Nichtlineare Gleichungssysteme

Funktionen mit mehreren Variablen

auch multivariat gennant, hier nur skalarwertige Funktionen (keine Komplexen Zahlen).
Definition:

Explizite Darstellung Funktionsgleichung ist nach einer variablen aufgelöst.

$$y = f(x_1, x_2, ..., x_n)$$

Implizite Darstellung nicht nach einer Variablen aufgelöst(nur n-1 unabhängige Variablen)

$$F(x_1, x_2, ..., x_n) = 0$$

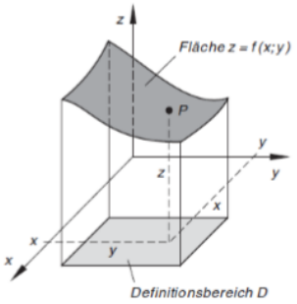
für vektorwertige funktionen

$$\vec{f}(x_1, ..., x_n) = \begin{pmatrix} y_1 = f_1(x_1, ..., x_n) \\ \dots \\ y_m = f_m(x_1, ..., x_n) \end{pmatrix}$$

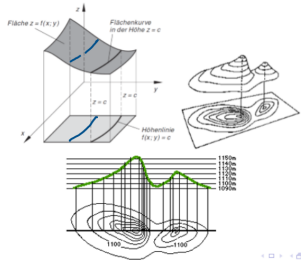
Graphische Darstellungsformen

Funktionen mit 2 Variablen können 3D dargestellt werden.
Interpretieren als $z = f(x, y)$

Fläche Punkte $(x, y, f(x, y))$

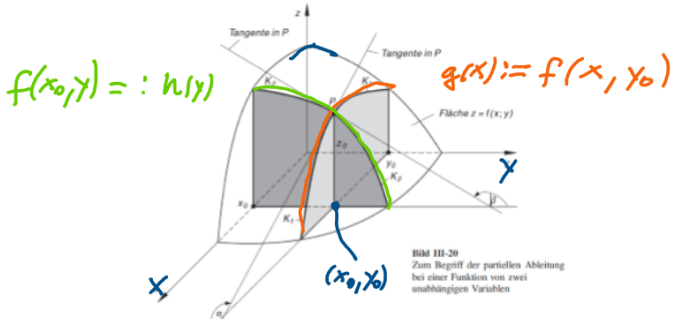


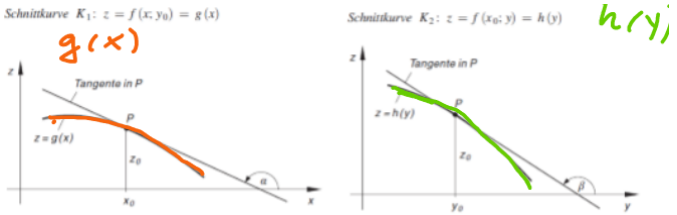
Schnittkurve bei konstanter Höhe z, auch Höhen-/Contour-Plot



Partielle Ableitungen

Nur eine der Variablen wird abgeleitet, der Rest als Konstante behandelt. Visuell entspricht dies der Steigung an einer Flächentangente.





Partielle Ableitungen 1. Ordnung

Beispiel nach x abgeleitet(normale Ableitungsregeln, andere Variablen als Konstanten betrachten):

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x}$$

$$f(x, y) = 3xy^3 + 10x^2y + 5y + 3y * sin(5xy)$$
$$\frac{\partial f}{\partial x} = 3 * 1 * y^3 + 10 * 2x + 0 + 3y * cos(5xy) * 5 * 1 * y$$

Linearisierung

Repetition Tangentengleichung

Dient als Annäherung für eindimensionale $f(x)$ in der Nähe von x_0 (Linearisierung): $g(x) = f(x_0) + f'(x_0)(x - x_0)$

Jacobi-Matrix

Sozusagen wie Tangentengleichung aber für mehrere Variablen

$$\vec{y} = \vec{f}(\vec{x}) = \begin{pmatrix} y_1 = f_1(x_1, ..., x_n) \\ ... \\ y_m = f_m(x_1, ..., x_n) \end{pmatrix}$$

Die Jacobi-Matrix enthält sämtliche Partielle Ableitungen 1. Ordnung von $\vec{f}$.
Auf jeder Spalte bleibt die funktion f_j die gleiche und in den Zeilen $x_i \rightarrow \frac{\partial f_j}{\partial x_i}$

$$D\vec{f}(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x) & \frac{\partial f_m}{\partial x_2}(x) & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

verallgemeinerte Tangentengleichung

$$\vec{g}(\vec{x}) = \vec{f}(\vec{x}^{(0)}) + D\vec{f}(\vec{x}^{(0)}) * (\vec{x} - \vec{x}^{(0)})$$

ist eine lineare Funktion und für $\vec{x}$ in der Umgebung von $\vec{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})$ gilt $\vec{f}(\vec{x}) \approx \vec{g}(\vec{x})$

Df entspricht der obigen Funktion zur Erzeugung einer Jacobi-Matrix.

Hochgestellte Zahlen in Klammern $(x^{(n)})$ stehen wie zuvor für eine Variable nach n Iterationsschritten.

Tangentialebene

- Für den speziellen Fall $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$ mit $y = f(x_1, x_2)$ und $\boldsymbol{x}^{(0)} = (x_1^{(0)}, x_2^{(0)})^T \in \mathbb{R}^2$ ist die Jacobi-Matrix nur ein Zeilenvektor mit zwei Elementen, nämlich

$$\boldsymbol{Df}(\boldsymbol{x}^{(0)}) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x_1^{(0)}, x_2^{(0)}) & \frac{\partial f}{\partial x_2}(x_1^{(0)}, x_2^{(0)}) \end{pmatrix}.$$

