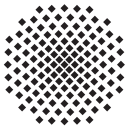
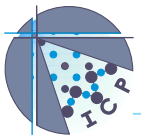


Universität Stuttgart



INSTITUTE FOR
COMPUTATIONAL
PHYSICS

Physik auf dem Computer

Mitschriften zur Vorlesung
Sommersemester 2017

Universität Stuttgart
Fakultät 8: Fachbereich Physik
B. Sc. Physik

31. Mai 2017

Dr. Jens Smiatek
Institut für Computerphysik
Universität Stuttgart
Allmandring 3
D-70569 Stuttgart
Germany

Email: smiatek@icp.uni-stuttgart.de

Inhaltsverzeichnis

Weiterführende Literatur

- Bronstein, Ilja N., et al. *Taschenbuch der Mathematik.* Eds. Eberhard Zeidler und Wolfgang Hackbusch. Springer-Verlag, Berlin 2012.
- Faires, J. Douglas und Burden, Richard L. *Numerische Methoden.* Spektrum Akademischer Verlag, Heidelberg 1994.
- Press, William H., et al. *Numerical Recipes.* Cambridge University Press, Cambridge 1989.

Allgemeine Hinweise zur Notation

Kurznotationen:

- $i = 1(1)n$ steht für $i = 1, 2, \ldots, n$
- $i = n(-a)1$ steht für $i = n, n - a, \ldots, 1 + a, 1$

1 Motivation: Physik auf dem Computer

1.1 Warum Physik auf dem Computer?

Numerische Methoden werden in vielen Bereichen der modernen Physik eingesetzt. Als Paradebeispiel hierfür gilt die neue *Dreieinigkeit der Physik (New Trinity of Physics)* (Abb. 1.1). Neben den klassischen Disziplinen Theorie und Experiment wird seit Mitte

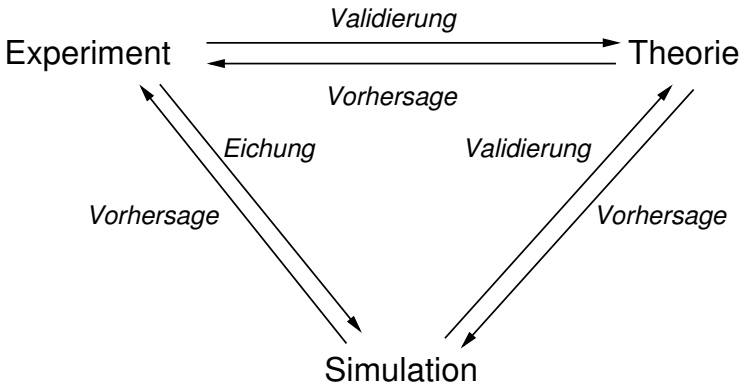


Abbildung 1.1: Die neue Dreieinigkeit der Physik mit den Säulen Experiment, Theorie und Simulation.

des letzten Jahrhunders auch der Simulationsansatz als dritte Säule in der modernen Physik angesehen. Theorien können nun durch Experiment **UND** Simulation verifiziert werden! Dies ist insbesondere wichtig für Wissenschaftstheorien wie dem Kritischen Rationalismus [1], welcher durch Karl R. Popper (1902 - 1994) begründet wurde und als wesentliche Kernaussage darauf abzielt, dass eine Theorie falsifizierbar sein muss, da sie ansonsten keine Theorie darstellt.

Einsatz von Rechnern und numerischen Methoden in der Theorie:

- numerische Lösungen von Gleichungen
- symbolische Mathematik (Mathematica©, ...)
- ...

Einsatz von Rechnern und numerischen Methoden im Experiment:

- Steuerung von Experimenten
- Auswertung von Daten
- ...

Fazit:
Computer und numerische Methoden werden in der modernen Physik in nahezu allen Bereichen eingesetzt.

1.2 Anwendung einer numerischen Methode zur Lösung der Bewegungsgleichung eines Fadenpendels (harmonischer Oszillator)

Im Folgendem soll nun ein einfaches Beispiel zur numerischen Lösung einer Bewegungsgleichung vorgestellt werden. Als Modellsystem dient das Fadenpendel (harmonischer

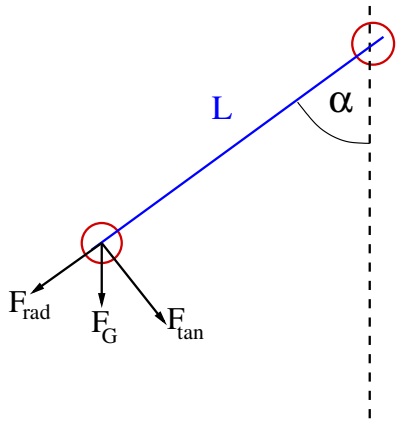


Abbildung 1.2: Schematische Darstellung eines Fadenependels.

Oszillator) in Abb. 1.2.

Annahmen:

- Steifer und masseloser Faden mit Längle L
- Kugel mit Masse m
- Es exisitert keine Reibung im System (keine Energiedissipation)

Vorherrschende Kräfte:

- radiale Komponente $\mathrm{F}_{\mathrm{rad}}$ (spielt keine Rolle für Bewegungsgleichung)
- Gravitation $\mathrm{F}_{\mathrm{G}} = m \cdot g$ mit Erdbeschleunigung $g = 9.81\ \mathrm{m/s^2}$
- tangentiale Kraft $\mathrm{F}_{\mathrm{tan}} = m \cdot g \sin\alpha = m \cdot a$

Mittels Newton lässt sich die Rückstellkraft berechnen,

$$\mathrm{F}_R = -\mathrm{F}_{\mathrm{tan}} = -m \cdot g \sin\alpha \tag{1.1}$$

welches

$$-\, m \cdot g \sin\alpha = m \cdot a \tag{1.2}$$

ergibt, wobei die Beschleunigung durch

$$a = -g \sin\alpha \tag{1.3}$$

gegeben ist. Die Tangentialbeschleunigung lässt sich auch mittels der Winkelbeschleunigung ausdrücken,

$$a = L\ddot{\alpha} \tag{1.4}$$

wodurch sich die allgemeine Bewegungsgleichung ergibt.

Allgemeine Bewegungsgleichung:

$$\ddot{\alpha}(t) = -\frac{g}{L} \sin\alpha(t) = -\omega^2 \sin\alpha(t) \tag{1.5}$$

1.2.1 Analytische Lösung mittels Näherung von kleinen Auslenkungen

Eine analytische Lösung für $\alpha(t)$ in Gleichung (1.5) kann für kleine Auslenkungen gefunden werden. Dann gilt $\sin\alpha \approx \alpha$ und Gleichung (1.5) vereinfacht sich zu

$$\ddot{\alpha}(t) = -\frac{g}{L}\alpha(t) \tag{1.6}$$

wobei eine Lösung von Gleichung (1.6) die folgende Grundform

$$\alpha(t) = A\cos\omega t + B\sin\omega t \tag{1.7}$$

mit Eigenfrequenz

$$\omega = \sqrt{\frac{g}{L}} \tag{1.8}$$

darstellt. Die Vorfaktoren A und B können nun durch

1. Ruhelage: Auslenkung $\alpha(t_0) = 0$ bei $t_0 = 0 \rightarrow A = \alpha(t_0)$
2. Geschwindigkeit: $\dot{\alpha}(t_0) = -\omega A \sin\omega t_0 + \omega B \cos\omega t_0 \rightarrow B = \frac{\dot{\alpha}(t_0)}{\omega}$

bestimmt werden. Nach Einsetzen von A und B in Gleichung (1.7) ergibt sich

$$\alpha(t) = \alpha(t_0)\cos\omega t + \frac{\dot{\alpha}(t_0)}{\omega}\sin\omega t \tag{1.9}$$

als allgemeine Lösung der Bewegungsgleichung (1.6).

1.2.2 Numerische Lösung mittels einer Simulation

Wie bereits angemerkt wurde, ist Gleichung (1.9) für kleine Auslenkungen gültig. Wie löst man aber die volle Bewegungsgleichung (1.5)? Hierfür ist keine einfache Näherung möglich und keine einfache analytische Lösung gegeben.
Als möglicher Ausweg kann eine **numerische Lösung** mittels einer Simulation gefunden werden.
Um die Simulation numerisch stabil zu halten, ist es sinnvoll, Parameterwerte für g und L innerhalb der gleichen Grössenordnung festzulegen. Da in obigem Beispiel die Erdbeschleunigung mit $g = 9.81\ \mathrm{m/s^2}$ festgelegt ist, kann $L = 1\ \mathrm{m}$ gewählt werden, so dass sich die Eigenfrequenz $\omega = \sqrt{g/L} \approx 3.13\ \mathrm{s^{-1}}$ ergibt. Dies sollte die numerische Stabilität der Simulation gewährleisten. Allerdings tritt noch ein weiterer wichtiger Punkt bei Simulationen auf.

Kernproblem von Simulationen:
Es existiert kein kontinuierlicher Zeitverlauf bei Simulationen und eine Diskretisierung der Zeit in diskrete Zeitintervalle ist unabdingbar.

Die Abfolge der Zeitintervalle ist durch

$$t_n = n \cdot \delta t \tag{1.10}$$

gegeben, wobei $n = 0, \ldots, N$ und δt die Breite des Zeitintervalls angibt. Das Ziel der Simulation ist nun, die allgemeine Bewegungsgleichung (1.5) numerisch zu lösen und als Funktion von $\alpha(t_n)$ darzustellen. Als erster Schritt für die Anwendung einer Simulation bedarf es einer willkürlichen, aber vernünftigen Wahl der Ausgangsposition und der Ausgangsgeschwindigkeit in Gleichung (1.5).
Hierzu werden

$$\dot{\alpha}(t_0) = v(t_0) \tag{1.11}$$

als Ausgangsgeschwindigkeit und $\alpha(t_0)$ als Ausgangsposition angenommen.
Als allgemeinen Ansatz wird eine Integration von Gleichung (1.5) durchgeführt, welche

$$\dot{\alpha}(t + \delta t) \equiv v(t_0 + \delta t) = v(t_0) + \int_{t_0}^{t_0+\delta t} [-\omega^2 \sin \alpha(\tau)]\, d\tau \tag{1.12}$$

ergibt. Die analytische Auswertung des Integrals ist jedoch in der Simulation nicht möglich, da nur die Stütz- und Auswertestellen bei ganzzahligen Vielfachen von δt bekannt sind oder berechnet werden können. Um dies zu ermöglichen und um die obige Gleichung numerisch handhabbar zu machen, wird folgende Näherung eingesetzt

$$f'(t)\, dt \approx f'(t) \lim_{\delta t \to 0} \delta t \tag{1.13}$$

welche zu

$$v(t_0 + \delta t) \approx v(t_0) - \omega^2 \sin \alpha(t_0)\, \delta t \tag{1.14}$$

führt, wobei $v(t_0 + \delta t)$ natürlich umso exakter ist, je kleiner der Wert für δt gewählt wurde. Allerdings gibt es bei physikalischen Simulationen auch eine untere Schranke, bei der die Gesetze der Quantenmechanik gelten und der Begriff von klassischen Trajektorien nicht mehr gültig ist.
Um die neue Position zu berechnen, wird dann nochmals "integriert", welches

$$\alpha(t_0 + \delta t) \approx \alpha(t_0) + v(t_0)\,\delta t \tag{1.15}$$

ergibt. Durch sukzessive und iterative Abfolge der Gleichungen (1.14) und (1.15) kann nun die Trajektorie des System näherungsweise berechnet werden.
Um die Qualität der numerischen Lösung einschätzen zu können, bedarf es der Überprüfung einer unabhängigen ud wohlbekannten Kenngrösse des Systems. Eine solche Kenngrösse stellt die Gesamtenergie dar, welche durch kinetische und potentielle Anteile festgelegt ist und welche per Definition konstant ist (s. Abschnitt 1.2).
Weitere Beispiele für den Einsatz von numerische Lösungen von Bewegungsgleichungen sind z. B. auch

- die Berechnung der Flugbahn eines Balls oder einer Rakete
- die Bewegung von Planeten im Sonnensystem
- die Kollision von harten Kugeln

Um die Bewegungsgleichungen noch realitätsnäher zu gestalten, können auch weitere Effekte wie Reibungskräfte in die Bewegungsgleichungen aufgenommen werden. Ebenso gibt es auch bessere Simulationsalgorithmen als die oben vorgestellte Variante. Ein vielfach in Simulationen eingesetzter Algorithmus ist der Velocity Verlet Algorithmus [3, 2] oder Varianten davon, welche folgende Grundabfolge aufzeigen:
1. Schritt: Berechnung der Geschwindigkeiten $v(t + \delta t/2)$

$$v(t + \delta t/2) = v(t) + \frac{\delta t}{2m} F(t) \tag{1.16}$$

2. Schritt: Berechnung der neuen Positionen $\alpha(t + \delta t)$

$$\alpha(t + \delta t) = \alpha(t) + \delta t\, v(t + \delta t/2) \tag{1.17}$$

3. Schritt: Berechnung der neuen Geschwindigkeiten $v(t + \delta t)$

$$v(t + \delta t) = v(t + \delta t/2) + \frac{\delta t}{2m} F(t + \delta t) \tag{1.18}$$

und nach Schritt 3 der Algorithmus wieder bei Schritt 1 startet.
Trotz der universellen Anwendbarkeit von numerischen Lösungen gibt es immer aber einen Punkt zu beachten.

