Endterm Recap

Dienstag, 20. Dezember 2016 09:25

Impulses

Filter: h in Matrixrepräsentation
Input: x
Output: y

duration(output signal) $\leq$ duration(input signal) + duration(impulse response) .

Therefore, if we know that all input signals are of the form $(\ldots, x_0, x_1, \ldots, x_{m-1}, 0, \ldots)$, we can model them as vectors $\mathbf{x} = [x_0, \ldots, x_{n-1}]^\top \in \mathbb{R}^n$, *cf.* § 4.0.1, and the filter can be viewed as a linear mapping $F : \mathbb{R}^n \to \mathbb{R}^{m+n-1}$, which takes us to the realm of linear algebra.

Thus, for the filter we have a matrix representation of (4.1.14). Writing $\mathbf{y} = [y_0, \ldots, y_{2n-2}]^\top \in \mathbb{R}^{2n-1}$ for the vector of the output signal we find in the case $m = n$

$$\begin{bmatrix} y_0 \\ \vdots \\ y_{2n-2} \end{bmatrix} = \begin{bmatrix} h_0 & 0 & \cdots & 0 \\ h_1 & & & \\ & \ddots & & 0 \\ h_{n-1} & & h_1 & h_0 \\ 0 & & & \\ & \ddots & & \\ 0 & & 0 & h_{n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix}. \qquad (4.1.18)$$

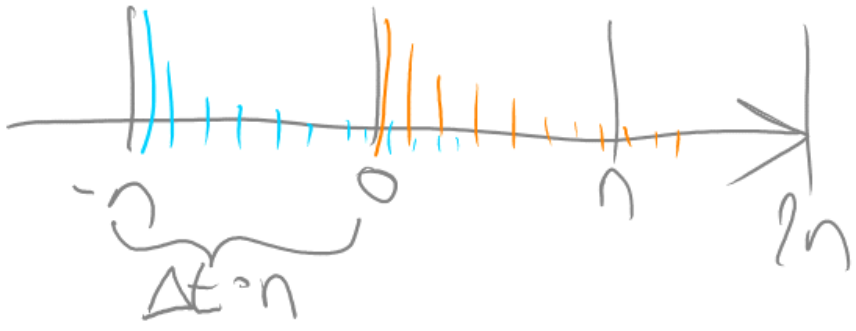
Matrix H ist zwei n hoch, damit auch h_{n-1} nicht von anderen h beeinflusst wird. Es ist also möglich, einen linearen Impuls so als periodischen darzustellen, indem man eine Periode von (2n-1) wählt und mit nullen auffüllt. (Das minus 1 ist wegen indexing)
We model the signals as vectors x and the Filter as linear mapping from $\mathbb{R}^n$ to $\mathbb{R}^{m+n-1}$

Def: Discrete Convolution

$y_i = H.row(i) * \vec{x} =$ nennt sich discrete Convolution$= \sum_{l=0}^{height(H)} x_l h_{j-l}$

n-Periodic setting

Impulses add up. In each row of H, the impulse h_i is one further to the right, so when each x is smaller than the one before it, we have something like the following. but blue and orange add up.



Formula is still $y_k = \sum_{l=0}^{n-1} p_{k-l} x_l$. at $l = 0$, we have
$p = h_l = (h_{n-1} \quad \ldots \quad h_0)$
and therefore we have the interference from n-1 (and all later too, but these are not in the drawing)

Because it is n-periodic, the dimension of H is only n, because there is always interference from before.

Circulant matrices and Discrete Fourier Transformation

Note that the coefficients $p_0, \ldots, p_{n-1}$ do **not** agree with the impulse response (→ Def. 4.1.3) of the filter. However, they can easily be deduced from it. In matrix notation (4.1.26) reads

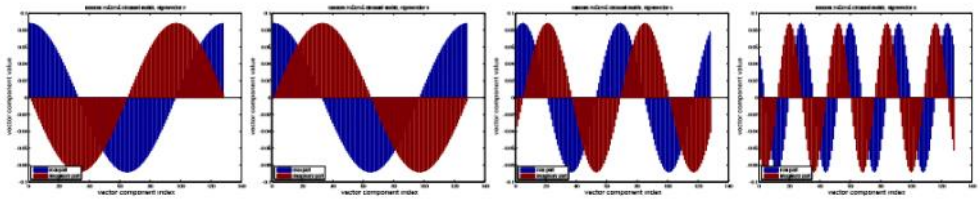
$$\begin{bmatrix} y_0 \\ \vdots \\ \\ \\ y_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} p_0 & p_{n-1} & p_{n-2} & \cdots & & \cdots & p_1 \\ p_1 & p_0 & p_{n-1} & & & & \vdots \\ p_2 & p_1 & p_0 & \ddots & & & \\ \vdots & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ \vdots & & & & \ddots & \ddots & p_{n-1} \\ p_{n-1} & \cdots & & & & p_1 & p_0 \end{bmatrix}}_{=:\mathbf{P}} \begin{bmatrix} x_0 \\ \vdots \\ \\ \\ x_{n-1} \end{bmatrix}. \qquad (4.1.27)$$

$$(\mathbf{P})_{ij} = p_{i-j}, 1 \le i,j \le n, \quad \text{with } p_j := p_{j+n} \text{ for } 1-n \le j < 0.$$

Definition 4.1.33. Circulant matrix → [8, Sect. 54]

A matrix $\mathbf{C} = (c_{ij})_{i,j=1}^{n} \in \mathbb{K}^{n,n}$ is circulant (*ger.:* zirkulant)
$:\Leftrightarrow \quad \exists\, (u_k)_{k\in\mathbb{Z}}$ n-periodic sequence: $c_{ij} = u_{j-i}, 1 \le i,j \le n.$

different random *circulant* matrices have the same eigenvectors.
Graph of larger circulant matrices' Eigenvectors:



The eigenvectors remind us of sampled *trigonometric functions* $\cos(k/n), \sin(k/n), k = 0, \ldots, n-1$!

All complex of course

Def. Root unity: $\omega_n := e^{\frac{-2\pi i}{n}} = \cos\left(\frac{2\pi}{n}\right) - i\sin\left(\frac{2\pi}{n}\right)$

These are equally spaced in the complex plane in a circle around (0,0)
$\omega_n^n = 1,$
$\boldsymbol{\omega_n^{-l}} = e^{\frac{2\pi i}{l}} = complex\ conjugate = \overline{\boldsymbol{\omega_n^l}}$

Random introduction of *Eigen*vector $\boldsymbol{v_k} = \left[\omega_n^{jk}\right]_{j=n}^{n-1} \in \mathbb{C}^n,\ 0 \le k < n$

Take any circulant Matrix C with dimension n, generated from vector u
$\Rightarrow row\ 1\ of\ C = (u_0 \quad u_{n-1} \quad u_{n-2} \quad \ldots \quad u_1)$ //like the p in the image
$Cv_k = \lambda_k v_k\ per\ definition$

Formula of periodic convolution $=> j-th\ line\ (C\ v_k)_j = \sum_{l=0}^{n-1} u_j \cdot (v_k)_{j-l}$
einsetzen der definition von $v_k \Rightarrow Cv_k = \omega_n^{kj} \sum_{l=0}^{n-1} u_l \omega_n^{-kl}$
$\Rightarrow \omega_n^{kj} = (v_k)_j = \lambda_k = Eigenwert\ von\ C\ and\ v_k\ is\ it's\ eigenvector$

orthogonal trigonometric basis of $\mathbb{C}^n$ = Eigenvector bassis for circulant matrices = vectors v_k defined as $\left[\omega_n^{jk}\right]_{j=n}^{n-1}$

Fourier-Matrix transforms this basis to the standard basis

The matrix effecting the change of basis trigonometrical basis → standard basis is called the Fourier-matrix

$$\mathbf{F}_n = \begin{bmatrix} \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \cdots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \cdots & \omega_n^{2n-2} \\ \vdots & \vdots & & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} = \left[\omega_n^{lj}\right]_{l,j=0}^{n-1} \in \mathbf{C}^{n,n}. \quad (4.2.13)$$

Lemma 4.2.14. Properties of Fourier matrix

The scaled Fourier-matrix $\frac{1}{\sqrt{n}}\mathbf{F}_n$ *is unitary (→ Def. 6.2.2) :* $\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^{\mathrm{H}} = \frac{1}{n}\overline{\mathbf{F}}_n$.

