Numerical Methods

8. Januar 2021

Teil I
Computing with Matrices

1 Computational Effort

We use *Asymptotic Computation Complexity* (O-bounds) to judge our algorithms. But note that in many cases this approach is imprecise because on today's computers, *memory* bandwidth and latency is a key bottleneck.

Lem. (Matrix Product) The matrix product of an $m \times n$ and $n \times k$ matrix has complexity $O(mnk)$

Tricks to reduce complexity:

- Exploit associativity of operations
- Exploit hidden summations

2 Machine Arithmetic

Machine Numbers have a finite precision, hence any involved computation suffers from **Roundoff**.

Def. (Cancellation) Subtraction of almost equal numbers and accordingly an extreme *amplification of relative errors.*

Tricks to avoid cancellation:

- Trigonometric Identities
- Case Distinction
- Taylor Approximations
- Computing Diff. Quot. through Approx.

Def. (Stability) An Algorithm F is **numerically stable** if for all $x \in X$ its result $F(x)$ (possibly affected by roundoff) is the exact result for 'slightly perturbed' data.

Teil II
Linear Systems

1 Theory

Def. The Condition Number of a matrix is

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

and is a measure for stability (≈ 1 for stability). Note that it is infinite for singular matrices. Intuitively *cond*(A) catches the maximum stretching factor divided by the minimum stretching factor.

Lem. If $m = n$ and $\mathbf{A}$ is regular/invertible, then its 2-norm condition number is $\text{cond}_2(\mathbf{A}) = \sigma_1/\sigma_n$

---

2 LU Decomposition

There are two methods to choose pivots in the elimination process with a Permutation Matrix:

- Partial Pivoting: Max of column
- Full Pivoting: Max of remaining matrix

Complexities:

- Gauss elim. $\mathcal{O}(n^3)$, back substitution $\mathcal{O}(n^2)$
- LU: decomp. $\mathcal{O}(n^3)$, solve $\mathcal{O}(n^2)$
- Inverse: compute $\mathcal{O}(n^3)$, solve $\mathcal{O}(n^2)$

3 Exploiting Structure when Solving Linear Systems

Thm. (Low Rank Modification) Use the *Sherman-Morrison-Woodbury* Formula to compute the inverse of slightly modified (rank-1-modification) matrix.

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

4 Matrix Storage Formats

Dense Matrices are either stored in Column-Major Ordering or Row-Major Ordering. By default Eigen employs Column-Major Ordering.

Sparse Matrices can be stored in

- COO/Triplet: Vector of triplets of (row, col, value)
- CCS: Compressed Column Storage
- CRS: Compressed Row Storage

where CRS consists of the arrays:

- val: All non-zero values in Row-Major Order
- col_idx: Column index of each entry in val
- row_ptr: Pointers to first elements of row

Note that size$(row_ptr) = m + 1$ (Sentinel).

Teil III
Linear Least Squares

Goal: Generalisation of solving $\mathbf{Ax} = \mathbf{b}$, find

$$\mathbf{x} \in \underset{\mathbf{y}\in\mathbb{R}^n}{\text{argmin}} \|\mathbf{Ay} - \mathbf{b}\|_2^2$$

Critical comparison of Methods:

- Normal Equations: $\mathcal{O}(n^2m + n^3)$
  → blows up instability
- Extended Normal Equations:
  → same conditioning as $\mathbf{A}$
  → sparsity preserved
- Householder QR: $\mathcal{O}(n^2m)$
  → always stable
  → loss of sparsity
- SVD:
  → no full rank requirement for $\mathbf{A}$

1 Normal Equation Methods

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$$

Warning: Normal Equations are vulnerable to instability since $\text{cond}_2(\mathbf{A}^T\mathbf{A}) = \text{cond}_2(\mathbf{A})^2$

---

2 Singular Value Decomposition

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathsf{H}} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\mathsf{H}}, \quad \mathbf{A} \in \mathbb{K}^{m\times n},$$

$p := \min\{m, n\}$, $r := \text{rank}(\mathbf{A})$,
$\mathbf{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_p)$ $\sigma_1 > \sigma_2 > \ldots > \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0$

Full:

- $\mathbf{U} \in \mathbb{K}^{m\times m}$ $[\mathcal{R}(\mathbf{A})|\mathcal{N}(\mathbf{A})]$ (unitary)
- $\mathbf{\Sigma} \in \mathbb{K}^{m\times n}$ (generalized diagonal)
- $\mathbf{V} \in \mathbb{K}^{n\times n}$ $[\mathcal{R}(\mathbf{A}^{\mathsf{H}})|\mathcal{N}(\mathbf{A}^{\mathsf{H}})]$ (unitary)

