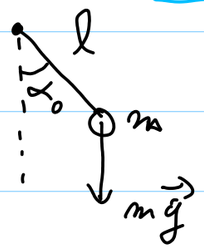
18.02.2020

§ Quadratur

§1.1. Einführung.

Bsp 1) Math. Pendulum



Periode $T = 4\sqrt{\frac{l}{g}}\, K(\sin\frac{\alpha_0}{2})$

$$K(a) = \int_0^{\pi/2} \frac{1}{\sqrt{1 - a^2 \sin^2 s}}\, ds$$

2) $$\sigma = \frac{1}{t^4} \int_0^{\infty} E(\lambda, t)\, d\lambda$$

mit $$E(\lambda, t) = \frac{c_1}{\lambda^5 \left(e^{\frac{c_2}{\lambda t}} - 1\right)}$$

Aufgabe: $f : [a,b] \to \mathbb{R}$

Berechne $$J = \int_a^b f(x)\, dx \approx Q(f,a,b) = \sum_{j=1}^{p} w_j f(x_j)$$

Gewichte, Knoten

Ziel: Wähle w_j, x_j, Strategie so dass Fehler $|J - Q|$ und Kosten klein sind!

Idee $f \approx$ einfache Funktion, dessen Integral analytisch berechenbar ist.

z.B. $f \approx \alpha_0 + \alpha_1 x + \dots + \alpha_n x^n = p_n(x)$

$f \approx$ trigonometrischen Polynom. $= \alpha_0 + \alpha_1 e^{i1x} + \alpha_2 e^{i2x} + \dots + \alpha_{-1} e^{i(-1)x} + \alpha_{-2} e^{i(-2)x} + \dots$

Gegeben Knoten $x_0, x_1, \dots, x_n$

kann man $\alpha_0, \alpha_1, \dots, \alpha_n$ berechnen, so dass

$$p_n(x) = \alpha_0 + \alpha_1 x + \dots + \alpha_n x^n \quad \text{erfüllt}$$

$$p_n(x_j) = f(x_j) \quad \text{für} \quad j = 0, 1, 2, \dots, n.$$

Im Prinzip ist das das LGS:

$$\begin{bmatrix} 1 & x_0 & \dots & x_0^n \\ 1 & x_1 & \dots & x_1^n \\ 1 & x_2 & \dots & x_2^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \dots & x_n^n \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}$$

Somit $f(x) \approx p(x)$ und dann

$$\int_a^b f(x)\,dx \approx \int_a^b p(x)\,dx = \text{exakt}$$

$\Rightarrow$ das liefert letztendlich Gewichte $w_0, \dots, w_n$.

Man kann beweisen:

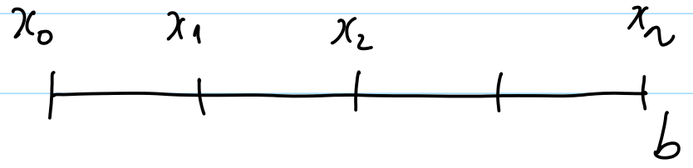
$f \in C^n[a,b]$ (falls f n Mal stetig differentierbar auf $[a,b]$)

$\Rightarrow$

$$|I - Q| \le \frac{1}{n!} (b-a)^{n+1} \max_{z \in [a,b]} |f^{(n)}(z)|$$

Länge des Intervals ist wichtig — Glattheit von f wichtig

Idee Zerlege $\int_a^b f(x)\,dx = \sum_{k=0}^{N-1} \int_{x_k}^{x_{k+1}} f(x)\,dx$



$x_k = x_0 + kh$ mit $h = \frac{b-a}{N}$ klein, $N \in \mathbb{N}$ gross.

und wende die Quadraturformel auf jeden kleine Interval der Länge h

$\Rightarrow$ zusammengesetzte Quadraturformel

Fehler:

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

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

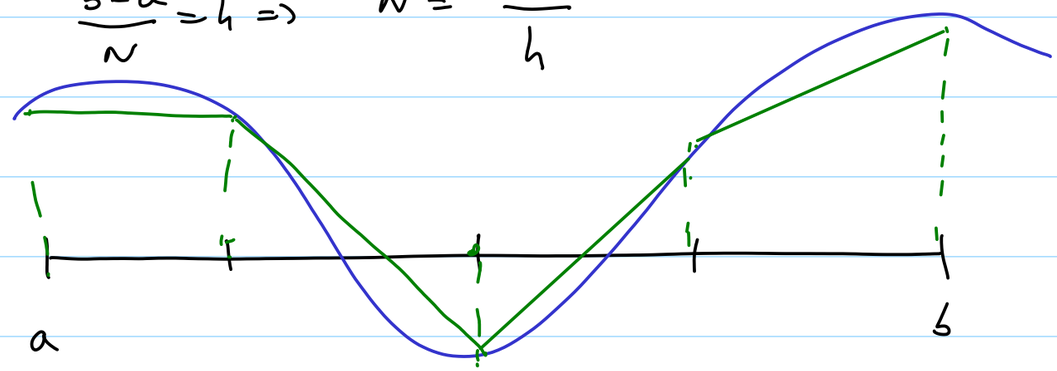
$$\leq \sum_{k=0}^{N-1} \left| \int_{x_k}^{x_{k+1}} f(x)\,dx - Q(f, x_k, x_{k+1}) \right| \leq$$

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

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

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

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



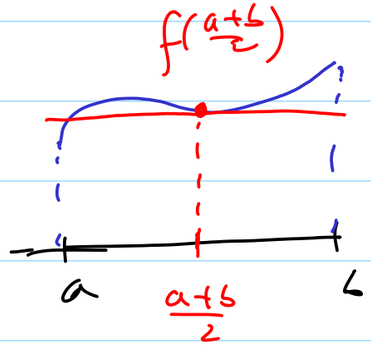
Def Quadraturformel hat Ordnung $n+1$ wenn sie Polynome vom Grad maximal n exakt integriert

das heisst, das erste falsche Ergebnis mit der Quadraturformel passiert x^{n+1}

$\int_a^b p(x)\,dx = Q(p, a, b)$ für alle Polynome vom Grad $\leq n$

und es gibt ein Polynom $\tilde{p}$ vom Grad $n+1$ so dass

$$\int_a^b \tilde{p}(x)\,dx \neq Q(\tilde{p}, a, b).$$



Bsp 1) Mittelpunktsregel:

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

Bem 1) Polynome von Grad 0 und 1 werden mit Q^M exakt integriert

$\Rightarrow$ MPR hat Quadraturordnung 2

Fehler: $\frac{(b-a)^3}{24} f''(\tau)$ mit $\tau \in [a,b]$

Bem 2) offene Quadraturformel:

Enden $[a,b]$ sind keine Knoten.

Knoten $s_1 = \frac{a+b}{2}$, $w_1 = (b-a)$.

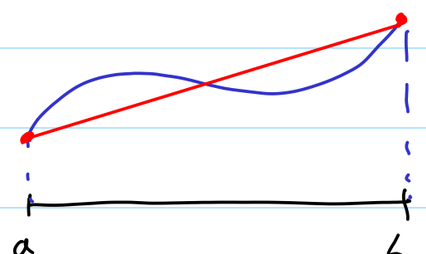
Zusammengesetzte MPR:

$$\int_a^b f(x)\,dx = \sum_{k=0}^{N-1} \int_{x_k}^{x_{k+1}} f(x)\,dx \approx \sum_{k=0}^{N-1} (x_{k+1}-x_k) f\Big(\underbrace{\tfrac{x_k+x_{k+1}}{2}}_{t_k}\Big)$$

$$t_k = \tfrac{1}{2}(x_k + x_{k+1}) = x_k + \tfrac{1}{2}h \quad \text{mit } h = x_{k+1}-x_k = \frac{b-a}{N}$$

$$\int_a^b f(x)\,dx \approx \sum_{k=0}^{N-1} h f(t_k) = \frac{b-a}{N} \sum_{k=0}^{N-1} f(t_k).$$

Bsp 2) Trapezregel



$$Q^T(f,a,b) = \frac{b-a}{2} f(a) + \frac{b-a}{2} f(b) =$$

$$= \sum_{j=1}^{2} w_j f(s_j) \quad \text{mit } s_1 = a,\ s_2 = b,$$

$$w_1 = w_2 = \frac{b-a}{2}$$

Bem 1) Ordnung 2; Fehler $\frac{1}{12}(b-a)^3 f''(z)$ mit $z \in [a,b]$

2) geschlossene Quadraturformel:
Enden von $[a,b]$ sind Knoten.

Zusammengesetzte Trapezregel:

$$\int_a^b f(x)\,dx = \sum_{k=0}^{N-1} \int_{x_k}^{x_{k+1}} f(x)\,dx \approx \sum_{k=0}^{N-1} \left(\frac{h}{2} f(x_k) + \frac{h}{2} f(x_{k+1}) \right)$$

$$= \frac{h}{2} f(x_0) + \boxed{\frac{h}{2} f(x_1) + \frac{h}{2} f(x_1)} + \boxed{\frac{h}{2} f(x_2)} + \ldots + \boxed{\frac{h}{2} f(x_{N-1}) + \frac{h}{2} f(x_N)} =$$

$$= \frac{h}{2} f(x_0) + h f(x_1) + h f(x_2) + \ldots + h f(x_{N-1}) + \frac{h}{2} f(x_N) =$$

$$= \frac{h}{2} f(x_0) + h \sum_{k=1}^{N-1} f(x_k) + \frac{h}{2} f(x_N).$$

Bsp 3) Simpson Regel

$f \approx$ Polynom von Grad 2 $\Rightarrow$

$$Q^S(f,a,b) = \underbrace{\frac{b-a}{6}}_{w_1} \underbrace{f(a)}_{=: x_1} + \underbrace{\frac{b-a}{6} 4}_{w_2} \underbrace{f\left(\frac{a+b}{2}\right)}_{=: x_2} + \underbrace{\frac{b-a}{6}}_{w_3} \underbrace{f(b)}_{=: x_3}$$

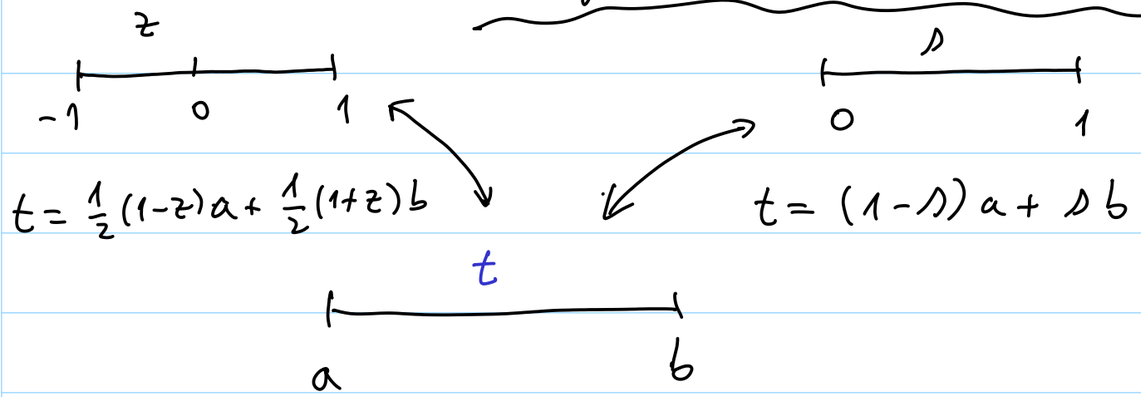
Fehler: $\frac{1}{90} \left(\frac{b-a}{2}\right)^5 f^{(4)}(z)$ mit $z \in [a,b]$

Zusammengesetzte Simpson-Regel $\Rightarrow$ Summen.

Bem Polynome vom Grad 3 werden exakt integriert

$\Rightarrow$ Quadraturordnung 4.

§1.2. Referenzintervalle und symmetrische Quadraturformel



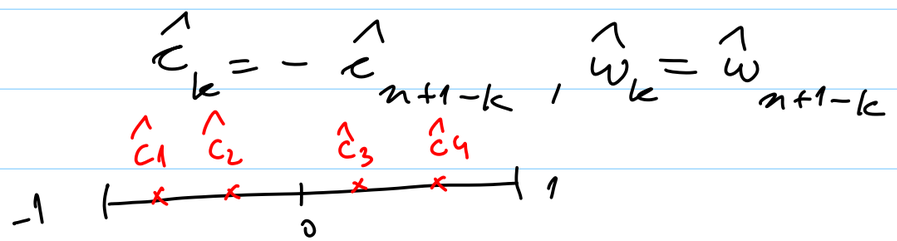
$$\int_a^b f(t)\,dt = \frac{b-a}{2}\int_{-1}^{1} \hat{f}(z)\,dz \approx \frac{b-a}{2}\sum_{j=1}^{n} \hat{\omega}_j \cdot \hat{f}(\hat{c}_j)$$

$[-1,1]$

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

Definition QF auf $[-1,1]$ heisst symmetrisch falls

$$\hat{c}_k = -\hat{c}_{n+1-k}, \quad \hat{\omega}_k = \hat{\omega}_{n+1-k}$$



Theorem

Die Quadraturordnung einer symmetrischen QF ist gerade.

Beweis Annahme: QF ist exakt für Polynome vom Grad $2m-2$

Nehme $f(x) = ax^{2m-1}$

Die QF ist exakt für ax^{2m-1} falls $Q(f,-1,1) = a\int_{-1}^{1} x^{2m-1}\,dx = 0$

$$Q(f,-1,1) = a\sum_{k=1}^{n} \hat{\omega}_k \hat{c}_k^{2m-1} = a\sum_{k=1}^{n} \hat{\omega}_{n+1-k}\left(-\hat{c}_{n+1-k}\right)^{2m-1} =$$

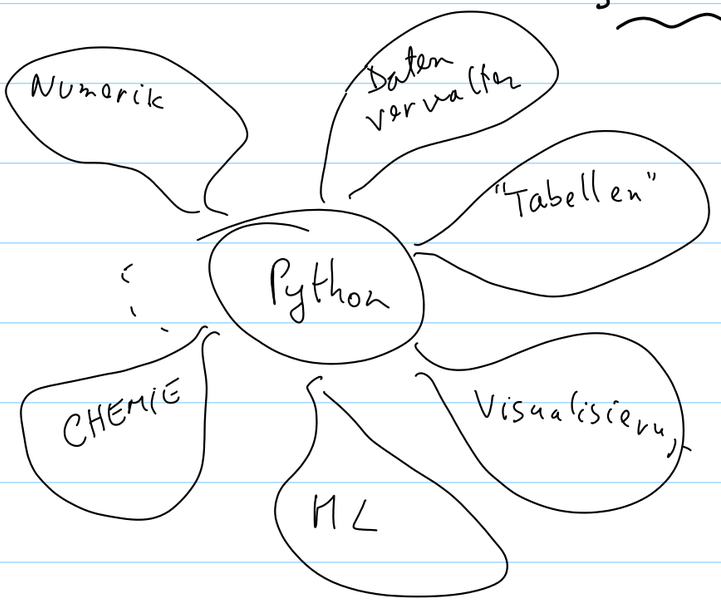
$$= -a\sum_{k=1}^{n} \hat{\omega}_{\underbrace{n+1-k}_{j}} \hat{c}_{n+1-k}^{2m-1} =$$

$$= -a\sum_{j=n}^{1} \hat{\omega}_j \hat{c}_j^{2m-1} = -a\sum_{j=1}^{n} \hat{\omega}_j \hat{c}_j^{2m-1} = -Q(f,-1,1) \Rightarrow$$

21.02.2020

$\Rightarrow Q(f,-1,1) = 0$.

§1.3. Implementierung in Python



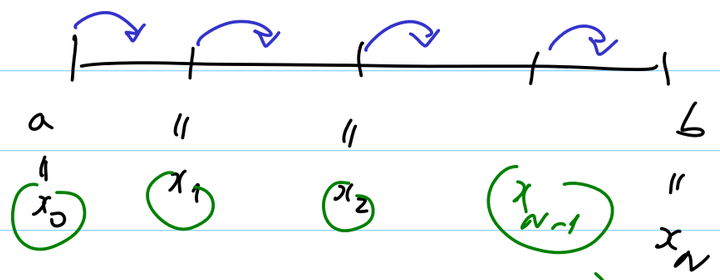
numpy

scipy

matplotlib

sympy

$[a,b]$



$x_k = a + kh$; $h = \frac{b-a}{N}$

$a + h\,[0, 1, 2, \ldots, N-1]$

$x = [a,\ a+h,\ a+2h,\ \ldots,\ a+(N-1)h]$

$t = \frac{h}{2} + x$

myquad01.py

```python
import numpy as np
import matplotlib.pyplot as plt

# function to integrate
def myfunc(s, a):
        res = 1./np.sqrt(1-a*np.sin(s)**2)
        return res

# let us plot it
x = np.linspace(0,np.pi/2) # points in interval
fx = myfunc(x,a=np.pi/6) # evaluate function in points

plt.plot(x, fx)
plt.show()

```

myquad02.py

```python
import numpy as np
import matplotlib.pyplot as plt


def Elam(x, t=1000.,c1 = 3.7413*1E8, c2=1.4388*1E4):
        res =c1/( x**5*(np.exp(c2/(x*t)) -1) )
        return res

x = np.linspace(0,10)
Ex = Elam(x)
#plt.plot(x,Ex)
#plt.show()

from scipy.integrate import quad
t=1000.
c1 = 3.7413*1E8
c2=1.4388*1E4
res, err = quad(Elam,0, 20, args=(t,c1,c2))
print(res/t**4, err)
res, err = quad(Elam,0, np.infty, args=(t,c1,c2))
print(res/t**4, err)

def mpr(f,a,b,N):
        '''
        integrates f from a to b with Midpoint Rule
        f : function to integrate
        a : left end
        b : right end
        N : number of subintervals
        '''
        h = (b-a)/N
        x = a+ h*np.arange(N)
        t = h/2 + x
        res = sum(f(t))*h
        return(res)

N = 50
print(mpr(Elam,0,20,N)/t**4)

```

myquad03.py

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

# variable number of arguments
def mpr(f,a,b,N, args=()):
        '''
        integrates f from a to b with Midpoint Rule
        f : function to integrate
        a : left end
        b : right end
        N : number of subintervals
        '''
        h = (b-a)/N
        x = a+ h*np.arange(N)
        t = h/2 + x
        res = sum(f(t, *args))*h
        return(res)

def Elam(x, T=1000.,c1 = 3.7413*1E8, c2=1.4388*1E4):
        res =c1/( x**5*(np.exp(c2/(x*T)) -1) )
        return res

T=1000.
c1 = 3.7413*1E8
c2=1.4388*1E4
res, err, info = quad(Elam,0, 20, args=(T,c1,c2),
  full_output=True)
print(res/T**4, err)
print('quad needs '+str(info['neval'])+' functions
  evaluations')
res, err, info = quad(Elam,0, np.infty, args=(T,c1,c2),
  full_output=True)
print(res/T**4, err)
print('quad needs '+str(info['neval'])+' functions
  evaluations')

N = 50
print(mpr(Elam,0,20,N, args=(T,c1,c2))/T**4)


```

myquad01.py | myquad02.py | myquad03.py | **myquad04.py**

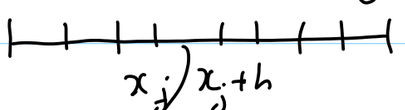
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
# variable named number of arguments

def mpr(f,a,b,N, **kwargs):
        '''
        integrates f from a to b with Midpoint Rule
        f : function to integrate
        a : left end
        b : right end
        N : number of subintervals
        '''
        h = (b-a)/N
        x = a+ h*np.arange(N)
        t = h/2 + x
        res = sum(f(t, **kwargs))*h
        return(res)

def Elam(x, T=1000.,c1 = 3.7413*1E8, c2=1.4388*1E4):
        res =c1/( x**5*(np.exp(c2/(x*T)) -1) )
        return res

T=1000.
c1 = 3.7413*1E8
c2=1.4388*1E4
res, err, info = quad(Elam,0, 20, args=(T,c1,c2),
full_output=True)
print(res/T**4, err)
print('quad needs '+str(info['neval'])+' functions
evaluations')
res, err, info = quad(Elam,0, np.infty, args=(T,c1,c2),
full_output=True)
print(res/T**4, err)
print('quad needs '+str(info['neval'])+' functions
evaluations')

N = 50
print(mpr(Elam,0,20,N,T=1000.,c1 = 3.7413*1E8,
c2=1.4388*1E4 )/T**4)
#print(mpr(Elam,0,20,N, args=(T,c1,c2))/T**4)


```

(C) V. Gradinaru

25.02.2020

§ 1.4. Fehler für Quadratur

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



$$\int_{x_j}^{x_{j+1}} f(x)dx = h\int_0^1 f(x_j+th)\,dt \qquad x = x_j+th$$

Referenzinterval $[0,1]$: Knoten (c_j), Gewichte (b_j)

$[0,1]$

Quadraturformel auf $[0,1]$ mit Knoten (c_j), Gewichte (b_j)

$$\sum_{j=1}^{n} b_j\, g(c_j) \approx \int_0^1 g(t)\,dt$$

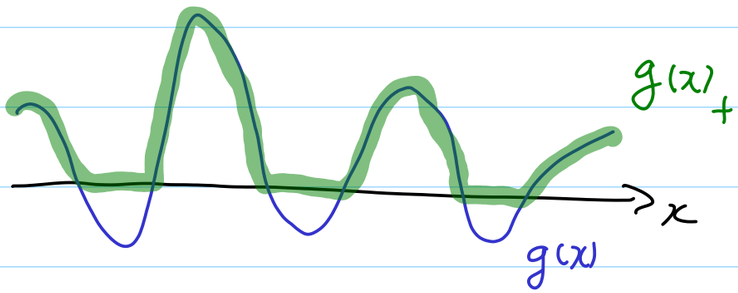
Fehler $E(g) = \int_0^1 g(t)dt - \sum_{j=1}^{n} b_j\, g(c_j)$

Bem Fehler ist linear in g

$E(Ag+Bf) = AE(g) + BE(f)$.

