

Rechnerarithmetik

Zahldarstellung

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

$$M = \{x \in \mathbb{R} \mid x = \pm 0.m_1m_2m_3 \dots m_n \cdot B^{\pm e_1e_2 \dots e_l}\} \cup \{0\}$$

- $m_1 \neq 0$  (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:

$$\underbrace{0.3211}_{n=4} \cdot \underbrace{4^{12}}_{l=2} \quad \text{Exponent: } \hat{e} = 1 \cdot 4^1 + 2 \cdot 4^0 = 6$$

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

**IEEE-754 Standard** definiert zwei wichtige Gleitpunktformate:

<b>Single Precision</b> (32 Bit)	<b>Double Precision</b> (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  $x$ :

<b>Absoluter Fehler:</b>	<b>Relativer Fehler:</b>
$ \tilde{x} - x $	$\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:** **Dezimal:**

$$\text{eps} := \frac{B}{2} \cdot B^{-n} \qquad \text{eps}_{10} := 5 \cdot 10^{-n}$$

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:

<b>Absoluter Fehler:</b>	<b>Relativer Fehler:</b>
$ rd(x) - x  \leq \frac{B}{2} \cdot B^{e-n-1}$	$\left  \frac{rd(x) - x}{x} \right  \leq \text{eps}$

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**

- Berechnen Sie  $f'(x)$
- Bestimmen Sie die Konditionszahl  $K$
- Schätzen Sie den absoluten Fehler ab
- Schätzen Sie den relativen Fehler ab
- Beurteilen Sie die Konditionierung anhand von  $K$

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

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

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

- $f'(x) = \cos(x)$
- $K = \frac{|x \cos(x)|}{|\sin(x)|}$
- Für  $x \rightarrow 0$ :  $K \rightarrow 1$  (gut konditioniert)
- Für  $x \rightarrow \pi$ :  $K \rightarrow \infty$  (schlecht konditioniert)
- 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**

- Identifizieren Sie Subtraktionen ähnlich großer Zahlen
- Suchen Sie nach algebraischen Umformungen
- Prüfen Sie alternative Berechnungswege
- 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)

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

Exponentielles Wachstum der Operationen  $\rightarrow$  Fehlerfortpflanzung

**Stabilitätsanalyse** Schritte zur Analyse der numerischen Stabilität:

- Bestimmen Sie kritische Operationen
- Schätzen Sie Rundungsfehler pro Operation ab
- Analysieren Sie die Fehlerfortpflanzung
- Berechnen Sie die worst-case Fehlerschranke
- Vergleichen Sie alternative Implementierungen

Praktische Implementierungen

**Implementierungsgenauigkeit eines Algorithmus**

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

**Robuste Implementierung von Algorithmen**

- Verwenden Sie stabile Grundoperationen
- Vermeiden Sie Differenzen ähnlich großer Zahlen
- Prüfen Sie auf Über- und Unterlauf
- Implementieren Sie Fehlerkontrollen
- Dokumentieren Sie numerische Einschränkungen

