Rechnerarithmetik

Numerische Mathematik

Lösen von mathematischen Problenstellung auf möglichst genaue, effiziente und stabile Weise auf einem Rechner mit endlicher Arithmetik.

Berechnung von Lösungen mit Hilfe von Algorithmen.
Zwei Typen von (iterativen) Verfahren:

1. Direkt: Liefert exakte Lösung eines Problems
   (z.B. gausssche Eliminationsverfahren eines linearen Gleichungssystems)
2. Näherungsverfahren / Approximation
   (z.B. Wert eines Integrals mittels Newton-Verfahren)

Fixpunktzahl (floating point) vs Gleitpunktzahl

Fixpunktzahl: 123.45
Gleitpunktzahl: $1.2345 \cdot 10^2$
Normierte Gleitpunktzahl: $0.12345 \cdot 10^3$

Der **Exponent** repräsentiert die **Position** des **Dezimalpunktes**.
z.B. $0.12345_{10} \cdot 10^3 = 123.45_{10}$

Der darstellbare Zahlenbereich bei gleichem Speicherbedarf bei Gleitpunktzahlen ist enorm gross.

Maschinenzahlen

Menge M = Alle Maschinenzahlen

$$M = \left\{ (\pm 0.m_1m_2)_2 \cdot 2^{(\pm e_1)_2} \mid m_1, m_2, e_1 \in \{0,1\} \text{ und } m_1 \neq 0 \right\} \cup \{0\}$$

Basis Mantisse Exponent

Maschinenzahl (beliebiger Basis) → Dezimalzahl

$$(0.111)_2 \cdot 2^{(3)_2} = (2^{-1} + 2^{-2} + 2^{-3})_{10} \cdot 2^{(3)_{10}} = 7$$
$$= (1/2 + 1/4 + 1/8)_{10} \cdot 2^{(3)_{10}} = 7$$
$$= (2^2 + 2^1 + 2^0)_{10} = 7$$

$$(0.205)_8 \cdot 8^{(2)_8} = (2 \cdot 8^{-1} + 5 \cdot 8^{-3})_{10} \cdot 8^{(2)_{10}} = 16.625$$
$$= (2 \cdot 8^1 + 5 \cdot 8^{-1})_{10} = 16.625$$

$$\hat{w} = \sum_{i=1}^{n} m_i B^{\hat{e}-i}$$

n = Anzahl Mantisse

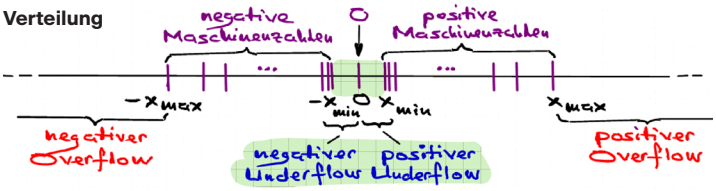
Dezimalzahl → Maschinenzahl (beliebiger Basis)

Bsp: 58.1424_{10} umrechnen in Basis 50 mit Exponent 2

$58.1424 \div 50^1 = 1$ Rest 8.1424
$8.1424 \div 50^0 = 8$ Rest 0.1424
$0.1424 \div 50^{-1} = 7$ Rest 0.0024
$0.0024 \div 50^{-2} = 6$ Rest 0

Kontrolle:
$0.1876_{50} \cdot 50^{(2)_{50}} = (1 \cdot 50^{-1} + 8 \cdot 50^{-2} + 7 \cdot 50^{-3} + 6 \cdot 50^{-4})_{10} \cdot 50^{(2)_{10}} = 58.1424$
$= (1 \cdot 50^1 + 8 \cdot 50^0 + 7 \cdot 50^{-1} + 6 \cdot 50^{-2})_{10} = 58.1424$

Verteilung



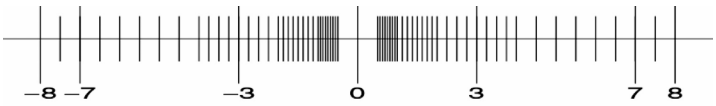
x_{min} = kleinse Positive Zahl, x_{max} = grösste positive Zahl
Zahlen im **Overflow:** Entweder Fehlermeldung (Java) oder auf ±inf (Unendlich) aufgerundet (Python).
Zahlen im **Underflow** werden auf 0 abgerundet.

Mehr Mantissenbits = mehr Zahlen (= mehr Genauigkeit)
Mehr Exponentenbits = grösserer Bereich

Bei Veränderung der Mantisse enstehen zwischen den alten Maschinenzahlen zusätzlich neue Maschinenzahlen.

Bei Veränderung des Exponenten entstehen unterhalb und auserhalb der alten maschinenzahlen zusätzliche neue Maschinenzahlen.

Maschinenzahlen sind **nicht gleichmässig verteilt**



Float64 (Python)

- Eine normale Zahl x in float64 hat folgende Gestalt:

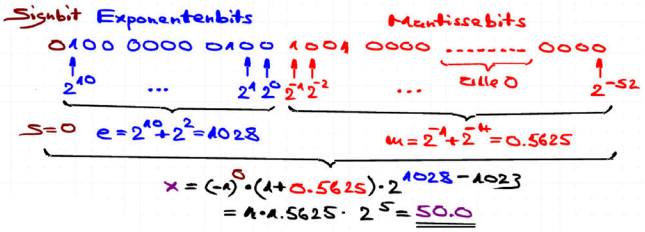
$$x = (-1)^S \cdot (1+m) \cdot 2^{e-b}$$

mit
S aus $\{0,1\}$
$m = (0.m_1m_2 \ldots m_{52})_2$ mit $m_1, m_2, \ldots, m_{52}$ aus $\{0,1\}$
$e = (e_1e_2 \ldots e_{11})_2$ mit $e_1, e_2, \ldots, e_{11}$ aus $\{0,1\}$,
wobei nicht alle 0 und nicht alle 1 sind
$b = (11\,1111\,1111)_2 = 1023$

- x wird wie folgt abgespeichert:

Bemerkung:
- Die Normierung der **Mantisse m** erfolgt dass **1** zu **m** addiert wird.
- **Exponent e** hat kein Vorzeichen. Es wird die Zahl **b von e abgezogen um negative Zahlen** zu erhalten. b heisst **Bias**
- **Kleinste Mantisse: 0, Kleinster Exponent: 1**

Bsp:



Kleinste positive normale Float64:

$$x_{min} = (-1)^0 \cdot (1+0) \cdot 2^{1-1023} = 1 \cdot 1 \cdot 2^{-1022} = 2.23 \cdot 10^{-308}$$

IEEE-Formate:
Single Precision (32 bit)
V EEEEEEEE MMMMMMMMMMMMMMMMMMMMMMM

Double Precision (64 bit), 11 Exponentenbits, 52 Mantissebits
V EEEEEEEEEEE MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM

$$0\ 10000000\ 00000000000000000000 = +1.0 * 2^{128-127} = 2$$
$$0\ 10000001\ 10100000000000000000 = +1.101 * 2^{129-127} = 6.5$$
$$1\ 10000001\ 10100000000000000000 = -1.101 * 2^{129-127} = -6.5$$
$$0\ 00000001\ 00000000000000000000 = +1.0 * 2^{1-127} = 2^{-126} = x_{\min}$$
$$0\ 00000000\ 10000000000000000000 = +0.1 * 2^{-126} = 2^{-127}$$
$$0\ 00000000\ 00000000000000000001 = +0.0\ldots01 * 2^{-126} = 2^{-149}$$
$$= \text{kleinste darstellbare Zahl}$$

Kleinste **positive** Maschinenzahl

$$x_{min} = B^{e_{min}-1}$$

e_{min} = kleinster negativer Exponent
e_{max} = grösster positiver Exponent
n = Anzahl Mantissestellen
B = Basis

Grösste **positive** Maschinenzahl

$$x_{max} = B^{e_{max}} - B^{e_{max}-n} = (1 - B^{-n})B^{e_{max}}$$

Anzahl Maschinenzahlen auf einem Rechner,
der 15-stellige Gleitpunktzahlen mit 5-stelligen Exponenten:

$$(\pm 0.\overset{=1}{m_1}m_2 \ldots m_{15})_2 \cdot 2^{(e_1e_2 \ldots e_5)_2}$$

$$2 \cdot 2^{14} \cdot (2 \cdot 2^5 - 1) + 1 = 2'064'385$$

Vorzeichen fällt weg wenn Exponent = 0 → 0

Rundungsfehler und Maschinengenauigkeit

Numerische Stabilität

Für einen Computer sind z.B. Additionen sind nicht assoziative:

$$(a+b)+c \neq a+(b+c)$$

Absoluter und relativer Fehler

Es seien
x der exakte Wert und $\tilde{x}$ ein approximativer Wert.
Dann heissen
$|\tilde{x}-x|$ absoluter Fehler und $\left|\frac{\tilde{x}-x}{x}\right|$ relativer Fehler.
Relative Fehler sind aussagekräftiger.

Grösster absoluter Fehler ist die Mitte zwischen den letzten beiden Maschinenzahlen (an beiden Enden).

Rundungsprinzipien

1. Rounding to nearest
2. Rounding **away from 0** (wenn genau in der Mitte zweier Maschinenzahlen)

Maschinengenauigkeit

Maximaler Relativer Rundungsfehler (= Obere Schranke)
Durch Runden einer **reelen Zahl** x auf eine **Maschinenzahl** $\tilde{x}$.

$$\left|\frac{\tilde{x}-x}{x}\right| \leq \frac{B}{2} \cdot B^{-n}$$

n = Anzahl Mantissestellen
B = Basis

Maschinenepsilon *eps*

$$eps := \frac{B}{2} \cdot B^{-n} = \frac{1}{2} \cdot B^{1-n}$$

eps ist die **grösste positive Maschinenzahl** x**,** für welche auf dem Rechner gilt:
1 + x = 1 für x > eps und **1 + x > 1** für x ⩾ eps

eps ist die **kleinste positive Maschinenzahl** x**,** für die auf dem Rechner gilt:
1 + eps ≠ 1

Bsp: $M = \{\pm 0.m_1 m_2 \cdot 10^{\pm e_1} \mid m_1, m_2, e_1 \in \{0, 1, ..., 9\}, m_1 \neq 0\} \cup \{0\}$
eps = 0.05 $(= 10/2 \cdot 10^{-2})$

x = 0.04 ist kleiner als eps, wird auf 1 gerundet:
$1 + 0.04 = 1.04 = 0.104 \cdot 10^1$, gerundet auf $0.10 \cdot 10^1 = 1$

x = 0.06 ist grösser als eps, wird nicht auf 1 gerundet:
$1 + 0.06 = 1.06 = 0.106 \cdot 10^1$, gerundet auf $0.11 \cdot 10^1 = 1.1$

Gleitpunktarithmetik

Bsp. **Addition** und **Multiplikation** mit Basis 10 und 2-stellige Mantisse
eps = 0.05

- **Addition** von $x = 1350$ und $y = 245$
  Exaktes Ergebnis: $x+y = 1595$
  Approximatives Ergebnis auf Maschine:
  $1350 \rightarrow 0.135 \cdot 10^4 \rightarrow 0.14 \cdot 10^4 \rightarrow 0.140 \cdot 10^4$
  $245 \rightarrow 0.245 \cdot 10^3 \rightarrow 0.25 \cdot 10^3 \rightarrow 0.025 \cdot 10^4$
  1. Normieren 2. Runden 3. Exponenten angleichen
  $\rightarrow 0.165 \cdot 10^4 \rightarrow 0.17 \cdot 10^4 \rightarrow 1700$
  4. Addieren 5. Runden 6. Denormieren
  Relativer Fehler: $\left|\frac{1700-1595}{1595}\right| = \frac{105}{1595} \approx 0.066 = 6.6\%$

- **Multiplikation** von $x = 1350$ und $y = 245$
  Exaktes Ergebnis: $x \cdot y = 330'750$
  Approximatives Ergebnis auf Maschine:
  $1350 \rightarrow 0.135 \cdot 10^4 \rightarrow 0.14 \cdot 10^4$
  $245 \rightarrow 0.245 \cdot 10^3 \rightarrow 0.25 \cdot 10^3$
  1. 2. 3. Multiplizieren $\rightarrow 0.035 \cdot 10^7$
  $\rightarrow 0.35 \cdot 10^6 \rightarrow 350'000$
  4. Normieren 5. Denormieren
  Relativer Fehler: $\left|\frac{350'000-330'750}{330'750}\right| = \frac{19'250}{330'750} \approx 0.058 = 5.8\%$

Auslöschung

Bei **Substraktion** von fast gleich grossen Zahlen, die nahe beeinander liegen, kann ein **grosser relativer Fehler** auftreten.