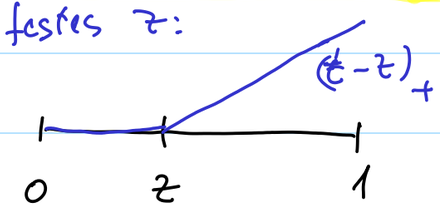
$A,B \in \mathbb{R}$, g,f Funktionen

das positive Teil $g(x)_+ = \begin{cases} g(x), & \text{falls } g(x) > 0 \\ 0, & \text{sonst} \end{cases}$

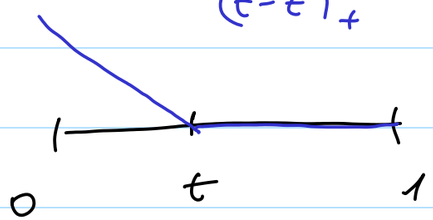


Peano-Kern: $\alpha(z,t) = \frac{1}{(n-1)!}(t-z)_+^{n-1}$

für festes z:



für festes t:



als Funktion in der Variable t

$$K_n(z) = E(\alpha(z,\cdot)) = \int_0^1 \frac{1}{(n-1)!}(t-z)_+^{n-1}\,dt - \sum_{j=1}^{s} b_j \frac{(c_j-z)_+^{n-1}}{(n-1)!} =$$

$$= \frac{(1-z)^n}{n!} - \sum_{j=1}^{s} b_j \frac{(c_j-z)_+^{n-1}}{(n-1)!}$$

Theorem

Sei Q eine Quadraturformel mit Quadraturordnung n und sei g n-mal stetig differentierbar.

Dann

$$E(g) = \int_0^1 K_n(z)\, g^{(n)}(z)\,dz.$$

Beweis Taylor um Punkt 0:

$$g(t) = \underbrace{g(0) + \frac{t}{1!}g'(0) + \dots + \frac{t^{n-1}}{(n-1)!}g^{(n-1)}(0)}_{q(t)\text{ Polynom vom Grad } n-1} + \underbrace{\int_0^t \frac{(t-z)^{n-1}}{(n-1)!}g^{(n)}(z)\,dz}_{=}$$

$$\int_0^1 \frac{(t-z)_+^{n-1}}{(n-1)!}g^{(n)}(z)\,dz$$

Linearität von E: $E(g) = E(q) + \int_0^1 E(\alpha(z,\cdot))\,g^{(n)}(z)\,dz$

$E(q) = 0$ (da QF hat Ordnung n.)

$K_n(z)$

Bemerkung Wenden wir jetzt den Satz auf

$$g(t) = f(x_0 + th) \qquad \left[\begin{matrix} x = x_0 + th \\ dx = h\,dt \end{matrix}\right]$$

$$\int_{x_0}^{x_0+h} f(x)\,dx = h\int_0^1 g(t)\,dt$$

$$\int_{x_0}^{x_0+h} f(x)\,dx - h\sum_{j=1}^{s} b_j \cdot f(x_0 + c_j h) =$$

$$= h\int_0^1 g(t)\,dt - h\sum_{j=1}^{s} b_j \cdot g(c_j) =$$

$$= h\left(\int_0^1 g(t)\,t - \sum_{j=1}^{s} b_j\, g(c_j)\right) = h\,E(g)$$

$$= h \cdot h^n \cdot \int_0^1 k_n(z) f^{(n)}(x_0 + hz)\,dz$$

$$E(g) = \int_0^1 K_n(z)\, g^{(n)}(z)\,dz$$

$$g(t) = f(x_0 + th) \Rightarrow g'(t) = f'(x_0 + th)\,h$$

$$g''(t) = f''(x_0 + th)\,h^2$$

$$\dots \qquad g^{(n)}(t) = f^{(n)}(x_0 + th)\,h^n$$

Fehler $\int_{x_0}^{x_0+h} f(x)\,dx - h\sum_{j=1}^{n} b_j \cdot f(x_0 + c_j \cdot h) =$

$$= h^{n+1} \int_0^1 k_n(z) f^{(n)}(x_0 + hz)\,dz$$

Zusammengefasst:

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

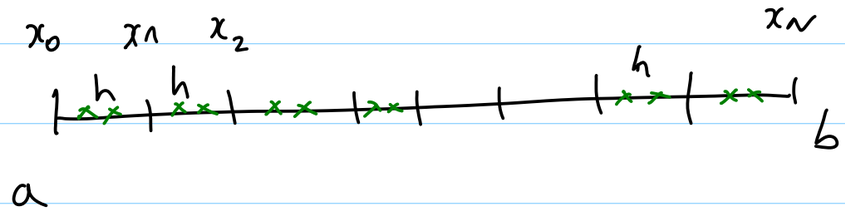
$$\leq \underbrace{(b-a)}_{C} \cdot h^{n} \max_{x \in [a,b]} |f^{(n)}(x)| \longrightarrow 0 \text{ für } h \to 0$$

n: Quadraturordnung n

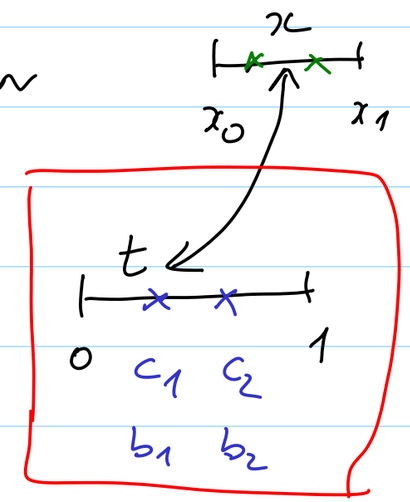
$f^{(n)}$: Glattheit von f

C: kommt aus dem Peano kern. $\int_0^1 k_n(z)\,dz = \text{konstante}$

MPR, TR $= \frac{1}{12}$ Simpson $\frac{1}{2880}$.



Gitter $x_0, x_1, \ldots, x_N$



Fehler $\leq \sum_{k=0}^{N-1} c \cdot h^{n+1} \max_{t \in [a,b]} \|f^{(n)}(t)\| = C \cdot \max_{[a,b]} \|f^{(n)}(t)\| \cdot h^{n+1} N =$

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

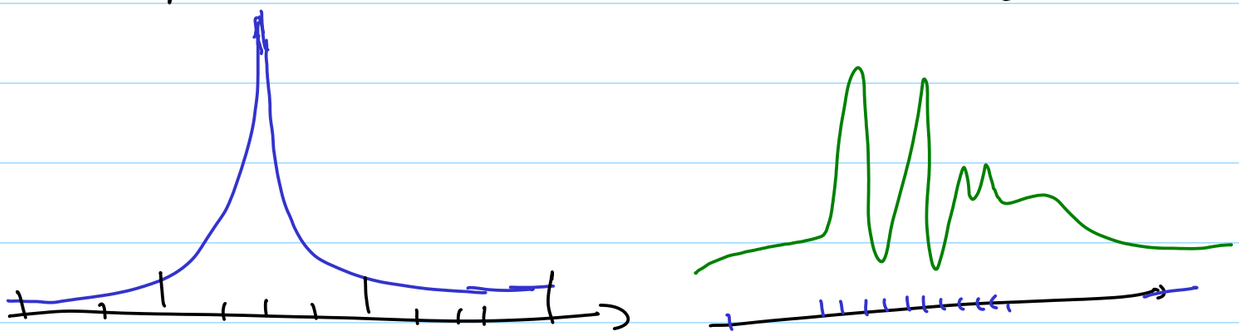
$= c \cdot h^{n} \max_{[a,b]} \|f^{(n)}(t)\|$

linear in $\log h$.

Fehler $= c h^n \Rightarrow \log \text{Fehler} = n \log h + \log c$

§1.5. Adaptive Quadratur

Ziel: Optimiere die Funktionsauswertungen.



Fehler $\sim h^n \max_{z\in(a,b)} \|f^{(n)}(z)\|$

↳ falls klein, dann brauchen wir nicht unbedingt ein kleines h.

lokale Fehler $\varepsilon_k \sim h^{n+1} \max_{x\in[x_k,x_{k+1}]} |f^{(n)}(x)|$

Wie schätzen wir den lokalen Fehler ohne weitere Informationen über f?

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

$$\int_{x_k}^{x_{k+1}} f(x)dx = Q^S(f, x_{k-1}, x_{k+1}) + ch^5 \max_{z\in[x_k,x_{k+1}]} |f^{(IV)}(z)|$$

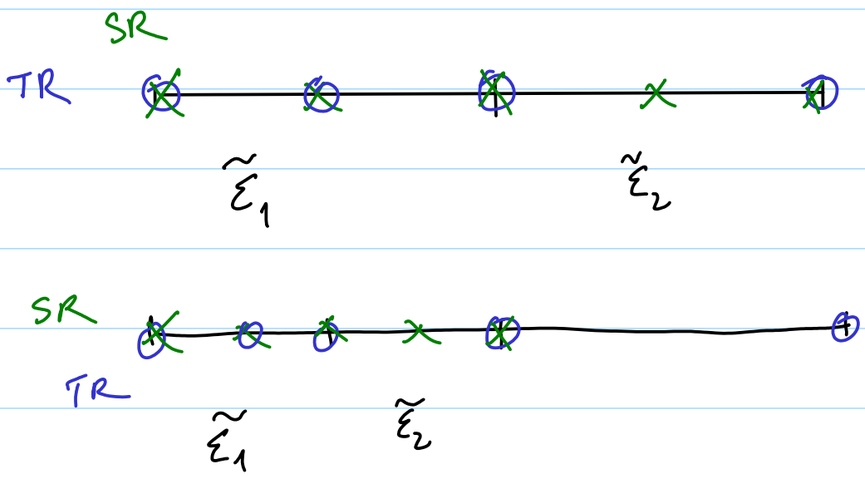
Idee:

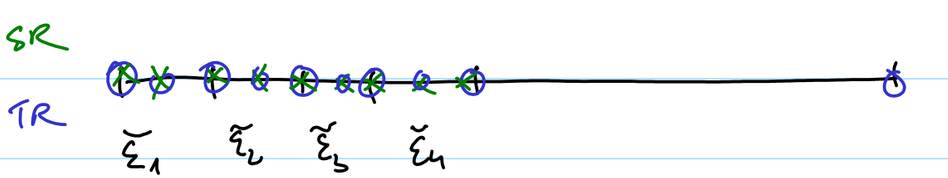
$$\varepsilon_k = \left| \int_{x_k}^{x_{k-1}} f(x)dx - Q^T(f, x_k, x_{k+1}) \right| \approx$$

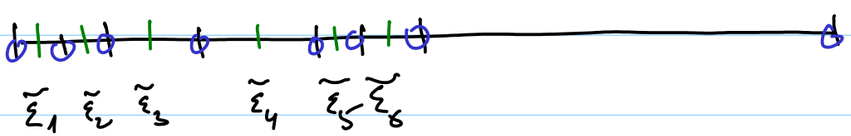
$$\approx \underbrace{\left| Q^S(f, x_k, x_{k+1}) - Q^T(f, x_k, x_{k+1}) \right|}_{=\tilde{\varepsilon}_k}$$ Schätzung. von lokalen Fehler ε_k

Idee: 1) Verfeinere die Intervalle wo $\tilde{\varepsilon}_k$ gross ist.

2) Verwende das Ergebniss von Q^S.



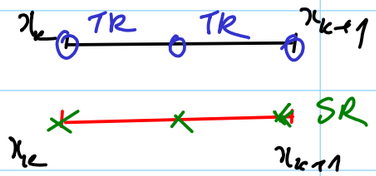




$h_k = x_{k+1} - x_k$

$\frac{1}{2}(x_k + x_{k+1}) \Rightarrow [mp_0, mp_1, mp_2, mp_3]$

auf $[x_k, x_{k+1}]$



array

```
def adaptquad(f,M,rtol,abstol):
    """
    adaptive quadrature using trapezoid and simpson rules
    Arguments:
    f       handle to function f
    M       initial mesh
    rtol    relative tolerance for termination
    abstol  absolute tolerance for termination, necessary in case the exact
integral value = 0, which renders a relative tolerance meaningless.
    """
    h = diff(M)                                  #  compute lengths of mesh intervals
    mp = 0.5*( M[:-1]+M[1:] )                    #  compute midpoint positions
    fx = f(M); fm = f(mp)                        #  evaluate function at positions and
midpoints
    trp_loc = h*( fx[:-1]+2*fm+fx[1:] )/4        #  local trapezoid rule
    simp_loc= h*( fx[:-1]+4*fm+fx[1:] )/6        #  local simpson rule
    I = sum(simp_loc)                            #  use simpson rule value as
intermediate approximation for integral value
    est_loc = abs(simp_loc - trp_loc)            #  difference of values obtained from
local composite trapezoidal rule and local simpson rule is used as an estimate
for the local quadrature error.
    err_tot  = sum(est_loc)                      #  estimate for global error (sum
moduli of local error contributions)
    #  if estimated total error not below relative or absolute threshold, refine
mesh
    if err_tot > rtol*abs(I) and err_tot > abstol:
        refcells = nonzero( est_loc > 0.9*sum(est_loc)/size(est_loc) )[0]
        I = adaptquad(f,sort(append(M,mp[refcells])),rtol,abstol)  #  add
midpoints of intervalls with large error contributions, recurse.
    return I

if __name__ == '__main__':
    f = lambda x: exp(6*sin(2*pi*x))
    #f = lambda x:  1.0/(1e-4+x*x)
    M = arange(11.)/10 # 0, 0.1, ...  0.9, 1
    rtol = 1e-6; abstol = 1e-10
    I = adaptquad(f,M,rtol,abstol)
    exact,e = integrate.quad(f,M[0],M[-1])
    print('adaptquad:',I, '"exact":',exact)
    print('error:',abs(I-exact))
```

x_0 x_1 $mp(1)$ x_2 $mp(2)$ x_3 x_4

$M = [x_0, x_1, x_2, x_3, x_4]$

$h = [h_0, h_1, h_2, h_3]$ $\quad h_0 = x_1 - x_0,\ h_1 = x_2 - x_1, \ldots$

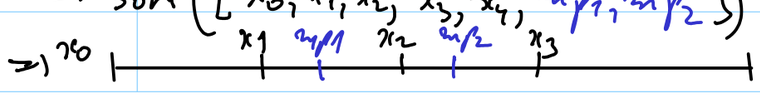
$\frac{\text{tot}}{\#\text{Intervale}}$ = Mittleren Fehler

[False, True, True, False]

nonzero ⇒ Welche Indizes ≠ 0 ⇒ 1, 2

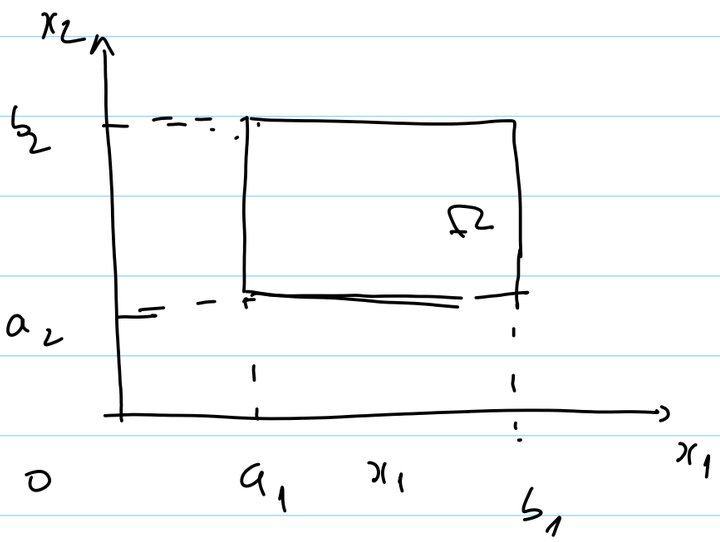
$mp[\text{refcells}] = [mp_1, mp_2]$ lese help für nonzero um [0] zu verstehen!

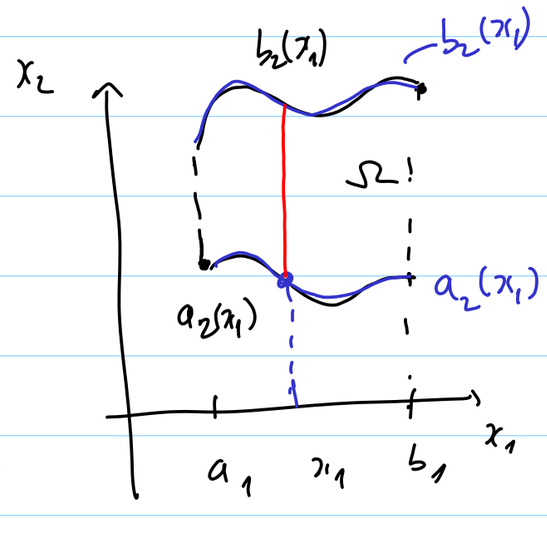
$\Rightarrow \text{sort}([x_0, x_1, x_2, x_3, x_4, mp_1, mp_2])$

$\Rightarrow$ 

neues Gitter

§1.6 Quadratur in $\mathbb{R}^d$, $d=2,3$

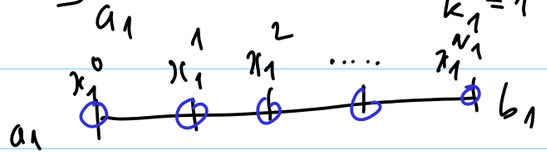




$$I = \int_\Omega f(x_2,x_1)\,dx_2\,dx_1 = \int_{a_1}^{b_1}\int_{a_2(x_1)}^{b_2(x_1)} f(x_2,x_1)\,dx_2\,dx_1 =$$

$$F(x_1)$$

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



$$\approx \underbrace{\frac{b_1-a_1}{N_1}}_{h_1} \sum_{k_1=1}^{N_1} \sum_{j_1=1}^{p_1} F\left(x_1^{k_1-1} + h_1 c^1_{j_1}\right) \cdot b^1_{j_1}$$

knoten in Ox_1-Richtung auf Interval $[x_1^{k_1-1}, x_1^{k_1}]$

$$= h_1 \sum_{k_1=1}^{N_1} \sum_{j_1=1}^{p_1} \int_{a_2(x_1^{k_1-1}+h_1 c^1_{j_1})}^{b_2(x_1^{k_1-1}+h_1 c^1_{j_1})} f\left(x_2, x_1^{k_1-1} + h_1 c^1_{j_1}\right) dx_2 \; b^1_{j_1}$$

$$= h_1 \sum_{k_1=1}^{N_1} \sum_{j_1=1}^{p_1} \frac{b_2-a_2}{N_2} \sum_{k_2=1}^{N_2} \sum_{j_2=1}^{p_2} f\left(x_2^{k_2-1} + h_2 c^2_{j_2}, x_1^{k_1-1} + h_1 c^1_{j_1}\right) b^2_{j_2} b^1_{j_1}$$

h_2

$$= h_1 h_2 \sum_{k_1=1}^{N_1} \sum_{k_2=1}^{N_2} \sum_{j_1=1}^{p_1} \sum_{j_2=1}^{p_2} f\left(x_2^{k_2-1} + h_2 c^2_{j_2}, x_1^{k_1-1} + h_1 c^1_{j_1}\right) b^2_{j_2} b^1_{j_1}$$

in d-Dimensionen:

$$h_1 h_2 \dots h_d \sum_{k_1} \sum_{k_2} \dots \sum_{k_d} f(\dots)\, b^d_{j_d}\, b^{d-1}_{j_{d-1}} \dots b^1_{j_1}$$

$N_1 \cdot N_2 \dots N_d$ Auswertungen von f

zu teuer!

