A Interpolation

Von einer Funktion ist oft nur eine gewisse Anzahl von Funktionswerten bekannt. Zum Beispiel bei der Konstruktion eines Tragflügels eines Flugzeuges hat man für das Tragflügelprofil die Werte in Tab. A.1 zur Verfügung.

Möchte man eine Konstruktionszeichnung machen, dann benötigt man eine kontinuierliche Kurve, welche durch die entsprechenden Punkte geht. Ganz allgemein besteht die **Interpolation** darin, eine einfache Funktion zu bestimmen, die eine gewisse Anzahl von vorgegebenen Werten annimmt und mit der dann beliebige Zwischenwerte berechnet werden können. Dies spielt in CAD-Systemen eine große Rolle. Eine andere Anwendung ist die Darstellung von Ergebnissen aus numerischen Simulationen, welche nur an den diskreten Gitterpunkten vorliegen. Die Interpolation wird auch zur Reduktion großer Datenmengen eingesetzt, wie sie etwa bei der Bildverarbeitung auftreten. Um die Daten zu reduzieren, werden nicht alle Farbwerte eines Bildes abgespeichert. Hat man eine große Fläche mit geringen Farbkontrasten, kann man die Farbwerte an einigen wenigen Punkten abspeichern und bei Bedarf die Werte an den Zwischenpunkten durch Interpolation wieder erzeugen. Ein Computerspiel kommt ohne Interpolation nicht aus.

Tab. A.1 Stützstellen für die Interpolation

NACA 1412			
Oberseite		Unterseite	
x	y	x	y
0	0	0	0
1,158	1,954	1,342	−1,830
2,378	2,733	2,622	−2,491
4,845	3,786	5,155	−3,318
7,330	4,537	7,670	−3,857
14,833	5,951	15,617	−4,733
40,000	6,803	40,000	−4,803
60,051	5,453	59,949	−3,675
80,058	3,178	79,942	−2,066
100,000	0,126	100,000	0,126

© Springer-Verlag Berlin Heidelberg 2019

C.-D. Munz, T. Westermann, *Numerische Behandlung gewöhnlicher und partieller Differenzialgleichungen*, https://doi.org/10.1007/978-3-662-55886-7

Eng verwandt mit der Interpolation ist **Approximation** einer Funktion. Es wird hier eine einfache Funktion gesucht, welche eine vorgegebene im Allgemeinen komplizierte Funktion approximiert. Dies wird zum Beispiel dann angewandt, wenn die Auswertung einer vorgegebenen Funktion sehr rechenaufwändig ist. Die Approximation ist aber auch wichtig zur Ableitung von Näherungsverfahren, da sie meist einfach integriert und differenziert werden kann. Es können dadurch Verfahren zur numerischen Differenziation und Integration abgeleitet werden.

Eine der Kandidaten von einfachen Funktionen für die Interpolation oder Approximation ist die Klasse der **Polynome**. Integration oder Differenziation liefern durch einfache Rechnung wieder ein Polynom. Ein Polynom n-ten Grades schreibt man meist in der Form

$$p_n(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + \ldots + a_n x^n \,. \tag{A.1}$$

Von diesem Polynom n-ten Grades mit den $(n+1)$ Koeffizienten a_i, $i = 0, 1, \ldots, n$, können wir fordern, dass es $n+1$ vorgegebenen Werte annimmt:

$$p_n(x_i) = f_i, \tag{A.2}$$

$$i = 0, 1, \ldots, n \,. \tag{A.3}$$

Man nennt die x_i die **Stützstellen** und die f_i die zugehörigen **Stützwerte**, beides zusammen (x_i, f_i) sind die **Stützpaare**.

Bei der Approximation einer Funktion $f = f(x)$ ergeben sich die Stützwerte als Funktionswerte an den Stützstellen $f_i = f(x_i)$. Bei der Interpolation sind sie als Datenwerte wie in Tabelle A.1 vorgegeben.

