Höhere Mathematik 2

Jil Zerndt
FS 2025

Numerische Lösung nicht linearer Gleichungssysteme

LGS = lineares Gleichungssystem, NGS = nichtlineares Gleichungssystem

Einleitendes Beispiel
Gesucht sind die Lösungen des Gleichungssystems:

$$f_1(x_1, x_2) = x_1^2 + x_2 - 11 = 0$$
$$f_2(x_1, x_2) = x_1 + x_2^2 - 7 = 0$$

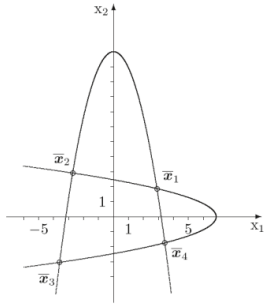
Diese lassen sich interpretieren als die Nullstellen der Funktion:

$$\mathbf{f}: \mathbb{R}^2 \to \mathbb{R}^2 \quad \mathbf{f}(x) = \begin{pmatrix} f_1(x_1,x_2) \\ f_2(x_1,x_2) \end{pmatrix} = \begin{pmatrix} x_1^2+x_2-11 \\ x_1+x_2^2-7 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Ein solches System lässt sich nicht in die Form $Ax = b$ bringen.

Geometrisch lassen sich die Lösungen als Schnittpunkte der beiden Funktionen interpretieren.
Explizite Darstellung der Kurven:

$$x_2 = 11 - x_1^2 \text{ und } x_2 = \sqrt{7 - x_1}$$



Schnittpunkte:

$$\overline{x_1} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}, \quad \overline{x_2} = \begin{pmatrix} -2.8 \\ 3.2 \end{pmatrix}$$
$$\overline{x_3} = \begin{pmatrix} -3.8 \\ -3.3 \end{pmatrix}, \quad \overline{x_4} = \begin{pmatrix} 3.4 \\ -1.7 \end{pmatrix}$$

Funktionen mit mehreren Variablen

Skalarwertige Funktionen $f: D \subset \mathbb{R}^n \to W \subset \mathbb{R}$

$$(x_1, x_2, \ldots, x_n) \mapsto y = f(x_1, x_2, \ldots, x_n)$$

Unter einer Funktion f mit n unabhängigen Variablen $x_1, \ldots, x_n$ und einer abhängigen Variablen y versteht man eine Vorschrift, die jedem geordneten Zahlentupel $(x_1, x_2, \ldots, x_n)$ aus einer Definitionsmenge $D \subset \mathbb{R}^n$ genau ein Element $y \in W \subset \mathbb{R}$ zuordnet.

Da das Ergebnis $y \in \mathbb{R}$ ein Skalar (eine Zahl) ist, redet man auch von einer **skalarwertigen Funktion**.

Vektorwertige Funktion gibt einen **Vektor** zurück (statt Skalar)
Sei $\mathbf{f}: \mathbb{R}^n \to \mathbb{R}^m$ eine Funktion mit n Variablen.

$$\mathbf{f}(x_1 \ldots, x_n) = \begin{pmatrix} y_1 = f_1(x_1, x_2, \ldots, x_n) \\ y_2 = f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ y_m = f_m(x_1, x_2, \ldots, x_n) \end{pmatrix}$$

wobei die m Komponenten $f_i: \mathbb{R}^n \to \mathbb{R}$ für $i = 1, 2, \ldots, n$ von $\mathbf{f}$ wieder **skalarwertige** Funktionen sind.

Eigenschaften von skalar- und vektorwertigen Funktionen

- Skalar- und vektorwertige Funktionen mit mehreren Variablen werden auch **multivariat** genannt.
- Wie bei einem Vektor $\mathbf{x}$ stellen wir zur besseren Unterscheidbarkeit vektorwertige Funktionen $\mathbf{f}$ fett dar, im Gegensatz zu Skalaren x und skalarwertigen Funktionen f.
- Wir werden uns bei der Lösung nichtlinearer Gleichungssysteme auf vektorwertige Funktionen $\mathbf{f} = \mathbb{R}^n \to \mathbb{R}^n$ konzentrieren.

Grundlegende Rechenoperationen
können als Skalarwertige Funktionen $f: \mathbb{R}^2 \to \mathbb{R}$ oder als Vektorwertige Funktionen $\mathbf{f}: \mathbb{R}^2 \to \mathbb{R}^2$ interpretiert werden:

$$f(x,y) = x + y, \quad g(x,y) = x \cdot y, \quad h(x,y) = x^2 + y^2$$

$$\mathbf{f}(x,y) = \begin{pmatrix} x+y \\ x \cdot y \end{pmatrix}, \quad \mathbf{g}(x,y) = \begin{pmatrix} x \cdot y \\ x^2 + y^2 \end{pmatrix}$$

Lineare Funktionen von LGS Gebe die lineare Funktion $\mathbf{f}: \mathbb{R}^3 \to \mathbb{R}^3$ an, für welche die Lösung $\mathbf{x}$ des LGS:

$\mathbf{Ax} = \mathbf{b}$ mit $\mathbf{A} = \begin{pmatrix} 4 & -1 & 1 \\ -2 & 5 & 1 \\ 1 & -2 & 5 \end{pmatrix}$ und $\mathbf{b} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ gerade $\mathbf{f(x)} = \mathbf{0}$ ergibt:

$$\mathbf{A}\vec{x} = \vec{b} \Rightarrow \underbrace{\mathbf{A}\vec{x} - \vec{b} = \vec{0}}_{\vec{f}(\vec{x})} \Rightarrow \vec{f}(x_1, x_2, x_3) = 0 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\vec{f}(\vec{x}) = \mathbf{A}\vec{x} - \vec{b} = \begin{pmatrix} 4 & -1 & 1 \\ -2 & 5 & 1 \\ 1 & -2 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - \begin{pmatrix} 5 \\ 11 \\ 12 \end{pmatrix}$$

$$\mathbf{f}(x_1, x_2, x_3) = \begin{pmatrix} f_1 = 4x_1 - x_2 + x_3 - 5 \\ f_2 = -2x_1 + 5x_2 + x_3 - 11 \\ f_3 = x_1 - 2x_2 + 5x_3 - 12 \end{pmatrix}$$

Darstellungsformen

Analytische Darstellung

- **Explizite Darstellung:** $y = f(x_1, \ldots, x_n)$
  - die Funktionsgleichung ist nach einer Variablen aufgelöst
  - Beispiel: $y = 2 \cdot e^(x_1^2 + x_2^2)$
- **Implizite Darstellung:** $F(x, y) = 0$
  - die Funktionsgleichung ist nicht nach einer Variablen aufgelöst
  - daher handelt es sich um eine Funktion mit nur $n - 1$ unabhängigen Variablen
  - Beispiel: $x_1^2 + x_2^2 - 1 = 0$
- **Parameterdarstellung:** $x = x(t), y = y(t)$
  - die Funktion wird durch eine Kurve im Raum beschrieben
  - Beispiel: $x(t) = \cos(t), y(t) = \sin(t)$

Darstellung durch Wertetabelle Sei $f: \mathbb{R}^n \to \mathbb{R}^m$ eine Funktion.
In die vorausgesetzte Funktionsgleichung $z = f(x, y)$ werden die Werte der unabhängigen Variablen x und y eingesetzt (der Reihe nach).
So erhält man eine Wertetabelle, bzw. Matrix:

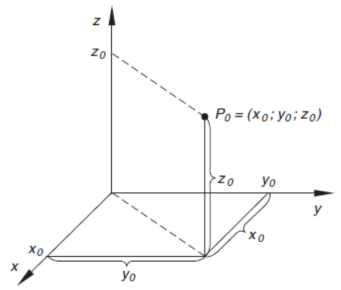
2. unabhängige Variable y

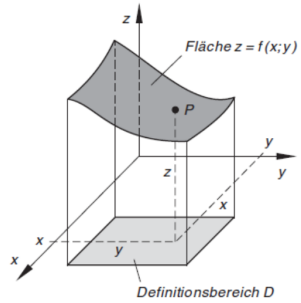
1. unabhängige Variable x \ y	y_1	y_2	…	y_k	…	y_n
x_1	z_{11}	z_{12}	…	z_{1k}	…	z_{1n}
x_2	z_{21}	z_{22}	…	z_{2k}	…	z_{2n}
⋮	⋮	⋮	…	⋮	…	⋮
x_i	z_{i1}	z_{i2}	…	z_{ik}	…	z_{in} ← i-te Zeile
⋮	⋮	⋮	…	⋮	…	⋮
x_m	z_{m1}	z_{m2}	…	z_{mk}	…	z_{mn}

↑ k-te Spalte

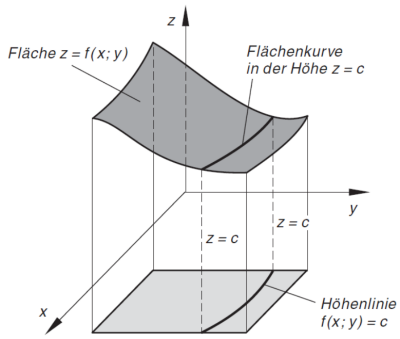
Grafische Darstellung Wir beschränken uns hier auf skalarwertige Funktionen mit zwei unabhängigen Variablen $f: \mathbb{R}^2 \to \mathbb{R}$.
Dazu betrachten wir die Funktion $z = f(x, y)$ in einem dreidimensionalen kartesischen Koordinatensystem:

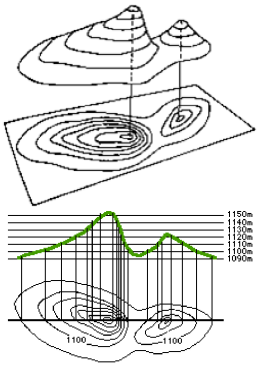
Darstellung einer Funktion als Fläche im Raum
Die Funktion f ordnet jedem Punkt $(x, y) \in D$ in der Ebene einen Wert $z = f(x, y)$ zu, der als Höhenkoordinate verstanden werden kann. Durch die Anordnung der Punkte $(x, y, f(x, y))$ im dreidimensionalen Koordinatensystem wird eine über dem Definitionsbereich D liegende Fläche ausgezeichnet:





Schnittkurvendiagramm
Wird die Fläche $z = f(x, y)$ bei einer konstanten Höhe $z = \text{const.}$ geschnitten, ergibt sich eine Schnittkurve. Wird diese in die (x, y)-Ebene projiziert, spricht man von einer Höhenlinie bzw. bei der Abbildung von einem Höhenliniendiagramm., wie wir es z.B. von Wanderkarten her kennen. Natürlich kann man auch andere Schnitte als $z = \text{const.}$ (Schnittebene parallel zur (x, y)-Ebene) wählen, z.B. $x = \text{const.}$ (Schnittebene parallel zur (y, z)-Ebene) oder $y = \text{const.}$ (Schnittebene parallel zur (x, z)-Ebene):





Partielle Ableitungen

Partielle Ableitungen 1. Ordnung

Unter den partiellen Ableitungen 1. Ordnung einer Funktion $z = f(x, y)$ an der Stelle (x, y) werden die folgenden Grenzwerte verstanden:

Partielle Ableitung nach x:

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

Partielle Ableitung nach y:

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

Alternative Notationen: $f_x(x,y)$, $f_y(x,y)$ oder f_x, f_y

Partielle Ableitung

In einer Dimension

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{\big(f(x_0+\Delta x) - f(x_0)\big)}{\Delta x}$$

In mehreren Dimensionen

1. Ableitung nach x: $f_x = \frac{\partial f}{\partial x}(x,y) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x, y) - f(x,y)}{\Delta x}$

2. Ableitung nach y: $f_y = \frac{\partial f}{\partial y}(x,y) = \lim_{\Delta y \to 0} \frac{f(x, y+\Delta y) - f(x,y)}{\Delta y}$

Partielle Ableitungen berechnen

1. Variable identifizieren: Bestimme, nach welcher Variable abgeleitet werden soll
2. Alle anderen Variablen werden während der Ableitung als Konstanten betrachtet
3. Standardableitungsregeln anwenden und Ergebnis korrekt notieren

Partielle Ableitungen berechnen

Berechne die partiellen Ableitungen von $z = f(x,y) = 3xy^2 + \ln(x^3y^2)$ an der Stelle $(x_0, y_0) = (-1, 1)$.

