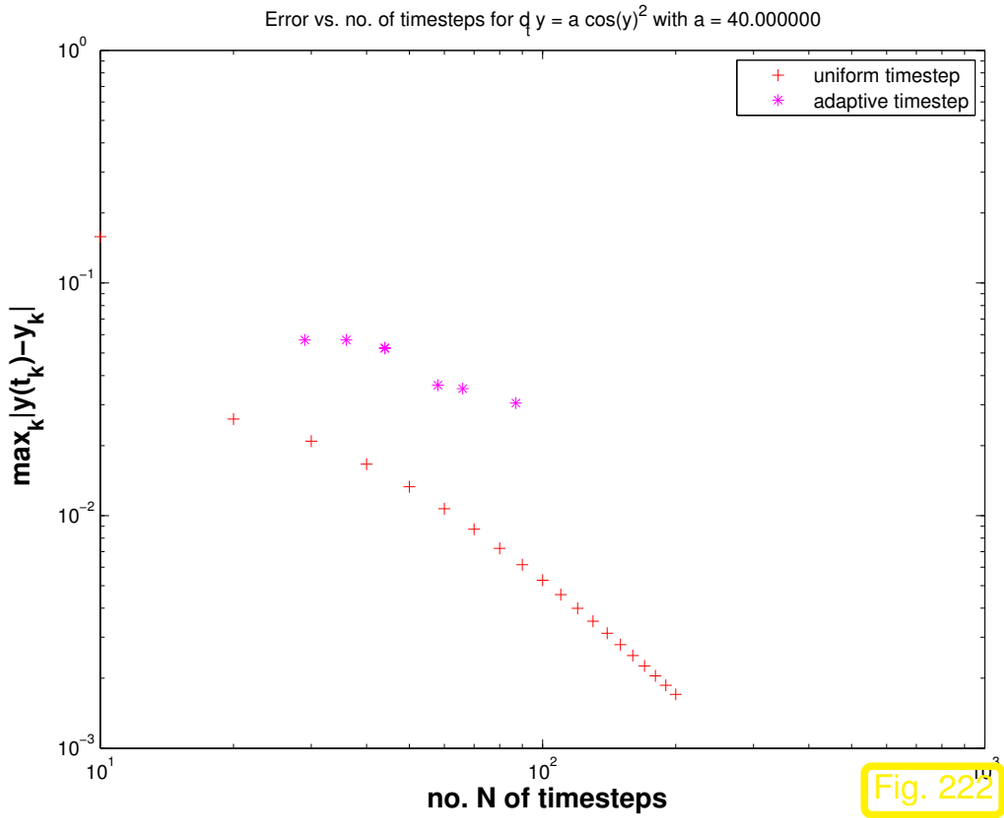


Fig. 222

Error vs. computational effort

R. Hiptmair

rev 30401, February 19, 2011

Observations:

☞ Adaptive timestepping leads to larger errors at the same computational cost as uniform timestepping.

Explanation: the position of the steep step of the solution has a sensitive dependence on an initial value $y(0) \approx -\pi/2$. Hence, small local errors in the initial timesteps will lead to large errors at around time $t \approx 1$. The stepsize control is mistaken in condoning these small one-step errors in the first few steps and, therefore, incurs huge errors later.

◇

Remark 12.5.22 (Refined local stepsize control).

The above algorithm (Code 12.5.6) is simple, but the rule for increasing/shrinking of timestep arbitrary "wastes" information contained in $\mathrm{EST}_k : \mathtt{TOL}$:

More ambitious goal **!**

When $\mathtt{EST}_k > \mathtt{TOL}$: stepsize adjustment better $h_k =$ **?**

When $\mathtt{EST}_k < \mathtt{TOL}$: stepsize prediction good $h_{k+1} =$ **?**

Assumption: At our disposal are two discrete evolutions:

- $\boldsymbol{\Psi}$ with order$(\boldsymbol{\Psi}) = p$ (➙ "low order" single step method)
- $\widetilde{\boldsymbol{\Psi}}$ with order$(\widetilde{\boldsymbol{\Psi}}) > p$ (➙ "higher order" single step method)

These are the same building blocks as for the simple adaptive strategy employed in Code 12.5.6 (, passed as arguments `Psilow`, `Psihigh` there).

Asymptotic expressions for one-step error for $h \to 0$:

$$\begin{aligned} \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &= ch^{p+1} + O(h_k^{p+2}) \,, \\ \widetilde{\boldsymbol{\Psi}}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &= O(h^{p+2}) \,, \end{aligned} \tag{12.5.23}$$

R. Hiptmair

rev 30401, February 19, 2011

with some $c > 0$.

Why h^{p+1}? Remember estimate (12.3.30) from the error analysis of the explicit Euler method: we also found $O(h_k^2)$ there for the one-step error of a single step method of order 1.

Heuristics: the timestep h is small ➥ "higher order terms" $O(h^{p+2})$ can be ignored.

$$\left.\begin{aligned}\boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &\doteq ch_k^{p+1} + O(h_k^{p+2}) \,,\\ \widetilde{\boldsymbol{\Psi}}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &\doteq O(h_k^{p+2}) \,.\end{aligned}\right\} \quad \Rightarrow \quad \boxed{\mathtt{EST}_k \doteq ch_k^{p+1}} \,. \tag{12.5.24}$$

✎ notation: $\doteq$ equality up to higher order terms in h_k

$$\mathtt{EST}_k \doteq ch_k^{p+1} \quad \Rightarrow \quad c \doteq \frac{\mathtt{EST}_k}{h_k^{p+1}} \,. \tag{12.5.25}$$

Available in algorithm, see (12.5.5)

For the sake of *accuracy* (stipulates "$\mathtt{EST}_k < \mathtt{TOL}$") & *efficiency* (favors "$>$") we aim for

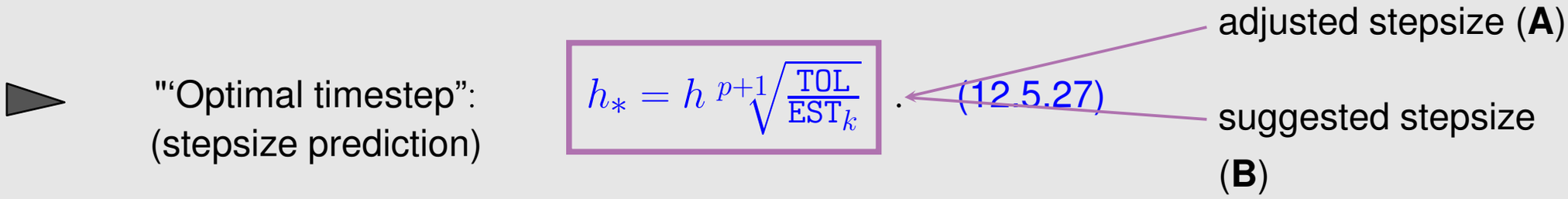
$$\mathtt{EST}_k \stackrel{!}{=} \mathtt{TOL} := \max\{\mathtt{ATOL}, \|\mathbf{y}_k\|\,\mathtt{RTOL}\} \,. \tag{12.5.26}$$

R. Hiptmair

rev 30401, February 19, 2011

What timestep h_* can actually achieve (12.5.26), if we "believe" in (12.5.24) (and, therefore, in (12.5.25))?

$$\text{(12.5.25) \& (12.5.26)} \quad \Rightarrow \quad \mathtt{TOL} = \frac{\mathtt{EST}_k}{h_k^{p+1}} h_*^{p+1} \, .$$



(**A**): In case $\mathtt{EST}_k > \mathtt{TOL}$ ➢ *repeat step* with stepsize h_*.

(**B**): If $\mathtt{EST}_k \leq \mathtt{TOL}$ ➢ use h_* as stepsize for *next step*.

Code 12.5.28: refined local stepsize control for single step methods

```
function [t,y] =
  odeintssctrl(Psilow,p,Psihigh,T,y0,h0,reltol,abstol,hmin)
t = 0; y = y0; h = h0;                    %
while ((t(end) < T)   (h > hmin)) %
  yh = Psihigh(h,y0);   %
  yH = Psilow(h,y0);    %
  est = norm(yH-yh);    %

  tol = max(reltol*norm(y(:,end)),abstol);                %
```

R. Hiptmair

rev 30401, February 19, 2011

```
9    h = h*max(0.5,min(2,(tol/est)^(1/(p+1))));                    %
10   if (est < tol)                                                  %
11     y0 = yh; y = [y,y0]; t = [t,t(end) + min(T-t(end),h)]; %
12   end
13 end
```

Comments on Code 12.5.27 (see comments on Code 12.5.6 for more explanations):

- Input arguments as for Code 12.5.6, except for $\mathtt{p} \hat{=}$ order of lower order discrete evolution.
- line 9: compute presumably better local stepsize according to (12.5.27),
- line 10: decide whether to repeat the step or advance,
- line 11: extend output arrays if current step has not been rejected.

Remark 12.5.29 (Stepsize control in MATLAB).

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` $\hat{=}$ $\mathbf{y}_k$)

△

Example 12.5.30 (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

$$\text{Newton's law:} \quad \ddot{\mathbf{y}} = F(\mathbf{y}) := -\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2} \,. \tag{12.5.31}$$

acceleration — force

Equivalent 1st-order ODE, see Rem. 12.1.15: with velocity $\mathbf{v} := \dot{\mathbf{y}}$

$$\begin{pmatrix} \dot{\mathbf{y}} \\ \dot{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ -\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2} \end{pmatrix} \,. \tag{12.5.32}$$

R. Hiptmair

rev 30401, February 19, 2011

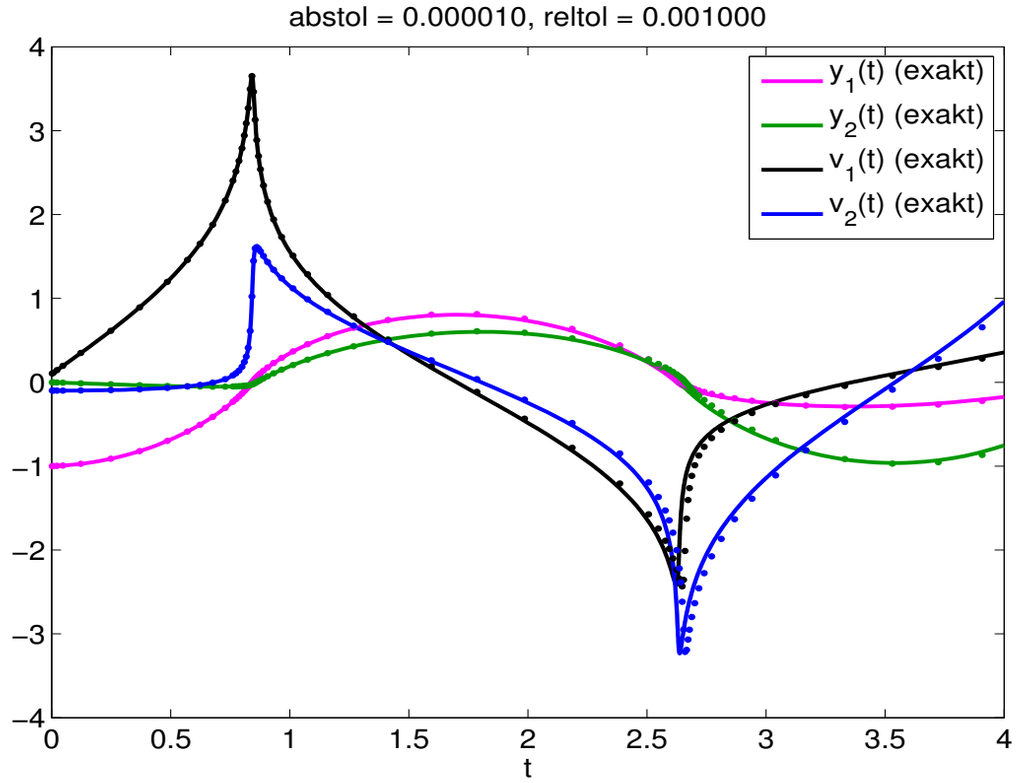
Initial values used in the experiment:

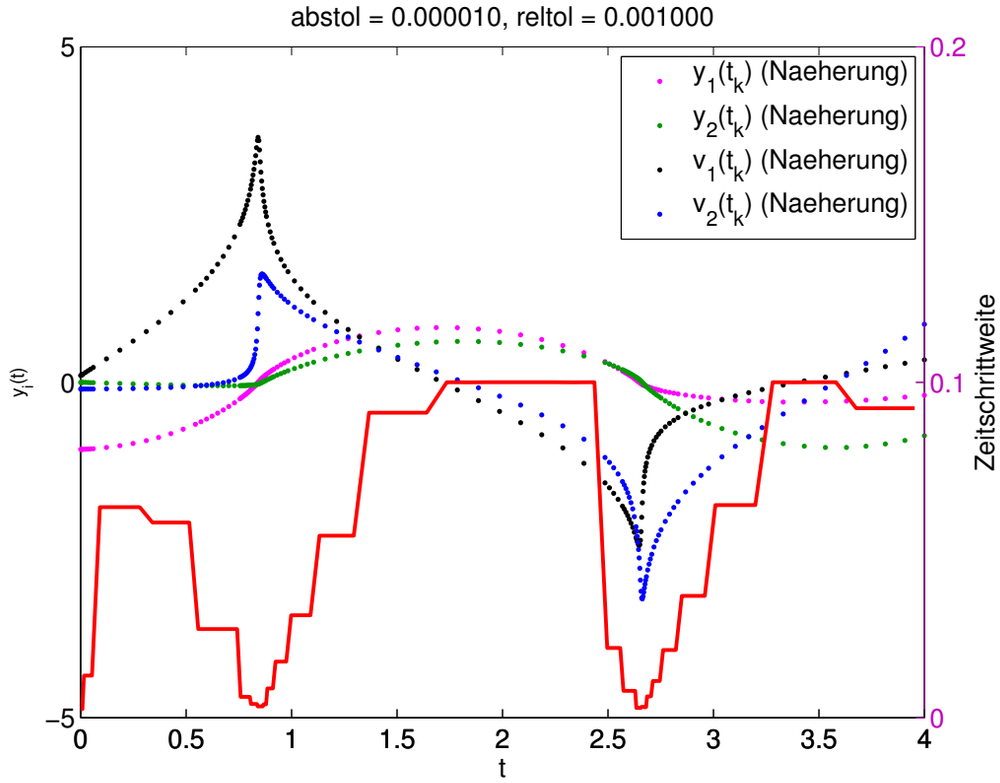
$$\mathbf{y}(0) := \begin{pmatrix} -1 \\ 0 \end{pmatrix} \quad , \quad \mathbf{v}(0) := \begin{pmatrix} 0.1 \\ -0.1 \end{pmatrix}$$

Adaptive integrator: `ode45(@(t,x) f(t,x),[0 4],[-1;0;0.1;-0.1,],options)`:

❶ `options = odeset('reltol',0.001,'abstol',1e-5);`

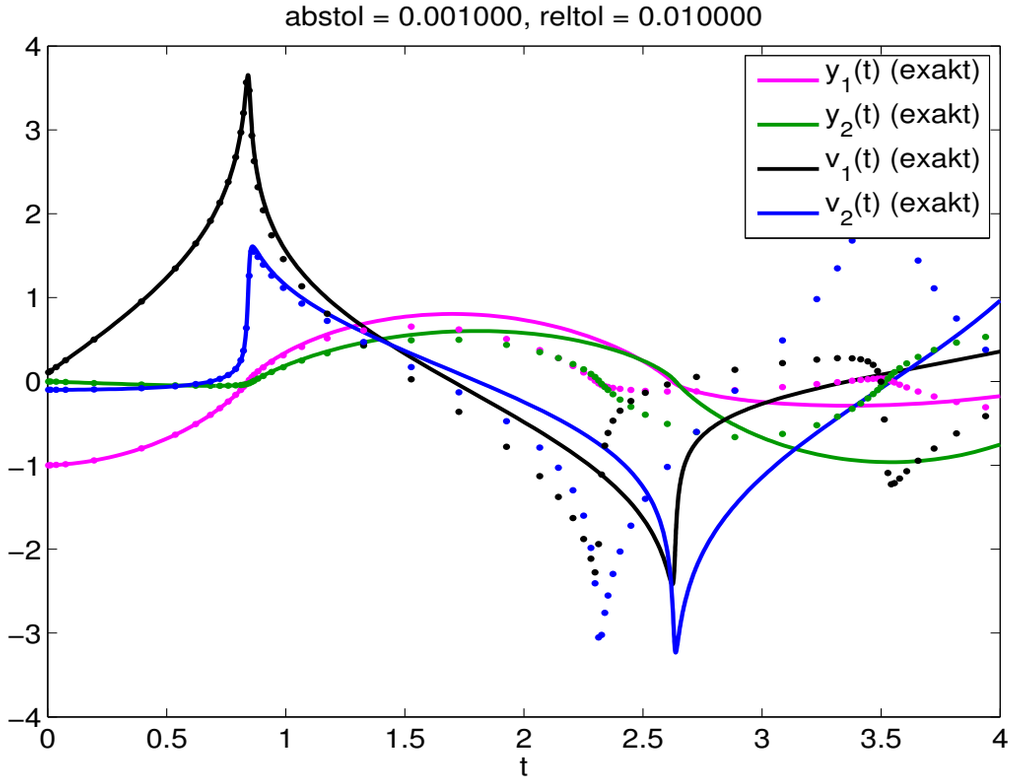
❷ `options = odeset('reltol',0.01,'abstol',1e-3);`

