Numerical methods for CSE [1]

Stefan Heule

February 2, 2010

Contents

Part I
Systems of equations

1 Computing with matrices and vectors

1.1 Elementary operations

1.1.1 Row-wise and column-wise view of matrix product

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

$$A \cdot B = \Big(A(B)_{:,1}, \ldots, A(B)_{:,k}\Big) \quad = \quad \begin{pmatrix} (A)_{1,:}B \\ \vdots \\ (A)_{n,:}B \end{pmatrix} = A \cdot B$$

1.1.2 Scaling

- Multiplication with diagonal matrix from left $\rightarrow$ row scaling

$$DA = \begin{pmatrix} d_1(A)_{1,:} \\ \vdots \\ d_n(A)_{n,:} \end{pmatrix}$$

- Multiplication with diagonal matrix from left $\rightarrow$ row scaling

$$AD = \Big(d_1(A)_{:,1}, \ldots, d_m(A)_{:,m}\Big)$$

1.2 BLAS - basic linear algebra subroutines

- Level 1: vector operations such as scalar products and vector norms. Asymptotic complexity is $\mathcal{O}(n)$.
- Level 2: vector-matrix operations such as matrix-vector multiplications. Asymptotic complexity is $\mathcal{O}(mn)$.
- Level 3: matrix operations such as matrix additions or multiplications. Asymptotic complexity is $\mathcal{O}(mnk)$.

2 Direct methods for linear systems of equations

2.1 Task

Given a quadratic matrix $A \in \mathbb{K}^{n,n}$ and a vector $b \in \mathbb{K}^n$, find the vector $x \in \mathbb{K}^n$ with

$$Ax = b$$

2.2 Theory

Definition 2.1 (Invertible matrix)
A matrix $A \in \mathbb{K}^{n,n}$ is called *invertible* or *regular*, if and only if there exists a matrix $B \in \mathbb{K}^{n,n}$ with $AB = BA = I$. Equivalently

$$A \in \mathbb{K}^{n,n} \text{ is invertible} \quad \Longleftrightarrow \quad \exists B \in \mathbb{K}^{n,n} : AB = BA = I$$

Definition 2.2 (Rank of a matrix)
The *rank* of a matrix $A \in \mathbb{K}m, n$, denoted by $\text{rank}(A)$, is the maximal number of linearly independent rows/columns of A.

Theorem 2.3 (Criteria for invertibility of a matrix)
A matrix $A \in \mathbb{K}^{n,n}$ is invertible/regular if one of the following equivalent conditions is satisfied:

- $\exists B \in \mathbb{K}^{n,n} : BA = AB = I$
- the columns of A are linearly independent (full column rank)
- the rows of A are linearly independent (full row rank)
- $\det(A) \neq 0$
- $\text{rank}(A) = n$ (full rank)

2.3 Gaussian elimination

The idea is to use row transformations to get a simpler, but equivalent linear system of equations (LSE). Row transformations can be performed by left-multiplications of transformation matrices. The Gaussian elimination of a $n \times n$-matrix has a computational effort of $\mathcal{O}(n^3)$.

2.4 LU-decomposition/LU-factorization

Definition 2.4 (Types of matrices)
A matrix $A = (a_{ij}) \in \mathbb{R}^{m,n}$ is called

- *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 $a_{ii} = 1$ for $i = 1, \ldots, \min\{m, n\}$,

Lemma 2.5 (Group of regular diagonal/triangular matrices)
If both A and B are diagonal (upper triangular) [lower triangular], then AB and A^{-1} (if A is regular) are diagonal (upper triangular) [lower triangular].

Lemma 2.6 (Existence of LU-decomposition (without pivoting))
The LU-decomposition of $A \in K^{n,n}$ exists if and only if all sub-matrices $(A)_{1:k,1:k}, 1 \leq k \leq n$ are regular.

Lemma 2.7 (Uniqueness of LU-decomposition)
The LU-decomposition $A = LU$ of $A \in K^{n,n}$ is unique.

Proof. Let L_1, U_1 and L_2, U_2 be two different LU-decompositions, thus $A = L_1U_1 = L_2U_2$. Because L_2 and U_1 are regular, we can transform the equation as follows:

$$L_1U_1 = L_2U_2 \quad \Longrightarrow \quad L_2^{-1}L_1 = U_2U_1^{-1}$$

On the left-hand side we now have the product of two normalized lower triangular matrix, which itself is also a normalized lower triangular matrix (see Lemma 2.5). The same holds for the right-hand side and upper triangular matrices.
The only common element of normalized lower triangular matrices and upper triangular matrices is the identity matrix, thus

$$L_2^{-1}L_1 = U_2U_1^{-1} = I$$

This directly implies $L_1 = L_2$ and $U_1 = U_2$. □

Definition 2.8 (LU-decomposition)
The LU decomposition of $A \in \mathbb{K}^{n,n}$ is a matrix decomposition which writes a matrix as the product of a lower triangular matrix L and an upper triangular matrix U

$$A = LU$$

Remark 2.8.1: The costs for an LU-decomposition are $\mathcal{O}(n^3)$, and $\mathcal{O}(n^2)$ for the forward/backward substitutions.

2.4.1 Block LU-decomposition

Let $A_{11} \in \mathbb{K}^{n,n}$ be regular, and $A_{12} \in \mathbb{K}^{n,m}, A_{21} \in \mathbb{K}^{m,n}, A_{22} \in \mathbb{K}^{m,m}$.

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ 0 & S \end{pmatrix}$$

where $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is called the *Schur complement*.

2.4.2 Pivoting

Due to rounding errors, it is important from what row the pivot is chosen. In practice, *partial pivoting* is mostly used, where the pivot row index j is chosen from $\{i, \dots, n\}$, such that

$$\frac{|a_{ji}|}{\sum_{l=i}^{n} |a_{jl}|} \to \max$$

Definition 2.9 (Permutation matrix)
An *n-permutation*, $n \in \mathbb{N}$, is a bijective mapping $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$. The corresponding *permutation matrix* $P_\pi \in \mathbb{K}^{n,n}$ is defined by

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

Remark 2.9.1 (Properties of permutation matrices): For any permutation matrix $P \in \mathbb{K}^{n,n}$ the following statements hold:

- $P_\pi^H = P_\pi^{-1} = P_{\pi^{-1}}$. That is, permutation matrices are orthogonal/unitary.
- $P_\pi A$ performs the π-permutation on the rows of $A \in \mathbb{K}^{n,n}$
- AP_π performs the π-permutation on the columns of $A \in \mathbb{K}^{n,n}$

Lemma 2.10 (Existence of LU-decomposition (with pivoting))
For any regular matrix $A \in K^{n,n}$ there is a permutation matrix $P \in \mathbb{K}^{n,n}$, a normalized lower triangular matrix $L \in \mathbb{K}^{n,n}$, and a regular upper triangular matrix $U \in \mathbb{K}^{n,n}$, such that

$$PA = LU$$

2.5 Machine Arithmetic

A computer can only handle finitely many number, not $\mathbb{R}$. These numbers are called *machine numbers*, denoted $\mathbb{M}$, and they are a discrete subset of $\mathbb{R}$.
Obviously $\mathbb{M}$ is not closed under elementary arithmetic operations such as $+, -, \cdot, /$, and therefore round-off errors are inevitable.

Definition 2.11 (Machine numbers)
Given

- a basis $B \in \mathbb{N} \setminus \{1\}$,
- an exponent range $\{e_{\min}, \dots, e_{\max}\}$, $e_{\min}, e_{\max} \in \mathbb{Z}$, $e_{\min} < e_{\max}$,
- and a number $m \in \mathbb{N}$ of digits for the mantissa,

the corresponding set of machine numbers is

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

Remark 2.11.1 (Largest/smallest elements in $\mathbb{M}$ (in modulus)): $x_{\max} = (1 - B^{-m}) \cdot B^{e_{\max}}$ and $x_{\min} = B^{-1} \cdot B^{e_{\min}}$

Definition 2.12 ("Axiom" of round-off analysis)
There is a small positive number eps, the machine precision, such that for elementary arithmetic operations $\star \in \{+, -, *, /\}$ and "hard-wired" functions $f \in \{\exp, \sin, \cos, \log, \ldots\}$ holds

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

with $|\delta| \leq$ eps.

Remark 2.12.1: The problem with these tiny round-off errors is that they can both amplify and/or accumulate over time.

2.6 Stability of Gaussian elimination

2.6.1 Vector norms and matrix norms

Definition 2.13 (Norm)
Let X be a vector space over a field $\mathbb{K}$. A mapping $\|\cdot\| : X \to \mathbb{R}_0^+$ is called a *norm* on X if it satisfies

1. $\forall x \in X : x \neq 0 \iff \|x\| > 0$ (definite),
2. $\|\lambda x\| = |\lambda| \, \|x\| \;\; \forall x \in X, \lambda \in \mathbb{K}$ (homogeneous),
3. $\|x + y\| \leq \|x\| + \|y\| \;\; \forall x, y \in X$ (triangle inequality).

Definition 2.14 (Matrix norm)
Given a vector norm $\|\cdot\|$ on $\mathbb{R}^n$, the associated *matrix norm* is defined by

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

Lemma 2.15
Given a matrix $A \in \mathbb{K}^{n,n}$ with $A = A^H$ (i.e. a hermitian matrix), the two-norm of A can be expressed as follows

$$\|A\|_2 = \max_{x \neq 0} \frac{|x^H A x|}{\|x\|_2^2}$$

Corollary 2.16
For $A \in \mathbb{K}^{m,n}$ the 2-matrix-norm $\|A\|_2$ is the square root of the largest (in modulus) eigenvalue of $A^H A$.

2.6.2 Numerical Stability

Definition 2.17 (Stable algorithm)
An algorithm $\tilde{F}$ for solving a problem $F : X \to Y$ is *numerically stable*, if for all $x \in X$ its result $\tilde{F}(x)$ (affected by round-off) is the exact result for "slightly perturbed" data:

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

Remark 2.17.1: The stability is a property of a given algorithm for a specific problem.

Theorem 2.18 (Stability of Gaussian elimination with partial pivoting)
Let $A \in \mathbb{R}^{n,n}$ be regular and $A^{(k)} \in \mathbb{R}^{n,n}$, $k = 1, \ldots, n-1$ denote the intermediate matrices arising in the k-th step of the Gaussian elimination with partial pivoting. For the approximate solution $\tilde{x} \in \mathbb{R}^n$ of the LSE $Ax = b$, computed with the Gaussian elimination with partial pivoting (based on machine arithmetic with machine precision eps) there is $\Delta A \in \mathbb{R}^{n,n}$ with

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

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

Remark 2.18.1: If ρ is small, the Gaussian elimination with partial pivoting is stable.

Remark 2.18.2: Even though ρ can grow exponentially with n, ρ is rather small for almost all practical system matrices.

Definition 2.19
Given an approximate solution $\tilde{x} \in \mathbb{K}^n$ of the LSE $Ax = b$, its *residual* is the vector

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

2.6.3 Conditioning

Definition 2.20 (Condition number)
The condition number of a matrix $A \in \mathbb{R}^{n,n}$ is given as follows

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

Remark 2.20.1: The condition number depends on the choice of norm $\|\cdot\|$.

Theorem 2.21 (Conditioning of LSEs)
If A is regular, $\|\Delta A\| < \left\|A^{-1}\right\|^{-1}$ and for the perturbed linear system

$$Ax = b \quad \leftrightarrow \quad (A + \Delta A)\tilde{x} = b + \Delta b \quad \Longrightarrow \quad (A + \Delta A)(\tilde{x} - x) = \Delta b - \Delta A x$$

then

$$\underbrace{\frac{\|x - \tilde{x}\|}{\|x\|}}_{\text{relative error}} \leq \frac{\operatorname{cond}(A)}{1 - \operatorname{cond}(A)\,\|\Delta A\| / \|A\|} \underbrace{\left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|}\right)}_{\text{relative perturbations}}$$

Remark 2.21.1: If $\operatorname{cond}(A) \gg 1$, even small perturbations in A can lead to large relative errors in the solution of the LSE. In this situation, a stable algorithm can produce solutions with large relative error!

Remark 2.21.2: If small changes in the data produce only small perturbations of the result, then this problem is *well-conditioned*. Otherwise, i.e. if small changes in the data lead to large perturbations in the result, the problem is called *ill-conditioned*.

Definition 2.22 (Sensitivity)
The *sensitivity* of a problem (i.e. for given data) measures the impact of small perturbations of the data on the result. The sensitivity is sometimes called *condition*.

Remark 2.22.1: Remember that the sensitivity is a property of the problem, whereas stability is a characteristic of an algorithm and does therefore not depend on the input.

2.7 Sparse matrices

Definition 2.23 (Sparse matrix)
Intuitively speaking, a matrix $A \in \mathbb{K}^{m,n}$ is called *sparse*, if

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

More rigorously, a family $(A^{(l)})_{l \in \mathbb{N}}$ of matrices with $A^{(l)} \in \mathbb{K}^{m_l, n_l}$ is sparse, if

$$nnz(A^{(l)}) := \#\{(i,j) \in \{1,\dots,m_l\} \times \{1,\dots,n_l\} : a_{ij}^{(l)} \neq 0\} = \mathcal{O}(n_l + m_l)$$

2.7.1 Sparse matrix storage formats

There is a wide range of sparse matrix storage formats, that only store non-zero entries. MATLAB uses for historical reasons the *compressed column storage* (CCS) format. As an example, here is how the compressed row storage format (CRS) works: This format stores all the data in three arrays:

1. `double* val` of size $\operatorname{nnz}(A)$
2. `unsigned int* col_ind` of size $\operatorname{nnz}(A)$
3. `unsigned int* row_ptr` of size $n+1$ with `row_ptr[n+1] = nnz(A)+1`

An access to the matrix entry $a_{ij} \neq 0$ then works the following way:

$$\texttt{val[k]} = a_{ij} \quad \Longleftrightarrow \quad \begin{cases} \texttt{col\_ind[k]} = j \\ \texttt{row\_ptr[i]} \leq k < \texttt{row\_ptr[i+1]} \end{cases} \qquad \text{for } 1 \leq k \leq \texttt{nnz(A)}$$

2.7.2 LU-factorization of sparse matrices

If a matrix A is sparse, this generally does not mean that the LU-factors L and U are sparse as well. The same holds for A^{-1}, which can be a full matrix even if A, L and U are sparse. This motivates the following definition:

Definition 2.24 (Fill-in)
Let $A = LU$ be an LU-factorization of $A \in \mathbb{K}^{n,n}$. If $l_{ij} \neq 0$ or $u_{ij} \neq 0$ even though $a_{ij} = 0$, then we encounter *fill-in* at position (i,j).

It is important to note that pivoting has a great influence on how dense the LU-factors will get. There exist matrices for which the LU-factors are full triangular matrices, whereas when computing the same factors without pivoting, both L and U are sparse.
Just stopping to use pivoting however is not a solution, because pivoting ensures stability.

2.7.3 Banded matrices

Definition 2.25 (Bandwidth of matrices)
For $A \in \mathbb{K}^{m,n}$ we define

- $\overline{m}(A) := \min\{k \in \mathbb{N} \mid j - i > k \to a_{ij} = 0\}$ the *upper bandwidth*, and
- $\underline{m}(A) := \min\{k \in \mathbb{N} \mid i - j > k \to a_{ij} = 0\}$ the *lower bandwidth*, and
- $m(A) := \overline{m}(A) + \underline{m}(A) + 1$ the *bandwidth* of A.