In beiden Dimensionen berechnen

$$f_x = 3 \cdot 1 \cdot y^2 + \frac{1}{x^3y^2} \cdot 3x^2 \cdot y^2 = 3y^2 + \frac{3}{x}$$

$$f_y = 3x \cdot 2y + \frac{1}{x^3y^2} \cdot x^3 \cdot 2y = 6xy + \frac{2}{y}$$

An der Stelle $(-1, 1)$:
$f_x(-1,1) = 3 \cdot 1^2 + \frac{3}{-1} = 3 - 3 = 0$
$f_y(-1,1) = 6 \cdot (-1) \cdot 1 + \frac{2}{1} = -6 + 2 = -4$

Tangentensteigung

$z = f(x,y) = 3xy^3 + 10x^2y + 5y + 3y \cdot \sin(5xy)$

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

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

Steigung der beiden Tangenten in $x-/y$-Richtung im Punkt: $(x_0, y_0) = (-1, 1)$

Jacobi-Matrix und Linearisierung

Jacobi-Matrix

Sei $f : \mathbb{R}^n \to \mathbb{R}^m$ mit $y = f(x)$ und $x = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$. Die Jacobi-Matrix enthält sämtliche partiellen Ableitungen 1. Ordnung von f und ist definiert als:

$$f(x) = \begin{pmatrix} y_1 = f_1(x) \\ y_2 = f_2(x) \\ \vdots \\ y_m = f_m(x) \end{pmatrix} \to Df(x) := \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \cdots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x) & \frac{\partial f_m}{\partial x_2}(x) & \cdots & \frac{\partial f_m}{\partial x_n}(x) \end{bmatrix}$$

Linearisierung

Die verallgemeinerte **Tangentengleichung**

$$g(x) = f(x^{(0)}) + Df(x^{(0)}) \cdot (x - x^{(0)})$$

beschreibt eine lineare Funktion und es gilt $f(x) \approx g(x)$ in einer Umgebung eines gegebenen Vektors $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^T \in \mathbb{R}^n$. Man spricht von der **Linearisierung** der Funktion $y = f(x)$ in einer Umgebung von $x^{(0)}$ (ein hochgestellter Index in Klammern $x^{(k)}$ bezeichnet wie bisher einen Vektor aus $\mathbb{R}^n$ nach der k-ten Iteration).

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:

$$Df(x^{(0)}) = \left(\frac{\partial f}{\partial x_1}(x_1^{(0)}, x_2^{(0)}), \frac{\partial f}{\partial x_2}(x_1^{(0)}, x_2^{(0)}) \right)$$

Dann liefert die Linearisierung $g(x_1, x_2)$:

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

$$= f(x_1^{(0)}, x_2^{(0)}) + \frac{\partial f}{\partial x_1}(x_1^{(0)}, x_2^{(0)}) \cdot (x_1 - x_1^{(0)}) + \frac{\partial f}{\partial x_2}(x_1^{(0)}, x_2^{(0)}) \cdot (x_2 - x_2^{(0)})$$

die Gleichung der Tangentialebene.
Sie enthält sämtliche im Flächenpunkt $\dot{P} = \left(x_1^{(0)}, x_2^{(0)}, f(x_1^{(0)}, x_2^{(0)}) \right)$ an die Bildfläche von $y = f(x_1, x_2)$ angelegten Tangenten.

Jacobi-Matrix berechnen und linearisieren

Sei $f : \mathbb{R}^n \to \mathbb{R}^m$ mit $y = f(x)$ und $x = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$.

1. Identifiziere die Komponentenfunktionen $f_1, f_2, ..., f_m$ und Variablen $x_1, x_2, ..., x_n$.
2. Berechne die partiellen Ableitungen $\frac{\partial f_i}{\partial x_j}$ für $i = 1, ..., m$ und $j = 1, ..., n$.
3. Stelle die Jacobi-Matrix $Df(x)$ auf, indem du die partiellen Ableitungen in Matrixform anordnest.
4. Werte die Jacobi-Matrix an einem Entwicklungspunkt $x^{(0)}$ aus, indem du die Werte der Variablen in die Matrix einsetzt.
5. Berechne die Linearisierung der Funktion $g(x)$ mit der Formel: $g(x) = f(x^{(0)}) + Df(x^{(0)}) \cdot (x - x^{(0)})$

Jacobi-Matrix und Linearisierung

Berechnen Sie die Jacobi-Matrix von $f(x_1, x_2) = \begin{pmatrix} 5x_1x_2 \\ x_1^2x_2^2 + x_1 + 2x_2 \end{pmatrix}$ an der Stelle $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ und linearisieren Sie die Funktion dort.

Schritt 1: Partielle Ableitungen berechnen

$$\frac{\partial f_1}{\partial x_1} = 5x_2, \quad \frac{\partial f_1}{\partial x_2} = 5x_1$$

$$\frac{\partial f_2}{\partial x_1} = 2x_1x_2^2 + 1, \quad \frac{\partial f_2}{\partial x_2} = 2x_1^2x_2 + 2$$

Schritt 2: Jacobi-Matrix aufstellen

$$Df(x_1, x_2) = \begin{bmatrix} 5x_2 & 5x_1 \\ 2x_1x_2^2 + 1 & 2x_1^2x_2 + 2 \end{bmatrix}$$

Schritt 3: An der Stelle $(1, 2)$ auswerten $Df(1,2) = \begin{bmatrix} 10 & 5 \\ 9 & 6 \end{bmatrix}$

Schritt 4: Funktionswert berechnen $f(1,2) = \begin{pmatrix} 10 \\ 9 \end{pmatrix}$

Schritt 5: Linearisierung $g(x) = \begin{pmatrix} 10 \\ 9 \end{pmatrix} + \begin{bmatrix} 10 & 5 \\ 9 & 6 \end{bmatrix} \begin{pmatrix} x_1 - 1 \\ x_2 - 2 \end{pmatrix}$

Linearisierung einer vektorwertigen Funktion

Linearisiere Sie für $x^{(0)} = (\pi/4, 0, \pi)^T$ der Funktion $f(x_1, x_2, x_3)$

1. Jacobi-Matrix $Df(x_1, x_2, x_3)$ bilden:

$$f(x_1, x_2, x_3) = \begin{pmatrix} \sin(x_2 + 2x_3) \\ \cos(2x_1 + x_2) \end{pmatrix}$$

$$Df(x_1, x_2, x_3) = \begin{bmatrix} 0 & \cos(x_2 + 2x_3) & 2\cos(x_2 + 2x_3) \\ -2\sin(2x_1 + x_2) & -\sin(2x_1 + x_2) & 0 \end{bmatrix}$$

2. Startvektor $x^{(0)}$ in Vektorwertige Funktion $f(x)$ und Jacobi-Matrix $Df(x)$ **einsetzen:**

$$f(\pi/4, 0, \pi) = f(x^{(0)}) = \begin{pmatrix} \sin(0 + 2\pi) \\ \cos(2 \cdot \pi/4 + 0) \end{pmatrix}$$

$$= \begin{pmatrix} 0 \\ 0 \end{pmatrix}, Df(x^{(0)}) = \begin{bmatrix} 0 & 1 & 2 \\ -2 & -1 & 0 \end{bmatrix}$$

3. Verallgemeinerte Tangentengleichung:

$$g(x) = \underbrace{\begin{pmatrix} 0 \\ 0 \end{pmatrix}}_{f(x_0)} + \underbrace{\begin{bmatrix} 0 & 1 & 2 \\ -2 & -1 & 0 \end{bmatrix}}_{Df(x_0)} \cdot \underbrace{\begin{pmatrix} x_1 - \pi/4 \\ x_2 - 0 \\ x_3 - \pi \end{pmatrix}}_{x - x_0} = \begin{pmatrix} x_2 + 2x_3 - 2\pi \\ -2x_1 - x_2 + \pi/2 \end{pmatrix}$$

Nullstellenbestimmung für NGS

Es gibt keine einfachen Methoden, um festzustellen, ob ein nichtlineares Gleichungssystem lösbar ist und wie viele Lösungen es hat. Deshalb entscheidet die Wahl einer «geeigneten Startnäherug» meist über erfolgt oder Misserfolg der eingesetzten numerischen Verfahren.

Problemstellung zur Nullstellenbestimmung

Gegeben sei $n \in \mathbb{N}$ und eine Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$. Gesucht ist ein Vektor $\bar{x} \in \mathbb{R}^n$ mit $f(\bar{x}) = 0$.
Komponentenweise bedeutet dies: Gegeben sind n Funktionen $f_i : \mathbb{R}^n \to \mathbb{R}$, die die Komponenten von f bilden. Gesucht ist ein Vektor $\bar{x} \in \mathbb{R}^n$ mit $f_i(\bar{x}) = 0$ für $i = 1, ..., n$.

Quadratisch konvergentes Newton-Verfahren (Quadratische Konv.)

Gesucht sind Nullstellen von $f : \mathbb{R}^n \to \mathbb{R}^n$. Sei $x^{(0)}$ ein Startvektor in der Nähe einer Nullstelle. Das Newton-Verfahren zur näherungsweisen Bestimmung dieser Nullstelle lautet:
Lösung von $f(x) = 0$ mit $f : \mathbb{R}^n \to \mathbb{R}^n$ für $n = 0, 1, 2, \ldots$
1. Berechne $f(x^{(n)})$ und $Df(x^{(n)})$
2. Berechne $\delta^{(n)}$ als Lösung des linearen Gleichungssystems

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

3. Setze $x^{(n+1)} := x^{(n)} + \delta^{(n)}$

Vereinfachtes Newton-Verfahren (Lineare Konvergenz)

Lösung von $f(x) = 0$ mit $f : \mathbb{R}^n \to \mathbb{R}^n$ für $n = 0, 1, 2, \ldots$
1. Berechne $f(x^{(n)})$ und $Df(x^{(0)})$
2. Berechne $\delta^{(n)}$ als Lösung des lin. GS $Df\left(x^{(0)}\right) \cdot \delta^{(n)} = -f\left(x^{(n)}\right)$
3. Setze $x^{(n+1)} := x^{(n)} + \delta^{(n)}$

Newton-Verfahren für nichtlineare Gleichungssysteme

Schritt 1: Funktionen und Jacobi-Matrix aufstellen

Definiere $f(x) = 0$ und berechne die Jacobi-Matrix $Df(x)$.