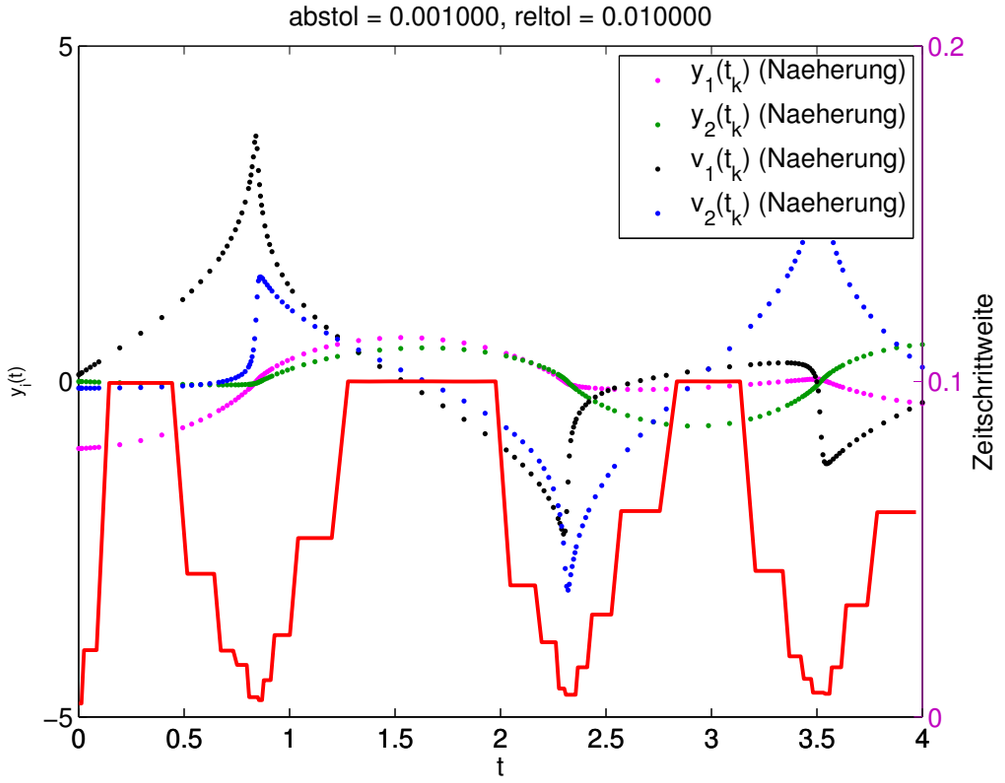




R. Hiptmair

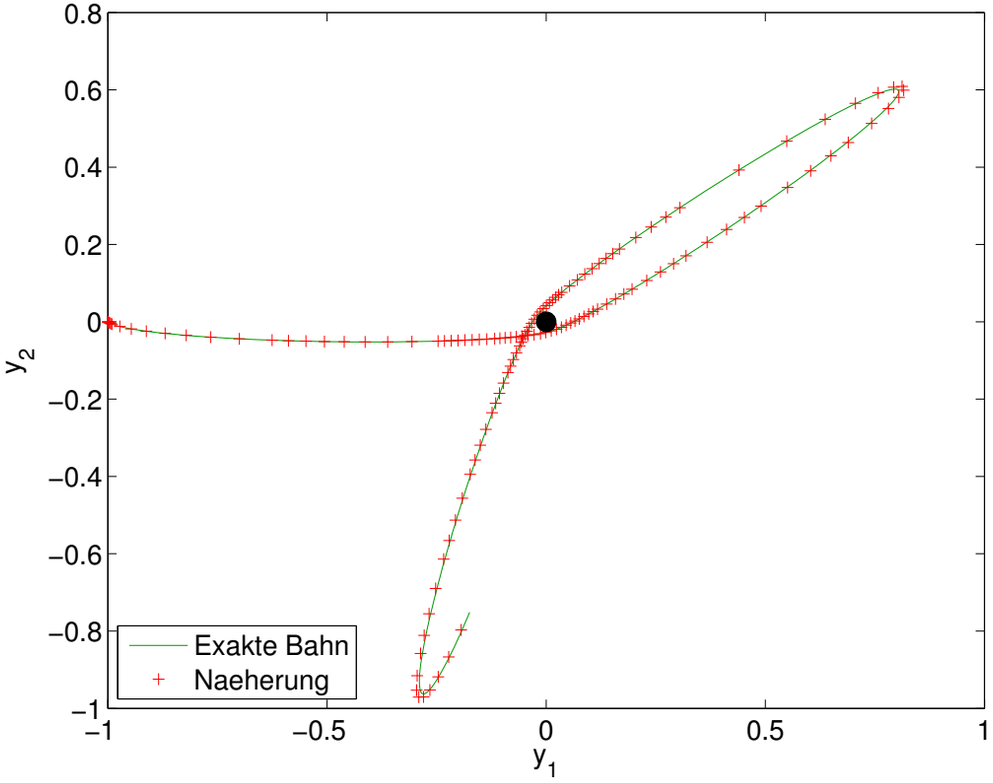
rev 30401, February 19, 2011



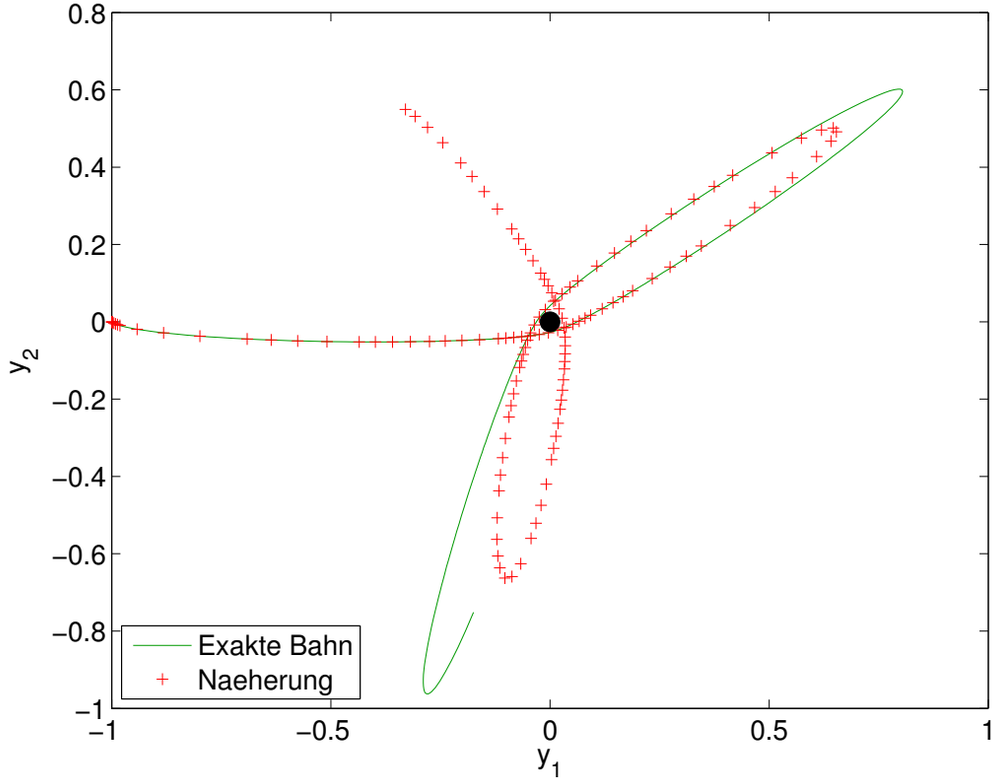


R. Hiptmair

rev 30401, February 19, 2011



reltol=0.001, abstol=1e-5



reltol=0.01, abstol=1e-3

Observations:

- ☞ Fast changes in solution components captured by adaptive approach through very small timesteps.
- ☞ Completely wrong solution, if tolerance reduced slightly.

R. Hiptmair

An inevitable consequence of time-local error estimation:

Absolute/relative tolerances do *not* allow to predict accuracy of solution!

13 Stiff Integrators [10, Sect. 11.9]

Explicit Runge-Kutta methods with stepsize control ($\rightarrow$ Sect. 12.5) seem to be able to provide approximate solutions for any IVP with good accuracy provided that tolerances are set appropriately.

Everything settled about numerical integration**?**

Example 13.0.1 (`ode45` for stiff problem).

IVP: $\dot{y} = \lambda y^2(1-y)$, $\lambda := 500$, $y(0) = \frac{1}{100}$.

```
1 fun = @(t,x) 500*x^2*(1-x);
2 options = odeset('reltol',0.1,'abstol',0.001,'stats','on');
3 [t,y] = ode45(fun,[0 1],y0,options);
```

The option `stats = 'on'` makes MATLAB print statistics about the run of the integrators.

```
186 successful steps
55 failed attempts
1447 function evaluations
```

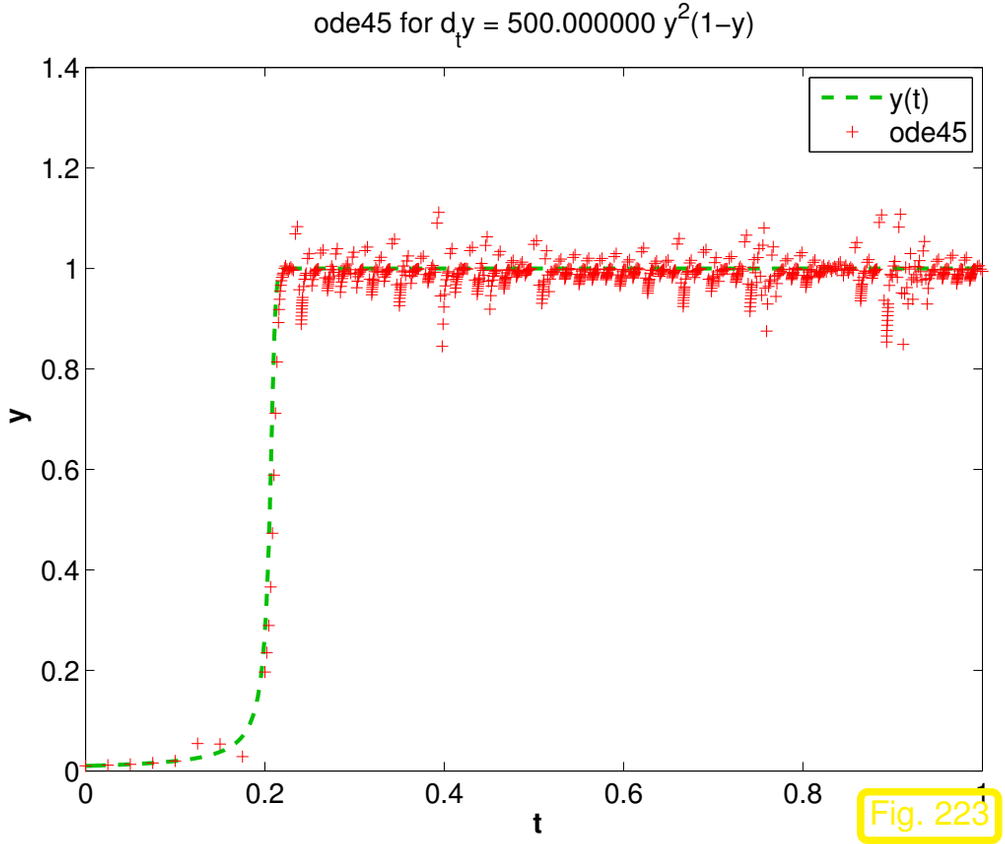


Fig. 223

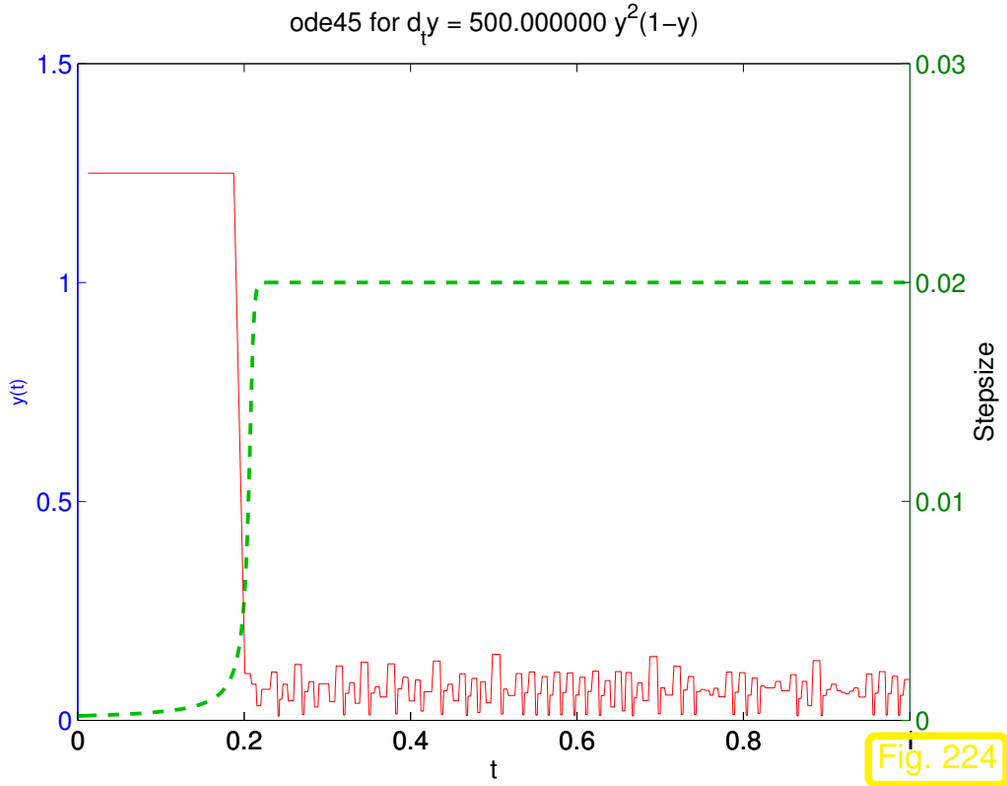


Fig. 224

Stepsize control of `ode45` running amok!

? The solution is virtually constant from $t > 0.2$ and, nevertheless, the integrator uses tiny timesteps until the end of the integration interval.

◇

13.1 Model problem analysis [29, Ch. 77]

Example 13.1.1 (Blow-up of explicit Euler method).

As in part II of Ex. 12.3.3:

- 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 $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.

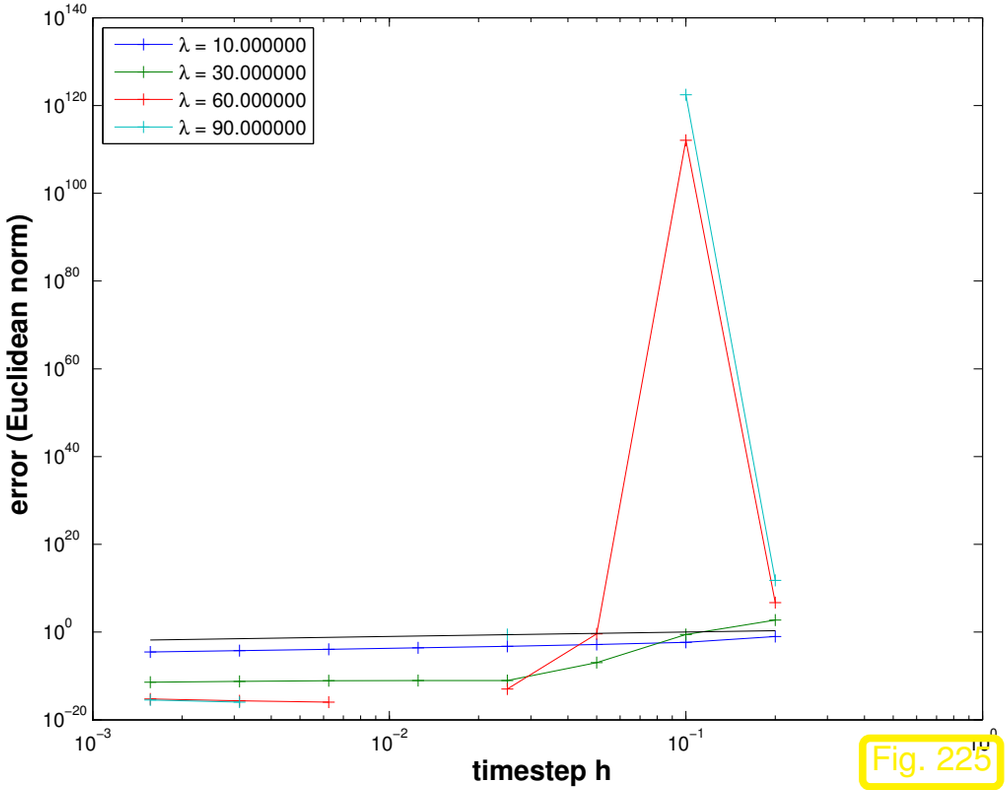


Fig. 225

λ large: blow-up of y_k for large timestep h

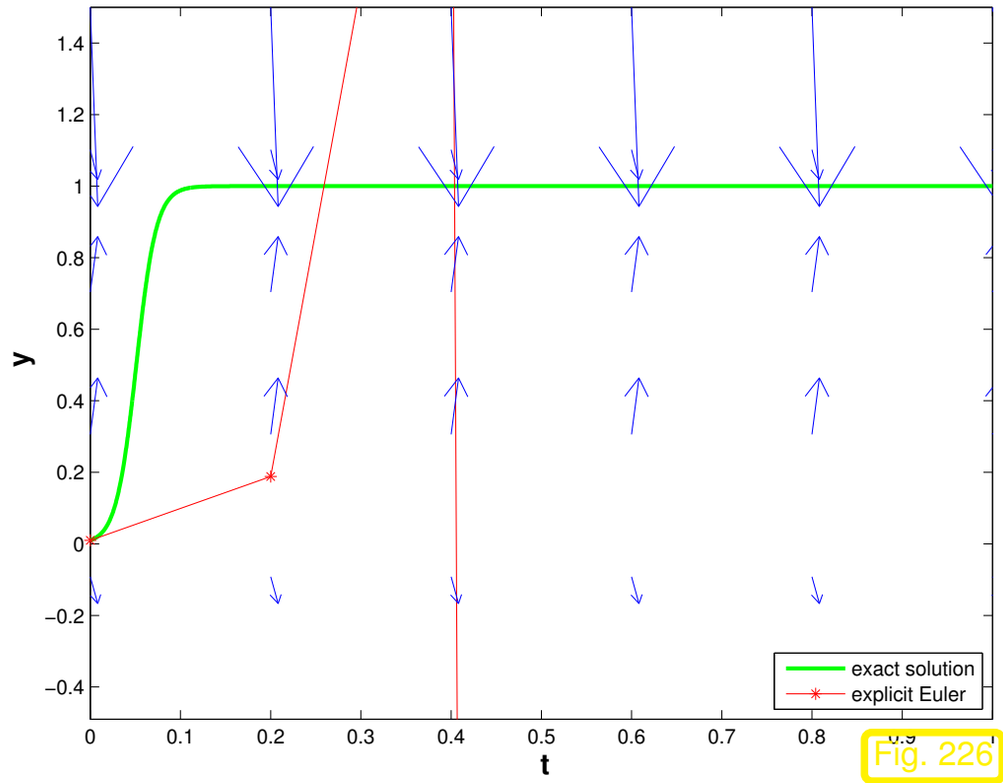


Fig. 226

$\lambda = 90$: — $\hat{=}$ $y(t)$, — $\hat{=}$ Euler polygon

Explanation: y_k way miss the stationary point $y = 1$ (overshooting).