2 Lineare Algebra: Lineare Gleichungssysteme

Eines der Grundlagen und der Hauptprobleme der numerischen Mathematik

- lineare Gleichungssysteme

Lineare Gleichungssysteme werden benutzt zum numerischen Lösen von Differentialgleichungen

- Rückführung auf Satz von linearen Gleichungssystemen

Durch die vorhandene Rechnerarchitektur ist es heutzutage möglich auch ein System von vielen Millionen Variablen zu lösen. In diesem Kapitel geht es hauptsächlich um grundlegende Methoden zum Lösen von linearen Gleichungssystemen wie z. B. die Gaußelimination, die LR- als auch die Choleskyzerlegung.

2.1 Lineare Gleichungssysteme

Bei vielen praktischen Aufgaben werden für n unbekannte Größen x_i (mit $i = 1, \ , 2, \ldots n$) m Bedingungen in Gleichungsform gestellt:

$$\begin{aligned} F_1(x_1, x_2, \ldots x_n) &= 0 \\ F_2(x_1, x_2, \ldots x_n) &= 0 \\ &\vdots \\ F_m(x_1, x_2, \ldots x_n) &= 0 \end{aligned} \tag{2.1}$$

Aufgabe:
Die Unbekannten x_i sind so zu bestimmen, dass sie eine Lösung des Gleichungssystems (2.1) darstellen.
Anmerkung:
In der Regel gilt $m = n$, d. h. die Anzahl der Unbekannten stimmt mit Anzahl der Gleichungen überein
Aber auch:

- $m > n$: überbestimmtes System, mehr Gleichungen als Unbekannte
- $m < n$: unterbestimmes System, weniger Gleichungen als Unbekannte

Überbestimmte Systeme:
In der Regel existiert keine Lösung!
Ersatz: Quadratmittelaufgabe:

$$\sum_{i=1}^{m} F_i^2(x_1, x_2, \dots x_n) = \min$$

Unterbestimmte Systeme:
$n - m$ Unbekannte können frei gewählt werden, d. h. die Lösung von Glg. (2.1) hängt von $n - m$ Parametern ab: $(n - m)$ - *dimensionale Lösungsmannigfaltigkeit*

Anmerkung: Im Gegensatz zu linearen Gleichungssystemen können in nicht-linearen Gleichungssystemen auch nicht-lineare Unbekannte auftreten ($\rightarrow$ Lösung typischerweise durch Iterationsverfahren)

2.1.1 Allgemeine Matrixschreibweise eines linearen Gleichungssystems

Gegeben sei das lineare Gleichungssystem für $m = n$

$$\begin{pmatrix} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \dots & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n & = & b_n \end{pmatrix} \tag{2.2}$$

mit der Bedingung $A = (a_{ik}) \in \mathbb{R}^{n \times n}$ für $i, k = 1, 2, \dots, n$ oder

$$\overleftrightarrow{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

mit $b \in \mathbb{R}^n$ und $x \in \mathbb{R}^n$. Das obige Gleichungssystem (Glg. (2.2)) kann auch in der kompakten Form

Allgemeine Matrizengleichung

$$\overleftrightarrow{A} \cdot \vec{x} = \vec{b} \tag{2.3}$$

oder schlicht $A \cdot x = b$ geschrieben werden. Im Allgemeinen wird nach den Lösungen $x \in \mathbb{R}^n$ gesucht, wobei nicht klar ist, ob es eine Lösung gibt oder ob diese eindeutig ist.

2.2 Dreiecksmatrizen

Im nun folgenden Abschnitt beschäftigen wir uns mit Dreiecksmatrizen. Unter einer Dreiecksmatrix versteht man eine quadratische Matrix ($m = n$), welche sich dadurch

auszeichnet, dass alle Eintärge unterhalb (obere Dreiecksmatrix) bzw. oberhalb (untere Dreiecksmatrix) der Hauptdiagonalen null sind.

Rechte obere Dreiecksmatrix R (im Englischen U)
Quadratische Matrix $A \in \mathbb{R}^{n \times n}$ mit $a_{ij} = 0$ für $i > j$.

Linke untere Dreiecksmatrix L (im Englischen L)
Quadratische Matrix $A \in \mathbb{R}^{n \times n}$ mit $a_{ij} = 0$ für $i < j$.

Dreiecksmatrizen werden zum Lösen von linearen Gleichungssystemen benötigt, wie wir in den späteren Abschnitten noch sehen werden.

2.3 Dreieckszerlegung einer Matrix

Durch elementare Umformungen wie dem Vertauschen von Zeilen oder Spalten, der Multiplikation einer Zeile mit einer von 0 verschiedenen Zahl oder der Addition eines Vielfachen einer Zeile zu einer anderen Zeile kann das System $\overleftrightarrow{A} \cdot \vec{x} = \vec{b}$ in ein gestaffeltes Gleichungssystem $\overleftrightarrow{R} \cdot \vec{x} = \vec{c}$ mit

$$\overleftrightarrow{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1n} \\ & r_{22} & r_{23} & \dots & r_{2n} \\ & & r_{33} & & r_{3n} \\ & \varnothing & & \ddots & \vdots \\ & & & & r_{nn} \end{pmatrix}$$

überführt werden, welches eine Lösung besitzt, falls das Gleichungssystem (2.2) **regulär** mittels

$$\det A \neq 0 \tag{2.4}$$

ist. Die Lösung kann dann eindeutig durch **Rückwärtssubstitution**

Rückwärtssubstitutionsgleichung

$$x_i = \frac{1}{r_{ii}} \left(c_i - \sum_{k=i+1}^{n} r_{ik} x_k \right) \tag{2.5}$$

mit $i = n-1, n-2, \dots, 1$ und $x_n = \frac{c_n}{r_{nn}}$

bestimmt werden.

Für reguläre linke untere Dreiecksmatrizen kann die **Vorwärtssubstitution** angewendet werden. Entsprechend $\overleftrightarrow{A} \cdot \vec{x} = \vec{b} \rightarrow \overleftrightarrow{L} \cdot \vec{x} = \vec{c}$ mit

$$\overleftrightarrow{L} = \begin{pmatrix} \ell_{11} & & & & \\ \ell_{21} & \ell_{22} & & \varnothing & \\ \ell_{31} & l_{32} & \ell_{33} & & \\ \vdots & \vdots & & \ddots & \\ \ell_{n1} & l_{n2} & \dots & \ell_{n,n-1} & \ell_{nn} \end{pmatrix}$$

gilt

Vorwärtssubstitutionsgleichung

$$x_i = \frac{1}{\ell_{ii}} \left(c_i - \sum_{k=1}^{i-1} \ell_{ik} x_k \right) \tag{2.6}$$

mit $i = 2, 3, \dots, n$ und $x_1 = \frac{c_1}{\ell_{11}}$

Des Weiteren gilt für Diagonalmatrizen

Diagonalmatrizensubstitution

$$x_i = \frac{c_i}{r_{ii}} \text{ bzw. } x_i = \frac{c_i}{\ell_{ii}} \text{ mit } i = 1, 2, \dots, n$$

Löseroutine in Python (SciPy):

- scipy.linalg.solve_triangular (...)

Frage: Wie gelingt nun aber die Transformation von $\overleftrightarrow{A}$ zu $\overleftrightarrow{R}$?

2.3.1 Prinzip des Gaußschen Eliminationsverfahrens

Das Gaußsche Eliminationsverfahren ist ein Verfahren, um den Übergang von $\overleftrightarrow{A}$ zu $\overleftrightarrow{R}$ in $n-1$ Eliminationsschritten durchzuführen.
Hauptprinzip:
Mithilfe einer Gleichung soll eine Unbekannte aus den restlichen Gleichungen entfernt werden $\rightarrow m-1$ Gleichungen mit $n-1$ Unbekannten (1. Schritt) usw.
Elementaroperationen:
Für das System (Glg. (2.3)) $\overleftrightarrow{A} \cdot \vec{x} = \vec{b}$ (im Folgenden als Koeffizientenmatrix $(A|b)$ dargestellt), können folgende Operationen vorgenommen werden:

Elementaroperationen

1. Vertauschung zweier Gleichungen (Zeilentausch in $(A|b)$)
2. Vertauschung zweier Spalten in $\vec{x}$ und $\overleftrightarrow{A}$ (Variablentausch)
3. Addition eines Vielfachen einer Zeile zu einer anderen Zeile
4. Multiplikation einer Zeile mit einer Konstanten $d \neq 0$

Ziel der Elementaroperationen:
Bringe Matrix $(A|b)$

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right)$$

spaltenweise auf Dreiecksform.
1. Schritt: Es sei $a_{11} \neq 0$ (Pivotelement, vgl. Abschnitt 2.3.3), ansonsten Vertauschung der Gleichungen
2. Schritt: Überführung der Matrix $\overleftrightarrow{A}$ nach Matrix $\overline{A}$; Vertausche die 1. und die r-te Zeile von $\overleftrightarrow{A}$
3. Schritt: Überführung der Matrix $\overline{A}$ nach $\overleftrightarrow{A}_1$ $((\overline{A}|b) \to (A_1|b_1)$; Subtrahiere für $i = 2, 3, \dots n$ das ℓ_{i1}-fache der 1. Zeile von der i-ten Zeile der Matrix $\overline{A}$

Ergebnis:

$$\overleftrightarrow{A}_1 = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \vdots & a_{32}^{(1)} & \dots & a_{3n}^{(1)} \\ \vdots & \vdots & & \\ 0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{pmatrix} \qquad \vec{b}_1 = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ \vdots \\ b_n^{(1)} \end{pmatrix}$$

Farbige Teilmatrix ist vom Typ $(n-1,\ n-1)$ und wird analog zu $\overleftrightarrow{A}$ im nächsten Schritt zu $\overleftrightarrow{A}_2$ überführt.
Die Elemente in $\overleftrightarrow{A}_1$ und $\vec{b}_1$ lassen sich mit:

$$a_{ik}^{(1)} = \overline{a}_{ik} - \ell_{i1} \cdot \overline{a}_{1k} \text{ mit } \ell_{i1} = \frac{\overline{a}_{i1}}{\overline{a}_{11}}$$

$$b_i^{(1)} = \overline{b}_i - \ell_{i1} \cdot \overline{b}_1$$

für $i, k = 2, 3, \dots n$ berechnen.
Durch sukzessives Wiederholen kann schlüssendlich die gewünschte Form $\overleftrightarrow{R} \cdot \vec{x} = \vec{c}$ ermittelt werden, d. h. das Verfahren wird beendet, wenn alle $a_{ik}^{(r-1)} = 0$ für $i, k \geq r$ sind. Dann gilt $(A_{r-1}|b_{r-1}) = (R|c)$.

Lösungsverhalten:
1. Fall: Das System ist unlösbar für $a_{ik}^{(r-1)} \neq 0$ für $i, k \geq r$.
2. Fall: Das System ist lösbar für $a_{ik}^{(r-1)} = 0$ für $i, k \geq r$.
Im Speziellen gilt dann:

i) $r = n$: Die Lösung ist eindeutig.

ii) $r < n$: Die Lösung ist nicht eindeutig, $n - r$ Unbekannte sind frei wählbar.

2.3.2 Ein Beispiel zur Gauß-Elimination

Gegeben sei folgendes Gleichungssystem

$$\begin{aligned} 5x_1 - x_2 + 2x_3 &= 3 \\ 7x_2 + x_3 &= 4 \\ 10x_1 + x_2 + x_3 &= 1 \end{aligned}$$

welches folgende Koeffizientenmatrix liefert:

$$\left(\begin{array}{ccc|c} 5 & -1 & 2 & 3 \\ 0 & 7 & 1 & 4 \\ 10 & 1 & 1 & 1 \end{array}\right) \begin{matrix} \mathrm{E}_1 \\ \mathrm{E}_2 \\ \mathrm{E}_3 \end{matrix}$$

mit dem kanonischen Pivotelement $a_{11} = 5$ (vgl. Abschnitt 2.3.3). Die jeweiligen E_1, E_2 und E_3 kennzeichnen die jeweiligen Zeilen in der Koeffizientenmatrix, wobei die veränderten Zeilen nach p Elementaroperationen mit $\mathrm{E}_i^{(p)}$ für $i = 1, 2, 3$ notiert werden.

1. Schritt: Vertausche E_3 mit E_2:

$$\begin{pmatrix} \mathrm{E}_1 \\ \mathrm{E}_2 \\ \mathrm{E}_3 \end{pmatrix} \rightarrow \begin{pmatrix} \mathrm{E}_1 \\ \mathrm{E}_3 \\ \mathrm{E}_2 \end{pmatrix} \equiv \begin{pmatrix} \mathrm{E}'_1 \\ \mathrm{E}'_2 \\ \mathrm{E}'_3 \end{pmatrix}$$

2. Schritt: Multipliziere E'_1 mit (-2) und addiere zu E'_2:

$$\left(\begin{array}{ccc|c} 5 & -1 & 2 & 3 \\ 0 & 3 & -3 & -5 \\ 0 & 7 & 1 & 4 \end{array}\right) \begin{matrix} \mathrm{E}''_1 \\ \mathrm{E}''_2 \\ \mathrm{E}''_3 \end{matrix}$$

3. Schritt: Multipliziere E_2'' mit $\left(-\frac{7}{3}\right)$ und addiere zu E_3'':

$$\left(\begin{array}{ccc|c} 5 & -1 & 2 & 3 \\ 0 & 3 & -3 & -5 \\ 0 & 0 & 8 & \frac{47}{3} \end{array}\right)$$

Obige Matrix hat die gewünschte rechte obere Dreiecksform und kann nun eindeutig durch Rückwärtssubstitution (Glg. (2.5)) oder durch Einsetzen der Werte gelöst werden.

