PVK Numerische Methoden
Tag 2

Lucas Böttcher

ETH Zürich
Institut für Baustoffe
Wolfgang-Pauli-Str. 27
HIT G 23.8
8093 Zürich

lucasb@ethz.ch

June 20, 2017

Übersicht

Kurstag	**Inhalt**
Tag 1	Differenzialgleichungen
Tag 2	Quadratur, Nichtlineare algebraische Gleichungen
Tag 3	Ausgleichsrechnung, Eigenwerte, Interpolation
Tag 4	Lineare ODEs, Exponentielle Integratoren, Splitting-Verfahren
Tag 5	Erweiterte Übungen

Kapitel II
Numerische Quadratur (pp. 158–207 im Skript)

Numerische Quadratur

Integrale sind oft analytisch nicht oder schwer zugänglich. Man kann diese jedoch auch oft numerisch approximieren.

Lernziele: Kennen und Anwenden geeigneter Quadraturverfahren zum numerischen Lösen eines gegebenen Integrals.

Numerische Quadratur

Lösungsverfahren:

1. Mittelpunkts-, Simpson- und Trapezregel
2. Nicht äquidistante Stützstellen
3. Adaptive Quadratur
4. Quadratur in $\mathbb{R}^d$
5. Monte-Carlo-Quadratur

Numerische Quadratur mit äquidistanten Stützstellen

Grundidee: Approximation des Integrals
$\int_a^b f(x)dx \approx Q_n(f; a, b) := \sum_{i=1}^n \omega_i^n f(c_i^n)$.

Numerische Quadratur mit äquidistanten Stützstellen

Numerische Quadratur mit äquidistanten Stützstellen

Grundidee: Approximation des Integrals
$\int_a^b f(x)dx \approx Q_n(f; a, b) := \sum_{i=1}^n \omega_i^n f(c_i^n)$.

Eine Quadraturformel besitzt Ordnung $n+1$, wenn sie ein Polynom vom Grad n exakt integriert.

Numerische Quadratur mit äquidistanten Stützstellen

Numerische Quadratur mit äquidistanten Stützstellen

Grundidee: Approximation des Integrals
$\int_a^b f(x)dx \approx Q_n(f; a, b) := \sum_{i=1}^n \omega_i^n f(c_i^n)$.

Eine Quadraturformel besitzt Ordnung $n+1$, wenn sie ein Polynom vom Grad n exakt integriert.

Numerische Quadratur mit äquidistanten Stützstellen

Numerische Quadratur mit äquidistanten Stützstellen

Grundidee: Approximation des Integrals
$\int_a^b f(x)dx \approx Q_n(f; a, b) := \sum_{i=1}^n \omega_i^n f(c_i^n)$.

Eine Quadraturformel besitzt Ordnung $n+1$, wenn sie ein Polynom vom Grad n exakt integriert.

Mittelpunktsregel: $Q^M(f; a, b) = (b-a) f\left(\frac{a+b}{2}\right)$ (Ordnung 2).

Numerische Quadratur mit äquidistanten Stützstellen

Numerische Quadratur mit äquidistanten Stützstellen

Grundidee: Approximation des Integrals
$\int_a^b f(x)dx \approx Q_n(f; a, b) := \sum_{i=1}^n \omega_i^n f(c_i^n)$.

Eine Quadraturformel besitzt Ordnung $n+1$, wenn sie ein Polynom vom Grad n exakt integriert.

Mittelpunktsregel: $Q^M(f; a, b) = (b-a) f\left(\frac{a+b}{2}\right)$ (Ordnung 2).

Trapezregel: $Q^T(f; a, b) = \frac{b-a}{2}(f(a) + f(b))$ (Ordnung 2).

Numerische Quadratur mit äquidistanten Stützstellen

Numerische Quadratur mit äquidistanten Stützstellen

Grundidee: Approximation des Integrals
$\int_a^b f(x)dx \approx Q_n(f;a,b) := \sum_{i=1}^n \omega_i^n f\left(c_i^n\right)$.

Eine Quadraturformel besitzt Ordnung $n+1$, wenn sie ein Polynom vom Grad n exakt integriert.

Mittelpunktsregel: $Q^M\left(f;a,b\right) = (b-a)\, f\left(\frac{a+b}{2}\right)$ (Ordnung 2).

Trapezregel: $Q^T\left(f;a,b\right) = \frac{b-a}{2}\left(f(a)+f(b)\right)$ (Ordnung 2).

Simpsonregel: $Q^S\left(f;a,b\right) = \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$ (Ordnung 4).

Numerische Quadratur mit äquidistanten Stützstellen

Numerische Quadratur mit äquidistanten Stützstellen

Grundidee: Approximation des Integrals
$\int_a^b f(x)dx \approx Q_n(f;a,b) := \sum_{i=1}^n \omega_i^n f\left(c_i^n\right)$.

Eine Quadraturformel besitzt Ordnung $n+1$, wenn sie ein Polynom vom Grad n exakt integriert.

Mittelpunktsregel: $Q^M\left(f;a,b\right) = (b-a)\, f\left(\frac{a+b}{2}\right)$ (Ordnung 2).

Trapezregel: $Q^T\left(f;a,b\right) = \frac{b-a}{2}\left(f(a)+f(b)\right)$ (Ordnung 2).

Simpsonregel: $Q^S\left(f;a,b\right) = \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$ (Ordnung 4).

Zusammengesetzte Regeln:
$\int_a^b f(x)dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx = Q_n(f;x_i,x_{i+1})$ mit $h = \frac{b-a}{N}$, $x_0 = a$, $x_i = x_0 + ih$, $x_N = b$.