28.02.2020

code.

```python
import numpy as np

def trapy(f, a, b, N):
        '''
        quadrature via trapezoidal rule
        f: function to integrate
        a: left end of interval
        b: right end of interval
        N: number of intervals
        '''
        x, h = np.linspace(a,b,N+1,retstep=True)
        res = 0.5*h*(f(x[0]) + 2*np.sum(f(x[1:-1])) +f(x[-1]) )
        return(res)

def simpson(f, a, b, N):
    '''
    Simpson quadrature of function f from a to b with N subintervals.

    f:      Function f(x)
    a, b:   Bounds of the integration interval
    N:      Number of subintervals
    '''
    x, h = np.linspace(a, b, 2*N+1, retstep=True)
    # quadrature weights:
    #   boundary nodes: w=1/3
    #   internal nodes: w=2/3
    #   midpoint nodes: w=4/3
    res = h/3.0 * np.sum(f(x[:-2:2]) + 4.0*f(x[1:-1:2]) + f(x[2::2]))
    return(res)

def zz(x):
    a = (1./3.)**0.3
    #b = (1./2.)**0.3
    p = 1.
    y = np.abs(np.cos(10*x)-a)**(p*0.5)
    #z = abs(x-b)**(p*0.5)
    return y

```

```
38
39 def gg(x):
40     a = 1/np.sqrt(2.)
41     b = 1/np.sqrt(3)
42     res = 0.
43     p = 2.5
44     F = 8.#20.
45     y = (np.abs(np.sin(F*x)-b))**p; y /= p
46
47     res = y
48     #res =  1.0 / (1.0 + (5*res)*1)
49     #if x>0: res += x
50     #else: res += -x
51     return res
52
53 if __name__ == "__main__":
54     from scipy import integrate
55
56     # Define a function and an interval:
57     #f = lambda x: 1.0 / (1.0 + (5*x)**2) # better than expecetd
58     #f = gg # worse than expected for simpson
59     f = zz # disaster of expectations
60     left = -1.0
61     right = 1.0
62
63     # Exact integration with scipy.integrate.quad:
64     print('----------------------------------------')
65     print('ntegrate.quad gives:')
66     rezquad = integrate.quad(f, left, right, limit= 100, full_output=True)
67     #rezquad = integrate.quad(f, left, right, full_output=True)
68     exact = rezquad[0]
69     e = rezquad[1]
70     neval = rezquad[2]['neval']
71     print('with ',neval,' function evaluations')
72     if len(rezquad) == 4: print('WARNING:\n', rezquad[3])
73     print('vaule =',exact, ' estimated error = ', e)
74     print('----------------------------------------')
75
```

```
76     # Simpson and trapezoidal rules for different number of quadrature points
77     N = np.arange(2, 301)
78     res = np.zeros(np.size(N))
79     res_T = np.zeros(np.size(N))
80     for i, n in enumerate(N):
81         res[i] = simpson(f, left, right, n)
82         res_T[i] = trapy(f, left, right, 2*n)
83
84     err = np.abs(res - exact)
85     err_T = np.abs(res_T - exact)
86
87     #print err
88     # Linear fit to determine convergence order
89     #p = polyfit(log(N[-50:]), log(err[-50:]), 1)
90     p = np.polyfit(np.log(N[20:]), np.log(err[20:]), 1)
91     print(("Convergence order Simpson: %f" % -p[0]))
92     p = np.polyfit(np.log(N[50:]), np.log(err_T[50:]), 1)
93     print(("Convergence order Trapez: %f" % -p[0]))
94     print('----------------------------------------')
95
96     from pylab import loglog, show, plot, figure, legend
97     figure()
98     x = np.linspace(left, right, N[-1])
99     plot(x, f(x))
00     show()
01     figure()
02     loglog(N,err,'x', label='Simpson')
03     loglog(N, err_T,'v', label='Trapez')
04     loglog(N,1./N**2, label='$1/N^2$')
05     loglog(N, 1./N**3, label = '$1/N^3$')
06     loglog(N, 1./N**4, label='$1/N^4$')
07     legend()
08     show()
09
```

Prüfung
Numerische Methoden D-PHYS, WS 2019/2020

1. *(8 Punkte) Simpsonregel*

Wir wollen die zusammengesetzte Simpsonregel verwenden, um das Integral

$$I = \int_0^1 f_i(x)\mathrm{d}x$$

von $f_i(x)$, $i = 1, 2$ auf N Teilintervallen oder mit n Funktionsauswertungen zu berechnen.
Die zwei Funktionen sind durch

$$f_1(x) := \frac{1}{1+4x^2} \qquad f_2(x) := x^{\frac{1}{3}}\,. \tag{1}$$

gebeben.

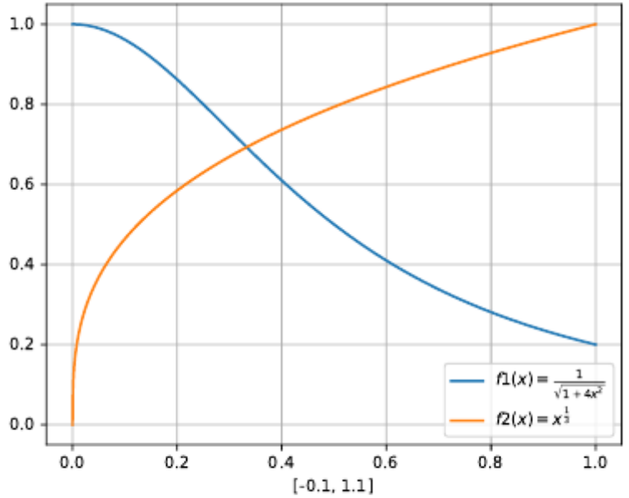


Abbildung 1 – Die Funktionen f_1 und f_2.

a) Im File `quadrature.py` implementieren Sie die Funktion `simpson(f,a,b,N)`. Dabei sollen die integrierende Funktion `f`, die untere und obene Grenzen `a` und `b` und die Anzahl `N` der Teilintervalle in der zusammengesetzten Regel eingegeben werden. Die Funktion soll das Wert `I` ausgeben.

b) Schreiben Sie die Funktion `quadrature_errr(quadrature_rule, f, exact)`; dabei sollen die Quadraturregel `quadrature_rule`, die integrierende Funktion `f` und das genaue Wert `exact` ihres Integrals eingegeben werden. Die Funktion `quadrature_errr` soll den Fehler `error` und die Anzahl Teilintervalle `n_chunks`$= 2^k$ mit $k \in \{3, \ldots, 9\}$ ausgeben.
Hinweis: Die Funktion `enumerate` kann nützlich sein.

c) Welche Konvergenzordnung beobachtet man für jede der zwei Funktionen? Erklären Sie die beobachteten/erwarteten Ergebnisse.

c) Für f_1 beobachten wir $O(h^4)$
(erwarten)
$h = \frac{1}{N}$

Für f_2 beobachten wir $O(h)$

Wir erwarten/beobachten
für f_1 $O(h^4)$ da f_1 glatt ist
für f_2 weniger da f_2 unglatt ist

```python
# -*- coding: utf-8 -*-
from sympy import *
import numpy as np
import matplotlib.pyplot as plt

def f1(x):
    return 1.0 / (1.0 + 4.0*x**2)

def f2(x):
    return (x)**(1./3.)

def simpson(f, a, b, N):
    """Approximiert das Integral von `f` mit der Simpsonregel.

    Input:
           f : Funktion f(x) welche integiert werden soll.
       a, b : untere/obere Grenze des Integrals.
           N : Anzahl Teilintervalle in der zusammengesetzten
               Simpsonregel.
    """
    x, h = np.linspace(a, b, 2*N+1, retstep=True)
    I = h/3.0 * np.sum(f(x[:-2:2]) + 4.0*f(x[1:-1:2]) + f(x[2::2]))

    return I
```

```python
def quadrature_error(quadrature_rule, f, exact):
    """Berechnet den Fehler und die Konvergenzrate.

    Input:
        quadrature_rule : Quadraturformel welche verwendet werden soll. Die
                          Funktion hat die gleiche Signature wie `simpson` und
                          `trapezoid`.

                      f : Funktion welche integriert werden soll. In der Aufgabe
                          heissen sie `f1` und `f2`.

                  exact : Exakte Referenzlösung des Integrals.

    Output:
                 errors : Fehler der Quadraturformel mit `n_chunks` Teilintervallen.
               n_chunks : Anzahl Teilintervalle.
    """

    n_chunks =  2**np.arange(3,10)
    errors = np.empty(n_chunks.shape)

    for i, N in enumerate(n_chunks):
        approx = quadrature_rule(f, 0., 1., N)
        errors[i] = np.abs(approx - exact)

    return errors, n_chunks
```

§1.6. Quadratur mit erhöhter Ordnung

s Knoten auf Referenzinterval, s Gewichte — so bestimmt, dass Polynome höchstes Grades exakt mit dieser QF integriert werden.

Wie hoch kann so eine Quadraturordnung sein?

$2s$ Unbekannte (Knoten, Gewichte);

$2s$ Gleichungen:

$$\begin{cases} \int_0^1 1\,dt = Q_s(1, 0,1) \\ \int_0^1 t\,dt = Q_s(t, 0,1) \\ \dots \quad \left(\frac{1}{3} = b_1 c_1^2 + b_2 c_2^2 + \dots\right) \\ \int_0^1 t^{2s-1}\,dt = Q_s(t^{2s-1}, 0,1) \end{cases}$$

Da QF exakt

Möchte Ordnung $p = s+m$

Jedes Polynom vom Grad $\le p-1 = s+m-1$ soll mit QF exakt integriert werden.

Sei f Polynom vom Grad $\le p-1 = s+m-1$

IDEE: teile das Polynom f durch

$$M(x) = (x-c_1)(x-c_2)\dots(x-c_s)$$

$$\text{Grad}(M) = s$$

$$f(x) = M(x)g(x) + r(x) \quad \text{mit} \quad \text{Grad}(r) \le s-1$$

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

$\Vert$ ← da QF exakt

$\to \Vert$

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

$M(c_j) = 0$

$= 0$

$\Rightarrow \int_0^1 M(t)\, g(t)\, dt = 0$ für alle Polynome g mit $\mathrm{Grad}(g) \le n-1$.

f = Polynom vom Grad $\le s+n-1$

$\langle M, g \rangle = \int_0^1 M(t)\, g(t)\, dt$ Skalarprodukt im Raum der Polynome

Theorem

Ordnung der QF ist $s+n$ $\Longleftrightarrow$

$\langle M, g \rangle = 0$ für alle Polynome vom Grad $\le n-1$.

$P_n = \mathrm{span}\{1, t, t^2, \dots, t^{n-1}\}$: $M \perp P_n$

Theorem Ordnung einer QF ist höchstens $2s$.

Beweis Annahme: $p \ge 2s+1 \Rightarrow$

$\int_0^1 M(t)\, g(t)\, dt = 0$ für alle Polynome vom Grad $\le s+1$ $\Big\} \Rightarrow$

Nehme $g = M$

$\Rightarrow \int_0^1 M(t)\, M(t)\, dt = 0 \Longleftrightarrow \int_0^1 M(t)^2\, dt = 0 \Rightarrow$

$M(t) \equiv 0$ ↯

Widerspruch $\Rightarrow$ wahr: $p \le 2s$.

Orthogonale Polynome

$w :]a,b[\to \mathbb{R}$ Gewichtfunktion

stetig, $w(x) > 0$ für alle $x \in]a,b[$

$\int_a^b |x|^k w(x)\, dx < \infty$ für $k = 0, 1, 2, \dots$

Betachte den linearen Raum:

$V = \left\{ f :]a,b[\to \mathbb{R},\ \text{stetig},\ \int_a^b |f(x)|^2 w(x)\, dx < \infty \right\} \Rightarrow$

Bem Alle Polynome liegen in V

V: Skalarproduct

$$\langle f,g\rangle = \int_a^b f(x)g(x)\,\omega(x)\,dx$$

Theorem [Gram-Schmidt in LA]

Es existiert eine eindeutige Folge von Polynomen $p_0, p_1, \dots$ mit

$$p_k(x) = x^k + \text{Polynom vom Grad} \le k-1$$

so dass

$$p_k \perp \text{span}\{p_0, p_1, \dots, p_{k-1}\}.$$

Diese Polynome baut man so: (3-Term Rekursion)

$$p_{k+1}(x) = (x - \beta_{k+1})\,p_k(x) - \gamma_{k+1}^2\,p_{k-1}(x)$$

mit $p_0(x) = 1$, $p_{-1}(x) = 0$ und

$$\beta_{k+1} = \frac{\langle x p_k, p_k\rangle}{\langle p_k, p_k\rangle}, \quad \gamma_{k+1}^2 = \frac{\langle p_k, p_k\rangle}{\langle p_{k-1}, p_{k-1}\rangle}.$$

Bem: oft werden sie aber zu $p_k(1) = 1$ "normiert". Darum. $p_2(x) = \frac{3}{2}(x^2 - \frac{1}{3})$

Bem $c_1, c_2, \dots, c_n$ die Nullstellen von (n) von p_n.

$n = p_n$

Bsp 1) $\omega(x) \equiv 1$, $a=0, b=1$: $\int_0^1 f(x)\,g(x)\,dx$

Gram-Schmidt $\Rightarrow$ orthogonale Polynome

Legendre Polynome

QF: Gauss-Quadratur

2) $]a,b[=]-\infty, \infty[$, $\omega(x) = e^{-x^2} \Rightarrow$ Hermite Polynome.

QF $\Rightarrow$ Hermite-Quadratur.

Bsp Gauss-Quadratur

$[0,1]$ Notiere Knoten x_j, $j = 1\dots n$

$[-1,1]$ Notiere Knoten c_j, $j = 1\dots n$.

Beweis des Theorems [Gram-Schmidt-Orthogonalisierung]

Mathematische Induktion:

Annahme: $p_0, p_1, \dots, p_k$ bereits bekannt

Ziel: Konstruiere: p_{k+1}

Nehme $p_{k+1} = x\,p_k + \sum_{j=0}^{k} \alpha_j p_j$

und berechne α_j, so dass $p_{k+1} \perp p_j$ für $j = 0, \dots, k$.

$p_{k+1} \perp p_k \iff 0 = \langle p_{k+1}, p_k \rangle =$

$= \langle x p_k, p_k \rangle + \sum_{j=0}^{k-1} \alpha_j \langle p_j, p_k \rangle + \alpha_k \langle p_k, p_k \rangle$

$\langle p_j, p_k \rangle = 0$ ← Induktionsannahme

$\iff \alpha_k = - \dfrac{\langle x p_k, p_k \rangle}{\langle p_k, p_k \rangle} = -\beta_{k+1}$

$p_{k+1} \perp p_{k-1} \iff 0 = \langle x p_k, p_{k-1} \rangle + \alpha_{k-1} \langle p_{k-1}, p_{k-1} \rangle + 0$

aber $\langle x p_k, p_{k-1} \rangle = \int_{-1}^{1} x\, p_k(x)\, p_{k-1}(x)\, dx = \langle p_k, x p_{k-1} \rangle$

Schreibe $x p_{k-1} = p_k + q$ mit $\operatorname{Grad} q \le k-1 \Rightarrow$

$\Rightarrow \langle x p_k, p_{k-1} \rangle = \langle p_k, p_k \rangle + \langle p_k, q \rangle$, $\langle p_k, q \rangle = 0$

Somit $p_{k+1} \perp p_{k-1} \iff \alpha_{k-1} = - \dfrac{\langle p_k, p_k \rangle}{\langle p_{k-1}, p_{k-1} \rangle} = -\gamma_{k+1}^2$

Für $j \le k-2$:

$p_{k+1} \perp p_j \iff 0 = \langle p_{k+1}, p_j \rangle = \langle x p_k, p_j \rangle + \alpha_j \langle p_j, p_j \rangle =$

$= \langle p_k, \underbrace{x p_j}_{\text{Grad } j+1 \le k-1} \rangle + \alpha_j \langle p_j, p_j \rangle \Rightarrow$

$\langle p_k, x p_j \rangle = 0$

$\Rightarrow \alpha_j = 0.$

1) $n=1$ auf $[0,1]$ $\Rightarrow$ $p_1(x) = x - \frac{1}{2}$ $\Rightarrow$ $\gamma_1 = \frac{1}{2}$, $b_1 = 1$

auf $[-1,1]$ $\Rightarrow$ $p_1(x) = x$ $\Rightarrow$ $c_1 = 0$, $b_1 = 1$

↳ Gauss-Quadratur mit $n=1$ Knoten $\equiv$ MPR.

Ordnung $2n = 2 \cdot 1 = 2$.

2) $n=2$ auf $[-1,1]$: $p_2(x) = x^2 - \frac{1}{3}$ $\Rightarrow$ $\frac{3}{2}\left(x^2 - \frac{1}{3}\right)$

$\gamma_{1,2} = \pm \frac{1}{\sqrt{3}}$, $c_1 = \frac{1}{2} - \frac{\sqrt{2}}{6}$, $c_2 = \frac{1}{2} + \frac{\sqrt{2}}{6}$.

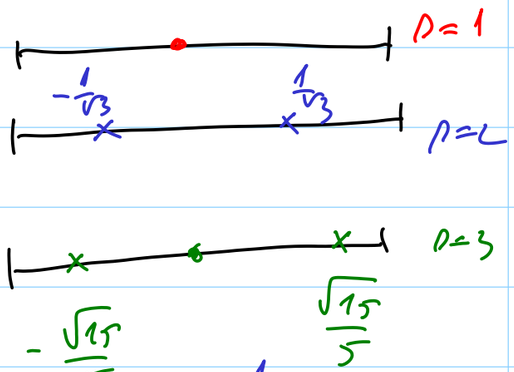
Ordnung $2n = 2 \cdot 2 = 4$

zu $p_3(1) = 1$ "normiert"

3) $n=3$ auf $[-1,1]$: $p_3(x) = \frac{5}{2}x^3 - \frac{3}{2}x$

$\gamma_2 = 0$, $\gamma_{1,3} = \pm \frac{\sqrt{15}}{5}$

$b_2 = \frac{8}{18}$, $b_1 = b_3 = \frac{5}{18}$



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

Ben: Gewichte wurden hier "Hand"-Berechnet via $b_i = \int_0^1 l_i(t)^2 dt$ (l_i = Lagrange Polynom)

Ben Gewichte der Gauss-QF sind positiv

Def Lagrange Polynome zu Stützstellen $x_0, x_1, \dots, x_n$

für $i = 0, 1, 2, \dots, n$

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Bsp x_0, x_1, x_2:

$$l_0(x) = \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_1}$$

$$l_1(x) = \frac{x - x_0}{x_1 - x_0} \quad \frac{x - x_2}{x_1 - x_2}$$

$$l_2(x) = \frac{x - x_0}{x_2 - x_0} \quad \frac{x - x_1}{x_2 - x_1}$$

Bem 1) $\ell_i(x_j) = 0$ für alle $i \neq j$

2) $\ell_i(x_i) = 1$

3) Grad $\ell_i = n$

4) $\sum_{i=0}^{n} \ell_i(x) = 1$ für alle $x \in \mathbb{R}$.

5) $\sum_{i=0}^{n} \ell_i^{(m)}(x) = 0$ für $m \geq 1$.

6) $\ell_0, \ell_1, \dots \ell_n$ bilden eine Basis im Raum der Polynom von Grad $\leq n$

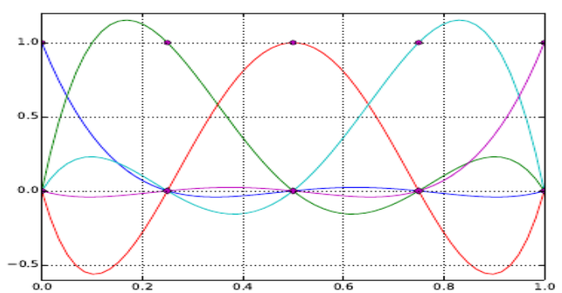


Abb. 8.3.2. Die 5 Lagrange Polynome zu Knoten $0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$.