Schritt 2: Startvektor wählen

Wähle einen geeigneten Startvektor $x^{(0)}$ nahe der vermuteten Lösung.

Schritt 3: Iterative Berechnung

Für jede Iteration k:
- **Linearisierung um $x^{(k)}$**: Berechne $f(x^{(k)})$ und $Df(x^{(k)})$
- **Nullstelle der Linearisierung berechnen**:
  Löse das LGS: $Df(x^{(k)})\delta^{(k)} = -f(x^{(k)})$
- **Nächste Iteration**: Setze $x^{(k+1)} = x^{(k)} + \delta^{(k)}$

Formel: $x^{(k+1)} = x^{(k)} - (Df(x^{(k)}))^{-1} \cdot f(x^{(k)})$

Schritt 4: Konvergenzprüfung

Prüfe Abbruchkriterien wie

$$\|f(x^{(k+1)})\|_2 < \text{TOL} \text{ oder } \|x^{(k+1)} - x^{(k)}\|_2 < \text{TOL}$$

Schritt 5: Lösung interpretieren

Die konvergierte Lösung $x^{(k)}$ ist eine Näherung für die Nullstelle.

Newton-Verfahren anwenden

Löse das Gleichungssystem

$$f(x_1, x_2) = \begin{pmatrix} 20 - 18x_1 - 2x_2^2 \\ -4x_2(x_1 - x_2^2) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

mit dem Newton-Verfahren für den Startvektor $x^{(0)} = (1.1, 0.9)^T$.

Schritt 1: Jacobi-Matrix berechnen

$$Df(x_1, x_2) = \begin{bmatrix} -18 & -4x_2 \\ -4x_2 & -4(x_1 - 3x_2^2) \end{bmatrix}$$

Schritt 2: Erste Iteration ($k = 0$)

$$f(1.1, 0.9) = \begin{pmatrix} -1.42 \\ -0.036 \end{pmatrix}$$

$$Df(1.1, 0.9) = \begin{bmatrix} -18 & -3.6 \\ -3.6 & -5.32 \end{bmatrix}$$

LGS lösen: $\begin{bmatrix} -18 & -3.6 \\ -3.6 & -5.32 \end{bmatrix} \delta^{(0)} = \begin{pmatrix} 1.42 \\ 0.036 \end{pmatrix} \Rightarrow \delta^{(0)} = \begin{pmatrix} -0.0822 \\ 0.0178 \end{pmatrix}$

$$x^{(1)} = \begin{pmatrix} 1.1 \\ 0.9 \end{pmatrix} + \begin{pmatrix} -0.0822 \\ 0.0178 \end{pmatrix} = \begin{pmatrix} 1.0178 \\ 0.9178 \end{pmatrix}$$

Weitere Iterationen führen zur Konvergenz.

Beispiel mit Newton-Verfahren

Gegeben sind zwei Gleichungen und der Start-Vektor $x^{(0)} = (2, -1)^T$

$$1 - x^2 = y^2, \quad \frac{(x-2)^2}{a} + \frac{(y-1)^2}{b} = 1$$

Umwandlung in Funktionen $f_1, f_2 = 0$

$$f_1(x, y) = 1 - x^2 - y^2 = 0, \quad f_2(x, y) = \frac{(x-2)^2}{a} + \frac{(y-1)^2}{b} - 1 = 0$$

Vektorwertige Funktion und Jacobi-Matrix bilden

$$Df(x, y) = \begin{pmatrix} -2x & -2y \\ \frac{2x-4}{a} & \frac{2y-2}{b} \end{pmatrix}, \quad f(x, y) = \begin{pmatrix} 1 - x^2 - y^2 \\ \frac{(x-2)^2}{a} + \frac{(y-1)^2}{b} - 1 \end{pmatrix}$$

Start-Vektor $x^{(0)}$ einsetzen

$$Df(2, -1) = \begin{pmatrix} -4 & 2 \\ 0 & -4/b \end{pmatrix}, \quad f(2, -1) = \begin{pmatrix} -4 \\ 4/b - 1 \end{pmatrix}$$

Berechne $\delta^{(0)}$

$$\left(Df\left(x^{(0)}\right) \mid -f\left(x^{(0)}\right)\right) = \left(\begin{array}{cc|c} -4 & 2 & 4 \\ 0 & -4/b & -4/b + 1 \end{array}\right) \to \underbrace{\begin{pmatrix} -1 \\ 0 \end{pmatrix}}_{\delta^{(0)}}$$

Berechne $x^{(1)}$

$$x^{(1)} = \underbrace{\begin{pmatrix} 2 \\ -1 \end{pmatrix}}_{x^{(0)}} + \underbrace{\begin{pmatrix} -1 \\ 0 \end{pmatrix}}_{\delta^{(0)}} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Gedämpftes Newton-Verfahren

Gedämpftes Newton-Verfahren

Nur in der Nähe der Nullstelle ist Konvergenz des Verfahrens garantiert!
1. Berechne $f\left(x^{(n)}\right)$ und $Df\left(x^{(n)}\right)$
2. Berechne $\delta^{(n)}$ als Lösung des lin. GS $Df\left(x^{(n)}\right) \cdot \delta^{(n)} = -f\left(x^{(n)}\right)$
3. Finde das minimale $k \in \{0, 1, \ldots, k_{\max}\}$ mit:

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

Kein minimales k gefunden $\to k = 0$
4. Setze

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

Dämpfung für bessere Konvergenz

Falls die Jacobi-Matrix $Df(x^{(n)})$ schlecht konditioniert ist, kann das Standard Newton-Verfahren divergieren. Das gedämpfte Newton-Verfahren verwendet eine variable Schrittweite:

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

wobei p das kleinste Element aus $\{0, 1, ..., p_{\max}\}$ ist, für das gilt:

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

Gedämpftes Newton-Verfahren

Schritt 1-3: Wie beim Standard Newton-Verfahren

Berechne $\delta^{(k)}$ durch Lösen von $Df(x^{(k)})\delta^{(k)} = -f(x^{(k)})$.

Schritt 4: Dämpfungsparameter bestimmen

Finde das kleinste $p \in \{0, 1, ..., p_{\max}\}$ mit:

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

Schritt 5: Gedämpften Schritt ausführen

Setze $x^{(k+1)} = x^{(k)} + \frac{\delta^{(k)}}{2^p}$.

Schritt 6: Bei Nicht-Konvergenz

Falls kein geeignetes p gefunden wird, setze $p = 0$ und fahre fort.

Anwendung des gedämpften Newton-Verfahrens

Aufgabe: Lösen Sie das System aus dem vorigen Beispiel mit gedämpftem Newton-Verfahren und einem 'schlechteren' Startvektor $x^{(0)} = (2, 2)^T$.

Lösung: Das gedämpfte Newton-Verfahren konvergiert auch für diesen weiter entfernten Startvektor, während das ungedämpfte Verfahren divergieren würde. Die Dämpfung sorgt dafür, dass in jeder Iteration das Fehlerfunktional $\|f(x)\|_2$ abnimmt, wodurch die Stabilität erhöht wird.

Das gedämpfte Newton-Verfahren ist besonders nützlich bei:
- Schlecht konditionierten Problemen
- Startvektoren, die weit von der Lösung entfernt sind
- Systemen mit mehreren Lösungen
- Praktischen Anwendungen, wo Robustheit wichtiger als Geschwindigkeit ist

Anwendungsbeispiele

LORAN-Navigationssystem

Aufgabe: Bestimmen Sie die Position eines Empfängers aus den hyperbelförmigen Ortskurven:

$$f_1(x,y) = \frac{x^2}{186^2} - \frac{y^2}{300^2 - 186^2} - 1 = 0$$

$$f_2(x,y) = \frac{(y-500)^2}{279^2} - \frac{(x-300)^2}{500^2 - 279^2} - 1 = 0$$

Lösung:

1. **Grafische Analyse:** Plotte beide Hyperbeln und bestimme visuell die vier Schnittpunkte
2. **Numerische Lösung:** Verwende die geschätzten Positionen als Startvektoren für das Newton-Verfahren
3. **Iteration:** Führe Newton-Verfahren mit Genauigkeit $\|f(x^{(k)})\|_2 < 10^{-5}$ durch

Die vier Lösungen entsprechen den möglichen Positionen des Empfängers, wobei durch zusätzliche Information (z.B. dritter Sender) die eindeutige Position bestimmt werden kann.

Dreidimensionales nichtlineares System

Aufgabe: Lösen Sie das System:

$$f(x_1,x_2,x_3) = \begin{pmatrix} x_1 + x_2^2 - x_3^2 - 13 \\ \ln\frac{x_2}{4} + e^{0.5x_3 - 1} - 1 \\ (x_2 - 3)^2 - x_3^3 + 7 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

mit Startvektor $x^{(0)} = (1.5, 3, 2.5)^T$.

Lösung: Dieses System erfordert das gedämpfte Newton-Verfahren aufgrund der komplexen nichtlinearen Terme. Die Jacobi-Matrix enthält sowohl polynomiale als auch transzendente Funktionen, was eine sorgfältige numerische Behandlung erfordert.

Ausgleichsrechnung

Interpolation

Die Interpolation ist ein Spezialfall der linearen Ausgleichsrechnung, bei dem wir zu einer Menge von vorgegebenen Punkten eine Funktion suchen, die exakt durch diese Punkte verläuft.

Interpolationsproblem

Gegeben sind $n+1$ Wertepaare (x_i, y_i), $i = 0, ..., n$, mit $x_i \neq x_j$ für $i \neq j$.
Gesucht ist eine stetige Funktion g mit der Eigenschaft $g(x_i) = y_i$ für alle $i = 0, ..., n$.
Die $n+1$ Wertepaare (x_i, y_i) heißen **Stützpunkte**, die x_i **Stützstellen** und die y_i **Stützwerte**.

Interpolation vs. Ausgleichsrechnung

- **Interpolation:** Die gesuchte Funktion geht **exakt** durch alle Datenpunkte
- **Ausgleichsrechnung:** Die gesuchte Funktion **approximiert** die Datenpunkte möglichst gut
- Interpolation ist ein Spezialfall der Ausgleichsrechnung ($m = n$, Fehlerfunktional $E(f) = 0$)

Polynominterpolation

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.

$P_n(x)$ lautet in der Lagrangeform: $P_n(x) = \sum_{i=0}^{n} l_i(x) y_i$

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

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

Fehlerabschätzung

Sind die y_i Funktionswerte einer genügend oft stetig differenzierbaren Funktion f (also $y_i = f(x_i)$), dann ist der Interpolationsfehler an einer Stelle x gegeben durch

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

Lagrange-Interpolation durchführen

Schritt 1: Stützpunkte identifizieren

Gegeben: $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$ und gesuchter Punkt x.

