Funktion mit mehreren Variablen

Auch **multivariate** Funktionen genannt mit $n > 1$ für n Paramter.
(Der Begriff macht keine Aussage zu vektorwertig od. skalarwertige.)

Skalarwertige Funktion

$n > 1$ und $m = 1$

$$f : \mathbb{R}^n \to \mathbb{R},\ (x_1, x_2, \dots, x_n) \mapsto y = f(x_1, x_2, \dots, x_n)$$

Vektorwertige Funktion

$n \geq 1$ und $m > 1$

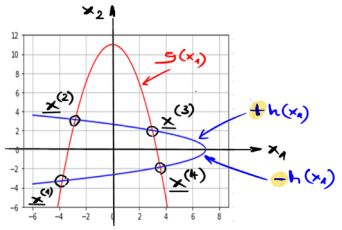
$$f : \mathbb{R}^n \to \mathbb{R}^m,\ x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \mapsto y = f(x) = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_m(x_1, x_2, \dots, x_n) \end{bmatrix}$$

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}^m \text{ mit } y = f(x) = \begin{pmatrix} y_1 = f_1(x) \\ y_2 = f_2(x) \\ \vdots \\ y_m = f_m(x) \end{pmatrix} \text{ und } x = (x_1, x_2, \dots, x_n)^T$$

Falls **n = 2** und **m = 1**:
Skalarwertig bzw. vekorwertig mit nur einer Komponente.

Die Nullstellen der beiden Funktion sind deren Schnittpunkte.

$x_1^2 + x_2 = 11 \longrightarrow x_2 = 11 - x_1^2$ $\quad g(x_1)$
$x_1 + x_2^2 = 7 \longrightarrow x_2 = \pm\sqrt{7 - x_1}$ $\quad h(x_1)$



Darstellungsformen

Für skalarwertige Funktionen mit zwei Variablen.

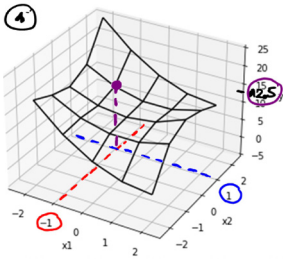
Wertetabelle

$$z = f(x, y)$$

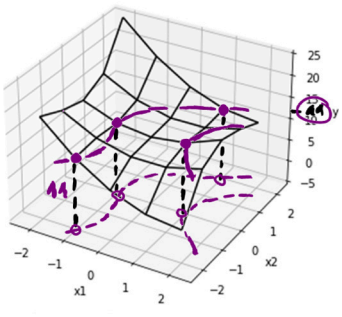
2. unabhängige Variable y

1. unabhängige Variable x \ y	y_1	y_2	…	y_k	…	y_n
x_1	z_{11}	z_{12}	…	z_{1k}	…	z_{1n}
x_2	z_{21}	z_{22}	…	z_{2k}	…	z_{2n}
⋮	⋮	⋮	…	⋮	…	⋮
x_i	z_{i1}	z_{i2}	…	z_{ik}	…	z_{in}
⋮	⋮	⋮	…	⋮	…	⋮
x_m	z_{m1}	z_{m2}	…	z_{mk}	…	z_{mn}

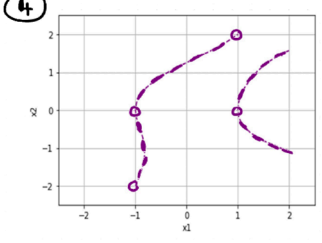
Graph

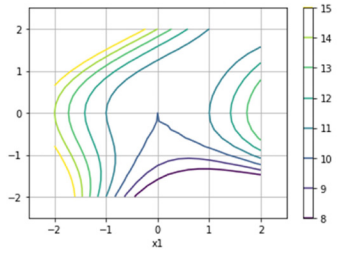


Höhenliniendiagramm (Relief)



Höhenlinie
= Kurve aller Punkte (x_1, x_2) mit gleicher Höhe





Partielle Ableitung

= Ableitung von Funktionen mit mehreren Variablen. (Serie 2)

Bsp. Ableitung von f nach x_1

$$\frac{\partial f}{\partial x_1}(x_1, x_2, \dots, x_n) \quad \text{oder} \quad f_{x_1}(x_1, x_2, \dots, x_n) \qquad \partial \text{ \partial}$$

Skalarwertige Funktionen

Es wird immer nur nach einem Paramter abgeleitet, die restlichen Paramater werden als Konstante betrachtet. Bsp. Ableitung von $f(x, y)$:

1. Partielle Ableitung von f nach x

$$\frac{\partial f}{\partial x}(x, y) = \frac{\partial}{\partial x}\left(x^2 - x \cdot y^2 + \frac{1}{2} \cdot y^3 + 10\right)$$

Konstante — konst. Faktor — konst. Summand

$$= 2 \cdot x - 1 \cdot y^2 + 0 + 0 = 2 \cdot x - y^2$$

2. Partielle Ableitung von f nach y

$$\frac{\partial f}{\partial y}(x, y) = \frac{\partial}{\partial y}\left(x^2 - x \cdot y^2 + \frac{1}{2} \cdot y^3 + 10\right)$$

Konstante — Summand — konst. Faktor — konst.

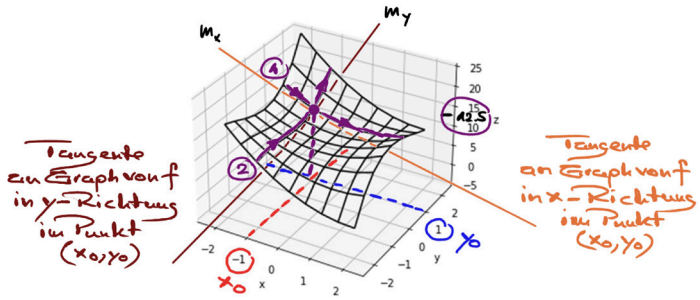
$$= 0 - x \cdot 2 \cdot y + \frac{3}{2} \cdot y^2 + 0 = -2 \cdot x \cdot y + \frac{3}{2} \cdot y^2$$

3. Partielle Ableitung von f an Stelle $(x_0, y_0) = (-1, 1)$

$$\frac{\partial f}{\partial x}(-1, 1) = 2 \cdot (-1) - 1^2 = -3 \quad = m_x$$

$$\frac{\partial f}{\partial y}(-1, 1) = -2 \cdot (-1) \cdot 1 + \frac{3}{2} \cdot 1^2 = 3.5 \quad = m_y$$

- Interpretation partielle Ableitung von f an Stelle $(x_0, y_0) = (-1, 1)$



Vektorwertige Funktionen

Es gibt $n \cdot m$ partielle Ableitungen: ***n* Paramter ··** und ***m* Funktionen :**

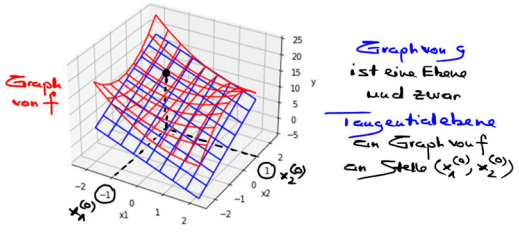
Jacobi-Matrix *Df* = Matrix der abgeleitenden Funktionen

$$Df(x) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \cdots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x) & \frac{\partial f_m}{\partial x_2}(x) & \cdots & \frac{\partial f_m}{\partial x_n}(x) \end{pmatrix}$$

Linearisierung:
= bilden der **Tangentialebene** (Funktion *g* an einer Stelle der Funktion *f*)

$$g(x) = f(x_0) + f'(x_0)(x - x_0) \qquad \text{Für skalarwertige Funktionen}$$

$$= \boldsymbol{f}(\boldsymbol{x}^{(0)}) + \boldsymbol{Df}(\boldsymbol{x}^{(0)}) \cdot (\boldsymbol{x} - \boldsymbol{x}^{(0)}) \qquad \text{Für vektorwertige Funktionen}$$



Bsp. nächste Seite

Bsp. Linearisierung

$f: \mathbb{R}^3 \to \mathbb{R}^2, \quad (x_1, x_2, x_3)^T \mapsto (y_1, y_2)^T$

$$f(\underline{x}) = \begin{pmatrix} x_1 \cdot \cos(x_2) + x_3 \\ x_1^2 + \sin(x_2) \cdot x_3^3 \end{pmatrix}, \quad \underline{x_0} = (1, 0, -1)^T$$

① DF bilden:

$$Df(\underline{x}) = \begin{pmatrix} \cos(x_2) & -x_1 \cdot \sin(x_2) & 1 \\ 2x_1 & \cos(x_2) \cdot x_3^3 & 3 \cdot \sin(x_2) \cdot x_3^2 \end{pmatrix}$$

② Auswerten an Stelle $\underline{x_0}$:

$$f(\underline{x_0}) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad Df(\underline{x_0}) = \begin{pmatrix} 1 & 0 & 1 \\ 2 & -1 & 0 \end{pmatrix},$$

$$\underline{x} - \underline{x_0} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} x_1 - 1 \\ x_2 \\ x_3 + 1 \end{pmatrix}$$

③ Tangentialebene (Linearisierung):

$$g(\underline{x}) = f(\underline{x_0}) + Df(\underline{x_0}) \cdot (\underline{x} - \underline{x_0}) = \begin{pmatrix} x_1 + x_3 \\ 2x_1 - x_2 - 1 \end{pmatrix}$$

Jede Funktion in $g(\underline{x})$ muss linear sein!

$g: \mathbb{R}^3 \to \mathbb{R}^2, \quad (x_1, x_2, x_3)^T \mapsto (m_1, m_2)^T$

Newton-Verfahren für Systeme

Nullstellenbestimmung von Funktionen mit mehreren Variablen. (Serie 3)

Gegeben sei eine Funktion $\boldsymbol{f}: \mathbb{R}^n \longrightarrow \mathbb{R}^n$.
Gesucht ist ein Vektor $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ mit $\boldsymbol{f}(\bar{\boldsymbol{x}}) = 0$.

$$\boldsymbol{f}(x_1, ..., x_n) = \begin{pmatrix} f_1(x_1, ..., x_n) \\ f_2(x_1, ..., x_n) \\ \vdots \\ f_n(x_1, ..., x_n) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Newton-Verfahren (Matrixversion)

Bei dieser Maxtrixversion wird die **Inverse** der Jacobi-Matrix *Df* berechnet.

