Datenverarbeitung für TPH II

P. Hansmann, K. Held, H. Leeb, C. Lemell, H. Müller

Fakultät für Physik

Technische Universität Wien

Diplomstudiengang für Technische Physik

Version vom

8. März 2011

Inhaltsverzeichnis

Vorwort

Die Beschreibung von beobachteten Phänomenen und Zusammenhängen mittels mathematischer Methoden ist ein wichtiges Anliegen der Physik und stellt auch die Basis für deren Anwendung in der Technik dar. Der quantitative Auswertung der entsprechenden Theorien erfordert numerische Rechnungen, die in der ersten Hälfte des letzten Jahrhunderts durch die beschränkte Schnelligkeit des Menschen auf wenige schematische Beispiele bzw. grobe Näherungen beschränkt waren. Die Entwicklung von Computern in den letzten Jahrzehnten hat die Lösungskapazität wesentlich erhöht, sodass wir heute auch komplexere Probleme quantitativ erfassen können. Die erzielten Leistungssteigerungen der Computer sind nicht nur auf die in den letzten Jahren erfolgte Steigerung der Prozessgeschwindigkeiten zurückzuführen, sondern auch auf signifikante Verbesserungen im Speicherbereich sowie im Einsatz von massivem Parallelismus. Diese Entwicklungen auf dem Rechnersektor haben große Auswirkungen auf die Gesellschaft im Allgemeinen. Im Bereich der Physik hat der Einsatz von Computern zum Teil methodische Veränderungen bewirkt. Insbesondere die Möglichkeit der Simulation von komplexen Systemen führte zur Etablierung eines neuen Zweiges der Physik, den man als *Numerische Physik* bezeichnet.

Numerische Methoden zur Lösung der ein physikalisches Problem beschreibenden mathematischen Zusammenhänge sind die Grundvoraussetzung für den Einsatz von Rechnern in der Physik. Die Lehrveranstaltung *Grundlagen der elektronischen Datenverarbeitung II* soll die wichtigsten numerischen Techniken vermitteln. Lehrziel der Lehrveranstaltung ist das Verständnis der grundlegenden Algorithmen wie sie in der Physik zum Einsatz kommen, die Verwendung von Programmbibliotheken und die konkrete Umsetzung physikalischer Problemstellungen. Der pragmatische Zugang steht dabei im Vordergrund, während die Behandlung im Sinne der *Numerischen Mathematik* nur auf das Notwendigste beschränkt wird. In der Lehrveranstaltung können nur sehr wenige Methoden behandelt werden. Für einen umfassenderen Überblick wird auf die *Numerical Recipes* [6] verwiesen, die nicht nur eine Beschreibung der mathematischen Grundlagen geben, sondern auch entsprechende Programme in C++ (ältere Versionen des Buches mit Programmen in FORTRAN77, Fortran90 und C können auf der webseite des Verlages gefunden werden).

Kapitel 1

Spezielle Funktionen

1.1 Einleitung

In diesem Kapitel über *Spezielle Funktionen* wollen wir einige für die mathematische Behandlung physikalischer Probleme nützliche Funktionen näher betrachten. Speziell werden wir auf die Generierung der Gamma-Funktion, der Legendre-Polynome und der sphärischen Bessel- und Neumannfunktionen eingehen. Weitere *Spezielle Funktionen* findet man in den kommerziell verfügbaren Programmbibliotheken wie z.B. die NAG- oder die IMSL-Library. Die Programme dieser Bibliotheken sind meist sehr umfangreich, da sie als *Black Boxes* verwendet werden und eine sorgfältige Kontrolle der Genauigkeit, auch für Sonderfälle, garantieren müssen. Bei den nachstehend angeführten Funktionen skizzieren wir nur die mathematischen Grundlagen, welche für die numerische Darstellung verwendet werden können.

1.2 Gamma-Funktion

In vielen Problemen der Physik und der Statistik wird die Gamma-Funktion $\Gamma(z)$ bzw. die Faktorielle benötigt. So tritt sie z.B. in quantenmechanischen Problemen der Atom- und Kernphysik in Normierungskonstanten auf. Die Gamma-Funktion ist über das Integral

$$\Gamma(z) = \int_0^\infty dt\ t^{z-1} e^{-t} \tag{1.1}$$

definiert. Ist das Argument eine natürliche Zahl, $z = n$ ($n \in \mathcal{N}$), so entspricht die Gamma-Funktion der Faktoriellen,

$$n! = \Gamma(n+1)\,. \tag{1.2}$$

Für allgemeines z erfüllt die Gamma-Funktion die Rekursionsbeziehung

$$\Gamma(z+1) = z\ \Gamma(z) \tag{1.3}$$

Kennt man die Gamma-Funktion für z-Werte in der gesamten Halbebene $\mathrm{Re}z > 1$, so lassen sich über die Reflexionsformel,

$$\Gamma(1-z) = \frac{\pi}{\Gamma(z)\sin(\pi z)} = \frac{\pi z}{\Gamma(1+z)\sin(\pi z)}\,, \tag{1.4}$$

auch die Werte der Gamma-Funktion für $\mathrm{Re}z < 1$ bestimmen. Die Gamma-Funktion hat Pole bei $z = 0$ und bei allen negativen ganzen Zahlen.

Für die numerische Berechnung von $\Gamma(z)$ kann man von einer der speziellen Darstellungen für bestimmte Intervalle ausgehen (siehe z.B. [1], Kapitel 6) und durch Anwendung der Rekursionsbeziehung (1.3) und der Reflexionsformel (1.4) die gesamte komplexe z-Ebene abdecken. Eine der Näherungen für reelle Argumentwerte x aus dem Intervall $[1, 2[$ ist von der Form

$$\Gamma(x) \approx \sum_{k=0}^{8} a_k (x-1)^k\,, \tag{1.5}$$

mit den Konstanten

a_0	=	1.000000	a_3	=	−0.897057	a_6	=	0.482199
a_1	=	−0.577192	a_4	=	0.918207	a_7	=	−0.193528
a_2	=	0.988206	a_5	=	−0.756704	a_8	=	0.035868

Im allgemeinen ist es günstiger den Logarithmus der Funktion, $\ln\Gamma(z)$, numerisch zu implementieren. Dadurch lässt sich ein *floating point overflow* des Rechners vermeiden. Die Rekursionsbeziehung (1.3) reduziert sich dann zu einer Summe

$$\ln\Gamma(z+N) = \sum_{n=0}^{N-1} \ln(z+n) + \ln\Gamma(z)\,. \tag{1.6}$$

Wählt man nun die Zahl N hinreichend groß, so kann man die folgende asymptotische Form zur Berechnung von $\ln\Gamma(z+N)$ verwenden (siehe [1])

$$\ln\Gamma(\bar{z}) \approx (\bar{z}-\tfrac{1}{2})\ln\bar{z} - \bar{z} + \tfrac{1}{2}\ln(2\pi) + \frac{1}{12\bar{z}} - \frac{1}{360\bar{z}^2} + \frac{1}{1260\bar{z}^5} - \frac{1}{1680\bar{z}^7} + \cdots \tag{1.7}$$

with $\bar{z} = z + N \to \infty$ in $|\arg\bar{z}| < \pi$. Für $\mathrm{Re}z < 1$ muss zusätzlich noch die Reflexionsformel (1.4) eingesetzt werden.

Zum Abschluss sei noch die spezielle Näherung von Lanczos genannt [1],

$$\begin{aligned}\Gamma(1+z) &= (z+\gamma+\tfrac{1}{2})^{z+0.5} e^{-(z+\gamma+0.5)} \\ &\times \sqrt{2\pi}\left[1 + \frac{c_1}{z+1} + \frac{c_2}{z+2} + \frac{c_5}{z+5} + \epsilon\right] \quad \text{für } \mathrm{Re}z > 0\end{aligned} \tag{1.8}$$

mit $\gamma = 5$ und den Konstanten

c_1	=	76.18009173	c_2	=	−86.50532033	c_3	=	24.01409822
c_4	=	−1.231739516	c_5	=	0.00120858003	c_6	=	0.000053682

Diese Formel kann in der gesamten komplexen Halbebene $\mathrm{Re}z > 0$ angewendet werden, wobei der Fehler $|\epsilon| < 2 \times 10^{-10}$ ist.

1.3 Legendre Polynome und Kugelflächenfunktionen

Kugelflächenfunktionen, $Y_{\ell m}(\theta, \varphi)$ werden in einer Vielzahl physikalischer Probleme verwendet. Ihre Verwendung ist besonders dann vorteilhaft, wenn das betrachtete Problem eine sphärische Symmetrie aufweist, wie dies z.B. in vielen Fragestellungen der Atom- und Kernphysik der Fall ist.

Die Kugelflächenfunktionen sind auf der Einheitskugel definiert und stellen einen orthonormierten Satz von Basisfunktionen dar,

$$\int_0^{2\pi} d\varphi \int_{-1}^{+1} d(\cos\theta) Y^*_{\ell',m'}(\theta,\varphi) Y_{\ell m}(\theta,\varphi) = \delta_{\ell'\ell}\delta_{mm'}\,, \tag{1.9}$$

wobei (*) komplexe Konjungation bedeutet.

Die Kugelflächenfunktionen haben die Form

$$Y_{\ell m}(\theta,\varphi) = \sqrt{\frac{2\ell+1}{4\pi}\frac{(\ell-m)!}{(\ell+m)!}}\; P_\ell^m(\cos\theta) e^{im\varphi} \tag{1.10}$$

und es gilt die Beziehung

$$Y_{\ell,-m}(\theta,\varphi) = (-1)^m Y^*_{\ell m}(\theta,\varphi)\,. \tag{1.11}$$

Die Abhängigkeit der Kugelflächenfunktionen vom Winkel θ ist durch die assoziierten Legendre Polynome, $P_\ell^m(\cos\theta)$, gegeben. Diese hängen über die Beziehung

$$P_\ell^m(x) = (-1)^m (1-x^2)^{m/2} \frac{d^m}{dx^m} P_\ell(x) \tag{1.12}$$

mit den gewöhnlichen Legendre Polynomen, $P_\ell(x)$, zusammen. An dieser Stelle soll bemerkt werden, dass die Position der magnetischen Quantenzahl m als oberer oder unterer Index beachtet werden muss. Es gilt

$$P_{\ell m}(x) = (-1)^m P_\ell^m(x)\,. \tag{1.13}$$

Die assoziierten Legendre Polynome niedriger Ordnung und die entsprechenden Kugelflächenfunktionen haben eine einfache Form und sind in der Tabelle 1.3 angegeben.

Bei der Berechnung der assoziierten Legendre Polynome sollte man nicht versuchen, die Polynomdarstellung numerisch zu implementieren, da dies zu Auslöschungsfehler bei der Addition benachbarter Terme mit entgegengesetztem Vorzeichen führt. Außerdem werden bei höheren Legendre Polynomen die Einzelterme wesentlich größer als die Summe, sodass zusätzlich Genauigkeit verloren wird. Auf diese Art können bei doppelt genauer Rechnung (REAL*8) nur Polynome bis etwa $\ell = 18$ berechnet werden.

$P_{00}(x) =$	1	$Y_{00} =$	$\sqrt{\frac{1}{4\pi}}$
$P_{11}(x) =$	$(1-x^2)^{1/2}$	$Y_{11} =$	$-\sqrt{\frac{3}{8\pi}}\sin\theta e^{i\varphi}$
$P_{10}(x) =$	x	$Y_{10} =$	$\sqrt{\frac{3}{4\pi}}\cos\theta$
$P_{22}(x) =$	$3(1-x^2)$	$Y_{22} =$	$\frac{1}{4}\sqrt{\frac{15}{2\pi}}\sin^2\theta e^{2i\varphi}$
$P_{21}(x) =$	$3x(1-x^2)^{1/2}$	$Y_{21} =$	$-\sqrt{\frac{15}{8\pi}}\sin\theta e^{i\varphi}$
$P_{20}(x) =$	$\frac{1}{2}(3x^2-1)$	$Y_{20} =$	$\sqrt{\frac{5}{4\pi}}(\frac{3}{2}\cos^2\theta-\frac{1}{2})$

Tabelle 1.1: Assoziierte Legendre Polynome und Kugelflächenfunktionen niedrigster Ordnung

Ein in Hinblick auf die numerische Genauigkeit robusterer Algorithmus kann auf der Basis der Rekursionsbeziehung

$$(\ell-m)P_{\ell m}(x) = (2\ell-1)xP_{\ell-1,m}(x) - (\ell+m-1)P_{\ell-2,m}(x) \tag{1.14}$$

aufgebaut werden. Startpunkt der Rekursion sind die Polynome $P_{\ell\ell}(x)$, die durch

$$P_{\ell\ell}(x) = (2\ell-1)!!(1-x^2)^{\ell/2} \tag{1.15}$$

gegeben sind. Setzt man außerdem $P_{\ell,m=\ell+1} = 0$, so lässt sich die Rekursion bis zu höchsten ℓ-Werten stabil durchführen. Man geht dabei in folgender Reihenfolge vor:

m	Berechnung nach (1.15)	Berechnung über Rekursionsformel (1.14)
0	P_{00}	$P_{10}, P_{20}, P_{30}, P_{40}, \ldots$
1	P_{11}	$P_{21}, P_{31}, P_{41}, \ldots$
2	P_{22}	$P_{32}, P_{42}, \ldots$
3	P_{33}	$P_{43}, \ldots$
$\vdots$	$\vdots$	$\vdots$

Abschließend soll noch darauf aufmerksam gemacht werden, dass nicht jede Rekursionsformel zu einem stabilen Algorithmus führt. Wir werden diesen Punkt noch genauer bei den sphärischen Bessel Funktionen behandeln. Im Fall der Legendre Polynome gibt

es eine Reihe anderer Rekursionsbeziehungen, die meist jedoch nicht zu einem stabilen Algorithmus führen und daher für numerische Berechnung der assoziierten Legendre Polynome nicht geeignet sind.

1.4 Sphärische Bessel Funktionen

Ein wichtiges Funktionensystem bei der Lösung der Schrödingergleichung stellen die sphärischen Bessel- und Neumannfunktionen dar. Diese sind Lösungen der gewöhnlichen Differentialgleichung

$$\left\{ \frac{d^2}{d\rho^2} + \frac{2}{\rho}\frac{d}{d\rho} + 1 - \frac{\ell(\ell+1)}{\rho^2} \right\} R_\ell(\rho) = 0\,, \tag{1.16}$$

und entspricht der radialen Schrödingergleichung für ein freies Teilchen. Die Differentialgleichung (1.16) hat zwei linear unabhängige Lösungen. Diese unterscheiden sich durch ihr Verhalten bei $\rho = 0$,

- reguläre Lösung bei $\rho = 0$: spärische Besselfunktion $j_\ell(\rho)$
- irreguläre Lösung bei $\rho = 0$: sphärische Neumannfunktion $n_\ell(\rho)$

Die mathematischen Eigenschaften der sphärischen Bessel- und Neumannfunktionen sind gut untersucht. Wichtige mathematische Zusammenhänge sind z.B. im Tabellenwerk von Abramowitz und Stegun [1] zusammengefasst. Die Funktionen zeigen unter anderem das folgende Verhalten am Ursprung

$$j_\ell(\rho) \overset{\rho\to 0}{\longrightarrow} \frac{\rho^\ell}{(2\ell+1)!!}\left(1 - \frac{\rho^2}{2(2\ell+3)} + \mathcal{O}(\rho^4)\right)\,, \tag{1.17}$$

$$n_\ell(\rho) \overset{\rho\to 0}{\longrightarrow} \frac{(2\ell-1)!!}{\rho^{\ell+1}}\left(1 + \frac{\rho^2}{2(2\ell-1)} + \mathcal{O}(\rho^4)\right)\,. \tag{1.18}$$

Das asymptotische Verhalten ist durch

$$j_\ell(\rho) \overset{\rho\to\infty}{\longrightarrow} \frac{\sin(\rho - \ell\frac{\pi}{2})}{\rho}\,, \tag{1.19}$$

$$n_\ell(\rho) \overset{\rho\to\infty}{\longrightarrow} -\frac{\cos(\rho - \ell\frac{\pi}{2})}{\rho} \tag{1.20}$$

gegeben. Für die Bessel- und Neumannfunktionen können geschlossene Ausdrücke angegeben werden,

$$\ell = 0 \quad j_0(\rho) = +\tfrac{\sin\rho}{\rho} \quad , \qquad n_0(\rho) = -\tfrac{\cos\rho}{\rho} \quad ,$$

$$\ell = 1 \quad j_1(\rho) = -\tfrac{\cos\rho}{\rho} + \tfrac{\sin(\rho)}{\rho^2} \quad , \qquad n_1(\rho) = -\tfrac{\sin\rho}{\rho} - \tfrac{\cos(\rho)}{\rho^2} \quad ,$$

$$\ell = 2 \quad j_2(\rho) = -3\tfrac{\cos\rho}{\rho^2} + \left(\tfrac{3}{\rho^3} - \tfrac{1}{\rho}\right)\sin\rho \quad , \quad n_2(\rho) = -3\tfrac{\sin\rho}{\rho^2} - \left(\tfrac{3}{\rho^3} - \tfrac{1}{\rho}\right)\cos\rho \quad .$$

Die numerische Berechnung der spärischen Bessel- und Neumannfunktionen erfolgt am Besten über die Rekursionsformeln

$$g_{\ell+1}(\rho) + g_{\ell-1}(\rho) = \frac{2\ell+1}{\rho} g_\ell(\rho)\,, \tag{1.21}$$

wobei $g_\ell(\rho)$ für eine sphärische Bessel- oder Neumannfunktion steht. Analog zu der Berechnung der Legendrefunktionen führt man auch für die sphärischen Neumannfunktionen eine Rekursion durch, wobei man von $n_0(\rho)$ und $n_1(\rho)$ ausgeht. Bei den sphärischen Neumannfunktionen ist die Vorwärtsrekursion stabil und führt für alle Argumentwerte auf zuverlässige Ergebnisse.

Für die Bestimmung von $j_\ell(\rho)$ ist die Vorwärtsrekursion leider nicht ausreichend stabil für $\ell > \rho$. Die Ursache für die Probleme bei der Vorwärtsrekursion lässt sich aus der folgenden Umformung der Rekursionsvorschrift erkennen,

$$j_{\ell+1}(\rho) - 2j_\ell(\rho) + j_{\ell-1}(\rho) = \frac{2\ell+1-2\rho}{\rho} j_\ell(\rho)\,. \tag{1.22}$$

Wie wir in Kapitel 4 sehen werden, kann die Gleichung (1.22) als Differenzendarstellung der Differentialgleichung

$$\frac{d^2}{d\lambda^2} j_\lambda(\rho) = \frac{2\lambda+1-2\rho}{\rho} j_\lambda(\rho) \tag{1.23}$$

aufgefasst werden. Eine solche Differentialgleichung hat zwei linear unabhängige Lösungen. Ist $(2\lambda + 1 - 2\rho) < 0$, so erhält man 2 in λ oszillierende Lösungen. Bei $(2\lambda + 1 - 2\rho) > 0$ erhält man eine exponentiell in λ ansteigende und eine abfallende Lösung. Für die Verlässlichkeit des Rekursionsverfahrens ist es nun maßgebend, dass die divergierende Lösung unterdrückt wird. Dies ist bei der Vorwärtsrekursion der sphärischen Neumannfunktionen der Fall, nicht aber bei der Rekursion der sphärischen Besselfunktionen, d.h. bei ansteigenden $\lambda > \rho$ wird der divergierende Term ab einem gewissen Punkt dominant werden.

Diese prinzipielle Eigenschaft der Rekursion kann man zur Berechnung der sphärischen Besselfunktion ausnützen, indem man eine sogenannte Rückwertsrekursion durchführt. Man startet die Rekursion bei einem sehr hohen ℓ-Wert, $\ell = L > \rho$. Für die numerische Rechnung kann man aufrund des oben Gesagten annehmen, dass die divergierende Lösung sicherlich den numerischen Ansatz dominiert. Man kann daher beliebige Werte für $j_L(\rho)$ und $j_{L+1}(\rho)$ ansetzen. Führt man nun die Rekursion rückwärts durch, d.h. man berechnet sukzessive $j_{L-1}, j_{L-2}, \ldots j_1, j_0$ über die Rekursionsformel (1.21), so verschwindet der divergierende Anteil sehr rasch und es verbleibt nur die für die sphärischen Besselfunktionen relevante Lösung. Die erhaltenen Werte sind allerdings noch Vielfaches der entsprechenden j_ℓ-Werte und müssen durch Vergleich mit $j_0(\rho)$ normiert werden. Mehr Details über die Rückwärtsrekursion wurden von Gillman und Fiebig [2] angegeben.

