Höhere Mathematik 2

Jil Zerndt
FS 2025

Numerische Lösung nicht linearer Gleichungssysteme

LGS = lineares Gleichungssystem, NGS = nichtlineares Gleichungssystem

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

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

f mit n unabhängigen Variablen $x_1, \ldots, x_n$ und einer abhängigen Variablen y, die jedem $(x_1, x_2, \ldots, x_n)$ aus Definitionsmenge $D \subset \mathbb{R}^n$ genau ein $y \in W \subset \mathbb{R}$ zuordnet. Ergebnis: $y \in \mathbb{R}$ = Skalar (eine Zahl)

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

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

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.

Nichtlineares Gleichungssystem (NGS)
Lösungen des NGS sind Nullstellen der Funktion:

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

Ein solches System lässt sich nicht in die Form $Ax = b$ bringen.
Geometrisch lassen sich die Lösungen als Schnittpunkte der beiden Funktionen interpretieren.

Lineare Funktionen von LGS

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

$$\vec{f}(\vec{x}) = \mathbf{A}\vec{x} - \vec{b} = \begin{pmatrix} 4 & -1 & 1 \\ -2 & 5 & 1 \\ 1 & -2 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - \begin{pmatrix} 5 \\ 11 \\ 12 \end{pmatrix}, \quad \mathbf{f}(x_1, x_2, x_3) = \begin{pmatrix} f_1 = 4x_1 - x_2 + x_3 - 5 \\ f_2 = -2x_1 + 5x_2 + x_3 - 11 \\ f_3 = x_1 - 2x_2 + 5x_3 - 12 \end{pmatrix}$$

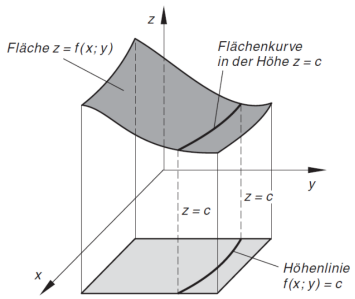
Analytische Darstellung
- **Explizite Darstellung:** $y = f(x_1, \ldots, x_n)$
- **Implizite Darstellung:** $F(x, y) = 0$
- **Parameterdarstellung:** $x = x(t), y = y(t)$

Darstellung durch Wertetabelle Sei $f : \mathbb{R}^n \to \mathbb{R}^m$ eine Funktion.
In $z = f(x, y)$ Werte von x und y einsetzen (der Reihe nach):

$$\begin{pmatrix} z_{11} & z_{12} & \cdots & z_{1m} \\ z_{m1} & z_{m2} & \cdots & z_{mn} \end{pmatrix}$$

Funktion als Fläche im Raum
f ordnet jedem Punkt $(x, y) \in D$ in Ebene Wert $z = f(x, y)$ zu ($\to$ Höhenkoordinate)

Schnittkurvendiagramm
Fläche $z = f(x, y)$ bei konstanten Höhe z schneiden: Schnittkurve. Diese in (x, y)-Ebene projizieren: Höhenlinie.



hellgraue Fläche = Definitionsbereich D

Partielle Ableitungen

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

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

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: nach welcher Variable ableiten?
2. Alle anderen Variablen während Ableitung nur Konstanten
3. Standardableitungsregeln anwenden und Ergebnis korrekt notieren

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

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

Linearisierung Die verallgemeinerte **Tangentengleichung**

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

beschreibt lineare Funktion, $f(x) \approx g(x)$ in Umgebung von $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^T \in \mathbb{R}^n$. Man spricht von der **Linearisierung** der Funktion $y = f(x)$ in einer Umgebung von $x^{(0)}$ ($x^{(k)}$ bezeichnet Vektor aus $\mathbb{R}^n$ nach k-ter Iteration).

Tangentialebene $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$, $y = f(x_1, x_2)$, $\boldsymbol{x}^{(0)} = (x_1^{(0)}, x_2^{(0)})^T \in \mathbb{R}^2$
Spezielle Jacobi-Matrix (nur ein Zeilenvektor mit zwei Elementen):

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

Linearisierung $g(x_1, x_2)$ die Gleichung der Tangentialebene:

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

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

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

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 partielle Ableitungen $\frac{\partial f_i}{\partial x_j}$ für $i = 1, ..., m$, $j = 1, ..., n$.
3. Stelle die Jacobi-Matrix $Df(x)$ auf
4. Werte Jacobi-Matrix an Entwicklungspunkt $x^{(0)}$ aus (Werte für $x_1, x_2, ..., x_n$ einsetzen)
5. Berechne Linearisierung $g(x)$ mit Tangentengleichung

Struggle $\in \mathbb{R}$

Jacobi-Matrix und Linearisierung $f(x, y, z) = \begin{pmatrix} e^{xy} + z^2 - 3 \\ \sin(x + y) - z \\ x^2 + y^2 + z^2 - 6 \end{pmatrix}$

Jacobi-Matrix: $Df(x, y, z) = \begin{bmatrix} ye^{xy} & xe^{xy} & 2z \\ \cos(x+y) & \cos(x+y) & -1 \\ 2x & 2y & 2z \end{bmatrix}$

Linearisierung an $(1, 0, 1)^T$:

$$f(1, 0, 1) = \begin{pmatrix} e^0 + 1 - 3 \\ \sin(1) - 1 \\ 1 + 0 + 1 - 6 \end{pmatrix} = \begin{pmatrix} -1 \\ \sin(1) - 1 \\ -4 \end{pmatrix}, \quad Df(1, 0, 1) = \begin{bmatrix} 0 & 1 & 2 \\ \cos(1) & \cos(1) & -1 \\ 2 & 0 & 2 \end{bmatrix}$$

Linearisierung: $g(x, y, z) = f(1, 0, 1) + Df(1, 0, 1) \cdot \begin{pmatrix} x - 1 \\ y - 0 \\ z - 1 \end{pmatrix}$

$$g(x, y, z) = \begin{pmatrix} -1 + y + 2(z - 1) \\ \sin(1) - 1 + \cos(1)(x - 1) + \cos(1)y - (z - 1) \\ -4 + 2(x - 1) + 2(z - 1) \end{pmatrix}$$

Geometrische Bedeutung: Linearisierung approximiert nichtlineare Funktion f nahe des Punktes $(1, 0, 1)^T$ durch lineare Funktion. Entspricht der Tangentialebene an die durch $f = 0$ definierte Fläche im 3D Raum.

Nullstellenbestimmung für NGS

Problemstellung zur Nullstellenbestimmung
Gegeben: $n \in \mathbb{N}$, $f : \mathbb{R}^n \to \mathbb{R}^n$ Gesucht: Vektor $\bar{x} \in \mathbb{R}^n$ mit $f(\bar{x}) = 0$
Komponentenweise: Gegeben: n Funktionen $f_i : \mathbb{R}^n \to \mathbb{R}$ (Komponenten von f) Gesucht: Vektor $\bar{x} \in \mathbb{R}^n$ mit $f_i(\bar{x}) = 0$ für $i = 1, ..., n$.

