Höhere Mathematik

Jil Zerndt, Lucien Perret
May 2024

Rechnerarithmetik

Zahlendarstellung

Maschinenzahlen Eine maschinendarstellbare Zahl zur Basis B ist ein Element der Menge:

$$M = \{x \in \mathbb{R} \mid x = \pm 0.m_1 m_2 m_3 \dots m_n \cdot B^{\pm e_1 e_2 \dots e_l}\} \cup \{0\}$$

- $m_1 \neq 0$ (Normalisierungsbedingung)
- $m_i, e_i \in \{0, 1, \dots, B-1\}$ für $i \neq 0$
- $B \in \mathbb{N}, B > 1$ (Basis)

Zahlenwert Der Wert $\hat{\omega}$ einer Maschinenzahl berechnet sich durch:

$$\hat{\omega} = \sum_{i=1}^{n} m_i B^{\hat{e}-i}, \quad \text{mit} \quad \hat{e} = \sum_{i=1}^{l} e_i B^{l-i}$$

Werteberechnung Berechnung einer vierstelligen Zahl zur Basis 4:

$\underbrace{0.3211}_{n=4} \cdot \underbrace{4^{12}}_{l=2}$

Exponent: $\hat{e} = 1 \cdot 4^1 + 2 \cdot 4^0 = 6$

Wert: $\hat{\omega} = 3 \cdot 4^3 + 2 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0 = 57$

IEEE-754 Standard definiert zwei wichtige Gleitpunktformate:

Single Precision (32 Bit)	**Double Precision** (64 Bit)
Vorzeichen(V): 1 Bit	Vorzeichen(V): 1 Bit
Exponent(E): 8 Bit (Bias 127)	Exponent(E): 11 Bit (Bias 1023)
Mantisse(M):	Mantisse(M):
23 Bit + 1 hidden bit	52 Bit + 1 hidden bit

Darstellungsbereich Für jedes Gleitpunktsystem existieren:

- Grösste darstellbare Zahl: $x_{\max} = (1 - B^{-n}) \cdot B^{e_{\max}}$
- Kleinste darstellbare positive Zahl: $x_{\min} = B^{e_{\min}-1}$

Approximations- und Rundungsfehler

Fehlerarten Sei $\tilde{x}$ eine Näherung des exakten Wertes x:

Absoluter Fehler: $|\tilde{x} - x|$

Relativer Fehler: $\left|\frac{\tilde{x} - x}{x}\right|$ bzw. $\frac{|\tilde{x} - x|}{|x|}$ für $x \neq 0$

Maschinengenauigkeit eps ist die kleinste positive Zahl, für die gilt:

Allgemein: $\text{eps} := \frac{B}{2} \cdot B^{-n}$

Dezimal: $\text{eps}_{10} := 5 \cdot 10^{-n}$

Sie begrenzt den maximalen relativen Rundungsfehler: $\left|\frac{rd(x) - x}{x}\right| \leq \text{eps}$

Rundungseigenschaften Für alle $x \in \mathbb{R}$ mit $|x| \geq x_{\min}$ gilt:

Absoluter Fehler: $|rd(x) - x| \leq \frac{B}{2} \cdot B^{e-n-1}$

Relativer Fehler: $\left|\frac{rd(x) - x}{x}\right| \leq \text{eps}$

Fehlerfortpflanzung

Konditionierung Die Konditionszahl K beschreibt die relative Fehlervergrösserung bei Funktionsauswertungen:

$$K := \frac{|f'(x)| \cdot |x|}{|f(x)|}$$

- $K \leq 1$: gut konditioniert
- $K > 1$: schlecht konditioniert
- $K \gg 1$: sehr schlecht konditioniert

Fehlerfortpflanzung Für f (differenzierbar) gilt näherungsweise:

Absoluter Fehler: $|f(\tilde{x}) - f(x)| \approx |f'(x)| \cdot |\tilde{x} - x|$

Relativer Fehler: $\frac{|f(\tilde{x}) - f(x)|}{|f(x)|} \approx K \cdot \frac{|\tilde{x} - x|}{|x|}$

Analyse der Fehlerfortpflanzung einer Funktion

1. Berechnen Sie $f'(x)$
2. Bestimmen Sie die Konditionszahl K
3. Schätzen Sie den absoluten Fehler ab
4. Schätzen Sie den relativen Fehler ab
5. Beurteilen Sie die Konditionierung anhand von K

$$\underbrace{|f(\tilde{x}) - f(x)|}_{\text{absoluter Fehler von } f(x)} \approx \left|f'(x)\right| \cdot \underbrace{|\tilde{x} - x|}_{\text{absoluter Fehler von } x}$$

$$\underbrace{\frac{|f(\tilde{x}) - f(x)|}{|f(x)|}}_{\text{relativer Fehler von } f(x)} \approx \underbrace{\frac{\left|f'(x)\right| \cdot |x|}{|f(x)|}}_{\text{Konditionszahl } K} \cdot \underbrace{\frac{|\tilde{x} - x|}{|x|}}_{\text{relativer Fehler von } x}$$