Numerische Quadratur mit äquidistanten Stützstellen

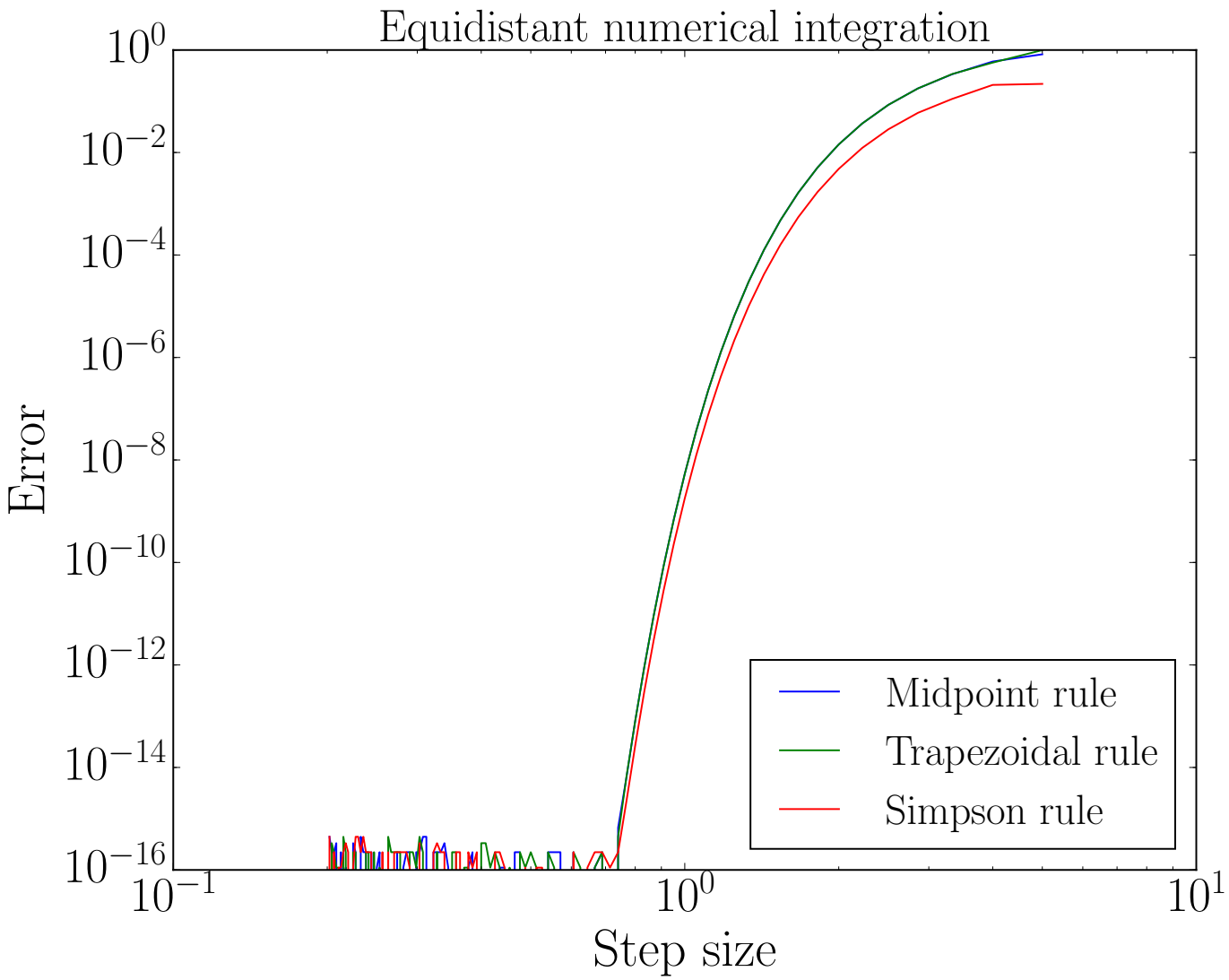
Beispiel Mittelpunkts-, Trapez- und Simpsonregel

Integriere die Funktion $f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-x^2/2\right)$ im Interval $[-10, 10]$ unter der Verwendung von Mittelpunkts-, Trapez- und Simpsonregel. Untersuche die Konvergenz mit der Annahme, dass das Integral einem Wert von 1 entspricht.

Numerische Quadratur mit äquidistanten Stützstellen

```
def mittelpunkt(f, a, b, N, h):
    res = 0
    for i in range(N):
        xi = a + i*h
        xip1 = a + (i+1)*h
        res += (xip1-xi)*f(0.5*(xi+xip1))
    return res

def trapez(f, a, b, N, h):
    res = 0
    for i in range(N):
        xi = a + i*h
        xip1 = a + (i+1)*h
        res += 0.5*(xip1-xi)*(f(xi)+f(xip1))
    return res
```

Numerische Quadratur mit äquidistanten Stützstellen



Numerische Quadratur

Lösungsverfahren:

1. Mittelpunkts-, Simpson- und Trapezregel
2. Nicht äquidistante Stützstellen
3. Adaptive Quadratur
4. Quadratur in $\mathbb{R}^d$
5. Monte-Carlo-Quadratur

Numerische Quadratur mit nicht äquidistanten Stützstellen

Grundidee: Approximation des Integrals $\int_a^b f(x)dx = \int_a^b \Phi(x)\omega(x)dx = \sum_{i=1}^n \Phi(x_i)\alpha_i$ bei der die Funktion $f(x)$ durch ein Polynom $\Phi(x)$ und eine Gewichtsfunktion $\omega(x)$ beschrieben wird. Man muss dann die Knoten x_i und Gewichte α_i bestimmen.

Weiter Informationen:
`https://pomax.github.io/bezierinfo/legendre-gauss.html`

Beispiel Gauss-Legendre-Quadratur

Beispiel Gauss-Legendre-Quadratur

Integriere die Funktion $f(x) = 1/(1 + 25x^2)$ im Interval $[0, 1]$ mit der Gauss-Legendre Quadratur unter Verwendung des Golub-Welsch Algorithmus.

