Höhere Mathematik 2

Jil Zerndt
FS 2025

Numerische Lösung nicht linearer Gleichungssysteme

Funktionen mit mehreren Variablen

Skalarwertige Funktionen mit mehreren Variablen

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

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

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

Da das Ergebnis $y \in \mathbb{R}$ ein Skalar ist, spricht man von einer skalarwertigen Funktion.

Vektorwertige Funktionen

Für vektorwertige Funktionen $f : \mathbb{R}^n \to \mathbb{R}^m$ gilt:

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

wobei die m Komponenten $f_i : \mathbb{R}^n \to \mathbb{R}$ wieder skalarwertige Funktionen sind.

Beispiele für Funktionen mehrerer Variablen

Aufgabe: Gegeben seien verschiedene Funktionen. Bestimmen Sie deren Typ.

Lösung:
1. Addition: $f(x, y) = x + y$ (skalarwertig, $\mathbb{R}^2 \to \mathbb{R}$)
2. Ohmsches Gesetz: $U(R, I) = R \cdot I$ (skalarwertig)
3. Wurfweite: $W(v_0, \alpha) = \frac{v_0^2 \sin(2\alpha)}{g}$ (skalarwertig)
4. Vektorwertige Funktion: $f(x_1, x_2, x_3) = \begin{pmatrix} x_1^2 + x_2^2 \\ x_1^2 + x_3^2 \\ x_2^2 + x_3^2 \\ x_1^2 + x_2^2 + x_3^2 \end{pmatrix}$

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 Ableitungen berechnen

Schritt 1: Variable identifizieren

Bestimme, nach welcher Variable abgeleitet werden soll.

Schritt 2: Andere Variablen als Konstanten behandeln

Alle anderen Variablen werden während der Ableitung als Konstanten betrachtet.

Schritt 3: Standardableitungsregeln anwenden

Verwende die bekannten Ableitungsregeln für Funktionen einer Variablen.

Schritt 4: Ergebnis notieren

Schreibe das Ergebnis mit der korrekten Notation auf.

Partielle Ableitungen berechnen

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

Lösung:

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

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

An der Stelle $(-1, 1)$:

$$f_x(-1, 1) = 3 \cdot 1^2 + \frac{3}{-1} = 3 - 3 = 0$$

$$f_y(-1, 1) = 6 \cdot (-1) \cdot 1 + \frac{2}{1} = -6 + 2 = -4$$

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:

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

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 $x^{(0)}$. Man spricht von der **Linearisierung** der Funktion $y = f(x)$ in einer Umgebung von $x^{(0)}$.

Jacobi-Matrix berechnen und linearisieren

Schritt 1: Komponentenfunktionen identifizieren

Bestimme alle Komponentenfunktionen $f_1, f_2, ..., f_m$ und Variablen $x_1, x_2, ..., x_n$.

Schritt 2: Partielle Ableitungen berechnen

Berechne alle partiellen Ableitungen $\frac{\partial f_i}{\partial x_j}$ für $i = 1, ..., m$ und $j = 1, ..., n$.

Schritt 3: Jacobi-Matrix aufstellen

Ordne die partiellen Ableitungen in Matrixform an.

Schritt 4: An Entwicklungspunkt auswerten

Setze den Entwicklungspunkt $x^{(0)}$ in die Jacobi-Matrix ein.

Schritt 5: Linearisierung berechnen

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

Jacobi-Matrix und Linearisierung

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

Lösung:
Schritt 1: Partielle Ableitungen berechnen

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

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

Schritt 2: Jacobi-Matrix aufstellen

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

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

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

Newton-Verfahren für Systeme

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 $i = 1, ..., n$.

Newton-Verfahren für Systeme

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:

Für $n = 0, 1, ...$:

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

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

2. 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:

- Berechne $f(x^{(k)})$ und $Df(x^{(k)})$
- Löse das LGS: $Df(x^{(k)})\delta^{(k)} = -f(x^{(k)})$
- Setze $x^{(k+1)} = x^{(k)} + \delta^{(k)}$

Schritt 4: Konvergenzprüfung

Prüfe Abbruchkriterien wie $\|f(x^{(k+1)})\|_2 <$ TOL oder $\|x^{(k+1)} - x^{(k)}\|_2 <$ TOL.

Schritt 5: Lösung interpretieren

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

Newton-Verfahren anwenden

Aufgabe: Lösen Sie 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 $x^{(0)} = (1.1, 0.9)^T$.

Lösung:

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.

Gedämpftes Newton-Verfahren

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 ßchlechterenSStartvektor $x^{(0)} = (2, 2)^T$.

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.

Persönliche Notizen

Numerische Lösung nichtlinearer Gleichungssysteme

In diesem Kapitel geht es um die Nullstellenbestimmung von nichtlinearen Funktionen $f : \mathbb{R}^n \to \mathbb{R}^n$.
Abkürzung: LGS = lineares Gleichungssystem, NGS = nichtlineares Gleichungssystem

Einleitendes Beispiel

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

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

Gesucht sind die Lösungen des Gleichungssystems. Diese lassen sich interpretieren als die Nullstellen der Funktion $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$ gemäss:

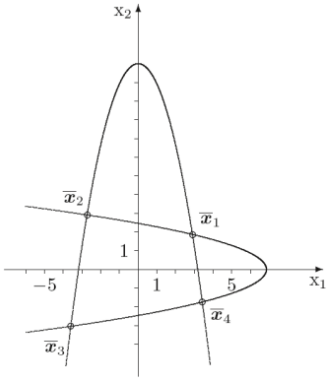
$$\mathbf{f}(x) = \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} x_1^2 + x_2 - 11 \\ x_1 + x_2^2 - 7 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

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

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

$$x_2 = 11 - x_1^2$$

$$x_2 = \sqrt{7 - x_1}$$



Schnittpunkte:

$$\overline{\mathbf{x}_1} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}, \quad \overline{\mathbf{x}_2} = \begin{pmatrix} -2.8 \\ 3.2 \end{pmatrix}$$

$$\overline{\mathbf{x}_3} = \begin{pmatrix} -3.8 \\ -3.3 \end{pmatrix}, \quad \overline{\mathbf{x}_4} = \begin{pmatrix} 3.4 \\ -1.7 \end{pmatrix}$$

Funktionen mit mehreren Variablen

Funktion mit abhängiger Variable x, unabhängiger Variable $y = f(x)$:

$$f : \mathbb{R} \to \mathbb{R}$$

$$x \mapsto f(x)$$

Skalarwertige Funktionen mit mehreren Variablen

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

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

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 Erweiterung der obigen Definition, gibt einen **Vektor** zurück (anstatt eines Skalars).
Sei $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ eine Funktion mit n Variablen. Dann ist die Funktion $\mathbf{f}$ definiert durch:

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

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.

Beispiele

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

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

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

Zusammenhang mit der Elektrotechnik

Ohmsches Gesetz

Die an einem ohmschen Widerstand R abfallende Spannung U hängt vom Widerstand R und der Stromstärke I gemäss dem ohmschen Gesetz $U = R \cdot I$ ab. Also haben wir für die abhängige Variable $U = f(R, I) = RI$ die skalarwertige Funktion $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$ mit den unabhängigen Variablen R und I. Häufig schreibt man auch direkt

$$U = U(R, I) = R \cdot I$$

und bringt dadurch die Abhängigkeit der Variable U von den unabhängigen Variablen R und I zum Ausdruck, wie wir es auch bereits vom eindimensionalen Fall kennen, z.B. $y = y(x)$.

Reihenschaltung von Widerständen

Bei der Reihenschaltung von n ohmschen Widerständen $R_1, R_2, \ldots, R_n$ ergibt sich der Gesamtwiderstand R gemäss

$$R = R(R_1, R_2, \ldots, R_n) = R_1 + R_2 + \ldots + R_n$$

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:

$$\mathbf{Ax} = \mathbf{b} \text{ mit } \mathbf{A} = \begin{pmatrix} 4 & -1 & 1 \\ -2 & 5 & 1 \\ 1 & -2 & 5 \end{pmatrix} \text{ und } \mathbf{b} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

gerade $\mathbf{f(x)} = \mathbf{0}$ ergibt.

Vorgehen:

$$\vec{\mathbf{f}}(x_1, x_2, x_3) = 0 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\mathbf{A}\vec{\mathbf{x}} = \vec{\mathbf{b}} \Rightarrow \underbrace{\mathbf{A}\vec{\mathbf{x}} - \vec{\mathbf{b}} = \vec{\mathbf{0}}}_{\vec{\mathbf{f}}(\vec{\mathbf{x}})}$$

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

Die Funktion $\mathbf{f}$ ist gegeben durch: (**solved by copilot so no guarantees**)

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

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.
Vorgehen:
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:

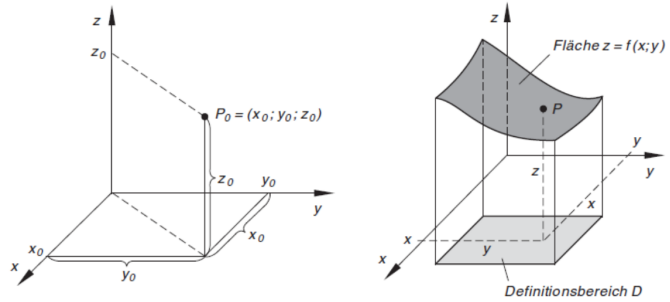
2. unabhängige Variable y

1. unabhängige Variable x \ y	y_1	y_2	$\cdots$	y_k	$\cdots$	y_n	
x_1	z_{11}	z_{12}	$\cdots$	z_{1k}	$\cdots$	z_{1n}	
x_2	z_{21}	z_{22}	$\cdots$	z_{2k}	$\cdots$	z_{2n}	
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	
x_i	z_{i1}	z_{i2}	$\cdots$	(z_{ik})	$\cdots$	z_{in}	$\leftarrow i$-te Zeile
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	
x_m	z_{m1}	z_{m2}	$\cdots$	z_{mk}	$\cdots$	z_{mn}	
				$\uparrow$			
				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:

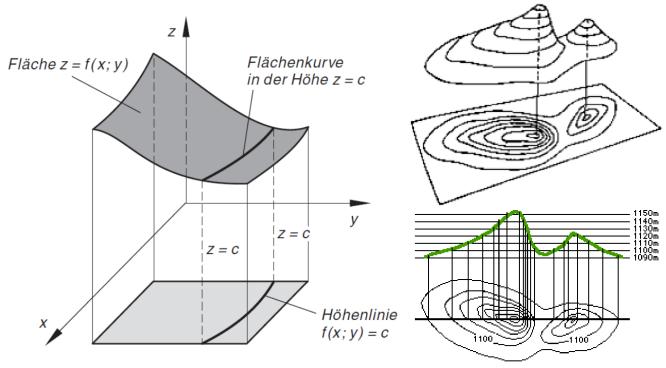
Darstellung einer Funktion als Fläche im Raum

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

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

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

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

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

Linearisierung von Funktionen mit mehreren Variablen

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

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

Verallgemeinerte Tangentengleichung

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

Beschreibt eine lineare Funktion und es gilt $f(x) \approx g(x)$ in einer Umgebung eines gegebenen Vektors $x^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)^T \in \mathbb{R}^n$.
Man spricht deshalb auch 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 $\boldsymbol{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}\right)^T \in \mathbb{R}^2$ ist die Jacobi-Matrix nur ein Zeilenvektor mit zwei Elementen, nämlich:

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

Dann liefert die Linearisierung

$$g(x_1, x_2) = f\left(x_1^{(0)}, x_2^{(0)}\right) + \left(\frac{\partial f}{\partial x_1}\left(x_1^{(0)}, x_2^{(0)}\right) \quad \frac{\partial f}{\partial x_2}\left(x_1^{(0)}, x_2^{(0)}\right)\right) \cdot \left($$
$$= f\left(x_1^{(0)}, x_2^{(0)}\right) + \frac{\partial f}{\partial x_1}\left(x_1^{(0)}, x_2^{(0)}\right) \cdot \left(x_1 - x_1^{(0)}\right) + \frac{\partial f}{\partial x_2}\left(x_1^{(0)}\right.$$

die Gleichung der Tangentialebene.

