Numerical Methods

for Computational Science and Engineering

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

Prof. Ralf Hiptmair and Dr. Vasile Gradinaru

Draft version December 18, 2009, Subversion rev #22659

http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE09.pdf

Contents

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

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)

Books

- 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.
- N. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 2 ed., 2002.
- L. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.
- P. Deuflhard and A. Hohmann, *Numerische Mathematik. Eine algorithmisch orientierte Einführung*, DeGruyter, Berlin, 1 ed., 1991.

General information

Lecturer:	Prof. Ralf Hiptmair	HG G 58.2	☎ 044 632 3404	hiptmair@sam.math.ethz.ch
Assistants:	Marcus Wittberger	CAB G 65.1	☎ 044 632 7547	marcus.wittberger@inf.ethz.ch
	Peter Kauf	HG J 46	☎ 044 632 6572	peter.kauf@sam.math.ethz.ch
	Dr. Julia Schweitzer	HG G 53.2	☎ 044 632 3455	julia.schweitzer@sam.math.ethz.ch
	Manfred Quack	CAB F 84	☎ 044 632 86 78	manfred.quack@inf.ethz.ch

Marcus Wittberger (D-INFK)

Dr. Julia Schweitzer (SAM, D-MATH)

Manfred Quack (D-INFK)

Peter Kauf (SAM, D-MATH)

Classes:	Mon, 8.15-10.00 (HG G5), Thu, 10.15-11.55 (HG G5)
Tutorials:	Mon 10.15-11.55 (HG E 21, for students of RW/CSE)
	Thu 8.15-10.00 (for students of computer science)

Assignments.

- One problem sheet will be handed out every week
- "Testatbedingung": *attempted* solutions for a least 60% of the problems.

Website: `http://www.math.ethz.ch/education/bachelor/lectures/hs2009/math/nummath_cs`

Examination

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

  Thu, Feb 4, 2010, 09:00 – 12:00

- Dry-run for computer based examination:

  Mon, Jan 18, 2010, 09:00, registration via course website

- Pre-exam question session:

  Mon, Jan 18, 2010, 10:15-12:009:00, room will be announced

- Subject of examination:
  - Chapters 1 through 11 of the course,
  - homework problems on sheet 1 through 13.
- Lecture documents will be available as PDF during the examination, both in four-page and one-page layout. The correspoding final version of the lecture documents will be made available in Jan, 18, 2010.
- 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,
- 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: 23.11.2009

01: I try to do all programming exercises.

02: The programming exercises help understand the numerical methods.

03: The programming exercises offer too little benefit for the effort spent on them.

04: Scope and limitations of methods are properly addressed in the course.

05: Numerical examples in class provide useful insights and motivation.

06: There should be more examples presented and discussed in class.

07: Too much information is crammed onto the lecture slides

08: The course requires too much prior knowledge in linear algebra

09: The course requires too much prior knowledge in analysis

10: My prior knowledge of MATLAB was insufficient for the course

11: More formal proofs would be desirable

12: The explanations on the blackboard promote understanding

13: The codes included in the lecture material convey useful information

14: The model solutions for exercise problems offer too little guidance.

15: The relevance of the numerical methods taught in the course is convincingly conveyed.

16: I would not mind the course being taught in English.

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
Julia Schweitzer	JUL
Manfred Quack	MAN
Peter Kauf	PET
Daniel Wright	DAN
Marcelo Serrano Zanetti	SER
Marcus Wittberger	MAR

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

Evaluation results: Fall semester 2009

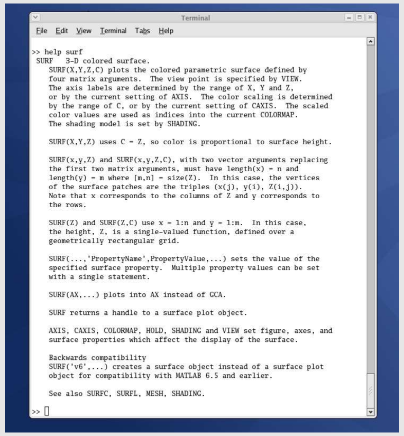
MATLAB

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)



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

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

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

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

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

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

    SURF returns a handle to a surface plot object.

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

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

    See also SURFC, SURFL, MESH, SHADING.

>> 
```

Useful links:

Matlab Online Documentation
MATLAB guide
MATLAB Primer

Part I

Systems of Equations

1 Computing with Matrices and Vectors

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 [18, Ch. 1].

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

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{K}^n \qquad \Big| \qquad (x_1 \cdots x_n)$$

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';`

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

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

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 $(a_{ij})_{\substack{i=k,\dots,l \\ j=r,\dots,s}} = (\mathbf{A})_{k:l,r:s}$ = A(k:l,r:s)
(sub-matrix)

- Transposed matrix

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

- Adjoint matrix

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

$\bar{a}_{ij} = \mathfrak{Re}(a_{ij}) - i\mathfrak{Im}(a_{ij})$ a_{ij}

-

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

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

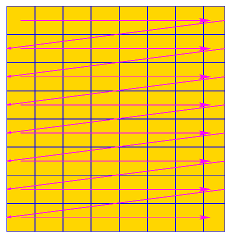
Access to entry a_{ij} of $\mathbf{A} \in \mathbb{K}^{n,m}$, $i = 1,\dots,n$, $j = 1,\dots,m$:

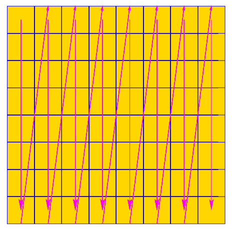
row major:

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

column major:

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


row major


column major

△

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

Cache hierarchies $\leadsto$ slow access of "remote" memory sites !

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

$n = 3000 \leadsto 0.1s$

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

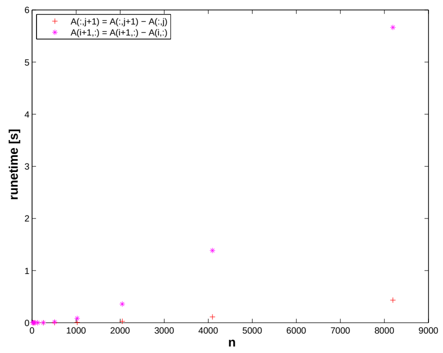
$n = 3000 \leadsto 0.3s$

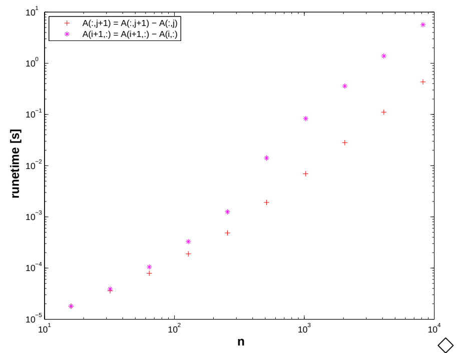
Code 1.1.3: timing for row and column oriented matrix access in MATLAB

```
%Timing for row/column operations
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

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

figure; loglog(res(:,1),res(:,2),'r+', res(:,1),res(:,3),'m*');
```

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





◇

1.2 Elementary operations

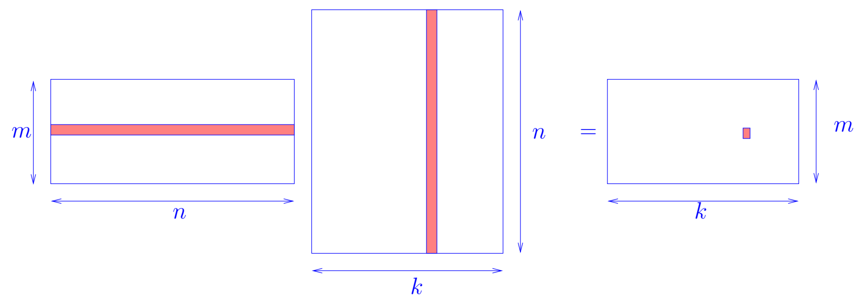
What you should know from linear algebra:

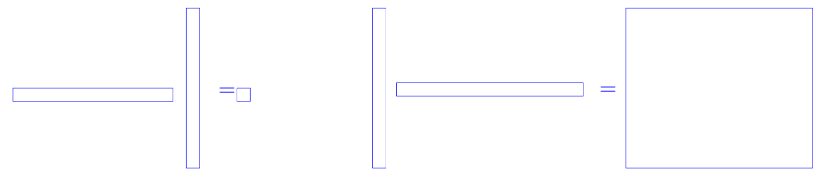
- vector space operations in $\mathbb{K}^{m,n}$ (addition, multiplication with scalars)
- dot product: $\mathbf{x},\mathbf{y}\in\mathbb{K}^n$, $n\in\mathbb{N}$: $\mathbf{x}\cdot\mathbf{y} := \mathbf{x}^H\mathbf{y} = \sum_{i=1}^{n}\bar{x}_i y_i \in \mathbb{K}$

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

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

"Visualization" of matrix product:





dot product   tensor product

Remark 1.2.1 (Row-wise & column-wise view of matrix product).

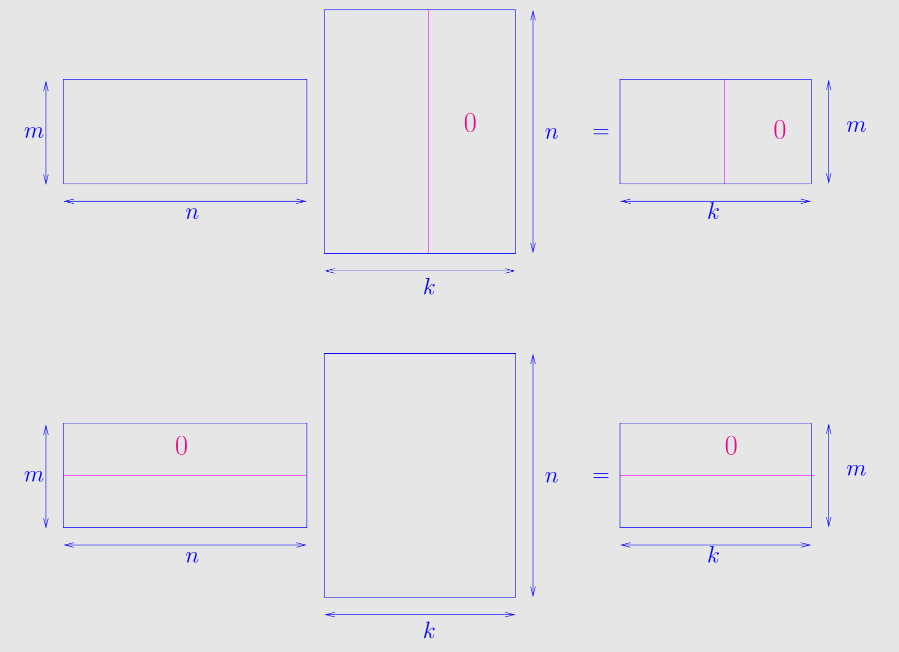
$\mathbf{A} \in \mathbb{K}^{m,n}$, $\mathbf{B} \in \mathbb{K}^{n,k}$:

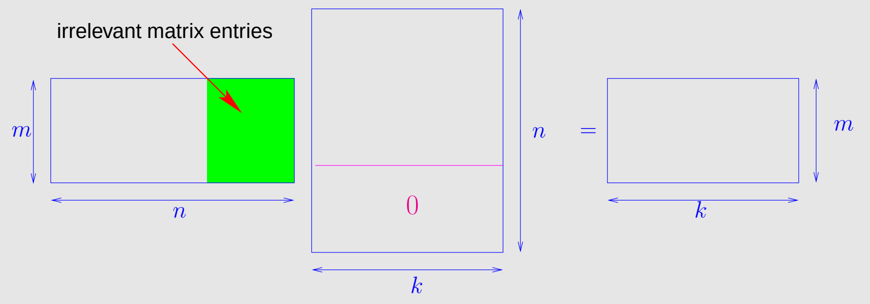
$$\mathbf{AB} = \begin{pmatrix} \mathbf{A}(\mathbf{B})_{:,1} & \dots & \mathbf{A}(\mathbf{B})_{:,k} \end{pmatrix} \quad , \quad \mathbf{AB} = \begin{pmatrix} (\mathbf{A})_{1,:}\mathbf{B} \\ \vdots \\ (\mathbf{A})_{m,:}\mathbf{B} \end{pmatrix} . \tag{1.2.2}$$

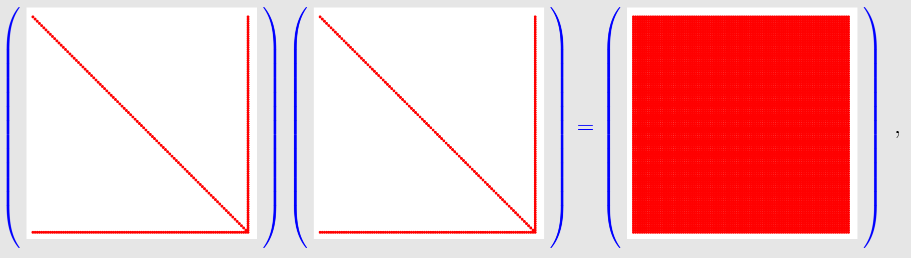
$\downarrow$ matrix assembled from columns $\qquad$ $\downarrow$ matrix assembled from rows

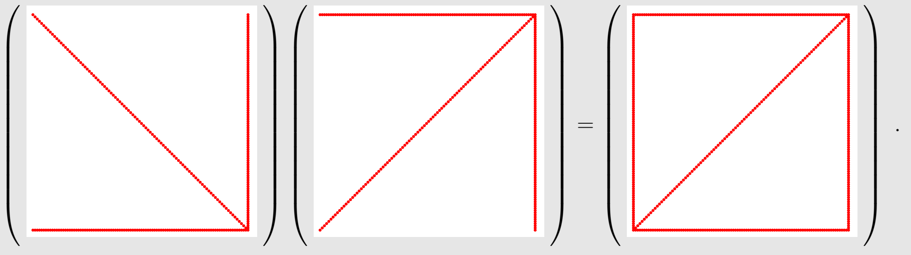
$\triangle$

Remark 1.2.2 (Understanding the structure of product matrices). A "mental image" of matrix multiplication is useful for telling special properties of product matrices









Code 1.2.3: 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 1.2.4 (Scalings).

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

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

$$\begin{pmatrix} d_1 & 0 & & 0 \\ 0 & d_2 & & 0 \\ & & \ddots & \\ 0 & 0 & & d_n \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & & a_{2m} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} = \begin{pmatrix} d_1a_{11} & d_1a_{12} & \dots & d_1a_{1m} \\ d_2a_{21} & d_2a_{22} & \dots & d_2a_{2m} \\ \vdots & & & \vdots \\ d_na_{n1} & d_na_{n2} & \dots & d_na_{nm} \end{pmatrix} = \begin{pmatrix} d_1(\mathbf{A})_{1,:} \\ \vdots \\ d_n(\mathbf{A})_{n,:} \end{pmatrix} .$$

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

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & & a_{2m} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \begin{pmatrix} d_1 & 0 & & 0 \\ 0 & d_2 & & 0 \\ & & \ddots & \\ 0 & 0 & & d_m \end{pmatrix} = \begin{pmatrix} d_1a_{11} & d_2a_{12} & \dots & d_ma_{1m} \\ d_1a_{21} & d_2a_{22} & \dots & d_ma_{2m} \\ \vdots & & & \vdots \\ d_1a_{n1} & d_2a_{n2} & \dots & d_ma_{nm} \end{pmatrix}$$
$$= \begin{pmatrix} d_1(\mathbf{A})_{:,1} & \dots & d_m(\mathbf{A})_{:,m} \end{pmatrix} .$$

△

Example 1.2.5 (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), $\alpha, \beta \in \mathbb{K}$

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

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

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

Remark 1.2.6 (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* $\mathbf{I}$.

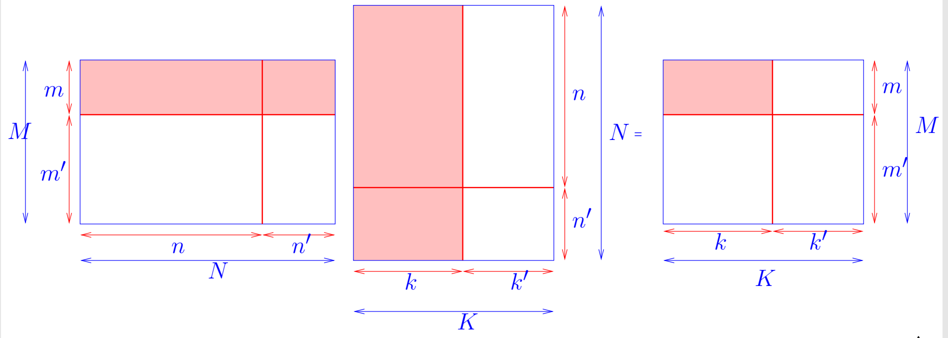
△

Remark 1.2.7 (Block matrix product).

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

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

$$\blacktriangleright \quad \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{pmatrix} . \qquad (1.2.3)$$



△

1.3 Complexity/computational effort

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

additions/multiplications

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

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

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"): $f(n)=O(g(n))\Leftrightarrow\exists C>0,N>0\colon |f(n)|\leq Cg(n)$ 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 [39] 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 (1.2.3) with $\mathbf{A}_{ij},\mathbf{B}_{ij}\in\mathbb{K}^{\ell,\ell}$, $i,j\in\{1,2\}$. Let $\mathbf{C}:=\mathbf{AB}$ be partioned 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 $\mathbf{Q}_k\in\mathbb{K}^{\ell,\ell}$, $k=1,\dots,7$ are obtained from

$$\begin{aligned}
\mathbf{Q}_0 &= (\mathbf{A}_{11}+\mathbf{A}_{22})*(\mathbf{B}_{11}+\mathbf{B}_{22})\,,\\
\mathbf{Q}_1 &= (\mathbf{A}_{21}+\mathbf{A}_{22})*\mathbf{B}_{11}\,,
\end{aligned}$$

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

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 [7].

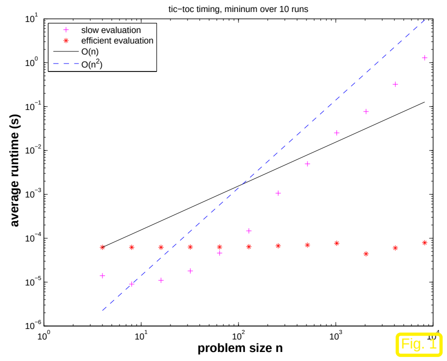
Example 1.3.2 (Efficient associative matrix multiplication).

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

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

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

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



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

Platform:

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

Code 1.3.3: 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
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 1.3.4 (Reading off complexity).

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

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

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

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

➣ quick "visual test" of conjectured asymptotic complexity

More rigorous: Perform linear regression on $(\log n_i, \log t_i)$, $i = 1, \dots, N$ (→ Ch. 6)

△

Remark 1.3.5 (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 [25].

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.

Grouping of BLAS routines ("levels") according to asymptotic complexity, see [18, Sect. 1.1.12]:

- **Level 1**: vector operations such as scalar products and vector norms.
  asymptotic complexity $O(n)$, (with $n \mathrel{\hat{=}}$ 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 $(m, n) \mathrel{\hat{=}}$ matrix size),
  e.g.: matrix×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 $(n, m, k) \mathrel{\hat{=}}$ matrix sizes),
  e.g.: matrix product: $\mathbf{C} = \mathbf{A}\mathbf{B}$

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

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

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

    - $\mathrm{x} \in \{\mathrm{S},\mathrm{D}\}$, 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)`

    - $\mathrm{x} \in \{\mathrm{S},\mathrm{D},\mathrm{C},\mathrm{Z}\}$, 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$\times$vector multiplication $\mathbf{y} = \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$

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

    - $\mathrm{x} \in \{\mathrm{S},\mathrm{D},\mathrm{C},\mathrm{Z}\}$, scalar type: S $\hat{=}$ type `float`, D $\hat{=}$ type `double`, C $\hat{=}$ type `complex`
    - $M, N \hat{=}$ 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.
    - $X \hat{=}$ vector $\mathbf{x}$: array of type x
    - `INCX` $\hat{=}$ stride for traversing vector X
    - BETA $\hat{=}$ scalar paramter β
    - $Y \hat{=}$ vector $\mathbf{y}$: array of type x
    - `INCY` $\hat{=}$ stride for traversing vector Y
- **BLAS LEVEL 3**: matrix-matrix operations, asymptotic complexity $O(mnk)$, $(m,n,k) \hat{=}$ matrix sizes
  - matrix$\times$matrix multiplication $\mathbf{C} = \alpha\mathbf{A}\mathbf{B} + \beta\mathbf{C}$

```
xGEMM(TRANSA,TRANSB,M,N,K,
      ALPHA,A,LDA,X,B,LDB,
      BETA,C,LDC)
```

(☞ meaning of arguments as above)

Example 1.4.1 (Gaining efficiency through use of BLAS).

Code 1.4.2: ColumnMajor Matrix Class in C++, Ex. 1.4.1

```
/*  Author: Manfred Quack
 *  ColumnMajorMatrix.h
 *  This Class Implements a ColumnMajor Matrix Structure in C++
 *  - it provides an access operator () using ColumnMajor Access
 *  - it also provides 4 different implementations of a Matrix-Matrix
    Multiplication
 */
#include <iostream>
#include <cstdlib>
#include <stdio.h>
#include <cassert>
#ifndef _USE_MKL
#ifdef _MAC_OS
#include <vecLib/cblas.h>  //part of Accelerate Framework
#endif
#ifdef _LINUX
extern "C" {
#include <cblas.h>
}
#endif
#else
#include <mkl.h>
```

```
#endif
#include <cmath>
typedef double Real;
using namespace std;

class ColumnMajorMatrix {
private:  //Data Members:
        Real* data;
        int n,m;
public:
        //——Class Managment———:
        //Constructors
        ColumnMajorMatrix(int _n,int _m);
        ColumnMajorMatrix(const ColumnMajorMatrix &B);
        //Destructor
        ~ColumnMajorMatrix();
        //Assignment operator
        ColumnMajorMatrix & operator=(const ColumnMajorMatrix &B);
        //Access for ColumnMajorArrangement
        inline Real& operator()(int i, int j)
        { assert(i<n && j<m); return data[n*j+i];}
        //—-Different Implementations for the Multiplication—-/
        // All of these implementations have only been checked for Square Matrices
```

```
        //straigthforward implementation for a Matrix Multiplication
        ColumnMajorMatrix standardMultiply( ColumnMajorMatrix &B);
        //Implementations using DOT-, GEMV- and GEMM from BLAS:
        ColumnMajorMatrix dotMultiply( ColumnMajorMatrix &B);
        ColumnMajorMatrix gemvMultiply( ColumnMajorMatrix &B);
        ColumnMajorMatrix gemmMultiply( ColumnMajorMatrix &B);
        //——Other Functions———:
        //Function to initialize matrix with 1,2,3...
        void initGrow();
        void initRand();
        void print();
        Real CalcErr(const ColumnMajorMatrix &B);
};
```

Code 1.4.3: A straightforward implementation for Matrix Multiplications in C++, Ex. 1.4.1

```
/* ColumnMajorMatrix_Multiplication: straightforward standard
  implementation for a Matrix Multiplication (3 loops) */
#include "ColumnMajorMatrix.h"
ColumnMajorMatrix ColumnMajorMatrix::standardMultiply( ColumnMajorMatrix
  &B)
{
        assert(this->n==B.n && this->m==B.m); // only support square matrices
        ColumnMajorMatrix C(n,m); //important: must be zero: (done in constructor)
```

```
        for (int j=0;j<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 1.4.5: Matrix Multiplication using DOT from BLAS and two nested loop, parameters are described above. Ex. 1.4.1

```
/* ColumnMajorMatrix_Multiplication.cpp using the DOT-routine from BLAS
  (and 2 loops) */
#include "ColumnMajorMatrix.h"
ColumnMajorMatrix ColumnMajorMatrix::dotMultiply( ColumnMajorMatrix &B)
{
        assert(this->n==B.n && this->m==B.m); // only support square matrices
        ColumnMajorMatrix C(n,m);
        for (int j=0;j<m;++j)
                for(int i=0;i<n;++i)
                        C(i,j)=cblas_ddot(this->m, &(operator()(i,0)) ,
                          n, &(B(0,j)), 1);
        return C;
}
```

Code 1.4.7: 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. Ex. 1.4.1

```
/* ColumnMajorMatrix_Multiplication using the GEMV-routine from BLAS (+1
  loop) */
#include "ColumnMajorMatrix.h"
ColumnMajorMatrix ColumnMajorMatrix::gemvMultiply( ColumnMajorMatrix &B)
{
        assert(this->n==B.n && this->m==B.m); // only support square matrices
        ColumnMajorMatrix C(n,m); //important: must be zero: (done in constructor)
        double alpha(1.0), beta(1.0);
        for (int j=0;j<m;++j)
                cblas_dgemv(CblasColMajor, CblasNoTrans, m, n, alpha,
                  data, n, &(B(0,j)),1, beta,&C(0,j),1);
        return C;
}
```

Code 1.4.9: 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. , Ex. 1.4.1

```
/* ColumnMajorMatrix_Multiplication using the GEMM-routine from BLAS */
#include "ColumnMajorMatrix.h"
ColumnMajorMatrix ColumnMajorMatrix::gemmMultiply( ColumnMajorMatrix &B)
```

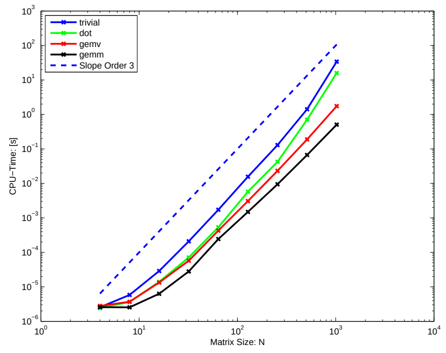
```
{
        assert(this->n==B.n && this->m==B.m);  // only support square matrices
        ColumnMajorMatrix C(n,m); //important: must be zero: (done in constructor)
        double alpha(1.0),beta(1.0);
        cblas_dgemm ( CblasColMajor, CblasNoTrans, CblasNoTrans, n, m,
          B.m, alpha, data, n, B.data, B.n, beta, C.data, C.n );
        return C;
}
```

Code 1.4.10: Timings of different Matrix Multiplications in C++, Ex. 1.4.1

```
#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:
        ColumnMajorMatrix C(n,n),D(n,n),E(n,n),F(n,n);
    //loop for repetitions (always take timing results over several measurements!)
        for (int r=0;r<rep;++r)
        {
        watch.start(); C=A.standardMultiply(B);
          T0=std::min(T0,watch.getTime()); watch.reset();
        watch.start(); D=A.dotMultiply(B);
          T1=std::min(T1,watch.getTime()); watch.reset();
        watch.start(); E=A.gemvMultiply(B);
          T2=std::min(T2,watch.getTime()); watch.reset();
        watch.start(); F=A.gemmMultiply(B);
          T3=std::min(T3,watch.getTime()); watch.reset();
        }
        printf("Timing Results: (min. of : %i Repetitions) \n",rep);
        printf("N: %i StraightForward: %g \n",n,T0);
```

```
        printf("N: %i dotMultiply: %g , error: %g \n",n,T1,D.CalcErr(C));
        printf("N: %i gemvMultiply: %g, error: %g \n",n,T2,E.CalcErr(C));
        printf("N: %i gemmMultiply: %g, error: %g \n",n,T3,F.CalcErr(C));
}
```



◁ 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 1.4.11: Measuring CPU time from C++ using clock command

```
#include <iostream>
#include <time.h>  //header for clock()

/*
 * simple Timer Class, using clock()-command from the time.h (should work
   on all platforms)
 * this class will only report the cputime (not walltime)
 */
class simpleTimer {
public:
        simpleTimer():time(0),bStarted(false)
        {}
        void start()
        {
                time=clock();
                bStarted=true;
        }
        double getTime()
        {
                assert(bStarted);
                return(clock()-time)/(double)CLOCKS_PER_SEC;;
        }
```

```
        void reset()
        {
                time=0;
                bStarted=false;
        }
private:
        double time;
        bool bStarted;
};
```

Code 1.4.12: 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;
                rtime=((double)(rt1-rt0))/ CLOCKS_PER_SEC*10000;
                bStarted=false;
                return rtime;
        }

        double user() { assert(!bStarted); return utime;}
        double system(){assert(!bStarted); return stime;}
        double real(){assert(!bStarted); return rtime;}

private:
        double utime,stime,rtime;
        tms t0;
        long  rt0;
        bool  bStarted;
};
```

2 Direct Methods for Linear Systems of Equations

The fundamental task:

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

Sought : solution vector $\mathbf{x} \in \mathbb{K}^n$: $\boxed{\mathbf{A}\mathbf{x} = \mathbf{b}}$ $\longleftarrow$ (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
- in the case of *discretized* boundary value problems for ordinary and partial differential equations (→ course "Numerical methods for partial differential equations")
- as a result of linearization (e.g, "Newton's method" → Sect. 3.4)

Example 2.0.1 (Nodal analysis (*ger.:* *Knotenanalyse*) of (linear) electric circuit (*ger.:* *elektrisches Netzwerk*)).

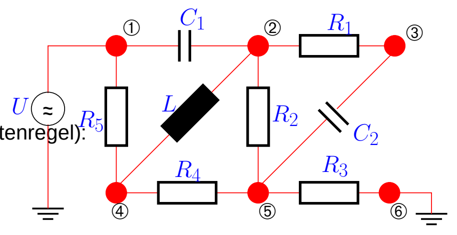
Node (*ger.:* Knoten) $\hat{=}$ junction of wires

☞ *number* nodes $1, \ldots, n$

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

Kirchhoff current law (KCL, *ger.:* Kirchhoffsche Knotenregel):
sum of node currents $= 0$:

$$\forall k \in \{1, \ldots, n\}: \quad \sum_{j=1}^{n} I_{kj} = 0 \ . \qquad (2.0.1)$$



Unknowns: nodal potentials U_k, $k = 1, \ldots, n$.
(some may be known: grounded nodes, voltage sources)

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

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

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

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

$$\text{voltage:} \quad u(t) = \mathrm{Re}\{U \exp(i\omega t)\} \quad , \quad \text{current:} \quad i(t) = \mathrm{Re}\{I \exp(\omega t)\} \ . \qquad (2.0.2)$$

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

$$\dot{u}(t) = \mathrm{Re}\{i\omega U \exp(i\omega t)\} \quad , \quad \dot{i}(t) = \mathrm{Re}\{i\omega I \exp(i\omega t)\} \ . \qquad (2.0.3)$$

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 (2.0.2) and (2.0.3) this leads to the above constitutive relations.

Constitutive relations **+** (2.0.1) ▶ linear system of equations:

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

▼

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

◇

Theory

Known from linear algebra:

Definition 2.0.1 (Invertible matrix).

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

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

Definition 2.0.2 (Rank of a matrix).
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* $\mathbf{M}$.

Theorem 2.0.3 (Criteria for invertibility of matrix)**.**
A matrix $\mathbf{A} \in \mathbb{K}^{n,n}$ *is invertible/regular if one of the following equivalent conditions is satisfied:*

1. $\exists \mathbf{B} \in \mathbb{K}^{n,n}$: $\quad \mathbf{BA} = \mathbf{AB} = \mathbf{I}$,
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

2.1 Gaussian Elimination

! Exceptional feature of linear systems of equations (LSE):
☞ "exact" solution computable with finitely many elementary operations

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

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 *row transformations*

Ex. 1.2.5: row transformations ↔ left-multiplication with transformation matrix

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

$$\mathbf{A}\mathbf{x}=\mathbf{b}\quad\Rightarrow\quad\mathbf{A}'\mathbf{x}=\mathbf{b}'\ \text{, if}\quad\mathbf{A}'=\mathbf{T}\mathbf{A},\ \mathbf{b}'=\mathbf{T}\mathbf{b}\,.$$

Example 2.1.1 (Gaussian elimination).

① (Forward) elimination:

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

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

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

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

▶ transformation of LSE to upper triangular form

② Solve by back substitution:

$$\begin{array}{rcrcrcr}x_1&+&x_2&&&=&4\\&-&x_2&-&x_3&=&-7\\&&&&3x_3&=&13\end{array}\quad\Rightarrow\quad\begin{array}{l}x_3=\frac{13}{3}\\x_2=7-\frac{13}{3}=\frac{8}{3}\\x_1=4-\frac{8}{3}=\frac{4}{3}\,.\end{array}$$

◇

More general:

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

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

$$\begin{array}{ccccccccc}a_{11}x_1&+&a_{12}x_2&+&\cdots&+&a_{1n}x_n&=&b_1\\&&a_{22}^{(1)}x_2&+&\cdots&+&a_{2n}^{(1)}x_n&=&b_2^{(1)}\\&&\vdots&&\vdots&&\vdots&&\vdots\\&&\vdots&&\vdots&&\vdots&&\vdots\\&&a_{n2}^{(1)}x_2&+&\cdots&+&a_{nn}^{(1)}x_n&=&b_n^{(1)}\end{array}\quad\text{with}\quad\begin{array}{rcll}a_{ij}^{(1)}&=&a_{ij}-a_{1j}l_{i1}, & i,j=2,\dots,n\,,\\b_i^{(1)}&=&b_i-b_1l_{i1}, & i=2,\dots,n\,.\end{array}$$

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

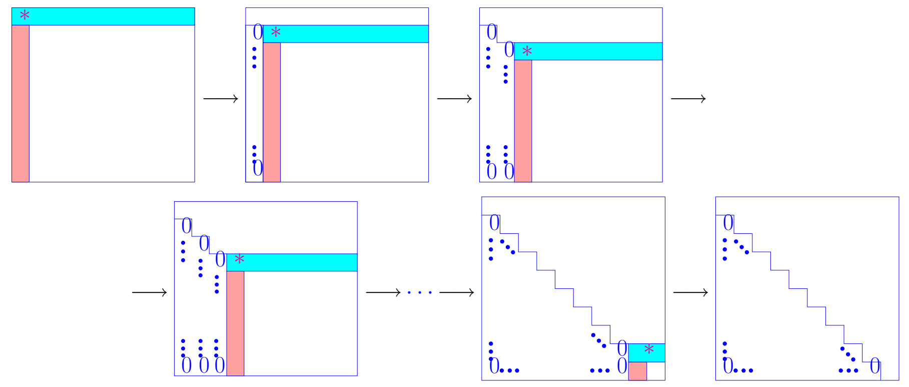
$$\begin{array}{ccccccccccc}a_{11}x_1&+&a_{12}x_2&+&a_{13}x_3&+&\cdots&+&a_{1n}x_n&=&b_1\\&&a_{22}^{(1)}x_2&+&a_{23}^{(1)}x_3&+&\cdots&+&a_{2n}^{(1)}x_n&=&b_2^{(1)}\\&&&&a_{33}^{(2)}x_3&+&\cdots&+&a_{3n}^{(2)}x_n&=&b_3^{(2)}\\&&&&\vdots&&\vdots&&\vdots&&\vdots\\&&&&a_{n3}^{(2)}x_3&+&\cdots&+&a_{nn}^{(2)}x_n&=&b_n^{(2)}\end{array}$$

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

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

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

Graphical depiction:



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

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

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

with

$$a'_{ij} := \begin{cases} a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj} & \text{for } k < i,j \le n\,, \\ 0 & \text{for } k < i \le n, j = k\,, \\ a_{ij} & \text{else,} \end{cases} \qquad b'_i := \begin{cases} b_i - \frac{a_{ik}}{a_{kk}} b_k & \text{for } k < i \le n\,, \\ b_i & \text{else.} \end{cases} \tag{2.1.1}$$

multipliers l_{ik}

Algorithm 2.1.2.
C++-code snippet:

In place (in-situ) implementation of Gaussian elimination for LSE $\mathbf{A}\mathbf{x} = \mathbf{b}$

Never implement Gaussian elimination yourself **!**

use numerical libraries (LAPACK) or MATLAB **!**

MATLAB operator: \

```
template<class Matrix,class Vector>
linsolve(Matrix &A,Vector &b) {
  int n = A.dim();
  for(int i=1;i<n;i++) {
    double pivot = A(i,i);
    for(int k=i+1;k<=n;i++) {
      double mult = A(k,i)/pivot;  (= l_ki)
      for(int l=i+1;l<=n;l++)
        A(k,l) -= mult*A(i,l);
      b(k) -= mult*b(i);  }}
  b(n) /= A(n,n);
  for(int i=n-1;i>0;i-) {
    for(int l=i+1;l<=n;l++)
      b(i) -= b(l)*A(i,l);
    b(i) /= A(i,i);
  }}
```

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

$$\begin{aligned} \text{elimination :} \quad & \textstyle\sum_{i=1}^{n-1} (n-i)(2(n-i)+3) = n(n-1)(\frac{2}{3}n + \frac{7}{6}) \text{ Ops.}\,, \\ \text{back substitution :} \quad & \textstyle\sum_{i=1}^{n} 2(n-i)+1 = n^2 \text{ Ops.}\,. \end{aligned} \tag{2.1.2}$$

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

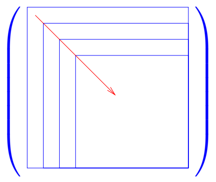
Remark 2.1.3 (Gaussian elimination via rank-1 modifications).

Block perspective (first step of Gaussian elimination with pivot $\alpha \neq 0$), *cf.* (2.1.1):

$$\mathbf{A} := \left(\begin{array}{c|c} \alpha & \mathbf{c}^T \\ \hline \mathbf{d} & \mathbf{C} \end{array}\right) \rightarrow \mathbf{A}' := \left(\begin{array}{c|c} \alpha & \mathbf{c}^T \\ \hline \mathbf{0} & \mathbf{C}' := \mathbf{C} - \frac{\mathbf{d}\mathbf{c}^T}{\alpha} \end{array}\right)\,. \tag{2.1.3}$$

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

repeat this step:



r.h.s. $\mathbf{b} \sim$ `A(:,end)`

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

△

Remark 2.1.4 (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 $\mathbf{b} \in \mathbb{K}^n$, $\mathbf{b}_1 = (\mathbf{b})_{1:k}$, $\mathbf{b}_2 = (\mathbf{b})_{k+1:n}$

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

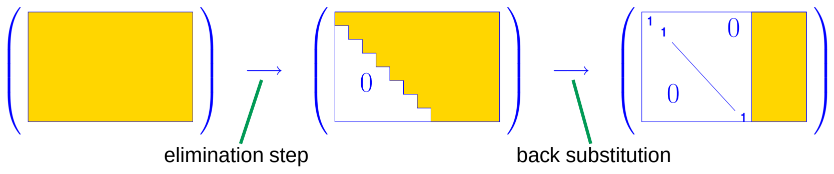
where $\mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$ (Schur complement, see Rem. 2.2.8), $\mathbf{b}_S := \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1$

❶: elimination step, ❷: backsubstitution step

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

Remark 2.1.5 (Gaussian elimination for non-square matrices).

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



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

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

MATLAB:

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

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

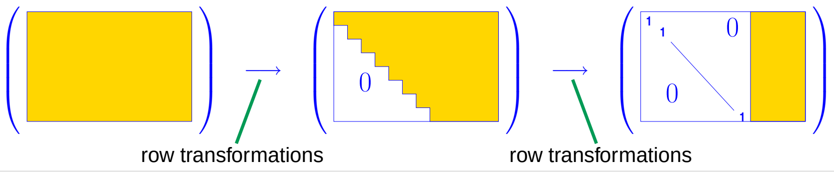
Code 2.1.6: Gaussian elimination with multiple r.h.s.

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

△

2.2 LU-Decomposition/LU-Factorization

The gist of Gaussian elimination:



row transformation = adding a multiple of a matrix row to another row, or
multiplying a row with a non-zero scalar (number)
(*ger.:* Zeilenumformung)

Note: row transformations preserve regularity of a matrix (why ?)

A matrix factorization (*ger.* Matrixzerlegung) writes 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

Example 2.2.1 (Gaussian elimination and LU-factorization).

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

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

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

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

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

Perspective: link Gaussian elimination to matrix factorization $\longrightarrow$ Ex. 2.2.1

(row transformation = multiplication with elimination matrix)

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

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

$$\mathbf{A} = \mathbf{L}_1 \cdot \cdots \cdot \mathbf{L}_{n-1}\mathbf{U}$$ with elimination matrices $\mathbf{L}_i$, $i = 1, \dots, n-1$, upper triangular matrix $\mathbf{U} \in \mathbb{R}^{n,n}$.

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

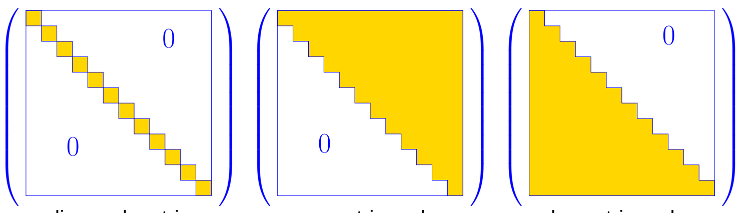
▶ $\mathbf{L}_1 \cdot \cdots \cdot \mathbf{L}_{n-1}$ are normalized lower triangular matrices
(entries = multipliers $-\frac{a_{ik}}{a_{kk}}$ from (2.1.1) $\rightarrow$ Ex. 2.1.1)

2.2
p. 85

Definition 2.2.1 (Types of matrices).
A matrix $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m,n}$ *is*

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

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



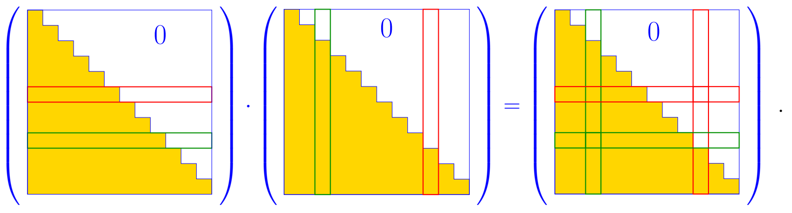
diagonal matrix upper triangular lower triangular

2.2
p. 86

Lemma 2.2.2 (Group of regular diagonal/triangular matrices).

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

(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 $\mathbf{U}$.

2.2
p. 87

Lemma 2.2.3 (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

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

Then solve for $\mathbf{x}, \mathbf{y}$ and $\xi \in \mathbb{K}$. Regularity of $\mathbf{A}$ involves $\xi \neq 0$ so that $\mathbf{U}$ will be regular, too.

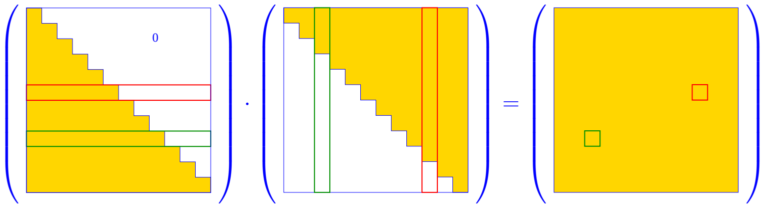
Remark: *Uniqueness* of LU-decomposition:

Regular upper triangular matrices and normalized lower triangular matrices form *matrix groups*. Their only common element is the identity matrix.

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

2.2
p. 88

A direct way to LU-decomposition:



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

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

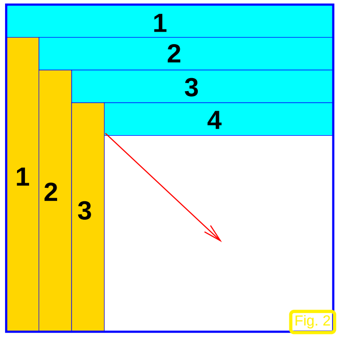
• column by column computation of $\mathbf{L}$

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

(Crout's algorithm)

🟦 $\hat{=}$ rows of $\mathbf{U}$

🟨 $\hat{=}$ columns of $\mathbf{L}$



LU-factorization = "inversion" of matrix multiplication:

Code 2.2.2: LU-factorization

```
function [L,U] = lufak(A)
% LU-factorization of A ∈ K^{n,n}
n = size(A,1); if (size(A,2) ~= n),
   error('n ~= m'); end
L = eye(n); U = zeros(n,n);
for k=1:n
    for j=k:n
        U(k,j) = A(k,j) -
          L(k,1:k-1)*U(1:k-1,j);
    end
    for i=k+1:n
        L(i,k) = (A(i,k) -
          L(i,1:k-1)*U(1:k-1,k))
          /U(k,k);
    end
end
```

Code 2.2.3: matrix multiplication $\mathbf{L}\cdot\mathbf{U}$

```
function A = lumult(L,U)
% Multiplication of normalized lower/upper
  triangular matrices
n = size(L,1); A = zeros(n,n);
if ((size(L,2) ~= n) || (size(U,1)
  ~= n) || (size(U,2) ~= n))
    error('size mismatch'); end
for k=n:-1:1
    for j=k:n
        A(k,j) = U(k,j) +
          L(k,1:k-1)*U(1:k-1,j);
    end
    for i=k+1:n
        A(i,k) =
          L(i,1:k-1)*U(1:k-1,k) +
          L(i,k)*U(k,k);
    end
end
```

asymptotic complexity of LU-factorization of $\mathbf{A}\in\mathbb{R}^{n,n}$ = $\frac{1}{3}n^3 + O(n^2)$ (2.2.1)

Remark 2.2.4 (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 2.2.5 (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 2.2.6 (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 !

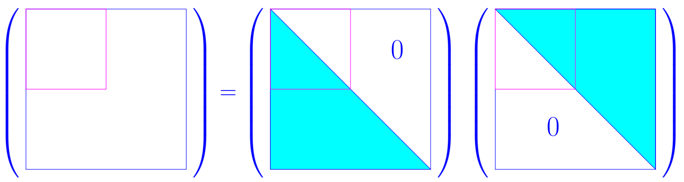
```
%Setting: N ≫ 1, large matrix A
[L,U] = lu(A);
for j=1:N
    x = U\(L\b);
    b = some_function(x);
end
```

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

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

△

Remark 2.2.7 ("'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 2.2.8. Block matrix multiplication (1.2.3) $\cong$ block LU-decomposition:

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

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & 0 \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{S} \end{pmatrix} \quad , \quad \overset{\text{Schur complement}}{\mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}} \tag{2.2.2}$$

$\rightarrow$ block Gaussian elimination, see Rem. 2.1.4.

△

2.3 Pivoting

Known from linear algebra:

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

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

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

Example 2.3.1 (Pivoting and numerical stability).

```
%Example: numerical instability without pivoting
A = [5.0E-17 , 1; 1 , 1];
b = [1;2];
x1 = A\b,
x2 =gausselim(A,b), %see Code 2.1.5
[L,U] = lufak(A); %see Code 2.2.1
z = L\b; x3 = U\z,
```

Ouput of MATLAB run:

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

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

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

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

$$► \quad \mathbf{L} = \begin{pmatrix} 1 & 0 \\ \epsilon^{-1} & 1 \end{pmatrix} \quad , \quad \mathbf{U} = \begin{pmatrix} \epsilon & 1 \\ 0 & 1-\epsilon^{-1} \end{pmatrix} \overset{(*)}{=} \widetilde{\mathbf{U}} := \begin{pmatrix} \epsilon & 1 \\ 0 & -\epsilon^{-1} \end{pmatrix} \quad \text{in } \mathbb{M}\,! \tag{2.3.1}$$

$(*)$: because $1 \widetilde{+} 2/\mathtt{eps} = 2/\mathtt{eps}$, see Rem. 2.4.9.

▶ 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:

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix} \Rightarrow \mathbf{L} = \begin{pmatrix} 1 & 0 \\ \epsilon & 1 \end{pmatrix}, \quad \mathbf{U} = \begin{pmatrix} 1 & 1 \\ 0 & 1-\epsilon \end{pmatrix} = \widetilde{\mathbf{U}} := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{in } \mathbb{M}. \tag{2.3.2}$$

▶ 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, $\rightarrow$ (2.3.1): $\mathbf{L}\widetilde{\mathbf{U}} = \mathbf{A} + \mathbf{E}$ with $\mathbf{E} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$ ▶ unstable !

after row swapping, $\rightarrow$ (2.3.2): $\mathbf{L}\widetilde{\mathbf{U}} = \widetilde{\mathbf{A}} + \mathbf{E}$ 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)

Example 2.3.2 (Gaussian elimination with pivoting for 3×3-matrix).

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

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 2.3.3.

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

Code 2.3.4: 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)), error('Nearly␣Singular␣matrix'); 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:

$$j \in \{i,\dots,n\} \text{ such that } \frac{|a_{ki}|}{\sum_{l=i}^{n} |a_{kl}|} \rightarrow \max \tag{2.3.3}$$

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

Explanations to Code 2.3.3:

Line 4: Find the relatively largest pivot element p and the index j of the corresponding row of the matrix.

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. 2.1.3 and Rem. 2.2.4.

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 2.3.5.

C++-code for in-situ LU-factorization of $\mathbf{A} \in \mathbb{R}^{n,n}$ with partial pivoting ➤

Row permutations recorded in vector p !

```
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);
      }}}}
```

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

$$\frac{|a_{ki}|}{\sum_{l=i}^{n} |a_{kl}|} \rightarrow \max \tag{2.3.4}$$

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

Why *relatively* largest pivot element in (2.3.4)? ➤ 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:

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

```
%Example: importance of scale-invariant pivoting
epsilon = 5.0E-17;
A = [epsilon , 1; 1 , 1]; b = [1;2];
D = [1/epsilon, 0; 0 ,1];
A = D*A; b = D*b;
x1 = A\b,
x2 =gausselim(A,b), %see Code 2.1.5
[L,U] = lufak(A); %see Code 2.2.1
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 2.3.1 (Permutation matrix)**.**
An n-*permutation*, $n \in \mathbb{N}$, *is a bijective mapping* $\pi : \{1,\dots,n\} \mapsto \{1,\dots,n\}$. *The corresponding permutation matrix* $\mathbf{P}_\pi \in \mathbb{K}^{n,n}$ *is defined by*

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

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

Note:
- $\mathbf{P}^H = \mathbf{P}^{-1}$ for any permutation matrix $\mathbf{P}$ ($\rightarrow$ permutation matrices are 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 2.3.2 (Existence of LU-factorization with pivoting)**.**
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. 2.2.1), 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 2.3.6 (Ex. 2.3.2 cnt'd).

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

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

◇

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

Remark 2.3.7 (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 2.2.1), 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

A very detailed exposition and in-depth discussion of the all the material in this section can be found in [24]:

- [24, Ch. 1]: excellent collection of examples concerning the impact of roundoff errors.
- [24, Ch. 2]: floating point arithmetic, see Def. 2.4.1 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!

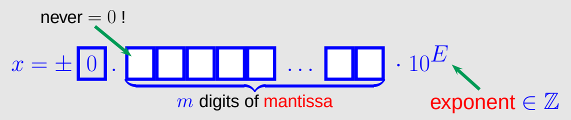
"Computers use floating point numbers (scientific notation)"

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:

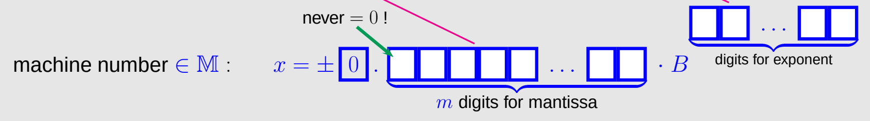


◇

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

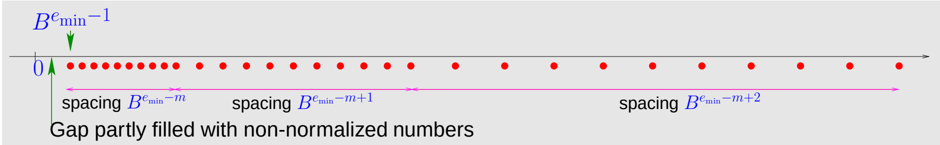
Definition 2.4.1 (Machine numbers)**.**
Given ☞ basis $B \in \mathbb{N} \setminus \{1\}$,
☞ exponent *range* $\{e_{\min}, \ldots, e_{\max}\}$, $e_{\min}, e_{\max} \in \mathbb{Z}$, $e_{\min} < e_{\max}$,
☞ *number* $m \in \mathbb{N}$ *of digits (for* mantissa*),*
the corresponding set of machine numbers *is*

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


never = 0 !
machine number $\in \mathbb{M}$:
$x = \pm 0 . \ldots \cdot B$
m digits for mantissa
digits for exponent

➤ 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:


$B^{e_{\min}-1}$
0
spacing $B^{e_{\min}-m}$
spacing $B^{e_{\min}-m+1}$
spacing $B^{e_{\min}-m+2}$
Gap partly filled with non-normalized numbers

Example 2.4.2 (IEEE standard 754 for machine numbers). $\longrightarrow$ [30], $\longrightarrow$ link

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

single precision : $m = 24^*, E \in \{-125, \dots, 128\}$ ➣ 4 bytes
double precision : $m = 53^*, E \in \{-1021, \dots, 1024\}$ ➣ 8 bytes

$*$: including bit indicating sign

◇

Remark 2.4.3 (Special cases in IEEE standard).

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

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

△

Example 2.4.4 (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 2.4.5 (Input errors and roundoff errors).

Code 2.4.6: 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:

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

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

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

$$\exists \mathtt{eps} \ll 1: \quad \frac{|\mathrm{rd}(x) - x|}{|x|} \leq \mathtt{eps} \quad \forall x \in \mathbb{R} . \tag{2.4.1}$$

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

$$\star \in \{+, -, \cdot, /\}: \qquad x \,\widetilde{\star}\, y := \mathrm{rd}(x \star y) \tag{2.4.2}$$

Assumption 2.4.2 ("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** $f \in \{\exp, \sin, \cos, \log, \dots\}$ *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 $|\delta| < \mathtt{eps}$.

$*$: 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 2.4.7 (Machine precision for MATLAB). (CPU Intel Pentium)

Code 2.4.8: Finding out eps in MATLAB

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

◇

Remark 2.4.9 (Adding eps to 1).

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

Code 2.4.10: $1+\epsilon$ in MATLAB

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

In fact $1 \widetilde{+} \texttt{eps} = 1$ would comply with the “axiom” of roundoff error analysis, Ass. 2.4.2:

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

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

△

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

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) $\forall \mathbf{x} \in X$: $\mathbf{x} \neq 0 \;\Leftrightarrow\; \|\mathbf{x}\| > 0$ *(definite),*

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

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

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

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

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

$$\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1 \leq \sqrt{n}\,\|\mathbf{x}\|_2\ , \tag{2.5.1}$$

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{n}\,\|\mathbf{x}\|_\infty\ , \tag{2.5.2}$$

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_1 \leq n\,\|\mathbf{x}\|_\infty\ . \tag{2.5.3}$$

Definition 2.5.2 (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: $\|\mathbf{x}\|_2 \to \|\mathbf{M}\|_2\ ,\quad \|\mathbf{x}\|_1 \to \|\mathbf{M}\|_1\ ,\quad \|\mathbf{x}\|_\infty \to \|\mathbf{M}\|_\infty$

Example 2.5.1 (Matrix norm associated with ∞-norm and 1-norm).

e.g. for $\mathbf{M} \in \mathbb{K}^{2,2}$:
$$\begin{aligned}\|\mathbf{M}\mathbf{x}\|_\infty &= \max\{|m_{11}x_1 + m_{12}x_2|, |m_{21}x_1 + m_{22}x_2|\} \\ &\leq \max\{|m_{11}| + |m_{12}|, |m_{21}| + |m_{22}|\}\ \|x\|_\infty\ , \\ \|\mathbf{M}\mathbf{x}\|_1 &= |m_{11}x_1 + m_{12}x_2| + |m_{21}x_1 + m_{22}x_2| \\ &\leq \max\{|m_{11}| + |m_{21}|, |m_{12}| + |m_{22}|\}(|x_1| + |x_2|)\ .\end{aligned}$$

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

➢ matrix norm $\leftrightarrow \|\cdot\|_\infty$ = row sum norm
$$\|\mathbf{M}\|_\infty := \max_{i=1,\dots,m} \sum_{j=1}^{n} |m_{ij}|\ , \qquad (2.5.4)$$

➢ matrix norm $\leftrightarrow \|\cdot\|_1$ = column sum norm
$$\|\mathbf{M}\|_1 := \max_{j=1,\dots,n} \sum_{i=1}^{m} |m_{ij}|\ . \qquad (2.5.5)$$

◇

Special formulas for Euclidean matrix norm [18, Sect. 2.3.3]:

Lemma 2.5.3 (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:

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

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

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

On the other hand, for the same reason:

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

□

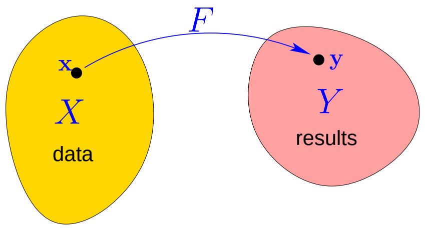
Corollary 2.5.4 (2-Matrixnorm and eigenvalues).
For $\mathbf{A} \in \mathbb{K}^{m,n}$ the 2-Matrixnorm $\|\mathbf{A}\|_2$ is the square root of the largest (in modulus) eigenvalue of $\mathbf{A}^H\mathbf{A}$.

2.5.2 Numerical Stability

Abstract point of view:

Our notion of "problem":

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



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

"The problem:"
- data $\hat{=}$ system matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, right hand side vector $\mathbf{b} \in \mathbb{R}^n$
  ➤ data space $X = \mathbb{R}^{n,n} \times \mathbb{R}^n$ with vector/matrix norms ($\rightarrow$ Defs. 2.5.1, 2.5.2)
- 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 2.5.5 (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:

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

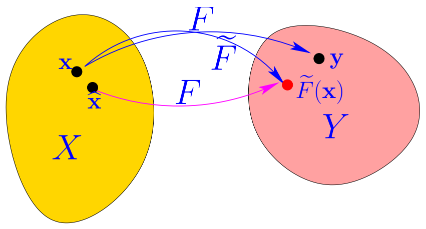
- Judgement about the stability of an algorithm depends on the chosen norms !
- What is meant by $C \approx 1$ in practice ?
  $C \approx 1 \leftrightarrow C \approx$ no. of elementary operations for computing $\widetilde{F}(\mathbf{x})$: The longer a computation takes the more accumulation of roundoff errors we are willing to tolerate.
- The use of computer arithmetic involves inevitable relative input errors ($\rightarrow$ Ex. 2.4.5) 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. 2.5.5 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 2.5.6 (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 2.1.2.*

❷ *LU-factorization of* $\mathbf{A}$ *(*$\rightarrow$ *Code 2.2.1) followed by forward and backward substitution, see Algorithm 2.2.5.*

Rem. 2.3.7 ➢ sufficient to consider LU-factorization without pivoting

A profound roundoff analysis of Gaussian eliminatin/LU-factorization can be found in [18, Sect. 3.3 & 3.5] and [24, Sect. 9.3]. A less rigorous, but more lucid discussion is given in [42, Lecture 22].

Here we only quote a result due to Wilkinson, [24, Thm. 9.5]:

Theorem 2.5.7 (Stability of Gaussian elimination with partial pivoting).
Let $\mathbf{A} \in \mathbb{R}^{n,n}$ *be regular and* $\mathbf{A}^{(k)} \in \mathbb{R}^{n,n}$, $k = 1, \dots, n-1$, *denote the intermediate matrix arising in the* k*-th step of Algorithm 2.3.5 (Gaussian elimination with partial pivoting).*
For the approximate solution $\widetilde{\mathbf{x}} \in \mathbb{R}^{n,n}$ *of the LSE* $\mathbf{Ax} = \mathbf{b}$, $\mathbf{b} \in \mathbb{R}^n$, *computed by Algorithm 2.3.5 (based on machine arithmetic with machine precision* $\mathtt{eps}$, $\rightarrow$ *Ass. 2.4.2) 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. 2.5.5)

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 $O(n^3\mathtt{eps})$).

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

Example 2.5.2 (Wilkinson's counterexample).

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

Partial pivoting does not trigger row permutations !

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

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

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 2.5.8 (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 2.5.3 (Small residuals by Gaussian elimination).

Numerical experiment with *nearly singular matrix*

$$\mathbf{A} = \underbrace{\mathbf{u}\mathbf{v}^T}_{\text{singular rank-1 matrix}} + \epsilon\mathbf{I} ,$$

with

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

Code 2.5.4: 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
```

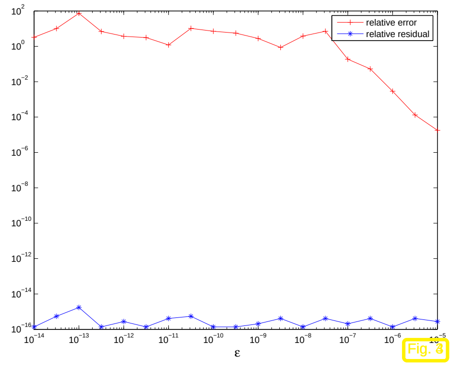


Fig. 3

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 **?**

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

$$\left\{ \begin{array}{ll} \mathbf{A}(\mathbf{x} - \widetilde{\mathbf{x}}) = \mathbf{r} & \Rightarrow \|\mathbf{x} - \widetilde{\mathbf{x}}\| \leq \left\|\mathbf{A}^{-1}\right\| \|\mathbf{r}\| \\ \mathbf{A}\mathbf{x} = \mathbf{b} & \Rightarrow \|\mathbf{b}\| \leq \|\mathbf{A}\| \, \|\mathbf{x}\| \end{array} \right. \quad \Rightarrow \quad \frac{\|\mathbf{x} - \widetilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}\| \left\|\mathbf{A}^{-1}\right\| \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} . \qquad (2.5.6)$$

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

◇

Example 2.5.5 (Instability of multiplication with inverse).

Nearly singular matrix from Ex. 2.5.3

Code 2.5.7: instability of multiplication with inverse

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

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

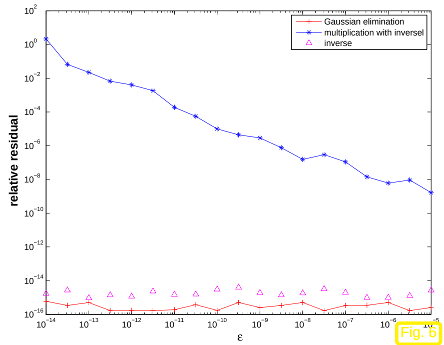


Fig. 6

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.

◇

2.5.4 Conditioning

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

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

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

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

Perturbed linear system:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \;\leftrightarrow\; (\mathbf{A} + \Delta\mathbf{A})\widetilde{\mathbf{x}} = \mathbf{b} + \Delta\mathbf{b} \;\blacktriangleright\; (\mathbf{A} + \Delta\mathbf{A})(\widetilde{\mathbf{x}} - \mathbf{x}) = \Delta\mathbf{b} - \Delta\mathbf{A}\mathbf{x} . \tag{2.5.7}$$

▼

Theorem 2.5.9 (Conditioning of LSEs)**.**
If $\mathbf{A}$ *regular,* $\|\Delta\mathbf{A}\| < \left\|\mathbf{A}^{-1}\right\|^{-1}$ *and* (2.5.7)*, then*

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

The proof is based on the following fundamental result:

Lemma 2.5.10 (Perturbation lemma)**.**

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

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

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

Proof (of Thm. 2.5.9) Lemma 2.5.10 $\succ$ $\left\|(\mathbf{A} + \Delta\mathbf{A})^{-1}\right\| \leq \dfrac{\left\|\mathbf{A}^{-1}\right\|}{1 - \left\|\mathbf{A}^{-1}\Delta\mathbf{A}\right\|}$ **&** (2.5.7)

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

Definition 2.5.11 (Condition (number) of a matrix)**.**
Condition (number) of a matrix $\mathbf{A} \in \mathbb{R}^{n,n}$*:* $\qquad \operatorname{cond}(\mathbf{A}) := \left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\|$

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

Rewriting estimate of Thm. 2.5.9 with $\Delta\mathbf{b} = 0$,

$$\epsilon_r := \frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\operatorname{cond}(\mathbf{A})\delta_A}{1 - \operatorname{cond}(\mathbf{A})\delta_A} \;, \quad \delta_A := \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \;. \tag{2.5.8}$$

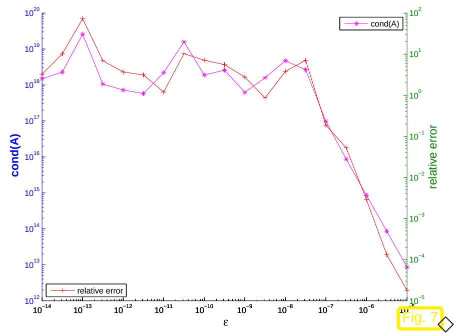
(2.5.8) ➣

- If $\operatorname{cond}(\mathbf{A}) \gg 1$, *small perturbations* in $\mathbf{A}$ can lead to *large relative errors* in the solution of the LSE.
- $\boxed{\text{If } \operatorname{cond}(\mathbf{A}) \gg 1 \text{, a stable algorithm } (\rightarrow \text{Def. 2.5.5}) \text{ can produce solutions with large relative error !}}$

Example 2.5.8 (Conditioning and relative error). $\longrightarrow$ Ex. 2.5.3 cnt'd

Numerical experiment with *nearly singular matrix* from Ex. 2.5.3

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



◇

Example 2.5.9 (Wilkinson's counterexample cnt'd). $\longrightarrow$ Ex. 2.5.2

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

$\updownarrow$ $(*)$

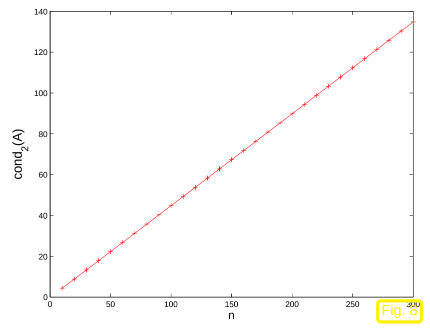
However, $\operatorname{cond}_2(\mathbf{A})$ is small!

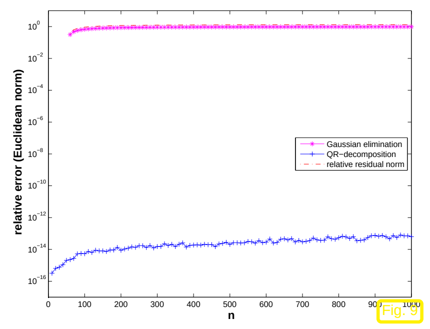
▷ Instability of Gaussian elimination !

Code 2.5.10: 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 $\operatorname{cond}_2(\mathbf{A})$ 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 (2.5.6) and the message of Thm. 2.5.9, (2.5.8). In this case, a stable algorithm can obviously produce a grossly "wrong" solution, as was already explained after (2.5.8).
Hence, lack of stability of Gaussian elimination will only become apparent for linear systems with well-conditioned system matrices.





These observations match Thm. 2.5.7, because in this case we encounter an exponential growth of $\rho = \rho(n)$, see Ex. 2.5.2.

◇

2.5.5 Sensitivity of linear systems

Recall Thm. 2.5.9: for regular $\mathbf{A} \in \mathbb{K}^{n,n}$, small $\Delta\mathbf{A}$, generic vector/matrix norm $\|\cdot\|$

$$\begin{array}{c}\mathbf{A}\mathbf{x} = \mathbf{b} \\ (\mathbf{A}+\Delta\mathbf{A})\tilde{\mathbf{x}} = \mathbf{b}+\Delta\mathbf{b}\end{array} \Rightarrow \frac{\|\mathbf{x}-\tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\operatorname{cond}(\mathbf{A})}{1-\operatorname{cond}(\mathbf{A})\|\Delta\mathbf{A}\|/\|\mathbf{A}\|}\left(\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}+\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}\right) . \tag{2.5.9}$$

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

$\boxed{\text{Sensitivity of a problem (for given data) gauges impact of small perturbations of the data on the result.}}$

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

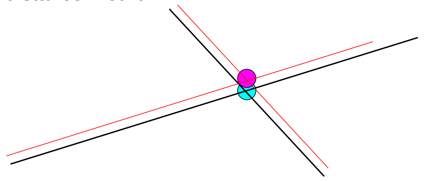
Terminology:

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

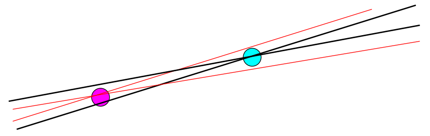
Note: sensitivity gauge depends on the chosen norm **!**

Example 2.5.11 (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: $\underbrace{\begin{pmatrix} \mathbf{n}_1^T \\ \mathbf{n}_2^T \end{pmatrix}}_{=:\mathbf{A}} \mathbf{x} = \underbrace{\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}}_{=:\mathbf{b}}$,

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

Code 2.5.12: 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';
```

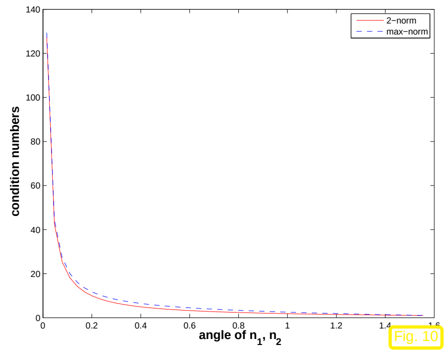


Fig. 10

◇

Heuristics:

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

2.6 Sparse Matrices

A classification of matrices:

Dense matrices (*ger.:* vollbesetzt) ◆ sparse matrices (*ger.:* dünnbesetzt)

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

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

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

A more rigorous “mathematical” definition:

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

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

Simple example: families of diagonal matrices (→ Def. 2.2.1)

Example 2.6.1 (Sparse LSE in circuit modelling).

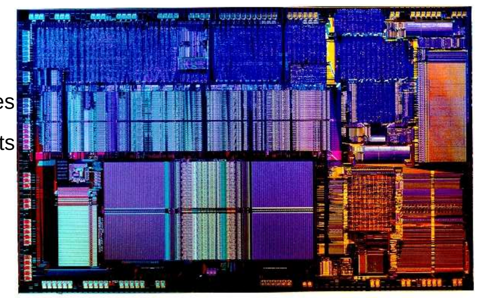
Modern electric circuits (VLSI chips):
$10^5 - 10^7$ circuit elements

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

[In the case of a linear circuit]

▼

nodal analysis ➤ sparse circuit matrix



◇

Another important context in which sparse matrices usually arise:

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

2.6.1 Sparse matrix storage formats

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

- Compressed Row Storage (CRS)
- Compressed Column Storage (CCS) $\longrightarrow$ 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 2.6.2 (Compressed row-storage (CRS) format).

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

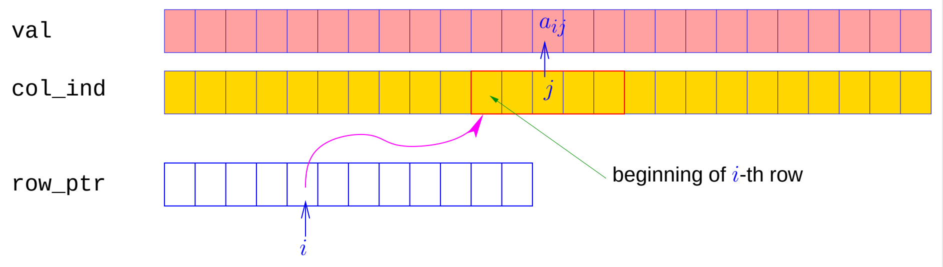
```
      double * val
unsigned int * col_ind
unsigned int * row_ptr
```

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

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

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

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



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

`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

Detailed discussion in [17]

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 2.6.3 (Accessing rows and columns of sparse matrices).

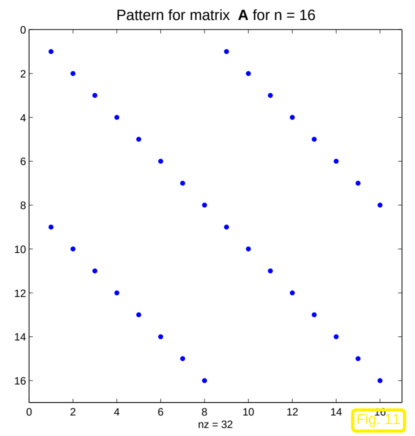


Fig. 11

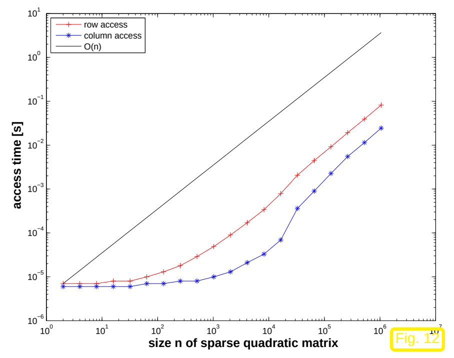


Fig. 12

Code 2.6.4: timing access to rows/columns of a sparse matrix

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

t = [];
```

```
for i=1:20
  n = 2^i; m = n/2;
  A = spdiags(repmat([-1 2 5],n,1),[-n/2,0,n/2],n,n);

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

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

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

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 2.6.5 (Efficient Initialization of sparse matrices in MATLAB).

Code 2.6.6: 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 2.6.7: Initialization of sparse matrices: version II

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

Code 2.6.8: Initialization of sparse matrices: version III

```
dat = zeros(6*n,3); k = 0;
for i=1:n
```

```
     for j=1:n
        if (abs(i-j) == 1), k=k+1; dat(k,:) = [i,j,1.0];
        end;
        if (abs(i-j) == round(n/3))
           k=k+1; dat(k,:) = [i,j,-1.0];
        end;
  end; end;
  A3 = sparse(dat(1:k,1),dat(1:k,2),dat(1:k,3),n,n);
```

Code 2.6.9: Initialization of sparse matrices: driver script

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

loglog(r(:,1),r(:,2),'r*',r(:,1),r(:,3),'m+',r(:,1),r(:,4),'b^');
xlabel('{\bf␣matrix␣size␣n}','fontsize',14);
ylabel('{\bf␣time␣[s]}','fontsize',14);
```

```
legend('Initialization␣I','Initialization␣II','Initialization␣III',...
    'location','northwest');
print -depsc2 '../PICTURES/sparseinit.eps';
```

Timings: ▷

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

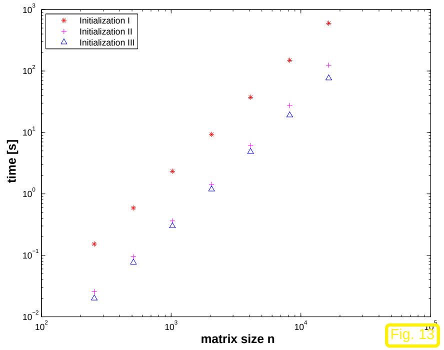


Fig. 13

☛ It is grossly inefficient to initialize a matrix in CCS format (→ Ex. 2.6.2) by setting individual entries one after another, because this usually entails moving large chunks of memory to create space for new non-zero entries.

Instead calls like

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

where

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

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

◇

Example 2.6.10 (Multiplication of sparse matrices).

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

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

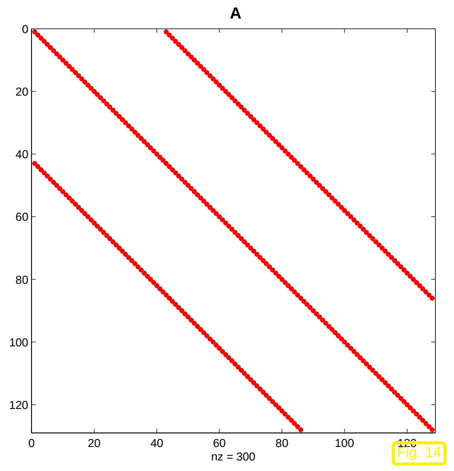


Fig. 14

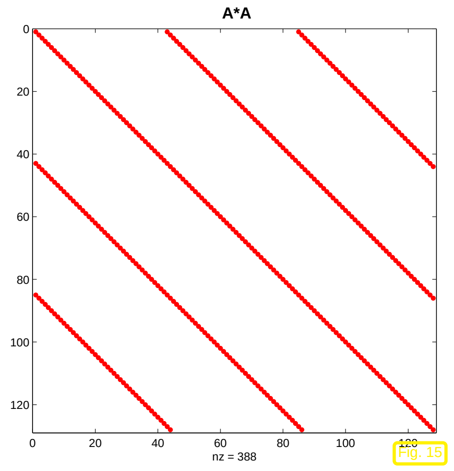


Fig. 15

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

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

(same platform as in Ex. 2.6.5)

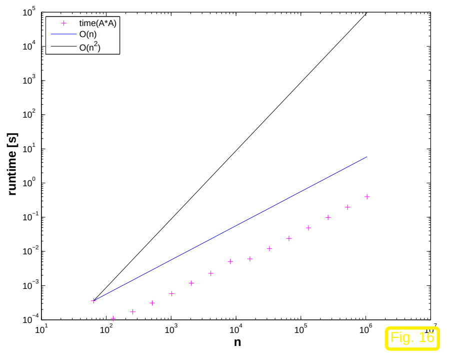


Fig. 16

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

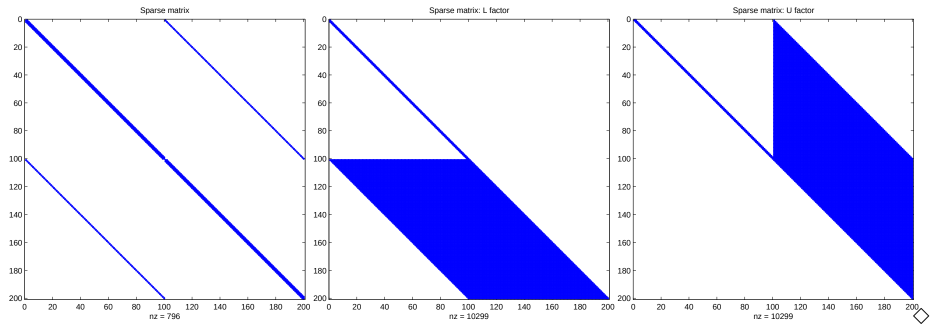
◇

2.6.3 LU-factorization of sparse matrices

Example 2.6.11 (LU-factorization of sparse matrices).

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

Code 2.6.12: 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';
```



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

Definition 2.6.3 (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).

Example 2.6.13 (Sparse LU-factors).

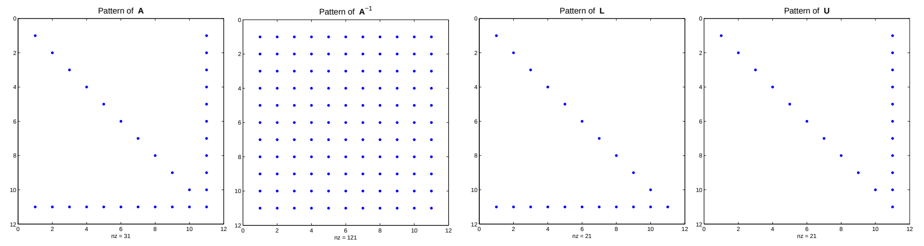
Ex. 2.6.11 ➢ massive fill-in can occur for sparse matrices

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. 2.2.7 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 $\not\Rightarrow$ $\mathbf{A}^{-1}$ sparse **!**

Besides stability issues, see Ex. 2.5.3, this is another reason why using `x = inv(A)*y` instead of `y = A\b` is usually a major blunder.

◇

Example 2.6.14 ("arrow matrix").

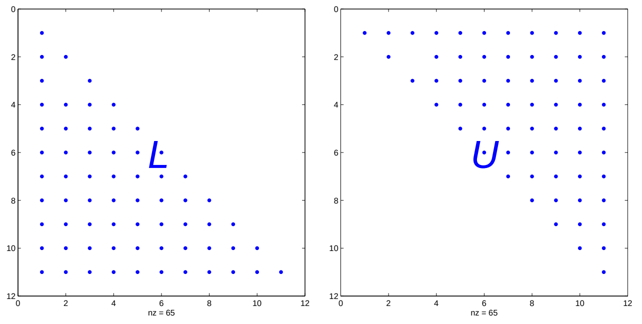
$$\mathbf{A} = \left(\begin{array}{c|c} \alpha & \mathbf{b}^T \\ \hline \mathbf{c} & \mathbf{D} \end{array}\right) , \quad \begin{array}{l} \alpha \in \mathbb{R}\,, \\ \mathbf{b}, \mathbf{c} \in \mathbb{R}^{n-1}\,, \\ \mathbf{D} \in \mathbb{R}^{n-1,n-1} \text{ regular diagonal matrix, } \to \text{Def. 2.2.1} \end{array} \tag{2.6.1}$$

Run algorithm 2.3.5 (Gaussian elimination without pivoting):

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

Code 2.6.15: LU-factorization of arrow matrix
```
n = 10; A = [ n+1, (n:-1:1) ;
  ones(n,1), eye(n,n)];
[L,U,P] = lu(A); spy(L); spy(U);
```

Obvious fill-in (⟶ Def. 2.6.3)



Cyclic permutation of rows/columns:

- 1st row/column ⟶ n-th row/column
- i-th row/column ⟶ $i-1$-th row/column, $i = 2,\dots,n$

➢ LU-factorization requires $O(n)$ operations, see Ex. 2.6.13.

$$\mathbf{A} = \left(\begin{array}{c|c} \mathbf{D} & \mathbf{c} \\ \hline \mathbf{b}^T & \alpha \end{array}\right) \qquad (2.6.2)$$

After permuting rows of $\mathbf{A}$ from (2.6.2) , *cf.* (2.2.2):

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

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

Solving LSE $\mathbf{A}\mathbf{x} = \mathbf{y}$ with $\mathbf{A}$ from 2.6.1: two MATLAB codes

"naive" implementation via "\":

Code 2.6.17: LSE with arrow matrix, implementation I

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

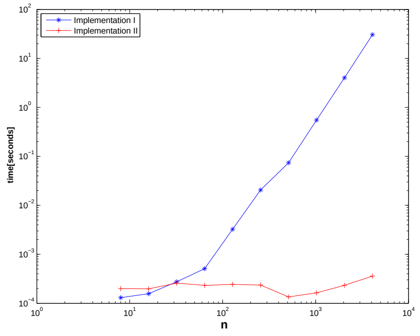
"structure aware" implementation:

Code 2.6.19: LSE with arrow matrix, implementation II

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

Measuring run times:

```
t = [];
for i=3:12
  n = 2^n; alpha = 2;
  b = ones(n,1); c = (1:n)';
  d = -ones(n,1); y = (-1).^(1:(n+1))';
  tic; x1 = sa1(alpha,b,c,d,y); t1 = toc;
  tic; x2 = sa2(alpha,b,c,d,y); t2 = toc;
  t = [t; n t1 t2];
end
loglog(t(:,1),t(:,2), ...
  ...  'b-*',t(:,1),t(:,3),'r-+');
```



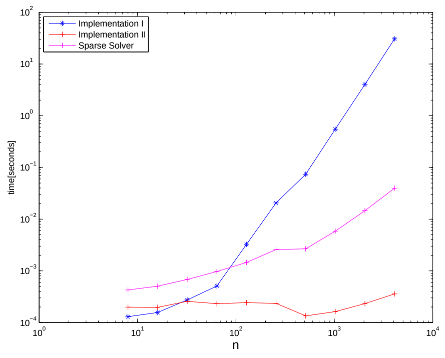
Platform as in Ex. 2.6.5

MATLAB can do much better **!**

Use sparse matrix format:

Code 2.6.20: sparse solver for arrow matrix

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



◇

Exploit structure of (sparse) linear systems of equations !

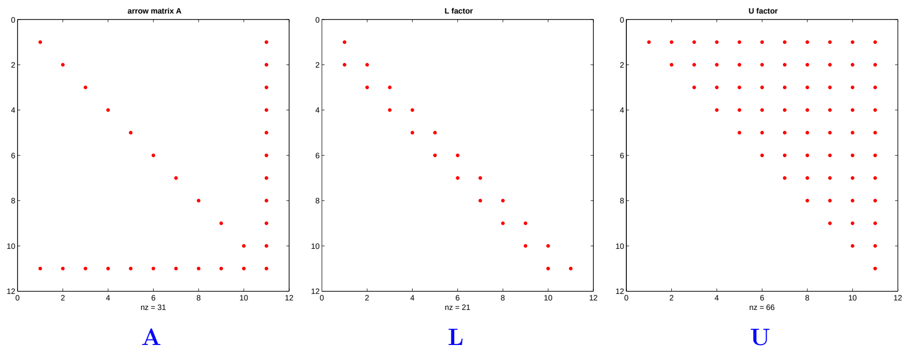
Caution: stability at risk

Example 2.6.21 (Pivoting destroys sparsity).

Code 2.6.22: 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';
```

$$\mathbf{A} = \begin{pmatrix} 1 & & & & 2 \\ & \frac{1}{2} & & & 2 \\ & & \ddots & & \vdots \\ & & & \frac{1}{10} & 2 \\ 2 & & \dots & & 2 \end{pmatrix} \quad \longrightarrow \quad \text{arrow matrix, Ex. 2.6.13}$$



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 $O(n^3)$.

◇

2.6.4 Banded matrices

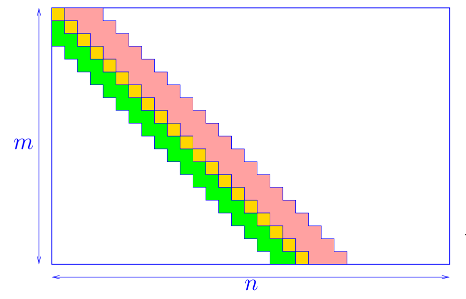
☛ a special class of sparse matrices with extra structure:

Definition 2.6.4 (bandwidth)**.**
For $\mathbf{A} = (a_{ij})_{i,j} \in \mathbb{K}^{m,n}$ *we call*

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

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

- $m(\mathbf{A}) = 1$ ▷ $\mathbf{A}$ diagonal matrix, → Def. 2.2.1
- $\overline{m}(\mathbf{A}) = \underline{m}(\mathbf{A}) = 1$ ▷ $\mathbf{A}$ tridiagonal matrix
- More general: $\mathbf{A} \in \mathbb{R}^{n,n}$ with $m(\mathbf{A}) \ll n \hat{=}$ banded matrix



🟨 : diagonal

🟥 : super-diagonals

🟩 : sub-diagonals

◁ $\overline{m}(\mathbf{A}) = 2$, $\underline{m}(\mathbf{A}) = 1$

➤ 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 !)

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

Definition 2.6.5 (Matrix envelope (*ger.:* Hülle))**.**

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

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

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

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

Example 2.6.23 (Envelope of a matrix).

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

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

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

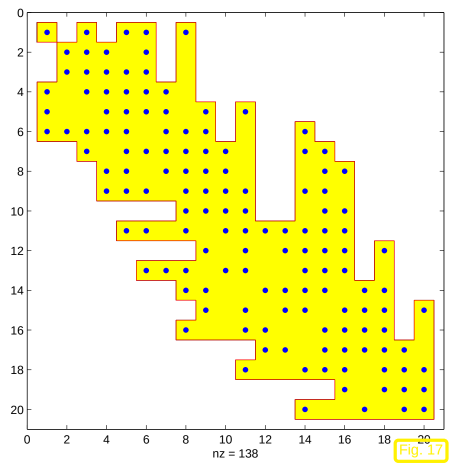


Fig. 17

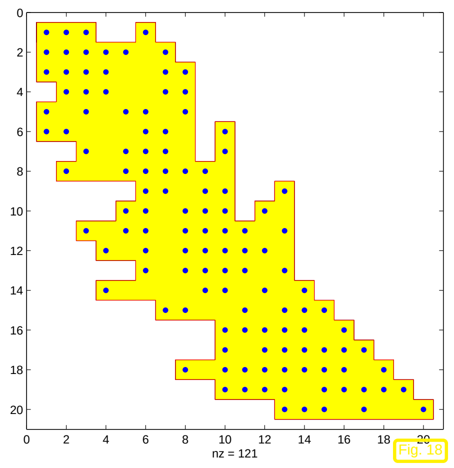


Fig. 18

Note: the envelope of the arrow matrix from Ex. 2.6.13 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 2.6.6 (Envelope and fill-in)**.**

If $\mathbf{A} \in \mathbb{K}^{n,n}$ *is regular with* LU*-decomposition* $\mathbf{A} = \mathbf{LU}$*, then fill-in (*$\rightarrow$ *Def. 2.6.3) is confined to* $\operatorname{env}(\mathbf{A})$*.*

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

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

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

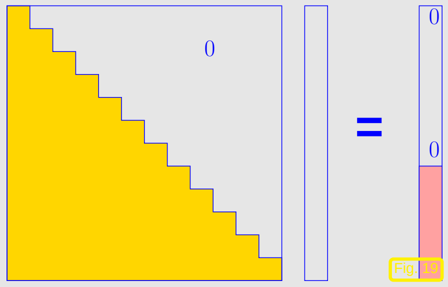
$$\Rightarrow \quad \operatorname{env}(\mathbf{L}^{(1)}) \subset \operatorname{env}(\mathbf{A}), \quad \operatorname{env}(\mathbf{U}^{(1)}) \subset \operatorname{env}(\mathbf{A}) \,.$$

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. 2.2.8 and proof of Lemma 2.2.3:

$$\left(\begin{array}{c|c} \tilde{\mathbf{A}} & \mathbf{b} \\ \hline \mathbf{c}^T & \alpha \end{array}\right) = \left(\begin{array}{c|c} \tilde{\mathbf{L}} & 0 \\ \hline \mathbf{l}^T & 1 \end{array}\right) \left(\begin{array}{c|c} \tilde{\mathbf{U}} & \mathbf{u} \\ \hline 0 & \xi \end{array}\right) \quad \Rightarrow \quad \begin{array}{l} \tilde{\mathbf{U}}^T \mathbf{l} = \mathbf{c} \,, \\ \tilde{\mathbf{L}} \mathbf{u} = \mathbf{b} \,. \end{array} \tag{2.6.3}$$

From Def. 2.6.5:



If $m_n^R(\mathbf{A}) = m$, then $c_1, \dots, c_{n-m} = 0$ (entries of $\mathbf{c}$ from (2.6.3))

If $m_n^C(\mathbf{A}) = m$, then $b_1, \dots, b_{n-m} = 0$ (entries of $\mathbf{b}$ from (2.6.3))

$\triangleleft$ for lower triagular LSE:

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

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

$\Downarrow$

assertion of the theorem $\square$

Envelope-aware LU-factorization:

Code 2.6.24: computing row bandwidths, $\rightarrow$ Def. 2.6.5

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

Code 2.6.25: envelope aware forward substitution

```
function y = substenv(L,y,mc)
% evelope aware forward substitution for $\mathbf{Lx} = \mathbf{y}$
% ($\mathbf{L}$ = 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. 2.2.5, for $\mathbf{Lx} = \mathbf{y}$, $\mathbf{L} \in \mathbb{K}^{n,n}$ regular lower triangular matrix is

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

By block LU-factorization $\rightarrow$ Rem. 2.2.8:

$$\left(\begin{array}{c|c} (\mathbf{A})_{1:n-1,1:n-1} & (\mathbf{A})_{1:n-1,n} \\ (\mathbf{A})_{n,1:n-1} & (\mathbf{A})_{n,n} \end{array}\right) = \left(\begin{array}{c|c} \mathbf{L}_1 & 0 \\ \mathbf{l}^T & 1 \end{array}\right) \left(\begin{array}{c|c} \mathbf{U}_1 & \mathbf{u} \\ 0 & \gamma \end{array}\right) , \tag{2.6.4}$$

$$\Rightarrow \quad (\mathbf{A})_{1:n-1,1:n-1} = \mathbf{L}_1\mathbf{U}_1 \,,\quad \mathbf{L}_1\mathbf{u} = (\mathbf{A})_{1:n-1,n} \,,\quad \mathbf{U}_1^T\mathbf{l} = (\mathbf{A})_{n,1:n-1}^T \,,\quad \mathbf{l}^T\mathbf{u} + \gamma = (\mathbf{A})_{n,n} \,. \tag{2.6.5}$$

Code 2.6.27: 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
```

$\triangleleft$ recursive implementation of envelope aware recursive LU-factorization (no pivoting !)

Assumption:

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

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

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

Definition 2.6.7 (Structurally symmetric matrix)**.**
$\mathbf{A} \in \mathbb{K}^{n,n}$ *is structurally symmetric, if*

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

▶ Store only a_{ij}, $(i,j) \in \operatorname{env}(\mathbf{A})$ 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 2.6.28 (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
```

$$P := n + \sum_{i=1}^{n} m_i(A) \; . \qquad (2.6.6)$$

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

Indexing rule:

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

	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										

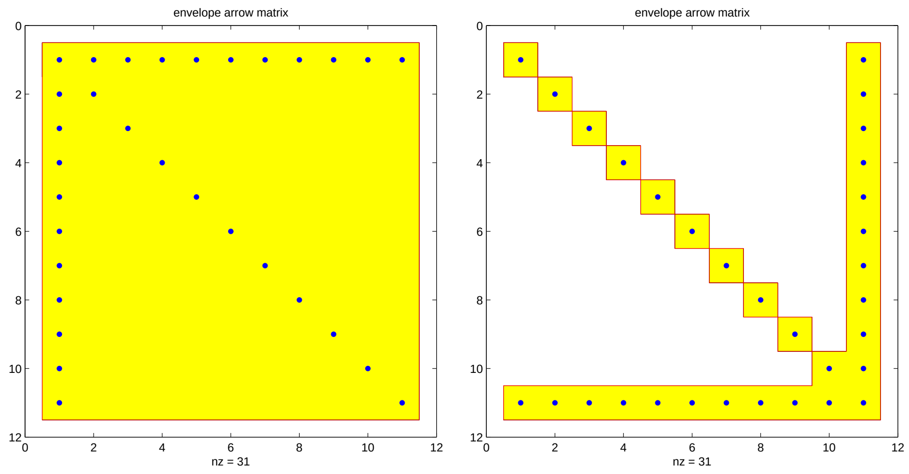
◇

Minimizing bandwidth:

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

Example 2.6.29 (Reducing bandwidth by row/column permutations).

Recall: cyclic permutation of rows/columns of arrow matrix → Ex. 2.6.14



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

$$\begin{pmatrix} * & 0 & 0 & * & * & * \\ 0 & * & 0 & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & 0 \\ * & 0 & 0 & * & * & * \\ * & 0 & 0 & * & * & * \\ * & 0 & 0 & * & * & * \end{pmatrix} \xrightarrow[i \leftarrow N+1-i]{} \begin{pmatrix} * & * & * & 0 & 0 & * \\ * & * & * & 0 & 0 & * \\ * & * & * & 0 & 0 & * \\ 0 & 0 & 0 & * & 0 & 0 \\ 0 & 0 & 0 & 0 & * & 0 \\ * & * & * & 0 & 0 & * \end{pmatrix}$$

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

◇

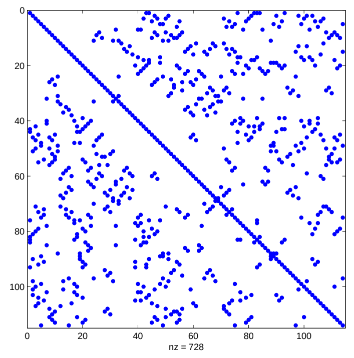
Example 2.6.30 (Reducing fill-in by reordering).

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

Code 2.6.31: preordering in MATLAB

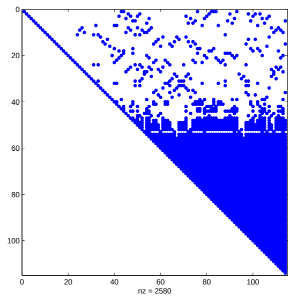
```
spy(M);
[L,U] = lu(M); spy(U);
r = symrcm(M);
[L,U] = lu(M(r,r)); spy(U);
m = symamd(M);
[L,U] = lu(M(m,m)); spy(U);
```

Pattern of $\mathbf{M}$ ➥

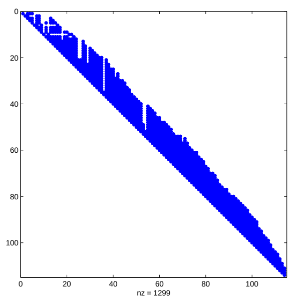


(Here: no row swaps from pivoting !)

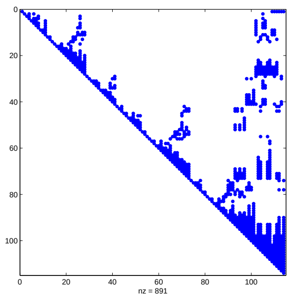
Examine patterns of LU-factors (→ Sect. 2.2) after reordering:



no reordering



reverse Cuthill-McKee



approximate minimum degree ◇

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

$\longrightarrow$ SuperLU (`http://www.cs.berkeley.edu/~demmel/SuperLU.html`)
$\longrightarrow$ UMFPACK (`http://www.cise.ufl.edu/research/sparse/umfpack/`)
$\longrightarrow$ Pardiso (`http://www.pardiso-project.org/`)
$\longrightarrow$ Matlab-\ (on sparse storage formats)

2.7 Stable Gaussian elimination without pivoting

Thm. 2.6.6 ➣ special structure of the matrix helps avoid fill-in in Gaussian elimination/LU-factorization *without* pivoting.

Ex. 2.6.21 ➣ pivoting can trigger huge fill-in that would not occur without it.

Ex. 2.6.30 ➣ fill-in reducing effect of reordering can be thwarted by later row swapping in the course of pivoting.

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

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

Definition 2.7.1 (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 2.7.2 (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. $m_{ii} > 0$, $i = 1, \dots, n$,
2. $m_{ii}m_{jj} - |m_{ij}|^2 > 0 \quad \forall 1 \leq i < j \leq n$,
3. *all eigenvalues of* $\mathbf{M}$ *are positive.* ($\longleftarrow$ *also sufficient for symmetric/Hermitian* $\mathbf{M}$)

Remark 2.7.1 (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 $D^2f(x^*)$ s.p.d.

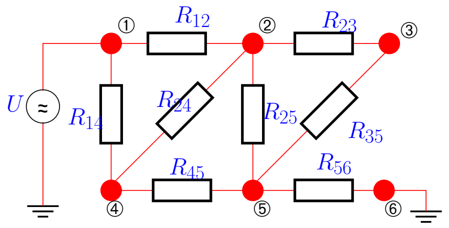
△

Example 2.7.2 (S.p.d. matrices from nodal analysis). $\longrightarrow$ Ex. 2.0.1

Consider:

electrical circuit entirely composed of Ohmic resistors.

Circuit equations from nodal analysis, see Ex. 2.0.1:



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

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

➢ 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 $k \neq j$, (2.7.1)
- $\sum_{j=1}^{n} a_{kj} \geq 0 \, , \quad k = 1, \dots, n \, ,$ (2.7.2)
- $\mathbf{A}$ is regular. (2.7.3)

➤ $\mathbf{A}$ is s.p.d., see Lemma 2.7.4 below.

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

Proof of (2.7.3): By Thm. 2.0.3 it suffices to show that the nullspace of $\mathbf{A}$ is trivial: $\mathbf{A}\mathbf{x} = 0 \;\Rightarrow\; \mathbf{x} = 0$.

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

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

Intermediate goal: show that all entries of $\mathbf{x}$ are the same

$$\mathbf{A}\mathbf{x} = 0 \quad \Rightarrow \quad x_i = \sum_{j \neq i} \frac{a_{ij}}{a_{ii}} x_j \quad \Rightarrow \quad |x_i| \leq \sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} |x_j| \, . \tag{2.7.4}$$

By (2.7.2) and the sign condition from (2.7.1) we conclude

$$\sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} \leq 1 \, . \tag{2.7.5}$$

Hence, (2.7.5) combined with the above estimate (2.7.4) that tells us that the maximum is smaller equal than a mean implies $|x_j| = |x_i|$ for all $j = 1, \dots, n$. Finally, the sign condition $a_{kj} \leq 0$ for $k \neq j$ enforces the same sign of all x_i.

◇

Definition 2.7.3 (Diagonally dominant matrix).
$\mathbf{A} \in \mathbb{K}^{n,n}$ *is diagonally dominant, if*

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

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

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

Lemma 2.7.4. *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 2.7.5 (Gaussian elimination for s.p.d. matrices).
Every symmetric/Hermitian positive definite matrix (→ Def. 2.7.1) 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 2.2.3, 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

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

➢ to show: $\widetilde{\mathbf{A}} - \frac{\mathbf{b}\mathbf{b}^T}{a_{11}}$ s.p.d. (→ 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. (→ Def. 2.7.1), for every $\boldsymbol{y} \in \mathbb{R}^{n-1} \setminus \{0\}$

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

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

The proof can also be based on the identities

$$\left(\begin{array}{c|c} (\mathbf{A})_{1:n-1,1:n-1} & (\mathbf{A})_{1:n-1,n} \\ \hline (\mathbf{A})_{n,1:n-1} & (\mathbf{A})_{n,n} \end{array}\right) = \left(\begin{array}{c|c} \mathbf{L}_1 & 0 \\ \hline \mathbf{l}^T & 1 \end{array}\right) \left(\begin{array}{c|c} \mathbf{U}_1 & \mathbf{u} \\ \hline 0 & \gamma \end{array}\right) , \qquad (2.6.4)$$

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

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 (→ Sect. 2.3)
(partial pivoting always picks current pivot row)

Lemma 2.7.6 (Cholesky decomposition for s.p.d. matrices)**.**
For any s.p.d. $\mathbf{A} \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$, *there is a unique upper triangular matrix* $\mathbf{R} \in \mathbb{K}^{n,n}$ *with* $r_{ii} > 0$, $i = 1, \dots, n$, *such that* $\mathbf{A} = \mathbf{R}^H\mathbf{R}$ *(Cholesky decomposition).*

Thm. 2.7.5 $\Rightarrow$ $\mathbf{A} = \mathbf{L}\mathbf{U}$ (unique LU-decomposition of $\mathbf{A}$, Lemma 2.2.3)

$\mathbf{A} = \mathbf{L}\mathbf{D}\widetilde{\mathbf{U}}$, $\mathbf{D} \mathrel{\hat=}$ diagonal of $\mathbf{U}$,
$\widetilde{\mathbf{U}} \mathrel{\hat=}$ *normalized* upper triangular matrix → Def. 2.2.1

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 ➨ $\mathbf{R} = \sqrt{\mathbf{D}}\mathbf{L}^T$. □

Code 2.7.3: simple Cholesky factorization

```
function R = cholfac(A)
% simple Cholesky factorization
n = size(A,1);
for k = 1:n
  for j=k+1:n
    A(j,j:n) = A(j,j:n) -
      A(k,j:n)*A(k,j)/A(k,k);
  end
  A(k,k:n) =
    A(k,k:n)/sqrt(A(k,k));
end
R = triu(A);
```

Computational costs (# elementary arithmetic operations) of Cholesky decomposition: $\frac{1}{6}n^3 + O(n^2)$
(➢ half the costs of LU-factorization, Code. 2.2.1)

MATLAB function: `R = chol(A)`

Solving LSE with s.p.d. system matrix via Cholesky decomposition + forward & backward substitution is numerically stable (→ Def. 2.5.5)

Recall Thm. 2.5.7: 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 (→ Def. 2.5.2) $\|\cdot\|_2$

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

➤ 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 $\mathbf{R}^{(k)}$.

