Numerische Mathematik für Naturwissenschaftler und Ingenieure

Mathematisches Institut
Fakultät für Mathematik und Physik
Universität Bayreuth

Inhaltsverzeichnis

Kapitel 1

Lineare Gleichungssysteme

Algorithmen zur Lösung linearer Gleichungssysteme bilden die Basis für viele Anwendungen der Numerik und stehen daher traditionell am Anfang vieler Numerik Vorlesungen. Ausführlich aufgeschrieben besteht das Problem darin, Zahlen $x_1, \ldots, x_n \in \mathbb{R}$ zu bestimmen, für die das Gleichungssystem

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & = & b_1 \\ & & & & & & \vdots & & \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \ldots & + & a_{nn}x_n & = & b_n \end{array} \tag{1.1}$$

erfüllt ist. Die ausführliche Schreibweise in (1.1) ist etwas unhandlich, weswegen wir lineare Gleichungsysteme üblicherweise in Matrix–Form schreiben werden, nämlich als

$$Ax = b, \tag{1.2}$$

mit

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}. \tag{1.3}$$

Diese Schreibweise hat nicht nur den Vorteil, dass man — wenn man die Konvention (1.3) einmal kennt — ein Gleichungssystem viel kürzer aufschreiben kann, es wird sich auch zeigen, dass gewisse Eigenschaften der Matrix A entscheiden, was für ein Verfahren zur Lösung von (1.2) sinnvollerweise eingesetzt werden kann.

Einige kurze Bemerkungen zur Notation: Wir werden Matrizen typischerweise mit großen Buchstaben bezeichnen (z.B. A) und Vektoren mit kleinen (z.B. b). Ihre Einträge werden wir mit indizierten Kleinbuchstaben bezeichnen, wie in (1.3). Mit einem hochgestellten „T“ bezeichnen wir transponierte Matrizen und Vektoren, für A und x aus (1.3) also z.B.

$$x = (x_1, \ldots, x_n)^T, \quad A^T = \begin{pmatrix} a_{11} & a_{21} & \ldots & a_{n1} \\ \vdots & \vdots & & \vdots \\ a_{1n} & a_{2n} & \ldots & a_{nn} \end{pmatrix}.$$

Da die Anzahl n der Gleichungen in (1.1) gleich der Anzahl der Unbekannten $x_1, \ldots, x_n$ ist, ist A in (1.2) eine quadratische Matrix der Dimension $n \times n$. Für quadratische Matrizen

ist aus der linearen Algebra bekannt, dass genau dann eine eindeutige Lösung von (1.2) existiert, wenn die Matrix invertierbar ist. Wir werden in diesem Kapitel immer annehmen, dass dies der Fall ist.

Tatsächlich kommt es eher selten vor, dass eine natur– oder ingenieurwissenschaftliche Anwendung *direkt* auf die Lösung eines linearen Gleichunssystems führt. Um zu zeigen, dass das Problem trotzdem wichtig ist, wollen wir zu Beginn dieses Kapitels kurz einige Algorithmen aus der Numerik betrachten, die auf die Lösung eines linearen Gleichungssystems führen.

1.1 Anwendungen linearer Gleichungssysteme

1.1.1 Ausgleichsrechnung

Das erste unserer Anwendungsbeispiele ist für viele praktische Zwecke besonders wichtig, weswegen wir es etwas genauer untersuchen wollen.

Nehmen wir an, wir haben ein Experiment durchgeführt, bei dem wir für verschiedene Eingabewerte $t_1, t_2, \ldots, t_k$ Messwerte $m_1, m_2, \ldots, m_k$ erhalten. Aufgrund von theoretischen Überlegungen (z.B. aufgrund eines zugrundeliegendes physikalisches Gesetzes), kennt man eine Funktion $f(t)$, für die $f(t_i) = m_i$ gelten sollte. Diese Funktion wiederum hängt aber nun von unbekannten Parametern $x_1, \ldots, x_n$ ab; wir schreiben $f(t; x)$ für $x = (x_1, \ldots, x_n)^T$, um dies zu betonen. Z.B. könnte $f(t; x)$ durch

$$f(t; x) = x_1 + x_2 t \quad \text{oder} \quad f(t; x) = x_1 + x_2 t + x_3 t^2$$

gegeben sein. Im ersten Fall beschreibt die gesuchte Funktion eine Gerade, im zweiten eine (verallgemeinerte) Parabel. Wenn wir annehmen, dass die Funktion f das Experiment wirklich exakt beschreibt und keine Messfehler vorliegen, so könnten wir die Parameter x_i durch Lösen des (im Allgemeinen nichtlinearen) Gleichungssystems

$$\begin{aligned} f(t_1; x) &= m_1 \\ &\vdots \\ f(t_k; x) &= m_k \end{aligned} \tag{1.4}$$

nach x bestimmen. In vielen praktischen Fällen ist dieses Gleichungssystem linear, so z.B. in den zwei obigen Beispielen, in denen sich (1.4) zu $\widetilde{A}x = m$ mit

$$\widetilde{A} = \begin{pmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_k \end{pmatrix} \quad \text{bzw.} \quad \widetilde{A} = \begin{pmatrix} 1 & t_1 & t_1^2 \\ \vdots & \vdots & \vdots \\ 1 & t_k & t_k^2 \end{pmatrix} \quad \text{und} \quad m = \begin{pmatrix} m_1 \\ \vdots \\ m_k \end{pmatrix}$$

ergibt. Diese linearen Gleichungssysteme haben k Gleichungen (eine für jedes Wertepaar (t_i, m_i)) und n Unbekannte (nämlich gerade die unbekannten Parameter x_i), wobei k üblicherweise sehr viel größer als n ist. Man sagt, dass das Gleichungsystem *überbestimmt* ist. Da Messwerte eines Versuchs praktisch immer mit Fehlern behaftet sind, ist es sicherlich zu optimistisch, anzunehmen, dass das Gleichungssystem $\widetilde{A}x = m$ lösbar ist (überbestimmte

Gleichungssysteme haben oft keine Lösung!). Statt also den (vermutlich vergeblichen) Versuch zu machen, eine exakte Lösung x dieses Systems zu finden, wollen wir probieren, eine möglichst gute Näherungslösung zu finden, d.h., wenn $\widetilde{A}x = m$ nicht lösbar ist wollen wir zumindest ein x finden, so dass $\widetilde{A}x$ möglichst nahe bei m liegt. Dazu müssen wir ein Kriterium für „möglichst nahe“ wählen, das sowohl sinnvoll ist als auch eine einfache Lösung zulässt. Hier bietet sich das sogenannte *Ausgleichsproblem* (auch *Methode der kleinsten Quadrate* genannt) an:

Finde $x = (x_1, \ldots, x_n)^T$, so dass $\varphi(x) := \|m - \widetilde{A}x\|^2$ minimal wird.

Hierbei bezeichnet $\|y\|$ die euklidische Norm eines Vektors $y \in \mathbb{R}^n$, also

$$\|y\| = \sqrt{\sum_{i=1}^{n} y_i^2}.$$

Um die Funktion φ zu minimieren, berechnet man den Gradienten $\nabla\varphi$ und setzt diesen gleich Null. Da man nachrechnen kann, dass die zweite Ableitung positiv definit ist, ist jede Nullstelle des Gradienten $\nabla\varphi$ ein Minimum von φ. Mit etwas Rechnung erhält man aus diesen Überlegungen, dass ein Vektor x die Funktion φ genau dann minimiert, wenn die sogennanten *Normalengleichungen* $\widetilde{A}^T\widetilde{A}x = \widetilde{A}^T m$ erfüllt ist. Das Ausgleichsproblem wird also wie folgt gelöst:

$$\text{löse} \quad Ax = b \quad \text{mit} \quad A = \widetilde{A}^T\widetilde{A} \text{ und } b = \widetilde{A}^T m.$$

Das zunächst recht kompliziert scheinende Minimierungsproblem für φ wird also auf die Lösung eines linearen Gleichungssystems zurückgeführt.

1.1.2 Diskrete L_2 Approximation

Auch bei diesem Problem geht es darum, Parameter einer Funktion zu bestimmen, um eine möglichst gute Approximation zu erhalten. Während wir beim Ausgleichsproblem aber den Abstand zwischen den Funktionswerten $f(t_j)$ und den Messwerten m_j klein halten wollten, geht es nun darum, Parameter so zu wählen, dass das Integral einer Funktion möglichst gut approximiert wird. Sei dazu $z : [c, d] \to \mathbb{R}$ eine integrierbare Funktion auf einem Intervall $[c, d]$. Für vorgegebene integrierbare Funktionen $v_1, \ldots, v_n : [c, d] \to \mathbb{R}$ wollen wir einen Parametervektor $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ bestimmen, so dass das Integral

$$\varphi(x) := \int_c^d |z(t) - \sum_{i=1}^{n} x_i v_i(t)|^2 dt$$

minimal wird.

Wiederum berechnet man hier den Gradienten $\nabla\varphi$ und leitet so geeignete *Normalengleichungen* her, die auch hier auf ein lineares Gleichungssystem $Ax = b$ führen, wobei die Einträge a_{ij} von A und b_i von b gegeben sind durch

$$a_{ij} = \int_c^d v_j(t)v_i(t)dt \quad \text{und} \quad b_i = \int_c^d z(t)v_i(t)dt.$$

1.1.3 Randwertaufgaben gewöhnlicher Differentialgleichungen

Differentialgleichungen sind Gleichungen, bei denen eine unbekannte differenzierbare Funktion dadurch charakterisiert ist, dass ihre Werte mit ihren Ableitungen in Beziehung gesetzt ist. Wir werden diese später in der Vorlesung genauer betrachten, wollen hier aber kurz ein spezielles Problem skizzieren, das wiederum auf ein lineares Gleichungssystem führt.

Gegeben sei ein Intervall $[c, d]$, reelle Zahlen y_c, y_d und eine stetige Funktion $g : [c, d] \to \mathbb{R}$. Gesucht ist eine differenzierbare Funktion $y : [c, d] \to \mathbb{R}$, welche die Gleichungen

$$\begin{aligned} y''(t) + y'(t) + y(t) &= g(t) \\ y(c) &= y_c \\ y(d) &= y_d \end{aligned}$$

erfüllt. Zur Lösung des Problems kann man die *konsistente Differenzenapproximation* verwenden. Hierbei zerlegt man das Intervall $[c, d]$ in gleichlange Teilintervalle

$$c = t_0 < t_1 < \ldots < t_N = d$$

mit $t_i - t_{i-1} = h := (d - c)/N$ für ein $N \in \mathbb{N}$, also $t_i = c + i(d - c)/N$, und ersetzt die Ableitungen y' und y'' an den Punkten t_i durch Differenzenquotienten

$$y'(t_i) \approx \frac{y(t_{i+1}) - y(t_{i-1})}{2h}, \quad y''(t_i) \approx \frac{y(t_{i+1}) - 2y(t_i) + y(t_{i-1})}{h^2}.$$

Mit der Schreibweise $\eta_i = y(t_i)$ und $g_i = g(t_i)$ erhalten wir damit aus den obigen Gleichungen für die inneren Gitterpunkte die Differenzengleichungen

$$\frac{\eta_{i+1} - 2\eta_i + \eta_{i-1}}{h^2} + \frac{\eta_{i+1} - \eta_{i-1}}{2h} + \eta_i = g_i, \quad i = 1, \ldots, N-1$$

und für die Randpunkte die Gleichungen

$$\eta_0 = y_c, \quad \eta_N = y_d$$

aus den Randbedingungen. Insgesamt erhalten wir so das lineare Gleichungssystem

$$A_h \eta = b$$

mit

$$A_h = \begin{pmatrix} 1 & & & & \\ \frac{1}{h^2} - \frac{1}{2h} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} + \frac{1}{2h} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{h^2} - \frac{1}{2h} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} + \frac{1}{2h} \\ & & & & 1 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} y_c \\ g_1 \\ \vdots \\ g_{N-1} \\ y_d \end{pmatrix}.$$

Die Einträge $\eta_0, \ldots, \eta_n$ von η liefern dann eine Approximation der Funktionswerte $y(t_0), \ldots, y(t_N)$.

1.1.4 Weitere Anwendungen

Es gibt viele weitere Anwendungen in der Numerik, die auf die Lösung eines linearen Gleichungssystems führen. Die meisten davon gehen über den Rahmen dieser Vorlesung hinaus. Mindestens zwei weitere Anwendungen werden uns aber im weiteren Verlauf dieser Vorlesung noch begegnen, nämlich zum einen die Lösung *nichtlinearer* Gleichungssysteme mit dem *Newton–Verfahren*, bei dem eine Folge linearer Gleichungssysteme gelöst werden muss, und zum anderen die *Interpolation* von Kurven mittels *Splines*.

1.2 Das Gauß'sche Eliminationsverfahren

Wir werden nun ein erstes Verfahren zur Lösung linearer Gleichungssysteme kennen lernen. Das *Gauß'sche Eliminationsverfahren* ist ein sehr anschauliches Verfahren, das zudem recht leicht implementiert werden kann. Es beruht auf der einfachen Tatsache, dass ein lineares Gleichungssystem $Ax = b$ leicht lösbar ist, falls die Matrix A in *oberer Dreiecksform* vorliegt, d.h.,

$$A = \begin{pmatrix} a_{1\,1} & a_{1\,2} & \dots & a_{1\,n} \\ 0 & a_{2\,2} & \dots & a_{2\,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & a_{n\,n} \end{pmatrix}.$$

In diesem Fall kann man $Ax = b$ leicht mittels der rekursiven Vorschrift

$$x_n = \frac{b_n}{a_{n\,n}}, \quad x_{n-1} = \frac{b_{n-1} - a_{n-1\,n}x_n}{a_{n-1\,n-1}}, \dots, \quad x_1 = \frac{b_1 - a_{1\,2}x_2 - \dots - a_{1\,n}x_n}{a_{1\,1}}$$

oder, kompakt geschrieben,

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i\,j}x_j}{a_{i\,i}}, \qquad i = n,\, n-1, \dots,\, 1 \tag{1.5}$$

(mit der Konvention $\sum_{j=n+1}^{n} a_{i\,j}x_j = 0$) lösen. Dieses Verfahren wird als *Rückwärtseinsetzen* bezeichnet.

Die Idee des Gauß'schen Eliminationsverfahrens liegt nun darin, das Gleichungssystem $Ax = b$ in ein Gleichungssystem $\widetilde{A}x = \tilde{b}$ umzuformen, so dass die Matrix $\widetilde{A}$ in oberer Dreiecksform vorliegt. Wir wollen dieses Verfahren zunächst an einem Beispiel veranschaulichen.

Beispiel 1.1 Gegeben sei das lineare Gleichungssystem (1.2) mit

$$A = \begin{pmatrix} 1 & 5 & 6 \\ 7 & 9 & 6 \\ 2 & 3 & 4 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 29 \\ 43 \\ 20 \end{pmatrix}.$$

Um die Matrix A auf obere Dreiecksgestalt zu bringen, müssen wir die drei Einträge 7, 2 und 3 unterhalb der Diagonalen auf 0 bringen („eliminieren“). Wir beginnen mit der 2. Hierzu subtrahieren wir 2–mal die erste Zeile von der letzten und erhalten so

$$A_1 = \begin{pmatrix} 1 & 5 & 6 \\ 7 & 9 & 6 \\ 0 & -7 & -8 \end{pmatrix}$$

Das Gleiche machen wir mit dem Vektor b. Dies liefert

$$b_{1\,3} = \begin{pmatrix} 29 \\ 43 \\ -38 \end{pmatrix}.$$

Nun fahren wir fort mit der 7: Wir subtrahieren 7–mal die erste Zeile von der zweiten, sowohl in A_1 als auch in b_1, und erhalten

$$A_2 = \begin{pmatrix} 1 & 5 & 6 \\ 0 & -26 & -36 \\ 0 & -7 & -8 \end{pmatrix} \quad \text{und} \quad b_2 = \begin{pmatrix} 29 \\ -160 \\ -38 \end{pmatrix}.$$

Im dritten Schritt „eliminieren“ wir die -7, die jetzt an der Stelle der 3 steht, indem wir $7/26$–mal die zweite Zeile von der dritten subtrahieren:

$$A_3 = \begin{pmatrix} 1 & 5 & 6 \\ 0 & -26 & -36 \\ 0 & 0 & \frac{22}{13} \end{pmatrix} \quad \text{und} \quad b_3 = \begin{pmatrix} 29 \\ -160 \\ \frac{66}{13} \end{pmatrix}.$$

Hiermit sind wir fertig und setzen $\widetilde{A} = A_3$ und $\tilde{b} = b_3$. Rückwärtseinsetzen gemäß (1.5) liefert dann

$$x_3 = \frac{\frac{66}{13}}{\frac{22}{13}} = 3, \quad x_2 = \frac{-160 - 3 \cdot (-36)}{-26} = 2 \ \text{ und } \ x_1 = \frac{29 - 2 \cdot 5 - 3 \cdot 6}{1} = 1.$$

□

Wir formulieren den Algorithmus nun für allgemeine quadratische Matrizen A.

Algorithmus 1.2 (Gauß–Elimination)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n \times n}$ und ein Vektor $b \in \mathbb{R}^n$.

(0) Setze $i = n$ und $j = 1$ (Zeilen– und Spaltenindex des zu eliminierenden Eintrags)

(1) Subtrahiere $a_{i\,j}/a_{j\,j}$–mal die j–te Zeile von der i–ten Zeile:
Setze $\alpha := a_{i\,j}/a_{j\,j}$ und berechne

$$a_{i\,k} := a_{i\,k} - \alpha\, a_{j\,k} \ \text{ für } k = 1, \ldots, n, \quad b_i := b_i - \alpha\, b_j$$

(2) Falls $i \geq j + 2$ ist, setze $i := i - 1$ und fahre fort bei (1), sonst:
Falls $j \leq n - 2$ ist, setze $j := j + 1$ und $i := n$ und fahre fort bei (1), sonst:
Ende des Algorithmus

□

Leider kann es passieren, dass dieser Algorithmus zu keinem Ergebnis führt, obwohl das Gleichungssystem lösbar ist. Der Grund dafür liegt in Schritt (1), in dem durch das Diagonalelement a_{jj} geteilt wird. Hierbei wurde stillschweigend angenommen, dass dieses ungleich Null ist, was aber im Allgemeinen nicht der Fall sein muss. Glücklicherweise gibt es eine Möglichkeit, dieses zu beheben:

Nehmen wir an, dass wir Schritt (1) für gegebene Indizes i und j ausführen möchten und $a_{jj} = 0$ ist. Nun gibt es zwei Möglichkeiten: Im ersten Fall ist $a_{ij} = 0$. In diesem Fall brauchen wir nichts zu tun, da das Element a_{ij}, das auf 0 gebracht werden soll, bereits gleich 0 ist. Im zweiten Fall gilt $a_{ij} \neq 0$. In diesem Fall können wir die i–te und j–te Zeile der Matrix A sowie die entsprechenden Einträge im Vektor b vertauschen, wodurch wir $a_{jj} \neq 0$ erreichen und mit dem Algorithmus fortfahren können. Dieses Verfahren nennt man *Pivotierung* und der folgende Algorithmus bringt nun tatsächlich jedes lineare Gleichungssystem in Dreiecksform.

Algorithmus 1.3 (Gauß–Elimination mit Pivotierung)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n\times n}$ und ein Vektor $b \in \mathbb{R}^n$.

(0) Setze $i = n$ und $j = 1$ (Zeilen– und Spaltenindex des zu eliminierenden Eintrags)

(1a) Falls $a_{ij} = 0$ gehe zu (2), sonst:
Falls $a_{jj} = 0$ vertausche a_{jk} und a_{ik} für $k = 1, \ldots, n$ sowie b_i und b_j

(1b) Subtrahiere a_{ij}/a_{jj}–mal die j–te Zeile von der i–ten Zeile:
Setze $\alpha := a_{ij}/a_{jj}$ und berechne

$$a_{ik} := a_{ik} - \alpha\, a_{jk} \text{ für } k = 1, \ldots, n, \quad b_i := b_i - \alpha\, b_j$$

(2) Falls $i \geq j + 2$ ist, setze $i := i - 1$ und fahre fort bei (1b), sonst:
Falls $j \leq n - 2$ ist, setze $j := j + 1$ und $i := n$ und fahre fort bei (1a), sonst:
Ende des Algorithmus

□

Das Element a_{ij}, das in Schritt (1a) mit a_{jj} getauscht wird, nennt man *Pivotelement*. Tatsächlich kann es sinnvoll sein, hierfür nicht das „aktuelle“ Element a_{ij} zu verwenden, sondern ein anderes Element $a_{kj} \neq 0$, wodurch man u.U. ein besseres Verhalten des Algorithmus' erhalten kann, siehe auch Abschnitt 1.4 weiter unten.

1.3 Das Choleski Verfahren

Im Gauß–Verfahren haben wir die Matrix A auf obere Dreiecksform gebracht und zugleich alle dafür notwendigen Operationen auch auf den Vektor b angewendet. Es gibt eine alternative Möglichkeit, lineare Gleichungssysteme zu lösen, bei denen der Vektor b unverändert

bleibt. Hierzu wird die Matrix A in ein Produkt von zwei Matrizen L und R zerlegt, also $A = LR$, wobei R in oberer Dreiecksform und L in *unterer Dreiecksform*

$$L = \begin{pmatrix} l_{1\,1} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ l_{n-1\,1} & \dots & l_{n-1\,n-1} & 0 \\ l_{n\,1} & l_{n\,2} & \dots & l_{n\,n} \end{pmatrix}.$$

vorliegt. Man kann sich leicht überlegen, dass L genau dann in unterer Dreiecksform ist, wenn L^T in oberer Dreiecksform ist. Die Zerlegung $A = LR$ wird als *LR–Zerlegung* bezeichnet.

Um $Ax = b$ zu lösen, kann man dann $LRx = b$ wie folgt in zwei Schritten lösen: Zunächst löst man das Gleichungssystem $Ly = b$. Dies kann, ganz analog zum Rückwärtseinsetzen (1.5) durch *Vorwärtseinsetzen* geschehen:

$$y_i = \frac{b_i - \sum_{j=1}^{i-1} l_{i\,j} y_j}{l_{i\,i}}, \qquad i = 1, 2, \dots, n \tag{1.6}$$

Im zweiten Schritt löst man dann durch Rückwärtseinsetzen das Gleichungssystem $Rx = y$. Dann gilt

$$Ax = LRx = Ly = b,$$

womit das gewünschte System $Ax = b$ gelöst ist.

Tatsächlich lässt sich das Gauß'sche Eliminationsverfahren so abändern, dass damit eine LR–Zerlegung einer invertierbaren Matrix möglich ist — zumindest im Prinzip: Die Zeilenvertauschungen bei der Pivotierung machen dabei technische Probleme auf die wir hier nicht näher eingehen können. Wir wollen hier ein anderes Verfahren zur LR–Zerlegung betrachten, das *Choleski–Verfahren*. Dieses funktioniert allerdings nicht für allgemeine Matrizen, sondern für *symmetrische, positiv definite* Matrizen.

Definition 1.4 (i) Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt *symmetrisch*, falls $A^T = A$ ist.

(ii) Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt *positiv definit*, falls $\langle x, Ax\rangle > 0$ ist für alle Vektoren $x \in \mathbb{R}^n$ mit $x \neq (0 \dots 0)^T$.

Hierbei bezeichnet $\langle x, y\rangle$ das *Standard–Skalarprodukt* im $\mathbb{R}^n$, d.h., für Vektoren $x, y \in \mathbb{R}^n$ ist

$$\langle x, y\rangle = x^T y = \sum_{i=1}^{n} x_i y_i.$$

□

Diese Bedingung ist natürlich recht einschränkend; trotzdem ist sie in vielen Anwendungen erfüllt. So führt z.B. das Ausgleichsproblem i.A. auf ein Gleichungssystem mit symmetrischer und positiv definiter Matrix A.

Für solche Matrizen kann man zeigen, dass immer eine LR–Zerlegung existiert, bei der zusätzlich die Gleichung $R = L^T$ gilt. Es genügt also, die untere Dreiecksmatrix L zu berechnen. Dies macht der folgende Algorithmus.

Algorithmus 1.5 (Choleski–Verfahren) Gegeben sei eine Matrix $A \in \mathbb{R}^{n\times n}$.

(0) Setze $i = 1$ und $j = 1$ (Zeilen– und Spaltenindex des aktuellen Eintrags l_{ij} von L)

(1) (a) Falls $i > j$ setze

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}$$

(b) Falls $i = j$ setze

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$$

(c) Falls $i < j$ setze

$$l_{ij} = 0$$

(2) Falls $i \leq n-1$ ist, setze $i := i+1$ und fahre fort bei (1), sonst:
Falls $j \leq n-1$ ist, setze $j := j+1$ und $i := 1$ und fahre fort bei (1), sonst:
Ende des Algorithmus

□

Leider ist dieser Algorithmus nicht so anschaulich wie die Gauß–Elimination. Um zu erläutern, warum dieser Algorithmus funktioniert, schreiben wir die Gleichung $A = LL^T$ explizit auf und versuchen, nach L aufzulösen. Dies ist jedoch direkt für beliebige Dimensionen sehr unübersichtlich, weswegen wir per Induktion über die Dimension der Matrix A vorgehen. Wir betrachten zunächst 2×2 Matrizen. Hier ergibt sich

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{pmatrix} = \begin{pmatrix} l_{11}^2 & l_{11}l_{22} \\ l_{21}l_{11} & l_{21}^2 + l_{22}^2 \end{pmatrix}. \tag{1.7}$$

Daraus erhalten wir

$$\begin{aligned} l_{11} &= \sqrt{a_{11}} \\ l_{21} &= a_{21}/l_{11} \\ l_{22} &= \sqrt{a_{22} - l_{21}^2} \end{aligned}$$

was genau den Berechnungen im Algorithmus 1.5 für $i = 1, 2$ und $j = 1, 2$ entspricht. Für größere Matrizen funktioniert die Rechnung im Prinzip auf die gleiche Art und Weise. Für $i = 2, \ldots, n$ schreiben wir

$$A_i = \begin{pmatrix} a_{11} & \ldots & a_{1i} \\ \vdots & & \vdots \\ a_{i1} & \ldots & a_{ii} \end{pmatrix}.$$

Falls A symmetrisch und positiv definit ist, so hat auch jede der Matrizen A_i, $i = 2, \ldots, n$ diese Eigenschaft. Für $i = 2$ haben wir gesehen, wie wir aus (1.7) die zu A_2 gehörige untere Dreiecksmatrix L_2 berechnen; es ergeben sich gerade die Gleichungen aus Algorithmus 1.5.

Für größere i können wir per Induktion vorgehen: Wir berechnen L_i unter der Voraussetzung, dass wir L_{i-1} bereits kennen. Dazu zerlegen wir die symmetrische $i \times i$ Matrix A_i mittels

$$A_i = \begin{pmatrix} A_{i-1} & \bar{a}_i \\ \bar{a}_i^T & a_{i\,i} \end{pmatrix}$$

in die $(i-1) \times (i-1)$ Matrix A_{i-1}, den $i-1$–dimensionalen Vektor $\bar{a}_i = (a_{i\,1} \ldots a_{i\,i-1})^T$ sowie die reelle Zahl $a_{i\,i}$. Da wir annehmen, dass wir L_{i-1} mit $L_{i-1}L_{i-1}^T = A_{i-1}$ kennen, können wir die untere Dreiecksmatrix L_i analog zu A_i als

$$L_i = \begin{pmatrix} L_{i-1} & 0 \\ \bar{l}_i^T & l_{i\,i} \end{pmatrix}$$

mit $\bar{l}_i = (l_{i\,1} \ldots l_{i\,i-1})^T$ schreiben, und erhalten aus $L_i L_i^T = A_i$ die Gleichung

$$\begin{pmatrix} A_{i-1} & \bar{a}_i \\ \bar{a}_i^T & a_{i\,i} \end{pmatrix} = \begin{pmatrix} L_{i-1} & 0 \\ \bar{l}_i^T & l_{i\,i} \end{pmatrix} \begin{pmatrix} L_{i-1}^T & \bar{l}_i \\ 0 & l_{i\,i} \end{pmatrix} = \begin{pmatrix} L_{i-1}L_{i-1}^T & L_{i-1}\bar{l}_i \\ (L_{i-1}\bar{l}_i)^T & \bar{l}_i^T\bar{l}_i + l_{i\,i}^2 \end{pmatrix}. \tag{1.8}$$

Bestimmt man nun die Einträge im Vektor $\bar{l}_i$ durch Vorwärtseinsetzen aus dem Gleichungssystem $L_{i-1}\bar{l}_i = \bar{a}_i$, so erhält man gerade die entsprechenden Gleichungen für $l_{i,j}$, $j = 1, \ldots, i-1$ aus Algorithmus 1.5. Löst man dann noch die Gleichung $\bar{l}_i^T\bar{l}_i + l_{i\,i}^2 = a_{i\,i}$, so ergibt sich gerade die Gleichung für $j = i$ in Algorithmus 1.5. Indem wir also die obige Rechnung zunächst für $i = 2$ mittels (1.7) und dann sukzessive für $i = 3, \ldots, n$ mittels (1.8) durchführen, erhalten wir gerade den Choleski Algorithmus.