Remark 2.25.1: A diagonal matrix A has $m(A) = 1$. On the other hand, if $\overline{m}(A) = \underline{m}(A) = 1$, then this matrix A is called tridiagonal. Generally, one speaks of a *banded matrix* if $m(A) \ll n$.

Remark 2.25.2: Also note that one can generalize the concept of the diagonal to super-diagonals (above the diagonal) and sub-diagonals. $\overline{m}(A)$ and $\underline{m}(A)$ give the number of nonzero super-diagonals and sub-diagonals, respectively.

Definition 2.26 (Matrix envelope)
For $A \in \mathbb{K}^{n,n}$ we define

row bandwidth	$m_i^R(A) := \max\{0, i - j \mid a_{ij} \neq 0, 1 \leq j \leq n\}, i \in \{1, \ldots, n\}$
column bandwidth	$m_i^R(A) := \max\{0, i - j \mid a_{ij} \neq 0, 1 \leq j \leq n\}, i \in \{1, \ldots, n\}$
envelope	$\operatorname{env}(A) := \left\{ (i,j) \in \{1, \ldots, n\}^2 \mid \begin{array}{l} i - m_i^R(A) \leq j \leq i, \\ j - m_j^C(A) \leq i \leq j \end{array} \right\}$

Theorem 2.27 (Envelope and fill-in)
If $A \in \mathbb{K}^{n,n}$ is regular with LU-decomposition $A = LU$, then fill-in is confined to $\operatorname{env}(A)$.

The last theorem can be exploited by reordering rows and columns in the system matrix, and therefore reducing the size of the envelope. There exists a wide range of numerical libraries for solving LSE with sparse system matrices, and it is highly recommended to use them, rather than start writing an own solver.

2.7.4 Stable Gaussian elimination without pivoting

Pivoting can trigger huge fill-in that would not occur without it. On the other hand, pivoting is essential for the stability of the Gaussian elimination/LU-factorization. It would be very desirable to have an a priori criteria, when the Gaussian elimination remains stable even without pivoting.

Definition 2.28 (Symmetric positive definite (s.p.d.) matrices)
A matrix $M \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$ is called *symmetric positive definite* (s.p.d.), if

$$M = M^H \quad \wedge \quad \forall x \neq 0 : x^H M x > 0$$

If $x^H M x \geq 0$ for all $x \in \mathbb{K}^n$, M is called *positive semi-definite*.

Lemma 2.29 (Necessary conditions for s.p.d.)
For a symmetric/Hermitian positive definite matrix $M = M^H \in \mathbb{K}^{n,n}$ holds true:

1. $m_{ii} > 0$, $i = 1, \ldots, n$,
2. $m_{ii} m_{jj} - |m_{ij}|^2 > 0$, $\forall 1 \leq i <\leq n$,
3. all eigenvalues of M are positive. (this criteria is also sufficient)

Definition 2.30 (Diagonally dominant matrix)
A matrix $A \in \mathbb{K}^{n,n}$ is called *diagonally dominant*, if

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

It is called *strictly diagonally dominant*, if

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

Lemma 2.31
A diagonally dominant Hermitian/symmetric matrix with non-negative diagonal entries is positive semi-definite.
A strictly diagonally dominant Hermitian/symmetric matrix with positive diagonal entries is positive definite.

Theorem 2.32 (Gaussian elimination for s.p.d. matrices)
Every symmetric/Hermitian positive definite matrix possesses an LU-decomposition (i.e. Gaussian elimination feasible without pivoting)

Lemma 2.33 (Cholesky decomposition)
For any s.p.d. $A \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$, there is a unique upper triangular matrix $R \in \mathbb{K}^{n,n}$ with $r_{ii} > 0$, $i = 1, \ldots, n$, such that $A = R^H R$. This decomposition is called the *Cholesky decomposition*.

Remark 2.33.1: The computational costs (both time and space) of the Cholesky decomposition are only half the costs of the LU-factorization.

Remark 2.33.2: Solving LSE with an s.p.d. system matrix via Cholesky decomposition and forward/backward substitution is numerically stable.

Lemma 2.34 (LU-factorization of diagonally dominant matrices)
For any regular, diagonally dominant matrix A with positive diagonal entries, the Gaussian elimination is feasible without pivoting (partial pivoting would select row i in step i).

2.8 QR-factorization/QR-decomposition

Definition 2.35 (Unitary and orthogonal matrices)
$Q \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$.

- Q is called *unitary*, if $Q^{-1} = Q^H$.
- Q is called *orthogonal*, if $Q^{-1} = Q^T$.

Remark 2.35.1: Remember that for any unitary Q it holds true:

- $\operatorname{cond}(Q) = 1$,
- all rows/columns have Euclidean norm = 1,
- all rows/columns are pairwise orthogonal with regard to the Euclidean inner product,
- $|\det(Q)| = 1$,
- all eigenvalues are from $\{z \in \mathbb{Z} \mid : |z| = 1\}$
- $\|QA\|_2 = \|A\|_2$ for any matrix $A \in \mathbb{K}^{n,n}$

Theorem 2.36 (Criteria for unitary)

$$Q \in \mathbb{C}^{n,n} \text{ unitary} \qquad \Longleftrightarrow \qquad \|Qx\|_2 = \|x\|_2 \quad \forall x \in \mathbb{C}^n$$

There are two drawbacks of the LU-factorization: Firstly, pivoting is often required, which destroys the structure and often leads to fill-in. And secondly, there is the possible (theoretical) instability of partial pivoting.
The stability problems of the Gaussian elimination without pivoting are due to the fact that row transformations can convert well-conditioned to ill-conditioned matrices. This motivates the use of unitary row transformations to convert a matrix to upper triangular from.
There are at least two possible choices of unitary row transformations, reflections or rotations. Both are described further below.

2.8.1 Householder reflections

Householder reflection, represented through an orthogonal matrix $Q \in \mathbb{R}^{n,n}$ transform any vector $a \in \mathbb{R}^n$ to the (scaled) unit vector: $Qa = \|a\|_2 e_1$.

Definition 2.37 (Householder reflection)

$$Q = H(v) := I - 2\frac{vv^H}{v^H v} \quad \text{with } v = \tfrac{1}{2}(a \pm \|a\|_2 e_1)$$

Remark 2.37.1: Because Householder reflections transform a whole row in one step, they should be used for fully populated matrices.

Remark 2.37.2: To store the successive transformations, only store the "Householder vectors" v_j. This can be done in place, by storing the vectors v_j (with decreasing size) in the lower triangular part of A.

2.8.2 Givens rotations

Givens rotations perform one single 0 in the target vector, as opposed to eliminations all but the first component like Householder reflections.

Definition 2.38

$$G_{1k}(a_1, a_k)a := \begin{pmatrix} \overline{\gamma} & \cdots & \overline{\sigma} & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots \\ -\sigma & \cdots & \gamma & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_k \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_1^{(1)} \\ \vdots \\ 0 \\ \vdots \\ a_n \end{pmatrix} \qquad \gamma = \frac{a_1}{\sqrt{|a_1|^2 + |a_k|^2}} \qquad \sigma = \frac{a_k}{\sqrt{|a_1|^2 + |a_k|^2}}$$

Remark 2.38.1: Because one Givens rotation is faster then one Householder reflection, this approach is preferable for spare matrices, where only few entries below the main diagonal are non-zero.

Remark 2.38.2: As with Householder reflections, storing the full representation of the orthogonal matrices Q_k is very inefficient. Rather store $G_{ij}(a, b)$ as triple (i, j, ρ) with ρ being a number that encodes both γ and σ.

2.8.3 QR-factorisation

Successive Householder reflections or Givens rotations give us now

$$Q_{n-1}Q_{n-2}\cdots Q_1 A = R$$

This can be easily transformed to the *QR-factorisation* $A = QR$ with $Q := Q_1^H \cdots Q_{n-1}^H$. In this decomposition, Q is an unitary matrix, and R is an upper triangular matrix.

Lemma 2.39 (Uniqueness of QR-factorization)
The "economical QR-factorization of $A \in \mathbb{K}^{m,n}$, $m \geq n$, with $\operatorname{rank}(A) = n$ is unique, if we demand $r_{ii} > 0$.

In MATLAB, there is `qr` to compute the QR-factorization:

`[Q,R] = qr(A)`	$Q \in \mathbb{K}^{m,m}, R \in \mathbb{K}^{m,n}$ for $A \in \mathbb{K}^{m,n}$	in $\mathcal{O}(m^2 n)$
`[Q,R] = qr(A,0)`	$Q \in \mathbb{K}^{m,n}, R \in \mathbb{K}^{n,n}$ for $A \in \mathbb{K}^{m,n}$, $m > n$	in $\mathcal{O}(mn^2)$

The QR-decomposition is *numerically stable* for any $A \in \mathbb{C}^{m,n}$. Therefore, one can solve any LSE in $\mathcal{O}(n^3)$ with the QR-decomposition ($\mathcal{O}(n^3)$ about twice as expensive as LU-decomposition without pivoting), orthogonal transformations ($\mathcal{O}(n^2)$, if stored efficiently) and backward substitution ($\mathcal{O}(n^2)$).
A last concern is the fill-in for the QR-decomposition: Let $A \in \mathbb{C}^{n,n}$ with $A = QR$ have bandwidth $m(A)$, then it holds true

$$m(R) \leq m(A)$$

2.9 Modification techniques

Lemma 2.40 (Sherman-Morrison-Woodbury formula)
For regular $A \in \mathbb{K}^{n,n}$, and $U, V \in \mathbb{K}^{n,k}$, $n, k \in \mathbb{N}$, $k \leq n$, holds

$$(A + UV^H)^{-1} = A^{-1} - A^{-1}U(I + V^H A^{-1} U)^{-1} V^H A^{-1}$$

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

Remark 2.40.1: For $k = 1$ this formula provides an efficient way to implement a rank-1-modification of the LU-decomposition. The modification however might suffer from numerical instability.

MATLAB provides a function `[Q1,R1] = qrupdate(Q,R,u,v)` to perform a rank-1-modification of the Q and R factors.

3 Iterative methods for non-linear systems of equations

The concept of non-linear systems of equations covers an extremely wide variety of problems. Nevertheless, this chapters will mainly look at "generic" methods to solve such systems.
The general problem can be described as follows: Given a function $F : D \subset \mathbb{R}^n \to \mathbb{R}^n$, $n \in \mathbb{N}$, find $x \in \mathbb{R}^n$ such that

$$F(x) = 0$$

Without further assumptions, it is easy to see that this type of problem does not need to have a solution, and if it has one, it may have multiple, or even infinitely many.

3.1 Iterative methods

In the previous chapter we were only concerned about finding numerically stable and efficient ways to solve the problems. Any method discussed would have provided the correct solution if carried out with arbitrary precision.
With non-linear systems we have a completely different situation. It is by no means always possible to even find a solution in finite many steps, so one tries to approximate the solution as far as possible.
Iterative methods are one way of doing this by generating a sequence $(x^{(k)})_{k\in\mathbb{N}_0}$ which (hopefully) converges against a solution x^*.

Definition 3.1 (m-point-method)
An iterative method, that computes $x^{(k)}$ based on F and (one or several) $x^{(n)}$ with $n < k$ and uses initial guess(es) $x^{(0)}, \ldots, x^{(m-1)}$ is called an *m-point-method*. Formally:

$$x^{(k)} = \Phi_F(x^{(k-1)}, \ldots, x^{(k-m)}) \qquad \text{with given} \quad x^{(0)}, \ldots, x^{(m-1)}$$

Definition 3.2 (Convergence of iterative methods)
An iterative method *converges* (for fixed initial guess(es)) if

$$x^{(k)} \to x^* \text{ and } F(x^*) = 0$$

Definition 3.3 (Consistency of iterative methods)
An iterative method is *consistent* with $F(x) = 0$ if

$$\Phi_F(x^*, \ldots, x^*) = x^* \quad \iff \quad F(x^*) = 0$$

Definition 3.4 (Local and global convergence)
An iterative method *converges locally* to $x^* \in \mathbb{R}^n$ if there is a neighbourhood $U \subset D$ of x^*, such that

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

for the sequence $(x^{(k)})_{k\in\mathbb{N}_0}$ of iterates. If $U = D$, the iterative method is *globally convergent*.

3.1.1 Speed of convergence

To measure the speed of convergence, we inspect the *iteration error*

$$e^{(k)} := x^{(k)} - x^*$$

and especially the decrease of the error norm $\varepsilon_k := \left\|e^{(k)}\right\|$.

Definition 3.5 (Linear convergence)
A sequence $x^{(k)}$, $k = 0, \ldots$, in $\mathbb{R}^n$ *converges linearly* to $x^* \in \mathbb{R}^n$, if

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

Remark 3.5.1: The least upper bound for L give the *rate of convergence*.

Remark 3.5.2: It is important to remember that certain facts depend on the choice of norm, while others don't:

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

Remark 3.5.3: If the norm of the iteration errors build a straight line (or are even faster decreasing) in the lin-log plot (against the iteration index), this indicates linear convergence.

$$\varepsilon_k \le L^k \varepsilon_0 \quad \Longrightarrow \quad \log(\varepsilon_k) \le k \log(L) + \log(\varepsilon_0)$$

Definition 3.6 (Order of convergence)
A convergent sequence $x^{(k)}$, $k = 0, 1, 2, \ldots$, in $\mathbb{R}^n$ converges with *order* p to $x^* \in \mathbb{R}^n$, if

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

with $C < 1$ for $p = 1$ (linear convergence).

Remark 3.6.1: One can determine an approximation of the convergence order in a numerical experiment.

$$\varepsilon_{k+1} \approx C\varepsilon_k^p \quad \Longrightarrow \quad \log \varepsilon_{k+1} \approx \log C + p \log \varepsilon_k \quad \Longrightarrow \quad p \approx \frac{\log \varepsilon_{k+1} - \log \varepsilon_k}{\log \varepsilon_k - \log \varepsilon_{k-1}}$$

Remark 3.6.2: The number of significant digits for any convergent second order iteration doubles with every step.

3.1.2 Termination criteria

Usually, even without round-off errors, we cannot expect to arrive at the exact solution x^* after finitely many steps. Thus, we can only hope to compute an approximate solution by accepting $x^{(K)}$ for some $K \in \mathbb{N}$ as result. Termination criteria are used to determine a suitable value for K.
An ideal solution would be $K = \operatorname{argmin} \left\{ k \in \mathbb{N}_0 \mid \left\| x^{(k)} - x^* \right\| < \theta \right\}$.
There are a wide range of termination criteria, but they can be categorized as follows:

- *A priori termination*: Stop iteration after a fixed number of steps, possibly depending on $x^{(0)}$. This is very nice, but hardly ever possible.
- *A posteriori termination*: Use the already computed iterates to decide when to stop.

Here are a few examples on termination criteria, though some might be stupid:

- *Reliable termination*: Stop the iteration after a prescribed tolerance τ is reached: $\left\| x^{(k)} - x^* \right\| \leq \tau$. The only problem is, that if x^* is known, we do not need to iterate at all, because we know the solution.
- *Use that* $\mathbb{M}$ *is finite*: Iterate until the solution becomes stationary. In general, this is extremely inefficient, and might even result in an infinite loop.
- *Residual based termination*: Stop the iteration when $\left\| F(x^{(k)}) \right\| \leq \tau$. This works, but there is no guaranteed accuracy.