Kapitel 2

Interpolation, Differentiation und Integration

2.1 Interpolation

2.1.1 Motivation und Klassifikation

Ein zentrales Thema bei der numerischen Behandlung von Fragestellungen stellt die Wahl einer geeigneten Interpolation zur Darstellung von Funktionen im Computer dar. Die Motivation für die Verwendung von Interpolationen ist vielfältig, z.B.

- die Zahl der Messpunkte ist gering,
- der Aufwand zur Berechnung eines Datenpunktes ist sehr hoch, d.h. man versucht möglichst wenige Punkte zu berechnen,
- die numerische Methode verlangt ein bestimmtes Gitter, die Mess- oder Rechenpunkte liegen aber auf einem anderen Gitter vor.

Diese Probleme treten sowohl in der Auswertung von Messreihen als auch bei numerischen Rechnungen im Rahmen einer Theorie oder eines vorgegebenen Modells auf. Der Ausgangspunkt für eine Interpolation ist stets ähnlich:

- Gegeben ist eine unabhängige Variable x, und für eine Menge von Werten $\{x_i\}$ aus einem Intervall $\mathcal{I}$ sind Funktionswerte $f(x_i) = f_i$ bekannt. Gesucht ist eine Funktion $\bar{f}(x)$ (eventuell geschlossener Ausdruck), sodass für beliebiges x aus $\mathcal{I}$ mit $\bar{f}(x)$ eine sinnvolle Näherung von $f(x)$ berechnet werden kann.
- Ziel der Interpolation ist nicht nur die Verwendung von $\bar{f}(x)$ zur näherungsweisen Berechnung von Funktionswerten $f(x)$ mit $x \in \mathcal{I}$ (Interpolation), sondern auch als Ausgangspunkt näherungsweiser Integrations-, Differentiations- und in einigen Fällen Extrapolationsmethoden.

Die entsprechende mathematische Problemstellung lässt sich kompakt wie folgt formulieren:

Interpolationsproblem:

Eine Funktion $f(x)$ sei an $n+1$ unterschiedlichen Argumentwerten $x_i,\ i = 0, 1, \ldots, n$, $x_i \neq x_k$ gegeben, $f_i = f(x_i)$. Ein Interpolationsproblem liegt dann vor, wenn für eine vorgegebene Funktionenklasse $\phi(x; a_0, a_1, \ldots, a_n)$ die Parameter a_i so bestimmt werden sollen, dass

$$\phi(x_i; a_0, a_1, \ldots, a_n) = f_i \quad \forall i = 0, 1, \ldots, n$$

Die Paare (x_i, f_i) werden als Stützstellen bezeichnet.

Eine besondere Klasse sind lineare Interpolationsprobleme. Bei diesen hängt die Interpolationsfunktion $\phi(x; a_0, a_1, \ldots, a_n)$ linear von den Parametern a_i ab,

$$\phi(x; a_0, a_1, \ldots, a_n) = \bar{\phi}(x) + a_0\bar{\phi}_0(x) + a_1\bar{\phi}_1(x) + \cdots + a_n\bar{\phi}_n(x)\,. \tag{2.1}$$

Die in diesem Skriptum behandelten Interpolationstechniken, (a) die Polynominterpolation und (b) die Spline-Interpolation, gehören beide der Klasse der linearen Interpolationsprobleme an. Beispiele für nichtlineare Interpolationsprobleme sind die Interpolation mit rationalen Funktionen sowie die Interpolation mit Exponentialsummen.

2.1.2 Polynominterpolation

Bei der Polynominterpolation wählt man die Interpolationsfunktion aus Π_n, der Menge der reellen oder komplexen Polynome $p(x)$ vom Grad $\leq n$,

$$\phi(x; a_0, a_1, \ldots, a_n) = p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n\,. \tag{2.2}$$

Das Interpolationsproblem besteht nun darin, dass man zu $(n+1)$ beliebig vorgegebenen Stützstellen (x_i, f_i) ein Polynom $p(x) \in \Pi_n$ bestimmt, welches an den $(n+1)$ Stützstellen mit der vorgegebenen Funktion übereinstimmt, d.h. $p(x_i) = \bar{f}(x_i) = f_i$ für $i = 0, 1, 2, \ldots, n$.

Es lässt sich nun zeigen, dass ein eindeutig bestimmtes Polynom $p(x)$ existiert, welches dieses Interpolationsproblem löst. Die Eindeutigkeit lässt sich anhand eines indirekten Beweises zeigen:

Angenommen, es existierten zwei Polynome $p_1, p_2 \in \Pi_n$ für die gilt: $p_1(x_i) = p_2(x_i) = f_i$ für alle $i = 0, 1, 2, \ldots, n$. Daraus würde folgen, dass das Polynom $p(x) = p_1(x) - p_2(x) \in \Pi_n$ zumindest $n+1$ Nullstellen $[\, p(x_i) = 0,\ i = 0, 1, \ldots, n]$ hätte. Da $p \in \Pi_n$ nur maximal n Nullstellen haben kann, folgt $p(x) \equiv 0$ und $p_1(x) \equiv p_2(x)$.

Die Existenz eines Polynoms $p(x)$, welches das Interpolationsproblem löst, ist durch die *Lagrange Interpolationsformel*

$$p(x) = \sum_{i=0}^{n} f_i L_i(x) \tag{2.3}$$

gegeben, wobei $L_i(x)$ die *Lagrangeschen Interpolationspolynome* sind

$$L_i(x) = \prod_{k=0, k\neq i}^{n} \frac{x - x_k}{x_i - x_k} \quad \text{für} \quad i = 0, 1, \ldots, n \tag{2.4}$$

Da alle $L_i(x) \in \Pi_n$ sind und $L_i(x_k) = \delta_{ik}$ gilt, liefert die Lagrangesche Interpolationsformel offensichtlich das gewünschte Interpolationspolynom.

Da die zu interpolierenden Funktionen selbst meist keine Polynome sind, wird bei der Interpolation ein Fehler gemacht. Sei $p_n(x)$ das Interpolationspolynom vom Grad n, dann ist der Fehler, $E(x) = f(x) - p_n(x)$ proportional zu [3]:

$$E(x) = \Pi_{i=0,n}(x - x_i)\frac{f^{(n+1)}(\xi)}{(n+1)!} \tag{2.5}$$

mit $\xi \in [x_0, x_n]$.

Für die praktische Verwendung in einem Computerprogramm ist die Lagrangesche Interpolationsformel ungeeignet. Sie benötigt viele arithmetischen Operationen und ist daher auch anfällig für Rundungsfehler. Je nachdem ob man die Funktion an einem oder mehreren Argumentwerten interpolieren will, bieten sich verschiedene Alternativen an.

Der Algorithmus von Neville

Will man den interpolierten Funktionswert $\bar{f}(x)$ nur an einem (wenigen) zusätzlichen Argumentwert(en) berechnen, bietet sich der Algorithmus von Neville zur Lösung des Interpolationsproblems an. Der Algorithmus ist flexibel bezüglich der Zahl der Datenpunkte und aufgrund der rekursiven Formulierung leicht implementierbar.

Wieder sei die Funktion an $(n+1)$ Stützstellen $(x_i, f_i),\ i = 0, 1, \ldots, n$ gegeben. Ausgehend von Polynomen nullten Grades

$$P_i(x) = f(x_i) = f_i \quad \forall\, x\,, \tag{2.6}$$

können wir mit der Rekursionsvorschrift

$$P_{i_j i_{j+1} \ldots i_{j+k}}(x) = \frac{(x - x_{i_j})P_{i_{j+1} \ldots i_{j+k}}(x) - (x - x_{i_{j+k}})P_{i_j \ldots i_{j+k-1}}(x)}{x_{i_{j+k}} - x_{i_j}} \tag{2.7}$$

den interpolierten Wert der Funktion an der Stelle x ermitteln,

$$p_n(x) = P_{i_0 i_1 \ldots i_n}(x)\,. \tag{2.8}$$

Der Algorithmus von Neville lässt sich in einem Schema übersichtlich darstellen und numerisch leicht implementieren. Als Beispiel betrachten wir das Schema für $n = 3$

	$k=0$	$k=1$	$k=2$	$k=3$
x_0	$f_0 = P_0(x)$			
		$P_{01}(x)$		
x_1	$f_1 = P_1(x)$		$P_{012}(x)$	
		$P_{12}(x)$		$P_{0123}(x)$
x_2	$f_2 = P_2(x)$		$P_{123}(x)$	
		$P_{23}(x)$		
x_3	$f_3 = P_3(x)$			

Man rechnet dabei spaltenweise und arbeitet sich zur höheren Ordnung (nach rechts) vor.

Newtonsche Interpolationsformel

Benötigt man die Funktion $\bar{f}(x)$ an mehreren Punkten, bzw. will man das Polynom selbst bestimmen, eignet sich der Algorithmus von Newton, welcher auf der Basis *Dividierter Differenzen* arbeitet.

Die Dividierte Differenz (*divided difference*) erster Ordnung zwischen zwei Stützstellen (x_i, f_i) und (x_j, f_j) ist folgendermaßen definiert

$$f[x_i, x_j] = \frac{f_i - f_j}{x_i - x_j} = f[x_j, x_i]. \tag{2.9}$$

Ausgehend von der Dividierten Differenz nullter Ordnung, $f[x_i] = f_i$, können nun rekursiv Dividierte Differenzen höherer Ordnung definiert werden,

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}. \tag{2.10}$$

Das gesuchte Interpolationspolynom $p_n(x)$, welches durch die $(n+1)$ Stützstellen (x_i, f_i) geht, ist gegeben durch

$$p_n(x) = a_0 + (x-x_0)a_1 + (x-x_0)(x-x_1)a_2 + \cdots + (x-x_0)(x-x_1)\ldots(x-x_{n-1})a_n, \tag{2.11}$$

wobei die Koeffizienten a_i durch

$$a_i = f[x_0, x_1, \ldots, x_i]. \tag{2.12}$$

gegeben sind. Die rekursive Bestimmung der $f[x_0, x_1, \ldots, x_i]$ und somit der a_i gemäß Gl. 2.10 macht eine sehr einfache numerische Bestimmung der Polynomkoeffizienten möglich.

Der rekursive Algorithmus (Gl. 2.10) kann leicht in ein symmetrisches Schema gebracht werden. Für die Stützstellen $\{(3.2, 22.0), (2.7, 17.8), (1.0, 14.2), (4.8, 38.3), (5.6, 51.7)\}$ ergeben sich folgende Werte:

x_i	$f_i = f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, \ldots, x_{i+2}]$	$f[x_i, \ldots, x_{i+3}]$	$f[x_i, \ldots, x_{i+4}]$
3.2	22.0				
		8.400			
2.7	17.8		2.856		
		2.118		-0.5280	
1.0	14.2		2.012		0.256
		6.342		0.0865	
4.8	38.3		2.263		
		16.750			
5.6	51.7				

Man nennt dies ein Differenzenschema nach Newton, bei dem die Dividierten Differenzen spaltenweise und dann von links nach rechts fortschreitend betimmt werden.

Haben die Stützstellen x_i gleichen Abstand voneinander, so ergeben sich wesentliche Vereinfachungen.

2.1.3 „Spline“-Interpolation

Motivation

In manchen Fällen kann die Interpolation eines vorgegebenen Satzes von Funktionswerten aufgrund der Oszillation von Polynomen hoher Ordnung zu unerwünschten Ergebnissen führen. Eine Verbesserung der Situation kann erreicht werden, indem man abschnittsweise (d.h. für eine kleine Untermenge der x_i) Polynome interpoliert, wobei man zusätzliche Randbedingungen an den Kontaktstellen der Polynome fordert.

Im Fall der (kubischen) Spline-Interpolation wird in jedem der Teilintervalle, gegeben durch $[x_i, x_{i+1}]$, ein kubisches Polynom mit unbekannten Koeffizienten a_i, b_i, c_i und d_i definiert, das die Funktionswerte an den Intervallgrenzen, $f(x_i)$ und $f(x_{i+1})$, reproduziert:

$$f(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \tag{2.13}$$

Im Folgenden bezeichnen wir den Abstand zwischen benachbarten Abszissenwerten mit $h_i = (x_{i+1} - x_i)$ und die zweiten Ableitungen der Polynome an den Stützstellen (x_i, f_i) mit S_i.

Aus der Stetigkeitsbedingung für die Funktion $\bar{f}(x)$

$$\begin{aligned} \bar{f}(x_i) &= d_i = f_i \\ \bar{f}(x_{i+1}) &= a_i(x_{i+1} - x_i)^3 + b_i(x_{i+1} - x_i)^2 + c_i(x_{i+1} - x_i) + d_i \\ &= a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i = f_{i+1}. \end{aligned} \tag{2.14}$$

Gleichheit der ersten und zweiten Ableitungen der Polynome an den Intervallgrenzen führt für S_i und S_{i+1} auf

$$S_i = 2b_i \qquad \text{und} \qquad S_{i+1} = 6a_i h_i + 2b_i. \tag{2.15}$$

Mit Hilfe dieser beiden Relationen lassen sich die unbekannten Koeffizienten a_i und b_i folgendermaßen ausdrücken:

$$b_i = \frac{1}{2} S_i \qquad \text{und} \qquad a_i = \frac{1}{6h_i}(S_{i+1} - S_i). \tag{2.16}$$

Die Koeffizienten $\{a_i\}$, $\{b_i\}$ und $\{d_i\}$ sind somit bestimmt, die noch unbestimmten Koeffizienten $\{c_i\}$ ergeben sich aus Gl. 2.14. Man erhält explizit

$$c_i = \frac{f_{i+1} - f_i}{h_i} - \frac{2h_i S_i + h_i S_{i+1}}{6}. \tag{2.17}$$

Anpassung der ersten Ableitungen der Polynome an der Stelle x_i führt für das rechts- bzw. linksseitige Polynom auf:

$$\begin{aligned} \bar{f}'(x_i) &= c_i \\ \bar{f}'(x_i) &= 3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) + c_{i-1} \\ &= 3a_{i-1}h_{i-1}^2 + 2b_{i-1}h_{i-1} + c_{i-1}. \end{aligned} \tag{2.18}$$

Gleichsetzen dieser beiden Relationen und Substitution der Ausdrücke für a_i, b_i, c_i und d_i ergibt

$$\begin{aligned} \frac{f_{i+1} - f_i}{h_i} - \frac{2h_i S_i + h_i S_{i+1}}{6} &= 3\left(\frac{S_i - S_{i-1}}{6h_{i-1}}\right) h_{i-1}^2 + 2\left(\frac{S_{i-1}}{2}\right) h_{i-1} \\ &\quad + \frac{f_i - f_{i-1}}{h_{i-1}} - \frac{2h_{i-1}S_{i-1} + h_{i-1}S_i}{6}, \end{aligned} \tag{2.19}$$

beziehungsweise, vereinfacht,

$$h_{i-1}S_{i-1} + (2h_{i-1} + 2h_i)S_i + h_i S_{i+1} = 6\left(\frac{f_{i+1} - f_i}{h_i} - \frac{f_i - f_{i-1}}{h_{i-1}}\right). \tag{2.20}$$

Insgesamt gibt es damit $(n-1)$ Gleichungen für die $(n+1)$ Unbekannten S_i. Es werden also noch zwei weitere Angaben für S_0 und S_n benötigt, die auf Annahmen beruhen. Folgende Festlegungen der Randwerte werden üblicherweise verwendet:

- $S_0 = S_n = 0$ („natürlicher spline“).
- $S_0 = S_1$ und $S_n = S_{n-1}$.
- S_0 ist ein aus S_1 und S_2 linear extrapolierter Wert; ebenso ist S_n ein aus S_{n-1} und S_{n-2} linear extrapolierter Wert.
- Schätzwerte für die Anstiege an den beiden Enden werden vorgegeben.

Vor- und Nachteile jeder dieser Annahmen müssen im Anlassfall genau untersucht werden.

Das resultierende Gleichungssystems ist tridiagonal, d.h., dass jedes S_i (ausgenommen S_0 und S_n) nur an seine Nachbarn S_{i-1} und S_{i+1} durch lineare Relationen gekoppelt ist. Das macht die Berechnung der S_i nicht nur einfach, sondern auch schnell ($O(N)$-Algorithmus). Sind alle S_i bestimmt, werden die unbekannten Koeffizienten der Interpolations Polynome folgendermaßen bestimmt:

$$a_i = \frac{1}{6h_i}(S_{i+1} - S_i) \tag{2.21}$$

$$b_i = \frac{1}{2}S_i \tag{2.22}$$

$$c_i = \frac{f_{i+1} - f_i}{h_i} - \frac{2h_iS_i + h_iS_{i+1}}{6} \tag{2.23}$$

$$d_i = f_i \tag{2.24}$$

2.2 Differentiation

Die genaue Bestimmung der Ableitungen einer Funktion, die durch einen Satz von gegebenen Datenpunkten $\{(x_i, f_i)\}$ beschrieben ist, gehört zu den grundlegenden Voraussetzungen jeder Kurvendiskussion. Weiters kann sie z.B. bei der Lösung von Differentialgleichungen von großer Bedeutung sein.

2.2.1 Ableitungen über Interpolationspolynome

Unter der Annahme, dass eine Funktion hinreichend gut durch ihr Interpolationspolynom approximiert wird (siehe vorhergehender Abschnitt), kann man hoffen, dass diese Näherung auch für die erste Ableitung verwendbar ist. Der Fehler in der ersten Ableitung ist allerdings größer als bei den Funktionswerten.

Ist das Interpolationspolynom z.B. durch die Newtonsche Interpolationsformel bestimmt, also

$$\begin{aligned} f(x) &= p_n(x) + E(x) \\ &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\ &\quad f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\cdots(x - x_{n-1}) + E(x) \end{aligned} \tag{2.25}$$

so ergibt sich für $f'(x)$

$$\begin{aligned} f'(x) &= f[x_0, x_1] + f[x_0, x_1, x_2](2x - x_0 - x_1) + \cdots \\ &\quad f[x_0, x_1, \ldots, x_n] \sum_{k=0}^{n-1} \frac{(x - x_0)\cdots(x - x_{n-1})}{(x - x_k)} + \tilde{E}(x). \end{aligned} \tag{2.26}$$

Die Abschätzung für $\tilde{E}(x)$ ist komplizierter als jene für $E(x)$. Unter gewissen Voraussetzungen lassen sich aber geschlossene Ausdrücke für diese Fehler angeben [3], aus denen ersichtlich ist, dass $\tilde{E}(x)$ grßer als $E(x)$ ist.

Höhere Ableitungen von $f(x)$ lassen sich formal leicht aus den obigen Ausdrücken herleiten.

2.2.2 Ableitungen bei gleichmäßig verteilten Datenpunkten

Sind die Datenpunkte auf einem regelmäßigen Gitter vorgegeben (äquidistante Datenpunkte), also $x_i = x_0 + ih$, $i = 1, \ldots, n$, können folgende Ausdrücke für die ersten und zweiten Ableitungen von $f(x)$ verwendet werden:

erste Ableitungen:

$$f'(x_0) = \frac{f_1 - f_0}{h} + O(h) \tag{2.27}$$

$$f'(x_0) = \frac{f_1 - f_{-1}}{2h} + O(h^2) \tag{2.28}$$

$$f'(x_0) = \frac{-f_2 + 4f_1 - 3f_0}{2h} + O(h^2) \tag{2.29}$$

$$f'(x_0) = \frac{-f_2 + 8f_1 - 8f_{-1} + f_{-2}}{12h} + O(h^4); \tag{2.30}$$

zweite Ableitungen:

$$f''(x_0) = \frac{f_2 - 2f_1 + f_0}{h^2} + O(h) \tag{2.31}$$

$$f''(x_0) = \frac{f_1 - 2f_0 + f_{-1}}{h^2} + O(h^2) \tag{2.32}$$

$$f''(x_0) = \frac{-f_3 + 4f_2 - 5f_1 + 2f_0}{h^2} + O(h^2) \tag{2.33}$$

$$f''(x_0) = \frac{-f_2 + 16f_1 - 30f_0 + 16f_{-1} - f_{-2}}{12h^2} + O(h^4). \tag{2.34}$$

Durch Indextransformation in den obigen Ausdrücken können Ableitungen für jeden der n Datenpunkte angegeben werden. Zu beachten ist, dass bei Randpunkten einige der Ausdrücke mangels vorhandener Funktionswerte nicht anwendbar sind. Genauere Angaben für die Fehler sind in [1] zu finden.