Lösung durch Einsetzen:

i) $8x_3 = \frac{47}{3} \rightarrow x_3 = \frac{47}{24}$

ii) $3x_2 - 3 \cdot \frac{47}{24} = -5 \rightarrow x_2 = \frac{7}{24}$

iii) $5x_1 - \frac{7}{24} + 2 \cdot \frac{47}{24} = 3 \rightarrow x_1 = -\frac{1}{8}$

Lösung durch Rückwartssubstitution (Glg. (2.5)):

i) $x_3 = \frac{\frac{47}{3}}{8}$ mit $i = n = 3$.

ii) $x_2 = \frac{1}{r_{22}}(c_2 - r_{23} \cdot x_3) = \frac{1}{3}\left(-5 + 3 \cdot \frac{47}{24}\right) = \frac{7}{24}$ mit $i = n - 1$.

iii) $x_1 = \ldots$

2.3.3 Zur Bestimmung des Pivotelements

Das bereits erwähnte Element a_{r1} heißt Pivotelement (franz./engl. "Drehpunkt"), da es den
Dreh- und Angelpunkt des iterativen Verfahrens der Gaussös-Elimination darstellt.
$\Rightarrow$ Numerisch günstig für Wahl des Pivotelements: $a_{r1} \gg 1$

Wahlmöglichkeiten für Pivotelement

1. **Kanonische Pivotwahl:** keine Vertauschungen!
   Einsatz nur bei Sicherstellung $a_{r1} \gg 1$
   Anmerkung: Verfahren scheitert schon bei $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
2. **Spaltenpivotwahl:** Pivotelement ist das betragsmäßig größte Element in der Spalte
   (Vertauschung der Zeilen, aber Variablenreihenfolge bleibt unverändert.
3. **Totalpivotwahl:** Bestimme als Pivotelement das betragsgrößte Element in der Restmatrix
   (Spalten- und Zeilentausch nötig!)
   Abfolge:
   a) Vertauschung der Zeilen
   b) Vertauschung der Spalten

Anmerkungen zur Gauß-Elimination:
Benötigte Anzahl von Rechenoperationen bei Gauß-Elimination: $O(n^3)$
In Python: Keine Gauß-Elimination (zu langsam)
Stattdessen in Python für eine eindeutig lösbare Matrix: scipy.linalg.solve()
Basiert auf LR-Zerlegung (vgl. Abschnitt 2.3.6)

2.3.4 Matrixinversion

Es sei $\overleftrightarrow{A} \in \mathbb{R}^{n \times n}$ regulär , d.h. eine quadratische Matrix mit existierender Inverser $\overleftrightarrow{A}^{-1}$.
Es gilt weiterhin das algemeine Gleichungssystem (Glg. (2.3)) $\overleftrightarrow{A} \cdot \vec{x} = \vec{b}$.
Man kann dann zeigen, dass zu einer regulären Matrix $\overleftrightarrow{A}$ immer eine inverse Matrix $\overleftrightarrow{A}^{-1}$ existiert mit

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

wobei $\overleftrightarrow{E}$ die Einheitsmatrix ist.
Anmerkung zum Abschnitt 2.3.2:
Rücksubstitutionsmethode (Glg. (2.3.1)) liefert implizit die Inverse von $\overleftrightarrow{A}$

Weitere Möglichkeit zur Lösung des Gleichungssystems (Glg. (2.3)**):**
Berechnung der inversen Matrix durch Gauß-Elimination
Dazu Transformation: $A|B \rightarrow E|A^{-1}$ mit n × n Einheitsmatrix E

Ablauf der Matrizeninversion

1. Gauß-Elimination mit Spaltenpivotwahl
   Elimination der Elemente unter- **und** oberhalb der Diagonalen
   (Erzeugung der Diagonalmatrix $\overleftrightarrow{A} \rightarrow \overleftrightarrow{D}$)
2. Multiplikation der Pivotzeile mit $\frac{1}{a_{ii}}$ für $\overleftrightarrow{D} \rightarrow \overleftrightarrow{E}$
   (Bedingung für Diagonalelemente)

2.3.5 Beispiel zur Matrizeninversion

Aufgabe:
Berechnung der Inversen von

$$\overleftrightarrow{A} = \begin{pmatrix} 3 & 5 & 1 \\ 2 & 4 & 5 \\ 1 & 2 & 2 \end{pmatrix} \begin{matrix} \mathrm{E}_1 \\ \mathrm{E}_2 \\ \mathrm{E}_3 \end{matrix}$$

unter Berücksichtigung der Benutzung der Notation von Abschnitt 2.3.2.
Genereller Ablauf:

1. Vertauschung von E_1 und E_3:

$$\left(\begin{array}{ccc|ccc} 3 & 5 & 1 & 1 & 0 & 0 \\ 2 & 4 & 5 & 0 & 1 & 0 \\ 1 & 2 & 2 & 0 & 0 & 1 \end{array}\right) \rightarrow \left(\begin{array}{ccc|ccc} 1 & 2 & 2 & 0 & 0 & 1 \\ 2 & 4 & 5 & 0 & 1 & 0 \\ 3 & 5 & 1 & 1 & 0 & 0 \end{array}\right) \begin{matrix} \mathrm{E}'_1 \\ \mathrm{E}'_2 \\ \mathrm{E}'_3 \end{matrix}$$

2. $\mathrm{E}'_2 - (\mathrm{E}'_1 \cdot 2)$:

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

3. $\mathrm{E}''_3 - (\mathrm{E}''_1 \cdot 3)$:

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

4. $\mathrm{E}_2''' \leftrightarrow (\mathrm{E}_3''' \cdot (-1))$:

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

5. $\mathrm{E}_1'''' - (\mathrm{E}_2'''' \cdot 2)$:

$$\left(\begin{array}{ccc|ccc} 1 & 0 & -8 & 2 & 0 & -5 \\ 0 & 1 & 5 & -1 & 0 & 3 \\ 0 & 0 & 1 & 0 & 1 & -2 \end{array}\right)$$

6. $\mathrm{E}_1^{\mathrm{V}} + (\mathrm{E}_3^{\mathrm{V}} \cdot 8)$:

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 2 & 8 & -21 \\ 0 & 1 & 5 & -1 & 0 & 3 \\ 0 & 0 & 1 & 0 & 1 & -2 \end{array}\right)$$

7. $\mathrm{E}_2^{\mathrm{VI}} - (\mathrm{E}_3^{\mathrm{VI}} \cdot 5)$:

$$\underbrace{\left(\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right.}_{\mathrm{E}} \underbrace{\left.\begin{array}{ccc} 2 & 8 & -21 \\ -1 & -5 & 13 \\ 0 & 1 & -2 \end{array}\right)}_{\mathrm{A}^{-1}}$$

Aber:

- Verfahren ist numerisch nicht sehr günstig
  - Skalierung wie bei der Gauß-Elimination: $O(n^3)$ Rechenoperationen
- Verfahren ist häufig numerisch nicht stabil

In Python: Lösungsroutine scipy.linalg.inv()

2.3.6 LR-Zerlegung

Wie schon erwähnt:
"Python"benutzt die LR-Zerlegung, anstatt der Gauß-Elimination um ein Gleichungssystem (Glg. (2.3)) zu lösen, da die LR-Zerlegung numerisch effizienter ist.
Anmerkung:
Im Englischen wird die LR-Zerlegung auch als LU-Decomposition bezeichnet.
Die grundlegende Idee der LR-Zerlegung basiert auf der **Matrixfaktorisierung**.

Matrixfaktorisierung in der LR-Zerlegung
Die quadratische Matrix $A \in \mathbb{R}^{n \times n}$ wird in die linke untere Dreiecksmatrix L und in die rechte obere Dreiecksmatrix R zerlegt:

$$\overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{R}$$

Nebenbedingung:
Definition der Diagonalelemente $\ell_{ii} = 1$ für $i = 1, 2, \ldots n$
(Zuordnung zur L-Matrix)

Durch Anwendung der Dreiecksform:
Lösung des Gleichungssystems (Glg. (2.3)) mittels

$$\overleftrightarrow{A} \cdot \vec{x} = \overleftrightarrow{L} \cdot \overleftrightarrow{R} \cdot \vec{x} = \vec{b}$$

durch Vorwärts (L)- und der Rückwärtssubstitution (R) entsprechend den Gleichungen (2.6) und (2.5)!

Beweis der Äquivalenz der LR-Zerlegung zum Ausgangssystem:
Das Gleichungssystem

$$\overleftrightarrow{A} \cdot \vec{x} = \vec{b} \tag{2.7}$$

kann mittels

$$\overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{R} \tag{2.8}$$

mit

$$\overleftrightarrow{L} \cdot \vec{y} = \vec{b} \text{ und } \overleftrightarrow{R} \cdot \vec{x} = \vec{y}. \tag{2.9}$$

dargestellt werden. Einsetzen von Gleichung (2.8) in obige allgemeine Matrizengleichung (vgl. Glg. (2.3)) ergibt

$$\overleftrightarrow{A} \cdot \vec{x} = (\overleftrightarrow{L} \cdot \overleftrightarrow{R}) \cdot \vec{x} = \overleftrightarrow{L} \cdot (\overleftrightarrow{R} \cdot \vec{x}) = \overleftrightarrow{L} \cdot (\vec{y}) = \vec{b}$$

welches genau mit den obigen Forderungen übereinstimmt.

Lösung eines Gleichungssystems mittels LR-Zerlegung

1. Berechnung des Hilfsvektors $\vec{y}$ durch Vorwärtssubstitution (Glg.(2.6)):

$$\overleftrightarrow{L} \cdot \vec{y} = \vec{b} \tag{2.10}$$

2. Berechnung der Lösung $\vec{x}$ durch Rückwärtssubstitution (Glg. (2.5)):

$$\overleftrightarrow{R} \cdot \vec{x} = \vec{y} \tag{2.11}$$

Ein Beispiel für eine Matrixfaktorisierung mittels LR-Zerlegung:

$$\overleftrightarrow{A} = \begin{pmatrix} 3 & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 1/2 & 1 \end{pmatrix}}_{\text{L}} \cdot \underbrace{\begin{pmatrix} 3 & 1 & 6 \\ 0 & 2/3 & -1 \\ 0 & 0 & -1/2 \end{pmatrix}}_{\text{R}} \tag{2.12}$$

Die Bestimmung der einzelnen Koeffizienten in den Dreicksmatrizen erfolgt nach:

$$\ell_{11} \cdot r_{11} = a_{11}$$

$$r_{1i} = \frac{a_{1i}}{\ell_{11}} \quad \text{und} \quad \ell_{i1} = \frac{a_{i1}}{r_{11}} \quad \text{für } i = 2, 3, \ldots n$$

$$\ell_{ii} \cdot r_{ii} = a_{ii} - \sum_{k=1}^{i-1} \ell_{ik} \cdot r_{ki} \quad \text{für } i = 2, 3, \ldots n-1$$

$$r_{ij} = \frac{1}{l_{ii}} \left(a_{ij} \sum_{k=1}^{i-1} l_{ik} \cdot u_{kj} \right) \quad \text{und} \quad \ell_{ji} = \frac{1}{r_{ii}} \left(a_{ji} \sum_{k=1}^{i-1} \ell_{jk} \cdot r_{ki} \right) \quad \text{für alle } j = i+1, i+2, \ldots n$$

$$\ell_{nn} \cdot r_{nn} = a_{nn} - \sum_{k=1}^{n-1} \ell_{nk} \cdot r_{kn}$$

Form der L und R - Matrizen:

$$\overleftrightarrow{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \ldots & r_{1n} \\ & r_{22} & r_{23} & \ldots & r_{2n} \\ & & r_{33} & & r_{3n} \\ & \varnothing & & \ddots & \vdots \\ & & & & r_{nn} \end{pmatrix}, \quad \overleftrightarrow{L} = \begin{pmatrix} 1 & & & & \\ \ell_{21} & 1 & & \varnothing & \\ \ell_{31} & \ell_{32} & 1 & & \\ \vdots & \vdots & & \ddots & \\ \ell_{n1} & \ell_{n2} & \ldots & \ell_{n,n-1} & 1 \end{pmatrix}$$

Anwendung der LR-Zerlegung

1. Durchführung der Dreieckszerlegung: $\overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{R}$
2. $\overleftrightarrow{L} \cdot \vec{y} = \vec{b}$ (Bestimmung des Hilfsvektors $\vec{y}$ durch Vorwärtssubstitution (Glg. (2.6)))
3. $\overleftrightarrow{R} \cdot \vec{x} = \vec{y}$ (Bestimmung der Lösung $\vec{x}$ durch Rückwärtssubstitution (Glg. (2.5)))

Rechenaufwand:
$O(n^2)$ Rechenoperationen, falls $\overleftrightarrow{L}$ und $\overleftrightarrow{R}$ bekannt sind.
Daher: Bei gegebener Matrix und vorhandener Faktorisierung werden $O(n^2)$ Rechenoperationen benötigt, um Gleichungssysteme zu lösen, welche Matrix $\overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{R}$ beinhalten.

Amerkungen:

- Keine LR-Zerlegung für einfache Matrizen (z. B. 2×2 Matrizen)
- Falls Zeilentausch für Bestimmung von $\overleftrightarrow{L}$ und $\overleftrightarrow{R}$ notwendig ist
  - Einführung der Permutationsmatrix $\overleftrightarrow{P}$

2.3.7 Permutationsmatrix

Die Permutationsmatrix kann eingesetzt werden, um den expliziten Zeilentausch bei der Gauß-Elimination im Rahmen der LU-Zerlegung zu vermeiden.

Definition der Permutationsmatrix

Eine Permutationsmatrix $\overleftrightarrow{P}$ $(n \times n)$ ist eine Matrix mit genau einem Element mit dem Wert 1 in jeder Spalte und jeder Zeile, alle anderen Elemente sind null.