1.4 Fehlerabschätzungen

Computer können nicht alle (unendlich vielen) reellen Zahlen darstellen, deswegen werden alle Zahlen intern gerundet, damit sie in die endliche Menge der *maschinendarstellbaren Zahlen* passen. Hierdurch entstehen *Rundungsfehler*. Selbst wenn sowohl die Eingabewerte als auch das Ergebnis eines Algorithmus maschinendarstellbare Zahlen sind, können solche Fehler auftreten, denn auch die (möglicherweise nicht darstellbaren) Zwischenergebnisse eines Algorithmus werden gerundet. Aufgrund von solchen Rundungsfehlern aber auch wegen Eingabe– bzw. Messfehlern in den vorliegenden Daten, wird durch einen Algorithmus üblicherweise nicht die Lösung x des linearen Gleichungssystems

$$Ax = b,$$

sondern die Lösung $\tilde{x}$ eines „benachbarten“ oder „gestörten“ Gleichungssystems

$$A\tilde{x} = b + \Delta b$$

berechnet. Der Vektor Δb heißt das *Residuum* oder der *Defekt* der Näherungslösung $\tilde{x}$. Den Vektor $\Delta x = \tilde{x} - x$ nennen wir den *Fehler* der Näherungslösung $\tilde{x}$. Das Ziel dieses Abschnittes ist es, aus der Größe des Residuum $\|\Delta b\|$ auf die Größe des Fehlers $\|\Delta x\|$ zu schließen. Insbesondere wollen wir untersuchen, wie *sensibel* die Größe $\|\Delta x\|$ von $\|\Delta b\|$ abhängt, d.h. ob kleine Residuen $\|\Delta b\|$ große Fehler $\|\Delta x\|$ hervorrufen können. Diese Analyse ist unabhängig von dem verwendeten Lösungsverfahren, da wir hier nur das Gleichungssystem selbst und kein explizites Verfahren betrachten.

Um diese Analyse durchzuführen brauchen wir das Konzept der *Matrixnorm*. Man kann Matrixnormen ganz allgemein definieren; für unsere Zwecke ist aber der Begriff der *induzierten Matrixnorm* ausreichend. Wir erinnern zunächst daran, dass es für Vektoren $x \in \mathbb{R}^n$ viele gebräuchliche Normen gibt. In dieser Vorlesung verwenden wir üblicherweise die *euklidische Norm* oder *2–Norm*

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2},$$

welche wir meistens einfach mit $\|x\|$ bezeichnen. Weitere Normen sind die *1–Norm*

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

und die *Maximums–* oder *∞–Norm*

$$\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|.$$

Für alle Normen im $\mathbb{R}^n$ kann man eine zugehörige *induzierte Matrixnorm* definieren.

Definition 1.6 Sei $\mathbb{R}^{n\times n}$ die Menge der $n \times n$–Matrizen und sei $\|\cdot\|_p$ eine Vektornorm im $\mathbb{R}^n$. Dann definieren wir für $A \in \mathbb{R}^{n\times n}$ die zu $\|\cdot\|_p$ gehörige *induzierte Matrixnorm* $\|A\|_p$ als

$$\|A\|_p := \max_{\substack{x\in\mathbb{R}^n \\ \|x\|_p=1}} \|Ax\|_p.$$

□

Da im Allgemeinen keine Verwechslungsgefahr besteht, bezeichnen wir die Vektornormen und die von ihnen induzierten Matrixnormen mit dem gleichen Symbol.

Man kann nachrechnen, dass die zu den obigen Vektornormen gehörigen induzierten Matrixnormen gegeben sind durch

$$\begin{aligned}
\|A\|_1 &= \max_{j=1,\ldots,n} \textstyle\sum_{i=1}^{n} |a_{ij}| && \text{(Spaltensummennorm)} \\
\|A\|_\infty &= \max_{i=1,\ldots,n} \textstyle\sum_{j=1}^{n} |a_{ij}| && \text{(Zeilensummennorm)} \\
\|A\|_2 &= \sqrt{\rho(A^T A)} && \text{(Spektralnorm),}
\end{aligned}$$

wobei $\rho(B)$ den maximalen Eigenwert einer symmetrischen Matrix B bezeichnet. Basierend auf einer Matrixnorm können wir die *Kondition* einer invertierbaren Matrix definieren.

Definition 1.7 Für eine gegebene Matrixnorm $\|\cdot\|_p$ ist die *Kondition* einer invertierbaren Matrix A (bzgl. $\|\cdot\|_p$) definiert durch

$$\mathrm{cond}_p(A) := \|A\|_p \|A^{-1}\|_p.$$

□

Wenn wir das Verhältnis zwischen dem Fehler Δx und dem Residuum Δb betrachten, können wir entweder die *absoluten Größen* dieser Werte, also $\|\Delta x\|_p$ und $\|\Delta b\|_p$ betrachten, oder, was oft sinnvoller ist, die *relativen Größen* $\|\Delta x\|_p/\|x\|_p$ und $\|\Delta b\|_p/\|b\|_p$. Der folgende Satz zeigt, wie man den Fehler durch das Residuum anschätzen kann.

Satz 1.8 Sei $\|\cdot\|_p$ eine Vektornorm mit zugehöriger (und gleich bezeichneter) induzierter Matrixnorm. Sei $A \in \mathbb{R}^{n\times n}$ eine gegebene invertierbare Matrix und b, $\Delta b \in \mathbb{R}^n$ gegebene Vektoren. Seien x, $\tilde{x} \in \mathbb{R}^n$ die Lösungen der linearen Gleichungssysteme

$$Ax = b \quad \text{und} \quad A\tilde{x} = b + \Delta b.$$

Dann gelten für den Fehler $\Delta x = \tilde{x} - x$ die Abschätzungen

$$\|\Delta x\|_p \leq \|A^{-1}\|_p \|\Delta b\|_p \tag{1.9}$$

und

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq \mathrm{cond}_p(A) \frac{\|\Delta b\|_p}{\|b\|_p}. \tag{1.10}$$

Beweis: Seien $C \in \mathbb{R}^{n\times n}$ und $y \in \mathbb{R}^n$ eine beliebige Matrix und ein beliebiger Vektor. Dann gilt

$$\|Cy\|_p = \left\| C \frac{y}{\|y\|_p} \right\|_p \|y\|_p \leq \max_{\substack{x\in\mathbb{R}^n \\ \|x\|_p=1}} \|Cx\|_p \|y\|_p = \|C\|_p \|y\|_p. \tag{1.11}$$

Schreiben wir $\tilde{x} = x + \Delta x$, und ziehen die Gleichung

$$Ax = b$$

von der Gleichung

$$A(x + \Delta x) = b + \Delta b$$

ab, so erhalten wir

$$A\Delta x = \Delta b.$$

Weil A invertierbar ist, können wir die Gleichung auf beiden Seiten von links mit A^{-1} multiplizieren und erhalten so

$$\Delta x = A^{-1}\Delta b.$$

Daraus folgt

$$\|\Delta x\|_p = \|A^{-1}\Delta b\|_p \leq \|A^{-1}\|_p \|\Delta b\|_p,$$

wobei wir (1.11) mit $C = A^{-1}$ und $y = \Delta b$ benutzt haben. Dies zeigt (1.9). Aus (1.11) mit $C = A$ und $y = x$ folgt

$$\|b\|_p = \|Ax\|_p \leq \|A\|_p \|x\|_p,$$

und damit

$$\frac{1}{\|x\|_p} \leq \frac{\|A\|_p}{\|b\|_p}.$$

Wenn wir erst diese Ungleichung und dann (1.9) anwenden, erhalten wir

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq \frac{\|A\|_p \|\Delta_x\|_p}{\|b\|_p} \leq \frac{\|A\|_p \|A^{-1}\|_p \|\Delta b\|_p}{\|b\|_p} = \mathrm{cond}_p(A) \frac{\|\Delta b\|_p}{\|b\|_p},$$

also (1.10). □

Für Matrizen, deren Kondition $\mathrm{cond}_p(A)$ groß ist, können sich kleine Fehler im Vektor b (bzw. Rundungsfehler im Verfahren) zu großen Fehlern im Ergebnis x verstärken. Man spricht in diesem Fall von *schlecht konditionierten* Matrizen.

Ein wichtiges Kriterium beim Entwurf eines Lösungsverfahrens ist es, dass das Verfahren auch für schlecht konditionierte Matrizen noch zuverlässig funktioniert. Beim Gauß–Verfahren kann z.B. die Auswahl der Pivot–Elemente so gestaltet werden, dass sich schlechte Konditionierung weniger stark auswirkt. Eine andere Strategie ist die sogannante *Präkonditionierung*, bei der eine Matrix $P \in \mathbb{R}^{n \times n}$ gesucht wird, für die die Kondition von PA kleiner als die von A ist, so dass dann das besser konditionierte Problem $PAx = Pb$ gelöst werden kann.

1.5 Aufwandsabschätzungen

Ein wichtiger Aspekt bei der Analyse numerischer Verfahren ist es zu untersuchen, wie lange diese Verfahren in der Regel benötigen, um zu dem gewünschten Ergebnis zu kommen. Da dies natürlich entscheidend von der Leistungsfähigkeit des verwendeten Computers abhängt, schätzt man nicht direkt die Zeit ab, sondern die Anzahl der Rechenoperationen, die ein Algorithmus benötigt. Da hierbei die *Gleitkommaoperationen*, also Addition, Multiplikation etc. von reellen Zahlen, die mit Abstand zeitintensivsten Operationen sind, beschränkt man sich in der Analyse üblicherweise auf diese.

Die Verfahren, die wir bisher betrachtet haben, liefern nach endlich vielen Schritten ein Ergebnis, wobei die Anzahl der Operationen von der Dimension n der Matrix abhängt. Zur *Aufwandsabschätzung* genügt es also, die Anzahl der Gleitkommaoperationen (in Abhängigkeit von n) „abzuzählen“. Wie man dies am geschicktesten macht, hängt dabei von der Struktur des Algorithmus ab. Zudem muss man einige Rechenregeln aus der elementaren Analysis ausnutzen, um zu schönen Ausdrücken zu kommen. Speziell benötigen wir hier die Gleichungen

$$\sum_{i=1}^{n} i = \frac{(n+1)n}{2} \quad \text{und} \quad \sum_{i=1}^{n} i^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n.$$

Wir beginnen mit dem **Rückwärtseinsetzen**, und betrachten zunächst die Multiplikationen und Divisionen: Für $i = n$ muss eine Division durchgeführt werden, für $i = n-1$ muss eine Multiplikation und eine Division durchgeführt werden, für $i = n-2$ müssen zwei Multiplikationen und eine Division durchgeführt werden, usw. So ergibt sich die Anzahl dieser Operationen als

$$1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = \frac{(n+1)n}{2} = \frac{n^2}{2} + \frac{n}{2}.$$

Für die Anzahl der Additionen (bzw. Subtraktionen, was aber rechnerintern das selbe ist) zählt man ab

$$0+1+2+\cdots+n-1=\sum_{i=1}^{n-1} i=\frac{n(n-1)}{2}=\frac{n^2}{2}-\frac{n}{2}.$$

Insgesamt kommt man also auf

$$\frac{n^2}{2}+\frac{n}{2}+\frac{n^2}{2}-\frac{n}{2}=n^2$$

Gleitkommaoperationen. Da das **Vorwärtseinsetzen** völlig analog funktioniert, gilt dafür die gleiche Abschätzung.

Bei der **Gauß–Elimination** gehen wir spaltenweise vor und betrachten die Elemente, die für jedes j eliminiert werden müssen. Wir berücksichtigen hierbei, dass in Schritt (1) nur die Elemente $a_{i\,k}$ mit $k \geq j$ berechnet werden müssen (die anderen sind und bleiben Null). Also benötigt man für festes j gerade je $n-(j-1)+1$ Additionen, Multiplikationen und Divisionen (die „+1“ ergibt sich aus der Operation für b), d.h., $3(n+2-j)$ Operationen. In der j–ten Spalte müssen dabei $n-j$ Einträge eliminiert werden, nämlich für $i=n,\ldots,j+1$ also ergeben sich für die j–te Spalte

$$(n-j)3(n+2-j)=3n^2+6n-3nj-3jn-6j+3j^2=3j^2-6(n+1)j+6n+3n^2$$

Operationen. Dies muss für die Spalten $j=1,\ldots,n-1$ durchgeführt werden, womit wir auf

$$\begin{aligned}
&\sum_{j=1}^{n-1}\left(3j^2-6(n+1)j+6n+3n^2\right)\\
&= 3\sum_{j=1}^{n-1} j^2-6(n+1)\sum_{j=1}^{n-1} j+\sum_{j=1}^{n-1}\left(6n+3n^2\right)\\
&= (n-1)^3+\frac{3}{2}(n-1)^2+\frac{1}{2}(n-1)-6(n+1)\frac{(n-1)n}{2}+(n-1)(6n+3n^2)\\
&= n^3+\frac{3}{2}n^2-\frac{5}{2}n
\end{aligned}$$

Operationen kommen.

Beim **Choleski Verfahren** kann man wieder direkt abzählen: Für jedes i muss man für $j<i$ je $j-1$ Multiplikationen und Additionen durchführen, dazu eine Division, also insgesamt

$$\sum_{j=1}^{i-1}(2(j-1)+1)=2\sum_{j=1}^{i-1} j+\sum_{j=1}^{i-1}(-1)=i(i-1)-(i-1)=i^2-2i+1$$

Operationen (beachte, dass diese Formel auch für $i=1$ stimmt). Für $i=j$ ergeben sich $i-1$ Additionen und Multiplikationen (zum Quadrieren der $l_{j\,j}$) sowie einmal Wurzelziehen, also $2(i-1)+1$ Operationen. Insgesamt erhalten wir also für jedes i

$$i^2-2i+1+2(i-1)+1=i^2-2i+1+2i-2+1=i^2$$

Operationen. Damit ergibt sich die Gesamtzahl der Operationen als

$$\sum_{i=1}^{n} i^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n.$$

Im Vergleich sind dies für große n nur ca. ein Drittel der Operationen des Gauß–Verfahrens.

Zur vollständigen **Lösung eines linearen Gleichungssystems** müssen wir nun einfach die Operationen der Teilalgorithmen aufaddieren.

Für den Gauß–Algorithmus kommt man so auf

$$n^3 + \frac{3}{2}n^2 - \frac{5}{2}n + n^2 = n^3 + \frac{5}{2}n^2 - \frac{5}{2}n$$

Operationen und für das Choleski–Verfahren auf

$$\frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n + 2n^2 = \frac{1}{3}n^3 + \frac{5}{2}n^2 + \frac{1}{6}n$$

Operationen.

Oft ist man nicht an der exakten Zahl der Operationen für ein gegebenes n interessiert, sondern nur an einer Abschätzung für große Dimensionen, d.h., man möchte wissen, wie schnell der Aufwand in Abhängigkeit von n wächst. Man spricht dann von der *Ordnung* eines Algorithmus. Für ein $q > 0$ sagt man, dass ein Algorithmus die Ordnung $O(n^q)$ hat, wenn es eine Konstante $C > 0$ gibt, so dass der Algorithmus für große n weniger als Cn^q Operationen benötigt. Diese Zahl q ist aus den obigen Aufwandsberechnungen leicht abzulesen: Es ist gerade die höchste auftretende Potenz von n. Somit haben Vorwärts– und Rückwärtseinsetzen die Ordnung $O(n^2)$, während Gauß– und Choleski–Verfahren die Ordnung $O(n^3)$ besitzen.

1.6 Iterative Verfahren

Wir haben im letzten Abschnitt gesehen, dass die bisher betrachteten Verfahren die Ordnung $O(n^3)$ besitzen: Wenn sich also n verzehnfacht, so vertausendfacht sich die Anzahl der Operationen und damit die Rechenzeit. Für große Gleichungssysteme mit mehreren 100 000 Unbekannten, die in der Praxis durchaus auftreten, führt dies zu unakzeptabel hohen Rechenzeiten.

Eine Klasse von Verfahren, die eine niedrigere Ordnung hat, sind die sogenannten *iterativen Verfahren*. Allerdings zahlt man für den geringeren Aufwand einen Preis: Man kann bei diesen Verfahren nicht mehr erwarten, eine (bis auf Rundungsfehler) exakte Lösung zu erhalten, sondern muss von vornherein eine gewisse Ungenauigkeit im Ergebnis in Kauf nehmen.

Das Prinzip dieser Verfahren funktioniert wie folgt: Man zerlegt die Matrix A in eine Differenz zweier Matrizen

$$A = M - N,$$

wobei man annimmt, dass M leicht (d.h. mit sehr wenig Aufwand) zu invertieren ist. Dann wählt man sich einen Startvektor $x^{(0)}$ (z.B. den Nullvektor) und berechnet iterativ

$$x^{(i+1)} = M^{-1}Nx^{(i)} + M^{-1}b, \quad i = 0, 1, 2, \ldots. \tag{1.12}$$

Wenn die Zerlegung geeignet gewählt wurde, kann man erwarten, dass sich die Vektoren x_i unter passenden Annahmen an A der gesuchten Lösung annähern.

Beispiel 1.9 Wir illustrieren ein solches Verfahren an dem dreidimensionalen linearen Gleichungssystem mit

$$A = \begin{pmatrix} 15 & 3 & 4 \\ 2 & 17 & 3 \\ 2 & 3 & 21 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 33 \\ 45 \\ 71 \end{pmatrix}.$$

Als Zerlegung $A = M - N$ wählen wir

$$M = \begin{pmatrix} 15 & 0 & 0 \\ 0 & 17 & 0 \\ 0 & 0 & 21 \end{pmatrix} \quad \text{und } N = -\begin{pmatrix} 0 & 3 & 4 \\ 2 & 0 & 3 \\ 2 & 3 & 0 \end{pmatrix},$$

d.h. wir zerlegen A in ihren Diagonalanteil M und den Nicht–Diagonalanteil $-N$. Diagonalmatrizen sind sehr leicht zu invertieren: Man muss einfach jedes Element durch seinen Kehrwert ersetzen, also

$$M^{-1} = \begin{pmatrix} 1/15 & 0 & 0 \\ 0 & 1/17 & 0 \\ 0 & 0 & 1/21 \end{pmatrix}.$$

Damit erhalten wir

$$M^{-1}N = \begin{pmatrix} 0 & -1/5 & -4/15 \\ -2/17 & 0 & -3/17 \\ -2/21 & -1/7 & 0 \end{pmatrix} \quad \text{und} \quad M^{-1}b = \begin{pmatrix} 11/5 \\ 45/17 \\ 71/21 \end{pmatrix}.$$

Wir berechnen nun gemäß der Vorschrift (1.12) die Vektoren $x^{(1)}, \ldots, x^{(10)}$, wobei wir $x^{(0)} = (0\,0\,0)^T$ setzen. Es ergeben sich (jeweils auf vier Nachkommastellen gerundet)

$$\begin{pmatrix} 2.2000 \\ 2.6471 \\ 3.3810 \end{pmatrix}, \begin{pmatrix} 0.7690 \\ 1.7916 \\ 2.7933 \end{pmatrix}, \begin{pmatrix} 1.0968 \\ 2.0637 \\ 3.0518 \end{pmatrix}, \begin{pmatrix} 0.9735 \\ 1.9795 \\ 2.9817 \end{pmatrix}, \begin{pmatrix} 1.0090 \\ 2.0064 \\ 3.0055 \end{pmatrix},$$

$$\begin{pmatrix} 0.9973 \\ 1.9980 \\ 2.9982 \end{pmatrix}, \begin{pmatrix} 1.0009 \\ 2.0006 \\ 3.0005 \end{pmatrix}, \begin{pmatrix} 0.9997 \\ 1.9998 \\ 2.9998 \end{pmatrix}, \begin{pmatrix} 1.0001 \\ 2.0001 \\ 3.0001 \end{pmatrix}, \begin{pmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{pmatrix}.$$

□

Je nach Wahl von M und N erhält man verschiedene Verfahren. Hier wollen wir zwei Verfahren genauer beschreiben und die Iteration (1.12) nicht mit Matrix–Multiplikationen sondern ausführlich für die Einträge $x_j^{(i+1)}$ der Vektoren $x^{(i+1)}$ aufschreiben, so dass die Verfahren dann direkt implementierbar sind. Das erste Verfahren ist das, welches wir auch im Beispiel 1.9 verwendet haben.

Algorithmus 1.10 (Jacobi–Verfahren oder **Gesamtschrittverfahren)**
Wir wählen M als Diagonalmatrix

$$M = \begin{pmatrix} a_{1\,1} & 0 & \dots & 0 \\ 0 & a_{2\,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_{n\,n} \end{pmatrix}$$

und $N = M - A$. Dann ergibt sich (1.12) zu

$$x_j^{(i+1)} = \frac{1}{a_{j\,j}} \left(b_j - \sum_{\substack{k=1 \\ k \neq j}}^{n} a_{j\,k} x_k^{(i)} \right), \quad \text{für } j = 1, \dots, n.$$

□

Eine etwas andere Zerlegung führt zu dem folgenden Verfahren.

Algorithmus 1.11 (Gauß–Seidel–Verfahren oder **Einzelschrittverfahren)**
Wir wählen M als untere Dreiecksmatrix

$$M = \begin{pmatrix} a_{1\,1} & 0 & \dots & 0 \\ a_{2\,1} & a_{2\,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n\,1} & \dots & a_{n\,n-1} & a_{n\,n} \end{pmatrix}$$

und $N = M - A$. Wenn wir (1.12) von links mit M multiplizieren ergibt sich

$$Mx^{(i+1)} = Nx^{(i)} + b.$$

Nun können wir die Komponenten $x_j^{(i+1)}$ mittels Rückwärtseinsetzen bestimmen und erhalten so

$$x_j^{(i+1)} = \frac{1}{a_{j\,j}} \left(b_j - \sum_{k=1}^{j-1} a_{j\,k} x_k^{(i+1)} - \sum_{k=j+1}^{n} a_{j\,k} x_k^{(i)} \right), \quad \text{für } j = 1, \dots, n.$$

□

In beiden Algorithmen brauchen wir noch ein Abbruchkriterium, um zu entscheiden, wann wir die Iteration stoppen. Hier gibt es mehrere Möglichkeiten; ein einfaches aber trotzdem effizientes Kriterium ist es, sich ein $\varepsilon > 0$ vorzugeben, und die Iteration dann abzubrechen, wenn die Bedingung

$$\|x^{(i+1)} - x^{(i)}\|_p < \varepsilon \tag{1.13}$$

für eine vorgegebene Vektornorm $\|\cdot\|_p$ erfüllt ist. Häufig konvergiert das Gauß–Seidel Verfahren schneller als das Jacobi–Verfahren, d.h. die Bedingung (1.13) ist nach weniger Schritten erfüllt; es gibt aber auch Beispiele, in denen es umgekehrt ist.

Bei beiden Verfahren ist es möglich, dass die Iteration nicht konvergiert, d.h. dass die Folge von Vektoren $x^{(i)}$ sich keinem Ergebnisvektor annähert. Man kann beweisen, dass beide Verfahren konvergieren, wenn die Matrix A invertierbar und *strikt diagonaldominant* ist, d.h. wenn

$$|a_{i\,i}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i\,j}|$$

für alle $i = 1, \ldots, n$ erfüllt ist. Für das Gauß–Seidel Verfahren kann man darüberhinaus Konvergenz für symmetrische und positiv definite Matrizen A beweisen.

Es gibt eine Reihe weiterer iterativer Verfahren, die z.T. unter geringeren Voraussetzungen an A konvergieren, in der Formulierung aber komplizierter sind, weswegen wir sie hier aus Zeitgründen nicht behandeln können.

Wir wollen nun noch kurz den Aufwand dieser Iterationsverfahren abschätzen. Nehmen wir an, dass für ein vorgegebenes $\varepsilon > 0$ die Anzahl N_ε der Iterationen bis zur gewünschten Genauigkeit unabhängig von n ist. Für einen Iterationsschritt und eine Komponente $x_j^{(i+1)}$ benötigen wir in beiden Verfahren $n-1$ Multiplikationen und Additionen für die Summe und dazu eine Division, also $2n-1$ Operationen. Für die n Komponenten von $x^{(i+1)}$ ergeben sich so $n(2n-1) = 2n^2 - n$ Operationen, und damit insgesamt

$$N_\varepsilon(2n^2 - n)$$

Operationen. Insbesondere haben diese Algorithmen die Ordnung $O(n^2)$, der Aufwand wächst also deutlich langsamer in n als bei der Gauß–Elimination oder beim Choleski–Verfahren.

Unter zusätzlichen Annahmen an A kann sich der Aufwand dieser Verfahren beträchtlich verringern: Viele sehr große Gleichungssysteme haben die Eigenschaft, dass in jeder Zeile der (sehr großen) Matrix A nur relativ wenige Einträge einen Wert ungleich Null besitzen, man sagt, die Matrix A ist *schwach besetzt*, siehe Abschnitt 1.1.3 für ein Beispiel. Wenn wir annehmen, dass — unabhängig von n — in jeder Zeile von A höchstens m Einträge ungleich Null sind, ist die Anzahl der Operationen in der Berechnung von $x_j^{(i+1)}$ höchstens gleich $2m-1$, und die Gesamtzahl der Operationen ergibt sich zu $N_\varepsilon 2(m-1)n$. Wir erhalten so die Ordnung $O(n)$, d.h. die Anzahl der Operationen wächst linear in n. Fairerweise sollte hier aber erwähnt werden, dass sich unter solchen Bedingungen auch die Anzahl der Operationen in der Gauß–Elimination oder im Choleski–Verfahren verringern kann, allerdings wird die Ordnung dort i.A. nicht kleiner als $O(n^2)$.

Kapitel 2

Interpolation

Die Interpolation von Funktionen oder Daten ist ein häufig auftretendes Problem sowohl in der Mathematik als auch in vielen Anwendungen.

Das allgemeine Problem, die sogenannte *Dateninterpolation*, entsteht, wenn wir eine Menge von Daten (x_i, f_i) für $i = 0, \ldots, n$ gegeben haben (z.B. Messwerte eines Experiments). Die Problemstellung ist nun wie folgt: Gesucht ist eine Funktion F, für die die Gleichung

$$F(x_i) = f_i \quad \text{für } i = 0, 1, \ldots, n \tag{2.1}$$

gilt.

Ein wichtiger Spezialfall dieses Problems ist die *Interpolation von Funktionen*: Nehmen wir an, dass wir eine reellwertige Funktion $f(x)$, $f : \mathbb{R} \to \mathbb{R}$ gegeben haben, die aber (z.B. weil keine explizite Formel bekant ist) sehr kompliziert auszuwerten ist. Das Ziel der Interpolation liegt nun darin eine Funktion $F(x)$ zu bestimmen, die leicht auszuwerten ist, und für die für vorgegebene *Stützstellen* x_0, $x_1, \ldots, x_n$ die Gleichung

$$F(x_i) = f(x_i) \quad \text{für } i = 0, 1, \ldots, n \tag{2.2}$$

gilt. Mit der Schreibweise

$$f_i = f(x_i),$$

erhalten wir hier wieder die Bedingung (2.1), weswegen (2.2) tatsächlich ein Spezialfall von (2.1) ist.

Wir werden in diesem Kapitel Verfahren zur Lösung von (2.1) entwickeln, die dann selbstverständlich auch auf den Spezialfall (2.2) anwendbar sind. Die Wichtigkeit dieses Spezialfalls liegt darin, dass man bei der Interpolation einer Funktion f in natürlicher Weise einen *Interpolationsfehler* über den Abstand zwischen f und F definieren kann, und daher ein Maß für die Güte des Verfahrens erhält. Bei der Dateninterpolation macht dies keinen rechten Sinn, das es ja keine Funktion f gibt, bezüglich der man einen Fehler messen könnte.

