

FS 2024

Güte?

Numerische Methoden
 VERSION 2V+1U(+2P) ITET+PHYS

Def algebraische Konvergenz $E(n) = O\left(\frac{1}{n^p}\right)$ mit $p > 0$
 $\sim \frac{1}{n^p}$ für n gross

§1 Quadratur 20.02.2024.

exponentielle Konvergenz
 $E(n) = O(q^n)$ mit $0 \leq q < 1$

§1.1. Grundlagen

Ben Riemann / Darboux Summe numerisch
 nicht verwendbar!

"gegeben" $f: [a, b] \rightarrow \mathbb{R} / \mathbb{C}$ "glatt"

Idee kompliziert einfach

Ziel $\int_a^b f(t) dt \approx \sum_{i=1}^n w_i f(\kappa_i) = Q_n(f, a, b)$
 $\swarrow \quad \downarrow \quad \searrow$
 Knoten $\kappa_i \in [a, b]$

$f \approx f_n$
 \mathbb{P}
 lin. Raum V
 ∞ -dim.
 $V_n = \text{span}\{b_1, \dots, b_n\}$
exakt

Quadraturformel Gewichte

f "bekannt" nur in $\kappa_1, \kappa_2, \dots, \kappa_n$

$$\int_a^b f(t) dt \approx \int_a^b f_n(t) dt = \sum_{i=1}^n \alpha_i \int_a^b b_i(t) dt$$

$$J_n = \alpha_1 b_1 + \alpha_2 b_2 + \dots + \alpha_n b_n$$

Möchte Fehler $E(n) = \left| \int_a^b f(t) dt - Q_n(f, a, b) \right| \rightarrow 0$
 "schnell", systematisch $n \rightarrow \infty$

Bsp $b_k(t) = t^k \Rightarrow$ Approximation mit Polynome.

oder

$b_2 = e^{ikx} = \cos kx + i \sin kx$ Fourier / Trigonometrische Polynome.

Ben es gibt bessere Methoden (Dokument Kap III, nicht besprechen)

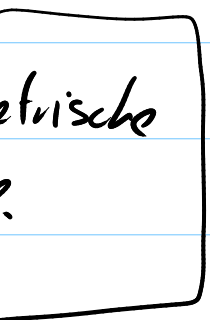
z.B. Polynomial Interpolation

Gegebene c_1, \dots, c_n

Suche $p(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \dots + \alpha_{n-1} t^{n-1}$

so dass $p(c_j) = f(c_j)$ für $j=0, 1, 2, \dots, n-1$

\Rightarrow LGS für $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$ mit $n \times n$ Vandermonde Matrix



Def Lagrange Polynome zu Knoten c_1, \dots, c_j

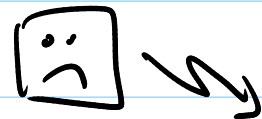
$l_j(t) = \prod_{\substack{i=1 \\ i \neq j}}^n \frac{t - c_i}{c_j - c_i}$ Polynom Grad $n-1$

$j = 1, 2, \dots, n$

$l_j(c_i) = \begin{cases} 0 & , i \neq j \\ 1 & , i = j \end{cases}$

\Rightarrow Basis in $\mathcal{P}_n = \{p \text{ Polynom von Grad } \leq n-1\}$

schlecht konditioniert (grosse Rundungsfehler)



poly fit

$f(t) = \sum_{j=1}^n f(c_j) l_j(t)$

Für Quadratur: $\int_a^b f(t) dt \approx \sum_{j=1}^n f(c_j) \underbrace{\int_a^b l_j(t) dt}_{w_j}$

- Ben 1) äquidistante c_1, \dots, c_n und D gross $\rightarrow \boxed{?}$ D.h.
- 2) es gibt optimale Verteilung von c_1, \dots, c_n
 [Kap III nicht besprochen]
- 3) Feste $c_1, c_2, \dots, c_n \Rightarrow$ nur w_1, w_2, \dots, w_n frei
 Dann kann man Polynome von Grad $\leq D-1$
- 4) wären $c_1, \dots, c_n, w_1, \dots, w_n$ frei wählbar
 $\rightarrow \mathbb{P}_{2D-1}$ Polynome von Grad $\leq 2D-1$

$$\int_a^b p(t) dt = Q(p, a, b)$$

für alle $p \in \mathbb{P}_n$ und Grad $n \leq 2$.
 $1, t, t^2, \dots, t^n$ exakt integriert
 $p \in \mathbb{P}_{n+1}$

Ben Man kann beweisen: (bis Grad $n-1$)

Wenn $f \in C^n[a, b]$ und QF exakt für alle in \mathbb{P}_n :

$$\left| \int_a^b f(t) dt - Q(f, a, b) \right| \leq \frac{1}{n!} (b-a)^{n+1} \max_{z \in [a, b]} |f^{(n)}(z)|$$

Länge des Intervalls \leftarrow \rightarrow Glattheit von \downarrow
 wichtig:

Zweck-Ziel: QF soll exakt sein für gewisses \mathbb{P}_n

Def Quadraturformel hat Ordnung $n+1$
 (Genauigkeitsgrad n)
 Wenn sie Polynome von Grad maximal n
 exakt integriert.

Idee Zerlege

$$\int_a^b f(t) dt = \sum_{k=0}^{N-1} \int_{x_k}^{x_{k+1}} f(t) dt$$

z.B. mit äquidistant $x_k = x_0 + kh$, $h = \frac{b-a}{N}$
 mit N gross $N = \frac{b-a}{h}$

und verwende die QF auf jedes $[x_k, x_{k+1}]$

⇒ zusammengesetzte Quadraturformel.

Fehler: $\left| \int_a^b f(t) dt - \sum_{k=0}^{N-1} Q(f, x_k, x_{k+1}) \right| \leq$

$$\leq \sum_{k=0}^{N-1} \left| \int_{x_k}^{x_{k+1}} f(t) dt - Q(f, x_k, x_{k+1}) \right| \leq$$

$$\leq \sum_{k=0}^{N-1} \frac{1}{n!} h^{n+1} \max_{z \in [x_k, x_{k+1}]} |f^{(n)}(z)|$$

$\leq \max_{z \in [a, b]} |f^{(n)}(z)| = C$

$$\leq C \cdot \frac{h^{n+1}}{n!} \sum_{k=0}^{N-1} 1 = C \cdot \frac{h^{n+1}}{n!} N = C \cdot \frac{h^{n+1}}{n!} \cdot \frac{b-a}{h} =$$

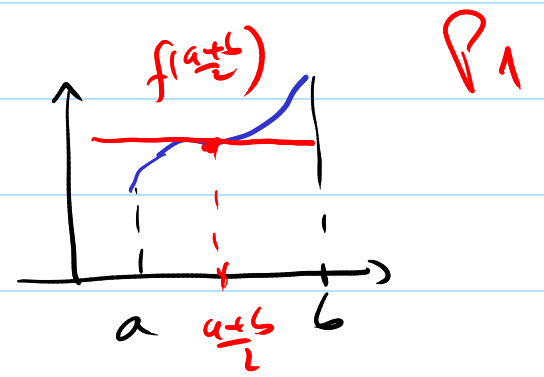
$$= C \cdot (b-a) \frac{h^n}{n!}$$

Idee

Wenn f glatt auf jedem $[x_k, x_{k+1}]$ verwende grosses n .

Bsp 1 Mittelpunktsregel.

(MPR) $Q(f, a, b) = (b-a) f\left(\frac{a+b}{2}\right)$



Beh 1) Polynome vom Grad 0 **und 1** werden durch (MPR) exakt integriert

(MPR) Q-Ordnung 2

$$\text{Fehler} \leq \frac{(b-a)^3}{3!} f''(z), z \in [a, b]$$

Zusammengefasst: $C \cdot h^2 \max_{x \in [a, b]} |f''(x)|$

$$\int_a^b f(t) dt \approx \sum_{k=0}^{N-1} h f(a + k \frac{h}{2}) = \frac{b-a}{N} \sum_{k=0}^{N-1} f(x_0 + k \frac{h}{2})$$

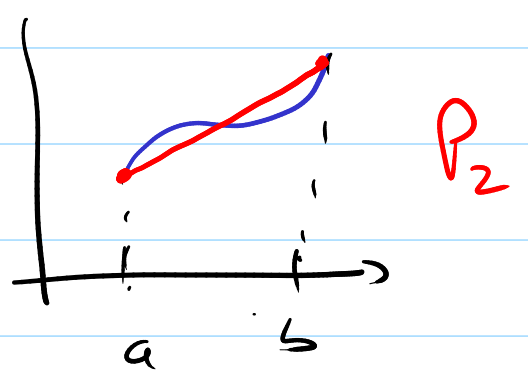
$$h = \frac{b-a}{N}$$

offene QF : Enden des Intervalls sind keine Knoten.

Knoten $c_1 = \frac{a+b}{2}$, Gewicht $w_1 = (b-a)$.

Bsp 2 Trapezregel

$$(TR) \quad \varphi(f, a, b) = \frac{b-a}{2} f(a) + \frac{b-a}{2} f(b)$$



$$c_1 = a, \quad c_2 = b, \quad w_1 = \frac{b-a}{2}, \quad w_2 = \frac{b-a}{2}$$

Q-Ordnung 2 ; lokaler Fehler $\frac{1}{2!} (b-a)^3 f''(\xi)$
 $\xi \in (a, b)$

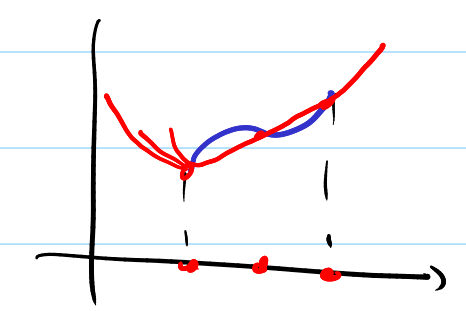
Zusammengesetzte (TR):
Fehler $O(h^2)$

$$\int_a^b f(x) dx \approx \frac{h}{2} f(x_0) + h \sum_{k=1}^{N-1} f(x_k) + \frac{h}{2} f(x_N)$$

geschlossene QF : beide Enden sind Knoten.

Bsp $f = f_n \in P_3$ exakt (Quadratische)
 Simpson Regel

$$(SR) \quad \varphi(f, a, b) = \frac{b-a}{6} f(a) + \frac{b-a}{6} 4 f(\frac{a+b}{2}) + \frac{b-a}{6} f(b)$$



$$c_1 = a, \quad c_2 = \frac{a+b}{2}, \quad c_3 = b$$

$$w_1 = \frac{b-a}{6}, \quad w_2 = \frac{b-a}{6} 4, \quad w_3 = \frac{b-a}{6}$$

Bei Polyn. bis Grad 2 **und Grad 3** exakt integrieren.

Q-Ordnung 4 \Rightarrow

$$\text{Idealer Fehler} \cdot c \cdot \left(\frac{b-a}{2}\right)^5 f^{(4)}(\xi), \xi \in (a,b)$$

Zusammengesetz: $O(h^4)$.

Den Normalerweise QF auf ein Referenzintervall!

$$t = \frac{1}{2}(1-z)a + \frac{1}{2}(1+z)b$$

$$t = (1-y)a + yb$$

$$dt = (b-a)dy$$

$$dt = \frac{b-a}{2} dz$$

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

$$\hat{f}(z) = f\left(\frac{1}{2}(1-z)a + \frac{1}{2}(1+z)b\right)$$

$$\text{(TR) auf } [0,1]: \quad c_1=0, \quad c_2=1$$

$$w_1=\frac{1}{2}, \quad w_2=\frac{1}{2}$$

$$\text{(TR) auf } [-1,1]: \quad c_1=-1, \quad c_2=1$$

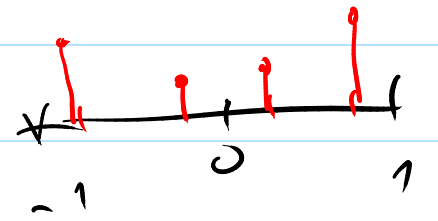
$$w_1=1, \quad w_2=1$$

Def

QF auf $[-1,1]$ heißt symmetrisch.

$$\text{falls } c_k = -c_{n+1-k}$$

$$w_k = w_{n+1-k}$$



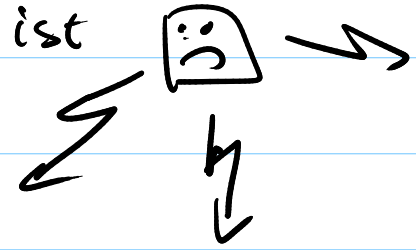
Bsp: MPR, Simpson sind symmetrisch.

T = Q-Ordnung einer sym. QF ist gerade

$$\left(\begin{array}{l} \Rightarrow \text{(MPR) Ordnung 2} \\ \text{(SR) Ordnung 4} \end{array} \right)$$

Bem grösseres n bei äquidistante Knoten.

ist



Leffe nicht!

Tschebyschöff | Chebyshev

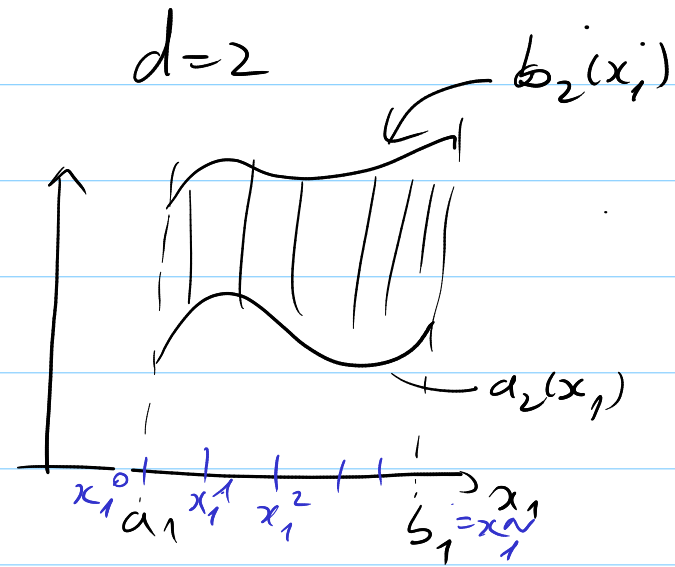
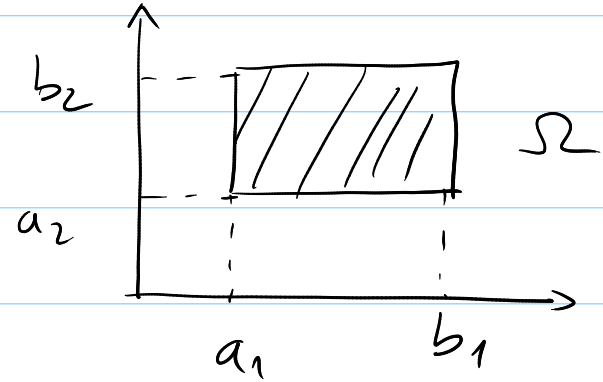
Theorem $f \in C^n([a, b])$, QF mit Q -Ordnung n :

$$1) \left| \int_a^b f(x) dx - Q(f, a, b) \right| \leq c (b-a)^{n+1} \max_{x \in [a, b]} |f^{(n)}(x)|$$

$$2) \left| \int_a^b f(x) dx - \sum_{k=0}^{n-1} h \sum_{j=1}^n b_j f(x_k + c_j h) \right| \leq$$

$$\leq c \cdot (b-a) h^n \max_{x \in [a, b]} |f^{(n)}(x)|$$

§1.2 Quadratur in \mathbb{R}^d



$$I = \int_{\Omega} f(x_1, x_2) dx_1 dx_2 = \int_{a_1}^{b_1} \int_{a_2(x_1)}^{b_2(x_1)} f(x_1, x_2) dx_2 dx_1$$

$$= \int_{a_1}^{b_1} F(x_1) dx_1 \approx \sum_{k_1=1}^{N_1} \int_{x_1^{k_1-1}}^{x_1^{k_1}} F(x_1) dx_1 =$$

$$\underbrace{\frac{b_1 - a_1}{N_1}}_{h_1} \sum_{k_1=1}^{N_1} \sum_{j_1=1}^{M_1} F(x_1^{k_1-1} + h_1 c_{j_1}^1) b_{j_1}$$

\uparrow Knoten \uparrow Gewichte auf Referenzintervall $(0,1)$

$$= h_1 \sum_{k_1=1}^{N_1} \sum_{j_1=1}^{M_1} b_{j_1} \int_{a_2(x_1^{k_1-1} + h_1 c_{j_1}^1)}^{b_2(x_1^{k_1-1} + h_1 c_{j_1}^1)} f(x_1^{k_1-1} + h_1 c_{j_1}^1, x_2) dx_2$$

= zusammengesetzte QF

$$= h_1 \sum_{k_1=1}^{N_1} \sum_{j_1=1}^{M_1} b_{j_1} h_2 \sum_{k_2=1}^{N_2} \sum_{j_2=1}^{M_2} b_{j_2} f(x_1^{k_1-1} + h_1 c_{j_1}^1, x_2^{k_2-1} + h_2 c_{j_2}^2) =$$

$$= h_1 h_2 \sum_{k_1=1}^{N_1} \sum_{k_2=1}^{N_2} \sum_{j_1=1}^{M_1} \sum_{j_2=1}^{M_2} b_{j_1} b_{j_2} f(x_1^{k_1-1} + h_1 c_{j_1}^1, x_2^{k_2-1} + h_2 c_{j_2}^2)$$

N_1, M_1, N_2, M_2 Knoten.

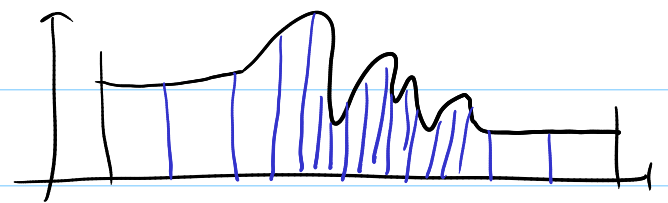
In d -Dimension $N_1, M_1, N_2, M_2, \dots, N_d, M_d$

$$N_1 = N_2 = \dots = N_d, \quad M_1 = M_2 = \dots = M_d$$

$\Rightarrow N^d, M^d$ zu teuer! ☹

§1.3 Adaptive Quadratur

Ziel: Optimiere die Anzahl der Funktionsauswertungen



Wie kann man beim Laufzeit des Algorithmus herausfinden, wo man mehr/wenigere Intervalle braucht?
Wie kann man das lokale Fehler beim Rechnen schätzen?

$$\text{lokale Fehler } \epsilon_k \sim h^{n+1} \max_{x \in [x_k, x_{k+1}]} |f^{(n+1)}(x)| \quad \text{2-Ordy } n$$

z.B. (TR):

$$\int_{x_k}^{x_{k+1}} f(x) dx = Q^T(f, x_k, x_{k+1}) + c h^3 \max_{x \in [x_k, x_{k+1}]} |f'''(x)|$$

$$(SR) \int_{x_k}^{x_{k+1}} f(x) dx = Q^S(f, x_k, x_{k+1}) + c \cdot h^5 \max_{x \in [x_k, x_{k+1}]} |f^{(5)}(x)|$$

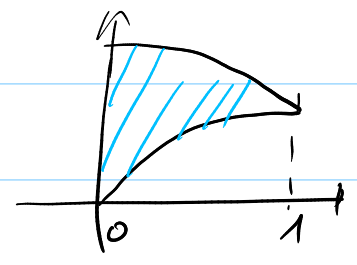
```
In [13]: from scipy import integrate
from numpy import sqrt
f = lambda x: 1/sqrt(abs(x))
integrate.quad(f, -1, 1)

<ipython-input-13-61ff175c15de>:2: RuntimeWarning: divide by zero encountered in double_scalars
f = lambda x: 1/sqrt(abs(x))
<ipython-input-13-61ff175c15de>:3: IntegrationWarning: The maximum number of subdivisions (50) has been achieved.
If increasing the limit yields no improvement it is advised to analyze the integrand in order to determine the difficulties. If the position of a local difficulty can be determined (singularity, discontinuity) one will probably gain from splitting up the interval and calling the integrator on the subranges. Perhaps a special-purpose integrator should be used.
integrate.quad(f, -1, 1)
```

Out[13]: (nan, nan)

```
In [14]: integrate.quad(f, -1, 1, points=[0]) # can deal with singularity if you tell it
```

Out[14]: (3.999999999999813, 5.684341886080802e-14)



$$a_1 = 0, \quad b_1 = 1$$

$$a_2(x) = \sin(x), \quad b_2(x) = 0.1 + \cos\left(\frac{x}{2}\right)$$

```
integrate.dblquad(f, 0, 1, a2, b2)
```

$$\epsilon_k = \left| \int_{x_k}^{x_{k+1}} f(x) dx - Q^T \right|$$

$Q^S \leftarrow$ approximiere \int mit einer besser-
 Q^S **IDEA!** Q^T Formel.

\Rightarrow erhalte eine Schätzung von $\epsilon_k \approx \tilde{\epsilon}_k$

$$\tilde{\epsilon}_k = |Q^S(f, x_k, x_{k+1}) - Q^T(f, x_k, x_{k+1})|$$

Idee: verfeinere dort wo $\tilde{\epsilon}_k$ gross ist

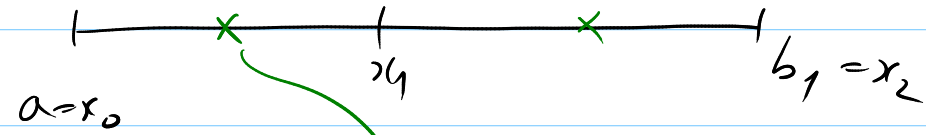
(vergrößere dort wo $\tilde{\epsilon}_k$ sehr klein ist)

verwende das Ergebnis von Q^S

Q^T wird verwendet nur um die Schätzung
 von lokalem Fehler zu berechnen.

Das kann man immer machen, sobald wir

2 Q -Formel verschiedener Ordnung.

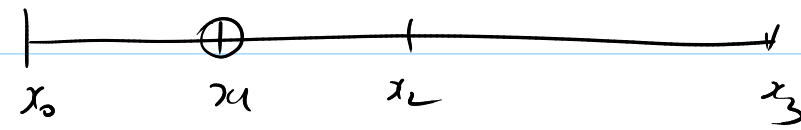


$$Q^T(f, x_0, x_1)$$

$$Q^S(f, x_0, x_1)$$

$$Q^S(f, x_1, x_2)$$

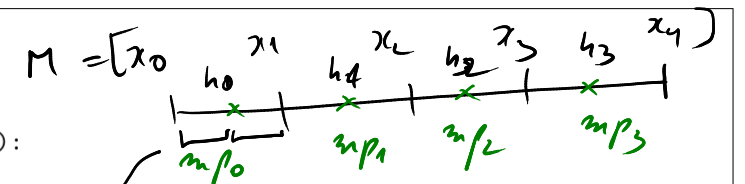
$$\epsilon_0 = |Q^S(f, x_0, x_1) - Q^T(f, x_0, x_1)| \text{ gross} \Rightarrow \text{verfeinere } [x_0, x_1]$$




```

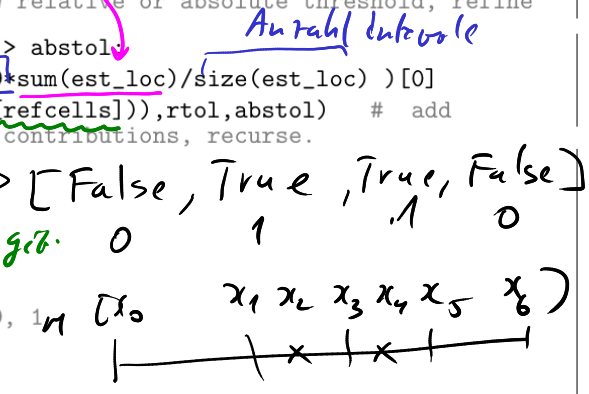
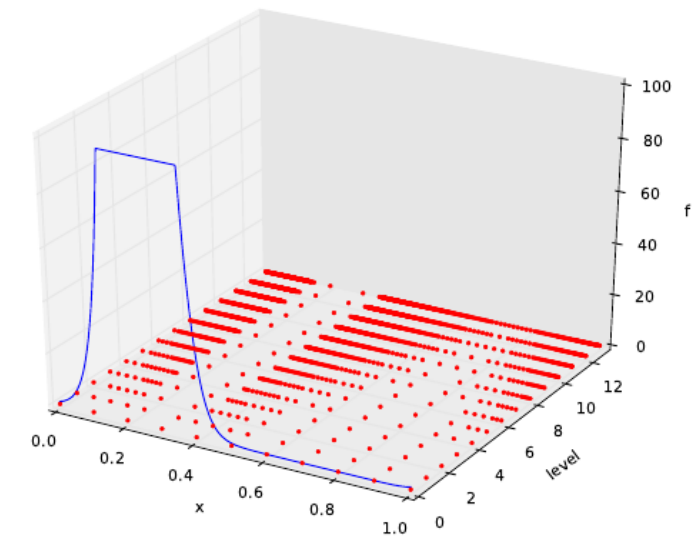
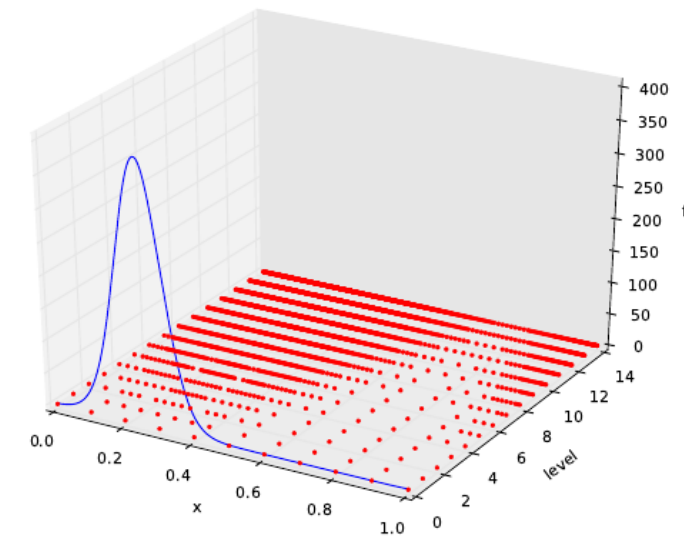
1 from numpy import *
2 from scipy import integrate
3
4 def adaptquad(f,M,rtol,abstol):
5     """
6     adaptive quadrature using trapezoid and simpson rules
7     Arguments:
8     f      handle to function f
9     M      initial mesh
10    rtol   relative tolerance for termination
11    abstol absolute tolerance for termination, necessary in case the exact
12    integral value = 0, which renders a relative tolerance meaningless.
13    """
14    h = diff(M) # compute lengths of mesh intervals
15    mp = 0.5*( M[:-1]+M[1:] ) # compute midpoint positions
16    fx = f(M); fm = f(mp) # evaluate function at positions and
17    midpoints
18    trp_loc = h*( fx[:-1]+2*fm+fx[1:] )/4 # local trapezoid rule
19    simp_loc = h*( fx[:-1]+4*fm+fx[1:] )/6 # local simpson rule
20    I = sum(simp_loc) # use simpson rule value as
21    intermediate approximation for integral value
22    est_loc = abs(simp_loc - trp_loc) # difference of values obtained from
23    local composite trapezoidal rule and local simpson rule is used as an estimate
24    for the local quadrature error.
25    err_tot = sum(est_loc) # estimate for global error (sum
26    moduli of local error contributions)
27    # if estimated total error not below relative or absolute threshold, refine
28    mesh
29    if err_tot > rtol*abs(I) and err_tot > abstol:
30        refcells = nonzero( est_loc > 0.9*sum(est_loc)/size(est_loc) )[0]
31        I = adaptquad(f,sort(append(M,mp[refcells])),rtol,abstol) # add
32        midpoints of intervals with large error contributions, recurse.
33    return I
34
35 if __name__ == '__main__':
36     f = lambda x: exp(6*sin(2*pi*x))
37     #f = lambda x: 1.0/(1e-4+x*x)
38     M = arange(11.)/10 # 0, 0.1, ... 0.9, 1
39     rtol = 1e-6; abstol = 1e-10
40     I = adaptquad(f,M,rtol,abstol)
41     exact,e = integrate.quad(f,M[0],M[-1])
42     print('adaptquad:',I, "exact":',exact)
43     print('error:',abs(I-exact))

```



$f(x) = e^{6 \sin(2\pi x)}$

$\min(f(x), 100)$



§ 1.4. Gauss-Quadratur

Möchte Knoten c_1, c_2, \dots, c_s und Gewichte b_1, \dots, b_s so dass

Q-Ordnung $p = s + m$, $m \in \mathbb{N}$, $m > 0$ maximal
(d.h. jedes Polynom vom Grad $p \leq s + m - 1$ soll exakt integriert werden)

Idee Nehme $M(t) = (t - c_1)(t - c_2) \dots (t - c_s)$
Grad $M = s$, $M(c_1) = 0 \dots = M(c_s) = 0$

Sei $f \in P_{s+m}$ (grad $f \leq s + m - 1$)
Polynom division.

$$f(t) = M(t)g(t) + r(t) \text{ mit}$$
$$\text{Grad } r \leq s - 1$$
$$\text{Grad } g \leq m - 1$$

Referenzintervall $[0, 1]$, $\int_0^1 dt = 1$

$$\int_0^1 f(t) dt = \int_0^1 M(t)g(t) dt + \int_0^1 r(t) dt$$

$\parallel \leftarrow$ QF exakt

QF exakt \parallel

$$\sum_{j=1}^s b_j f(c_j) = \sum_{j=1}^s b_j M(c_j) g(c_j) + \sum_{j=1}^s b_j r(c_j)$$

\parallel

$$\Rightarrow \int_0^1 M(t)g(t) dt = 0 \Leftrightarrow M \perp g \text{ für alle } g \in P_m$$
$$\langle M, g \rangle = \int_0^1 M(t)g(t) dt \text{ Skalarprodukt}$$
$$\Leftrightarrow M \perp P_m$$

Theorem Q-Ordnung ist $s + m \Leftrightarrow M \perp P_m$
Theorem Q-Ordnung ist max. $2s$ (d.h. $m \leq s$)

Beweis R.A. Q-Ordnung ist $2s + 1$ (d.h. $m = s + 1$)

$$\langle M, g \rangle = 0 \text{ für alle } g \in P_{s+1} \text{ (d.h. Grad } g \leq s)$$

nehme $g = M$ $\Rightarrow \langle M, M \rangle = 0 \Rightarrow M = 0$
 \searrow
Grad $M = s$

Somit Q-Ordnung $\leq 2s$.

Bem Verallgemeinern mit Gewichtsfunktion in Integri!

$$\int_a^b f(t) dt \text{ mit } \int_a^b f(t) w(t) dt$$

w stetig
 $w > 0$, $\int_a^b |x|^k w(x) dx < \infty$

$\int_0^1 f(t) \cdot g(t) dt$ ersetzt durch $\int_a^b f(t) g(t) w(t) dt$

Für Integrale der Form

$$\int_a^b f(x) \omega(x) dx$$

Ben Gram-Schmidt $\Rightarrow P_0, P_1, P_2, \dots$

$$P_k(t) = t^k + \text{Polyn. von Grad} \leq k-1$$

$P_k \perp \text{span} \{P_0, P_1, \dots, P_{k-1}\}$

GS \Rightarrow 3-Term-Rekurrenz für die P_k .

spielen verschiedene orthogonale Polynome eine wesentliche Rolle:

Quadratur	Intervall	Gewichtsfunktion	Polynom	Not.	scipy.special
Gauss	(-1, 1)	1	Legendre	P_k	roots_legendre
Chebyshev I	(-1, 1)	$\frac{1}{\sqrt{1-x^2}}$	Chebyshev I	T_k	roots_chebyt
Chebyshev II	(-1, 1)	$\sqrt{1-x^2}$	Chebyshev II	U_k	roots_chebyu
Jacobi $\alpha, \beta > 1$	(-1, 1)	$(1-x)^\alpha (1+x)^\beta$	Jacobi	$P_k^{(\alpha, \beta)}$	roots_jacobi
Hermite	\mathbb{R}	e^{-x^2}	Hermite	H_k	roots_hermite
Laguerre	(0, ∞)	$x^\alpha e^{-x}$	Laguerre	L_k	roots_genlaguerre

Abb. 1.5.10. Gewichtsfunktionen für Quadraturformeln

Ben $c_1, \dots, c_n =$ Nullstellen P_n

Ben $[0, 1]$, $w \equiv 1 \Rightarrow$ Legendre Polynome \Rightarrow Gauss-Legendre Quadratur

$$\text{auf } (0, 1): c_1 = \frac{1}{2} - \frac{\sqrt{2}}{6}, c_2 = \frac{1}{2} + \frac{\sqrt{2}}{6}$$

$$b_1 = \frac{1}{2}, b_2 = \frac{1}{2}$$

\mathbb{R} , $w(t) = e^{-t^2} \Rightarrow$ Hermite Polynome \Rightarrow Gauss-Hermite Quadratur

Ordnung $2n = 2 \cdot 2 = 4 \Rightarrow$ exakt bis Polynom von Grad ≤ 3

analog: Laguerre, Jacobi, Tschebyschöff I, II
 \Rightarrow Quadratur

3) $n=3$ auf $(-1, 1)$

$$P_3(t) = \frac{5}{2} t^3 - \frac{3}{2} t$$

$$c_1 = -\frac{\sqrt{15}}{5}, c_2 = 0, c_3 = \frac{\sqrt{15}}{5}$$

$$b_1 = \frac{8}{9}, b_2 = \frac{8}{9}, b_3 = \frac{1}{9}$$

Ordnung $2n = 2 \cdot 3 = 6$

Bsp Gauss-Quadratur

1) $n=1$ auf $(0, 1) \Rightarrow P_1(t) = t - \frac{1}{2} \Rightarrow c_1 = \frac{1}{2}, b_1 = 1 \Rightarrow$ (MPR)

2) $n=2$ auf $(-1, 1) \Rightarrow P_2(t) = t^2 - \frac{1}{3} \Rightarrow c_1 = -\frac{1}{\sqrt{3}}, c_2 = \frac{1}{\sqrt{3}}$
 $b_1 = 1, b_2 = 1$

Bez Gewichte zu c_1, \dots, c_n
Lagrange Polynome dazu: l_1, \dots, l_n , Grad $n-1$

$f(t) = \sum l_i(t)^2$, Grad $f = 2(n-1) \leq 2n-1 \rightarrow$ QF exist

$0 < \int_0^1 l_i(t)^2 dt = \sum_{j=1}^n b_j \int_0^1 l_i(t) l_j(t) dt = b_i$
|| $\int_0^1 l_i(t) l_j(t) dt = \delta_{ij}$

Bez 1) Gauss-Knoten sind nicht verschachtelt ☹️
Nachteil bei Adaptivität

2) offene

3) 1 Ende = Knoten (fix) \Rightarrow nur max Ordng $2n-1$ Rodas
2 Enden = Knoten (fix) \Rightarrow nur max Ordng $2n-2$ Lobatto

4) Fehler für zusammengesetzt Gauss-Qt:

$\left| \int_a^b f(t) dt - \sum_{k=1}^n G_p(f, x_{k-1}, x_k) \right| \leq c \cdot h^{2D} \max_{z \in (a,b)} |f^{(2D)}(z)|$

5) elegante Berechnung der Knoten & Gewichte
der Gauss-Quadratur mit Golub-Welsch Alg
(1969) EW, EV einer Matrix
aber nicht sehr stabil für grosses n
teuer!

2012-2014 Bogert-Townsend: bessere
(stabiler & schneller) Alg.

Arbeits	Genauigkeit	
	Knoten	Gewichte
GW	$O(n^3) / O(n^2)$	$O(n) / O(n^2)$
BT	$O(n)$	$O(1) / O(1)$
CC	$O(n \log n)$	$O(1) / O(1)$

05.03.2024

§2 Trigonometrische / Fourier Approximation

§2.1. Grundlagen

Approx: Taylor: $f(t) = f(t_0) + (t-t_0)f'(t_0) + \frac{(t-t_0)^2}{2!} f''(t_0) + \dots$

Lokal. \downarrow Potenzen \uparrow $|t-t_0| \leq \text{klein}$ \uparrow Rest Fehler

Besser: Interpolation (weder lokal noch global)

$$L^2(0,1) = \{ f: (0,1) \rightarrow \mathbb{R}, \|f\|_{L^2(0,1)} < \infty \}$$

$$\|f\|_{L^2(0,1)}^2 = \int_0^1 |f(t)|^2 dt$$

$$\|f\|^2 = \langle f, f \rangle, \quad \langle g, f \rangle = \int_0^1 \overline{g(t)} f(t) dt$$

$L^2(0,1)$ lin. Raum mit Norm $\|\cdot\|$
aus dem Skalarprodukt $\langle \cdot, \cdot \rangle$

$$f \in L^2(0,1) \Rightarrow f = \sum_{k=-\infty}^{\infty} \hat{f}(k) e^{2\pi i k t} \quad \text{Fourier Reihe}$$

$$\hat{f}(k) = \int_0^1 f(t) e^{-2\pi i k t} dt \quad \text{Fourier Koeffizient}$$

$$\text{Parseval } \|f\|_{L^2}^2 = \sum_{k=-\infty}^{\infty} |\hat{f}(k)|^2$$

$$L^2(0,1) = \text{span} \{ f_k; k \in \mathbb{Z} \}$$

$$f_k(t) = e^{2\pi i k t}$$

(f_k) vollständige ONB in $L^2(0,1)$

Falls erlaubt:

$$\widehat{f^{(n)}}(k) = (2\pi i k)^n \hat{f}(k)$$

$$\|f^{(n)}\|_2^2 = (2\pi)^{2n} \sum_{k=-\infty}^{\infty} k^{2n} |\hat{f}(k)|^2$$

Glattheit von f (in \mathbb{C}) \leftrightarrow Abfallverhalten von $\hat{f}(k)$

$$\int_0^1 |f^{(n)}(t)| dt < \infty \Rightarrow \hat{f}(k) \sim k^{-n}$$

Approximieren:

$N = \text{gerade}$, verwende nur die ersten $\frac{N}{2}$ Frequenzen.

$$T_N = \text{span} \left\{ \underbrace{t_{-\frac{N}{2}}, t_{-\frac{N}{2}+1}, \dots, t_0}_{\frac{N}{2}}, \underbrace{t_1, \dots, t_{\frac{N}{2}-1}}_{\frac{N}{2}} \right\}$$

Raum trigonometrischer Polynome

$$\dim T_N = N, \quad T_N \subset L^2(0,1) \text{ Unterraum.}$$

$$f(t) = \sum_{k=-\infty}^{\infty} \hat{f}(k) \varphi_k(t) \approx \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}(k) \varphi_k(t)$$

f glatt \Rightarrow sehr gute (globale) Approximation.
 N gross

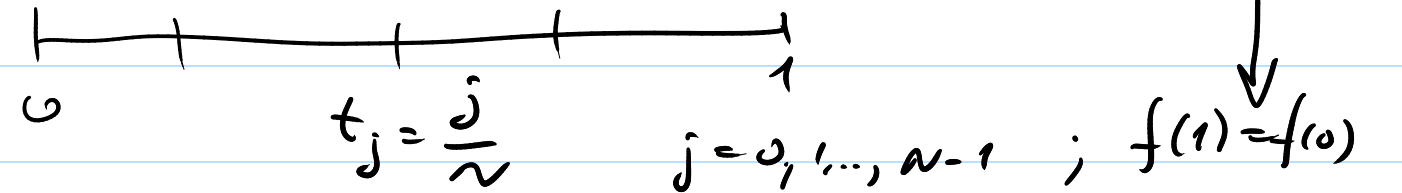
Aliasing für $\frac{N}{2}$ Frequenzen brauche zumindest
 N Punkte / Werte

f periodisch

Wie berechnen wir $\hat{f}(k)$?

Sum. TR:

$$\hat{f}(k) = \int_0^1 f(t) e^{-2\pi i k t} dt$$



$t_j = \frac{j}{N} \quad j=0, 1, \dots, N-1; \quad f(1) = f(0)$
(N äquidistante Punkte zw. 0 und 1)

Sum. TR:

$$\begin{aligned} \hat{f}(k) &\approx \frac{1}{N} \left[\frac{1}{2} f(0) + \sum_{j=1}^{N-1} f(t_j) e^{-2\pi i k \frac{j}{N}} + \frac{1}{2} f(1) \right] = \\ &= \frac{1}{N} \sum_{j=0}^{N-1} f\left(\frac{j}{N}\right) e^{-2\pi i k \frac{j}{N}} =: \hat{f}_N(k) \end{aligned}$$

$$f(t) \approx p_N(t) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_N(k) p_k(t) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_N(k) e^{2\pi i k t}$$

$$\hat{f}(k) \approx \hat{f}_N(k) = \frac{1}{N} \sum_{j=0}^{N-1} f\left(\frac{j}{N}\right) \omega_N^{kj}$$

§2.2. Diskrete Fourier Transformation

N gerade, $i^2 = -1$, Notiere $\omega_N = e^{-\frac{2\pi i}{N}}$, $\omega_N = \bar{\omega}_N = e^{\frac{2\pi i}{N}}$

Dann: $\omega_N^{k+N} = \omega_N^k$ für alle $k \in \mathbb{Z}$

$\omega_N^{t+kN} = \omega_N^t$ für alle $t \in \mathbb{R}$, $k \in \mathbb{Z}$

$$\omega_N^N = 1, \quad \omega_N^{N/2} = -1$$

$$\sum_{k=0}^{N-1} \omega_N^{kj} = \begin{cases} N, & j=0 \\ 0, & j \neq 0 \end{cases}$$

Fixiere l :

$$p_N\left(\frac{l}{N}\right) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \left(\frac{1}{N} \sum_{j=0}^{N-1} f\left(\frac{j}{N}\right) \omega_N^{kj} \right) e^{2\pi i k \frac{l}{N}} =$$

$$\underbrace{\omega_N^{kl} = \omega_N^{-kl}}$$

$$= \frac{1}{N} \sum_{j=0}^{N-1} f\left(\frac{j}{N}\right) \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \omega_N^{k(j-l)} =$$

0 falls $j-l \neq 0$
 N falls $j-l=0$

$$= \frac{1}{N} (0 + 0 + 0 + \dots + f\left(\frac{l}{N}\right) \cdot N + 0 + \dots + 0) =$$

$$= \frac{1}{N} f\left(\frac{l}{N}\right) N = f\left(\frac{l}{N}\right) \rightarrow$$

$$p_N\left(\frac{l}{N}\right) = f\left(\frac{l}{N}\right)$$

$\Rightarrow P_N(t)$ erfüllt die Interpolationsbedingung:

$$\left. \begin{array}{l} P_N(t_j) = f(t_j) \\ j = 0, 1, \dots, N-1 \end{array} \right\} \quad \left(t_j = \frac{j}{N} \right)$$

Sei nun in \mathbb{C}^N die trigonometrische Basis

$$\{ \underline{v}_0, \underline{v}_1, \dots, \underline{v}_{N-1} \} \subset \mathbb{C}^N$$

$$\underline{v}_{-k} = \begin{bmatrix} \omega_N^{0 \cdot k} \\ \omega_N^{1 \cdot k} \\ \vdots \\ \omega_N^{(N-1) \cdot k} \end{bmatrix} \in \mathbb{C}^N \quad \text{für } k=0, 1, \dots, N-1$$

$$\underline{V} = [\underline{v}_0 \quad \underline{v}_1 \quad \underline{v}_2 \quad \dots \quad \underline{v}_{N-1}] \in \mathbb{C}^{N \times N}$$

$$\underline{V}^H \underline{V} = N \underline{I}_N$$

\underline{V} symmetrisch, nicht Hermite-symmetrisch.

$$\underline{v}_{-k} = \underline{V} \underline{e}_{-k} \quad \underline{e}_{-k} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \begin{array}{l} \text{te Position} \\ (k+1) \end{array}$$

$\underline{e}_0, \underline{e}_1, \dots, \underline{e}_{N-1}$ Standardbasis

$$\mathbb{C}^N \Rightarrow \underline{y} = \sum_{k=0}^{N-1} y_k \underline{e}_{-k} = \sum_{k=0}^{N-1} z_k \underline{v}_{-k} = \underline{V} \underline{z}$$

Standard Basis
trigonometrische Basis

$$\underline{y} = \underline{V} \underline{z} \Leftrightarrow \underline{z} = \underline{V}^{-1} \underline{y} = \frac{1}{N} \underline{V}^H \underline{y}$$

\underline{F}_N

$$\underline{F}_N = \underline{V}^H \quad \text{Fourier Matrix}$$

$$\underline{F}_N = \left(\omega_N^{ij} \right)_{i,j=0,1,\dots,N-1}$$

Def $F_N = \mathbb{C}^N \rightarrow \mathbb{C}^N, F_N(\underline{y}) = \underline{F_N y}$

Diskrete Fourier Transformation.

$[fft(y)]$

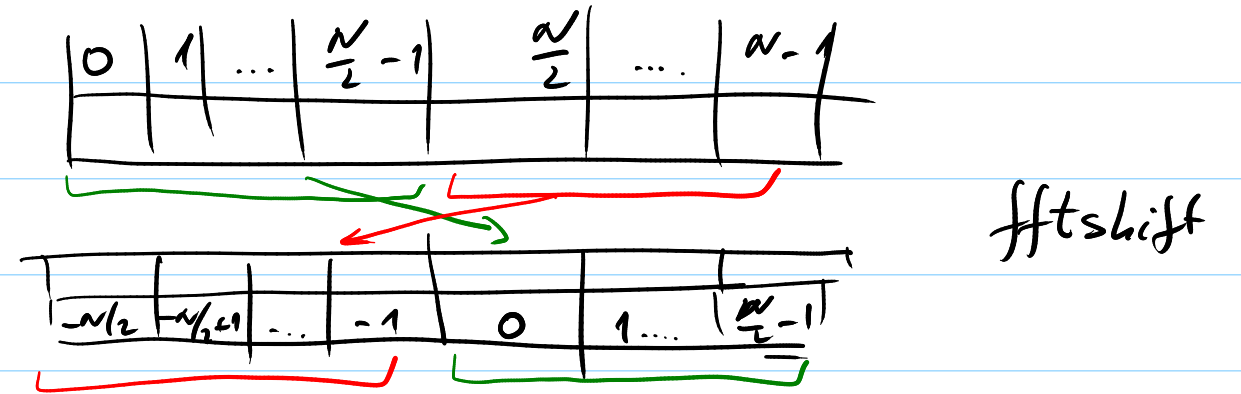
Bez 1) $\frac{1}{\sqrt{N}} F_N$ unitär

2) $\frac{1}{\sqrt{N}} F_N F_N \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} y_{N-1} \\ y_{N-2} \\ \vdots \\ y_0 \end{bmatrix}$

3) $\frac{1}{\sqrt{N^2}} F_N^4 = \underline{I} \Rightarrow \text{EW von } F_N \text{ sind } \{1, -1, i, -i\}$

Bez 1) $\underline{z} = \frac{1}{\sqrt{N}} F_N \underline{y}, \quad fft(\underline{z}) = F_N \underline{z} = N \underline{z}$
 $ifft(N \underline{z}) = \underline{z}$

2) fft: die Frequenzen stehen am "falschen" Ort:



$\underline{y} = \sum_{k=0}^{N-1} z_k \underline{v}_k = \sum_{k=0}^{N-1} z_k \begin{bmatrix} w_N^{0k} \\ \vdots \\ w_N^{(N-1)k} \end{bmatrix}$

Daraus schauen wir Komponente l:

$y_l = \sum_{k=0}^{N-1} z_k w_N^{lk} = \sum_{k=0}^{\frac{N}{2}-1} z_k w_N^{lk} + \sum_{k=\frac{N}{2}}^{N-1} z_k w_N^{lk}$

$\sum_{j=-\frac{N}{2}}^{-1} z_{N+j} w_N^{lj} =$
 $k = N+j \Leftrightarrow j = k - N$

$$= \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} r_k w_N^{lk}$$

Theorem

$$\|f - p_N\|_{L^2} \leq \sqrt{1 + c_k} \cdot N^{-k} \|f^{(k)}\|_{L^2}$$

mit $c_k = 2 \sum_{l=1}^{\infty} (2l-1)^{-2k}$

FFT (fast Fourier Transform)

$N = 2n$

$$r_k = \sum_{j=0}^{n-1} y_j e^{-\frac{2\pi i}{N} jk} = \sum_{j=0}^{n-1} y_{\text{gerade}} e^{-\frac{2\pi i}{2n} jk} +$$

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

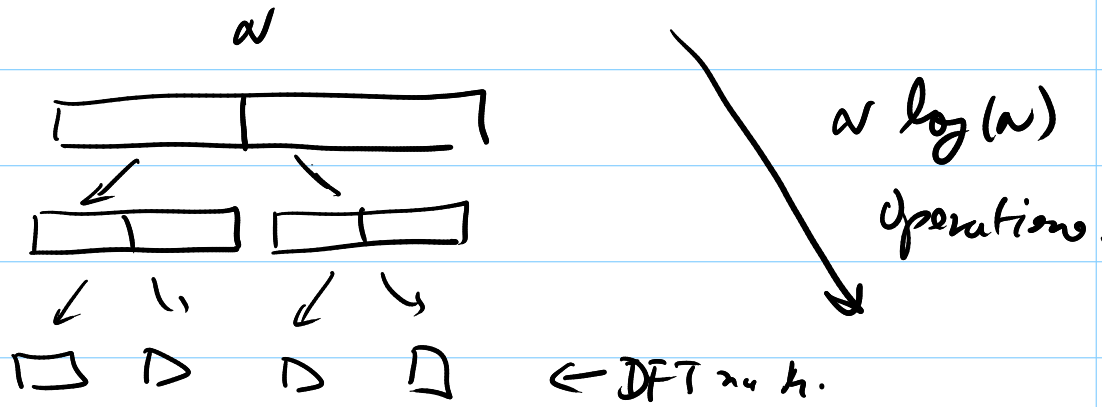
$$\sum_{j=0}^{n-1} y_{2j+1} e^{-\frac{2\pi i}{2n} jk} e^{-\frac{2\pi i}{2n} k} = e^{-\frac{2\pi i}{2n} k} \sum_{j=0}^{n-1} y_{2j+1} e^{-\frac{2\pi i}{2n} jk}$$

DFT für $n = \frac{N}{2}$ Koeffizienten

DFT für $n = \frac{N}{2}$ Koeffizienten!

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

$N = 2^p$



statt N^2 wie mit FFT

$N \log N \ll N^2$ für N gross!

Auswertung in weitere Punkte $\frac{k}{M}$ mit $M \gg N$
"zero padding"

$$P_N(t) = \sum_{j=-n}^{n-1} z_j e^{2\pi i j t} = \sum_{j=-n}^{n-1} \tilde{z}_j e^{2\pi i j t}$$

$$\text{mit } \tilde{z}_j = \begin{cases} z_j & \text{für } -n \leq j \leq n-1 \\ 0 & \text{sonst} \end{cases}$$

Verwende die inverse FFT auf \tilde{z}
 \Rightarrow werte $P_N(\frac{k}{M})$ in $M \log M$ Operationen!

Ben $\underline{z} = \frac{1}{N} \underline{F}_N \underline{y} \Rightarrow \underline{\tilde{z}} = \frac{1}{N} \underline{F}_N \underline{\tilde{y}} = \underline{F}_N^{-1} \underline{\tilde{y}}$

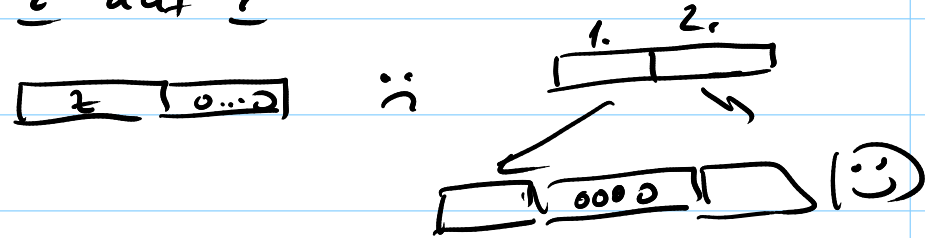
$\Rightarrow \underline{\tilde{y}} = \underline{F}_N^{-1} \underline{\tilde{z}}$ kann manchmal schneller sein
braucht keine Division mit N mehr!

$\underline{\tilde{y}} = \underline{F}_N \underline{F}_N^{-1} \underline{\tilde{y}}$ Wenn $\underline{y} \in \mathbb{R}^N \Rightarrow$ brauche kein $\underline{\tilde{y}}$
NICE für $\underline{y} \in \mathbb{R}^N$ nur!

Technik f gegeben, Werte $\underline{y} = f(\underline{t})$, $\underline{t} = \begin{bmatrix} 0 \\ \Delta t \\ 2\Delta t \\ \vdots \\ (n-1)\Delta t \\ n\Delta t \end{bmatrix}$ Den Berechnung der Ableitungen.

$$\underline{z} = \frac{1}{N} F_N \underline{y} = \frac{1}{N} \text{fft}(\underline{y})$$

erweitere \underline{z} auf $\tilde{\underline{z}}$



$$\tilde{\underline{y}} = \text{ifft}(M \tilde{\underline{z}})$$

$O(M \log M) \leq nM$ direkte Auswertung.

code 2.4.4.

Direkt:

$$f(t) \approx p_n(t) = \sum_{j=-n}^n z_j e^{2\pi i j t} =)$$

$$p_n'(t) = 2\pi i \sum_{j=-n}^n j z_j e^{2\pi i j t}$$

$$\underline{z} = \frac{1}{N} F_N \underline{y}$$

$$\underline{z}' = 2\pi i \text{diag}(0, 1, \dots, n-1, -n, -n+1, \dots, -1) \underline{z}$$

$$\underline{y}' = \text{real}(N F_N^{-1} \underline{z}') \approx [f'(t_j)]_{j=0,1,\dots,n-1}$$

Indirekt

$$\underline{c} = F_N^{-1}(\underline{y}) \quad (= \overline{\underline{c}})$$

NUR

Für

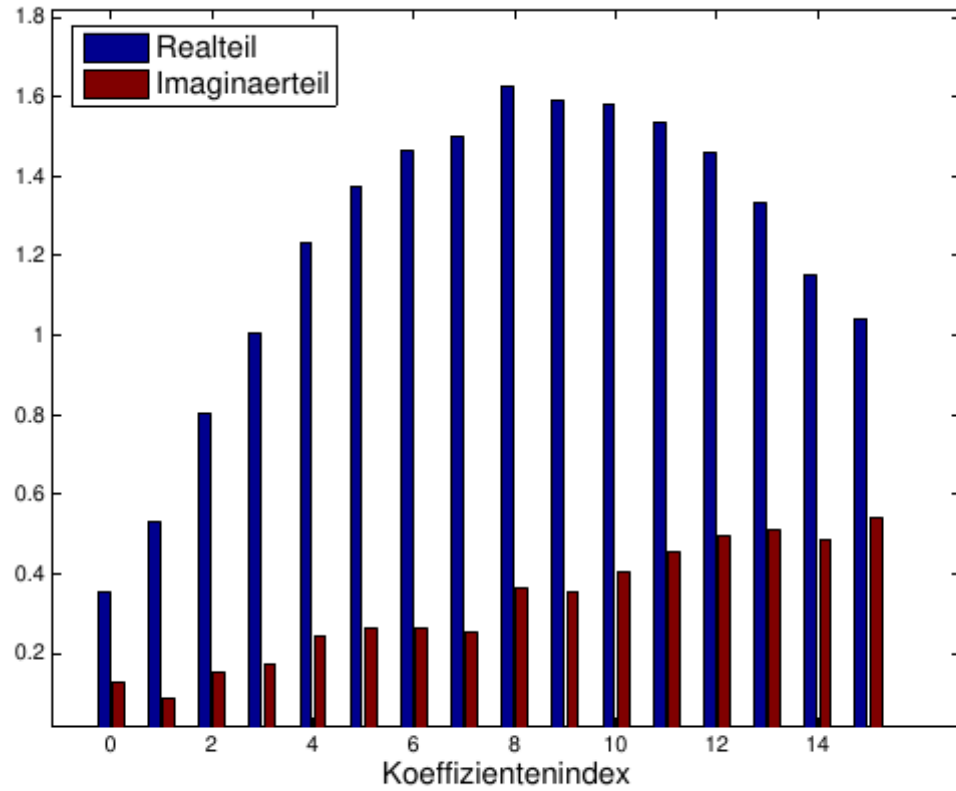
reelle

Signale \underline{y} !

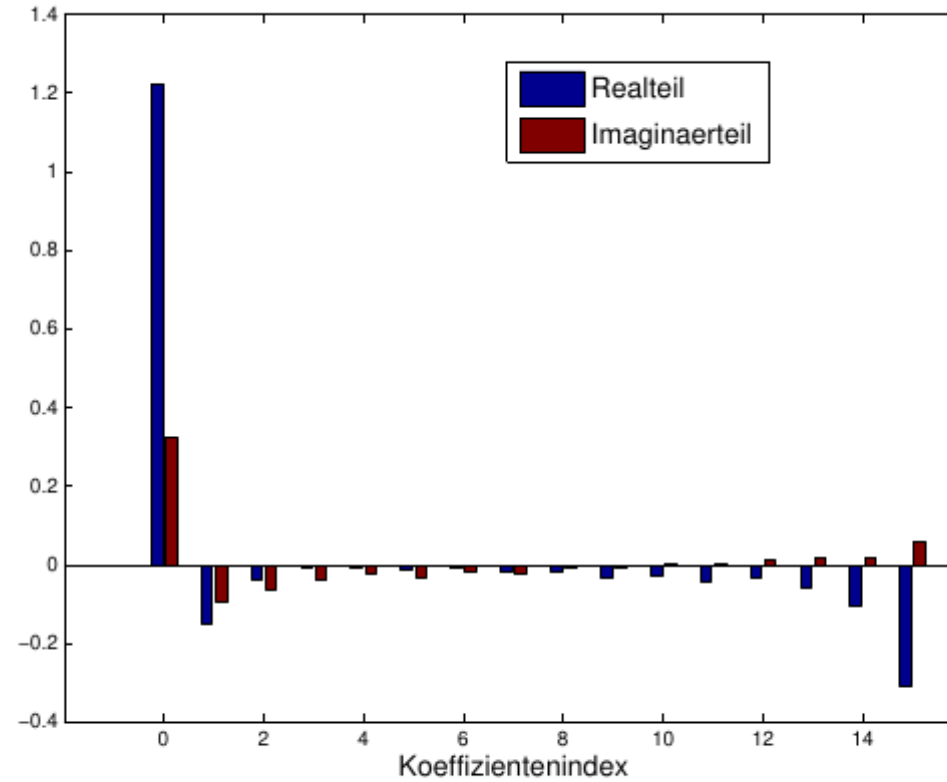
$$\underline{c}' = (-2\pi i) \underline{D} \underline{c} = -2\pi i \underline{D} \overline{\underline{c}}$$

$$\underline{y}' = \text{real}(\underline{F}_N \underline{c}') \quad \text{conj}(\underline{F}_N \underline{c}')$$

12.03.2024



Vektor in Standardbasis



Vektor in der trigonometrische Basis

Der ℓ -te Eintrag in \mathbf{y} ist

$$y_\ell = \sum_{k=0}^{N-1} z_k w_N^{\ell k} = \sum_{k=0}^{N/2-1} z_k w_N^{\ell k} + \sum_{k=N/2}^{N-1} z_k w_N^{\ell k} = \sum_{k=0}^{N/2-1} z_k w_N^{\ell k} + \sum_{k=-N/2}^{-1} z_{N+j} w_N^{\ell j} = \sum_{k=-N/2}^{N/2-1} \gamma_k \exp\left(\frac{2\pi i}{N} \ell k\right),$$

was einem Teil der Fourier-Reihe ausgewertet im Punkt $\frac{\ell}{N} \in [0, 1[$ entspricht, wobei die (diskreten) Fourier-Koeffizienten durch eine Umsortierung der Koeffizienten in der trigonometrischen Basis gegeben sind:

$$\gamma_k = \begin{cases} z_k, & \text{für } 0 \leq k \leq n-1 = \frac{N}{2}-1 \\ z_{k+N}, & \text{für } -\frac{N}{2} = -n \leq k < 0 \end{cases} \quad (2.2.15)$$

Bemerkung 2.2.8. numpy stellt die Funktionen

$$\mathbf{c} = \text{fft}(\mathbf{y}) \text{ und } \mathbf{y} = \text{ifft}(\mathbf{c})$$

für

$$\mathbf{c} = \mathbf{F}_N \mathbf{y} \text{ und } \mathbf{y} = \frac{1}{N} \mathbf{F}_N^H \mathbf{c}$$

zur Verfügung.