Lemma 2.7.7 (LU-factorization of diagonally dominant matrices)**.**

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

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

Proof. (2.1.3) $\to$ induction w.r.t. n: After 1st step of elimination:

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

$$\blacktriangleright\quad |a_{ii}^{(1)}| - \sum_{\substack{j=2\\ j\neq i}}^{n} |a_{ij}^{(1)}| = \left|a_{ii} - \frac{a_{i1}}{a_{11}} a_{1i}\right| - \sum_{\substack{j=2\\ j\neq i}}^{n} \left|a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}\right|$$

$$\geq a_{ii} - \frac{|a_{i1}||a_{1i}|}{a_{11}} - \sum_{\substack{j=2\\ j\neq i}}^{n} |a_{ij}| - \frac{|a_{i1}|}{a_{11}} \sum_{\substack{j=2\\ j\neq i}}^{n} |a_{1j}|$$

$$\geq a_{ii} - \frac{|a_{i1}||a_{1i}|}{a_{11}} - \sum_{\substack{j=2\\ j\neq i}}^{n} |a_{ij}| - |a_{i1}| \frac{a_{11} - |a_{1i}|}{a_{11}} \geq a_{ii} - \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}| \geq 0\,.$$

$\mathbf{A}$ regular, diagonally dominant $\Rightarrow$ partial pivoting according to (2.3.4) selects i-th row in i-th step.

Remark 2.7.4 (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)`

`opts` $\in$ {	`LT`	$\leftrightarrow$	A lower triangular matrix	
	`UT`	$\leftrightarrow$	A upper triangular matrix	
	`UHESS`	$\leftrightarrow$	A upper Hessenberg matrix	
	`SYM`	$\leftrightarrow$	A Hermitian matrix	
	`POSDEF`	$\leftrightarrow$	A positive definite matrix	}

$\triangle$

2.8 QR-Factorization/QR-decomposition

Remark 2.8.1 (Sensitivity of linear mappings).

Consider problem map ($\to$ 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 $\mathbf{A} \in \mathbb{K}^{n,n}$ ➣ $\mathbf{x} \,\hat{=}\,$ "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. 2.5.9)

We assume that $\mathbb{K}^n$ is equipped with *some* vector norm ($\to$ Def. 2.5.1) and we use the induced matrix norm ($\to$ Def. 2.5.2) on $\mathbb{K}^{n,n}$.

$$\mathbf{A}\mathbf{x} = \mathbf{y} \quad \Rightarrow \quad \|\mathbf{x}\| \leq \left\|\mathbf{A}^{-1}\right\| \|\mathbf{y}\|$$

$$\mathbf{A}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{y} + \Delta\mathbf{y} \quad \Rightarrow \quad \mathbf{A}\Delta\mathbf{x} = \Delta\mathbf{y} \quad \Rightarrow \quad \|\Delta\mathbf{y}\| \leq \|\mathbf{A}\| \|\Delta\mathbf{x}\|$$

$$\Rightarrow \quad \frac{\|\Delta\mathbf{y}\|}{\|\mathbf{y}\|} \leq \frac{\|\mathbf{A}\| \|\Delta\mathbf{x}\|}{\|\mathbf{A}^{-1}\|^{-1} \|\mathbf{x}\|} = \mathrm{cond}(\mathbf{A}) \frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|}\,. \tag{2.8.1}$$

relative perturbation in result   relative perturbation in data

➣ Condition number $\mathrm{cond}(\mathbf{A})$ ($\to$ Def. 2.5.11) bounds amplification of relative error in argument vector in matrix$\times$vector-multiplication $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$.

$\triangle$

Example 2.8.2 (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 $\mathbf{T}(\mu)$ $\triangleright$

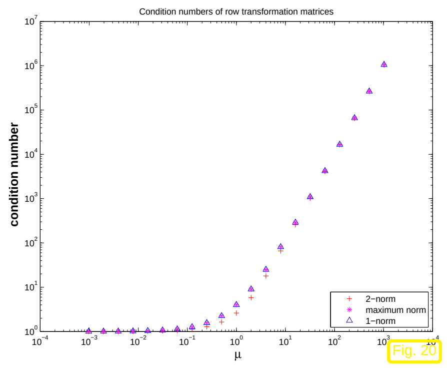


Fig. 20

Code 2.8.3: computing condition numbers of row transoformation 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';
```

Observation: $\operatorname{cond}(\mathbf{T}(\mu))$ 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. 2.5.5).

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

◇

Recall from linear algebra:

Definition 2.8.1 (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 2.8.2 (Criteria for Unitarity)**.**

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

► $\boxed{\mathbf{Q} \text{ unitary} \quad \Rightarrow \quad \operatorname{cond}(\mathbf{Q}) = 1}$ (2.8.1) ➤ unitary transformations enhance (numerical) stability

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

- all rows/columns (regarded as vectors $\in \mathbb{K}^n$) have Euclidean norm $= 1$,

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

Drawbacks of LU-factorization:

☹ often pivoting required (➞ destroys structure, Ex. 2.6.21, leads to fill-in)

☹ Possible (theoretical) instability of partial pivoting → Ex. 2.5.2

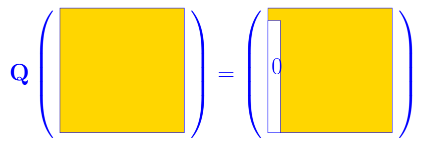
Stability problems of Gaussian elimination without pivoting are due to the fact that row transformations can convert well-conditioned matrices to ill-conditioned matrices, *cf.* Ex. 2.5.2

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

➢ 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.

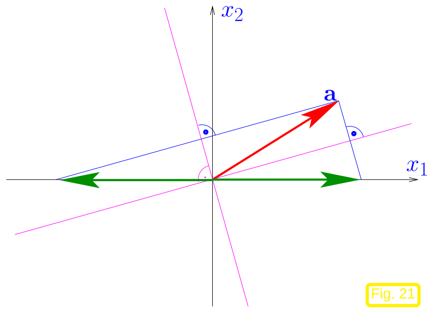
Goal: find unitary row transformation rendering certain matrix elements zero.



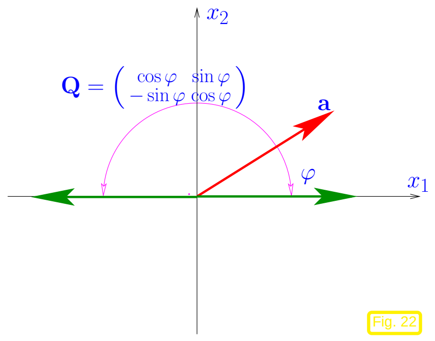
with $\mathbf{Q}^H = \mathbf{Q}^{-1}$.

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.

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



reflection at angle bisector,



$\mathbf{Q} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix}$
x_2
x_1
$\mathbf{a}$
φ
Fig. 22

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

➣ Note: two possible reflections/rotations

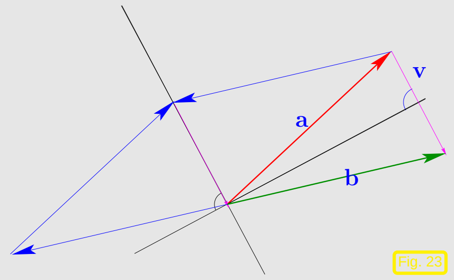
In nD: given $\mathbf{a} \in \mathbb{R}^n$ find orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n,n}$ such that $\mathbf{Q}\mathbf{a} = \|\mathbf{a}\|_2\,\mathbf{e}_1$, $\mathbf{e}_1 \hat{=}$ 1st unit vector.

Choice 1: Householder reflections

$$\mathbf{Q} = \mathbf{H}(\mathbf{v}) := \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^H}{\mathbf{v}^H\mathbf{v}} \quad \text{with} \quad \mathbf{v} = \tfrac{1}{2}(\mathbf{a} \pm \|\mathbf{a}\|_2\,\mathbf{e}_1)\,. \tag{2.8.2}$$

"Geometric derivation" of Householder reflection, see Figure 21

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.



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

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 2.8.4 (Details of Householder reflections).

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

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

  However, this is not really needed [24, Sect. 19.1] !

- If $\mathbb{K} = \mathbb{C}$ and $a_1 = |a_1|\exp(i\varphi)$, $\varphi \in [0, 2\pi[$, then choose

$$\mathbf{v} = \tfrac{1}{2}(\mathbf{a} \pm \|\mathbf{a}\|_2\,\mathbf{e}_1\exp(-i\varphi)) \quad \text{in (2.8.2).}$$

- efficient storage of Householder matrices → [2]

△

Choice 2: successive Givens rotations (→ 2D case)

$$\mathbf{G}_{1k}(a_1, a_k)\mathbf{a} := \begin{pmatrix} \bar\gamma & \cdots & \bar\sigma & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots \\ -\sigma & \cdots & \gamma & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_k \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_1^{(1)} \\ \vdots \\ 0 \\ \vdots \\ a_n \end{pmatrix}, \text{ if } \begin{array}{l} \gamma = \frac{a_1}{\sqrt{|a_1|^2+|a_k|^2}}\,, \\ \sigma = \frac{a_k}{\sqrt{|a_1|^2+|a_k|^2}}\,. \end{array} \tag{2.8.3}$$

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

Code 2.8.5: (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
```

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 (2.8.3)).

However, we aim to map *all* components to zero except for the first.

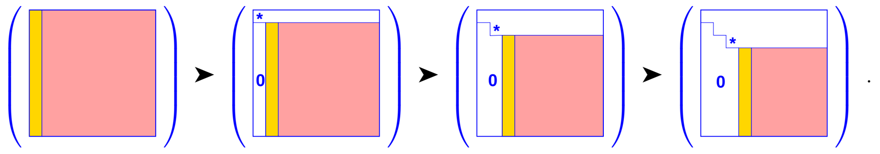
This can be achieved by $n-1$ *successive* Givens rotations.

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

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

Transformation to *upper triangular form* (→ Def. 2.2.1) by successive unitary transformations:

We may use either Householder reflections or successive Givens rotations as explained above.

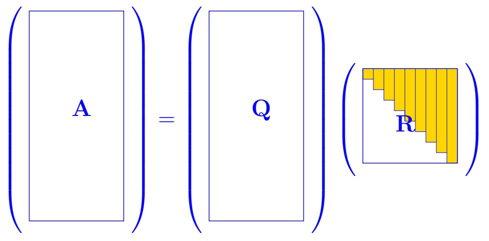


🟨 = "target column $\mathbf{a}$" (determines unitary transformation),
🟥 = modified in course of transformations.

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

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

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

$m > n$:


$, \quad \mathbf{A} = \mathbf{QR}\,, \quad \mathbf{Q} \in \mathbb{K}^{m,n}\,, \quad \mathbf{R} \in \mathbb{K}^{n,n}\,,$ (2.8.4)

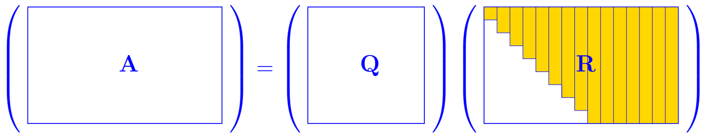
where $\mathbf{Q}^H\mathbf{Q} = \mathbf{I}$ (orthonormal columns), $\mathbf{R}$ upper triangular matrix.

Lemma 2.8.3 (Uniqueness of QR-factorization).
The "economical" QR-factorization (2.8.4) *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 2.7.6. □

$m < n$:


,

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

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

Remark 2.8.6 (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 (← 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 $\mathbf{A} \in \mathbb{K}^{m,n}$, $m > n$:

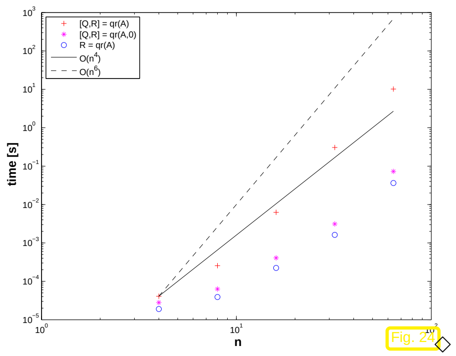
`[Q,R] = qr(A)` ➔ Costs: $O(m^2 n)$
`[Q,R] = qr(A,0)` ➔ Costs: $O(mn^2)$

Example 2.8.7 (Complexity of Householder QR-factorization).

Code 2.8.8: timing MATLAB QR-factorizations

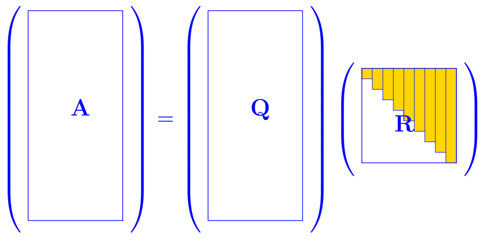
```
% Timing QR factorizations

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

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

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



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

Remark 2.8.9 (QR-orthogonalization).



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

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

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

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

△

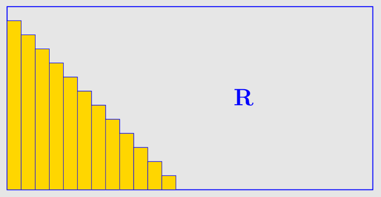
Remark 2.8.10 (Keeping track of unitary transformations).

How to store $\begin{matrix} \mathbf{G}_{i_1j_1}(a_1,b_1) \cdot \dots \cdot \mathbf{G}_{i_kj_k}(a_k,b_k) , \\ \mathbf{H}(\mathbf{v}_1) \cdot \dots \cdot \mathbf{H}(\mathbf{v}_k) \end{matrix}$?

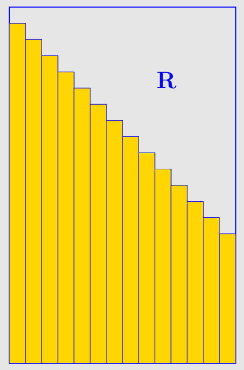
☛ For Householder reflections

$\mathbf{H}(\mathbf{v}_1) \cdot \dots \cdot \mathbf{H}(\mathbf{v}_k)$: store $\mathbf{v}_1, \dots, \mathbf{v}_k$

For in place QR-factorization of $\mathbf{A} \in \mathbb{K}^{m,n}$: store "'Householder vectors" $\mathbf{v}_j$ (decreasing size !) in lower triangle of $\mathbf{A}$



↑ Case $m < n$



🟨 = Householder vectors

Case $m > n$ →

☛ Convention for Givens rotations ($\mathbb{K} = \mathbb{R}$)

$$\mathbf{G} = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \quad \Rightarrow \quad \text{store} \quad \rho := \begin{cases} 1 & \text{, if } \gamma = 0 , \\ \frac{1}{2}\operatorname{sign}(\gamma)\sigma & \text{, if } |\sigma| < |\gamma| , \\ 2\operatorname{sign}(\sigma)/\gamma & \text{, if } |\sigma| \ge |\gamma| . \end{cases}$$

$$\blacktriangleright \quad \begin{cases} \rho = 1 & \Rightarrow \quad \gamma = 0 , \quad \sigma = 1 \\ |\rho| < 1 & \Rightarrow \quad \sigma = 2\rho , \quad \gamma = \sqrt{1-\sigma^2} \\ |\rho| > 1 & \Rightarrow \quad \gamma = 2/\rho , \quad \sigma = \sqrt{1-\gamma^2} . \end{cases}$$

Then store $\mathbf{G}_{ij}(a,b)$ as triple (i,j,ρ)

The rationale for this convention is to curb the impact of roundoff errors.

Storing orthogonal transformation matrices is usually inefficient !

Algorithm 2.8.11 (Solving linear system of equations by means of QR-decomposition).

$\mathbf{A}\mathbf{x} = \mathbf{b}$:

① QR-decomposition $\mathbf{A} = \mathbf{Q}\mathbf{R}$, 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 $\mathbf{R}\mathbf{x} = \mathbf{z}$, computational costs $\frac{1}{2}n(n+1)$

☞ Computing the generalized QR-decomposition $\mathbf{A} = \mathbf{Q}\mathbf{R}$ 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

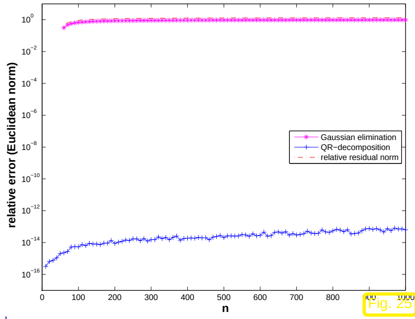
QR-decomposition **+** orthogonal transformation **+** backward substitution

in a stable manner.

Example 2.8.12 (Stable solution of LSE by means of QR-decomposition). $\rightarrow$ Ex. 2.5.2

Code 2.8.13: QR-fac. $\leftrightarrow$ Gaussian elimination

```
res = [];
for n=10:10:1000
  A=[tril(-ones(n,n-1))+2*[eye(n-1);.
        zeros(1,n-1)],ones(n,1)];
  x=((-1).^(1:n))';
  b=A*x;
  [Q,R]=qr(A);

  errlu=norm(A\b-x)/norm(x);
  errqr=norm(R\(Q'*b)-x)/norm(x);
  res=[res; n,errlu,errqr];
end
semilogy(res(:,1),res(:,2),'m-*',...
         res(:,1),res(:,3),'b-+');
```



◁ superior stability of QR-decomposition !

◇

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

bandwidth

$\mathbf{A} \in \mathbb{C}^{n,n}$ with QR-decomposition $\mathbf{A} = \mathbf{Q}\mathbf{R}$ $\Rightarrow$ $m(\mathbf{R}) \leq m(\mathbf{A})$ ($\rightarrow$ Def. 2.6.4)

Example 2.8.14 (QR-based solution of tridiagonal LSE).

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

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

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

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

Code 2.8.15: 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 $O(n)$

◇

Remark 2.8.16 (Storing the $\mathbf{Q}$-factor).

The previous example (Code 2.8.14) showed that assembly of the $\mathbf{Q}$-factor in the QR-factorization of $\mathbf{A}$ is *not needed*, when the linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ 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. 2.2.6 for QR-factorization:

Inefficient (!) code ▷

```
% Setting: $N \gg 1$,
% large tridiagonal matrix $\mathbf{A} \in \mathbb{R}^{n,n}$
[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 2.8.14, Sect. 2.6.2.
- 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. 2.8.10.

△

Remark 2.8.17 (Testing for near singularity of a matrix).

Very small (w.r.t. matrix norm) element r_{ii} in QR-factor $\mathbf{R}$ $\leftrightarrow$ $\mathbf{A}$ "nearly singular"

△

2.9 ModificationTechniques

Example 2.9.1 (Resistance to currents map).

Large (linear) electric circuit ▷

Sought:

Dependence of (certain) branch currents on "continuously varying" resistance R_x

(➣ currents for many different values of R_x)

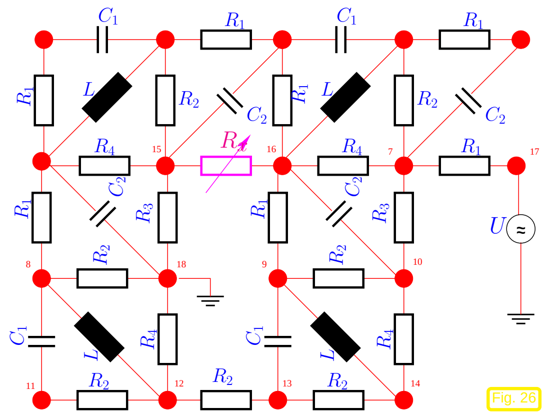


Fig. 26

- Only *a few* entries of the nodal analysis matrix $\mathbf{A}$ ($\rightarrow$ Ex. 2.0.1) are affected by variation of R_x ! (If R_x connects nodes i & j $\Rightarrow$ 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 [18, Sect. 12.6], [38, 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 $x \in \mathbb{K}$

$$\mathbf{A}, \widetilde{\mathbf{A}} \in \mathbb{K}^{n,n}: \quad \tilde{a}_{ij} = \begin{cases} a_{ij} & \text{, if } (i,j) \neq (i^*,j^*) \,, \\ x + a_{ij} & \text{, if } (i,j) = (i^*,j^*) \,, \end{cases} \quad , \quad i^*, j^* \in \{1,\dots,n\} \,. \tag{2.9.1}$$

$$\blacktriangleright \quad \boxed{\widetilde{\mathbf{A}} = \mathbf{A} + x \cdot \mathbf{e}_{i^*}\mathbf{e}_{j^*}^T} \,. \tag{2.9.2}$$

Recall: $\mathbf{e}_i \hat{=}$ i-th unit vector

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

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

$$\blacktriangleright \quad \boxed{\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{e}_{i^*}\mathbf{x}^T} \,. \tag{2.9.3}$$

Both matrix modifications (2.9.1) and (2.9.3) are specimens of a rank-1-modifications.

◇

$$\mathbf{A} \in \mathbb{K}^{n,n} \quad \mapsto \quad \widetilde{\mathbf{A}} := \mathbf{A} + \boxed{\mathbf{u}\mathbf{v}^H} \,, \quad \mathbf{u}, \mathbf{v} \in \mathbb{K}^n \,. \tag{2.9.4}$$

general rank-1-matrix

Remark 2.9.3 (Solving LSE in the case of rank-1-modification).

Lemma 2.9.1 (Sherman-Morrison-Woodbury formula). *For regular* $\mathbf{A} \in \mathbb{K}^{n,n}$, *and* $\mathbf{U}, \mathbf{V} \in \mathbb{K}^{n,k}$, $n, k \in \mathbb{N}$, $k \leq n$, *holds*

$$(\mathbf{A} + \mathbf{U}\mathbf{V}^H)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^H\mathbf{A}^{-1} \,,$$

if $\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U}$ *regular.*

Task: Solve $\widetilde{\mathbf{A}}\mathbf{x} = \mathbf{b}$, when LU-factorization $\mathbf{A} = \mathbf{L}\mathbf{U}$ already known

Apply Lemma 2.9.1 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 ! ➤

Code 2.9.4: solving a rank-1 modified LSE

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

△

The approach of Rem. 2.9.3 is certainly efficient, but may *suffer from instability* similar to Gaussian elimination without pivoting, *cf.* Ex. 2.3.1.

This can be avoided by using QR-factorization (→ Sect. 2.8) and corresponding update techniques. This is the principal 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 6.

Task: Efficient computation of QR-factorization (→ Sect. 2.8) $\widetilde{\mathbf{A}} = \widetilde{\mathbf{Q}}\widetilde{\mathbf{R}}$ of $\widetilde{\mathbf{A}}$ from (2.9.4), when QR-factorization $\mathbf{A} = \mathbf{Q}\mathbf{R}$ already known

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

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

② Objective: $\mathbf{w} \longrightarrow \|\mathbf{w}\|\,\mathbf{e}_1$ ➤ via $n-1$ Givens rotations, see (2.8.3).

$$\mathbf{w} = \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ * \\ * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-1,n}} \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ * \\ 0 \end{pmatrix} \xrightarrow{\mathbf{G}_{n-2,n-1}} \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ 0 \\ 0 \end{pmatrix} \xrightarrow{\mathbf{G}_{n-3,n-2}} \cdots \xrightarrow{\mathbf{G}_{1,2}} \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.9.5}$$

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 in crucial in the current context.

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

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

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

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

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

➡ Asymptotic complexity $O(n^2)$

Imagine that in (2.9.5) we had chosen to annihilate the components $2,\dots,n$ of $\mathbf{w}$ by the product of Givens rotations $\mathbf{G}_{12}\mathbf{G}_{13}\cdots\mathbf{G}_{1,n-1}$. This would have resulted in a fully populated matrix $\mathbf{R}_1$!

In this case, the next step could be carried out with an effort $O(n^3)$ only.

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

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

$$\xrightarrow{\mathbf{G}_{n-2,n-1}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{n-1,n}} \begin{pmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \end{pmatrix} =: \widetilde{\mathbf{R}}\,. \qquad (2.9.6)$$

➡ Asymptotic complexity $O(n^2)$

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

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

Special case: rank-1-modifications preserving symmetry & *positivity* ($\rightarrow$ Def. 2.7.1):

$$\mathbf{A}=\mathbf{A}^H\in\mathbb{K}^{n,n} \;\mapsto\; \widetilde{\mathbf{A}}:=\mathbf{A}+\alpha\mathbf{v}\mathbf{v}^H\,,\quad \mathbf{v}\in\mathbb{K}^n\,,\;\alpha>0\,. \qquad (2.9.7)$$

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 $\widetilde{\mathbf{A}}=\widetilde{\mathbf{R}}^H\widetilde{\mathbf{R}}$ ($\rightarrow$ Lemma 2.7.6) of $\widetilde{\mathbf{A}}$ from (2.9.7), when Cholesky factorization $\mathbf{A}=\mathbf{R}^H\mathbf{R}$ of $\mathbf{A}$ already known

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

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

② Idea: formal Gaussian elimination: with $\widetilde{\mathbf{w}}=(w_2,\dots,w_n)^T$ $\rightarrow$ see (2.1.3)

$$\mathbf{I}+\alpha\mathbf{w}\mathbf{w}^H=\left(\begin{array}{c|c} 1+\alpha w_1^2 & \alpha w_1\widetilde{\mathbf{w}}^H \\ \hline \alpha w_1\widetilde{\mathbf{w}} & \mathbf{I}+\alpha\widetilde{\mathbf{w}}\widetilde{\mathbf{w}}^H \end{array}\right) \rightarrow \left(\begin{array}{c|c} 1+\alpha w_1^2 & \alpha w_1\widetilde{\mathbf{w}}^H \\ \hline 0 & \mathbf{I}+\alpha^{(1)}\widetilde{\mathbf{w}}\widetilde{\mathbf{w}}^H \end{array}\right) \qquad (2.9.8)$$

where $\alpha^{(1)}:=1-\dfrac{\alpha^2 w_1^2}{1+\alpha v_1^2}$. same structure ➢ recursion

▶ Computation of Cholesky-factorization

$$\mathbf{I}+\alpha\mathbf{w}\mathbf{w}^H=\mathbf{R}_1^H\mathbf{R}_1\ .$$

Motivation: "recursion" (2.9.8).

➜ asymptotic complexity $O(n^2)$ ($O(n)$, if only d, s computed $\rightarrow$ (2.9.9))

Code 2.9.6: Cholesky factorization of rank-1-modified identity matrix

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

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

$$\mathbf{R}_1=\begin{pmatrix}d_1&&&\\&\ddots&&\\&&\ddots&\\&&&d_n\end{pmatrix}+\begin{pmatrix}s_1&&&\\&\ddots&&\\&&\ddots&\\&&&s_n\end{pmatrix}\begin{pmatrix}0&w_2&w_3&\cdots&\cdots&w_n\\0&0&w_3&\cdots&\cdots&w_n\\\vdots&&\ddots&&&\vdots\\\vdots&&&0&w_{n-1}&w_n\\0&\cdots&&\cdots&0&w_n\\0&\cdots&&&\cdots&0\end{pmatrix} \tag{2.9.9}$$

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

▶ Smart multiplication

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

➜ Complexity $O(n^2)$

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

Code 2.9.8: Update of Cholesky factorization in the case of s.p.d. preserving rank-1-modification

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

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

2.9.0.2 Adding a column

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 6.

$$\mathbf{A}\in\mathbb{K}^{m,n}\ \mapsto\ \widetilde{\mathbf{A}}=\left[(\mathbf{A})_{:,1},\dots,(\mathbf{A})_{:,k-1},\mathbf{v},(\mathbf{A})_{:,k},\dots,(\mathbf{A})_{:,n}\right]\ ,\quad \mathbf{v}\in\mathbb{K}^m\ . \tag{2.9.10}$$

Known: QR-factorization $\mathbf{A}=\mathbf{Q}\mathbf{R}$, $\mathbf{Q}\in\mathbb{K}^{m,m}$ 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 (2.9.10), $\widetilde{\mathbf{Q}}\in\mathbb{K}^{m,m}$ unitary, $\widetilde{\mathbf{R}}\in\mathbb{K}^{m,n+1}$ upper triangular

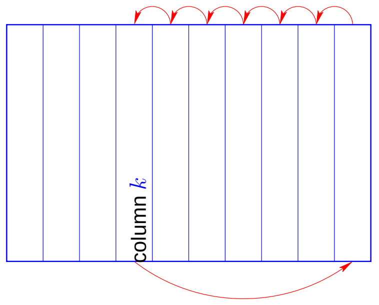
Idea: Easy, if $k=n+1$ (adding last column)

▶ $\exists$ column permutation

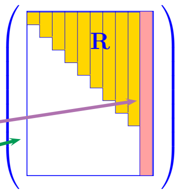
$$k\mapsto n+1\ ,\ i\mapsto i-1\ ,\ i=k+1,\dots,n+1$$

$\sim$ permutation matrix

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



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



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

case $m>n+1$

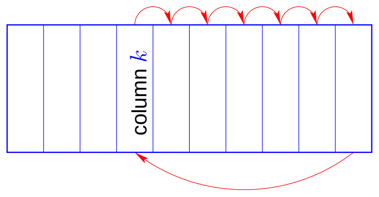
① If $m>n+1$: $\exists$ orthogonal transformation $\mathbf{Q}_1\in\mathbb{K}^{m,m}$ (Householder reflection) with

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

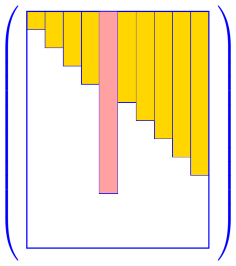
➡ Computational effort $O(m-n)$ (a single reflection)

inverse permutation:

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

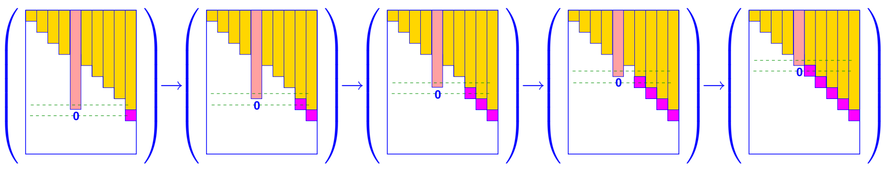


$$\mathbf{Q}_1\mathbf{Q}^H\widetilde{\mathbf{A}} = \mathbf{Q}_1\mathbf{Q}^H\mathbf{A}_1\mathbf{P}^H = \begin{pmatrix} * & \cdots & & * & & \cdots & * \\ 0 & * & & * & & & \vdots \\ \vdots & & \ddots & \vdots & & & \\ & & & \vdots & \ddots & & \vdots \\ & & & * & & * & * \\ \vdots & & & * & & & * \\ 0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



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

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



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

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

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

2.9.0.3 Adding a row

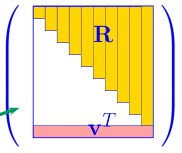
$$\mathbf{A} \in \mathbb{K}^{m,n} \mapsto \widetilde{\mathbf{A}} = \begin{bmatrix} (\mathbf{A})_{1,:} \\ \vdots \\ (\mathbf{A})_{k-1,:} \\ \mathbf{v}^T \\ (\mathbf{A})_{k,:} \\ \vdots \\ (\mathbf{A})_{m,:} \end{bmatrix}, \quad \mathbf{v} \in \mathbb{K}^n . \tag{2.9.11}$$

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

Task: efficient computation of QR-factorization $\widetilde{\mathbf{A}} = \widetilde{\mathbf{Q}}\widetilde{\mathbf{R}}$ of $\widetilde{\mathbf{A}}$ from (2.9.11) , $\widetilde{\mathbf{Q}} \in \mathbb{K}^{m+1,m+1}$ unitary, $\widetilde{\mathbf{R}} \in \mathbb{K}^{m+1,n+1}$ upper triangular matrix.

① $\exists$ (partial) cyclic row permutation $m+1 \leftarrow k$, $i \leftarrow i+1$, $i = k, \dots, m$:
$\rightarrow$ unitary permutation matrix ($\rightarrow$ Def. 2.3.1) $\mathbf{P} \in \{0,1\}^{m+1,m+1}$

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



Fall $m = n$

② Transform into upper triangular form by $m-1$ successive Givens rotations:

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

$$\cdots \xrightarrow{\mathbf{G}_{m-2,m}} \begin{pmatrix} * & \cdots & & & \cdots & * \\ 0 & * & & & & \vdots \\ \vdots & 0 & \ddots & & & \\ \vdots & \vdots & & * & & \vdots \\ 0 & 0 & & 0 & * & * \\ 0 & \cdots & & 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{G}_{m-1,m}} \begin{pmatrix} * & \cdots & & & \cdots & * \\ 0 & * & & & & \vdots \\ \vdots & 0 & \ddots & & & \\ \vdots & \vdots & & * & & \vdots \\ 0 & 0 & & 0 & * & * \\ 0 & \cdots & & & 0 & * \end{pmatrix} := \widetilde{\mathbf{R}} \quad (2.9.12)$$

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

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

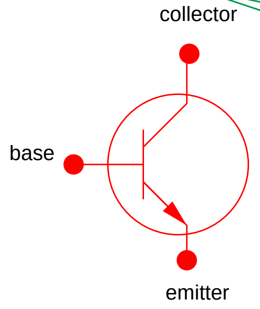
☞ Similar update algorithms exist for modifications arising from dropping one row or column of a matrix.

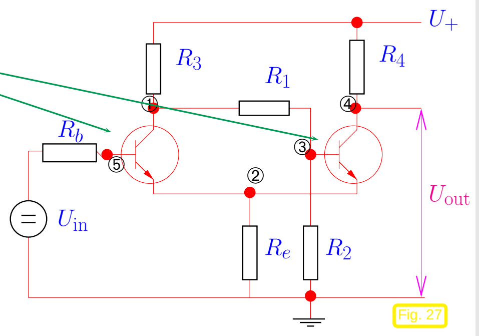
3 Iterative Methods for Non-Linear Systems of Equations

Example 3.0.1 (Non-linear electric circuit).

Schmitt trigger circuit ▷

NPN bipolar junction transistor:





Ebers-Moll model (large signal approximation):

$$\begin{aligned} I_C &= I_S \left(e^{\frac{U_{BE}}{U_T}} - e^{\frac{U_{BC}}{U_T}} \right) - \frac{I_S}{\beta_R} \left(e^{\frac{U_{BC}}{U_T}} - 1 \right) = I_C(U_{BE}, U_{BC}) , \\ I_B &= \frac{I_S}{\beta_F} \left(e^{\frac{U_{BE}}{U_T}} - 1 \right) + \frac{I_S}{\beta_R} \left(e^{\frac{U_{BC}}{U_T}} - 1 \right) = I_B(U_{BE}, U_{BC}) , \\ I_E &= I_S \left(e^{\frac{U_{BE}}{U_T}} - e^{\frac{U_{BC}}{U_T}} \right) + \frac{I_S}{\beta_F} \left(e^{\frac{U_{BE}}{U_T}} - 1 \right) = I_E(U_{BE}, U_{BC}) . \end{aligned} \quad (3.0.1)$$

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).)

Non-linear system of equations from nodal analysis (→ Ex. 2.0.1):

$$\begin{aligned}
① :\;& R_3(U_1-U_+)+R_1(U_1-U_3)+I_{\mathsf{B}}(U_5-U_1,U_5-U_2) &= 0\,,\\
② :\;& R_eU_2+I_{\mathsf{E}}(U_5-U_1,U_5-U_2)+I_{\mathsf{E}}(U_3-U_4,U_3-U_2) &= 0\,,\\
③ :\;& R_1(U_3-U_1)+I_{\mathsf{B}}(U_3-U_4,U_3-U_2) &= 0\,,\\
④ :\;& R_4(U_4-U_+)+I_{\mathsf{C}}(U_3-U_4,U_3-U_2) &= 0\,,\\
⑤ :\;& R_b(U_5-U_{\mathrm{in}})+I_{\mathsf{B}}(U_5-U_1,U_5-U_2) &= 0\,.
\end{aligned} \tag{3.0.2}$$

5 equations ↔ 5 unknowns U_1,U_2,U_3,U_4,U_5

Formally: (3.0.2) $\longleftrightarrow$ $F(\mathbf{u})=0$ ◇

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

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

Note: $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ ↔ "same number of equations and unknowns"

In general no existence & uniqueness of solutions

3.1 Iterative methods

Remark 3.1.1 (Necessity of iterative approximation).

Gaussian elimination (→ 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.

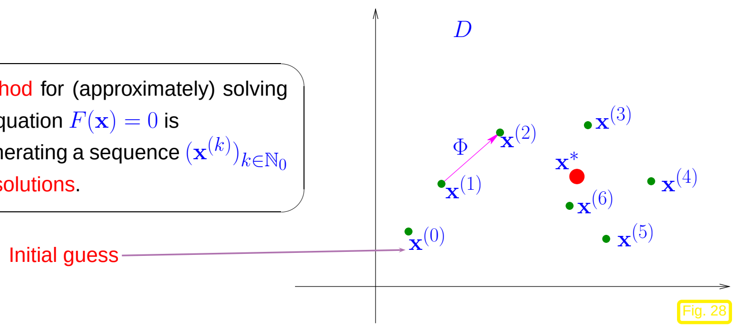


Fig. 28

Fundamental concepts: convergence ➡ speed of convergence

consistency

• iterate $\mathbf{x}^{(k)}$ depends on F and (one or several) $\mathbf{x}^{(n)}$, $n<k$, e.g.,

$$\mathbf{x}^{(k)} = \underbrace{\Phi_F(\mathbf{x}^{(k-1)},\dots,\mathbf{x}^{(k-m)})}_{\text{iteration function for } m\text{-point method}} \tag{3.1.1}$$

• $\mathbf{x}^{(0)},\dots,\mathbf{x}^{(m-1)}$ = initial guess(es) (*ger.:* Anfangsnäherung)

Definition 3.1.1 (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 3.1.2 (Consistency of iterative methods).
An iterative method is consistent with $F(\mathbf{x})=0$

$$:\Leftrightarrow \quad \Phi_F(\mathbf{x}^*,\dots,\mathbf{x}^*)=\mathbf{x}^* \quad \Leftrightarrow \quad F(\mathbf{x}^*)=0$$

Terminology: error of iterates $\mathbf{x}^{(k)}$ is defined as: $\mathbf{e}^{(k)} := \mathbf{x}^{(k)} - \mathbf{x}^*$

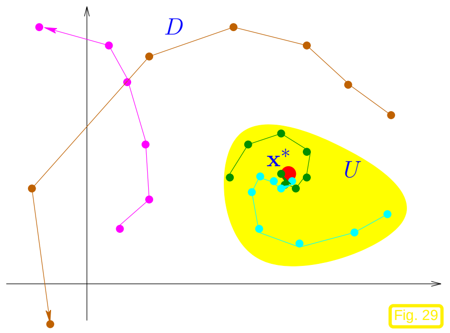
Definition 3.1.3 (Local and global convergence)**.**
An iterative method converges locally *to* $\mathbf{x}^* \in \mathbb{R}^n$*, if there is a neighborhood* $U \subset D$ *of* $\mathbf{x}^*$*, such that*

$$\mathbf{x}^{(0)},\dots,\mathbf{x}^{(m-1)} \in U \quad \Rightarrow \quad \mathbf{x}^{(k)} \text{ well defined } \quad \wedge \quad \lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$$

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.)



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?

3.1.1 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. 2.5.2.

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 3.1.4 (Linear convergence)**.**
A sequence $\mathbf{x}^{(k)}$, $k = 0,1,2,\dots$, *in* $\mathbb{R}^n$ converges linearly *to* $\mathbf{x}^* \in \mathbb{R}^n$*, if*

$$\exists L < 1: \quad \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \le L \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \quad \forall k \in \mathbb{N}_0 .$$

Terminology: least upper bound for L gives the rate of convergence

Remark 3.1.2 (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 3.1.5 (Equivalence of norms).
Two norms $\|\cdot\|_1$ *and* $\|\cdot\|_2$ *on a vector space* V *are equivalent if*

$$\exists \underline{C}, \overline{C} > 0: \quad \underline{C}\,\|v\|_1 \le \|v\|_2 \le \overline{C}\,\|v\|_2 \quad \forall v \in V .$$

Theorem 3.1.6 (Equivalence of all norms on finite dimensional vector spaces).
If $\dim V < \infty$ *all norms (→ Def. 2.5.1) on* V *are equivalent (→ Def. 3.1.5).*

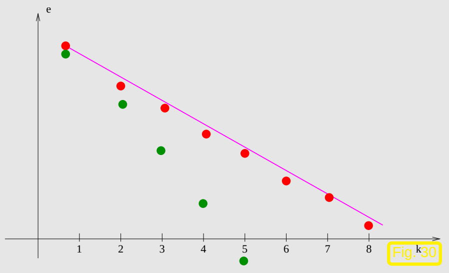
△

Remark 3.1.3 (Seeing linear convergence).

norms of iteration errors
$\updownarrow$
$\sim$ straight line in lin-log plot

$$\left\|\mathbf{e}^{(k)}\right\| \le L^k \left\|\mathbf{e}^{(0)}\right\| ,$$
$$\log(\left\|\mathbf{e}^{(k)}\right\|) \le k\log(L) + \log(\left\|\mathbf{e}^{(0)}\right\|) .$$

(●: Any "faster" convergence also qualifies as linear !)



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. 3.1.4) assume (with $0 < L < 1$)

$$\epsilon_{k+1} \approx L\epsilon_k \quad\Rightarrow\quad \log\epsilon_{k+1} \approx \log L + \log\epsilon_k \quad\Rightarrow\quad \log\epsilon_{k+1} \approx k\log L + \log\epsilon_0 \;. \qquad (3.1.2)$$

We conclude that $\log L < 0$ describes slope of graph in lin-log error chart.

Example 3.1.4 (Linearly convergent iteration).

Iteration ($n = 1$):

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

Code 3.1.5: 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);
```

x has to be initialized with the different values for x_0.

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

k	$x^{(0)} = 0.4$		$x^{(0)} = 0.6$		$x^{(0)} = 1$	
	$x^{(k)}$	$\frac{\lvert x^{(k)}-x^{(15)}\rvert}{\lvert x^{(k-1)}-x^{(15)}\rvert}$	$x^{(k)}$	$\frac{\lvert x^{(k)}-x^{(15)}\rvert}{\lvert x^{(k-1)}-x^{(15)}\rvert}$	$x^{(k)}$	$\frac{\lvert x^{(k)}-x^{(15)}\rvert}{\lvert x^{(k-1)}-x^{(15)}\rvert}$
2	3.3887	0.1128	3.4727	0.4791	2.9873	0.4959
3	3.2645	0.4974	3.3056	0.4953	3.0646	0.4989
4	3.2030	0.4992	3.2234	0.4988	3.1031	0.4996
5	3.1723	0.4996	3.1825	0.4995	3.1224	0.4997
6	3.1569	0.4995	3.1620	0.4994	3.1320	0.4995
7	3.1493	0.4990	3.1518	0.4990	3.1368	0.4990
8	3.1454	0.4980	3.1467	0.4980	3.1392	0.4980

▶ Rate of convergence ≈ 0.5

Linear convergence as in Def. 3.1.4

$\Updownarrow$

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

$\longrightarrow$ Rem. 3.1.3

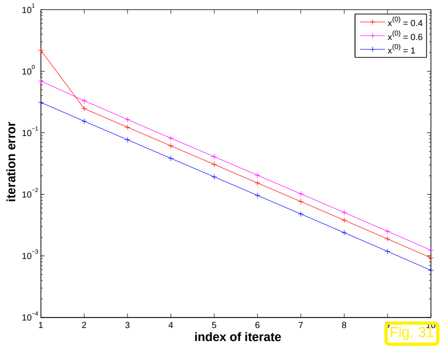


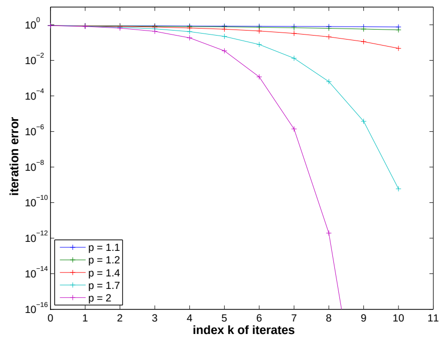
Fig. 31

◇

Definition 3.1.7 (Order of convergence).
*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*

$$\exists C > 0: \quad \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \le C \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|^p \quad \forall k \in \mathbb{N}_0 \,,$$

and, in addition, $C < 1$ in the case $p = 1$ (linear convergence $\longrightarrow$ Def. 3.1.4).



◁ Qualitative error graphs for convergence of order p

(lin-log scale)

In the case of convergence of order p ($p > 1$) (see Def. 3.1.7):

$$\epsilon_{k+1} \approx C\epsilon_k^p \quad\Rightarrow\quad \log\epsilon_{k+1} = \log C + p\log\epsilon_k \quad\Rightarrow\quad \log\epsilon_{k+1} = \log C \sum_{l=0}^{k} p^l + p^{k+1}\log\epsilon_0$$

$$\Rightarrow\quad \log\epsilon_{k+1} = -\frac{\log C}{p-1} + \left(\frac{\log C}{p-1} + \log\epsilon_0\right) p^{k+1} \;.$$

In this case, the error graph is a concave power curve (for sufficiently small ϵ_0 !)

Example 3.1.6 (quadratic convergence). (= convergence of order 2)

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

$$x^{(k+1)} = \frac{1}{2}(x^{(k)} + \frac{a}{x^{(k)}}) \quad \Rightarrow \quad |x^{(k+1)} - \sqrt{a}| = \frac{1}{2x^{(k)}}|x^{(k)} - \sqrt{a}|^2 . \qquad (3.1.3)$$

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 (3.1.3) converges with order 2 to $\sqrt{a}$

Note: $x^{(k+1)} < x^{(k)}$ for all $k \geq 2$ ➣ $(x^{(k)})_{k\in\mathbb{N}_0}$ converges as a decreasing sequence that is bounded from below ($\rightarrow$ analysis course)

How to guess the order of convergence in a numerical experiment?
Abbreviate $\epsilon_k := \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|$ and then

$$\epsilon_{k+1} \approx C\epsilon_k^p \quad \Rightarrow \quad \log\epsilon_{k+1} \approx \log C + p\log\epsilon_k \quad \Rightarrow \quad \frac{\log\epsilon_{k+1} - \log\epsilon_k}{\log\epsilon_k - \log\epsilon_{k-1}} \approx p .$$

Numerical experiment: iterates for $a = 2$:

k	$x^{(k)}$	$e^{(k)} := x^{(k)} - \sqrt{2}$	$\log\frac{\lvert e^{(k)}\rvert}{\lvert e^{(k-1)}\rvert} : \log\frac{\lvert e^{(k-1)}\rvert}{\lvert e^{(k-2)}\rvert}$
0	2.00000000000000000	0.58578643762690485	
1	1.50000000000000000	0.08578643762690485	
2	1.41666666666666652	0.00245310429357137	1.850
3	1.41421568627450966	0.00000212390141452	1.984
4	1.41421356237468987	0.00000000000159472	2.000
5	1.41421356237309492	0.00000000000000022	$\boxed{0.630}$

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:

$$x^{(k)} = x^*(1 + \delta_k) \quad \Rightarrow \quad x^{(k)} - x^* = \delta_k x^* .$$
$$\Rightarrow |x^*\delta_{k+1}| = |x^{(k+1)} - x^*| \leq C|x^{(k)} - x^*|^2 = C|x^*\delta_k|^2$$
$$\Rightarrow \quad |\delta_{k+1}| \leq C|x^*|\delta_k^2 . \qquad (3.1.4)$$

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 (3.1.6) implies $\delta_{k+1} \approx 10^{-2\ell}$.

◇

3.1.2 Termination criteria

Usually (even without roundoff errors) the iteration will never arrive at an/the exact solution $\mathbf{x}^*$ after finitely many steps. Thus, we can only hope to compute an *approximate* solution by accepting $\mathbf{x}^{(K)}$ as result for some $K \in \mathbb{N}_0$. Termination criteria (*ger.:* Abbruchbedingungen) are used to determine a suitable value for K.

For the sake of efficiency: ▷ stop iteration when iteration error is just “small enough”

“small enough” depends on concrete setting:

Usual goal: $\left\|\mathbf{x}^{(K)} - \mathbf{x}^*\right\| \leq \tau$, τ $\hat{=}$ prescribed tolerance.

$$\text{Ideal:} \quad K = \operatorname{argmin}\{k \in \mathbb{N}_0 \colon \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| < \tau\} . \qquad (3.1.5)$$

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

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \leq \tau , \quad \tau \hat{=} \text{ prescribed tolerance } > 0 . \qquad (3.1.6)$$

$\mathbf{x}^*$ 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 (3.1.5), see Rem. 3.1.8.

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 3.1.7: stationary iteration in $\mathbb{M}$, $\rightarrow$ Ex. 3.1.6

```
function x = sqrtit(a)
x_old = -1; x = a;
while (x_old ~= x)
  x_old = x;
  x = 0.5*(x+a/x);
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

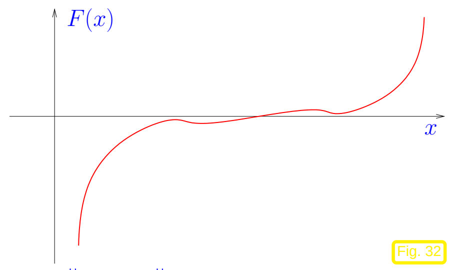


Fig. 32

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

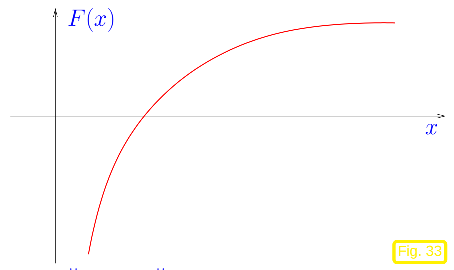


Fig. 33

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

Sometimes extra knowledge about the type/speed of convergence allows to achieve reliable termination in the sense that (3.1.6) can be guaranteed though the number of iterations might be (slightly) too large.

Remark 3.1.8 (A posteriori termination criterion for linearly convergent iterations).

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:

$$\boxed{\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \leq \frac{L}{1-L}\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\|} \;. \qquad (3.1.7)$$

This suggests that we take the right hand side of (3.1.7) as a posteriori error bound.

△

Example 3.1.9 (A posteriori error bound for linearly convergent iteration).

Iteration of Example 3.1.4:

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

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

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

k	$\lvert x^{(k)} - \pi \rvert$	$\frac{L}{1-L}\lvert x^{(k)} - x^{(k-1)} \rvert$	slack of bound
1	2.191562221997101	4.933154875586894	2.741592653589793
2	0.247139097781070	1.944423124216031	1.697284026434961
3	0.122936737876834	0.124202359904236	0.001265622027401
4	0.061390835206217	0.061545902670618	0.000155067464401
5	0.030685773472263	0.030705061733954	0.000019288261691
6	0.015341682696235	0.015344090776028	0.000002408079792
7	0.007670690889185	0.007670991807050	0.000000300917864
8	0.003835326638666	0.003835364250520	0.000000037611854
9	0.001917660968637	0.001917665670029	0.000000004701392
10	0.000958830190489	0.000958830778147	0.000000000587658
11	0.000479415058549	0.000479415131941	0.000000000073392
12	0.000239707524646	0.000239707533903	0.000000000009257
13	0.000119853761949	0.000119853762696	0.000000000000747
14	0.000059926881308	0.000059926880641	0.000000000000667
15	0.000029963440745	0.000029963440563	0.000000000000181

Hence: the a posteriori error bound is highly accurate in this case!

◇

Note: If L not known then using $\widetilde{L} > L$ in error bound is playing safe.

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

$$\succ \underbrace{\boxed{\text{iterates } (\mathbf{x}^{(k)})_{k\in\mathbb{N}_0}\text{:}\quad \mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})}}_{\longrightarrow \text{ 1-point method, } cf. \text{ (3.1.1)}} .$$

Sequence of iterates need not be well defined: $\mathbf{x}^{(k)} \notin U$ possible !

3.2.1 Consistent fixed point iterations

Definition 3.2.1 (Consistency of fixed point iterations, *c.f.* Def. 3.1.2)**.**
A fixed point iteration $\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)})$ *is consistent with* $F(\mathbf{x}) = 0$, *if*

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

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.

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

use the fixed point iteration $\quad \mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})$. (3.2.1)

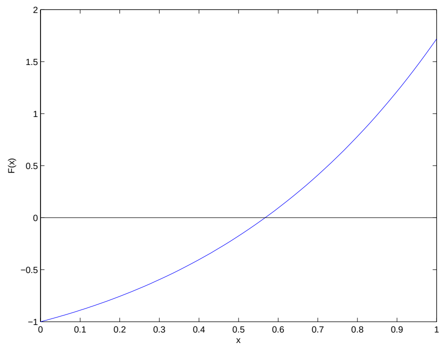
Note: there are *many ways* to transform $F(\mathbf{x}) = 0$ into a fixed point form **!**

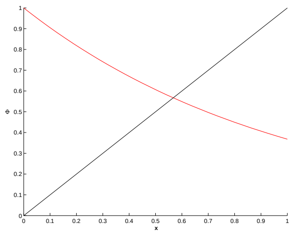
Example 3.2.1 (Options for fixed point iterations).

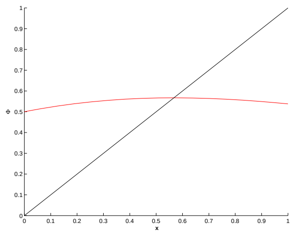
$$F(x) = xe^x - 1 \,, \quad x \in [0,1] \,.$$

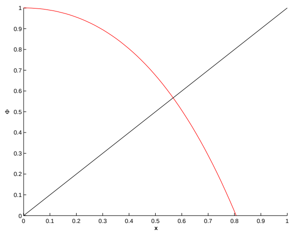
Different fixed point forms:

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




function Φ_1


function Φ_2


function Φ_3

k	$x^{(k+1)} := \Phi_1(x^{(k)})$	$x^{(k+1)} := \Phi_2(x^{(k)})$	$x^{(k+1)} := \Phi_3(x^{(k)})$
0	0.500000000000000	0.500000000000000	0.500000000000000
1	0.606530659712633	0.566311003197218	0.675639364649936
2	0.545239211892605	0.567143165034862	0.347812678511202
3	0.579703094878068	0.567143290409781	0.855321409174107
4	0.560064627938902	0.567143290409784	-0.156505955383169
5	0.571172148977215	0.567143290409784	0.977326422747719
6	0.564862946980323	0.567143290409784	-0.619764251895580
7	0.568438047570066	0.567143290409784	0.713713087416146
8	0.566409452746921	0.567143290409784	0.256626649129847
9	0.567559634262242	0.567143290409784	0.924920676910549
10	0.566907212935471	0.567143290409784	-0.407422405542253

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

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

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

◇

3.2.2 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.

Definition 3.2.2 (Contractive mapping)**.**
$\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ *is contractive (w.r.t. norm* $\|\cdot\|$ *on* $\mathbb{R}^n$*), if*

$$\exists L < 1: \quad \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \leq L\,\|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in U\,. \tag{3.2.2}$$

A simple consideration: if $\Phi(\mathbf{x}^*) = \mathbf{x}^*$ (fixed point), then a fixed point iteration induced by a contractive mapping Φ satisfies

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| = \left\|\Phi(\mathbf{x}^{(k)}) - \Phi(\mathbf{x}^*)\right\| \overset{(3.2.3)}{\leq} L\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| ,$$

that is, the iteration converges (at least) linearly ($\rightarrow$ Def. 3.1.4).

Remark 3.2.2 (Banach's fixed point theorem). $\longrightarrow$ [40, Satz 6.5.2]

A key theorem in calculus (also functional analysis):

Theorem 3.2.3 (Banach's fixed point theorem).
If $D \subset \mathbb{K}^n$ *(*$\mathbb{K} = \mathbb{R}, \mathbb{C}$*) closed 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 $\mathbf{x}^{(k)} = \Phi(\mathbf{x}^{(k-1)})$, $\mathbf{x}^{(0)} \in D$:

$$\left\|\mathbf{x}^{(k+N)} - \mathbf{x}^{(k)}\right\| \leq \sum_{j=k}^{k+N-1} \left\|\mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}\right\| \leq \sum_{j=k}^{k+N-1} L^j \left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\|$$
$$\leq \frac{L^k}{1-L}\left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\| \xrightarrow{k\to\infty} 0\,.$$

$(\mathbf{x}^{(k)})_{k\in\mathbb{N}_0}$ 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 3.2.4 (Sufficient condition for local linear convergence of fixed point iteration)**.**
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* (3.2.1) *converges locally and at least linearly.*

matrix norm, Def. 2.5.2 !

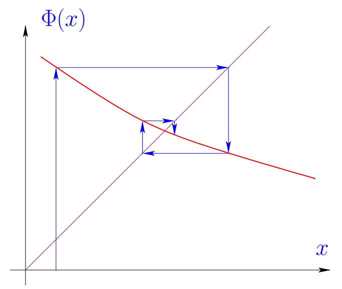
✎ notation: $D\Phi(\mathbf{x}) \hat{=}$ Jacobian (*ger.:* Jacobi-Matrix) of Φ at $\mathbf{x} \in D$
$\longrightarrow$ [40, Sect. 7.6]

Example 3.2.3 (Fixed point iteration in 1D).

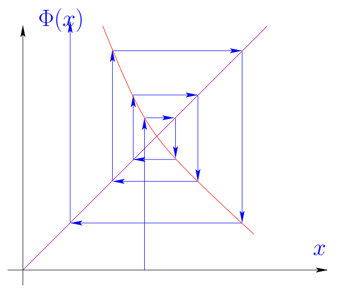
1D setting ($n = 1$): $\Phi : \mathbb{R} \mapsto \mathbb{R}$ continuously differentiable, $\Phi(x^*) = x^*$

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

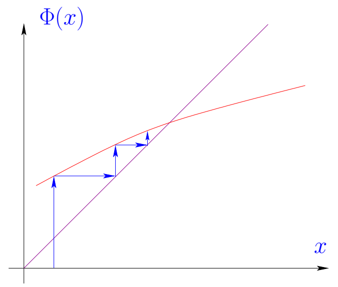
"Visualization" of the statement of Lemma 3.2.4: The iteration converges *locally*, if Φ is flat in a neighborhood of x^*, it will diverge, if Φ is steep there.



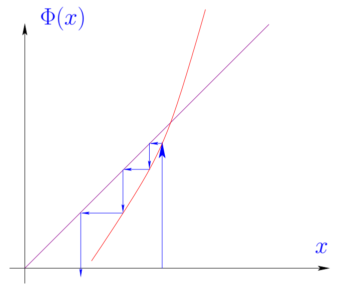
$-1 < \Phi'(x^*) \leq 0$ ➣ convergence



$\Phi'(x^*) < -1$ ➣ divergence



$0 \leq \Phi'(x^*) < 1$ ➣ convergence



$1 < \Phi'(x^*)$ ➣ divergence

◇

Proof. (of Lemma 3.2.4) 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

$$\|\Phi(\mathbf{x}) - \mathbf{x}^*\| - \|D\Phi(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*)\| \leq \psi(\|\mathbf{x} - \mathbf{x}^*\|)\,\|\mathbf{x} - \mathbf{x}^*\|$$

$$\blacktriangleright \quad \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \leq (\psi(t) + \|D\Phi(\mathbf{x}^*)\|)\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \leq L\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \ ,$$

if $\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| < \delta$. □

Contractivity also guarantees the *uniqueness* of a fixed point, see the next Lemma.

Recalling the Banach fixed point theorem Thm. 3.2.3 we see that under some additional (usually mild) assumptions, it also ensures the *existence* of a fixed point.

Lemma 3.2.5 (Sufficient condition for local linear convergence of fixed point iteration (II))**.**
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* $\mathbf{x}^*$ *locally at least linearly.*

Recall: $U \subset \mathbb{R}^n$ convex $:\Leftrightarrow$ $(t\mathbf{x} + (1-t)\mathbf{y}) \in U$ for all $\mathbf{x}, \mathbf{y} \in U$, $0 \le t \le 1$

Proof. (of Lemma 3.2.5) By the mean value theorem

$$\Phi(\mathbf{x}) - \Phi(\mathbf{y}) = \int_0^1 D\Phi(\mathbf{x} + \tau(\mathbf{y}-\mathbf{x}))(\mathbf{y}-\mathbf{x})\,\mathrm{d}\tau \quad \forall \mathbf{x},\mathbf{y} \in \mathrm{dom}(\Phi) .$$

$$\blacktriangleright \qquad \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \le L\,\|\mathbf{y}-\mathbf{x}\| .$$

There is $\delta > 0$: $B := \{\mathbf{x}: \|\mathbf{x}-\mathbf{x}^*\| \le \delta\} \subset \mathrm{dom}(\Phi)$. Φ is contractive on B with unique fixed point $\mathbf{x}^*$.

Remark 3.2.4.
If $0 < \|D\Phi(\mathbf{x}^*)\| < 1$, $\mathbf{x}^{(k)} \approx \mathbf{x}^*$ then the asymptotic rate of linear convergence is $L = \|D\Phi(\mathbf{x}^*)\|$

$\triangle$

Example 3.2.5 (Multidimensional fixed point iteration).

System of equations in fixed point form:

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

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

Choose *appropriate* norm: $\|\cdot\|$ = ∞-norm $\|\cdot\|_\infty$ ($\rightarrow$ Example 2.5.1) ;

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

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

The existence of a fixed point is also guaranteed, because Φ maps into the closed set $[-3,3]^2$. Thus, the Banach fixed point theorem, Thm. 3.2.3, can be applied.

$\diamond$

What about higher order convergence ($\rightarrow$ Def. 3.1.7) ?

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

Theorem 3.2.6 (Taylor's formula). $\rightarrow$ *[40, Sect. 5.5]*
If $\Phi : U \subset \mathbb{R} \mapsto \mathbb{R}$, U interval, is $m+1$ times continuously differentiable, $x \in U$

$$\Phi(y) - \Phi(x) = \sum_{k=1}^{m} \frac{1}{k!}\Phi^{(k)}(x)(y-x)^k + O(|y-x|^{m+1}) \quad \forall y \in U . \tag{3.2.3}$$

Apply Taylor expansion (3.2.3) to iteration function Φ:

If $\Phi(x^*) = x^*$ and $\Phi : \mathrm{dom}(\Phi) \subset \mathbb{R} \mapsto \mathbb{R}$ is "sufficiently smooth"

$$x^{(k+1)} - x^* = \Phi(x^{(k)}) - \Phi(x^*) = \sum_{l=1}^{m} \frac{1}{l!}\Phi^{(l)}(x^*)(x^{(k)} - x^*)^l + O(|x^{(k)} - x^*|^{m+1}) . \tag{3.2.4}$$

Lemma 3.2.7 (Higher order local convergence of fixed point iterations).
If $\Phi : U \subset \mathbb{R} \mapsto \mathbb{R}$ is $m+1$ times continuously differentiable, $\Phi(x^*) = x^*$ for some x^* in the interior of U, and $\Phi^{(l)}(x^*) = 0$ for $l = 1, \dots, m$, $m \ge 1$, then the fixed point iteration (3.2.1) *converges locally to x^* with order $\ge m+1$ ($\rightarrow$ Def. 3.1.7).*

Proof. For neighborhood $\mathcal{U}$ of x^*

$$(3.2.4) \Rightarrow \exists C > 0: \ |\Phi(y) - \Phi(x^*)| \le C\,|y - x^*|^{m+1} \quad \forall y \in \mathcal{U} .$$

$$\delta^m C < 1/2: \ |x^{(0)} - x^*| < \delta \Rightarrow |x^{(k)} - x^*| < 2^{-k}\delta \quad ➢ \text{ local convergence .}$$

Then appeal to (3.2.4) $\square$

Example 3.2.1 continued:

$$\Phi_2'(x) = \frac{1 - xe^x}{(1+e^x)^2} = 0 \ \text{, if } \ xe^x - 1 = 0 \quad \text{hence quadratic convergence ! .}$$

Example 3.2.1 continued: Since $x^*e^{x^*} - 1 = 0$

$$\Phi_1'(x) = -e^{-x} \Rightarrow \Phi_1'(x^*) = -x^* \approx -0.56 \quad \text{hence local linear convergence .}$$

$$\Phi_3'(x) = 1 - xe^x - e^x \Rightarrow \Phi_3'(x^*) = -\frac{1}{x^*} \approx -1.79 \quad \text{hence no convergence .}$$

Remark 3.2.6 (Termination criterion for contractive fixed point iteration).

Recap of Rem. 3.1.8:

Termination criterion for contractive fixed point iteration, *c.f.* (3.2.3), with contraction factor $0 \leq L < 1$:

$$\begin{aligned}\left\|\mathbf{x}^{(k+m)}-\mathbf{x}^{(k)}\right\| &\overset{\triangle\text{-ineq.}}{\leq} \sum_{j=k}^{k+m-1}\left\|\mathbf{x}^{(j+1)}-\mathbf{x}^{(j)}\right\| \leq \sum_{j=k}^{k+m-1} L^{j-k}\left\|\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right\| \\ &= \frac{1-L^m}{1-L}\left\|\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right\| \leq \frac{1-L^m}{1-L}L^{k-l}\left\|\mathbf{x}^{(l+1)}-\mathbf{x}^{(l)}\right\| .\end{aligned}$$

hence for $m \to \infty$, with $\mathbf{x}^* := \lim_{k\to\infty} \mathbf{x}^{(k)}$:

$$\left\|\mathbf{x}^*-\mathbf{x}^{(k)}\right\| \leq \frac{L^{k-l}}{1-L}\left\|\mathbf{x}^{(l+1)}-\mathbf{x}^{(l)}\right\| . \qquad (3.2.5)$$

Set $l = 0$ in (3.2.5)

a priori termination criterion

$$\left\|\mathbf{x}^*-\mathbf{x}^{(k)}\right\| \leq \frac{L^{k}}{1-L}\left\|\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right\| \qquad (3.2.6)$$

Set $l = k-1$ in (3.2.5)

a posteriori termination criterion

$$\left\|\mathbf{x}^*-\mathbf{x}^{(k)}\right\| \leq \frac{L}{1-L}\left\|\mathbf{x}^{(k)}-\mathbf{x}^{(k-1)}\right\| \qquad (3.2.7)$$

△

3.3 Zero Finding

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

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

3.3.1 Bisection

Idea: use ordering of real numbers & intermediate value theorem

Input: $a, b \in I$ such that $F(a)F(b) < 0$
(different signs !)

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

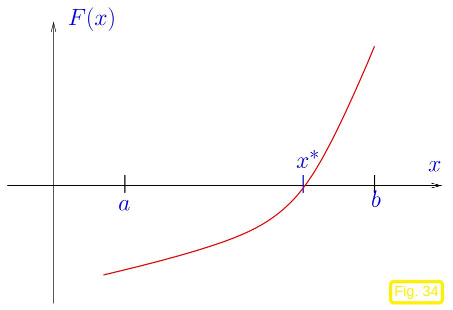


Fig. 34

Algorithm 3.3.1 (Bisection method).

MATLAB-CODE: bisection method

```
function x = bisect(F,a,b,tol)
% Searching zero by bisection
if (a>b), t=a; a=b; b=t; end;
fa = F(a); fb = F(b);
if (fa*fb>0)
  error('f(a), f(b) same sign'); end;
if (fa > 0), v=-1; else v = 1; end
x = 0.5*(b+a);
while((b-a > tol) & ((a<x) & (x<b)))
  if (v*F(x)>0), b=x; else a=x; end;
  x = 0.5*(a+b)
end
```

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

Handle to MATLAB function providing F.

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

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. 3.1.2. The iteration will terminate, when, e.g., $a \widetilde{+} \frac{1}{2}(b-a) = a$, which, by the Ass. 2.4.2 can only happen, when

$$\left|\tfrac{1}{2}(b-a)\right| \leq \texttt{eps} \cdot |a| .$$

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: $|x^{(k)} - x^*| \le 2^{-k}|b-a|$

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

Remark 3.3.2. MATLAB function `fzero` is based on bisection approach. △

3.3.2 Model function methods

$\hat{=}$ class of iterative methods for finding zeroes of F:

Idea: Given: approximate zeroes $x^{(k)}, x^{(k-1)}, \dots, x^{(k-m)}$

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

❷ $x^{(k+1)} :=$ zero of $\widetilde{F}$
(has to be readily available ↔ analytic formula)

Distinguish (see (3.1.1)):

one-point methods : $x^{(k+1)} = \Phi_F(x^{(k)})$, $k \in \mathbb{N}$ (e.g., fixed point iteration → Sect. 3.2)
multi-point methods : $x^{(k+1)} = \Phi_F(x^{(k)}, x^{(k-1)}, \dots, x^{(k-m)})$, $k \in \mathbb{N}$, $m = 2, 3, \dots$.

3.3.2.1 Newton method in scalar case

Assume: $F : I \mapsto \mathbb{R}$ continuously differentiable

model function:= tangent at F in $x^{(k)}$:

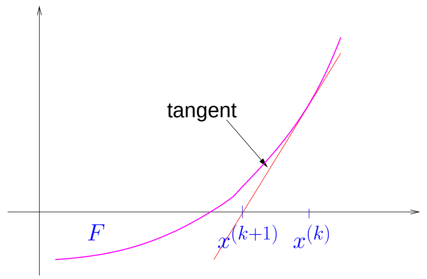
$$\widetilde{F}(x) := F(x^{(k)}) + F'(x^{(k)})(x - x^{(k)})$$

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

We obtain Newton iteration

$$\boxed{x^{(k+1)} := x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})}}, \qquad (3.3.1)$$

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



Example 3.3.3 (Newton method in 1D). (→ Ex. 3.2.1)

Newton iterations for two different scalar non-linear equation with the same solution sets:

$$F(x) = xe^x - 1 \Rightarrow F'(x) = e^x(1+x) \Rightarrow x^{(k+1)} = x^{(k)} - \frac{x^{(k)}e^{x^{(k)}} - 1}{e^{x^{(k)}}(1+x^{(k)})} = \frac{(x^{(k)})^2 + e^{-x^{(k)}}}{1 + x^{(k)}}$$

$$F(x) = x - e^{-x} \Rightarrow F'(x) = 1 + e^{-x} \Rightarrow x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - e^{-x^{(k)}}}{1 + e^{-x^{(k)}}} = \frac{1 + x^{(k)}}{1 + e^{x^{(k)}}}$$

Ex. 3.2.1 shows quadratic convergence ! (→ Def. 3.1.7)

◇

Newton iteration (3.3.1) $\hat{=}$ fixed point iteration (→ Sect.3.2) with iteration function

$$\Phi(x) = x - \frac{F(x)}{F'(x)} \Rightarrow \Phi'(x) = \frac{F(x)F''(x)}{(F'(x))^2} \Rightarrow \Phi'(x^*) = 0\,, \quad \text{, if } F(x^*) = 0,\ F'(x^*) \neq 0\,.$$

From Lemma 3.2.7:

Newton method locally quadratically convergent (→ Def. 3.1.7) to zero x^*, if $F'(x^*) \neq 0$

3.3.2.2 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 3.3.4 (Halley's iteration).

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

k	$x^{(k)}$	$F(x^{(k)})$	$x^{(k)} - x^{(k-1)}$	$x^{(k)} - x^*$
1	0.19865959351191	10.90706835180178	-0.19865959351191	-0.84754290138257
2	0.69096314049024	0.94813655914799	-0.49230354697833	-0.35523935440424
3	1.02335017694603	0.03670912956750	-0.33238703645579	-0.02285231794846
4	1.04604398836483	0.00024757037430	-0.02269381141880	-0.00015850652965
5	1.04620248685303	0.00000001255745	-0.00015849848821	-0.00000000804145

Compare with Newton method (3.3.1) 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. 3.3.3) ◇

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

$$\frac{d}{dy}(\widehat{F}^{-1})(y) = \frac{1}{\widehat{F}'(\widehat{F}^{-1}(y))} \quad \Rightarrow \quad g'(x) = \frac{1}{\widehat{F}'(g(x))} \cdot F'(x) \;.$$

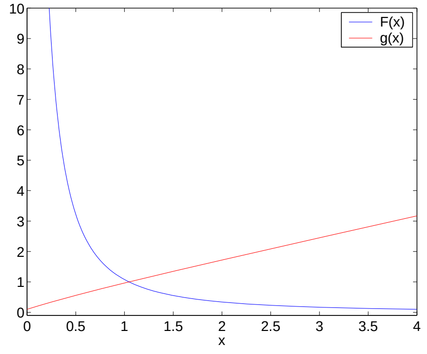
Example 3.3.5 (Adapted Newton method).

As in Ex. 3.3.4: $F(x) = \frac{1}{(x+1)^2} + \frac{1}{(x+0.1)^2} - 1 \;, \quad x > 0 :$

Observation:

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

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



Idea: instead of $F(x) \overset{!}{=} 0$ tackle $g(x) \overset{!}{=} 1$ with Newton's method (3.3.1).

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

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

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

k	$x^{(k)}$	$F(x^{(k)})$	$x^{(k)}-x^{(k-1)}$	$x^{(k)}-x^*$
1	0.91312431341979	0.24747993091128	0.91312431341979	-0.13307818147469
2	1.04517022155323	0.00161402574513	0.13204590813344	-0.00103227334125
3	1.04620244004116	0.00000008565847	0.00103221848793	-0.00000005485332
4	1.04620249489448	0.00000000000000	0.00000005485332	-0.00000000000000

◇

For zero finding there is wealth of iterative methods that offer higher order of convergence.

One idea: consistent modification of the Newton-Iteration:

➤ fixed point iteration : $\Phi(x) = x - \dfrac{F(x)}{F'(x)}H(x)$ with "proper" $H : I \mapsto \mathbb{R}$.

Aim: find H such that the method is of p-th order; tool: Lemma 3.2.7.

Assume: F smooth "enough" and $\exists\, x^* \in I$: $F(x^*) = 0$, $F'(x^*) \neq 0$.

$$\Phi = x - uH \quad , \quad \Phi' = 1 - u'H - uH' \quad , \quad \Phi'' = -u''H - 2u'H - uH'' \,,$$

$$\text{with } u = \frac{F}{F'} \quad \Rightarrow \quad u' = 1 - \frac{FF''}{(F')^2} \quad , \quad u'' = -\frac{F''}{F'} + 2\frac{F(F'')^2}{(F')^3} - \frac{FF'''}{(F')^2} \,.$$

$F(x^*) = 0$ ➤ $u(x^*) = 0$, $u'(x^*) = 1$, $u''(x^*) = -\dfrac{F''(x^*)}{F'(x^*)}$.

$$\Phi'(x^*) = 1 - H(x^*) \quad , \quad \Phi''(x^*) = \frac{F''(x^*)}{F'(x^*)}H(x^*) - 2H'(x^*) \,. \tag{3.3.2}$$

Lemma 3.2.7 ➢ **Necessary** conditions for local convergencd of order p:

$p = 2$ (quadratical convergence): $H(x^*) = 1$,

$p = 3$ (cubic convergence): $H(x^*) = 1 \;\wedge\; H'(x^*) = \dfrac{1}{2}\dfrac{F''(x^*)}{F'(x^*)}$.

In particular: $H(x) = G(1 - u'(x))$ with "proper" G

➤ fixed point iteration
$$x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})} G\left(\frac{F(x^{(k)})F''(x^{(k)})}{(F'(x^{(k)}))^2}\right) . \tag{3.3.3}$$

Lemma 3.3.1. *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* (3.3.3) *converge locally cubically to* x^*.

Proof: Lemma 3.2.7, (3.3.2) and

$$H(x^*) = G(0) \quad , \quad H'(x^*) = -G'(0)u''(x^*) = G'(0)\frac{F''(x^*)}{F'(x^*)} \,.$$

Example 3.3.6 (Example of modified Newton method).

- $G(t) = \dfrac{1}{1 - \frac{1}{2}t}$ ➡ Halley's iteration (→ Ex. 3.3.4)
- $G(t) = \dfrac{2}{1 + \sqrt{1 - 2t}}$ ➡ Euler's iteration
- $G(t) = 1 + \frac{1}{2}t$ ➡ quadratic inverse interpolation

Numerical experiment:

$$F(x) = xe^x - 1 \,, \quad x^{(0)} = 5$$

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
6		0.14709716035310	
7		0.00109463314926	
8		0.00000000107549	

◇

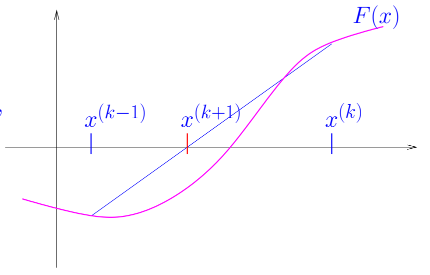
3.3.2.3 Multi-point methods

Idea: Replace F with interpolating polynomial
producing interpolatory model function methods

Simplest representative: secant method

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

$$s(x) = F(x^{(k)}) + \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}(x - x^{(k)}) \,, \tag{3.3.4}$$



➤
$$x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})(x^{(k)} - x^{(k-1)})}{F(x^{(k)}) - F(x^{(k-1)})} \,. \tag{3.3.5}$$

secant method
(MATLAB implementation)

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

Code 3.3.7: secant method

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

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

k	$x^{(k)}$	$F(x^{(k)})$	$e^{(k)} := x^{(k)} - x^*$	$\frac{\log\lvert e^{(k+1)}\rvert - \log\lvert e^{(k)}\rvert}{\log\lvert e^{(k)}\rvert - \log\lvert e^{(k-1)}\rvert}$
2	0.00673794699909	-0.99321649977589	-0.56040534341070	
3	0.01342122983571	-0.98639742654892	-0.55372206057408	24.43308649757745
4	0.98017620833821	1.61209684919288	0.41303291792843	2.70802321457994
5	0.38040476787948	-0.44351476841567	-0.18673852253030	1.48753625853887
6	0.50981028847430	-0.15117846201565	-0.05733300193548	1.51452723840131
7	0.57673091089295	0.02670169957932	0.00958762048317	1.70075240166256
8	0.56668541543431	-0.00126473620459	-0.00045787497547	1.59458505614449
9	0.56713970649585	-0.00000990312376	-0.00000358391394	1.62641838319117
10	0.56714329175406	0.00000000371452	0.00000000134427	
11	0.56714329040978	-0.00000000000001	-0.00000000000000	

A startling observation: the method seems to have a *fractional* (!) order of convergence, see Def. 3.1.7.

◇

Remark 3.3.9 (Fractional order of convergence of secant method).

Indeed, a fractional order of convergence can be proved for the secant method, see[23, Sect. 18.2]. Here is a crude outline of the reasoning:

Asymptotic convergence of secant method: error $e^{(k)} := x^{(k)} - x^*$

$$e^{(k+1)} = \Phi(x^* + e^{(k)}, x^* + e^{(k-1)}) - x^* \quad \text{, with} \quad \Phi(x,y) := x - \frac{F(x)(x-y)}{F(x)-F(y)}\,. \tag{3.3.6}$$

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:

$$e^{(k+1)} \doteq \frac{1}{2}\frac{F''(x^*)}{F'(x^*)} e^{(k)} e^{(k-1)} = C e^{(k)} e^{(k-1)}\,. \tag{3.3.7}$$

Try $e^{(k)} = K(e^{(k-1)})^p$ to determine the order of convergence ($\rightarrow$ Def. 3.1.7):

$$\Rightarrow \quad e^{(k+1)} = K^{p+1}(e^{(k-1)})^{p^2}$$

$$\Rightarrow \quad (e^{(k-1)})^{p^2-p-1} = K^{-p}C \;\Rightarrow\; p^2 - p - 1 = 0 \;\Rightarrow\; p = \tfrac{1}{2}(1 \pm \sqrt{5})\,.$$

As $e^{(k)} \rightarrow 0$ for $k \rightarrow \infty$ we get the order of convergence $p = \frac{1}{2}(1+\sqrt{5}) \approx 1.62$ (see Ex. 3.3.8 !)

△

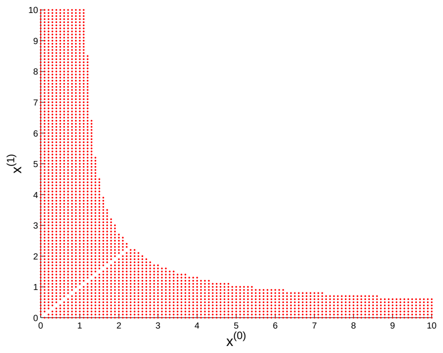
Example 3.3.10 (local convergence of secant method).

$$F(x) = \arctan(x)$$

$\cdot \;\hat{=}$ secant method converges for a pair $(x^{(0)}, x^{(1)})$ of initial guesses.

= local convergence $\rightarrow$ Def. 3.1.3

▷



◇

Another class of multi-point methods: *inverse interpolation*

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

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

- Interpolate F^{-1} by polynomial p of degree d determined by

$$p(F(x^{(k-m)})) = x^{(k-m)} \,, \quad m = 0, \dots, d \,.$$

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

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

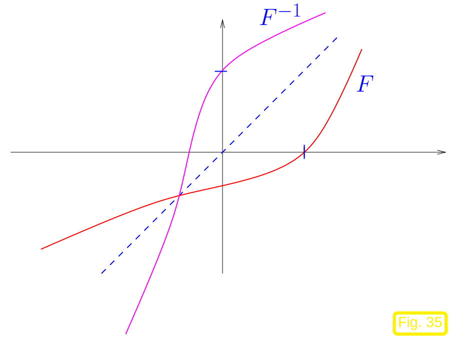


Fig. 35

Case $m = 1$ ➢ secant method

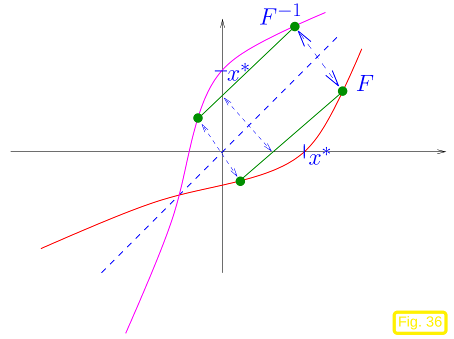


Fig. 36

Case $m = 2$: quadratic inverse interpolation, see [27, 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);
```

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

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

Example 3.3.11 (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. 3.3.9.

$\diamond$

3.3.3 Note on Efficiency

Efficiency of an iterative method (for solving $F(\mathbf{x}) = 0$)	$\leftrightarrow$	computational effort to reach prescribed number of significant digits in result.

Abstract: $W \hat{=}$ computational effort per step

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

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

$$\left\| e^{(k)} \right\| \leq \rho \left\| e^{(0)} \right\| \,, \quad \rho > 0 \text{ prescribed ?} \tag{3.3.8}$$

Error recursion can be converted into expressions (3.3.9) and (3.3.10) 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 (3.3.8) for iterative method of order p, $p \geq 1$ ($\rightarrow$ Def. 3.1.7):

$$\exists C > 0\text{:} \quad \left\| e^{(k)} \right\| \leq C \left\| e^{(k-1)} \right\|^p \quad \forall k \geq 1 \quad (C < 1 \text{ for } p = 1) \,.$$

Assuming $C\left\|e^{(0)}\right\|^{p-1} < 1$ (guarantees convergence):

$$p=1: \quad \left\|e^{(k)}\right\| \overset{!}{\leq} C^k \left\|e^{(0)}\right\| \quad \text{requires} \quad k \geq \frac{\log\rho}{\log C}\ , \tag{3.3.9}$$

$$p>1: \quad \left\|e^{(k)}\right\| \overset{!}{\leq} C^{\frac{p^k-1}{p-1}} \left\|e^{(0)}\right\|^{p^k} \quad \text{requires} \quad p^k \geq 1 + \frac{\log\rho}{\log C/_{p-1} + \log(\left\|e^{(0)}\right\|)}$$

$$\Rightarrow \quad k \geq \log(1+\frac{\log\rho}{\log L_0})/\log p\ , \tag{3.3.10}$$

$$L_0 := C^{1/p-1}\left\|e^{(0)}\right\| < 1\ .$$

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.

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

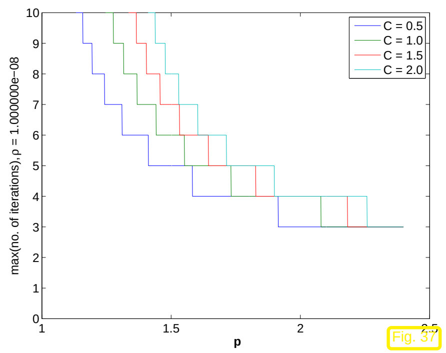
Measure for efficiency:

$$\boxed{\text{Efficiency} := \frac{\text{no. of digits gained}}{\text{total work required}} = \frac{|\log\rho|}{k(\rho)\cdot W}} \tag{3.3.11}$$

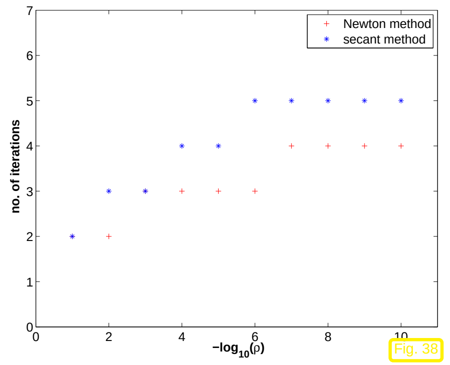
▶ **asymptotic** efficiency w.r.t. $\rho \to 0$ ➜ $|\log\rho| \to \infty$):

$$\text{Efficiency}_{|\rho\to 0} = \begin{cases} -\dfrac{\log C}{W} & \text{, if } \quad p=1\ , \\ \dfrac{\log p|\log\rho|}{W\log|\log\rho|} & \text{, if } \quad p>1\ . \end{cases} \tag{3.3.12}$$

Example 3.3.12 (Efficiency of iterative methods).



Evaluation (3.3.10) for $\left\|e^{(0)}\right\| = 0.1$, $\rho = 10^{-8}$



Newton's method ↔ 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!

◇

3.4 Newton's Method

Non-linear system of equations: for $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ find $\mathbf{x}^* \in D$: $\quad F(\mathbf{x}^*) = 0$

Assume: $\quad F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n \quad$ continuously differentiable

3.4.1 The Newton iteration

Idea (→ Sect. 3.3.2.1): 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), $DF(\mathbf{x}) = \left(\dfrac{\partial F_j}{\partial x_k}(\mathbf{x})\right)_{j,k=1}^{n}$.

▶ Newton iteration: $\quad$ (↔ (3.3.1) for $n=1$)

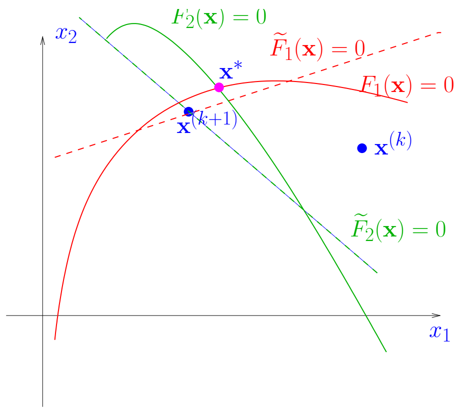
$$\boxed{\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - DF(\mathbf{x}^{(k)})^{-1}F(\mathbf{x}^{(k)})}\ , \quad [\text{ if } DF(\mathbf{x}^{(k)}) \text{ regular }] \tag{3.4.1}$$

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

Illustration of idea of Newton's method for $n = 2$: ▷

Sought: intersection point $\mathbf{x}^*$ of the curves $F_1(\mathbf{x}) = 0$ and $F_2(\mathbf{x}) = 0$.

Idea: $\mathbf{x}^{(k+1)}$ = the intersection of two straight lines (= zero sets of the components of the model function, *cf.* Ex. 2.5.11) that are approximations of the original curves



MATLAB template for Newton method:

Solve linear system:

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

Example 3.4.1 (Newton 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): $DF(\mathbf{x}) = \begin{pmatrix} \partial_{x_1} F_1(x) & \partial_{x_2} F_1(x) \\ \partial_{x_1} F_2(x) & \partial_{x_2} F_2(x) \end{pmatrix} = \begin{pmatrix} 2x_1 & -4x_2^3 \\ 1 & -3x_2^2 \end{pmatrix}$

Realization of Newton iteration (3.4.1):

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 $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}$

Code 3.4.2: Newton iteration in 2D

```
F = @(x) [ x(1)^2-x(2)^4; x(1)-x(2)^3 ];
DF = @(x) [2*x(1), -4*x(2)^3; 1, -3*x(2)^2];
x = [0.7; 0.7]; x_ast = [1;1]; tol = 1E-10;

res = [0,x',norm(x-x_ast)];
s = feval(DF,x)\feval(F,x); x = x-s;
res = [res; 1,x',norm(x-x_ast)]; k=2;
while (norm(s) > tol*norm(x))
  s = DF(x)\F(x); x = x-s;
  res = [res; k,x',norm(x-x_ast)];
  k = k+1;
end

logdiff = diff(log(res(:,4)));
rates = logdiff(2:end)./logdiff(1:end-1);
```

k	$\mathbf{x}^{(k)}$	$\epsilon_k := \|\mathbf{x}^* - \mathbf{x}^{(k)}\|_2$
0	$(0.7,\ 0.7)^T$	4.24e-01
1	$(0.87850000000000,\ 1.06428571428571)^T$	1.37e-01
2	$(1.01815943274188,\ 1.00914882463936)^T$	2.03e-02
3	$(1.00023355916300,\ 1.00015913936075)^T$	2.83e-04
4	$(1.00000000583852,\ 1.00000002726552)^T$	2.79e-08
5	$(0.999999999999998,\ 1.000000000000000)^T$	2.11e-15
6	$(1,\ 1)^T$	

◇

!

New aspect for $n \gg 1$ (compared to $n = 1$-dimensional case, section. 3.3.2.1):

Computation of the Newton correction is eventually costly!

Remark 3.4.3 (Affine invariance of Newton method).

An important property of the Newton iteration (3.4.1): affine invariance $\longrightarrow$ [11, Sect .1.2.2]

set $G(\mathbf{x}) := \mathbf{A}F(\mathbf{x})$ with regular $\mathbf{A} \in \mathbb{R}^{n,n}$ so that $F(\mathbf{x}^*) = 0 \quad \Leftrightarrow \quad G(\mathbf{x}^*) = 0$.

affine invariance: Newton iteration for $G(\mathbf{x}) = 0$ is the same for all regular $\mathbf{A}$!

This is a simple computation:

$$DG(\mathbf{x}) = \mathbf{A}DF(\mathbf{x}) \quad\Rightarrow\quad DG(\mathbf{x})^{-1}G(\mathbf{x}) = DF(\mathbf{x})^{-1}\mathbf{A}^{-1}\mathbf{A}F(\mathbf{x}) = DF(\mathbf{x})^{-1}F(\mathbf{x}) .$$

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.

$\triangle$

Remark 3.4.4 (Differentiation rules). $\rightarrow$ Repetition: basic analysis

Statement of the Newton iteration (3.4.1) 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*

linear mapping $\quad DF(\mathbf{x}) : V \mapsto W$,

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

$$D(G \circ F)(\mathbf{x})\mathbf{h} = DG(F(\mathbf{x}))(DF(\mathbf{x}))\mathbf{h} , \quad \mathbf{h} \in V , \mathbf{x} \in D . \tag{3.4.2}$$

- Product rule: $F : D \subset V \mapsto W$, $G : D \subset V \mapsto U$ sufficiently smooth, $b : W \times U \mapsto Z$ bilinear

$$T(\mathbf{x}) = b(F(\mathbf{x}), G(\mathbf{x})) \quad\Rightarrow\quad \begin{aligned} &DT(\mathbf{x})\mathbf{h} = b(DF(\mathbf{x})\mathbf{h}, G(\mathbf{x})) + b(F(\mathbf{x}), DG(\mathbf{x})\mathbf{h}) , \\ &\mathbf{h} \in V , \mathbf{x} \in D . \end{aligned} \tag{3.4.3}$$

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

$$\mathbf{grad}\, F(\mathbf{x})^T\mathbf{h} := DF(\mathbf{x})\mathbf{h} , \qquad \mathbf{h}_1^T HF(\mathbf{x})\mathbf{h}_2 := D(DF(\mathbf{x})(\mathbf{h}_1))(\mathbf{h}_2) , \quad \mathbf{h}, \mathbf{h}_1, \mathbf{h}_2 \in V .$$

Remark 3.4.5 (Simplified Newton method).

Simplified Newton Method: use the same $DF(\mathbf{x}^{(k)})$ for more steps
➣ (usually) merely linear convergence instead of quadratic convergence

$\triangle$

Remark 3.4.6 (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 **!**

$\triangle$

Example 3.4.7 (Roundoff errors and difference quotients).

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
```

$\log_{10}(h)$	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

Recorded relative error, $f(x) = e^x$, $x = 0$ $\quad\triangleright$

Note: An analysis based on expressions for remainder terms of Taylor expansions shows that the approximation error cannot be blamed for the loss of accuracy as $h \rightarrow 0$ (as expected).

Explanation relying on roundoff error analysis, see Sect. 2.4:

MATLAB-CODE: Numerical differentiation of exp(x)

```
h = 0.1; x = 0.0;
for i = 1:16
    df =    (exp(x+h)-exp(x))/h;
    fprintf('%d %16.14f\n',i,df-1);
    h = h*0.1;
end
```

Obvious cancellation $\longrightarrow$ 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 $f(x) = \exp(x)$:

$$\mathtt{df} = \frac{e^{x+h}\boxed{(1+\delta_1)} - e^x\boxed{(1+\delta_2)}}{h}$$

correction factors take into account roundoff:
(→ "'axiom of roundoff analysis", Ass. 2.4.2)

$$= e^x\left(\frac{e^h-1}{h} + \frac{\delta_1 e^h - \delta_2}{h}\right) \qquad |\delta_1|, |\delta_2| \le \mathtt{eps} .$$

$1 + O(h)$ $\quad$ $O(h^{-1})$ für $h \to 0$

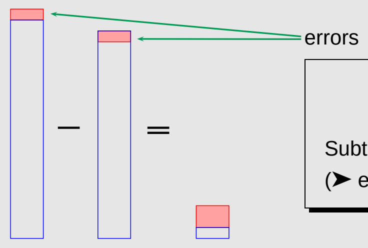
$$\Rightarrow \quad |\mathtt{df}| \le e^x\left(\tfrac{e^h-1}{h} + \mathtt{eps}\,\tfrac{1+e^h}{h}\right)$$

▶ relative error: $\left|\dfrac{e^x - \mathtt{df}}{e^x}\right| \approx h + \dfrac{2\mathtt{eps}}{h} \to \min$ for $h = \sqrt{2\,\mathtt{eps}}$.

In double precision: $\sqrt{2\mathtt{eps}} = 2.107342425544702 \cdot 10^{-8}$

◇

What is this mysterious cancellation (*ger.:* Auslöschung) **?**



Cancellation
$\hat{=}$
Subtraction of almost equal numbers
(➤ extreme amplification of relative errors)

Example 3.4.8 (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

◇

3.4.2 Convergence of Newton's method

Newton iteration (3.4.1) $\hat{=}$ fixed point iteration (→ Sect. 3.2) with

$$\Phi(\mathbf{x}) = \mathbf{x} - DF(\mathbf{x})^{-1}F(\mathbf{x}) .$$

▼

["product rule" : $D\Phi(\mathbf{x}) = \mathbf{I} - D(DF(\mathbf{x})^{-1})F(\mathbf{x}) - DF(\mathbf{x})^{-1}DF(\mathbf{x})$]

▼

$$F(\mathbf{x}^*) = 0 \quad \Rightarrow \quad D\Phi(\mathbf{x}^*) = 0 .$$

Lemma 3.2.7 suggests conjecture:

Local quadratic convergence of Newton's method, if $DF(\mathbf{x}^*)$ regular

Example 3.4.9 (Convergence of Newton's method).

Ex. 3.4.1 cnt'd: record of iteration errors, see Code 3.4.1:

k	$\mathbf{x}^{(k)}$	$\epsilon_k := \lVert\mathbf{x}^* - \mathbf{x}^{(k)}\rVert_2$	$\dfrac{\log\epsilon_{k+1} - \log\epsilon_k}{\log\epsilon_k - \log\epsilon_{k-1}}$
0	$(0.7,\ 0.7)^T$	4.24e-01	
1	$(0.87850000000000,\ 1.064285714285714)^T$	1.37e-01	1.69
2	$(1.01815943274188,\ 1.00914882463936)^T$	2.03e-02	2.23
3	$(1.00023355916300,\ 1.00015913936075)^T$	2.83e-04	2.15
4	$(1.00000000583852,\ 1.00000002726552)^T$	2.79e-08	1.77
5	$(0.999999999999998,\ 1.000000000000000)^T$	2.11e-15	
6	$(1,\ 1)^T$		

◇

There is a sophisticated theory about the convergence of Newton's method. For example one can find the following theorem in [13, Thm. 4.10], [11, Sect. 2.1]):

Theorem 3.4.1 (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) $\exists L \geq 0$: $\left\lVert DF(\mathbf{x})^{-1}(DF(\mathbf{x}+\mathbf{v}) - DF(\mathbf{x}))\right\rVert_2 \leq L\,\lVert\mathbf{v}\rVert_2 \quad \begin{matrix}\forall \mathbf{v} \in \mathbb{R}^n, \mathbf{v}+\mathbf{x} \in D, \\ \forall \mathbf{x} \in D\end{matrix}$,

(E) $\exists \mathbf{x}^*$: $F(\mathbf{x}^*) = 0$ *(existence of solution in* D*)*

(F) initial guess $\mathbf{x}^{(0)} \in D$ *satisfies* $\rho := \left\lVert \mathbf{x}^* - \mathbf{x}^{(0)}\right\rVert_2 < \dfrac{2}{L} \ \wedge\ B_\rho(\mathbf{x}^*) \subset D$.

then *the Newton iteration* (3.4.1) *satisfies:*

(i) $\mathbf{x}^{(k)} \in B_\rho(\mathbf{x}^*) := \{\mathbf{y} \in \mathbb{R}^n,\ \lVert\mathbf{y} - \mathbf{x}^*\rVert < \rho\}$ *for all* $k \in \mathbb{N}$,

(ii) $\lim\limits_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$,

(iii) $\left\lVert\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\rVert_2 \leq \frac{L}{2}\left\lVert\mathbf{x}^{(k)} - \mathbf{x}^*\right\rVert_2^2$ *(local quadratic convergence) .*

✎ notation: ball $B_\rho(\mathbf{z}) := \{\mathbf{x} \in \mathbb{R}^n\colon\ \lVert\mathbf{x} - \mathbf{z}\rVert_2 \leq \rho\}$

Terminology: (D) $\hat{=}$ affine invariant Lipschitz condition

Problem: Usually neither ω nor x^* are known !

➤ In general: a priori estimates as in Thm. 3.4.1 are of little practical relevance.

3.4.3 Termination of Newton iteration

A first viable idea:

Asymptotically due to quadratic convergence:

$$\left\lVert\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\rVert \ll \left\lVert\mathbf{x}^{(k)} - \mathbf{x}^*\right\rVert \quad\Rightarrow\quad \left\lVert\mathbf{x}^{(k)} - \mathbf{x}^*\right\rVert \approx \left\lVert\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\rVert . \tag{3.4.4}$$

➢ quit iterating as soon as $\left\lVert\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\rVert = \left\lVert DF(\mathbf{x}^{(k)})^{-1}F(\mathbf{x}^{(k)})\right\rVert < \tau\left\lVert\mathbf{x}^{(k)}\right\rVert$,

with τ = tolerance

$\longrightarrow$ uneconomical: one needless update, because $\mathbf{x}^{(k)}$ already accurate enough **!**

Remark 3.4.10. 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:

$$\boxed{DF(\mathbf{x}^{(k-1)}) \approx DF(\mathbf{x}^{(k)})\text{: quit as soon as } \left\lVert DF(\mathbf{x}^{(k-1)})^{-1}F(\mathbf{x}^{(k)})\right\rVert < \tau\left\lVert\mathbf{x}^{(k)}\right\rVert}$$

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 (3.4.4).

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. 2.2.6) 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.

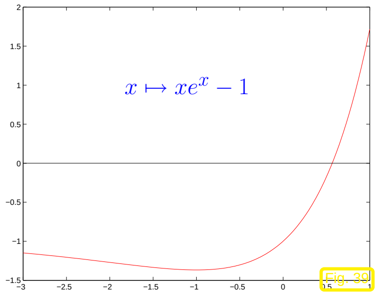
3.4.4 Damped Newton method

Example 3.4.11 (Local convergence of Newton's method).

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

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

◇

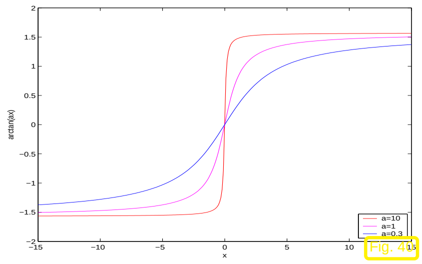


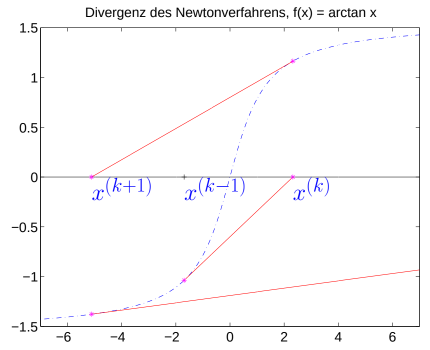
Example 3.4.12 (Region of convergence of Newton method).

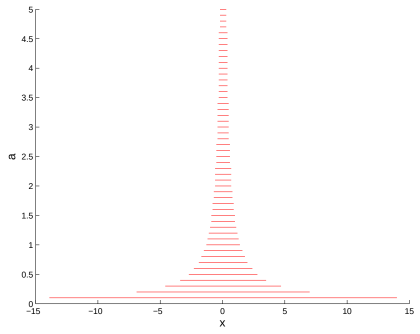
$$F(x) = \arctan(ax) \,, \quad a > 0, x \in \mathbb{R}$$

with zero $x^* = 0$.