Fehlerfortpflanzung

Funktionsauswertung

Näherung für den **Absoluter Fehler**

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

Näherung für den **Relativer Fehler**

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

Konditionszahl *K*

Fehlervortpflanzung
Verhältnis **relativer Outputfehler** zu **relativer Inputfehler**.

Gibt näherungsweise an, um wieviel sich der
relative Fehler von *x* vergrössert bei einer Funktinosauswertung *f(x)*.
Relativer Fehler von *x* pflanzt sich näherungsweise mit der Konditionszahl fort.

$$K(x) := \frac{|f'(x)| \cdot |x|}{|f(x)|} \approx \frac{1}{\frac{|\tilde{x}-x|}{|x|}} \cdot \frac{|f(\tilde{x})-f(x)|}{|f(x)|}$$

$$\frac{|f(\tilde{x})-f(x)|}{|f(x)|} \approx K(x) \cdot \frac{|\tilde{x}-x|}{|x|} \qquad \frac{|\tilde{x}-x|}{|x|} \approx \frac{1}{K(x)} \cdot \frac{|f(\tilde{x})-f(x)|}{|f(x)|}$$

Mit welchem absoluten Fehler $\tilde{x}$ höchstens behaftet sein darf,
damit der relative Fehler von f(x) höchstens ε beträgt.

$$|\tilde{x}-x| \leq \frac{\varepsilon}{K(x) \cdot x}$$

ε = Maximaler relativer Fehler

Maximaler relativer Fehler = $K \cdot eps$

Bsp $f(x) = x^n$

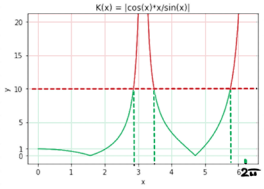
$$K(x) = \left|\frac{f'(x) \cdot x}{f(x)}\right| = \left|\frac{n \cdot x^{n-1} \cdot x}{x^n}\right| = |n|$$

Für grosse Exponenten n wird der relative Fehler stark vergrössert. Der relative hängt nicht von x ab.

x^n ist gut kondizioniert für n ∈ [-10, 10]

Beispiel Sinusfunktion $f(x) = \sin(x)$ mit x im Bogenmass

$K(x) = \left|\frac{f'(x) \cdot x}{f(x)}\right| = \left|\frac{\cos(x) \cdot x}{\sin(x)}\right|$



sin(x) ist im Bereich $0 \leq x \leq 2\pi$
gut konditioniert für $0 \leq x \leq 2.86$ und $3.48 \leq x \leq 5.76$
ansonsten schlecht konditioniert

Gute / schlechte Konditionierung

$K(x) \leq 10$ heisst $f(x)$ gut konditioniert,
$K(x) > 10$ heisst $f(x)$ schlecht konditioniert.

$K(x) < 1$ wird der relative Fehler kleiner,
$K(x) = 1$ bleibt der relative Fehler gleich,
$1 < K(x) \leq 10$ wird der relative Fehler leicht grösser,
$K(x) > 10$ wird der relative Fehler stark grösser.

Ist die Kondition sehr schlecht (z.B. hat eine Polstelle), kann eine **Auslöschung** trotz algebraischer Umformung an dieser Stelle nicht vermieden werden. Grosse Fehler bei schlechter Konditionierung sind demzufolge unvermeidlich.

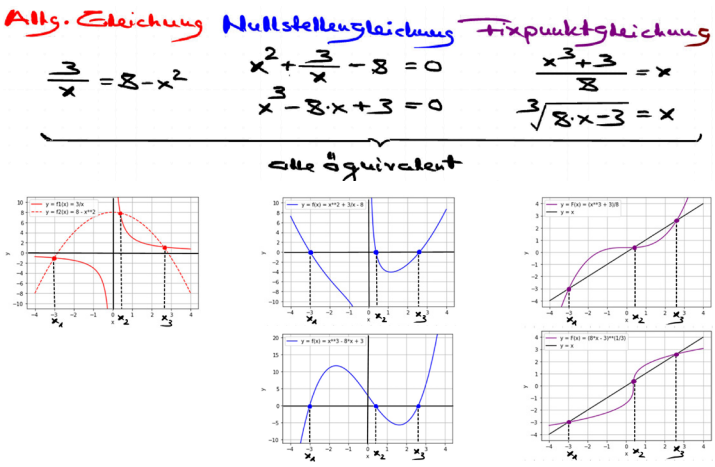
Absoluter Fehlerfaktor

Absoluter Fehler von x : pflanzt sich näherungsweise mit dem Faktor $f'(x)$ fort
$|f'(x)| < 1$ absoluter Fehler wird kleiner
$|f'(x)| > 1$ absoluter Fehler wird grösser

$$f'(x) \approx \frac{f(\tilde{x})-f(x)}{\tilde{x}-x}$$

Fixpunktlgleichung $x = F(x)$

Numerisches Bestimmung von Nullstellen



Fixpunktlgleichung $F(x) = x$

Für eine **nichtlineare Gleichung.**
Zu jedem Fixpunktproblem gibt es ein äquivalentes Nullstellenproblem.
Die Funktion *F* bildet die Punkte *x* auf sich selbst ab. Einfach und effizient.
Vorgehen: Nullstellengleichung umformen nach $F(x) = x$

Fixpunktieration

$x_{n+1} = F(x_n)$: x_0, $x_1 = F(x_0)$, $x_2 = F(x_1)$, $x_{n+1} = F(x_n)$

Konvergenz: Fixpunkt $\overline{x}$ ist **anziehend** wenn

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

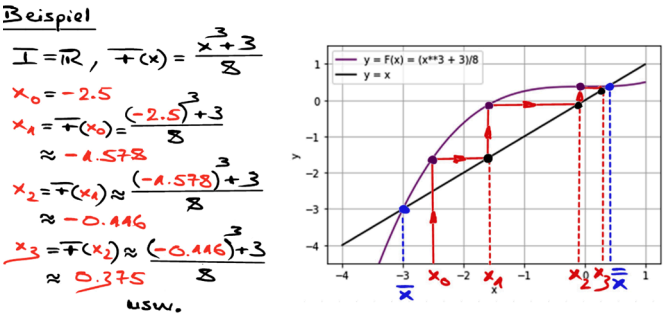
x_n bzw. die Fixpunktiteration konvergiert gegen $\overline{x}$, falls der Startwert x_0 nahe genug bei $\overline{x}$ liegt.

Differgenz: Fixpunkt $\overline{x}$ ist **abstossend** wenn

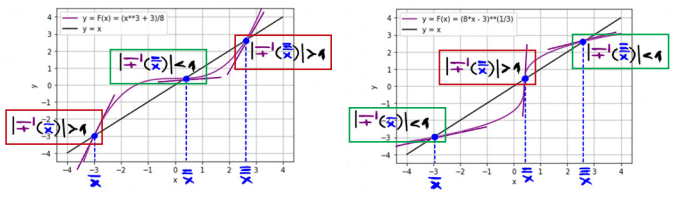
$$| F'(\overline{x}) | > 1$$

x_n bzw. Fixpunkiteration divergiert.

Kriterium: Fixpukte mit **Steigung betragsmässig > 1 sind abstossend**.
D.h. **nicht alle** Fixpunkte können mit Fixpunktieration bestimmt werden.




```
def fixpunktiteration(F, x0, n):
    x = np.zeros(n+1)
    x[0] = x0
    for i in range(1, n+1):
        x[i] = F(x[i-1])
    return x
```



Achtung, für **gerade Wurzeln:** $\pm\sqrt[4]{...}$

Eine Funktion *f* ist **differenzierbar,** wenn es eine Ableitung *f'* gibt über einem gegeben Interval (Definitionsbereich). D.h. gibt es einen Grenzwert an einer Stelle x_0 im Interval.

Banachsche Fixpunktsatz

Kriterien für die **Existenz** und **Eindeutigkeit** von Fixpunkten und deren **Fehlerabschätzungen.**

Voraussetzungen: (siehe Grafik unten zur Verdeutlichung)
- $F : [a,b] \longrightarrow [a,b]$ (d.h. F bildet $[a,b]$ auf sich selber ab)
- Ableitung durchgehend < 1, d.h. $\alpha < 1$

Lipschitz-Konstante α

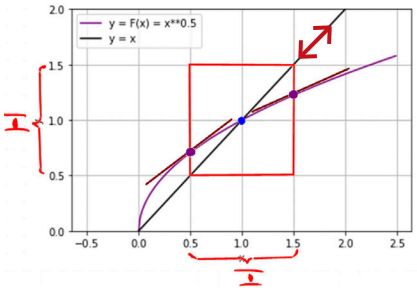
= Betragsmässig grösster Wert der Ableitung *F'*

$$\alpha = \max | F'(x_o) | \quad \text{mit } x_0 \in [a,b]$$

$$| F(x) - F(y) | \leq \alpha | x - y | \qquad \text{für alle } x, y \in [a,b]$$

$$\frac{| F(x) - F(y) |}{| x - y |} \leq \alpha \qquad \text{Steigung} \leq \alpha$$

Wenn $0 < \alpha < 1$ (Steigung ist durchgehend < 1), dann gilt:
- F hat genau einen Fixpunkt $\overline{x}$ in [a,b]
- Die Fixpunktiteration konvergiert gegen $\overline{x}$ für alle Startwerte $x_0 \in [a,b]$
- Es gelten die Fehlerabschätzungen



$\alpha = \max_{0.5 \leq x \leq 1.5} |F'(x)| = |F'(0.5)| = \frac{1}{2\sqrt{0.5}} \approx 0.707$

Weil $\alpha < 1$ ist gilt laut dem Banachschen Fixpunktsatz:
- $F(x) = \sqrt{x}$ besitzt genau einen Fixpunkt in $[0.5, 1.5]$,

Fehlerabschätzung

x_n = n-te Iteration, Max. absoluter Fehler (ε)

$$| x_n - \overline{x} | \leq \frac{\alpha^n}{1-\alpha} | x_1 - x_0 | \qquad \text{a-priori Abschätzung}$$

Basierend auf den **ersten** beiden Iterationen x_0 und x_1
Für (sehr grobe) **Abschlätzung** im Voraus.

$$| x_n - \overline{x} | \leq \frac{\alpha}{1-\alpha} | x_n - x_{n-1} | \qquad \text{a-posteriori Abschätzung}$$

Basierend auf den **letzten** beiden Iterationen x_{n-1} und x_n
Zur **Überprüfung** im Nachhinein.

Absoluter Fehler $| x_n - \overline{x} |$

Mindestanzahl Iterationen *n*

(Beispiel weiter unten)

$$n = \frac{\ln\left(\frac{\epsilon(1-\alpha)}{|x_1 - x_0|}\right)}{\ln(\alpha)}$$

ε = Max. absoluter Fehler

```
def n_apriori(x0, x1, e, a):
    return np.log(e * (1 - a) / np.abs(x1 - x0)) / np.log(a)
```

- Wie viele Iterationen n sind nötig, damit startend mit $x_0 = 1.44$ der absolute Fehler von x_n als Näherung von $\overline{x} = 1$ kleiner als 10^{-6} ist?