This leads to a sequence $(y_k)_k$ with exponentially increasing oscillations.

Deeper analysis:

For $y \approx 1$: $f(y) \approx \lambda(1-y)$ ➢ If $y(t_0) \approx 1$, then the solution of the IVP will behave like the solution of $\dot{y} = \lambda(1-y)$, which is a linear ODE. Similary, $z(t) := 1 - y(t)$ will behave like the solution of the "decay equation" $\dot{z} = -\lambda z$.

◇

Motivated by the considerations in Ex. 13.1.1 we study the explicit Euler method (12.2.4) for the

$$\text{linear model problem: } \dot{y} = \lambda y \;, \quad y(0) = y_0 \;, \quad \text{with} \quad \lambda \ll 0 \;, \tag{13.1.4}$$

and *exponentially decaying* exact solution

$$y(t) = y_0 \exp(\lambda t) \to 0 \quad \text{for } t \to \infty \;.$$

R. Hiptmair

rev 30401, December 23, 2010

Recursion of explicit Euler method for (13.1.4):

$$\text{(12.2.4) for } f(y)=\lambda y\text{:} \qquad y_{k+1} = y_k(1+\lambda h)\ . \tag{13.1.5}$$

$$\blacktriangleright \quad y_k = y_0(1+\lambda h)^k \quad \Rightarrow \quad |y_k| \to \begin{cases} 0 & \text{, if } \lambda h > -2 \quad \text{(qualitatively correct)}\ , \\ \infty & \text{, if } \lambda h < -2 \quad \text{(qualitatively wrong)}\ . \end{cases}$$

Timestep constraint: only if $|\lambda| h < 2$ we obtain decaying solution by explicit Euler method!

Could it be that the timestep control is desperately trying to enforce the qualitatively correct behavior of the numerical solution in Ex. 13.1.1? Let us examine how the simple stepsize control of Code 12.5.6 fares for model problem (13.1.4):

R. Hiptmair

rev 30401, December 23, 2010

Example 13.1.6 (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

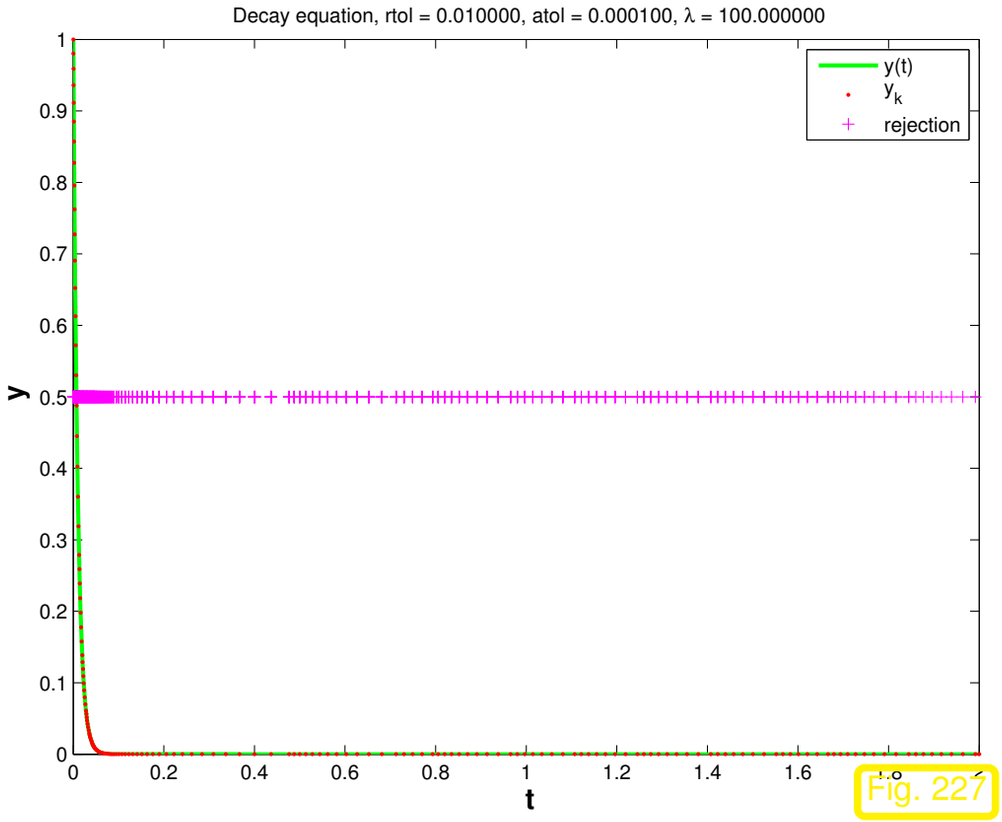


Fig. 227

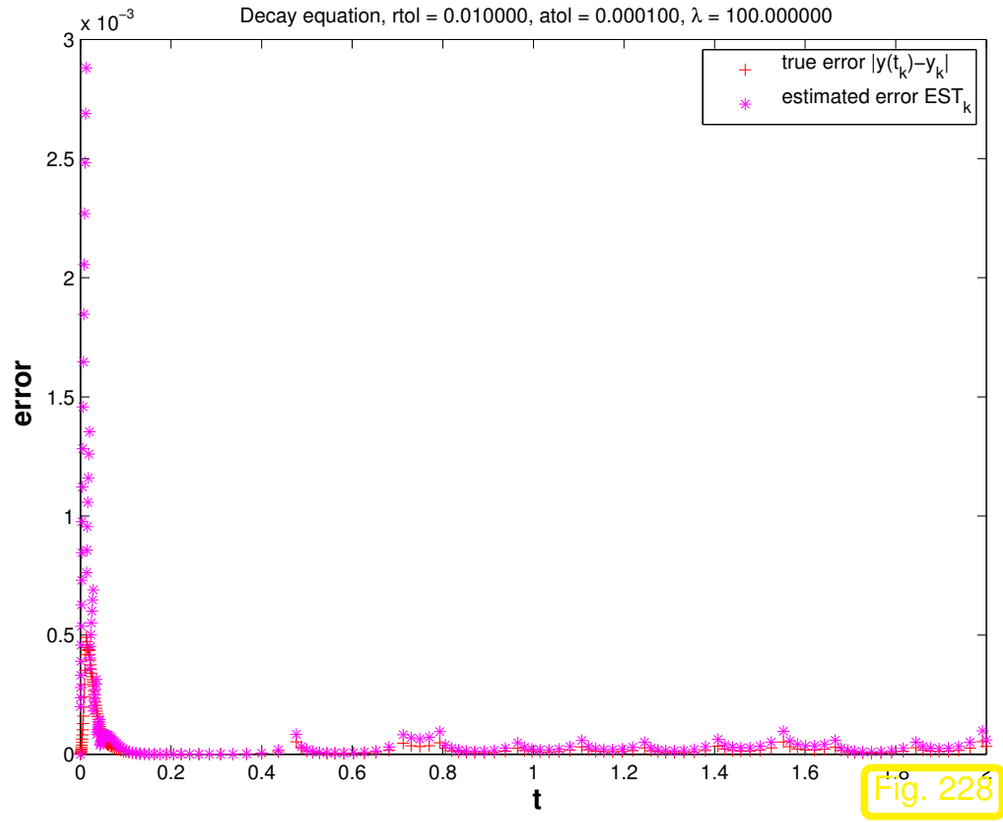


Fig. 228

R. Hiptmair

rev 30401, December 23, 2010

Observation: in fact, stepsize control enforces small timesteps even if $\mathbf{y}(t) \approx 0$ and persistently triggers rejections of timesteps. This is necessary to prevent overshooting in the Euler method, which contributes to the estimate of the one-step error.

Is this a particular "flaw" of the explicit Euler method? Let us study the behavior of another simple explicit Runge-Kutta method applied to the linear model problem.

Example 13.1.7 (Explicit trapzoidal rule for decay equation). → [10, Ex. 11.29]

Recall recursion for explicit trapezoidal rule:

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0) \;, \quad \mathbf{k}_2 = \mathbf{f}(t_0 + h, \mathbf{y}_0 + h\mathbf{k}_1) \;, \quad \mathbf{y}_1 = \mathbf{y}_0 + \tfrac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2) \;. \tag{12.4.6}$$

Apply this to the model problem (13.1.4), that is $\mathbf{f}(y) = f(y) = \lambda y$, $\lambda < 0$:

$$k_1 = \lambda y_0 \;, \quad k_2 = \lambda(y_0 + hk_1) \quad \Rightarrow \quad y_1 = \underbrace{(1 + \lambda h + \tfrac{1}{2}(\lambda h)^2)}_{=:S(h\lambda)} y_0 \;. \tag{13.1.10}$$

sequence generated by explicit trapezoidal rule:

$$y_k = S(h\lambda)^k y_0 \;, \quad k = 0, \ldots, N \;. \tag{13.1.11}$$

R. Hiptmair

rev 30401, December 23, 2010

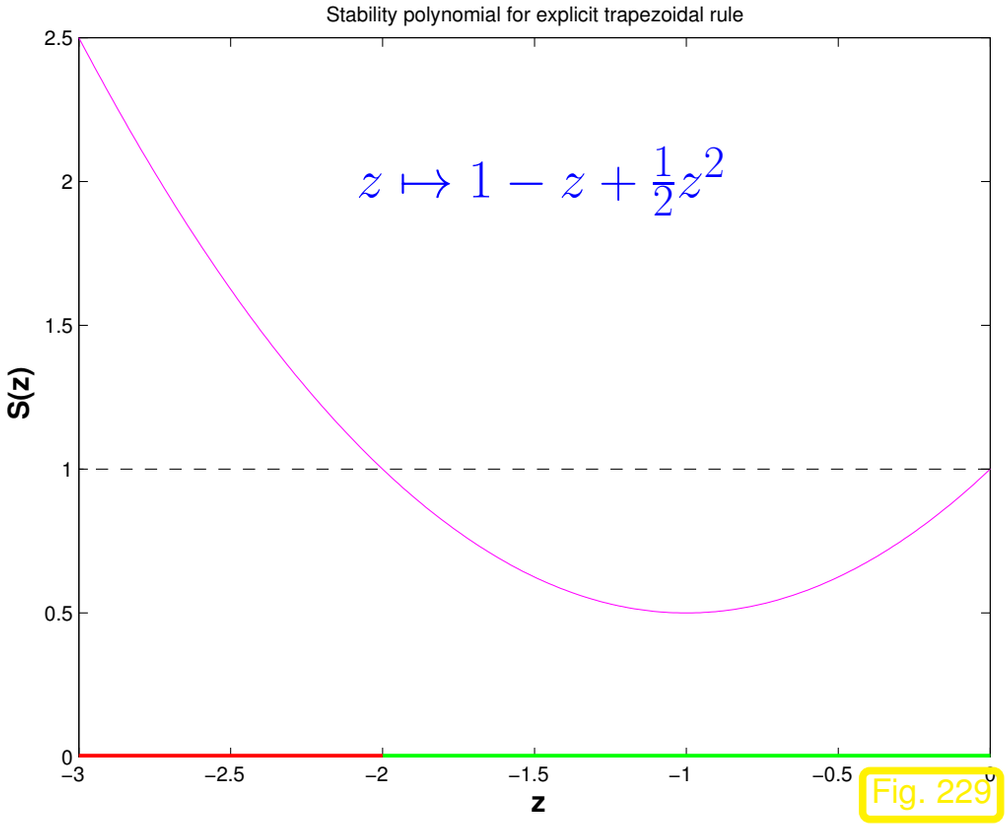


Fig. 229

$$|S(h\lambda)| < 1 \quad \Leftrightarrow \quad -2 < h\lambda < 0 \,.$$

Qualitatively correct decay behavior of $(y_k)_k$ only under timestep constraint

$$h \leq |2/\lambda| \,. \tag{13.1.13}$$

◇

R. Hiptmair

rev 30401, December 23, 2010

Model problem analysis for general explicit Runge-Kutta method (→ Def. 12.4.9): apply Runge-Kutta method encoded by Butcher scheme $\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$ to (13.1.4)

$$k_i = \lambda(y_0 + h\sum_{j=1}^{i-1} a_{ij}k_j) \,, \quad i = 1, \ldots, s \quad \Rightarrow \quad \begin{pmatrix} \mathbf{I} - z\mathfrak{A} & 0 \\ -z\mathbf{b}^T & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{k} \\ y_1 \end{pmatrix} = y_0 \begin{pmatrix} \mathbf{1} \\ 1 \end{pmatrix} \,, \tag{13.1.16}$$

where $\boldsymbol{k} \in \mathbb{R}^s \hat{=}$ denotes the vector $(k_1, \ldots, k_s)^T/\lambda$ of increments, and $z := \lambda h$.

$$\blacktriangleright \quad y_1 = S(z)y_0 \quad \text{with} \quad S(z) := 1 + z\mathbf{b}^T(\mathbf{I} - z\mathfrak{A})^{-1}\mathbf{1} = \det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T)\,. \tag{13.1.17}$$

The first formula for $S(z)$ immediately follows from (13.1.16), the second is a consequence of Cramer's rule.

Thus we have proved the following theorem.

Theorem 13.1.18 (Stability function of explicit Runge-Kutta methods)**.** $\rightarrow$ *[29, Thm. 77.2]*

The discrete evolution Ψ_λ^h of an explicit s-stage Runge-Kutta single step method ($\rightarrow$ Def. 12.4.9) with Butcher scheme $\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$ *(see* (12.4.10)*) for the ODE $\dot{y} = \lambda y$ is a multiplication operator according to*

$$\Psi_\lambda^h = \underbrace{1 + z\mathbf{b}^T(\mathbf{I} - z\mathfrak{A})^{-1}\mathbf{1}}_{\text{stability function } S(z)} = \det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T)\,, \quad z := \lambda h\,, \quad \mathbf{1} = (1, \ldots, 1)^T \in \mathbb{R}^s\,.$$

R. Hiptmair

rev 30401, December 23, 2010

Thm. 13.1.18 $\Rightarrow$ $S \in \mathcal{P}_s$ (polynomial stability function)

Remember from Ex. 13.1.7: for sequence $(|y_k|)_{k=0}^{\infty}$ produced by explicit Runge-Kutta method applied to IVP (13.1.4) holds $y_k = S(\lambda h)^k y_0$.

$$\boxed{(|y_k|)_{k=0}^{\infty} \text{ non-increasing} \Leftrightarrow |S(\lambda h)| \leq 1} \,, \tag{13.1.20}$$

$$(|y_k|)_{k=0}^{\infty} \text{ exponentially increasing} \Leftrightarrow |S(\lambda h)| > 1 \,.$$

On the other hand: $\forall S \in \mathcal{P}_s$: $\lim_{|z|\to\infty} |S(z)| = \infty$

timestep constraint: In order to avoid exponentially increasing (qualitatively wrong for $\lambda < 0$) sequences $(y_k)_{k=0}^{\infty}$ we must have $|\lambda h|$ *sufficiently small*.

Small timesteps may have to be used for stability reasons,
though accuracy may not require them!

Inefficient numerical integration

R. Hiptmair

rev 30401, December 23, 2010

Remark 13.1.21 (Stepsize control detects instability).