If the convergence rate of a linearly convergent iteration is known, one can derive an *a posteriori criterion for linearly convergent iterations*:

$$\left\| x^{(k+1)} - x^* \right\| \leq \frac{L}{1 - L} \left\| x^{(k+1)} - x^{(k)} \right\|$$

3.2 Fixed point iterations

Definition 3.7 (Fixed point iteration)
A *fixed point iteration* is defined by the iteration function $\Phi : U \subset \mathbb{R}^n \to \mathbb{R}^n$ and an initial guess $x^{(0)} \in U$. The iterates are defined by

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

Remark 3.7.1: The sequence of iterates need not to be well defined, because $x^{(k)} \notin U$ is possible.

Remark 3.7.2: A fixed point iteration is a 1-point-method.

Definition 3.8 (Consistency of fixed point iterations)
A fixed point iteration $x^{(k+1)} = \Phi(x^{(k)})$ is *consistent* with $F(x) = 0$, if

$$F(x) = 0 \quad \text{and} \quad x \in U \cap D \qquad \Longleftrightarrow \qquad \Phi(x) = x$$

Lemma 3.9 (Construction of fixed point iteration)
To construct a fixed point iteration that is consistent with $F(x) = 0$, rewrite $F(x) = 0 \iff \Phi(x) = x$ and then use the fixed point iteration defined by

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

Remark 3.9.1: Note that in general there might be many ways to transform $F(x) = 0$ into a fixed point form.

3.2.1 Convergence of fixed point iterations

Definition 3.10 (Contractive mapping)
A mapping $\Phi : U \subset \mathbb{R}^n \to \mathbb{R}^n$ is called *contractive* (with regard to the norm $\|\cdot\|$ on $\mathbb{R}^n$), if

$$\exists L < 1 : \quad \|\Phi(x) - \Phi(y)\| \leq L \|x - y\| \quad \forall x, y \in U$$

Remark 3.10.1: If a fixed point iteration induced by a contractive mapping Φ converges at least linearly.

Theorem 3.11 (Banach's fixed point theorem)
If $D \subset \mathbb{K}^n$ with $\mathbb{K} = \mathbb{R}, \mathbb{C}$ is closed and $\Phi : D \to D$ is contractive, then there exists a unique fixed point $x^* \in D$ with $\Phi(x^*) = x^*$, which is the limit of the sequence of iterates $x^{(k+1)} := \Phi(x^{(k)})$ for any $x^{(0)} \in D$.

Lemma 3.12 (Sufficient condition for contractive mapping)
If $\Phi : U \subset \mathbb{R}^n \to \mathbb{R}^n$, $\Phi(x^*) = x^*$, Φ differentiable in x^* and $\|D\Phi(x^*)\| < 1$, then mapping Φ is contractive.

Remark 3.12.1: Note that such a mapping Φ defines a fixed point iteration that converges locally and at least linearly.

Lemma 3.13 (Sufficient condition for local linear convergence of fixed point iteration)
Let U be convex and $\Phi : U \subset \mathbb{R}^n \to \mathbb{R}^n$ continuously differentiable with

$$L := \sup_{x \in U} \|D\Phi(x)\| < 1$$

If $\Phi(x^*) = x^*$ for some interior point $x^* \in U$, then the fixed point iteration given by $x^{(k+1)} = \Phi(x^{(k)})$ converges to x^* locally at least linearly.

Remark 3.13.1: A set U is *convex*, if and only if for every two points, the straight line connecting the two lies completely in U.

Theorem 3.14 (Taylor's formula)
If $\Phi : U \subset \mathbb{R} \to \mathbb{R}$, U interval, is $m+1$ times continuously differentiable, $x \in U$

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

Lemma 3.15 (High order local convergence of fixed point iterations)
If $\Phi : U \subset \mathbb{R} \to \mathbb{R}$ is $m+1$ times continuously differentiable, $\Phi(x^*) = x^*$ for some x^* in the interior of U, and $\Phi^{(l)}(x^*) = 0$ for $l = 1, \ldots, m$, $m \geq 1$, then the fixed point iteration converges locally to x^* with *order* $\geq m + 1$.

If the contraction factor $0 \leq L < 1$ of a contractive fixed point iteration, there are two termination criteria available:

a priori termination criterion	$\left\|x^* - x^{(k)}\right\| \leq \frac{L^k}{1-L} \left\|x^{(1)} - x^{(0)}\right\|$
a posteriori termination criterion	$\left\|x^* - x^{(k)}\right\| \leq \frac{L}{1-L} \left\|x^{(k)} - x^{(k-1)}\right\|$

3.3 Zero finding

In this section we focus on the scalar case $n = 1$. Given a continuous function $F : I \subset \mathbb{R} \to \mathbb{R}$ with an interval I, find $x^* \in I$ such that $F(x^*) = 0$.

3.3.1 Bisection

The idea is to use the ordering of the real numbers and the intermediate value theorem. Given $a, b \in I$ with $F(a)F(b) < 0$ (i.e. $F(a)$ and $F(b)$ have different signs), then there is at leas one $x^* \in]\min\{a,b\}, \max\{a,b\}[$ with $F(x^*) = 0$.
This can be used to create a sequence of values that converge to a zero of F.

```
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

end
```

Remark 3.15.1: 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. The iteration will terminate when $a \tilde{+} \frac{1}{2}(b-a) = a$, which implies that the relative error is smaller or equal `eps`.

Remark 3.15.2: The advantages of bisection are that it is "foolproof" and only requires evaluations of F. On the other hand the convergence speed is not satisfactory, it converges merely linear: $|x^{(k)} - x^*| \leq 2^{-k}|b-a|$ and hence a total of $\log_2\left(\frac{|b-a|}{\texttt{tol}}\right)$ is necessary.

Remark 3.15.3: The MATLAB function `fzero` is based on bisection approach.

3.3.2 Model function methods

A whole class of iterative methods is given by the following idea: Instead of finding zeros of F, the function F is replaced by a *model function* $\tilde{F}$ based on function values/derivative values in $x^{(k)}, \ldots, x^{(k-m)}$. The next approximation of x^* is then given by $x^{(k+1)} :=$ zero of $\tilde{F}$. Again one needs to find a zero of a function, but the model function $\tilde{F}$ is chosen such that finding zeros there is easy, e.g. if there is an analytic formula for the zero.

3.3.2.1 Newton method in scalar case

For the Newton method one chooses the tangent of F in $x^{(k)}$ as model function, which leads the the *Newton iteration*:

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

Obviously this requires $F'(x^{(k)}) \neq 0$. If $F'(x^*) \neq 0$, the Newton method converges locally quadratically.

```
function [x, i] = scalar_newton(x0,F,dF,tol)
    x = x0;

    MAXIT = 1000;
    for i=1:MAXIT
        x = x-F(x) / dF(x);
        if (abs(s) < tol*abs(x)) return; end
    end

end
```

3.3.2.2 Special one-point methods

Instead of the tangent, we could also use better model function, i.e. non-linear local approximation. This requires the smoothness of F: $F \in C^m(I)$ for some $m > 1$ to find the free parameters of the model function h.

Definition 3.16 (Haley's iteration)
The model function h is given by

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

with free parameters a, b and c. The three conditions on F, F' and F'' give a system of equations, which leads to the following fixed point iteration:

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

Remark 3.16.1: Note that the model function of Haley's iteration is a *rational function* and therefore is especially useful for rational F.

3.3.2.3 Adapted Newton method

For functions F which are highly non-linear, the Newton method usually performs not as good as one wishes. The convergence however can often be boosted by converting the non-linear equation to an equivalent one (i.e. one that has the same solutions) for another function g, which is closer to a linear function.
One way to do this is the following: If we can find $\hat{F} \approx F$, where $\hat{F}$ is invertible with an inverse $\hat{F}^{-1}$ that can be evaluated with little effort, we can use

$$g(x) = \hat{F}^{-1}(F(x)) \quad \approx x$$

Now we can use Newton's method for $g(x) - \hat{F}^{-1}(0) = 0$, with the following derivative:

$$\frac{\mathrm{d}}{\mathrm{d}y}\left(\hat{F}^{-1}\right)(y) = \frac{1}{\hat{F}'(\hat{F}^{-1}(y))} \quad \Longrightarrow \quad g'(x) = \frac{1}{\hat{F}'(g(x))} \cdot F'(x)$$

3.3.2.4 Multi-point methods

One idea is to replace F with an *interpolation polynomial*, producing interpolatory model functions methods. Clearly the simplest methods of this class is the secant method, described next.

Definition 3.17 (Secant method)
The *secant method* uses the two last iterates $x^{(k)}$ and $x^{(k-1)}$ to compute $x^{(k+1)}$ (therefore being a two-point-method). $x^{(k+1)}$ is then the zero of the secant between the two previous points.

$$x^{(k+1)} = x^{(k)} - \frac{F\left(x^{(k)}\right)\left(x^{(k)} - x^{(k-1)}\right)}{F\left(x^{(k)}\right) - F\left(x^{(k-1)}\right)}$$

Remark 3.17.1: Note that the secant method only needs evaluations of F, no derivatives are required.

Remark 3.17.2: The convergence order of the secant method is $p = \frac{1}{2}(1+\sqrt{5}) \approx 1.62$.

```
function x = secant(x0,x1,F,tol)

    fo = F(x0);
    MAXIT = 1000;
    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

end
```

A second idea which leads to a yet another class of multi-point methods is *inverse interpolation*. We assume that $F : I \subset \mathbb{R} \to \mathbb{R}$ is a one-to-one mapping, and use the following observation:

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

We interpolate F^{-1} by a polynomial p of degree d determined by

$$p(F(x^{(k-m)})) = x^{(k-m)}, \quad m = 0, \ldots, d$$

The new approximate zero $x^{(k+1)}$ is then given by $p(0)$.

3.3.3 Note on efficiency

The efficiency of an iterative method for solving $F(x) = 0$ is the computational effort to reach a prescribed number of significant digits in the results. We are interested how much resources we need to spend to achieve a relative reduction of the error

$$\left\|e^{(k)}\right\| \leq \rho \left\|e^{(0)}\right\| \quad \text{with prescribed } 0 < \rho \ll 1$$

To that end, we look at the number of steps $k = k(\rho)$ and W, the computational effort per step. This effort W could look like

$$W \approx \frac{\#\{\text{evaluations of } F\}}{\text{step}} + \frac{\#\{\text{evaluations of } F'\}}{\text{step}} + \ldots$$

Note that $|\log \rho|$ is the number of significant digits or $x^{(k)}$. We new define the efficiency E as

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

It is now possible to use an asymptotic analysis with regard to $\rho \to 0$, which results in

$$E_{|\rho \to 0} = \begin{cases} -\frac{\log C}{W} & \text{if } p = 1 \\ \frac{\log p \cdot |\log \rho|}{W \log |\log \rho|} & \text{if } p > 1 \end{cases} \qquad \text{for any iterative method with convergence order } p$$

3.4 Newton's method

In *3.3.2.1 Newton method in scalar case*, we already have studied Newton's method for scalar functions. We now want to generalize that method to work with non-linear systems of equations. Given a function $F : D \subset \mathbb{R}^n \to \mathbb{R}^n$ we want to find $x^* \in D$ such that $F(x^*) = 0$, and we assume that F is continuously differentiable.
Again we use an affine linear model function that approximates F locally around $x^{(k)}$ to find $x^{(k+1)}$:

$$F(x) \approx \tilde{F}(x) := F(x^{(k)}) + \mathrm{D}F(x^{(k)})(x - (x^{(k)})) \qquad \text{with the Jacobian} \quad \mathrm{D}F(x) = \left(\frac{\partial F_j}{\partial x_k}(x)\right)_{j,k=1}^{n}$$

Definition 3.18 (Newton iteration)
The *Newton iteration* to solve $F(x) = 0$ is now defined as

$$x^{(k+1)} := x^{(k)} - \mathrm{D}F(x^{(k)})^{-1}F(x^{(k)}) \qquad \text{if } \mathrm{D}F(x^{(k)}) \text{ regular}$$

whereby the term $\mathrm{D}F(x^{(k)})^{-1}F(x^{(k)})$ is called the *Newton correction.*

Remark 3.18.1 (Affine invariance): The newton iteration is affine invariant. That is, for any regular $A \in \mathbb{R}^{n,n}$ and $G(x) = AF(x)$ it holds

$$F(x^*) = 0 \quad \Longleftrightarrow \quad G(x^*) = 0$$

This can be used as guideline for convergence theory (assumptions and results should be affine invariant, too), and modifying/extending Newton's method (resulting schemes should preserver affine invariance).

```
function [x, i] = newton(x,F,DF,tol)

    MAXIT = 1000;
    for i=1:MAXIT
        s = DF(x) \ F(x) % newton correction
        x = x-s;
        if (norm(s) < tol*norm(x)) return; end
    end

end
```

3.4.1 Numerical differentiation

Obviously, if $\mathrm{D}F(x)$ is not available for the Newton iteration, one can use *numerical differentiation* instead:

$$\frac{\partial F_i}{\partial x_j}(x) \approx \frac{F_i(x + h \cdot \vec{e}_j) - F_i(x)}{h}$$

But it is easy to get wrong results for very small h, due to round-off errors like cancellation.

3.4.2 Convergence of Newton's method

The Newton fixed point iteration is defined as $\Phi(x) = x - \mathrm{D}F(x)^{-1}F(x)$, and with the product rule we get:

$$\mathrm{D}\Phi(x) = I - \mathrm{D}(\mathrm{D}(x)^{-1})F(x) - \mathrm{D}F(x)^{-1}\mathrm{D}F(x)$$

which implies

$$F(x^*) = 0 \quad \Longrightarrow \quad \mathrm{D}\Phi(x^*) = 0$$

Because of this, Lemma 3.15 suggest the conjecture that Newton's method has local quadratic convergence, if $\mathrm{D}F(x^*)$ is regular. There is a sophisticated theory about the convergence of this method, and one result of this theory is the following theorem.

Theorem 3.19 (Local quadratic convergence of Newton's method)
If all of the following conditions are met,

1. $D \subset \mathbb{R}^n$ open and convex,
2. $F : D \to \mathbb{R}^n$ continuously differentiable,
3. $\mathrm{D}F(x)$ regular $\forall x \in D$,
4. $\exists L \geq 0 : \left\|\mathrm{D}F(x)^{-1}(\mathrm{D}F(x+v) - \mathrm{D}F(x))\right\|_2 \leq L \left\|v\right\|_2$ for all $v \in \mathbb{R}^n, v + x \in D$ and all $x \in D$,
5. $\exists x^* : \ F(x^*) = 0$,
6. initial guess $x^{(0)} \in D$ satisfies $\rho := \left\|x^* - x^{(0)}\right\|_2 < \frac{2}{L} \wedge B_\rho(x^*) \subset D$.

then the Newton iteration satisfies:

1. $x^{(k)} \in B_\rho(x^*) := \{y \in \mathbb{R}^n \mid \|y - x^*\| < \rho\}$ for all $k \in \mathbb{N}$,
2. $\lim_{k\to\infty} x^{(k)} = x^*$,
3. $\left\|x^{(k+1)} - x^*\right\|_2 \leq \frac{L}{2}\left\|x^{(k)} - x^*\right\|_2^2$ (local *quadratic convergence).*

3.4.3 Termination of Newton iteration

We expect that $\left\|x^{(k+1)} - x^*\right\| \ll \left\|x^{(k)} - x^*\right\|$ and therefore $\left\|x^{(k)} - x^*\right\| \approx \left\|x^{(k+1)} - x^{(k)}\right\|$. This gives us a first viable a posteriori termination criterion, namely by stopping as soon as

$$\left\|x^{(k+1)} - x^{(k)}\right\| = \left\|\mathrm{D}F(x^{(k)})^{-1}F(x^{(k)})\right\| < \tau \left\|x^{(k)}\right\| \qquad \text{for some prescribed tolerance } \tau$$

This criterion is unfortunate, because we compute the Newton correction one time too much. And since n can be quite large, the costs for this extra computation might be huge. We would like to use an a posteriori termination criterion that dispenses with computing (and "inverting") another Jacobian $\mathrm{D}F(x^{(k)})$ just to tell us that $x^{(k)}$ is already accurate enough.
Again we approximate: $\mathrm{D}F(x^{(k-1)}) \approx \mathrm{D}F(x^{(k)})$, so we quit as soon as

$$\left\|\mathrm{D}F(x^{(k-1)})^{-1}F(x^{(k)})\right\| < \tau \left\|x^{(k)}\right\| \qquad \text{for some prescribed tolerance } \tau$$

Now we can use $\mathrm{D}F(x^{(k)})$ twice, once to compute the newton correction, and again for the termination criterion. Note that both times we need to invert this matrix, so we can compute the LU-factorisation, and reuse the factors, resulting in $\mathcal{O}(n^2)$ operations rather then $\mathcal{O}(n^3)$.
Note that this is an affine invariant termination criterion, meaning that the algorithm is invariant to multiplication with any regular matrix A. If we used a residual based termination criterion, $\left\|F(x^{(k)})\right\| \leq \tau$, the resulting method would not be affine invariant, because for $F(x) = 0$ and $AF(x) = 0$ with regular $A \in \mathbb{R}^{n,n}$ the Newton iteration might terminate with different iterates.