2.3 Integration

Um das bestimmte Integral

$$I = \int_a^b f(x)\,dx$$

zu lösen, werden im Folgenden zwei Klassen von Methoden diskutiert: einerseits sogenannte *Newton-Cotes Integrationsalgorithmen*, die auf der analytischen Integration stückweis aneinandergereihter Interpolationspolynome durch äquidistante Datenpunkte $\{(x_i, f_i)\}$ beruhen, andererseits *Gauss'sche Integrationsmethoden*, für die die Abszissen $\{x_i\}$ der Stützstellen durch die Wahl der Methode vorbestimmt sind.

2.3.1 Newton-Cotes Integrationsalgorithmen

Wir beschränken uns hier auf Integrationsalgorithmen, die sich auf Sätze von Datenpunkten $\{(x_i, f_i)\}$ mit äquidistanten x_i-Werten beziehen, also $x_0(= a), \ldots, x_n(= b)$ mit $x_i = x_0 + ih$. Die grundlegende Idee ist, dass durch die Punkte $\{(x_i, f_i)\}$ stückweise Polynome gelegt werden, die dann über den x-Bereich der Datenpunkte integriert werden. Der dadurch ermittelte Wert ist ein Näherungswert für das Integral über die durch die Datenpunkte dargestellte Funktion.

Je nachdem, ob man die Funktion $f(x)$ zwischen den Stützstellen durch Polynome ersten, zweiten oder dritten Grades ersetzt, erhält man folgende Ausdrücke

$$\int_a^b dx\ f(x) = \frac{h}{2}\left(f_0 + 2f_1 + 2f_2 + \cdots 2f_{n-1} + f_n\right) - \frac{b-a}{12}h^2 f''(\xi), \tag{2.35}$$

$$\int_a^b dx\ f(x) = \frac{h}{3}\left(f_0 + 4f_1 + 2f_2 + 4f_3 \cdots 2f_{n-2} + 4f_{n-1} + f_n\right) - \frac{b-a}{180}h^4 f^{IV}(\xi), \tag{2.36}$$

$$\int_a^b dx\ f(x) = \frac{3h}{8}\left(f_0 + 3f_1 + 3f_2 + 2f_3 + 3f_4 + 3f_5 \cdots 3f_{n-1} + f_n\right) - \frac{b-a}{80}h^4 f^{IV}(\xi). \tag{2.37}$$

Die jeweils letzten Terme in den obigen Gleichungen erlauben eine Fehlerabschätzung mit $\xi \in [a, b]$.

Die erste Integrationsregel wird in der Literatur *Trapezregel* genannt, die Zahl der Stützstellen n ist beliebig. Bei der zweiten Regel (*Simpson-Regel*) muss n durch zwei teilbar sein, bei der dritten Regel (*Simpson 3/8 Regel*) muss n durch drei teilbar sein.

2.3.2 Gauss Integration

Gleichungen 2.35–2.37 zeigen, dass sich bei der Wahl äquidistanter Stützstellen die Näherungsformeln für die Integrale durch Summen über die Funktionswerte an diesen Stützstellen multipliziert mit geeigneten Vorfaktoren (Gewichtsfaktoren) schreiben lassen. Die Gauss Integration erweitert diese Idee folgendermaßen: geht man von den äquidistanten Stützstellen ab, lässt man also die Wahl der n Stützstellen offen, so gewinnt man weitere n freie Parameter, die man im Sinne einer Optimierung der Näherungsformeln wählt. So kann ein Polynom bis zum Grad $(2n-1)$ durch die zu integrierende Funktion gelegt werden. Die Näherungsformeln, die auf dieser Idee basieren, werden Gauss-Integrationsformeln genannt. Sie sollten allerdings nur dann verwendet werden, wenn die zu integrierende Funktion $f(x)$ explizit bekannt ist.

Das Prinzip der Gauss Integration wird im folgenden für $n = 2$ veranschaulicht. Gegeben sei das Integral $\int_a^b dx\ f(x)$, wobei a und b endliche Werte haben sollen. Nach

der Variablensubstitution

$$x = \frac{b-a}{2}t + \frac{b+a}{2}, \quad dx = \frac{b-a}{2}\,dt \tag{2.38}$$

erfolgt die Integration über das symmetrische Intervall $[-1, +1]$. Der Ansatz von Gauss lautet damit

$$\int_a^b dx\, f(x) = \frac{b-a}{2} \cdot \int_{-1}^{1} dt\, f(t) = \frac{b-a}{2} \cdot [a_1 f(t_1) + a_2 f(t_2)] \tag{2.39}$$

mit unbekannten Gewichtsfaktoren a_1 und a_2 und unbekannten Stützstellen t_1 und t_2. Da diese Formel für Polynome bis zum Grad $(2n-1) = 3$ gelten muss, können diese vier unbekannten Parameter aus folgenden Gleichungen bestimmt werden:

$$\begin{aligned}
f(t) = t^3 \quad &: \quad \int_{-1}^{1} dt\, t^3 = 0 = a_1 t_1^3 + a_2 t_2^3, \\
f(t) = t^2 \quad &: \quad \int_{-1}^{1} dt\, t^2 = 2/3 = a_1 t_1^2 + a_2 t_2^2, \\
f(t) = t^1 \quad &: \quad \int_{-1}^{1} dt\, t = 0 = a_1 t_1 + a_2 t_2, \\
f(t) = 1 \quad &: \quad \int_{-1}^{1} dt = 2 = a_1 + a_2.
\end{aligned} \tag{2.40}$$

Lösung dieser vier Gleichungen führt auf

$$a_1 = a_2 = 1 \qquad t_2 = -t_1 = \sqrt{\frac{1}{3}} = 0.5773 \tag{2.41}$$

und somit

$$\int_{-1}^{1} dt\ f(t) = f(-0.5773) + f(0.5773). \tag{2.42}$$

Ist $f(t)$ ein beliebiges Polynom dritten Grades, so ist diese Formel – konstruktionsgemäß – exakt.

Die Formeln für die Gauss-Integration lassen sich für beliebiges n erweitern: sie lauten dann (beachten Sie im Unterschied zu den vorhergehenden Abschnitten den veränderten Startindex $i = 1$)

$$\int_{-1}^{1} dt f(t) = \sum_{i=1}^{n} w_i f(t_i), \tag{2.43}$$

wobei die $2n$ Unbekannten w_i und t_i, $i = 1, \ldots, n$, aus den Gleichungen

$$\sum_{i=1}^{n} = \begin{cases} 0 & k = 1, 3, \ldots, 2n-1 \\ \frac{2}{k+1} & k = 0, 2, \ldots, 2n-2 \end{cases} \tag{2.44}$$

bestimmt werden. Es zeigt sich, dass die gesuchten t_i, die Nullstellen des Legendre-Polynoms $P_n(t)$ vom Grad n sind und dass sich die w_i mithilfe von

$$w_i = [P_{n-1}(t_i) \cdot P'_n(t_i)]^{-1} \tag{2.45}$$

berechnen lassen. Die Legendre-Polynome werden mithilfe der rekursiven Relation

$$(n+1)P_{n+1}(t) - (2n+1)\,t\,P_n(t) + nP_{n-1}(t) = 0 \tag{2.46}$$

mit

$$P_0(t) = 1, \quad \text{und} \quad P_1(t) = t. \tag{2.47}$$

bestimmt. Die Legendre-Polynome sind auf dem Intervall $[-1, 1]$ orthogonal, d.h. es gilt

$$\int_{-1}^{1} P_n(t)P_m(t)dt = \begin{cases} 0 & n \neq m \\ > 0 & n = m \end{cases} . \tag{2.48}$$

Konkret ergibt sich $P_2(t)$ aus (2.46) und (2.47) zu

$$P_2(t) = \frac{3}{2}t^2 - \frac{1}{2} \tag{2.49}$$

und

$$w_i = [P_1(t_i) \cdot P'_2(t_i)]^{-1} = [t_i \cdot 3t_i]^{-1} . \tag{2.50}$$

Aus Gl. 2.49 sieht man leicht, dass die Nullstellen von $P_2(t)$ tatsächlich durch $t_i = \pm\sqrt{1/3}$ gegeben sind. Setzt man diese Werte in Gl. 2.50 ein, so erhält man $w(t_i) = 1$ (vgl. Gl. 2.41).

In der nachfolgenden Tabelle sind die Legendre-Polynome vom Grad $n \leq 4$, ihre jeweiligen Nullstellen t_i und die Werte der Gewichtsfaktoren, $w(t_i)$, angegeben.

n	$L_n(t)$	Nullstellen t_i	Gewichtsfaktoren $w(t_i)$
0	1	–	–
1	t	0.0	1
2	$\frac{1}{2}(3t^2-1)$	-0.57735	1
		0.57735	1
3	$\frac{1}{2}(5t^3-3t)$	-0.77459	0.55555
		0.0	0.88888
		0.77459	0.55555
4	$\frac{1}{8}(35t^4-30t^2+3)$	-0.86113	0.34785
		-0.33998	0.65214
		0.33998	0.65214
		0.86113	0.34785

Ganz allgemein kann die Gauss Integration auf Integrale vom Typ $\int_a^b dx\, W(x)f(x)$ erweitert werden. Die Näherungsformel lautet dann

$$\int_a^b dx\, W(x) f(x) \approx \sum_{i=1}^{n} w_i f(x_i). \tag{2.51}$$

Dieser Ausdruck ist exakt, wenn $f(x)$ ein Polynom ist, dessen Grad kleiner oder gleich $(2n-1)$ ist. $W(x)$ wird Gewichtsfunktion genannt.

Die Berechnung der Gewichte w_i erfolgt mit Hilfe der zu $W(x)$ assoziierten orthogonalen Polynome $p_i(x)$. Ist $W(x)$ und ein Intervall $\mathcal{I} = [a, b]$ gegeben, so heißen Polynome $p_i(x)$, $i = 0, 1, 2 \ldots$, orthonormiert bezüglich $W(x)$ auf $\mathcal{I}$, wenn gilt

$$\int_a^b dx\; p_i(x) p_j(x) W(x) = \delta_{ij}. \tag{2.52}$$

Die Abszissen x_i sind die Nullstellen von $p_n(x)$, die Gewichte sind gegeben durch

$$w_i = \left[p_{n-1}(x_i) p_n'(x_i) \right]^{-1}. \tag{2.53}$$

Die Gauss-Integration erfolgt also bei vorgegebener Gewichtsfunktion $W(x)$ und Intervall $\mathcal{I}$ in zwei Stufen:

1. Bestimmung der zu $W(x)$ assoziierten orthonormalen Polynome $p_i(x)$;
2. Wahl von n, Bestimmung der Nullstellen von $p_n(x)$ und Berechnung der Gewichte w_i.

Abschließend wird die Summe Gl. 2.51 berechnet.

Für spezielle Gewichtsfunktionen $W(x)$ sind die orthogonalen Polynome bekannt, z.B.

$W(x)$	$\mathcal{I}$	Polynom		
1	$[-1, 1]$	$P_n(x)$	...	Legendre-Polynome
e^{-x}	$[0, \infty]$	$L_n(x)$	...	Laguerre-Polynome
e^{-x^2}	$[-\infty, \infty]$	$H_n(x)$	...	Hermite-Polynome
$x^\alpha,\ \alpha > -1$	$[-1, 1]$	$P_{2n}^{\alpha,0}$	...	Jakobi-Polynome

Kapitel 3

Zufallszahlen

Die moderne Physik hat ein grundlegendes Überdenken elementarer Stützen der klassischen Philosophie bewirkt, insbesondere des Kausalitätsprinzips. Als Physiker akzeptieren wir gewisse quantenmechanische Phänomene als statistisch im Sinne der prinzipiellen Unvorhersagbarkeit des Einzelereignisses. Ein Beispiel ist die Ungewißheit des genauen Zeitpunkts der Relaxation eines angeregten Atoms oder eines radioaktiven Zerfallsakts, die sich im Experiment mit einem einfachen Geigerzähler anschaulich zeigen lässt: die Zeitabstände aufeinanderfolgender *ticks* sind im Einzelfall prinzipiell unvorhersehbar und zufällig, sie gehorchen aber statistischen Gesetzmäßigkeiten. Wir definieren **wahre Zufallszahlen** als Zahlenfolgen, deren Einzelwerte im physikalischen Sinn prinzipiell unvorhersehbar sind, deren Häufigkeitsverteilung und statistische Erwartungswerte aber physikalischen Gesetzmäßigkeiten folgen, oder aus mathematischer Sicht: „. . . ein k-Tupel von Zahlen, das mit der statistischen Hypothese in Einklang steht, eine Realisierung eines zufälliges Vektors mit unabhängigen, identisch nach einer Verteilungsfunktion v verteilten Komponenten zu sein, nennt man ein k-Tupel von nach v verteilten Zufallszahlen.“(Überhuber).
„Zufälle nennt man in der Natur, was beim Menschen Freiheit heißen würde.“ (Goethe, 1811)

Viele Größen, die landläufig als statistisch angesehen werden, lassen sich auf prinzipiell determinierte Vorgänge zurückführen, die jedoch entweder aus praktischen Gründen nicht im Detail einzeln verfolgt werden oder aber grundsätzlich unzugänglich sind, wie z.B. in chaotische Vorgängen, wo die Anfangsbedingungen einer Zustandsänderung nicht vollständig und ausreichend genau bestimmbar sind. Beispiele sind viele Meßfehler oder die Trajektorien eines Einzelmoleküls im Gas. Wenn daraus resultierende Zahlenfolgen den Gesetzmäßigkeiten der Folge wahrer Zufallszahlen gehorchen, heißen sie (experimentelle) Pseudozufallszahlen. Analog sind (algorithmische) Pseudozufallszahlen Zahlenfolgen, die aus sorgfältig entwickelten Algorithmen gewonnen werden. Algorithmen für Pseudozufallszahlen und ihre Anwendung sind Gegenstand der folgenden Ausführungen [5].

3.1 Generierung von Zufallszahlen

3.1.1 Generierung „wahrer“ Zufallszahlen

Manche Computer sind mit einem physikalischen Zufallszahlengenerator ausgestattet, der aus dem Rauschen einer Widerstandsmessung oder mithilfe einer radioaktiven Quelle eine zufällige Bitfolge bestimmen. Leichter zugänglich ist die Aufnahmefunktion vieler Computer, die in Abwesenheit eines Eingangssignals ebenfalls ein zufälliges Signal produziert, das in eine Sequenz echter Zufallszahlen umgewandelt werden kann.

3.1.2 Pseudozufallszahlen

Lineare Kongruenz Methode nach D.H. Lehmer, 1949 (Linear Congruential Generator). Sie liegt den meisten Routineanwendungen zu Grunde und ist Basis vieler Bibliotheksfunktionen. Ihr Kern ist die Rekursionsvorschrift

$$a_{i+1} = (a_i \cdot b + c) \,\mathrm{mod} m\,, \tag{3.1}$$

die eine Folge von natürlichen Pseudozufallszahlen $a_1, a_2, a_3, \ldots$ liefert.

Die Parameter b, c sind **geeignet gewählte** natürliche Zahlen. Der Startwert a_0 ist eine beliebige natürliche Zahl im Bereich $[0, m-1]$. Oft wird $c = 0$ (bei passenden Werten für b und m) verwendet.

Beispiel: $m = 32$, $b = 7$, $c = 1$, $a_0 = 0$ (das sind einfache Demo-Parameter. Empfohlene Parameter für reale Anwendungen: siehe Tabelle 3.1):

$$\begin{aligned} a_1 &= (07+1) \mod 32 &&= 1\%32 = 1 \\ a_2 &= (17+1) \mod 32 &&= 8\%32 = 8 \\ a_3 &= (87+1) \mod 32 &&= 57\%32 = 25... \end{aligned} \tag{3.2}$$

Wie man sieht, liegen die so gewonnenen Zufallszahlen im Bereich $(0, m-1)$. Will man große Zufallszahlen erzeugen, muß m groß sein, was nur bei entsprechend großen Werten von $(a_i b + c)$ möglich ist. Hier besteht das Problem darin, dass dieser Zwischenwert maschinenintern durch die größte darstellbare Integerzahl begrenzt ist und die Gefahr eines Speicherüberlaufs besteht (overflow). Dieser ist schwer erkennbar, weil keine Fehlermeldungen oder Warnungen erzeugt werden und mit den niedrigstelligen (least significant) Bits weitergerechnet wird. Abhilfe durch geeignete Algorithmen ist möglich.

Die von der linearen Kongruenzmethode gelieferten Zahlenfolgen sind grundsätzlich periodisch. Bei schlechter Parameterwahl ist die Periode sehr kurz und das Verfahren mit diesen Parametern unbrauchbar, z.B. $a_0 = 0, b = 19, c = 1, m = 381 \Longrightarrow 0, 1, 20, 0, 1, 20, ...$ Ein anderes Beispiel für eine schlechte Parameterwahl ist $a_0 = 10^8, a = 31415821, b = 1, m = 1234567$. Damit ergibt sich die Folge 35884508, 80001069, 63512650, 43635651, 1034472, 87181513, 6917174, 209855, 67115956, 59939877, ...,

overflow bei	m	b	c	overflow bei	m	b	c
2^{20}	6075	106	1283	2^{27}	121500	1021	25673
2^{22}	7875	211	1663	2^{27}	259200	421	54773
2^{22}	7875	421	1663				
2^{23}	6075	1366	1283	2^{28}	117128	1277	24749
2^{23}	6655	936	1399	2^{28}	121500	2041	25673
2^{23}	11979	430	2531	2^{28}	312500	741	66037
2^{24}	14406	967	3041	2^{29}	145800	3661	30809
2^{24}	29282	419	6173	2^{29}	175000	2661	36979
2^{24}	53125	171	11213	2^{29}	233280	1861	49297
				2^{29}	244944	1597	51749
2^{25}	12960	1741	2731				
2^{25}	14000	1541	2957	2^{30}	139968	3877	29573
2^{25}	21870	1291	4621	2^{30}	214326	3613	45289
2^{25}	31104	625	6571	2^{30}	714025	1366	150889
2^{25}	139968	205	29573				
				2^{31}	134456	8121	28411
2^{26}	29282	1255	6173	2^{31}	259200	7141	54773
2^{26}	81000	421	17117				
2^{26}	134456	281	28411	2^{32}	233280	9301	49297
				2^{32}	714025	4096	150889
2^{27}	86436	1093	18257				

Tabelle 3.1: Parameter für die lineare Kongruenzmethode (aus: Numerical Recipies)

bei der die letzte Stelle jeweils um 1 wächst. Die Ermittlung guter Parametersätze ist aufwändig, und es sollten nur erprobte Algorithmen und Parameterkombinationen eingesetzt werden.

3.1.3 Andere Methoden

- Kombinationsmethoden verwenden Kombinationen von Zufallszahlen verschiedener Generatoren. Mitunter können damit sehr große Periodenlängen erzielt werden. Zum unten angeführten *non-linear additive feedback random number generator* siehe die Dokumentation in Linux. Ausführliche Beschreibungen von Algorithmen finden sich u.a. in [5, 6].

- Fibonacci-Generator: $a_{n+1} = (a_n + a_{n-1}) \mod m$

- von Neumann Generator („Quadratmittengenerator“): Man quadriert eine Zahl, schneidet aus der Mitte der Ziffernfolge eine Zahl heraus, die man wieder quadriert, usw.).