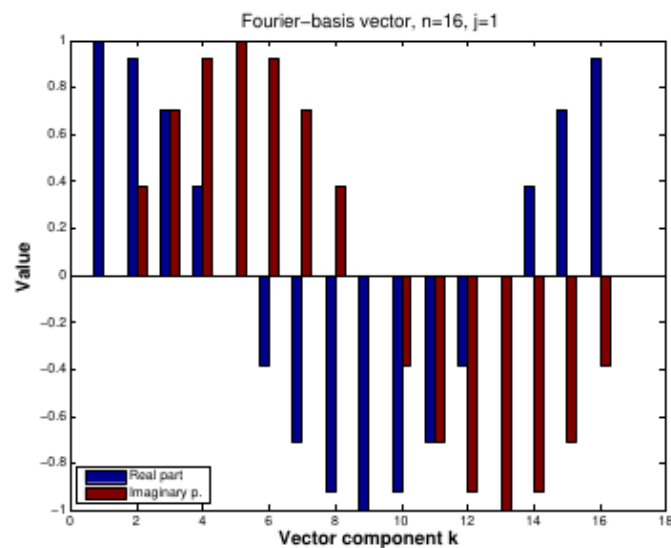
$$f(x) \approx \sum_{j=0}^{\frac{N}{2}-1} z_j e^{2\pi i j x} + \sum_{j=\frac{N}{2}}^{N-1} z_j e^{2\pi i (j-N)x},$$

$$f(x) \approx \sum_{k=0}^{\frac{N}{2}-1} z_k e^{2\pi i k x} + \sum_{k=-\frac{N}{2}}^{-1} z_{k+N} e^{2\pi i k x},$$

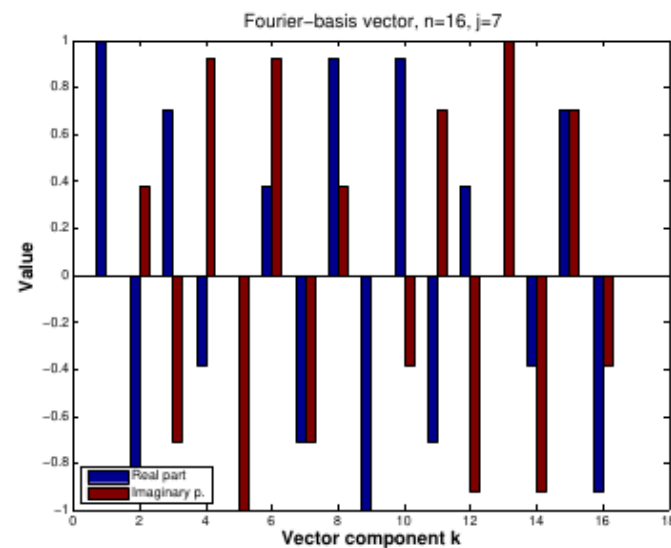
$\zeta = \text{fft.fftshift}(z)$:

$$f(x) \approx \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \zeta_k e^{2\pi i k x}.$$

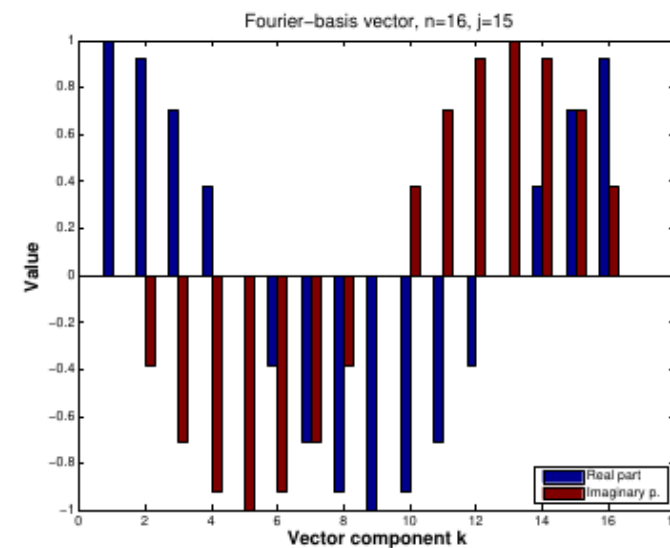
Beispiel 2.2.11. Wenn wir `fft.fft` auf einen Vektor y der Länge 8 anwenden, erhalten wir den Vektor $c[0], \dots, c[7]$, und `fft.fftshift` stellt diese Einträge in der Reihenfolge $c[4], c[5], c[6], c[7], c[0], c[1], c[2], c[3]$, die der $\gamma_{-4}, \gamma_{-3}, \gamma_2, \gamma_1, \gamma_0, \gamma_1, \gamma_2, \gamma_3$ entsprechen.



“niedrige Frequenz”



“hohe Frequenz”



“niedrige Frequenz”

Beispiel 2.2.13 (Frequenzanalyse mittels DFT). Nun nehmen wir ein periodisches Signal mit zwei Frequenzen (1 und 7); dazu addieren wir eine zufällige Störung. Wir transformieren dann dieses gestörte Signal in die Fourier-Basis und wir schauen uns das sogenannte “power spectrum” (die spektrale Leistungsdichte) an. Die ursprünglichen zwei Frequenzen sind eindeutig sichtbar in der Abbildung 2.2.14. In diesem Beispiel sind die Periodizität und die Abtastpunkte des Signals allerdings wichtig; wir schauen uns auch nur die erste Hälfte der Koeffizienten. Probieren Sie auch `bar(t[:]-64/2,fft.fftshift(p[:]))!`

```

1 from numpy import sin, pi, linspace, random, fft
2 from pylab import plot, bar, show
3 t = linspace(0,63,64)
4 x = sin(2*pi*t/64)+sin(7*2*pi*t/64)
5 y = x + random.randn(len(t)) #distortion
6 c = fft.fft(y)
7 p = abs(c)**2; p /= 64.
8 plot(t,y,'-+'); show()
9 bar(t[:32],p[:32]); show()

```

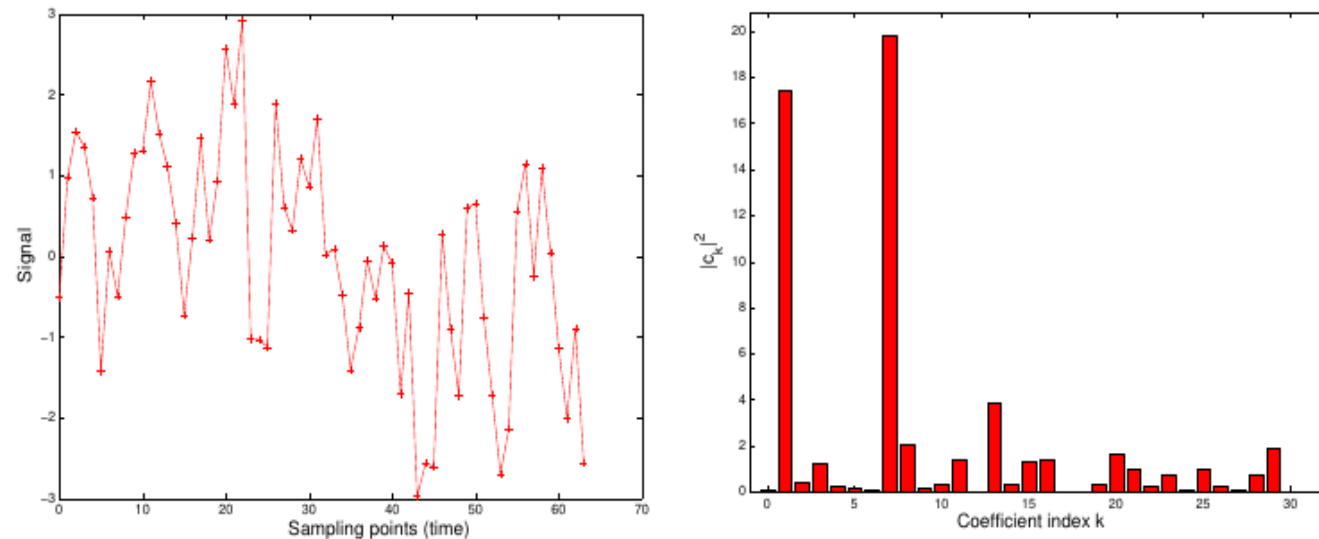
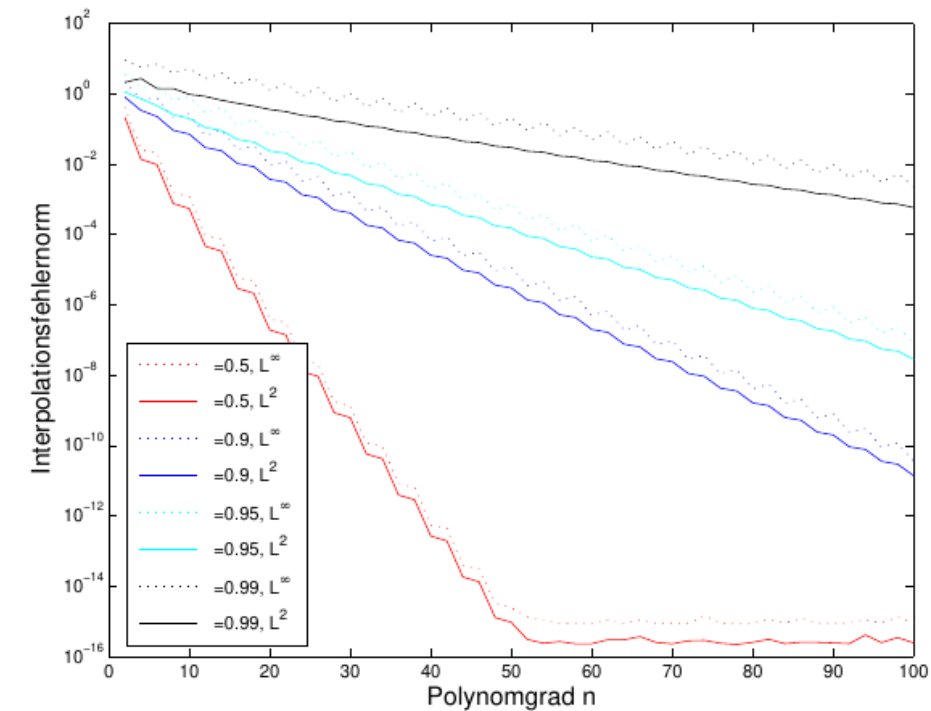


Abb. 2.2.14. Frequenzanalyse mittels DFT

Beispiel 2.5.3. Für $\alpha \in [0, 1)$ definieren wir:

$$f(t) = \frac{1}{\sqrt{1 - \alpha \sin(2\pi t)}} \text{ auf } I = [0, 1],$$

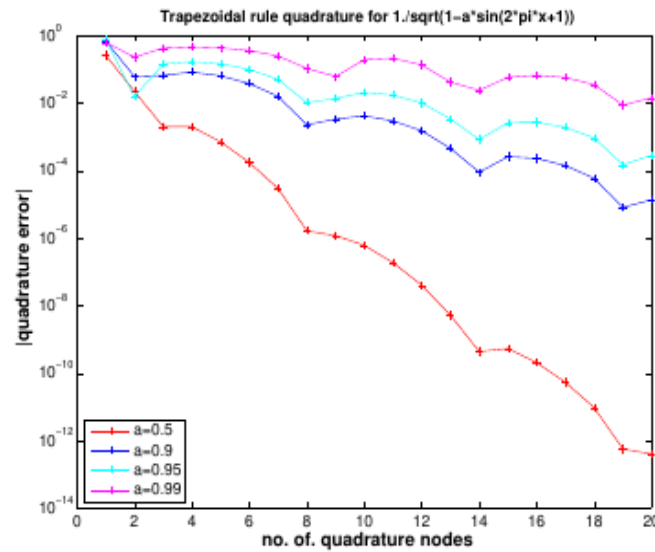
und wir beobachten exponentielle Konvergenz in n , schneller für kleineres α .



Die Trapez-Regel auf äquidistanten Punkten hat Ordnung 2, aber wenn wir sie auf den 1-periodischen glatten (analytischen) Integranden anwenden

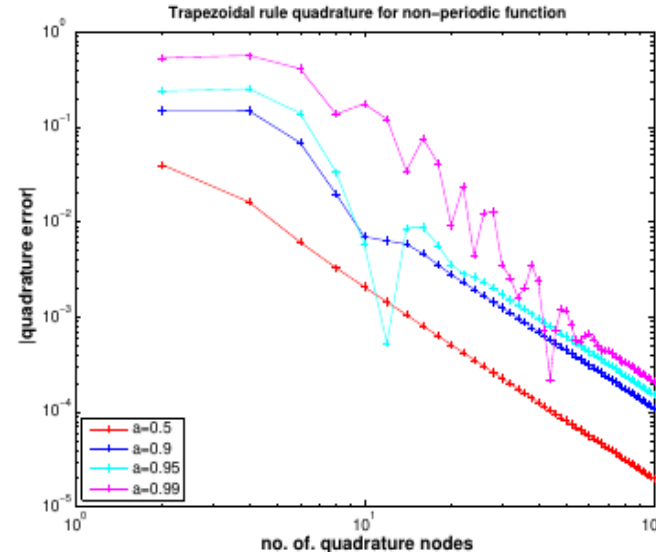
$$f(t) = \frac{1}{\sqrt{1 - a \sin(2\pi t - 1)}}, \quad 0 < a < 1$$

erhalten wir folgende erstaunliche Ergebnisse (als "exakten Wert des Integrals" verwenden wir T_{500}):



Quadratur-Fehler für $T_n(f)$ auf $[0, 1]$

Exponentielle Konvergenz!



Quadratur-Fehler für $T_n(f)$ auf $[0, \frac{1}{2}]$

nur algebraische Konvergenz...

Die Erklärung basiert auf der Tatsache, dass T_m exakt für trigonometrische Polynome vom Grad $< 2m$ sind:

$$f(t) = e^{2\pi ikt} \begin{cases} \int_0^1 f(t) dt = \begin{cases} 0, & \text{wenn } k \neq 0, \\ 1, & \text{wenn } k = 0. \end{cases} \\ T_m(f) = \frac{1}{m} \sum_{l=0}^{m-1} e^{\frac{2\pi i}{m}lk} \stackrel{(2.2.13)}{=} \begin{cases} 0, & \text{wenn } k \notin m\mathbb{Z}, \\ 1, & \text{wenn } k \in m\mathbb{Z}. \end{cases} \end{cases}$$

§2.3

Clenshaw-Curtis Formeln

Die Quadraturformeln von Clenshaw-Curtis, bzw. Fejer, basieren auf der Idee der Approximation des Integranden mittels Chebyshev-Polynomen. Für die Integration beliebiger Funktionen wird die Variablensubstitution $x = \cos(\theta)$ verwendet. Hieraus folgt:

$$\int_{-1}^1 f(x) dx = \int_0^\pi f(\cos(\theta)) \sin(\theta) d\theta.$$

Wir definieren $F(\theta) := f(\cos(\theta))$. Da $F(\theta)$ 2π -periodisch und gerade ist, lässt es sich in eine Kosinus-Reihe entwickeln: $F(\theta) = \sum_{k=0}^\infty a_k \cos(k\theta)$, woraus folgt:

$$\int_0^\pi F(\theta) \sin(\theta) d\theta = \int_0^\pi \sum_{k=0}^\infty a_k \cos(k\theta) \sin(\theta) d\theta = \sum_{k=0}^\infty a_k \int_0^\pi \cos(k\theta) \sin(\theta) d\theta = \sum_{\substack{k \text{ gerade} \\ k \geq 0}} \frac{2a_k}{1 - k^2}. \quad (1.5.7)$$

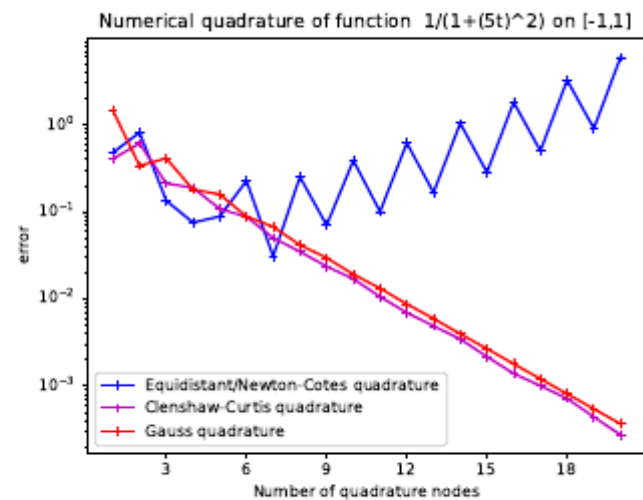
Die Koeffizienten a_k lassen sich mittels FFT³ oder DCT⁴ berechnen.

Für die Stützstellen der Clenshaw-Curtis Quadraturformel gilt der folgende Satz, den wir hier ohne Beweis angeben:

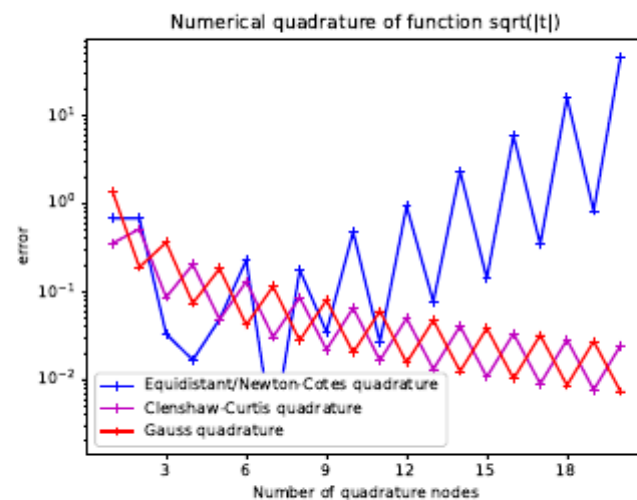
Theorem 1.5.4. Die Stützstellen der Clenshaw-Curtis Quadraturformel sind die Extreme der Chebyshev-Polynome und die Nullstellen der Chebyshev-Polynome 2-ter Art.

Code 1.5.5: Clenshaw-Curtis: direkte Implementierung

Code 1.5.6: Clenshaw-Curtis: Gewichte und Knoten



Fehler für $f_1(t) := \frac{1}{1+(5t)^2}$ auf $[0, 1]$



Fehler für, $f_2(t) := \sqrt{t}$ auf $[0, 1]$

Für Integrale der Form

$$\int_a^b f(x) \omega(x) dx$$

spielen verschiedene orthogonale Polynome eine wesentliche Rolle:

Quadratur	Intervall	Gewichtsfunktion	Polynom	Not.	scipy.special.
Gauss	$(-1, 1)$	1	Legendre	P_k	roots_legendre
Chebyshev I	$(-1, 1)$	$\frac{1}{\sqrt{1-x^2}}$	Chebyshev I	T_k	roots_chebyt
Chebyshev II	$(-1, 1)$	$\sqrt{1-x^2}$	Chebyshev II	U_k	roots_chebyu
Jacobi $\alpha, \beta > -1$	$(-1, 1)$	$(1-x)^\alpha(1+x)^\beta$	Jacobi	$P_k^{(\alpha, \beta)}$	roots_jacobi
Hermite	\mathbb{R}	e^{-x^2}	Hermite	H_k	roots_hermite
Laguerre	$(0, \infty)$	$x^\alpha e^{-x}$	Laguerre	L_k	roots_genlaguerre

Abb. 1.5.10. Gewichtsfunktionen für Quadraturformeln

§3 Einfache Verfahren für ODEs

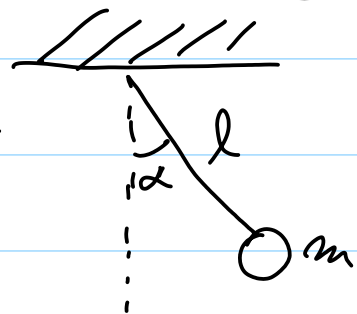
§3.1. Lokale Linearisierung

ODE = ordinary differential equation

Finde $y: [0, T] \rightarrow \mathbb{R}^d$ so dass

$$\begin{cases} \dot{y}(t) = f(t, y) & f: [0, T] \times \mathbb{R}^d \rightarrow \mathbb{R}^d \\ y(0) = y_0 \text{ Startwert} \end{cases}$$

Mathematisches Pendel



$$m l \ddot{\alpha}(t) = -m g \sin \alpha(t)$$

Physik $\sin \alpha(t) \approx \alpha(t)$

↳ gut nur für $\alpha(t)$ klein

(globale Linearisierung)

$$(0) \begin{cases} \ddot{\alpha}(t) = -\frac{g}{l} \sin \alpha(t) \\ \alpha(0) = \alpha_0 \\ \dot{\alpha}(0) = \dot{\alpha}_0 \end{cases}$$

ODE 2. Ordnung, da die 2te Ableitung vorkommt.

$$\dot{\alpha}(t) = \frac{d}{dt} \alpha(t)$$

$$\ddot{\alpha}(t) = \frac{d^2}{dt^2} \alpha(t)$$

$$\sin \alpha = \alpha - \frac{1}{3!} \alpha^3 + \dots = \alpha + O(\alpha^3)$$

↳ klein für α klein

Neues Model.

$$(1) \begin{cases} \ddot{\beta}(t) = -\frac{g}{l} \beta(t) & \text{hat exakte Lösung.} \\ \beta(0) = \alpha_0 \\ \dot{\beta}(0) = \dot{\alpha}_0 \end{cases}$$

$$\omega^2 = \frac{g}{l}$$

$$\beta(t) = \frac{\dot{\alpha}_0}{\omega} \sin(\omega t) + \alpha_0 \cos(\omega t)$$

Trick: Reduziere die Ordnung der ODE:

$$\text{Notieren } p(t) = \dot{\alpha}(t) \Rightarrow$$

$$\dot{p}(t) = \ddot{\alpha}(t) \Rightarrow$$

$$\dot{p}(t) = -m g \sin \alpha(t)$$

Somit (0) \Leftrightarrow System ODEs 1. Ordnung

$$\begin{cases} \dot{\alpha} = p \\ \dot{p} = -m g \sin \alpha(t) \\ \alpha(0) = \alpha_0 \\ p(0) = \dot{\alpha}_0 \end{cases}$$

$$\begin{pmatrix} y_1 = \alpha \\ y_2 = p \end{pmatrix} \begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = -m g \sin(y_1) \end{cases}$$

$$f\left(\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}\right) = \begin{bmatrix} y_2 \\ -m g \sin(y_1) \end{bmatrix}$$

$\underline{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ \rho \end{bmatrix}$ $\underline{y} = [0, T] \rightarrow \mathbb{R}^2$ Unbekannte Funktion.

$f(\underline{y}) = \begin{bmatrix} y_2 \\ -mgy - y_1 \end{bmatrix}$ $\underline{f}: \mathbb{R}^2 \rightarrow \mathbb{R}^2$
 $\underline{y} \mapsto \underline{f}(\underline{y})$

BSP Newton Gleichung:

Notiere $m \ddot{u} = F(t, u)$
 $y_1 = u, y_2 = \dot{u} \Rightarrow \underline{\dot{y}} = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ \frac{1}{m} F(t, y_1) \end{bmatrix}$

$\underline{f}(t, \underline{y}) = \begin{bmatrix} y_2 \\ \frac{1}{m} F(t, y_1) \end{bmatrix} = \begin{bmatrix} f_1(t, \underline{y}) \\ f_2(t, \underline{y}) \end{bmatrix}$

$f_1(t, \underline{y}) = y_2, f_2(t, \underline{y}) = \frac{1}{m} F(t, y_1)$

Trick: Man kann jede nicht-autonome ODE zu einem System autonomer ODEs

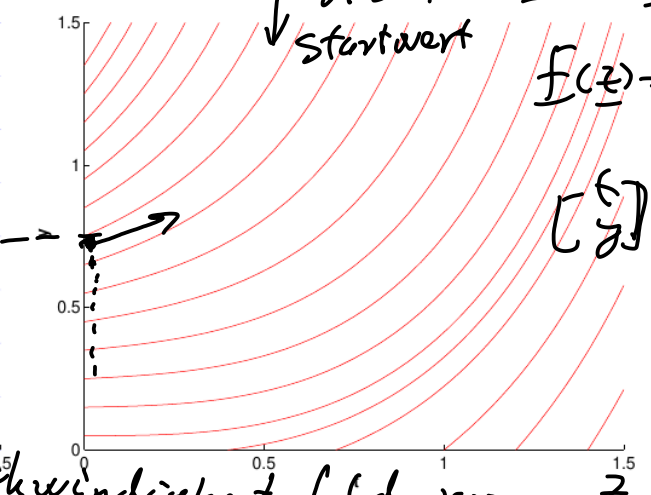
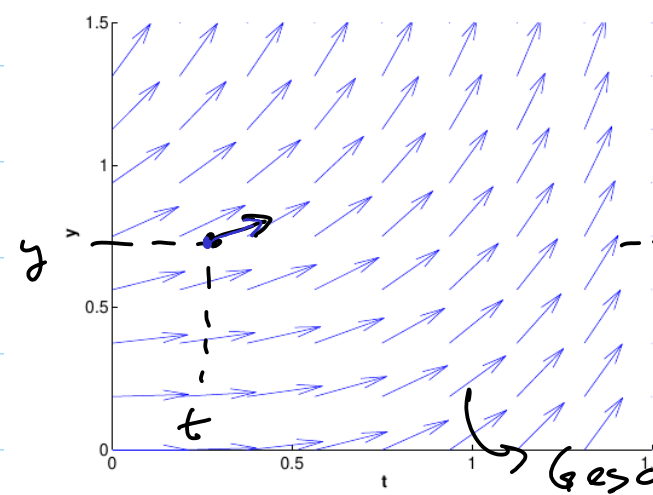
$\underline{\dot{y}} = \underline{f}(t, \underline{y})$, Notiere $\underline{z} = \begin{bmatrix} y \\ t \end{bmatrix} \in \mathbb{R}^{d+1}$

$\underline{\dot{z}} = g(\underline{z})$ mit $g(\underline{z}) = \begin{bmatrix} f(\underline{z}_{d+1}) \begin{bmatrix} z_1 \\ \vdots \\ z_d \end{bmatrix} \\ 1 \end{bmatrix}$

Beispiel 2.1.13. (Richtungsfeld und Lösungskurven)

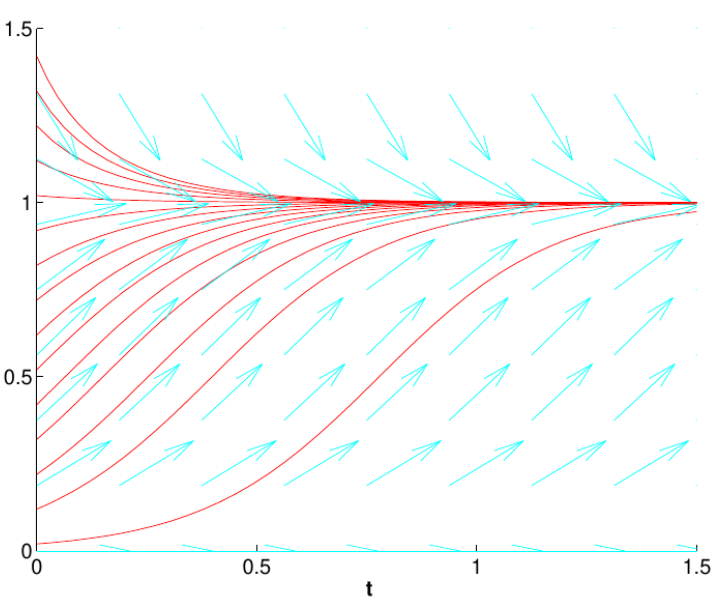
Die Riccati-Differentialgleichung lautet:

$\dot{y} = y^2 + t^2$; hier: $d = 1, D = \mathbb{R}^+$ und $t \in \mathbb{R}^+$.



Lösungskurve $\underline{z} = \underline{j}$
 $\underline{\dot{z}} = \underline{f}(\underline{z})$
 $\underline{f}(\underline{z}) = \begin{bmatrix} y^2 + t^2 \\ 1 \end{bmatrix}$
 $\begin{bmatrix} t \\ y \end{bmatrix} \mapsto \underline{f} \left(\begin{bmatrix} t \\ y \end{bmatrix} \right)$

Richtungsfeld
 Geschwindigkeitsfeld von $\underline{\dot{z}}$
 Lösungskurven

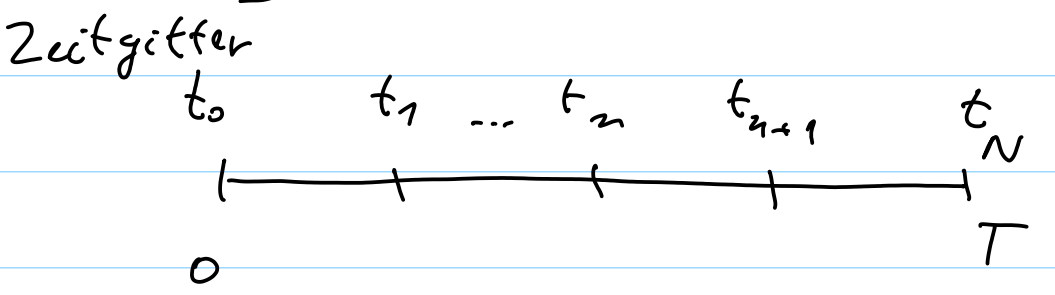


Logistische Differentialgleichung.
 $\dot{y} = (\alpha - \beta y)y$
 $\alpha = \beta = 5$
 ← attraktiver Fixpunkt
 $f(y) = 5(1-y)y$
 Fixpunkte: $y = 1$
 $y = 0$
 ← repulsiver Fixpunkt

exakte Lösung (2.1.7)

Bei lineare ODE $\dot{z} = A z + b \rightarrow$ spezielle Techniken. 29

Ziel: Möchte Approximation von $\underline{y}(T) = \text{exakte Lösung zur Endzeit } T$

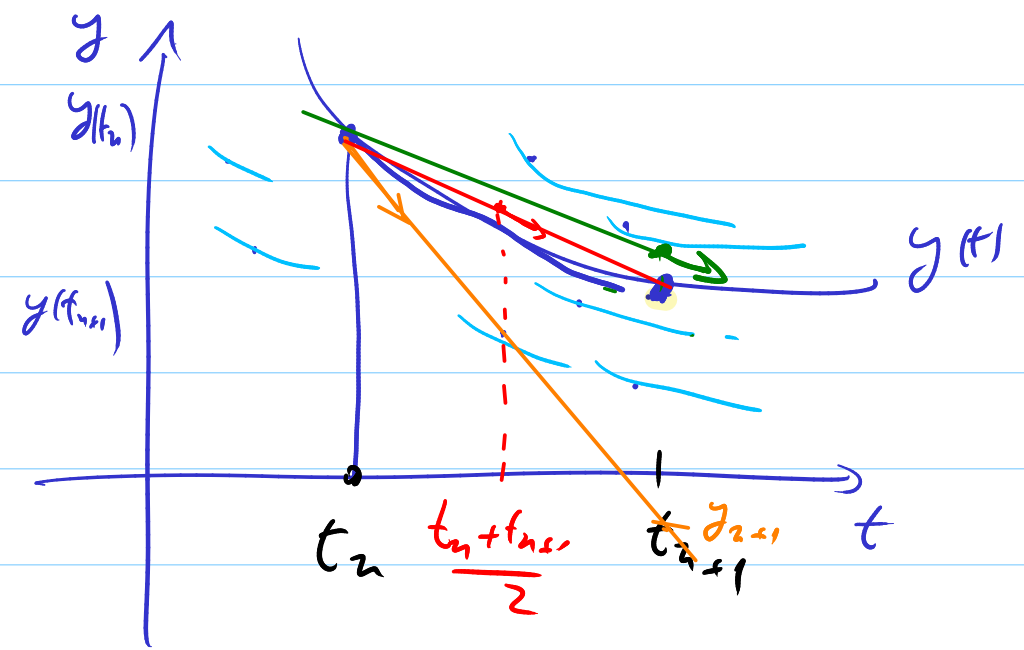
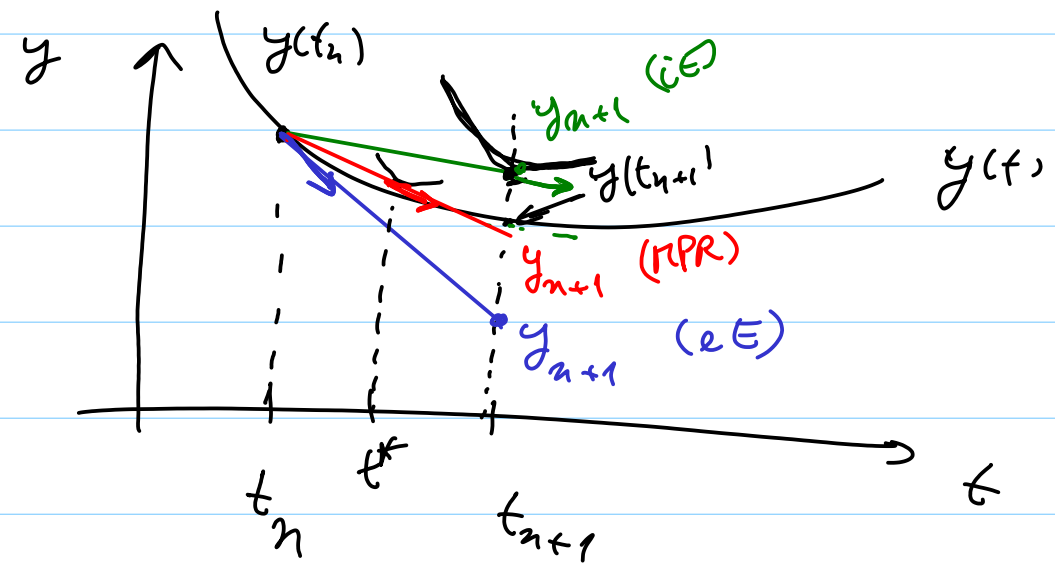


$0 = t_0 < t_1 < t_2 < \dots < t_n < t_{n+1} < \dots < t_N = T$

Zeitschritt $h_n = t_{n+1} - t_n$

Baue Approximationen $\underline{y}_n \approx \underline{y}(t_n)$

Idee lokal: linearisiere die Lösung lokal



1) Taylor um t_n für $y(t)$ ausgewertet in $t_{n+1} = t_n + h$ 2) Taylor um t_{n+1} für $y(t)$ ausgewertet in $t_n = t_{n+1} - h$

$y(t_n + h) = y(t_n) + h \dot{y}(t_n) + O(h^2)$
 $\underline{y}_{n+1} \quad \underline{y}_n \quad \underline{f}(t_n, \underline{y}(t_n))$
 \dot{y} ← aus der ODE

$y(t_{n+1} - h) = y(t_{n+1}) - h \dot{y}(t_{n+1}) + O(h^2)$
 $\underline{y}_n \quad \underline{y}_{n+1} \quad \underline{f}(t_{n+1}, \underline{y}(t_{n+1}))$
 \dot{y} ← aus der ODE

(eE) $\begin{cases} \underline{y}_{n+1} = \underline{y}_n + h \underline{f}(t_n, \underline{y}_n) \\ \underline{y}_0 = \underline{y}(t_0) \end{cases}$ expliziter Euler
lokaler Fehler $O(h^2)$
globaler Fehler $O(h)$

(iE) $\begin{cases} \underline{y}_n = \underline{y}_{n+1} - h \underline{f}(t_{n+1}, \underline{y}_{n+1}) \\ \underline{y}_0 = \underline{y}(t_0) \end{cases}$ impliziter Euler
lokaler Fehler $O(h^2)$
globaler Fehler $O(h)$

andere Herleitung

$\underline{f}(t_n, \underline{y}(t_n)) = \dot{\underline{y}}(t_n) = \lim_{h \rightarrow 0} \frac{\underline{y}(t_n + h) - \underline{y}(t_n)}{h} \approx \frac{\underline{y}_{n+1} - \underline{y}_n}{h}$

$\Rightarrow \underline{y}_{n+1} = \underline{y}_n + h \underline{f}(t_n, \underline{y}_n)$

Bez $f(t, y) = -\frac{g}{l} \sin y$

(eE) $\underline{y}_{n+1} = \underline{y}_n + h \left(-\frac{g}{l}\right) \sin \underline{y}_n$
 $\underline{y}_0 \rightarrow \underline{y}_1 \rightarrow \underline{y}_2 \rightarrow \dots \rightarrow \underline{y}_n$
 explizit

(iE) $\underline{y}_n = \underline{y}_{n+1} - h \left(-\frac{g}{l}\right) \sin \underline{y}_{n+1}$
 $\underline{y}_0 \rightarrow \underline{y}_1$ aus einer nicht-linearen Gleichung ausrechnen.
 implizit

$\underline{y}_0 = \underline{y}_1 + h \frac{g}{l} \sin \underline{y}_1 \Leftrightarrow$

$\underline{y}_1 + h \frac{g}{l} \sin \underline{y}_1 - \underline{y}_0 = 0 \Leftrightarrow F(\underline{y}_1) = 0$
 Nullstelle verwenden fsolve

Fig 1)

3) Taylor um $t^* = \frac{1}{2}(t_n + t_{n+1}) = t_n + \frac{h}{2}$ für $y(t)$

ausgewertet in $t_{n+1} = t^* + \frac{h}{2}$:

$$\underline{y}(t_{n+1}) = \underline{y}(t^*) + \frac{h}{2} \underline{\dot{y}}(t^*) + \frac{1}{2} \left(\frac{h}{2}\right)^2 \underline{\ddot{y}}(t^*) + O\left(\left(\frac{h}{2}\right)^3\right)$$

ausgewertet in $t_n = t^* - \frac{h}{2}$:

$$\underline{y}(t_n) = \underline{y}(t^*) - \frac{h}{2} \underline{\dot{y}}(t^*) + \frac{1}{2} \left(-\frac{h}{2}\right)^2 \underline{\ddot{y}}(t^*) - O\left(\left(\frac{h}{2}\right)^3\right)$$

Subtraktion \Rightarrow

$$\underline{y}(t_{n+1}) - \underline{y}(t_n) = h \underline{\dot{y}}(t^*) + O(h^3)$$

$\underline{y}_{n+1} - \underline{y}_n = h \underline{f}(t^*, \underline{y}(t^*))$

da $\underline{y}(t^*)$ stört noch, brauche noch ein Trick!

3a) Trick: $\underline{y}\left(\frac{1}{2}(t_n + t_{n+1})\right) \approx \frac{1}{2} (\underline{y}(t_n) + \underline{y}(t_{n+1})) \Rightarrow$

$\overset{?}{\underline{y}_n} \quad \overset{?}{\underline{y}_{n+1}}$

$$\underline{y}_{n+1} = \underline{y}_n + h \underline{f}\left(t_n + \frac{h}{2}, \frac{1}{2}(\underline{y}_n + \underline{y}_{n+1})\right) \quad (\text{iMP})$$

implizite Mittelpunktsregel.

lokaler Fehler $O(h^3)$

3b) Trick: $\underline{f}(t^*, \underline{y}(t^*)) \approx \frac{1}{2} (\underline{f}(t_n, \underline{y}_n) + \underline{f}(t_{n+1}, \underline{y}_{n+1}))$

$$\Rightarrow \underline{y}_{n+1} = \underline{y}_n + \frac{h}{2} (\underline{f}(t_n, \underline{y}_n) + \underline{f}(t_{n+1}, \underline{y}_{n+1})) \quad (\text{iTR})$$

implizite Trapezregel.

lokaler Fehler $O(h^3)$

$$\Rightarrow \text{global } \|\underline{y}(T) - \underline{y}_n\| \leq C \cdot h^2$$

Bei Das geht nur wenn $y(t)$ glatt genug ist.

Bez Addiere die 2 Taylor-Entwicklungen:

$$\underline{y}(t_{n+1}) + \underline{y}(t_n) = 2\underline{y}(t^*) + \left(\frac{h}{2}\right)^2 \ddot{y}(t^*) + O(h^4)$$

$$\Leftrightarrow \ddot{y}(t^*) = \frac{\underline{y}(t_{n+1}) - 2\underline{y}(t^*) + \underline{y}(t_n)}{\left(\frac{h}{2}\right)^2} + O(h^2)$$

Darum:
$$\ddot{y}\left(t_n + \frac{h}{2}\right) \approx \frac{\underline{y}_{n+1} - 2\underline{y}_{n+\frac{1}{2}} + \underline{y}_n}{\left(\frac{h}{2}\right)^2}$$
 mit lokaler Fehler $O(h^2)$

Den Selbe Rechnung um t_n ausgewertet in t_{n+1}, t_{n-1} gibt

$$\ddot{y}(t_n) \approx \frac{\underline{y}_{n+1} - 2\underline{y}_n + \underline{y}_{n-1}}{h^2}$$

 mit lokaler Fehler $O(h^2)$

§3.2. Störmer-Verlet Verfahren

Newton! Gleichung = ODE n 2. Ordnung

$$\begin{cases} \ddot{y} = f(t, \underline{y}) \\ \underline{y}(t_0) = \underline{y}_0 \\ \dot{\underline{y}}(t_0) = \underline{v}_0 \end{cases}$$

$$f(t_n, \underline{y}_n) = \ddot{y}(t_n) \approx \frac{\underline{y}_{n+1} - 2\underline{y}_n + \underline{y}_{n-1}}{h^2} \Rightarrow$$

$$\underline{y}_{n+1} = -\underline{y}_{n-1} + 2\underline{y}_n + h^2 f(t_n, \underline{y}_n)$$

 (St-V)

Störmer-Verlet

explizit, 2-Schritt-Verfahren

=> brauche Startwerte $\underline{y}(t_0) = \underline{y}_0$

$\underline{y}_1 \approx \underline{y}(t_1)$

Startwert = Taylor mit 3 Termen:

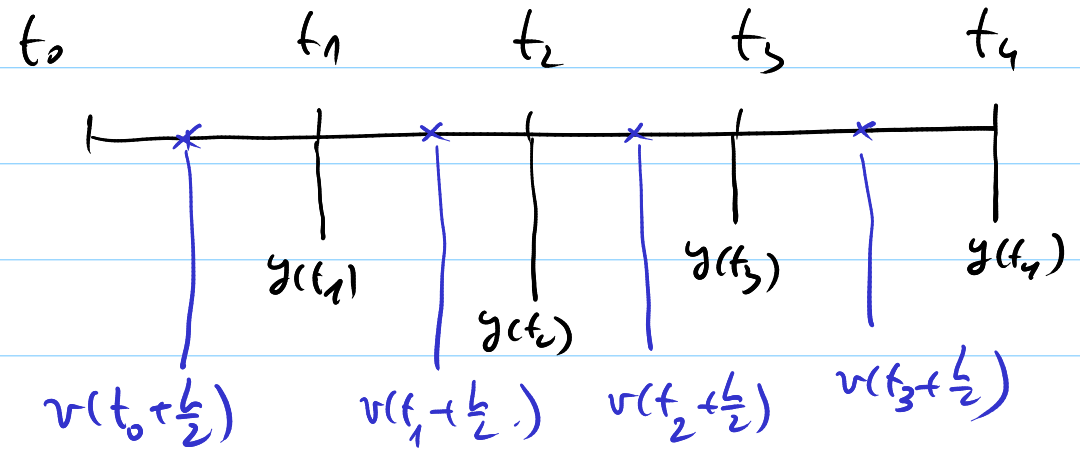
$$\underline{y}(t_1) = \underline{y}(t_0) + h \dot{\underline{y}}(t_0) + \frac{h^2}{2} \ddot{y}(t_0) + O(h^3) \Rightarrow$$

$$\underline{y}_1 = \underline{y}_0 + h \underline{v}_0 + \frac{h^2}{2} f(t_0, \underline{y}_0)$$

(St-v) lokal $O(h^3)$ und global $O(h^2)$

$$\| \underline{y}(T) - \underline{y}_N \| \leq c \cdot h^2$$

Vofiere $v_{n+\frac{1}{2}} = \frac{y_{n+1} - y_n}{h}$ entspricht $\approx v(t_n + \frac{1}{2}h) = \dot{y}(t_n + \frac{1}{2}h)$



in (St-v) : $y_{n+1} = y_n + (v_n - v_{n-1}) + h^2 f(t_n, y_n)$

$$y_{n+1} = y_n + h \left(\frac{v_n - v_{n-1}}{h} + h f(t_n, y_n) \right)$$

$$y_{n+1} = y_n + h \underbrace{\left(v_{n-\frac{1}{2}} + h f(t_n, y_n) \right)}_{v_{n+\frac{1}{2}}}$$

Leap-Frog:

$$\begin{cases} v_{n+\frac{1}{2}} = v_{n-\frac{1}{2}} + h f(t_n, y_n) \\ y_{n+1} = y_n + h v_{n+\frac{1}{2}} \end{cases}$$



=> versetzte Gitter für Position \underline{y} und Geschwindigkeit \underline{v}
"staggered grids"

Besser noch "Velocity-Verlet"-Methode:

$$\begin{cases} y_{n+1} = y_n + h v_n + \frac{h^2}{2} f(t_n, y_n) \\ v_{n+1} = v_n + h \frac{1}{2} \left(f(t_n, y_n) + f(t_n, y_{n+1}) \right) \end{cases}$$



Beweise die Herleitung von (VV) aus (StV)

(StV) $y_{n+1} = -y_{n-1} + 2y_n + h^2 f(t_n, y_n)$

Notiere $v_n = \frac{y_{n+1} - y_{n-1}}{2h} \Rightarrow y_{n+1} - y_{n-1} = 2h v_n$

$2y_{n+1} = 2y_n + 2h v_n + h^2 f(t_n, y_n) \Rightarrow y_{n+1} = y_n + h v_n + \frac{h^2}{2} f(t_n, y_n)$

Update von v_n ?

$v_n = \frac{y_{n+1} - y_{n-1}}{2h} = \frac{2y_n - 2y_{n-1} + h^2 f(t_n, y_n)}{2h} = \frac{y_n - y_{n-1}}{h} + \frac{h}{2} f(t_n, y_n) \Rightarrow$

$v_n = \frac{y_n - y_{n-1}}{h} + \frac{h}{2} f(t_n, y_n)$

$v_{n+1} = \frac{y_{n+1} - y_n}{h} + \frac{h}{2} f(t_{n+1}, y_{n+1})$

$v_n + v_{n+1} = \frac{y_{n+1} - y_{n-1}}{h} + \frac{h}{2} [f(t_n, y_n) + f(t_{n+1}, y_{n+1})] \Rightarrow$

$\Rightarrow v = v + \frac{h}{2} [f(t_n, y_n) + f(t_{n+1}, y_{n+1})]$

Vorteile von VV:

- + stabiler als (St-V)
- + $y_n \approx y(t_n)$, $v_n \approx \dot{y}(t_n)$ auf demselben Zeitgitter.
- + global $O(h^2)$ also genauer als (eE), (iE)
- + explizit (vergleiche iMP, iTR)
- + erhält die Energie

Beweis für die Konvergenzordnung: FS2019.

$$\dot{u}(t) = f(t, u)$$

$$y_1 = t \Rightarrow \dot{y}_1 = 1$$

$$y_2 = u \Rightarrow \dot{y}_2 = \dot{u} = f(y_1, y_2)$$

$$\underline{\dot{y}} = \underline{F}(\underline{y}) \quad \text{mit} \quad \underline{F} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{bmatrix} 1 \\ f(y_1, y_2) \end{bmatrix}$$

autonom!

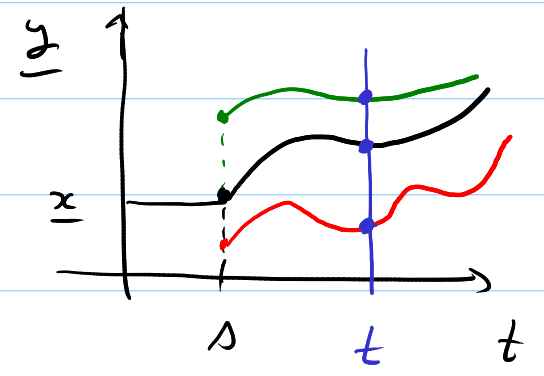
§3.3 Splitting Verfahren

19.03.2024

→ Autonome ODEs

Bem $\dot{y} = f(t, y)$

$y(1) = \underline{x} \in \mathbb{R}^d$



Notiere $\Phi^{s,t}(\underline{x}) = \underline{y}(t)$ mit $y(1) = \underline{x}$

$\Phi^{s,t} : D \rightarrow D$ zweiparametrische Familie von Abbildungen $\mathbb{R}^d \rightarrow \mathbb{R}^d$

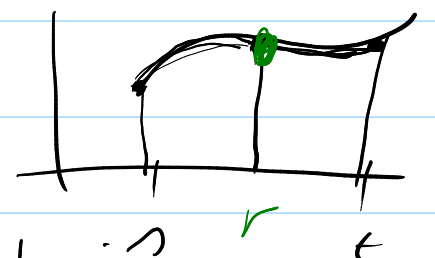
Fluss der ODE

1) $\Phi^{t,t}(\underline{x}) = \underline{x}$ für alle \underline{x}

2) $\Phi^{s,t} = \Phi^{r,t} \circ \Phi^{s,r}$

3) autonome ODEs sind invariant:

$\Phi^{s,t} = \Phi^{0,t-s} = \Phi^{t-s}$
notation



Beweis Sei $t^* \in \mathbb{R}$ fest. Translation in der Zeit:

$\underline{u}(t) = \underline{y}(t + t^*)$

Berechne $\frac{d}{dt} \underline{u}(t) = \dot{\underline{y}}(t + t^*) \cdot 1 = \underline{f}(\underline{y}(t + t^*)) = \underline{f}(\underline{u}(t))$

$\Rightarrow \dot{\underline{u}} = \underline{f}(\underline{u})$

Idee vom Splitting: kombiniere Flüsse verschiedener ODEs

Wie? Begründung für lineare ODEs:

Bsp $e^{\lambda h} = \sum_{n=0}^{\infty} \frac{1}{n!} (\lambda h)^n$ Exponentialfunktion

auch als die Lösung der ODE:

$\dot{y}(t) = \lambda y(t)$ mit $y(0) = 1$

Für mehrere lineare entkoppelten ODEs:

$\dot{y}_1(t) = \lambda_1 y_1(t) \Rightarrow y_1(h) = e^{\lambda_1 h} \quad y_1(0) = 1$

$\dot{y}_2(t) = \lambda_2 y_2(t) \Rightarrow y_2(h) = e^{\lambda_2 h}$

$\dot{y}_d(t) = \lambda_d y_d(t) \Rightarrow y_d(h) = e^{\lambda_d h} \quad y_d(0) = 1$

$$\underline{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix}; \quad \underline{\dot{y}} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_d) \underline{y}$$

mit Lösung

$$\underline{y}(h) = \text{diag}(e^{\lambda_1 h}, e^{\lambda_2 h}, \dots, e^{\lambda_d h}) \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

!! Notation!

$$e^{h \text{diag}(\lambda_1, \dots, \lambda_d)}$$

$$e^{h \text{diag}(\lambda_1, \dots, \lambda_d)} = \sum_{n=0}^{\infty} \frac{1}{n!} (\text{diag}(\lambda_1, \dots, \lambda_d))^n$$

Verallgemeinert: $\underline{\dot{y}} = \underline{M} \underline{y}$ mit $\underline{M} \in \mathbb{R}^{d \times d}$

mit Lösung $\underline{y}(t) = e^{\underline{M}t} \underline{y}_0$

alternativ

$$e^{\underline{M}t} = \sum_{n=0}^{\infty} \frac{1}{n!} (\underline{M}t)^n$$

NICHT für numerische Berechnungen.

Numerisch: "expm" = Padé-Approximation $O(d^3)$

$$e^{nh} \approx \frac{P_n(h)}{Q_n(h)}$$

Oder Krylov-Verfahren.



auch nicht so: EW, EV !!

$$\underline{M} \text{ sym.} \quad \underline{M} = \underline{Q} \underline{D} \underline{Q}^T \Rightarrow e^{nh} = \underline{Q} e^{\underline{D}h} \underline{Q}^T$$

Bsp $\underline{M} = \underline{A} + \underline{B}$

$$e^{(\underline{A} + \underline{B})h} = \underline{I} + (\underline{A} + \underline{B})h + \frac{1}{2} (\underline{A}^2 + \underline{A}\underline{B} + \underline{B}\underline{A} + \underline{B}^2)h^2 + \dots$$

$$e^{\underline{A}h} e^{\underline{B}h} = (\underline{I} + \underline{A}h + \frac{1}{2} \underline{A}^2 h^2 + \dots) (\underline{I} + \underline{B}h + \frac{1}{2} \underline{B}^2 h^2 + \dots) =$$

$$= \underline{I} + (\underline{A} + \underline{B})h + (\frac{1}{2} \underline{A}^2 + \underline{A}\underline{B} + \frac{1}{2} \underline{B}^2)h^2 + \dots$$

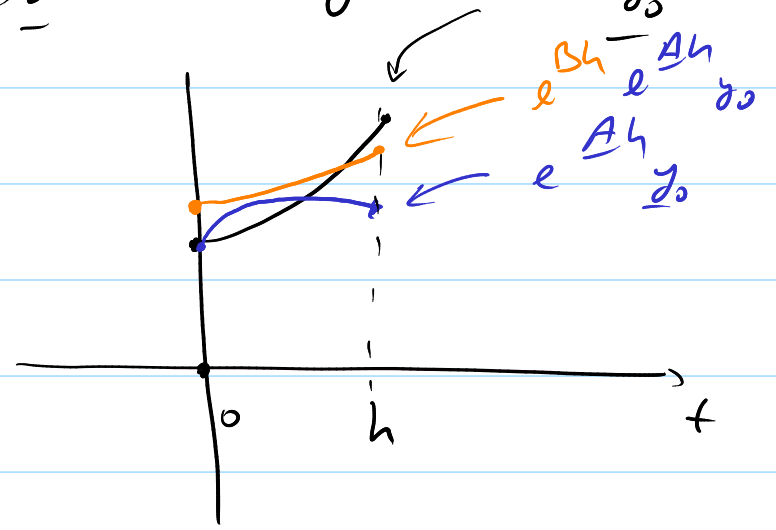
$$\underline{A}\underline{B} \neq \underline{B}\underline{A} \Rightarrow e^{(\underline{A} + \underline{B})h} \neq e^{\underline{A}h} e^{\underline{B}h}$$

aber man kann erweisen dass

$$(e^{(A+B)h} - e^{Ah} e^{Bh}) \approx O(h^2)$$

↑
lokaler Fehler, global zu T $\Rightarrow O(h)$
 $(e^{(A+B)h} - e^{Bh} e^{Ah}) \approx O(h^2)$

$\dot{y} = (A+B)y$ mit Lösung $e^{(A+B)t} y_0$



$$\dot{y} = Ay$$

$$\dot{y} = By$$

SPLITTING!

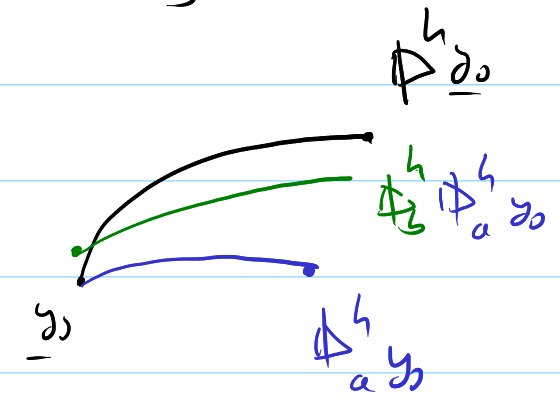
Ben Für autonome ODE

$$(i) \dot{y} = f(y) \text{ mit } f(y) = f_a(y) + f_b(y)$$

Idee wähle f_a und f_b so dass

$$(a) \dot{y} = f_a(y) \text{ und.}$$

$$(b) \dot{y} = f_b(y)$$



einfach oder exakt lösbar sind.