3.4.4 Summary of the Newton Method

The Newton method converges asymptotically very fast, the number of significant digits doubles in every step. On the other hand, the region of convergence is often very small, requiring an initial guess rather close to the solution.

3.4.5 Damped Newton method

In practise, one often observes some kind of "overshooting", meaning that the Newton correction is much too big. This motivates the damping of the Newton correction by a damping factor $\lambda^{(k)}$.
The damping factor is chosen by an affine invariant natural monotonicity test:

$$\text{maximal } \lambda^{(k)} > 0 : \left\|\Delta\overline{x}(\lambda^{(k)})\right\| \leq \left(1 - \frac{\lambda^{(k)}}{2}\right) \left\|\Delta x^{(k)}\right\|$$

where

$$\begin{aligned} \Delta x^{(k)} &:= \mathrm{D}F(x^{(k)})^{-1}F\left(x^{(k)}\right) && \text{current Newton correction} \\ \Delta\overline{x}(\lambda^{(k)}) &:= \mathrm{D}F(x^{(k)})^{-1}F\left(x^{(k)} + \lambda^{(k)}\Delta x^{(k)}\right) && \text{tentative Newton correction} \end{aligned}$$

```
function [x,cvg] = dampnewton(x, F, DF, tol)

    LMIN = 0.001;
    [L,U] = lu(DF(x)); s = U \ (L \ F(x)) ;
    xn = x-s; lambda = 1; cvg = 0;
    f = F(xn) ; st = U \ (L \ f) ;

    % a posteriori termination based on simplified Newton correction
    while (norm(st) > tol*norm(xn))
        % natural monocity test
        while (norm(st) > (1-lambda/2)*norm(s))
            lambda = lambda / 2; % reduce damping
            if (lambda < LMIN), cvg = -1; return; end % failed
            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); % remove damping
        xn = x-lambda*s; f = F(xn); st = U \ (L \ f) ;
    end
    x = xn;

end
```

4 Krylov methods for linear systems of equations

In cases where the system matrix is very large (and possibly sparse), or only given as procedure `y=evalA(x)`, direct methods fail. This is where iterative methods, such as Krylov methods are important.

4.1 Descent methods

In this section, the focus lies on linear systems $Ax = b$ with symmetric positive definite system matrices A. Remember that such a matrix defines an inner product by $(x, y) \to x^T Ay$.

Definition 4.1 (Energy norm)
A s.p.d. matrix $A \in \mathbb{R}^{n,n}$ induces an *energy norm*

$$\|x\|_A := \left(x^T Ax\right)^{\frac{1}{2}}, \quad x \in \mathbb{R}^n$$

4.1.1 Quadratic minimization context

Lemma 4.2 (S.p.d. LSE and quadratic minimization problem)
A LSE with $A \in \mathbb{R}^{n,n}$ s.p.d. and $b \in \mathbb{R}^n$ is equivalent to the following minimization problem:

$$Ax = b \quad \Longleftrightarrow \quad x = \arg\min_{y \in \mathbb{R}^n} J(y), \quad J(y) := \tfrac{1}{2} y^T Ay - b^T y$$

The idea now is to solve the LSE iteratively by successive solutions of *simpler* minimisation problems.

4.1.2 Abstract steepest descent

Given a continuously differentiable $F : D \subset \mathbb{R}^n \to \mathbb{R}$, find a minimizer $x^* \in D : x^* = \arg\min_{x \in D} F(x)$. The most natural iteration is to follow the *steepest descent*, i.e. the inverse of the gradient, and search for the minimum in this direction (1D minimization: use Newton's method).

Initial guess $x^{(0)} \in D, k = 0$
repeat
  $d_k := -\operatorname{grad} F(x^{(k)})$
  $t^* := \arg\min_{t \in \mathbb{R}} F(x^{(k)} + t \cdot d_k)$
  $x^{(k+1)} := x^{(k)} + t^* \cdot d_k$
  $k := k + 1$
until $\left\|x^{(k)} - x^{(k-1)}\right\| \leq \tau \left\|x^{(k)}\right\|$

Obviously there are various problems with this approach, like getting stuck in local minima, convergence problems and the like, but for the quadratic minimization problem, this algorithm will converge.
For the search direction d_k, differentiation yields

$$\operatorname{grad} J(x) = Ax - b \quad \Longrightarrow \quad d_k = b - Ax^{(k)} =: r_k \quad \text{the residual}$$

The line search reduces to finding the minimum of a parabola, so there is a unique minimizer

$$t^* = \frac{d_k^T d_k}{d_k^T A d_k}$$

Note that if $d_k = 0$, the solution has been found.

```
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

end
```

One single iteration of this algorithm only requires one matrix-vector product and operations with costs of $\mathcal{O}(n)$.

4.1.3 Convergence of gradient method

With numerical experiments one can observe that methods converges linearly, and the rate of convergence increases (i.e. speed of convergence decreases) with the spread of the spectrum of A. Also successive residuals r_k and r_{k+1} are orthogonal.

Theorem 4.3 (Convergence of gradient method/steepest descent)
The iterates of the gradient method satisfy

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

that is, the iteration converges at least linearly with regard to the energy norm.

Remark 4.3.1: Note that if A is symmetric and regular, the condition number can be written as

$$\operatorname{cond}_2(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} \qquad \text{with} \qquad \begin{array}{l} \lambda_{\max}(A) := \max(|\sigma(A)|) \\ \lambda_{\min}(A) := \min(|\sigma(A)|) \end{array}$$

4.2 Conjugate gradient method (CG)

An obvious drawback of the gradient method, is that previous knowledge is not used, only the last point is considered. This is a typical situation with 1-point methods. Instead of a linear search, we now use a *subspace correction*. To do this, we look at nested subspaces $U_1 \subset U_2 \subset \cdots U_n = \mathbb{R}^n$ with $\dim U_k = k$. The iteration now is defined as

$$x^{(k)} := \underset{x \in U_k + x^{(0)}}{\arg\min}\ J(x)$$

Note that once U_k and $x^{(0)}$ are fixed, the iteration is well defined, because there is always a unique minimizer. Obviously we will find the exact solution in the end: $x^{(n)} = x^* = A^{-1}b$.
The idea to find such subspaces U_k is to use the previous subspace, and a "local steepest descent direction", given by $-\operatorname{grad} J(x^{(k)}) = b - Ax^{(k)} = r_k$:

$$U_{k+1} = \operatorname{span}\{U_k, r_k\}$$

Lemma 4.4 ($r_k \perp U_k$)
With $x^{(k)}$ like defined above, U_k and the residual $r_k = b - Ax^{(k)}$ satisfies

$$r_k^T u = 0 \quad \forall u \in U_k$$

Corollary 4.5
If $r_l \neq 0$ for $l = 0, \ldots, k$, $k \leq n$, then $\{r_0, \ldots, r_k\}$ is an *orthogonal basis* of U_k.

4.2.1 Krylov spaces

Definition 4.6 (Krylov space)
For $A \in \mathbb{R}^{n,n}$, $z \in \mathbb{R}^n$, $z \neq 0$, the l-th *Krylov space* is defined as

$$\mathcal{K}_l(A, z) := \operatorname{span}\left\{z, Az, \ldots, A^{l-1}z\right\}$$

Lemma 4.7
The subspaces $U_k \subset \mathbb{R}^n$, $k \geq 1$, defined above satisfy

$$U_k = \operatorname{span}\left\{r_0, Ar_0, \ldots, A^{k-1}r_0\right\} = \mathcal{K}_k(A, r_0)$$

where $r_0 = b - Ax^{(0)}$ is the initial residual.

4.2.2 Implementation of CG

```
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
end
```

4.2.3 Convergence of CG

Note that CG is a direct solve, because (in exact arithmetic) $x^{(k)} = x^*$ for some $k \leq n$. Numerical instability however makes it completely pointless to (try to) use CG as a direct solver. In practise, CG is used for large n as *iterative solver*, and $x^{(k)}$ for some $k \ll n$ is expected to provide a good approximation for x^*.

Corollary 4.8 ("Optimality of CG method")
Writing $x^* \in \mathbb{R}^n$ for the exact solution of $Ax = b$, the CG iterates satisfy

$$\left\| x - x^{(l)} \right\|_A = \min \left\{ \|y - x\|_A : y \in x^{(0)} + \mathcal{K}_l(A, r_0) \right\}, \quad r_0 = b - Ax^{(0)}$$

Theorem 4.9 (Convergence of CG method)
The iterates of the CG method for solving $Ax = b$ with $A = A^T$ s.p.d. satisfy

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

where $\kappa(A) = \text{cond}_2(A)$ is the spectral condition number.

Remark 4.9.1: The estimate of this theorem confirms *asymptotic linear convergence* with a rate of $\frac{\sqrt{\kappa(A)-1}}{\sqrt{\kappa(A)+1}}$.

4.3 Preconditioning

The CG methods potentially converges only slowly if $\kappa(A) \gg 1$. The idea of precondition is to transform the system $Ax = b$ to $\tilde{A}\tilde{x} = \tilde{b}$ with $\tilde{A} = B^{-\frac{1}{2}}AB^{\frac{1}{2}}$, $\tilde{x} = B^{\frac{1}{2}}x$ and $\tilde{b} = B^{-\frac{1}{2}}b$ for some *preconditioner* B (s.p.d.).
$B^{\frac{1}{2}}$ is defined as $B^{\frac{1}{2}} = Q \cdot \text{diag}(\sqrt{d_1}, \ldots, \sqrt{d_n}) \cdot Q^T$ with $B = QDQ^T$.

Definition 4.10
A s.p.d. matrix $B \in \mathbb{R}^{n,n}$ is called *preconditioner* for the s.p.d. matrix $A \in \mathbb{R}^{n,n}$, if

1. $\kappa(B^{-\frac{1}{2}}AB^{\frac{1}{2}})$ is "small" and,
2. the evaluation of $B^{-1}x$ is about as expensive (in terms of elementary operations) as the matrix-vector multiplication Ax with $x \in \mathbb{R}^n$.

If one formally applies the CG method to the transformed system, it becomes apparent that, after suitable transformations, $B^{\frac{1}{2}}$ and its inverse never actually are required, only $B^{-1}x$ is needed.
There are several posibilities to choose a preconditioner, including

1. $B = \text{diag}(A)$, Jacobi preconditioner,
2. symmetric Gauss-Seidel preconditioner,
3. incomplete Cholesky factorisation (MATLAB: `ichol`),
4. sparse approximate inverse preconditioner (SPAI)

4.4 Survey of Krylov subspace methods

Advantages of Krylov methods versus direct elimination (**if** they converge at all/sufficiently fast):

- They require system matrix only in procedural form `y=evalA(x)`.
- They can perfectly exploit sparsity of the system matrix.
- They can cash in on low accuracy requirements.
- They can benefit from good initial guesses.

5 Eigenvalues

5.1 Theory

Definition 5.1 (Eigenvalues and eigenvectors)
For any matrix $A \in \mathbb{K}^{n,n}$

- $\lambda \in \mathbb{C}$ is called *eigenvalue* of A if and only if $\chi(\lambda) := \det(\lambda \cdot I - A) = 0$, where $\chi(\lambda)$ is called the characteristic polynomial.
- $\sigma(A) := \{\lambda \in \mathbb{C} \mid \lambda \text{ is eigenvalue of } A\}$ is called the *spectrum* of A.
- The *eigenspace* associated with an eigenvalue $\lambda \in \sigma(A)$ is defined as
$$\mathrm{Eig}_A(\lambda) := \mathrm{Ker}(\lambda \cdot I - A)$$
- If $x \in \mathrm{Eig}_A(\lambda) \setminus \{0\}$, x is called *eigenvector*.
- The geometric multiplicity of an eigenvalue $\lambda \in \sigma(A)$ is defined as
$$m(\lambda) := \dim \mathrm{Eig}_A(\lambda)$$

Remark 5.1.1: Note that $\det(A) = \det(A^T)$ and thus $\sigma(A) = \sigma(A^T)$.

Remark 5.1.2: The *spectral radius* is defined as $\rho(A) := \max\{|\lambda| \mid \lambda \in \sigma(A)\}$.

Theorem 5.2 (Bound for the spectral radius)
For any matrix norm $\|\cdot\|$ induced by a vector norm it holds true
$$\rho(A) \leq \|A\|$$

Lemma 5.3 (Gershgorian circle theorem)
For any matrix $A \in \mathbb{K}^{n,n}$ holds true
$$\sigma(A) \subset \bigcup_{j=1}^{n} \left\{ z \in \mathbb{C} : |z - a_{jj}| \leq \sum_{i \neq j} |a_{ji}| \right\}$$

Lemma 5.4
Similar matrices $A, B \in \mathbb{K}^{n,n}$ share the same spectrum. If A is similar to B, then there exists a regular matrix $S \in \mathbb{K}^{n,n}$ with $B = S^{-1}AS$.

Corollary 5.5
If $A \in \mathbb{K}^{n,n}$, $AA^H = A^HA$: $\exists U \in \mathbb{C}^{n,n}$ unitary: $U^HAU = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, $\lambda_i \in \mathbb{C}$.

Remark 5.5.1: A matrix $A \in \mathbb{K}^{n,n}$ with $AA^H = A^HA$ is called *normal*. Examples of normal matrices include Hermitian matrices, unitary matrices and skew-Hermitian matrices ($A = -A^H$).

Definition 5.6 (Linear generalized eigenvalue problem)
The linear generalized eigenvalue problem is for given $A \in \mathbb{C}^{n,n}$, regular $B \in \mathbb{C}^{n,n}$ to find $x \neq 0$ and $\lambda \in \mathbb{C}$ such that
$$Ax = \lambda Bx \quad \Longleftrightarrow \quad B^{-1}Ax = \lambda x$$

5.2 "Direct" eigensolvers

These "direct" eigensolvers are used to find the solution of an eigenvalue problem for *dense* matrices "up to machine precision". In MATLAB this can be done with `eig`.
Library implementation provide numerically stable eigensolvers!