Newton-Verfahren für NGS (Quadratische Konv.)
Gesucht: Nullstellen von $f : \mathbb{R}^n \to \mathbb{R}^n$
$x^{(0)}$ = Startvektor nahe einer Nullstelle
Vorbereitung: definiere $f(x) = 0$, berechne $Df(x)$, wähle $x^{(0)}$
Für jede Iteration n:
1. Linearisierung um x^n: Berechne $f(x^{(n)})$ und $Df(x^{(n)})$
2. Nullstellen der Linearisierung: $\delta^{(n)}$ als Lösung des LGS $Df(x^{(n)}) \cdot \delta^{(n)} = -f(x^{(n)})$
3. Setze $x^{(n+1)} := x^{(n)} + \delta^{(n)}$ (nächste Iteration)
4. Weiterführen bis: $\|f(x^{(n+1)})\|_2 < \text{TOL}$ oder $\|x^{(n+1)} - x^{(n)}\|_2 < \text{TOL}$.

Interpretation: Konvergierte Lösung $x^{(n)}$ = Näherung für Nullstelle von f

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 LGS $Df(x^{(0)}) \cdot \delta^{(n)} = -f(x^{(n)})$
3. Setze $x^{(n+1)} := x^{(n)} + \delta^{(n)}$
4. Weiterführen bis: $\|f(x^{(n+1)})\|_2 < \text{TOL}$ oder $\|x^{(n+1)} - x^{(n)}\|_2 < \text{TOL}$.

Fehler-Normen
- $\|f(x)\|_2 = \sqrt{\sum_{i=1}^n f_i(x)^2}$: Euklidische Norm
- $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$: Euklidische Norm für Vektoren
- $\|A\|_2 = \max_{\|x\|_2 = 1} \|Ax\|_2$: Operatornorm für Matrizen

Newton-Verfahren $f(x_1, x_2) = \begin{pmatrix} 20 - 18x_1 - 2x_2^2 \\ -4x_2(x_1 - x_2^2) \end{pmatrix}$, $x^{(0)} = (1.1, 0.9)^T$

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

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.

Gedämpftes Newton-Verfahren

Dämpfung für bessere Konvergenz Für schlecht konditionierte Jacobi-Matrix $Df(x^{(n)})$ kann Standard Newton-Verfahren divergieren.
Dämpfung = variable Schrittweite: $x^{(n+1)} = x^{(n)} + \frac{\delta^{(n)}}{2^p}$
p = kleinstes Element aus $\{0, 1, ..., p_{\max}\}$ für das gilt:
$\|f(x^{(n)} + \frac{\delta^{(n)}}{2^p})\|_2 < \|f(x^{(n)})\|_2$

Gedämpftes Newton-Verfahren
Nur in der Nähe der Nullstelle ist Konvergenz des Verfahrens garantiert!
1. Berechne $f(x^{(n)})$ und $Df(x^{(n)})$
2. Berechne $\delta^{(n)}$ als Lösung des lin. GS $Df(x^{(n)}) \cdot \delta^{(n)} = -f(x^{(n)})$
3. Finde das minimale $p \in \{0, 1, \dots, p_{\max}\}$ mit:
$$\|f(x^{(n)} + \frac{\delta^{(n)}}{2^k})\|_2 < \|f(x^{(n)})\|_2$$
Kein minimales k gefunden $\rightarrow k = 0$
4. Setze $x^{(n+1)} := x^{(n)} + \frac{\delta^{(n)}}{2^k}$

Ausgleichsrechnung

Ausgleichsproblem ('Polyfit')
Gegeben: n Wertepaare (x_i, y_i), $i = 1, ..., n$ mit $x_i \neq x_j$ für $i \neq j$.
Gesucht: stetige Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$, die die Wertepaare bestmöglich annähert (es soll gelten) $f(x_i) \approx y_i$ für alle $i = 1, ..., n$

Fehlerfunktional und kleinste Fehlerquadrate
Eine Ausgleichsfunktion f minimiert das **Fehlerfunktional**:
$$E(f) := \|y - f(x)\|_2^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2$$
Gefundenes f optimal im Sinne der **kleinsten Fehlerquadrate** (least squares fit).

Lineare Ausgleichsprobleme

Lineares Ausgleichsproblem
Gegeben: n Wertepaare (x_i, y_i) und m Basisfunktionen $f_1, ..., f_m$
Ansatzfunktion: $f(x) = \lambda_1 f_1(x) + \lambda_2 f_2(x) + \cdots + \lambda_m f_m(x)$
Fehlerfunktional: $E(f) = \|y - A\lambda\|_2^2$
wobei A die $n \times m$ Matrix ist: $A = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_m(x_2) \\ \cdots & \cdots & \cdots & \cdots \\ f_1(x_n) & f_2(x_n) & \cdots & f_m(x_n) \end{bmatrix}$

Normalgleichungen Die Lösung des linearen Ausgleichsproblems ergibt sich aus dem **Normalgleichungssystem**: $A^T A\lambda = A^T y$
Nach Lösung des Normalgleichungssystems erhält man die Koeffizienten für die optimale Ausgleichsfunktion.

Lineare Ausgleichsrechnung durchführen
- **Basisfunktionen bestimmen** $f_1(x), f_2(x), ..., f_m(x)$
- **Matrix** A Berechne $A_{ij} = f_j(x_i)\ \forall\ i = 1, ..., n$ und $j = 1, ..., m$
- **Normalgleichungssystem** Berechne $A^T A$ und $A^T y$
- **LGS lösen** Löse $A^T A\lambda = A^T y$
- **Ausgleichsfunktion** $f(x) = \lambda_1 f_1(x) + \lambda_2 f_2(x) + \cdots + \lambda_m f_m(x)$
- **Fehlerfunktional** Berechne $E(f) = \|y - A\lambda\|_2^2$
- **Konvergenz** Prüfe, ob $E(f)$ klein genug ist (z.B. $\leq 10^{-6}$)

Lineare Ausgleichsrechnung

Ausgleichsgerade $f(x) = ax + b$ für:

x_i	1	2	3	4
y_i	6	6.8	10	10.5

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

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

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

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: $f(x) = 1.67x + 4.15$

Residuen berechnen: $r_i = y_i - f(x_i)$

i	y_i	$f(x_i)$	r_i
1	6	5.82	0.18
2	6.8	7.49	-0.69
3	10	9.16	0.84
4	10.5	10.83	-0.33

Residuenvektor: $r = \begin{pmatrix} 0.18 \\ -0.69 \\ 0.84 \\ -0.33 \end{pmatrix}$