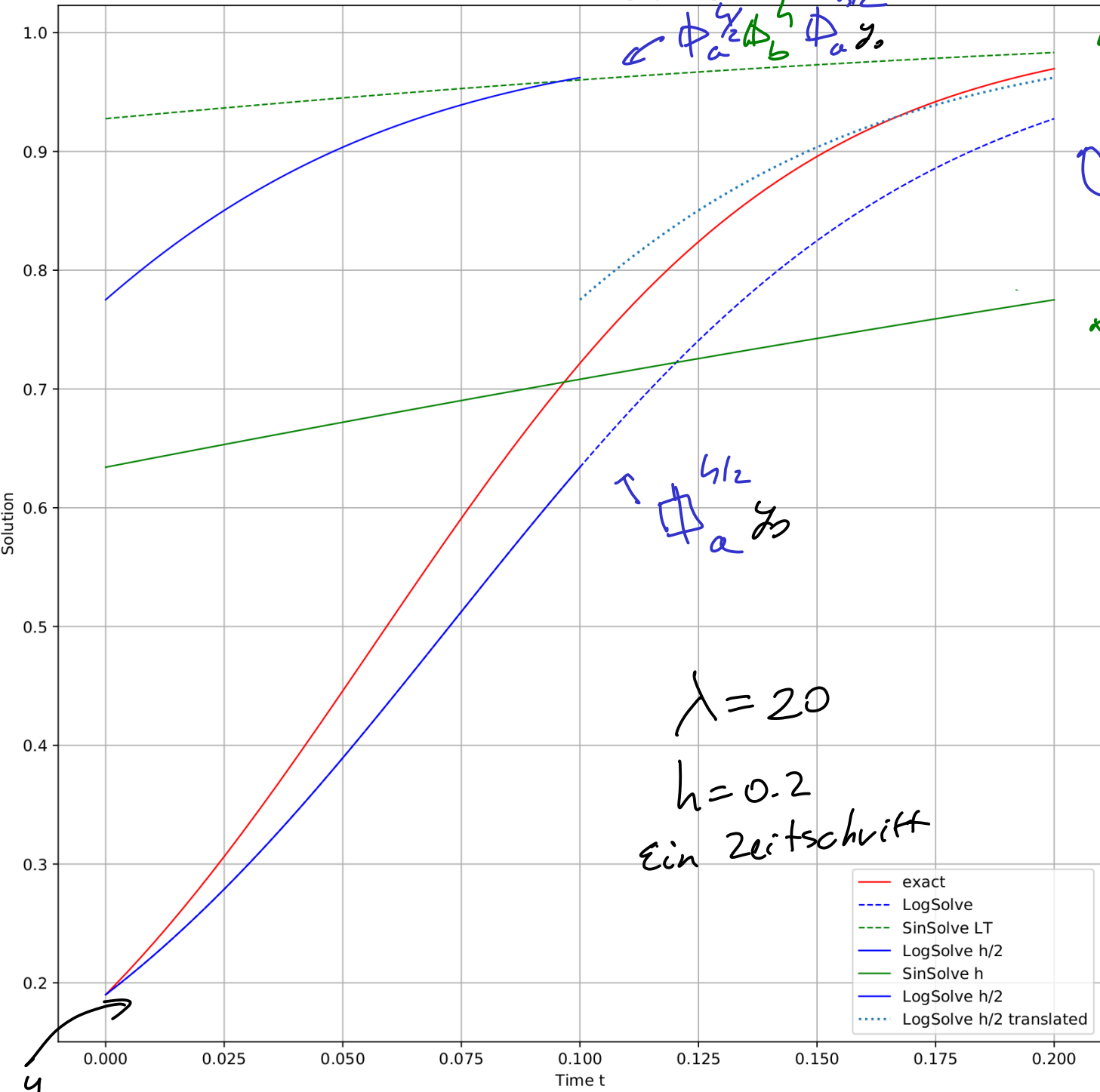
$$\Psi_1^h = \Phi_{y_0}^h \Phi_a^h \approx \Phi^h \text{ Lie-Trotter Splitting.}$$

Man kann beweisen

$$\|\underbrace{\Psi_1^h \dots \Psi_1^h}_{\text{mal } N} y_0 - \Phi^T y_0\| \leq c \cdot h$$

mal $Nh = T$
lokal $\|\Psi_1^h y_0 - \Phi^h y_0\| \leq c \cdot h^2$

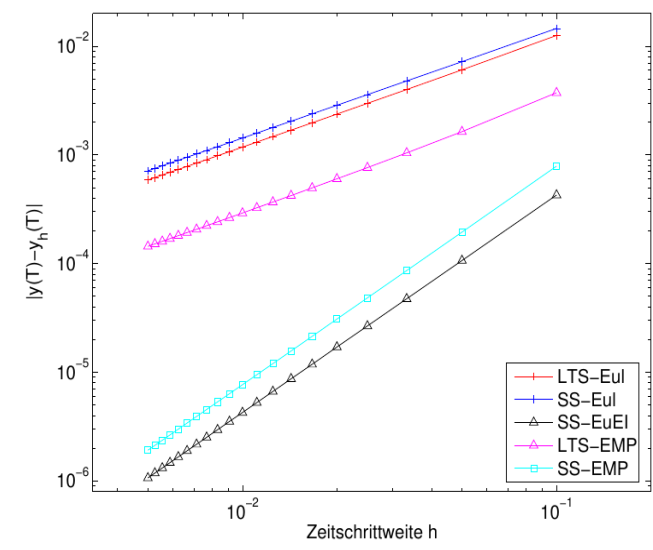
Lie-Trotter vs Strang Splitting



Lie-Trotter-Lösung = $\Phi_b^h \Phi_a^h y_0$
 "exacte" Lösung $\Phi^{(t)} y_0$
 Strang-Lösung = $\Phi_a^{h/2} \Phi_b^h \Phi_a^{h/2} y_0$

Wenn (a) oder (b) nicht exakt lösbar, dann nimmt man numerische Methoden dafür.

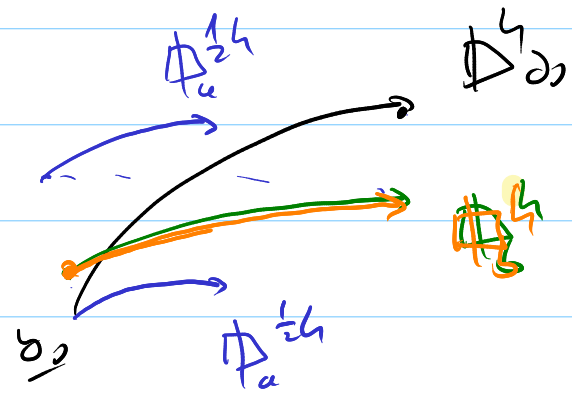
Bsp



- LTS-Eul explizites Euler als $\Psi_{a,b}^h$, $\Psi_{a,b}^h$ und Lie-Trotter-Splitting
- SS-Eul explizites Euler als $\Psi_{a,b}^h$, $\Psi_{a,b}^h$ und Strang-Splitting
- SS-EuEI Strang-Splitting: explizites Euler als $\Psi_a^{h/2}$, exaktes Φ_b^h und implizites Euler als $\Psi_a^{h/2}$
- LTS-EMP explizite Mittelpunkt-Regel als $\Psi_{a,b}^h$, $\Psi_{a,b}^h$ und Lie-Trotter-Splitting
- SS-EMP explizite Mittelpunkt-Regel als $\Psi_{h,g}^h$, $\Psi_{h,f}^h$ und Strang-Splitting

Abb. 2.4.5. Einfache Splitting-Verfahren

Bez Symmetrie schenkt uns eine Ordnung mehr:



Strong splitting

$$\Phi^h \approx \Phi_a^{1/2 h} \Phi_u^h \Phi_a^{1/2 h}$$

lokal $O(h^3)$
 global $O(h^2)$

Allgemeines Splittingverfahren.

$$\Psi^h = \prod_{i=1}^D \Phi_{b_i}^{b_i h} \Phi_a^{a_i h} \quad \sum_{i=1}^D a_i = 1 = \sum_{i=1}^D b_i$$

$D=1$ $a_1 = b_1 = 1$ Lie-Trotter

$D=2$ $a_1 = a_2 = \frac{1}{2}$, $b_1 = 1$, $b_2 = 0$ Strong.

- Code 4.4.9: Einige Splitting Routinen
- Code 4.4.11: Energieerhaltung beim Splitting Verfahren
- Code 4.4.13: Fehler der Splitting Verfahren

Bsp Splitting Verfahren für Newton' Gleichung.

$$\ddot{r} = \underline{a}(r) \Leftrightarrow \dot{z} = \begin{bmatrix} \dot{r} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ a(r) \end{bmatrix} =: \underline{F}(z)$$

$$\underline{F}(z) = \underline{F} \left(\begin{bmatrix} r \\ v \end{bmatrix} \right) = \begin{bmatrix} v \\ a(r) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 \\ a(r) \end{bmatrix}}_f + \underbrace{\begin{bmatrix} v \\ 0 \end{bmatrix}}_g$$

$$\dot{z} = \boxed{f(z)} + \boxed{g(z)}$$

Startwert $z_0 = \begin{bmatrix} r_0 \\ v_0 \end{bmatrix}$

(a) $\dot{z} = f(z) \Leftrightarrow \begin{cases} \dot{r} = 0 \\ \dot{v} = a(r) \end{cases}$ Löse "exakt" von $t=0$ zu h

$$\dot{r} = 0 \Rightarrow r(t) = \text{konstant} = r_0 \Rightarrow r(h) = r(0) = r_0$$

$$\dot{v} = a(r_0) \Rightarrow \dot{v}(t) = a(r_0) \Rightarrow v(t) = a(r_0)t + v_0$$

$$\Rightarrow v(h) = a(r_0)h + v_0$$

EXAKT 😊

$$\Phi_f^h \begin{bmatrix} r_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} r_0 \\ a(r_0)h + v_0 \end{bmatrix}$$

(b) $\dot{y} = g(y) \quad \left\{ \begin{array}{l} \dot{r} = v \\ \dot{v} = 0 \Rightarrow v(t) = v_0 = v_0 \end{array} \right. \Rightarrow$

$\dot{r} = v_0 \Rightarrow r(t) = v_0 t + r_0 \quad \text{EXAKT } \text{☺}$

$$\Phi_g^h \begin{bmatrix} r_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} v_0 h + r_0 \\ v_0 \end{bmatrix}$$

Kombiniere diese 2 exakten Lösungen:

(1) Lie-Trotter

$$\Psi^h \begin{bmatrix} r \\ v \end{bmatrix} = \Phi_g^h \Phi_f^h \begin{bmatrix} r \\ v \end{bmatrix} = \Phi_g^h \begin{bmatrix} r \\ a(r)h + v \end{bmatrix} = \begin{bmatrix} r + (a(r)h + v)h \\ a(r)h + v \end{bmatrix}$$

symplektisches Euler-Verfahren $g \text{losa}(O(h))$

→ diese Zerlegung separiert 2 Variablen.

\underline{f} hängt effektiv nur von r ab $\left. \begin{array}{l} \underline{v} \text{ ist} \\ \underline{g} \end{array} \right\} \Rightarrow$

⇒ EXAKTE LÖSUNGEN!

(2) Strong-Splitting.

$$\begin{aligned} \Psi^h \begin{bmatrix} r \\ v \end{bmatrix} &= \Phi_g^{\frac{h}{2}} \Phi_f^h \Phi_g^{\frac{h}{2}} \begin{bmatrix} r \\ v \end{bmatrix} = \Phi_g^{\frac{h}{2}} \Phi_f^h \begin{bmatrix} r + v \frac{h}{2} \\ v \end{bmatrix} = \\ &= \Phi_g^{\frac{h}{2}} \begin{bmatrix} r + v \frac{h}{2} \\ v + h a(r + v \frac{h}{2}) \end{bmatrix} = \\ &= \begin{bmatrix} r + v \frac{h}{2} + h \left(\underbrace{v + h a(r + v \frac{h}{2})}_{r_{k+\frac{1}{2}}} \right) \\ v + h a(r + v \frac{h}{2}) \end{bmatrix} \end{aligned}$$

Notiere

$$\begin{cases} r_{k+\frac{1}{2}} = r_k + \frac{1}{2} h v_k \\ v_{k+1} = v_k + h a(r_{k+\frac{1}{2}}) \\ r_{k+1} = r_k + \frac{h}{2} v_{k+1} \end{cases}$$

ein-Schritt-Formulierung des St-Venets Verfahrens!

Ben Trick geht genau so für
separablen Hamilton'system:

$$H(\underline{p}, \underline{q}) = T(\underline{p}) + V(\underline{q})$$

Gesucht $\underline{p}, \underline{q}: [0, T] \rightarrow \mathbb{R}$

$$\begin{cases} \dot{p}_j = -\frac{\partial H}{\partial q_j}(\underline{p}(t), \underline{q}(t)) & \text{autonomes} \\ & \text{Ho'-system,} \\ \dot{q}_j = \frac{\partial H}{\partial p_j}(\underline{p}(t), \underline{q}(t)) & \text{mit Hamiltonfunktion } H \end{cases}$$

für $j=1, 2, \dots, d$

Ben $H(\underline{p}, \underline{q})$ ist Invariante für
das Ho'system!

Ben Leicht gestörte Probleme:

$$\dot{y} = f_a(y) + \underbrace{\varepsilon f_b(y)}_{\text{Perturbation}} \quad \text{mit } \varepsilon \text{ klein}$$

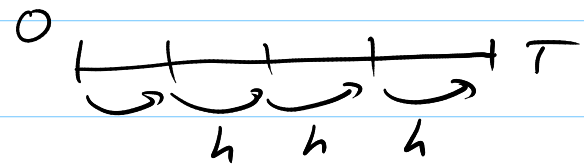
↓
 dominant

optimierte Splitting-Verfahren
 $O(\varepsilon h^{r_1} + \varepsilon^2 h^{r_2} + \varepsilon^3 h^{r_3} + \dots)$
 mit $r_1 > r_2 + 1 > \dots$

z.B.: $O(\varepsilon h^4 + \varepsilon^2 h^2)$

Achtung: Perturbiertes Problem, nicht-autonom.
 in Splitting: t zu f_a

Ben Splitting ist explizit
 Sobald wir Ordnung > 2 haben wollen,
 gibt es negative Parameter a_i, b_i

Processing Methoden

die: rechne mit der billigeren Methode Ψ
 2-Teilschritt und verwende $\bar{a}^h, (\pi^h)^{-1}$ nur
 ein Mal (oder bei Ausgaben)

$$\hat{\Psi}^h \stackrel{\text{def}}{=} \pi^h \circ \Psi^h \circ (\pi^h)^{-1}$$

↳ post-processor ↳ pre-processor

Bsp Strong-Splitting:

$$\begin{aligned}
 (\hat{\Psi}^h)^n &= \pi^h \circ \Psi^h \circ (\pi^h)^{-1} \circ \pi^h \circ \Psi^h \circ (\pi^h)^{-1} \circ \dots \circ \pi^h \circ \Psi^h \circ (\pi^h)^{-1} \\
 &= \pi^h \circ (\Psi^h)^n \circ (\pi^h)^{-1}
 \end{aligned}$$

$$\Psi^h = \Phi_a^{\frac{h}{2}} \Phi_b^h \Phi_a^{\frac{h}{2}} \cdot \mathbb{I} = \Phi_a^{\frac{h}{2}} \underbrace{(\Phi_b^h \Phi_a^h)}_{\text{Lie-Trotter}} (\Phi_a^{\frac{h}{2}})^{-1}$$

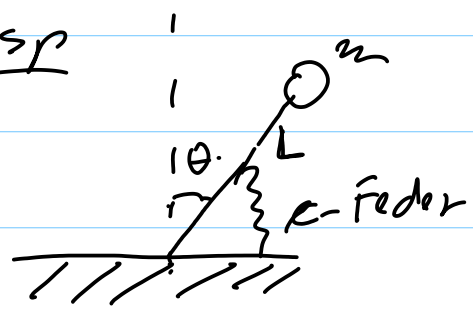
Vorteile falls:

+ $\hat{\Psi}^h$ genauer als Ψ^h

+ $\pi^h, (\pi^h)^{-1}$ günstig.

+ keine/wenige Ausgaben der Lösung vor T nötig.

Bsp



$$\ddot{\theta} = -\frac{c}{mL^2} \theta + \left[\frac{g}{L} \sin \theta \right]$$

A B

$$\ddot{\theta} = A\theta + B \sin \theta$$

1) Hamilton's system?

$$q = \theta, \quad p = \dot{\theta}$$

$$\begin{cases} \dot{q} = p \\ \dot{p} = Aq + B \sin q \end{cases} \quad \begin{aligned} &= \frac{\partial H}{\partial p} \\ &= -\frac{\partial H}{\partial q} \end{aligned}$$

$$H(q, p) = \frac{1}{2} p^2 - \frac{A}{2} q^2 + B \cos q = \frac{1}{mL^2} E_{\text{tot}}$$

$$E_{\text{tot}} = E_{\text{kin}} + E_{\text{pot}}, \quad E_{\text{kin}} = \frac{1}{2} m L^2 \dot{\theta}^2$$

$$V(q) = \frac{1}{2} c q^2 + g m L \cos(q)$$

$$(1) \begin{bmatrix} p \\ Aq + B \sin q \end{bmatrix} = \begin{bmatrix} p \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ Aq + B \sin q \end{bmatrix}$$

$$(a) \begin{cases} \dot{q} = p \\ \dot{p} = 0 \end{cases} \Rightarrow p(t) = p_0 \Rightarrow \dot{q} = p_0 \Rightarrow q(t) = q_0 + t p_0$$

$$\Phi_a^t \begin{bmatrix} q_0 \\ p_0 \end{bmatrix} = \begin{bmatrix} q_0 + t p_0 \\ p_0 \end{bmatrix}$$

$$(b) \begin{cases} \dot{q} = 0 \\ \dot{p} = Aq + B \sin q \end{cases} \Rightarrow q(t) = q_0 \downarrow p(t) = p_0 + (Aq_0 + B \sin q_0) t$$

$$\Phi_b^t \begin{bmatrix} q_0 \\ p_0 \end{bmatrix} = \begin{bmatrix} q_0 \\ p_0 + (Aq_0 + B \sin q_0) t \end{bmatrix}$$

```

58 def phiA(dt, u):
59     r""" Operator A: Phi_A(q,p)
60     INPUT:
61         dt :           time step
62         u  :           solution at the known time level t=t_n, i.e. y_n.
63                       3d array : u[0] = angle, u[1] = angle velocity, u[2]
64     =time
65     OUTPUT:
66         v  :           solution at time u[2]+dt
67     """
68     v = np.array([0.,0.,0.])
69     t = 1.*u[2]
70     p = 1.*u[1]
71     q = u[0] + dt*p
72     v = np.array([q,p,t])
73     return v
74
75 def phiB(dt, u):
76     r""" Operator B: Phi_B(q,p)
77     INPUT:
78         dt :           time step
79         u  :           solution at the known time level t=t_n, i.e. y_n.
80                       3d array : u[0] = angle, u[1] = angle velocity, u[2]
81     =time
82     OUTPUT:
83         u  :           solution at time u[2]+dt
84     """
85     v = np.array([0.,0.,0.])
86     t = u[2] + dt
87     q = 1.*u[0]
88     p = u[1] + dt*(A*q+B*np.sin(q))
89     v = np.array([q,p,t])
90     return v
91
92 def Approximate(T, N, y0, method="LT") :
93
94     tspan = [0., T]
95     ys = np.array([y0[0],y0[1],0.])
96     a,b = S.build(method)
97     t, y = S.intsplit(phiA, phiB,a,b,tspan,N,ys,getall=True)
98     return t, y
99

```

```

.f __name__ == "__main__":
    N_SV = 5000
    N_6 = 2500

    y0 = np.array([theta_0, 0])

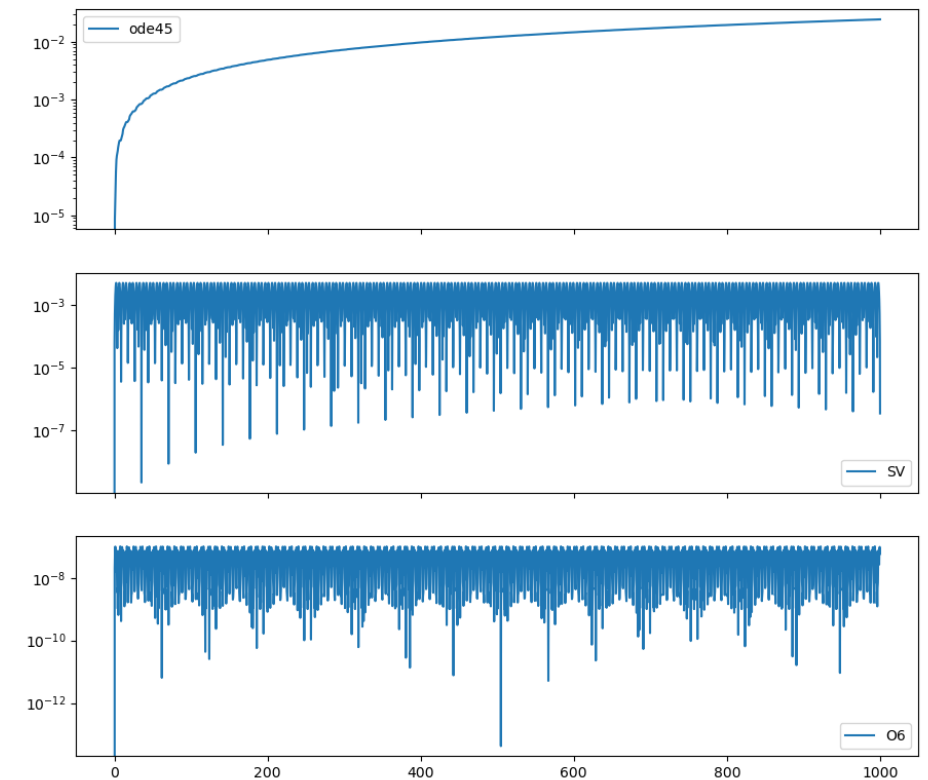
    # #For comparison at sub-task d: uncomment once you have implemented odefun!
    t_45, y_45 = ode45(odefun,(t0,T), y0)
    pE_45, kE_45, tE_45 = Energien(y_45[:,0], y_45[:,1])

    # SV
    #t_SV, y_SV = np.linspace(t0,T,N_SV), 2*np.ones((N_SV,2))*y0 #TODO: Compute t_SV and y_SV
    method_ss = 'SS'
    t_SV, y_SV = Approximate(T, N_SV, y0, method=method_ss)
    pE_SV, kE_SV, tE_SV = Energien(y_SV[:,0], y_SV[:,1])

    # order 6
    #t_6, y_6 = np.linspace(t0,T,N_6), 2*np.ones((N_6,2))*y0 #TODO: Compute t_6 and y_6
    method_2 = 'BM63'
    t_6, y_6 = Approximate(T, N_6, y0, method=method_2)

    pE_6, kE_6, tE_6 = Energien(y_6[:,0], y_6[:,1])

```



Finde $u: [0, T] \times [0, 1] \rightarrow \mathbb{R}$ so dass

$$\begin{cases} \frac{\partial u}{\partial t}(t, x) + c(x) \frac{\partial u}{\partial x}(t, x) = 0 \\ u(0, x) = \exp(-100(2\pi x - 1)^2) \end{cases}$$

mit

$$c(x) = 0.2 + \sin(2\pi x - 1)^2$$

Bez Glatte Anfangswert, alles periodisch.

\Rightarrow Verwende Fourier Ansatz in Ort x :

$$u(t, x) = \sum_{k=-\infty}^{\infty} z_k(t) e^{-2\pi i k x} \Rightarrow$$

$$\frac{\partial u}{\partial x}(t, x) = \sum_{k=-\infty}^{\infty} z_k(t) (-2\pi i k) e^{-2\pi i k x}$$

$k \in \mathbb{Z}$ $k \in \{-\frac{N}{2}, \dots, 1, 0, 1, \dots, \frac{N}{2}-1\}$

Collokation: möchte dass die Dgl. in den Punkten

(t_n, x_k) erfüllt ist

$$(N=2^n)$$

Nehme N äquidistante Punkte in $[0, 1]$

$$x_0 = 0, x_1 = \frac{1}{N}, \dots, x_{N-1} = \frac{N-1}{N} \text{ (Ortsgitter)}$$

$$\underline{x} = [x_0, x_1, \dots, x_{N-1}]^T \in \mathbb{R}^N$$

$$u(t, \underline{x}) = [u(t, x_0), u(t, x_1), \dots, u(t, x_{N-1})]^T \in \mathbb{R}^N$$

$$c(\underline{x}) = [c(x_0), c(x_1), \dots, c(x_{N-1})]^T \in \mathbb{R}^N$$

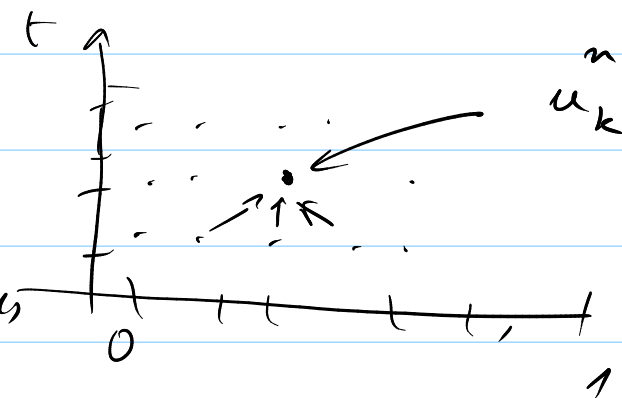
$$\frac{\partial}{\partial t} u(t, \underline{x}) + c(\underline{x}) \underline{F}^{-1} \underline{D} \underline{F} u(t, \underline{x}) = 0$$

$$\underline{D} = \text{diag}(-2\pi i k) \quad N \times N$$

Vergleiche mit

Finite Differenzen

\hookrightarrow lokale Effekte



1) Leap-frog / zentraler Differenz

$$= c(x) \left(c'(x) \frac{\partial u}{\partial x} + c(x) \frac{\partial^2 u}{\partial x^2} \right) \Rightarrow \left\{ \begin{array}{l} \text{Nur 1. Elliptisch. DG.} \\ \text{hyperbol. DG} \end{array} \right.$$

$$\frac{u(t_{n+1}, x) - u(t_{n-1}, x)}{2h} = -c(x) \mathcal{F}^{-1} \underline{\underline{D}} \mathcal{F} u(t_n, x) \Rightarrow$$

Lax-Wendroff:

$$u(t_{n+1}, x) = u(t_{n-1}, x) - 2h c(x) \mathcal{F}^{-1} \underline{\underline{D}} \mathcal{F} u(t_n, x)$$

$$u(t+h, x) = u(t, x) - h c(x) \mathcal{F}^{-1} \underline{\underline{D}} \mathcal{F} u(t, x) + \frac{h^2}{2} c(x) \left(c'(x) \mathcal{F}^{-1} \underline{\underline{D}} \mathcal{F} u(t, x) + c(x) \mathcal{F}^{-1} \underline{\underline{D}}^2 \mathcal{F} u(t, x) \right)$$

einfach!

Brauche $u(t_0 - h) \approx u(0, x - 0.2h)$



besser: eine Methode $O(h^2)$ für $u(t_1)$ (später)
eTR

Alternative Finite Differenzen

2) Lax-Wendroff



technik von Cauchy-Kowalevsky:

$$u(t_n, x_k) = u_k^n \quad c_k = c(x_k)$$

$$y(t+h) = y(t) + h\dot{y}(t) + \frac{h^2}{2} \ddot{y}(t) + O(h^3)$$

upwind:

$$\frac{u_k^{n+1} - u_k^n}{h} + \bar{c}_k \frac{u_k^n - u_{k-1}^n}{\Delta x} = 0$$

$$\frac{\partial u}{\partial t} = -c(x) \frac{\partial u}{\partial x}$$

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial t} \left(-c(x) \frac{\partial u}{\partial x} \right) = -c(x) \frac{\partial}{\partial x} \left(\frac{\partial u}{\partial t} \right) = -c(x) \frac{\partial}{\partial x} \left(-c(x) \frac{\partial u}{\partial x} \right) =$$

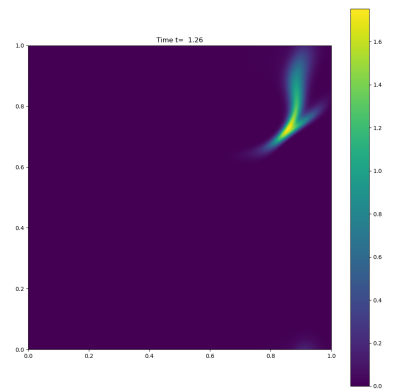
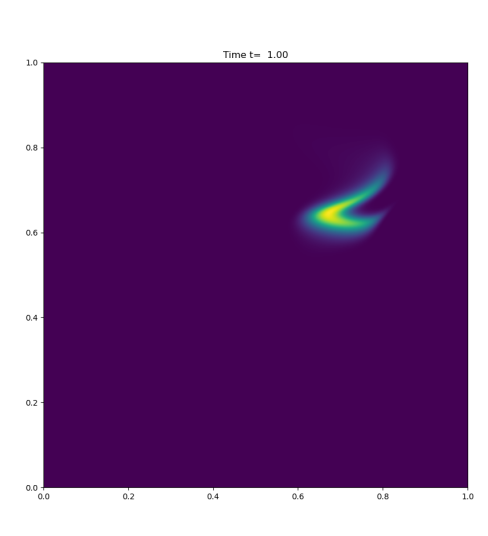
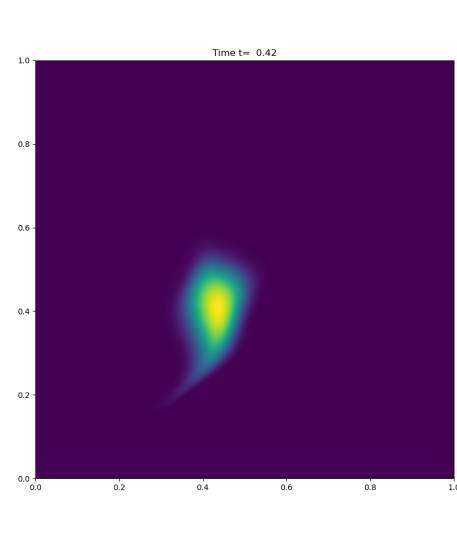
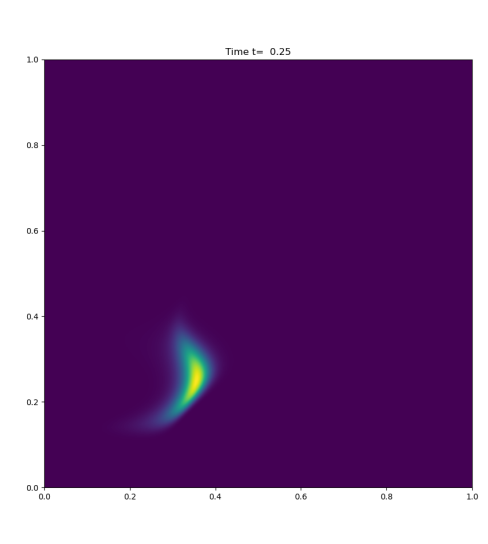
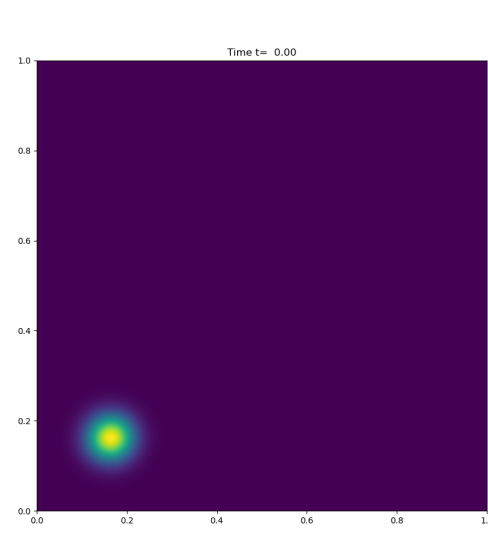
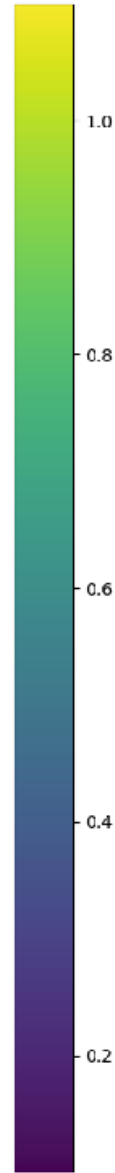
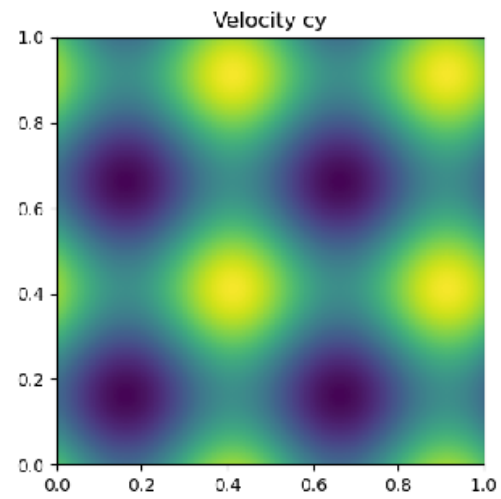
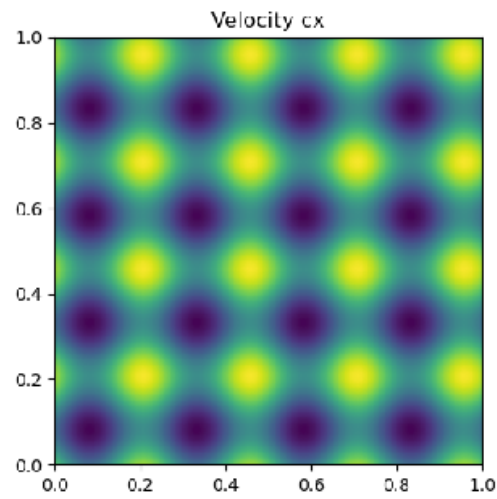
(dD):

mit $\bar{c}_k = \frac{1}{2} (c_k + c_{k-1})$ Mittelwert

Finite Elemente!



wave2d02.py
dataEnv16AsymDicht



2 mehrere Dimensionen

Finde:

$$\begin{cases} u: [0, T] \times \mathbb{R}^d \rightarrow \mathbb{R} \\ u(0, \underline{x}) = u_0(\underline{x}) \end{cases} \quad \text{so dass} \quad \underline{c}: \mathbb{R}^d \rightarrow \mathbb{R}^d \quad \underline{c}(\underline{x}) = \begin{bmatrix} c_1(\underline{x}) \\ \vdots \\ c_d(\underline{x}) \end{bmatrix}$$

$$\frac{\partial u}{\partial t} + \sum_{j=1}^d c_j(\underline{x}) \frac{\partial u}{\partial x_j} = 0 \quad (\Rightarrow) \quad \frac{\partial u(\underline{x})}{\partial t} + \underline{c}(\underline{x}) \cdot \underline{\text{grad}}_{\underline{x}} u(t, \underline{x}) = 0$$

$$\underline{c}: \mathbb{R}^d \rightarrow \mathbb{R}^d \quad \underline{\underline{Dc}} = \left[\frac{\partial c_j}{\partial c_l} \right]_{j,l=1,\dots,d} \in \mathbb{R}^{d \times d}$$

$$= + \underline{c}(\underline{x}) \cdot \underline{\text{grad}}_{\underline{x}} \left(\underline{c}(\underline{x}) \cdot \underline{\text{grad}}_{\underline{x}} u(t, \underline{x}) \right) =$$

$$= \underline{c}(\underline{x}) \cdot \left(\underline{\underline{Dc}} \cdot \underline{\text{grad}}_{\underline{x}} u(t, \underline{x}) + \underline{c}(\underline{x}) \Delta u(t, \underline{x}) \right)$$

Stokes

$$\sum_{l=1}^d \left(\sum_{j=1}^d c_j \frac{\partial c_l}{\partial x_j} \right) \cdot \frac{\partial u}{\partial x_l} \quad \underline{\underline{\underline{\text{grad}} u \cdot (\underline{\underline{Dc}} \cdot \underline{c})}}$$

$$\sum_{j=1}^d \frac{\partial^2 u}{\partial x_j^2}$$

Taylor in t:

$$u(t+h) = u(t) + h \frac{\partial u}{\partial t}(t, \underline{x}) + \frac{h^2}{2} \frac{\partial^2 u}{\partial t^2}(t, \underline{x}) + O(h^3)$$

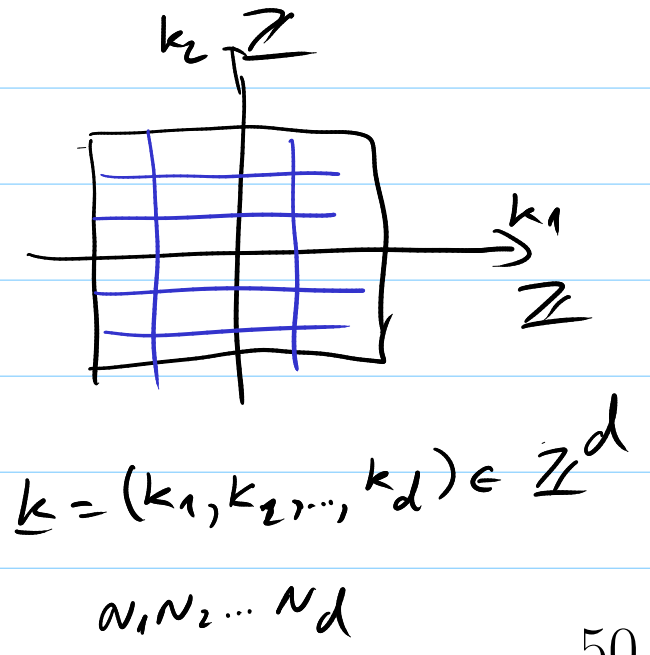
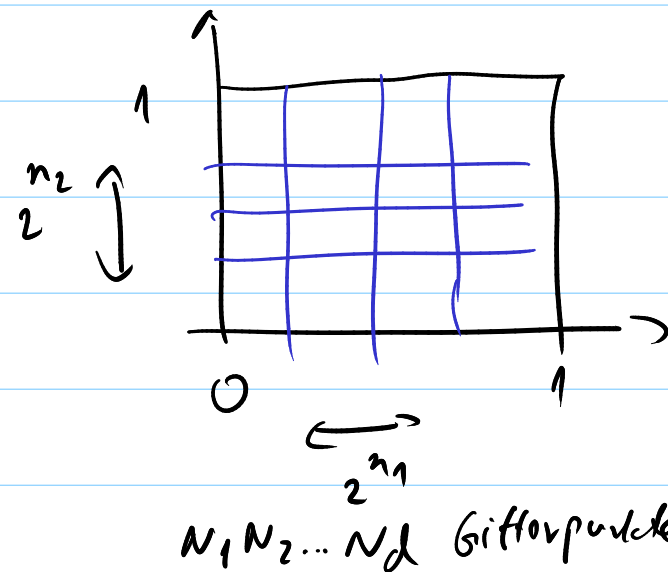
$$= \underline{c}(\underline{x}) \cdot \underline{\text{grad}}_{\underline{x}} u(t, \underline{x})$$

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial t} \left(\frac{\partial u}{\partial t} \right) = \frac{\partial}{\partial t} \left(-\underline{c}(\underline{x}) \cdot \underline{\text{grad}}_{\underline{x}} u(t, \underline{x}) \right) =$$

$$= -\underline{c}(\underline{x}) \cdot \frac{\partial}{\partial t} \underline{\text{grad}}_{\underline{x}} u(t, \underline{x}) = -\underline{c}(\underline{x}) \cdot \underline{\text{grad}}_{\underline{x}} \frac{\partial u}{\partial t}(t, \underline{x})$$

$$u(t, \underline{x}) = \sum_{\underline{k} \in \mathbb{Z}^d} c_{\underline{k}}(t) e^{-2\pi i \underline{k} \cdot \underline{x}}$$

$$\approx \sum_{\substack{\underline{k} \in \mathcal{K} \\ k_2 \in \mathbb{Z}}} c_{\underline{k}}(t) e^{-2\pi i \underline{k} \cdot \underline{x}}$$



$$\frac{\partial u}{\partial x_j} = \sum_{\underline{k} \in \mathcal{K}} c_{\underline{k}}(t) (-\pi i k_j) e^{-\pi i \underline{k} \cdot \underline{x}}, \quad j=1,2,\dots,d$$

$$\frac{\partial^2 u}{\partial y_j^2} = \sum_{\underline{k} \in \mathcal{K}} c_{\underline{k}}(t) (-\pi i k_j)^2 e^{-\pi i \underline{k} \cdot \underline{x}}, \quad j=1,2,\dots,d$$

$$\Delta u = \sum_{\underline{k} \in \mathcal{K}} c_{\underline{k}}(t) (-\pi i)^2 \|\underline{k}\|_2^2 e^{-\pi i \underline{k} \cdot \underline{x}} \in \mathbb{R}$$

$$\|\underline{k}\|_2^2 = k_1^2 + k_2^2 + \dots + k_d^2$$

d=2 Implementierung.

$$\underline{x}_1 = \left[\frac{0}{N_1}, \frac{1}{N_1}, \dots, \frac{N_1-1}{N_1} \right], \quad \underline{x}_2 = \left[\frac{0}{N_2}, \frac{1}{N_2}, \dots, \frac{N_2-1}{N_2} \right]$$

$$X, Y = \text{meshgrid}(x_1, x_2, \text{indexing} = 'ij')$$



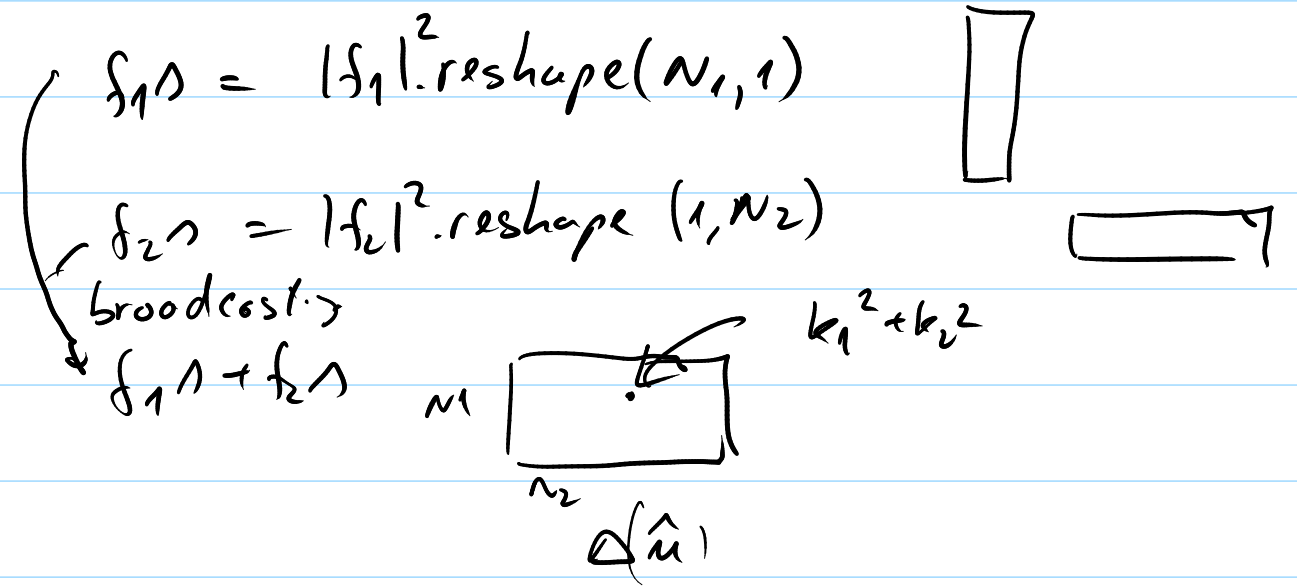
x, y $f(x, y)$ `imshow(origin = "lower")`

$$f_1 = f_1 \cdot \underline{x} \rightarrow \quad , \quad f_2 = f_2 \cdot \underline{x} \rightarrow$$

$$f[X, Y] = \text{meshgrid}(f_1, f_2, \text{indexing} = 'ij')$$

$$\frac{\partial \hat{u}}{\partial x} \text{ broadcast } f[X]$$

$$\frac{\partial \hat{u}}{\partial y} \quad f[Y]$$



§4 Runge-Kutta-Verfahren

§4.1 Grundidee

$$\begin{cases} \dot{y} = f(t, y) \\ y(t_0) = y_0 \end{cases} \Rightarrow \int_{t_0}^t f(t, y(t)) dt \Rightarrow y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(t, y(t)) dt$$

$h = t_1 - t_0$, Referenzintervall $[0, 1]$
 QF mit Knoten $c_i \in [0, 1]$ und Gewichte b_i

$$y(t_1) \approx y(t_0) + h \sum_{i=1}^n b_i f(t_0 + hc_i, y(t_0 + hc_i))$$

k_i \hookrightarrow Inkremente

noch unbekannt

$$y(t_1) \approx y(t_0) + h \sum_{i=1}^n b_i k_i$$

h erlaubt uns ein lokales Fehler $O(h^{p+1})$ für $y(t_1)$ zu bekommen, auch wenn wir für $y(t_0 + hc_i)$ nur eine Approx. der Ordnung $O(h^p)$ verwenden.

Bsp 1) QF = Trapezregel. $c_1=0, c_2=1$
 $b_1 = \frac{1}{2}, b_2 = \frac{1}{2}$

$$y(t_1) \approx y_0 + h \left(\frac{1}{2} f(t_0, y_0) + \frac{1}{2} f(t_0+h, y(t_0+h)) \right)$$

(iTR)

Idee: verwende etwas billiges für $y(t_0+h)$
 $y_0(t_0+h) \approx y_1 + h f(t_0, y_0)$ (eE)

$$\begin{cases} k_1 = f(t_0, y_0) \\ k_2 = f(t_0+h, y_0 + h k_1) \\ y_1 = y_0 + h \left(\frac{1}{2} k_1 + \frac{1}{2} k_2 \right) \end{cases}$$

explizite Trapezregel

Bsp 2) QF = MPR $c_1 = \frac{1}{2}, b_1 = 1$

$y(t_1) \approx y_0 + h f(t_0 + h \frac{1}{2}, y(t_0 + h \frac{1}{2}))$

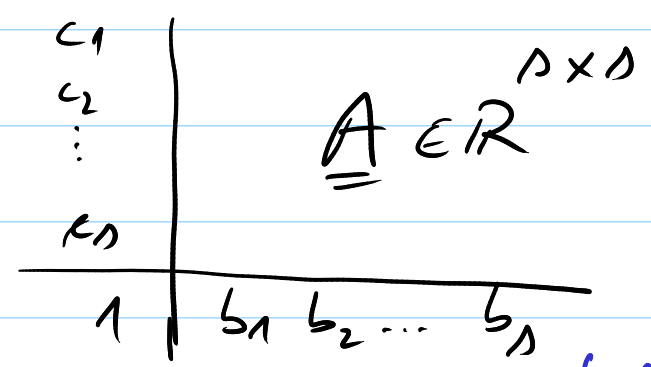
$y(t_0 + h \frac{1}{2}) \approx y_0 + h \frac{1}{2} f(t_0, y_0) \quad (e \in \epsilon)$

$$\begin{cases} k_1 = f(t_0, y_0) \\ k_2 = f(t_0 + \frac{h}{2}, y_0 + \frac{h}{2} k_1) \\ y_1 = y_0 + h k_2 \end{cases}$$
 explizite MPR.

hätte man hier (i ∈) statt (e ∈) → implizite MPR.

Def Runge-kutta-Verfahren mit Δ Stufen

Gegeben Butcher-Tableau (Schema)



So dass $b_1 + b_2 + \dots + b_n = 1$ notwendig für die Konvergenzordnung.
 $\sum_{j=1}^n a_{ij} = c_i$ für $i=1,2,\dots,n$ nicht notwendig aber meistens erfüllt (z.B. Kollimation) und wurde von Butcher in der Konstruktion verwendet

$$k_i = f(t_0 + h c_i, y_0 + h \sum_{j=1}^n a_{ij} k_j)$$
 Stufen
für $i=1,2,\dots,n$

$$y_1 = y_0 + h \sum_{i=1}^n b_i k_i$$

 $k_i = \text{Inkrement}$

ein $(n \cdot d) \times (n \cdot d)$ nicht-linearen algebraischer Gleichungssystem

$$\underline{\underline{A}} = \begin{bmatrix} 0 & & & \\ & 0 & & \\ & & \ddots & \\ * & & & 0 \end{bmatrix} \quad a_{ij} = 0 \text{ für } i \leq j$$

\Rightarrow RK explizit

$$\underline{k}_i = \underline{f}(t_0 + hc_i, \underline{y}_0 + h \sum_{j=1}^{i-1} a_{ij} \underline{k}_j)$$

$$\underline{\underline{A}} = \begin{bmatrix} & & & 0 \\ & & & \\ * & & & \\ & & & \end{bmatrix} \Rightarrow \text{diagonal implizite RK}$$

(DIRK)

$$\begin{cases} \underline{k}_i = \underline{f}(t_0 + hc_i, \underline{y}_0 + h \sum_{j=1}^{i-1} a_{ij} \underline{k}_j + h a_{ii} \underline{k}_i) \end{cases}$$

$d \times d$ nicht-lineares algebraisches Gleichungssystem.

Def Konsistenzordnung q wenn
lokale Fehler $\|\underline{y}_0(t_0+h) - \underline{y}_1\| \leq C \cdot h^{q+1}$

Theorem RK hat Konsistenzordnung $q \Rightarrow$

\Rightarrow QF hat Ordnung q

(ist exakt für Polynom vom Grad max $q-1$)

Beweis

$$\text{Nehme } \begin{cases} y = t^2 \\ y(0) = 0 \end{cases} \Rightarrow \underline{y}(t) = \frac{1}{2} t^2$$

Fehler:

$$|y(h) - \underline{y}_1| = \left| \frac{1}{2} h^2 - h \sum_{j=1}^n b_j (hc_j)^2 \right| \leq c \cdot h^{q+1} \quad | = h^{2+1}$$

$$\Rightarrow 0 \leq \left| \frac{1}{2} - \sum_{j=1}^n b_j c_j^2 \right| \leq c \cdot h^{q-2} \Rightarrow$$

Für $n=0, 1, 2, \dots, q-1 \Rightarrow q-2 > 0 \xrightarrow{(h \rightarrow 0)} 0$

\Rightarrow QF ist exakt genau für $n=0, 1, 2, \dots, q-1$

\Rightarrow QF hat Ordnung q .

Konsequenz RK mit s Stufen hat
maximale Konsistenzordnung $2s$

$$1) \quad \sum_{j=1}^s b_j = 1 \Leftrightarrow \text{mindestens Konsistenzordnung } q=1$$

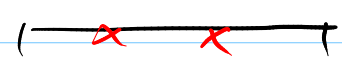
$$2) \quad \text{Konsistenzordnung } 2 \Rightarrow \sum_{j=1}^s b_j c_j = \frac{1}{2}$$

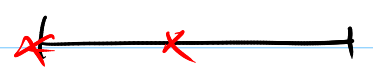
$$3) \quad q=3 \Rightarrow \sum_{j=1}^s b_j c_j^2 = \frac{1}{3}$$

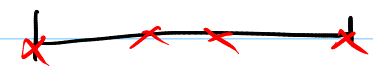
$$\oplus \quad \sum_{j=1}^s b_j \sum_{n=1}^s a_{jn} c_n = \frac{1}{6}$$

usw es wird sehr kompliziert ☹️

Theorem RK explizit $\Rightarrow q \leq s$

Gauss-Quadratur $\Rightarrow q=2s$ 

Radau-Quadratur \Rightarrow Radau-Verfahren
für ODE $q=2s-1$ 

Lobatto-Quadratur \Rightarrow
Lobatto-Verfahren für ODE $q=2s-2$ 

Theorem RK hat Konsistenzordnung $q \Rightarrow$

RK hat globale Konvergenzordnung q

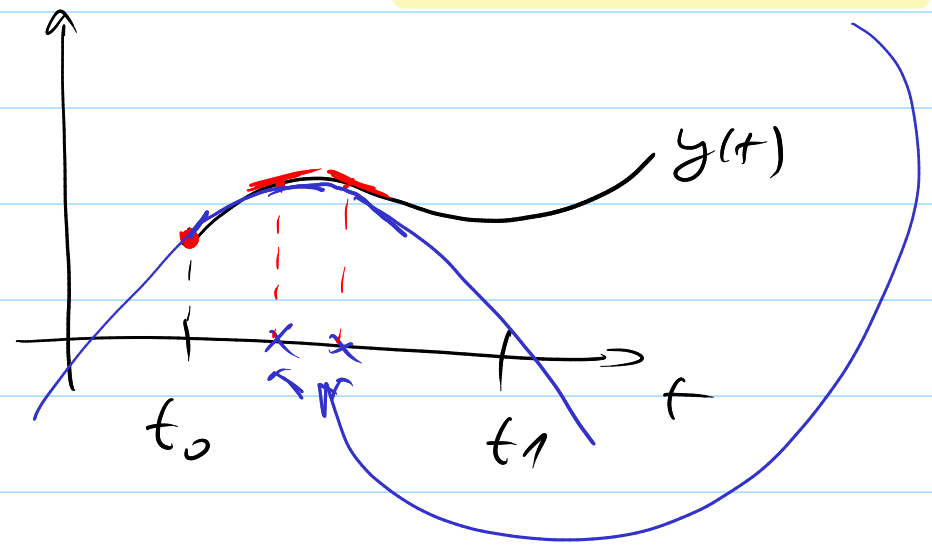
$$\|y(t_i) - \underline{y}_i\| \leq c \cdot h^q \quad \text{für alle } i=1,2,\dots,n, \quad T=nh$$

§4.2. Kollokation

Def $c_1, c_2, \dots, c_n \in [0, 1]$ verschieden
Kollokationspolynom $u(t)$ vom Grad n :

$$\begin{cases} u(t_0) = y_0 \\ \dot{u}(t_0 + hc_i) = f(t_0 + hc_i, u(t_0 + hc_i)) \end{cases} \quad \text{für } i=1, 2, \dots, n$$

Kollokationspunkte



Bsp 1) $n=1$ Polynom vom Grad 1
 $u(t) = y_0 + (t-t_0)k$

mit k so bestimmt, dass

$$\dot{u}(t_0 + hc_1) = f(t_0 + hc_1, u(t_0 + hc_1))$$

Kollokationspunkt $t_0 + hc_1$

$c_1=0 \Rightarrow (E)$

$c_1=1 \Rightarrow (iE)$

$c_1 = \frac{1}{2} \Rightarrow (iMP)$

2) $n=2$; $c_1=0, c_2=1 \Rightarrow$ implizite Trapezregel
 $c_{1,2} = \frac{1}{2} \pm \frac{\sqrt{3}}{6} \Rightarrow$ Gauss-Verfahren. $O(h^4)$

Theorem Die Kollokation mit Knoten c_1, \dots, c_n



n -Stufiges RK-Verfahren mit $a_{ij} = \int_0^{c_i} l_j(\tau) d\tau$
 $b_i = \int_0^{c_i} l_i(\tau) d\tau$

Wobei $l_i(z)$ = Lagrange Polynome zu c_1, \dots, c_s

$$l_i(z) = \prod_{\substack{j=1 \\ j \neq i}}^s \frac{z - c_j}{c_i - c_j} ; l_i(c_j) = \begin{cases} 1, & i=j \\ 0, & i \neq j \end{cases}$$

Beweis Notiere $k_i := u(t_0 + c_i h)$

$$u(t_0 + hz) = \sum_{j=1}^s k_j l_j(z) \quad \int_0^{c_i} \Rightarrow$$

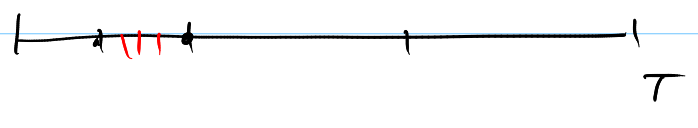
↳ hat Grad $s-1$

$$\Rightarrow u(t_0 + h c_i) = y_0 + h \sum_{j=1}^s k_j \int_0^{c_i} l_j(z) dz$$

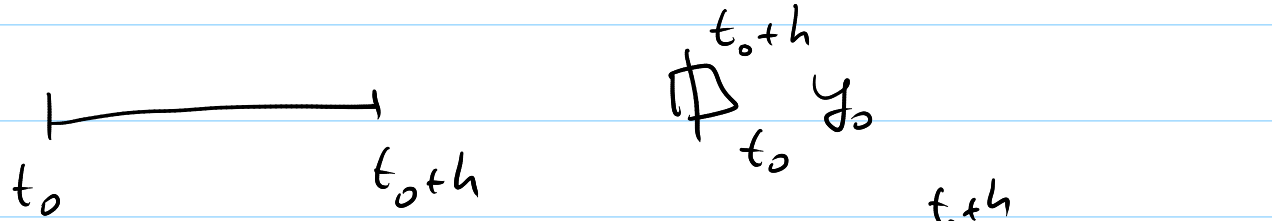
$$\int_0^1 \Rightarrow u(t_0 + h) = y_0 + h \sum_{j=1}^s k_j \int_0^1 l_j(z) dz$$

Konsequenz Kollokationsmethode hat dieselbe Ordnung wie die entsprechende φF .

§ 4.3. Adaptivität



Wunsch: vergrößere h , wenn die Lösung gut genug ist
verkleinere h , wenn die Lösung nicht gut genug ist.



Ordnung p : lokale Fehler $\| \Phi_{t_0}^{t_0+h} y_0 - \Psi_{t_0}^{t_0+h} y_0 \| \leq c \cdot h^{p+1}$

verwende $\Psi_{t_0}^{t_0+h} y_0$ bessere Approximation.

$$EST_k = \| \Phi_{t_k}^{t_k+h} y(t_k) - \Psi_{t_k}^{t_k+h} \underline{y}(t_k) \| \approx c h^{p+1} \Big|_{TOL}$$

→ berechne $h \Rightarrow$ Skript.

Professionell: 2 Rk. explizite $O(h^4)$ & $O(h^5)$
"ode45"; Dormand-Prince 5(4)

§4. Partitionierte RK-Verfahren

System ODE partitioniert:

$$\begin{cases} \dot{y} = \underline{f}(y, z) \\ \dot{z} = \underline{g}(y, z) \end{cases}$$

Idee: Verwende 2 verschiedene RK-Verfahren für y, z :

für y : $\begin{array}{c|c} c & \underline{A} \\ \hline 1 & \underline{b} \end{array}$ für z : $\begin{array}{c|c} \hat{c} & \hat{\underline{A}} \\ \hline 1 & \hat{\underline{b}} \end{array}$

$$\begin{cases} \underline{k}_j = \underline{f}(y_0 + h \sum_{i=1}^n a_{ij} \underline{k}_i, z_0 + h \sum_{i=1}^n \hat{a}_{ij} \underline{l}_i) \\ \underline{l}_j = \underline{g}(y_0 + h \sum_{i=1}^n a_{ij} \underline{k}_i, z_0 + h \sum_{i=1}^n \hat{a}_{ij} \underline{l}_i) \end{cases}$$

$$\underline{y}_1 = \underline{y}_0 + h \sum_{j=1}^n b_j \underline{k}_j$$

$$\underline{z}_1 = \underline{z}_0 + h \sum_{j=1}^n \hat{b}_j \underline{l}_j$$

Bsp 1) $(eE) \quad b_1=1, a_{11}=1$ } \Rightarrow anwenden an die
 $(iE) \quad \hat{b}_1=1, \hat{a}_{11}=0$ } Newton-Gleichg.
 $y = \text{Position}$
 $z = \text{Geschwindigkeit}$

\Rightarrow symplektische Euler-Verfahren.

Bsp 2) $\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline 1 & \frac{1}{2} & \frac{1}{2} \end{array} \quad \begin{array}{c|cc} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline 1 & \frac{1}{2} & \frac{1}{2} \end{array}$ für Newton-Gleichg.

\Rightarrow Störmer-Verlet Verfahren.

Bsp 3) 3-stufige Lobatto-Paar:

0	0	0	0	0	$\frac{1}{6}$	$-\frac{1}{6}$	0
$\frac{1}{2}$	$\frac{5}{24}$	$\frac{1}{3}$	$-\frac{1}{24}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{3}$	0
1	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$	1	$\frac{1}{6}$	$\frac{5}{6}$	0
1	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$	1	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$

=> $O(h^4)$ Verallgemeinerung von Störmer-Verlet.