**Robuste Implementation** Beispiel: Quadratische Gleichung

```
1 def quadratic_stable(a, b, c):
2     # ax^2 + bx + c = 0
3     if a == 0:
4         return [-c/b] if b != 0 else []
5
6     # Calculate discriminant
7     disc = b*b - 4*a*c
8     if disc < 0:
9         return []
10
11    # Choose numerically stable formula
12    if b >= 0:
13        q = -0.5*(b + sqrt(disc))
14    else:
15        q = -0.5*(b - sqrt(disc))
16    x1 = q/a
17    x2 = c/(q)
18
19    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  $F : [a, b] \rightarrow \mathbb{R}$  mit  $x_0 \in [a, b]$ . Die rekursive Folge

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

heisst Fixpunktiteration von  $F$  zum Startwert  $x_0$ .

### Konvergenzverhalten

Sei  $F : [a, b] \rightarrow \mathbb{R}$  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:**

**Abstossender Fixpunkt:**

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

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

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

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

### Banachscher Fixpunktsatz

Sei  $F : [a, b] \rightarrow [a, b]$  und es existiere eine Konstante  $\alpha$  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  $F : [a, b] \rightarrow [a, b]$  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  $\text{tol}$ :

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

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

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

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

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

### Sekantenverfahren

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

### Fehlerabschätzung

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

## LGS und Matrizen

### Matrizen

#### Matrix, Element, Zeilen, Spalten und Typ

Eine *Matrix* ist (simpler gesagt) ein Vektor mit mehreren Spalten und wird mit Grossbuchstaben bezeichnet. Ein *Element*  $a_{ij}$  ist ein Wert aus dieser Matrix, auf den über die Zeile und Spalte zugegriffen wird (**Zeile** zuerst, **Spalte** später). Der einer Matrix ergibt sich aus der Anzahl ihren Zeilen und Spalten. Matrizen mit  $m$ -Zeilen und  $n$ -Spalten werden  $m \times n$ -Matrizen genannt.

**Matrix** Tabelle mit  $m$  Zeilen und  $n$  Spalten:  $m \times n$ -Matrix  $A$   
 $a_{ij}$ : Element in der  $i$ -ten Zeile und  $j$ -ten Spalte

**Nullmatrix** Eine Matrix, deren Elemente alle gleich 0 sind, heisst *Nullmatrix* und wird mit 0 bezeichnet.

**Spaltenmatrix** Besteht eine Matrix nur aus einer Spalte, so heisst diese *Spaltenmatrix*. Können als Vektoren aufgefasst werden und können mit einem kleinen Buchstaben sowie einem Pfeil darüber notiert werden ( $\vec{a}$ ).

#### Addition und Subtraktion

- $A + B = C$
- $c_{ij} = a_{ij} + b_{ij}$

#### Skalarmultiplikation

- $k \cdot A = B$
- $b_{ij} = k \cdot a_{ij}$

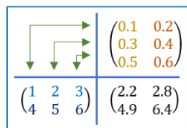
#### Rechenregeln für die Addition und skalare Multiplikation von Matrizen

- Kommutativ-Gesetz:  $A + B = B + A$
- Assoziativ-Gesetz:  $A + (B + C) = (A + B) + C$
- Distributiv-Gesetz:  
 $\lambda \cdot (A + B) = \lambda \cdot A + \lambda \cdot B$  sowie  $(\lambda + \mu) \cdot A = \lambda \cdot A + \mu \cdot A$

#### Matrixmultiplikation $A^{m \times n}, B^{n \times k}$

Bedingung:  $A$   $n$  Spalten,  $B$   $n$  Zeilen.  
Resultat:  $C$  hat  $m$  Zeilen und  $k$  Spalten.

- $A \cdot B = C$
- $c_{ij} = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + \dots + a_{in} \cdot b_{nj}$
- $A \cdot B \neq B \cdot A$