Residuenquadrate:
$r_i^2 = 0.0324, 0.4761, 0.7056, 0.1089$

Fehlerfunktional: (Summe der Residuenquadrate)
$E(f) = \|y - A\lambda\|_2^2 = \sum_{i=1}^{n} r_i^2 = 1.323$

Nichtlineare Ausgleichsprobleme

Allgemeines Ausgleichsproblem Gegeben: n Wertepaare (x_i, y_i) und nichtlineare Ansatzfunktion $f_p(x, \lambda_1, ..., \lambda_m)$ mit m Parametern
Allgemeines Ausgleichsproblem: bestimme Parameter $\lambda_1, ..., \lambda_m$ 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
löst nichtlineare Ausgleichsprobleme durch Linearisierung:
$$g(\lambda) := y - f(\lambda)$$
$\rightarrow$ Problem äquivalent zur Minimierung von $\|g(\lambda)\|_2^2$.
In jeder Iteration $g(\lambda)$ linearisieren:
$$g(\lambda) \approx g(\lambda^{(k)}) + Dg(\lambda^{(k)}) \cdot (\lambda - \lambda^{(k)})$$

Gauss-Newton-Verfahren
Funktionen definieren $g(\lambda) := y - f(\lambda)$ und $Dg(\lambda)$ berechnen
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 $\lambda^{(k+1)} = \lambda^{(k)} + \delta^{(k)}$

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

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

Wahl zwischen linearer und nichtlinearer Ausgleichsrechnung:
- **Linear:** Wenn die Ansatzfunktion linear in den Parametern ist
- **Nichtlinear:** Wenn Parameter "verwoben"mit der Funktionsgleichung
- **Stabilität:** Gedämpfte Verfahren sind robuster, aber aufwendiger

Gauss-Newton-Verfahren

Aufgabe: Fitten Sie die Funktion $f(x) = a \cdot e^{-bx} + c$ an die Datenpunkte:

x	0	1	2	3
y	5.2	3.8	3.1	2.9

Funktionen definieren: $g(\lambda) = y - f(\lambda) = \begin{pmatrix} 5.2 \\ 3.8 \\ 3.1 \\ 2.9 \end{pmatrix} - \begin{pmatrix} ae^{-b\cdot 0} + c \\ ae^{-b\cdot 1} + c \\ ae^{-b\cdot 2} + c \\ ae^{-b\cdot 3} + c \end{pmatrix}$

Jacobi-Matrix von g: $\frac{\partial g_i}{\partial a} = -e^{-bx_i}$, $\frac{\partial g_i}{\partial b} = ax_i e^{-bx_i}$, $\frac{\partial g_i}{\partial c} = -1$

$$Dg(\lambda) = \begin{bmatrix} -e^{-b\cdot 0} & a\cdot 0 \cdot e^{-b\cdot 0} & -1 \\ -e^{-b\cdot 1} & a\cdot 1 \cdot e^{-b\cdot 1} & -1 \\ -e^{-b\cdot 2} & a\cdot 2 \cdot e^{-b\cdot 2} & -1 \\ -e^{-b\cdot 3} & a\cdot 3 \cdot e^{-b\cdot 3} & -1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & -1 \\ -e^{-b} & ae^{-b} & -1 \\ -e^{-2b} & 2ae^{-2b} & -1 \\ -e^{-3b} & 3ae^{-3b} & -1 \end{bmatrix}$$

Gauss-Newton-Schritt mit $\lambda^{(0)} = (2, 0.5, 2.5)^T$**:**

$$f(\lambda^{(0)}) = \begin{pmatrix} 2+2.5 \\ 2e^{-0.5}+2.5 \\ 2e^{-1}+2.5 \\ 2e^{-1.5}+2.5 \end{pmatrix} = \begin{pmatrix} 4.5 \\ 3.71 \\ 3.24 \\ 2.95 \end{pmatrix}$$

$$g(\lambda^{(0)}) = \begin{pmatrix} 5.2 \\ 3.8 \\ 3.1 \\ 2.9 \end{pmatrix} - \begin{pmatrix} 4.5 \\ 3.71 \\ 3.24 \\ 2.95 \end{pmatrix} = \begin{pmatrix} 0.7 \\ 0.09 \\ -0.14 \\ -0.05 \end{pmatrix}$$

$$Dg(\lambda^{(0)}) = \begin{bmatrix} -1 & 0 & -1 \\ -0.606 & 1.213 & -1 \\ -0.368 & 1.472 & -1 \\ -0.223 & 1.340 & -1 \end{bmatrix}$$

Normalgleichungssystem: $Dg^T Dg\delta = -Dg^T g$

Nach Lösung: $\delta^{(0)} = \begin{pmatrix} 0.32 \\ -0.18 \\ 0.61 \end{pmatrix}$ $\lambda^{(1)} = \lambda^{(0)} + \delta^{(0)} = \begin{pmatrix} 2.32 \\ 0.32 \\ 3.11 \end{pmatrix}$

Physikalische Interpretation: Die Funktion $f(x) = ae^{-bx} + c$ beschreibt einen exponentiellen Abfall mit:
- $a = 2.32$: Anfangsamplitude des abfallenden Anteils
- $b = 0.32$: Abfallkonstante (je größer, desto schneller der Abfall)
- $c = 3.11$: Asymptotischer Grenzwert für $x \rightarrow \infty$

Dies könnte z.B. einen Abkühlungsprozess, radioaktiven Zerfall oder Entladung eines Kondensators beschreiben.

Interpolation

Interpolation: Spezialfall der linearen Ausgleichsrechnung. Suche zu einer Menge von vorgegebenen Punkten eine Funktion, die exakt durch diese Punkte verläuft.

Interpolationsproblem Gegeben: $n + 1$ Wertepaare (x_i, y_i)
$i = 0, ..., n$, mit $x_i \neq x_j$ für $i \neq j$.
Gesucht: stetige Funktion g mit Eigenschaft $g(x_i) = y_i\ \forall i = 0, ..., n$.
Stützpunkte: $n + 1$ (x_i, y_i), **Stützstellen**: x_i, **Stützwerte**: y_i

Interpolation vs. Ausgleichsrechnung
- **Interpolation:** Gesuchte Funktion geht **exakt** durch alle Datenpunkte
- **Ausgleichsrechnung:**
Funktion **approximiert** die Datenpunkte möglichst gut
- Interpolation: Spezialfall der Ausgleichsrechnung ($m = n$, $E(f) = 0$)

Polynominterpolation

Lagrange Interpolationsformel

Durch $n+1$ Stützpunkte mit verschiedenen Stützstellen $\exists$ genau ein Polynom $P_n(x)$ vom Grade $\leq n$, das alle Stützpunkte interpoliert.
$P_n(x)$ lautet in der Lagrangeform: $P_n(x) = \sum_{i=0}^{n} l_i(x) y_i$
$l_i(x)$ (Lagrangepolynome vom Grad n): $l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$