Setzen wir die Punkte (x_i, f_i) in das Polynom p_n ein:

$$p_n(x_i) = f_i = a_0 + a_1 x_i + \cdots + a_n x_i^n \,, \tag{A.4}$$

$$i = 0, 1, \ldots, n \,, \tag{A.5}$$

so erhalten wir ein System von $(n+1)$ linearen Gleichungen zur Berechnung der Koeffizienten $a_0, a_1, \ldots, a_n$. Sind die Stützstellen x_i sämtlich verschieden, dann hat dieses Gleichungssystem für jeden Satz von Stützwerten (A.2) eine eindeutige Lösung.

Da die Konstruktion des Interpolationspolynoms über das Lösen eines linearen Gleichungssystems aufwändig und für praktische Zwecke ungünstig ist, wurden in der Mathematik schon früh andere Konstruktionsmethoden entwickelt. Wir betrachten im Folgenden zwei dieser Verfahren. Da das Polynom durch die Vorgabe der Stützwerte eindeutig bestimmt ist, führen alle Konstruktionsverfahren auf das gleiche Polynom. Die unterschiedlichen Darstellungen des Polynoms haben jedoch unterschiedliche Eigenschaften und Anwendungen. Ob eine Interpolation oder die Approximation einer Funktion ausgeführt wird, unterscheidet sich in den Formeln zunächst nicht. Immer dann, wenn der Fehler oder die Güte des Verfahrens betrachtet wird, dann brauchen wir einen Vergleich. Insofern

liegt dann zwingend eine Approximationsaufgabe zu Grunde mit einer approximierten Funktion, welche genügend oft stetig differenzierbar ist, so dass die vorkommenden Ableitungen existieren und das Ganze Sinn macht.

A.1 Die Interpolationsformel von Lagrange

Die Funktion

$$L_i^n(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} \tag{A.6}$$

ist ein Polynom vom Grad n mit den Eigenschaften

$$L_i^n(x_j) = \begin{cases} 1 \text{ für } i = j \ , \\ 0 \text{ für } i \neq j \end{cases} \tag{A.7}$$

und

$$\sum_{i=0}^{n} L_i^n(x) = 1 \ . \tag{A.8}$$

Mit diesen so genannten **Lagrangeschen Stützpolynomen** (A.6) kann man ein Polynom n-ten Grades p_n angeben, welches durch die vorgegebenen Punkte (A.2) verläuft. Es hat die Form

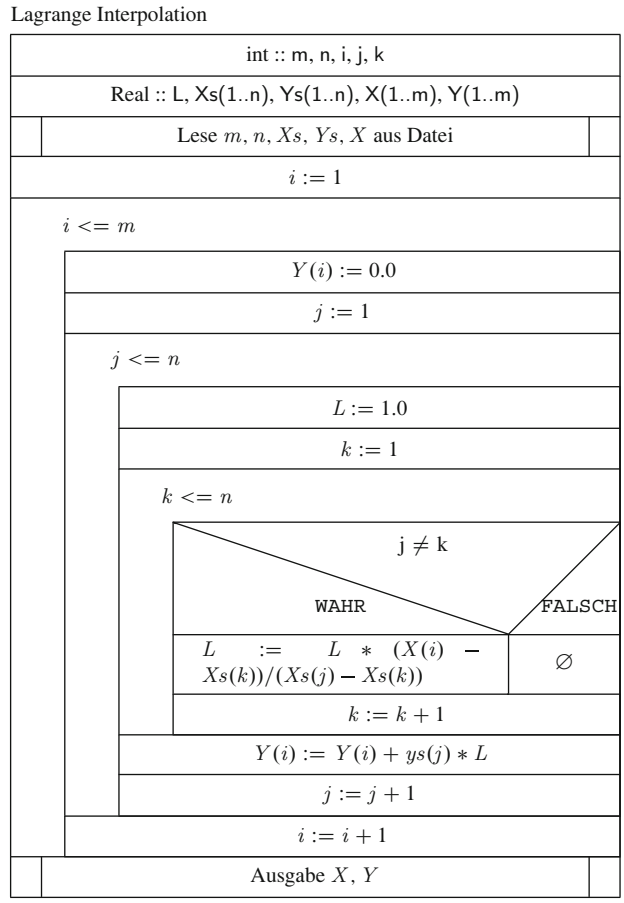
$$p_n(x) = \sum_{i=0}^{n} f_i L_i^n(x) \ . \tag{A.9}$$

Aus der Eigenschaft (A.7) ergibt sich sofort, dass die Werte f_i an den Stützstellen x_i für $i = 0, 1, \ldots, n$ angenommen werden. Ein Struktogramm für die Lagrange Interpolation ist in Abb. A.1 angegeben.