Schritt 2: Lagrangepolynome berechnen

Für jeden Index $i = 0, 1, ..., n$ berechne:

$$l_i(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)}$$

Schritt 3: Interpolationspolynom aufstellen

$$P_n(x) = y_0 \cdot l_0(x) + y_1 \cdot l_1(x) + \cdots + y_n \cdot l_n(x)$$

Schritt 4: Funktionswert berechnen

Setze den gewünschten x-Wert ein: $P_n(x) =$ gesuchter Interpolationswert.

Lagrange-Interpolation anwenden Bestimme den Atmosphärendruck in 3750m Höhe mittels Lagrange-Interpolation:

Höhe [m]	0	2500	5000	10000
Druck [hPa]	1013	747	540	226

Lösung: Verwende die Stützpunkte $(0, 1013)$, $(2500, 747)$, $(5000, 540)$ für $x = 3750$.
Schritt 1: Lagrangepolynome berechnen

$$l_0(3750) = \frac{(3750 - 2500)(3750 - 5000)}{(0 - 2500)(0 - 5000)} = \frac{1250 \cdot (-1250)}{(-2500) \cdot (-5000)} = -0.125$$

$$l_1(3750) = \frac{(3750 - 0)(3750 - 5000)}{(2500 - 0)(2500 - 5000)} = \frac{3750 \cdot (-1250)}{2500 \cdot (-2500)} = 0.75$$

$$l_2(3750) = \frac{(3750 - 0)(3750 - 2500)}{(5000 - 0)(5000 - 2500)} = \frac{3750 \cdot 1250}{5000 \cdot 2500} = 0.375$$

Schritt 2: Interpolationswert berechnen

$$P(3750) = 1013 \cdot (-0.125) + 747 \cdot 0.75 + 540 \cdot 0.375 = 636.0 \text{ hPa}$$

Splineinterpolation

Probleme der Polynominterpolation

Polynome mit hohem Grad oszillieren stark, besonders an den Rändern des Interpolationsintervalls. Für viele Stützpunkte ist Polynominterpolation daher ungeeignet.
Lösung: Spline-Interpolation verwendet stückweise kubische Polynome mit glatten Übergängen.

Natürliche kubische Splinefunktion

Eine natürliche kubische Splinefunktion $S(x)$ ist in jedem Intervall $[x_i, x_{i+1}]$ durch ein kubisches Polynom dargestellt:

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

mit den Randbedingungen $S_0''(x_0) = 0$ und $S_{n-1}''(x_n) = 0$.

Natürliche kubische Splinefunktion berechnen

Schritt 1: Parameter und Randbedingungen

Parameter initialisieren: $a_i = y_i$ und $h_i = x_{i+1} - x_i$
Randbedingungen setzen: $c_0 = 0$ und $c_n = 0$ (natürliche Spline).

Schritt 2: Gleichungssystem für c_i lösen

Für $i = 1, ..., n-1$:

$$h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} = 3\left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \right)$$

Schritt 3: Restliche Koeffizienten berechnen

$$b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{3}(c_{i+1} + 2c_i)$$

$$d_i = \frac{1}{3h_i}(c_{i+1} - c_i)$$

Kubische Splinefunktion berechnen

Aufgabe: Bestimmen Sie die natürliche kubische Splinefunktion für die Stützpunkte:

x_i	4	6	8	10
y_i	6	3	9	0

Lösung:
Schritt 1:
Parameter: $a_0 = 6, a_1 = 3, a_2 = 9$, $h_0 = h_1 = h_2 = 2$
Randbedingungen: $c_0 = 0, c_3 = 0$ (natürliche Spline)
Schritt 2: Gleichungssystem für c_1, c_2:

$$2 \cdot 8 \cdot c_1 + 2 \cdot c_2 = 3(3 - (-1.5)) = 13.5$$

$$2 \cdot c_1 + 2 \cdot 8 \cdot c_2 = 3((-4.5) - 3) = -22.5$$

Lösung: $c_1 = 1.2, c_2 = -1.8$
Schritt 3: Restliche Koeffizienten: $b_0 = -2.8, b_1 = 2.2, b_2 = -7.2$ $d_0 = 0.6, d_1 = -1.5, d_2 = 0.9$
Die Splinefunktionen sind: $S_0(x) = 6 - 2.8(x - 4) + 0.6(x - 4)^3$ $S_1(x) = 3 + 2.2(x - 6) + 1.2(x - 6)^2 - 1.5(x - 6)^3$ $S_2(x) = 9 - 7.2(x - 8) - 1.8(x - 8)^2 + 0.9(x - 8)^3$

Ausgleichsrechnung

Ausgleichsproblem

Gegeben sind n Wertepaare (x_i, y_i), $i = 1, ..., n$ mit $x_i \neq x_j$ für $i \neq j$. Gesucht ist eine stetige Funktion $f : \mathbb{R} \to \mathbb{R}$, die die Wertepaare in einem gewissen Sinn bestmöglich annähert, d.h. dass möglichst genau gilt:

$$f(x_i) \approx y_i \quad \text{für alle } i = 1, ..., n$$

Fehlerfunktional und kleinste Fehlerquadrate

Eine Ausgleichsfunktion f minimiert das **Fehlerfunktional**:

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

Man nennt das so gefundene f optimal im Sinne der **kleinsten Fehlerquadrate** (least squares fit).

Lineare Ausgleichsprobleme

Lineares Ausgleichsproblem

Gegeben seien n Wertepaare (x_i, y_i) und m Basisfunktionen $f_1, ..., f_m$. Die Ansatzfunktion hat die Form:

$$f(x) = \lambda_1 f_1(x) + \lambda_2 f_2(x) + \cdots + \lambda_m f_m(x)$$

Das Fehlerfunktional lautet:

$$E(f) = \|y - A\lambda\|_2^2$$

wobei A die $n \times m$ Matrix ist:

$$A = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_n) & f_2(x_n) & \cdots & f_m(x_n) \end{bmatrix}$$

Normalgleichungen

Die Lösung des linearen Ausgleichsproblems ergibt sich aus dem **Normalgleichungssystem**:

$$A^T A\lambda = A^T y$$

Für bessere numerische Stabilität sollte die QR-Zerlegung $A = QR$ verwendet werden:

$$R\lambda = Q^T y$$

Lineare Ausgleichsrechnung durchführen

Schritt 1: Ansatzfunktion festlegen

Bestimme die Basisfunktionen $f_1(x), f_2(x), ..., f_m(x)$.

Schritt 2: Matrix A aufstellen

Berechne $A_{ij} = f_j(x_i)$ für alle $i = 1, ..., n$ und $j = 1, ..., m$.

Schritt 3: Normalgleichungssystem aufstellen

Berechne $A^T A$ und $A^T y$.

Schritt 4: LGS lösen

Löse $A^T A\lambda = A^T y$ (bevorzugt mit QR-Zerlegung).

Schritt 5: Ausgleichsfunktion angeben

$f(x) = \lambda_1 f_1(x) + \lambda_2 f_2(x) + \cdots + \lambda_m f_m(x)$

Lineare Ausgleichsrechnung - Ausgleichsgerade

Bestimme die Ausgleichsgerade $f(x) = ax + b$ für die Datenpunkte:

x_i	1	2	3	4
y_i	6	6.8	10	10.5

Lösung: Basisfunktionen: $f_1(x) = x$, $f_2(x) = 1$

Schritt 1: Matrix A aufstellen

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}, \quad y = \begin{pmatrix} 6 \\ 6.8 \\ 10 \\ 10.5 \end{pmatrix}$$

Schritt 2: Normalgleichungen berechnen

$$A^T A = \begin{bmatrix} 30 & 10 \\ 10 & 4 \end{bmatrix}, \quad A^T y = \begin{pmatrix} 91.6 \\ 33.3 \end{pmatrix}$$

Schritt 3: LGS lösen: $\begin{bmatrix} 30 & 10 \\ 10 & 4 \end{bmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 91.6 \\ 33.3 \end{pmatrix}$

Lösung: $a = 1.67$, $b = 4.15$

Die Ausgleichsgerade lautet: $f(x) = 1.67x + 4.15$

Dichte von Wasser - Polynomfit

Fitte die Wasserdichte $\rho(T)$ mit einem Polynom 2. Grades $f(T) = aT^2 + bT + c$:

T [°C]	0	20	40	60	80	100
ρ [g/l]	999.9	998.2	992.2	983.2	971.8	958.4

Lösung: Basisfunktionen: $f_1(T) = T^2$, $f_2(T) = T$, $f_3(T) = 1$

Die Matrix A ist:

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 400 & 20 & 1 \\ 1600 & 40 & 1 \\ 3600 & 60 & 1 \\ 6400 & 80 & 1 \\ 10000 & 100 & 1 \end{bmatrix}$$

Nach Lösung des Normalgleichungssystems erhält man die Koeffizienten für die optimale Ausgleichsfunktion.

Nichtlineare Ausgleichsprobleme

Allgemeines Ausgleichsproblem

Gegeben seien n Wertepaare (x_i, y_i) und eine nichtlineare Ansatzfunktion $f_p(x, \lambda_1, ..., \lambda_m)$ mit m Parametern.

Das allgemeine Ausgleichsproblem besteht darin, die Parameter $\lambda_1, ..., \lambda_m$ zu bestimmen, so dass das Fehlerfunktional minimal wird:

$$E(\lambda) = \sum_{i=1}^{n} (y_i - f_p(x_i, \lambda_1, ..., \lambda_m))^2$$

Gauss-Newton-Verfahren

Das Gauss-Newton-Verfahren löst nichtlineare Ausgleichsprobleme durch Linearisierung:

Definiere $g(\lambda) := y - f(\lambda)$, dann ist das Problem äquivalent zur Minimierung von $\|g(\lambda)\|_2^2$.

In jeder Iteration wird $g(\lambda)$ linearisiert:

$$g(\lambda) \approx g(\lambda^{(k)}) + Dg(\lambda^{(k)}) \cdot (\lambda - \lambda^{(k)})$$

Gauss-Newton-Verfahren

Schritt 1: Funktionen definieren

$g(\lambda) := y - f(\lambda)$ und Jacobi-Matrix $Dg(\lambda)$ berechnen.

Schritt 2: Iterationsschleife

Für $k = 0, 1, ...$:

- Löse das lineare Ausgleichsproblem: $\min \|g(\lambda^{(k)}) + Dg(\lambda^{(k)}) \cdot \delta^{(k)}\|_2^2$
- Das ergibt: $Dg(\lambda^{(k)})^T Dg(\lambda^{(k)}) \delta^{(k)} = -Dg(\lambda^{(k)})^T g(\lambda^{(k)})$
- Setze $\lambda^{(k+1)} = \lambda^{(k)} + \delta^{(k)}$

Schritt 3: Dämpfung (optional)

Bei Konvergenzproblemen: $\lambda^{(k+1)} = \lambda^{(k)} + \frac{\delta^{(k)}}{2^p}$ mit geeignetem p.

Schritt 4: Konvergenzprüfung

Abbruch wenn $\|\delta^{(k)}\| < \text{TOL}$ oder $\|g(\lambda^{(k+1)})\| < \text{TOL}$.

Exponentialfunktion fitten Fitte die Funktion $f(x) = ae^{bx}$ an die Daten:

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

Methode 1: Linearisierung durch Logarithmieren

$$\ln f(x) = \ln(a) + bx$$

Setze $\tilde{y}_i = \ln(y_i)$ und löse lineares Problem.