3.1.4 Bibliotheksfunktionen

In ANSI-C werden die Funktionen `int rand()` [in Linux/GNU oft `long rand()`] und `srand()` verwendet (`include math.h`). Sie beruhen häufig auf der linearen Kongruenzmethode. Der maximale Wert der darstellbaren Zufallszahl ist durch die systemdefinierte Makro-Variable `RAND_MAX` gegeben; sie ist oft gleich `INT_MAX` (oder `LONG_MAX`, für `long rand()`). Die ANSI-Norm verlangt ein `RAND_MAX` von lediglich 32767, was für technisch-wissenschaftliche Anwendungen völlig unzureichend ist. Wird die lineare Kongruenzmethode verwendet, entspricht `RAND_MAX` dem Parameter m. Die größte erreichbare Zufallszahl ist daher `RAND_MAX-1`, die kleinste Null.

Die oben erwähnte Unzulänglichkeit des Mindeststandards in der ANSI-Norm hat in der Unix-/Linux-Welt zur Implementierung von alternativen Funktionen (z.B. dem *non-linear additive feedback random number generator*) geführt. Statt `ran()` und `srand()` werden `random()` und `srandom()` eingesetzt (relativ große Periode von $(16 * (2^{31} - 1))$, sowie die Hilfsfunktionen `initstate()` und `setstate()`. Hier besteht in der Qualität der Zufallszahlen kein Unterschied zwischen low bytes und high bytes.

In Fortran 77 sieht der Standard keinen intrinsischen Zufallszahlengenerator vor (der Fortran2003-Standard definiert die Routinen `RAND, IRAND` und `RANDOM_NUMBER` samt der dazugehörigen Initialisierungsroutinen `SRAND` bzw. `RANDOM_SEED`). Allerdings gibt es eine Vielzahl frei erhältlicher Bibliotheken mit Zufallszahlengeneratoren. Unter Linux sind - compilerabhängig - für Fortran 77 oft die gleichen Generatoren wie unter C verfügbar. Allerdings sind für komplexere Anwendungen, z.B. Monte-Carlo Methoden, mitunter auch gute Standard-Bibliotheksfunktionen nicht ausreichend, und eine sorgfältige Vorarbeit ist unabhängig von der gewählten Sprache unerlässlich.

Anfangswerte

Die Funktion `srand(int/long Anfangswert)` ist die seed-Funktion, die im Rahmen der Initialisierungen (Programmstart) den Anfangswert der Zufallsfolge festlegt, also das a_0 in der oben verwendeten Bezeichnungsweise. **Vorsicht:** Wenn im Code keine Vorkehrungen getroffen werden, läuft ein Programm immer mit der gleichen Zufallszahlenfolge ab. Dieses Verhalten kann zwar während der Programmentwicklung wünschenswert sein, wird aber ein zufälliger Anfangswert gewünscht, bietet sich z.B. die Funktion `time(NULL)` an, die als Rückgabewert die seit dem 1.1.1970, 0.00 Uhr ('Unix-Urknall') vergangene Sekundenanzahl liefert. `srand()` setzt eine globale Systemvariable und wirkt daher auf alle `rand()`-Aufrufe im Programm unabhängig von ihrer Position im Code.

3.2 Häufigkeitsverteilungen

Die meisten Standardgeneratoren (und Bibliotheksfunktionen) liefern gleichverteilte Zufallszahlen, d.h. jede Integerzahl (oder jedes beliebige Sub-Intervall bei floating point Zahlen) tritt mit gleicher Wahrscheinlichkeit und damit näherungsweise gleich häufig auf. Manchmal will man aber ungleichverteilte Zahlenfolgen generieren, deren Werte

gemäß einer bestimmten Häufigkeitsverteilung auftreten, z.B. einer Gaußverteilung, Exponentialverteilung, logarithmischen Verteilung, dem Planck'schen Strahlungsgesetz, einer eigenen Vorschrift folgend, usw.

3.2.1 Zurückweisungsmethode

Diese Methode ist allgemein anwendbar und leicht zu implementieren, mitunter aber ineffizient. Die vorgegebene Verteilungsfunktion $y = v(x)$ sei im Intervall $A \leq x \leq B$ anzuwenden und habe darin einen maximalen Funktionswert y_{max}. Nun werden zwei unabhängige, gleichverteilte Zufallszahlen R_1 und R_2 im Intervall $(0, R_{max})$ generiert.

- Die erste wird auf das Intervall (A, B) abgebildet und wählt darin einen zufälligen x-Wert aus: $x_{R1} = A + R_1(B - A)/(R_{max})$. Nun wird der Funktionswert an dieser Stelle bestimmt: $v(x_{R_1})$.

- Dann wird R_2 auf den Bereich der Funktionswerte von $v(x)$, also $(0, y_{max})$ abgebildet und geprüft, ob $R_2/R_{max} < v(x_{R_1})/y_{max}$ ist. Bildlich entspricht das der Frage, ob der (auf y_{max} normierte) Zufallswert unterhalb des Funktionsgraphen, also unter $v(x_{R_1})$ liegt oder darüber.

- Gegebenenfalls wird R_1 als gültiger Wert akzeptiert. Andernfalls, also wenn $R_2/R_{max} > v(x_{R_1})/y_{max}$ ist, wird der gesamte Schritt verworfen.

Es führt also nicht jeder Schritt zum Erfolg, insbesondere ist die Methode dann äußerst ineffizient, wenn $v(x)$ nur eine lokale Spitze hat und überall sonst sehr klein ist, weil dann fast jeder Schritt verworfen werden muss. Eine mögliche Verbesserung für solche Fälle besteht darin, eine Funktion $V(x)$ zu suchen, deren Werte im Intervall $A \leq x \leq B$ leicht berechenbar sind und die eine möglichst eng anliegende Hülle um $v(x)$ bildet: an jeder Stelle muß aber $V(x) \geq v(x)$ sein. An die Stelle des Vergleichs $R_2/R_{max} < v(x_{R_1})/y_{max}$ tritt der Vergleich $R_2/R_{max} < v(x_{R_1})/V(x_{R_1})$. Die Funktion $V(x)$ kann auch stückweise zusammengesetzt werden und braucht lediglich eindeutig berechenbar zu sein; sie muss insbesondere nicht stetig sein.

3.2.2 Transformationsmethode

Hier wird versucht, jene Funktion $f(R)$ von Zufallszahlen R explizit zu finden, mit der die Werte $x_i = f(R_i)$ einer vorgegebenen Wahrscheinlichkeitsdichte (Häufigkeitsverteilung) $v(x)$ auftreten. Beispielsweise liefern gaußverteilte Zufallsfolgen mit

$$v(x) = \frac{1}{\sqrt{2\pi}} \exp\Big(- \frac{(x - x_0)^2}{2} \Big) \tag{3.3}$$

bevorzugt Werte x nahe x_0. Es ist $v(x)dx$ die Wahrscheinlichkeit für das Auftreten einer Zufallszahl im Intervall $(x, x + dx)$ und (Normierung der Gesamtwahrscheinlichkeit auf

1):

$$\int_{x=-\infty}^{x=+\infty} v(x)dx = \frac{1}{\sqrt{2\pi}} \int_{x=-\infty}^{x=+\infty} \exp\Big(-\frac{(x-x_0)^2}{2}\Big) = 1 \tag{3.4}$$

Analoges gilt für eine Wahrscheinlichkeitsdichte (Häufigkeitsverteilung) $Z(R)$ von gleichverteilten Zufallszahlen R, die von R unabhängig ist, für die also $Z(R) = const = 1$. Nach einer Transformation $Z(R) \to v(x)$ mit Hilfe von $x = f(R)$ muß jede Wahrscheinlichkeit $Z(R)dR$ einer Wahrscheinlichkeit $v(x)dx$ entsprechen, und es muss in jedem Intervall

$$\Big|Z(R)dR\Big| = \Big|v(x)dx\Big| \to v(x) = Z(R)\left|\frac{dR}{dx}\right| = f(R) \tag{3.5}$$

sein. Mit $Z(R) = 1$ wird

$$\frac{dR}{dx} = v(x) \to R = \int v(x)dx = W(x) \to x = W^{-1}(R) \tag{3.6}$$

Die praktische Anwendbarkeit der Transformationsmethode hängt davon ab, wie leicht $W^{-1}(R)$ berechenbar oder in einem Computeralgorithmus (eventuell näherungsweise) darstellbar ist.

3.2.3 Poissonverteilte Zufallszahlen und Exponentialverteilung

Die Poissonverteilung $P(n)$ beschreibt die Wahrscheinlichkeit für das Auftreten von n zufallsbedingten Ereignissen in einem dimensionslosen Intervall (interpretierbar als Zeit, Strecke, Raum, auch das Auftreten von x Fehlern in einer Produktionscharge, usw.). Sie hat einen einzigen Parameter, λ, der gleichzeitig Mittelwert und Standardabweichung darstellt und eine Rate (mittlere Anzahl von Ereignissen in einem Intervall) angibt:

$$P_n(n) = \frac{\lambda^n}{n!}e^{-\lambda} \tag{3.7}$$

n sind ganze Zahlen. Für große Werte (>10) von λ wird die Poissonverteilung näherungsweise zur Gaußverteilung mit $\sigma^2 = \lambda$. Ersetzt man λ für physikalische Anwendungen durch ein Produkt aus (Mittelwert/Einheitsintervall) × Intervallgröße, z.B. λ_t $[s^{-1}]$ (Ereignisse pro Zeiteinheit, Rate) × Zeitintervallgröße t [s], ist $\lambda_t t$ wieder eine dimensionslose Zahl und man erhält

$$P_x(x) = \frac{(\lambda_t t)^x}{x!}e^{-\lambda_t t} \tag{3.8}$$

Mit der Poissonverteilung eng verknüpft ist die Exponentialverteilung, welche die dimensionslose Intervalllänge (beispielsweise als Wartezeit interpretierbar) bis zum Eintreten des nächsten (ersten) Ereignisses angibt. Wählt man wieder Zeitintervalle, muss

man dazu die obige Gleichung nach der Zeit differenzieren und erhält

$$P_t(t) = \lambda_t e^{-\lambda_t t} \tag{3.9}$$

Der Erwartungswert und die Standardabweichung sind beide gleich $1/\lambda$ [s]. Für die Exponentialverteilung lassen sich das Integral $W(t)$ und die Umkehrfunktion W^{-1} leicht berechnen (Transformationsmethode):

$$W(t) = R = \int \lambda_t e^{-\lambda_t t} dt = 1 - e^{-\lambda_t t}$$
$$t = W^{-1}(1-R) = -\frac{1}{\lambda_t}\ln(1-R) \to -\frac{1}{\lambda_t}\ln R \tag{3.10}$$

Im letzten Schritt wurde die Tatsache verwendet, dass R und $1-R$ äquivalente gleichverteilte, auf 1 normierte Zufallszahlenfolgen sind. Gibt man eine mittlere Ereignisrate λ_t vor, z.B. 100 radioaktive Zerfälle pro Sekunde, bilden die Werte $t_i = -0.01 \cdot \ln(R_i)$ eine Folge von exponentialverteilten Wartezeiten, die aus gleichverteilten Zufallszahlen R_i berechnet werden kann.

3.2.4 Gaußverteilte Zufallzahlen

Gaußverteilte Zufallzahlen folgen der Verteilungsfunktion

$$v(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{3.11}$$

Es ist keine analytische Darstellung der in der Transformationsmethode definierten Funktionen W und W^{-1}bekannt, aber es gibt einige effiziente Algorithmen:

- **Box-Muller Algorithmus:** (ohne Beweis)
  - Generiere zwei gleichverteilte, auf 1 normierte Zufallszahlen R_1 und R_2.
  - Berechne $\omega = 2\pi R_1$ und $\varrho = -2\ln(R_2)$
  - Berechne $X_1 = \sqrt{\varrho}\cos\omega$ und $X_2 = \sqrt{\varrho}\sin\omega$

  X_1 und X_2 sind die gesuchten gaußverteilten Zufallszahlen. Es werden also Paare von gleichverteilten Zufallszahlen benötigt, um Paare von gaußverteilten Zufallszahlen zu erhalten.

- Polarmethode von Marsaglia: (ohne Beweis)
  - Generiere zwei gleichverteilte, auf 1 normierte Zufallszahlen R_1 und R_2.
  - Setze $S = 2R_1 - 1$ und $T = 2R_2 - 1$
  - Berechne $W^2 = S^2 + T^2 > 1$. Abbrechen/Neubeginn, wenn $W^2 > 1$
  - Berechne $X_1 = (S/W)\sqrt{(-2\ln(W^2))}$ und $X_2 = (T/W)\sqrt{(-2\ln(W^2))}$.

X_1 und X_2 sind die gesuchten gaußverteilten Zufallszahlen. Wiederum führt ein Paar von gleichverteilten Zufallszahlen zu einem Paar gaußverteilter Zufallszahlen. Da es sich um eine Version der Verwerfungsmethode handelt, ergibt nicht jede Kombination von Werten R_1 und R_2 ein Paar X_1 und X_2. Die Polarmethode ist gegenüber dem Box-Muller-Algorithmus vorteilhaft, wenn die Berechnung von Winkelfunktionen zu zeitaufwändig ist.

3.2.5 Auf benutzerdefinierte Intervalle beschränkte (ganzzahlige) Zufallszahlen

Die mit Bibliotheksfunktionen generierten, ganzzahligen Zufallszahlen liegen im Bereich von 0 und einem Maximalwert, der z.B. in C für `rand()` durch das Makro `RAND_MAX` gegeben ist. Will man den Bereich einschränken (etwa auf die Werte $(1, 2, 3, 4, 5, 6)$ zur Simulation eines Würfels), liegt eine Modulo-Division nahe: $1 + \texttt{rand}()\%m$ mit $m = 6$. Die Methode hat zwei Nachteile: erstens beruht das Ergebnis auf den least significant bits, was, wie erwähnt, bei manchen Generatoren unvorteilhaft ist. Zweitens ist im allgemeinen der interessierende Bereich (mit Startwert $0 : 0, 1, 2, 3, 4, 5$) nicht ganzzahlig in `RAND_MAX` enthalten. Es gibt also einige Zahlen, die ein Mal öfter vorkommen als andere, nämlich die zwischen $m * (\texttt{int})(\texttt{RAND\_MAX}/m)$ und `RAND_MAX`. Im vorliegenden Fall ist das nicht kritisch, weil `RAND_MAX`$/5$ sehr groß im Vergleich zum gewählten Bereich der Zufallszahlen ist. Ist aber entweder `RAND_MAX` klein oder der gewünschte Bereich groß, kann das Probleme ergeben. Man muss korrekterweise alle Zufallszahlen $R \geq m * (\texttt{int})(\texttt{RAND\_MAX}/m)$ verwerfen.

3.2.6 Beliebig normierte (Gleitkomma-) Zufallszahlen

Auf 1 normierte Zufallszahlen

Die ganzzahligen Zufallszahlen liegen im Bereich von 0 und `RAND_MAX`. Die Normierung auf 1 erfolgt durch `(double) rand()/(RAND_MAX+1.0)` oder `(double) rand()/(double)(RAND_MAX))` je nachdem, ob `RAND_MAX` die größtmögliche Zufallszahl miteinschließt oder nicht und ob im Ergebnis der Wert 1 inkludiert sein soll (letzteres meist nicht). Man beachte den Dezimalpunkt nach `1.`, der den Ausdruck zu einer `double` Größe macht. Weglassen erzeugt möglicherweise eine kleine Katastrophe, weil `RAND_MAX` oft gleich `LONG_MAX` ist und `RAND_MAX+1` dann durch overflow -2147483648 ergibt!

Andere Normierungen

Grundsätzlich gilt das oben Gesagte. Die Linux-Dokumentation empfiehlt für den non-linear additive feedback random number generator mit Bezug auf die Numerical recipies [6] als Beispiel für die Normierung auf 10 `j=1+(int) (10.0*rand()/(RAND_MAX+1.0));` Beachten Sie, daß sowohl der Token 1.0 als auch

der Token 10.0 die Berechnung des inneren Ausdrucks (Klammer) mit `double` Präzision bewirken. Das double-Zwischenergebnis wird in ein `int` konvertiert.

Kapitel 4

Nullstellenbestimmung und χ^2-Anpassung

4.1 Nullstellenbestimmung

Im vorangegangenen Kapitel wurde die Frage der numerischen Auswertung von Integralen behandelt. Dabei hat sich im Falle der Gauss-Quadratur

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i) \tag{4.1}$$

die Notwendigkeit der Bestimmung der Nullstellen der den Berechnungen zugrundeliegenden Polynome ergeben.

Wir werden uns daher diesem numerischen Problem zuwenden und die gängigsten Methoden vorstellen. Die Vorgangsweise wird in jedem Fall sehr ähnlich sein und kann wie folgt zusammengefasst werden:

1. finde einen Bereich, der (zumindest) eine Nullstelle enthält

2. wähle eine geeignete Methode der Nullstellenbestimmung.

4.1.1 Bracketing und Bisektionsverfahren

Die überwiegende Zahl von Nullstellen lässt sich durch den Vorzeichenwechsel der Funktion $f(x)$ an x_0 charakterisieren. Ausnahmen bilden komplexe Nullstellen (z.B. $x^2 + a = 0,\ a > 0$), doppelt zu zählende Nullstellen (z.B. $x^2 = 0$) oder Singularitäten der Form $f(x) = 1/g(x)$ mit ungerader Funktion $g(x)$. *Es gibt keine Methode, die für eine unbekannte Funktion f(x) mit Sicherheit eine Nullstelle finden könnte.* Es ist daher ratsam, sich vor Beginn des Programmierens mit Form und Art der zu untersuchenden Funktionen vertraut zu machen.

Den Vorzeichenwechsel von $f(x)$ macht man sich beim Bracketing zunutze, indem man, ausgehend von einem vorgegebenen Ausgangspunkt x_i, durch sukzessive Vergrößerung des Intervalls $[x_i, x_f]$ das erste x_f bestimmt, für das die Bedingung

$$f(x_i) \cdot f(x_f) \leq 0 \tag{4.2}$$

erfüllt ist. Anschließend kann man unter Wahrung der Bedingung 4.2 das Intervall wieder verkleinern und dadurch die (eine der) eingeschlossene(n) Nullstelle(n) auf beliebige Genauigkeit bestimmen (Bisektionsverfahren).

4.1.2 Newton-Raphson-Verfahren

Ist die analytische Form der ersten Ableitung von $f(x)$ bekannt, so empfiehlt sich die Verwendung des Newton-Raphson-Verfahrens. Es basiert auf der Taylorreihenentwicklung von $f(x)$ in der Nähe der Nullstelle $x_0 = x + \Delta x$

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{f''(x)}{2}\Delta x^2 + \dots . \tag{4.3}$$

Wegen $f(x + \Delta x) = 0$ folgt (unter Vernachlässigung aller Terme $O(\Delta x^2)$) $\Delta x = -f(x)/f'(x)$. Die Iterationsvorschrift für x_i lautet daher

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{4.4}$$

Das Newton-Raphson-Verfahren konvergiert quadratisch gegen die Nullstelle, d.h., dass sich mit jedem Iterationsschritt die Zahl der signifikanten Stellen verdoppelt. Ist die analytische Form der ersten Ableitung von $f(x)$ nicht bekannt und muss durch ein Differenzenverfahren bestimmt werden, verschlechtert sich die Effizienz dramatisch. In diesem Fall empfiehlt sich die Verwendung anderer Verfahren, die ohne Berechnung der Ableitung auskommen.

Nachteil des NR-Verfahrens: Da in Gleichung 4.3 alle Terme $O(\Delta x^2)$ vernachlässigt werden, kann es bei Startwerten, die weit von der Nullstelle entfernt sind, zu unvorhersagbaren Ergebnissen kommen. Liegen weiters Extrema von $f(x)$ zwischen dem Startwert x_i und der Nullstelle x_0 vor, so sind ebenfalls numerische Instabilitäten (wegen $f'(x) \to 0$) möglich.

4.1.3 Sekanten-Verfahren

Drei eng miteinander verwandte Verfahren begnügen sich mit der Auswertung der Funktion an zwei bzw. drei Werten für x, um eine iterative Annäherung an x_0 zu finden.