Wird durch das Lagrangesche Interpolationspolynom n-ten Grades eine gegebene $(n+1)$-mal stetig differenzierbare Funktion $y = f(x)$ mit $x \in [a, b]$ approximiert, dann ergibt sich die Fehlerformel

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n) \tag{A.10}$$

Abb. A.1 Struktogramm für die Lagrange Interpolation



mit einer Zwischenstelle $\xi \in (a, b)$, welche von dem Punkt x abhängt. Für praktische Fehlerabschätzungen ist die Formel so nicht sehr brauchbar, da die Zwischenstelle ξ nicht bekannt ist. Man muss die $(n + 1)$-te Ableitung von f nach oben und unten abschätzen, um eine konkrete Fehlerschranke zu bekommen.

Beispiel A.1 (Lagrange Interpolation, mit Script-File)

Wir schauen uns die einfachen Fälle $n = 1$ und $n = 2$ genauer an. Im **Falle** $n = 1$ sind zwei Wertepaare

$$(x_0, f_0) \quad \text{und} \quad (x_1, f_1) \tag{A.11}$$

vorgegeben. Die beiden Lagrangeschen Stützpolynome haben nach der Formel (A.6) die Gestalt:

$$L_0^{n=1}(x) = \frac{(x - x_1)}{(x_0 - x_1)} \quad \text{und} \quad L_1^{n=1}(x) = \frac{(x - x_0)}{(x_1 - x_0)} \,. \tag{A.12}$$

Mit Hilfe der Lagrange-Interpolationsformel ergibt sich das Polynom vom Grad $n = 1$, das heißt die Gerade, in der Gestalt

$$p_{n=1}(x) = f_0 L_0^{n=1}(x) + f_1 L_1^{n=1}(x) = f_0 \frac{(x - x_1)}{(x_0 - x_1)} + f_1 \frac{(x - x_0)}{(x_1 - x_0)} \,. \tag{A.13}$$

Im **Falle** $n = 2$ müssen drei Wertepaare vorgegeben sein:

$$(x_0, f_0),\ (x_1, f_1) \quad \text{und} \quad (f_2, f_2) \,. \tag{A.14}$$

Die Lagrangeschen Stützpolynome haben dann nach der Formel (A.6) die Gestalt:

$$L_0^{n=2}(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \,, \tag{A.15}$$

$$L_1^{n=2}(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \,, \tag{A.16}$$

$$L_2^{n=2}(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \,. \tag{A.17}$$

Mit Hilfe der Lagrangeschen Interpolationsformel (A.9) ergibt sich das Polynom vom Grad $n = 2$ zu

$$p_{n=2}(x) = f_0\, L_0^{n=2}(x) + f_1\, L_1^{n=2}(x) + f_2\, L_2^{n=2}(x) \,. \tag{A.18}$$

Das Polynom ist eine Parabel. □

Beispiel A.2 (Lagrange Interpolation, mit Script-File)

Mit Hilfe der Lagrangeschen Interpolationsformel werden Polynome vom Grad 1, 2, 3, 4 und 5 berechnet, welche die Funktion

$$y = e^x \tag{A.19}$$

im Intervall $[0, 2]$ approximieren. Die Stützstellen werden äquidistant im vorgegebenen Intervall gewählt. Abb. A.2 zeigt im Vergleich mit der exakten Lösung das Schaubild des Polynoms vom Grad 1 bis 4.