Methode 2: Gauss-Newton-Verfahren $g(a, b) = y - f(a, b)$ mit $f_i(a, b) = ae^{bx_i}$

Jacobi-Matrix: $Dg(a, b) = \begin{bmatrix} -e^{bx_1} & -ax_1 e^{bx_1} \\ -e^{bx_2} & -ax_2 e^{bx_2} \\ \vdots & \vdots \\ -e^{bx_5} & -ax_5 e^{bx_5} \end{bmatrix}$

Mit Startvektor $(a^{(0)}, b^{(0)}) = (3, -1)$ konvergiert das Verfahren zu $a \approx 2.98$, $b \approx -1.00$.

Komplexere nichtlineare Funktion Fitte die Funktion

$$f(x) = \frac{\lambda_0 + \lambda_1 \cdot 10^{\lambda_2 + \lambda_3 x}}{1 + 10^{\lambda_2 + \lambda_3 x}}$$

an Datenpunkte mit dem gedämpften Gauss-Newton-Verfahren.

Lösung: Diese sigmoide Funktion erfordert:

- Sorgfältige Wahl des Startvektors
- Verwendung der Dämpfung für Stabilität
- Berechnung der komplexen Jacobi-Matrix mit partiellen Ableitungen
- Iterative Lösung bis zur gewünschten Genauigkeit

Das gedämpfte Gauss-Newton-Verfahren ist hier dem ungedämpften überlegen, da es auch bei schlechten Startwerten konvergiert.

Wahl zwischen linearer und nichtlinearer Ausgleichsrechnung:

- **Linear:** Wenn die Ansatzfunktion linear in den Parametern ist
- **Nichtlinear:** Wenn Parameter "verwoben"mit der Funktionsgleichung auftreten
- **Linearisierung:** Manchmal kann durch Transformation (z.B. Logarithmieren) ein nichtlineares Problem linearisiert werden
- **Stabilität:** Gedämpfte Verfahren sind robuster, aber aufwendiger

Numerische Integration

Numerische Integration (Quadratur)

Für eine Funktion $f : \mathbb{R} \to \mathbb{R}$ soll das bestimmte Integral

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

auf einem Intervall $[a, b]$ numerisch berechnet werden.
Quadraturverfahren haben im Allgemeinen die Form:

$$I(f) = \sum_{i=1}^{n} a_i f(x_i)$$

wobei die x_i die **Stützstellen** oder **Knoten** und die a_i die **Gewichte** der Quadraturformel sind.

Warum numerische Integration?

Im Gegensatz zur Ableitung können Integrale für viele Funktionen nicht analytisch gelöst werden. Beispiele:

- $\int e^{-x^2}dx$ (Gaußsche Fehlerfunktion)
- $\int \sin(x^2)dx$ (Fresnel-Integral)
- $\int \frac{\sin x}{x}dx$ (Integral-Sinus)

Numerische Verfahren sind daher essentiell für praktische Anwendungen.

Newton-Cotes Formeln

Rechteck- und Trapezregel

Einfache Rechteck- und Trapezregel

Die **Rechteckregel** (Mittelpunktsregel) und die **Trapezregel** zur Approximation von $\int_a^b f(x)dx$ sind definiert als:

Rechteckregel: $Rf = f\left(\frac{a+b}{2}\right) \cdot (b-a)$

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

Geometrische Interpretation

- **Rechteckregel:**
  Approximiert die Fläche durch ein Rechteck mit Höhe $f(\frac{a+b}{2})$
- **Trapezregel:** Approximiert die Fläche durch ein Trapez zwischen $(a, f(a))$ und $(b, f(b))$

Summierte Rechteck- und Trapezregel

Sei $f : [a, b] \to \mathbb{R}$ stetig, $n \in \mathbb{N}$ die Anzahl Subintervalle mit konstanter Breite $h = \frac{b-a}{n}$ und $x_i = a + ih$ für $i = 0, ..., n$.

Summierte Rechteckregel:

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

Summierte Trapezregel:

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

Summierte Trapezregel anwenden

Schritt 1: Parameter bestimmen

Gegeben: Intervall $[a, b]$, Anzahl Subintervalle n
Berechne: Schrittweite $h = \frac{b-a}{n}$

Schritt 2: Stützstellen berechnen

$x_i = a + ih$ für $i = 0, 1, ..., n$

Schritt 3: Funktionswerte berechnen

$f(x_i)$ für alle Stützstellen

Schritt 4: Trapezregel anwenden

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

Schritt 5: Für nicht-äquidistante Stützstellen

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

Trapezregel berechnen

Berechne $\int_2^4 \frac{1}{x}dx$ mit der summierten Trapezregel für $n = 4$.

Schritt 1: $h = \frac{4-2}{4} = 0.5$
Schritt 2: Stützstellen: $x_0 = 2, x_1 = 2.5, x_2 = 3, x_3 = 3.5, x_4 = 4$
Schritt 3: Funktionswerte:

$$f(2) = 0.5, f(2.5) = 0.4, f(3) = 0.333, f(3.5) = 0.286, f(4) = 0.25$$

Schritt 4: Trapezregel anwenden:

$$Tf(0.5) = 0.5 \cdot \left(\frac{0.5+0.25}{2} + 0.4 + 0.333 + 0.286\right) = 0.697$$

Vergleich:
Exakter Wert: $\ln(4) - \ln(2) = \ln(2) \approx 0.693$
Absoluter Fehler: $|0.697 - 0.693| = 0.004$

Simpson-Regel

Simpson-Regel

Die Simpson-Regel approximiert $f(x)$ durch ein Polynom 2. Grades an den Stellen $x_1 = a$, $x_2 = \frac{a+b}{2}$ und $x_3 = b$.
Einfache Simpson-Regel:

$$Sf = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Summierte Simpson-Regel:

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

Simpson-Regel als gewichtetes Mittel

Die summierte Simpson-Regel kann als gewichtetes Mittel der summierten Trapez- und Rechteckregel interpretiert werden:

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

Bewegung durch Flüssigkeit - Verschiedene Regeln

Aufgabe: Ein Teilchen mit Masse $m = 10$ kg bewegt sich durch eine Flüssigkeit mit Widerstand $R(v) = -v\sqrt{v}$. Für die Verlangsamung von $v_0 = 20$ m/s auf $v = 5$ m/s gilt:

$$t = \int_5^{20} \frac{m}{R(v)}dv = \int_5^{20} \frac{10}{-v\sqrt{v}}dv$$

Berechnen Sie das Integral mit $n = 5$ für:
a) Summierte Rechteckregel
b) Summierte Trapezregel
c) Summierte Simpson-Regel

Parametrisation: $h = \frac{20-5}{5} = 3$, Stützstellen: $5, 8, 11, 14, 17, 20$

a) Rechteckregel: $Rf(3) = 3 \cdot \sum_{i=0}^{4} f(x_i + 1.5)$ Mittelpunkte: $6.5, 9.5, 12.5, 15.5, 18.5$ $Rf(3) = 3 \cdot (-0.154 - 0.108 - 0.090 - 0.081 - 0.076) = -1.527$

b) Trapezregel: $Tf(3) = 3 \cdot \left(\frac{f(5)+f(20)}{2} + \sum_{i=1}^{4} f(x_i)\right)$ $Tf(3) = 3 \cdot \left(\frac{-0.179-0.056}{2} + (-0.125 - 0.096 - 0.082 - 0.072)\right) = -1.477$

c) Simpson-Regel: $Sf(3) = \frac{1}{3}(Tf(3) + 2Rf(3)) = \frac{1}{3}(-1.477 + 2(-1.527)) = -1.510$

Exakter Wert: $\int_5^{20} \frac{-10}{v^{3/2}}dv = \left[\frac{20}{\sqrt{v}}\right]_5^{20} = -1.506$

Absolute Fehler:

- Rechteckregel: $|-1.527 - (-1.506)| = 0.021$
- Trapezregel: $|-1.477 - (-1.506)| = 0.029$
- Simpson-Regel: $|-1.510 - (-1.506)| = 0.004$

Fehlerabschätzung

Fehlerabschätzung für summierte Quadraturformeln

Für genügend glatte Funktionen gelten folgende Fehlerabschätzungen:

Summierte Rechteckregel:

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

Summierte Trapezregel:

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

Summierte Simpson-Regel:

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

Schrittweite für gewünschte Genauigkeit bestimmen

Schritt 1: Gewünschte Genauigkeit festlegen

Maximaler absoluter Fehler: ϵ

Schritt 2: Höchste Ableitung abschätzen

Berechne $\max_{x\in[a,b]}|f^{(k)}(x)|$ für entsprechendes k.

Schritt 3: Schrittweite berechnen

Für Trapezregel:

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

Für Simpson-Regel:

$$h \leq \sqrt[4]{\frac{2880\epsilon}{(b-a)\max|f^{(4)}(x)|}}$$

Schritt 4: Anzahl Intervalle bestimmen

$n = \frac{b-a}{h}$ (aufrunden auf ganze Zahl)

Schrittweite für gewünschte Genauigkeit

Aufgabe: Bestimmen Sie die Schrittweite h, um $I = \int_0^{0.5} e^{-x^2}dx$ mit der summierten Trapezregel auf einen absoluten Fehler von maximal 10^{-5} genau zu berechnen.

Lösung:

Schritt 1: $\epsilon = 10^{-5}$, $a = 0$, $b = 0.5$

Schritt 2: Zweite Ableitung bestimmen: $f(x) = e^{-x^2}$ $f'(x) = -2xe^{-x^2}$ $f''(x) = -2e^{-x^2} + 4x^2e^{-x^2} = e^{-x^2}(4x^2-2)$

Auf $[0, 0.5]$: $\max|f''(x)| = \max|e^{-x^2}(4x^2-2)| = 2$ (bei $x = 0$)

Schritt 3: Schrittweite berechnen:

$$h \leq \sqrt{\frac{12\cdot 10^{-5}}{0.5\cdot 2}} = \sqrt{0.00012} \approx 0.011$$

Schritt 4: $n = \frac{0.5}{0.011} \approx 46$ Intervalle

Romberg-Extrapolation

Idee der Romberg-Extrapolation

Die Romberg-Extrapolation verbessert systematisch die Genauigkeit der Trapezregel durch Verwendung mehrerer Schrittweiten und anschließende Extrapolation.

Basis: Trapezregel mit halbierten Schrittweiten $h_j = \frac{b-a}{2^j}$ für $j = 0, 1, 2, ..., m$.

Romberg-Extrapolation

Für die summierte Trapezregel $Tf(h)$ 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.

Die verwendete Schrittweitenfolge $h_j = \frac{b-a}{2^j}$ heißt **Romberg-Folge**.

Romberg-Extrapolation durchführen

Schritt 1: Trapezwerte für erste Spalte berechnen

Berechne T_{j0} mit der summierten Trapezregel

für $h_j = \frac{b-a}{2^j}$, $j = 0, 1, ..., m$.

Schritt 2: Extrapolationsschema aufstellen

T_{00}			
T_{10}	T_{01}		
T_{20}	T_{11}	T_{02}	
T_{30}	T_{21}	T_{12}	T_{03}

Schritt 3: Rekursionsformel anwenden

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

Schritt 4: Genaueste Näherung

Der Wert rechts unten im Schema ist die genaueste Approximation.