Economical:

- $\mathbf{U} \in \mathbb{K}^{m\times p}$ $[\mathcal{R}(\mathbf{A})]$ (orthogonal columns)
- $\mathbf{\Sigma} \in \mathbb{K}^{p\times p}$ (diagonal)
- $\mathbf{V} \in \mathbb{K}^{n\times p}$ $[\mathcal{R}(\mathbf{A}^{\mathsf{H}})]$ (orthogonal columns)

Numerical Rank $r := \max_{j\in\{1,\ldots,p\}} \left(\frac{\sigma_j}{\sigma_1} \geq TOL\right)$
Cost of Eco SVD $\mathcal{O}(\min\{m,n\}^2 \max\{m,n\})$

Lem. (LSQ by SVD)

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min} \|\mathbf{Ax} - \mathbf{b}\|_2 = \mathbf{V}_1\mathbf{\Sigma}_r^{-1}\mathbf{U}_1^T\mathbf{b}$$

If the lsq has multiple solutions, $\mathbf{x}^*$ has minimal norm.

Thm. (Low Rank Approximation) Let $\mathbf{A}_k$ be the matrix resulting from only keeping the first k singular values of $\mathbf{A}$. $\mathbf{A}_k$ is the best rank-k approximation of $\mathbf{A}$.

Method: (Principal Component Analysis)
Given a dataset in the form of a matrix $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathsf{H}}$, such that every column represents a time series.

- Columns of $\mathbf{U}$ capture dominant trends.
- Columns of $\mathbf{V}$ capture how strong the trends are in a particular column of $\mathbf{A}$
- Diagonal Entries of $\mathbf{\Sigma}$ tell us how dominant the trend is overall.

Method: (Proper orgthogonal decomp.)
Closely related is the POD problem, where we seek k-dimensional subspace U_k of $\mathbb{R}^m$, for which the sum of squared distances of the data points $\mathbf{A} = [\mathbf{a}_1, \ldots \mathbf{a}_n]$ to U_k is minimal.
Using the *Low Rank Approximation* Theorem we can prove that

$$U_k = \mathcal{R}(\mathbf{U}_{:,1:k})$$

3 QR Decomposition

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

Idea: LLQ Problems are easier to solve for triangular matrices, hence we manage to decompose $\mathbf{A}$ into $\mathbf{QR}$ we have such a System since orthogonal matrices are norm-invariant.

Method: (Householder Reflections) The following Householder matrix performs a reflection

$$\mathbf{H}(\mathbf{v}) := \mathbf{I} - 2\mathbf{v}\mathbf{v}^\top$$

at the hyperplane with normal unit vector $\mathbf{v}$ (Intuitively H$(\mathbf{v})$ subtracts the projection of $\mathbf{x}$ on $\mathbf{v}$ twice).
We can reduce a matrix $\mathbf{A}$ to $\mathbf{R}$ using n successive transformations

$$\mathbf{H}(\mathbf{v}_n)\ldots\mathbf{H}(\mathbf{v}_1)\mathbf{A} = \mathbf{R}$$

, where $\mathbf{H}(\mathbf{v}_i)$ reflects the lower part of the i'th column of the current $\mathbf{A}$ on $\mathbf{e}_i$. E.g

$$\mathbf{v}_1 = \mathbf{a}_1 \pm \|\mathbf{a}_1\| \mathbf{e}_1$$

Method: (Givens Rotations) We can selectively eliminate entries with Givens rotations. The following matrix rotates everything in the hyperplane defined by $\mathbf{e}_1, \mathbf{e}_k$:

$$\mathbf{G}(1,k,\theta) = \begin{bmatrix} c & \cdots & s & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots \\ -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ \vdots \\ a_n \end{bmatrix}$$

Note that $c = \cos(\theta), s = \sin(\theta)$. Two eliminate a_k we solve the equations.

$$c^2 + s^2 = 1 \quad \text{and} \quad -sa_1 + ca_k = 0$$

As with householder reflections, there are always two possibilites, one of whom is cancellation free.

Lem. (Modifications Techniques) Computing the QR decomposition of a slightly modified matrix (rank-1-modification, adding a row, adding a column) can be done efficiently in $\mathcal{O}(mn + n^2)$.