Bei dieser einfachen Funktion, die sich auch noch in einfacher Weise differenzieren lässt, kann man den Fehler mit wenig Aufwand bestimmen. Setzen wir für das Polynom 5. Ordnung die Ableitung $f^{(6)}(x) = e^x$ in die Fehlerformel (A.10) ein, so erhalten wir

$$f(x) - p_{n=5}(x) = \frac{e^\xi}{6!}(x - x_0)(x - x_1)\cdots(x - x_5) \,. \tag{A.20}$$

Diesen Ausdruck kann man leicht nach oben abschätzen. Da die Exponentialfunktion monoton wachsend ist, ergibt sich der Maximalwert zu e^2 und man erhält

$$|f(x) - p_{n=5}(x)| \le \frac{e^2}{6!}|(x - x_0)(x - x_1)\cdots(x - x_5)| \,. \tag{A.21}$$

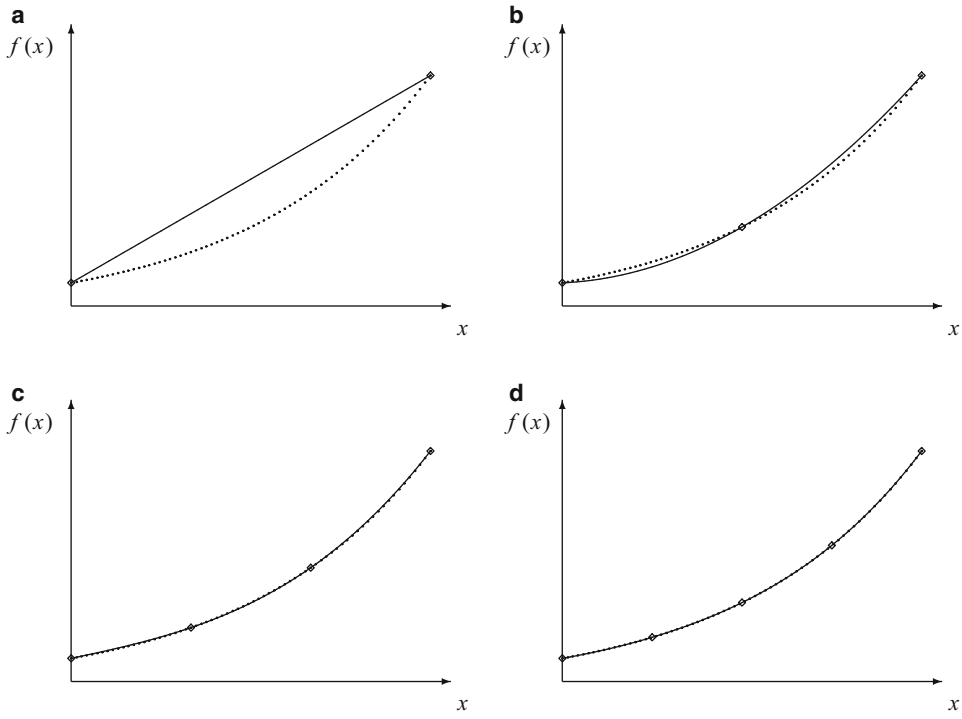


Abb. A.2 Approximation der e-Funktion durch Polynome vom Grad 2 bis 5

Trägt man dies mit MAPLE als Funktion von x auf, sieht man, dass sich der Fehler zwischen -0.0008 und 0.0003 bewegt. □

A.2 Die Interpolationsformel von Newton

Eine andere Möglichkeit der Darstellung des Interpolationspolynoms ist die **Interpolationsformel von Newton**. Dabei werden die sogenannten **Newtonschen Stützpolynome**

$$N_0(x) := 1, N_j(x) := (x - x_0)(x - x_1)\cdots(x - x_{j-1}), \quad j = 1, 2, \ldots, n \quad \text{(A.22)}$$

benutzt. Das zugehörige Interpolationspolynom ergibt sich aus der Formel

$$\begin{aligned} p_n(x) &= \sum_{i=0}^{n} c_i N_i(x) \\ &= c_0 + c_1(x - x_0) + \cdots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \,. \end{aligned} \quad \text{(A.23)}$$

Beispiel A.3

Wir betrachten zunächst den **Fall n = 1**. Das Interpolationspolynom in der Newtonschen Darstellung ergibt sich zu

$$p_1(x) = c_0 + c_1(x - x_0) \; . \tag{A.24}$$

Um die Stützwerte f_0 und f_1 an den Stützstellen x_0, x_1 anzunehmen, muss $c_0 = f_0$ gelten, da der zweite Summand auf der rechten Seite bei $x = x_0$ verschwindet. Man erhält c_1, indem man $x = x_1$ einsetzt und nach c_1 auflöst. Es ergibt sich somit das Polynom ersten Grades

$$p_1(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x - x_0) \; . \tag{A.25}$$

Dies ist nichts anderes als die Zweipunkteform einer Geraden.