◇







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

➤ we observe "overshooting" of Newton correction

Idea: damping of Newton correction:

$$\text{With} \quad \lambda^{(k)} > 0: \quad \mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \lambda^{(k)}\, DF(\mathbf{x}^{(k)})^{-1}F(\mathbf{x}^{(k)}) \,. \tag{3.4.5}$$

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

Choice of damping factor: affine invariant natural monotonicity test [11, Ch. 3]

$$\text{"maximal"} \quad \lambda^{(k)} > 0: \quad \left\| \Delta\bar{\mathbf{x}}(\lambda^{(k)}) \right\| \leq \left(1 - \frac{\lambda^{(k)}}{2}\right) \left\| \Delta\mathbf{x}^{(k)} \right\|_2 \tag{3.4.6}$$

where $\Delta\mathbf{x}^{(k)} := DF(\mathbf{x}^{(k)})^{-1}F(\mathbf{x}^{(k)})$ → current Newton correction ,
$\Delta\bar{\mathbf{x}}(\lambda^{(k)}) := DF(\mathbf{x}^{(k)})^{-1}F(\mathbf{x}^{(k)} + \lambda^{(k)}\Delta\mathbf{x}^{(k)})$ → tentative simplified Newton correction .

Heuristics: convergence ⇔ size of Newton correction decreases

Code 3.4.14: 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. 2.2.6

a-posteriori termination criterion (based on simplified Newton correction, *cf.* Sect. 3.4.3)

Natural monotonicity test (3.4.6)

Reduce damping factor λ

Policy: Reduce damping factor by a factor $q \in]0,1[$ (usually $q = \frac{1}{2}$) until the affine invariant natural monotonicity test (3.4.6) passed.

Example 3.4.15 (Damped Newton method). (⟶ Ex. 3.4.12)

$F(x) = \arctan(x)$,

- $x^{(0)} = 20$
- $q = \frac{1}{2}$
- `LMIN = 0.001`

Observation: asymptotic quadratic convergence

k	$\lambda^{(k)}$	$x^{(k)}$	$F(x^{(k)})$
1	0.03125	0.94199967624205	0.75554074974604
2	0.06250	0.85287592931991	0.70616132170387
3	0.12500	0.70039827977515	0.61099321623952
4	0.25000	0.47271811131169	0.44158487422833
5	0.50000	0.20258686348037	0.19988168667351
6	1.00000	-0.00549825489514	-0.00549819949059
7	1.00000	0.00000011081045	0.00000011081045
8	1.00000	-0.00000000000001	-0.00000000000001

◇

Example 3.4.16 (Failure of damped Newton method).

- As in Ex. 3.4.11: $F(x) = xe^x - 1$,
- Initial guess for damped Newton method $x^{(0)} = -1.5$

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`!			

Observation: Newton correction pointing in “wrong direction” so no convergence.

◇

3.4.5 Quasi-Newton Method

What to do when $DF(\mathbf{x})$ is not available and numerical differentiation (see remark 3.4.6) is too expensive**?**

Idea: in one dimension ($n = 1$) apply the secant method (3.3.4) of section 3.3.2.3

$$F'(x^{(k)}) \approx \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \quad \text{"difference quotient"} \qquad (3.4.7)$$

already computed ! ⟶ cheap

Generalisation for $n > 1$ **?**

Idea: rewrite (3.4.7) as a secant condition for the approximation $\mathbf{J}_k \approx DF(\mathbf{x}^{(k)})$, $\mathbf{x}^{(k)} \hat{=}$ iterate:

$$\mathbf{J}_k(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}) = F(\mathbf{x}^{(k)}) - F(\mathbf{x}^{(k-1)}) \; . \qquad (3.4.8)$$

BUT: many matrices $\mathbf{J}_k$ fulfill (3.4.8)

Hence: we need more conditions for $\mathbf{J}_k \in \mathbb{R}^{n,n}$

Idea: get $\mathbf{J}_k$ by a modification of $\mathbf{J}_{k-1}$

Broyden conditions: $\mathbf{J}_k\mathbf{z} = \mathbf{J}_{k-1}\mathbf{z} \quad \forall \mathbf{z}\colon \mathbf{z} \perp (\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})$. (3.4.9)

i.e.:

$$\boxed{\mathbf{J}_k := \mathbf{J}_{k-1} + \frac{F(\mathbf{x}^{(k)})(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})^T}{\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\|_2^2}} \qquad (3.4.10)$$

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

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}, \; \Delta\mathbf{x}^{(k)} := -\mathbf{J}_k^{-1}F(\mathbf{x}^{(k)}) \, , \; \mathbf{J}_{k+1} := \mathbf{J}_k + \frac{F(\mathbf{x}^{(k+1)})(\Delta\mathbf{x}^{(k)})^T}{\left\|\Delta\mathbf{x}^{(k)}\right\|_2^2} \qquad (3.4.11)$$

Initialize $\mathbf{J}_0$ e.g. with the exact Jacobi matrix $DF(\mathbf{x}^{(0)})$.

Remark 3.4.17 (Minimal property of Broydens rank 1 modification).

Let $\mathbf{J} \in \mathbb{R}^{n,n}$ fulfill (3.4.8) and $\mathbf{J}_k$, $\mathbf{x}^{(k)}$ from (3.4.11) 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,
(3.4.10) gives the $\|\cdot\|_2$-minimal relative correction of $\mathbf{J}_{k-1}$, such that the secant condition (3.4.8) holds.

$\triangle$

Example 3.4.18 (Broydens Quasi-Newton Method: Convergence).

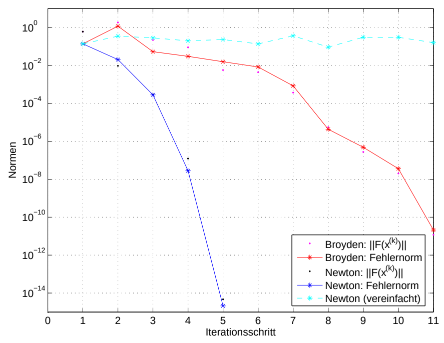
- In the non-linear system of the example 3.4.1, $n=2$ take $\mathbf{x}^{(0)}=(0.7.0.7)^T$ and $\mathbf{J}_0 = DF(\mathbf{x}^{(0)})$

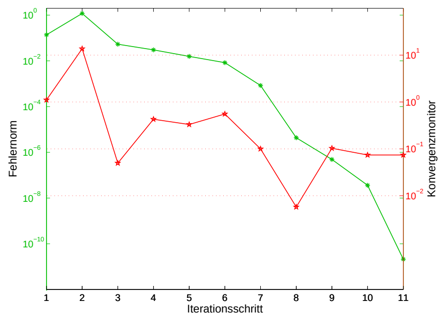
The numerical example shows that the method is:

slower than Newton method (3.4.1), but

better than simplified Newton method (see remark. 3.4.5)

$\triangleright$





convergence monitor

=

quantity that displays 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 $\mu > 1$

$\diamond$

Remark 3.4.19. Option: damped Broyden method (as for the Newton method, section 3.4.4)

$\triangle$

Implementation of (3.4.11): with Sherman-Morrison-Woodbury Update-Formula

$$\mathbf{J}_{k+1}^{-1} = \left(\mathbf{I} - \frac{\mathbf{J}_k^{-1}F(\mathbf{x}^{(k+1)})(\Delta\mathbf{x}^{(k)})^T}{\left\|\Delta\mathbf{x}^{(k)}\right\|_2^2 + \Delta\mathbf{x}^{(k)}\cdot\mathbf{J}_k^{-1}F(\mathbf{x}^{(k+1)})}\right)\mathbf{J}_k^{-1} = \left(\mathbf{I} + \frac{\Delta\mathbf{x}^{(k+1)}(\Delta\mathbf{x}^{(k)})^T}{\left\|\Delta\mathbf{x}^{(k)}\right\|_2^2}\right)\mathbf{J}_k^{-1} \tag{3.4.12}$$

that makes sense in the case that

$$\left\|\mathbf{J}_k^{-1}F(\mathbf{x}^{(k+1)})\right\|_2 < \left\|\Delta\mathbf{x}^{(k)}\right\|_2$$

"simplified Quasi-Newton correction"

MATLAB-CODE: Broyden method (3.4.11)

```
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 (3.4.12))

solve two SLEs

Termination, see 3.4.2

construct $\mathbf{w} := \mathbf{J}_k^{-1}F(\mathbf{x}^{(k)})$ ($\rightarrow$ recursion (3.4.12))

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

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

Computational cost : N steps

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

Memory cost : N steps

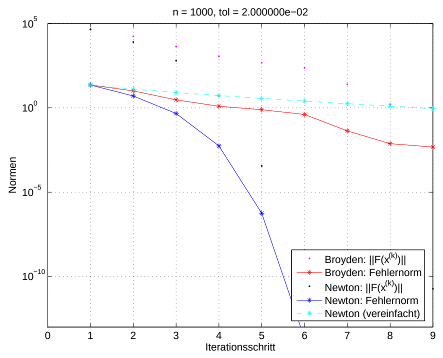
- LU-factors of J + auxiliary vectors $\in \mathbb{R}^n$
- N vectors $\mathbf{x}^{(k)} \in \mathbb{R}^n$

Example 3.4.20 (Broyden method for a large non-linear system).

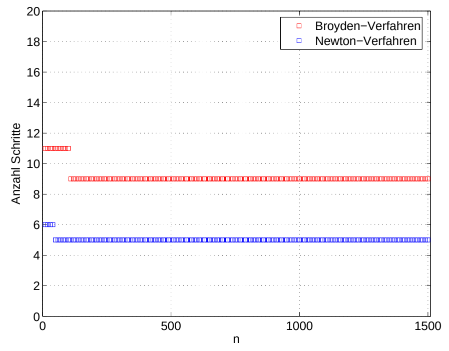
$$F(\mathbf{x}) = \begin{cases} \mathbb{R}^n & \mapsto \mathbb{R}^n \\ \mathbf{x} & \mapsto \operatorname{diag}(\mathbf{x})\mathbf{A}\mathbf{x} - \mathbf{b} \end{cases},$$
$$\mathbf{b} = (1,2,\dots,n) \in \mathbb{R}^n ,$$
$$\mathbf{A} = \mathbf{I} + \mathbf{a}\mathbf{a}^T \in \mathbb{R}^{n,n} ,$$
$$\mathbf{a} = \frac{1}{\sqrt{\mathbf{1}\cdot\mathbf{b} - 1}}(\mathbf{b} - \mathbf{1}) .$$

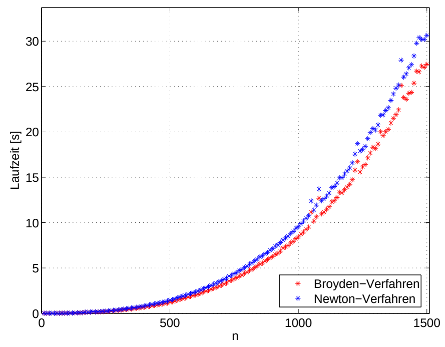
The interpretation of the results resemble the example 3.4.18 ▷

`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.

◇

4 Krylov Methods for Linear Systems of Equations

= A class of iterative methods ($\rightarrow$ section 3.1) for approximate solution of large linear systems of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A} \in \mathbb{K}^{n,n}$.

BUT, we have reliable *direct* methods (Gauss eliminination $\longrightarrow$ Sect. 2.1, LU-factorization $\longrightarrow$ Alg. 2.2.5, QR-factorization $\longrightarrow$ Alg. 2.8.11) 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), $\rightarrow$ (2.2.1),
- inevitable fill-in may exhaust main memory,
- the system matrix may be available only as procedure `y=evalA(x)` $\leftrightarrow$ $\mathbf{y} = \mathbf{A}\mathbf{x}$

4.1 Descent Methods

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

with symmetric positive definite (s.p.d., $\longrightarrow$ Def. 2.7.1) system matrix $\mathbf{A}$

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

Definition 4.1.1 (Energy norm)**.**
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 4.1.1 (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 $\mathbf{A}\mathbf{x}=\mathbf{b}$ 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 $\mathbb{K}=\mathbb{C}$.

△

4.1.1 Quadratic minimization context

Lemma 4.1.2 (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:*

$$\mathbf{A}\mathbf{x}=\mathbf{b}\quad\Leftrightarrow\quad \mathbf{x}=\arg\min_{\mathbf{y}\in\mathbb{R}^n} J(\mathbf{y})\,,\quad J(\mathbf{y}):=\tfrac{1}{2}\mathbf{y}^T\mathbf{A}\mathbf{y}-\mathbf{b}^T\mathbf{y}\,. \tag{4.1.1}$$

A quadratic functional

Proof. If $\mathbf{x}^*:=\mathbf{A}^{-1}\mathbf{b}$ a straightforward computation using $\mathbf{A}=\mathbf{A}^T$ shows

$$\begin{aligned} J(\mathbf{x})-J(\mathbf{x}^*) &= \tfrac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}-\mathbf{b}^T\mathbf{x}-\tfrac{1}{2}(\mathbf{x}^*)^T\mathbf{A}\mathbf{x}^*+\mathbf{b}^T\mathbf{x}^* \\ &\overset{\mathbf{b}=\mathbf{A}\mathbf{x}^*}{=} \tfrac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}-(\mathbf{x}^*)^T\mathbf{A}\mathbf{x}+\tfrac{1}{2}(\mathbf{x}^*)^T\mathbf{A}\mathbf{x}^* \\ &= \tfrac{1}{2}\|\mathbf{x}-\mathbf{x}^*\|_A^2\,. \end{aligned} \tag{4.1.2}$$

Then the assertion follows from the properties of the energy norm.

Example 4.1.2 (Quadratic functional in 2D).

Plot of J from (4.1.1) for $\mathbf{A}=\begin{pmatrix}2&1\\1&2\end{pmatrix}$, $\mathbf{b}=\begin{pmatrix}1\\1\end{pmatrix}$.

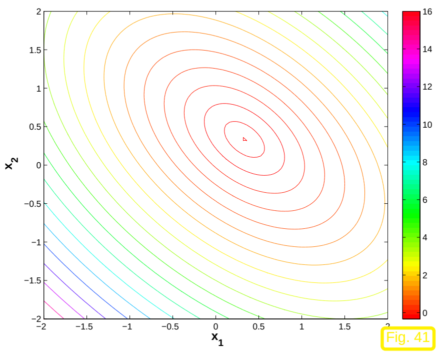


Fig. 41

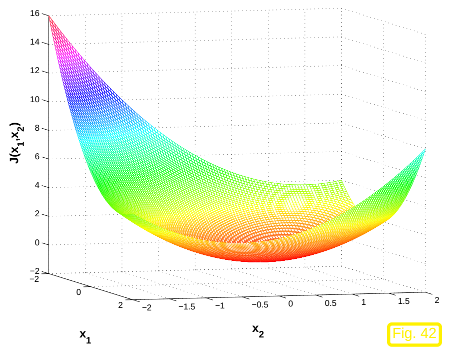


Fig. 42

Level lines of quadratic functionals are (hyper)ellipses

◇

Algorithmic idea: (Lemma 4.1.2 ➣) Solve $\mathbf{A}\mathbf{x}=\mathbf{b}$ iteratively by successive solution of *simpler* minimization problems

4.1.2 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 ($\longrightarrow$ Analysis).

The most natural iteration:

Algorithm 4.1.3 (Steepest descent). (*ger.:* steilster Abstieg)

```
Initial guess   $\mathbf{x}^{(0)} \in D$, $k = 0$
repeat
    $\mathbf{d}_k := -\mathbf{grad}\, F(\mathbf{x}^{(k)})$
    $t^* := \underset{t\in\mathbb{R}}{\operatorname{argmin}}\, F(\mathbf{x}^{(k)} + t\mathbf{d}_k)$   ( 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 $\hat{=}$ 1D minimization: use Newton's method ($\rightarrow$ Sect. 3.3.2.1) on derivative
- a posteriori termination criterion, see Sect. 3.1.2 for a discussion. ($\tau \mathrel{\hat{=}}$ prescribed tolerance)

The gradient ($\rightarrow$ [40, Kapitel 7])

$$\mathbf{grad}\, F(\mathbf{x}) = \begin{pmatrix} \frac{\partial F}{\partial x_i} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n \qquad (4.1.3)$$

provides the direction of local steepest ascent/descent of F

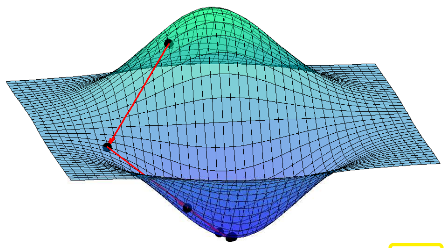


Fig. 43

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.

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

However, for the quadratic minimization problem (4.1.1) Alg. 4.1.3 will converge:

Adaptation: steepest descent algorithm Alg. 4.1.3 for quadratic minimization problem (4.1.1)

$$F(\mathbf{x}) := J(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} \quad \Rightarrow \quad \mathbf{grad}\, J(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}\,. \qquad (4.1.4)$$

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 (4.1.3) of the gradient.

➣ For the descent direction in Alg. 4.1.3 applied to the minimization of J from (4.1.1) holds

$$\mathbf{d}_k = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} =: \mathbf{r}_k \quad \text{the residual } (\rightarrow \text{Def. 2.5.8) for } \mathbf{x}^{(k-1)}.$$

Alg. 4.1.3 for $F = J$ from (4.1.1): function to be minimized in line search step:

$$\varphi(t) := J(\mathbf{x}^{(k)} + t\mathbf{d}_k) = J(\mathbf{x}^{(k)}) + t\mathbf{d}_k^T(\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}) + \tfrac{1}{2}t^2\mathbf{d}_k^T\mathbf{A}\mathbf{d}_k \quad \rightarrow \text{a parabola ! .}$$

$$\frac{d\varphi}{dt}(t^*) = 0 \quad \Leftrightarrow \quad \boxed{t^* = \frac{\mathbf{d}_k^T\mathbf{d}_k}{\mathbf{d}_k^T\mathbf{A}\mathbf{d}_k}} \quad \text{(unique minimizer) .}$$

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. 2.7.1) $\Rightarrow$ $\mathbf{d}_k^T\mathbf{A}\mathbf{d}_k > 0$, if $\mathbf{d}_k \neq 0$

► $\varphi(t)$ is a parabola that is bounded from below (upward opening)

Based on (4.1.4) and (4.1.3) we obtain the following steepest descent method for the minimization problem (4.1.1):

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 4.1.4 (Gradient method).

```
Initial guess $\mathbf{x}^{(0)} \in \mathbb{R}^n$, $k = 0$
$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
repeat
    $t^* := \dfrac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{r}_k^T\mathbf{A}\mathbf{r}_k}$
    $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + t^*\mathbf{r}_k$
    $\mathbf{r}_{k+1} := \mathbf{r}_k - t^*\mathbf{A}\mathbf{r}_k$
    $k := k+1$
until $\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\| \leq \tau \left\|\mathbf{x}^{(k)}\right\|$
```

Code 4.1.6: gradient method for $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A}$ s.p.d.

```
function x = gradit(A,b,x,tol,maxit)
r = b-A*x;
for k=1:maxit
  p = A*r;
  ts = (r'*r)/(r'*p);
  x = x + ts*r;
  if (abs(ts)*norm(r) < tol*norm(x))
    return; end
  r = r - ts*p;
end
```

Recursion for residuals:

$$\mathbf{r}_{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{A}(\mathbf{x}^{(k)} + t^*\mathbf{r}_k) = \mathbf{r}_k - t^*\mathbf{A}\mathbf{r}_k \,. \tag{4.1.5}$$

One step of gradient method involves

- $\boxed{\text{\textit{A single} matrix}\times\text{vector product with } \mathbf{A}}$,
- 2 AXPY-operations (→ 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 (→ 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.

4.1.4 Convergence of gradient method

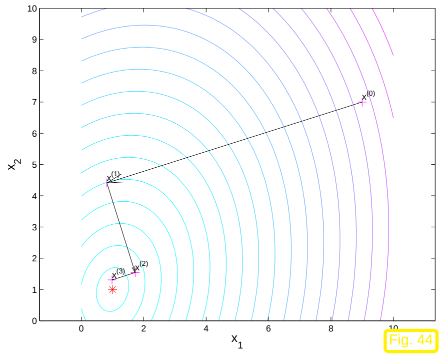
Example 4.1.7 (Gradient method in 2D).

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

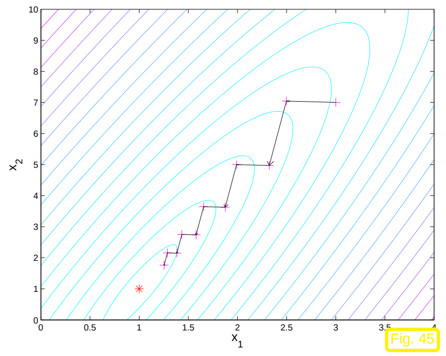
$$\mathbf{A}_1 = \begin{pmatrix} 1.9412 & -0.2353 \\ -0.2353 & 1.0588 \end{pmatrix} \quad , \quad \mathbf{A}_2 = \begin{pmatrix} 7.5353 & -1.8588 \\ -1.8588 & 0.5647 \end{pmatrix}$$

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

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



iterates of Alg. 4.1.4 for $\mathbf{A}_1$



iterates of Alg. 4.1.4 for $\mathbf{A}_2$

Recall (→ linear algebra) that every real symmetric matrix can be diagonalized by orthogonal similarity transformations, see Cor. 5.1.7: $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T$, $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n) \in \mathbb{R}^{n,n}$ diagonal,

$\mathbf{Q}^{-1} = \mathbf{Q}^T$.

$$J(\mathbf{Q}\widehat{\mathbf{y}}) = \tfrac{1}{2}\widehat{\mathbf{y}}^T\mathbf{D}\widehat{\mathbf{y}} - \underbrace{(\mathbf{Q}^T\mathbf{b})^T}_{=:\widehat{\mathbf{b}}^T}\widehat{\mathbf{y}} = \tfrac{1}{2}\sum_{i=1}^{n} d_i\widehat{y}_i^2 - \widehat{b}_i\widehat{y}_i \,.$$

Hence, a congruence transformation maps the level surfaces of J from (4.1.1) 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 slower convergence of gradient method
- Orthogonality of successive residuals $\mathbf{r}_k$, $\mathbf{r}_{k+1}$

Clear from definition of Alg. 4.1.4:

$$\mathbf{r}_k^T\mathbf{r}_{k+1} = \mathbf{r}_k^T\mathbf{r}_k - \mathbf{r}_k^T\frac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{r}_k^T\mathbf{A}\mathbf{r}_k}\mathbf{A}\mathbf{r}_k = 0 \,.$$

◇

Example 4.1.8 (Convergence of gradient method).

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

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

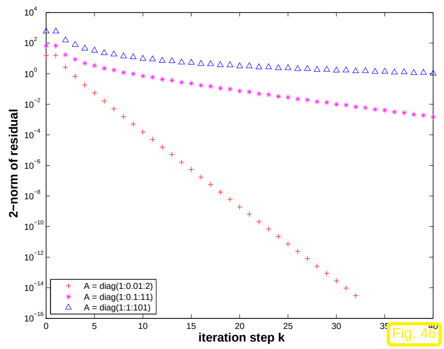


Fig. 46

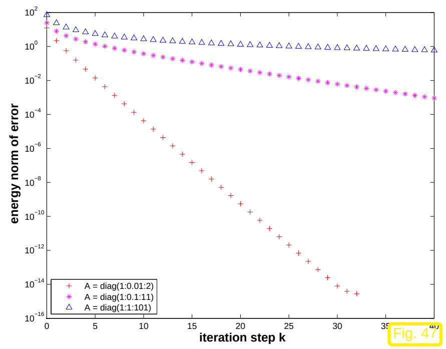


Fig. 47

Note: To study convergence it is *sufficient to consider diagonal matrices*, because

1. 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, $\longrightarrow$ linear algebra course & Chapter 5, Cor. 5.1.7),
2. when applying the gradient method Alg. 4.1.4 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 $\mathbf{Q}\mathbf{x}^{(k)} = \widetilde{\mathbf{x}}^{(k)}$.

Observation:
- linear convergence ($\longrightarrow$ Def. 3.1.4), see also Rem. 3.1.3
- 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

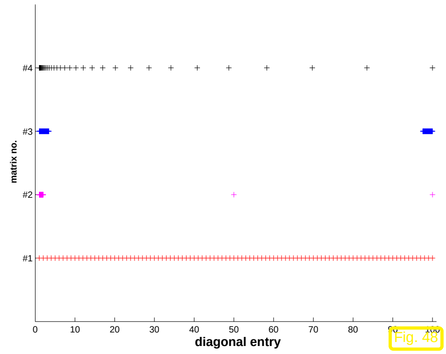
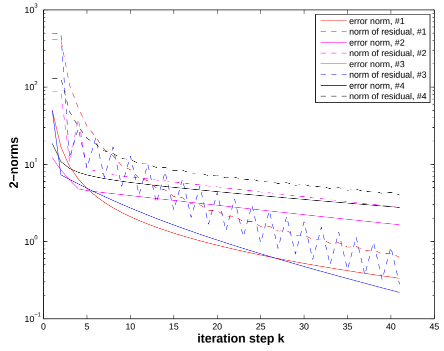


Fig. 48



Observation: Matrices #1, #2 & #4 ➣ little impact of distribution of eigenvalues on *asymptotic* convergence (exception: matrix #2)

◇

Theory [20, Sect. 9.2.2]:

Theorem 4.1.3 (Convergence of gradient method/steepest descent).
The iterates of the gradient method of Alg. 4.1.4 satisfy

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

that is, the iteration converges at least linearly (w.r.t. energy norm $\longrightarrow$ Def. 4.1.1).

✎ notation: $\operatorname{cond}_2(\mathbf{A}) \hat{=}$ condition number of $\mathbf{A}$ induced by 2-norm

Remark 4.1.9 (2-norm from eigenvalues).

$$\mathbf{A} = \mathbf{A}^T \quad \Rightarrow \quad \begin{aligned} \|\mathbf{A}\|_2 &= \max(|\sigma(\mathbf{A})|) \;, \\ \left\|\mathbf{A}^{-1}\right\|_2 &= \min(|\sigma(\mathbf{A})|)^{-1} \;, \text{ if } \mathbf{A} \text{ regular.} \end{aligned} \tag{4.1.6}$$

$$\boxed{\mathbf{A} = \mathbf{A}^T \quad \Rightarrow \quad \operatorname{cond}_2(\mathbf{A}) = \frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})} \;, \quad \text{where} \quad \begin{aligned} \lambda_{\max}(\mathbf{A}) &:= \max(|\sigma(\mathbf{A})|) \;, \\ \lambda_{\min}(\mathbf{A}) &:= \min(|\sigma(\mathbf{A})|) \;. \end{aligned}} \tag{4.1.7}$$

✎ other notation $\quad \kappa(\mathbf{A}) := \dfrac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})} \quad \hat{=}$ spectral condition number of $\mathbf{A}$

(for general $\mathbf{A}$: $\lambda_{\max}(\mathbf{A})/\lambda_{\min}(\mathbf{A})$ largest/smallest eigenvalue *in modulus*)

These results are an immediate consequence of the fact that

$$\forall \mathbf{A} \in \mathbb{R}^{n,n}, \quad \mathbf{A}^T = \mathbf{A} \quad \exists \mathbf{U} \in \mathbb{R}^{n,n}, \quad \mathbf{U}^{-1} = \mathbf{U}^T: \quad \mathbf{U}^T\mathbf{A}\mathbf{U} \quad \text{is diagonal.}$$

$\longrightarrow$ linear algebra course & Chapter 5, Cor. 5.1.7.

Please note that for general regular $\mathbf{M} \in \mathbb{R}^{n,n}$ we *cannot* expect $\mathrm{cond}_2(\mathbf{M}) = \kappa(\mathbf{M})$.

$\triangle$

4.2 Conjugate gradient method (CG)

Again we consider a linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ with s.p.d. ($\rightarrow$ Def. 2.7.1) system matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ and given $\mathbf{b} \in \mathbb{R}^n$.

Liability of gradient method of Sect. 4.1.3: NO MEMORY

1D line search in Alg. 4.1.4 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 $U_1 \subset U_2 \subset U_3 \subset \cdots \subset U_n = \mathbb{R}^n$, $\dim U_k = k$

$$\mathbf{x}^{(k)} := \operatorname*{argmin}_{\mathbf{x} \in U_k + \mathbf{x}^{(0)}} J(x) \,, \tag{4.2.1}$$

quadratic functional from (4.1.1)

Note: Once the subspaces U_k and $\mathbf{x}^{(0)}$ are fixed, the iteration (4.2.1) is well defined, because $J_{|U_k+\mathbf{x}^{(0)}}$ always possess a unique minimizer.

Obvious (from Lemma 4.1.2): $\mathbf{x}^{(n)} = \mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$

Thanks to (4.1.2), definition (4.2.1) 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. 2.5.8)

$$\boxed{U_{k+1} = \mathrm{Span}\,\{U_k, \mathbf{r}_k\}} \,, \quad \mathbf{x}^{(k)} \text{ from (4.2.1).} \tag{4.2.2}$$

Obvious: $\mathbf{r}_k = 0 \quad \Rightarrow \quad \mathbf{x}^{(k)} = \mathbf{x}^* := \mathbf{A}^{-1}\mathbf{b}$ done ✔

Lemma 4.2.1 ($\mathbf{r}_k \perp U_k$)**.**
With $\mathbf{x}^{(k)}$ *according to* (4.2.1), U_k *from* (4.2.2) *the residual* $\mathbf{r}_k := \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$ *satisfies*

$$\mathbf{r}_k^T\mathbf{u} = 0 \quad \forall \mathbf{u} \in U_k \quad (\text{"}\mathbf{r}_k \perp U_k\text{"}).$$

Proof. Consider

$$\psi(t) = J(\mathbf{x}^{(k)} + t\mathbf{u}) \,, \quad \mathbf{u} \in U_k, \quad t \in \mathbb{R} \,.$$

By (4.2.1), $t \mapsto \psi(t)$ 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. $\square$

Corollary 4.2.2. *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 4.2.1 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.

(4.2.1) and (4.2.2) define the conjugate gradient method (CG) for the iterative solution of
$\mathbf{A}\mathbf{x} = \mathbf{b}$

(hailed as a "top ten algorithm" of the 20th century, SIAM News, 33(4))

4.2.1 Krylov spaces

Definition 4.2.3 (Krylov space)**.**
For $\mathbf{A} \in \mathbb{R}^{n,n}$, $\mathbf{z} \in \mathbb{R}^n$, $\mathbf{z} \neq 0$, the l-th Krylov space is defined as

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

Equivalent definition: $\mathcal{K}_l(\mathbf{A},\mathbf{z}) = \{p(\mathbf{A})\mathbf{z}$: p polynomial of degree $\leq l\}$

Lemma 4.2.4. *The subspaces $U_k \subset \mathbb{R}^n$, $k \geq 1$, defined by* (4.2.1) *and* (4.2.2) *satisfy*

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

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 4.2.1 finish the proof. $\square$

4.2.2 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

(4.2.1) ➣ set $\quad \mathbf{x}^{(l)} = \mathbf{x}^{(0)} + \gamma_1\mathbf{p}_1 + \cdots + \gamma_l\mathbf{p}_k$.

For $\psi(\gamma_1, \ldots, \gamma_l) := J(\mathbf{x}^{(0)} + \gamma_1\mathbf{p}_1 + \cdots + \gamma_l\mathbf{p}_l)$ holds

$$\text{(4.2.1)} \quad \Leftrightarrow \quad \frac{\partial\psi}{\partial\gamma_j} = 0 , \quad j = 1, \ldots, l .$$

This leads to a linear system of equations by which the coefficients γ_j can be computed:

$$\begin{pmatrix} \mathbf{p}_1^T\mathbf{A}\mathbf{p}_1 & \cdots & \mathbf{p}_1^T\mathbf{A}\mathbf{p}_l \\ \vdots & & \vdots \\ \mathbf{p}_l^T\mathbf{A}\mathbf{p}_1 & \cdots & \mathbf{p}_l^T\mathbf{A}\mathbf{p}_l \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_l \end{pmatrix} = \begin{pmatrix} \mathbf{p}_1^T\mathbf{r} \\ \vdots \\ \mathbf{p}_l^T\mathbf{r} \end{pmatrix} . \tag{4.2.3}$$

Great simplification, if $\{\mathbf{p}_1, \ldots, \mathbf{p}_l\}$ $\mathbf{A}$-orthogonal basis of $\mathcal{K}_l(\mathbf{A},\mathbf{r})$: $\quad \mathbf{p}_j^T\mathbf{A}\mathbf{p}_i = 0$ for $i \neq j$.

Assume: $\mathbf{A}$-orthogonal basis $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ of $\mathbb{R}^n$ available, such that

$$\boxed{\operatorname{Span}\{\mathbf{p}_1, \ldots, \mathbf{p}_l\} = \mathcal{K}_l(\mathbf{A},\mathbf{r})} .$$

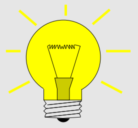
▶ (Efficient) successive computation of $\mathbf{x}^{(l)}$ becomes possible
(LSE (4.2.3) becomes diagonal !)

```
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 { $\mathbf{x}^{(j)} := \mathbf{x}^{(j-1)} + \frac{\mathbf{p}_j^T\mathbf{r}_0}{\mathbf{p}_j^T\mathbf{A}\mathbf{p}_j}\mathbf{p}_j$ }        (4.2.4)
```

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 iteration*.

Lemma 4.2.1 implies orthogonality $\mathbf{p}_j \perp \mathbf{r}_m := \mathbf{b} - \mathbf{A}\mathbf{x}^{(m)}$, $1 \leq j \leq m \leq l$

$$\mathbf{p}_j^T(\mathbf{b} - \mathbf{A}\mathbf{x}^{(m)}) = \mathbf{p}_j^T\left(\underbrace{\mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}}_{=\mathbf{r}_0} - \sum_{k=1}^{m} \frac{\mathbf{p}_k^T\mathbf{r}_0}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k}\mathbf{A}\mathbf{p}_k\right) = 0 . \tag{4.2.5}$$



(4.2.5) $\Rightarrow$ Idea: Gram-Schmidt orthogonalization, of residuals $\mathbf{r}_j := \mathbf{b} - \mathbf{A}\mathbf{x}^{(j)}$ w.r.t. $\mathbf{A}$-inner product:

$$\mathbf{p}_1 := \mathbf{r}_0 , \; \mathbf{p}_{j+1} := \underbrace{(\mathbf{b} - \mathbf{A}\mathbf{x}^{(j)})}_{=:\mathbf{r}_j} - \sum_{k=1}^{j} \frac{\mathbf{p}_k^T\mathbf{A}\mathbf{r}_j}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k}\mathbf{p}_k , \quad j = 1, \ldots, l-1 \quad . \tag{4.2.6}$$

Lemma 4.2.5 (Bases for Krylov spaces in CG)**.**
If they do not vanish, the vectors $\mathbf{p}_j$, $1 \leq j \leq l$, and $\mathbf{r}_j := \mathbf{b} - \mathbf{A}\mathbf{x}^{(j)}$, $0 \leq j \leq l$, from (4.2.4), (4.2.6) *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. 4.2.2

Proof. $\mathbf{A}$-orthogonality of $\mathbf{p}_j$ by construction, study (4.2.6).

$$\text{(4.2.4) \& (4.2.6)} \quad \Rightarrow \quad \mathbf{p}_{j+1} = \mathbf{r}_0 - \sum_{k=1}^{j} \frac{\mathbf{p}_k^T \mathbf{r}_0}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \mathbf{A}\mathbf{p}_k - \sum_{k=1}^{j} \frac{\mathbf{p}_k^T \mathbf{A}\mathbf{r}_j}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \mathbf{p}_k \,.$$

$$\Rightarrow \quad \mathbf{p}_{j+1} \in \operatorname{Span}\left\{\mathbf{r}_0, \mathbf{p}_1, \ldots, \mathbf{p}_j, \mathbf{A}\mathbf{p}_1, \ldots, \mathbf{A}\mathbf{p}_j\right\} \,.$$

A simple induction argument confirms (i)

$$\text{(4.2.6)} \quad \Rightarrow \quad \mathbf{r}_j \in \operatorname{Span}\left\{\mathbf{p}_1, \ldots, \mathbf{p}_{j+1}\right\} \quad \& \quad \mathbf{p}_j \in \operatorname{Span}\left\{\mathbf{r}_0, \ldots, \mathbf{r}_{j-1}\right\} \,. \tag{4.2.7}$$

$$\blacktriangleright \quad \boxed{\operatorname{Span}\left\{\mathbf{p}_1, \ldots, \mathbf{p}_j\right\} = \operatorname{Span}\left\{\mathbf{r}_0, \ldots, \mathbf{r}_{j-1}\right\} = \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)} \,. \tag{4.2.8}$$

$$\text{(4.2.5)} \quad \Rightarrow \quad \mathbf{r}_j \perp \operatorname{Span}\left\{\mathbf{p}_1, \ldots, \mathbf{p}_j\right\} = \operatorname{Span}\left\{\mathbf{r}_0, \ldots, \mathbf{r}_{j-1}\right\} \,. \tag{4.2.9}$$

□

Orthogonalities from Lemma 4.2.5 ➤ short recursions for $\mathbf{p}_k$, $\mathbf{r}_k$, $\mathbf{x}^{(k)}$!

$$\text{(4.2.5)} \quad \Rightarrow \quad \text{(4.2.6) collapses to} \quad \mathbf{p}_{j+1} := \mathbf{r}_j - \frac{\mathbf{p}_j^T \mathbf{A}\mathbf{r}_j}{\mathbf{p}_j^T \mathbf{A}\mathbf{p}_j} \mathbf{p}_j \,, \quad j = 1, \ldots, l \,.$$

recursion for residuals:

$$\text{(4.2.4)} \quad \blacktriangleright \quad \mathbf{r}_j = \mathbf{r}_{j-1} - \frac{\mathbf{p}_j^T \mathbf{r}_0}{\mathbf{p}_j^T \mathbf{A}\mathbf{p}_j} \mathbf{A}\mathbf{p}_j \,.$$

$$\text{Lemma 4.2.5, (i)} \quad \blacktriangleright \quad \mathbf{r}_{j-1}^H \mathbf{p}_j = \left(\mathbf{r}_0 + \sum_{k=1}^{m-1} \frac{\mathbf{r}_0^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \mathbf{A}\mathbf{p}_k \right)^T \mathbf{p}_j = \mathbf{r}_0^T \mathbf{p}_j \,. \tag{4.2.10}$$

The orthogonality (4.2.10) together with (4.2.9) permits us to replace $\mathbf{r}_0$ with Vr_{j-1} in the actual implementation.

Algorithm 4.2.1 (CG method for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A}$ s.p.d.).

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

▶ 1 matrix×vector product, 3 dot products, 3 AXPY-operations per step:
If $\mathbf{A}$ sparse, $\operatorname{nnz}(\mathbf{A}) \sim n$ ➤ computational effort $O(n)$ per step

Code 4.2.2: basic CG iteration

```
function x = cg(A,b,x,tol,maxit)
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;
   if (norm(b - A * x) <= tol*n0) return; end
   r = r - alpha * q;
end
```

MATLAB-function:

`x=pcg(A,b,tol,maxit,[],[],x0)`	: Solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ with at most `maxit` CG steps: stop, when $\|\mathbf{r}_l\| : \|\mathbf{r}_0\| < \mathrm{tol}$.
`x=pcg(Afun,b,tol,maxit,[],[],x0)`	: `Afun` = handle to function for computing $\mathbf{A}$×vector.
`[x,flag,relr,it,resv] = pcg(...)`	: diagnostic information about iteration

Remark 4.2.3 (A posteriori termination criterion for plain CG).

For any vector norm and associated matrix norm (→ Def. 2.5.2) hold (with residual $\mathbf{r}_l := \mathbf{b} - \mathbf{A}\mathbf{x}^{(l)}$)

$$\frac{1}{\operatorname{cond}(\mathbf{A})}\frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|} \le \underbrace{\frac{\|\mathbf{x}^{(l)}-\mathbf{x}^*\|}{\|\mathbf{x}^{(0)}-\mathbf{x}^*\|}}_{\text{relative decrease of iteration error}} \le \operatorname{cond}(\mathbf{A})\frac{\|\mathbf{r}_l\|}{\|\mathbf{r}_0\|}\,. \tag{4.2.11}$$

(4.2.11) can easily be deduced from the error equation $\mathbf{A}(\mathbf{x}^{(k)}-\mathbf{x}^*) = \mathbf{r}_k$, see Def. 2.5.8 and (2.5.6).

△

4.2.3 Convergence of CG

Note: CG is a *direct solver*, because (in exact arithmetic) $\mathbf{x}^{(k)} = \mathbf{x}^*$ for some $k \le n$

Example 4.2.4 (Impact of roundoff errors on CG).

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

Hilbert-Matrix: extremely ill-conditioned

residual norms during CG iteration $\mathbf{R} = \left[\mathbf{r}_0,\dots,\mathbf{r}^{(10)}\right]$

▷:

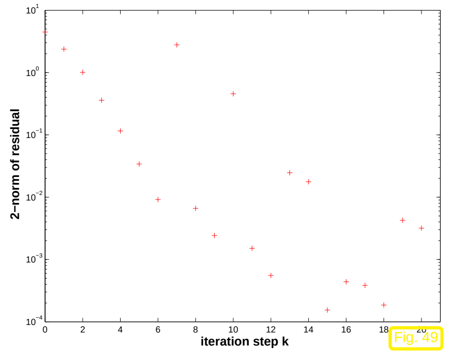


Fig. 49

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

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

➢ Roundoff
- destroys orthogonality of residuals
- prevents computation of exact solution after n steps.

▶ Numerical instability (→ Def. 2.5.5) ➢ 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 4.2.5 (Convergence of CG as iterative solver).

CG (Code 4.2.1)& gradient method (Code 4.1.4) for LSE with sparse s.p.d. "Poisson matrix"

```
A = gallery('poisson',m); x0 = (1:n)'; b = zeros(n,1);
```

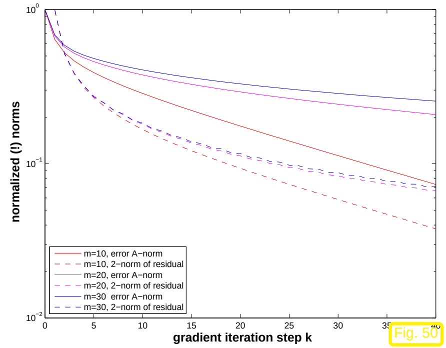


Fig. 50

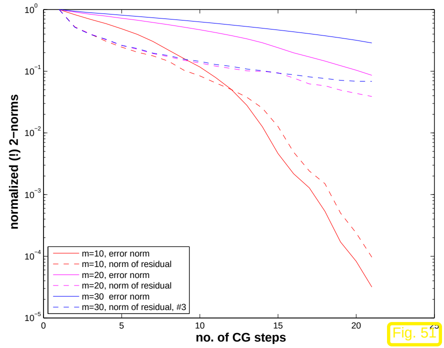


Fig. 51

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 (4.1.2) and (4.2.1):

Corollary 4.2.6 (“Optimality” of CG iterates).
Writing $\mathbf{x}^* \in \mathbb{R}^n$ for the exact solution of $\mathbf{Ax} = \mathbf{b}$ the CG iterates satisfy

$$\left\|\mathbf{x}^* - \mathbf{x}^{(l)}\right\|_A = \min\{\|\mathbf{y} - \mathbf{x}^*\|_A : \mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\} \quad , \quad \mathbf{r}_0 := \mathbf{b} - \mathbf{Ax}^{(0)} \; .$$

This paves the way for a quantitative convergence estimate:

$$\mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}) \quad \Leftrightarrow \quad \mathbf{y} = \mathbf{x}^{(0)} + \mathbf{A}\, p(\mathbf{A})(\mathbf{x} - \mathbf{x}^{(0)}) \;, \quad p \text{ = polynomial of degree } \leq l-1 \; .$$

$$\blacktriangleright \quad \mathbf{x} - \mathbf{y} = q(\mathbf{A})(\mathbf{x} - \mathbf{x}^{(0)}), \quad q \text{ = polynomial of degree } \leq l \;, \quad q(0) = 1 \; .$$

$$\blacktriangledown$$

$$\left\|\mathbf{x} - \mathbf{x}^{(l)}\right\|_A \leq \boxed{\min\{\max_{\lambda \in \sigma(\mathbf{A})} |q(\lambda)| : q \text{ polynomial of degree } \leq l \;, \quad q(0) = 1\}} \cdot \left\|\mathbf{x} - \mathbf{x}^{(0)}\right\|_A \; . \tag{4.2.12}$$

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 [20, Satz 9.4.2]

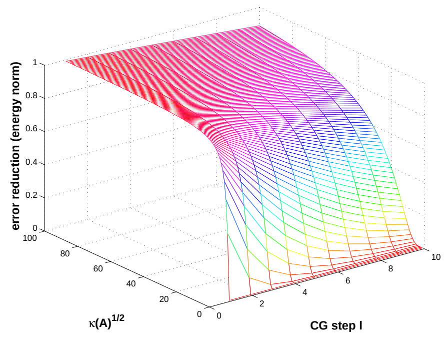
Theorem 4.2.7 (Convergence of CG method).
The iterates of the CG method for solving $\mathbf{Ax} = \mathbf{b}$ (see Code 4.2.1) with $\mathbf{A} = \mathbf{A}^T$ s.p.d. satisfy

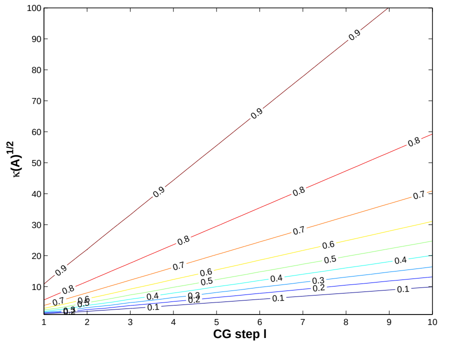
$$\begin{aligned}\left\|\mathbf{x} - \mathbf{x}^{(l)}\right\|_A &\leq \frac{2\left(1 - \frac{1}{\sqrt{\kappa(\mathbf{A})}}\right)^l}{\left(1 + \frac{1}{\sqrt{\kappa(\mathbf{A})}}\right)^{2l} + \left(1 - \frac{1}{\sqrt{\kappa(\mathbf{A})}}\right)^{2l}} \left\|\mathbf{x} - \mathbf{x}^{(0)}\right\|_A \\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

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

The estimate of this theorem confirms *asymptotic linear convergence* of the CG method ($\rightarrow$ Def. 3.1.4) 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. 4.2.7:





Code 4.2.6: plotting theoretical bounds for CG convergence rate

```
function plottheorate
[X,Y] = meshgrid(1:10,1:100); R = zeros(100,10);
for l=1:100
  t = 1/l;
  for j=1:10
    R(l,j) = 2*(1-t)^j/((1+t)^(2*j)+(1-t)^(2*j));
  end
end
```

```
figure; view([-45,28]); mesh(X,Y,R); colormap hsv;
xlabel('{\bf␣CG␣step␣l}','Fontsize',14);
ylabel('{\bf␣\kappa(A)^{1/2}}','Fontsize',14);
zlabel('{\bf␣error␣reduction␣(energy␣norm)}','Fontsize',14);

print -depsc2 '../PICTURES/theorate1.eps';

figure; [C,h] = contour(X,Y,R); clabel(C,h);
xlabel('{\bf␣CG␣step␣l}','Fontsize',14);
ylabel('{\bf␣\kappa(A)^{1/2}}','Fontsize',14);

print -depsc2 '../PICTURES/theorate2.eps';
```

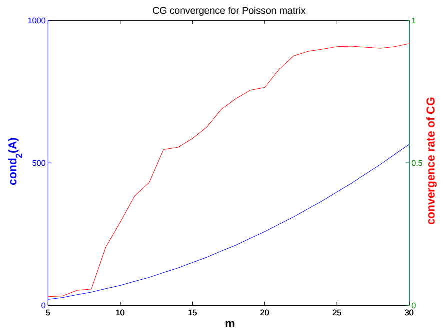
Example 4.2.7 (Convergence rates for CG method).

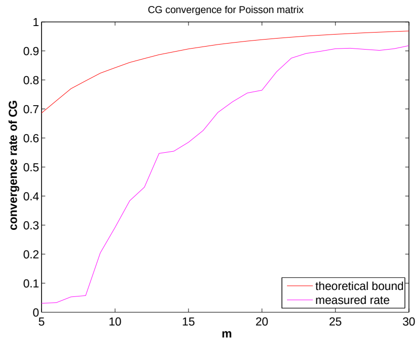
Code 4.2.8: CG for Poisson matrix

```
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:

$$\text{rate} \approx \sqrt[10]{\frac{\left\|\mathbf{r}^{(30)}\right\|_2}{\left\|\mathbf{r}^{(20)}\right\|_2}} .$$



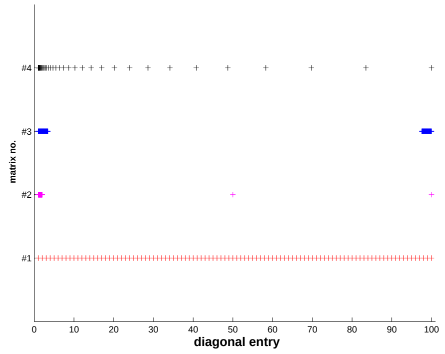


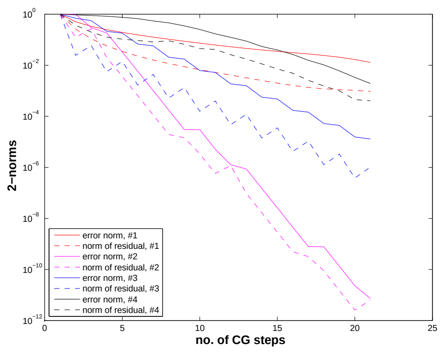
◇

Example 4.2.9 (CG convergence and spectrum). → Ex. 4.1.8

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 (4.2.12) 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. 4.1.8)

CG convergence boosted by clustering of eigenvalues

◇

4.3 Preconditioning

Thm. 4.2.7 ➣ (Potentially) slow convergence of CG in case $\kappa(\mathbf{A}) \gg 1$.

Idea: Preconditioning

Apply CG method to transformed linear system

$$\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{b}} \quad , \quad \widetilde{\mathbf{A}} := \mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2} \, , \; \widetilde{\mathbf{x}} := \mathbf{B}^{1/2}\mathbf{x} \, , \; \widetilde{\mathbf{b}} := \mathbf{B}^{-1/2}\mathbf{b} \, , \tag{4.3.1}$$

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

What is meant by the "square root" $\mathbf{B}^{1/2}$ of a s.p.d. matrix $\mathbf{B}$?

Recall: 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 $\mathbf{D}$ (→ linear algebra, Chapter 5, Cor. 5.1.7). If $\mathbf{B}$ is s.p.d. the (diagonal) entries of $\mathbf{D}$ are strictly positive and we can define

$$\mathbf{D} = \operatorname{diag}(\lambda_1,\dots,\lambda_n),\quad \lambda_i > 0 \quad\Rightarrow\quad \mathbf{D}^{1/2} := \operatorname{diag}(\sqrt{\lambda_1},\dots,\sqrt{\lambda_n}) \;.$$

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.

Notion 4.3.1 (Preconditioner)**.**
A s.p.d. matrix $\mathbf{B} \in \mathbb{R}^{n,n}$ is called a preconditioner (ger.: Vorkonditionierer) for the s.p.d. matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, if

1. *$\kappa(\mathbf{B}^{-1/2}\mathbf{A}\mathbf{B}^{-1/2})$ is “small” and*
2. *the evaluation of $\mathbf{B}^{-1}\mathbf{x}$ is about as expensive (in terms of elementary operations) as the matrix$\times$vector multiplication $\mathbf{A}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^n$.*

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 5.1.4)
- $\exists 0 < \gamma < \Gamma,\quad \Gamma/\gamma$ "‘small”: $\quad \gamma\,(\mathbf{x}^T\mathbf{B}\mathbf{x}) \le \mathbf{x}^T\mathbf{A}\mathbf{x} \le \Gamma\,(\mathbf{x}^T\mathbf{B}\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^n$,

  where equivalence is seen by transforming $\mathbf{y} := \mathbf{B}^{-1/2}\mathbf{x}$ and appealing to the min-max Theorem 5.3.5.

“Reader’s digest” version of notion 4.3.1:

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 (4.3.1) is usually not available with acceptable computational costs.

However, if one formally applies Algorithm 4.2.1 to the transformed system from (4.3.1), 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!

Algorithm 4.3.1 (Preconditioned CG method (PCG)).

Input: initial guess $\mathbf{x} \in \mathbb{R}^n \mathrel{\hat=} \mathbf{x}^{(0)} \in \mathbb{R}^n$, tolerance $\tau > 0$
Output: approximate solution $\mathbf{x} \mathrel{\hat=} \mathbf{x}^{(l)}$

```
$\mathbf{p} := \mathbf{r} := \mathbf{b} - \mathbf{A}\mathbf{x}$;   $\mathbf{p} := \mathbf{B}^{-1}\mathbf{r}$; $\mathbf{q} := \mathbf{p}$; $\tau_0 := \mathbf{p}^T\mathbf{r}$;
for $l = 1$ to $l_{\max}$ do {
    $\beta := \mathbf{r}^T\mathbf{q}$;   $\mathbf{h} := \mathbf{A}\mathbf{p}$;   $\alpha := \frac{\beta}{\mathbf{p}^T\mathbf{h}}$;
    $\mathbf{x} := \mathbf{x} + \alpha\mathbf{p}$;
    $\mathbf{r} := \mathbf{r} - \alpha\mathbf{h}$;                                        (4.3.2)
    $\mathbf{q} := \mathbf{B}^{-1}\mathbf{r}$;   $\beta := \frac{\mathbf{r}^T\mathbf{q}}{\beta}$;
    if $|\mathbf{q}^T\mathbf{r}| \le \tau \cdot \tau_0$ then stop;
    $\mathbf{p} := \mathbf{q} + \beta\mathbf{p}$;
}
```

▶ Computational effort per step: 1 evaluation $\mathbf{A}\times$vector, 1 evaluation $\mathbf{B}^{-1}\times$vector, 3 dot products, 3 AXPY-operations

Remark 4.3.2 (Convergence theory for PCG).

Assertions of Thm. 4.2.7 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 4.3.3 (Simple preconditioners).

$\mathbf{B}$ = easily invertible “part” of $\mathbf{A}$

- $\mathbf{B} = \mathtt{diag}(\mathbf{A})$: Jacobi preconditioner (diagonal scaling)
- $(\mathbf{B})_{ij} = \begin{cases} (\mathbf{A})_{ij} & \text{, if } \; |i-j| \le k \;, \\ 0 & \text{else,} \end{cases}$ for some $k \ll n$.
- Symmetric Gauss-Seidel preconditioner

Idea: Solve $\mathbf{A}\mathbf{x}=\mathbf{b}$ approximately in two stages:

① Approximation $\mathbf{A}^{-1}\approx \texttt{tril}(\mathbf{A})$ (lower triangular part): $\widetilde{\mathbf{x}}=\texttt{tril}(\mathbf{A})^{-1}\mathbf{b}$

② Approximation $\mathbf{A}^{-1}\approx \texttt{triu}(\mathbf{A})$ (upper triangular part) and use this to approximately "solve" the error equation $\mathbf{A}(\mathbf{x}-\widetilde{\mathbf{x}})=\mathbf{r}$, with residual $\mathbf{r}:=\mathbf{b}-\mathbf{A}\widetilde{\mathbf{x}}$:

$$\mathbf{x}=\widetilde{\mathbf{x}}+\texttt{triu}(\mathbf{A})^{-1}(\mathbf{b}-\mathbf{A}\widetilde{\mathbf{x}}) .$$

With $\mathbf{L}_A:=\texttt{tril}(\mathbf{A})$, $\mathbf{U}_A:=\texttt{triu}(\mathbf{A})$ one finds

$$\mathbf{x}=(\mathbf{L}_A^{-1}+\mathbf{U}_A^{-1}-\mathbf{U}_A^{-1}\mathbf{A}\mathbf{L}_A^{-1})\mathbf{b} \quad ➤ \quad \mathbf{B}^{-1}=\mathbf{L}_A^{-1}+\mathbf{U}_A^{-1}-\mathbf{U}_A^{-1}\mathbf{A}\mathbf{L}_A^{-1} .$$

◇

More complicated preconditioning strategies:

- Incomplete Cholesky factorization, MATLAB-`ichol`
- Sparse approximate inverse preconditioner (SPAI)

Example 4.3.4 (Tridiagonal preconditioning).

Efficacy of preconditioning of sparse LSE with tridiagonal part:

Code 4.3.5: LSE for Ex. 4.3.4
```
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
invB = @(x) x; [x,rn] = pcgbase(evalA,b,tol,maxit,invB,x0);

%tridiagonal preconditioning
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);
```

Code 4.3.6: simple PCG implementation
```
function [x,rn,xk] = pcgbase(evalA,b,tol,maxit,invB,x)
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
```

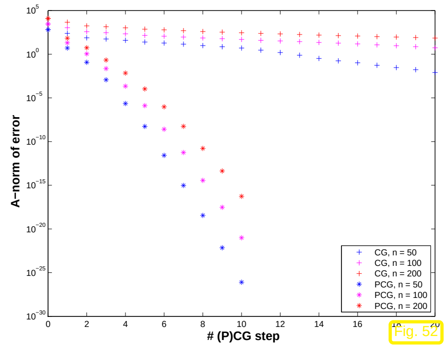


Fig. 52

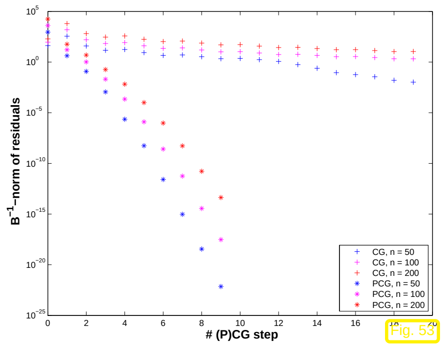


Fig. 53

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

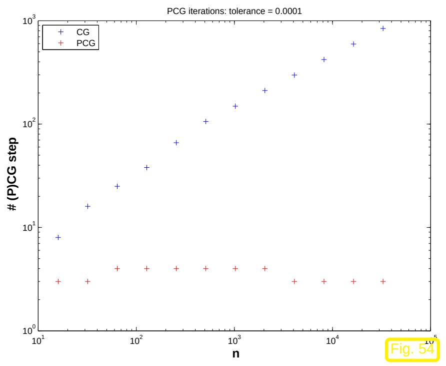


Fig. 54

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 4.3.7 (Termination of PCG).

Rem. 4.2.3, (4.2.11) ➤ Monitor transformed residual

$$\widetilde{\mathbf r} = \widetilde{\mathbf b} - \widetilde{\mathbf A}\widetilde{\mathbf x} = \mathbf B^{-1/2}\mathbf r \quad\Rightarrow\quad \|\widetilde{\mathbf r}\|_2^2 = \mathbf r^T\mathbf B^{-1}\mathbf r\ .$$

▶ Estimates for energy norm of error $\mathbf e^{(l)} := \mathbf x - \mathbf x^{(l)}$, $\mathbf x^* := \mathbf A^{-1}\mathbf b$

Use error equation $\mathbf A\mathbf e^{(l)} = \mathbf r_l$:

$$\mathbf r_l^T\mathbf B^{-1}\mathbf r_l = (\mathbf B^{-1}\mathbf A\mathbf e^{(l)})^T\mathbf A\mathbf e^{(l)} \le \lambda_{\max}(\mathbf B^{-1}\mathbf A)\left\|\mathbf e^{(l)}\right\|_A^2\ ,$$

$$\left\|\mathbf e^{(l)}\right\|_A^2 = (\mathbf A\mathbf e^{(l)})^T\mathbf e^{(l)} = \mathbf r_l^T\mathbf A^{-1}\mathbf r_l = \mathbf B^{-1}\mathbf r^T\mathbf B\mathbf A^{-1}\mathbf r_l \le \lambda_{\max}(\mathbf B\mathbf A^{-1})\,(\mathbf B^{-1}\mathbf r_l)^T\mathbf r_l\ .$$

available during PCG iteration (4.3.2)

$$\boxed{\frac{1}{\kappa(\mathbf B^{-1}\mathbf A)}\frac{\left\|\mathbf e^{(l)}\right\|_A^2}{\left\|\mathbf e^{(0)}\right\|_A^2} \le \frac{(\mathbf B^{-1}\mathbf r_l)^T\mathbf r_l}{(\mathbf B^{-1}\mathbf r_0)^T\mathbf r_0} \le \kappa(\mathbf B^{-1}\mathbf A)\frac{\left\|\mathbf e^{(l)}\right\|_A^2}{\left\|\mathbf e^{(0)}\right\|_A^2}} \qquad (4.3.3)$$

$\kappa(\mathbf B^{-1}\mathbf A)$ "'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 (4.3.2))

△

MATLAB-function: `[x,flag,relr,it,rv] = pcg(A,b,tol,maxit,B,[],x0);`
(A, B may be handles to functions providing $\mathbf A\mathbf x$ and $\mathbf B^{-1}\mathbf x$, resp.)

Remark 4.3.8 (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 !

△

4.4 Survey of Krylov Subspace Methods

4.4.1 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 [20, Sect. 9.5.2] (for *any* symmetric matrix **!**)

Theorem 4.4.1. *For $\mathbf A = \mathbf A^H \in \mathbb R^{n,n}$ the residuals $\mathbf r_l$ generated in the MINRES iteration satisfy*

$$\|\mathbf r_l\|_2 = \min\{\|\mathbf A\mathbf y - \mathbf b\|_2 : \mathbf y \in \mathbf x^{(0)} + \mathcal K_l(\mathbf A, \mathbf r_0)\}$$

$$\blacktriangleright \quad \|\mathbf r_l\|_2 \le \frac{2\left(1 - \frac{1}{\kappa(\mathbf A)}\right)^l}{\left(1 + \frac{1}{\kappa(\mathbf A)}\right)^{2l} + \left(1 - \frac{1}{\kappa(\mathbf A)}\right)^{2l}}\|\mathbf r_0\|_2\ .$$

Note: similar formula for (linear) rate of convergence as for CG, see Thm. 4.2.7, but with $\sqrt{\kappa(\mathbf A)}$

replaced with $\kappa(\mathbf{A})$!

Iterative solver for $\mathbf{A}\mathbf{x} = \mathbf{b}$ with *symmetric* system matrix $\mathbf{A}$:

MATLAB-functions:
- `[x,flg,res,it,resv] = minres(A,b,tol,maxit,B,[],x0);`
- `[...] = minres(Afun,b,tol,maxit,Binvfun,[],x0);`

Computational costs : 1 $\mathbf{A}\times$vector, 1 $\mathbf{B}^{-1}\times$vector per step, a few dot products & SAXPYs
Memory requirement: a few vectors $\in \mathbb{R}^n$

Extension to general regular $\mathbf{A} \in \mathbb{R}^{n,n}$:

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 6.

$$\mathbf{x}^{(l)} = \operatorname{argmin}\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbf{x}^{(0)} + \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\} .$$

➤ GMRES method for general matrices $\mathbf{A} \in \mathbb{R}^{n,n}$

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 4.4.1 (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). △

4.4.2 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 4.4.2 (Failure of Krylov iterative solvers).

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & \cdots & & \cdots & 0 \\ 0 & 0 & 1 & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \\ & & & & & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ 0 & & & & & 0 & 1 \\ 1 & 0 & \cdots & & & \cdots & 0 \end{pmatrix} , \quad \mathbf{b} = \begin{pmatrix} 0 \\ \vdots \\ \\ \vdots \\ 0 \\ 1 \end{pmatrix} \quad ► \quad \mathbf{x} = \mathbf{e}_1 .$$

$\mathbf{x}^{(0)} = 0 \;\succ\; \mathbf{r}_0 = \mathbf{e}_n \;\succ\; \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0) = \mathrm{Span}\,\{\mathbf{e}_n, \mathbf{e}_{n-1}, \dots, \mathbf{e}_{n-l+1}\}$

$$\blacktriangleright \quad \min\{\|\mathbf{y} - \mathbf{x}\|_2 : \mathbf{y} \in \mathcal{K}_l(\mathbf{A}, \mathbf{r}_0)\} = \begin{cases} 1 & \text{, if } l \le n \text{ ,} \\ 0 & \text{, for } l = n \text{ .} \end{cases}$$

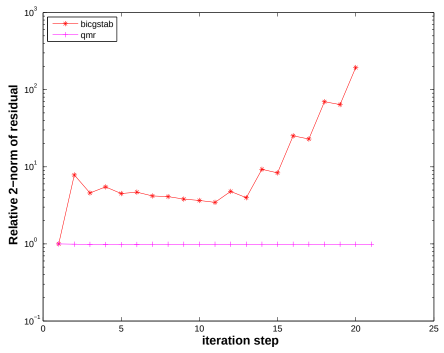
◇

TRY & PRAY

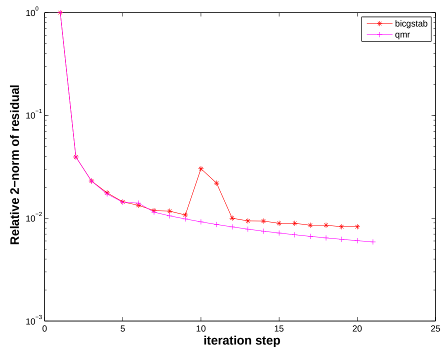
Example 4.4.3 (Convergence of Krylov subspace methods for non-symmetric system matrix).

```
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).

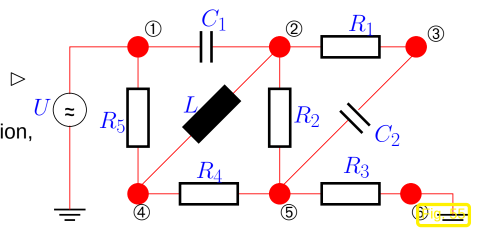
- They require system matrix $\mathbf{A}$ in procedural form `y=evalA(x)` $\leftrightarrow \mathbf{y} = \mathbf{A}\mathbf{x}$ only.
- They can perfectly exploit sparsity of system matrix.
- They can cash in on low accuracy requirements (, **IF** viable termination criterion available).
- They can benefit from a good initial guess.

5 Eigenvalues

Example 5.0.1 (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 (2.0.2))

$\succ$ linear system of equations for nodal potentials with complex system matrix $\mathbf{A}$

For circuit of Fig. 55 at angular frequency $\omega > 0$:

$$\mathbf{A} = \begin{pmatrix} i\omega C_1 + \frac{1}{R_1} - \frac{i}{\omega L} + \frac{1}{R_2} & -\frac{1}{R_1} & \frac{i}{\omega L} & -\frac{1}{R_2} \\ -\frac{1}{R_1} & \frac{1}{R_1} + i\omega C_2 & 0 & -i\omega C_2 \\ \frac{i}{\omega L} & 0 & \frac{1}{R_5} - \frac{i}{\omega L} + \frac{1}{R_4} & -\frac{1}{R_4} \\ -\frac{1}{R_2} & -i\omega C_2 & -\frac{1}{R_4} & \frac{1}{R_2} + i\omega C_2 + \frac{1}{R_4} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{R_1} + \frac{1}{R_2} & -\frac{1}{R_1} & 0 & -\frac{1}{R_2} \\ -\frac{1}{R_1} & \frac{1}{R_1} & 0 & 0 \\ 0 & 0 & \frac{1}{R_5} + \frac{1}{R_4} & -\frac{1}{R_4} \\ -\frac{1}{R_2} & 0 & -\frac{1}{R_4} & \frac{1}{R_2} + \frac{1}{R_4} \end{pmatrix} + i\omega \begin{pmatrix} C_1 & 0 & 0 & 0 \\ 0 & C_2 & 0 & -C_2 \\ 0 & 0 & 0 & 0 \\ 0 & -C_2 & 0 & C_2 \end{pmatrix} - i/\omega \begin{pmatrix} \frac{1}{L} & 0 & -\frac{1}{L} & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{L} & 0 & \frac{1}{L} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{A}(\omega) := \mathbf{W} + i\omega\mathbf{C} - i\omega^{-1}\mathbf{S} \quad , \quad \mathbf{W}, \mathbf{C}, \mathbf{S} \in \mathbb{R}^{n,n} \text{ symmetric} . \tag{5.0.1}$$

resonant frequencies $= \omega \in \{\omega \in \mathbb{R} \colon \mathbf{A}(\omega) \text{ singular}\}$

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:

$$\text{Find} \quad \omega \in \mathbb{C} \setminus \{0\} \colon \quad \mathbf{W} + i\omega\mathbf{C} - i\omega^{-1}\mathbf{S} \quad \text{singular} .$$

This is a quadratic eigenvalue problem: find $\mathbf{x} \neq 0$, $\omega \in \mathbb{C} \setminus \{0\}$,

$$\mathbf{A}(\omega)\mathbf{x} = (\mathbf{W} + i\omega\mathbf{C} - i\omega^{-1}\mathbf{S})\mathbf{x} = 0 . \tag{5.0.2}$$

Substitution: $\mathbf{y} = -i\omega^{-1}\mathbf{x}$ [41, Sect. 3.4]:

$$(5.0.2) \quad \Leftrightarrow \quad \underbrace{\begin{pmatrix} \mathbf{W} & \mathbf{S} \\ \mathbf{I} & 0 \end{pmatrix}}_{:=\mathbf{M}} \underbrace{\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}}_{:=\mathbf{z}} = \omega \underbrace{\begin{pmatrix} -i\mathbf{C} & 0 \\ 0 & -i\mathbf{I} \end{pmatrix}}_{:=\mathbf{B}} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$$

➣ generalized linear eigenvalue problem of the form: find $\omega \in \mathbb{C}$, $\mathbf{z} \in \mathbb{C}^{2n} \setminus \{0\}$ such that

$$\mathbf{M}\mathbf{z} = \omega\mathbf{B}\mathbf{z} . \tag{5.0.3}$$

In this example one is mainly interested in the eigenvalues ω, whereas the eigenvectors $\mathbf{z}$ usually need not be computed.

◇

Example 5.0.2 (Analytic solution of homogeneous linear ordinary differential equations). → [40, Remark 5.6.1]

Autonomous homogeneous linear ordinary differential equation (ODE):

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \quad , \quad \mathbf{A} \in \mathbb{C}^{n,n} . \tag{5.0.4}$$

$$\mathbf{A} = \mathbf{S} \underbrace{\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}}_{=:\mathbf{D}} \mathbf{S}^{-1} , \quad \mathbf{S} \in \mathbb{C}^{n,n} \text{ regular} \quad \Longrightarrow \quad \left(\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \quad \overset{\mathbf{z} = \mathbf{S}^{-1}\mathbf{y}}{\longleftrightarrow} \quad \dot{\mathbf{z}} = \mathbf{D}\mathbf{z} \right) .$$

➣ 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

$$\mathbf{z}_i(t) = \exp(\lambda_i t)(\mathbf{z}_0)_i = \exp(\lambda_i t) \left((\mathbf{S}^{-1})^T_{i,:} \mathbf{y}_0 \right) .$$

In light of Rem. 1.2.1:

$$\mathbf{A} = \mathbf{S} \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \mathbf{S}^{-1} \quad \Leftrightarrow \quad \mathbf{A} \left((\mathbf{S})_{:,i} \right) = \lambda_i \left((\mathbf{S})_{:,i} \right) \quad i = 1, \ldots, n . \tag{5.0.5}$$

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.

◇

5.1 Theory of eigenvalue problems

Definition 5.1.1 (Eigenvalues and eigenvectors)**.**

- $\lambda \in \mathbb{C}$ *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* $\mathbf{A} \in \mathbb{K}^{n,n}$: $\sigma(\mathbf{A}) := \{\lambda \in \mathbb{C}\colon \lambda \text{ eigenvalue of } \mathbf{A}\}$
- *eigenspace (ger.: Eigenraum) associated with eigenvalue* $\lambda \in \sigma(\mathbf{A})$:
  $$\operatorname{Eig}_{\mathbf{A}}(\lambda) := \operatorname{Ker}(\lambda\mathbf{I} - \mathbf{A})$$
- $\mathbf{x} \in \operatorname{Eig}_{\mathbf{A}}(\lambda) \setminus \{0\} \quad \Rightarrow \quad \mathbf{x}$ *is eigenvector*
- *Geometric multiplicity (ger.: Vielfachheit) of an eigenvalue* $\lambda \in \sigma(\mathbf{A})$:
  $$m(\lambda) := \dim \operatorname{Eig}_{\mathbf{A}}(\lambda)$$

Two simple facts:

$$\lambda \in \sigma(\mathbf{A}) \quad \Rightarrow \quad \dim \operatorname{Eig}_{\mathbf{A}}(\lambda) > 0\,, \tag{5.1.1}$$

$$\det(\mathbf{A}) = \det(\mathbf{A}^T) \quad \forall \mathbf{A} \in \mathbb{K}^{n,n} \quad \Rightarrow \quad \sigma(\mathbf{A}) = \sigma(\mathbf{A}^T)\,. \tag{5.1.2}$$

✎ notation: $\rho(\mathbf{A}) := \max\{|\lambda|\colon \lambda \in \sigma(\mathbf{A})\} \mathrel{\hat=}$ spectral radius of $\mathbf{A} \in \mathbb{K}^{n,n}$

Theorem 5.1.2 (Bound for spectral radius)**.**
For any matrix norm $\|\cdot\|$ *induced by a vector norm* (→ *Def. 2.5.2*)

$$\rho(\mathbf{A}) \le \|\mathbf{A}\|\ .$$

Lemma 5.1.3 (Gershgorin circle theorem)**.** *For any* $\mathbf{A} \in \mathbb{K}^{n,n}$ *holds true*

$$\sigma(\mathbf{A}) \subset \bigcup_{j=1}^{n} \{z \in \mathbb{C}\colon |z - a_{jj}| \le \textstyle\sum_{i \ne j} |a_{ji}|\}\ .$$

Lemma 5.1.4 (Similarity and spectrum)**.**
The spectrum of a matrix is invariant with respect to similarity transformations:

$$\forall \mathbf{A} \in \mathbb{K}^{n,n}\colon \quad \sigma(\mathbf{S}^{-1}\mathbf{A}\mathbf{S}) = \sigma(\mathbf{A}) \quad \forall \text{ regular } \mathbf{S} \in \mathbb{K}^{n,n}\ .$$

Lemma 5.1.5. *Existence of a one-dimensional invariant subspace*

$$\forall \mathbf{C} \in \mathbb{C}^{n,n}\colon \quad \exists \mathbf{u} \in \mathbb{C}^{n}\colon \quad \mathbf{C}(\operatorname{Span}\{\mathbf{u}\}) \subset \operatorname{Span}\{\mathbf{u}\}\ .$$

Theorem 5.1.6 (Schur normal form)**.**

$$\forall \mathbf{A} \in \mathbb{K}^{n,n}\colon \quad \exists \mathbf{U} \in \mathbb{C}^{n,n} \text{ unitary}\colon \quad \mathbf{U}^H\mathbf{A}\mathbf{U} = \mathbf{T} \quad \text{with } \mathbf{T} \in \mathbb{C}^{n,n} \text{ upper triangular}\ .$$

Corollary 5.1.7 (Principal axis transformation)**.**

$$\mathbf{A} \in \mathbb{K}^{n,n},\ \mathbf{A}\mathbf{A}^H = \mathbf{A}^H\mathbf{A}\colon \quad \exists \mathbf{U} \in \mathbb{C}^{n,n} \text{ unitary}\colon \quad \mathbf{U}^H\mathbf{A}\mathbf{U} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)\,, \quad \lambda_i \in \mathbb{C}\,.$$

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: $\mathbf{A} = -\mathbf{A}^H$ ➤ $\sigma(\mathbf{A}) \subset i\mathbb{R}$

➤

Normal matrices can be diagonalized by *unitary* similarity transformations

Symmetric real matrices can be diagonalized by *orthogonal* similarity transformations

In Thm. 5.1.7:
- $\lambda_1, \dots, \lambda_n$ = eigenvalues of $\mathbf{A}$
- Columns of $\mathbf{U}$ = orthonormal basis of eigenvectors of $\mathbf{A}$

Eigenvalue problems: (EVPs)

❶ Given $\mathbf{A} \in \mathbb{K}^{n,n}$ find **all** eigenvalues (= spectrum of $\mathbf{A}$).
❷ Given $\mathbf{A} \in \mathbb{K}^{n,n}$ find $\sigma(\mathbf{A})$ plus **all** eigenvectors.
❸ Given $\mathbf{A} \in \mathbb{K}^{n,n}$ find **a few** eigenvalues and associated eigenvectors

(Linear) generalized eigenvalue problem:

Given $\mathbf{A} \in \mathbb{C}^{n,n}$, regular $\mathbf{B} \in \mathbb{C}^{n,n}$, seek $\mathbf{x} \neq 0$, $\lambda \in \mathbb{C}$

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x} \quad \Leftrightarrow \quad \mathbf{B}^{-1}\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \,. \tag{5.1.3}$$

$\mathbf{x} \mathrel{\hat{=}}$ generalized eigenvector, $\lambda \mathrel{\hat{=}}$ generalized eigenvalue

Obviously every generalized eigenvalue problem is equivalent to a standard eigenvalue problem

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x} \quad \Leftrightarrow \quad \mathbf{B}^{-1}\mathbf{A} = \lambda\mathbf{x} \,.$$

However, usually it is not advisable to use this equivalence for numerical purposes!

Remark 5.1.1 (Generalized eigenvalue problems and Cholesky factorization).

If $\mathbf{B} = \mathbf{B}^H$ s.p.d. (→ Def. 2.7.1) with Cholesky factorization $\mathbf{B} = \mathbf{R}^H\mathbf{R}$

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x} \quad \Leftrightarrow \quad \widetilde{\mathbf{A}}\mathbf{y} = \lambda\mathbf{y} \quad \text{where } \widetilde{\mathbf{A}} := \mathbf{R}^{-H}\mathbf{A}\mathbf{R}^{-1},\ \mathbf{y} := \mathbf{R}\mathbf{x} \,.$$

➞ This transformation can be used for efficient computations.

△

5.2 “Direct” Eigensolvers

Purpose: solution of eigenvalue problems ❶, ❷ for dense matrices “up to machine precision”

MATLAB-function: `eig`

`d = eig(A)` : computes spectrum $\sigma(\mathbf{A}) = \{d_1, \dots, d_n\}$ of $\mathbf{A} \in \mathbb{C}^{n,n}$
`[V,D] = eig(A)` : computes $\mathbf{V} \in \mathbb{C}^{n,n}$, *diagonal* $\mathbf{D} \in \mathbb{C}^{n,n}$ such that $\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{D}$

Remark 5.2.1 (QR-Algorithm). → [18, Sect. 7.5]

Note: All “direct” eigensolvers are iterative methods

Idea: Iteration based on successive **unitary** similarity transformations

$$\mathbf{A} = \mathbf{A}^{(0)} \longrightarrow \mathbf{A}^{(1)} \longrightarrow \dots \longrightarrow \begin{cases} \text{diagonal matrix} & \text{, if } \mathbf{A} = \mathbf{A}^H \,, \\ \text{upper triangular matrix} & \text{, else.} \\ (\rightarrow \text{Thm. 5.1.6}) & \end{cases}$$

(superior stability of unitary transformations, see Rem. 2.8.1)

▶ QR-algorithm (with shift)

- in general: quadratic convergence
- cubic convergence for normal matrices

(→ [18, Sect. 7.5,8.2])

Code 5.2.2: QR-algorithm with shift

```
function d = eigqr(A,tol)
n = size(A,1);
while (norm(tril(A,-1)) > tol*norm(A))
  shift = A(n,n);
  [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

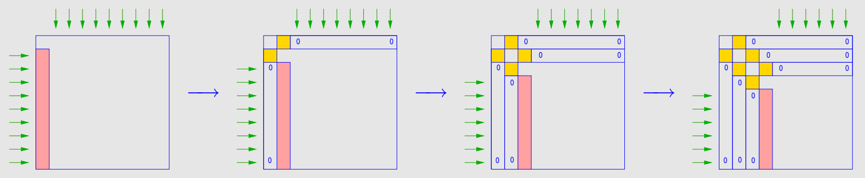
Library implementations of the QR-algorithm provide *numerically stable* eigensolvers (→ Def.2.5.5)

△

Remark 5.2.3 (Unitary similarity transformation to tridiagonal form).

Successive Householder similarity transformations of $\mathbf{A} = \mathbf{A}^H$:

(→ $\mathrel{\hat{=}}$ affected rows/columns, 🟥 $\mathrel{\hat{=}}$ 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 $\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}$, $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{n,n}$

`d = eig(A,B)` : computes all generalized eigenvalues
`[V,D] = eig(A,B)` : computes $\mathbf{V} \in \mathbb{C}^{n,n}$, *diagonal* $\mathbf{D} \in \mathbb{C}^{n,n}$ such that $\mathbf{A}\mathbf{V} = \mathbf{B}\mathbf{V}\mathbf{D}$

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 $\mathbf{D} = \operatorname{diag}(d_1, \dots, d_n)$.

Remark 5.2.4 (Computational effort for eigenvalue computations).

Computational effort (#elementary operations) for `eig()`:

$$\left.\begin{array}{ll} \text{eigenvalues \& eigenvectors of } \mathbf{A} \in \mathbb{K}^{n,n} & \sim 25n^3 + O(n^2) \\ \text{only eigenvalues of } \mathbf{A} \in \mathbb{K}^{n,n} & \sim 10n^3 + O(n^2) \\ \text{eigenvalues and eigenvectors } \mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n} & \sim 9n^3 + O(n^2) \\ \text{only eigenvalues of } \mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n} & \sim \tfrac{4}{3}n^3 + O(n^2) \end{array}\right\} \quad O(n^3)!$$

$$\text{only eigenvalues of tridiagonal } \mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n} \quad \sim 30n^2 + O(n)$$

Note: `eig` not available for sparse matrix arguments

Exception: `d=eig(A)` for sparse *Hermitian* matrices

△

Example 5.2.5 (Runtimes of `eig`).

Code 5.2.6: 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

figure;
loglog(times(:,1),times(:,2),'r+', times(:,1),times(:,3),'m*',...
       times(:,1),times(:,4),'cp', times(:,1),times(:,5),'b^',...
       times(:,1),times(:,6),'k.');
xlabel('{\bf␣matrix␣size␣n}','fontsize',14);
ylabel('{\bf␣time␣[s]}','fontsize',14);
title('eig␣runtimes');
```

```
legend('d␣=␣eig(A)','[V,D]␣=␣eig(A)','d␣=␣eig(B)','[V,D]␣=␣eig(B)','d␣=␣
  eig(C)',...
       'location','northwest');

print -depsc2 '../PICTURES/eigtimingall.eps'

figure;
loglog(times(:,1),times(:,2),'r+', times(:,1),times(:,3),'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␣matrix');
legend('d␣=␣eig(A)','[V,D]␣=␣eig(A)','O(n^3)','location','northwest');

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'
```

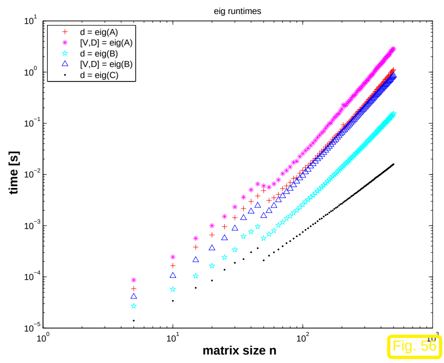


Fig. 56

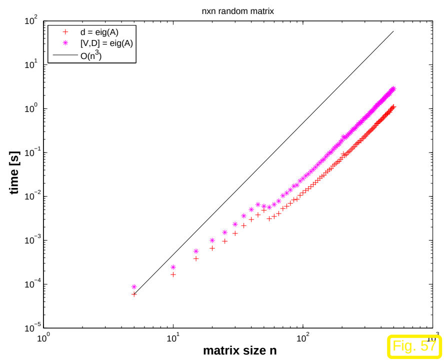


Fig. 57

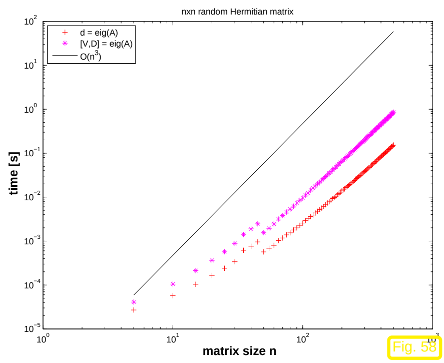


Fig. 58

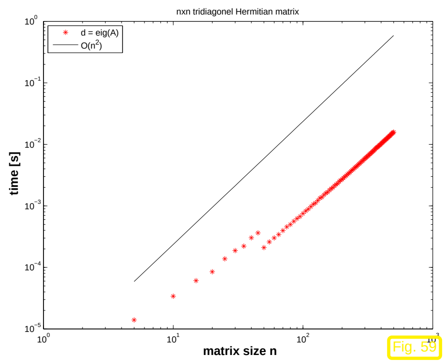


Fig. 59

☛ 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. 5.3, 5.4.

◇

5.3 Power Methods

5.3.1 Direct power method

Example 5.3.1 (Page rank algorithm).

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 $x_i \in [0,1]$)

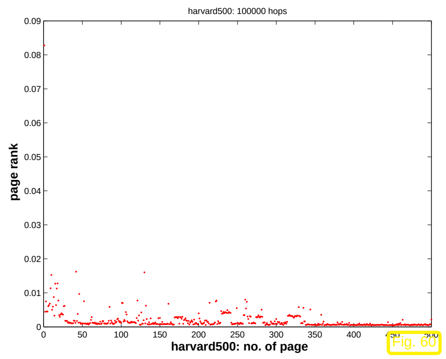
Stochastic simulation ▷

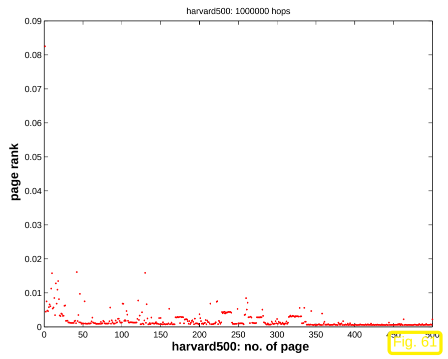
`rand` generates uniformly distributed pseudo-random numbers $\in [0,1[$

Web graph encoded in $\mathbf{G} \in \{0,1\}^{N,N}$

$(\mathbf{G})_{ij} = 1 \quad \Rightarrow \quad$ link $j \rightarrow i$,

Code 5.3.2: stochastic page rank simulation
```
function prstochsim(Nhops)
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
  idx = find(G(:,cp)); l = size(idx,1);
  rn = rand();
  if (isempty(idx)), cp = floor(rn*N)+1;
  elseif (rn < d), cp = floor(rn/d*N)+1;
  else cp = idx(floor((rn-d)/(1-d)*l)+1,1);
  end
  count(cp) = count(cp) + 1;
  plot(1:N,count/n,'r.'); axis([0 N+1 0 0.1]);
  xlabel('{\bf␣harvard500:␣no.␣of␣
    page}','fontsize',14);
  ylabel('{\bf␣page␣rank}','fontsize',14);
  title(sprintf('{\\bf␣page␣rank,␣harvard500:␣
    %d␣hops}',n),'fontsize',14);
  drawnow;
end
```





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 $1, \dots, N$, $\ell_i \hat{=}$ number of links from page i
- $N \times N$-matrix of transition probabilities page $i \longrightarrow$ page j: $\mathbf{A} = (a_{ij})_{i,j=1}^{N} \in \mathbb{R}^{N,N}$

$a_{ij} \in [0,1]$ $\hat{=}$ probability to jump from page j to page i.

$$\Rightarrow \quad \sum_{i=1}^{N} a_{ij} = 1 \,. \tag{5.3.1}$$

A matrix $\mathbf{A} \in [0,1]^{N,N}$ with the property (5.3.1) is called a (column) stochastic matrix.

“Meaning” of $\mathbf{A}$: given $\mathbf{x} \in [0,1]^N$, $\|\mathbf{x}\|_1 = 1$, where x_i is the probability of the surfer to visit page i, $i = 1, \dots, N$, at an instance t in time, $\mathbf{y} = \mathbf{A}\mathbf{x}$ satisfies

$$y_j \geq 0 \quad , \quad \sum_{j=1}^{N} y_j = \sum_{j=1}^{N}\sum_{i=1}^{N} a_{ji} x_i = \sum_{i=1}^{N} x_i \underbrace{\sum_{j=1}^{N} a_{ij}}_{=1} = \sum_{i=1}^{N} x_i = 1 \,.$$

► $y_j \hat{=}$ probability for visiting page j at time $t + \Delta t$.

Transition probability matrix for page rank computation

$$(\mathbf{A})_{ij} = \begin{cases} \frac{1}{N} & \text{, if } (\mathbf{G})_{ij} = 0 \;\forall i = 1,\dots,N \,, \\ d/N + (1-d)(\mathbf{G})_{ij}/\ell_j & \text{else.} \end{cases} \tag{5.3.2}$$

Code 5.3.3: transition probability matrix for page rank
```
function A = prbuildA(G,d)
N = size(G,1);
l = full(sum(G)); idx = find(l>0);
s = zeros(N,1); s(idx) = 1./l(idx);
ds = ones(N,1)/N; ds(idx) = d/N;
A = ones(N,1)*ones(1,N)*diag(ds) +
  (1-d)*G*diag(s);
```

Note: special treatment of zero columns!

Thought experiment: Instead of a single random surfer we may consider $m \in \mathbb{N}$ 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.

Now let $m \to \infty$ and instead of counting the surfers we weigh them and renormalize their total mass to 1. Let $\mathbf{x}^{(0)} \in [0,1]^N$, $\left\|\mathbf{x}^{(0)}\right\| = 1$ be the initial mass distribution of the surfers on the net. Then

$$\mathbf{x}^{(k)} = \mathbf{A}^k \mathbf{x}^{(0)}$$

will be their mass distribution after k hops. If the limit exists, the i-th component of $\mathbf{x}^* := \lim_{k\to\infty} \mathbf{x}^{(k)}$ 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 5.3.4: tracking fractions of many surfers
```
function prpowitsim(d,Nsteps)
if (nargin < 2), Nsteps = 5; end
if (nargin < 1), d = 0.15; end
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]);
for l=1:Nsteps
  pause; x = A*x; plot(1:N,x,'r+'); axis([0 N+1 0 0.1]);
  title(sprintf('{\\bf step %d}',l),'fontsize',14);
  xlabel('{\bf harvard500: no. of page}','fontsize',14);
  ylabel('{\bf page rank}','fontsize',14); drawnow;
end
```

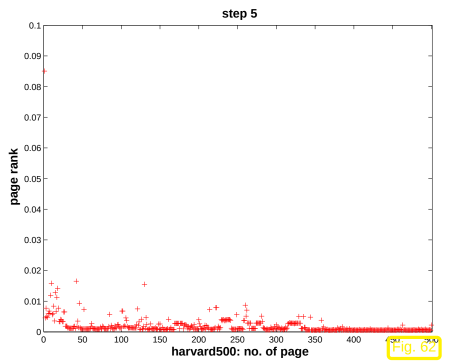


Fig. 62

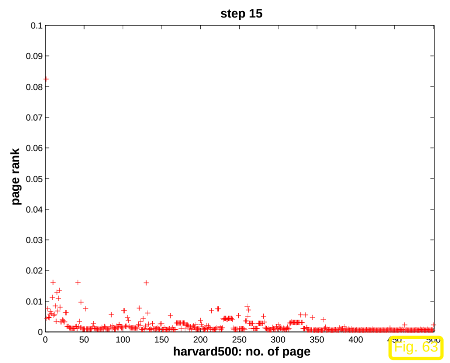


Fig. 63

Observation: Convergence of the $\mathbf{x}^{(k)} \to \mathbf{x}^*$, and the limit must be a fixed point of the iteration function:

$$\succ \quad \mathbf{A}\mathbf{x}^* = \mathbf{x}^* \quad \Rightarrow \quad \mathbf{x}^* \in \mathrm{Eig}_{\mathbf{A}}(1) \; .$$

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 $\mathbf{A}$:

For every stochastic matrix $\mathbf{A}$, by definition,

$$\mathbf{A}^T\mathbf{1} = \mathbf{1} \overset{(5.1.2)}{\Rightarrow} 1 \in \sigma(\mathbf{A}) \; ,$$

$$(2.5.5) \quad \Rightarrow \quad \|\mathbf{A}\|_1 = 1 \overset{\text{Thm 5.1.2}}{\Rightarrow} \rho(\mathbf{A}) = 1 \; .$$

For $\mathbf{r} \in \mathrm{Eig}_{\mathbf{A}}(1)$, that is, $\mathbf{A}\mathbf{r} = \mathbf{r}$, denote by $|\mathbf{r}|$ the vector $(|r_i|)_{i=1}^N$. Since all entries of $\mathbf{A}$ are non-negative, we conclude by the triangle inequality that $\|\mathbf{A}\mathbf{r}\|_1 \leq \|\mathbf{A}|\mathbf{r}|\|_1$

$$\Rightarrow \quad 1 = \|\mathbf{A}\|_1 = \sup_{\mathbf{x}\in\mathbb{R}^N} \frac{\|\mathbf{A}\mathbf{x}\|_1}{\|\mathbf{x}\|_1} \geq \frac{\|\mathbf{A}|\mathbf{r}|\|_1}{\||\mathbf{r}|\|_1} \geq \frac{\|\mathbf{A}\mathbf{r}\|_1}{\|\mathbf{r}\|_1} = 1 \; .$$

$$\Rightarrow \quad \|\mathbf{A}|\mathbf{r}|\|_1 = \|\mathbf{A}\mathbf{r}\|_1 \overset{\text{if } a_{ij}>0}{\Rightarrow} |\mathbf{r}| = \pm\mathbf{r} \; ,$$

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 (5.3.2). After normalization $\|\mathbf{r}\|_1 = 1$ the eigenvector can be regarded as a probability distribution on $\{1,\ldots,N\}$.

If $\mathbf{A}\mathbf{r} = \mathbf{r}$ and $\mathbf{A}\mathbf{s} = \mathbf{s}$ with $(\mathbf{r})_i \geq 0$, $(\mathbf{s})_i \geq 0$, $\|\mathbf{r}\|_1 = \|\mathbf{s}\|_1 = 1$, then $\mathbf{A}(\mathbf{r}-\mathbf{s}) = \mathbf{r}-\mathbf{s}$. Hence, by the above considerations, also all the entries of $\mathbf{r}-\mathbf{s}$ are either non-negative or non-positive. By the assumptions on $\mathbf{r}$ and $\mathbf{s}$ this is only possible, if $\mathbf{r}-\mathbf{s} = 0$. We conclude that

$$\mathbf{A} \in]0,1]^{N,N} \text{ stochastic} \quad \Rightarrow \quad \dim \mathrm{Eig}_{\mathbf{A}}(1) = 1 \; . \qquad (5.3.3)$$

Sorting the pages according to the size of the corresponding entries in $\mathbf{r}$ yields the famous "page rank".