#### Rechenregeln für die Multiplikation von Matrizen

- Assoziativ-Gesetz:  $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- Distributiv-Gesetz:  
 $A \cdot (B + C) = A \cdot B + A \cdot C$  und  $(A + B) \cdot C = A \cdot C + B \cdot C$
- Skalar-Koeffizient:  $(\lambda \cdot A) \cdot B = \lambda \cdot (A \cdot B) = A \cdot (\lambda \cdot B)$

#### Transponierte Matrix $A^{m \times n} \rightarrow (A^T)^{n \times m}$

- $A^T$ : Spalten und Zeilen vertauscht
- $(A^T)_{ij} = A_{ji}$

$$(A \cdot B)^T = B^T \cdot A^T$$

#### Spezielle Matrizen

- Symmetrische Matrix:**  $A^T = A$
- Einheitsmatrix/Identitätsmatrix:**  $E$  bzw.  $I$   
mit  $e_{ij} = 1$  für  $i = j$  und  $e_{ij} = 0$  für  $i \neq j$
- Diagonalmatrix:**  $a_{ij} = 0$  für  $i \neq j$
- Dreiecksmatrix:**  $a_{ij} = 0$  für  $i > j$  (obere Dreiecksmatrix) oder  $i < j$  (untere Dreiecksmatrix)

## Lineare Gleichungssysteme (LGS)

**Lineares Gleichungssystem (LGS)** Ein *lineares Gleichungssystem* ist eine Sammlung von Gleichungen, die linear in den Unbekannten sind. Ein LGS kann in Matrixform  $A \cdot \vec{x} = \vec{b}$  dargestellt werden.

$A$ : Koeffizientenmatrix

$\vec{x}$ : Vektor der Unbekannten

$\vec{b}$ : Vektor der Konstanten

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

**Rang einer Matrix**  $rg(A) = \text{Anzahl Zeilen} - \text{Anzahl Nullzeilen}$   
 $\Rightarrow$  Anzahl linear unabhängiger Zeilen- oder Spaltenvektoren

#### Zeilenstufenform (Gauss)

- Alle Nullen stehen unterhalb der Diagonalen, Nullzeilen zuunterst
- Die erste Zahl  $\neq 0$  in jeder Zeile ist eine führende Eins
- Führende Einsen, die weiter unten stehen  $\rightarrow$  stehen weiter rechts

#### Reduzierte Zeilenstufenform: (Gauss-Jordan)

Alle Zahlen links und rechts der führenden Einsen sind Nullen.

#### Gauss-Jordan-Verfahren

- bestimme linkeste Spalte mit Elementen  $\neq 0$  (Pivot-Spalte)
  - oberste Zahl in Pivot-Spalte = 0  
 $\rightarrow$  vertausche Zeilen so dass  $a_{11} \neq 0$
  - teile erste Zeile durch  $a_{11} \rightarrow$  so erhalten wir führende Eins
  - Nullen unterhalb führender Eins erzeugen (Zeilenoperationen)
- nächste Schritte: ohne bereits bearbeitete Zeilen Schritte 1-4 wiederholen, bis Matrix Zeilenstufenform hat

**Zeilenoperationen** erlaubt bei LGS (z.B. Gauss-Verfahren)

- Vertauschen von Zeilen
- Multiplikation einer Zeile mit einem Skalar
- Addition eines Vielfachen einer Zeile zu einer anderen

#### Lösbarkeit von linearen Gleichungssystemen

- Lösbar:  $rg(A) = rg(A|b)$  • unendlich viele Lösungen:
- genau eine Lösung:  $rg(A) = n$  •  $rg(A) < n$

**Parameterdarstellung** bei unendlich vielen Lösungen

Führende Unbekannte: Spalte mit führender Eins

Freie Unbekannte: Spalten ohne führende Eins

Auflösung nach der führenden Unbekannten:

- $1x_1 - 2x_2 + 0x_3 + 3x_4 = 5$   $x_2 = \lambda \rightarrow x_1 = 5 + 2 \cdot \lambda - 3 \cdot \mu$
- $0x_1 + 0x_2 + 1x_3 + 1x_4 = 3$   $x_4 = \mu \rightarrow x_3 = 3 - \mu$

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 5+2\lambda-3\mu \\ \lambda \\ 3-\mu \\ \mu \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \\ 3 \\ 0 \end{pmatrix} + \lambda \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \mu \begin{pmatrix} -3 \\ 0 \\ -1 \\ 1 \end{pmatrix}$$

**Homogenes LGS**  $\vec{b} = \vec{0} \rightarrow A \cdot \vec{x} = \vec{0} \rightarrow rg(A) = rg(A | \vec{b})$

nur zwei Möglichkeiten:

- eine Lösung  $x_1 = x_2 = \dots = x_n = 0$ , die sog. *triviale Lösung*.
- unendlich viele Lösungen

#### Koeffizientenmatrix, Determinante, Lösbarkeit des LGS

Für  $n \times n$ -Matrix  $A$  sind folgende Aussagen äquivalent:

- $\det(A) \neq 0$
- Spalten von  $A$  sind linear unabhängig.
- $rg(A) = n$
- Zeilen von  $A$  sind linear unabhängig.
- $A$  ist invertierbar
- LGS  $A \cdot \vec{x} = \vec{0}$  hat eindeutige Lösung  $x = A^{-1} \cdot \vec{0} = \vec{0}$

## Quadratische Matrizen

**Umformen** bestimme die Matrix  $X$ :  $A \cdot X + B = 2 \cdot X$

$$\begin{aligned} \Rightarrow A \cdot X &= 2 \cdot X - B \Rightarrow A \cdot X - 2 \cdot X = -B \Rightarrow (A - 2 \cdot E) \cdot X = -B \\ \Rightarrow (A - 2 \cdot E) \cdot (A - 2 \cdot E)^{-1} \cdot X &= (A - 2 \cdot E)^{-1} \cdot -B \\ \Rightarrow X &= (A - 2 \cdot E)^{-1} \cdot -B \end{aligned}$$

### Inverse

**Inverse einer quadratischen Matrix  $A$**   $A^{-1}$

$A^{-1}$  existiert, wenn  $rg(A) = n$ .  $A^{-1}$  ist eindeutig bestimmt.

Eine Matrix heisst *invertierbar* / *regulär*, wenn sie eine Inverse hat. Andernfalls heisst sie *singulär*

#### Eigenschaften invertierbarer Matrizen

- $A \cdot A^{-1} = A^{-1} \cdot A = E$
- $(A^{-1})^{-1} = A$
- $(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$  Die Reihenfolge ist relevant!  
 $A$  und  $B$  invertierbar  $\Rightarrow AB$  invertierbar
- $(A^T)^{-1} = (A^{-1})^T$   $A$  invertierbar  $\Rightarrow A^T$  invertierbar

**Inverse einer  $2 \times 2$ -Matrix**  $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$  mit  $\det(A) = ad - bc$

$$A^{-1} = \frac{1}{\det(A)} \cdot \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

NUR Invertierbar falls  $ad - bc \neq 0$

**Inverse berechnen** einer quadratischen Matrix  $A^{n \times n}$

$$A \cdot A^{-1} = E \rightarrow (A|E) \rightsquigarrow \text{Zeilenoperationen} \rightsquigarrow (E|A^{-1})$$

$$\underbrace{\begin{pmatrix} 4 & -1 & 0 \\ 0 & 2 & 1 \\ 3 & -5 & -2 \end{pmatrix}}_A \cdot \underbrace{\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix}}_{A^{-1}} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_E$$
$$\rightarrow \left( \begin{array}{ccc|ccc} 4 & -1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 3 & -5 & -2 & 0 & 0 & 1 \end{array} \right)$$

Zeilenstufenform (linke Seite)

$$\rightsquigarrow \left( \begin{array}{ccc|ccc} 1 & -1/4 & 0 & 1/4 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1 & -6 & 17 & 8 \end{array} \right)$$

Reduzierte Zeilenstufenform (linke Seite)

$$\rightsquigarrow \left( \begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & -2 & -1 \\ 0 & 1 & 0 & 3 & -8 & -4 \\ 0 & 0 & 1 & -6 & 17 & 8 \end{array} \right) \Rightarrow A^{-1} = \begin{pmatrix} 1 & -2 & -1 \\ 3 & -8 & -4 \\ -6 & 17 & 8 \end{pmatrix}$$

**LGS mit Inverse lösen**  $A \cdot \vec{x} = \vec{b}$

$$A^{-1} \cdot A \cdot \vec{x} = A^{-1} \cdot \vec{b} \rightarrow \vec{x} = A^{-1} \cdot \vec{b}$$

Beispiel:

$$\underbrace{\begin{pmatrix} 1/2 & 0 \\ 0 & 1/3 \end{pmatrix}}_{A^{-1}} \cdot \underbrace{\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}}_A \cdot \underbrace{\begin{pmatrix} x \\ y \end{pmatrix}}_{\vec{x}} = \underbrace{\begin{pmatrix} 1/2 & 0 \\ 0 & 1/3 \end{pmatrix}}_{A^{-1}} \cdot \underbrace{\begin{pmatrix} 4 \\ 5 \end{pmatrix}}_{\vec{b}}$$

## Numerische Lösung linearer Gleichungssysteme

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

#### Gauss-Algorithmus mit Pivotisierung

##### 1. Elimination (Vorwärts)

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

##### 2. Rückwärtseinsetzen

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

#### Gauss mit Pivotisierung

Gegebenes System:  $A = \begin{pmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{pmatrix}, \quad b = \begin{pmatrix} 4 \\ 2 \\ 36 \end{pmatrix}$