Abschätzung von n mit a-priori-Fehlerabschätzungsformel:

$$|x_n - \overline{x}| \leq \frac{\alpha^n}{1-\alpha} \cdot |x_1 - x_0| = 10^{-6}$$

$$\Rightarrow \alpha^n = \frac{10^{-6} \cdot (1-\alpha)}{|x_1 - x_0|} \Rightarrow n = \frac{\log\left(\frac{10^{-6} \cdot (1-0.707)}{0.24}\right)}{\log(1-0.707)} \approx 35$$

Newton-Verfahren

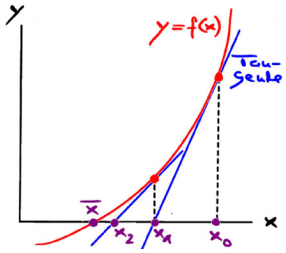
Tangentenverfahren von Newton. **Numerisches Bestimmung von Nullstellen.**

Newton-Verfahren

Vorteil gegenüber Fixpunktiteration: konvergiert fast **immer** (**ausser** $f'(x_0) = 0$ oder f schneidet Abzisse nie, Bsp. $x \cdot e^x$ für $x_0 < -1$).
Nachteil: Funktion muss differenzierbar sein.

Funktion f muss auf **Nullstellenform** gebracht werden.

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




```
def newton(f, df, x0, n):
    x = np.zeros(n+1)
    x[0] = x0
    for i in range(1, n+1):
        x[i] = x[i-1] - f(x[i-1])/df(x[i-1])
    return x
```

Vereinfachtes Newton-Verfahren

Es wird immer die Startableitung $f'(x0)$ verwendet.
Konvergiert dafür langsamer.

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

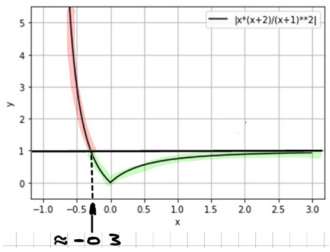
```
def newton_vereinfacht(f, dfx0, x0, n):
    x = np.zeros(n+1)
    x[0] = x0
    for i in range(1, n+1):
        x[i] = x[i-1] - f(x[i-1])/dfx0
    return x
```

Konvergenzkriterium

Die Nullstelle $\underline{x}$, der Startwert und alle Iterierten $x_1, x_2, \ldots$ in einem Intervall [a, b] liegen, so dass:

$$\left| \frac{f(x) \cdot f''(x)}{[f'(x)]^2} \right| < 1 \quad \text{für alle } x \text{ aus [a, b]}$$

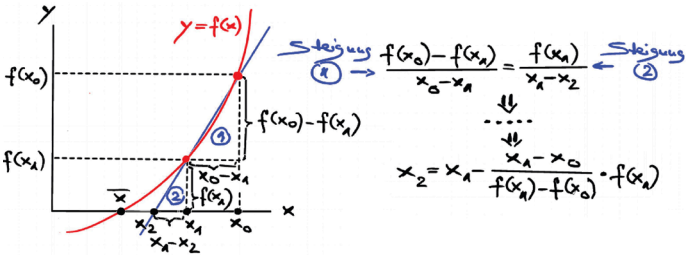
dann **konvergiert** das Newton-Verfahren **garantiert.** Konvergiert auch oft, ohne, dass diese Bedingung zutrifft!



Sekantenverfahren

Keine Ableitung nötig. Es wird der Schnittpunkt von Sekanten durch jeweils zwei Punkte $(x_0, f(x_0))$ und $(x_1, f(x_1))$ mit der x-Achse berechnet.
Benötigt zwei Startwerte.

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




```
def sekantenverfahren(f, x0, x1, n):
    x = np.array([x0, x1])
    for i in range(2, 2+n):
        x0 = x[i-2]
        x1 = x[i-1]
        f1 = f(x1)
        x = np.append(x, x1 - (x1 - x0) / (f1 - f(x0)) * f1)
    return x
```

```
def sekantenverfahren_tol(f, x0, x1, tol):
    x = np.array([x0, x1])
    while f(x1 + tol) * f(x1 - tol) >= 0:
        x0 = x[-2]
        f1 = f(x1)
        x1 = x1 - (x1 - x0) / (f1 - f(x0)) * f1
        x = np.append(x, x1)
    return x
```

Konvergenzgeschwindigkeit

Konvergenzordnung q

Mass für die Konvergenz-Geschwindigkeit.

Für *einfache Nullstellen* konvergiert

- **Vereinfachte Newton-Verfahren,** linear $q = 1$ mit $c < 1$ (langsamstes)
- **< Sekantenverfahren** $q = (1 + \sqrt{5}) / 2 = 1.618\ldots$
- **< Newton-Verfahren,** quadratisch $q = 2$ (schnellstes)

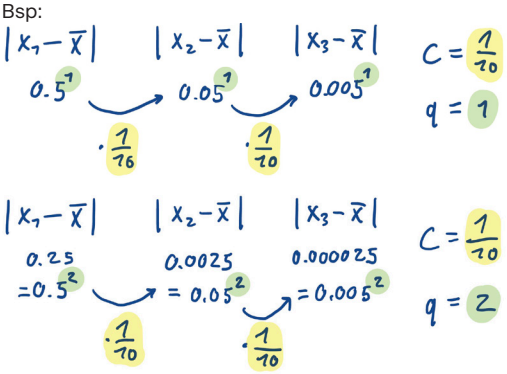
$$\left| x_{n+1} - \overline{x} \right| \leq c \cdot \left| x_n - \overline{x} \right|^q$$

Für $q = 2$:

$$C = \frac{1}{2} \cdot \left| \frac{f''(\overline{x})}{f'(\overline{x})} \right|$$

Falls $q = 1$ verlangt man noch $c < 1$.

Bsp:



Fehlerabschätzungskriterium

Achtung: **Nur für Nullstellenform.**

Kriterium: Wenn $f(a) \cdot f(b) < 0$ (= Vorzeichenwechsel),
dann besitzt f mindestens eine Nullstelle $\bar{x}$ im Interval $[a, b]$.

Wenn $f(x_n - \epsilon) \cdot f(x_n + \epsilon) < 0$ dann gilt $\left| x_n - \bar{x} \right| < \epsilon$

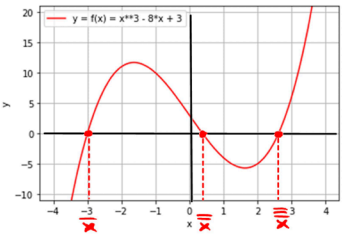
```
def newton(f, df, x0, epsilon):
    x = x0
    while f(x - epsilon)*f(x + epsilon) > 0:
        x = x - f(x)/df(x)
    return x

def f(x): return x**3 - 8.*x + 3
def df(x): return 3*x**2 - 8.
epsilon = 1.e-8

x0 = -2.
xquer = newton(f, df, x0, epsilon)
print('%9.8f' %xquer) # -3.00000000

x0 = 0.
xzweiquer = newton(f, df, x0, epsilon)
print('%9.8f' %xzweiquer) # 0.38196601

x0 = 2.
xdreiquer = newton(f, df, x0, epsilon)
print('%9.8f' %xdreiquer) # 2.61803400
```



Semilogy und loglog

- **Semilogy:** Der Graph einer Exponentialfunktion **f(x) = c · aˣ** in einem Koordinatensystem mit **logarithmischer y-Achse** ist eine Gerade.
  `plt.semilogy(x, y)`
- **Loglog:** Der Graph einer Potenzfunktion **f(x) = c · xᵃ** ist eine Gerade, wenn man **beide Koordinatenachsen logarithmisch** wählt.
  `plt.loglog(x, y)`

$y = \ln(f(x))$ $\quad x \cdot m + b$

Semilogy: $\ln(c \cdot a^x) = \ln(c) + \ln(a^x) = x \cdot \ln(a) + \ln(c)$

Loglog: $\ln(c \cdot x^a) = \ln(c) + \ln(x^a) = \ln(x) \cdot a + \ln(c)$

(i) $10^5 \cdot (2e)^{-\frac{x}{700}} = 10^5 \cdot \left((2e)^{-\frac{1}{700}}\right)^x \rightarrow$ Steigung $\log\left((2e)^{-\frac{1}{700}}\right)$, Achsenabschnitt $\log(10^5)$

(ii) $\frac{5}{\sqrt[3]{2x^2}} = 5 \cdot (2x^2)^{-\frac{1}{3}} = 5 \cdot 2^{-\frac{1}{3}} \cdot x^{-\frac{2}{3}} \rightarrow$ Steigung $-\frac{2}{3}$, Achsenabschnitt $\log(5 \cdot 2^{-\frac{1}{3}})$

Lineare Gleichungssysteme (LGS)

Reguläres LGS $a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n$

Anzahl Gleichungen = Anzahl Unbekannte (= quadratische Matrix)
$\underline{A} \cdot \underline{x} = \underline{b}$ ist genau dann **regulär** wenn $\det(\underline{A}) \neq 0$ → LGS hat **genau 1 Lösung**

Vektor-Matrix Form $\underline{A} \cdot \underline{x} = \underline{b}$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$A \in \mathbb{R}^{n \times n}$ $\quad x \in \mathbb{R}^n$ $\quad b \in \mathbb{R}^n$

$$\begin{array}{l} 20a + 10b \quad\quad = 150 \\ 50a + 30b + 20c = 470 \\ 200a + 150b + 100c = 2150 \end{array} \rightarrow \begin{array}{c|ccc|c} & a & b & c & \\ \hline & 20 & 10 & 0 & 150 \\ & 50 & 30 & 20 & 470 \\ & 200 & 150 & 100 & 2150 \end{array}$$

Determinante einer Koeffizientenmatrix (Dreiecksmatrix):

$$\det(\underline{A}) = (-1)^z \cdot a_{11} \cdot a_{22} \cdot \ldots \cdot a_{nn}$$ z = Anzahl Zeilentausch

Eine det($\underline{A}$) zu berechnen ist **auffändiger**, als eine Lösung zu berechnen.

Gauss-Verfahren

Zum lösen eines LGS Ax = b.

Für **höchstens hundert Unbekannte** reicht das Gauss-Verfahren aus.
Für grössere LGS werden iterative Verfahren verwendet (Fixpunktiteration).

In der Numerik werden nur folgende Äquivalenzumformungen verwendet:

— **Zeilen-Substraktion:** Einer Zeile wird ein Vielfaches einer darüberliegenden Zeile substrahiert.

— **Zeilenvertausch:** Zwei Zeilen werden miteinander vertauscht.

= Standardisierte Vorwärtselimination

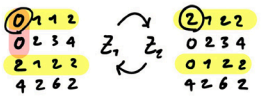
Standardisierte Vorwärtselimination

Ziel ist die Matrix auf eine **rechtsobere Dreiecksmatrix** zu bringen, so dass dann einfach **Rückwärtssubstituiert** (Rückwaertseinsetzen) werden kann.

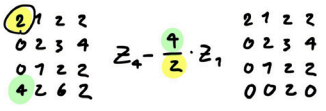
$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}}$$

a_{ij} $\quad i$ = Zeilen $\quad j$ = Spalten

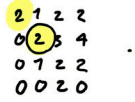
1. Ist $a_{ii} = 0$: Zeilenwechsel mit erster unterhalbliegenden Zeile mit führender Variabel:



2. Subtrahiere jeder unterhalb liegenden Zeilen das $\frac{a_{ji}}{a_{ii}}$-fache der aktuellen Zeile:



3. Gehe zu $a_{j+1\,j+1}$ und wiederhole Verfahren.



Problem → Numerische Fehlerfortpflanzung

In jedem Eliminationsschritt vergrössert sich der absolute Fehler um den Faktor $\left|\frac{a_{ji}}{a_{ii}}\right|$. Wünschenswert wäre $\left|\frac{a_{ji}}{a_{ii}}\right| < 1$.