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

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

1. Jacobi-Matrix $\mathrm{Df}(x_1, x_2, x_3)$ bilden

$$f(x_1, x_2, x_3) = \begin{pmatrix} \sin(x_2 + 2x_3) \\ \cos(2x_1 + x_2) \end{pmatrix}, \quad Df(x_1, x_2, x_3) = \begin{pmatrix} 0 \\ -\sin(2x_1 + x_2) \cdot 2 \end{pmatrix}$$

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

$$f(\pi/4, 0, \pi) = f\left(x^{(0)}\right) = \begin{pmatrix} \sin(0 + 2\pi) \\ \cos(2 \cdot \pi/4 + 0) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad Df\left(x^{(0)}\right) = \begin{pmatrix} 0 & 1 & 2 \\ -2 & -1 & 0 \end{pmatrix}$$

3. Verallgemeinerte Tangentengleichung

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

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.

Newton-Verfahren

????? Das Newton-Verfahren ist ein iteratives Verfahren zur Bestimmung von Nullstellen einer Funktion. Es basiert auf der Linearisierung der Funktion um einen Startwert $x^{(0)}$.

- **Vorgehen:**
  1. Startwert $x^{(0)}$ wählen
  2. Linearisierung der Funktion um $x^{(0)}$
  3. Nullstelle der Linearisierung berechnen
  4. Neue Linearisierung um die berechnete Nullstelle
  5. Iteration bis Konvergenz
- **Formel:**

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

Quadratisch konvergentes Newton-Verfahren (Quadratische Konvergenz)

Lösung von $f(x) = 0$ mit $f : \mathbb{R}^n \to \mathbb{R}^n$ für $n = 0, 1, 2, \ldots$

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

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 $\rightarrow k = 0$

4. Setze

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

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 $f_1, f_2 = 0$

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

1: Vektorwertige Funktion und Jacobi-Matrix bilden

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

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

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

2: Berechne $\delta^{(0)}$

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

3: Berechne $x^{(1)}$

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

Claude try 1

Numerical Solution of Nonlinear Equation Systems

Introduction and Motivation

Nonlinear Equation Systems

A nonlinear equation system is a system of n equations with n unknowns that cannot be written in the form $Ax = b$ where A is a matrix, x is the vector of unknowns, and b is a constant vector. Solving such systems requires finding values for variables that simultaneously satisfy all equations.

Consider the nonlinear system:

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

This can be represented as finding the zeros of a vector-valued function $f : \mathbb{R}^2 \to \mathbb{R}^2$:

$$f(x) = \begin{pmatrix} x_1^2 + x_2 - 11 \\ x_1 + x_2^2 - 7 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The geometric interpretation is finding the intersection points of two curves: $x_1^2 + x_2 = 11$ and $x_1 + x_2^2 = 7$.

Functions of Several Variables

Functions of Several Variables

A function with n independent variables $x_1, x_2, \ldots, x_n$ and one dependent variable y is a mapping that assigns to each point $(x_1, x_2, \ldots, x_n)$ from a domain $D \subset \mathbb{R}^n$ exactly one element y from a range $W \subset \mathbb{R}$.
We write: $f : D \subset \mathbb{R}^n \to W \subset \mathbb{R}$, $(x_1, x_2, \ldots, x_n) \mapsto y = f(x_1, x_2, \ldots, x_n)$

Vector-Valued Functions

A vector-valued function maps points from $\mathbb{R}^n$ to $\mathbb{R}^m$: $f : \mathbb{R}^n \to \mathbb{R}^m$

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

Where the m components $f_i : \mathbb{R}^n \to \mathbb{R}$ are scalar-valued functions.

Partial Derivatives

Partial Derivatives

For a function $z = f(x, y)$, the partial derivatives at the point (x, y) are defined as:

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

Alternative notations include: $f_x(x, y)$, $f_y(x, y)$ or $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$.

Geometrically, $\frac{\partial f}{\partial x}(x_0, y_0)$ represents the slope of the tangent to the surface $z = f(x, y)$ at the point (x_0, y_0, z_0) in the positive x-direction, and similarly for $\frac{\partial f}{\partial y}(x_0, y_0)$ in the y-direction.

For the function $z = f(x, y) = 3xy^2 + \ln(x^3y^2)$, the partial derivatives are:

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

Linearization of Functions of Several Variables

Jacobian Matrix and Linearization

For a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $y = f(x) = \begin{pmatrix} y_1 = f_1(x) \\ \vdots \\ y_m = f_m(x) \end{pmatrix}$ and $x = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$, the Jacobian matrix contains all first-order partial derivatives and is defined as:

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

The linearization of f at a point $x^{(0)}$ is given by:

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

This gives a linear approximation of $f(x)$ in the neighborhood of $x^{(0)}$.

Consider $f(x_1, x_2) = \begin{pmatrix} x_1^2 + x_2 - 11 \\ x_1 + x_2^2 - 7 \end{pmatrix}$, linearized at $x^{(0)} = (1, 1)^T$.

The Jacobian matrix is:

$$Df(x_1, x_2) = \begin{pmatrix} 2x_1 & 1 \\ 1 & 2x_2 \end{pmatrix}$$

At the point $x^{(0)} = (1, 1)^T$:

$$Df(1, 1) = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

The linearization is:

$$\begin{aligned} g(x) &= f(x^{(0)}) + Df(x^{(0)}) \cdot (x - x^{(0)}) \\ &= \begin{pmatrix} -9 \\ -5 \end{pmatrix} + \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 - 1 \\ x_2 - 1 \end{pmatrix} \\ &= \begin{pmatrix} -9 + 2(x_1 - 1) + (x_2 - 1) \\ -5 + (x_1 - 1) + 2(x_2 - 1) \end{pmatrix} \\ &= \begin{pmatrix} 2x_1 + x_2 - 12 \\ x_1 + 2x_2 - 8 \end{pmatrix} \end{aligned}$$

Newton's Method for Systems

Newton's Method for Nonlinear Systems

We seek to find zeros of a function $f : \mathbb{R}^n \to \mathbb{R}^n$.

Algorithm

1. Start with an initial guess $x^{(0)}$ close to a zero of f.
2. For $n = 0, 1, 2, \ldots$ until convergence:
   - Compute the correction $\delta^{(n)}$ by solving the linear system: $Df(x^{(n)})\delta^{(n)} = -f(x^{(n)})$
   - Update the approximation: $x^{(n+1)} = x^{(n)} + \delta^{(n)}$
3. Convergence is achieved when a suitable stopping criterion is met, such as:
   - $n \geq n_{max}$ (maximum iterations reached)
   - $\|x^{(n+1)} - x^{(n)}\| \leq \epsilon \|x^{(n+1)}\|$
   - $\|x^{(n+1)} - x^{(n)}\| \leq \epsilon$
   - $\|f(x^{(n+1)})\| \leq \epsilon$

Convergence

Newton's method converges quadratically if the starting point is sufficiently close to a zero, $Df(x)$ is regular, and f is three times continuously differentiable.

Newton's Method Application

Let's solve the system:

$$f(x_1, x_2) = \begin{pmatrix} 2x_1 + 4x_2 \\ 4x_1 + 8x_2^3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The Jacobian matrix is:

$$Df(x_1, x_2) = \begin{pmatrix} 2 & 4 \\ 4 & 24x_2^2 \end{pmatrix}$$

Starting with $x^{(0)} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$:

1. Compute:

$$Df(4, 2) = \begin{pmatrix} 2 & 4 \\ 4 & 96 \end{pmatrix}$$
$$f(4, 2) = \begin{pmatrix} 16 \\ 80 \end{pmatrix}$$

2. Solve the linear system:

$$\begin{pmatrix} 2 & 4 \\ 4 & 96 \end{pmatrix} \delta^{(0)} = -\begin{pmatrix} 16 \\ 80 \end{pmatrix}$$

Yielding $\delta^{(0)} = \begin{pmatrix} -\frac{76}{11} \\ \frac{6}{11} \end{pmatrix}$

3. Update:

$$x^{(1)} = \begin{pmatrix} 4 \\ 2 \end{pmatrix} + \begin{pmatrix} -\frac{76}{11} \\ \frac{6}{11} \end{pmatrix} = \begin{pmatrix} -\frac{32}{11} \\ \frac{16}{11} \end{pmatrix} \approx \begin{pmatrix} -2.909 \\ 1.455 \end{pmatrix}$$

The sequence converges to $(-2, 1)^T$, one of the solutions.

Simplified Newton's Method

Simplified Newton's Method

To reduce computational effort per iteration, we can use the same Jacobian matrix for all iterations:

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

This method converges only linearly but requires fewer computations per step.

Damped Newton's Method

Damped Newton's Method

When the Jacobian matrix is nearly singular, the standard Newton step may cause the iterations to diverge. The damped Newton's method adjusts the step size to ensure convergence.

Algorithm

1. Start with an initial guess $x^{(0)}$ close to a zero of f.
2. For $n = 0, 1, 2, \ldots$ until convergence:
   - Compute $\delta^{(n)}$ by solving:
     $Df(x^{(n)})\delta^{(n)} = -f(x^{(n)})$
   - Find the smallest $k \in \{0, 1, \ldots, k_{max}\}$ such that:
     $\|f(x^{(n)} + \frac{\delta^{(n)}}{2^k})\|_2 < \|f(x^{(n)})\|_2$
   - If no such k is found, use $k = 0$
   - Update:
     $x^{(n+1)} = x^{(n)} + \frac{\delta^{(n)}}{2^k}$

Notes

- The damping parameter k_{max} is problem-dependent
- A value of $k_{max} = 4$ is often a good starting point
- Damped Newton's method generally converges more reliably than the standard method

Ausgleichsrechnung

Interpolation

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:

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

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

Aufgabe: Bestimmen Sie 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 initialisieren

$a_i = y_i$ und $h_i = x_{i+1} - x_i$

Schritt 2: Randbedingungen setzen

$c_0 = 0$, $c_n = 0$

Schritt 3: 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 4: 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: $a_0 = 6, a_1 = 3, a_2 = 9$ $h_0 = h_1 = h_2 = 2$
Schritt 2: $c_0 = 0, c_3 = 0$
Schritt 3: 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 4: 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

Aufgabe: Bestimmen Sie 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

Aufgabe: Fitten Sie 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

Aufgabe: Fitten Sie 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

Lösung:

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

Aufgabe: Fitten Sie 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

Persönliche Notizen

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 $(x_i, y_i), i = 0, \ldots, n$ 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 $i = 0, \ldots, n$.
Die Polynominterpolation kann mittels Linearem Gleichungssystem gelöst werden (Vandermonde-Matrix). Dies führt jedoch zu einer schlechten Konditionierung. Daher müssen anderen Verfahren verwendet werden.