Beispiel:

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

Beispielhafter Einsatz der Permutationsmatrix:

i) $\overleftrightarrow{P} \cdot \overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{R}$

ii) $\overleftrightarrow{L} \cdot \vec{y} = \overleftrightarrow{P} \cdot \vec{b}$

iii) $\overleftrightarrow{R} \cdot \vec{x} = \vec{y}$

2.3.8 Diagonal dominante und positiv definite Matrizen

Eine $n \times n$ Matrix ist diagonal-dominant, wenn

Definition von diagonal dominanten Matrizen

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \text{ für alle } i = 1, 2, \ldots n$$

gilt.

Vorteile von diagonal-dominanten Matrizen:

- $\overleftrightarrow{A}$ ist nicht-singulär (det A $\neq 0 \rightarrow$ eindeutige Lösbarkeit des Gleichungssystems (z. B. zur Interpolation von Funktionen))
- Anwendung der Gauß-Elimination auf jedes Linearsystem (Glg. (2.3)) $\overleftrightarrow{A} \cdot \vec{x} = \vec{b}$ ohne Zeilen- und Spaltentausch

Definition von positiv definiten Matrizen

Eine Matrix $\overleftrightarrow{A}$ ist positiv definit, falls sie symmetrisch ist ($a_{ij} = a_{ji}$) und falls $\vec{x}^{\mathrm{T}} \cdot \overleftrightarrow{A} \cdot \vec{x} > 0$ für jede n-dimensionale Spaltenmatrix $\vec{x} \neq 0$ gilt.

Anmerkung zur Notation:
Das Superscript T kennzeichnet die Transponierte eines Vektors oder einer Matrix, d. h. eine Matrix, in der die Koeffizienten entlang der Hauptdiagonalen der Ursprungsmatrix gespiegelt sind.

Eine positiv definite Matrixäquivalenz zu $\overleftrightarrow{A}$ mit den Vorteilen von diagonal dominanten Matrizen kann erzeugt werden durch:

- Faktorisierung von $\overleftrightarrow{A}$ in $\overleftrightarrow{L} \cdot \overleftrightarrow{L}^{\mathrm{T}}$, wobei $\overleftrightarrow{L}$ eine untere Dreiecksmatrix mit positiven Diagonalelementen darstellt ($\overleftrightarrow{A}$ muss symetrisch sein $\Rightarrow$ Nutzung des Cholesky-Verfahren (Abschnitt 2.3.9))
- Faktorisierung von $\overleftrightarrow{A}$ in $\overleftrightarrow{L} \cdot \overleftrightarrow{D} \cdot \overleftrightarrow{L}^{\mathrm{T}}$, wobei $\overleftrightarrow{L}$ eine untere Dreiecksmatrix mit Einsen auf ihrer Diagonalen und $\overleftrightarrow{D}$ eine Diagonalmatrix mit positiven Diagonalelementen darstellt

Falls $\overleftrightarrow{L}^{\mathrm{T}} = \overleftrightarrow{R}$: $\quad \overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{D} \cdot \overleftrightarrow{L}^{\mathrm{T}} = \overleftrightarrow{L} \cdot \overleftrightarrow{D} \cdot \overleftrightarrow{R}$ (LDR-Zerlegung)

LR-Zerlegung in Python:
`scipy.linalg.lu()` oder `scipy.linalg.lu_solve()` (Lösen des linearen Gleichungssystems)

2.3.9 Cholesky-Zerlegung

Bedingung für Cholesky-Zerlegung:
Matrix $\overleftrightarrow{A}$ ist symmetrisch und positiv definit, d.h. für die zugehörige quadratische Form $\overleftrightarrow{Q}(\vec{x})$ gilt:

$$\overleftrightarrow{Q}(\vec{x}) = \vec{x}^{\mathrm{T}} \cdot \overleftrightarrow{A} \cdot \vec{x} = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} x_i x_k > 0 \text{ für alle } \vec{x} \in \mathbb{R},\ \vec{x} \neq 0$$

(Anm.: Erzeugung der transponierten Matrix: Spiegelung an der Hauptdiagonalen).

Zu jeder symmetrischen und positiv definiten Matrix $\overleftrightarrow{A}$ (vergleiche auch Abschnitt 2.3.8) existiert eine eindeutige Dreieckszerlegung $\overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{L}^{\mathrm{T}}$ mit unterer linker Dreiecksmatrix $\overleftrightarrow{L}$:

$$\overleftrightarrow{L} = \begin{pmatrix} \ell_{11} & & & & \\ \ell_{21} & \ell_{22} & & \varnothing & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \vdots & & \ddots & \\ \ell_{n1} & \ell_{n2} & \dots & \dots & \ell_{nn} \end{pmatrix}$$

mit $\ell_{11} = \sqrt{a_{11}}$

$$\ell_{j1} = \frac{a_{j1}}{\ell_{11}} \text{ für alle } j = 2, 3, \dots n$$

$$\ell_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2 \right)^{\frac{1}{2}} \text{ für alle } i = 2, 3, \dots n-1$$

$$\ell_{ji} = \frac{1}{\ell_{ii}} \left(a_{ji} - \sum_{k=1}^{i-1} \ell_{jk} \ell_{ik} \right) \text{ für alle } j = i+1, i+2, \dots n$$

$$\ell_{nn} = \left(a_{nn} - \sum_{k=1}^{n-1} \ell_{nk}^2 \right)^{\frac{1}{2}}$$

Ablauf des Cholesky-Verfahrens

1. $\overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{L}^{\mathrm{T}}$ (Ermittlung der Cholesky-Zerlegung und Substitution $\overleftrightarrow{L}^{\mathrm{T}} \cdot \vec{x} = \vec{c}$ für Lösung $\overleftrightarrow{A} \cdot \vec{x} = \vec{b}$)
2. $\overleftrightarrow{L} \cdot \vec{c} = \vec{b}$ (Bestimmung des Hilfsvektors $\vec{c}$ durch Vorwärtssubstitution (Glg. (2.6)))
3. $\overleftrightarrow{L}^{\mathrm{T}} \cdot \vec{x} = \vec{c}$ (Bestimmung der Lösung $\vec{x}$ durch Rückwärtssubstitution (Glg. (2.5))

Anmerkung:
Für große Werte von n ist der Aufwand beim Cholesky-Verfahren etwa halb so groß wie bei der LR-Zerlegung.

2.4 Zusammenfassung der wichtigsten Punkte des Kapitels

Kurzzusammenfassung: Lösung von linearen Gleichungssystemen

- Hauptproblem: Lösung von $\overleftrightarrow{A} \cdot \vec{x} = \vec{b}$ (Bestimmung von $\vec{x}$)
- Standardverfahren: Gauß-Elimination (bringt Matrix $\overleftrightarrow{A}$ in $\overleftrightarrow{R}$ Form)
- Nachteile: langsam $O(n^3)$, numerisch instabil, keine Garantie für Lösung
- Ausweg:
  1. Matrixinversion
  2. LR-Zerlegung ($\overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{R}$)
     Berechnung von $\vec{x}$ durch Vorwärts- und Rückwärtssubstitution nach Anwendung der Matrizen $\overleftrightarrow{L}$ und $\overleftrightarrow{R}$
  3. Cholesky-Verfahren (analog zur LR-Zerlegung, hier aber $\overleftrightarrow{A} = \overleftrightarrow{L} \cdot \overleftrightarrow{L}^{\mathrm{T}}$)
     Falls $\overleftrightarrow{L}$ ermittelt wurde, einfache Berechnung von $\overleftrightarrow{L}^{\mathrm{T}}$
  4. Orthogonalisierungsverfahren, z.B. Householder-Verfahren (nicht besprochen)
     $\overleftrightarrow{A} = \overleftrightarrow{Q} \cdot \overleftrightarrow{R}$ mit $\overleftrightarrow{Q} \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ und $\overleftrightarrow{R} \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$

3 Analysis: Darstellung von Funktionen

Ausgangsfrage:
Wie können Computer Funktionen wie z.B. die Sinusfunktion darstellen oder berechnen?
→ Prozessoren können nur Grundrechenarten
Lösung:
Darstellung von Funktionen durch stückweise definierte Polynome!
Idee der Interpolation:
Rückführung eines Polynoms auf eine Ansammlung von Werten
Polynome besitzen Ableitungen und Integrale: Natürliche Auswahl von Polynomen zur Approximation von Funktionen!

3.1 Effiziente Berechnung von Polynomen: Horner-Schema

Gegeben sei ein Polynom

$$P(x) = \sum_{i=0}^{n} c_i \cdot x^i$$

mit $(n+1)$ Termen (Polynom vom Grad n)

Rechenaufwand zur Auswertung:

- $O(n)$ Multiplikationen der Potenzen mit Koeffizienten
- $O(n-1)$ Mulitplikationen zur Bildung der Potenzen

Einfache Auswertung resultiert in $O(2n)$ Rechenoperationen!
Außerdem:
Benutzung des Zwischenspeichers für Werte der Potenzen bei gegebenen Wert von x.

Höhere Effizienz: Horner-Schema

$$P(x) = \sum_{i=0}^{n} c_i \cdot x^i = c_0 + x(c_1 + x(c_2 + x(...(c_{n-1} + c_n x)))...)$$

Rechenaufwand des Horner-Schemas:

- $O(n)$ Additionen (keine explizite Neuberechnung der Potenzen)
- Kein Speichern der Zwischenwerte nötig!

Fazit:
Das Horner-Schema ermöglicht eine schnellere Berechnung von Polynomen mit $O(n)$ Rechenoperationen und Werte müssen nicht zwischengespeichert werden!

Beispiel:
$1 + 7x - 5x^2 - 3x^3 + x^4 = P(x)$

Einfacher Ansatz: $7 \cdot x - (5 \cdot x) \cdot x - (3 \cdot x^2) \cdot x + (4 \cdot x^3) \cdot x$
$\Rightarrow$ Rechenoperationen: 7 Multiplikationen!

Horner: $x \cdot (7 + x \cdot (-5 + x \cdot (-3 + x \cdot 4)))$
$\Rightarrow$ Rechenoperationen: 4 Multiplikationen

In Python: `numpy.polyval()`
Anm.: Auswertung hier in umgekehrter Reihenfolge $\sum_{i=0}^{n} c_i \cdot x^{n-i}$

3.1.1 Polynomdivision zur Bestimmung von Nullstellen mittels des Horner-Schemas

Beweis für Nullstellenbestimmung mit Horner-Schema:

$$
\begin{aligned}
P(x) &= \sum_{i=0}^{n} c_i \cdot x^i = x \cdot \left(\sum_{i=0}^{n-1} c_{i+1} \cdot x^i \right) + c_0 \\
&= x \cdot \left(\sum_{i=0}^{n-2} d'_{i+1} \cdot x^i \cdot (x - x_0) + d'_0 \right) + c_0 \\
&\text{mit } d'_i = c_{i+1} + x_0 \cdot (c_{i+2} + x_0 \cdot (\dots \ (c_{n-1} + x_0 \cdot c_n)) \dots) \equiv d_{i+1} \\
P(x) &= \sum_{i=0}^{n-2} d_{i+2} \cdot x^{i+1} \cdot (x - x_0) + \underbrace{d_1 \cdot x + c_0}_{d_0} \\
&= \left(\sum_{i=0}^{n-1} d_{i+1} \cdot x^i \right) (x - x_0) + d_0
\end{aligned}
$$

$d_0 \dots$ Divisionsrest; bei Teilbarkeit von $P(x)$ durch $(x - x_0)$ ist $d_0 = 0$

3.2 Taylor-Polynome

Die Grundlage in der Anwendung von Polynomen zur Interpolation liegt im Weierstraßschen Approximationssatz

Weierstraßscher Approximationssatz

Gegeben sei eine auf $[a, b]$ definierte und stetige Funktion $f(x)$. Dann gibt es zu jedem $\epsilon > 0$ ein auf $[a, b]$ definiertes Polynom $P(x)$, so dass

$$|f(x) - P(x)| < \epsilon \text{ für alle } x \in [a, b]$$

gilt.

Eine einfache Ausnutzung des obigen Approximationssatzes liegt in der Anwendung von Taylor-Polynomen

Definition: Taylor-Reihe und Taylor-Polynome

Ist eine Funktion $f(x)$ um einen Punkt hinreichend gut differenzierbar, so lässt sie sich als Taylor-Reihe

$$f(x) = P_n(x) + R_n(x)$$

mit

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

und Restglied

$$R_n(x) = \frac{f^{(n-1)}(\zeta(x))}{(n+1)!}(x - x_0)^{n+1}$$

mit Zahl $\zeta(x)$ zwischen x und x_0 darstellen.
Die Polynome $P_n(x)$ heißen Taylorpolynome mit

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + ...$$

$f^{(k)}$ sind die entsprechenden Ableitungen und Restglied $R_n(x)$ kennzeichnet den Abbruchfehler, falls $n \neq \infty$ (endliche Summation).

Bei Existenz der Ableitungen und $(x - x_0) \ll 1$: schnelle Konvergenz der Taylor-Reihe!
Bedeutung:
Endliche Anzahl von Taylor-Termen stellt eine gute polynomielle Näherung dar!
Problem:
Taylor-Polynome stimmen zwar mit einer gegebenen Funktion an einer festen Stützstelle sehr gut überein, aber Genauigkeit bezieht sich auf Stützstelle!
Ein gutes Interpolationspolynom muss aber über ein ganzes Intervall eine gute Approximation liefern!