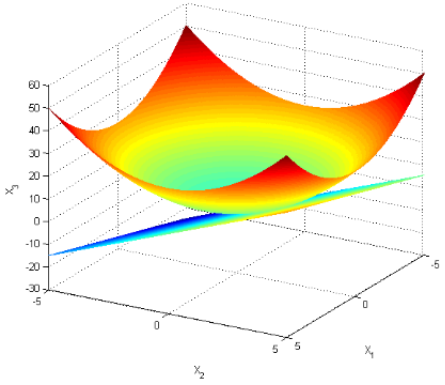
  Dann liefert die Linearisierung

$$\begin{aligned} g(x_1, x_2) &= f(x_1^{(0)}, x_2^{(0)}) + \begin{pmatrix} \frac{\partial f}{\partial x_1}(x_1^{(0)}, x_2^{(0)}) & \frac{\partial f}{\partial x_2}(x_1^{(0)}, x_2^{(0)}) \end{pmatrix} \cdot \begin{pmatrix} x_1 - x_1^{(0)} \\ x_2 - x_2^{(0)} \end{pmatrix} \\ &= f(x_1^{(0)}, x_2^{(0)}) + \frac{\partial f}{\partial x_1}(x_1^{(0)}, x_2^{(0)}) \cdot \begin{pmatrix} x_1 - x_1^{(0)} \end{pmatrix} + \frac{\partial f}{\partial x_2}(x_1^{(0)}, x_2^{(0)}) \cdot \begin{pmatrix} x_2 - x_2^{(0)} \end{pmatrix} \end{aligned}$$

  die Gleichung der **Tangentialebene**.

- Sie enthält sämtliche im Flächenpunkt $P = (x_1^{(0)}, x_2^{(0)}, f(x_1^{(0)}, x_2^{(0)}))$ an die Bildfläche von $y = f(x_1, x_2)$ angelegten Tangenten.

Graphische Darstellung der Fläche $x_3 = f(x_1, x_2) = x_1^2 + x_2^2$ sowie Tangentialebene durch den Flächenpunkt $(x_1^{(0)} = 1, x_2^{(0)} = 2, f(x^{(0)}) = 5)$



Nullstellenbestimmung für nichtlineare Systeme

- Gegeben: $n \in \mathbb{N}$ und eine Funktion $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$
- Gesucht: $\vec{x} \in \mathbb{R}^n \quad \text{mit} \quad \vec{f}(\vec{x}) = \vec{0}$

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

Newton-Verfahren für Systeme

Herleitung

Repetition 1-Dimensional: (nur für $f : \mathbb{R} \rightarrow \mathbb{R}$)

Aus der Linearisierung der Funktion f mittels der Tangente g an der Selle x_n

$$f(x) \approx g(x) = f(x_n) + f'(x_n)(x - x_n)$$

folgte die Iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (n = 0, 1, 2, 3, ...).$$

Mit der Jacobi-Matrix $Df(x)$ kann das analog für Vektor-wertige Funktionen $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ angewendet werden.

$$\vec{x}^{(n+1)} = \vec{x}^{(n)} - (Df(\vec{x}^{(n)}))^{-1} * \vec{f}(\vec{x}^{(n)})$$

Das Inverse der Jacobi-Matrix wird aber nie berechnet sondern die obige Gleichung via Substitution als lineares Gleichungsystem aufgefasst.

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

als lineares Gleichungssystem auffasst gemäss

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

und so δ^n bestimmen und anschliessend

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

Quadratisch-konvergentes Newton-Verfahren für Systeme

Gesucht: Nullstellen von $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ mit Startvektor $\vec{x}^{(0)}$ Nahe der Nullstelle.
es kann passieren, dass das Newton-Verfahren Statt einer Nullstelle ein lokales Minimum $x_{min}! = 0$ findet, in diesem Fall ist $Df(x_{min})$ aber immer nicht regulär.

für $n = 0, 1, ...$:

1. Berechne $\delta^{(n)}$ als Lösung des linearen Gleichungsystems

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

2. Setze

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

mögliche Abbruchkriterien

- $n \geq n_{max},\ n_{max} \in \mathbb{N}$
- $||x^{(n+1)} - x^{(n)}|| \leq \epsilon \qquad \leftrightarrow \qquad ||\delta^{(n)}|| \leq \epsilon$
- $||x^{(n+1)} - x^{(n)}|| \leq \epsilon * ||x^{(n+1)}||$
- $||\vec{f}(x^{(n+1)})|| \leq \epsilon$

Konvergenz

Das Newton-Verfahren konvergiert quadratisch für nahe genug an einer Nullstelle $\vec{x}$ liegende Startvektoren, wenn $D\vec{f}(\vec{x})$ regulär und $\vec{f}$ dreimal stetig differenzierbar ist.

Vereinfachtes Newton-Verfahren für Systeme

Gesucht: Nullstellen von $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ mit Startvektor $\vec{x}^{(0)}$ Nahe der Nullstelle.

Der Unterschied zum Quadratisch-konvergenten Newton-Verfahren liegt darin, dass nur einmal die Jacobi-Matrix für den Startvektor berechnet werden muss (rot) → weniger Aufwand aber konvergiert langsamer (linear).

für $n = 0, 1, ...$:

1. Berechne $\delta^{(n)}$ als Lösung des linearen Gleichungsystems

$$D\vec{f}(\vec{x}^{(0)})\delta^{(n)} = -\vec{f}(\vec{x}^{(n)})$$

2. Setze

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

Gedämpftes Newton-Verfahren

Die Idee ist bei jedem Iterationsschritt zu überprüfen ob es sich um eine Verbesserung handelt und falls nicht, es stattdessen mit einem gedämpften Schritt zu versuchen.

Die gewählte Dämpfungsgrösse k_{max} ist stark vom Problem abhängig. $k_{max} = 4$ ist ein vernünftiger Standard Wert.

für $n = 0, 1, ...$:

1. Berechne $\delta^{(n)}$ als Lösung des linearen Gleichungsystems

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

2. Finde das minimale $k \in 0, 1, ..., k_{max}$ mit

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

3. Setze (falls kein minimales $k \to k = 0$)

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

Ausgleichsrechnung / Interpolation

Zur Auswertung Datenpunkte mit Streuung durch eine Funktion annähern.

Interpolation eine Funktion die exakt durch die Messpunkte geht. Geeignet falls:
- wenig Datenpunkte
- (fast) keine Messfehler

Ausgleichsrechnung eine Funktion die summiert die kleinste Abweichung von den Messpunkten hat. Zwischen den Messpunkten oftmals stabiler als Interpolation
- typischerweise viele Datenpunkte
- fehlerbehaftet

Interpolation

- Gegeben: $n+1$ Wertepaare (x_i, y_i), $i = 0, ..., n$ mit $x_i \neq x_j \mid i \neq j$
- Gesucht: stetige Funktion $g(x)$ mit $g(x_i) = y_i \; \forall i = 0, 1, ..., n$
  (was zwischen diesen Punkten für Resultate herauskommen kann stark variieren)

Polynominterpolation

Zu den $n+1$ Stützpunkten ist ein Polynom $P_n(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$ vom Grad n gesucht.