Always look at the bright side of life:

Ex. 13.0.1, 13.1.6: Stepsize control guarantees acceptable solutions, with a hefty price tag however. △

13.2 Stiff problems

Objection: The IVP (13.1.4) may be an oddity rather than a model problem: the weakness of explicit Runge-Kutta methods discussed in the previous section may be just a peculiar response to an unusual situation.

This section will reveal that the behavior observed in Ex. 13.0.1 and Ex. 13.1.1 is typical for a large class of problems and that the model problem (13.1.4) really represents a “generic case”.

Example 13.2.1 (Transient simulation of RLC-circuit).

Circuit from Ex. 12.1.8 ▷

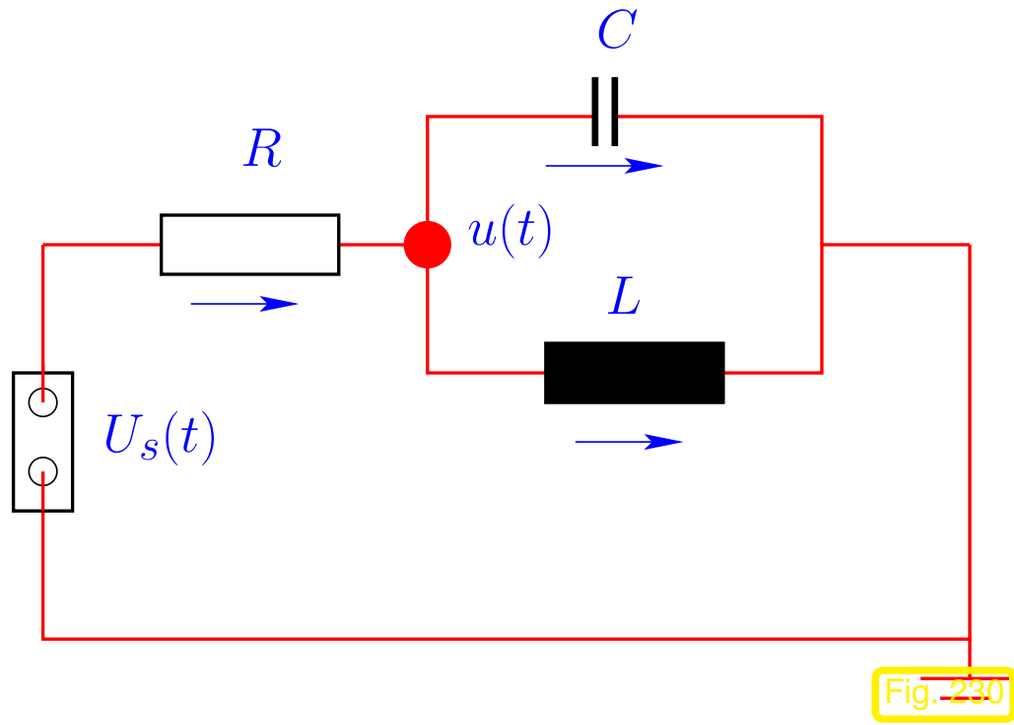


Fig. 230

$$\ddot{u} + \alpha\dot{u} + \beta u = g(t) \ ,$$

$\alpha := (RC)^{-1}$, $\beta = (LC)^{-1}$, $g(t) = \alpha\dot{U}_s$.

Transformation to linear 1st-order ODE, see Rem. 12.1.15, $v := \dot{u}$

$$\underbrace{\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix}}_{=:\dot{\mathbf{y}}} = \underbrace{\begin{pmatrix} 0 & 1 \\ -\beta & -\alpha \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} - \begin{pmatrix} 0 \\ g(t) \end{pmatrix}}_{=:\mathbf{f}(t,\mathbf{y})} \ .$$

R. Hiptmair

rev 30401, December 23, 2010

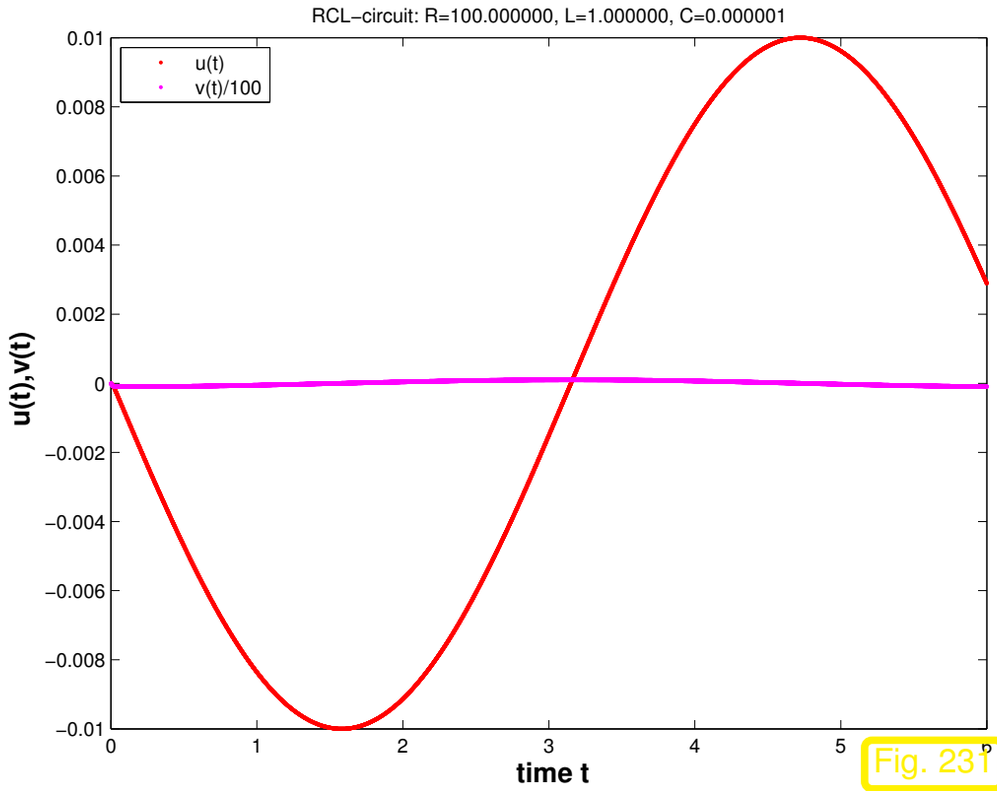


Fig. 231

$R = 100\Omega$, $L = 1\mathrm{H}$, $C = 1\mu\mathrm{F}$, $U_s(t) = 1\mathrm{V}\sin(t)$,
$u(0) = v(0) = 0$ (“switch on”)

`ode45` statistics:

```
17897 successful steps
1090 failed attempts
113923 function evaluations
```

Maybe the time-dependent right hand side due to the time-harmonic excitation severly affects `ode45`? Let us try a constant exciting voltage:

R. Hiptmair

rev 30401, December 23, 2010

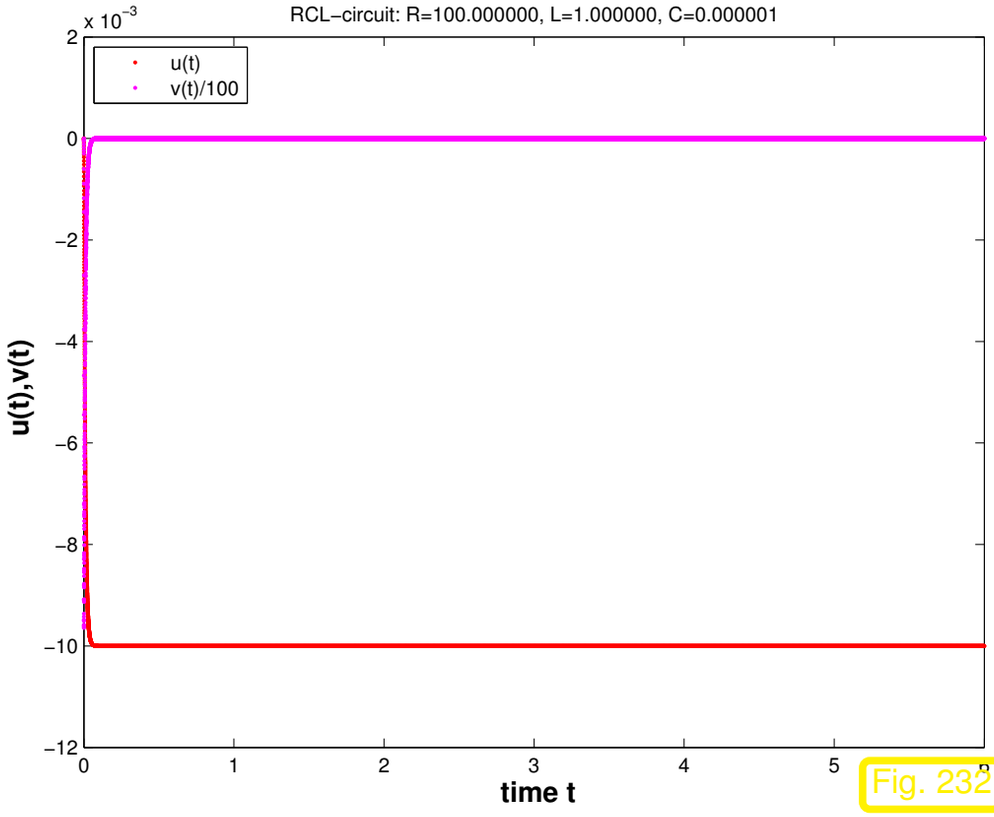


Fig. 232

$R = 100\Omega$, $L = 1\mathrm{H}$, $C = 1\mu\mathrm{F}$, $U_s(t) = 1\mathrm{V}$, $u(0) = v(0) = 0$ (“switch on”)

`ode45` statistics:

```
17901 successful steps
1210 failed attempts
114667 function evaluations
```

Code 13.2.2: simulation of linear RLC circuit using `ode45`

```
1 function stiffcircuit(R,L,C,Us,tspan,filename)
2 % Transient simulation of simple linear circuit of Ex. refex:stiffcircuit
3 % R,L,C: paramters for circuits elements (compatible units required)
4 % Us:  exciting time-dependent voltage U_s = U_s(t), function handle
5 % zero initial values
6
7 % Coefficient for 2nd-order ODE ü + αu̇ + β = g(t)
```

R. Hiptmair

rev 30401, December 23, 2010

```
8  alpha = 1/(R*C); beta = 1/(C*L);
9  % Conversion to 1st-order ODE y = My + (0; g(t)). Set up right hand side function.
10 M = [0 , 1; -beta , -alpha]; rhs = @(t,y) (M*y - [ 0 ;
     alpha*Us(t)]);
11 % Set tolerances for MATLAB integrator, see Rem. 12.5.29
12 options = odeset('reltol',0.1,'abstol',0.001,'stats','on');
13 y0 = [0;0]; [t,y] = ode45(rhs,tspan,y0,options);
14
15 % Plot the solution components
16 figure('name','Transient circuit simulation');
17 plot(t,y(:,1),'r.',t,y(:,2)/100,'m.');
18 xlabel('{\bf time t}','fontsize',14);
19 ylabel('{\bf u(t),v(t)}','fontsize',14);
20 title(sprintf('RCL-circuit: R=%f, L=%f, C=%f',R,L,C));
21 legend('u(t)','v(t)/100','location','northwest');
22
23 print('-depsc2',sprintf('../PICTURES/%s.eps',filename));
```

Observation: stepsize control of `ode45` (→ Sect. 12.5) enforces extremely small timesteps though solution almost constant except at $t = 0$.

R. Hiptmair

rev 30401, December 23, 2010

Motivated by Ex. 13.2.1 we examine linear homogeneous IVP of the form

$$\dot{\mathbf{y}} = \underbrace{\begin{pmatrix} 0 & 1 \\ -\beta & -\alpha \end{pmatrix}}_{=:\mathbf{M}} \mathbf{y} \quad , \quad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{R}^2 \, . \tag{13.2.8}$$

In Ex .13.2.1: $\beta \gg \frac{1}{4}\alpha^2 \gg 1$.

[52, Sect. 5.6]: general solution of $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, $\mathbf{M} \in \mathbb{R}^{2,2}$, by diagonalization of $\mathbf{M}$ (if possible):

$$\mathbf{M}\mathbf{V} = \mathbf{M}\,(\mathbf{v}_1, \mathbf{v}_2) = (\mathbf{v}_1, \mathbf{v}_2) \begin{pmatrix} \lambda_1 & \\ & \lambda_2 \end{pmatrix} \, . \tag{13.2.9}$$

► $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^2 \setminus \{0\} \hat{=}$ eigenvectors of $\mathbf{M}$, $\lambda_1, \lambda_2 \hat{=}$ eigenvalues of $\mathbf{M}$, see Def. 6.1.1.

Idea: transform $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$ into *decoupled* scalar linear ODEs!

$$\dot{\mathbf{y}} = \mathbf{M}\mathbf{y} \quad \Leftrightarrow \quad \mathbf{V}^{-1}\dot{\mathbf{y}} = \mathbf{V}^{-1}\mathbf{M}\mathbf{V}(\mathbf{V}^{-1}\mathbf{y}) \quad \overset{\mathbf{z}(t):=\mathbf{V}^{-1}\mathbf{y}(t)}{\Leftrightarrow} \quad \dot{\mathbf{z}} = \begin{pmatrix} \lambda_1 & \\ & \lambda_2 \end{pmatrix} \mathbf{z} \, . \tag{13.2.10}$$

R. Hiptmair

rev 30401, December 23, 2010

This yields the general solution of the ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$

$$\mathbf{y}(t) = A\mathbf{v}_1 \exp(\lambda_1 t) + B\mathbf{v}_2 \exp(\lambda_2 t) , \quad A, B \in \mathbb{R} . \tag{13.2.11}$$

Note: $t \mapsto \exp(\lambda_i t)$ is general solution of the ODE $\dot{z}_i = \lambda_i z_i$.

Consider discrete evolution of explicit Euler method (12.2.4) for ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$

$$\boldsymbol{\Psi}^h \mathbf{y} = \mathbf{y} + h\mathbf{M}\mathbf{y} \quad \leftrightarrow \quad \mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{M}\mathbf{y}_k .$$

Perform the same transformation as above on the discrete evolution:

$$\mathbf{V}^{-1}\mathbf{y}_{k+1} = \mathbf{V}^{-1}\mathbf{y}_k + h\mathbf{V}^{-1}\mathbf{M}\mathbf{V}(\mathbf{V}^{-1}\mathbf{y}_k) \quad \overset{\mathbf{z}_k := \mathbf{V}^{-1}\mathbf{y}_k}{\Leftrightarrow} \quad \underbrace{(\mathbf{z}_{k+1})_i = (\mathbf{z}_k)_i + h\lambda_i (\mathbf{z}_k)_i}_{\hat{=} \text{ explicit Euler step for } \dot{z}_i = \lambda_i z_i} .$$

(13.2.12)

Crucial insight:

> The explicit Euler method generates uniformly bounded solution sequences $(\mathbf{y}_k)_{k=0}^{\infty}$ for $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$ with diagonalizable matrix $\mathbf{M} \in \mathbb{R}^{d,d}$ with eigenvalues $\lambda_1, \ldots, \lambda_d$, if and only if it generates uniformly bounded sequences for **all** the scalar ODEs $\dot{z} = \lambda_i \dot{z}$, $i = 1, \ldots, d$.

An analoguous statement is true for all Runge-Kutta methods!

(This is revealed by simple algebraic manipulations of the increment equations.)

So far we conducted the model problem analysis under the premises $\lambda < 0$.

However: in Ex. 13.2.1 we have $\lambda_{1/2} = -\frac{1}{2}\alpha \pm i\sqrt{\beta - \frac{1}{4}\alpha^2}$ (complex eigenvalues!). How will explicit Euler/explicit RK-methods respond to them?

Example 13.2.13 (Explicit Euler method for damped oscillations).

Consider linear model IVP (13.1.4) for $\lambda \in \mathbb{C}$:

$$\operatorname{Re}\lambda < 0 \quad \Rightarrow \quad \text{exponentially decaying solution} \quad y(t) = y_0 \exp(\lambda t) \ ,$$

because $|\exp(\lambda t)| = \exp(\operatorname{Re}\lambda t)$.

Model problem analysis ($\rightarrow$ Ex. 13.1.1, Ex. 13.1.7) for explicit Euler method and $\lambda \in \mathbb{C}$:

Sequence generated by explicit Euler method (12.2.4) for model problem (13.1.4):

$$y_{k+1} = y_k(1 + h\lambda) \; . \tag{13.1.5}$$

$$\lim_{k\to\infty} y_k = 0 \quad \Leftrightarrow \quad |1 + h\lambda| < 1 \; .$$

timestep constraint to get decaying (discrete) solution !