Fehlerabschätzung

y_i = Funktionswerte einer genügend oft stetig differenzierbaren Funktion f (also $y_i = f(x_i)$), dann Interpolationsfehler an Stelle x:
$\left| f(x) - P_n(x) \right| \leq \frac{|(x-x_0)(x-x_1)\ldots(x-x_n)|}{(n+1)!} \max_{x_0 \leq \xi \leq x_n} f^{(n+1)}(\xi)$

Lagrange-Interpolation durchführen

Gegeben: $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$ und gesuchter Punkt x.
Lagrangepolynome $l_i(x)$: Für $i = 0, 1, ..., n$ berechne:

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

Interpolationspolynom: $P_n(x) = y_0 \cdot l_0(x) + y_1 \cdot l_1(x) + \cdots + y_n \cdot l_n(x)$
Funktionswert berechnen: Setze gewünschten x-Wert ein:
$P_n(x)$ = gesuchter Interpolationswert

Lagrange-Interpolation

Bestimme Atmosphärendruck bei 3750m:

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

Stützpunkte $(0, 1013)$, $(2500, 747)$, $(5000, 540)$ für $x = 3750$.
Lagrangepolynome:

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

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

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

Interpolationswert:
$P(3750) = 1013 \cdot (-0.125) + 747 \cdot 0.75 + 540 \cdot 0.375 = 636.0$ 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

Natürliche kubische Splinefunktion $S(x)$ ist in jedem Intervall $[x_i, x_{i+1}]$ durch 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

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).
Gleichungssystem für c_i lösen Für $i = 1, ..., n-1$:

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

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

Kubische Splinefunktion

Stützpunkte:

x_i	4	6	8	10
y_i	6	3	9	0

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)
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: $c_1 = 1.2, c_2 = -1.8$
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:
$S_0(x) = 6 - 2.8(x-4) + 0.6(x-4)^3$
$S_1(x) = 3 + 2.2(x-6) + 1.2(x-6)^2 - 1.5(x-6)^3$
$S_2(x) = 9 - 7.2(x-8) - 1.8(x-8)^2 + 0.9(x-8)^3$

Numerische Integration

Numerische Integration (Quadratur)

Für $f : \mathbb{R} \to \mathbb{R}$ soll das bestimmte Integral $I(f) = \int_a^b f(x)dx$ auf einem Intervall $[a, b]$ numerisch berechnet werden.

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

wobei x_i = **Stützstellen** oder **Knoten** und a_i = **Gewichte**

Newton-Cotes Formeln

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

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

- $n \in \mathbb{N}$ = Anzahl Subintervalle
- Schrittweite $h = \frac{b-a}{n}$
- Stützstellen $x_i = a + ih$ für $i = 0, 1, ..., n$

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

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

Trapezregel für nicht-äquidistante Stützstellen

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

Simpson-Regel

approximiert $f(x)$ durch Polynom 2. Grades an den Stellen $x_1 = a$, $x_2 = \frac{a+b}{2}$ und $x_3 = b$.

Einfache Simpson-Regel: $Sf = \frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b))$

Summierte Simpson-Regel:

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

Simpson-Regel als gewichtetes Mittel

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

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

Fehlerabschätzung

für **summierte** Quadraturformeln
Für genügend glatte Funktionen gelten folgende Fehlerabschätzungen:

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

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

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

Maximaler absoluter Fehler: ϵ

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

Schrittweite berechnen:

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

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

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

Anwendung Newton-Cotes Formeln

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 $n = 5$

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

Rechteckregel:

$$Rf(3) = 3 \cdot \sum_{i=0}^{4} f(x_i + 1.5)$$

Mittelpunkte: $6.5, 9.5, 12.5, 15.5, 18.5$

$Rf(3) = 3 \cdot (-0.154 - 0.108 - 0.090 - 0.081 - 0.076) = -1.527$

Trapezregel:

$$Tf(3) = 3 \cdot (\frac{f(5) + f(20)}{2} + \sum_{i=1}^{4} f(x_i))$$

$Tf(3) = 3 \cdot (\frac{-0.179-0.056}{2} + (-0.125-0.096-0.082-0.072)) = -1.477$

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: $|-1.510 - (-1.506)| = 0.004$

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.

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

Zweite Ableitung bestimmen: für $f(x) = e^{-x^2}$

$f'(x) = -2xe^{-x^2}$

$f''(x) = -2e^{-x^2} + 4x^2e^{-x^2} = e^{-x^2}(4x^2 - 2)$

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

Schrittweite berechnen:

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

Anzahl Intervalle: $n = \frac{0.5}{0.011} \approx 46$ 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 $T_{j0} = Tf(\frac{b-a}{2^j})$ 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:

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

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

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

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

Differentialgleichungen

Differentialgleichung n-ter Ordnung ist ein Gleichung von der Form:

$$F(x, y, y', y'', \ldots, y^{(n)}) = 0$$

- Eine Differentialgleichung für eine gesuchte Funktion $y = y(x)$, in der Ableitungen von $y(x)$ bis zur n-ten Ordnung auftreten.
- Falls die DGL nach $y^{(n)}$ aufgelöst ist, nennt man sie explizit, ansonsten implizit. Oft können implizite DGL durch einfaches Umformen in explizite DGL umgewandelt werden.

Arten von DGL

- **Separierbar:** falls $F(x, y)$ als Produkt eines x- und eines y-Anteils geschrieben werden kann, d.h. es hat die Form:

$$y' = g(x) \cdot h(y)$$

- **Autonom:** falls $F(x, y)$ nur von y abhängt, d.h. es hat die Form:

$$y' = f(y)$$

- **Linear:** falls die Variabel welche abgeleitet wird, nur in der ersten Potenz vorkommt und nicht multipliziert miteinander oder mit der unabhängigen Variabel wird.

Lösen von Separierbaren DGL

DGL: $\frac{\mathrm{d}y}{\mathrm{d}x} = g(x) \cdot h(y)$

$\rightarrow$ Falls $h(y_0) = 0$, ist $y = y_0$ eine Lösung der DGL.

- Trennung aller x- und y-Terme: $\frac{1}{h(y)} \cdot \mathrm{d}y = g(x) \cdot \mathrm{d}x$
- Integration auf beiden Seiten: $\int \frac{1}{h(y)} \mathrm{d}y = \int g(x) \mathrm{d}x$

Auflösen nach y, Anfangsbedingungen einsetzen:

$$\int_{y_0}^{y} \frac{1}{h(s)} \mathrm{d}s = \int_{x_0}^{x} g(t) \mathrm{d}t$$

Homogenität von DGL

- Homogene DGL: $F(x, y, y', y'', \ldots, y^{(n)}) = 0$
- Inhomogene DGL: $F(x, y, y', y'', \ldots, y^{(n)}) = g(x)$
  - $g(x)$ ist die Störfunktion

Allgemeine Lösung der inhomogenen DGL $y' + f(x)y = g(x)$ ist gegeben durch:

$$y = e^{-F(x)} \cdot \int g(x) e^{F(x)} \mathrm{d}x$$

wobei $F(x)$ eine Stammfunktion von $f(x)$ ist.