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} - \left(\boldsymbol{Df}(\boldsymbol{x}^{(n)})\right)^{-1} \cdot \boldsymbol{f}(\boldsymbol{x}^{(n)})$$

Variante ohne Berechnung der Inverse:
Folgende Substitution

$$\boldsymbol{\delta}^{(n)} := -\left(\boldsymbol{Df}(\boldsymbol{x}^{(n)})\right)^{-1} \cdot \boldsymbol{f}(\boldsymbol{x}^{(n)})$$

wird als lineares Gleichungssystem ***Ax = y*** aufgefasst,
das bei jedem Iterantionsschritt gelöst werden muss:

$$\boldsymbol{Df}(\boldsymbol{x}^{(n)})\boldsymbol{\delta}^{(n)} = -\boldsymbol{f}(\boldsymbol{x}^{(n)})$$

```
delta = np.linalg.solve(Df(x), -f(x))
```

$$\boldsymbol{x}^{(n+1)} := \boldsymbol{x}^{(n)} + \boldsymbol{\delta}^{(n)}$$

$\boldsymbol{\delta}^{(n)}$ kann als Zuwachs verstanden werden, denn $\boldsymbol{\delta}^{(n)} = \boldsymbol{x}^{(n+1)} + \boldsymbol{x}^{(n)}$

Diese Methode ist **effizienter,** aber in der Umgebung von lokalen Maxima oder Minima von *f(x)* ist die Jacobi-Matrix *Df* **schlecht konditioniert** und (fast) **nicht invertierbar** → **Newton-Vefahren mit Dämpfung.**

Startvektor: $x^{(0)} = (0, 1, 1)^T$, Exakt: $x^* = (1, 2, 3)^T$

$$f(\underline{x}) = \begin{pmatrix} x_1 \cdot x_2 + x_3 - 5 \\ x_1^2 + x_2^2 - 5 \\ x_2^3 \cdot x_3 - 24 \end{pmatrix}, \quad Df(\underline{x}) = \begin{pmatrix} x_2, & x_1, & 1 \\ 2x_1, & 2x_2, & 0 \\ 0, & 3x_2^2 \cdot x_3, & 1 \end{pmatrix}$$

$$f(x^{(0)}) = \begin{pmatrix} -4 \\ -4 \\ -23 \end{pmatrix}, \quad Df(x^{(0)}) = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 0 \\ 0 & 3 & 1 \end{pmatrix}$$

LSG: $Df \cdot \delta = -f(x^{(0)})$

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 2 & 0 & 4 \\ 0 & 3 & 1 & 23 \end{array}\right) \Rightarrow \delta_2 = 2, \ \delta_3 = 17, \ \delta_1 = -13 \qquad \delta^{(0)} = (-13, 2, 17)^T$$

$$x^{(1)} = x^{(0)} + \delta^{(0)} \Rightarrow \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} -13 \\ 2 \\ 17 \end{pmatrix} = \begin{pmatrix} -13 \\ 3 \\ 18 \end{pmatrix}$$

Fazit: Verfahren divigiert oder konvergiert gegen eine andere Lösung.

Normales Newton-Vefahren mit Dämpfung

$$\boldsymbol{Df}(\boldsymbol{x}^{(n)})\boldsymbol{\delta}^{(n)} = -\boldsymbol{f}(\boldsymbol{x}^{(n)})$$

Oft konvergiert das Newton-Verfahren nicht. Falls die Jacobi-Matrix *Df*(*x*(*n*)) beim *n*-ten Iterationsschritt **schlecht konditioniert** ist → dämpfen bzw. **verkleinern der Schrittweite** $\delta^{(n)}$

Das gedämpfte Newton-Verfahren ist im Allgemeinen weit effizienter als das normale Newton-Verfahren oder andere Gradientenverfahren.

Wir akzeptieren einen Iterationsschritt erst wenn folgende Gleichung zutrifft.
D.h. finde das **minimale *k*** = {0, 1, ..., k_{max} }:

$$\| \boldsymbol{f}\left(\boldsymbol{x}^{(n)} + \frac{\boldsymbol{\delta}^{(n)}}{2^k}\right) \|_2 \ < \ \| \boldsymbol{f}\left(\boldsymbol{x}^{(n)}\right) \|_2$$

Falls **kein minimales *k*** gefunden werden kann, rechne mit ***k* = 0** weiter.

$$\boldsymbol{x}^{(n+1)} := \boldsymbol{x}^{(n)} + \frac{\boldsymbol{\delta}^{(n)}}{2^k}$$

Mögliche Abbruchkriterien:

$$\| \boldsymbol{x}^{(n+1)} - \boldsymbol{x}^{(n)} \| \leq \| \boldsymbol{x}^{(n+1)} \| \cdot \epsilon$$
$$\| \boldsymbol{x}^{(n+1)} - \boldsymbol{x}^{(n)} \| \leq \epsilon$$
$$\| \boldsymbol{f}\left(\boldsymbol{x}^{(n+1)}\right) \| \leq \epsilon$$

Konvergenzgeschwindigkeit:

$$| x_{n+1} - \overline{x} | \ \leq \ c \cdot | x_n - \overline{x} |^q$$

Das Newton-Vefahren (Matrixversion) hat **Konvergenzordnung *q* = 2** bzw. konvergiert **quadratisch.**

Fehler nach Schritt *i*+1 = (***C*** · Fehler)²

$\lvert x_1 - \overline{x} \rvert$	$\lvert x_2 - \overline{x} \rvert$	$\lvert x_3 - \overline{x} \rvert$	
0.5^1	0.05^1	0.005^1	$C = \frac{1}{10}$
$\cdot\frac{1}{16}$	$\cdot\frac{1}{10}$		$q = 1$

Vereinfachtes Newston-Vefahren

Die Jacobi-Matrix *Df* wird **nur einmal berechnet** mit *D* = *Df*(x_0).

Dieses Variante ist unstabiler und divergiert öfters als das normale Verfahren. Der Aufwand pro Schritt wird so zwar reduziert, wegen der linearen Konvergenzgeschwindigkeit sollte trotzdem das normale Verfahren mit quadratischer Geschwindigkeit verwendet werden. Dämpfung kann auch für das vereinfachte Vefahren verwendet werden.

$$\boldsymbol{D} = \boldsymbol{Df}(\boldsymbol{x}^{(0)})$$

$$\boldsymbol{D}\boldsymbol{\delta}^{(n)} = -\boldsymbol{f}(\boldsymbol{x}^{(n)})$$

```
delta = np.linalg.solve(D, -f(x))
```

$$\boldsymbol{x}^{(n+1)} := \boldsymbol{x}^{(n)} + \boldsymbol{\delta}^{(n)}$$

Konvergenzgeschwindigkeit:

Das vereinfachte Newton-Vefahren hat **Konvergenzordnung *q* = 1** bzw. konvergiert **linear.**

Fehler nach Schritt *i*+1 = ***C*** · (Fehler nach Schritt *i*)

$\lvert x_1 - \overline{x} \rvert$	$\lvert x_2 - \overline{x} \rvert$	$\lvert x_3 - \overline{x} \rvert$	
$0.25 = 0.5^2$	$0.0025 = 0.05^2$	$0.000025 = 0.005^2$	$C = \frac{1}{10}$
$\cdot\frac{1}{10}$	$\cdot\frac{1}{10}$		$q = 2$

Polynominterpolation

Ausgleichsrechnung (Serie 4)

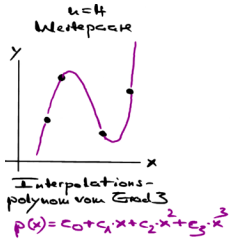
Interpolation vs. Regression

Interpolation: Funkiton geht durch alle *m* Punkte/Wertepaare (x_i, y_i)
Regression: Funktion nähert Punkte möglichst gut an.

Interplationspolynom *P(x)*

Für *m* Stützpunkte/Wertepaare wird ein Polynon *P* von Grad *m–1* bzw. mit *m* Koeffizienten benötigt = Gesamthafte Interpolation der Daten.

$$\begin{aligned} P_n(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_n x_0^n &= y_0 \\ \vdots \quad \vdots \quad \vdots & \\ P_n(x_n) = a_0 + a_1 x_n + a_2 x_n^2 + \ldots + a_n x_n^n &= y_n \end{aligned}$$

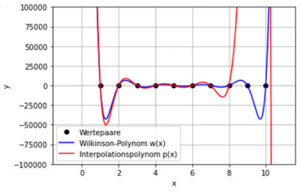


Vandermonde-Matrix *V*

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_n \end{pmatrix}$$

Das **LGS** bzs. die Koeffizienten *a* können z.B. mit dem Gauss-Verfahren gelöst werden.

Problem: Die Vandermonde-Matrix ist in der Regel sehr schlecht konditioniert und durch Lösen des LGS stark fehlerbehaftet.



Vegleich zu Wilkinson-Polynom (Wertepaare *x* = 1, 2, ... und *y* = 0):

$$p(x) = \prod_{k=1}^{n} (x - k)$$

Lagrange Interpolationsformel

Bei *n*+1 Knotenpunkte werden *n*+1 Polynome von Grad *n* berechnet.

$$\begin{aligned} P_n(x) &= \sum_{i=0}^{n} l_i(x) y_i \\ &= l_0(x) \cdot y_0 + l_1(x) \cdot y_1 + \ldots + l_n(x) \cdot y_n \end{aligned}$$

Lagrange-Polynome (Bsp. mit Grad *n* = 3, d.h. n+1 Knotenpunkte)

$$l_0(x) = \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)}$$

$$l_1(x) = \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)}$$

$$\ldots$$

$$l_3(x) = \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)}$$

Numerisch gesehen ist diese Formel problemlos. Erst wenn die x-Werte nahe beieinander liegen besteht Gefahr auf Auslöschung (durch Substraktion).

Beispiel von Hand siehe **nächste Seite** oder Serie 4, Aufgabe 1.