#### Eliminationsschritte:

$$\left( \begin{array}{ccc|c} 2 & 4 & -2 & 2 \\ 0 & 3 & 15 & 36 \\ 0 & 1 & 1 & 4 \end{array} \right) \Rightarrow \left( \begin{array}{ccc|c} 2 & 4 & -2 & 2 \\ 0 & 3 & 15 & 36 \\ 0 & 0 & -2 & -8 \end{array} \right)$$

**Rückwärtseinsetzen:**

$$\begin{aligned} x_3 &= \frac{-8}{-2} = 4 \\ x_2 &= \frac{36 - 15(4)}{3} = 1 \\ x_1 &= \frac{2 - 4(4) + 2}{2} = -6 \end{aligned}$$

#### Permutationsmatrizen

Eine Permutationsmatrix  $P$  ist eine Matrix, die aus der Einheitsmatrix durch Zeilenvertauschungen entsteht. Für die Vertauschung der  $i$ -ten und  $j$ -ten Zeile hat  $P_k$  die Form:

- $p_{ii} = p_{jj} = 0$
- $p_{ij} = p_{ji} = 1$
- Alle anderen Elemente wie in  $I_n$

Wichtige Eigenschaften:

- $P^{-1} = P^T = P$
- Mehrere Vertauschungen:  $P = P_l \cdot \dots \cdot P_1$

**Zeilenvertauschung** für Matrix  $A$  mit Permutationsmatrix  $P_1$ :

$$\underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}}_A \cdot \underbrace{\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}}_{P_1} = \begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$$

$\Rightarrow A \cdot P_1$  bewirkt die Vertauschung von Zeile 1 und 3

## Matrix-Zerlegungen

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

**Untere Dreiecksmatrix  $L$ :**

$l_{ij} = 0$  für  $j > i$

Diagonale normiert ( $l_{ii} = 1$ )

**Obere Dreiecksmatrix  $R$ :**

$r_{ij} = 0$  für  $i > j$

Diagonalelemente  $\neq 0$

### LR-Zerlegung

#### 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**  $A = \begin{pmatrix} -1 & 1 & 1 \\ 1 & -3 & -2 \\ 5 & 1 & 4 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 5 \\ 3 \end{pmatrix}$

##### Schritt 1: Erste Spalte

Max. Element in 1. Spalte:  $|a_{31}| = 5$ , also Z1 und Z3 tauschen:

$$P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad A^{(1)} = \begin{pmatrix} 5 & 1 & 4 \\ 1 & -3 & -2 \\ -1 & 1 & 1 \end{pmatrix}$$

Berechne Eliminationsfaktoren:  $l_{21} = \frac{1}{5}, \quad l_{31} = -\frac{1}{5}$

Nach Elimination:  $A^{(2)} = \begin{pmatrix} 5 & 1 & 4 \\ 0 & -3.2 & -2.8 \\ 0 & 1.2 & 1.8 \end{pmatrix}$

##### Schritt 2: Zweite Spalte

Max. Element in 2. Spalte unter Diagonale:  $|-3.2| > |1.2|$ , keine Vertauschung nötig.

Berechne Eliminationsfaktor:  $l_{32} = -\frac{1.2}{-3.2} = \frac{3}{8}$

Nach Elimination:  $R = \begin{pmatrix} 5 & 1 & 4 \\ 0 & -3.2 & -2.8 \\ 0 & 0 & 2.85 \end{pmatrix}$

##### Endergebnis

Die LR-Zerlegung mit  $PA = LR$  ist:

$$P = P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

,

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{5} & 1 & 0 \\ -\frac{1}{5} & \frac{3}{8} & 1 \end{pmatrix}, R = \begin{pmatrix} 5 & 1 & 4 \\ 0 & -3.2 & -2.8 \\ 0 & 0 & 2.85 \end{pmatrix}$$

### Lösung des Systems

1.  $Pb = \begin{pmatrix} 3 \\ 5 \\ 0 \end{pmatrix}$

2. Löse  $Ly = Pb$  durch Vorwärtseinsetzen:  $y = \begin{pmatrix} 3 \\ 4.4 \\ 2.85 \end{pmatrix}$

3. Löse  $Rx = y$  durch Rückwärtseinsetzen:  $x = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$

### Probe

$$Ax = \begin{pmatrix} -1 & 1 & 1 \\ 1 & -3 & -2 \\ 5 & 1 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 5 \\ 3 \end{pmatrix} = b$$

#### Zeilenvertauschungen verfolgen

1. Initialisiere  $P = I_n$
2. Für jede Vertauschung von Zeile  $i$  und  $j$ :
  - Erstelle  $P_k$  durch Vertauschen von Zeilen  $i, j$  in  $I_n$
  - Aktualisiere  $P = P_k \cdot P$
  - Wende Vertauschung auf Matrix an:  $A := P_k A$
3. Bei der LR-Zerlegung mit Pivotisierung:
  - $PA = LR$
  - Löse  $Ly = Pb$  und  $Rx = y$

#### LR-Zerlegung mit Pivotisierung