Seite 289 im Skript

Theorem Die Gewichte der Gauss QF sind positiv.

Beweis Verwende die Lagrange-Polynome zu den Stützstellen. $c_1, c_2, \dots, c_p$ = Knoten der QF

Grad $\ell_i = p-1$ für $i = 1, 2, \dots p$ } exakt $\Rightarrow$

$$\int_0^1 f(t)\,dt \approx \sum_{i=1}^{p} b_i f(c_i)$$

$f(t) = \ell_i(t)^2$ Polynom vom Grad $2(p-1)$ }

$$0 < \int_0^1 \ell_i(t)^2\,dt = \sum_{j=1}^{p} b_j \ell_i(c_j)^2 = b_i \cdot 1$$

$\ell_i(c_j) = 0$ für $j \neq i$

1 für $j = i$

Im Prinzip $b_i = \int_0^1 \ell_i(t)^2\,dt$ könnte man verwenden um Gewichte zu rechnen

Es gibt eine bessere Methode.

3-Term Rekurenz für Polynome:

(Achtung $\underline{a}, \underline{b}, \underline{c}$ sind allgemein, $\underline{c} \neq$ Knoten $t_1, t_2, \dots, t_n$)

$$\begin{cases} p_k(x) = (a_k x + b_k)\, p_{k-1}(x) - c_k p_{k-2}(x) \\ p_{-1}(x) = 0, \quad p_0(x) = 1 \end{cases} \Rightarrow$$

$$\begin{cases} x\, p_{k-1}(x) = \boxed{\frac{c_k}{a_k}}\, p_{k-2}(x) \boxed{- \frac{b_k}{a_k}}\, p_{k-1}(x) + \boxed{\frac{1}{a_k}}\, p_k(x) \\ \text{für } k = 1, 2, 3, \dots, n \end{cases}$$

$\underline{A}$ $\qquad \underline{e}_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$

$\underline{p}(x)$

$$x \begin{bmatrix} p_0(x) \\ p_1(x) \\ \vdots \\ p_{k-1}(x) \\ \vdots \\ p_{n-2}(x) \\ p_{n-1}(x) \end{bmatrix} = \begin{bmatrix} -\frac{b_1}{a_1} & \frac{1}{a_1} & & & 0 \\ \frac{c_2}{a_2} & -\frac{b_2}{a_2} & \frac{1}{a_2} & & \\ & \ddots & \ddots & \ddots & \\ & \frac{c_k}{a_k} & -\frac{b_k}{a_k} & \frac{1}{a_k} & \\ & & \ddots & \ddots & \frac{1}{a_{n-1}} \\ 0 & & & \frac{c_n}{a_n} & -\frac{b_n}{a_n} \end{bmatrix} \begin{bmatrix} p_0(x) \\ p_1(x) \\ \vdots \\ p_{k-1}(x) \\ \vdots \\ p_{n-2}(x) \\ p_{n-1}(x) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \frac{1}{a_n} p_n \end{bmatrix}$$

3-Term Rekursion $\Leftrightarrow$

$$x\, \underline{p}(x) = \underline{\underline{A}}\, \underline{p}(x) + \frac{1}{a_n} p_n(x)\, \underline{e}_n$$

Knoten der Gauss-QF sind die Nullstellen von $p_n(x)$

$\hookrightarrow t_1, t_2, \dots, t_n \Rightarrow p_n(t_j) = 0$ für $j = 1, 2, \dots, n$

$$\boxed{t_j\, \underline{p}(t_j) = \underline{\underline{A}}\, \underline{p}(t_j)} \quad +0$$

$\Rightarrow$ t_j Eigenwert der Matrix $\underline{\underline{A}}$

$\underline{p}(t_j)$ Eigenvektor der Matrix $\underline{\underline{A}}$

$$\underline{p}(t_j) = \begin{bmatrix} p_0(t_j) \\ p_1(t_j) \\ \vdots \\ p_{n-1}(t_j) \end{bmatrix}$$

$\text{eig}(\underline{\underline{A}})$

$\underline{\underline{A}}$ symmetrisch.

$\boxed{\text{eigh}(\underline{\underline{A}})}$

Bem Es gibt ziemlich gute numerische Verfahren um EW-Probleme zu lösen

```python
from numpy import arange, diag, sqrt
from numpy.linalg import eigh

def gaussquad(n):
    r"""
        Compute nodes and weights for Gauss-Legendre quadrature.

        n:  Number of node-weight pairs
        """
    i = arange(n)   #i = array([0,1,...,n-1])
    b = (i+1) / sqrt(4*(i+1)**2 - 1)
    # now we generate the matrix; it is symmetric
    J = diag(b, -1) + diag(b, 1)
    # in order to find the eigenvalues we can use eigh since J is symmetric
    x, ev = eigh(J)
    # finally, we apply the formula for the weights
    w = 2 * ev[0,:]**2
    return x, w
```

auf $[-1,1]$

```python
from sympy import *

x = Symbol("x")
n = Symbol("n", positive=True)
# First we generate two streams of symbols corresponding to knots and weights
xigen = numbered_symbols(prefix="xi", start=1)
omegagen = numbered_symbols(prefix="omega", start=1)
# Now we specify how many parameters we need.  In this case, two knots and two
weights.  Therefore we set N = 2
N = 2  # Choose <= 3
xis = [ next(xigen) for i in range(N) ]
wis = [ next(omegagen) for i in range(N) ]
# Using sympy one can perform simple symbolic integrations, which deliver the
exact result.
# In order to do this we use the function "integrate"
# For every n in the interval [0, 2*N-1] we set the condition that the result of
the numerical integration
#be equal to the analytical value
eqns = [ integrate(x**n, (x, -1, 1)) - (sum([wi*xi**n for xi,wi in
zip(xis,wis)])) for n in range(2*N)]
pprint(eqns)
# Therefore we get a system of equations which we need to solve:
#we set eqns = 0 and find the corresponding knots and weights
sols = solve(eqns, numerical=True)
pprint(sols)
```

Berechnung der Gewichte für Gauss-Quadratur auf $[-1,1]$

$$\langle P_i, P_k \rangle = \int_{-1}^{1} \underbrace{P_i(x) P_k(x)}_{\text{Grad} \le 2n-2} dx = \sum_{j=1}^{n} P_i(t_j) P_k(t_j) w_j$$

$\langle P_i, P_k \rangle = \begin{cases} 0, & i \neq k \\ 1, & i = k \end{cases}$

Konstruktion: Gram-Schmidt

QF exakt bis Grad $2n-1$.

$$\underline{f}(x) = \begin{bmatrix} P_0(x) \\ P_1(x) \\ \vdots \\ P_{n-1}(x) \end{bmatrix} \quad \text{in } t_1, \dots, t_n \text{ ausgewertet}$$

Notiere

$$\underline{\underline{M}} = \left[\underline{f}(t_1) \; \underline{f}(t_2) \dots \underline{f}(t_n) \right] \Longrightarrow$$

$$\underline{\underline{I}} = \underline{\underline{M}} \, \mathrm{diag}(w_1, \dots, w_n) \, \underline{\underline{M}}^T$$

$\Rightarrow \underline{\underline{M}}$ invertierbar

$\underline{\underline{M}}^{-1}$ | | $\underline{\underline{M}}^{-T}$

$\Rightarrow$

$$\mathrm{diag}(w_1, \dots, w_n) = (\underline{\underline{M}}^T \underline{\underline{M}})^{-1} \Rightarrow$$

$$\mathrm{diag}(w_1, \dots, w_n)^{-1} = \underline{\underline{M}}^T \underline{\underline{M}} \Rightarrow$$

$$\frac{1}{w_j} = \underline{f}(t_j)^T \underline{f}(t_j) = \| \underline{f}(t_j) \|^2 = \sum_{k=0}^{n-1} P_k(t_j)^2$$

oder $$w_j = \frac{1}{\| \underline{f}(t_j) \|^2}$$

ABER Die Eigenvektoren sind nicht eindeutig. ausserdem liefert eig normierte EV.

Sei $\underline{v}^j$ ein Eigenvektor $\Rightarrow$ es gibt c Konstante so dass

$$\underline{v}^j = c\,\underline{f}(t_j) = c \begin{bmatrix} p_0(t_j) \\ \vdots \\ p_{n-1}(t_j) \end{bmatrix}$$

$$1 = \langle p_0, p_0 \rangle = \int_{-1}^{1} p_0(t_1)\, p_0(t_1)\, dx \quad \Rightarrow \quad p_0(t_1) = \frac{1}{\sqrt{2}}$$

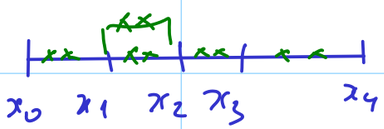
$$(\underline{v}^j)_1 = c \cdot p_0(t_1) = c \cdot \frac{1}{\sqrt{2}} \quad \Rightarrow \quad c = \sqrt{2}\,(\underline{v}^j)_1$$

$$\underline{f}(t_j) = \frac{1}{c}\,\underline{v}^j = \frac{1}{\sqrt{2}\,(\underline{v}^j)_1}\,\underline{v}^j \quad \Rightarrow$$

$$\omega_j^{-1} = \| \underline{f}(t_j) \|^2 = \frac{1}{2} \frac{\|\underline{v}^j\|^2}{(v_1^j)^2} \quad \Rightarrow$$

$$w_j = \frac{2\,(\underline{v}^j)_1^2}{\|\underline{v}^j\|^2} \; ; \text{ eig liefert normiertes } v \Rightarrow$$

$$\Rightarrow w_j = 2\,(\underline{v}_1^j)^2$$



Bem 1) Gauss-Knoten sind nicht verschachtelt
$\Rightarrow$ Teuer bei Adaptivität, ...

2) Endpunkte sind keine Knoten $\Rightarrow$
Gauss-QF ist offen.

3) Manchmal braucht man ein oder beide Endpunkte des Integrationsintervals als Knoten $\Rightarrow$
1 Knoten fest $\Rightarrow$ max Ordnung $2N-1$ (Radau)
2 Knoten fest $\Rightarrow$ max Ordnung $2N-2$ (Lobatto)

4) Fehler bei Gauss-Quadratur:

$$\int_a^b f(x)\,dx - \underbrace{\sum_{j=1}^{N} b_j f(c_j)}_{G_N(f,a,b)} = \frac{f^{(2N)}(\xi)}{(2N)!} \quad \text{mit } a < \xi < b$$

kombiniert:

$$\left| \int_a^b f(x)\,dx - \sum_{k=1}^{M} G_N(f, x_{k-1}, x_k) \right| \le c \cdot h^{2N} \max_{\xi \in [a,b]} |f^{(2N)}(\xi)| .$$

Zusammenfassung

$$\int_a^b f(x)\,dx \approx \sum_{j=1}^{p} b_j f(c_j)$$

Def Quadraturformel hat Ordnung n wenn sie Polynome vom Grad maximal $n-1$ exakt integriert

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

$\leq (b-a) \cdot C \cdot h^{n} \max_{x\in[a,b]} |f^{(n)}(x)| \longrightarrow 0$ für $h \to 0$

- n: Quadraturordnung n
- $\max |f^{(n)}(x)|$: Glattheit von f
- C: kommt aus dem Peano kern. $\int_0^1 k_n(z)\,dz = \text{Konstante}$

Gegeben Knoten $x_0, x_1, \dots, x_n$

kann man $\alpha_0, \alpha_1, \dots, \alpha_n$ berechnen, so dass

$p_n(x) = \alpha_0 + \alpha_1 x + \dots + \alpha_n x^n$ erfüllt

$$p_n(x_j) = f(x_j) \quad \text{für} \quad j = 0, 1, 2, \dots, n.$$

Im Prinzip ist das das LGS.

$$\begin{bmatrix} 1 & x_0 & \dots & x_0^n \\ 1 & x_1 & \dots & x_1^n \\ 1 & x_2 & \dots & x_2^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \dots & x_n^n \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}$$

Somit $f(x) \approx p(x)$ und dann

$$\int_a^b f(x)\,dx \approx \int_a^b p(x)\,dx = \text{exakt}$$

$\Rightarrow$ das liefert letztendlich Gewichte $w_0, \dots, w_n$.

Für äquidistante Knoten $\Rightarrow$ Δ Gewichte,

die wir aus dem System finden können:

Δ Knoten auf Referenzinterval Δ Gewichte $\Big]$ so bestimmt, dass Polynome höchstes Grades exakt mit dieser QF integriert werden.

$$f(x) := p_k(x) = x^k \Rightarrow \int_{-1}^{1} P_k(x)\,dx =$$

$$= \int_{-1}^{1} x^k\,dx = \frac{1}{k+1} x^{k+1}\Big|_{-1}^{1} = \frac{1-(-1)^{k+1}}{k+1}$$

Wie hoch kann so eine Quadraturordnung sein?

2Δ Unbekannte (Knoten, Gewichte);

2Δ Gleichungen:

$$\begin{cases} \sum_{j=1}^{\Delta} b_j c_j^k = \dfrac{1-(-1)^{k+1}}{k+1} \end{cases}$$

LGS mit Unbekannten $b_1, \dots, b_\Delta$.

$$\begin{cases} \int_0^1 1\,dt = Q_\Delta(1,0,1) \\ \int_0^1 t\,dt = Q_\Delta(t,0,1) \\ \dots \quad \left(\frac{1}{3} = b_1 c_1^2 + b_2 c_2^2 + \dots\right) \\ \int_0^1 t^{2\Delta-1}\,dt = Q_\Delta(t^{2\Delta-1},0,1) \end{cases}$$

$\rightarrow$ MPR, TR, SR.

1) $n=1$ auf $[0,1]$ $\Rightarrow$ $p_1(x) = x - \frac{1}{2}$ $\Rightarrow$ $\gamma_1 = \frac{1}{2}$, $b_1 = 1$

auf $[-1,1]$ $\Rightarrow$ $p_1(x) = x$ $\Rightarrow$ $c_1 = 0$, $b_1 = 1$

$\hookrightarrow$ Gauss-Quadratur mit $n=1$ Knoten $\equiv$ MPR.

Ordnung $2n = 2 \cdot 1 = 2$.

2) $n=2$ auf $[-1,1]$: $p_2(x) = x^2 - \frac{1}{3}$ $\Rightarrow$ $\frac{3}{2}\left(x^2 - \frac{1}{3}\right)$

$\gamma_{1,2} = \pm \frac{1}{\sqrt{3}}$, $c_1 = \frac{1}{2} - \frac{\sqrt{2}}{6}$, $c_2 = \frac{1}{2} + \frac{\sqrt{2}}{6}$.

Ordnung $2n = 2 \cdot 2 = 4$

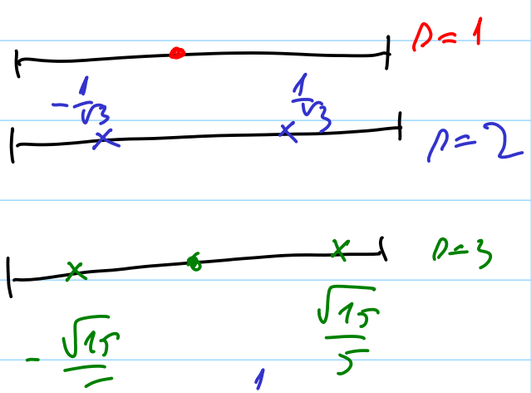
zu $p_3(1) = 1$ "normiert"

3) $n=3$ auf $[-1,1]$: $p_3(x) = \frac{5}{2}x^3 - \frac{3}{2}x$

$\gamma_2 = 0$, $\gamma_{1,3} = \pm \frac{\sqrt{15}}{5}$

$b_2 = \frac{8}{18}$, $b_1 = b_3 = \frac{5}{18}$

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



1) Gauss QF sind die QF mit maximal möglicher Ordnung.

n Knoten
n Gewichte
$\Rightarrow$ $2n$ Ordnung.

2) Falls 1 Knoten Fest

z.B. ein Ende des Intervals

$\Rightarrow$ max Ordnung $2n-1$ (Radau)

Falls 2 Knoten Fest

z.B. beide Enden des Intervals

$\Rightarrow$ max Ordnung $2n-2$ (Lobatto)

3) Gauss-Formel sind offen

4) Gauss-Knoten sind nicht verschachtelt

$\Rightarrow$ teuer bei Adaptivität

4) Fehler wie vorher.

für ein Interval:

$$\int_a^b f(x)dx - \sum_{j=1}^{n} b_j f(c_j) = \frac{f^{(2n)}(\xi)}{(2n)!}(b-a)$$

$G_n(f,a,b)$ mit $a<\xi<b$.

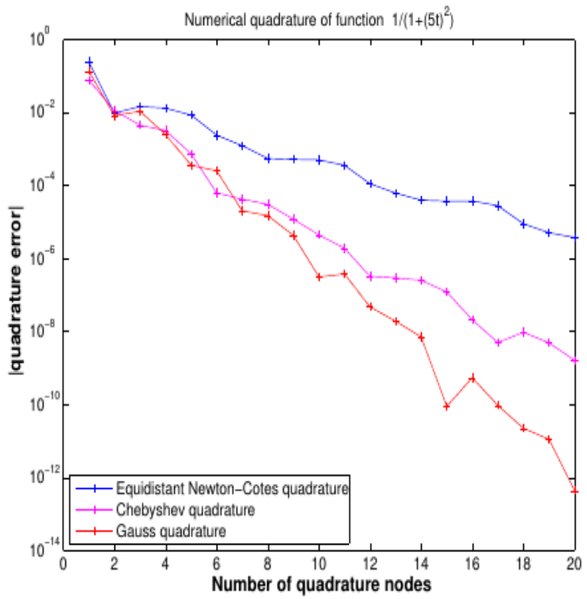
Zusammen gesetzt:

$$\left| \int_a^b f(x)dx - \sum_{k=1}^{N} G_n(f, x_{k-1}, x_k) \right| \le$$

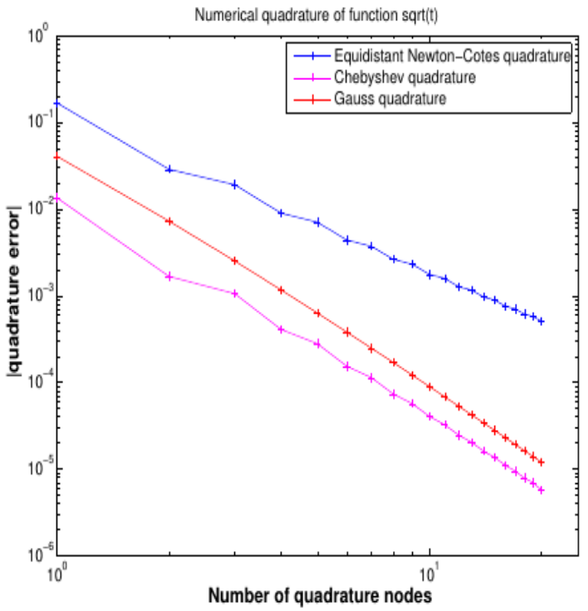
$$\le C \cdot h^{(2n)} \max |f^{(2n)}(\xi)|$$

$\xi \in [a,b]$.

Gauss-QF lohnen sich eventuell aber nur wenn f sehr glatt.



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



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

Prüfung
Numerische Methoden D-PHYS, FS 2018

2. *(8 Punkte) Lobatto-Quadratur*

Lobatto-Quadraturformeln besitzen die maximale Ordnung unter jenen Quadraturformeln, die die Intervallendpunkte als Knoten haben:

$(b_i, c_i)_{i=1}^s$ mit $c_1 = 0$, $c_s = 1$ der Ordnung $2s - 2$.

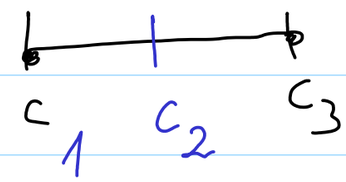
a) Geben Sie die Lobatto-Quadraturformeln der Ordnungen 2 und 4 an ($s = 2$ und $s = 3$).

b) Berechnen Sie die Knoten und Gewichte der Lobatto-Quadraturformel der Ordnung 6 ($s = 4$).

Hint: : Die Berechnung vereinfacht sich, wenn Sie verwenden, dass die gesuchte Quadraturformel symmetrisch ist.