Beispiel für Taylor-Polynome:

$$f(x) = e^x \text{ mit } x_0 = 0$$

Resultierende Taylor-Polynome:
$P_0(x) = 1,\ \ P_1(x) = 1 + x,\ \ P_2(x) = 1 + x + \frac{x^2}{2},\ \ P_3(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$

x	$P_0(x)$	$P_1(x)$	$P_2(x)$	$P_3(x)$	e^x
0	1	1	1	1	1
0.5	1	1.5	1.625	1.646	1.649
1.0	1	2.0	2.5	2.667	2.718

- Grössere Abweichungen für größere Intervalle $(x - x_0)$

Oft: Bessere Approximation bei Verwendung von Taylor-Polynomen höherer Ordnung!

Weiteres Beispiel: Taylor-Polynome zur Approximation der Sinusfunktion
$f(x) = \sin(x); \quad f'(x) = \sin'(x) = \cos(x); \quad f''(x) = \cos'(x) = -\sin(x)$
Für Intervall $[0; \pi/2]$ mit $x_0 = 0$:

$$\sin(x) = \sum_{k=0}^{\infty} \frac{\sin^{(k)}(0)}{k!} x^k = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot x^{2k+1}}{(2k+1)!}$$

Veranschaulichung der Abweichung für f(x) = e^x (in Abbildung 3.1)

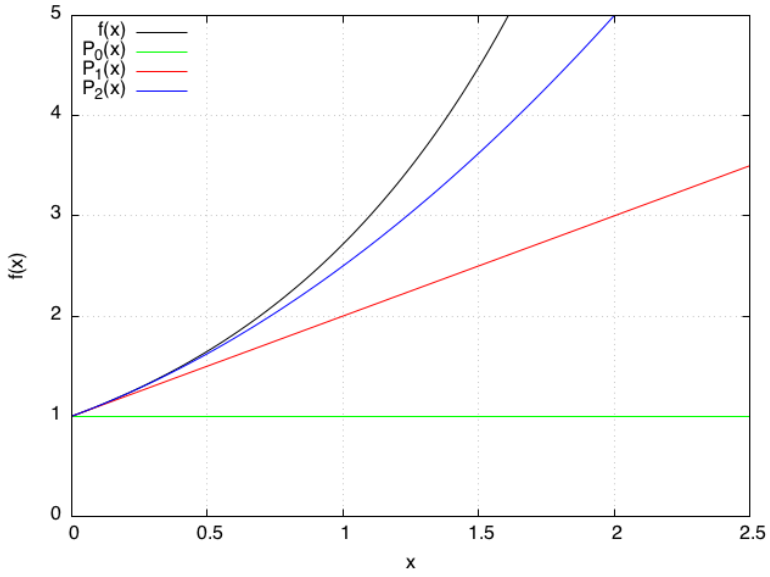


Abbildung 3.1: Veranschaulichung der Abweichung der Taylor-Polynome für $f(x) = e^x$

Genauigkeit der Approximation steigt mit der Ordnung der Polynome!

Anmerkung: Abweichung der Taylor-Polynome ist von genereller Natur!
→ Konzentration der zur Approximation verwendeten Informationen auf Stützstelle x_0!

Wichtig:
Einsatz von Taylorpolynomen nur bei Approximation von Stellen x mit $(x - x_0) \ll 1$

Ausweg: Bessere Interpolationspolynome:
Hinzunahme mehrerer Stützstellen!

3.3 Lagrange Polynome

Feststellung: Taylor-Polynome sind meistens ungeeignet für Approximationen

- nur gü für kleine Intervalle
- Ableitungen müssen exisiteren

Idee: Approximierende Polynome mit spezifizierenden festen Punkten (z. B. Datenmenge)

Ziel: Gute Polynome auch für Vorhersagen außerhalb der Intervallgrenzen!

Beispiel: Bestimmung eines Polynoms vom Grad 1.
Wertepaare (x_0, y_0) und (x_1, y_1) ist äquivalent zur Darstellung mittels einer Funktion f mit $f(x_0) = y_0$ und $f(x_1) = y_1$

Ansatz: Lineares Polynom

$$P(x) = \frac{(x - x_1)}{(x_0 - x_1)} f(x_0) + \frac{(x - x_0)}{(x_1 - x_0)} f(x_1) \tag{3.1}$$

Für $x = x_0$: $P(x_0) = f(x_0) = y_0$
Für $x = x_1$: $P(x_1) = f(x_1) = y_1$

Somit erfüllt $P(x)$ die gestellten Forderungen zur Approximation von Wertepunkten!

3.3.1 Konzept der linearen Interpolation

Nutzung von Polynomen höheren Grades

Ziel: Konstruktion eines Polynoms des Grades höchstens gleich n für Interpolation von $(n + 1)$ Punkten
$(x_0, f(x_0))$, $(x_1, f(x_1))$, $\ldots (x_n, f(x_n))$

s. obiges Beispiel für $(x_0, f(x_0))$, $(x_1, f(x_1))$ (Gleichung (3.1))

Definition:

$$L_0(x) = \frac{(x - x_1)}{(x_0 - x_1)}, \quad L_1(x) = \frac{(x - x_0)}{(x_1 - x_0)}$$

Gleichung (3.1) $\rightarrow P(x) = L_0(x)f(x_0) + L_1(x)f(x_1)$

$$\begin{array}{llllll} \text{Für } x = x_0: & L_0(x_0) = 1 & \text{und} & L_1(x_0) = 0 & \rightarrow & P_1(x_0) = f(x_0) \\ \text{Für } x = x_1: & L_1(x_1) = 1 & \text{und} & L_0(x_1) = 0 & \rightarrow & P_1(x_1) = f(x_1) \end{array}$$

Lagrangesches Interpolationspolynom

Konstruktion eines Quotienten $L_{n,k}(x)$ für jedes $k = 0, 1, \ldots, n$ mit der Eigenschaft: $L_{n,k}(x_i) = 0$ für $i \neq k$ und $L_{n,k}(x_k) = 1$
Um $L_{n,k}(x_i) = 0$ für $i \neq k$ zu genügen, muss der Zähler von $L_{n,k}$ den Term $(x - x_0)(x - x_1)(x - x_2)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n)$ enthalten.
Um $L_{n,k}(x_k) = 1$ zu genügen, muss der Nenner von $L_{n,k}(x)$ gleich diesem Term in $x = x_k$ entwickelt sein!

Daraus folgt:

$$L_{n,k}(x) = \frac{(x - x_0)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n)}{(x_k - x_0)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_n)} = \prod_{i \neq k}^{k} \frac{(x - x_i)}{(x_k - x_i)} \tag{3.2}$$

für jedes $k = 0, 1, ..., n$ und die Definition des **n-tes Lagrangesches Interpolationspolynoms** ergibt sich mit

$$P_n(x) = f(x_0)L_{n,0}(x) + \cdots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x) \tag{3.3}$$

Also: Sind x_0, x_1, ..., x_n $(n+1)$ verschiedene Stützstellen und $f(x)$ eine Funktion, deren Werte an diesen Stellen gegeben sind, dann ist $P_n(x)$ das einzige Polynom des Grades höchstens gleich n, das mit $f(x)$ in x_0, x_1, ..., x_n übereinstimmt.
Falls Grad n bekannt ist: $L_{n,k}(x) \equiv L_k(x)$.

Beispiel:

$$x_0 = 2, \quad x_1 = 2.5, \quad x_2 = 4; \quad f(x) = \frac{1}{x}$$

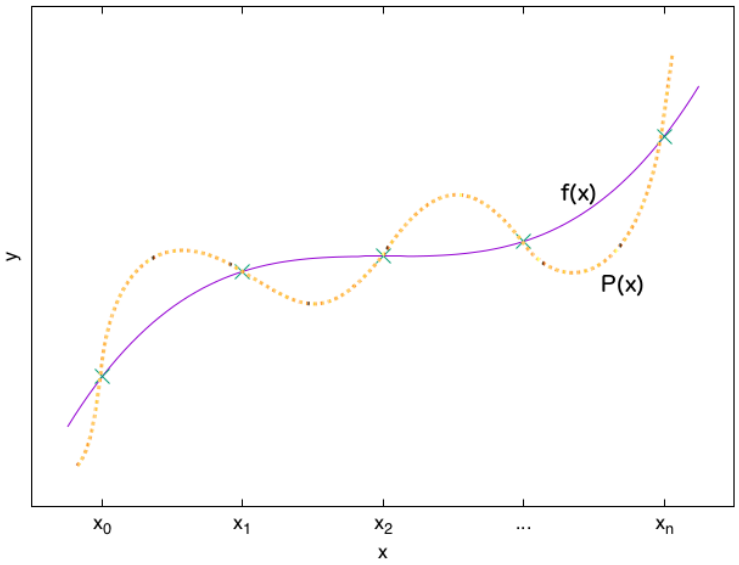


Abbildung 3.2: $P(x)$ und $f(x)$

$$\rightarrow \quad L_0(x) = \frac{(x-2.5)(x-4)}{(2-2.5)(2-4)}$$

$$L_1(x) = \frac{(x-2)(x-4)}{(2.5-2)(2.5-4)}$$

$$L_2(x) = \frac{(x-2)(x-2.5)}{(4-2)(4-2.5)}$$

$$\text{Mit } f(x) = \frac{1}{x} \quad \rightarrow \quad f(x_0) = \frac{1}{2}; \quad f(x_1) = 0.4; \quad f(x_2) = 0.25$$

Damit:

$$P_2(x) = \sum_{k=0}^{2} f(x_k) L_k(x) = (0.05 \cdot x - 0.425) \cdot x + 1.15$$

Approximation von $f(3) = \frac{1}{3}$ ist $f(3) \approx P(3) = 0.325$

Fehler der Lagrangeschen Polynome:

Restglied:

$$\frac{f^{(n+1)}(\zeta(x))}{(n+1)!} \cdot (x - x_0)^{n+1} \quad \text{mit } \zeta(x) \in [x, x_0]$$

Generelle Fehlerformel des Lagrangeschen Polynoms:

$$f(x) = P_n(x) + \frac{f^{(n+1)}}{(n+1)!}(\zeta(x))(x - x_0)(x - x_1)\dots(x - x_n)$$

mit $\zeta(x)$ beliebig zwischen x_0, x_1, …, x_n und x.

Problem der Methode:
Durch schwere Anwendbarkeit des Fehlergliedes

- Schwierigkeit der Bestimmung des Grades der am besten interpolierenden Approximation.

Generelles Problem:

- Keine Kenntnis über höhere Ableitungen der Funktion!

In Python: `scipy-interpolate.lagrange(...)`

3.3.2 Rekursionsformel für Lagrangesche Polynome

Rekursionsformel von Aitken
Die Funktion $f(x)$ sei in x_0, x_1, …, x_k, definiert und x_j und x_i seien zwei verschiedene Stützstellen in dieser Menge. Falls

$$P(x) = \frac{(x - x_j)P_{0,1,\dots,j-1,j+1,\dots,k}(x) - (x - x_i)P_{0,1,\dots,i-1,i+1,\dots,k}(x)}{(x_i - x_j)} \tag{3.4}$$

gilt, dann ist P das k-te Polynom, das f an den $k+1$ Stützstellen x_0, x_1, …, x_k interpoliert.

Nutzung von Glg. (3.4) zur effektiven Berechnung des Lagrangepolynoms:

Notation: $P_{m_1,m_2,m_3,\dots,m_k}(x)$ bezeichnet das Lagrangesche Polynom, welches mit $f(x)$ an den k Stützstellen x_{m_1}, x_{m_2}, …, x_{m_k} übereinstimmt.

Definition:

$$Q \equiv P_{0,1,\dots,i-1,i+1,\dots,k}; \quad \hat{Q} \equiv P_{0,1,\dots,j-1,j+1,\dots,k}$$

Q und $\hat{Q}$ sind Polynome vom Grad $k-1$ oder weniger!
$\rightarrow$ Grad $P(x)$ kann höchstens k sein!

Für $0 \leqslant r \leqslant k$ und $r \neq i, j$ folgt:

$$P(x_r) = \frac{(x_r - x_j)\hat{Q}(x_r) - (x_r - x_i)Q(x_r)}{x_i - x_j} = \frac{x_i - x_j}{x_i - x_j} f(x_r) = f(x_r)$$

Achtung: Dies gilt nur für $\hat{Q}(x_r) = Q(x_r) = f(x_r)$!
Es gilt:

$$P(x_i) = \frac{(x_i - x_j)\hat{Q}(x_i) - (x_i - x_i)Q(x_i)}{x_i - x_j} = \frac{x_i - x_j}{x_i - x_j} f(x_i) = f(x_i)$$

und äquivalent für $P(x_j) = f(x_j)$.

Da es nur ein Polynom vom Grad höchstens gleich k gibt, welches mit $f(x)$ in $x_0, x_1, \ldots x_k$ Übereinstimmt, folgt daraus dass $P_{0,1,\ldots k}(x)$ nur durch obige Gleichung (3.4) dargestellt werden kann.

Anmerkung:
Ein rekursiver Aufbau des Polynoms ist möglich!
(Effektive Berechnung durch Neville-Schema (s. Abschnitt 3.3.3)

3.3.3 Das Neville-Schema

Notation: $Q_{i,j} = P_{i-j,i-j+1,\ldots,i-1,i}$

x_0	$P_0 = Q_{0,0}$				
x_1	$P_1 = Q_{1,0}$	$P_{0,1} = Q_{1,1}$			
x_2	$P_2 = Q_{2,0}$	$P_{1,2} = Q_{2,1}$	$P_{0,1,2} = Q_{2,2}$		
x_3	$P_3 = Q_{3,0}$	$P_{2,3} = Q_{3,1}$	$P_{1,2,3} = Q_{3,2}$	$P_{0,1,2,3} = Q_{3,3}$	
x_4	$P_4 = Q_{4,0}$	$P_{3,4} = Q_{4,1}$	$P_{2,3,4} = Q_{4,2}$	$P_{1,2,3,4} = Q_{4,3}$	$P_{0,1,2,3,4} = Q_{4,4}$

Tabelle 3.1: Schema nach Neville.