```
1 def lr_decomposition_with_pivoting(A):
2     n = len(A)
3     P = np.eye(n) # Permutationsmatrix
4     L = np.eye(n) # Untere Dreiecksmatrix
5     R = A.copy()  # Wird zur oberen Dreiecksmatrix
6
7     for k in range(n-1):
8         # Finde Pivotelement
9         pivot = np.argmax(abs(R[k:,k])) + k
10
11         if pivot != k:
12             # Erzeuge Permutationsmatrix
13             P_k = np.eye(n)
14             P_k[[k,pivot]] = P_k[[pivot,k]]
15
16             # Aktualisiere Matrizen
17             P = P_k @ P
18             R[[k,pivot]] = R[[pivot,k]]
19             if k > 0:
20                 L[[k,pivot], :k] = L[[pivot,k], :k]
21
22         # Elimination durchfuehren
23         for i in range(k+1, n):
24             factor = R[i,k] / R[k,k]
25             L[i,k] = factor
26             R[i,k:] -= factor * R[k,k:]
27
28     return P, L, R
```

#### Vorteile der Permutationsmatrix

- Exakte Nachverfolgung aller Zeilenvertauschungen
- Einfache Rückführung auf ursprüngliche Reihenfolge durch  $P^{-1}$
- Kompakte Darstellung mehrerer Vertauschungen
- Numerisch stabile Implementierung der Pivotisierung

QR-Zerlegung

Eine orthogonale Matrix  $Q \in \mathbb{R}^{n \times n}$  erfüllt:  $Q^T Q = Q Q^T = I_n$   
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

- Initialisierung:  $R := A$ ,  $Q := I_n$
- Für  $i = 1, \dots, n - 1$ :
  - Bilde Vektor  $v_i$  aus  $i$ -ter Spalte von  $R$  ab Position  $i$
  - $w_i := v_i + \text{sign}(v_{i1})\|v_i\|e_1$
  - $u_i := w_i / \|w_i\|$
  - $H_i := I_{n-i+1} - 2u_i u_i^T$
  - Erweitere  $H_i$  zu  $Q_i$  durch  $I_{i-1}$  links oben
  - $R := Q_i R$  und  $Q := Q Q_i^T$

QR-Zerlegung mit Householder  $A = \begin{pmatrix} 2 & 5 & -1 \\ -1 & -4 & 2 \\ 0 & 2 & 1 \end{pmatrix}$

Schritt 1: Erste Spalte

Erste Spalte  $a_1$  und Einheitsvektor  $e_1$ :  $a_1 = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix}$ ,  $e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

Householder-Vektor für erste Spalte:

- Berechne Norm:  $|a_1| = \sqrt{2^2 + (-1)^2 + 0^2} = \sqrt{5}$
- Bestimme Vorzeichen:  $\text{sign}(a_{11}) = \text{sign}(2) = 1$ 
  - Wähle positives Vorzeichen, da erstes Element positiv
  - Dies maximiert die erste Komponente von  $v_1$
  - Verhindert Auslöschung bei der Subtraktion

3.  $v_1 = a_1 + \text{sign}(a_{11})|a_1|e_1 = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} + \sqrt{5}\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2+\sqrt{5} \\ -1 \\ 0 \end{pmatrix}$

4. Normiere  $v_1$ :  $|v_1| = \sqrt{(2+\sqrt{5})^2 + 1} \Rightarrow u_1 = \frac{v_1}{|v_1|} = \begin{pmatrix} 0.91 \\ -0.41 \\ 0 \end{pmatrix}$

Householder-Matrix berechnen:  $H_1 = I - 2u_1 u_1^T = \begin{pmatrix} -0.67 & -0.75 & 0 \\ -0.75 & 0.67 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

A nach erster Transformation:  $A^{(1)} = H_1 A = \begin{pmatrix} -\sqrt{5} & -6.71 & 0.45 \\ 0 & -0.89 & 1.79 \\ 0 & 2.00 & 1.00 \end{pmatrix}$

Schritt 2: Zweite Spalte

Untermatrix für zweite Transformation:  $A_2 = \begin{pmatrix} -0.89 & 1.79 \\ 2.00 & 1.00 \end{pmatrix}$

Householder-Vektor für zweite Spalte:

- $|a_2| = \sqrt{(-0.89)^2 + 2^2} = 2.19$
- $\text{sign}(a_{22}) = \text{sign}(-0.89) = -1$  (da erstes Element negativ)
- $v_2 = \begin{pmatrix} -0.89 \\ 2.00 \end{pmatrix} - 2.19\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -3.09 \\ 2.00 \end{pmatrix}$
- $u_2 = \frac{v_2}{|v_2|} = \begin{pmatrix} -0.84 \\ 0.54 \end{pmatrix}$

Erweiterte Householder-Matrix:  $Q_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.41 & -0.91 \\ 0 & -0.91 & 0.41 \end{pmatrix}$

nach 2. Transformation:  $R = Q_2 A^{(1)} = \begin{pmatrix} -\sqrt{5} & -6.71 & 0.45 \\ 0 & -2.19 & 1.34 \\ 0 & 0 & -1.79 \end{pmatrix}$

Endergebnis

Die QR-Zerlegung  $A = QR$  ist:

$Q = H_1^T Q_2^T = \begin{pmatrix} -0.89 & -0.45 & 0 \\ 0.45 & -0.89 & 0 \\ 0 & 0 & 1 \end{pmatrix}, R = \begin{pmatrix} -\sqrt{5} & -6.71 & 0.45 \\ 0 & -2.19 & 1.34 \\ 0 & 0 & -1.79 \end{pmatrix}$

Probe

- $QR = A$  (bis auf Rundungsfehler)
- $Q^T Q = Q Q^T = I$  (Orthogonalität)
- $R$  ist obere Dreiecksmatrix

Wichtige Beobachtungen

- Die Wahl des Vorzeichens bei der Berechnung von  $v_k$  ist entscheidend für die numerische Stabilität
- Ein falsches Vorzeichen kann zu Auslöschung führen
- Der Betrag der Diagonalelemente in  $R$  entspricht der Norm der transformierten Spalten
- $Q$  ist orthogonal: Spaltenvektoren sind orthonormal

QR-Zerlegung Implementation