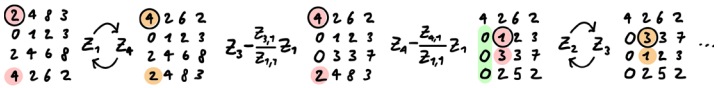
Zusätzlich wird mit jedem Rückwärtseinsetzen der relative Fehler verstärkt (Runden von Gleitpunktzahlen).

Spaltenpivotisierung

Pivot = Führende Variable
Minimiert numerische **Fehler** die durch Rückwärtseinsetzen und Gleitpunktoperationen entstehen.

1. Ermittle das betragsmässig grösste Element in der aktuellen Spalte und vertausche mit aktueller Zeile.

2/3. Wie Standardisierte Vorwärtselimination.



```
def gauss(A,b):
    # Erweiterte Koeffizientenmatrix M = (A|b)
    M = np.c_[A,b].astype(float)
    n = np.size(b)
    for j in range(n): # Spaltenindex
        # Spaltenpivotisierung
        k = np.argmax(np.abs(M[j:n,j])) + j
        if k != j:
            M[[j,k],:] = M[[k,j],:]
        # Vorwaertselimination
        for i in range(j+1, n): # Zeilenindex
            M[i,j+1:] = M[i,j+1:] - M[i,j] / M[j,j] * M[j,j+1:]
            M[i,j] = 0
    # Rueckwaertseinsetzen
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (M[i,n] - np.dot(M[i,0:n], x)) / M[i,i]
    return M, x
```

Dreieckszerlegung

LR-Zerlegung

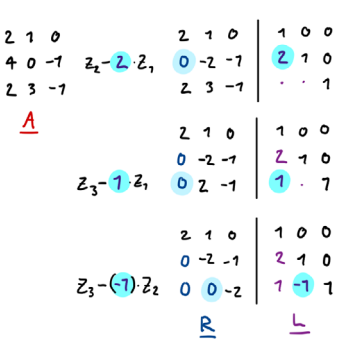
Wenn eine n×n-Matrix mit dem Gauss-Algorithmus **ohne Spaltenpivotisierung** in eine rechtsobere Dreiecksmatrix gebracht werden kann. Dann gilt:

$$\underline{A} = \underline{L} \cdot \underline{R} \qquad \underline{A} \cdot \underline{x} = (\underline{L} \cdot \underline{R}) \cdot \underline{x} = \underline{L} \cdot (\underline{R} \cdot \underline{x}) = \underline{L} \cdot \underline{y} = \underline{b}$$

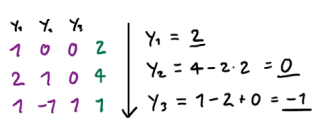
R = rechtsobere Dreiecksmatrix
L = Vielfache der Zeilensubstraktionen (linksuntere Dreiecksmatrix)

Vorgehen

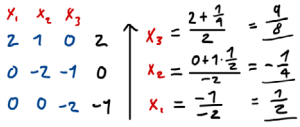
1. LR-Zerlegung:



2. $\underline{L} \cdot \underline{y} = \underline{b}$ Vorwärts einsetzen:



3. $\underline{R} \cdot \underline{x} = \underline{y}$ Rückwärts einsetzen



PLR-Zerlegung

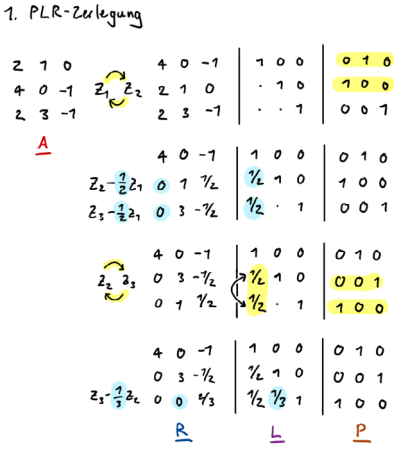
Wenn eine n×n-Matrix mit dem Gauss-Algorithmus **mit Spaltenpivotisierung** in eine rechtsobere Dreiecksmatrix gebracht werden kann. Dann gilt:

$$\underline{P} \cdot \underline{A} = \underline{L} \cdot \underline{R}$$

P: Permutationsmatrix (Einheitsmatrix mit Spalten in anderer Reihenfolge)
L: Achtung, Zeilentausch in L unterhalb der Diagonalen.

Vorgehen

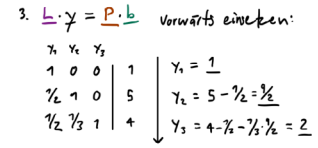
1. PLR-Zerlegung



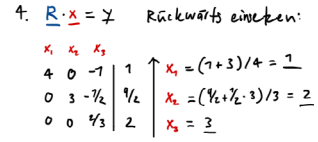
2. $\underline{P} \cdot \underline{b}$

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

3. $\underline{L} \cdot \underline{y} = \underline{P} \cdot \underline{b}$ Vorwärts einsetzen:



4. $\underline{R} \cdot \underline{x} = \underline{y}$ Rückwärts einsetzen:



QR-Zerlegung

$A = Q \cdot R$

$R \cdot x = Q^T \cdot b$ ← Durch Rückwärtseinsetzen lösbar

Zerlegung mit Housholder-Verfahren

$A_1 = \begin{bmatrix} 2 & 2 & -1 \\ 1 & -1 & 0 \\ 2 & 0 & 1 \end{bmatrix}$

1. Iteration (i=1)

$R = A_1 = \begin{bmatrix} 2 & 2 & -1 \\ 1 & -1 & 0 \\ 2 & 0 & 1 \end{bmatrix}$, $a_1 = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}$

$v_1 = a_1 + \text{sign}(a_{11}) \cdot |a_1| \cdot e_1 = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} + (1) \cdot 3 \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix}$

$u_1 = \frac{v_1}{|v_1|} = \frac{1}{\sqrt{30}} \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix}$ u ist v normiert, d.h. u notfalls **normieren**: $\tilde{u} = \frac{u}{|u|}$

$H_1 = I_3 - 2 \cdot u_1 \cdot u_1^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - 2 \cdot \frac{1}{\sqrt{30}} \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix} \cdot \frac{1}{\sqrt{30}} \begin{bmatrix} 5 & 1 & 2 \end{bmatrix}$

$Q_1 = H_1 = \begin{bmatrix} -2/3 & -1/3 & -2/3 \\ -1/3 & 14/15 & -2/15 \\ -2/3 & -4/15 & 11/15 \end{bmatrix}$ Spiegelmatrix

$R_{\text{neu}} = Q_1 \cdot R_{\text{alt}} = \begin{bmatrix} -3 & -1 & 0 \\ 0 & -8/5 & 1/5 \\ 0 & -6/5 & 3/5 \end{bmatrix}$ Spiegelung von A, so dass a_{1y} und a_{1z} auf der x-Achse liegen.

$Q_{\text{neu}} = Q_1^T = \begin{bmatrix} -2/3 & -1/3 & -2/3 \\ -1/3 & 14/15 & -2/15 \\ -2/3 & -4/15 & 11/15 \end{bmatrix}$

2. (und letzte) Iteration (i=2)

$A_2 = \begin{bmatrix} 3 & -1 & 0 \\ 0 & -8/5 & 1/5 \\ 0 & -6/5 & 3/5 \end{bmatrix} = \begin{bmatrix} -8/5 & 1/5 \\ -6/5 & 3/5 \end{bmatrix}$, $a_2 = \begin{bmatrix} -8/5 \\ -6/5 \end{bmatrix}$

... wie 1. Iteration

$Q_2 = \begin{bmatrix} I_1 & 0 \\ 0 & H_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -4/5 & -3/5 \\ 0 & -3/5 & 4/5 \end{bmatrix}$

$R_{\text{neu}} = Q_2 \cdot R_{\text{alt}} = \begin{bmatrix} -3 & -1 & 0 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}$

$R = Q_1 \cdot A_1 \cdot Q_2 \cdot Q_n$
$= Q_n \cdot Q_2 \cdot Q_1 \cdot A_1$

$Q_{\text{neu}} = Q_{\text{alt}} \cdot Q_2^T = \begin{bmatrix} -2/3 & 2/3 & -1/3 \\ -1/3 & -2/3 & -2/3 \\ -2/3 & -1/3 & 2/3 \end{bmatrix}$

$Q = Q_1^T \cdot Q_2^T \cdot Q_n$
$= Q_1^{-1} \cdot Q_2^{-1} \cdot Q_n$
$= (Q_1 \cdot Q_2 \cdot Q_n)^{-1}$

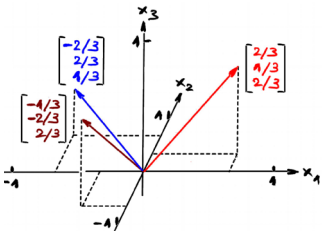
- n–1 Iterationsschritte für eine n×n-Matrix
- Verfahren ergibt immer die gleiche QR-Zerlegung (QR-Zerl. ist eindeutig)
- Verfahren **kann immer durchgeführt** werden
- Benötigt doppelt so viele Punktoperationen wie für die LR-Zerlegung, kann dafür aber numerisch stabiler sein.
- Bestimmung von Eigenwerten von Matrizen.

```
Q,R = np.linalg.qr(A)
x = np.linalg.solve(R, Q.T @ b)
```

```
def qr(A):
    n = np.shape(A)[0]
    R = np.copy(A)
    Q = np.eye(n)
    for i in np.arange(0, n-1):
        a = R[i:,i]
        e = np.zeros(n-i)
        e[0] = 1
        v = a + np.sign(a[0]) * np.linalg.norm(a) * e
        u = v / np.linalg.norm(v)
        H = np.eye(n-i) - 2 * np.outer(u, u)
        Qi = np.eye(n)
        Qi[i:n,i:n] = H
        R = Qi @ R
        Q = Q @ Qi.T
    return Q, R
```

Eigenschaften von Q

- Q ist **regulär** = besitzt eine **Inverse** (d.h. det(Q) ≠ 0 und Q ist eine n×n-Matrix) $Q^T \cdot Q = I_n$
- Zusätzlich ist Q eine **Orthogonalmatrix:** Die **Inverse** einer orthogonalen Matrixist **gleichzeitig ihre Transponierte.** $Q^{-1} = Q^T$
- Die Spalten-Vektoren sind **normiert** und stehen **paarweise senkrecht:**



$Q = \begin{bmatrix} 2/3 & -2/3 & -1/3 \\ 1/3 & 2/3 & -2/3 \\ 2/3 & 1/3 & 2/3 \end{bmatrix}$

- Orthogonale Matrizen (= Rotationsmatrix) beschreiben **Drehungen** und **Spiegelungen** oder Kombinationen daraus. Durch Multiplikation mit einer orthogonalen Matrix. Q können Vektoren gedreht oder gespiegelt werden.

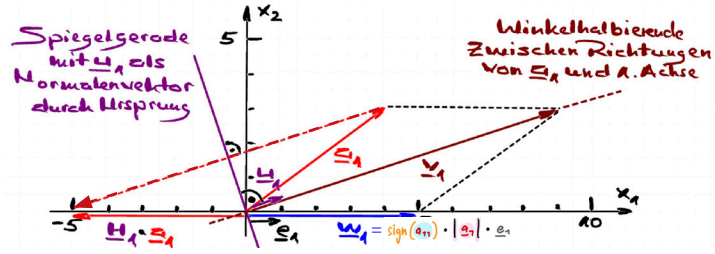
$Q = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$ ist eine Orthogonalmatrix und dreht einen Vektor um den Winkel α (relativ zum Ursprung)

- Alle Vektoren (Spalten) in einer orthogonale Matrix stehen alle **senkrecht** zueinander (= linear unabhängige Basisvektoren).
- Orthogonale Matrizen sind gut konditioniert: Q · x und Q · $\tilde{x}$ liegen nahe beieinander: Fehler von Q · x und Q · $\tilde{x}$ ist gleich dem Fehler von x *und* $\tilde{x}$.