5.3 Power methods

To find the largest (in modulus) eigenvalue of A and (an) associated eigenvector, a first simple approach is the *direct power method*:

---

1 $z^{(0)}$:= "arbitrary" initial guess
2 **repeat**
3 $\quad w := Az^{(k-1)}$
4 $\quad z^{(k)} := \frac{w}{\|w\|_2}$
5 $\quad k := k + 1$
6 **until** "convergence"

Definition 5.7 (Rayleigh quotient)
For any matrix $A \in \mathbb{K}^{n,n}$ and $u \in \mathbb{K}^n$, the *Rayleigh quotient* is defined by

$$\rho_A(u) := \frac{u^H A u}{u^H u}$$

Remark 5.7.1: An immediate consequence of the definition is

$$\lambda \in \sigma(A), z \in \mathrm{Eig}_\lambda(A) \quad \Longrightarrow \quad \rho_A(z) = \lambda$$

Theorem 5.8 (Convergence of direct power method)
Let $\lambda_n > 0$ be the largest (in modulus) eigenvalue of $A \in \mathbb{K}^{n,n}$ and have (algebraic) multiplicity 1. Let v, y be left and right eigenvectors of A for λ_n normalized according to $\|y\|_2 = \|u\|_2 = 1$. Then there is convergence

$$\left\|Az^{(k)}\right\|_2 \to \lambda_n, \quad z^{(k)} \to \pm v, \quad \textit{linearly with rate} \quad \frac{\lambda_{n-1}}{\lambda_n}$$

where $z^{(k)}$ are the iterates of the direct power iteration and $y^H z^{(0)} \neq 0$ is assumed.

Remark 5.8.1: Round-off errors normally guarantee $y^H z^{(0)} \neq 0$ in practical computations.

Remark 5.8.2: A usual (not the best however) choice for $z^{(0)}$ is a random vector.

Remark 5.8.3: A termination criterion can an adapted version of the a posteriori termination criterion be used: "relative change" $\leq$ `tol`.

$$\left\|z^{(k)} - z^{(k-1)}\right\| \leq \left(\tfrac{1}{L} - 1\right) \cdot \texttt{tol} \quad \Longleftrightarrow \quad \left| \frac{\left\|Az^{(k)}\right\|}{\left\|z^{(k)}\right\|} - \frac{\left\|Az^{(k-1)}\right\|}{\left\|z^{(k-1)}\right\|} \right| \leq \left(\tfrac{1}{L} - 1\right) \cdot \texttt{tol}$$

5.3.1 Image segmentation

Given a grey-scale image with an intensity matrix $P \in \{0, \ldots, 255\}^{m,n}$, the task of *image segmentation* is to find connected patches of the image with the same shade/color.
To make our lives easier, we define $p_k = (P)_{ij}$ where $k = (i-1)n + j$. Also, we need a similarity function $\sigma(x, y)$, that measures the "similarity" of two neighboring pixels. If $\sigma(x, y)$ is large, the colors x and y are very alike. With this, we can define a *local similarity matrix* $W \in \mathbb{R}^{N,N}$ with $N = mn$:

$$(W)_{ij} = \begin{cases} 0 & \text{if pixels } i \text{ and } j \text{ are not adjacent} \\ 0 & \text{if } i = j \\ \sigma(p_i, p_j) & \text{if pixels } i \text{ and } j \text{ are adjacent} \end{cases}$$

Definition 5.9 (Normalized cut)
For $\mathcal{X} \subset \mathcal{V}$ the *normalized cut* is defined as

$$\mathrm{Ncut}(\mathcal{X}) = \frac{\mathrm{cut}(\mathcal{X})}{\mathrm{weight}(\mathcal{X})} + \frac{\mathrm{cut}(\mathcal{X})}{\mathrm{weight}(\mathcal{X} \setminus \mathcal{V})}$$

with

$$\mathrm{cut}(\mathcal{X}) = \sum_{\substack{i \in \mathcal{X} \\ j \notin \mathcal{X}}} w_{ij} \qquad \mathrm{weight}(\mathcal{X}) = \sum_{\substack{i \in \mathcal{X} \\ j \in \mathcal{V}}} w_{ij}$$

The segmentation can now be formulated as follows: Find $\mathcal{X}^* \subset \mathcal{V}$ such that $\mathcal{X}^* = \arg\min_{\mathcal{X} \subset \mathcal{V}} \mathrm{Ncut}(\mathcal{X})$.
To simplify the problem, we first introduce an indicator function $z : \{1, \ldots, N\} \to \{-1, 1\}$ with

$$z_i = z(i) = \begin{cases} 1 & \text{if } i \in \mathcal{X} \\ -1 & \text{else} \end{cases}$$

This leads to the following formulation

$$\mathrm{Ncut}(\mathcal{X}) = \frac{\sum\limits_{z_i > 0, z_j < 0} -w_{ij} z_i z_j}{\sum\limits_{z_i > 0} d_i} + \frac{\sum\limits_{z_i > 0, z_j < 0} -w_{ij} z_i z_j}{\sum\limits_{z_i < 0} d_i}$$

with

$$d_i = \sum_{j \in \mathcal{V} \setminus \{i\}} w_{ij} = \mathrm{weight}(\{i\})$$

From this, we deduce the following sparse matrices: $D = \mathrm{diag}(d_1, \ldots, d_N) \in \mathbb{R}^{N,N}$, $A = D - W = A^T$ with $1^T A = A1 = 0$.
With these matrices, we can write the normalized cut as follows

Lemma 5.10 (Ncut and Rayleigh quotient)
The normalized cut can be written as generalized Rayleigh quotient

$$\mathrm{Ncut}(\mathcal{X}) = \frac{y^T A y}{y^T D y}, \qquad y = (1+z) - \beta(1-z), \qquad \beta = \frac{\sum_{z_i>0} d_i}{\sum_{z_i<0} d_i}$$

With this, we can rewrite the segmentation problem as

$$\underset{y \in \{2,-2\beta\}^N, 1^T D y = 0}{\arg\min} \rho_{A,D}(y)$$

and by relaxation, we transform the problem further to a continuous optimization problem:

$$\underset{y \in \mathbb{R}^N, \|y\|_D = 1, 1^T D y = 0}{\arg\min} \rho_{A,D}(y)$$

The constraints $\|y\|_D = 1, 1^T D y = 0$ compound the difficulties of solving this problem. However, the next theorem establishes a link to a generalized eigenvalue problem.

Theorem 5.11 (Courant-Fischer min-max theorem)
Let $\lambda_1 < \lambda_2 < \cdots < \lambda_m$, $m \leq n$ be the sorted sequence of the (real) eigenvalues of $A = A^H \in \mathbb{C}^{n,n}$. Write

$$U_0 = \{0\}, \quad U_l = \sum_{j=1}^{l} \mathrm{Eig}_A(\lambda_j), \quad l = 1, \dots m \qquad \text{and} \qquad U_l^\perp = \{x \in \mathbb{C}^n : u^T x = 0 \quad \forall u \in U_l\}$$

Then

$$\min_{y \in U_{l-1}^\perp \setminus \{0\}} \rho_A(y) = \lambda_l, \quad 1 \leq l \leq m \qquad \underset{y \in U_{l-1}^\perp \setminus \{0\}}{\arg\min} \rho_A(y) \subset \mathrm{Eig}_A(\lambda_l)$$

With this, we can give the outline of an algorithm to solve the image segmentation problem:

1. Given a similarity function σ compute the (sparse) matrices $W, D, A \in \mathbb{R}^{N,N}$.
2. Compute x^* with $\|x^*\|_2 = 1$ as eigenvector for the 2nd smallest eigenvalue of the generalized eigenvalue problem $Ax = \lambda D x$.
3. Define the image segment as pixel set:

$$\mathcal{X} = \{i \in \{1, \dots, N\} : x_i^* > \frac{1}{N} \sum_{i=1}^{N} x_i^*\}$$

5.3.2 Inverse iteration

To find the smallest (in modulus) eigenvalue of a regular matrix $A \in \mathbb{K}^{n,n}$ and an associated eigenvector, one can use the following fact:

$$\text{Smallest EV of } A = \left(\text{Largest EV of } A^{-1}\right)^{-1}$$

Thus, it is possible to apply the direct power method to A^{-1}, which directly yields the inverse iteration.

```
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
end
```

Note that instead of computing `x = A \ y` (or even compute A^{-1}), the LU factorisation is reused.

5.3.2.1 Shifted inverse iteration

The inverse power iteration converges linearly with rate $\left|\frac{\lambda_1(A)}{\lambda_2(A)}\right|$. To speed up the convergence rate, we shift the equation

$$x^{(k+1)} = (A - \sigma I)^{-1} x^{(k)}$$

This shift does not change the eigenvectors, but shifts the eigenvalues. Thus, the new convergence rate is $\left|\frac{\lambda_1(A)-\sigma}{\lambda_2(A)-\sigma}\right|$. We try to have $\sigma \approx \lambda_1(A)$. Note, that the resulting linear system of equations may be very badly conditioned, and thus the results may be highly inaccurate. However, only the length of $x^{(k+1)}$ is affected, not its direction.
We adapt the shift in every step and use the approximation of $\lambda_1(A)$ of the previous step. However, this means that we cannot reuse the LU factorisation any longer, but rather have to fully solve the system of equations in every step.

```
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
end
```

5.3.3 Subspace iterations

To compute m, $m \ll n$ of the largest/smallest (in modulus) eigenvalues of $A = A^H \in \mathbb{C}^{n,n}$ and the m associated eigenvectors, we now introduce subspace iterations.
If we just carry out the direct power iteration for two vectors, both sequences will converge to the largest (in modulus) eigenvector. However, all eigenvectors are mutually orthogonal. This suggests, that we orthogonalize the iterates of the second power iteration with respect to those of the first.

5.4 Singular value decomposition

Theorem 5.12 (Singular value decomposition)
For any $A \in \mathbb{K}^{n,n}$ there are unitary matrices $U \in \mathbb{K}^{m,m}$, $V \in \mathbb{K}^{n,n}$ and a (generalized) diagonal matrix $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m,n}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ such that

$$A = U\Sigma V^H$$

This decomposition is called the *singular value decomposition (SVD)* of A. The diagonal entries σ_i of Σ are the *singular values* of A.

Remark 5.12.1: The singular value decomposition is not (necessarily) unique, but the singular values are.

Remark 5.12.2: In MATLAB there is `[U,S,V] = svd(A)` to compute the singular value decomposition. Note that the algorithm is numerically stable.

Lemma 5.13 (Singular values and eigenvalues)
The squares σ_i^2 of the non-zero singular values of A are the non-zero eigenvalues of $A^H A$, AA^H with associated eigenvectors $(V)_{:,1}, \ldots, (V)_{:,p}$ and $(U)_{:,1}, \ldots, (U)_{:,p}$ respectively.

Remark 5.13.1 (SVD and additive rank-1 decomposition): The singular value decomposition provides an additive decomposition into rank-1 matrices:

$$A = U\Sigma V^H = \sum_{j=1}^{p} \sigma_i (U)_{:,j} (V)_{:,j}^H$$

This can be used for a low rank approximation of any matrix A. In fact, this is the best rank-k approximation.

Lemma 5.14 (SVD and the rank of a matrix)
If the singular values of A satisfy $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0$, then

- $\operatorname{rank}(A) = r$
- $\operatorname{Ker}(A) = \operatorname{span}\{(V)_{:,r+1}, \ldots, (V)_{:,n}\}$
- $\operatorname{Im}(A) = \operatorname{span}\{(U)_{:,1}, \ldots, (U)_{:,r}\}$

Lemma 5.15 (SVD and Euclidean matrix norm)
It holds true

- $\forall A \in \mathbb{K}^{m,n}: \quad \|A\|_2 = \sigma_1(A)$
- $\forall A \in \mathbb{K}^{n,n}$ regular: $\quad \operatorname{cond}_2(A) = \frac{\sigma_1}{\sigma_n}$

Lemma 5.16 (Frobenius norm)
The *Frobenius norm* of $A \in \mathbb{K}^{m,n}$ is defined as

$$\|A\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2$$

With the singular value decomposition, it holds

$$\|A\|_F^2 = \sum_{j=1}^{p} \sigma_j^2$$

6 Least squares

Definition 6.1 ((Linear) least squares problem)
Given $A \in \mathbb{K}^{n,n}$, $m, n \in \mathbb{N}$ and $b \in \mathbb{K}^m$, find $x \in \mathbb{K}^n$ such that

1. $\|Ax - b\|_2 = \inf\{\|Ay - b\|_2 : y \in \mathbb{K}^n\}$
2. $\|x\|_2$ is minimal under condition 1.

Lemma 6.2 (Existence and uniqueness of solution)
The least squares problem for $A \in \mathbb{K}^{m,n}$, $A \neq 0$, has a unique solution for every $b \in \mathbb{K}^m$.

The solution operator of the least squares problem defines a linear mapping $b \to x$, which has a matrix representation.

Definition 6.3 (Pseudoinverse)
The *pseudoinverse* $A^+ \in \mathbb{K}^{n,m}$ of $A \in \mathbb{K}^{m,n}$ is the matrix representation of the (linear) solution operator $\mathbb{R}^m \to \mathbb{R}^n$, $b \to x$ of the least squares problem.

Remark 6.3.1: In MATLAB the command `pinv(A)` computes the pseudoinverse.

Definition 6.4 (Generalized condition (number) of a matrix)
Let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_p = 0$, $p = \min\{m, n\}$, be the singular valus of $A \in \mathbb{K}^{m,n}$. Then

$$\operatorname{cond}_2(A) := \frac{\sigma_1}{\sigma_r}$$

is the *generalized condition (number)* (with regard to the 2-norm) of A.

Theorem 6.5
For $m \geq n$, $A \in \mathbb{K}^{m,n}$, $\operatorname{rank}(A) = n$, let $x \in \mathbb{K}^n$ be the solution of the least squares problem $\|Ax - b\| \to \min$ and $\hat{x}$ the solution of the perturbed least squares problem $\|(A + \Delta A)\hat{x} - b\| \to \min$. Then

$$\frac{\|x - \hat{x}\|_2}{\|x\|_2} \leq \left(2\operatorname{cond}_2(A) + \operatorname{cond}_2^2(A)\frac{\|r\|_2}{\|A\|_2\|x\|_2}\right)\frac{\|\Delta A\|_2}{\|A\|_2}$$

holds, where $r = Ax - b$ is the *residual*.

Remark 6.5.1: This means if $\|r\|_2 \ll 1$, the condition of the least squares problem is $\approx \operatorname{cond}_2(A)$, and if $\|r\|_2$ is "large", the condition is $\approx \operatorname{cond}_2^2(A)$.

6.1 Normal Equations

One can consider the function $f(x) = \|Ax - b\|_2^2$ and minimize it. This leads to the normal equations

$$\operatorname{grad}(f(x)) = 2A^HAx - 2A^Hb \stackrel{!}{=} 0$$

Definition 6.6 (Normal equation)
To find the solution to the least squares problem it is equivalent to solve the normal equation:

$$A^HAx = A^Hb$$

Remark 6.6.1: Solving the normal equation is numerically instable. The condition of A^HA is $(\operatorname{cond}_2(A))^2$. Additionally one cannot expect A^HA to be sparse if A is sparse.