Weil das Polynom vom Grad n ist lässt sich zusammen mit den Stützpunkten ein lineares Gleichungsystem dazu aufstellen.

$$\begin{aligned} a_0 + a_1 x_0 + a_2 x_0^2 + ... + a_n x_0^n &= y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + ... + a_n x_1^n &= y_1 \\ &\vdots \\ a_0 + a_1 x_n + a_2 x_n^2 + ... + a_n x_n^n &= y_n \end{aligned}$$

bzw.

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

Lagrange Interpolationsformel

Durch $n+1$ Stützpunkte mit verschiedenen Stützstellen gibt es genau EIN Polynom $P_n(x)$ vom Grade $\leq n$ welches alle Stützpunkte interpoliert.

Lagrangeform für $P_n(x)$:

$$P_n(x) = \sum_{i=0}^{n} l_i(x) y_i$$

Die Lagrangepolynome vom Grad n ($l_i(x)$) sind definiert durch:

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \qquad i = 0, 1, ..., n$$

Beispiel:

t	08.00	10.00	12.00	14.00	Uhr
T	11.2	13.4	15.3	19.5	°C
	y_0	y_1	y_2	y_3	

Wir haben $n+1 = 4$ Stützpunkte, also ist $n = 3$ und das Interpolationspolynom hat die Form

$$P_n(x) = \sum_{i=0}^{3} l_i(x) y_i = 11.2 \cdot l_0(x) + 13.4 \cdot l_1(x) + 15.3 \cdot l_2(x) + 19.5 \cdot l_3(x).$$

Die Lagrangepolynome berechnen sich zu

$$\begin{aligned} l_0(x) &= \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} = \frac{(x-10)(x-12)(x-14)}{(-2)(-4)(-6)} = -\frac{1}{48}x^3 + \frac{3}{4}x^2 - \frac{107}{12}x + 35 \\ l_1(x) &= \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} = \frac{(x-8)(x-12)(x-14)}{(2)(-2)(-4)} = +\frac{1}{16}x^3 - \frac{17}{8}x^2 + \frac{47}{2}x - 84 \\ l_2(x) &= \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} = \frac{(x-8)(x-10)(x-14)}{(4)(2)(-2)} = -\frac{1}{16}x^3 + 2x^2 - \frac{83}{4}x + 70 \\ l_3(x) &= \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} = \frac{(x-8)(x-10)(x-12)}{(6)(4)(2)} = +\frac{1}{48}x^3 - \frac{5}{8}x^2 + \frac{37}{6}x - 20 \end{aligned}$$

Fehlerabschätzung

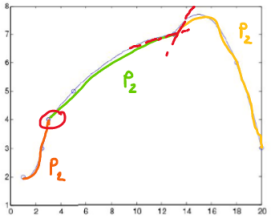
Rein theoretisch weil man die tatsächliche Funktion f kennen müsste.
Gegeben $y_i = f(x_i)$ und f genügend oft stetig differenzierbar:

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

Spline-Interpolation

Kontext (hoffentlich nicht Prüfungsrelevant)

Die Annhäherung durch ein einziges Polynom ist zwischen den Messpunkten oft hoch instabil. Als Alternative kann stattdessen stückweise interpoliert werden.



- hier stören die Knicke(Ableitung verschieden von beiden Seiten) an den Übergängen
- die Spline-Interpolation versucht durch Polynome niederen Grades zu interpolieren und damit Schwingungen unterdrücken → keine Knicke
- Polynome müssen dazu an den Messpunkten nicht nur selben Funktionswert sondern auch selbe Ableitung haben (1. und 2. Ableitung).

Man kann so für jedes Interval $[x_i, x_{i+1}]$, $i = 0, 1, 2, ..., n-1$ genau ein Polynom s_i ansetzen (i Bezeichnet die Nummer des Intervalls).

Dieses Polynom muss folgende Bedingungen erfüllen:

Interpolation $s_i(x_i) = y_i \quad , \quad s_{i+1}(x_{i+1}) = y_{i+1}$

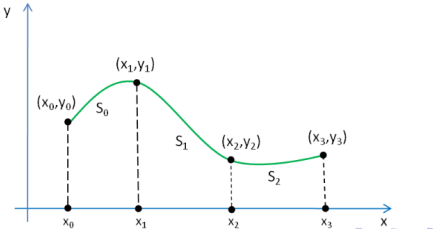
stetiger Übergang $s_i(x_{i+1}) = s_{i+1}(x_{i+1}) \quad , \quad s_{i+1}(x_{i+2}) = s_{i+2}(x_{i+2})$

keine Knicke $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \quad , \quad s_{i+1}'(x_{i+2}) = s_{i+2}'(x_{i+2})$

gleiche Krümmung $s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}) \quad , \quad s_{i+1}''(x_{i+2}) = s_{i+2}''(x_{i+2})$

Zusatzbedingungen Damit es genug Gleichungen für ein reguläres System sind müssen noch Zusatzbediungen gesetzt werden, diese unterscheiden sich je nach Art des gewählten Splines

Beispiel (6.5 in Folien):
Gegeben sind die vier Stützpunkte (x_i, y_i) für $i = 0, ..., 3$. Wir suchen nun die Splinefunktion S, die durch diese vier Punkte geht und sich zusammensetzt aus den drei kubischen Polynomen S_0, S_1 und S_2.



Mit dem Ansatz

$$\begin{aligned} S_0 &= a_0 + b_0(x-x_0) + c_0(x-x_0)^2 + d_0(x-x_0)^3, \; x \in [x_0, x_1] \\ S_1 &= a_1 + b_1(x-x_1) + c_1(x-x_1)^2 + d_1(x-x_1)^3, \; x \in [x_1, x_2] \\ S_2 &= a_2 + b_2(x-x_2) + c_2(x-x_2)^2 + d_2(x-x_2)^3, \; x \in [x_2, x_3] \end{aligned}$$

müssen wir $3 \cdot 4 = 12$ Koeffizienten berechnen, wofür wir 12 Bedingungen benötigen.