Fehlerabschätzung (absoluter Maximalfehler)

Approximation einer Funktion durch Polynominterpolation.
Tatsächlich absoluter Fehler

$$| f(x) - P_n(x) | \leq \frac{| (x - x_0)(x - x_1) \cdots (x - x_n) |}{(n+1)!} \cdot \max_{x_0 \leq \xi \leq x_n} | f^{(n+1)}(\xi) |$$

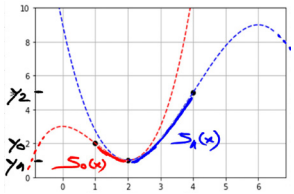
Splineinterpolation

Ausgleichsrechnung (Serie 5)

Kubische Splineinterpolation

Stückweise Interpolitaion durch *m–1* Polyome von jeweils Grad 3.
m Stützpunkte
m–1 Polynome
4 · *m* Gleichungen (4 unbekannte Koeffizienten je Polynom)

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$



Übergangsbedingungen

Die Polynome s_i müssen dann folgende Bedingungen erfüllen:

$$\begin{aligned} s_i(x_i) &= y_i, & s_{i+1}(x_{i+1}) &= y_{i+1}, \ldots && \text{Interpolation} \\ s_i(x_{i+1}) &= s_{i+1}(x_{i+1}), & s_{i+1}(x_{i+2}) &= s_{i+2}(x_{i+2}) \ldots && \text{stetiger Übergang} \\ s_i'(x_{i+1}) &= s_{i+1}'(x_{i+1}), & s_{i+1}'(x_{i+2}) &= s_{i+2}'(x_{i+2}), \ldots && \text{keine Knicke} \\ s_i''(x_{i+1}) &= s_{i+1}''(x_{i+1}), & s_{i+1}''(x_{i+2}) &= s_{i+2}''(x_{i+2}), \ldots && \text{gleiche Krümmung} \end{aligned}$$

(Übergangspunkte haben die gleiche Krümmung)

Randbedingungen

Natürlich

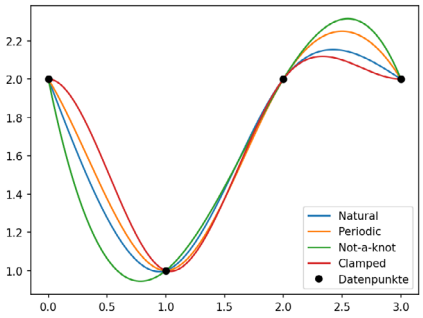
$$\begin{aligned} S_0''(x_0) &= 0 \\ S_2''(x_3) &= 0 \end{aligned}$$

Periodisch

$$\begin{aligned} S_0'(x_0) &= S_2'(x_3) \\ S_0''(x_0) &= S_2''(x_3) \end{aligned}$$

Not-a-Knot

$$\begin{aligned} S_0'''(x_1) &= S_1'''(x_1) \\ S_1'''(x_2) &= S_2'''(x_2) \end{aligned}$$



Lösungsalgorithmus

Koeffizienten a_i, b_i, c_i und d_i der Polynome $S_i(x)$ für $i = 0, 1, ..., n-1$ berechnen

1. $a_i = y_i$
2. $h_i = x_{i+1} - x_i$
3. $c_0 = 0,\ c_n = 0$
4. Berechnung der Koeffizienten $c_1, c_2, ..., c_{n-1}$ aus dem Gleichungssystem
   (a) $i = 1$:
   $2(h_0 + h_1)c_1 + h_1 c_2 = 3\frac{y_2 - y_1}{h_1} - 3\frac{y_1 - y_0}{h_0}$
   (b) falls $n \geq 4$ gilt für $i = 2, ..., n-2$:
   $h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} = 3\frac{y_{i+1} - y_i}{h_i} - 3\frac{y_i - y_{i-1}}{h_{i-1}}$
   (c) $i = n - 1$:
   $h_{n-2}c_{n-2} + 2(h_{n-2} + h_{n-1})c_{n-1} = 3\frac{y_n - y_{n-1}}{h_{n-1}} - 3\frac{y_{n-1} - y_{n-2}}{h_{n-2}}$
5. $b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{3}(c_{i+1} + 2c_i)$
6. $d_i = \frac{1}{3h_i}(c_{i+1} - c_i)$

Lösen des LGS:

$$\boldsymbol{Ac} = \boldsymbol{z}$$

$$\boldsymbol{A} = \begin{pmatrix} 2(h_0 + h_1) & h_1 & & & & \\ h_1 & 2(h_1 + h_2) & h_2 & & & \\ & h_2 & 2(h_2 + h_3) & h_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-3} & 2(h_{n-3} + h_{n-2}) & h_{n-2} \\ & & & & h_{n-2} & 2(h_{n-2} + h_{n-1}) \end{pmatrix}$$

$$\boldsymbol{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix}, \quad \boldsymbol{z} = \begin{pmatrix} 3\frac{y_2 - y_1}{h_1} - 3\frac{y_1 - y_0}{h_0} \\ 3\frac{y_3 - y_2}{h_2} - 3\frac{y_2 - y_1}{h_1} \\ \vdots \\ 3\frac{y_n - y_{n-1}}{h_{n-1}} - 3\frac{y_{n-1} - y_{n-2}}{h_{n-2}} \end{pmatrix}$$

Beispiel siehe Serie 5, Aufgabe 1.

Bsp. Splineinterpolation mit Lagrange

$f(x) = \sin(x), \quad 0 \le x \le \frac{\pi}{2}$

$\underline{x} = (0, \frac{\pi}{6}, \frac{\pi}{2}), \quad \underline{y} = (0, \frac{1}{2}, 1), \quad x^* = \frac{\pi}{3}$

Berechne $p(x^*)$ mit Lagrange:

$n = 2$ (grad)

$$P_2(x) = \sum_{i=0}^{2} L_i(x) \cdot y_i$$

$$= 0 \cdot L_0(\tfrac{\pi}{3}) + \tfrac{1}{2} \cdot L_1(\tfrac{\pi}{3}) + 1 \cdot L_2(\tfrac{\pi}{3}) = \frac{5}{6}$$

$$L_1(\tfrac{\pi}{3}) = \frac{(\frac{\pi}{3} - 0)(\frac{\pi}{3} - \frac{\pi}{2})}{(\frac{\pi}{6} - 0)(\frac{\pi}{6} - \frac{\pi}{2})} = 1$$

$$L_2(\tfrac{\pi}{3}) = \frac{(\frac{\pi}{3} - 0)(\frac{\pi}{3} - \frac{\pi}{6})}{(\frac{\pi}{2} - 0)(\frac{\pi}{2} - \frac{\pi}{6})} = \frac{\frac{\pi}{3} \cdot \frac{\pi}{6}}{\frac{\pi}{2} \cdot \frac{\pi}{3}} = \frac{1}{3}$$

Lineare Ausgleichsprobleme (Regression)

Ausgleichsrechnung (Serie 6)

Datenpunkte werden durch eine Funktion angenähert und möglichst gut approximiert. Anwendung: Wenn es eine grosse Anzahl von Datenpunkten gibt (die meist zusätzlich noch fehlerbehaftet sind).

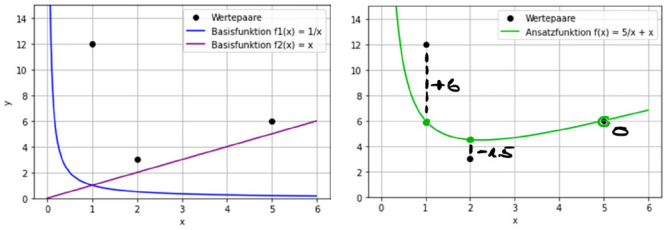
Ansatzfunktionen $f \in F$

Kandidat-Funktionen. Die **Ansatzfunktion** f ist eine Linearkombination von festen **Basisfunktionen** f_i

$$f(x) = \lambda_1 \cdot \underbrace{\sin(x)}_{f_1(x)} + \lambda_2 \cdot \underbrace{\cos(x)}_{f_2(x)} + \lambda_3 \cdot \underbrace{x}_{f_3(x)} + \lambda_4 \cdot \underbrace{\exp(x)}_{f_4(x)} + \dots$$

Bps:

$f(x) = -2x + 6 \Rightarrow f_1(x) = x, \; f_2(x) = 1, \; \lambda_1 = -2, \; \lambda_2 = 6$



Nicht-linear wenn gesuchte Ansatzfunktion f sich nicht als Linearkombination schreiben lässt. Wenn es sich verkettete Funktionen handelt (**Konstante** mal x), z.B: $f(x) = \lambda_1 \cos(\lambda_2 x)$

Fehlerfunktional $E(f)$

Error. Wie gut eine **Ansatzfunktion** die Datenpunkte approximiert. Desto kleiner umso besser. Im Regelfall immer positiv.
Null wenn die Ausgleichsfunktion durch alle Punkte geht (Spezialfall).

$$E(f) = \| \boldsymbol{y} - f(\boldsymbol{x}) \|_2^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Ausgleichsfunktion / Regressfunktion $f \in F$

Die gesuchten **Parameter** $\lambda_1, \dots, \lambda_m$ für welche die **Ansatzfunktion** $f \in F$ für welche $E(f)$ **am kleinsten** ist. Sie besitzt das kleinste Fehlerquadrat (least square fit).

Fehlergleichungssystem (Vektorisierung)

Nicht lösbares Gleichungssystem.

$A\lambda = y$ heisst Fehlergleichungssystem, $y - A\lambda$ heisst Fehlervektor

A = Basisfunktionenwert-Matrix

$$\boldsymbol{A} = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_m(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(x_n) & f_2(x_n) & \cdots & f_m(x_n) \end{pmatrix}, \; \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \; \boldsymbol{\lambda} = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_m \end{pmatrix}$$

***n* Gleichungen ≥ *m* Unbekannte** (A ist keine quadratische Matrix).

Regelfall $n > m$: Das Fehlergleichungssystem ist überbestimmt (mehr Gleichungen als Unbekannte) und hat **keine Lösung → man kann nicht damit rechnen.** Spezialfalls $n = m$: Das Fehlergleichungssystem hat genau ein Lösung λ. Dann ist $E(f) = 0$ (Ausgleichsfunktion geht durch alle Puntke).