Housholder-Matrizen

Orthogonale Matrix, Transformationsmatrix

= **Householdertransformation:** Beschreibt eine **Spiegelung** an einer zu *v* **senkrechten Geraden** oder **Hyperebene** (Ebene im 3-dimensionalem Raum) durch den Ursprung:



v ist die Winkelhalbierende zwischen a und x-Achse und Normalenvektor der Spiegelgeraden oder Hyperebene.

Die Housholder-Matrizen *H* benutzen um eine n×n-Matrix schrittweise auf eine rechts-obere Dreiecksform zu bringen.

Zum nachrechnen muss die erste Spalte unterhalb der Diagonalen von A zu Null werden, wenn man H mit A multipliziert. Der erste Spaltenvektoren wurde so rotiert, dass alle Komponenten auf der x-Achse liegen.

$H_1 \cdot A = \left(\begin{array}{c|cccc} * & * & * & * & * \\ 0 & & & & \\ 0 & & A_2 & & \\ 0 & & & & \\ 0 & & & & \end{array}\right)$

Vektor- und Matrixnormen

Grössenmasse für Vektoren und Matrizen können unterschiedlich gemessen werden.

Vektornormen

$\|\underline{x}\|_1 = |x_1| + \ldots + |x_n|$ **1-Norm** (Summennorm)

$\|\underline{x}\|_2 = \sqrt{x_1^2 + \ldots + x_n^2}$ **2-Norm** (Euklidsche Norm)

$\|\underline{x}\|_\infty = \max\{|x_1|, \ldots, |x_n|\}$ **∞-Norm** (Maximumnorm)

Matrixnormen

1-Norm (Spaltensummennorm), einzelne Spalten summieren ↓↓

$$\|\underline{A}\|_1 = \max\{|a_{11}| + \ldots + |a_{n1}|, \ \ldots, \ |a_{1n}| + \ldots + |a_{nn}|\}$$

2-Norm (Spektralnorm):

$$\|\underline{A}\|_2 = \max\{\sqrt{|\lambda|} \mid \lambda \text{ ist Eigenwert von } \underline{A}^T \cdot \underline{A}\} = \sqrt{\rho(A^T A)}$$

∞-Norm (Zeilensummennorm), einzelne Zeilen summieren →

$$\|\underline{A}\|_\infty = \max\{|a_{11}| + \ldots + |a_{1n}|, \ \ldots, \ |a_{1n}| + \ldots + |a_{n1}|\}$$

```
np.linalg.norm(x, 1)
np.linalg.norm(x, 2)
np.linalg.norm(x, np.inf)
```

Eigenschaften

$\|\boldsymbol{x}\| \geq 0$ und $\|\boldsymbol{x}\| = 0 \iff x = 0$

$\|\lambda \boldsymbol{x}\| = |\ \lambda\ | \ \|\boldsymbol{x}\|$ λ ist eine reelle Zahl.

$\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ "Dreiecksungleichung"

} Gilt auch für **Matrizen**.

$\| \boldsymbol{Ax} \|_i \ \leq \ \| \boldsymbol{A} \|_i \cdot \| \boldsymbol{x} \|_i \quad (i = 1, 2, \infty)$

$||W \cdot (u + v)|| \ \leq \ ||W|| \cdot (||u|| + ||v||)$

Fehlerrechnung und Aufwandabschätzung

Fehlerfortpflanzung und -abschätzung für **LGS:** $\boldsymbol{A\widetilde{x} = \widetilde{b} = b + \triangle b}$

Abschätzung wenn nur Vektor *b* fehlerbehaftet ist: $A\widetilde{x} = \widetilde{b}$

Maximaler absoluter Fehler von $\widetilde{x}$ (rechte Seite ist zu berechnen)

$$\| \boldsymbol{x} - \widetilde{\boldsymbol{x}} \| \ \leq \ \| \boldsymbol{A}^{-1} \| \cdot \| \boldsymbol{b} - \widetilde{\boldsymbol{b}} \|$$

Maximaler relativer Fehler von $\widetilde{x}$ (rechte Seite ist zu berechnen)

$$\frac{\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|} \leq \| \boldsymbol{A} \| \cdot \| \boldsymbol{A}^{-1} \| \cdot \frac{\|\boldsymbol{b} - \widetilde{\boldsymbol{b}}\|}{\|\boldsymbol{b}\|} \qquad \text{falls } \| \boldsymbol{b} \| \neq 0$$

Konditionszahl der Matrix A bzgl. der verwendeten Norm.
= Faktor, mit der sich der Fehler verstärkt:

$$\text{cond}(\boldsymbol{A}) \ = \ \| \boldsymbol{A} \| \cdot \| \boldsymbol{A}^{-1} \|$$

Schlecht konditioniert: Ist die Kondizionszahl gross (>10), können sich kleine Fehler im Vektor $\underline{b}$ zu grosen Fehlern im Ergebnis $\underline{x}$ verstärken.

```
np.linalg.cond(A, np.inf)
```

Abschätzung wenn Matrix *A* und Vektor *b* fehlerbehaftet sind: $\widetilde{A}\widetilde{x} = \widetilde{b}$

Maximaler relativer Fehler von $\widetilde{x}$

$$\frac{\| \boldsymbol{x} - \widetilde{\boldsymbol{x}} \|}{\| \boldsymbol{x} \|} \leq \frac{\text{cond}(\boldsymbol{A})}{1 - \text{cond}(\boldsymbol{A}) \cdot \frac{\|\boldsymbol{A} - \widetilde{\boldsymbol{A}}\|}{\|\boldsymbol{A}\|}} \cdot \left(\frac{\| \boldsymbol{A} - \widetilde{\boldsymbol{A}} \|}{\| \boldsymbol{A} \|} + \frac{\| \boldsymbol{b} - \widetilde{\boldsymbol{b}} \|}{\| \boldsymbol{b} \|} \right)$$

Weiter sei:

$$\text{cond}(\boldsymbol{A}) \cdot \frac{\| \boldsymbol{A} - \widetilde{\boldsymbol{A}} \|}{\| \boldsymbol{A} \|} \ < \ 1$$

Aufwandabschätzung

Grundrechenoperationen / Floating Point Operations (Flops):
n = Anzahl Unbekannte / Matrix-Grösse

Gauss-Elimination: $\frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{13}{6}n$

LR-Zerlegung: $\frac{2}{3}n^3 + \frac{7}{2}n^2 - \frac{13}{6}n$

QR-Zerlegung: $\frac{5}{3}n^3 + 4n^2 + \frac{7}{3}n - 7$

QR-Zerlegung beansprucht etwas mehr als doppelt so viel Zeit wie die Gauss–Elimination bzw. die LR-Zerlegung.

Ein **G**Hz-Prozessor kann 10^9 Flops/Sekunde ausführen.

Vergleich zu iterativen Lösungsverfahren: Vorteil: Iterativen Lösungsverfahren haben deutlich kleineren Rechenaufwand, nur n^2 statt n^3 Flops. Nachteil: Liefert nur Näherungslösungen und manchmal keine Konvergenz.

Jacobi-Verfahren

Auch **Gesamtschrittverfahren** gennant.

Altes **iteratives Lösungsverfahren** eines LGS. Aufwand n^2 Flops.
Liefert nur Näherungslösungen und haben manchmal keine Konvergenz.

Komponenten werden gleichzeitig berechnet.
Kann parallel berechnet werden.

Idee: LGS in **Fixpunktform** bringen:

$$A \cdot x = b \quad \rightarrow \quad x^{k+1} = \boldsymbol{B}x^{(k)} + \boldsymbol{c} \quad = F(x^k)$$

Vorgehen:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \quad b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$D = \begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{pmatrix}, L = \begin{pmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & a_{32} & 0 \end{pmatrix}, R = \begin{pmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{pmatrix}$$

Diagonalelemente von D dürfen nicht 0 sein → Zeilen tauschen!

$$\begin{aligned} x^{k+1} &= -D^{-1} \cdot (L + R) \cdot x^k + D^{-1} \cdot b \\ &= B \cdot x^k + c \\ &= F(x^k) \end{aligned}$$

x^0 = Startvektor

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

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

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

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

Inverse von *D*

a, b, oder c dürfen nicht 0 sein → Zeilen tauschen!

$$D = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}, \quad D^{-1} = \begin{pmatrix} \frac{1}{a} & 0 & 0 \\ 0 & \frac{1}{b} & 0 \\ 0 & 0 & \frac{1}{c} \end{pmatrix}$$

```
def jaccobi_fast(A, b, x0, k):
    d = np.diag(A); D = np.diag(d)
    LplusR = A - D
    Dinv = 1. / d
    c = Dinv * b
    n = np.shape(A)[0]; B = np.zeros((n, n))
    for i in range(n): B[:,i] = -Dinv * LplusR[:,i]
    for j in range(k): x0 = B @ x0 + c
    return x0
```

Gauss-Seidel-Verfahren

Auch **Einzelschrittverfahren** genannt.

Altes iteratives Lösungsverfahren eines LGS.
Konvergiert schneller als Jacobi-Verfahren. Allerdings wird das Jacobi-Verfahren in der Praxis bevorzugt.

Komponenten werden einzeln berechnet.

Idee: LGS in **Fixpunktform** bringen:

$$\begin{aligned} x^{k+1} &= -(D+L)^{-1} \cdot R \cdot x^k + (D+L)^{-1} \cdot b \\ &= B \cdot x^k + c \\ &= F(x^k) \end{aligned}$$

$$\begin{aligned} D \cdot x^{(k+1)} &= -L \cdot x^{(k+1)} - R \cdot x^{(k)} + b \\ D \cdot x^{(k+1)} + L \cdot x^{(k+1)} &= -R \cdot x^{(k)} + b \\ (D+L) \cdot x^{(k+1)} &= -R \cdot x^{(k)} + b \\ x^{(k+1)} &= (D+L)^{-1} \cdot (-R \cdot x^{(k)} + b) \\ x^{(k+1)} &= -(D+L)^{-1} \cdot R \cdot x^{(k)} + (D+L)^{-1} \cdot b \end{aligned}$$

```
def gauss_seidel(A, b, x0, k):
    d = np.diag(A)
    D = np.diag(d)
    R = np.triu(A) - D
    LplusD = A - R
    LplusDinv = np.linalg.inv(LplusD)
    B = -LplusDinv @ R
    c = LplusDinv @ b
    for j in range(k): x0 = B @ x0 + c
    return x0
```

Bsp. mit Unbekannter

$$A = \begin{pmatrix} 1 & \varepsilon \\ \varepsilon & -1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Jacobi

$$x^{(1)} = -\underbrace{\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}}_{D^{-1}} \cdot \underbrace{\begin{pmatrix} 0 & \varepsilon \\ \varepsilon & 0 \end{pmatrix}}_{L+R} \cdot \underbrace{\begin{pmatrix} 1 \\ -1 \end{pmatrix}}_{x^{(0)}} + \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}}_{D^{-1}} \cdot \underbrace{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}_{b}$$

$$= \begin{pmatrix} \varepsilon \\ \varepsilon \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} \varepsilon + 1 \\ \varepsilon - 1 \end{pmatrix}$$

Gauss-Seidel

$$(D+L)^{-1} = \begin{pmatrix} 1 & 0 \\ \varepsilon & -1 \end{pmatrix}^{-1} = \frac{1}{-1} \begin{pmatrix} -1 & 0 \\ -\varepsilon & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \varepsilon & -1 \end{pmatrix}$$

$$x^{(1)} = -\underbrace{\begin{pmatrix} 1 & 0 \\ \varepsilon & -1 \end{pmatrix}}_{(D+L)^{-1}} \cdot \underbrace{\begin{pmatrix} 0 & \varepsilon \\ \varepsilon & 0 \end{pmatrix}}_{R} \cdot \underbrace{\begin{pmatrix} 1 \\ -1 \end{pmatrix}}_{x^{(0)}} + \underbrace{\begin{pmatrix} 1 & 0 \\ \varepsilon & -1 \end{pmatrix}}_{(D+L)^{-1}} \cdot \underbrace{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}_{b}$$

$$= \begin{pmatrix} \varepsilon \\ \varepsilon^2 \end{pmatrix} + \begin{pmatrix} 1 \\ \varepsilon - 1 \end{pmatrix} = \begin{pmatrix} \varepsilon + 1 \\ \varepsilon^2 + \varepsilon - 1 \end{pmatrix}$$

Konvergenz (Jacobi und Gauss-Seidel)

Fixpunktiteration für ein LGS.