Diese lauten:
1. Interpolation der Stützpunkte:

$$\begin{aligned} S_0(x_0) &= y_0 \\ S_1(x_1) &= y_1 \\ S_2(x_2) &= y_2 \\ S_2(x_3) &= y_3 \end{aligned}$$

2. Stetiger Übergang an den Stellen x_1 und x_2:

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

3. Erste Ableitung an den Übergangsstellen muss übereinstimmen (keine Knicke):

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

4. Zweite Ableitung an den Übergangsstellen muss übereinstimmen (gleiche Krümmung):

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

Jetzt haben wir allerdings erst 10 Bedingungen für die 12 Koeffizienten. Das heisst, wir brauchen noch zwei zusätzliche. Diese können, in Abhängigkeit der Problemstellung, 'frei' gewählt werden und beziehen sich häufig auf die beiden Randstellen, hier x_0 und x_3.

Man unterscheidet zum Beispiel:

- die *natürliche kubische Splinefunktion*:

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

mit einem möglichen Wendepunkt im Anfangs- und Endpunkt.

- die *periodische kubische Splinefunktion*

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

wenn man die Periode $p = x_3 - x_0$ hat und damit $y_0 = y_3$ bzw. $S_0(x_0) = S_2(x_3)$ gilt.

- die kubische Splinefunktion mit *not-a-knot Bedingung*:

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

natürliche kubische Splinefunktion

Gegeben $n+1$ Stützpunkte (x_i, y_i) mit monoton aufsteigenden Stützstellen $x_0 < x_1 < ... < x_n \quad | \quad \text{und } n \geq 2$

Gesucht natürliche kubische Splinefunktion $S(x)$, in jedem Intervall $[x_i, x_{i+1}]$, $i = 0, 1, ..., n-1$ definiert durch das Polynom

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

Als Gleichungsystem $Ac = z$ (nur für c_i, entspricht 4. aus Algorithmus)

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

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

Beispiel als Tabelle für manuelle Berechnung:

i	0	1	2	3
x_i	0	1	2	3
y_i	2	1	2	2
a_i	2	1	2	-
h_i	1	1	1	-
c_i	0	?	?	0

Kompletter Algorithmus

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

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
   1. $i = 1$:
      $2(h_0 + h_1)c_1 + h_1 c_2 = 3\frac{y_2-y_1}{h_1} - 3\frac{y_1-y_0}{h_0}$
   2. falls $n \geq 4$ gilt für $i = 2, ..., n-2$:
      $h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} = 3\frac{y_{i+1}-y_i}{h_i} - 3\frac{y_i-y_{i-1}}{h_{i-1}}$
   3. $i = n-1$:
      $h_{n-2}c_{n-2} + 2(h_{n-2} + h_{n-1})c_{n-1} = 3\frac{y_n-y_{n-1}}{h_{n-1}} - 3\frac{y_{n-1}-y_{n-2}}{h_{n-2}}$
5. $b_i = \frac{y_{i+1}-y_i}{h_i} - \frac{h_i}{3}(c_{i+1} + 2c_i)$
6. $d_i = \frac{1}{3h_i}(c_{i+1} - c_i)$

Ausgleichsrechnung

Unterschied zur Interpolation: Funktion soll nicht exakt durch alle Messpunkte gehen sondern möglichst gut approximieren → stabileres Verhalten abseits der Messpunkte. Besonders sinnvoll bei vielen Daten (meist zusätzlich fehlerbehaftet)

Ausgleichsproblem

Gegeben Wertepaare (x_i, y_i) $i = 1, ..., n$ mit $x_i \neq x_j \mid i \neq j$

Gesucht stetige Funktion $f : \mathbb{R} \to \mathbb{R}$ die bestmöglich Annähert. Heisst $f(x_i) \approx y_i$

Ansatzfunktionen/Ausgleichsfunktion/Fehlerfunktional/ kleinste Fehlerquadrate

Gegeben:

- Menge F von stetigen Ansatzfunktionen f
- n Wertepaare (x_i, y_i)

Eine Funktion $f \in F$ heisst Ausgleichsfunktion von F zu den Wertepaaren, falls das Fehlerfunktional E minimal wird.
Das f mit dem minimum nennt man optimal im Sinne der kleinsten Fehlerquadrate

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

lineare Ausgleichsrechnung

Gesuchte Funktion f ist Linearkombination vom m Basisfunktionen f_i, $i = 1, 2, ..., m \mid m \leq n$ und den gesuchten Parametern λ_i

Wenn $m \leq n$ ist das Gleichungsystem überbestimmt und es existiert keine Lösung $E(f) = 0$, ist $m = n$ gibt es diese Lösung jedoch und wir sind wieder beim Spezialfall der Interpolation

$$f(x) = \lambda_1 f_1(x) + ... + \lambda_m f_m(x)$$

Lineares Ausgleichsproblem

- Gegeben seien n Wertepaare $(x_i, y_i), i = 1, ..., n$, und m Basisfunktionen $f_1, ..., f_m$ auf einem Intervall $[a, b]$. Wir wählen F als die Menge der Ansatzfunktionen $f := \lambda_1 f_1 + ... + \lambda_m f_m$ mit $\lambda_j \in \mathbb{R}$, also $F = \{f = \lambda_1 f_1 + ... + \lambda_m f_m \mid \lambda_j \in \mathbb{R}, j = 1, ..., m\}$.
- Es liegt dann ein **lineares Ausgleichproblem** vor mit dem Fehlerfunktional

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

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

- Das System $\boldsymbol{A}\lambda = \boldsymbol{y}$ heisst **Fehlergleichungssystem**.

Normalengleichung

Um das Fehlerfunktional zu minimieren müssen alle partiellen Ableitungen von $E(f)$ verschwinden, also $\frac{\partial E(f)}{\partial \lambda_j} = 0$

- Die Gleichungen

$$\frac{\partial E(f)(\lambda_1, ..., \lambda_m)}{\partial \lambda_j} = 0, \quad j = 1, ..., m$$

heissen **Normalgleichungen** des linearen Ausgleichproblems.

- Das System sämtlicher Normalgleichungen heisst **Normalgleichungssystem** und lässt sich als lineares Gleichungssystem schreiben

$$\boldsymbol{A}^T \boldsymbol{A}\lambda = \boldsymbol{A}^T \boldsymbol{y}$$

- Die Lösung $\lambda = (\lambda_1, ..., \lambda_m)^T$ des Normalgleichungssystems beinhaltet die gesuchten Parameter des linearen Ausgleichproblems.