Code 5.3.5: computing $\mathbf{r}$

```
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!

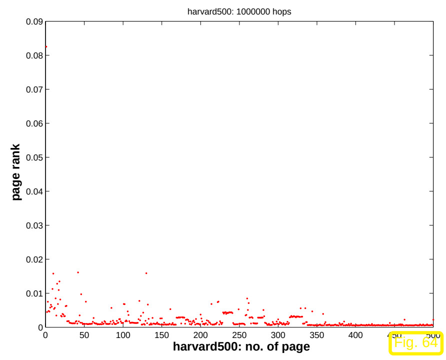


Fig. 64

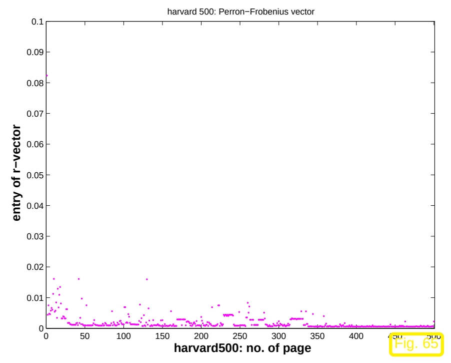


Fig. 65

stochastic simulation

eigenvector computation

The possibility to compute the stationary probability distribution of a Markov chain through an eigenvector of the transition probability matrix is due to a property of stationary Markov chains called ergodicity.

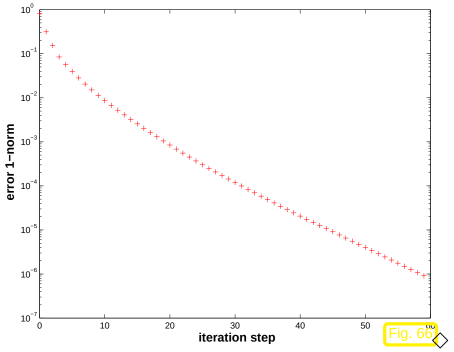
$\mathbf{A} \hat{=}$ page rank transition probability matrix, see Code 5.3.2, $d = 0.15$, harvard500 example.

Errors: ▷

$$\left\| \mathbf{A}^k \mathbf{x}_0 - \mathbf{r} \right\|_1 ,$$

with $\mathbf{x}_0 = \mathbf{1}/N$, $N = 500$.

linear convergence! (→ Def. 3.1.4)



Task:	given $\mathbf{A} \in \mathbb{K}^{n,n}$, find largest (in modulus) eigenvalue of $\mathbf{A}$ and (an) associated eigenvector.

Idea for $\mathbf{A} \in \mathbb{K}^{n,n}$ diagonalizable: $\mathbf{S}^{-1}\mathbf{A}\mathbf{S} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$

$$\mathbf{z} = \sum_{j=1}^{n} \zeta_j (\mathbf{S})_{:,j} \quad \Rightarrow \quad \mathbf{A}^k \mathbf{z} = \sum_{j=1}^{n} \zeta_j \lambda_j^k (\mathbf{S})_{:,j} .$$

If $|\lambda_1| \leq |\lambda_2| \leq \cdots \leq |\lambda_{n-1}| < |\lambda_n|$, $\left\|(\mathbf{S})_{:,j}\right\|_2 = 1$, $j = 1, \ldots, n$, $\zeta_n \neq 0$

$$\frac{\mathbf{A}^k \mathbf{z}}{\|\mathbf{A}^k \mathbf{z}\|} \rightarrow \pm(\mathbf{S})_{:,n} \quad \text{= eigenvector for } \lambda_n \quad \text{for } k \rightarrow \infty . \tag{5.3.4}$$

Suggests direct power method (*ger.:* Potenzmethode): iterative method (→ Sect. 3.1)

$$\text{initial guess:} \quad \mathbf{z}^{(0)} \text{ "arbitrary" ,}$$
$$\text{next iterate:} \quad \mathbf{w} := \mathbf{A}\mathbf{z}^{(k-1)} , \quad \mathbf{z}^{(k)} := \frac{\mathbf{w}}{\|\mathbf{w}\|_2} , \quad k = 1, 2, \ldots . \tag{5.3.5}$$

Computational effort: $1\times$ matrix×vector per step ➢ inexpensive for sparse matrices

$\mathbf{z}^{(k)} \rightarrow$ eigenvector, but how do we get the associated eigenvalue λ_n **?**

❶ upon convergence from (5.3.4) ➢ $\mathbf{A}\mathbf{z}^{(k)} \approx \lambda_n \mathbf{z}^{(k)}$ ➢ $|\lambda_n| \approx \dfrac{\left\|\mathbf{A}\mathbf{z}^{(k)}\right\|}{\left\|\mathbf{z}^{(k)}\right\|}$

❷ for $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$: $\lambda_n \approx \operatorname{argmin}_{\theta \in \mathbb{R}} \left\|\mathbf{A}\mathbf{z}^{(k)} - \theta \mathbf{z}^{(k)}\right\|_2^2$ ➤ $\lambda_n \approx \dfrac{(\mathbf{z}^{(k)})^T \mathbf{A}\mathbf{z}^{(k)}}{\left\|\mathbf{z}^{(k)}\right\|^2}$.

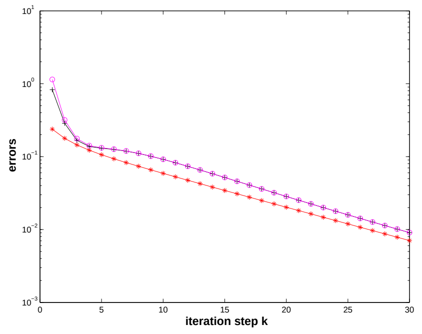
Definition 5.3.1. *For $\mathbf{A} \in \mathbb{K}^{n,n}$, $\mathbf{u} \in \mathbb{K}^n$ the Rayleigh quotient is defined by*

$$\rho_{\mathbf{A}}(\mathbf{u}) := \frac{\mathbf{u}^H \mathbf{A} \mathbf{u}}{\mathbf{u}^H \mathbf{u}} .$$

An immediate consequence of the definitions:

$$\lambda \in \sigma(\mathbf{A}) \quad , \quad \mathbf{z} \in \operatorname{Eig}_\lambda(\mathbf{A}) \quad \Rightarrow \quad \rho_{\mathbf{A}}(\mathbf{z}) = \lambda . \tag{5.3.6}$$

Example 5.3.6 (Direct power method). → Ex. 5.3.1



```
n = length(d);
S = triu(diag(n:-1:1,0)+...
    ones(n,n));
A = S*diag(d,0)*inv(S);
```

◁

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

Test matrices:

① `d=(1:10)';` ➢ $|\lambda_{n-1}| : |\lambda_n| = 0.9$

② `d = [ones(9,1); 2];` ➢ $|\lambda_{n-1}| : |\lambda_n| = 0.5$

③ `d = 1-2.^(-(1:0.5:5)');` ➢ $|\lambda_{n-1}| : |\lambda_n| = 0.9866$

$$\rho_{\mathrm{EV}}^{(k)} := \frac{\left\|\mathbf{z}^{(k)} - \mathbf{s}_{\cdot,n}\right\|}{\left\|\mathbf{z}^{(k-1)} - \mathbf{s}_{\cdot,n}\right\|} ,$$

$$\rho_{\mathrm{EW}}^{(k)} := \frac{|\rho_{\mathbf{A}}(\mathbf{z}^{(k)}) - \lambda_n|}{|\rho_{\mathbf{A}}(\mathbf{z}^{(k-1)}) - \lambda_n|} .$$

	①		②		③	
k	$\rho_{\mathrm{EV}}^{(k)}$	$\rho_{\mathrm{EW}}^{(k)}$	$\rho_{\mathrm{EV}}^{(k)}$	$\rho_{\mathrm{EW}}^{(k)}$	$\rho_{\mathrm{EV}}^{(k)}$	$\rho_{\mathrm{EW}}^{(k)}$
22	0.9102	0.9007	0.5000	0.5000	0.9900	0.9781
23	0.9092	0.9004	0.5000	0.5000	0.9900	0.9791
24	0.9083	0.9001	0.5000	0.5000	0.9901	0.9800
25	0.9075	0.9000	0.5000	0.5000	0.9901	0.9809
26	0.9068	0.8998	0.5000	0.5000	0.9901	0.9817
27	0.9061	0.8997	0.5000	0.5000	0.9901	0.9825
28	0.9055	0.8997	0.5000	0.5000	0.9901	0.9832
29	0.9049	0.8996	0.5000	0.5000	0.9901	0.9839
30	0.9045	0.8996	0.5000	0.5000	0.9901	0.9844

Observation: linear convergence (⟶ Def. 3.1.4)

◇

Theorem 5.3.2 (Convergence of direct power method).
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

$$\left\|\mathbf{A}\mathbf{z}^{(k)}\right\|_2 \to \lambda_n \quad , \quad \mathbf{z}^{(k)} \to \pm\mathbf{v} \quad \textit{linearly with rate } \frac{|\lambda_{n-1}|}{|\lambda_n|} ,$$

where $\mathbf{z}^{(k)}$ are the iterates of the direct power iteration and $\mathbf{y}^H\mathbf{z}^{(0)} \neq 0$ is assumed.

Remark 5.3.7 (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 5.3.8 (Termination criterion for direct power iteration). (⟶ Sect. 3.1.2)

Adaptation of a posteriori termination criterion (3.2.7)

"relative change" $\leq$ `tol`:
$$\begin{cases} \left\|\mathbf{z}^{(k)} - \mathbf{z}^{(k-1)}\right\| \leq (1/L - 1)\texttt{tol} , \\ \left\| \frac{\left\|\mathbf{A}\mathbf{z}^{(k)}\right\|}{\left\|\mathbf{z}^{(k)}\right\|} - \frac{\left\|\mathbf{A}\mathbf{z}^{(k-1)}\right\|}{\left\|\mathbf{z}^{(k-1)}\right\|} \right\| \leq (1/L - 1)\texttt{tol} \quad \text{see (3.1.7)} . \end{cases}$$

Estimated rate of convergence

△

5.3.2 Inverse Iteration

Example 5.3.9 (Image segmentation).

Given: grey-scale image: intensity matrix $\mathbf{P} \in \{0, \dots, 255\}^{m,n}$, $m, n \in \mathbb{N}$
$((\mathbf{P}))_{ij} \leftrightarrow$ pixel, $0 \hat{=}$ black, $255 \hat{=}$ white)

Loading and displaying images in MATLAB ▷

Code 5.3.10: loading and displaying an image
```
M = imread('eth.pbm');
[m,n] = size(M);
fprintf('%dx%d␣grey␣scale␣pixel␣image\n',m,n);
figure; image(M); title('ETH␣view');
col = [0:1/215:1]'*[1,1,1]; colormap(col);
```

(Fuzzy) task: Local segmentation

Find connected patches of image of the same shade/color

More general segmentation problem (non-local): identify parts of the image, not necessarily connected, with the same texture.

Next: Statement of (rigorously defined) problem, *cf.* Sect. 2.5.2:

Preparation: Numbering of pixels $1 \dots, mn$, e.g, lexicographic numbering:
- pixel set $\mathcal{V} := \{1 \dots, nm\}$
- indexing: index$(\text{pixel}_{i,j}) = (i-1)n + j$

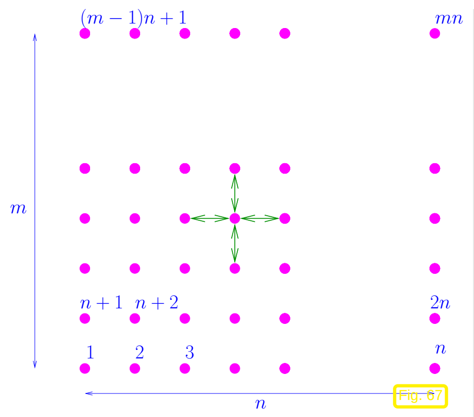
✎ notation: $p_k := (\mathbf{P})_{ij}$, if $k =$ index$(\text{pixel}_{i,j}) = (i-1)n + j$, $k = 1, \dots, N := mn$

Local similarity matrix:

$$\mathbf{W} \in \mathbb{R}^{N,N} \;,\quad N := mn \;, \tag{5.3.7}$$

$$(\mathbf{W})_{ij} = \begin{cases} 0 & \text{, if pixels } i, j \text{ not adjacent,} \\ 0 & \text{, if } i = j \;, \\ \sigma(p_i, p_j) & \text{, if pixels } i, j \text{ adjacent.} \end{cases}$$

$\leftrightarrow \mathrel{\hat=}$ adjacent pixels ▷



Similarity function, e.g., with $\alpha > 0$

$$\sigma(x,y) := \exp(-\alpha(x-y)^2) \;,\quad x,y \in \mathbb{R} \;.$$

Lexikographic numbering ▷

The entries of the matrix $\mathbf{W}$ measure the "similarity" of neighboring pixels: if $(\mathbf{W})_{ij}$ is large, they encode (almost) the same intensity, if $(\mathbf{W})_{ij}$ is close to zero, then they belong to parts of the picture with very different brightness. In the latter case, the boundary of the segment may separate the two pixels.

Definition 5.3.3 (Normalized cut). *(→ [36, Sect. 2])*
For $\mathcal{X} \subset \mathcal{V}$ define the normalized cut as

$$\mathrm{Ncut}(\mathcal{X}) := \frac{\mathrm{cut}(\mathcal{X})}{\mathrm{weight}(\mathcal{X})} + \frac{\mathrm{cut}(\mathcal{X})}{\mathrm{weight}(\mathcal{V} \setminus \mathcal{X})} \;,$$

with $\quad \mathrm{cut}(\mathcal{X}) := \sum_{i \in \mathcal{X}, j \notin \mathcal{X}} w_{ij} \;,\quad \mathrm{weight}(\mathcal{X}) := \sum_{i \in \mathcal{X}, j \in \mathcal{V}} w_{ij} \;.$

Segmentation problem (rigorous statement):

$$\text{find} \quad \mathcal{X}^* \subset \mathcal{V}: \quad \mathcal{X}^* = \operatorname*{argmin}_{\mathcal{X} \subset \mathcal{V}} \mathrm{Ncut}(\mathcal{X}) \;. \tag{5.3.8}$$

NP-hard combinatorial optimization problem !

Equivalent reformulation:

indicator function: $\quad z : \{1, \dots, N\} \mapsto \{-1, 1\} \;,\quad z_i := z(i) = \begin{cases} 1 & \text{, if } i \in \mathcal{X} \;, \\ -1 & \text{, if } i \notin \mathcal{X} \;. \end{cases}$ (5.3.9)

$$\mathrm{Ncut}(\mathcal{X}) = \frac{\sum_{z_i>0, z_j<0} -w_{ij} z_i z_j}{\sum_{z_i>0} d_i} + \frac{\sum_{z_i>0, z_j<0} -w_{ij} z_i z_j}{\sum_{z_i<0} d_i} \;, \tag{5.3.10}$$

$$d_i = \sum_{j \in \mathcal{V} \setminus \{i\}} w_{ij} = \mathrm{weight}(\{i\}) \;. \tag{5.3.11}$$

Sparse matrices:

$$\mathbf{D} := \mathrm{diag}(d_1, \dots, d_N) \in \mathbb{R}^{N,N} \;,\quad \mathbf{A} := \mathbf{D} - \mathbf{W} = \mathbf{A}^T \;. \quad \Rightarrow \quad \mathbf{1}^T \mathbf{A} = \mathbf{A}\mathbf{1} = 0 \;. \tag{5.3.12}$$

Code 5.3.11: assembly of $\mathbf{A}$, $\mathbf{D}$

```
function [A,D] = imgsegmat(P)
P = double(P); [n,m] = size(P);
spdata = zeros(4*n*m,1); spidx  = zeros(4*n*m,2);
k = 1;
for ni=1:n
  for mi=1:m
    mni = (mi-1)*n+ni;
    if (ni-1>0), spidx(k,:) = [mni,mni-1];
      spdata(k) = Sfun(P(ni,mi),P(ni-1,mi));
      k = k + 1;
    end
    if (ni+1<=n), spidx(k,:) = [mni,mni+1];
      spdata(k) = Sfun(P(ni,mi),P(ni+1,mi));
      k = k + 1;
    end
    if (mi-1>0), spidx(k,:) = [mni,mni-n];
      spdata(k) = Sfun(P(ni,mi),P(ni,mi-1));
      k = k + 1;
    end
    if (mi+1<=m), spidx(k,:) = [mni,mni+n];
      spdata(k) = Sfun(P(ni,mi),P(ni,mi+1));
      k = k + 1;
    end
  end
end

W = sparse(spidx(1:k-1,1),spidx(1:k-1,2),spdata(1:k-1),n*m,n*m);
D = spdiags(full(sum(A')'),[0],n*m,n*m);
A = W - D;
```

Lemma 5.3.4 (Ncut and Rayleigh quotient). *(→ [36, Sect. 2])*
With $\mathbf{z} \in \{-1,1\}^N$ *according to* (5.3.9) *there holds*

$$\mathrm{Ncut}(\mathcal{X}) = \underbrace{\frac{\mathbf{y}^T\mathbf{A}\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}}_{\text{generalized Rayleigh quotient } \rho_{\mathbf{A},\mathbf{D}}(\mathbf{y})} \;,\quad \mathbf{y} := (\mathbf{1}+\mathbf{z}) - \beta(\mathbf{1}-\mathbf{z}) \;,\quad \beta := \frac{\sum_{z_i>0} d_i}{\sum_{z_i<0} d_i} \;.$$

Proof. Note that by (5.3.9) $(\mathbf{1}-\mathbf{z})_i = 0 \;\Leftrightarrow\; i \in \mathcal{X}$, $(\mathbf{1}+\mathbf{z})_i = 0 \;\Leftrightarrow\; i \notin \mathcal{X}$. 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 $\kappa := \sum_{z_i>0} d_i / \sum_i d_i = \frac{\beta}{1+\beta}$. Also observe

$$\mathbf{y}^T\mathbf{D}\mathbf{y} = (\mathbf{1}+\mathbf{z})^T\mathbf{D}(\mathbf{1}+\mathbf{z}) + \beta^2(\mathbf{1}-\mathbf{z})^T\mathbf{D}(\mathbf{1}-\mathbf{z}) =$$
$$4\Big(\sum_{z_i>0} d_i + \beta^2\sum_{z_i<0} d_i\Big) = 4\beta\sum_i d_i = 4\beta\mathbf{1}^T\mathbf{D}\mathbf{1} \;.$$

This finishes the proof. □

► segmentation problem (5.3.8) $\Leftrightarrow$ $$\underset{\mathbf{y}\in\{2,-2\beta\}^N,\,\mathbf{1}^T\mathbf{D}\mathbf{y}=0}{\mathrm{argmin}} \rho_{\mathbf{A},\mathbf{D}}(\mathbf{y}) \;. \tag{5.3.13}$$
still NP-hard

➢ Minimizing $\mathrm{Ncut}(\mathcal{X})$ amounts to minimizing a (generalized) Rayleigh quotient (→ Def. 5.3.1) over a *discrete* set of vectors, which is still an NP-hard problem.

Idea: Relaxation

Discrete optimization problem → continuous optimization problem

(5.3.13) → $$\underset{\mathbf{y}\in\mathbb{R}^N,\,\|\mathbf{y}\|_{\mathbf{D}}=1,\,\mathbf{1}^T\mathbf{D}\mathbf{y}=0}{\mathrm{argmin}} \rho_{\mathbf{A},\mathbf{D}}(\mathbf{y}) \;. \tag{5.3.14}$$

The constraints $\|\mathbf{y}\|_{\mathbf{D}} = 1$, $\mathbf{1}^T\mathbf{D}\mathbf{y} = 0$ compound the difficulties of solving (5.3.14). However, the next theorem establishes a link to a generalized eigenvalue problem.

Theorem 5.3.5 (Courant-Fischer min-max theorem). *→ [18, 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*

$$U_0 = \{0\} \;,\quad U_\ell := \sum_{j=1}^{\ell} \mathrm{Eig}_{\mathbf{A}}(\lambda_j) \;,\; \ell = 1,\dots,m \quad \textit{and} \quad U_\ell^\perp := \{\mathbf{x}\in\mathbb{C}^n\colon \mathbf{u}^T\mathbf{x} = 0 \;\forall \mathbf{u}\in U_\ell\} \;.$$

Then

$$\min_{\mathbf{y}\in U_{\ell-1}^\perp\setminus\{0\}} \rho_{\mathbf{A}}(\mathbf{y}) = \lambda_\ell \;,\quad 1\leq \ell \leq m \;,\qquad \underset{\mathbf{y}\in U_{\ell-1}^\perp\setminus\{0\}}{\mathrm{argmin}} \rho_{\mathbf{A}}(\mathbf{y}) \subset \mathrm{Eig}_{\mathbf{A}}(\lambda_\ell) \;.$$

Proof. For diagonal $\mathbf{A} \in \mathbb{R}^{n,n}$ the assertion of the theorem is obvious. Thus, Cor. 5.1.7 settles everything. □

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

$$\lambda_1 = \min_{\mathbf{z}\in\mathbb{R}^n} \rho_{\mathbf{A}}(\mathbf{z}) \quad,\quad \lambda_2 = \min_{\mathbf{z}\in\mathbb{R}^n,\,\mathbf{z}\perp\mathbf{v}_1} \rho_{\mathbf{A}}(\mathbf{z}) \;,$$

where $\mathbf{v}_1 \in \mathrm{Eig}_{\mathbf{A}}(\lambda_1)\setminus\{0\}$.

Thm. 5.3.5 & $\mathbf{A}\mathbf{1} = 0$ $\Rightarrow$ solution of (5.3.14) is eigenvector for smallest non-zero eigenvalue of $\mathbf{A}\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$.

Algorithm 5.3.12 (Binary greyscale image segmentation (outline)).

❶ Given similarity function σ compute (sparse!) matrices $\mathbf{W}, \mathbf{D}, \mathbf{A} \in \mathbb{R}^{N,N}$, see (5.3.7), (5.3.12).

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

$$\mathcal{X} := \{i \in \{1,\dots,N\}\colon x_i^* > \underbrace{\tfrac{1}{N}\sum_{i=1}^{N} x_i^*}_{\text{mean value of entries of } \mathbf{x}^*}\} \;. \tag{5.3.15}$$

Code 5.3.13: 1st stage of segmentation of greyscale image
```
P = imread('image.pbm'); [m,n] = size(P); [A,D] = imgsegmat(P);
[V,E] = eig(full(A+D),full(D)); % grossly inefficient!
xs = reshape(V(:,2),m,n); Xidx = find(xs>(sum(sum(xs))/(n*m)));
```

1st-stage of segmentation of 31×25 greyscale pixel image (`root.pbm`, red pixels $\hat{=} \mathcal{X}$, $\sigma(x,y) = \exp(-(x-y/10)^2)$)

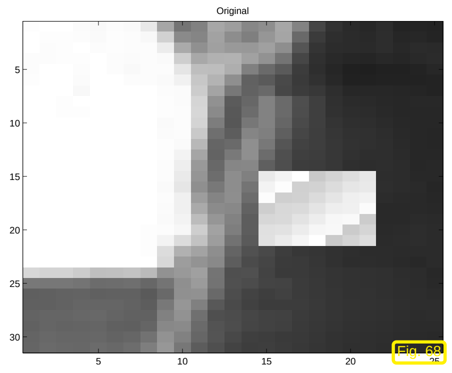


Fig. 68

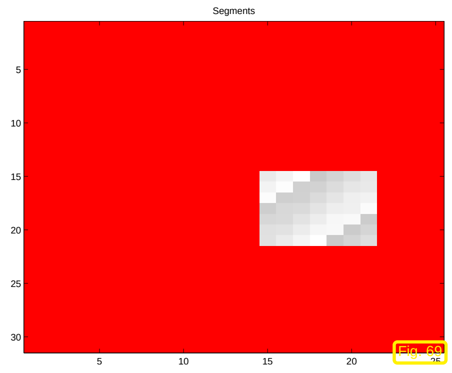


Fig. 69

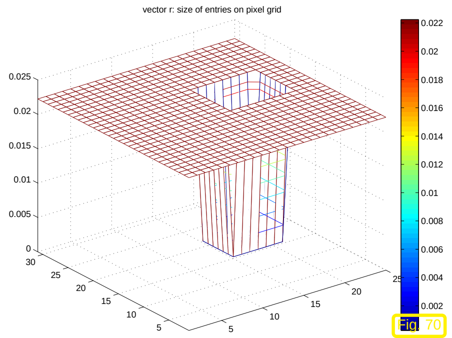


Fig. 70

Image from Fig. 68:

◁ eigenvector $\mathbf{x}^*$ plotted on pixel grid

To identify more segments, the same algorithm is *recursively applied* to segment parts of the image already determined.

Practical segmentation algorithms rely on many more steps of which the above algorithm is only one, 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 [36].

◇

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:

Smallest (in modulus) EV of $\mathbf{A}$ = $\left(\text{Largest (in modulus) EV of } \mathbf{A}^{-1}\right)^{-1}$

▶ Direct power method ($\to$ Sect. 5.3.1) for $\mathbf{A}^{-1}$ = inverse iteration

Code 5.3.14: inverse iteration for computing $\lambda_{\min}(\mathbf{A})$ and associated eigenvector

```
function [lmin,y] = invit(A,tol)
[L,U] = lu(A); n = size(A,1); x = rand(n,1); x = x/norm(x);
y = U\(L\x); lmin = 1/norm(y); y = y*lmin; lold = 0;
while(abs(lmin-lold) > tol*lmin)
  lold = lmin; x = y; y = U\(L\x); lmin = 1/norm(y); y = y*lmin;
end
```

Note: reuse of LU-factorization, see Rem. 2.2.6

Remark 5.3.15 (Shifted inverse iteration).

More general task:

For $\alpha \in \mathbb{C}$ find $\lambda \in \sigma(\mathbf{A})$ such that $|\alpha - \lambda| = \min\{|\alpha - \mu|, \mu \in \sigma(\mathbf{A})\}$

▶ Shifted inverse iteration:

$$\mathbf{z}^{(0)} \text{ arbitrary}, \quad \mathbf{w} = (\mathbf{A} - \alpha\mathbf{I})^{-1}\mathbf{z}^{(k-1)}, \quad \mathbf{z}^{(k)} := \frac{\mathbf{w}}{\|\mathbf{w}\|_2}, \quad k = 1,2,\dots, \tag{5.3.16}$$

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 5.3.13).

What if "by accident" $\alpha \in \sigma(\mathbf{A})$ ($\Leftrightarrow \mathbf{A} - \alpha\mathbf{I}$ singular) **?**

Stability of Gaussian elimination/LU-factorization ($\longrightarrow$ Sect. 2.5.3) will ensure that "$\mathbf{w}$ from (5.3.16) 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. 5.3.2 ➢ Convergence of shifted inverse iteration for $\mathbf{A}^H = \mathbf{A}$:

Asymptotic linear convergence, Rayleigh quotient $\rightarrow \lambda_j$ with rate

$$\frac{|\lambda_j - \alpha|}{\min\{|\lambda_i - \alpha|, i \neq j\}} \quad \text{with} \quad \lambda_j \in \sigma(\mathbf{A}) \ , \quad |\alpha - \lambda_j| \leq |\alpha - \lambda| \quad \forall \lambda \in \sigma(\mathbf{A}) \ .$$

▶ 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 5.3.16 (Rayleigh quotient iteration).

Rayleigh quotient iteration

(for normal $\mathbf{A} \in \mathbb{K}^{n,n}$)

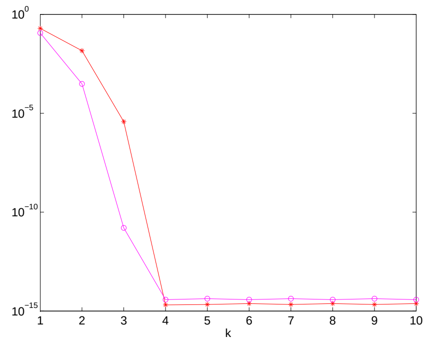
preserves sparsity, see Sect. 2.6.2

MATLAB-CODE : Rayleigh quotient iteration
```
function [z,lmin] = rqui(A,tol,maxit)
alpha = 0; n = size(A,1);
z = rand(size(A,1),1); z = z/norm(z);
for i=1:maxit
  z = (A-alpha*speye(n))\z;
  z = z/norm(z); lmin=dot(A*z,z);
  if (abs(alpha-lmin) < tol), break; end;
  alpha = lmin;
end
```
(5.3.17)

- Drawback compared with Code 5.3.13: 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. 5.3.15.

Example 5.3.17 (Rayleigh quotient iteration).

Monitored: iterates of Rayleigh quotient iteration (5.3.17) 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 : $|\lambda_{\min} - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})|$

$*$: $\left\|\mathbf{z}^{(k)} - \mathbf{x}_j\right\|$, $\lambda_{\min} = \lambda_j$, $\mathbf{x}_j \in \mathrm{Eig}_{\mathbf{A}}(\lambda_j)$,

: $\left\|\mathbf{x}_j\right\|_2 = 1$

k	$\lvert\lambda_{\min} - \rho_{\mathbf{A}}(\mathbf{z}^{(k)})\rvert$	$\left\|\mathbf{z}^{(k)} - \mathbf{x}_j\right\|$
1	0.09381702342056	0.20748822490698
2	0.00029035607981	0.01530829569530
3	0.00000000001783	0.00000411928759
4	0.00000000000000	0.00000000000000
5	0.00000000000000	0.00000000000000

Theorem 5.3.6. *If* $\mathbf{A} = \mathbf{A}^H$, *then* $\rho_{\mathbf{A}}(\mathbf{z}^{(k)})$ *converges locally of order* 3 *(→ Def. 3.1.7) to the smallest eigenvalue (in modulus), when* $\mathbf{z}^{(k)}$ *are generated by the Rayleigh quotient iteration* (5.3.17).

◇

5.3.3 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 5.3.13) and Rayleigh quotient iteration (5.3.17).

? What if direct solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ not feasible ?

This can happen, in case

- for large sparse $\mathbf{A}$ the amount of fill-in exhausts memory, despite sparse elimination techniques (→ 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. 4. 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. 4.3.1) 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 $\mathbf{B}^{-1} \approx \mathbf{A}^{-1}$

➢ $\mathbf{B} \hat{=}$ Preconditioner for $\mathbf{A}$, see Def. 4.3.1

Possible to replace $\mathbf{A}^{-1}$ with $\mathbf{B}^{-1}$ in inverse iteration **?**

NO, because we are not interested in smallest eigenvalue of $\mathbf{B}$!

Replacement $\mathbf{A}^{-1} \rightarrow \mathbf{B}^{-1}$ possible only when applied to residual quantity

residual quantity = quantity that $\rightarrow 0$ in the case of convergence to exact solution

Natural residual quantity for eigenvalue problem $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$:

$$\mathbf{r} := \mathbf{A}\mathbf{z} - \rho_{\mathbf{A}}(\mathbf{z})\mathbf{z} \quad , \quad \rho_{\mathbf{A}}(\mathbf{z}) = \text{Rayleigh quotient} \rightarrow \text{Def. 5.3.1} .$$

Note: only *direction* of $\mathbf{A}^{-1}\mathbf{z}$ matters in inverse iteration (5.3.16)

$$(\mathbf{A}^{-1}\mathbf{z}) \parallel (\mathbf{z} - \mathbf{A}^{-1}(\mathbf{A}\mathbf{z} - \rho_{\mathbf{A}}(\mathbf{z})\mathbf{z})) \quad \Rightarrow \quad \text{defines same next iterate!}$$

[Preconditioned inverse iteration (PINVIT) for s.p.d. $\mathbf{A}$]

$$\mathbf{z}^{(0)} \text{ arbitrary,} \quad \begin{aligned} \mathbf{w} &= \mathbf{z}^{(k-1)} - \mathbf{B}^{-1}(\mathbf{A}\mathbf{z}^{(k-1)} - \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{z}^{(k-1)}) , \\ \mathbf{z}^{(k)} &= \frac{\mathbf{w}}{\|\mathbf{w}\|_2} , \end{aligned} \quad k = 1, 2, \ldots . \tag{5.3.18}$$

Code 5.3.18: preconditioned inverse iteration (5.3.18)

```
function [lmin,z,res] = pinvit(evalA,n,invB,tol,maxit)
z = (1:n)'; z = z/norm(z); res = []; rho = 0;
for i=1:maxit
  v = evalA(z); rhon = dot(v,z); r = v - rhon*z;
  z = z - invB(r); z = z/norm(z); res = [res; rhon];
  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 5.3.19 (Convergence of PINVIT).

S.p.d. matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, tridiagonal preconditioner, see Ex. 4.3.4

```
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 5.3.17: $|\rho_{\mathbf{A}}(\mathbf{z}^{(k)}) - \lambda_{\min}(\mathbf{A})|$

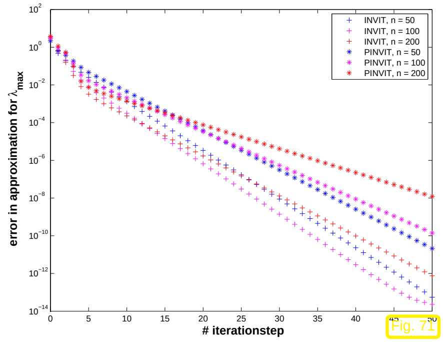


Fig. 71

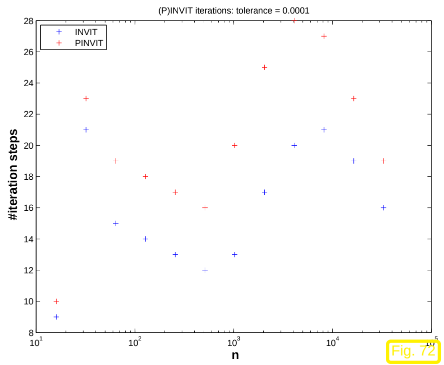


Fig. 72

Observation: linear convergence of eigenvectors also for PINVIT.

◇

Theory:
- linear convergence of (5.3.18)
- fast convergence, if spectral condition number $\kappa(\mathbf{B}^{-1}\mathbf{A})$ small

The theory of PINVIT is based on the identity

$$\mathbf{w} = \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{A}^{-1}\mathbf{z}^{(k-1)} + (\mathbf{I} - \mathbf{B}^{-1}\mathbf{A})(\mathbf{z}^{(k-1)} - \rho_{\mathbf{A}}(\mathbf{z}^{(k-1)})\mathbf{A}^{-1}\mathbf{z}^{(k-1)}) \; . \qquad (5.3.19)$$

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.

5.3.4 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.

Recall that this task has to be tackled in step ❷ of the image segmentation algorithm Alg. 5.3.12.

Preliminary considerations:

According to Cor. 5.1.7: 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 $\mathbf{D} = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$, $\lambda_j \in \mathbb{R}$, $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ $\mathbf{U}$ orthogonal. Thus, $\mathbf{u}_j := (\mathbf{U})_{:,j}$, $j = 1, \dots, n$, are (mutually orthogonal) eigenvectors of $\mathbf{A}$.

Assume $0 \leq \lambda_1 \leq \dots \leq \lambda_{n-2} < \lambda_{n-1} < \lambda_n$ (largest eigenvalues are simple).

If we just carry out the direct power iteration (5.3.5) for two vectors both sequences will converge to the largest (in modulus) eigenvector. However, we recall that all eigenvectors are mutually orthogonal. This suggests that we orthogonalize the iterates of the second power iteration (that is to yield the eigenvector for the second largest eigenvalue) with respect to those of the first. This idea spawns the following iteration, *cf.* Gram-Schmidt orthogonalization in (4.2.6):

Code 5.3.20: one step of subspace power iteration, $m = 2$

```
function [v,w] = sspowitstep(A,v,w)
v = A*v; w = A*w; v = v/norm(v); w = w - dot(v,w)*v; w = w/norm(w);
```

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 (5.3.5). 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).

$$\mathbf{z} := \mathbf{A}\mathbf{w} - (\mathbf{v}_0^T \mathbf{A}\mathbf{w})\mathbf{v}_0 = 0 \cdot \mathbf{u}_n + \sum_{i=1}^{n-1} \lambda_j \beta_j \mathbf{u}_j \; ,$$

$$\mathbf{w}^{(\mathrm{new})} := \frac{\mathbf{z}}{\|\mathbf{z}\|_2} = \Big(\sum_{i=1}^{n-1} \lambda_j^2 \beta_j^2\Big)^{-1/2} \sum_{i=1}^{n-1} \lambda_j \beta_j \mathbf{u}_j \; .$$

The sequence $\mathbf{w}^{(k)}$ produced by repeated application of the mapping given by Code 5.3.19 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 $\widetilde{\mathbf{A}} := \mathbf{U}\,\mathrm{diag}(\lambda_1, \dots, \lambda_{n-1}, 0)$. Its convergence will be governed by the relative gap $\lambda_{n-1}/\lambda_{n-2}$, see Thm. 5.3.2.

Remark 5.3.21 (Generalized normalization).

The following two MATLAB code snippets perform the same function, *cf.* Code 5.3.19:

```
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. 2.8.9

△

We revisit the above setting, Code 5.3.19. 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. 5.3.5

$$\mathbf{u}_n = \operatorname{argmax}_{\mathbf{x}\in\mathbb{R}^n} \rho_{\mathbf{A}}(\mathbf{x}) \quad , \quad \mathbf{u}_{n-1} = \operatorname{argmax}_{\mathbf{x}\in\mathbb{R}^n,\, \mathbf{x}\perp\mathbf{u}_n} \rho_{\mathbf{A}}(\mathbf{x}) \; . \tag{5.3.20}$$

Idea: maximize Rayleigh quotient over $\operatorname{Span}\{\mathbf{v},\mathbf{w}\}$, where $\mathbf{v}$, $\mathbf{w}$ are output by Code 5.3.19. This leads to the optimization problem

$$(\alpha^*,\beta^*) := \underset{\alpha,\beta\in\mathbb{R},\,\alpha^2+\beta^2=1}{\operatorname{argmax}} \rho_{\mathbf{A}}(\alpha\mathbf{v}+\beta\mathbf{w}) = \underset{\alpha,\beta\in\mathbb{R},\,\alpha^2+\beta^2=1}{\operatorname{argmax}} \rho_{(\mathbf{v},\mathbf{w})^T\mathbf{A}(\mathbf{v},\mathbf{w})}\left(\begin{pmatrix}\alpha\\ \beta\end{pmatrix}\right) . \tag{5.3.21}$$

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 5.3.19.

Then, orthogonalizing $\mathbf{w}$ w.r.t $\mathbf{v}^*$ will produce a new iterate $\mathbf{w}^*$.

Again the min-max theorem Thm. 5.3.5 tells us that we can find $(\alpha^*,\beta^*)^T$ as eigenvector to the largest eigenvalue of

$$(\mathbf{v},\mathbf{w})^T\mathbf{A}(\mathbf{v},\mathbf{w})\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \lambda\begin{pmatrix}\alpha\\ \beta\end{pmatrix} . \tag{5.3.22}$$

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 (5.4.1) belonging to the smallest eigenvalue. This assumes orthonormal vectors $\mathbf{v}$, $\mathbf{w}$.

Summing up the following MATLAB-function performs these computations:

Code 5.3.22: one step of subspace power iteration, $m = 2$, with Ritz projection

```
function [v,w] = sspowitsteprp(A,v,w)
v = A*v; w = A*w; [Q,R] = qr([v,w],0); [U,D] = eig(Q'*A*Q);
ev = diag(D); [dummy,idx] = sort(abs(ev));
w = Q*U(:,idx(1)); v = Q*U(:,idx(2));
```

General technique: Ritz projection

= "projection of a (symmetric) eigenvalue problem onto a subspace"

Example: Ritz projection of $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ onto $\operatorname{Span}\{\mathbf{v},\mathbf{w}\}$:

$$(\mathbf{v},\mathbf{w})^T\mathbf{A}(\mathbf{v},\mathbf{w})\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \lambda(\mathbf{v},\mathbf{w})^T(\mathbf{v},\mathbf{w})\begin{pmatrix}\alpha\\ \beta\end{pmatrix} .$$

More general: Ritz projection of $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ onto $\operatorname{Im}(\mathbf{V})$ (subspace spanned by columns of $\mathbf{V}$)

$$\mathbf{V}^H\mathbf{A}\mathbf{V}\mathbf{w} = \lambda\mathbf{V}^H\mathbf{V}\mathbf{w} \; . \tag{5.3.23}$$

If $\mathbf{V}$ is unitary, then this generalized eigenvalue problem will become a standard linear eigenvalue problem.

Note that he orthogonalization step in Code 5.3.21 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. 2.5.5), *cf.* the discussion in Sect. 2.8.

In MATLAB-implementations the vectors $\mathbf{v}$, $\mathbf{w}$ can be collected in a matrix $\mathbf{V}\in\mathbb{R}^{n,2}$:

Code 5.3.23: one step of subspace power iteration with Ritz projection, matrix version

```
function V = sspowitsteprp(A,V)
V = A*V; [Q,R] = qr(V,0); [U,D] = eig(Q'*A*Q); V = Q*U;
```

Algorithm 5.3.24 (Subspace variant of direct power method with Ritz projection).

Assumption: $\mathbf{A} = \mathbf{A}^H \in \mathbb{K}^{n,n}$, $k \ll n$

MATLAB-CODE : Subspace variant of direct power method for s.p.d. $\mathbf{A}$

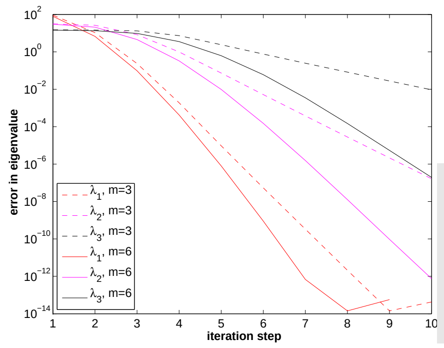
```
function [V,ev] = spowit(A,k,m,maxit)
n = size(A,1); V = rand(n,m); d = zeros(k,1);
for i=1:maxit
  V=A*V;
 [Q,R] = qr(V,0);
  T=Q'*A*Q;
  [S,D] = eig(T); [l,perm] = sort(-abs(diag(D)));
  V = Q*S(:,perm);
  if (norm(d+l(1:k)) < tol), break; end;
  d = -l(1:k);
end
V = V(:,1:k); ev = diag(D(perm(1:k),perm(1:k)));
```

(5.3.24)

Ritz projection

Generalized normalization to $\|\mathbf{z}\|_2 = 1$

Example 5.3.25 (Convergence of subspace variant of direct power method).



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

$\diamond$

Remark 5.3.26 (Subspace power methods).

Analoguous to Alg. 5.3.24: construction of subspace variants of inverse iteration ($\rightarrow$ Code 5.3.13), PINVIT (5.3.18), and Rayleigh quotient iteration (5.3.17).

$\triangle$

5.4 Krylov Subspace Methods

All power methods ($\rightarrow$ Sect. 5.3) for the eigenvalue problem (EVP) $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ only rely on the last iterate to determine the next one (1-point methods, *cf.* (3.1.1))

➣ NO MEMORY, *cf.* discussion in the beginning of Sect. 4.2.

"Memory for power iterations": pursue same idea that led from the gradient method, Alg. 4.1.4, to the conjugate gradient method, Alg. 4.2.1: use information from previous iterates to achieve efficient minimization over larger and larger subspaces.

Min-max theorem, Thm. 5.3.5	:	$\mathbf{A} = \mathbf{A}^H$	$\Rightarrow$	EVPs	$\Leftrightarrow$	Finding extrema/stationary points of Rayleigh quotient ($\rightarrow$ Def. 5.3.1)

Setting: EVP $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ for real s.p.d. ($\rightarrow$ Def. 2.7.1) matrix $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n,n}$

notations used below: $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$: eigenvalues of $\mathbf{A}$, counted with multiplicity, see Def. 5.1.1,

$\mathbf{u}_1, \dots, \mathbf{u}_n \hat{=}$ corresponding orthonormal eigenvectors, *cf.* Cor. 5.1.7.

$$\blacktriangleright\quad \mathbf{A}\mathbf{U} = \mathbf{D}\mathbf{U} \quad , \quad \mathbf{U} = (\mathbf{u}_1, \dots, \mathbf{u}_n) \in \mathbb{R}^{n,n} \ , \quad \mathbf{D} = \operatorname{diag}(\lambda_1, \dots, \lambda_n) \ .$$

Idea: Better $\mathbf{z}^{(k)}$ from Ritz projection onto $V := \operatorname{Span}\left\{\mathbf{z}^{(0)}, \dots, \mathbf{z}^{(k)}\right\}$
(= space spanned by previous iterates)

Recall ($\rightarrow$ Code 5.3.22) Ritz projection of an EVP $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ onto a subspace $V := \operatorname{Span}\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$, $m < n$ ➡ smaller $m \times m$ generalized EVP

$$\underbrace{\mathbf{V}^T\mathbf{A}\mathbf{V}}_{:=\mathbf{H}}\mathbf{x} = \lambda\mathbf{V}^T\mathbf{V}\mathbf{x} \quad , \quad \mathbf{V} := (\mathbf{v}_1, \dots, \mathbf{v}_m) \in \mathbb{R}^{n,m} \ . \tag{5.4.1}$$

From min-max theorem Thm. 5.3.5:

$$\mathbf{u}_n \in V \quad \Rightarrow \quad \text{largest eigenvalue of (5.4.1)} = \lambda_{\max}(\mathbf{A}) \ ,$$

$$\mathbf{u}_1 \in V \quad \Rightarrow \quad \text{smallest eigenvalue of (5.4.1)} = \lambda_{\min}(\mathbf{A}) \ .$$

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 (5.4.1) 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 (5.4.1).

$\blacktriangleright$ For direct power method (5.3.5): $\mathbf{z}^{(k)} || \mathbf{A}^k\mathbf{z}^{(0)}$

$$V = \operatorname{Span}\left\{\mathbf{z}^{(0)}, \mathbf{A}\mathbf{z}^{(0)}, \dots, \mathbf{A}^{(k)}\mathbf{z}^{(0)}\right\} = \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{z}^{(0)}) \quad \text{a Krylov space}, \rightarrow \text{Def. 4.2.3} \ . \tag{5.4.2}$$

Code 5.4.1: Ritz projections onto Krylov space (5.4.2)

```
function [V,D] = kryleig(A,m)
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 (5.4.2), *cf.* Code 5.3.21.

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 5.4.2 (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
```

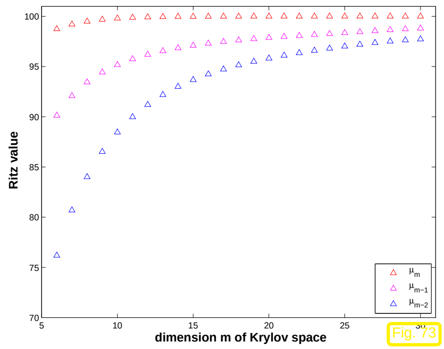


Fig. 73

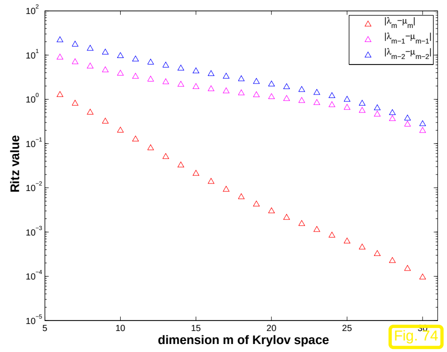


Fig. 74

Observation: "vaguely linear" convergence of largest Ritz values to largest eigenvalues. Fastest convergence of largest Ritz value ⟶ largest eigenvalue of $\mathbf{A}$

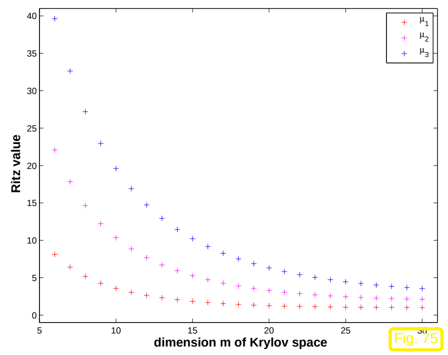


Fig. 75

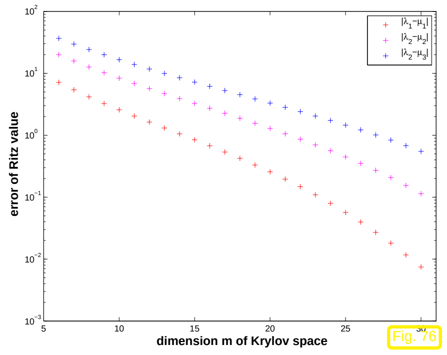


Fig. 76

Observation: *Also the smallest Ritz values converge* "vaguely linearly" to the smallest eigenvalues of $\mathbf{A}$. Fastest convergence of smallest Ritz value ⟶ smallest eigenvalue of $\mathbf{A}$.

◇

? Why do smallest Ritz values converge to smallest eigenvalues of $\mathbf{A}$?

Consider direct power method (5.3.5) for $\widetilde{\mathbf{A}} := \nu\mathbf{I} - \mathbf{A}$, $\nu > \lambda_{\max}(\mathbf{A})$:

$$\mathbf{z}^{(0)} \text{ arbitrary }, \quad \widetilde{\mathbf{z}}^{(k+1)} = \frac{(\nu\mathbf{I}-\mathbf{A})\widetilde{\mathbf{z}}^{(k)}}{\left\|(\nu\mathbf{I}-\mathbf{A})\widetilde{\mathbf{z}}^{(k)}\right\|_2} \tag{5.4.3}$$

As $\sigma(\nu\mathbf{I}-\mathbf{A}) = \nu - \sigma(\mathbf{A})$ and eigenspaces agree, we infer from Thm. 5.3.2

$$\lambda_1 < \lambda_2 \quad \Rightarrow \quad \mathbf{z}^{(k)} \xrightarrow{k\to\infty} \mathbf{u}_1 \quad \& \quad \rho_{\mathbf{A}}(\mathbf{z}^{(k)}) \xrightarrow{k\to\infty} \lambda_1 \quad \text{linearly} . \tag{5.4.4}$$

By the binomial theorem (also applies to matrices, if they commute)

$$(\nu\mathbf{I}-\mathbf{A})^k = \sum_{j=0}^{k} \binom{k}{j} \nu^{k-j}\mathbf{A}^j \quad \Rightarrow \quad (\nu\mathbf{I}-\mathbf{A})^k\widetilde{\mathbf{z}}^{(0)} \in \mathcal{K}_k(\mathbf{A},\mathbf{z}^{(0)}) ,$$

$$\boxed{\mathcal{K}_k(\nu\mathbf{I}-\mathbf{A},\mathbf{x}) = \mathcal{K}_k(\mathbf{A},\mathbf{x})} . \tag{5.4.5}$$

➣ $\mathbf{u}_1$ 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 $\lambda_{\min}(\mathbf{A})$.

Recall from Sect. 4.2.2 , Lemma 4.2.5:

Residuals $\mathbf{r}_0,\dots,\mathbf{r}_{m-1}$ generated in CG iteration, Alg. 4.2.1 applied to $\mathbf{A}\mathbf{x}=\mathbf{z}$ with $\mathbf{x}^{(0)}=0$, provide *orthogonal basis* for $\mathcal{K}_m(\mathbf{A},\mathbf{z})$ (, if $\mathbf{r}_k \neq 0$).

▶ Inexpensive Ritz projection of $\mathbf{A}\mathbf{x}=\lambda\mathbf{x}$ onto $\mathcal{K}_m(\mathbf{A},\mathbf{z})$:

orthogonal matrix

$$\mathbf{V}_m^T\mathbf{A}\mathbf{V}_m\mathbf{x}=\lambda\mathbf{x}\,,\quad \mathbf{V}_m := \left(\frac{\mathbf{r}_0}{\|\mathbf{r}_0\|},\dots,\frac{\mathbf{r}_{m-1}}{\|\mathbf{r}_{m-1}\|}\right)\in\mathbb{R}^{n,m}\,. \tag{5.4.6}$$

recall: residuals generated by *short recursions*, see Alg. 4.2.1

Lemma 5.4.1 (Tridiagonal Ritz projection from CG residuals)**.**
$\mathbf{V}_m^T\mathbf{A}\mathbf{V}_m$ *is a tridiagonal matrix.*

Proof. Lemma 4.2.5: $\{\mathbf{r}_0,\dots,\mathbf{r}_{\ell-1}\}$ is an orthogonal basis of $\mathcal{K}_\ell(\mathbf{A},\mathbf{r}_0)$, if all the residuals are non-zero. As $\mathbf{A}\mathcal{K}_{\ell-1}(\mathbf{A},\mathbf{r}_0)\subset\mathcal{K}_\ell(\mathbf{A},\mathbf{r}_0)$, we conclude the orthogonality $\mathbf{r}_m^T\mathbf{A}\mathbf{r}_j$ for all $j=0,\dots,m-2$. Since

$$\left(\mathbf{V}_m^T\mathbf{A}\mathbf{V}_m\right)_{ij}=\mathbf{r}_{i-1}^T\mathbf{A}\mathbf{r}_{j-1}\,,\quad 1\leq i,j\leq m\,,$$

the assertion of the theorem follows. □

$$\mathbf{V}_l^H\mathbf{A}\mathbf{V}_l=\begin{pmatrix}\alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & & \\ & & & & \ddots & \\ & & & \ddots & \ddots & \beta_{k-1} \\ & & & & \beta_{k-1} & \alpha_k\end{pmatrix}=:\mathbf{T}_l\in\mathbb{K}^{k,k}\quad\text{[tridiagonal matrix]}$$

Algorithm for computing $\mathbf{V}_l$ and $\mathbf{T}_l$:

Lanczos process

Computational effort/step:

1× $\mathbf{A}$×vector
2 dot products
2 AXPY-operations
1 division

Code 5.4.3: Lanczos process
```
function [V,alph,bet] = lanczos(A,k,z0)
V = z0/norm(z0);
alph=zeros(k,1);
bet = zeros(k,1);
for j=1:k
  p = A*V(:,j); alph(j) = dot(p,V(:,j));
  w = p - alph(j)*V(:,j);
  if (j > 1), w = w - bet(j-1)*V(:,j-1); end;
  bet(j) = norm(w); V = [V,w/bet(j)];
end
bet = bet(1:end-1);
```

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 5.4.2 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. 5.3.7.

Convergence (what we expect from the above considerations) ⟶ [13, Sect. 8.5])

$$\text{In } l\text{-th step:}\quad \lambda_n\approx\mu_l^{(l)}\,,\ \lambda_{n-1}\approx\mu_{l-1}^{(l)}\,,\dots,\ \lambda_1\approx\mu_1^{(l)}\,,$$
$$\sigma(\mathbf{T}_l)=\{\mu_1^{(l)},\dots,\mu_l^{(l)}\},\quad \mu_1^{(l)}\leq\mu_2^{(l)}\leq\dots\leq\mu_l^{(l)}\,.$$

Example 5.4.4 (Lanczos process for eigenvalue computation).

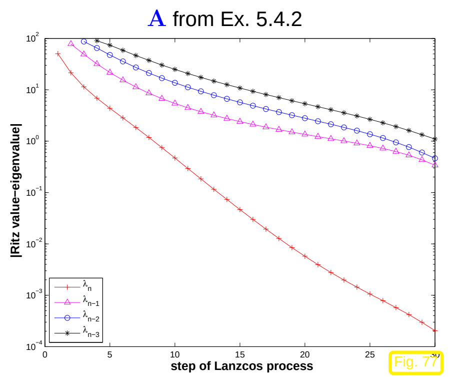


Fig. 77

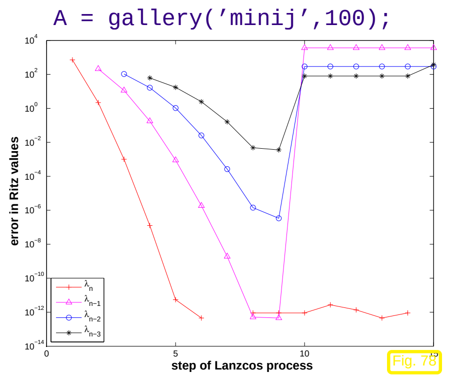


Fig. 78

Observation: same as in Ex. 5.4.2, 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. 4.2.4

◇

Example 5.4.5 (Impact of roundoff on Lanczos process).

$\mathbf{A} \in \mathbb{R}^{10,10}$, $a_{ij} = \min\{i,j\}$. `A = gallery('minij',10);`

Computed by `[V,alpha,beta] = lanczos(A,n,ones(n,1));`, see Code 5.4.2:

$$\mathbf{T} = \begin{pmatrix}
38.500000 & 14.813845 & & & & & & & & \\
14.813845 & 9.642857 & 2.062955 & & & & & & & \\
 & 2.062955 & 2.720779 & 0.776284 & & & & & & \\
 & & 0.776284 & 1.336364 & 0.385013 & & & & & \\
 & & & 0.385013 & 0.826316 & 0.215431 & & & & \\
 & & & & 0.215431 & 0.582380 & 0.126781 & & & \\
 & & & & & 0.126781 & 0.446860 & 0.074650 & & \\
 & & & & & & 0.074650 & 0.363803 & 0.043121 & \\
 & & & & & & & 0.043121 & 3.820888 & 11.991094 \\
 & & & & & & & & 11.991094 & 41.254286
\end{pmatrix}$$

$$\sigma(\mathbf{A}) = \{0.255680, 0.273787, 0.307979, 0.366209, 0.465233, 0.643104, 1.000000, 1.873023, 5.048917, 44.766069\}$$
$$\sigma(\mathbf{T}) = \{0.263867, 0.303001, 0.365376, 0.465199, 0.643104, 1.000000, 1.873023, 5.048917, 44.765976, 44.766069\}$$

► Uncanny cluster of computed eigenvalues of $\mathbf{T}$ ("ghost eigenvalues", [18, 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. 4.2.4

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 4.2.5

• use explicit Gram-Schmidt orthogonalization

Details: inductive approach: given $\{\mathbf{v}_1,\ldots,\mathbf{v}_l\}$ ONB of $\mathcal{K}_l(\mathbf{A},\mathbf{z})$

$$\blacktriangleright\quad \widetilde{\mathbf{v}}_{l+1} := \mathbf{A}\mathbf{v}_l - \sum_{j=1}^{l}(\mathbf{v}_j^H\mathbf{A}\mathbf{v}_l)\,\mathbf{v}_j\,,\quad \mathbf{v}_{l+1} := \frac{\widetilde{\mathbf{v}}_{l+1}}{\|\widetilde{\mathbf{v}}_{l+1}\|_2} \;\Rightarrow\; \mathbf{v}_{l+1} \perp \mathcal{K}_l(\mathbf{A},\mathbf{z})\,. \tag{5.4.7}$$

(Gram-Schmidt, *cf.* (4.2.6)) 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 of $\mathbf{A}$: $O(nkl^2)$

`H(l+1,l) = 0` ➞ STOP !

Code 5.4.6: Arnoldi process

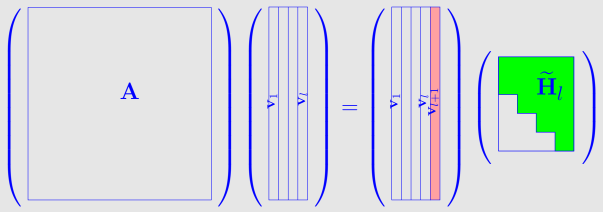
```
function [V,H] = arnoldi(A,k,v0)
V = [v0/norm(v0)];
H = zeros(k+1,k);
for l=1:k
  vt = A*V(:,l);
  for j=1:l
    H(j,l) = dot(V(:,j),vt);
    vt = vt - H(j,l)*V(:,j);
  end
  H(l+1,l) = norm(vt);
  if (H(l+1,l) == 0), break; end
  V = [V, vt/H(l+1,l)];
end
```

If it does not stop prematurely, the Arnoldi process of Code 5.4.5 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 5.4.5, meaning of output H:

$$\mathbf{V}_l = [\mathbf{v}_1,\dots,\mathbf{v}_l] \;:\; \mathbf{A}\mathbf{V}_l = \mathbf{V}_{l+1}\widetilde{\mathbf{H}}_l \;,\; \widetilde{\mathbf{H}}_l \in \mathbb{K}^{l+1,l} \text{ mit } \widetilde{h}_{ij} = \begin{cases} \mathbf{v}_i^H\mathbf{A}\mathbf{v}_j & \text{, if } i \le j\,, \\ \|\widetilde{\mathbf{v}}_i\|_2 & \text{, if } i = j+1\,, \\ 0 & \text{else.}\end{cases}$$

➡ $\widetilde{\mathbf{H}}_l$ = non-square upper Hessenberg matrices



Translate Code 5.4.5 to matrix calculus:

Lemma 5.4.2 (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, $l \le n$, 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* $1 \le i,j \le l$,

(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 5.4.5. □

Remark 5.4.7 (Arnoldi process and Ritz projection).

Interpretation of Lemma 5.4.2 (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:

In l-th step: $\lambda_n \approx \mu_l^{(l)}$, $\lambda_{n-1} \approx \mu_{l-1}^{(l)}, \dots, \lambda_1 \approx \mu_1^{(l)}$,
$\sigma(\mathbf{H}_l) = \{\mu_1^{(l)},\dots,\mu_l^{(l)}\}$, $|\mu_1^{(l)}| \le |\mu_2^{(l)}| \le \dots \le |\mu_l^{(l)}|$.

Code 5.4.8: 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 $\mathbf{A} \in \mathbb{C}^{n,n}$

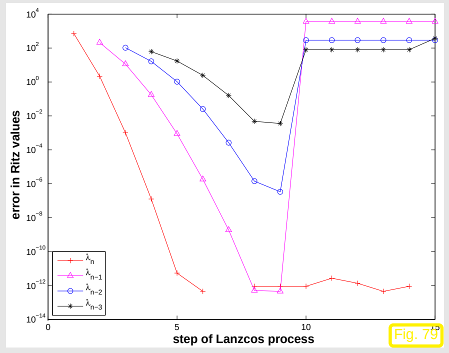
1 $\mathbf{A}\times$vector per step (➣ attractive for sparse matrices)

However: required storage increases with number of steps, *cf.* situation with GMRES, Sect. 4.4.1.

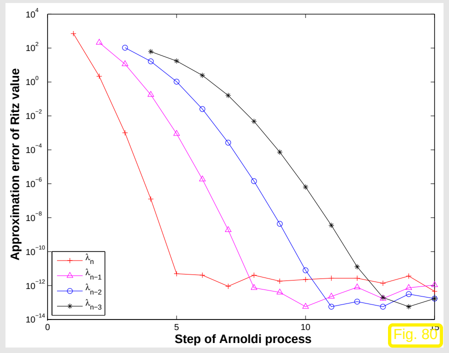
Heuristic termination criterion

Example 5.4.9 (Stabilty of Arnoldi process).

$\mathbf{A} \in \mathbb{R}^{100,100}$, $a_{ij} = \min\{i,j\}$.   `A = gallery('minij',100);`



Lanczos process: Ritz values



Arnoldi process: Ritz values

Ritz values during Arnoldi process for `A = gallery('minij',10);` ↔ Ex. 5.4.4

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 (→ Sect. 2.5.2, Def. 2.5.5) compared to the Lanczos process. ◇

Example 5.4.10 (Eigenvalue computation with Arnoldi process).

Eigenvalue approximation from Arnoldi process for *non-symmetric* $\mathbf{A}$, initial vector `ones(100,1);`

```
n=100;
M=full(gallery('tridiag',-0.5*ones(n-1,1),2*ones(n,1),-1.5*ones(n-1,1)));
A=M*diag(1:n)*inv(M);
```

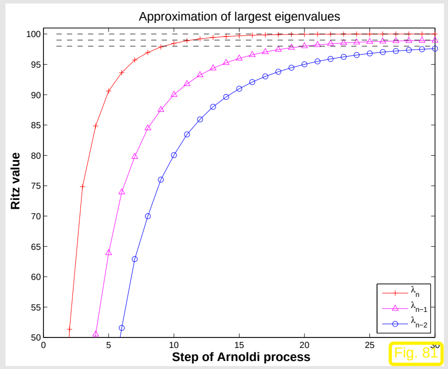


Fig. 81

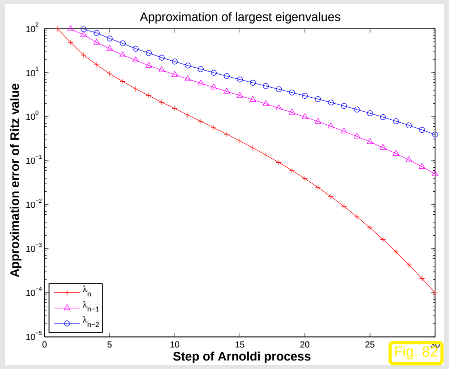


Fig. 82

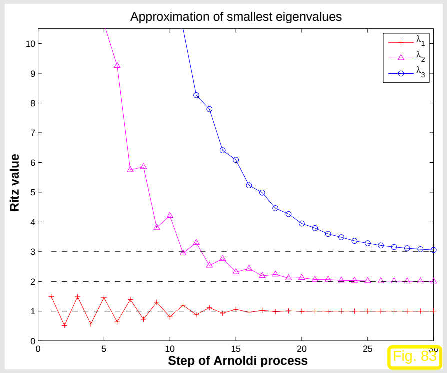


Fig. 83

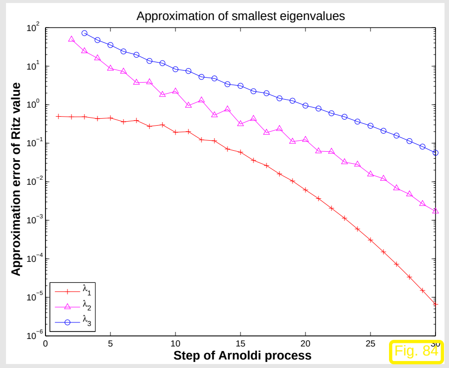


Fig. 84

Observation: "vaguely linear" convergence of largest and smallest eigenvalues, *cf.* Ex. 5.4.2. ◇

Krylov subspace iteration methods (= Arnoldi process, Lanczos process) attractive for computing *a few* of the largest/smallest eigenvalues and associated eigenvectors of *large sparse matrices*.

Remark 5.4.11 (Krylov subspace methods for generalized EVP).

Adaptation of Krylov subspace iterative eigensolvers to generalized EVP: $\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}$, $\mathbf{B}$ s.p.d.: replace Euclidean inner product with "$\mathbf{B}$-inner product" $(\mathbf{x},\mathbf{y}) \mapsto \mathbf{x}^H\mathbf{B}\mathbf{y}$. △

MATLAB-functions:

`d = eigs(A,k,sigma)`	: k largest/smallest eigenvalues of $\mathbf{A}$
`d = eigs(A,B,k,sigma)`	: k largest/smallest eigenvalues for generalized EVP $\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}$,$\mathbf{B}$ 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.

5.5 Singular Value Decomposition

Remark 5.5.1 (Principal component analysis (PCA)).

Given: n data points $\mathbf{a}_j \in \mathbb{R}^m$, $j = 1, \dots, n$, in m-dimensional (feature) space

Conjectured: "linear dependence": $\mathbf{a}_j \in V$, $V \subset \mathbb{R}^m$ p-dimensional subspace, $p < \min\{m, n\}$ *unknown*
(➢ possibility of dimensional reduction)

Task (PCA): determine (minimal) p and (orthonormal basis of) V

Perspective of linear algebra:

Conjecture $\Leftrightarrow$ $\operatorname{rank}(\mathbf{A}) = p$ for $\mathbf{A} := (\mathbf{a}_1, \dots, \mathbf{a}_n) \in \mathbb{R}^{m,n}$, $\operatorname{Im}(\mathbf{A}) = V$

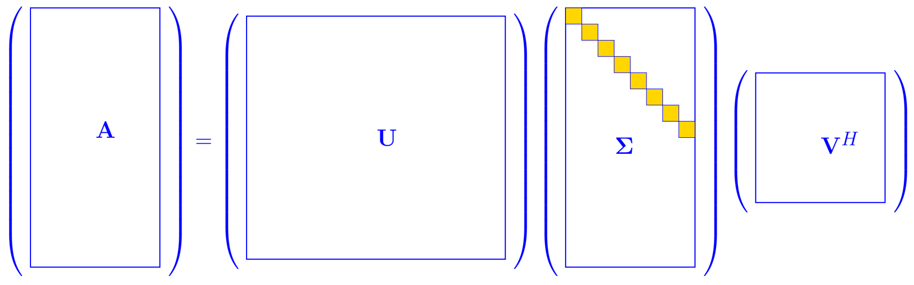
Extension: Data affected by measurement errors
(but conjecture upheld for unperturbed data)

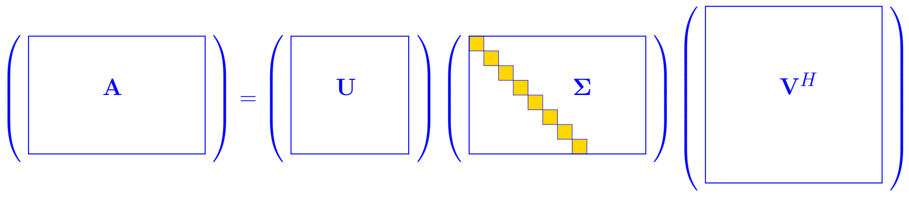
$\triangle$

Theorem 5.5.1. *For any* $\mathbf{A} \in \mathbb{K}^{m,n}$ *there are unitary matrices* $\mathbf{U} \in \mathbb{K}^{m,m}$, $\mathbf{V} \in \mathbb{K}^{n,n}$ *and a (generalized) diagonal* $^{(*)}$ *matrix* $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m,n}$, $p := \min\{m, n\}$, $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$ *such that*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H .$$

$(*)$: $\mathbf{\Sigma}$ (generalized) diagonal matrix $:\Leftrightarrow$ $(\mathbf{\Sigma})_{i,j} = 0$, if $i \neq j$, $1 \leq i \leq m$, $1 \leq j \leq n$.





Proof. (of Thm. 5.5.1, by induction)

[40, Thm. 4.2.3]: Continuous functions attain extremal values on compact sets (here the unit ball $\{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_2 \leq 1\}$)

$$➤ \quad \exists \mathbf{x} \in \mathbb{K}^n, \mathbf{y} \in \mathbb{K}^m , \quad \|\mathbf{x}\| = \|\mathbf{y}\|_2 = 1 : \quad \mathbf{A}\mathbf{x} = \sigma\mathbf{y} , \quad \sigma = \|\mathbf{A}\|_2 ,$$

where we used the definition of the matrix 2-norm, see Def. 2.5.2. 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

$$\mathbf{V} = (\mathbf{x} \; \widetilde{\mathbf{V}}) \in \mathbb{K}^{n,n} \quad , \quad \mathbf{U} = (\mathbf{y} \; \widetilde{\mathbf{U}}) \in \mathbb{K}^{m,m} \quad \text{are unitary.}$$

$$\blacktriangleright \quad \mathbf{U}^H\mathbf{A}\mathbf{V} = (\mathbf{y} \; \widetilde{\mathbf{U}})^H \mathbf{A} (\mathbf{x} \; \widetilde{\mathbf{V}}) = \left(\begin{array}{c|c} \mathbf{y}^H\mathbf{A}\mathbf{x} & \mathbf{y}^H\mathbf{A}\widetilde{\mathbf{V}} \\ \hline \widetilde{\mathbf{U}}^H\mathbf{A}\mathbf{x} & \widetilde{\mathbf{U}}^H\mathbf{A}\widetilde{\mathbf{V}} \end{array}\right) = \left(\begin{array}{c|c} \sigma & \mathbf{w}^H \\ \hline 0 & \mathbf{B} \end{array}\right) =: \mathbf{A}_1 .$$

$$\left\| \mathbf{A}_1 \begin{pmatrix} \sigma \\ \mathbf{w} \end{pmatrix} \right\|_2^2 = \left\| \begin{pmatrix} \sigma^2 + \mathbf{w}^H\mathbf{w} \\ \mathbf{B}\mathbf{w} \end{pmatrix} \right\|_2^2 = (\sigma^2 + \mathbf{w}^H\mathbf{w})^2 + \|\mathbf{B}\mathbf{w}\|_2^2 \geq (\sigma^2 + \mathbf{w}^H\mathbf{w})^2 ,$$

$$\|\mathbf{A}_1\|_2^2 = \sup_{0 \neq \mathbf{x} \in \mathbb{K}^n} \frac{\|\mathbf{A}_1\mathbf{x}\|_2^2}{\|\mathbf{x}\|_2^2} \geq \frac{\left\|\mathbf{A}_1 \binom{\sigma}{\mathbf{w}}\right\|_2^2}{\left\|\binom{\sigma}{\mathbf{w}}\right\|_2^2} \geq \frac{(\sigma^2 + \mathbf{w}^H\mathbf{w})^2}{\sigma^2 + \mathbf{w}^H\mathbf{w}} = \sigma^2 + \mathbf{w}^H\mathbf{w} . \tag{5.5.1}$$

$$\sigma^2 = \|\mathbf{A}\|_2^2 = \left\|\mathbf{U}^H\mathbf{A}\mathbf{V}\right\|_2^2 = \|\mathbf{A}_1\|_2^2 \overset{(5.5.1)}{\Longrightarrow} \|\mathbf{A}_1\|_2^2 = \|\mathbf{A}_1\|_2^2 + \|\mathbf{w}\|_2^2 \quad \Rightarrow \quad \mathbf{w} = 0 .$$

$$\blacktriangleright \quad \mathbf{A}_1 = \left(\begin{array}{c|c} \sigma & 0 \\ \hline 0 & \mathbf{B} \end{array}\right) .$$

Then apply induction argument to $\mathbf{B}$ $\square$.

Definition 5.5.2 (Singular value decomposition (SVD)).
The decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$ *of Thm. 5.5.1 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 5.5.3. *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* $(\mathbf{V})_{:,1},\dots,(\mathbf{V})_{:,p}$, $(\mathbf{U})_{:,1},\dots,(\mathbf{U})_{:,p}$, *respectively.*

Proof. $\mathbf{A}\mathbf{A}^H$ and $\mathbf{A}^H\mathbf{A}$ are similar (⟶ Lemma 5.1.4) to diagonal matrices with non-zero diagonal entries σ_i^2 ($\sigma_i \neq 0$), e.g.,

$$\mathbf{A}\mathbf{A}^H = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H\mathbf{V}\mathbf{\Sigma}^H\mathbf{U}^H = \mathbf{U}\underbrace{\mathbf{\Sigma}\mathbf{\Sigma}^H}_{\text{diagonal matrix}}\mathbf{U}^H . \qquad \square$$

Remark 5.5.2 (SVD and additive rank-1 decomposition).

Recall from linear algebra: rank-1 matrices are tensor products of vectors

$$\mathbf{A} \in \mathbb{K}^{m,n} \ \text{and} \ \operatorname{rank}(\mathbf{A}) = 1 \ \Leftrightarrow \ \exists \mathbf{u} \in \mathbb{K}^m, \mathbf{v} \in \mathbb{K}^n\colon \ \mathbf{A} = \mathbf{u}\mathbf{v}^H , \qquad (5.5.2)$$

because $\operatorname{rank}(\mathbf{A}) = 1$ means that $\mathbf{A}\mathbf{x} = \mu(\mathbf{x})\mathbf{u}$ for some $\mathbf{u} \in \mathbb{K}^m$ and linear form $\mathbf{x} \mapsto \mu(\mathbf{x})$. By the Riesz representation theorem the latter can be written as $\mu(\mathbf{x}) = \mathbf{v}^H\mathbf{x}$.

▶ Singular value decomposition provides additive decomposition into rank-1 matrices:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H = \sum_{j=1}^{p} \sigma_j \, (\mathbf{U})_{:,j}(\mathbf{V})_{:,j}^H . \qquad (5.5.3)$$

Remark 5.5.3 (Uniqueness of SVD).

SVD of Def. 5.5.2 is not (necessarily) unique, but the singular values are.

Assume that $\mathbf{A}$ has two singular value decompositions

$$\mathbf{A} = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^H = \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^H \ \Rightarrow \ \mathbf{U}_1 \underbrace{\mathbf{\Sigma}_1\mathbf{\Sigma}_1^H}_{=\operatorname{diag}(s_1^1,\dots,s_m^1)} \mathbf{U}_1^H = \mathbf{A}\mathbf{A}^H = \mathbf{U}_2 \underbrace{\mathbf{\Sigma}_2\mathbf{\Sigma}_2^H}_{=\operatorname{diag}(s_1^2,\dots,s_m^2)} \mathbf{U}_2^H .$$

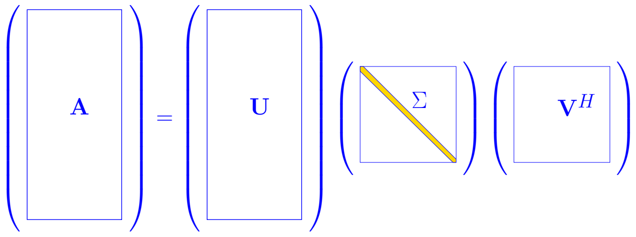
Two similar diagonal matrices are equal ! □

△

MATLAB-functions (for algorithms see [18, 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. 5.5.1
`[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:



(MATLAB) algorithm for computing SVD is (numerically) stable ⟶ Def. 2.5.5

Complexity:

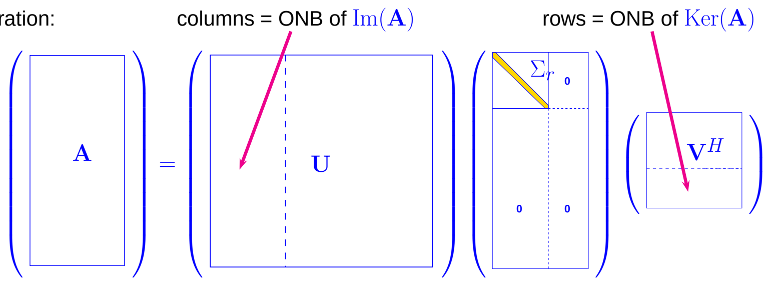
$2mn^2 + 2n^3 + O(n^2) + O(mn)$ for `s = svd(A)`,
$4m^2n + 22n^3 + O(mn) + O(n^2)$ for `[U,S,V] = svd(A)`,
$O(mn^2) + O(n^3)$ for `[U,S,V]=svd(A,0)`, $m \gg n$.

• Application of SVD: computation of rank (⟶ Def. 2.0.2), kernel and range of a matrix

Lemma 5.5.4 (SVD and rank of a matrix)**.**
If the singular values of $\mathbf{A}$ *satisfy* $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots \sigma_p = 0$, *then*

- $\operatorname{rank}(\mathbf{A}) = r$,
- $\operatorname{Ker}(\mathbf{A}) = \operatorname{Span}\left\{(\mathbf{V})_{:,r+1},\dots,(\mathbf{V})_{:,n}\right\}$,
- $\operatorname{Im}(\mathbf{A}) = \operatorname{Span}\left\{(\mathbf{U})_{:,1},\dots,(\mathbf{U})_{:,r}\right\}$.

Illustration:



columns = ONB of $\mathrm{Im}(\mathbf{A})$
rows = ONB of $\mathrm{Ker}(\mathbf{A})$
$\mathbf{A}$
=
$\mathbf{U}$
Σ_r
0
0
0
$\mathbf{V}^H$

(5.5.4)

Remark: MATLAB function r=rank(A) relies on svd(A)

Lemma 5.5.4 ▶ PCA by SVD

❶ no perturbations:

SVD: $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$ satisfies
$$\sigma_1 \geq \sigma_2 \geq \dots \sigma_p > \sigma_{p+1} = \dots = \sigma_{\min\{m,n\}} = 0\,,$$
$$V = \mathrm{Span}\,\underbrace{\{(\mathbf{U})_{:,1},\dots,(\mathbf{U})_{:,p}\}}_{\text{ONB of } V}\,.$$

❷ with perturbations:

SVD: $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$ satisfies
$$\sigma_1 \geq \sigma_2 \geq \dots \sigma_p \gg \sigma_{p+1} \approx \dots \approx \sigma_{\min\{m,n\}} \approx 0\,,$$
$$V = \mathrm{Span}\,\underbrace{\{(\mathbf{U})_{:,1},\dots,(\mathbf{U})_{:,p}\}}_{\text{ONB of } V}\,.$$

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 $\mathrm{Im}(\mathbf{A})$ has essentially dimension p. It depends on the application what one accepts as a “pronounced gap”.

Code 5.5.4: 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';
```

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.

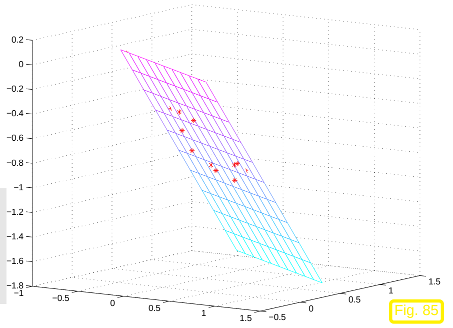


Fig. 85

Example 5.5.5 (Principal component analysis for data analysis).

$\mathbf{A} \in \mathbb{R}^{m,n}$, $m \gg n$:

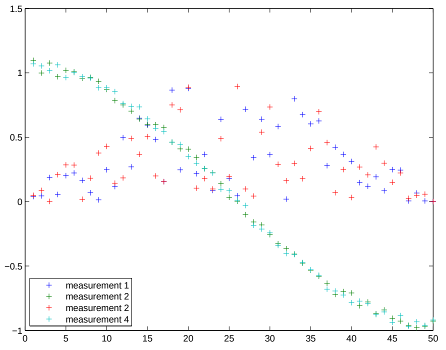
Columns $\mathbf{A}$	$\longrightarrow$	series of measurements at different times/locations etc.
Rows of $\mathbf{A}$	$\longrightarrow$	measured values corresponding to one time/location etc.

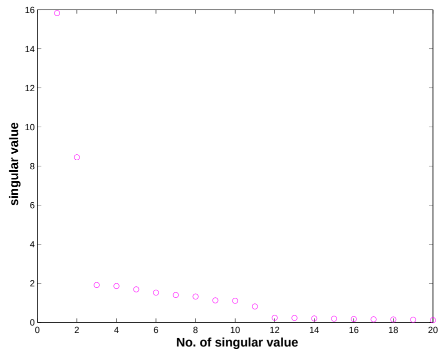
Goal: detect linear correlations

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
```





$\longleftarrow$ distribution of singular values of matrix

two dominant singular values !

measurements display linear correlation with **two** principal components

principal components	=	$\mathbf{u}_{\cdot,1}, \mathbf{u}_{\cdot,2}$	(leftmost columns of $\mathbf{U}$-matrix of SVD)
their relative weights	=	$\mathbf{v}_{\cdot,1}, \mathbf{v}_{\cdot,2}$	(leftmost columns of $\mathbf{V}$-matrix of SVD)

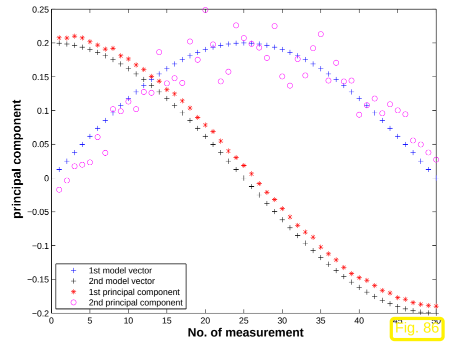


Fig. 86

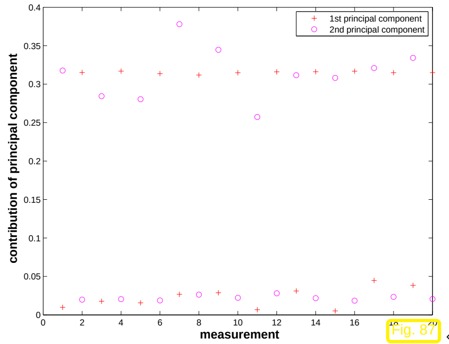


Fig. 87

◇

● Application of SVD: extrema of quadratic forms on the unit sphere

A minimization problem on the Euclidean unit sphere $\{\mathbf{x} \in \mathbb{K}^n\colon \|\mathbf{x}\|_2 = 1\}$:

$$\text{given } \mathbf{A} \in \mathbb{K}^{m,n},\ m > n, \quad \text{find } \mathbf{x} \in \mathbb{K}^n,\ \|\mathbf{x}\|_2 = 1,\quad \|\mathbf{A}\mathbf{x}\|_2 \to \min\ . \tag{5.5.5}$$

Use that multiplication with unitary matrices preserves the 2-norm ($\rightarrow$ Thm. 2.8.2) and the singular value decomposition $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$ ($\rightarrow$ Def. 5.5.2):

$$\min_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2^2 = \min_{\|\mathbf{x}\|_2=1} \left\|\mathbf{U}\Sigma\mathbf{V}^H\mathbf{x}\right\|_2^2 = \min_{\|\mathbf{V}^H\mathbf{x}\|_2=1} \left\|\mathbf{U}\Sigma(\mathbf{V}^H\mathbf{x})\right\|_2^2$$

$$= \min_{\|\mathbf{y}\|_2=1} \|\Sigma\mathbf{y}\|_2^2 = \min_{\|\mathbf{y}\|_2=1} (\sigma_1^2 y_1^2 + \cdots + \sigma_n^2 y_n^2) \geq \sigma_n^2\ .$$

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:

$$\sigma_1 = \max_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2 \quad , \quad (\mathbf{V})_{:,1} = \operatorname{argmax}_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2\ . \tag{5.5.6}$$

Recall: 2-norm of the matrix $\mathbf{A}$ ($\rightarrow$ Def. 2.5.2) is defined as the maximum in (5.5.6). Thus we have proved the following theorem:

Lemma 5.5.5 (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 5.5.6 (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 $\mathbf{A}$

$$\text{Frobenius norm and SVD:} \quad \boxed{\|\mathbf{A}\|_F^2 = \sum_{j=1}^{p} \sigma_j^2} \tag{5.5.7}$$

✎ 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 5.5.7 (best low rank approximation)**.**
Let $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$ *be the SVD of* $\mathbf{A} \in \mathbb{K}^{m.n}$ *(→ Thm. 5.5.1). For* $1 \leq k \leq \operatorname{rank}(\mathbf{A})$ *set* $\mathbf{U}_k := [\mathbf{u}_{\cdot,1}, \ldots, \mathbf{u}_{\cdot,k}] \in \mathbb{K}^{m,k}$, $\mathbf{V}_k := [\mathbf{v}_{\cdot,1}, \ldots, \mathbf{v}_{\cdot,k}] \in \mathbb{K}^{n,k}$, $\mathbf{\Sigma}_k := \operatorname{diag}(\sigma_1, \ldots, \sigma_k) \in \mathbb{K}^{k,k}$.
Then, for $\|\cdot\| = \|\cdot\|_F$ *and* $\|\cdot\| = \|\cdot\|_2$, *holds true*

$$\left\|\mathbf{A} - \mathbf{U}_k\Sigma_k\mathbf{V}_k^H\right\| \leq \|\mathbf{A} - \mathbf{F}\| \quad \forall \mathbf{F} \in \mathcal{R}_k(m,n) \ .$$

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 $\mathbf{B} \in \mathbb{K}^{n,n}$, $\operatorname{rank} \mathbf{B} = k$.

$$\blacktriangleright \quad \dim \operatorname{Ker}(\mathbf{B}) = n - k \quad \Rightarrow \quad \operatorname{Ker}(\mathbf{B}) \cap \operatorname{Span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\} \neq \{0\} \ ,$$

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 $\mathbf{Z} = [\mathbf{z}_1 \ldots \mathbf{z}_{n-k}] \in \mathbb{K}^{n,n-k}$

$$\|\mathbf{A} - \mathbf{B}\|_F^2 \geq \|(\mathbf{A} - \mathbf{B})\mathbf{Z}\|_F^2 = \|\mathbf{A}\mathbf{Z}\|_F^2 = \sum_{i=1}^{n-k} \|\mathbf{A}\mathbf{z}_i\|_2^2 = \sum_{i=1}^{n-k}\sum_{j=1}^{r} \sigma_j^2(\mathbf{v}_j^H\mathbf{z}_i)^2 \qquad \square$$

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

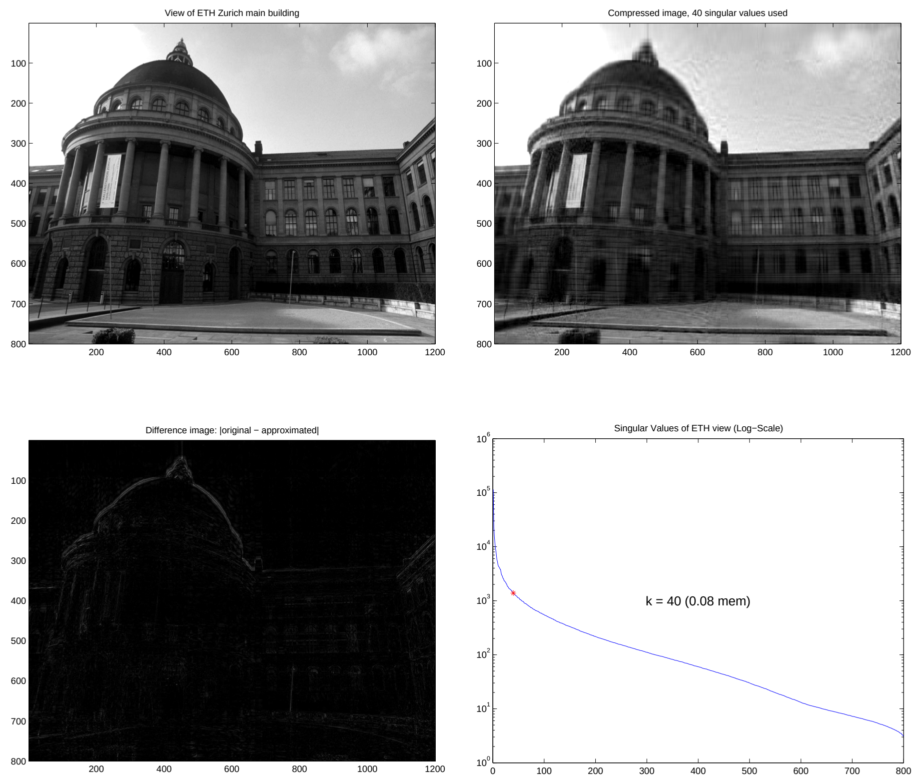
Example 5.5.6 (Image compression).

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. 5.3.9

$\blacktriangleright$ Thm. 5.5.7 ➢ best low rank approximation of image: $\widetilde{\mathbf{A}} = \mathbf{U}_k\Sigma_k\mathbf{V}^T$

Code 5.5.7: 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');
```



However: there are better and faster ways to compress images than SVD (JPEG, Wavelets, etc.)

◇

6 Least Squares

Example 6.0.1 (linear regression).

Given: *measured* data $y_i, \mathbf{x}_i$, $y_i \in \mathbb{R}, \mathbf{x}_i \in \mathbb{R}^n, i = 1,\dots,m, m \geq n+1$
(y_i, $\mathbf{x}_i$ have 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

$$(\mathbf{a}, c) = \underset{\mathbf{p}\in\mathbb{R}^n, q\in\mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{m} |y_i - \mathbf{p}^T\mathbf{x}_i - q|^2 \quad (6.0.1)$$

linear regression for $n = 2, m = 8$ $\triangleright$.

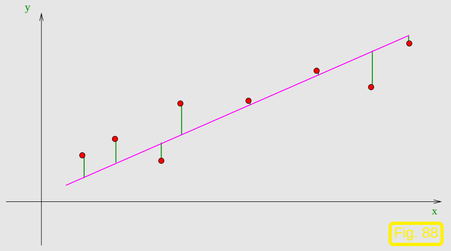


Fig. 88

Remark: In statistics we learn that the least squares estimate provides a maximum likelihood estimate, if the measurement errors are uniformly and independently normally distributed.

Example 6.0.2 (Linear data fitting). ($\to$ Ex. 6.5.1 for a related problem)

Given: "nodes" $(t_i, y_i) \in \mathbb{K}^2$, $i = 1,\dots,m$, $t_i \in I \subset \mathbb{K}$,
basis functions $b_j : I \mapsto \mathbb{K}$, $j = 1,\dots,n$.

Find: coefficients $x_j \in \mathbb{K}$, $j = 1,\dots,n$, such that

$$\sum_{i=1}^{m} |f(t_i) - y_i|^2 \to \min \quad , \quad f(t) := \sum_{j=1}^{n} x_j b_j(t) \, . \qquad (6.0.2)$$

Special case: polynomial fit: $b_j(t) = t^{j-1}$.

MATLAB-function: `p = polyfit(t,y,n);` n = polynomial degree. $\diamond$

Remark 6.0.3 (Overdetermined linear systems).

In Ex. 6.0.1 we could try to find $\mathbf{a}$, c by solving the linear system of equations

$$\begin{pmatrix} \mathbf{x}_1^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^T & 1 \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} ,$$

but in case $m > n+1$ we encounter more equations than unknowns.

In Ex. 6.0.2 the same idea leads to the linear system

$$\begin{pmatrix} b_1(t_1) & \dots & b_n(t_1) \\ \vdots & & \vdots \\ b_1(t_m) & \dots & b_n(t_m) \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} ,$$

with the same problem in case $m > n$. $\triangle$

(Linear) least squares problem:

given: $\mathbf{A} \in \mathbb{K}^{m,n}$, $m, n \in \mathbb{N}$, $\mathbf{b} \in \mathbb{K}^m$,
find: $\mathbf{x} \in \mathbb{K}^n$ such that

$$\text{(i) } \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \inf\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbb{K}^n\}, \qquad (6.0.3)$$

(ii) $\|\mathbf{x}\|_2$ is minimal under the condition (i).

Recast as linear least squares problem, *cf.* Rem. 6.0.3:

Ex. 6.0.1: $\mathbf{A} = \begin{pmatrix} \mathbf{x}_1^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^T & 1 \end{pmatrix} \in \mathbb{R}^{m,n+1}$, $\mathbf{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \in \mathbb{R}^n$, $\mathbf{x} = \begin{pmatrix} \mathbf{a} \\ c \end{pmatrix} \in \mathbb{R}^{n+1}$.

Ex. 6.0.2: $\mathbf{A} = \begin{pmatrix} b_1(t_1) & \dots & b_n(t_1) \\ \vdots & & \vdots \\ b_1(t_m) & \dots & b_n(t_m) \end{pmatrix} \in \mathbb{R}^{m,n}$, $\mathbf{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \in \mathbb{R}^m$, $\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$.

In both cases the residual norm $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2$ allows to gauge the quality of the model.

Lemma 6.0.1 (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 (6.2.5) and its derivation, see Sect. 6.2. $\square$

MATLAB "black-box" solver for linear least squares problems:

`x = A\b` ("backslash") solves (6.0.3) for $\mathbf{A} \in \mathbb{K}^{m,n}$, $m \neq n$.

Reassuring: stable ($\rightarrow$ Def.2.5.5) implementation (for dense matrices).

Remark 6.0.4 (Pseudoinverse).

By Lemma 6.0.1 the solution operator of the least squares problem (6.0.3) defines a linear mapping $\mathbf{b} \mapsto \mathbf{x}$, which has a matrix representation.

Definition 6.0.2 (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* (6.0.3) $\|\mathbf{A}\mathbf{x} - \mathbf{b}\| \rightarrow \min$, $\|\mathbf{x}\| \rightarrow \min$.

MATLAB: `P = pinv(A)` computes the pseudoinverse.

$\triangle$

Remark 6.0.5 (Conditioning of the least squares problem).

Definition 6.0.3 (Generalized condition (number) of a matrix, $\rightarrow$ Def. 2.5.11).
Let $\sigma_1 \geq \sigma_2 \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_p = 0$, $p := \min\{m, n\}$, *be the singular values* ($\rightarrow$ *Def. 5.5.2) 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 6.0.4. *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* $\|\mathbf{A}\mathbf{x} - \mathbf{b}\| \rightarrow \min$ *and* $\widehat{\mathbf{x}}$ *the solution of the perturbed least squares problem* $\|(\mathbf{A} + \Delta\mathbf{A})\widehat{\mathbf{x}} - \mathbf{b}\| \rightarrow \min$. *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 $\approx \operatorname{cond}_2(\mathbf{A})$
if $\|\mathbf{r}\|_2$ "large" ➤ condition of the least squares problem $\approx \operatorname{cond}_2^2(\mathbf{A})$

6.1 Normal Equations

Setting: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, with full rank $\operatorname{rank}(\mathbf{A}) = n$.

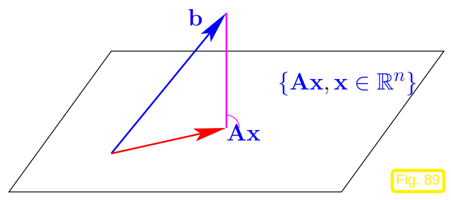


Fig. 89

Geometric interpretation of linear least squares problem (6.0.3):

$\mathbf{x} \mathrel{\hat{=}}$ orthogonal projection of $\mathbf{b}$ on the subspace $\operatorname{Im}(\mathbf{A}) := \operatorname{Span}\left\{(\mathbf{A})_{:,1}, \ldots, (\mathbf{A})_{:,n}\right\}$.

Geometric interpretation: the least squares problem (6.0.3) amounts to searching the point $\mathbf{p} \in \operatorname{Im}(\mathbf{A})$ nearest (w.r.t. Euclidean distance) to $\mathbf{b} \in \mathbb{R}^m$.

Geometric intuition, see Fig. 89: $\mathbf{p}$ is the orthogonal projection of $\mathbf{b}$ onto $\operatorname{Im}(\mathbf{A})$, that is $\mathbf{b} - \mathbf{p} \perp \operatorname{Im}(\mathbf{A})$. Note the equivalence

$$\mathbf{b} - \mathbf{p} \perp \operatorname{Im}(\mathbf{A}) \quad \Leftrightarrow \quad \mathbf{b} - \mathbf{p} \perp (\mathbf{A})_{:,j}\,, \quad j = 1, \ldots, n \quad \Leftrightarrow \quad \mathbf{A}^H(\mathbf{b} - \mathbf{p}) = 0\,,$$

Representation $\mathbf{p} = \mathbf{A}\mathbf{x}$ leads to normal equations (6.1.2).

Solve (6.0.3) for $\mathbf{b} \in \mathbb{R}^m$

$$\mathbf{x} \in \mathbb{R}^n\colon \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \rightarrow \min \quad \Leftrightarrow \quad f(\mathbf{x}) := \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \rightarrow \min\,. \tag{6.1.1}$$

A quadratic functional, *cf.* (4.1.1)

$$f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 = \mathbf{x}^H(\mathbf{A}^H\mathbf{A})\mathbf{x} - 2\mathbf{b}^H\mathbf{A}\mathbf{x} + \mathbf{b}^H\mathbf{b}\,.$$

Minimization problem for f ➢ study gradient, *cf.* (4.1.4)

$$\mathbf{grad}\, f(\mathbf{x}) = 2(\mathbf{A}^H\mathbf{A})\mathbf{x} - 2\mathbf{A}^H\mathbf{b}\,.$$

$$\mathbf{grad}\, f(\mathbf{x}) \overset{!}{=} 0\colon \quad \boxed{\mathbf{A}^H\mathbf{A}\mathbf{x} = \mathbf{A}^H\mathbf{b}} \quad = \text{normal equation of (6.1.1)} \tag{6.1.2}$$

Notice: $\operatorname{rank}(\mathbf{A}) = n \quad \Rightarrow \quad \mathbf{A}^H\mathbf{A} \in \mathbb{R}^{n,n}$ s.p.d. ($\rightarrow$ Def. 2.7.1)

Remark 6.1.1 (Conditioning of normal equations).

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 (6.1.2) to solve the linear least squares problem (6.1.1) 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. 2.5.5) of normal equation approach.

△

Example 6.1.2 (Instability of normal equations).

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 $\delta < \sqrt{\text{eps}} \Rightarrow 1+\delta^2 = 1$ 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. 2.4.9.

◇

Another *reason not to compute* $\mathbf{A}^H\mathbf{A}$, when both m, n large:

$$\boxed{\mathbf{A} \text{ sparse} \;\not\Rightarrow\; \mathbf{A}^T\mathbf{A} \text{ sparse}}$$

▶
- Potential memory overflow, when computing $\mathbf{A}^T\mathbf{A}$
- Squanders possibility to use efficient sparse direct elimination techniques, see Sect. 2.6.3

A way to avoid the computation of $\mathbf{A}^H\mathbf{A}$:

Expand normal equations (6.1.2): introduce residual $\mathbf{r} := \mathbf{A}\mathbf{x} - \mathbf{b}$ as new unknown:

$$\mathbf{A}^H\mathbf{A}\mathbf{x} = \mathbf{A}^H\mathbf{b} \quad \Leftrightarrow \quad \mathbf{B}\begin{pmatrix}\mathbf{r}\\ \mathbf{x}\end{pmatrix} := \begin{pmatrix} -\mathbf{I} & \mathbf{A} \\ \mathbf{A}^H & 0 \end{pmatrix}\begin{pmatrix}\mathbf{r}\\ \mathbf{x}\end{pmatrix} = \begin{pmatrix}\mathbf{b}\\ 0\end{pmatrix} \,. \tag{6.1.3}$$

More general substitution $\mathbf{r} := \alpha^{-1}(\mathbf{A}\mathbf{x} - \mathbf{b})$, $\alpha > 0$ to improve the condition:

$$\mathbf{A}^H\mathbf{A}\mathbf{x} = \mathbf{A}^H\mathbf{b} \quad \Leftrightarrow \quad \mathbf{B}_\alpha\begin{pmatrix}\mathbf{r}\\ \mathbf{x}\end{pmatrix} := \begin{pmatrix} -\alpha\mathbf{I} & \mathbf{A} \\ \mathbf{A}^H & 0 \end{pmatrix}\begin{pmatrix}\mathbf{r}\\ \mathbf{x}\end{pmatrix} = \begin{pmatrix}\mathbf{b}\\ 0\end{pmatrix} \,. \tag{6.1.4}$$

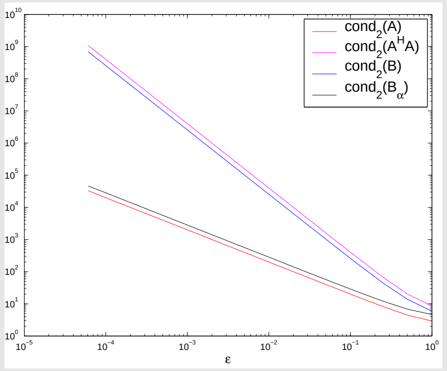
For $m, n \gg 1$, $\mathbf{A}$ sparse, both (6.1.3) and (6.1.4) lead to large sparse linear systems of equations, amenable to sparse direct elimination techniques, see Sect. 2.6.3

Example 6.1.3 (Condition of the extended system).

Consider (6.1.3), (6.1.4) for

$$\mathbf{A} = \begin{pmatrix} 1+\epsilon & 1 \\ 1-\epsilon & 1 \\ \epsilon & \epsilon \end{pmatrix} \,.$$

Plot of different condition numbers in dependence on ϵ
($\alpha = \|\mathbf{A}\|_2/\sqrt{2}$) ▷



◇

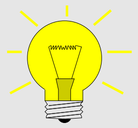
6.2 Orthogonal Transformation Methods

Consider the linear least squares problem (6.0.3)

$$\text{given} \quad \mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{b} \in \mathbb{R}^m \quad \text{find} \quad \mathbf{x} = \operatorname*{argmin}_{\mathbf{y} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 .$$

Assumption: $m \geq n$ and $\mathbf{A}$ has full (maximum) rank: $\operatorname{rank}(\mathbf{A}) = n$.

Recall Thm. 2.8.2: orthogonal (unitary) transformations ($\rightarrow$ Def. 2.8.1) leave 2-norm invariant.



Idea: Transformation of $\mathbf{A}\mathbf{x} - \mathbf{b}$ to simpler form by *orthogonal* row transformations:

$$\operatorname*{argmin}_{\mathbf{y} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 = \operatorname*{argmin}_{\mathbf{y} \in \mathbb{R}^n} \left\|\widetilde{\mathbf{A}}\mathbf{y} - \widetilde{\mathbf{b}}\right\|_2 ,$$

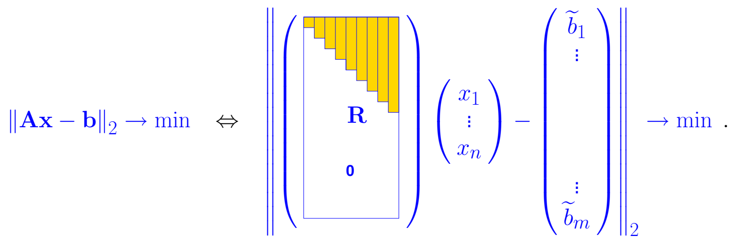
where $\widetilde{\mathbf{A}} = \mathbf{Q}\mathbf{A}$, $\widetilde{\mathbf{b}} = \mathbf{Q}\mathbf{b}$ with orthogonal $\mathbf{Q} \in \mathbb{R}^{m,m}$.

As in the case of LSE ($\rightarrow$ Sect. 2.8): "simpler form" = triangular form.

Concrete realization of this idea by means of QR-decomposition ($\rightarrow$ Section 2.8).

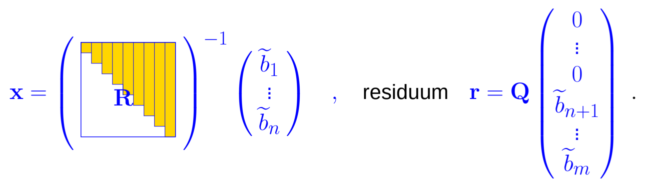
QR-decomposition: $\mathbf{A} = \mathbf{Q}\mathbf{R}$, $\mathbf{Q} \in \mathbb{K}^{m,m}$ unitary, $\mathbf{R} \in \mathbb{K}^{m,n}$ (regular) upper triangular matrix.

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \left\|\mathbf{Q}(\mathbf{R}\mathbf{x} - \mathbf{Q}^H\mathbf{b})\right\|_2 = \left\|\mathbf{R}\mathbf{x} - \widetilde{\mathbf{b}}\right\|_2 , \quad \widetilde{\mathbf{b}} := \mathbf{Q}^H\mathbf{b} .$$



$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \rightarrow \min \quad \Leftrightarrow$
R
0
x_1
x_n
$\widetilde{b}_1$
$\widetilde{b}_m$
$\rightarrow \min$.

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 $\mathbf{x}$.



Note: 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 (2.8.2) for general matrices, and Givens rotations (2.8.3) 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 $(\mathbf{R}, \widetilde{\mathbf{b}})$

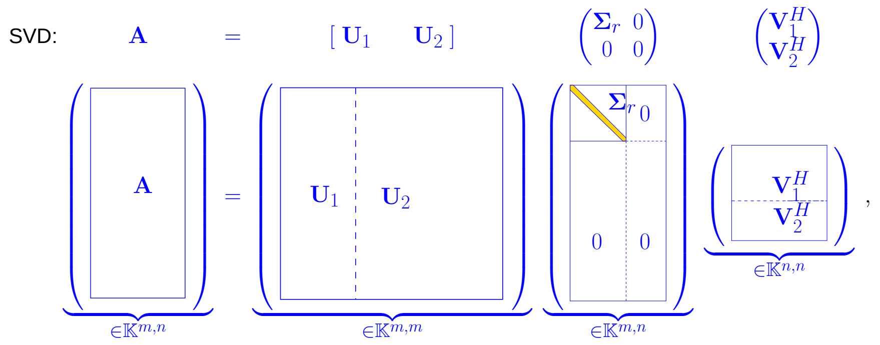
$\mathbf{Q}$ need not be stored !

➤ A QR-based algorithm is implemented in the least-squares-solver of the MATLAB-operator "\" (for dense matrices).

Alternative: Solving linear least squares problem (6.0.3) by SVD

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 (6.0.3) of a linear least squares problem may be required for singling out a unique solution.



SVD: $\mathbf{A} = [\,\mathbf{U}_1 \quad \mathbf{U}_2\,] \begin{pmatrix} \mathbf{\Sigma}_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{pmatrix}$
A
=
$\mathbf{U}_1$
$\mathbf{U}_2$
$\mathbf{\Sigma}_r$
0
0
0
$\mathbf{V}_1^H$
$\mathbf{V}_2^H$
$\in \mathbb{K}^{m,n}$
$\in \mathbb{K}^{m,m}$
$\in \mathbb{K}^{m,n}$
$\in \mathbb{K}^{n,n}$

(6.2.1)

with $\mathbf{U}_1 \in \mathbb{K}^{m,r}$, $\mathbf{U}_2 \in \mathbb{K}^{m,m-r}$, $\mathbf{\Sigma}_r = \operatorname{diag}(\sigma_1, \ldots, \sigma_r) \in \mathbb{R}^{r,r}$, $\mathbf{V}_1 \in \mathbb{K}^{n,r}$, $\mathbf{V}_2 \in \mathbb{K}^{n,n-r}$.

Note that by definition of the SVD, Def. 5.5.2, the columns of $\mathbf{U}_1, \mathbf{U}_2, \mathbf{V}_1, \mathbf{V}_2$ are *orthonormal*.

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. 2.8.2, together with the fact that $\mathbf{U}$ is unitary, see Def. 2.8.1:

$$[\mathbf{U}_1, \mathbf{u}_2] \cdot \left[\begin{pmatrix} \mathbf{U}_1^H \\ \mathbf{U}_2^H \end{pmatrix}\right] = \mathbf{I} \; .$$

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \left\| [\mathbf{U}_1 \; \mathbf{U}_2] \begin{pmatrix} \boldsymbol{\Sigma}_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{pmatrix} \mathbf{x} - \mathbf{b} \right\|_2 = \left\| \begin{pmatrix} \boldsymbol{\Sigma}_r \mathbf{V}_1^H \mathbf{x} \\ 0 \end{pmatrix} - \begin{pmatrix} \mathbf{U}_1^H \mathbf{b} \\ \mathbf{U}_2^H \mathbf{b} \end{pmatrix} \right\|_2 \qquad (6.2.2)$$

Logical strategy: choose $\mathbf{x}$ such that the first r components of $\begin{pmatrix} \boldsymbol{\Sigma}_r \mathbf{V}_1^H \mathbf{x} \\ 0 \end{pmatrix} - \begin{pmatrix} \mathbf{U}_1^H \mathbf{b} \\ \mathbf{U}_2^H \mathbf{b} \end{pmatrix}$ vanish:

➣ (possibly *underdetermined*) $r \times n$ linear system $\quad \boldsymbol{\Sigma}_r \mathbf{V}_1^H \mathbf{x} = \mathbf{U}_1^H \mathbf{b} \; . \qquad (6.2.3)$

To fix a unique solution in the case $r < n$ we appeal to the minimal norm condition in (6.0.3): solution $\mathbf{x}$ of (6.2.3) is unique up to contributions from

$$\mathrm{Ker}(\mathbf{V}_1^H) = \mathrm{Im}(\mathbf{V}_1)^\perp = \mathrm{Im}(\mathbf{V}_2) \; . \qquad (6.2.4)$$

Since $\mathbf{V}$ is unitary, the minimal norm solution is obtained by setting contributions from $\mathrm{Im}(\mathbf{V}_2)$ to zero, which amounts to choosing $\mathbf{x} \in \mathrm{Im}(\mathbf{V}_1)$. This converts (6.2.3) into

$$\boldsymbol{\Sigma}_r \underbrace{\mathbf{V}_1^H \mathbf{V}_1}_{=\mathbf{I}} \mathbf{z} = \mathbf{U}_1^H \mathbf{b} \quad \Rightarrow \quad \mathbf{z} = \boldsymbol{\Sigma}_r^{-1} \mathbf{U}_1^H \mathbf{b} \; .$$

► solution $\boxed{\mathbf{x} = \mathbf{V}_1 \boldsymbol{\Sigma}_r^{-1} \mathbf{U}_1^H \mathbf{b}} \; , \quad \|\mathbf{r}\|_2 = \left\| \mathbf{U}_2^H \mathbf{b} \right\|_2 \; . \qquad (6.2.5)$

Practical implementation:

"numerical rank" test:

$r = \max\{i \colon \sigma_i/\sigma_1 > \mathtt{tol}\}$

Code 6.2.1: Solving LSQ problem via SVD