$s=3 \Rightarrow$ Ordnung $2 \cdot 3 - 2 = 4$

$c_1 = 0, \quad c_3 = 1$



Simpson: $c_2 = \frac{1}{2}$

b) $n=4$ knoten $\Rightarrow$ Ordn. $2 \cdot 4 - 2 = 6$

a) $s=2 \Rightarrow$ Ordnung $2 \cdot 2 - 2 = 2$

$\Downarrow$

2 knoten $c_0 = 0$, $c_1 = 1$

$\Rightarrow$ Trapezregel. $b_1 = b_2 = \frac{1}{2}$

QF symmetrisch in $[0,1]$:

$$\left.\begin{array}{l} c_1 = 0, \; c_4 = 1, \; c_3 = 1 - c_2 \\ b_1 = b_4, \; b_2 = b_3 \end{array}\right\} \Rightarrow$$

3 Unbekannten: b_1, b_2, c_2

QF hat Ordnung 6 $\Rightarrow$ alle Polynome bis Grad 5 exakt integriert werden mit der QF.

Polynom von Grad 0

$$b_1 + b_2 + b_3 + b_4 = 1$$

x^2:

$$b_2 c_2^2 + b_3 c_3^2 + b_4 \cdot 1^2 = \frac{1}{3} = \int_0^1 x^2 dx$$

x^5: $b_2 c_2^4 + b_3 c_3^4 + b_4 \cdot 1^4 = \frac{1}{5} = \int_0^1 x^4 dx$

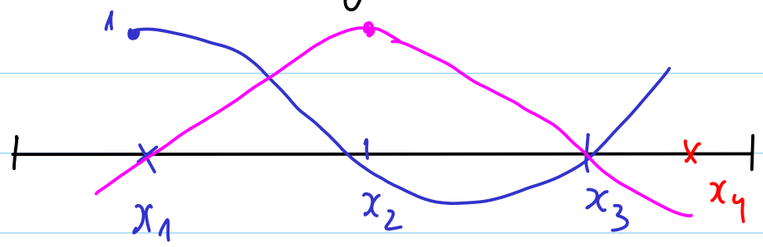
$\Rightarrow$ System

$$\begin{cases} b_1 + b_2 = \frac{1}{2} \\ b_2 c_2^2 + b_2 (1-c_2)^2 + b_4 = \frac{1}{3} \\ b_2 c_2^2 + b_2 (1-c_2)^2 + b_1 = \frac{1}{5} \end{cases}$$

```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt

def f(v):
        x = v[0]; y = v[1]; z = v[2]
        r1 = x+y-0.5
        r2 = x+ y*(z**2 + (1-z)**2) - 1./3.
        r3 = x + y*(z**4 +(1-z)**4) - 1./5.
        res = np.array([r1,r2,r3])
        return res

b1,b2, c2 = optimize.fsolve(f,[0.24, 0.25, 0.3])
print(b1,b2,c2)
# 0.0833333333326 0.416666666667 0.276393202253
```

Freitag 13.03.2020 Fragestunde (m)

Was noch unklar geblieben ist...



$$\ell_1(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}$$

$$\ell_1(x_1) = \frac{x_1-x_2}{x_1-x_2}\cdot\frac{x_1-x_3}{x_1-x_3} = 1$$

$$\ell_1(x_2) = 0 = \ell_1(x_3)$$

$$\ell_2(x) = \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}$$

$$\ell_3(x) = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

ℓ_1, ℓ_2, ℓ_3 begründet von x_1, x_2, x_3

Für x_1, x_2, x_3, x_4 neue Polynome:

$$\ell_1(x) = \frac{(x-x_2)(x-x_3)(x-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)}$$

$\underline{f}, \underline{g} \in \mathbb{R}^3$ $\quad \underline{f}\cdot\underline{g} = f_1g_1 + f_2g_2 + f_3g_3$

$\underline{f}, \underline{g} \in \mathbb{R}^d$ $\quad \underline{f}\cdot\underline{g} = \sum_{j=1}^{d} f_j g_j$

$f, g : [0,1] \to \mathbb{R}$ $\quad \langle f, g\rangle = \int_0^1 f(x)g(x)\,dx$

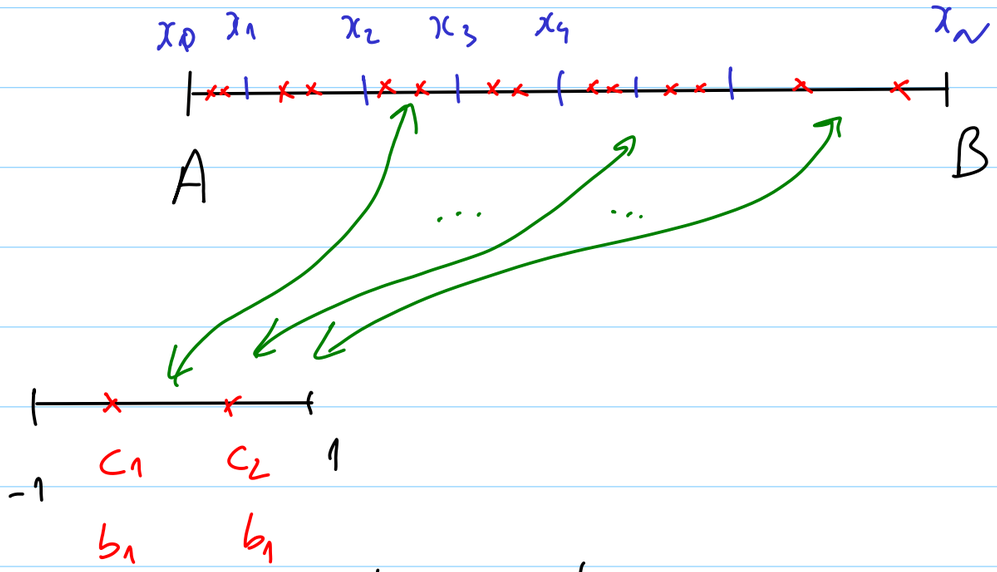
$P_n = \{$ Polynome vom Grad $\leq n-1\}$

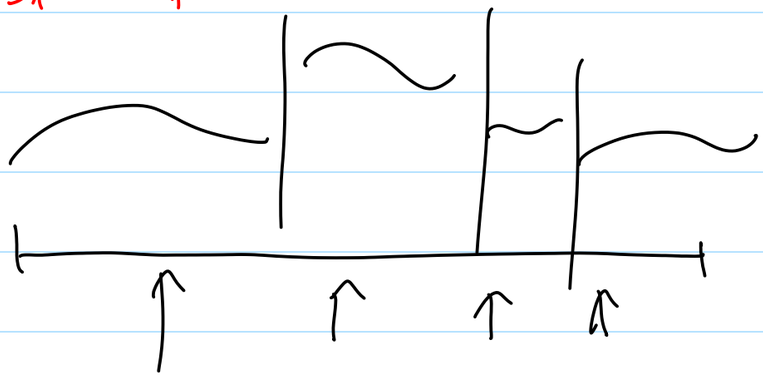
↳ Basis von Monome $1, x, x^2, \dots, x^{n-1}$

aber nicht orthogonal für $\langle\cdot,\cdot\rangle$

↳ Gram-Schmid

ONB von Polynome.





jl Gen.

$$\dot{y} = f(t,y) \qquad \text{ODE 1. Ordnung.}$$

$$\dddot{w} + 2\ddot{w} + t\dot{w} = \sin t + w$$

$$y_1 = w \qquad y_2 = \dot{w} \qquad y_3 = \ddot{w}$$

$$\Rightarrow \begin{cases} \dot{y}_1 = \dot{w} = y_2 \\ \dot{y}_2 = \ddot{w} = y_3 \\ \dot{y}_3 = \dddot{w} = -2y_3 - t y_2 + \sin t + y_1 \end{cases}$$

$$\dot{\underline{y}} = \underline{f}(t,\underline{y}) \qquad \underline{f}(t,\underline{y}) = \begin{bmatrix} y_2 \\ y_3 \\ -2y_3 - t y_2 + y_1 + \sin t \end{bmatrix}$$

↳ ODE 1. Ordnung, nicht-autonom.

autonomisieren: $z = t \qquad \dot{z} = 1.$

$$\begin{bmatrix} \dot{z} \\ \dot{\underline{y}} \end{bmatrix} = \begin{bmatrix} 1 \\ \swarrow \begin{matrix} y_2 \\ y_3 \end{matrix} \\ -2y_3 - z\,y_2 + y_1 + \sin z \end{bmatrix}$$

↳ ODE 1. Ordnung. autonom.

eE

$$\underline{y}_{k+1} = \underline{y}_k + h\,\underline{f}(\underline{y}_k) \quad \in \mathbb{R}^4$$

def f(v):

$$v = \begin{bmatrix} t \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

return $\begin{bmatrix} 1 \\ y_2 \\ y_3 \\ -2y_3 - t y_2 + y_1 + \sin(t) \end{bmatrix}$

iE

$$\underline{y}_{k+1} = \underline{y}_k + h\,\underline{f}(\underline{y}_{k+1}) \iff$$

$$\underline{y}_{k+1} - \underline{y}_k - h\,\underline{f}(\underline{y}_{k+1}) = 0$$

$$F(v) = v - y_k - h f(v)$$

solve: $F(v) = 0$

↳ v = Ergebn's. als $\underline{y}_{k+1}$

Freitag 20.03.2020 Fragestunde

$H : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ Hamilton-Function

$H(\underline{p}, \underline{q}) \in \mathbb{R}$

Hamilton' System:

$$\begin{cases} \dot{p}_j = -\dfrac{\partial H}{\partial q_j}(\underline{p}(t), \underline{q}(t)) \\ \dot{q}_j = \dfrac{\partial H}{\partial p_j}(\underline{p}(t), \underline{q}(t)) \end{cases}$$

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

$\underline{p}, \underline{q} : [0, T] \to \mathbb{R}^d$

$$\underline{y} = \begin{bmatrix} \underline{p} \\ \underline{q} \end{bmatrix}; \quad \dot{\underline{y}} = \begin{bmatrix} 0 & -I \\ I & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial H}{\partial \underline{p}} \\ \frac{\partial H}{\partial \underline{q}} \end{bmatrix}$$

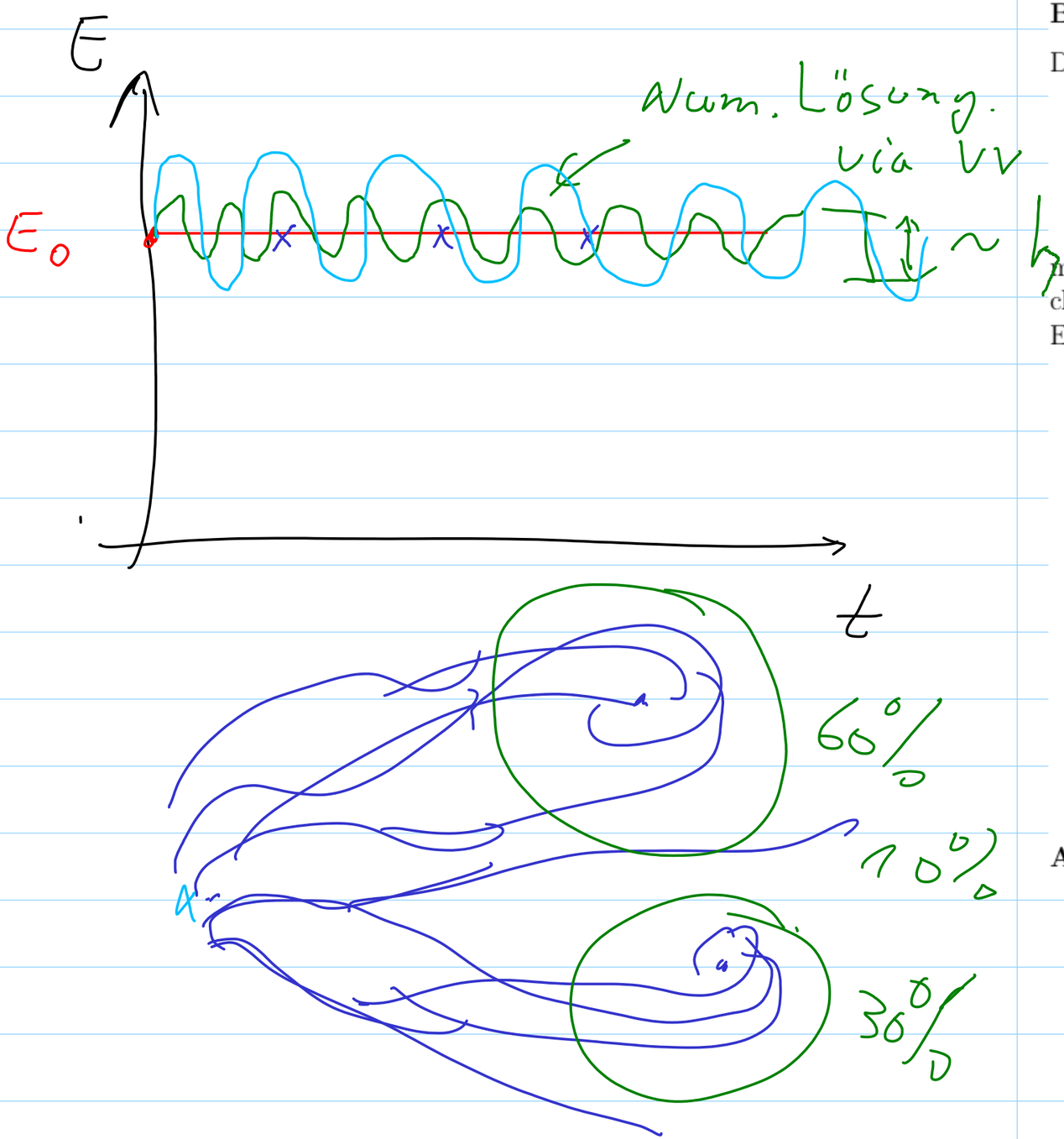
$$\underline{\underline{J}} = \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{I}}_d \\ -\underline{\underline{I}}_d & \underline{\underline{0}} \end{bmatrix} = \left[\begin{array}{cc|cc} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{array}\right]$$

$$\underline{\underline{J}} \cdot \underline{\underline{J}} = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} =$$

$$\frac{\partial H}{\partial \underline{q}} = \begin{bmatrix} \frac{\partial H}{\partial q_1} \\ \vdots \\ \frac{\partial H}{\partial q_d} \end{bmatrix} = \begin{bmatrix} -I & 0 \\ 0 & -I \end{bmatrix} = -\underline{\underline{I}}_{2d}$$

$$(-\underline{\underline{J}}) \cdot \underline{\underline{J}} = I_{2d} \Rightarrow \underline{\underline{J}}^{-1} = -\underline{\underline{J}}$$

$$= \begin{bmatrix} 0 & -I \\ I & 0 \end{bmatrix}$$



Beispiel 2.1.23. (Lorenz-System)

Die autonome Differentialgleichung

$$\begin{aligned} \dot{x} &= \sigma(y-x)\,, \\ \dot{y} &= x(\rho-z)-y\,, \\ \dot{z} &= xy-\beta z\,, \end{aligned} \tag{2.1.10}$$

mit $D=\mathbb{R}^3$ und Konstanten $\sigma,\rho,\beta\in\mathbb{R}^+$ ist das typische Beispiel eines chaotischen Systems. Kleine Mess- oder Rundungsfehler wirken sich in unvorhersehbaren Evolutionen.

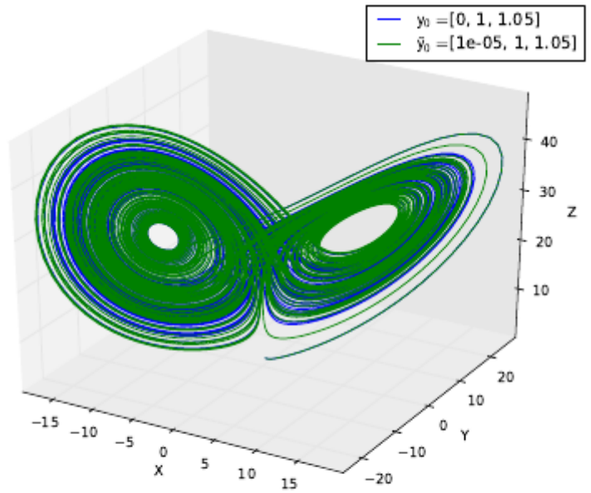


Abb. 2.1.24. Zwei Lösungskurven des Lorenz-Systems für $\sigma=10, \rho=28, \beta=8/3$.

5. *(8 Punkte): Modifizierte Euler Verfahren*

a) Begründen Sie das modifizierte explizite Euler-Verfahren

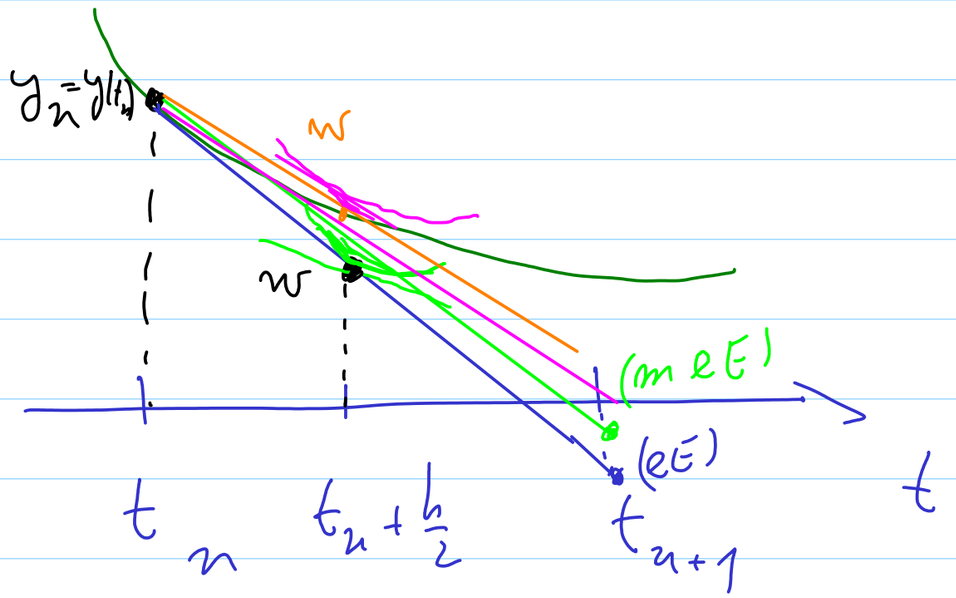
$$\begin{aligned} w &= y_n + \frac{h}{2} f(t_n, y_n) \\ y_{n+1} &= y_n + h f(t_n + \frac{h}{2}, w) \end{aligned} \tag{9}$$

und das modifizierte implizite Euler-Verfahren

$$\begin{aligned} w &= y_n + \frac{h}{2} f(t_n + \frac{h}{2}, w) \\ y_{n+1} &= y_n + h f(t_n + \frac{h}{2}, w) \end{aligned} \tag{10}$$

als Polygonzugverfahren zur numerischen Lösung einer gewöhnlicher Differentialgleichung