Vorteil des Neville-Schemas:
Effizienz durch Nutzung sukzessiver Interpolationspolynome höherer Ordnung bei kolonnenweiser Berechnung!
Erzeugung der Ordnung: Nutzung von zwei Indizes ($Q_{i,j}$)

Richtung in Tabelle 3.1:

Von links nach rechts: Anstieg des Grades des interpolierenden Polynoms

Von oben nach unten: Entwicklung in Richtung aufeinanderfolgender Stützstellen x_i

Durch sukzessives Auftreten der Punkte:
Einzig Kenntnis des Startwertes (Startpunkt) und Anzahl zusätzlicher Punkte nötig für Aufbau eines rekursiven Polynoms aus Neville-Schema.

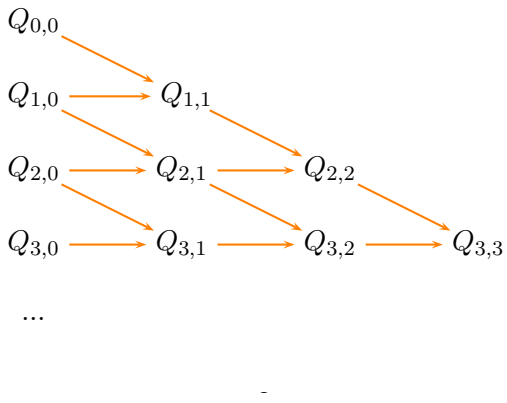
Anwendung des Neville-Schemas:
$Q_{i,0}$ sind Polynome vom Grad 0 (Konstanten); $Q_{i,0} := f(x_i)$

$$Q_{1,1} = Q_{0,0} + \frac{(x - x_0)}{(x_1 - x_0)}(Q_{1,0} - Q_{0,0})$$

$$Q_{2,1} = Q_{1,0} + \frac{(x - x_1)}{(x_2 - x_1)}(Q_{2,0} - Q_{1,0})$$

$$Q_{2,2} = Q_{1,1} + \frac{(x - x_0)}{(x_2 - x_0)}(Q_{2,1} - Q_{1,1})$$

$$\vdots$$



3.3.4 Newtonsche Interpolation: Dividierte Differenzen

Bisher: Iterierte Interpolation an fester Stelle durch sukzessive Polynomnäherungen höherer Ordnung

Nun: Einführung von dividierten Differenzen

Vorteil: Sukzessive Erzeugung der Polynome

Definition: Nullte dividierte Differenz von $f(x)$ bezüglich x_i:

$$f[x_i] = f(x_i)$$

Induktive Definition der restlichen dividierten Differenzen!

Die erste dividierte Differenz von $f(x)$ bzgl. x_i und x_{i+1}:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{(x_{i+1} - x_i)}$$

Daraus ergibt sich für das resultierende Polynom:

$$P_n(x) = f[x_0] + f[x_0, x_1] \cdot (x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \dots$$

Newtonsche Interpolationsformel der dividierten Differenzen

$$P(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \dots x_k] \cdot (x - x_0) \cdots (x - x_{k-1}) \tag{3.5}$$

Daraus ergibt sich wieder ein rekursiver Aufbau des Polynoms ähnlich dem Neville-Schema (Abschnitt 3.3.3 und Tabelle 3.1):

x	$f(x)$	1.dividierte Differenz	2. dividierte Differenz
x_0	$f[x_0]$		
x_1	$f[x_1]$	$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0}$	
x_2	$f[x_2]$	$f[x_1, x_2] = \frac{f[x_2] - f[x_1]}{x_2 - x_1}$	$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$
x_3	$f[x_3]$	$f[x_2, x_3] = \frac{f[x_3] - f[x_2]}{x_3 - x_2}$	$f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1}$

Die resultierende 3. dividierte Differenz ergibt sich aus

$$f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0}$$

3.4 Die Hermitesche Interpolation

Problem bei Lagrange-Interpolation (Abschnitt 3.3 und Gleichung 3.2):

- Polynome stimmen mit einer Funktion $f(x)$ an einer festen Anzahl von Stützstellen überein.

Aber: keine Gewährleisung der guten Approximation an Punkten zwischen Stützstellen!

Ausweg: Hermite-Polynome

Prinzip: Bestimmung eines Polynoms, welches mit der Funktion und ihrer ersten Ableitung an festen Stützstellen übereinstimmt.

Allgemein gilt für das Lagrangepolynom (Glg. 3.3):

- Für $(n+1)$ Stützstellen: Polynom vom Grad n (Lagrange)

Bei Hinzunahme der Ableitungen:

- $n+1$ zusätzliche Bedingungen

Erwarteter Grad des Polynoms: $2n+1$

Definition des Hermiteschen Polynoms

Es sei $f(x) \in C^1[a,b]$ (Menge aller Funktionen mit 1. stetigen Ableitungen) zwischen a, b und $x_i \in [a,b]$ mit $x_i = x_0, x_1, \ldots x_n$ seien verschieden, dann ist das einzige Polynom kleinsten Grades, das mit $f(x)$ und $f'(x)$ in $x_0, \ldots x_n$ übereinstimmt, das durch

$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j) H_{n,j}(x) + \sum_{j=0}^{n} f'(x_j) \hat{H}_{n,j}(x)$$

gegebene Polynom vom Grade höchstens gleich $2n+1$, wobei

$$H_{n,j}(x) = [1 - 2(x - x_j) \cdot L'_{n,j}(x_j)] \cdot L^2_{n,j}(x) \quad \text{und}$$

$$\hat{H}_{n,j}(x) = (x - x_j) \cdot L^2_{n,j}(x) \quad \text{ist.}$$

Notationen nach Gleichungen 3.2 und 3.3:

- $L_{n,j}$ bezeichnet das j-te Lagrangesche Koeffizientenpolynom vom Grad n
- $L'_{n,j}$ bezeichnet die Ableitung des j-ten Lagrangeschen Koeffizientenpolynom vom Grad n.

Insgesamt zeichnen sich die Hermiteschen Polynome durch eine höhere Genauigkeit aus. Der Fehler ist proportional zu

$$f^{(2n+2)}(x - x_0)^2 \cdots (x - x_n)^2$$

Zusammenhang mit dividierten Differenzen (Abschnitt 3.3.4)

Es sei $f \in C^1[a,b]$ und $x_0, x_1, \ldots x_n$ seien verschieden in $[a,b]$, dann ist

$$H_{2n+1}(x) = f[z_0] + \sum_{k=1}^{2n+1} f[z_0, z_1, \ldots z_n](x - z_0) \cdots (x - z_{k-1})$$

wobei $z_{2k} = z_{2k+1} := x_k$ und $f[z_{2k}, z_{2k+1}] := f'(x_k)$ für jedes $k = 1, 2, \ldots n$.

Anmerkung zu $f[z_{2k}, z_{2k+1}]$:
$f[x_0, x_1] = f'(\zeta)$ und $\lim_{x_1 \to x_0} f[x_0, x_1] = f'(x_0)$
Weitere Anmerkungen:

- Leichte Anwendung von Rekursionsvorschriften (Abschnitt 3.3.3)

ABER: Ableitungen müssen bekannt sein!
Durch Einführung von Hermite-Polynomen: Extrapolation auch außerhalb des Wertebereichs!
Aber Vorsicht! Bessere Variante: Romberg-Integration

3.5 Spline-Interpolation

Grundlegende Idee:
Bessere Extrapolation durch stückweise definierte Polynomapproximation von Funktionen

Einfachste Näherung: Verbindung von Funktionswerten durch mehrere Geraden (Abb. 3.3)

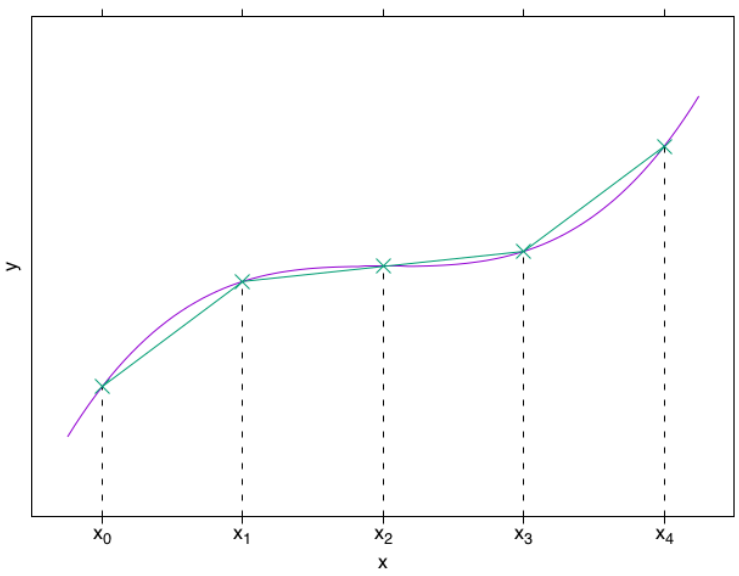


Abbildung 3.3: Einfachste Näherung mit einer Reihe von Geraden

Problem:	Keine Ableitungen an Intervallgrenzen möglich! (keine glatte Funktion, nicht stetig differenzierbar)
Option:	Nutzung von kubischen Hermite-Polynomen zwischen Intervallen.
Aber:	Dafür muss Ableitung bekannt sein!

Ausweg: kubische Spline-Interpolation zwischen Punkten (Knoten)

Typischerweise: Kubisches Spline mit vier Konstanten
→ zwei stetige Ableitungen auf dem Intervall

Bedingungen an kubische Spline-Interpolierende

Gegeben sei eine auf $[a, b]$ definierte Funktion $f(x)$ und eine Menge von Knoten, $a \equiv x_0 < x_1 < x_2 < \cdots < x_n \equiv b$.
Eine kubische Spline-Interpolierende S von $f(x)$ ist eine Funktion, die folgenden Bedingungen genügt:

i) S ist ein kubisches Polynom, bezeichnet mit S_j auf dem Teilintervall $[x_j, x_{j+1}]$ für jedes $j = 0, 1, \ldots n-1$.

ii) $S(x_j) = f(x_j)$ für jedes $j = 0, 1, \ldots n$

iii) $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$ für jedes $j = 0, 1, \ldots n-2$

iv) $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$ für jedes $j = 0, 1, \ldots n-2$

v) $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$ für jedes $j = 0, 1, \ldots n-2$

Zusätzlich ist noch eine der folgenden Randbedingungen erfüllt:

1) $S''(x_0) = S''(x_n) = 0$ (natürlicher oder freier Rand)

2) $S'(x_0) = f'(x_0)$ und $S'(x_n) = f'(x_n)$ (Hermite-Rand)

Die resultierende kubische Spline-Interpolation ist in Abb. 3.4 veranschaulich.

Anmerkungen:
Bedingungen 1) und 2) werden häufig in der Praxis angewandt, obwohl auch andere Randbedingungen existieren!

Typische Normalform eines kubischen Spline-Polynoms

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \quad \text{für jedes } j = 0, 1, \ldots, n-1 \tag{3.6}$$

Bestimmung der Koeffizienten:
Mittels $S_j(x_j) = a_j = f(x_j)$ gibt Bedingung iii)

$$a_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1}) = a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 + d_j(x_{j+1} - x_j)^3$$
$$\text{für jedes } j = 0, 1, \ldots, n-2$$

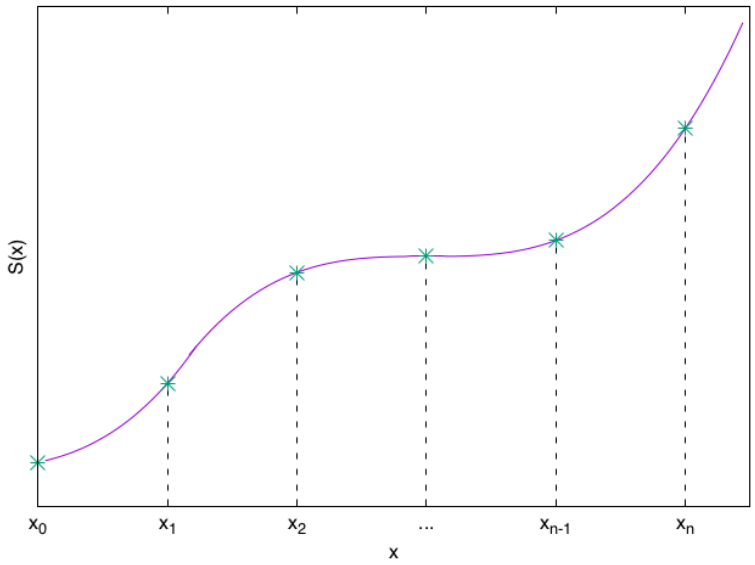


Abbildung 3.4: Kubische Splineinterpolation

Mit Definition $a_n = f(x_n)$ und $h_j = x_{j+1} - x_j$ ergibt sich

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \quad \text{für jedes } j = 0, 1, \ldots, n-1 \tag{3.7}$$

Mit Definition

$$a_n \coloneqq f(x_n); \quad h_j \coloneqq (x_{j+1} - x_j)$$

folgt

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \quad \text{für jedes } j = 0, 1, \ldots n-1 \tag{3.8}$$

Für $b_n = S'(x_n)$ gilt

$$S'_j = b_j + 2c_j(x - x_j) + 3d_j(x - x_j)^2 \tag{3.9}$$

mittels $S'_j(x_j) = b_j$ für jedes $j = 0, 1, \ldots n-1$.
Mit Bedingung iv) ergibt sich

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2 \quad \text{für jedes } j = 0, 1, \ldots, n-1$$

und mittels $c_n = S''(x_n)/2$ und Bedingung iii) folgt

$$c_{j+1} = c_j + 3d_j h_j \quad \text{für jedes } j = 0, 1, \ldots, n-1 \tag{3.10}$$

Auflösen von Gleichung (3.10) nach d_j und Einsetzen in Gleichung (3.8) und Gleichung (3.9) ergibt

$$a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3} \cdot (2c_j + c_{j+1}) \quad \text{und} \tag{3.11}$$

$$b_{j+1} = b_j + h_j(c_j + c_{j+1}) \quad \text{für jedes } j = 0, 1, \ldots n-1 \tag{3.12}$$

Ermittlung der Koeffizienten:

Auflösung von Gleichung (3.11) nach b_j

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$

und nach der Reduktion des Index ergibt

$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j).$$

Einsetzen in Gleichung (3.12) und erneute Reduktion des Index liefert

$$b_j = b_{j-1} + h_{j-1}(c_{j-1} + c_j) \tag{3.13}$$

und damit folgendes Gleichungssystem

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}); \quad j = 1, \ldots, n-1 \tag{3.14}$$

wobei die einzigen Unbekannten die c_j für $j = 0, 1, \ldots n$ sind, welche durch die im Abschnitt 2.3 vorgestellten Lösungsverfahren für lineare Gleichungssysteme aus Gleichung (3.14) bestimmt werden können.