```
function y = lsqsvd(A,b)
[U,S,V] = svd(A,0);
sv = diag(S);
r = max(find(sv/sv(1) > eps));
y = V(:,1:r)*(diag(1./sv(1:r))*...
          (U(:,1:r)'*b));
```

Remark 6.2.2 (Pseudoinverse and SVD). $\quad\rightarrow$ Rem. 6.0.4

The solution formula (6.2.5) directly yields a representation of the pseudoinverse $\mathbf{A}^+$ ($\rightarrow$ Def. 6.0.2) of any matrix $\mathbf{A}$:

Theorem 6.2.1 (Pseudoinverse and SVD).
If $\mathbf{A} \in \mathbb{K}^{m,n}$ *has the SVD decomposition* (6.2.1), *then* $\quad \mathbf{A}^+ = \mathbf{V}_1 \boldsymbol{\Sigma}_r^{-1} \mathbf{U}_1^H \quad$ *holds.*

$\triangle$

Remark 6.2.3 (Normal equations vs. orthogonal transformations method).

Superior numerical stability ($\rightarrow$ Def. 2.5.5) of orthogonal transformations methods:

► Use orthogonal transformations methods for least squares problems (6.0.3), 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 (6.1.3)/(6.1.4), when $\mathbf{A} \in \mathbb{R}^{m,n}$ sparse ($\rightarrow$ Def. 2.6.1) and m, n big.

$\triangle$

Example 6.2.4 (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:

$$\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^d \colon c + \mathbf{n}^T \mathbf{x} = 0\} \; , \quad \|\mathbf{n}\|_2 = 1 \; . \qquad (6.2.6)$$

► Euclidean distance of $\mathbf{y}\in\mathbb{R}^d$ from the plane: $\mathrm{dist}(\mathcal{H},\mathbf{y}) = |c+\mathbf{n}^T\mathbf{y}|$. (6.2.7)

Goal: given the points $\mathbf{y}_1,\dots,\mathbf{y}_m$, $m>d$, find $\mathcal{H}\leftrightarrow\{c\in\mathbb{R},\mathbf{n}\in\mathbb{R}^d, \|\mathbf{n}\|_2=1\}$, such that

$$\sum_{j=1}^{m}\mathrm{dist}(\mathcal{H},\mathbf{y}_j)^2=\sum_{j=1}^{m}|c+\mathbf{n}^T\mathbf{y}_j|^2\to\min\ . \qquad (6.2.8)$$

Note: (6.2.8) $\neq$ linear least squares problem due to constraint $\|\mathbf{n}\|_2=1$.

$$(6.2.8)\ \Leftrightarrow\ \left\|\underbrace{\begin{pmatrix}1 & y_{1,1} & \cdots & y_{1,d}\\ 1 & y_{2,1} & \cdots & y_{2,d}\\ \vdots & \vdots & & \vdots\\ 1 & y_{m,1} & \cdots & y_{m,d}\end{pmatrix}}_{=:\mathbf{A}}\begin{pmatrix}c\\ n_1\\ \vdots\\ n_d\end{pmatrix}\right\|_2\to\min \quad \text{under constraint} \quad \|\mathbf{n}\|_2=1\ .$$

Step ❶: QR-decomposition (→ Section 2.8)

$$\mathbf{A}:=\begin{pmatrix}1 & y_{1,1} & \cdots & y_{1,d}\\ 1 & y_{2,1} & \cdots & y_{2,d}\\ \vdots & \vdots & & \vdots\\ 1 & y_{m,1} & \cdots & y_{m,d}\end{pmatrix}=\mathbf{QR}\quad,\quad \mathbf{R}:=\begin{pmatrix}r_{11} & r_{12} & \cdots & \cdots & r_{1,d+1}\\ 0 & r_{22} & \cdots & \cdots & r_{2,d+1}\\ \vdots & & \ddots & & \vdots\\ 0 & & & & r_{d+1,d+1}\\ 0 & \cdots & & \cdots & 0\\ \vdots & & & & \vdots\\ 0 & \cdots & & \cdots & 0\end{pmatrix}\in\mathbb{R}^{m,d+1}\ .$$

$$\|\mathbf{Ax}\|_2\to\min\ \Leftrightarrow\ \|\mathbf{Rx}\|_2=\left\|\begin{pmatrix}r_{11} & r_{12} & \cdots & \cdots & r_{1,d+1}\\ 0 & r_{22} & \cdots & \cdots & r_{2,d+1}\\ \vdots & & \ddots & & \vdots\\ 0 & & & & r_{d+1,d+1}\\ 0 & \cdots & & \cdots & 0\\ \vdots & & & & \vdots\\ 0 & \cdots & & \cdots & 0\end{pmatrix}\begin{pmatrix}c\\ n_1\\ \vdots\\ \vdots\\ n_d\end{pmatrix}\right\|_2\to\min\ . \qquad (6.2.9)$$

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 (6.2.9) to

$$\left\|\begin{pmatrix}r_{22} & r_{23} & \cdots & \cdots & r_{2,d+1}\\ 0 & r_{33} & \cdots & \cdots & r_{3,d+1}\\ \vdots & & \ddots & & \vdots\\ 0 & & & & r_{d+1,d+1}\end{pmatrix}\begin{pmatrix}n_1\\ \vdots\\ \vdots\\ n_d\end{pmatrix}\right\|_2\to\min\quad,\quad\|\mathbf{n}\|_2=1\ . \qquad (6.2.10)$$

(6.2.10) **=** problem of type (5.5.5), minimization on the Euclidean sphere.

➢ Solve (6.2.10) using SVD !

Note: Since $r_{11}=\|(\mathbf{A})_{:,1}\|_2=\sqrt{d+1}\neq0\ \Rightarrow\ c=-r_{11}^{-1}\sum_{j=1}^{d}r_{1,j+1}n_j$.

MATLAB-function:

For $\mathbf{A}\in\mathbb{K}^{m,n}$ find $\mathbf{n}\in\mathbb{R}^d$, $\mathbf{c}\in\mathbb{R}^{n-d}$ such that

$$\left\|\mathbf{A}\begin{pmatrix}\mathbf{c}\\ \mathbf{n}\end{pmatrix}\right\|_2\to\min$$

with the constraint:

$$\|\mathbf{n}\|_2=1\ .$$

Code 6.2.5: (Generalized) distance fitting a hyperplane

```
function [c,n] = clsq(A,dim);
[m,p] = size(A);
if p < dim+1, error ('not␣enough␣unknowns'); end;
if m < dim, error ('not␣enough␣equations'); end;
m = min (m, p);
R = triu (qr (A));
[U,S,V] = svd(R(p-dim+1:m,p-dim+1:p));
n = V(:,dim);
c = -R(1:p-dim,1:p-dim)\R(1:p-dim,p-dim+1:p)*n;
```

6.3 Total Least Squares

Given: overdetermined linear system of equations $\mathbf{Ax}=\mathbf{b}$, $\mathbf{A}\in\mathbb{R}^{m,n}$, $\mathbf{b}\in\mathbb{R}^m$, $m\geq n$.

Known: LSE solvable $\Leftrightarrow$ $\mathbf{b}\in\mathrm{Im}(\mathbf{A})$, *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!

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

$$\Big\|\underbrace{[\mathbf{A}\ \ \mathbf{b}]}_{=:\mathbf{C}}-\underbrace{[\widehat{\mathbf{A}}\ \ \widehat{\mathbf{b}}]}_{=:\widehat{\mathbf{C}}}\Big\|_F\to\min\quad,\quad\widehat{\mathbf{b}}\in\mathrm{Im}(\widehat{\mathbf{A}})\ . \qquad (6.3.1)$$

$$\widehat{\mathbf{b}} \in \operatorname{Im}(\widehat{\mathbf{A}}) \quad\Rightarrow\quad \operatorname{rank}(\widehat{\mathbf{C}}) = n \qquad ▶ \qquad \text{(6.3.1)} \;\Rightarrow\; \min_{\operatorname{rank}(\widehat{\mathbf{C}})=n} \left\| \mathbf{C} - \widehat{\mathbf{C}} \right\|_F .$$

Thm. 5.5.7 ➤ use the SVD decomposition of $\mathbf{C}$ to construct $\widehat{\mathbf{C}}$:

$$\mathbf{C} = \mathbf{U}\Sigma\mathbf{V}^H = \sum_{j=1}^{n+1} \sigma_j (\mathbf{U})_{:,j} (\mathbf{V})_{:,j}^H$$

$$\widehat{\mathbf{C}} = \sum_{j=1}^{n} \sigma_j (\mathbf{U})_{:,j} (\mathbf{V})_{:,j}^H \quad\Rightarrow\quad \widehat{\mathbf{C}}(\mathbf{V})_{:,n+1} = 0 .$$

If $(\mathbf{V})_{n+1,n+1} \neq 0$, then

$$\widehat{\mathbf{A}}\mathbf{x} = \widehat{\mathbf{b}} \quad \text{with} \quad \mathbf{x} = (\mathbf{v})_{n+1,n+1}^{-1} (\mathbf{V})_{\cdot,n+1} .$$

Code 6.3.2: Total least squares via SVD

```
function x = lsqtotal(A,b);
[m,n]=size(A);
[U, Sigma, V] = svd([A,b]);
s = V(n+1,n+1);
if s == 0 ,
  error('No␣solution')
end
x = -V(1:n,n+1)/s;
```

6.4 Constrained Least Squares

Given: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, $\operatorname{rank}(\mathbf{A}) = n$, $\mathbf{b} \in \mathbb{R}^m$,
$\mathbf{C} \in \mathbb{R}^{p,n}$, $p < n$, $\operatorname{rank}(\mathbf{C}) = p$, $\mathbf{d} \in \mathbb{R}^p$

Find: $\mathbf{x} \in \mathbb{R}^n$ with

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \to \min \quad , \quad \boxed{\mathbf{C}\mathbf{x} = \mathbf{d}} . \tag{6.4.1}$$

Linear constraint

Solution via normal equations

Idea: coupling the constraint using the Lagrange multiplier $\mathbf{m} \in \mathbb{R}^p$

$$\mathbf{x} = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \max_{\mathbf{m}\in\mathbb{R}^p} L(\mathbf{x},\mathbf{m}) \;,\quad L(\mathbf{x},\mathbf{m}) := \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \mathbf{m}^H(\mathbf{C}\mathbf{x} - \mathbf{d}) .$$

Necessary (and sufficient) condition for the solution (→ Section 6.1)

$$\frac{\partial L}{\partial \mathbf{x}}(\mathbf{x},\mathbf{m}) = \mathbf{A}^H(\mathbf{A}\mathbf{x} - \mathbf{b}) + \mathbf{C}^H\mathbf{m} \overset{!}{=} 0 \quad , \quad \frac{\partial L}{\partial \mathbf{m}}(\mathbf{x},\mathbf{m}) = \mathbf{C}\mathbf{x} - \mathbf{d} \overset{!}{=} 0.$$

▼

$$\begin{pmatrix} \mathbf{A}^H\mathbf{A} & \mathbf{C}^H \\ \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{m} \end{pmatrix} = \begin{pmatrix} \mathbf{A}^H\mathbf{b} \\ \mathbf{d} \end{pmatrix}$$ Extended normal equations (saddle point problem)

Algorithm (based on block-LU-decomposition):

$$\begin{pmatrix} \mathbf{A}^H\mathbf{A} & \mathbf{C}^H \\ \mathbf{C} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{R}^H & 0 \\ \mathbf{G} & -\mathbf{S}^H \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{G}^H \\ 0 & \mathbf{S} \end{pmatrix} \quad , \quad \begin{array}{l} \mathbf{R},\mathbf{S} \in \mathbb{R}^{n,n} \text{ upper triangular matrix,} \\ \mathbf{G} \in \mathbb{R}^{p,n} . \end{array}$$

$\mathbf{R}$ from $\mathbf{R}^H\mathbf{R} = \mathbf{A}^H\mathbf{A}$ → Cholesky decomposition → Sect. 2.7,
$\mathbf{G}$ from $\mathbf{R}^H\mathbf{G}^H = \mathbf{C}^H$ → n forward substitution → Sect. 2.2,
$\mathbf{S}$ from $\mathbf{S}^H\mathbf{S} = \mathbf{G}\mathbf{G}^H$ → Cholesky decomposition → Sect. 2.7.

Caution Sect. 6.1: 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. (6.1.3))

$$\begin{pmatrix} -\mathbf{I} & \mathbf{A} & 0 \\ \mathbf{A}^H & 0 & \mathbf{C}^H \\ 0 & \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \\ \mathbf{m} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \\ \mathbf{d} \end{pmatrix} . \tag{6.4.2}$$

Solution via SVD:

① Compute orthonormal basis of $\operatorname{Ker}(\mathbf{C})$ using SVD (→ Section 6.2):

$$\mathbf{C} = \mathbf{U}\,[\Sigma\ 0] \begin{bmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{bmatrix} \;,\quad \mathbf{U} \in \mathbb{R}^{p,p},\ \Sigma \in \mathbb{R}^{p,p},\ \mathbf{V}_1 \in \mathbb{R}^{n,p},\ \mathbf{V}_2 \in \mathbb{R}^{n,n-p}$$

▶ $\operatorname{Ker}(\mathbf{C}) = \operatorname{Im}(\mathbf{V}_2)$.

and the particular solution

$$\mathbf{x}_0 := \mathbf{V}_1\Sigma^{-1}\mathbf{U}^H\mathbf{d} .$$

Representation of the solution $\mathbf{x}$ of (6.4.1): $\mathbf{x} = \mathbf{x}_0 + \mathbf{V}_2\mathbf{y}$, $\mathbf{y} \in \mathbb{R}^{n-p}$.

② Insert this representation in (6.4.1) ➤ standard linear least squares

$$\|\mathbf{A}(\mathbf{x}_0 + \mathbf{V}_2\mathbf{y}) - \mathbf{b}\|_2 \to \min \quad\Leftrightarrow\quad \|\mathbf{A}\mathbf{V}_2\mathbf{y} - (\mathbf{b} - \mathbf{A}\mathbf{x}_0)\| \to \min .$$

Exercise 6.4.1. Given a regular tridiagonal matrix $\mathbf{T} \in \mathbb{R}^{n,n}$, develop an algorithm for solving the linear least squares problem

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \;,$$

where

$$\mathbf{A} = \begin{pmatrix} \mathbf{T}^{-1} \\ \vdots \\ \mathbf{T}^{-1} \end{pmatrix} \in \mathbb{R}^{pn,n} .$$

6.5 Non-linear Least Squares

Example 6.5.1 (Non-linear data fitting (parametric statistics)).

Given: data points (t_i, y_i), $i = 1, \dots, m$ with measurements errors.

Known: $y = f(t, \mathbf{x})$ through a function $f : \mathbb{R} \times \mathbb{R}^n \mapsto \mathbb{R}$ 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:

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} |f(t_i, \mathbf{x}) - y_i|^2 = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \tfrac{1}{2} \|F(\mathbf{x})\|_2^2 \,, \tag{6.5.1}$$

$$\text{with} \quad F(\mathbf{x}) = \begin{pmatrix} f(t_1, \mathbf{x}) - y_1 \\ \vdots \\ f(t_m, \mathbf{x}) - y_m \end{pmatrix} .$$

◇

Non-linear least squares problem

Given: $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^m, \quad m, n \in \mathbb{N}, \; m > n.$

Find: $\mathbf{x}^* \in D$: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in D} \Phi(\mathbf{x}) \,, \quad \Phi(\mathbf{x}) := \tfrac{1}{2}\|F(\mathbf{x})\|_2^2 \,.$ (6.5.2)

Terminology: $D \hat{=}$ parameter space, $x_1, \dots, x_n \hat{=}$ parameter.

As in the case of linear least squares problems (→ Rem. 6.0.3): 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 (→ Chapter 3): existence and uniqueness of $\mathbf{x}^*$ in (6.5.2) has to be established in each concrete case!

We require "independence for each parameter":

$\exists$ neighbourhood $\mathcal{U}(\mathbf{x}^*)$such that $DF(\mathbf{x})$ has full rank $n \quad \forall\, \mathbf{x} \in \mathcal{U}(\mathbf{x}^*)$. (6.5.3)

(It means: the columns of the Jacobi matrix $DF(\mathbf{x})$ are linearly independent.)

If (6.5.3) is not satisfied, then the parameters are redundant in the sense that fewer parameters would be enough to model the same dependence (locally at $\mathbf{x}^*$).

6.5.1 (Damped) Newton method

$$\Phi(\mathbf{x}^*) = \min \;\Rightarrow\; \mathbf{grad}\,\Phi(\mathbf{x}) = 0, \quad \mathbf{grad}\,\Phi(\mathbf{x}) := \left(\frac{\partial\Phi}{\partial x_1}(\mathbf{x}), \dots, \frac{\partial\Phi}{\partial x_n}(\mathbf{x})\right)^T \in \mathbb{R}^n.$$

Simple idea: use Newton's method (→ Sect. 3.4) to determine a zero of $\mathbf{grad}\,\Phi : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$.

Newton iteration (3.4.1) for non-linear system of equations $\mathbf{grad}\,\Phi(\mathbf{x}) = 0$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - H\Phi(\mathbf{x}^{(k)})^{-1}\, \mathbf{grad}\,\Phi(\mathbf{x}^{(k)}) \,, \quad (H\Phi(\mathbf{x}) = \text{Hessian matrix}) \,. \tag{6.5.4}$$

Expressed in terms of $F : \mathbb{R}^n \mapsto \mathbb{R}^n$ from (6.5.2):

chain rule (3.4.2) ➤ $\mathbf{grad}\,\Phi(\mathbf{x}) = DF(\mathbf{x})^T F(\mathbf{x})$,

product rule (3.4.3) ➤ $H\Phi(\mathbf{x}) := D(\mathbf{grad}\,\Phi)(\mathbf{x}) = DF(\mathbf{x})^T DF(\mathbf{x}) + \sum_{j=1}^{m} F_j(\mathbf{x}) D^2 F_j(\mathbf{x})$,

$$\Updownarrow$$

$$(H\Phi(\mathbf{x}))_{i,k} = \sum_{j=1}^{n} \frac{\partial^2 F_j}{\partial x_i \partial x_k}(\mathbf{x}) F_j(\mathbf{x}) + \frac{\partial F_j}{\partial x_k}(\mathbf{x}) \frac{\partial F_j}{\partial x_i}(\mathbf{x}) \,.$$

This allows to rewrite (6.5.4) in concrete terms:

➤ For Newton iterate $\mathbf{x}^{(k)}$: Newton correction $\mathbf{s} \in \mathbb{R}^n$ from LSE

$$\underbrace{\left(DF(\mathbf{x}^{(k)})^T DF(\mathbf{x}^{(k)}) + \sum_{j=1}^{m} F_j(\mathbf{x}^{(k)}) D^2 F_j(\mathbf{x}^{(k)})\right)}_{=H\Phi(\mathbf{x}^{(k)})} \mathbf{s} = -\underbrace{DF(\mathbf{x}^{(k)})^T F(\mathbf{x}^{(k)})}_{=\mathbf{grad}\,\Phi(\mathbf{x}^{(k)})} \,. \tag{6.5.5}$$

Remark 6.5.2 (Newton method and minimization of quadratic functional).

Newton's method (6.5.4) for (6.5.2) can be read as *successive minimization* of a local quadratic approximation Φ

$$\Phi(\mathbf{x}) \approx Q(\mathbf{s}) := \Phi(\mathbf{x}^{(k)}) + \mathbf{grad}\,\Phi(\mathbf{x}^{(k)})^T\mathbf{s} + \tfrac{1}{2}\mathbf{s}^T H\Phi(\mathbf{x}^{(k)})\mathbf{s}\,, \qquad (6.5.6)$$

$$\mathbf{grad}\,Q(\mathbf{s}) = 0 \quad\Leftrightarrow\quad H\Phi(\mathbf{x}^{(k)})\mathbf{s} + \mathbf{grad}\,\Phi(\mathbf{x}^{(k)}) = 0 \quad\Leftrightarrow\quad \text{(6.5.5)}\,.$$

➢ Another model function method (→ Sect. 3.3.2) with quadratic model function for Q.

6.5.2 Gauss-Newton method

Idea: local linearization of F: $F(x) \approx F(y) + DF(\mathbf{y})(\mathbf{x}-\mathbf{y})$

➢ sequence of *linear* least squares problems

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \|F(\mathbf{x})\|_2 \text{ approximated by } \underbrace{\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \|F(\mathbf{x}_0) + DF(\mathbf{x}_0)(\mathbf{x}-\mathbf{x}_0)\|_2}_{(\spadesuit)}\,,$$

where $\mathbf{x}_0$ is an approximation of the solution $\mathbf{x}^*$ of (6.5.2).

$$(\spadesuit) \quad\Leftrightarrow\quad \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \|\mathbf{A}\mathbf{x}-\mathbf{b}\| \quad \text{with} \quad \mathbf{A} := DF(\mathbf{x}_0) \in \mathbb{R}^{m,n}\,,\quad \mathbf{b} := F(\mathbf{x}_0) - DF(\mathbf{x}_0)\mathbf{x}_0 \in \mathbb{R}^m\,.$$

This is a linear least squares problem of the form (6.0.3).

Note: (6.5.3) ⇒ $\mathbf{A}$ 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 (6.5.2), see Sect. 6.5.1, Rem. 6.5.2.

► Gauss-Newton iteration (under assumption (6.5.3))

Initial guess $\mathbf{x}^{(0)} \in D$

$$\mathbf{x}^{(k+1)} := \underbrace{\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \left\|F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})(\mathbf{x}-\mathbf{x}^{(k)})\right\|_2}_{\text{linear least squares problem}}\,. \qquad (6.5.7)$$

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 $\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2$
with minimal 2-norm

Code 6.5.4: 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 6.5.2:

☞ 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`.

Summary:

Advantage of the Gauss-Newton method : second derivative of F not needed.
Drawback of the Gauss-Newton method : no local quadratic convergence.

Example 6.5.5 (Non-linear data fitting (II)). → Ex. 6.5.1

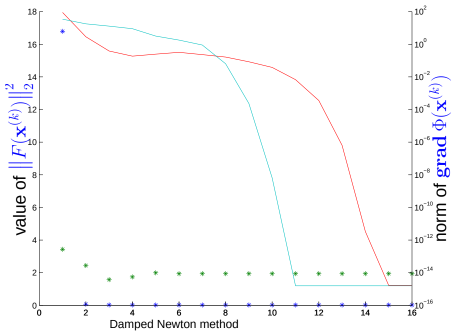
Non-linear data fitting problem (6.5.1) for $f(t) = x_1 + x_2\exp(-x_3 t)$.

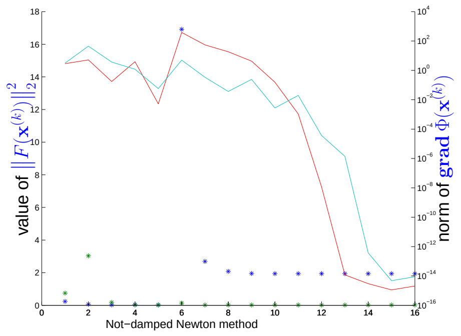
$$F(\mathbf{x}) = \begin{pmatrix} x_1 + x_2\exp(-x_3 t_1) - y_1 \\ \vdots \\ x_1 + x_2\exp(-x_3 t_m) - y_m \end{pmatrix} : \mathbb{R}^3 \mapsto \mathbb{R}^m\,,\quad DF(\mathbf{x}) = \begin{pmatrix} 1 & e^{-x_3 t_1} & -x_2 t_1 e^{-x_3 t_1} \\ \vdots & \vdots & \vdots \\ 1 & e^{-x_3 t_m} & -x_2 t_m e^{-x_3 t_m} \end{pmatrix}$$

Numerical experiment:

convergence of the Newton method, damped Newton method (→ Section 3.4.4) 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);
```





Convergence behaviour of the Newton method:

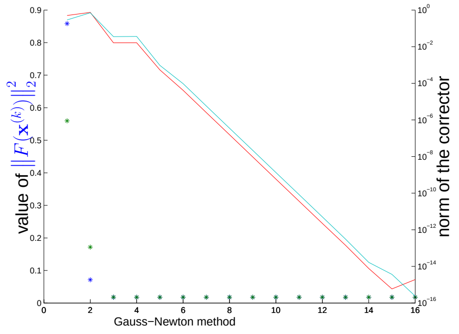
initial value $(1.8, 1.8, 0.1)^T$ (red curve) ➤ Newton method caught in local minimum,
initial value $(1.5, 1.5, 0.1)^T$ (cyan curve) ➤ fast (locally quadratic) convergence.

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.



◇

6.5.3 Trust region method (Levenberg-Marquardt method)

As in the case of Newton's method for non-linear systems of equations, see Sect. 3.4.4: 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 (6.5.7) using a penalty term

instead of $\left\|F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})\mathbf{s}\right\|^2$ minimize $\left\|F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})\mathbf{s}\right\|^2 + \lambda\,\|\mathbf{s}\|_2^2$.

$\lambda > 0 \;\hat{=}$ penalty parameter (how to choose it **?** $\longrightarrow$ heuristic)

$$\lambda = \gamma \left\|F(\mathbf{x}^{(k)})\right\|_2 \quad , \quad \gamma := \begin{cases} 10 & \text{, if } \left\|F(\mathbf{x}^{(k)})\right\|_2 \geq 10 \,, \\ 1 & \text{, if } 1 < \left\|F(\mathbf{x}^{(k)})\right\|_2 < 10 \,, \\ 0.01 & \text{, if } \left\|F(\mathbf{x}^{(k)})\right\|_2 \leq 1 \,. \end{cases}$$

▶ Modified (regularized) equation for the corrector $\mathbf{s}$:

$$\left(DF(\mathbf{x}^{(k)})^T DF(\mathbf{x}^{(k)}) + \lambda\mathbf{I}\right)\mathbf{s} = -DF(\mathbf{x}^{(k)})F(\mathbf{x}^{(k)}) \,. \tag{6.5.8}$$

7 Filtering Algorithms

Perspective of signal processing:

vector $\mathbf{x} \in \mathbb{R}^n$ $\leftrightarrow$ finite discrete (= sampled) signal.

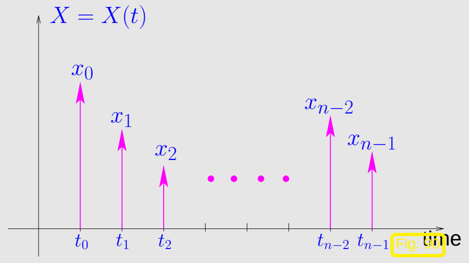
$X = X(t) \;\hat{=}$ *time-continuous* signal, $0 \leq t \leq T$,

"sampling": $x_j = X(j\Delta t)$, $j = 0, \dots, n-1$,
$n \in \mathbb{N}$, $n\Delta t \leq T$.

$\Delta t > 0 \;\hat{=}$ time between samples.

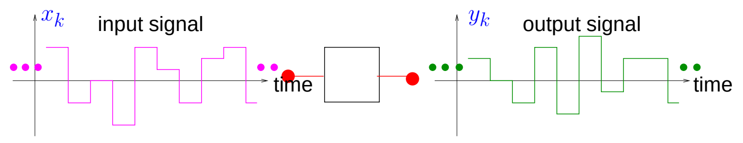
Sampled values arranged in a vector $\mathbf{x} = (x_0, \dots, x_{n-1})^T \in \mathbb{R}^n$.

Note: vector indices $0, \dots, n-1$ **!**
("C-style indexing").

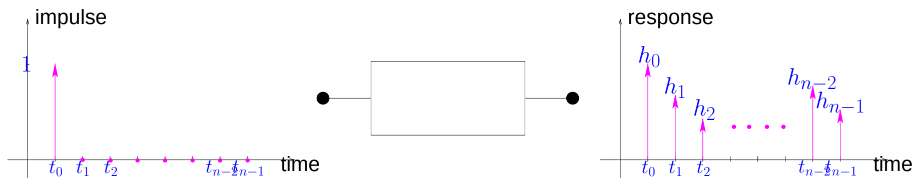


7.1 Discrete convolutions

Example 7.1.1 (Discrete finite linear time-invariant causal channel (filter)).



Impulse response of channel (filter): $\mathbf{h} = (h_0, \dots, h_{n-1})^T$



impulse
1
t_0 t_1 t_2 t_{n-2} t_{n-1}
time
response
h_0
h_1
h_2
h_{n-2}
h_{n-1}
t_0 t_1 t_2 t_{n-2} t_{n-1}
time

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

output$(\mu \cdot$ signal 1 $+ \lambda \cdot$ signal 2$) = \mu \cdot$ output(signal 1$) + \lambda \cdot$ output(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 $\mathbf{x} = (x_0, \dots, x_{n-1})^T \in \mathbb{R}^n$ is

a superposition of x_j-weighted $j\Delta t$-shifted impulse responses

channel is causal!

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j \ , \quad k = 0, \dots, 2n-2 \quad (h_j := 0 \text{ for } j < 0 \text{ and } j \geq n) \ . \tag{7.1.1}$$

$\mathbf{x} = (x_0, \dots, x_{n-1})^T \in \mathbb{R}^n \mathrel{\hat=}$ input signal $\mapsto$ $\mathbf{y} = (y_0, \dots, y_{2n-2})^T \in \mathbb{R}^{2n-1} \mathrel{\hat=}$ output signal.

Matrix notation of (7.1.1):

$$\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \\ \vdots \\ y_{2n-2} \end{pmatrix} = \begin{pmatrix} h_0 & 0 & \cdots & 0 \\ h_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ h_{n-1} & \cdots & h_1 & h_0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & h_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ \\ \vdots \\ x_{n-1} \end{pmatrix} . \tag{7.1.2}$$

◇

Example 7.1.2 (Multiplication of polynomials).

$$p(z) = \sum_{k=0}^{n-1} a_k z^k \ , \quad q(z) = \sum_{k=0}^{n-1} b_k z^k \quad \blacktriangleright \quad (pq)(z) = \sum_{k=0}^{2n-2} \underbrace{\Big(\sum_{j=0}^{k} a_j b_{k-j} \Big)}_{=:c_k} z^k \tag{7.1.3}$$

➢ coefficients of product polynomial by discrete convolution of coefficients of polynomial factors!

◇

Both in (7.1.1) and (7.1.3) we recognize the same pattern of a particular *bi-linear* combination of

- discrete signals in Ex. 7.1.1,
- polynomial coefficient sequences in Ex. 7.1.2.

Definition 7.1.1 (Discrete convolution)**.**
Given $\mathbf{x} = (x_0, \dots, x_{n-1})^T \in \mathbb{K}^n$, $\mathbf{h} = (h_0, \dots, h_{n-1})^T \in \mathbb{K}^n$ *their discrete convolution (*ger.: *diskrete Faltung) is the vector* $\mathbf{y} \in \mathbb{K}^{2n-1}$ *with components*

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j \;, \quad k = 0, \dots, 2n-2 \quad (h_j := 0 \text{ for } j < 0) \;. \tag{7.1.4}$$

✎ Notation for discrete convolution (7.1.4): $\mathbf{y} = \mathbf{h} * \mathbf{x}$.

Defining $x_j := 0$ for $j < 0$, we find that *discrete convolution is commutative*:

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j = \sum_{l=0}^{n-1} h_l x_{k-l} \;, \quad k = 0, \dots, 2n-2 \;, \qquad (\text{that is, } \mathbf{h} * \mathbf{x} = \mathbf{x} * \mathbf{h}\;)\;,$$

obtained by index transformation $l \leftarrow k - j$.

Remark 7.1.3 (Convolution of sequences).

The notion of a discrete convolution of Def. 7.1.1 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 7.1.4 (Linear filtering of periodic signals).

n-periodic signal ($n \in \mathbb{N}$) = sequence $(x_j)_{j\in\mathbb{Z}}$ with $\boxed{x_{j+n} = x_j \quad \forall j \in \mathbb{Z}}$

➢ n-periodic signal $(x_j)_{j\in\mathbb{Z}}$ fixed by $x_0, \dots, x_{n-1} \leftrightarrow$ vector $\mathbf{x} = (x_0, \dots, x_{n-1})^T \in \mathbb{R}^n$.

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

$$y_k = \sum_{j=0}^{n-1} p_{k-j} x_j \quad \text{for some} \quad p_0, \dots, p_{n-1} \in \mathbb{R} \;. \tag{7.1.5}$$

Note: $p_0, \dots, p_{n-1}$ does not agree with the impulse response of the filter.

Matrix notation:

$$\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \vdots \\ y_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} p_0 & p_{n-1} & p_{n-2} & \cdots & & \cdots & p_1 \\ p_1 & p_0 & p_{n-1} & & & & \vdots \\ p_2 & p_1 & p_0 & \ddots & & & \\ \vdots & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ \vdots & & & & \ddots & \ddots & p_{n-1} \\ p_{n-1} & \cdots & & & & p_1 & p_0 \end{pmatrix}}_{=:\mathbf{P}} \begin{pmatrix} x_0 \\ \vdots \\ \\ \\ \vdots \\ x_{n-1} \end{pmatrix} . \tag{7.1.6}$$

► $(\mathbf{P})_{ij} = p_{i-j}$, $1 \le i,j \le n$, with $p_j := p_{j+n}$ for $1-n \le j < 0$.

◇

Definition 7.1.2 (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. 7.1.2 ➢ discrete periodic convolution of vectors: $\mathbf{z} = \mathbf{x} *_n \mathbf{y} \in \mathbb{K}^n$, $\mathbf{x}, \mathbf{y} \in \mathbb{K}^n$.

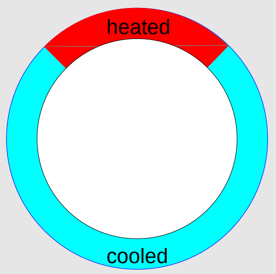
Example 7.1.5 (Radiative heat transfer).

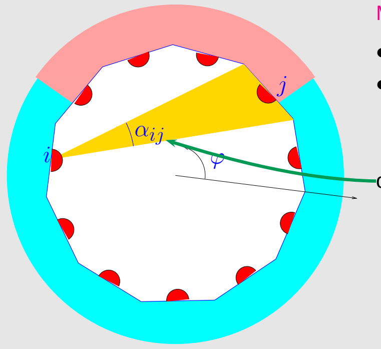
Beyond signal processing discrete periodic convolutions occur in mathematical models:

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).

Task: compute local heat fluxes.





Modeling (discretization):

- approximation by regular n-polygon, edges Γ_j,
- isotropic radiation of each edge Γ_j (power I_j),

radiative heat flow $\Gamma_j \rightarrow \Gamma_i$: $P_{ji} := \frac{\alpha_{ij}}{\pi} I_j$,

opening angle: $\alpha_{ij} = \pi\,\gamma_{|i-j|}$, $1 \le i,j \le n$,

power balance:
$$\underbrace{\sum_{i=1,i\neq j}^{n} P_{ji}}_{=I_j} - \sum_{i=1,i\neq j}^{n} P_{ij} = Q_j \ . \qquad (7.1.7)$$

$Q_j \hat{=}$ heat flux through Γ_j, satisfies

$$Q_j := \int_{\frac{2\pi}{n}(j-1)}^{\frac{2\pi}{n}j} q(\varphi)\,\mathrm{d}\varphi \ , \quad q(\varphi) := \begin{cases} \text{local heating} & \text{, if } \varphi \in \Gamma_H \ , \\ -\frac{1}{|\Gamma_K|}\int_{\Gamma_H} q(\varphi)\,\mathrm{d}\varphi & \text{(const.), if } \varphi \in \Gamma_K \ . \end{cases}$$

$$(7.1.7) \quad \Rightarrow \quad \text{LSE:} \quad I_j - \sum_{i=1,i\neq j}^{n} \frac{\alpha_{ij}}{\pi} I_i = Q_j \ , \quad j = 1,\dots,n \ .$$

$$n = 8: \quad \begin{pmatrix} 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 \\ -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 \\ -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 \\ -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_2 & -\gamma_4 \\ -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 & -\gamma_3 \\ -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 & -\gamma_2 \\ -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 & -\gamma_1 \\ -\gamma_1 & -\gamma_2 & -\gamma_3 & -\gamma_4 & -\gamma_3 & -\gamma_2 & -\gamma_1 & 1 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \\ I_8 \end{pmatrix} = \begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \\ Q_7 \\ Q_8 \end{pmatrix} . \qquad (7.1.8)$$

This is a linear system of equations with symmetric, singular, and (by Thm. 5.1.3, $\sum \gamma_i \le 1$) positive semidefinite ($\rightarrow$ Def. 2.7.1) system matrix.

Note: matrices from (7.1.6) and (7.1.8) have the same structure !

Observe: LSE from (7.1.8) can be written by means of the discrete periodic convolution ($\rightarrow$ Def. 7.1.2) of vectors $\mathbf{y} = (1, -\gamma_1, -\gamma_2, -\gamma_3, -\gamma_4, -\gamma_3, -\gamma_2, -\gamma_1)$, $\mathbf{x} = (I_1, \dots, I_8)$

$$(7.1.8) \quad \leftrightarrow \quad \mathbf{y} *_8 \mathbf{x} = (Q_1, \dots, Q_8)^T \ .$$

Definition 7.1.3 (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-periodic sequence:* $\quad c_{ij} = u_{j-i}$, $1 \le i,j \le n$.

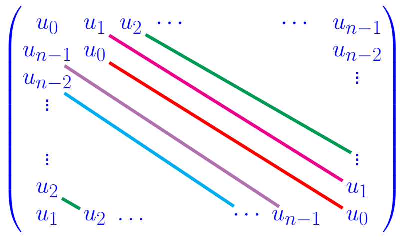
☞ Circulant matrix has constant (main, sub- and super-) diagonals (for which indices $j - i = \text{const.}$).

☞ 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 $\mathbf{C} \in \mathbb{K}^{n,n}$)

Structure of circulant matrix $\quad \triangleright$


u_0 u_1 u_2 $\cdots$ $\cdots$ u_{n-1}
u_{n-1} u_0 u_{n-2}
u_{n-2}
u_2 u_1
u_1 u_2 $\cdots$ $\cdots$ u_{n-1} u_0

Remark 7.1.6 (Reduction to periodic convolution).

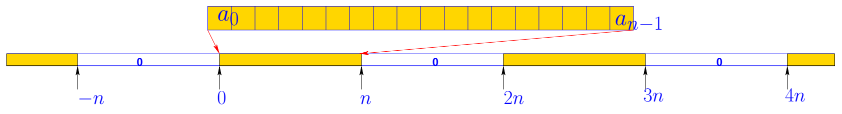
Recall discrete convolution ($\rightarrow$ Def. 7.1.1) of $\mathbf{a} = (a_0, \dots, a_{n-1})^T \in \mathbb{K}^n$, $\mathbf{b} = (b_0, \dots, b_{n-1})^T \in \mathbb{K}^n$:

$$(\mathbf{a} * \mathbf{b})_k = \sum_{j=0}^{n-1} a_j b_{k-j} \ , \quad k = 0, \dots, 2n-2 \ .$$

Expand $a_0,\dots,a_{n-1}$ and $b_0,\dots,b_{n-1}$ to $2n-1$-periodic sequences by zero padding:

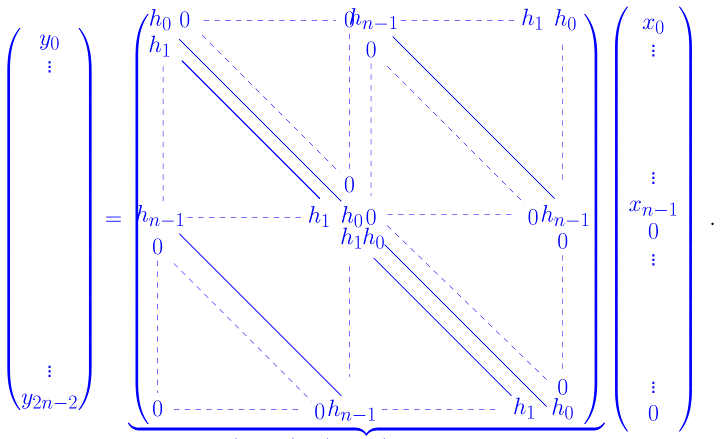
$$x_k := \begin{cases} a_k & \text{, if } 0 \le k < n \,, \\ 0 & \text{, if } n \le k < 2n-1 \end{cases} \quad , \quad y_k := \begin{cases} b_k & \text{, if } 0 \le k < n \,, \\ 0 & \text{, if } n \le k < 2n-1 \,, \end{cases} \tag{7.1.9}$$

and periodic extension: $x_k = x_{2n-1+k}$, $y_k = y_{2n-1+k}$ for all $k \in \mathbb{Z}$.



$$\blacktriangleright \quad (\mathbf{a} * \mathbf{b})_k = (\mathbf{x} *_{2n-1} \mathbf{y})_k \;, \quad k = 0,\dots,2n-2 \,. \tag{7.1.10}$$

Matrix view of reduction to periodic convolution, *cf.* (7.1.2)



a $(2n-1)\times(2n-1)$ circulant matrix!

△

7.2 Discrete Fourier Transform (DFT)

Example 7.2.1 (Eigenvectors of circulant matrices).

Code 7.2.2: 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);

for j=1:n
```

```
  figure; bar(1:n,[real(V2(:,j)),imag(V2(:,j))],1,'grouped');
  title(sprintf('Circulant␣matrix␣2,␣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/circeig2ev%d.eps',j));
end

figure; plot(1:n,real(diag(D1)),'r+',1:n,imag(diag(D1)),'b+',...
          1:n,real(diag(D2)),'m*',1:n,imag(diag(D2)),'k*');
ax = axis; axis([0 n+1 ax(3) ax(4)]);
xlabel('{\bf␣index␣of␣eigenvalue}','fontsize',14);
ylabel('{\bf␣eigenvalue}','fontsize',14);
legend('C_1:␣real(ev)', 'C_1:␣imag(ev)', 'C_2:␣real(ev)', 'C_2:␣
  imag(ev)', 'location','northeast');

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

Random 8×8 circulant matrices $\mathbf{C}_1$, $\mathbf{C}_2$ ($\rightarrow$ Def. 7.1.3)

eigenvalues ▷

Generated by MATLAB-command:

C = gallery('circul',rand(n,1));

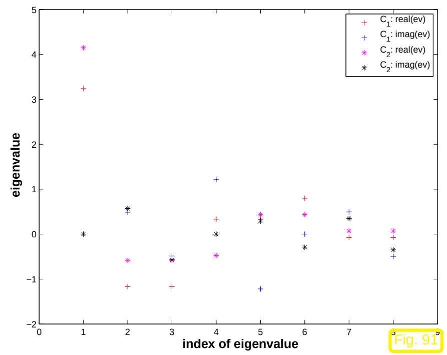
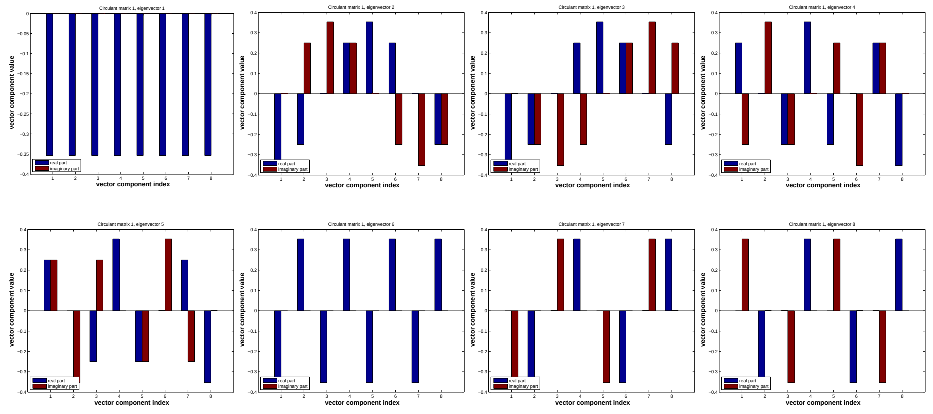


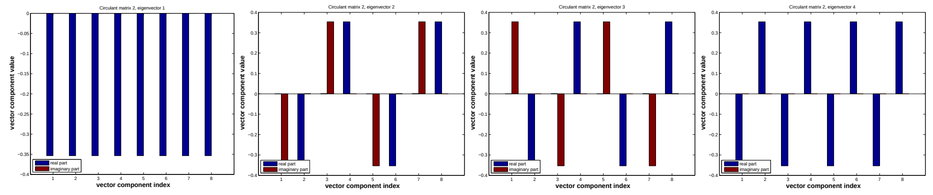
Fig. 91

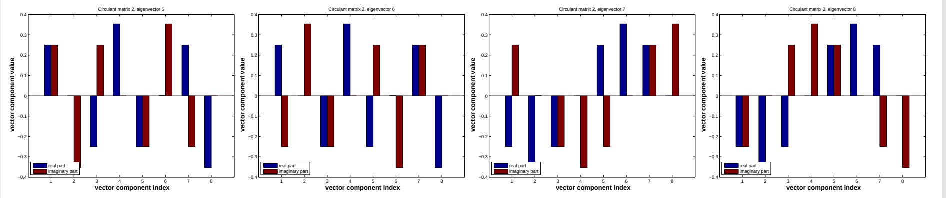
Little relationship between (complex!) eigenvalues can be observed, as can be expected from random matrices with entries $\in [0, 1]$.

Eigenvectors of matrix $\mathbf{C}_1$:



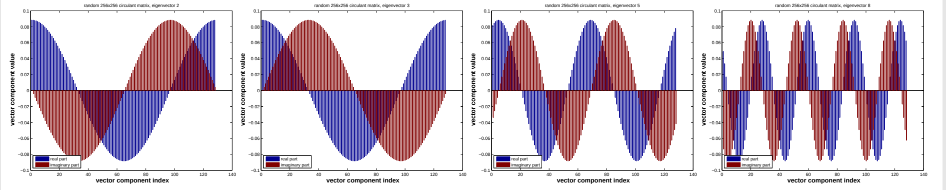
Eigenvectors of matrix $\mathbf{C}_2$





Observation: the different random circulant matrices have the same eigenvectors!

Eigenvectors of C = gallery('circul',(1:128)'); :



The eigenvectors remind us of sampled *trigonometric functions* $\cos(k/n)$, $\sin(k/n)$, $k = 0, \dots, n-1$!

◇

Remark 7.2.3 (Why using $\mathbb{K} = \mathbb{C}$?).

Ex. 7.2.1: *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 $\overline{\omega}_n = \omega_n^{-1}$, $\boxed{\omega_n^n = 1}$, $\omega_n^{n/2} = -1$, $\omega_n^k = \omega_n^{k+n} \quad \forall\, k \in \mathbb{Z}$, (7.2.1)

$$\sum_{k=0}^{n-1} \omega_n^{kj} = \begin{cases} n & \text{, if } j = 0 \mod n , \\ 0 & \text{, if } j \neq 0 \mod n . \end{cases} \qquad (7.2.2)$$

(7.2.2) is a simple consequence of the geometric sum formula

$$\sum_{k=0}^{n-1} q^k = \frac{1-q^n}{1-q} \quad \forall\, q \in \mathbb{C} \setminus \{1\}\,, \quad n \in \mathbb{N}\,. \tag{7.2.3}$$

$$\Rightarrow \quad \sum_{k=0}^{n-1} \omega_n^{kj} = \frac{1-\omega_n^{nj}}{1-\omega_n^j} = \frac{1-\exp(-2\pi i j)}{1-\exp(-2\pi i j/n)} = 0\,,$$

because $\exp(-2\pi i j) = \omega_n^{nj} = (\omega_n^n)^j = 1$ for all $j \in \mathbb{Z}$.

Now we want to confirm the conjecture gleaned from Ex. 7.2.1 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 (→ Def. 7.1.3), $c_{ij} = u_{i-j}$, for n-periodic sequence $(u_k)_{k\in\mathbb{Z}}$, $u_k \in \mathbb{C}$

$\mathbf{v}_k \in \mathbb{C}^n$ with $\mathbf{v}_k := (\omega_n^{jk})_{j=0}^{n-1} \in \mathbb{C}^n$, $k \in \{0,\dots,n-1\}$.

$$(\mathbf{C}\mathbf{v}_k)_j = \sum_{l=0}^{n-1} u_{j-l}\omega_n^{lk} \overset{(u_{j-l}\omega_n^{lk})_{l\in\mathbb{Z}} \text{ is } n\text{-periodic!}}{=} \sum_{l=j-n+1}^{j} u_{j-l}\omega_n^{lk}$$

$$\underset{\text{change of summation index}}{=} \sum_{l=0}^{n-1} u_l \omega_n^{(j-l)k} = \omega_n^{jk} \underbrace{\boxed{\sum_{l=0}^{n-1} u_l \omega_n^{-lk}}}_{\text{independent of } j\,!} = \lambda_k \cdot \omega_n^{jk} = \lambda_k \cdot (\mathbf{v}_k)_j \;. \tag{7.2.4}$$

▶ $\boxed{\mathbf{v} \text{ is eigenvector of } \mathbf{C} \text{ to eigenvalue } \lambda_k = \textstyle\sum_{l=0}^{n-1} u_l \omega_n^{-lk}.}$

Orthogonal trigonometric basis of $\mathbb{C}^n$ = *eigenvector basis* for circulant matrices

$$\left\{ \begin{pmatrix} \omega_n^0 \\ \vdots \\ \\ \vdots \\ \omega_n^0 \end{pmatrix} \begin{pmatrix} \omega_n^0 \\ \omega_n^1 \\ \vdots \\ \vdots \\ \omega_n^{n-1} \end{pmatrix} \cdots \begin{pmatrix} \omega_n^0 \\ \omega_n^{n-2} \\ \omega_n^{2(n-2)} \\ \vdots \\ \vdots \\ \omega_n^{(n-1)(n-2)} \end{pmatrix} \begin{pmatrix} \omega_n^0 \\ \omega_n^{n-1} \\ \omega_n^{2(n-1)} \\ \vdots \\ \vdots \\ \omega_n^{(n-1)^2} \end{pmatrix} \right\}.$$

(7.2.2) $\Rightarrow$ orthogonality of basis vectors:

$$\mathbf{v}_k := (\omega_n^{jk})_{j=0}^{n-1} \in \mathbb{C}^n: \quad \mathbf{v}_k^H \mathbf{v}_m = \sum_{j=0}^{n-1} \omega_n^{-jk}\omega_n^{jm} = \sum_{j=0}^{n-1} \omega_n^{(m-k)j} \overset{(7.2.2)}{=} 0 \quad \text{, if } k \neq m\,. \tag{7.2.5}$$

Matrix of change of basis trigonometrical basis → standard basis: Fourier-matrix

$$\mathbf{F}_n = \begin{pmatrix} \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \cdots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \cdots & \omega_n^{2n-2} \\ \vdots & \vdots & & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{pmatrix} = \left(\omega_n^{lj}\right)_{l,j=0}^{n-1} \in \mathbb{C}^{n,n}\,. \tag{7.2.6}$$

Lemma 7.2.1 (Properties of Fourier matrix)**.**

The scaled Fourier-matrix $\frac{1}{\sqrt{n}}\mathbf{F}_n$ *is unitary (→ Def. 2.8.1):* $\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^H = \frac{1}{n}\overline{\mathbf{F}}_n$.

Proof. The lemma is immediate from (7.2.5) and (7.2.2),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 7.2.4 (Spectrum of Fourier matrix).

$$\tfrac{1}{n^2}\mathbf{F}_n^4 = I \quad \Rightarrow \quad \sigma(\tfrac{1}{\sqrt{n}}\mathbf{F}_n) \subset \{1,-1,i,-i\}\,,$$

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. 5.1.1.

△

Lemma 7.2.2 (Diagonalization of circulant matrices (→ Def. 7.1.3))**.**
For any circulant matrix $\mathbf{C} \in \mathbb{K}^{n,n}$, $c_{ij} = u_{i-j}$, $(u_k)_{k\in\mathbb{Z}}$ *n-periodic sequence, holds true*

$$\mathbf{C}\overline{\mathbf{F}}_n = \overline{\mathbf{F}}_n \operatorname{diag}(d_1,\dots,d_n) \quad , \quad \mathbf{d} = \mathbf{F}_n(u_0,\dots,u_{n-1})^T\,.$$

Proof. Straightforward computation, see (7.2.4). ☐

Conclusion (from $\overline{\mathbf{F}}_n = n\mathbf{F}_n^{-1}$): $\boxed{\mathbf{C} = \mathbf{F}_n^{-1}\operatorname{diag}(d_1, \dots, d_n)\mathbf{F}_n}$. (7.2.7)

Lemma 7.2.2, (7.2.7) $\succ$ 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 7.2.3 (Discrete Fourier transform (DFT)).
The linear map $\mathcal{F}_n : \mathbb{C}^n \mapsto \mathbb{C}^n$, $\mathcal{F}_n(\mathbf{y}) := \mathbf{F}_n\mathbf{y}$, $\mathbf{y} \in \mathbb{C}^n$, *is called discrete Fourier transform (DFT), i.e. for* $\mathbf{c} := \mathcal{F}_n(\mathbf{y})$

$$c_k = \sum_{j=0}^{n-1} y_j\,\omega_n^{kj} \quad , \quad k = 0, \dots, n-1 \,. \tag{7.2.8}$$

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);`

7.2.1 Discrete convolution via DFT

Recall discrete periodic convolution $z_k = \sum_{j=0}^{n-1} u_{k-j}x_j$ ($\rightarrow$ Def. 7.1.2), $k = 0, \dots, n-1$

$\updownarrow$

multiplication with circulant matrix ($\rightarrow$ Def. 7.1.3) $\mathbf{z} = \mathbf{C}\mathbf{x}$, $\mathbf{C} := (u_{i-j})_{i,j=1}^{n}$.

Idea: (7.2.7) $\succ$ $\mathbf{z} = \mathbf{F}_n^{-1}\operatorname{diag}(\mathbf{F}_n\mathbf{u})\mathbf{F}_n\mathbf{x}$

Code 7.2.6: discrete periodic convolution: straightforward implementation

```
function z=pconv(u,x)
n = length(x); z = zeros(n,1);
for i=1:n, z(i)=dot(conj(u),
   x([i:-1:1,n:-1:i+1])); 
end
```

Code 7.2.8: discrete periodic convolution: DFT implementation

```
function z=pconvfft(u,x)
z = ifft(fft(u).*fft(x));
```

Rem. 7.1.6: discrete convolution of n-vectors ($\rightarrow$ Def. 7.1.1) by *periodic* discrete convolution of $2n-1$-vectors (obtained by zero padding):

Implementation of discrete convolution ($\rightarrow$ Def. 7.1.1) based on periodic discrete convolution ▷

Built-in MATLAB-function:

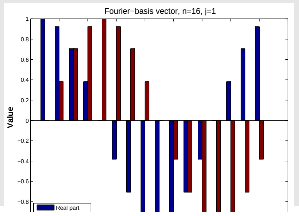
`y = conv(h,x);`

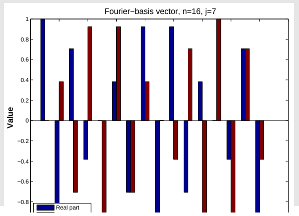
Code 7.2.9: discrete convolutiuon: DFT implementation

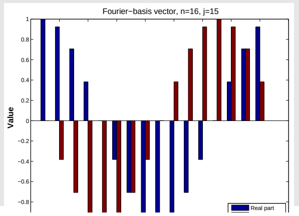
```
function y = myconv(h,x)
n = length(h);
% Zero padding
h = [h;zeros(n-1,1)]; x =
   [x;zeros(n-1,1)];
% Periodic discrete convolution of length 2n - 1
y = pconvfft(h,x);
```

7.2.2 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 ($n = 16$):







▶ 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 7.2.10 (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
```

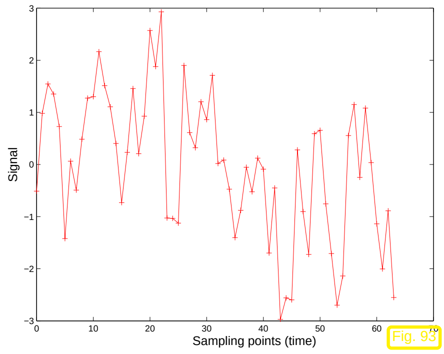


Fig. 93

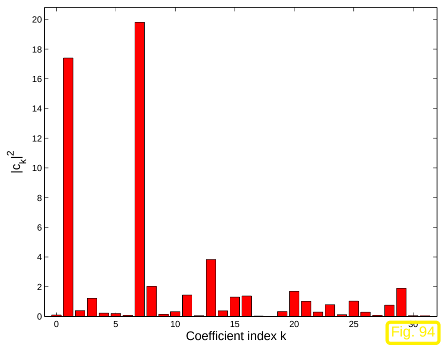


Fig. 94

Generating Fig. 94:

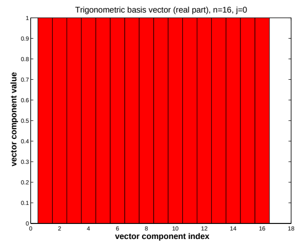
```
c = fft(y); p = (c.*conj(c))/64;

figure('Name','power␣spectrum');
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);
```

Frequencies present in unperturbed signal become evident in frequency domain.

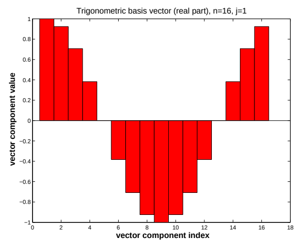
◇

Remark 7.2.11 (“Low” and “high” frequencies).

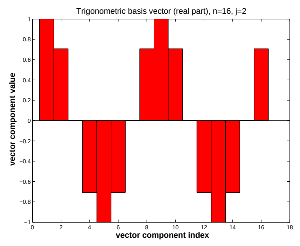
Plots of real parts of trigonometric basis vectors $(\mathbf{F}_n)_{:,j}$ (= columns of Fourier matrix), $n = 16$.



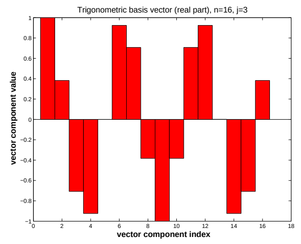
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,0}$



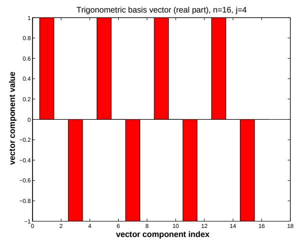
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,1}$



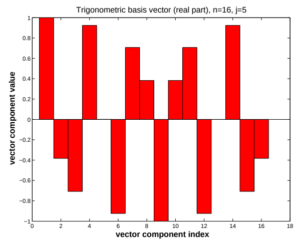
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,2}$



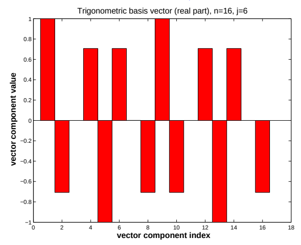
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,3}$



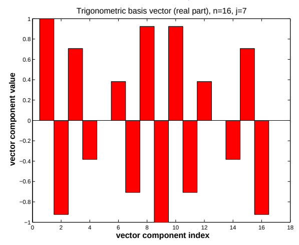
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,4}$



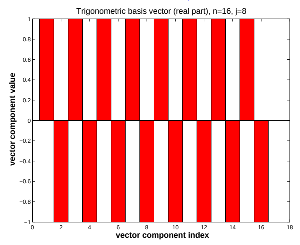
$\mathrm{Re}\,(\mathbf{F}_{16})_{:,5}$



$\mathrm{Re}\,(\mathbf{F}_{16})_{:,6}$



$\mathrm{Re}\,(\mathbf{F}_{16})_{:,7}$



$\mathrm{Re}\,(\mathbf{F}_{16})_{:,8}$

△

By elementary trigonometric identities:

$$\mathrm{Re}\,(\mathbf{F}_n)_{:,j} = \left(\mathrm{Re}\,\omega_n^{(j-1)k}\right)_{k=0}^{n-1} = (\mathrm{Re}\exp(2\pi i(j-1)k/n))_{k=0}^{n-1} = (\cos(2\pi(j-1)x))_{x=0,\frac{1}{n},\dots,1-\frac{1}{n}} .$$

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 7.2.12: DFT-based frequency filtering

```
function [low,high] =
   freqfilter(y,k)
m = length(y)/2; c = fft(y);
clow = c; clow(m+1-k:m+1+k) = 0;
chigh = c-clow;
low = real(ifft(clow));
high = real(ifft(chigh));
```

(can be optimised exploiting $y_j \in \mathbb{R}$ and $c_{n/2-k} = \bar{c}_{n/2+k}$)

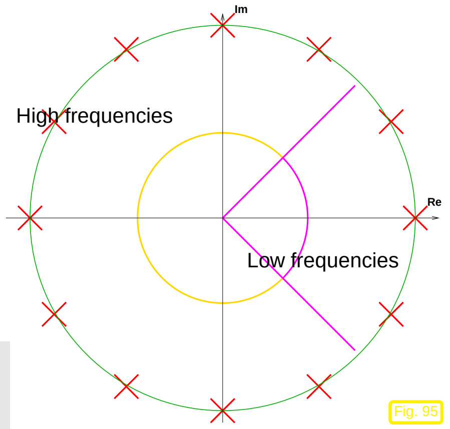


Fig. 95

Map y $\mapsto$ `low` (in Code 7.2.11) $\hat{=}$ low pass filter (*ger.:* Tiefpass).

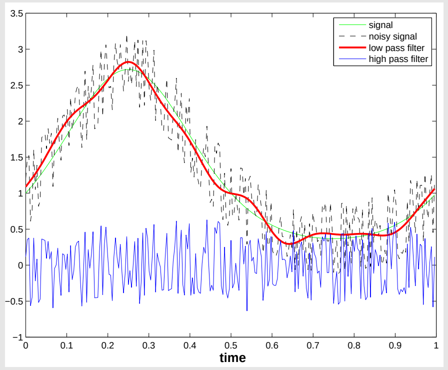
Map y $\mapsto$ `high` (in Code 7.2.11) $\hat{=}$ high pass filter (*ger.:* Hochpass).

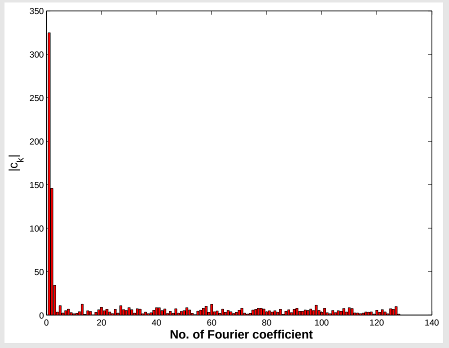
Example 7.2.13 (Frequency filtering by DFT).

Noisy signal:

```
n = 256; y = exp(sin(2*pi*((0:n-1)')/n)) + 0.5*sin(exp(1:n)');
```

Frequency filtering by Code 7.2.11 with $k = 120$.





Low pass filtering can be used for *denoising*, that is, the removal of high frequency perturbations of a signal.

◇

Example 7.2.14 (Sound filtering by DFT).

Code 7.2.15: 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';
```

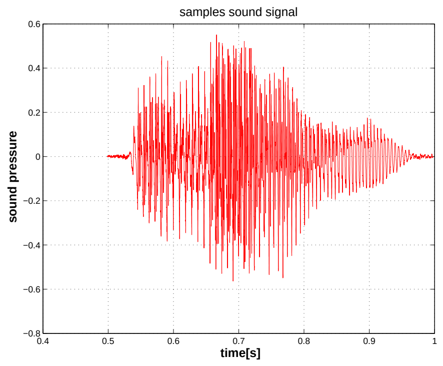
```
c = fft(y);

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-');
title('low␣frequency␣power␣spectrum','fontsize',14);
```

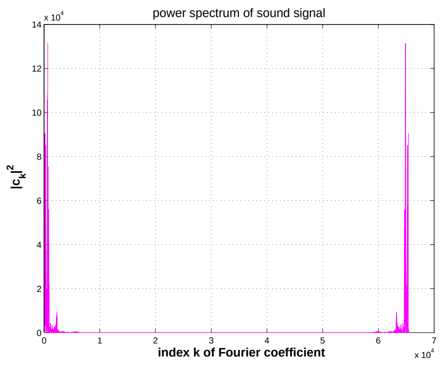
```
xlabel('{\bf␣index␣k␣of␣Fourier␣coefficient}','fontsize',14);
ylabel('{\bf␣|c_k|^2}','fontsize',14);
grid on;

print -depsc2 '../PICTURES/soundlowpower.eps';

for m=[1000,3000,5000]

  % Low pass filtering
  cf = zeros(1,n);
  cf(1:m) = c(1:m); cf(n-m+1:end) = c(n-m+1:end);

  % Reconstruct filtered signal
  yf = ifft(cf);
  wavwrite(yf,freq,nbits,sprintf('hellof%d.wav',m));

  k = k+1;
  s{k} = real(yf);
  leg{k} = sprintf('cutt-off␣=␣%d',m');
end

% Plot original signal and filtered signals
figure('name','sound␣filtering');
```

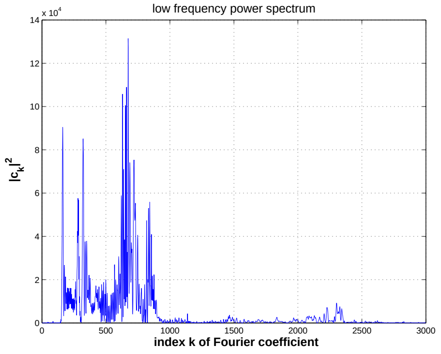
```
plot((30000:32000)/freq,s{1}(30000:32000),'r-',...
     (30000:32000)/freq,s{2}(30000:32000),'b--',...
     (30000:32000)/freq,s{3}(30000:32000),'m--',...
     (30000:32000)/freq,s{2}(30000:32000),'k--');
xlabel('{\bf␣time[s]}','fontsize',14);
ylabel('{\bf␣sound␣pressure}','fontsize',14);
legend(leg,'location','southeast');

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

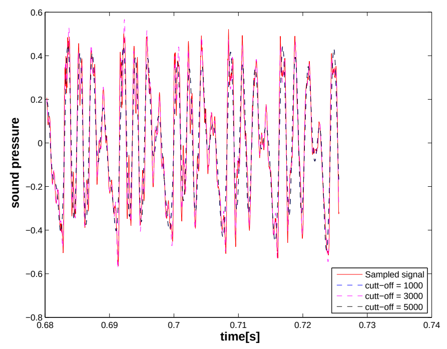
DFT based low pass frequency filtering of sound

```
[y,sf,nb] = wavread('hello.wav');
c = fft(y); c(m+1:end-m) = 0;
wavwrite(ifft(c),sf,nb,'filtered.wav');
```









The power spectrum of a signal $\mathbf{y} \in \mathbb{C}^n$ is the vector $\left(|c_j|^2\right)_{j=0}^{n-1}$, where $\mathbf{c} = \mathbf{F}_n\mathbf{y}$ is the discrete Fourier transform of $\mathbf{y}$.

◇

7.2.3 Real DFT

Signal obtained from sampling a time-dependent voltage: a real vector.

Aim: efficient DFT (Def. 7.2.3) $(c_0,\dots,c_{n-1})$ for *real* coefficients $(y_0,\dots,y_{n-1})^T \in \mathbb{R}^n$, $n = 2m$, $m \in \mathbb{N}$.

If $y_j \in \mathbb{R}$ in DFT formula (7.2.8), we obtain redundant output

$$\omega_n^{(n-k)j} = \overline{\omega}_n^{kj} \;,\quad k = 0,\dots,n-1 \;,$$

$$\Rightarrow\quad \overline{c}_k = \sum_{j=0}^{n-1} y_j \overline{\omega}_n^{kj} = \sum_{j=0}^{n-1} y_j \omega_n^{(n-k)j} = c_{n-k} \;,\quad k = 1,\dots,n-1 \;.$$

Idea: map $\mathbf{y} \in \mathbb{R}^n$, to $\mathbb{C}^m$ and use DFT of length m.

$$h_k = \sum_{j=0}^{m-1} (y_{2j} + iy_{2j+1})\,\omega_m^{jk} = \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} + i \cdot \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}} \;, \tag{7.2.9}$$

$$\overline{h}_{m-k} = \sum_{j=0}^{m-1} \overline{y_{2j} + iy_{2j+1}}\,\omega_m^{j(m-k)} = \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} - i \cdot \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}} \;. \tag{7.2.10}$$

$$\Rightarrow\quad \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} = \tfrac{1}{2}(h_k + \overline{h}_{m-k}) \quad,\quad \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}} = -\tfrac{1}{2}i(h_k - \overline{h}_{m-k}) \;.$$

Use simple identities for roots of unity:

$$c_k = \sum_{j=0}^{n-1} y_j\,\omega_n^{jk} = \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j}\,\omega_m^{jk}} + \omega_n^k \cdot \boxed{\textstyle\sum_{j=0}^{m-1} y_{2j+1}\,\omega_m^{jk}} \;. \tag{7.2.11}$$

$$\Rightarrow\quad \begin{cases} c_k = \tfrac{1}{2}(h_k + \overline{h}_{m-k}) - \tfrac{1}{2}i\omega_n^k(h_k - \overline{h}_{m-k}) \;, & k = 0,\dots,m-1 \;, \\ c_m = \mathrm{Re}\{h_0\} - \mathrm{Im}\{h_0\} \;, \\ c_k = \overline{c}_{n-k} \;, \quad k = m+1,\dots,n-1 \;. \end{cases} \tag{7.2.12}$$

MATLAB-Implementation (by a DFT of length $n/2$):

(Note: not really optimal MATLAB implementation)

Code 7.2.16: 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))];
```

7.2.4 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 $$\left\{(\mathbf{F}_m)_{:,j}(\mathbf{F}_n)^T_{:,\ell},\ 1\leq j\leq m,\ 1\leq \ell\leq n\right\}\ . \qquad (7.2.13)$$

Basis transform: for $y_{j_1,j_2}\in\mathbb{C}$, $0\leq j_1<m$, $0\leq j_2<n$ compute (nested DFTs !)

$$c_{k_1,k_2}=\sum_{j_1=0}^{m-1}\sum_{j_2=0}^{n-1} y_{j_1,j_2}\,\omega_m^{j_1k_1}\omega_n^{j_2k_2}\quad,\quad 0\leq k_1<m\,,\ 0\leq k_2<n\,.$$

MATLAB function: fft2(Y) .

Two-dimensional DFT by *nested one-dimensional DFTs* (7.2.8):

```
fft2(Y) = fft(fft(Y).').'
```

Here: `.'` simply transposes the matrix (no complex conjugation)

Example 7.2.17 (Deblurring by DFT).

Gray-scale pixel image $\mathbf{P}\in\mathbb{R}^{m,n}$, actually $\mathbf{P}\in\{0,\dots,255\}^{m,n}$, see Ex. 5.3.9.

$(p_{l,k})_{l,k\in\mathbb{Z}} \mathrel{\hat=}$ periodically extended image:

$$p_{l,j}=(\mathbf{P})_{l+1,j+1}\quad\text{for}\quad 1\leq l\leq m,\ 1\leq j\leq n\ ,\quad p_{l,j}=p_{l+m,j+n}\quad \forall l,k\in\mathbb{Z}\,.$$

Blurring = pixel values get replaced by weighted averages of near-by pixel values
(effect of distortion in optical transmission systems)

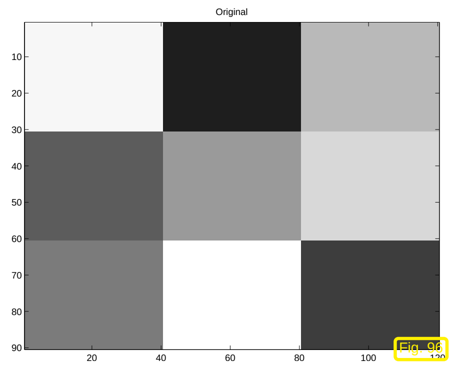
$$\underbrace{c_{l,j}}_{\text{blurred image}}=\sum_{k=-L}^{L}\sum_{q=-L}^{L}\underbrace{s_{k,q}}_{\text{point spread function}}p_{l+k,j+q}\ ,\quad \begin{array}{l}0\leq l<m\,,\\ 0\leq j<n\,,\end{array}\quad L\in\{1,\dots,\min\{m,n\}\}\,. \qquad (7.2.14)$$

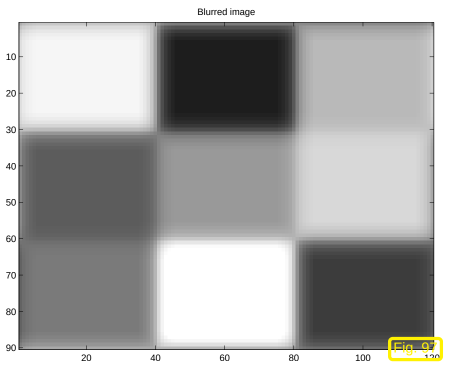
Usually: L small, $s_{k,m}\geq 0$, $\sum_{k=-L}^{L}\sum_{q=-L}^{L}s_{k,q}=1$ (an averaging)

Used in test calculations: $L=5$

$$s_{k,q}=\frac{1}{1+k^2+q^2}\,.$$

Code 7.2.18: point spread function
```
function S = psf(L)
[X,Y] = meshgrid(-L:1:L,-L:1:L);
S = 1./(1+X.*X+Y.*Y);
S = S/sum(sum(S));
```





Code 7.2.19: 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));
```

Note:

(7.2.14) defines a linear operator $\mathcal{B} : \mathbb{R}^{m,n} \mapsto \mathbb{R}^{m,n}$ ("blurring operator")

Note: more efficient implementation via MATLAB function conv2(P,S)

Code 7.2.20: blurring operator

```
function C = blur(P,S)
[m,n] = size(P); [M,N] = size(S);
if (M ~= N), error('S␣not␣quadratic'); end
L = (M-1)/2; C = zeros(m,n);
for l=1:m, for j=1:n
    s = 0;
    for k=1:(2*L+1), for q=1:(2*L+1)
        kl = l+k-L-1;
        if (kl < 1), kl = kl + m; end
        if (kl > m), kl = kl - m; end
        jm = j+q-L-1;
        if (jm < 1), jm = jm + n; end
        if (jm > n), jm = jm - n; end
        s = s+P(kl,jm)*S(k,q);
      end, end
    C(l,j) = s;
  end, end
```

Recall: derivation of (7.2.4) and Lemma 7.2.2. Try this in 2D!

$$\left(\mathcal{B}\left(\left(\omega_m^{\nu k}\omega_n^{\mu q}\right)_{k,q\in\mathbb{Z}}\right)\right)_{l,j} = \sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\,\omega_m^{\nu(l+k)}\omega_n^{\mu(j+q)} = \omega_m^{\nu l}\omega_n^{\mu j}\sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\,\omega_m^{\nu k}\omega_n^{\mu q}\,.$$

► $\mathbf{V}_{\nu,\mu} := \left(\omega_m^{\nu k}\omega_n^{\mu q}\right)_{k,q\in\mathbb{Z}}$, $0 \le \mu < m$, $0 \le \nu < n$ are the eigenvectors of $\mathcal{B}$:

$$\mathcal{B}\mathbf{V}_{\nu,\mu} = \lambda_{\nu,\mu}\mathbf{V}_{\nu,\mu}\quad,\quad\text{eigenvalue}\quad \lambda_{\nu,\mu} = \underbrace{\sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\,\omega_m^{\nu k}\omega_n^{\mu q}}_{\text{2-dimensional DFT of point spread function}} \tag{7.2.15}$$

Inversion of blurring operator
$\Updownarrow$
componentwise scaling in "Fourier domain"

Code 7.2.21: DFT based deblurring

```
function D = deblur(C,S,tol)
[m,n] = size(C); [M,N] = size(S);
if (M ~= N), error('S␣not␣quadratic'); end
L = (M-1)/2; Spad = zeros(m,n);
% Zero padding
Spad(1:L+1,1:L+1) = S(L+1:end,L+1:end);
Spad(m-L+1:m,n-L+1:n) = S(1:L,1:L);
Spad(1:L+1,n-L+1:n) = S(L+1:end,1:L);
Spad(m-L+1:m,1:L+1) = S(1:L,L+1:end);
% Inverse of blurring operator
SF = fft2(Spad);
% Test for invertibility
if (nargin < 3), tol = 1E-3; end
if (min(min(abs(SF))) < tol*max(max(abs(SF))))
  error('Deblurring␣impossible');
end
% DFT based deblurring
D = fft2(ifft2(C)./SF);
```

◇

7.3 Fast Fourier Transform (FFT)

At first glance (at (7.2.8)): DFT in $\mathbb{C}^n$ seems to require asymptotic computational effort of $O(n^2)$ (matrix×vector multiplication with dense matrix).

Example 7.3.1 (Efficiency of `fft`).

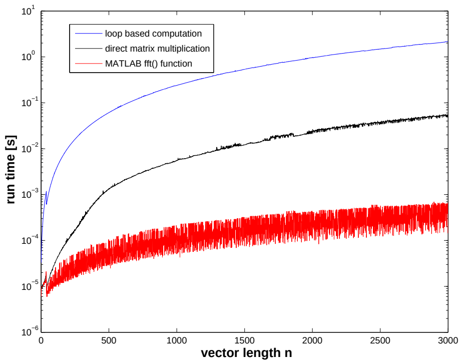
`tic-toc`-timing in MATLAB: compare `fft`, loop based implementation, and direct matrix multiplication
(MATLAB V6.5, Linux, Mobile Intel Pentium 4 - M CPU 2.40GHz, minimum over 5 runs)

Code 7.3.2: 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'
```

MATLAB-CODE naive DFT-implementation

```
c = zeros(n,1);
omega = exp(-2*pi*i/n);
c(1) = sum(y); s = omega;
for j=2:n
  c(j) = y(n);
  for k=n-1:-1:1
    c(j) = c(j)*s+y(k);
  end
  s = s*omega;
end
```



Incredible! The MATLAB `fft()`-function clearly beats the $O(n^2)$ asymptotic complexity of the other implementations. Note the logarithmic scale!

◇

The secret of MATLAB's `fft()`:

the Fast Fourier Transform algorithm [14]

(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 (7.2.8) for $n = 2m$, $m \in \mathbb{N}$:

$$\begin{aligned} c_k &= \sum_{j=0}^{n-1} y_j e^{-\frac{2\pi i}{n} jk} \\ &= \sum_{j=0}^{m-1} y_{2j} e^{-\frac{2\pi i}{n} 2jk} + \sum_{j=0}^{m-1} y_{2j+1} e^{-\frac{2\pi i}{n}(2j+1)k} \\ &= \underbrace{\sum_{j=0}^{m-1} y_{2j} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}}}_{=:\tilde{c}_k^{\text{even}}} + e^{-\frac{2\pi i}{n} k} \cdot \underbrace{\sum_{j=0}^{m-1} y_{2j+1} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}}}_{=:\tilde{c}_k^{\text{odd}}} . \end{aligned} \tag{7.3.1}$$

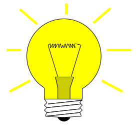
Note m-periodicity: $\tilde{c}_k^{\text{even}} = \tilde{c}_{k+m}^{\text{even}}$, $\tilde{c}_k^{\text{odd}} = \tilde{c}_{k+m}^{\text{odd}}$.

Note: $\tilde{c}_k^{\text{even}}$, $\tilde{c}_k^{\text{odd}}$ from DFTs of length m!

with $\mathbf{y}_{\text{even}} := (y_0, y_2, \dots, y_{n-2})^T \in \mathbb{C}^m$: $\left(\tilde{c}_k^{\text{even}}\right)_{k=0}^{m-1} = \mathbf{F}_m \mathbf{y}_{\text{even}}$,

with $\mathbf{y}_{\text{odd}} := (y_1, y_3, \dots, y_{n-1})^T \in \mathbb{C}^m$: $\left(\tilde{c}_k^{\text{odd}}\right)_{k=0}^{m-1} = \mathbf{F}_m \mathbf{y}_{\text{odd}}$.

(7.3.1): DFT of length $2m$ = 2$\times$ DFT of length m + 2m additions & multiplications



Idea:
divide & conquer recursion

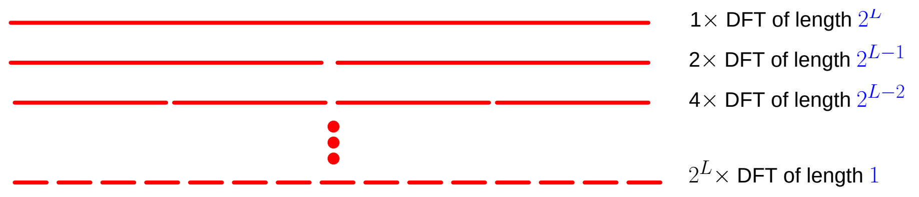
(for DFT of length $n = 2^L$)

FFT-algorithm

Code 7.3.3: Recursive FFT

```
function c = fftrec(y)
n = length(y);
if (n == 1), c = y; return;
else
  c1 = fftrec(y(1:2:n));
  c2 = fftrec(y(2:2:n));
  c = [c1;c1] +
    (exp(−2*pi*i/n).^((0:n−1)'))
    .*[c2;c2];
end
```

Computational cost of `fftrec`:



$1\times$ DFT of length 2^L
$2\times$ DFT of length 2^{L-1}
$4\times$ DFT of length 2^{L-2}
$2^L\times$ DFT of length 1

Code 7.3.2: each level of the recursion requires $O(2^L)$ elementary operations.

➢

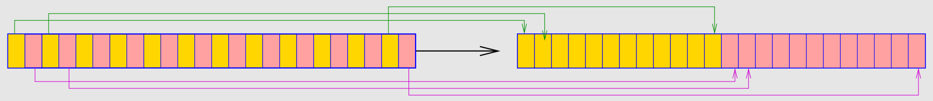
Asymptotic complexity of FFT algorithm, $n = 2^L$: $O(L2^L) = O(n\log_2 n)$

(MATLAB `fft`-function: cost $\approx 5n\log_2 n$).

Remark 7.3.4 (FFT algorithm by matrix factorization).

For $n = 2m$, $m \in \mathbb{N}$,

permutation $P_m^{\mathrm{OE}}(1,\dots,n) = (1,3,\dots,n-1,2,4,\dots,n)$.



As $\omega_n^{2j} = \omega_m^j$:

permutation of rows
$$P_m^{\mathrm{OE}}\mathbf{F}_n = \left(\begin{array}{c|c} \mathbf{F}_m & \mathbf{F}_m \\ \hline \mathbf{F}_m \begin{pmatrix} \omega_n^0 & & & \\ & \omega_n^1 & & \\ & & \ddots & \\ & & & \omega_n^{n/2-1} \end{pmatrix} & \mathbf{F}_m \begin{pmatrix} \omega_n^{n/2} & & & \\ & \omega_n^{n/2+1} & & \\ & & \ddots & \\ & & & \omega_n^{n-1} \end{pmatrix} \end{array}\right)$$

$$\left(\begin{array}{c|c} \mathbf{F}_m & \\ \hline & \mathbf{F}_m \end{array}\right) \left(\begin{array}{c|c} \mathbf{I} & \mathbf{I} \\ \hline \begin{matrix} \omega_n^0 & & & \\ & \omega_n^1 & & \\ & & \ddots & \\ & & & \omega_n^{n/2-1} \end{matrix} & \begin{matrix} -\omega_n^0 & & & \\ & -\omega_n^1 & & \\ & & \ddots & \\ & & & -\omega_n^{n/2-1} \end{matrix} \end{array}\right)$$

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, and then computes q transforms of size p. The decomposition is applied recursively to both the p- and q-point DFTs until the problem can be solved using one of several machine-generated fixed-size "codelets." The codelets in turn use several algorithms in combination, including a variation of Cooley-Tukey, a prime factor algorithm, and a split-radix algorithm. The particular factorization of n is chosen heuristically.

The execution time for fft depends on the length of the transform. It is fastest for powers of two. It is almost as fast for lengths that have only small prime factors. It is typically several times slower for lengths that are prime or which have large prime factors → Ex. 7.3.1.

Remark 7.3.5 (FFT based on general factorization).

Fast Fourier transform algorithm for DFT of length $n = pq$, $p, q \in \mathbb{N}$ (Cooley-Tuckey-Algorithm)

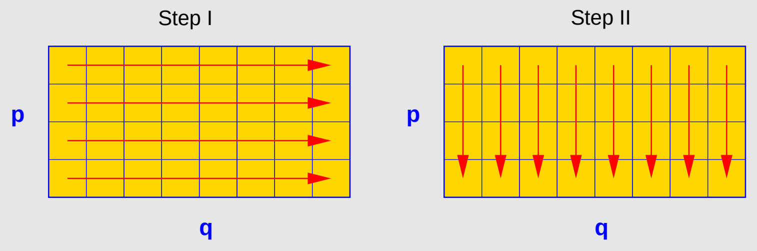
$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{jk} \overset{[j=:lp+m]}{=} \sum_{m=0}^{p-1}\sum_{l=0}^{q-1} y_{lp+m} e^{-\frac{2\pi i}{pq}(lp+m)k} = \sum_{m=0}^{p-1} \omega_n^{mk} \sum_{l=0}^{q-1} y_{lp+m}\, \omega_q^{l(k \mod q)} . \tag{7.3.2}$$

Step I: perform p DFTs of length q $\quad z_{m,k} := \sum_{l=0}^{q-1} y_{lp+m}\, \omega_q^{lk}$, $\quad 0 \leq m < p$, $0 \leq k < q$.

Step II: for $k =: rq + s$, $0 \leq r < p, 0 \leq s < q$

$$c_{rq+s} = \sum_{m=0}^{p-1} e^{-\frac{2\pi i}{pq}(rq+s)m} z_{m,s} = \sum_{m=0}^{p-1} \left(\omega_n^{ms} z_{m,s}\right) \omega_p^{mr}$$

and hence q DFTs of length p give all c_k.



Remark 7.3.6 (FFT for prime n).

When $n \neq 2^L$, even the Cooley-Tuckey algorithm of Rem. 7.3.5 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 [33]. It then uses the Cooley-Tukey decomposition described above to compute the $(n-1)$-point DFTs.

Details of Rader's algorithm: a theorem from number theory:

$$\forall p \in \mathbb{N} \text{ prime} \quad \exists g \in \{1, \dots, p-1\}: \quad \{g^k \mod p: k = 1, \dots, p-1\} = \{1, \dots, p-1\} ,$$

▶ permutation $P_{p,g} : \{1, \dots, p-1\} \mapsto \{1, \dots, p-1\}$, $P_{p,g}(k) = g^k \mod p$,
reversing permutation $P_k : \{1, \dots, k\} \mapsto \{1, \dots, k\}$, $P_k(i) = k - i + 1$.

For Fourier matrix $\mathbf{F} = (f_{ij})_{i,j=1}^{p}$: $P_{p-1}P_{p,g}(f_{ij})_{i,j=2}^{p}P_{p,g}^{T}$ 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 \begin{array}{c|cccccccccccc}
\omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \hline
\omega^0 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 \\
\omega^0 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 \\
\omega^0 & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} \\
\omega^0 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 \\
\omega^0 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 \\
\omega^0 & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} \\
\omega^0 & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} \\
\omega^0 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 & \omega^6 \\
\omega^0 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 & \omega^3 \\
\omega^0 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 & \omega^8 \\
\omega^0 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2 & \omega^4 \\
\omega^0 & \omega^4 & \omega^8 & \omega^3 & \omega^6 & \omega^{12} & \omega^{11} & \omega^9 & \omega^5 & \omega^{10} & \omega^7 & \omega^1 & \omega^2
\end{array}$$

Then apply fast (FFT based!) algorithms for multiplication with circulant matrices to right lower $(n-1) \times (n-1)$ block of permuted Fourier matrix .

△

Asymptotic complexity of `c=fft(y)` for $\mathbf{y} \in \mathbb{C}^n$ = $O(n \log n)$.

▼ ← Sect. 7.2.1

Asymptotic complexity of discrete periodic convolution/multiplication with circulant matrix, see Code 7.2.6:

Cost(`z = pconvfft(u,x)`, $\mathbf{u}, \mathbf{x} \in \mathbb{C}^n$) = $O(n \log n)$.

Asymptotic complexity of discrete convolution, see Code 7.2.8:

Cost(`z = myconv(h,x)`, $\mathbf{h}, \mathbf{x} \in \mathbb{C}^n$) = $O(n \log n)$.

7.4 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.

7.4.1 Sine transform

Another trigonometric basis transform in $\mathbb{R}^{n-1}$, $n \in \mathbb{N}$:

Standard basis of $\mathbb{R}^{n-1}$ "Sine basis"

$$\left\{ \begin{pmatrix}1\\0\\ \vdots \\ \\ \vdots \\ 0\end{pmatrix} \begin{pmatrix}0\\1\\0\\ \vdots \\ \vdots \\ 0\end{pmatrix} \cdots \begin{pmatrix}0\\ \vdots \\ \vdots \\ 0\\1\\0\end{pmatrix} \begin{pmatrix}0\\ \vdots \\ \\ \vdots \\ 0\\1\end{pmatrix} \right\} \longleftarrow \left\{ \begin{pmatrix}\sin(\frac{\pi}{n})\\ \sin(\frac{2\pi}{n})\\ \vdots \\ \\ \vdots \\ \sin(\frac{(n-1)\pi}{n})\end{pmatrix} \begin{pmatrix}\sin(\frac{2\pi}{n})\\ \sin(\frac{4\pi}{n})\\ \vdots \\ \\ \vdots \\ \sin(\frac{2(n-1)\pi}{n})\end{pmatrix} \cdots \begin{pmatrix}\sin(\frac{(n-1)\pi}{n})\\ \\ \sin(\frac{2(n-1)\pi}{n})\\ \vdots \\ \vdots \\ \sin(\frac{(n-1)^2\pi}{n})\end{pmatrix} \right\}$$

Basis transform matrix (sine basis $\rightarrow$ standard basis): $\mathbf{S}_n := (\sin(jk\pi/n))_{j,k=1}^{n-1} \in \mathbb{R}^{n-1,n-1}$.

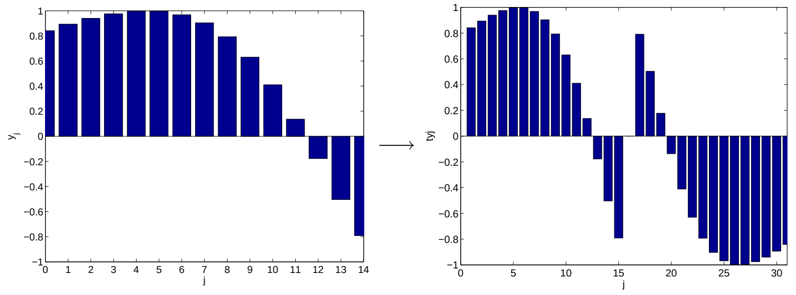
Lemma 7.4.1 (Properties of the sine matrix)**.**

$\sqrt{2/n}\,\mathbf{S}_n$ is real, symmetric and orthogonal ($\rightarrow$ Def. 2.8.1)

Sine transform: $\boxed{s_k = \sum_{j=1}^{n-1} y_j \sin(\pi jk/n)}$, $k = 1,\dots,n-1$. (7.4.1)

DFT-based algorithm for the sine transform ($\hat{=}\ \mathbf{S}_n\times$vector):

"wrap around": $\tilde{\mathbf{y}} \in \mathbb{R}^{2n}$: $\tilde{y}_j = \begin{cases} y_j & \text{, if } j = 1,\dots,n-1\ , \\ 0 & \text{, if } j = 0, n\ , \\ -y_{2n-j} & \text{, if } j = n+1,\dots,2n-1\ . \end{cases}$ ($\tilde{\mathbf{y}}$ "odd")



$$\begin{aligned}(\mathbf{F}_{2n}\tilde{\mathbf{y}})_k &\overset{(7.2.8)}{=} \sum_{j=1}^{2n-1} \tilde{y}_j e^{-\frac{2\pi}{2n}kj} \\ &= \sum_{j=1}^{n-1} y_j e^{-\frac{\pi}{n}kj} - \sum_{j=n+1}^{2n-1} y_{2n-j} e^{-\frac{\pi}{n}kj} \\ &= \sum_{j=1}^{n-1} y_j (e^{-\frac{\pi}{n}kj} - e^{\frac{\pi}{n}kj}) \\ &= -2i\,(\mathbf{S}_n\mathbf{y})_k \quad ,k = 1,\dots,n-1\ .\end{aligned}$$

Wrap-around implementation

```
function c = sinetrans(y)
n = length(y)+1;
yt = [0,y,0,-y(end:-1:1)];
ct = fft(yt);
c = -ct(2:n)/(2*i);
```

MATLAB-CODE sine transform

Remark 7.4.1 (Sine transform via DFT of half length).

Step ①: transform of the coefficients

$$\tilde{y}_j = \sin(j\pi/n)(y_j + y_{n-j}) + \tfrac{1}{2}(y_j - y_{n-j})\ , \quad j = 1,\dots,n-1 \quad , \quad \tilde{y}_0 = 0\ .$$

Step ②: real DFT ($\rightarrow$ Sect. 7.2.3) of $(\tilde{y}_0,\dots,\tilde{y}_{n-1}) \in \mathbb{R}^n$: $c_k := \sum_{j=0}^{n-1} \tilde{y}_j\, e^{-\frac{2\pi i}{n}jk}$

Hence

$$\begin{aligned}\mathrm{Re}\{c_k\} &= \sum_{j=0}^{n-1} \tilde{y}_j \cos(-\tfrac{2\pi i}{n}jk) = \sum_{j=1}^{n-1} (y_j + y_{n-j}) \sin(\tfrac{\pi j}{n}) \cos(\tfrac{2\pi i}{n}jk) \\ &= \sum_{j=0}^{n-1} 2y_j \sin(\tfrac{\pi j}{n}) \cos(\tfrac{2\pi i}{n}jk) = \sum_{j=0}^{n-1} y_j \left(\sin(\frac{2k+1}{n}\pi j) - \sin(\frac{2k-1}{n}\pi j)\right) \\ &= s_{2k+1} - s_{2k-1}\ . \\ \mathrm{Im}\{c_k\} &= \sum_{j=0}^{n-1} \tilde{y}_j \sin(-\tfrac{2\pi i}{n}jk) = -\sum_{j=1}^{n-1} \tfrac{1}{2}(y_j - y_{n-j}) \sin(\tfrac{2\pi i}{n}jk) = -\sum_{j=1}^{n-1} y_j \sin(\tfrac{2\pi i}{n}jk) \\ &= -s_{2k}\ .\end{aligned}$$

Step ③: extraction of s_k

s_{2k+1} , $k = 0,\dots,\frac{n}{2}-1$ ➤ from recursion $s_{2k+1} - s_{2k-1} = \mathrm{Re}\{c_k\}$, $s_1 = \sum_{j=1}^{n-1} y_j \sin(\pi j/n)$,

s_{2k}, $k = 1,\dots,\frac{n}{2}-2$ ➤ $s_{2k} = -\mathrm{Im}\{c_k\}$.

MATLAB-Implementation (via a `fft` of length $n/2$):

MATLAB-CODE Sine transform

```
function s = sinetrans(y)
n = length(y)+1;
sinevals = imag(exp(i*pi/n).^(1:n-1));
yt = [0 (sinevals.*(y+y(end:-1:1)) + 0.5*(y-y(end:-1:1)))];
c = fftreal(yt);
s(1) = dot(sinevals,y);
for k=2:N-1
if (mod(k,2) == 0), s(k) = -imag(c(k/2+1));
else, s(k) = s(k-2) + real(c((k-1)/2+1)); end
end
```

△

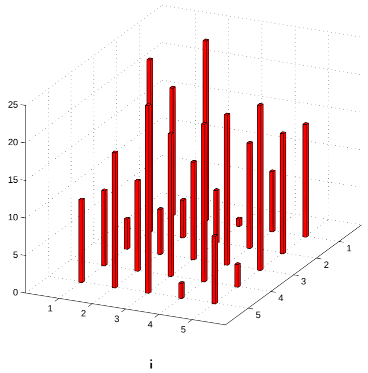
Application: diagonalization of local translation invariant linear operators.

5-points-stencil-operator on $\mathbb{R}^{n,n}$, $n \in \mathbb{N}$, in grid representation:

$$T : \mathbb{R}^{n,n} \mapsto \mathbb{R}^{n,n}, \quad \begin{array}{c} \mathbf{X} \mapsto T(\mathbf{X}) \\ (T(\mathbf{X}))_{ij} := c x_{ij} + c_y x_{i,j+1} + c_y x_{i,j-1} + c_x x_{i+1,j} + c_x x_{i-1,j} \end{array}$$

with $c, c_y, c_x \in \mathbb{R}$, convention: $x_{ij} := 0$ for $(i,j) \notin \{1,\dots,n\}^2$.

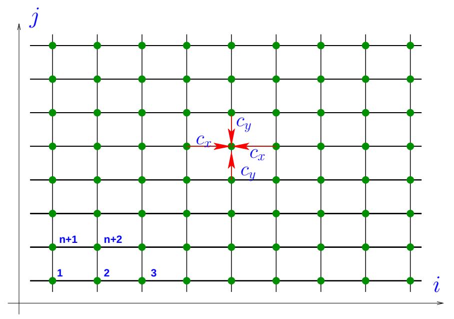
$$\mathbf{X} \in \mathbb{R}^{n,n}$$
$$\updownarrow$$
grid function $\in \{1,\dots,n\}^2 \mapsto \mathbb{R}$



Identification $\mathbb{R}^{n,n} \cong \mathbb{R}^{n^2}$, $x_{ij} \sim \widetilde{x}_{(j-1)n+i}$ gives matrix representation $\mathbf{T} \in \mathbb{R}^{n^2,n^2}$ of T:

$$\mathbf{T} = \begin{pmatrix} \mathbf{C} & c_y\mathbf{I} & 0 & \cdots & \cdots & 0 \\ c_y\mathbf{I} & \mathbf{C} & c_y\mathbf{I} & & & \vdots \\ 0 & \ddots & \ddots & \ddots & & \\ \vdots & & & c_y\mathbf{I} & \mathbf{C} & c_y\mathbf{I} \\ 0 & \cdots & \cdots & 0 & c_y\mathbf{I} & \mathbf{C} \end{pmatrix} \in \mathbb{R}^{n^2,n^2},$$

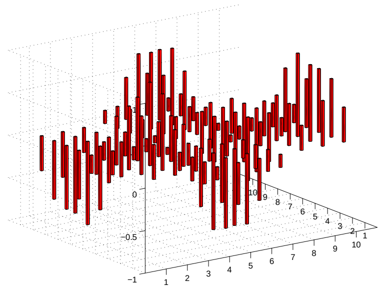
$$\mathbf{C} = \begin{pmatrix} c & c_x & 0 & \cdots & \cdots & 0 \\ c_x & c & c_x & & & \vdots \\ 0 & \ddots & \ddots & \ddots & & \\ \vdots & & & c_x & c & c_x \\ 0 & \cdots & \cdots & 0 & c_x & c \end{pmatrix} \in \mathbb{R}^{n,n}.$$



Sine basis of $\mathbb{R}^{n,n}$:

$$\mathbf{B}^{kl} = \left(\sin(\tfrac{\pi}{n+1}ki)\sin(\tfrac{\pi}{n+1}lj)\right)_{i,j=1}^{n}. \qquad (7.4.2)$$

$n = 10$: grid function $\mathbf{B}^{2,3}$ ➢



$$\begin{aligned}(T(\mathbf{B}^{kl}))_{ij} &= c\sin(\tfrac{\pi}{n}ki)\sin(\tfrac{\pi}{n}lj) + c_y\sin(\tfrac{\pi}{n}ki)\left(\sin(\tfrac{\pi}{n+1}l(j-1)) + \sin(\tfrac{\pi}{n+1}l(j+1))\right) + \\ &\qquad c_x\sin(\tfrac{\pi}{n}lj)\left(\sin(\tfrac{\pi}{n+1}k(i-1)) + \sin(\tfrac{\pi}{n+1}k(i+1))\right) \\ &= \sin(\tfrac{\pi}{n}ki)\sin(\tfrac{\pi}{n}lj)(c + 2c_y\cos(\tfrac{\pi}{n+1}l) + 2c_x\cos(\tfrac{\pi}{n+1}k))\end{aligned}$$

Hence $\mathbf{B}^{kl}$ is eigenvector of $T \leftrightarrow \mathbf{T}$ corresponding to eigenvalue $c + 2c_y\cos(\frac{\pi}{n+1}l) + 2c_x\cos(\frac{\pi}{n+1}k)$.