2.1 Polynominterpolation

Eine einfache aber oft sehr effektive Methode zur Interpolation ist die Wahl von F als Polynom, also als Funktion der Form

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m. \tag{2.3}$$

Hierbei werden die Werte a_i, $i = 0, \ldots, m$, die *Koeffizienten* des Polynoms genannt. Die höchste auftretende Potenz (hier m) heißt der *Grad* des Polynoms. Um zu betonen, dass wir hier Polynome verwenden, schreiben wir in diesem Abschnitt „P“ statt „F“ für die Interpolationsfunktion.

Das Problem der Polynominterpolation liegt nun darin, die Koeffizienten $a_0, a_1, \ldots, a_m$ so zu bestimmen, dass (2.1) erfüllt ist. Zunächst einmal müssen wir uns dazu überlegen, welchen Grad das gesuchte Polynom haben soll. Hier hilft uns der folgende Satz.

Satz 2.1 Sei $n \in \mathbb{N}$ und seien Daten $(x_i; f_i)$ für $i = 0, \ldots, n$ gegeben, so dass für die Stützstellen $x_i \neq x_j$ gilt für alle $i \neq j$. Dann gibt es genau ein Polynom vom Grad $m = n$, das die Bedingung

$$P(x_i) = f_i \quad \text{für } i = 0, 1, \ldots, n$$

erfüllt ist.

Beweis: Die Koeffizienten a_i sind bestimmt durch das lineare Gleichungssystem

$$\begin{pmatrix} 1 & x_0 & \ldots & x_0^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ \vdots \\ f_n \end{pmatrix}.$$

Die Determinante dieser Matrix ist

$$\prod_{i=1}^{n} \left(\prod_{j=i+1}^{n} (x_i - x_j) \right)$$

und ist damit ungleich Null, falls die x_i paarweise verschieden sind. (Das Symbol $\prod$ ist das Analogon des Summenzeichens $\sum$ für Produkte, d.h. $\prod_{i=1}^{n} x_i = x_1 \cdot x_2 \cdots x_n$.) Also ist die Matrix invertierbar und das Gleichungssystem besitzt eine eindeutige Lösung. □

Für $n + 1$ gegebene Datenpunkte $(x_i; f_i)$ “passt“ also gerade ein Polynom vom Grad n.

Nun ist es aus verschiedenen Gründen nicht besonders effizient, dieses lineare Gleichungssystem tatsächlich zu lösen, um die a_i zu bestimmen (wir erinnern daran, dass die direkte Lösung des linearen Gleichungssystems den Aufwand der Ordnung $O(n^3)$ hat). Wir betrachten daher zwei andere Techniken zur Berechnung des Polynoms P. Beachte, dass beide Techniken das gleiche Polynom berechnen, auch wenn dieses auf verschiedene Arten dargestellt wird.

2.1.1 Lagrange–Polynome

Die Idee der Lagrange–Polynome beruht auf einer geschickten Schreibweise für ein Polynom. Für die vorgegebenen Stützstellen x_0, $x_1 \ldots, x_n$ definieren wir für $i = 0, \ldots, n$ die *Lagrange–Polynome* L_i als

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

Man rechnet leicht nach, dass diese Polynome alle vom Grad n sind, und darüberhinaus die Gleichung

$$L_i(x_k) = \begin{cases} 1 & \text{für } i = k \\ 0 & \text{für } i \neq k \end{cases}$$

erfüllen. Unser gesuchtes Polynom ist damit gegeben durch

$$P(x) = \sum_{i=0}^{n} f_i L_i(x),$$

denn es gilt

$$P(x_k) = \sum_{i=0}^{n} \underbrace{f_i L_i(x_k)}_{\substack{=0 \text{ falls } i \neq k \\ = f_k \text{ falls } i = k}} = f_k,$$

also gerade die gewünschte Bedingung (2.1).

Beispiel 2.2 Betrachte die Daten (3; 68), (2; 16), (5; 352). Die zugehörigen Lagrange–Polynome sind gegeben durch

$$L_0(x) = \frac{x-2}{3-2}\,\frac{x-5}{3-5} = -\frac{1}{2}(x-2)(x-5),$$

$$L_1(x) = \frac{x-3}{2-3}\,\frac{x-5}{2-5} = \frac{1}{3}(x-3)(x-5),$$

$$L_2(x) = \frac{x-2}{5-2}\,\frac{x-3}{5-3} = \frac{1}{6}(x-2)(x-3).$$

Damit erhalten wir

$$P(x) = -68\,\frac{1}{2}(x-2)(x-5) + 16\,\frac{1}{3}(x-3)(x-5) + 352\frac{1}{6}(x-2)(x-3).$$

Für $x = 3$ ergibt sich $P(3) = -68\frac{1}{2}(3-2)(3-5) = 68$, für $x = 2$ berechnet man $P(2) = 16\frac{1}{3}(2-3)(2-5) = 16$ und für $x = 5$ erhalten wir $P(5) = 352\frac{1}{6}(5-2)(5-3) = 352$. □

Durch Abzählen der notwendigen Operationen sieht man, dass die Auswertung des Polynoms P in dieser Form den Aufwand $O(n^2)$ besitzt, also deutlich effizienter als die Lösung eines linearen Gleichungssystems ist. Für eine effiziente Auswertung sollte man die Nenner der Lagrange–Polynome vorab berechnen und speichern, damit diese nicht bei jeder Auwertung von P erneut berechnet werden müssen.

Die Lagrange–Darstellung ist praktisch, wenn man Messwerte f_i nachträglich verändern will, da diese explizit in der Polynomdarstellung auftauchen. Sie ist aber im Allgemeinen unpraktisch, wenn man Datenpunkte hinzufügen will, da dann sämtliche L_i neu berechnet werden müssen.

2.1.2 Das Newton–Schema

In diesem Fall, also wenn Datenpunkte hinzugefügt werden sollen, ist eine andere Berechnungsart geeigneter, nämlich das sogenannte *Newton–Schema*. Hierbei definiert man zunächst die folgenden Werte:

$$f_{[x_i]} = f_i, \quad i = 0, \ldots, n$$

und berechnet daraus für Stützstellen–Mengen der Form $\{x_l, x_{l+1}, \ldots, x_{l+k}\}$ mit $0 \leq l < l + k \leq n$ rekursiv die sogenannten *dividierten Differenzen*

$$f_{[x_l, x_{l+1}, \ldots, x_{l+k}]} := \frac{f_{[x_l, x_{l+1}, \ldots, x_{l+k-1}]} - f_{[x_{l+1}, x_{l+2}, \ldots, x_{l+k}]}}{x_l - x_{l+k}}.$$

Abbildung 2.1 veranschaulicht diese rekursive Berechnung

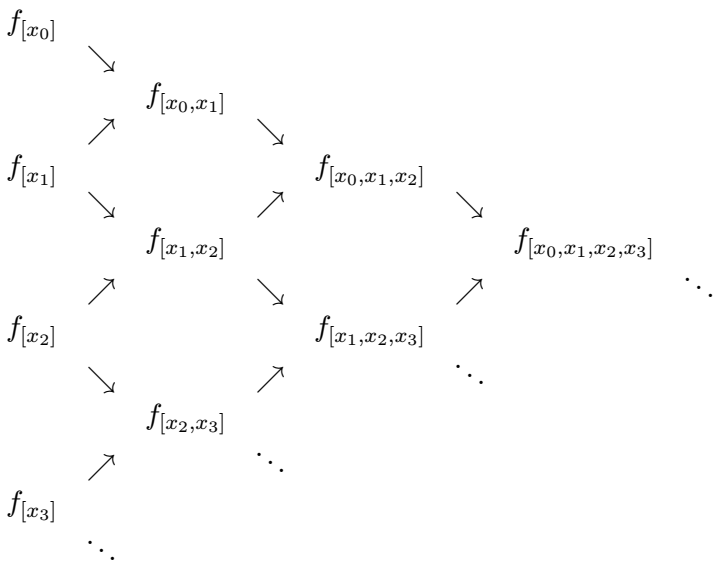


Abbildung 2.1: Illustration des Newton–Schemas

Das gesuchte Interpolationspolynom ergibt sich nun mit Hilfe dieser Werte als

$$\begin{aligned} P(x) &= f_{[x_0]} + f_{[x_0,x_1]}(x - x_0) + f_{[x_0,x_1,x_2]}(x - x_0)(x - x_1) \\ &\quad + \ldots + f_{[x_0,\ldots,x_n]}(x - x_0)(x - x_1)\cdots(x - x_{n-1}). \end{aligned}$$

Beachte, dass wir für dieses Polynom nur jeweils die Werte aus den ersten Zeilen des Schemas benötigen. Die anderen werden jedoch zur Berechnung dieser Werte gebraucht.

Um zu beweisen, dass dies wirklich das gesuchte Polynom ist, definieren wir zunächst für $k = 0, 1, \ldots, n$ die Hilfspolynome

$$\begin{aligned} P_k(x) &= f_{[x_0]} + f_{[x_0,x_1]}(x - x_0) + f_{[x_0,x_1,x_2]}(x - x_0)(x - x_1) \\ &\quad + \ldots + f_{[x_0,\ldots,x_k]}(x - x_0)(x - x_1)\cdots(x - x_{k-1}) \end{aligned}$$

(also $P(x) = P_n(x)$) und

$$\begin{aligned} \widetilde{P}_k(x) &= f_{[x_1]} + f_{[x_1,x_2]}(x - x_1) + f_{[x_1,x_2,x_3]}(x - x_1)(x - x_2) \\ &\quad + \ldots + f_{[x_1,\ldots,x_{k+1}]}(x - x_1)(x - x_2)\cdots(x - x_k). \end{aligned}$$

Damit rechnet man — unter Verwendung der Definition der dividierten Differenzen — nach, dass die Gleichung

$$P_{k+1}(x) = \frac{(x_k - x)P_k(x) + (x - x_0)\widetilde{P}_k(x)}{x_k - x_0} \tag{2.4}$$

gilt. Per Induktion zeigen wir nun

$$P_k(x_j) = f_j \text{ für } j = 0, \ldots, k \quad \text{und} \quad \widetilde{P}_k(x_j) = f_j \text{ für } j = 1, \ldots, k+1.$$

Für $k = 0$ ist dies sofort klar, für den Schritt von k nach $k + 1$ verwendet man (2.4) für P_{k+1}; die Behauptung für $\widetilde{P}_{k+1}$ folgt daraus, da man $\widetilde{P}_{k+1}$ aus P_{k+1} erhält, indem man die Daten $(x_0, f_0), \ldots, (x_k, f_k)$ durch $(x_1, f_1), \ldots, (x_{k+1}, f_{k+1})$ ersetzt, wenn also $P_{k+1}(x_j) = f_j$ für $j = 0, \ldots, k$ ist, muss — da beide Polynome die gleiche Struktur haben und die Daten beliebig sind — auch $\widetilde{P}_{k+1}(x_j) = f_j$ für $j = 1, \ldots, k + 1$ gelten. □

Wir wiederholen unser Beispiel aus dem letzten Abschnitt.

Beispiel 2.3 Betrachte die Daten (3; 68), (2; 16), (5; 352). Die dividierten Differenzen ergeben sich als

$$\begin{array}{ccccc} f_{[x_0]} = 68 & & & & \\ & \searrow & & & \\ & & f_{[x_0,x_1]} = \frac{68-16}{3-2} = 52 & & \\ & \nearrow & & \searrow & \\ f_{[x_1]} = 16 & & & & f_{[x_0,x_1,x_2]} = \frac{52-112}{3-5} = 30 \\ & \searrow & & \nearrow & \\ & & f_{[x_1,x_2]} = \frac{16-352}{2-5} = 112 & & \\ & \nearrow & & & \\ f_{[x_2]} = 352 & & & & \end{array}$$

Damit erhalten wir das Interpolationspolynom

$$P(x) = P_2(x) = 68 + 52(x - 3) + 30(x - 3)(x - 2).$$

Für $x = 3$ erhalten wir $P(3) = 68$, für $x = 2$ ergibt sich $P(2) = 68+52(2-3) = 68-52 = 16$ und für $x = 5$ berechnet man $P(5) = 68 + 52(5 - 3) + 30(5 - 3)(5 - 2) = 68 + 104 + 180 = 352$. □

Auch wenn die Berechnung des Interpolationspolynoms über das Newton–Schema komplizierter aussieht, hat sie den Vorteil, dass wir weitere Stützstellen leicht hinzufügen können, da sich die Polynome für mehr Stützstellen einfach durch Addition zusätzlicher Terme aus denen für weniger Stützstellen ergeben. Dafür müssen hier bei der Änderung von Messwerten f_i Daten alle dividierten Differenzen neu berechnet werden.

Die Berechnung der dividierten Differenzen kann sehr effizient mit dem folgenden Algorithmus durchgeführt werden, wobei die Schreibweise

$$F_0 = f_{[x_0]}, \;\; F_1 = f_{[x_0,x_1]}, \ldots, \;\; F_n = f_{[x_0,x_1,\ldots,x_n]}$$

verwendet wird.

Algorithmus 2.4 (Berechnung der dividierten Differenzen)
Gegeben seien die Daten $(x_0, f_0), \ldots, (x_n, f_n)$.

(0) Für i von 0 bis n setze $F_i := f_i$

(1) Für k von 1 bis n
Für i von n bis k (nach unten zählend) berechne $F_i := \frac{F_{i-1}-F_i}{x_{i-k}-x_i}$

□

Dieser Algorithmus hat einen Aufwand von der Ordnung $O(n^2)$. Sind diese F_i einmal berechnet, so lässt sich das Interpolationspolynom effizient mit der folgenden Formel, dem sogenannten *Horner–Schema* berechnen.

Algorithmus 2.5 (Berechnung von $P(x)$ mittels Horner–Schema)
Gegeben seien die Stützstellen $x_0, \ldots, x_n$, die dividierten Differerenzen $F_0, \ldots, F_n$ aus Algorithmus 2.4 und ein $x \in \mathbb{R}$. Dann erhalten wir $p = P(x)$ mittels

(0) Setze $p := F_n$

(1) Für i von $n-1$ bis 0 berechne $p := F_i + (x - x_i)p$

□

Beachte, dass dieser Algorithmus nur einen Aufwand der Ordnung $O(n)$ besitzt, und damit viel effizienter als die Auswertung des Polynoms in Lagrange–Darstellung ist. Das Newton–Schema ist also vorzuziehen, wenn ein Interpolations–Polynom einmal berechnet und dann sehr oft ausgewertet werden soll.

2.1.3 Fehlerabschätzungen

Wir betrachten in diesem Abschnitt das Problem der Funktionsinterpolation (2.2) für eine gegebene Funktion $f : \mathbb{R} \to \mathbb{R}$. Wir wollen abschätzen, wie groß der Abstand des interpolierenden Polynoms P von der Funktion f ist. Hierbei bezeichnen wir mit I das von den Stützstellen $x_0, \ldots, x_n$ definierte Intervall $I = [\min\limits_{i=0,\ldots,n} x_i, \max\limits_{i=0,\ldots,n} x_i]$.

Den Abstand zwischen P und f definieren wir nun punktweise als

$$|f(x) - P(x)| \quad \text{für } x \in I.$$

Der folgende Satz gibt eine Abschätzung für den Abstand in Abhängigkeit von den Stützstellen an. Hierbei bezeichnet $f^{(k)}$ die *k–te Ableitung* der Funktion f. Das Ausrufezeichen "!" bezeichnet die übliche *Fakultät* für natürliche Zahlen k, d.h. $k! = 1 \cdot 2 \cdot 3 \cdots k$.

Satz 2.6 Sei f $(n+1)$–fach stetig differenzierbar und sei P das Interpolationspolynom zu den paarweise verschiedenen Stützstellen $x_0, \ldots, x_n$. Dann gelten die folgenden Ausagen.

(i) Für alle $x \in I$ gibt es ein $\xi \in I$, so dass die Gleichung

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1) \cdots (x - x_n)$$

gilt.

(ii) Es gilt die Abschätzung

$$|f(x) - P(x)| \leq \max_{y \in I} \left| \frac{f^{(n+1)}(y)}{(n+1)!} (x - x_0)(x - x_1) \cdots (x - x_n) \right|.$$

(iii) Wenn wir das Intervall I als $I = [a, b]$ schreiben, gilt die Abschätzung

$$|f(x) - P(x)| \leq \max_{y \in I} \left| f^{(n+1)}(y) \right| \frac{(b-a)^{n+1}}{(n+1)!}.$$

Beweis: (i) Wähle ein $x \in I$ und setze

$$c = \frac{f(x) - P(x)}{(x - x_0) \cdots (x - x_n)}.$$

Mit diesem c definieren wir die Funktion

$$\Delta(y) = f(y) - P(y) - c(y - x_0) \cdots (y - x_n).$$

Dann ist $\Delta(y)$ wieder $(n+1)$ mal stetig differenzierbar und hat (mindestens) die $n+2$ Nullstellen $y = x_0, \ldots, x_n$ und $y = x$. Wir nummerieren diese Nullstellen aufsteigend mit der Bezeichnung $y_0^{(0)} < y_1^{(0)} < \ldots < y_{n+1}^{(0)}$. Nach dem Satz von Rolle aus der Analysis gilt: Zwischen je zwei Nullstellen der Funktion $\Delta(y)$ liegt (mindestens) eine Nullstelle ihrer Ableitung $\Delta'(y)$ (hier wie im Folgenden leiten wir nach der Variablen "y" ab). Also liegt für jedes $i = 0, \ldots, n$ zwischen den Werten $y_i^{(0)}$ und $y_{i+1}^{(0)}$ ein Wert $y_i^{(1)}$ mit $\Delta(y_i^{(1)}) = 0$, und wir erhalten $n+1$ Nullstellen $y_0^{(1)} < y_1^{(1)} < \ldots < y_n^{(1)}$ für die Ableitung $\Delta'(y) = \Delta^{(1)}(x)$. Indem wir induktiv fortfahren, erhalten wir $n+1-k$ Nullstellen $y_0^{(k)} < y_1^{(k)} < \ldots < y_{n+1-k}^{(k)}$ für die Ableitung $\Delta^{(k)}$ und damit für $k = n+1$ eine Nullstelle $\xi = y_0^{(n+1)}$ für die Funktion $\Delta^{(n+1)}$.

Aus den Ableitungsregeln für Polynome folgt, dass $P^{(n+1)}(y) = 0$ und $[(y - x_0) \cdots (y - x_n)]^{(n+1)} = (n + 1)!$ ist für alle $x \in \mathbb{R}$. Damit erhalten wir

$$0 = \Delta^{(n+1)}(\xi) = f^{(n+1)}(\xi) - c(n + 1)! = f^{(n+1)}(\xi) - \frac{f(x) - P(x)}{(x - x_0) \cdots (x - x_n)}(n + 1)!,$$

also

$$f^{(n+1)}(\xi) = \frac{f(x) - P(x)}{(x - x_0) \cdots (x - x_n)}(n + 1)!.$$

Auflösen nach $f(x) - P(x)$ liefert die Gleichung in (i).

(ii) Diese Abschätzung folgt aus (i) wegen

$$\begin{aligned} |f(x) - P(x)| &= \left| \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1) \cdots (x - x_n) \right| \\ &\leq \max_{y \in I} \left| \frac{f^{(n+1)}(y)}{(n+1)!}\,(x - x_0)(x - x_1) \cdots (x - x_n) \right|, \end{aligned}$$

da ξ aus I ist.

(iii) Wenn wir das Intervall I als $I = [a, b]$ schreiben, folgt für alle x und x_i aus I die Abschätzung

$$|x - x_i| \leq b - a.$$

Damit erhalten wir aus Satz 2.6 (ii) die Abschätzung

$$|f(x) - P(x)| \leq \max_{y \in I} \left| f^{(n+1)}(y) \right| \frac{(b - a)^{n+1}}{(n + 1)!},$$

also gerade die Behauptung. □

Wir illustrieren diese Abschätzung an 2 Beispielen, vgl. dazu das 4. Übungsblatt.

Beispiel 2.7 Betrachte die Funktion $f(x) = \sin(x)$ auf dem Intervall $I = [0, \pi]$. Die Ableitungen von "f" sind

$$f^{(1)}(x) = \cos(x), \;\; f^{(2)}(x) = -\sin(x), \;\; f^{(3)}(x) = -\cos(x), \;\; f^{(4)}(x) = \sin(x), \;\; \ldots$$

Für alle diese Funktionen gilt $|f^{(k)}(x)| \leq 1$ für alle $x \in \mathbb{R}$. Mit äquidistanten Stützstellen $x_i = 2\pi i/n$ ergibt sich damit die Abschätzung

$$|f(x) - P(x)| \leq \max_{y \in I} \left| f^{(n+1)}(y) \right| \frac{(b - a)^{n+1}}{(n + 1)!} \leq \frac{(2\pi)^{n+1}}{(n + 1)!}.$$

Dieser Term konvergiert für wachsende n sehr schnell gegen 0, weswegen man schon für kleine n eine sehr gute Übereinstimmung der Funktionen erwarten kann. □

Beispiel 2.8 Betrachte die Funktion $f(x) = 1/(1 + x^2)$ auf dem Intervall $I = [-5, 5]$. Die exakten Ableitungen führen zu ziemlich komplizierten Termen, man kann aber nachprüfen, dass für $C \approx 5$ die Abschätzung

$$\max_{y \in I} |f^{(n)}(y)| \approx \frac{C}{n} \frac{n!}{C^n}$$

gilt. Damit ergibt sich

$$\begin{aligned} |f(x) - P(x)| &\leq \max_{y \in I} \left| f^{(n+1)}(y) \right| \frac{(b-a)^{n+1}}{(n+1)!} \\ &= \frac{C}{n+1} \frac{(n+1)!}{C^{n+1}} \frac{(b-a)^{n+1}}{(n+1)!} \leq \frac{5}{n+1} \frac{(n+1)!}{5^{n+1}} \frac{10^{n+1}}{(n+1)!} = 5 \frac{2^{n+1}}{n+1}. \end{aligned}$$

Dieser Term wächst für große n gegen unendlich, weswegen die Abschätzung hier keine brauchbare Fehlerschranke liefert, und tatsächlich zeigen sich bei dieser Funktion für äquidistante Stützstellen bei numerischen Tests große Probleme; insbesondere lässt sich für wachsende n keine Konvergenz erzielen, statt dessen stellt man für große n starke Schwankungen ("Oszillationen") des interpolierenden Polynoms fest. □

Bemerkung 2.9 (In der Vorlesung nicht behandelt)
Bei der Interpolation von Funktionen hat man oft die Möglichkeit, die Stützstellen x_i frei zu wählen. Daher stellt sich die Frage, ob man diese so wählen kann, dass der Fehler $|f(x) - P(x)|$ minimal wird.

Dazu betrachten wir die Abschätzung (ii) aus dem obigen Satz. Offenbar können wir die Ableitung $f^{(n+1)}$ nicht beeinflussen, da die Funktion f ja vorgegeben ist. Wir können aber die Stützstellen so wählen, dass der von den x_i abhängige Teil der Abschätzung, also der Term $|(x - x_0)(x - x_1) \cdots (x - x_n)|$ minimal wird. Genauer suchen wir zu vorgegebener Anzahl $n > 0$ Stützstellen $x_0, \ldots, x_n$, so dass der Ausdruck

$$\max_{x \in I} |(x - x_0)(x - x_1) \cdots (x - x_n)|$$

minimal wird.

Dieses Problem wurde von dem russischen Mathematiker Tschebyscheff gelöst, der für das Interpolationsintervall $I = [a, b]$ die optimalen Stützstellen

$$x_i = \frac{a+b}{2} - \frac{b-a}{2} \cos\left(\frac{(2\,i+1)\pi}{2(n+1)} \right), \quad i = 0, 1, \ldots, n$$

errechnet hat. Beachte, dass die extremalen Stützstellen x_0 und x_n hier nicht mit den Randwerten a und b übereinstimmen.

Dieses Verfahren mach natürlich nur dann Sinn, wenn wirklich eine Funktion interpoliert werden soll, da man bei vorgegebenen Daten (x_i, f_i) üblicherweise keine Freiheiten bei der Wahl der x_i hat. □

Abschließend sei zur Polynominterpolation noch bemerkt, dass die Berechnung von interpolierenden Polynomen für eine große Anzahl n von Stützstellen i.A. problematisch ist, da Polynome hohen Grades leicht zu starken Oszillationen neigen, die sich für dicht liegende Stützstellen durch Rundungsfehler selbst dann einstellen können, wenn die zu interpolierende Funktion gutartig ist.

Wir werden daher in nächsten Abschnitt eine weitere Möglichkeit zur Interpolation betrachten, die auch auf Polynomen beruht, aber die Verwendung von Polynomen hohen Grades vermeidet.

2.2 Splineinterpolation

Wir betrachten in diesem Abschnitt wiederum das Interpolationsproblem (2.1), nehmen aber jetzt der einfacheren Schreibweise wegen an, dass die Stützstellen aufsteigend angeordnet sind, also $x_0 < x_1 < \ldots < x_n$ gilt.

Die Grundidee der *Splineinterpolation* liegt darin, die interpolierende Funktion nicht global, sondern nur auf jedem Teilintervall $[x_i, x_{i+1}]$ als Polynom zu wählen. Diese Teilpolynome sollten dabei and den Intervallgrenzen nicht beliebig sondern möglichst glatt zusammenlaufen. Eine solche Funktion, die aus glatt zusammengefügten stückweisen Polynomen besteht, nennt man *Spline*.

Die verschiedenen Klassen von Splines unterscheiden sich nun dadurch, dass sie auf Polynomen verschiedenen Grades aufbauen. Wir werden hier exemplarisch die sogenannten *kubischen Splines* behandeln, die — wie der Name bereits andeutet — auf Polynomen dritten Grades beruhen. Kubische Splines sind in den Anwendungen weit verbreitet, weswegen wir sie hier als Beispielklasse verwenden. Splines, die auf anderen Polynomen aufbauen, werden aber ganz ähnlich behandelt.

Wir müssen zunächst die obige umgangssprachliche Definition in mathematische Ausdrücke fassen.

Definition 2.10 Seien $x_0 < x_1 < \ldots < x_n$ Stützstellen. Eine stetige Funktion $S : [x_0, x_n] \to \mathbb{R}$ heißt *kubischer Spline*, falls die folgenden zwei Bedingungen erfüllt sind.

(i) Auf jedem Intervall $I_k = [x_{k-1}, x_k]$ mit $k = 1, \ldots, n$ ist S gegeben durch ein Polynom S_k dritten Grades, d.h. für $x \in I_k$ gilt

$$S(x) = S_k(x) = a_k + b_k(x - x_{k-1}) + c_k(x - x_{k-1})^2 + d_k(x - x_{k-1})^3.$$

(ii) Die Ableitungen der Polynome S_k an den Stützstellen erfüllen die Bedingungen

$$S_k'(x_k) = S_{k+1}'(x_k) \quad \text{und} \quad S_k''(x_k) = S_{k+1}''(x_k)$$

für alle $k = 1, \ldots, n-1$. □

Bedingung (i) sagt einfach, dass S aus Polynomen zusammengesetzt ist, während Bedingung (ii) das "glatte Zusammenstoßen" präzisiert: Die Bedingungen an die Ableitungen garantieren, dass die Splines an den "Nahtstellen" keine "Knicke" haben und ihre Krümmungungen stetig ineinander übergehen.

Ein solcher kubischer Spline aus Definition 2.10 löst dann das Interpolationsproblem, falls zusätzlich die Bedingung (2.1) erfüllt ist, also $S(x_i) = f_i$ für alle $i = 0, \ldots, n$ gilt.

Wir müssen uns zunächst Gedanken darüber machen, ob das Problem der Splineinterpolation so wohldefiniert ist, d.h. ob ein interpolierender Spline immer existiert und ob er eindeutig ist. Zur Erinnerung: Nach den Ableitungsregeln für Polynome gilt

$$S_k'(x) = b_k + 2c_k(x - x_{k-1}) + 3d_k(x - x_{k-1})^2 \quad \text{und} \quad S_k''(x) = 2c_k + 6d_k(x - x_{k-1}).$$

Zur Bestimmung von S müssen wir $4n$ Werte a_i, b_i, c_i und d_i für $i = 1, \ldots, n$ berechnen.

Aus Definition 2.10 (ii) erhalten wir hierbei $2n-2$ lineare Gleichungen und aus (2.1) erhalten wir weitere $2n$ lineare Gleichungen.