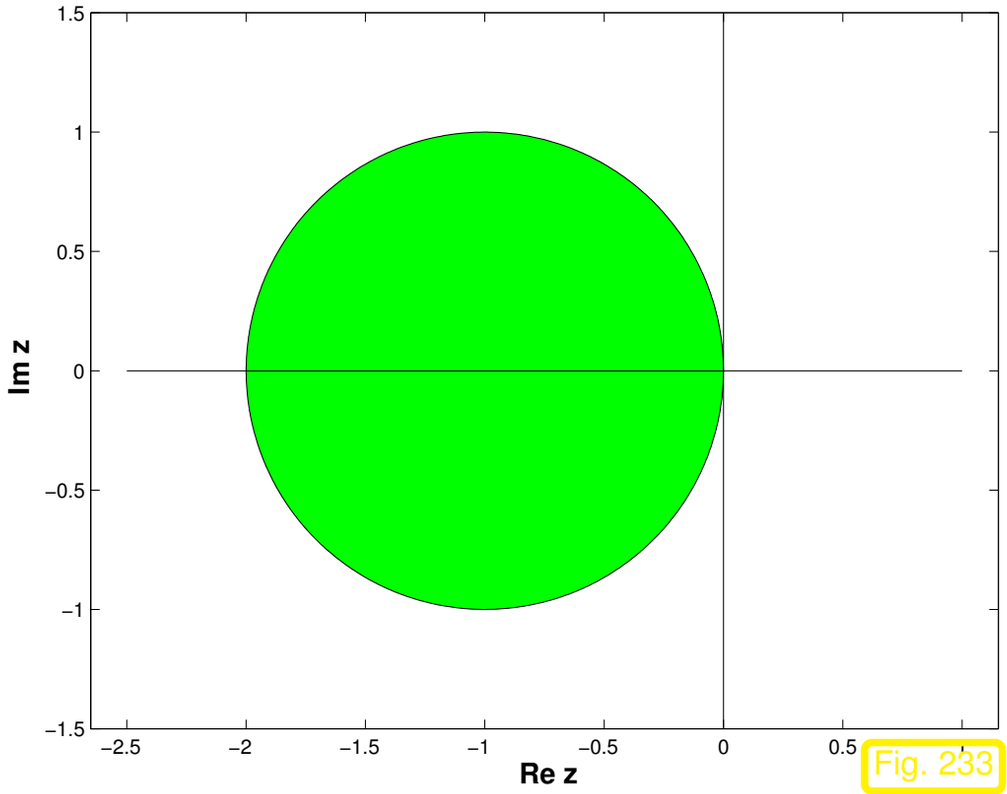


Fig. 233

◁ $\{z \in \mathbb{C}: \; |1 + z| < 1\}$

Now we can conjecture what happens in Ex. 13.2.1: the eigenvalues $\lambda_{1/2} = -\frac{1}{2}\alpha \pm i\sqrt{\beta - \frac{1}{4}\alpha^2}$ of $\mathbf{M}$ have a very large (in modulus) negative real part. Since `ode45` can be expected to behave as if it integrates $\dot{z} = \lambda_2 z$, it faces a severe timestep constraint, if exponential blow-up is to be avoided, see Ex. 13.1.1. Thus stepsize control must resort to tiny timesteps.

◇

Can we predict this kind of difficulty ?

Example 13.2.14 (Kinetics of chemical reactions). → [29, Ch. 62]

reaction:
$$\underbrace{\mathrm{A} + \mathrm{B} \underset{k_1}{\overset{k_2}{\rightleftharpoons}} \mathrm{C}}_{\text{fast reaction}} \quad , \quad \underbrace{\mathrm{A} + \mathrm{C} \underset{k_3}{\overset{k_4}{\rightleftharpoons}} \mathrm{D}}_{\text{slow reaction}} \tag{13.2.15}$$

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 $\mathbf{y}(t) = (c_A(t), c_B(t), c_C(t), c_D(t))^T$

$$\dot{\mathbf{y}} := \frac{d}{dt}\begin{pmatrix} c_A \\ c_B \\ c_C \\ c_D \end{pmatrix} = \mathbf{f}(\mathbf{y}) := \begin{pmatrix} -k_1 c_A c_B + k_2 c_C - k_3 c_A c_C + k_4 c_D \\ -k_1 c_A c_B + k_2 c_C \\ k_1 c_A c_B - k_2 c_C - k_3 c_A c_C + k_4 c_D \\ k_3 c_A c_C - k_4 c_D \end{pmatrix} . \tag{13.2.16}$$

MATLAB computation: $t_0 = 0$, $T = 1$, $k_1 = 10^4$, $k_2 = 10^3$, $k_3 = 10$, $k_4 = 1$

Code 13.2.18: Simulation of "stiff" chemical reaction

```
function chemstiff
% Simulation of kinetics of coupled chemical reactions with vastly different
  reaction
% rates, see (13.2.16) for the ODE model.
% reaction rates k_1, k_2, k_3, k_4, k_1, k_2 >> k_3, k_4.
k1 = 1E4; k2 = 1E3; k3 = 10; k4 = 1;
% definition of right hand side function for ODE solver
fun = @(t,y) ([-k1*y(1)*y(2) + k2*y(3) - k3*y(1)*y(3) +  k4*y(4);
               -k1*y(1)*y(2) + k2*y(3);
               k1*y(1)*y(2) - k2*y(3) - k3*y(1)*y(3) + k4*y(4);
               k3*y(1)*y(3) - k4*y(4)]);
tspan = [0 1]; % Integration time interval
L = tspan(2)-tspan(1); % Duration of simulation
y0 = [1;1;10;0]; % Initial value y_0

```

R. Hiptmair

```
% compute "exact" solution, using ode113 with tight error tolerances
options = odeset('reltol',10*eps,'abstol',eps,'stats','on');
% get the 'exact' solution using ode113
[tex,yex] = ode113(fun,[0 1],y0,options);
% plot 'exact' results
figure; h = plot(tex,yex(:,1),'c--',tex,yex(:,3),'m--');
set(h(1),'linewidth',2); set(h(2),'linewidth',2);
leg{1} = 'c_A(t)'; leg{2} = 'c_C(t)'; hold on;

% Compute solution with ode45 and moderate tolerances
options = odeset('reltol',0.1,'abstol',0.001,'stats','on');
[t,y] = ode45(fun,[0 1],y0,options);
% plot the solution
plot(t,y(:,1),'b.',t,y(:,3),'r.');
leg{3} = 'c_{A,k}, ode45';leg{4} = 'c_{C,k}, ode45';
set(gca,'fontsize',14);
xlabel('{\bf t}');
ylabel('{\bf concentrations}');
title('Chemical reaction: concentrations');
legend(leg);
print -depsc2 '../PICTURES/chemstiff.eps';

% Plot stepsizes together with "exact" solution
figure('Name','Stepsizes');
[ax,h1,h2] = plotyy(tex,yex(:,3),0.5*(t(1:end-1)+t(2:end)),diff(t));
% set legend, labels, axes, ...
set(ax(1),'fontsize',14); set(ax(2),'fontsize',14);
set(ax(2),'xcolor',[ 0 0 0 ]);
xlabel('t');
```


R. Hiptmair

rev 30401, December 23, 2010

```
44 set(h1,'linestyle','--','color',[0 0.75 0],'linewidth',2);
45 set(h2,'linestyle','-','color','r');
46 set(get(ax(1),'ylabel'),'string','c_C(t)');
47 set(get(ax(2),'ylabel'),'string','timestep','color',[ 0 0 0],'fontsize',14);
48 title('Chemical reaction: stepsize');
49 print -depsc2 'chemstiffss.eps';
```

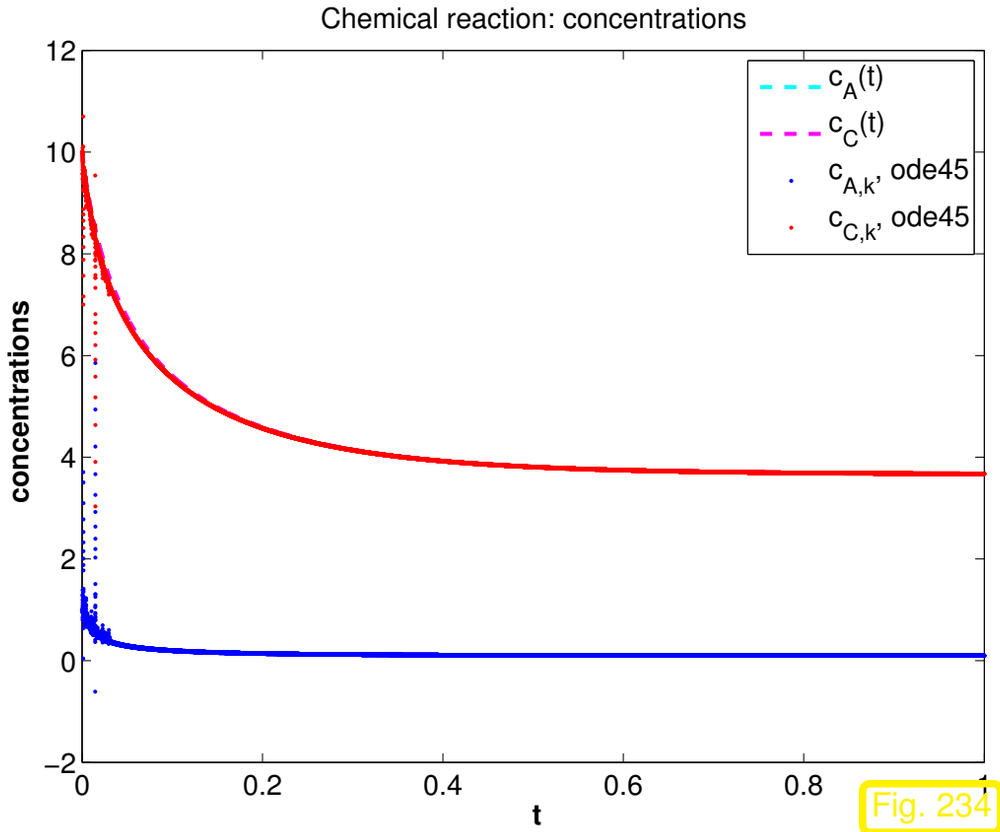


Fig. 234

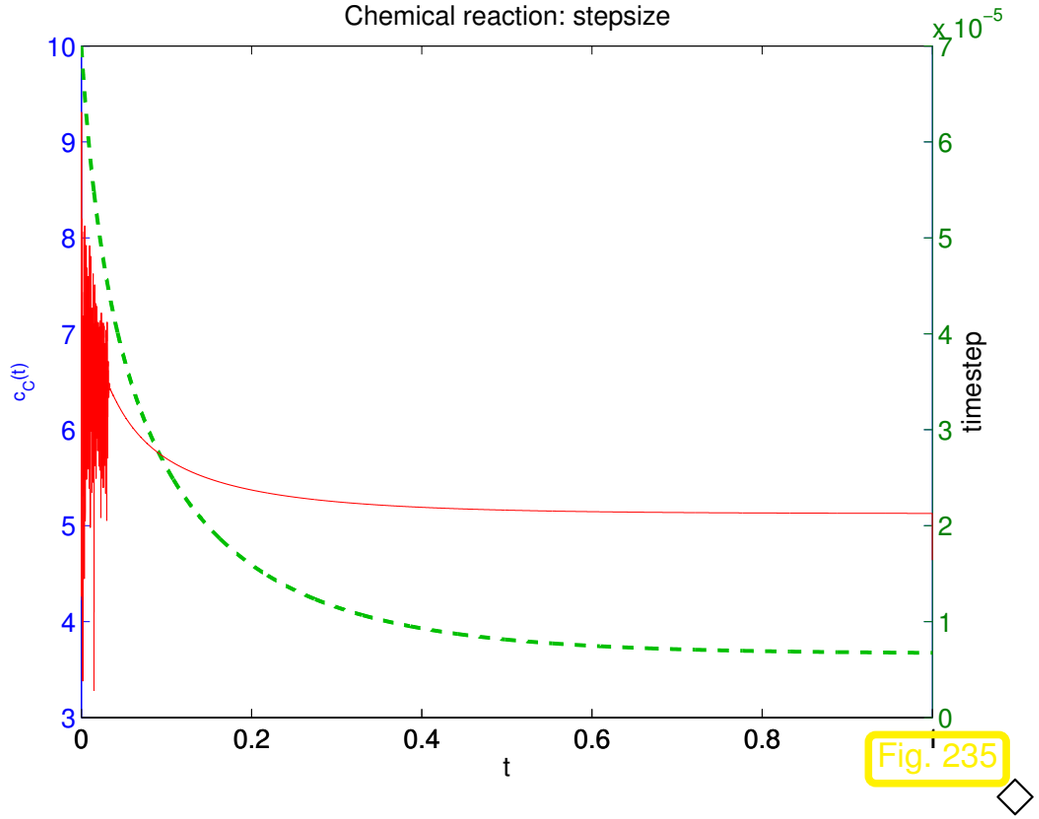


Fig. 235

Example 13.2.19 (Strongly attractive limit cycle).

Autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$

$$\mathbf{f}(\mathbf{y}) := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \mathbf{y} + \lambda(1 - \|\mathbf{y}\|^2)\,\mathbf{y}\;, \tag{13.2.20}$$

on state space $D = \mathbb{R}^2 \setminus \{0\}$

For $\lambda = 0$, the initial value problem $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix}$, $\varphi \in \mathbb{R}$ has the solution

$$\mathbf{y}(t) = \begin{pmatrix} \cos(t-\varphi) \\ \sin(t-\varphi) \end{pmatrix}\;, \quad t \in \mathbb{R}\;. \tag{13.2.22}$$

For this solution we have $\|\mathbf{y}(t)\|_2 = 1$ for all times.

(13.2.22) provides a solution even for $\lambda \neq 0$, if $\|\mathbf{y}(0)\|_2 = 1$, because in this case the term $\lambda(1 - \|\mathbf{y}\|^2)\,\mathbf{y}$ will never become non-zero on the solution trajectory.

R. Hiptmair

rev 30401, December 23, 2010

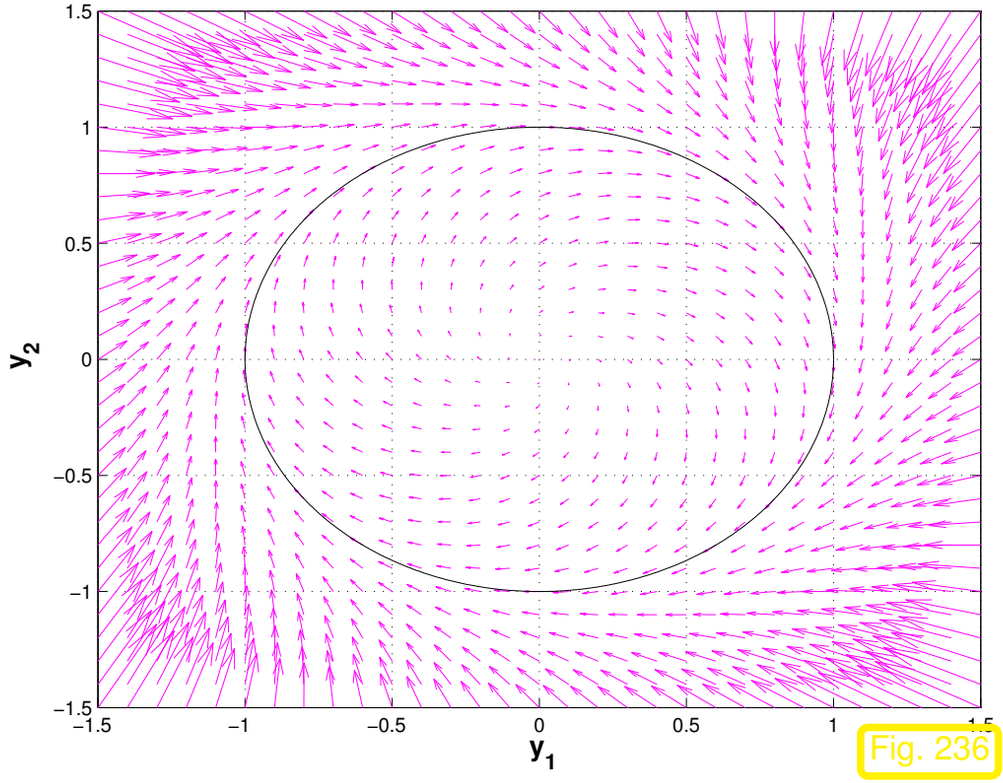


Fig. 236

vectorfield $\mathbf{f}$ ($\lambda = 1$)

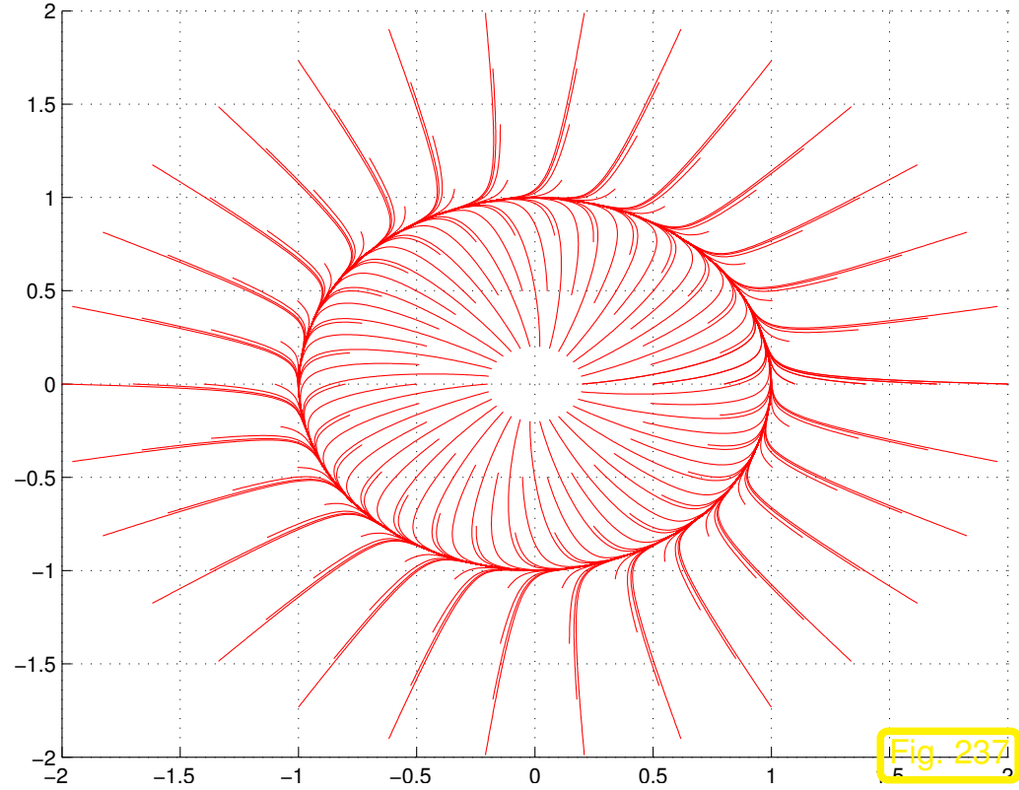


Fig. 237

solution trajectories ($\lambda = 10$)