Fixpunktiteration

$$x^{(n+1)} = Bx^{(n)} + c = F(x^{(n)})$$

Für Jacobi (Gesamtschrittverfahren)

$$B = -D^{-1}(L+R) \qquad c = D^{-1} \cdot b$$

Für Gauss-Seidel (Einzelschrittverfahren)

$$B = -(D+L)^{-1}R \qquad c = (D+L)^{-1} \cdot b$$

Konvergenz gegen $\bar{x}$

1. Diagonaldominante Matrix

Falls *A* **diagonaldominant** ist, **konvergiert** das **Jacobi** (Gesamtschrittverfahren) und auch das **Gauss-Seidel** (Einzelschrittverfahren) für *Ax* = *b*.

Es gibt nicht diagonaldominante Matrizen, für die die Verfahren dennoch konvergieren. Das notwendige Kriterium für Konvergenz ist, dass der Spektralradius $\rho(B) < 1$.

Diagonaldominant, falls **eines** der **beiden** Kriterien gilt:

- In jeder **Zeile oder Spalte** ist das Diagonalelement betragsmässig grösser als die Summe der Beträge aller anderen Elemente der **Zeile oder Spalte.**

Bsp:

$$A = \begin{bmatrix} 4 & -1 & 1 \\ -2 & 5 & 1 \\ 1 & -2 & 5 \end{bmatrix} \begin{matrix} \rightarrow |4| > |-1| + |1| \checkmark \\ \rightarrow |5| > |-2| + |1| \checkmark \\ \rightarrow |5| > |1| + |-2| \checkmark \end{matrix} \Big\} \text{ diagonaldominant}$$

Wenn Diagonaldominantz nicht zutrifft, muss **zusätzlich Fixpunktsatz** überprürt werden:

2. Fixpunktsatz

- $\bar{x}$ anziehender Fixpunkt, falls $\| B \| < 1$
  d.h. *F*(*x*) besitzt **genau einen** Fixpunktvektor $\bar{x}$.
- $\bar{x}$ abstossender Fixpunkt, falls $\| B \| > 1$
  d.h. *F*(*x*) besitzt **genau einen** Fixpunktvektor $\bar{x}$.

B ist Ableitungsmatrix von *F*.

Vorgehen:

1. Ist $\bar{x}$ ein Fixpunktvektor von *F*?
Es muss gelten: $B \cdot x + c = x$

2. Ist $\bar{x}$ anziehend oder abstossend?
Anziehend wenn $\| B \| < 1$, sonst abstossend.

Fehlerabschätzungen

Der Abstand von $x^{(k)}$ zu $x_{Fixpunkt}$ kann abgeschätzt werden durch:

(*n*) = n-te Iteration, *n* = Exponent

$$\| x^{(n)} - \bar{x} \| \leq \frac{\| B \|^n}{1 - \| B \|} \| x^{(1)} - x^{(0)} \| \quad \text{a-priori}$$

Basierend auf den **ersten** beiden Iterationen x_0 und x_1
Für (sehr grobe) **Abschätzung** im Voraus.

$$\| x^{(n)} - \bar{x} \| \leq \frac{\| B \|}{1 - \| B \|} \| x^{(n)} - x^{(n-1)} \| \quad \text{a-posteriori}$$

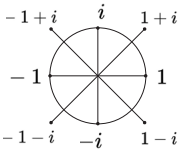
Basierend auf den **letzten** beiden Iterationen x_{n-1} und x_n
Zur **Überprüfung** im Nachhinein.

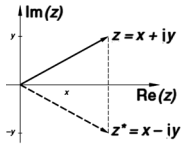
Mindestanzahl Iterationen *n*

$$n = \left\lceil \log_{\|B\|} \left(\frac{\varepsilon(1 - \|B\|)}{\|x^{(1)} - x^{(0)}\|} \right) \right\rceil$$

$$= \left\lceil \log \left(\frac{\varepsilon(1 - \|B\|)}{\|x^{(1)} - x^{(0)}\|} \right) \cdot \frac{1}{\log \|B\|} \right\rceil$$

Komplexe Zahlen $\mathbb{C}$





Realteil $\quad Re(z) = x$

Imaginärteil $\quad Im(z) = y$

$$-i = e^{-i\frac{\pi}{2}} = e^{i\frac{3}{2}\pi}$$
$$i = e^{i\frac{\pi}{2}}$$
$$-1 = e^{i\pi}$$
$$1 = e^{i0}$$

$$1+i = \sqrt{2}\cdot e^{i\frac{\pi}{4}}$$
$$1-i = \sqrt{2}\cdot e^{i\frac{7}{4}\pi} = \sqrt{2}\cdot e^{-i\frac{1}{4}\pi}$$
$$-1+i = \sqrt{2}\cdot e^{i\frac{3}{4}\pi}$$
$$-1-i = \sqrt{2}\cdot e^{i\frac{5}{4}\pi} = \sqrt{2}\cdot e^{-i\frac{3}{4}\pi}$$

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

$$i = \sqrt{-1}$$
$$i^2 = -1$$
$$i^3 = -\sqrt{-1}$$
$$i^4 = 1$$
$$i^5 = i$$
$$\dots$$

$x = \sqrt{-16} = \sqrt{16\cdot(-1)} = \sqrt{16}\cdot\sqrt{-1} = 4i$

$x^2 + 1 = 0 \Rightarrow x^2 = -1 \Rightarrow x = \sqrt{-1} = i$

Darstellungsformen

Normalform (x, y)

$z = x + yi \leftrightarrow P_z = (x, y)$

$z^* = x - yi$ **Konjugation**

Polarform (r, φ)

Trigonometrische Form

$z = r(\cos\varphi + \sin\varphi\cdot i) = r\cdot\cos\varphi + r\cdot\sin\varphi\cdot i$

Exponentialform

$z = r\cdot e^{i\varphi} = r(\cos\varphi + \sin\varphi\cdot i) \qquad e^{i\cdot\pi} = -1$

$z^n = (r\cdot e^{i\varphi})^n = r^n\cdot e^{in\varphi} = r^n(\cos(n\varphi) + \sin(n\varphi)\cdot i)$

Normalform (x, y) → Polarform (r, φ)

$z = (x + yi)$

$r = |z| = \sqrt{x^2+y^2}$

$$\varphi = \begin{cases} \tan^{-1}\left(\frac{y}{x}\right) & \text{für } x \geq 0 \\ \tan^{-1}\left(\frac{y}{x}\right) + \pi & \text{für } x < 0 \end{cases}$$

Polarform (r, φ) → Normalform (x, y)

$z = r\cdot e^{i\varphi} \qquad x = r\cdot\cos\varphi \qquad y = r\cdot\sin\varphi$

Grundrechenoperationen

Multiplizieren

Am einfachsten in **Exponentialform**

Multiplizieren heisst: Ihre **Winkel addieren** und ihre **Längen multiplizieren.**
Multiplizieren mit (0 + i) rotiert ½π.

$$z_1\cdot z_2 = (x_1 + y_1 i)\cdot(x_2 + y_2 i) = \begin{pmatrix} x_1 & -y_1 \\ y_1 & x_1 \end{pmatrix}\cdot\begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$$

$$= (x_1x_2 - y_1y_2) + (x_1y_2 + x_2y_1)\,i$$

$$z_1\cdot z_2 = r_1 e^{i\varphi_1}\cdot r_2 e^{i\varphi_2} = r_1 r_2\, e^{i(\varphi_1+\varphi_2)}$$

Addieren / Subtrahieren

Immer in der **Normalform**

$$z_1 \pm z_2 = (x_1 + y_1 i) \pm (x_2 + y_2 i) = (x_1 \pm x_2) + (y_1 \pm y_2)\,i$$

Dividieren

Am einfachsten in **Exponentialform**

$$\frac{z_1}{z_2} = \frac{z_1\cdot z_2^*}{z_2\cdot z_2^*} = \frac{(x_1 + y_1 i)\cdot(x_2 - y_2 i)}{(x_2 + y_2 i)\cdot(x_2 - y_2 i)}$$

$$= \frac{(x_1x_2 + y_1y_2)}{x_2^2 + y_2^2} + \frac{(x_2y_1 - x_1y_2)}{x_2^2 + y_2^2}\,i$$

$$\frac{z_1}{z_2} = \frac{r_1 e^{i\varphi_1}}{r_2 e^{i\varphi_2}} = \frac{r_1}{r_2}\,e^{i(\varphi_1-\varphi_2)}$$

Potenzieren

Immer in der **Exponentialform**

$$z^n = (r\cdot e^{i\varphi})^n = r^n\cdot e^{in\varphi} = r^n(\cos(n\varphi) + \sin(n\varphi)\cdot i)$$

$$z_1^n = z_2 \Rightarrow z_1 = \sqrt[n]{z_2} = z_2^{\frac{1}{n}}$$

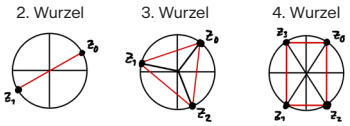
Radizieren (Wurzelziehen)

Eine komplexte Zahl mit Exponentialform $w = r\cdot e^{i\varphi}$ besitzt *n* Wurzeln.
n = Wurzelexponent, z_k = Wurzel von w, k = Laufindex

$$z_k = r_k\cdot e^{i\varphi k}$$

$$r_k = \sqrt[n]{r} \qquad \varphi_k = \frac{\varphi + k\cdot 2\pi}{n} \qquad k = 0, 1, 2, \dots, n-1$$

Die *n* Wurzeln $z_0, z_1, \dots, z_{n-1}$ bilden in der komplexen Zahlenebene die Ecken einer regelmässigen *n*-Ecks:



Polynomialgleichung

Ein **Polynom *n*-ten Grades mit komplexen Koeffizienten** besitzt **genau *n* Lösungen**. (Im Gegensatz ein Polynom mit reellen Koeffizienten besitzt höchstens *n* reelle Lösungen).

Ein Polynom vom Grad n lässt sich wie im Reellen in Linearfaktoren zerlegen mit den Nullstellen z_i. Bsp:

$$P(z) = z^3 - z^2 + 4z - 4 \quad \rightarrow \quad P(z) = (z-1)(z-2i)(z+2i)$$

Das Polynom hat also eine reelle Nullstelle $z_1 = 1$ und
zwei zueinander komplex konjugiert Nullstellen $z_2 = 2i$ und $z_3 = -2i$

$$p(z) = z^2 + \underbrace{(-4+i)}_{b}\cdot z + \underbrace{(5-5i)}_{c} \qquad a = (1 + 0i) = 1$$

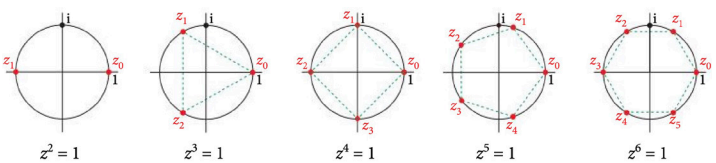
Mitternachtsformel

$$z_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Kreisteilungsgleichung (kein Modulstoff)

$z^n = 1$ hat in $\mathbb{C}$ genau *n* Lösungen.