Das Sekanten-Verfahren und die *regula-falsi*-Methode nähern $f(x)$ in der Umgebung der Nullstelle durch die Sekante definiert durch die Funktionswerte an den Grenzen von $[x_{i-1}, x_i]$ an. Der Schnittpunkt der Sekante mit der x-Achse definiert den neuen

Näherungswert x_{i+1}, der entweder den Grenzwert x_{i-1} (Sekantenverfahren) ersetzt oder das Suchintervall so umdefiniert, dass Bedingung 4.2 erfüllt bleibt (*regula-falsi*).

Die Konvergenz von *regula-falsi* ist oft besser als linear, führt aber sicher zum Erfolg. Im Sekantenverfahren muss die Möglichkeit der Divergenz aufgrund des lokalen Verhaltens der Funktion $f(x)$ gegen die bessere Konvergenz ($\propto \epsilon^{1.618}$) abgewogen werden.

Eine interssante Variante der *regula-falsi*-Methode, die Sicherheit und Schnelligkeit ($\propto \epsilon^{\sqrt{2}}$) verbindet, ist die Nullstellenbestimmung nach Ridder. Zunächst wird f nicht nur an den Stellen x_{i-1} und x_i sondern auch in der Mitte des Intervalls $x_m = (x_{i-1} + x_i)/2$ berechnet.

Nach Abspalten eines Exponentialfaktors (für Details der Ableitung siehe z.B. [6]) erhält man für den neuen Wert x_{i+1}

$$x_{i+1} = x_m + (x_m - x_{i-1}) \frac{\operatorname{sgn}[f(x_{i-1}) - f(x_i)] f(x_m)}{\sqrt{f(x_m)^2 - f(x_{i-1}) f(x_i)}}. \tag{4.5}$$

4.1.4 Nullstellenbestimmung nach Brent

Garantierte Konvergenz bei gutem Konvergenzverhalten bietet das Verfahren nach Brent, das, grob gesprochen, die Kombination eines leicht veränderten Sekantenverfahrens und eingeschobener Bisektionsschritte darstellt. Ausgehend von den drei Zahlenpaaren $[x_a, f(x_a)]$, $[x_b, f(x_b)]$ und $[x_c, f(x_c)]$, wobei x_b die bisher beste Annäherung an x_0 sei, errechnet sich der neue Näherungswert als

$$x = x_b + \frac{S[T(R-T)(x_c - x_b) - (1-R)(x_b - x_a)]}{(T-1)(R-1)(S-1)}. \tag{4.6}$$

Die Hilfsgrößen S, R und T sind durch

$$S = f(x_b)/f(x_a), \quad R = f(x_b)/f(x_c), \quad T = f(x_a)/f(x_c) \tag{4.7}$$

bestimmt. Ist man mit dem Iterationsergebnis nicht zufrieden, ersetzt man es durch das Resultat eines Bisektionsschrittes.

4.1.5 Nullstellen von Polynomen

Bekannterweise haben Polynome vom Grad n ebensoviele Nullstellen und können als Produkt

$$p_n(x) = (x - x_1)(x - x_2) \dots (x - x_n), \quad x_n \in \tag{4.8}$$

angeschrieben werden. Da komplexe Nullstellen paarweise auftreten ($x_0 = a \pm ib$), ist leicht einsichtig, dass Polynome ungeraden Grades zumindest eine reelle Nullstelle aufweisen. Nachdem diese abgespalten wurde (Polynomdivision, siehe unten), muss die Nullstellensuche auf die komplexe Zahlenebene ausgedehnt werden.

Bei *Mullers Methode* handelt es sich um eine komplexe Verallgemeinerung der Sekantenmethode, wobei aber eine quadratische statt der linearen Interpolation angewandt wird. Ausgehend von drei x-Werten (können äquidistant auf der reellen Achse liegen) kommt folgende Iterationsformel zur Anwendung:

$$x_{i+1} = x_i - (x_i - x_{i-1}) \frac{2C}{B \pm \sqrt{B^2 - 4AC}}, \tag{4.9}$$

wobei die Hilfsgrössen q, A, B und C durch

$$\begin{aligned} q &= \frac{x_i - x_{i-1}}{x_{i-1} - x_{i-2}} \\ A &= qP_n(x_i) - q(1+q)P_n(x_{i-1}) + q^2 P_n(x_{i-2}) \\ B &= (2q+1)P_n(x_i) - (1+q)^2 P_n(x_{i-1}) + q^2 P_n(x_{i-2}) \\ C &= (1+q)P_n(x_i) \end{aligned} \tag{4.10}$$

gegeben sind. Das Vorzeichen in Gleichung 4.9 wird so gewählt, dass der Betrag des Nenners maximal wird.

Ein weiteres, von *Laguerre* erdachtes Verfahren beruht auf einer gewagten Abschätzung für den Abstand der tatsächlichen Nullstellen vom derzeit untersuchten Punkt x. Zunächst definiert man wiederum zwei Hilfsgrössen

$$\frac{d \ln |p_n(x)|}{dx} = \sum_{i=1}^{n} \frac{1}{x - x_i} = \frac{p_n'}{p_n} = G \tag{4.11}$$

$$-\frac{d^2 \ln |p_n(x)|}{dx^2} = \sum_{i=1}^{n} \frac{1}{(x - x_i)^2} = \left(\frac{p_n'}{p_n}\right)^2 - \frac{p_n''}{p_n} = H \ . \tag{4.12}$$

Unter der Annahme, dass die Nullstelle x_1 einen Abstand a, alle anderen Nullstellen aber Abstand b von x haben, können die Ausdrücke für G und H wesentlich vereinfacht werden

$$\frac{1}{a} + \frac{n-1}{b} = G \tag{4.13}$$

$$\frac{1}{a^2} + \frac{n-1}{b^2} = H \tag{4.14}$$

Damit ergibt sich für a

$$a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}} \tag{4.15}$$

was zum neuen Iterationswert $x_{i+1} = x_i - a$ führt. Auch in Gl. 4.15 wird der Nenner maximiert. Ähnlich wie das Bisektionsverfahren konvergiert das Laguerre-Verfahren immer zu einer der reellen Nullstellen. Für komplexe Nullstellen ist ein Misserfolg sehr unwahrscheinlich.

Auffinden mehrerer/aller Nullstellen von Polynomen

Eine einfache Methode, die für reelle Nullstellen funktioniert, besteht darin, dass man das zu untersuchende Intervall in n Unterintervalle teilt und diese auf einen Vorzeichenwechsel der Funktion innerhalb deren Grenzen untersucht. Danach werden durch eine der oben beschriebenen Methoden in den gefundenen Unterintervallen die Nullstellen von $f(x)$ bestimmt.

Aus Polynomen können Nullstellen auch abdividiert werden. Dadurch wird ein Polynom geringeren Grades erzeugt und die Prozedur wiederholt. Im Falle eines komplexen Paares von Nullstellen dividiert man durch

$$[x - (a + ib)][x - (a - ib)] = x^2 - 2ax + (a^2 + b^2). \tag{4.16}$$

Dabei ist zu beachten, dass die gefundenen Nullstellen nur als gute Näherung an die tatsächlichen Nullstellen des Polynoms betrachtet werden sollten, da sich numerische Fehler mit jeder Division aufaddieren. Alle gefundenen Nullstellen sollten daher nach Beendigung der Routine „aufpoliert“ werden.

- Einschub: einfache algebraische Manipulation von Polynomen:

  Für ein Polynom vom Grad n, das durch $p_n = \sum_{i=0}^{n} c_i x^i$ gegeben ist, führt man eine Multiplikation mit dem Faktor $(x - a)$ durch einen Programmteil durch, der

  1. c_n zu c_{n+1} macht und
  2. eine Zeile enthält, die den neuen Koeffizienten c_j durch $c_{j-1} - c_j * a$ berechnet (von $j = n$ beginnend).

  Will man das Polynom p_n durch $(x - a)$ dividieren, sind folgende Rechenschritte erforderlich:

  1. c_n in einem Zwischenspeicher $ZS1$ ablegen und $c_n = 0$ setzen (der Grad des Polynoms wird um eins erniedrigt)
  2. von $j = n-1$ beginnend c_j in $ZS2$ ablegen und auf $c_j = ZS1$ setzen. Danach wird $ZS1$ durch $ZS1 = ZS2 + ZS1 * a$ neu berechnet.

Der Vollständigkeit halber sei noch die Möglichkeit erwähnt, die Nullstellen eines Polynoms durch die Berechnung der Eigenwerte einer Matrix $\mathbf{A}$ zu bestimmen. Sie ist definiert durch die Gleichung

$$P_n(x) = \det[\mathbf{A} - x\ ] \tag{4.17}$$

Für Polynome hohen Grades und auf für lineare Algebra optimierten Computern kann sich dies als die schnellste und stabilste Methode herausstellen.

4.2 χ^2-Anpassung

Bevor wir zur Beschreibung der Anpassungsmethoden kommen, sollten ein paar Worte darüber gesagt werden, worum es bei einer Anpassung eigentlich geht. Es ist nicht das Ziel einer Fitprozedur, durch eine Reihe von Datenpunkten eine möglichst glatte Kurve zu legen, um die graphische Darstellung zu optimieren, sondern es handelt sich bei der χ^2-Anpassung um eine Methode zu entscheiden, wie gut das physikalische Bild, das man sich von einem Prozess gemacht hat, die gemessene Realität erklären kann. *Eine Fitprozedur alleine kann Ihnen niemals eine Erklärung für die gemessenen Daten liefern.* Zwei Beispiele:

- Sie untersuchen den radioaktiven Zerfall einer Ihnen unbekannten Substanz. Da Sie wissen, dass Zerfallsprozesse durch eine abklingende Exponentialkurve beschrieben werden können, passen Sie eine Funktion der Form $c \cdot e^{-\Gamma t}$ an Ihre Datenpunkte an, ermitteln daraus z.B. die Zerfallsrate des Materials und bestimmen mithilfe einer Datensammlung das untersuchte Element.

- Sie haben eine Experiment durchgeführt, für dessen Ergebnis Ihnen zwei mögliche Erklärungen einfallen. Ein Fit kann Ihnen die Wahl zwischen den beiden Modellen erleichtern, kann Ihnen aber auch sagen, dass keines der beiden Modelle in der Lage ist, die gemessenen Daten zufriedenstellend zu erklären.

Hat man also einen Satz von N Datenpunkten (x_i, y_i) gemessen und hat man die Vermutung, dass die Daten durch eine Funktion mit M Fitparametern

$$y(x) = y(x; a_1 \dots a_M) \tag{4.18}$$

beschrieben werden können, so sollte die Fitprozedur 1. die Parameter a_i, 2. eine Abschätzung über den Fehler jedes einzelnen Parameters und 3. ein statistisches Maß für die Qualität der Fitfunktion liefern. Geben Sie sich niemals mit dem ersten Punkt alleine zufrieden, oftmals täuscht die graphische Darstellung das Auge über die Unzulänglichkeit des zugrundegelegten Modells hinweg (eine geschickt gewählte Darstellung der Daten hilft oft zu verschleiern, dass man keine Ahnung hat, was man gemessen oder gerechnet hat!). Selbstverständlich sollte die Anzahl der Fitparameter möglichst gering sein, um den Wert eines Modells abschätzen zu können. Der leider oftmals begangene Irrweg, ein fehlerhaftes Modell durch zusätzliche Parameter a_i zu retten, sollten Sie tunlichst vermeiden.

Definiert man als Maß für die Abweichung

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - y(x_i; a_1 \dots a_M)}{\sigma_i} \right)^2 \tag{4.19}$$

wobei die Parameter σ_i in Gl. 4.19 die individuelle Standardabweichungen zu jedem Messpunkt (x_i, y_i) bedeuten, so besteht die Aufgabe in der Minimierung von χ^2 bezüglich der Fitparameter a_i.

Kennt man σ_i nicht und muss/kann man von einer für alle Datenpunkte identischen Standardabweichung ausgehen, setzt man zunächst alle σ_i auf eins, minimiert Gl. 4.19 und errechnet aus dem Ergebnis σ mittels

$$\sigma^2 = \frac{\sum_{i=1}^{N}[y_i - y(x_i; a_1 \dots a_M)]^2}{N - M}. \tag{4.20}$$

Numerisch gesehen geht es bei einer χ^2-Anpassung darum, die Nullstellen der M Ableitungen von Gl. 4.19 nach den Fitparametern a_k zu finden, also das Gleichungssystem

$$0 = \sum_{i=1}^{N} \left(\frac{y_i - y(x_i)}{\sigma_i} \right) \left(\frac{\partial y(x_i; \dots a_k \dots)}{\partial a_k} \right) \qquad k = 1, \dots, M \tag{4.21}$$

zu lösen.

Wir wollen uns hier auf Modelle beschränken, die linear in den Fitparametern a_i sind, also Summen von Produkten der a_i mit beliebigen Basisfunktionen X_i. Handelt es sich bei den X_i z.B. um Potenzen von x, fitten wir unsere Daten an ein Polynom an, bei $X_i = \sin(\alpha_i x)$ oder $X_i = \cos(\alpha_i x)$ an eine harmonische Serie. Damit sieht unsere ursprüngliche Gl. 4.19 nun folgendermaßen aus:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - \sum_{k=1}^{M} a_k X_k(x_i)}{\sigma_i} \right)^2. \tag{4.22}$$

Definiert man das Problem durch Definition einer N-Zeilen $\times M$-Spalten Matrix $\mathbf{A}$ und eines Vektors $\vec{b}$ mit N Komponenten durch

$$A_{ij} = \frac{X_j(x_i)}{\sigma_i}, \qquad b_i = \frac{y_i}{\sigma_i} \tag{4.23}$$

um, wird Gl. 4.22 zu

$$\chi^2 = |\mathbf{A} \cdot \vec{a} - \vec{b}|^2. \tag{4.24}$$

Setzt man weiters die Basisfunktionen X_i in Gl. 4.21 ein, führt dies auf das Gleichungssystem

$$0 = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left(y_i - \sum_{j=1}^{M} a_j X_j(x_i) \right) X_k(x_i) \qquad k = 1, \dots, M \tag{4.25}$$

das äquivalent zu

$$\left(\mathbf{A}^T \cdot \mathbf{A}\right)_{M \times M} \cdot \vec{a}_M = \mathbf{A}^T_{M \times N} \cdot \vec{b}_N \tag{4.26}$$

ist. Die Lösung eines Gleichungssystems der Form $\mathbf{A}\vec{a} = \vec{c}$ gehört zu den Standardproblemen der linearen Algebra und kann getrost den optimierten Bibliotheksfunktionen überlassen werden. Die Auswahl der geeigneten Routine hängt im wesentlichen davon ab, welche Zahlenwerte die Matrixelemente A_{ij} haben. Dabei ist für die χ^2-Anpassung die Methode der *singular value decomposition* am wenigsten empfindlich auf Rundungsfehler.

Was die Abschätzung der Fehler der Fitparameter betrifft, die wir zu Beginn als Bedingung für eine sinnvolle Fitprozedur gefordert hatten, so haben wir Glück. Sie fallen als Nebenprodukt der Berechnung an, denn die Varianzen der a_i sind durch

$$\sigma^2(a_i) = C_{ii} \qquad \text{mit} \qquad C = \left(\mathbf{A}^T\mathbf{A}\right)^{-1} \tag{4.27}$$

gegeben. Hat man das einmal akzeptiert, glaubt man auch, dass es sich bei den Nebendiagonalelementen C_{ij} um die Kovarianzen zwischen a_i und a_j handelt.

Der dritte Punkt unserer Liste war die Abschätzung der Güte des Fits. Sie wird als χ^2-Wahrscheinlichkeitsfunktion $Q(\chi^2|\nu)$ bezeichnet. Vereinfacht gesagt beschreibt sie die Wahrscheinlichkeit, dass man trotz eines richtigen Modells für die Anpassung einen besseren Wert für χ^2 finden kann und sollte daher möglichst klein sein.

Berechnet wird sie mithilfe der unvollständigen Γ-Funktion

$$Q(a,x) = \frac{\int_x^\infty e^{-t}t^{a-1}\,dt}{\int_0^\infty e^{-t}t^{a-1}\,dt} = \frac{1}{\Gamma(a)}\int_x^\infty e^{-t}t^{a-1}\,dt\ , \tag{4.28}$$

wobei für $a = (N-M)/2$ und für $x = \chi^2/2$ einzusetzen sind.

Kapitel 5

Lineare Algebra

Im ersten Teil dieses Kapitels behandeln wir Methoden zur Lösung von linearen Gleichungssystemen - eines der zentralen Probleme in der numerischen Mathematik. Wir beschränken uns in diesem Teil auf die sogenannten direkten Verfahren, welche in einer endlichen Anzahl von Schritten zum exakten Ergebnis führen. Im zweiten Teil des Kapitel lernen wir den iterativen Lanczos Algorithmus kennen, mit welchem Eigenwert und Eigenvektor des Grundzustandes einer hermiteschen Matrix berechnet werden können.

5.1 Lineare Gleichungssysteme

In der numerischen Mathematik werden viele Probleme, z.B. die Interpolation von Funktionen, die Lösung von Differentialgleichungen, die Lösung von Integralgleichungen, etc., auf die Lösung eines linearen Gleichungssystem zurückgeführt. Lineare Gleichungssysteme sind mathematisch bestens untersucht und können unter vorgegebenen (meist schwachen) Bedingungen gelöst werden. Die Eigenschaften linearer Gleichungssysteme finden auch in der formalen Mathematik in vielen Beweisen ihre Anwendung.

Ein lineares Gleichungssystem für die Unbekannten $x_1, x_2, \ldots, x_N$ ist von der Form

$$\mathcal{A}\mathbf{x} = \mathbf{b} \quad \Longleftrightarrow \quad \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1N} \\ A_{21} & A_{22} & \ldots & A_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ A_{M1} & A_{M2} & \ldots & A_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{pmatrix}, \tag{5.1}$$

wobei $\mathbf{x}$ und $\mathbf{b}$ Vektoren der Dimension N sind und $\mathcal{A}$ eine $M \times N$ Matrix ist. Die Matrix $\mathcal{A}$ und der Vektor $\mathbf{b}$ werden als bekannt vorausgesetzt. Man muss nun die folgenden Fälle unterscheiden:

- $M < N$ oder $M = N, \det \mathcal{A} = 0 \rightarrow$ Das Gleichungssystem ist unterbestimmt,
- $M = N, \det \mathcal{A} \neq 0 \rightarrow$ Das Gleichungssystem ist eindeutig lösbar,

- $M > N \rightarrow$ Das Gleichungssystem ist überbestimmt, ausgenommen für den Fall, dass $M - N$ Gleichungen über lineare Abhängigkeiten reduziert werden können.

Im Folgenden beschränken wir uns auf den Fall $M = N$, sodass $\mathcal{A}$ eine quadratische Matrix ist. Ist nun $\det \mathcal{A} = 0$ so spricht man von einem singulären Gleichungssystem, welches nicht eindeutig nach $\mathbf{x}$ aufgelöst werden kann. Nichtsinguläre Gleichungssysteme, d.h. $\det \mathcal{A} \neq 0$, können im Prinzip eindeutig gelöst werden. Bei der numerischen Rechnung hat man aber Schwierigkeiten mit der Genauigkeiten zu erwarten, wenn $\det \mathcal{A}$ sehr klein wird, bzw. wenn die Eigenwerte von $\mathcal{A}$ eine hohe *Dynamik* aufweisen, d.h. über viele Größenordnungen gehen.

Zur Lösung von linearen Gleichungssystemen gibt es je nach Form von $\mathcal{A}$ eine Vielzahl spezieller Verfahren, welche in zwei Gruppen unterteilt werden können,

a Direkte Verfahren,

b Iterative Verfahren.

Direkte Verfahren liefern in endlich vielen Schritten das exakte Ergebnis, wenn man von Rundungsfehlern absieht. Bei iterativen Verfahren erreicht man das exakte Ergebnis erst nach unendlich vielen Schritten. Im vorliegenden Kapitel beschränken wir uns auf die Darstellung von zwei direkten Verfahren, und zwar (a) den Gaußschen Eliminationsalgorithmus und (b) eine spezielle Methode für tridiagonale Matrizen.
Bei den sogenannten iterativen Verfahren nähert man sich, ausgehend von einer Startnäherung, schrittweise der gesuchten Lösung an, bis ein festgelegtes Konvergenzkriterium erreicht ist. Beispiele für solche iterative Methoden sind das Jacobi- bzw. das Gauß-Seidel-Verfahren. Besonders bei dünnbesetzten Matrizen können diese Verfahren sehr effektiv sein. Da die iterativen Verfahren jedoch nicht notwendigerweise konvergieren und in der praktischen Anwendung häufig sehr langsam sind, verzichten wir auf eine weitere Diskussion dieser Methoden.