4 Constrained Least Squares

Problem: Find $x \in \mathbb{R}^n$ such that

$$\|\mathbf{Ax} - \mathbf{b}\| \to \min \quad \text{and} \quad \mathbf{Cx} = \mathbf{d}.$$

Method: (Lagrangian Multipliers) We intoduce the multiplier $\mathbf{m} \in \mathbb{R}^p$ to solve

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \sup_{\mathbf{m}\in\mathbb{R}^p} \|\mathbf{Ax} - \mathbf{b}\|_2 + \mathbf{m}^T(\mathbf{Cx} - \mathbf{d})$$

and we notice that for any finite solution $(\mathbf{Cx}-\mathbf{d}) = 0$. By realising that that fo the solution all partial derivatives must be zero we can obtain the *augmented normal equations*.

Method: (By SVD) We have

$$\mathbf{x} \in \mathbf{x}_0 + \mathcal{N}(\mathbf{C}) \quad \mathbf{x}_0 = \text{particular solution}$$

Hence since the SVD gives a basis of $\mathcal{N}(\mathbf{C})$ we write

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{V}_2\mathbf{y}$$

for some $\mathbf{y}$ which leads to the standard lsq:

$$\|\mathbf{AV}_2\mathbf{y} - (\mathbf{b} - \mathbf{Ax}_0)\| \to \min$$

Teil IV
Filtering Algorithms

1 Filters & Convolution

Def. (Filter) A function $F : l^\infty(\mathbb{Z}) \to l^\infty(\mathbb{Z})$ where $l^\infty(\mathbb{Z})$ is the space of bounded infinite sequences

$$l^\infty(\mathbb{Z}) = \left\{(x_j)_{j\in\mathbb{Z}} : \sup|x_j| < \infty\right\}$$

Def. (LT-FIR) A Filter that is:

- Linear
- Time-Invariant: Shifting the input in time leads to the same output shifted in time.
- Finite: $\exists M \forall j : x_j = 0$ if $|j| > M \implies \exists N \forall j : y_j = 0$ if $|j| > N$
- Causal: The output does not start before the input.

Such a filter is uniquely characterized by its **Impulse response**:

$$F(\delta_{j,0}) = \ldots, 0, h_0, h_1, \ldots, h_{n-1}, 0, \ldots$$

For inputs $\mathbf{x} \in \mathbb{R}^m$ we get outputs $\mathbf{y} \in \mathbb{R}^{m+n-1}$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 \\ h_3 & h_2 & h_1 & h_0 \\ 0 & h_3 & h_2 & h_1 \\ 0 & 0 & h_3 & h_2 \\ 0 & 0 & 0 & h_3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_4 \end{bmatrix}$$

Def. (Discrete Convolution) A Filter where given $\mathbf{x} = [x_0, \ldots, x_{m-1}]^\top \in \mathbb{K}^m, \mathbf{h} = [h_0, \ldots, h_{n-1}]^\top \in \mathbb{K}^n$ their DCONV is the vector $\mathbf{y} \in \mathbb{K}^{m+n-1}$ with components.

$$y_k = \sum_{j=0}^{m-1} h_{k-j} x_j$$

Def. (Periodic Convolution) Given two n-periodic signals $(u_k)_{k\in\mathbb{Z}}, (x_k)_{k\in\mathbb{Z}}$ PCONV yields the n-periodic signal:

$$(y_k) := (u_k) *_n (x_k), y_k := \sum_{j=0}^{n-1} u_{k-j} x_j$$

PCONV can be represtend by a matrix

$$\mathbf{C} = [c_{ij}]_{i,j=1}^n, c_{ij} = u_{j-i}$$

. Such a matrix is called **circulant**. The vector $\mathbf{u}$ s.t. $\mathbf{C} = \text{circul}(\mathbf{u})$ corresponds to $\mathbf{C}_{:,1}$.

2 Discrete Fourier Transform

The Fourier Matrix for inputs $\mathbf{y} \in \mathbb{C}^n$ is given by

$$\mathbf{F}_n = \left[\omega_n^{lj}\right]_{l,j=0}^{n-1} \in \mathbb{C}^{n,n} \quad \omega_n = \exp\left(\frac{-2\pi i}{n}\right)$$

and $\text{DFT}_n(\mathbf{y}) := \mathbf{F}_n\mathbf{y} = \left[\sum_{j=0}^{n-1} y_j \omega_n^{kj}\right]_{k=0}^{n-1}$.
Properties include

$$\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^{\mathsf{H}} = \frac{1}{n}\overline{\mathbf{F}}_n$$

Method: (Frequency Filtering) The columns $\mathbf{v}_k$ of $\mathbf{F}_n$ are trigonometric basis vectors, where

$$\mathbf{v}_k = \left[\cos\left(j\frac{2\pi}{n}\cdot k\right) + \imath \sin\left(\frac{2\pi jk}{n}\right)\right]_{j=0}^{n-1}$$

k is the 'frequency' and $j2\pi/n$ is the sample point. Hence the DFT_n is a basis transformation

$$B_E \leftrightarrow B_{trig}$$

Denoising and low/high filters can be implemented by manipulating the signal in the frequency domain. Given the n coeffiecient c_j we can analyse the strength of frequencies by plotting $|c_j|^2, 0 \le j \le n/2$, since $\mathbf{v}_k = \mathbf{v}_{n-k}, k = 0 \ldots n-1$.