Normalengleichung

Ausgleichsrechnung > Lineare Ausgleichsprobleme (Regression) (Serie 6)

Lösungsverfahren für **lineare Gleichungssysteme.**

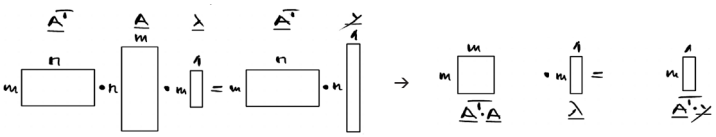
Bei einem **linearen (oder nicht-linearen)** Ausgleichsproblem hat die Ansatzfunktion das kleinste Fehlerfunktional wenn die **Parameter λ Lösung** des Gleichungssystems sind, dann gilt:

$$\frac{\partial E(f)}{\partial \lambda_j} = 0 \qquad j = 1, \dots, m$$

Normalengleichungssystem

$$\boldsymbol{A^T A \lambda = A^T y} \qquad \boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_m)^T$$

m Gleichungen für m Unbekannte $\lambda_1, \dots, \lambda_m$, in der Regel $|y| \ge |\lambda|$
Hat im Regelfall genau eine Lösung.
Dieses LGS ist wegen A^TA **schlecht konditioniert** → QR-Verfahren ist besser.



QR-Zerlegung

Verfahren zum lösen des Normalengleichungssystems.
I.d.R. besser konditioniert als das Normalengleichungssystem.

Q ist eine Orthogonalmatrix, d.h. $\boldsymbol{Q^T Q = I_n}$
R ist eine rechtsobere Dreiecksmatrix

$$\boldsymbol{A^T A \lambda = A^T y}$$
$$\boldsymbol{(QR)^T (QR) \lambda = (QR)^T y}$$

$$\boldsymbol{R\lambda = Q^T y} \quad \rightarrow \lambda \text{ ist durch } \textbf{Rückwärtseinsetzen} \text{ lösbar}$$
$$\boldsymbol{y = A\lambda}$$

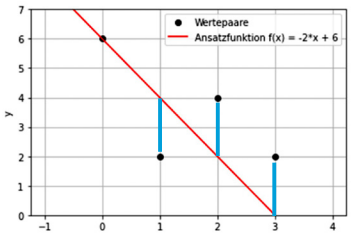
```
Q, R = np.linalg.qr(A)
lambdas = np.linalg.solve(R, Q.T @ y) # gauss_pivot(R, Q.T @ y)
y_fit = A @ lambdas
```

Die gesuchten Parameter λ eines nichtlilinearen Ausgleichsproblems könnten ebenfalls über die Lösung des Normalgleichungssystems bestimmt werden, es gibt aber bessere Methoden.

Fehlerfunktional $E(f)$

$$E(f) = \| \boldsymbol{y} - \boldsymbol{A\lambda} \|_2^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{m} \lambda_j f_j(x_i) \right)^2$$

Bsp:



$E(f) = (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2 + \dots + (y_n - f(x_n))^2$

$= 0^2 + (-2)^2 + 2^2 + 2^2 = 12$

Nicht-lineare Ausgleichsprobleme (Regression)

Ausgleichsrechnung (Serie 7)

Bei einem **nicht-linearen (oder linearen)** Ausgleichsproblem hat die Ansatzfunktion das kleinste Fehlerfunktional wenn die **Parameter λ Lösung** des Gleichungssystems sind, dann gilt:

$$\frac{\partial E(f)}{\partial \lambda_j} = 0 \quad j = 1, ..., m$$

Beispiel siehe nächste Seite

Fehlerfunktional

Nichtlineare Ausgleichsprobleme sind **Quadratmittelprobleme,** das Problem, einen Vektor $\boldsymbol{\lambda} \in \mathbb{R}^m$ zu finden, für den $E(\boldsymbol{\lambda})$ bzw. $E(f)$ **minimiert** wird.

$$\begin{aligned} E(f) &= \sum_{i=1}^{n} (y_i - f_p(\lambda_1, \lambda_2, ..., \lambda_m, x_i))^2 \\ &= \| \begin{pmatrix} y_1 - f_p(\lambda_1, \lambda_2, ..., \lambda_m, x_1) \\ y_2 - f_p(\lambda_1, \lambda_2, ..., \lambda_m, x_2) \\ \vdots \\ y_n - f_p(\lambda_1, \lambda_2, ..., \lambda_m, x_n) \end{pmatrix} \|_2^2 \\ &= \| \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\lambda}) \|_2^2 \end{aligned}$$

$\mathbf{y} \in \mathbb{R}^n, \boldsymbol{\lambda} \in \mathbb{R}^m$
$E : \mathbb{R}^m \longrightarrow \mathbb{R}$
$\boldsymbol{g} : \mathbb{R}^m \longrightarrow \mathbb{R}^n$

$$E(\boldsymbol{\lambda}) := \| \boldsymbol{g}(\boldsymbol{\lambda}) \|_2^2 = \| \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\lambda}) \|_2^2$$

$$\boldsymbol{g}(\boldsymbol{\lambda}) \approx \underbrace{\boldsymbol{g}(\boldsymbol{\lambda}^{(0)})}_{\widetilde{\boldsymbol{y}}} + \underbrace{\boldsymbol{Dg}(\boldsymbol{\lambda}^{(0)})}_{-\widetilde{\boldsymbol{A}}} \cdot \underbrace{(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{(0)})}_{\boldsymbol{\delta}}$$

$$\tilde{E}(\boldsymbol{\lambda}) = \| \widetilde{\boldsymbol{y}} - \widetilde{\boldsymbol{A}}\boldsymbol{\delta} \|_2^2$$

← Ein nichtlineares Ausgleichsproblem wird auf ein lineares Ausgleichsproblem zurückgeführt.

Fehlergleichungssystem (Vektorisierung)

Spezialfall: Ein lineares Ausgleichsproblem kann als Quadratmittelproblem interpretiert werden mit der Funktion:

$\boldsymbol{g(\lambda) = y - A\lambda}$ wobei $E(\boldsymbol{\lambda}) = ||\boldsymbol{g}(\boldsymbol{\lambda})||_2^2$ **minimiert** werden muss.

```
E = np.linalg.norm(g(l), 2)**2
```

Gauss-Newton-Verfahren (ungedämpft)

Sei $\boldsymbol{\lambda}^{(0)}$ ein Startvektor in der Nähe des Minimums des Fehlerfunktionals E der Ausgleichsfunktion $f(\boldsymbol{\lambda})$. Für $k = 0, 1, ...$:

1. Berechne die Funktion

$$\boldsymbol{g}(\boldsymbol{\lambda}) = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\lambda}) = \begin{pmatrix} g_1(\lambda_1, ..., \lambda_m) \\ \vdots \\ g_n(\lambda_1, ..., \lambda_m) \end{pmatrix} = \begin{pmatrix} y_1 - f(x_1; \lambda_1, ..., \lambda_m) \\ \vdots \\ y_n - f(x_n; \lambda_1, ..., \lambda_m) \end{pmatrix}$$

2. Jacobi-Matrix Dg ($n \times m$): Partielle Ableitungen von $g(\boldsymbol{\lambda})$

$$\boldsymbol{Dg}(\boldsymbol{\lambda}^{(0)}) := \begin{pmatrix} \frac{\partial g_1}{\partial \lambda_1}(\boldsymbol{\lambda}^{(0)}) & \frac{\partial g_1}{\partial \lambda_2}(\boldsymbol{\lambda}^{(0)}) & \cdots & \frac{\partial g_1}{\partial \lambda_m}(\boldsymbol{\lambda}^{(0)}) \\ \frac{\partial g_2}{\partial \lambda_1}(\boldsymbol{\lambda}^{(0)}) & \frac{\partial g_2}{\partial \lambda_2}(\boldsymbol{\lambda}^{(0)}) & \cdots & \frac{\partial g_2}{\partial \lambda_m}(\boldsymbol{\lambda}^{(0)}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial \lambda_1}(\boldsymbol{\lambda}^{(0)}) & \frac{\partial g_n}{\partial \lambda_2}(\boldsymbol{\lambda}^{(0)}) & \cdots & \frac{\partial g_n}{\partial \lambda_m}(\boldsymbol{\lambda}^{(0)}) \end{pmatrix}$$

3. Löse das **Normalgleichungssystem** zur Berechnung von $\boldsymbol{\delta}^{(k)}$ (delta)

$$\widetilde{\boldsymbol{A}}^T \widetilde{\boldsymbol{A}} \boldsymbol{\delta} = \widetilde{\boldsymbol{A}}^T \widetilde{\boldsymbol{y}}$$

$$\boldsymbol{Dg}(\boldsymbol{\lambda}^{(k)})^T \boldsymbol{Dg}(\boldsymbol{\lambda}^{(k)}) \boldsymbol{\delta}^{(k)} = -\boldsymbol{Dg}(\boldsymbol{\lambda}^{(k)})^T \cdot \boldsymbol{g}(\boldsymbol{\lambda}^{(k)})$$

Wegen der besseren **Konditionierung** wir das Normalgleichungssystem nicht mit dem Gauss-Algorithmus sondern mit der QR-Zerlegung gelöst:

(a) Berechne die QR-Zerlegung

$$\boldsymbol{Dg}(\boldsymbol{\lambda}^{(k)}) = \boldsymbol{Q}^{(k)} \boldsymbol{R}^{(k)}$$

(b) Löse nach $\boldsymbol{\delta}^{(k)}$ auf durch Räckwärtseinsetzen

$$\boldsymbol{R}^{(k)} \boldsymbol{\delta}^{(k)} = -\boldsymbol{Q}^{(k)\,T} \boldsymbol{g}(\boldsymbol{\lambda}^{(k)})$$

4. Setze

$$\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \boldsymbol{\delta}^{(k)}$$

$\boldsymbol{\delta}$ (delta) liefert uns eine Korrektur/Verbesserung zum Startwert $\lambda^{(0)}$.

Gedämpftes Gauss-Newton-Verfahren

Konvergiert im der Regel für einen grösseren Bereich vom Startvektor als das ungedäpfte Verfahren.
Dämpfung ist aber **keine Garantie** für Konvergenz.

Schritte **1. bis 3.** wie im ungedäpften Verfahren.

4. Finde das minimale $p \in \{0, 1, ..., p_{max}\}$ mit

$$\| \boldsymbol{g}\underbrace{\left(\boldsymbol{\lambda}^{(k)} + \frac{\boldsymbol{\delta}^{(k)}}{2^p}\right)}_{\boldsymbol{\lambda}^{(k+1)}} \|_2^2 < \| \boldsymbol{g}\left(\boldsymbol{\lambda}^{(k)}\right) \|_2^2$$

Falls kein minimales p gefunden werden kann, rechne mit $p = 0$ weiter.