Lagrange-Interpolationspolynom

Durch $n+1$ Stützpunkte mit verschiedenen Stützstellen gibt es genau ein Polynom $P_n(x)$ vom Grade $\leq n$, welches alle Stützpunkte interpoliert, d.h. wo gilt

$$P_n(x_i) = y_i, \quad i = 0, 1, \ldots, n$$

$P_n(x)$ lautet in der Lagrangeform

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

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

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

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

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

Berechnung der Lagrange-Interpolationspolynome

Gegeben sind $x = [0, 250, 500, 1000]$, $y = [1013, 747, 540, 226]$
Berechnung der Lagrangepolynome

$$l_0 = \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_2} \cdot \frac{x - x_3}{x_0 - x_3} = \frac{375 - 250}{0 - 250} \cdot \frac{375 - 500}{0 - 500} \cdot \frac{375 - 1000}{0 - 1000} = -$$

$$l_1 = \frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} \cdot \frac{x - x_3}{x_1 - x_3} = \frac{375 - 0}{250 - 0} \cdot \frac{375 - 500}{250 - 500} \cdot \frac{375 - 1000}{250 - 1000} = 0.62$$

$$l_2 = \frac{x - x_0}{x_2 - x_0} \cdot \frac{x - x_1}{x_2 - x_1} \cdot \frac{x - x_3}{x_2 - x_3} = \frac{375 - 0}{500 - 0} \cdot \frac{375 - 250}{500 - 250} \cdot \frac{375 - 1000}{500 - 1000} = 0.46$$

$$l_3 = \frac{x - x_0}{x_3 - x_0} \cdot \frac{x - x_1}{x_3 - x_1} \cdot \frac{x - x_2}{x_3 - x_2} = \frac{375 - 0}{1000 - 0} \cdot \frac{375 - 250}{1000 - 250} \cdot \frac{375 - 500}{1000 - 500} = -$$

Gesucht ist der y-Wert an der Stelle $x = 375$

$$P_n(375) = \sum_{i=0}^{n} l_i(375) \cdot y_i$$

$$P_n(375) = -0.078 \cdot 1013 + 0.625 \cdot 747 + 0.469 \cdot 540 + -0.016 \cdot 226$$

$$P_n(375) = 637.328\text{hPa}$$

Algorithmus: natürliche kubische Splinefunktion

Gegeben seien $n+1$ Stützpunkte (x_i, y_i) mit monoton aufsteigenden Stützstellen (Knoten) $x_0 < x_1 < \cdots < x_n$ ($n \geq 2$). Gesucht ist die natürliche kubische Splinefunktion $S(x)$, welche in jedem Intervall $\left[x_i, x_{i+1}\right]$ mit $= 0, 1, \ldots, n-1$ durch ein kubisches Polynom:

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

Berechnung der Polynome $S_i(x)$ für $i = 0, 1, \ldots, n-1$:
1: Koeffizienten a_i, h_i, c_0, c_n

$$a_i = y_i, \quad h_i = x_{i+1} - x_i, \quad c_0 = 0, \quad c_n = 0$$

2: Koeffizienten $c_1, c_2, \ldots c_{n-1}$ aus dem Gleichungssystem $(Ac = z)$
- $i = 1$

$$2(h_0 + h_1) \cdot c_1 + h_1 c_2 \quad = 3\frac{(y_2 - y_1)}{h_1} - 3\frac{(y_1 - y_0)}{h_0}$$

- $n \geq 4 \rightarrow i = 2, \ldots, n-2$

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

- $i = n-1$

$$h_{n-2} c_{n-2} + 2(h_{n-2} + h_{n-1}) \cdot c_{n-1} = 3\frac{(y_n - y_{n-1})}{h_{n-1}} - 3\frac{(y_{n-1} - y_{n-2})}{h_{n-2}}$$

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

Beispiel Verwendung des Algorithmus

$$x = [4, 6, 8, 10], \quad y = [6, 3, 9, 0], \quad i = [0, 1, 2, 3], n = 3$$

Koeffizienten a_i, h_i, c_0, c_n - $a_i = y_i$ $a = [6, 3, 9]$ - $h_i = x_{i+1} - x_i$ $h = [2, 2, 2]$ - $c_0 = 0, c_n = 0$
Koeffizienten $c_1, c_2, \ldots c_{n-1}$ aus dem Gleichungssystem ($Ac = z$)

$$A = \begin{pmatrix} 2(h_0 + h_1) & h_1 \\ h_1 & 2(h_1 + h_2) \end{pmatrix}, \quad z = \begin{pmatrix} 3\frac{(y_2 - y_1)}{h_1} - 3\frac{(y_1 - y_0)}{h_0} \\ 3\frac{(y_3 - y_2)}{h_2} - 3\frac{(y_2 - y_1)}{h_1} \end{pmatrix}$$

$$Ac = z \rightarrow \left(\begin{array}{cc|c} 8 & 2 & 6 \\ 2 & 8 & -3 \end{array}\right), \quad c = \begin{pmatrix} 2.55 \\ -3.45 \end{pmatrix}$$

Koeffizienten b_i und d_i

$$A = \begin{pmatrix} 2(h_0 + h_1) & h_1 & & \square \\ h_1 & 2(h_1 + h_2) & \ddots & \vdots \\ \square & \ddots & \ddots & h_{n-2} \\ \square & \square & h_{n-2} & 2(h_{n-2} + h_{n-1}) \end{pmatrix}, \quad c =$$

Claude try 1

Curve Fitting and Approximation

Introduction and Motivation

Curve Fitting and Approximation

Curve fitting (or approximation) is a mathematical optimization method used to find a function that best represents a given set of data points. In general, these problems are overdetermined, meaning there are more data points than parameters to determine.

Types of Approximation Problems

We distinguish between two main types of approximation problems:

- **Interpolation:** Finding a function that passes exactly through all given data points.
- **Regression:** Finding a function that approximates the data points as closely as possible according to some error criterion (typically least squares).

Interpolation

Interpolation Problem

Given $n+1$ data points (x_i, y_i), $i = 0, \ldots, n$, with $x_i \neq x_j$ for $i \neq j$, find a continuous function g with the property $g(x_i) = y_i$ for all $i = 0, \ldots, n$.

Polynomial Interpolation

Lagrange Interpolation Formula

Through $n+1$ data points with distinct x-values, there exists exactly one polynomial $P_n(x)$ of degree $\leq n$ that interpolates all points, i.e., $P_n(x_i) = y_i$ for $i = 0, 1, \ldots, n$.
The polynomial in Lagrange form is:

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

where the Lagrange basis polynomials $l_i(x)$ are defined by:

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

Lagrange Interpolation

Given the temperature data:

t [hour]	8:00	10:00	12:00	14:00
T [°C]	11.2	13.4	15.3	19.5

Let's find the temperature at 11:00 using Lagrange interpolation.
For simplicity, we'll use $t = 0, 2, 4, 6$ for the four times (8:00, 10:00, 12:00, 14:00). Then $t = 3$ corresponds to 11:00.
The Lagrange basis polynomials are:

$$l_0(t) = \frac{(t-2)(t-4)(t-6)}{(0-2)(0-4)(0-6)} = -\frac{1}{48}t^3 + \frac{3}{4}t^2 - \frac{107}{12}t + 35$$

$$l_1(t) = \frac{(t-0)(t-4)(t-6)}{(2-0)(2-4)(2-6)} = \frac{1}{16}t^3 - \frac{17}{8}t^2 + \frac{47}{2}t - 84$$

$$l_2(t) = \frac{(t-0)(t-2)(t-6)}{(4-0)(4-2)(4-6)} = -\frac{1}{16}t^3 + 2t^2 - \frac{83}{4}t + 70$$

$$l_3(t) = \frac{(t-0)(t-2)(t-4)}{(6-0)(6-2)(6-4)} = \frac{1}{48}t^3 - \frac{5}{8}t^2 + \frac{37}{6}t - 20$$

The interpolation polynomial is:

$$\begin{aligned} P_3(t) &= 11.2 \cdot l_0(t) + 13.4 \cdot l_1(t) + 15.3 \cdot l_2(t) + 19.5 \cdot l_3(t) \\ &= \frac{13}{240}t^3 - \frac{133}{80}t^2 + \frac{2137}{120}t - \frac{263}{5} \end{aligned}$$

At $t = 3$ (11:00), we get $P_3(3) = 14.225$°C.

For direct evaluation at $t = 3$ without computing the full polynomial:

$$l_0(3) = \frac{(3-2)(3-4)(3-6)}{(0-2)(0-4)(0-6)} = \frac{(1)(-1)(-3)}{(-2)(-4)(-6)} = -\frac{1}{16}$$

$$l_1(3) = \frac{(3-0)(3-4)(3-6)}{(2-0)(2-4)(2-6)} = \frac{(3)(-1)(-3)}{(2)(-2)(-4)} = \frac{9}{16}$$

$$l_2(3) = \frac{(3-0)(3-2)(3-6)}{(4-0)(4-2)(4-6)} = \frac{(3)(1)(-3)}{(4)(2)(-2)} = \frac{9}{16}$$

$$l_3(3) = \frac{(3-0)(3-2)(3-4)}{(6-0)(6-2)(6-4)} = \frac{(3)(1)(-1)}{(6)(4)(2)} = -\frac{1}{16}$$

And:

$$\begin{aligned} P_3(3) &= 11.2 \cdot (-\frac{1}{16}) + 13.4 \cdot (\frac{9}{16}) + 15.3 \cdot (\frac{9}{16}) + 19.5 \cdot (-\frac{1}{16}) \\ &= 14.225\text{°C} \end{aligned}$$

Interpolation Error

If the y_i are function values of a sufficiently differentiable function f (thus $y_i = f(x_i)$), then the interpolation error at a point x is given by:

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

Spline Interpolation

Cubic Spline Interpolation

A cubic spline $S(x)$ for data points (x_i, y_i), $i = 0, \ldots, n$ is a piecewise polynomial function that:

- Consists of cubic polynomials $S_i(x)$ on each interval $[x_i, x_{i+1}]$
- Interpolates all data points: $S(x_i) = y_i$
- Is continuous: $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$ for $i = 0, \ldots, n-2$
- Has continuous first derivatives: $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$ for $i = 0, \ldots, n-2$
- Has continuous second derivatives: $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$ for $i = 0, \ldots, n-2$

Two additional conditions are needed to uniquely define the spline. For a natural cubic spline, these are:

$$S''(x_0) = S''(x_n) = 0$$

Algorithm for Natural Cubic Spline

Given $n+1$ data points (x_i, y_i) with $x_0 < x_1 < \ldots < x_n$ (where $n \geq 2$), the natural cubic spline $S(x)$ consists of cubic polynomials

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

for $x \in [x_i, x_{i+1}]$ with $i = 0, 1, \ldots, n-1$.