Weil $A^T A$ aber oft schlecht konditioniert ist wird A mittels QR-Verfahren zerlegt

$$A^T A\lambda = A^T y \to A = QR \to R\lambda = Q^T y$$

nichtlineare Ausgleichsrechnung

Parameter λ treten verwoben in der Funktion f auf

$$f = f(\lambda_1, ..., \lambda_m, x)$$

nichtlineare Ausgleichsprobleme

- Gegeben seien n Wertepaare $(x_i, y_i), i = 1, ..., n$, und die Menge F der Ansatzfunktionen $f_p = f_p(\lambda_1, \lambda_2, ..., \lambda_m, x)$ mit m Parametern $\lambda_j \in \mathbb{R}$, $j = 1, ..., m$, also $F = \{ f_p(\lambda_1, \lambda_2, ..., \lambda_m, x) \mid \lambda_j \in \mathbb{R},\ j = 1, ..., m \}$.
- Das **allgemeine Ausgleichproblem** besteht darin, die m Parameter $\lambda_1, ..., \lambda_m$ zu bestimmen, so dass das Fehlerfunktional E

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

  minimal wird unter allen zulässigen Parameterbelegungen.

- Dabei ist

$$\begin{aligned} \boldsymbol{f}(\lambda) &= \boldsymbol{f}(\lambda_1, \lambda_2, ..., \lambda_m) := \begin{pmatrix} f_1(\lambda_1, \lambda_2, ..., \lambda_m) \\ f_2(\lambda_1, \lambda_2, ..., \lambda_m) \\ \vdots \\ f_n(\lambda_1, \lambda_2, ..., \lambda_m) \end{pmatrix} \\ &= \begin{pmatrix} f_p(\lambda_1, \lambda_2, ..., \lambda_m, x_1) \\ f_p(\lambda_1, \lambda_2, ..., \lambda_m, x_2) \\ \vdots \\ f_p(\lambda_1, \lambda_2, ..., \lambda_m, x_n) \end{pmatrix} \\ \boldsymbol{y} &= \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},\ \lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_m \end{pmatrix} \end{aligned}$$

Quadratmittelproblem

Gegeben Funktion $\vec{g} : \mathbb{R}^m \to \mathbb{R}^n$ und zugehöriges Fehlerfunktional $E : \mathbb{R}^m \to \mathbb{R} = ||\vec{g}(x)||_2^2$
Das Problem $\vec{x} \in \mathbb{R}^m$ zu finden für den $E(x)$ minimal wird heisst Quadratmittelproblem.

nichtlineare Ausgleichsprobleme sind Quadratmittelprobleme mit $\vec{g}(x) = \vec{g}(\lambda)$. Als Fehlerfunktional dient die differenz zwischen dem tatsächlichen Wert und dem Wert der Funktion $\vec{f}$.

$$\vec{g}(\vec{\lambda}) = \vec{y} - \vec{f}(\vec{\lambda}), \vec{y} \in \mathbb{R}^n, \vec{\lambda} \in \mathbb{R}^m$$

Gauss Newton-Verfahren

- Kombination aus linearer Ausgleichsrechnung und Newton-Verfahren
- In $||\vec{g}(\lambda)||_2^2$ wird $\vec{g}(\lambda)$ ersetzt durch folgende Linearkombination

$$\vec{g}(\lambda) \approx \vec{g}(\lambda^{(0)}) + D\vec{g}(\lambda^{(0)}) * (\lambda - \lambda^{(0)})$$

- $Dg(..)$ entspricht hierbei der Jacobi-Matrix von $\vec{g}$
- Durch folgende Substitutionen haben wir die Form einer linearen Ausgleichsrechnung $E(\lambda) = ||y - A\delta||_2^2$
  $y = \vec{g}(\lambda^{(0)})$
  $A = -D\vec{g}(\lambda^{(0)})$
  $\delta = \lambda - \lambda^{(0)}$

mögliches Abbruchkriterium: $||\delta^{(k)}||_2 < TOL$

1. Startvektor $\vec{\lambda}^{(0)}$ in der Nähe des Minimums des Fehlerfunktionals E der Ausgleichsfunktion $f(\lambda)$
2. berechne Funktion $g(\lambda) = y - f(\lambda)$
3. und zugehörige Jacobi-Matrix $Dg(\lambda)$
4. Für $k = 0, 1, ...$
   a) $\delta^{(k)}$ aus linearen Ausgleichsproblems (Substitutionen von vorherigem Abschnitt)

$$min||g(\lambda^{(k)}) + Dg(\lambda^{(k)}) * \delta^{(k)}||_2^2$$

   d.h. konkret wieder wie linear ACHTUNG Unterschied resultat ist hierbei aber nur $\delta^{(k)}$ NICHT direkt λ wie bei linear. Das resultierende $\lambda^{(k+1)}$ wird im Schritt c) berechnet

$$A^T A\delta^{(k)} = A^T y$$

   b) stabilere Berechnung durch Verwendung von QR:
      i.

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

      ii. auflösen nach $\delta^{(k)}$

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

   c) erhalte λ für nächsten Iterationsschritt

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

gedämpftes Gauss-Newton-Verfahren

Konvergiert in der Regel für einen grösseren Bereich an Startvektoren, immer noch keine Garantie für Konvergenz.

Mögliches Abbruchkriterium (keine Garantie für richtige Lösung)

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

Im regulären Gauss-Newton-Verfahren kommt unter b) noch ein zusätzlicher Dritter schritt.