```
1 def householder_vector(x):
2     # Berechne Householder-Vektor fuer Spalte x
3     alpha = np.linalg.norm(x)
4     v = x.copy()
5     v[0] += np.sign(x[0]) * alpha
6     v = v / np.linalg.norm(v)
7     return v
8
9 def householder_reflection(A, k):
10    m, n = A.shape
11    v = householder_vector(A[k:, k])
12
13    # Householder-Matrix anwenden
14    H = np.eye(m-k)
15    H -= 2 * np.outer(v, v)
16
17    # Auf Untermatrix anwenden
18    A[k:, k:] = H @ A[k:, k:]
19    return A
20
21 def qr_householder(A):
22    m, n = A.shape
23    R = A.copy()
24    Q = np.eye(m)
25
26    for k in range(n):
27        v = householder_vector(R[k:, k])
28        H = np.eye(m)
29        H[k:, k:] -= 2 * np.outer(v, v)
30        R = H @ R
31        Q = Q @ H.T
32
33    return Q, R
```

Numerische Vorteile

- Numerisch stabil
- Keine Wurzeloperationen während der Elimination
- Orthogonalität der Transformation bleibt erhalten
- Gute Eignung für Eigenwertberechnung

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:

$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}, \|A\|_2 = \sqrt{\rho(A^T A)}$

$\infty$ -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  $Ax = b, A\tilde{x} = \tilde{b}$

Absoluter Fehler:

Relativer Fehler:

$\|x - \tilde{x}\| \leq \|A^{-1}\| \cdot \|b - \tilde{b}\| \quad \frac{\|x - \tilde{x}\|}{\|x\|} \leq \text{cond}(A) \cdot \frac{\|b - \tilde{b}\|}{\|b\|}$

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
- $\text{cond}(A) \rightarrow \infty$ : singular

Konditionierung

$A = \begin{pmatrix} 1 & 1 \\ 1 & 1.01 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 2.01 \end{pmatrix}$

Konditionszahl:  $\text{cond}(A) = \|A\| \cdot \|A^{-1}\| \approx 400$

Fehlerabschätzung

Absoluter Fehler:

$\|x - \tilde{x}\| \leq 400 \cdot 0.01 = 4$

Relativer Fehler:

$\frac{\|x - \tilde{x}\|}{\|x\|} \leq 400 \cdot \frac{0.01}{2} = 2$



Iterative Verfahren

Zerlegung der Systemmatrix

Für iterative Verfahren wird  $A$  zerlegt in:  $A = L + D + R$

- $L$ : 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:

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

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  $\epsilon$  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:  $\text{Re}(z) = x$  Konjugation:  $\bar{z} = x - iy$

Imaginärteil:  $\text{Im}(z) = y$  Betrag:  $|z| = \sqrt{x^2 + y^2} = \sqrt{z \cdot z^*}$

Darstellungsformen

- Normalform:  $z = x + iy$
- Trigonometrische Form:  $z = r(\cos \varphi + i \sin \varphi)$
- Exponentialform:  $z = re^{i\varphi}$

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

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

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

Umrechnung zwischen Darstellungsformen komplexer Zahlen

Von Normalform in trigonometrische Form/Exponentialform

- 1. Berechne Betrag  $r = \sqrt{x^2 + y^2}$
- 2. Berechne Winkel mit einer der Formeln:
  - $\varphi = \arctan(\frac{y}{x})$  falls  $x > 0$
  - $\varphi = \arctan(\frac{y}{x}) + \pi$  falls  $x < 0$
  - $\varphi = \frac{\pi}{2}$  falls  $x = 0, y > 0$
  - $\varphi = -\frac{\pi}{2}$  falls  $x = 0, y < 0$
  - $\varphi$  unbestimmt falls  $x = y = 0$
- 3. Trigonometrische Form:  $z = r(\cos \varphi + i \sin \varphi)$
- 4. Exponentialform:  $z = re^{i\varphi}$

Von trigonometrischer Form in Normalform

- 1. Realteil:  $x = r \cos \varphi$
- 2. Imaginärteil:  $y = r \sin \varphi$
- 3. Normalform:  $z = x + iy$

Von Exponentialform in Normalform/trigonometrische Form

- 1. Trigonometrische Form durch Euler-Formel:

$$re^{i\varphi} = r(\cos \varphi + i \sin \varphi)$$

- 2. Dann wie oben in Normalform umrechnen

Wichtige Hinweise:

- Achten Sie auf das korrekte Quadranten beim Winkel
- Winkelfunktionen im Bogenmaß verwenden
- Bei Umrechnung in Normalform Euler-Formel nutzen
- Vorzeichen bei Exponentialform beachten

Darstellungsformen Gegeben:  $z = 3 - 11i$  in Normalform

$$r = \sqrt{3^2 + 11^2} = \sqrt{130}, \quad \varphi = \arcsin\left(\frac{11}{\sqrt{130}}\right) = 1.3\text{rad} = 74.74^\circ$$

Trigonometrische Form:  $z = \sqrt{130}(\cos(1.3) + i \sin(1.3))$

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:

$$\begin{aligned} z_1 \cdot z_2 &= (x_1 x_2 - y_1 y_2) + i(x_1 y_2 + x_2 y_1) \\ &= r_1 r_2 e^{i(\varphi_1 + \varphi_2)} \text{ (in Exponentialform)} \end{aligned}$$

Division:

$$\begin{aligned} \frac{z_1}{z_2} &= \frac{z_1 \cdot z_2^*}{z_2 \cdot z_2^*} = \frac{(x_1 x_2 + y_1 y_2) + i(y_1 x_2 - x_1 y_2)}{x_2^2 + y_2^2} \\ &= \frac{r_1}{r_2} e^{i(\varphi_1 - \varphi_2)} \text{ (in Exponentialform)} \end{aligned}$$

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:  $z_k = \sqrt[n]{r} e^{i \frac{\varphi + 2\pi k}{n}}, k = 0, 1, \dots, n - 1$

Fundamentalsatz der Algebra

Eine algebraische Gleichung n-ten Grades mit komplexen Koeffizienten:

$$a_n z^n + a_{n-1} z^{n-1} + \dots + a_1 z + a_0 = 0$$

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  $x \in \mathbb{C}^n \setminus \{0\}$  gibt mit:

$Ax = \lambda x$

Der Vektor  $x$  heißt dann Eigenvektor zum Eigenwert  $\lambda$ .

Bestimmung von Eigenwerten

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

Vielfachheiten

Für einen Eigenwert  $\lambda$  unterscheidet man:

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

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  $\lambda_i$ :
  - System  $(A - \lambda_i I_n)x = 0$  aufstellen
  - Lösungsraum = Eigenraum bestimmen
  - Basis des Eigenraums = linear unabhängige Eigenvektoren

Eigenwertberechnung

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

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.  $\text{tr}(A) = \lambda_1 + \lambda_2 + \lambda_3 = 6$
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  $\lambda$ , so ist  $Tx$  Eigenvektor von  $A$  zum Eigenwert  $\lambda$
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  $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$  konvergiert die Folge:

$v^{(k+1)} = \frac{Av^{(k)}}{\|Av^{(k)}\|_2}, \quad \lambda^{(k+1)} = \frac{(v^{(k)})^T Av^{(k)}}{(v^{(k)})^T v^{(k)}}$

gegen einen Eigenvektor  $v$  zum betragsmäßig größten Eigenwert  $\lambda_1$ .

Von-Mises-Iteration durchführen

1. Wähle Startvektor  $v^{(0)}$  mit  $\|v^{(0)}\|_2 = 1$
2. Für  $k = 0, 1, 2, \dots$ :
  - 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