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. 7.2.4) for rows/columns of $\mathbf{Y} = (y_{kl})_{k,l=1}^{n}$.

Here: implementation of sine transform (7.4.1) with "wrapping"-technique.

MATLAB-CODE two dimensional sine tr.

```
function C = sinft2d(Y)
[m,n] = size(Y);
C = fft([zeros(1,n); Y;...
        zeros(1,n);...
        -Y(end:-1:1,:)]);
C = i*C(2:m+1,:)'/2;
C = fft([zeros(1,m); C;...
        zeros(1,m);...
        -C(end:-1:1,:)]);
C= i*C(2:n+1,:)'/2;
```

MATLAB-CODE FFT-based solution of local translation invariant linear operators

```
function X = fftsolve(B,c,cx,cy)
[m,n] = size(B);
[I,J] = meshgrid(1:m,1:n);
X = 4*sinft2d(sinft2d(B)...
    ./(c+2*cx*cos(pi/(n+1)*I)+...
         2*cy*cos(pi/(m+1)*J)))...
     /((m+1)*(n+1));
```

Diagonalization of $\mathbf{T}$
via 2D sine transform

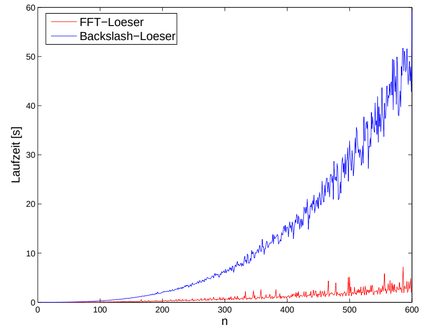
$\blacktriangledown$

efficient algorithm
for solving linear system of equations $T(\mathbf{X}) = \mathbf{B}$
computational cost $O(n^2 \log n)$.

Example 7.4.2 (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;
```



$\diamond$

7.4.2 Cosine transform

Another trigonometric basis transform in $\mathbb{R}^n$, $n \in \mathbb{N}$:

standard basis of $\mathbb{R}^n$ "cosine basis"

$$\left\{ \begin{pmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \cdots \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ 1 \end{pmatrix} \right\} \longleftarrow \left\{ \begin{pmatrix} 2^{-1/2} \\ \cos(\frac{\pi}{2n}) \\ \cos(\frac{2\pi}{2n}) \\ \vdots \\ \vdots \\ \cos(\frac{(n-1)\pi}{2n}) \end{pmatrix} \begin{pmatrix} 2^{-1/2} \\ \cos(\frac{3\pi}{2n}) \\ \cos(\frac{6\pi}{2n}) \\ \vdots \\ \vdots \\ \cos(\frac{3(n-1)\pi}{2n}) \end{pmatrix} \cdots \begin{pmatrix} 2^{-1/2} \\ \cos(\frac{(2n-1)\pi}{2n}) \\ \cos(\frac{2(2n-1)\pi}{2n}) \\ \vdots \\ \vdots \\ \cos(\frac{(n-1)(2n-1)\pi}{2n}) \end{pmatrix} \right\}$$

Basis transform matrix (cosine basis $\rightarrow$ standard basis):

$$\mathbf{C}_n = (c_{ij}) \in \mathbb{R}^{n,n} \quad \text{with} \quad c_{ij} = \begin{cases} 2^{-1/2} & \text{, if } i = 1 \,, \\ \cos((i-1)\frac{2j-1}{2n}\pi) & \text{, if } i > 1 \,. \end{cases}$$

Lemma 7.4.2 (Properties of cosine matrix).

$\sqrt{2/n}\,\mathbf{C}_n$ is real and orthogonal ($\rightarrow$ Def. 2.8.1).

Note: $\mathbf{C}_n$ is not symmetric

$$\text{cosine transform:} \quad \boxed{c_k = \sum_{j=0}^{n-1} y_j \cos(k\frac{2j+1}{2n}\pi)} \quad , \quad k = 1, \ldots, n-1 \,, \tag{7.4.3}$$

$$c_0 = \frac{1}{\sqrt{2}} \sum_{j=0}^{n-1} y_j \,.$$

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):

MATLAB-CODE : Inverse cosine transform

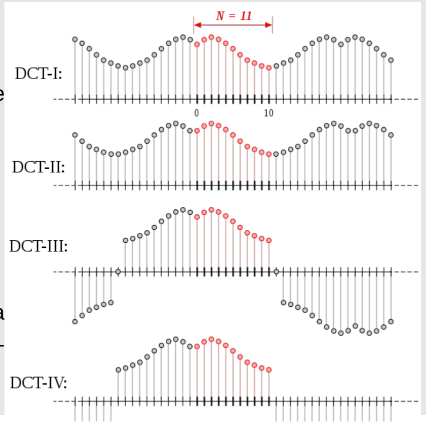
```
function y=icostrans(c)
n = length(c);
y = [sqrt(2)*c(1),(exp(i*pi/(2*n)).^(1:n-1)).*c(2:end)];
y = ifft([y,0,conj(y(end:-1:2))]);
y = 2*y(1:n);
```

Remark 7.4.3 (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.



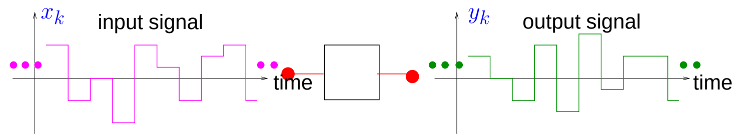
△

7.5 Toeplitz matrix techniques

Example 7.5.1 (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. 7.1.1.
- Known: impulse response of filter has maximal duration $n\Delta t$, $n\in\mathbb{N}$, $n\leq m$

cf. (7.1.1) ▶ $\exists\mathbf{h}=(h_0,\dots,h_{n-1})^T\in\mathbb{R}^n,\quad n\leq m:\ y_k=\sum_{j=0}^{n-1}h_jx_{k-j}\,.$ (7.5.1)



Parameter identification problem: seek $\mathbf{h}=(h_0,\dots,h_{n-1})^T\in\mathbb{R}^n$ with

$$\|\mathbf{A}\mathbf{h}-\mathbf{y}\|_2=\left\|\begin{pmatrix} x_0 & x_{-1} & \cdots & \cdots & x_{1-n}\\ x_1 & x_0 & x_{-1} & & \vdots\\ \vdots & x_1 & x_0 & \ddots & \\ & & \ddots & \ddots & \vdots\\ \vdots & & & \ddots & x_{-1}\\ x_{n-1} & & & x_1 & x_0\\ x_n & x_{n-1} & & & x_1\\ \vdots & & & & \vdots\\ x_{m-1} & \cdots & & \cdots & x_{m-n}\end{pmatrix}\begin{pmatrix}h_0\\ \vdots\\ \vdots\\ h_{n-1}\end{pmatrix}-\begin{pmatrix}y_0\\ \vdots\\ \\ \vdots\\ y_{m-1}\end{pmatrix}\right\|_2\rightarrow\min\,.$$

➣ Linear least squares problem, → Ch. 6 with Toeplitz matrix $\mathbf{A}$: $(\mathbf{A})_{ij}=x_{i-j}$.

System matrix of normal equations (→ Sect. 6.1)

$$\mathbf{M}:=\mathbf{A}^H\mathbf{A}\quad,\quad(\mathbf{M})_{ij}=\sum_{k=1}^{m}x_{k-i}x_{k-j}=z_{i-j}\quad\text{due to periodicity of }(x_k)_{k\in\mathbb{Z}}\,.$$

➣ $\mathbf{M}\in\mathbb{R}^{n,n}$ is a *matrix with constant diagonals* & s.p.d.
(“constant diagonals” $\Leftrightarrow$ $(\mathbf{M})_{i,j}$ depends only on $i-j$)

◇

Example 7.5.2 (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

Expectation $\mathcal{E}(\mathsf{Y}_{i-j}\mathsf{Y}_{i-k})=u_{k-j}\quad\forall i,j,k\in\mathbb{Z}\,,\quad u_i=u_{-i}\,.$

Model: finite linear relationship

$$\exists\mathbf{x}=(x_1,\dots,x_n)^T\in\mathbb{R}^n:\quad\mathsf{Y}_k=\sum_{j=1}^{n}x_j\mathsf{Y}_{k-j}\quad\forall k\in\mathbb{Z}\,.$$

with *unknown* parameters x_j, $j=1,\dots,n$: for fixed $i\in\mathbb{Z}$

$$\text{Estimator}\qquad\mathbf{x}=\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}}\,E\Big|\mathsf{Y}_i-\sum_{j=1}^{n}x_j\mathsf{Y}_{i-j}\Big|^2$$

▶ $$E|\mathsf{Y}_i|^2-2\sum_{j=1}^{n}x_ju_k+\sum_{k,j=1}^{n}x_kx_ju_{k-j}\rightarrow\min\,.$$

▶ $\mathbf{x}^T\mathbf{A}\mathbf{x}-2\mathbf{b}^T\mathbf{x}\rightarrow\min$ with $\mathbf{b}=(u_k)_{k=1}^n\,,\quad\mathbf{A}=(u_{i-j})_{i,j=1}^n\,.$

Lemma 4.1.2 $\Rightarrow$ $\mathbf{x}$ solves $\mathbf{A}\mathbf{x}=\mathbf{b}$ (Yule-Walker-equation, see below)

$\mathbf{A}$ $\hat{=}$ Covariance matrix: s.p.d. matrix with constant diagonals.

◇

Matrices with constant diagonals occur frequently in mathematical models. They generalize of circulant matrices → Def. 7.1.3.

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 7.5.1 (Toeplitz matrix)**.**
$\mathbf{T}=(t_{ij})_{i,j=1}^n\in\mathbb{K}^{m,n}$ is a Toeplitz matrix, if there is a vector $\mathbf{u}=(u_{-m+1},\dots,u_{n-1})\in\mathbb{K}^{m+n-1}$ such that $t_{ij}=u_{j-i}$, $1\leq i\leq m$, $1\leq j\leq n$.

$$\mathbf{T}=\begin{pmatrix} u_0 & u_1 & \cdots & & \cdots & u_{n-1}\\ u_{-1} & u_0 & u_1 & & & \vdots\\ \vdots & & \ddots & \ddots & \ddots & \vdots\\ \vdots & & & \ddots & \ddots & \ddots & \vdots\\ \vdots & & & & \ddots & \ddots & u_1\\ u_{1-m} & \cdots & & & \cdots & u_{-1} & u_0\end{pmatrix}$$

7.5.1 Toeplitz matrix arithmetic

$\mathbf{T}=(u_{j-i})\in\mathbb{K}^{m,n}$ = Toeplitz matrix with generating vector $\mathbf{u}=(u_{-m+1},\dots,u_{n-1})\in\mathbb{C}^{m+n-1}$

Task: efficient evaluation of matrix×vector product $\mathbf{T}\mathbf{x}$, $\mathbf{x} \in \mathbb{K}^n$

Note: this extended matrix is circulant (⟶ Def. 7.1.3)

$$\mathbf{C} = \begin{pmatrix} \mathbf{T} & \mathbf{S} \\ \mathbf{S} & \mathbf{T} \end{pmatrix} = \left(\begin{array}{cccccc|cccccc}
u_0 & u_1 & \cdots & & \cdots & u_{n-1} & 0 & u_{1-n} & \cdots & & \cdots & u_{-1} \\
u_{-1} & u_0 & u_1 & & & \vdots & u_{n-1} & 0 & \ddots & & & \vdots \\
\vdots & \ddots & \ddots & \ddots & & \vdots & \vdots & \ddots & \ddots & & & \\
\vdots & & \ddots & \ddots & \ddots & \vdots & & & & & \ddots & \\
\vdots & & & \ddots & \ddots & u_1 & \vdots & & & \ddots & \ddots & u_{1-n} \\
u_{1-n} & \cdots & & \cdots & u_{-1} & u_0 & u_1 & & & & u_{n-1} & 0 \\
\hline
0 & u_{1-n} & \cdots & & \cdots & u_{-1} & u_0 & u_1 & \cdots & & \cdots & u_{n-1} \\
u_{n-1} & 0 & \ddots & & & \vdots & u_{-1} & u_0 & u_1 & & & \vdots \\
\vdots & \ddots & \ddots & & & & \vdots & \ddots & \ddots & \ddots & & \vdots \\
 & & & & \ddots & & \vdots & & \ddots & \ddots & \ddots & \vdots \\
\vdots & & & \ddots & \ddots & u_{1-n} & \vdots & & & \ddots & \ddots & u_1 \\
u_1 & & & & u_{n-1} & 0 & u_{1-n} & \cdots & & \cdots & u_{-1} & u_0
\end{array}\right)$$

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)]);
```

$$\blacktriangleright \quad \mathbf{C}\begin{pmatrix} \mathbf{x} \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{T}\mathbf{x} \\ \mathbf{S}\mathbf{x} \end{pmatrix}$$

$\blacktriangleright$ Computational effort $O(n \log n)$ for computing $\mathbf{T}\mathbf{x}$ (FFT based, Sect. 7.3)

7.5.2 The Levinson algorithm

Given: S.p.d. Toeplitz matrix $\mathbf{T} = (u_{j-i})_{i,j=1}^n$, generating vector $\mathbf{u} = (u_{-n+1}, \dots, u_{n-1}) \in \mathbb{C}^{2n-1}$
(Symmetry ↔ $u_{-k} = u_k$, w.l.o.g $u_0 = 1$)

Task: efficient solution algorithm for LSE $\mathbf{T}\mathbf{x} = \mathbf{b}$, $\mathbf{b} \in \mathbb{C}^n$
(Yule-Walker problem)

Recursive (inductive) solution strategy:

Define:
- $\mathbf{T}_k := (u_{j-i})_{i,j=1}^k \in \mathbb{K}^{k,k}$ (left upper block of $\mathbf{T}$) ➢ $\boxed{\mathbf{T}_k \text{ is s.p.d. Toeplitz matrix}}$,
- $\mathbf{x}^k \in \mathbb{K}^k$: $\mathbf{T}_k\mathbf{x}^k = (b_1, \dots, b_k)^T \quad \Leftrightarrow \quad \mathbf{x}^k = \mathbf{T}_k^{-1}\mathbf{b}^k$,
- $\mathbf{u}^k := (u_1, \dots, u_k)^T$

Block-partitioned LSE, *cf.* Rem. 2.1.4, Rem. 2.2.8

$$\mathbf{T}_{k+1}\mathbf{x}^{k+1} = \left(\begin{array}{ccc|c} & & & u_k \\ & \mathbf{T}_k & & \vdots \\ & & & u_1 \\ \hline u_k & \cdots & u_1 & 1 \end{array}\right) \left(\begin{array}{c} \widetilde{\mathbf{x}}^{k+1} \\ \hline x_{k+1}^{k+1} \end{array}\right) = \left(\begin{array}{c} b_1 \\ \vdots \\ b_k \\ \hline b_{k+1} \end{array}\right) = \begin{pmatrix} \widetilde{\mathbf{b}}^{k+1} \\ b_{k+1} \end{pmatrix} \tag{7.5.2}$$

Reversing permutation: $P_k : \{1, \dots, k\} \mapsto \{1, \dots, k\}$, $P_k(i) := k - i + 1$

$$\blacktriangleright \quad \begin{aligned} \widetilde{\mathbf{x}}_{k+1} &= \mathbf{T}_k^{-1}(\widetilde{\mathbf{b}}^{k+1} - x_{k+1}^{k+1}P_k\mathbf{u}^k) = \mathbf{x}^k - x_{k+1}^{k+1}\mathbf{T}_k^{-1}P_k\mathbf{u}^k \,, \\ x_{k+1}^{k+1} &= b_{k+1} - P_k\mathbf{u}^k \cdot \widetilde{\mathbf{x}}^{k+1} = b_{k+1} - P_k \cdot \mathbf{x}^k + x_{k+1}^{k+1}P_k \cdot \mathbf{T}_k^{-1}P_k\mathbf{u}^k \,. \end{aligned} \tag{7.5.3}$$

Efficient algorithm by using *auxiliary vectors*: $\mathbf{y}^k := \mathbf{T}_k^{-1}P_k\mathbf{u}^k$

$$\mathbf{x}^{k+1} = \begin{pmatrix} \widetilde{\mathbf{x}}^{k+1} \\ x_{k+1}^{k+1} \end{pmatrix} \quad \text{with} \quad \begin{aligned} x_{k+1}^{k+1} &= (b_{k+1} - P_k\mathbf{u}^k)/\sigma_k \\ \widetilde{\mathbf{x}}^{k+1} &= \mathbf{x}^k - x_{k+1}^{k+1}\mathbf{y}^k \end{aligned} \quad , \quad \sigma_k := 1 - P_k\mathbf{u}^k \cdot \mathbf{y}^k \,. \tag{7.5.4}$$

Levinson algorithm ▷
(recursive, u_{n+1} not used!)

Linear recursion:
Computational cost $\sim (n-k)$ on level k, $k = 0, \dots, n-1$

➢ Asymptotic complexity $O(n^2)$

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 7.5.3 (Fast Toeplitz solvers).

FFT-based algorithms for solving $\mathbf{T}\mathbf{x} = \mathbf{b}$ with asymptotic complexity $O(n \log^3 n)$ [37] ! △

Supplementary and further reading:

[8, 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.

[23, 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.

[35, 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, [14, 4]. Issues and technical details way beyond the scope of the course are treated there.

Part II

Interpolation and Approximation

Introduction

Distinguish two fundamental concepts:

(I) **data interpolation** (point interpolation, also includes CAD applications):

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

$$\boxed{f(\mathbf{x}_i) = \mathbf{y}_i, \quad i = 1, \ldots, m}$$

Additional requirements:
- smoothness of $\mathbf{f}$, e.g. $\mathbf{f} \in C^1$, etc.
- shape of $\mathbf{f}$ (positivity, monotonicity, convexity)

(II) **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"$^{(♣)}$

$^{(*)}$: "simple" $\sim$ described by small amount of information, easy to evaluate (e.g, polynomial or piecewise polynomial $\widetilde{\mathbf{f}}$)

$^{(♣)}$ "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 $\|\mathbf{g}\|_2^2 := \int_D |\mathbf{g}(x)|^2 dx$, maximum norm $\|\mathbf{g}\|_\infty := \max_{x \in D} |\mathbf{g}(x)|$

Example 7.0.1 (Taylor approximation).

$$f \in C^k(I), \quad I \text{ interval}, \quad k \in \mathbb{N}, \qquad T_k(t) := \frac{f^{(k)}(t_0)}{k!}(t - t_0)^k, \quad t_0 \in I\ .$$

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 gives the same accuracy.

◇

Another technique: Approximation by interpolation

$$\mathbf{f} \xrightarrow{\text{sampling}} (\mathbf{x}_i, \mathbf{y}_i := f(\mathbf{x}_i))_{i=1}^m \xrightarrow{\text{interpolation}} \widetilde{\mathbf{f}}: \quad \widetilde{\mathbf{f}}(\mathbf{x}_i) = \mathbf{y}_i\ .$$

↑
free choice of nodes $\mathbf{x}_i$

Remark 7.0.2 (Interpolation and approximation: enabling technologies).

Approximation and interpolation are useful for several numerical tasks, like integration, differentiation and computation of the solutions of differential equations, as well as for computer graphics: smooth curves and surfaces.

➤ this is a “foundations” part of the course △

Remark 7.0.3 (Function representation).

! General function $f : D \subset \mathbb{R} \mapsto \mathbb{K}$, D interval, contains an “infinite amount of information”.

? How to represent f on a computer?

➜ Idea: parametrization, a finite number of parameters $\alpha_1, \dots, \alpha_n$, $n \in \mathbb{N}$, characterizes f.

Special case: Representation with finite linear combination of basis functions $b_j : D \subset \mathbb{R} \mapsto \mathbb{K}$, $j = 1, \dots, n$:

$$f = \sum_{j=1}^{n} \alpha_j b_j \quad , \quad \alpha_j \in \mathbb{K} .$$

➜ $f \in$ finite dimensional function space $V_n := \mathrm{Span}\,\{b_1, \dots, b_n\}$. △

8 Polynomial Interpolation

8.1 Polynomials

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

$$\mathcal{P}_k := \{t \mapsto \alpha_k t^k + \alpha_{k-1} t^{k-1} + \dots + \alpha_1 t + \alpha_0 \, , \, \alpha_j \in \mathbb{K}\} . \tag{8.1.1}$$

Terminology: the functions $t \mapsto t^k$, $k \in \mathbb{N}_0$, are called monomials

$t \mapsto \alpha_k t^k + \alpha_{k-1} t^{k-1} + \dots + \alpha_0$ = monomial representation of a polynomial.

Obvious: $\mathcal{P}_k$ is a vector space. What is its dimension?

Theorem 8.1.1 (Dimension of space of polynomials)**.**
$\dim \mathcal{P}_k = k + 1$ *and* $\mathcal{P}_k \subset C^\infty(\mathbb{R})$.

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 8.1.1 (Polynomials in Matlab).

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

Remark 8.1.2 (Horner scheme).

Evaluation of a polynomial in monomial representation: Horner scheme

$$p(t) = (t \cdots t(t(\alpha_n t + \alpha_{n-1}) + \alpha_{n-2}) + \dots + \alpha_1) + \alpha_0 . \tag{8.1.2}$$

Code 8.1.3: Horner scheme, polynomial in MATLAB format

```
function y = polyval(p,x)
y = p(1); for i=2:length(p), y = x*y+p(i); end
```

Asymptotic complexity: $O(n)$

Use: MATLAB "built-in"-function polyval(p,x);. $\triangle$

8.2 Polynomial Interpolation: Theory

Goal: (re-)construction of a function from pairs of values (fit).

Lagrange polynomial interpolation problem

Given the simple nodes $t_0, \dots, t_n$, $n \in \mathbb{N}$, $-\infty < t_0 < t_1 < \dots < t_n < \infty$ and the values $y_0, \dots, y_n \in \mathbb{K}$ compute $p \in \mathcal{P}_n$ such that

$$p(t_j) = y_j \quad \text{for} \quad j = 0, \dots, n\,. \tag{8.2.1}$$

General polynomial interpolation problem

Given the (eventually multiple) nodes $t_0, \dots, t_n$, $n \in \mathbb{N}$, $-\infty < t_0 \le t_1 \le \dots \le t_n < \infty$ and the values $y_0, \dots, y_n \in \mathbb{K}$ compute $p \in \mathcal{P}_n$ such that

$$\frac{d^k}{dt^k} p(t_j) = y_j \quad \text{for} \quad k = 0, \dots, l_j \quad \text{and} \quad j = 0, \dots, n\,, \tag{8.2.2}$$

where $l_j := \max\{i - i' \colon t_j = t_i = t_{i'},\ i, i' = 0, \dots, n\}$ is the multiplicity of the nodes t_j.

When all the multiplicities are equal to 2: Hermite interpolation (or osculatory interpolation)
$t_0 = t_1 < t_2 = t_3 < \dots < t_{n-1} = t_n$ $\rightarrow$ $p(t_{2j}) = y_{2j}, p'(t_{2j}) = y_{2j+1}$, (double nodes).

8.2.1 Lagrange polynomials

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

$$\text{Lagrange polynomials} \quad L_i(t) := \prod_{\substack{j=0\\ j \ne i}}^{n} \frac{t - t_j}{t_i - t_j}\,, \quad i = 0, \dots, n\,. \tag{8.2.3}$$

$\rightarrow$ $L_i \in \mathcal{P}_n$ 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 8.2.1. Lagrange polynomials for uniformly spaced nodes

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

Plot $n = 10$, $j = 0, 2, 5$ $\rightarrow$

$\diamond$

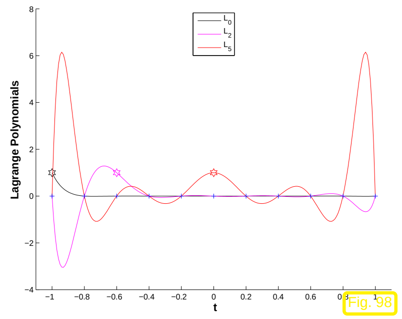


Fig. 98

The Lagrange interpolation polynomial p for data $(t_i, y_i)_{i=0}^n$ has the representation:

$$p(t) = \sum_{i=0}^{n} y_i L_i(t)\,, \quad \Rightarrow \quad p \in \mathcal{P}_n \quad \text{and} \quad p(t_i) = y_i\,. \tag{8.2.4}$$

Theorem 8.2.1 (Existence & uniqueness of Lagrange interpolation polynomial)**.**
The general polynomial interpolation problem (8.2.1) *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. 8.1.1.

Representation (8.2.4) $\Rightarrow$ existence of interpolating polynomial
$\Rightarrow$ $\mathrm{eval}_{\mathcal{T}}$ is surjective

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 surjective $\Leftrightarrow$ T bijective $\Leftrightarrow$ T injective.

Applying this equivalence to $\mathrm{eval}_{\mathcal{T}}$ yields the assertion of the theorem □

Lagrangian polynomial interpolation leads to linear systems of equations:

$$p(t_j) = y_j \quad \Longleftrightarrow \quad \sum_{i=0}^{n} a_i t_j^i = y_j,\ j = 0,\dots,n$$

$\Longleftrightarrow$ solution of $(n+1)\times(n+1)$ linear system $\mathbf{Va} = \mathbf{y}$ with matrix

$$\mathbf{V} = \begin{pmatrix} 1 & t_0 & t_0^2 & \cdots & t_0^n \\ 1 & t_1 & t_1^2 & \cdots & t_1^n \\ 1 & t_2 & t_2^2 & \cdots & t_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^n \end{pmatrix} . \tag{8.2.5}$$

Existence of a solution for a square system gives uniqueness. □

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;
```
or
```
for j = 1 : length(t);  V(:,j) = t.^(j − 1);  end;
```

Theorem 8.2.2 (Lagrange interpolation as linear mapping)**.**
The polynomial interpolation in the nodes $\mathcal{T} := \{t_j\}_{j=0}^{n}$ *defines a linear operator*

$$\mathsf{I}_{\mathcal{T}} : \begin{cases} \mathbb{K}^{n+1} & \to \mathcal{P}_n , \\ (y_0,\dots,y_n)^T & \mapsto \text{interpolating polynomial } p . \end{cases} \tag{8.2.6}$$

Remark 8.2.2 (Matrix representation of interpolation operator).

In the case of Lagrange interpolation:

- if Lagrange polynomials are chosen as basis for $\mathcal{P}_n$, $\rightarrow$ $\mathsf{I}_{\mathcal{T}}$ is represented by the identity matrix;
- 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 (8.2.5).

△

Definition 8.2.3 (Generalized Lagrange polynomials)**.**
The generalized Lagrange polynomials *on the nodes* $\mathcal{T} = \{t_j\}_{j=0}^{n} \subset \mathbb{R}$ *are* $L_i := \mathsf{I}_{\mathcal{T}}(\mathbf{e}_{i+1})$, $i = 0,\dots,n$, *where* $\mathbf{e}_i = (0,\dots,0,1,0,\dots,0)^T \in \mathbb{R}^{n+1}$ *are the unit vectors.*

Theorem 8.2.4 (Existence & uniqueness of generalized Lagrange interpolation polynomials)**.**
The general polynomial interpolation problem (8.2.2) *admits a unique solution* $p \in \mathcal{P}_n$.

Example 8.2.3. (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 $\longrightarrow$ see (9.3.2). ◇

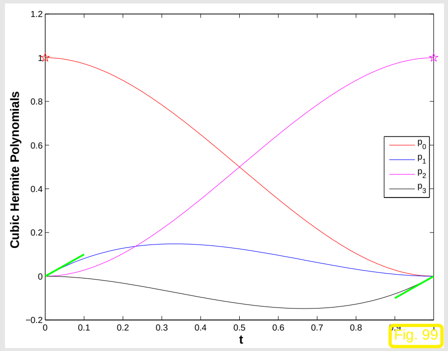


Fig. 99

8.2.2 Conditioning of polynomial interpolation

Necessary for studying the conditioning: norms on vector space of continuous functions $C(I)$, $I \subset \mathbb{R}$

$$\text{supremum norm} \quad \|f\|_{L^\infty(I)} := \sup\{|f(t)|: t \in I\}\,, \tag{8.2.7}$$

$$L^2\text{-norm} \quad \|f\|^2_{L^2(I)} := \int_I |f(t)|^2\,\mathrm{d}t\,, \tag{8.2.8}$$

$$L^1\text{-norm} \quad \|f\|_{L^1(I)} := \int_I |f(t)|\,\mathrm{d}t\,. \tag{8.2.9}$$

Lemma 8.2.5 (Absolute conditioning of polynomial interpolation)**.**
Given a mesh $\mathcal{T} \subset \mathbb{R}$ with generalized Lagrange polynomials L_i, $i = 0,\dots,n$, and fixed $I \subset \mathbb{R}$, the norm of the interpolation operator satisfies

$$\|\mathsf{I}_\mathcal{T}\|_{\infty\to\infty} := \sup_{\mathbf{y}\in\mathbb{K}^{n+1}\setminus\{0\}} \frac{\|\mathsf{I}_\mathcal{T}(\mathbf{y})\|_{L^\infty(I)}}{\|\mathbf{y}\|_\infty} = \left\|\sum_{i=0}^n |L_i|\right\|_{L^\infty(I)}\,, \tag{8.2.10}$$

$$\|\mathsf{I}_\mathcal{T}\|_{2\to2} := \sup_{\mathbf{y}\in\mathbb{K}^{n+1}\setminus\{0\}} \frac{\|\mathsf{I}_\mathcal{T}(\mathbf{y})\|_{L^2(I)}}{\|\mathbf{y}\|_2} \leq \left(\sum_{i=0}^n \|L_i\|^2_{L^2(I)}\right)^{\frac12}\,. \tag{8.2.11}$$

Proof. (for the L^∞-Norm) By $\triangle$-inequality

$$\|\mathsf{I}_\mathcal{T}(\mathbf{y})\|_{L^\infty(I)} = \left\|\sum_{j=0}^n y_j L_j\right\|_{L^\infty(I)} \leq \sup_{t\in I} \sum_{j=0}^n |y_j||L_j(t)| \leq \|\mathbf{y}\|_\infty \left\|\sum_{i=0}^n |L_i|\right\|_{L^\infty(I)}\,,$$

equality in (8.2.10) for $\mathbf{y} := (\mathrm{sgn}(L_j(t^*)))_{j=0}^n$, $t^* := \mathrm{argmax}_{t\in I} \sum_{i=0}^n |L_i(t)|$. □

Proof. (for the L^2-Norm) By $\triangle$-inequality and Cauchy-Schwarz inequality

$$\|\mathsf{I}_\mathcal{T}(\mathbf{y})\|_{L^2(I)} \leq \sum_{j=0}^n |y_j|\,\|L_j\|_{L^2(I)} \leq \left(\sum_{j=0}^n |y_j|^2\right)^{\frac12} \left(\sum_{j=0}^n \|L_j\|^2_{L^2(I)}\right)^{\frac12}\,. \quad □$$

Terminology: Lebesgue constant of $\mathcal{T}$: $\lambda_\mathcal{T} := \left\|\sum_{i=0}^n |L_i|\right\|_{L^\infty(I)}$

Example 8.2.4 (Computation of the Lebesgue constant).

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

Asymptotic estimate (with (8.2.3) and Stirling formula): for $n = 2m$

$$|L_m(1-\tfrac1n)| = \frac{\frac1n\cdot\frac1n\cdot\frac3n\cdot\ldots\cdot\frac{n-3}{n}\cdot\frac{n+1}{n}\cdot\ldots\cdot\frac{2n-1}{n}}{\left(\frac2n\cdot\frac4n\cdot\ldots\cdot\frac{n-2}{n}\cdot1\right)^2} = \frac{(2n)!}{(n-1)2^{2n}((n/2)!)^2 n!} \sim \frac{2^{n+3/2}}{\pi\,(n-1)\,n}$$

Theory [6]: for uniformly spaced nodes $\boxed{\lambda_\mathcal{T} \geq Ce^{n/2}}$ for $C > 0$ independent of n. ◇

Example 8.2.5 (Oscillating interpolation polynomial: Runge's counterexample).

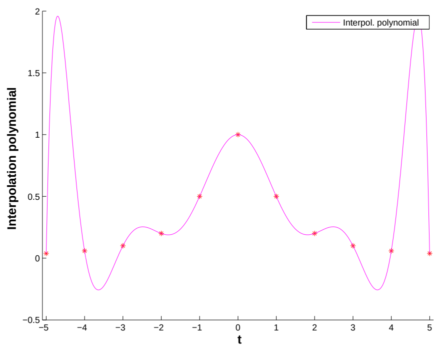
Between the nodes the interpolation polynomial can oscillate excessively and overestimate the changes in the values: bad approximation of functions!

Interpolation polynomial with uniformly spaced nodes:

$$\mathcal{T} := \left\{-5 + \tfrac{10}{n}\,j\right\}_{j=0}^n\,,$$
$$y_j = \frac{1}{1+t_j^2}\,, \quad j = 0,\dots\, n.$$

Plot $n = 10$ ➔

See example 8.4.3.



Attention:
strong oscillations of the interpolation polynomials of high degree on uniformly spaced nodes!

◇

8.3 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 8.2.1.

8.3.1 Multiple evaluations

Task: evaluation of p in many points $x_1, \ldots, x_N \in \mathbb{R}$, $N \gg 1$.

- Interpolation with Lagrange polynomials (8.2.3) , (8.2.4) is not efficient: $O(n^2)$ operations for every value $t \in \mathbb{R}$.
- More efficient formula:

$$p(t) = \sum_{i=0}^{n} L_i(t)\, y_i = \sum_{i=0}^{n} \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{t-t_j}{t_i-t_j}\, y_i = \sum_{i=0}^{n} \lambda_i \prod_{\substack{j=0\\ j\neq i}}^{n} (t-t_j)\, y_i = \prod_{j=0}^{n}(t-t_j) \cdot \sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}\, y_i \,.$$

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 $p(t) \equiv 1$, $y_i = 1$:

$$1 = \prod_{j=0}^{n}(t-t_j) \sum_{i=0}^{n} \frac{\lambda_i}{t-t_i} \quad \Rightarrow \quad \prod_{j=0}^{n}(t-t_j) = \frac{1}{\sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}}$$

Barycentric interpolation formula $\quad p(t) = \dfrac{\sum_{i=0}^{n} \dfrac{\lambda_i}{t-t_i}\, y_i}{\sum_{i=0}^{n} \dfrac{\lambda_i}{t-t_i}} \,.$ (8.3.1)

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 8.3.1: Evaluation of the interpolation polynomials with barycentric formula
```
1  function p = intpolyval(t,y,x) %Arguments must be row vectors!
2  n = length(t); N = length(x);
3  for k = 1:n, lambda(k) = 1 / prod(t(k) - t([1:k-1,k+1:n])); end;
4  for i = 1:N
5    z = (x(i)-t); j = find(z == 0);
6    if (~isempty(j)), p(i) = y(j);
7    else
8      mu = lambda./z; p(i) = dot(mu,y)/sum(mu);
9    end
10 end
```

Lines 6-7 $\longrightarrow$ avoid division by zero.

`tic-toc`-computational time, Matlab polyval vs. barycentric formula $\longrightarrow$ Ex. 8.3.3.

8.3.2 Single evaluation

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:

$$
\begin{aligned}
p_i(t) &\equiv y_i\,, \qquad\qquad i = 0,\dots,n\,,\\
p_{i_0,\dots,i_m}(t) &= \frac{(t-t_{i_0})p_{i_1,\dots,i_m}(t) - (t-t_{i_m})p_{i_0,\dots,i_{m-1}}(t)}{t_{i_m}-t_{i_0}}\,.
\end{aligned}
\tag{8.3.2}
$$

Aitken-Neville algorithm:

$$\begin{tikzcd}
 & n = \quad 0 & 1 & 2 & 3 \\
t_0 & y_0 =: p_0(x) \arrow[r] & p_{01}(x) \arrow[r] & p_{012}(x) \arrow[r] & p_{0123}(x) \\
t_1 & y_1 =: p_1(x) \arrow[r] \arrow[ur] & p_{12}(x) \arrow[r] \arrow[ur] & p_{123}(x) \arrow[ur] & \\
t_2 & y_2 =: p_2(x) \arrow[r] \arrow[ur] & p_{23}(x) \arrow[ur] & & \\
t_3 & y_3 =: p_3(x) \arrow[ur] & & &
\end{tikzcd}$$

Code 8.3.2: Aitken-Neville algorithm

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

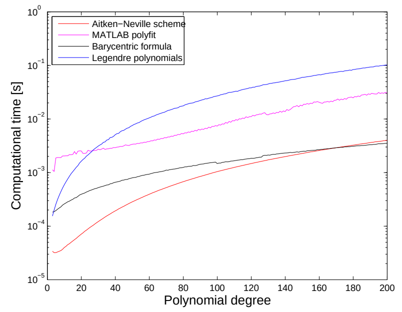
Example 8.3.3 (Timing polynomial evaluations).

Comparison of the computational time needed for polynomial interpolation of

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

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

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



Code 8.3.4: 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;
```

This uses functions given in Code 8.3.0, Code 8.3.1 and the MATLAB function `polyfit` (with a clearly greater computational effort !)

Code 8.3.5: 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 8.3.6: 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
```

◇

8.3.3 Extrapolation to zero

Extrapolation is the same as interpolation but the evaluation point t is outside the interval $[\inf_{j=0,\dots,n} t_j, \sup_{j=0,\dots,n} t_j]$. Assume $t = 0$.

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

Known: 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 $\quad |R(h)| = O(h^{2n+2})$ for $h \to 0$.

Idea:

① evaluation of $f(t_i)$ for different t_i, $i = 0,\dots,n$, $|t_i| > 0$.

② $f(0) \approx p(0)$ with interpolation polynomial $p \in \mathcal{P}_n$, $p(t_i) = f(t_i)$.

Example 8.3.7 (Numeric differentiation through extrapolation).

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)

$$T(h) := \frac{f(x+h) - f(x-h)}{2h} \sim f'(x) + \sum_{k=1}^{n} \frac{1}{(2k)!}\frac{\mathrm{d}^{2k} f}{\mathrm{d}x^{2k}}(x)\, h^{2k} + \frac{1}{(2n+2)!} f^{(2n+2)}(\xi(x))\,.$$

Since $\lim_{h\to 0} T(h) = f'(x)$ ➔ estimate of $f'(x)$ by interpolation of T in points h_i.

MATLAB-CODE : Numeric differentiation through interpolation with Aitken Neville scheme: nodes $x \pm h_0/2^i$.

```
function d = diffex(f,x,h0,tol)
h = h0;
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
  if (abs(y(2)-y(1)) < tol*abs(y(1))), break; end
end
d = y(1);
```

A posteriori error estimate

`diffex2(@atan,1.1,0.5)` `diffex2(@sqrt,1.1,0.5)` `diffex2(@exp,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

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

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

Comparison: numeric differentiation with finite (forward) differences
➔ cancellation ➔ smaller accuracy.

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+x2̂))*(1+x2̂);
sqrterr = abs(dirnumdiff(sqrt,x,h)-1/(2*sqrt(x)))*(2*sqrt(x));
experr = abs(dirnumdiff(exp,x,h)-exp(x))/exp(x);

function[df]=dirnumdiff(f,x,h)
df=(f(x+h)-f(x))./h;
end
```

$f(x) = \arctan(x)$

h	Relative error
2^{-1}	0.20786640808609
2^{-6}	0.00773341103991
2^{-11}	0.00024299312415
2^{-16}	0.00000759482296
2^{-21}	0.00000023712637
2^{-26}	0.00000001020730
2^{-31}	0.00000005960464
2^{-36}	0.00000679016113

$f(x) = \sqrt{x}$

h	Relative error
2^{-1}	0.09340033543136
2^{-6}	0.00352613693103
2^{-11}	0.00011094838842
2^{-16}	0.00000346787667
2^{-21}	0.00000010812198
2^{-26}	0.00000001923506
2^{-31}	0.00000001202188
2^{-36}	0.00000198842224

$f(x) = \exp(x)$

h	Relative error
2^{-1}	0.29744254140026
2^{-6}	0.00785334954789
2^{-11}	0.00024418036620
2^{-16}	0.00000762943394
2^{-21}	0.00000023835113
2^{-26}	0.00000000429331
2^{-31}	0.00000012467100
2^{-36}	0.00000495453865

◇

8.3.4 Newton basis and divided differences

Drawback of the Lagrange basis: adding another data point affects *all* basis polynomials!

Alternative, "update friendly" method: Newton basis for $\mathcal{P}_n$

$$N_0(t) := 1\,,\quad N_1(t) := (t - t_0)\,,\quad \dots\quad,\quad N_n(t) := \prod_{i=0}^{n-1}(t - t_i)\,. \tag{8.3.3}$$

➣ LSE for polynomial interpolation problem in Newton basis:

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

⇔ triangular linear system

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

Solution of the system with forward substitution:

$$a_0 = y_0\,,$$

$$a_1 = \frac{y_1 - a_0}{t_1 - t_0} = \frac{y_1 - y_0}{t_1 - t_0} \,,$$
$$a_2 = \frac{y_2 - a_0 - (t_2 - t_0)a_1}{(t_2 - t_0)(t_2 - t_1)} = \frac{y_2 - y_0 - (t_2 - t_0)\frac{y_1 - y_0}{t_1 - t_0}}{(t_2 - t_0)(t_2 - t_1)} \,,$$
$$\vdots$$

Observation: same quantities computed again and again !

In order to find a better algorithm, we turn to a new interpretation of the coefficients a_j of the interpolating polynomials in Newton basis.

Newton basis polynomial $N_j(t)$: degree j and leading coefficient 1
$\Rightarrow$ a_j is the leading coefficient of the interpolating polynomial $p_{0,\dots,j}$

(notation $p_{0,\dots,j}$ introduced in Sect. 8.3.2, see (8.3.2))

➣ Recursion (8.3.2) implies recursion for leading coefficients $a_{\ell,\dots,m}$ of interpolating polynomials $p_{\ell,\dots,m}$, $0 \leq \ell \leq m \leq n$:

$$a_{\ell,\dots,m} = \frac{a_{\ell+1,\dots,m} - a_{\ell,\dots,m-1}}{t_m - t_\ell} \,.$$

Simpler and more efficient algorithm using divided differences:

$$\begin{aligned} y[t_i] &= y_i \\ y[t_i,\dots,t_{i+k}] &= \frac{y[t_{i+1},\dots,t_{i+k}] - y[t_i,\dots,t_{i+k-1}]}{t_{i+k} - t_i} \quad \text{(recursion)} \end{aligned} \tag{8.3.4}$$

Recursive calculation by divided differences scheme, *cf.* Aitken-Neville scheme, Code 8.3.1:

$$\begin{array}{c|l}
t_0 & y[t_0] \\
 & \quad > y[t_0,t_1] \\
t_1 & y[t_1] \qquad\qquad > y[t_0,t_1,t_2] \\
 & \quad > y[t_1,t_2] \qquad\qquad > y[t_0,t_1,t_2,t_3], \\
t_2 & y[t_2] \qquad\qquad > y[t_1,t_2,t_3] \\
 & \quad > y[t_2,t_3] \\
t_3 & y[t_3]
\end{array}$$

the elements are computed from left to right, every "$>$" means recursion (8.3.4).
If a new datum (t_{n+1}, y_{n+1}) is added, it is enough to compute $n+2$ new terms

$$y[t_{n+1}],\ y[t_n,t_{n+1}],\ \dots, y[t_0,\dots,t_{n+1}].$$

Code 8.3.8: Divided differences, recursive implementation
```
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:

$$p(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)(t - t_1) + \cdots + a_n \prod_{j=0}^{n-1}(t - t_j) \tag{8.3.5}$$

$$a_0 = y[t_0],\ a_1 = y[t_0,t_1],\ a_2 = y[t_0,t_1,t_2],\ \dots .$$

"Backward evaluation" of $p(t)$ in the spirit of Horner's scheme:

$$p \leftarrow a_n, \quad p \leftarrow (t - t_{n-1})p + a_{n-1}, \quad p \leftarrow (t - t_{n-2})p + a_{n-2}, \quad \dots .$$

Code 8.3.9: Divided differences evaluation by modified Horner scheme
```
function p = evaldivdiff(t,y,x)
n = length(y)-1;
dd=divdiff(t,y);
p=dd(n+1);
for j=n:-1:1
    p = (x-t(j)).*p+dd(j);
end
```

Computational effort:
- $O(n^2)$ for computation of divided differences,
- $O(n)$ for every single evaluation of $p(t)$.

Remark 8.3.10 (Divided differences and derivatives).

If $y_0,\dots,y_n$ are the values of a smooth function f in the points $t_0,\dots,t_n$, that is, $y_j := f(t_j)$, then

$$y[t_i,\dots,t_{i+k}] = \frac{f^{(k)}(\xi)}{k!}$$

for a certain $\xi \in [t_i, t_{i+k}]$.

$\triangle$

8.4 Interpolation Error Estimates

Perspective approximation of a function by polynomial interpolation

Remark 8.4.1 (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 [32, pag. 331]:
Given a sequence of meshes of increasing size $\{\mathcal{T}_j\}_{j=1}^{\infty}$, $\mathcal{T}_j = \{x_1^{(j)}, \dots, x_j^{(j)}\} \subset [a,b]$, $a \leq x_1^{(j)} < x_2^{(j)} < \cdots < x_j^{(j)} \leq b$, 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 $j \to \infty$.

△

We consider Lagrangian polynomial interpolation on node set

$$\mathcal{T} := \{t_0, \dots, t_n\} \subset I,\ I \subset \mathbb{R},\ \text{interval of length } |I|.$$

Notation: For a continuous function $f : I \mapsto \mathbb{K}$ we define the polynomial interpolation operator, see Thm. 8.2.2

$$\mathsf{I}_{\mathcal{T}}(f) := \mathsf{I}_{\mathcal{T}}(\mathbf{y}) \in \mathcal{P}_n \qquad \text{with} \qquad \mathbf{y} := (f(t_0), \dots, f(t_n))^T \in \mathbb{K}^{n+1}.$$

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

Example 8.4.2 (Asymptotic behavior of polynomial interpolation error).

Interpolation of $f(t) = \sin t$ on equispaced nodes in $I = [0,\pi]$: $\mathcal{T} = \{j\pi/n\}_{j=0}^{n}$.
Interpolation polynomial $p := \mathsf{I}_{\mathcal{T}} f \in \mathcal{P}_n$.

By Thm. 8.4.2:

$$\left\| f^{(k)} \right\|_{L^\infty(I)} \leq 1, \quad \forall k \in \mathbb{N}_0 \quad \Rightarrow \quad \|f-p\|_{L^\infty(I)} \leq \frac{1}{(1+n)!} \max_{t \in I} \left| (t-0)(t-\tfrac{\pi}{n})(t-\tfrac{2\pi}{n}) \cdot \cdots \cdot (t-\pi) \right| \leq \frac{1}{n+1} \left(\frac{\pi}{n}\right)^{n+1}.$$

➜ Uniform exponential convergence of the interpolation polynomials
(It holds for every mesh of nodes $\mathcal{T}$)

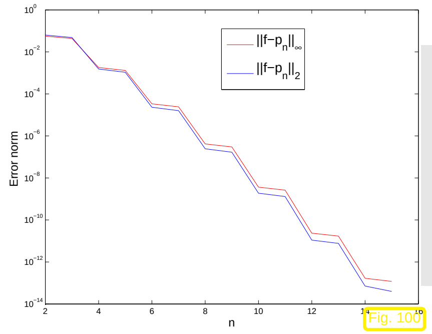


Fig. 100

MATLAB-experiment: computation of the norms.

- L^∞-norm: sampling on a grid of meshsize $\pi/1000$.
- L^2-norm: numeric quadrature (→ Chapter 10) with trapezoidal rule on a grid of meshsize $\pi/1000$.

◇

Example 8.4.3 (Runge's example).

Polynomial interpolation of $f(t) = \frac{1}{1+t^2}$ with equispaced nodes:

$$\mathcal{T} := \left\{ t_j := -5 + \frac{10}{n} j \right\}_{j=0}^{n}, \quad y_j = \frac{1}{1+t_j^2} \, . j = 0, \dots, n \, .$$

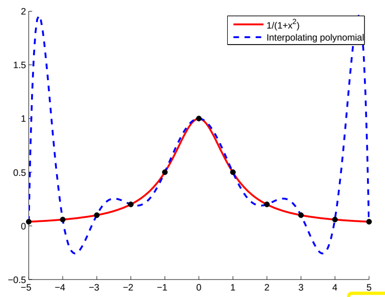


Fig. 101

Interpolating polynomial, $n = 10$

Observations: Strong oscillations of $\mathsf{I}_{\mathcal{T}} f$ near the endpoints of the interval:

$$\|f-\mathsf{I}_{\mathcal{T}} f\|_{L^\infty(]-5,5[)} \xrightarrow{n\to\infty} \infty \,.$$

How can this be reconciled with Thm. 8.4.2 ?

Here $f(t)=\frac{1}{1+t^2}$ implies $|f^{(n)}(t)| = 2^n n! \cdot O(|t|^{-2-n})$.

➜ The error bound from Thm. 8.4.1 $\to\infty$ for $n\to\infty$.

◇

$$\exists\, C \neq C(n):\quad \|f-\mathsf{I}_{\mathcal{T}} f\| \le C\,T(n) \quad \text{for } n\to\infty \,. \tag{8.4.1}$$

Classification (best bound for $T(n)$):

$\exists\, p>0$: $T(n)\le n^{-p}$: algebraic convergence, with rate $p>0$,
$\exists\, 0<q<1$: $T(n)\le q^n$: exponential convergence .

Remark 8.4.4 (Exploring convergence).

Given: pairs (n_i,ϵ_i), $i=1,2,3,\dots$, $n_i \,\hat{=}$ polynomial degrees, $\epsilon_i \,\hat{=}$ norms of interpolation error

❶ Conjectured: algebraic convergence: $\epsilon_i \approx C n^{-p}$

$\log(\epsilon_i)\approx\log(C)-p\log n_i$ (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 (MATLAB `polyfit`) to points $(n_i,\log\epsilon_i)$ ➢ estimate for $q:=\exp(-\beta)$.

△

Beware: same concept $\leftrightarrow$ different meanings:

- convergence of a sequence (e.g. of iterates $\boldsymbol{x}^{(k)}$ $\to$ Sect. 3.1)
- convergence of an approximation (dependent on an approximation parameter, e.g. n)

Theorem 8.4.1 (Representation of interpolation error)**.**
$f\in C^{n+1}(I)$: $\forall\, t\in I$: $\exists\, \tau_t\in]\min\{t,t_0,\dots,t_n\},\max\{t,t_0,\dots,t_n\}[$:

$$f(t)-\mathsf{I}_{\mathcal{T}}(f)(t)=\frac{f^{(n+1)}(\tau_t)}{(n+1)!}\cdot\prod_{j=0}^{n}(t-t_j)\,. \tag{8.4.2}$$

The theorem can also be proved using the following lemma.

Lemma 8.4.2 (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 (8.3.2) and the fundamental theorem of calculus [34, Sect. 3.1]:

Remark 8.4.5. Lemma 8.4.2 holds also for Hermite Interpolation.

△

Interpolation error estimate requires smoothness!

Remark 8.4.6 (L^2-error estimates).

Thm. 8.4.1 gives error estimates for the L^∞-Norm. And the other norms?

From Lemma. 8.4.2 using Cauchy-Schwarz inequality:

$$\|f-\mathsf{I}_{\mathcal{T}}(f)\|^2_{L^2(I)}=\int_I\left|\int_0^1\int_0^{\tau_1}\cdots\int_0^{\tau_{n-1}}\int_0^{\tau_n} f^{(n+1)}(\dots)\,\mathrm{d}\tau\mathrm{d}\tau_n\cdots\mathrm{d}\tau_1\cdot\underbrace{\prod_{j=0}^{n}(t-t_j)}_{|t-t_j|\le|I|}\right|^2\mathrm{d}t$$
$$\le\int_I|I|^{2n+2}\underbrace{\mathrm{vol}_{(n+1)}(S_{n+1})}_{=1/(n+1)!}\int_{S_{n+1}}|f^{(n+1)}(\dots)|^2\,\mathrm{d}\boldsymbol{\tau}\,\mathrm{d}t$$

$$= \int_I \frac{|I|^{2n+2}}{(n+1)!} \int_I \underbrace{\mathrm{vol}_{(n)}(C_{t,\tau})}_{\leq 2^{(n-1)/2}/n!} |f^{(n+1)}(\tau)|^2 \,\mathrm{d}\tau\mathrm{d}t \;,$$

$$S_{n+1} := \{\mathbf{x} \in \mathbb{R}^{n+1} \colon 0 \leq x_n \leq x_{n-1} \leq \cdots \leq x_1 \leq 1\} \quad \text{(unit simplex)} \;,$$
$$C_{t,\tau} := \{\mathbf{x} \in S_{n+1} \colon t_0 + x_1(t_1 - t_0) + \cdots + x_n(t_n - t_{n-1}) + x_{n+1}(t - t_n) = \tau\} \;.$$

This gives the bound for the L^2-norm of the error:

$$\Rightarrow \quad \|f - \mathsf{I}_{\mathcal{T}}(f)\|_{L^2(I)} \leq \frac{2^{(n-1)/4}|I|^{n+1}}{\sqrt{(n+1)!n!}} \left(\int_I |f^{(n+1)}(\tau)|^2 \,\mathrm{d}\tau \right)^{1/2} . \qquad (8.4.3)$$

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).

8.5 Chebychev Interpolation

Perspective: function approximation by polynomial interpolation

➢ Freedom to choose interpolation nodes judiciously

8.5.1 Motivation and definition

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. 8.4.1:
$$\|f - p\|_{L^\infty(I)} \leq \frac{1}{(n+1)!} \left\| f^{(n+1)} \right\|_{L^\infty(I)} \|w\|_{L^\infty(I)} \;,$$
$$w(t) := (t - t_0) \cdot \cdots \cdot (t - t_n) \;.$$

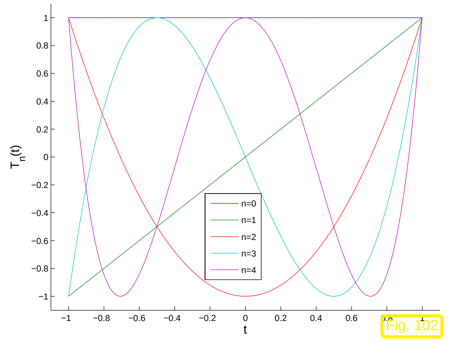
Idea: choose nodes $t_0, \ldots, t_n$ such that $\|w\|_{L^\infty(I)}$ is minimal!

Equivalent to finding $q \in \mathcal{P}_{n+1}$, with leading coefficient $= 1$, such that $\|q\|_{L^\infty(I)}$ is minimal.

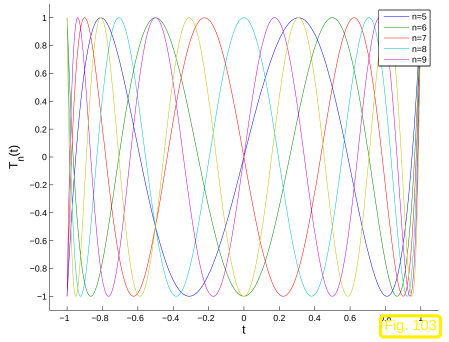
Choice of $t_0, \ldots, t_n$ = zeros of q (caution: t_j must belong to I).

Heuristic:
- t^* extremal point of q ➔ $|q(t^*)| = \|q\|_{L^\infty(I)}$,
- q has $n+1$ zeros in I,
- $|q(-1)| = |q(1)| = \|q\|_{L^\infty(I)}$.

Definition 8.5.1 (Chebychev polynomial)**.**
The n^{th} *Chebychev polynomial is* $T_n(t) := \cos(n \arccos t)$, $\quad -1 \leq t \leq 1$.



Chebychev polynomials $T_0, \ldots, T_4$



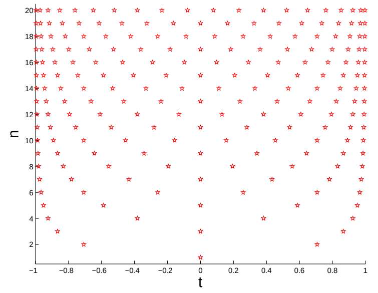
Chebychev polynomials $T_5, \ldots, T_9$

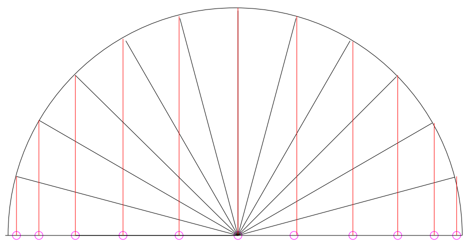
Zeros of T_n:
$$t_k = \cos\left(\frac{2k-1}{2n}\pi\right) \;, \quad k = 1, \ldots, n \;. \qquad (8.5.1)$$

Extrema (alternating signs) of T_n:

$$|T_n(\bar{t}_k)| = 1 \Leftrightarrow \exists\, k = 0, \ldots, n\colon \; \bar{t}_k = \cos\frac{k\pi}{n} \;, \qquad \|T_n\|_{L^\infty([-1,1])} = 1 \;.$$

Chebychev nodes t_k from (8.5.1):





Remark 8.5.1 (3-term recursion for Chebychev polynomial).

3-term recursion by $\cos(n+1)x = 2\cos nx \cos x - \cos(n-1)x$ with $\cos x = t$:

$$T_{n+1}(t) = 2t\,T_n(t) - T_{n-1}(t) \quad , \quad T_0 \equiv 1 \; , \quad T_1(t) = t \; , \quad n \in \mathbb{N} \; . \tag{8.5.2}$$

This implies:
- $T_n \in \mathcal{P}_n$,
- leading coefficients equal to 2^{n-1},
- T_n linearly independent,
- T_n basis of $\mathcal{P}_n = \operatorname{Span}\{T_0, \ldots, T_n\}$, $n \in \mathbb{N}_0$.

△

Theorem 8.5.2 (Minimax property of the Chebychev polynomials)**.**

$$\|T_n\|_{L^\infty([-1,1])} = \inf\{\|p\|_{L^\infty([-1,1])} : p \in \mathcal{P}_n, p(t) = 2^{n-1}t^n + \cdots\} \; , \quad \forall n \in \mathbb{N} \; .$$

Proof. See [13, Section 7.1.4.] □

Application to approximation by polynomial interpolation:

For $I = [-1,1]$
- "optimal" interpolation nodes $\mathcal{T} = \left\{\cos\left(\frac{2k+1}{2(n+1)}\pi\right) \; , k = 0, \ldots, n\right\}$,
- $w(t) = (t - t_0)\cdots(t - t_{n+1}) = 2^{-n}T_{n+1}(t) \quad , \quad \|w\|_{L^\infty(I)} = 2^{-n}$,

  with leading coefficient 1.

Then, by Thm. 8.4.1,

$$\|f - \mathsf{I}_{\mathcal{T}}(f)\|_{L^\infty([-1,1])} \leq \frac{2^{-n}}{(n+1)!} \left\|f^{(n+1)}\right\|_{L^\infty([-1,1])} \; . \tag{8.5.3}$$

Remark 8.5.2 (Chebychev polynomials on arbitrary interval).

How to use Chebychev polynomial interpolation on an arbitrary interval?

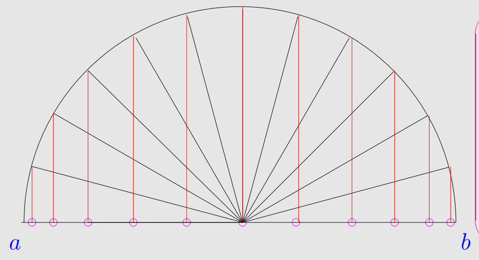
Scaling argument: interval transformation requires the transport of the functions

$$[-1,1] \xrightarrow{\widehat{t} \mapsto t := a + \frac{1}{2}(\widehat{t}+1)(b-a)} [a,b] \quad \leftrightarrow \quad \widehat{f}(\widehat{t}) := f(t) \; .$$

$$p \in \mathcal{P}_n \;\; \wedge \;\; p(t_j) = f(t_j) \quad \Leftrightarrow \quad \widehat{p} \in \mathcal{P}_n \;\; \wedge \;\; \widehat{p}(\widehat{t}_j) = \widehat{f}(\widehat{t}_j) \; .$$

With transformation formula for the integrals & $\dfrac{d^n\widehat{f}}{d\widehat{t}^n}(\widehat{t}) = (\tfrac{1}{2}|I|)^n \dfrac{d^n f}{dt^n}(t)$:

$$\begin{aligned}\|f - \mathsf{I}_{\mathcal{T}}(f)\|_{L^\infty(I)} = \left\|\widehat{f} - \mathsf{I}_{\widehat{\mathcal{T}}}(\widehat{f})\right\|_{L^\infty([-1,1])} &\leq \frac{2^{-n}}{(n+1)!} \left\|\frac{d^{n+1}\widehat{f}}{d\widehat{t}^{n+1}}\right\|_{L^\infty([-1,1])} \\ &\leq \frac{2^{-2n-1}}{(n+1)!} |I|^{n+1} \left\|f^{(n+1)}\right\|_{L^\infty(I)} \; .\end{aligned} \tag{8.5.4}$$



The Chebychev nodes in the interval $I = [a,b]$ are

$$t_k := a + \tfrac{1}{2}(b-a)\left(\cos\left(\frac{2k+1}{2(n+1)}\pi\right) + 1\right) \; , \tag{8.5.5}$$

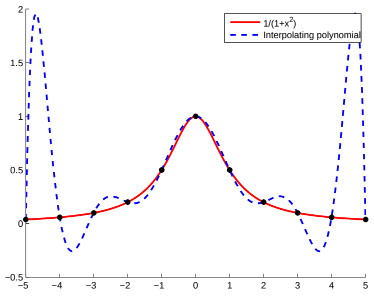
$$k = 0, \ldots, n \; .$$

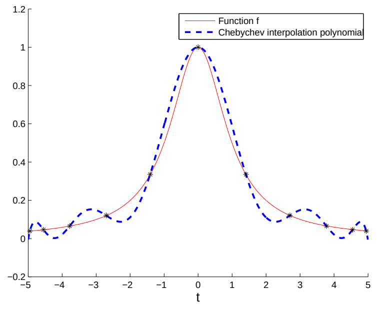
△

8.5.2 Chebychev interpolation error estimates

Example 8.5.3 (Polynomial interpolation: Chebychev nodes versus equidistant nodes).

Runge's function $f(t) = \frac{1}{1+t^2}$, see Ex. 8.4.3, polynomial interpolation based on uniformly spaced nodes and Chebychev nodes:





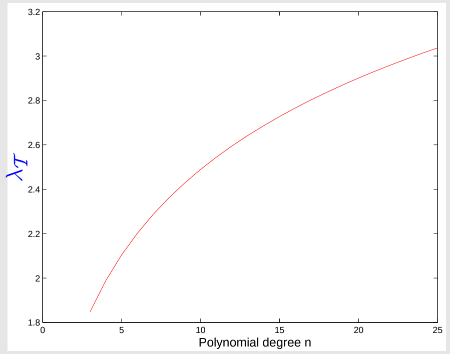
◇

Remark 8.5.4 (Lebesgue Constant for Chebychev nodes).

Theory [5, 44, 43]:

$$\lambda_{\mathcal{T}} \sim \frac{2}{\pi}\log(1+n) + o(1)\ ,$$

$$\boxed{\lambda_{\mathcal{T}} \leq \tfrac{2}{\pi}\log(1+n) + 1}\ . \qquad (8.5.6)$$



△

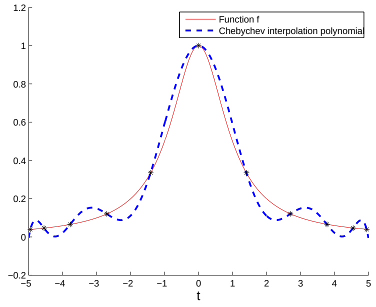
Example 8.5.5 (Chebychev interpolation error).

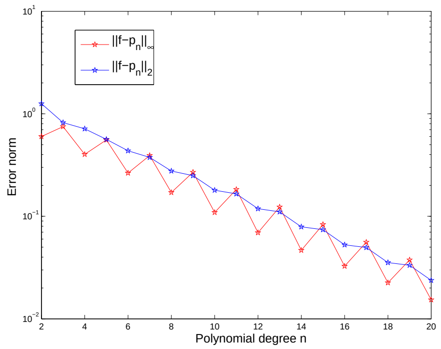
For $I=[a,b]$ let $x_l := a + \frac{b-a}{N}l,\ l = 0,\dots,N,\ N = 1000$ we approximate the norms of the error

$$||f-p||_\infty \approx \max_{0\leq l\leq N} |f(x_l) - p(x_l)|$$

$$||f-p||_2^2 \approx \frac{b-a}{2N}\sum_{0\leq l<N}\left(|f(x_l)-p(x_l)|^2 + |f(x_{l+1})-p(x_{l+1})|^2\right)$$

① $f(t) = (1+t^2)^{-1}$, $I = [-5,5]$ (see Ex. 8.4.3)
Interpolation with $n = 10$ Chebychev nodes (plot on the left).





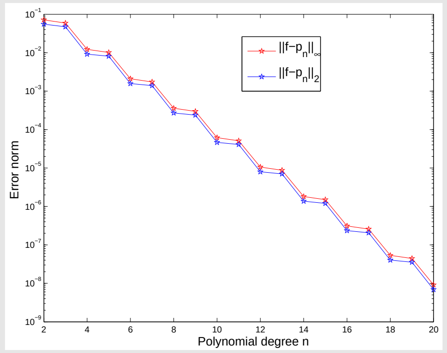
Notice: exponential convergence of the Chebychev interpolation:

$$p_n \to f, \quad \|f - \mathsf{I}_n f\|_{L^2([-5,5])} \approx 0.8^n$$

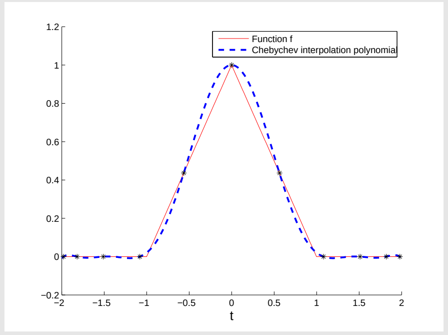
.

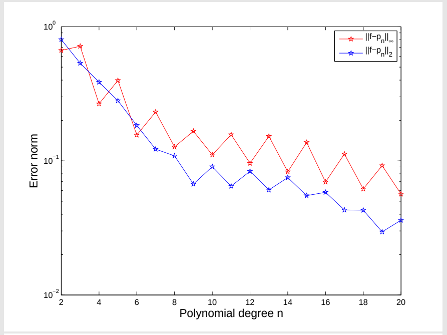
Now: the same function $f(t) = (1+t^2)^{-1}$
on a smaller interval $I = [-1,1]$.
(Faster) exponential convergence:

$$\|f - \mathsf{I}_n f\|_{L^2([-1,1])} \approx 0.42^n .$$



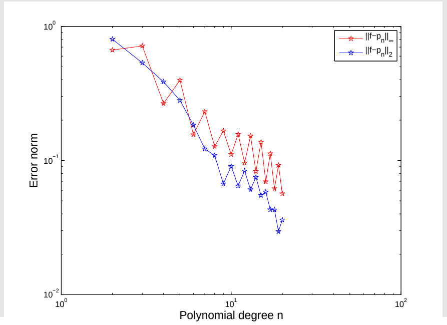
② $f(t) = \max\{1-|t|, 0\}$, $I = [-2,2]$, $n = 10$ nodes (plot on the left).
$f \in C^0(I)$ but $f \notin C^1(I)$.



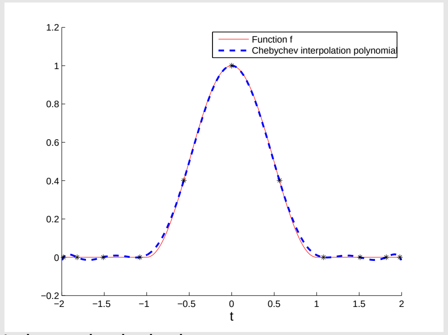


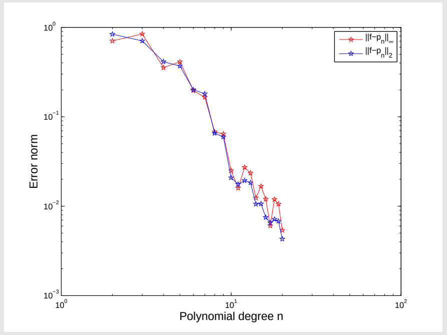
From the double logarithmic plot, notice $\rightarrow$

- no exponential convergence
- algebraic convergence (**?**)



③ $f(t) = \begin{cases} \frac{1}{2}(1+\cos \pi t) & |t| < 1 \\ 0 & 1 \le |t| \le 2 \end{cases}$ $\quad I = [-2, 2], \quad n = 10$ (plot on the left).





Notice: only algebraic convergence.

◇

8.5.3 Chebychev interpolation: computational aspects

Theorem 8.5.3 (Orthogonality of Chebychev polynomials)**.**
The Chebychev polynomials are orthogonal with respect to the scalar product

$$\langle f, g \rangle = \int_{-1}^{1} f(x) g(x) \frac{1}{\sqrt{1-x^2}} dx \; . \tag{8.5.7}$$

Theorem 8.5.4 (Discrete orthogonality of Chebychev polynomials)**.**
The Chebychev polynomials $T_0, \ldots, T_n$ are orthogonal in the space $\mathcal{P}_n$ with respect to the scalar product:

$$(f, g) = \sum_{k=0}^{n} f(x_k) g(x_k) \; , \tag{8.5.8}$$

where $x_0, \ldots, x_n$ are the zeros of T_{n+1}.

① Computation of the coefficients of the interpolation polynomial in Chebychev form:

Theorem 8.5.5 (Representation formula)**.**
The interpolation polynomial p of f in the Chebychev nodes $x_0, \ldots, x_n$ (the zeros of T_{n+1}) is given by:

$$p(x) = \frac{1}{2} c_0 + c_1 T_1(x) + \ldots + c_n T_n(x) \; , \tag{8.5.9}$$

with

$$c_k = \frac{2}{n+1} \sum_{l=0}^{n} f\left(\cos\left(\frac{2l+1}{n+1} \cdot \frac{\pi}{2}\right)\right) \cdot \cos\left(k \frac{2l+1}{n+1} \cdot \frac{\pi}{2}\right) \; . \tag{8.5.10}$$

For sufficiently large n ($n \ge 15$) it is convenient to compute the c_k with the FFT; the direct computation of the coefficients needs $(n+1)^2$ multiplications, while FFT needs only $O(n \log n)$.

② Evaluation of polynomials in Chebychev form:

Theorem 8.5.6 (Clenshaw algorithm)**.**
Let $p \in \mathcal{P}_n$ be an arbitrary polynomial,

$$p(x) = \frac{1}{2}c_0 + c_1T_1(x) + \ldots + c_nT_n(x)\;.$$

Set

$$\begin{aligned} d_{n+2} &= d_{n+1} = 0 \\ d_k &= c_k + (2x)\cdot d_{k+1} - d_{k+2} \quad \text{for } k = n, n-1, \ldots, 0. \end{aligned} \tag{8.5.11}$$

Then $p(x) = \frac{1}{2}(d_0 - d_2)$.

Matlab file: `clenshaw.m`

Code 8.5.6: Clenshaw algorithm

```
n=10;                                  %degree of the interpolating polynomial
f=@(x) 1./(1+25*x.^2);                 %function to approximate, Runge example
x =-1:0.01:1;   fx =f(x);
t = cos(pi*(2*(n+1:-1:1)-1)/(2*n+2));              %Chebychev nodes (n+1)
y = f(t);

c=zeros(1,n+1);
for k=1:n+1;
    c(k)= y * 2/(n+1)*cos(pi/2*(k-1)*(1:2:(2*n+1))/(n+1))';
end;
d=zeros(n+3,length(x));
for k=n+1:-1:1;
    d(k,:)=c(k)+2*x.*d(k+1,:)-d(k+2,:);
end;
p=(d(1,:)-d(3,:))/2;
figure;plot(x,fx, 'r', x, p, 'b--',t,y,'k*');
```

While using recursion it is important how the error (e.g. rounding error) propagates.

Simple example:

$$\begin{aligned} x_{n+1} &= 10x_n - 9, \\ x_0 = 1 &\Rightarrow x_n = 1 \quad \forall n \\ x_0 = 1+\epsilon &\Rightarrow \widetilde{x}_n = 1 + 10^n\epsilon. \end{aligned}$$

This is not a problem here: Clenshaw algorithm is stable.

Theorem 8.5.7 (Stability of Clenshaw algorithm)**.**
Consider the perturbed Clenshaw algorithm recursion:

$$\widetilde{d}_k = c_k + 2x\cdot\widetilde{d}_{k+1} - \widetilde{d}_{k+2} + \epsilon_k\,, \quad k = n, n-1, \ldots, 0$$

with $\widetilde{d}_{n+2} = \widetilde{d}_{n+1} = 0$*. Set* $\widetilde{p}(x) = \frac{1}{2}(\widetilde{d}_0 - \widetilde{d}_2)$*. Then* $|\widetilde{p}(x) - p(x)| \le \sum_{j=0}^{n}|\epsilon_j|$ *for* $|x| \le 1$.

Remark 8.5.7 (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 through Clenshaw algorithm is much more efficient than with Taylor approximation.

△

9 Piecewise Polynomials

Perspective: data interpolation

Problem: model a functional relation $f : I \subset \mathbb{R} \mapsto \mathbb{R}$ from the (exact) measurements (t_i, y_i), $i = 0, \ldots, n$:

➜ Interpolation constraint $f(t_i) = y_i$, $i = 0, \ldots, n$.

9.1 Shape preserving interpolation

When reconstructing a quantitative dependence of quantities from measurements, first principles from physics often stipulated qualitative constraints, which translate into *shape properties* of the function f, e.g., when modelling the material law for a gas:

$$t_i \text{ pressure values, } y_i \text{ densities} \quad \succ \quad f \text{ positive \& monotone.}$$

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

Definition 9.1.1 (monotonic data)**.**
The data (t_i, y_i) *are called* monotonic *when* $y_i \geq y_{i-1}$ *or* $y_i \leq y_{i-1}$, $i = 1, \dots, n$.

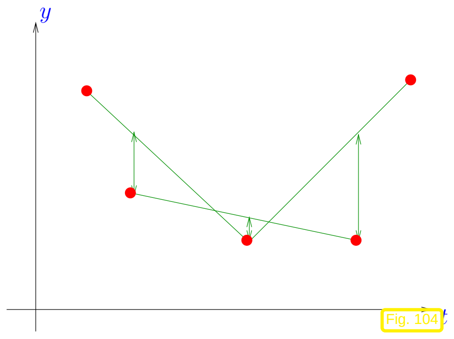
Definition 9.1.2 (Convex/concave data)**.**
The data $\{(t_i, y_i)\}_{i=0}^{n}$ *are called* convex (concave) *if*

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

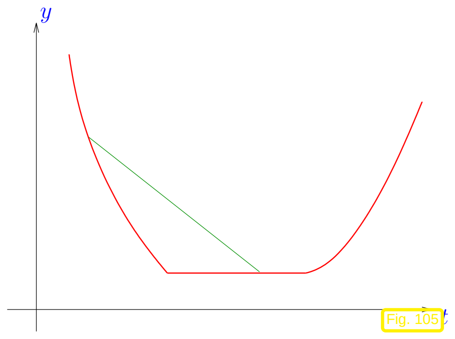
Mathematical characterization of convex data:

$$y_i \leq \frac{(t_{i+1} - t_i) y_{i-1} + (t_i - t_{i-1}) y_{i+1}}{t_{i+1} - t_{i-1}} \qquad \forall\, i = 1, \dots, n-1,$$

i.e., each data point lies below the line segment connecting the other data.



Convex data



Convex function

Definition 9.1.3 (Convex/concave function)**.**

$$f : I \subset \mathbb{R} \mapsto \mathbb{R} \quad \begin{matrix} \text{convex} \\ \text{concave} \end{matrix} \quad :\Leftrightarrow \quad \begin{matrix} f(\lambda x + (1-\lambda) y) \leq \lambda f(x) + (1-\lambda) f(y) \\ f(\lambda x + (1-\lambda) y) \geq \lambda f(x) + (1-\lambda) f(y) \end{matrix} \quad \begin{matrix} \forall\, 0 \leq \lambda \leq 1, \\ \forall\, x, y \in I\ . \end{matrix}$$

Data (t_i, y_i), $i = 0, \dots, n$ $\quad \longrightarrow \quad$ 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 $f\vert_I$,
monotonic data in I	$\longrightarrow$	locally monotonic interpolant $f\vert_I$,
convex data in I	$\longrightarrow$	locally convex interpolant $f\vert_I$.

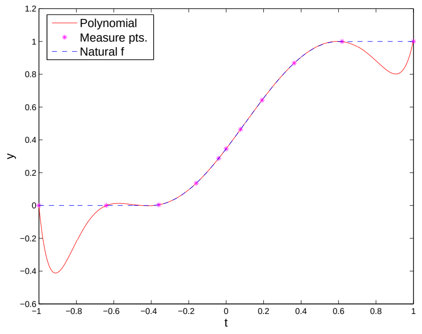
Example 9.1.1 (Bad behavior of global polynomial interpolants).

Positive and monotonic data:

t_i	-1.0000	-0.6400	-0.3600	-0.1600	-0.0400	0.0000	0.0770	0.1918	0.3631	0.6187	1.0000
y_i	0.0000	0.0000	0.0039	0.1355	0.2871	0.3455	0.4639	0.6422	0.8678	1.0000	1.0000

created by taking points on the graph of

$$f(t) = \begin{cases} 0 & \text{if } t < -\frac{2}{5}\,, \\ \frac{1}{2}(1+\cos(\pi(t-\frac{3}{5}))) & \text{if } -\frac{2}{5} < t < \frac{3}{5}\,, \\ 1 & \text{otherwise.} \end{cases}$$



$\leftarrow$ Interpolating polynomial, degree = 10

Oscillations at the endpoints of the interval (see Ex. 8.4.3)

- No locality
- No positivity
- No monotonicity
- No local conservation of the curvature

$\Diamond$

9.2 Piecewise Lagrange interpolation

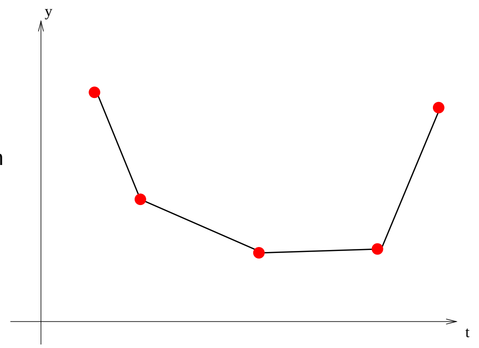
Idea:
use piecewise polynomials with respect to a partition (mesh) $\mathcal{M} := \{a = x_0 < x_1 < \dots < x_m = b\}$ of the interval $I := [a, b]$, $a < b$.

9.2.1 Piecewise linear interpolation

Data: $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \dots, n$, $n \in \mathbb{N}$, $t_0 < t_1 < \dots < t_n$.
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. 104:



Piecewise linear interpolation means simply "connect the data points in $\mathbb{R}^2$ using straight lines".

Theorem 9.2.1 (Local shape preservation by piecewise linear interpolation)**.**
Let $s \in C([t_0, t_n])$ be the piecewise linear interpolant of $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \dots, n$, for every subinterval $I = [t_j, t_k] \subset [t_0, t_n]$:

if $(t_i, y_i)|_I$ are positive/negative $\Rightarrow$ $s|_I$ is positive/negative,
if $(t_i, y_i)|_I$ are monotonic (increasing/decreasing) $\Rightarrow$ $s|_I$ is monotonic (increasing/decreasing),
if $(t_i, y_i)|_I$ are convex/concave $\Rightarrow$ $s|_I$ is convex/concave.

Local shape preservation = perfect shape preservation!
None of this properties carries over to higher polynomial degrees $d > 1$.

Drawback: f is only C^0 but not C^1 (no continuous derivative).

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

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

9.2.2 Piecewise polynomial interpolation

For polynomial degree $d \in \mathbb{N}$, we gather $d+1$ nodes ➔ grid $\mathcal{M}$

:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}
$d=2$:	x_0		x_1		x_2		x_3		x_4		x_5		x_6		x_7		x_8		x_9		x_{10}
$d=4$:	x_0				x_1				x_2				x_3				x_4				x_5

The piecewise interpolation polynomial is $s : [x_0, x_n] \to \mathbb{K}$ such that:

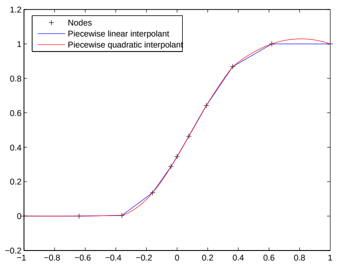
$$s_j := s_{|[x_{j-1},x_j]} \in \mathcal{P}_d \quad \text{and} \quad s_j(t_i) = y_i \quad i = (j-1)\cdot d, \dots, j\cdot d\,, \quad j = 1, \dots, \frac{n}{d}\,.$$

Drawbacks:
- f is only C^0 but not C^1,
- no shape preservation for $d \geq 1$

Locality : $s(t)$ (with $t \in [x_j, x_{j+1}]$) depends only on $d+1$ data points (t_j, y_j), $t_j \in [x_j, x_{j+1}]$.

Example 9.2.1 (Piecewise polynomial interpolation from nodes).
Nodes as in Ex. 9.1.1

Piecewise linear/quadratic interpolation



No shape preservation for piecewise quadratic interpolant

9.2.3 Approximation via piecewise polynomial interpolation

Local Lagrange interpolation of $f \in C(I)$:

on the grid $\mathcal{M} = \{x_0, \dots, x_m\}$ choose a nodes mesh $\mathcal{T}^j := \{t_0^j, \dots, t_{n_j}^j\} \subset I_j$ for each grid cell $I_j := [x_{j-1}, x_j]$ of $\mathcal{M}$.

Piecewise interpolation polynomial $s : [x_0, x_m] \to \mathbb{K}$:

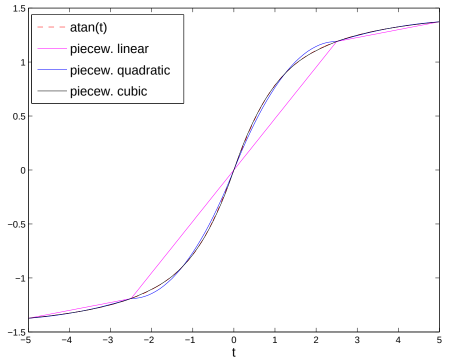
$$s_j := s_{|I_j} \in \mathcal{P}_{n_j} \quad \text{and} \quad s_j(t_i^j) = f(t_i^j) \quad i = 0, \dots, n_j\,, \quad j = 1, \dots, m\,.$$

Problem: asymptotic of the interpolation errors for n constant, $m \to \infty$
(interpolation error $\leq T(h)$, mesh width $h := \max\{|x_j - x_{j-1}|: j = 1, \dots, m\}$)

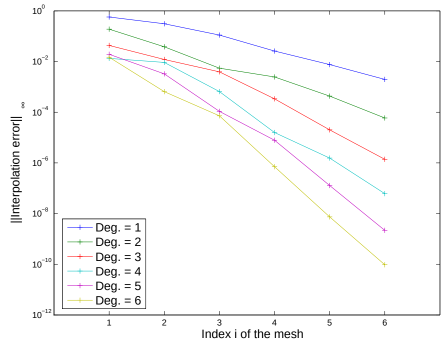
Example 9.2.2 (Piecewise polynomial interpolation).

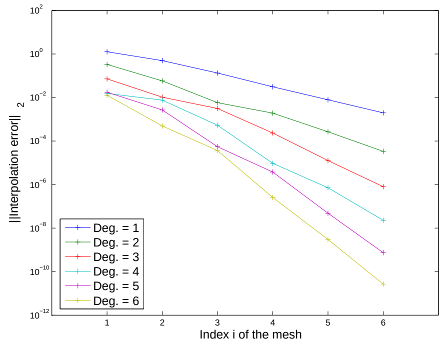
Compare Ex. 9.1.1:
$f(t) = \arctan t$, $I = [-5, 5]$
Grid $\mathcal{M} := \{-5, -\frac{5}{2}, 0, \frac{5}{2}, 5\}$
$\mathcal{T}^j$ equidistant in I_j.

Plots of the piecewise linear, quadratic and cubic polynomial interpolants $\longrightarrow$



Interpolation error in L^∞- and L^2-norm on the grids $\mathcal{M}_i := \{-5 + j\,2^{-i}10\}_{j=0}^{2^i}$, $i = 1, \dots, 6$, equidistant nodes meshes:

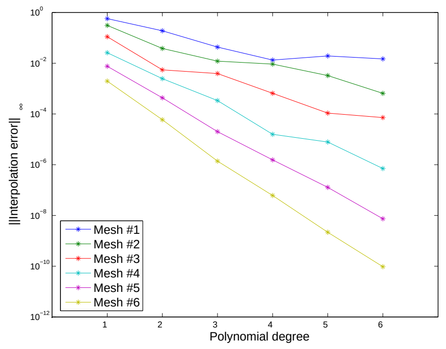


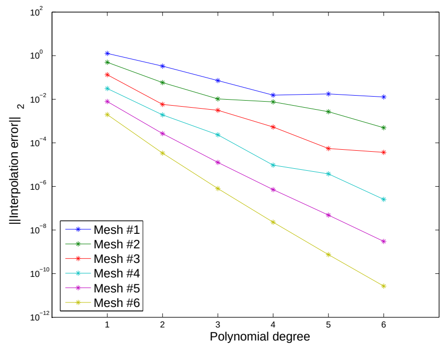


➤ Algebraic convergence in meshwidth

(Meshwidth $h = 10 \cdot 2^i$, index $i = \log_2(10/h)$)

➤ Exponential convergence in polynomial degree





◇

Interpolation error estimate:

- For constant polynomial degree $n = n_j$, $j = 1, \dots, m$:

  Thanks to Thm. 8.4.1 applied on every interval: if $f \in C^{n+1}([x_0, x_m])$

$$\|f - s\|_{L^\infty([x_0,x_m])} \leq \frac{h^{n+1}}{(n+1)!} \left\| f^{(n+1)} \right\|_{L^\infty([x_0,x_m])} \quad , \tag{9.2.1}$$

  with mesh width $h := \max\{|x_j - x_{j-1}|: j = 1, \dots, m\}$.
- For fixed mesh: the situation is the same as with standard polynomial interpolation, see Section 8.4.

9.3 Cubic Hermite Interpolation

Idea: choose nodes $t_0, \dots, t_n$ such that $\|w\|_{L^\infty(I)}$ is minimal!

Equivalent to find $q \in \mathcal{P}_{n+1}$, with leading coefficient $= 1$, such that $\|q\|_{L^\infty(I)}$ is minimal.

Choice of $t_0, \dots, t_n$ = zeros of q (caution: t_j must belong to I).

9.3.1 Definition and algorithms

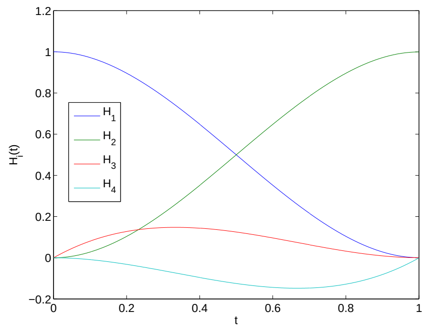
Given: mesh points $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \dots, n$, $t_0 < t_1 < \dots < t_n$

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

➜ Piecewise cubic Hermite interpolation polynomial $s \in C^1([t_0, t_n])$:
with given slopes $c_i \in \mathbb{R}$, $i = 0, \dots, n$

$$s_{|[t_{i-1},t_i]} \in \mathcal{P}_3 \; , \quad i = 1, \dots, n \quad , \quad s(t_i) = y_i \; , \quad i = 0, \dots, n \quad , \quad s'(t_i) = c_i \; , \quad i = 0, \dots, n \; .$$

▶ $$s(t) = y_{i-1}H_1(t) + y_i H_2(t) + c_{i-1}H_3(t) + c_i H_4(t) \quad , \quad t \in [t_{i-1}, t_i] \; , \tag{9.3.1}$$



$$\begin{aligned} H_1(t) &:= \phi(\tfrac{t_i - t}{h_i}) \; , \quad H_2(t) := \phi(\tfrac{t - t_{i-1}}{h_i}) \; , \\ H_3(t) &:= -h_i \psi(\tfrac{t_i - t}{h_i}) \; , \quad H_4(t) := h_i \psi(\tfrac{t - t_{i-1}}{h_i}) \; , \\ h_i &:= t_i - t_{i-1} \quad , \\ \phi(\tau) &:= 3\tau^2 - 2\tau^3 \quad , \\ \psi(\tau) &:= \tau^3 - \tau^2 \; . \end{aligned} \tag{9.3.2}$$

◁ Local basis polynomial on $[0, 1]$

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 9.3.1: Hermite local evaluation

```
function s=hermloceval(t,t1,t2,y1,y2,c1,c2)
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 ?

Average of local slopes:

$$c_i = \begin{cases} \Delta_1 & , \text{ for } i = 0 \; , \\ \Delta_n & , \text{ for } i = n \; , \\ \frac{t_{i+1} - t_i}{t_{i+1} - t_{i-1}} \Delta_i + \frac{t_i - t_{i-1}}{t_{i+1} - t_{i-1}} \Delta_{i+1} & , \text{ if } 1 \leq i < n \; . \end{cases} \quad , \quad \Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}} \; , j = 1, \dots, n \; . \tag{9.3.3}$$

▶ Linear local interpolation operator

See (9.3.4) for a different choice of the slopes.

Example 9.3.2 (Piecewise cubic Hermite interpolation).

Interpolation of the function:

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

in the interval $I = [-1, 1]$
on 11 equispaced points

$$t_j = -1 + 0.2\, j, \quad j = 0, \dots, 10.$$

Use of weighted averages of slopes as in (9.3.3).
See Code 9.3.2.

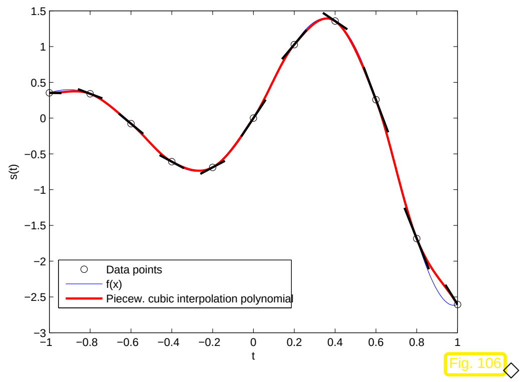


Fig. 106

Code 9.3.3: Piecewise cubic Hermite interpolation

```
% 25.11.2009 hermintp1.m
% compute and plot the Hermite interpolant of the function f in the nodes t
% choice of the slopes using weighted averages of the local slopes

function hermintp1(f,t)
n = length(t);
```

```
h = diff(t);                                    % increments in t
y = feval(f,t);
c=slopes1(t,y);

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
  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)],'k-','LineWidth',2);
end
plot([t(1),t(1)+0.3*h(1)],[y(1),y(1)+0.3*h(1)*c(1)],'k-',
  'LineWidth',2);
plot([t(end)-0.3*h(end),t(end)],[y(end)-0.3*h(end)*c(end),y(end)],'k-',
  'LineWidth',2);
xlabel('t');
ylabel('s(t)');
```

```
legend('Data␣points','f(x)','Piecew.␣cubic␣interpolation␣polynomial␣');
hold off;

%-----------------------------------------------------------
% slopes for interpolation: version (1) middle points of slopes
function c=slopes1(t,y)
n = length(t);
h = diff(t);                                    % increments in t
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)];
```

Try: `hermintp1( @(x)sin(5*x).*exp(x), [-1:0.2:1]);`

9.3.2 Interpolation error estimates

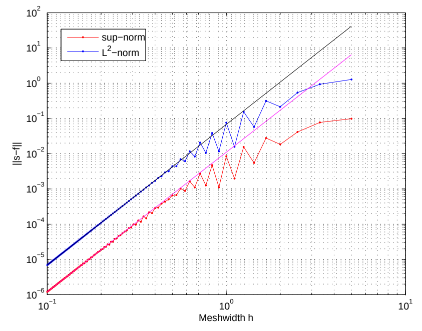
Example 9.3.4 (Convergence of Hermite interpolation with exact slopes).

Piecewise cubic Hermite interpolation of

$$f(x) = \arctan(x)\ .$$

- domain: $I = (-5, 5)$
- mesh $\mathcal{T} = \{-5 + hj\}_{j=0}^{n} \subset I$, $h = \frac{10}{n}$,
- exact slopes $c_i = f'(t_i)$, $i = 0, \dots, n$

▶ algebraic convergence $O(h^{-4})$



Formulas for approximate computation of errors:
d = vector of absolute values of errors $|f(x_j) - s(x_j)|$ on a fine uniform mesh $\{x_j\}$,
h = meshsize $x_{j+1} - x_j$,
L^∞-error = `max(d);`
L^2-error = `sqrt(h*(sum(d(2:end-1).∧2)+(d(1)∧2+d(end)∧2)/2) );`
(trapezoidal rule).

Code 9.3.5: Hermite approximation and orders of convergence with exact slopes

```
% 16.11.2009 hermiteapprox1.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: exact slopes of f
%
% inputs: f function to be interpolated
% df derivative of f
% a, b left and right extremes of the interval
% N maximum number of subdomains

function hermiteapprox1(f,df,a,b,N)
err = [];
for j=2:N
  xx=a;          % xx is the mesh on which the error is computed, built in the k-loop
  val=f(a);

  t = a:(b-a)/j:b;          % nodes
  y = f(t);
  c=df(t);          % compute exact slopes

  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)];
  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('Meshwidth h');
ylabel('||s-f||');
legend('sup-norm','L^2-norm');

% compute algebraic orders of convergence,
% using polynomial fit on half of the plot
pl = polyfit(log(err(ceil(N/2):N-2,1)),log(err(ceil(N/2):N-2,2)),1);
  exp_rate_Linf=pl(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)].^pl(1)*exp(pl(2)),'m')
plot([err(1,1),err(N-1,1)],
  [err(1,1),err(N-1,1)].^pL2(1)*exp(pL2(2)),'k')
```

Try: `hermiteapprox1(@atan, @(x) 1./(1+x.^2), -5,5,100);`

◇

Example 9.3.6 (Convergence of Hermite interpolation with averaged slopes).

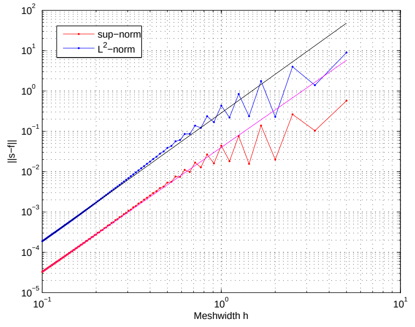
Piecewise cubic Hermite interpolation of

$$f(x) = \arctan(x) \;.$$

- domain: $I = (-5,5)$
- equidistant mesh $\mathcal{T}$ in I, see Ex. 9.3.4,
- averaged local slopes, see (9.3.3)

▶ algebraic convergence in meshwidth

See Code 9.3.6.



Lower order of convergence due to the choice of the slopes (9.3.3):

from the plot L^∞-norm $\sim$ L^2-norm $\sim O(h^3)$

◇

Code 9.3.7: 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)
err = [];
for j=2:N
  xx=a;          % xx is the mesh on which the error is computed, built in the k-loop
  val=f(a);

  t = a:(b-a)/j:b;          % nodes
  y = f(t);
  c=slopes1(t,y);           % compute average slopes

  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)];
  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('Meshwidth h');
ylabel('||s-f||');
legend('sup-norm','L^2-norm');

% compute algebraic orders of convergence
% using polynomial fit on half of the plot
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')

%-------------------------------
function   c=slopes1(t,y)
h = diff(t);                          % increments in t
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)];
```

Try: `hermiteapprox(@atan, -5,5,100);`

9.3.3 Shape preserving Hermite interpolation

Slopes according to (9.3.3) $\succ$ Hermite interpolation does not preserve monotonicity.
Remedy: choice of the slopes c_i via "limiter" $\longrightarrow$ conservation of monotonicity.

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

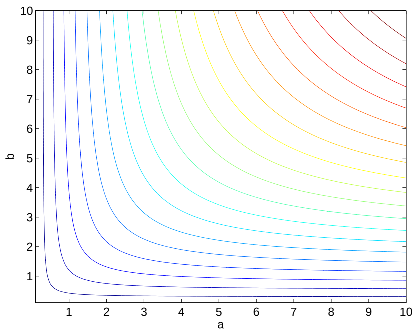
Which kind of average ?

$$c_i = \frac{1}{\frac{w_a}{\Delta_i} + \frac{w_b}{\Delta_{i+1}}} \tag{9.3.4}$$

= weighted harmonic mean of the slopes with weights w_a, w_b, $(w_a + w_b = 1)$.

Harmonic mean = "smoothed $\min(\cdot,\cdot)$-function".

Contour plot of the harmonic mean of a and b ➜ ($w_a = w_b = 1/2$).



Concrete choice of the weights:

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

$$\rightarrow \quad c_i = \begin{cases} \Delta_1 & \text{, if } i=0\,, \\ \dfrac{3(h_{i+1}+h_i)}{\frac{2h_{i+1}+h_i}{\Delta_i}+\frac{2h_i+h_{i+1}}{\Delta_{i+1}}} & \text{, for } i\in\{1,\dots,n-1\}\,, \\ \Delta_n & \text{, if } i=n\,, \end{cases} \quad , \quad h_i := t_i - t_{i-1}\,. \tag{9.3.5}$$

Example 9.3.8 (Monotonicity preserving piecewise cubic polynomial interpolation).

Data from ex. 9.1.1

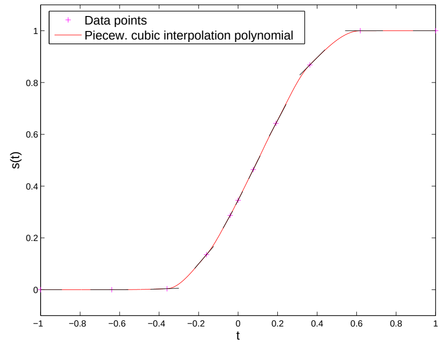
MATLAB-function:

```
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 9.3.1 (Monotonicity preservation of limited cubic Hermite interpolation)**.**
The cubic Hermite interpolation polynomial with slopes as in (9.3.5) *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 9.3.9 (Non-linear interpolation).

The monotonicity preserving cubic Hermite interpolation is non-linear !

Terminology: An interpolation operator $I : \mathbb{R}^{n+1} \mapsto C^0([t_0,t_n])$ on the given nodes $t_0 < t_1 < \cdots < t_n$ is called *linear*, if

$$I(\alpha\mathbf{y}+\beta\mathbf{z}) = \alpha I(\mathbf{y}) + \beta I(\mathbf{z}) \quad \forall \mathbf{y},\mathbf{z}\in\mathbb{R}^{n+1},\ \alpha,\beta\in\mathbb{R}\,.$$

△

Calculation of the c_i in `pchip` (details in [15]):

Code 9.3.10: Monotonicity preserving slopes in pchip

```
%PCHIPSLOPES Slopes for shape-preserving Hermite cubic
%pchipslopes(x,y) computes c(k) = P'(x(k)).
%
%Slopes at interior points
%delta = diff(y)./diff(x).
%c(k) = 0 if delta(k-1) and delta(k) have opposite signs or either is zero.
```

```
%c(k) = weighted harmonic mean of delta(k-1) and delta(k) if they have the same sign.

function c = pchipslopes(x,y)
   n = length(x); h = diff(x); delta = diff(y)./h;
   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));

%Slopes at endpoints
   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
```

9.4 Splines

Definition 9.4.1 (Spline space)**.**
Given an interval $I := [a,b] \subset \mathbb{R}$ and a 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

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

Spline spaces mapped onto each other by differentiation & integration:

$$s \in \mathcal{S}_{d,\mathcal{M}} \quad \Rightarrow \quad s' \in \mathcal{S}_{d-1,\mathcal{M}} \quad \wedge \quad \int_a^t s(\tau)\,\mathrm{d}\tau \in \mathcal{S}_{d+1,\mathcal{M}} \ .$$

- $d = 0$: $\mathcal{M}$-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.

9.4.1 Cubic spline interpolation

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. 9.3)

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

$$s(t_j) = y_j \quad , \quad j = 0,\ldots,n \ . \tag{9.4.1}$$

$\hat{=}$ interpolation at nodes of mesh

Remark 9.4.1 (Extremal properties of 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, $s^{(4)} \equiv 0$:

$$\frac{\mathrm{d}h}{\mathrm{d}\lambda}\Big|_{\lambda=0} = -\sum_{j=1}^{n} \Big(s'''(t_j^-)\underbrace{k(t_j)}_{=0} - s'''(t_{j-1}^+)\underbrace{k(t_{j-1})}_{=0} \Big) + \underbrace{s''(t_n)}_{=0} k'(t_n) - \underbrace{s''(t_0)}_{=0} k'(t_0) = 0 \ .$$

Theorem 9.4.2 (Optimality of natural cubic spline interpolant).
The natural cubic spline interpolant minimizes the elastic curvature energy among all interpolating functions in $C^2([a,b])$.