Bem Newton' Gleichung:

$\begin{cases} \dot{y} = z \\ \dot{z} = g(t, y, z) \end{cases}$
PRK \implies
RK-Nystrom-Verfahren (RKV)

$$\begin{cases}
 \underline{l}_i = g(t_0 + c_i h, y_0 + c_i h z_0 + h^2 \sum_{j=1}^p \bar{a}_{ij} l_j, z_0 + h \sum_{j=1}^p \hat{a}_{ij} l_j) \\
 \underline{y}_1 = y_0 + h \left(z_0 + h \sum_{i=1}^p \bar{b}_i l_i \right) \\
 \underline{z}_1 = z_0 + h \sum_{i=1}^p \hat{b}_i l_i
 \end{cases}$$

aus der $k_i = \dots$

mit $\bar{b}_i = \sum_{k=1}^p b_k \hat{a}_{ki}$

$\bar{a}_{ij} = \sum_{k=1}^p a_{ik} \hat{a}_{kj}$

Wenn g nicht explizit y abhängig ist
 => braucht man \hat{a}_{kj} nicht.

Bez PRK = Splitting art

$$\underline{u} = \begin{bmatrix} \underline{q} \\ \underline{z} \end{bmatrix}, \quad \underline{f}_a = \begin{bmatrix} \underline{f}(\underline{u}) \\ 0 \end{bmatrix}, \quad \underline{f}_b = \begin{bmatrix} 0 \\ \underline{g}(\underline{u}) \end{bmatrix}$$

Splitting: $\underline{\dot{u}} = \underline{f}_a(\underline{u})$ mit RK-Verfahren
 u.d. $\underline{\dot{u}} = \underline{f}_b(\underline{u})$ mit RK-Verfahren.

\Rightarrow PRK

Bsp BM 42 mit $O(4^4)$] symplektischer
 BM 63 mit $O(6^6)$] Verfahren.

§ Nichtlineare Algebraische Gleichungen

§ 1. Konvergenz, Fixpunktiterationen

Gegeben: $F: \mathbb{R}^d \rightarrow \mathbb{R}^d$

Gesucht: $\underline{x}^* \in \mathbb{R}^d$ so dass $F(\underline{x}^*) = \underline{0}$

Wie: Iteration $\underline{x}^{(k+1)} = \Phi(\underline{x}^{(k)})$ so dass $\underline{x}^{(k)} \rightarrow \underline{x}^*$

Konvergenzbegriff: die Iteration $\underline{x}^{(k+1)} = \Phi(\underline{x}^{(k)})$ heißt

+ linear konvergent nach \underline{x}^* , falls es gibt $L < 1$ so dass

$$\|\underline{x}^{(k+1)} - \underline{x}^*\| \leq L \|\underline{x}^{(k)} - \underline{x}^*\| \text{ für alle } k \in \mathbb{N}$$

+ konvergent mit Ordnung $p > 1$, falls es gibt C so dass

$$\|\underline{x}^{(k+1)} - \underline{x}^*\| \leq C \|\underline{x}^{(k)} - \underline{x}^*\|^p \text{ für alle } k \in \mathbb{N}$$

[für $p=1$ muss $C < 1$ damit lineare Konvergenz]

Ben lin. konvergente It \Leftrightarrow Graph von $\log \|\underline{x}^k - \underline{x}^*\| =$ Gerade mit Steigung $\log L$

Bsp Bisektionsverfahren - nur für $d=1$; lineare Konvergenz

F stetig; $F(a)F(b) < 0$

$c = \frac{a+b}{2}$ und Rekursion auf $[a, c]$ falls $F(c)F(a) < 0$
 $[c, b]$ falls $F(c)F(b) < 0$

Analysis: Banach' Fixpunktsatz \Rightarrow wichtiges Tool: Fixpunktiteration

Idee: Schreibe Nullstellenproblem $F(\underline{x}^*) = 0$ in

Fixpunktiteration $\Phi(\underline{x}^*) = \underline{x}^*$ um.

Verwende dann Φ als Iterationsvorschrift.

viele Möglichkeiten: wähle die Φ , die (schnelle) Konvergenz ergibt!

Bsp $F: \mathbb{R} \rightarrow \mathbb{R}, F(x) = xe^x - 1$

1) $xe^x - 1 = 0 \Leftrightarrow xe^x = 1 \Leftrightarrow x = e^{-x}$, nehme $\Phi_1(x) = e^{-x}$

2) $xe^x - 1 = 0 \Leftrightarrow xe^x - 1 + x = x \Leftrightarrow x(e^x + 1) = x + 1 \Leftrightarrow x = \frac{x+1}{e^x + 1}$, $\Phi_2(x) = \frac{x+1}{e^x + 1}$

3) $xe^x - 1 = 0 \Leftrightarrow xe^x - 1 - x = -x \Leftrightarrow x = x + 1 - xe^{-x}$, $\Phi_3(x) = x + 1 - xe^{-x}$

Beobachte:

k	$x^{(k+1)} := \phi_1(x^{(k)})$	$x^{(k+1)} := \phi_2(x^{(k)})$	$x^{(k+1)} := \phi_3(x^{(k)})$	k	$ x_1^{(k+1)} - x^* $	$ x_2^{(k+1)} - x^* $	$ x_3^{(k+1)} - x^* $
0	0.5000000000000000	0.5000000000000000	0.5000000000000000	0	0.067143290409784	0.067143290409784	0.067143290409784
1	0.606530659712633	0.566311003197218	0.675639364649936	1	0.039387369302849	0.000832287212566	0.108496074240152
2	0.545239211892605	0.567143165034862	0.347812678511202	2	0.021904078517179	0.00000125374922	0.219330611898582
3	0.579703094878068	0.567143290409781	0.855321409174107	3	0.012559804468284	0.000000000000003	0.288178118764323
4	0.560064627938902	0.567143290409784	-0.156505955383169	4	0.007078662470882	0.000000000000000	0.723649245792953
5	0.571172148977215	0.567143290409784	0.977326422747719	5	0.004028858567431	0.000000000000000	0.410183132337935
6	0.564862946980323	0.567143290409784	-0.619764251895580	6	0.002280343429460	0.000000000000000	1.186907542305364
7	0.568438047570066	0.567143290409784	0.713713087416146	7	0.001294757160282	0.000000000000000	0.146569797006362
8	0.566409452746921	0.567143290409784	0.256626649129847	8	0.000733837662863	0.000000000000000	0.310516641279937
9	0.567559634262242	0.567143290409784	0.924920676910549	9	0.000416343852458	0.000000000000000	0.357777386500765
10	0.566907212935471	0.567143290409784	-0.407422405542253	10	0.000236077474313	0.000000000000000	0.974565695952037

lineare quadratische keine lineare quadratische keine Konvergenz

\hookrightarrow Anzahl exakter Stellen nach der Komma

verdoppelt sich in jeder Iteration.

Worum ist es so?

Banach' Fixpunktsatz: Φ Kontraktion \Rightarrow Konvergente Iteration.

$\Phi: B \rightarrow B$, B kompakt im normierten Raum

$\Phi =$ Kontraktion: $\|\Phi(x) - \Phi(y)\| \leq L \|x - y\|$ für alle $x, y \in B$ mit $L < 1$

Ben Falls Φ stetig diff'bar im konvexen $U \subset \mathbb{R}^d$,

reicht $L = \sup_{x \in U} \|\mathcal{D}\Phi(x)\| < 1$ für lokale lin. Konvergenz

Theorem Sei $\Phi(x^*) = x^*$ mit Φ $(m+1)$ -mal stetig differenzierbar.

Falls $\Phi^{(l)}(x^*) = 0$ für $l = 1, 2, \dots, m \geq 1$,

dann konvergiert $x^{k+1} = \Phi(x^k)$ gegen x^* lokal mit Ordnung $p \geq m+1$

Beweis

Taylor um x : $\Phi(y) = \Phi(x) + \sum_{k=1}^m \frac{1}{k!} \Phi^{(k)}(x)(y-x)^k + o(|y-x|^{m+1})$

Nehme $x = x^*$, $y = x^{(k)}$:

$x^{(k+1)} - x^* = \Phi(x^{(k)}) - \Phi(x^*) = \sum_{k=1}^m \frac{1}{k!} \underbrace{\Phi^{(k)}(x^*)}_{=0} (x^{(k)} - x^*)^k + o(\|x^{(k)} - x^*\|^{m+1})$

$\Rightarrow \|x^{(k+1)} - x^*\| \leq C \cdot \|x^{(k)} - x^*\|^{m+1}$

Bsp $\Phi_2(x) = \frac{x+1}{e^x+1} \Rightarrow \Phi_2'(x) = \frac{1-xe^x}{(e^x+1)^2} = \frac{F(x)}{(e^x+1)^2} \Rightarrow \Phi_2'(x^*) = 0$ da $F(x^*) = 0$

$\Rightarrow \Phi_2$ mindestens Ordnung 2.

Wann sollen wir eine Iteration abbrechen?

1) nach k Schritte \square (endet sicher)

2) falls $\|x^{(k)} - x^*\| \leq \text{TOL}$ \square

Ersatz: $\|x^{(k)} - x^{(k+1)}\| \leq \text{TOL}$

Alternative: falls kleines Residuum: $\|F(x^{(k)})\|_2 < \text{TOL}$

$\rightarrow \infty$ Iteration möglich \rightarrow verwende 1)+2) zu früh stoppen möglich!

3) Falls lin. Konvergenz mit bekannter $L \Rightarrow$ nutze

$\|x^{(k)} - x^{(k+1)}\| \leq \frac{1-L}{L} \text{TOL}$

L unbekannt \Rightarrow Schätze $L \approx \frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k)} - x^{(k-1)}\|} =: \frac{\epsilon_{k+1}}{\epsilon_k}$ mit N gross

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

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

$x^0 = 0.4:$

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

§5.2 Newton-Verfahren

Suche $\underline{x}^* \in \mathbb{R}^d$ so dass $\underline{F}(\underline{x}^*) = \underline{0} \in \mathbb{R}^d$

Idee: linearisiere \underline{F} und löse das lineare Problem \Rightarrow Iteration

$$\underline{x}^{(0)} \text{ gegeben } \Rightarrow \underline{F}(\underline{x}) = \underbrace{\underline{F}(\underline{x}^{(0)}) + \underline{DF}(\underline{x}^{(0)}) (\underline{x} - \underline{x}^{(0)})}_{\tilde{\underline{F}}(\underline{x}) \text{ linear}} + O(\|\underline{x} - \underline{x}^{(0)}\|^2)$$

Statt $\underline{F}(\underline{x}) = \underline{0}$ löse $\tilde{\underline{F}}(\underline{x}) = \underline{0} \Leftrightarrow$

$$\underline{DF}(\underline{x}^{(0)}) (\underline{x}^{(0)} - \underline{x}) = \underline{F}(\underline{x}^{(0)}) \Leftrightarrow \text{löse } \underline{DF}(\underline{x}^{(0)}) \underline{\Delta} = \underline{F}(\underline{x}^{(0)})$$

dann $\underline{x}^{(1)} = \underline{x}^{(0)} - \underline{\Delta}$

= Newton Korrektur

Methode: $\underline{x}^{(k+1)} = \underline{x}^{(k)} - \underline{DF}(\underline{x}^{(k)})^{-1} \underline{F}(\underline{x}^{(k)})$ niemals so implementiert

sondern immer:

$$\left[\begin{array}{l} \text{löse } \underline{DF}(\underline{x}^{(k)}) \underline{\Delta} = \underline{F}(\underline{x}^{(k)}) \\ \text{update } \underline{x}^{(k+1)} = \underline{x}^{(k)} - \underline{\Delta} \end{array} \right]$$

numerische lineare algebra

Newton-Verfahren

Theorem Newton-Verfahren konvergiert lokal mit Ordnung $p=2$

$\Phi(x) = x - \underline{DF}(x)^{-1} \underline{F}(x)$ ist Fixpunktiteration

Rechnen: $\underline{D}\Phi(x) = \underline{I} - \underline{D}[\underline{DF}(x)^{-1}] \underline{F}(x) - \underbrace{\underline{DF}(x)^{-1} \cdot \underline{DF}(x)}_{\underline{I}} \Rightarrow$

$$\underline{D}\Phi(x^*) = -\underline{D}[\underline{DF}(x^*)^{-1}] \underline{F}(x^*) = \underline{0} \Rightarrow \text{Ordnung } p=2.$$

Bedingung: $\underline{DF}(x^*)$ ist invertierbar! \parallel_0

(für $d=1$: $f'(x^*) \neq 0!$)

\rightarrow Bsp 5.7.3. Seite 238

Achtung: \underline{DF} am besten analytisch berechnen!

z.B. sympy + lambdify.

ML: nur einfache Funktionen \Rightarrow automatische Differentiation (backpropagation)

Vermeide Finite Differenzen um \underline{DF} zu approximieren (mögliche Rundungsfehler)

optimize.fsolve, optimize.root

$d=1$: viele Methoden $d>1$: broyden / LM / hybr

d gross \Rightarrow $\underline{D}F(x)$ kann sehr teuer sein

Idee: verwende dieselbe Matrix $\underline{J} = \underline{D}F(x)$ in mehrere Iterationen

\Rightarrow vereinfachtes Newton:

$\underline{J} = \underline{D}F(x^{(0)})$ Auswertung

$\underline{J} = \underline{L} \underline{U}$ $O(d^3)$

für $k=1, 2, \dots$:

löse $\underline{L} \underline{U} \underline{\Delta} = \underline{F}(x^{(k)})$ $O(d^2)$

$x^{(k+1)} = x^{(k)} - \underline{\Delta}$

Nachteil: nur lineare Konvergenz!

Quasi-Newton und Broyden

$\underline{D}F(x)$ steht nicht zur Verfügung!

Idee $d=1$: Statt Tangente, verwende eine Sekante

\Rightarrow brauche 2 Startwerte: $x^{(0)}, x^{(1)}$ gegeben

$F'(x^{(k)}) \approx \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}$

Sekantenverfahren

Update $\underline{\Delta}^{(k)} = \frac{x^{(k)} - x^{(k-1)}}{F(x^{(k)}) - F(x^{(k-1)})} F(x^{(k)})$
 $x^{(k+1)} = x^{(k)} - \underline{\Delta}^{(k)}$ Ordnung < 2

$d > 1$: statt $\underline{D}F(x^{(k)})$ verwende eine Matrix \underline{J}_k sodass

$\underline{J}_k (x^{(k)} - x^{(k-1)}) = \underline{F}(x^{(k)}) - \underline{F}(x^{(k-1)})$

Ben Viele \underline{J}_k sind möglich!

Idee von Broyden: Baue \underline{J}_k als Rang-1-Änderung von \underline{J}_{k-1} :

$\underline{J}_k = \underline{J}_{k-1} + \frac{1}{\|x^{(k)} - x^{(k-1)}\|_2^2} \underline{F}(x^{(k)}) (x^{(k)} - x^{(k-1)})^T$

Vorteil #1: F nur ein einziges Mal auswerten!

Broyden-Verfahren: naive Implementierung:

$x^{(0)}$ gegeben, $\underline{J}_0 = \underline{D}F(x^{(0)})$

für $k=1, 2, \dots$

löse $\underline{J}_k \underline{\Delta} = \underline{F}(x^{(k)})$

$x^{(k+1)} = x^{(k)} - \underline{\Delta}$

$\underline{J}_{k+1} = \underline{J}_k + \frac{1}{\|\underline{\Delta}\|_2^2} \underline{F}(x^{(k+1)}) (-\underline{\Delta})^T$

Man kann beweisen: $\lim_{k \rightarrow \infty} \frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k)} - x^*\|} = 0$ Superlineare Konvergenz

Bessere Implementierung via Sherman-Morrisson-Formel.

$\underline{\partial}_{k+1} = \underline{\partial}_k + \text{Rang-1-Matrix.}$

$$\underline{(A + uv^T)^{-1}} = \underline{A}^{-1} - \frac{1}{1 + v^T \underline{A}^{-1} u} \underline{A}^{-1} \underline{uv^T} \underline{A}^{-1}$$

$$\underline{\partial}_{k+1}^{-1} = \underline{\partial}_k^{-1} + \frac{\underline{\partial}_k^{-1} F(x^{k+1}) \underline{\partial}_k^T \underline{\partial}_k^{-1}}{\|\underline{\partial}_k\|^2 - \underline{\partial}_k^T \underline{\partial}_k^{-1} F(x^{k+1})}$$

Schreibe die Iteration um

gegeben: $\underline{x}^0, \underline{\partial}_0 = \underline{DF}(x^0)$

zerlege $\underline{\partial}_0 = \underline{L} \underline{U}$

löse $\underline{L} \underline{U} \underline{\rho}^0 = \underline{F}(x^0)$

$$\underline{x}^1 = \underline{x}^0 - \underline{\rho}^0$$

$$\underline{\partial}_1^{-1} = \underline{\partial}_0^{-1} + \frac{\underline{\partial}_0^{-1} F(x^1) \underline{\partial}_0^T \underline{\partial}_0^{-1}}{\|\underline{\partial}_0\|^2 - \underline{\partial}_0^T \underline{\partial}_0^{-1} F(x^1)}$$

$$\underline{z}^2 = \underline{x}^1 - \underline{\partial}_1^{-1} F(x^1)$$

$$\underline{\rho}^1 = \underline{\partial}_1^{-1} F(x^1) = \underline{\partial}_0^{-1} F(x^1) + \frac{\underline{w} \underline{\partial}_0^T \underline{\partial}_0^{-1} F(x^1)}{\|\underline{\partial}_0\|^2 - \underline{\partial}_0^T \underline{w}}$$

$\underline{w} = \underline{\partial}_0^{-1} F(x^1) \Rightarrow \underline{w}$ aus lösen $\underline{\partial}_0 \underline{w} = F(x^1)$
mit $(\underline{L}, \underline{U})$
 $z \in \mathbb{R}$

$$\underline{\rho}^1 = \underline{w} + \frac{\underline{w} z}{\|\underline{\partial}_0\|^2 - z} = \left(1 + \frac{z}{\|\underline{\partial}_0\|^2 - z} \right) \underline{w}$$

$$\Rightarrow \underline{x}^2 = \underline{x}^1 - \underline{\rho}^1$$

"BFGS"

→ Code 5.9.4. Bsp 5.9.5, 5.9.2.

Ben Konvergenz ist immer von einem guten Startwert abhängig.

Wie? Bild / Intuition / Randomization / Dämpfung der Schritte.

§.3 Anwendung zur Optimierung

$f: D \subset \mathbb{R}^d \rightarrow \mathbb{R}$, $f(x^*) = \min_{x \in D} f(x)$, f diff ^{bar}

kritische Stellen: $Df(x) = 0$ $F(x) := Df(x) = \text{grad } f(x)$

x^* lok. Min: $F(x^*) = \text{grad}(x^*) = 0$ & $D^2 F(x^*) > 0$ positiv def.

↓
Nullstellenproblem; Newton braucht $D^2 F = D^2 f$ Hesse matrix
↓ teuer!

verwende Broyden-Varianten: BFGS

C-DFGS-G

constraint (Nebenbedingungen)

optimize: minimize \Rightarrow viele Möglichkeiten: BFGS, CG, ...

ML: least squares: $\min_{\theta} \frac{1}{N} \sum_{i=1}^N \|f(x_i, \theta) - y_i\|^2 = \min_{\theta} h(\theta)$
↑ \hookrightarrow Messungen
↓ Funktionswerte zu x_i parameter (to learn)

N sehr gross

$\theta \in \mathbb{R}^d$, d sehr gross

\Rightarrow auch BFGS kann zu langsam sein.

Gradienten-Verfahren: Iterationen in Richtung steilsten Abstieg.

\hookrightarrow zu langsam für lange Täler
↓
grosses d
grosses N
don conjugate gradient ok.

\Rightarrow wähle zufällig einige n Richtungen und führe Schritte des Gradienten-Verfahrens nur dort, d.h. ersetze den Schritt $\theta^{k+1} = \theta^k - \eta \text{grad } h(\theta^k)$

learning rate = Schrittlänge in Abstieg

mit: wähle zufällig "batch" und verwende nur:

$\theta^{k+1} = \theta^k - \frac{1}{2|\text{batch}|} \sum_{i \in \text{batch}} \|f(x_i, \theta) - y_i\| \frac{\partial}{\partial \theta} f(x_i, \theta)$

\Rightarrow stochastik gradient pytorch
adam, etc

§6 Steife Differentialgleichungen

Somit: $y_N \rightarrow 0$ für $N \rightarrow \infty$ nur wenn $0 < h < \frac{2}{|\lambda|}$
klein genug!

§6.1. Einführung



Def ODE heißt steif, falls explizite Verfahren einen Zeitschritt h sehr klein brauchen, kleiner als die Genauigkeit verlangt!

Modellproblem

$\dot{y} = \lambda y$ mit $\lambda < 0$
 $y(t) = e^{\lambda t} y(0) \rightarrow 0$ für $t \rightarrow \infty$

Bsp

$$\frac{d}{dt} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \underbrace{\begin{bmatrix} -50 & 49 \\ 49 & -50 \end{bmatrix}}_{\underline{\underline{B}}} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Interesse:

asymptotisches Verhalten der numerischen Lösung **sollte** qualitativ (zumindest) ähnlich der exakten Lösung sein.

$$\underline{\underline{\dot{y}}} = \underline{\underline{B}} \underline{\underline{y}}$$

$\underline{\underline{B}}$ symmetrisch \Rightarrow $\underline{\underline{B}}$ diagonalisierbar, d.h.

es gibt $\underline{\underline{S}}$ (mit $\underline{\underline{S}} \underline{\underline{S}}^T = \underline{\underline{I}}$) so dass

(EF) $y_1 = y_0 + h f(y_0) = y_0 + h \lambda y_0 = (1+h\lambda) y_0$
 $y_2 = y_1 + h f(y_1) = (1+h\lambda) y_0 + h \lambda (1+h\lambda) y_0 = (1+h\lambda)^2 y_0$
...
 $y_N = (1+h\lambda)^N y_0 \rightarrow 0$ für $N \rightarrow \infty$.
nur wenn $|1+h\lambda| < 1$

$$\underline{\underline{B}} = \underline{\underline{S}} \underline{\underline{D}} \underline{\underline{S}}^T \quad \text{mit } \underline{\underline{D}} = \text{Diagonalmatrix}$$

$$\underline{\dot{y}} = \underline{S} \underline{D} \underline{S}^T \underline{y} \Rightarrow \underline{S}^T \underline{\dot{y}} = \underline{D} \underline{S}^T \underline{y} \Rightarrow$$

$$\underline{\dot{z}} = \underline{D} \underline{z} \Rightarrow \begin{cases} \dot{z}_1 = \lambda_1 z_1 \\ \dot{z}_2 = \lambda_2 z_2 \end{cases}$$

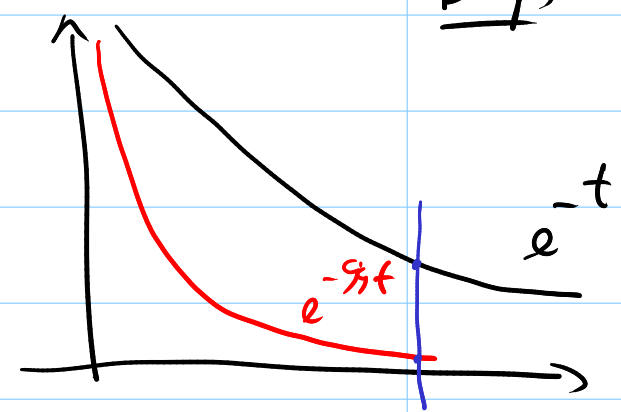
$$\underline{y} = \underline{S} \underline{z}$$

$$\underline{D} = \begin{bmatrix} -1 & 0 \\ 0 & -99 \end{bmatrix} \Rightarrow \begin{cases} \dot{z}_1 = -z_1 \Rightarrow h < 2 \\ \dot{z}_2 = -99 z_2 \Rightarrow h < \frac{2}{99} \end{cases}$$

\Rightarrow (e.E) verlangt ein Zeitschritt $h < \frac{2}{99}$

Die exakte Lösung:

$$\begin{cases} y_1(t) = e^{-t} + e^{-99t} \\ y_2(t) = e^{-t} - e^{-99t} \end{cases}$$



$$y_1(t) = e^{-t} + e^{-99t}$$

sehr klein, irrelevant!

aber der Zeitschritt h wird durch e^{-99} bestimmt!
 $h < \frac{2}{99}$

(iE) für das Modellproblem $\dot{y} = \lambda y$ mit $\lambda < 0$
 $y(t) = e^{\lambda t} y_0$
 $y_1 = y_0 + h f(y_1) = y_0 + h \lambda y_1 \Rightarrow y_1 = \frac{1}{1 - \lambda h} y_0$

$$y_N = \frac{1}{(1 - \lambda h)^N} y_0 \rightarrow 0 \text{ für } N \rightarrow \infty \text{ für jedes } \lambda, \text{ och.}$$

Ben Implizite Verfahren stellen keine Bedingung an h .

Bsp explizite Trapezregel.

$$\begin{cases} k_1 = \lambda y_0 \\ k_2 = \lambda (y_0 + h k_1) = \lambda (y_0 + h \lambda y_0) = \lambda y_0 + h \lambda^2 y_0 \end{cases}$$

$$y_1 = y_0 + \frac{1}{2} h k_1 + \frac{1}{2} h k_2 = y_0 + \frac{1}{2} h \lambda y_0 + \frac{1}{2} h \lambda y_0 + \frac{1}{2} h^2 \lambda^2 y_0$$

$$\Rightarrow y_1 = \underbrace{\left(1 + h\lambda + \frac{1}{2}(h\lambda)^2\right)}_{S(h\lambda)} y_0 = S(h\lambda) y_0$$

$$y_N = (S(h\lambda))^N y_0 \rightarrow 0 \text{ nur wenn } |S(h\lambda)| < 1$$

§6.2. Stabilität der RK-Verfahren

$$\begin{cases} \dot{y} = \lambda y \text{ mit } \lambda \in \mathbb{C}, \operatorname{Re} \lambda < 0 \\ y(t) = e^{\lambda t} y_0 \end{cases}$$

Numerisches Verfahren. $y_N = [S(h\lambda)]^N y_0$

Frage: Wann $|y_N| \rightarrow 0$ für $N \rightarrow \infty$.

$S(z)$ = Stabilitätsfunktion.

Bsp

(eE) $S(z) = 1 + z$

(iE) $S(z) = \frac{1}{1-z}$

(eTR) $S(z) = 1 + z + \frac{1}{2} z^2$

RK-Verfahren mit Δ Stufen:

$$y_1 = y_0 + h \sum_{j=1}^{\Delta} b_j \cdot k_j$$

$$\begin{cases} k_i = f(t_0 + h c_i, y_0 + h \sum_{j=1}^{\Delta} a_{ij} k_j) \\ i = 1, 2, \dots, \Delta \end{cases}$$

für $\dot{y} = \lambda y$ d.h. $f(t, y) = \lambda y \Rightarrow$

$$\begin{cases} k_i = \lambda y_0 + h \lambda \sum_{j=1}^{\Delta} a_{ij} k_j \\ i = 1, 2, \dots, \Delta \end{cases} \quad \underline{k} = \begin{bmatrix} k_1 \\ \vdots \\ k_{\Delta} \end{bmatrix} \quad (z = h\lambda)$$

$$\Rightarrow \underline{k} = \lambda y_0 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} + z \underline{A} \underline{k} \Leftrightarrow$$

$$\left(\underline{I} - z \underline{A} \right) \underline{k} = \lambda y_0 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Falls $\underline{I} - z \underline{A}$ singular \rightarrow

Falls $\underline{I} - z \underline{A}$ regulär $\Rightarrow \underline{k} = \lambda y_0 \left(\underline{I} - z \underline{A} \right)^{-1} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$

$$y_1 = y_0 + h \lambda y_0 \sum_{i=1}^n b_i \left((\underline{I} - z \underline{A})^{-1} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \right)_i$$

$$y_1 = y_0 \left(1 + z \underline{b}^T (\underline{I} - z \underline{A})^{-1} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \right)$$

$$S(z) = 1 + z \underline{b}^T (\underline{I} - z \underline{A})^{-1} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

↓ Stabilitätsfunktion des n-stufiges RkV.

$|y_n| \rightarrow 0$ für $n \rightarrow \infty$ nur wenn $|S(z)| < 1$

Theorem

Die Stabilitätsfunktion eines n-stufiges RkV ist eine (komplexwertige) rationale Funktion.

$$S(z) = \frac{P(z)}{Q(z)} \text{ mit } P, Q \text{ Polynome von Grad } \leq n$$

und $Q(z) = 0$ für $z = \frac{1}{\mu}$ mit $\mu \in \text{EK von } \underline{A}$

Falls RkV explizit ist, dann $Q(z) \equiv 1$.

Beweis explizites RkV $\underline{A} = \begin{bmatrix} 0 & & 0 \\ * & 0 & 0 \\ & & 0 \end{bmatrix} \Rightarrow$

$$\underline{A}^n = \underline{A} \cdot \underline{A} \cdot \dots \cdot \underline{A} = \begin{bmatrix} 0 & \dots & 0 \\ 0 & \dots & 0 \end{bmatrix}$$

$$(\underline{I} - z \underline{A})^{-1} = \underline{I} + z \underline{A} + (z \underline{A})^2 + \dots + (z \underline{A})^{n-1} + \dots$$

$$= \underline{I} + z \underline{A} + z^2 \underline{A}^2 + \dots + z^{n-1} \underline{A}^{n-1}$$

= Polynom von Grad n-1 in z.

$$\underline{v} = (\underline{I} - z \underline{A})^{-1} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \Rightarrow v_i(z) = \frac{P(z)}{\det(\underline{I} - z \underline{A})} = \frac{P(z)}{Q(z)}$$

Cramer'sche Regel

$$Q(z) = 0 \Leftrightarrow \det(\underline{I} - z \underline{A}) = 0 \Leftrightarrow z = \frac{1}{\mu} \text{ mit } \mu \in \text{EK von } \underline{A}$$

Konsequenz

1) Falls $\frac{1}{h\lambda}$ nicht EW von \underline{A} ist, dann

$$y_n = S(h\lambda)^n y_0 \text{ mit } n=0,1,2,\dots,$$

mit wohldefinierter Stabilitätsfunktion $S(z)$.

2) $y_0=1 \Rightarrow$ exakte Lösung $y(t) = e^{\lambda t}$
 num. Lösung $y_n = S(h\lambda)^n$

RK hat Konsistenzordng. q :

$$\text{Fehler } |e^{nh\lambda} - S(h\lambda)^n| \leq Ch^{q+1}$$

$\Rightarrow S(z)$ ist eine Approximation an e^z

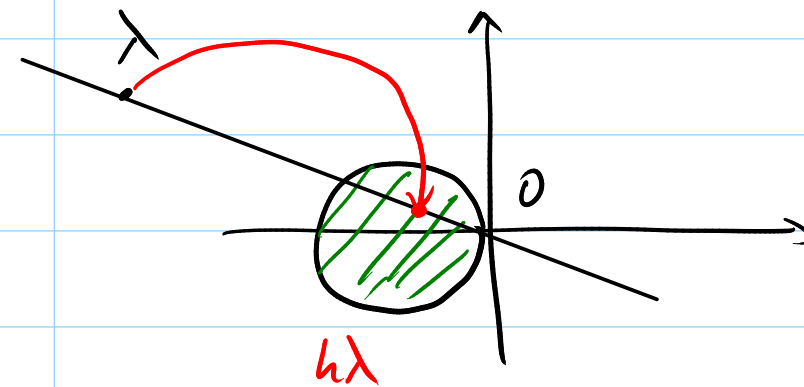
$$|e^z - S(z)| \leq c \cdot |z|^{q+1}$$

\Rightarrow Taylorpolynom von e^z , $S(z)$ um $z=0$
 sind identisch bis zum Grad q .

Def Stabilitätsgebiet des num. Verfahrens \mathcal{M}

$$S_{\mathcal{M}} = \{z \in \mathbb{C} \text{ sodass } |S(z)| < 1\}$$

$$y_n = S(z)^n y_0 \rightarrow 0 \text{ nur wenn } z \in S_{\mathcal{M}}$$



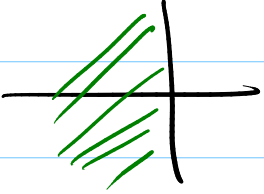
für gegebenes λ ,
 wie klein muss h
 gewählt werden?

$$h\lambda \in S_{\mathcal{M}}$$

Bem RK explizit $\Rightarrow S(z) = \text{Polynom von Grad } s-1$
 \Rightarrow Stabilitätsbereich ist beschränkt!
 \Rightarrow immer eine Schranke an h haben!

Bem ode 45 / dopri54 verwenden explizite RK
 $\Rightarrow S_{\mathcal{M}}$ beschränkt \Rightarrow eventuell kleines h .

Def Ein Verfahren heißt A-stabil, falls $\{z \in \mathbb{C}; \operatorname{Re} z < 0\} \subset S_H$

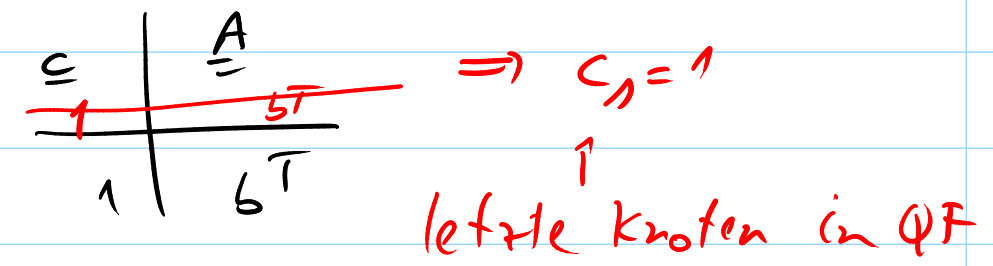


Bsp (iE) ist A-stabil
 RK-Gauss Verfahren sind A-stabil.
 z.B. (iMP) ist A-stabil.

Bew $S(z) = e^z, z \rightarrow -\infty : e^z \rightarrow 0$
 $S(-\infty) = 0$?

Def Num. Verfahren heißt L-stabil falls
 $\lim_{z \rightarrow -\infty} S(z) = 0$

Bew L-stabil falls $\underline{b}^T = a_{11}$ = letzte Zeile in A



\Rightarrow Runge-Kutta-Verfahren (basiert auf Runge-QF)
 L-stabil und hat höchstmögliche Ordnung $(2s-1)$

Runge-Verfahren. von Ordnung 3,5 \rightarrow Skript.

§6.3. Linear-implizite Einschrittverfahren

autonome ODE $\dot{y} = f(y)$ mit impl. RK:
 $f: \mathbb{R}^d \rightarrow \mathbb{R}^d$
 $A \in \mathbb{R}^{s \times s}$

$$\begin{cases} \underline{k}_i = f\left(\underline{y}_0 + h \sum_{j=1}^s a_{ij} \underline{k}_j\right) \\ i=1,2,\dots,s \end{cases}$$
 NLGS mit s Gleichungen.

$$\underline{y}_1 = \underline{y}_0 + h \sum_{i=1}^s b_i \underline{k}_i$$

einfache Idee ☹ linearisiere

$$\begin{cases} \underline{k}_i = f(\underline{y}_0) + h \underline{D}f(\underline{y}_0) \sum_{j=1}^s a_{ij} \underline{k}_j \\ i=1,2,\dots,s \end{cases}$$
 ein LGS mit s Gleichungen.

Bsp $\begin{cases} \dot{y} = \lambda y(1-y), \lambda = 5 \\ y(0) = 0.1 \end{cases}$

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$

Radau mit $n=2$ Stufen.

$\Rightarrow O(h^3)$

mit Linearisierung $\Rightarrow O(h^2)$ ☹️

Idee verwende DIRK (diagonal implizite RK)

$\underline{A} = \begin{bmatrix} \text{triangle} & & \\ & \text{triangle} & \\ & & \text{triangle} \end{bmatrix} \Rightarrow$ gestaffelter System (NLGS)

$$\begin{cases} k_i = f\left(\underline{y}_0 + h \sum_{j=1}^{i-1} a_{ij} k_j + h a_{ii} k_i\right) \\ i = 1, 2, \dots, n \end{cases}$$

\Rightarrow löse n NLGS der Dimension d .

Idee: verwende vereinfachtes Newton Verfahren mit \underline{Df} und $a_{ii} = \delta \Rightarrow$ nur eine LU-Zerleg.

\rightarrow einfach-diagonal-impliziten RK SDIRK-Verfahren.

$$\begin{array}{c|ccc} \delta & \delta & & \\ c_2 & a_{21} \delta & & 0 \\ \vdots & \vdots & \ddots & \\ c_n & \vdots & \dots & a_{n,n-1} \delta \\ \hline & b_1 & b_2 & \dots & b_{n-1} & b_n \end{array}$$

$n=2 \Rightarrow O(h^3)$
 $\delta = \frac{1}{2} + \frac{1}{6}\sqrt{3}$
 $c_2 = 1 - \delta$
 $b_1 = \frac{c_2 - \frac{1}{2}}{c_2 - \delta}, b_2 = \frac{\frac{1}{2} - \delta}{c_2 - \delta}$

3-stufige SDIRK mit $O(h^4)$

$$\begin{array}{c|ccc} \delta & \delta & & \\ \frac{1}{2} & -\frac{\alpha}{2} \delta & & \\ \frac{1-\alpha}{2} & 1+\alpha & -1-2\alpha & \delta \\ \hline & \frac{1}{6\alpha^2} & 1-\frac{1}{3\alpha^2} & \frac{1}{6\alpha^2} \end{array}$$

mit $\alpha^3 - \alpha = \frac{1}{3}$
 $\alpha_1 = \frac{2}{3}\sqrt{3} \cos 10^\circ$
 $-\frac{2}{3}\sqrt{3} \cos 70^\circ$
 $-\frac{2}{3}\sqrt{3} \cos 50^\circ$

Idee: nur ein Schritt von Newton
mit besonders gut gewähltes Startwert.

→ siehe Skript! Row

}7.

Intermezzo: Lineare Algebra

23.04.2024

(praktische Aspekte)

$$\underline{P} \underline{A} = \underline{L} \underline{U}$$

Gauss-Elimination

Reduktion zur oberen Zeilen-Stufenform

$$\underline{A} = \underline{Q} \underline{R} \quad (\underline{Q}^T \underline{Q} = \underline{I})$$

Gram-Schmidt / Rotationen / Spiegelungen

$$\underline{A} = \underline{U} \underline{\Sigma} \underline{V}^T \quad \text{Singularwertzerlegung (SVD)}$$

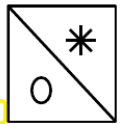
$$\underline{A} = \underline{Q} \underline{T} \underline{Q}^H \quad \text{Schur-Zerlegung.}$$

$$\mathbf{Ax} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \end{bmatrix} \quad \mathbf{Ax} = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + x_3\mathbf{a}_3.$$

Definition 1.1.0.6. Matrix mal Vektor Multiplikation

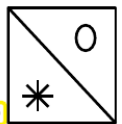
Das Ergebnis **b** der Multiplikation einer Matrix **A** mit einem Vektor **x** (in dieser Reihenfolge) ist die **lineare Kombination von den Spalten der Matrix A mit den Koeffizienten aus dem Vektor x**. Diese Operation ist nur dann möglich, wenn die Matrix **A** genau so viele Spalten wie der Vektor **x** Einträge hat.

Definition 1.2.0.1. Obere Dreiecksmatrix



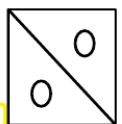
Die obere Dreiecksmatrix hat unterhalb der Hauptdiagonale nur Nullen und beliebige Zahlen oberhalb. Sie wird oft mit **U** für das Englische "upper" oder **R** für das Deutsche "rechts" notiert.

Definition 1.2.0.2. Untere Dreiecksmatrix



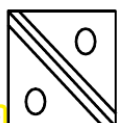
Die untere Dreiecksmatrix hat oberhalb der Hauptdiagonale nur Nullen und beliebige Zahlen unterhalb. Sie wird oft mit **L** für das Englische "lower", oder das Deutsche "links" notiert.

Definition 1.2.0.3. Diagonalmatrix



Die Diagonalmatrix hat nur auf der Hauptdiagonale Zahlen und sonst überall Nullen. Sie wird oft mit **D** notiert. Eine wichtige Diagonalmatrix ist die Identitätsmatrix $\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, bei welcher alle Diagonaleinträge Eins sind.

Definition 1.2.0.4. Tridiagonalmatrix



Die Tridiagonalmatrix hat eine Hauptdiagonale und zwei direkt anliegende Nebendiagonale mit Zahlen und sonst überall Nullen.

Bemerkung 1.6.0.1 (Die Gauss-Elimination liefert die LU-Zerlegung)

Die Gauss-Elimination einer Matrix **A**, bei welcher keine Zeilen vertauscht werden, liefert zwei Matrizen **L**, **U** mit der Eigenschaft, dass

$$\mathbf{A} = \mathbf{LU}.$$

Falls bei der Gauss-Elimination auch Zeilen vertauscht werden, so bekommt man noch eine dritte Matrix **P**, die Permutationsmatrix, und es gilt

$$\mathbf{PA} = \mathbf{LU}.$$

Die Matrix **A** wurde also in eine untere Dreiecksmatrix **L** und eine Matrix mit oberer Zeilenstufenform **U** zerlegt; dies wird *LU-Zerlegung* oder auch *LR-Zerlegung* einer Matrix genannt.

Bemerkung 1.6.0.2 (LGS löst man mit der LU-Zerlegung, nicht mit der Inversen)

Wo müssen wir bei der Gauss-Elimination am meisten Rechenarbeit leisten? Wenn ein LGS $\mathbf{Ax} = \mathbf{b}$ gelöst werden muss, werden die folgenden Schritte gemacht:

$$\mathbf{Ax} = \mathbf{b} \xrightarrow{\mathbf{P}} \mathbf{PAx} = \mathbf{Pb} \implies \mathbf{L} \underbrace{\mathbf{Ux}}_{\mathbf{y}} = \mathbf{Pb}.$$

$$\mathbf{A} = \mathbf{LU} = \mathbf{LD}\tilde{\mathbf{U}}.$$

Die Einträge von **D** sind die Diagonaleinträge d_1, d_2, \dots, d_n von **U**. Die Einträge von $\tilde{\mathbf{U}}$ entsprechen denen von **U**, wobei die *i*-te Zeile aber durch das dazugehörige d_i geteilt wurde:

$$\mathbf{D} = \begin{bmatrix} d_1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & & \\ 0 & \dots & 0 & d_n \end{bmatrix} \quad \tilde{\mathbf{U}} = \begin{bmatrix} 1 & \frac{u_{12}}{d_1} & \dots & \frac{u_{1n}}{d_1} \\ 0 & & & \\ \vdots & & & \\ 0 & \dots & 0 & 1 \end{bmatrix}.$$

LDU-Zerlegung symmetrischer Matrizen

Die transponierte Matrix A^T von A lässt sich also schreiben als:

$$A^T = \tilde{U}^T D L^T.$$

Wenn wir auch noch die Annahme treffen, dass die Matrix A symmetrisch ist, dass also gilt $A = A^T$ so haben wir $L^T = \tilde{U}$ und $\tilde{U}^T = L$, denn wir haben gesehen, dass die LU-Zerlegung einer regulären Matrix eindeutig ist.

In diesem Fall liefert die Gauss-Elimination die folgende Zerlegung einer symmetrischen regulären Matrix:

$$A = LDL^T.$$

Cholesky-Zerlegung

Wenn nun auch noch gilt, dass alle (Pivote in der Gauss-Elimination) $d_1, \dots, d_n > 0$, so kann für die symmetrische Matrix A die sogenannte *Cholesky-Zerlegung* definiert werden.

Für diese Zerlegung wird D in zwei identische Matrizen zerteilt $D = \sqrt{D} \sqrt{D}$ mit

$$\sqrt{D} = \begin{bmatrix} \sqrt{d_1} & & 0 \\ 0 & \dots & \sqrt{d_n} \end{bmatrix}.$$

Dies wird nun mit der LDU-Zerlegung von symmetrischen Matrix kombiniert um eine weitere Zerlegung zu erhalten:

$$A = LDL^T = \underbrace{L \sqrt{D}}_{R^T} \underbrace{\sqrt{D} L^T}_R = R^T R$$

$$A = R^T R, .$$

Das R ist eine obere Dreiecksmatrix. Diese Zerlegung von A in $R^T R$ ist die *Cholesky-Zerlegung*. Die Cholesky-Zerlegung hat wichtige Anwendungen in der Statistik, in der Physik und in der Numerik. Man kann die Symmetrie der Matrix A ausnutzen, um die Cholesky-Zerlegung mit nur einem Drittel der Operationen, die man für die standardmässige Gauss-Elimination braucht, zu realisieren.

Definition 1.7.0.1. Orthogonale Vektoren

Zwei Vektoren $x, y \in \mathbb{R}^n$ (keiner davon darf Null sein), heissen orthogonal, notiert als $x \perp y$, falls

$$\langle x, y \rangle = 0.$$

Definition 1.7.0.2. Orthogonale Matrix

Eine reelle $n \times n$ Matrix A heisst orthogonal, wenn

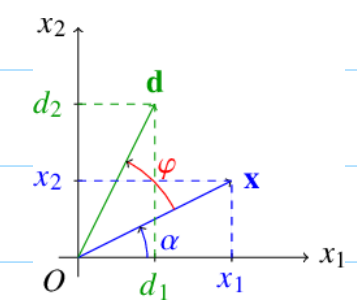
$$A^T A = I.$$

Eine komplexe $n \times n$ Matrix A heisst unitär, wenn

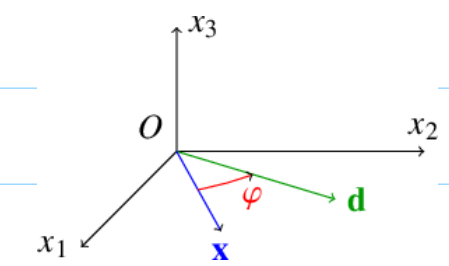
$$A^H A = I.$$

Satz 1.7.0.6. Erhaltungssatz

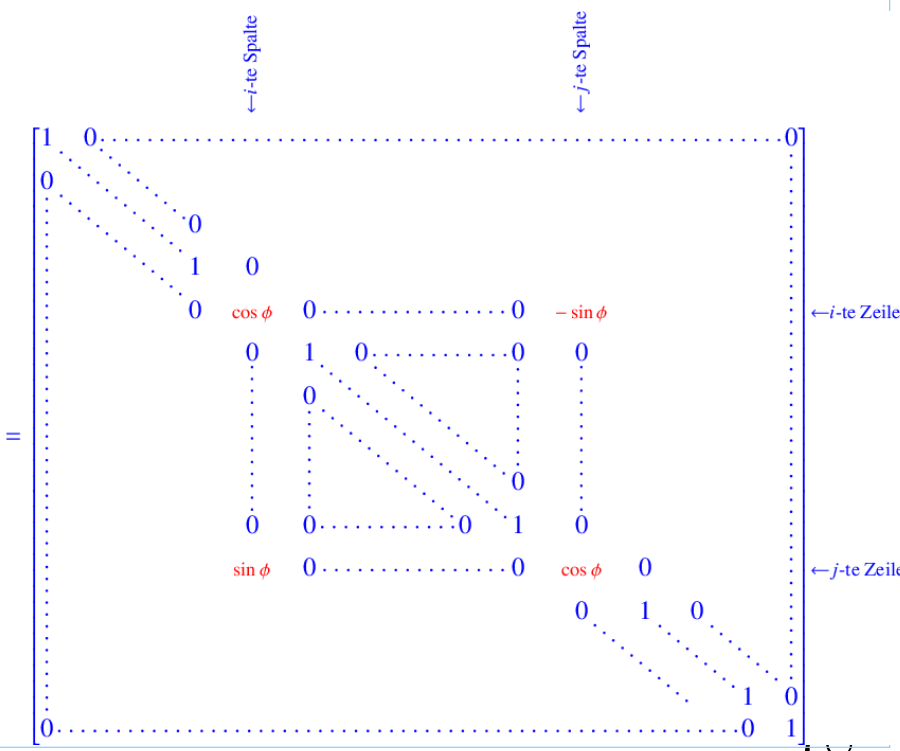
Orthogonale Matrizen verändern Längen und Winkel nicht.

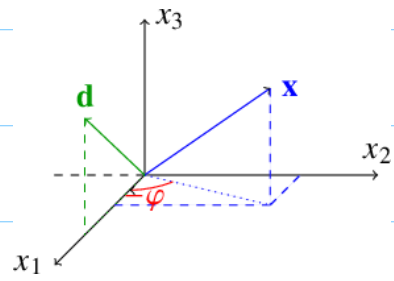


$$D(\phi) = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}.$$

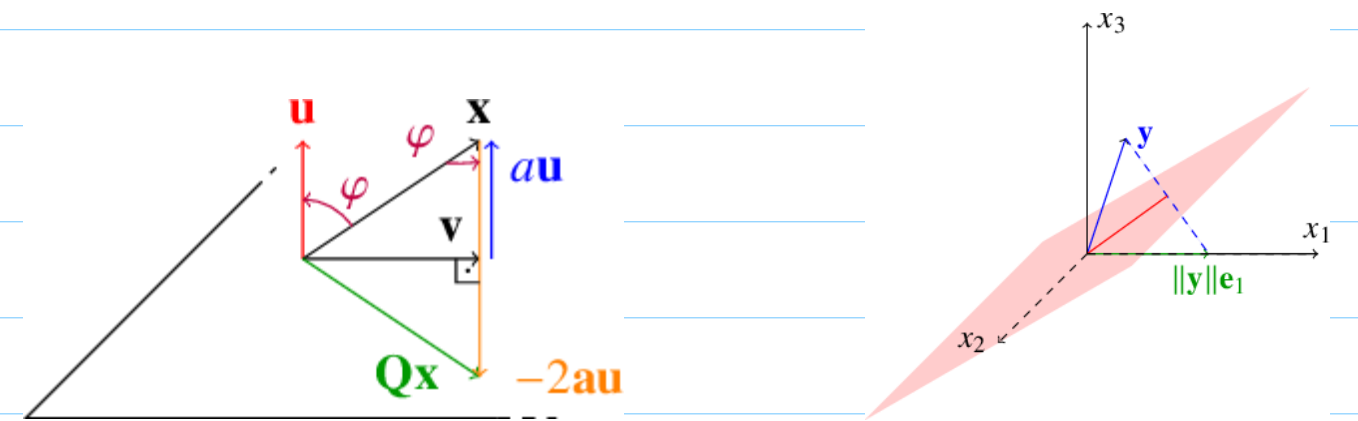


$$D_{ij}(\phi) =$$





$$D_{ij}(-\phi) = \begin{bmatrix} x_1 & & & & & \\ & \ddots & & & & \\ & & x_j & & & \\ & & & \ddots & & \\ & & & & x_j & \\ & & & & & \ddots \\ & & & & & & x_n \end{bmatrix} = \begin{bmatrix} x_1 & & & & & \\ & \ddots & & & & \\ & & r & & & \\ & & & \ddots & & \\ & & & & 0 & \\ & & & & & \ddots \\ & & & & & & x_n \end{bmatrix}$$



$$G_{1k}(a_1 a_k) \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ \vdots \\ a_n \end{bmatrix} := \begin{bmatrix} \bar{\gamma} & \cdots & \bar{\sigma} & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots \\ -\sigma & \cdots & \gamma & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} a_1^{(1)} \\ \vdots \\ \mathbf{0} \\ \vdots \\ a_n \end{bmatrix}, \quad \begin{aligned} \gamma &= \frac{a_1}{\sqrt{|a_1|^2 + |a_k|^2}}, \\ \sigma &= \frac{a_k}{\sqrt{|a_1|^2 + |a_k|^2}}. \end{aligned}$$

Definition 1.7.0.13. Householder Matrix
 $Q = I - 2uu^T$ ist die Householder Matrix, welche einen Vektor an einer Ebene mit normalen Vektor u spiegelt. Der normalen Vektor u muss Norm 1 haben, falls dies nicht der Fall ist, muss anstelle von u mit $\frac{u}{\|u\|}$ gerechnet werden.

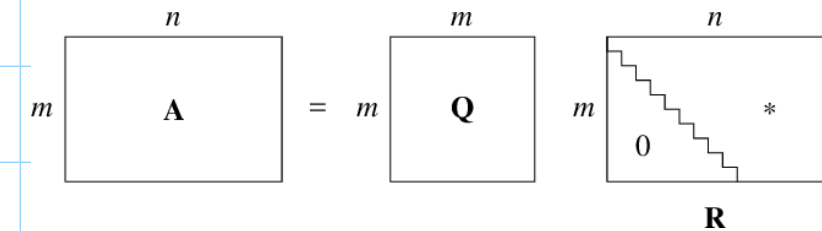
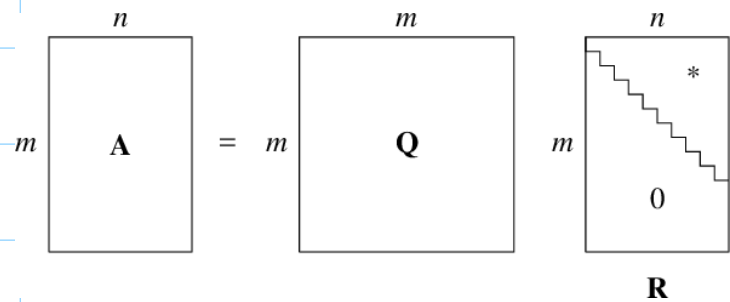
Definition 1.8.0.1. QR-Zerlegung
 Die QR-Zerlegung ist die Zerlegung einer Matrix A in eine orthogonale Matrix Q und eine obere Dreiecksmatrix R , so dass $A = QR$.

Man verwendet die folgende Konvention für die Givens-Drehungen ($\mathbb{K} = \mathbb{R}$):

$$G = \begin{bmatrix} \bar{\gamma} & \bar{\sigma} \\ -\sigma & \gamma \end{bmatrix} \implies \text{speichere } \rho := \begin{cases} 1, & \text{if } \gamma = 0, \\ \frac{1}{2} \text{sign}(\gamma)\sigma, & \text{if } |\sigma| < |\gamma|, \\ 2 \text{sign}(\sigma)/\gamma, & \text{if } |\sigma| \geq |\gamma|. \end{cases}$$

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

Man speichert die Rotationsmatrix $G_{ij}(ab)$ als (i, j, ρ) , was effizient ist, falls die Matrix A dünnbestetzt ist.



$$Q_2 Q_1 A = \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{bmatrix} \quad Q_{n-1} \dots Q_2 Q_1 A = R \implies A = QR.$$

Satz 1.8.0.6. QR-Zerlegung einer allgemeinen Matrix

Für jedes $A \in \mathbb{R}^{m \times n}$ mit $m \geq n$ und $\text{Rang}(A) = n$ gibt es eine eindeutige orthogonale Matrix $Q \in \mathbb{R}^{m \times m}$ so dass

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$$

mit R obere Dreiecksmatrix mit allen Diagonalelementen ≥ 0 .

Falls alle Diagonalelemente > 0 dann ist R die Cholesky-Zerlegung von $A^T A$.

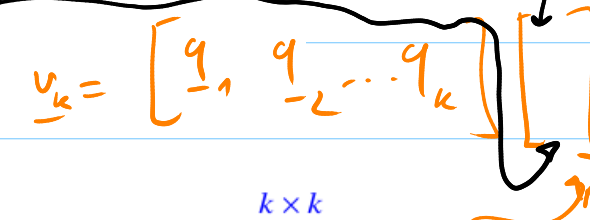
$$A^T A = \begin{bmatrix} R^T & 0 \end{bmatrix} \underbrace{Q^T Q}_I \begin{bmatrix} R \\ 0 \end{bmatrix} = R^T R + 0 \cdot 0 = R^T R.$$

$$c_k = v_k - P_{\text{Span}\{q_1, \dots, q_{k-1}\}}^\perp v_k = v_k - \langle q_1, v_k \rangle q_1 - \langle q_2, v_k \rangle q_2 - \dots - \langle q_{k-1}, v_k \rangle q_{k-1}$$

$$q_k = \frac{c_k}{\|c_k\|}$$

$$\implies c_k = \|c_k\| q_k = \langle q_k, v_k \rangle q_k$$

$$\implies v_k = \langle q_1, v_k \rangle q_1 + \langle q_2, v_k \rangle q_2 + \dots + \langle q_k, v_k \rangle q_k.$$



$$\begin{bmatrix} | & | & \dots & | \\ v_1 & v_2 & \dots & v_k \\ | & | & \dots & | \end{bmatrix} = \begin{bmatrix} | & | & \dots & | \\ q_1 & q_2 & \dots & q_k \\ | & | & \dots & | \end{bmatrix} \begin{bmatrix} q_1^H v_1 & q_1^H v_2 & \dots & q_1^H v_k \\ 0 & q_2^H v_2 & \dots & q_2^H v_k \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & q_k^H v_k \end{bmatrix}$$

$\underbrace{\hspace{10em}}_{A_k} = \underbrace{\hspace{10em}}_{Q_k} \underbrace{\hspace{10em}}_{R_k}$

$$\begin{bmatrix} | & \dots & | \\ v_1 & \dots & v_n \\ | & \dots & | \end{bmatrix} = \begin{bmatrix} | & \dots & | \\ q_1 & \dots & q_n \\ | & \dots & | \end{bmatrix} \begin{bmatrix} * & \dots & * \\ \vdots & \ddots & \vdots \\ 0 & \dots & * \end{bmatrix}$$

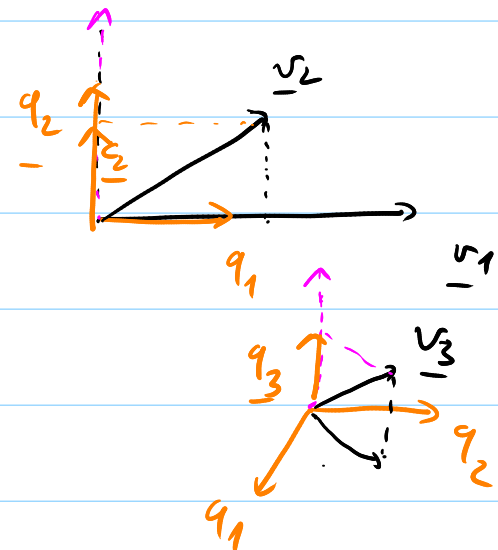
$\underbrace{\hspace{10em}}_{A_n} = \underbrace{\hspace{10em}}_{Q_n} \underbrace{\hspace{10em}}_{R_n}$

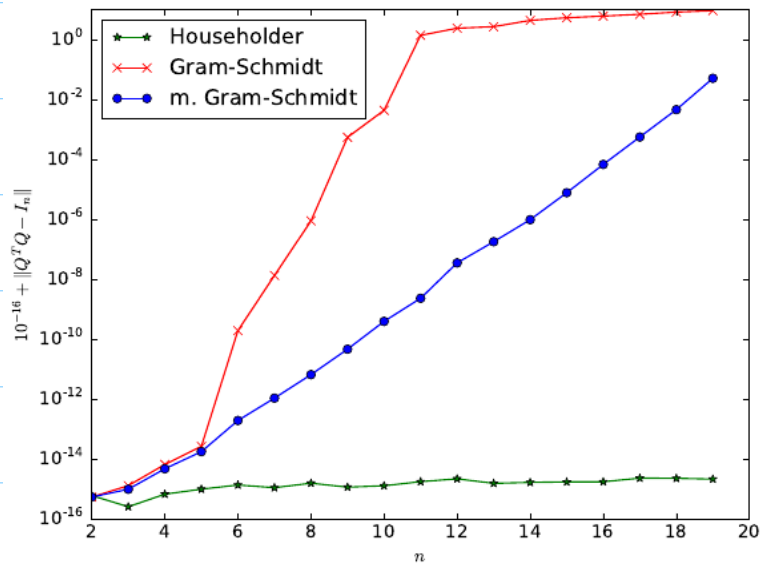
Gram-Schmidt:

$q_1 :$ $q_1 = \frac{1}{\|v_1\|} v_1$
 $\implies v_1 = \|v_1\| q_1 = \langle q_1, v_1 \rangle q_1.$

$q_2 :$ $c_2 = v_2 - P_{q_1}^\perp v_2 = v_2 - \langle q_1, v_2 \rangle q_1$
 $q_2 = \frac{c_2}{\|c_2\|}$
 $\implies c_2 = \|c_2\| q_2 = \langle q_2, v_2 \rangle q_2$
 $\implies v_2 = \langle q_1, v_2 \rangle q_1 + \langle q_2, v_2 \rangle q_2.$

$q_3 :$ $c_3 = v_3 - P_{\text{Span}\{q_1, q_2\}}^\perp v_3 = v_3 - \langle q_1, v_3 \rangle q_1 - \langle q_2, v_3 \rangle q_2$
 $q_3 = \frac{c_3}{\|c_3\|}$
 $\implies c_3 = \|c_3\| q_3 = \langle q_3, v_3 \rangle q_3$
 $\implies v_3 = \langle q_1, v_3 \rangle q_1 + \langle q_2, v_3 \rangle q_2 + \langle q_3, v_3 \rangle q_3.$





Bemerkung 7.1.31. Es gibt mehrere Möglichkeiten, die QR-Zerlegung einer Matrix A zu berechnen:

- 1) Der modifizierte Gram-Schmidt Algorithmus orthogonalisiert A via trianguläre Operationen (obere Dreiecksmatrizen); Vorteil: wir können früher aufhören und haben schon die ersten orthogonalen Spalten. Nachteil: die Stabilität ist nicht garantiert.
- 2) Orthogonale Transformationen (Householder-Spiegelungen für eine vollbesetzte Matrix oder Givens-Rotationen für eine dünnbestetzte Matrix) triangularisieren A ; Vorteil: Q ist orthogonal trotz Rundungsfehler.