Steps

1. Set $a_i = y_i$ for all i
2. Calculate intervals $h_i = x_{i+1} - x_i$ for $i = 0, 1, \ldots, n-1$
3. Set boundary conditions $c_0 = 0$ and $c_n = 0$ (natural spline)
4. Calculate coefficients $c_1, c_2, \ldots, c_{n-1}$ by solving the tridiagonal system:

$$2(h_0 + h_1)c_1 + h_1 c_2 = 3\frac{y_2 - y_1}{h_1} - 3\frac{y_1 - y_0}{h_0}$$

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

$$h_{n-2}c_{n-2} + 2(h_{n-2} + h_{n-1})c_{n-1} = 3\frac{y_n - y_{n-1}}{h_{n-1}} - 3\frac{y_{n-1} - y_{n-2}}{h_{n-2}}$$

for $i = 2, \ldots, n-2$.

5. Calculate remaining coefficients:

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

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

Natural Cubic Spline

Calculate the natural cubic spline for the data points:

x_i	0	1	2	3
y_i	2	1	2	2

Following the algorithm:

1. $a_0 = 2$, $a_1 = 1$, $a_2 = 2$
2. $h_0 = 1$, $h_1 = 1$, $h_2 = 1$
3. $c_0 = 0$, $c_3 = 0$ (boundary conditions)
4. Solving the tridiagonal system:

$$2(h_0 + h_1)c_1 + h_1 c_2 = 3\frac{y_2 - y_1}{h_1} - 3\frac{y_1 - y_0}{h_0}$$
$$2(1+1)c_1 + 1c_2 = 3\frac{2-1}{1} - 3\frac{1-2}{1}$$
$$4c_1 + c_2 = 3 + 3 = 6$$

$$h_1 c_1 + 2(h_1 + h_2)c_2 = 3\frac{y_3 - y_2}{h_2} - 3\frac{y_2 - y_1}{h_1}$$
$$1c_1 + 2(1+1)c_2 = 3\frac{2-2}{1} - 3\frac{2-1}{1}$$
$$c_1 + 4c_2 = 0 - 3 = -3$$

Solving this system:

$$4c_1 + c_2 = 6$$
$$c_1 + 4c_2 = -3$$

We get $c_1 = \frac{27}{15} = 1.8$ and $c_2 = -\frac{12}{15} = -0.8$

5. Calculate b_i and d_i:

$$\begin{aligned} b_0 &= \frac{y_1 - y_0}{h_0} - \frac{h_0}{3}(c_1 + 2c_0) \\ &= \frac{1-2}{1} - \frac{1}{3}(1.8 + 2 \cdot 0) \\ &= -1 - 0.6 = -1.6 \end{aligned}$$

$$\begin{aligned} d_0 &= \frac{1}{3h_0}(c_1 - c_0) \\ &= \frac{1}{3 \cdot 1}(1.8 - 0) \\ &= 0.6 \end{aligned}$$

$$\begin{aligned} b_1 &= \frac{y_2 - y_1}{h_1} - \frac{h_1}{3}(c_2 + 2c_1) \\ &= \frac{2-1}{1} - \frac{1}{3}(-0.8 + 2 \cdot 1.8) \\ &= 1 - \frac{2.8}{3} \approx 0.0667 \end{aligned}$$

$$\begin{aligned} d_1 &= \frac{1}{3h_1}(c_2 - c_1) \\ &= \frac{1}{3 \cdot 1}(-0.8 - 1.8) \\ &= \frac{-2.6}{3} \approx -0.8667 \end{aligned}$$

$$b_2 = \frac{y_3 - y_2}{h_2} - \frac{h_2}{3}(c_3 + 2c_2)$$

Regression Analysis

Regression Problem

Given n data points (x_i, y_i), $i = 1, \ldots, n$, find a continuous function $f : \mathbb{R} \to \mathbb{R}$ that approximates the data points in a certain optimal sense, meaning that $f(x_i) \approx y_i$ for all $i = 1, \ldots, n$.

Ansatz Functions / Error Functional / Least Squares

Given a set F of continuous ansatz functions f on an interval $[a, b]$ and n data points (x_i, y_i), $i = 1, \ldots, n$.
An element $f \in F$ is called a regression function if the error functional

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

is minimized for f, i.e., $E(f) = \min\{E(g) | g \in F\}$.
This approach is called the method of least squares.

Linear Regression

Linear Regression Problem

Given n data points (x_i, y_i), $i = 1, \ldots, n$, and m basis functions $f_1, \ldots, f_m$ on an interval $[a, b]$. We choose F as the set of ansatz functions $f := \lambda_1 f_1 + \ldots + \lambda_m f_m$ with $\lambda_j \in \mathbb{R}$, so $F = \{f = \lambda_1 f_1 + \ldots + \lambda_m f_m | \lambda_j \in \mathbb{R}, j = 1, \ldots, m\}$.
The error functional is:

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

where

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

The system $A\lambda = y$ is called the error equation system.

Normal Equations

The equations

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

are called the normal equations of the linear regression problem.
The system of all normal equations is called the normal equation system and can be written as a linear system:

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

The solution $\lambda = (\lambda_1, \ldots, \lambda_m)^T$ of the normal equation system contains the desired parameters of the linear regression problem.

Solving Linear Regression

Steps for solving a linear regression problem:

1. Identify the basis functions $f_1, f_2, \ldots, f_m$ for your model
2. Set up the matrix A using the basis functions evaluated at each data point x_i
3. Form the normal equations: $A^T A\lambda = A^T y$
4. Solve for the parameter vector λ

Alternative method using QR decomposition

The normal equations can be ill-conditioned. A more stable approach is to use QR decomposition:

1. Compute the QR decomposition of $A = QR$, where Q is orthogonal ($Q^T Q = I_n$) and R is upper triangular
2. Solve the equivalent, but often better-conditioned system:

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

by back substitution

Linear Regression

Find the best-fit line $f(x) = ax + b$ for the data:

x_i	1	2	3	4
y_i	6	6.8	10	10.5

We have basis functions $f_1(x) = x$ and $f_2(x) = 1$, so the matrix A is:

$$A = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{pmatrix}$$

The normal equations are:

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

Computing:

$$A^T A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 30 & 10 \\ 10 & 4 \end{pmatrix}$$

$$A^T y = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 6 \\ 6.8 \\ 10 \\ 10.5 \end{pmatrix} = \begin{pmatrix} 91.6 \\ 33.3 \end{pmatrix}$$

The normal equations become:

$$\begin{pmatrix} 30 & 10 \\ 10 & 4 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 91.6 \\ 33.3 \end{pmatrix}$$

Solving this system gives $a = 1.67$ and $b = 4.15$, so our best-fit line is:

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

Nonlinear Regression

General Regression Problem

Given n data points (x_i, y_i), $i = 1, \ldots, n$, and the set F of ansatz functions $f_p = f_p(\lambda_1, \lambda_2, \ldots, \lambda_m, x)$ with m parameters $\lambda_j \in \mathbb{R}$, $j = 1, \ldots, m$.

The general regression problem consists of determining the m parameters $\lambda_1, \ldots, \lambda_m$ so that the error functional

$$E(f) = \sum_{i=1}^{n} (y_i - f_p(\lambda_1, \lambda_2, \ldots, \lambda_m, x_i))^2 = \|y - f(\lambda)\|_2^2$$

is minimal among all admissible parameter values.

Gauss-Newton Method

The Gauss-Newton method is an iterative procedure for solving nonlinear regression problems.

Algorithm

1. Define the function $g(\lambda) := y - f(\lambda)$ and the error functional $E(\lambda) := \|g(\lambda)\|_2^2$
2. Start with an initial parameter vector $\lambda^{(0)}$ close to the minimum of the error functional
3. For $k = 0, 1, 2, \ldots$ until convergence:
   - Compute $\delta^{(k)}$ as the solution of the linear least squares problem

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

     where $Dg(\lambda^{(k)})$ is the Jacobian matrix of g at $\lambda^{(k)}$
   - This is equivalent to solving the normal equations:

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

   - Update the parameter vector:

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

Damped Gauss-Newton Method

If the correction $\delta^{(k)}$ does not lead to a decrease in the error functional, the step size can be reduced:

1. Compute $\delta^{(k)}$ as in the standard method
2. Find the minimum $p \in \{0, 1, \ldots, p_{max}\}$ such that

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

3. If no such p is found, use $p = 0$
4. Update:

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

Nonlinear Regression

Fit the function $f(x) = ae^{bx}$ to the data:

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

We need to find parameters a and b that minimize:

$$E(f) = \sum_{i=1}^{5} (y_i - ae^{bx_i})^2$$

Using the Gauss-Newton method with initial values $a^{(0)} = 3$ and $b^{(0)} = -1$:

For the first iteration, we define $g(a, b) = y - f(a, b) = \begin{pmatrix} y_1 - ae^{bx_1} \\ \vdots \\ y_5 - ae^{bx_5} \end{pmatrix}$

The Jacobian is:

$$Dg(a, b) = \begin{pmatrix} -e^{bx_1} & -ax_1 e^{bx_1} \\ \vdots & \vdots \\ -e^{bx_5} & -ax_5 e^{bx_5} \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ -e^{-1} & -ae^{-1} \\ -e^{-2} & -2ae^{-2} \\ -e^{-3} & -3ae^{-3} \\ -e^{-4} & -4ae^{-4} \end{pmatrix}_{a=3, b=-1}$$

Solving the normal equations gives a correction vector, and after several iterations, we get the parameters $a \approx 2.98$ and $b \approx -1.00$.

Numerische Integration

Problemstellung

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

Aufgabe: Berechnen Sie $\int_2^4 \frac{1}{x}dx$ mit der summierten Trapezregel für $n = 4$.

Lösung:
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

Lösung:
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

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

Lösung:
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

Claude try 1

Numerical Integration

Introduction and Motivation

Numerical Integration

Numerical integration, also known as quadrature, refers to numerical approximation methods for evaluating definite integrals. While differentiation can be performed analytically for all differentiable functions, many integrals cannot be solved analytically, making numerical integration a crucial component of numerical methods.