R. Hiptmair

rev 30401, December 23, 2010

Code 13.2.24: Application of `ode45` for limit cycle problem

```
1 % MATLAB script for solving limit cycle ODE (13.2.20)
2 % define right hand side vectorfield
3 fun = @(t,y) ([-y(2);y(1)] +
    lambda*(1-y(1)\symbol{94}2-y(2)\symbol{94}2)*y);
4 % standard invocation of MATLAB integrator, see Ex. 12.4.15
5 tspan = [0,2*pi]; y0 = [1,0];
```

```
opts = odeset('stats','on','reltol',1E-4,'abstol',1E-4);
[t45,y45] = ode45(fun,tspan,y0,opts);
```

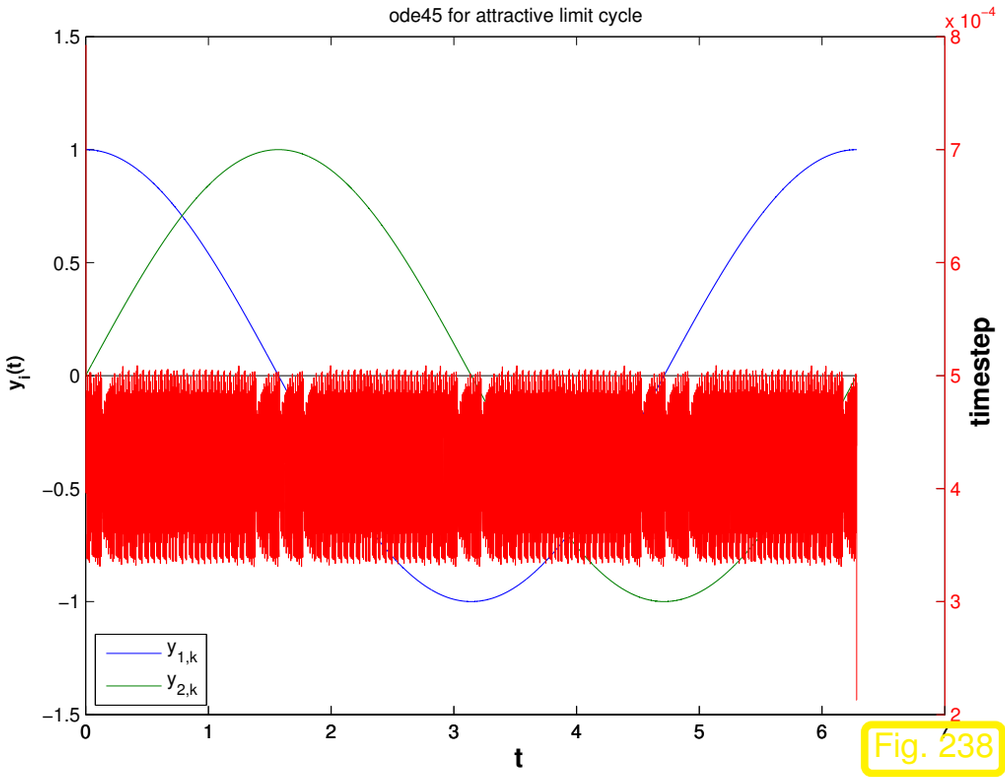
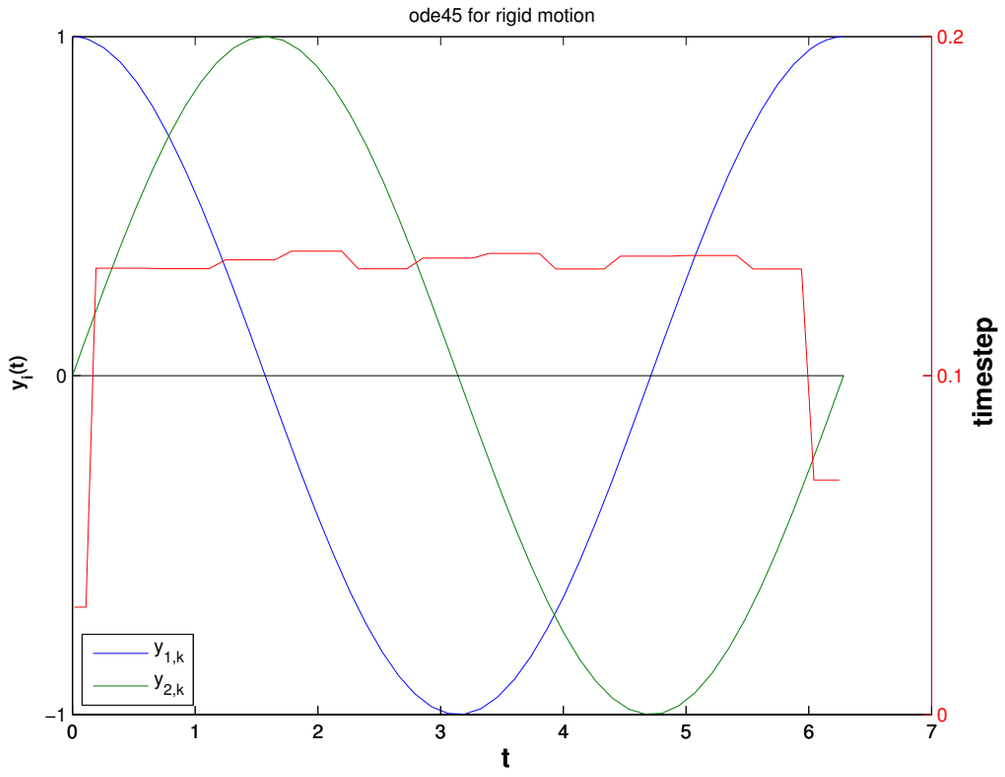



Fig. 238

many (3794) steps ($\lambda = 1000$)



accurate solution with few steps ($\lambda = 0$)

Confusing observation: we have $\|\mathbf{y}_0\| = 1$, which implies $\|\mathbf{y}(t)\| = 1 \quad \forall t$!

R. Hiptmair

rev 30401, December 23, 2010

Thus, the term of the right hand side, which is multiplied by λ will always vanish on the exact solution trajectory, which stays on the unit circle.

Nevertheless, `ode45` is forced to use tiny timesteps by the mere presence of this term.

◇

Notion 13.2.25 (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. 13.2.19

13.3 Implicit Runge-Kutta methods [10, Sect. 11.6.2]

Example 13.3.1 (Implicit Euler timestepping for decay equation).

Again, model problem analysis: study implicit Euler method (12.2.10) for IVP (13.1.4)

$$\blacktriangleright \quad \text{sequence} \qquad y_k := \left(\frac{1}{1-\lambda h}\right)^k y_0 \; . \tag{13.3.4}$$

$$\Rightarrow \quad \boxed{\operatorname{Re}\lambda < 0 \quad \Rightarrow \quad \lim_{k\to\infty} y_k = 0 \; !} \tag{13.3.5}$$

No timestep constraint: qualitatively correct behavior of $(y_k)_k$ for $\operatorname{Re}\lambda < 0$ and **any** $h > 0$!

◇

R. Hiptmair

rev 30401, December 23, 2010

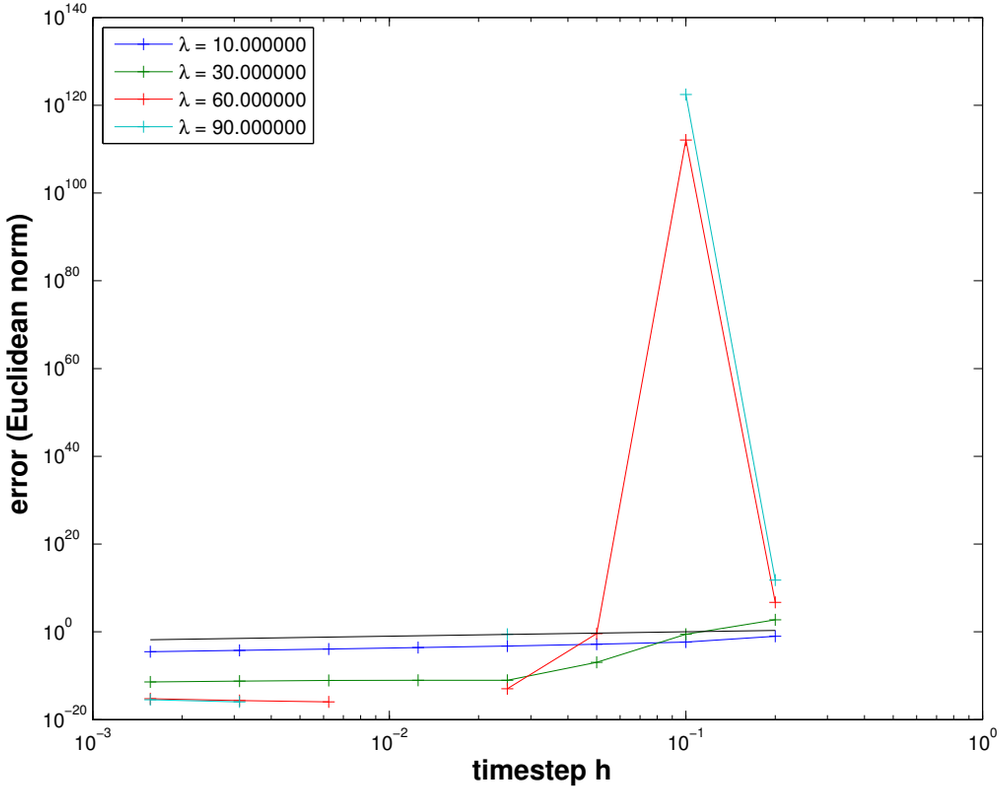
Observe: transformation idea, see (13.2.10), (13.2.12), applies to explicit *and implicit* Euler method alike.

Conjecture: implicit Euler method will not face timestep constraint for stiff problems ($\rightarrow$ Notion 13.2.25).

Example 13.3.6 (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 (12.2.10)

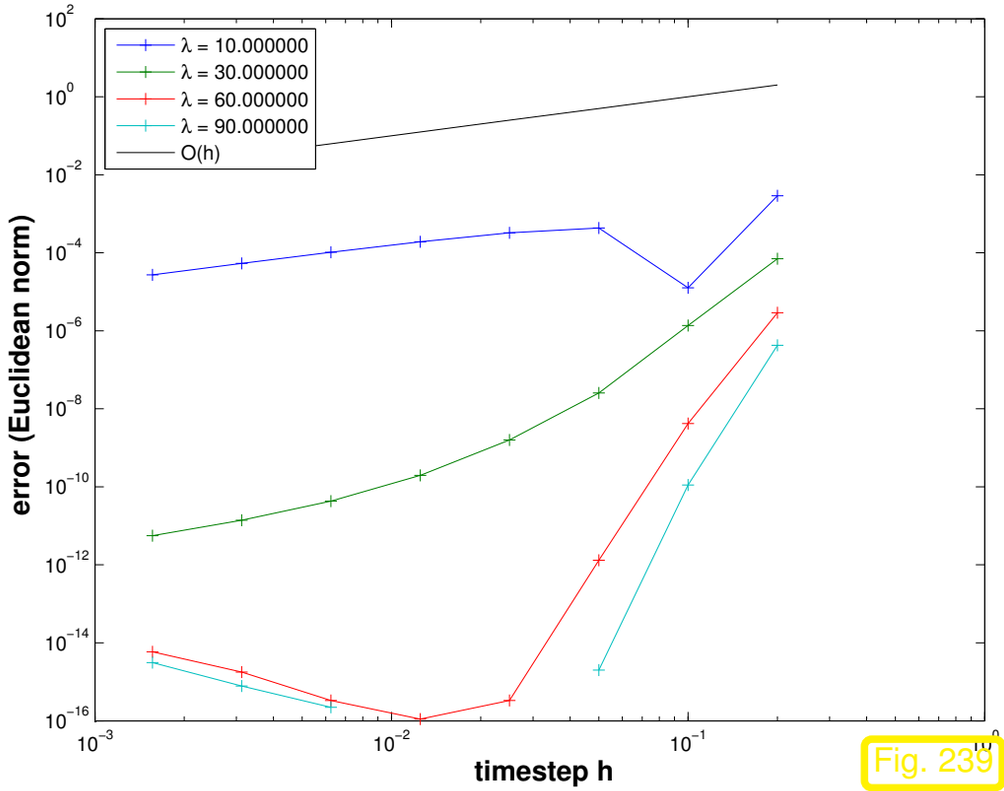


Fig. 239

λ large: stable for all timesteps h **!**

Well, we see what we expected!

◇

Unfortunately the implicit Euler method is of first order only, see Ex. 12.3.3. Can the Runge-Kutta design principle for integrators also yield higher order methods, which can cope with stiff problems?

YES !

Definition 13.3.7 (General Runge-Kutta method)**.** *(cf. Def. 12.4.9)*

For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{s} a_{ij}$, $i,j = 1,\ldots,s$, $s \in \mathbb{N}$, *an* s*-stage Runge-Kutta single step method (RK-SSM) for the IVP* (12.1.13) *is defined by*

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j) \;, \quad i = 1,\ldots,s \quad , \quad \mathbf{y}_1 := \mathbf{y}_0 + h\sum_{i=1}^{s} b_i \mathbf{k}_i \;.$$

As before, the $\mathbf{k}_i \in \mathbb{R}^d$ *are called increments.*

Note: computation of increments $\mathbf{k}_i$ may now require the solution of *(non-linear) systems of equations* of size $s \cdot d$ ($\rightarrow$ "implicit" method)

R. Hiptmair

rev 30401, December 23, 2010

Shorthand notation for Runge-Kutta methods

Butcher scheme ▷

$$\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} := \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} . \quad (13.3.8)$$

Note: now $\mathfrak{A}$ can be a general $s \times s$-matrix.

$\mathfrak{A}$ strict lower triangular matrix ➤ explicit Runge-Kutta method, Def. 12.4.9
$\mathfrak{A}$ lower triangular matrix ➤ diagonally-implicit Runge-Kutta method (DIRK)

Model problem analysis for general Runge-Kutta single step methods ($\rightarrow$ Def. 13.3.7): exactly the same as for explicit RK-methods, see (13.1.16), (13.1.17)!

Theorem 13.3.9 (Stability function of Runge-Kutta methods)**.**

The discrete evolution Ψ_λ^h of an s-stage Runge-Kutta single step method ($\rightarrow$ Def. 13.3.7) with Butcher scheme $\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$ *(see* (13.3.8)*) for the ODE $\dot{y} = \lambda y$ is a multiplication operator according to*

$$\Psi_\lambda^h = \underbrace{1 + z\mathbf{b}^T \left(\mathbf{I} - z\mathfrak{A}\right)^{-1} \mathbf{1}}_{\text{stability function } S(z)} = \frac{\det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T)}{\det(\mathbf{I} - z\mathfrak{A})} \;, \quad z := \lambda h \;, \quad \mathbf{1} = (1, \ldots, 1)^T \in \mathbb{R}^s \;.$$

Note: from the determinant represenation of $S(z)$ we infer that the stability function of an s-stage Runge-Kutta method is a rational function of the form $S(z) = \dfrac{P(z)}{Q(z)}$ with $P \in \mathcal{P}_s$, $Q \in \mathcal{P}_s$.