Lem. (Diagonalizing circulant matrices) For any circulant matrix $\mathbf{C} \in \mathbb{C}^{n,n}$ we have

$$\mathbf{C} = \mathbf{F}_n^{-1} \operatorname{diag}(d_1, \ldots, d_n) \mathbf{F}_n,$$
$$[d_0, \ldots, d_{n-1}]^\top = \mathbf{F}_n [u_0, \ldots, u_{n-1}]^\top$$

In other words, the columns of $\mathbf{F}_n$ are eigenvectors of $\mathbf{C}$.

Thm. (Convolution Theorem) Periodic convolution in the time-domain equals to multiplication in the frequency-domain.

$$(\mathbf{u}) *_n (\mathbf{x}) = \mathbf{F}_n^{-1}\left[(\mathbf{F}_n\mathbf{u})_j (\mathbf{F}_n\mathbf{x})_j\right]_{j=1}^n$$

Implementation Speedup: $\mathcal{O}(n^2) \to \mathcal{O}(n\log_2(n))$

Thm. (2D DFT) Is given by

$$\mathbf{C} = \mathbf{F}_m\left(\mathbf{F}_n\mathbf{Y}^\top\right)^\top = \mathbf{F}_m\mathbf{Y}\mathbf{F}_n$$

The real basis vectors can be visualized. Analogously there is a 2D-Convolution-Theorem:

$$\mathbf{X} *_{m,n} \mathbf{Y} = \text{DFT}_{m,n}^{-1}(\text{DFT}_{m,n}(\mathbf{X}) \odot \text{DFT}_{m,n}(\mathbf{Y}))$$

Method: (Deblurring an image) A blurred image can be modeled as the 2D-Convolution of the actual image with a small filter. In the 2D-frequency-domain deblurring equals component wise division.

3 Fast Fourier Transform

The DFT of a vector of length pq can be divided in p DFT's of vectors of length q. The asymptotic runtime for this special 'divide and conquer' is

$$\text{FFT}(n) \in \mathcal{O}(n\log n)$$

Special Case $p = 2$:

$$\text{DFT}(\mathbf{v}) = \text{DFT}(\mathbf{v}_{even}) + s\,\text{DFT}(\mathbf{v}_{odd})$$

Def. (Toeplitz Matrix) $\mathbf{T} = (t_{ij})_{i,j=1}^n \in \mathbb{K}^{mn}$ is a Toeplitz matrix if it has constant diagonals given by a vector $\mathbf{u} \in \mathbb{K}^{m+n-1}$. Note that every circulant matrix is also a Toeplitz matrix.

Lem. (Fast Arithmetic for Toeplitz)
Vector Multiplication in $\mathcal{O}((n+m)\log(n+m))$: Construct Circulant Matrix $\mathbf{C} \in \mathbb{K}^{m+n,m+n}$ s.t.

$$\begin{bmatrix} \mathbf{Tx} \\ \star \end{bmatrix} = \mathbf{C}\begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}$$

which can be solved in the frequency domain (Convolution Theorem).
LSE Solution in $\mathcal{O}(n^2)$: The Levinson Algorithm.

Teil V

Data Interpolation in 1D

Goal: Given n data points (t_i, y_i) and a finite-dimensional vector-space of functions V, reconstruct $f \in V$ such that $\forall i \quad f(t_i) = y_i$.
We denote with $\mathrm{I}_{\mathcal{T}}(\mathbf{y})$ the polynomial interpolation operator.

Def. (Interpol. as linear mapping)
For data points $(t_i, y_i) \quad i = 0, \dots, n$ and V spanned by basis functions $b_0, \dots b_m$ we have

$$\mathbf{Ac} := \begin{bmatrix} b_0(t_0) & \dots & b_m(t_0) \\ \vdots & & \vdots \\ b_0(t_n) & \dots & b_m(t_n) \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_m \end{bmatrix} = \mathbf{y}$$

→ The interpolant is uniquely solv. iff $m = n$
→ The basis is **cardinal** if $b_j(t_i) = \delta_{ij}$

1 Global Polynomial Interpol.

In general global polynomial interpolation is not suitable due to potentially very high sensitivity.
We differentiate the following bases for polynomial representation. In any of these bases, evaluation is possible in $\mathcal{O}(n)$ by Horner-Like schemes that exploit associativity.