It is however possible to avoid the computation of A^HA by expanding the normal equations and introduce the residual $r = Ax - b$ as a new unknown:

$$A^HAx = A^Hb \quad \Longleftrightarrow \quad B\begin{pmatrix} r \\ x \end{pmatrix} := \begin{pmatrix} -I & A \\ A^H & 0 \end{pmatrix}\begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

For $m, n \gg 1$ and A sparse, this leads to large sparse linear systems of equations.

6.2 Orthogonal transformation methods

The idea is to use orthogonal (unitary) transformations, which leave the 2-norm invariant, to transform $Ax - b$ to a simpler form. As with LSE, "simpler form" means triangular form.
If A has full rank (i.e. $\operatorname{rank}(A) = n$), one can use the QR-decomposition:

$$\|Ax - b\|_2 = \left\|Q(Rx - Q^Hb)\right\|_2 = \left\|Rx - \tilde{b}\right\|_2, \quad \tilde{b} = Q^Hb$$

with $Q \in \mathbb{K}^{m,m}$, $R \in \mathbb{K}^{m,n}$. The solution x and the residual r are then given by

$$x = \hat{R}^{-1} \begin{pmatrix} \tilde{b}_1 \\ \vdots \\ \tilde{b}_n \end{pmatrix} \qquad r = Q \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \tilde{b}_{n+1} \\ \vdots \\ \tilde{b}_m \end{pmatrix}$$

where $\hat{R} \in \mathbb{K}^{n,n}$ is the triangular part of R.
If A doesn't have full rank, it is possible to use the singular value decomposition:

$$\|Ax - b\|_2 = \left\| \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_1^H \\ V_2^H \end{pmatrix} x - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} \Sigma_r V_1^H x \\ 0 \end{pmatrix} - \begin{pmatrix} U_1^H b_1 \\ U_2^H b_2 \end{pmatrix} \right\|_2$$

It is obvious that one chooses x such that the first r components of $\begin{pmatrix} \Sigma_r V_1^H x \\ 0 \end{pmatrix} - \begin{pmatrix} U_1^H b_1 \\ U_2^H b_2 \end{pmatrix}$ vanish. This leads to an underdetermined linear system $\Sigma_r V_1^H x = U_1^H b_1$. To fix a unique solution we appeal to the minimal norm condition: The solution x is unique up to contributions from $\mathrm{Ker}(V_1) = \mathrm{Im}(V_2)$. Since V is orthogonal, the minimal norm solution is obtained by setting contributions from $\mathrm{Im}(V_2)$ to zero, with amounts to choosing $x \in \mathrm{Im}(V_1)$.
The solution x and the norm of the residual r are given by

$$x = \left(V_1 \Sigma_r^{-1} U_1^H\right) b_1 \qquad \|r\|_2 = \left\|U_2^H b_2\right\|_2$$

This result directly yields a representation of the pseudoinverse A^+ of any matrix A:

Theorem 6.7 (Pseudoinverse via SVD)
If $A \in \mathbb{K}^{m,n}$ has the singular value decomposition

$$A = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_1^H \\ V_2^H \end{pmatrix}$$

then $A^+ = V_1 \Sigma_r^{-1} U_1^H$ holds.

Remark 6.7.1: Orthogonal transformation methods have superior numerical stability, use them whenever $A \in \mathbb{K}^{m,n}$ dense and n small. On the other hand, these methods cannot exploit sparsity, thus use the normal equations in the expanded form, when $A \in \mathbb{K}^{m,n}$ sparse and m, n big.

6.3 Non-linear least squares

Definition 6.8 (Non-linear least squares problem)
Given $F : D \subset \mathbb{R}^n \to \mathbb{R}^m$, $m, n \in \mathbb{N}$, $m > n$, find $x^* \in D$ such that

$$x^* = \underset{x \in D}{\arg\min}\, \Phi(x), \quad \Phi(x) := \tfrac{1}{2} \|F(x)\|_2^2$$

Remark 6.8.1: Existence and uniqueness of x^* needs to be considered in each concrete case individually.

Remark 6.8.2: We require "independence for each parameter": $\exists$ neighbourhood $\mathcal{U}(x^*)$ such that $\mathrm{D}F(x)$ has full rank n for all $x \in \mathcal{U}(x^*)$. This means that the columns of the Jacobi matrix $\mathrm{D}F(x)$ are linearly independent.

6.3.1 (Damped) Newton method

We are searching a minimum of $\Phi(x)$, thus a simple idea is to use Newton's method to find a zero of $\operatorname{grad} \Phi(x)$:

$$x^{(k+1)} = x^{(k)} - H\Phi(x^{(k)})^{-1} \operatorname{grad} \Phi(x^{(k)}), \qquad \text{with the Hessian matrix } H\Phi(x)$$

Expressed in terms of F we get

$$\begin{aligned} \operatorname{grad} \Phi(x) &= \mathrm{D}F(x)^T F(x) \\ H\Phi(x) = \mathrm{D}(\operatorname{grad} \Phi)(x) &= \mathrm{D}F(x)^T \mathrm{D}F(x) + \sum_{j=1}^{m} F_j(x) \mathrm{d}^2 F_j(x) \\ (H\Phi(x))_{i,k} &= \sum_{j=1}^{n} \frac{\partial^2 F_j}{\partial x_i \partial x_k}(x) \cdot F_j(x) + \frac{\partial F_j}{\partial x_k}(x) \cdot \frac{\partial F_j}{\partial x_i}(x) \end{aligned}$$

6.3.2 Gauss-Newton method

Another idea is to use *local linearization* of F, $F(x) \approx F(y) + \mathrm{D}F(y)(x-y)$, and thereby get a sequence of linear least square problems.

$$\arg\min_{x\in\mathbb{R}^n} \|F(x)\|_2 \text{ is approximated by } \arg\min_{x\in\mathbb{R}^n} \|F(x_0) + \mathrm{D}F(x_0)(x-x_0)\|_2$$

where x_0 is an approximation of the solution x^*. Note that this is a linear least square problem of the form $\arg\min \|Ax-b\|_2$ with $A = \mathrm{D}F(x_0)$ and $b = F(x_0) - \mathrm{D}F(x_0)\cdot x_0$.

```
function x = gaussnewton(x, F , DF, tol)
    s = DF(x) \ F(x);
    x = x-s;
    while (norm(s) > tol*norm(x))
        s = DF(x) \ F(x);
        x = x-s;
    end
end
```

The Gauss-Newton method has the advantage (compared to Newton's method), that no second derivative of F is needed. On the other hand, there is no local quadratic convergence.

7 Filtering algorithms

In this chapter, we study *finite linear time-invariant causal channel (filter).*

- finite: the impulse response is of finite duration and hence can be described by a finite vector h of length n.
- time-invariant: when the input is shifted in time, the output is shifted by the same amount of time.
- linear: input $\rightarrow$ output is a linear mapping.
- causal (or physical): output depends only on past and present inputs, not on the future.

The impulse response $h = (h_0, \ldots, h_{n-1})^T$ is the output when the filter is fed with a single impulse of strength one, corresponding to input e_1. Given an input signal x, the output y can be computed as follows:

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j, \quad k = 0, \ldots, 2n-2 \quad (h_j := 0 \text{ for } j < 0 \text{ and } j \geq n)$$

Definition 7.1 (Discrete convolution)
Given $x = (x_0, \ldots, x_{n-1})^T \in \mathbb{K}^n$, $h = (h_0, \ldots, h_{n-1})^T \in \mathbb{K}^n$ their *discrete convolution* is the vector $y \in \mathbb{K}^{2n-1}$ with

$$y = h * x \qquad \Longleftrightarrow \qquad y_k = \sum_{j=0}^{n-1} h_{k-j} x_j, \quad k = 0, \ldots, 2n-2 \quad (h_j := 0 \text{ for } j < 0)$$

Remark 7.1.1: The discrete convolution is commutative.

This definition naturally expands to sequences $\mathbb{N}_0 \rightarrow \mathbb{K}$ and 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

$$z_k = \sum_{j=0}^{k} x_{k-j} y_j = \sum_{j=0}^{k} x_j y_{k-j}, \quad k \in \mathbb{N}_0$$

Whenever the input signal of a time-invariant filter is n-periodic, so will be the output signal. Thus, the n-periodic setting, a causal linear time-invariant filter will give a linear mapping $\mathbb{R}^n \rightarrow \mathbb{R}^n$ according to

$$y_k = \sum_{j=0}^{n-1} h_{k-j} x_j \quad \text{for some } h_0, \ldots, h_{n-1} \in \mathbb{R}$$

Note that h is no longer the impulse response of the filter.

Definition 7.2 (Discrete periodic convolution)
The *discrete periodic convolution* of two n-periodic sequences $(x_j)_{j \in \mathbb{N}_0}$, $(y_j)_{j \in \mathbb{N}_0}$ yields the n-periodic sequence

$$(z_k) = (x_k) *_n (y_k) \qquad \Longleftrightarrow \qquad z_k = \sum_{j=0}^{k} x_{k-j} y_j = \sum_{j=0}^{k} x_j y_{k-j}, \quad k \in \mathbb{Z}$$

Remark 7.2.1: 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: $z = x *_n y \in \mathbb{K}^n$ with $x, y \in \mathbb{K}^n$.

Definition 7.3
A matrix $C = (c_{ij})_{i,j=1}^n \in \mathbb{K}^{n,n}$ is *circulant*, if and only if

$$\exists (u_k)_{k \in \mathbb{Z}} \; n\text{-periodic sequence:} \quad c_{ij} = u_{i-j}, 1 \leq i, j \leq n$$

Remark 7.3.1: Circulant matrices have constant (main, sub- and super-) diagonals and their columns/rows arise by cyclic permutation from their first column/row.

7.1 Discrete Fourier transformation (DFT)

Circular matrices have the same (complex) eigenvectors, that remind us of sampled trigonometric functions sin and cos.
Notation: The n-th *root uf unity* is

$$\omega_n := \exp(-\tfrac{2\pi i}{n}) = \cos(\tfrac{2\pi}{n}) - i \sin(\tfrac{2\pi}{n})$$

and satisfies

$$\overline{\omega}_n = \omega_n^{-1} \qquad \omega_n^n = 1 \qquad \omega_n^{\frac{n}{2}} = -1 \qquad \omega_n^k = \omega_n^{k+n} \quad \forall k \in \mathbb{Z} \qquad \sum_{k=0}^{n-1} \omega_n^{kj} = \begin{cases} n & \text{if } j = 0 \\ 0 & \text{if } j \neq 0 \end{cases}$$

Now we want to confirm that eigenvectors of circulant matrices are vectors with powers of roots of unity. Consider $C \in \mathbb{C}^{n,n}$ circulant matrix with $c_{ij} = u_{i-j}$ for a n-periodic sequence $(u_k)_{k\in\mathbb{Z}}$, $u_k \in \mathbb{C}$. Let v_k be defined as

$$v_k \in \mathbb{C}^n \quad \text{with} \quad v_k = (\omega_n^{jk})_{j=0}^{n-1} \quad k \in \{0, \dots, n-1\}$$

With elementary mathematics, we can conclude, that

$$(C \cdot v_k)_j = \lambda_k \cdot (v_k)_j \qquad \text{with} \quad \lambda_k = \sum_{l=0}^{n-1} u_l \omega_n^{-lk}$$

This means, that v_k is eigenvector of C to the eigenvalue λ_k. Therefore, an orthogonal basis, the *trigonometric basis*, of $\mathbb{C}^n$ is

$$\{v_0, v_1, \dots, v_{n-1}\}$$

The matrix of change of basis, from the trigonometrical basis to the standard basis is the *Fourier matrix*, and is defined as

$$F_n = \left(\omega_n^{lj}\right)_{l,j=0}^{n-1} = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix}$$

Lemma 7.4
The scaled Fourier-matrix $\frac{1}{\sqrt{n}} F_n$ is unitary: $F_n^{-1} = \frac{1}{n} F_n^H = \frac{1}{n} \overline{F}_n$

Remark 7.4.1: For the spectrum of the Fourier-matrix is holds true: $\sigma(\frac{1}{\sqrt{n}} F_n) \subset \{1, -1, i, -i\}$

Remark 7.4.2: $\overline{F}_n = n \cdot F_n^{-1}$

Lemma 7.5
For any circulant matrix $C \in \mathbb{K}^{n,n}$, $c_{ij} = u_{i-j}$, $(u_k)_{k\in\mathbb{Z}}$ a n-periodic sequence, holds true

$$C\overline{F}_n = \overline{F}_n \operatorname{diag}(d_1, \dots, d_n), \qquad d = F_n(u_0, \dots, u_{n-1})^T$$

Definition 7.6 (Discrete Fourier transform (DFT))
The linear map $\mathcal{F}_n : \mathbb{C}^n \to \mathbb{C}^n$ with $\mathcal{F}_n(y) = F_n \cdot y$, $y \in \mathbb{C}^n$, is called *discrete Fourier transform* (DFT), i.e. for $c = \mathcal{F}_n(y)$

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{kj}, \quad k = 0, \dots, n-1$$

Remark 7.6.1: In MATLAB there is `c=fft(y)` and `y=ifft(c)` to do the Fourier transform and its inverse.

7.1.1 Discrete convolution via DFT

Recall that the discrete convolution can be written as a multiplication with a circulant matrix. We can use the Fourier transform to diagonalize this matrix:

$$z = u * x \qquad \Longleftrightarrow \qquad z = Cx, C = (u_{i-j})_{i,j=1}^{n} \qquad \Longleftrightarrow \qquad z = F_n^{-1} \operatorname{diag}(F_n u) F_n x$$

Therefore, a straightforward implementation of a discrete convolution would be

```
function z = pconvfft(u, x)
    z = ifft( fft(u) .* fft(x) );
end
```

7.1.2 Frequency filtering via DFT

Given a signal that is represented as vector with regard to the standard basis, we can transform it to the *frequency domain* by representing it with another vector with regard to the trigonometric basis. We say that the original signal is in the *time domain*.
Slow oscillations (i.e. low frequencies) correspond to coefficients $j \approx 1$ or $j \approx n$. High frequencies on the other hand occur at $j \approx \frac{n}{2}$.
This circumstance can be used to build frequency filters, e.g. a low pass filter.

7.1.3 Two-dimensional DFT

Naturally, while one-dimensional data ("signals") were represented as vector, two-dimensional data can be represented by a matrix. Two-dimensional DFT's can be implemented by nested one-dimensional DFT's:

`fft2(Y)= fft(fft(Y).').';` where `.'` transposes the matrix (without complex conjugation

7.2 Fast Fourier transform (FFT)

Because $\omega_n^{2j} = \omega_m^j$ if $n = 2m$, the Fourier transformation can be implemented via divide and conquer, at least for $n = 2^l$. If $n \neq 2^l$, there exist various tricks to still use the divide and conquer algorithm, resulting in a complexity of $\mathcal{O}(n \log n)$.
The execution time of `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.

Part II
Interpolation and approximation

8 Polynomial interpolation

8.1 Polynomials

We denote the vector space of the polynomials of degree $\leq k$, $k \in \mathbb{N}$ by

$$\mathcal{P}_k := \left\{ t \to \alpha_k \cdot t^k + \alpha_{k-1} \cdot t^{k-1} + \cdots + \alpha_1 \cdot t + \alpha_0, \quad \alpha_j \in \mathbb{K} \right\}$$

In MATLAB, polynomials can be represented by the vector $(\alpha_k, \alpha_{k-1}, \ldots, \alpha_0)$ and there exists `polyval(p,x)` that evaluates such a polynomial.