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



(m ∈ E): $\frac{y_{n+1} - y_n}{h} = f(t_n + \frac{h}{2}, w)$

steigung an der Traj. in $(t_n + \frac{h}{2}, w)$

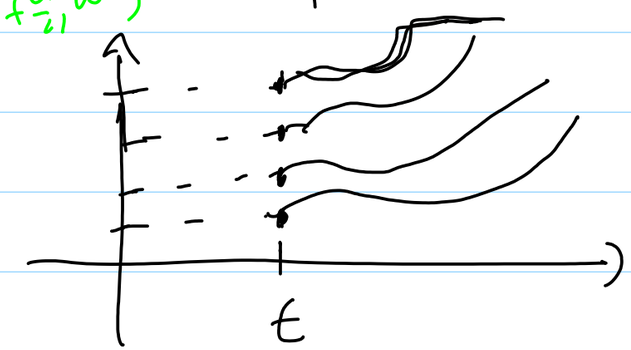
$w = y_n + \frac{h}{2} f(t_n, y_n)$

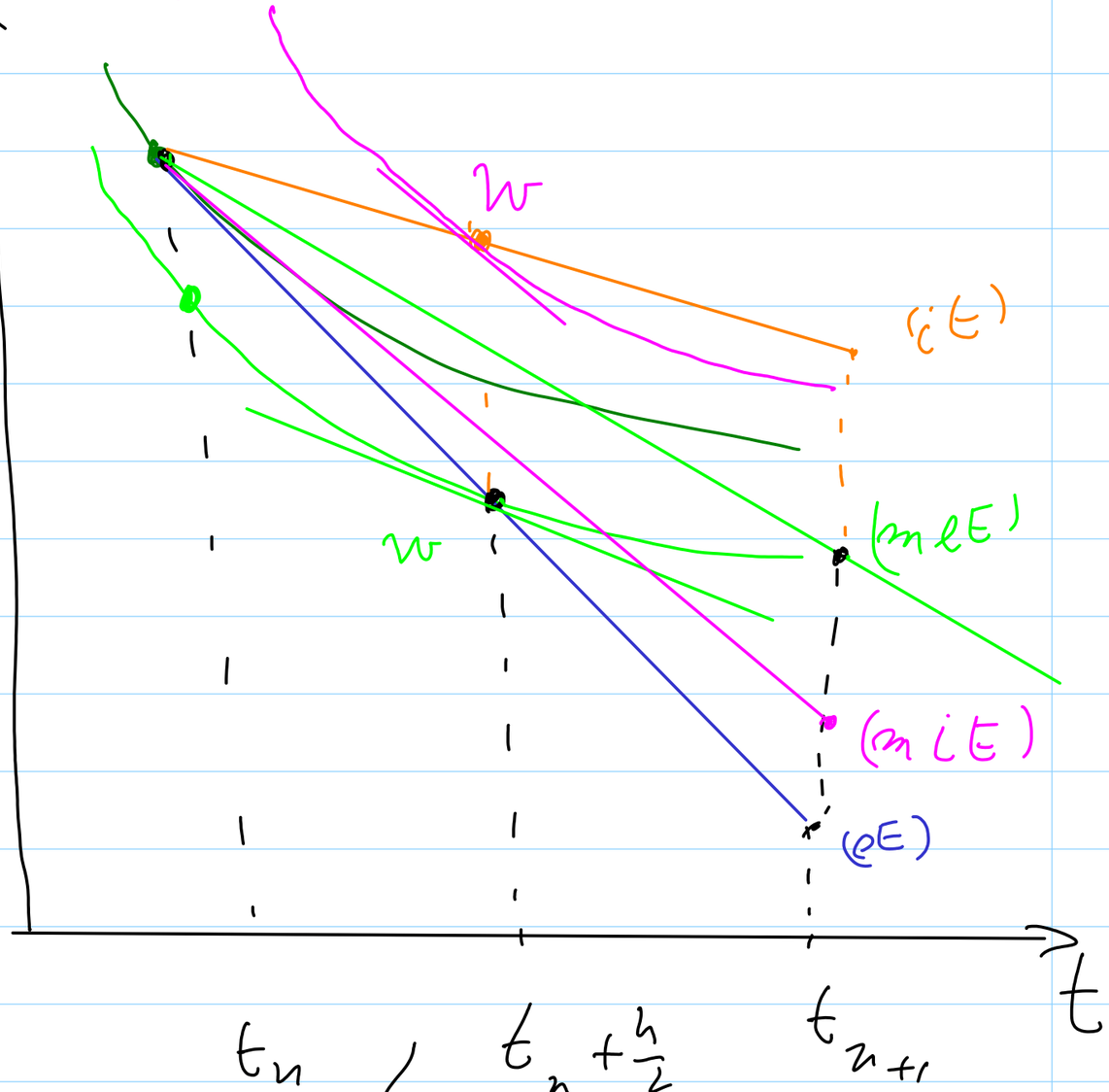
e ∈ mit $h/2$ statt

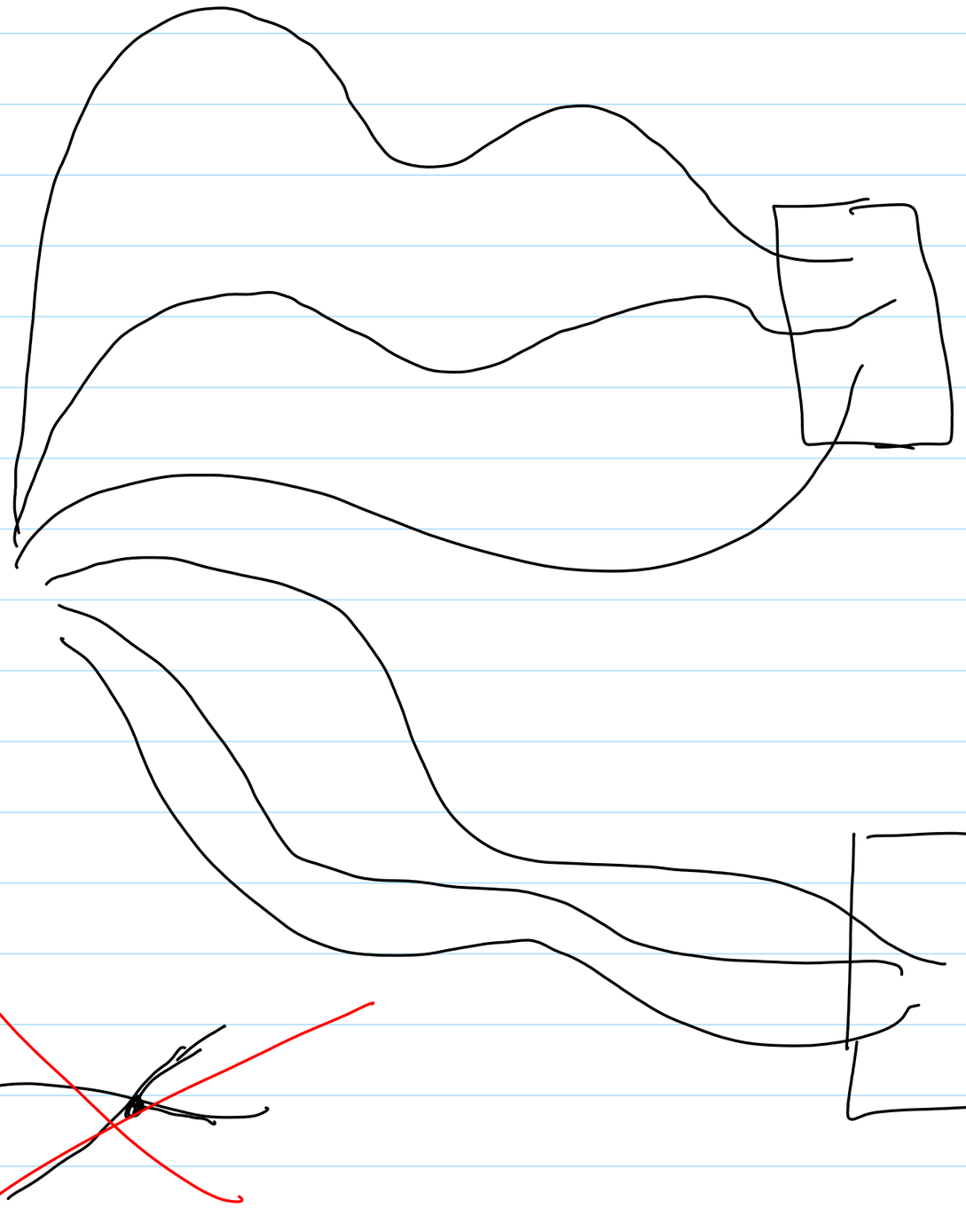
(m ∈ E): $w = y_n + \frac{h}{2} f(t_n + \frac{h}{2}, w)$

↳ i ∈ mit $h/2$

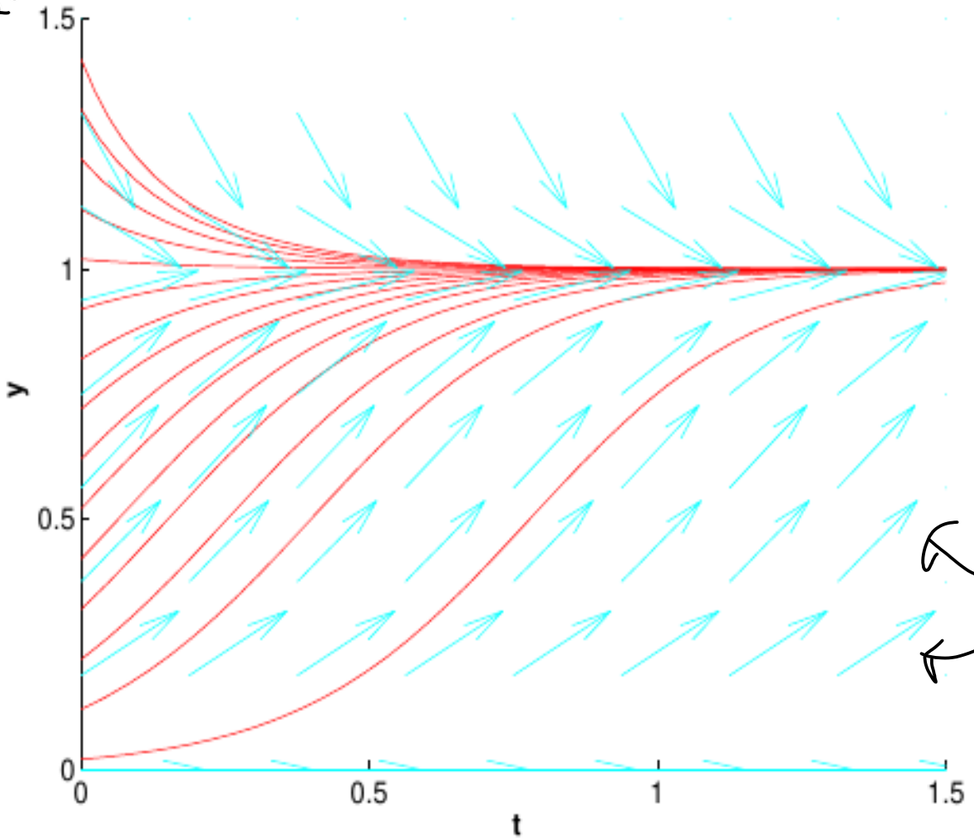
$y_{n+1} = y_n + h f(t_n + \frac{h}{2}, w)$



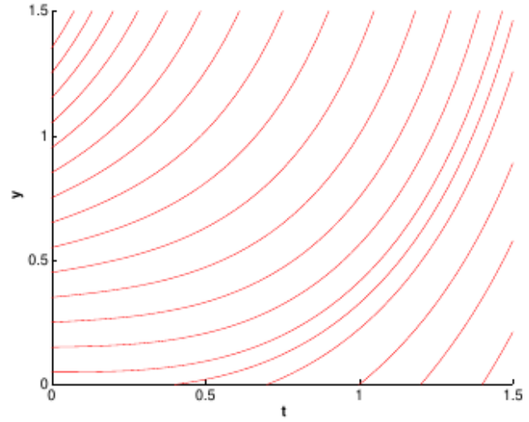




viele Trajektorien



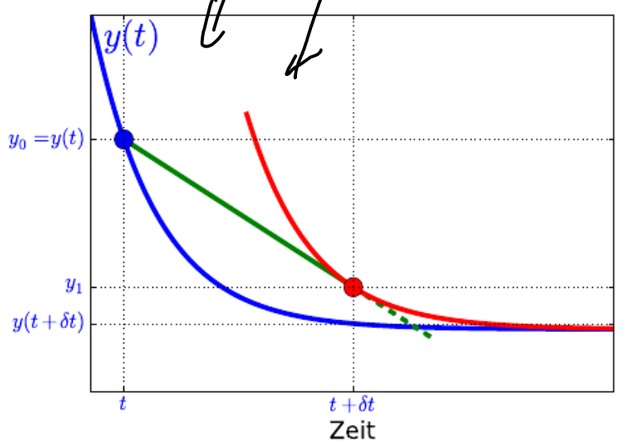
← Richtungsfeld



Lösungskurven

Abb. 2.1.16. Richtungsfeld und Lösungskurven der Gleichung (2.1.6) für $\alpha, \beta = 5$

2 Trajektorien

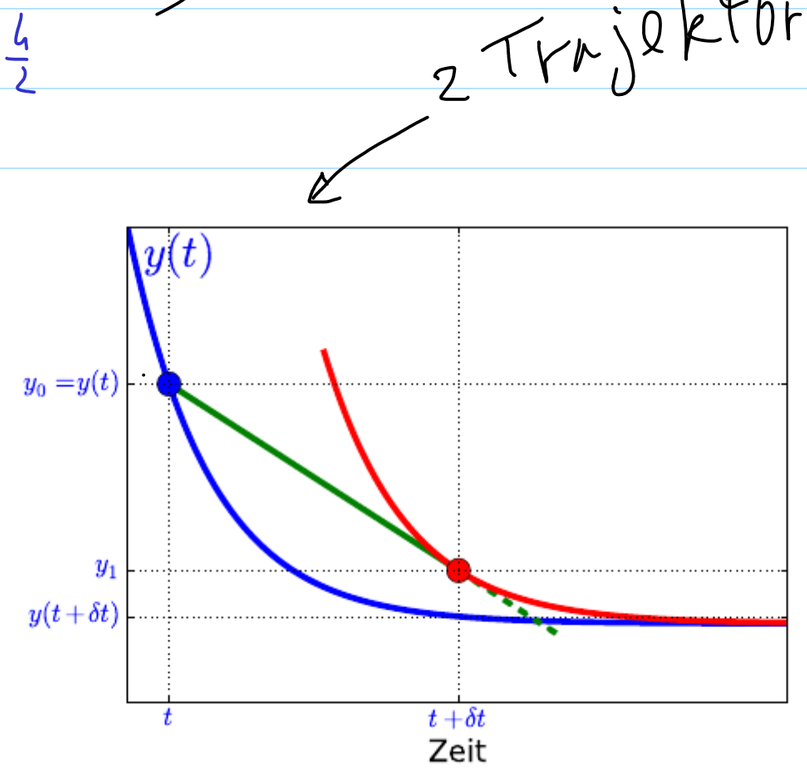


b) Beweisen Sie dass das (mi E) $\rightsquigarrow$ (imp)

(m i E) identisch mit der (imp) ist.

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

(m iE): $\begin{cases} w = y_n + \frac{h}{2} f(t_n + \frac{h}{2}, w) \\ y_{n+1} = y_n + h f(t_n + \frac{h}{2}, w) \end{cases}$



(m i E)

$$y_{n+1} = y_n + h f\left(t_n + \frac{h}{2}, w\right) \Rightarrow$$

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

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

$$\Rightarrow \quad w = y_n + \frac{1}{2}\left(y_{n+1} - y_n\right) \Rightarrow$$

$$w = \frac{y_n}{2} + \frac{y_{n+1}}{2} = \frac{y_n + y_{n+1}}{2}$$

$$\downarrow$$

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

$$\Rightarrow \quad (\text{i } \Pi P)$$

Freitag 27.03.2020 Frageruncle

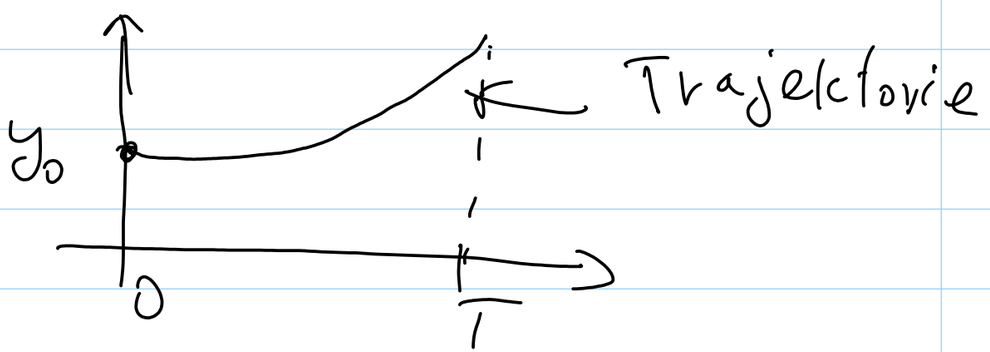
1) Idee der Fluss der ODE.

$\dot{y} = f(y)$ Diff.-Gleichung.
1. Ordnung, autonom.
$f: \mathbb{R} \to \mathbb{R}$

+ Anfangswert $y(0) = y_0$

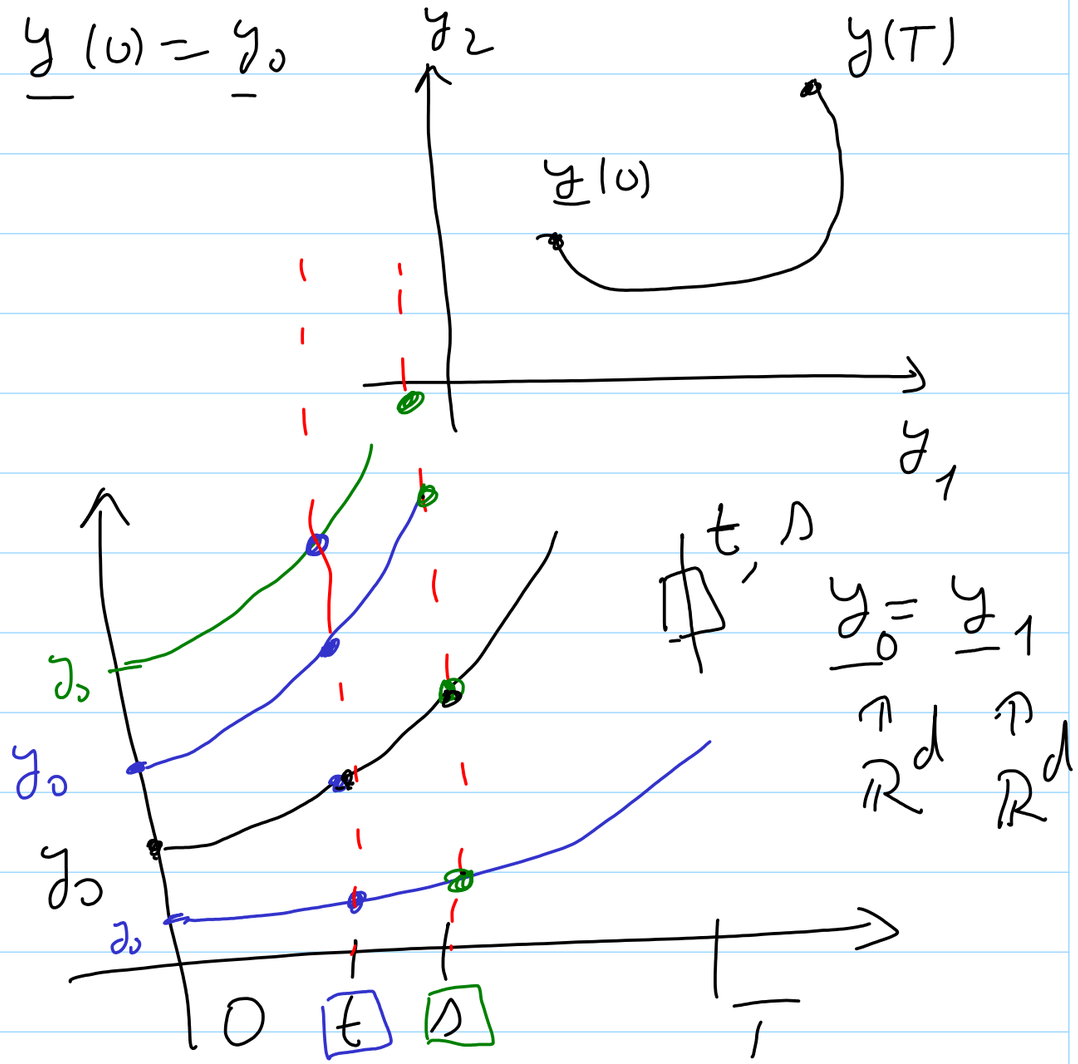
$\Rightarrow$ eine Lösung

$y : [0, T] \to \mathbb{R}$



$\underline{\dot{y}} = \underline{f}(\underline{y})$ $\quad \underline{f}: \mathbb{R}^2 \to \mathbb{R}^2$