Anmerkungen für Gleichungssystem (3.14)**:**

- Die Koeffizienten h_j und a_j sind durch die Einteilung der Knoten x_j und die Werte $f(x_j)$ wohldefiniert.
- Werte für x_j sind ebenfalls vorgegeben.

Nach Bestimmung der Koeffizienten c_j:
Weitere Bestimmung der b_j aus Gleichung (3.13) und der d_j für alle $j = 0, 1, \ldots n-1$ aus Gleichung (3.10).

Nach Kenntnis aller Koeffizienten:

- Einsetzen in kubisches Spline-Polynom (Gleichung (3.6))

Anmerkungen zu Splines:

- Gute Approximation durch Splines für kleine Intervalle zwischen x_{j+1} und x_j
- Gute Approximation durch Splines für Wohlverhalten der 4. Ableitung

In Python: `scipy.interpolate1d( )`

3.6 Approximation mittels trigonometrischer Polynome

Ziel:

- Approximation von Funktionen mit periodischem Verhalten

$$f(x+T) = f(x) \quad \text{für alle } x \text{ mit Konstante } T$$

Approximiertes Intervall:

- Approximation auf Intervall $[-\pi, \pi]$ für $T = 2\pi$

Hauptanwendungsgebiet von trigonometrischen Approximationen:

- Analyse von periodischen Signalen (Frequenzaufschlüsselung)
- Lösen von Differentialgleichungen (z.B. mittels Fourierdarstellung)

Beispiel für trigonometrische Funktionen zur Approximation:
Fourierdarstellung von $f(x)$ (komplexe Darstellung)

$$S(x) = \sum_{k=-\infty}^{k=\infty} c_k e^{ik\omega x} \quad \text{mit } \omega = \frac{2\pi}{T}$$

$$c_k = \frac{1}{T}\int_0^T f(x)e^{-ik\omega x}\mathrm{dx} = \begin{cases} \frac{1}{2}a_0 & \text{für } k = 0 \\ \frac{1}{2}(a_k - i \cdot b_k) & \text{für } k > 0 \\ \frac{1}{2}(a_{-k} + i \cdot b_{-k}) & \text{für } k < 0 \end{cases}$$

Trigonometrisch:

$$S_n(x) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cdot \cos(k\omega x) + \sum_{k=1}^{n} b_k \cdot \sin(k\omega x) \tag{3.15}$$

$$a_k = \frac{2}{T}\int_0^T f(x)\cos(k\omega x)\mathrm{dx} = \frac{2}{T}\int_0^{T/2} (f(x) + f(-x))\cos(k\omega x)\mathrm{dx}$$

$$b_k = \frac{2}{T}\int_0^T f(x)\sin(k\omega x)\mathrm{dx} = \frac{2}{T}\int_0^{T/2} (f(x) - f(-x))\sin(k\omega x)\mathrm{dx}$$

- Für gerade (symmetrische) Funktionen: $b_k = 0$
- Für ungerade Funktionen: $a_k = 0$

Eigenschaften von trigonometrischen Polynomen können anhand von Fourierdarstellungen verstanden werden!

Fourierdarstellungen

Für jede positive, ganze Zahl n sei die Menge $\mathcal{F}_n$ der trigonometrischen Polynome vom Grade kleiner oder gleich n die Menge aller Linearkombinationen von $\{\phi_0, \phi_1, \dots \phi_{2n-1}\}$, wobei

$$\phi_0(x) = \frac{1}{\sqrt{2\pi}} \quad \text{für alle } k = 1, 2, \dots, n \tag{3.16}$$

$$\phi_k(x) = \frac{1}{\sqrt{\pi}} \cos(kx) \quad \text{für alle } k = 1, 2, \dots, n \tag{3.17}$$

$$\phi_{n+k}(x) = \frac{1}{\sqrt{\pi}} \sin(kx) \quad \text{für alle } k = 1, 2, \dots, n-1 \tag{3.18}$$

gilt.
Die Menge $\{\phi_0, \phi_1, \dots \phi_{2n-1}\}$ ist orthonormal auf $[-\pi, \pi]$ für die Gewichtsfunktion $\omega(x)$ (zur Streckung / Stauchung).
Für $k \neq j$ und $j \neq 0$ gilt dementsprechend

$$\int_{-\pi}^{\pi} \phi_{n+k}(x)\phi_j(x)\mathrm{dx} = \frac{1}{\pi} \int_{-\pi}^{\pi} \sin(kx)\cos(jx)\mathrm{dx} \tag{3.19}$$

Trigonometrische Identität:

$$\sin(kx)\cos(jx) = \frac{1}{2}\sin((k+j)x) + \frac{1}{2}\sin((k-j)x)$$

Damit ergibt sich das Integral in Gleichung (3.19) zu

$$\int_{-\pi}^{\pi} \phi_{n+k}(x)\phi_j(x)\mathrm{dx} = \cdots = \frac{1}{2\pi}\left[\frac{-\cos((k+j)x)}{k+j} - \frac{\cos((k-j)x)}{k-j}\right]_{-\pi}^{\pi} = 0$$

mit

$$\cos((k+j)\pi) = \cos((k+j)(-\pi)) \quad \text{und} \quad \cos((k-j)\pi) = \cos((k-j)(-\pi)).$$

Anmerkungen:

- Ebenso gilt Orthonormalität auch für $k = j$ mittels

$$\sin((k-j)x) = 0$$

- Durch weitere trigonometrische Identitäten können die Produkte auch in Summen überführt werden.

Fourier-Reihe in allgemeiner Form

$$S_n(x) = \sum_{k=0}^{2n-1} a_k \phi_k(x) \quad \text{mit} \quad a_k = \int_{-\pi}^{\pi} f(x)\phi_k(x)dx \quad \text{für alle } k = 0, 1, \ldots, 2n-1 \tag{3.20}$$

Beispiel für trigonometrische Approximation:
Bestimmung des trigonometrischen Polynoms aus $\mathcal{F}_n$, welches $f(x) = |x|$ für $-\pi < x < \pi$ approximiert.
Bestimmung der Koeffizienten aus Gleichung (3.16)

$$a_0 = \int_{-\pi}^{\pi} |x| \cdot \frac{1}{\sqrt{2\pi}} \mathrm{dx} = -\frac{1}{\sqrt{2\pi}} \int_{-\pi}^{0} x \mathrm{dx} + \frac{1}{\sqrt{2\pi}} \int_{0}^{\pi} x \mathrm{dx} = \frac{\pi^2}{\sqrt{2\pi}}$$

und aus Gleichung (3.17)

$$a_k = \frac{1}{\sqrt{\pi}} \int_{-\pi}^{\pi} |x| \cos(kx) \mathrm{dx} = \frac{2}{\sqrt{\pi}} \int_{0}^{\pi} x \cos(kx) \mathrm{dx} = \frac{2}{\sqrt{\pi}k^2}((-1)^k - 1) \quad \text{für alle } k = 1, 2, \ldots, n$$

Schließlich werden die a_{n+k} (konventionell bezeichnet mit $b_k := a_{n+k}$ für $k = 1, 2, \ldots, n-1$) aus Gleichung (3.18) bestimmt:

$$b_k = \frac{1}{\sqrt{\pi}} \int_{-\pi}^{\pi} |x| \sin(kx) \mathrm{dx} = 0 \quad \text{für alle } k = 1, 2, \ldots, n-1$$

Anmerkung: Der Integrand in obiger Gleichungen ist ungerade! $\rightarrow b_k = 0$

Daraus ergibt sich der folgende Ausdruck für das trigonometrische Polynom

$$S_n(x) = \frac{\pi}{2} + \frac{2}{\pi} \sum_{k=0}^{n} \frac{((-1)^k - 1)}{k^2} \cos(kx)$$

und für die resultierende Fourier-Reihe (mit $n \rightarrow \infty$):

$$S(x) = \lim_{n\to\infty} S_n(x) = \frac{\pi}{2} + \frac{2}{\pi} \sum_{k=1}^{\infty} \frac{(-1)^k - 1}{k^2} \cos(kx)$$

Anmerkung:

- Für $|\cos(kx)| \leq 1$ für alle k und x: Schnelle Konvergenz und $S(x)$ existiert für alle reellen Zahlen x.

In Abbildung 3.5 wird $S_1(x)$ als erste Approximation dargestellt.

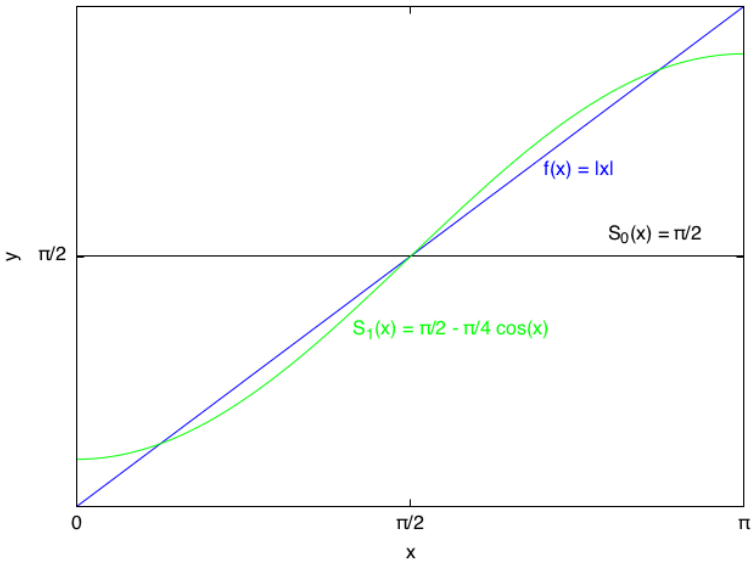


Abbildung 3.5: Darstellung des trignonometrischen Polynoms $S_1(x)$ für $f(x) = |x|$.

3.6.1 Diskrete Fourier-Reihe (Transformation)

Diskretes Analogon für Fourier-Reihe:

- Nutzbar und besonders effektiv für Interpolation von großen Datenmengen
- **ABER:** Werte müssen an äquidistanten Punkten (Stützstellen) auftreten

Beispiel:
$2m$ paarweise definierte Datenpunkte $\{(x_j, y_j)\}_{j=0}^{2m-1}$ an äquidistanten Stellen über geschlossenes Intervall $[-\pi, \pi]$ (z. B. Zeitreihen)
Transformation in Intervall $[-\pi; \pi]$ mit $x_j = -\pi + (j/m) \cdot \pi$ für alle $j = 0, 1, \ldots, 2m-1$ (s. Abbildung 3.6).

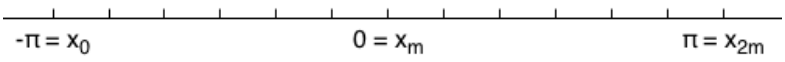


Abbildung 3.6: Transformation von äquidistanten Datenpunkten in das Intervall $[-\pi;]$.

Möglichkeit:
Überführung der Daten in gewünschte Form mittels Linearkombination!

Für ein festes $n < m$ sei die Menge der Funktion $\hat{\mathcal{F}}_n$ betrachtet, welche aus allen Linearkombinationen von

$$\hat{\phi}_0(x) = \frac{1}{2}; \quad \hat{\phi}_k(x) = \cos(kx) \text{ für } k = 1, 2, \ldots n; \quad \hat{\phi}_{n+k}(x) = \sin(kx) \text{ für } k = 1, 2, \ldots n-1$$

bestehen.