→ billig, Rundungsfehler sind tödlich 😊 ☹️

→ teuer ☹️, Q immer orthogonal!

Gegeben $\underline{a}_1, \underline{a}_2, \dots, \underline{a}_n$ GS

für $j=1, 2, \dots, n$

$\underline{v} = \underline{a}_j$

für $i=1, 2, \dots, j-1$

$r_{ij} = \underline{q}_i^T \underline{a}_j$

$\underline{v} = \underline{v} - r_{ij} \underline{q}_i$

$r_{jj} = \|\underline{v}\|$

$\underline{q}_j = \underline{v} \cdot \frac{1}{r_{jj}}$

modifizierte GS

$r_{ij} = \underline{q}_i^T \underline{v}$

d.h. Projektion auf die gefundene \underline{q}_i wird.

$\underbrace{P_{q_{j-1}} \dots P_{q_2} P_{q_1}}_{i=j-1} \underline{v}$

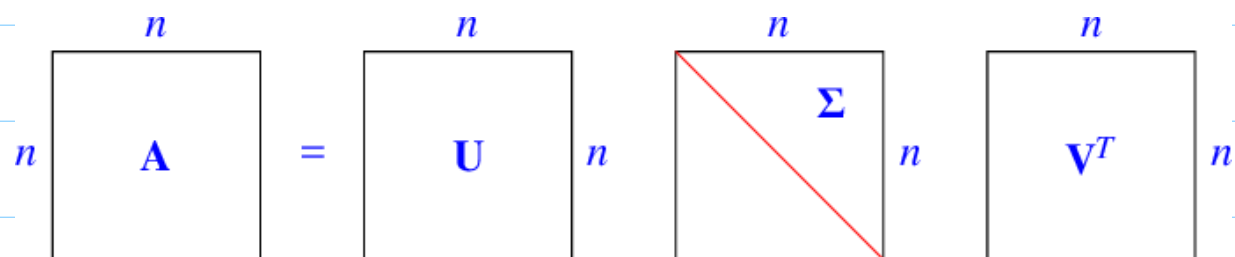
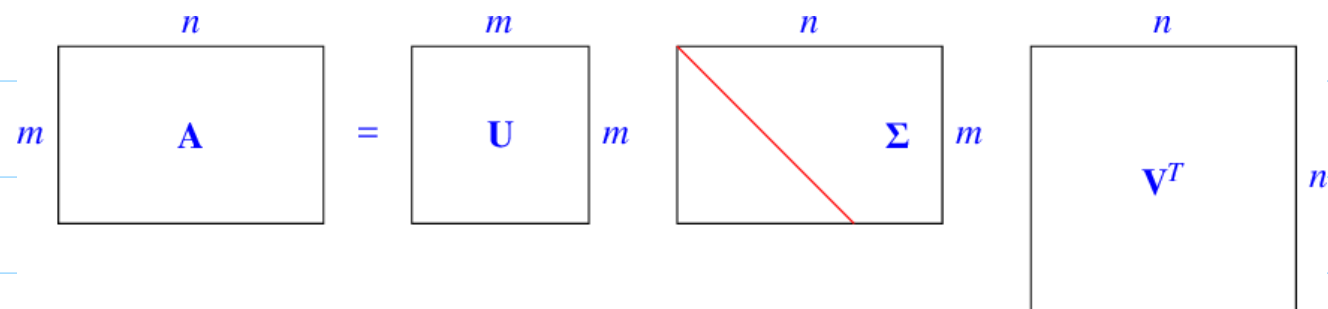
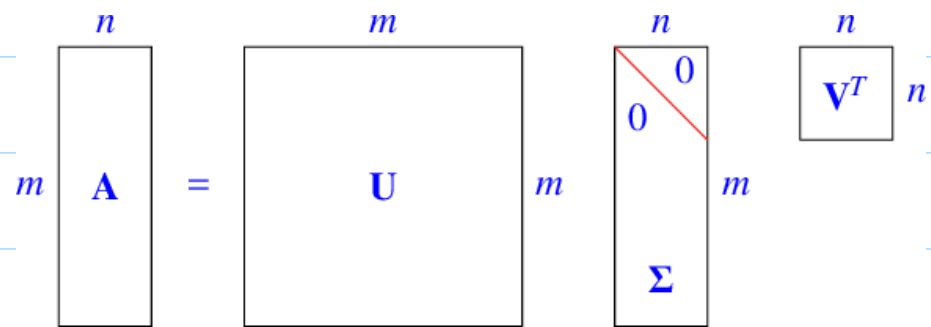
Satz 8.1.0.1. Singulärwertzerlegung (Singular Value Decomposition/SVD)

Sei eine Matrix $A \in \mathbb{R}^{m \times n}$ beliebig. Es gibt zwei orthogonale Matrizen $U \in \mathbb{R}^{m \times m}$ und $V \in \mathbb{R}^{n \times n}$ sodass:

$$A = U \Sigma V^T, \tag{8.1.0.2}$$

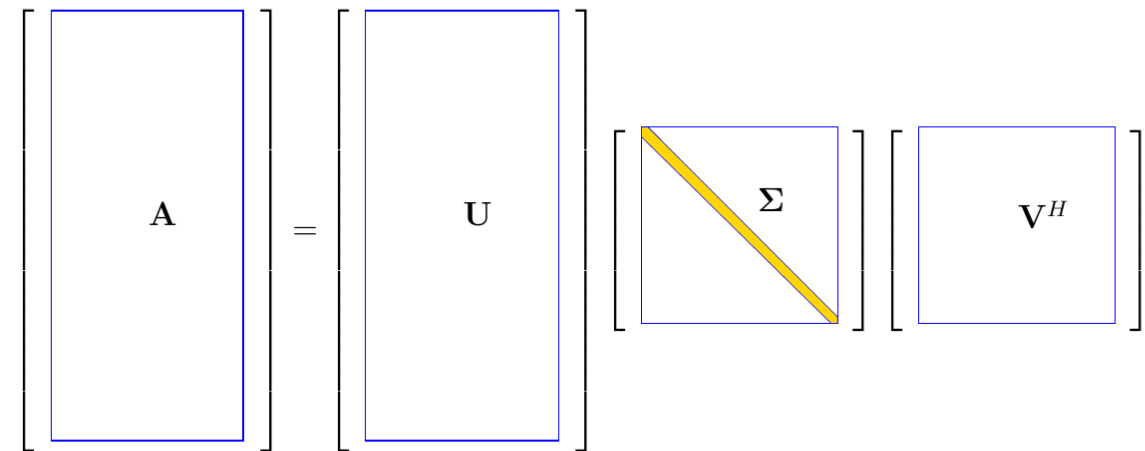
wobei die Matrix $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_p) \in \mathbb{R}^{m \times n}$ ($p = \min(m, n)$) die Singulärwerte der Matrix A auffasst. Die Singulärwerte stehen auf der Hauptdiagonale von Σ und sind der Grösse nach absteigend geordnet: σ_1 ist der grösste und σ_p ist der kleinste Singulärwert:

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0. \tag{8.1.0.3}$$



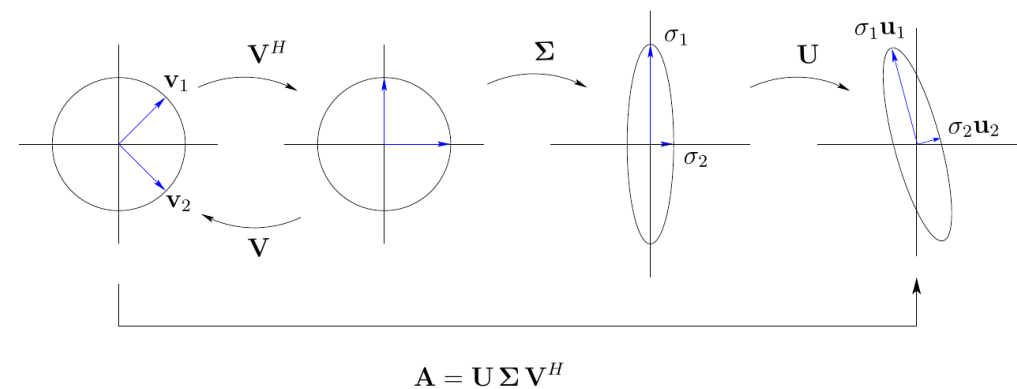
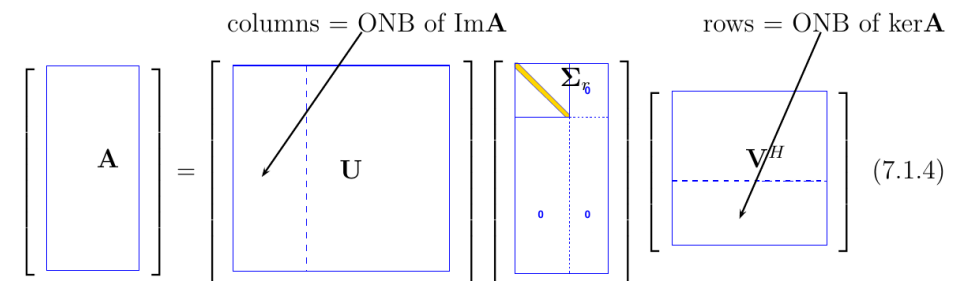
Die entsprechende Python-Funktion ist: `scipy.linalg.svd`.

`full_matrices=False`: sparsame SVD:



Komplexität:

$$\begin{aligned} &2mn^2 + 2n^3 + O(n^2) + O(mn) && \text{für } s = \text{svdvals}(A), \\ &4m^2n + 22n^3 + O(mn) + O(n^2) && \text{für } U, S, V = \text{svd}(A), \\ &O(mn^2) + O(n^3) && \text{für } U, S, V = \text{svd}(A, \text{full_matrices=False}), m \gg n. \end{aligned}$$



Satz 8.1.0.4. Fundamentalsatz der linearen Algebra - Teil II.

Mit den Notationen aus der Singulärwertzerlegung gilt:

1. $\mathbf{v}_{r+1}, \dots, \mathbf{v}_n$ ist eine Orthonormalbasis vom Kern von \mathbf{A} ;
2. $\mathbf{v}_1, \dots, \mathbf{v}_r$ spannen das Bild von \mathbf{A}^T und heissen rechte Singulärvektoren von \mathbf{A} ;
3. $\mathbf{u}_1, \dots, \mathbf{u}_r$ spannen das Bild von \mathbf{A} und heissen linke Singulärvektoren von \mathbf{A} ;

4.

$$\mathbf{u}_1^T \mathbf{A} \mathbf{v}_1 = \sigma_1, \quad \mathbf{u}_2^T \mathbf{A} \mathbf{v}_2 = \sigma_2, \dots, \mathbf{u}_r^T \mathbf{A} \mathbf{v}_r = \sigma_r;$$

5.

$$\mathbf{u}_1 = \frac{1}{\sigma_1} \mathbf{A} \mathbf{v}_1 \Rightarrow \mathbf{A} \mathbf{v}_1 = \sigma_1 \mathbf{u}_1$$

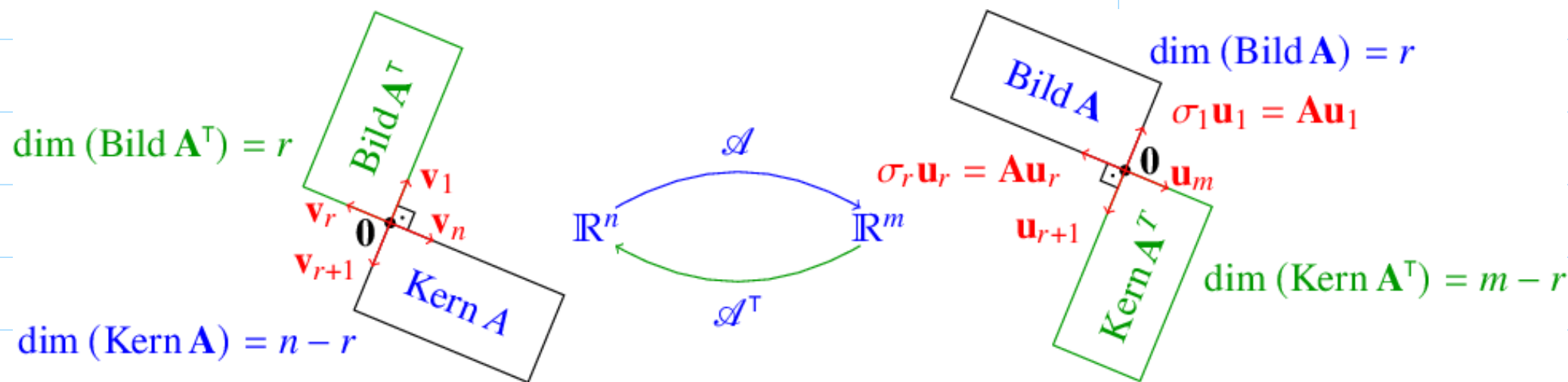
\vdots

$$\mathbf{u}_r = \frac{1}{\sigma_r} \mathbf{A} \mathbf{v}_r \Rightarrow \mathbf{A} \mathbf{v}_r = \sigma_r \mathbf{u}_r$$

$$\mathbf{A} \mathbf{v}_{r+1} = 0$$

\vdots

$$\mathbf{A} \mathbf{v}_n = 0.$$



8. Ausgleichsrechnung

8.1. Lineare Ausgleichsrechnung

Modell: $\underline{d} \in \mathbb{R}^D, c \in \mathbb{R}$
 $\mathbb{R} \ni b = \underline{d}^T \underline{t} + c \quad \underline{t} \in \mathbb{R}^s$

Messpunkte: t_1, t_2, \dots, t_m

Gemessen: b_1, b_2, \dots, b_m

$$\Rightarrow \begin{cases} \underline{d}^T \underline{t}_1 + c - b_1 = r_1 \\ \dots \\ \underline{d}^T \underline{t}_m + c - b_m = r_m \end{cases} \Rightarrow \underline{r} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix} \quad \text{Residuum}$$

$$\min_{\underline{p} \in \mathbb{R}^n, q \in \mathbb{R}} \{|r_1|^2 + \dots + |r_m|^2\} = \min_{\underline{p} \in \mathbb{R}^n, q \in \mathbb{R}} \sum_{i=1}^s |\underline{p}^T \underline{t}_i + q - b_i|^2 = \min_{q \in \mathbb{R}, \underline{p} \in \mathbb{R}^n} \sum_{i=1}^s |1 \cdot q + \underline{t}_i^T \underline{p} + q - b_i|^2$$

$$\underline{x} = \begin{bmatrix} q \\ \underline{p} \end{bmatrix} \in \mathbb{R}^n, \quad n = D+1$$

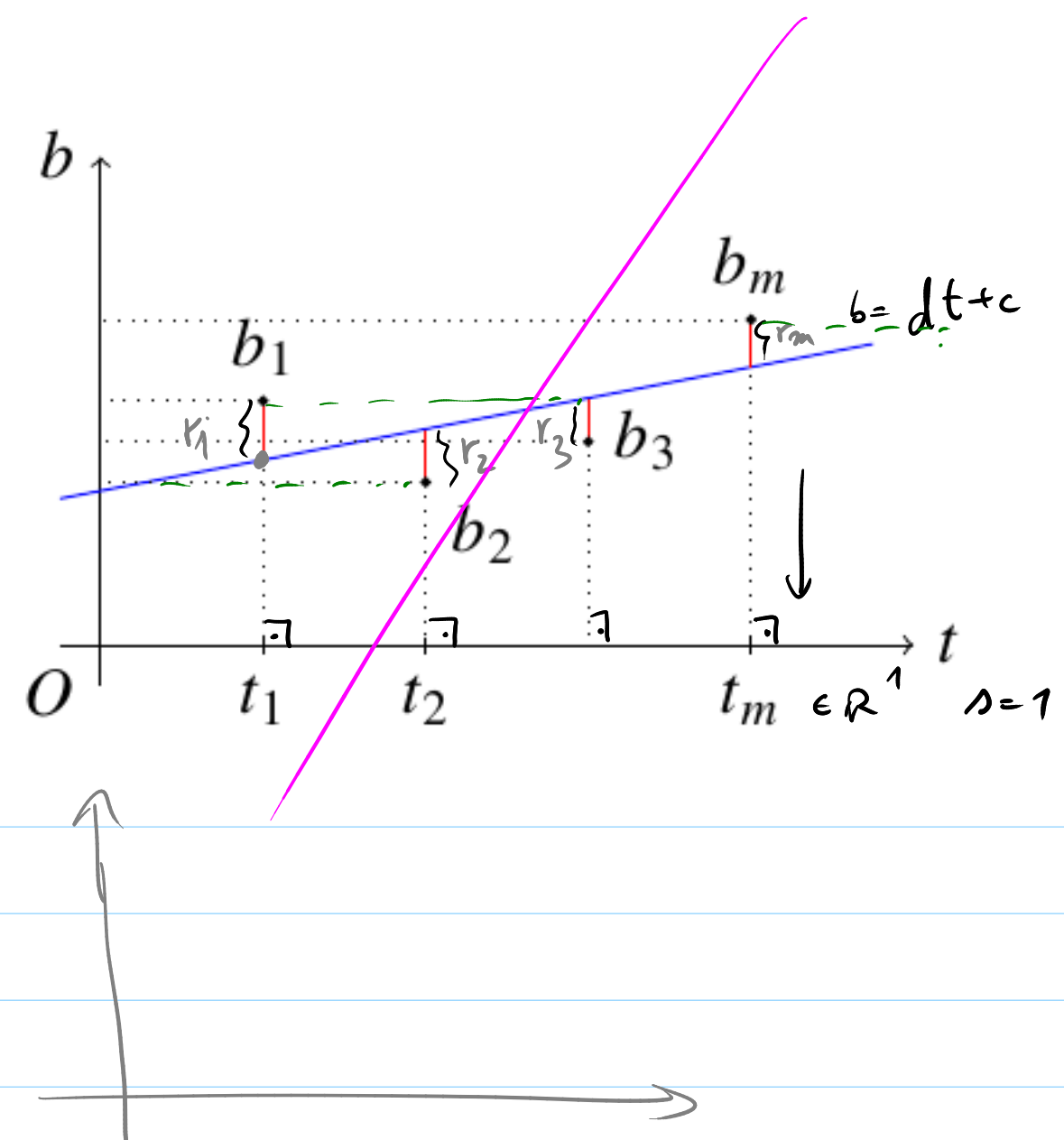
$$\mathbf{A} = \begin{bmatrix} 1 & \underline{t}_1^T \\ 1 & \underline{t}_2^T \\ \vdots & \vdots \\ 1 & \underline{t}_m^T \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$\min_{\underline{x} \in \mathbb{R}^n} \|\mathbf{A}\underline{x} - \mathbf{b}\|_2^2$$

Definition 5.1.0.2. Lineare Ausgleichsrechnung als lineares Gleichungssystem

Sei eine $m \times n$ Matrix \mathbf{A} und der Vektor \mathbf{b} mit m Einträgen. Gesucht ist ein $\hat{\mathbf{x}}$, so dass

$$\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_2 = \min_{\underline{x} \in \mathbb{R}^n} \|\mathbf{A}\underline{x} - \mathbf{b}\|_2$$



Beispiel 5.1.0.4 (Modell im Unterraum)

Sei $f \in V$, wobei V ein ∞ -dimensionaler linearer Raum (z.B. L^2) ist. Sei nun $V_n \subset V$ ein endlichdimensionaler Unterraum. Wir wollen eine Funktion $f_n \in V_n$ finden, die $f \in V$ im gewissen Sinne approximiert.

Sei b_1, \dots, b_n eine Basis in V_n . Das Ziel ist also:

$$f \approx f_n = \sum_{j=1}^n x_j b_j.$$

Ist das System z.B. zeitabhängig, so wählen wir eine zeitabhängige Basis:

$$f(t) \approx f_n(t) = \sum_{j=1}^n x_j b_j(t).$$

Verwenden wir nur die Werte/Messungen $y_i = f(t_i)$, so bekommen wir sehr einfach eine Approximation: Für die gegebenen $t_i, y_i, i = 1, 2, \dots, m$, finden wir die Koeffizienten x_1, \dots, x_n , sodass die Summe der Quadrate der punktwisen Approximationsfehler

$$\sum_{i=1}^m |f_n(t_i) - y_i|^2,$$

Nehmen wir beispielsweise an, dass $V = L^2(0, 1) = \{f : [0, 1] \rightarrow \mathbb{R}, \int_0^1 |f(t)|^2 dt < \infty\}$ und wählen die Basis der Monome $b_i(t) = t^{i-1}$, so dass wir V durch eine Polynommenge \mathcal{P} beschreiben können ($V_n = \mathcal{P}_n$). Dieser Ansatz produziert dann das »beste« Polynom p_n vom Grad maximal $n - 1$, so dass die euklidische Norm des Residuumsvektor

$$\sum_{i=1}^m |p_n(t_i) - y_i|^2$$

minimal ist.

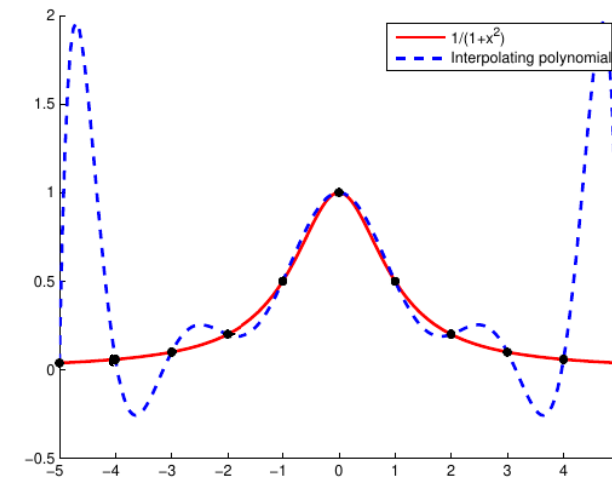
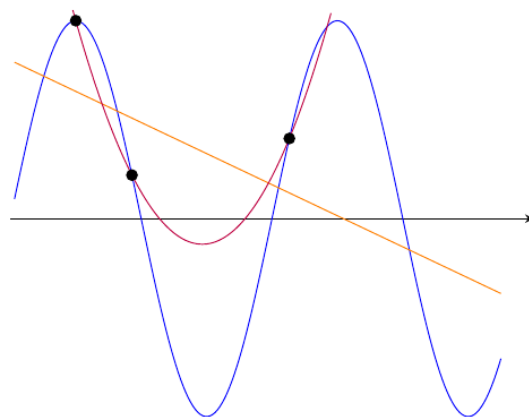


Abb. 3.4.18. Äquidistante Stützstellen

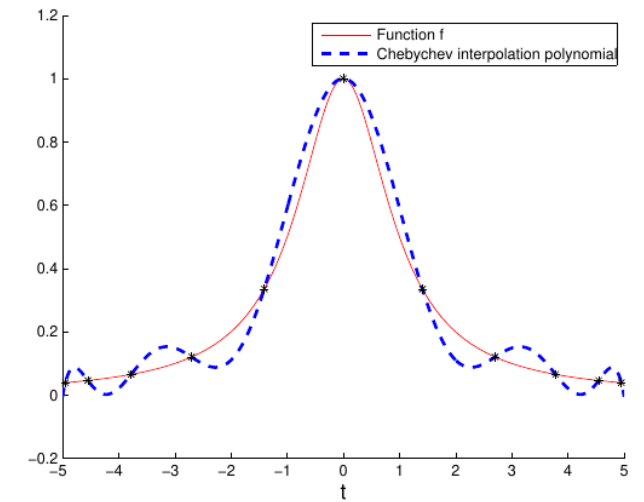


Abb. 3.4.19. Chebyshev-Knoten

Definition 5.2.0.1. Normalengleichung
Die Gleichung $A^T A x = A^T b$ heisst *Normalengleichung*.

Satz 5.2.0.2. Lösung der Ausgleichsrechnung mit der Normalengleichung
Sei eine Matrix $A \in \mathbb{R}^{m \times n}$ gegeben. Falls der Rang dieser Matrix gleich n ist, so ist $A^T A$ invertierbar. In diesem Fall hat die Normalengleichung also genau eine Lösung, die die Lösung des linearen Ausgleichsproblems ist.

8.2 Lineare Ausgleichsrechnung: via orthogonaler Transformationen

Wir schauen uns eine Alternativberechnung an, um numerische Fehler für grosse Matrizen zu vermeiden. Dazu bedienen wir uns wieder bei der QR-Zerlegung:

$$A = QR = Q \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}.$$

Wenn die $m \times n$ Matrix A Rang n hat, dann hat die $m \times n$ Matrix R auch Rang n , so dass die ersten n Zeilen eine $n \times n$ invertierbare obere Dreiecksmatrix \hat{R} sind. Q ist eine $m \times m$ orthogonale (bzw. unitäre) Matrix: $Q^T Q = I$, bzw. $Q^H Q = I$. Da orthogonale/unitäre Matrizen die Längen nicht verändern, haben wir:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2 &= \min_{x \in \mathbb{R}^n} \|QRx - QQ^H b\|_2^2 = \min_{x \in \mathbb{R}^n} \|Q(Rx - Q^H b)\|_2^2 \\ &= \min_{x \in \mathbb{R}^n} \|Rx - Q^H b\|_2^2. \end{aligned}$$

Definieren wir nun einen neuen Vektor $\hat{b} = Q^H b$. Wir fassen die ersten n Komponenten von \hat{b} im Vektor c zusammen und können dann die Block-Matrix-Schreibweise verwenden:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \left\| \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} - \begin{bmatrix} \hat{b}_1 \\ \vdots \\ \hat{b}_n \\ \hat{b}_{n+1} \\ \vdots \\ \hat{b}_m \end{bmatrix} \right\|_2 &= \min_{x \in \mathbb{R}^n} \left\| \begin{bmatrix} \hat{R}x \\ 0 \end{bmatrix} - \begin{bmatrix} c \\ \hat{b}_{n+1} \\ \vdots \\ \hat{b}_m \end{bmatrix} \right\|_2 \\ &= \min_{x \in \mathbb{R}^n} \left\| \begin{bmatrix} \hat{R}x - c \\ \hat{b}_{n+1} \\ \vdots \\ \hat{b}_m \end{bmatrix} \right\|_2 = \min_{x \in \mathbb{R}^n} (\|\hat{R}x - c\|_2^2 + \underbrace{|\hat{b}_{n+1}|^2 + \dots + |\hat{b}_m|^2}_{= 0 + |\hat{b}_{n+1}|^2 + \dots + |\hat{b}_m|^2}) \end{aligned}$$

falls $\hat{R}x = c$ ist. Wir sehen, dass in der letzten Gleichung die Terme ab \hat{b}_{n+1} unabhängig von x sind. Dieses x ist somit die Lösung im Sinne der kleinsten Quadrate, wobei der Fehler gleich der Norm des Residuums ist.

Das Residuum $r = Ax - b$ wird als eine weitere Unbekannte behandelt. Man kann die Gleichung (8.1.4) folgendermassen umschreiben:

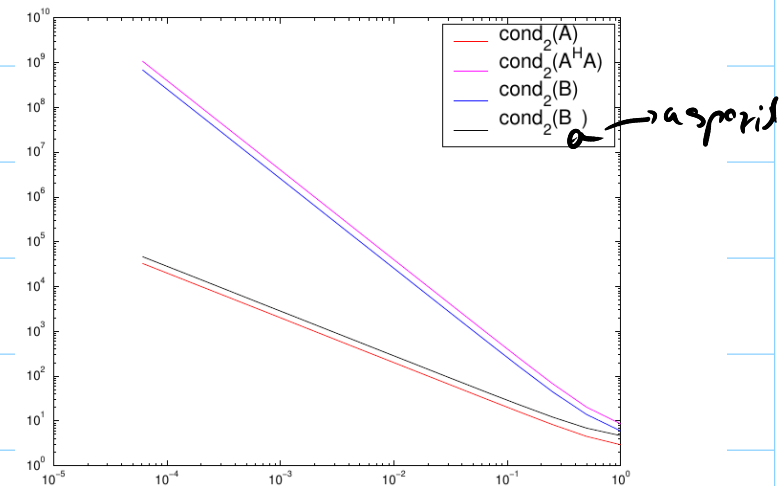
$$A^H Ax = A^H b \Leftrightarrow B \begin{bmatrix} r \\ x \end{bmatrix} := \begin{bmatrix} -I & A \\ A^H & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}. \tag{8.1.5}$$

Allgemeiner definiert man $r := \frac{1}{a}(Ax - b)$, wobei $a > 0$ gilt; dann kann man in Gleichung (8.1.5) die Matrix B durch

$$B_a := \begin{bmatrix} -aI & A \\ A^H & 0 \end{bmatrix}$$

ersetzen. Dabei wählt man a so, dass $\kappa(B_a)$ minimal wird. Die Umformung (8.1.5) von Gleichung (8.1.4) sowie ihre Variante führen bei dünn besetztem A wieder zu einer dünn besetzten Matrix B_a und sind somit mit Hilfe von effizienten Verfahren für dünn besetzte Matrizen lösbar.

$$A = \begin{bmatrix} 1 + \epsilon & 1 \\ 1 - \epsilon & 1 \\ \epsilon & \epsilon \end{bmatrix}.$$



$$\|\mathbf{Ax} - \mathbf{b}\|_2 = \|\mathbf{U}\Sigma\mathbf{V}^H\mathbf{x} - \mathbf{b}\|_2 = \|\Sigma\mathbf{V}^H\mathbf{x} - \mathbf{U}^H\mathbf{b}\|_2.$$

Wenn $\text{rank}(A) = r < \min(m, n)$, dann

$$\Sigma = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix},$$

und alle Zeilenvektoren \mathbf{v}_j der Matrix \mathbf{V} , für die gilt: $j > r$, liegen im Kern von \mathbf{A} . Ähnliches gilt auch für die Matrix \mathbf{U} . Beide werden folgendermassen unterteilt:

$$\mathbf{A} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{bmatrix}$$

mit je orthonormalen Spalten in den Matrizen U_1, U_2, V_1 und V_2 :

$$\mathbf{U}_1 \in \mathbb{K}^{m \times r}, \mathbf{U}_2 \in \mathbb{K}^{m \times (n-r)}, \mathbf{V}_1 \in \mathbb{K}^{n \times r}, \mathbf{V}_2 \in \mathbb{K}^{n \times (n-r)}.$$

Damit gilt für das Residuum:

$$\|\mathbf{Ax} - \mathbf{b}\|_2 = \left\| \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^H \\ \mathbf{V}_2^H \end{bmatrix} \mathbf{x} - \mathbf{b} \right\|_2 = \left\| \begin{bmatrix} \Sigma_r \mathbf{V}_1^H \mathbf{x} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{U}_1^H \mathbf{b} \\ \mathbf{U}_2^H \mathbf{b} \end{bmatrix} \right\|_2.$$

Somit wird das Residuum genau dann minimal, wenn

$$\Sigma_r \mathbf{V}_1^H \mathbf{x} = \mathbf{U}_1^H \mathbf{b} \Leftrightarrow \mathbf{V}_1^H \mathbf{x} = \Sigma_r^+ \mathbf{U}_1^H \mathbf{b}, \tag{8.1.8}$$

wobei $\Sigma_r^+ = \text{diag}(\frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_r})$ die "Inverse" von Σ_r ist. Das Residuum dazu ist $\|\mathbf{U}_2^H \mathbf{b}\|_2$. Wir definieren nun

$$\mathbf{x} = \mathbf{V}_1 \Sigma_r^+ \mathbf{U}_1^H \mathbf{b}, \tag{8.1.9}$$

Definition 8.2.2.1. Pseudoinverse nach Moore (1920) und Penrose (1955)

Sei $\mathbf{A} \in \mathbb{R}^{m \times n}$ gegeben und sei $\mathbf{U}\Sigma\mathbf{V}^T$ ihre Singulärwertzerlegung. Die Pseudoinverse (Moore-Penrose) ist dann definiert als:

$$\mathbf{A}^\dagger = \mathbf{V}\Sigma^\dagger\mathbf{U}^H,$$

mit der $n \times m$ Diagonalmatrix:

$$\Sigma^\dagger = \begin{bmatrix} \sigma_1^{-1} & 0 & \dots & 0 \\ 0 & \sigma_2^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_r^{-1} & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 \end{bmatrix}.$$

$$\begin{aligned} \|\mathbf{Ax}\|_2^2 &= \|\mathbf{U}\Sigma\mathbf{V}^T\mathbf{x}\|_2^2 = \|\Sigma\mathbf{V}^T\mathbf{x}\|_2^2 \\ &= \sigma_1^2 |\mathbf{v}_1^T \mathbf{x}|^2 + \sigma_2^2 |\mathbf{v}_2^T \mathbf{x}|^2 + \dots + \sigma_r^2 |\mathbf{v}_r^T \mathbf{x}|^2. \end{aligned}$$

$$\|\mathbf{A}\|_2 = \max_{\|\mathbf{x}\|_2=1} \|\mathbf{Ax}\|_2 = \max_{\|\mathbf{x}\|_2=1} \|\Sigma\mathbf{V}^T\mathbf{x}\|_2 = \max_{\|\mathbf{x}\|_2=1} \|\Sigma\mathbf{y}\|_2.$$

$$\begin{aligned} \|\Sigma\mathbf{y}\|_2^2 &= \sigma_1^2 |y_1|^2 + \dots + \sigma_r^2 |y_r|^2 \\ &= \sigma_1^2 [|y_1|^2 + \frac{\sigma_2^2}{\sigma_1^2} |y_2|^2 + \dots + \frac{\sigma_r^2}{\sigma_1^2} |y_r|^2] \\ &\leq \sigma_1^2 \|\mathbf{y}\|_2^2. \end{aligned}$$

numerischer Rang der Matrix

$$\sigma_1 \geq \sigma_2 \dots \geq \sigma_k \gg \sigma_{k+1} \geq \dots \geq \sigma_r > 0.$$

$$\|A\|_2 = \max_{x \in \mathbb{R}^n} \frac{\|Ax\|_2}{\|x\|_2} = \sigma_1 \text{ erreicht f\u00fcr } x = V \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = v_1.$$

Definition 7.3.0.11. Konditionszahl

Die Konditionszahl einer invertierbaren Matrix **A** ist

$$k(A) = \frac{\|A\|_2}{\|A^{-1}\|_2} = \sqrt{\frac{d_{\max}}{d_{\min}}}, = \frac{\Delta_1}{\Delta_r} \quad !!$$

wobei d_{\max} das gr\u00f6sste und d_{\min} das kleinste Eigenwert von $A^H A$ sind.

Satz 8.2.6.2. Beste Approximation an einer Matrix: Schmidt 1907 wiederentdeckt von Eckart 1933 und Young 1936

Sei $A \in \mathbb{R}^{m \times n}$ beliebig. F\u00fcr jedes $k \leq \text{Rang } A$ gibt die abgebrochene Singul\u00e4rwertzerlegung

$$A_k = \sum_{j=1}^k \sigma_j u_j v_j^T$$

die beste Approximation von Rang k an A im Sinne von:

$$\|A - A_k\| = \min_{\substack{B \in \mathbb{R}^{m \times n} \\ \text{Rang } B \leq k}} \|A - B\| = \sigma_{k+1}.$$

Dabei kann $\|\cdot\|$ die euklidische, die Frobenius oder Nuklearnorm der Matrix sein.

$$\max_{\substack{x \in \mathbb{R}^n \\ x \notin \text{span}\{v_1\}}} \frac{\|Ax\|_2}{\|x\|_2} = \sigma_2 \text{ erreicht f\u00fcr } x = V \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = v_2.$$

$$\max_{\substack{x \in \mathbb{R}^n \\ x \notin \text{span}\{v_1, v_2\}}} \frac{\|Ax\|_2}{\|x\|_2} = \sigma_3 \text{ erreicht f\u00fcr } x = V \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} = v_3,$$

Definition 8.2.3.2. Wichtige Matrizenormen

Sei A eine $m \times n$ Matrix.
Die *euklidische Norm* oder (wie gerade bewiesen) auch *Spektralnorm* ist

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sigma_1.$$

Die *Frobenius Norm* ist

$$\|A\|_F = \sqrt{\sum_{i,j} |A_{i,j}|^2} = \text{Spur}(A^T A) = \sqrt{\sigma_1^2 + \sigma_2^2 + \dots + \sigma_r^2}.$$

Die *Nuklearnorm* ist

$$\|A\|_N = \sigma_1 + \sigma_2 + \dots + \sigma_r.$$

Satz 8.2.3.3. Nützliche Ungleichungen zwischen Matrixnormen

Für $\mathbf{A} \in \mathbb{R}^{m \times n}$ von Rang r gelten die Ungleichungen:

$$\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{r} \|\mathbf{A}\|_2$$

$$\|\mathbf{A}\|_F \leq \|\mathbf{A}\|_N \leq \sqrt{r} \|\mathbf{A}\|_F$$

$$\|\mathbf{A}\|_{\max} \leq \|\mathbf{A}\|_2 \leq \sqrt{mn} \|\mathbf{A}\|_{\max}$$

$$\frac{1}{\sqrt{n}} \|\mathbf{A}\|_{\infty} \leq \|\mathbf{A}\|_2 \leq \sqrt{m} \|\mathbf{A}\|_{\infty}$$

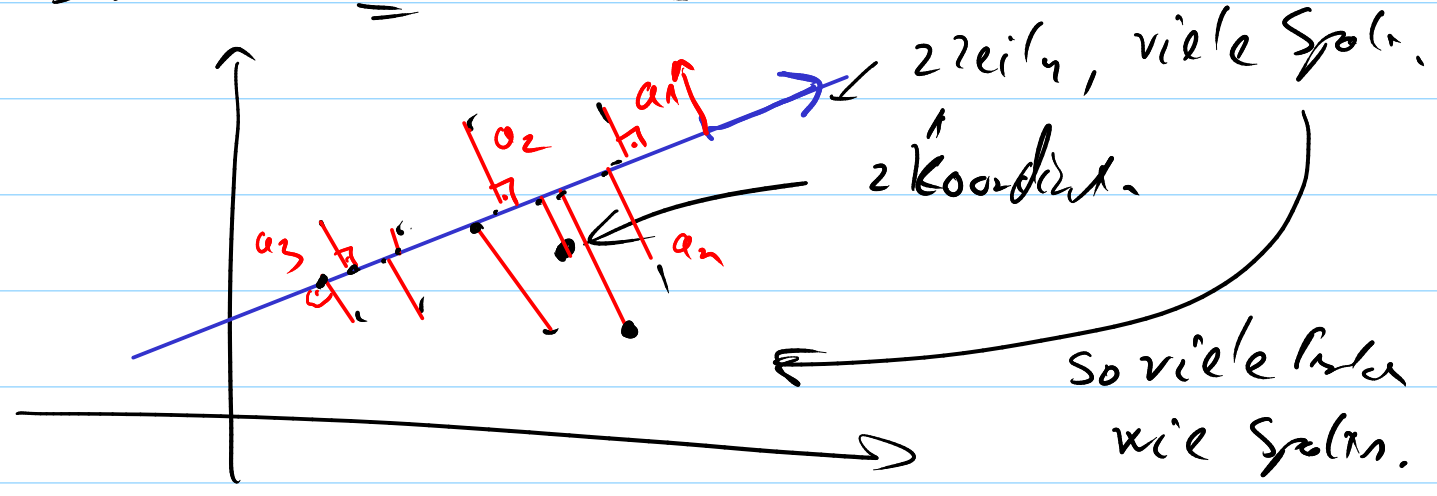
$$\frac{1}{\sqrt{m}} \|\mathbf{A}\|_1 \leq \|\mathbf{A}\|_2 \leq \sqrt{n} \|\mathbf{A}\|_1,$$

und ausserdem:

$$\|\mathbf{A}\|_2 \leq \sqrt{\|\mathbf{A}\|_1 \|\mathbf{A}\|_{\infty}}.$$

8.3 Hauptkomponentenanalyse PCA = principal component analysis

Data A Matrix



$$\min \{ a_1^2 + a_2^2 + \dots + a_n^2 \}$$

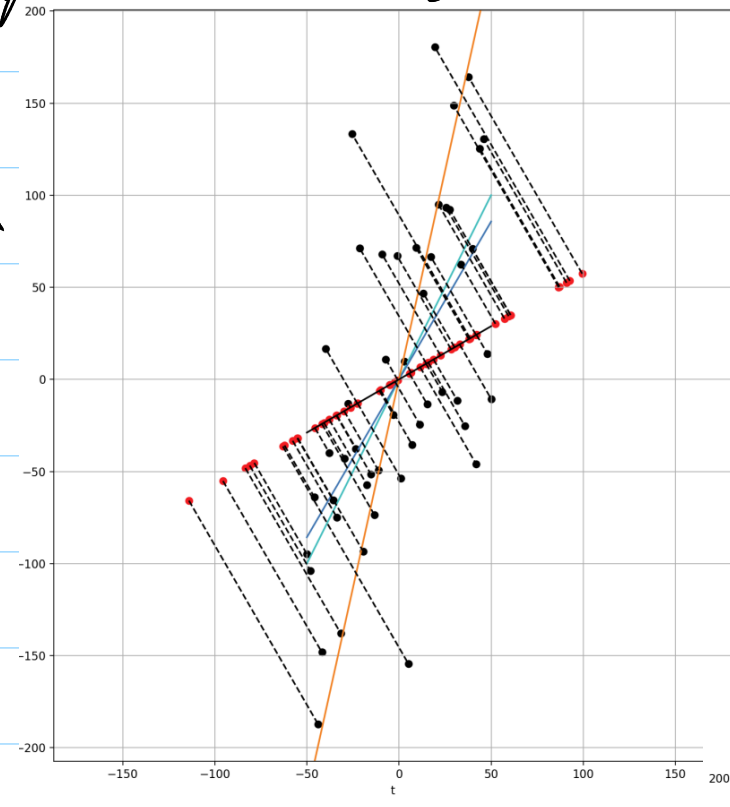
LA: Welche Spalten sind linear unabhängig.

⇒ Hauptkomponentenanalyse

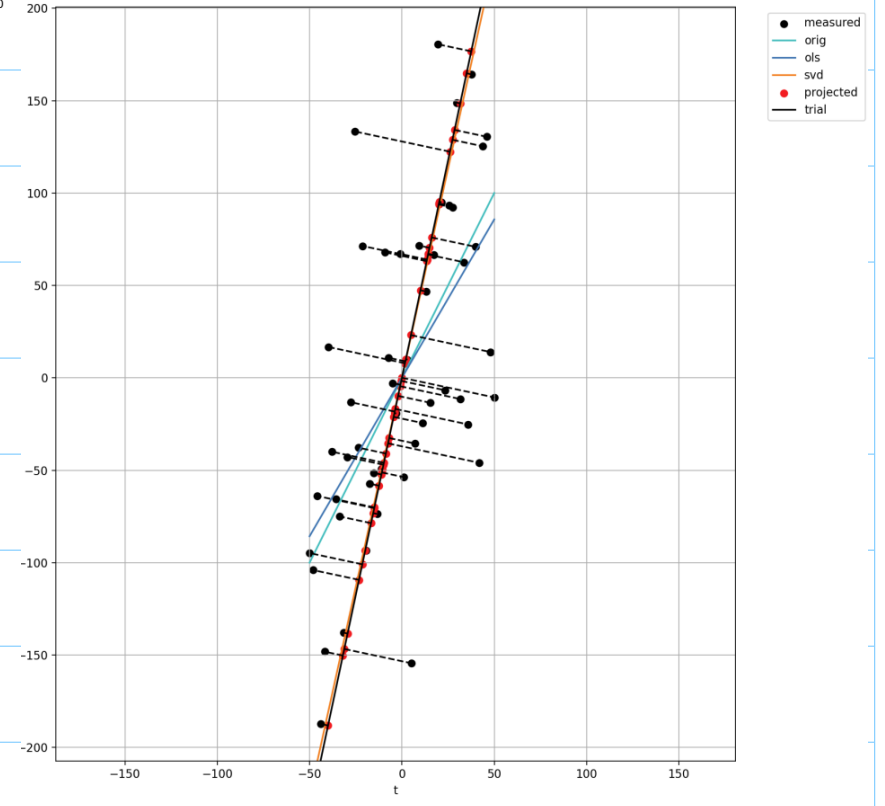
$$\underline{A} = \underline{U} \underline{\Sigma} \underline{V}^T \approx \sum_{k=1}^r \left[\frac{u_k}{\sigma_k} \right] \sigma_k \underline{v}_k^T$$

Hauptkomponenten

Gewichte



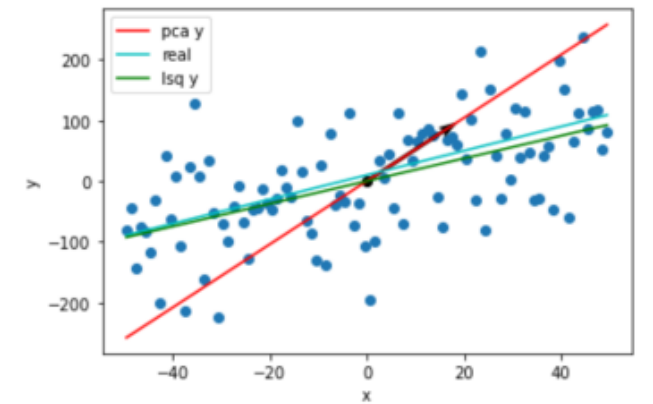
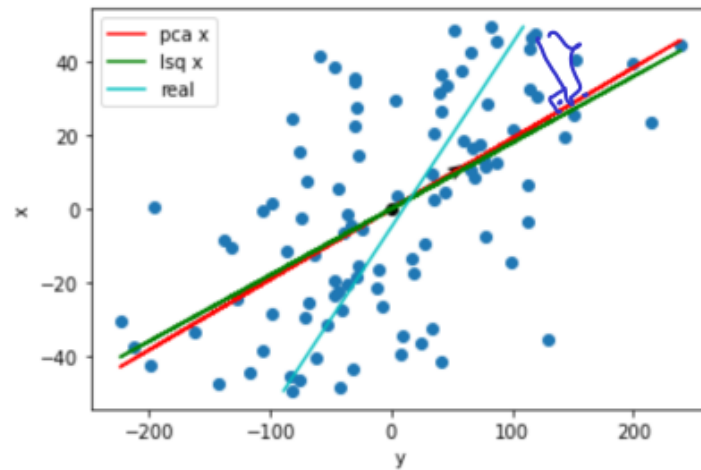
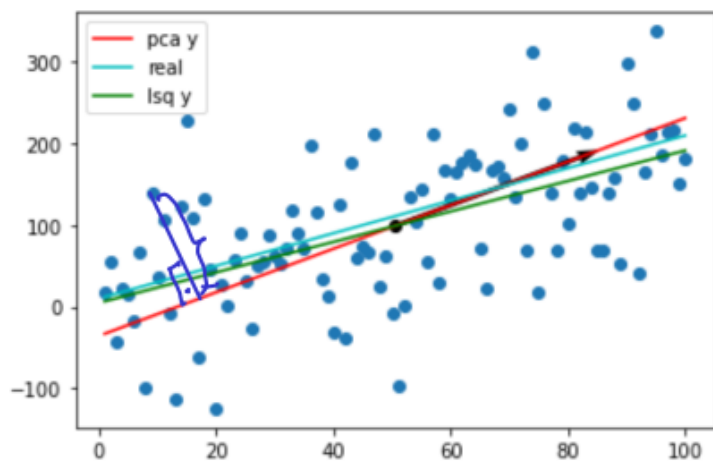
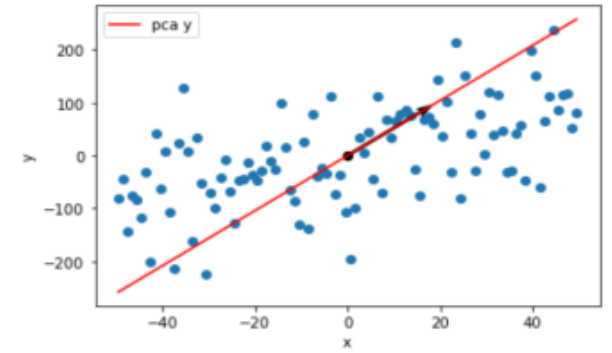
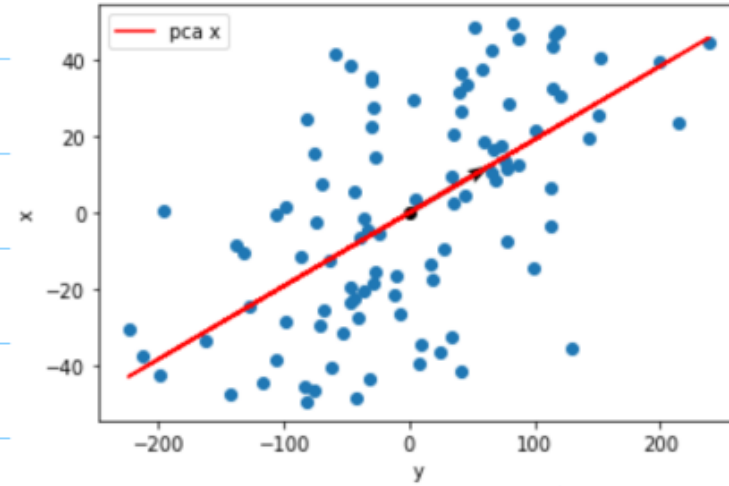
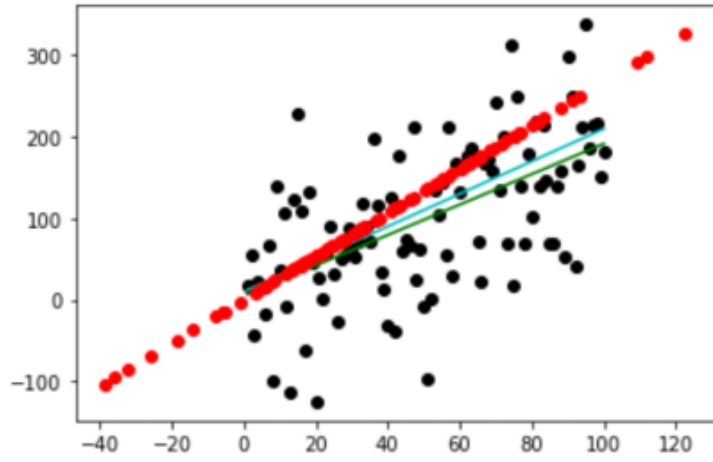
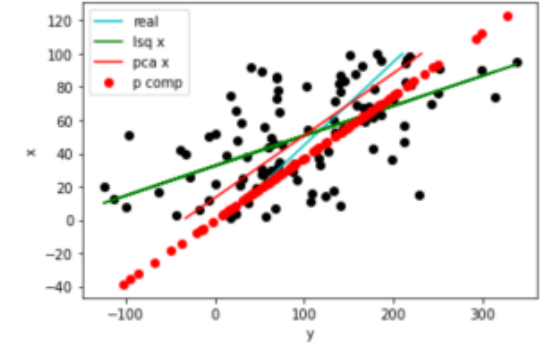
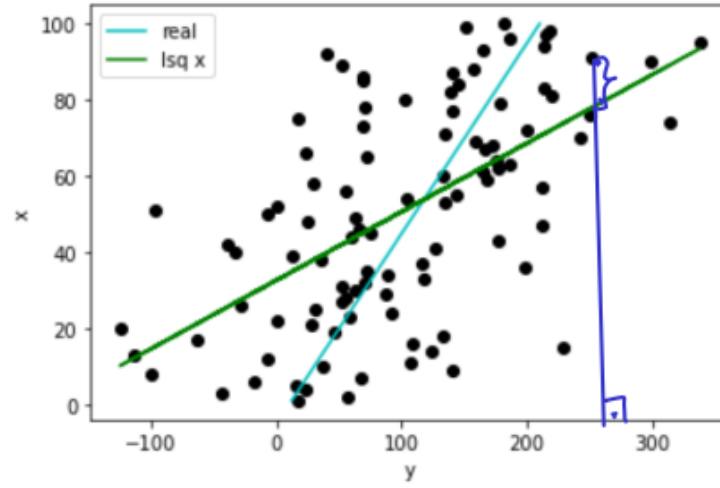
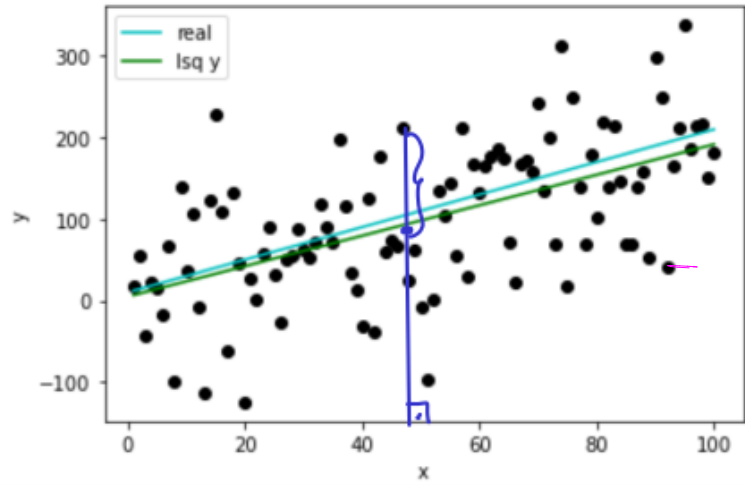
OLS = LSQ




Für ein lineares Model:


$$y = ax + c ; \quad c=10, a=2 \\ x \in [0, 100]$$

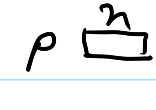
$$y_b = y + \text{noise} \dots$$




Q.4. Lineare Ausgleichsrechnung mit linearen Nebenbedingungen

$\underline{A} \in \mathbb{R}^{m \times n}$, $m \geq n = \text{Rang } \underline{A}$ 

$\underline{b} \in \mathbb{R}^m$ 

$\underline{C} \in \mathbb{R}^{p \times n}$, $n \geq p = \text{Rang } \underline{C}$ 

$\underline{d} \in \text{Bild } \underline{C} \subset \mathbb{R}^p$ 

$\min_{\underline{x} \in \mathbb{R}^n} \|\underline{A}\underline{x} - \underline{b}\|^2$

$\underline{C}\underline{x} = \underline{d}$ Nebenbedingung

Methoden: Lagrange Multiplikatoren $\underline{m} \in \mathbb{R}^p$

$L(\underline{x}, \underline{m}) = \frac{1}{2} \|\underline{A}\underline{x} - \underline{b}\|_2^2 + \underline{m}^T (\underline{C}\underline{x} - \underline{d})$

Löse

$\min_{\underline{x} \in \mathbb{R}^n} \max_{\underline{m} \in \mathbb{R}^p} L(\underline{x}, \underline{m})$

Im Sattelpunkt:

$\begin{cases} \frac{\partial L}{\partial \underline{x}}(\underline{x}, \underline{m}) = 0 \\ \frac{\partial L}{\partial \underline{m}}(\underline{x}, \underline{m}) = 0 \end{cases}$

$\begin{cases} \underline{A}^T(\underline{A}\underline{x} - \underline{b}) + \underline{C}^T \underline{m} = 0 \\ \underline{C}\underline{x} - \underline{d} = 0 \end{cases}$

$\begin{bmatrix} \underline{A}^T \underline{A} & \underline{C}^T \\ \underline{C} & \underline{0} \end{bmatrix} \begin{bmatrix} \underline{x} \\ \underline{m} \end{bmatrix} = \begin{bmatrix} \underline{A}^T \underline{b} \\ \underline{d} \end{bmatrix}$

Wie lösen? Block-LR-Zerlegung.