From dimensional considerations it is clear that the interpolation conditions will fail to fix the interpolating cubic spline uniquely:

$$\dim \mathcal{S}_{3,\mathcal{M}} - \#\{\text{interpolation conditions}\} = (n+3) - (n+1) = 2 \text{ free d.o.f.}$$

“two conditions are missing”

Algorithmic approach to finding s:

Reuse representation through cubic Hermite basis polynomials from (9.3.2):

$$\overset{(9.3.1)}{\blacktriangleright} \quad \begin{aligned} s_{|[t_{j-1},t_j]}(t) = \; & s(t_{j-1}) && \cdot(1-3\tau^2+2\tau^3)+ \\ & s(t_j) && \cdot(3\tau^2-2\tau^3)+ \\ & h_j s'(t_{j-1}) && \cdot(\tau-2\tau^2+\tau^3)+ \\ & h_j s'(t_j) && \cdot(-\tau^2+\tau^3)\;, \end{aligned} \tag{9.4.2}$$

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. 9.3.1, Code 9.3.0.

Note: if $s(t_j)$, $s'(t_j)$, $j=0,\ldots,n$, are fixed, then the representation (9.4.2) already guarantees $s \in C^1([t_0,t_n])$, *cf.* the discussion for cubic Hermite interpolation, Sect. 9.3.

➣ only continuity of s'' • has to be enforced by choice of $s'(t_j)$
$\Updownarrow$
• will yield extra conditions to fix the $s'(t_j)$

However, do the

- interpolation conditions (9.4.1) $s(t_j)=y_j$, $j=0,\ldots,n$, 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

$$s''_{|[t_{j-1},t_j]}(t_j) = s''_{|[t_j,t_{j+1}]}(t_j)\;, \quad j=1,\ldots,n-1\;. \tag{9.4.3}$$

Based on (9.4.2), we express (9.4.3) in concrete terms, using

$$\begin{aligned} s''_{|[t_{j-1},t_j]}(t) = \; & s(t_{j-1})h_j^{-2}6(-1+2\tau) + s(t_j)h_j^{-2}6(1-2\tau) \\ & + h_j^{-1}s'(t_{j-1})(-4+6\tau) + h_j^{-1}s'(t_j)(-2+6\tau)\;, \end{aligned} \tag{9.4.4}$$

which can be obtained by the chain rule and from $\frac{d\tau}{dt} = h_j^{-1}$.

$$\overset{(9.4.4)}{\Rightarrow} \quad \begin{aligned} s''_{|[t_{j-1},t_j]}(t_{j-1}) &= -6\cdot s(t_{j-1})h_j^{-2} + 6\cdot s(t_j)h_j^{-2} - 4\cdot h_j^{-1}s'(t_{j-1}) - 2\cdot h_j^{-1}s'(t_j)\;, \\ s''_{|[t_{j-1},t_j]}(t_j) &= 6\cdot s(t_{j-1})h_j^{-2} + -6\cdot s(t_j)h_j^{-2} + 2\cdot h_j^{-1}s'(t_{j-1}) + 4\cdot h_j^{-1}s'(t_j)\;. \end{aligned}$$

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

$$\frac{1}{h_j}c_{j-1} + \left(\frac{2}{h_j}+\frac{2}{h_{j+1}}\right)c_j + \frac{1}{h_{j+1}}c_{j+1} = 3\left(\frac{y_j-y_{j-1}}{h_j^2} + \frac{y_{j+1}-y_j}{h_{j+1}^2}\right)\;, \tag{9.4.5}$$

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

(9.4.5) $\Leftrightarrow$ *undetermined* $(n-1)\times(n+1)$ linear system of equations

$$\begin{pmatrix} b_0 & a_1 & b_1 & 0 & \cdots & \cdots & 0 \\ 0 & b_1 & a_2 & b_2 & & & \\ & 0 & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \\ & & & \ddots & a_{n-1} & b_{n-2} & 0 \\ 0 & \cdots & \cdots & 0 & b_{n-2} & a_0 & b_{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} 3\left(\frac{y_1-y_0}{h_1^2}+\frac{y_2-y_1}{h_2^2}\right) \\ \vdots \\ 3\left(\frac{y_{n-1}-y_{n-2}}{h_{n-1}^2}+\frac{y_n-y_{n-1}}{h_n^2}\right) \end{pmatrix}\;. \tag{9.4.6}$$

➔ two additional constraints are required, (at least) three different choices are possible:

① Complete cubic spline interpolation: $s'(t_0)=c_0$, $s'(t_n)=c_n$ prescribed.

② Natural cubic spline interpolation: $s''(t_0)=s''(t_n)=0$

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

➤ Linear system of equations with tridiagonal s.p.d. (→ Def. 2.7.1, Lemma 2.7.4) coefficient matrix $\longrightarrow$ $c_0,\dots,c_n$

Thm. 2.6.6 $\Rightarrow$ 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

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

Solved with rank-1-modifications technique (see Section 2.9.0.1, Lemma 2.9.1) + tridiagonal elimination, computational effort $O(n)$

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

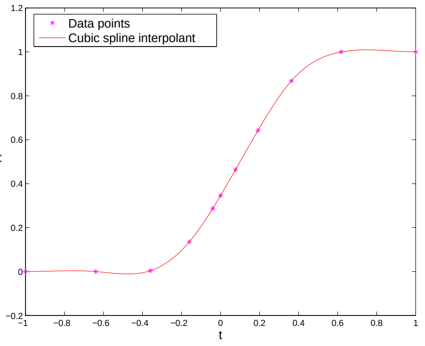
Notice analogies and differences:

		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.

Remark 9.4.2 (Shape preservation).
Data $s(t_j) = y_j$ from Ex. 9.1.1 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!)



△

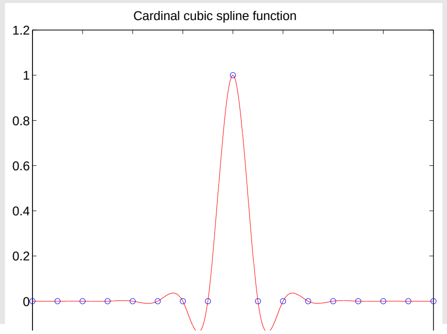
Example 9.4.3 (Locality of the natural cubic spline interpolation).

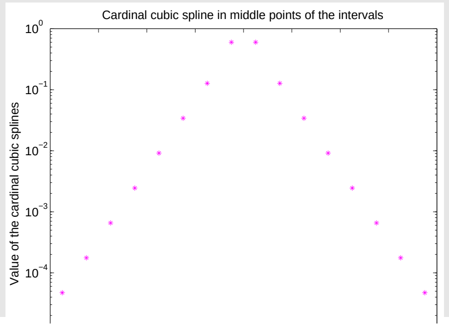
Given a grid $\mathcal{M} := \{t_0 < t_1 < \cdots < t_n\}$ the ith natural cardinal spline is defined as

$$L_i \in \mathcal{S}_{3,\mathcal{M}}, \quad L_i(t_j) = \delta_{ij} , \quad L_i''(t_0) = L_i''(t_n) = 0 .$$

Natural spline interpolant: $s(t) = \sum_{j=0}^{n} y_j L_j(t)$.

Decay of L_i $\leftrightarrow$ locality of the cubic spline interpolation.



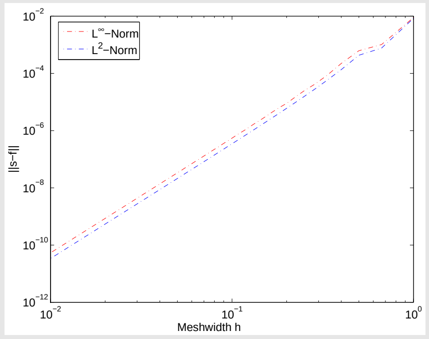


Exponential decay of the cardinal splines ➞ cubic spline interpolation is “almost local” ◇

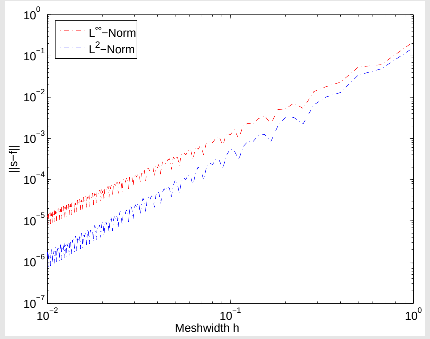
Example 9.4.4 (Approximation by complete cubic spline interpolants).

Grid $\mathcal{M} := \{-1 + \frac{2}{n} j\}_{j=0}^{n}$, $n \in \mathbb{N}$ ➞ meshwidth $h = 2/n$, $I = [-1, 1]$

$$f_1(t) = \frac{1}{1 + e^{-2t}} \in C^\infty(I) \quad , \quad f_2(t) = \begin{cases} 0 & \text{, if } t < -\frac{2}{5} , \\ \frac{1}{2}(1 + \cos(\pi(t - \frac{3}{5}))) & \text{, if } -\frac{2}{5} < t < \frac{3}{5} , \\ 1 & \text{otherwise.} \end{cases} \in C^1(I) .$$



$\|f_1 - s\|_{L^\infty([-1,1])} = O(h^4)$



$\|f_2 - s\|_{L^\infty([-1,1])} = O(h^2)$

Algebraic order of convergence in h = min { 1 + regularity of f, 4 }.

Theory [22]: $f \in C^4([t_0,t_n])$ ➔ $\|f-s\|_{L^\infty([t_0,t_n])} \le \frac{5}{384}h^4 \left\|f^{(4)}\right\|_{L^\infty([t_0,t_n])}$ ◇

Code 9.4.5: Spline approximation error

```
%22.06.2009 splineapprox.m
%Plot convergence of approximation error of cubic spline interpolation
%with respect to the meshwidth
%print the algebraic order of convergence in sup and L^2 norms
%
%inputs: f function to be interpolated
%df derivative of f, to be computed in a and b
%a, b left and right extremes of the interval
%N maximum number of subdomains

function splineapprox(f,df,a,b,N)
x = a:0.00025:b;      fv = feval(f,x);
dfa = feval(df,a);    dfb = feval(df,b);
err = [];
for j=2:N
```

```
  t = a:(b-a)/j:b;                                    %spline nodes
  y = [dfa,feval(f,t),dfb];
  %compute complete spline imposing exact first derivative at the extremes:
  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
  l2 = sqrt(0.5*dot(h,(d(1:end-1).^2+d(2:end).^2)));
  %columns of err = meshwidth, L^∞ 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('t');  ylabel('s(t)');
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)
```

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);
```

9.4.2 Shape Preserving Spline Interpolation

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

Find: grid $\mathcal{M} \subset [t_0,t_n]$ &
an interpolating quadratic spline function $s \in \mathcal{S}_{2,\mathcal{M}}$, $s(t_i)=y_i$, $i=0,\dots,n$ that preserves the "shape" of the data.

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 do four steps:

① "Shape-faithful" choice of slopes c_i, $i=0,\dots,n$ [26, 31] → Section 9.3

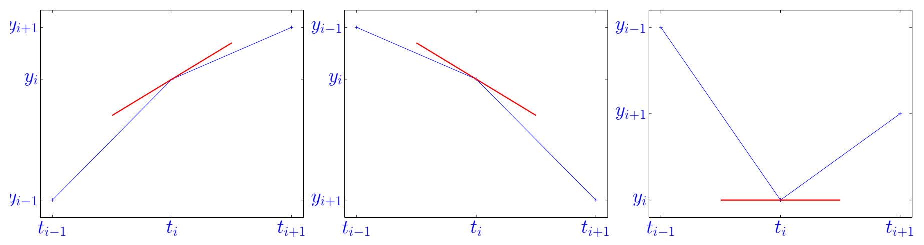
We fix the slopes c_i in the nodes using the harmonic mean of data slopes Δ_j, the final interpolant will be tangents to these segments in the points (t_i,y_i). If (t_i,y_i) is a local maximum or minimum of the data, c_j is set to zero.

$$\text{Limiter}\quad c_i := \begin{cases} \frac{2}{\Delta_i^{-1}+\Delta_{i+1}^{-1}} & \text{if } \operatorname{sign}(\Delta_i)=\operatorname{sign}(\Delta_{i+1}) , \\ 0 & \text{otherwise,} \end{cases} \qquad i=1,\dots,n-1 .$$

$$c_0 := 2\Delta_1 - c_1 , \qquad c_n := 2\Delta_n - c_{n-1} ,$$

where $\Delta_j = \frac{y_j-y_{j-1}}{t_j-t_{j-1}}$.

$$c_i = \frac{2}{\Delta_i^{-1}+\Delta_{i+1}^{-1}} \quad = \quad \text{harmonic mean of the slopes,} \qquad \text{see (9.3.4).}$$

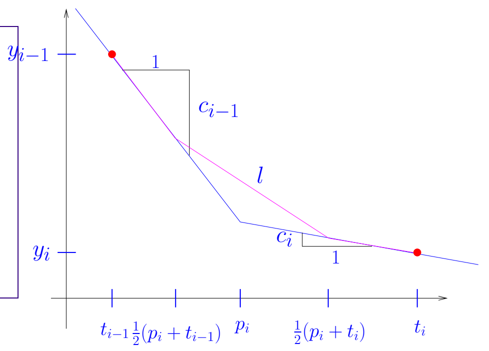


② Choice of "middle points" $p_i \in (t_{i-1}, t_i]$, $i = 1,\dots,n$:

$$p_i = \begin{cases} \text{intersection of the two straight lines} & \\ \quad \text{resp. through } (t_{i-1}, y_{i-1}), (t_i, y_i) & \text{if the intersection point belongs to } (t_{i-1}, t_i]\,, \\ \quad \text{with slopes } c_{i-1}, c_i & \\ \frac{1}{2}(t_{i-1} + t_i) & \text{otherwise}\,. \end{cases}$$

This points will be used to build the grid for the final quadratic spline:
$\mathcal{M} = \{t_0 < p_1 \le t_1 < p_2 \le \dots < p_n \le t_n\}$.

```
p = (t(1)-1)*ones(1,length(t)-1);
for j=1:n-1
  if (c(j) ~= c(j+1))
    p(j)=(y(j+1)-y(j)+...
        t(j)*c(j)-t(j+1)*c(j+1))/...
          (c(j)-c(j+1));
  end
  if ((p(j)<t(j))|(p(j)>t(j+1)))
    p(j) = 0.5*(t(j)+t(j+1));
  end;
end
```



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

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

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

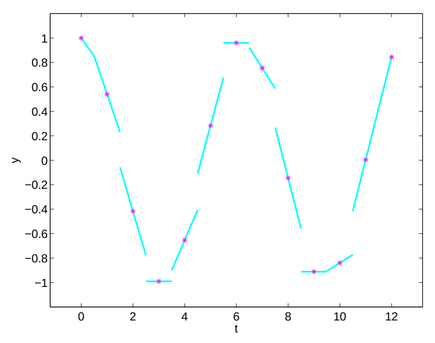
In each interval $(\frac{1}{2}(p_j + t_j), \frac{1}{2}(t_j + p_{j+1}))$ the spline corresponds to the segment of slope c_j passing through the data node (t_j, y_j).

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.

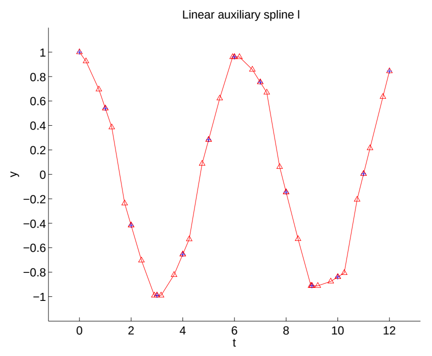
l "inherits" local monotonicity and curvature from the data.

Example 9.4.6 (Auxiliary construction for shape preserving quadratic spline interpolation).

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



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



Linear auxiliary spline l ◇

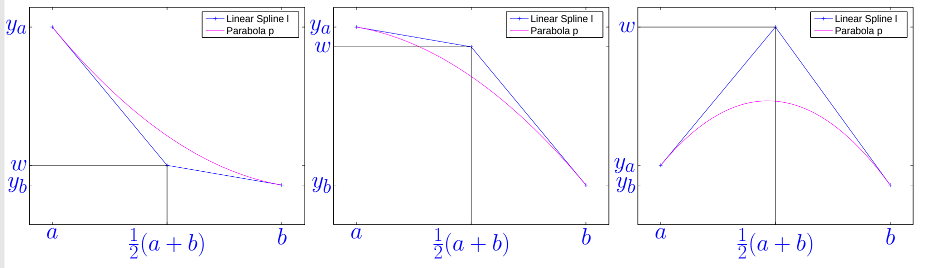
④ Local quadratic approximation / interpolation of l:

If g is a linear spline through three points (a, y_a), $(\frac{1}{2}(a+b), w)$, (b, y_b), $a < b$, $y_a, y_b, w \in \mathbb{R}$,

the parabola $p(t) := (y_a(b-t)^2 + 2w(t-a)(b-t) + y_b(t-a)^2)/(b-a)^2$, $a \le t \le b$,

satisfies $p(a) = y_a$, $p(b) = y_b$, $p'(a) = g'(a)$, $p'(b) = g'(b)$.

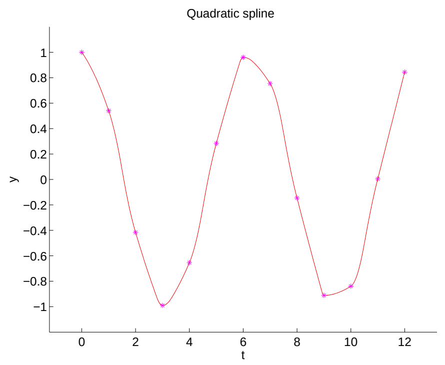
g monotonic increasing / decreasing $\Rightarrow$ p monotonic increasing / decreasing
g convex / concave $\Rightarrow$ p convex / concave

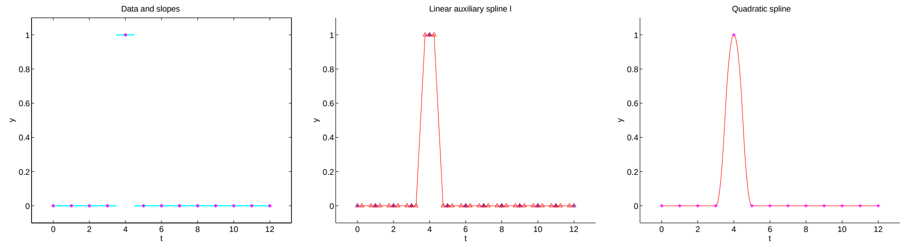


This implies that the final quadratic spline that passes through the points (t_j, y_j) with slopes c_j can be built locally as p using the linear spline l, in place of g.

Continuation of Ex. 9.4.6:

Interpolating quadratic spline ➜

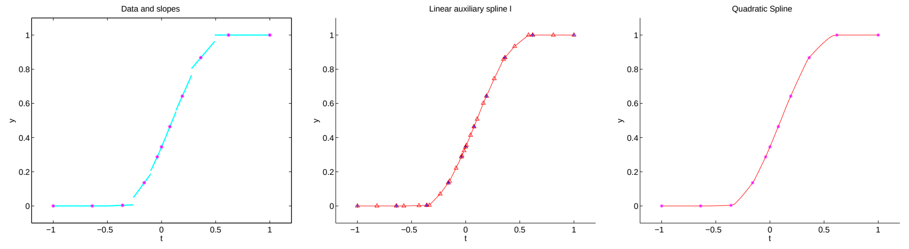




Shape preserving quadratic spline interpolation = local + not linear

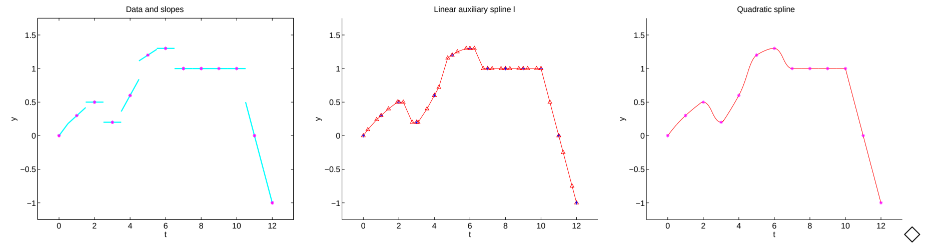
Example 9.4.7 (Shape preserving quadratic spline interpolation).

Data from Ex. 9.1.1:



Data from [26]:

t_i	0	1	2	3	4	5	6	7	8	9	10	11	12
y_i	0	0.3	0.5	0.2	0.6	1.2	1.3	1	1	1	1	0	-1



◇

Code 9.4.8: Step by step shape preserving spline interpolation

```
%22.06.2009 shape-pres-intp.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])

function shapepresintp(t,y)
```

```

n=length(t)-1;
%plot the data and prepare the figure
figure;
hplot(1)=plot(t,y,'k*-');
hold on;
newaxis=axis;
newaxis([3,4])=[newaxis(3)-0.5, newaxis(4)+0.5];
axis(newaxis);                %enlarge the vertical size of the plot
title('Data␣points␣-␣Press␣enter␣to␣continue')
plot(t,ones(1,n+1)* (newaxis(3)+0.25),'k.');
set(gca, 'XTick', t)
leg={'Data','Slopes','Middle␣points','Linear␣spline' ,'Sh.␣pres.␣
    spline'};
legend(hplot(1),leg{1});
pause;

%===== Step 1: choice of slopes =====
%shape-faithful slopes (c) in the nodes using harmonic mean of data slopes
%the final interpolant will be tangents to these segments

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);
legend([hplot(1), hplotsl(1)], leg{1:2});
pause;
```

```

%===== Step 2: choice of middle points =====
%fix points p(j) in [t(j), t(j+1)], depending on the slopes c(j), c(j+1)

disp('STEP␣2')
title('Middle␣points␣-␣Press␣enter␣to␣continue')
set(hplotsl,'linewidth',1)

p = (t(1)-1)*ones(1,length(t)-1);
for j=1:n
  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
  %check and repair if p(j) is outside its interval:
  if ((p(j)<t(j))||(p(j)>t(j+1)));   p(j) = 0.5*(t(j)+t(j+1));  end;
end

hplot(2)=plot(p,ones(1,n)*(newaxis(3)+0.25),'go');
legend([hplot(1), hplotsl(1), hplot(2)], leg{1:3});
pause;

%===== Step 3: auxiliary linear spline =====
%build the linear spline with nodes in:
```

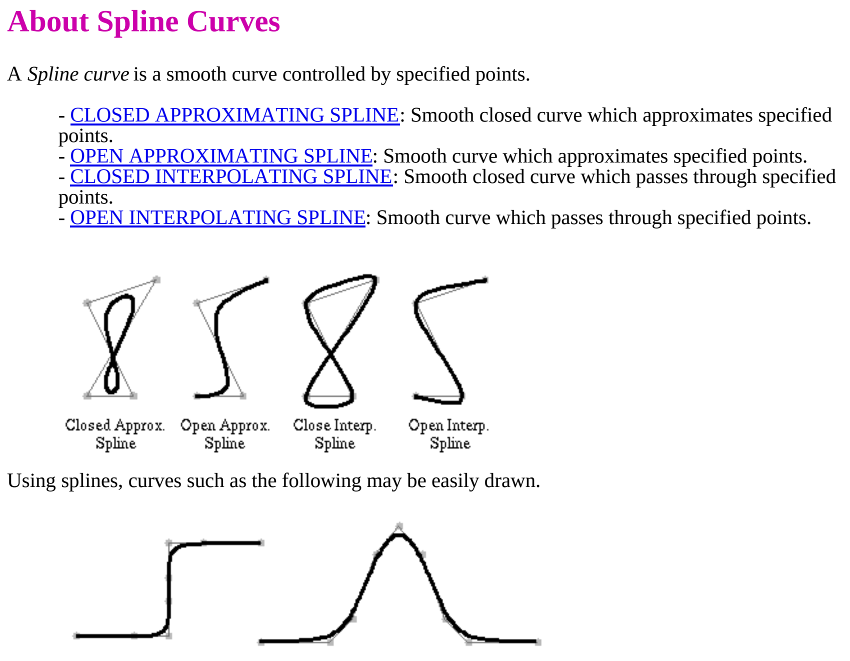
```
%-t(j)
%-the middle points between t(j) and p(j)
%-the middle points between p(j) and t(j+1)
%-t(j+1)
%and with slopes c(j) in t(j), for every j

disp('STEP␣3')
title('Auxiliary␣linear␣spline␣-␣Press␣enter␣to␣continue')

for j=1:n
  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-^');
  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^');
end
legend([hplot(1), hplotsl(1), hplot(2), hplot(3)], leg{1:4});
pause;

%===== 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);
    ya = yb;
    yb = y(j+1);
    w = y(j+1)+0.5*(p(j)-t(j+1))*c(j+1);
    x = (a:(b–a)/100:b);
    pb = (ya*(b–x).^2 + 2*w*(x–a).*(b–x)+yb*(x–a).^2)/((b–a)^2);
    plot(x,pb,'r–', 'linewidth',2);

    plot(p(j),ya,'go');
end

%replot initial nodes over the other plots:
plot(t,y,'k*');
%plot(p,yb,'go')
legend([hplot(1), hplotsl(1), hplot(2:4)], leg);
title('Shape␣preserving␣interpolation')
```

9.5 Bezier Techniques

Goal: Curves approximation (**not** interpolation) by piecewise polynomials

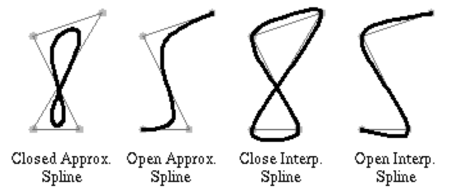
A page from the XFIG-manual (`http://www.xfig.org/`):



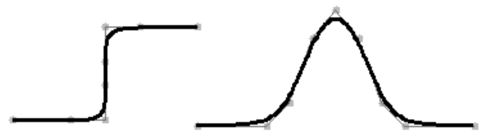
About Spline Curves

A *Spline curve* is a smooth curve controlled by specified points.

- CLOSED APPROXIMATING SPLINE: Smooth closed curve which approximates specified points.
- OPEN APPROXIMATING SPLINE: Smooth curve which approximates specified points.
- CLOSED INTERPOLATING SPLINE: Smooth closed curve which passes through specified points.
- OPEN INTERPOLATING SPLINE: Smooth curve which passes through specified points.



Using splines, curves such as the following may be easily drawn.



A proof of Weierstrass approximation theorem (see [9, Sect. 6.2]):

Theorem 9.5.1 (Approximation by Bernstein polynomials). *If* $f \in C([0,1])$ *and*

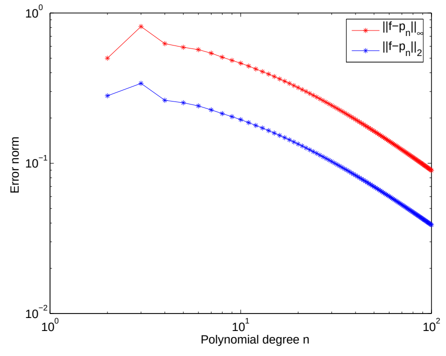
$$p_n(t) := \sum_{j=0}^{n} f(j/n)\binom{n}{j} t^j(1-t)^{n-j}\ , \quad 0 \le t \le 1\ ,$$

then $p_n \to f$ *uniformly for* $n \to \infty$. *When* $f \in C^m([0,1])$, *then* $p_n^{(k)} \to f^{(k)},\ 0 \le k \le m$, *uniformly for* $n \to \infty$.

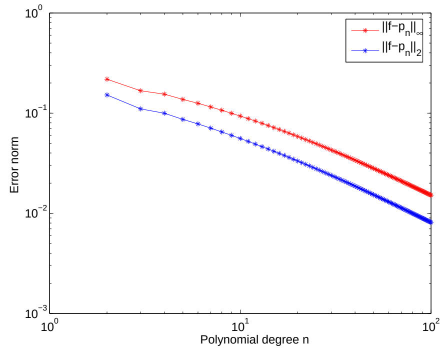
Example 9.5.1 (Bernstein approximation).

$$f_1(t) := \begin{cases} 0 & \text{, if } |2t-1| > \frac{1}{2}\ , \\ \frac{1}{2}(1+\cos(2\pi(2t-1))) & \text{otherwise} \end{cases} \quad , \qquad f_2(t) := \frac{1}{1+e^{-12(x-1/2)}}\ .$$

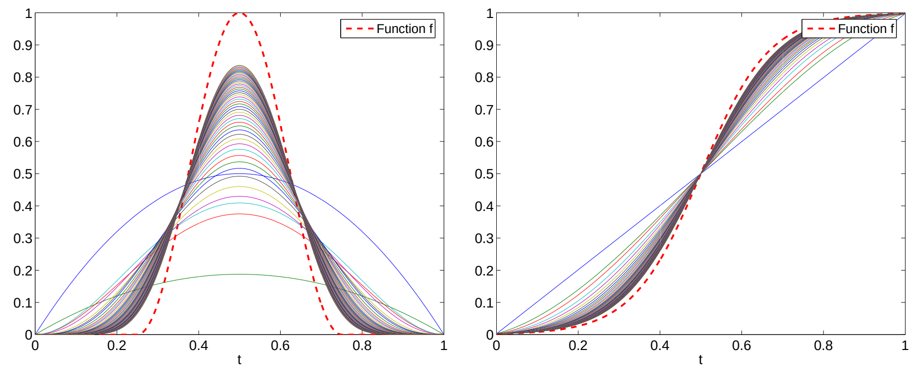
Norms of the approximation errors $f - p_n$, p_n from Thm. 9.5.1:



Bernstein approximation of f_1



Bernstein approximation of f_2



Bad approximation, but good “shape reproduction” by p_n

Quoted from [9, Sect. 6.3]: *Monotonic and convex functions yield monotonic and convex approximants, respectively. In a work, the Bernstein approximants mimic the behavior of the function to a remarkable degree. There is a price that must be paid for these beautiful approximation properties: the convergence of Bernstein polynomials is very slow.*

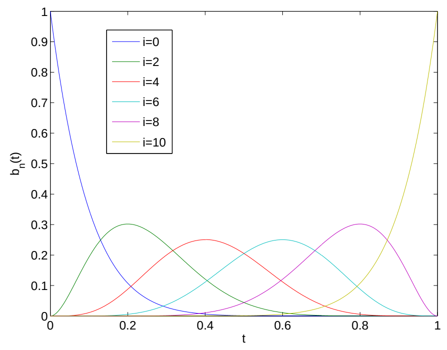
It is far slower than what can be achieved by other means. If f is bounded, then at a point t where $f'(t)$ exists and does not vanish, $p_n(t)$ converges to $f(t)$ precisely like C/n. This fact seems to have precluded any numerical application of Bernstein polynomials from having been made (1975!). Perhaps they will find application when the properties of the approximant in the large are of more importance than closeness of the approximation.

Definition 9.5.2 (Bernstein polynomials).
Bernstein polynomials of degree n:

$$b_{i,n}(t) := \binom{n}{i} t^i (1-t)^{n-i} \,, \quad i = 0, \dots, n \,.$$

($b_{i,n} :\equiv 0$ for $i < 0$, $i > n$)

Plot of Bernstein polynomials for $n = 10$ ➔



Lemma 9.5.3 (Properties of Bernstein polynomials).

(i) $t = 0$ *is a zero of order* i *of* $b_{i,n}$, $t = 1$ *is a zero of order* $n - i$ *of* $b_{i,n}$,

(ii) $\sum_{i=0}^{n} b_{i,n} \equiv 1$ *(partition of unity),*

(iii) $b_{i,n}(t) = (1-t)\, b_{i,n-1} + t\, b_{i-1,n-1}(t)$,

(iv) $b_{i,n}(t) \geq 0 \quad \forall\, 0 \leq t \leq 1$.

Lemma 9.5.3 (iii) provides an efficient computation of $b_{i,n}(t)$ (recursion)

Code 9.5.2: Bernstein polynomial

```
 function V = bernstein(n,x)
V = [ones(1,length(x));...
     zeros(n,length(x))];
for j=1:n
  for k=j+1:-1:2
    V(k,:) = x.*V(k-1,:) + (1-x).*V(k,:);
  end
  V(1,:) = (1-x).*V(1,:);
end
```

MATLAB file: given the function `f`, plot Bernstein approximating polynomials of degrees from 2 to n in the interval $[0, 1]$

Code 9.5.3: Bernstein approximation

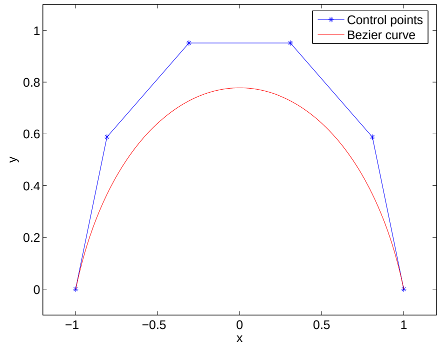
```
function dbsplot(f,n)
x=linspace(0,1,200);
figure('Name','Bernstein␣approximation');
plot(x,feval(f,x),'r--','linewidth',2);
legend('Function␣f');hold on;
X = [];
for k=2:n;     X = [X,bsapprox(f,k,x)'];     end;
plot(x,X,'-');   xlabel('t');   hold off;

function v = bsapprox(f,n,x);
fv = feval(f,(0:1/n:1));                  %n+1 line vector
V = bernstein(n,x);                       %(n+1)*(length(x)) matrix
v = fv*V;
```

Try:

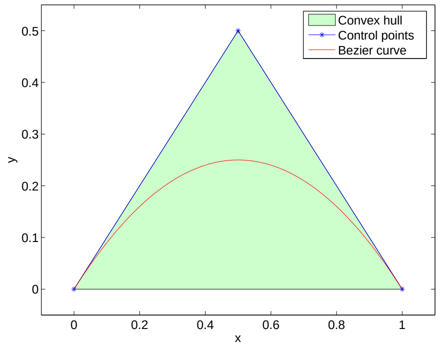
```
>> dbsplot(@(x) max(0,-(x-0.5).^2+0.08), 50);
>> dbsplot(@(x) atan((x-0.5)*2*pi), 50);
```

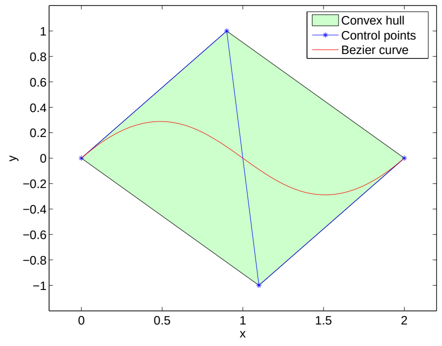
Definition 9.5.4 (Bézier curves)**.** *The Bézier curves in the control points* $\mathbf{d}_i \in \mathbb{R}^d$, $i = 0,\dots,n$, $d \in \mathbb{N}$ *is:*

$$\gamma : \begin{cases} [0,1] \mapsto \mathbb{R}^d \\ t \mapsto \sum_{i=0}^{n} \mathbf{d}_i\, b_{i,n}(t) \,. \end{cases}$$



- $\gamma(0) = \mathbf{d}_0$, $\gamma(1) = \mathbf{d}_n$ (interpolation property),
- $\gamma'(0) = n(\mathbf{d}_1 - \mathbf{d}_0)$, $\gamma'(1) = n(\mathbf{d}_n - \mathbf{d}_{n-1})$ (tangents),
- $\gamma_{[\mathbf{d}_0,\dots,\mathbf{d}_n]}(t) = (1-t)\gamma_{[\mathbf{d}_0,\dots,\mathbf{d}_{n-1}]}(t) + t\,\gamma_{[\mathbf{d}_1,\dots,\mathbf{d}_n]}(t)$ (recursion).





Lemma 9.5.3 (ii) $\Rightarrow$ $\gamma \subset \mathrm{convex}\{\mathbf{d}_0,\dots,\mathbf{d}_n\}$

Convex hull: $\mathrm{convex}\{d_0,\dots,d_n\} := \left\{ \mathbf{x} = \sum_{i=0}^{n} \lambda_i \mathbf{d}_i \,,\, 0 \le \lambda_i \le 1 \,,\, \sum_{i=0}^{n} \lambda_i = 1 \right\}$.

MATLAB file: draw the Bezier curve through the control points d ($2 \times n$ matrix), draw also the control-points and the convex hull

Code 9.5.4: Bezier curve

```
function bezcurv(d)
n = size(d,2)-1;                         % number of segments
figure('Name','Bezier␣curve');
k = convhull(d(1,:),d(2,:));             % k =indices of re-ordered vertices of the c.h.
fill(d(1,k),d(2,k),[0.8 1 0.8]);         % draw the c.h. using reordered vertices
hold on;
plot(d(1,:),d(2,:),'b-*');               % draw the nodes

x = 0:0.001:1;
V = bernstein(n,x);
bc = d*V;                                % compute the Bezier curve
plot(bc(1,:),bc(2,:),'r-');              % plot the Bezier curve

xlabel('x');
ylabel('y');
legend('Convex␣Hull','Control␣points','Bezier␣curve');
v = axis;                                % resize the plot
wx = v(2)-v(1);  wy = v(4)-v(3);
axis([v(1)-0.1*wx v(2)+0.1*wx v(3)-0.1*wy v(4)+0.1*wy]);hold off;
```

Try:

```
>> bezcurv([0 1 1 2;  0 1 -1 0]);>> bezcurv([0 1 1 0 0 1;  0 0 1 1 0
```

- piecewise Bézier curves ➔ spline curves,
- X-Splines & NURBS (CAGD, computer graphic),
- parametric B-splines = Bézier curves that approximate better the polygon defined by $\mathbf{d}_0,\dots,\mathbf{d}_n$

10 Numerical Quadrature

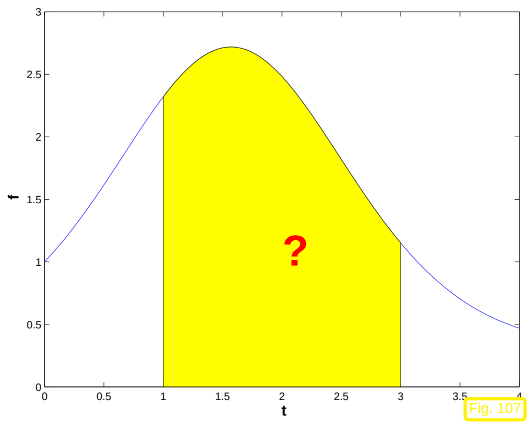
Numerical quadrature

= Approximate evaluation of $\int_\Omega f(\mathbf{x})\,\mathrm{d}\mathbf{x}$, integration domain $\Omega \subset \mathbb{R}^d$

Continuous function $f : \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}$ only available as `function y = f(x)` (point evaluation)

Special case $d = 1$: $\Omega = [a, b]$ (interval)

☞ Numerical quadrature methods are key building blocks for methods for the numerical treatment of partial differential equations.



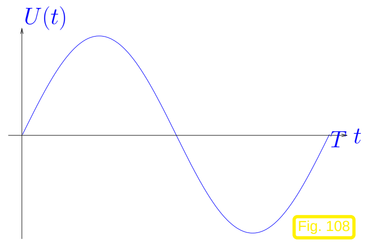
Numerical quadrature methods

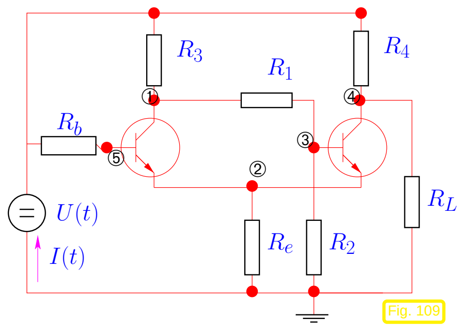
approximate

$$\int_a^b f(t)\,\mathrm{d}t$$

Example 10.0.1 (Heating production in electrical circuits).

Time-harmonic excitation:





Integrating power $P = UI$ over period $[0, T]$ yields heat production per period:

$$W_{\mathrm{therm}} = \int_0^T U(t)I(t)\,\mathrm{d}t\ , \quad \text{where} \quad I = I(U)\ .$$

`function I = current(U)` involves solving non-linear system of equations, see Ex. 3.0.1! ◇

10.1 Quadrature Formulas

n-point quadrature formula on $[a, b]$: (n-point quadrature rule)

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \sum_{j=1}^n \omega_j^n\, f(\xi_j^n)\ . \qquad (10.1.1)$$

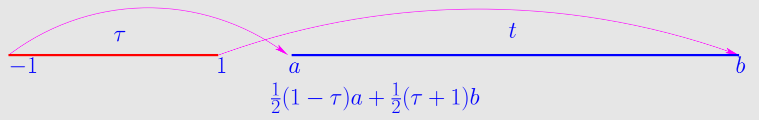
ω_j^n :	quadrature weights	$\in \mathbb{R}$	(*ger.:* Quadraturgewichte)
ξ_j^n :	quadrature nodes	$\in [a, b]$	(*ger.:* Quadraturknoten)

Remark 10.1.1 (Transformation of quadrature rules).

Given: quadrature formula $\left(\widehat{\xi}_j, \widehat{\omega}_j\right)_{j=1}^n$ on reference interval $[-1, 1]$

Idea: transformation formula for integrals

$$\int_a^b f(t)\,\mathrm{d}t = \tfrac{1}{2}(b-a)\int_{-1}^1 \widehat{f}(\tau)\,\mathrm{d}\tau\ , \quad \widehat{f}(\tau) := f(\tfrac{1}{2}(1-\tau)a + \tfrac{1}{2}(\tau+1)b)\ . \qquad (10.1.2)$$



▶ quadrature formula for general interval $[a,b]$, $a,b\in\mathbb{R}$:

$$\int_a^b f(t)\,\mathrm{d}t \approx \tfrac{1}{2}(b-a)\sum_{j=1}^{n}\widehat{\omega}_j\widehat{f}(\widehat{\xi}_j) = \sum_{j=1}^{n}\omega_j f(\xi_j) \quad\text{with}\quad \begin{aligned}\xi_j &= \tfrac{1}{2}(1-\widehat{\xi}_j)a+\tfrac{1}{2}(1+\widehat{\xi}_j)b\,,\\ \omega_j &= \tfrac{1}{2}(b-a)\widehat{\omega}_j\,.\end{aligned}$$

▶ A 1D quadrature formula on arbitrary intervals can be specified by providing its weights $\widehat{\omega}_j$ /nodes $\widehat{\xi}_j$ for integration domain $[-1,1]$. Then the above transformation is assumed.

Other common choice of reference interval: $[0,1]$

△

Inevitable for generic integrand:

$$\text{quadrature error} \qquad E(n) := \left|\int_a^b f(t)\,\mathrm{d}t - Q_n(f)\right|$$

Our focus (*cf.* interpolation error estimates, Sect. 8.4):

given families of quadrature rules $\{Q_n\}_n$ with quadrature weights $\left\{\omega_j^n,\ j=1,\dots,n\right\}_{n\in\mathbb{N}}$ and quadrature nodes $\left\{\xi_j^n,\ j=1,\dots,n\right\}_{n\in\mathbb{N}}$ we

study *asymptotic* behavior of quadrature error $E(n)$ for $n\to\infty$

Qualitative distinction, see (8.4.1):
▷ 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

Idea: replace integrand f with $p_{n-1}\in\mathcal{P}_{n-1}$ = polynomial interpolant of f for given interpolation nodes $\{t_0,\dots,t_{n-1}\}\subset[a,b]$

$$▶\quad \int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \int_a^b p_{n-1}(t)\,\mathrm{d}t\,. \tag{10.2.1}$$

Lagrange polynomials: $L_i(t) := \prod_{\substack{j=0\\ j\neq i}}^{n-1}\frac{t-t_j}{t_i-t_j}$, $i=0,\dots,n-1$ $\overset{(8.2.4)}{▶}$ $p_{n-1}(t)=\sum_{i=0}^{n-1}f(t_i)L_i(t)$.

$$\int_a^b p_{n-1}(t)\,\mathrm{d}t = \sum_{i=0}^{n-1} f(t_i)\int_a^b L_i(t)\,\mathrm{d}t \quad ▶ \quad \begin{aligned}&\text{nodes} && \xi_i = t_{i-1}\,,\\ &\text{weights} && \omega_i := \int_a^b L_{i-1}(t)\,\mathrm{d}t\,.\end{aligned} \tag{10.2.2}$$

Example 10.2.1 (Midpoint rule).

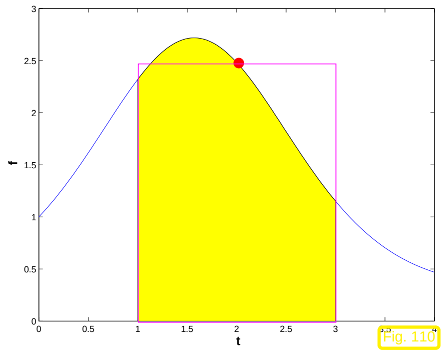


Fig. 110

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 10.2.2 (Newton-Cotes formulas).

Equidistant quadrature nodes $t_j := a+hj$, $h:=\frac{b-a}{n}$, $j=0,\dots,n$:

Symbolic computation of quadrature formulas on $[0,1]$ using MAPLE:

```
> newtoncotes := n -> factor(int(interp([seq(i/n, i=0..n)],
                    [seq(f(i/n), i=0..n)], z),z=0..1)):
```

• $n = 1$: Trapezoidal rule

```
> trapez := newtoncotes(1);
```

$$\widehat{Q}_{\text{trp}}(f) := \frac{1}{2}(f(0) + f(1)) \tag{10.2.3}$$

$$\left(\int_a^b f(t)\,\mathrm{d}t \approx \frac{b-a}{2}(f(a) + f(b)) \right)$$

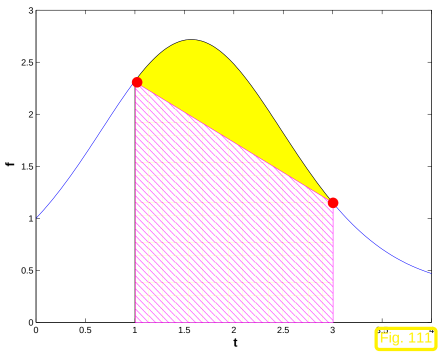


Fig. 111

• $n = 2$: Simpson rule

```
> simpson := newtoncotes(2);
```

$$\frac{h}{6}\left(f(0) + 4\,f(\tfrac{1}{2}) + f(1)\right) \quad \left(\int_a^b f(t)\,\mathrm{d}t \approx \frac{b-a}{6}\left(f(a) + 4\,f\left(\frac{a+b}{2}\right) + f(b)\right) \right) \tag{10.2.4}$$

• $n = 4$: 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 (→ Def. 2.5.5) **!**

Alternative: If t_j = Chebychev nodes (8.5.1) ➤ Clenshaw-Curtis rule

Remark 10.2.3 (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. 8.4.1:

$$f \in C^n([a,b]) \quad \Rightarrow \quad \left| \int_a^b f(t)\,\mathrm{d}t - Q_n(f) \right| \leq \frac{1}{n!}(b-a)^{n+1} \left\| f^{(n)} \right\|_{L^\infty([a,b])} . \tag{10.2.5}$$

(Separate estimates for Clenshaw-Curtis rules and analytic integrands)

10.3 Composite Quadrature

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 (10.2.5): for polynomial quadrature rule (10.2.1) 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: Increasse in total no. of f-evaluations incurred, which has to be balanced with the gain in accuracy to achieve optimal efficiency, *cf.* Sect. 3.3.3 and Sect. 10.6 for algorithmic realization.

Idea:
- Partition integration domain $[a,b]$ by mesh (grid, $\longrightarrow$ Sect.9.2) $\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_m = b\}$
- Apply quadrature formulas from Sects. 10.2, 10.4 on sub-intervals $I_j := [x_{j-1}, x_j]$, $j = 1, \ldots, m$, and sum up.

$\blacktriangleright$ composite quadrature rule

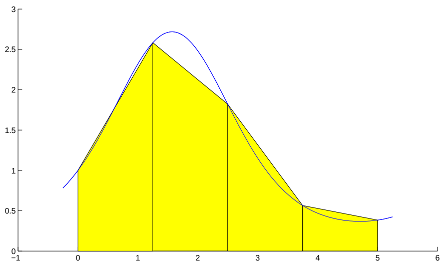
Analogy: global polynomial interpolation $\longleftrightarrow$ piecewise polynomial interpolation ($\longrightarrow$ Sect. 9.2)

Note: Here we only consider one and the same quadrature formula (local quadrature formula) applied on all sub-intervals.

Example 10.3.1 (Simple composite polynomial quadrature rules).

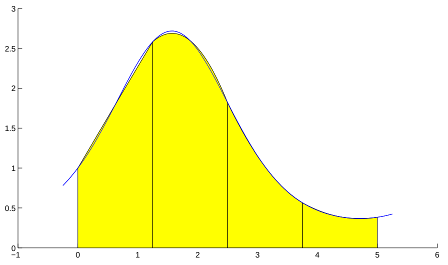
Composite trapezoidal rule, *cf.* (11.4.2)

$$\int_a^b f(t)\mathrm{d}t = \tfrac{1}{2}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \tfrac{1}{2}(x_{j+1} - x_{j-1})f(x_j) + \tfrac{1}{2}(x_m - x_{m-1})f(b)\ . \tag{10.3.2}$$



Composite Simpson rule, *cf.* (10.2.4)

$$\int_a^b f(t)\mathrm{d}t = \tfrac{1}{6}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \tfrac{1}{6}(x_{j+1} - x_{j-1})f(x_j) + \sum_{j=1}^{m} \tfrac{2}{3}(x_j - x_{j-1})f(\tfrac{1}{2}(x_j + x_{j-1})) + \tfrac{1}{6}(x_m - x_{m-1})f(b)\ . \tag{10.3.3}$$



Formulas (10.3.2), (10.3.3) directly suggest efficient implementation with minimal number of f-evaluations.

$\Diamond$

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: Predict decay of quadrature error as $m \to \infty$ (asymptotic perspective) for certain classes of integrands and “uniform” meshes.

Gauge for “quality” of a quadrature formula Q_n:

$$\text{Order}(Q_n) := \max\{n \in \mathbb{N}_0\colon\ Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \quad \forall p \in \mathcal{P}_n\} + 1$$

By construction: polynomial quadrature formulas (10.2.1) exact for $f \in \mathcal{P}_{n-1}$

$\Rightarrow$ n-point polynomial quadrature formula has at least order n

Remark 10.3.2 (Orders of simple polynomial quadrature formulas).

n		Order
0	midpoint rule	2
1	trapezoidal rule (11.4.2)	2
2	Simpson rule (10.2.4)	4
3	$\frac{3}{8}$-rule	4
4	Milne rule	6

$\triangle$

Focus: *asymptotic* behavior of quadrature error for

mesh width $\quad h := \max_{j=1,\ldots,m} |x_j - x_{j-1}| \to 0$

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 10.3.1 (Convergence of composite quadrature formulas)**.**
For a composite quadrature formula Q *based on a local quadrature formula of order* $p \in \mathbb{N}$ *holds*

$$\exists C > 0 \colon \quad \left| \int_I f(t)\,\mathrm{d}t - Q(f) \right| \le C h^p \left\| f^{(p)} \right\|_{L^\infty(I)} \quad \forall f \in C^p(I), \forall \mathcal{M} \; .$$

Proof. Apply interpolation error estimate (9.2.1). □

Example 10.3.3 (Quadrature errors for composite quadrature rules).

Composite quadrature rules based on

- trapezoidal rule (11.4.2) ➢ local order 2 (exact for linear functions),
- Simpson rule (10.2.4) ➢ local order 3 (exact for quadratic polynomials)

on equidistant mesh $\mathcal{M} := \{jh\}_{j=0}^{n}$, $h = 1/n$, $n \in \mathbb{N}$.

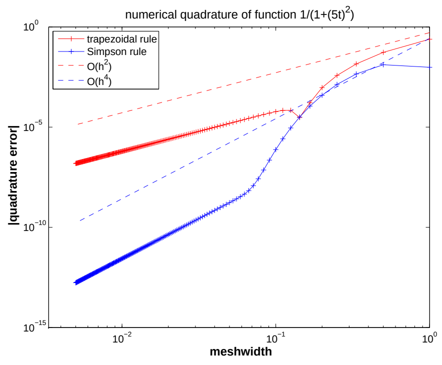
Code 10.3.4: composite trapezoidal rule (10.3.2)

```
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
```

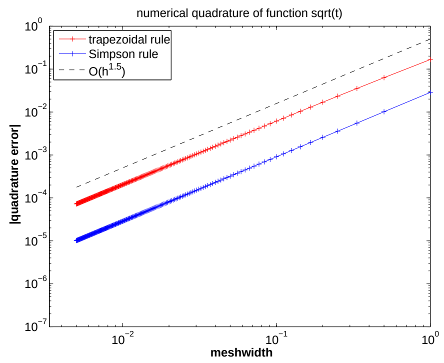
Code 10.3.5: composite Simpson rule (10.3.3)

```
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
```



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 $\to \alpha = 4$!?

➢ singular integrand f_2: $\alpha = 3/2$ for trapezoidal rule & Simpson rule !

(lack of) smoothness of integrand limits convergence !

Simpson rule: order = 4 ? investigate 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);
```

$$err := \left(\frac{1}{90}\left(D^{(4)}\right)(f)(0)\,h^5 + O\left(h^6\right), h, 6\right)$$

➢ Simpson rule is of order 4, indeed !

Code 10.3.6: errors of composite trapezoidal and Simpson rule

```
function comruleerrs()
% Numerical quadrature on [0,1]

figure('Name','1/(1+(5t)^2)');
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)-exact)),1
smp1 =
  polyfit(log(smres(end-100:end,1)),log(abs(smres(end-100:end,2)-exact)),1
print -dpsc2 '../PICTURES/compruleerr1.eps';

figure('Name','sqrt(t)');
exact = 2/3;
trres = trapezoidal(inline('sqrt(x)'),0,1,1:200);
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)-exact)),1
smp2 =
  polyfit(log(smres(end-100:end,1)),log(abs(smres(end-100:end,2)-exact)),1
print -dpsc2 '../PICTURES/compruleerr2.eps';
```

◇

Remark 10.3.7 (Removing a singularity by transformation).

Ex. 10.3.3 ➢ 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 (→ Ex. 10.3.3)

substitution $s = \sqrt{t}$: $$\int_0^b \sqrt{t} f(t)\,\mathrm{d}t = \int_0^{\sqrt{b}} \boxed{2s^2 f(s^2)}\,\mathrm{d}s\,. \qquad (10.3.4)$$

Then: Apply quadrature rule to smooth integrand

△

Example 10.3.8 (Convergence of equidistant trapezoidal rule).

Sometimes there are surprises: convergence of a composite quadrature rule is much better than predicted by the order of the local quadrature formula, see [?] for an explanation.

Equidistant trapezoidal rule (order 2), see (10.3.2)

$$\int_a^b f(t)\,\mathrm{d}t \approx T_m(f) := h\left(\tfrac{1}{2}f(a) + \sum_{k=1}^{m-1} f(kh) + \tfrac{1}{2}f(b)\right), \quad h := \frac{b-a}{m}. \tag{10.3.5}$$

Code 10.3.9: equidistant trapezoidal quadrature formula

```
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
```

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})

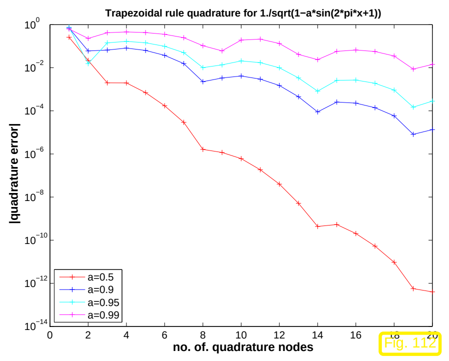


Fig. 112

quadrature error for $T_n(f)$ on $[0,1]$

exponential convergence !!

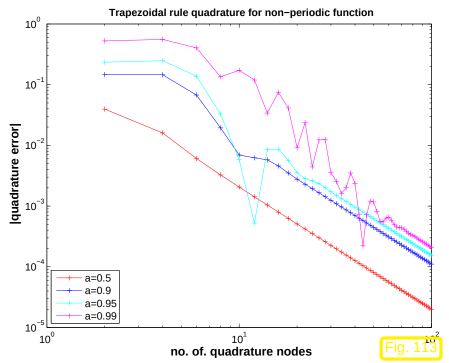


Fig. 113

quadrature error for $T_n(f)$ on $[0,\frac{1}{2}]$

merely algebraic convergence

Code 10.3.10: tracking error of equidistant trapezoidal quadrature formula

```
function traperr()
```

```

clear a;
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);

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);
title('{\bf␣Trapezoidal␣rule␣quadrature␣for␣
  1./sqrt(1-a*sin(2*pi*x+1))}','fontsize',12);

print -depsc2 '../PICTURES/traperr1.eps';
```

Explanation:

$$f(t) = e^{2\pi i k t} \quad \blacktriangleright \quad \begin{cases} \int_0^1 f(t)\,\mathrm{d}t = \begin{cases} 0 & \text{, if } k \neq 0 \,, \\ 1 & \text{, if } k = 0 \,. \end{cases} \\ T_m(f) = \frac{1}{m}\sum_{l=0}^{m-1} e^{\frac{2\pi i}{m}lk} \overset{(7.2.2)}{=} \begin{cases} 0 & \text{, if } k \notin m\mathbb{Z} \,, \\ 1 & \text{, if } k \in m\mathbb{Z} \,. \end{cases} \end{cases}$$

▶ Equidistant trapezoidal rule T_m is exact for trigonometric polynomials of degree $< 2m$ **!**

It takes sophisticated tools from complex analysis to conclude exponential convergence for analytic integrands from the above observation.

◇

Remark 10.3.11 (Choice of (local) quadrature weights).

Beyond local Newton-Cotes formulas from Ex. 10.2.2:

Given: arbitrary nodes $\xi_1, \dots, \xi_n$ for n-point (local) quadrature formula on $[a, b]$

Take cue from polynomial quadrature formulas: choice of weights ω_j according to (10.2.2) ensures order $\geq n$.

There is a more direct way without detour via Lagrange polynomials:

If $p_0, \dots, p_{n-1}$ is a basis of $\mathcal{P}_n$, then, thanks to the linearity of the integral and quadrature formulas,

$$Q_n(p_j) \underset{\uparrow}{=} \int_a^b p_j(t)\,\mathrm{d}t \quad \forall j = 0, \dots, n-1 \quad \Leftrightarrow \quad Q_n \text{ has order } \geq n \,. \tag{10.3.6}$$

➢ $n \times n$ linear system of equations, see (10.4.1) for an example:

$$\begin{pmatrix} p_0(\xi_1) & \dots & p_0(\xi_n) \\ \vdots & & \vdots \\ p_{n-1}(\xi_n) & \dots & p_{n-1}(\xi_n) \end{pmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_n \end{pmatrix} = \begin{pmatrix} \int_a^b p_0(t)\,\mathrm{d}t \\ \vdots \\ \int_a^b p_{n-1}(t)\,\mathrm{d}t \end{pmatrix} . \tag{10.3.7}$$

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 10.3.2 (Bound for order of quadrature formula)**.**
There is no n-point quadrature formula of order $2n+1$

Proof. (indirect) Assume there was an n-point quadrature formula with nodes $a \leq \xi_1 < \xi_2 < \dots < \xi_n \leq b$ of order $2n+1$.

➤ Construct polynomial $p(t) := \prod_{j=1}^n (t - \xi_j)^2 \in \mathcal{P}_{2n}$

$$\blacktriangleright \quad Q_n(p) = 0 \quad \text{but} \quad \int_a^b p(t)\,\mathrm{d}t > 0 \,.$$

Thus, the assumption leads to a contradiction. □

10.4 Gauss Quadrature

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. 8.1.1).

$$\Updownarrow$$

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.* (10.3.7):

$$Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \;\forall p \in \mathcal{P}_3 \quad\Leftrightarrow\quad Q_n(t^q) = \frac{1}{q+1}(b^{q+1} - a^{q+1}) \;, \quad q = 0,1,2,3 \;.$$

4 equations for weights ω_j and nodes ξ_j, $j = 1,2$ $\quad(a = -1, b = 1)$, *cf.* Rem. 10.3.11

$$\begin{aligned} \int_{-1}^{1} 1\,\mathrm{d}t = 2 = 1\omega_1 + 1\omega_2 \quad &, \quad \int_{-1}^{1} t\,\mathrm{d}t = 0 = \xi_1\omega_1 + \xi_2\omega_2 \\ \int_{-1}^{1} t^2\,\mathrm{d}t = \frac{2}{3} = \xi_1^2\omega_1 + \xi_2^2\omega_2 \quad &, \quad \int_{-1}^{1} t^3\,\mathrm{d}t = 0 = \xi_1^3\omega_1 + \xi_2^3\omega_2 \;. \end{aligned} \tag{10.4.1}$$

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: $\quad \left\{\omega_2 = 1, \omega_1 = 1, \xi_1 = 1/3\sqrt{3}, \xi_2 = -1/3\sqrt{3}\right\}$

quadrature formula:
$$\int_{-1}^{1} f(x)\,dx \approx f\left(\frac{1}{\sqrt{3}}\right) + f\left(-\frac{1}{\sqrt{3}}\right) \tag{10.4.2}$$

◇

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 $2n$ $\quad\Leftrightarrow\quad$ exact for polynomials $\in \mathcal{P}_{2n-1}$. $\qquad$ (10.4.3)

Define $\quad \bar{P}_n(t) := (t - \xi_1^n) \cdot \dots \cdot (t - \xi_n^n) \;, \quad t \in \mathbb{R} \quad\Rightarrow\quad \bar{P}_n \in \mathcal{P}_n \;.$

Note: $\bar{P}_n$ has leading coefficient $= 1$.

By assumption on the order of Q_n: $\quad$ for any $q \in \mathcal{P}_{n-1}$

$$\int_{-1}^{1} \underbrace{q(t)\bar{P}_n(t)}_{\in \mathcal{P}_{2n-1}}\,\mathrm{d}t \overset{(10.4.3)}{=} \sum_{j=1}^{n} \omega_j^n q(\xi_j^n)\,\underbrace{\bar{P}_n(\xi_j^n)}_{=0} = 0 \;.$$

$$\Rightarrow \quad \text{orthogonality} \quad \underbrace{\int_{-1}^{1} q(t)\bar{P}_n(t)\,\mathrm{d}t}_{L^2(]-1,1[)\text{-inner product of } q \text{ and } \bar{P}_n} = 0 \quad \forall q \in \mathcal{P}_{n-1} \;. \tag{10.4.4}$$

Recall: $\quad (f,g) \mapsto \int_a^b f(t)g(t)\,\mathrm{d}t$ is an inner product on $C^0([a,b])$

➢ Abstract techniques for vector spaces with inner product can be applied to polynomials, for instance Gram-Schmidt orthogonalization, *cf.* (4.2.6) ($\rightarrow$ linear algebra).

Abstract Gram-Schmidt orthogonalization: in a vector space with inner product $\cdot$ orthogonal vectors $q_0, q_1, \dots$ spanning the same subspaces as the linearly independent vectors $v_0, v_1, \dots$ are constructed recursively via

$$q_{n+1} := v_{n+1} - \sum_{k=0}^{n} \frac{v_{n+1} \cdot q_k}{q_k \cdot q_k} q_k \quad , \quad q_0 := v_0 \;.$$

➢ Construction of $\bar{P}_n$ by Gram-Schmidt orthogonalization of monomial basis $\{1, t, t^2, \dots, t^{n-1}\}$ of $\mathcal{P}_{n-1}$ w.r.t. $L^2(]-1,1[)$-inner product:

$$\bar{P}_0(t) := 1 \;, \quad \bar{P}_{n+1}(t) = t^n - \sum_{k=0}^{n} \frac{\int_{-1}^{1} t^n \bar{P}_k(t)\,\mathrm{d}t}{\int_{-1}^{1} \bar{P}_k^2(t)\,\mathrm{d}t} \cdot \bar{P}_k(t) \tag{10.4.5}$$

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 10.4.1 (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. 10.3.11 provides a quadrature formula of order $2n$ *on* $[-1,1]$.

Proof. Conclude from the orthogonality of the $\bar{P}_n$ that $\{\bar{P}_k\}_{k=0}^n$ is a basis of $\mathcal{P}_n$ and

$$\int_{-1}^{1} h(t)\bar{P}_n(t)\,\mathrm{d}t = 0 \quad \forall h \in \mathcal{P}_{n-1}\,. \tag{10.4.6}$$

Recall division of polynomials with remainder (Euclid's algorithm $\longrightarrow$ Course "Diskrete Mathematik"):

for any $p \in \mathcal{P}_{2n-1}$

$$p(t) = h(t)\bar{P}_n(t) + r(t)\,, \quad \text{for some} \quad h \in \mathcal{P}_{n-1},\, r \in \mathcal{P}_{n-1}\,. \tag{10.4.7}$$

Apply this representation to the integral:

$$\int_{-1}^{1} p(t)\,\mathrm{d}t = \underbrace{\int_{-1}^{1} h(t)\bar{P}_n(t)\,\mathrm{d}t}_{=0 \text{ by (10.4.6)}} + \int_{-1}^{1} r(t)\,\mathrm{d}t \overset{(*)}{=} \sum_{j=1}^{m} \omega_j^n r(\xi_j^n)\,, \tag{10.4.8}$$

$(*)$: by choice of weights according to Rem. 10.3.11 Q_n is exact for polynomials of degree $\le n-1$!

By choice of nodes as zeros of $\bar{P}_n$ using (10.4.6):

$$\sum_{j=1}^{m} \omega_j^n p(\xi_j^n) \overset{(10.4.7)}{=} \sum_{j=1}^{m} \omega_j^n h(\xi_j^n) \underbrace{\bar{P}_n(\xi_j^n)}_{=0} + \sum_{j=1}^{m} \omega_j^n r(\xi_j^n) \overset{(10.4.8)}{=} \int_{-1}^{1} p(t)\,\mathrm{d}t\,. \qquad \square$$

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 10.4.2 (Legendre polynomials).
The n-th Legendre polynomial P_n is defined by

- $P_n \in \mathcal{P}_n$,
- $\int_{-1}^{1} P_n(t)q(t)\,\mathrm{d}t = 0 \;\forall q \in \mathcal{P}_{n-1}$,
- $P_n(1) = 1$.

Legendre polynomials $P_0,\dots,P_5$ ➢

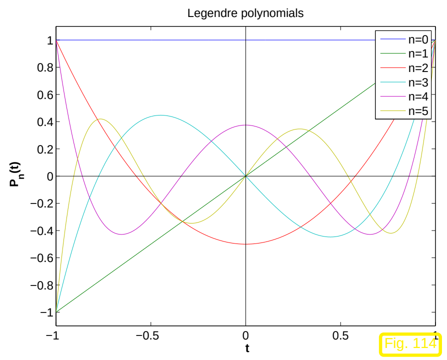


Fig. 114

Notice: the polynomials $\bar{P}_n$ defined by (10.4.5) and the Legendre polynomials P_n of Def. 10.4.2 (merely) *differ* by a constant factor!

► Gauss points ξ_j^n = zeros of Legendre polynomial P_n

Note: the above considerations, recall (10.4.4), show that the nodes of an n-point quadrature formula

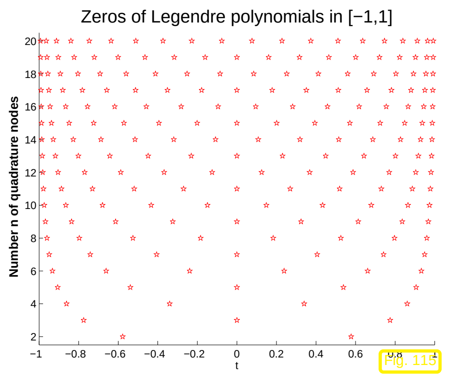
of order $2n$ on $[-1,1]$ must agree with the zeros of $L^2(]-1,1[)$-orthogonal polynomials.

► 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 (10.4.5) may lie outside $[-1,1]$.
In principle $\bar{P}_n$ could also have less than n real zeros.

The next lemma shows that all this cannot happen.



◁ Obviously:

Lemma 10.4.3 (Zeros of Legendre polynomials).

P_n has n distinct zeros in $]-1,1[$.

Zeros of Legendre polynomials = Gauss points

Proof. (indirect) Assume that P_n has only $m < n$ zeros $\zeta_1,\dots,\zeta_m$ in $]-1,1[$ *at which it changes sign*. Define

$$q(t) := \prod_{j=1}^{m}(t-\zeta_j) \quad\Rightarrow\quad qP_n \geq 0 \quad\text{or}\quad qP_n \leq 0\,.$$

$$\Rightarrow \quad \int_{-1}^{1} q(t)P_n(t)\,\mathrm{d}t \neq 0\,.$$

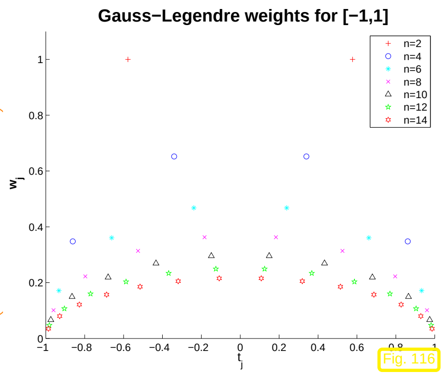
As $q \in \mathcal{P}_{n-1}$, this contradicts (10.4.6). □

Quadrature formula from Thm. 10.4.1: Gauss-Legendre quadrature
(nodes ξ_j^n = Gauss points)

Obviously ▷

Lemma 10.4.4. *(Positivity of Gauss-Legendre quadrature weights)*

The weights of Gauss-Legendre quadrature formulas are positive.



Proof. Writing ξ_j^n, $j=1,\dots,n$, for the nodes (Gauss points) of the n-point Gauss-Legendre quadrature formula, $n\in\mathbb{N}$, we define

$$q_k(t) = \prod_{\substack{j=1\\ j\neq k}}^{n}(t-\xi_j^n)^2 \quad\Rightarrow\quad q_k \in \mathcal{P}_{2n-2}\,.$$

This polynomial is integrated exactly by the quadrature rule: since $q_k(\xi_j^n)=0$ for $j\neq k$

$$0 < \int_{-1}^{1} q(t)\,\mathrm{d}t = \omega_k^n \underbrace{q(\xi_k^n)}_{>0}\,,$$

where ω_j^n are the quadrature weights. □

Remark 10.4.2 (3-Term recursion for Legendre polynomials).

Note: polynomials $\bar{P}_n$ from (10.4.5) are uniquely characterized by the two properties (try a proof!)

- $\bar{P}_n \in \mathcal{P}_n$ with leading coefficient 1: $\bar{P}(t) = t^n + \dots$,
- $\int_{-1}^{1}\bar{P}_k(t)\bar{P}_j(t)\,\mathrm{d}t = 0$, if $j\neq k$ ($L^2(]-1,1[)$-orthogonality).

➣ same polynomials $\bar{P}_n$ by another Gram-Schmidt orthogonalization procedure, *cf.* (10.4),

$$\bar{P}_{n+1}(t) = t\bar{P}_n(t) - \sum_{k=0}^{n}\frac{\int_{-1}^{1}\tau\bar{P}_n(\tau)\bar{P}_k(\tau)\,\mathrm{d}\tau}{\int_{-1}^{1}\bar{P}_k^2(\tau)\,\mathrm{d}\tau}\cdot\bar{P}_k(t)$$

By orthogonality (10.4.6) the sum collapses, since $\int_{-1}^{1}\tau\bar{P}_n(\tau)\bar{P}k(\tau)\,\mathrm{d}\tau = \int_{-1}^{1}\bar{P}_n(\tau)\underbrace{(\tau\bar{P}k(\tau))}_{\in\mathcal{P}_{k+1}}\,\mathrm{d}\tau = 0$, if $k+1<n$:

$$\bar{P}_{n+1}(t) = t\bar{P}_n(t) - \frac{\int_{-1}^{1}\tau\bar{P}_n(\tau)\bar{P}_n(\tau)\,\mathrm{d}\tau}{\int_{-1}^{1}\bar{P}_n^2(\tau)\,\mathrm{d}\tau}\cdot\bar{P}_n(t) - \frac{\int_{-1}^{1}\tau\bar{P}_n(\tau)\bar{P}_{n-1}(\tau)\,\mathrm{d}\tau}{\int_{-1}^{1}\bar{P}_{n-1}^2(\tau)\,\mathrm{d}\tau}\cdot\bar{P}_{n-1}(t)\,. \qquad (10.4.9)$$

After rescaling (tedious!): 3-term recursion for Legendre polynomials

$$P_{n+1}(t) := \frac{2n+1}{n+1}tP_n(t) - \frac{n}{n+1}P_{n-1}(t)\quad,\quad P_0 := 1\,,\quad P_1(t) := t\,. \qquad (10.4.10)$$

Efficient and *stable* evaluation of Legendre polynomials by means of 3-term recursion (10.4.10)

Code 10.4.3: computing Legendre polynomials
```
function V= legendre(n,x)
V = ones(size(x)); V = [V; x];
for j=1:n-1
  V = [V; ((2*j+1)/(j+1)).*x.*V(end,:) - j/(j+1)*V(end-1,:)]; end
```

Comments on Code 10.4.2:

☛ return value: matrix $\mathbf{V}$ with $(\mathbf{V})_{ij} = P_i(x_j)$

☛ line 2: takes into account initialization of Legendre 3-term recursion (10.4.10)

Remark 10.4.4 (Computing Gauss nodes and weights).

Compute nodes/weights of Gaussian quadrature by solving an eigenvalue problem! (Golub-Welsch algorithm [16, Sect. 3.5.4])

In codes: ξ_j, ω_j from tables!

Code 10.4.5: Golub-Welsch algorithm

```matlab
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);
for i=1:n, ev(:,i) = ev(:,i)./norm(ev(:,i)); end
x=diag(ew); w=(2*(ev(1,:).*ev(1,:)))';
```

Justification: rewrite 3-term recurrence (10.4.10) for scaled Legendre polynomials $\widetilde{P}_n = \frac{1}{\sqrt{n+1/2}} P_n$

$$t\widetilde{P}_n(t) = \underbrace{\frac{n}{\sqrt{4n^2-1}}}_{=:\beta_n} \widetilde{P}_{n-1}(t) + \underbrace{\frac{n+1}{\sqrt{4(n+1)^2-1}}}_{=:\beta_{n+1}} \widetilde{P}_{n+1}(t) \,. \tag{10.4.11}$$

For fixed $t \in \mathbb{R}$ (10.4.11) can be expressed as

$$t \underbrace{\begin{pmatrix} \widetilde{P}_0(t) \\ \widetilde{P}_1(t) \\ \vdots \\ \widetilde{P}_{n-1}(t) \end{pmatrix}}_{=:\mathbf{p}(t)\in\mathbb{R}^n} = \underbrace{\begin{pmatrix} 0 & \beta_1 & & & & \\ \beta_1 & 0 & \beta_2 & & & \\ & \beta_2 & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & & 0 & \beta_{n-1} \\ & & & & \beta_{n-1} & 0 \end{pmatrix}}_{=:\mathbf{J}_n\in\mathbb{R}^{n,n}} \begin{pmatrix} \widetilde{P}_0(t) \\ \widetilde{P}_1(t) \\ \vdots \\ \widetilde{P}_{n-1}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \beta_n \widetilde{P}_n(t) \end{pmatrix}$$

$$\blacktriangleright \quad \widetilde{P}_n(\xi) = 0 \quad \Leftrightarrow \quad \xi\mathbf{p}(\xi) = \mathbf{J}_n\mathbf{p}(\xi) \,.$$

▶ 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 10.4.4. The computation of the weights in line 6 of Code 10.4.4 is explained in [16, Sect. 3.5.4].

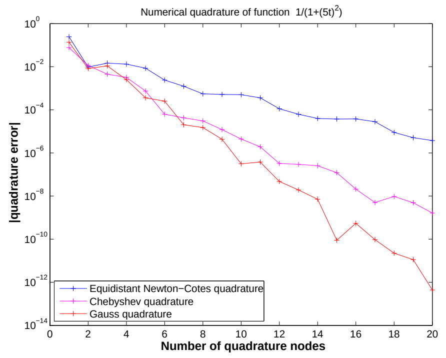
Example 10.4.6 (Error of (non-composite) quadratures).

Code 10.4.7: important polynomial quadrature rules

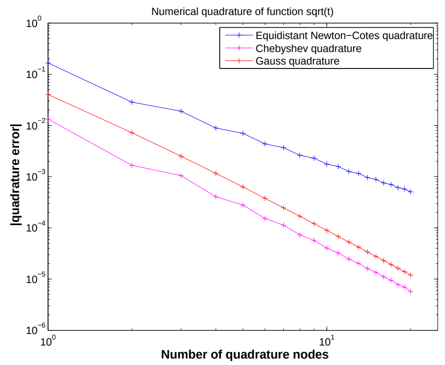
```matlab
function res = numquad(f,a,b,N,mode)
```

```matlab
% 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, Gauss

if (nargin < 5), mode = 'equidistant'; end

res = [];

if strcmp(mode,'Gauss')
  for deg=1:N
    [gx,w] = gaussQuad(deg);
    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
  p = (N+1:-1:1);
  w = (b.^p - a.^p)./p;
  for deg=1:N
    if strcmp(mode,'Chebychev')
      x = 0.5*(b-a)*cos((2*(0:deg)+1)/(2*deg+2)*pi)+0.5*(a+b);
```

```matlab
    else
      x = (a:(b-a)/deg:b);
    end
    y = feval(f,x);
    poly = polyfit(x,y,deg);
    res = [res; deg, dot(w(N+1-deg:N+1),poly)];
  end
end
```



quadrature error, $f_1(t) := \frac{1}{1+(5t)^2}$ on $[0,1]$



quadrature error, $f_2(t) := \sqrt{t}$ on $[0,1]$

Asymptotic behavior of quadrature error $\epsilon_n := \left|\int_0^1 f(t)\,\mathrm{d}t - Q_n(f)\right|$ for "$n \to \infty$":

➣ exponential convergence $\epsilon_n \approx O(q^n)$, $0 < q < 1$, for C^∞-integrand f_1 $\leadsto$: 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 $t = 0$ $\leadsto$ Newton-Cotes quadrature : $\alpha \approx 1.8$, Clenshaw-Curtis quadrature : $\alpha \approx 2.5$, Gauss-Legendre quadrature : $\alpha \approx 2.7$

Code 10.4.8: tracking errors on 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',...
       'Clenshaw-Curtis␣quadrature',...
```

```
       'Gauss␣quadrature',3);
eqdp1 = polyfit(eqdres(:,1),log(abs(eqdres(:,2)-exact)),1)
chbp1 = polyfit(chbres(:,1),log(abs(chbres(:,2)-exact)),1)
gaup1 = polyfit(gaures(:,1),log(abs(gaures(:,2)-exact)),1)
print -depsc2 '../PICTURES/numquaderr1.eps';

figure('Name','sqrt(t)');
exact = 2/3;
eqdres = numquad(inline('sqrt(x)'),0,1,N,'equidistant');
chbres = numquad(inline('sqrt(x)'),0,1,N,'Chebychev');
gaures = numquad(inline('sqrt(x)'),0,1,N,'Gauss');
loglog(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);
axis([1 25 0.000001 1]);
title('Numerical␣quadrature␣of␣function␣sqrt(t)');
xlabel('{\bf␣Number␣of␣quadrature␣nodes}','fontsize',14);
ylabel('{\bf␣|quadrature␣error|}','fontsize',14);
legend('Equidistant␣Newton-Cotes␣quadrature',...
       'Clenshaw-Curtis␣quadrature',...
       'Gauss␣quadrature',1);
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 Oscillatory Integrals

10.6 Adaptive Quadrature

Example 10.6.1 (Rationale for adaptive quadrature).

Consider composite trapezoidal rule (10.3.2) 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" ▷

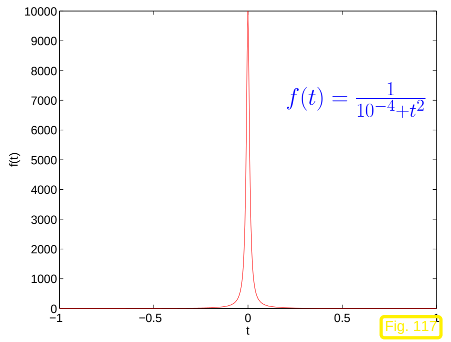


Fig. 117

◇

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* → [13, Sect. 9.7]

Idea: local error estimation by comparing local results of two quadrature formulas Q_1, Q_2 of *different* order → local error estimates

heuristics: error$(Q_2) \ll$ error(Q_1) $\Rightarrow$ error$(Q_1) \approx Q_2(f) - Q_1(f)$.

Now: Q_1 = trapezoidal rule (order 2) $\leftrightarrow$ Q_2 = Simpson rule (order 4)

Given: mesh $\mathcal{M} := \{a = x_0 < x_1 < \cdots < x_m = b\}$

❶ (error estimation)

For $I_k = [x_{k-1}, x_k], k = 1,\ldots,m$ (midpoints $p_k := \frac{1}{2}(x_{k-1} + x_k)$)

$$\mathtt{EST}_k := \left| \underbrace{\frac{h_k}{6}(f(x_{k-1}) + 4f(p_k) + f(x_k))}_{\text{Simpson rule}} - \underbrace{\frac{h_k}{4}(f(x_{k-1}) + 2f(p_k) + f(x_k))}_{\text{trapezoidal rule on split mesh interval}} \right| . \qquad (10.6.1)$$

❷ (Termination)

Simpson rule on $\mathcal{M}$ $\Rightarrow$ preliminary result I

$$\text{If} \quad \sum_{k=1}^{m} \mathtt{EST}_k \leq \mathtt{RTOL} \cdot I \quad (RTOL := \text{prescribed tolerance}) \quad \Rightarrow \quad \textbf{STOP} \qquad (10.6.2)$$

❸ (local mesh refinement)

$$\mathcal{S} := \{k \in \{1,\ldots,m\} \colon \mathtt{EST}_k \geq \eta \cdot \frac{1}{m}\sum_{j=1}^{m} \mathtt{EST}_j\} \;, \quad \eta \approx 0.9 \;. \qquad (10.6.3)$$

▶ new mesh: $\mathcal{M}^* := \mathcal{M} \cup \{p_k \colon k \in \mathcal{S}\}$.

Then continue with step ❶ and mesh $\mathcal{M} \leftarrow \mathcal{M}^*$.

Non-optimal recursive MATLAB implementation:

Code 10.6.2: h-adaptive numerical quadrature
```
1  function I = adaptquad(f,M,rtol,abstol)
2  h = diff(M);                                  %
3  mp = 0.5*(M(1:end-1)+M(2:end));               %
4  fx = f(M); fm = f(mp);                        %
5  trp_loc  = h.*(fx(1:end-1)+2*fm+fx(2:end))/4; %
6  simp_loc = h.*(fx(1:end-1)+4*fm+fx(2:end))/6; %
7  I = sum(simp_loc);                            %
8  est_loc  = abs(simp_loc -trp_loc);            %
9  err_tot  = sum(est_loc);                      %
10 %
11 if ((err_tot > rtol*abs(I)) and (err_tot > abstol))
12   refcells = find(est_loc > 0.9*sum(est_loc)/length(h));
13   I = adaptquad(f,sort([M,mp(refcells)]),rtol,abstol); %
14 end
```

Comments on Code 10.6.1:

- 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 (10.3.2),
- line 6: local simpson rule (10.2.4),
- 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 (10.6.3) to the mesh and continue.

Example 10.6.3 (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 10.6.1

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.

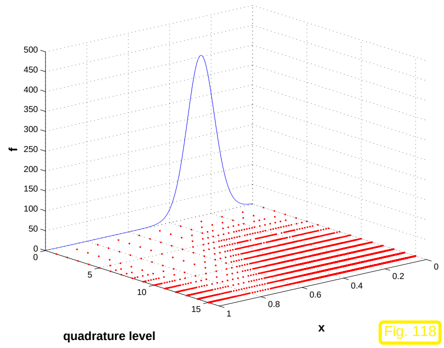


Fig. 118

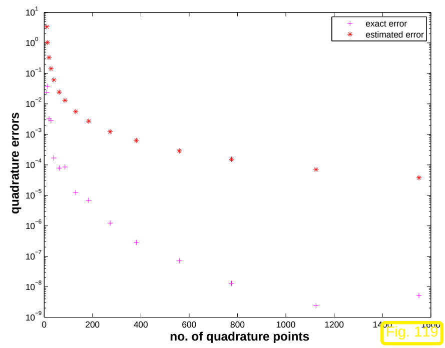


Fig. 119

- approximate $\int_0^1 \min\{\exp(6\sin(2\pi t)), 100\}\,\mathrm{d}t$, initial mesh as above

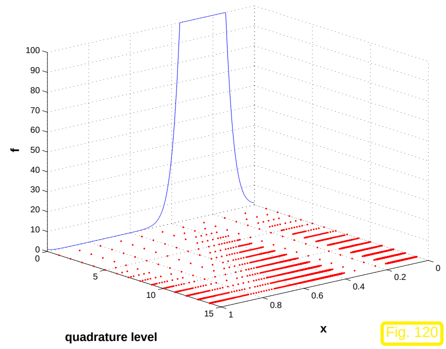


Fig. 120

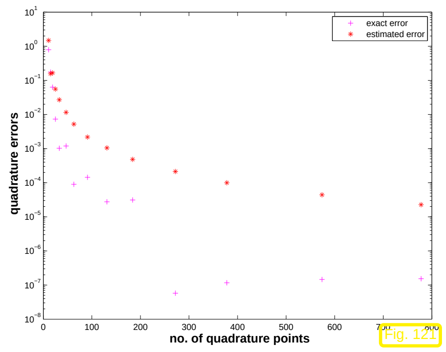


Fig. 121

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 (10.6.1)

However, the important information we want to glean from EST_k is about the *distribution* of the quadrature error.

◇

Remark 10.6.4 (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

△

10.7 Multidimensional Quadrature

Part III

Integration of Ordinary Differential Equations

11 Single Step Methods

11.1 Initial value problems (IVP) for ODEs

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 [40, Sect. 5.6, 5.7, 6.5].

Example 11.1.1 (Growth with limited resources). [1, Sect. 1.1]

$y : [0,T] \mapsto \mathbb{R}$: bacterial population density as a function of time

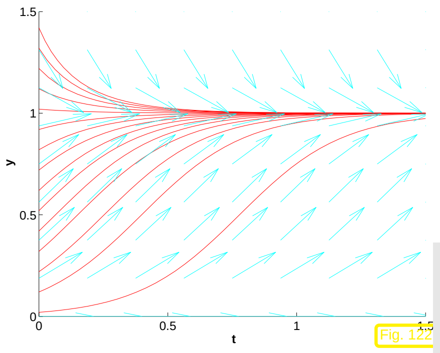
Model: autonomous logistic differential equations

$$\dot{y} = f(y) := (\alpha - \beta y)\, y \qquad (11.1.1)$$

✎ 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 (11.1.1), when provided with an initial value $y(0)$.



Solution for different $y(0)$ $(\alpha, \beta = 5)$

By separation of variables
➥ solution of (11.1.1)
for $y(0) = y_0 > 0$

$$y(t) = \frac{\alpha y_0}{\beta y_0 + (\alpha - \beta y_0)\exp(-\alpha t)} , \qquad (11.1.2)$$

for all $t \in \mathbb{R}$

$f'(y^*) = 0$ for $y^* \in \{0, \alpha/\beta\}$, which are the stationary points for the ODE (11.1.1). If $y(0) = y^*$ the solution will be constant in time.

◇

Example 11.1.2 (Predator-prey model). [1, Sect. 1.1] & [21, Sect. 1.1.1]

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:

$$\begin{aligned}\dot{u} &= (\alpha - \beta v)u\\ \dot{v} &= (\delta u - \gamma)v\end{aligned} \quad\leftrightarrow\quad \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad\text{with}\quad \mathbf{y} = \begin{pmatrix}u\\v\end{pmatrix}, \quad \mathbf{f}(\mathbf{y}) = \begin{pmatrix}(\alpha-\beta v)u\\(\delta u-\gamma)v\end{pmatrix}. \qquad (11.1.3)$$

population sizes:

$u(t) \longrightarrow$ no. of prey at time t,
$v(t) \longrightarrow$ no. of predators at time t

vector field f for Lotka-Volterra ODE ▷

Solution curves are trajectories of particles carried along by velocity field $\mathbf{f}$.

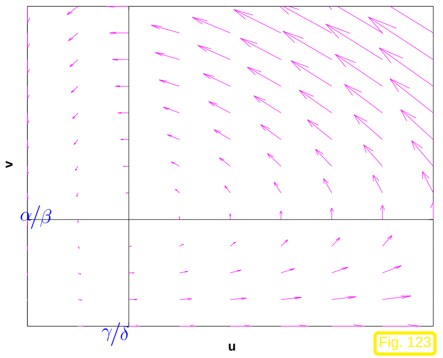


Fig. 123

Parameter values for Fig. 123: $\alpha = 2, \beta = 1, \delta = 1, \gamma = 1$

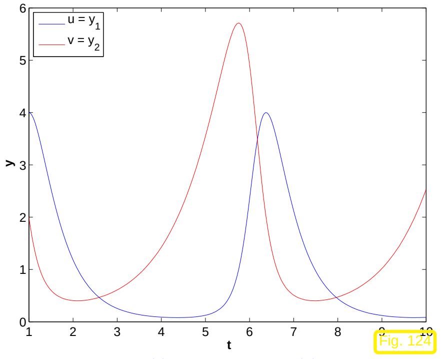


Fig. 124

Solution $\binom{u(t)}{v(t)}$ for $\mathbf{y}_0 := \binom{u(0)}{v(0)} = \binom{4}{2}$

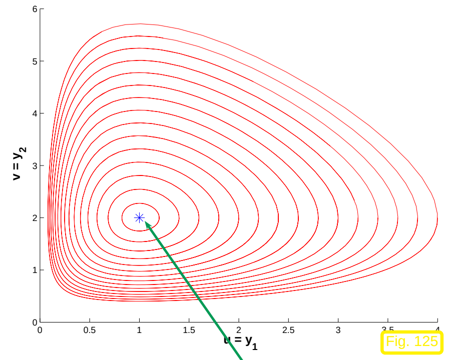


Fig. 125

Solution curves for (11.1.3)

Parameter values for Figs. 125, 124: $\alpha = 1, \beta = 1, \delta = 1, \gamma = 2$

stationary point

◇

Example 11.1.3 (Heartbeat model). $\longrightarrow$ [10, p. 655]

State of heart described by quantities:

$l = l(t)$	$\hat{=}$	length of muscle fiber
$p = p(t)$	$\hat{=}$	electro-chemical potential

Phenomenological model:
$$\begin{aligned}\dot{l} &= -(l^3 - \alpha l + p)\,,\\ \dot{p} &= \beta l\,,\end{aligned} \qquad (11.1.4)$$

with parameters:

α	$\hat{=}$	pre-tension of muscle fiber
β	$\hat{=}$	(phenomenological) feedback parameter

This is the so-called Zeeman model: it is a phenomenological model entirely based on macroscopic observations without relying on knowledge about the underlying molecular mechanisms.

Vector fields and solutions for different choices of parameters:

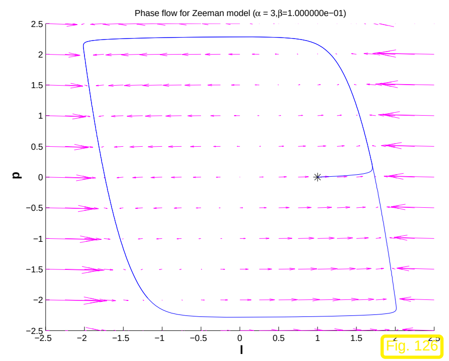


Fig. 126

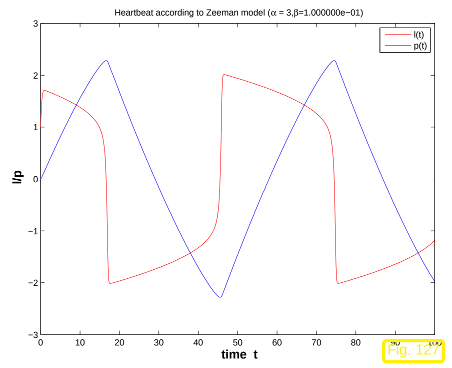
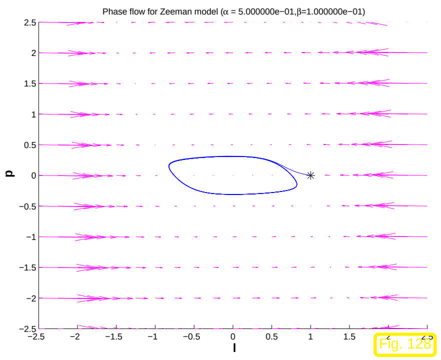
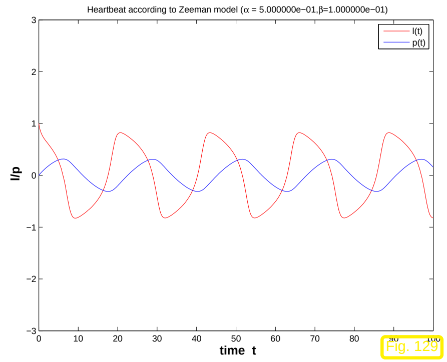


Fig. 127





Observation: $\alpha \ll 1$ ➤ atrial fibrillation

◇

Abstract mathematical description:

Initial value problem (IVP) for first-order ordinary differential equation (ODE): ($\rightarrow$ [40, Sect. 5.6])

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \; . \tag{11.1.5}$$

- $\mathbf{f} : I \times D \mapsto \mathbb{R}^d \hat{=}$ 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 ➢ initial conditions

For $d > 1$ $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ can be viewed as a system of ordinary differential equations:

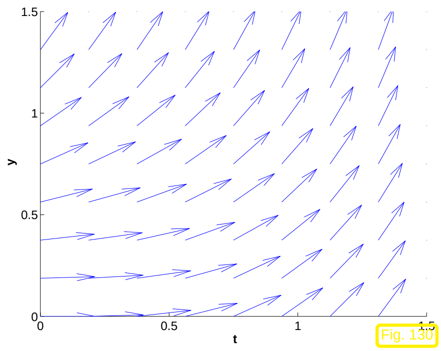
$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \Longleftrightarrow \quad \begin{pmatrix} y_1 \\ \vdots \\ y_d \end{pmatrix} = \begin{pmatrix} f_1(t, y_1, \ldots, y_d) \\ \vdots \\ f_d(t, y_1, \ldots, y_d) \end{pmatrix} .$$

Example 11.1.4 (Tangent field and solution curves).

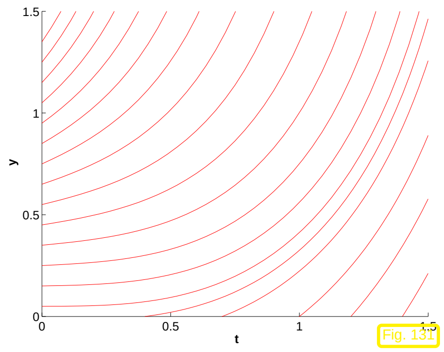
Riccati differential equation

scalar ODE

$$\dot{y} = y^2 + t^2 \;\; ➤ \;\; d = 1, \quad I, D = \mathbb{R}^+ \; . \tag{11.1.6}$$



tangent field



solution curves

solution curves run tangentially to the tangent field in each point of the extended state space.

◇

Terminology: $\mathbf{f} = \mathbf{f}(\mathbf{y})$, r.h.s. does not depend on time ➔ $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ is autonomous ODE

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 above led to autonomous ODEs.

Remark 11.1.5 (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 $= 1$

$$\mathbf{z}(t) := \begin{pmatrix} \mathbf{y}(t) \\ t \end{pmatrix} = \begin{pmatrix} \mathbf{z}' \\ z_{d+1} \end{pmatrix} : \quad \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad \leftrightarrow \quad \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}) \; , \quad \mathbf{g}(\mathbf{z}) := \begin{pmatrix} \mathbf{f}(z_{d+1}, \mathbf{z}') \\ 1 \end{pmatrix} \; .$$

△

Example 11.1.6 (Transient circuit simulation).

Transient nodal analysis, *cf.* Ex. 2.0.1:
Kirchhoff current law

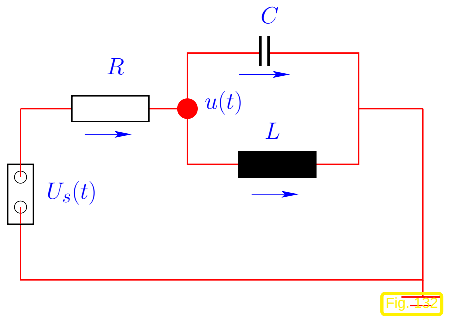
$$i_R(t) - i_L(t) - i_C(t) = 0 \; . \qquad (11.1.7)$$

Transient constitutive relations:

$$i_R(t) = R^{-1} u_R(t) \; , \qquad (11.1.8)$$

$$i_C(t) = C \frac{du_C}{dt}(t) \; , \qquad (11.1.9)$$

$$u_L(t) = L \frac{di_L}{dt}(t) \; . \qquad (11.1.10)$$



Given: source voltage $U_s(t)$

Differentiate (11.1.7) w.r.t. t and plug in constitutive relations for circuit elements:

$$R^{-1}\frac{du_R}{dt}(t) - L^{-1}u_L(t) - C\frac{d^2u_C}{dt^2}(t) = 0 \; .$$

We follow the policy of nodal analysis and express all voltages by potential differences between nodes of the circuit. For this simple circuit there is only one node with unknown potential, see Fig. 132. Its

time-dependent potential will be denoted by $u(t)$.

$$R^{-1}(\dot{U}_s(t) - \dot{u}(t)) - L^{-1}u(t) - C\frac{d^2u}{dt^2}(t) = 0 \; .$$

➤ autonomous 2nd-order ordinary differential equation:

$$C\ddot{u} + R^{-1}\dot{u} + L^{-1}u = R^{-1}\dot{U}_s \; .$$

◇

Remark 11.1.7 (From higher order ODEs to first order systems).

Ordinary differential equation of order $n \in \mathbb{N}$:

$$\boxed{\mathbf{y}^{(n)} = \mathbf{f}(t, \mathbf{y}, \dot{\mathbf{y}}, \dots, \mathbf{y}^{(n-1)})} \; . \qquad (11.1.11)$$

✎ Notation: superscript $^{(n)}$ $\hat{=}$ n-th temporal derivative t

➤ Conversion into 1st-order ODE (system of size nd)

$$\mathbf{z}(t) := \begin{pmatrix} \mathbf{y}(t) \\ \mathbf{y}^{(1)}(t) \\ \vdots \\ \mathbf{y}^{(n-1)}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_n \end{pmatrix} \in \mathbb{R}^{dn} : \quad (11.1.11) \quad \leftrightarrow \quad \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}) \; , \quad \mathbf{g}(\mathbf{z}) := \begin{pmatrix} \mathbf{z}_2 \\ \mathbf{z}_3 \\ \vdots \\ \mathbf{z}_n \\ \mathbf{f}(t, \mathbf{z}_1, \dots, \mathbf{z}_n) \end{pmatrix} . \qquad (11.1.12)$$

Note: n initial values $\mathbf{y}(t_0), \dot{\mathbf{y}}(t_0), \dots, \mathbf{y}^{(n-1)}(t_0)$ required!

△

Basic assumption: right hand side $\mathbf{f} : I \times D \mapsto \mathbb{R}^d$ locally Lipschitz continuous in $\mathbf{y}$

Definition 11.1.1 (Lipschitz continuous function). *(→ [40, Def. 4.1.4])*
$\mathbf{f} : \Omega \mapsto \mathbb{R}^d$ is Lipschitz continuous (in the second argument), if

$$\exists L > 0: \quad \|\mathbf{f}(t,\mathbf{w}) - \mathbf{f}(t,\mathbf{z})\| \le L \,\|\mathbf{w} - \mathbf{z}\| \quad \forall (t,\mathbf{w}), (y,\mathbf{z}) \in \Omega \; .$$

Definition 11.1.2 (Local Lipschitz continuity). *(→ [40, Def. 4.1.5])*
$\mathbf{f} : \Omega \mapsto \mathbb{R}^d$ is locally Lipschitz continuous, if

$$\forall (t,\mathbf{y}) \in \Omega: \quad \exists \delta > 0, \; L > 0:$$
$$\|\mathbf{f}(\tau,\mathbf{z}) - \mathbf{f}(\tau,\mathbf{w})\| \le L \,\|\mathbf{z} - \mathbf{w}\|$$
$$\forall \mathbf{z}, \mathbf{w} \in D: \; \|\mathbf{z} - \mathbf{y}\| < \delta, \; \|\mathbf{w} - \mathbf{y}\| < \delta, \forall \tau \in I: |t - \tau| < \delta \; .$$

✎ Notation: $D_{\mathbf{y}}\mathbf{f}$ $\hat{=}$ derivative of $\mathbf{f}$ w.r.t. state variable (= Jacobian $\in \mathbb{R}^{d,d}$!)

A simple criterion for local Lipschitz continuity:

Lemma 11.1.3 (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. 11.1.2).

Theorem 11.1.4 (Theorem of Peano & Picard-Lindelöf)**.** *[1, Satz II(7.6)], [40, Satz 6.5.1]*
If $\mathbf{f} : \hat{\Omega} \mapsto \mathbb{R}^d$ *is locally Lipschitz continuous (→ Def. 11.1.2) then for all initial conditions* $(t_0, \mathbf{y}_0) \in \hat{\Omega}$ *the IVP* (11.1.5) *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 11.1.8 (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.

$\triangle$

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. 11.1.5 and Thm. 11.1.4: we consider only

$$\text{autonomous IVP:} \quad \boxed{\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad , \quad \mathbf{y}(0) = \mathbf{y}_0} \; , \tag{11.1.13}$$

with locally Lipschitz continuous (→ Def. 11.1.2) right hand side $\mathbf{f}$.

Assumption 11.1.5 (Global solutions)**.**
All solutions of (11.1.13) *are global:* $J(\mathbf{y}_0) = \mathbb{R}$ *for all* $\mathbf{y}_0 \in D$.