There is a Diagonalmatrix D such that

$$CF_n = F_n D$$

This is obvious if I look at it line by line, as F_n contains the Eigenvectors of C. These stay the same, but scaled, so only the diagonal will be set. It works because (grey arrows above) F_n has the same rows as columns.

$$D_{k,k} = \sum_{l=0}^{n-1} u_l \, \omega_n^{-kl}$$

additional Fun Fact: $\omega_n^{-kl} = \left(\overline{F}_n\right)_{k,l} \Rightarrow D = \overline{F}_n \vec{u}$

$\Rightarrow$ *all Circulant matrices have the same Eigenvalues*. Only some scaling determined by u, but as scaling an Eigenvector/value pair is always possible…

From the wikipedia article

> In numerical analysis, circulant matrices are important because they are diagonalized by a discrete Fourier transform, and hence linear equations that contain them may be quickly solved using a fast Fourier transform.[1]

[1]: Davis, Philip J., Circulant Matrices, Wiley, New York, 1970 ISBN 0471057711

http://www-ee.stanford.edu/~gray/toeplitz.pdf

In simple terms, why this holds is that:

1. A circulant matrix has all its rows being cyclic permutations (cyclic shifts) of the same row.
2. The fourier transform (DFT) is *circular*, meaning its basis is polynomials on the roots of unity which are invariant under cyclic shifts (on the unit circle).
3. Thus if expressed on the DFT basis, each row of the matrix is only a shift away from the reference (original) row. But a shift, in DFT terms, is simple multiplication by a root of unity, thus only the diagonal elements need be non-zero, to describe the appropriate shift (think of a clock, and shifting as changing the angle of the pointer, i.e simple multiplication by a root of unity)

Fourier Matrix contains the Eigenvectors AND the Eigenvalues of any Circulant Matrix

Eigenvectors $\Rightarrow$ *pairwise orthogonal*

$\Rightarrow$ *scale by the right factor and* F_n *is unitary*

$$v_k^H v_m = \begin{cases} n, & for\ m = k \\ 0, & for\ m \neq k \end{cases} \quad because \quad \sum_{l=0}^{n-1} \omega_n^{ml-kl} = 1 * n \quad something$$

"something":

$$\sum_{i=0}^{n-1} a^i = \frac{a^n - 1}{a - 1}$$

$$\text{With } a = \omega_n^{m+k} = \frac{\left(\omega_n^{m+k}\right)^n - 1}{\omega_n^{m+k} - 1} = \frac{1-1}{\omega_n^{m+k} - 1} = 0$$

//Geometrische Summenformel

Lemma 4.2.14. Properties of Fourier matrix

The scaled Fourier-matrix $\frac{1}{\sqrt{n}}\mathbf{F}_n$ *is unitary (→ Def. 6.2.2) :* $\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^{\mathrm{H}} = \frac{1}{n}\overline{\mathbf{F}}_n$.

Lemma 4.2.16. Diagonalization of circulant matrices (→ Def. 4.1.34)

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

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

//Why does that Lemma follow?

$\overline{F_n} = F_n^H$

Zeilen in Fouriermatrix = Spalten = Eigenvektoren von C

$C(\overline{F_n})_{Spalte\ i} = C(F_n)_{Zeile\ i^T} = Cv_i = v_i \cdot eigenvalue = v_i \cdot u_i$

$\Rightarrow Diagonalmatrix\ skaliert\ jeden\ Spaltenvektor\ im\ ergebnis\ um\ das\ i - te\ diagonalelement$

$$C\begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} = \cdots = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix}\begin{pmatrix} d_0 & 0 & 0 \\ 0 & d_1 & 0 \\ 0 & 0 & d_3 \end{pmatrix} = \left(d_0 \begin{bmatrix} a \\ b \\ c \end{bmatrix}, d_1 \begin{bmatrix} b \\ d \\ e \end{bmatrix}, d_2 \begin{bmatrix} c \\ e \\ f \end{bmatrix} \right)$$

$\Rightarrow Weil\ die\ eigenvektoren\ mit\ den\ Eigenwerten\ skaliert\ werden, gilt\ D$

$$= diag(eigenwerte) = \sum_{l=0}^{n-1} u_l \omega_n^{-kl}$$

From Lemma 4.2.16 follows that we can Calculate C and therefore Cx using Fourier:

$$Cx = \frac{1}{n}\overline{F_n} diag(d_1,..,d_n)\overline{F_n}^{-1} x$$

This is a periodic discrete convolution.

$= F_n^{-1}\ diag \cdot F_n\ x$

Therefore this operation has been given a special name:

Definition 4.2.18. Discrete Fourier transform (DFT)

The linear map $\mathcal{F}_n : \mathbb{C}^n \mapsto \mathbb{C}^n$, $\mathcal{F}_n(\mathbf{y}) := \mathbf{F}_n\mathbf{y}$, $\mathbf{y} \in \mathbb{C}^n$, is called discrete Fourier transform (DFT), i.e. for $\mathbf{c} := \mathcal{F}_n(\mathbf{y})$

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{kj} \quad , \quad k = 0,\ldots,n-1 . \qquad (4.2.19)$$

⇒ instead of having to calculate a big Circulant Matrix multiplication (height 2n-1), we can calculate a convolution by multiplying x with the fourier Matrix, scaling this with the transformation of u and then inverting the whole transformation again

This is nice because DFT ∈ $O(n^2)$ can be computed fast using FFT ∈ $O(n\log(n))$.

Zero Padding

We now have a function that takes u and x as argument and performs a convolution $y = C(u) \cdot x = \overline{F_n} \cdot (F_n\vec{u}) \cdot F_n \cdot x$

In the beginning we had a H-Matrix (Filter) which is always circulant so this works always. If we don't have periodicity, we can pad with zeros to remove interference:

$\vec{u}$ has size n

$C(u)\ has\ size\ n \times n\ for\ n - periodicity$

pad $u' \rightarrow size\ 2n-1$

$\Rightarrow C(u')\ has\ size\ (2n-1)\times(2n-1) \Rightarrow no\ interference$

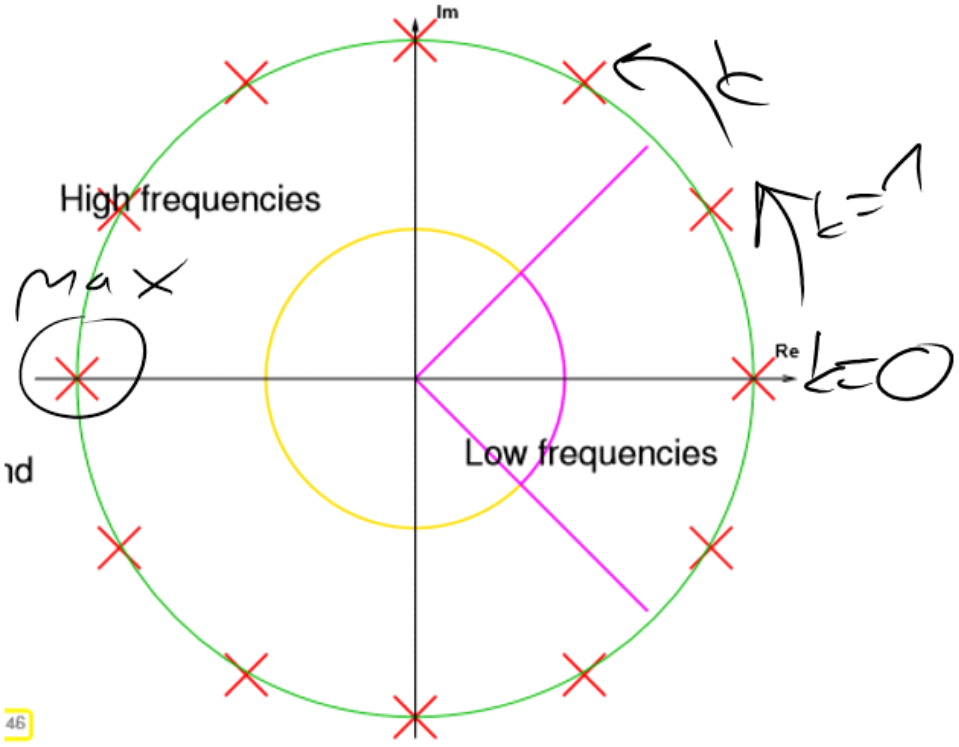
$x\ has\ to\ fit\ the\ second\ dimension\ of\ C \Rightarrow x\ has\ size\ (2n-1)$

Frequencies

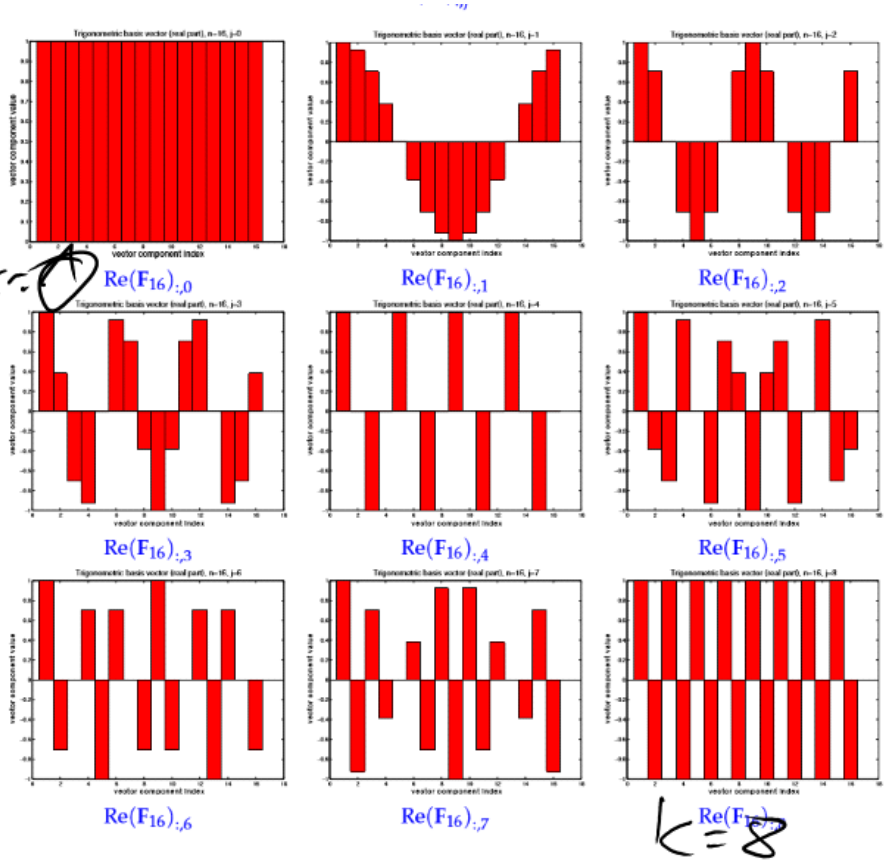
orthogonal Trigonometrical basis (Fourier basis) ⇔ oscillations

$$v_k = \left[\omega_n^{kj}\right]_{j=0}^{n-1} = \left[\cos\left(\frac{2\pi kj}{n}\right)\right]_{j=0}^{n-1} - i\left[\sin\left(\frac{2\pi kj}{n}\right)\right]_{j=0}^{n-1}$$