1) Cholesky-Zerlegung: $\underline{A}^T \underline{A} = \underline{R}^T \underline{R}$

2) berechne \underline{G} aus $\underline{R}^T \underline{G}^T = \underline{C}^T$

3) berechne \underline{S} aus Cholesky-Zerlegung

$$\underline{S}^T \underline{S} = -\underline{G} \underline{G}^T$$

$$\begin{bmatrix} \underline{A}^T \underline{A} & -\underline{C}^T \\ \underline{C} & \underline{0} \end{bmatrix} = \begin{bmatrix} \underline{R}^T & \underline{0} \\ \underline{G} & \underline{S} \end{bmatrix} \begin{bmatrix} \underline{R} & \underline{0} \\ \underline{0} & \underline{S}^T \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}^{\text{Not.}} = \begin{bmatrix} \underline{R} & \underline{G}^T \\ \underline{0} & \underline{S}^T \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\begin{bmatrix} \underline{R}^T & \underline{0} \\ \underline{G} & \underline{S} \end{bmatrix} \begin{bmatrix} \underline{y}_1 \\ \underline{y}_2 \end{bmatrix} = \begin{bmatrix} \underline{A}^T \underline{b} \\ \underline{d} \end{bmatrix}$$

$$\Rightarrow \underline{R}^T \underline{y}_1 = \underline{A}^T \underline{b} \rightarrow \underline{y}_1 = (\underline{R}^T)^{-1} \underline{A}^T \underline{b}$$

$$\underline{y}_1 = \text{solve}(\underline{R}^T, \underline{A}^T \underline{b})$$

$$\underline{G} \underline{y}_1 + \underline{S} \underline{y}_2 = \underline{d} \Rightarrow \underline{S} \underline{y}_2 = \underline{d} - \underline{G} \underline{y}_1$$

obere Dreiecksmatrix.

$$\underline{y}_2 = \text{solve}(\underline{S}, \underline{d} - \underline{G} \underline{y}_1)$$

$$\begin{bmatrix} \underline{R} & \underline{G}^T \\ \underline{0} & \underline{S}^T \end{bmatrix} \begin{bmatrix} \underline{x} \\ \underline{z} \end{bmatrix} = \begin{bmatrix} \underline{y}_1 \\ \underline{y}_2 \end{bmatrix}$$

$$\underline{S}^T \underline{z} = \underline{y}_2 \rightarrow \underline{z} = \text{solve}(\underline{S}^T, \underline{y}_2)$$

$$\underline{R} \underline{x} = \underline{y}_1 - \underline{G}^T \underline{z} \rightarrow \underline{x} = \text{solve}(\underline{R}, \underline{y}_1 - \underline{G}^T \underline{z})$$

Methode 2: Singularwertzerlegung von \underline{C} :

Vorteil: stabiler

Nachteil: Struktur von \underline{A} kann nicht verwendet werden!

$$p \uparrow \underline{C} = \underline{U} \begin{bmatrix} \underline{\Sigma} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \underline{V}_1^T \\ \underline{V}_2^T \end{bmatrix}$$

$$\text{Rang } \underline{C} = p$$

$$\begin{bmatrix} \sigma_1 & \sigma_2 & \dots & \sigma_p & & \mathbf{0} \end{bmatrix}$$

$$\text{Ker } \underline{C} = \text{Bild } \{ \underline{V}_2 \}$$

$$\underline{x}_0 = \underline{V}_1 \underline{\Sigma}^{-1} \underline{U}^T \underline{d}$$

Suche:

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

$$\underline{C} \underline{x} = \underline{C} \underline{x}_0 + \underline{C} (\underline{V}_2 \underline{y})$$

$$= \underline{U} \underline{\Sigma} \begin{bmatrix} \underline{V}_1^T \underline{V}_1 \end{bmatrix} \underline{\Sigma}^{-1} \underline{U}^T \underline{d} = \underline{U} \underline{U}^T \underline{U} \underline{d} = \underline{d} \Rightarrow \text{Nebenbedingung ist von diesem } \underline{x} \text{ exakt erf\u00fcllt.}$$

$$\|\underline{Ax} - \underline{b}\|_2^2 = \|\underline{Ax}_0 + \underline{AV}_2 \underline{y} - \underline{b}\|_2^2 = \|\underline{AV}_2 \underline{y} - (\underline{b} - \underline{Ax}_0)\|_2^2$$

↑
frei nur fest

Das ist ein standard Ausgleichsproblem mit Matrix \underline{AV}_2 und rechte Seite $\underline{b} - \underline{Ax}_0$

8.5 Nichtlineare Ausgleichsrechnung

Modell: $f(t, \underline{x}) = y$

$\underline{x} \in \mathbb{R}^n$ Parameter, aus Messungen zu bestimmen

Messungen: $f(t_i, \underline{x}) = y_i$ für $i=1, 2, \dots, m$

Residuum:

$$F(\mathbf{x}) = \begin{bmatrix} f(t_1, \mathbf{x}) - y_1 \\ \vdots \\ f(t_m, \mathbf{x}) - y_m \end{bmatrix} \quad F: \mathbb{R}^n \rightarrow \mathbb{R}^m$$

Aufgabe (nichtlineare Ausgleichsrechnung): Finde $\mathbf{x} \in \mathbb{R}^n$, so dass

$$\mathbf{x} = \operatorname{argmin}_{\mathbf{u} \in \mathbb{R}^n} \left(\sum_{i=1}^m |f(t_i, \mathbf{u}) - y_i|^2 \right) = \operatorname{argmin}_{\mathbf{u} \in \mathbb{R}^n} \underbrace{\frac{1}{2} \|F(\mathbf{u})\|_2^2}_{\Phi(\underline{u})}$$

Für \underline{x}^* = Lösung von argmin :

$$\operatorname{grad} \Phi(\underline{x}^*) = 0$$

$$\Phi(\underline{x}) = \frac{1}{2} \|F(\underline{x})\|_2^2$$

$$\operatorname{grad} \Phi(\mathbf{x}) = (\underline{DF}(\mathbf{x}))^T \underline{F}(\mathbf{x})$$

$$D(\operatorname{grad} \Phi)(\mathbf{x}) = (\underline{DF}(\mathbf{x}))^T \underline{DF}(\mathbf{x}) + \sum_{j=1}^m F_j(\mathbf{x}) \underline{D^2 F_j}(\mathbf{x})$$

$$\begin{matrix} \parallel \\ \parallel \\ H_{\Phi}(\underline{x}) \end{matrix}$$

Hesse-Matrix

Newton:

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} - \underbrace{(D \operatorname{grad} \Phi(\mathbf{x}))^{-1}} \operatorname{grad} \Phi(\mathbf{x})$$

oder:

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} - \underline{\Delta}$$

mit Newton-Korrektur aus dem LGS

$$H_{\Phi}(\underline{x}^{(k)}) \underline{\Delta} = \underline{DF}(\underline{x}^{(k)})^T \underline{F}(\underline{x}^{(k)})$$

Hesse Matrix.

Alternative: Gauss-Newton-Verfahren

Idee: linearisiere F um y .

$$\begin{aligned}
 F(\mathbf{x}) &\approx F(\mathbf{y}) + DF(\mathbf{y})(\mathbf{x} - \mathbf{y}) = F(\mathbf{y}) + \mathbf{J}_F(\mathbf{y})(\mathbf{x} - \mathbf{y}) \\
 &= F(\mathbf{y}) + \mathbf{J}_F(\mathbf{y})\mathbf{x} - \mathbf{J}_F(\mathbf{y})\mathbf{y}
 \end{aligned}$$

lineare Version von F

Standard LSQ !

$$\operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \|F(\mathbf{x})\|_2^2 \approx \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \|F(\mathbf{y}) + \mathbf{J}_F(\mathbf{y})\mathbf{x} - \mathbf{J}_F(\mathbf{y})\mathbf{y}\|_2^2 = \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

Setzt man $\mathbf{x} = \mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{s}$ und $\mathbf{y} = \mathbf{x}^k$, dann erhält man

erste Ableitung von F

$$\begin{aligned}
 \operatorname{argmin}_{\mathbf{z} \in \mathbb{R}^n} \frac{1}{2} \|F(\mathbf{x}^k - \mathbf{z})\| &\approx \operatorname{argmin}_{\mathbf{z} \in \mathbb{R}^n} \frac{1}{2} \|F(\mathbf{x}^k) + \mathbf{J}_F(\mathbf{x}^k)(\mathbf{x}^k - \mathbf{z} - \mathbf{x}^k)\| \\
 &= \operatorname{argmin}_{\mathbf{z} \in \mathbb{R}^n} \frac{1}{2} \|F(\mathbf{x}^k) - \mathbf{J}_F(\mathbf{x}^k)\mathbf{z}\|
 \end{aligned}$$

+ Code

+ Bilder im Skript

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{s} \quad \text{mit} \quad \mathbf{s} := \operatorname{argmin}_{\mathbf{z} \in \mathbb{R}^n} \|F(\mathbf{x}^{(k)}) - \mathbf{J}_F(\mathbf{x}^{(k)})\mathbf{z}\|_2^2.$$

Bsp Modell $f(t, \underline{x}) = x_1 + x_2 e^{-x_3 t}$

$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^3$ $\underline{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_m \end{bmatrix} \in \mathbb{R}^m$

Messungen $t_1, \dots, t_m \in \mathbb{R}$ $\underline{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m$

$y_1, \dots, y_m \in \mathbb{R}$

Residuen: $\underline{F}(t, \underline{x}) = f(\underline{t}, \underline{x}) - \underline{y} \in \mathbb{R}^m$

$\left[\begin{array}{l} f(t_j, \underline{x}) - y_j \\ \vdots \end{array} \right]_{j=1,2,\dots,m}$

$\underline{D}F(\underline{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(t_j, \underline{x}) \\ \vdots \\ \frac{\partial}{\partial x_3} f(t_j, \underline{x}) \end{bmatrix}$

$i=1,2,3$ [Index der Parameter]
 $j=1,2,\dots,m$ [Indice der Messungen]

$\frac{\partial f}{\partial x_1} = 1$ $\frac{\partial f}{\partial x_2} = e^{-x_3 t}$ $\frac{\partial f}{\partial x_3} = -t x_2 e^{-x_3 t}$

$\underline{D}F(\underline{x}) = \begin{bmatrix} 1 \\ e^{-x_3 t_1} \\ \vdots \\ e^{-x_3 t_m} \end{bmatrix} \in \mathbb{R}^{m \times 3}$

$\underline{f} \in \mathbb{R}^m$ $\underline{f} = F(\underline{t}, \underline{x})$ $d\underline{f} = \underline{D}F(\underline{x}) \in \mathbb{R}^{m \times 3}$

2. Ableitung:

$$\left[\begin{array}{ccc} \frac{\partial^2 f}{\partial x_1^2} = 0 & \frac{\partial^2 f}{\partial x_2 \partial x_1} = 0 & \frac{\partial^2 f}{\partial x_3 \partial x_1} = 0 \\ 0 & \frac{\partial^2 f}{\partial x_2^2} = 0 & \frac{\partial^2 f}{\partial x_3 \partial x_2} = -t e^{-x_3 t} \\ 0 & t e^{-x_3 t} & \frac{\partial^2 f}{\partial x_2^2} = t^2 x_2 e^{-x_3 t} \end{array} \right]$$

↪ Punktweise in \underline{t} auswerten!

$\underline{t}_{\text{ap}} = \underline{t} e^{-x_3 \underline{t}} \in \mathbb{R}^m$ Punktweise Auswertung

$\sum_{j=1}^m F_j(\underline{x}) \underline{D}^2 F_j(\underline{x})$

Damit: Newton, Gauss-Newton, ...

GPS: N Satelliten: für $i=1, \dots, N$ bekannt $[t_{s,i}, x_{s,i}, y_{s,i}, z_{s,i}]$

Gesucht: $\underline{x} = [t_r, x_r, y_r, z_r]$ Zeit + Position Receiver

↑ ungenau im Receiver, darum mit z bestimmen

Laufzeit messung \Rightarrow

Residuum: $F: \mathbb{R}^4 \rightarrow \mathbb{R}^N$

$c =$ Lichtgeschwindigkeit

$$F(\underline{x}) = \left[\begin{array}{c} -c^2(t_r - t_{s,i})^2 + (x_r - x_{s,i})^2 + (y_r - y_{s,i})^2 + (z_r - z_{s,i})^2 \\ \vdots \\ \vdots \end{array} \right]_{\hat{c}=1,2,\dots,N}$$

$$\Rightarrow \min_{\underline{x} \in \mathbb{R}^4} \|F(\underline{x})\|_2^2$$

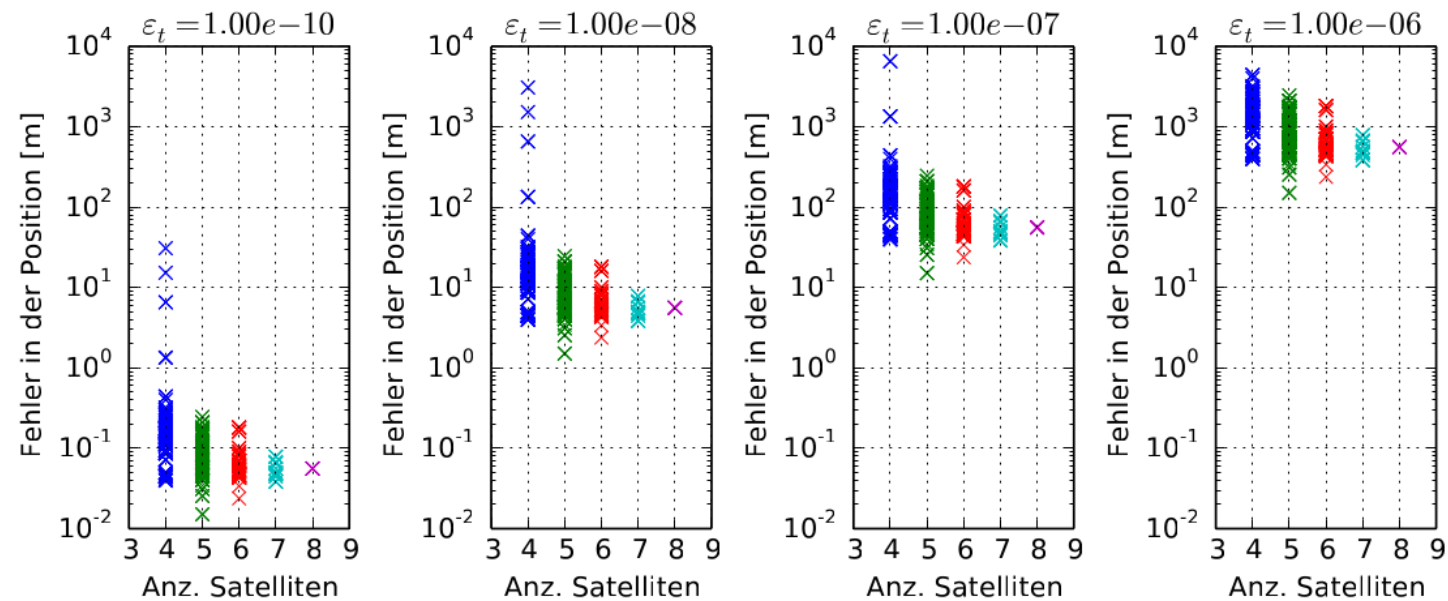


Abbildung 4: Fehler in der Positionsbestimmung in Abhängigkeit der Anz. Satelliten und dem Messfehler in der Signallaufzeit für 1000 Versuche in dem jeder Satellit mit Wahrscheinlichkeit 0.5 ausgewählt wurde. ϵ_t : Standardabweichung im Messfehler der Signallaufzeit.

Ben Viele Methoden zur Minimierung.
Welche ist billig



Gradientenverfahren: Laufe in der Richtung des steilsten Abstiegs.

Hier: $\arg\min_{\underline{x} \in \mathbb{R}^n} \Phi(\underline{x})$ mit

$$\Phi(\underline{x}) = \frac{1}{2} \|F(\underline{x})\|_2^2 \quad \text{mit } F: \mathbb{R}^n \rightarrow \mathbb{R}^m$$

steilsten Abstieg: Richtung des $\text{grad } \Phi(\underline{x}) = \underline{D}F(\underline{x})^T F(\underline{x})$

Schrittweite / Learning rate.

$$\underline{x}_{k+1} = \underline{x}_k - \lambda_k \underbrace{\underline{D}F(\underline{x}^k)^T F(\underline{x}^k)}_{\text{grad } \Phi(\underline{x}^k)}$$

Mit s-Notation:

GD = gradient descent

$$(GD) \quad \left[\begin{array}{l} \underline{x}_{k+1} = \underline{x}_k - \Delta \\ \text{mit } \underline{\Delta} = \lambda_k \underline{D}F(\underline{x}^k)^T F(\underline{x}^k) \end{array} \right]$$

(N) update

$$\underline{x}^{k+1} = \underline{x}^k - \underline{\rho} \quad \text{mit Newton korrekture}$$

$$\underline{D}^2 \phi(\underline{x}^k) \underline{\rho} = \underline{DF}(\underline{x}^k)^T \underline{F}(\underline{x}^k)$$

↳ zu teuer für n, m gross!

(GN) update

$$\underline{x}^{k+1} = \underline{x}^k - \underline{\rho} \quad \text{mit GN-update}$$

(argu. $\frac{1}{2} \| \underline{F}(\underline{x}^k) - \underline{DF}(\underline{x}^k) \underline{\rho} \|_2^2$)

z.B. mit Normalengleich $\underline{b} \quad \underline{A}$

$$\underline{DF}(\underline{x}^k)^T \underline{DF}(\underline{x}^k) \underline{\rho} = \underline{DF}(\underline{x}^k)^T \underline{F}(\underline{x}^k)$$

minimize.: BFGS: Broyden mit spezielle/schnelle Auswertung.

↳ D.

→ code + Bilder GD, CG

/scratch/users/gvasile/LaTeX_doc/Numcourses/NumPhys/Slides/p1_SystemsOfEquations/ch4_IterativeMethodsLSE/PYTHON

/scratch/users/gvasile/LaTeX_doc/Numcourses/NumPhys/Vorlesungen/Prepare
vizGD03.py vizGD02.py

Ben

$$\phi(x) = \frac{1}{2} (x_1^2 + b x_2^2)$$

$$\text{grad } \phi(x) = \begin{bmatrix} x_1 \\ b x_2 \end{bmatrix}; \quad \underline{x}^{(k+1)} = \underline{x}^{(k)} - \lambda \begin{bmatrix} x_1^{(k)} \\ b x_2^{(k)} \end{bmatrix} = \begin{bmatrix} (1-\lambda) x_1^{(k)} \\ (1-\lambda b) x_2^{(k)} \end{bmatrix}$$

$$\Rightarrow \phi(\underline{x}^{(k)}) = \frac{1}{2} (x_1^{(0)} + b(1-b)^{k-1} x_2^{(0)})$$

mit optimal gewähltes λ in jedem Schritt

$$\underline{x}^{(k)} = r^k \begin{bmatrix} (-1)^k b \\ 1 \end{bmatrix} \Rightarrow \text{zick-zack!}$$

→ min. nie erreicht

$0 < b < 1$.
(dazu besser CG)

In Praxis beobachtet man:

GD bewegt sich am Anfang schnell gegen das Min. dann zick-zack!

GN braucht einen guten Startpunkt!

→ Idee: kombiniere die beiden!

$$\Rightarrow \text{LM} : \left[\underline{DF}(\underline{x}^k)^T \underline{DF}(\underline{x}^k) + \lambda \underline{I} \right] \underline{\rho} = \underline{DF}(\underline{x}^k)^T \underline{F}(\underline{x}^k)$$

Wie beschleunigt man GD?
"CG"

Momentum-Methoden ADAM

Idee: kein zick-zack für schweres Ball in engen Tal!

=> füge Trägheit hinzu:
 $(1-\beta)$

$$\underline{x}^{k+1} = \underline{x}^{(k)} - \lambda \underline{z}^{(k)} \quad \text{mit } \underline{z}^k = \text{grad } \phi(\underline{x}^k) + \beta \underline{z}^{(k-1)}$$

λ, β zu wählen, z-Schritt-Methode

Umschreibe in.

$$\begin{cases} \underline{x}^{k+1} = \underline{x}^k - \lambda \underline{z}^k \\ \underline{z}^{k+1} = \text{grad } \phi(\underline{x}^{k+1}) + \beta \underline{z}^k \end{cases}$$

(ähnlich wie bei Velocity-Verlet!)

Kunst: λ, β zu wählen (ADAM, ...)

Bew für $\phi(\underline{x}) = \frac{1}{2} \underline{x}^T \underline{S} \underline{x}$ mit $\underline{S} = \text{spd.}$

=> optimale λ, β findet.

=> viel schnellere Konvergenz!

Bew Es gibt auch andere Arten GD zu beschleunigen.

z.B. Numerov-Verfahren.

Bsp

$$\text{1) } \phi(x) = \frac{1}{2} (x_1^2 + b x_2^2), \quad b = 0.02$$
$$\mathbb{D}\phi, \mathbb{D}^2\phi \text{ exakt, } \underline{x}_0 = \begin{bmatrix} 2b \\ 1 \end{bmatrix} = \begin{bmatrix} 0.04 \\ 1 \end{bmatrix}$$

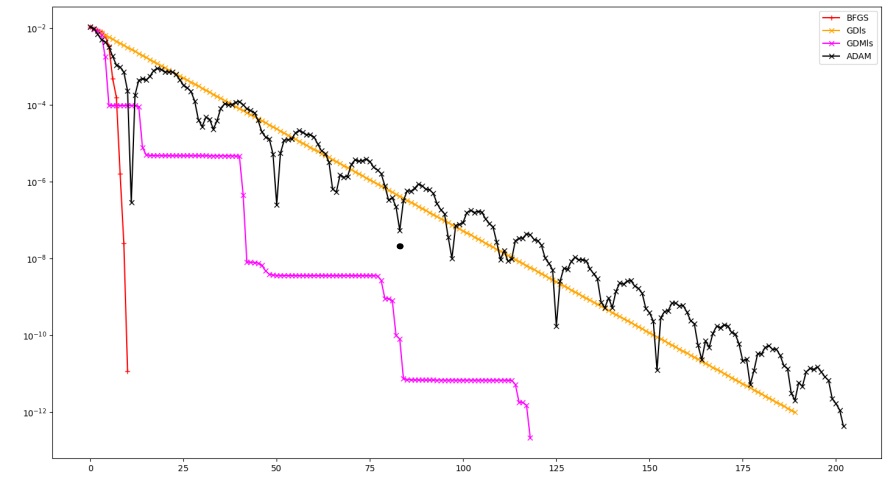
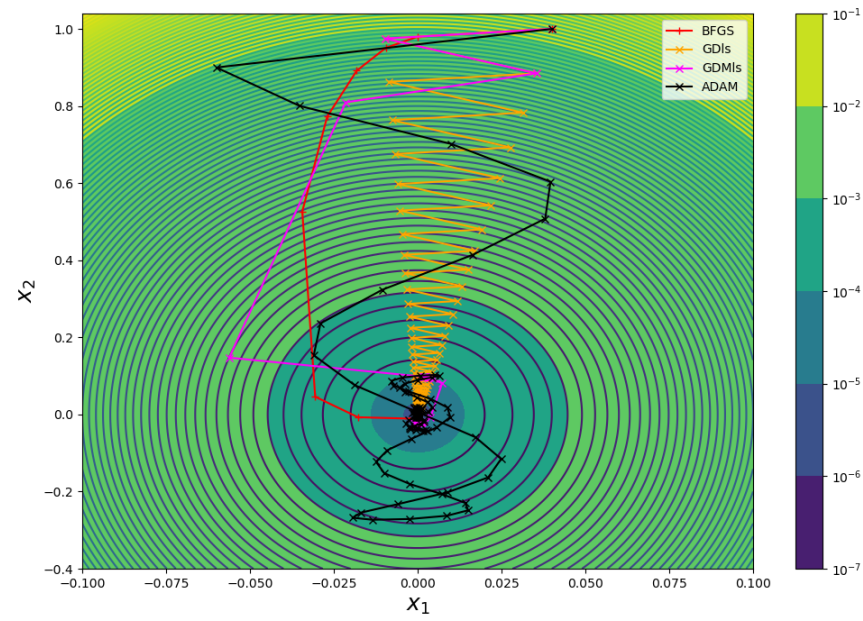
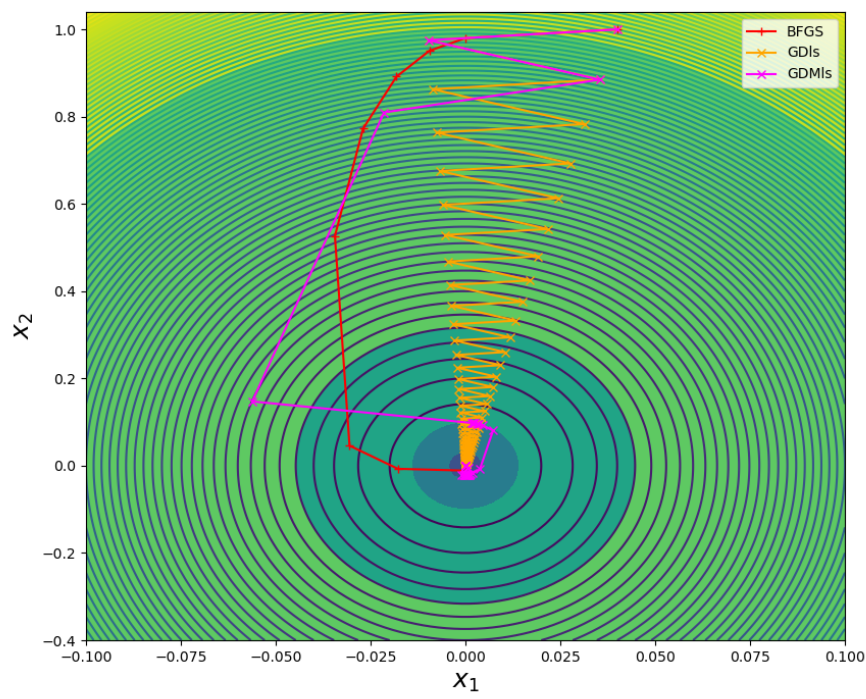
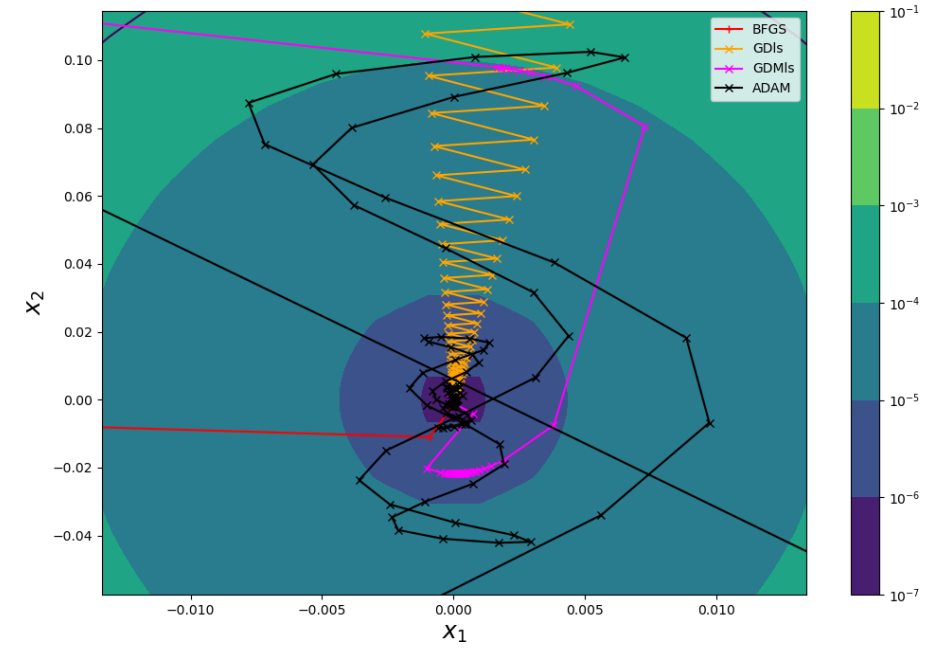
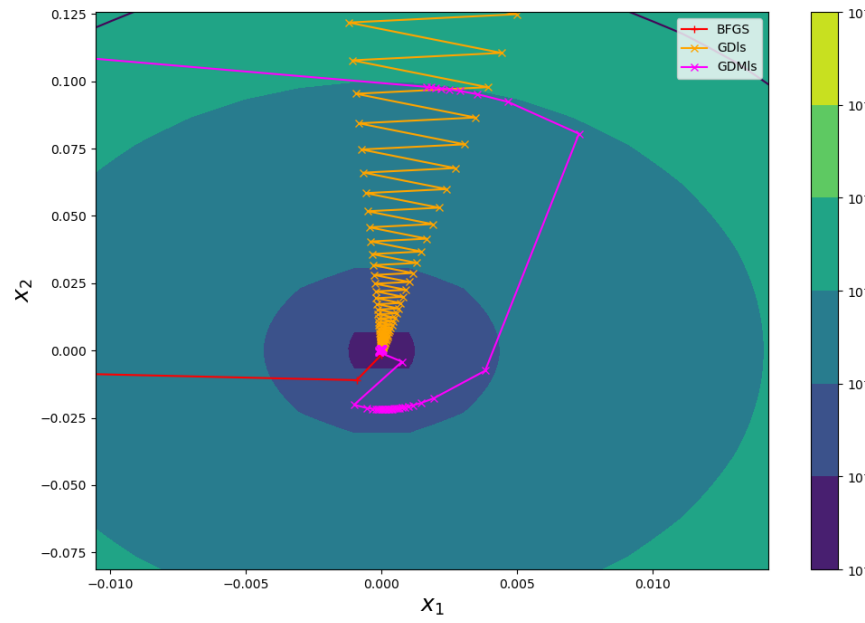
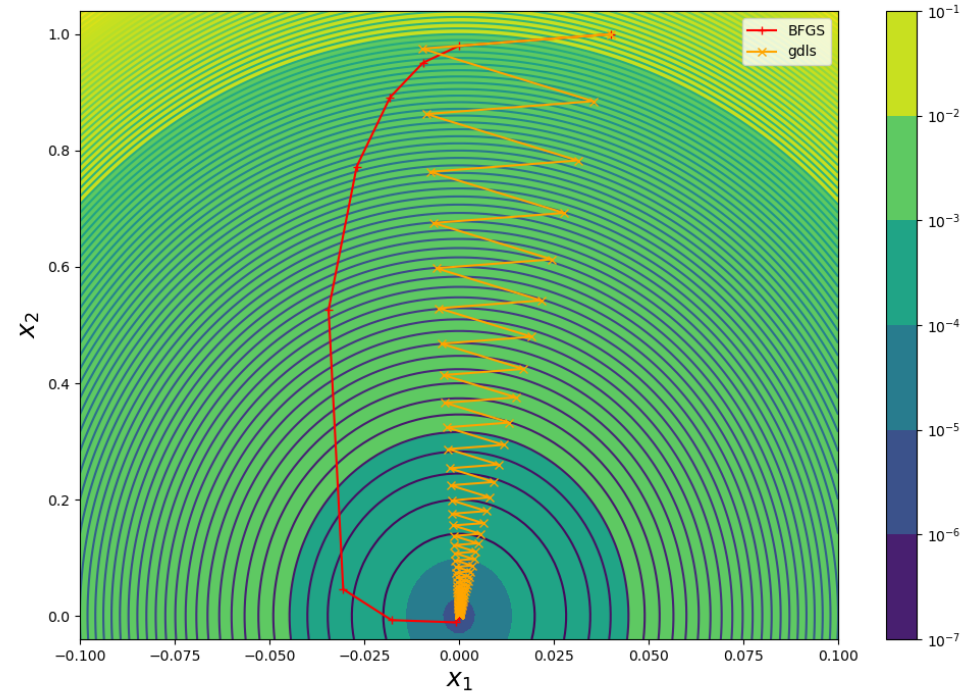
ϕ billig => statt festes Schritt in 1D

nutze minimize für 1D

(nicht praktikabel für ϕ teuer,

z.B. in ML ist ϕ teuer)

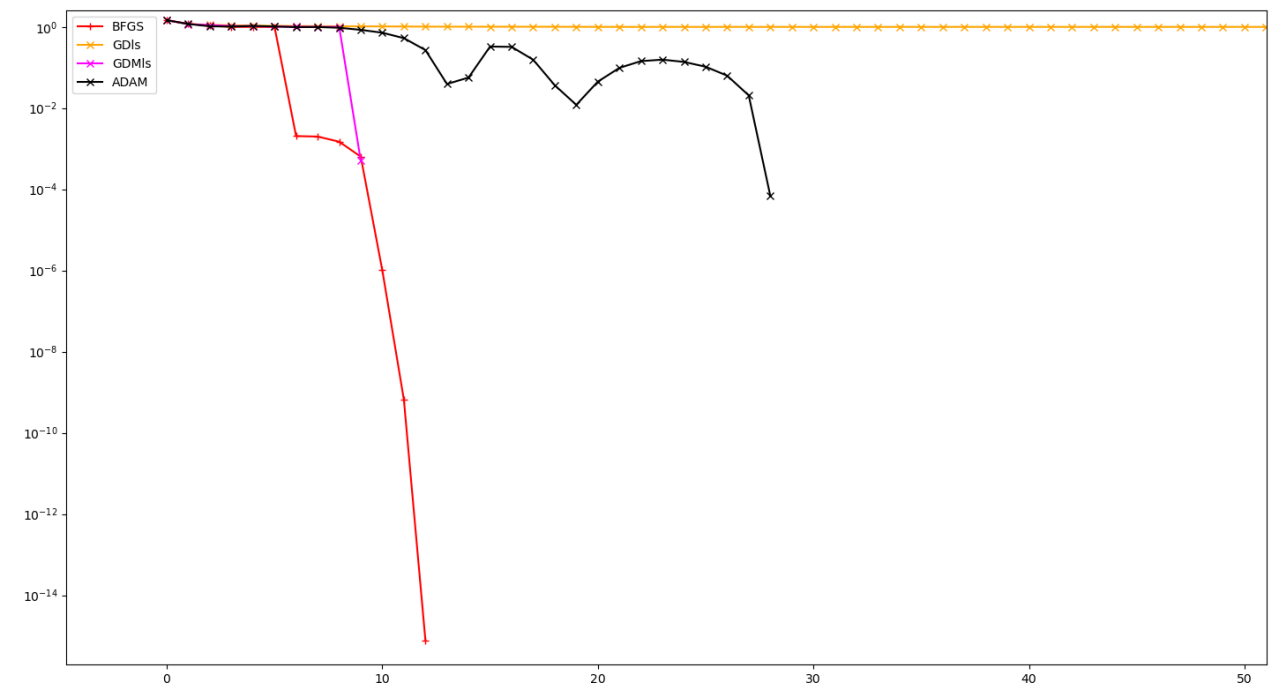
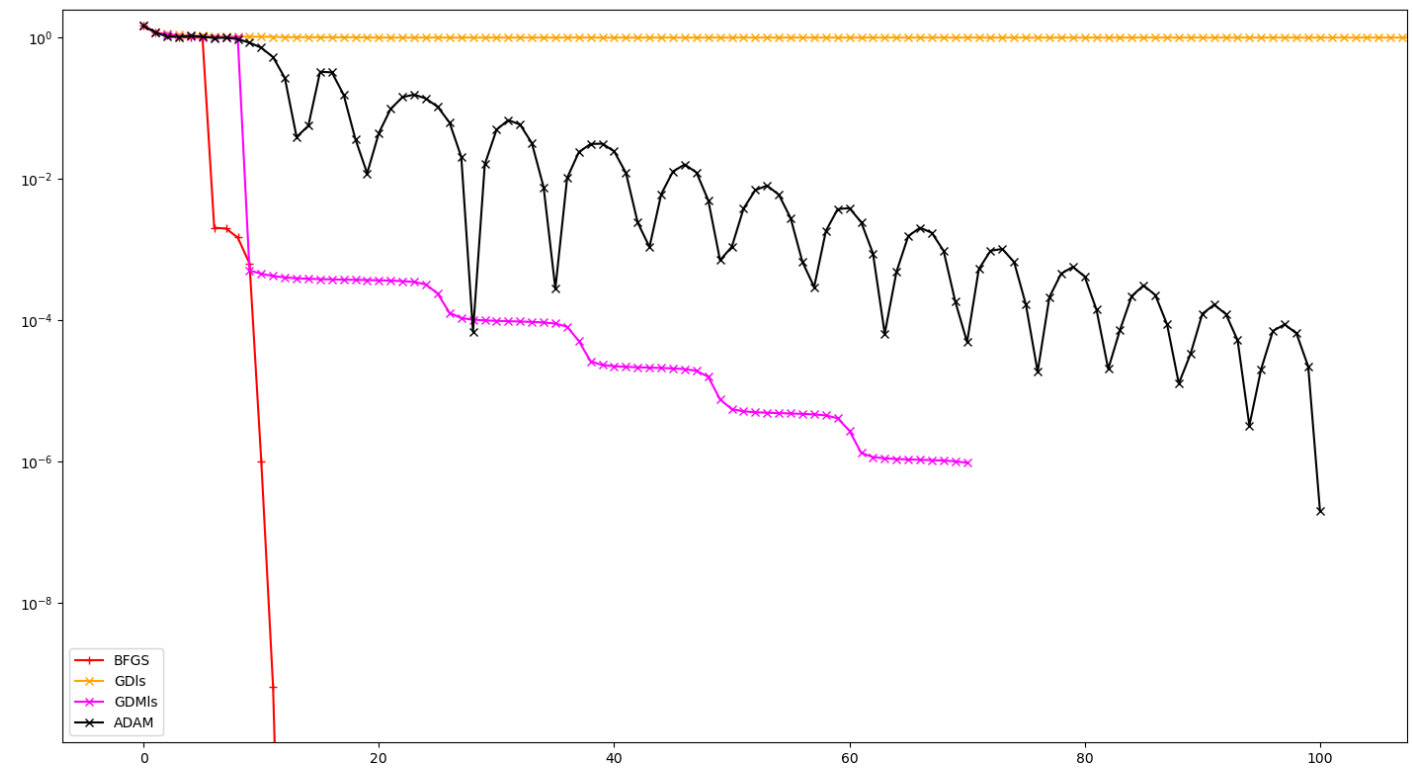
↳ feste Schrittweite in 1D
(learning rate)

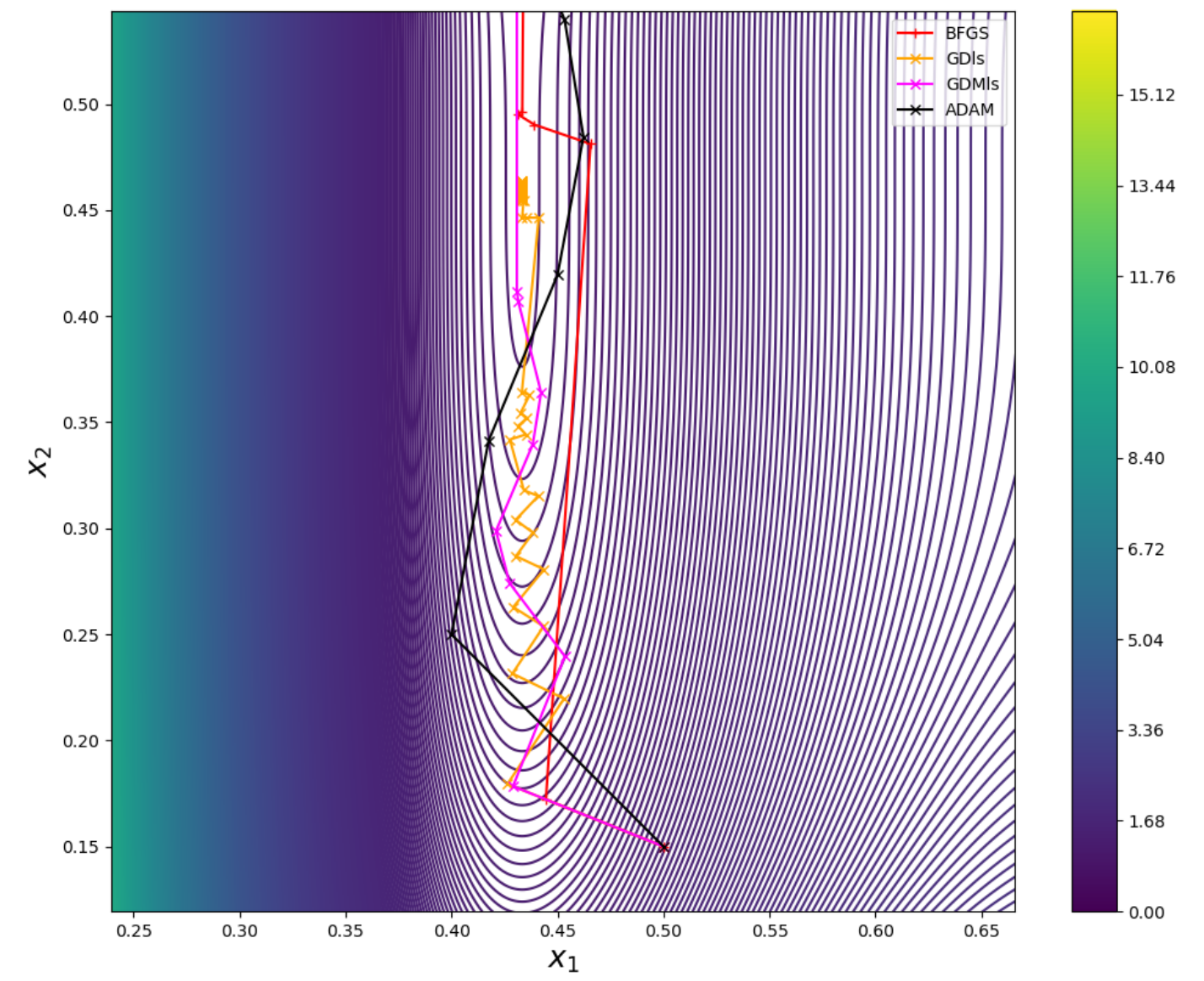
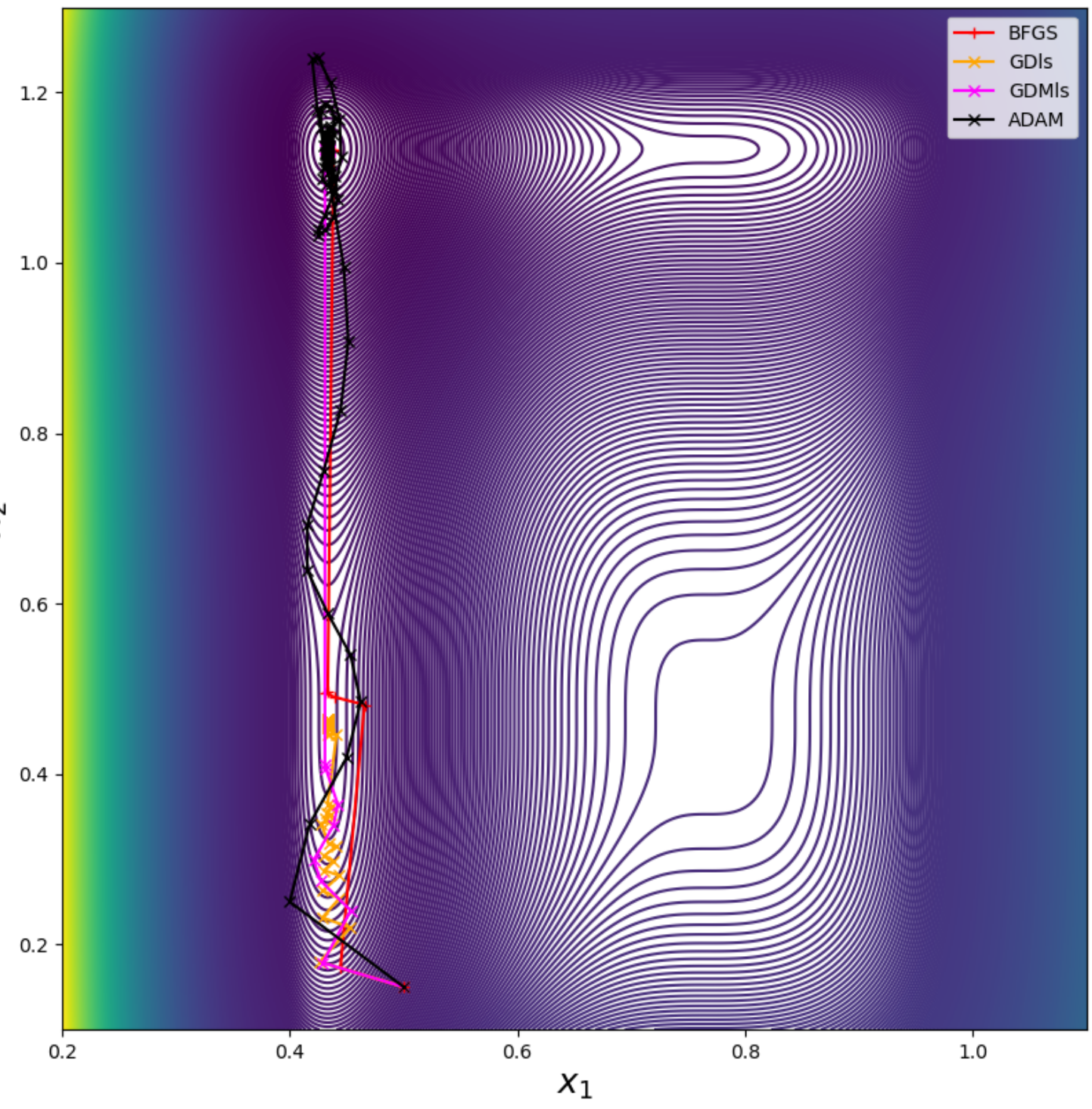


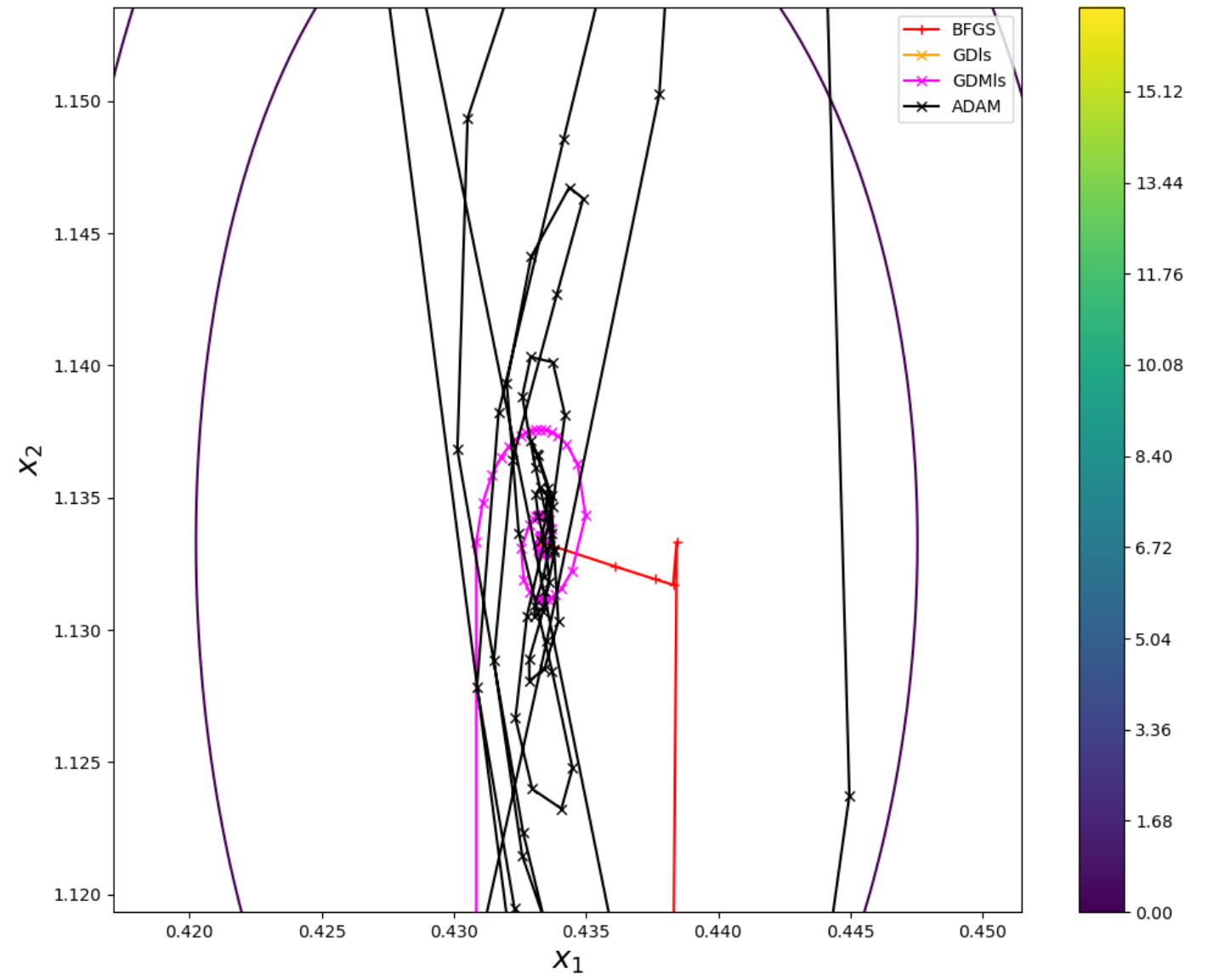
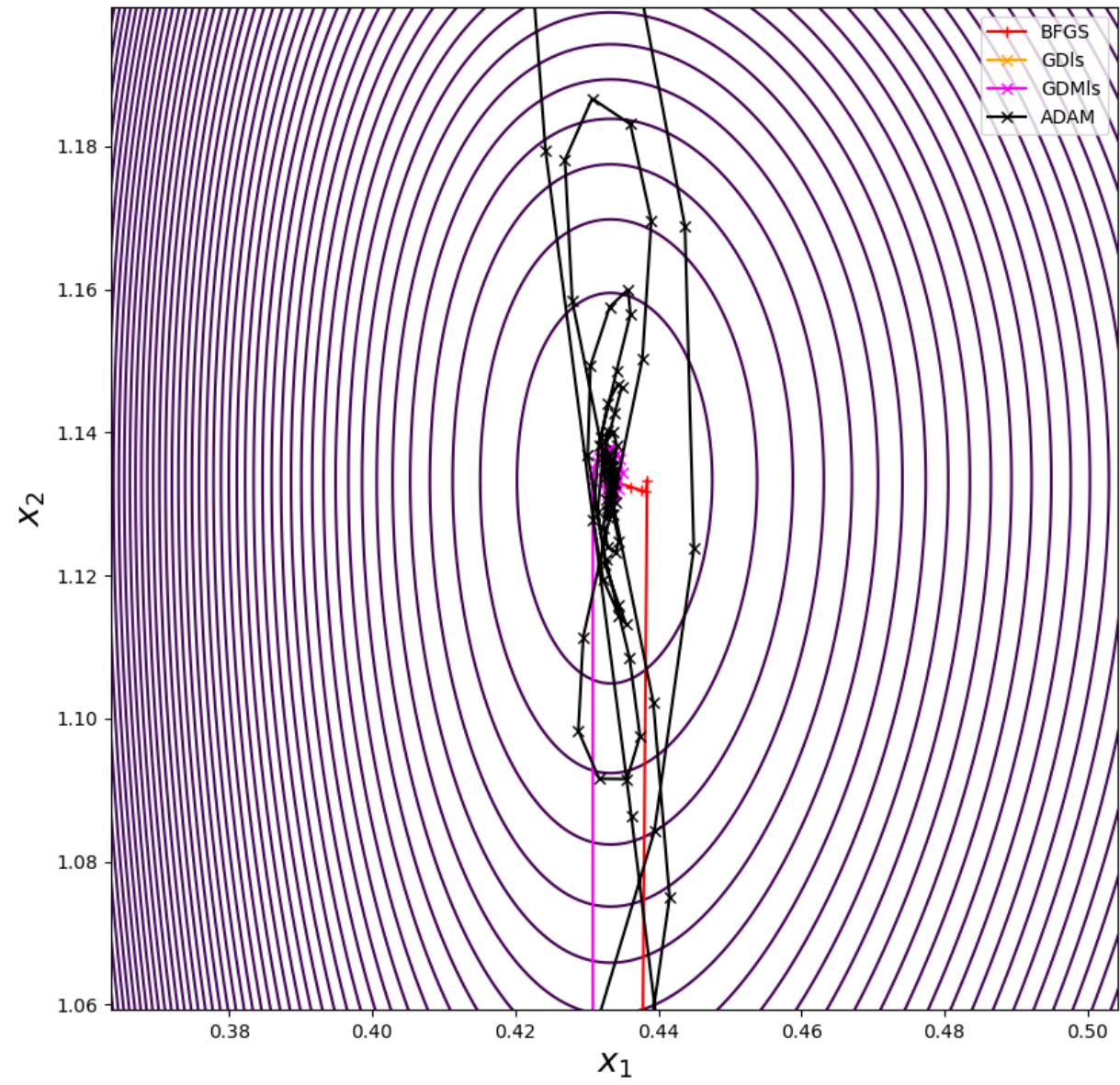
Bsp 2 $\Phi(\underline{x}) = (1 + (3x_1 - 2.3)^3)^2 + (1 + (0.7 - \frac{3}{2}x_2)^3)^2$

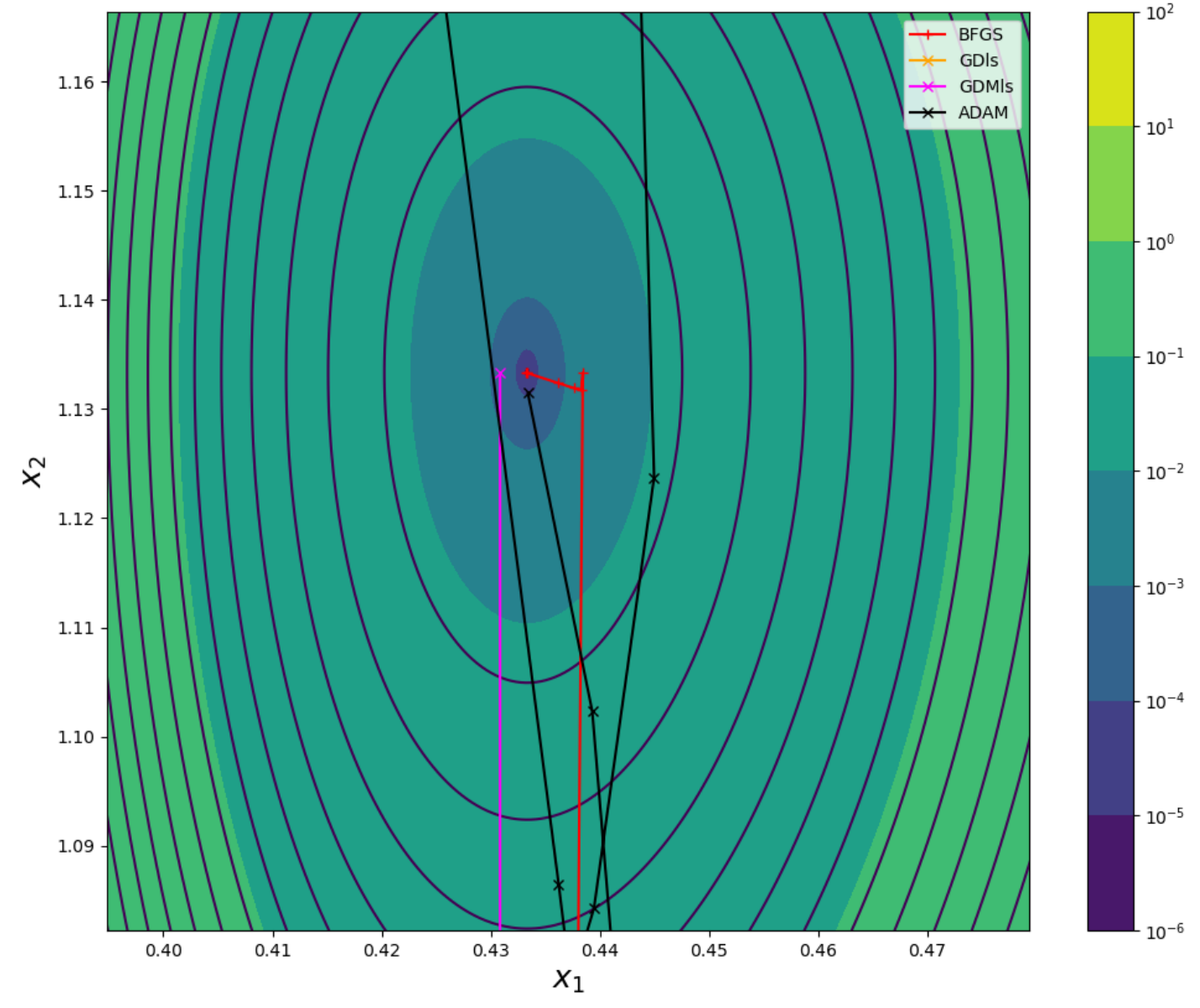
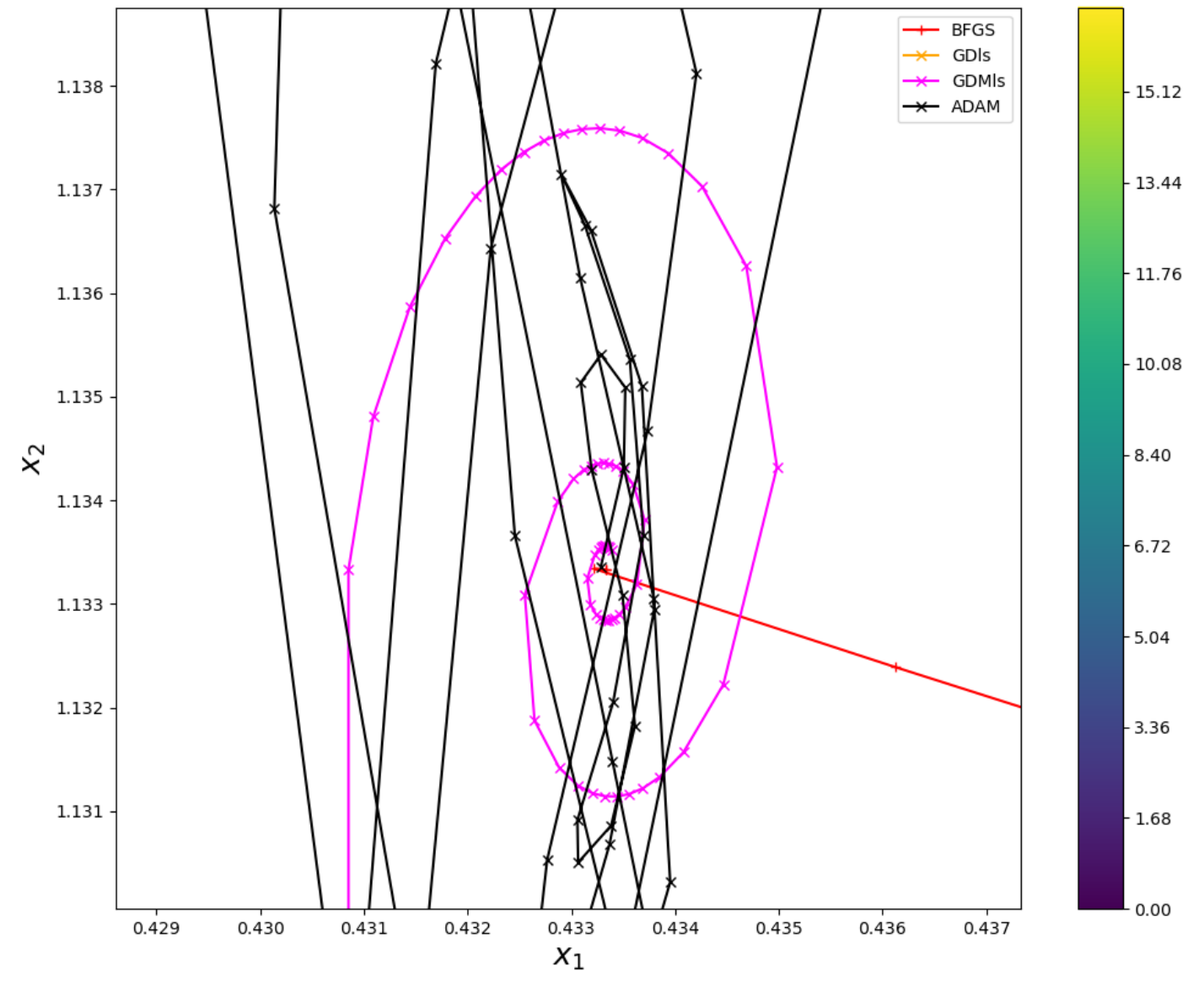
$\nabla\Phi, \nabla^2\Phi$ exakt + 1D minimize

TOL = 10^{-6} oder TOL = 10^{-2}









8.6

Parameter estimation aus ODE

$$(1) \begin{cases} \dot{\underline{u}} = \underline{g}(t, \underline{x}, \underline{u}) \in \mathbb{R}^d & t \in \mathbb{R} \text{ Zeit} \\ \underline{x} \in \mathbb{R}^n \text{ Parametr.} \\ \underline{u}(0) = \underline{u}_0 \in \mathbb{R}^d \\ \underline{u}(t, \underline{x}) \in \mathbb{R}^d \text{ Zustand} \end{cases}$$

Messungen $\underline{y}_1 \approx \underline{u}(t_1, \underline{x}^*), \dots, \underline{y}_m = \underline{y}(t_m, \underline{x}^*)$

Residuum $F: \mathbb{R}^n \rightarrow \mathbb{R}^{m \cdot d}$ $j=1, 2, \dots, d$

$$F_{(m,j)}(\underline{x}) = u_j(t_m, \underline{x}) - (y_m)_j$$

Zielfunktion: $\Phi(\underline{x}) = \frac{1}{2} \|\underline{F}(\underline{x})\|_2^2$

$$\Phi(\underline{x}) = \frac{1}{2} \sum_{m=1}^m \sum_{j=1}^d (u_j(t_m, \underline{x}) - (y_m)_j)^2$$

$$= \frac{1}{2} \sum_{m,j} F_{(m,j)}(\underline{x})^2$$

Ziel: $\underline{x}^* = \arg \min \Phi(\underline{x})$

$$\underline{x} \in \bar{B} \subset \mathbb{R}^n$$

↳ Bereich admissible Parameter.