```
1 import numpy as np
2 def power_iteration(A, tol=1e-10, max_iter=100):
3     n = A.shape[0]
4     v = np.random.rand(n)
5     v = v / np.linalg.norm(v)
6     for i in range(max_iter):
7         w = A @ v
8         v_new = w / np.linalg.norm(w)
9         # Rayleigh-Quotient
10        lambda_k = v_new.T @ A @ v_new
11        if np.linalg.norm(v_new - v) < tol:
12            return lambda_k, v_new
13        v = v_new
14    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  $i = 0, 1, 2, \dots$ :
  - QR-Zerlegung:  $A_i = Q_i R_i$
  - Neue Matrix:  $A_{i+1} = R_i Q_i$
  - Update:  $P_{i+1} = P_i Q_i$

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  $\approx 0$ ?
3. Eigenwerte: Diagonalelemente der Endmatrix
4. Eigenvektoren: Spalten von  $P = P_1 P_2 \dots P_k$

**QR-Verfahren** Implementation in Python

```
1 def qr_algorithm(A, tol=1e-10, max_iter=100):
2     n = A.shape[0]
3     Q_prod = np.eye(n)
4     A_k = A.copy()
5
6     for k in range(max_iter):
7         # QR decomposition
8         Q, R = np.linalg.qr(A_k)
9         # New iteration
10        A_k = R @ Q
11        # Update transformation matrix
12        Q_prod = Q_prod @ Q
13
14        # Check convergence
15        if np.abs(np.tril(A_k, -1)).max() < tol:
16            break
17
18    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

```
1 from decimal import Decimal, getcontext
2
3 # Set precision
4 getcontext().prec = 40
5
6 # Precise calculation
7 x = Decimal('1.0') / Decimal('7.0')
8 print(x)    # 0.1428571428571428571428571428571428571428
```

## Examples