DFT = Additive decomposition of a signal into frequency contribution



Plots of real parts of Columns of Fourier matrix F_{16} (trigon. basis). k stands for the k-th column.



for high k, the frequency seems slow again (but in other direction, like with a stroboscope view)

DFT in 2D

Basis of 2D: because F is basis in 1D

$$V_{m,n} = \left\{ \left[\omega_m^{lv}\right]_{l=0}^{m-1} \left(\left[\omega_n^{kp}\right]_{k=0}^{n} \right)^T \right\}_{\substack{v=0,\dots,m-1 \\ p=0,\dots,n-1}}$$

$$\left(\mathcal{B}\left(\left(\omega_m^{vk}\omega_n^{\mu q}\right)_{k,q\in\mathbb{Z}} \right) \right)_{l,j} = \sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\omega_m^{v(l+k)}\omega_n^{\mu(j+q)} = \omega_m^{vl}\omega_n^{\mu j} \sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\omega_m^{vk}\omega_n^{\mu q}$$

V_{pv} ← eigenvector $= (V_{pv})_{q,j}$ eigenvalue λ_{pv}

is eigenvector of B because it stays there (definition of eigenvector and eigenvalues)
B is the Blurring operator.
DFT in 2D is defined like that V => same as just two DFTs after each other

(4.2.43) Matrix Fourier modes

A two-dimensional trigonometric basis of $\mathbb{C}^{m,n}$ is give by the tensor product matrices

$$\left\{ (\mathbf{F}_m)_{:,j}(\mathbf{F}_n)_{:,\ell}^{\mathsf{T}},\ 1 \le j \le m,\ 1 \le \ell \le n \right\} \subset \mathbb{C}^{m,n}. \tag{4.2.44}$$

Let a matrix $\mathbf{C} \in \mathbb{C}^{m,n}$ be given as a linear combination of these basis matrices with coefficients $y_{j_1,j_2} \in \mathbb{C}$, $0 \le j_1 < m$, $0 \le j_2 < n$:

$$\mathbf{C} = \sum_{j_1=0}^{m-1}\sum_{j_2=0}^{n-1} y_{j_1,j_2} (\mathbf{F}_m)_{:,j_1}(\mathbf{F}_n)_{:,j_2}^{\mathsf{T}}. \tag{4.2.45}$$

Then the entries of $\mathbf{C}$ can be computed by two nested discrete Fourier transforms:

$$(\mathbf{C})_{k_1,k_2} = \sum_{j_1=0}^{m-1}\sum_{j_2=0}^{n-1} y_{j_1,j_2}\omega_m^{j_1k_1}\omega_n^{j_2k_2} = \sum_{j_1=0}^{m-1}\omega_m^{j_1k_1}\left(\sum_{j_2=0}^{n-1}\omega_n^{j_2k_2}y_{j_1,j_2}\right), \quad 0 \le k_1 < m, 0 \le k_2 < n.$$

The coefficients can also be regarded as entries of a matrix $\mathbf{Y} \in \mathbb{C}^{m,n}$. Thus we can rewrite the above expressions: for all $0 \le k_1 < m$, $0 \le k_2 < n$

$$(\mathbf{C})_{k_1,k_2} = \sum_{j_1=0}^{m-1} (\mathbf{F}_n(\mathbf{Y})_{j_1,:})_{k_2}\omega_m^{j_1k_1} \quad \blacktriangleright \quad \mathbf{C} = \mathbf{F}_m(\mathbf{F}_n\mathbf{Y}^{\mathsf{T}})^{\mathsf{T}} = \mathbf{F}_m\mathbf{Y}\mathbf{F}_n. \tag{4.2.46}$$

=>

$C = F_m Y F_n$

$Inverse: Y = F_m^{-1} C F_n^{-1} = \frac{1}{mn}\overline{F_m} C \overline{F_n}$

//because scaled F is unitary. Remember: $\frac{1}{n}\overline{F}_n = F_n^{-1}$

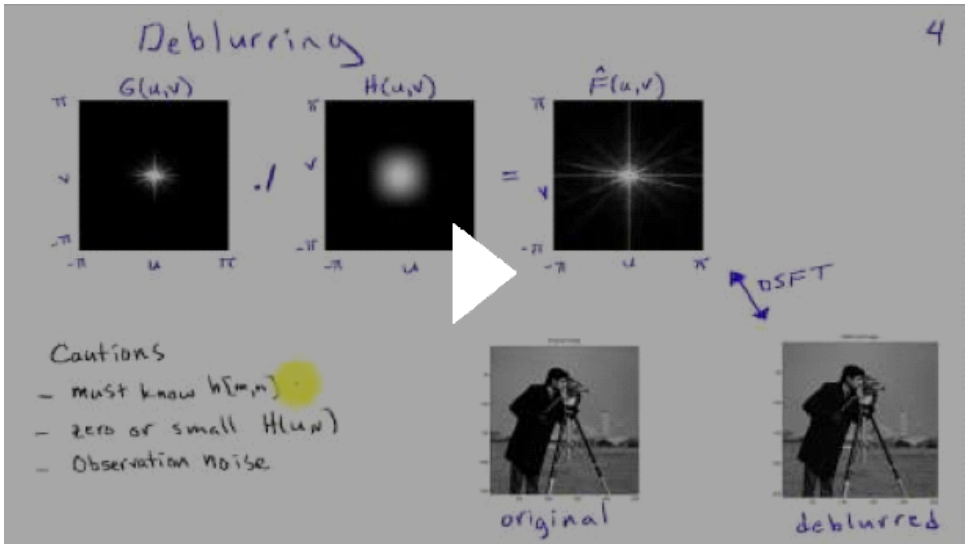
$$\mathbf{C} = \sum_{j_1=0}^{m-1}\sum_{j_2=0}^{n-1} y_{j_1,j_2}(\mathbf{F}_m)_{:,j_1}(\mathbf{F}_n)_{:,j_2}^{\mathsf{T}} \Rightarrow \mathbf{Y} = \mathbf{F}_m^{-1}\mathbf{C}\mathbf{F}_n^{-1} = \frac{1}{mn}\overline{\mathbf{F}}_m\mathbf{C}\overline{\mathbf{F}}_n. \tag{4.2.47}$$

Fourier basis coefficients pixel image

Probably Fehler in den Notes! C sollte nicht das pixel image sein, sondern das transformierte Bild. Y ist das Bild im Original

Why loop through rows instead of matrix mult? (Because the function fft was defined for vectors. these both work, because F is symmetric)
Is Fourier transform not from normal to fourier? -> Probably tablet notes wrong

Deblurring
https://www.youtube.com/watch?v=YhLF95crTWs



Blurring Operator B is given as pixelwise

Blurring = pixel values get replaced by weighted averages of near-by pixel values
(effect of distortion in optical transmission systems)

$$c_{l,j} = \sum_{k=-L}^{L} \sum_{q=-L}^{L} s_{k,q} p_{l+k,j+q}\,, \quad \begin{matrix} 0 \le l < m\,, \\ 0 \le j < n\,, \end{matrix} \quad L \in \{1, \ldots, \min\{m,n\}\}\,. \qquad (4.2.57)$$

$c_{l,j}$: blurred image; $p_{l+k,j+q}$: point spread function

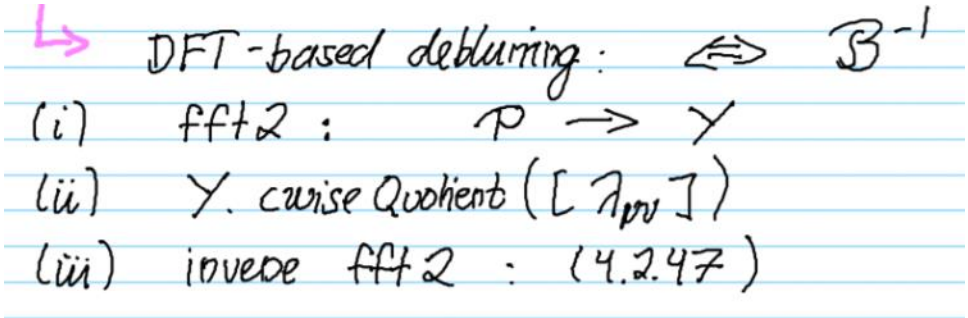
What's cool is that if we insert $\omega_n^{vk}\omega_m^{\mu q}$ for the point, we get the same thing again but scaled, so this is an eigenvector of B. Dividing $Bv = \lambda v$ by lambda gives that B is a simple cwise multiplication in Fourier space. So the solution for deblurring is to switch to Fourier space, divide cwise by s and then switch back.

$$\left(\mathcal{B}\left(\left(\omega_m^{vk}\omega_n^{\mu q}\right)_{k,q\in\mathbb{Z}}\right)\right)_{l,j} = \sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\omega_m^{v(l+k)}\omega_n^{\mu(j+q)} = \omega_m^{vl}\omega_n^{\mu j}\sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\omega_m^{vk}\omega_n^{\mu q}\,.$$

$\mathbf{V}_{v,\mu} := \left(\omega_m^{vk}\omega_n^{\mu q}\right)_{k,q\in\mathbb{Z}}$, $0 \le \mu < m$, $0 \le v < n$ are the eigenvectors of $\mathcal{B}$:

$$\mathcal{B}\mathbf{V}_{v,\mu} = \lambda_{v,\mu}\mathbf{V}_{v,\mu} \quad , \quad \text{eigenvalue} \quad \lambda_{v,\mu} = \underbrace{\sum_{k=-L}^{L}\sum_{q=-L}^{L} s_{k,q}\omega_m^{vk}\omega_n^{\mu q}}_{\text{2-dimensional DFT of point spread function !}}$$

Note: Inversion of blurring operator $\Leftrightarrow$ componentwise scaling in "Fourier domain"



Estimating PSF : $S_{k,q}$

$P_i \hat{=}$ test images

$C_i \hat{=}$ blurred images $\quad i = 1, \dots, m$

$$[S_{k,q}] := \underset{[r_{k,q}]}{\operatorname{argmin}} \sum_{i=1}^{m} \| \mathcal{B}([r_{k,q}]; P_i) - C_i \|_F^2$$

//Skript example code probably confused ifft and fft in the last line
//Actually, Hiptmair just replaced ifft and fft until it worked, the problem is that the point spread function has positive exponent instead of our usual negative exponents for fourier

Fast Fourier Transform

Divide Sums into a sum for the even and a sum for the odd indizes of the given Vector.