Im **Fall n = 2** erhöht man den Grad des Interpolationspolynoms nach (A.23) und nimmt eine neue Stützstelle x_2 hinzu, dann bleiben die Koeffizienten c_0 und c_1 erhalten und es kommt der Koeffizient c_2 hinzu. Berechnet man diesen aus der Bedingung $p_2(x_2) = f_2$, so hat das Interpolationspolynom die Gestalt

$$\begin{aligned} p_2(x) &= p_1(x) \\ &\quad + \left(\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0} \right) \frac{1}{x_2 - x_0}(x - x_0)(x - x_1) \; . \end{aligned} \tag{A.26}$$

Es zeigt sich, dass durch die Hinzunahme einer neuen Stützstelle zu dem Polynom p_1 nur ein neuer Term addiert wird. Die schon berechneten Koeffizienten c_0 und c_1 bleiben erhalten. Das ist ein großer Vorteil der Newtonschen Interpolationsformel. Bei der Lagrangeschen Formel müssen bei Hinzunahme eines neuen Stützpaares alle Koeffizienten neu berechnet werden. □

Wie man schon in (A.26) sieht, werden die Koeffizienten mit wachsendem Grad der Stützpolynome immer aufwändiger zu berechnen. Es gibt allerdings eine effiziente rekursive Berechnung. Der Koeffizient c_1, wir er in (A.25) ausgeschrieben ist, entspricht gerade der Steigung der Sekante durch die Punkte (x_0, f_0) und (x_1, f_1). Man bezeichnet diesen Wert als die **dividierte Differenz** bezüglich x_0 und x_1 und schreibt

$$f[x_0, x_1] := \frac{f_1 - f_0}{x_1 - x_0} \; .$$

Die zweite dividierte Differenz bezüglich x_0, x_1 und x_2 ist dann gegeben durch

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

Tab. A.2 Dividierte Differenzen

$f(x)$	1. dividierte Differenz	2. dividierte Differenz	3. dividierte Differenz
$f[x_0] = c_0$			
	$f[x_0, x_1] = c_1$		
$f[x_1]$		$f[x_0, x_1, x_2] = c_2$	
	$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3] = c_3$
$f[x_2]$		$f[x_1, x_2, x_3]$	
	$f[x_2, x_3]$		$f[x_1, x_2, x_3, x_4]$
$f[x_3]$		$f[x_2, x_3, x_4]$	
$\vdots$	$f[x_3, x_4]$		$f[x_2, x_3, x_4, x_5]$
	$\vdots$		

und gerade identisch mit dem Koeffizienten c_2 des Interpolationspolynoms (A.26). Definieren wir allgemein

$$\begin{aligned} f[x_i] &:= f(x_i)\ , \\ f[x_i, x_{i+1}] &:= \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}\ , \\ f[x_i, x_{i+1}, \ldots, x_{i+k}] &:= \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}\ , \end{aligned} \tag{A.27}$$

dann können die Koeffizienten der Newtonschen Interpolationsformel aus einer **Tabelle der dividierten Differenzen** abgelesen werden. In Tab. A.2 sind die dividierten Differenzen der verschiedenen Ordnungen spaltenweise aufgeschrieben. Jede neue Spalte wird aus der vorherigen nach (A.27) entsprechend berechnet. Die Koeffizienten der Newtonschen Interpolationsformel ergeben sich direkt als die Werte der oberen Schrägzeile.

Ein Rechenprogramm mit der Interpolationsformel von Newton läuft in zwei Schritten ab. Im ersten wird die Tabelle mit den für das Polynom benötigten dividierten Differenzen erzeugt und die Koeffizienten c_j berechnet. Für die Auswertung wird das Polynom noch etwas umgeschrieben. Hier wieder das Beispiel $n = 2$:

$$\begin{aligned} p_2 &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) \\ &= c_0 + (x - x_0)(c_1 + c_2(x - x_1))\ . \end{aligned}$$

Die gemeinsamen Faktoren werden nach vorne vor die Klammern gezogen. Die Klammern werden dann von Innen nach Außen ausgewertet, um so die Anzahl der Rechenoperationen so klein wie möglich zu halten.