Insgesamt ergeben sich so $4n-2$ lineare Gleichungen für $4n$ Unbekannte; dieses lineare Gleichungssystem ist damit lösbar, allerdings gibt es unendlich viele Lösungen. Der Grund für die fehlenden Gleichungen liegt in den Randpunkten x_0 und x_n, in denen wir keine Glattheitsbedingungen fordern müssen.

Um eine eindeutige Lösung zu erhalten, müssen wir zwei weitere Gleichungen festlegen, welche sich üblicherweise aus *Randbedingungen* in den Punkten x_0 und x_n ergeben. Wir wollen hierbei die sogenannten *natürlichen Randbedingungen* betrachten. Hier wird gefordert, dass die zweiten Ableitungen am Rand gleich Null sein sollen, also $S''(x_0) = S''(x_n) = 0$. Dies führt auf zwei zusätzlichen Gleichungen, womit das Gleichungssystem dann eindeutig lösbar ist (tatsächlich muss man hier natürlich noch nachrechnen, dass das entstehende System eine invertierbare Matrix besitzt, was wir hier nicht explizit durchführen werden).

Natürlich kann man viele andere Randbedingungen festlegen, die dann auf andere zusätzliche Gleichungen führen.

Um die Koeffizienten a_k, b_k, c_k und d_k für $k = 1, \ldots, n$ tatsächlich numerisch zu berechnen empfiehlt es sich, zunächst die Werte

$$f_k'' = S''(x_k) \quad \text{und} \quad h_k = x_k - x_{k-1}$$

für $k = 0, \ldots, n$ bzw. $k = 1, \ldots, n$ zu definieren. Löst man dann die 4 Gleichungen

$$S_k(x_{k-1}) = f_{k-1}, \;\; S_k(x_k) = f_k, \;\; S_k''(x_{k-1}) = f_{k-1}'', \;\; S_k''(x_k) = f_k'' \tag{2.5}$$

— unter Ausnutzung der Ableitungsregeln für Polynome — nach a_k, b_k, c_k und d_k auf, so erhält man

$$\begin{aligned} a_k &= f_{k-1} \\ b_k &= \frac{f_k - f_{k-1}}{h_k} - \frac{h_k}{6}(f_k'' + 2f_{k-1}'') \\ c_k &= \frac{f_{k-1}''}{2} \\ d_k &= \frac{f_k'' - f_{k-1}''}{6h_k}. \end{aligned}$$

Da die Werte h_k und f_k ja direkt aus den Daten verfügbar sind, müssen lediglich die Werte f_k'' berechnet werden. Da aus den natürlichen Randbedingungen sofort $f_0'' = 0$ und $f_n'' = 0$ folgt, brauchen nur die Werte $f_1'', \ldots, f_{n-1}''$ berechnet werden.

Beachte, dass wir in (2.5) bereits die Bedingungen an S_k und S_k'' in den Stützstellen verwendent haben. Aus den noch nicht benutzten Gleichungen für die ersten Ableitungen erhält man nun die Gleichungen für die f_k'': Aus $S_k'(x_k) = S_{k+1}'(x_k)$ erhält man

$$b_k + 2c_k(x_k - x_{k-1}) + 3d_k(x_k - x_{k-1})^2 = b_{k+1}$$

für $k = 1, \ldots, n-1$. Indem man hier die Werte f_k'' und h_k gemäß den obigen Gleichungen bzw. Definitionen einsetzt erhält man

$$h_k f_{k-1}'' + 2(h_k + h_{k+1}) f_k'' + h_{k+1} f_{k+1}'' = 6\frac{f_{k+1} - f_k}{h_{k+1}} - 6\frac{f_k - f_{k-1}}{h_k} =: \delta_k$$

für $k = 1, \ldots, n-1$. Dies liefert genau $n-1$ Gleichungen für die $n-1$ Unbekannten $f_1'', \ldots, f_{n-1}''$. In Matrixform geschrieben erhalten wir so das Gleichungssystem

$$\begin{pmatrix} 2(h_1+h_2) & h_2 & 0 & \ldots & \ldots & 0 \\ h_2 & 2(h_2+h_3) & h_3 & \ddots & \ddots & \vdots \\ 0 & h_3 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & 0 & h_{n-1} & 2(h_{n-1}+h_n) \end{pmatrix} \begin{pmatrix} f_1'' \\ f_2'' \\ \vdots \\ \vdots \\ f_{n-1}'' \end{pmatrix} = \begin{pmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \vdots \\ \delta_{n-1} \end{pmatrix}$$

Zur Berechnung des Interpolationssplines löst man also zunächst dieses Gleichungssystem und berechnet dann gemäß der obigen Formel die Koeffizienten a_k, b_k, c_k, d_k aus den f_k''.

2.3 Trigonometrische Interpolation und Fourier–Transformation

Zum Abschluss unseres Kapitels über Interpolation wollen wir uns mit einer weiteren Klasse von Interpolationsfunktionen beschäftigen. Das Verfahren der *trigonometrischen Interpolation*, das wir hier behandeln werden, hat in erster Linie ganz andere Anwendungen als die reine Interpolation; diese Anwendungen werden wir im Abschnitt 2.3.3 skizzeren. Zunächst aber betrachten wir das Verfahren als Interpolationsproblem.

2.3.1 Interpolation durch trigonometrische Polynome

Statt als Summe von Potenzen x^k sind trigonometrische Polynome als Summe von Sinus– und Kosinus–Funktionen definiert.

Definition 2.11 Ein *trigonometrisches Polynom* vom Grad N ist eine komplexwertige Funktion $T : \mathbb{R} \to \mathbb{C}$, die für $n = 2N + 1$ gegeben ist durch

$$\begin{aligned} T(x) &= d_0 + \sum_{k=1}^{N} (d_k + d_{n-k}) \cos\left(k 2\pi \frac{x - x_0}{p}\right) \\ &\quad + i \sum_{k=1}^{N} (d_k - d_{n-k}) \sin\left(k 2\pi \frac{x - x_0}{p}\right). \end{aligned}$$

Hierbei sind $x_0 \in \mathbb{R}$ und $p > 0$ gegebene Zahlen und $d_0, \ldots, d_{n-1} \in \mathbb{C}$ die *Koeffizienten* des trigonometrischen Polynoms. □

Trigonometrische Polynome werden üblicherweise mit komplexen Werten angegeben, im reellwertigen Fall gelten für die Koeffizienten $d_k = a_k + ib_k$ die Bedingungen

$$b_0 = 0, \quad a_k = a_{n-k}, \;\; b_k = -b_{n-k} \text{ für alle } k = 1, \ldots, N. \tag{2.6}$$

In diesem Fall ergibt sich das trigonometrische Polynom mit Koeffizienten $d_k = a_k + ib_k$ zu

$$\begin{aligned} T(x) \;=\; a_0 \;+\; & \sum_{k=1}^{N} 2a_k \cos\left(k2\pi\frac{x-x_0}{p}\right) \\ - \; & \sum_{k=1}^{N} 2b_k \sin\left(k2\pi\frac{x-x_0}{p}\right). \end{aligned}$$

Da ein trigonometrisches Polynom i.A. komplexwertig ist, kann man in den zu interpolierenden Daten (x_j, f_j) auch die f_j als komplexe Zahlen wählen. Sind die f_i reell, erfüllen die berechneten Koeffizienten automatisch die Bedingungen (2.6). Auch im reellwertigen Fall empfiehlt es sich allerdings, zunächst das trigonometrische Polynom in der komplexen Form zu verwenden, da die Formeln zur Berechnung der d_k deutlich einfacher sind als die Formeln zur direkten Berechnung der a_k und b_k.

Das trigonometrische Polynom löst das folgende Interpolationsproblem:
Seien Daten $(x_0, f_0), \ldots, (x_n, f_n)$ gegeben, die die folgenden Bedingungen erfüllen:

(i) Die Stützstellen $x_j \in \mathbb{R}$ sind äquidistant verteilt auf dem Intervall $[x_0, x_0 + p]$:

$$x_j = x_0 + j\frac{p}{n}$$

(ii) Die Werte $f_j \in \mathbb{C}$ erfüllen $f_0 = f_n$

Dann gibt es komplexe Koeffizienten $d_0, \ldots, d_{n-1} \in \mathbb{C}$, so dass das zugehörige trigonometrische Polynom gerade das Interpolationsproblem

$$T(x_j) = f_j \text{ für alle } j = 0, \ldots, n$$

erfüllt. Diese sind gegeben durch die Matrix–Multiplikation

$$\begin{pmatrix} d_0 \\ \vdots \\ d_{n-1} \end{pmatrix} = \frac{1}{n}\,\overline{V}\begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \end{pmatrix}, \tag{2.7}$$

wobei $\overline{V}$ eine sogenannte *Vandermondesche Matrix*

$$\overline{V} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \bar{\omega} & \bar{\omega}^2 & \ldots & \bar{\omega}^{n-1} \\ 1 & \bar{\omega}^2 & \bar{\omega}^4 & \ldots & \bar{\omega}^{2(n-1)} \\ 1 & \bar{\omega}^3 & \bar{\omega}^6 & \ldots & \bar{\omega}^{3(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \bar{\omega}^{n-1} & \bar{\omega}^{(n-1)2} & \ldots & \bar{\omega}^{(n-1)^2} \end{pmatrix}$$

mit $\bar{\omega} = e^{-i2\pi/n}$ ist.

Wir können hier aus Zeitgründen weder auf den Beweis dieser Behauptung noch auf eine Analyse des Interpolationsfehlers eingehen. Wenn die f_j sich mittels (2.2) aus Funktionswerten $f(x_j)$ einer Funktion $f : \mathbb{R} \to \mathbb{C}$ ergeben, kann man beweisen, dass das zugehörige trigonometrische Polynom T — unter geeigneten Bedingungen an f — eine brauchbare Approximation für f darstellt.

Wir werden nun eine Methode skizzieren, mit der die Koeffizienten d_i schnell und numerisch effizient berechnet werden können.

2.3.2 Schnelle Fourier–Transformation

Die Koeffizienten $d_0, \ldots, d_{n-1}$ des trigonometrischen Interpolationspolynoms werden üblicherweise als die *diskreten Fourier–Koeffizienten* der Daten $f_0, \ldots, f_{n-1}$ bezeichnet (wegen der Annahmen (i) und (ii) von oben werden weder die x_j noch der Wert f_n in der Berechnung der d_j benötigt). Die Abbildung

$$\begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \end{pmatrix} \mapsto \begin{pmatrix} d_0 \\ \vdots \\ d_{n-1} \end{pmatrix}, \tag{2.8}$$

die man als Abbildung des $\mathbb{C}^n$ in den $\mathbb{C}^n$ auffassen kann, wird als *diskrete Fourier–Transformation* oder kurz als *DFT* bezeichnet.

Aus (2.7) folgt, dass dies eine lineare Transformation ist; darüberhinaus rechnet man leicht nach, dass die *Rücktransformation*

$$\begin{pmatrix} d_0 \\ \vdots \\ d_{n-1} \end{pmatrix} \mapsto \begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \end{pmatrix}, \tag{2.9}$$

durch die lineare Abbildung

$$\begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \end{pmatrix} = V \begin{pmatrix} d_0 \\ \vdots \\ d_{n-1} \end{pmatrix} \tag{2.10}$$

gegeben ist, wobei V wiederum eine Vandermondesche Matrix

$$V = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \ldots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{(n-1)2} & \ldots & \omega^{(n-1)^2} \end{pmatrix}$$

mit $\omega = e^{i2\pi/n} = \bar{\omega}^{-1}$ ist. Um dies zu überprüfen, muss man nur nachweisen, dass

$$\frac{1}{n} \overline{V}\, V = \mathrm{Id},$$

also gleich der Einheitsmatrix ist. Dies sieht man leicht, indem man die Gleichungen $\omega^k \bar{\omega}^j = e^{(k-j)i2\pi/n}$ und $e^{li2\pi} = 1$ für alle ganzen Zahlen $k, j, l \in \mathbb{Z}$ verwendet.

Sowohl die diskrete Fourier–Transformation (2.8) als auch die Rücktransformation (2.9) sind wichtige mathematische Operationen. Diese könnten über die Abbildungen (2.7) bzw. (2.10) ausgeführt können, allerdings hat die Matrix–Vektor–Multiplikation den Aufwand der Ordnung $O(n^2)$, weswegen es sinnvoll ist, einen anderen Algorithmus zu verwenden. Hier bietet sich die sogenannte *schnelle Fourier–Transformation* (meist als *FFT = Fast Fourier Transformation* abgekürzt) an. Die Idee dieses Algorithmus liegt in der Beobachtung, dass sich die Fourier–Koeffizienten für einen Datensatz $(f_0, \ldots, f_{n-1})$ mit gerader Anzahl von Koeffizienten $n = 2m$ leicht berechnen lassen, wenn sie für die "halbierten" Datensätze $(f_0, f_2, \ldots, f_{2m-2})$ und $(f_1, f_3, \ldots, f_{2m-1})$ mit geraden bzw. ungeraden Koeffizienten bereits bekannt sind.

Bezeichnen nämlich $d_k^{[m,\mathrm{g}]}$ und $d_k^{[m,\mathrm{u}]}$ die Fourier–Koeffizienten der halbierten Datensätze, so gelten die Gleichungen

$$d_k^{[2m]} = \frac{1}{2}\left(d_k^{[m,\mathrm{g}]} + e^{-ik\pi/m} d_k^{[m,\mathrm{u}]}\right) \tag{2.11}$$

$$d_{m+k}^{[2m]} = \frac{1}{2}\left(d_k^{[m,\mathrm{g}]} - e^{-ik\pi/m} d_k^{[m,\mathrm{u}]}\right) \tag{2.12}$$

für $k = 0, \ldots, m-1$. Dies folgt direkt aus der Matrix–Multiplikation, denn danach gilt für $\bar{\omega} = e^{-i2\pi/n} = e^{-i\pi/m}$

$$\begin{aligned} d_k^{[2m]} &= \frac{1}{n}\sum_{j=0}^{n-1} f_j \bar{\omega}^{jk} \\ &= \frac{1}{2m}\left(\sum_{\substack{j=0 \\ j \text{ gerade}}}^{n-1} f_j \bar{\omega}^{jk} + \sum_{\substack{j=0 \\ j \text{ ungerade}}}^{n-1} f_j \bar{\omega}^{jk}\right) \\ &= \frac{1}{2}\left(\underbrace{\frac{1}{m}\sum_{j=0}^{m-1} f_{2j}\bar{\omega}^{2jk}}_{=d_k^{[m,\mathrm{g}]}} + \bar{\omega}^k \underbrace{\frac{1}{m}\sum_{j=0}^{m-1} f_{2j+1}\bar{\omega}^{2jk}}_{=d_k^{[m,\mathrm{u}]}}\right), \end{aligned}$$

da $\bar{\omega}^{2jk} = \tilde{\omega}^{jk}$ ist mit $\tilde{\omega} = \bar{\omega}^2 = e^{-i2\pi/m}$. Dies zeigt (2.11).

Analog erhalten wir (2.12) aus

$$\begin{aligned} d_{m+k}^{[2m]} &= \frac{1}{n}\sum_{j=0}^{n-1} f_j \bar{\omega}^{jk} \\ &= \frac{1}{2}\left(\underbrace{\frac{1}{m}\sum_{j=0}^{m-1} f_{2j}\bar{\omega}^{2j(m+k)}}_{=d_k^{[m,\mathrm{g}]}} + \underbrace{\bar{\omega}^{m+k}}_{=-\bar{\omega}^k} \underbrace{\frac{1}{m}\sum_{j=0}^{m-1} f_{2j+1}\bar{\omega}^{2j(m+k)}}_{=d_k^{[m,\mathrm{u}]}}\right), \end{aligned}$$

da

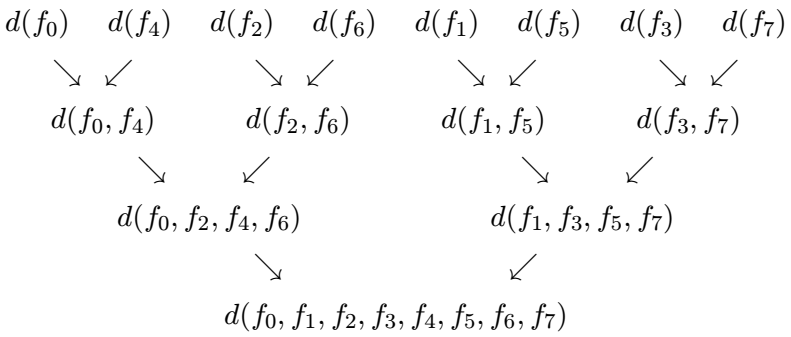
$$\bar{\omega}^{m+k} = \bar{\omega}^m \bar{\omega}^k = \underbrace{e^{-i\pi}}_{=-1} \bar{\omega}^k = -\bar{\omega}^k$$

und

$$\bar{\omega}^{2j(m+k)} = \bar{\omega}^{2jm} \bar{\omega}^{2jk} = (\underbrace{e^{-i2\pi}}_{=1})^m \bar{\omega}^{2jk} = \bar{\omega}^{2jk}$$

ist.

Da die Fourier–Transformation für Datensätze der Länge 1 einfach $d_0 = f_0$ ist, führen die Gleichungen (2.11) und (2.12) auf das folgende Schema zur Berechnung der Fourier–Koeffizienten, in dem $d(\ldots)$ die Fourier–Koeffizienten für den in Klammern angegebenen Datensatz bezeichnet.



Da die Fourier–Rücktransformation auf einer Matrix–Multiplikation mit der gleichen Struktur beruht, kann man diese durch ein ähnliches Schema durchführen. Die eigentliche Implementierung dieses Schemas ist nicht ganz einfach, da die Sortierung der Koeffizienten effizient durchgeführt werden muss; wir wollen diesen Aspekt hier aber nicht vertiefen. In MATLAB ist eine Version dieses Algorithmus implementiert, die auch für Datensätze funktioniert, bei denen die Anzahl der Daten n keine Zweierpotenz ist, die allerdings für Nicht–Zweierpotenzen etwas langsamer ist als für Zweierpotenzen. In der MAPLE–Implenentierung muss immer $n = 2^m$ für ein $m \in \mathbb{N}$ gelten.

Beachte, dass auf jeder Ebene des Schemas $O(n)$ Operationen durchgeführt werden müssen, da jeweils n Werte zu berechnen sind: Zwar nimmt die Anzahl der Datensätze nach unten ab, dafür nimmt die Zahl der zu berechnenden Koeffizienten im gleichen Maße zu. Die Anzahl der Ebenen ist dabei gerade $\log_2 n$. Man sagt in diesem Fall, dass das Verfahren die Ordnung $O(n \log n)$ hat. Die Anzahl der Operationen wächst mit wachsendem n also etwas schneller als linear ($O(n)$), aber deutlich langsamer als quadratisch ($O(n^2)$).

2.3.3 Anwendungen

Wie bereits erwähnt ist die trigonometrische Interpolation nicht die Hauptanwendung der Fourier–Transformation. Die Form der trigonometrischen Polynome ist allerdings hilfreich zum Verständnis einiger wesentlicher Anwendungen, die wir hier kurz skizzieren werden. Wir beschränken uns auf den reellen Fall, d.h. wir nehmen an, dass das trigonometrische

Polynom in der Form

$$T(x) \;=\; a_0 + \sum_{k=1}^{N} 2a_k \cos\left(k2\pi\frac{x-x_0}{p}\right) - \sum_{k=1}^{N} 2b_k \sin\left(k2\pi\frac{x-x_0}{p}\right).$$

mit $d_k = a_k + ib_k$ vorliegt. Wir nehmen weiterhin an, dass die Interpolationsdaten aus einer Funktion f mittels $f_j = f(x_j)$ erzeugt wurden, wobei f eine periodische Funktion mit Periode $p > 0$ sei, d.h. es gelte $f(x+p) = f(x)$ für alle $x \in \mathbb{R}$.

Solche f treten vorwiegend in der Signalverarbeitung auf, wobei sie z.B. Signale beschreiben, die über eine Datenleitung übermittelt werden (x wäre in dieser Interpretation eine Zeitvariable).

Wir werden die Tatsache, dass $T(x)$ die Funktion $f(x)$ approximiert in keiner der folgenden Anwendungen explizit benötigen. Es muss aber betont werden, dass diese Tatsache wesentlich dafür ist, dass diese Anwendungen funktionieren.

Frequenzanalyse

Im Gegensatz zu den Koeffizienten in üblichen Polynomen oder Splines haben die Fourier–Koeffizienten der trigonometrischen Polynome eine klare physikalische Bedeutung: Jeder der Fourier–Koeffizienten a_k bzw. b_k gehört zu einer bestimmten Frequenz im Sinus– bzw. Kosinus. Schreiben wir kurz

$$c_{k,p}(x) = \cos\left(k2\pi\frac{x-x_0}{p}\right) \;\text{ und }\; s_{k,p}(x) = \sin\left(k2\pi\frac{x-x_0}{p}\right)$$

so gilt $c_{k,p}(x) = c_{k,p}(x+q)$ und $s_{k,p}(x) = s_{k,p}(x+q)$ für $q = p/k$, die Funktionen sind also q–periodisch. Wenn wir x als Zeit in der Einheit Sekunden interpretieren ergibt sich so für diese Funktionen eine Wiederholfrequenz von k/p Hertz. Die Größe der Koeffizienten a_k und b_k gibt also an, wie stark der Anteil einer Schwingung von k/p Hz in dem Signal f ist.

Stellt man die Größe der Fourier–Koeffizienten grafisch in Abhängigkeit von der Frequenz dar, kann man an der entstehenden Grafik erkennen, welche Frequenzen in der betrachteten Funktion vorhanden sind. Als "Größe" betrachtet man dabei die Werte $2\sqrt{a_j^2 + b_j^2}$, die gerade der skalierten (komplexen) Norm $2|d_k|$ des Fourier–Koeffizienten d_k entspricht. Die Skalierung bewirkt hierbei, dass eine einfache Sinus–Schwingung der Frequenz k/p Hz gerade dem Wert $2|d_k| = 1$ entspricht. Die in dem so erhaltenen Diagramm dargestellte Funktion nennt man "Spektraldichte".

Nehmen wir als Beispiel die Funktion

$$f(x) = \sin(2\pi\, 50x) + \sin(2\pi\, 120x).$$

Hier überlagern sich zwei Schwingungen mit Frequenzen 50 Hz und 120 Hz, was in der grafischen Darstellung der Funktion in Abbildung 2.2 links in etwa zu erkennen ist. In der Darstellung der Spektraldichte in Abbildung 2.2 rechts hingegen sind die zwei Frequenzen deutlich zu erkennen.

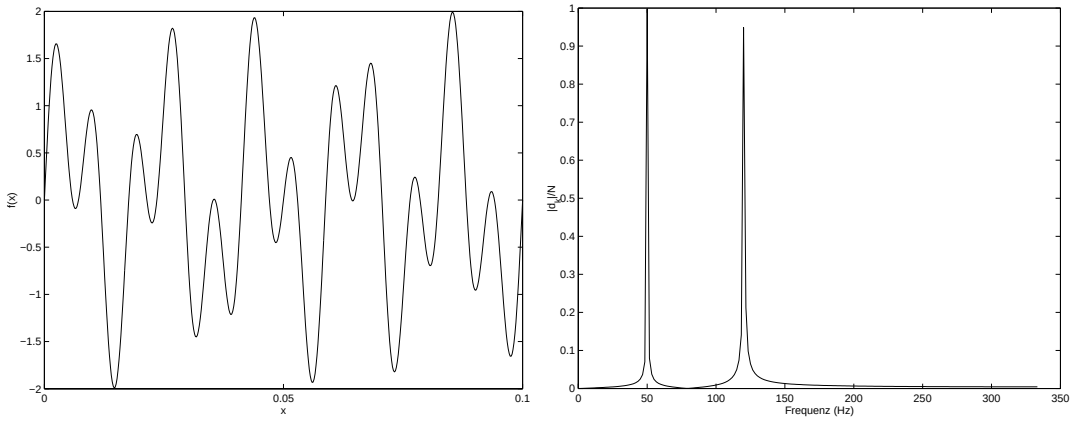


Abbildung 2.2: Graph und Fourier–Koeffizienten von $f(x) = \sin(2\pi 50x) + \sin(2\pi 120x)$

Die Leistungsfähigkeit dieser Frequenzanalyse wird besonders deutlich, wenn wir eine zufällige Störung $g(x)$ zu der Funktion hinzuaddieren (hierbei ist $g(x)$ für jedes x eine normalverteilter Zufallswert mit Erwartungswert 0). Solche Störungen sind bei der Signalübertragung praktisch unvermeidbar, da in jeder Form der technischen Übermittlung Störsignale auftreten, wenn auch nicht immer so drastisch wie in unserem Modellbeispiel hier. Aus dem Graphen der so gestörten Funktion in Abbildung 2.3 links ist praktisch nichts mehr zu erkennen, in den Fourier–Koeffizienten hingegen in Abbildung 2.3 rechts sind die zwei Frequenzen des Original–Signals hingegen immer noch gut zu erkennen.

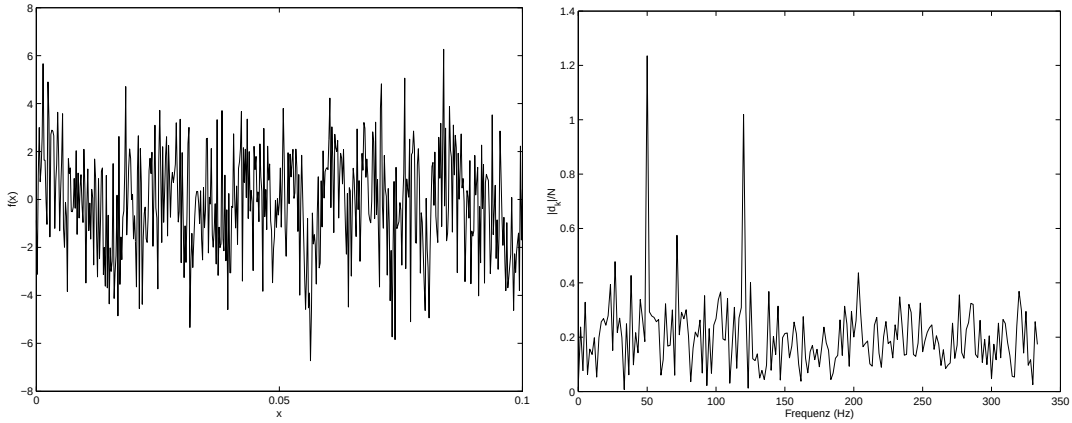


Abbildung 2.3: Graph und Fourier–Koeff. von $f(x) = \sin(2\pi 50x) + \sin(2\pi 120x) + g(x)$

Filterung

Wie wir oben gesehen haben, hat die Fourier–Transformation die Eigenschaft, dass man selbst bei gestörten Daten die "Haupt"frequenzen noch erkennen kann. Diese Eigenschaft kann man zum Filtern gestörter Daten verwenden.

Wir beschreiben hier ein ganz einfaches Verfahren, einen sogenannten *Tiefpassfilter*:

Wir betrachten die Funktion $f(x) = \sin(2\pi x)$ und ihre wie oben gestörte Variante $f(x) = \sin(2\pi x) + g(x)$, siehe Abbildung 2.4. Abbildung 2.5 links zeigt die Fourier–Koeffizienten der gestörten Funktion. Wir legen nun einen Schwellenwert s fest und setzen alle Fourier–Koeffizienten, deren Betrag die Ungleichung $2|d_k| < s$ erfüllt, auf Null. Abbildung 2.5 rechts zeigt die so gefilterten Koeffizienten $(\tilde{d}_0, \ldots, \tilde{d}_{n-1})$ für $s = 0.5$. Schließlich machen wir eine Rücktransformation der gefilterten Koeffizienten und stellen den so erhaltenen Datensatz $(\tilde{f}_0, \ldots, \tilde{f}_{n-1})$ grafisch dar (Abbildung 2.6). Das ursprüngliche Signal ist nun wieder deutlich sichtbar.