Recursion => $O(n \log(n))$

$for\ size\ of\ y\ called\ n = 2m$:

$$c_k = \sum_{j=0}^{n-1} y_j e^{-\frac{2\pi i}{n} jk}$$

$n = 2m$

$$= \sum_{j=0}^{m-1} y_{2j} e^{-\frac{2\pi i}{n} 2jk} + \sum_{j=0}^{m-1} y_{2j+1} e^{-\frac{2\pi i}{n}(2j+1)k}$$

$$= \sum_{j=0}^{m-1} y_{2j} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}} + e^{-\frac{2\pi i}{n} k} \cdot \sum_{j=0}^{m-1} y_{2k+1} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}}.$$

Toeplitz Matrix

Given the duration of the Impulse response: $n \leq m$

m-periodic Linear Time-invariant filter.

Measured y_k

known Input x

Sought: Estimate of the filters impulse response

$$\exists \vec{h} \in \mathbb{R}^n : y_k = \sum_{j=0}^{n-1} h_j x_{k-j}$$

measurement errors => use all available h (not just the first n)

$$\|\mathbf{Ah}-\mathbf{y}\|_2 = \left\| \begin{bmatrix} x_0 & x_{-1} & \cdots & & \cdots & x_{1-n} \\ x_1 & x_0 & x_{-1} & & & \vdots \\ \vdots & x_1 & x_0 & \ddots & & \\ & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & x_{-1} \\ x_{n-1} & & & & x_1 & x_0 \\ x_n & x_{n-1} & & & & x_1 \\ \vdots & & & & & \vdots \\ x_{m-1} & \cdots & & & \cdots & x_{m-n} \end{bmatrix} \begin{bmatrix} h_0 \\ \vdots \\ \\ \vdots \\ h_{n-1} \end{bmatrix} - \begin{bmatrix} y_0 \\ \vdots \\ \\ \\ \vdots \\ y_{m-1} \end{bmatrix} \right\|_2 \to \min .$$

➢ Linear least squares problem, → Chapter 3 with a coefficient matrix $\mathbf{A}$ that enjoys the property that $(\mathbf{A})_{ij} = x_{i-j}$ (constant entries of diagonals).

The coefficient matrix for the normal equations (→ **??**, Thm. 3.1.10) corresponding to the above linear least squares problem is

$$\mathbf{M} := \mathbf{A}^{\mathrm{H}}\mathbf{A} \quad , \quad (\mathbf{M})_{ij} = \sum_{k=1}^{m} x_{k-i}x_{k-j} = z_{i-j} \quad \text{due to periodicity of } (x_k)_{k\in\mathbb{Z}} .$$

➢ Again, $\mathbf{M} \in \mathbb{R}^{n,n}$ is a *matrix with constant diagonals* & s.p.d.
("constant diagonals" ⇔ $(\mathbf{M})_{i,j}$ depends only on $i-j$)

Normal equation $A^T Ax = A^T b$

$$A \text{ has constant Diagonals: } x_{i,j} = x_{i-j} \Rightarrow \left(A^T A\right)_{i,j} = \sum_k x_{i,k}x_{k,j}$$

$$= \sum_{k=0}^{m-1} x_{i-k}x_{k-j} \Rightarrow Data\ only\ dependent\ on\ Tuple\ (i,j)$$

constant diagonals => m+n-1 actual information content numbers ⇒ Toeplitz matrix can be displayed with a vector $u = (u_{-m+1}, \dots, u_{n-1})$

Extend to Circulant matrix of size 2m × 2n

The following formula demonstrates the structure of $\mathbf{C}$ in the case $m = n$.

$$\mathbf{C} = \left[\begin{array}{cccccc|cccccc} u_0 & u_1 & \cdots & & u_{n-1} & & 0 & u_{1-n} & \cdots & & u_{-1} \\ u_{-1} & u_0 & u_1 & & \vdots & & u_{n-1} & 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & & \vdots & \ddots & \ddots & & \\ \vdots & & \ddots & \ddots & u_1 & & \vdots & & \ddots & \ddots & u_{1-n} \\ u_{1-n} & \cdots & & u_{-1} & u_0 & & u_1 & & u_{n-1} & 0 & \\ 0 & u_{1-n} & \cdots & & u_{-1} & & u_0 & u_1 & \cdots & & u_{n-1} \\ u_{n-1} & 0 & \ddots & & \vdots & & u_{-1} & u_0 & u_1 & & \vdots \\ \vdots & \ddots & \ddots & & & & \vdots & \ddots & \ddots & \ddots & \vdots \\ & & & \ddots & & & \vdots & & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & u_{1-n} & & \vdots & & & \ddots & u_1 \\ u_1 & & & u_{n-1} & 0 & & u_{1-n} & \cdots & & u_{-1} & u_0 \end{array}\right]$$

Now instead of Tx, *we can calculate* $C\begin{pmatrix}\vec{x}\\0\end{pmatrix} = \begin{pmatrix} Tx \\ idgaf \end{pmatrix}$

This is like a convolution, so we can solve this using FFT.

Householder QR repetition

Householder matrix creates a rotation of the vector such that the result is just as long but a multiple of the first unit vector. If we construct a Matrix Q in such a way, that it turns the first column of a matrix into such a vector, then we can apply multiple of these Q until the Result R is upper triangular.

$Q_n \dots Q_1 A = R$

if the Q are orthogonal, then we can multiply on the left with their Hermitian Transposed to get

$A = Q_1^H \dots Q_n^H R$

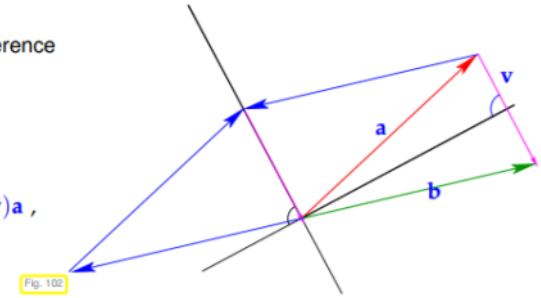
and these Q^H multiplied together are the orthogonal matrix Q.

To construct a Q, we choose a b that is parallel to a unit vector and use

Fig. 102 sketches a "geometric derivation" of Householder reflections:

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

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



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

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

//smart zero additions

$\Rightarrow Q_i = H(v) = I - \frac{2vv^H}{v^Hv}$

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

which is orthogonal, though I don't know why

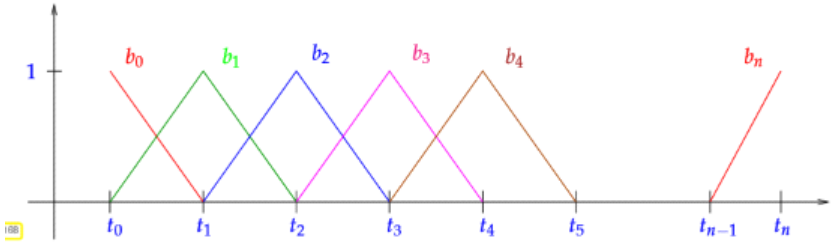
Interpolation in 1D

Piecewise Linear Interpolation

Connect measured points by lines.

for $Form\ ax = b$: $a = slope = \frac{y_i - y_{i-1}}{t_i - t_{i-1}}$, $b = where\ line\ cuts\ y$

adding up tent functions: they have to be not influencing any other points and add up to a line in between points.



height 1 because they can be scaled using coefficients

Interpolation as linear Mapping

$BaseFunctions * Coefficients = measurements$

$$\mathbf{Ac} := \begin{bmatrix} b_0(t_0) & \dots & b_m(t_0) \\ \vdots & & \vdots \\ b_0(t_n) & \dots & b_m(t_n) \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} =: \mathbf{y}\,.$$