5. Setze

$$\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \frac{\boldsymbol{\delta}^{(k)}}{2^p}$$

6. Überprüfe nach jedem Iterationsschritt folgende Bedingung

$$E(\boldsymbol{\lambda}^{(k+1)}) = \| \boldsymbol{g}(\boldsymbol{\lambda}^{(k+1)}) \|_2^2 < \| \boldsymbol{g}(\boldsymbol{\lambda}^{(k)}) \|_2^2 = E(\boldsymbol{\lambda}^{(k)})$$

7. Abbruchkriterium

$$\| \frac{\boldsymbol{\delta}^{(k)}}{2^p} \|_2 < TOL$$

Beispiel mit Python: Serie 7, Aufgabe 2

Einfache, nicht-lineare Regression

Ausgangsfunktion	Transformation
$y = q \cdot x^m$	$\log(y) = \log(q) + m \cdot \log(x)$
$y = q \cdot m^x$	$\log(y) = \log(q) + \log(m) \cdot x$
$y = q \cdot e^{m \cdot x}$	$\ln(y) = \ln(q) + m \cdot x$
$y = \frac{1}{q + m \cdot x}$	$V = q + m \cdot x; \quad V = \frac{1}{y}$
$y = q + m \cdot \ln(x)$	$y = q + m \cdot U; \quad U = \ln(x)$
$y = \frac{1}{q \cdot m^x}$	$\log\left(\frac{1}{y}\right) = \log(q) + \log(m) \cdot x$

Vorgehen

1. y-Werte transformieren gemäss Tabelle (e.g. $\log(y_i)$ oder $\ln(y_i)$)
2. m und d berechnen (mit neuen transformieren y-Werten)
3. m und d Rücktransformieren (e.g. 10^m und 10^b resp. e^m und e^d)
4. Rücktransformierte m- und d-Werte in Ausgangsfunktion einsetzen

Ausgangsfunktion

$\underbrace{f(x)}_{y} = \lambda_1 \cdot e^{\lambda_2 \cdot x}$

1. Transformation (Linearisierung)

$\underbrace{\ln(f(x))}_{\tilde{y}} = \ln(\lambda_1 e^{\lambda_2 \cdot x}) = \ln(\lambda_1) + \lambda_2 \cdot x$

2. Ansatzfunktionen

$\tilde{f}(x) = \tilde{\lambda}_1 \cdot \underbrace{1}_{f_1(x)} + \tilde{\lambda}_2 \cdot \underbrace{x}_{f_2(x)} \Rightarrow \tilde{f}(x) = \ln(f(x))$, $\tilde{\lambda}_1 = \ln(\lambda_1)$, $\tilde{\lambda}_2 = \lambda_2$, $f_1(x) = 1$, $f_2(x) = x$

3. Matrix A bilden und LGS lösen

$A = \begin{pmatrix} f_1(x_1) & f_2(x_1) \\ \vdots & \vdots \\ f_1(x_n) & f_2(x_n) \end{pmatrix}$, $R\tilde{\lambda} = Q^T \underbrace{\ln(f(x))}_{\tilde{y}} \Rightarrow \tilde{\lambda}_1, \tilde{\lambda}_2$

4. Rücktransformation

$\lambda_1 = e^{\tilde{\lambda}_1}, \quad \lambda_2 = e^{\tilde{\lambda}_2}$

$\Rightarrow$ in Ausgangsfunktion einsetzen.

Beispiel mit Python: Serie 6, Aufgabe 3

Bsp. Berechnen von:

$$\frac{\partial E(f)}{\partial \lambda_j} = 0 \qquad j = 1, \ldots, m$$

Gegeben:

Wertepaare $(x_1, y_1) = (0,1), (x_2, y_2) = (1,2), (x_3, y_3) = (2,8)$

Ansatzfunktionen $f(x) = \lambda_1 \cdot e^{\lambda_2 \cdot x}$

1. Fehlerfunktional *E(f)* bestimmen:

Fehlerfunktional

$$E(f) = (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2 + (y_3 - f(x_3))^2$$
$$= (1 - \lambda_1 \cdot e^{\lambda_2 \cdot 0})^2 + (2 - \lambda_1 \cdot e^{\lambda_2 \cdot 1})^2 + (8 - \lambda_1 \cdot e^{\lambda_2 \cdot 2})^2$$

2. Ableitungen von *E(f)* nach λ_1 und λ_1

$E(\lambda_2, \lambda_2)$ minimal $\Rightarrow \frac{\partial E}{\partial \lambda_1} = 0$ und $\frac{\partial E}{\partial \lambda_2} = 0$

$$\frac{\partial E}{\partial \lambda_1} = 2 \cdot (1 - \lambda_1) \cdot (-1) + 2 \cdot (2 - \lambda_1 \cdot e^{\lambda_2}) \cdot (-e^{\lambda_2}) + 2 \cdot (8 - \lambda_1 \cdot e^{2\lambda_2}) \cdot (-\lambda_1 \cdot e^{2\lambda_2}) = 0$$

$$\frac{\partial E}{\partial \lambda_2} = 0 + 2 \cdot (2 - \lambda_1 \cdot e^{\lambda_2}) \cdot (-\lambda_1 e^{\lambda_2}) + 2 \cdot (8 - \lambda_1 \cdot e^{2\lambda_2}) \cdot (-2 \cdot \lambda_1 \cdot e^{2\lambda_2}) = 0$$

3. λ_1 und λ_1 einsetzen

Numerische Integration

(Serie 8/9)

Problemstellung: Gesucht ist eine möglichst genauer Näherngswert eines bestimmten Integrals. Viele Integrale sind algebraisch nicht berechenbar. $\int_a^b f(x)\,dx$

Quadraturformeln

Zur Approximation von ein bestimmen Integrals:

Rechteckregel ***Rf***
Approximation durch konstante Funktion(en). Hat Fehlerordnung 2

Trapezregel ***Rf***
Approximation durch lineare Funktion(en). Hat Fehlerordnung 2

Simpson-Regel ***Sf***
Approximation durch quadratische Funktion(en). Hat Fehlerordnung 4

Rechteckregel (auch Mittelpunktsregel) und Trapezregel

$$Rf = f(\frac{a+b}{2}) \cdot (b-a) \qquad Tf = \frac{f(a)+f(b)}{2} \cdot (b-a)$$

Summierte Rechteckregel / Trapezregel

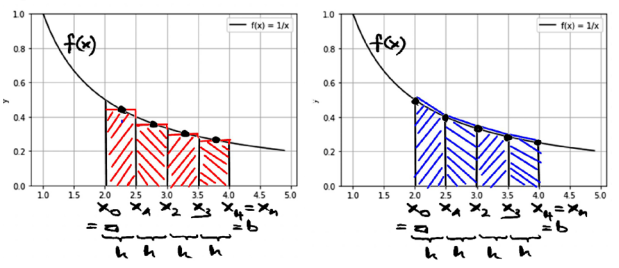
$$Rf(h) = h \cdot \sum_{i=0}^{n-1} f(x_i + \frac{h}{2})$$

$$Tf(h) = h \cdot \left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i) \right)$$

Sei $n \in \mathbb{N}$ die Anzahl Subintervalle $[x_i, x_{i+1}]$ auf $[a, b]$

$h = \frac{b-a}{n}$ Balkenbreite bzw. Schrittweite $= x_{i+1} - x_i$

$x_i = a + i \cdot h \qquad i = 0, \ldots, n-1$



Beispiel von Hand: HM2_Woche09_Vorlesungsnotizen_adel.pdf, Seite 4

Simpson-Regel

$$f(x) \approx p(x) = \alpha + \beta(x-a) + \gamma(x-a)(x-b)$$

Löst man das LSG erhält man:

$$\alpha = f(a)$$

$$\beta = \frac{f(b)-f(a)}{b-a}$$

$$\gamma = \frac{f(\frac{b+a}{2}) - f(a) - \frac{f(b)-f(a)}{b-a} \cdot (\frac{b-a}{2})}{-(\frac{b-a}{2})^2} = \frac{f(a) - 2f(\frac{b+a}{2}) + f(b)}{2(\frac{b-a}{2})^2}$$

Simpsonregel (Näherungspoylnom integriert):

$$\int_a^b f(x)dx \approx \int_a^b p(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Summierte Simpsonregel

$$Sf(h) \equiv \frac{h}{3}\left(\frac{1}{2}f(a) + \sum_{i=1}^{n-1} f(x_i) + 2\sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right) + \frac{1}{2}f(b)\right)$$

Beispiel von Hand: HM2_Woche09_Vorlesungsnotizen_adel.pdf, Seite 10

Fehlerabschätzung (maximaler absoluter Fehler)

Genauigkeit: (Ordnung)
Simpson-Regel (4) > Gauss-Formel (4) > Rechteckregel (2) > Trapezregel (4)

Der Fehler der summierten Rechteckregel ist um den Faktor 2 kleiner, als die der summierten Trapezregel.

Je gekrümmter eine Funktion ist, umso schwieriger ist es, sie zu approximieren.

$$\left| \int_a^b f(x)\,dx - Rf(h) \right| \leq \frac{h^2}{24}(b-a) \cdot \max_{x \in [a,b]} |f''(x)|$$

Rechteckregel: Wird *h* um Faktor 10 verkleinert, so verkleinert sich der absolute Fehler um Faktor 100 (10^2) = **Fehlerordnung 2**

$$\left| \int_a^b f(x)\,dx - Tf(h) \right| \leq \frac{h^2}{12}(b-a) \cdot \max_{x \in [a,b]} |f''(x)|$$

Trapezregel: Wird *h* um Faktor 10 verkleinert, so verkleinert sich der absolute Fehler um Faktor 100 (10^2) = **Fehlerordnung 2**

$$\left| \int_a^b f(x)\,dx - Sf(h) \right| \leq \frac{h^4}{2880}(b-a) \cdot \max_{x \in [a,b]} |f^{(4)}(x)|$$

Simpson-Regel: Wird *h* um Faktor 10 verkleinert, so verkleinert sich der absolute Fehler um Faktor 10'000 (10^4) = **Fehlerordnung 4**

Ab einer Anzahl Intervalle *n* wird der **Fehler wieder grösser** wegen bemerkbaren Rundungsfehler aufgrund vieler Rechenoperationen.