Problem Statement

For a function $f : \mathbb{R} \to \mathbb{R}$, we want to numerically calculate the definite integral

$$I(f) = \int_a^b f(x)\,dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right) dt$$

The Gauss-Legendre formulas for $n = 1, 2, 3$ are:

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

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

$$G_3 f = \frac{b-a}{2}\left[\frac{5}{9} \cdot f\left(-\sqrt{0.6} \cdot \frac{b-a}{2} + \frac{b+a}{2}\right) + \frac{8}{9} \cdot f\left(\frac{b+a}{2}\right)\right] + \frac{b-a}{2}\left[\frac{5}{9} \cdot f\left(\sqrt{0.6} \cdot \frac{b-a}{2} + \frac{b+a}{2}\right)\right]$$

Romberg Integration

Romberg Integration

Romberg integration is an extrapolation technique that uses the trapezoidal rule with successively halved step sizes to obtain higher-order accuracy.

For the composite trapezoidal rule $Tf(h)$ approximating $I(f) = \int_a^b f(x)\,dx$:

Let $T_{j0} = Tf\left(\frac{b-a}{2^j}\right)$ for $j = 0, 1, \ldots, m$. Then, the extrapolations are defined by the recursion:

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

for $k = 1, 2, \ldots, m$ and $j = 0, 1, \ldots, m-k$.

These extrapolations T_{jk} have error order $2k+2$. The sequence of step sizes $h_j = \frac{b-a}{2^j}$ is called the Romberg sequence.

Romberg Integration Algorithm

Steps

1. For $j = 0, 1, \ldots, m$, compute T_{j0} using the composite trapezoidal rule with step size $h_j = \frac{b-a}{2^j}$:

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

   where $n_j = 2^j$ and $x_i = a + ih_j$.
2. For $k = 1, 2, \ldots, m$ and $j = 0, 1, \ldots, m-k$, compute the extrapolations:

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

3. The value T_{0m} provides the highest-order approximation to the integral.

Extrapolation Scheme

The extrapolations form a triangular pattern:

$$\begin{array}{ccccccc} & & & T_{00} & & & \\ & & & \searrow & & & \\ & & T_{10} & \to & T_{01} & & \\ & & & \searrow\nearrow & & \searrow & \\ & T_{20} & \to & T_{11} & \to & T_{02} & \\ & & \searrow\nearrow & & \searrow\nearrow & & \searrow \\ T_{30} & \to & T_{21} & \to & T_{12} & \to & T_{03} \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \end{array}$$

Efficiency

Each extrapolation significantly improves accuracy without requiring additional function evaluations. The first column requires $O(2^m)$ function evaluations, but each subsequent column only requires algebraic operations, making this method highly efficient for achieving high accuracy.

Romberg Integration

Approximate $\int_2^4 \frac{1}{x}\,dx$ using Romberg integration with $j = 0, 1, 2, 3$.

First, compute the first column T_{j0} using the trapezoidal rule with different step sizes:

For $j = 0, h_0 = \frac{4-2}{2^0} = 2, n_0 = 2^0 = 1$:

$$T_{00} = h_0 \cdot \frac{f(2)+f(4)}{2} = 2 \cdot \frac{\frac{1}{2}+\frac{1}{4}}{2} = 2 \cdot \frac{0.75}{2} = 0.75$$

For $j = 1, h_1 = 1, n_1 = 2$:

$$T_{10} = h_1 \cdot \left(\frac{f(2)+f(4)}{2} + f(3)\right) = 1 \cdot \left(\frac{\frac{1}{2}+\frac{1}{4}}{2} + \frac{1}{3}\right) = 1 \cdot \left(\frac{0.75}{2} + \frac{1}{3}\right)$$

For $j = 2, h_2 = 0.5, n_2 = 4$:

$$\begin{aligned} T_{20} &= h_2 \cdot \left(\frac{f(2)+f(4)}{2} + f(2.5) + f(3) + f(3.5)\right) \\ &= 0.5 \cdot \left(\frac{\frac{1}{2}+\frac{1}{4}}{2} + \frac{1}{2.5} + \frac{1}{3} + \frac{1}{3.5}\right) \\ &= 0.5 \cdot (0.375 + 0.4 + 0.3333 + 0.2857) \approx 0.6970 \end{aligned}$$

For $j = 3, h_3 = 0.25, n_3 = 8$:

$$T_{30} = 0.25 \cdot \left(\frac{f(2)+f(4)}{2} + f(2.25) + f(2.5) + \ldots + f(3.75)\right) \approx 0.6941$$

Now, compute the extrapolations:

For $k = 1, j = 0$:

$$T_{01} = \frac{4^1 \cdot T_{10} - T_{00}}{4^1 - 1} = \frac{4 \cdot 0.7083 - 0.75}{3} = \frac{2.8332 - 0.75}{3} \approx 0.6944$$

For $k = 1, j = 1$:

$$T_{11} = \frac{4 \cdot T_{21} - T_{10}}{3} = \frac{4 \cdot 0.6970 - 0.7083}{3} = \frac{2.788 - 0.7083}{3} \approx 0.6932$$

For $k = 1, j = 2$:

$$T_{21} = \frac{4 \cdot T_{31} - T_{20}}{3} = \frac{4 \cdot 0.6941 - 0.6970}{3} = \frac{2.7764 - 0.6970}{3} \approx 0.6931$$

For $k = 2, j = 0$:

$$T_{02} = \frac{4^2 \cdot T_{11} - T_{01}}{4^2 - 1} = \frac{16 \cdot 0.6932 - 0.6944}{15} = \frac{11.0912 - 0.6944}{15} \approx 0.6931$$

For $k = 2, j = 1$:

$$T_{12} = \frac{16 \cdot T_{21} - T_{11}}{15} = \frac{16 \cdot 0.6931 - 0.6932}{15} \approx 0.6931$$

For $k = 3, j = 0$:

$$T_{03} = \frac{4^3 \cdot T_{12} - T_{02}}{4^3 - 1} = \frac{64 \cdot 0.6931 - 0.6931}{63} \approx 0.6931$$

The final approximation $T_{03} \approx 0.6931$ is very close to the exact value $\ln(2) \approx 0.693147$. b f(x) dx on an interval $[a, b]$.

Numerical quadrature methods typically have the form

Newton-Cotes Formulas

Rectangle and Trapezoidal Rules

Rectangle Rule / Trapezoidal Rule

The rectangle rule (or midpoint rule) Rf and the trapezoidal rule Tf for approximating $\int_a^b f(x)\,dx$ are defined as:

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

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

The rectangle rule approximates the integral by the area of a rectangle with height equal to the function value at the midpoint, while the trapezoidal rule approximates the integral by the area of a trapezoid.

Composite Rectangle Rule / Composite Trapezoidal Rule

To improve accuracy, we can divide the interval $[a,b]$ into n subintervals $[x_i, x_{i+1}]$ of equal width $h = \frac{b-a}{n}$ with $x_i = a + ih$ for $i = 0, \ldots, n$ and sum the results.
The composite rectangle rule $Rf(h)$ and the composite trapezoidal rule $Tf(h)$ are given by:

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

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

Applying the Trapezoidal Rule

Approximate $\int_2^4 \frac{1}{x}\,dx$ using the composite trapezoidal rule with $n = 4$ subintervals.
The exact value is $\int_2^4 \frac{1}{x}\,dx = \ln(4) - \ln(2) = \ln(2) \approx 0.6931$.
We have:

$$h = \frac{b-a}{n} = \frac{4-2}{4} = 0.5$$
$$x_i = 2 + i \cdot 0.5 \quad \text{for } i = 0,1,2,3,4$$

So $x_0 = 2$, $x_1 = 2.5$, $x_2 = 3$, $x_3 = 3.5$, $x_4 = 4$.
Applying the composite trapezoidal rule:

$$\begin{aligned} Tf(h) &= h \cdot \left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right) \\ &= 0.5 \cdot \left(\frac{f(2)+f(4)}{2} + f(2.5) + f(3) + f(3.5)\right) \\ &= 0.5 \cdot \left(\frac{\frac{1}{2}+\frac{1}{4}}{2} + \frac{1}{2.5} + \frac{1}{3} + \frac{1}{3.5}\right) \\ &= 0.5 \cdot \left(\frac{0.5+0.25}{2} + 0.4 + 0.333... + 0.2857...\right) \\ &= 0.5 \cdot (0.375 + 1.0187...) = 0.5 \cdot 1.3937... \approx 0.6969 \end{aligned}$$

The approximation 0.6969 is close to the exact value 0.6931, with an error of about 0.0038.

Simpson's Rule

Simpson's Rule

Simpson's rule extends the idea of the trapezoidal rule by approximating $f(x)$ with a quadratic polynomial instead of a linear function. For an interval $[a,b]$, Simpson's rule is:

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

The composite Simpson's rule for n subintervals (where n must be even) is:

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

Relation between Rules

The composite Simpson's rule can be expressed as a weighted average of the composite trapezoidal and rectangle rules:

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

Error Analysis of Quadrature Methods

Error Bounds for Composite Rules

Let $f : [a,b] \to \mathbb{R}$ be sufficiently differentiable. Then:

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

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

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

These error bounds indicate that the rectangle and trapezoidal rules have convergence order $O(h^2)$, while Simpson's rule has convergence order $O(h^4)$.

Determining Optimal Step Size

To achieve a desired accuracy $\epsilon > 0$ when approximating an integral, we can determine the necessary step size h (or number of subintervals n) using the error bound formulas.

For the composite trapezoidal rule

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

$$h \le \sqrt{\frac{12\epsilon}{(b-a) \cdot \max_{x\in[a,b]} |f''(x)|}}$$

Since $h = \frac{b-a}{n}$, this gives:

$$n \ge (b-a) \cdot \sqrt{\frac{(b-a) \cdot \max_{x\in[a,b]} |f''(x)|}{12\epsilon}}$$

For the composite Simpson's rule

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

$$h \le \sqrt[4]{\frac{2880\epsilon}{(b-a) \cdot \max_{x\in[a,b]} |f^{(4)}(x)|}}$$

Since $h = \frac{b-a}{n}$, this gives:

$$n \ge (b-a) \cdot \sqrt[4]{\frac{(b-a) \cdot \max_{x\in[a,b]} |f^{(4)}(x)|}{2880\epsilon}}$$

Approach

1. Determine the maximum value of the relevant derivative of f on $[a,b]$
2. Calculate the minimum required n using the appropriate formula
3. If necessary, round up to the next integer (for Simpson's rule, round up to the next even integer)
4. Apply the quadrature method with this value of n

Determining Step Size

Find the number of subintervals needed to approximate $\int_0^{0.5} e^{-x^2}\,dx$ using the composite trapezoidal rule with an absolute error less than 10^{-5}.

First, we need to find $\max_{x\in[0,0.5]} |f''(x)|$ where $f(x) = e^{-x^2}$.

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

The maximum of $|f''(x)|$ on $[0, 0.5]$ occurs at $x = 0$ and equals $|-2 \cdot e^0| = 2$.