=> need $m = n \Rightarrow Anzahl\ measurements = Anzahl\ base\ vectors$

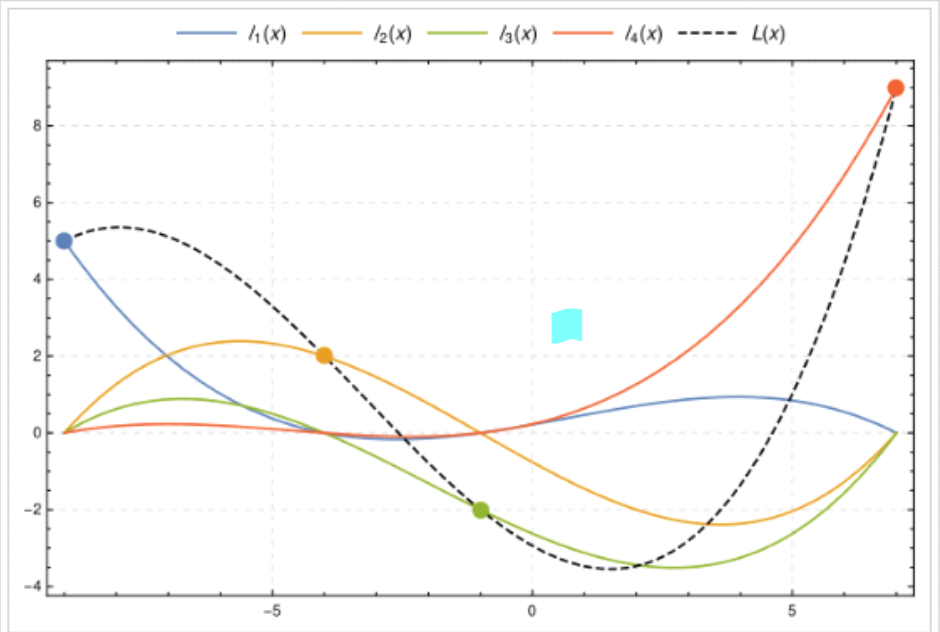
Polynomial Interpolation

Horner Scheme: Instead of calculating a polynomial $p(t)$ by calculating all powers of t, you can calculate it recursively

$$p(t) = t(\cdots t(t(\alpha_n t + \alpha_{n-1}) + \alpha_{n-2}) + \cdots + \alpha_1) + \alpha_0\,. \qquad (5.2.6$$

=> lineare zeit

Lagrange Polynomials as Cardinal Basis



This image shows, for four points ((−9, 5), (−4, 2), (−1, −2), (7, 9)), the (cubic) interpolation polynomial $L(x)$ (dashed, black), which is the sum of the *scaled* basis polynomials $y_0\ell_0(x)$, $y_1\ell_1(x)$, $y_2\ell_2(x)$ and $y_3\ell_3(x)$. The interpolation polynomial passes through all four control points, and each *scaled* basis polynomial passes through its respective control point and is 0 where x corresponds to the other three control points.

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

Lagrange polynomials $$L_i(t) := \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t-t_j}{t_i-t_j}, \quad i = 0,\ldots,n\,. \tag{5.2.11}$$

➞ Evidently, the Lagrange polynomials satisfy $L_i \in \mathcal{P}_n$ and $L_i(t_j) = \delta_{ij}$

and based on that Basis the Lagrange interpoland $p(t) = \sum_{i=0}^{n} y_i L_i(t)$ which is just the sum of all Lagrange polynomials. It fulfills $p(t_i) = y_i$.

fast by precomputing part of L_i

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

where $$\lambda_i = \frac{1}{(t_i-t_0)\cdots(t_i-t_{i-1})(t_i-t_{i+1})\cdots(t_i-t_n)}, i = 0,\ldots,n.$$

$\Rightarrow$ precompute

$\prod_{j=0}^{n}(t-t_j)$ *or the sum, and lambda. Calculate the other one from it*

Laufzeit: without precomputing: Sum n, product n, evaluating all t_s N => $O(Nn^2)$

with precomputing: $O(nN)$

From above formula, with $p(t) \equiv 1$, $y_i = 1$:

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

➞ Barycentric interpolation formula $$p(t) = \frac{\sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}\, y_i}{\sum_{i=0}^{n} \frac{\lambda_i}{t-t_i}}\,. \tag{5.2.28}$$

this works because we know that p has to be 1 where 1 is 1

partial Lagrange interpolant

Aitken-Neville scheme: $k < l$. ***Good for single evaluation***

$p_{k,l}$
$= unique\ polynomial\ of\ degree\ (l$
$- k)\ through\ the\ known\ points\ (t,y)_k \dots (t,y)_l$
First, set polynomials through just the k-th point:

$$p_{k,k}(x) = y_k$$

From these, derive interpolating polynomials through more points:

$$p_{k,l}(x) = \frac{1}{t_l - t_k}\Big((x - t_k)p_{k+1,l}(x) - (x - t_l)p_{k,l-1}(x)\Big)$$

so we weigh the polynomial in the interval to the right based on how much to the right x is, and the left polynomial based on how much to the left x is. (Assuming $t_k < x < t_l$, this will result in an addition. If x is not in the interval, then what?)
Dividing the whole thing to rescale it back to normal

Extrapolation to zero
same as interpolation but with x outside the interval.
Lagrangian
works well if function is even: $\phi(t) = \phi(-t)$ and $\phi\ behaves\ nicely\ around\ h$
Given: smooth function f in procedural form
Sought: approximation of f'
Idea: approx using difference quotient. but then we have cancellation.
Neville Aitken
Numerically stable alternative: symmetric (thus function even needed) difference quotient behaves like a polynomial for $2(n+1)$ times continuously diffable function. We use the fact that the differencequotient is approximable as a taylorpolynome and therefore with a polynome. Use neville-Aitken starting with small intervals from (x-h) to (x+h). The longer it takes, the larger the intervals and the better the approximation. Thus, the error is estimated by the difference between two last approximations.
Newton Basis (Update-friendly)

$$N_0(t) = 1, N_1(t) = (t - t_0), N_n(t) = \prod_{t=0}^{n}(t - t_i)$$

leading coefficient is 1 because else polynome wouldn't have the rank n
$\Rightarrow System\ of\ Linear\ equations$

$$a_0 N_0(t_j) + a_1 N_1(t_j) + \cdots + a_n N_n(t_j) = y_j, \qquad j = 0, \dots, n$$

n Equations. Solving with forward substitution

Our tool now: "update friendly" representation: Newton basis for $\mathcal{P}_n$

$$N_0(t) := 1\,, \quad N_1(t) := (t - t_0)\,, \quad \dots \quad , \quad N_n(t) := \prod_{i=0}^{n-1}(t - t_i)\,. \tag{5.2.49}$$

Note: $N_n \in \mathcal{P}_n$ with *leading coefficient* 1 ➢ linear indepdence ➢ basis property.

The abstract considerations of § 5.1.11 still apply and we get a linear system of equations for the coefficients a_j of the polynomial interpolant in Newton basis:

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

$\Leftrightarrow$ triangular linear system

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

=> same values computed again in each row. We can reuse them. If we add a point, we can just add a row and compute the latest a.

Newton basis polynomial $N_j(t)$: degree j and leading coefficient 1
$\Rightarrow$ a_j is the leading coefficient of the interpolating polynomial $p_{0,j}$.

(the notation $p_{\ell,m}$ for partial polynomial interpolants through the data points $(t_\ell, y_\ell), \ldots, (t_m, y_m)$ was introduced in Section 5.2.3.2, see (5.2.34))

apply recursion from aitken neville:

$$a_{\ell,m} = \frac{a_{\ell+1,m} - a_{\ell,m-1}}{t_m - t_\ell}\,. \qquad (5.2.51)$$

then we use the "divided differences" instead of solving the triangular linear system above.

$y(t_i) = y_i$

$$y(t_i, \ldots, t_{i+k}) = \frac{y(t_{i+1}, \ldots, t_{i+k}) - y(t_i, \ldots, t_{i+k-1})}{t_{i+k} - t_i}$$

$$\frac{f(y) - f(x)}{y - x} \approx f'(x) \text{ for } x \approx y$$

This approximation can be turned into an identity whenever Taylor's theorem applies.

$$f(y) = f(x) + f'(x) \cdot (y - x) + f''(x) \cdot \frac{(y-x)^2}{2!} + f'''(x) \cdot \frac{(y-x)^3}{3!} + \ldots$$

$$\Rightarrow \frac{f(y) - f(x)}{y - x} = f'(x) + f''(x) \cdot \frac{y - x}{2!} + f'''(x) \cdot \frac{(y-x)^2}{3!} + \ldots$$

Trigonometric interpolation

Reduction to lagrange

Das Problem wird einfacher, wenn wir es in der komplexen Ebene beschreiben. Wir können die Formel für ein trigonometrisches Polynom umschreiben zu

$$p(x) = \sum_{j=-n}^{n} c_j e^{ijx},$$

wobei i die imaginäre Einheit ist. Setzen wir $z = e^{ix}$, dann wird daraus

$$p(z) = \sum_{j=-n}^{n} c_j z^j.$$

construct sine from complex numbers

$$e^{it} = \cos t + i \sin t \quad \Rightarrow \quad \begin{cases} \cos t = \frac{1}{2}(e^{it} + e^{-it}) \\ \sin t = \frac{1}{2i}(e^{it} - e^{-it}) \end{cases}.$$

$$\Rightarrow q(t) = e^{-2\pi i n t} \cdot p\left(e^{2\pi i t}\right), \qquad p(z) = \sum_{j=0}^{2n} \gamma_j z^j$$

Don't really get this….

Equidistant trigonometric interpolation

Don't really get this…

Approximation by global Polynomials

By Taylor

$f \in C^{k+1}: f(t)$

$$\approx f(t_0) + f'(t_0)(t - t_0) + \frac{f''(t_0)}{2}(t - t_0)^2 + \cdots + \frac{f^k(t_0)}{k!}(t - t_0)^k$$

smoothness requirement!

Problems for Numerics:

- Stability
- Need derivatives to calculate. But we don't have the higher derivatives.