Abschließend sei noch erwähnt, dass die Verfahren zur Lösung eines linearen Gleichungssystems auch für die Bestimmung der inversen Matrix $\mathcal{A}^{-1}$ verwendet werden können, wenn man die N Gleichungssysteme $(i = 1, 2, \ldots, N)$

$$\begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1N} \\ A_{21} & A_{22} & \ldots & A_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ A_{N1} & A_{N2} & \ldots & A_{NN} \end{pmatrix} \begin{pmatrix} (A^{-1})_{1i} \\ (A^{-1})_{2i} \\ \vdots \\ (A^{-1})_{Ni} \end{pmatrix} = \begin{pmatrix} \delta_{1i} \\ \delta_{2i} \\ \vdots \\ \delta_{Ni} \end{pmatrix} \tag{5.2}$$

löst.

5.1.1 Das Eliminationsverfahren von Gauß

Der Algorithmus

Beim Gaußschen Eliminationsverfahren zur Lösung eines linearen Gleichungssystems wird durch geeignete Vertauschung und Linearkombination von Zeilen des Gleichungs-

system (5.2) die Matrix $\mathcal{A}$ schrittweise auf Dreiecksgestalt gebracht:

$$\begin{pmatrix} r_{11} & r_{12} & \dots & r_{1N} \\ 0 & r_{22} & \dots & r_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & r_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix} . \tag{5.3}$$

Die Bestimmung der Lösung $\mathbf{x}$ aus einem Gleichungssystem mit Dreiecksgestalt ist sehr einfach. Man erhält:

$$x_i = \frac{1}{r_{ii}} \left(c_i - \sum_{\ell=i+1}^{N} r_{il} x_l \right) \quad , i = N, N-1, \dots, 1 . \tag{5.4}$$

Die Erzeugung eines äquivalenten linearen Gleichungssystems mit Dreiecksgestalt ist ein elementarer Prozess, welcher in $N-1$ Schritten leicht erreicht werden kann. Zur Illustration des Prozesses zur Umformung des Gleichungssystems betrachten wir den i-ten Schritt,

$$\mathcal{A}^{(i)}\mathbf{x} = \mathbf{c}^{(i)} \quad \Longrightarrow \quad \mathcal{A}^{(i+1)}\mathbf{x} = \mathbf{c}^{(i+1)} . \tag{5.5}$$

Hierbei ist $\mathbf{c}^{(i)}$ der aus $\mathbf{b}$ nach i Schritten abgeleitete Vektor der rechten Seite von (5.2) und $\mathcal{A}^{(i)}$ bis zur Zeile i bereits eine obere Dreiecksmatrix, d.h.

$$\mathcal{A}^{(i)} = \begin{pmatrix} a_{11}^{(i)} & a_{12}^{(i)} & \dots & a_{1i-1}^{(i)} & a_{1i}^{(i)} & a_{1i+1}^{(i)} & \dots & a_{1N}^{(i)} \\ 0 & a_{22}^{(i)} & \dots & a_{2i-1}^{(i)} & a_{2i}^{(i)} & a_{2i+1}^{(i)} & \dots & a_{2N}^{(i)} \\ \vdots & 0 & \ddots & \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & \dots & a_{i-1i-1}^{(i)} & a_{i-1i}^{(i)} & a_{i-1i+1}^{(i)} & \dots & a_{i-1N}^{(i)} \\ 0 & 0 & \dots & 0 & a_{ii}^{(i)} & a_{ii+1}^{(i)} & \dots & a_{iN}^{(i)} \\ 0 & 0 & \dots & 0 & a_{i+1i}^{(i)} & a_{i+1i+1}^{(i)} & \dots & a_{i+1N}^{(i)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_{Ni}^{(i)} & a_{Ni+1}^{(i)} & \dots & a_{NN}^{(i)} . \end{pmatrix} \tag{5.6}$$

Startpunkt ist $\mathcal{A}^{(1)} = \mathcal{A}$ und $\mathbf{c}^{(1)} = \mathbf{b}$. Um die entsprechenden Umformungen des i-ten Schrittes (5.5) durchzuführen, sind drei Teilschritte erforderlich:

a) `Suche eines geeigneten Pivotelementes` r_{ii}.
Hier wollen wir uns auf die Spalten-Pivotsuche beschränken, d.h. man sucht das Element mit dem größten Absolutwert aus der i-ten Teilspalte $a_{ji}^{(i)}, j = i, \dots, N$. Sei dieses das Element der r-ten Zeile, dann setzt man den Pivot $r_{ii} = a_{ri}^{(i)}$. Existiert kein nichtverschwindendes Element in der i-ten Teilspalte, d.h. $a_{ji}^{(i)} = 0 \; \forall j, j = i, \dots, N$, so ist die Matrix $\mathcal{A}$ singulär.

b) `Vertauschen der` r`-ten mit der` i`-ten Zeile in` $\mathcal{A}^{(i)}$ `und` $\mathbf{c}^{(i)}$.
Bezeichnet man mit $\bar{\mathcal{A}}^{(i)}$ und $\bar{\mathbf{c}}^{(i)}$ die Matrix $\mathcal{A}^{(i)}$ und den Vektor $\mathbf{c}^{(i)}$ nach Vertauschen der Zeilen, so lautet nun das Gleichungssystem $\bar{\mathcal{A}}^{(i)}\mathbf{x} = \bar{\mathbf{c}}^{(i)}$.

c) `Berechnen der Elemente von` $\mathcal{A}^{(i+1)}$ `und` $\mathbf{c}^{(i+1)}$.
Die unteren $N-i$ Zeilen sind von der Neuberechnung betroffen. Man zieht dabei von der m-ten Zeile ein Vielfaches der i-ten Zeile der Matrix $\bar{\mathcal{A}}^{(i)}$ ab, sodass das Matrixelement $a_{mi}^{(i+1)} = 0$ wird. Dies wird für alle $m = i+1, i+2, \ldots, N$ durchgeführt.

- Die Matrixelemente der ersten i Zeilen bleiben unverändert, d.h. $a_{mn}^{(i+1)} = \bar{a}_{mn}^{(i)} = r_{mn}$ und $c_m^{(i+1)} = \bar{c}^{(i)}$ für alle $m = 1, 2, \ldots, i$ und $n = 1, 2, \ldots, N$.
- Die Transformation der Matrixelemente der übrigen Zeilen $m = i+1, i+2, \ldots, N$ erfolgt nach der Vorschrift

$$a_{mn}^{(i+1)} = \bar{a}_{mn}^{(i)} - \lambda_{mi}\bar{a}_{in}^{(i)}\,, \quad n = 1, 2, \ldots, N \tag{5.7}$$

mit

$$\lambda_{mi} = \frac{\bar{a}_{mi}^{(i)}}{r_{ii}} \quad \text{für alle } m = i+1, i+2, \ldots, N\,. \tag{5.8}$$

- Die Transformation der übrigen Zeilen des Vektors $\mathbf{c}^{(i)}$ erfolgt nach der Vorschrift

$$c_m^{(i+1)} = \bar{c}_m^{(i)} - \lambda_{mi}\bar{c}_i^{(i)}\,, \quad \text{für alle } m = i+1, i+2, \ldots, N\,. \tag{5.9}$$

Nach $N-1$ Schritten ergibt sich für $\mathcal{A}^{(N)}$ die gewünschte obere Dreiecksmatrix und der entsprechende Vektor $\mathbf{c}^{(N)}$, sodass man mittels (5.4) den Lösungsvektor $\mathbf{x}$ des linearen Gleichungssystems (5.2) erhält. Wie man leicht durch Abzählen der erforderlichen Operationen in diesem Algorithmus sieht, erhöht sich mit steigender Zahl N der Unbekannten der Aufwand mit N^3.

Betrachtet man nun den Algorithmus in seiner Gesamtheit, so stellt man fest, dass die Multiplikationsfaktoren λ_{mn} mit $\lambda_{mm} = 1$ eine untere Dreiecksmatrix $\mathcal{L}$ bilden. Nimmt man keine Zeilenvertauschungen im Rahmen der Pivotsuche vor, dann gilt der Zusammenhang

$$\mathcal{L}\mathcal{R} = \mathcal{A}\,, \tag{5.10}$$

wobei die Bezeichnung $\mathcal{R} = \mathcal{A}^{(N)}$ verwendet wurde. Im Gaußschen Eliminationsverfahren wird also eine Darstellung der Matrix $\mathcal{A}$ als Produkt einer unteren mit einer oberen Dreiecksmatrix realisiert. Ohne Pivotsuche lässt sich daher der Eliminationsalgorithmus kompakt darstellen

$$r_{ik} = a_{ik} - \sum_{j=1}^{i-1} \lambda_{ij} r_{jk} \quad k = i, i+1, \ldots, N\,, \tag{5.11}$$

$$\lambda_{ki} = \frac{1}{r_{ii}}\left(a_{ki} - \sum_{j=1}^{i-1} \lambda_{kj} r_{ji}\right) \quad k = i+1, i+2, \ldots, N\,. \tag{5.12}$$

Zur Berechnung der Matrixelemente r_{ik}, λ_{ik} können die Indices in unterschiedlicher Reihenfolgen durchlaufen werden. Dementsprechend lassen sich zwei unterschiedliche

Algorithmen formulieren und zwar jenen nach Banachiewicz und das Verfahren nach Crout. In der Literatur (z.B. NAG Programmbibliothek) gibt es daher stets Hinweise auf den aktuell verwendeten Algorithmus.

Abschließend soll noch festgehalten werden, dass im Gaußschen Eliminationsalgorithmus durch die Zerlegung in Dreiecksmatrizen auch die Determinante der Matrix $\mathcal{A}$ gegeben ist,

$$\det \mathcal{A} = \prod_{i=1}^{N} r_{ii} \,. \tag{5.13}$$

Pivotsuche und Rundungsfehlereinfluss

In Hinblick auf die numerische Genauigkeit, d.h. die Minimierung von Rundungsfehlern, ist die Wahl des Pivot ausschlaggebend. Dies lässt sich leicht an einem kleinem Beispiel demonstrieren, das wir auf zwei Stellen Gleitkommagenauigkeit berechnen wollen.

Abhängigkeit der Genauigkeit von Pivotwahl

Wir betrachten in zweistelliger Gleitpunktrechnung das Gleichungssystem

$$\begin{pmatrix} 0.5 \times 10^{-2} & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5 \\ 1.0 \end{pmatrix} \quad \Longrightarrow \text{ exakte Lösung } \begin{array}{l} x = \frac{5000}{9950} = 0.503 \\ y = \frac{4950}{9950} = 0.497 \end{array}$$

$r_{11} = 0.005$

$$\begin{pmatrix} 0.005 & 1 \\ 0 & -200 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5 \\ -99.0 \end{pmatrix} \quad \Longrightarrow \text{ Lösung } \begin{array}{l} x = \frac{0}{200} = 0.00 \\ y = \frac{99}{200} = 0.50 \end{array}$$

$r_{11} = 1$

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1.0 \\ 0.5 \end{pmatrix} \quad \Longrightarrow \text{ Lösung } \begin{array}{l} x = \frac{5}{10} = 0.50 \\ y = \frac{5}{10} = 0.50 \end{array}$$

Aus dem Beispiel könnte der falsche Eindruck entstehen, dass die Wahl des größten Elements als Pivot (wie es auch im oben ausgeführten Algorithmus enthalten ist) zu dem geringsten Rundungsfehler führt. Dies ist aber im allgemeinen nicht der Fall, da es auf die Gesamtheit der involvierten Restmatrix ankommt.

Neben der *Teilpivotsuche* oder *Spaltenpivotsuche* wie sie im oben besprochenen Algorithmus verwendet wurde, gibt es noch die *Totalpivotsuche*. Bei letzterer wird das absolut größte Matrixelement der gesamten Restmatrix (gebildet aus i-ter bis N-ter Zeile und Spalte) als Pivotelement r_{ii} verwendet. Die Totalpivotsuche ist allerdings programmtechnisch wesentlich aufwändiger und erfordert nicht nur ein Vertauschen der Zeilen, sondern auch der Spalten. Dies impliziert einen wesentlich höheren Verwaltungsaufwand, da man auch die Reihenfolge der Elemente in den Vektoren $\mathbf{x}$ und $\mathbf{b}$ umstellen und registrieren muss.

Beim Verfahren von *Householder und Schmidt* werden unitäre Matrizen $\mathcal{P}^{(i)}, \quad i = 1, 2, \ldots, N-1$ bestimmt, welche die folgenden Transformationen vermitteln

$$\mathcal{P}^{(i)} \begin{pmatrix} a_{ii}^{(i)} \\ a_{i+1i}^{(i)} \\ \vdots \\ a_{Ni}^{(i)} \end{pmatrix} = \begin{pmatrix} r_{ii} \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{mit} \quad |r_{ii}| = \sqrt{\sum_{j=i}^{N} \left(a_{ji}^{(i)}\right)^2}. \tag{5.14}$$

In der Householder Transformation sind alle Zeilen der jeweiligen Restmatrix gleichermaßen involviert. Durch die Unitarität der Transformation bleiben die Längen jedes Spaltenvektors konstant und man erhält eine niedrige Konditionszahl des Algorithmus, d.h. die Rundungsfehler bleiben minimal. Für Details der Transformation von Householder und Schmidt wird auf die Literatur in Numerischer Mathematik verwiesen (siehe z.B. Stoer [7]).

5.1.2 Lineare Gleichungssysteme mit Tridiagonaler Matrix

Viele Probleme werden auf die Lösung von Gleichungssystemen mit tridiagonaler Matrix

$$\mathcal{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & \ldots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & a_{N-1N-2} & a_{N-1N-1} & a_{N-1N} \\ 0 & \ldots & 0 & 0 & a_{NN-1} & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{N-1} \\ b_N \end{pmatrix} \tag{5.15}$$

zurückgeführt. Die spezielle Strukur der Matrix erlaubt eine wesentliche Reduktion der Rechenzeit. Zunächst machen wir den Ansatz einer linearen Rekursionsvorschrift zwischen den Unbekannten

$$x_{i+1} = \alpha_i x_i + \beta_i \,, \tag{5.16}$$

wobei α_i und β_i noch zu bestimmende Größen sind. Einsetzen des Ansatzes (5.16) in das lineare Gleichungssystem (5.15) liefert für die i-te Zeile

$$a_{ii-1}x_{i-1} + a_{ii}x_i + a_{ii+1}x_{i+1} = a_{ii-1}x_{i-1} + a_{ii}x_i + a_{ii+1}(\alpha_i x_i + \beta_i) = b_i. \tag{5.17}$$

Separation von x_i und Vergleich der Koeffizienten liefert die Rekursionsvorschrift für die Koeffizienten α_i und β_i

$$\alpha_{i-1} = -\frac{a_{ii-1}}{a_{ii} + a_{ii+1}\alpha_i}\,, \quad \beta_{i-1} = -\frac{\beta_i a_{ii+1} - b_i}{a_{ii} + a_{ii+1}\alpha_i}\,. \tag{5.18}$$

Den Rekursionsbeginn erhält man aus einer näheren Betrachtung der letzten Zeile

$$a_{NN-1}x_{N-1} + a_{NN}x_N = b_N \quad \Longrightarrow x_N = -\frac{a_{NN-1}}{a_{NN}}x_{N-1} + \frac{b_N}{a_{NN}} = \alpha_{N-1}x_{N-1} + \beta_{N-1}\,. \tag{5.19}$$

Vergleich mit der Rekursionsvorschrift (5.18) zeigt, dass man diese Werte am einfachsten erhält, wenn man $\alpha_N = \beta_N = 0$ verwendet.

Die Lösung des Gleichungssystems mit tridiagonaler Matrix erfordert $2N$ Schritte. Beginnend mit dem Ansatz $\alpha_N = \beta_N = 0$ berechnet man nun α_i und β_i für $i = N-1, N-2, \ldots, 1$ über die Rekursionsvorschrift (5.18). Mit der Kenntnis von α_1 und β_1 lässt sich die Unbekannte x_1 berechnen,

$$x_1 = \frac{b_1 - a_{12}\beta_1}{a_{11} + a_{12}\alpha_1}\,. \tag{5.20}$$

Anschließend kann man über die Rekursionsvorschrift (5.16) die Unbekannten $x_2, x_3, \ldots, x_N$ berechnen.

5.1.3 Einbinden von Bibliotheksprogrammen

Die Lösung von linearen Gleichungssystemen ist eine der wichtigsten Problemstellungen im Zusammenhang mit der numerischen Behandlung physikalischer Fragen. Die verschiedenen Facetten solcher Gleichungssysteme erfordern zum Teil spezielle Algorithmen, um auch die geforderten Genauigkeiten zu erreichen. Es ist daher im Allgemeinen sinnvoll für die Lösung von Problemen der linearen Algebra auf sogenannte Programmbibliotheken zurückzugreifen, die spezielle und ausgereifte Algorithmen enthalten. Beispiel für eine allgemein zugängliche Programmbibliothek ist die CERNLIB, welche insbesondere die für die numerische Physik relevanten Algorithmen enthält.

5.2 Der Lanczos Algorithmus

In diesem zweiten Teil des Kapitels zur linearen Algebra lernen wir den sogenannten Lanczos Algorithmus kennen. Mit diesem Algorithmus kann man extrem effektiv die Grundzustandsenergie, sowie der Eigenvektor des Grundzustands einer hermitschen Matrix bestimmen. Diese Problemstellung hat offensichtlich eine direkte physikalische Motivation, denn im Grenzfall niedriger Temperaturen ist genau der Grundzustand eines Hamilton-Operators von besonderem Interesse. Im letzten Abschnitt über die linearen Gleichungssysteme haben wir gesehen, dass die Handhabung von Matritzen in tridiagonaler Form besonders einfach ist. Dies kann man sich auch zu Nutze machen, wenn man das Eigensystem einer Matrix bestimmen möchte. Die fundamentale Idee hinter dem Lanczos Algorithmus ist es nämlich eine spezielle Basis zu finden, in welcher die Hamilton-Matrix in tridiagonaler Form geschrieben werden kann. Die Effizienz des Verfahrens begründet sich darauf, dass diese Transformation der Hamilton-Matrix auf speziell ausgewählten kleineren Unterräumen geschieht, die solange vergrößert werden, bis die Grundzustandsenergie konvergiert ist.