Fehleranalyse Beispiel: Fehleranalyse von $f(x) = \sin(x)$

1. $f'(x) = \cos(x)$
2. $K = \frac{|x \cos(x)|}{|\sin(x)|}$
3. Für $x \to 0$: $K \to 1$ (gut konditioniert)
4. Für $x \to \pi$: $K \to \infty$ (schlecht konditioniert)
5. Der absolute Fehler wird nicht vergrössert, da $|\cos(x)| \leq 1$

Praktische Fehlerquellen der Numerik

Kritische Operationen häufigste Fehlerquellen:

- Auslöschung bei Subtraktion ähnlich großer Zahlen
- Überlauf (overflow) bei zu großen Zahlen
- Unterlauf (underflow) bei zu kleinen Zahlen
- Verlust signifikanter Stellen durch Rundung

Vermeidung von Auslöschung

1. Identifizieren Sie Subtraktionen ähnlich großer Zahlen
2. Suchen Sie nach algebraischen Umformungen
3. Prüfen Sie alternative Berechnungswege
4. Verwenden Sie Taylorentwicklungen für kleine Werte

Auslöschung bei der Berechnung von $\sqrt{x^2+1} - 1$:

Für kleine x führt die direkte Berechnung zu Auslöschung:

Für $x = 10^{-8}$: $\sqrt{10^{-16}+1} - 1 \approx 1.000000000 - 1 = 0$

Korrekte Lösung durch Umformung: $\sqrt{x^2+1} - 1 = \frac{x^2}{\sqrt{x^2+1}+1}$

Auslöschung Bei der Subtraktion fast gleich großer Zahlen können signifikante Stellen verloren gehen. Beispiel:

- $1.234567 - 1.234566 = 0.000001$
- Aus 7 signifikanten Stellen wird 1 signifikante Stelle

Analyse von Algorithmen

Fehlerakkumulation Bei n aufeinanderfolgenden Operationen mit relativen Fehlern $\leq \varepsilon$ gilt für den Gesamtfehler:

- Best case: $\mathcal{O}(n\varepsilon)$ bei gleichverteilten Fehlern
- Worst case: $\mathcal{O}(2^n\varepsilon)$ bei systematischen Fehlern

Numerische Stabilität eines Algorithmus

- Kleine Eingabefehler führen zu kleinen Ausgabefehlern
- Rundungsfehler akkumulieren sich nicht übermäßig
- Konditionszahl des Problems wird nicht künstlich verschlechtert

Instabilität bei rekursiver Berechnung: (Fibonacci-Zahlen)

```python
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)
```

Exponentielles Wachstum der Operationen → Fehlerfortpflanzung

Stabilitätsanalyse Schritte zur Analyse der numerischen Stabilität:

1. Bestimmen Sie kritische Operationen
2. Schätzen Sie Rundungsfehler pro Operation ab
3. Analysieren Sie die Fehlerfortpflanzung
4. Berechnen Sie die worst-case Fehlerschranke
5. Vergleichen Sie alternative Implementierungen

Praktische Implementierungen

Implementierungsgenauigkeit eines Algorithmus

- Relative Genauigkeit der Ausgabe
- Maximale Anzahl korrekter Dezimalstellen
- Stabilität gegenüber Eingabefehlern

Robuste Implementierung von Algorithmen

1. Verwenden Sie stabile Grundoperationen
2. Vermeiden Sie Differenzen ähnlich großer Zahlen
3. Prüfen Sie auf Über- und Unterlauf
4. Implementieren Sie Fehlerkontrollen
5. Dokumentieren Sie numerische Einschränkungen

Robuste Implementation Beispiel: Quadratische Gleichung

```python
def quadratic_stable(a, b, c):
    # ax^2 + bx + c = 0
    if a == 0:
        return [-c/b] if b != 0 else []

    # Calculate discriminant
    disc = b*b - 4*a*c
    if disc < 0:
        return []

    # Choose numerically stable formula
    if b >= 0:
        q = -0.5*(b + sqrt(disc))
    else:
        q = -0.5*(b - sqrt(disc))
    x1 = q/a
    x2 = c/(q)

    return sorted([x1, x2])
```

Numerische Lösung von Nullstellenproblemen

NSP: Nullstellenproblem, NS: Nullstelle

Fixpunktgleichung ist eine Gleichung der Form:

$$F(x) = x$$

Die Lösungen $\bar{x}$, für die $F(\bar{x}) = \bar{x}$ erfüllt ist, heissen Fixpunkte.

Fixpunktiteration

Grundprinzip der Fixpunktiteration

Gegeben sei $F : [a, b] \to \mathbb{R}$ mit $x_0 \in [a, b]$. Die rekursive Folge

$$x_{n+1} \equiv F(x_n), \quad n = 0, 1, 2, \ldots$$

heisst Fixpunktiteration von F zum Startwert x_0.