$\underline{y}(0) = \underline{y}_0$



$\dot{y} = f(y) + g(y) \qquad f, g : \mathbb{R}^d \to \mathbb{R}^d.$

$\Phi^{t,s}_{f+g} : \mathbb{R}^d \to \mathbb{R}^d$

$\Phi^{t,s}_{f+g}(y_0) = y_1 \in \mathbb{R}^d$

$\hookrightarrow$

$y_1 = y(1)$ wobei

$y : [0,T] \to \mathbb{R}^d$

$y(t) = y_0$ Startwert

y erfüllt: ODE

$\dot{y} = f(y) + g(y)$

$\dot{y} = f(y)$

$\Phi^{t,s}_{f}(y_0) = y_1 \in \mathbb{R}^d$

$y_1 = y(1)$ wobei

$y(t) = y_0$ und

$\dot{y} = f(y)$

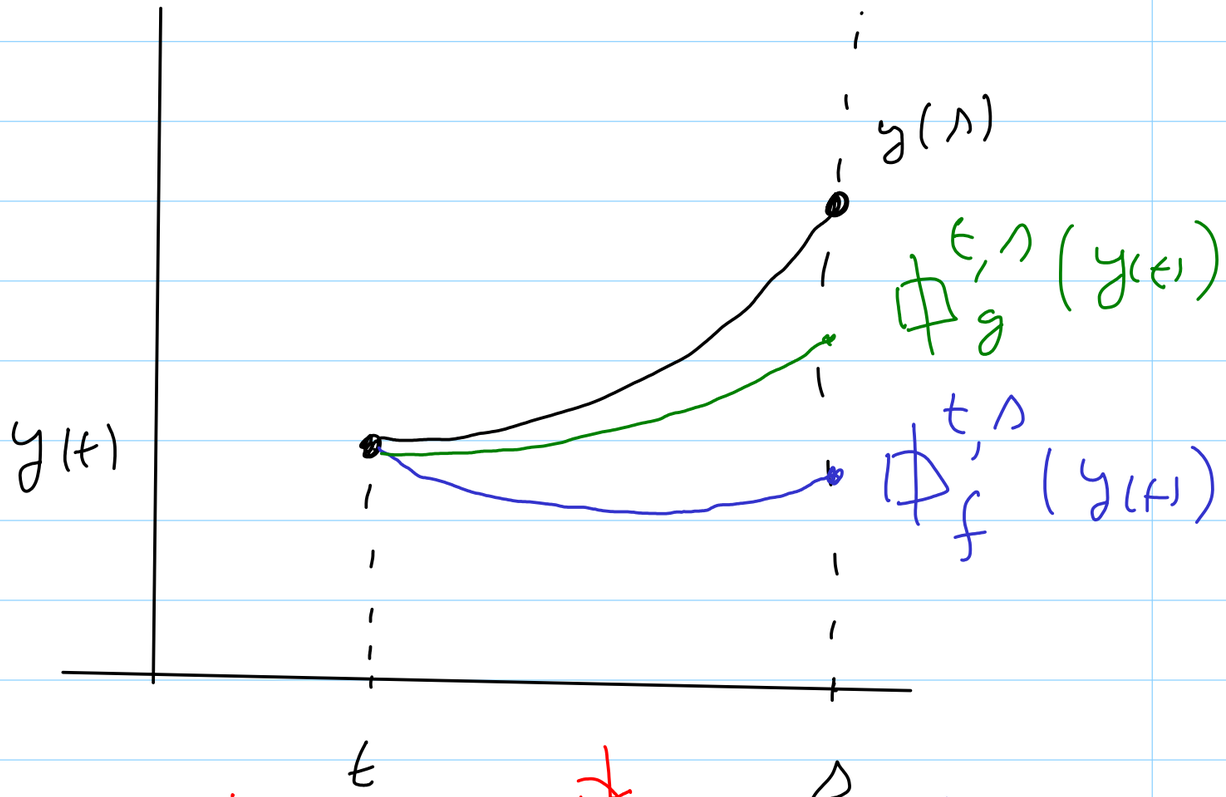
$\dot{y} = g(y) \qquad y : [0,T] \to \mathbb{R}^d$

$y(t) \in \mathbb{R}^d$

$\Phi^{t,1}_{g}(y_0) = y_1 \in \mathbb{R}^d$

$y_1 = \overbrace{y(1)}^{\text{Wert} \in \mathbb{R}^d}$ wobei

$y(t) = y_0$ und $\dot{y} = g(y)$



$\dot{y} = M y = (A + B) y$

autonome ODE : Zeit verschiebe.

$\Rightarrow$ immer $t_0 = 0$

$$\Phi_f^{t,s} = \Phi_f^{0,s-t} = \Phi_f^{s-t}$$

$$y(t) = e^{Mt} y_0 \quad \text{Lösf}$$

$\dot{y} = My$ mit $y(0) = y_0$

$$y(t) = e^{At} y$$

$\rightarrow \psi_{\sin\alpha}$

Φ_α

Φ_{rest}

$\sin\alpha = \alpha + O(\alpha^3)$

$\sin\alpha \approx \alpha \Rightarrow$ exakte Lösung.

$f + g \approx f \quad ; \Phi_{f+g} \approx \Phi_f$

$\Phi_M^t = e^{Mt}$; $\Phi_A^t = e^{At}$; $\Phi_B^t = e^{Bt}$

$M = A + B$

e^{Mt} , e^{At} , e^{Bt}

$\Rightarrow$ $\Phi_A^t \Phi_B^t \approx \Phi_{A+B}^t$

$\Phi_B^t \Phi_A^t \approx \Phi_{A+B}^t$

$O(h^2) \Rightarrow$ global $O(h)$

Erhöhe Ordng. indem man andere Hintereinander-ausführungen der Φ_A , Φ_B macht

$\Phi_B^{t/2} \Phi_A^t \Phi_B^{t/2} \approx \Phi_{A+B}^t$ $O(h^3)$

$\Downarrow$

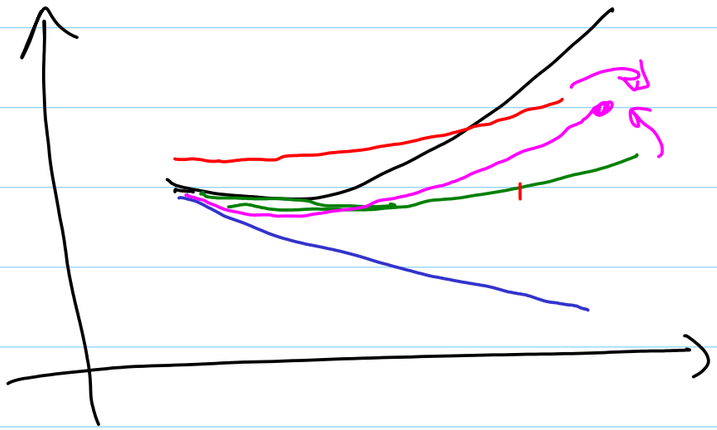
global $O(h^2)$.

$\prod_{i=1}^{n} \Phi_B^{b_i h} \Phi_A^{a_i h} \approx \Phi_{A+B}^h$

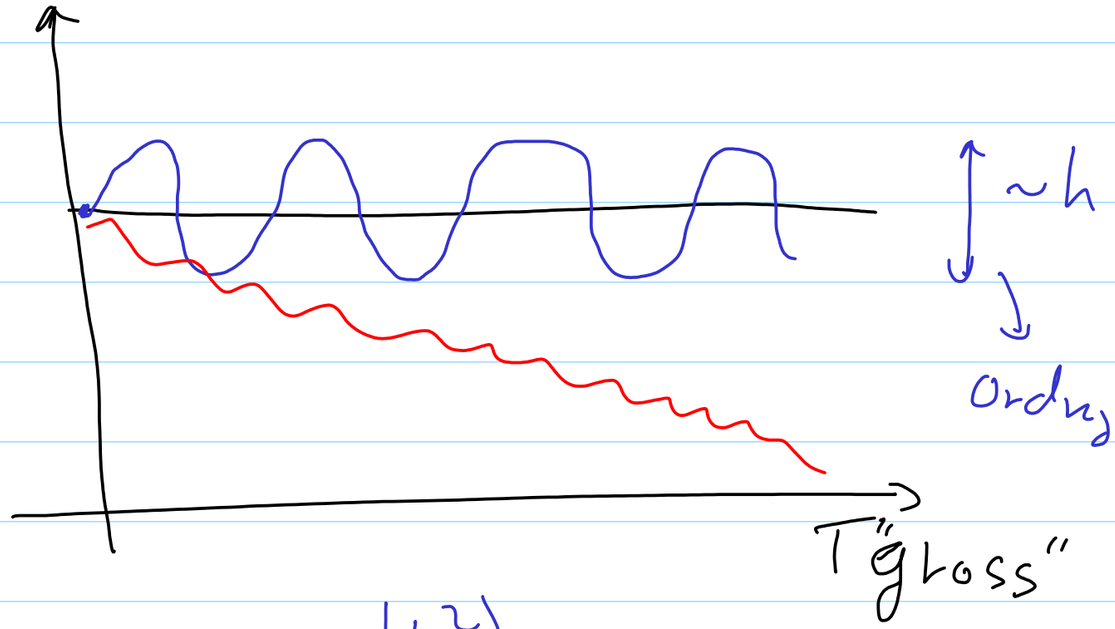
$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i = 1$

LT: $n=1$ $b_1 = a_1 = 1$

SS: $n=2$ $b_1 = \frac{1}{2} = b_2$, $a_1 = 1$, $a_2 = 0$



$$\ldots \circ \Phi_B^{b_2 h} \circ \Phi_A^{a_2 h} \circ \Phi_B^{b_1 h} \circ \Phi_A^{a_1 h}$$



2) Bleibt die Energie erhalten.

Nein!

Exakt erhalten bleibt (IMP) nur wenn die Energie quadratisch.

$$E(y) = \tfrac{1}{2} y^T \Pi y \Rightarrow E(y_1) = E(y_2)$$

∈ Gauss-Kollokationsverfahren.

Die Differentialgleichung für den Auslenkungswinkel θ des invertierten Federpendels im Bild ist

$$\ddot{\theta} = -\frac{c}{mL^2}\theta + \frac{g}{L}\sin\theta\,. \qquad (4)$$

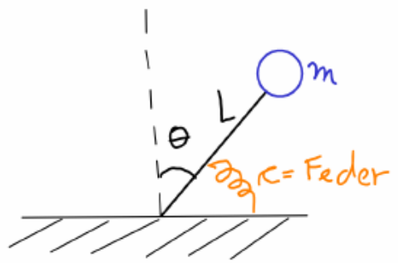


Abbildung 2 – Federpendel.

Die physikalischen Konstanten sind im File angegeben. Die Endzeit ist im Folgenden $T = 1000$.

a) Schreiben Sie die Gleichung als Hamilton-System und geben Sie die Hamilton Funktion an.

b) Benutzen Sie das Hamilton-System, um die Gleichung mit $N = 5000$ Zeitschritten des Störmer-Verlet Verfahrens zu lösen.
Hinweis: das Strömer-Verlet Verfahren kann auch als Splitting Verfahren verstanden werden. Damit können die implementierte Funktionen `Approximate`, `splitting`, `splitting_step` und `integrate` nützlich sein.

c) Benutzen Sie das Hamilton-System, um die Gleichung mit $N = 2500$ Zeitschritten eines symplektischen partitionierten Runge-Kutta (spRK) Verfahren der Ordnung 6 zu lösen.

d) Plotten Sie in linlog-Skala die Abweichungen der Totalenergie für die Approximationen aus (b) und (c). Im template `Federpendel.py` wird das Plot für `ode45` bereits gegeben. Erklären Sie das beobachtete Verhalten für alle drei Methoden.

Hinweis: Die Funktion `semilogy` unter `matplotlib.pyplot` kann nützlich sein.

e) Schreiben Sie welcher dieser drei Lösungen Sie am meisten und welcher Sie am wenigsten trauen und begründen Sie Ihre Wahl.

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

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

$$\begin{cases} \dot{q} = \dot{\theta} = p = \frac{\partial H}{\partial p} \\ \dot{p} = -\frac{c}{mL^2}\theta + \frac{g}{L}\sin\theta = -\frac{c}{mL^2}q + \frac{g}{L}\sin q = -\frac{\partial H}{\partial q} \end{cases}$$

$$\Rightarrow H(q,p) = \frac{1}{2}p^2 + \frac{c}{2mL^2}q^2 + \frac{g}{L}\cos(q)$$

$$= \frac{1}{mL^2}E_{tot}$$

$$E_{kin} = \frac{1}{2}mL^2p^2, \quad V(q) = \frac{1}{2}cq^2 + gmL\cos(q).$$

$$\begin{cases} \dot q = p \\ \dot p = \underbrace{-\frac{c}{mL^2}}_{A} q + \underbrace{\frac{g}{L}}_{B} \sin q \end{cases} = \begin{bmatrix} p \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ Aq + B\sin q \end{bmatrix}$$

2) $\begin{cases} \dot q = 0 \\ \dot p = Aq + B\sin q \end{cases} \Rightarrow$ $\dot q = 0 \Rightarrow q(h) = q_0$

$\Rightarrow p(h) = p_0 + (Aq_0 + B\sin q_0)h$

(A)

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

$p(h) = p_0$ (B)

phi A: $\begin{bmatrix} q_0 \\ p_0 \end{bmatrix} \to \begin{bmatrix} q_0 \\ p_0 + (Aq_0 + B\sin q_0)h \end{bmatrix}$

phi B: $\begin{bmatrix} q_0 \\ p_0 \end{bmatrix} \to \begin{bmatrix} q_0 + p_0 h \\ p_0 \end{bmatrix}$

Fragerundevom 03.04.2020

a) Schreiben Sie das explizite Euler Verfahren und die implizite Mittelpunktsregel als Runge-Kutta Verfahren; geben Sie die entsprechenden Butcher-Tabellen an.

b) Programmieren Sie die Methoden von (a) als Runge-Kutta Verfahren. Verwenden Sie diese, um jeweils eine numerische Approximation mit $N = 100$ äquidistanten Zeitschritte an der Lösung der Gleichung

$$\begin{aligned} \dot{y} &= y(t) - 2\sin(t), \qquad t \in [0,4], \\ y(0) &= 1. \end{aligned}$$

zu berechnen und zu ploten.

c) Verwenden Sie Ihren Code, um die entsprechenden Konvergenzordungen dieser zwei Methoden empirisch zu finden; ploten Sie dabei den Fehler und Referenzkurven. Die exakte Lösung ist $y(t) = \cos(t) + \sin(t)$.
Hinweis: Benutzen Sie die folgenden Schrittanzahl $N = 2^k$ mit $k = 5, \ldots, 11$.

(eE):

0 | 0	$y_1 = y_0 + h \sum_{j=1}^{1} b_j k_j$
1	

(iMP)

$$y_1 = y_0 + h f\left(t_0 + \frac{h}{2}, \frac{1}{2} y_0 + \frac{1}{2} y_1\right)$$

$s = 1, \; b_1 = 1$

k_1

$\frac{1}{2}$ | $\frac{1}{2}$	
1	

$$y_1 = y_0 + h k_1$$

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

$$k_i = f\left(t_0 + c_i h, y_0 + h \sum_{j=1}^{s} a_{ij} k_j\right)$$

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

(eE): $y_{n+1} = y_n + h f(t_n, y_n)$

(meE):
$$\begin{cases} w = y_0 + h \frac{1}{2} f(t_0, y_0) \\ y_1 = y_0 + h f(t_0 + \frac{1}{2}h, w) \end{cases}$$

$w = k_2$

Wie man dies als RK schreibt?

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

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

0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0
	0	1

(mcE)

$$y_1 = y_0 + h f(t_0 + \frac{h}{2}, w)$$

k_2

$$k_2 = f(t_0 + \frac{h}{2}, w) = \cancel{f(t_0 + \frac{h}{2}, y_0 + \frac{h}{2} f(}$$

$$= f(t_0 + \frac{h}{2}, y_0 + \frac{h}{2} f(t_0 + \frac{h}{2}, w))$$

$$w = y_0 + \frac{h}{2} f(t_0 + \frac{h}{2}, w)$$

$$= y_0 + \frac{h}{2} k_2 \Rightarrow$$

$$k_1 = f(t_0 + \frac{h}{2}, y_0 + \frac{h}{2} k_1)$$

$\frac{1}{2}$	$\frac{1}{2}$	
		1

Zusatzbesprechung vom 15.04.2020

5. *(8 Punkte): Modifizierte Euler Verfahren*

a) Begründen Sie das modifizierte explizite Euler-Verfahren

$$\begin{aligned} w &= y_n + \frac{h}{2} f(t_n, y_n) \\ y_{n+1} &= y_n + h f(t_n + \frac{h}{2}, w) \end{aligned} \tag{5}$$

und das modifizierte implizite Euler-Verfahren

$$\begin{aligned} w &= y_n + \frac{h}{2} f(t_n + \frac{h}{2}, w) \\ y_{n+1} &= y_n + h f(t_n + \frac{h}{2}, w) \end{aligned} \tag{6}$$

als Polygonzugverfahren zur numerischen Lösung einer gewöhnlicher Differentialgleichung

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

b) Beweisen Sie, dass das modifizierte implizite Euler-Verfahren identisch mit der impliziten Mittelpunktsregel ist.

c) Finden Sie die Stabilitätsfunktionen der beiden modifizierten Euler-Verfahren.

d) Bestimmen Sie empirisch die Konvergenzrate der beiden Euler-Verfahren indem Sie die Differentialgleichung

$$\dot{y} = -\sin(t) + \lambda(y - \cos(t))\,, \qquad t \in [0, 2] \tag{7}$$

mit dem Startwert $y(0) = 1$ bis zuer Endzeit $T = 2$ für $\lambda = -10$ lösen. Die exakte Lösung ist $y(t) = \cos(t)$.
Hinweis: Sie können die Funktion `compute_error` im File `Modified_Euler.py` benutzen, um den Fehler zu berechnen.

e) Betrachten Sie die mit dem modifizierten expliziten und modifizierten impliziten Euler Verfahren berechneten Lösungen dieser Gleichung für $\lambda = -2100$ mit N äquidistanten Schritten bis zur Endzeit $T = 2$.

(i) Plotten Sie die zwei Lösungen für $N = 1000$ und erklären Sie das beobachtete Verhalten,

(ii) Ab welchen N beobachten Sie einen Fehler zur Endzeit $T = 2$ kleiner als 10^{-4} für beide Verfahren?

$\dot{y} = \lambda y$, mit $\lambda \in \mathbb{C}$, $\mathrm{Re}\,\lambda < 0$

$(m \in E)$ $\quad w = y_n + \frac{h}{2}\lambda y_n$

$y_{n+1} = y_n + h\lambda w = y_n + h\lambda\left(y_n + \frac{h}{2}\lambda y_n\right)$

$= y_n + h\lambda y_n + \frac{h^2}{2}\lambda^2 y_n = \left(1 + h\lambda + (h\lambda)^2 \frac{1}{2!}\right) y_n$

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

$(m \notin E)$ $\quad w = y_n + \frac{h}{2}\lambda w \Rightarrow \left(1 - \frac{h}{2}\lambda\right) w = y_n \Rightarrow$

$\Rightarrow w = \frac{1}{1 - \frac{h}{2}\lambda}\, y_n \Rightarrow$

$y_{n+1} = y_n + h\frac{\lambda}{1 - \frac{h}{2}\lambda} y_n \Rightarrow y_{n+1} = \frac{1 + \frac{1}{2}\lambda h}{1 - \frac{1}{2}\lambda h} y_n$

$\Rightarrow \quad S(z) = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z}$

d) Code : Ordnung 2

e) $\lambda = -2100 \Rightarrow$ Stabilitätsprobleme für (mEE)

$\lambda < 1$

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

(mEE) stabil falls $|S(\lambda h)| < 1 \Rightarrow$

$\left|1 + \lambda h + \frac{1}{2}(\lambda h)^2\right| < 1$

$-1 < 1 + \lambda h + \frac{1}{2}(\lambda h)^2 < 1$

$-2 < \lambda h\left(1 + \frac{1}{2}\lambda h\right) < 0 \Rightarrow |\lambda h| < 2$

$\Rightarrow h < \frac{2}{|\lambda|} \Rightarrow N > \frac{|\lambda|}{2} T = 2100.$

Fragerunde vom 24.04.2020

1) ODE höherer Ordnung $\Rightarrow$ umschreiben in System ODE 1. Ordnung.

$\underline{\dot{y}} = \underline{f}(t, \underline{y})$ $\quad \underline{y} : [0,T] \to \mathbb{R}^d$

$\underline{f} : [0,T] \times \mathbb{R}^d \to \mathbb{R}^d$

from scipy.integrate import solve_ivp

2) Konvergenzordnung bei Nullstellenproblem.