Ziel:
Bestimmung der Linearkombination dieser Funktion, für die

$$E(S_n) = \sum_{j=0}^{2m-1} (y_i - S_n(x_j))^2$$

minimal ist.
Minimierung des totalen Fehlers von:

$$E(S_n) = \sum_{j=0}^{2m-1} (y_i - [\frac{a_0}{2} + a_n \cos(nx_j) + \sum_{k=1}^{n-1}(a_k \cos(kx_j) + a_{n+k} \sin(kx_j)])^2$$

durch Methode der kleinsten Quadrate (Abschnitt 4.1). Hieraus ergibt sich

Koeffizienten der diskreten Fourier-Reihe

$$a_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \cos(kx_j) \quad \text{für alle } k = 0, 1, \ldots n \tag{3.21}$$

und

$$b_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \sin(kx_j) \quad \text{für alle } k = 0, 1, \ldots n-1 \tag{3.22}$$

Anmerkungen zur Bestimmung der Konstanten:

- Vereinfachung durch äquidistant eingeteilte Punkte $\{x_j\}_{j=0}^{2m-1}$

- Summation über Punkte in $[-\pi, \pi]$ erfüllt Orthogonalität, d.h. für jedes $k \neq \ell$ folgt

$$\sum_{j=0}^{2m-1} \hat{\phi}(x_j)\hat{\phi}_\ell(x_j) = 0$$

  durch

$$\sum_{j=0}^{2m-1} \cos(rx_j) = 0 \quad \text{und} \quad \sum_{j=0}^{2m-1} \sin(rx_j) = 0$$

  für r, m als positive, ganze Zahlen mit $r < 2m$.

3.6.2 Schnelle Fourier-Transformation

Rekapitulation und Grundlage der schnellen Fouriertransformation: Diskrete Fourier-Reihe (analog zu Gleichung (3.15)):

$$S_n(k) = \frac{a_0}{2} + a_n \cos(nx) + \sum_{k=1}^{n-1} (a_k \cos(kx) + b_k \sin(kx)) \tag{3.23}$$

mit den Koeffizienten a_k und b_k (Gleichungen (3.21) und (3.22))

$$a_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \cdot \cos(kx_j) \quad \text{für alle k=0, 1, ... n}$$

und

$$b_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \cdot \sin(kx_j) \quad \text{für alle k=0, 1, ... n-1}$$

Für $n = m$ ergibt sich aus Gleichung (3.23) dementsprechend

$$S_m(x) = \frac{a_0 + a_m \cos(mx)}{2} + \sum_{k=1}^{m-1} (a_k \cos(kx) + b_k \sin(kx))$$

Anwendungen der schnellen Fouriertransformation:

- Interpolationen großer Datenmengen
  - Filter, Frequenzanalysen, Antennenstrahlungsdiagramme, Optik, digitale Fehler

Aber: große Anzahl von arithmetischen Operationen zur Bestimmung der Koeffizienten

Rechenaufwand für $2m$ Datenpunkte: $(2m)^2$ Multiplikationen und $(2m)^2$ Additionen

Steigerung der Effizienz: Einführung der schnellen Fouriertransformation (Fast Fourier Transformation, FFT) durch J. W. Cooley und J. W. Tuckey im Jahr 1965

Resultierender Rechenaufwand der FFT: $\mathcal{O}(m \log_2 m)$

Herleitung der FFT: Komplexe Darstellung der Fourier-Reihe (analog zu Gleichung (3.23)):
Mit $\cos(kx) = \frac{1}{2}(e^{ikx} + e^{-ikx})$ und $\sin(kx) = \frac{1}{2i}(e^{ikx} - e^{-ikx}) = \frac{i}{2}(e^{-ikx} - e^{ikx})$

$$\begin{aligned} g(x) &= \frac{1}{2}a_0 + \sum_{k=1}^{m} (a_k \cos(kx) + b_k \sin(kx)) \\ &= \frac{1}{2}a_0 + \sum_{k=1}^{m} \left[\left(\frac{a_k - ib_k}{2} \right) e^{ikx} + \left(\frac{a_k + ib_k}{2} \right) e^{-ikx} \right] \end{aligned}$$

Mit $c_k \coloneqq a_k - ib_k$; komplexe Darstellung der diskreten Fourier-Reihe:

$$g(x) = \sum_{k=-m}^{m} c_k e^{ikx}$$

Falls c_k bekannt:

$$a_0 = 2 \cdot c_0; \quad a_k = 2 \cdot \Re(c_k); \quad b_k = -2 \cdot \Im(c_k) \quad \text{für alle } k = 1, 2, \ldots m$$

Numerische Berechnung der komplexen Fourier-Koeffizienten mit $N = 2m$:

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j) \cdot e^{-ikx_j} \equiv \sum_{j=0}^{N-1} f_j \cdot \omega_N^{kj} \quad \text{mit k = 0, 1, \ldots n} \tag{3.24}$$

und den Definitionen

$$\begin{aligned} f_j &\equiv \frac{1}{N} f(x_j) = y_j; \\ x_j &\equiv \frac{2\pi j}{N} \quad \text{für alle } j = 0, 1, \ldots N-1); \\ \omega_N &\equiv e^{\frac{-2\pi i}{N}} \end{aligned}$$

Potenzen $\omega_N^j = z \quad (j = 0, 1, 2, \ldots N-1)$ genügen $z^N = 1$
Durch „N-te Einheitswurzel“ $e^{-2\pi i} = 1$ gilt:

$$\omega_N^N = 1, \quad \omega_N^{N+1} = \omega_N^1, \quad \omega_N^{N+2} = \omega_N^2 \quad \ldots$$

Berechnung der Summe (3.24) der Länge $2m$ durch Transformation auf Summen der Länge $\frac{N}{2} = m$

a) Für alle Koeffizienten c_k mit geradem Index ($k = 2\ell$)

$$\begin{aligned} c_{2\ell} &= \sum_{j=0}^{2m-1} f_j \omega_N^{2\ell j} = \sum_{j=0}^{m-1} \left[f_j \omega_N^{2j\ell} + f_{m+j} \omega_N^{2\ell(m+j)} \right] \\ &= \sum_{j=0}^{m-1} (f_j + f_{m+j}) \omega_N^{2\ell j} \end{aligned}$$

mit $\omega_N^{2\ell(m+j)} = \omega_N^{2\ell m} \cdot \omega_N^{2\ell j} = \omega_N^{2\ell j}$
Mittels $y \coloneqq f_j + f_{m+j}$ für alle $j = 0, 1, \ldots m-1$ und $\omega_N^2 = \omega_m$:

$$c_{2\ell} = \sum_{j=0}^{m-1} y_j \cdot \omega_m^{\ell j}$$

b) Für alle Koeffizienten c_k mit ungeradem Index ($k = 2\ell + 1$)

$$c_{2\ell+1} = \sum_{j=0}^{2m-1} f_j \omega_N^{(2\ell+1)j} = \sum_{j=0}^{m-1} \left[(f_j - f_{j+m}) \omega_N^j \right] \omega_N^{2\ell j}$$

Mittels

$$y_{m+j} = (f_j - f_{m+j}) \cdot \omega_N^j \quad \text{für alle } j = 0, 1, 2, \ldots m-1 \quad \text{und} \quad \omega_N^2 = \omega_m$$

$$c_{2\ell+1} = \sum_{j=0}^{m-1} y_{m+j} \cdot \omega_m^{\ell j} \quad \text{mit Länge } m = \frac{N}{2}$$

Die Reduzierung mittels a) und b), d. h. die Zurückführung einer diskreten komplexen Fourier-Transformation auf jeweils zwei diskrete komplexe Fourier-Transformationen der halben Länge, lässt sich fortsetzen, wenn N eine Potenz von 2, bzw. $N = 2^p$ mit natürlicher Zahl p ist.

Interpretation der FFT: p-malige Anwendung der Reduzierung
Jeder Reduktionsschritt benötigt $N/2$ komplexe Multiplikationen:

$$\frac{N}{2} \cdot p \text{ mit } p = \log_2 N$$

FFT-Rechenaufwand: $O(N \log_2 N)$

- $\log_2(N)$ Unterteilungsschritte mit $O(\frac{N}{2})$ Rechenoperationen

Effiziente Auswertungsmethoden:

- Butterfly-Schema
- Radix 4-Algorithmus
- Winograd-Algorithmus

Aber: Vorher Umsortierung der Daten f_j (auch für Unter-FFTs)

Implementierungen:

- „Fastest Fourier Transform in the West“ (FFTW, www.fftw.org)
- Numpy: numpy.fft.fft( )

Anwendungen:

- Datenformate JPEG und MPEG
- in Simulationen (Berechnung langreichweitiger Kräfte)

4 Datenanalyse und Fehlerrechnung

4.1 Methode der kleinsten Quadrate

Ziel der Methode:
Bestimmung der optimalen Näherungsgeraden zur Interpolation einer Funktion, die die Werte y_i in x_i für alle $i = 1, 2, \ldots, n$ annimmt.

Ansatz:
Mittels Gleichung der linearen Approximation $f(x) = a \cdot x + b$ soll Abweichung der Betragsfunktion minimal sein:

$$E_1(a, b) = \sum_{i=1}^{n} |y_i - (a \cdot x_i + b)| = \min$$

$E_1(a, b) \ldots$ absolute Abweichung

Minimierung einer Funktion mit zwei Variablen:
Partielle Ableitungen werden gleich 0 gesetzt und die resultierenden Gleichungen werden gelöst:

$$\frac{\partial}{\partial a} \sum_{i=1}^{n} |y_i - (a \cdot x_i + b)| = 0, \quad \frac{\partial}{\partial b} \sum_{i=1}^{n} |y_i - (a \cdot x_i + b)| = 0$$

Problem: Betragsfunktion ist an der Stelle 0 nicht differenzierbar (nicht stetig)

Ausweg: Totaler Fehler der Methode der kleinsten Quadrate

$$E_2(a, b) = \sum_{i=1}^{n} [y_i - (a \cdot x_i + b)]^2$$

Weiterhin: Minimierung von $E_2(a, b)$ bzgl. a, b:

$$\frac{\partial}{\partial a} E_2(a, b) = 2 \cdot \sum_{i=1}^{n} (y_i - a \cdot x_i - b) \cdot (-x_i) = 0, \quad \frac{\partial}{\partial b} E_2(a, b) = 2 \cdot \sum_{i=1}^{n} (y_i - a \cdot x_i - b) \cdot (-1) = 0$$

Die Normalgleichungen ergeben sich durch

$$a \cdot \sum_{i=1}^{n} x_i^2 + b \cdot \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i \cdot y_i$$
$$a \cdot \sum_{i=1}^{n} x_i + b \cdot n = \sum_{i=1}^{n} y_i$$

welche nach a, b aufgelöst werden können.

Koeffizienten in der Methode der kleinsten Quadrate

Die lineare Lösung nach der Methode der kleinsten Quadrate einer gegebenen Wertemenge (x_i, y_i) für $i = 1, 2, \ldots n$ besitzt die Form $y = a \cdot x + b$, wenn

$$a = \frac{n\left(\sum_{i=1}^{n} x_i y_i\right) - \left(\sum_{i=1}^{n} x_i\right) \cdot \left(\sum_{i=1}^{n} y_i\right)}{n\left(\sum_{i=1}^{n} x_i^2\right) - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$b = \frac{\left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i\right) - \left(\sum_{i=1}^{n} x_i y_i\right) \cdot \left(\sum_{i=1}^{n} x_i\right)}{n\left(\sum_{i=1}^{n} x_i^2\right) - \left(\sum_{i=1}^{n} x_i\right)^2}$$

Anstatt linearer Näherung:
Vorgehen der Minimierung von E_2 bei Polynom $P_n(x) = \sum_{k=0}^{n} a_k \cdot x^k$ vom Grad $n < m-1$ ist äquivalent.
Dementsprechend:
Bestimmung der Koeffizienten mittels Minimierung von $E_2 = \sum_{i=1}^{n} (y_i - P_n(x_i))^2$ und

$$\frac{\partial E}{\partial a_j} = 0 \quad \text{für jedes } j = 0, 1, \ldots n$$

ergibt $n + 1$ Normalgleichungen und $n + 1$ Unbekannte a_j.

Beispiel:
Methode der kleinsten Quadrate zur Koeffizientenberechnung der diskreten Fouriertransformation (s. Abschnitt 3.6.1):
Um die Konstanten a_k für $k = 0, 1, \ldots n$ und $b_k \equiv a_{k+n}$ für $k = 1, 2, \ldots n-1$ in der Summation (Gleichung (3.15))

$$S_n(x) = \frac{a_0}{2} + a_n \cos(nx) + \sum_{k=1}^{n-1} (a_k \cos(kx) + b_k \sin(kx))$$

zu erhalten, wird die Summe nach der Methode der kleinsten Quadrate

$$E_2(a_0, \ldots a_n, b_1, \ldots b_{n-1}) = \sum_{j=0}^{2m-1} [y_j - S_n(x_j)]^2$$

minimiert.
Mittels

$$E(a_0, \dots a_n, b_1, \dots b_{n-1}) = \sum_{j=0}^{2m-1} (y_j - S_n(x_j))^2$$

$$\frac{\partial E}{\partial a_k}(a_0, a \dots a_n, b_1, \dots b_{n-1}) = 0$$

$$\frac{\partial E}{\partial b_k}(a_0, a \dots a_n, b_1, \dots b_{n-1}) = 0$$

ergeben sich die Koeffizienten in der diskreten Fourierreihe mit festen Stützstellen zu

$$a_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \cdot \cos(kx_j) \quad \text{für alle k=0, 1, \dots n}$$

und

$$b_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \cdot \sin(kx_j) \quad \text{für alle k=0, 1, \dots n-1}$$

analog zu Gleichungen (3.21) und (3.22).

5 Literaturverzeichnis

[1] Böhm, Jan M. ; Hoock, Claudia ; Popper, Karl R.: *Karl Poppers kritischer Rationalismus heute: zur Aktualität kritisch-rationaler Wissenschaftstheorie.* Tübingen, Deutschland : Mohr Siebeck, 2002

[2] Frenkel, Daan ; Smit, Berend: *Understanding Molecular Simulation: From Algorithms to Applications.* Orlando, FL, USA : Academic Press, Inc., 1996

[3] Verlet, Loup: Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. In: *Phys. Rev.* 159 (1967), Nr. 1, S. 98