Def. (Monomial Basis)

$$p_i(t) = t^i$$

Def. (Lagrange Basis)

$$L_i(t) := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - t_j}{t_i - t_j} = I_{\mathcal{T}}(\mathbf{e}_{i+1})$$

→ Cardinal Basis.

Def. (Newton Basis)

$$N_i(t) := \prod_{j=0}^{i-1} t - t_j$$

Method: (Multiple Evaluations)
→ Barycentric Interpolation Formula.
Using precomputed weights in $\mathcal{O}(n^2)$ it achieves $\mathcal{O}(Nn)$ to evaluate N points.

Method: (Single Point Evaluation)
→ Aitken-Neville scheme.
Evaluation in $\mathcal{O}(n^2)$ but addPoint in $\mathcal{O}(n)$.

Method: (Newton Interpolation)
→ Coefficients in $\mathcal{O}(n^2)$, Evaluation in $\mathcal{O}(n)$ and Adding-a-Node in $\mathcal{O}(n)$.

Coefficients are computed with the *Divided Differences* method, which is based on the following:

$$a_{\ell,m} = \frac{a_{\ell+1,m} - a_{\ell,m-1}}{t_m - t_\ell}$$

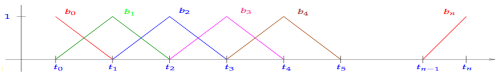
where $a_{\ell,m}$ denotes the leading coefficient of the the polynomial interpolating points $l \ \dots m$

Method: (Extrapolation)

2 Shape-Preserving Interpol.

Def. (Shape preserving)
positive data ⟶ positive interpolant.
monotonic data ⟶ monotonic interpolant.
convex data ⟶ convex interpolant.

Method: (Piecewise linear Interpol.)
→ locally shape preserving
Tent functions make a cardinal basis:



Method: (Cubic Hermite Interpol.)
→ f $\in C^1([t_0, t_n])$
Idea: Interpolate each interval as cubic polynomial but edge points have to match, and derivatives on edges have to match. But we have to decide what we want derivates to be at edge:

- Linear: (weighted mean of adjacent derivatives) -¿ No Preservation of monotonicity
- Pchip: Since slope must be zero at boundary in some cases -¿ Loss of linearity

3 Splines

Instead of fixing the boundary slopes, we add additional continuity constraints. The interpolating function f is then in the Spline Space.

Def. (Spline Space)

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

$\mathcal{S}_{d,\mathcal{M}}$ is the space of piecewise polynomial functions in C^{d-1}. One can easily see that

$$\dim \mathcal{S}_{d,\mathcal{M}} = n + d$$

Method: (Cubic Spline Interpolation) The cubic spline interpolant is a function $s \in \mathcal{S}_{3,\mathcal{M}}$ that complies with the $n+1$ interpolation conditions. Note that we still have 2 degrees of freedom, this leads to the following flavours:

- natural: $s'(t_0) = 0, s'(t_n) = 0$
- periodic: $s'(t_0) = s'(t_n), s''(t_0) = s''(t_n)$
- complete: $s'(t_0) = c_0, s'(t_n) = c_n$

The resulting functions is in C^2, weakly-local and not strictly shape preserving.

4 Trigonometric Interpol.

bla

5 Least Squares Data Fitting

Goal: Given m data points (t_i, y_i) and a set of functions V, find a continuous function $f \in V$ such that

$$f \in \operatorname*{argmin}_{\mathbf{g} \in S} \|\mathbf{g}(t_i) - y_i\|_2^2$$

We focus on **linear** data fitting, i.o.w. let V be an n-dimensional vector space spanned by functions $b_1(t), \dots, b_n(t)$. We look for coefficients

$$[x_1, \dots, x_n]^T = \operatorname*{argmin}_{\mathbf{z} \in \mathbb{R}^n} \sum_{i=1}^{m} \left| \sum_{j=1}^{n} (\mathbf{z})_j b_j(t_i) - y_i \right|^2$$

Thm. (Linear Data Fitting Solution)
The solution to the linear least squares fitting problem is the least squares solution of the system

$$\begin{bmatrix} b_1(t_1) & \dots & b_n(t_1) \\ \vdots & & \vdots \\ b_1(t_m) & \dots & b_n(t_m) \end{bmatrix} \mathbf{x} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

Method: (Regularization) We can punish large oscillations by introducing a regularization term:

$$f \in \operatorname*{argmin}_{g \in V} \left\{ \sum_{i=0}^{n} |g(t_i) - y_i|^2 + \alpha \int_a^b |g''(t)|^2 \ \mathrm{d}t \right\}$$

Teil VI

Approximation in 1D

Goal: For a given function f on $I \subset \mathbb{R}$ find a simpler function $\tilde{f}$ (called *surrogate*) such that the approximation error $\left\| f - \tilde{f} \right\|$ is small for some norm.