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

Iteratives Verfahren. $\Rightarrow (\underline{x}^{(k)})_k \subset \mathbb{R}^d$

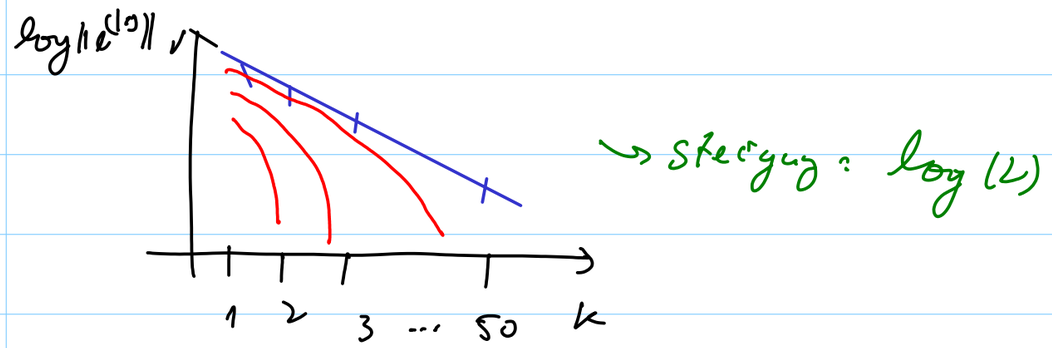
$\underline{x}^{(0)}, \underline{x}^{(1)}, \underline{x}^{(2)}, \dots, \underline{x}^{(k)}, \dots \to \underline{x}^*$

Lineare Konvergenz falls

$$\|x^{(k+1)} - x^*\| \leq L \|x^{(k)} - x^*\| \quad \text{für } k \geq k_0$$

mit $0 < L < 1$

↳ Konvergenzrate



↳ Steigung: $\log(L)$

Achtung: Dieser Begriff unterscheidet sich fundamental von der Konvergenzordnung bei ODE; Quadratur.

Konvergenzordnung p:

$$\|x^{(k+1)} - x^*\| \leq C \|x^{(k)} - x^*\|^p \quad \text{für } k \geq k_0$$

Achtung, für $p=1$ noch die Bedingung $C<1$

$$\underbrace{\|x^{(k+1)} - x^*\|}_{e^{(k+1)}} \leq L \underbrace{\|x^{(k)} - x^*\|}_{e^{(k)}} \leq L^2 \underbrace{\|x^{(k-1)} - x^*\|}_{e^{(k-1)}} \leq \dots$$

$$\|e^{(k+1)}\| \leq L^{k+1} \|e^{(0)}\|$$

$$\|e^{(k)}\| \leq L^k \|e^{(0)}\| \Rightarrow \log\|e^{(k)}\| \leq k \log L + \log\|e^{(0)}\|$$

3. *(8 Punkte) Nichtlineares System*

Das nichtlineare Gleichungssystem

$$F = \begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} \sin\left(\pi\sqrt{x^2+4y^2}\right) \\ x^2 + y - 4\sin^2(\pi x) + 1 \end{pmatrix} = \mathbf{0} \tag{3}$$

ist zu lösen.

a) Implementieren Sie im File `Nullstellensuche.py` die Funktion F, deren Nullstellen wir suchen, und die Ableitung DF, so dass diese mit `numpy.array` arbeiten können;

b) Implementieren Sie ein Newton-Verfahren in $\mathbb{R}^2$ in `newton(x0, F, DF, rtol, maxiter)` in `Nullstellensuche.py`, wobei

- `x0` der Initialwert,
- `F` die Funktion F,
- `DF` die Ableitung DF,
- `rtol` die relative Toleranz,
- `maxiter` die maximale Anzahl Iterationen,

sind. Die Funktion `newton` soll die approximierte Lösung und die Anzahl benötigten Iterationen zurückgeben.
Hinweis: Die Funktion `np.linalg.solve` kann nützlich sein.

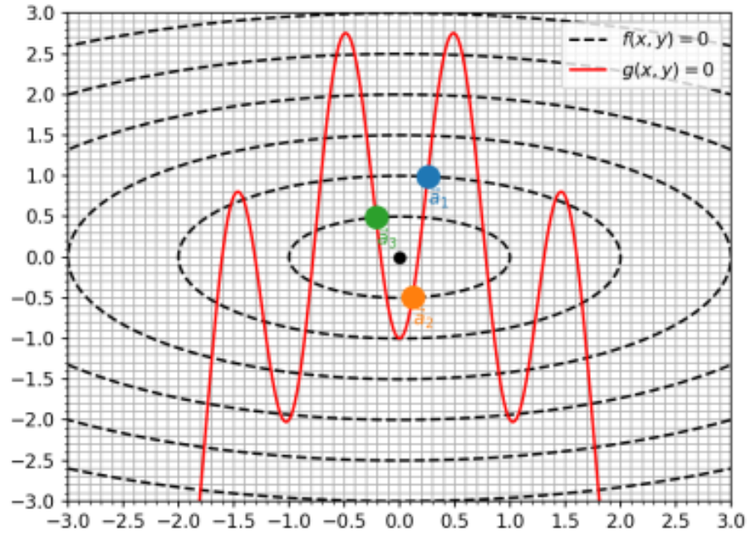


Abbildung 1 – Die Nullstellen von f und g.

c) Schauen Sie sich die Figur 1 an. Hier sind drei Nullstellen eingezeichnet.

(i) Finden Sie eine Approximation an a_2 mit der Funktion `fsolve` aus `scipy.optimize`;

(ii) Finden Sie einen Startwert x_0^2, so dass das Newton-Verfahren mit Initialwert x_0^2 in maximal 100 Schritten mit relativer Toleranz tol $= 10^{-12}$ zur eingezeichneter Nullstelle a_2 konvergiert;

(iii) Für solchen Startwert x_0^2 geben Sie a_2^{fsolve}, a_2^{newton}, $F(a_2^{\text{newton}})$ und die verwendete Anzhal Newton-Schritte N_{it} in folgender Form aus

`Start` : x_0^2
`fsolve` : a_2^{fsolve}
`Newton` : a_2^{newton}
`F` : $F(a_2^{\text{newton}})$
`Iterationen` : N_{it}
`-----------------------------------`

Hinweis: zur Verfügung steht neben das Template `Nullstellensuche.py` auch das File `plot_contour.py`, das die Abbildung 1 generiert und nicht modifiziert werden muss.

Fragerunde vom 8.05.2020

Butcher

$$\underline{\underline{A}} = \begin{bmatrix} & & \\ & & 0 \\ & & \end{bmatrix}$$

$$\begin{cases} \underline{k}_1 = f(y_0 + h a_{11} \underline{k}_1 + \dots + h a_{1s} \underline{k}_s) \\ \dots \end{cases}$$

stufen = System alg. nichtlineare Gleichungen.

$$\begin{cases} \underline{k}_1 = \underline{f}(\underline{y}_0 + h\, a_{11}\, \underline{k}_1) & \underline{k}_1 \in \mathbb{R}^d \\ \underline{k}_2 = \underline{f}(\underline{y}_0 + h\, a_{21}\, \underline{k}_1 + h\, a_{22}\, \underline{k}_2) & \underline{k}_2 \in \mathbb{R}^d \\ \underline{k}_3 = \underline{f}(\underline{y}_0 + h\, a_{31}\, \underline{k}_1 + h\, a_{32}\, \underline{k}_2 + a_{33}\, \underline{k}_3) & \underline{k}_3 \in \mathbb{R}^d \\ \dots \\ \underline{k}_s = \underline{f}(\underline{y}_0 + h\, a_{31}\, \underline{k}_1 + \dots + h\, a_{s\,s-1}\, \underline{k}_{s-1} + h\, a_{ss}\, \underline{k}_s) & \underline{k}_s \in \mathbb{R}^d \end{cases}$$

bekannt aus den vorigen Stufen.

also in diesen Fall (A = untere Dreiecksmatrix):

wir haben s nichtlin.-alg. Probleme in $\mathbb{R}^d$

$\longrightarrow$ leichter als ein nichtlin.-alg. Problem in $\mathbb{R}^{sd}$.

sammelt alles Bekannte ($\underline{k}_1, \dots, \underline{k}_{i-1}$)

$$F(\underline{k}) = \underline{k} - f(\underline{y}_0 + \underline{z} + h\, a_{ii}\, \underline{k}) \qquad \underline{k} \in \mathbb{R}^d$$

Löse $F(\underline{k}) = 0$

$$\underline{z} = h \sum_{j=1}^{i-1} a_{ij}\, \underline{k}_j$$

Newton-Schritt

Row: Wählt spezielle Form von Startwert

in Newton:

$$\underline{k}_i^{(0)} = \sum_{j=1}^{i-1} \frac{d_{ij}}{a_{ij}}\, \underline{k}_j$$

Fragerunde 15.05.2020

$$M_i(t) = M_i(0)\, e^{-\lambda_i t}, \quad t \geq 0 \quad \text{für } i = 1,2,\dots,n$$

$$G(t) = \sum_{i=1}^{n} G_i(t) = \sum_{i=1}^{n} \lambda_i M_i(t)$$

Bekannt: $\lambda_1, \lambda_2, \dots, \lambda_n$

Messungen an $t_1, t_2, \dots, t_j, \dots, t_m$

↳ $G(t_j)$ $j = 1, \dots, m$

t_1: $\lambda_1 M_1(t_1) + \lambda_2 M_2(t_1) + \dots + \lambda_n M_n(t_1) = G(t_1) = G_1$

t_2: $\lambda_1 M_1(t_2) + \lambda_2 M_2(t_2) + \dots + \lambda_n M_n(t_2) = G(t_2) = G_2$

$\vdots$

t_j: $\lambda_1 M_1(t_j) + \lambda_2 M_2(t_j) + \dots + \lambda_n M_n(t_j) = G(t_j) = G_j$

$\vdots$

t_m: $\lambda_1 M_1(t_m) + \lambda_2 M_2(t_m) + \dots + \lambda_n M_n(t_m) = G(t_m) = G_m$

$$\begin{cases} \sum_{i=1}^{n} \lambda_i M_i(t_j) = G(t_j) \\ j = 1,2,\dots,m \end{cases}$$

$$M_i(t_j) = M_i(0)\, e^{-\lambda_i t_j}$$

$$\begin{cases} \sum_{i=1}^{n} \lambda_i e^{-\lambda_i t_j} M_i(0) = G(t_j) \\ j = 1,2,\dots,m \end{cases}$$

$$\underline{\underline{A}} := \left[\lambda_i e^{-\lambda_i t_j} \right]_{i=1,\dots,n}$$

$$\underline{x} = \left[M_i(0) \right]_{i=1,2,\dots,n} \quad j = 1,\dots,m.$$

$$\boxed{\underline{\underline{A}}\,\underline{x} = \underline{G}}$$

$$\underline{G} = \left[G(t_j) \right]_{j=1,\dots,m}$$

$$\min_{x \in \mathbb{R}^n} \| \underline{\underline{A}}\,\underline{x} - \underline{G} \|_2$$

$G[t_j]$ = exakt $\Rightarrow$ $\underline{G}$ — randn() Gauss Verteilung.

np.random.rand()

$\underline{G} = \underline{G} + np.rand(m)$ — gleichmässig verteilt.

↳ Zufallszahlen zwischen 0,1

$$\underline{A} = \begin{bmatrix} \sin(2\pi t_j) & \cos(2\pi t_j) & \sin(2\pi \cdot 2t_j) & \cos(2\pi \cdot 2t_j) \end{bmatrix}$$

$\underline{A} \in \mathbb{R}^{m,4}$ $\quad \underline{A}\underline{x} = \underline{G}$

$\min \| \underline{A}x - G \|_2$

$\underline{x} \in \mathbb{R}^4$

5. *(8 Punkte) Ausgleichsrechnung*

Sei das Naturgesetz

$$f(t) = x_0 \sin(2\pi t) + x_1 \cos(2\pi t) + x_2 \sin(2\pi 2t) + x_3 \cos(2\pi 2t)\,.$$

Um die Konstanten $x_0, \ldots, x_3$ zu bestimmen, werden es 21 äquidistante Messungen zwischen 0 und 1 gemacht, die Ergebnisse G sind im Templatefile.

a) Formulieren Sie ein Ausgleichsproblem zur Bestimmung von $x_0, \ldots x_3$.

b) Lösen Sie dieses Ausgleichsproblem für die Daten im Templatefile mittels Normalengleichung, QR-Zerlegung und mit Standardsoftware aus `numpy`. Geben Sie jeweils das Ergebnis $x_0, \ldots, x_3$ und das Quadrat der Euklidischer Norm des Residuums aus.

b)

$\underline{A}\underline{x} = \underline{b} \Rightarrow \underline{A}^T \underline{A}\underline{x} = \underline{A}^T \underline{b}$

$\underline{A}^T \underline{A}$ symmetrisch

a)

$t_0, t_1, \ldots, t_{M-1}$ mit $M = 21$.

$f(t_j) = G_j \qquad j = 0, 1, \ldots, M-1$

$$\begin{cases} \sin(2\pi t_j) \cdot x_0 + \cos(2\pi t_j) \cdot x_1 + \sin(2\pi 2t_j)\, x_2 + \cos(2\pi 2t_j)\, x_3 = G_j \\ j = 0, 1, \ldots, M-1 \end{cases}$$

$\underline{y} : [0,T] \to \mathbb{R}^3 \qquad \underline{y} = [y_1, y_2, y_3]^T$

$\dot{\underline{y}} = \underline{f}(\underline{y}, y_1^2, y_2^2, y_3^2, \cos(y_1), \cos(y_2), \cos(y_3))$

$\| \quad \| \quad \| \qquad \| \quad \| \quad \|$

$z_1 \quad z_2 \quad z_3 \qquad u_1 \quad u_2 \quad u_3$

$\underline{w} = [y_1, y_2, y_3, z_1, z_2, z_3, u_1, u_2, u_3]$

$$\begin{cases} \dot{\underline{w}} = \underline{f}(\underline{w}) \to \text{ODE} \\ z_1 = y_1^2 \\ z_2 = y_2^2 \\ z_3 = y_3^2 \\ u_1 = \cos(y_1) \\ u_2 = \cos(y_2) \\ u_3 = \cos(y_3) \end{cases} \quad \text{Algebraische Gleichungen} \quad \Big\} \text{DAE}$$

Differential Algebraical Equations

Differential-Algebraische-Gleichungen.

↳ Spezielle Techniken.

Fragerunde 22.05.2020

$\underline{A} \in \mathbb{R}^{n \times n}$ invertierbar / voller Rang.

$$\text{cond}(\underline{A}) = \|\underline{A}^{-1}\| \, \|\underline{A}\|$$

$$\text{cond}(\underline{A}) = \frac{\max\limits_{\|x\|=1} \|\underline{A}x\|}{\min\limits_{\|x\|=1} \|\underline{A}x\|}$$

$$\Rightarrow \max_{\|x\|=1} \|\underline{A}x\| = \sigma_1 \qquad \min_{\|x\|=1} \|\underline{A}x\| = \sigma_n$$

$$\|A\| = \max_{\|x\|=1} \|\underline{A}\underline{x}\| = \sigma_1$$

$$\underline{A}^{-1} = \underline{V} \, \underline{\Sigma}^{-1} \, \underline{U}^T \Rightarrow \|A^{-1}\| = \max_{j=1,2,\dots,n} \frac{1}{\sigma_j} = \frac{1}{\min\limits_{j=1,\dots,n} \sigma_j} = \frac{1}{\sigma_n}$$

Somit sind die 2 Definitionen äquivalent.

$\underline{A} = \underline{U} \, \underline{\Sigma} \, \underline{V}^T \qquad \underline{U}, \underline{V}^T \in \mathbb{R}^{n \times n}$ orthogonal

$$\|\underline{A}\underline{x}\| = \|\underline{U} \, \underline{\Sigma} \, \underbrace{\underline{V}^T \underline{x}}_{\underline{z}}\| = \|\underline{U} \, \underline{\Sigma} \, \underline{y}\| = \|\underline{\Sigma} \, \underline{z}\|$$

$$\|\underline{x}\| = 1 \iff \|\underline{y}\| = \|\underline{V}^T \underline{x}\| = 1$$

$$\|\underline{A}\underline{x}\| = \sum_{j=1}^{n} \sigma_j \, y_j^2 \quad \text{mit} \quad \|\underline{y}\| = 1$$

Annahme $\underline{\underline{A}} \in \mathbb{R}^{n\times n}$ diagonalisierbar

$\underline{\underline{A}} = \underline{\underline{S}}^{-1} \underline{\underline{\Lambda}}\, \underline{\underline{S}}$

$|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$

$$\frac{\underline{\underline{A}}^k x}{\|\underline{\underline{A}}^k x\|_2} \longrightarrow \pm \underline{s}_1$$

$$\rho_{\underline{\underline{A}}}(x_k) = \lambda_1 + O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right)$$

$\rho_A(x_k) \to \lambda_1$ linear mit Rate $\left|\frac{\lambda_2}{\lambda_1}\right|$

Falls $\underline{\underline{A}}$ normale ($A^H A = A A^H$)

$\Rightarrow$ orthogonale EV

$\Rightarrow$ Fehler $O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right)$

$$\rho_A(x) - \rho_{\underline{\underline{A}}}(s_1) = O\left(\|x - \underline{s}_1\|_2^2\right)$$

$\hookrightarrow$ Falls $\underline{\underline{A}}$ symmetrisch

$$|\rho_A(x_k) - \lambda_1| \leq c \cdot \left|\frac{\lambda_2}{\lambda_1}\right|^k = c \cdot r^k$$

$$r = \left|\frac{\lambda_2}{\lambda_1}\right|$$

$$|\rho_A(x_{k+1}) - \lambda_1| \leq r\, |\rho_A(x) - \lambda_1|$$

$$\rho_{\underline{\underline{A}}}(x_k) = \lambda_1 + c \cdot \left|\frac{\lambda_2}{\lambda_1}\right|^k$$

$$|\rho_A(x_k) - \lambda_1| \leq c|r|^k$$

$$|\rho_A(x_{k+1}) - \lambda_1| \leq c|r|^{k+1} = r \cdot |\rho_A(x_k) - \lambda_1|$$

inverse Potenzmethode.

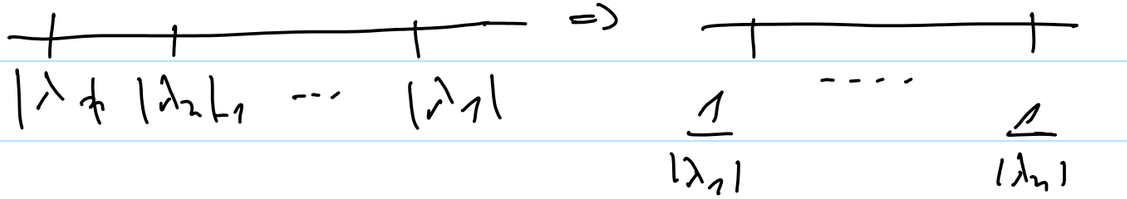
<u>Annahme</u> A invertierbar.

$$Ax = \lambda x \Rightarrow x = \lambda A^{-1}x \Rightarrow$$

$A^{-1}|$

$$\frac{1}{\lambda}x = A^{-1}x$$

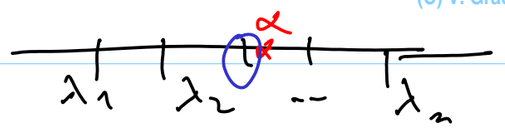
λ EW von A $\Leftrightarrow$ $\frac{1}{\lambda}$ EW von A^{-1}



Potenzmethode für A^{-1} $\Rightarrow$ $\frac{1}{\lambda_n}$ $\Rightarrow$

λ_n = kleinste EW von A.

Suchen: EW nach α

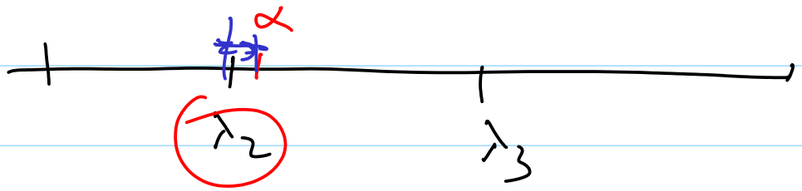


$|\alpha - \lambda| = \min\{|\alpha - \mu|;\ \mu \text{ EW von } A\}$.

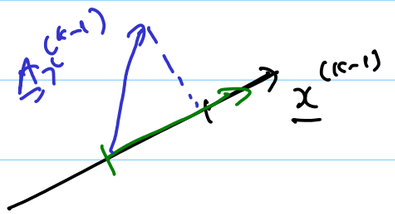
$$Ax = \lambda x \quad \Big| - \alpha I x \quad \Rightarrow (A - \alpha I)x = (\lambda - \alpha)x$$

$$\frac{1}{\lambda - \alpha}x = (A - \alpha I)^{-1}x$$

Potenzmethode für $(A - \alpha I)^{-1}$ $\Rightarrow$ $\frac{1}{\lambda - \alpha}$ $\Rightarrow$ λ



↳ Idee: wähle α adaptiv. $\alpha = \rho_A(x^{(k-1)})$



$\rho_A(x^{(k-1)})$ $\downarrow$ λ