Theorem 8.1 (Dimension of the space of polynomials)

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

8.2 Theory of polynomial interpolation

Definition 8.2 (Lagrange polynomial interpolation problem)
Given the *simple nodes* $t_0, \ldots, t_n$, $n \in \mathbb{N}$, $-\infty < t_0 < t_1 < \cdots < t_n < \infty$ and the values $y_0, \ldots, y_n \in \mathbb{K}$ compute $p \in \mathcal{P}_n$ such that

$$p(t_j) = y_j \quad \text{for} \quad j = 0, \ldots, n$$

Definition 8.3 (General polynomial interpolation problem)
Given the *(eventually multiple) nodes* $t_0, \ldots, t_n$, $n \in \mathbb{N}$, $-\infty < t_0 \leq t_1 \leq \cdots \leq t_n < \infty$ and the values $y_0, \ldots, y_n \in \mathbb{K}$ compute $p \in \mathcal{P}_n$ such that

$$\frac{\mathrm{d}^k}{\mathrm{d}t^k} p(t_j) = y_j \quad \text{for} \quad k = 0, \ldots, l_j \quad \text{and} \quad j = 0, \ldots, n$$

where $l_j := \max\{ i - i' : t_j = t_i = t_{i'}, \quad i, i' = 0, \ldots, n \}$

8.2.1 Lagrange polynomials

Definition 8.4
For nodex $t_0 < t_1 < \cdots < t_n$ consider the *Lagrange polynomials*

$$L_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - t_j}{t_i - t_j} \quad i = 0, \ldots, n$$

Remark 8.4.1: $L_i \in \mathcal{P}_n$ and $L_i(t_j) = \delta_{ij}$.

Theorem 8.5 (Existence and uniqueness of Lagrange interpolation polynomial)
The Lagrange polynomial interpolation problem (as defined by definition 8.2) admits a unique solution $p \in \mathcal{P}_n$.

Remark 8.5.1: The solution $p \in \mathcal{P}_n$ is directly given by the Lagrange polynomials:

$$p(t) = \sum_{j=0}^{n} y_j \cdot L_j(t) \qquad \text{mit} \quad p(t_j) = y_j$$

Theorem 8.6 (Lagrange interpolation as linear mapping)
The polynomial interpolation in the nodes $\mathcal{T} = \{t_j\}_{j=0}^n$ defines a linear operator

$$I_{\mathcal{T}} : \begin{cases} \mathbb{K}^{n+1} & \to \mathcal{P}_n \\ (y_0, \ldots, y_n)^T & \to \text{ interpolation polynomial } p \end{cases}$$

Definition 8.7
The *generalized Lagrange polynomials* on the nodes $\mathcal{T} = \{t_j\}_{j=0}^n \subset \mathbb{R}$ are $L_i := I_{\mathcal{T}}(e_{i+1})$, $i = 0, \ldots, n$ where $e_i \in \mathbb{R}^{n+1}$ are the unit vectors.

Theorem 8.8 (Existence and uniqueness of generalized Lagrange interpolation polynomial)
The general polynomial interpolation problem (as defined by definition 8.3) admits a unique solution $p \in \mathcal{P}_n$.

8.3 Algorithms for polynomial interpolation

We now study algorithms to determine the unique Lagrange interpolation polynomial $p = I_{\mathcal{T}}(y)$ for nodes $\mathcal{T}$ and values y with

$$\mathcal{T} := \{-\infty < t_0 < t_1 \cdots < t_n < \infty\} \qquad y := \{y_0, y_1, \ldots, y_n\}$$

8.3.1 Multiple evaluations

In this section we want ot evaluate p in many points $x_1, \ldots x_N \in \mathbb{R}$, $N \gg 1$. The interpolation with the Lagrange polynomials is not efficient, it requires $\mathcal{O}(n^2)$ for every evaluation of $x_i \in \mathbb{R}$.
A more efficient formula can be derived by the following observation

$$p(t) = \sum_{i=0}^{n} L_i(t) \cdot y_i = \sum_{i=0}^{n} \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t - t_j}{t_i - t_j} y_i = \sum_{i=0}^{n} \lambda_i \prod_{\substack{j=0 \\ j\neq i}}^{n} (t - t_j) y_i = \prod_{j=0}^{n} (t - t_j) \cdot \sum_{i=0}^{n} \frac{\lambda_i}{t - t_i} y_i$$

With $p(t) \equiv 1$, $y_i = 1$ we get

$$1 = \prod_{j=0}^{n} (t - t_j) \cdot \sum_{i=0}^{n} \frac{\lambda_i}{t - t_i} \qquad \Longrightarrow \qquad \prod_{j=0}^{n} (t - t_j) = \frac{1}{\sum_{i=0}^{n} \frac{\lambda_i}{t - t_i}}$$

This results in the *Barycentric interpolation formula*, witch is given by

$$p(t) = \frac{\sum_{i=0}^{n} \frac{\lambda_i}{t - t_i} y_i}{\sum_{i=0}^{n} \frac{\lambda_i}{t - t_i}}$$

The computational costs for λ_i are $\mathcal{O}(n^2)$, but these values have to be calculated only once. Every subsequent evaluation of $p(t)$ then costs only $\mathcal{O}(n)$, which results in a total effort of $\mathcal{O}(Nn) + \mathcal{O}(n^2)$.

```
function p = intpolyval(t ,y ,x)
    n = length(t); N = length(x);
    for k = 1:n, lambda(k) = 1 / prod(t(k)-t([1:k-1,k+1:n])); end;
    for i = 1:N
        z = (x(i)-t); j = find(z == 0);
        if (~isempty(j)), p(i) = y(j); % prevent division by 0
        else
            mu = lambda ./ z; p(i) = dot(mu, y)/sum(mu);
        end
    end
end
```

8.3.2 Single evaluation

Single evaluations of p can be done using the *Aitken-Neville scheme*. It uses a simple recursion (though implemented as a loop for efficiency) to calculate $p(t)$ in $\mathcal{O}(n^2)$.

```
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) ;
end
```

8.4 Interpolation error estimates

In this section we are interested in an estimate of the interpolation error norm $\|f - I_{\mathcal{T}} f\|$ for some norm on $C(I)$. The focus lies on the asymptotic behaviour of the interpolation error.
There are two classes for the error, if $\exists C \neq C(n) : \|f - I_{\mathcal{T}} f\| \leq C \cdot T(n)$ for $n \to \infty$:

algebraic convergence with rate $p > 0$	$\exists p > 0 :$	$T(n) \leq n^{-p}$
exponential convergence	$\exists 0 < q < 1 :$	$T(n) \leq q^n$

In numerical experiments, one can determine the convergence class by measuring (n_i, ε_i) for $i = 1, 2, \ldots$, where n_i is the polynomial degree and ε_i the norm of the interpolation error.

- algebraic convergence: $\varepsilon_i \approx C n_i^{-p}$
  $\log \varepsilon_i \approx \log C - p \log n_i$, thus the plot is affine linear in log-log scale. By applying a linear regression (MATLAB `polyfit`) to points $(\log n_i, \log \varepsilon_i)$ we can find an estimate for the rate p.
- exponential convergence: $\varepsilon_i \approx C \exp(-\beta n_i)$
  $\log \varepsilon_i \approx \log C - \beta n_i$, thus the plot is affine linear in lin-log scale. By applying a linear regression (MATLAB `polyfit`) to points $(n_i, \log \varepsilon_i)$ we can find an estimate for $q := \exp(-\beta)$.

8.5 Chebychev interpolation

When we approximate a function by polynomial interpolation, we have the freedom to choose the interpolation nodes any way we like.

Definition 8.9 (Chebychev polynomial)
The n-th *Chebychev polynomial* is $T_n(t) := \cos(n \arccos t) \in \mathcal{P}_n, \quad -1 \le t \le 1$

Remark 8.9.1: Another, recursive, definition would be: $T_n(t) = 2T_{n-1}(t) \cdot t - T_{n-2}(t)$ with $T_0(t) = 1, T_1(t) = t$

Remark 8.9.2: The zeros of T_n are $t_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \dots n$

9 Piecewise polynomials

In this chapter we study data interpolation. That is, we want to model a functional relation $f : I \subset \mathbb{R} \to \mathbb{R}$ from the (exact) measurements (t_i, y_i), $i = 0, \ldots, n$. This gives already a first interpolation constraint: $f(t_i) = y_i$.

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, like monotonicity or positivity.

Definition 9.1 (Monotonic data)
The data $\{(t_i, y_i)\}_{i=0}^n$ are called *monotonic* when $y_i \geq y_{i-1}$ or $y \leq y_{i-1}$ for all i.

Definition 9.2 (Convex/concave data)
The data$\{(t_i, y_i)\}_{i=0}^n$ are called *convex* (*concave)* if

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

Definition 9.3
$f : I \subset \mathbb{R} \to \mathbb{R}$ is called *convex*, if

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y) \quad \forall\, 0 \leq t \leq 1 \quad \forall\, x, y \in I$$

Similarly, f is called *concave* if

$$f(tx + (1-t)y) \geq tf(x) + (1-t)f(y) \quad \forall\, 0 \leq t \leq 1 \quad \forall\, x, y \in I$$

9.2 Piecewise Lagrange interpolation

A first idea is to use *piecewise polynomials* with respect to a partition (mesh) $\mathcal{M} := \{a = x_0 < x_1 < \cdots < x_m = b\}$ of the interval $I = [a, b]$.

9.2.1 Piecewise linear interpolation

The simplest piecewise interpolation is the linear interpolation, with interpolant

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

Theorem 9.4 (Local shape preservation by piecewise linear interpolation)
Let $s \in C([t_0, t_n])$ be the piecewise linear interpolant of $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \ldots, n$, for every subinterval $I = [t_j, t_k] \subset [t_0, t_n]$ holds true:

if $(t_i, y_i)_{\vert I}$ are positive/negative	$\Longrightarrow$	$s_{\vert I}$ is positive/negative.
if $(t_i, y_i)_{\vert I}$ are monotonic increasing/decreasing	$\Longrightarrow$	$s_{\vert I}$ is monotonic increasing/decreasing.
if $(t_i, y_i)_{\vert I}$ are convex/concave	$\Longrightarrow$	$s_{\vert I}$ is convex/concave.

The drawback of piecewise linear interpolation is that f is only C^0, not even C^1.

9.2.2 Piecewise polynomial interpolation

Instead of linear interpolation, we can also choose a polynomial degree $d \in \mathbb{N}$, and gather $d + 1$ nodes. The problem is that f is still only C^0, and for $d > 1$, we do not have shape preservation.

9.3 Cubic Hermite interpolation

The goal is to find $f \in C^1$ with $f(t_i) = y_i$ and $f'(t_i) = c_i$ for given points (t_i, y_i) and slopes c_i. This is done with the following four local basis polynomials:

$$H_1(t) = \Phi\left(\tfrac{t_i - t}{h_i}\right) \qquad H_2(t) = \Phi\left(\tfrac{t - t_{i-1}}{h_i}\right) \qquad H_3(t) = -h_i\psi\left(\tfrac{t_i - t}{h_i}\right) \qquad H_3(t) = h_i\psi\left(\tfrac{t - t_{i-1}}{h_i}\right)$$

where

$$h_i = t_i - t_{i-1} \qquad \Phi(\tau) = 3\tau^2 - 2\tau^3 \qquad \psi(\tau) = \tau^3 - \tau^2$$

The interpolation polinomial s is then given by

$$s(t) = y_{i-1}H_1(t) + y_iH_2(t) + c_{i-1}H_3(t) + c_iH_4(t) \quad \text{for} \quad t \in [t_{i-1}, t_i]$$

To locally evaluate, one can use the following MATLAB function:

```
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;
end
```

9.3.1 Choice of slopes c_i

9.3.1.1 Average of local slopes

One possiblity is to choose the slopes acording to the avarage of the local slopes:

$$c_i = \begin{cases} \Delta_1 & \text{if } i = 0 \\ \Delta_2 & \text{if } i = n \\ \frac{t_{i+1}-t_i}{t_{i+1}-t_{i-1}}\Delta_i + \frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}}\Delta_{i+1} & \text{else} \end{cases} \qquad \Delta_i = \frac{y_i - y_{i-1}}{t_i - t_{i-1}}$$

This results in a *linear* local interpolation operator, but it is not shape preserving.

9.3.1.2 Shape preserving Hermite interpolation

To preserve monotonicity, one can use a "limiter":

$$c_i = \begin{cases} 0 & \text{if } \operatorname{sgn}(\Delta_i) \neq \operatorname{sgn}(\Delta_{i+1}) \\ \text{weighted average of } \Delta_i \text{ and } \Delta_{i+1} & \text{otherwise} \end{cases}$$

As average one can use the weighted harmonic mean with weights w_a, w_b with $w_a + w_b = 1$:

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

Note that this cubic Hermite interpolation is *non-linear*.

9.4 Splines

Definition 9.5 (Spline space)
Given an interval $I = [a, b] \subset \mathbb{R}$ and a mesh $\mathcal{M} = \{a = t_0 < t_1 < \cdots < t_n = b\}$, the vector space $\mathcal{S}_{d,\mathcal{M}}$ of the *spline functions* of degree d (or order $d + 1$) is defined by

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

Remark 9.5.1: The dimension of the spline space can be determined by the following counting argument:

$$\dim \mathcal{S}_{d,\mathcal{M}} = n \cdot \dim \mathcal{P}_d - \#\{C^{d-1} \text{ continuity constraints}\} = n \cdot (d+1) - (n-1) \cdot d = n + d$$

9.4.1 Cubic spline interpolation

Cubic splines with $d = 3$ are of special interest, because they are C^2-functions, which are perceived (by humans) as "smooth". One can use the same representation as with the Hermite interpolation. This already guarantees the continuity of s', only the continuity of s'' has to be enforced by the choice of $s'(t_i)$. This does however result in an under determined $(n-1) \times (n+1)$ linear system of equations. To find to additional constraints, there are (at least) three possible choices:

1. Complete cubic spline interpolation: $s'(t_0) = c_0$ and $s'(t_n) = c_n$ prescribed.
2. Natural cubic spline interpolation: $s''(t_0) = s''(t_n) = 0$.
3. Periodic cubic spline interpolation: $s'(t_0) = s'(t_n)$ and $s''(t_0) = s''(t_n)$.

In MATLAB, this can be done using `spline(t,y,x)`.

10 Numerical quadrature

The goal of this chapter is to find an approximate solution to $\int_\Omega f(x)\,\mathrm{d}x$, where $f : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ is only available as function (i.e. only point evaluations are possible).

10.1 Quadrature formulas

Definition 10.1 (n-point quadrature formula)
An n-point *quadrature formula* on the interval $[a, b]$ is defined as

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \sum_{j=1}^{n} \omega_j^n f(\xi_j^n)$$

where w_j^n are the *quadrature weights* and ξ_j^n are the quadrature nodes.

Remark 10.1.1: Using the transformation formula for integrals, it is possible to reduce any integral to the reference interval $[-1, 1]$:

$$\int_a^b f(t)\,\mathrm{d}t = \tfrac{1}{2}(b-a)\int_{-1}^{1} f(\tfrac{1}{2}(1-t)a + \tfrac{1}{2}(t+1)b)\,\mathrm{d}t$$

As always, we study the error, and in this case, the quadrature error

$$E(n) := \left| \int_a^b f(t)\,\mathrm{d}t - Q_n(f) \right|$$

and its asymptotic behaviour.