Konvergenzverhalten

Sei $F : [a, b] \to \mathbb{R}$ mit stetiger Ableitung F' und $\bar{x} \in [a, b]$ ein Fixpunkt von F. Dann gilt für die Fixpunktiteration $x_{n+1} = F(x_n)$:

Anziehender Fixpunkt:

$$|F'(\bar{x})| < 1$$

x_n konvergiert gegen $\bar{x}$, falls x_0 nahe genug bei $\bar{x}$

Abstossender Fixpunkt:

$$|F'(\bar{x})| > 1$$

x_n konvergiert für keinen Startwert $x_0 \neq \bar{x}$

Banachscher Fixpunktsatz

Sei $F : [a, b] \to [a, b]$ und es existiere eine Konstante α mit:

- $0 < \alpha < 1$ (Lipschitz-Konstante)
- $|F(x) - F(y)| \leq \alpha |x - y|$ für alle $x, y \in [a, b]$

Dann gilt:

- F hat genau einen Fixpunkt $\bar{x}$ in $[a, b]$
- Die Fixpunktiteration konvergiert gegen $\bar{x}$ für alle $x_0 \in [a, b]$
- Fehlerabschätzungen:

$$\text{a-priori: } |x_n - \bar{x}| \leq \frac{\alpha^n}{1 - \alpha} \cdot |x_1 - x_0|$$

$$\text{a-posteriori: } |x_n - \bar{x}| \leq \frac{\alpha}{1 - \alpha} \cdot |x_n - x_{n-1}|$$

Konvergenznachweis für Fixpunktiteration

So überprüfen Sie, ob eine Fixpunktiteration konvergiert:

1. Prüfen Sie, ob $F : [a, b] \to [a, b]$ gilt: $F(a) > a$ und $F(b) < b$
2. Bestimmen Sie $\alpha = \max_{x \in [a,b]} |F'(x)|$
3. Prüfen Sie, ob $\alpha < 1$
4. Berechnen Sie die nötigen Iterationen für Toleranz tol:

$$n \geq \frac{\ln(\frac{tol \cdot (1-\alpha)}{|x_1 - x_0|})}{\ln \alpha}$$

Fixpunktiteration Nullstellen von $p(x) = x^3 - x + 0.3$

Fixpunktgleichung: $x_{n+1} = F(x_n) = x_n^3 + 0.3$

1. $F'(x) = 3x^2$ steigt monoton
2. Für $I = [0, 0.5]$: $F(0) = 0.3 > 0$, $F(0.5) = 0.425 < 0.5$
3. $\alpha = \max_{x \in [0,0.5]} |3x^2| = 0.75 < 1$
4. Konvergenz für Startwerte in $[0, 0.5]$ gesichert

Newton-Verfahren

Grundprinzip Newton-Verfahren

Approximation der NS durch sukzessive Tangentenberechnung:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Konvergiert, wenn für alle x im relevanten Intervall gilt:

$$\left| \frac{f(x) \cdot f''(x)}{[f'(x)]^2} \right| < 1$$

Newton-Verfahren anwenden

So finden Sie eine Nullstelle mit dem Newton-Verfahren:

1. Funktion $f(x)$ und Ableitung $f'(x)$ aufstellen
2. Geeigneten Startwert x_0 nahe der Nullstelle wählen
3. Iterieren bis zur gewünschten Genauigkeit: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
4. Konvergenz prüfen durch Vergleich aufeinanderfolgender Werte

Vereinfachtes Newton-Verfahren

Alternative Variante mit konstanter Ableitung:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_0)}$$

Konvergiert langsamer, aber benötigt weniger Rechenaufwand.

Sekantenverfahren

Alternative zum Newton-Verfahren ohne Ableitungsberechnung. Verwendet zwei Punkte $(x_{n-1}, f(x_{n-1}))$ und $(x_n, f(x_n))$:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \cdot f(x_n)$$

Benötigt zwei Startwerte x_0 und x_1.

Konvergenzverhalten

Konvergenzordnung

Sei (x_n) eine gegen $\bar{x}$ konvergierende Folge. Die Konvergenzordnung $q \geq 1$ ist definiert durch:

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

wobei $c > 0$ eine Konstante ist. Für $q = 1$ muss zusätzlich $c < 1$ gelten.

Konvergenzordnungen der Verfahren

Konvergenzgeschwindigkeiten

Newton-Verfahren: Quadratische Konvergenz: $q = 2$

Vereinfachtes Newton: Lineare Konvergenz: $q = 1$

Sekantenverfahren: Superlineare Konvergenz: $q = \frac{1+\sqrt{5}}{2} \approx 1.618$

Konvergenzgeschwindigkeit Vergleich der Verfahren:

Startwert $x_0 = 1$, Funktion $f(x) = x^2 - 2$, Ziel: $\sqrt{2}$

n	Newton	Vereinfacht	Sekanten
1	1.5000000	1.5000000	1.5000000
2	1.4166667	1.4500000	1.4545455
3	1.4142157	1.4250000	1.4142857
4	1.4142136	1.4125000	1.4142136

Fehlerabschätzung

Nullstellensatz von Bolzano

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

$$f(a) \cdot f(b) < 0$$

dann existiert mindestens eine Nullstelle $\xi \in (a, b)$.