Beispiel Gauss-Legendre-Quadratur

```
# Zusammengesetzte Gauss–Legendre Quadratur
def composite_legendre(f, a, b, N, n):

    I = 0.0

    dx = (b - a)/N
    for i in range(N):
        I += gauss_legendre(f, a + i*dx, a + (i+1)*dx, n)

    return I
```

Beispiel Gauss-Legendre-Quadratur

```
# Nichtzusammengesetzte Gauss–Legendre Quadratur
def gauss_legendre(f, a, b, n):

    x_ref, w_ref = golub_welsch(n)

    x = a + (x_ref + 1.0)*(b - a)*0.5
    w = 0.5*(b - a)*w_ref

    return np.sum(w*f(x))
```

Beispiel Gauss-Legendre-Quadratur

```
# Knoten und Gewichte
def golub_welsch(n):

    i = np.arange(n-1)
    b = (i+1.0) / np.sqrt(4.0*(i+1)**2 - 1.0)
    J = np.diag(b, -1) + np.diag(b, 1)
    x, ev = np.linalg.eigh(J)
    w = 2 * ev[0,:]**2

    return x, w
```

Numerische Quadratur

Lösungsverfahren:

1. Mittelpunkts-, Simpson- und Trapezregel
2. Nicht äquidistante Stützstellen
3. Adaptive Quadratur
4. Quadratur in $\mathbb{R}^d$
5. Monte-Carlo-Quadratur

Adaptive Quadratur

Adaptive Quadratur

Grundidee: Unterteilung des Integrationsgebietes in Subintervalle, welche gleich zum Gesamtfehler beitragen.

Adaptive Quadratur

Adaptive Quadratur

Grundidee: Unterteilung des Integrationsgebietes in Subintervalle, welche gleich zum Gesamtfehler beitragen.
Man schätzt dazu den lokalen Fehler, um die Schrittweite entsprechend anzupassen.

Adaptive Quadratur

Adaptive Quadratur

Grundidee: Unterteilung des Integrationsgebietes in Subintervalle, welche gleich zum Gesamtfehler beitragen.
Man schätzt dazu den lokalen Fehler, um die Schrittweite entsprechend anzupassen.
Dazu vergleicht man zum Beispiel die lokalen Werte einer Trapez- und Simpsonregelintegration und unterteilt das Interval, wenn die Abweichung zu gross ist.

Adaptive Quadratur

```
def adaptive_quadrature(f,M,rtol,abstol):
    h = diff(M)
    mp = 0.5*( M[:-1]+M[1:] )
    fx = f(M); fm = f(mp)
    trp_loc = h*( fx[:-1]+fx[1:] )/2
    simp_loc= h*( fx[:-1]+4*fm+fx[1:] )/4
    I = sum(simp_loc)
    est_loc = abs(simp_loc - trp_loc)
    err_tot = sum(est_loc)
    if err_tot > rtol*abs(I) and err_tot > abstol:
        refcells = nonzero( est_loc > 0.9*sum(est_loc)
        /size(est_loc) )[0]
        I = adaptive_quadrature(f,
        sort(append(M,mp[refcells])),rtol,abstol)
    return I
```

Numerische Quadratur

Lösungsverfahren:

1. Mittelpunkts-, Simpson- und Trapezregel
2. Nicht äquidistante Stützstellen
3. Adaptive Quadratur
4. Quadratur in $\mathbb{R}^d$
5. Monte-Carlo-Quadratur

Numerische Quadratur in $\mathbb{R}^d$

Numerische Quadratur in $\mathbb{R}^d$

Grundidee: Das Integral $I = \int_a^b \int_c^d f(x,y)dxdy = \int_a^b F(y)dy$ wird berechnet mit $F(y) = \int_a^b f(x,y)dx$ und einem geeigneten Quadraturverfahren für beide Integrale.

Numerische Quadratur in $\mathbb{R}^d$

Numerische Quadratur in $\mathbb{R}^d$

Grundidee: Das Integral $I = \int_a^b \int_c^d f(x,y)dxdy = \int_a^b F(y)dy$ wird berechnet mit $F(y) = \int_a^b f(x,y)dx$ und einem geeigneten Quadraturverfahren für beide Integrale.