4. b) iii. Finde minimales $p \in [0, p_max]$ mit: (sonst $p = 0$)

$$||g(\lambda^{(k)} + \frac{\delta^{(k)}}{2^P})||_2^2 < ||g(\lambda^{(k)})||_2^2$$

   c) Berechnung von $\lambda^{(k+1)}$ wird auch dementsprechend angepasst

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

Beispiel 6.10 (bis zum ersten Iterationsschritt)

Wir wollen also eine Ansatzfunktion $f(x) = ae^{bx}$ bestmöglich im Sinn der kleinsten Fehlerquadrate an die gegebenen Daten anpassen und vergleichen die Resultate des ungedämpften und des gedämpften Gauss-Newton-Verfahrens, einmal für den Startvektor $\lambda^{(0)} = (a^{(0)}, b^{(0)})^T = (1, -1.5)^T$ und dann nochmals für den Startvektor $\lambda^{(0)} = (a^{(0)}, b^{(0)})^T = (2, 2)^T$

x_i	0	1	2	3	4
y_i	3	1	0.5	0.2	0.05

$$\boldsymbol{g}(a,b) := \boldsymbol{y} - \boldsymbol{f}(a,b) = \begin{pmatrix} g_1(a,b) \\ \vdots \\ g_5(a,b) \end{pmatrix} = \begin{pmatrix} y_1 - ae^{bx_1} \\ \vdots \\ y_5 - ae^{bx_5} \end{pmatrix} = \begin{pmatrix} 3 - ae^{b\cdot 0} \\ 1 - ae^{b\cdot 1} \\ 0.5 - ae^{b\cdot 2} \\ 0.2 - ae^{b\cdot 3} \\ 0.05 - ae^{b\cdot 4} \end{pmatrix}$$

$$\boldsymbol{Dg}(a,b) := \begin{pmatrix} \frac{\partial g_1}{\partial a}(a,b) & \frac{\partial g_1}{\partial b}(a,b) \\ \frac{\partial g_2}{\partial a}(a,b) & \frac{\partial g_2}{\partial b}(a,b) \\ \vdots & \vdots \\ \frac{\partial g_5}{\partial a}(a,b) & \frac{\partial g_5}{\partial b}(a,b) \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ -e^{b\cdot 1} & -ae^{b\cdot 1} \\ -e^{b\cdot 2} & -2ae^{b\cdot 2} \\ -e^{b\cdot 3} & -3ae^{b\cdot 3} \\ -e^{b\cdot 4} & -4ae^{b\cdot 4} \end{pmatrix}$$

Für den ersten Schritt des ungedämpften Verfahrens erhalten wir also

$$\boldsymbol{Dg}(1,-1.5) = \begin{pmatrix} -1 & 0 \\ -e^{-1.5} & -e^{-1.5} \\ -e^{-3} & -2e^{-3} \\ -e^{-4.5} & -3e^{-4.5} \\ -e^{-6} & -4e^{-6} \end{pmatrix},\ \boldsymbol{g}(1,-1.5) = \begin{pmatrix} 3-1 \\ 1-e^{-1.5} \\ 0.5-e^{-3} \\ 0.2-e^{-4.5} \\ 0.05-e^{-6} \end{pmatrix}$$

...weiter mit Algorithmus

numerische Integration

Eine Funktion durch aufteilen in Teilintervalle + Approximation angenähert Integrieren.

Rechteck-/Trapezregel

Zur Approximation von Integral

$$I = \int_a^b f(x)dx$$

durch ein Rechteck/Trapez

Rechteckregel

$$Rf = f(\frac{a+b}{2}) * (b-a)$$

Trapezregel

$$Tf = \frac{f(a)+f(b)}{2} * (b-a)$$

summierte Rechteck-/Trapezregel

- $f : [a,b] \to \mathbb{R}$ stetig
- $n \in \mathbb{N}$ Anzahl Subintervalle $[x_i, x_{i+1}]$
- $i \in [0, n-1]$ und $x_n = b$
- Subintervalle von konstanter Breite $h = x_{i+1} - x_i = \frac{b-a}{n} \to x_i = a + i * h$

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

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

Simpson-Regel

Annäherung durch ein Polynom 2. Grades anstatt Rechteck/Trapez

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

summierte Simpson-Regel

Ausgangslage identisch zu summierte Rechteck-/Trapezregel

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

Kann auch als gewichtetes Mittel von Trapez- & Rechteckregel geschrieben werden:

$$Sf(h) = \frac{1}{3}(Tf(h) + 2Rf(h))$$

Fehlerabschätzung der summierten Formeln (Rechteck/Trapez/Simpson)

Die summierte Rechteckregel ist erwartungsweise doppelt so genau wie die Trapezregel, Simpson mit Abstand am genauesten. ($h = \frac{b-a}{n}$)

$$\begin{aligned} |\int_a^b f(x)dx - Rf(h)| &\leq \frac{h^2}{24}(b-a) * \max_{x\in[a,b]} |f''(x)| \\ |\int_a^b f(x)dx - Tf(h)| &\leq \frac{h^2}{12}(b-a) * \max_{x\in[a,b]} |f''(x)| \\ |\int_a^b f(x)dx - Sf(h)| &\leq \frac{h^4}{2880}(b-a) * \max_{x\in[a,b]} |f^{(4)}(x)| \end{aligned}$$

Gaussformeln

Bisher wurden Stützstellen x_i mit konstanter Schrittweite gewählt.
Sie können aber auch so gewählt werden, dass das Integral "optimal" approximiert wird.

Die Idee ist die x_i & a_i aus $I(f) = \sum_{i=1}^{n} a_i f(x_i)$ so zu wählen dass der Fehler möglichst klein wird.

Nach viel rechnen kommt man auf folgende Formeln(n Anzahl Stützstellen):

Die Gauss Formeln für $n = 1, 2, 3$ für $\int_a^b f(x)dx \approx \frac{b-a}{2} \sum_{i=1}^{n} a_i f(x_i)$ lauten:

- **n = 1:** $G_1 f = (b-a) \cdot f(\frac{b+a}{2})$
- **n = 2:** $G_2 f = \frac{b-a}{2} \left[f\left(-\frac{1}{\sqrt{3}} \cdot \frac{b-a}{2} + \frac{b+a}{2}\right) + f\left(\frac{1}{\sqrt{3}} \cdot \frac{b-a}{2} + \frac{b+a}{2}\right) \right]$
- **n = 3:** $G_3 f = \frac{b-a}{2} \left[\frac{5}{9} \cdot f\left(-\sqrt{0.6} \cdot \frac{b-a}{2} + \frac{b+a}{2}\right) + \frac{8}{9} \cdot f\left(\frac{b+a}{2}\right) \right]$

$$+ \frac{b-a}{2} \left[\frac{5}{9} \cdot f\left(\sqrt{0.6} \cdot \frac{b-a}{2} + \frac{b+a}{2}\right) \right]$$

Romberg-Extrapolation

Für die summierte Trapezregel $Tf(h)$ zur näherungsweisen Berechnung von $I(f) = \int_a^b f(x)\,dx$ gilt:
Sei $T_{j0} = Tf\left(\frac{b-a}{2^j}\right)$ für $j = 0, 1, ..., m$. Dann sind durch die Rekursion

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

für $k = 1, 2, ..., m$ und $j = 0, 1, ..., m-k$ Näherungen der Fehlerordnung $2k+2$ gegeben. Diese Methode heisst **Romberg-Extrapolation**. Die verwendete Schrittweitenfolge $h_j = \frac{b-a}{2^j}$ heisst auch Romberg-Folge.