Of course, such rational functions can satisfy $|S(z)| < 1$ for all $z < 0$. For example, the stability function of the implicit Euler method (12.2.10) is

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \quad \overset{\text{Thm. 13.3.9}}{\Rightarrow} \quad S(z) = \frac{1}{1-z} \;. \tag{13.3.10}$$

R. Hiptmair

rev 30401, December 23, 2010

In light of the previous detailed analysis we can now state what we expect from the stability function of a Runge-Kutta method that is suitable for stiff IVP ($\rightarrow$ Notion 13.2.25):

Definition 13.3.11 (L-stable Runge-Kutta method)**.** $\rightarrow$ *[29, Ch. 77]*

A Runge-Kutta method ($\rightarrow$ Def. 13.3.7) is L-stable/asymptotically stable, if its stability function ($\rightarrow$ Def. 13.3.9) satisfies

$$(i) \qquad \operatorname{Re} z < 0 \quad \Rightarrow \quad |S(z)| < 1 \;, \tag{13.3.12}$$

$$(ii) \qquad \lim_{\operatorname{Re} z \to -\infty} S(z) = 0 \;. \tag{13.3.13}$$

Remark 13.3.17 (Necessary condition for L-stability of Runge-Kutta methods)*.*

Consider: Runge-Kutta method ($\rightarrow$ Def. 13.3.7) with Butcher scheme $\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$

Assume: $\mathfrak{A} \in \mathbb{R}^{s,s}$ is regular

R. Hiptmair

rev 30401, December 23, 2010

For a rational function $S(z) = \frac{P(z)}{Q(z)}$ the limit for $|z| \to \infty$ exists and can easily be expressed by the leading coefficients of the polynomials P and Q:

$$\text{Thm. 13.3.9} \quad \Rightarrow \quad S(-\infty) = 1 - \mathbf{b}^T \mathfrak{A}^{-1} \mathbf{1} \,. \tag{13.3.18}$$

$$\blacktriangleright \quad \boxed{\text{If } \mathbf{b}^T = (\mathfrak{A})^T_{:,j} \text{ (row of } \mathfrak{A}) \quad \Rightarrow \quad S(-\infty) = 0} \,. \tag{13.3.19}$$

Butcher scheme (13.3.8) for L-stable RK-methods, see Def. 13.3.11

$$\triangleright \quad \begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} := \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_{s-1} & a_{s-1,1} & \cdots & a_{s-1,s} \\ 1 & b_1 & \cdots & b_s \\ \hline & b_1 & \cdots & b_s \end{array} \,. \qquad \triangle$$

Example 13.3.20 (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

Further information about Radau-Runge-Kutta single step methods can be found in [29, Ch. 79].

◇

13.4 Semi-implicit Runge-Kutta methods [29, Ch. 80]

Equations fixing increments $\mathbf{k}_i \in \mathbb{R}^d$, $i = 1, \ldots, s$, for s-stage implicit RK-method

=

(Non-)linear system of equations with $s \cdot d$ unknowns

On the other hand, we are computing approximate solutions anyway, and the increments *are weighted with stepsize* $h \ll 1$, see Def. 13.3.7. So there is no point in determining them with high accuracy.

➢ use a fixed *small* number of Newton steps to solve for $\mathbf{k}_i$, $i = 1, \ldots, s$.

Extreme case: use only a single Newton step! Let's try.

R. Hiptmair

rev 30401, December 23, 2010

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 (12.2.10) 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

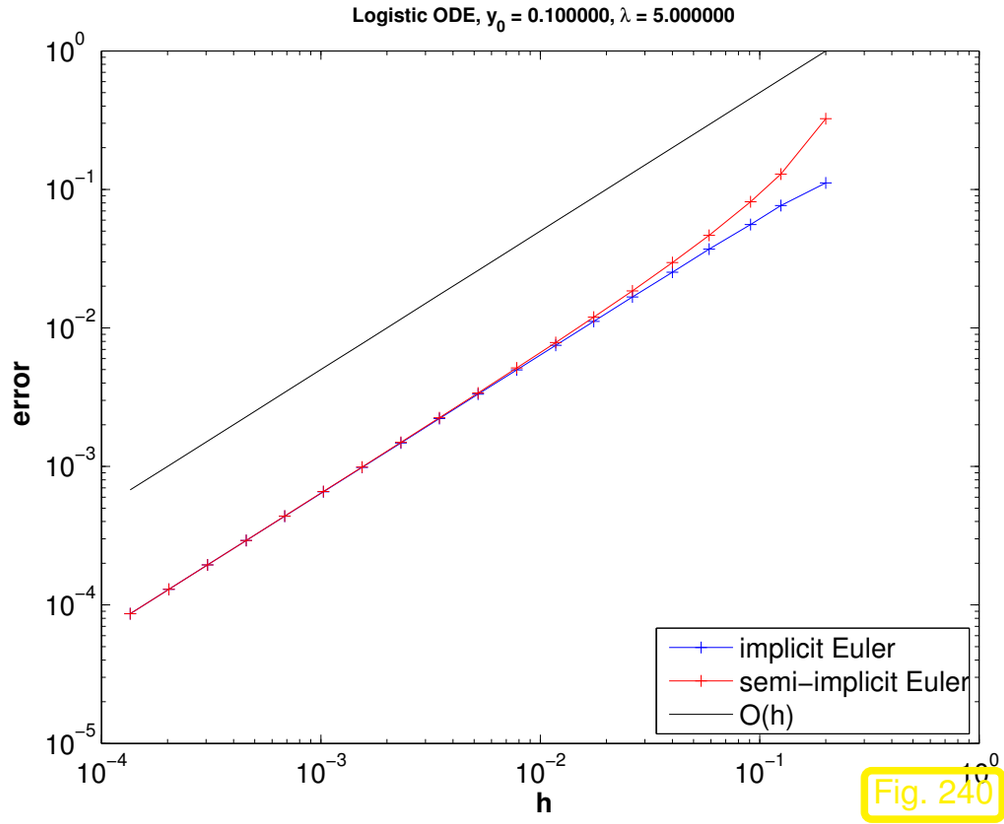


Fig. 240

- Measured error $\mathrm{err} = \max_{j=1,\ldots,n} |y_j - y(t_j)|$

From (12.2.10) with timestep $h > 0$

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}(\mathbf{y}_{k+1}) \quad \Leftrightarrow \quad F(\mathbf{y}_{k+1}) := \mathbf{y}_{k+1} - h\mathbf{f}(\mathbf{y}_{k+1}) - \mathbf{y}_k = 0 \, .$$

One Newton step applied to $F(\mathbf{y}) = 0$ with initial guess $\mathbf{y}_k$ yields

$$\mathbf{y}_{k+1} = \mathbf{y}_k - D\mathbf{f}(\mathbf{y}_k)^{-1} F(\mathbf{y}_k) = \mathbf{y}_k + (\mathbf{I} - hD\mathbf{f}(\mathbf{y}_k))^{-1} h\mathbf{f}(\mathbf{y}_k) \ .$$

Note: for linear ODE with $\mathbf{f}(\mathbf{y}) = \mathbf{A}\mathbf{y}$, $\mathbf{A} \in \mathbb{R}^{d,d}$, we recover the original implicit Euler method!

Observation: Approximate evaluation of defining equation for y_{k+1} preserves 1st order convergence.

$\diamond$

R. Hiptmair

rev 30401, December 23, 2010

Idea: Use linearized increment equations for implicit RK-SSM

$$\mathbf{k}_i = \mathbf{f}(\mathbf{y}_0) + hD\mathbf{f}(\mathbf{y}_0) \left(\sum_{j=1}^{s} a_{ij} \mathbf{k}_j \right) \ , \quad i = 1, \ldots, s \ . \tag{13.4.2}$$

Linearization does nothing for linear ODEs ➢ stability function ($\rightarrow$ Thm. 13.3.9) 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:

△

R. Hiptmair

rev 30401, December 23, 2010

14 Structure Preservation

14.1 Dissipative Evolutions

14.2 Quadratic Invariants

14.3 Reversible Integrators

14.4 Symplectic Integrators

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:

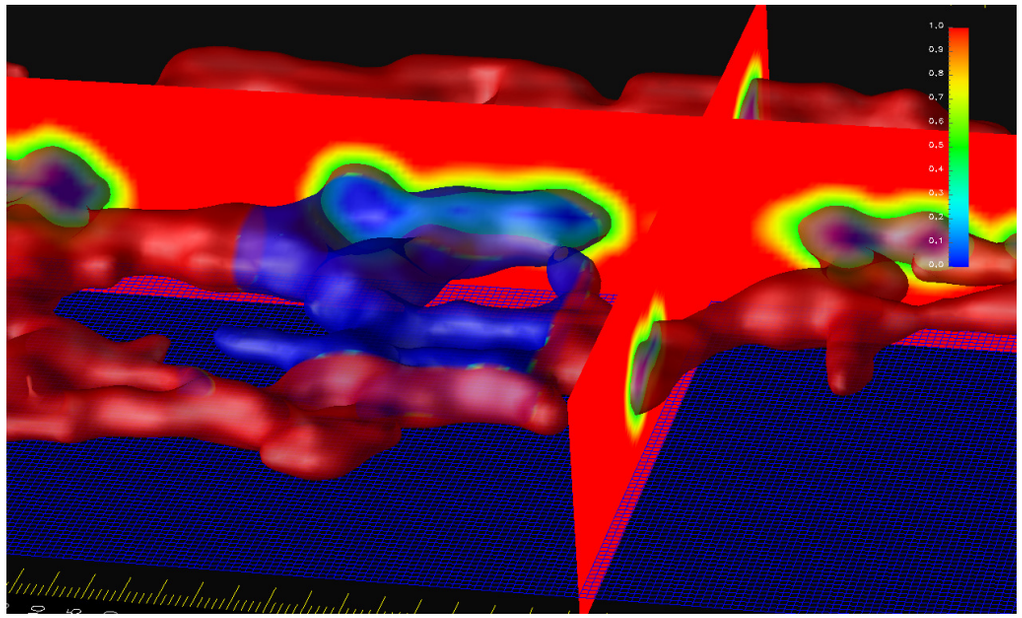
- Issue: Appropriate spatial (and temporal) discretization of PDE and boundary conditions
- Issue: fast solution methods for resulting *large* (non-)linear systems of equations

(initial) boundary value problems and techniques covered in the course:

❶ Stationary 2nd-order scalar elliptic boundary value problems

R. Hiptmair

rev 30401, November 10, 2009



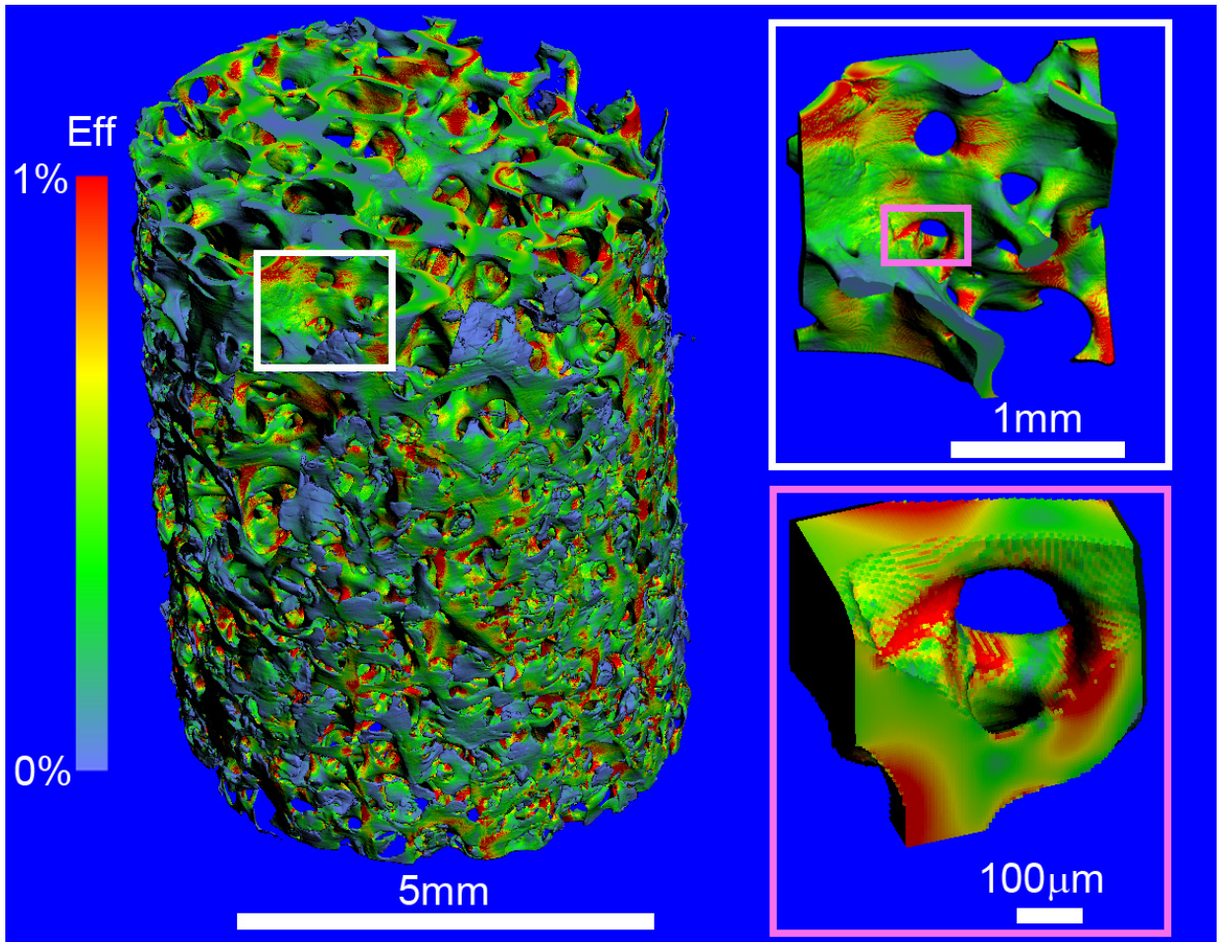
Diffusion boundary value problem:

$$\begin{aligned} -\operatorname{div}(\mathbf{A}(\boldsymbol{x})\,\mathbf{grad}\,u(\boldsymbol{x})) &= f(\boldsymbol{x}) \quad \text{in } \Omega \subset \mathbb{R}^d\,, \\ u &= g \qquad \text{on } \partial\Omega\,. \end{aligned}$$

◁ diffusion on the surface (membrane) of the endoplasmic reticulum (I. Sbalzarini, D-INFK, ETH Zürich)

R. Hiptmair

rev 30401, November 10, 2009



◁ 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

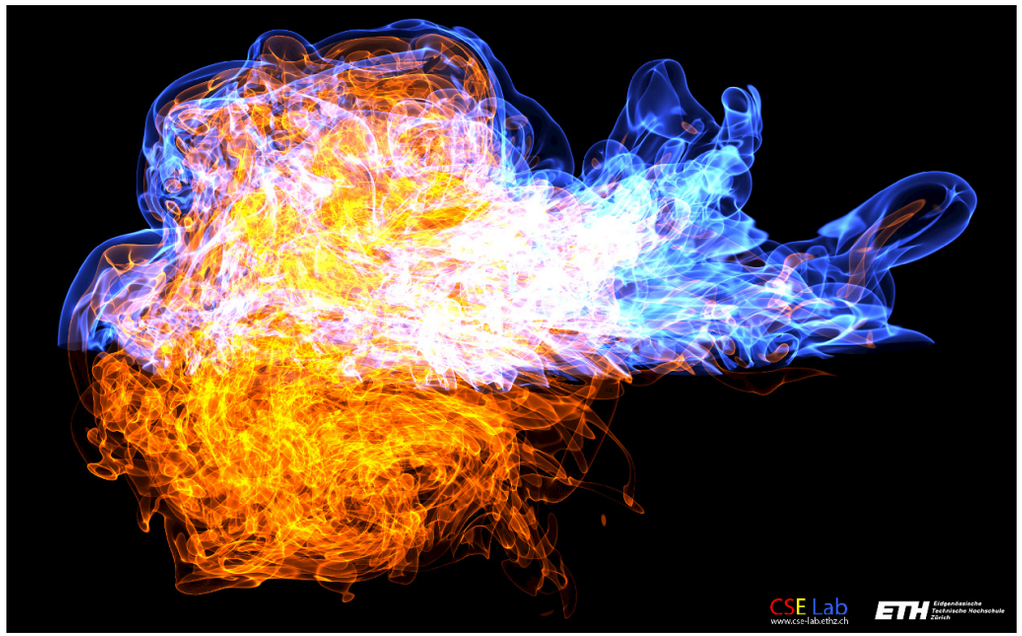
$$-\epsilon\Delta u + \mathbf{v}(\boldsymbol{x}) \cdot \mathbf{grad}\, u = 0 \quad \text{in } \Omega \quad , \quad u = g \quad \text{on } \partial\Omega \, .$$