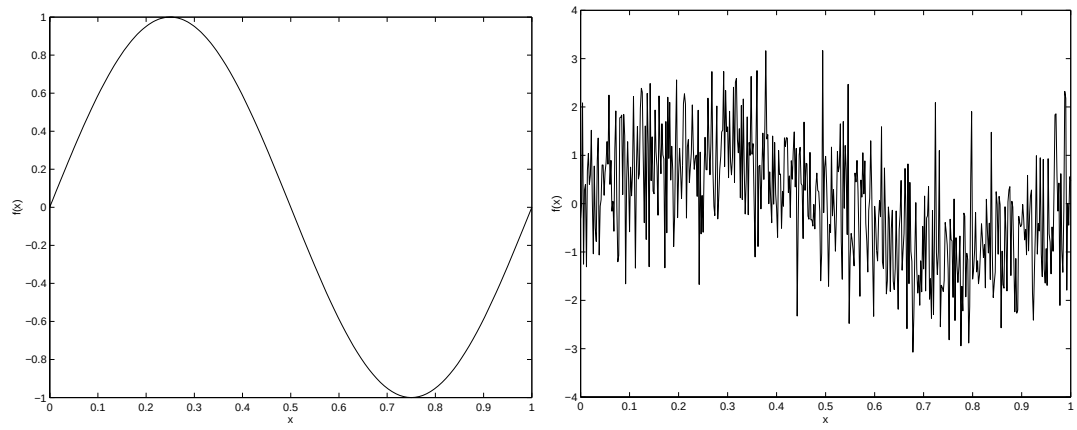


Abbildung 2.4: Graph $f(x) = \sin(2\pi x)$ und $f(x) = \sin(2\pi x) + g(x)$

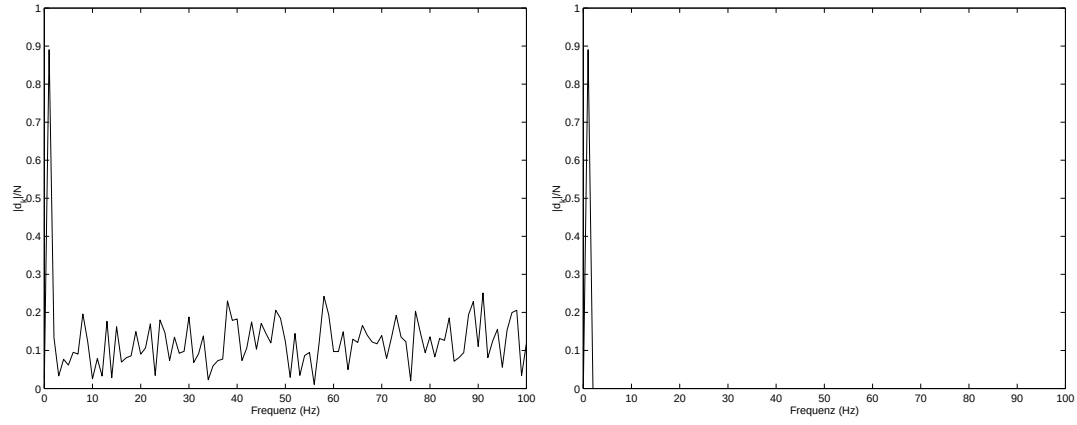


Abbildung 2.5: Original und gefilterte Fourier–Koeffizienten von $f(x) = \sin(2\pi\, x) + g(x)$

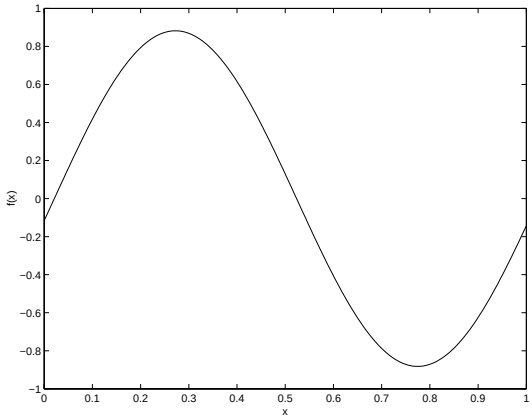


Abbildung 2.6: Rücktransformierte Funktion aus gefilterten Fourier–Koeffizienten

Weitere Anwendungen

Weitere Anwendungen sind z.B. die Zerlegung von Signalen in verschiedene Frequenzbereiche (mittels Fourier–Transformation → Null–Setzen aller Koeffizienten außerhalb des gewünschten Frequenzbereichs → Rücktransformation) oder Datenkompression (mittels Fourier–Transformation → Null–Setzen aller Koeffizienten mit "geringem Beitrag" (bzgl. eines geeigneten Kriteriums, z.B. Berücksichtigung eines vorgegebenen Prozentsatzes der betragsgrößten Koeffizienten) → kompakte Speicherung).

Da viele der hier beschriebenen oder angedeuteten Anwendungen in Echtzeit durchgeführt werden müssen, ist die Notwendigkeit für schnelle Algorithmen und effiziente Implementierung offensichtlich.

Kapitel 3

Integration

Die Integration von Funktionen ist eine elementare mathematische Operation, die in vielen Formeln benötigt wird. Im Gegensatz zur Ableitung, die für praktisch alle mathematischen Funktionen explizit analytisch berechnet werden kann, gibt es viele Funktionen, deren Integrale man nicht explizit angeben kann. Verfahren zur numerischen Integration (man spricht auch von *Quadratur*) spielen daher eine wichtige Rolle, sowohl als eigenständige Algorithmen als auch als Basis für andere Anwendungen wie z.B. der numerischen Lösung von Differentialgleichungen.

Das Problem lässt sich hierbei ganz einfach beschreiben: Für eine Funktion $f : \mathbb{R} \to \mathbb{R}$ soll das Integral

$$\int_a^b f(x)dx \tag{3.1}$$

auf einem Intervall $[a, b]$ berechnet werden.

Aus Zeitgründen können wir das Thema hier nicht in der Breite behandeln, die es eigentlich verdient hätte. Wir beschränken uns daher auf zwei einfache Algorithmen, die nichtsdestotrotz in vielen Anwendungen gute Ergebnisse erzielen, nämlich die *Newton–Cotes–Formeln* und die *zusammengesetzten Newton–Cotes–Formeln*, die auch als *iterierte* oder *aufsummierte* Newton–Cotes–Formeln bezeichnet werden.

3.1 Newton–Cotes–Formeln

Die Grundidee der numerischen Integration liegt darin, das Integral (3.1) durch eine Summe

$$\int_a^b f(x)dx \;\approx\; (b-a)\sum_{i=0}^{n} \alpha_i f(x_i) \tag{3.2}$$

zu approximieren. Hierbei heißen die x_i die *Stützstellen* und die α_i die *Gewichte* der Integrationsformel.

Die Stützstellen x_i können hierbei beliebig vorgegeben werden, folglich benötigen wir eine Formel, mit der wir zu den x_i sinnvolle Gewichte α_i berechnen können.

Die Idee der Newton–Cotes Formeln liegt nun darin, die Funktion F zunächst durch ein Interpolationspolynom P vom Grad n (zu den Daten $(x_i, f(x_i))$, $i = 0, \ldots, n$) zu approximieren und dann das Integral über dieses Polynom zu berechnen. Wir führen diese Konstruktion nun durch:

Da wir einen expliziten Ausdruck in den $f(x_i)$ erhalten wollen, bietet sich die Darstellung von P mittels der Lagrange–Polynome an, also

$$P(x) = \sum_{i=0}^{n} f(x_i) L_i(x)$$

mit

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

vgl. Abschnitt 2.1.1. Das Integral über P ergibt sich dann zu

$$\begin{aligned} \int_a^b P(x)dx &= \int_a^b \sum_{i=0}^{n} f(x_i) L_i(x)\, dx \\ &= \sum_{i=0}^{n} f(x_i) \int_a^b L_i(x)\, dx. \end{aligned}$$

Um die Gewichte α_i in (3.2) zu berechnen, setzen wir

$$(b-a) \sum_{i=0}^{n} \alpha_i f(x_i) = \sum_{i=0}^{n} f(x_i) \int_a^b L_i(x) dx.$$

Auflösen nach α_i liefert dann

$$\alpha_i = \frac{1}{b-a} \int_a^b L_i(x) dx. \tag{3.3}$$

Diese α_i können dann explizit berechnet werden, denn die Integrale über die Lagrange–Polynome L_i sind explizit lösbar. Hierbei hängen die Gewichte α_i von der Wahl der Stützstellen x_i ab, nicht aber von den Funktionswerten $f(x_i)$. Für äquidistante Stützstellen

$$x_i = a + \frac{i(b-a)}{n}$$

sind die Gewichte aus (3.3) in Tabelle 3.1 für $n = 1, \ldots, 7$ angegeben.

Beachte, dass sich die Gewichte immer zu 1 aufsummieren und symmetrisch in i sind, d.h. es gilt $\alpha_i = \alpha_{n-i}$. Außerdem sind die Gewichte unabhängig von den Intervallgrenzen a und b.

Aus der Abschätzung des Interpolationsfehlers kann man eine Abschätzung für den Integrationsfehler

$$F_n[f] := \int_a^b f(x)dx - (b-a) \sum_{i=0}^{n} \alpha_i f(x_i)$$

ableiten. Hierbei müssen die Stützstellen x_i nicht unbedingt äquidistant liegen.

n	α_0	α_1	α_2	α_3	α_4	α_5	α_6	α_7
1	$\frac{1}{2}$	$\frac{1}{2}$						
2	$\frac{1}{6}$	$\frac{4}{6}$	$\frac{1}{6}$					
3	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$				
4	$\frac{7}{90}$	$\frac{32}{90}$	$\frac{12}{90}$	$\frac{32}{90}$	$\frac{7}{90}$			
5	$\frac{19}{288}$	$\frac{75}{288}$	$\frac{50}{288}$	$\frac{50}{288}$	$\frac{75}{288}$	$\frac{19}{288}$		
6	$\frac{41}{840}$	$\frac{216}{840}$	$\frac{27}{840}$	$\frac{272}{840}$	$\frac{27}{840}$	$\frac{216}{840}$	$\frac{41}{840}$	
7	$\frac{751}{17280}$	$\frac{3577}{17280}$	$\frac{1323}{17280}$	$\frac{2989}{17280}$	$\frac{2989}{17280}$	$\frac{1323}{17280}$	$\frac{3577}{17280}$	$\frac{751}{17280}$

Tabelle 3.1: Gewichte der Newton–Cotes Formeln aus (3.3) für äquidistante Stützstellen x_i

Satz 3.1 Seien α_i die Gewichte, die gemäß (3.3) zu den Stützstellen $a \le x_0 < \ldots < x_n \le b$ berechnet wurden und sei $h = (b-a)/n$. Dann gibt es von a, b und f unabhängige Konstanten c_n, so dass für alle $(n+1)$–mal differenzierbaren Funktionen f die Abschätzung

$$|F_n[f]| \le c_n h^{n+2} \max_{y\in[a,b]} |f^{(n+1)}(y)|$$

gilt, wobei $f^{(n+1)}$ wie im Abschnitt 2.1.3 die $(n+1)$–te Ableitung der Funktion f bezeichnet.

Beweis: Aus dem Ansatz zur Berechnung der Gewichte α_i folgt direkt die Gleichung

$$(b-a)\sum_{i=0}^{n} \alpha_i f(x_i) = \int_a^b P(x)dx$$

und damit

$$F_n[f] = \int_a^b f(x)dx - \int_a^b P(x)dx = \int_a^b f(x) - P(x)\,dx.$$

Aus Satz 2.6(i) folgt daher

$$\begin{aligned} |F_n[f]| &= \left| \frac{1}{(n+1)!} \int_a^b f^{(n+1)}(\xi(x)) \prod_{i=0}^{n} (x-x_i)\,dx \right| \\ &\le \frac{1}{(n+1)!} \max_{y\in[a,b]} \left| f^{(n+1)}(y) \right| \int_a^b \prod_{i=0}^{n} |x-x_i|\,dx, \end{aligned}$$

mit Werten $\xi(x) \in (a,b)$. Daraus folgt die behauptete Abschätzung mit

$$\begin{aligned} c_n &= \frac{1}{(n+1)!}\frac{1}{h^{n+2}} \int_a^b \prod_{i=0}^{n} |x-x_i|\,dx = \frac{1}{(n+1)!} \left(\frac{n}{b-a}\right)^{n+2} \int_a^b \prod_{i=0}^{n} |x-x_i|\,dx \\ \left(\text{Substitution: } z = n\tfrac{x-a}{b-a},\ z_i = n\tfrac{x_i-a}{b-a}\right) &= \frac{1}{(n+1)!} \int_0^n \prod_{i=0}^{n} |z-z_i|\,dz \end{aligned}$$

□

Bemerkung 3.2 Für gerades n, $(n+2)$–mal differenzierbares f und symmetrisch verteilte Stützstellen x_i (z.B. äquidistante Stützstellen) lässt sich die bessere Abschätzung

$$|F_n[f]| \leq d_n h^{n+3} \max_{y\in[a,b]} |f^{(n+2)}(y)|$$

beweisen, wobei die d_n ähnlich wie die c_n in Satz 3.1 berechnet werden. □

Die Konstanten c_n und d_n können für gegebenes n explizit berechnet werden. In Tabelle 3.2 sind die darauf basierenden Fehlerabschätzungen für $n = 1, \ldots, 7$ und äquidistante Stützstellen approximativ angegeben, wobei $M_n := \max_{y\in[a,b]} |f^{(n)}(y)|$ ist.

n	1	2	3	4	5	6	7
	$\frac{(b-a)^3 M_2}{12}$	$\frac{(b-a)^5 M_4}{2880}$	$\frac{(b-a)^5 M_4}{6480}$	$\frac{5.2(b-a)^7 M_6}{10^7}$	$\frac{2.9(b-a)^7 M_6}{10^7}$	$\frac{6.4(b-a)^9 M_8}{10^{10}}$	$\frac{3.9(b-a)^9 M_8}{10^{10}}$

Tabelle 3.2: Fehlerabschätzungen der Newton–Cotes Formeln für äquidistante Stützstellen

Vereinfacht gesagt erhöht sich also die Genauigkeit mit wachsendem n, allerdings nur dann, wenn nicht zugleich die höheren Ableitungen $|f^{(n)}|$ zunehmen. Es tauchen also die gleichen prinzipiellen Probleme wie bei den Interpolationspolynomen auf, was nicht weiter verwunderlich ist, da diese ja dem Verfahren zu Grunde liegen. Hier kommt aber noch ein weiteres Problem hinzu, nämlich kann man für $n = 8$ und $n \geq 10$ beobachten, dass einige der Gewichte α_i negative werden. Dies kann zu numerischen Problemen (z.B. Auslöschungen) führen, die man möglichst vermeiden möchte. Aus diesem Grunde ist es nicht ratsam, den Grad des zugrundeliegenden Polynoms immer weiter zu erhöhen. Statt dessen wählt man ein anderes Verfahren, das im folgenden Abschnitt beschrieben ist.

3.2 Zusammengesetzte Newton–Cotes–Formeln

Der Ausweg aus den Problemen mit immer höheren Polynomgraden ist bei der Integration mittels Newton–Cotes–Formeln ganz ähnlich wie bei der Interpolation — nur einfacher. Bei der Interpolation sind wir von Polynomen zu Splines, also stückweisen Polynomen übergegangen. Um dort weiterhin eine "schöne" Approximation zu erhalten, mussten wir Bedingungen an den Nahtstellen festlegen, die eine gewisse Glattheit der approximierenden Funktion erzwingen, weswegen wir die Koeffizienten recht kompliziert über ein lineares Gleichungssystem herleiten mussten.

Bei der Integration fällt diese Prozedur weg. Wie bei den Splines verwenden wir zur Herleitung der zusammengesetzten Newton–Cotes–Formeln stückweise Polynome, verzichten aber auf aufwändige Bedingungen an den Nahtstellen, da wir ja nicht an einer schönen Approximation der Funktion, sondern "nur" an einer guten Approximation des Integrals interessiert sind. In der Praxis berechnet man die zugrundeliegenden stückweisen Polynome nicht wirklich, sondern wendet die Newton–Cotes–Formeln wir folgt auf den Teilintervallen an:

Sei N die Anzahl von Teilintervallen, auf denen jeweils die Newton–Cotes–Formel vom Grad n verwendet werden soll. Wir setzen

$$x_i = a + ih, \;\; i = 0, 1, \ldots, nN, \quad h = \frac{b-a}{nN}$$

und zerlegen das Integral (3.1) mittels

$$\int_a^b f(x)dx \;=\; \int_{x_0}^{x_n} f(x)dx + \int_{x_n}^{x_{2n}} f(x)dx + \cdots + \int_{x_{(N-1)n}}^{x_{Nn}} f(x)dx.$$

Auf jedem Teilintervall $[x_{jn}, x_{(j+1)n}]$ wenden wir nun die Newton–Cotes–Formel an, d.h. wir approximieren

$$\int_{x_{jn}}^{x_{(j+1)n}} f(x)dx \;\approx\; nh\sum_{i=0}^{n}\alpha_i f(x_{jn+i})$$

und addieren die Teilapproximationen auf, also

$$\int_a^b f(x)dx \;\approx\; nh\sum_{j=0}^{N-1}\sum_{i=0}^{n}\alpha_i f(x_{jn+i}).$$

Der entstehende Approximationsfehler

$$F_{N,n}[f] := \int_a^b f(x)dx - nh\sum_{j=0}^{N-1}\sum_{i=0}^{n}\alpha_i f(x_{jn+i})$$

ergibt sich einfach als Summe der Fehler $F_n[f]$ auf den Teilintervallen, weswegen man aus Satz 3.1 die Abschätzung

$$\begin{aligned}|F_{N,n}[f]| &\le \sum_{j=0}^{N-1} c_n h^{n+2}\max_{y\in[x_{jn},x_{(j+1)n}]}|f^{(n+1)}(y)|\\ &\le Nc_nh^{n+2}\max_{y\in[a,b]}|f^{(n+1)}(y)| \;=\; \frac{c_n}{n}(b-a)h^{n+1}\max_{y\in[a,b]}|f^{(n+1)}(y)|\end{aligned}$$

und aus Bemerkung 3.2 für gerades n die Abschätzung

$$|F_{N,n}[f]| \le \frac{d_n}{n}(b-a)h^{n+2}\max_{y\in[a,b]}|f^{(n+2)}(y)|$$

erhält. Im Folgenden geben wir die zusammengesetzten Newton–Cotes–Formeln für $n = 1, 2, 4$ mitsamt ihren Fehlerabschätzungen an; in allen Formeln sind die Stützstellen x_i als $x_i = a + ih$ gewählt.

$n = 1$**, Trapez–Regel:**

$$\int_a^b f(x)dx \;\approx\; \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{N-1} f(x_i) + f(b)\right)$$

$$|F_{N,1}[f]| \le \frac{b-a}{12}h^2\max_{y\in[a,b]}|f^{(2)}(y)|, \quad h = \frac{b-a}{N}$$

$n = 2$**, Simpson–Regel:**

$$\int_a^b f(x)dx \;\approx\; \frac{h}{3}\left(f(a) + 2\sum_{i=1}^{N-1} f(x_{2i}) + 4\sum_{i=0}^{N-1} f(x_{2i+1}) + f(b)\right)$$

$$|F_{N,2}[f]| \leq \frac{b-a}{180} h^4 \max_{y\in[a,b]} |f^{(4)}(y)|, \quad h = \frac{b-a}{2N}$$

$n = 4$**, Milne–Regel:**

$$\begin{aligned}\int_a^b f(x)dx \;\approx\; & \frac{2h}{45}\left(7\Big(f(a)+f(b)\Big) + 14\sum_{i=1}^{N-1} f(x_{4i})\right.\\ & \left. +\; 32\sum_{i=0}^{N-1}\Big(f(x_{4i+1})+f(x_{4i+3})\Big) + 12\sum_{i=0}^{N-1} f(x_{4i+2})\right)\end{aligned}$$

$$|F_{N,4}[f]| \leq \frac{2(b-a)}{945} h^6 \max_{y\in[a,b]} |f^{(6)}(y)|, \quad h = \frac{b-a}{4N}$$

Zum Abschluss wollen wir an einem Beispiel die praktischen Auswirkungen der Fehlerabschätzungen illustrieren.

Beispiel 3.3 Das Integral

$$\int_0^1 e^{-x^2/2} dx$$

soll mit einer garantierten Genauigkeit von $\varepsilon = 10^{-10}$ numerisch approximiert werden. Die Ableitungen der Funktion $f(x) = e^{-x^2/2}$ lassen sich leicht berechnen; es gilt

$$f^{(2)}(x) = (x^2-1)f(x), \;\; f^{(4)}(x) = (3-6x^2+x^4)f(x), \;\; f^{(6)}(x) = (-15+45x^2-15x^4+x^6)f(x).$$

Mit etwas Rechnung (oder aus der grafischen Darstellung) sieht man, dass all diese Funktionen ihr betragsmäßiges Maximum auf $[0,1]$ in $y = 0$ annehmen, woraus die Gleichungen

$$\max_{y\in[0,1]} |f^{(2)}(y)| = 1, \;\; \max_{y\in[0,1]} |f^{(4)}(y)| = 3 \;\; \text{und} \;\; \max_{y\in[0,1]} |f^{(6)}(y)| = 15$$

folgen.

Löst man die oben angegebenen Fehlerabschätzungen $F_{N,n}[f] \leq \varepsilon$ für die Trapez–, Simpson– und Milne–Regel nach h auf und setzt die Abschätzungen für $\max_{y\in[0,1]} |f^{(n)}(y)|$ ein, so erhält man die folgenden Bedingungen an h

$$h \;\leq\; \sqrt{\frac{12\varepsilon}{(b-a)|f^{(2)}(0)|}} \;\approx\; \frac{1}{28867.51} \qquad \text{(Trapez–Regel)}$$

$$h \;\leq\; \sqrt[4]{\frac{180\varepsilon}{(b-a)|f^{(4)}(0)|}} \;\approx\; \frac{1}{113.62} \qquad \text{(Simpson–Regel)}$$

$$h \;\leq\; \sqrt[6]{\frac{945\varepsilon}{2(b-a)|f^{(6)}(0)|}} \;\approx\; \frac{1}{10.59} \qquad \text{(Milne–Regel)}$$

Der Bruch auf der rechten Seite gibt dabei die maximal erlaubte Größe für h vor. Um diese zu realisieren, muss $1/(nN) \leq h$ gelten für die Anzahl $nN+1$ der Stützstellen. Da nN ganzzahlig ist, braucht man also 28869 Stützstellen für die Trapez–Regel, 115 Stützstellen für die Simpson–Regel und 13 Stützstellen für die Milne–Regel. □

Kapitel 4

Nichtlineare Gleichungen und Gleichungssysteme

Nichtlineare Gleichungen oder Gleichungssysteme müssen in vielen Anwendungen der Mathematik gelöst werden. Typischerweise werden die Lösungen nichtlinearer Gleichungen über die Nullstellen einer Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$ definiert, für die dann ein $x^* \in \mathbb{R}^n$ mit

$$f(x^*) = 0$$

gesucht wird.

Beispiel (Berechnung von Quadratwurzeln): Berechne $x^* \in \mathbb{R}$ mit $f(x) = 0$ für $f(x) = x^2 - 2$. Die Lösung ist entweder $\sqrt{2}$ oder $\sqrt{-2}$; ein numerisches Verfahren zur Berechnung von Nullstellen kann also insbesondere zur Berechnung von Quadratwurzeln verwendet werden.

Wir werden uns hier zunächst mit dem Spezialfall $n = 1$ beschäftigen und den mehrdimensionalen Fall $n > 1$ am Ende des Kapitels behandeln. Wir beginnen das Kapitel mit einem sehr einfachen und anschaulichen Verfahren.

4.1 Das Bisektionsverfahren

Wir betrachten hier eine stetige reelle Funktion $f : \mathbb{R} \to \mathbb{R}$ und suchen eine Nullstelle, also einen Wert $x^* \in \mathbb{R}$ mit $f(x^*) = 0$. Der folgende Algorithmus berechnet solch eine Nullstelle.

Algorithmus 4.1 (Bisektionsverfahren) Gegeben seien eine Funktion $f : \mathbb{R} \to \mathbb{R}$ und Werte $a < b$, so dass $f(a)f(b) < 0$ gilt (d.h., $f(a)$ und $f(b)$ haben unterschiedliches Vorzeichen). Weiterhin sei eine gewünschte Genauigkeit $\varepsilon > 0$ gegeben.

(0) Setze $i = 0$ (Zählindex) und $a_0 = a$, $b_0 = b$.

(1) Setze $x_i = a_i + (b_i - a_i)/2$.

(2) Falls $f(x_i) = 0$ oder $(b_i - a_i)/2 < \varepsilon$ beende den Algorithmus

(3) Falls $f(x_i)f(a_i) < 0$ ist setze $a_{i+1} = a_i$, $b_{i+1} = x_i$
Falls $f(x_i)f(a_i) > 0$ ist setze $a_{i+1} = x_i$, $b_{i+1} = b_i$
Setze $i = i + 1$ und gehe zu Schritt (1).

□

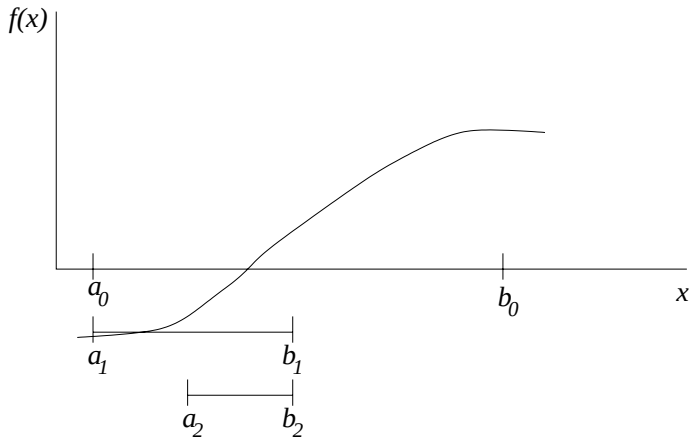


Abbildung 4.1: Bisektionsverfahren

Abbildung 4.1 illustriert dieses Verfahren. Man kann die Punkte a_i, b_i als Intervallgrenzen der Intervalle $[a_i, b_i]$ verstehen, mit denen die Nullstelle durch immer weitere Halbierung eingeschachtelt wird. Daher stammt der Name "Bisektion" (= Zweiteilung).

Die Auswahlbedingung der neuen Werte a_{i+1} und b_{i+1} stellt sicher, dass $f(a_{i+1})$ und $f(b_{i+1})$ unterschiedliches Vorzeichen haben; deswegen muss (da f stetig ist) sich eine Nullstelle zwischen diesen Werten befinden. Wenn die Abbruchbedingung $(x_i - a_i) < \varepsilon$ erreicht ist, ist also sichergestellt, dass es eine Nullstelle x^* mit $|x^* - x_i| < \varepsilon$ gibt, dass also x_i eine approximative Nullstelle ist.

Das Bisektionsverfahren hat einige sehr vorteilhafte Eigenschaften:

- Es funktioniert für allgemeine stetige Funktionen.
- Es liefert immer ein Ergebnis, vorausgesetzt dass man geeignete Startwerte a und b finden kann (man sagt, dass das Verfahren "global konvergiert").
- Die Anzahl der Schritte, nach der die gewünschte Genauigkeit erreicht ist, hängt nur von a und b aber nicht von f ab.

Der Grund, warum in der Praxis trotzdem oft andere Verfahren eingesetzt werden, liegt darin, dass das Verfahren relativ langsam gegen den gesuchten Wert x^* konvergiert. Um dies zu verstehen, müssen wir zunächst geeignete Konzepte zum Messen von Konvergenzgeschwindigkeiten einführen.

4.2 Konvergenzordnung

Der Begriff der Konvergenzordnung liefert eine Möglichkeit, iterative Verfahren auf ihre Geschwindigkeit hin zu untersuchen. Wir werden hier drei verschiedene Konvergenzordnungen betrachten: Lineare Konvergenz, superlineare Konvergenz und quadratische Konvergenz.

Iterative Verfahren liefern eine Folge approximativer Lösungen x_i, die gegen die exakte Lösung x^* konvergieren. Die Konvergenzordnung wird über den Fehler

$$\|x_i - x^*\|$$

definiert und gibt an, wie schnell dieser Fehler gegen Null konvergiert. Wir verwenden hier die Norm $\|\cdot\|$, weil wir dieses Konzept gleich allgemein für beliebige Algorithmen definieren wollen; für x_i, $x^* \in \mathbb{R}$ ist dies einfach der Betrag $|x_i - x^*|$.