Anzahl Schritte *n*

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

Schrittweite *h* berechnen

Rechteckregel:

$$h = \sqrt{\frac{24 \cdot \epsilon}{(b-a) \cdot \max|f''(x)|}}$$

Trapezregel:

$$h = \sqrt{\frac{12 \cdot \epsilon}{(b-a) \cdot \max|f''(x)|}}$$

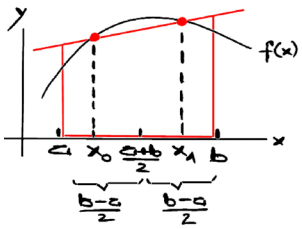
Simpson-Regel:

$$h = \left(\frac{2880 \cdot \epsilon}{(b-a) \cdot \max|f^{(4)}(x)|}\right)^{\frac{1}{4}}$$

Gauss-Formel

$$G = \frac{h}{2} \cdot (f(m-\alpha) + f(m+\alpha))$$

$$m = \frac{a+b}{2}, \quad h = b-a, \quad \alpha = \frac{1}{\sqrt{3}} \cdot \frac{h}{2}$$



Summierte Gauss-Formel

$$G = \frac{h}{2} \sum_{i=0}^{n-1} f(m_i - \alpha) + f(m_i + \alpha) \qquad m_i = a + i \cdot h + \frac{h}{2}$$

Wird h um Faktor 10 verkleinert, so verkleinert sich der absolute Fehler um Faktor 10'000 (10^{*4}) = **Fehlerordnung 4**

Romberg-Extrapolation

Idee: Schrittweiter *h* fortlaufend halbieren und summierte Trapezformel *T*(*h*) miteinander kombinieren.

1. Für $j = 0, 1, ..., m$

$$T_{j0} = Tf\left(h_j\right) = h_j \cdot \left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n_j - 1} f(x_i)\right)$$

$$h_j = \frac{h}{2^j} = \frac{b-a}{2^j} \qquad x_i = a + ih_j \qquad n_j = \frac{a-b}{h_j} = 2^j$$

2. Für $j = 0, 1, ..., m-k, \quad k = 1, 2, ..., m$

$$T_{jk} = \frac{4^k \cdot T_{j+1,k-1} - T_{j,k-1}}{4^k - 1}$$

Beispiel mit $m = 3$ Stufen

T_{j0}	T_{j1}	T_{j2}	T_{j3}
T_{00}			
	$T_{01} = \frac{4T_{10}-T_{00}}{3}$		
T_{10}		$T_{02} = \frac{16T_{11}-T_{01}}{15}$	
	$T_{11} = \frac{4T_{20}-T_{10}}{3}$		$T_{03} = \frac{64T_{12}-T_{02}}{63}$
T_{20}		$T_{12} = \frac{16T_{21}-T_{11}}{15}$	
	$T_{21} = \frac{4T_{30}-T_{20}}{3}$		
T_{30}			

Lösung ist T_{0m} bzw. T_{03}

Fehlerordnung beträgt **2(*m* + 1)** bzw. **2*k* + 2**

Beispiel von Hand: Serie 9 Aufgabe 2

Beispiel Runge-Kutta (Woche 12, Seite 12)

AWP: $y' = \frac{x}{y}$, $y(1) = 2$

$c_1 = 0, \quad c_2 = 1/3, \quad c_3 = 2/3 \qquad x_0 = 1$

$b_1 = 1/4, \quad b_2 = 0, \quad b_3 = 3/4 \qquad y_0 = 2$

$a_{21} = 1/3, \quad a_{31} = 0, \quad a_{32} = 2/3 \qquad h = 3$

$i = 0$

$k_1 = f(x_0 + c_1 \cdot h, y_0) = \frac{1}{2} = 0.5$

$k_2 = f(x_0 + c_2 \cdot h, y_0 + h \cdot a_{21} \cdot k_1) = 0.8$

$k_3 = f(x_0 + c_3 \cdot h, y_0 + h \cdot (a_{31} \cdot k_1 + a_{32} \cdot k_2)) = 0.8\overline{3}$

$k = b_1 \cdot k_1 + b_2 \cdot k_2 + b_3 \cdot k_3 = 0.75$

$x_1 = x_0 + h = 4$

$y_1 = y_0 + h \cdot k = 4.25$

Gewöhnliche Differentialgleichungen (DGL)

(Serie 11)

Problemstellung: Eine **zeitabhängige Grösse *y(t)*** ist abhängig ihrer **zeitlichen Änderung *y'(t)***: ***y(t)* = *y'(t)* + t^2**
Bekannt ist nur ihre **Änderung** bei *t*: ***y'(t)* = *y(t)* − t^2**

$$\frac{dy}{dt} = f(t, y(t))$$

Gewöhnliche Differentialgleichung *n*-ter Ordnung

Eine DGL ist eine Gleichung für eine **gesuchte Funktionen** $y = y(x)$ deren Lösung *y* auf dem Interval $y : [a, b] \to \mathbb{R}$ definiert ist.

In der gesuchten Funktion treten mindestens eine ihrere Ableitungen $y'(x), \ldots, y^{(n)}$ auf.

Ordnung *n* = höchste auftetende Ableitung $y^{(n)}$

Explizit = wenn nach der höchsten vorkommenden Ableitung $y^{(n)}(x)$ aufgelöst werden kann (sonst implizit):

$$y^{(n)}(x) = f\left(x, y(x), y'(x), \ldots, y^{(n-1)}(x)\right) \qquad y''(x) = \underbrace{2 \cdot y'(x) - y(x) + x - 2}_{f(x, y(x), y'(x))}$$

Bsp. Implizit: Wenn nach 0 aufgelöst:

$$F(x, y(x), y'(x), y''(x), \ldots, y^{(n)}(x)) = 0 \qquad \underbrace{(y(x))^2 + (y'(x))^2 - 1}_{+ (y(x), y'(x))} = 0$$

Anfangswertprobleme (AWP)

Ein AWP besteht aus:

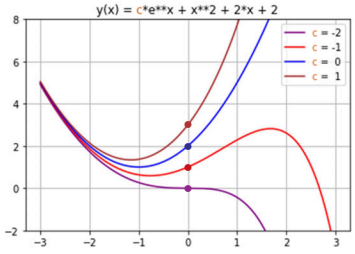
1. Einer **expliziten** DGL *n*-ter Ordnung sowie Auswertungen an Stelle x_0
2. Funktionswert von $y(x)$
3. Werte der ersten $n-1$ Ableitungen $y'(x), \ldots, y^{(n-1)}(x)$

$$\left.\begin{aligned} y(x_0) &= y_0 \\ y'(x_0) &= y_0' \\ y''(x_0) &= y_0'' \\ &\ldots \\ y^{(n-1)}(x_0) &= y_0^{(n-1)} \end{aligned}\right\} \text{ mit } x_0, y_0, y_0', y_0'', \ldots, y_0^{(n-1)} \text{ vorgegebene Zahlen}$$

Bsp:

Das AWP
$y'(x) = y(x) - x^2$
$y(0) = 1$

besitzt von den unendlich vielen Lösungen
$y(x) = c \cdot e^x + x^2 + 2x + 2$
der DGL diejenige, für welche gilt
$y(0) = c \cdot e^0 + 0^2 + 2 \cdot 0 + 2 = 1$,
$c + 2$



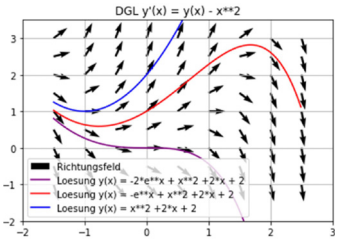
also diejenige für
$c = -1$,
das heisst
$y(x) = -e^x + x^2 + 2x + 2$.

Richtungsfeld (expliziter DGL 1. Ordnung)

Richtungsfeld = **bekannte Steigungen** der gesuchten Funktionen

Idee: Anhand eines ausgewählten Punktes (Anfangswert) im Richtungsfeld die gesuchte Funktion, die durch diesen Punkt geht, berechen.

Die rechte Seite $f(x, y(x))$ für jeden Punkt $(x, y(x))$ in der Ebene gibt an, wie gross die Steigung *y'* der Lösung $y(x)$ dort ist.

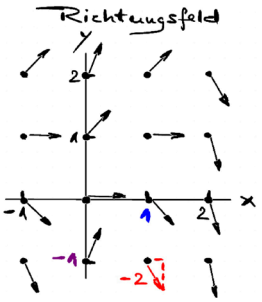


Bsp: $y'(x) = y(x) - x^2$ bzw. $y' = y - x^2$

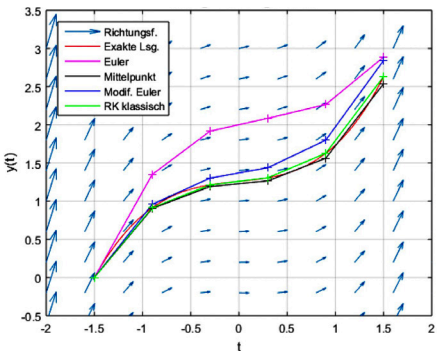
Steigungstabelle

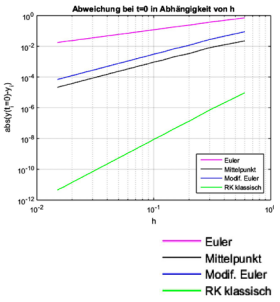
y \ x	-1	0	1	2
-1	-2	-1	-2	-5
0	-1	0	-1	-4
1	0	1	0	-3
2	1	2	1	-2

$y' = (-1) - 1^2$



Übersicht und Vergleich der Verfahren





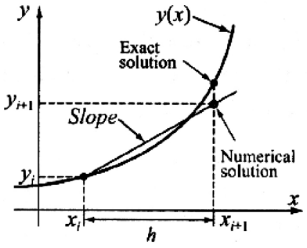
Euler
Mittelpunkt
Modif. Euler
RK klassisch

Klassisches Euler-Verfahren

(Auch Polygonzugmethode)

Numerische Approximation der Lösung y(x) des AWP für explizite DGL 1. Ordnung. Einsatz: Wenn eine DGL **weder separierbar noch linear** ist.