Using the trapezoidal rule error formula:

$$n \geq (b-a) \cdot \sqrt{\frac{(b-a) \cdot \max_{x\in[a,b]} |f''(x)|}{12\epsilon}}$$
$$= 0.5 \cdot \sqrt{\frac{0.5 \cdot 2}{12 \cdot 10^{-5}}}$$
$$= 0.5 \cdot \sqrt{\frac{1}{12 \cdot 10^{-5}}}$$
$$= 0.5 \cdot \sqrt{\frac{10^5}{12}}$$
$$\approx 0.5 \cdot 91.29 \approx 45.64$$

Rounding up, we need at least $n = 46$ subintervals to achieve the desired accuracy.

Gaussian Quadrature

Gaussian Quadrature

While Newton-Cotes formulas use equally spaced nodes, Gaussian quadrature methods allow for optimal placement of the nodes x_i and weights a_i in the quadrature formula

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

Gaussian quadrature formulas are designed to exactly integrate polynomials of as high a degree as possible. A Gaussian quadrature with n nodes can exactly integrate polynomials of degree up to $2n-1$, making them highly efficient.

Gauss-Legendre Formulas

The most common type of Gaussian quadrature is Gauss-Legendre quadrature, which is optimal for integrating functions on $[-1, 1]$. For an interval $[a, b]$, we can transform the integral via:

$$\int_a^b f(x)\,dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right) dt$$

The Gauss-Legendre formulas for $n = 1, 2, 3$ are:

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

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

$$G_3 f = \frac{b-a}{2}\left[\frac{5}{9} \cdot f\left(-\sqrt{0.6} \cdot \frac{b-a}{2} + \frac{b+a}{2}\right) + \frac{8}{9} \cdot f\left(\frac{b+a}{2}\right)\right]$$
$$+ \frac{b-a}{2}\left[\frac{5}{9} \cdot f\left(\sqrt{0.6} \cdot \frac{b-a}{2} + \frac{b+a}{2}\right)\right]$$

Gauss-Legendre Quadrature

Approximate $\int_0^{0.5} e^{-x^2}\,dx$ using the 3-point Gauss-Legendre formula.

First, we transform the integral:

$$\int_0^{0.5} e^{-x^2}\,dx = \frac{0.5-0}{2} \int_{-1}^{1} e^{-\left(\frac{0.5-0}{2}t + \frac{0+0.5}{2}\right)^2} dt = \frac{0.25}{2} \int_{-1}^{1} e^{-(0.125t-}$$

Using the 3-point Gauss-Legendre formula ($G_3 f$):

$$G_3 f = \frac{0.5-0}{2}\left[\frac{5}{9} \cdot e^{-\left(0.125\cdot(-\sqrt{0.6})+0.25\right)^2} + \frac{8}{9} \cdot e^{-(0.125\cdot 0+0.25)^2}\right]$$
$$+ \frac{0.5-0}{2}\left[\frac{5}{9} \cdot e^{-\left(0.125\cdot\sqrt{0.6}+0.25\right)^2}\right]$$
$$= 0.25 \cdot \left[\frac{5}{9} \cdot e^{-\left(0.25-0.125\cdot\sqrt{0.6}\right)^2} + \frac{8}{9} \cdot e^{-0.25^2} + \frac{5}{9} \cdot e^{-\left(0.25+0.125\right.}\right.$$
$$\approx 0.25 \cdot \left[\frac{5}{9} \cdot e^{-0.15^2} + \frac{8}{9} \cdot e^{-0.0625} + \frac{5}{9} \cdot e^{-0.35^2}\right]$$
$$\approx 0.25 \cdot \left[\frac{5}{9} \cdot 0.978 + \frac{8}{9} \cdot 0.939 + \frac{5}{9} \cdot 0.884\right]$$
$$\approx 0.25 \cdot 0.936 \approx 0.234$$

The exact value is approximately 0.2327, so this is a very accurate approximation with just 3 evaluation points.

Romberg Integration

Romberg Integration

Romberg integration is an extrapolation technique that uses the trapezoidal rule with successively halved step sizes to obtain higher-order accuracy.

For the composite trapezoidal rule $Tf(h)$ approximating $I(f) = \int_a^b f(x)\,dx$:

Let $T_{j0} = Tf\left(\frac{b-a}{2^j}\right)$ for $j = 0, 1, \ldots, m$. Then, the extrapolations are defined by the recursion:

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

for $k = 1, 2, \ldots, m$ and $j = 0, 1, \ldots, m-k$.

These extrapolations T_{jk} have error order $2k+2$. The sequence of step sizes $h_j = \frac{b-a}{2^j}$ is called the Romberg sequence.

Romberg Integration Algorithm

Steps

1. For $j = 0, 1, \ldots, m$, compute T_{j0} using the composite trapezoidal rule with step size $h_j = \frac{b-a}{2^j}$:

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

   where $n_j = 2^j$ and $x_i = a + ih_j$.
2. For $k = 1, 2, \ldots, m$ and $j = 0, 1, \ldots, m-k$, compute the extrapolations:

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

3. The value T_{0m} provides the highest-order approximation to the integral.

Extrapolation Scheme

The extrapolations form a triangular pattern:

$$\begin{array}{ccccccc}
T_{00} & & & & & & \\
\searrow & & & & & & \\
T_{10} & \to & T_{01} & & & & \\
\searrow & \nearrow & \searrow & & & & \\
T_{20} & \to & T_{11} & \to & T_{02} & & \\
\searrow & \nearrow & \searrow & \nearrow & \searrow & & \\
T_{30} & \to & T_{21} & \to & T_{12} & \to & T_{03} \\
\downarrow & & \downarrow & & \downarrow & & \downarrow
\end{array}$$

Efficiency

Each extrapolation significantly improves accuracy without requiring additional function evaluations. The first column requires $O(2^m)$ function evaluations, but each subsequent column only requires algebraic operations, making this method highly efficient for achieving high accuracy.

Relationship to Simpson's Rule

The first extrapolation step of Romberg integration (T_{01}) is equivalent to Simpson's rule:

$$T_{01} = \frac{4T_{10} - T_{00}}{3} = \frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b)) = Sf$$

Similarly, each column in the Romberg table corresponds to a higher-order integration method, with the k-th column having error order $O(h^{2k+2})$.

Romberg Integration

Approximate $\int_2^4 \frac{1}{x}\,dx$ using Romberg integration with $j = 0, 1, 2, 3$.
First, compute the first column T_{j0} using the trapezoidal rule with different step sizes:
For $j = 0, h_0 = \frac{4-2}{2^0} = 2, n_0 = 2^0 = 1$:

$$T_{00} = h_0 \cdot \frac{f(2) + f(4)}{2} = 2 \cdot \frac{\frac{1}{2} + \frac{1}{4}}{2} = 2 \cdot \frac{0.75}{2} = 0.75$$

For $j = 1, h_1 = 1, n_1 = 2$:

$$T_{10} = h_1 \cdot \left(\frac{f(2) + f(4)}{2} + f(3)\right) = 1 \cdot \left(\frac{\frac{1}{2} + \frac{1}{4}}{2} + \frac{1}{3}\right) = 1 \cdot \left(\frac{0.75}{2} + \frac{1}{3}\right)$$

For $j = 2, h_2 = 0.5, n_2 = 4$:

$$\begin{aligned} T_{20} &= h_2 \cdot \left(\frac{f(2) + f(4)}{2} + f(2.5) + f(3) + f(3.5)\right) \\ &= 0.5 \cdot \left(\frac{\frac{1}{2} + \frac{1}{4}}{2} + \frac{1}{2.5} + \frac{1}{3} + \frac{1}{3.5}\right) \\ &= 0.5 \cdot (0.375 + 0.4 + 0.3333 + 0.2857) \approx 0.6970 \end{aligned}$$

For $j = 3, h_3 = 0.25, n_3 = 8$:

$$\begin{aligned} T_{30} &= 0.25 \cdot \left(\frac{f(2) + f(4)}{2} + f(2.25) + f(2.5) + \ldots + f(3.75)\right) \\ &\approx 0.6941 \end{aligned}$$

Now, compute the extrapolations:
For $k = 1, j = 0$:

$$T_{01} = \frac{4^1 \cdot T_{10} - T_{00}}{4^1 - 1} = \frac{4 \cdot 0.7083 - 0.75}{3} = \frac{2.8332 - 0.75}{3} \approx 0.6944$$

For $k = 1, j = 1$:

$$T_{11} = \frac{4 \cdot T_{21} - T_{10}}{3} = \frac{4 \cdot 0.6970 - 0.7083}{3} = \frac{2.788 - 0.7083}{3} \approx 0.6932$$

For $k = 1, j = 2$:

$$T_{21} = \frac{4 \cdot T_{31} - T_{20}}{3} = \frac{4 \cdot 0.6941 - 0.6970}{3} = \frac{2.7764 - 0.6970}{3} \approx 0.6931$$

For $k = 2, j = 0$:

$$T_{02} = \frac{4^2 \cdot T_{11} - T_{01}}{4^2 - 1} = \frac{16 \cdot 0.6932 - 0.6944}{15} = \frac{11.0912 - 0.6944}{15} \approx 0.6931$$

For $k = 2, j = 1$:

$$T_{12} = \frac{16 \cdot T_{21} - T_{11}}{15} = \frac{16 \cdot 0.6931 - 0.6932}{15} \approx 0.6931$$

For $k = 3, j = 0$:

$$T_{03} = \frac{4^3 \cdot T_{12} - T_{02}}{4^3 - 1} = \frac{64 \cdot 0.6931 - 0.6931}{63} \approx 0.6931$$

The final approximation $T_{03} \approx 0.6931$ is very close to the exact value $\ln(2) \approx 0.693147$.

Special Considerations and Practical Implementation

Handling Singularities

When the integrand has singularities, standard quadrature methods may fail. Techniques to handle singularities include:

- Variable substitution to remove the singularity
- Using specialized quadrature formulas designed for specific types of singularities
- Breaking the interval at the singular point and applying appropriate methods on each subinterval

Adaptive Quadrature

Adaptive quadrature methods adjust the integration strategy based on the behavior of the integrand:

- Use smaller intervals where the integrand changes rapidly
- Use larger intervals where the integrand is smooth
- Recursively divide intervals until a desired accuracy is achieved

This approach can significantly improve efficiency while maintaining accuracy.

Multidimensional Integration

For integrals in two or more dimensions:

- Product rules apply one-dimensional methods repeatedly along each dimension
- Monte Carlo methods use random sampling and are particularly effective for high-dimensional integrals
- Adaptive strategies become especially important as the dimensionality increases

Implementation in Python

Using SciPy for Numerical Integration

Python's SciPy library provides several functions for numerical integration:

- `scipy.integrate.quad`: General-purpose integration using adaptive quadrature
- `scipy.integrate.fixed_quad`: Integration using fixed-order Gaussian quadrature
- `scipy.integrate.romberg`: Implementation of Romberg integration
- `scipy.integrate.trapz`: Implementation of the composite trapezoidal rule
- `scipy.integrate.simps`: Implementation of Simpson's rule

Example Usage

```python
from scipy import integrate
import numpy as np

# Define the function to integrate
def f(x):
    return np.exp(-x**2)

# Using quad (adaptive quadrature)
result, error = integrate.quad(f, 0, 0.5)
print(f"Result: {result}, Error estimate: {error}")

# Using fixed_quad (7-point Gaussian quadrature)
result = integrate.fixed_quad(f, 0, 0.5, n=7)[0]
print(f"Fixed-quad result: {result}")

# Using Romberg integration
result = integrate.romberg(f, 0, 0.5, tol=1e-8)
print(f"Romberg result: {result}")

# Using composite rules with array data
x = np.linspace(0, 0.5, 100)
y = f(x)
trapz_result = integrate.trapz(y, x)
simps_result = integrate.simps(y, x)
print(f"Trapz result: {trapz_result}")
print(f"Simpson result: {simps_result}")
```

Einführung in gewöhnliche Differentialgleichungen

Problemstellung

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.