Anfangswertproblem DGL mit Anfangsbedingung

Anfangswertproblem n-ter Ordnung:

$$\begin{cases} F(x, y, y', y'', \ldots, y^{(n)}) &= 0, \ (x, y, \ldots, y^{(n)}) \in \Omega \\ y(x_0) &= y_0 \\ y'(x_0) &= y_1 \\ &\vdots \\ y^{n-1}(x_0) &= y_{n-1} \end{cases}$$

Anfangswertproblem für explizite DGL 1. Ordnung:

$$\begin{cases} y' &= G(x, y), \quad (x, y, y') \in \Omega \subseteq \mathbb{R} \times \mathbb{R}^2 \\ y(x_0) &= y_0 \end{cases}$$

- Allgemeine Lösung: Menge aller Lösungen einer DGL
- Spezielle/partikuläre Lösung: Lösung eines Anfangswertproblems

Lösung von Anfangswertproblemen mit seperiarbaren DGL

- Sind $g(x)$ und $h(y)$ stetige Funktionen und $(x_0, y_0) \in \mathbb{R}^2$ mit $h(y_0) \neq 0$, hat das Anfangswertproblem:

$$\begin{cases} y' &= g(x)h(y) \\ y(x_0) &= y_0 \end{cases}$$

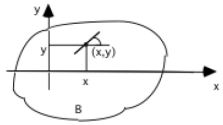
genau eine Lösung. Sie kann gefunden werden, indem beide Seiten von

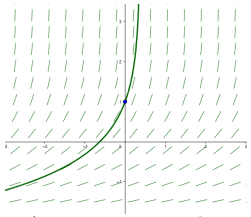
$$\int_{y_0}^{y} \frac{1}{h(s)} \mathrm{d}s = \int_{x_0}^{x} g(t) \mathrm{d}t$$

berechnet werden und nach y aufgelöst werden.

Richtungsfeder

Richtungsfeld geometrisches Verständnis von expliziten DGL 1. Ordnung, d.h. DGL der Form: $y' = f(x, y)$



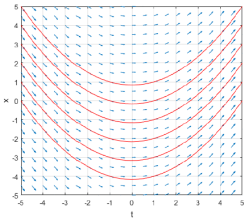


$f(x, y)$ gibt also die Steigung der Lösungskurve am Punkt (x, y) an

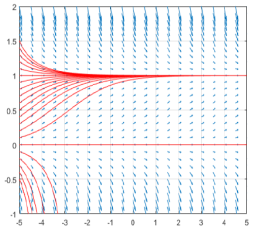
Jeder Punkt ist somit die Tangente einer spezifischen Lösungskurve

Richtungsfelder von Speziellen DGL

Unbestimmtes Integral: $y' = f(x)$
das Richtungsfeld ist unabhängig von y
die verschiedenen Lösungen unterscheiden sich nur durch eine verschiebung in y-Richtung durch die Konstante C.



Autonome DGL:$y' = f(y)$
das Richtungsfeld ist unabhängig von x
die Verschiedenen Lösungen gehen durch Verschiebung in x-Richtung in einander über.



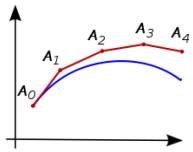
Numerische Verfahren

Eulerverfahren Gleichung einer beliebigen Geraden mit Steigung m am Punkt (x_k, y_k):

$$y = y_k + m \cdot (x - x_k)$$

DGL am Punkt (x_k, y_k):

$$y = y_k + f(x_k, y_k) \cdot (x - x_k)$$



- Für $k = 0$ und $x = x_0$:

$$\underbrace{y_1}_{\approx y(x_1)} = y_0 + f(x_0, y_0) \cdot \underbrace{(x_1 - x_0)}_{=h}$$

- Algorithmus für beliebige k:

$$\begin{cases} x_k &= x_0 + k \cdot h \\ y_{k+1} &= y_k + h \cdot f(x_k, y_k) \end{cases}$$

Problem: Die Steigung wird nur am linken Ende des Intervalls berücksichtigt!
$\Rightarrow$ Lösung: Verbesserte numerische Verfahren!

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:

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

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 - **Schritt 3: System aufstellen**

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

Schritt 4: Vektorielle Schreibweise

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

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.

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.

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:

$$v(t) = v_{rel} \ln(\frac{m_A}{m_A - \mu t}) - gt$$

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

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 $y(0) = 1$.

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)
# erweitert fuer Systeme: 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 2: 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 $[0, 10]$.

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 $\omega = 2$.

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: $\frac{dP}{dt} = rP(1 - \frac{P}{K})$ mit Wachstumsrate $r = 0.1$ und Kapazität $K = 1000$. Anfangspopulation: $P(0) = 50$.

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

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.

Prüfungsaufgabe 8.1 - Vergleich numerischer Verfahren

Aufgabe: Lösen Sie das Anfangswertproblem: $\frac{dy}{dx} = x + y^2, \quad y(0) = 0$ auf dem Intervall $[0, 1]$ mit Schrittweite $h = 0.2$.
a) Verwenden Sie das Euler-Verfahren b) Verwenden Sie das klassische Runge-Kutta Verfahren c) Vergleichen Sie die Genauigkeit bei $x = 1$ d) Welche Konvergenzordnung erwarten Sie?