10.2 Polynomial quadrature formulas

One idea is to replace the integrand f with $p_{n-1} \in \mathcal{P}_{n-1}$, a polynomial of degree $n-1$ for given interpolation nodes $\{t_0, \ldots, t_{n-1}\} \subset [a, b]$. It is possible to use the Lagrange polynomials $L_i(t)$: $\quad p_{n-1}(t) = \sum_{i=0}^{n-1} f(t_i) L_i(t)$.
The quadrature formula in this case looks as follows

$$\int_a^b f(t)\,\mathrm{d}t \approx \int_a^b p_{n-1}(t)\,\mathrm{d}t = \sum_{i=0}^{n-1} f(t_i) \int_a^b L_i(t)\,\mathrm{d}t \qquad \text{weights } \omega_i := \int_a^b L_i(t)\,\mathrm{d}t \qquad \text{nodes } \xi_i = t_{i-1}$$

For $n = 0$ we get a 1-point quadrature formula, the so-called *mid-point formula*:

$$\int_a^b f(t)\,\mathrm{d}t \approx (b-a) f(\tfrac{1}{2}(a+b))$$

The above idea yields also the *Newton-Cotes formulas*:

- $n = 1$: Trapezoidal rule
- $n = 2$: Simpson rule
- $n = 3$: Milne rule
- $n = 4$: Weddle rule

For $n > 7$ however, the weights get negative and this compromises the numerical stability.

10.3 Composite quadrature

A reduction of the quadrature error can be achieved by splitting the integration interval as follows, for $a = x_0 < x_1 < \cdots < x_{m-1} < x_m = b$

$$\int_a^b f(t)\,\mathrm{d}t = \sum_{j=1}^{m} \int_{x_{j-1}}^{x_j} f(t)\,\mathrm{d}t$$

This also increases the total number of evaluations of f, and thus needs to be balanced with the gain in accuracy to achieve optimal efficiency.
It seems difficult to predict the quadrature error without knowing the integrand. It is however possible to look at the asymptotic error for $m \to \infty$ for certain classes of integrands and "uniform" meshes.

Definition 10.2 (Order of a quadrature formula)
A gauge for the "quality" of a quadrature formula Q_n is given by the *order*:

$$\operatorname{order}(Q_n) := \max\left\{ n \in \mathbb{N}_0 : \quad Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \quad \forall\, p \in \mathcal{P}_n \right\} + 1$$

Remark 10.2.1: Polynomial quadrature formulas are by construction exact for $f \in \mathcal{P}_{n-1}$, and thus a n-point polynomial quadrature formula has at least order n.

Theorem 10.3 (Convergence of composite quadrature formulas)
For a composite quadrature formula Q based on a local quadrature formula of order $p \in \mathbb{N}$ holds

$$\exists C > 0 : \quad \left| \int_I f(t)\,\mathrm{d}t - Q(f) \right| \leq C h^p \left\| f^{(p)} \right\|_{L^\infty(I)} \quad \forall f \in C^p(I), \forall \mathcal{M}$$

Lemma 10.4 (Bound for the order of quadrature formulas)
There is no n-point quadrature formula of order $2n+1$.

10.4 Gauss quadrature

Lemma 10.4 tells us, that there might be a n-point quadrature formula of order $2n$. In this section, we try to find such a formula.

Definition 10.5 (Legendre polynomials)
The n-th *Legendre polynomial* P_n is defined by

- $P_n \in \mathcal{P}_n$
- $\int_{-1}^{1} P_n(t) \cdot q(t)\,\mathrm{d}t = 0 \quad \forall q \in \mathcal{P}_{n-1}$ (orthogonality)
- $P_n(1) = 1$

Lemma 10.6 (Zeros of Legendre polynomials)
P_n has n distinct zeros in $]-1, 1[$.

Remark 10.6.1: The zeros of the Legendre polynomials form the *Gauss points*.

The gauss nodes and weights can be computed with the Golub-Welsch algorithm:

```
function [x,w] = gaussquad(n)
    b = zeros(n-1,1);
    for i = 1:(n-1), b(i) = i / sqrt(4*i*i-1); end
    J = diag(b,-1)+diag(b,1); [ev,ew] = eig(J);
    for i = 1:n, ev(:,i) ./ norm(ev(:,i)); end
    x = diag(ew); w = (2*(ev(1,:) .* ev(1,:)))';
end
```

10.5 Adaptive quadrature

Obviously there is a wide range of functions, where an equidistant mesh does not make sense. It is much more appropriate to use more points only locally, where f'' is large. The idea to do exactly this, is to use a local error estimation by comparing local results of two quadrature formulas Q_1 and Q_2 of different order. For example, one could use the trapezoidal rule of order 2 for Q_1 and the Simpson rule (order 4) as Q_2.
It can be expected, that the error $\operatorname{error}(Q_1)$ can be estimated by $Q_2(f) - Q_1(f)$. The adaptive quadrature then works as follows:

- Error estimation
  For $I_k = [x_{k-1}, x_k]$ and midpoints $p_k := \frac{1}{2}(x_{k-1} + x_k)$:

$$\mathrm{EST}_k := \Big| \underbrace{\frac{h_k}{6}\left(f(x_{k-1}) + 4f(p_k) + f(x_k)\right)}_{\text{Simpson rule}} - \underbrace{\frac{h_k}{4}\left(f(x_{k-1}) + 2f(p_k) + f(x_k)\right)}_{\text{trapezoidal rule on split mesh interval}} \Big|$$

- Termination
  The Simpson rule on $\mathcal{M}$ yields a preliminary result I, and we stop, if

$$\sum_{k=1}^{m} \mathrm{EST}_k \leq \mathrm{RTOL} \cdot I$$

- Local mesh refinement
  We refine the mesh $\mathcal{M}$ to $\mathcal{M}^* := \mathcal{M} \cup \{p_k : k \in S\}$, where

$$S = \left\{ k \in \{1, \ldots, m\} : \mathrm{EST}_k \geq \eta \cdot \frac{1}{m} \sum_{j=1}^{m} \mathrm{EST}_j \right\} \qquad \text{with} \quad \eta \approx 0.9$$

Part III
Integration of ordinary differential equations

11 Single step methods

11.1 Initial value problems for ODEs

Definition 11.1 (Initial value problem for first-order ordinary differential equation)
An *initial value problem* (IVP) for a *first-order ordinary differential equation* (ODE) is given by

$$\dot{y} = f(t, y) \qquad y(t_0) = y_0$$

- $f : I \times D \to \mathbb{R}^d$ is the *right hand side* with $d \in \mathbb{N}$. Often given in procedural form.
- $I \subset \mathbb{R}$ is the *time interval* of the time variable t.
- $D \subset \mathbb{R}^d$ is the *state space/phase space* of the state variable y.
- $\Omega = I \times D$ is the *extended state space* of tupels (t, y).
- t_0 is the initial time with the initial state y_0.

Remark 11.1.1: If $f = f(y)$, i.e. does not depend on the time, the ODE is called *autonomous*. In this case, if $t \to y(t)$ is a solution, for any $\tau \in \mathbb{R}$, $t \to y(t + \tau)$ is a solution, too. Therefore, the initial time is not relevant.
Remark 11.1.2: Every ODE can be transformed to an autonomous ODE by introducing another component in the state vector:

$$\dot{y} = f(t, y) \qquad \Longrightarrow \qquad z(t) = \begin{pmatrix} y(t) \\ t \end{pmatrix} \quad \dot{z} = \begin{pmatrix} f(z_{d+1}, (z)_{1:d}) \\ 1 \end{pmatrix}$$

Definition 11.2
An ordinary differential equation of order n is defined as

$$y^{(n)} = f(t, y, \dot{y}, \ldots, y^{(n-1)})$$

where $y^{(n)}$ stands for the n-th temporal derivative of y.

Remark 11.2.1: Every high order ODE can be transformed to a first-order ODE of size nd:

$$y^{(n)} = f(t, y, \dot{y}, \ldots, y^{(n-1)}) \qquad \Longrightarrow \qquad z(t) := \begin{pmatrix} y(t) \\ \dot{y}(t) \\ \vdots \\ y^{(n-1)}(t) \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} \in \mathbb{R}^{dn} \quad \dot{z} = \begin{pmatrix} z_2 \\ z_3 \\ \vdots \\ z_n \\ f(t, z_1, \ldots, z_n) \end{pmatrix}$$

Theorem 11.3 (Theorem of Peano & Picard-Lindelöf)
If $f : \hat{\Omega} \to \mathbb{R}^d$ is locally Lipschitz continuous, then for all initial conditions $(t_0, y_0) \in \hat{\Omega}$ the IVP has a solution $y \in C^1(J(t_0, y_0), \mathbb{R}^d)$ with maximal (temporal) domain of definition $J(t_0, y_0) \subset \mathbb{R}$.

Remark 11.3.1: The domain of definition/domain of existence $J(t_0, y_0)$ usually depends on (t_0, y_0).

Remark 11.3.2: If $J(t_0, y_0) = I$, the solution $y : I \to \mathbb{R}^d$ is called *global*.

Definition 11.4 (Evolution operator)
Consider the autonomous IVP $\dot{y} = f(y)$ with $y(0) = y_0$, that has a global solution $J(y_0) = \mathbb{R}$ for all $y_0 \in D$. Then the mapping

$$\Phi : \begin{cases} \mathbb{R} \times D \to D \\ (t, y_0) \to \Phi^t y_0 := y(t) \end{cases}$$

where $t \to y(t) \in C^1(\mathbb{R}, \mathbb{R}^d)$ is the unique (global) solution of the IVP, is the *evolution operator* for the autonomous ODE $\dot{y} = f(y)$.

11.2 Euler methods

The goal is to find an approximate solution of $\dot{y} = f(t, y)$ with $y(t_0) = y_0$ on $[t_0, T]$ up to the *final time* T. The idea of Euler methods essentially is *time-stepping*: Successively approximate the evolution on small intervals $[t_{k-1}, t_k], k = 1, \ldots, N$ with $t_N = T$.

11.2.1 Explicit Euler method

The solution on $[t_{k-1}, t_k]$ is approximated by the tangent curve to the current initial condition. The explicit Euler method generates a sequence $(y_k)_{k=0}^N$ by the recursion

$$y_{k+1} = y_k + h_k f(t_k, y_k), \quad k = 0, \dots, N-1$$

with the local timestep (stepsize) $h_k = t_{k-1} - t_k$.

11.2.2 Implicit Euler method

Instead of considering the forward difference quotient, one could also take the backward difference quotient. This leads to another simple time-stepping scheme, the implicit Euler method:

$$y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1}), \quad k = 0, \dots, N-1$$

Note that this requires solving a (possibly non-linear) system of equations to obtain y_{k+1}. Also note that both the implicit and explicit Euler method are first order methods.

11.3 Single step methods

Definition 11.5 (Single step method (for autonomous ODEs))
Given a discrete evolution $\Psi : \Omega \subset \mathbb{R} \times D \to \mathbb{R}^d$, an initial state y_0 and a temporal mesh $\mathcal{M} = \{t_0 < t_1 < \cdots < t_N = T\}$ the recursion

$$y_{k+1} := \Psi(t_{k+1} - t_k, y_k), \quad k = 0, \dots, N-1$$

defines a single step methods (SSM) for the autonomous IVP $\dot{y} = f(y)$ with $y(0) = y_0$.

Remark 11.5.1 (Notation): $\Psi^h y := \Psi(h, y)$

Remark 11.5.2: The mapping Ψ tries to approximate $y(t_k)$ by y_k, and thus $\Psi(h, y) \approx \Phi^h y$.

11.4 Convergence of single step methods

An important issue is the accuracy of the approximation $y_k \approx y(t_k)$. As with composite numerical quadrature, it is in general impossible to predict the error $\|y_n - y(T)\|$ for a particular choice of timesteps. Again, we consider the asymptotic behaviour of the error for timestep $h := \max_k h_k \to 0$.

Definition 11.6 (Single step method of order p)
The sequence $(y_k)_k$ generated by a single step method of order $p \in \mathbb{N}$ for $\dot{y} = f(t, y)$ on a mesh $\mathcal{M} := \{t_0 < t_1 < \cdots < t_N = T\}$ satisfies

$$\max_k \|y_k - y(t_k)\| \le Ch^p \qquad \text{for} \quad h := \max_{k=1,\dots,N} |t_k - t_{k-1}| \to 0$$

with $C > 0$ independent of $\mathcal{M}$, provided that f is sufficiently smooth.

11.5 Runge-Kutta methods

Definition 11.7 (Explicit Runge-Kutta method)
For $b_i, a_{ij} \in \mathbb{R}$, $c_i = \sum_{j=1}^{i-1} a_{ij}$, $i, j = 1, \dots, s$ and $s \in \mathbb{N}$, an *s-stage Runge-Kutta single step method* (RK-SSM) for the IVP is defined by

$$k_i := f\big(t_0 + c_i h, y_0 + h \sum_{j=0}^{i-1} a_{ij} k_j\big) \qquad y_1 := y_0 + h \sum_{i=1}^{s} b_i k_i$$

The $k_i \in \mathbb{R}^d$ are called increments.

There is a shorthand notation for (explicit) Runge-Kutta methods, the so-called *Butcher scheme*:

$$\begin{array}{c|c} c & \mathcal{U} \\ \hline & b^T \end{array} \quad := \quad \begin{array}{c|ccc} c_1 & 0 & \cdots & 0 \\ c_2 & a_{21} & \ddots & \vdots \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{s,s-1} \quad 0 \\ \hline & b_1 & \cdots & b_s \end{array}$$

Note that U is a strictly lower triangular $s \times s$ matrix. Also it is necessary that $\sum_{i=1}^{s} b_i = 1$.

In MATLAB there is `ode45(odefun, tspan, y0)` to solve ODEs.

11.6 Stepsize control

As with numerical quadrature, an equidistant mesh is in most cases not optimal. In fact, it usually totally fails if the solution has a singularity. Therefore, it is important that the stepsize is adjusted dynamically, based on the solution.
Of course, there are two (contradicting) goals: On one side, the method should be efficient (i.e. N should be as small as possible), and at the same time we want accuracy, i.e $\max \|y(t_k) - y_k\| < \text{TOL}$.
To achieve this, local-in-time stepsize control is used. It tries to balance the one-step-error by adjusting the current stepsize h_k and predicting a suitable next timestep h_{k+1}.
Note that this is not optimal. In fact, if a small error in a single timestep leads to a large error at later times, then the local-in-time timestep control is powerless and will not even notice anything. Nevertheless it is used almost exclusively today, for several reasons:

- We don't want to discard past timesteps, which could amount to tremendous waste of computational resources.
- It is inexpensive and works for many practical problems.
- There is no reliable method that can deliver guaranteed accuracy for general IVP.

The stepsize control estimates the one-step-error by the same heuristics as for adaptive quadrature. Two discrete evolutions of different order are compared, to get an estimate for the current one-step-error: Let Ψ^h and $\tilde{\Psi}^h$ be two discrete convolutions with $\text{order}(\hat{\Psi}^h) > \text{order}(\Psi^h)$.

$$\underbrace{\Phi^h y(t_k) - \Psi^h y(t_k)}_{\text{one-step-error}} \approx \text{EST}_k := \hat{\Psi}^h y(t_k) - \Psi^h y(t_k)$$

Now EST_k is compared against an absolute and relative tolerance.

$\text{EST}_k \leq \max\{\text{ATOL}, \|y_k\| \, \text{RTOL}\}$:	Carry out next timestep, and use larger stepsize (e.g. αh with some $\alpha > 1$) for the following step.
$\text{EST}_k > \max\{\text{ATOL}, \|y_k\| \, \text{RTOL}\}$:	Repeat current step with smaller stepsize, e.g. $\frac{1}{2}h$.