Lösung:

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.

Lösung:
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 $p = 4$:

$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right)$$
$$k_4 = f(x_i + h, y_i + hk_3)$$
$$x_{i+1} = x_i + h$$
$$y_{i+1} = y_i + h \cdot \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Butcher-Schema

Runge-Kutta Verfahren werden durch Butcher-Schemata charakterisiert:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Runge-Kutta Verfahren anwenden

Schritt 1: Steigungen berechnen

$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right)$$
$$k_4 = f(x_i + h, y_i + hk_3)$$

Schritt 2: Gewichtetes Mittel bilden

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

Schritt 3: Nächsten Punkt berechnen

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

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 $y(t) = \frac{t^2+9}{2t}$.

Lösung:
Implementierung in Python:

```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 Hamiltonssche Systeme, geometrische Integratoren

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

Claude try 1

Ordinary Differential Equations

Introduction and Motivation

Ordinary Differential Equation

An ordinary differential equation (ODE) is an equation that contains derivatives of an unknown function with respect to a single independent variable.

A first-order ODE has the form:

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

where $y(t)$ is the unknown function, t is the independent variable, and f is a given function.

An n^{th}-order ODE has the form:

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

where $y^{(n)}$ denotes the n^{th} derivative of y.

Initial Value Problem

An initial value problem (IVP) for an n^{th}-order ODE specifies the function values and its first $n-1$ derivatives at a particular point x_0:

$$\begin{aligned} y^{(n)}(x) &= f(x, y(x), y'(x), ..., y^{(n-1)}(x)) \\ y(x_0) &= y_0 \\ y'(x_0) &= y_1 \\ &\vdots \\ y^{(n-1)}(x_0) &= y_{n-1} \end{aligned}$$

For a first-order ODE, an IVP simply specifies $y(x_0) = y_0$.

Examples from Natural Sciences

Free Fall

Free Fall

Consider an object falling freely in a vacuum near Earth's surface. If we denote the position by $s(t)$ (with positive direction upward), the motion is governed by:

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

where $g \approx 9.81$ m/s^2 is the gravitational acceleration, and $\ddot{s}(t)$ is the second derivative of position with respect to time (i.e., acceleration).
This is a second-order ODE, which can be solved by direct integration:

$$\begin{aligned}\ddot{s}(t) &= -g \\ \dot{s}(t) &= -gt + v_0 \\ s(t) &= -\frac{1}{2}gt^2 + v_0 t + s_0\end{aligned}$$

Here, $v_0 = \dot{s}(0)$ is the initial velocity, and $s_0 = s(0)$ is the initial height. For $s_0 = 1000$ m, $v_0 = 0$, and $g = 10$ m/s^2, the position after 3 seconds is:

$$\begin{aligned}s(3) &= -\frac{1}{2} \cdot 10 \cdot 3^2 + 0 \cdot 3 + 1000 \\ &= -45 + 1000 = 955 \text{ m}\end{aligned}$$

The object reaches the ground ($s = 0$) when:

$$-\frac{1}{2} \cdot 10 \cdot t^2 + 1000 = 0$$

$$t = \sqrt{\frac{2 \cdot 1000}{10}} = 14.14 \text{ s}$$

At impact, its velocity is:

$$\dot{s}(14.14) = -10 \cdot 14.14 = -141.4 \text{ m/s} \approx -509.1 \text{ km/h}$$

Harmonic Oscillator

Harmonic Oscillator

The motion of a mass m attached to a spring with spring constant c is governed by:

$$m\ddot{x} = -cx$$

where x is displacement from equilibrium.
This equation can be rewritten as:

$$\ddot{x} = -\frac{c}{m}x = -\omega_0^2 x$$

where $\omega_0 = \sqrt{\frac{c}{m}}$ is the natural frequency.
The general solution is:

$$x(t) = A\sin(\omega_0 t + \phi)$$

where A is the amplitude and ϕ is the phase shift, determined by initial conditions.
The position $x(t)$ oscillates with period $T = \frac{2\pi}{\omega_0}$, demonstrating the characteristic behavior of simple harmonic motion.

Direction Fields

Direction Field

A direction field (or slope field) is a graphical representation of a first-order ODE $y' = f(x, y)$. At each point (x, y) in the plane, a small line segment with slope $f(x, y)$ is drawn, indicating the direction a solution curve would take through that point.
Direction fields help visualize the behavior of solutions without actually solving the ODE. Solution curves to the ODE are always tangent to the direction field at each point.

Direction Field Example

Consider the ODE $y' = x^2 + 0.1y$. The direction field shows small line segments with slope $x^2 + 0.1y$ at various points in the (x, y)-plane.
For example:

- At point $(-1, 1)$, the slope is $(-1)^2 + 0.1 \cdot 1 = 1.1$
- At point $(0.5, 1)$, the slope is $(0.5)^2 + 0.1 \cdot 1 = 0.35$

Solution curves pass through these points tangent to these slopes, gradually revealing the complete solution path.

Numerical Methods for First-Order ODEs

Euler's Method

Euler's Method

Euler's method is the simplest numerical approach for solving first-order IVPs. It approximates the solution by using the tangent line at each step.

Problem Statement

Given an IVP $y' = f(x, y)$ with $y(a) = y_0$ on interval $[a, b]$.

Algorithm

1. Choose a step size $h = \frac{b-a}{n}$ for some integer $n > 0$
2. Set $x_0 = a$ and calculate $x_i = a + ih$ for $i = 1, 2, \ldots, n$
3. Set $y_0 = y(a)$
4. For $i = 0, 1, \ldots, n-1$, compute:

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

Error Analysis

The local truncation error (error after one step) is $O(h^2)$, and the global truncation error (total accumulated error) is $O(h)$.

Interpretation

Euler's method follows the tangent line at (x_i, y_i) for a distance h. It approximates the curve by a sequence of short line segments.

Euler's Method Application

Solve the IVP $y' = t^2 + 0.1y$ with $y(-1.5) = 0$ on $[-1.5, 1.5]$ using Euler's method with $n = 5$ steps.
Step size: $h = \frac{1.5-(-1.5)}{5} = 0.6$
Points: $x_0 = -1.5, x_1 = -0.9, x_2 = -0.3, x_3 = 0.3, x_4 = 0.9, x_5 = 1.5$
Computations:

$y_0 = 0$

$y_1 = y_0 + h \cdot f(x_0, y_0) = 0 + 0.6 \cdot ((-1.5)^2 + 0.1 \cdot 0) = 0.6 \cdot 2.25 = 1.35$

$y_2 = y_1 + h \cdot f(x_1, y_1) = 1.35 + 0.6 \cdot ((-0.9)^2 + 0.1 \cdot 1.35)$
$= 1.35 + 0.6 \cdot (0.81 + 0.135) = 1.35 + 0.6 \cdot 0.945 = 1.917$

Continuing this process gives the approximate solution at the specified points.

Midpoint Method

Midpoint Method

The midpoint method (also called the modified Euler method or the second-order Runge-Kutta method) improves on Euler's method by using the slope at the midpoint of each interval.

Problem Statement

Given an IVP $y' = f(x, y)$ with $y(a) = y_0$ on interval $[a, b]$.

Algorithm

1. Choose a step size $h = \frac{b-a}{n}$ for some integer $n > 0$
2. Set $x_0 = a$ and calculate $x_i = a + ih$ for $i = 1, 2, \ldots, n$
3. Set $y_0 = y(a)$
4. For $i = 0, 1, \ldots, n-1$:
   - Calculate midpoint values:

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

   - Update using the midpoint slope:

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

Error Analysis

Both the local and global truncation errors are better than Euler's method, with a global error of $O(h^2)$.

Midpoint Method Application

Solve the IVP $y' = t^2 + 0.1y$ with $y(-1.5) = 0$ on $[-1.5, 1.5]$ using the midpoint method with $n = 5$ steps.

Step size: $h = \frac{1.5-(-1.5)}{5} = 0.6$

First step:

$$\begin{aligned}
x_{h/2} &= x_0 + \frac{h}{2} = -1.5 + 0.3 = -1.2 \\
y_{h/2} &= y_0 + \frac{h}{2} \cdot f(x_0, y_0) = 0 + 0.3 \cdot ((-1.5)^2 + 0.1 \cdot 0) \\
&= 0.3 \cdot 2.25 = 0.675 \\
y_1 &= y_0 + h \cdot f(x_{h/2}, y_{h/2}) \\
&= 0 + 0.6 \cdot ((-1.2)^2 + 0.1 \cdot 0.675) \\
&= 0.6 \cdot (1.44 + 0.0675) \\
&= 0.6 \cdot 1.5075 = 0.9045
\end{aligned}$$

Similar calculations are performed for each subsequent step.

Heun's Method (Modified Euler Method)

Heun's Method

Heun's method, also known as the improved Euler method, uses the average of the slopes at the beginning and end of each interval.

Problem Statement

Given an IVP $y' = f(x, y)$ with $y(a) = y_0$ on interval $[a, b]$.

Algorithm

1. Choose a step size $h = \frac{b-a}{n}$ for some integer $n > 0$
2. Set $x_0 = a$ and calculate $x_i = a + ih$ for $i = 1, 2, \ldots, n$
3. Set $y_0 = y(a)$
4. For $i = 0, 1, \ldots, n-1$:
   - Calculate predictor (Euler step):

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

   - Calculate corrector (average slope):

$$k_2 = f(x_{i+1}, y_{i+1}^{Euler})$$
$$y_{i+1} = y_i + h \cdot \frac{k_1 + k_2}{2}$$

Error Analysis

Like the midpoint method, Heun's method has a global error of $O(h^2)$, making it significantly more accurate than Euler's method.

Heun's Method Application

Solve the IVP $y' = t^2 + 0.1y$ with $y(-1.5) = 0$ on $[-1.5, 1.5]$ using Heun's method with $n = 5$ steps.

Step size: $h = \frac{1.5-(-1.5)}{5} = 0.6$

First step:

$$\begin{aligned}
k_1 &= f(x_0, y_0) = (-1.5)^2 + 0.1 \cdot 0 = 2.25 \\
y_1^{Euler} &= y_0 + h \cdot k_1 = 0 + 0.6 \cdot 2.25 = 1.35 \\
k_2 &= f(x_1, y_1^{Euler}) = (-0.9)^2 + 0.1 \cdot 1.35 = 0.81 + 0.135 = 0.945 \\
y_1 &= y_0 + h \cdot \frac{k_1 + k_2}{2} \\
&= 0 + 0.6 \cdot \frac{2.25 + 0.945}{2} \\
&= 0.6 \cdot 1.5975 = 0.9585
\end{aligned}$$

Similar calculations are performed for each subsequent step.

Runge-Kutta Methods

Runge-Kutta Methods

Runge-Kutta methods are a family of iterative methods for approximating solutions to ODEs, with varying orders of accuracy. They evaluate the function $f(x, y)$ at several points within each step to build a more accurate approximation.

Fourth-Order Runge-Kutta Method

Classical Fourth-Order Runge-Kutta Method

The most commonly used Runge-Kutta method is the fourth-order method (RK4), which provides an excellent balance of accuracy and computational efficiency.

Problem Statement

Given an IVP $y' = f(x, y)$ with $y(a) = y_0$ on interval $[a, b]$.

Algorithm