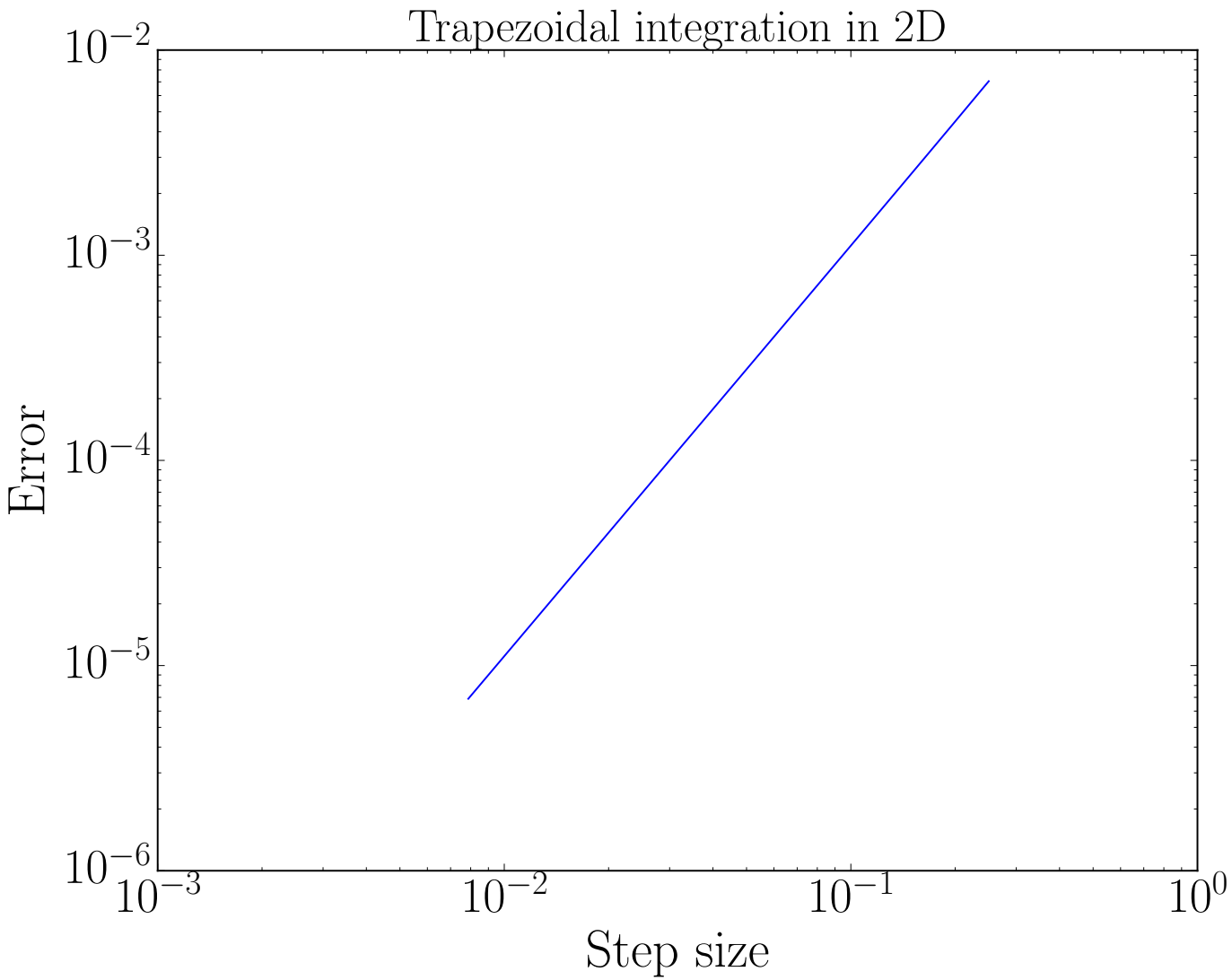
Beispiel numerische Integration in $\mathbb{R}^2$

Integriere die Funktion $\int_0^1 \int_0^1 x^2 y^2 dxdy$ numerisch mit der Trapezregel und studiere die Konvergenzeigenschaften für verschiedene Gittergrössen in dem die Abweichung von der analytischen Lösung analysiert wird.

Numerische Quadratur in $\mathbb{R}^d$

```
f = lambda x, y: x**2*y**2

def trapez(f, a, b, N):
    x, h = linspace(a, b, N+1, retstep=True)

    I = h*(0.5*f(x[0]) + sum(f(xx) for xx in x[1:-1])
        + 0.5*f(x[-1]))

    return I

def trapez2d(f, a, b, Nx, c, d, Ny):
    F = lambda y: trapez(lambda x: f(x, y), a, b, Nx)

    return trapez(F, c, d, Ny)
```

Numerische Quadratur in $\mathbb{R}^d$



Numerische Quadratur

Lösungsverfahren:

1. Mittelpunkts-, Simpson- und Trapezregel
2. Nicht äquidistante Stützstellen
3. Quadratur in $\mathbb{R}^d$
4. Adaptive Quadratur
5. Monte-Carlo-Quadratur

Monte-Carlo-Quadratur

Monte-Carlo-Quadratur

Grundidee: Approximieren des Integrals
$I = \int_a^b z(s)ds = |b-a| \int_0^1 z(a+(b-a)t)dt \approx |b-a| \frac{1}{N} \sum_{i=1}^N z(s_i)$ mit $s_i = a + (b-a)t_i$ und $t_i \in [0,1]$.