Die folgende Definition beschreibt die drei Arten der Konvergenzordnung, die wir hier betrachten wollen.

Definition 4.2 Betrachte ein iteratives Verfahren, dass eine Folge von approximativen Lösungen x_i für die exakte Lösung x^* liefert. Dann definieren wir die folgenden Konvergenzordnungen:

(i) Das Verfahren heißt *linear konvergent*, falls eine Konstante $c \in (0, 1)$ existiert, so dass die Abschätzung

$$\|x_{i+1} - x^*\| \leq c\|x_i - x^*\| \text{ für alle } i = 0, 1, 2, \ldots$$

gilt.

(ii) Das Verfahren heißt *superlinear konvergent*, falls Konstanten $c_i \in (0, 1)$ für $i = 0, 1, 2, \ldots$ existieren, so dass die Bedingungen

$$c_{i+1} \leq c_i,\ i = 0, 1, 2, \ldots, \quad \lim_{i \to \infty} c_i = 0$$

und die Abschätzung

$$\|x_{i+1} - x^*\| \leq c_i\|x_i - x^*\| \text{ für alle } i = 0, 1, 2, \ldots$$

gelten.

(iii) Das Verfahren heißt *quadratisch konvergent*, falls eine Konstante $q > 0$ existiert, so dass die Abschätzung

$$\|x_{i+1} - x^*\| \leq q\|x_i - x^*\|^2 \text{ für alle } i = 0, 1, 2, \ldots$$

gilt.

□

Bemerkung 4.3 Durch iterative Anwendung dieser Ungleichungen erhält man die Fehlerabschätzungen

$$\begin{aligned}\|x_{i+1} - x^*\| &\leq c^i \|x_0 - x^*\| \\ \|x_{i+1} - x^*\| &\leq \prod_{k=0}^{i} c_i \|x_0 - x^*\| \\ \|x_{i+1} - x^*\| &\leq \frac{1}{q}(q\|x_0 - x^*\|)^{2^i}.\end{aligned}$$

Beachte, dass die dritte Ungleichung nur dann eine sinnvolle Fehlerabschätzung liefert, wenn $q\|x_0 - x^*\| < 1$ bzw. $\|x_0 - x^*\| < 1/q$ gilt, d.h. wenn der Anfangswert x_0 bereits nahe genug am exakten Ergebnis x^* liegt. □

Abbildung 4.2 zeigt die Abhängigkeit der Fehler für $c = 0.5$, $c_i = \frac{4}{(i+1)^2+7}$, $q = 2$ und $\|x_0 - x^*\| = 1/4$.

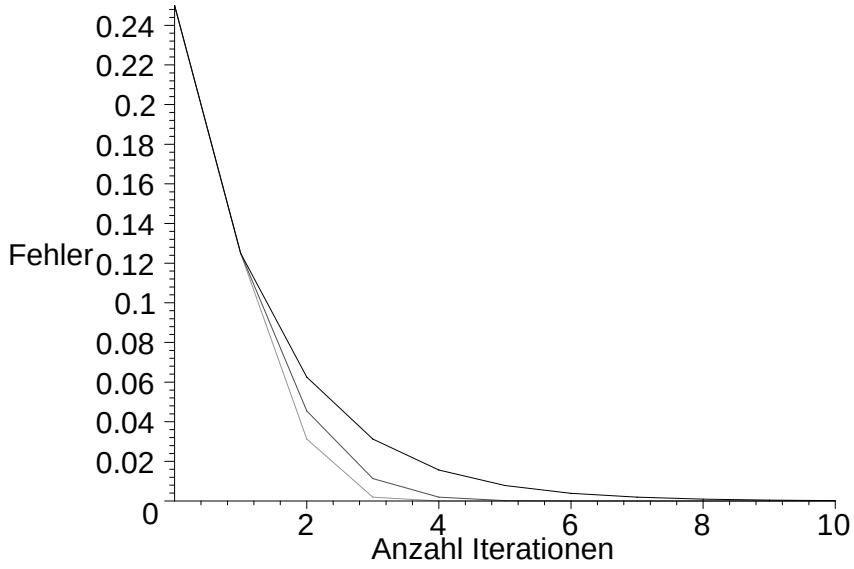


Abbildung 4.2: Konvergenzordnungen: linear, superlinear und quadratisch (von oben nach unten)

Zwar kann man erkennen, dass die quadratische Konvergenz schneller gegen Null tendiert als die superlineare, und diese wiederum als die lineare, allerdings lässt sich aus dieser Grafik nicht direkt erkennen, welche Konvergenzordnung einer bestimmten Kurve zu Grunde liegt.

Dies lässt sich viel leichter feststellen, wenn man statt des Fehlers den *Logarithmus des Fehlers* betrachtet. Für den Logarithmus gelten die Rechenregeln

$$\log(ab) = \log(a) + \log(b),\ \log(1/q) = -\log(q) \quad \text{und} \quad \log(c^d) = d\log(c).$$

Damit erhalten wir für die drei Konvergenzarten aus Definition 4.2 die Ungleichungen

$$\begin{aligned} \log(\|x_{i+1} - x^*\|) &\leq \log(\|x_i - x^*\|) + \log(c) \\ \log(\|x_{i+1} - x^*\|) &\leq \log(\|x_i - x^*\|) + \log(c_i) \\ \log(\|x_{i+1} - x^*\|) &\leq 2\log(\|x_i - x^*\|) + \log(q). \end{aligned} \tag{4.1}$$

und mit der Abkürzung $K = \log(\|x_0 - x^*\|)$ aus Bemerkung 4.3 die Abschätzungen

$$\begin{aligned} \log(\|x_{i+1} - x^*\|) &\leq i\log(c) + K \\ \log(\|x_{i+1} - x^*\|) &\leq \sum_{k=0}^{i} \log(c_i) + K \\ \log(\|x_{i+1} - x^*\|) &\leq 2^i(\log(q) + K) - \log(q). \end{aligned}$$

Beachte, dass der Logarithmus dabei gegen minus unendlich strebt, wenn der Fehler gegen Null konvergiert.

Wenn man nun diese letzten drei Abschätzungen grafisch darstellt, so erhält man im linearen Fall eine Gerade mit negativer Steigung, im quadratischen Fall eine Kurve der Form $i \mapsto -C2^i + D$ für $C > 0$, $D \in \mathbb{R}$ und im superlinearen Fall eine dazwischenliegende Kurve, deren Neigung immer weiter zunimmt, die also negative Krümmung besitzt. Abbildung 4.3 zeigt das typische Verhalten dieser Kurven für den Logarithmus der Basis 10.

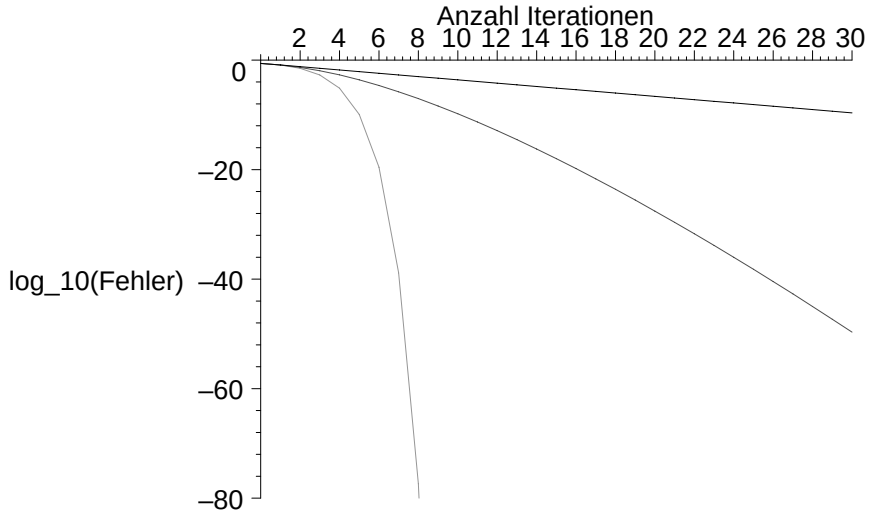


Abbildung 4.3: Konvergenzordnungen: linear, superlinear und quadratisch (von oben nach unten)

Tatsächlich würden diese Kurven für jeden anderen Logarithmus ähnlich aussehen, der Logarithmus zur Basis 10 hat aber die spezielle Eigenschaft, dass man die Genauigkeit direkt ablesen kann, wenn man sie in der Anzahl der korrekten Nachkommastellen des Ergebnisses misst, zumindest im eindimensionalen Fall, also für $x \in \mathbb{R}$:

Sei dazu $d = \log_{10}(|x_i - x^*|)$. Wir nehmen an, dass d negativ ist, was genau dann der Fall ist, wenn $|x_i - x^*| < 1$ ist (die folgenden Überlegungen gelten also nur, wenn x_i bereits hinreichend nahe an x^* liegt). Sei nun $m > 0$ die größte ganze Zahl, die echt kleiner als $-d$ ist. Dann gilt

$$|x_i - x^*| = 10^d < 10^{-m} = 0.\underbrace{0\cdots0}_{(m-1)-\text{mal}}1.$$

Jede Zahl, die kleiner als $10^{-m} =$ ist, ist von der Form

$$0.\underbrace{0\cdots0}_{m-\text{mal}}***\cdots,$$

wobei der Stern "$*$" beliebige Ziffern symbolisiert. Also ist

$$|x_i - x^*| \leq 0.\underbrace{0\cdots0}_{m-\text{mal}}***\cdots,$$

weswegen x_i und x^* mindestens in den ersten m Nachkommastellen übereinstimmen müssen. Auf diese Weise lässt sich aus d die Anzahl der korrekten Nachkommastellen des Ergebnisses direkt ablesen.

Aus den Ungleichungen (4.1) kann man daraus die Konsequenzen ableiten, dass sich die Anzahl der korrekten Stellen bei linearer Konvergenz etwa alle $1/(-\log(c))$ Schritte um 1 erhöht und bei quadratischer Konvergenz (wenn man den $\log(q)$ Summanden vernachlässigt) in jedem Schritt in etwa verdoppelt. Superlineare Konvergenz liegt auch hier zwischen diesen Werten: Die Anzahl der neu hinzukommenden korrekten Stellen nimmt in jedem Schritt zu.

In Tabelle 4.1 werden die Charakteristika der verschiedenen betrachteten Konvergenzordnungen noch einmal zusammengefasst.

Konvergenzordnung	linear	superlinear	quadratisch
Definition	$\|x_{i+1} - x^*\|$ $\leq c\|x_i - x^*\|$ für ein $c \in (0,1)$	$\|x_{i+1} - x^*\|$ $\leq c_i\|x_i - x^*\|$ für $c_i \in (0,1)$, $c_i \searrow 0$	$\|x_{i+1} - x^*\|$ $\leq q\|x_i - x^*\|^2$ für ein $q > 0$
Kurve im log–Diagramm	Gerade	Kurve mit negativer Krümmung	$\approx i \mapsto -C2^i + D$
Anzahl korrekter Nachkommastellen	erhöht sich um 1 nach ca. $1/(-\log(c))$ Schritten	wie linear, aber mit immer schnellerer Zunahme	verdoppelt sich ca. nach jedem Schritt

Tabelle 4.1: Charakteristika verschiedener Konvergenzordnungen

Wir kommen nun zurück zu dem Bisektionsverfahren aus Algorithmus 4.1. Bei diesem Verfahren kann der Fall eintreten, dass der Wert x_i zufällig sehr nahe an der gesuchten Nullstelle liegt, und sich in weiteren Iterationsschritten zunächst wieder entfernt, bevor er letztendlich konvergiert. Tatsächlich sollte man hier den Fehler nicht über den Abstand $|x_i - x^*|$ sondern über die Intervallgröße $(b_i - a_i)$ definieren, da aus der Konstruktion sofort

die Abschätzung $|x_i - x^*| \leq (b_i - a_i)/2$ folgt. Diese Intervallgröße halbiert sich in jedem Schritt; man erhält also

$$(b_{i+1} - a_{i+1}) \leq \frac{1}{2}(b_i - a_i)$$

und damit lineare Konvergenz mit $c = 1/2$. In den folgenden Abschnitten werden wir Verfahren erläutern, die quadratisch oder zumindest superlinear konvergieren.

Bemerkung 4.4 Wir haben bereits im Kapitel über lineare Gleichungssysteme iterative Verfahren, nämlich das Jacobi– und das Gauss–Seidel–Verfahren kennen gelernt. Diese Verfahren haben beide lineare Konvergenzordnung. □

4.3 Das Newton–Verfahren

In diesem Abschnitt werden wir ein weiteres Verfahren zur Lösung nichtlinearer Gleichungen betrachte, das Newton–Verfahren.

Im Gegensatz zum Bisektions–Verfahren wird hier nicht nur die Funktion f selbst sondern auch ihre Ableitung benötigt. Wir beschreiben das Verfahren zunächst formal und betrachten dann die Darstellung der Idee.

Algorithmus 4.5 (Newton–Verfahren) Gegeben sei eine Funktion $f : \mathbb{R} \to \mathbb{R}$, ihre Ableitung $f' : \mathbb{R} \to \mathbb{R}$ sowie ein Anfangswert $x_0 \in \mathbb{R}$ und eine gewünschte Genauigkeit $\varepsilon > 0$. Setze $i = 0$.

(1) Berechne $x_{i+1} = x_i - f(x_i)/f'(x_i)$

(2) Falls $|x_{i+1} - x_i| < \varepsilon$, beende den Algorithmus,
ansonsten setzte $i = i + 1$ und gehe zu (1)

□

Die Idee des Newton–Verfahrens ist wie folgt: Berechne die Tangente $g(x)$ von f im Punkt x_i, d.h. die Gerade

$$g(x) = f(x_i) + f'(x_i)(x - x_i)$$

und wähle x_{i+1} als Nullstelle von g, also

$$f(x_i) + f'(x_i)(x_{i+1} - x_i) = 0 \;\Leftrightarrow\; f'(x_i)x_{i+1} = f'(x_i)x_i - f(x_i) \;\Leftrightarrow\; x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

Die Idee ist in Abbildung 4.4 grafisch dargestellt.

Der folgende Satz zeigt die Konvergenzordnungen dieses Verfahrens.

Satz 4.6 Sei $f : \mathbb{R} \to \mathbb{R}$ stetig differenzierbar und sei x_0 hinreichend nahe bei x^*. Weiterhin gelte $f'(x) \neq 0$. Dann konvergiert das Newton–Verfahren superlinear.

Ist f zwei mal stetig differenzierbar, so konvergiert das Newton–Verfahren sogar quadratisch.

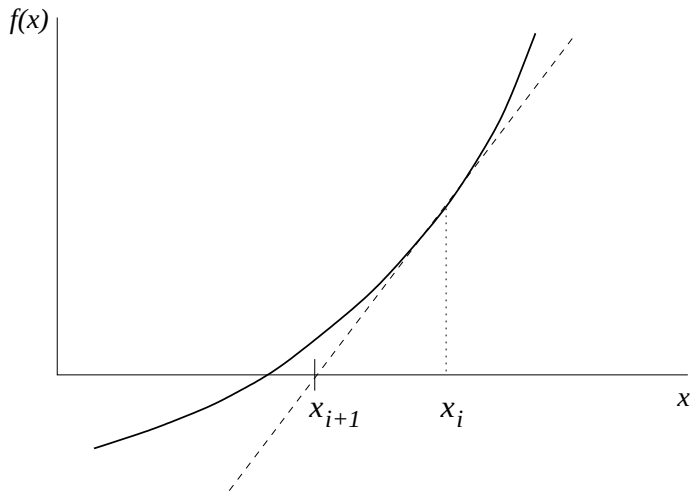


Abbildung 4.4: Newton–Verfahren

Beweis: Der Beweis beruht auf einem Satz aus der Analysis, dem sogenannten *Taylor–Satz*.

Falls f stetig differenzierbar ist, besagt dieser Satz (in diesem Fall auch *Mittelwertsatz* genannt), dass für je zwei Werte $x, y \in \mathbb{R}$ ein $\xi \in [x, y]$ bzw. $\xi \in [y, x]$ (je nach dem ob $x < y$ oder $y < x$ ist) existiert mit

$$f(x) = f(y) + f'(\xi)(x - y). \tag{4.2}$$

Falls f zwei mal stetig differenzierbar ist, folgt die Existenz von ξ mit

$$f(x) = f(y) + f'(y)(x - y) + \frac{1}{2} f''(\xi)(x - y)^2. \tag{4.3}$$

Wir betrachten nun den ersten Fall: Sei $\delta > 0$ so klein, dass $|f'(y_1)/f'(y_2)| < 1/2$ ist für alle $y_1, y_2 \in \mathbb{R}$ mit $|y_1 - x^*| \leq \delta$ und $|y_2 - x^*| \leq \delta$ und sei $|x_0 - x^*| \leq \delta$. Dann erhalten wir unter Anwendung von (4.2) mit $x = x_i$ und $y = x^*$ und wegen $f(x^*) = 0$ die Gleichung

$$|x_{i+1} - x^*| = \left| x_i - \frac{f(x_i)}{f'(x_i)} - x^* \right| = \underbrace{\left| 1 - \frac{f'(\xi)}{f'(x_i)} \right|}_{=:c_i} |x_i - x^*|.$$

Da $|x_i - x^*|$ immer kleiner wird, liegen ξ und x_i immer näher beieiander, also liegt

$$c_i = \left| 1 - f'(\xi)/f'(x_i) \right| = \left| \frac{f'(x_i) - f'(\xi)}{f'(x_i)} \right|$$

immer näher bei 0. Damit folgt die superlineare Konvergenz.

Im zweiten Fall, d.h. falls f zwei mal stetig differenzierbar ist wählen wir $\delta > 0$ so klein, dass $\delta^2 < 2|f'(y_1)|/|f''(y_2)|$ ist für alle $y_1, y_2 \in \mathbb{R}$ mit $|y_1 - x^*| \leq \delta$ und $|y_2 - x^*| \leq \delta$. Sei $|x_0 - x^*| \leq \delta$. Wir setzen $q = 1/\delta^2$. Dann folgt aus (4.3) mit $x = x^*$ und $y = x_i$ die Gleichung

$$0 = f(x^*) = f(x_i) + f'(x_i)(x^* - x_i) + \frac{1}{2} f''(\xi)(x^* - x_i)^2.$$

Dividieren durch $f'(x_i)$ liefert

$$x_{i+1} - x^* = x_i - \frac{f(x_i)}{f'(x_i)} - x^* = \underbrace{\frac{1}{2}\frac{f''(\xi)}{f'(x_i)}}_{\leq q}(x_i - x^*)^2,$$

also die gewünschte Konvergenz. □

Das folgende Beispiel zeigt den Verlauf des Newton–Verfahrens für ein Beispiel.

Beispiel 4.7 Betrachte die Funktion $f(x) = x^2 - 2$ mit $f'(x) = 2x$ und Nullstelle $x^* = \sqrt{2} \approx 1.4142135623731$. Die Iterationsvorschrift des Newton–Verfahrens ergibt hier

$$x_{i+1} = x_i - f(x_i)/f'(x_i) = x_i - \frac{x_i^2 - 2}{2x_i} = \frac{1}{2}x_i + \frac{1}{x_i}$$

Wir wählen den Startwert $x_0 = 2$. Damit erhalten wir (korrekte Nachkommastellen sind unterstrichen)

$$\begin{aligned}
x_1 &= \frac{1}{2}2 + \frac{1}{2} &&= \frac{3}{2} &&= 1.5 \\
x_2 &= \frac{1}{2}\frac{3}{2} + \frac{1}{\frac{3}{2}} &&= \frac{17}{12} &&= 1.\underline{41}\overline{6} \\
x_3 &= \frac{1}{2}\frac{17}{12} + \frac{1}{\frac{17}{12}} &&= \frac{577}{408} &&\approx 1.\underline{4142}156862745 \\
x_4 &= \frac{1}{2}\frac{577}{408} + \frac{1}{\frac{577}{408}} &&= \frac{665857}{470832} &&\approx 1.\underline{4142135623}746
\end{aligned}$$

□

In Satz 4.6 haben wir die Voraussetzung "... sei x_0 hinreichend nahe bei x^*...", was sich im Beweis durch die Wahl eines $\delta > 0$ mit $|x_0 - x^*| < \delta$ ausdrückt. Dies ist keine Bedingung, die nur aus beweistechnischen Gründen eingeführt wurde: Tatsächlich kann es Situationen geben, in denen das Newton–Verfahren nicht oder nur sehr langsam konvergiert. Das folgende Beispiel soll dies verdeutlichen.

Beispiel 4.8 Betrachte die Funktion $f(x) = 5x/4 - x^3/4$ mit Ableitung $f'(x) = 5/4 - 3x^2/4$. Offenbar hat diese Funktion eine Nullstelle in $x^* = 0$ mit $f'(0) = 5/4 \neq 0$ und ist beliebig oft stetig differenzierbar. Die Iterationsvorschrift ergibt sich hier als

$$x_{i+1} = x_i - f(x_i)/f'(x_i) = x_i - \frac{5x_i/4 - x_i^3/4}{5/4 - 3x_i^2/4} = \frac{2x_i^3}{-5 + 3x_i^2}.$$

Mit Startwert $x_0 = 1$ erhalten wir daraus

$$\begin{aligned}
x_1 &= \frac{2 \cdot 1}{-5 + 3 \cdot 1} = \frac{2}{-2} = -1 \\
x_2 &= \frac{2 \cdot (-1)}{-5 + 3 \cdot 1} = \frac{-2}{-2} = 1 \\
x_3 &= \frac{2 \cdot 1}{-5 + 3 \cdot 1} = \frac{2}{-2} = -1 \\
x_4 &= \frac{2 \cdot (-1)}{-5 + 3 \cdot 1} = \frac{-2}{-2} = 1 \\
&\vdots
\end{aligned}$$

Das Verfahren springt also für alle Zeiten zwischen 1 und -1 hin und her und konvergiert nicht. □

Beim Newton–Verfahren ist es also wichtig, bereits einen brauchbaren Startwert x_0 für die Iteration zu haben, man sagt, dass dieses Verfahren nur *lokal konvergiert*.

Solch ein geeigneter Startwert x_0 kann z.B. mit dem Bisektionsverfahren berechnet werden, da dieses global konvergiert und somit immer eine brauchbare Lösung liefert.

Ein wesentlicher Nachteil des Newton–Verfahrens ist, dass die Ableitung der Funktion f in der Iterationsvorschrift benötigt wird. Zwar könnte man die Ableitung durch eine numerische Näherung ersetzen (ersetzt man z.B. $f'(x)$ durch $(f(x+f(x))-f(x))/f(x)$ so erhält man das *Steffensen–Verfahren*), die numerische Approximation von Ableitungen ist aber im Allgemeinen sehr anfällig gegenüber Rundungsfehlern, weswegen dies in der Praxis nur selten verwendet wird.

Will man also die Ableitung vermeiden, verwendet man lieber ein alternatives Verfahren, das im folgenden Abschnitt beschrieben ist.

4.4 Das Sekanten–Verfahren

Das Sekanten–Verfahren unterscheidet sich in der Konzeption leicht vom Newton–Verfahren, da hier die neue Näherung x_{i+1} nicht nur aus x_i sondern aus x_{i-1} und x_i berechnet wird.

Wir geben wiederum zunächst die formale und dann eine anschauliche Beschreibung.

Algorithmus 4.9 (Sekanten–Verfahren) Gegeben sei eine Funktion $f : \mathbb{R} \to \mathbb{R}$ sowie zwei Anfangswerte $x_0 \in \mathbb{R}$ und $x_1 \in \mathbb{R}$ und eine gewünschte Genauigkeit $\varepsilon > 0$. Setze $i = 1$.

(1) Berechne $x_{i+1} = x_i - \dfrac{(x_i - x_{i-1})f(x_i)}{f(x_i) - f(x_{i-1})}$

(2) Falls $|x_{i+1} - x_i| < \varepsilon$, beende den Algorithmus,
ansonsten setzte $i = i + 1$ und gehe zu (1)

□

Die Idee hinter diesem Verfahren ist wie folgt: man betrachtet zunächst die Sekante $g(x)$ durch $f(x_{i-1})$ und $f(x_i)$, d.h. die Gerade

$$g(x) = f(x_i) + \frac{(x_i - x)}{x_i - x_{i-1}}\Big(f(x_{i-1}) - f(x_i)\Big)$$

und wählt x_{i+1} als Nullstelle dieser Geraden, also

$$\begin{aligned} 0 &= f(x_i) + \frac{(x_i - x_{i+1})}{x_i - x_{i-1}}\Big(f(x_{i-1}) - f(x_i)\Big) \\ \Leftrightarrow \quad x_i - x_{i+1} &= -f(x_i)\frac{x_i - x_{i-1}}{f(x_{i-1}) - f(x_i)} \\ \Leftrightarrow \quad x_{i+1} &= x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \end{aligned}$$

Die Idee ist in Abbildung 4.5 grafisch veranschaulicht.

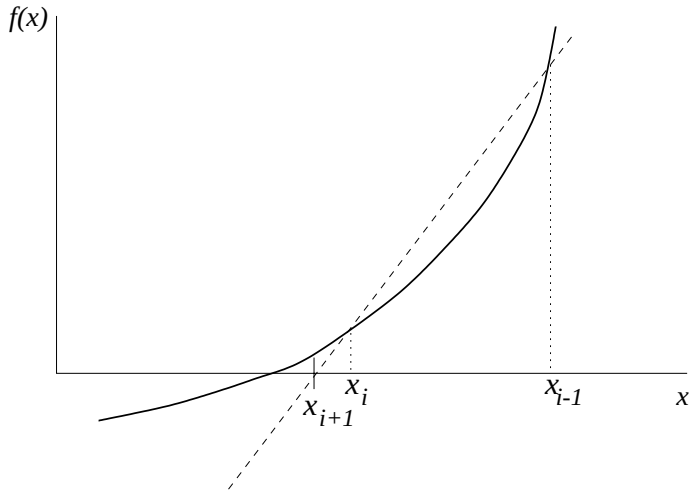


Abbildung 4.5: Sekanten–Verfahren

Der folgende Satz zeigt die Konvergenzordnungen dieses Verfahrens.

Satz 4.10 Sei $f : \mathbb{R} \to \mathbb{R}$ zwei mal stetig differenzierbar und sei x_0 hinreichend nahe bei x^*. Weiterhin gelte $f'(x) \neq 0$. Dann konvergiert das Sekanten–Verfahren superlinear.

Beweisskizze: Wenn x_i und x_{i-1} hinreichend nahe bei x^* liegen, folgt aus dem Taylor–Satz (vgl. den Beweis von Satz 4.6) die Abschätzung

$$f(x_i) - f(x_{i-1}) \approx f'(x^*)(x_i - x_{i-1}) \tag{4.4}$$

sowie für $j = i - 1$ und $j = i$ die Abschätzungen

$$f(x_j) \approx f'(x^*)(x_j - x^*) + \frac{1}{2} f''(x^*)(x_j - x^*)^2. \tag{4.5}$$

Aus der Iterationsvorschrift ergibt sich die Gleichung

$$x_{i+1} - x^* = \frac{(x_{i-1} - x^*)f(x_i) - (x_i - x^*)f(x_{i-1})}{f(x_i) - f(x_{i-1})}.$$

Aus (4.4) erhalten wir für den Nenner die Abschätung

$$f(x_i) - f(x_{i-1}) \approx f'(x^*)(x_i - x_{i-1})$$

und mittels (4.5) erhalten wir für den Zähler

$$\begin{aligned}
&(x_{i-1} - x^*)f(x_i) - (x_i - x^*)f(x_{i-1}) \\
&\approx \; (x_{i-1} - x^*)\Big(f'(x^*)(x_i - x^*) + \frac{1}{2} f''(x^*)(x_i - x^*)^2\Big) \\
&\quad - \; (x_i - x^*)\Big(f'(x^*)(x_{i-1} - x^*) + \frac{1}{2} f''(x^*)(x_{i-1} - x^*)^2\Big) \\
&= \; \frac{1}{2} f''(x^*)(x_{i-1} - x^*)(x_i - x^*)\Big((x_i - x^*) - (x_{i-1} - x^*)\Big) \\
&= \; \frac{1}{2} f''(x^*)(x_{i-1} - x^*)(x_i - x^*)(x_i - x_{i-1}).
\end{aligned}$$

Also folgt

$$|x_{i+1} - x^*| \approx \underbrace{\frac{1}{2}\left|\frac{f''(x^*)}{f'(x^*)}\right| |x_{i-1} - x^*|}_{\approx c_i} |x_i - x^*|$$

und damit die behauptete superlineare Konvergenz. □

Wir wiederholen Beispiel 4.7 für dieses Verfahren.