5.2.1 Rayleigh-Quotient & Gram-Schmidt Orthonormierung

Sei $\mathcal{A}$ eine nun eine $n \times n$ hermitesche Matrix mit reellen Eigenwerten $\lambda_1[A] \leq \cdots \leq \lambda_n[A]$ und $\mathbf{x} \in \mathbb{R}^n$, dann ist der Rayleigh Quotient definiert als

$$\rho(\mathbf{x}, \mathcal{A}) = \frac{\mathbf{x}^T \mathcal{A} \mathbf{x}}{\mathbf{x}^T \mathbf{x}} \tag{5.21}$$

und die extremalen Eigenwerte $\lambda_1[A]$ und $\lambda_n[A]$ können geschrieben werden als

$$\lambda_1[A] = \min \rho(\mathbf{x}, \mathcal{A}) \qquad \lambda_n[A] = \max \rho(\mathbf{x}, \mathcal{A}) \tag{5.22}$$

Das kann man nachvollziehen, wenn man den Rayleigh Quotienten umschreibt mittels des Orthonormierten Eigensystems $\mathbf{v}_i$ von $\mathcal{A}$. Damit folgt mit $U = [\mathbf{v}_1 \cdots \mathbf{v}_n]$, dass

$$U^T \mathcal{A} U = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \tag{5.23}$$

Schreiben wir nun $\mathbf{x}$ um als Linearkombination der $\mathbf{v}_i$ als $\mathbf{x} = U\mathbf{z}$ folgt für den Rayleigh Quotienten

$$\rho(\mathbf{x}, \mathcal{A}) = \frac{\mathbf{z}^T U^T \mathcal{A} U \mathbf{z}}{\mathbf{z}^T U^T U \mathbf{z}} = \frac{z_1^2 \lambda_1 + \cdots + z_n^2 \lambda_n}{z_1^2 + \cdots + z_n^2} \tag{5.24}$$

und damit der Zusammenhang (5.22)

Sei nun $[\mathbf{q}_1, \cdots, \mathbf{q}_m]$ eine allgemeine orthonormale Basis auf $\mathbb{R}^m$ wobei $\mathbf{q}_i \in \mathbb{R}^n$ und $\mathbb{R}^m \subset \mathbb{R}^n$. Dann kann jeder Vektor $\mathbf{x} \in \mathbb{R}^n$ als Linearkombination der $\mathbf{q}_i$ geschrieben werden

$$\mathbf{x} = [\mathbf{q}_1, \cdots, \mathbf{q}_m]\mathbf{z} = Q\mathbf{z} \tag{5.25}$$

wobei $\mathbf{z} \in \mathbb{R}^m$ und Q eine $m \times n$ Matrix ist. Die beste Näherung für die Eigenwerte von $\mathcal{A}$ erhält man dann aus der Diagonalisierung der $m \times m$ Matrix $\mathcal{H}$

$$\mathcal{H} = Q^T \mathcal{A} Q \tag{5.26}$$

Die orthonormale Basis $[\mathbf{q}_1, \cdots, \mathbf{q}_m]$ kann man nun aus einem beliebigen Satz linear unabhängiger Vektoren $S_m = [\mathbf{s}_1, \cdots, \mathbf{s}_m]$ mit $\mathbf{s}_i \in \mathbb{R}^n$ konstruieren. Man benutzt dabei das Verfahren von *Gram-Schmidt*, bei welchem die Vektoren $\mathbf{q}_i$, ausgehend

von einem beliebigen normierten Startvektor $\mathbf{q}_1 = \mathbf{s}_1/|\mathbf{s}_1|$ sukzessive konstruiert werden

$$\mathbf{q'}_i = \mathbf{s}_i - \sum_{j=1}^{i-1} \mathbf{q}_j(\mathbf{q}_j \cdot \mathbf{s}_i) \qquad\qquad \mathbf{q}_i = \frac{\mathbf{q'}_i}{|\mathbf{q'}_i|} \tag{5.27}$$

oder, als Faktorisierung von $S = QR$ geschrieben wobei R eine $m \times m$ obere Dreiecksmatrix ist

$$[\mathbf{s}_1, \cdots, \mathbf{s}_m] = [\mathbf{q}_1, \cdots, \mathbf{q}_m] \begin{pmatrix} |\mathbf{q'}_1| & \mathbf{q}_1 \cdot \mathbf{s}_1 & \cdots & \mathbf{q}_1 \cdot \mathbf{s}_m \\ 0 & |\mathbf{q'}_2| & \cdots & \mathbf{q}_2 \cdot \mathbf{s}_m \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & |\mathbf{q'}_m| \end{pmatrix} \tag{5.28}$$

An diesem Punkt halten wir also fest, dass wir die extremalen Eigenwerte $\lambda_1[\mathcal{A}]$ und $\lambda_n[\mathcal{A}]$ der $n \times n$ Matrix $\mathcal{A}$ annähern können, indem wir eine kleinere $m \times m$ Matrix $\mathcal{H} = Q^T \mathcal{A} Q$ diagonalisieren und damit als Eigenwerte $\lambda_1[\mathcal{H}]$ und $\lambda_m[\mathcal{H}]$ erhalten. Die entsprechenden (genäherten) Eigenvektoren von A sind dann gegeben durch $\mathbf{v}_i = Q\mathbf{g}_i$ $(i = 1, m)$, wobei $\mathbf{g}_i$ die Eigenvektoren von $\mathcal{H}$ sind.

5.2.2 Krylov-Basis und Lanczos Rekursionsformel

Der letzte Schritt besteht nun darin, eine Basis zu wählen, in der $\mathcal{H}$ besonders einfach diagonalisiert werden kann. Wir werden sehen, dass diese Basis durch die sogenannte Krylov Matrix gegeben ist.

Sei S_m der Krylov Unterraum, der durch die Krylov Matrix aufgespannt wird

$$\mathcal{K}^m(\mathbf{f}) = [\mathbf{f}, \mathcal{A}\mathbf{f}, \mathcal{A}^2\mathbf{f}, \cdots, \mathcal{A}^{m-1}\mathbf{f}] \tag{5.29}$$

wobei $\mathbf{f} \in \ ^n$. Mit dieser Definition folgt, dass die Faktorisierung (5.28) von $\mathcal{K}^m(\mathbf{f}) = Q_m R$ eine Matrix Q_m liefert, für welche

$$\mathcal{T}_m = Q_m^T \mathcal{A} Q_m \tag{5.30}$$

eine tridiagonale Matrix ist. Dies kann man sich folgendermaßen klarmachen: Für $i > j + 1$ gilt $\mathbf{q}_i^T(\mathcal{A}\mathbf{q}_j) = 0$ da zum einen per definitionem $\mathcal{A}\mathbf{q}_j$ eine Teilmenge des Krylov Unterraumes S_{j+1} ist $\mathcal{A}\mathbf{q}_j \subset S_{j+1}$. Darüber hinaus ist durch die Gram-Schmidt Orthonormierung sichergestelt, daß $\mathbf{q}_i$ für $i > j + 1$ orthogonal zu allen Vektoren $\mathbf{s}_{j+1}$ ist: $\mathbf{q}_i \perp \mathbf{s}_{j+1}$ für $i > j + 1$.
Durch die Hermitizität von $\mathcal{A}$ gilt des weiteren $\mathbf{q}_i^T(\mathcal{A}\mathbf{q}_j) = \mathbf{q}_j^T(\mathcal{A}^T\mathbf{q}_i) = \mathbf{q}_j^T(\mathcal{A}\mathbf{q}_i) = 0$

für $j > i+1$ bzw. $i < j-1$. Damit folgt:

$$\mathcal{T}_m = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-2} & \alpha_{m-1} & \beta_{m-1} \\ & & & \beta_{m-1} & \alpha_m \end{pmatrix} \tag{5.31}$$

wobei wir die Elemente der tridiagonalen Matrix folgendermaßen definieren

$$\begin{aligned} \alpha_j &= \mathbf{q}_j^T \mathcal{A} \mathbf{q}_j & j &= 1, \cdots, m \\ \beta_j &= \mathbf{q}_{j+1}^T \mathcal{A} \mathbf{q}_j & j &= 1, \cdots, m-1 \end{aligned}$$

Aus der Tridiagonalität folgt, dass

$$\mathcal{A}\mathbf{q}_i = \beta_{i-1}\mathbf{q}_{i-1} + \alpha_i \mathbf{q}_i + \beta_i \mathbf{q}_{i+1} \quad (2 \leq i \leq m-1) \tag{5.32}$$

Definieren wir außerdem $\mathbf{q}_0 = 0$, dann gilt Gleichung (5.32) auch für $i = 1$. Sei nun $\mathbf{r}_i \equiv \beta_i \mathbf{q}_{i+1}$, dann können wir schreiben

$$\mathbf{r}_i = \mathcal{A}\mathbf{q}_i - \beta_{i-1}\mathbf{q}_{i-1} - \alpha_i \mathbf{q}_i \quad (1 \leq i \leq m-1) \tag{5.33}$$

Mit diesen Voraussetzungen können wir den eigentlichen Lanczos Algorithmus formulieren als:

Gegeben: $\mathbf{r}_0, \beta_0 = |\mathbf{r}_0| \quad (\mathbf{q}_0 = 0)$
  for $i = 1$ to m **do**
    $\mathbf{q}_i \leftarrow \mathbf{r}_{i-1}/\beta_{i-1}$
    $\mathbf{r}_i \leftarrow \mathcal{A}\mathbf{q}_i - \beta_{i-1}\mathbf{q}_{i-1}$
    $\alpha_i \leftarrow \mathbf{q}_i^T \mathbf{r}_i$
    $\mathbf{r}_i \leftarrow \mathbf{r}_i - \alpha_i \mathbf{q}_i$
    $\beta_i = |\mathbf{r}_i|$ (dann wenn$i \leq m-1$)
  end for

Man erhöht dann die Dimension des Krylov Unterraumes solange, bis $\lambda_1[\mathcal{T}]$ konvergiert ist.

Kapitel 6

Gewöhnlichen Differentialgleichungen

6.1 Einleitung

Viele physikalische und technische Probleme erfordern in ihrer mathematischen Formulierung die Lösung einer Differentialgleichung. Im einfachsten Fall ist eine differenzierbare Funktion $y(x)$ gesucht, welche von einer reelen Veränderlichen x abhängt und die Gleichung

$$y' = f(x, y) \tag{6.1}$$

erfüllt. In der Lehrveranstaltung *Datenverarbeitung für TPH 2* beschäftigen wir uns ausschlielich mit diesen sogenannten *gewöhnlichen Differentialgleichungen 1. Ordnung*. Im Allgemeinen wird die Differentialgleichung (6.1) von unendlich vielen Funktionen $y(x)$ gelöst. Für eine eindeutige Lösung sind immer zusätzliche Forderungen, wie *Rand- oder Anfangswertbedingungen* erforderlich. Je nach Problemstellung spricht man von *Anfangs-* oder *Randwertproblemen*.

Für die numerische Lösung gewöhnlicher Differentialgleichungen stehen heute eine Vielzahl von allgemeinen und speziellen Verfahren zur Verfügung, die wichtigsten Klassen von Algortihmen sind:

a. Einschrittverfahren
b. Mehrschrittverfahren
c. Extrapolationsverfahren
d. Variationsverfahren

Im Rahmen dieser Lehrveranstaltung behandeln wir nur Einschrittverfahren, das Konzept von Mehrschrittverfahren und das Numerov-Verfahren, einen effiziente Methode zur Lösung spezieller Differentialgleichungen 2. Ordnung. Für vertiefte Behandlung und weitere Algorithmen verweisen wir auf die einschlägige Literatur der Numerischen Mathematik, z.B. Stoer [7].

In den entsprechenden Kapiteln des Abschnitts *Computational Physics* (Kapitel 1 und 2) werden Anwendungen der Methoden zur Lösung von gewöhnlichen Differentialgleichungen an zwei typischen Fragestellungen der Physik diskutiert. Kapitel 1

beschäftigt sich mit grundlegenden Fragestellungen zu Schwingungsphänomenen, welche im Zusammenhang mit verschiedenen Variationen eines Pendels behandelt werden. Im Kapitel 2 werden am Beispiel des Wasserstoff-Atoms die Lösungen von Bindungs- und Streuproblemen in einfachen quantenmechanischen Systemen vorgestellt.

6.2 Systeme gewöhnlicher Differentialgleichungen erster Ordnung

Ein System von gewöhnlichen Differentialgleichungen erster Ordnung hat die Form

$$\begin{array}{rcl} y_1' &=& f_1(x, y_1, \ldots, y_n) \\ y_2' &=& f_2(x, y_1, \ldots, y_n) \\ \vdots & & \vdots \\ y_n' &=& f_n(x, y_1, \ldots, y_n) \end{array} \quad , \tag{6.2}$$

wobei die Differentialgleichungen durch n Funktionen gegeben sind, $f_i, i = 1, \ldots, n$. Das Differentialgleichungssystem lässt sich durch eine Vektornotation kompakt anschreiben

$$\underline{y'} = \underline{f}(x, \underline{y}) \, . \tag{6.3}$$

Die Vektoren sind dabei wie folgt definiert

$$\underline{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \quad \text{und} \quad \underline{f}(x, \underline{y}) = \begin{pmatrix} f_1(x, y_1, \ldots, y_n) \\ \vdots \\ f_n(x, y_1, \ldots, y_n) \end{pmatrix} \, . \tag{6.4}$$

Es ist wichtig festzuhalten, dass man jedes Anfangswertproblem einer Einzeldifferentialgleichung n-ter Ordrnung,

$$y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)}) \, , \tag{6.5}$$

auf ein Anfangswertproblem eines Systems von n Differentialgleichungen erster Ordnung zurückführen kann. Definiert man nun

$$z_1 = y \, , \quad z_2 = y' \, , \quad \ldots \, , \quad z_n = y^{(n-1)} \tag{6.6}$$

so ergibt sich das System von Differentialgleichungen erster Ordnung

$$\begin{array}{rcl} z_1' &=& z_2 \\ z_2' &=& z_3 \\ \vdots & & \vdots \\ z_n' &=& f_n(x, z_1, \ldots, z_n) \end{array} \quad . \tag{6.7}$$

In jedem System von gewöhnlichen Differentialgleichungen erster Ordnung lässt sich durch die zusätzliche Definition von $y_0 = x$, die unabhängig Veränderliche x auf der rechten Seite eliminieren, d.h. man erhält ein System der Form

$$\begin{array}{rcl} y_0' &=& 1 \\ y_1' &=& f_1(y_0, y_1, \ldots, y_n) \\ y_2' &=& f_2(y_0, y_1, \ldots, y_n) \\ \vdots & & \vdots \\ y_n' &=& f_n(y_0, y_1, \ldots, y_n) \end{array} \quad . \tag{6.8}$$

Dies wird als autonomes System von Differentialgleichungen bezeichnet und vereinfacht die numerische Implementierung ohne Genauigkeitsverlust.

Explizite Einschrittverfahren zur Lösung von gewöhnlichen Differentialgleichungen sind durch die Rekursionsvorschrift

$$\underline{y}(x_{i+1}) = \underline{y}(x_i) + h_{i+1}\underline{\Phi}(x_i, \underline{y}(x_i); h_{i+1}) \quad \text{mit} \quad \underline{y}(x_0) = \underline{y}(x_a) \tag{6.9}$$

gegeben, wobei $x_i, i = 0, \ldots, N$ adequat gewählte Stützstellen im zu untersuchenden Intervall $[x_a, x_e]$, $h_{i+1} = x_{i+1} - x_i$ und $\underline{y}(x_i)$ der Wert der Lösung $\underline{y}(x)$ an der Stelle $x = x_i$ sind. Die Funktion $\Phi(x, \underline{y}(x), h)$ ist charakteristisch für das verwendete Einschrittverfahren. Beginnend vom Anfangs- bzw. Randwert $\underline{y}(x_a)$ lässt sich die Lösung $\underline{y}(x)$ sukzessive berechnen. In den folgenden Unterkapiteln werden die einfachsten Einschrittverfahren kurz beschrieben.

6.3 Einschrittverfahren

6.3.1 Einschrittverfahren erster und zweiter Ordnung

Das einfachste Einschrittverfahren ist ohne Zweifel das Euler- oder Polygonzugverfahren, welches wir hier am Beispiel einer Einzeldifferentialgleichung,

$$y' = f(x, y)\,, \tag{6.10}$$

betrachten wollen. Ist y hinreichend oft differenzierbar, so kann man eine Reihenentwicklung der Lösung $y(x)$ durchführen

$$y(x+h) = \sum_{k=0}^{m} \frac{h^k}{k!}\, y^{(k)}(x) + \frac{h^{m+1}}{(m+1)!} y^{(m+1)}(x + \theta h) \quad \text{mit} \quad 0 \le \theta \le 1\,. \tag{6.11}$$

Man benötigt in dieser Entwicklung die Kenntnis der Ableitungen $y^{(k)}(x)$. Die niedrigsten Ableitungen lassen sich einfach aus der Funktion $f(x, y)$ bestimmen:

$$y' = f(x, y)\,, \tag{6.12}$$

$$y'' = \frac{d^2y}{dx^2} = \frac{df}{dx} + \frac{df}{dy}\frac{dy}{dx} = f_x + f_y f\,, \tag{6.13}$$

$$y^{(3)} = f_{xx} + 2f f_{xy} + f_{yy} f^2 f_x f_y + f f_y^2\,, \tag{6.14}$$

$$\ldots \quad \ldots \quad \ldots \tag{6.15}$$

Je nach der Wahl von m lassen sich unter Vernachlässigung des Resttermes verschiedene Verfahren ableiten.

Das *Euler- oder Poligonzugverfahren* ergibt sich bei der Wahl $m = 1$. Vernachlässigung des Resttermes für $m = 1$ führt auf die Rekursionsbeziehung,

$$y(x+h) = y(x) + hf(x, y(x)) \quad \text{d.h.} \quad \Phi(x, y; h) = f(x, y)\,. \tag{6.16}$$

Beim Euler- oder Polygonzugverfahren wird also der Differentialquotient durch den Differenzenquotienten ersetzt, d.h.

$$y' \;\approx\; \frac{y(x+h) - y(x)}{h} \tag{6.17}$$

Will man die Entwicklung (6.11) bis $m = 2$ berücksichtigen, so setzt man die Funktion Φ in folgender Form an,

$$\Phi(x, y; h) = a_1 f(x, y) + a_2 f(x + p_1 h, y + p_2 h f(x, y)) \tag{6.18}$$

Eine Entwicklung von (6.18) bis zur ersten Ordnung in h liefert

$$\Phi(x, y; h) = (a_1 + a_2) f(x, y) + a_2 h \left[p_1 f_x(x, y) + p_2 f_y(x, y) f(x, y)\right] + o(h^2)\,. \tag{6.19}$$

Wählt man $a_1 + a_2 = 1$ und $p_1 a_2 = p_2 a_2 = \frac{1}{2}$ so sind die Terme der Reihenentwicklung (6.11) bis $m = 2$ durch die Funktion $f(x, y)$ ausgedrückt. Man erhält damit das *Verfahren von Heun*, welches auch als *verbessertes Polygonzugverfahren* bezeichnet wird,

$$\begin{aligned} a_1 = a_2 &= \tfrac{1}{2}\,, \qquad p_1 = p_2 = 1 \\ \Phi(x, y; h) &= \frac{1}{2}\left[f(x, y) + f(x + h, y + h f(x, y))\right]\,, \end{aligned} \tag{6.20}$$

und das *modifizierte Euler-Verfahren*

$$\begin{aligned} a_1 = 0\;; \qquad & a_2 = 1\;; \quad p_1 = p_2 = \tfrac{1}{2} \\ \Phi(x, y; h) \;=\; & f(x + \tfrac{1}{2}h, y + \tfrac{1}{2} h f(x, y))\,. \end{aligned} \tag{6.21}$$

Betrachtet man die Ausdrücke für die Funktion Φ, so sieht man, dass in den letzten beiden Verfahren die Funktion $f(x, y)$ zweimal aufgerufen werden muss, während im Euler-Verfahren nur ein Aufruf von $f(x, y)$ pro Schritt erforderlich ist.

6.3.2 Konsistenz und Konvergenz von Einschrittverfahren

Bei der Diskussion der Güte eines Einschrittverfahrens muss man zwischen Konsistenz und Konvergenz des Verfahrens unterscheiden. Die Konsistenz eines Verfahrens bezieht sich auf die Eigenschaften der Funktion Φ, während Konvergenz die Lösung der Differentialgleichung betrifft.

Man nennt das Einschrittverfahren *konsistent*, wenn die Funktion Φ der Bedingung genügt

$$\lim_{h \to 0} \Phi(x, y; h) = f(x, y), \quad x \epsilon [a, b], \quad y \in R\,. \tag{6.22}$$

Man spricht von einem Verfahren der Ordnung p, falls

$$\tau(x,y;h) = \Delta(x,y;h) - \Phi(x,y;h) = \mathcal{O}(h^p) \tag{6.23}$$

gilt, wobei der Differenzenquotient folgendermaßen definiert ist

$$\Delta(x,y;h) = \begin{cases} \frac{z(y+h)-z(x)}{h} & h \neq 0, \\ f(x,y) & h = 0, \end{cases} \tag{6.24}$$

und $z(x)$ die exakte Lösung der Differentialgleichung ist.

Durch Einsetzen in die entsprechenden Ausdrücke sieht man, dass das Euler-Verfahren konsistent in 1. Ordnung ist, während das Verfahren von Heun und das modifizierte Eulerverfahren konsistent in 2. Ordnung sind.