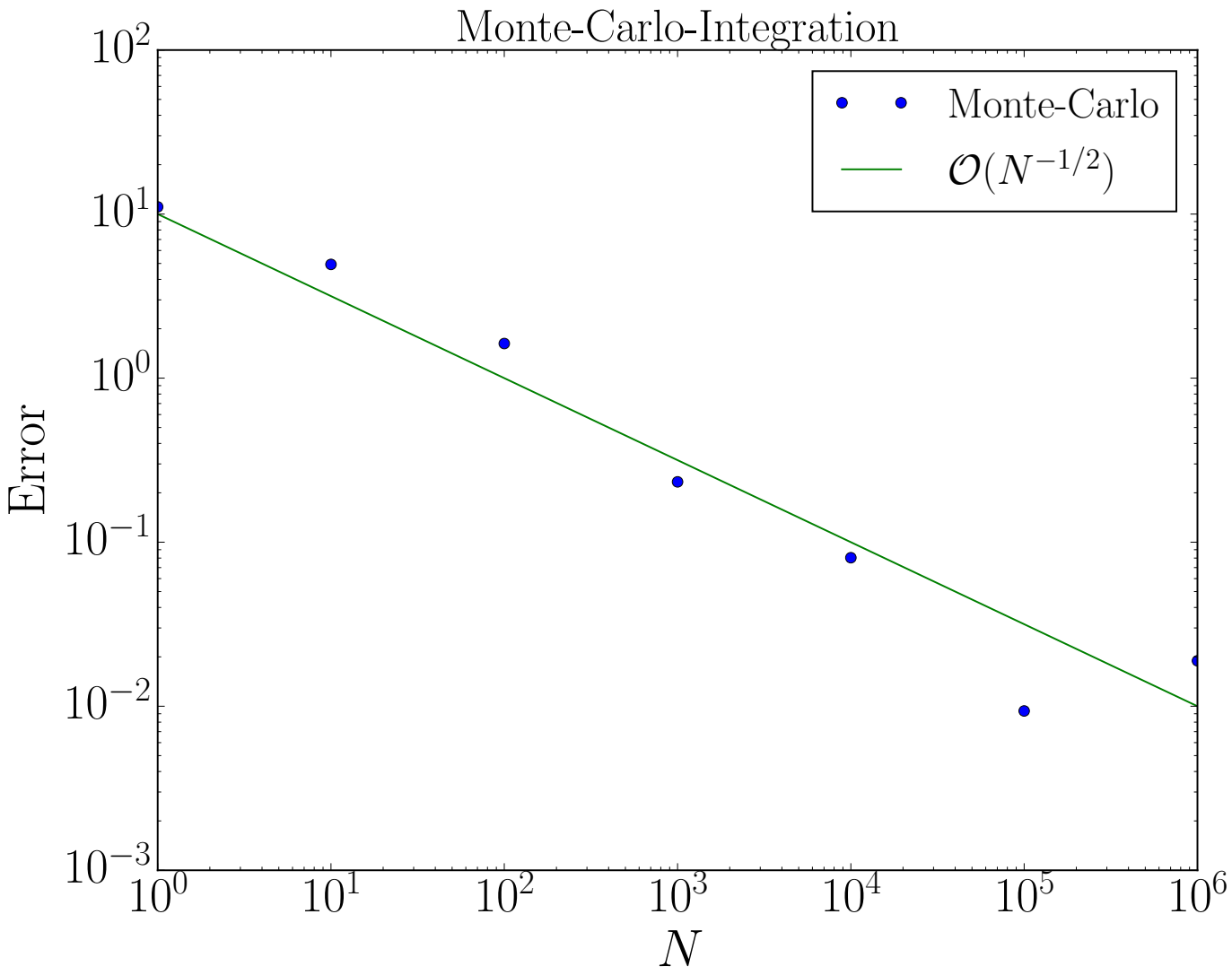
Die Konvergenz hängt nicht von der Glattheit der Funktion oder der Dimension ab. Jedoch ist die Konvergenz langsam mit $\mathcal{O}(N^{-1/2})$.

Beispiel Monte-Carlo-Integration

Beispiel Monte-Carlo-Integration

Berechne das Volumen der n-dimensionalen Einheitskugel mit der Monte-Carlo-Integration von $f(\mathbf{x}) = x_1^2 + x_2^2 + \cdots + x_n^2$ und bestimme die Konvergenzeigenschaft mithilfe der analytischen Lösung $V_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$.

Beispiel Monte-Carlo-Integration



Kapitel III
Nullstellensuche (pp. 7–36 im Skript)

Nullstellensuche

Für die Funktion $F\colon D \subset \mathbb{R}^n \to \mathbb{R}^n$, $n \in \mathbb{N}$ suchen wir ein $x^* \subset D$, so dass $F(x^*) = 0$.

Lernziele: Kennen und Anwenden geeigneter Lösungverfahren zum numerischen Lösen eines gegebenen Nullstellenproblems $F(x^*) = 0$.

Nullstellensuche

Für die Funktion $F\colon D \subset \mathbb{R}^n \to \mathbb{R}^n$, $n \in \mathbb{N}$ suchen wir ein $x^* \subset D$, so dass $F(x^*) = 0$.

Lernziele: Kennen und Anwenden geeigneter Lösungverfahren zum numerischen Lösen eines gegebenen Nullstellenproblems $F(x^*) = 0$.

Motivation:

1. Gleichungen mit analytischer Lösung
   z.B. $ax^2 + bx + c = 0$ hat die Lösung $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.
2. Gleichungen ohne analytische Lösung
   z.B. $xe^x - 1 = 0$.

Im zweiten Beispiel können wir numerische Löser verwenden.

Nullstellensuche

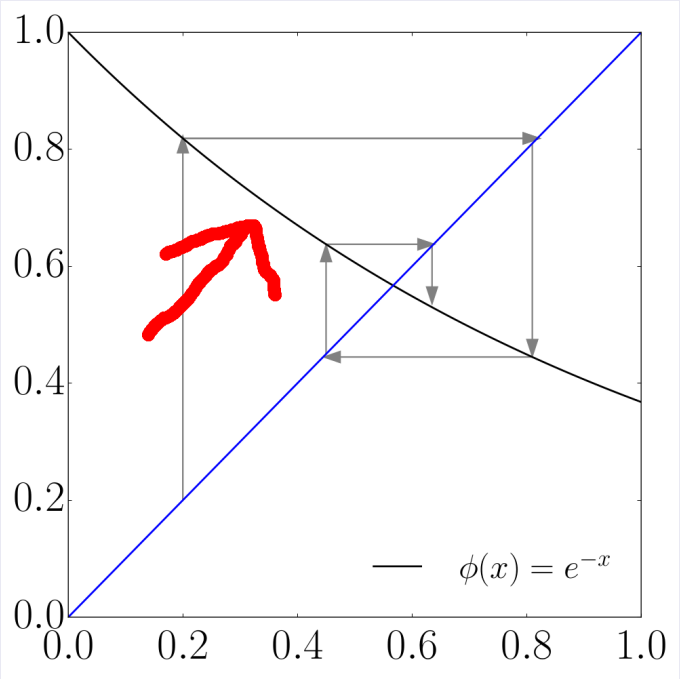
Lösungsverfahren:

1. Fixpunktiteration
2. Intervallhalbierungsverfahren
3. Newtonverfahren
4. Sekantenverfahren
5. Quasi-Newton-Verfahren

Nullstellensuche mit Fixpunktiteration

Beispiel 1: Numerisches Lösen von $xe^x - 1 = 0$

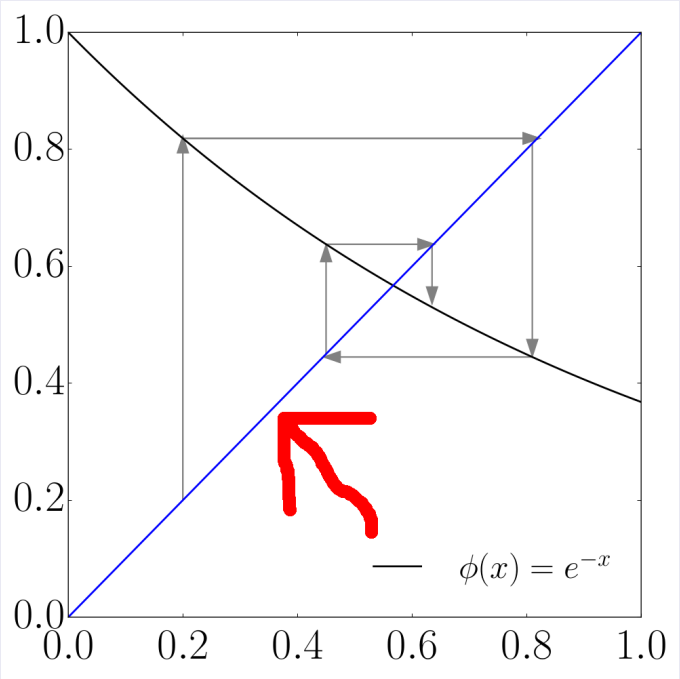
Fixpunktiteration mit $x^{(k+1)} = \phi\left(x^{(k)}\right)$, wobei $\phi(x) = e^{-x}$.



Nullstellensuche mit Fixpunktiteration

Beispiel 1: Numerisches Lösen von $xe^x - 1 = 0$

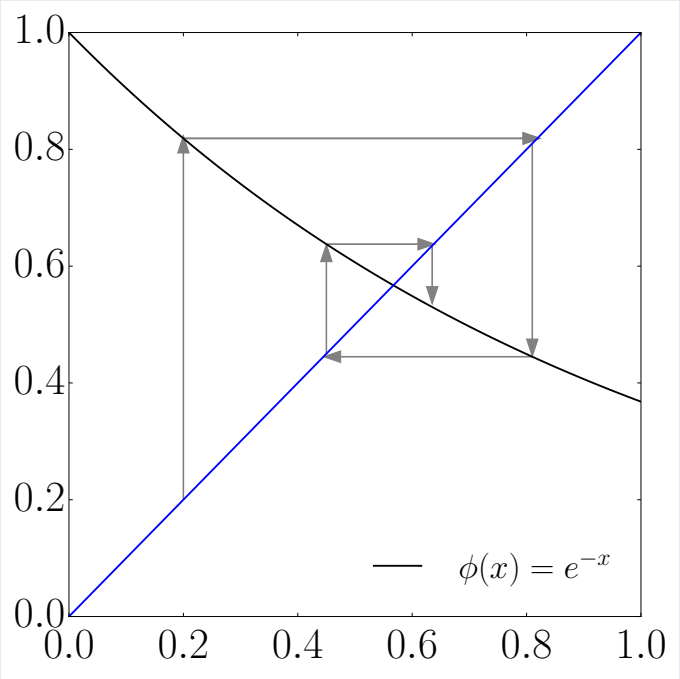
Fixpunktiteration mit $x^{(k+1)} = \phi\left(x^{(k)}\right)$, wobei $\phi(x) = e^{-x}$.



Nullstellensuche mit Fixpunktiteration

Beispiel 1: Numerisches Lösen von $xe^x - 1 = 0$

Fixpunktiteration mit $x^{(k+1)} = \phi\left(x^{(k)}\right)$, wobei $\phi(x) = e^{-x}$.



Iteration	Value
0	0.20
1	0.82
2	0.44
3	0.64
4	0.53
5	0.59
6	0.55

Quiz

1. Überlegt zuerst allein und dann diskutiert gemeinsam für 2 min.
2. Teilt die Lösung mit dem Kurs.

Quiz

Quiz 1: Was sind die zwei Fehler im folgenden Code zur Fixpunktiteration für $xe^x - 1 = 0$?

```
import numpy as np

def phi(x):
    return x*np.exp(-x)

x0 = 0.2
x = []
x.append(x0)

for i in range(10):
    x.append(phi(x[i-1]))

print x
```

Classroom activity

Lösung 1: Was sind die zwei Fehler im folgenden Code zur Fixpunktiteration für $xe^x - 1 = 0$?

```
import numpy as np

def phi(x):
    return —> np.exp(-x) <—

x0 = 0.2
x = []
x.append(x0)

for i in range(10):
    —> x.append(phi(x[i])) <—

print x
```

Nullstellensuche mit Fixpunktiteration

Wichtige Eigenschaften:

Der Banach'sche Fixpunktsatz (*Satz 1.3.7.*) garantiert die Existenz eines eindeutigen Fixpunktes für eine Kontraktion, d.h. eine Funktion ϕ mit $L < 1$, sodass $\|\phi(x) - \phi(y)\| \leq L\|x - y\|$ für alle x, y.

Nullstellensuche mit Fixpunktiteration

Wichtige Eigenschaften:

Der Banach'sche Fixpunktsatz (*Satz 1.3.7.*) garantiert die Existenz eines eindeutigen Fixpunktes für eine Kontraktion, d.h. eine Funktion ϕ mit $L < 1$, sodass $\|\phi(x) - \phi(y)\| \leq L\|x - y\|$ für alle x, y.

Erfüllt die Funktion ϕ die Eigenschaft $\phi^{(l)}(x^*) = 0$, $l = 1, 2, \ldots, m + 1$ ($m \geq 1$), dann ist die lokale Konvergenzordnung $p \geq m + 1$ (*Hilfssatz 1.3.14.*).