1. Choose a step size $h = \frac{b-a}{n}$ for some integer $n > 0$
2. Set $x_0 = a$ and calculate $x_i = a + ih$ for $i = 1, 2, \ldots, n$
3. Set $y_0 = y(a)$
4. For $i = 0, 1, \ldots, n-1$:

$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right)$$
$$k_4 = f(x_i + h, y_i + hk_3)$$
$$y_{i+1} = y_i + h \cdot \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Error Analysis

The local truncation error is $O(h^5)$, and the global truncation error is $O(h^4)$, making RK4 significantly more accurate than lower-order methods for the same step size.

Interpretation

RK4 combines four evaluations of the function f at different points to create a weighted average slope. This approach effectively samples the behavior of the solution over the interval to achieve higher accuracy.

Fourth-Order Runge-Kutta Method

Solve the IVP $y' = t^2 + 0.1y$ with $y(-1.5) = 0$ on $[-1.5, 1.5]$ using the RK4 method with $n = 5$ steps.
Step size: $h = \frac{1.5-(-1.5)}{5} = 0.6$
First step:

$$
\begin{aligned}
k_1 &= f(x_0, y_0) = (-1.5)^2 + 0.1 \cdot 0 = 2.25 \\
k_2 &= f\left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right) \\
&= f(-1.5 + 0.3, 0 + 0.3 \cdot 2.25) \\
&= f(-1.2, 0.675) \\
&= (-1.2)^2 + 0.1 \cdot 0.675 = 1.44 + 0.0675 = 1.5075 \\
k_3 &= f\left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_2\right) \\
&= f(-1.2, 0 + 0.3 \cdot 1.5075) \\
&= f(-1.2, 0.45225) \\
&= 1.44 + 0.1 \cdot 0.45225 = 1.44 + 0.045225 = 1.485 \\
k_4 &= f(x_0 + h, y_0 + hk_3) \\
&= f(-1.5 + 0.6, 0 + 0.6 \cdot 1.485) \\
&= f(-0.9, 0.891) \\
&= (-0.9)^2 + 0.1 \cdot 0.891 = 0.81 + 0.0891 = 0.8991 \\
y_1 &= y_0 + h \cdot \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
&= 0 + 0.6 \cdot \frac{1}{6}(2.25 + 2 \cdot 1.5075 + 2 \cdot 1.485 + 0.8991) \\
&= 0.6 \cdot \frac{1}{6}(2.25 + 3.015 + 2.97 + 0.8991) \\
&= 0.6 \cdot \frac{9.1341}{6} = 0.6 \cdot 1.5224 = 0.9134
\end{aligned}
$$

Similar calculations would be performed for each subsequent step.

General Runge-Kutta Methods

General Runge-Kutta Method

A general s-stage Runge-Kutta method for solving an ODE $y' = f(x, y)$ is defined by:

$$
k_n = f\left(x_i + c_n h, y_i + h \sum_{m=1}^{n-1} a_{nm} k_m\right), \text{ for } n = 1, 2, \ldots, s
$$

$$
y_{i+1} = y_i + h \sum_{n=1}^{s} b_n k_n
$$

where $s \in \mathbb{N}$ is the number of stages, and a_{nm}, b_n, and c_n are constants. The choice of these constants determines the specific method and its order of accuracy.

Butcher Tableau

The coefficients of a Runge-Kutta method are typically presented in a Butcher tableau:

$$
\begin{array}{c|cccc}
c_1 & & & & \\
c_2 & a_{21} & & & \\
c_3 & a_{31} & a_{32} & & \\
\vdots & \vdots & \vdots & \ddots & \\
c_s & a_{s1} & a_{s2} & \ldots & a_{s,s-1} \\
\hline
 & b_1 & b_2 & \ldots & b_s
\end{array}
$$

For example, the classical fourth-order Runge-Kutta method (RK4) has the Butcher tableau:

$$
\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
1/2 & 0 & 1/2 & & \\
1 & 0 & 0 & 1 & \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$

Systems of Differential Equations

Reduction of Higher-Order ODEs to First-Order Systems

Converting an n^{th}-order ODE to a System of First-Order ODEs

Any n^{th}-order ODE can be converted into a system of n first-order ODEs, which can then be solved using numerical methods for first-order systems.

Steps

1. Given an n^{th}-order ODE $y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)})$
2. Solve for the highest-order derivative: $y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)})$
3. Introduce new variables:

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

4. Derive a system of first-order ODEs:

$$
\begin{aligned}
z_1' &= z_2 \\
z_2' &= z_3 \\
&\vdots \\
z_{n-1}' &= z_n \\
z_n' &= f(x, z_1, z_2, \ldots, z_n)
\end{aligned}
$$

5. Express in vector form:

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

where $\mathbf{z} = (z_1, z_2, \ldots, z_n)^T$

6. Apply numerical methods for first-order systems to solve this system

Converting a Third-Order ODE to a System

Consider the third-order ODE:

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

with initial conditions $y(0) = 2$, $y'(0) = y''(0) = 0$.
Step 1: Solve for the highest-order derivative:

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

Step 2: Introduce new variables:

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

Step 3: Derive the system of first-order ODEs:

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

Step 4: Initial conditions for the system:

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

This system can now be solved using numerical methods for first-order systems.

Solving Systems of First-Order ODEs

Numerical Methods for Systems of ODEs

The numerical methods previously discussed for scalar first-order ODEs can be extended to systems of first-order ODEs.

Problem Statement

Consider a system of n first-order ODEs:

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

with initial condition $\mathbf{y}(x_0) = \mathbf{y}_0$, where $\mathbf{y}(x) \in \mathbb{R}^n$ and $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$.

Applying Numerical Methods

For any numerical method defined for scalar ODEs:

$$y_{i+1} = y_i + h \cdot \text{Steigung}$$

the extension to systems is:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h \cdot \mathbf{Steigung}$$

where:

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

For example, Euler's method becomes:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h \cdot \mathbf{f}(x_i, \mathbf{y}_i)$$

The fourth-order Runge-Kutta method becomes:

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(x_i, \mathbf{y}_i) \\ \mathbf{k}_2 &= \mathbf{f}\left(x_i + \frac{h}{2}, \mathbf{y}_i + \frac{h}{2}\mathbf{k}_1\right) \\ \mathbf{k}_3 &= \mathbf{f}\left(x_i + \frac{h}{2}, \mathbf{y}_i + \frac{h}{2}\mathbf{k}_2\right) \\ \mathbf{k}_4 &= \mathbf{f}(x_i + h, \mathbf{y}_i + h\mathbf{k}_3) \\ \mathbf{y}_{i+1} &= \mathbf{y}_i + h \cdot \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \end{aligned}$$

Stability of Numerical Methods

Stability

Stability refers to the behavior of a numerical method when applied to an ODE. A method is stable if small errors in the computation do not lead to unbounded growth in the numerical solution.
For explicit methods, stability often depends on the step size h. A method is said to be conditionally stable if it is stable only for certain ranges of h.

Stability Analysis

Consider the test equation $y' = -\alpha y$ with $\alpha > 0$ and initial condition $y(0) = 1$. The exact solution is $y(x) = e^{-\alpha x}$, which decays to zero as x increases.
Applying Euler's method:

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

Iterating recursively:

$$y_{i+1} = (1 - h\alpha)^{i+1} y_0 = (1 - h\alpha)^{i+1}$$

For the numerical solution to decay (as the exact solution does), we need $|1 - h\alpha| < 1$, which leads to the condition $0 < h\alpha < 2$.
Thus, for stability with Euler's method, the step size must satisfy $h < \frac{2}{\alpha}$.

Stability Function and Stability Interval

When applying a numerical method to the test equation $y' = -\alpha y$, if the numerical solution can be written in the form:

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

then $g(z)$ is called the stability function of the method (with $z = h\alpha$). The open interval $z \in (0, \alpha)$ for which $|g(z)| < 1$ is called the stability interval of the method.

Advanced Topics

Single-Step vs. Multi-Step Methods

Single-Step vs. Multi-Step Methods

Single-step methods (like Euler and Runge-Kutta) calculate y_{i+1} using only information from the previous point (x_i, y_i).
Multi-step methods use information from several previous points $(x_{i-k}, y_{i-k}), \ldots, (x_i, y_i)$ to calculate y_{i+1}. Examples include Adams-Bashforth methods, like:

$$y_{i+1} = y_i + \frac{h}{12}(23f(x_i, y_i) - 16f(x_{i-1}, y_{i-1}) + 5f(x_{i-2}, y_{i-2}))$$

Multi-step methods can be more efficient but require special starting procedures since multiple previous points are needed.

Implicit vs. Explicit Methods

Implicit vs. Explicit Methods

Explicit methods express y_{i+1} directly in terms of previously computed values. All methods discussed so far have been explicit.
Implicit methods involve y_{i+1} on both sides of the equation, requiring the solution of an (often nonlinear) equation at each step. The implicit Euler method has the form:

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

While more computationally intensive, implicit methods often have better stability properties and can handle ßtiffÖDEs more effectively.

Stiff ODEs and Adaptive Step Size

Stiff ODEs

A stiff ODE exhibits behavior at widely varying time scales, making numerical solution challenging. For stiff equations, explicit methods require extremely small step sizes to maintain stability, making them impractical.
Implicit methods are generally preferred for stiff problems because of their superior stability properties.

Adaptive Step Size Control

Instead of using a fixed step size h, adaptive methods adjust the step size based on local error estimates:

- Smaller steps are used in regions where the solution changes rapidly
- Larger steps are used in regions where the solution changes slowly

This approach improves efficiency while maintaining accuracy.

Python Implementation

SciPy's ODE Solvers

Python's SciPy library provides several functions for solving ODEs, including:

- `scipy.integrate.solve_ivp()`: A comprehensive function that supports various methods including RK45, Radau, BDF, and LSODA
- Methods are specified as parameters, along with options for error control and step size adaptation

SciPy can handle both single equations and systems of equations, with support for events and dense output options.