Romberg-Extrapolation anwenden

Berechne $\int_0^{\pi}\cos(x^2)dx$ mit Romberg-Extrapolation für $m = 4$ (d.h. $j = 0, 1, 2, 3, 4$).

Schritt 1: Erste Spalte berechnen $T_{00} = Tf(\pi)$ mit $h_0 = \pi$ (1 Intervall) $T_{10} = Tf(\pi/2)$ mit $h_1 = \pi/2$ (2 Intervalle) $T_{20} = Tf(\pi/4)$ mit $h_2 = \pi/4$ (4 Intervalle) $T_{30} = Tf(\pi/8)$ mit $h_3 = \pi/8$ (8 Intervalle) $T_{40} = Tf(\pi/16)$ mit $h_4 = \pi/16$ (16 Intervalle)

Beispielrechnung für T_{00}:

$$T_{00} = \pi\cdot\frac{\cos(0)+\cos(\pi^2)}{2} = \frac{\pi}{2}(1+\cos(\pi^2))$$

Schritt 2: Extrapolationsschema:

T_{00}				
T_{10}	T_{01}			
T_{20}	T_{11}	T_{02}		
T_{30}	T_{21}	T_{12}	T_{03}	
T_{40}	T_{31}	T_{22}	T_{13}	T_{04}

Der Wert T_{04} liefert die beste Approximation des Integrals.

Gauss-Formeln

Optimale Stützstellen

Bei Newton-Cotes Formeln sind die Stützstellen äquidistant gewählt. Gauss-Formeln wählen sowohl Stützstellen x_i als auch Gewichte a_i optimal, um die Fehlerordnung zu maximieren.

Gauss-Formeln für $n = 1, 2, 3$

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

$n = 1$: $G_1 f = (b-a)\cdot f\left(\frac{b+a}{2}\right)$

$n = 2$: $G_2 f = \frac{b-a}{2}\left[f\left(-\frac{1}{\sqrt{3}}\cdot\frac{b-a}{2}+\frac{b+a}{2}\right) + f\left(\frac{1}{\sqrt{3}}\cdot\frac{b-a}{2}+\frac{b+a}{2}\right)\right]$

$n = 3$: $G_3 f = \frac{b-a}{2}\left[\frac{5}{9}f(x_1) + \frac{8}{9}f\left(\frac{b+a}{2}\right) + \frac{5}{9}f(x_3)\right]$

wobei $x_1 = -\sqrt{0.6}\cdot\frac{b-a}{2}+\frac{b+a}{2}$ und $x_3 = \sqrt{0.6}\cdot\frac{b-a}{2}+\frac{b+a}{2}$.

Erdmasse berechnen

Aufgabe: Berechnen Sie die Masse der Erde mit der nicht-äquidistanten Dichteverteilung:

$$m = \int_0^{6370}\rho(r)\cdot 4\pi r^2 dr$$

r [km]	0	800	1200	1400	2000	...
ρ [kg/m³]	13000	12900	12700	12000	11650	...

Lösung:

Da die Stützstellen nicht äquidistant sind, verwenden wir die summierte Trapezregel für nicht-äquidistante Daten:

$$\int_0^{6370}\rho(r)\cdot 4\pi r^2 dr \approx \sum_{i=0}^{n-1}\frac{[\rho(r_i)\cdot 4\pi r_i^2] + [\rho(r_{i+1})\cdot 4\pi r_{i+1}^2]}{2}\cdot(r_{i+1}-r_i)$$

Wichtig: Umrechnung der Einheiten: r in km $\rightarrow$ m, ρ in kg/m³

Ergebnis: $m_{Erde} \approx 5.94\times 10^{24}$ kg

Vergleich mit Literaturwert: 5.97×10^{24} kg Relativer Fehler: $\approx 0.5\%$

Wahl des Integrationsverfahrens:

- **Trapezregel:** Einfach, für glatte Funktionen ausreichend
- **Simpson-Regel:** Höhere Genauigkeit, besonders für polynomähnliche Funktionen
- **Romberg-Extrapolation:** Sehr hohe Genauigkeit mit systematischer Verbesserung
- **Gauss-Formeln:** Optimal für begrenzte Anzahl von Funktionsauswertungen
- **Nicht-äquidistante Daten:** Spezielle Trapezregel für tabellarische Daten

Einführung in gewöhnliche Differentialgleichungen

Gewöhnliche Differentialgleichung n-ter Ordnung

Eine Gleichung, in der Ableitungen einer unbekannten Funktion $y = y(x)$ bis zur n-ten Ordnung auftreten, heißt eine gewöhnliche Differentialgleichung n-ter Ordnung. Sie hat die explizite Form:

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

Gesucht sind die Lösungen $y = y(x)$ dieser Gleichung, wobei die Lösungen y auf einem Intervall $[a, b]$ definiert sein sollen.

Notationen für Ableitungen:

Lagrange: $y'(x), y''(x), y'''(x), y^{(4)}(x), ..., y^{(n)}(x)$

Newton: $\dot{y}(x), \ddot{y}(x), \dddot{y}(x), ...$

Leibniz: $\frac{dy}{dx}, \frac{d^2y}{dx^2}, \frac{d^3y}{dx^3}, ..., \frac{d^ny}{dx^n}$

Anfangswertproblem (AWP)

Bei einem Anfangswertproblem für eine Differentialgleichung n-ter Ordnung werden der Lösungsfunktion $y = y(x)$ noch n Werte vorgeschrieben:

DGL 1. Ordnung:
Gegeben ist $y'(x) = f(x, y(x))$ und der Anfangswert $y(x_0) = y_0$.

DGL 2. Ordnung: Gegeben ist $y''(x) = f(x, y(x), y'(x))$ und die Anfangswerte $y(x_0) = y_0$, $y'(x_0) = y'_0$.

Beispiele aus den Naturwissenschaften

Aufgabe: Klassifizieren Sie die folgenden DGL und geben Sie physikalische Interpretationen an.

1. Radioaktiver Zerfall:

$$\frac{dn}{dt} = -\lambda n$$

DGL 1. Ordnung, Lösung: $n(t) = n_0 e^{-\lambda t}$

2. Freier Fall:

$$\ddot{s}(t) = -g$$

DGL 2. Ordnung, Lösung: $s(t) = -\frac{1}{2}gt^2 + v_0 t + s_0$

3. Harmonische Schwingung (Federpendel):

$$m\ddot{x} = -cx \Rightarrow \ddot{x} + \frac{c}{m}x = 0$$

DGL 2. Ordnung, Lösung: $x(t) = A\sin(\omega_0 t + \varphi)$ mit $\omega_0 = \sqrt{\frac{c}{m}}$

Richtungsfelder

Geometrische Interpretation

Die DGL $y'(x) = f(x, y(x))$ gibt uns einen Zusammenhang zwischen der Steigung $y'(x)$ der gesuchten Funktion und dem Punkt $(x, y(x))$. Im Richtungsfeld wird an jedem Punkt (x, y) die Steigung $y'(x) = f(x, y)$ durch einen kleinen Pfeil dargestellt. Die Lösungskurven verlaufen stets tangential zu diesen Pfeilen.

Richtungsfeld zeichnen und interpretieren

Schritt 1: Steigungen berechnen

Für eine gegebene DGL $y' = f(x, y)$ berechne für verschiedene Punkte (x_i, y_j) die Steigung $f(x_i, y_j)$.

Schritt 2: Richtungspfeile einzeichnen

Zeichne an jedem Punkt (x_i, y_j) einen kleinen Pfeil mit der Steigung $f(x_i, y_j)$.

Schritt 3: Lösungskurven folgen

Von einem Anfangspunkt (x_0, y_0) ausgehend folge den Richtungspfeilen, um die Lösungskurve zu approximieren.

Schritt 4: Python-Implementierung

Verwende `numpy.meshgrid()` und `pyplot.quiver()` zur automatischen Darstellung.

Richtungsfeld interpretieren

Aufgabe: Zeichnen Sie das Richtungsfeld für $\frac{dy}{dt} = -\frac{1}{2} \cdot y \cdot t^2$ und bestimmen Sie die Lösungskurve für $y(0) = 3$.

Lösung:
Steigungen an ausgewählten Punkten:

$\frac{dy}{dt}$	$t = 0$	$t = 1$	$t = 2$	$t = 3$
$y = 0$	0	0	0	0
$y = 1$	0	-0.5	-2	-4.5
$y = 2$	0	-1	-4	-9
$y = 3$	0	-1.5	-6	-13.5

Die Lösungskurve für $y(0) = 3$ folgt den Richtungspfeilen und zeigt exponentiellen Abfall für $t > 0$.

Numerische Lösungsverfahren

Das Euler-Verfahren

Klassisches Euler-Verfahren

Gegeben sei das AWP $y' = f(x, y)$ mit $y(a) = y_0$ auf dem Intervall $[a, b]$.
Das Euler-Verfahren mit Schrittweite $h = \frac{b-a}{n}$ lautet:

$$x_{i+1} = x_i + h$$

$$y_{i+1} = y_i + h \cdot f(x_i, y_i)$$

wobei $x_0 = a$, $x_i = a + ih$ für $i = 0, ..., n-1$ und y_0 der gegebene Anfangswert ist.

Idee des Euler-Verfahrens

Das Euler-Verfahren folgt der Tangente im Punkt (x_i, y_i) mit der Steigung $f(x_i, y_i)$ um die Schrittweite h. Es ist das einfachste Einschrittverfahren mit Konvergenzordnung $p = 1$.

Euler-Verfahren anwenden

Schritt 1: Parameter bestimmen

Gegeben: AWP $y' = f(x, y)$, $y(a) = y_0$, Intervall $[a, b]$, Anzahl Schritte n Berechne: $h = \frac{b-a}{n}$

Schritt 2: Startwerte setzen

$x_0 = a$, $y_0 =$ gegebener Anfangswert

Schritt 3: Iteration

Für $i = 0, 1, ..., n-1$:

- Berechne $f(x_i, y_i)$
- Setze $x_{i+1} = x_i + h$
- Setze $y_{i+1} = y_i + h \cdot f(x_i, y_i)$

Schritt 4: Lösung interpretieren

Die Punkte (x_i, y_i) approximieren die Lösung $y(x)$ an den Stützstellen.

Euler-Verfahren berechnen

Aufgabe: Lösen Sie $\frac{dy}{dx} = \frac{x^2}{y}$ mit $y(0) = 2$ auf dem Intervall $[0, 1.4]$ mit $h = 0.7$ (Euler-Verfahren).