Bernstein

Theorem 6.1.6. Uniform approximation by polynomials

For $f \in C^0([0,1])$, define the n-th Bernstein approximant as

$$p_n(t) = \sum_{j=0}^{n} f(j/n)\binom{n}{j} t^j (1-t)^{n-j}, \quad p_n \in \mathcal{P}_n . \tag{6.1.7}$$

It satisfies $\|f - p_n\|_\infty \to 0$ for $n \to \infty$. If $f \in C^m([0,1])$, then even $\left\|f^{(k)} - p_n^{(k)}\right\|_\infty \to 0$ for $n \to \infty$ and all $0 \le k \le m$.

✎ Notation: $g^{(k)} \triangleq k$-th derivative of a function $g : I \subset \mathbb{R} \to \mathbb{K}$

Polynomial Best

Interval from -1 to 1

Theorem 6.1.11. L^∞ polynomial best approximation estimate

If $f \in C^r([-1,1])$ (r times continuously differentiable), $r \in \mathbb{N}$, then, for $n \ge r$,

$$\inf_{p \in \mathcal{P}_n} \|f - p\|_{L^\infty([-1,1])} \le (1 + \pi^2/2)^r \frac{(n-r)!}{n!} \left\|f^{(r)}\right\|_{L^\infty([-1,1])} .$$

r is the smoothness. Error Converges with rate r.

Transformation to Polynomial Best in interval

Assume that an interval $[a,b] \subset \mathbb{R}$, $a < b$, and a polynomial approximation scheme $\widehat{A} : C^0([-1,1]) \to \mathcal{P}_n$ are given. Based on the affine linear mapping

$$\Phi : [-1,1] \to [a,b] \quad , \quad \Phi(\hat{t}) := a + \tfrac{1}{2}(\hat{t}+1)(b-a) , \quad -1 \le \hat{t} \le 1 , \tag{6.1.15}$$

we can introduce the affine pullback of functions:

$$\Phi^* : C^0([a,b]) \to C^0([-1,1]) \quad , \quad \Phi^*(f)(\hat{t}) := f(\Phi(\hat{t})) , \quad -1 \le \hat{t} \le 1 . \tag{6.1.16}$$

Lemma 6.1.20. Transformation of norms under affine pullbacks

For every $f \in C^0([a,b])$ we have

$$\|f\|_{L^\infty([a,b])} = \|\Phi^* f\|_{L^\infty([-1,1])} \quad , \quad \|f\|_{L^2([a,b])} = \sqrt{|b-a|}\, \|\Phi^* f\|_{L^2([-1,1])} . \tag{6.1.21}$$

Error estimates

Theorem 6.1.37. Representation of interpolation error [4, Thm. 8.22], [7, Thm. 37.4]

We consider $f \in C^{n+1}(I)$ and the Lagrangian interpolation approximation scheme (→ Def. 6.1.25) for a node set $\mathcal{T} := \{t_0, \ldots, t_n\} \subset I$. Then, for every $t \in I$ there exists a $\tau_t \in]\min\{t, t_0, \ldots, t_n\}, \max\{t, t_0, \ldots, t_n\}[$ such that

$$f(t) - \mathsf{L}_\mathcal{T}(f)(t) = \frac{f^{(n+1)}(\tau_t)}{(n+1)!} \cdot \prod_{j=0}^{n} (t - t_j) . \tag{6.1.38}$$

Lagrange error

Thm. 6.1.44 $\Rightarrow$ $$\|f - \mathsf{L}_\mathcal{T} f\|_{L^\infty(I)} \le \frac{\|f^{(n+1)}\|_{L^\infty(I)}}{(n+1)!} \max_{t \in I} |(t - t_0) \cdot \cdots \cdot (t - t_n)| . \tag{6.1.50}$$

Chebychev

We cannot control f but we can play with the position of the t. We try to minimize $\max(t - t_0) \ldots \ldots (t - t_n)$

Are there polynomials satisfying these requirements? If so, do they allow a simple characterization?

Definition 6.1.75. Chebychev polynomials → [5, Ch. 32]

The n^{th} Chebychev polynomial is $T_n(t) := \cos(n \arccos t), \quad -1 \le t \le 1, n \in \mathbb{N}.$

The next result confirms that the T_n are polynomials, indeed.

Theorem 6.1.76. 3-term recursion for Chebychev polynomials → [5, (32.2)]

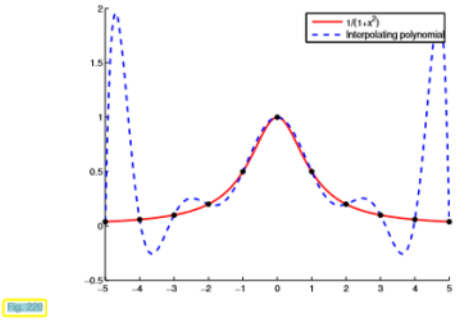
The function T_n defined in Def. 6.1.75 satisfy the 3-term recursion

$$T_{n+1}(t) = 2t\,T_n(t) - T_{n-1}(t) \quad , \quad T_0 \equiv 1\,, \quad T_1(t) = t\,, \quad n \in \mathbb{N}\,. \tag{6.1.77}$$

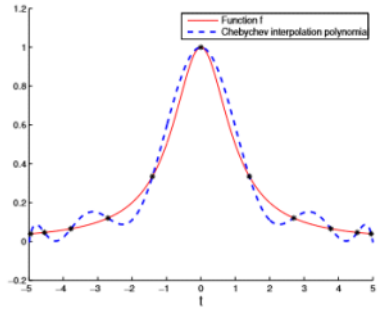
Proof. Just use the trigonometric identity $\cos(n+1)x = 2\cos nx \cos x - \cos(n-1)x$ with $\cos x = t$. □

Optimal because that yellow theorem 6.1.81
these clutter at the endpoints. That was the problem at the equidistant try. if we fix the endpoints better, we don't have weird oscillations there

We consider Runge's function $f(t) = \frac{1}{1+t^2}$, see Ex. 6.1.41, and compare polynomial interpolation based on uniformly spaced nodes and Chebychev nodes in terms of behavior of interpolants.



Equidistant nodes



Chebychev nodes

We observe that the Chebychev nodes cluster at the endpoints of the interval, which successfully suppresses the huge oscillations haunting equidistant interpolation there.

The Chebychev nodes in the interval $I = [a,b]$ are

$$t_k := a + \tfrac{1}{2}(b-a)\left(\cos\left(\frac{2k+1}{2(n+1)}\,\pi\right) + 1\right), \tag{6.1.86}$$

$$k = 0, \ldots, n\,.$$

b

lebesgue constant was also something here

seems important: Approximation by piecewise polynomials

Standard product rule:

By simple induction using the standard product rule $(fg)' = f'g + fg'$ one shows

$$h'(t) = \sum_{j=1}^{m} \left[g_j'(t) \cdot \prod_{\substack{k=1 \\ k \neq j}}^{m} g_k(t) \right] \tag{5.0.14}$$

Quadrature

To approximate integral by using a weighted sum of point values.

Definition 7.1.1. Quadrature formula/quadrature rule

An n-point quadrature formula/quadrature rule on $[a,b]$ provides an approximation of the value of an integral through a *weighted sum* of point values of the integrand:

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \sum_{j=1}^{n} w_j^n f(c_j^n)\,. \qquad (7.1.2)$$

In this definition you can essentially ignore the superscript n

Terminology:

w_j^n : quadrature weights $\in \mathbb{R}$

c_j^n : quadrature nodes $\in [a,b]$

cost of quadrature formula = n

quadrature formula is different depending on the Interval, but we can construct them based on other quadrature formulas

Given QF on [-1,1]

$\Phi\colon [-1,1] \to [a,b]$ $\phi(\mathrm{T}) = a + \frac{1}{2}(b-a)(\tau+1)$

//Verschieben, skalieren, an richtigen ort schieben

$$\Rightarrow \int_a^b f(t)\,dt = \int_{-1}^{1} f\big(\phi(\tau)\big)\left|\frac{d\phi}{d\tau}(\tau)\right| d\tau = \frac{1}{2}(b-a)\int_{-1}^{1} f\big(\phi(\tau)\big)\,d\tau$$

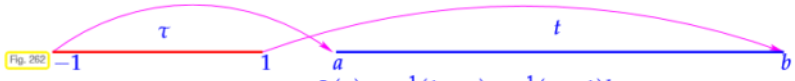
Integration durch substitution

$//\frac{dt}{d\tau} = \phi'$

Given: quadrature formula $(\widehat{c}_j, \widehat{w}_j)_{j=1}^n$ on reference interval $[-1,1]$

Idea: transformation formula for integrals

$$\int_a^b f(t)\,\mathrm{d}t = \tfrac{1}{2}(b-a)\int_{-1}^{1} \widehat{f}(\tau)\,\mathrm{d}\tau\,, \qquad (7.1.5)$$
$$\widehat{f}(\tau) := f(\tfrac{1}{2}(1-\tau)a + \tfrac{1}{2}(\tau+1)b)\,.$$

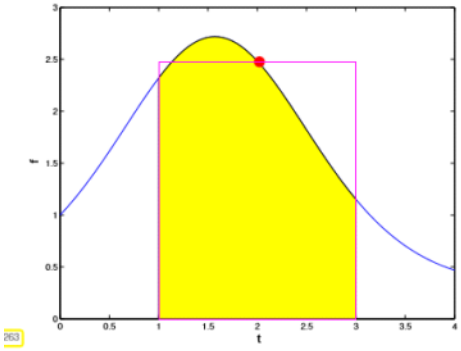


$$\tau \mapsto t := \Phi(\tau)t := \tfrac{1}{2}(1-\tau)a + \tfrac{1}{2}(\tau+1)b$$

Note that $\widehat{f}$ is the affine pullback $\Phi^* f$ of f to $[-1,1]$ as defined in Eq. (6.1.20).