---

a) Euler-Verfahren: $f(x, y) = x + y^2$, $h = 0.2$, $n = 5$
Iteration 0 → 1: $x_0 = 0, y_0 = 0$ $f(0, 0) = 0 + 0^2 = 0$ $x_1 = 0.2, y_1 = 0 + 0.2 \cdot 0 = 0$
Iteration 1 → 2: $x_1 = 0.2, y_1 = 0$ $f(0.2, 0) = 0.2 + 0^2 = 0.2$ $x_2 = 0.4, y_2 = 0 + 0.2 \cdot 0.2 = 0.04$
Iteration 2 → 3: $x_2 = 0.4, y_2 = 0.04$ $f(0.4, 0.04) = 0.4 + 0.04^2 = 0.4016$ $x_3 = 0.6, y_3 = 0.04 + 0.2 \cdot 0.4016 = 0.1203$
Iteration 3 → 4: $x_3 = 0.6, y_3 = 0.1203$ $f(0.6, 0.1203) = 0.6 + 0.1203^2 = 0.6145$ $x_4 = 0.8, y_4 = 0.1203 + 0.2 \cdot 0.6145 = 0.2432$
Iteration 4 → 5: $x_4 = 0.8, y_4 = 0.2432$ $f(0.8, 0.2432) = 0.8 + 0.2432^2 = 0.8591$ $x_5 = 1.0, y_5 = 0.2432 + 0.2 \cdot 0.8591 = 0.4150$
Euler-Ergebnis: $y(1) \approx 0.4150$
b) Runge-Kutta Verfahren:
Schritt 0 → 1: $x_0 = 0, y_0 = 0$ $k_1 = f(0, 0) = 0$ $k_2 = f(0.1, 0) = 0.1$ $k_3 = f(0.1, 0.01) = 0.1 + 0.01^2 = 0.1001$ $k_4 = f(0.2, 0.02002) = 0.2 + 0.02002^2 = 0.2004$ $y_1 = 0 + \frac{0.2}{6}(0 + 2 \cdot 0.1 + 2 \cdot 0.1001 + 0.2004) = 0.0200$
Schritt 1 → 2: $x_1 = 0.2, y_1 = 0.0200$ $k_1 = f(0.2, 0.0200) = 0.2 + 0.0004 = 0.2004$ $k_2 = f(0.3, 0.0400) = 0.3 + 0.0016 = 0.3016$ $k_3 = f(0.3, 0.0502) = 0.3 + 0.0025 = 0.3025$ $k_4 = f(0.4, 0.0805) = 0.4 + 0.0065 = 0.4065$ $y_2 = 0.0200 + \frac{0.2}{6}(0.2004 + 2 \cdot 0.3016 + 2 \cdot 0.3025 + 0.4065) = 0.0801$
Fortführung analog...
Runge-Kutta Ergebnisse: $y_1 = 0.0200, y_2 = 0.0801, y_3 = 0.1806, y_4 = 0.3214, y_5 = 0.5027$
Runge-Kutta-Ergebnis: $y(1) \approx 0.5027$
c) Genauigkeitsvergleich: Für diese DGL gibt es keine einfache analytische Lösung, aber numerische Referenzlösungen mit sehr kleiner Schrittweite ergeben: $y(1) \approx 0.5463$
Fehler:

- Euler: $|0.5463 - 0.4150| = 0.1313$
- Runge-Kutta: $|0.5463 - 0.5027| = 0.0436$

Das Runge-Kutta Verfahren ist etwa 3-mal genauer.
d) Konvergenzordnung:

- Euler-Verfahren: Konvergenzordnung $p = 1$
- Runge-Kutta 4: Konvergenzordnung $p = 4$

Bei Halbierung der Schrittweite erwarten wir:

- Euler: Fehler halbiert sich
- RK4: Fehler wird um Faktor 16 kleiner

Prüfungsaufgabe 8.2 - System von DGL

Aufgabe: Ein Satellit kreist um die Erde. Seine Bewegung wird beschrieben durch: $\ddot{x} = -\frac{GMx}{(x^2+y^2)^{3/2}}, \quad \ddot{y} = -\frac{GMy}{(x^2+y^2)^{3/2}}$
mit $GM = 3.986 \times 10^{14}$ m³/s².
Anfangsbedingungen: $x(0) = 7000$ km, $y(0) = 0$, $\dot{x}(0) = 0$, $\dot{y}(0) = 7500$ m/s
a) Formen Sie in ein System 1. Ordnung um b) Implementieren Sie einen Schritt des Mittelpunkt-Verfahrens mit $h = 10$ s c) Interpretieren Sie die Bewegung physikalisch

---

a) System 1. Ordnung: Einführung der Hilfsvariablen: $z_1 = x$ (Position x) $z_2 = \dot{x}$ (Geschwindigkeit x) $z_3 = y$ (Position y) $z_4 = \dot{y}$ (Geschwindigkeit y)
System: $z_1' = z_2$ $z_2' = -\frac{GMz_1}{(z_1^2+z_3^2)^{3/2}}$ $z_3' = z_4$ $z_4' = -\frac{GMz_3}{(z_1^2+z_3^2)^{3/2}}$
Vektorschreibweise: $\mathbf{z}' = \mathbf{f}(t, \mathbf{z}) = \begin{pmatrix} z_2 \\ -\frac{GMz_1}{(z_1^2+z_3^2)^{3/2}} \\ z_4 \\ -\frac{GMz_3}{(z_1^2+z_3^2)^{3/2}} \end{pmatrix}$
b) Mittelpunkt-Verfahren:
Anfangswerte: $\mathbf{z}(0) = \begin{pmatrix} 7\times10^6 \\ 0 \\ 0 \\ 7500 \end{pmatrix}$ (in SI-Einheiten)
Schritt 1: Berechne $\mathbf{f}(0, \mathbf{z}_0)$ $r_0 = \sqrt{(7 \times 10^6)^2 + 0^2} = 7 \times 10^6$ m
$a_0 = \frac{GM}{r_0^2} = \frac{3.986\times10^{14}}{(7\times10^6)^2} = 8.13$ m/s² $\mathbf{f}(0, \mathbf{z}_0) = \begin{pmatrix} 0 \\ -8.13 \\ 7500 \\ 0 \end{pmatrix}$
Schritt 2: Mittelpunkt berechnen
$\mathbf{z}_{1/2} = \mathbf{z}_0 + \frac{h}{2}\mathbf{f}(0, \mathbf{z}_0) = \begin{pmatrix} 7\times10^6 \\ 0 \\ 0 \\ 7500 \end{pmatrix} + 5\begin{pmatrix} 0 \\ -8.13 \\ 7500 \\ 0 \end{pmatrix} = \begin{pmatrix} 7\times10^6 \\ -40.65 \\ 37500 \\ 7500 \end{pmatrix}$
Schritt 3: $\mathbf{f}$ am Mittelpunkt berechnen
$r_{1/2} = \sqrt{(7 \times 10^6)^2 + (37500)^2} = 7.0001 \times 10^6$ m $\mathbf{f}(5, \mathbf{z}_{1/2}) = \begin{pmatrix} -40.65 \\ -8.128 \\ 7500 \\ -0.043 \end{pmatrix}$
Schritt 4: Neuen Punkt berechnen
$\mathbf{z}_1 = \mathbf{z}_0 + h \cdot \mathbf{f}(5, \mathbf{z}_{1/2}) = \begin{pmatrix} 7\times10^6 - 406.5 \\ -81.28 \\ 75000 \\ 7499.57 \end{pmatrix} = \begin{pmatrix} 6.9996\times10^6 \\ -81.28 \\ 75000 \\ 7499.57 \end{pmatrix}$
c) Physikalische Interpretation:

- Der Satellit startet in 7000 km Höhe (≈ 400 km über Erdoberfläche)
- Anfangsgeschwindigkeit 7500 m/s ist nahe der Kreisbahngeschwindigkeit
- Die Gravitationskraft sorgt für die zentripetale Beschleunigung
- Nach 10 s hat sich der Satellit bereits deutlich bewegt: $\Delta x = -406.5$ m, $\Delta y = 75000$ m
- Die Bahn ist eine Ellipse (oder bei passender Geschwindigkeit ein Kreis)
- Erhaltungsgrößen: Energie und Drehimpuls (sollten bei genauer numerischer Lösung konstant bleiben)

Prüfungsaufgabe 8.3 - Stabilität und Fehleranalyse

Aufgabe: Betrachten Sie die DGL: $y' = -5y + \cos(x), \quad y(0) = 1$
a) Bestimmen Sie die analytische Lösung b) Untersuchen Sie die Stabilität des Euler-Verfahrens c) Für welche Schrittweiten ist das Verfahren stabil? d) Vergleichen Sie numerische und analytische Lösung für $h = 0.3$ und $h = 0.5$

---

a) Analytische Lösung: Die DGL $y' + 5y = \cos(x)$ ist eine lineare DGL 1. Ordnung.
Homogene Lösung: $y_h = Ce^{-5x}$
Partikuläre Lösung durch Ansatz $y_p = A\cos(x) + B\sin(x)$: $y_p' = -A\sin(x) + B\cos(x)$
Einsetzen: $-A\sin(x) + B\cos(x) + 5(A\cos(x) + B\sin(x)) = \cos(x)$ $(-A + 5B)\sin(x) + (B + 5A)\cos(x) = \cos(x)$
Koeffizientenvergleich: $-A + 5B = 0 \Rightarrow A = 5B$ $B + 5A = 1 \Rightarrow B + 25B = 1 \Rightarrow B = \frac{1}{26}, A = \frac{5}{26}$
$y_p = \frac{5}{26}\cos(x) + \frac{1}{26}\sin(x)$
Allgemeine Lösung: $y = Ce^{-5x} + \frac{5}{26}\cos(x) + \frac{1}{26}\sin(x)$
Anfangsbedingung: $y(0) = C + \frac{5}{26} = 1 \Rightarrow C = \frac{21}{26}$
$y(x) = \frac{21}{26}e^{-5x} + \frac{5}{26}\cos(x) + \frac{1}{26}\sin(x)$
b) Stabilität des Euler-Verfahrens: Für die Testgleichung $y' = -\alpha y$ (hier $\alpha = 5$) lautet die Stabilitätsfunktion: $g(z) = 1 - z$ mit $z = h\alpha = 5h$
Stabilitätsbedingung: $|1 - 5h| < 1$ $-1 < 1 - 5h < 1$ $-2 < -5h < 0$ $0 < h < \frac{2}{5} = 0.4$
c) Stabilitätsbereich: Das Euler-Verfahren ist stabil für $0 < h < 0.4$.
d) Numerischer Vergleich:
Für $h = 0.3$ (stabil): Euler-Iteration mit $f(x, y) = -5y + \cos(x)$:
$x_0 = 0, y_0 = 1$ $y_1 = 1 + 0.3(-5 \cdot 1 + \cos(0)) = 1 + 0.3(-4) = -0.2$
$y_2 = -0.2 + 0.3(-5 \cdot (-0.2) + \cos(0.3)) = -0.2 + 0.3(1 + 0.955) = 0.387$
$y_3 = 0.387 + 0.3(-5 \cdot 0.387 + \cos(0.6)) = 0.387 + 0.3(-1.935 + 0.825) = 0.054$
Bei $x = 0.9$: $y_{\text{Euler}} \approx 0.054$ Analytisch: $y(0.9) = \frac{21}{26}e^{-4.5} + \frac{5}{26}\cos(0.9) + \frac{1}{26}\sin(0.9) = 0.197$
Für $h = 0.5$ (instabil): $x_0 = 0, y_0 = 1$ $y_1 = 1 + 0.5(-5 \cdot 1 + 1) = 1 + 0.5(-4) = -1$ $y_2 = -1 + 0.5(-5 \cdot (-1) + \cos(0.5)) = -1 + 0.5(5 + 0.878) = 1.939$ $y_3 = 1.939 + 0.5(-5 \cdot 1.939 + \cos(1)) = 1.939 + 0.5(-9.695 + 0.540) = -2.639$
Die Lösung oszilliert mit wachsender Amplitude → instabil!
Interpretation:

- Für $h = 0.3$: Das Verfahren ist stabil, aber nicht sehr genau
- Für $h = 0.5$: Das Verfahren wird instabil und divergiert
- Die Stabilitätsgrenze $h < 0.4$ muss eingehalten werden
- Der schnell abfallende Term e^{-5x} macht die DGL steif

Prüfungsaufgabe 8.4 - Anwendung: Populationsdynamik

Aufgabe: Das Räuber-Beute-System wird beschrieben durch: $\frac{dx}{dt} = ax - bxy$ $\frac{dy}{dt} = -cy + dxy$
wobei $x(t)$ die Beutepopulation und $y(t)$ die Räuberpopulation darstellt. Parameter: $a = 1.0$, $b = 0.5$, $c = 0.75$, $d = 0.25$ Anfangsbedingungen: $x(0) = 4$, $y(0) = 4$
a) Interpretieren Sie die Gleichungen biologisch b) Lösen Sie das System numerisch mit Runge-Kutta für $t \in [0, 15]$ mit $h = 0.1$ c) Analysieren Sie das Langzeitverhalten d) Was passiert mit der Gesamtenergie des Systems?

---

a) Biologische Interpretation: Beutegleichung: $\frac{dx}{dt} = ax - bxy$

- ax: Exponentielles Wachstum der Beute bei Abwesenheit von Räubern
- $-bxy$: Verluste durch Räuber (proportional zu beiden Populationen)
- $a = 1.0$: Wachstumsrate der Beute
- $b = 0.5$: Effizienz der Räuber beim Beutefang

Räubergleichung: $\frac{dy}{dt} = -cy + dxy$

- $-cy$: Exponentieller Rückgang bei Abwesenheit von Beute
- dxy: Wachstum durch erfolgreiche Jagd
- $c = 0.75$: Sterberate der Räuber
- $d = 0.25$: Effizienz der Nahrungsumwandlung