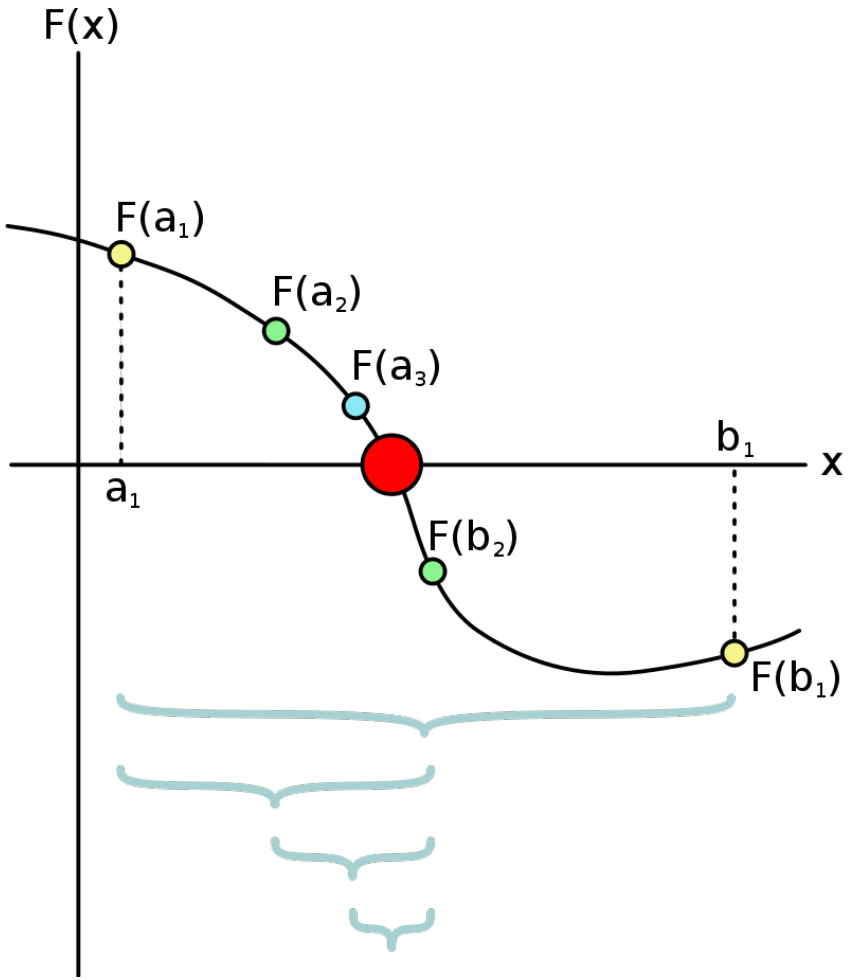
Weitere Eigenschaften im Skript (pp. 14–19).

Nullstellensuche

Lösungsverfahren:

1. Fixpunktiteration
2. Intervallhalbierungsverfahren
3. Newtonverfahren
4. Sekantenverfahren
5. Quasi-Newton-Verfahren

Nullstellensuche mit Intervallhalbierungsverfahren



Der Zwischenwertsatz garantiert die Existenz einer Nullstelle im Intervall $[a, b]$ für eine stetige Funktion mit $f(a) > 0$ und $f(b) < 0$.

Nullstellensuche mit Intervallhalbierungsverfahren

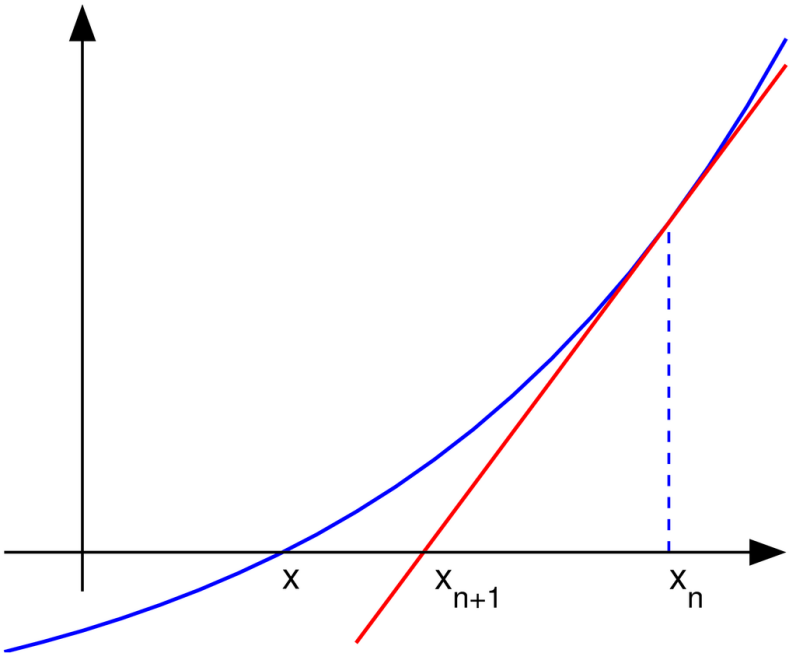
```
def f(x):
    return x**5 + x -1

def bisection(a,b,tol):
    c = (a+b)/2.0
    while (b-a)/2.0 > tol:
        if f(c) == 0:
            return c
        elif f(a)*f(c) < 0:
            b = c
        else:
                a = c
        c = (a+b)/2.0
    return c

print bisection(0,5,10**-4)
```

Nullstellensuche

Lösungsverfahren:

1. Fixpunktiteration
2. Intervalhalbierungsverfahren
3. Newtonverfahren
4. Sekantenverfahren
5. Quasi-Newton-Verfahren

Nullstellensuche mit Newtonverfahren



By Olegalexandrov at English Wikipedia - Transferred from en.wikipedia to Commons., Public Domain, https://commons.wikimedia.org/w/index.php?curid=44101

Unter Zuhilfenahme der ersten Ableitung erhält man ein Iterationsverfahren mit quadratischer Konvergenz. Die Newton-Methode ist gegeben durch folgende Iterationsvorschrift: $x^{(k+1)} = x^{(k)} - F\left(x^k\right)/F'\left(x^k\right)$, mit $F'\left(x^k\right) \neq 0$.