quadrature formula for general interval $[a,b]$, $a,b \in \mathbb{R}$:

$$\int_a^b f(t)\,\mathrm{d}t \approx \tfrac{1}{2}(b-a)\sum_{j=1}^{n} \widehat{w}_j \widehat{f}(\widehat{c}_j) = \sum_{j=1}^{n} w_j f(c_j) \quad \text{with} \quad \begin{aligned} c_j &= \tfrac{1}{2}(1-\widehat{c}_j)a + \tfrac{1}{2}(1+\widehat{c}_j)b\,, \\ w_j &= \tfrac{1}{2}(b-a)\widehat{w}_j\,. \end{aligned}$$



The midpoint rule is (7.2.2) for $n = 1$ and $t_0 = \frac{1}{2}(a+b)$. It leads to the 1-point quadrature formula

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

"midpoint"

◁ the area under the graph of f is approximated by the area of a rectangle.

```
> trapez := newtoncotes(1);
```

$$\widehat{Q}_{\mathrm{trp}}(f) := \frac{1}{2}(f(0)+f(1)) \qquad (7.2.5)$$

$$\left(\int_a^b f(t)\,\mathrm{d}t \approx \frac{b-a}{2}(f(a)+f(b)) \right)$$

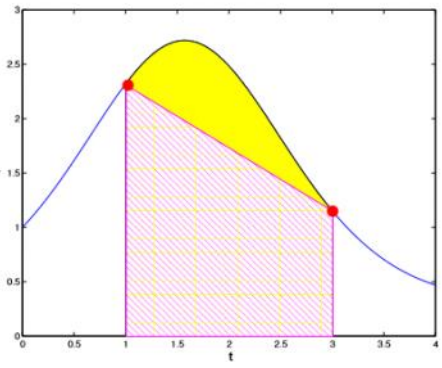


Fig. 264

Definition 7.3.1. Order of a quadrature rule

The order of quadrature rule $Q_n : C^0([a,b]) \to \mathbb{R}$ is defined as

$$\mathrm{order}(Q_n) := \max\{m \in \mathbb{N}_0 : \quad Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \quad \forall p \in \mathcal{P}_m\} + 1\,, \qquad (7.3.2)$$

that is, as the maximal degree $+1$ of polynomials for which the quadrature rule is guaranteed to be exact.

This is a convention: if it gives up to polynomial of degree 2, it has degree 3
Order is invariant under affine transformation.

Theorem 7.3.38. Quadrature error estimate for quadrature rules with positive weights

For every n-point quadrature rule Q_n as in (7.1.2) of order $q \in \mathbb{N}$ with weights $w_j \geq 0$, $j = 1,\ldots,n$ the quadrature error satisfies

$$E_n(f) := \left| \int_a^b f(t)\,dt - Q_n(f) \right| \leq 2|b-a| \underbrace{\inf_{p \in \mathcal{P}_{k-1}} \|f-p\|_{L^\infty([a,b])}}_{\text{best approximation error}} \quad \forall f \in C^0([a,b])\,. \quad (7.3.39)$$

Theorem 6.1.15. L^∞ polynomial best approximation estimate

If $f \in C^r([-1,1])$ (r times continuously differentiable), $r \in \mathbb{N}$, then, for any polynomial degree $n \geq r$,

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

Lemma 7.3.41. Quadrature error estimates for C^r-integrands

Under the assumptions of Thm. 7.3.38 and with the notations introduced there, we find for $f \in C^r([a,b])$, $r \in \mathbb{N}_0$, that the quadrature error $E_n(f)$ satisfies

$$\text{in the case } q \geq r: \quad E_n(f) \leq C q^{-r} |b-a|^{r+1} \left\| f^{(r)} \right\|_{L^\infty([a,b])}\,, \qquad (7.3.42)$$

$$\text{in the case } q < r: \quad E_n(f) \leq \frac{|b-a|^{q+1}}{q!} \left\| f^{(q)} \right\|_{L^\infty([a,b])}\,, \qquad (7.3.43)$$

with a constant $C > 0$ independent of n, f, and $[a,b]$.

Theorem 7.3.5. Sufficient order conditions for quadrature rules

An n-point quadrature rule on $[a,b]$ ($\to$ Def. 7.1.1)

$$Q_n(f) := \sum_{j=1}^n w_j f(t_j)\,, \quad f \in C^0([a,b])\,,$$

with nodes $t_j \in [a,b]$ and weights $w_j \in \mathbb{R}$, $j = 1,\ldots,n$, has order $\geq n$, if and only if

$$w_j = \int_a^b L_{j-1}(t)\,dt\,, \quad j = 1,\ldots,n\,,$$

where L_k, $k = 0,\ldots,n-1$, is the k-th Lagrange polynomial (5.2.11) associated with the ordered node set $\{t_1, t_2, \ldots, t_n\}$.

Smoothing integrands by transformation

$$\int_a^b \sqrt{t} f(t) dt \text{ with } f \in C^{\infty([0,b])}$$

non-smooth integrand : slow algebraic convergence of quad. error

Idea: substitution: $s = \sqrt{t} \Rightarrow \frac{ds}{dt} = \frac{1}{2\sqrt{t}} \Rightarrow dt = 2\sqrt{t}ds = 2sds$

$$\int_0^b \sqrt{t} f(t) dt = \int_0^{\sqrt{b}} s f(s^2) 2s ds$$

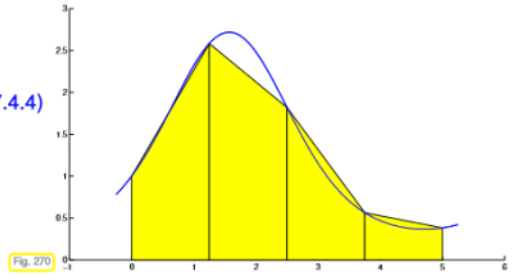
f was smooth, so the integrand here is now C^∞

Simpson rule, trapezoidal rule … for quadrature

Example 7.4.3 (Simple composite polynomial quadrature rules)

Composite trapezoidal rule, *cf.* (7.2.5)

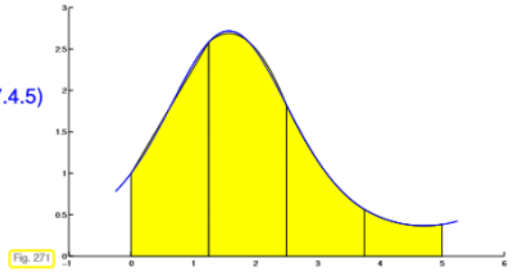
$$\int_a^b f(t)\mathrm{d}t = \tfrac{1}{2}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \tfrac{1}{2}(x_{j+1} - x_{j-1})f(x_j) + \tfrac{1}{2}(x_m - x_{m-1})f(b)\,. \qquad (7.4.4)$$



➢ arising from piecewise linear interpolation of f.

Composite Simpson rule, *cf.* (7.2.6)

$$\int_a^b f(t)\mathrm{d}t = \tfrac{1}{6}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \tfrac{1}{6}(x_{j+1} - x_{j-1})f(x_j) + \sum_{j=1}^{m} \tfrac{2}{3}(x_j - x_{j-1})f(\tfrac{1}{2}(x_j + x_{j-1})) + \tfrac{1}{6}(x_m - x_{m-1})f(b)\,. \qquad (7.4.5)$$



➢ related to piecewise quadratic Lagrangian interpolation.

Iterative methods

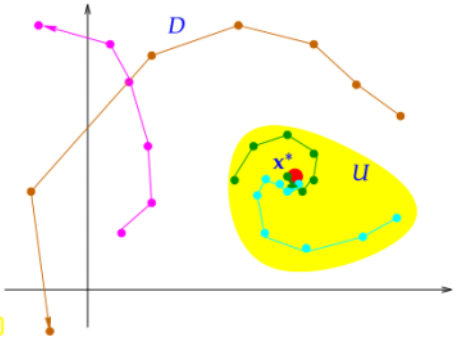
Spiralish -> limit is convergence

Only works locally

Illustration of local convergence ▷

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

Unfortunately, the neighborhood U is rarely known a priori. It may also be very small.



Definition 8.1.9. Linear convergence

A sequence $\mathbf{x}^{(k)}$, $k = 0, 1, 2, \ldots$, in $\mathbb{R}^n$ converges linearly to $\mathbf{x}^* \in \mathbb{R}^n$,

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

The next step converges as fast as the current step, duh.

Detecting order of convergence

Remark 8.1.19 (Detecting order of convergence)

How to guess the order of convergence (→ Def. 8.1.17) from tabulated error norms measured in a numerical experiment?

Abbreviate $\epsilon_k := \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|$ (norm of iteration error):

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

➢ monitor the quotients $(\log \epsilon_{k+1} - \log \epsilon_k)/(\log \epsilon_k - \log \epsilon_{k-1})$ over several steps of the iteration.

$Assumption: E_{k+1} = C \cdot E_k^p \Rightarrow \log(E_{k+1}) = \log(C) + \log(E_{k+1}^p)$
$= \log(C) + p \cdot \log(E_{k+1}^p)$
Subtract two such error equations => get p.

Recap Interpolations: B 9: Overview

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

Goal: build function $f \in C^1([t_0, t_n])$ satisfying the interpolation conditions $f(t_i) = y_i, i = 0, \ldots, n$.