b) Numerische Lösung:
System in Vektorform:
$\mathbf{z}' = \mathbf{f}(t, \mathbf{z}) = \begin{pmatrix} 1.0z_1 - 0.5z_1z_2 \\ -0.75z_2 + 0.25z_1z_2 \end{pmatrix}$ mit $\mathbf{z}(0) = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$
Runge-Kutta Implementation (Auszug):
Schritt 0 → 1: $t_0 = 0$, $\mathbf{z}_0 = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$
$\mathbf{k}_1 = \mathbf{f}(0, \begin{pmatrix} 4 \\ 4 \end{pmatrix}) = \begin{pmatrix} 4-8 \\ -3+4 \end{pmatrix} = \begin{pmatrix} -4 \\ 1 \end{pmatrix}$
$\mathbf{k}_2 = \mathbf{f}(0.05, \begin{pmatrix} 4-0.2 \\ 4+0.05 \end{pmatrix}) = \mathbf{f}(0.05, \begin{pmatrix} 3.8 \\ 4.05 \end{pmatrix})$
$= \begin{pmatrix} 3.8-3.8\cdot4.05 \\ -3.0375+0.25\cdot3.8\cdot4.05 \end{pmatrix} = \begin{pmatrix} -11.59 \\ 0.81 \end{pmatrix}$
$\mathbf{k}_3 = \mathbf{f}(0.05, \begin{pmatrix} 4-0.58 \\ 4+0.041 \end{pmatrix}) = \begin{pmatrix} -11.73 \\ 0.69 \end{pmatrix}$
$\mathbf{k}_4 = \mathbf{f}(0.1, \begin{pmatrix} 4-1.17 \\ 4+0.069 \end{pmatrix}) = \begin{pmatrix} -15.77 \\ -0.23 \end{pmatrix}$
$\mathbf{z}_1 = \begin{pmatrix} 4 \\ 4 \end{pmatrix} + \frac{0.1}{6}\begin{pmatrix} -4-23.18-23.46-15.77 \\ 1+1.62+1.38-0.23 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix} + \begin{pmatrix} -1.11 \\ 0.063 \end{pmatrix} = \begin{pmatrix} 2.89 \\ 4.063 \end{pmatrix}$
Typische Ergebnisse nach vollständiger Integration:

t	$x(t)$	$y(t)$
0	4.00	4.00
3	1.32	2.85
6	2.67	1.13
9	6.21	2.34
12	3.89	5.67
15	1.45	3.12

c) Langzeitverhalten:

- Das System zeigt periodische Oszillationen (Grenzzyklus)
- Periode: $T \approx 6.3$ Zeiteinheiten
- Phasenverschiebung: Räuberpopulation folgt der Beutepopulation mit Verzögerung
- Typischer Zyklus:
  1. Viel Beute → Räuber vermehren sich
  2. Viele Räuber → Beute wird dezimiert
  3. Wenig Beute → Räuber sterben aus
  4. Wenige Räuber → Beute erholt sich

d) Erhaltungsgrößen: Das Lotka-Volterra System hat eine Erhaltungsgröße (Hamiltonfunktion): $H(x, y) = d \cdot x + b \cdot y - c \cdot \ln(x) - a \cdot \ln(y)$
$= 0.25x + 0.5y - 0.75\ln(x) - \ln(y)$

Prüfungsaufgabe 8.4- continued

Numerische Verifikation:

- $H(4, 4) = 1 + 2 - 0.75 \cdot 1.386 - 1.386 = 0.575$
- Nach einer Periode sollte $H \approx 0.575$ sein
- Abweichungen zeigen numerische Fehler an
- Das System ist konservativ (keine Dämpfung)

Praktische Bedeutung: Diese Gleichungen beschreiben reale Ökosysteme nur näherungsweise, da sie:

- Kapazitätsgrenzen ignorieren
- Andere Nahrungsquellen vernachlässigen
- Umwelteinflüsse nicht berücksichtigen
- Trotzdem wichtig für das Verständnis von Populationsdynamik

Zusätzliche Musteraufgaben

Musteraufgabe: Kombinierte Anwendung

Aufgabe: Ein Unternehmen modelliert seine Umsatzentwicklung durch die DGL: $\frac{dU}{dt} = 0.1U(100 - U) - S(t)$
wobei $U(t)$ der Umsatz in Millionen € und $S(t) = 20\sin(2\pi t)$ saisonale Schwankungen sind.
a) Klassifizieren Sie die DGL b) Für konstante Terme ($S = 0$): Bestimmen Sie Gleichgewichtspunkte c) Lösen Sie numerisch für $U(0) = 30$ und $t \in [0, 5]$ d) Erstellen Sie eine Ausgleichsrechnung für die numerischen Daten

---

a) Klassifikation:

- Nichtlineare DGL 1. Ordnung (wegen $U(100 - U)$ Term)
- Zeitabhängiger Störterm $S(t)$
- Ähnlich zur logistischen Gleichung mit Störung

b) Gleichgewichtspunkte für $S = 0$: $\frac{dU}{dt} = 0.1U(100 - U) = 0$
Lösungen: $U^* = 0$ oder $U^* = 100$
Stabilität: $\frac{d}{dU}(0.1U(100 - U)) = 0.1(100 - 2U)$

- Bei $U^* = 0$: $\frac{df}{dU} = 10 > 0$ → instabil
- Bei $U^* = 100$: $\frac{df}{dU} = -10 < 0$ → stabil

c) Numerische Lösung: Mit Runge-Kutta ($h = 0.1$):
System: $\frac{dU}{dt} = 0.1U(100 - U) - 20\sin(2\pi t)$
Typische Ergebnisse:

t	$U(t)$	$S(t)$	$\frac{dU}{dt}$
0	30.0	0	210.0
1	45.2	0	247.6
2	62.8	0	233.9
3	76.1	0	182.0
4	85.3	0	125.4
5	91.1	0	81.2

d) Ausgleichsrechnung: Fitten der numerischen Daten mit logistischer Funktion: $U(t) = \frac{K}{1 + Ae^{-rt}}$
Linearisierung durch: $\ln(\frac{K-U}{U}) = \ln(A) - rt$
Nach linearer Regression: $K \approx 100$, $A \approx 2.33$, $r \approx 0.693$
Güte des Fits: $R^2 > 0.98$ (sehr gute Anpassung an ungestörtes Wachstum)

Last Words

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.

Äpgjy Tipps für die Prüfungsvorbereitung:

- **Newton-Verfahren:** Üben Sie das Aufstellen und Lösen der Jacobi-Matrix
- **Ausgleichsrechnung:** Unterscheiden Sie klar zwischen linearen und nichtlinearen Problemen
- **Integration:** Beherrschen Sie Trapez-, Simpson-Regel und Romberg-Extrapolation
- **DGL:** Können Sie Systeme aufstellen und verschiedene Verfahren anwenden
- **Stabilität:** Verstehen Sie die Stabilitätsbedingungen der numerischen Verfahren
- **Interpretation:** Verbinden Sie mathematische Ergebnisse mit physikalischen/praktischen Bedeutungen
- **Fehleranalyse:** Können Sie Konvergenzordnungen bestimmen und Fehler abschätzen

Häufige Prüfungsthemen:

- Kombination von Verfahren (z.B. Newton + Ausgleichsrechnung)
- Anwendungsbezogene Aufgaben mit Interpretation
- Vergleich verschiedener numerischer Methoden
- Fehler- und Stabilitätsanalyse
- Implementierung in Python (Pseudocode)