Nullstellensuche mit Newtonverfahren (1D)

```
f = lambda x: x**5 + x - 1
fp = lambda x: 5*x**4 + 1

def newton_method(x0, tol):
    delta = 1.0*f(x0)/fp(x0)
    while delta > tol:
        x0 -= delta
        delta = f(x0)/fp(x0)
    return x0

print newton_method(1, 10**-4)
```

Nullstellensuche mit Newtonverfahren (mit *scipy.optimize*)

```
from scipy.optimize import newton

f = lambda x: x**5 + x - 1

x0 = 1
print newton(f, x0)
```

Nullstellensuche mit Newtonverfahren

Verallgemeinerungen und Varianten:

In mehreren Dimensionen entspricht die Ableitung der Funktion $f\colon U \subset \mathbb{R}^n \to \mathbb{R}^m$ an der Stelle $a \subset U$ der Jacobimatrix:

$$J_F(a) := \left(\frac{\partial F_i}{\partial x_j}(a)\right)_{i=1,\dots,m;\ j=1,\dots,n} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(a) & \frac{\partial F_1}{\partial x_2}(a) & \dots & \frac{\partial F_1}{\partial x_n}(a) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1}(a) & \frac{\partial F_m}{\partial x_2}(a) & \dots & \frac{\partial F_m}{\partial x_n}(a) \end{pmatrix}.$$

Nullstellensuche mit Newtonverfahren

Verallgemeinerungen und Varianten:

In mehreren Dimensionen entspricht die Ableitung der Funktion $f\colon U \subset \mathbb{R}^n \to \mathbb{R}^m$ an der Stelle $a \subset U$ der Jacobimatrix:

$$J_F(a) := \left(\frac{\partial F_i}{\partial x_j}(a)\right)_{i=1,\dots,m;\ j=1,\dots,n} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(a) & \frac{\partial F_1}{\partial x_2}(a) & \dots & \frac{\partial F_1}{\partial x_n}(a) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1}(a) & \frac{\partial F_m}{\partial x_2}(a) & \dots & \frac{\partial F_m}{\partial x_n}(a) \end{pmatrix}.$$

Manchmal ist es auch nötig die Newtonkorrektur mit einem Vorfaktor zu dämpfen. Ansonsten würde man keine Konvergenz finden.

Nullstellensuche mit Newtonverfahren

Verallgemeinerungen und Varianten:

In mehreren Dimensionen entspricht die Ableitung der Funktion $f\colon U \subset \mathbb{R}^n \to \mathbb{R}^m$ an der Stelle $a \subset U$ der Jacobimatrix:

$$J_F(a) := \left(\frac{\partial F_i}{\partial x_j}(a)\right)_{i=1,\dots,m;\ j=1,\dots,n} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(a) & \frac{\partial F_1}{\partial x_2}(a) & \dots & \frac{\partial F_1}{\partial x_n}(a) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1}(a) & \frac{\partial F_m}{\partial x_2}(a) & \dots & \frac{\partial F_m}{\partial x_n}(a) \end{pmatrix}.$$

Manchmal ist es auch nötig die Newtonkorrektur mit einem Vorfaktor zu dämpfen. Ansonsten würde man keine Konvergenz finden.

Im vereinfachten Newtonverfahren nimmt man den gleichen Korrekturfaktor für mehrere Schritte.

Nullstellensuche mit Newtonverfahren (2D)

```
from numpy import array
from scipy.linalg import norm, solve

F = lambda x: array([x[0]**2-x[1]**4, x[0]-x[1]**3])
DF = lambda x: array([[2*x[0],-4*x[1]**3],[1,-3*x[1]**2]])
x = array([0.7, 0.7])
tol = 10**(-10)
s = solve(DF(x),F(x))
x -= s

while norm(s) > tol*norm(x):
    s = solve(DF(x),F(x))
    x -= s

print x
```

Nullstellensuche

Lösungsverfahren:

1. Fixpunktiteration
2. Intervalhalbierungsverfahren
3. Newtonverfahren
4. Sekantenverfahren
5. Quasi-Newton-Verfahren

Nullstellensuche mit Sekantenverfahren

Wir erinnern uns an das Newton-Verfahren mit
$x^{(k+1)} = x^{(k)} - F\left(x^k\right)/F'\left(x^k\right)$, wobei $F'\left(x^k\right) \neq 0$.

Was ist jedoch, wenn man die Auswertung der Ableitung umgehen möchte?

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

Nullstellensuche

Lösungsverfahren:

1. Fixpunktiteration
2. Intervalhalbierungsverfahren
3. Newtonverfahren
4. Sekantenverfahren
5. Quasi-Newton-Verfahren

Nullstellensuche mit Quasi-Newton-Verfahren

Ähnlich zum Sekantenverfahren kann man auch in n Dimensionen die Berechnung der Jakobimatrix erleichtern. Man nutzt dazu das Broyden-Quasi-Newton-Verfahren, cf. pp. 32-34 im Skript.

$$\Delta x^{(k)} := -J_k^{-1} F\left(x^{(k)}\right)$$
$$x^{(k+1)} := x^{(k)} + \Delta x^{(k)}$$
$$J_{k+1} := J_k + \frac{F\left(x^{(k+1)}\right)\left(\Delta x^{(k)}\right)^T}{\|\Delta x^{(k)}\|_2^2}$$