Neben der Konsistenz eines Verfahrens gibt es noch den Begriff der Konvergenz. Ein Einschrittverfahren heißt an der Stelle $x\epsilon[x_a, x_e]$ *konvergent*, wenn

$$\lim_{h\to 0}(y_i^h) - z(x)) = 0 \quad \text{für} \quad x_i = x\epsilon[x_a, x_e]\,, \tag{6.25}$$

wobei y_i^h die numerische Lösung an der Stelle x_i und $z(x = x_i)$ die exakte Lösung der Differentialgleichung ist. Das Einschrittverfahren heißt konvergent zur Ordnung $p > 0$, wenn

$$y_i^h - z(x) = \mathcal{O}(h^p) \quad \text{für} \quad h \to 0, j = 0, 1, \ldots, N \tag{6.26}$$

gilt.

Bei Einschrittverfahren läßt sich leicht ein Zusammenhang zwischen Konsistenz- und Konvergenzordnung herstellen. Es gilt folgender Satz:

Für eine Einzeldifferentialgleichung $y' = f(x,y)$ sei ein Einschrittverfahren $\Phi(x,y;h)$ mit folgenden Eigenschaften gegeben:

- $\Phi(x,y;h)$ ist bezüglich aller Veränderlicher $(x,y;h)$ stetig für $x_a \leq x \leq x_e$, $-\infty < y < +\infty$, $0 \leq h \leq h_0$, wobei h_0 hinreichend klein ist,
- Das Verfahren sei konsistent von der Ordnung $p > 0$,
- Die Funktion $\Phi(x,y;h)$ erfülle bezüglich y eine Lipschitzbedingung: "Es gibt eine Konstante $L > 0$ sodass $|f(x) - f(x+h)| < L|h|$".

Für diese so gegebene Einschrittverfahren gilt, dass es konvergent von der Ordnung p.

6.3.3 Runge-Kuttaverfahren

Die Standardmethode

Eine der beliebtesten Methoden zur Integration von gewöhnlichen Differentialgleichungen ist das *Verfahren von Runge-Kutta*. Dieses Verfahren wurde 1985 erstmals formuliert und basiert auf einen allgmeinen Ansatz für die Funktion $\Phi(x,y;h)$. In der Standardform ist das Runge-Kutta-Verfahren für eine Funktion $y' = f(x,y)$ durch folgende Funktion $\Phi(x,y;h)$ gegeben:

Die Standardform des Runge-Kutta-Verfahrens

$$\Phi(x,y;h) = \tfrac{1}{6}\left[k_1 + 2k_2 + 2k_3 + k_4\right] + O(h^5)$$

$$k_1 = f(x,y) \qquad k_2 = f(x+\tfrac{h}{2}, y+\tfrac{h}{2}k_1)$$
$$k_3 = f(x+\tfrac{h}{2}, y+\tfrac{h}{2}k_2) \qquad k_4 = f(x+h, y+h\ k_3)$$

Entwickelt man $\Phi(x,y;h)$ in eine Taylorreihe nach h, so findet man, dass die oben angegebene Form des Runge-Kutta-Verfahrens konisistent in vierter Ordnung ist. Der Beweis der Konsistenzordnung ist trivial, erfordert aber eine längere Rechnung.

Das Runge-Kutta-Verfahren ist einfach, effizient und bestens geeignet, wenn an die Genauigkeit der Lösungen nicht allzu große Anforderungen gestellt werden. Für hohe Anforderungen an die Genauigkeit, sollte man auf eine *Predictor-Korrektor-Methode* übergehen [7]. Das Runge-Kutta-Verfahren lässt sich auch auf Systeme von Differentialgleichungen erster Ordnung in analoger Weise anwenden. Die Funktionen $\Phi(x,y;h), k_1, k_2, k_3$ und k_4 sowie die Variable y erhalten dann Vektorcharakter (siehe 6.2). Die Programmierung wird besonders einfach, wenn man auf ein autonomes Gleichungssystem übergeht, d.h. man definiert die unabhängige Variable x als die nullte Komponente des Lösungsvektors y (siehe Gleichung (6.8).

Runge-Kutta-Methode mit angepasster Schrittweite

Ein guter Algorithmus für die Integration gewöhnlicher Differentialgleichungen sollte die Güte der Lösung laufend kontrollieren und entsprechende Adaptierung der Parameter selbst vornehmen, um eine bestimmte vorgegebene Genauigkeit zu garantieren. In Einschrittverfahren lässt sich die Genauigkeit über die Schrittweite kontrollieren. In Bereichen starker Änderung der Lösung $y(x)$ wird man zur Erreichung einer bestimmten Genauigkeit viele kleine Schritte benötigen, während man in Bereichen eines weitgehend glatten Funktionsverlaufes von $y(x)$ mit wenigen großen Schritten das Auslangen findet. Sieht man eine entsprechende Anpassung der Schrittweite im Algorithmus vor, so kann man die Effizienz des Algorithmus wesentlich steigern (je nach Funktion $y(x)$ sind Effizienzsteigerungen von 100 und mehr möglich).

Die Festlegung einer der jeweiligen Lösung angepassten Schrittweite erfordert die Bereitstellung einer Information über die Güte der Lösung bei jedem Integrationsschritt. Bei der Standard Runge-Kutta-Methode (konsistent in vierter Ordnung) erreicht man dies am einfachsten durch Verdopplung der Schritte, d.h. man rechnet jeden Schritt zweimal, einmal in einem Schritt und einmal in zwei Schritten,

$$y(x+2h) = y_1(x) + (2h)^5\phi + O(h^6) + \cdots, \tag{6.27}$$
$$y(x+2h) = y_2(x) + 2h^5\phi + O(h^6) + \cdots. \tag{6.28}$$

Die exakte Lösung wurde mit $y(x)$, bezeichnet, die numerischen Lösungen, y_1, y_2, wurden mit einer Schrittweite $2h$ bzw. h gerechnet. Insgesamt erfordert jeder Schritt 11

Aufrufe der Funktion $f(x, y)$. Dies muss man vergleichen mit 8 Aufrufen, wenn man mit der Schrittweite h integriert. Die Rechenzeit erhöht sich zwar um den Faktor $1,375$, aber der Algorithmus mit Schrittverdopplung liefert einen Parameter, $\Delta = y_2 - y_1$, der die Güte der Lösung harakterisiert. Die Forderung, dass Δ kleiner als ein vorgegebener Genauigkeitsparameter `acc` ist, erlaubt eine automatisierte Schrittweitenanpassung des Runge-Kutta-Verfahrens. Erfüllt die gewählte Schrittweite das Kriterium $\Delta <$`acc` nicht, so wird die Schrittweite verkleinert und die Integration noch einmal ausgeführt. Ist das Kriterium $\Delta <$`acc` erfüllt, so kann y_2 als Wert der Lösung verwendet werden und man führt den nächsten Integrationsschritt aus. Alternativ lässt sich auch ein korrigierter Wert der Lösung angeben,

$$\Delta = y_2 - y_1 = 30h^5\phi + O(h^6)\,, \tag{6.29}$$

$$y^{(corr)} = y_2(x) + \frac{1}{15}\Delta + O(h^6)\,. \tag{6.30}$$

Die neue Abschätzung ist nun fünfter Ordnung und wird allgemein als *lokale Approximation* bezeichnet. Man kann nun mit diesem Wert die Lösung fortsetzen.

Eine alternative Konstruktion eines Differentialgleichungsintegrators mit automatisierter Schrittweitenanpassung beruht auf den von Fehlberg abgeleiteten *Eingebetteten Runge-Kutta-Formeln*. Fehlberg entdeckte eine Methode fünfter Ordnung, welche sechs Aufrufe von $f(x, y)$ erfordert[1],

$$k_i = hf(x_n + a_i h, y_n + \sum_{j=1}^{i-1} b_{ij}k_j) \quad j = 1, \ldots, 6\,, \tag{6.31}$$

$$y_{n+1} = y_n + \sum_{i=1}^{6} c_i k_i + O(h^6)\,. \tag{6.32}$$

Die Koeffizienten a_i, b_{ij}, c_i sind in Tab. 6.1 gegeben. Eine andere Kombination der gleichen sechs Aufrufe liefert ein Verfahren vierter Ordnung,

$$\tilde{y}_{n+1} = y_n + \sum_{j=1}^{6} \tilde{c}_j k_j + O(h^5)\,, \tag{6.33}$$

Durch Differenzbildung erhält man wieder ein Kriterium für die Güte und damit für die Schrittweitenanpassung,

$$\Delta = y_{n+1} - \tilde{y}_{n+1} = \sum_{j=1}^{6} (c_j - \tilde{c}_j)\ k_j\,. \tag{6.34}$$

Abweichend von den Koeffizienten von Fehlberg, haben sich die von Cash und Karp [8] angegebenen Werte (siehe Tab. 6.1) im allgemeinen bewährt.

[1] Allgemein lässt sich sagen, dass ein Runge-Kutta-Verfahren M. Ordnung zumindest M Aufrufe von $f(x, y)$, aber niemals mehr als $M + 2$ Aufrufe erfordert.

i	a_i	b_{ij}					c_i	$\tilde{c}_i$
1	0						$\frac{37}{378}$	$\frac{2825}{27648}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0	0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$	$\frac{18575}{48384}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$	$\frac{13525}{55296}$
5	1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		0	$\frac{277}{14336}$
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$	$\frac{1}{4}$

Tabelle 6.1: Parameter von Cash und Karp [8] für die eingebettete Runge-Kutta-Methode fünfter Ordnung.

6.4 Rundungsfehler

Die Abschätzung der Verläßlichkeit numerischer Lösungen ist für die Beurteilung der Güte eines Resultates unerläßlich. Bei numerischen Rechnungen lassen sich verschiedene Klassen von Fehlern unterscheiden:

- *Unsicherheit in den Eingabedaten:* Diese können im allgemeinen in der numerischen Rechnung nicht beeinflußt werden. Man muss jedoch die Stabilität bezüglich solcher Unsicherheiten diskutieren.

- *Abbrechfehler:* Darunter versteht man Fehler durch Vernachlässigung von Resttermen bzw. nicht vollständig durchgeführte Iterationen. Solche Fehler lassen sich über den verwendeten Algorithmus steuern.

- *Rundungsfehler:* Diese werde durch die Abbildung der numerischen Werte auf die Menge der Maschinenzahlen verursacht. Durch Auslöschungseffekte kann es zu einer unerwünschten Verstärkung dieser Fehler kommen.

Im Unterkapitel 6.3.2 haben wir die Konvergenz der Verfahren betrachtet. Im nächsten Schritt wollen wir die Gesamtheit aller Fehler, und zwar Rundungs- und Verfahrensfehler gemeinsam betrachten. Sei $z(x)$ die exakte Lösung und η_i die numerische Lösung an der Stelle x_i, dann gelten die Beziehungen

$$\text{Numerische Lösung:} \qquad \tilde{\eta}_{i+1} = \tilde{\eta}_i + h\Phi(x_i, \tilde{\eta}_i; h) + \varepsilon_{i+1}\,, \tag{6.35}$$

$$\text{Exakte Lösung:} \qquad z(x_{i+1}) = z(x_i) + h\Phi(x_i, z_i; h) + \mathcal{O}(h^{p+1})\,, \tag{6.36}$$

wobei ε_{i+1} der gesamte Rundungsfehler für die Integration von x_i nach x_{i+1} ist. Subtraktion der beiden Gleichungen liefert,

$$\begin{aligned} r_{i+1} &= \tilde{\eta}_{i+1} - z(x_{i+1}) \\ &= \tilde{\eta}_i - z(x_i) + h\left[\Phi(x_i, \tilde{\eta}_i; h) - \Phi(x_i, z(x_i); h)\right]\varepsilon_{i+1} + \mathcal{O}(h^{p+1}). \end{aligned} \tag{6.37}$$

Die Größe des Gesamtfehlers lässt sich unter der Annahme einer Lipschitz Bedingung für Φ abschätzen

$$|r_{i+1}| \leq |r_i| + h\ L|r_i||\varepsilon_{i+1}| + Mh^{p+1}\,. \tag{6.38}$$

Lösung dieser Differenzengleichung liefert die Abschätzung

$$|r_i| \leq \left[Mh^p + \frac{|\varepsilon_i|}{h}\right] \frac{e^{iM(x_e - x_a)} - 1}{L}\,. \tag{6.39}$$

Diese Abschätzung ist für praktische Anwendungen zu grob und hat daher nur qualitativen Wert. Sie zeigt aber das wesentliche Verhalten der Abhängigkeit des Gesamtfehlers von der Schrittweite. Es gibt einen optimalen Wert für die Schrittweite h. Wird h kleiner, so steigt aufgrund der Rundungsfehler der Gesamtfehler an (Abb. 6.1).

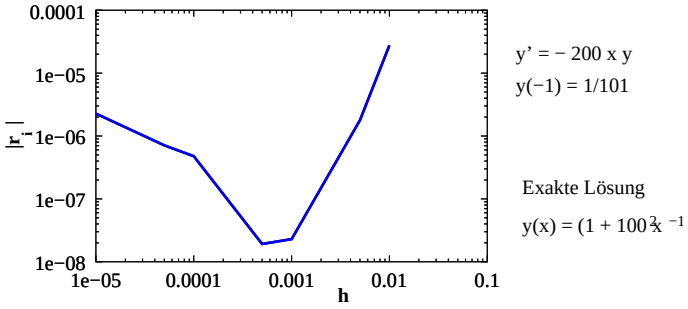

Abbildung 6.1: Beispiel für den Gesamtfehler bei der numerischen Lösung einer Differentialgleichung.

6.5 Numerov Verfahren

Viele Fragestellungen der Quantenphysik und der Elektrodynamik lassen sich auf eine gewöhnliche Differentialgleichung zweiter Ordnung der Form

$$\left[\frac{d^2}{dx^2} + w(x)\right] y(x) = S(x) \tag{6.40}$$

zurückführen. Dabei ist $w(x)$ eine Funktion, welche die Stärke des *Response* angibt und $S(x)$ stellt einen Quellterm dar.

Die numerische Lösung der Differentialgleichung (6.40) kann im Prinzip durch ein Einschrittverfahren, z.B. das Runge-Kutta-Verfahren, erfolgen. Diese Vorgangsweise ist zweckmäßig, falls neben der Funktion $y(x)$ auch die Ableitung dy/dx in der weiteren Rechnung benötigt wird. In den meisten Fällen geht allerdings die erste Ableitung nicht explizit ein.

Eine sehr einfache Methode zur numerischen Lösung von Differentialgleichungen von Typ (6.40) geht auf Numerov [9] zurück. Man bezeichnet den Algorithmus auch als *Cowling Methode* oder *Fox-Godwin-Verfahren* [10]. Wir definieren nun ein äquidistantes

Gitter $\{x_n = x_0 + nh\}, n = 0, 1, 2, \ldots$ mit der Schrittweite h auf dem Definitionsbereich und entwickeln die Lösung $y(x)$ an der Stelle $y_n = y(x_n)$ in eine Taylor-Reihe. Man erhält somit für

$$y_{n\pm1} = y_n \pm h\, y'_n + \frac{h^2}{2}\, y''_n \pm \frac{h^3}{6}\, y'''_n + \frac{h^4}{24}\, y_n^{(\mathrm{iv})} \pm \frac{h^5}{120}\, y_n^{(\mathrm{v})} + O(h^6) \tag{6.41}$$

Aus Gleichung (6.41) ergibt sich der Ausdruck

$$\frac{y_{n+1} - 2\, y_n + y_{n-1}}{h^2} = y''_n + \frac{h^2}{12}\, y_n^{(\mathrm{iv})} + O(h^4)\,. \tag{6.42}$$

Die zweite Ableitung y''_n kann also als Differenzenausdruck plus einem Korrekturterm dargestellt werden, welcher die vierte Ableitung der Funktion enthält. Der charakteristische Schritt des Numerov-Verfahrens besteht nun in der Darstellung der vierten Ableitung durch die Differentialgleichung (6.40),

$$y^{(\mathrm{iv})} = \frac{d^2}{dx^2} y'' = \frac{d^2}{dx^2}\left[-w(x)y(x) + S(x)\right]\,. \tag{6.43}$$

Damit lässt sich der Korrekturterm wieder durch die rechte Seite der Differentialgleichung ausdrücken, sodass wir nun folgenden Differenzenausdruck für die Lösung der Differentialgleichung erhalten,

$$\begin{aligned} y''_n &= \frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} - \frac{h^2}{12} y_n^{(iv)} + O(h^4) = \frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} \\ &+ \frac{h^2}{12}\left[\frac{w_{n+1}y_{n+1} - 2w_n y_n + w_{n-1}y_{n-1}}{h^2} - \frac{S_{n+1} - 2S_n + S_{n-1}}{h^2} + O(h^2)\right] \\ &= -w_n y_n + S_n + O(h^4)\,. \end{aligned} \tag{6.44}$$

Ordnet man nun die Terme, so ergibt sich die Grundgleichung für die Lösung der Differentialgleichung (6.40) im Numerov-Verfahren

$$\begin{aligned} &y_{n+1}\left[1 + \frac{h^2}{12} w_{n+1}\right] - y_n\left[2 - \frac{10h^2}{12} w_n\right] + y_{n-1}\left[1 + \frac{h^2}{12} w_{n-1}\right] \\ &\quad = \frac{h^2}{12}\left[S_{n+1} + 10S_n + S_{n-1}\right] + O(h^6)\,. \end{aligned} \tag{6.45}$$

Das Numerov-Verfahren ist konsistent und konvergent in vierter Ordnung und eignet sich bestens für die Lösung von Anfangswertproblemen.

Für den Start des Verfahrens sollte die Funktion y(x) an zwei nebeneinanderliegenden Stützstellen bekannt sein. Dies entspricht auch den erforderlichen zwei Anfangsbedingungen für die eindeutige Lösung gewöhnlicher Differentialgleichung zweiter Ordnung. So kann z.B. die Kenntnis von $y(x)$ und $y'(x)$ an einer Stützstelle $\bar{x}$ durch einen linearisierten Zusammenhang zur Berechnung der Funktion an zwei Stützstellen ausgenützt werden. $y_j = y(\bar{x}) + (x_j - \bar{x})y'(\bar{x})$ für $j = n, n+1$.

Für die praktische Rechnung ist es zweckmäßig die Größe $Q(r)$ zu definieren,

$$Q_n = \left[1 + \frac{h^2}{12} w_n\right] y_n \,. \tag{6.46}$$

Die Rekursionsformel (6.45) reduziert sich dann auf die einfache Form

$$Q_{n+1} + 10Q_n + Q_{n-1} - 12y_n = \frac{h^2}{12} \left[S_{n+1} + 10S_n + S_{n-1}\right] + O(h^6) \,, \tag{6.47}$$

welche eine Rekursionsvorschrift für die Hilfsgröße $Q(r)$ darstellt. Bei der praktischen Anwendung muss man allerdings beachten, dass die Funktion $Q(x)$ auch bei verschwindendem Funktionswert der Lösung $y(x)$ einen endlichen Wert aufweisen kann, falls $w(x)$ eine Singularität an dieser Stelle hat.

Literaturverzeichnis

[1] M. Abramowitz and I.A. Stegun, *Handbook of Mathematical Functions* (National Bureau of Standards, Washington, 1972).

[2] Gillman und Fiebig, Computers in Physics, Jän/Feb (1988) 62.

[3] C.F. Gerald und P.O. Wheatley, *Applied Numerical Analysis* (Addison-Wesley, Reading, 1989).

[4] C. Überhuber, *Computernumerik I und II* (Springer, 1995)

[5] http://www.cs.sunysb.edu/∼algorith/files/random-numbers.shtml (Downloads, Hinweise)

[6] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes*, (Cambridge University Press, 2007); http://www.nrbook.com

[7] J. Stoer, *Einführung in die Numerische Mathematik I* (Springer, Berlin-Heidelberg, 1972); *Numerische Mathematik II* (Springer, Heidelberg, 1980).

[8] J.R. Cash, A.H. Karp, ACM Trans. Math. Software **16**, 201 (1990).

[9] B. Numerov, Publs.observatoire central astrophys.Russ., **2** (1933) 188.

[10] L. Fox, E. T. Goodwin, Proc. Camb. Phil. Soc. **45** (1949) 373.