⟶ Focus on **Approximation by interpolation**
⟶ Hence, good interpolation nodes are key.

1 Global Polynomial Approx.

Def. (Polynomial Interp. Approx. scheme)
Given a node set $\mathcal{T} = \{t_0, \dots, t_n\} \subset I$, define

$$\tilde{f} = p \in \mathcal{P}_n \quad \text{s.t } p(t_j) = f(t_j)$$

as the interpolating polynomial.

Thm. (Weierstrass) Every continuous function defined on a close interval can be uniformly approximated by a polynomial (of increasing degree).

Thm. (Best approximation, Jackson) For $f \in C^r([-1, 1])$ we have

$$\inf_{p \in \mathcal{P}_n} \|f - p\|_\infty \leq \left(1 + \pi^2/2\right)^r \frac{(n-r)!}{n!} \left\| f^{(r)} \right\|_\infty$$
$$\in \quad \mathcal{O}(n^{-r})$$

Lem. (Sensitivity of Interpolation)
For data y and noise δ we have

$$\|\mathrm{I}_{\mathcal{T}}(y + \delta) - \mathrm{I}_{\mathcal{T}}(y)\| = \|\mathrm{I}_{\mathcal{T}}(\delta)\| \leq \|\mathrm{I}_{\mathcal{T}}\| \, \|\delta\|$$

where $\|\mathrm{I}_{\mathcal{T}}\|$ denotes the operator norm, for the infinity norm we have

$$\|I_{\mathcal{T}}\|_\infty := \sup_{\mathbf{y} \in \mathbb{R}^{n+1}} \frac{\|I_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)}}{\|\mathbf{y}\|_\infty} = \left\| \sum_{i=0}^{n} |L_i| \right\|_{L^\infty(I)}$$

which coincides with the Lebesgue Constant $\lambda_{\mathcal{T}}$.
This is important because a result for $f \in C^0(I)$ is

$$\left\| f - \tilde{f} \right\|_{L^\infty(I)} \leq (1 + \lambda_{\mathcal{T}}) \inf_{p \in \mathcal{P}_n} \|f - p\|_{L^\infty(I)}$$

Def. (Error convergence) We distinguish

- Algebraic convergence: $\left\| f - \tilde{f} \right\| = \mathcal{O}(n^{-p})$ with a rate $p > 0$
- Exponential convergence: $\left\| f - \tilde{f} \right\| = \mathcal{O}(q^n)$ for $0 < q < 1$

Def. (Chebychev interpolation)
Polynomial Approximation based on equidistant

nodes does not necessarily converge, hence we want a better node set. Based on the error bound

$$\left\|f-\tilde{f}\right\|_\infty \leq \frac{1}{(n+1)!}\left\|f^{(n+1)}\right\|_\infty \|w\|_{L^\infty(I)}$$

we look for a nodes that minimize the polynomial

$$w(t) := (t-t_0)\cdots(t-t_n)$$

On $I=[-1,1]$ these nodes are the zeros of the $n+1$'th Chebychev Polynomial

$$t_k = \cos\left(\frac{2k+1}{2n}\pi\right), \quad k=0,\ldots,n-1$$

. These nodes are a priori optimal for the approximation error, note that they gather around the borders
Error Estimate: $\lambda_{\mathcal{T}} \in \mathcal{O}(\log n)$, hence almost algebraic
Computational Aspect: There is a more efficient algorithm to evaluate the resulting interpolating polynomial then standard interpolation algorithms. The key idea is representing $\tilde{f}$ in Chebychev Polynomial Basis:
Evaluation: $\mathcal{O}(nm)$ for m points. (Clenshaw Alg.)
Comp. of coeffs.: $\mathcal{O}(n\log n)$ (FFT Alg.)

2 Trigonometric Poly. Approx.

x
x
x

3 Piecewise Polynomial Approx.

Method: (Piecewise Lagrange)
Given an interval $[a,b]$ endowed with a mesh $\mathcal{M}$, choose local degree and local interpolation nodes for each mesh cell to perform Lagrange Interpolation.
Error Estimate: For constant polynomial degree n and mesh width $h_{\mathcal{M}}$ we have

$$\|f-s\|_\infty \leqslant \frac{h_{\mathcal{M}}^{n+1}}{(n+1)!}\left\|f^{(n+1)}\right\|_\infty$$

Method: (Piecewise Cubic Hermite)
Given $f,\mathcal{M}$, the piecewise cubic hermite interpolant is defined as (Note that we use exact slopes!)

$$s_{|[x_{j-1},x_j]} \in \mathcal{P}_3, \quad s(x_j)=f(x_j), \quad s'(x_j)=f'(x_j)$$

Teil VII
Numerical Quadrature

Goal: Numerically evaluate integrals $\int_\Omega f(x)dx$ with as few evaluation points as possible (imagine f to be a computationally expensive function).