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

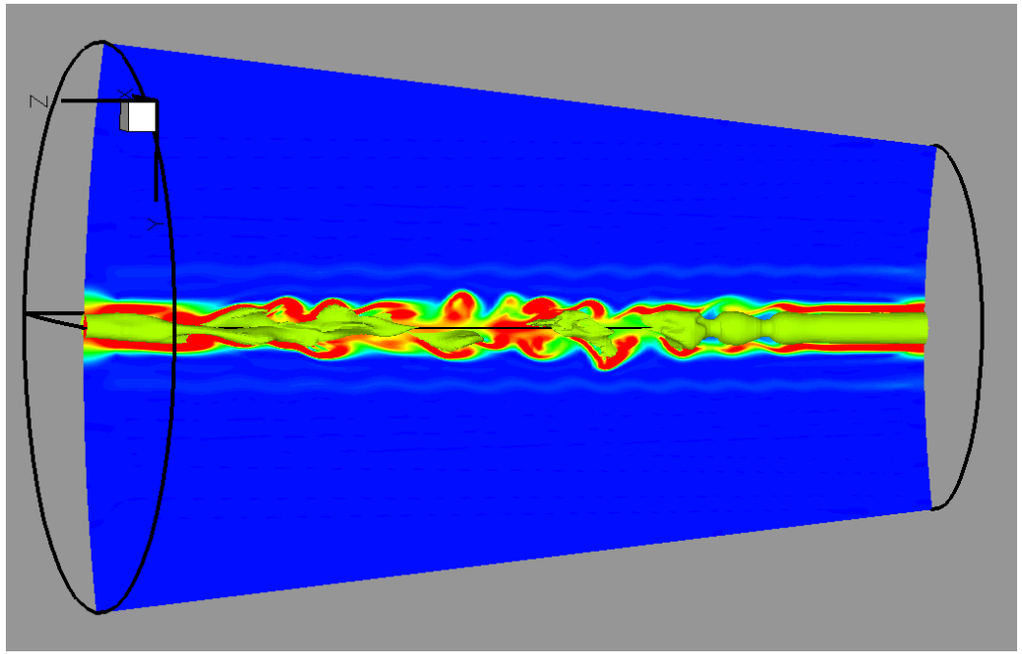
R. Hiptmair

rev 30401, November 10, 2009

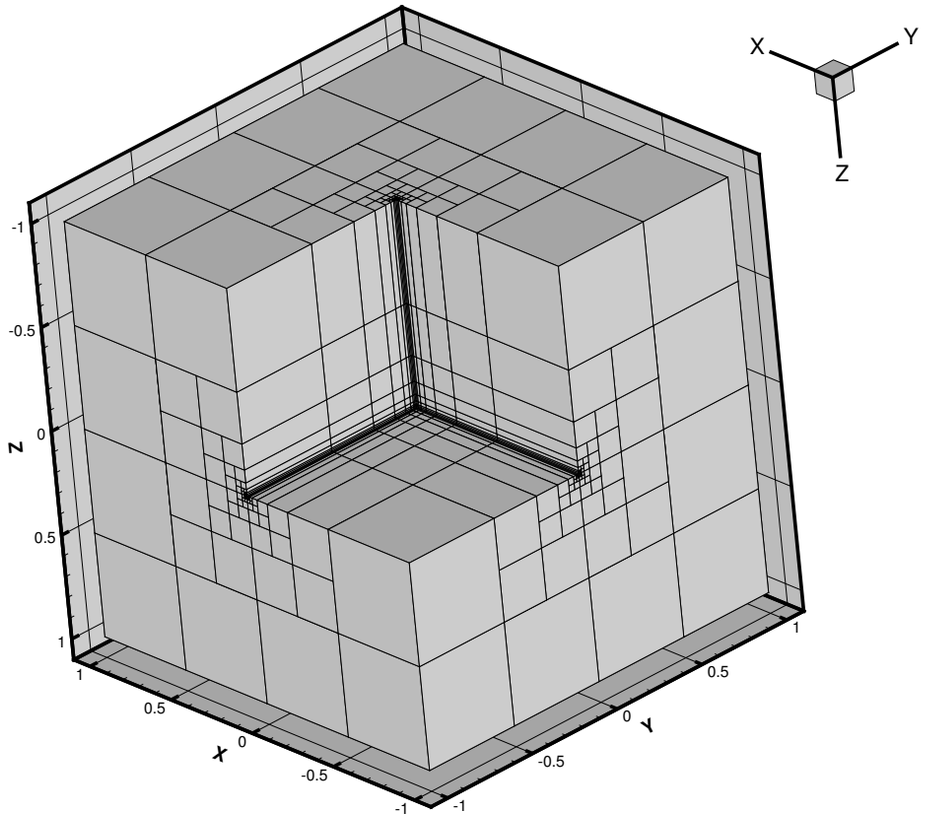
1D scalar conservation law with flux f:

$$\begin{aligned} \frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}(f(u)) &= 0 && \text{in } \mathbb{R}\times\mathbb{R}^+ \,, \\ u(x,0) &= u_0(x) && \text{for } x \in \mathbb{R} \,. \end{aligned}$$

Inviscid fluid flow in 3D (SAM, D-MATH, ETH Zürich) ▷



❻ Adaptive finite element methods



◁ Adaptive FEM for diffusion problem:

Geometrically graded mesh at re-entrant corner (SAM, D-MATH, ETH Zürich)

❼ Multilevel preconditioning

R. Hiptmair

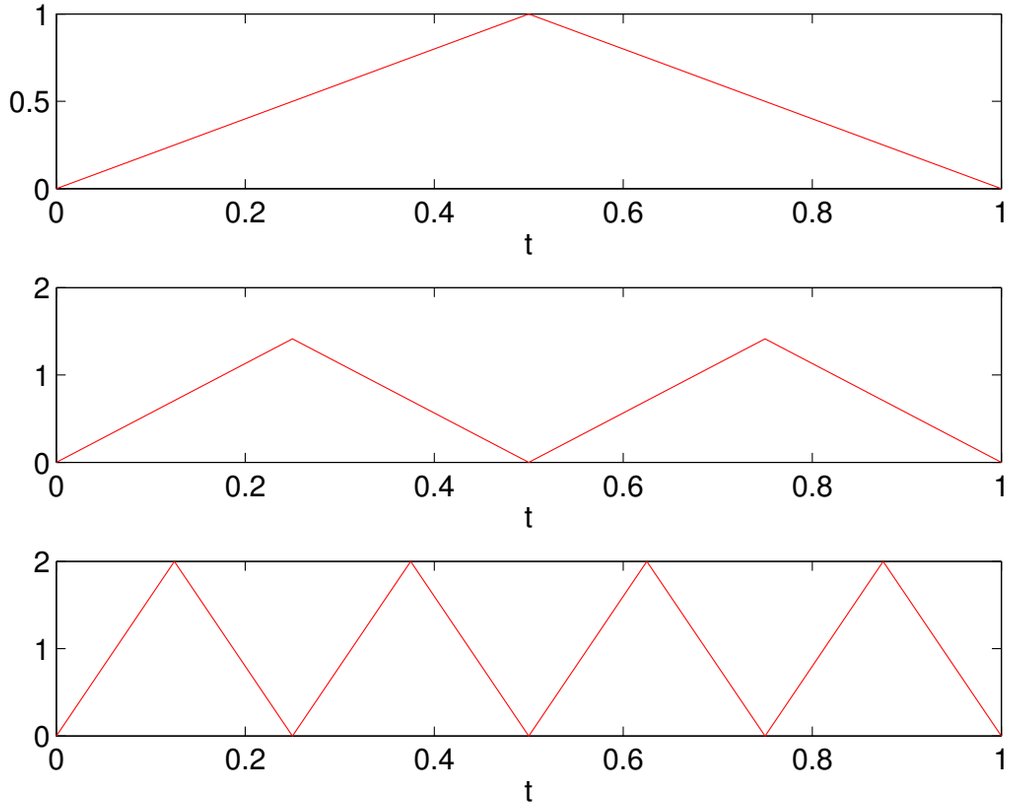
rev 30401, November 10, 2009

FEM, FD, FV

Huge sparse systems of equations

Efficient preconditioners required

1D hierarchical basis ▷



In SS11: Lecturers: Prof. Petros Koumoutsakos, Dr. Shilpa Khatri (D-INFK)

Classes: Mon 15-17, CAB G 11 and Fri 13-15, HG D 3.2

Tutorials: Thu 13-15 HG D 7.2, Fri 10-12 HG D 3.2

LINK to lecture notes

R. Hiptmair

rev 30401, November 10, 2009

Course: Parallel Computing for Scientific Simulations

This course will start in autumn term 2011 and deals with methods and techniques for numerical simulation on high performance (parallel) computers.

Bibliography

[1] H. AMANN, *Gewöhnliche Differentialgleichungen*, Walter de Gruyter, Berlin, 1st ed., 1983.

[2] 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.

[3] C. BISCHOF AND C. VAN LOAN, *The WY representation of Householder matrices*, SIAM J. Sci. Stat. Comput., 8 (1987).

[4] S. BÖRM, L. GRASEDYCK, AND W. HACKBUSCH, *Introduction to hierarchical matrices with applications*, Engineering Analysis with Boundary Elements, 27 (2003), pp. 405–422.

[5] F. BORNEMANN, *A model for understanding numerical stability*, IMA J. Numer. Anal., 27 (2006), pp. 219–231.

[6] E. BRIGHAM, *The Fast Fourier Transform and Its Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[7] 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.

[8] D. COPPERSMITH AND T. RIVLIN, *The growth of polynomials bounded at equally spaced points*, SIAM J. Math. Anal., 23 (1992), pp. 970–983.

[9] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progression*, J. Symbgolic Computing, 9 (1990), pp. 251–280.

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

[11] M. DEAKIN, *Applied catastrophe theory in the social and biological sciences*, Bulletin of Mathematical Biology, 42 (1980), pp. 647–679.

[12] P. DEUFLHARD, *Newton Methods for Nonlinear Problems*, vol. 35 of Springer Series in Computational Mathematics, Springer, Berlin, 2004.

[13] P. DEUFLHARD AND F. BORNEMANN, *Numerische Mathematik II*, DeGruyter, Berlin, 2 ed., 2002.

[14] P. DEUFLHARD AND A. HOHMANN, *Numerische Mathematik I*, DeGruyter, Berlin, 3 ed., 2002.

[15] P. DUHAMEL AND M. VETTERLI, *Fast fourier transforms: a tutorial review and a state of the art*, Signal Processing, 19 (1990), pp. 259–299.

[16] A. DUTT AND V. ROKHLIN, *Fast Fourier transforms for non-equispaced data II*, Appl. Comput. Harmon. Anal., 2 (1995), pp. 85–100.

R. Hiptmair

rev 30401, November 10, 2009

[17] F. FRITSCH AND R. CARLSON, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal., 17 (1980), pp. 238–246.

[18] M. GANDER, W. GANDER, G. GOLUB, AND D. GRUNTZ, *Scientific Computing: An introduction using MATLAB*, Springer, 2005. In Vorbereitung.

[19] 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.

[20] G. GOLUB AND C. VAN LOAN, *Matrix computations*, John Hopkins University Press, Baltimore, London, 2nd ed., 1989.

[21] 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.

[22] 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.

[23] M. GUTKNECHT, *Lineare algebra*, lecture notes, SAM, ETH Zürich, 2009. http://www.sam.math.ethz.ch/m̃hg/unt/LA/HS07/.

[24] W. HACKBUSCH, *Iterative Lösung großer linearer Gleichungssysteme*, B.G. Teubner–Verlag, Stuttgart, 1991.

[25] ——, *Iterative Solution of Large Sparse Systems of Equations*, vol. 95 of Applied Mathematical Sciences, Springer-Verlag, New York, 1993.

[26] W. HACKBUSCH AND S. BÖRM, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.

[27] E. HAIRER, C. LUBICH, AND G. WANNER, *Geometric numerical integration*, vol. 31 of Springer Series in Computational Mathematics, Springer, Heidelberg, 2002.

[28] C. HALL AND W. MEYER, *Optimal error bounds for cubic spline interpolation*, J. Approx. Theory, 16 (1976), pp. 105–122.

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

[30] N. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 2 ed., 2002.

[31] S. JOHNSON, *Notes on the convergence of trapezoidal-rule quadrature*. MIT online course notes, http://math.mit.edu/ stevenj/trapezoidal.pdf, 2008.

[32] 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.

[33] 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.

[34] A. LENGVILLE AND C. MEYER, *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, Princeton, NJ, 2006.

[35] D. MCALLISTER AND J. ROULIER, *An algorithm for computing a shape-preserving osculatory quadratic spline*, ACM Trans. Math. Software, 7 (1981), pp. 331–347.

[36] C. MOLER, *Numerical Computing with MATLAB*, SIAM, Philadelphia, PA, 2004.

[37] 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.

[38] ___, *A geometric theory for preconditioned inverse iteration: III. Sharp convergence estimates*, Tech. Rep. 130, SFB 382, Universität Tübingen, Tübingen, Germany, November 1999.

[39] K. NIPP AND D. STOFFER, *Lineare Algebra*, vdf Hochschulverlag, Zürich, 5 ed., 2002.

[40] M. OVERTON, *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, Philadelphia, PA, 2001.

[41] 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.

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

[43] C. RADER, *Discrete Fourier transforms when the number of data samples is prime*, Proceedings of the IEEE, 56 (1968), pp. 1107–1108.

[44] R. RANNACHER, *Einführung in die numerische mathematik*. Vorlesungsskriptum Universität Heidelberg, 2000. http://gaia.iwr.uni-heidelberg.de/.

[45] V. ROKHLIN, *Rapid solution of integral equations of classical potential theory*, J. Comp. Phys., 60 (1985), pp. 187–207.

[46] 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.

[47] T. SAUER, *Numerical analysis*, Addison Wesley, Boston, 2006.

[48] J.-B. SHI AND J. MALIK, *Normalized cuts and image segmentation*, IEEE Trans. Pattern Analysis and Machine Intelligence, 22 (2000), pp. 888–905.

[49] M. STEWART, *A superfast toeplitz solver with improved numerical stability*, SIAM J. Matrix Analysis Appl., 25 (2003), pp. 669–693.

[50] J. STOER, *Einführung in die Numerische Mathematik*, Heidelberger Taschenbücher, Springer, 4 ed., 1983.

[51] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[52] M. STRUWE, *Analysis für informatiker*. Lecture notes, ETH Zürich, 2009. https://moodle-app1.net.ethz.ch/lms/mod/resource/index.php?id=145.

[53] F. TISSEUR AND K. MEERBERGEN, *The quadratic eigenvalue problem*, SIAM Review, 43 (2001), pp. 235–286.

[54] L. TREFETHEN AND D. BAU, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[55] P. VERTESI, *On the optimal lebesgue constants for polynomial interpolation*, Acta Math. Hungaria, 47 (1986), pp. 165–178.

[56] ——, *Optimal lebesgue constant for lagrange interpolation*, SIAM J. Numer. Aanal., 27 (1990), pp. 1322–1331.

Index

R. Hiptmair

rev 30401, November 10, 2009

R. Hiptmair

rev 30401, November 10, 2009

R. Hiptmair

rev 30401, November 10, 2009

R. Hiptmair

rev 30401, November 10, 2009

R. Hiptmair

rev 30401, November 10, 2009

List of Symbols

$(x_k) *_n (y_k) \hat{=}$ discrete periodic convolution, 738

$C^0_{\mathrm{pw}}(I) \hat{=}$ space of piecewise continuous functions on interval I, 880

$C^1([a,b]) \hat{=}$ space of continuously differentiable functions $[a,b] \mapsto \mathbb{R}$, 321

$D\Phi \hat{=}$ Jacobian of $\Phi : D \mapsto \mathbb{R}^n$ at $\mathbf{x} \in D$, 393

$D_{\mathbf{y}}\mathbf{f} \hat{=}$ Derivative of $\mathbf{f}$ w.r.t.. $\mathbf{y}$ (Jacobian), 1024

$J(t_0, \mathbf{y}_0) \hat{=}$ maximal domain of definition of a solution of an IVP, 1025

$O \hat{=}$ zero matrix, 29

$O(n)$, 45

$\mathcal{E} \hat{=}$ expected value of a random variable, 809

$\mathcal{P}_n^T \hat{=}$ space of trigonometric polynomials of degree n, 860

$\mathcal{R}_k(m,n)$, 672

$\mathrm{eps} \hat{=}$ machine precision, 129

$\mathrm{Eig}_{\mathbf{A}}(\lambda) \hat{=}$ eigenspace of $\mathbf{A}$ for eigenvalue λ, 533

$\mathrm{Im}(\mathbf{A}) \hat{=}$ range/column space of matrix $\mathbf{A}$, 654

$\mathrm{Ker}(\mathbf{A}) \hat{=}$ nullspace of matrix $\mathbf{A}$, 654

$\mathcal{K}_l(\mathbf{A}, \mathbf{z}) \hat{=}$ Krylov subspace, 481

$\|\mathbf{A}\|_F^2$, 671

$\|\mathbf{x}\|_A \hat{=}$ energy norm induced by s.p.d. matrix $\mathbf{A}$, 459

$\|f\|_{L^\infty(I)}$, 279

$\|f\|_{L^1(I)}$, 280

$\|f\|^2_{L^2(I)}$, 280

$\mathcal{P}_k$, 268

R. Hiptmair

rev 30401, November 10, 2009

R. Hiptmair

rev 30401, November 10, 2009

List of Definitions

R. Hiptmair

rev 30401, November 10, 2009

Examples and Remarks

R. Hiptmair

R. Hiptmair

rev 30401, November 10, 2009

R. Hiptmair

rev 30401, November 10, 2009

German terms

L^2-inner product = L^2-Skalarprodukt, 940
], 83

bandwidth = Bandbreite, 185

cancellation = Auslöschung, 301
capacitance = Kapazität, 77
capacitor = Kondensator, 77
coil = Spule, 77
column (of a matrix) = (Matrix)spalte, 27
column transformation = Spaltenumformung, 41
column vector = Spaltenvektor, 25
composite quadrature formulas = zusammengesetzte Quadraturformeln, 913
computational effort = Rechenaufwand, 44
consistency = Konsistenz, 365
constitutive relation = Kennlinie, 263
constitutive relations = Bauelementgleichungen, 76
constrained least squares = Ausgleichsproblem mit Nebenbedingungen, 712
convergence = Konvergenz, 365

damped Newton method = gedämpftes Newton-Verfahren, 442
dense matrix = vollbesetzte Matrix, 159
descent methods = Abstiegsverfahren, 459
divided differences = dividierte Differenzen, 305
dot product = (Euklidisches) Skalarprodukt, 34

eigenspace = Eigenraum, 533
eigenvalue = Eigenwert, 533

R. Hiptmair

rev 30401, November 10, 2009

R. Hiptmair

rev 30401, November 10, 2009