- Wir müssen also zuerst für $k = 0$ die T_{j0} mit der summierten Trapezformel für die fortlaufend halbierten Schrittweiten $h_j = \frac{b-a}{2^j}$ ($j = 0, 1, 2, ...m$) berechnen, z.B. h, $\frac{h}{2}$, $\frac{h}{4}$, $\frac{h}{8}$ für $m = 3$.
- Damit erhalten wir T_{00}, T_{10}, T_{20}, T_{30} in der ersten Spalte des Extrapolationsschemas (s.u.).
- Anschliessend können wir diese Werte extrapolieren und erhalten die T_{jk} (für $k = 1, 2, ...m$) ohne grossen Rechenaufwand, wie im Schema dargestellt.

Die Extrapolation ergibt folgendes Schema ($m = 3$)

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}			

Bemerkungen zur Romberg-Extrapolation

- In der ersten Spalte T_{j0} muss für die summierte Trapezregel Aufgrund von $h_j = \frac{b-a}{2^j}$ natürlich auch $n_j = 2^j$ verwenden
- Da in der ersten Spalte T_{j0} die Trapezeregel für immer kleiner werdende Schrittweiten h_j angewandt wird, werden oft unnötig $f(x_i)$ doppelt ausgewertet. Stattdessen kann man diese spalte auch rekursiv definieren.

$$T_{j0} = \frac{1}{2}T_{j-1,0} + h_j \sum_{i=1}^{n_{j-1}} f(a + (2i-1)h_j)$$

wobei: $n_{j-1} = 2^{j-1}$ & $h_j = \frac{h_{j-1}}{2}$

Gewöhnliche Differentialgleichungen (DGL)

Kurz DGL, die Veränderung ($y'(x)$) einer Funktion an einem Punkt ist abhängig vom Wert der Funktion selbst

$$\frac{dy}{dx} = f(x, y(x))$$

Eine solche gleichung nennt man gewöhnliche Differentialgleichung 1. Ordnung (1. Ordnung weil nur die erste Ableitung vorkommt, engl. ODE = Ordinary Differential Equation)

gewöhnliche Differentialgleichung n-ter Ordnung

Eine Gleichung in der Ableitungen einer unbekannten Funktion $y = y(x)$ bis zur n-ten Ordnung auftreten.

Explizite Form:

$$y^{(n)}(x) = f(x, y(x), y'(x), ..., y^{(n-1)}(x))$$

Beispiel:

$$y''(x) = y'(x)^3 * y(x) + cos(x) = f(x, y(x), y'(x))$$

Gesucht sind Lösungen $y : [a, b] \to \mathbb{R} = y(x)$ dieser Gleichung wobei die Lösungen y auf einem Interval $[a, b]$ definiert sind.

Anfangswertproblem (AWP)

- Bei einem Anfangswertproblem (AWP) für eine Differentialgleichung $n-$ter Ordnung werden der Lösungsfunktion $y = y(x)$ noch n Werte vorgeschrieben, nämlich der Funktionswert an einer bestimmten Stelle x_0 sowie die Werte der ersten $n-1$ Ableitungen an der gleichen Stelle.
- Für die hier betrachteten Differentialgleichungen 1. und 2. Ordnung heisst das:
  - Differentialgleichung 1. Ordnung: Gesucht ist diejenige *spezifische* Lösungskurve $y = y(x)$, die durch den vorgegebenen Punkt $P = (x_0, y(x_0))$ verläuft. Gegeben ist beim AWP also die DGL 1. Ordnung $y'(x) = f(x, y(x))$ und der Anfangswert $y(x_0)$.
  - Differentialgleichung 2. Ordnung: Gesucht ist diejenige *spezifische* Lösungskurve $y = y(x)$, die durch den vorgegebenen Punkt $P = (x_0, y(x_0))$ verläuft und im Punkt x_0 die vorgegebene Steigung $y'(x_0) = m$ besitzt. Gegeben ist beim AWP also die DGL 2. Ordnung $y''(x) = f(x, y(x), y'(x))$ und die Anfangswerte $y(x_0)$ und $y'(x_0)$.

Richtungsfelder (1. Ordnung)

Die verschiedenen Lösungen einer DGL 1. Ordnung lassen sich als Richtungsfeld darstellen. Hierbei wird in jedem Punkt $(x, y(x))$ die Steigung $y'(x) = f(x, y(x))$ als Pfeil eingezeichnet.

In dem man diesen Pfeilen entlang einer Kurve "folgt" hat man eine spezifische Lösung.

Einzelschrittverfahren

- gesucht: Funktion $y : [a, b] \to \mathbb{R}$
- Anfangsbediung $y(a) = y_0$
- Schrittweite $h = \frac{b-a}{n}$ wählen und Intervall $[a, b]$ mit $n + 1$ Gittsterstellen $x_i = a + i * h \mid i = 0, 1, ..., n$ aufteilen
- Ziel: für alle Gitterstellen x_i die Näherungen y_i bestimmen
- im Einzelschrittverfahren wird die nächste stelle x_{i+1} Anhand der bekannten Lösung für $x_i \mid (x_0 = a, y(x_0) = y_0)$ berechnet

$$x_{i+1} = x_i + h$$
$$y_{i+1} = y_i + \text{Steigung} * h$$

Mit der Steigung als eine numerische Näherung für $y'(x) \mid x \in [x_i, x_{i+1}]$

- Die verschiedenen Einzelschrittverfahren unterscheiden sich nur in der Bestimmung der Steigung

Euler-Verfahren

klassisches Euler-Verfahren

Gegeben Anfangswertproblem:

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

Algorithmus:

Für $i = 0, 1, .., n - 1$:

$$x_{i+1} = x_i + h$$
$$k_1 = f(x_i, y_i)$$
$$y_{i+1} = y_i + h * k_1$$

Wobei:

$$x_0 = a$$
$$x_i = a + i * h$$
$$h = (b - a)/n$$

Mittelpunkt Euler-Verfahren

Anstatt das wie beim klassischen direkt die Steigung am Punkt zu x_i, y_i nehmen wird die Steigung im Mittelpunkt $x_i + h/2$ verwendet.