Note: Quadrature Rules are defined on the reference interval $[-1,1]$ but from Analysis we have

$$\int_a^b f(t)\,dt = \frac{b-a}{2}\int_{-1}^1 \hat{f}(t)\,dt$$

Def. (Quadrature Formula/Rule)
An n-point quadrature rule approximates an integral through a weighted sum of point values.

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

1 Polynomial Quadrature

Approximation schemes induce quadrature rules. From Lagrange Interpolation we get

$$\int_a^b f(t)dt \approx \int_a^b \tilde{f}(t)dt = \sum_{i=0}^{n-1} f(t_i)\int_a^b L_i(t)dt$$

Def. (Newton-Cotes rules) The n-point quadrature rule results from polynomial approximation with equidistant quadrature nodes

$$t_j = a + j\frac{b-a}{n-1} \quad j=0,\ldots,n-1$$

Def. (Clenshaw-Curtis rules) The n-point QR results from polynomial approximation with chebychev nodes. The weights can be computed with FFT in $\mathcal{O}(n\log n)$.

Error bounds on approximation induce quadrature error bounds since:

$$\left|\int_a^b f(t)-Q_n(f)\right| \leq |b-a|\|f-\tilde{f}\|_{L^\infty([a,b])}$$

Lem. (Clenshaw-Curtis error) From approximation error estimates on chebychev interpolation we get error convergence:

- $\mathcal{O}(\log n/n^r)$ for limited smoothness integrands
- $\mathcal{O}(x^{-n})$ for analyticly extendable integrands

2 Gauss Quadrature

Gauss Quadrature maximises order, which is the polynomial degree +1 for which the quadrature rule is guaranteed to be exact.

Lem. (Order) Gauss Quadrature is the unique rule of order $2n$. This is intuitively possible due to n degrees of freedoms in nodes and n degrees of freedom in weights.

For an n-point quadrature rule Q_n of order q with positive weight we have that

$$\forall p\in\mathcal{P}_{q-1} \quad \mathrm{Error}(f)=\mathrm{Error}(f-p)$$

from this we can derive the bound

$$\left|\int_a^b f(t)dt - Q_n(f)\right| \leq 2|b-a| \underbrace{\inf_{p\in\mathcal{P}_{q-1}}\|f-p\|_\infty}_{\text{best approx. error}}$$

Which implies algebraic convergence for limited smoothness integrands and exponential smoothness otherwise for Gauss quadrature.

3 Composite Quadrature

Piecewise polynomial quadrature gives rise to new quadrature rules:
Def. (Composite trapezoidal rule) Arises from applying Newton-Cotes rule for $n=2$ in each interval. Equivalent to quadrature of piecewise linear approximation.
Def. (Composite Simpson rule) Arises from applying Newton-Cotes rule for $n=3$ in each interval. Equivalent to quadrature of piecewise quadratic approximation.

In general gauss-quadrature is at least as good as composite quadrature on equidistant meshes.
Except in case of quadrature over the interval of periodicity of a periodic integrand, then the trapezoidal rule with equidistant nodes converges exponentially.

4 Adaptive Quadrature

In adaptive quadrature schemes the nodes are chosen depending on the integrand. Here we present an *a posteriori* algorithm, meaning that the node-set is modified in a loop until sufficient accuracy is reached.

Algorithm: (Adapt. multilevel quadrature)

$\mathcal{M} \leftarrow$ Initial mesh
while EST($\mathcal{M}$) > TOL **do**
  Mark Intervals with big error
  for Marked intervals **do**
    Add point in middle
  end for
end while

Where we estimate the error by computing the difference between composite Simpson and Trapezoidal rule.

Teil VIII

Iterative Methods

This chapter focuses on solving generic non-linear systems of equations. Without loss of generality we assume problems of the form $F(\mathbf{x}) = \mathbf{0}$.