Change of perspective: fix "time of interest" $t \in \mathbb{R} \setminus \{0\}$

➣ mapping $\Phi^t : \begin{cases} D & \mapsto D \\ \mathbf{y}_0 & \mapsto \mathbf{y}(t) \end{cases}$, $t \mapsto \mathbf{y}(t)$ solution of IVP (11.1.13) ,

is well-defined mapping of the state space into itself, by Thm. 11.1.4 and Ass. 11.1.5

Now, we may also let t vary, which spawns a *family* of mappings $\{\Phi^t\}$ of the state space into itself. However, it can also be viewed as a mapping with two arguments, a time t and an initial state value $\mathbf{y}_0$!

Definition 11.1.6 (Evolution operator)**.**
Under Assumption 11.1.5 the mapping

$$\Phi : \begin{cases} \mathbb{R} \times D & \mapsto D \\ (t, \mathbf{y}_0) & \mapsto \Phi^t \mathbf{y}_0 := \mathbf{y}(t) \end{cases} ,$$

where $t \mapsto \mathbf{y}(t) \in C^1(\mathbb{R}, \mathbb{R}^d)$ *is the unique (global) solution of the IVP* $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$, *is the evolution operator for the autonomous ODE* $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

Note: $t \mapsto \Phi^t \mathbf{y}_0$ describes the solution of $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ for $\mathbf{y}(0) = \mathbf{y}_0$ (a trajectory)

Remark 11.1.9 (Group property of autonomous evolutions).

Under Assumption 11.1.5 the evolution operator gives rise to a group of mappings $D \mapsto D$:

$$\Phi^s \circ \Phi^t = \Phi^{s+t} \quad , \quad \Phi^{-t} \circ \Phi^t = Id \quad \forall t \in \mathbb{R} \, . \tag{11.1.14}$$

This is a consequence of the uniqueness theorem Thm. 11.1.4. 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 $s + t$.

$\triangle$

11.2 Euler methods

Targeted: initial value problem (11.1.5)

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \, . \tag{11.1.5}$$

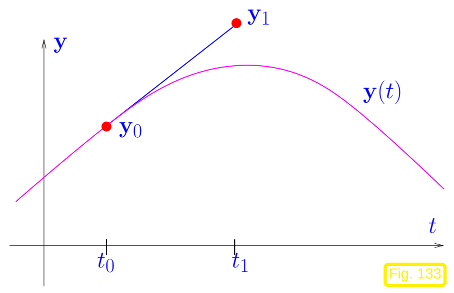
Sought: *approximate* solution of (11.1.5) on $[t_0, T]$ up to final time $T \neq t_0$

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.)

Idea: ❶ timestepping: successive approximation of evolution on *small* intervals $[t_{k-1}, t_k]$, $k = 1, \dots, N$, $t_N := T$,
❷ approximation of solution on $[t_{k-1}, t_k]$ by tangent curve to current initial condition.



<span style="color:red">explicit Euler method</span> (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!

Example 11.2.1 (Visualization of explicit Euler method).

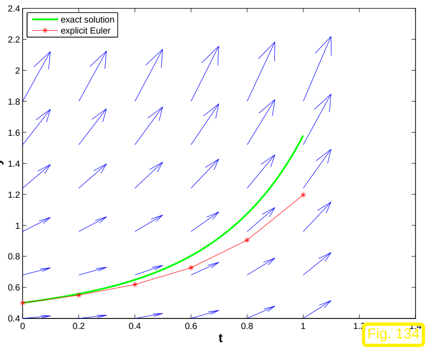
IVP for Riccati differential equation, see Ex. 11.1.4

$$\dot{y} = y^2 + t^2 . \qquad (11.1.6)$$

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



◇

Formula: explicit Euler method generates a *sequence* $(\mathbf{y}_k)_{k=0}^N$ by the recursion

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_k, \mathbf{y}_k) \;, \quad k = 0, \dots, N-1 \quad ,} \qquad (11.2.1)$$

with local (size of) timestep (stepsize) $h_k := t_{k+1} - t_k$.

Remark 11.2.2 (Explicit Euler method as difference scheme).

(11.2.1) by approximating derivative $\frac{d}{dt}$ by forward difference quotient on a (temporal) mesh $\mathcal{M} := \{t_0, t_1, \dots, t_N\}$:

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \longleftrightarrow \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f(t_k, \mathbf{y}_h(t_k)) \;, \quad k = 0, \dots, N-1 \, . \qquad (11.2.2)$$

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).

△

Why forward difference quotient and not backward difference quotient? Let's try!

On (temporal) mesh $\mathcal{M} := \{t_0, t_1, \dots, t_N\}$ we obtain

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad \longleftrightarrow \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f(t_{k+1}, \mathbf{y}_h(t_{k+1})) \;, \quad k = 0, \dots, N-1 \, . \qquad (11.2.3)$$

Backward difference quotient

This leads to another simple timestepping scheme analoguous to (11.2.1):

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) \;, \quad k = 0, \dots, N-1} \;, \qquad (11.2.4)$$

with local timestep (stepsize) $h_k := t_{k+1} - t_k$.

(11.2.4) = implicit Euler method

Note: (11.2.4) requires solving of a (possibly non-linear) system of equations to obtain $\mathbf{y}_{k+1}$!
(➤ Terminology "implicit")

Remark 11.2.3 (Feasibility of implicit Euler timestepping).

Consider autonomous ODE and assume continuously differentiable right hand side: $\mathbf{f} \in C^1(D, \mathbb{R}^d)$.

(11.2.4) $\leftrightarrow$ 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: for *sufficiently small* $|h|$ the equation $G(h,\mathbf{z}) = 0$ defines a continuous function $\mathbf{z} = \mathbf{z}(h)$.

△

How to interpret the sequence $(\mathbf{y}_k)_{k=0}^N$ from (11.2.1)?

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 $\mathbf{y}_k$, $k = 0,\dots,N$

Technique: *interpolation*, see Ch. 8

Simplest option: piecewise linear interpolation (→ Sect. 9.2.1) ➜ Euler polygon, see Fig. 134.

Abstract single step methods

Recall Euler methods for autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$:

$$\begin{aligned}\text{explicit Euler:}\quad & \mathbf{y}_{k+1} = \mathbf{y}_k + h_k\mathbf{f}(\mathbf{y}_k)\,,\\ \text{implicit Euler:}\quad & \mathbf{y}_{k+1}:\quad \mathbf{y}_{k+1} = \mathbf{y}_k + h_k\mathbf{f}(\mathbf{y}_{k+1})\,.\end{aligned}$$

Both formulas provide a mapping

$$(\mathbf{y}_k, h_k) \mapsto \mathbf{\Psi}(h,\mathbf{y}_k) := \mathbf{y}_{k+1}\,. \tag{11.2.5}$$

Recall the interpretation of the $\mathbf{y}_k$ as approximations of $\mathbf{y}(t_k)$:

$$\mathbf{\Psi}(h,\mathbf{y}) \approx \mathbf{\Phi}^h\mathbf{y}\,, \tag{11.2.6}$$

where $\mathbf{\Phi}$ is the evolution operator (→ Def. 11.1.6) 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 $\mathbf{\Phi}$ for an ODE by a mapping of the type (11.2.5).

➜ mapping $\mathbf{\Psi}$ from (11.2.5) is called discrete evolution.

Vice versa: a mapping $\mathbf{\Psi}$ as in (11.2.5) defines a single step method.

Definition 11.2.1 (Single step method (for autonomous ODE)).
Given a discrete evolution $\mathbf{\Psi} : \Omega \subset \mathbb{R} \times D \mapsto \mathbb{R}^d$, *an initial state* $\mathbf{y}_0$, *and a temporal mesh* $\mathcal{M} := \{t_0 < t_1 < \dots < t_N = T\}$ *the recursion*

$$\mathbf{y}_{k+1} := \mathbf{\Psi}(t_{k+1} - t_k, \mathbf{y}_k)\,,\quad k = 0,\dots,N-1\,, \tag{11.2.7}$$

defines a single step method (SSM, ger.: Einschrittverfahren) for the autonomous IVP $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$.

Procedural view of discrete evolutions:

$\mathbf{\Psi}^h\mathbf{y} \longleftrightarrow$ `function y1 = esvstep(h,y0)` .
(`function y1 = esvstep(@(y) rhs(y),h,y0)`)

✎ Notation: $\mathbf{\Psi}^h\mathbf{y} := \mathbf{\Psi}(h,\mathbf{y})$

Concept of single step method according to Def. 11.2.1 can be generalized to non-autonomous ODEs, which leads to recursions of the form:

$$\mathbf{y}_{k+1} := \mathbf{\Psi}(t_k, t_{k+1}, \mathbf{y}_k)\,,\quad k = 0,\dots,N-1\,,$$

for discrete evolution defined on $I \times I \times D$.

Remark 11.2.4 (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.

$\triangle$

11.3 Convergence of single step methods

Important issue: accuracy of approximation $\mathbf{y}_k \approx \mathbf{y}(t_k)$ **?**

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 11.3.1 (Speed of convergence of Euler methods).

- IVP for Riccati ODE (11.1.6) on $[0,1]$
- explicit Euler method (11.2.1) 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 $\mathrm{err}_h = O(h)$

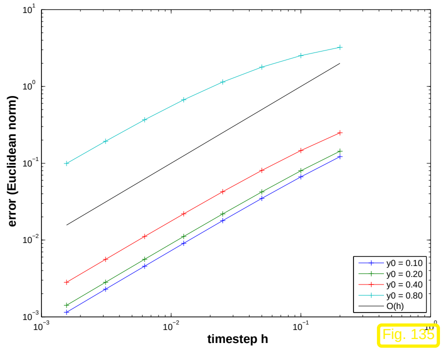


Fig. 135

- IVP for logistic ODE, see Ex. 11.1.1

$$\dot{y} = \lambda y(1-y) \quad , \quad y(0) = 0.01\ .$$

- Explicit and implicit Euler methods (11.2.1)/(11.2.4) with uniform timestep $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.
- Monitored: Error at final time $E(h) := |y(1) - y_N|$

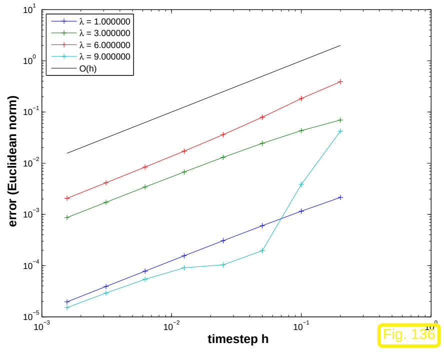


Fig. 136

explicit Euler method

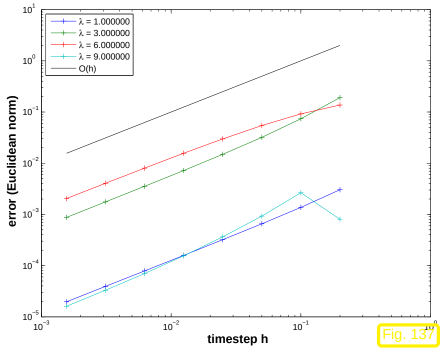


Fig. 137

implicit Euler method

► $O(h)$ algebraic convergence in both cases

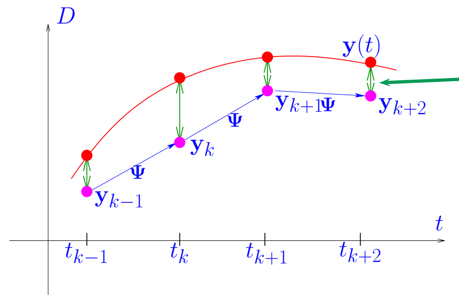
$\diamond$

Convergence analysis for explicit Euler method (11.2.1) for autonomous IVP (11.1.5) with sufficiently smooth and (*globally*) Lipschitz continuous $\mathbf{f}$, that is,

$$\exists L>0:\quad \|\mathbf{f}(t,\mathbf{y})-\mathbf{f}(t,\mathbf{z})\|\leq L\,\|\mathbf{y}-\mathbf{z}\|\quad \forall \mathbf{y},\mathbf{z}\in D\,. \tag{11.3.1}$$

Recall: recursion for explicit Euler method

$$\mathbf{y}_{k+1}=\mathbf{y}_k+h_k\mathbf{f}(\mathbf{y}_k)\,,\quad k=1,\dots,N-1\,. \tag{11.2.1}$$



Error sequence: $\mathbf{e}_k:=\mathbf{y}_k-\mathbf{y}(t_k)$.

◁ — $\hat{=}$ $t\mapsto\mathbf{y}(t)$
— $\hat{=}$ Euler polygon,
● $\hat{=}$ $\mathbf{y}(t_k)$,
● $\hat{=}$ $\mathbf{y}_k$,
$\longrightarrow$ $\hat{=}$ discrete evolution $\mathbf{\Psi}^{t_{k+1}-t_k}$

① Abstract splitting of error:

Here and in what follows we rely on the abstract concepts of the evolution operator $\mathbf{\Phi}$ associated with the ODE $\dot{\mathbf{y}}=\mathbf{f}(\mathbf{y})$ (→ Def. 11.1.6) and discrete evolution operator $\mathbf{\Psi}$ defining the explicit Euler single step method, see Def. 11.2.1:

$$(11.2.1)\quad\Rightarrow\quad \mathbf{\Psi}^h\mathbf{y}=\mathbf{y}+h\mathbf{f}(\mathbf{y})\,. \tag{11.3.2}$$

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

$$\begin{aligned}\mathbf{e}_{k+1}&=\mathbf{\Psi}^{h_k}\mathbf{y}_k-\mathbf{\Phi}^{h_k}\mathbf{y}(t_k)\\&=\underbrace{\mathbf{\Psi}^{h_k}\mathbf{y}_k-\mathbf{\Psi}^{h_k}\mathbf{y}(t_k)}_{\text{propagated error}}\\&\quad+\underbrace{\mathbf{\Psi}^{h_k}\mathbf{y}(t_k)-\mathbf{\Phi}^{h_k}\mathbf{y}(t_k)}_{\text{one-step error}}\,.\end{aligned} \tag{11.3.3}$$

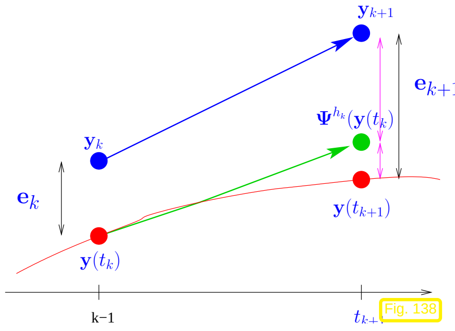


Fig. 138

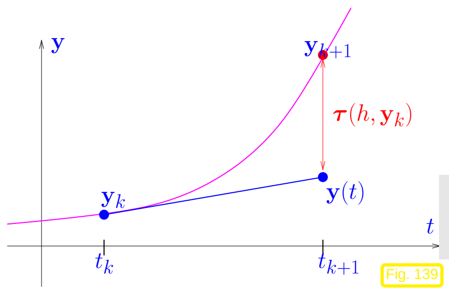


Fig. 139

one-step error:

$$\boldsymbol{\tau}(h,\mathbf{y}):=\mathbf{\Psi}^h\mathbf{y}-\mathbf{\Phi}^h\mathbf{y}\,. \tag{11.3.4}$$

◁ geometric visualisation of one-step error for explicit Euler method (11.2.1), *cf.* Fig. 133.

✎ notation: $t\mapsto\mathbf{y}(t)$ $\hat{=}$ (unique) solution of IVP, *cf.* Thm. 11.1.4.

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. 141).

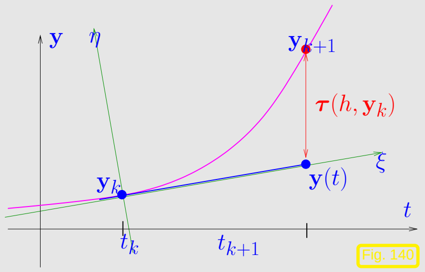


Fig. 140

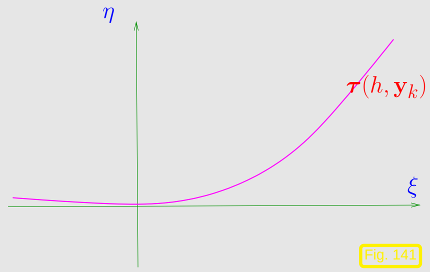


Fig. 141

Analytic considerations: recall Taylor's formula for function $\mathbf{y}\in C^{K+1}$

$$\mathbf{y}(t+h)-\mathbf{y}(t)=\sum_{j=0}^{K}\mathbf{y}^{(j)}(t)\frac{h^j}{j!}+\underbrace{\int_t^{t+h}f^{(K+1)}(\tau)\frac{(t+h-\tau)^K}{K!}\,\mathrm{d}\tau}_{=\frac{f^{(K+1)}(\xi)}{K!}h^{K+1}}\,, \tag{11.3.5}$$

for some $\xi\in[t,t+h]$

⇒ if $\mathbf{y}\in C^2([0,T])$, then

▶ $$\begin{aligned}\mathbf{y}(t_{k+1})-\mathbf{y}(t_k) &= \dot{\mathbf{y}}(t_k)h_k+\tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2 \quad \text{for some } t_k\le\xi_k\le t_{k+1}\\ &= \mathbf{f}(\mathbf{y}(t_k))h_k+\tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2\ ,\end{aligned}$$

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 (11.3.4)

$$\begin{aligned}\boldsymbol{\tau}(h_k,\mathbf{y}(t_k)) &= \boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k)-\mathbf{y}(t_k+h_k)\\ &\overset{(11.3.2)}{=} \mathbf{y}(t_k)+h_k\mathbf{f}(\mathbf{y}(t_k))-\mathbf{y}(t_k)-\mathbf{f}(\mathbf{y}(t_k))h_k+\tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2\\ &= \tfrac{1}{2}\ddot{\mathbf{y}}(\xi_k)h_k^2\ .\end{aligned} \tag{11.3.6}$$

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 (11.3.3)

$$\begin{aligned}\left\|\boldsymbol{\Psi}^{h_k}\mathbf{y}_k-\boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k)\right\| &= \left\|\mathbf{y}_k+h_k\mathbf{f}(\mathbf{y}_k)-\mathbf{y}(t_k)-h_k\mathbf{f}(\mathbf{y}(t_k))\right\|\\ &\overset{(11.3.1)}{\le} (1+Lh_k)\left\|\mathbf{y}_k-\mathbf{y}(t_k)\right\|\ .\end{aligned} \tag{11.3.7}$$

③ *Recursion* for error norms $\epsilon_k:=\|\mathbf{e}_k\|$ by △-inequality:

$$\epsilon_{k+1}\le(1+h_kL)\epsilon_k+\rho_k\ ,\quad \rho_k:=\tfrac{1}{2}h_k^2\max_{t_k\le\tau\le t_{k+1}}\|\ddot{\mathbf{y}}(\tau)\|\ . \tag{11.3.8}$$

Taking into account $\epsilon_0=0$ this leads to

$$\epsilon_k\le\sum_{l=1}^{k}\prod_{j=1}^{l-1}(1+Lh_j)\,\rho_l\ ,\quad k=1,\dots,N\ . \tag{11.3.9}$$

Use the elementary estimate $(1+Lh_j)\le\exp(Lh_j)$ (by convexity of exponential function):

(11.3.9) $\Rightarrow$ $$\epsilon_k\le\sum_{l=1}^{k}\prod_{j=1}^{l-1}\exp(Lh_j)\cdot\rho_l=\sum_{l=1}^{k}\exp(L\sum_{j=1}^{l-1}h_j)\rho_l\ .$$

Note: $\sum_{j=1}^{l-1}h_j\le T$ for final time T

▶ $$\begin{aligned}\epsilon_k &\le \exp(LT)\sum_{l=1}^{k}\rho_l\le\exp(LT)\max_k\frac{\rho_k}{h_k}\sum_{l=1}^{k}h_l\\ &\le T\exp(LT)\max_{l=1,\dots,k}h_l\cdot\max_{t_0\le\tau\le t_k}\|\ddot{\mathbf{y}}(\tau)\|\ .\end{aligned}$$

▶ $$\|\mathbf{y}_k-\mathbf{y}(t_k)\|\le T\exp(LT)\max_{l=1,\dots,k}h_l\cdot\max_{t_0\le\tau\le t_k}\|\ddot{\mathbf{y}}(\tau)\|\ .$$

Total error arises from accumulation of one-step errors!

- error bound $=O(h)$, $h:=\max_l h_l$ (➤ 1st-order algebraic convergence)
- Error bound grows *exponentially* with the length T of the integration interval.

Most commonly used single step methods display algebraic convergence of integer order with respect to the meshwidth $h:=\max_k h_k$. This offers a criterion for gauging their quality.

The sequence $(\mathbf{y}_k)_k$ generated by a

single step method ($\rightarrow$ Def. 11.2.1) of order (of consistency) $p\in\mathbb{N}$

for $\dot{\mathbf{y}}=\mathbf{f}(t,\mathbf{y})$ on a mesh $\mathcal{M}:=\{t_0<t_1<\dots<t_N=T\}$ satisfies

$$\max_k\|\mathbf{y}_k-\mathbf{y}(t_k)\|\le Ch^p \quad\text{for}\quad h:=\max_{k=1,\dots,N}|t_k-t_{k-1}|\to 0\ ,$$

with $C>0$ independent of $\mathcal{M}$, provided that $\mathbf{f}$ is *sufficiently smooth*.

11.4 Runge-Kutta methods

So far we only know first order methods, the explicit and implicit Euler method (11.2.1) and (11.2.4), respectively.

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:

IVP: $$\begin{aligned}\dot{\mathbf{y}}(t)&=\mathbf{f}(t,\mathbf{y}(t))\ ,\\ \mathbf{y}(t_0)&=\mathbf{y}_0\end{aligned}\quad\Rightarrow\quad \mathbf{y}(t_1)=\mathbf{y}_0+\int_{t_0}^{t_1}\mathbf{f}(\tau,\mathbf{y}(\tau))\,d\tau$$

Idea: approximate integral by means of s-point quadrature formula ($\rightarrow$ Sect. 10.1, defined on reference interval $[0,1]$) with nodes $c_1,\dots,c_s$, weights $b_1,\dots,b_s$.

$$\mathbf{y}(t_1)\approx\mathbf{y}_1=\mathbf{y}_0+h\sum_{i=1}^{s}b_i\mathbf{f}(t_0+c_ih,\boxed{\mathbf{y}(t_0+c_ih)})\ ,\quad h:=t_1-t_0\ . \tag{11.4.1}$$

Obtain these values by bootstrapping

bootstrapping = use the same idea in a simpler version to get $\mathbf{y}(t_0+c_ih)$, noting that these values can be replaced by other approximations obtained by methods already constructed (this approach will be elucidated in the next example).

What error can we afford in the approximation of $\mathbf{y}(t_0+c_ih)$ (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

$\mathbf{y}(t_0+c_ih)$ is approximated up to an error $O(h^p)$,

because in (11.4.1) a factor of size h multiplies $\mathbf{f}(t_0+c_i,\mathbf{y}(t_0+c_ih))$.

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 11.4.1 (Construction of simple Runge-Kutta methods).

Quadrature formula = trapezoidal rule (11.4.2):

$$Q(f)=\tfrac{1}{2}(f(0)+f(1)) \quad\leftrightarrow\quad s=2:\quad c_1=0, c_2=1\ ,\quad b_1=b_2=\frac{1}{2}\ , \tag{11.4.2}$$

and $\mathbf{y}(T)$ approximated by explicit Euler step (11.2.1)

$$\mathbf{k}_1=\mathbf{f}(t_0,\mathbf{y}_0)\ ,\quad \mathbf{k}_2=\mathbf{f}(t_0+h,\mathbf{y}_0+h\mathbf{k}_1)\ ,\quad \mathbf{y}_1=\mathbf{y}_0+\tfrac{h}{2}(\mathbf{k}_1+\mathbf{k}_2)\ . \tag{11.4.3}$$

(11.4.3) = explicit trapezoidal rule (for numerical integration of ODEs)

Quadrature formula $\to$ simplest Gauss quadrature formula = midpoint rule ($\to$ Ex. 10.2.1) & $\mathbf{y}(\frac{1}{2}(t_1-$ $t_0))$ approximated by explicit Euler step (11.2.1)

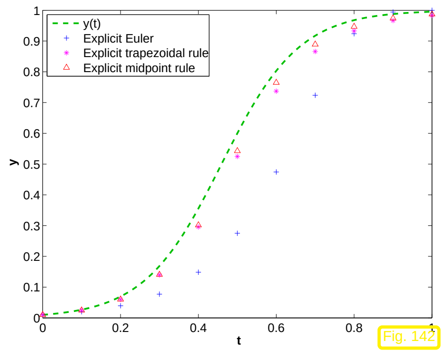
$$\mathbf{k}_1=\mathbf{f}(t_0,\mathbf{y}_0)\ ,\quad \mathbf{k}_2=\mathbf{f}(t_0+\tfrac{h}{2},\mathbf{y}_0+\tfrac{h}{2}\mathbf{k}_1)\ ,\quad \mathbf{y}_1=\mathbf{y}_0+h\mathbf{k}_2\ . \tag{11.4.4}$$

(11.4.4) = explicit midpoint rule (for numerical integration of ODEs)

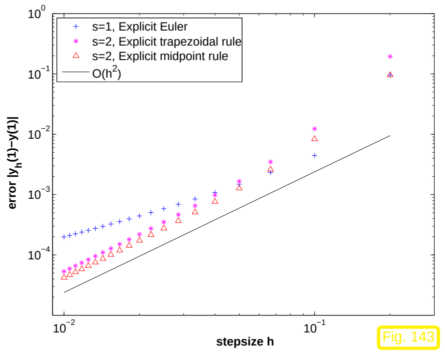
◇

Example 11.4.2 (Convergence of simple Runge-Kutta methods).

- IVP: $\dot{y}=10y(1-y)$ (logistic ODE (11.1.1)), $y(0)=0.01$, $T=1$,
- Explicit single step methods, uniform timestep h.



$y_h(j/10)$, $j=1,\ldots,10$ for explicit RK-methods



Errors at final time $y_h(1)-y(1)$

Observation: obvious algebraic convergence with integer rates/orders

explicit trapezoidal rule (11.4.3) order 2
explicit midpoint rule (11.4.4) order 2

◇

The formulas that we have obtained follow a general pattern:

Definition 11.4.1 (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 (11.1.5) is defined by

$$\mathbf{k}_i:=\mathbf{f}(t_0+c_ih,\mathbf{y}_0+h\sum_{j=1}^{i-1}a_{ij}\mathbf{k}_j)\ ,\quad i=1,\ldots,s\quad ,\quad \mathbf{y}_1:=\mathbf{y}_0+h\sum_{i=1}^{s}b_i\mathbf{k}_i\ .$$

The $\mathbf{k}_i\in\mathbb{R}^d$ are called increments.

Recall Rem. 11.2.4 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. 11.4.1 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

Butcher scheme ▷

(Note: $\mathfrak{A}$ is strictly lower triangular $s \times s$-matrix)

$$\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} := \begin{array}{c|cccc} c_1 & 0 & & \cdots & 0 \\ c_2 & a_{21} & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ c_s & a_{s1} & & \cdots & a_{s,s-1} \quad 0 \\ \hline & b_1 & & \cdots & b_s \end{array} . \tag{11.4.5}$$

Note that in Def. 11.4.1 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 $\displaystyle\sum_{i=1}^{s} b_i = 1$

Example 11.4.3 (Butcher scheme for some explicit RK-SSM).

• Explicit Euler method (11.2.1): $\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$ ➢ order = 1

• explicit trapezoidal rule (11.4.3): $\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$ ➢ order = 2

• explicit midpoint rule (11.4.4): $\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$ ➢ order = 2

• Classical 4th-order RK-SSM: $\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array}$ ➢ order = 4

• Kutta's 3/8-rule: $\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{2}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 1 & 1 & -1 & 1 & 0 \\ \hline & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{array}$ ➢ order = 4

◇

Remark 11.4.4 ("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 11.4.5 (Explicit ODE integrator in MATLAB).

Syntax:

$$\boxed{\texttt{[t,y] = ode45(odefun,tspan,y0);}}$$

`odefun`	:	Handle to a function of type `@(t,y)` $\leftrightarrow$ r.h.s. $\mathbf{f}(t,\mathbf{y})$
`tspan`	:	vector $(t_0, T)^T$, initial and final time for numerical integration
`y0`	:	(vector) passing initial state $\mathbf{y}_0 \in \mathbb{R}^d$

Return values:

`t`	:	temporal mesh $\{t_0 < t_1 < t_2 < \cdots < t_{N-1} = t_N = T\}$
`y`	:	sequence $(\mathbf{y}_k)_{k=0}^{N}$ (column vectors)

Code 11.4.6: parts of MATLAB integrator `ode45`

```matlab
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.

ynew = y + f∗hB(:,6);
%: (stepsize control, see Sect. 11.5 dropped
```

△

Example 11.4.7 (Numerical integration of logistic ODE in MATLAB).

MATLAB-integrator: `ode45()`:

MATLAB-CODE: usage of ode45

```
fn = @(t,y) 5*y*(1-y);
[t,y] = ode45(fn,[0 1.5],y0);
plot(t,y,'r-');
```

- Handle passing r.h.s.
- initial and final time
- initial state $\mathbf{y}_0$

◇

11.5 Stepsize control

Example 11.5.1 (Oregonator reaction).

Special case of oscillating Zhabotinski-Belousov reaction [19]:

$$\begin{aligned} \mathrm{BrO}_3^- + \mathrm{Br}^- &\mapsto \mathrm{HBrO}_2 \\ \mathrm{HBrO}_2 + \mathrm{Br}^- &\mapsto \mathrm{Org} \\ \mathrm{BrO}_3^- + \mathrm{HBrO}_2 &\mapsto 2\,\mathrm{HBrO}_2 + \mathrm{Ce(IV)} \\ 2\,\mathrm{HBrO}_2 &\mapsto \mathrm{Org} \\ \mathrm{Ce(IV)} &\mapsto \mathrm{Br}^- \end{aligned} \tag{11.5.1}$$

$$\begin{aligned} y_1 &:= c(\mathrm{BrO}_3^-): & \dot{y}_1 &= -k_1y_1y_2 - k_3y_1y_3\,, \\ y_2 &:= c(\mathrm{Br}^-): & \dot{y}_2 &= -k_1y_1y_2 - k_2y_2y_3 + k_5y_5\,, \\ y_3 &:= c(\mathrm{HBrO}_2): & \dot{y}_3 &= k_1y_1y_2 - k_2y_2y_3 + k_3y_1y_3 - 2k_4y_3^2\,, \\ y_4 &:= c(\mathrm{Org}): & \dot{y}_4 &= k_2y_2y_3 + k_4y_3^2\,, \\ y_5 &:= c(\mathrm{Ce(IV)}): & \dot{y}_5 &= k_3y_1y_3 - k_5y_5\,, \end{aligned} \tag{11.5.2}$$

with (non-dimensionalized) reaction constants:

$$k_1 = 1.34\,,\quad k_2 = 1.6\cdot 10^9\,,\quad k_3 = 8.0\cdot 10^3\,,\quad k_4 = 4.0\cdot 10^7\,,\quad k_5 = 1.0\,.$$

▶ periodic chemical reaction ➨ Video 1, Video 2

MATLAB simulation with initial state $y_1(0) = 0.06$, $y_2(0) = 0.33\cdot 10^{-6}$, $y_3(0) = 0.501\cdot 10^{-10}$, $y_4(0) = 0.03$, $y_5(0) = 0.24\cdot 10^{-7}$:

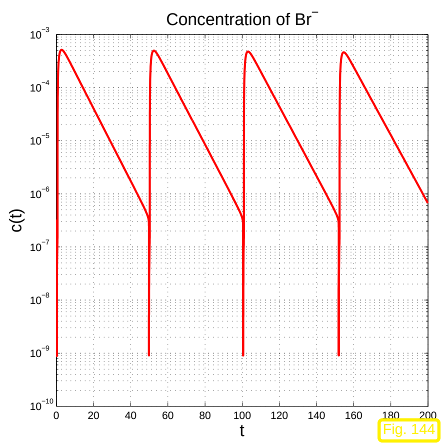


Fig. 144

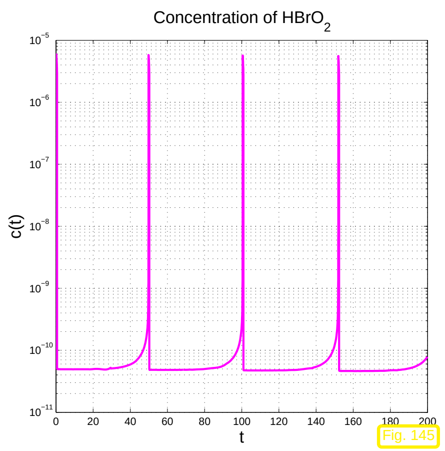


Fig. 145

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 11.5.2 (Blow-up).

Scalar autonomous IVP:

$$\dot{y} = y^2 \,, \quad y(0) = y_0 > 0 \,.$$

$$\blacktriangleright \quad y(t) = \frac{y_0}{1 - y_0 t} \,, \quad t < 1/y_0 \,.$$

Solution exists only for finite time and then suffers a Blow-up, that is, $\lim_{t \to 1/y_0} y(t) = \infty$: $J(y_0) =]-\infty, 1/y_0]$!

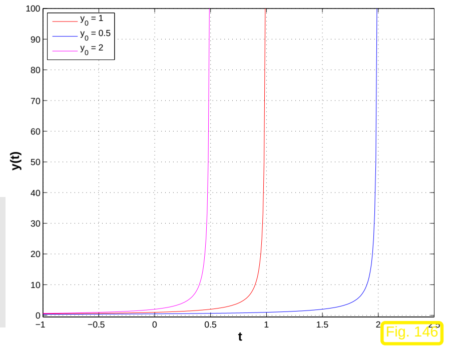


Fig. 146

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 (11.2.1) applied to the above IVP?

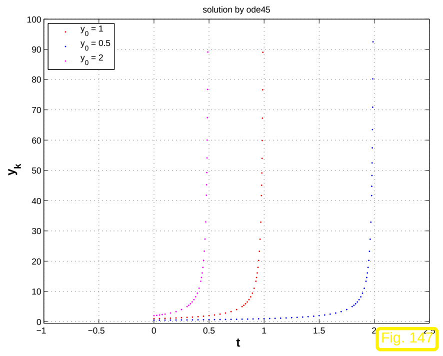


Fig. 147

```
fun = @(t,y) y.^2;
[t1,y1] = ode45(fun,[0 2],1);
[t2,y2] = ode45(fun,[0 2],0.5);
[t3,y3] = ode45(fun,[0 2],2);
```

MATLAB warning messages:

```
  Warning: Failure at t=9.999694e-01.  Unable to meet integration
  tolerances without reducing the step size below the smallest
  value allowed (1.776357e-15) at time t.
> In ode45 at 371
  In simpleblowup at 22
Warning: Failure at t=1.999970e+00.  Unable to meet integration
  tolerances without reducing the step size below the smallest
  value allowed (3.552714e-15) at time t.
> In ode45 at 371
  In simpleblowup at 23
Warning: Failure at t=4.999660e-01.  Unable to meet integration
```

```
  tolerances without reducing the step size below the smallest
  value allowed (8.881784e-16) at time t.
> In ode45 at 371
  In simpleblowup at 24
```

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 **&** $\max_{k=1,\dots,N} \|\mathbf{y}(t_k) - \mathbf{y}_k\| < \mathtt{TOL}$ or $\|\mathbf{y}(T) - \mathbf{y}_N\| < \mathtt{TOL}$, $\mathtt{TOL}$ = 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 $\mathbf{y}_k, h_{k-1}$)

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. 10.6:

Idea: *Estimation of one-step error*, *cf.* Sect. 10.6

Compare two discrete evolutions $\mathbf{\Psi}^h$, $\widetilde{\mathbf{\Psi}}^h$ of different order for *current timestep* h:

If Order$(\widetilde{\mathbf{\Psi}})$ > Order$(\mathbf{\Psi})$

$$\Rightarrow \quad \underbrace{\mathbf{\Phi}^h\mathbf{y}(t_k) - \mathbf{\Psi}^h\mathbf{y}(t_k)}_{\text{one-step error}} \approx \mathtt{EST}_k := \widetilde{\mathbf{\Psi}}^h\mathbf{y}(t_k) - \mathbf{\Psi}^h\mathbf{y}(t_k)\;. \tag{11.5.3}$$

Heuristics for concrete h

absolute tolerance

► Compare $\mathtt{EST}_k \leftrightarrow \mathtt{ATOL}$, $\mathtt{EST}_k \leftrightarrow \mathtt{RTOL}\,\|\mathbf{y}_k\|$ ➢ Reject/accept current step (11.5.4)

relative tolerance

► 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 $(*)$

$\mathtt{EST}_k > \max\{\mathtt{ATOL}, \|\mathbf{y}_k\|\,\mathtt{RTOL}\}$: *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 11.5.3: simple local stepsize control for single step methods

```
1  function [t,y] = odeintadapt(Psilow,Psihigh,T,y0,h0,reltol,abstol,hmin)
2  t = 0; y = y0; h = h0;                    %
```

```
3  while ((t(end) < T)  (h > hmin)) %
4    yh = Psihigh(h,y0);  %
5    yH = Psilow(h,y0);   %
6    est = norm(yH-yh);   %
7
8    if (est < max(reltol*norm(y0),abstol))                %
9      y0 = yh; y = [y,y0]; t = [t,t(end) + min(T-t(end),h)]; %
10     h = 1.1*h;                                             %
11   else, h = h/2; end                                       %
12 end
```

Comments on Code 11.5.2:

- 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 (11.5.4),
  - `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 (**??**).
- line 8: make comparison (11.5.4) 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 (11.5.3) it seems that EST_k measures the one-step error for the low-order method $\mathbf{\Psi}$ and that we should use $\mathbf{y}_{k+1} = \mathbf{\Psi}^{h_k}\mathbf{y}_k$, if the timestep is accepted.

However, it would be foolish not to use the better value $\mathbf{y}_{k+1} = \widetilde{\boldsymbol{\Psi}}^{h_k}\boldsymbol{y}_k$, since it is available for free. This is what is done in every implementation of adaptive methods, also in Code 11.5.2, and this choice can be justified by control theoretic arguments [12, Sect. 5.2].

Example 11.5.4 (Simple adaptive stepsize control).

- IVP for ODE $\dot{y} = \cos(\alpha y)^2$, $\alpha > 0$, solution $y(t) = \arctan(\alpha(t-c))/\alpha$ for $y(0) \in]-\pi/2, \pi/2[$
- Simple adaptive timestepping based on explicit Euler (11.2.1) and explicit trapezoidal rule (11.4.3)

Code 11.5.5: MATLAB function for Ex. 11.5.4

```
function odeintadaptdriver(T,a,reltol,abstol)
% Simple adaptive timestepping strategy of Code 11.5.2
% based on explicit Euler (11.2.1) and explicit trapezoidal
% rule (11.4.3)

% 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 $\dot{y} = \cos(ay)$ and its general solution
f = @(y) (cos(a*y).^2); sol = @(t) (atan(a*(t-1))/a);
% Initial state $\mathbf{y}_0$
y0 = sol(0);

% Discrete evolution operators, see Def. 11.2.1
Psilow = @(h,y) (y + h*f(y)); % Explicit Euler (11.2.1)
% Explicit trapzoidal rule (11.4.3)
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;
% Main adaptive timestepping loop, see Code 11.5.2
[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';
```

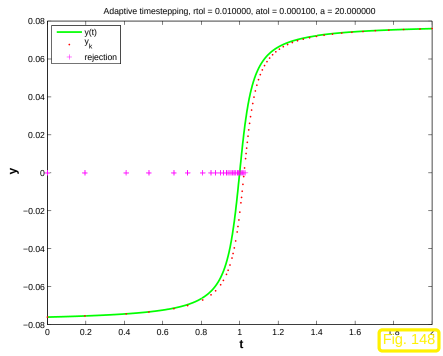


Fig. 148

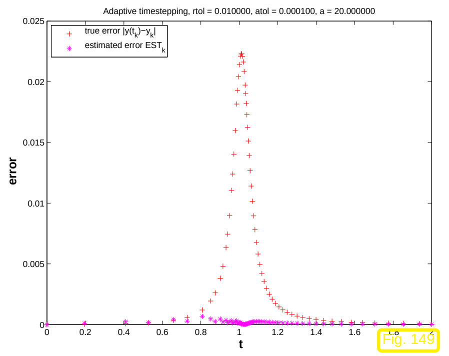


Fig. 149

Statistics: `66 timesteps, 131 rejected timesteps`

Observations:

- ☞ Adaptive timestepping well resolves local features of solution $y(t)$ at $t = 1$
- ☞ Estimated error (an estimate for the one-step error) and true error are **not** related!

◇

Example 11.5.6 (Gain through adaptivity). → Ex. 11.5.4

Simple adaptive timestepping from previous experiment Ex. 11.5.4.

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 11.5.7: MATLAB function for Ex. 11.5.6

```
function adaptgain(T,a,reltol,abstol)
% Experimental study of gasin through simple adaptive timestepping
% strategy of Code 11.5.2 based on explicit Euler
% (11.2.1) and explicit trapezoidal
% rule (11.4.3)

% 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 $\dot{y} = \cos(ay)$ and its general solution
f = @(y) (cos(a*y).^2); sol = @(t) (atan(a*(t))/a);
% Initial state $\mathbf{y}_0$
y0 = sol(0);

% Discrete evolution operators, see Def. 11.2.1
Psilow = @(h,y) (y + h*f(y)); % Explicit Euler (11.2.1)
% Explicit trapzoidal rule (11.4.3)
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 (11.4.3)
erruf = [];
for N=10:10:200
  h = T/N; t = 0; y = y0; err = 0;
  for k=1:N
    y = Psihigh(h,y); t = t+h;
    err = max(err,abs(sol(t) - y));
  end
  erruf = [erruf;N, err];
```

```
end

% Run adaptive timestepping with various tolerances, which is the only way
% to make it use a different total number of timesteps.
% Plot the solution sequences for different values of the relative tolerance.
figure('name','adaptive␣timestepping');
axis([0 2 0 0.05]); hold on; col = colormap;
errad = []; l = 1;
for rtol=reltol*2.^(2:-1:-4)
% Crude choice of initial timestep and $h_{\min}$
  h0 = T/10; hmin = h0/10000;
% Main adaptive timestepping loop, see Code 11.5.2
  [t,y,rej,ee] =
    odeintadapt_ext(Psilow,Psihigh,T,y0,h0,rtol,0.01*rtol,hmin);
  errad = [errad; length(t)-1, max(abs(sol(t) - y)), rtol,
    length(rej)];
  fprintf('rtol␣=␣%d:␣%d␣timesteps,␣%d␣rejected␣timesteps\n',
    rtol,length(t)-1,length(rej));
  plot(t,y,'.','color',col(10*(l-1)+1,:));
  leg{l} = sprintf('rtol␣=␣%f',rtol); l = l+1;
end
xlabel('{\bf␣t}','fontsize',14);
ylabel('{\bf␣y}','fontsize',14);
```

```
legend(leg,'location','southeast');
title(sprintf('Solving␣d_t␣y␣=␣a␣cos(y)^2␣with␣a␣=␣%f␣by␣simple␣
  adaptive␣timestepping',a));
print -depsc2 '../PICTURES/adaptgainsol.eps';

% Plotting the errors vs. the number of timesteps
figure('name','gain␣by␣adaptivity');
loglog(erruf(:,1), erruf(:,2),'r+',errad(:,1), errad(:,2),'m*');
xlabel('{\bf␣no.␣N␣of␣timesteps}','fontsize',14);
ylabel('{\bf␣max_k|y(t_k)-y_k|}','fontsize',14);
title(sprintf('Error␣vs.␣no.␣of␣timesteps␣for␣d_t␣y␣=␣a␣cos(y)^2␣with␣a
  ␣=␣%f',a));
legend('uniform␣timestep','adaptive␣timestep','location','northeast');
print -depsc2 '../PICTURES/adaptgain.eps';
```

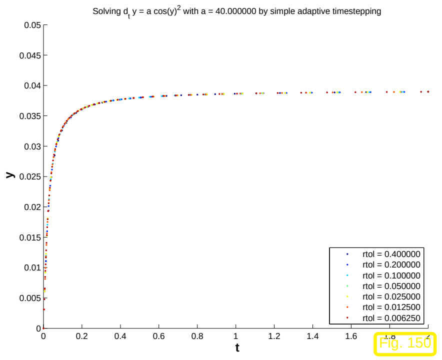


Fig. 150

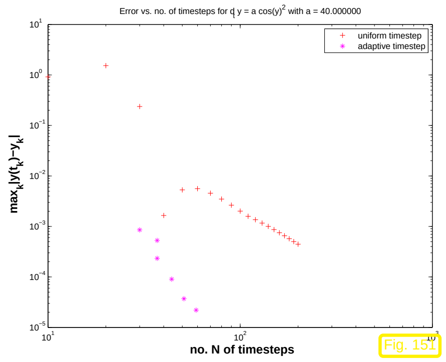


Fig. 151

Solutions $(\mathbf{y}_k)_k$ for different values of `rtol`

Error vs. computational effort

Observations:

☞ Adaptive timestepping achieves much better accuracy for a fixed computational effort.

◇

Example 11.5.8 ("Failure" of adaptive timestepping). $\longrightarrow$ Ex. 11.5.6

Same ODE and simple adaptive timestepping as in previous experiment Ex. 11.5.6. Same evaluations.

Now: initial state $y(0) = -0.0386$ as in Ex. 11.5.4

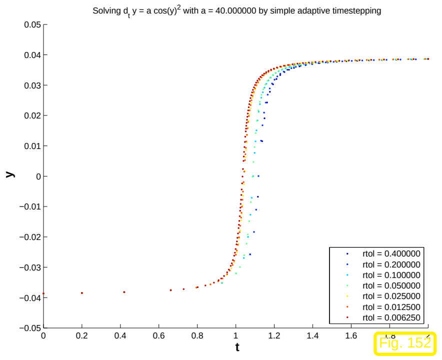


Fig. 152

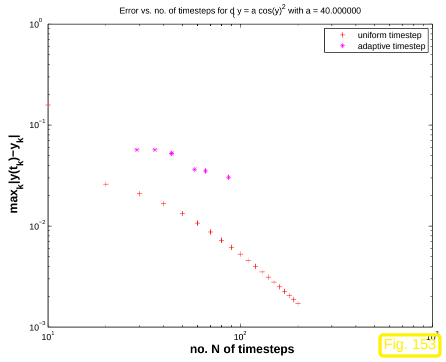


Fig. 153

Solutions $(\mathbf{y}_k)_k$ for different values of `rtol`

Error vs. computational effort

Observations:

☞ Adaptive timestepping leads to larger errors at the same computational cost as uniform timestepping.

Explanation: the position of the steep step of the solution has a sensitive dependence on an initial value $y(0) \approx -\pi/2$. 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 11.5.9 (Refined local stepsize control).

The above algorithm (Code 11.5.2) 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 11.5.2 (, passed as arguments `Psilow`, `Psihigh` there).

Asymptotic expressions for one-step error for $h \to 0$:

$$\begin{aligned}\boldsymbol{\Psi}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &= ch^{p+1} + O(h_k^{p+2}) \,,\\ \widetilde{\boldsymbol{\Psi}}^{h_k}\mathbf{y}(t_k) - \boldsymbol{\Phi}^{h_k}\mathbf{y}(t_k) &= O(h^{p+2}) \,,\end{aligned} \tag{11.5.5}$$

with some $c > 0$.

Why h^{p+1}? Remember estimate (11.3.6) 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.

$$\begin{aligned}\Psi^{h_k}\mathbf{y}(t_k) - \Phi^{h_k}\mathbf{y}(t_k) &\doteq c h_k^{p+1} + O(h_k^{p+2}) ,\\ \widetilde{\Psi}^{h_k}\mathbf{y}(t_k) - \Phi^{h_k}\mathbf{y}(t_k) &\doteq O(h_k^{p+2}) .\end{aligned} \quad \Rightarrow \quad \boxed{\mathtt{EST}_k \doteq c h_k^{p+1}} . \tag{11.5.6}$$

✎ notation: $\doteq$ equality up to higher order terms in h_k

$$\mathtt{EST}_k \doteq c h_k^{p+1} \quad \Rightarrow \quad c \doteq \frac{\mathtt{EST}_k}{h_k^{p+1}} . \tag{11.5.7}$$

Available in algorithm, see (11.5.3)

For the sake of *accuracy* (stipulates "$\mathtt{EST}_k < \mathtt{TOL}$") & *efficiency* (favors "$>$") we aim for

$$\mathtt{EST}_k \overset{!}{=} \mathtt{TOL} := \max\{\mathtt{ATOL}, \|\mathbf{y}_k\|\,\mathtt{RTOL}\} . \tag{11.5.8}$$

What timestep h_* can actually achieve (11.5.8), if we "believe" in (11.5.6) (and, therefore, in (11.5.7))?

$$\text{(11.5.7) \& (11.5.8)} \quad \Rightarrow \quad \mathtt{TOL} = \frac{\mathtt{EST}_k}{h_k^{p+1}} h_*^{p+1} .$$

► "'Optimal timestep": (stepsize prediction)

$$\boxed{h_* = h \sqrt[p+1]{\frac{\mathtt{TOL}}{\mathtt{EST}_k}}} . \tag{11.5.9}$$

adjusted stepsize (**A**)

suggested stepsize (**B**)

(**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 11.5.10: refined local stepsize control for single step methods

```
1  function [t,y] =
     odeintssctrl(Psilow,p,Psihigh,T,y0,h0,reltol,abstol,hmin)
2  t = 0; y = y0; h = h0;                    %
3  while ((t(end) < T) (h > hmin)) %
4    yh = Psihigh(h,y0);   %
5    yH = Psilow(h,y0);    %
6    est = norm(yH-yh);    %
7
8    tol = max(reltol*norm(y(:,end)),abstol);               %
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 11.5.9 (see comments on Code 11.5.2 for more explanations):

• Input arguments as for Code 11.5.2, except for p $\hat{=}$ order of lower order discrete evolution.

• line 9: compute presumably better local stepsize according to (11.5.9),

• line 10: decide whether to repeat the step or advance,

• line 11: extend output arrays if current step has not been rejected.

Remark 11.5.11 (Stepsize control in MATLAB).

Ψ $\hat{=}$ RK-method of order 4 $\widetilde{\Psi}$ $\hat{=}$ RK-method of order 5

`ode45`

Specifying tolerances for MATLAB's integrators:

```
options = odeset('abstol',atol,'reltol',rtol,'stats','on');
[t,y] = ode45(@(t,x) f(t,x),tspan,y0,options);
```

(`f` = function handle, `tspan` $\hat{=}$ $[t_0, T]$, `y0` $\hat{=}$ $\mathbf{y}_0$, `t` $\hat{=}$ t_k, `y` $\hat{=}$ $\mathbf{y}_k$)

△

Example 11.5.12 (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

$$\text{Newton's law:} \quad \underbrace{\ddot{\mathbf{y}}}_{\text{acceleration}} = F(\mathbf{y}) := \underbrace{-\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2}}_{\text{force}} . \tag{11.5.10}$$

Equivalent 1st-order ODE, see Rem. 11.1.7: with velocity $\mathbf{v} := \dot{\mathbf{y}}$

$$\begin{pmatrix}\dot{\mathbf{y}}\\ \dot{\mathbf{v}}\end{pmatrix} = \begin{pmatrix}\mathbf{v}\\ -\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2}\end{pmatrix} . \tag{11.5.11}$$

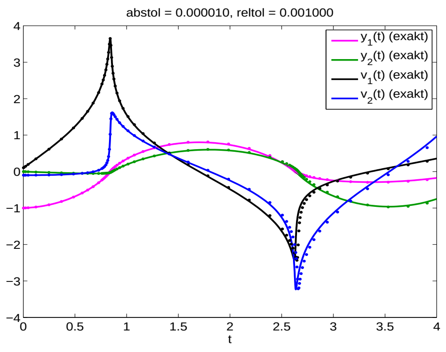
Initial values used in the experiment:

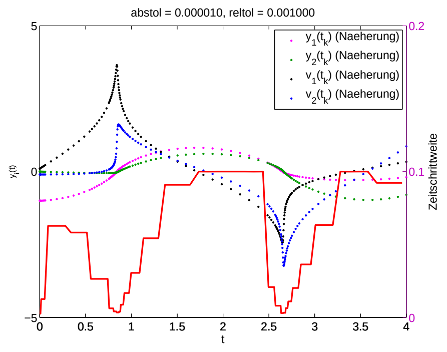
$$\mathbf{y}(0) := \begin{pmatrix}-1\\ 0\end{pmatrix} , \quad \mathbf{v}(0) := \begin{pmatrix}0.1\\ -0.1\end{pmatrix}$$

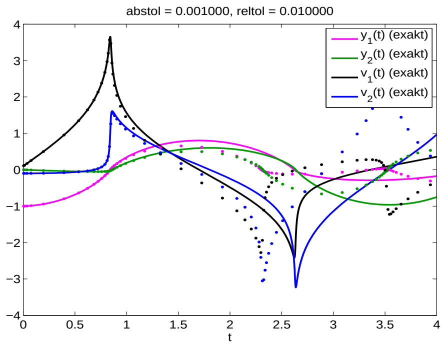
Adaptive integrator: `ode45(@(t,x) f,[0 4],[-1;0;0.1;-0.1,],options)`:

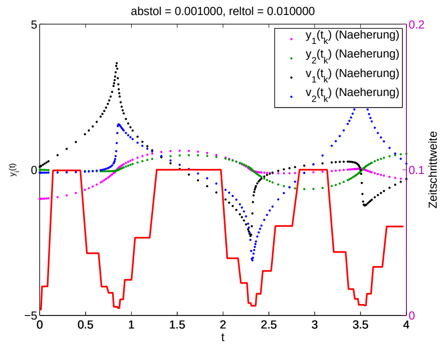
❶ `options = odeset('reltol',0.001,'abstol',1e-5);`

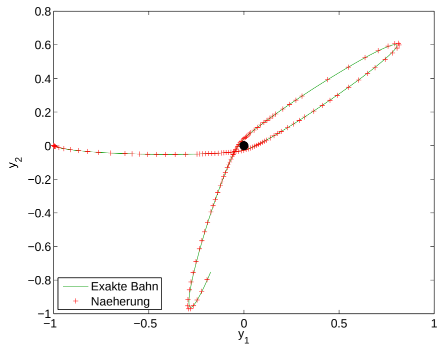
❷ `options = odeset('reltol',0.01,'abstol',1e-3);`



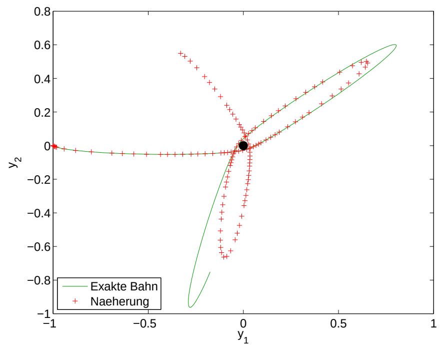








reltol=0.001, abstol=1e-5



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.

◇

An inevitable consequence of time-local error estimation:

Absolute/relative tolerances do *not* allow to predict accuracy of solution!

12 Stiff Integrators

Explicit Runge-Kutta methods with stepsize control (→ Sect. 11.5) 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**?**

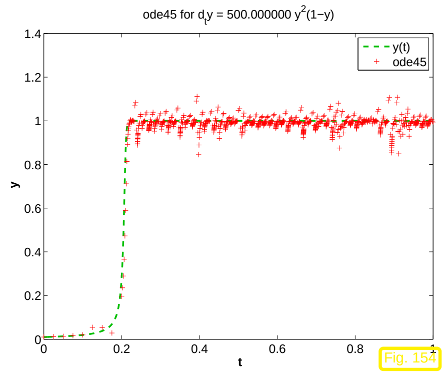
Example 12.0.1 (ode45 for stiff problem).

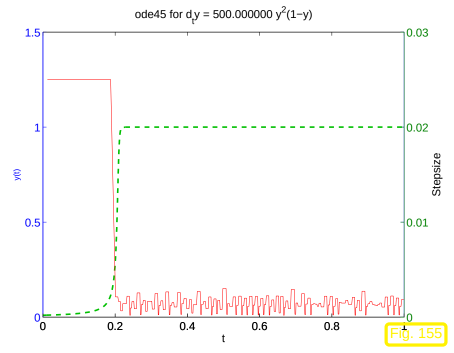
IVP: $\dot{y} = \lambda y^2(1-y)$, $\lambda := 500$, $y(0) = \frac{1}{100}$.

```
fun = @(t,x) 500*x^2*(1-x);
options = odeset('reltol',0.1,'abstol',0.001,'stats','on');
[t,y] = ode45(fun,[0 1],y0,options);
```

The option `stats = 'on'` makes MATLAB print statistics about the run of the integrators.

```
186 successful steps
55 failed attempts
1447 function evaluations
```





Stepsize control of ode45 running amok!

? The solution is virtually constant from $t > 0.2$ and, nevertheless, the integrator uses tiny timesteps until the end of the integration interval.

◇

12.1 Model problem analysis

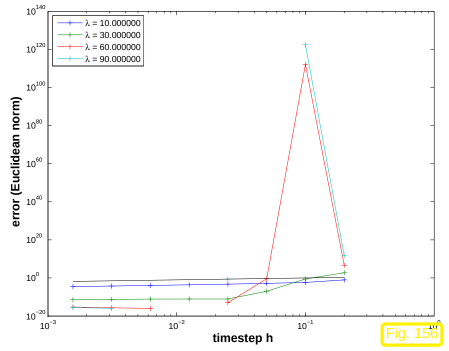
Example 12.1.1 (Blow-up of explicit Euler method).

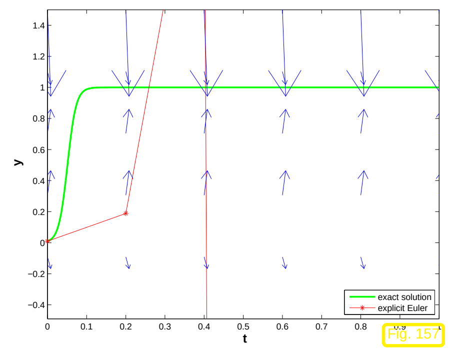
As in part II of Ex. 11.3.1:

- IVP for logistic ODE, see Ex. 11.1.1

$$\dot{y} = f(y) := \lambda y(1-y) \quad , \quad y(0) = 0.01 \; .$$

- Explicit Euler method (11.2.1) with uniform timestep $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.





λ large: blow-up of y_k for large timestep h $\qquad$ $\lambda = 90$: — $\hat{=}$ $y(t)$, — $\hat{=}$ 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. 12.1.1 we study the explicit Euler method (11.2.1) for the

$$\text{linear model problem:}\quad \dot{y} = \lambda y\ ,\quad y(0) = y_0\ ,\quad \text{with}\quad \lambda \ll 0\ , \tag{12.1.1}$$

and *exponentially decaying* exact solution

$$y(t) = y_0 \exp(\lambda t) \to 0 \quad \text{for } t \to \infty\ .$$

Recursion of explicit Euler method for (12.1.1):

$$\text{(11.2.1) for } f(y) = \lambda y\text{:}\qquad y_{k+1} = y_k(1 + \lambda h)\ . \tag{12.1.2}$$

$$\blacktriangleright\quad y_k = y_0(1 + \lambda h)^k \quad\Rightarrow\quad |y_k| \to \begin{cases} 0 & \text{, if } \lambda h > -2 \quad \text{(qualitatively correct)}\ ,\\ \infty & \text{, if } \lambda h < -2 \quad \text{(qualitatively wrong)}\ .\end{cases}$$

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. 12.1.1? Let us examine how the simple stepsize control of Code 11.5.2 fares for model problem (12.1.1):

Example 12.1.2 (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. 11.5.4, see Code 11.5.2

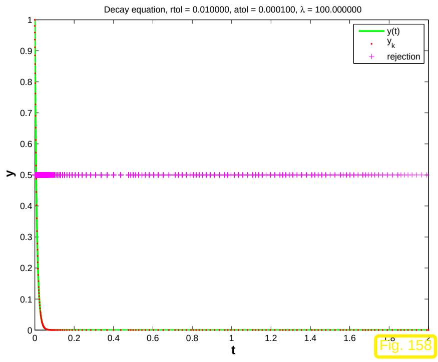


Fig. 158

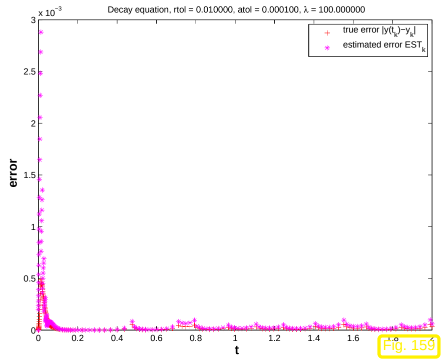


Fig. 159

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 12.1.3 (Explicit trapzoidal rule for decay equation).

Recall recursion for explicit trapezoidal rule:

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0)\ ,\quad \mathbf{k}_2 = \mathbf{f}(t_0 + h, \mathbf{y}_0 + h\mathbf{k}_1)\ ,\quad \mathbf{y}_1 = \mathbf{y}_0 + \tfrac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2)\ . \tag{11.4.3}$$

Apply this to the model problem (12.1.1), that is $\mathbf{f}(y) = f(y) = \lambda y$, $\lambda < 0$:

$$\blacktriangleright\quad k_1 = \lambda y_0\ ,\quad k_2 = \lambda(y_0 + hk_1) \quad\Rightarrow\quad y_1 = \underbrace{(1 + \lambda h + \tfrac{1}{2}(\lambda h)^2)}_{=:S(h\lambda)} y_0\ . \tag{12.1.3}$$

$\blacktriangleright$ sequence generated by explicit trapezoidal rule:

$$y_k = S(h\lambda)^k y_0\ ,\quad k = 0, \dots, N\ . \tag{12.1.4}$$

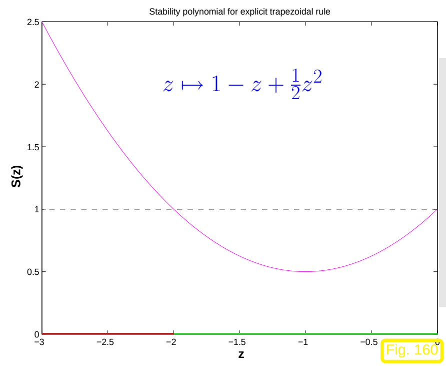


Fig. 160

$$|S(h\lambda)| < 1 \quad\Leftrightarrow\quad -2 < h\lambda < 0\ .$$

Qualitatively correct decay behavior of $(y_k)_k$ only under timestep constraint

$$h \le |2/\lambda|\ . \tag{12.1.5}$$

◇

Mode problem analysis for general explicit Runge-Kutta method (→ Def. 11.4.1): apply Runge-Kutta method $\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$ to (12.1.1)

$$\blacktriangleright\quad \begin{aligned} k_i &= \lambda\Big(y_0 + h\sum_{j=1}^{i-1} a_{ij}k_j\Big)\ ,\\ y_1 &= y_0 + h\sum_{i=1}^{s} b_i k_i \end{aligned} \quad\Rightarrow\quad \begin{pmatrix} \mathbf{I} - z\mathfrak{A} & 0 \\ -z\mathbf{b}^T & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{k} \\ y_1 \end{pmatrix} = y_0 \begin{pmatrix} \mathbf{1} \\ 1 \end{pmatrix}\ , \tag{12.1.6}$$

where $\boldsymbol{k} \in \mathbb{R}^s \mathrel{\hat=}$ denotes the vector $(k_1, \dots, k_s)^T/\lambda$ of increments, and $z := \lambda h$.

$$\blacktriangleright \quad y_1 = S(z) y_0 \quad \text{with} \quad S(z) := 1 + z\mathbf{b}^T (\mathbf{I} - z\mathfrak{A})^{-1} \mathbf{1} = \det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T) \,. \tag{12.1.7}$$

The first formula for $S(z)$ immediately follows from (12.1.6), the second is a consequence of Cramer's rule.

Thus we have proved the following theorem.

Theorem 12.1.1 (Stability function of explicit Runge-Kutta methods)**.**
The discrete evolution Ψ^h_λ *of an explicit* s*-stage Runge-Kutta single step method* (→ *Def. 11.4.1*) *with Butcher scheme* $\begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$ *(see* (11.4.5)*) for the ODE* $\dot{y} = \lambda y$ *is a multiplication operator according to*

$$\Psi^h_\lambda = \underbrace{1 + z\mathbf{b}^T (\mathbf{I} - z\mathfrak{A})^{-1} \mathbf{1}}_{\text{stability function } S(z)} = \det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T) \,, \quad z := \lambda h \,, \quad \mathbf{1} = (1, \dots, 1)^T \in \mathbb{R}^s \,.$$

Thm. 12.1.1 $\Rightarrow$ $S \in \mathcal{P}_s$

Remember from Ex. 12.1.3: for sequence $(|y_k|)_{k=0}^\infty$ produced by explicit Runge-Kutta method applied to IVP (12.1.1) holds $y_k = S(\lambda h)^k y_0$.

$$\blacktriangleright \quad \begin{array}{l} \boxed{(|y_k|)_{k=0}^\infty \text{ non-increasing} \Leftrightarrow |S(\lambda h)| \leq 1} \,, \\ (|y_k|)_{k=0}^\infty \text{ exponentially increasing} \Leftrightarrow |S(\lambda h)| > 1 \,. \end{array} \tag{12.1.8}$$

On the other hand: $\forall S \in \mathcal{P}_s$: $\lim\limits_{|z| \to \infty} |S(z)| = \infty$

▶ 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

Remark 12.1.4 (Stepsize control detects instability).

Always look at the bright side of life:

Ex. 12.0.1, 12.1.2: Stepsize control guarantees acceptable solutions, with a hefty price tag however.
△

12.2 Stiff problems

Objection: The IVP (12.1.1) 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. 12.0.1 and Ex. 12.1.1 is typical for a large class of problems and that the model problem (12.1.1) really represents a "generic case".

Example 12.2.1 (Transient simulation of RLC-circuit).

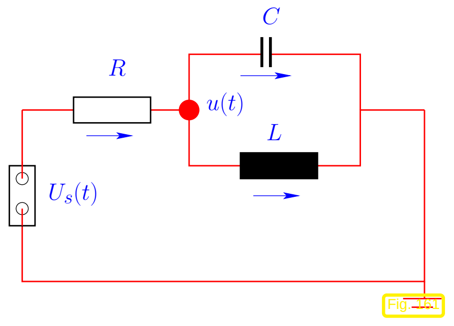
Circuit from Ex. 11.1.6 ▷

$$\ddot{u} + \alpha\dot{u} + \beta u = g(t) \,,$$

$\alpha := (RC)^{-1}$, $\beta = (LC)^{-1}$, $g(t) = \alpha \dot{U}_s$.

Transformation to linear 1st-order ODE, see Rem. 11.1.7, $v := \dot{u}$

$$\underbrace{\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix}}_{=:\dot{\mathbf{y}}} = \underbrace{\begin{pmatrix} 0 & 1 \\ -\beta & -\alpha \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} - \begin{pmatrix} 0 \\ g(t) \end{pmatrix}}_{=:\mathbf{f}(t,\mathbf{y})} \,.$$



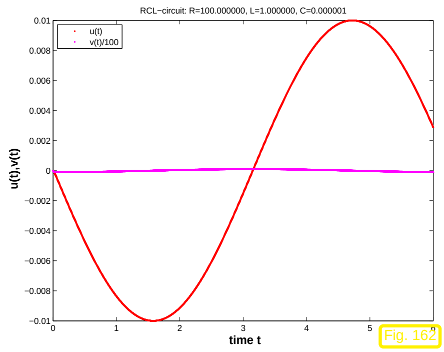


Fig. 162

$R = 100\Omega$, $L = 1\mathrm{H}$, $C = 1\mu\mathrm{F}$, $U_s(t) = 1\mathrm{V}\sin(t)$,
$u(0) = v(0) = 0$ (“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:

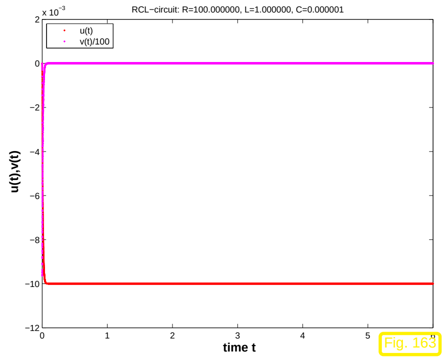


Fig. 163

$R = 100\Omega$, $L = 1\mathrm{H}$, $C = 1\mu\mathrm{F}$, $U_s(t) = 1\mathrm{V}$,
$u(0) = v(0) = 0$ (“switch on”)

```
ode45 statistics:
17901 successful steps
1210 failed attempts
114667 function evaluations
```

Code 12.2.2: simulation of linear RLC circuit using `ode45`

```
function stiffcircuit(R,L,C,Us,tspan,filename)
% Transient simulation of simple linear circuit of Ex. refex:stiffcircuit
% R,L,C: paramters for circuits elements (compatible units required)
% Us: exciting time-dependent voltage $U_s = U_s(t)$, function handle
% zero initial values

% Coefficient for 2nd-order ODE $\ddot{u} + \alpha\dot{u} + \beta = g(t)$
alpha = 1/(R*C); beta = 1/(C*L);
% Conversion to 1st-order ODE   $\mathbf{y} = \mathbf{M}\mathbf{y} + \binom{0}{g(t)}$. Set up right hand side function.
```

```
M = [0 , 1; -beta , -alpha]; rhs = @(t,y) (M*y - [ 0 ; alpha*Us(t)]);
% Set tolerances for MATLAB integrator, see Rem. 11.5.11
options = odeset('reltol',0.1,'abstol',0.001,'stats','on');
y0 = [0;0]; [t,y] = ode45(rhs,tspan,y0,options);

% Plot the solution components
figure('name','Transient circuit simulation');
plot(t,y(:,1),'r.',t,y(:,2)/100,'m.');
xlabel('{\bf time t}','fontsize',14);
ylabel('{\bf u(t),v(t)}','fontsize',14);
title(sprintf('RCL-circuit: R=%f, L=%f, C=%f',R,L,C));
legend('u(t)','v(t)/100','location','northwest');

print('-depsc2',sprintf('../PICTURES/%s.eps',filename));
```

Observation: stepsize control of `ode45` ($\rightarrow$ Sect. 11.5) enforces extremely small timesteps though solution almost constant except at $t = 0$.

◇

Motivated by Ex. 12.2.1 we examine linear homogeneous IVP of the form

$$\dot{\mathbf{y}} = \underbrace{\begin{pmatrix} 0 & 1 \\ -\beta & -\alpha \end{pmatrix}}_{=:\mathbf{M}} \mathbf{y} \quad , \quad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{R}^2 . \qquad (12.2.1)$$

In Ex .12.2.1: $\beta \gg \frac{1}{4}\alpha^2 \gg 1$.

[40, 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):

$$\mathbf{M}\mathbf{V} = \mathbf{M}(\mathbf{v}_1, \mathbf{v}_2) = (\mathbf{v}_1, \mathbf{v}_2)\begin{pmatrix} \lambda_1 & \\ & \lambda_2 \end{pmatrix} . \qquad (12.2.2)$$

▶ $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^2 \setminus \{0\} \;\hat{=}$ eigenvectors of $\mathbf{M}$, $\lambda_1, \lambda_2 \;\hat{=}$ eigenvalues of $\mathbf{M}$, see Def. 5.1.1.

Idea: transform $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$ into *decoupled* scalar linear ODEs!

$$\dot{\mathbf{y}} = \mathbf{M}\mathbf{y} \;\Leftrightarrow\; \mathbf{V}^{-1}\dot{\mathbf{y}} = \mathbf{V}^{-1}\mathbf{M}\mathbf{V}(\mathbf{V}^{-1}\mathbf{y}) \overset{\mathbf{z}(t):=\mathbf{V}^{-1}\mathbf{y}(t)}{\Leftrightarrow} \dot{\mathbf{z}} = \begin{pmatrix} \lambda_1 & \\ & \lambda_2 \end{pmatrix} \mathbf{z} . \qquad (12.2.3)$$

This yields the general solution of the ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$

$$\mathbf{y}(t) = A\mathbf{v}_1 \exp(\lambda_1 t) + B\mathbf{v}_2 \exp(\lambda_2 t) \; , \quad A, B \in \mathbb{R} . \qquad (12.2.4)$$

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 (11.2.1) for ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$

$$\Psi^h \mathbf{y} = \mathbf{y} + h\mathbf{M}\mathbf{y} \quad \leftrightarrow \quad \mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{M}\mathbf{y}_k \,.$$

Perform the same transformation as above on the discrete evolution:

$$\mathbf{V}^{-1}\mathbf{y}_{k+1} = \mathbf{V}^{-1}\mathbf{y}_k + h\mathbf{V}^{-1}\mathbf{M}\mathbf{V}(\mathbf{V}^{-1}\mathbf{y}_k) \quad \overset{\mathbf{z}_k := \mathbf{V}^{-1}\mathbf{y}_k}{\Leftrightarrow} \quad \underbrace{(\mathbf{z}_{k+1})_i = (\mathbf{z}_k)_i + h\lambda_i\,(\mathbf{z}_k)_i}_{\hat{=}\ \text{explicit Euler step for } \dot{z}_i = \lambda_i z_i} \,. \tag{12.2.5}$$

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 $\dot{z} = \lambda_i z$, $i = 1, \ldots, d$.

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. 12.2.1 we have $\lambda_{1/2} = \frac{1}{2}\alpha \pm i\sqrt{\beta - \frac{1}{4}\alpha^2}$ (complex eigenvalues!). How will explicit Euler/explicity RK-methods respond to them?

Example 12.2.3 (Explicit Euler method for damped oscillations).

Consider linear model IVP (12.1.1) 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 $|\exp(\lambda t)| = \exp(\operatorname{Re}\lambda t)$.

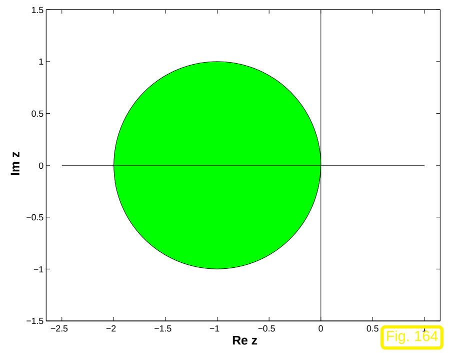
Model problem analysis (→ Ex. 12.1.1, Ex. 12.1.3) for explicit Euler method and $\lambda \in \mathbb{C}$:

Sequence generated by explicit Euler method (11.2.1) for model problem (12.1.1):

$$y_{k+1} = y_k(1 + h\lambda) \,. \tag{12.1.2}$$

▶ $\lim_{k\to\infty} y_k = 0 \quad \Leftrightarrow \quad |1 + h\lambda| < 1 \,.$

timestep constraint to get decaying (discrete) solution !



◁ $\{z \in \mathbb{C}\colon\ |1 + z| < 1\}$

Now we can conjecture what happens in Ex. 12.2.1: the eigenvalue $\lambda_2 = \frac{1}{2}\alpha - i\sqrt{\beta - \frac{1}{4}\alpha^2}$ of $\mathbf{M}$ has 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. 12.1.1. Thus stepsize control must resort to tiny timesteps.

◇

Can we predict this kind of difficulty ?

Example 12.2.4 (Chemical reaction kinetics).

$$\text{reaction:} \quad \underbrace{\mathrm{A} + \mathrm{B} \ \underset{k_1}{\overset{k_2}{\leftrightarrows}}\ \mathrm{C}}_{\text{fast reaction}} \quad , \quad \underbrace{\mathrm{A} + \mathrm{C} \ \underset{k_3}{\overset{k_4}{\leftrightarrows}}\ \mathrm{D}}_{\text{slow reaction}} \tag{12.2.6}$$

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 $\mathbf{y}(t) = (c_A(t), c_B(t), c_C(t), c_D(t))^T$

$$\dot{\mathbf{y}} := \frac{d}{dt}\begin{pmatrix} c_A \\ c_B \\ c_C \\ c_D \end{pmatrix} = \mathbf{f}(\mathbf{y}) := \begin{pmatrix} -k_1 c_A c_B + k_2 c_C - k_3 c_A c_C + k_4 c_D \\ -k_1 c_A c_B + k_2 c_C \\ k_1 c_A c_B - k_2 c_C - k_3 c_A c_C + k_4 c_D \\ k_3 c_A c_C - k_4 c_D \end{pmatrix} .$$

MATLAB computation: $t_0 = 0$, $T = 1$, $k_1 = 10^4$, $k_2 = 10^3$, $k_3 = 10$, $k_4 = 1$

MATLAB-CODE : Explicit integration of stiff chemical reaction equations

```
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];
y0 = [1;1;10;0];
options = odeset('reltol',0.1,'abstol',0.001,'stats','on');
[t,y] = ode45(fun,[0 1],y0,options);
```

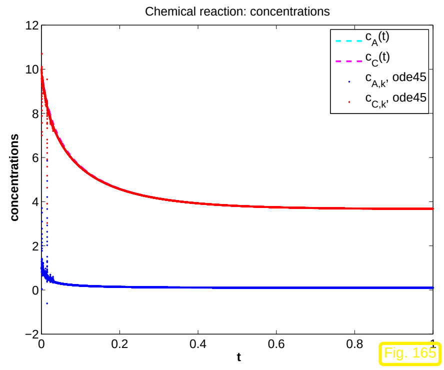


Fig. 165

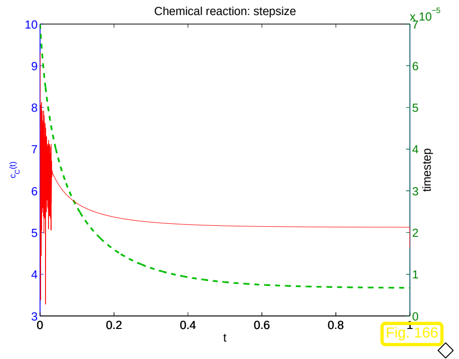


Fig. 166

Example 12.2.5 (Strongly attractive limit cycle).

Autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$

$$\mathbf{f}(\mathbf{y}) := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \mathbf{y} + \lambda(1 - \|\mathbf{y}\|^2)\,\mathbf{y} \,,$$

on state space $D = \mathbb{R}^2 \setminus \{0\}$.

Solution trajectories ($\lambda = 10$) ▷

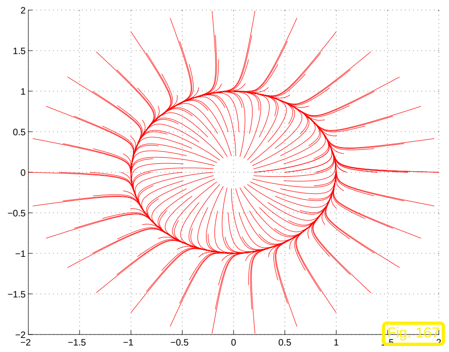


Fig. 167

MATLAB-CODE : Integration of IVP with limit cycle

```
fun = @(t,y) ([-y(2);y(1)] + lambda*(1-y(1)^2-y(2)^2)*y);
tspan = [0,2*pi]; y0 = [1,0];
opts = odeset('stats','on','reltol',1E-4,'abstol',1E-4);
[t45,y45] = ode45(fun,tspan,y0,opts);
```

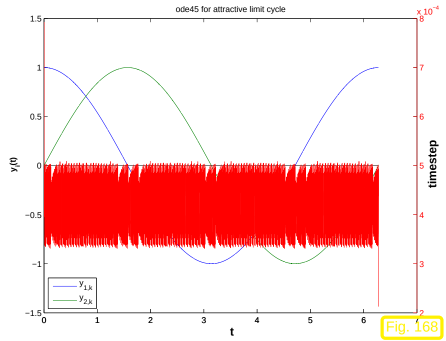
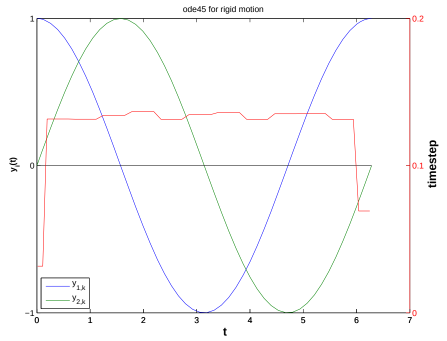


Fig. 168

many (3794) steps ($\lambda = 1000$)



accurate solution with few steps ($\lambda = 0$)

Confusing observation: we have $\|\mathbf{y}_0\| = 1$, which implies $\|\mathbf{y}(t)\| = 1 \quad \forall t$!

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 12.2.1 (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. 12.1.1, 12.2.1,
- Occurrence of strongly attractive fixed points/limit cycles, see Ex. 12.2.5

12.3 (Semi-)implicit Runge-Kutta methods

Example 12.3.1 (Implicit Euler timestepping for decay equation).

Again, model problem analysis: study implicit Euler method (11.2.4) for IVP (12.1.1)

$$\blacktriangleright \quad \text{sequence} \quad y_k := \left(\frac{1}{1-\lambda h}\right)^k y_0 \,. \qquad (12.3.1)$$

$$\Rightarrow \quad \boxed{\operatorname{Re}\lambda < 0 \quad \Rightarrow \quad \lim_{k\to\infty} y_k = 0\,!} \qquad (12.3.2)$$

No timestep constraint: qualitatively correct behavior of $(y_k)_k$ for $\operatorname{Re}\lambda < 0$ and **any** $h > 0$!

◇

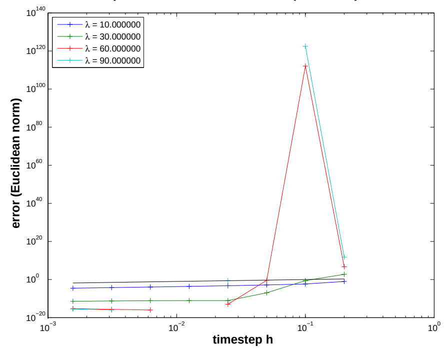
Observe: transformation idea, see (12.2.3), (12.2.5), applies to explicit *and implicit* Euler method alike.

Conjecture: implicit Euler method will not face timestep constraint for stiff problems (⟶ Notion 12.2.1).

Example 12.3.2 (Euler methods for stiff logistic IVP).

☞ Redo Ex. 12.1.1 for implicit Euler method:

Explicit Euler method (11.2.1)



λ large: blow-up of y_k for large timestep h

Implicit Euler method (11.2.4)

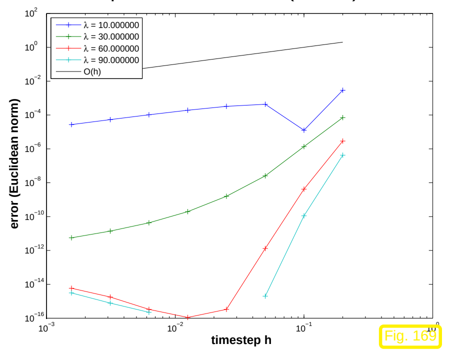


Fig. 163

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. 11.3.1. Can the Runge-Kutta design principle for integrators also yield higher order methods, which can cope with stiff problems?

YES !

Definition 12.3.1 (General Runge-Kutta method). *(cf. Def. 11.4.1)*
For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{s} a_{ij}$, $i,j = 1,\dots,s$, $s \in \mathbb{N}$, *an* s*-stage Runge-Kutta single step method (RK-SSM) for the IVP* (11.1.5) *is defined by*

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j) \,, \quad i = 1,\dots,s \quad , \quad \mathbf{y}_1 := \mathbf{y}_0 + h\sum_{i=1}^{s} b_i \mathbf{k}_i \,.$$

As before, the $\mathbf{k}_i \in \mathbb{R}^d$ *are called* increments*.*

Note: computation of increments $\mathbf{k}_i$ may now require the solution of *(non-linear) systems of equations* of size $s \cdot d$ (⟶ "implicit" method)

Shorthand notation for Runge-Kutta methods

Butcher scheme $\triangleright$

$$\begin{array}{c|c}\mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T\end{array} := \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} . \qquad (12.3.3)$$

Note: now $\mathfrak{A}$ can be a general $s \times s$-matrix.

$\mathfrak{A}$ strict lower triangular matrix	➤ explicit Runge-Kutta method, Def. 11.4.1
$\mathfrak{A}$ lower triangular matrix	➤ diagonally-implicit Runge-Kutta method (DIRK)

Model problem analysis for general Runge-Kutta single step methods (→ Def. 12.3.1): exactly the same as for explicit RK-methods, see (12.1.6), (12.1.7)!

Theorem 12.3.2 (Stability function of Runge-Kutta methods).
The discrete evolution Ψ^h_λ of an s-stage Runge-Kutta single step method (→ Def. 12.3.1) with Butcher scheme $\begin{array}{c|c}\mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T\end{array}$ (see (12.3.3)) for the ODE $\dot{y} = \lambda y$ is a multiplication operator according to

$$\Psi^h_\lambda = \underbrace{1 + z\mathbf{b}^T(\mathbf{I} - z\mathfrak{A})^{-1}\mathbf{1}}_{\text{stability function } S(z)} = \frac{\det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T)}{\det(\mathbf{I} - z\mathfrak{A})} , \quad z := \lambda h , \quad \mathbf{1} = (1,\dots,1)^T \in \mathbb{R}^s .$$

Note: from the determinant 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 (11.2.4) is

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \quad \overset{\text{Thm. 12.3.2}}{\Rightarrow} \quad S(z) = \frac{1}{1-z} . \qquad (12.3.4)$$

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 (→ Notion 12.2.1):

Definition 12.3.3 (L-stable Runge-Kutta method).
A Runge-Kutta method (→ Def. 12.3.1) is L-stable/asymptotically stable, if its stability function (→ Def. 12.3.2) satisfies

$$(i) \qquad \operatorname{Re} z < 0 \;\Rightarrow\; |S(z)| < 1 , \qquad (12.3.5)$$

$$(ii) \qquad \lim_{\operatorname{Re} z \to -\infty} S(z) = 0 . \qquad (12.3.6)$$

Remark 12.3.3 (Necessary condition for L-stability of Runge-Kutta methods).

Consider: Runge-Kutta method (→ Def. 12.3.1) 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

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:

$$\text{Thm. 12.3.2} \;\Rightarrow\; S(-\infty) = 1 - \mathbf{b}^T\mathfrak{A}^{-1}\mathbf{1} . \qquad (12.3.7)$$

$$► \quad \boxed{\text{If } \mathbf{b}^T = (\mathfrak{A})^T_{:,j} \text{ (row of } \mathfrak{A}) \;\Rightarrow\; S(-\infty) = 0} . \qquad (12.3.8)$$

Butcher scheme (12.3.3) for L-stable RK-methods, see Def. 12.3.3 $\triangleright$

$$\begin{array}{c|c}\mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T\end{array} := \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_{s-1} & a_{s-1,1} & \cdots & a_{s-1,s} \\ 1 & b_1 & \cdots & b_s \\ \hline & b_1 & \cdots & b_s \end{array} .$$

Example 12.3.4 (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

◇

Equations fixing increments $\mathbf{k}_i \in \mathbb{R}^d$, $i = 1,\dots,s$, for s-stage implicit RK-method
=
(Non-)linear system of equations with $s \cdot d$ unknowns

Example 12.3.5 (Linearization of increment equations).

- Initial value problem for logistic ODE, see Ex. 11.1.1

$$\dot{y} = \lambda y(1-y) \quad , \quad y(0) = 0.1 \quad , \quad \lambda = 5 \; .$$

- Implicit Euler method (11.2.4) with uniform timestep $h = 1/n$, $n \in \{5, 8, 11, 17, 25, 38, 57, 85, 128, 192, 288, , 432, 649, 973, 1460, 2189, 3284, 4926, 7389\}$.

  & **approximate** computation of y_{k+1} by **1 Newton step** with initial guess y_k

  = semi-implicit Euler method

- Measured error $\mathrm{err} = \max\limits_{j=1,\dots,n} |y_j - y(t_j)|$

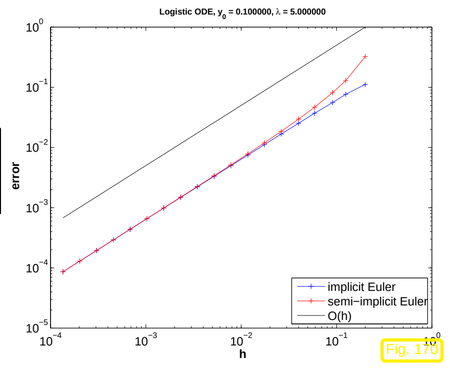


Fig. 170

From (11.2.4) with timestep $h > 0$

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}(\mathbf{y}_{k+1}) \quad \Leftrightarrow \quad F(\mathbf{y}_{k+1}) := \mathbf{y}_{k+1} - h\mathbf{f}(\mathbf{y}_{k+1}) - \mathbf{y}_k = 0 \; .$$

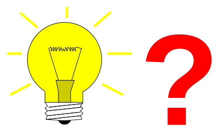
One Newton step applied to $F(\mathbf{y}) = 0$ with initial guess $\mathbf{y}_k$ yields

$$\mathbf{y}_{k+1} = \mathbf{y}_k - D\mathbf{f}(\mathbf{y}_k)^{-1}F(\mathbf{y}_k) = \mathbf{y}_k + (\mathbf{I} - hD\mathbf{f}(\mathbf{y}_k))^{-1}h\mathbf{f}(\mathbf{y}_k) \; .$$

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.

◇



Idea: Use linearized increment equations for implicit RK-SSM

$$\mathbf{k}_i = \mathbf{f}(\mathbf{y}_0) + hD\mathbf{f}(\mathbf{y}_0)\left(\sum_{j=1}^{s} a_{ij}\mathbf{k}_j\right) \quad , \quad i = 1, \dots, s \; . \tag{12.3.9}$$

Linearization does nothing for linear ODEs ➢ stability function (→ Thm. 12.3.2) not affected!

► Class of semi-implicit (linearly implicit) Runge-Kutta methods (Rosenbrock-Wanner (ROW) methods):

$$(\mathbf{I} - ha_{ii}\mathbf{J})\mathbf{k}_i = \mathbf{f}(\mathbf{y}_0 + h\sum_{j=1}^{i-1}(a_{ij} + d_{ij})\mathbf{k}_j) - h\mathbf{J}\sum_{j=1}^{i-1} d_{ij}\mathbf{k}_j \; , \tag{12.3.10}$$

$$\mathbf{J} := D\mathbf{f}\big(\mathbf{y}_0 + h\sum_{j=1}^{i-1}(a_{ij} + d_{ij})\mathbf{k}_j\big) \; , \tag{12.3.11}$$

$$\mathbf{y}_1 := \mathbf{y}_0 + \sum_{j=1}^{s} b_j\mathbf{k}_j \; . \tag{12.3.12}$$

Remark 12.3.6 (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. 11.5:

$\Psi \hat{=}$ RK-method of order 2 $\quad$ $\widetilde{\Psi} \hat{=}$ RK-method of order 3

`ode23s`

integrator for stiff IVP

△

12.4 Differential-algebraic equations

13 Structure Preservation

13.1 Dissipative Evolutions

13.2 Quadratic Invariants

13.3 Reversible Integrators

13.4 Symplectic Integrators

Outlook

Course 401-0674-00: Numerical Methods for Partial Differential Equations

Many fundamental models in science & engineering boil down to

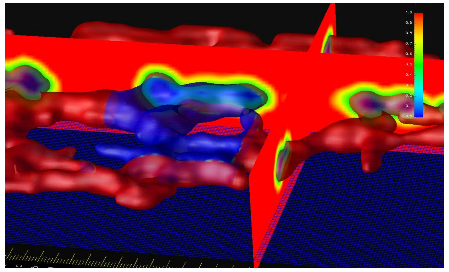
(initial) boundary value problems for partial differential equations (PDEs)

► 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:

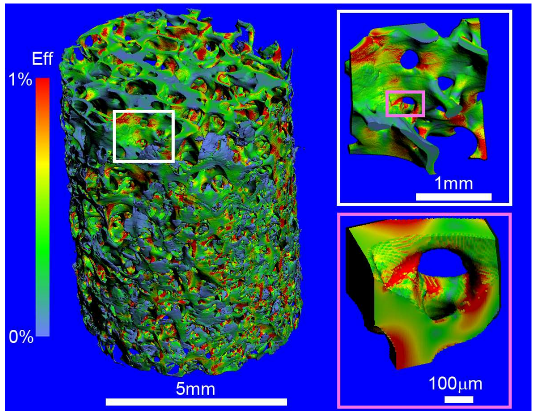
❶ Stationary 2nd-order scalar elliptic boundary value problems



Diffusion boundary value problem:

$$-\operatorname{div}(\mathbf{A}(\boldsymbol{x})\,\mathbf{grad}\,u(\boldsymbol{x})) = f(\boldsymbol{x}) \quad \text{in } \Omega\subset\mathbb{R}^d\,,$$
$$u = g \quad \text{on } \partial\Omega\,.$$

◁ diffusion on the surface (membrane) of the endoplasmic reticulum (I. Sbalzarini, D-INFK, ETH Zürich)



◁ Elastic deformation of human bone (P. Arbenz, D-INFK, ETH Zürich)

❷ Singularly perturbed elliptic boundary value problems

Stationary pollutant transport in water: find concentration $u = u(\boldsymbol{x})$ such that

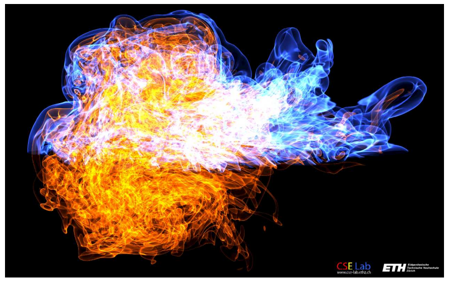
$$-\epsilon\Delta u + \mathbf{v}(\boldsymbol{x})\cdot\mathbf{grad}\,u = 0 \quad \text{in } \Omega \quad , \quad u = g \quad \text{on } \partial\Omega\,.$$

❸ 2nd-order parabolic evolution problems

Heat conduction: find temperature $u = u(\boldsymbol{x}, t)$

$$\frac{\partial}{\partial t}u(\boldsymbol{x},t) - \operatorname{div}(\mathbf{A}(\boldsymbol{x})\,\mathbf{grad}\,u(\boldsymbol{x},t)) = 0 \quad \text{in } \Omega \times [0,T] \quad , \quad \begin{aligned} u(\cdot,t) &= g(t) \quad \text{on } \partial\Omega\,, \\ u(\cdot,0) &= u_0 \quad \text{in } \Omega\,. \end{aligned}$$

❹ Viscous fluid flow problems



Stokes equations:

$$\begin{aligned} -\Delta\mathbf{u} + \mathbf{grad}\,p &= \mathbf{f} \quad \text{in } \Omega\,, \\ \operatorname{div}\mathbf{u} &= 0 \quad \text{in} \Omega\,, \\ \mathbf{u} &= 0 \quad \text{on } \partial\Omega\,. \end{aligned}$$

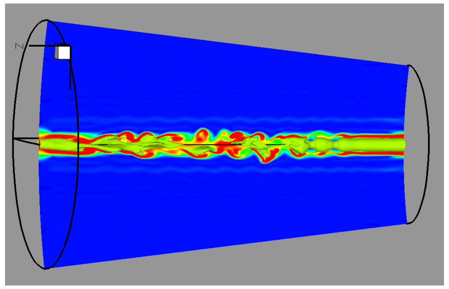
◁ Vortex ring in flow at $\mathtt{Re} = 7500$, (P. Koumoutsakos, D-INFK, ETH Zürich)

❺ Conservation laws

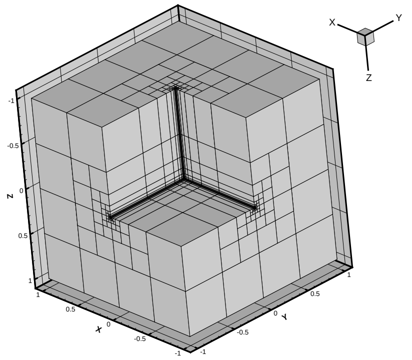
1D scalar conservation law with flux f:

$$\begin{aligned} \frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}(f(u)) &= 0 \qquad \text{in } \mathbb{R}\times\mathbb{R}^+\,, \\ u(x,0) &= u_0(x) \quad \text{for } x \in \mathbb{R}\,. \end{aligned}$$

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)

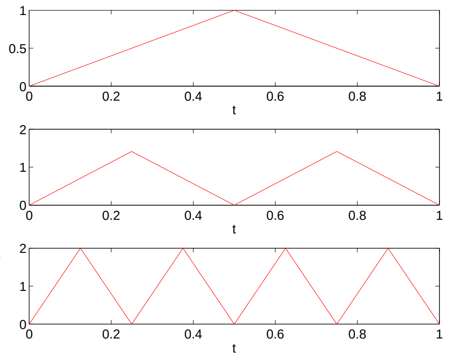
❼ Multilevel preconditioning

FEM, FD, FV

Huge sparse systems of equations

▶ Efficient preconditioners required

1D hierarchical basis ▷



In SS10: Classes: Wed 8-10, HG E 3 and Fri 10-12, HG E 5
Tutorials: Tue 13-15 HG E 21, Thu 13-15 HG D 7.2, Fri 15-17 G E 21

Course: Parallel Computing for Scientific Simulations

Bibliography

[1] H. Amann, *Gewöhnliche Differentialgleichungen*, Walter de Gruyter, Berlin, 1st ed., 1983.

[2] C. Bischof and C. Van Loan, *The WY representation of Householder matrices*, SIAM J. Sci. Stat. Comput., 8 (1987).

[3] F. Bornemann, *A model for understanding numerical stability*, IMA J. Numer. Anal., 27 (2006), pp. 219–231.

[4] E. Brigham, *The Fast Fourier Transform and Its Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[5] 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.

[6] D. Coppersmith and T. Rivlin, *The growth of polynomials bounded at equally spaced points*, SIAM J. Math. Anal., 23 (1992), pp. 970–983.

[7] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progression*, J. Symbgolic Computing, 9 (1990), pp. 251–280.

[8] W. Dahmen and A. Reusken, *Numerik für Ingenieure und Naturwissenschaftler*, Springer, Heidelberg, 2006.

[9] P. Davis, *Interpolation and Approximation*, Dover, New York, 1975.

[10] M. Deakin, *Applied catastrophe theory in the social and biological sciences*, Bulletin of Mathematical Biology, 42 (1980), pp. 647–679.

[11] P. Deuflhard, *Newton Methods for Nonlinear Problems*, vol. 35 of Springer Series in Computational Mathematics, Springer, Berlin, 2004.

[12] P. Deuflhard and F. Bornemann, *Numerische Mathematik II*, DeGruyter, Berlin, 2 ed., 2002.

[13] P. Deuflhard and A. Hohmann, *Numerische Mathematik I*, DeGruyter, Berlin, 3 ed., 2002.

[14] P. Duhamel and M. Vetterli, *Fast fourier transforms: a tutorial review and a state of the art*, Signal Processing, 19 (1990), pp. 259–299.

[15] F. Fritsch and R. Carlson, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal., 17 (1980), pp. 238–246.

[16] M. Gander, W. Gander, G. Golub, and D. Gruntz, *Scientific Computing: An introduction using MATLAB*, Springer, 2005. In Vorbereitung.

[17] 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.

[18] G. Golub and C. Van Loan, *Matrix computations*, John Hopkins University Press, Baltimore, London, 2nd ed., 1989.

[19] 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.

[20] W. Hackbusch, *Iterative Lösung großer linearer Gleichungssysteme*, B.G. Teubner–Verlag, Stuttgart, 1991.

[21] E. Hairer, C. Lubich, and G. Wanner, *Geometric numerical integration*, vol. 31 of Springer Series in Computational Mathematics, Springer, Heidelberg, 2002.

[22] C. Hall and W. Meyer, *Optimal error bounds for cubic spline interpolation*, J. Approx. Theory, 16 (1976), pp. 105–122.

[23] M. Hanke-Bourgeois, *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*, Mathematische Leitfäden, B.G. Teubner, Stuttgart, 2002.

[24] N. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 2 ed., 2002.

[25] 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.

[26] D. McAllister and J. Roulier, *An algorithm for computing a shape-preserving osculatory quadratic spline*, ACM Trans. Math. Software, 7 (1981), pp. 331–347.

[27] C. Moler, *Numerical Computing with MATLAB*, SIAM, Philadelphia, PA, 2004.

[28] 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.

[29] ——, *A geometric theory for preconditioned inverse iteration: III. Sharp convergence estimates*, Tech. Rep. 130, SFB 382, Universität Tübingen, Tübingen, Germany, November 1999.

[30] M. Overton, *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, Philadelphia, PA, 2001.

[31] 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.

[32] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*, vol. 37 of Texts in Applied Mathematics, Springer, New York, 2000.

[33] C. Rader, *Discrete Fourier transforms when the number of data samples is prime*, Proceedings of the IEEE, 56 (1968), pp. 1107–1108.

[34] R. Rannacher, *Einführung in die numerische mathematik*. Vorlesungsskriptum Universität Heidelberg, 2000. http://gaia.iwr.uni-heidelberg.de/.

[35] T. Sauer, *Numerical analysis*, Addison Wesley, Boston, 2006.

[36] J.-B. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE Trans. Pattern Analysis and Machine Intelligence, 22 (2000), pp. 888–905.

[37] M. Stewart, *A superfast toeplitz solver with improved numerical stability*, SIAM J. Matrix Analysis Appl., 25 (2003), pp. 669–693.

[38] J. Stoer, *Einführung in die Numerische Mathematik*, Heidelberger Taschenbücher, Springer, 4 ed., 1983.

[39] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[40] M. Struwe, *Analysis für informatiker*. Lecture notes, ETH Zürich, 2009. https://moodle-app1.net.ethz.ch/lms/mod/resource/index.php?id=145.

[41] F. Tisseur and K. Meerbergen, *The quadratic eigenvalue problem*, SIAM Review, 43 (2001), pp. 235–286.

[42] L. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[43] P. Vertesi, *On the optimal lebesgue constants for polynomial interpolation*, Acta Math. Hungaria, 47 (1986), pp. 165–178.

[44] ——, *Optimal lebesgue constant for lagrange interpolation*, SIAM J. Numer. Aanal., 27 (1990), pp. 1322–1331.

Index

List of Symbols

List of Definitions

Examples and Remarks