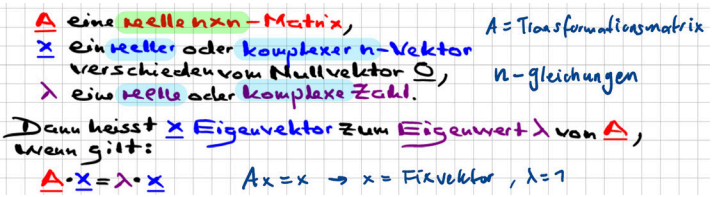
$$z_k = e^{ik\frac{2\pi}{n}} \qquad k = 0, 1, 2, \dots, n-1$$



Eigenwerte und Eigenvektoren

Definition $A \cdot x = \lambda \cdot x$

Vektor x (in R^2) liegt nach der Transformation auf der selben Linie.



Spur / trace (engl)

Die Summe der Eigenwerte ist gleich der Summe der Diagonalemente von A.

$$\mathrm{tr}\boldsymbol{A} = a_{11} + a_{22} + \ldots + a_{nn} = \lambda_1 + \lambda_2 + \ldots + \lambda_n$$

Charakteristischen Polynoms *p*(λ)

Zum **bestimmen von Eigenwerten** λ_i ➔ **1. Schritt**
Die **Eigenwerte** von A sind die **Nullstellen** des Polynoms.
A hat genau n Eigenwerte, von denen manche mehrfach vorliegen können.

$$\lambda \text{ ist Eigenwert von } \boldsymbol{A} \iff \det(\boldsymbol{A} - \lambda \boldsymbol{I}_n) = 0$$

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \Rightarrow \boldsymbol{A} - \lambda \boldsymbol{I}_2 = \begin{pmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{pmatrix}$$

$$p(\lambda) = \det(\boldsymbol{A} - \lambda \boldsymbol{I}_n) = (a_{11} - \lambda)(a_{22} - \lambda) - a_{12}a_{21}$$

Für eine **Dreiecksmatrix** gilt:

$$\boldsymbol{A} - \lambda \boldsymbol{I}_n = \begin{pmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} - \lambda \end{pmatrix}$$

$$p(\lambda) = \det(\boldsymbol{A} - \lambda \boldsymbol{I}_n) = (a_{11} - \lambda)(a_{22} - \lambda)\ldots(a_{nn} - \lambda)$$

Die Diagonalelemente einer Dreiecksmatrix ***A*** sind deren Eigenwerte.

Falls ***A*** eine **Diagonalmatrix** ist, dann sind die Diagonalelemente zusätzlich die Nullstellen des charakteristischen Polynoms = **Algebraische Vielfachheit**

Bsp. Eigenwerte λ_i bestimmen:

$$\boldsymbol{A} = \begin{pmatrix} 2 & 5 \\ -1 & -2 \end{pmatrix}$$

$$p(\lambda) = \det \begin{pmatrix} 2 - \lambda & 5 \\ -1 & -2 - \lambda \end{pmatrix} = (2 - \lambda)(-2 - \lambda) - 5 \cdot (-1)$$

$$= \lambda^2 + 1 = 0 \quad \Rightarrow \quad \lambda_{1,2} = \pm\sqrt{-1} = \pm i$$

Algebraische Vielfachheit von λ = 2

Eigenraum

Zum **bestimmen von Eigenvektoren** ➔ **2. Schritt**
= Lineare Hülle (Erzeugnis, Spann) der Eigenvektoren zu einem bestimmten Eigenwert.

Der Eigenraum des Eigenwertes λ ist die **Lösungsmenge** des homogenen linearen Gleichungssystems:

$$(\boldsymbol{A} - \lambda \boldsymbol{I}_n)\boldsymbol{x} = 0$$

Eine nicht-triviale (d.h. von 0 verschiedene) Lösung, wenn:

$$\mathrm{Rg}(\boldsymbol{A} - \lambda \boldsymbol{I}_n) < n$$

- **Rang** = Anzahl linear unabhängiger Zeilenvektoren (**führende Variablen**)
- Die **Dimension** eines Vektorraumes ist die **Anzahl** seiner **Basisvektoren**.
- **Basis**(-Vektoren) = linear unabhängige Vektoren eines Vektorraumes, mit denen sich jeder andere Vektor des Raumes eindeutig als Linearkombination darstellen lässt.

Bsp. Eigenvektor zu $\lambda_1 = +i$
Homogene Gleichungssystem lösen:

$$(\boldsymbol{A} - \lambda_1 \boldsymbol{I}_2)\boldsymbol{x} = \begin{pmatrix} 2 - i & 5 \\ -1 & -2 - i \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

bzw.

$$(\boldsymbol{A} - \lambda_1 \boldsymbol{I}_2 \mid \boldsymbol{0}) = \left(\begin{array}{cc|c} 2 - i & 5 & 0 \\ -1 & -2 - i & 0 \end{array}\right)$$

Varia

Die Eigenvektoren einer symmetrischen Matrix sind parallel (orthogonal) zueinander.

Geometrische Vielfachheit von λ

= Dimension des **Eigenraumes eines EW** λ.
= **Anzahl der linear unabhängigen Eigenvektoren** zum Eigenwert λ.

$$n - \mathrm{Rg}(\boldsymbol{A} - \lambda \boldsymbol{I}_n)$$

n = Anzahl Spalten/Zeilen einer n x n Matrix.
Rg = Anzahl Führende Variablen

Geometrische und algebraische Vielfachheit eines Eigenwerts müssen nicht gleich sein. Aber: **geom. Vielfachheit** $\leq$ **algeb. Vielfachheit**

Algebraische Vielfachheit von λ

Die Vielfachheit **(Anzahl gleiche Nullstellen)**, mit der λ **als Nullstelle** des charakteristischen Polynoms von *A* auftritt. Bsp:

$p(\lambda) = \lambda \cdot (\lambda - 3) \cdot (\lambda - 3)$

$\lambda = 0$ ist EW von *A* mit algebr. Vielfachheit 1
$\lambda = 3$ ist EW von *A* mit algebr. Vielfachheit 2 (doppelte Nullstelle)

Spektrum *σ(A)*

Menge aller Eigenwerte von *A*.

Eigenschaften

Hat man zwei Eigenvektoren x und y zum selben Eigenwert λ einer Matrix A, so ist **x + y** und auch **jedes Vielfache von x** ebenfalls ein **Eigenvektor** zum Eigenwert λ:

$$\begin{aligned} \boldsymbol{A}(\boldsymbol{x} + \boldsymbol{y}) &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{A}\boldsymbol{y} = \lambda\boldsymbol{x} + \lambda\boldsymbol{y} = \lambda(\boldsymbol{x} + \boldsymbol{y}) \\ \boldsymbol{A}(\mu x) &= \mu\boldsymbol{A}\boldsymbol{x} = \mu\lambda\boldsymbol{x} = \lambda\mu\boldsymbol{x} \end{aligned}$$

$$\det \boldsymbol{A} = \lambda_1 \cdot \lambda_2 \cdots \lambda_n$$

Für reelle quadratische Matrix A gilt:
- Zu jedem Eigenwert von A gibt es unendlich viele Eigenvektoren
- Der Eigenraum zu einem Eigenwert λ von A besteht genau aus allen Eigenvektoren zum Eigenwert λ und dem Nullvektor.

Spektralradius und Spektralnorm (2-Norm)

Spektralradius: Betragsmässig grösster Eigenwert von A.

$$\rho(\boldsymbol{A}) = \max\{|\ \lambda\ ||\ \lambda \text{ ist ein Eigenwert von } \boldsymbol{A} \in \mathbb{R}^{n \times n}\}$$

Spektralnorm (2-Norm):

$$\| \boldsymbol{A} \|_2 = \sqrt{\rho(\boldsymbol{A}^T\boldsymbol{A})}$$

Vektoriteration / von-Mises-Iteration (engl. Power Method)

Verfahren zum **berechnen des Spektralradius** und den zugehörigen **betragsmässig grössten Eigenvektor** einer **diagonalisierbaren Matrix A.**

Not as numerically stable as SVD (Principal Component Analysis).

$$\boldsymbol{v}^{(k+1)} = \frac{\boldsymbol{A}\boldsymbol{v}^{(k)}}{\| \boldsymbol{A}\boldsymbol{v}^{(k)} \|_2} \qquad \text{Eigenvektor}$$

$$\lambda^{(k+1)} = \frac{(\boldsymbol{v}^{(k)})^T \boldsymbol{A}\boldsymbol{v}^{(k)}}{(\boldsymbol{v}^{(k)})^T \boldsymbol{v}^{(k)}} \qquad \text{Eigenwert}$$

```
def vektoriteration(A, v, n, norm=2):
    for i in range(n):
        ew = v.T @ A @ v / (v.T @ v)
        v = A @ v / np.linalg.norm(A @ v, norm)
    return v, ew
```

Eigenwerte und Eigenvektoren bestimmen

1. Eigenwerte: $\det(\boldsymbol{A} - \lambda \boldsymbol{I}_n) = 0$
2. Eigenvektoren: $(\boldsymbol{A} - \lambda \boldsymbol{I}_n)\boldsymbol{x} = 0$

$$A = \begin{pmatrix} a & 1 & 0 \\ 1 & a & 2 \\ 0 & 2 & a \end{pmatrix} \qquad \det(A - \lambda I_n) = 0$$

$$\Rightarrow \begin{matrix} a-\lambda & 1 & 0 \\ 1 & a-\lambda & 2 \\ 0 & 2 & a-\lambda \end{matrix} \begin{matrix} a-\lambda & 1 \\ 1 & a-\lambda \\ 0 & 2 \end{matrix} \quad a-\lambda = (a-\lambda)^3 - 5(a-\lambda) = 0$$

$$\Rightarrow \underbrace{(a-\lambda)}_{=0 \text{ Fall 1}} \underbrace{\left((a-\lambda)^2 - 5\right)}_{=0 \text{ Fall 2}} = 0$$

Fall 1: $\lambda_1 = a$

Fall 2: $\underbrace{(a-\lambda)}_{\sqrt{5}}{}^2 - 5 = \underbrace{(a-\lambda)}_{\sqrt{5}}\underbrace{(a-\lambda)}_{\sqrt{5}} - 5 = 0$

$$\lambda_{2,3} = a \pm \sqrt{5}$$

$$\sigma(A) = \{a,\ a+\sqrt{5},\ a-\sqrt{5}\}$$

$$A = \begin{pmatrix} 2 & 5 \\ -1 & -2 \end{pmatrix}$$

1. EW: $\det(A-\lambda I) = 0$

$$(2-\lambda)(-2-\lambda) - (-1)5 = 0 \Rightarrow \lambda^2 + 1 = 0$$

$$\Rightarrow \lambda^2 = -1 \Rightarrow \lambda = \pm\sqrt{-1} \Rightarrow \lambda_{1,2} = \pm i$$

2. EV: $(A - \lambda I)x = 0$

- $\lambda_1 = -i$

$$\begin{pmatrix} 2+i & 5 \\ -1 & -2+i \end{pmatrix} \begin{matrix} 0 \\ 0 \end{matrix} \quad z_2 - \frac{z_{21}}{z_{11}} z_1 \quad \begin{pmatrix} 2+i & 5 \\ 0 & \underbrace{-2+i-5\frac{-1}{2+i}}_{0} \end{pmatrix} \begin{matrix} 0 \\ 0 \end{matrix}$$

$$= \begin{pmatrix} 2+i & 5 \\ 0 & 0 \end{pmatrix} \begin{matrix} 0 \\ 0 \end{matrix} \quad x_2 = a$$

$$x_1 = \frac{-5a}{2+i} = \frac{-5a}{2+i}\frac{(2-i)}{(2-i)}$$

$$= \frac{-10a + 5ai}{4 - 2i + 2i - \underbrace{i^2}_{-1}} = \frac{-10a + 5ai}{5} = a(-2+i)$$

$$z_1 = \underbrace{a\begin{pmatrix} -2+i \\ 1 \end{pmatrix}}_{\text{Eigenraum } \lambda_1 \text{ mit } a \in \mathbb{C}} \quad \text{EV zum EW } \lambda = -i$$

- $\lambda_2 = i$ (Ähnliches Verfahren)

$$z_2 = a\begin{pmatrix} -2-i \\ 1 \end{pmatrix} = z_1^* \quad \text{EV zum EW } \lambda = i$$

Geometrische Vielfachheit: $n - Rg(A-\lambda I) = 2 - 1 = 1$

Eigenraum $E_{\lambda_1} = \{z \mid z = a\begin{pmatrix} -2+i \\ 1 \end{pmatrix},\ a \in \mathbb{C}\}$

Numerische Berechnung von λ und Eigenvektoren

Diagonalisierbarkeit / Ähnliche Matrizen

$$B = T^{-1}AT, \quad A = TBT^{-1} = T^{-1}BT$$

B und *A* heissen zueinander ähnlich. *T* ist eine reguläre Matrix. Die Gleichung ist von Rechts nach Links zu lesen.