Def. (m-point iterative method) An iterative method solves an equation by generating an arbitrarily long sequence $\mathbf{x}^{(k)}$ of approximative solutions. If the next iterate only depends on F and the m most recent iterates, we label the method m-point. We denote the *iteration function*:

$$\mathbf{x}^{(k+1)} = \Phi_F\left(\mathbf{x}^{(k)}, \ldots, \mathbf{x}^{(k-m+1)}\right)$$

Thm. (Consistency and convergence) An iterative method is consistent with $F(\mathbf{x}) = \mathbf{0}$ if

$$\Phi_F\left(\mathbf{x}^*, \ldots, \mathbf{x}^*\right) = \mathbf{x}^* \Longleftrightarrow F\left(\mathbf{x}^*\right) = \mathbf{0}$$

. The limit of a convergent and consistent iterative method with a continuous iterative function is a solution.

Def. (Speed of convergence) A convergent sequence $\mathbf{x}^{(k)}$ converges with **order** $p \geq 1$) if $\exists L > 0$:

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \leq L \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|^p \quad \forall k \in \mathbb{N}_0$$

. Order 1 convergence is called **linear**, and in this case the upper bound of L is called *rate*. Note that linear convergences already implies exponential error convergence in k. Denote $\epsilon_k := \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|$, with simple calculations we derive

$$\frac{\log \epsilon_{k+1} - \log \epsilon_k}{\log \epsilon_k - \log \epsilon_{k-1}} \approx p$$

Def. (Termination criterion) A stopping rule is the algorithm deciding wether the iterative method should Stop. In practice we use the posteriori correction based termination rule:

$$\text{STOP} \Leftarrow \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\| \leq \left\{ \begin{array}{c} \tau_{\text{abs}} \text{ or} \\ \tau_{\text{rel}} \left\|\mathbf{x}^{(k+1)}\right\| \end{array} \right.$$

Note that this criterion has no guaranteed reliability.

1 Fixed Point Iterations

:= 1-point iterative methods.

Lem. (Local linear convergence) If $\|D\Phi(\mathbf{x}^*)\| < 1$ then the iteration converges locally at least linearly.

Lem. (Higher order local convergence) If the iteration function is $m+1$ times continuously differentiable and $\Phi^{(l)}(x^*) = 0$ for $l = 1, \ldots, m$, then the iteration converges locally with order $\geq m+1$.

2 Zeros of Scalar Functions

Method: (Bisection) Similar to binary search, in each iteration the search interval is reduced to one of the halves of the current interval.
→ Linear Type Convergence (but not truly linear)

Method: (1D Newton) The concept behind Model Function Methods is that at every step: $\mathbf{x}^{(k+1)} :=$ zero of $\tilde{F}_k$, where $\tilde{F}_k$ is a suitable model function based on recent iterates.
Newton determines the next iterate $\mathbf{x}^{(k+1)}$ as a the zero of the tangent function of F at $\mathbf{x}^{(k)}$. We get:

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

→ Local quadratic convergence by previous Lemma if $F'(\mathbf{x}^*) \neq 0$

Method: (Secant Method) Define the model function as the interpolant of the last two iterates (a line).
→ Local order 1.62 convergence by extensive calc., and no derivates required.
→ Asymptotically more efficient than Newton Method, since only one evaluation per step required.

Method: (Inverse Interpolation) At step k

1. Interpolate F^{-1} by a polynomial p with the j last iterates: $p(F(x^{(k)}) = x^{(k)}$
2. $x^{(k+1)} = p(0)$

3 Newton's Method in $\mathbb{R}^n$

Method: (Newton) Analogously to 1D we get:

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

where $DF(x)$ is the Jacobian.

4 Non-linear Least Squares

Method: (Newton for LLSQ) If there are more equations than unknown, solving the system of equations becomes a minimisation problem. Formally given $F : \mathbb{R}^n \mapsto R^m, m > n$ we look for

$$\mathbf{x}^* = \arg\min \|F(\mathbf{x})\|_2^2$$

From elementary analysis we know that

$$\operatorname{grad} \|F(\mathbf{x}^*)\|_2^2 \, \mathbf{x}^* = \mathbf{0}$$

. Assuming the second derivative is available we can apply Newton's method to find a zero.

Method: (Gauss-Newton)
→ only relies on the first derivative, but no quadratic convergence, usually larger convergence domain than Newton.

1. Linearly Approximate F with
$$F(\mathbf{x}) \approx F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)})$$
2. $x^{(k+1)}$
$$= \arg\min_{\mathbf{x}} \left\|F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)})\right\|_2$$
$$= x^{(k)} + \arg\min_{\mathbf{s}} \left\|F(\mathbf{x}^{(k)}) + DF(\mathbf{x}^{(k)})\mathbf{s}\right\|_2$$
3. Solve the last linear least squares problem.

In order not to overshoot, we can add a penalty term to the minimisation problem in (2.).