Idee: Wir folgen der Tangente um die Schrittweit *h*.



- Gegeben sei für $x \in [a,b]$ das Anfangswertproblem

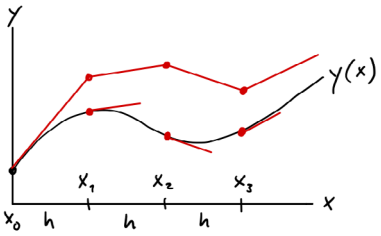
  $y' = f(x,y) \quad \text{mit } y(a) = y_0$

- Das Euler-Verfahren zur numerischen Lösung lautet

$$\begin{aligned} x_{i+1} &= x_i + h \\ y_{i+1} &= y_i + h \cdot f(x_i, y_i) \end{aligned}$$

wobei $x_0 = a \qquad x_i = a + ih \qquad i = 0, \ldots, n-1 \qquad h = \frac{b-a}{n}$

Wird *h* um Faktor 10 verkleinert, so verkleinert sich der absolute Fehler des Endwerts von *y* um Faktor 10. **Konvergenzordnung 1**



$y' = f(x, y(x)) = \sqrt{x + y(x)}$

$y(1) = 1, \quad h = 0.05$

$y_0 = 1$

$y_1 = y_0 + h \cdot f(x_0, y_0)$

$= 1 + 0.05 \cdot \sqrt{1+1} \approx 1.07$

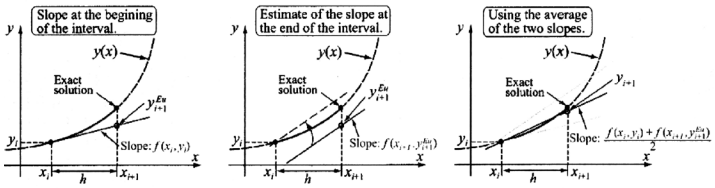
$y_2 = y_1 + h \cdot f(x_1, y_1)$

$= 1.07 + 0.05 \cdot \sqrt{1.05 + 1.07} \approx 1.1435$

...

Modifiziertes Euler Verfahren

Verwendet den Durchschnitt zweier Steigungen (mittlere Steigung).



(a) Führe das klassische Euler-Verfahren durch und speichere die erste Tangentensteigung in der Variable k_1:

$$\begin{aligned} x_{i+1} &= x_i + h \\ y_{i+1}^{Euler} &= y_i + h \cdot f(x_i, y_i) \\ k_1 &= f(x_i, y_i) \end{aligned}$$

(b) Berechne die zweite Tangentensteigung am Punkt $(x_{i+1}, y_{i+1}^{Euler})$ und speichere sie in der Variablen k_2:

$$k_2 = f(x_{i+1}, y_{i+1}^{Euler})$$

(c) Bilde den Durchschnitt der beiden Steigungen $(k_1 + k_2)/2$ und mache einen Schritt h ausgehend vom ursprünglichen Punkt (x_i, y_i) zur Berechnung der Näherung (x_{i+1}, y_{i+1}):

$$y_{i+1} = y_i + h \cdot \frac{(k_1 + k_2)}{2}$$

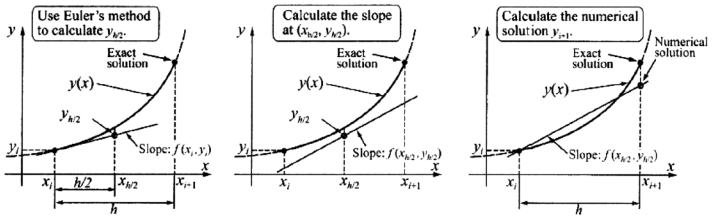
(d) Wiederhole diese Schritte ausgehend von

$x_0 = a \qquad x_i = a + ih \qquad i = 0, \ldots, n-1 \qquad h = \frac{b-a}{n}$

Wird *h* um Faktor 10 verkleinert, so verkleinert sich der absolute Fehler des Endwerts von *y* um Faktor 100 = 10^2. **Konvergenzordnung 2**

Mittelpunkt-Verfahren

Folgt der Tangente nur die halbe Schrittweite und berechnet beim Mittelpunkt die neue Steigung.



- Gegeben sei für $x \in [a,b]$ das Anfangswertproblem

  $y' = f(x,y) \quad \text{mit } y(a) = y_0$

- Das Mittelpunkt-Verfahren zur numerischen Lösung lautet

$$\begin{aligned} x_{h/2} &= x_i + \frac{h}{2} \\ y_{h/2} &= y_i + \frac{h}{2} \cdot f(x_i, y_i) \\ x_{i+1} &= x_i + h \\ y_{i+1} &= y_i + h \cdot f(x_{h/2}, y_{h/2}) \end{aligned}$$

wobei $x_0 = a \qquad x_i = a + ih \qquad i = 0, \ldots, n-1 \qquad h = \frac{b-a}{n}$

1. Für h bis 10^{-4} gilt: Wird *h* um Faktor 10 verkleinert, so verkleinert sich der Fehler um Faktor 100 = 10^2. **Konvergenzordnung 2**
2. Für *h* ab 10^{-5} nimmt der Fehler wieder zu. Grund dafür sind Rundungsfehler.

Beispiel von Hand: Serie 11, Aufgabe 2

Fehlerordnung eines Verfahrens

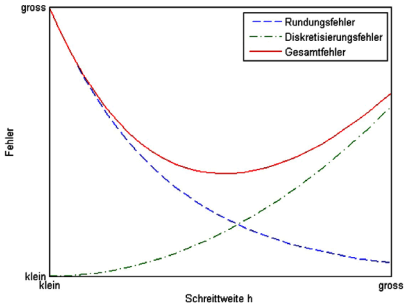
Gewöhnliche Differentialgleichungen (DGL) (Serie 12)

Rundungsfehler und Diskretierungsfehler

Rundungsfehler
= Maschinenfehler
Aufgrund endlicher Rechnerarithmetik

Diskretierungsfehler
= Verfahrensfehler
Aufgrund endlicher Schrittweite

Gesamtfehler
Rundungsfehler
+ Diskretierungsfehler



Konvergenzordnung *p*

Verwendbar sind nur Verfahren mit der Konvergenzordnung $p \geq 1$,
da dann der **globale Fehler (Diskretierungsfehler)** gegen **Null** strebt:

$$\lim_{h \to 0} \mid y(x_n) - y_n \mid \leq \lim_{h \to 0} C \cdot h^p = 0$$

Wird bei Konsistenzordnung *p* die Schrittweite *h* um Faktor 10 verkleinert, so verkleinert sich der globaler Fehler um Faktor 10^p:

Lokaler Fehler:
Differenz zwischen dem **exakten Wert** $y(x_{i+1})$
und dem Näherungwert y_{i+1} **nach einer Iteration.**

Ein numerisches Verfahren hat die Konsistenzordnung *p* falls gilt:

$$\mid y(x_{i+1}) - y_{i+1} \mid \leq C \cdot h^{p+1}$$

Globaler Fehler (Diskretierungsfehler):
Differenz zwischen dem **exakten Wert** $y(x_n)$ und dem Näherungwert y_n (letzter Iterationsschritt) für **nach *n* Iterationen.**

Ein numerisches Verfahren hat die Konsistenzordnung *p* falls gilt:

$$\mid y(x_n) - y_n \mid \leq C \cdot h^p$$

Runge-Kutta-Verfahren

Gewöhnliche Differentialgleichungen (DGL) (Serie 12)

Klassisches vierstufiges Runge-Kutta-Verfahren

Basierend auf dem gewichteten Mittelwert aus den Steigungen k_1, k_2, k_3 und k_4.
Standardverfahren für «gutartige» Anfangswertprobleme (AWP).

$$\begin{aligned} k_1 &= f(x_i,\, y_i) \\ k_2 &= f(x_i + \frac{h}{2},\, y_i + \frac{h}{2} k_1) \\ k_3 &= f(x_i + \frac{h}{2},\, y_i + \frac{h}{2} k_2) \\ k_4 &= f(x_i + h,\, y_i + h k_3) \\ k &= b_1 k_1 + b_2 k_2 + b_3 k_3 + b_4 k_4 \\ x_{i+1} &= x_i + h \\ y_{i+1} &= y_i + h \cdot k \end{aligned}$$

wobei $x_0 = a \quad x_i = a + ih \quad i = 0, ..., n-1$

$$y(x_0) = y_0 \quad h = \frac{b-a}{n} \quad b = h \cdot n + a$$

1. Für h bis 10^{-4} gilt: Wird *h* um Faktor 10 verkleinert, so verkleinert sich der Fehler um Faktor 10'000 = 10^4. **Konvergenzordnung 4**
2. Für *h* ab 10^{-5} nimmt der Fehler wieder zu. Grund dafür sind Rundungsfehler.

Ein vierstufiges RK-Verfahren hat **höchstens** Konvegenzordnung 4.

Beispiel siehe Seite 7.

Allgemeines s-stufiges Runge-Kutta-Verfahren

$$\begin{aligned} k_n &= f\left(x_i + c_n h,\, y_i + h \sum_{m=1}^{n-1} a_{nm} k_m\right) \quad \text{für } n = 1, ..., s \\ y_{i+1} &= y_i + h \sum_{n=1}^{s} b_n k_n \end{aligned}$$

a_{nm}, b_n, c_n sind Konstanten. Die Konsistenz- und Konvergenzordnung hängt von der Wahl dieser Konstanten ab.