Definition 5.4.1. Cubic Hermite polynomial interpolant

Given data points $(t_j, y_j) \in \mathbb{R} \times \mathbb{R}, j = 0, \ldots, n$, with pairwise distinct ordered nodes t_j, and slopes $c_j \in \mathbb{R}$, the piecewise cubic Hermite interpolant $s : [t_0, t_n] \to \mathbb{R}$ is defined by the requirements

$$s_{|[t_{i-1},t_i]} \in \mathcal{P}_3, \quad i = 1, \ldots, n \quad , \quad s(t_i) = y_i \quad , \quad s'(t_i) = c_i \quad , \quad i = 0, \ldots, n .$$

splines
derivative of the left side should be the same as on the right side (continuous).
$S_{[x_i,x_{i+1}]} = S_{[x_{i+1},x_{i+2}]}$

Data fitting

find f(x) such that the difference between y_i and $f(x_i)$ is minimal
Usually when you have more data points than unknowns in your function.
=> solve $\min \sum_i ||y_i - f(x_i)||$ with least squares

Termination

$$\left\|\mathbf{x}^* - \mathbf{x}^{(k)}\right\| \overset{k\to\infty}{\leq} \frac{L^{k-l}}{1-L}\left\|\mathbf{x}^{(l+1)} - \mathbf{x}^{(l)}\right\| . \qquad (8.2.21)$$

Set $l = 0$ in (8.2.21) → a priori termination criterion

$$\left\|\mathbf{x}^* - \mathbf{x}^{(k)}\right\| \leq \frac{L^k}{1-L}\left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\| \qquad (8.2.22)$$

Set $l = k - 1$ in (8.2.21) → a posteriori termination criterion

$$\left\|\mathbf{x}^* - \mathbf{x}^{(k)}\right\| \leq \frac{L}{1-L}\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\| \qquad (8.2.23)$$

Contractive Fixed point iteration

Definition 8.2.6. Contractive mapping

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

$$\exists L < 1: \quad \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in U . \qquad (8.2.7)$$

Theorem 8.2.9. Banach's fixed point theorem

If $D \subset \mathbb{K}^n$ ($\mathbb{K} = \mathbb{R}, \mathbb{C}$) closed and bounded and $\Phi : D \mapsto D$ satisfies

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

then there is a unique fixed point $\mathbf{x}^* \in D$, $\Phi(\mathbf{x}^*) = \mathbf{x}^*$, which is the limit of the sequence of iterates $\mathbf{x}^{(k+1)} := \Phi(x^{(k)})$ for any $\mathbf{x}^{(0)} \in D$.

Lemma 8.2.10. Sufficient condition for local linear convergence of fixed point iteration → **[5, Thm. 17.2], [3, Cor. 5.12]**

If $\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$, $\Phi(\mathbf{x}^*) = \mathbf{x}^*$, Φ differentiable in $\mathbf{x}^*$, and $\|D\Phi(\mathbf{x}^*)\| < 1$, then the fixed point iteration

$$\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)}) , \qquad (8.2.2)$$

converges locally and at least linearly. — *matrix norm, Def. 1.5.76 !*

Multipoint methods Model function M
Newton's method detailed expl.

Derivative of Matrix inversion

Application: Derivative of matrix inversion

Tool: Product rule for general derivatives

- Vector spaces: V, W, U, Z
- F: $V \to W$, $G: V \to U$ differentiable mappings
- b: W × U → Z bilinear (linear in each argument)
- T(x) := $b\big(F(x), G(x)\big)$, $T: V \to Z$

$$V = W = U = Z = R, \qquad b(\xi, \eta) = \xi \cdot \eta$$

$$T(x) = F(x) \cdot G(x), \qquad x \in \mathbb{R}$$
$$\Rightarrow T'(x) = F'(x) \cdot G(x) + F(x) \cdot G'(x) \in \mathbb{R}$$
$$T'(x)h = F'(x)h \cdot G(x) + F(x) \cdot (G'(x)h)$$

- For $F : V \mapsto W$ linear, we have $DF(\mathbf{x}) = F$ for all $\mathbf{x} \in V$
  (For instance, if $F : \mathbb{R}^n \to \mathbb{R}^m$, $F(\mathbf{x}) = \mathbf{A}\mathbf{x}$, $\mathbf{A} \in \mathbb{R}^{m,n}$, then $DF(\mathbf{x}) = \mathbf{A}$ for all $\mathbf{x} \in \mathbb{R}^n$.)
- Chain rule: For $F : V \mapsto W$, $G : W \mapsto U$ sufficiently smooth

$$D(G \circ F)(\mathbf{x})\mathbf{h} = DG(F(\mathbf{x}))(DF(\mathbf{x}))\mathbf{h} , \quad \mathbf{h} \in V, \mathbf{x} \in D . \qquad (8.4.8)$$

- Product rule: $F : D \subset V \mapsto W$, $G : D \subset V \mapsto U$ sufficiently smooth, $b : W \times U \mapsto Z$ bilinear, *ie.*, linear in each arguent:

$$T(\mathbf{x}) = b(F(\mathbf{x}), G(\mathbf{x})) \;\Rightarrow\; DT(\mathbf{x})\mathbf{h} = b(DF(\mathbf{x})\mathbf{h}, G(\mathbf{x})) + b(F(\mathbf{x}), DG(\mathbf{x})\mathbf{h}) , \qquad (8.4.9)$$
$$\mathbf{h} \in V, \mathbf{x} \in D .$$

$$Dinv(X)H = -X^{-1}HX^{-1} , \qquad H \in R^{n,n}$$

Termination: Newton's method detailed exp
Damping

❷ Newton correction is too large:

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

with zero $x^* = 0$.

If $x^{(k)}$ is located where the function is "flat", the intersection of the tangents with the x-axis is "far out", see Fig. 299.

Fig. 298

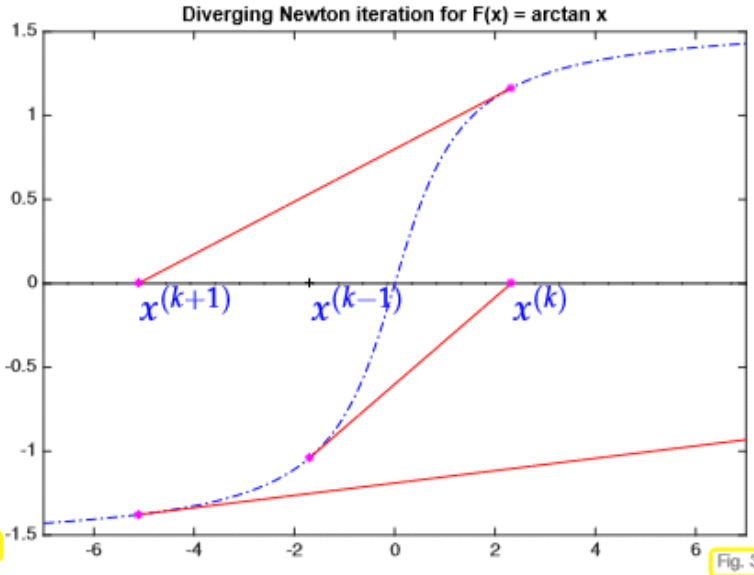


Fig. 299 Fig. 300

Solution approach: "damping". That means decreasing the size of the shift

Idea: damping of Newton correction:

With $\lambda^{(k)} > 0$: $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \lambda^{(k)}\, \mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$. (8.4.47)

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

λ *is between* 0 *and* 1

The art is to find a good damping factor

This is weird and maybe heuristics but it works amazingly if newton is lost otherwise with the **Affine invariant damping strategy**

Affine invariant damping strategy

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

choose "maximal" $0 < \lambda^{(k)} \le 1$: $\left\| \Delta\bar{\mathbf{x}}(\lambda^{(k)}) \right\| \le (1 - \frac{\lambda^{(k)}}{2}) \left\| \Delta\mathbf{x}^{(k)} \right\|_2$ (8.4.49)

where

$\Delta\mathbf{x}^{(k)} := \mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$ → current Newton correction ,

$\Delta\bar{\mathbf{x}}(\lambda^{(k)}) := \mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)} + \lambda^{(k)}\Delta\mathbf{x}^{(k)})$ → tentative simplified Newton correction .

If we correct, then next time it should be a bit smaller. This is what we should get if quadratic convergence had already set in

"Theory/assumption": If quadratic convergence => $\left|\left|\Delta\, \bar{x}(\lambda = 1)\right|\right| \le \frac{1}{2} \left|\left|\Delta \mathrm{x}^{(\mathrm{k})}\right|\right|$

If naturalMonotonicityTest passed, reduce damping aka increase lambda
if failed, increase damping (usually by factor of 2)

Fun Fact Summenformel

Mittwoch, 21. Dezember 2016 09:20

(unnötig aber yey)
geometrische Summenformel

Beweis (Geometrische Summenformel)

Es ist

$$
\begin{aligned}
& \sum_{k=0}^{n} q^k && = 1 + q + q^2 + \ldots + q^n \\
& && \downarrow \text{beide Seiten mit } q \text{ multiplizieren} \\
\Rightarrow \quad & q \cdot \sum_{k=0}^{n} q^k && = q + q^2 + q^3 + \ldots + q^{n+1} \\
& && \downarrow \text{zweite von erster Gleichung subtrahieren} \\
\Rightarrow \quad & \sum_{k=0}^{n} q^k - q \cdot \sum_{k=0}^{n} q^k && = (1 + q + q^2 + \ldots + q^n) - (q + q^2 + q^3 + \ldots + q^{n+1}) = 1 - q^{n+1} \\
& && \downarrow \sum_{k=0}^{n} q^k \text{ ausklammern} \\
\Rightarrow \quad & (1 - q) \cdot \sum_{k=0}^{n} q^k && = 1 - q^{n+1} \\
& && \downarrow \cdot \frac{1}{1-q} \\
\Rightarrow \quad & \sum_{k=0}^{n} q^k && = \frac{1 - q^{n+1}}{1 - q}
\end{aligned}
$$