Beispiel 4.11 Betrachte wiederum die Funktion $f(x) = x^2 - 2$ mit Nullstelle $x^* = \sqrt{2} \approx 1.4142135623731$. Die Iterationsvorschrift des Sekanten–Verfahrens ergibt hier

$$x_{i+1} = x_i - \frac{(x_i - x_{i-1})(x_i^2 - 2)}{x_i^2 - x_{i-1}^2} = x_i - \frac{(x_i - x_{i-1})(x_i^2 - 2)}{(x_i - x_{i-1})(x_i + x_{i-1})} = x_i - \frac{x_i^2 - 2}{x_i + x_{i-1}}.$$

Mit $x_0 = 2$ und $x_1 = 1$ ergibt sich (korrekte Nachkommastellen sind unterstrichen)

$$\begin{aligned}
x_2 &= 1 - \frac{1^2 - 2}{1 + 2} &&= \frac{4}{3} &&= 1.\overline{3} \\
x_3 &= \frac{4}{3} - \frac{(\frac{4}{3})^2 - 2}{\frac{4}{3} + 1} &&= \frac{10}{7} &&= 1.\underline{4}285714\overline{} \\
x_4 &= \frac{10}{7} - \frac{(\frac{10}{7})^2 - 2}{\frac{10}{7} + \frac{4}{3}} &&= \frac{41}{29} &&\approx 1.\underline{41}37931034483 \\
x_5 &= \frac{41}{29} - \frac{(\frac{41}{29})^2 - 2}{\frac{41}{29} + \frac{10}{7}} &&= \frac{816}{577} &&\approx 1.\underline{4142}14384749 \\
x_6 &= \frac{816}{577} - \frac{(\frac{816}{577})^2 - 2}{\frac{816}{577} + \frac{41}{29}} &&= \frac{66922}{47321} &&\approx 1.\underline{41421356}2\,6889
\end{aligned}$$

Das Sekanten–Verfahren konvergiert also deutlich schneller als linear aber langsamer als das Newton–Verfahren. □

Auch das Sekanten–Verfahren ist nur lokal konvergent, d.h. die Anfangswerte müssen genügend nahe an x^* liegen, um die Konvergenz des Verfahrens zu garantieren. Auch hier bietet sich das Bisektions–Verfahren an, um gute Anfangswerte in der Nähe von x^* zu finden. Eine Strategie dieser Art wird z.B. in der Nullstellenroutine `fzero` in MATLAB benutzt.

4.5 Das Newton–Verfahren in höheren Dimensionen

Wir betrachten nun noch kurz das Problem der Lösung nichtlinearer Gleichungssysteme, also Probleme der Form: finde $x^* \in \mathbb{R}^n$ mit $f(x^*) = 0$ für eine gegebene Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$.

Beispiel 4.12 Gesucht ist eine Lösung des nichtlinearen Gleichungssystems

$$\begin{aligned} x_1^2 + x_2^2 &= 1 \\ x_1 &= 0 \end{aligned}$$

Dies ist äquivalent zum Suchen einer Nullstelle $x^* \in \mathbb{R}^2$ der Funktion $f : \mathbb{R}^2 \to \mathbb{R}^2$ gegeben durch

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} = \begin{pmatrix} x_1^2 + x_2^2 - 1 \\ x_1 \end{pmatrix}.$$

Für die gesuchte Lösung x^* muss also gleichzeitig $f_1(x^*) = 0$ und $f_2(x^*) = 0$ gelten. Für die hier gegebene Funktion f ist die Lösung leicht zu sehen: Die Funktion f_1 ist genau dann gleich Null, wenn $x_1^2 + x_2^2 = 1$, also $\|x\| = 1$ ist. Die Funktion f_2 ist gleich Null, wenn $x_1 = 0$ ist. Die Menge der möglichen Lösungen bestehe also aus allen Vektoren $(x_1, x_2)^T$ der Länge $\|x\| = 1$, für die $x_1 = 0$ ist, also $x^* = (0, 1)^T$ oder $x^* = (0, -1)^T$. □

Von den bisher besprochenen Verfahren lässt sich nur das Newton–Verfahren direkt auf höherdimensionale Probleme verallgemeinern, weswegen wir uns hier auf dieses Verfahren beschränken.

Wenn wir die Iterationsvorschrift des Newton–Verfahrens für $f : \mathbb{R} \to \mathbb{R}$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

betrachten, stellt sich die Frage, wie eine geeignete Verallgemeinerung aussehen kann.

Sei dazu $f : \mathbb{R}^n \to \mathbb{R}^n$ nun eine vektorwertige Funktion. Die Ableitung an einer Stelle $x \in \mathbb{R}^n$, die wir — um den Vektorfall zu betonen — nun mit $Df(x)$ bezeichnen, ist jetzt keine reelle Zahl mehr, sondern eine Matrix. Genauer ist für

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad \text{und} \quad f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix}$$

mit $f_i : \mathbb{R}^n \to \mathbb{R}$ die Ableitung an einer Stelle x gegeben durch eine Matrix der Form

$$Df(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \dots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \dots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \frac{\partial f_n}{\partial x_2}(x) & \dots & \frac{\partial f_n}{\partial x_n}(x) \end{pmatrix},$$

wobei $\frac{\partial f_i}{x_j}(x)$ die partielle Ableitung von f_i nach x_j bezeichnet. Wenn man also $Df(x)$ als Funktion in $x \in \mathbb{R}^n$ auffasst, ist dies eine Abbildung $DF : \mathbb{R}^n \to \mathbb{R}^{n \times n}$.

Beispiel 4.12 (Fortsetzung) Betrachte die Funktion $f : \mathbb{R}^2 \to \mathbb{R}^2$ gegeben durch

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} = \begin{pmatrix} x_1^2 + x_2^2 - 1 \\ x_1 \end{pmatrix}.$$

Die partiellen Ableitungen von f_1 und f_2 lauten

$$\frac{\partial f_1}{x_1}(x) = 2x_1, \quad \frac{\partial f_1}{x_2}(x) = 2x_2, \quad \frac{\partial f_2}{x_1}(x) = 1, \quad \frac{\partial f_2}{x_2}(x) = 0$$

also ist

$$Df(x) = \begin{pmatrix} 2x_1 & 2x_2 \\ 1 & 0 \end{pmatrix}$$

Damit ist z.B. für $x = (0,1)^T$ die Ableitung gegeben durch

$$Df(x) = \begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix}.$$

□

Natürlich können wir diese Ableitung $Df(x)$ nicht einfach in die Iterationsvorschrift für x_{i+1} einsetzen, da man ja durch eine Matrix nicht teilen kann. Man kann also nicht einfach $f(x_i)/f'(x_i)$ durch $f(x_i)/Df(x_i)$ ersetzen, sondern muss, um den selben Effekt zu erzielen, die entsprechende Operation für Matrizen verwenden. Statt durch $Df(x_i)$ zu teilen, multiplizieren wir nun mit $Df(x_i)^{-1}$, berechnen also $Df(x_i)^{-1}f(x_i)$. Dies führt zum folgenden Algorithmus

Algorithmus 4.13 (Newton–Verfahren im $\mathbb{R}^n$, Version 1)
Gegeben sei eine Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$, ihre Ableitung $Df : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ sowie ein Anfangswert $x_0 \in \mathbb{R}^n$ und eine gewünschte Genauigkeit $\varepsilon > 0$. Setze $i = 0$.

(1) Berechne $x_{i+1} = x_i - Df(x_i)^{-1}f(x_i)$

(2) Falls $\|x_{i+1} - x_i\| < \varepsilon$, beende den Algorithmus,
ansonsten setzte $i = i + 1$ und gehe zu (1)

□

Wir illustrieren den Ablauf dieses Algorithmus an dem obigen Beispiel.

Beispiel 4.12 (Fortsetzung) Die Inverse der Ableitung

$$Df(x) = \begin{pmatrix} 2x_1 & 2x_2 \\ 1 & 0 \end{pmatrix}$$

ist gegeben durch

$$Df(x)^{-1} = \begin{pmatrix} 0 & 1 \\ \frac{1}{2x_2} & -\frac{x_1}{x_2} \end{pmatrix}.$$

Die Iterationsvorschrift ergibt sich mit $x_i = (x_{i1}, x_{i2})^T$ also zu

$$x_{i+1} = x_i - \begin{pmatrix} 0 & 1 \\ \frac{1}{2x_{i2}} & -\frac{x_{i1}}{x_{i2}} \end{pmatrix} \begin{pmatrix} x_{i1}^2 + x_{i2}^2 - 1 \\ x_{i1} \end{pmatrix}.$$

Mit $x_0 = (1, 1)$ ergibt sich so (korrekte Nachkommastellen sind wieder unterstrichen)

$$
\begin{aligned}
x_1 &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{1}{2} & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} &&= \begin{pmatrix} 0 \\ \frac{3}{2} \end{pmatrix} &&= \begin{pmatrix} 0 \\ 1.5 \end{pmatrix} \\
x_2 &= \begin{pmatrix} 0 \\ \frac{3}{2} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{1}{3} & 0 \end{pmatrix} \begin{pmatrix} \frac{5}{4} \\ 0 \end{pmatrix} &&= \begin{pmatrix} 0 \\ \frac{13}{12} \end{pmatrix} &&= \begin{pmatrix} 0 \\ 1.\underline{0}8\overline{3} \end{pmatrix} \\
x_3 &= \begin{pmatrix} 0 \\ \frac{13}{12} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{6}{13} & 0 \end{pmatrix} \begin{pmatrix} \frac{25}{144} \\ 0 \end{pmatrix} &&= \begin{pmatrix} 0 \\ \frac{313}{312} \end{pmatrix} &&\approx \begin{pmatrix} 0 \\ 1.\underline{00}32051282 \end{pmatrix} \\
x_4 &= \begin{pmatrix} 0 \\ \frac{313}{312} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{156}{313} & 0 \end{pmatrix} \begin{pmatrix} \frac{625}{97344} \\ 0 \end{pmatrix} &&= \begin{pmatrix} 0 \\ \frac{195313}{195312} \end{pmatrix} &&\approx \begin{pmatrix} 0 \\ 1.\underline{00000}51200 \end{pmatrix}
\end{aligned}
$$

□

Auch für dieses Verfahren kann man mit ähnlichen aber durch die auftretenden Vektoren etwas komplizierteren Methoden als in $\mathbb{R}$ nachweisen, dass das Verfahren lokal quadratisch gegen den richtigen Wert konvergiert.

In der Praxis wird man $Df(x)^{-1}$ natürlich nicht "per Hand" berechnen, sondern eine numerische Routine verwenden. Tatsächlich ist es numerisch nicht besonders effizient, die Inverse der Matrix $Df(x_i)$ wirklich zu berechnen, statt dessen löst man das lineare Gleichungssystem $Df(x_i)v = f(x_i)$, das einen Vektor v mit $v = Df(x_i)^{-1}f(x_i)$ liefert. Dies führt zu der folgenden effizienteren Version des Newton–Verfahrens.

Algorithmus 4.14 (Newton–Verfahren im $\mathbb{R}^n$, Version 2)
Gegeben sei eine Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$, ihre Ableitung $Df : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ sowie ein Anfangswert $x_0 \in \mathbb{R}^n$ und eine gewünschte Genauigkeit $\varepsilon > 0$. Setze $i = 0$.

(1) Löse das lineare Gleichungssystem $Df(x_i)v = f(x_i)$
und berechne $x_{i+1} = x_i - v$

(2) Falls $\|x_{i+1} - x_i\| < \varepsilon$, beende den Algorithmus,
ansonsten setzte $i = i + 1$ und gehe zu (1)

□

Kapitel 5

Gewöhnliche Differentialgleichungen

Differentialgleichungen sind aus der Modellierung von Phänomenen in fast allen Wissenschaften kaum mehr wegzudenken. Im naturwissenschaftlich–technischen Bereich können sie z.B. zur Beschreibung physikalischer, chemischer, mechanischer, elektronischer und biologischer Systeme verwendet werden.

Wir werden uns hier auf numerische Verfahren für *gewöhnliche Differentialgleichungen* beschränken und das Hauptaugenmerk auf die sogenannten *Anfangswertprobleme* legen.

Eine gewöhnliche Differentialgleichung ist eine Gleichung, bei der die Ableitung einer Funktion nach *einer eindimensionalen* unabhängigen Variablen mit der Funktion selbst in Beziehung gesetzt wird. Wir machen hier die Konvention, dass die unabhängige Variable immer mit "t" bezeichnet wird, was auf die physikalische Interpretation von t als Zeit anspielt.

Gesucht ist also eine Funktion $x(t)$ mit Werten $x(t) = (x_1(t), \ldots, x_n(t))^T \in \mathbb{R}^n$, die die Gleichung

$$\frac{d}{dt}x(t) = f(t, x(t))$$

für eine vorgegebene Funktion $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ erfüllt. Statt $\frac{d}{dt}x(t)$ schreiben wir üblicherweise kurz $\dot{x}(t)$. Um unsere Betrachtungen hier etwas zu vereinfachen, werden wir uns auf *zeitunabhängige* gewöhnliche Differentialgleichungen beschränken, d.h. auf Gleichungen der Form

$$\dot{x}(t) = f(x(t)). \tag{5.1}$$

Bevor man sich an die numerische Behandlung dieses Problems macht, sollte man sich überlegen, unter welchen Voraussetzungen die Gleichung (5.1) überhaupt eine Lösung $x(t)$ besitzt.

Wir zitieren den folgenden Satz und erinnern dazu daran, dass die Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$ *global Lipschitz stetig* heißt, wenn eine Konstante $L > 0$ existiert, so dass die Ungleichung

$$\|f(x_1) - f(x_2)\| \leq L\|x_1 - x_2\|$$

für alle $x_1, x_2 \in \mathbb{R}^n$ gilt.

Satz 5.1 Sei $f : \mathbb{R}^n \to \mathbb{R}^n$ global Lipschitz stetig. Dann existiert für jede Zeit $t_0 \in \mathbb{R}$ und jeden Wert $x_0 \in \mathbb{R}^n$ genau eine für alle $t \in \mathbb{R}$ definierte Lösung $x(t)$ der Differentialgleichung (5.1), die die *Anfangsbedingung* $x(t_0) = x_0$ erfüllt. Diese Lösung bezeichnen wir mit $x(t; t_0, x_0)$. Die Zeit $t_0 \in \mathbb{R}$ heißt hierbei *Anfangszeit*, der Wert $x_0 \in \mathbb{R}^n$ heißt *Anfangswert*.

Dieser Satz legt die Minimalbedingung fest, die wir im Folgenden an f stellen wollen, nämlich globale Lipschitz–Stetigkeit. Falls die globale Lipschitz Stetigkeit nicht erfüllt ist (was in vielen Anwendungen der Fall ist), kann man sich damit behelfen, dass man diese Eigenschaft nur für die Menge fordert, in der die Lösung $x(t; t_0, x_0)$ ihre Werte annimmt; dies führt zur *lokalen Lipschitz–Stetigkeit*, die für die üblichen Anwendungen in der Regel erfüllt ist.

5.1 Beispiele

Wir stellen hier einige Beispiele dar, die insbesondere auch die verschiedenen grafischen Darstellungsmöglichkeiten der Lösungen illustrieren sollen.

Beispiel 5.2 (Radioaktiver Zerfall)
Wenn $x(t)$ die Masse einer radioaktiven Substanz zur Zeit t darstellt, so lässt sich die Abnahme dieser Masse aus dem radioaktiven Zerfallsgesetz über die eindimensionale Differentialgleichung

$$\dot{x}(t) = -\lambda x(t)$$

für ein $\lambda > 0$ beschreiben. (Die Abnahme ist proportional zur vorhandenen Masse.) Diese Differentialgleichung lässt sich explizit lösen; die Lösung ist gegeben durch

$$x(t; t_0, x_0) = e^{-\lambda(t-t_0)} x_0.$$

Abbildung 5.1 zeigt die Lösung in Abhängigkeit von t für $t_0 = 0$, $x_0 = 1$ und $\lambda = 0.5$. □

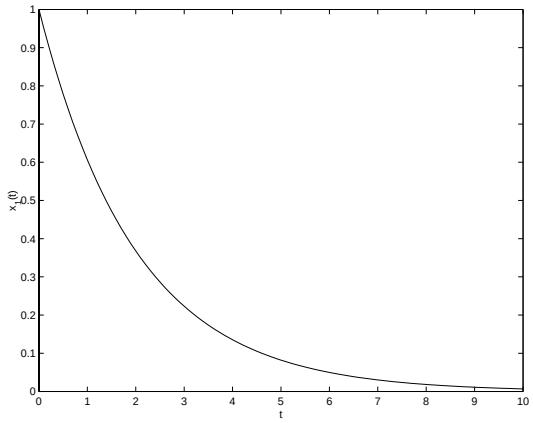


Abbildung 5.1: Lösung von Beispiel 5.2 in Abhängigkeit von t

Beispiel 5.3 (Drei–Körper–Problem)
Die folgende vierdimensionale Differentialgleichung beschreibt das sogenannte restringierte Drei–Körper–Problem.

$$\begin{aligned}
\dot{x}_1(t) &= x_2(t) \\
\dot{x}_2(t) &= x_1(t) + 2x_4(t) - \mu_2 \frac{x_1(t)+\mu_1}{D_1} - \mu_1 \frac{x_1(t)-\mu_2}{D_2} \\
\dot{x}_3(t) &= x_4(t) \\
\dot{x}_4(t) &= x_3(t) - 2x_2(t) - \mu_2 \frac{x_3(t)}{D_1} - \mu_1 \frac{x_3(t)}{D_2}
\end{aligned}$$

$$\begin{aligned}
D_1 &= \left((x_1(t)+\mu_1)^2 + x_3(t)^2\right)^{3/2}, \quad D_2 = \left((x_1(t)-\mu_2)^2 + x_3(t)^2\right)^{3/2} \\
\mu_1 &= 0.012277471, \quad \mu_2 = 1-\mu_1
\end{aligned}$$

Hierbei sind x_1 und x_3 die Koordinaten und x_2 und x_4 die zugehörigen Geschwindigkeiten eines Satelliten der Erde–Mond–Ebene, mit der Erde im Punkt $(0,0)$ und dem Mond im Punkt $(1,0)$. Die Werte μ_1 und μ_2 sind die skalierten Massen von Mond und Erde. Abbildung 5.2 zeigt die x_1 und x_3–Komponente der Lösung für Anfangszeit $t_0 = 0$ und Anfangswert $x_0 = (0.994, 0, 0, -2.00158510637908252240537862224)^T$. Die rechte Grafik zeigt die Lösung in Abhängigkeit von t, links ist die Lösung als Kurve dargestellt, d.h. es wird $x_2(t)$ gegen $x_1(t)$ geplottet, wobei die Zeit t hier nicht mehr sichtbar ist.

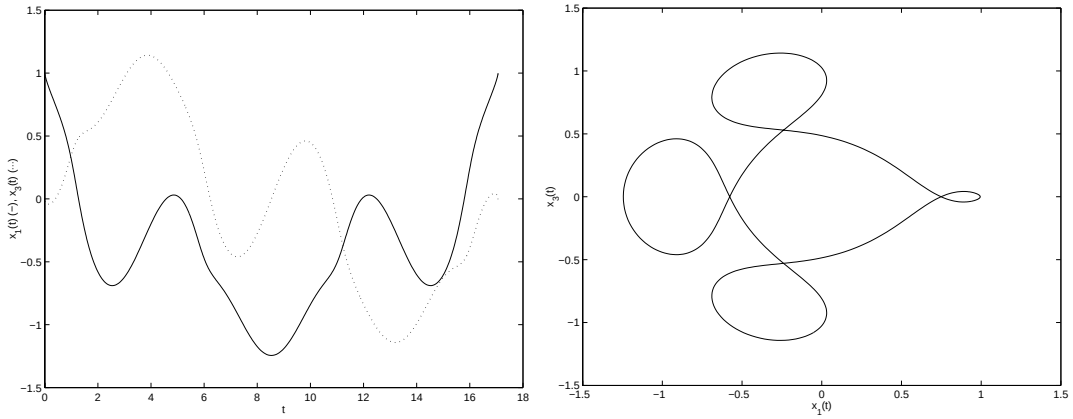


Abbildung 5.2: Lösung von Beispiel 5.3 (x_1 und x_3–Komponente) in Abhängigkeit von t und als Kurve

□

Beispiel 5.4 (Lorenz–Gleichung) Die folgende dreidimensionale Differentialgleichung wurde von dem Metereologen E. Lorenz in den 1960er Jahren aufgestellt und beschreibt ein (sehr einfaches) Modell für den Wärmeaustausch in Luftströmungen.

$$\begin{aligned}
\dot{x}_1(t) &= 10(x_2(t) - x_1(t)) \\
\dot{x}_2(t) &= -x_1(t)x_3(t) + 28x_1(t) - x_2(t) \\
\dot{x}_3(t) &= x_1(t)x_2(t) - \frac{8}{3}x_3(t)
\end{aligned}$$

Interessanter als die physikalische Interpretation (die durch die starke Vereinfachung dieses Modells sowieso nur von geringer Aussagekraft ist) ist bei diesem Modell das sehr seltsame Verhalten der Lösungen. Alle Lösungen streben auf eine "schmetterlingsförmige" Menge zu (den sogenannten *Lorenz–Attraktor*). Auf dieser Menge lässt sich allerdings kein systematisches Verhalten der Lösungen erkennen; trotz der rein deterministischen Natur der Gleichungen erscheint das Verhalten wie zufällig. Dieses Verhalten wird üblicherweise als chaotisch bezeichnet, der Lorenz–Attraktor ist damit ein Beispiel für einen sogenanten *chaotischen Attraktor*.

Abbildung 5.3 zeigt die Lösung für den Anfangswert $x_0 = (0.1, 0, 0)^T$ zur Anfangszeit $t_0 = 0$. Die Lösung in Abhängigkeit von t ist links für $t \in [0, 40]$, die Lösung als Kurve ist rechts für $t \in [0, 100]$ dargestellt. □

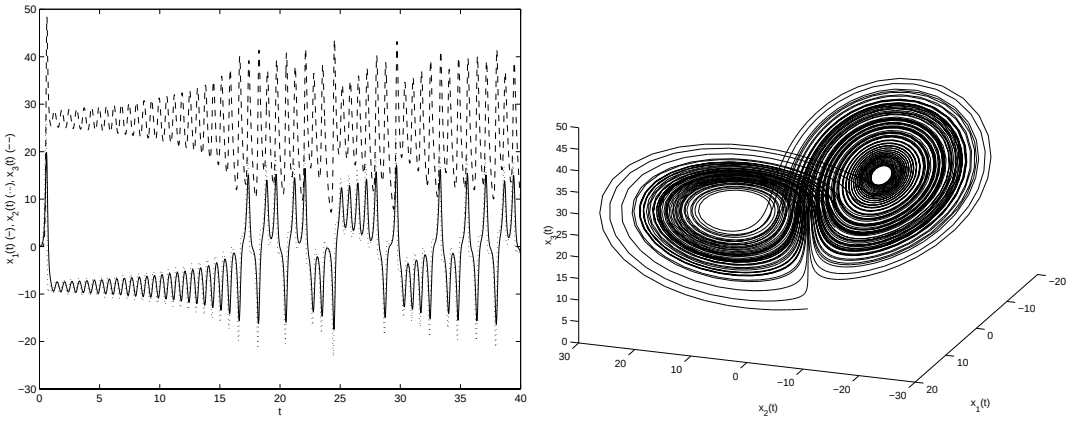


Abbildung 5.3: Lösung von Beispiel 5.4 in Abhängigkeit von t und als Kurve

5.2 Einschrittverfahren

Die einfachste und am weitesten verbreitete Klasse von numerischen Lösungsverfahren für gewöhnliche Differentialgleichungen sind die sogenannten *Einschrittverfahren*.

Wir betrachten diese Verfahren hier zunächst mit *konstanter Schrittweite* h. Ein *explizites Einschrittverfahren* für eine gewöhnliche Differentialgleichung (5.1) ist dann gegeben durch die Iterationsvorschrift

$$x_{i+1} = \Phi(h, x_i), \quad i = 0, 1, 2, 3$$

wobei x_0 der vorgegebene Anfangswert ist und Φ eine Abbildung ist, die von f aus (5.1) abhängt und im Computer auswertbar ist. Der Wert x_i stellt dann eine Approximation für den Wert $x(ih + t_0; t_0, x_0)$ dar. Der Name Einschrittverfahren kommt daher, dass man den Wert x_{i+1} in einem Schritt, d.h. mit einer Auswertung der Abbildung Φ aus dem vorhergehenden Wert x_i berechnet. Abbildung 5.4 illustriert das Prinzip dieser Approximation für die Differentialgleichung aus Beispiel 5.2 mit $h = 1$, wobei die einzelnen Punkte x_i durch gestrichelte Geradenstücke verbunden sind.

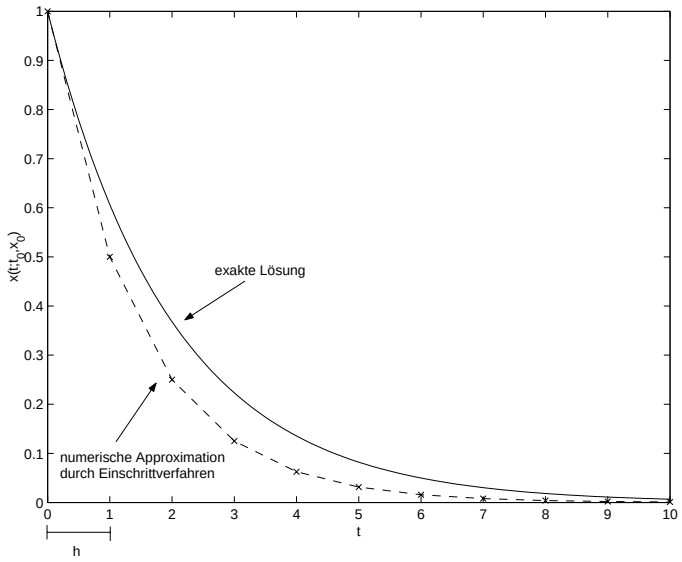


Abbildung 5.4: Illustration eines Einschrittverfahrens

Um ein Einschrittverfahren zu konstruieren, genügt es also, eine geeignete Abbildung $\Phi(h, x)$ zu bestimmen, die angibt, wie man aus der Approximation x_i die Approximation x_{i+1} nach dem nächsten Zeitschritt berechnet.

Beispiele: Das einfachste Verfahren dieser Art ist das *Euler–Verfahren*, bei dem Φ gegeben ist durch

$$\Phi(h, x) = x + hf(x),$$

also

$$x_{i+1} = x_i + hf(x_i).$$

Ein etwas komplizierteres Verfahren ist das *Heun–Verfahren*, das gegeben ist durch

$$\Phi(h, x) = x + \frac{h}{2}\Big(f(x) + f(x + hf(x))\Big).$$

Wir werden später eine systematische Art und Weise zur Darstellung von Einschrittverfahren kennen lernen. Vorher wollen wir uns allerdings mit der Konvergenztheorie dieser Verfahren beschäftigen.

5.3 Konvergenztheorie

Wir wollen hier Bedingungen an die Abbildung Φ angeben, unter denen man beweisen kann, dass die aus einem Einschrittverfahren gewonnene Folge x_i, $i = 0, 1, 2, 3, \ldots$ tatsächlich eine Approximation der Lösung $x(t; t_0, x_0)$ der gewöhnlichen Differentialgleichung (5.1) darstellt. Genauer wollen wir zeigen, dass — und in welchem Sinne — die Werte x_i gegen die Werte $x(hi + t_0; t_0, x_0)$ konvergieren, falls $h \to 0$ geht. Da es üblicherweise nicht sinnvoll ist, die Zeitschrittweite h beliebig groß zu wählen, wählen wir einen beliebigen Wert h_0 und betrachten Zeitschrittweiten $0 < h \leq h_0$.