Lösung:
Parameter: $n = 2$, $h = 0.7$, $f(x, y) = \frac{x^2}{y}$
Iteration:

- $i = 0$: $x_0 = 0$, $y_0 = 2$
- $f(0, 2) = \frac{0^2}{2} = 0$
- $x_1 = 0 + 0.7 = 0.7$, $y_1 = 2 + 0.7 \cdot 0 = 2$
- $i = 1$: $x_1 = 0.7$, $y_1 = 2$
- $f(0.7, 2) = \frac{0.7^2}{2} = 0.245$
- $x_2 = 0.7 + 0.7 = 1.4$, $y_2 = 2 + 0.7 \cdot 0.245 = 2.1715$

Exakte Lösung: $y(x) = \sqrt{\frac{2x^3}{3} + 4}$ $y(1.4) = \sqrt{\frac{2 \cdot 1.4^3}{3} + 4} = 2.253$
Absoluter Fehler: $|2.253 - 2.1715| = 0.0815$

Verbesserte Euler-Verfahren

Mittelpunkt-Verfahren

Das Mittelpunkt-Verfahren berechnet die Steigung in der Mitte des Intervalls:

$$x_{h/2} = x_i + \frac{h}{2}$$

$$y_{h/2} = y_i + \frac{h}{2} \cdot f(x_i, y_i)$$

$$x_{i+1} = x_i + h$$

$$y_{i+1} = y_i + h \cdot f(x_{h/2}, y_{h/2})$$

Konvergenzordnung: $p = 2$

Modifiziertes Euler-Verfahren (Heun-Verfahren)

Das modifizierte Euler-Verfahren verwendet den Durchschnitt zweier Steigungen:

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + h, y_i + h \cdot k_1)$$

$$x_{i+1} = x_i + h$$

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

Konvergenzordnung: $p = 2$

Vergleich der Euler-Verfahren

Aufgabe: Lösen Sie das AWP aus dem vorigen Beispiel mit Mittelpunkt- und modifiziertem Euler-Verfahren. Vergleichen Sie die Genauigkeit.

Mittelpunkt-Verfahren:

- $x_{1/2} = 0.35$, $y_{1/2} = 2$, $f(0.35, 2) = 0.061$
- $y_1 = 2 + 0.7 \cdot 0.061 = 2.043$
- $x_{3/2} = 1.05$, $y_{3/2} = 2.128$, $f(1.05, 2.128) = 0.518$
- $y_2 = 2.043 + 0.7 \cdot 0.518 = 2.406$

Modifiziertes Euler-Verfahren:

- $k_1 = 0$, $k_2 = f(0.7, 2) = 0.245$
- $y_1 = 2 + 0.7 \cdot \frac{0+0.245}{2} = 2.086$
- $k_1 = 0.245$, $k_2 = f(1.4, 2.257) = 0.866$
- $y_2 = 2.086 + 0.7 \cdot \frac{0.245+0.866}{2} = 2.475$

Fehlervergleich bei $x = 1.4$:

- Exakt: $y(1.4) = 2.253$
- Euler: $|2.253 - 2.172| = 0.081$
- Mittelpunkt: $|2.253 - 2.406| = 0.153$
- Modifiziert: $|2.253 - 2.475| = 0.222$

Runge-Kutta Verfahren

Klassisches vierstufiges Runge-Kutta Verfahren

Das klassische Runge-Kutta Verfahren verwendet vier Steigungen und hat Konvergenzordnung $p = 4$:

$$k_1 = f(x_i, y_i), \quad k_2 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1)$$

$$k_3 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2), \quad k_4 = f(x_i + h, y_i + hk_3)$$

$$x_{i+1} = x_i + h, \quad y_{i+1} = y_i + h \cdot \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Butcher-Schema

Runge-Kutta Verfahren werden durch Butcher-Schemata charakterisiert:

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Interpretation: Die erste Spalte gibt die Stufen c_i, die zweite Spalte die Koeffizienten a_{ij} für die Steigungen k_j und die letzte Zeile die Gewichtung der Steigungen für die nächste Iteration an.

Runge-Kutta Verfahren anwenden

Schritt 1: Steigungen berechnen

$$k_1 = f(x_i, y_i), \quad k_2 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1)$$

$$k_3 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2), \quad k_4 = f(x_i + h, y_i + hk_3)$$

Schritt 2: Gewichtetes Mittel bilden

$$\text{Steigung} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Schritt 3: Nächsten Punkt berechnen

$$x_{i+1} = x_i + h, \quad y_{i+1} = y_i + h \cdot \text{Steigung}$$

Schritt 4: Iteration fortsetzen

Wiederhole bis zum Ende des Intervalls.

Runge-Kutta vs. andere Verfahren

Aufgabe: Lösen Sie $y' = 1 - \frac{y}{t}$ mit $y(1) = 5$ für $t \in [1, 6]$ mit $h = 0.01$ und vergleichen Sie mit der exakten Lösung $y(t) = \frac{t^2+9}{2t}$.

```python
def runge_kutta_4(f, a, b, n, y0):
    h = (b - a) / n
    x = a
    y = y0

    for i in range(n):
        k1 = f(x, y)
        k2 = f(x + h/2, y + h/2 * k1)
        k3 = f(x + h/2, y + h/2 * k2)
        k4 = f(x + h, y + h * k3)

        x += h
        y += h * (k1 + 2*k2 + 2*k3 + k4) / 6

    return x, y
```

Fehlervergleich bei $t = 6$:

- Exakt: $y(6) = 3.25$
- Euler: Fehler ≈ 0.1
- Runge-Kutta: Fehler $\approx 10^{-6}$

Systeme von Differentialgleichungen

DGL höherer Ordnung → System 1. Ordnung

Jede DGL n-ter Ordnung kann in ein System von n DGL 1. Ordnung umgewandelt werden durch Einführung von Hilfsvariablen für die Ableitungen.

DGL höherer Ordnung auf System 1. Ordnung zurückführen

Schritt 1: Nach höchster Ableitung auflösen

Bringe die DGL in die Form $y^{(n)} = f(x, y, y', ..., y^{(n-1)})$.

Schritt 2: Hilfsvariablen einführen

$$z_1(x) = y(x)$$
$$z_2(x) = y'(x)$$
$$z_3(x) = y''(x)$$
$$\vdots$$
$$z_n(x) = y^{(n-1)}(x)$$

Schritt 3: System aufstellen

$$z_1' = z_2$$
$$z_2' = z_3$$
$$\vdots$$
$$z_{n-1}' = z_n$$
$$z_n' = f(x, z_1, z_2, ..., z_n)$$

Schritt 4: Vektorielle Schreibweise

$$\mathbf{z}' = \mathbf{f}(x, \mathbf{z})$$

mit $\mathbf{z}(x_0) = \begin{pmatrix} y(x_0) \\ y'(x_0) \\ \vdots \\ y^{(n-1)}(x_0) \end{pmatrix}$

Landende Boeing - DGL 2. Ordnung

Aufgabe: Eine Boeing 737-200 landet mit $v_0 = 100$ m/s und erfährt die Bremskraft $F = -5\dot{x}^2 - 570000$. Die Bewegungsgleichung ist:

$$m\ddot{x} = -5\dot{x}^2 - 570000$$

mit $m = 97000$ kg. Formen Sie in ein System 1. Ordnung um.

Lösung:

Schritt 1: Nach $\ddot{x}$ auflösen:

$$\ddot{x} = \frac{-5\dot{x}^2 - 570000}{97000}$$

Schritt 2: Hilfsvariablen:

$$z_1(t) = x(t) \quad \text{(Position)}$$
$$z_2(t) = \dot{x}(t) = v(t) \quad \text{(Geschwindigkeit)}$$

Schritt 3: System 1. Ordnung:

$$z_1' = z_2$$
$$z_2' = \frac{-5z_2^2 - 570000}{97000}$$

Schritt 4: Anfangsbedingungen:

$$\mathbf{z}(0) = \begin{pmatrix} 0 \\ 100 \end{pmatrix}$$

Das System kann nun mit Runge-Kutta gelöst werden.

Raketenbewegung

Aufgabe: Die Bewegungsgleichung einer Rakete lautet:

$$a(t) = \ddot{h}(t) = v_{rel} \cdot \frac{\mu}{m_A - \mu \cdot t} - g$$

mit $v_{rel} = 2600$ m/s, $m_A = 300000$ kg, $m_E = 80000$ kg, $t_E = 190$ s. Berechnen Sie Geschwindigkeit und Höhe als Funktion der Zeit.

Lösung:

Parameter: $\mu = \frac{m_A - m_E}{t_E} = \frac{220000}{190} = 1158$ kg/s

System 1. Ordnung:

$$z_1' = z_2 \quad \text{(Höhe)}$$
$$z_2' = 2600 \cdot \frac{1158}{300000 - 1158t} - 9.81 \quad \text{(Geschwindigkeit)}$$

Anfangsbedingungen: $z_1(0) = 0$, $z_2(0) = 0$

Numerische Lösung mit Trapezregel:

$$v(t) = \int_0^t a(\tau)d\tau$$
$$h(t) = \int_0^t v(\tau)d\tau$$

Analytische Vergleichslösung:

$$v(t) = v_{rel} \ln\left(\frac{m_A}{m_A - \mu t}\right) - gt$$
$$h(t) = -\frac{v_{rel}(m_A - \mu t)}{\mu} \ln\left(\frac{m_A}{m_A - \mu t}\right) + v_{rel}t - \frac{1}{2}gt^2$$

Ergebnisse nach 190s:

- Geschwindigkeit: ≈ 2500 m/s
- Höhe: ≈ 180 km
- Beschleunigung: $\approx 2.5g$

Stabilität

Stabilitätsproblem

Bei der numerischen Lösung von DGL kann es vorkommen, dass der numerische Fehler unbeschränkt wächst, unabhängig von der Schrittweite h. Dies führt zu **instabilen** Lösungen.

Die Stabilität hängt ab von:

- Dem verwendeten Verfahren
- Der Schrittweite h
- Dem spezifischen Anfangswertproblem

Stabilitätsfunktion

Für die DGL $y' = -\alpha y$ (mit $\alpha > 0$) kann die numerische Lösung in der Form

$$y_{i+1} = g(h\alpha) \cdot y_i$$

geschrieben werden. Die Funktion $g(z)$ heißt **Stabilitätsfunktion** des Verfahrens.

Das offene Intervall $z \in (0, \alpha)$, in dem $|g(z)| < 1$ gilt, bezeichnet man als **Stabilitätsintervall**.

Stabilität des Euler-Verfahrens

Aufgabe: Untersuchen Sie die Stabilität des Euler-Verfahrens für $y' = -2.5y$ mit $y(0) = 1$.