Fehlerabschätzung für Nullstellen

So schätzen Sie den Fehler einer Näherungslösung ab:

1. Sei x_n der aktuelle Näherungswert
2. Wähle Toleranz $\epsilon > 0$
3. Prüfe Vorzeichenwechsel: $f(x_n - \epsilon) \cdot f(x_n + \epsilon) < 0$
4. Falls ja: Nullstelle liegt in $(x_n - \epsilon, x_n + \epsilon)$
5. Damit gilt: $|x_n - \xi| < \epsilon$

Praktische Fehlerabschätzung Fehlerbestimmung bei $f(x) = x^2 - 2$

1. Näherungswert: $x_3 = 1.4142157$
2. Mit $\epsilon = 10^{-5}$:
3. $f(x_3 - \epsilon) = 1.4142057^2 - 2 < 0$
4. $f(x_3 + \epsilon) = 1.4142257^2 - 2 > 0$
5. Also: $|x_3 - \sqrt{2}| < 10^{-5}$

Abbruchkriterien Praktische Implementierung

In der Praxis verwendet man meist mehrere Abbruchkriterien:

- Absolute Änderung: $|x_n - x_{n-1}| < \epsilon_1$
- Funktionswert: $|f(x_n)| < \epsilon_2$
- Maximale Iterationszahl: $n < n_{max}$
- Kombination dieser Kriterien

Newton-Verfahren

```python
def newton(f, df, x0, tol=1e-6, max_iter=100):
    for n in range(max_iter):
        x1 = x0 - f(x0) / df(x0)
        if abs(x1 - x0) < tol:
            return x1
        x0 = x1
    raise ValueError("No convergence")
```

Sekantenverfahren

```python
def secant(f, x0, x1, tol=1e-6, max_iter=100):
    for n in range(max_iter):
        x2 = x1 - (x1 - x0) / (f(x1) - f(x0)) * f(x1)
        if abs(x2 - x1) < tol:
            return x2
        x0, x1 = x1, x2
    raise ValueError("No convergence")
```

Fehlerabschätzung

```python
def error_estimate(f, x, eps=1e-5):
    if f(x - eps) * f(x + eps) < 0:
        return eps
    return None
```

Numerische Lösung linearer Gleichungssysteme

Grundlagen

Lineares Gleichungssystem

Ein lineares Gleichungssystem der Form $Ax = b$ besteht aus:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \in \mathbb{R}^{n \times n}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{R}^n$$

Der Gauss-Algorithmus

Grundidee Gauss-Elimination

Transformation des Gleichungssystems $Ax = b$ in ein äquivalentes System $\tilde{A}x = \tilde{b}$, wobei $\tilde{A}$ eine obere Dreiecksmatrix ist.
Erlaubte Operationen:

- $z_j := z_j - \lambda z_i$ für $i < j$ und $\lambda \in \mathbb{R}$
- $z_i \to z_j$ (Vertauschen von Zeilen)

Gauss-Algorithmus

1. Elimination (Vorwärts)

1. Für $i = 1, \ldots, n-1$:
2. Für $j = i+1, \ldots, n$:
3. Berechne $\lambda_{ji} = a_{ji}/a_{ii}$
4. $z_j := z_j - \lambda_{ji} z_i$

2. Rückwärtseinsetzen

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}}, \quad i = n, n-1, \ldots, 1$$

Gauss-Elimination

Gegebenes System: $A = \begin{pmatrix} 2 & 1 & -1 \\ 4 & -1 & 3 \\ -2 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 1 \\ 4 \end{pmatrix}$

Eliminationsschritte:

$$\left(\begin{array}{ccc|c} 2 & 1 & -1 & 3 \\ 0 & -3 & 5 & -5 \\ 0 & 3 & -1 & 7 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|c} 2 & 1 & -1 & 3 \\ 0 & -3 & 5 & -5 \\ 0 & 0 & 4 & 2 \end{array}\right)$$

Rückwärtseinsetzen:

$$\begin{aligned} x_3 &= \tfrac{2}{4} = \tfrac{1}{2} \\ x_2 &= \tfrac{-5-5(\frac{1}{2})}{-3} = -2 \\ x_1 &= \tfrac{3-1(-2)-(-1)(\frac{1}{2})}{2} = 1 \end{aligned}$$

1. Elimination der ersten Spalte:

$$\left(\begin{array}{ccc|c} 2 & 1 & -1 & 3 \\ 0 & -3 & 5 & -5 \\ 0 & 3 & -1 & 7 \end{array}\right)$$

2. Elimination der zweiten Spalte:

$$\left(\begin{array}{ccc|c} 2 & 1 & -1 & 3 \\ 0 & -3 & 5 & -5 \\ 0 & 0 & 4 & 2 \end{array}\right)$$

3. Rückwärtseinsetzen:

$$x_3 = \frac{2}{4} = \frac{1}{2}$$
$$x_2 = \frac{-5 - 5(\frac{1}{2})}{-3} = -2$$
$$x_1 = \frac{3 - 1(-2) - (-1)(\frac{1}{2})}{2} = 1$$

Pivotisierung

Spaltenpivotisierung

Strategie zur numerischen Stabilisierung des Gauss-Algorithmus durch Auswahl des betragsmäßig größten Elements als Pivotelement.
Vor jedem Eliminationsschritt in Spalte i:

- Suche k mit $|a_{ki}| = \max\{|a_{ji}| \mid j = i, \ldots, n\}$
- Falls $a_{ki} \neq 0$: Vertausche Zeilen i und k
- Falls $a_{ki} = 0$: Matrix ist singulär

Gauss mit Pivotisierung

Erweiterter Gauss-Algorithmus mit Spaltenpivotisierung:

1. Für $i = 1, \ldots, n-1$:
2. Finde $k \geq i$ mit $|a_{ki}| = \max\{|a_{ji}| \mid j = i, \ldots, n\}$
3. Falls $a_{ki} = 0$: Stop (Matrix singulär)
4. Vertausche Zeilen i und k
5. Für $j = i+1, \ldots, n$:
6. $z_j := z_j - \frac{a_{ji}}{a_{ii}} z_i$

Pivotisierung

Gauss-Elimination mit Pivotisierung System:

$$\begin{pmatrix} 1 & 2 & -1 \\ 4 & 8 & -3 \\ 9 & 18 & -8 \end{pmatrix} x = \begin{pmatrix} 1 \\ 4 \\ 9 \end{pmatrix}$$

1. Erste Spalte: Pivot $|9| \to$ Tausche Zeilen 1 und 3
2. Nach Elimination der ersten Spalte:

$$\left(\begin{array}{ccc|c} 9 & 18 & -8 & 9 \\ 0 & 0 & 0.89 & 0 \\ 0 & 0 & 0.89 & 0 \end{array}\right)$$

3. System ist schlecht konditioniert (identische Zeilen)

Matrix-Zerlegungen

Dreieckszerlegung

Eine Matrix $A \in \mathbb{R}^{n \times n}$ kann zerlegt werden in:

Untere Dreiecksmatrix L:	**Obere Dreiecksmatrix R:**
$l_{ij} = 0$ für $j > i$	$r_{ij} = 0$ für $i > j$
Diagonale meist normiert ($l_{ii} = 1$)	Diagonalelemente $\neq 0$

LR-Zerlegung

Jede reguläre Matrix A, für die der Gauss-Algorithmus ohne Zeilenvertauschungen durchführbar ist, lässt sich zerlegen in:

$$A = LR$$

wobei L eine normierte untere und R eine obere Dreiecksmatrix ist.

Berechnung der LR-Zerlegung

So berechnen Sie die LR-Zerlegung:

1. Führen Sie Gauss-Elimination durch
2. R ist die resultierende obere Dreiecksmatrix
3. Die Eliminationsfaktoren $-\frac{a_{ji}}{a_{ii}}$ bilden L
4. Lösen Sie dann nacheinander:
   - $Ly = b$ (Vorwärtseinsetzen)
   - $Rx = y$ (Rückwärtseinsetzen)

LR-Zerlegung

Berechnung einer LR-Zerlegung Matrix:

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 4 & -1 & 0 \\ -2 & 3 & 1 \end{pmatrix}$$

Schrittweise Elimination führt zu:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 2 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 2 & 1 & 1 \\ 0 & -3 & -2 \\ 0 & 0 & -2 \end{pmatrix}$$

QR-Zerlegung

Eine orthogonale Matrix $Q \in \mathbb{R}^{n \times n}$ erfüllt: $Q^T Q = QQ^T = I_n$
Die QR-Zerlegung einer Matrix A ist:

$$A = QR$$

wobei Q orthogonal und R eine obere Dreiecksmatrix ist.

Householder-Transformation
Eine Householder-Matrix hat die Form:

$$H = I_n - 2uu^T$$

mit $u \in \mathbb{R}^n$, $\|u\| = 1$. Es gilt:
- H ist orthogonal ($H^T = H^{-1}$)
- H ist symmetrisch ($H^T = H$)
- $H^2 = I_n$

QR-Zerlegung mit Householder
So berechnen Sie die QR-Zerlegung:
1. Initialisierung: $R := A$, $Q := I_n$
2. Für $i = 1, \ldots, n-1$:
   - Bilde Vektor v_i aus i-ter Spalte von R ab Position i
   - $w_i := v_i + \text{sign}(v_{i1})\|v_i\|e_1$
   - $u_i := w_i/\|w_i\|$
   - $H_i := I_{n-i+1} - 2u_iu_i^T$
   - Erweitere H_i zu Q_i durch I_{i-1} links oben
   - $R := Q_iR$
   - $Q := QQ_i^T$

QR-Zerlegung Berechnung einer QR-Zerlegung Matrix:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Erste Householder-Transformation:
1. $v_1 = (1, 1, 0)^T$
2. $w_1 = (1, 1, 0)^T + \sqrt{2}(1, 0, 0)^T$
3. $u_1 = \frac{1}{\sqrt{6}}(1 + \sqrt{2}, 1, 0)^T$
4. $H_1 = I_3 - 2u_1u_1^T$

Nach allen Transformationen:

$$Q = \begin{pmatrix} -0.7071 & -0.5774 & -0.4082 \\ -0.7071 & 0.5774 & 0.4082 \\ 0 & -0.5774 & 0.8165 \end{pmatrix}, \quad R = \begin{pmatrix} -1.4142 & -0.7071 \\ 0 & -1.2247 \\ 0 & 0 \end{pmatrix}$$

Fehleranalyse

Matrix- und Vektornormen
Eine Vektornorm $\|\cdot\|$ erfüllt für alle $x, y \in \mathbb{R}^n, \lambda \in \mathbb{R}$:
- $\|x\| \geq 0$ und $\|x\| = 0 \Leftrightarrow x = 0$
- $\|\lambda x\| = |\lambda| \cdot \|x\|$
- $\|x + y\| \leq \|x\| + \|y\|$ (Dreiecksungleichung)