Ist ***B*** zusätzlich eine **Diagonalmatrix**, so ist ***A* diagonalisierbar,** d.h. *A* lässt sich mittels eines **Basiswechsels** (also der Konjugation mit einer regulären Matrix *T*) in eine Diagonalmatrix *B* transformieren, d.h. ***B*** ist ***A*** in der **neuen Basis. *B*** ist **eindeutig bestimmbar** (die Werte ändern sich nicht bei anderen *T*). *T* hingegen ist nicht eindeutig.

Eigenwerte und Eigenvektoren

1. B und A sind **zueinander ähnlich:**
   B und A haben **dieselben Eigenwerte**
   (inkl. gleiche algebraischen und geometrischen Vielfachheit)
   Ist B eine **Dreiecksmatrix:**
   Diagonalelemente von *B* sind **EW von *A***. (Siehe Matrix unten)
2. ***x*** ist EV zum EW λ von B → ***Tx*** ist EV zum EW λ von A
   $$Bx = \lambda x \quad \rightarrow \quad ATx = \lambda Tx$$
3. A ist **diagonalisierbar / *B*** ist eine **Diagonalmatrix:**
   - **Diagonalelemente von *B*** sind **EW von *A***.
   - Die **Spalten von *T*** sind linear unabhängige **EV von *A*** zu den **EW von *B*.**

$$\overset{B}{\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}} = \overset{T^{-1}}{\begin{pmatrix} 5 & 9 \\ 2 & 4 \end{pmatrix}} \cdot A \cdot \overset{T}{\begin{pmatrix} 2 & -4.5 \\ -1 & 2.5 \end{pmatrix}}$$

A ist diagonalisierbar (weil B eine Diagonalmatrix ist), somit:
Eigenwerte von A: $\lambda_1 = 1$, $\lambda_2 = 3$
Eigenvektoren von A: $v_1 = (2, -1)$ zum λ_1 und $v_2 = (-4.5, 2.5)$ zum λ_2

Matrix *T* bestimmen

$A = \begin{pmatrix} 5 & -2 \\ -2 & 2 \end{pmatrix}$ hat EW $\lambda_1 = 6$, $\lambda_2 = 1$ und zugehörige EV $x_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$, $x_2 = \begin{pmatrix} -1 \\ -2 \end{pmatrix}$.

A ist diagonalisierbar, somit gilt $D = T^{-1} \cdot A \cdot T$ mit

$$D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} = \begin{pmatrix} 6 & 0 \\ 0 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} x_1 & x_2 \end{pmatrix} = \begin{pmatrix} 2 & -1 \\ 1 & -2 \end{pmatrix}$$

```
D = np.linalg.inv(T) @ A @ T
```

Singuläre Matrix: det(A) = 0, non-invertible
Regulär Matrix: det(A) ≠ 0, invertable

Inverse Matrix A⁻¹

Eine Inverse **existiert nur** wenn $\det(A) \neq 0$
und nur für **quadratische** Matrizen.

$A \cdot A^{-1} = A^{-1} \cdot A = I_n$

Für 2×2-Matrizen

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

Für n × n-Matrizen

$$\left(\begin{array}{ccc|ccc} \frac{1}{2} & 0 & -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ -7 & 3 & 3 & 0 & 0 & 1 \end{array}\right) \rightarrow \text{Auf erweiterte ZSF bringen} \rightarrow \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & -\frac{6}{5} & -\frac{6}{5} & -\frac{2}{5} \\ 0 & 1 & 0 & -\frac{6}{5} & -\frac{11}{5} & -\frac{2}{5} \\ 0 & 0 & 1 & -\frac{8}{5} & -\frac{3}{5} & -\frac{1}{5} \end{array}\right)$$

$$A^{-1} = \begin{pmatrix} -\frac{6}{5} & -\frac{6}{5} & -\frac{2}{5} \\ -\frac{6}{5} & -\frac{11}{5} & -\frac{2}{5} \\ -\frac{8}{5} & -\frac{3}{5} & -\frac{1}{5} \end{pmatrix} = -\frac{1}{5} \begin{pmatrix} 6 & 6 & 2 \\ 6 & 11 & 2 \\ 8 & 3 & 1 \end{pmatrix}$$

Inverse einer Diagonalmatrix = Kehrwerte der Diagonalelemente

$$D = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}, \quad D^{-1} = \begin{pmatrix} \frac{1}{a} & 0 & 0 \\ 0 & \frac{1}{b} & 0 \\ 0 & 0 & \frac{1}{c} \end{pmatrix}$$

Eigenschaften einer Dreiecksmatrix (obere oder untere)
- Die Determinante ist das Produkt ihrer Hauptdiagonalelemente.
- Die Eigenwerte sind die Elemente der Hauptdiagonalen.

Determinante

Koeffizientenmatrix (Dreiecksmatrix)

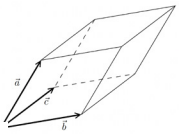
Determinante ist gleich dem **Produkt der Hauptdiagonalelemente**:

$$\begin{vmatrix} 4 & 8 & 0 & -2 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & 12 & 1 \\ 0 & 0 & 0 & -3 \end{vmatrix} = (-1)^z \cdot 4 \cdot 1 \cdot 12 \cdot (-3)$$

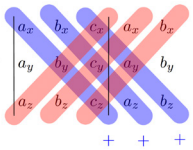
$det(A) = (-1)^z \cdot a_{11} \cdot a_{22} \cdot \ldots \cdot a_{nn}$ **z = Anzahl Zeilentausch**

3-reihige Determinante berechen

$[\vec{a} \quad \vec{b} \quad \vec{c}]$ (Spatprodukt / Regel von Sarrus)



Ber Betrag der Determinante einer 3×3 Matrix A (Spatprodukt) ist gleich dem **Volumen** des Spats.



$$= a_x b_y c_z + a_z b_x c_y + a_y b_z c_x - a_z b_y c_x - a_x b_z c_y - a_y b_x c_z$$
$$= a_x(b_y c_z - b_z c_y) + a_y(b_z c_x - b_x c_z) + a_z(b_x c_y - b_y c_x).$$

n-reihige Determinante

Vorzeichenfaktor wird nach der Schachbrettregel bestimmt

+	−	+	−
−	+	−	+
+	−	+	−
−	+	−	+

Bei jedem **Zeilen- oder Spaltenwechsel** ändert isch das **Vorzeichen** der Determinante.

Bsp:

3. Zeile - 4 (2. Zeile)

$$\begin{vmatrix} 2 & 1 & 3 & 0 \\ 4 & 0 & 2 & 1 \\ 1 & 3 & -1 & 4 \\ -1 & 3 & 2 & 0 \end{vmatrix} = \begin{vmatrix} 2 & 1 & 3 & 0 \\ 4 & 0 & 2 & 1 \\ -15 & 3 & -9 & 0 \\ -1 & 3 & 2 & 0 \end{vmatrix}$$

Entwicklung nach der 4. Spalte

$$= 1 \cdot \begin{vmatrix} 2 & 1 & 3 \\ -15 & 3 & -9 \\ -1 & 3 & 2 \end{vmatrix} = -3 \begin{vmatrix} 2 & 1 & 3 \\ 5 & -1 & 3 \\ -1 & 3 & 2 \end{vmatrix}$$

2. Zeile + 1. Zeile und
3. Zeile- 3(1. Zeile)

$$= -3 \begin{vmatrix} 2 & 1 & 3 \\ 7 & 0 & 6 \\ -7 & 0 & -7 \end{vmatrix}$$

Entwicklung nach der 2. Spalte

$$= -(-3) \begin{vmatrix} 7 & 6 \\ -7 & -7 \end{vmatrix}$$
$$= 3 \cdot (-7) \begin{vmatrix} 7 & 6 \\ 1 & 1 \end{vmatrix} = -21(7 - 6) = -21$$

$$A = \begin{pmatrix} 1 & 2 & -3 & 5 \\ 0 & 12 & 0 & 1 \\ 1 & 0 & -1 & 2 \\ -1 & 2 & 2 & 1 \end{pmatrix}$$

$$\det(A) = 12 \cdot \det(A_{22}) + \det(A_{24})$$
$$= 12 \begin{vmatrix} 1 & -3 & 5 \\ 1 & -1 & 2 \\ -1 & 2 & 1 \end{vmatrix} + \begin{vmatrix} 1 & 2 & -3 \\ 1 & 0 & -1 \\ -1 & 2 & 2 \end{vmatrix}$$

QR-Verfahren

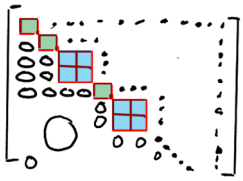
Numerische **Berechnung des betragsmässig grössten EW (dominater EW)** und **zugehörigen EV.** Ziel: Rechtsobere Dreiecksmatrix *A*.
Diagonalelemente von A(k) sind dann genau die **EW von A.**

$$A_{i+1} = R_i \cdot Q_i \qquad \text{wobei} \quad A_i = Q_i \cdot R_i$$

```
def qr_eigen(A, n):
    for i in range(n):
        Q, R = np.linalg.qr(A)
        A = R @ Q
    return A
```

Initialisierung	Schritt	Ausserdem gilt:
$A^{(0)} = A$	$A^{(k+1)} = R^{(k)} \cdot Q^{(k)}$	$A^{(k)} = Q^{(k)} \cdot R^{(k)}$
$P^{(0)} = I_n$	$P^{(k+1)} = P^{(k)} \cdot Q^{(k)}$	$A^{(k+1)} = (Q^{(k)})^{-1} \cdot A^{(k)} \cdot Q^{(k)}$
		$Q^{-1} = Q^T$

$A^{(k)}$ konvergiert gegen eine reelle Matrix.

$A^{(\infty)} =$ 

Reelle Eigenwerte
Komplexe Eigenwerte

«Perfekte» obere Dreiecksmatrix:
Wenn **alle EW** von A **betragsmässig verschieden** sind → keine 2 x 2 Blöcke.
Bsp. Verfahren liefert keine Dreiecksmatrix:

$$A = \left(\begin{array}{cc|c} -0.1072 & -2.5543 & -0.6586 \\ 2.3955 & 1.1072 & -0.2575 \\ \hline 0 & 0 & 1 \end{array}\right)$$

$$p(\lambda) = (-0.1072 - \lambda)(1.1072 - \lambda) - (-2.5543)(2.3955)$$
$$\approx \lambda^2 - 1\lambda + 6$$

$$\lambda_{1,2} = 0.5 \pm 2.3979i \qquad \lambda_3 = a_{33} = 1$$

Wenn **A eine symmetrische Matrix (Diagonalmatrix)** ist ($A^T = A$),
dann sind alle EW reell und die **Spalten der Matrix P zugehörige EV** von A der Länge 1. Matrix *P* ist normiert.