Im Prinzip: $DF(\underline{x}^*)^T - F(\underline{x}^*) = 0$ ist möglich, bräuche zusätzliche ODE,

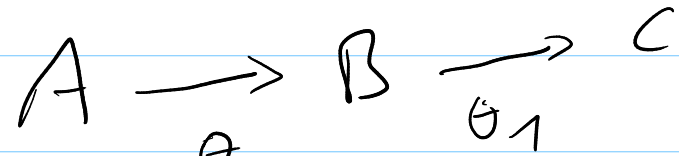
ist zu kompliziert

aber in Praxis \swarrow funktioniert selten

Einfacher: kombiniere die starken Optimierungstools aus scipy mit ivpsolve

ACHTUNG: für gewisse Parameter kann die ODE sehr steif sein \Rightarrow implizite Solver
 \uparrow
 teuer!

Bsp Reaktion:



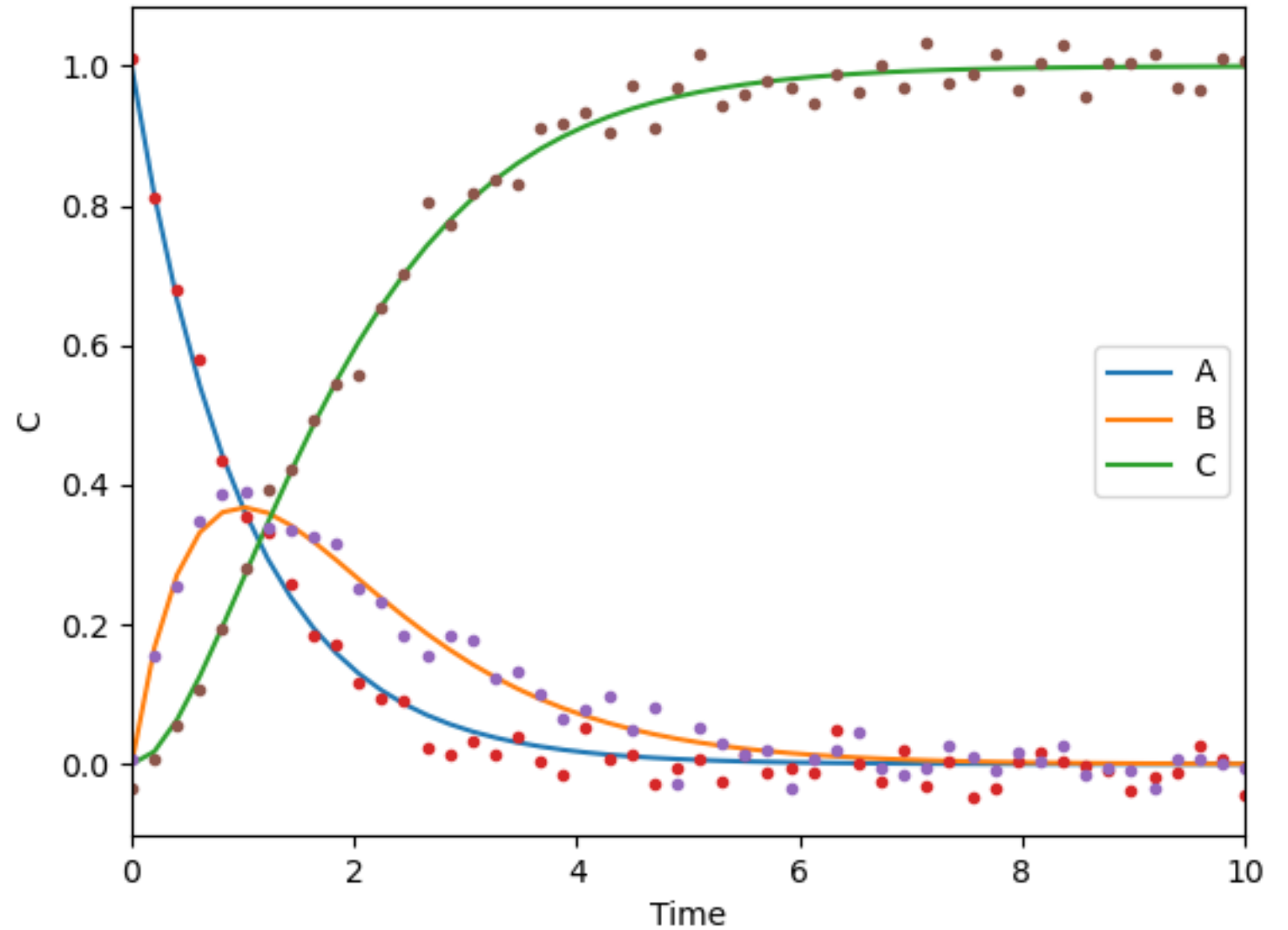
$$\underline{u} = \begin{bmatrix} c_A \\ c_B \\ c_C \end{bmatrix} \quad \dot{\underline{u}} = \begin{bmatrix} -\theta_0 & 0 & 0 \\ \theta_0 & -\theta_1 & 0 \\ 0 & \theta_1 & 0 \end{bmatrix} \underline{u}$$

$$\underline{u}(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad u: [0, 10] \rightarrow \mathbb{R}_+$$

$$f(\theta) = \underline{u}(10)$$

$\theta = [1, 1]$; Messungen

↓
errate θ daraus



start optimisation
standard: BFGS
message: Optimization terminated successfully.
success: True
status: 0
fun: 0.08322427869240534
x: [1.007e+00 1.000e+00]
nit: 8
jac: [2.557e-06 -1.062e-07]
hess_inv: [[2.463e-01 -1.206e-01]
[-1.206e-01 3.300e-01]]
nfev: 36
njev: 12

standard BFGS, jac = 3-point
message: Optimization terminated successfully.
success: True
status: 0
fun: 0.08322427869242109
x: [1.007e+00 1.000e+00]
nit: 8
jac: [2.562e-06 -9.856e-08]
hess_inv: [[2.463e-01 -1.206e-01]
[-1.206e-01 3.300e-01]]
nfev: 60
njev: 12

tnc
message: Converged ($|f_n - f_{(n-1)}| \sim 0$)
success: True
status: 1
fun: 0.08322427883444601
x: [1.007e+00 1.000e+00]
nit: 6
jac: [1.973e-05 -1.765e-05]
nfev: 105

least_squares: standard, i.e. trf
[1.00900421 1.02286028] `ftol` termination condition is satisfied. 24 15

least_squares: lm
[1.00900588 1.02285591] `ftol` termination condition is satisfied. 60 None

$\mu = 0.34, \gamma_1=0.001, \gamma_2=0.001, \alpha=0.001$

computed till time: 1.264252850570114

start optimisation

----- brute random search with 1000 samples... -----

RuntimeWarning: invalid value encountered in scalar divide

dudt[3] = (fric*(u[1]*u[2]**2 - R*dudt[2] - g*np.cos(u[0]) - igam1/m*A*B) - R*dudt[2] - g - igam1/m*A*B)

===== brute random search gives starting point: =====

[0.33720436 0.00101525 0.00463321 0.00400535] 0.007009948080765383

run L-BFGS-B...

/usr/lib64/python3.11/site-packages/scipy/optimize/_numdiff.py:576: RuntimeWarning: invalid value encountered in scalar divide

df = fun(x) - f0

standard: L-BFGS-B

message: CONVERGENCE: REL_REDUCTION_OF_F_<= _FACTR*EPSMCH

success: True

status: 0

fun: 0.007009948080765383

x: [3.372e-01 1.015e-03 4.633e-03 4.005e-03]

nit: 1

jac: [1.295e+03 -1.309e+02 1.446e+00 -5.657e-02]

nfev: 15

njev: 3

hess_inv: <4x4 LbfgsInvHessProduct with dtype=float64>

error = 0.005481623309378464

run TNC...

/scratch/users/gvasile/LaTeX_doc/Numcourses/NumPhys/Vorlesungen/Prepare/paramode04.py:46

dudt[3] = (fric*(u[1]*u[2]**2 - R*dudt[2] - g*np.cos(u[0]) - igam1/m*A*B) - R*dudt[2] - g - igam1/m*A*B)

TNC

message: Max. number of function evaluations reached

success: False

status: 3

fun: 0.006952453003604948

x: [3.372e-01 1.017e-03 4.634e-03 4.005e-03]

nit: 5

jac: [5.774e+03 2.250e+03 5.773e+03 1.276e+02]

nfev: 505

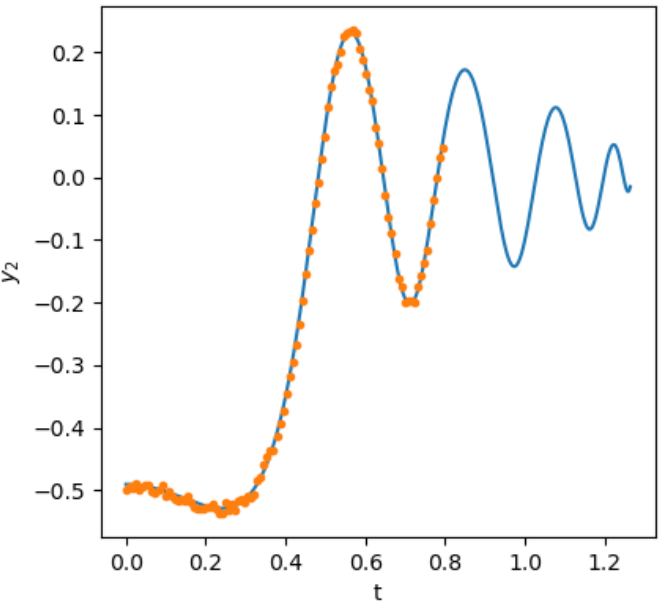
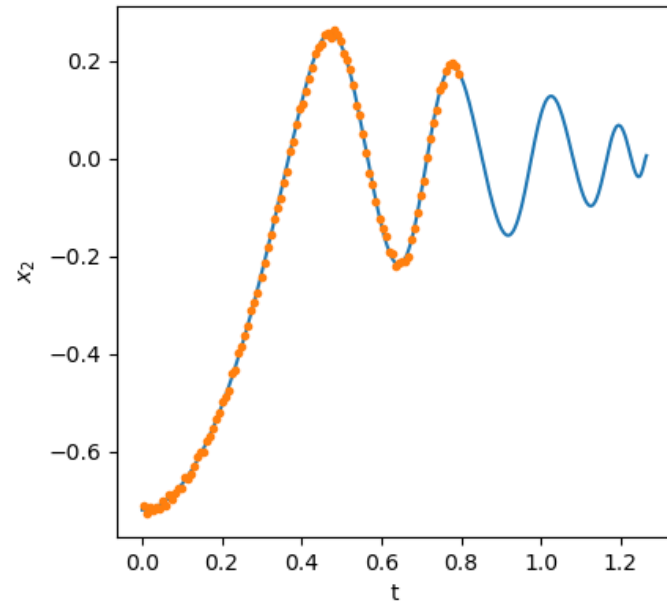
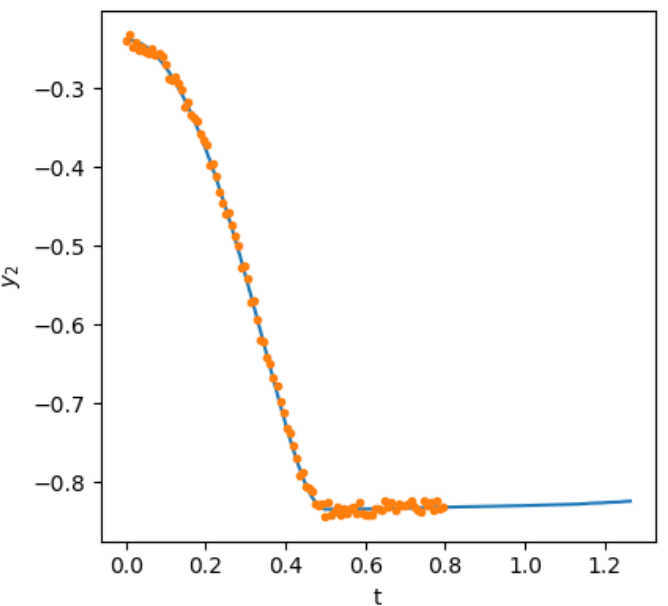
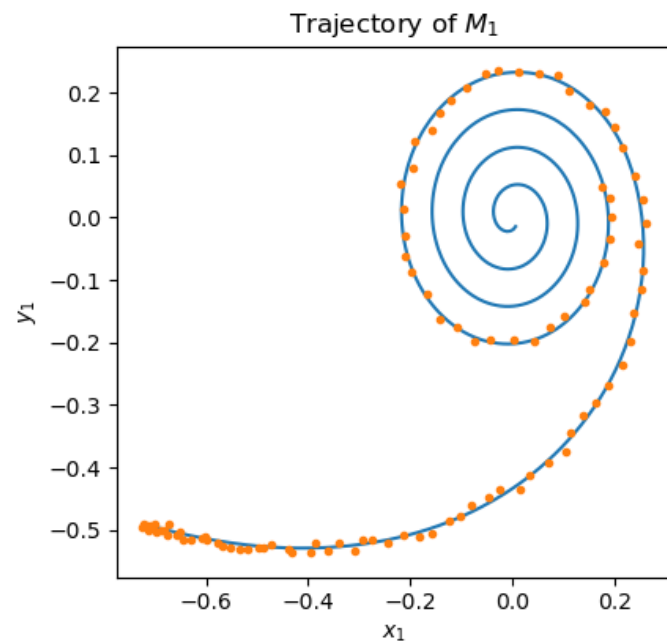
error = 0.005484330850227615

run Least Squares TRF.....

least_squares: standard, i.e. trf

[0.33701153 0.00101449 0.00482116 0.00707985] `xtol` termination condition is satisfied. 16

error = 0.007777984926671802



Recap:
message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
success: True
status: 0

fun: 0.007009948080765383
x: [3.372e-01 1.015e-03 4.633e-03 4.005e-03]
nit: 1
jac: [1.295e+03 -1.309e+02 1.446e+00 -5.657e-02]
nfev: 15
njev: 3

hess_inv: <4x4 LbfgsInvHessProduct with dtype=float64>
error = 0.005481623309378464

message: Max. number of function evaluations reached
success: False
status: 3

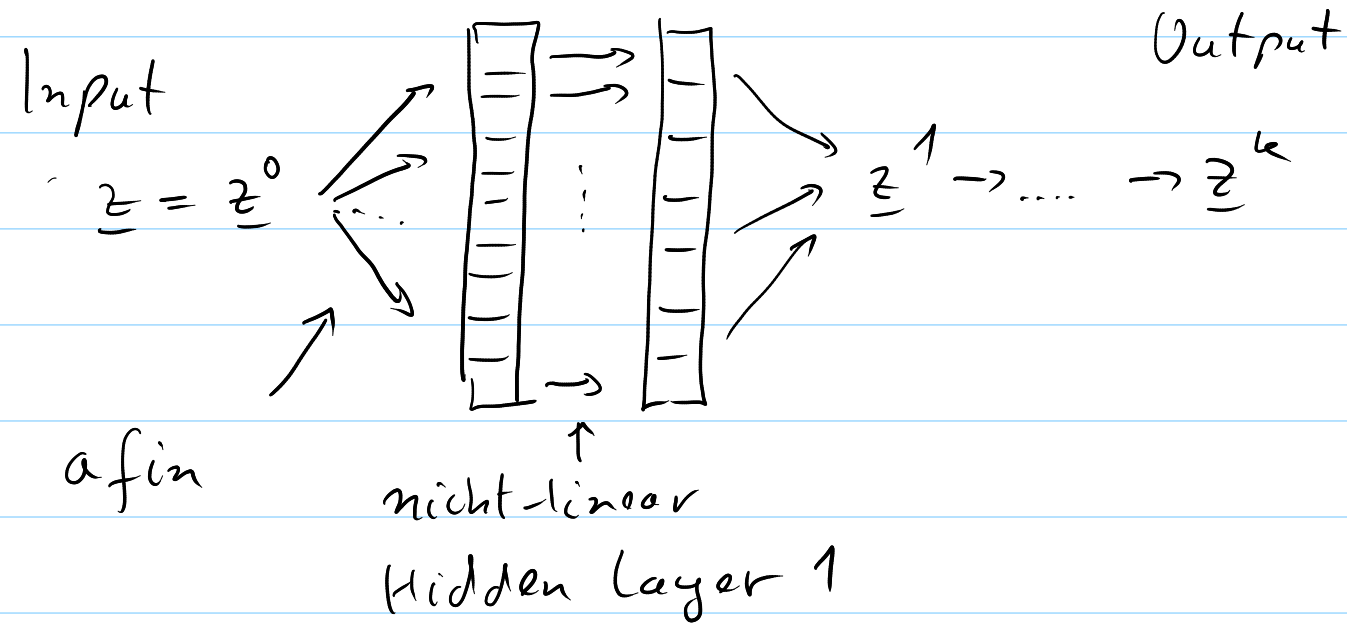
fun: 0.006952453003604948
x: [3.372e-01 1.017e-03 4.634e-03 4.005e-03]
nit: 5
jac: [5.774e+03 2.250e+03 5.773e+03 1.276e+02]
nfev: 505

error = 0.005484330850227615

[0.33701153 0.00101449 0.00482116 0.00707985] `xtol` termination condition is satisfied. 16 9
error = 0.007777984926671802

8.7. DNNs + PINNs

DNN = Deep Neuronal Network



$z \in \mathbb{R}^d$ Gewichte

$z_1 = z$ ← Bias

$z_1 = \underline{W}^1 z + \underline{b}^1$ ← activation function

$z_2 = \sigma(\underline{W}^1 z + \underline{b}^1)$

$k = \text{Tiefe des DNNs.}$

$$f(z) = \sigma \circ c_k \circ \sigma \circ c_{k-1} \dots \circ c_2 \circ \sigma \circ c_1(z)$$

$$W = \{ \underline{W}^j, j=1, \dots, k \} \quad B = \{ \underline{b}^j, j=1, \dots, k \}$$

$$\Theta = (W, B) \in \mathbb{R}^{\sum_{j=1}^k (d_{j+1}) d_j} \quad 10^5$$

Wahl der "activation funktion" $\sim 10^{11}, 10^{12}$

Wikipedia:

Name	Plot	Function, $g(x)$	Derivative of g , $g'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty, \infty)$	C^∞
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	0	$\{0, 1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1 + e^{-x}}$	$g(x)(1 - g(x))$	$(0, 1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - g(x)^2$	$(-1, 1)$	C^∞
Soboleva modified hyperbolic tangent (smht)		$\text{smht}(x) \doteq \frac{e^{ax} - e^{-bx}}{e^{cx} + e^{-dx}}$		$(-1, 1)$	C^∞
Rectified linear unit (ReLU) ^[10]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max(0, x) = x \mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$[0, \infty)$	C^0
Gaussian Error Linear Unit (GELU) ^[2]		$\frac{1}{2}x \left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$	$(-0.17\dots, \infty)$	C^∞
Softplus ^[11]		$\ln(1 + e^x)$	$\frac{1}{1 + e^{-x}}$	$(0, \infty)$	C^∞
Exponential linear unit (ELU) ^[12]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C^1 & \text{if } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$
Scaled exponential linear unit (SELU) ^[13]		$\lambda \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$	$\lambda \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\lambda\alpha, \infty)$	C^0
Leaky rectified linear unit (Leaky ReLU) ^[14]		$\begin{cases} 0.01x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$(-\infty, \infty)$	C^0
Parametric rectified linear unit (PReLU) ^[15]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α	$\begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
Sigmoid linear unit (SiLU, ^[2] Sigmoid shrinkage, ^[16] SiL, ^[17] or Swish-1 ^[18])		$\frac{x}{1 + e^{-x}}$	$\frac{1 + e^{-x} + xe^{-x}}{(1 + e^{-x})^2}$	$[-0.278\dots, \infty)$	C^∞
Gaussian		e^{-x^2}	$-2xe^{-x^2}$	$(0, 1]$	C^∞

Supervised Learning

Zielfunktion $f: \mathbb{R}^d \rightarrow \mathbb{R}$
 $y_i \Rightarrow f(y_i) + \text{Rausch} = \tilde{f}_i$

Trainingsmenge $\mathcal{J} = \{ \underline{y}_n \in \mathbb{R}^d, n=1, \dots, M \}$

Finde $\Theta \in \Theta \subset \mathbb{R}^{\sum_{j=1}^k (d_j + 1) d_j}$ so dass $f^*(\Theta) \approx f$

$\Theta = \{ \underline{W}_1, \underline{b}_1, \underline{W}_2, \underline{b}_2, \dots, \underline{W}_k, \underline{b}_k \}$

+ stetige aber stückweise lineare Funktionen
 => viele Parameter möglich, leicht zu berechnen

+ Nacheinanderausführung => deep network

Loss:
$$J(\theta) = \frac{1}{N} \sum_{n=1}^N |f(y_n) - f_{\theta}^*(y_n)|^p$$

$$1 \leq p < \infty$$

$p=2 \Rightarrow$ Methode der kleinsten Quadrate

$p=1 \Rightarrow$ "sparse" / dünn besetzte Darstellung.

solche θ minimiere \Rightarrow "trained NN" = θ
 verwende $f_{\theta}^*(y)$ für neue y .
 \Rightarrow neue Werte.

Bem.: ML funktioniert am besten, wenn das Problem **unterdeterminiert** ist
 ML funktioniert gar nicht, wenn das Problem überdeterminiert ist: overfitting!

Kunst von K.I. = aus der grossen Menge möglicher Lösungen finde eine, die sich auf neue Daten/messungen verallgemeinern lässt

2012: ImageNet: 1.2 Millionen Bilder; AlexNet verwendete 60 Millionen Gewichte!
 (5 Tage GradientDescent)



Schlüsselideen für DNNs:

- + Nacheinanderausführung (go deep)
- + Kettenregel (Ableitungen nach Parameter sind möglich)
- + Stochastik Gradient Descent
- + Backpropagation (schnelle Kettenregel)
- + Nichtlinearität der Aktivierungsfunktion

$$F_k(z) = \sigma(\underline{w}^k z + \underline{b}^k) = \sigma(e_k)$$

$$f(z) = \sigma(e_k) \circ e_{k-1} \dots \sigma(e_1) = F_k(F_{k-1} \dots F_1(z))$$

Parameter: $\underline{w}^1, \underline{b}^1, \dots, \underline{w}^k, \underline{b}^k$

$$\sigma(z) = \text{RE}(u(z))$$

Ziel: Klassifizierung:

Hund \leftrightarrow Katze
 an fahrendes Auto/Zug
 weg fahrendes Auto/Zug

Backpropagation:

Kettenregel: $\frac{\partial F}{\partial \theta} = \frac{\partial F_k}{\partial F_{k-1}} \frac{\partial F_{k-1}}{\partial F_{k-2}} \dots \frac{\partial F_1}{\partial \theta}$

Bei $\frac{\partial F}{\partial \theta_j}$ verwendet nur $\theta_j, \theta_{j+1}, \dots, \theta_k$

\Rightarrow Schelle n^2 -Operationen
(von rechts nach links
wie n^3 Operationen
da Matrix \times Matrix!)

Notiere $l = (f - \underline{F}(\theta, y))^T (f - \underline{F}(\theta, y))$

2. Idee: verwende die Vorarbeit

1. Idee:

$\frac{\partial l}{\partial F} = 2(f - \underline{F}(\theta, y))$ vektor

$\frac{\partial l}{\partial F_k}$ für alle $j \leq k$

$M_{\rightarrow k} = \frac{\partial F_k}{\partial F_{k-1}}$ Matrix

$\frac{\partial l}{\partial \theta_j} = \left(\left(\frac{\partial l}{\partial F} M_k \right) M_{k-1} \dots M_{j+1} \right)$

$\square \square = n^2$ Operationen

Alg: Forward (Initialisierung)
 $\underline{y} \mapsto$ speichere $\bar{F}_j := \sigma c_j$

für $j = k, k-1, \dots, 1$:

$$\frac{\partial \ell}{\partial b_j} = \frac{\partial \ell}{\partial \bar{F}_j} \frac{\partial \bar{F}_j}{\partial b_j}$$

$$\frac{\partial \ell}{\partial w_j} = \frac{\partial \ell}{\partial \bar{F}_j} \frac{\partial \bar{F}_j}{\partial w_j} \quad \swarrow \text{speichere}$$

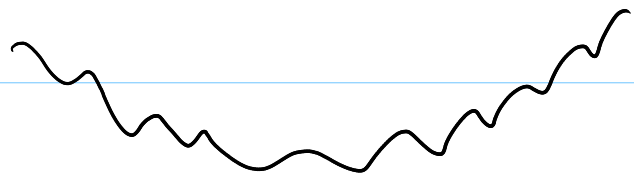
falls $j \neq 1$:

$$\frac{\partial \ell}{\partial \bar{F}_{j-1}} = \frac{\partial \ell}{\partial \bar{F}_j} \frac{\partial \bar{F}_j}{\partial \bar{F}_{j-1}}$$

Keras; TensorFlow; PyTorch

Optimierung

J nicht konvex aber diff^{bar} (leicht zu berechnen)



↑
automatisch.

Gradientenverfahren

Für $l=0, 1, 2, \dots$

$$\theta_{l+1} = \theta_l - \eta_l \text{grad}_{\theta} J(\theta_l) \quad \eta_l \ll 1$$

↳ (adaptiv) learning rate

konvergiert zu einem lokalen Minimum.

$$\text{grad}_{\theta} J(\theta_l) = \frac{1}{n} \sum_{m=1}^n \text{grad}_{\theta} J_m(\theta_l)$$

$$J_m(\theta_l) = (f(y_m) - f_{\theta_l}(y_m))^2$$

$n = \text{sehr gross} \neq \text{Speicher} \neq \text{Zeit}$

↑
steepest descent

Stochastic Gradient

1-SG

Im l -ten Schritt θ_l gegeben.

wähle $m_l \in \{1, \dots, n\}$ zufällig.

$$\theta_{l+1} = \theta_l - \eta_l \text{grad}_{\theta} J_{m_l}(\theta)$$

n-SG

$S = \cup S_j$, $\# S_j = n$ und $\frac{M}{n}$ Untergruppen.
↳ Batches der Länge n

Im l -te Schritt:

$$\theta_{l+1} = \theta_l - \eta_l \sum_{m \in S_j} \text{grad}_{\theta} J_m(\theta_l)$$

$S_1, S_2, \dots, S_{\frac{M}{n}}$ Batches

Ein durchlauf aller Batches = 1 Epoch
Dann Batches umordnen. (zufällig)

Methoden: GD, ADAM, SGD, AdaGrad, GDM, RMSProp, etc.

L-BFGS-R 2. ordnung, kann mit Batches arbeiten aber meistens zu teuer

Theorem Gegeben $\varepsilon > 0$ gibt es ein NN $f^* = f_{\theta^*}$
mit $\theta^* \in \Theta \subseteq \mathbb{R}^d$ so dass

$$\|f - f^*\|_X < \varepsilon$$

"Approximations eigenschaft der NN"

∇ = stetig. kein Polynom!

→ kein Hinweis auf Konstruktion. ☹️

PINN = Physical instructed Neuronal Network.

$$\dot{u} = g(t, u) \quad u \approx u_{\theta}$$

Trainingsmenge $\mathcal{T} = \{t_1, t_2, \dots, t_M\}$

Testmenge $\tilde{\mathcal{T}} = \{\tilde{t}_1, \tilde{t}_2, \dots, \tilde{t}_N\}$

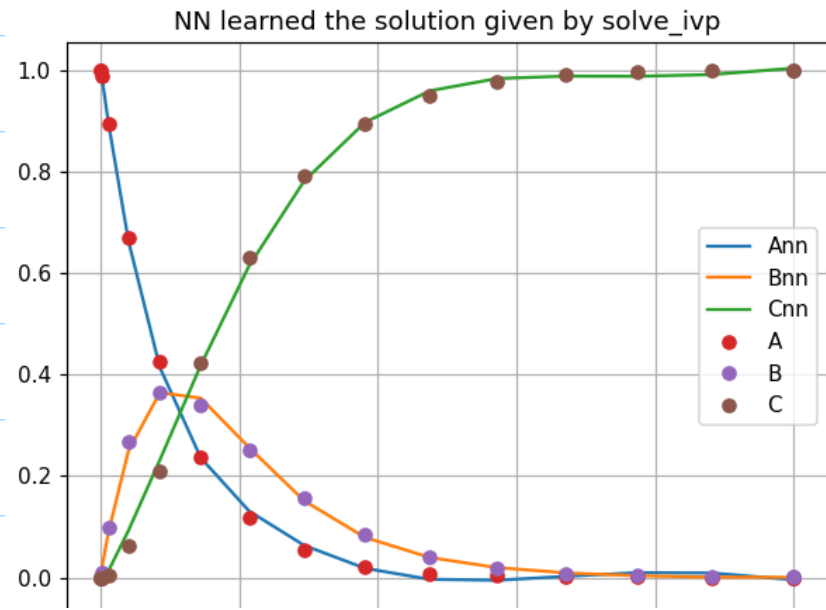
$$R_T(t_i) = u_{\theta}(t_i) - u_i \quad \begin{array}{l} \text{Kollokationspunkte} \\ \text{auch automatisch berechnen.} \end{array}$$

$$R_D(\tilde{t}_i) = \dot{u}_{\theta}(\tilde{t}_i) - g(\tilde{t}_i, u_{\theta}(\tilde{t}_i))$$

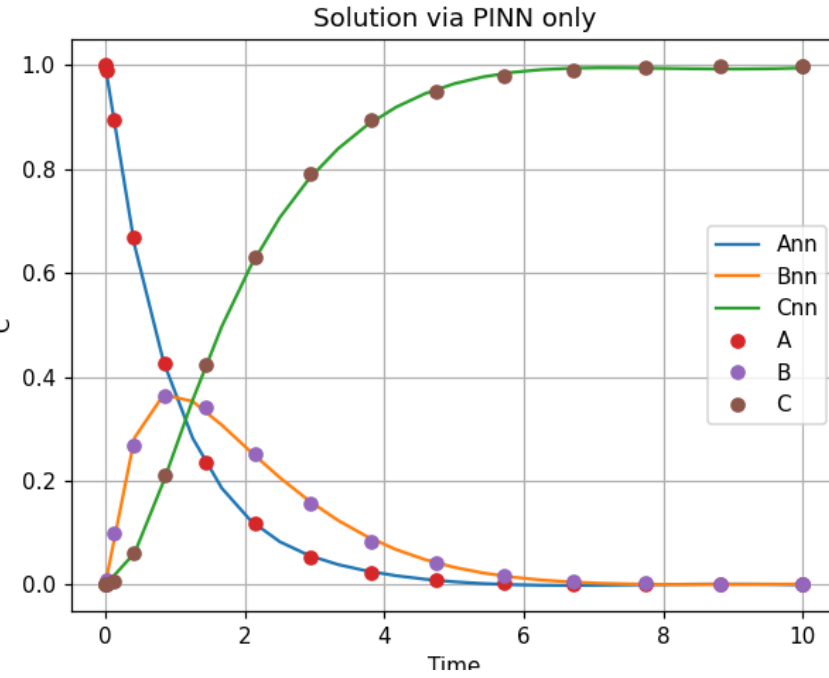
$$\mathcal{J}(\theta) = \frac{1}{M} \sum_{m=1}^M R_T(t_m)^2 w_m + \frac{1}{N} \sum_{j=1}^N R_D(\tilde{t}_j) \tilde{w}_j$$

$$\ominus \hat{\theta} = \arg \min_{\theta} \mathcal{J}(\theta) \quad \Rightarrow \quad u_{\theta^*}(t)$$

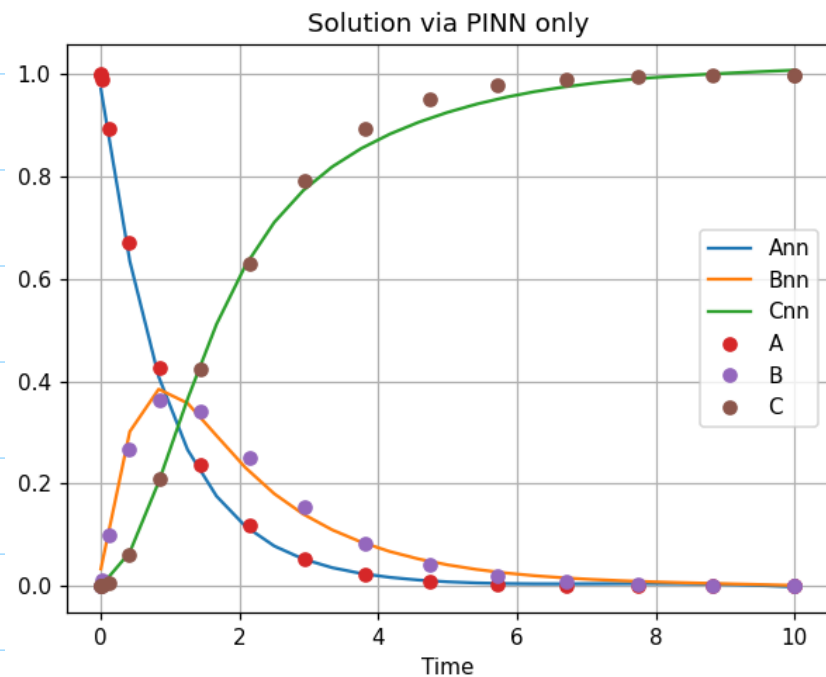
$\dot{u} = g(t, u, \mu)$ noch μ auch optimieren!



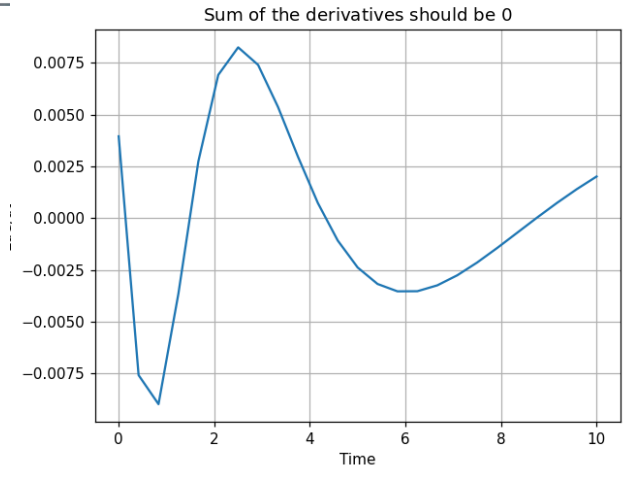
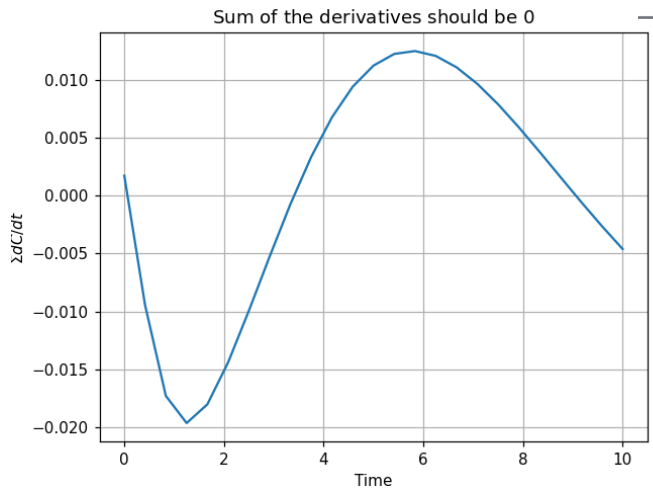
Layer: [1, 8, 3], $Lr = 10^{-2}$
 $[1, 8, 3], Lr = 10^{-3}$
 5000 steps



python onlyautograd02.py



500 steps



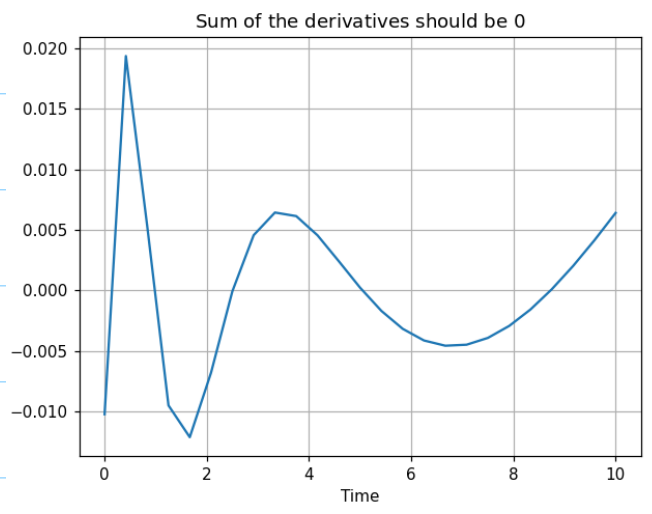
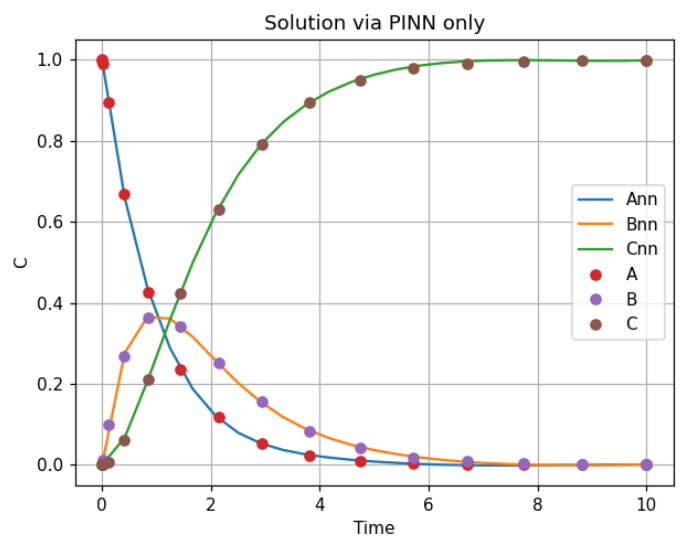
A NN can learn the solution...
 Solve the ODE with a PINN using adam
 Iteration 0 objective 4.901094881441509
 Iteration 100 objective 0.6899595433185512
 Iteration 200 objective 0.5256140910911087
 Iteration 300 objective 0.30172420828432256
 Iteration 400 objective 0.06713165176847836
 Iteration 500 objective 0.020769341937793753
 0.2 minutes elapsed this time. Total time = 0.19 min. Total epochs = 501.

```

Iteration 2300 objective 0.21837992273256943
Iteration 2400 objective 0.15067501923377044
Iteration 2500 objective 0.10750242875475156
Iteration 2600 objective 0.08290887034465583
Iteration 2700 objective 0.06843241704758321
Iteration 2800 objective 0.05949095869641032
Iteration 2900 objective 0.053362240536638864
Iteration 3000 objective 0.048461984808600256
Iteration 3100 objective 0.044106230164504705
Iteration 3200 objective 0.040059307858911604
Iteration 3300 objective 0.03624299672367966
Iteration 3400 objective 0.03262181017619072
Iteration 3500 objective 0.029170944087145
Iteration 3600 objective 0.025872821077357253
Iteration 3700 objective 0.022721699315778818
Iteration 3800 objective 0.01972928511329969
Iteration 3900 objective 0.016927537127111775
Iteration 4000 objective 0.014364117294925314
Iteration 4100 objective 0.012086484009305466
Iteration 4200 objective 0.010119205240268183
Iteration 4300 objective 0.008452583236811351
Iteration 4400 objective 0.007054863163250895
Iteration 4500 objective 0.005891236235982176
Iteration 4600 objective 0.004930063096682349
Iteration 4700 objective 0.004142388843457355
Iteration 4800 objective 0.0035023909476854915
Iteration 4900 objective 0.0029875362493808344
Iteration 5000 objective 0.0025780349882505854
2.0 minutes elapsed this time. Total time = 2.02 min.

```

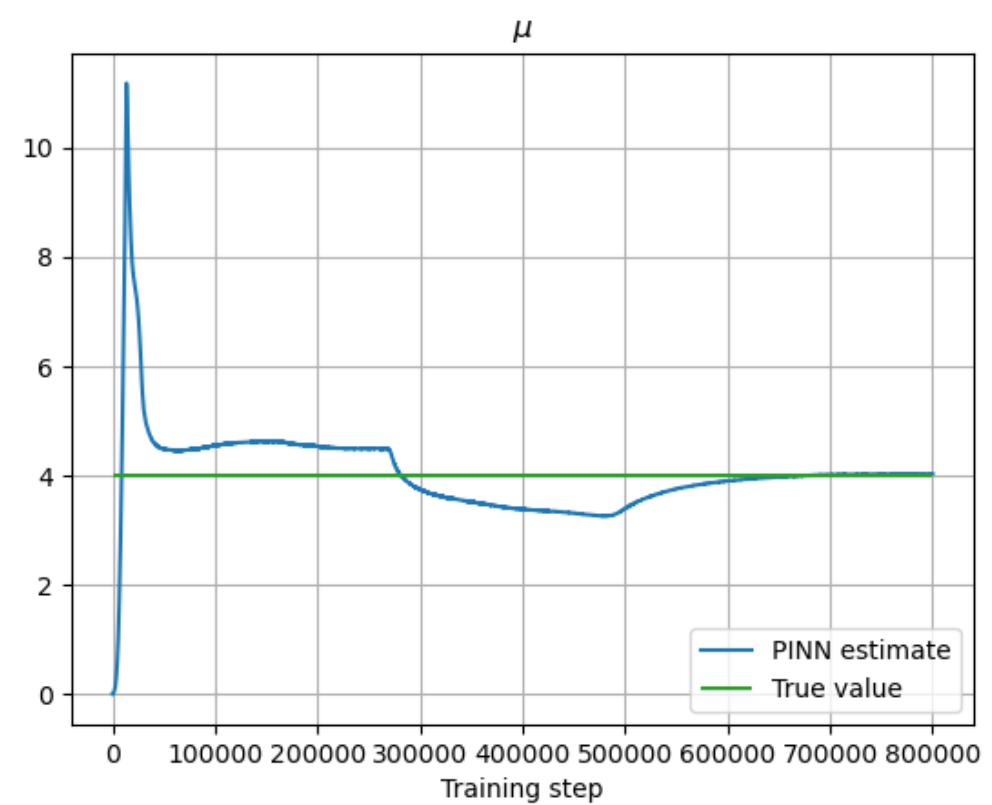
$[1, 8, 8, 3], lr = 10^{-5}$



```

A NN can learn the solution...
Solve the ODE with a PINN using adam
Iteration 0 objective 1.2060713847102091
Iteration 100 objective 0.06426963241254897
Iteration 200 objective 0.0037654361568503357
Iteration 300 objective 0.0013668805784190507
Iteration 400 objective 0.0007517324457044832
Iteration 500 objective 0.0004981669314175369
0.3 minutes elapsed this time. Total time = 0.32 min.

```

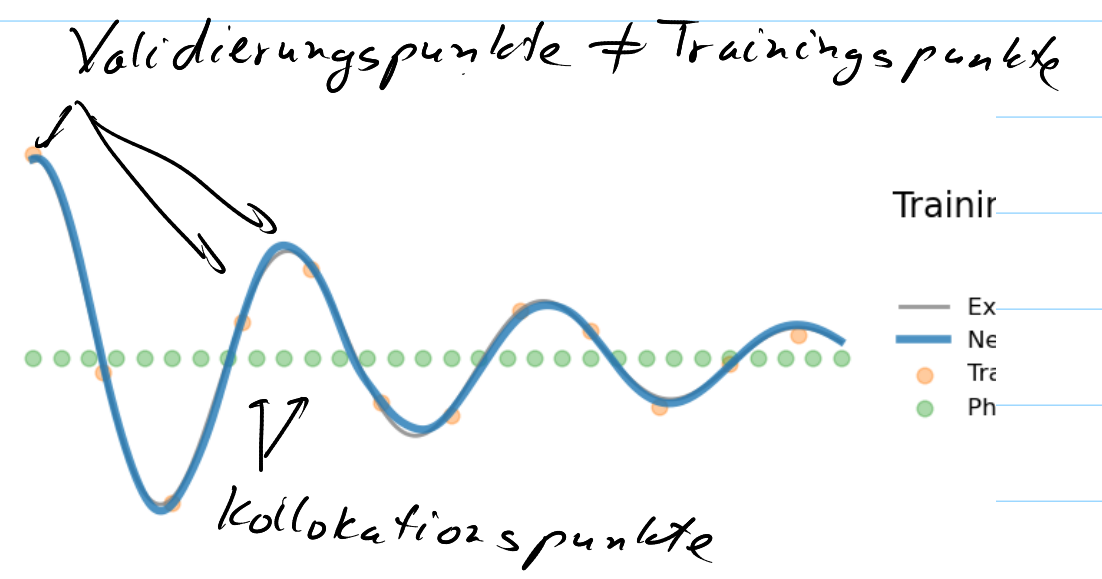


$\ddot{u} + \mu \dot{u} + u = 0$

noise = $0.65 * \text{uniform}(-1, 1)$

w^2

3 layers 32 neurons



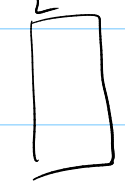
$$\| \underline{Ax} - \underline{b} \|_2$$

$$\underline{Ax} = \underline{b}$$

$\underline{A} \in \mathbb{R}^{m \times n}$ m, n sehr sehr gross

Zugang nur zu Zeilen von \underline{A}

Hyperebene: $\underline{A}_{i,:} \cdot \underline{x} = \underline{b}_i$

n 

$$\underline{A} = \begin{bmatrix} 4 & 1 \\ 2 & 1/2 \\ 3 & 3/2 \\ 0 & 1 \end{bmatrix}; \underline{b} = \begin{bmatrix} 2 \\ 3 \\ 6 \\ 2 \end{bmatrix}$$

$$\underline{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

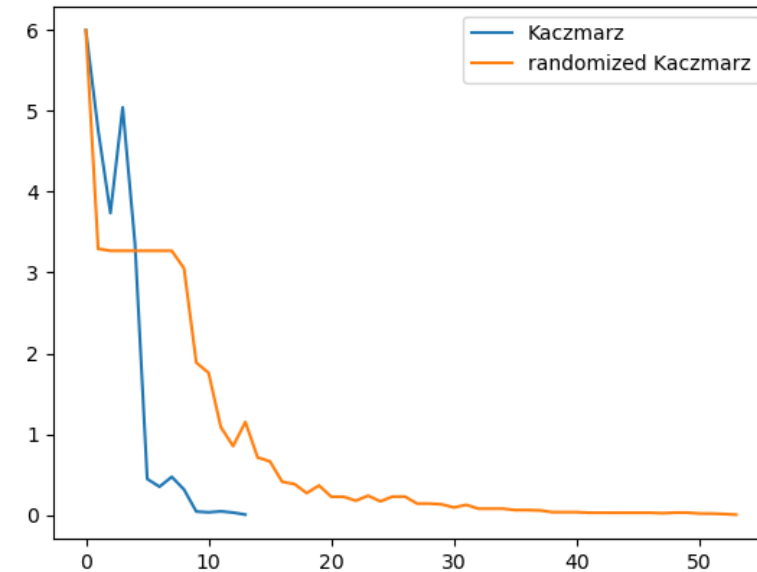
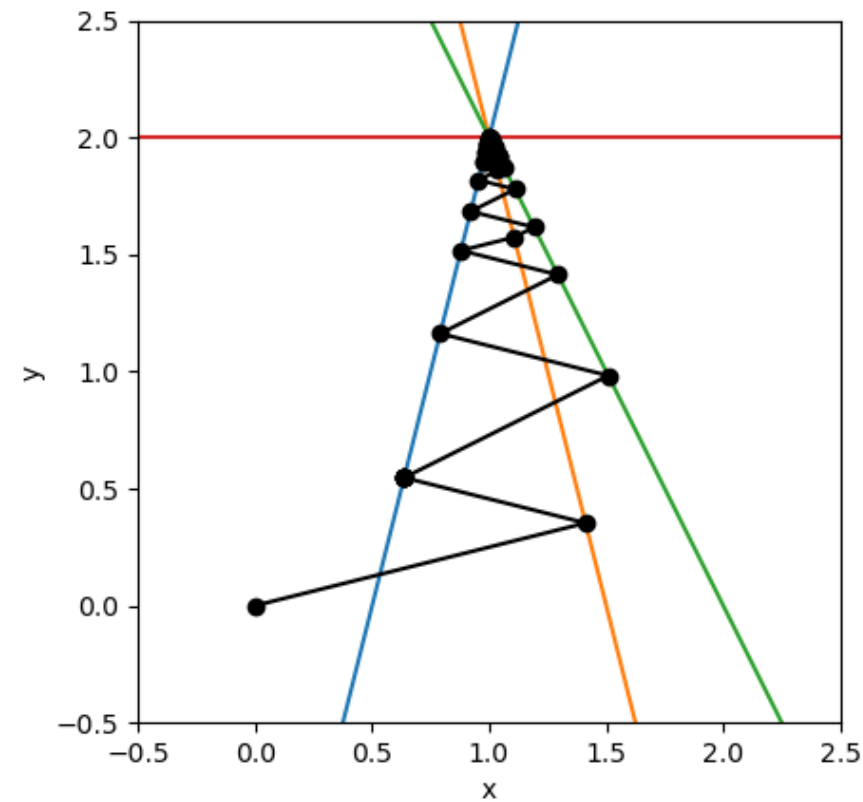
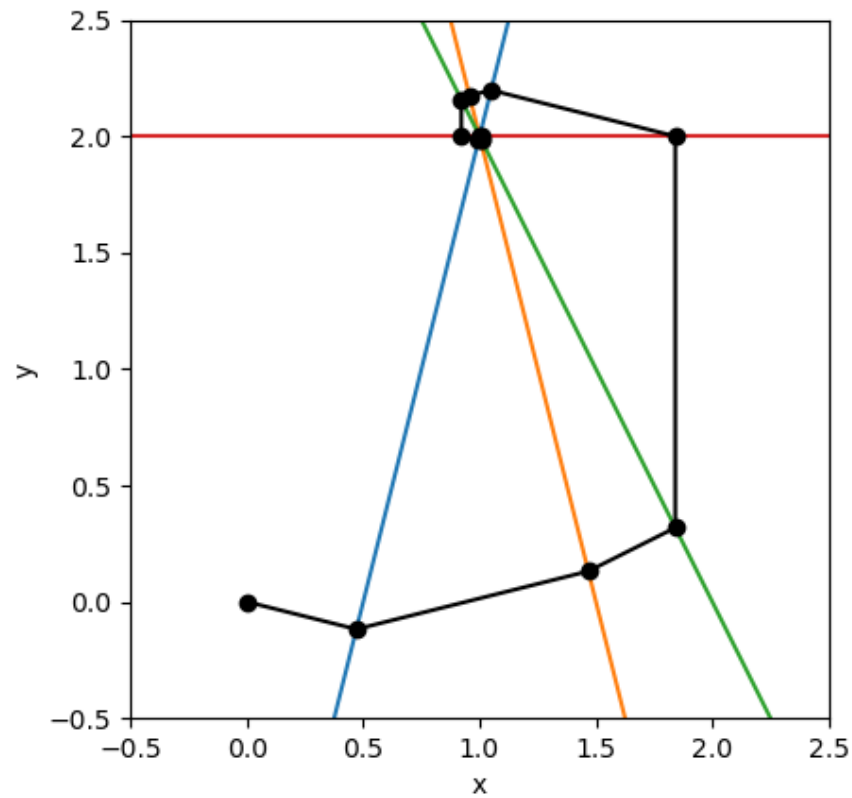
randkacz.py

$$m \begin{bmatrix} n \end{bmatrix}$$

$m=10^3, n=10^5, \text{density} = 0.05, \text{random sparse}$

$$+ \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

2. spalte \Rightarrow rank(A) = m



Abhaengig von der density kann sparse besser als voll sein.
Etwa gleich schnell sind sie bei density = 0.1 (d.h 10% der Eintraege sind nicht Null).
density < 0.1 => sparse gewinnt.

bei density=0.05 hatte ich:

```
In [149]: %%timeit None
...:     ind_s, ind_e = A.indptr[i:i+2]
...:     vs = A.data[ind_s:ind_e]
...:     ind = A.indices[ind_s:ind_e]
...:     aiX = (vs*X[ind]).sum()
...:     nai = (vs**2).sum()
...:     X[ind] += ((b[i] - aiX)/nai) * vs
...:
29.4 µs ± 207 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

```
In [150]: ai = MA[i]
```

```
In [151]: %%timeit None
...: (b[i] - ai @ X) / np.linalg.norm(ai)**2 * ai
...:
...:
50.8 µs ± 654 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

```
In [141]: timeit spsolve_kaczmarz(A, b, 1.e-2, randomize=Tr
...: rue)
687 ms ± 81.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

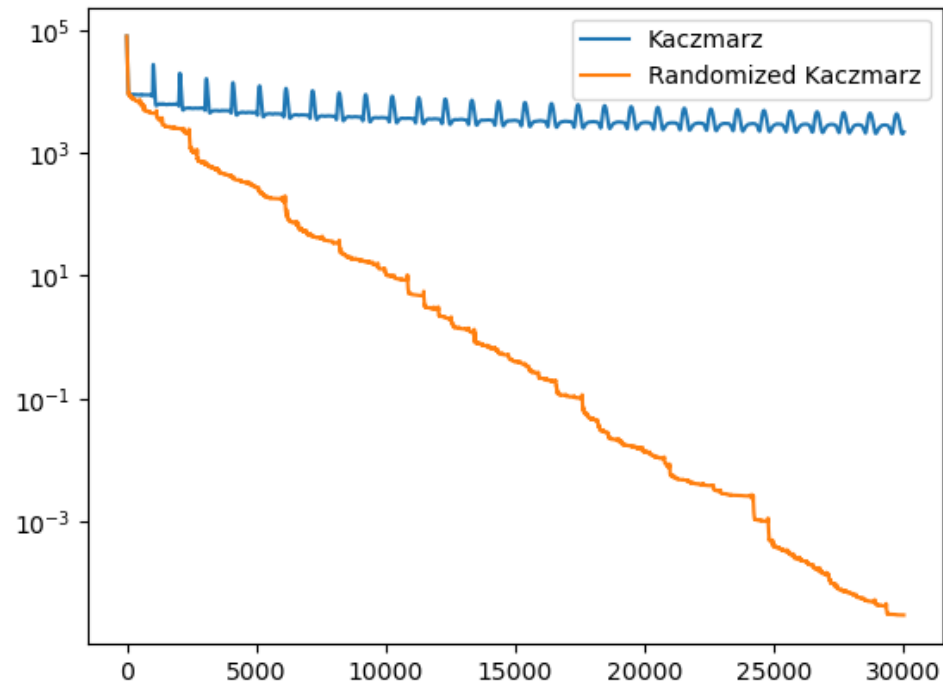
```
In [142]: timeit spsla.lsqr(A,b)
14.2 ms ± 5.07 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
In [143]: timeit spsla.lsqr(A,b)
The slowest run took 10.40 times longer than the fastest. This could mean that an intermediate
47.3 ms ± 49.9 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
In [144]: timeit spsla.lsqr(A,b)
16.4 ms ± 7.64 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
In [145]: timeit np.linalg.lstsq(MA,b, rcond=-1)
The slowest run took 5.07 times longer than the fastest. This could mean that an intermediate
1.39 s ± 896 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

sparse lsqr gewinnt immer locker, auch bei density 0.25! Arpack ist schwer zu schlagen!



§9 Eigenwerte

§9.1 Einführung

* kein Alg. um exakt die EW, EV für eine allgemeine Matrix zu berechnen!
 => Approximation von EW, EV braucht iterativen Verfahren!