Lösung:
Euler-Verfahren: $y_{i+1} = y_i - h \cdot 2.5y_i = y_i(1 - 2.5h)$
Stabilitätsfunktion: $g(z) = 1 - z$ mit $z = 2.5h$
Stabilitätsbedingung: $|1 - 2.5h| < 1 \Rightarrow 0 < 2.5h < 2 \Rightarrow 0 < h < 0.8$
Verhalten:
- $h = 0.2$: Stabile Lösung (exponentieller Abfall)
- $h = 0.85$: Instabile Lösung (Oszillation mit wachsender Amplitude)

Exakte Lösung: $y(x) = e^{-2.5x}$ (streng monoton fallend)

Praktische Hinweise zur Stabilität:
- **Schrittweiten-Kontrolle:** Beginne mit kleiner Schrittweite und prüfe Konvergenz
- **Verfahrensvergleich:** Teste verschiedene Verfahren und vergleiche Ergebnisse
- **Analytische Kontrolle:** Vergleiche mit bekannten analytischen Lösungen
- **Steife DGL:** Verwende implizite Verfahren für steife Probleme
- **Python-Tools:** Nutze `scipy.integrate.solve_ivp()` für robuste Implementierungen

Python-Implementierung

DGL-Löser in Python

Standard-Bibliothek für DGL-Probleme:

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

def f(t, y):
    # DGL: y' = -2*y + sin(t)
    return -2*y + np.sin(t)

# Anfangswertproblem loesen
sol = solve_ivp(f, [0, 5], [1], dense_output=True)

# Losung plotten
t = np.linspace(0, 5, 100)
y = sol.sol(t)
plt.plot(t, y[0])
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Numerische Loesung der DGL')
plt.show()
```

Verfügbare Methoden:
- `'RK45'`: Runge-Kutta 5(4) (Standard)
- `'RK23'`: Runge-Kutta 3(2)
- `'DOP853'`: Runge-Kutta 8. Ordnung
- `'Radau'`: Implizites Runge-Kutta
- `'BDF'`: Backward Differentiation (für steife DGL)
- `'LSODA'`: Automatische Steifigkeits-Erkennung

Eigene DGL-Löser implementieren

Schritt 1: Grundstruktur

```python
def runge_kutta_4(f, a, b, n, y0):
    h = (b - a) / n
    x = np.linspace(a, b, n+1)
    y = np.zeros(n+1)
    y[0] = y0

    for i in range(n):
        k1 = f(x[i], y[i])
        k2 = f(x[i] + h/2, y[i] + h/2 * k1)
        k3 = f(x[i] + h/2, y[i] + h/2 * k2)
        k4 = f(x[i] + h, y[i] + h * k3)

        y[i+1] = y[i] + h/6 * (k1 + 2*k2 + 2*k3 + k4)

    return x, y
```

Schritt 2: Für Systeme erweitern

```python
def runge_kutta_system(f, a, b, n, y0):
    h = (b - a) / n
    x = np.linspace(a, b, n+1)
    y = np.zeros((n+1, len(y0)))
    y[0] = y0

    for i in range(n):
        k1 = f(x[i], y[i])
        k2 = f(x[i] + h/2, y[i] + h/2 * k1)
        k3 = f(x[i] + h/2, y[i] + h/2 * k2)
        k4 = f(x[i] + h, y[i] + h * k3)

        y[i+1] = y[i] + h/6 * (k1 + 2*k2 + 2*k3 + k4)

    return x, y
```

Schritt 3: Richtungsfeld visualisieren

```python
def plot_direction_field(f, xmin, xmax, ymin, ymax,
    hx, hy):
    x = np.arange(xmin, xmax, hx)
    y = np.arange(ymin, ymax, hy)
    X, Y = np.meshgrid(x, y)

    DX = np.ones_like(X)
    DY = f(X, Y)

    plt.quiver(X, Y, DX, DY, alpha=0.6)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Richtungsfeld')
```

Fehlerordnung und Konvergenz

Lokaler und globaler Fehler

Lokaler Fehler: Der Fehler nach einer Iteration $\varphi(x_i, h) := y(x_{i+1}) - y_{i+1}$
Globaler Fehler: Der Fehler nach n Iterationen $y(x_n) - y_n$
Konsistenzordnung p: Ein Verfahren hat Konsistenzordnung p, falls $|\varphi(x_i, h)| \leq C \cdot h^{p+1}$
Konvergenzordnung p: Ein Verfahren hat Konvergenzordnung p, falls $|y(x_n) - y_n| \leq C \cdot h^p$

Konvergenzordnungen der Verfahren

- **Euler-Verfahren:** Konvergenzordnung $p = 1$
- **Mittelpunkt-Verfahren:** Konvergenzordnung $p = 2$
- **Modifiziertes Euler-Verfahren:** Konvergenzordnung $p = 2$
- **Klassisches Runge-Kutta:** Konvergenzordnung $p = 4$

Praktische Bedeutung: Bei Halbierung der Schrittweite h reduziert sich der Fehler um den Faktor 2^p.

Konvergenzverhalten untersuchen

Aufgabe: Untersuchen Sie das Konvergenzverhalten verschiedener Verfahren für $\frac{dy}{dx} = \frac{x^2}{y}$ mit $y(0) = 2$ auf $[0, 10]$.

Lösung:

Exakte Lösung: $y(x) = \sqrt{\frac{2x^3}{3} + 4}$

Fehler bei $x = 10$ für verschiedene h:

h	Euler	Mittelpunkt	Mod. Euler	Runge-Kutta
0.1	10^{-1}	10^{-2}	10^{-2}	10^{-5}
0.05	5×10^{-2}	2.5×10^{-3}	2.5×10^{-3}	6×10^{-7}
0.025	2.5×10^{-2}	6×10^{-4}	6×10^{-4}	4×10^{-8}

Beobachtung: Bei Halbierung von h:
- Euler: Fehler halbiert sich (Ordnung 1)
- Mittelpunkt/Mod. Euler: Fehler viertelt sich (Ordnung 2)
- Runge-Kutta: Fehler wird um Faktor 16 kleiner (Ordnung 4)

Spezielle Anwendungen

Schwingungsgleichung - Gekoppeltes System

Aufgabe: Lösen Sie die Schwingungsgleichung $\ddot{x} + \omega^2 x = 0$ mit $x(0) = 1$, $\dot{x}(0) = 0$ und $\omega = 2$.

Lösung:
System 1. Ordnung: $z_1' = z_2$ $z_2' = -\omega^2 z_1 = -4z_1$
Anfangsbedingungen: $z_1(0) = 1$, $z_2(0) = 0$
Analytische Lösung: $x(t) = \cos(2t)$
Numerische Implementierung:

```python
def harmonic_oscillator(t, z):
    # z[0] = x, z[1] = dx/dt
    return [z[1], -4*z[0]]

# Losung mit scipy
sol = solve_ivp(harmonic_oscillator, [0, 10], [1, 0],
                method='RK45', dense_output=True)

t = np.linspace(0, 10, 1000)
z = sol.sol(t)
x_num = z[0]
x_exact = np.cos(2*t)

plt.plot(t, x_num, 'b-', label='Numerisch')
plt.plot(t, x_exact, 'r--', label='Exakt')
plt.legend()
```

Energieerhaltung prüfen: $E = \frac{1}{2}\dot{x}^2 + \frac{1}{2}\omega^2 x^2 = \text{const}$

Populationsdynamik - Logistisches Wachstum

Aufgabe: Das logistische Wachstumsmodell lautet: $\frac{dP}{dt} = rP\left(1 - \frac{P}{K}\right)$ mit Wachstumsrate $r = 0.1$ und Kapazität $K = 1000$. Anfangspopulation: $P(0) = 50$.

Lösung:

Analytische Lösung: $P(t) = \frac{K}{1+\left(\frac{K}{P_0}-1\right)e^{-rt}}$

Numerische Lösung:

```python
def logistic_growth(t, P, r=0.1, K=1000):
    return r * P * (1 - P/K)

# Parameter
r, K, P0 = 0.1, 1000, 50

# Numerische Loesung
sol = solve_ivp(lambda t, P: logistic_growth(t, P, r, K),
                [0, 100], [P0], dense_output=True)

# Analytische Loesung
def P_exact(t):
    return K / (1 + (K/P0 - 1) * np.exp(-r*t))

t = np.linspace(0, 100, 1000)
P_num = sol.sol(t)[0]
P_ana = P_exact(t)

plt.plot(t, P_num, 'b-', label='Numerisch')
plt.plot(t, P_ana, 'r--', label='Analytisch')
plt.axhline(y=K, color='k', linestyle=':', label='Kapazitaet K')
plt.xlabel('Zeit t')
plt.ylabel('Population P(t)')
plt.legend()
```

Charakteristisches Verhalten: Exponentielles Wachstum für kleine P, Sättigung bei K.

Erweiterte Themen

Mehrschrittverfahren

Im Gegensatz zu Einschrittverfahren verwenden Mehrschrittverfahren mehrere vorhergehende Punkte:
Adams-Bashforth 2. Ordnung: $y_{i+1} = y_i + \frac{h}{2}(3f(x_i, y_i) - f(x_{i-1}, y_{i-1}))$
Adams-Bashforth 3. Ordnung: $y_{i+1} = y_i + \frac{h}{12}(23f(x_i, y_i) - 16f(x_{i-1}, y_{i-1}) + 5f(x_{i-2}, y_{i-2}))$
Vorteil: Effizienter (weniger Funktionsauswertungen pro Schritt) **Nachteil:** Benötigen Startwerte von Einschrittverfahren

Implizite Verfahren

Implizite Verfahren sind stabiler, aber aufwendiger zu berechnen:
Implizites Euler-Verfahren: $y_{i+1} = y_i + h \cdot f(x_{i+1}, y_{i+1})$
Erfordert Lösung einer nichtlinearen Gleichung in jedem Schritt (z.B. mit Newton-Verfahren).
Anwendung: Steife Differentialgleichungen, bei denen explizite Verfahren sehr kleine Schrittweiten erfordern.

Steife Differentialgleichungen

Steife DGL haben stark unterschiedliche Zeitskalen und erfordern spezielle Behandlung:
Beispiel: $y' = -1000y + \sin(x)$
Die exakte Lösung hat sowohl schnell abklingende (e^{-1000x}) als auch langsam variierende ($\sin(x)$) Komponenten.
Problem: Explizite Verfahren benötigen $h < \frac{2}{1000}$ für Stabilität, auch wenn nur die langsame Komponente interessiert.
Lösung: Implizite Verfahren (BDF, Radau) oder adaptive Verfahren mit Steifigkeitserkennung.

Verfahren auswählen

Schritt 1: Problemanalyse

- Ist die DGL steif? → Implizite Verfahren
- Hohe Genauigkeit erforderlich? → Runge-Kutta höherer Ordnung
- Lange Zeitintegrationen? → Adaptive Schrittweiten
- Einfache Probleme? → RK4 oder scipy.solve_ivp

Schritt 2: Implementierungsstrategie

- Beginne mit Standard-Verfahren (RK4)
- Teste Konvergenz durch Schrittweiten-Variation
- Vergleiche mit analytischen Lösungen (falls verfügbar)
- Bei Instabilität: Kleinere Schrittweiten oder andere Verfahren

Schritt 3: Qualitätskontrolle

- Energieerhaltung bei konservativen Systemen
- Monotonie-Eigenschaften
- Langzeit-Stabilität
- Vergleich verschiedener Verfahren

Zusammenfassung - Wann welches Verfahren?

- **Euler:** Einfachste Implementierung, Lernzwecke, grobe Näherungen
- **Mittelpunkt/Modifiziert:** Bessere Genauigkeit als Euler, moderater Aufwand
- **Runge-Kutta 4:** Standard für die meisten Probleme, gute Balance zwischen Genauigkeit und Aufwand
- **Adaptive Verfahren:** Komplexe Probleme, automatische Schrittweitenkontrolle
- **Implizite Verfahren:** Steife Probleme, Langzeit-Stabilität
- **Spezialisierte Methoden:** Symplektische Integratoren für Hamiltonsche Systeme, geometrische Integratoren

Praktischer Tipp: Verwende `scipy.integrate.solve_ivp()` mit automatischer Methodenwahl für die meisten Anwendungen.