Wichtige Normen
1-Norm:
$\|x\|_1 = \sum_{i=1}^n |x_i|$
$\|A\|_1 = \max_j \sum_{i=1}^n |a_{ij}|$

2-Norm:
$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$
$\|A\|_2 = \sqrt{\rho(A^TA)}$

∞-Norm:
$\|x\|_\infty = \max_i |x_i|$
$\|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|$

Fehlerabschätzung für LGS
Sei $\|\cdot\|$ eine Norm, $A \in \mathbb{R}^{n\times n}$ regulär und $Ax = b$, $A\tilde{x} = \tilde{b}$. Dann gilt:

Absoluter Fehler:
$\|x - \tilde{x}\| \leq \|A^{-1}\| \cdot \|b - \tilde{b}\|$

Relativer Fehler:
$\frac{\|x-\tilde{x}\|}{\|x\|} \leq \text{cond}(A) \cdot \frac{\|b-\tilde{b}\|}{\|b\|}$

Mit der Konditionszahl $\text{cond}(A) = \|A\| \cdot \|A^{-1}\|$

Konditionierung
Die Konditionszahl beschreibt die numerische Stabilität eines LGS:
- $\text{cond}(A) \approx 1$: gut konditioniert
- $\text{cond}(A) \gg 1$: schlecht konditioniert
- $\text{cond}(A) \to \infty$: singulär

Iterative Verfahren

Zerlegung der Systemmatrix
Für iterative Verfahren wird A zerlegt in:

$$A = L + D + R$$

wobei:
- L: streng untere Dreiecksmatrix
- D: Diagonalmatrix
- R: streng obere Dreiecksmatrix

Jacobi-Verfahren
Gesamtschrittverfahren mit der Iteration:

$$x^{(k+1)} = -D^{-1}(L + R)x^{(k)} + D^{-1}b$$

Komponentenweise:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^{(k)}\right)$$

Gauss-Seidel-Verfahren
Einzelschrittverfahren mit der Iteration:

$$x^{(k+1)} = -(D + L)^{-1}Rx^{(k)} + (D + L)^{-1}b$$

Komponentenweise:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

Konvergenzkriterien
Ein iteratives Verfahren konvergiert, wenn:
1. Die Matrix A diagonaldominant ist:
   $|a_{ii}| > \sum_{j\neq i} |a_{ij}|$ für alle i
2. Der Spektralradius der Iterationsmatrix kleiner 1 ist:
   $\rho(B) < 1$ mit B als jeweilige Iterationsmatrix

Implementation iterativer Verfahren
So implementieren Sie iterative Verfahren:
1. Wählen Sie Startvektor $x^{(0)}$
2. Wählen Sie Abbruchkriterien:
   - Maximale Iterationszahl k_{max}
   - Toleranz ϵ für Änderung $\|x^{(k+1)} - x^{(k)}\|$
   - Toleranz für Residuum $\|Ax^{(k)} - b\|$
3. Führen Sie Iteration durch bis Kriterien erfüllt

Iterative Verfahren Vergleich Jacobi und Gauss-Seidel System:

$$\begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} x = \begin{pmatrix} 1 \\ 5 \\ 0 \end{pmatrix}$$

k	Jacobi		Gauss-Seidel	
0	$(0, 0, 0)^T$		$(0, 0, 0)^T$	
1	$(0.25, 1.25, 0)^T$	1.25	$(0.25, 1.31, 0.08)^T$	1.31
2	$(0.31, 1.31, 0.31)^T$	0.31	$(0.33, 1.33, 0.33)^T$	0.02
3	$(0.33, 1.33, 0.33)^T$	0.02	$(0.33, 1.33, 0.33)^T$	0.00

Eigenwerte und Eigenvektoren

Komplexe Zahlen

Komplexe Zahlen
Die Menge der komplexen Zahlen $\mathbb{C}$ erweitert die reellen Zahlen $\mathbb{R}$ durch Einführung der imaginären Einheit i mit der Eigenschaft:

$$i^2 = -1$$

Eine komplexe Zahl z ist ein geordnetes Paar (x, y) mit $x, y \in \mathbb{R}$:

$$z = x + iy$$

Die Menge aller komplexen Zahlen ist definiert als:

$$\mathbb{C} = \{z \mid z = x + iy \text{ mit } x, y \in \mathbb{R}\}$$

Bestandteile komplexer Zahlen
Realteil:
$\text{Re}(z) = x$

Imaginärteil:
$\text{Im}(z) = y$

Konjugiert komplexe Zahl:
$z^* = x - iy$

Betrag:
$|z| = \sqrt{x^2 + y^2} = \sqrt{z \cdot z^*}$

Darstellungsformen
Eine komplexe Zahl kann dargestellt werden als:
- Normalform: $z = x + iy$
- Trigonometrische Form: $z = r(\cos\varphi + i\sin\varphi)$
- Exponentialform: $z = re^{i\varphi}$