* Software: $\text{eig}(A)$ kostet $(\approx N^3)$ für $N \times N$
 $\text{eigh}(A)$ kostet $(\approx N^2)$ für $A^H = A$

↳ QR-Algorithmus mit Shift

* EV werden typischerweise nicht so gut approximiert.

$$\underline{A}^H = \left(\underline{A}\right)^T \quad \underline{A} \in \mathbb{C}^{n \times n}$$

§9.2 Potenzmethoden

A $n \times n$ Matrix

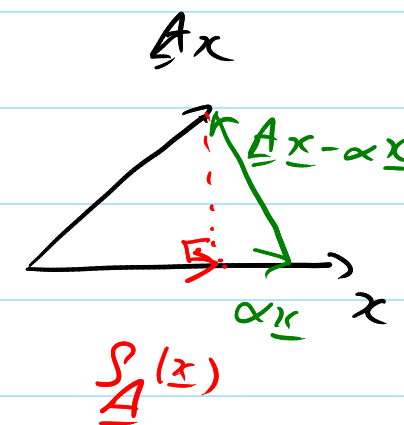
Def Rayleigh-Quotienten

$$\rho_{\underline{A}}(\underline{x}) = \frac{\underline{x}^H \underline{A} \underline{x}}{\underline{x}^H \underline{x}} \quad \text{für } \underline{x} \neq \underline{0}$$

Beh Falls $\underline{x} \in V$ von A dann $\underline{A}\underline{x} = \lambda \underline{x} \Rightarrow$

$$\rho_{\underline{A}}(\underline{x}) = \frac{\underline{x}^H \underline{A} \underline{x}}{\underline{x}^H \underline{x}} = \frac{\underline{x}^H \lambda \underline{x}}{\underline{x}^H \underline{x}} = \lambda$$

Beh $\rho_{\underline{A}}(\underline{x}) = \underset{\alpha}{\text{argmin}} \| \underline{A}\underline{x} - \alpha \underline{x} \|^2$



$$\frac{d}{d\alpha} \| \underline{A}\underline{x} - \alpha \underline{x} \|^2 = 2 \sum_{j=1}^n \left((\underline{A}\underline{x})_j - \alpha x_j \right) x_j = 0$$

$\Rightarrow (\underline{A}\underline{x} - \alpha \underline{x}) \perp \underline{x} \Rightarrow$ min. ist erreicht

wenn $\underline{A}\underline{x} - \alpha \underline{x} \perp \underline{x} \Leftrightarrow \alpha \underline{x} = P_{\underline{x}}(\underline{A}\underline{x}) \Rightarrow$

$$\alpha \underline{x} = \frac{1}{\underline{x}^H \underline{x}} \underline{x} \underline{x}^H (\underline{A} \underline{x}) \quad (\Rightarrow) \quad \alpha \underline{x} = \frac{\underline{x}^H \underline{A} \underline{x}}{\underline{x}^H \underline{x}} \underline{x}$$

$$\Leftrightarrow \alpha = \underline{\rho}_A(\underline{x})$$

Ben $\underline{A} \in \mathbb{R}^{n \times n}$, $\underline{x} \in \mathbb{R}^n$

$$D \underline{\rho}_A(\underline{x}) = \frac{(\underline{x}^T \underline{x}) (\underline{A}^T + \underline{A}) \underline{x} - 2 (\underline{x}^T \underline{A} \underline{x}) \underline{x}}{(\underline{x}^T \underline{x})^2} =$$

$$= \frac{1}{\underline{x}^T \underline{x}} \left[(\underline{A}^T + \underline{A}) - 2 \underline{\rho}_A(\underline{x}) \underline{I} \right] \underline{x}$$

Falls \underline{x}^* EV von $\underline{A} = \underline{A}^T$ ist:

$$D \underline{\rho}_A(\underline{x}^*) = \frac{1}{\underline{x}^{*T} \underline{x}^*} \left(\lambda \underline{x}^* + \lambda \underline{x}^* - 2 \lambda \underline{x}^* \right) = 0$$

$\underline{x}^* \text{ EV von } \underline{A} \Rightarrow \underline{x}^* \text{ Stationärpunkt von } \underline{\rho}_A(\cdot)$

\Rightarrow Taylor für $\underline{\rho}_A(\cdot)$ um $\underline{x}^* \text{ EV von } \underline{A} \Rightarrow$

$$\underline{\rho}_A(\underline{x}) - \overset{\text{EV}}{\underline{\rho}_A(\underline{x}^*)} = 0 + O(\|\underline{x} - \underline{x}^*\|_2^2)$$

für \underline{x} nah an EV \underline{x}^* .

$\Rightarrow \underline{\rho}_A(\underline{x})$ kann als gute Approximation an $\underline{\rho}_A(\underline{x}^*) = \lambda$ funktionieren.

Direkte Potenzmethode

Ziel: finde das betragsgrößte EW von \underline{A} !
und ein EV dazu.

Annahme: \underline{A} diagonalisierbar: $\underline{S}^{-1} \underline{A} \underline{S} = \text{diag}(\lambda_1, \dots, \lambda_n)$

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \dots \geq |\lambda_n| \geq 0$$

$\lambda_1, \dots, \lambda_n \in \mathbb{C}$

$\underline{s}_1, \dots, \underline{s}_n \in \mathbb{C}$
 \hookrightarrow Spalten von \underline{S} $\|\underline{s}_j\|_2 = 1$

$$\mathbb{R}^n \Rightarrow \underline{x} = \sum_{j=1}^n c_j \underline{\Delta}_j \quad \text{mit } c_1 \neq 0, \text{ sonst beliebig.}$$

$$\underline{A} \underline{x} = \sum_{j=1}^n c_j \underline{A} \underline{\Delta}_j = \sum_{j=1}^n c_j \lambda_j \underline{\Delta}_j$$

$$\underline{A}^2 \underline{x} = \sum_{j=1}^n c_j \lambda_j^2 \underline{\Delta}_j$$

$$\dots$$

$$\underline{A}^k \underline{x} = \sum_{j=1}^n c_j \lambda_j^k \underline{\Delta}_j = \lambda_1^k \left(c_1 \underline{\Delta}_1 + \sum_{j=2}^n \left(\frac{\lambda_j}{\lambda_1} \right)^k c_j \underline{\Delta}_j \right)$$

$$\left| \frac{\lambda_j}{\lambda_1} \right| < 1$$

↓ für $k \rightarrow \infty$
0

=> für grosses k zeigt $\underline{A}^k \underline{x}$ in der "Richtung" von $\underline{\Delta}_1$

da alle $\underline{\Delta}_2, \dots, \underline{\Delta}_n$ den Verfaktor

$$\left(\frac{\lambda_2}{\lambda_1} \right)^k, \dots, \left(\frac{\lambda_n}{\lambda_1} \right)^k \text{ haben.}$$

$$\ll 1$$

$$\Rightarrow \frac{\underline{A}^k \underline{x}}{\|\underline{A}^k \underline{x}\|} \rightarrow \pm \underline{\Delta}_1$$

Idee: notiere $\underline{x}_k = \underline{A}^k \underline{x}$ und berechne.

$$\rho_{\underline{A}}(\underline{x}_k) = \frac{\underline{x}_k^H \underline{A} \underline{x}_k}{\underline{x}_k^H \underline{x}_k} = \frac{1}{\underline{x}_k^H \underline{x}_k} \left(\underline{x}_k^H \sum_{j=1}^n c_j \lambda_j^k \underline{\Delta}_j \right)$$

$$= \lambda_1 + O\left(\left| \frac{\lambda_2}{\lambda_1} \right|^k \right)$$

Potenzmethode:

wähle \underline{x}_0 zufällig, $\|\underline{x}_0\|=1$

für $k=1, 2, \dots$

$$w = \underline{A} \underline{x}_{k-1}$$

$$\underline{x}_k = \frac{w}{\|w\|}$$

$$\lambda_k = \underline{x}_{k-1}^H \underline{A} \underline{x}_{k-1}$$

Theorem Die Potenzmethode liefert eine Iteration, die linear gegen λ_1 konvergiert mit der Konvergenzrate $|\frac{\lambda_2}{\lambda_1}|$

Ben Falls \underline{A} normal \Rightarrow EV orthogonal \Rightarrow Fehler $O(|\frac{\lambda_2}{\lambda_1}|^{2k}) \Rightarrow$ quadratische Konvergenz!

Grundlage PageRank-Algorithmus von Google!

Beweis $\underline{A} = \underline{A}^H \Rightarrow$ ONB von EV $\underline{u}_1, \dots, \underline{u}_n$

$$\underline{U}^H \underline{A} \underline{U} = \text{diag}(\lambda_1, \dots, \lambda_n) = \underline{\Lambda}$$

$$\underline{x} = \sum_{j=1}^n c_j \underline{u}_j = \underline{U} \underline{c} \text{ mit } \underline{U} \text{ unitär.}$$

$$\frac{\underline{x}^H \underline{A} \underline{x}}{\underline{x}^H \underline{x}} = \frac{\underline{c}^H \underline{U}^H \underline{A} \underline{U} \underline{c}}{\underline{c}^H \underline{U}^H \underline{U} \underline{c}} = \frac{\underline{c}^H \underline{\Lambda} \underline{c}}{\underline{c}^H \underline{c}} = \frac{\lambda_1 |c_1|^2 + \lambda_2 |c_2|^2 + \dots + \lambda_n |c_n|^2}{|c_1|^2 + \dots + |c_n|^2}$$

$$= \lambda_1 + B \left| \frac{\lambda_2}{\lambda_1} \right|^{2k} + \dots$$

Ziel: Finde das kleinste EW

Annahme: \underline{A} invertierbar

$$\underline{A}^{-1} \mid \underline{A} \underline{x} = \lambda \underline{x} \Leftrightarrow \underline{x} = \lambda \underline{A}^{-1} \underline{x} \Leftrightarrow \frac{1}{\lambda} \underline{x} = \underline{A}^{-1} \underline{x}$$

λ_n = betrags kleinste EW von $\underline{A} \Leftrightarrow$

$\frac{1}{\lambda_n}$ = betrags grösste EW von \underline{A}^{-1}

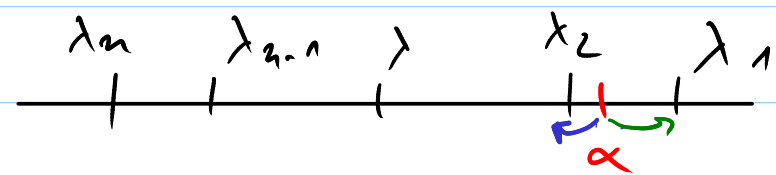
inverse Potenzmethode = Potenzmethode für \underline{A}^{-1}

Ben Wir berechnen nicht \underline{A}^{-1} sondern nur eine LU-Zerlegung von \underline{A}

am besten mit einem Alg., der die Struktur der Matrix \underline{A} beibehält, falls möglich.

Dann löse LGS mit Matrizen $\underline{L}, \underline{U}$ um $\underline{A}^{-1} \underline{x}$ zu implementieren.

Ziel: Gegeben $\alpha \in \mathbb{C}$, finde $\in W$ nah an α



$$|\alpha - \lambda| = \min \{ |\alpha - \mu| \text{ mit } \mu \in W \text{ von } \underline{A} \}$$

$\frac{1}{|\lambda_2 - \alpha|}$ gross, $\frac{1}{|\lambda_1 - \alpha|}$, $\frac{1}{|\lambda_3 - \alpha|}$ klein

Bez: shifted inverse iteration
ist schneller wenn α nah an einem $\in W$ ist.

$$\underline{A} \underline{x} = \lambda \underline{x} \Leftrightarrow \underline{A} \underline{x} - \alpha \underline{I} \underline{x} = \lambda \underline{x} - \alpha \underline{x}$$

Shift

Idee wähle Shift α adaptiv
(in jedem Iterationsschritt)

$$\Leftrightarrow (\underline{A} - \alpha \underline{I}) \underline{x} = (\lambda - \alpha) \underline{x}$$

z.B. $\alpha = \rho_{\underline{A}}(\underline{x}^{k-1})$ im k -ten Schritt.

$$\Leftrightarrow \frac{1}{\lambda - \alpha} \underline{x} = (\underline{A} - \alpha \underline{I})^{-1} \underline{x}$$

\Rightarrow beschleunigte Konvergenz

Potenzmethode für $(\underline{A} - \alpha \underline{I})^{-1} \Rightarrow \frac{1}{|\lambda - \alpha|} \max$

Rayleigh-Quotienten-Iteration (RQI)

\hookrightarrow "shifted inverse iteration"
 $\Rightarrow |\lambda - \alpha| \min.$

Ben Wir brauchen einen guten Startwert
z.B. für RQI einige Schritte von LT.

\Rightarrow Konvergenzordnung 3

Preis: für jedes neue α brauchen wir eine neue ZU-Zerlegung: $\underline{A} - \alpha \underline{I}$.

Beh $\underline{A} = \underline{A}^H$, $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$; \underline{x} beliebig
 $\underline{x} = \underline{U} \underline{c}$

$$\Rightarrow \rho_{\underline{A}}(\underline{x}) = \frac{\lambda_1 |c_1|^2 + \dots + \lambda_n |c_n|^2}{|c_1|^2 + \dots + |c_n|^2}$$

$$\Rightarrow \lambda_1 \leq \rho_{\underline{A}}(\underline{x}) \leq \lambda_n$$

$$\lambda_1 = \min_{\underline{x} \in \mathbb{C}^n} \rho_{\underline{A}}(\underline{x}) \text{ erreicht für } \underline{c} = \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$\lambda_n = \max_{\underline{x} \in \mathbb{C}^n} \rho_{\underline{A}}(\underline{x}) \text{ erreicht für } \underline{c} = \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix}$$

Beh $\underline{x} \in \text{span} \{ \underline{u}_1 \}^\perp \Rightarrow c_1 = 0$
 $\underline{x} = c_2 \underline{u}_2 + \dots + c_n \underline{u}_n$

$$\Rightarrow \rho_{\underline{A}}(\underline{x}) = \frac{\lambda_2 |c_2|^2 + \dots + \lambda_n |c_n|^2}{|c_2|^2 + \dots + |c_n|^2} \Rightarrow \lambda_2 \leq \rho_{\underline{A}}(\underline{x})$$

$$\lambda_2 = \min_{\substack{\rho_{\underline{A}}(\underline{x}) \\ \underline{x} \in \text{span} \{ \underline{u}_1 \}^\perp}} \text{ erreicht für } \underline{c} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$\lambda_{n-1} = \max_{\substack{\rho_{\underline{A}}(\underline{x}) \\ \underline{x} \in \text{span} \{ \underline{u}_n \}^\perp}}$$

und so weiter.

$$\lambda_k = \min_{\substack{\rho_{\underline{A}}(\underline{x}) \\ \underline{x} \in \text{span} \{ \underline{u}_1, \dots, \underline{u}_{k-1} \}^\perp}} = \max_{\substack{\rho_{\underline{A}}(\underline{c}) \\ \underline{c} \in \text{span} \{ \underline{u}_1, \dots, \underline{u}_k \}}}$$

Theorem (Courant-Fisher)

$$\lambda_k = \min_{\substack{U \\ \dim U = k}} \max_{x \in U} \underbrace{S_A(x)} =$$

$$= \max_{\substack{U \\ \dim U = n-k+1}} \min_{x \in U} \underbrace{S_A(x)}$$

Bem $\underline{x} = A^k \underline{x}, \underline{y} = A^k \underline{y}$ mit orthogonalisierung liefert
 $\underline{x}_k \rightarrow \underline{\rho}_1, \underline{y}_k \rightarrow \underline{\rho}_2$

2. Möglichkeiten das zu verallgemeinern

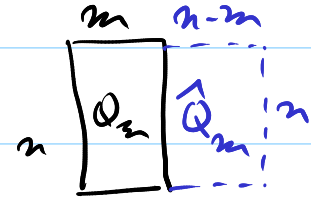
- mit Orthogonalen Transformationen (H-Spiegelungen) ⇒ "QR"-Algorithmus
- Gram-Schmidt orthogonalisierung

Theorem (Cauchy)

$$\lambda_1 < \dots < \lambda_n \in \mathbb{W} \text{ von } \underline{A} \in \mathbb{C}^{n \times n}$$

$$\underline{A} = \begin{bmatrix} \underline{H} & \underline{B}^H \\ \underline{B} & \underline{R} \end{bmatrix} \text{ mit } \underline{H} \in \mathbb{C}^{m \times m} \text{ mit } \mathbb{W} \theta_1 < \dots < \theta_m \text{ (} m < n \text{)}$$

Dann $\lambda_k \leq \theta_k \leq \lambda_{k+n-m}$

Idee $\underline{Q}_m \in \mathbb{C}^{n \times m}$  $= \underline{Q} \in \mathbb{C}^{n \times n}$ unitäre Matrix.

mit $\underline{Q}_m^H \underline{Q}_m = \underline{I}_m$

erweitere \underline{Q}_m auf ONB in \mathbb{C}^n $m \times m$

$$\underline{Q}^H \underline{A} \underline{Q} = \begin{bmatrix} \underline{Q}_m^H \underline{A} \underline{Q}_m & \underline{Q}_m^H \underline{A} \hat{\underline{Q}}_m \\ \hat{\underline{Q}}_m^H \underline{A} \underline{Q}_m & \hat{\underline{Q}}_m^H \underline{A} \hat{\underline{Q}}_m \end{bmatrix}$$

hat diese $\mathbb{W} \in \mathbb{W}$ wie \underline{A}

$$\theta_1 < \theta_2 < \dots < \theta_m \in \mathbb{W} \text{ von } \underline{Q}_m^H \underline{A} \underline{Q}_m$$

$$\lambda_1 < \theta_1 < \lambda_2 < \theta_2 < \dots < \lambda_k < \theta_k < \lambda_{k+n-m} < \dots < \lambda_m < \theta_m < \lambda_n$$

Idea: Für $m \ll n$, wähle \underline{Q}_m so dass

Bild $\underline{Q}_m \approx \text{span} \{ \underline{u}_1, \dots, \underline{u}_m \}$

\rightarrow EV von \underline{A} zu $\lambda_1, \dots, \lambda_m$

\Rightarrow gute Approximation $\sigma_k \approx \lambda_k$ für $k=1, \dots, m$

Wie? modifiziertes Gram-Schmidt für

$\{ \underline{v}, \underline{A}\underline{v}, \underline{A}^2\underline{v}, \dots, \underline{A}^{m-1}\underline{v} \}$

= Krylov-Verfahren Arnoldi/Lanczos

Bem für "kleinere" Matrizen: eig (QR-Alg) gut
"grössere" : eig zu langsam

: eig zu langsam
nutzt Struktur
von \underline{A} nicht!

Krylov: geeignet für grosse, dünnbesetzte
Matrizen.

" \underline{A} mal \underline{x} "

§ 9.3 Krylov-Verfahren

Def Sei $0 \neq \underline{z} \in \mathbb{C}^n$, $\underline{A} \in \mathbb{C}^{n \times n}$

$\mathcal{K}_l(\underline{A}, \underline{z}) = \text{span} \{ \underline{z}, \underline{A}\underline{z}, \underline{A}^2\underline{z}, \dots, \underline{A}^{l-1}\underline{z} \}$

$= \{ p(\underline{A})\underline{z} ; p = \text{Polynom, von Grad} \leq l-1 \}$

Krylov-Raum.

Suche eine ONB in $\mathcal{K}_l(\underline{A}, \underline{z}) \subseteq \mathbb{C}^n$

$\mathcal{K}_1 = \text{span} \{ \underline{z} \} \subset \text{span} \{ \underline{z}, \underline{A}\underline{z} \} = \mathcal{K}_2 \subset \mathcal{K}_3 \subset \dots \subset \mathcal{K}_l$

Iterativ: gegeben $\underline{v}_1, \underline{v}_2, \dots, \underline{v}_l$ ONB in $\mathcal{K}_l(\underline{A}, \underline{z})$

mit $\text{span} \{ \underline{v}_1, \dots, \underline{v}_j \} = \mathcal{K}_j(\underline{A}, \underline{z})$

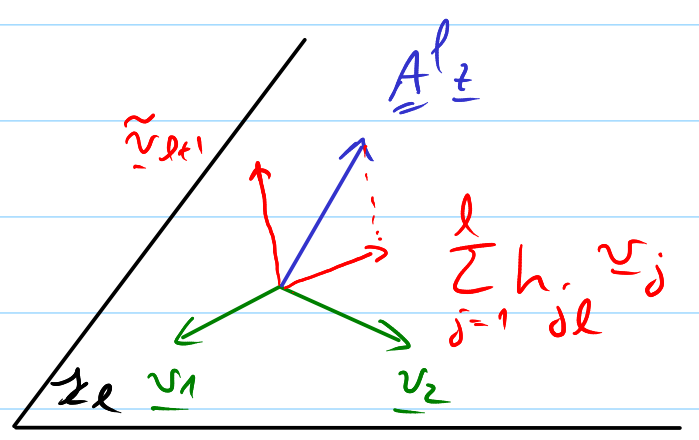
für $j=1, 2, \dots, l$

baue $\underline{v}_1, \underline{v}_2, \dots, \underline{v}_l, \underline{v}_{l+1}$ ONB in $\mathcal{K}_{l+1}(\underline{A}, \underline{z})$

$\text{span} \{ \underline{z}, \underline{A}\underline{z}, \dots, \underline{A}^{l-1}\underline{z}, \underline{A}^l\underline{z} \}$

Entweder $\underline{A}^l \underline{z} \in \mathcal{Z}_l(\underline{A}, \underline{z})$ d.h. $\underline{A}^l \underline{z}$ linear abhängig von $\underline{z}, \underline{A}\underline{z}, \dots, \underline{A}^{l-1}\underline{z}$

oder $\underline{A}^l \underline{z} \notin \mathcal{Z}_l(\underline{A}, \underline{z}) \Rightarrow \underline{A}^l \underline{z} \in \mathcal{Z}_{l+1} \setminus \mathcal{Z}_l$
dann \underline{v}_{l+1} aus (modifizierten) Gram-Schmidt



$$\tilde{\underline{v}}_{l+1} = \underline{A}^l \underline{z} - \sum_{j=1}^l h_{jl} \underline{v}_j$$
$$\underline{v}_{l+1} = \frac{1}{\|\tilde{\underline{v}}_{l+1}\|} \tilde{\underline{v}}_{l+1}$$

Dabei sind $h_{je} = \underline{v}_j^H \underline{A} \underline{v}_e$ da $\underline{v}_1, \dots, \underline{v}_e$ ONB in \mathcal{Z}_e .

Algorithmus (Arnoldi Prozess)

$\underline{z} =$ beliebig $\neq 0$
 $\underline{v}_1 = \frac{1}{\|\underline{z}\|} \underline{z}$

für $l = 1, 2, \dots, k-1$

$\underline{v} = \underline{A} \underline{v}_l$

für $j = 1, 2, \dots, l$

$h_{je} = \underline{v}_j^H \underline{v}$

mod.! $\underline{v} = \underline{v} - h_{je} \underline{v}_j$

$h_{l+1, l} = \|\underline{v}\|$

$\underline{v}_{l+1} = \underline{v} / h_{l+1, l}$

	$l=1$	$l=2$	$l=3$
h_{11}	h_{12}	h_{13}	
	h_{22}	h_{23}	
		h_{33}	
h_{21}	h_{23}	h_{34}	

es entsteht eine "Hessenberg" Matrix

x	x	x	...
x	x	x	
0	x	x	
0	0	x	
...	
0	0	0	

Bew Falls $h_{l+1,l} = 0 \Rightarrow$ Abbruch der Iteration

Fall $\underline{A} \underline{v}_l \in \mathcal{Z}_l(\underline{A}, \underline{z})$

$$\underline{V}_l = [\underline{v}_1 \ \underline{v}_2 \ \dots \ \underline{v}_l] \in \mathbb{C}^{n \times l}$$

$$\underline{H}_l = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & h_{32} & h_{33} \\ \hline 0 & 0 & h_{43} \end{bmatrix} \in \mathbb{C}^{l+1, l}$$

$\underline{H}_l \in \mathbb{C}^{l \times l}$

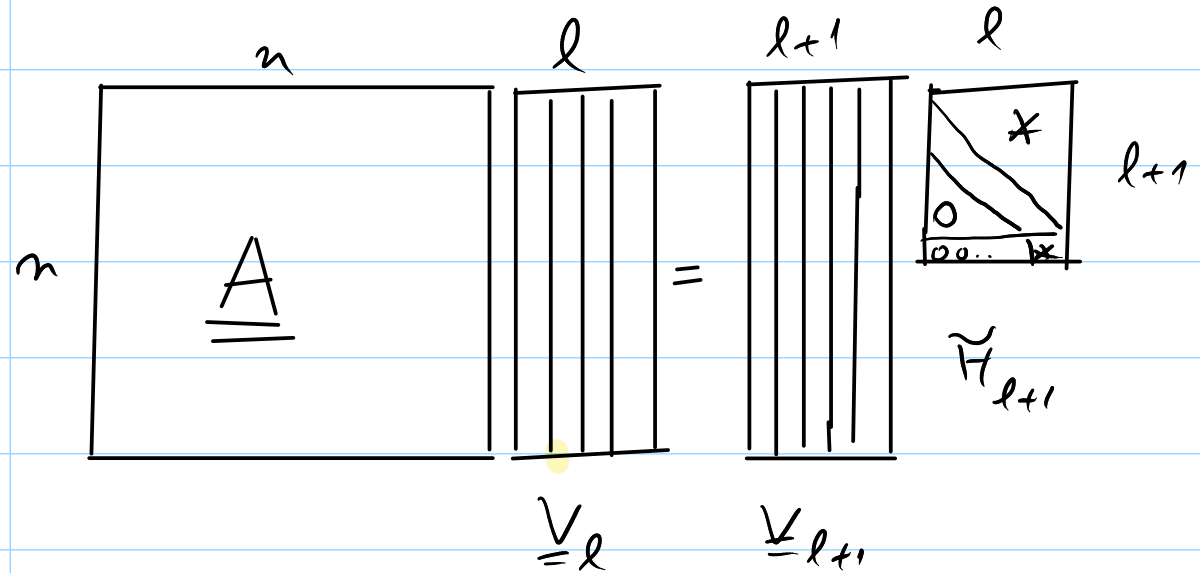
$$\underline{H}_l = \begin{bmatrix} \diagup & & \\ & \diagup & \\ 0 & & \diagdown \end{bmatrix} \text{ obere Hessenberg matrix}$$

Aus Konstruktion,

$$\underline{A} \underline{v}_k = h_{k+1,k} \underline{v}_{k+1} + \sum_{j=1}^k h_{jk} \underline{v}_j \quad \text{für } k=1, 2, \dots, l$$

$$\underline{A} \underline{V}_l = \underline{V}_{l+1} \underline{H}_l = \begin{bmatrix} \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix} + \underline{V}_l \underline{H}_l$$

$l < n$

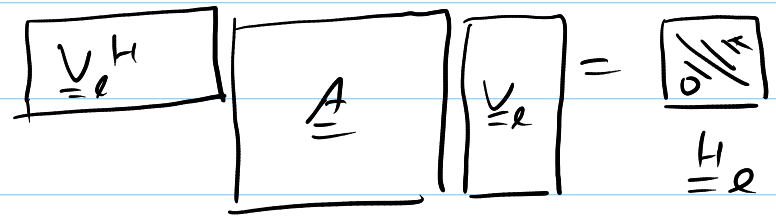


Bew 1) $\underline{V}_l^H \underline{V}_l = \underline{I}_l$

$$\underline{V}_l^H \underline{V}_l = \underline{I}_l$$

2) $\underline{V}_l^H \underline{A} \underline{V}_l = \underline{V}_l^H \begin{bmatrix} \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix} + \underline{V}_l^H \underline{V}_l \underline{H}_l = \underline{H}_l$

$$\underline{V}_l^H \underline{A} \underline{V}_l = \underline{H}_l$$



3) falls $h_{l+1,l} = 0 \Rightarrow \mathcal{K}_{l+1} = \mathcal{K}_l$ und

(statt variable Länge l)

$$\underline{v}_{l+1} = \underline{A} \underline{v}_l - h_{ll} \underline{v}_l - h_{l-1,l} \underline{v}_{l-1}$$

\Rightarrow Lanczos-Verfahren. $O(nk)$

$$\underline{A} \underline{V}_l = \underline{V}_l \underline{H}_l \Rightarrow \underline{V}_l^H \underline{A} \underline{V}_l = \underline{H}_l$$

Für allgemeine Matrizen: Arnoldi $O(nk^2)$

$k =$ Anzahl Krylov-Schritte

Allgemeiner Name: Krylov-Raum-Verfahren

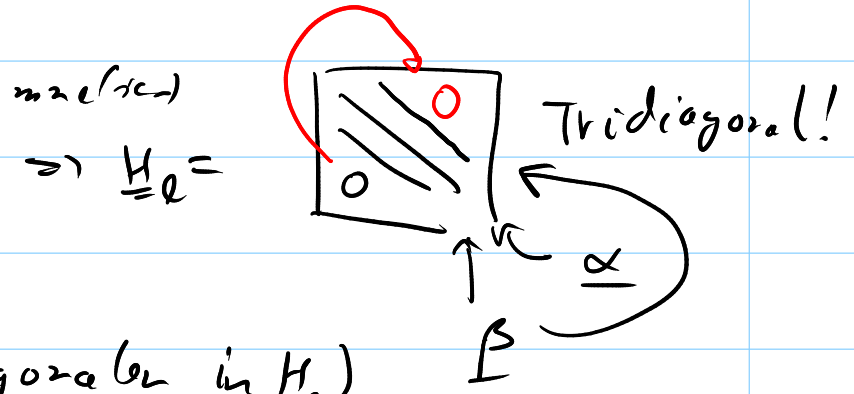
4) falls $\underline{A}^H = \underline{A}$ (\underline{A} Hermite symmetrische)

$$\underline{H}_l = \underline{V}_l^H \underline{A} \underline{V}_l \Rightarrow \underline{H}_l^H = \underline{V}_l^H \underline{A}^H \underline{V}_l = \underline{V}_l^H \underline{A} \underline{V}_l = \underline{H}_l$$

Theorem

Falls $h_{l+1,l} = 0$ und $h_{j+1,j} \neq 0$ für $j=1, \dots, l-1$, dann

$\Rightarrow \underline{H}_l$ auch Hermite symmetrisch



(1) jeder EW von \underline{H}_l ist auch EW von \underline{A}

(2) falls \underline{A} regulär, dann gibt es $\underline{y} \in \mathbb{C}^l$ so dass

$$\underline{A} \underline{x} = \underline{b} \text{ mit } \underline{x} = \underline{V}_l \underline{y}$$

\Rightarrow es reicht α, β (=Diagonale in \underline{H}_l)

zu speichern / berechnen

\Rightarrow innere Schleife in Arnoldi hat konstante (l) Länge.

Beweis (1) Sei $\lambda \in \mathbb{K}$ von \underline{H}_ℓ zu $\underline{v} \in V$ $\underline{z} \neq \underline{0}$

$$\underline{H}_\ell \underline{y} = \lambda \underline{y} \Rightarrow \underline{V}_\ell \underline{H}_\ell \underline{y} = \lambda \underline{V}_\ell \underline{y}$$

\underline{V}_ℓ

Iteration bricht ab $\Rightarrow \underline{A} \underline{V}_\ell = \underline{V}_\ell \underline{H}_\ell \Rightarrow$

$$\Rightarrow \underline{A} \underline{V}_\ell \underline{y} = \underline{V}_\ell \underline{H}_\ell \underline{y} = \lambda \underline{V}_\ell \underline{y} \Rightarrow$$

$\lambda \in \mathbb{K}$ von \underline{A} zu $\underline{v} \in V$ $\underline{V}_\ell \underline{y}$

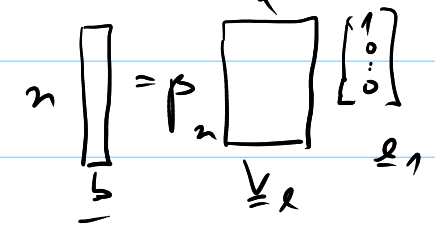
(2) Gegeben: \underline{b} , \underline{A} regulär

\Downarrow
 $\underline{0}$ ist kein EW von \underline{A}

\Downarrow
 $\underline{0}$ ist kein EW von \underline{H}_ℓ

Sei \underline{V}_ℓ gebaut für $\mathcal{Z}_\ell(\underline{A}, \underline{b})$

$$\underline{b} = \beta \underline{v}_1 = \underline{V}_\ell \beta \underline{e}_1 \quad \text{mit } \underline{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



Betrachte LGS $\ell \times \ell$

$$\underline{H}_\ell \underline{y} = \beta \underline{e}_1 \quad \text{hat Lösung } \underline{y} \in \mathbb{C}^\ell$$

Somit $\underline{A} \underline{V}_\ell \underline{y} = \underline{V}_\ell \underline{H}_\ell \underline{y} = \underline{V}_\ell \beta \underline{e}_1 = \underline{b}$

$\Rightarrow \underline{x} = \underline{V}_\ell \underline{y}$ ist die Lösung von $\underline{A} \underline{x} = \underline{b}$.

Ben

$$\underline{H}_\ell \in \mathbb{C}^{\ell \times \ell}, \ell \ll n$$

obere Hessenberg / falls $\underline{A}^H = \underline{A} \Rightarrow$ tridiagonal.

$\text{eig}(\underline{H}_\ell)$ ist sehr schnell ($\ell \ll n$) \Rightarrow anwendbar.

Theorie \rightarrow Erzeugende $\{\underline{v}_1, \dots, \underline{v}_\ell\}$ zu ONB in \mathbb{C}^n

$$\underline{Q} = \begin{bmatrix} \underline{v}_1 & \underline{v}_2 & \dots & \underline{v}_\ell \end{bmatrix} \stackrel{\hat{=}}{=} \underline{H}_\ell$$

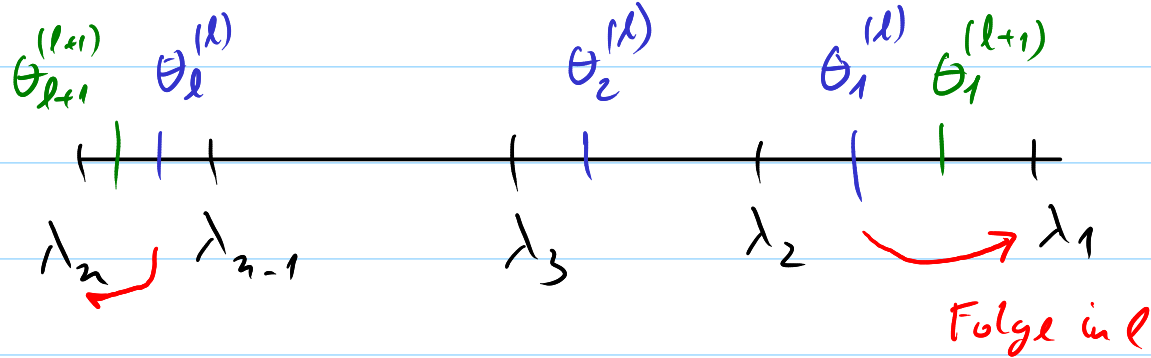
$$\underline{Q}^H \underline{A} \underline{Q} = \begin{bmatrix} \underline{V}_\ell^H \underline{A} \underline{V}_\ell & \times \\ \times & \times \end{bmatrix}$$

Cauchy.
 $\lambda_k < \sigma_k < \lambda_{k+n-\ell}$
 \uparrow
EW von \underline{H}_ℓ

Theorem Sei $\underline{A} \in \mathbb{C}^{n \times n}$ Hermite symmetrisch,
 und $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \in \mathbb{W}$ von \underline{A}

$$\theta_1^{(l)} \geq \theta_2^{(l)} \geq \dots \geq \theta_l^{(l)} \in \mathbb{W} \text{ von } \underline{H}_l = \underline{V}_l^H \underline{A} \underline{V}_l$$

für $l=1,2,3,\dots$ $\underline{H}_{l+1} = \underline{V}_{l+1}^H \underline{A} \underline{V}_{l+1}$



Dann gelten für $1 \leq j \leq l$ die Ungleichungsketten:

$$\theta_j^{(l)} \leq \theta_j^{(l+1)} \leq \lambda_j$$

$$\lambda_{n-j+1} \leq \theta_{l+1-j+1}^{(l+1)} \leq \theta_{l-j+1}^{(l)}$$

Bem $l=n \Rightarrow$ Iteration bricht automatisch ab
 $\Rightarrow \in \mathbb{W} \lambda_1, \lambda_2, \dots, \lambda_n$

Skript \rightarrow einfache Implementierungen. Arnoldi
 LGS OS

ARPACK \rightarrow eigvals [eigs]

Tricks anwendbar mit Krylov-Verfahren

\rightarrow Shift $\sigma \in \mathbb{W} \quad \underline{A} + \sigma \underline{I}$

\rightarrow Shift & inverso: $(\underline{A} - \sigma \underline{I})^{-1}$

\rightarrow Cayley Transformation: $(\underline{A} - \sigma \underline{I})^{-1} (\underline{A} + \tau \underline{I})$

\rightarrow Folding $(\underline{A} + \sigma \underline{I})^2$

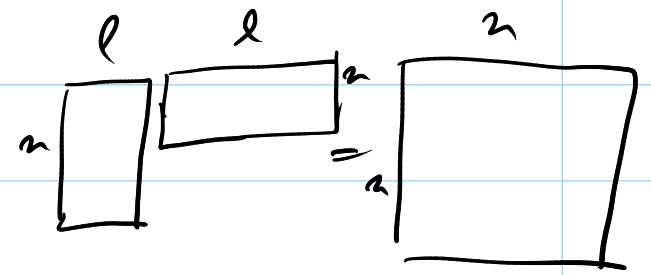
"LGS Lösen"

$\mathcal{K}_l = \text{Bild } \{ \underline{v}_1, \dots, \underline{v}_l \} \subset \mathbb{C}^n$
ONB

Projektor (orthogonaler Projektor)

$$P : \mathbb{C}^n \rightarrow \mathcal{K}_l$$

als Matrix $P_{\mathcal{K}_l} = \underline{V}_l \underline{V}_l^H$



$$P_{\mathcal{K}_l} \underline{A} = \underline{V}_l \underline{V}_l^H \underline{A}$$

$\underline{u} \in \mathcal{K}_l \Rightarrow \underline{u} = c_1 \underline{v}_1 + c_2 \underline{v}_2 + \dots + c_l \underline{v}_l \in \mathcal{K}_l \subset \mathbb{C}^n$
 $\underline{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_l \end{bmatrix}$
"Basiswechsel"

$$P_{\mathcal{K}_l} \underline{A} \underline{u} = \underline{V}_l \underline{V}_l^H \underline{A} \underline{u} = \underline{V}_l \underbrace{\underline{V}_l^H \underline{A} \underline{V}_l}_{\underline{H}} \underline{c} =$$

$= \underline{V}_l (\underline{H} \underline{c}) \Rightarrow \underline{H} \underline{c}$ gibt die Koeffizienten in der Basis $\{ \underline{v}_1, \dots, \underline{v}_l \}$ von $P_{\mathcal{K}_l} \underline{A} \underline{u}$

$\Rightarrow P_{\mathcal{K}_l} \underline{A}$ wird auf \mathcal{K}_l durch \underline{H}_l dargestellt

$$\underline{A} : \mathbb{C}^n \rightarrow \mathbb{C}^n$$

$\underline{U} \quad \underline{V}$

$$P_{\mathcal{K}_l} \underline{A} : \mathcal{K}_l \rightarrow \mathcal{K}_l$$

↳ Raum kleiner Dimension.

Standard Idee aus der Approximation:

ersetze den unbequemen Raum grosser / unendlicher Dimension durch bequemen Raum kleiner Dimension

$$\underline{A} \text{ durch } P_{\mathcal{K}_l} \underline{A}$$

und man schreibt es in der ONB von \mathcal{K}_l

$$\underline{H}_l \in \mathbb{C}^{l \times l} \text{ mit } l \ll n \text{ und berechne die EW von } \underline{H}_l \text{ mit eig. (QR-Iteration)}$$

Arnoldi-Verfahren sind gut nutzbar wenn

$A^H = A$ und A dünn besetzt.

Bem

Numerik \rightarrow

Schur-Zerlegung

$A = U T U^H$
uniforme
Lobere Dreiecksmatrix

Für nicht-symmetrische Matrizen ist es

Theorie \rightarrow

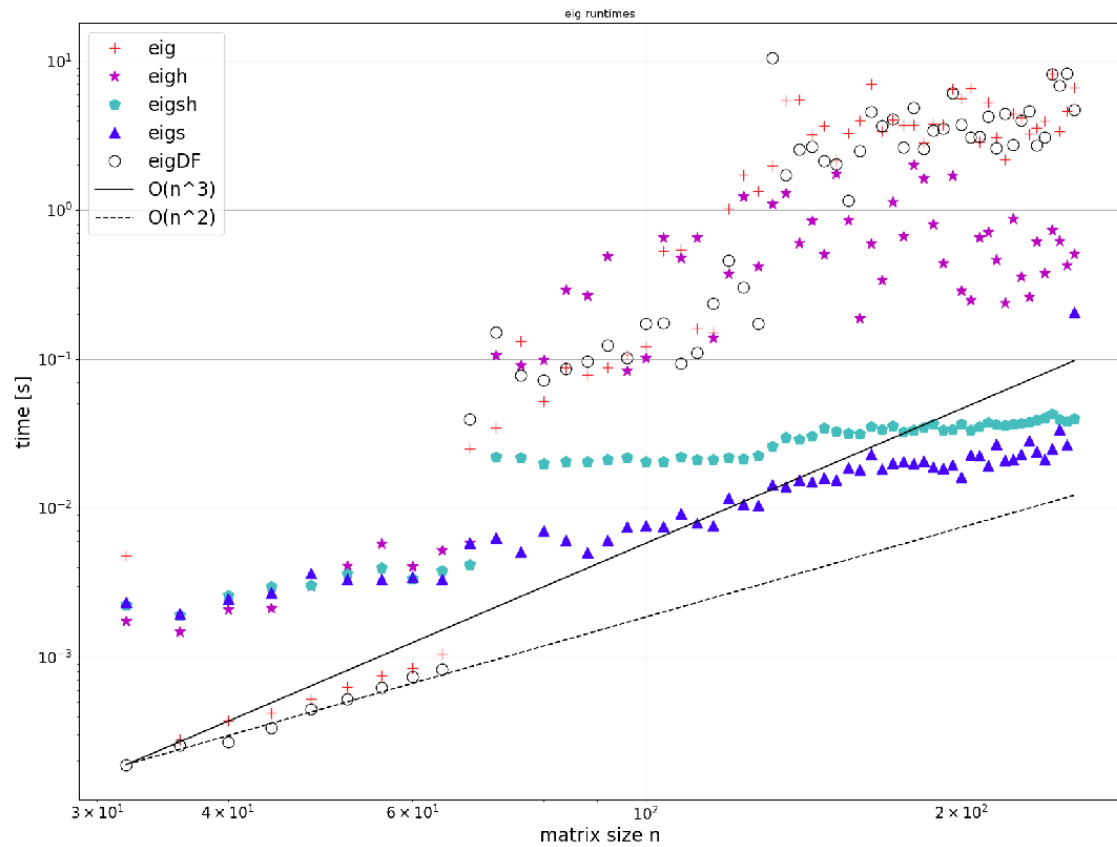
Jordan-Zerlegung.

deutlich problematischer

\hookrightarrow numerisch sehr schlecht konditioniert

Wenn A voll besetzt ist kann das zu teuer werden.

Darum ist eig immer noch nützlich!



§ 10 Lineare Anfangswertprobleme

1. Fall $\begin{cases} \dot{\underline{y}} = \underline{A} \underline{y} & \text{Falls } \underline{A} \text{ diagonalisierbar} \\ \underline{y}(0) = \underline{y}_0 & \underline{A} = \underline{S}^{-1} \underline{D} \underline{S} \end{cases}$

Variablenwechsel $\hat{\underline{y}} = \underline{S}^{-1} \underline{y} \Rightarrow$ entkoppeln

$$\begin{cases} \dot{\hat{y}}_1 = \lambda_1 \hat{y}_1 \\ \vdots \\ \dot{\hat{y}}_d = \lambda_d \hat{y}_d \end{cases} \Rightarrow \hat{y}_i(t) = (\underline{S}^{-1} \underline{y}_0)_i e^{\lambda_i t} \text{ f\u00fcr } t \in \mathbb{R}$$

$$\underline{y}(t) = \underline{S} \begin{bmatrix} e^{\lambda_1 t} & & \\ & \ddots & \\ & & e^{\lambda_d t} \end{bmatrix} \underline{S}^{-1} \underline{y}_0$$

ok f\u00fcr kleines d oder exact / analytisch.
sonst instabil \hookrightarrow

F\u00fcr $d=5, \dots, 50, 100$

$$\underline{y}(t) = e^{\underline{A}t} \underline{y}_0 \quad \text{Pad\u00e9-Approximation}$$

$$\hookrightarrow \exp_m(\underline{A}t) \underline{y}_0$$

d gross, \underline{A} d\u00fcnn besetzt: Krylov-Verfahren.

Krylov: f\u00fcr \underline{A} gibt es $\underline{V} \in \mathbb{C}^{d \times m}$ mit orthonormalen Spalten.

$$\underline{V}_m^H \underline{A} \underline{V}_m = \underline{H}_m \quad m \times m \text{ mit } m \ll d$$

\hookrightarrow obere Hessenberg Matrix.

$$\underline{y}(t) \in \mathbb{R}^d$$

$\{ \cup \}$

$$\mathcal{K}_m(\underline{A}, \underline{y}_0) = \text{span} \{ \underline{y}_0, \underline{A} \underline{y}_0, \dots, \underline{A}^{m-1} \underline{y}_0 \}$$

$= \text{span} \{ \underline{v}_1, \dots, \underline{v}_m \}$ \leftarrow Spalten von \underline{V} ORB.

$$\Rightarrow \underline{\dot{c}}(t) = \underline{H}_m \underline{c}(t)$$

$$\underline{y}(0) = \underline{y}_0 \Rightarrow \underline{c}(0) = \|\underline{y}_0\| \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^m$$

mit $m \ll d \Rightarrow \exp_m(\underline{H}_m)$ Poole ist günstig!

$$\Rightarrow \underline{c}(t) = \exp_m(\underline{H}_m t) \underline{c}(0)$$

$$\Rightarrow \underline{u}_m(t) = \sum_{k=1}^m c_k(t) \underline{v}_k = \|\underline{y}_0\| \underline{V}_m e^{\underline{H}_m t} \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix}$$

2. Fall $\underline{\dot{y}}(t) = \underline{A} \underline{y}(t) + \underline{g}(t)$ inhomogener Fall

$$\underline{y}(t) = e^{\underline{A}(t-t_0)} \underline{y}_0 + \int_{t_0}^t e^{\underline{A}(t-\tau)} \underline{g}(\tau) d\tau$$

Formel der Variation der Konstanten

3. Fall

$$\begin{cases} \underline{\dot{y}}(t) = \underline{A}(t) \underline{y}(t) \\ \underline{y}(t_0) = \underline{y}_0 \end{cases}$$

Unter bestimmten Voraussetzungen

$$\underline{y}(t) = e^{\underline{\Omega}(t, t_0)} \underline{y}_0$$

$$\underline{\Omega} = \sum_{k=1}^{\infty} \underline{\Omega}_k$$

Magnus Entwicklung

$$\underline{\Omega}_1 = \int_{t_0}^t \underline{A}(\tau_1) d\tau_1 \quad ([A, B] = \underline{AB} - \underline{BA} \text{ Kommutator})$$

$$\underline{\Omega}_2 = \frac{1}{2} \int_{t_0}^t \int_{t_0}^{\tau_1} [\underline{A}(\tau_1), \underline{A}(\tau_2)] d\tau_2 d\tau_1$$

$$\underline{\Omega}_3 = \frac{1}{12} \int_{t_0}^t \int_{t_0}^{\tau_1} \int_{t_0}^{\tau_2} \left([[\underline{A}(\tau_1), \underline{A}(\tau_2)], \underline{A}(\tau_3)] + [\underline{A}(\tau_1), [\underline{A}(\tau_2), \underline{A}(\tau_3)]] \right) d\tau_3 d\tau_2 d\tau_1$$

Idee Statt $\dot{\underline{y}} = \underline{A}(t) \underline{y}$ löse $\dot{\underline{\hat{y}}} = \underline{\hat{A}}(t) \underline{\hat{y}}$

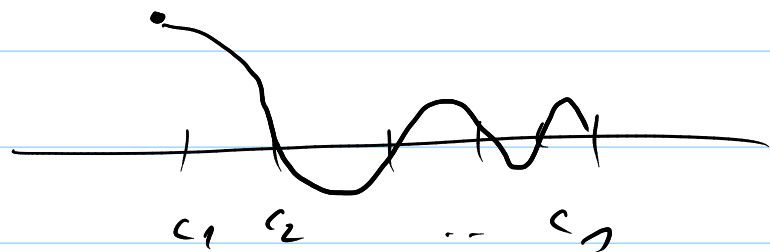
wobei
$$\underline{\hat{A}}(t) = \sum_{i=1}^s l_i(t) \underline{A}(t_n + c_i h)$$

$l_i(t)$ = Lagrange-Polynom in $t_n + c_i h$

c_i = Quadraturknoten in $[0, 1]$

$\underline{\hat{A}}(t)$ = Polynom vom Grad $(s-1)$ in t auf $[t_n, t_n+h)$

$$\underline{\hat{A}}(t_n + c_i h) = \underline{A}(t_n + c_i h) \text{ für } i = 1, 2, \dots, s$$



Dann Magnus Entwicklung für $\dot{\underline{\hat{y}}} = \underline{\hat{A}}(t) \underline{\hat{y}}$

Vorteil: Integration nur für Polynome ist \Rightarrow einfach exakt berechnen.

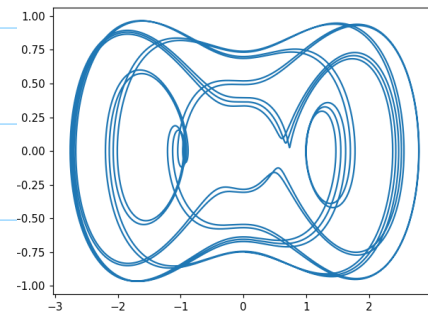
$\underline{\hat{A}}(t)$ glatt

Man kann zeigen:

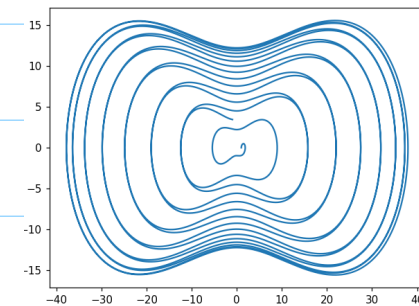
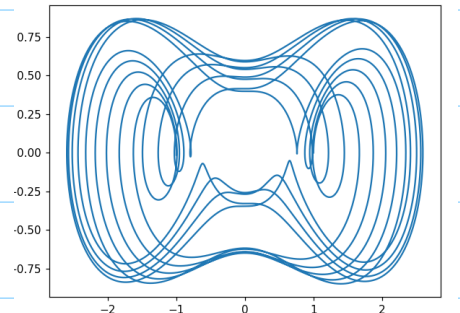
Rest in der Magnusentwicklung $\underline{\hat{\Omega}}$ ist $O(h^5)$ nach 4 Terme.

Theorem $(b_i, c_i)_{i=1, \dots, s}$ Quadraturformel der Ordnung $p \geq s$

$$\underline{y}(t_n+h) - \underline{\hat{y}}(t_n+h) = O(h^{p+1})$$



Bsp Mathieu-Gleichung.



§ 11 Exponentielle Integratoren

$$\begin{cases} \dot{\underline{y}} = \underline{f}(\underline{y}) & \text{autonom mit } \underline{f} \text{ stetig differenzierbar} \\ \underline{y}(0) = \underline{y}_0 \end{cases}$$

$$\int_0^h e^{\underline{A}(h-s)} \underline{g}(\underline{y}(s)) ds \approx \int_0^h e^{\underline{A}(h-s)} \underline{g}(\underline{y}_0) ds = h \ell(\underline{A}) \underline{g}(\underline{y}_0)$$

Notiere / Definiere:

Idee der Linearisierung $\underline{A} = \underline{Df}(\underline{y})$

$$\ell(z) = \frac{e^z - 1}{z} = \sum_{n=1}^{\infty} \frac{1}{n!} z^{n-1} = \sum_{n=0}^{\infty} \frac{z^n}{(n+1)!}$$

$$\dot{\underline{y}} = \boxed{\underline{A} \underline{y}} + \underbrace{\underline{f}(\underline{y}) - \underline{A} \underline{y}}_{\underline{g}(\underline{y})}$$

linear

Für Matrizen: $\ell(\underline{A}) = \sum_{n=0}^{\infty} \frac{1}{(n+1)!} \underline{A}^n = \underbrace{(e^{\underline{A}} - \underline{I}) \underline{A}^{-1}}$

Variation der Konstanten:

$$\underline{y}(h) = e^{\underline{A}h} \underline{y}_0 + \int_0^h e^{\underline{A}(h-s)} \underline{g}(\underline{y}(s)) ds$$

Beweis

$$\int_0^h e^{\underline{A}(h-s)} \underline{g}(\underline{y}_0) ds = \int_0^h \sum_{n=0}^{\infty} \frac{1}{n!} (\underline{A}(h-s))^n \underline{g}(\underline{y}_0) ds =$$

↑ ↓
Umtauschen $= \int_0^h (h-s)^n ds \underline{g}(\underline{y}_0)$

Ersetze $\underline{y}(s)$ durch $\underline{y}_0 \Rightarrow$ "Quadratur" / Approximation.

$$= \sum_{n=0}^{\infty} \frac{1}{n!} \underline{A}^n \int_0^h (h-s)^n ds \underline{g}(\underline{y}_0) =$$

$$= \sum_{n=0}^{\infty} \frac{1}{n!} \frac{\partial^n}{\partial t^n} \frac{h^{n+1}}{n+1} g(\underline{y}_0) = h \underbrace{\sum_{n=0}^{\infty} \frac{1}{(n+1)!} \partial^{n+1} h^n}_{f(h)} g(\underline{y}_0) = h f(h) g(\underline{y}_0)$$

$$\Rightarrow \underline{y}(h) \approx \underline{y}_0 + h f(h) \underline{f}(\underline{y}_0)$$

exponentielles Eulerverfahren.

$$\underline{\partial} = \underline{D}f(\underline{y}_0)$$

$$f(h \underline{\partial}) = \left(e^{h \underline{\partial}} - \underline{I} \right) (h \underline{\partial})^{-1}$$

teuer für grosses d.

Bem Definition von $\underline{g}(\underline{y}_0) = \underline{f}(\underline{y}_0) - \underline{\partial} \underline{y}_0$

$$h f(h) \underline{g}(\underline{y}_0) = h f(h) \underline{f}(\underline{y}_0) - h f(h) \underline{\partial} \underline{y}_0$$

Nehme $h f(h) \underline{\partial} \underline{y}_0 = h \sum_{n=0}^{\infty} \frac{1}{(n+1)!} \partial^{n+1} h^n \underline{\partial} \underline{y}_0 = e \underline{\partial} \underline{y}_0 - \underline{y}_0$

$$\Rightarrow h f(h) \underline{g}(\underline{y}_0) = h f(h) \underline{f}(\underline{y}_0) - e \underline{\partial} \underline{y}_0 + \underline{y}_0$$

Bem Für grosses d kann man das Krylov-Verfahren für $e^{h \underline{\partial}}$ verwenden!

$$f(\underline{A}) \underline{b} = \underline{V}_m f(\underline{H}_m) \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix}$$

aus Krylov-Verfahren mit $\mathcal{K}_m(\underline{A}, \underline{b})$

Somit erhalten wir zusammen:

$$\underline{y}(h) \approx e \underline{y}_0 + h f(h) \underline{f}(\underline{y}_0) - e \underline{\partial} \underline{y}_0 + \underline{y}_0$$

Bem Stabilitätsfunktion. $S(z) = e^z$

\Rightarrow das ideale Stabilitätsgebiet!

da exakt für $\dot{\underline{y}} = \underline{A} \underline{y} + \underline{y}_{konstant}$

Verallgemeinerung: exponentielle RK-Verfahren.

$$\underline{\partial} = D\underline{f}(\underline{x})$$

semi-implizite Euler: $\underline{y}_1 = \underline{y}_0 + (\underline{I} - h\underline{\partial})^{-1} h\underline{f}(\underline{y}_0)$

exponentielle Euler: $\underline{y}_1 = \underline{y}_0 + \ell(h\underline{\partial}) h\underline{f}(\underline{y}_0)$

Idee: ersetze $\frac{1}{1-z}$ durch $\ell(z) = \frac{e^z - 1}{z}$ in Row

$$\left\{ \begin{aligned} \underline{k}_i &= \ell(\alpha h \underline{\partial}) \left(\underline{f}(\underline{u}_i) + h \underline{\partial} \sum_{j=1}^{i-1} \alpha_{ij} \underline{k}_j \right) \\ \underline{u}_i &= \underline{y}_0 + h \sum_{j=1}^{i-1} \alpha_{ij} \underline{k}_j \\ \underline{y}_1 &= \underline{y}_0 + h \sum_{i=1}^1 b_i \underline{u}_i \end{aligned} \right.$$

explizite RK: $\ell(z) = 1$ und $\underline{\partial} = \underline{0}$

Row $\ell(z) = \frac{1}{1-z}$

exp RK: $\ell(z) = \frac{e^z - 1}{z}$

ODE

nicht-steife

methode

ivp-solve

explizit

RK 4,5, adaptiv

stabilität! $h < \frac{1}{\lambda}$

u. Umstände h muss klein sein.

splitting

Erhaltung. wichtig.

- Quadratur FFT
- Fourier mit Anwendung von Differentialgleichung
- Numerische LA: QR, s.v.d. → PCA.

EW, EV

↓ Arnoldi

exp.

stefo.

ivp-solve (method=)

implizite

h beliebig

Row

(RK, O2, adaptiv)

oscillationsit.

exp

h beliebig

Arnoldi

↪ Ausgleichsprobleme

↓
M.L. (A.I.)



- c1. Quadratur
 - 1.1 Grundlagen (Seite 1):
 - QFormel, Knoten, gewichte, alg. & exp. Konvergenz, Polynomiale Interpolation, Lagrange Polynome, Ordnung einer QFormel, Fehler, zusammengesetzte QFormel, MPR, TR, SR, Referenzintervale, Fehler
 - 1.2 Quadratur in \mathbb{R}^d (Seite 8)
 - 1.3 Adaptive Quadratur (Seite 9)
 - 1.4 Gauss-Quadratur (Seite 12)
- c2 Trigonometrische (Fourier) Approximation
 - 2.1 Grundlagen (Seite 15)
 - 2.2 Diskrete Fourier Transformation (Seite 17)
 - 2.3 Clenshaw-Curtis-Quadratur (Seite 26)
- c3 Einfache Verfahren fuer ODEs
 - 3.1 Lokale Linearisierung (Seite 28)
 - 3.2 Stoermer-Verlet (Seite 33)
 - 3.3 Splitting Verfahre (Seite 37)
 - 3.4 Lineare Transportgleichung (Seite 47)
- c4 Runge-Kutta Verfahren
 - 4.1 Grundidee (Seite 52)
 - 4.2 Kollokation (Seite 56)
 - 4.3 Adaptivitaet (Seite 57)
 - 4.4 Partitionierte Runge-Kutta Verfahren (Seite 58)
- c5 Nichtlineare algebraische Gleichungen
 - 5.1 Konvergenz, Fixpunktiteration (Seite 61)
 - 5.2 Newton-Verfahren (Seite 64)
 - 5.3 Anwendungen zur Optimierung:
 - BFGS, Gradienten-Verfahren (sehr kurz, Details bei DNNs) (Seite 67)
- c6 Steife Differentialgleichungen
 - 6.1 Einfuehrung (Seite 68)
 - 6.2 Stabilitaet der Runge-Kutta Verfahren (Seite 70)
 - 6.3 Linear-Implicite Einschrittverfahren, Rosenbrock-Wanner Methoden (Skript) (Seite 73)
naechstes Mal ein bisschen mehr ueber ROW schreiben
- c7 Intermezzo ueber Numerische Lineare Algebra (Seite 76)
- c8 Ausgleichsrechnung
 - 8.1 Lineare Ausgleichsrechnung: via Normalgleichung (Seite 84)
 - 8.2 Lineare Ausgleichsrechnung: via orthogonaler Transformationen (Seite 86)
 - 8.3 PCA und LSQ (Seite 90)
 - 8.4 Lineare Ausgleichsrechnung mit linearen Nebenbedingungen (Seite 108)
Newton, Gauss-Newton, Gradienten-Verfahren, GDM, ADAM
 - 8.6 Schaetzung der Parameter aus ODEs (Seite 108)
 - 8.7 DNNs und PINNs (Seite 113):
 - Backpropagation, stochastik Gradient
- c9 Eigenwerte
 - 9.1 Einfuehrung (Seite 124)
 - 9.2 Potenzmethoden (Seite 124)
 - 9.3 Krylov-Verfahren (Seite 131)