Algorithmus:

Für $i = 0, 1, .., n - 1$:

$$x_{h/2} = x_i + h/2$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + h/2,\ y_i + h/2 * k_1)$$
$$x_{i+1} = x_i + h$$
$$y_{i+1} = y_i + h * k_2$$

modifiziertes Euler-Verfahren

Unterschied hier ist das der Durchschnitt der Steigungen an x_i und x_{i+1} verwendet wird.

Algorithmus:

Für $i = 0, 1, .., n - 1$:

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + h,\ y_i + h * k_1)$$
$$x_{i+1} = x_i + h$$
$$y_{i+1} = y_i + \frac{h}{2}k_1 + \frac{h}{2}k_1$$
$$= y_i + h * \left(\frac{k_1 + k_2}{2}\right)$$

Fehlerordnung

Bemerkungen:

- bei unseren Verfahren ist Konsistenzordnung = Konvergenzordnung
- durch Verkleinern von h kann der Fehler theoretisch beliebig klein werden aber dann überwiegen Rundungsfehler
- das klassiche Euler-Verfahren hat $p = 1$, Mittelpunkt & modifiziertes $p = 2$
- vierstufiges Runge-Kutta hat $p = 4$

lokaler Fehler/Konsistenzordnung

Gegeben:

- DGL $y'(x) = f(x, y(x))$ mit Anfangsbediung $y(x_i) = y_i$
- Exakte Lösung $y(x)$
- Näherungswert y_{i+1} für $y(x_{i+1})$ aus numerischem Verfahren mit Schrittweite h

Der lokale Fehler nach einer Iteration ist dann definiert als:

$$\varphi(x_i, h) = y(x_{i+1}) - y_{i+1}$$

Ein numerisches Verfahren hat Konsistenzordnung p falls:

$$|\varphi(x_i, h)| \leq C * h^{p+1}$$

- für genügend kleine Schrittweite h
- Konstante $C > 0$ hängt von DGL ab
- wird h halbiert, geht der lokale Fehler um Faktor $(\frac{1}{2})^{p+1}$ runter

Da C konstant $\to$ Fehler ist rein von gewählter Schrittweite abhängig und nimmt mit h^{p+1} rasant ab für Verfahren mit hoher Fehlerordnung (GUT)

globaler Fehler/Konvergenzordnung

Gegeben:

- DGL $y'(x) = f(x, y(x))$ mit Anfangsbediung $y(x_i) = y_i$
- Exakte Lösung $y(x)$
- Näherungswert y_n für $y(x_n = x_0 + nh)$ aus numerischem Verfahren mit Schrittweite h

Der globale Fehler nach n Iterationen ist dann definiert als:

$$\varphi(x_i, h) = y(x_n) - y_n$$

Ein Verfahren hat Konvergenzordnung p falls:

$$|y(x_n) - y_n| \leq C * h^p$$

Runge-Kutta Verfahren

Wie Euler-Verfahren aber nochmehr k_i Zwischenpunkte für die Steigung.

Gegeben Anfangswertproblem:

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

klassisches vierstufiges Runge-Kutta Verfahren

Für $i = 0, 1, .., n-1$:

$$\begin{aligned} x_{h/2} &= x_i + h/2 \\ k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + h/2,\ y_i + h/2 * k_1) \\ k_3 &= f(x_i + h/2,\ y_i + h/2 * k_2) \\ k_4 &= f(x_i + h,\ y_i + h * k_3) \\ x_{i+1} &= x_i + h \\ y_{i+1} &= y_i + h * \frac{1}{6}(k_1 + 2k_2 + *2k_3 + k4) \end{aligned}$$

allgemeines s-stufiges Runge-Kutta Verfahren

- Ein allgemeines (explizites) $s-$stufiges Runge-Kutta Verfahren ist gegeben durch die Formeln

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

- Hierbei ist $s \in \mathbb{N}$ die Stufenzahl und a_{nm}, b_n, c_n sind Konstanten. Die Konsistenz- und Konvergenzordnung hängt von der Wahl dieser Konstanten ab.

Die Koeffizienten können wie folgt in einer Matrix notiert werden

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

Die Euler-Verfahren können auch so dargestellt werden

- 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, $s = 4$:

$$\begin{array}{c|cccc} 0 & & & & \\ 0.5 & 0.5 & & & \\ 0.5 & 0 & 0.5 & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Systeme von Differentialgleichungen

DGL von k-ter Ordnung auf ein System von DGL 1. Ordnung zurückführen.

Anhand Beispiel: (8.9 im Skript)

- 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:

$$\begin{aligned} z_1(x) &= y(x) \\ z_2(x) &= y'(x) \\ z_3(x) &= y''(x) \end{aligned}$$

- 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

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

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

$$\vec{f}(x, \vec{z}) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -5 & -8 & -6 \end{pmatrix} \vec{z} + \begin{pmatrix} 0 \\ 0 \\ 10e^{-x} \end{pmatrix}$$

So wurde aus einer DGL 3. Ordnung $\rightarrow$ drei DGL 1. Ordnung mit folgendem Anfangswertproblem 1. Ordnung

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

Lösen eines DGL Systemes

Identisch zu bekannten Runge-Kutta verfahren, einfach alles Vektoren. (Hochgestellter Index in Klammern wieder für Iterationen weil tiefe für einzelne Vektorelemente)

Beispiel Euler: für $i = 0, 1, .., n-1$:

$$\begin{aligned} x_{i+1} &= x_i + h \\ \vec{k}_1 &= \vec{f}(x_i, \vec{y}^{(i)}) \\ \vec{y}^{(i+1)} &= \vec{y}^{(i)} + h * \vec{k}_1 \end{aligned}$$

mit:

$$\vec{y}(x) = \begin{pmatrix} y_1(x) \\ \vdots \\ y_k(x) \end{pmatrix} \qquad \vec{f}(x, \vec{y}(x)) = \begin{pmatrix} f_1(x, \vec{y}(x)) \\ \vdots \\ f_1(x, \vec{y}(x)) \end{pmatrix}$$

$$\vec{y}'(x) = \begin{pmatrix} y_1'(x) \\ \vdots \\ y_k'(x) \end{pmatrix}$$