Mit:

$$x = r\cos\varphi, \quad y = r\sin\varphi, \quad r = \sqrt{x^2 + y^2}$$

$$\varphi = \arcsin\left(\frac{y}{r}\right) = \arccos\left(\frac{x}{r}\right)$$

$$e^{i\varphi} = \cos\varphi + i\sin\varphi \text{ (Euler-Formel)}$$

Darstellungsformen Umrechnung verschiedener Darstellungen Gegeben: $z = 3 - 11i$

1. Berechnung von r: $r = \sqrt{3^2 + 11^2} = \sqrt{130}$
2. Berechnung von φ: $\varphi = \arcsin\left(\frac{11}{\sqrt{130}}\right) = 1.3$
3. Trigonometrische Form: $z = \sqrt{130}(\cos(1.3) + i\sin(1.3))$
4. Exponentialform: $z = \sqrt{130}e^{i \cdot 1.3}$

Rechenoperationen mit komplexen Zahlen

Für $z_1 = x_1 + iy_1$ und $z_2 = x_2 + iy_2$ gilt:

Addition: $z_1 + z_2 = (x_1 + x_2) + i(y_1 + y_2)$

Subtraktion: $z_1 - z_2 = (x_1 - x_2) + i(y_1 - y_2)$

Multiplikation:

$$\begin{aligned} z_1 \cdot z_2 &= (x_1x_2 - y_1y_2) + i(x_1y_2 + x_2y_1) \\ &= r_1r_2e^{i(\varphi_1+\varphi_2)} \text{ (in Exponentialform)} \end{aligned}$$

Division:

$$\begin{aligned} \frac{z_1}{z_2} &= \frac{z_1 \cdot z_2^*}{z_2 \cdot z_2^*} = \frac{(x_1x_2 + y_1y_2) + i(y_1x_2 - x_1y_2)}{x_2^2 + y_2^2} \\ &= \frac{r_1}{r_2}e^{i(\varphi_1-\varphi_2)} \text{ (in Exponentialform)} \end{aligned}$$

Potenzen und Wurzeln

Für eine komplexe Zahl in Exponentialform $z = re^{i\varphi}$ gilt:

- n-te Potenz: $z^n = r^n e^{in\varphi} = r^n(\cos(n\varphi) + i\sin(n\varphi))$
- n-te Wurzel: $z_k = \sqrt[n]{r}e^{i\frac{\varphi+2\pi k}{n}}$, $k = 0, 1, \ldots, n-1$

Fundamentalsatz der Algebra

Eine algebraische Gleichung n-ten Grades mit komplexen Koeffizienten:

$$a_nz^n + a_{n-1}z^{n-1} + \cdots + a_1z + a_0 = 0$$

besitzt in $\mathbb{C}$ genau n Lösungen (mit Vielfachheiten gezählt).

Eigenwerte und Eigenvektoren

Eigenwerte und Eigenvektoren

Für eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt $\lambda \in \mathbb{C}$ Eigenwert von A, wenn es einen Vektor $x \in \mathbb{C}^n \backslash \{0\}$ gibt mit:

$$Ax = \lambda x$$

Der Vektor x heißt dann Eigenvektor zum Eigenwert λ.

Bestimmung von Eigenwerten

Ein Skalar λ ist genau dann Eigenwert von A, wenn gilt:

$$\det(A - \lambda I_n) = 0$$

Diese Gleichung heißt charakteristische Gleichung. Das zugehörige Polynom

$$p(\lambda) = \det(A - \lambda I_n)$$

ist das charakteristische Polynom von A.

Eigenschaften von Eigenwerten

Für eine Matrix $A \in \mathbb{R}^{n\times n}$ gilt:

- $\det(A) = \prod_{i=1}^{n} \lambda_i$ (Produkt der Eigenwerte)
- $\mathrm{tr}(A) = \sum_{i=1}^{n} \lambda_i$ (Summe der Eigenwerte)
- Bei einer Dreiecksmatrix sind die Diagonalelemente die Eigenwerte
- Ist λ Eigenwert von A, so ist $\frac{1}{\lambda}$ Eigenwert von A^{-1}

Vielfachheiten

Für einen Eigenwert λ unterscheidet man:

- Algebraische Vielfachheit: Vielfachheit als Nullstelle des charakteristischen Polynoms
- Geometrische Vielfachheit: Dimension des Eigenraums $= n - \mathrm{rg}(A - \lambda I_n)$

Die geometrische Vielfachheit ist stets kleiner oder gleich der algebraischen Vielfachheit.

Bestimmung von Eigenwerten und Eigenvektoren

1. Charakteristisches Polynom aufstellen: $p(\lambda) = \det(A - \lambda I_n)$
2. Eigenwerte durch Lösen von $p(\lambda) = 0$ bestimmen
3. Für jeden Eigenwert λ_i:
   - System $(A - \lambda_i I_n)x = 0$ aufstellen
   - Lösungsraum = Eigenraum bestimmen
   - Basis des Eigenraums = linear unabhängige Eigenvektoren

Eigenwertberechnung Matrix mit drei Eigenwerten Gegeben sei: $A = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 0 & 1 & 2 \end{pmatrix}$

1. Da A eine Dreiecksmatrix ist, sind die Diagonalelemente die Eigenwerte: $\lambda_1 = 1, \lambda_2 = 3, \lambda_3 = 2$
2. $\det(A) = \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 6$
3. $\mathrm{tr}(A) = \lambda_1 + \lambda_2 + \lambda_3 = 6$
4. Spektrum: $\sigma(A) = \{1, 2, 3\}$