Hierzu benötigen wir die folgenden Definitionen

Definition 5.5 Ein Einschrittverfahren heißt *konsistent*, falls Konstanten $C, p > 0$ existieren, so dass die Ungleichung

$$\|\Phi(h, x_0) - x(h + t_0; t_0, x_0)\| \leq Ch^{p+1}$$

gilt für alle $t_0 \in \mathbb{R}$, alle $x_0 \in \mathbb{R}^n$, alle mindestens p–mal stetig differenzierbaren $f : \mathbb{R}^n \to \mathbb{R}^n$ und alle $0 < h \leq h_0$. Der Wert p wird dabei die *Konsistenzordnung* des Verfahrens genannt. □

Die Konsistenz vergleicht die exakte und die numerische Lösung nach *einem* Zeitschritt und verlangt, dass diese durch eine Potenz der Form Ch^{p+1} beschränkt ist. Der direkte Nachweis dieser Eigenschaft ist nicht ganz leicht, da darin die im Allgemeinen unbekannten Lösungen $x(h + t_0; t_0, x_0)$ vorkommen.

Für die Konsistenz eines Verfahrens gibt es aber einen einfachen Test, der die Lösungen vermeidet: Falls die Bedingung

$$\lim_{h \to 0,\, h>0} \left\| \frac{\Phi(h, x) - x}{h} - f(x) \right\| = 0 \tag{5.2}$$

gilt, so ist das Verfahren konsistent. Diese Bedingung ist leichter nachzuprüfen als Definition 5.5, da nur bekannte Funktionen vorkommen; allerdings kann man aus (5.2) keine Abschätzung der Konsistenzordnung gewinnen.

Beispiele: Das Euler–Verfahren ist offenbar konsistent, da

$$\left\| \frac{\Phi(h, x) - x}{h} - f(x) \right\| = \|f(x) - f(x)\| = 0$$

ist, also (5.2) erfüllt ist.

Das Heun–Verfahren ist ebenfalls konsistent, denn aus der Lipschitz–Stetigkeit von f folgt

$$\begin{aligned} \left\| \frac{\Phi(h, x) - x}{h} - f(x) \right\| &= \left\| \frac{1}{2}\Big(f(x) + f(x + hf(x))\Big) - f(x) \right\| \\ &= \frac{1}{2}\|f(x + hf(x)) - f(x)\| \\ &\leq \frac{1}{2}L\|hf(x)\| \;\to\; 0 \end{aligned}$$

für $h \to 0$.

Mit mehr Aufwand (den wir hier aus Zeitgründen nicht betreiben können) kann man mit Hilfe des Taylor–Satzes nachrechnen, dass das Euler–Verfahren Konsistenzordnung $p = 1$ hat, während das Heun–Verfahren die Konsistenzordnung $p = 2$ hat.

Konsistenz ist eine *notwendige* Eigenschaft für Konvergenz, denn damit die x_i für $h \to 0$ gegen $x(hi + t_0; t_0, x_0)$ konvergieren, ist es natürlich notwendig, dass insbesondere $x_1 = x_0 + \Phi(h, x_0)$ gegen $x(h + t_0; t_0, x_0)$ konvergiert. Sie ist aber nicht hinreichend, denn sie gibt nur Auskunft über einen Schritt des Verfahrens, wenn die exakte und die approximative Lösung

in t_0 übereinstimmen. Bereits nach dem ersten Schritt ist dies aber nicht mehr der Fall, denn x_2 wird auf Basis des fehlerhaften Wertes x_1 berechnet, x_3 aus dem fehlerhaften Wert x_2 usw. Um zu vermeiden, dass diese Fehler sich im Laufe der Berechnung aufschaukeln, muss das Einschrittverfahren die folgende *Stabilitätsbedingung* erfüllen.

Definition 5.6 Ein Einschrittverfahren heißt *stabil*, falls eine Konstante $M > 0$ existiert, so dass die Ungleichung

$$\|\Phi(h, x_1) - \Phi(h, x_2)\| \leq (1 + hM)\|x_1 - x_2\|$$

für alle $x_1, x_2 \in \mathbb{R}^n$ und alle $0 < h \leq h_0$ gilt. □

Beispiele: Für das Euler–Verfahren gilt

$$\begin{aligned} \|\Phi(h, x_1) - \Phi(h, x_2)\| &= \|x_1 + hf(x_1) - x_2 - hf(x_2)\| \\ &\leq \|x_1 - x_2\| + h \underbrace{\|f(x_1) - f(x_2)\|}_{\leq L\|x_1 - x_2\|} \leq (1 + hL)\|x_1 - x_2\|, \end{aligned}$$

also Stabilität mit $M = L$, wobei L die Lipschitz–Konstante von f ist.

Für das Heun–Verfahren gilt

$$\begin{aligned} \|\Phi(h, x_1) - \Phi(h, x_2)\| &= \Big\| x_1 + \frac{h}{2}\Big(f(x_1) + f(x_1 + hf(x_1))\Big) - \\ &\qquad x_2 - \frac{h}{2}\Big(f(x_2) + f(x_2 + hf(x_2))\Big) \Big\| \\ &\leq \|x_1 - x_2\| + \frac{h}{2}\|f(x_1) - f(x_2)\| \\ &\qquad + \frac{h}{2}\|f(x_1 + hf(x_1)) - f(x_2 + hf(x_2))\| \\ &\leq \|x_1 - x_2\| + \frac{hL}{2}\|x_1 - x_2\| + \frac{hL}{2} \underbrace{\|x_1 + hf(x_1) - x_2 + hf(x_2)\|}_{\leq \|x_1 - x_2\| + Lh\|x_1 - x_2\|} \\ &\leq \|x_1 - x_2\| + hL\|x_1 - x_2\| + \frac{h^2 L^2}{2}\|x_1 - x_2\| \\ &\leq (1 + h(L + h_0 L^2/2))\|x_1 - x_2\| \end{aligned}$$

also Stabilität mit $M = L + h_0 L^2/2$.

Der folgende Satz besagt, dass aus Konsistenz und Stabilität die Konvergenz des Verfahrens folgt.

Satz 5.7 Betrachte ein Einschrittverfahren mit Abbildung Φ, das konsistent und stabil ist. Dann gibt es für jedes $T > 0$ eine Konstante $K(T) > 0$, so dass für alle $0 < h \leq h_0$ und alle $i \in \mathbb{N}$ mit $ih \leq T$ die Abschätzung

$$\|x_i - x(ih + t_0; t_0, x_0)\| \leq K(T)h^p$$

gilt, falls f aus (5.1) p–mal stetig differenzierbar ist, wobei $p > 0$ gerade die Konsistenzordnung des Verfahrens ist.

Beweis: Wir zeigen per Induktion über i die Abschätzung

$$\|x_i - x(ih + t_0; t_0, x_0)\| \leq \frac{e^{Mih} - 1}{M} Ch^p, \tag{5.3}$$

mit C aus Definition 5.5, woraus die Behauptung mit

$$K(T) = \frac{e^{MT} - 1}{M} C$$

folgt. Hierzu benötigen wir die Ungleichung

$$e^{hM} = 1 + hM + \frac{1}{2}h^2M^2 + \frac{1}{6}h^3M^3 + \ldots \geq 1 + hM.$$

Induktionsanfang: $i = 0$. Hier gilt

$$\|x_0 - x(0h + t_0; t_0, x_0)\| = \|x_0 - x_0\| = \frac{e^{M0h} - 1}{M} h^p,$$

also (5.3).

Induktionsschritt: $i \to i + 1$. Wir nehmen an, dass (5.3) für ein $i \geq 0$ gilt und wollen nun (5.3) für $i+1$ zeigen. Hierzu benötigen wir die folgende Eigenschaft der Lösungen der Differentialgleichung (5.1): Sei $\tilde{x}(t)$ eine Lösung von (5.1) zu beliebigem Anfangswert und beliebiger Anfangszeit. Dann stimmt $\tilde{x}(t)$ für jedes $\tilde{t}_0 \in \mathbb{R}$ mit der Lösung $x(t; \tilde{t}_0, \tilde{x}(\tilde{t}_0))$ überein. Dies folgt aus dem Eindeutigkeitssatz, da sowohl $\tilde{x}(t)$ als auch $x(t; \tilde{t}_0, \tilde{x}(\tilde{t}_0))$ die Anfangsbedingung $x(\tilde{t}_0) = \tilde{x}_0$ mit $\tilde{x}_0 = \tilde{x}(\tilde{t}_0)$ erfüllen, und es genau eine Funktion gibt, die dieses tut. Wenden wir diese Eigenschaft nun mit $\tilde{t}_0 = ih$ auf $y(t) = x(t; t_0, x_0)$ an, so erhalten wir mit der Abkürzung $\tilde{x}_0 = x(ih + t_0; t_0, x_0)$ die Gleichung

$$x((i+1)h + t_0; t_0, x_0) = x(h + \tilde{t}_0; \tilde{t}_0, \tilde{x}_0). \tag{5.4}$$

Also folgt

$$\begin{aligned}
&\|x_{i+1} - x((i+1)h + t_0; t_0, x_0)\| \\
&\quad = \|\Phi(h, x_i) - x(h + \tilde{t}_0; \tilde{t}_0, \tilde{x}_0)\| \\
&\quad = \|\Phi(h, x_i) - \Phi(h, \tilde{x}_0)) + \Phi(h, \tilde{x}_0)) - x(h + \tilde{t}_0; \tilde{t}_0, \tilde{x}_0)\| \\
&\quad \leq \|\Phi(h, x_i) - \Phi(h, \tilde{x}_0))\| + \|\Phi(h, \tilde{x}_0)) - x(h + \tilde{t}_0; \tilde{t}_0, \tilde{x}_0)\|
\end{aligned}$$

Der erste Summand ist nun nach Induktionsannahme ((5.3) gilt für i) und der Stabilitätsbedingung kleiner oder gleich

$$\begin{aligned}
(1 + hM)\frac{e^{ihM} - 1}{M} Ch^p &\leq \frac{(1 + hM)e^{ihM} - (1 + hM)}{M} Ch^p \\
&\leq \frac{e^{hM}e^{ihM} - 1}{M} Ch^p - \frac{hM}{M} Ch^p = \frac{e^{(i+1)hM} - 1}{M} Ch^p - Ch^{p+1}.
\end{aligned}$$

Der zweite Summand ist wegen der Konsistenzbedingung kleiner oder gleich Ch^{p+1}, also ergibt sich für die Summe die Abschätzung

$$\begin{aligned}
\|\Phi(h, \tilde{x}_0)) - x(h + \tilde{t}_0; \tilde{t}_0, \tilde{x}_0)\| &\leq \frac{e^{(i+1)hM} - 1}{M} Ch^p - Ch^{p+1} + Ch^{p+1} \\
&= \frac{e^{(i+1)hM} - 1}{M} Ch^p
\end{aligned}$$

und damit die gewünschte Abschätzung (5.3) für $i+1$. □

Beachte, dass die Konstante $K(T)$ immer größer wird, je größer T wird. Üblicherweise muss daher der Zeitschritt h immer kleiner gewählt werden, je größer das Intervall wird, auf dem man die Lösung berechnen will.

5.4 Runge–Kutta–Verfahren

Die direkte Art, in der wir das Heun–Verfahren aufgeschrieben haben, wird für kompliziertere Verfahren, in denen viele Auswertungen von f in einem Schritt durchgeführt werden, sehr unübersichtlich. Wir wollen daher ein Verfahren beschreiben, mit dem man auch sehr komplizierte Einschrittverfahren noch übersichtlich aufschreiben kann. Die Klasse der Verfahren, die man so darstellen kann, wird — nach den Entwicklern dieser Darstellung — Runge–Kutta Verfahren genannt. Um die Verfahren in ihrer üblichen Allgemeinheit zu behandeln, erlauben wir in diesem Abschnitt, dass f von t abhängt, womit dann auch Φ von t abhängen muss.

Wir betrachten zunächst das Heun–Verfahren, das für zeitabhängiges f gegeben ist durch

$$\Phi(h,t,x) = x + \frac{h}{2}\Big(f(t,x) + f(t+h, x+hf(t,x))\Big).$$

Mit der Abkürzung

$$\begin{aligned} k_1 &= f(t,x) \\ k_2 &= f\left(t+h, x+hk_1\right) \end{aligned}$$

lässt sich dies Verfahren auch als

$$\Phi(h,t,x) = x + h\left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right)$$

schreiben.

Indem man weitere k_i–Terme einführt, kann man weitere Auswertungen von f hinzufügen. Das sogenannte *klassische Runge–Kutta–Verfahren* (das die Konsistenzordnung $p=4$ besitzt) ist z.B. gegeben durch

$$\begin{aligned} k_1 &= f(t,x) \\ k_2 &= f\left(t+h\frac{1}{2}, x+h\frac{1}{2}k_1\right) \\ k_3 &= f\left(t+h\frac{1}{2}, x+h\frac{1}{2}k_2\right) \\ k_4 &= f(t+h, x+hk_3) \end{aligned}$$

und

$$\Phi(h,t,x) = x + h\left(\frac{1}{6}k_1 + \frac{2}{6}k_2 + \frac{2}{6}k_3 + \frac{1}{6}k_4\right).$$

Alle diese Verfahren lassen sich durch eine rekursive Konstruktion der folgenden Art darstellen (der Vollständigkeit halber hängt f hier von t ab)

$$\begin{aligned} k_1 &= f(t + hc_1, x) \\ k_2 &= f\left(t + hc_2, x + h\alpha_{2\,1}k_1\right) \\ k_3 &= f\left(t + hc_3, x + h\alpha_{3\,1}k_1 + h\alpha_{3\,2}k_2\right) \\ &\vdots \\ k_m &= f\left(t + hc_m, x + h\alpha_{m\,1}k_1 + \cdots + h\alpha_{m\,m-1}k_{m-1}\right) \end{aligned}$$

und

$$\Phi(h, t, x) = x + h\left(\beta_1 k_1 + \beta_2 k_2 + \cdots + \beta_m k_m\right).$$

Ein Verfahren dieser Art wird *m–stufiges explizites Runge–Kutta–Verfahren* bezeichnet und ist durch die Koeffizienten $\alpha_{i\,j}$ und β_i eindeutig bestimmt. Die Zahl m ist hierbei gerade die Anzahl der Auswertungen von f. Zur Angabe eines solchen Runge–Kutta–Verfahrens henügt es, die Koeffizienten anzugeben, was üblicherweise in der Form eines *Butcher–Tableaus* gemacht wird:

$$\begin{array}{c|ccccc} c_1 & & & & & \\ c_2 & \alpha_{2\,1} & & & & \\ c_3 & \alpha_{3\,1} & \alpha_{3\,2} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_m & \alpha_{m\,1} & \alpha_{m\,2} & \cdots & \alpha_{m\,m-1} & \\ \hline & \beta_1 & \beta_2 & \cdots & \beta_{m-1} & \beta_m \end{array}$$

In dieser Schreibweise sind also z.B. das Euler–, Heun– und das klassische Runge–Kutta–Verfahren gegeben durch die Butcher–Tableaus (von links nach rechts)

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array} \qquad \begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \qquad \begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ \frac{1}{2} & 0 & \frac{1}{2} & \\ 1 & 0 & 0 & 1 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array}$$

Bemerkung 5.8 (i) Mit Hilfe des Taylor–Satzes kann man ein systematisches Verfahren zur Herleitung von Runge–Kutta–Verfahren von im Prinzip beliebig hoher Konsistenzordnung p erhalten, das — vereinfacht gesagt — auf ein Gleichungssystem zur Berechnung der Koeffizienten im Butcher–Tableau führt. Die obigen Verfahren legen nahe, dass man mit einem m–stufigen Verfahren immer die Konsistenzordnung $p = m$ erreichen kann. Leider ist dies nicht so; die folgende Tabelle gibt die zum Erreichen einer vorgegebenen Konsistenzordnung p minimal nötige Stufenzahl m an.

Konsistenzordnung p	1	2	3	4	5	6	7	8	≥ 9
minimale Stufenzahl m	1	2	3	4	6	7	9	11	$\geq p + 3$

(ii) In den hier betrachteten expliziten Runge–Kutta–Verfahren sind alle Koeffizienten $\alpha_{i\,j}$ mit $i \leq j$ gleich Null, weswegen wir sie in den Butcher–Tableaus einfach weggelassen haben. Bei den sogenannten *impliziten* Runge–Kutta–Verfahren dürfen diese Koeffizienten auch ungleich Null sein, dies führt dann auf ein nichtlineares Gleichungssystem für die k_i, das z.B. durch das Newton–Verfahren gelöst werden kann. Dies bewirkt einen großen zusätzlichen numerischen Aufwand; es gibt aber eine Klasse von Differentialgleichungen (die sogenanten *steifen* Differentialgleichungen), bei denen die impliziten Verfahren viel besser funktionieren so dass sich dieser zusätzliche Aufwand durchaus lohnt. □

5.5 Schrittweitensteuerung

Bisher haben sind wir in der Diskussion von Einschrittverfahren immer davon ausgegangen, dass die Schrittweite $h > 0$ vorgegeben ist und während der gesamten Rechnung konstant bleibt. Dies wird im Allgemeinen nicht besonders effizient sein, da die Lösungen gewöhnlicher Differentialgleichungen in manchen Abschnitten rech kompliziert zu lösen sein können (wegen schneller Änderung, starker Krümmung etc.), in anderen Bereichen aber sehr einfache Struktur haben. Wählt man die Schrittweite $h > 0$ konstant, so muss man bei der Wahl vom schlimmsten Fall ausgehen und rechnet dann gezwungenermaßen auch in "einfachen" Regionen mit sehr aufwändiger hoher Genauigkeit.

Das Konzept der *Schrittweitensteuerung* dient dazu, diesen Nachteil aufzuheben, und funktioniert wie folgt in drei Schritten:

(1) Nach Durchführung eines Schrittes mit Schrittweite h wird aus der numerischen Lösung der vorhandene numerische Fehler geschätzt.

(2) Wenn der Fehler über einer vorgegebenen Genauigkeitsschranke *tol* liegt, wird auf Basis des Fehlers eine kleinere Schrittweite h_{neu} für den aktuellen Schritt berechnet und der aktuelle Schritt wird wiederholt.

(3) Auf Basis des geschätzten Fehlers wird eine optimale Schrittweite h für den nächsten Schritt errechnet.

Wahlweise kann der 2. Schritt wegfallen; in diesem Fall wird die Fehlerschätzung nur dazu verwendet, eine gute Schrittweite für den nächsten Schritt zu berechnen, ohne dass dabei die Größe des Fehlers tatsächlich überprüft wird.

Wir besprechen nun die einzelnen Schritte im Detail.

Schritt (1): Fehlerschätzung

Die Idee der Fehlerschätzung im ersten Schritt liegt darin, ausgehend von der numerischen Lösung x_i die Approximation x_{i+1} mit zwei verschiedenen Verfahren Φ und $\widehat{\Phi}$ zu berechnen, wobei $\widehat{\Phi}$ das genauere Verfahren sein soll, also höhere Konsistenzordnung besitzen soll. Da bei der Schrittweitensteuerung nur die Schrittweiten des aktuellen und der zukünftigen Schritte gesteuert werden, sollten Fehler im Wert x_i, die vorher angefallen sind, nicht berücksichtigt werden. In der Fehlerschätzung wollen wir nur den Anteil des Fehlers messen, der der numerischen Lösung mittels Φ im aktuellen Schritt von x_i nach x_{i+1} hinzugefügt

wird. Dies erreichen wir, indem wir so tun als ob x_i die exakte Lösung wäre und dann die Norm $\|\varepsilon\|$ für

$$\varepsilon := x(t_{i+1}; t_i, x_i) - \Phi(h, t_i, x_i)$$

betrachten, wobei t_i und $t_{i+1} = t_i + h$ die zu x_i und x_{i+1} gehörigen Zeiten sind und h die (im vorherigen Schritt bestimmte, bzw. für den ersten Schritt vom Anwender vorgegebene) aktuelle Schrittweite ist. Zur Schätzung dieses (unbekannten) Fehlers verwenden wir den Fehler des genaueren Verfahrens

$$\hat{\varepsilon} := x(t_{i+1}; t_i, x_i) - \widehat{\Phi}(h, t_i, x_i)$$

sowie die Differenz der Verfahren

$$\varepsilon_d := \widehat{\Phi}(h, t_i, x_i) - \Phi(h, t_i, x_i).$$

Beachte, dass ε_d die einzige Größe ist, die wir wirklich messen können, weswegen am Ende unserer Umformungen ein Ausdruck stehen sollte, in dem nur ε_d vorkommt.

Da $\widehat{\Phi}$ nach Annahme höhere Konsistenzordnung besitzt, wird

$$\theta = \frac{\|\hat{\varepsilon}\|}{\|\varepsilon\|}$$

immer kleiner, wenn h gegen Null geht. Hiermit folgt

$$\frac{\|\varepsilon - \varepsilon_d\|}{\|\varepsilon\|} = \theta$$

und damit

$$\|\varepsilon - \varepsilon_d\| = \|\varepsilon\|\theta.$$

Aus den Dreiecks–Ungleichungen

$$\|\varepsilon\| - \|\varepsilon_d\| \leq \|\varepsilon - \varepsilon_d\| \leq \|\varepsilon\| + \|\varepsilon_d\|$$

folgt

$$\frac{1}{1+\theta}\|\varepsilon_d\| \leq \|\varepsilon\| \leq \frac{1}{1-\theta}\|\varepsilon_d\|.$$

Für θ nahe Null gilt also

$$\|\varepsilon_d\| \approx \|\varepsilon\|,$$

weswegen $\|\varepsilon_d\|$ ein brauchbarer Schätzwert für den Fehler $\|\varepsilon\|$ ist.

Schritt (2): Schrittweitenberechnung für aktuellen Schritt

Wenn $\|\varepsilon_d\|$ größer als eine vorgegebene Fehlertoleranz *tol* ist, soll die verwendete Schrittweite h verkleinert werden und der durchgeführte Schritt nochmals mit der neuen Schrittweite h_{neu} wiederholt werden.

Aus der Konsistenzabschätzung wissen wir, dass für den Fehler die Abschätzung

$$\|\varepsilon\| \leq Ch^{p+1}$$

gilt, wobei p bekannt, C aber unbekannt ist. Im schlechtesten Fall gilt hierbei Gleichheit. Wir wollen nun aus der verwendeten Schrittweite h und der Fehlerschätzung $\|\varepsilon_d\|$ eine Schrittweite h_{neu} berechnen, so dass der erwartete Fehler $\|\varepsilon_{neu}\|$ für h_{neu} die Abschätzung

$$\|\varepsilon_{neu}\| \leq C h_{neu}^{p+1} \leq tol$$

für eine vorgegebene Fehlertoleranz tol gilt. Aus

$$\|\varepsilon_d\| \approx \|\varepsilon\| = C h^{p+1}$$

erhalten wir

$$C \approx \frac{\|\varepsilon_d\|}{h^{p+1}}.$$

Um die Bedingung

$$C h_{neu}^{p+1} \approx \frac{\|\varepsilon_d\|}{h^{p+1}} h_{neu}^{p+1} \leq tol$$

zu erfüllen, muss also

$$h_{neu} \leq \sqrt[p+1]{\frac{tol}{\|\varepsilon_d\|}}\, h$$

gewählt werden. Um etwaige Ungenauigkeiten auszugleichen, wird hierbei üblicherweise

$$h_{neu} = \eta \sqrt[p+1]{\frac{tol}{\|\varepsilon_d\|}}\, h$$

gesetzt, wobei η eine Konstante kleiner 1 ist, z.B. $\eta = 0.9$. Außerdem muss sichergestellt werden, dass $h_{neu} \leq h_0$ für eine vorgegebene maximale Schrittweite $h_0 > 0$ ist.

Mit dieser Wahl der Schrittweite wird (im Rahmen der Genauigkeit der Fehlerschätzung) garantiert, dass die Abschätzung

$$\|\varepsilon\| \leq tol$$

gilt. Beachte, dass $\|\varepsilon\|$ hier die Genauigkeit der Lösung $x_{i+1} = \Phi(h_{neu}, t_i, x_i)$ des weniger genauen Verfahrens Φ ist. Nachdem man zur Berechnung von $\|\varepsilon_d\|$ aber sowieso schon einen Schritt mit dem genaueren Verfahren $\widehat{\Phi}$ durchgeführt hat, bietet es sich an, $x_{i+1} = \widehat{\Phi}(h_{neu}, t_i, x_i)$ zj setzen, womit man in der Regel noch einen weiteren Genauigkeitsgewinn erzielen kann.

Schritt (3): Schrittweitenberechnung für nächsten Schritt

Der Schrittweitenvorschlag für den nächsten Schritt wird mit der gleichen Formel wie in Schritt (2) berechnet: Wenn h_i den im vorherigen Schritt verwendeten Zeitschritt bezeichnet, so wählt man

$$h_{i+1} = \eta \sqrt[p+1]{\frac{tol}{\|\varepsilon_d\|}}\, h_i.$$

Beachte, dass diese Berechnung wichtig für eine effiziente Wahl der Schrittweite ist, da in Schritt (2) nur verkleinert wird, während hier auch eine Vergrößerung möglich ist, wenn der Fehler $\|\varepsilon_d\|$ größer als tol ist.

5.6 Eingebettete Verfahren

Zur Berechnung der neuen Schrittweite muss — neben der Auswertung des Verfahrens höherer Konsistenzordnung Φ — in jedem Schritt auch das Verfahren niedrigerer Konsistenzordnung $\widehat{\Phi}$ ausgewertet werden, was einen hohen Aufwand darstellt, da sich die Anzahl der nötigen Auswertungen von f (die im Allgemeinen den "teuren" Anteil des Verfahrens ausmachen) in etwa verdoppelt.

Der Aufwand dieser Berechnung kann durch die Verwendung sogenannter *eingebetteter Runge–Kutta–Verfahren* deutlich reduziert werden. Die Idee dabei ist, dass diejenigen k_i, die zur Berechnung von Φ benötigt werden, auch in der Berechnung von $\widehat{\Phi}$ vorkommen, so dass Φ ohne weitere Auswertungen von f aus $\widehat{\Phi}$ berechnet werden kann. Hierdurch erhöht sich zwar die zum Erreichen einer vorgegebenen Konsistenzordnung p nötige Stufenzahl zwar im Vergleich zur Tabelle in Bemerkung 5.8(i) geringfügig, was durch die effiziente Programmierung der Fehlerschätzung aber mehr als ausgeglichen werden kann. Zudem können durch geschickte Wahl der Koeffizienten die Berechnung verschiedener Werte in einem Schritt ausgeführt werden.

Diese eingebetteten Verfahren lassen sich ebenfalls in einem Butcher–Tableau darstellen, wobei wir jetzt zwei b_i–Zeilen erhalten, nämlich eine für $\widehat{\Phi}$ ($\hat{b}_i$) und eine für Φ (b_i). Typische Verfahren sind ein auf dem klassischen Runge–Kutta Verfahren basierende Verfahren mit den Konsistenzordnungen 4(3) (links, hiebei ist $\widehat{\Phi}$ gerade das klassische Runge–Kutta Verfahren) und das Dormand–Prince 5(4) Verfahren (rechts). Dieses Verfahren steht in MATLAB unter `ode45` zur Verfügung.

0					
$\frac{1}{2}$	$\frac{1}{2}$				
$\frac{1}{2}$	0	$\frac{1}{2}$			
1	0	0	1		
1	$\frac{1}{6}$	$\frac{2}{6}$	$\frac{2}{6}$	$\frac{1}{6}$	
$\hat{b}_i$:	$\frac{1}{6}$	$\frac{2}{6}$	$\frac{2}{6}$	$\frac{1}{6}$	0
b_i :	$\frac{1}{6}$	$\frac{2}{6}$	$\frac{2}{6}$	0	$\frac{1}{6}$

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
$\hat{b}_i$:	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
b_i :	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

Literaturverzeichnis

[1] P. Deuflhard and F. Bornemann, *Numerische Mathematik. II: Integration gewöhnlicher Differentialgleichungen*, de Gruyter, Berlin, 1994.

[2] P. E. Kloeden, *Einführung in die Numerische Mathematik.* Vorlesungsskript, J.W. Goethe–Universität Frankfurt am Main, 2002. Veröffentlichung in Vorbereitung, nach Fertigstellung erhältlich unter `www.math.uni-frankfurt.de/~numerik/kloeden/`.

[3] F. Lempio, *Numerische Mathematik I: Methoden der Linearen Algebra*, Bayreuther Mathematische Schriften, Band 51, 1997.

[4] ——, *Numerische Mathematik II: Methoden der Analysis*, Bayreuther Mathematische Schriften, Band 56, 1998.

[5] W. Oevel, *Einführung in die Numerische Mathematik*, Spektrum Verlag, Heidelberg, 1996.

Index