Die Koeffizienten werden in der *Butcher-Tableau* angeordnet:

$$\begin{array}{c|cccc} c_1 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & & & & \\ c_n & a_{n1} & a_{n2} & \dots & a_{n,n-1} \\ \vdots & & & & \\ c_s & a_{s1} & a_{s2} & \dots & a_{s,s-1} \\ \hline & b_1 & b_2 & \dots & b_{s-1} \quad b_s \end{array}$$

mit $b_1 + b_2 + ... + b_s = 1$

Alle bisher vorgestellten Verfahren entsprechen diesem Schema:

– Euler-Verfahren, $s = 1$: $\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$

– Mittelpunkt-Verfahren, $s = 2$: $\begin{array}{c|cc} 0 & & \\ 0.5 & 0.5 & \\ \hline & 0 & 1 \end{array}$

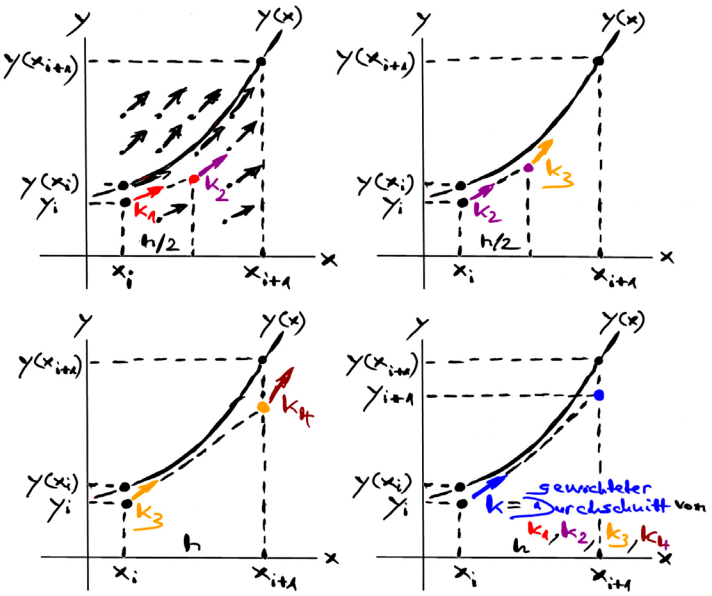
– Modifiziertes Euler-Verfahren, $s = 2$: $\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 0.5 & 0.5 \end{array}$

– klass. Runge-Kutta Verfahren, $s = 4$: $\begin{array}{c|ccc} 0 & & & \\ 0.5 & 0.5 & & \\ 0.5 & 0 & 0.5 & \\ 1 & 0 & 0 & 1 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} \quad \frac{1}{6} \end{array}$

Hat **höchstens** (je nach Wahl der Konstanten) **Konvergenzordnung *s***.
(s = Stufen)

Allgemeines klassisches vierstufiges Runge-Kutta-Verfahren:

$k_1 = f(x_i + c_1 \cdot h,\, y_i)$
$k_2 = f(x_i + c_2 \cdot h,\, y_i + h \cdot c_{21} \cdot k_1)$
$k_3 = f(x_i + c_3 \cdot h,\, y_i + h \cdot (c_{31} \cdot k_1 + a_{32} \cdot k_2))$
$k_4 = f(x_i + c_4 \cdot h,\, y_i + h \cdot (c_{41} \cdot k_1 + a_{42} \cdot k_2 + c_{43} \cdot k_3))$
$k = b_1 \cdot k_1 + b_2 \cdot k_2 + b_3 \cdot k_3 + b_4 \cdot k_4$ mit $b_1 + b_2 + b_3 + b_4 = 1$
$x_{i+1} = x_i + h$
$y_{i+1} = y_i + h \cdot k$



Systeme von Differentialgleichungen

Aus eine Differentialgleichung k-ter Ordnung kann ein System von k Differentialgleichungen 1. Ordnung gemacht werden. Diese können mit den bekannten Verfahren gelöst werden. (Serie 13)

Eine DGL k-ter Ordnung auf k DGL 1. Ordnung zurückführen

Wir betrachten die Differentialgleichung 3. Ordnung

$$y''' + 5y'' + 8y' + 6y = 10e^{-x}$$

mit der Anfangsbedingung $y(0) = 2,\ y'(0) = y''(0) = 0$

– 1. Schritt: wir lösen nach der höchsten Ableitung auf:

$$y''' = 10e^{-x} - 5y'' - 8y' - 6y$$

– 2. Schritt: wir führen Hilfsfunktionen $z_1,\ z_2,\ z_3$ bis zur zweiten Ableitung ein:

$$\begin{aligned} z_1(x) &= y(x) \quad \leftarrow \text{Gesuchte Funktion} \\ z_2(x) &= y'(x) \\ z_3(x) &= y''(x) \end{aligned}$$

– 3. Schritt: wir leiten die Hilfsfunktionen ab und setzen sie in $z_3' = y'''$ ein

$$\begin{aligned} z_1' &= y' \ (= z_2) \\ z_2' &= y'' \ (= z_3) \\ z_3' &= y''' \\ &= 10e^{-x} - 5y'' - 8y' - 6y \\ &= 10e^{-x} - 5z_3 - 8z_2 - 6z_1 \end{aligned}$$

– 4. Schritt: wir schreiben die DGL in vektorieller Form

$$\boldsymbol{z}(0) = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$$

und

$$\boldsymbol{z}' = \begin{pmatrix} z_1' \\ z_2' \\ z_3' \end{pmatrix} = \begin{pmatrix} z_2 \\ z_3 \\ 10e^{-x} - 5z_3 - 8z_2 - 6z_1 \end{pmatrix} = \boldsymbol{f}(x, z)$$

$$y''' = 10e^{-x} - 5y'' - 8y' - 6y$$

direkt ↓

$$z' = f(x, y) = \begin{pmatrix} y' \\ y'' \\ 10e^{-x} - 5y'' - 8y' - 6y \end{pmatrix}$$

und erhalten so ein Anfangswertproblem 1. Ordnung

$$\boldsymbol{z}' = \boldsymbol{f}(x, \boldsymbol{z}) \quad \text{mit} \quad \boldsymbol{z}(0) = \begin{pmatrix} y(0) \\ y'(0) \\ y''(0) \end{pmatrix}$$

Bemerkung: da es sich um eine lineare DGL 3. Ordnung gehandelt hat, lässt sich $\mathbf{z}'$ in diesem Beispiel als lineares Gleichungssystem schreiben

$$\boldsymbol{z}' = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{b}$$

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -8 & -5 \end{pmatrix},\ \boldsymbol{b} = \begin{pmatrix} 0 \\ 0 \\ 10e^{-x} \end{pmatrix}$$

Lösen eines Systems von k DGL 1. Ordnung

$$\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$$

Gegeben:

k explizite DGL 1. Ordnung

$$\begin{aligned} y_1'(x) &= f_1(x, y_1(x), \ldots, y_k(x)) \\ y_2'(x) &= f_2(x, y_1(x), \ldots, y_k(x)) \\ &\vdots \\ y_k'(x) &= f_k(x, y_1(x), \ldots, y_k(x)) \end{aligned}$$

mit k Anfangswerte

$$\boldsymbol{y}(x_0) = \boldsymbol{y}^{(0)} = \begin{pmatrix} y_1(x_0) \\ y_2(x_0) \\ \vdots \\ y_k(x_0) \end{pmatrix}$$

In Vektorform:

$$\boldsymbol{y}' = \begin{pmatrix} y_1'(x) \\ y_2'(x) \\ \vdots \\ y_n'(x) \end{pmatrix} = \boldsymbol{f}(x, \boldsymbol{y}(x)) = \begin{pmatrix} f_1(x, \boldsymbol{y}(x)) \\ f_2(x, \boldsymbol{y}(x)) \\ \vdots \\ f_k(x, \boldsymbol{y}(x)) \end{pmatrix}$$

Gesucht: k Funktionen $y_1(x), \ldots, y_k(x)$

$$\boldsymbol{y} = \boldsymbol{y}(x) = \begin{pmatrix} y_1(x) \\ y_2(x) \\ \vdots \\ y_k(x) \end{pmatrix}$$

Ein Lösungsverfahren für die eindimensionale Gleichung

$$\begin{aligned} x_{i+1} &= x_i + h \\ y_{i+1} &= y_i + \text{Steigung} \cdot h \end{aligned} \qquad y'(x) = f(x, y(x)),\ y(x_0) = y_0$$

kann für ein System analog erweitert werden

$$\begin{aligned} x_{i+1} &= x_i + h \\ \boldsymbol{y}^{(i+1)} &= \boldsymbol{y}^{(i)} + \text{Steigung} \cdot h \\ \boldsymbol{z}^{(i+1)} &= \boldsymbol{z}^{(i)} + \boldsymbol{f}(x_i, \mathbf{z}^{(i)}) \cdot h \end{aligned} \qquad \boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y}(x)),\ \boldsymbol{y}(x_0) = \boldsymbol{y}^{(0)}$$

$\boldsymbol{y}^{(i)}$ Vektor nach der i-ten Iteration.

AWP: $y'''(x) - (y'(x))^2 + x \cdot y(x) = 0,\ y(0) = 2,\ y'(0) = 1,\ y''(0) = 0$

a) AWP als System von expliziten DGLs 1. Ordnung:

$y''' = (y')^2 - x \cdot y \quad + (0 \cdot y'')$

$z_4 = z_2^2 - x \cdot z_1 \quad + (0 \cdot z_3)$

$z' = \begin{pmatrix} z_2 \\ z_3 \\ z_2^2 - x \cdot z_1 \end{pmatrix},\ z(0) = y_0 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} y(0) \\ y'(0) \\ y''(0) \end{pmatrix}$

b) Erster Schritt mit Mittelpunktverfahren:

$h = 2,\ x_0 = 0,\ y_0 = z_0 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} y(0) \\ y'(0) \\ y''(0) \end{pmatrix}$

- $x_{1/2} = 0 + \frac{2}{2} = 1$
- $y_{1/2} = y_0 + \frac{h}{2} \cdot f(x_0, y_0) = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} + \frac{2}{2} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$

$f(x_0, y_0) = z' = \begin{pmatrix} z_2 \\ z_3 \\ z_2^2 - x \cdot z_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1^2 - 0 \cdot 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$

- $x_1 = x_0 + h = 0 + 2 = 2$
- $y_1 = y_0 + h \cdot f(x_{1/2}, y_{1/2}) = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} 1 \\ 1 \\ -2 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ -4 \end{pmatrix} = z^{(1)}$

$f(x_{1/2}, y_{1/2}) = \begin{pmatrix} 1 \\ 1 \\ 1^2 - 1 \cdot 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ -2 \end{pmatrix}$

Absoluter Fehler (2. Norm)

$$\|\mathbf{y}^{(i)} - \mathbf{y}(x_0)\|_2$$

$\mathbf{y}^{(i)}$ = Approximation nach i-ten Iterationsschritt, $\mathbf{y}(x_0)$ = exakter Wert