Numerische Berechnung von Eigenwerten

Ähnliche Matrizen

Zwei Matrizen $A, B \in \mathbb{R}^{n\times n}$ heißen ähnlich, wenn es eine reguläre Matrix T gibt mit:

$$B = T^{-1}AT$$

Eine Matrix A heißt diagonalisierbar, wenn sie ähnlich zu einer Diagonalmatrix D ist:

$$D = T^{-1}AT$$

Eigenschaften ähnlicher Matrizen

Für ähnliche Matrizen A und $B = T^{-1}AT$ gilt:

1. A und B haben dieselben Eigenwerte mit gleichen algebraischen Vielfachheiten
2. Ist x Eigenvektor von B zum Eigenwert λ, so ist Tx Eigenvektor von A zum Eigenwert λ
3. Bei Diagonalisierbarkeit:
   - Die Diagonalelemente von D sind die Eigenwerte von A
   - Die Spalten von T sind die Eigenvektoren von A

Spektralradius

Der Spektralradius einer Matrix A ist definiert als:

$$\rho(A) = \max\{|\lambda| \mid \lambda \text{ ist Eigenwert von } A\}$$

Er gibt den Betrag des betragsmäßig größten Eigenwerts an.

Iterative Verfahren

Von-Mises-Iteration (Vektoriteration)

Für eine diagonalisierbare Matrix A mit Eigenwerten $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$ konvergiert die Folge:

$$\begin{aligned} v^{(k+1)} &= \frac{Av^{(k)}}{\|Av^{(k)}\|_2} \\ \lambda^{(k+1)} &= \frac{(v^{(k)})^T Av^{(k)}}{(v^{(k)})^T v^{(k)}} \end{aligned}$$

gegen einen Eigenvektor v zum betragsmäßig größten Eigenwert λ_1.

Von-Mises-Iteration durchführen

1. Wähle Startvektor $v^{(0)}$ mit $\|v^{(0)}\|_2 = 1$
2. Für $k = 0, 1, 2, \ldots$:
   - Berechne $w^{(k)} = Av^{(k)}$
   - Normiere: $v^{(k+1)} = \frac{w^{(k)}}{\|w^{(k)}\|_2}$
   - Berechne Rayleigh-Quotienten $\lambda^{(k+1)}$
   - Prüfe Konvergenz

Von-Mises-Iteration Berechnung des größten Eigenwerts

```python
import numpy as np

def power_iteration(A, tol=1e-10, max_iter=100):
    n = A.shape[0]
    v = np.random.rand(n)
    v = v / np.linalg.norm(v)

    for i in range(max_iter):
        w = A @ v
        v_new = w / np.linalg.norm(w)

        # Rayleigh-Quotient
        lambda_k = v_new.T @ A @ v_new

        if np.linalg.norm(v_new - v) < tol:
            return lambda_k, v_new

        v = v_new

    return lambda_k, v_new
```

QR-Verfahren

Das QR-Verfahren transformiert die Matrix A iterativ in eine obere Dreiecksmatrix, deren Diagonalelemente die Eigenwerte sind:

1. Initialisierung: $A_0 := A$, $P_0 := I_n$
2. Für $i = 0, 1, 2, \ldots$:
   - QR-Zerlegung: $A_i = Q_iR_i$
   - Neue Matrix: $A_{i+1} = R_iQ_i$
   - Update: $P_{i+1} = P_iQ_i$

QR-Verfahren anwenden

1. Matrix $A_0 = A$ vorbereiten
2. In jedem Schritt i:
   - QR-Zerlegung mit Householder oder Givens
   - Neue Matrix durch Multiplikation $R_i Q_i$
   - Konvergenz prüfen: Subdiagonalelemente ≈ 0?
3. Eigenwerte: Diagonalelemente der Endmatrix
4. Eigenvektoren: Spalten von $P = P_1 P_2 \cdots P_k$

QR-Verfahren Implementation in Python

```python
def qr_algorithm(A, tol=1e-10, max_iter=100):
    n = A.shape[0]
    Q_prod = np.eye(n)
    A_k = A.copy()

    for k in range(max_iter):
        # QR decomposition
        Q, R = np.linalg.qr(A_k)
        # New iteration
        A_k = R @ Q
        # Update transformation matrix
        Q_prod = Q_prod @ Q

        # Check convergence
        if np.abs(np.tril(A_k, -1)).max() < tol:
            break

    return np.diag(A_k), Q_prod
```

Numerische Stabilität

- QR-Verfahren ist numerisch stabiler als Vektoriteration
- Findet alle Eigenwerte, nicht nur den größten
- Benötigt mehr Rechenaufwand
- Konvergiert linear für reelle, quadratisch für komplexe Eigenwerte

Python

Numerische Bibliotheken Verwendung spezialisierter Bibliotheken

Für kritische numerische Berechnungen:

- NumPy: Optimierte Array-Operationen
- SciPy: Wissenschaftliches Rechnen
- Mpmath: Beliebige Präzision
- Decimal: Dezimalarithmetik

Bibliotheksverwendung Beispiel: Präzise Berechnung mit Decimal

```python
from decimal import Decimal, getcontext

# Set precision
getcontext().prec = 40

# Precise calculation
x = Decimal('1.0') / Decimal('7.0')
print(x)  # 0.1428571428571428571428571428571428571428
```

Examples