Für die Lagrangesche Interpolationsformel haben wir eine Fehlerabschätzung angegeben. Wie schon erwähnt, ist diese in der Praxis nur selten nützlich. Bei der Interpolation ist die Funktion f im Allgemeinen nicht gegeben, sondern liegt nur in Form einer

Wertetabelle vor. Zum Anderen muss die $(n+1)$-te Ableitung berechnet und an einer Zwischenstelle ausgewertet werden, die nicht bekannt ist. Die Berechnung von Maximum und Minimum der $(n+1)$-ten Ableitung ist meist aufwändig und der berechnete maximale Fehler kann zudem eine schlechte Abschätzung für den tatsächlichen Fehler liefern.

In Beispiel A.2 haben wir Werte der e-Funktion interpoliert und konnten dort den Fehler abschätzen, da die Ableitungen der e-Funktion sehr einfach sind.

Bei der Newtonschen Interpolationsformel gibt es jedoch eine einfache Möglichkeit, zumindest eine Näherung für den Fehler zu erhalten, falls man ein zusätzliches Wertepaar (x_{n+1}, f_{n+1}) zur Verfügung hat. Wir betrachten hier den Fall der Approximation und nehmen an, dass $f = f(x)$ eine mindestens $(n+1)$-mal stetig differenzierbare Funktion ist. Mit dem Restglied $R_n = R_n(x)$ gilt

$$f(x) = p_n(x) + R_n(x) \ ,$$

wobei $p_n(x)$ das Interpolationspolynom vom Grad n ist. Dabei lässt sich $R_n(x)$ in der Form

$$R_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n) f[x, x_0, x_1, \ldots, x_n]$$

schreiben. Dies lässt sich nicht ausrechnen, da das unbekannte $f(x)$ auf der rechten Seite auftritt. Die Idee ist nun, zur Approximation dieser dividierten Differenz das zusätzliche Wertepaar zusätzliche Wertepaar (x_{n+1}, f_{n+1}) einzusetzen. Die Abschätzung ist dann natürlich nicht mehr exakt und hängt von dem vorgegebenen Wertepaar ab. Man kann diesen Wert dann lediglich als ein Näherungswert für den Fehler betrachten.

Bemerkung Es zeigt sich, dass speziell bei Polynomen höherer Ordnung unangenehme Oszillationen auftreten. Ist der Grad des Polynoms größer als fünf, d. h. es liegen mehr als sechs Wertepaare vor, dann sollte man für praktische Anwendungen keine dieser Interpolationen benutzen, sondern die Spline-Interpolation verwenden.

Beispiel A.4 (Newtonsche Interpolation, mit Script-File)

Wir greifen den Tragflügel des Flugzeugs auf, den wir in diesem Anhang zu Beginn tabelliert haben. Wir nehmen ein Polynom 5. Ordnung. Die sechs Stützstellen und Stützwerte sind dabei wie folgt gewählt:

Oberseite		Unterseite	
x	y	x	y
0	0	0	0
14,833	5,951	15,617	−4,733
40,000	6,803	40,000	−4,803
60,051	5,453	59,949	−3,675
80,058	3,178	79,942	−2,066
100,000	0,126	100,000	0,126

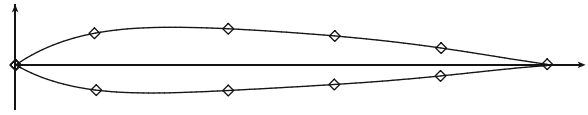


Abb. A.3 Interpolation eines NACA-Profils mit sechs Wertepaaren

Die Lösung mit dem Worksheet **Newton-Interpolation** ist in Abb. A.3 zu sehen. Bei dieser geringen Anzahl von Punkten ist die Vorderkante des Tragflügelprofils etwas zu spitz. Hier sollte man somit noch ein oder zwei Stützstellen mehr einführen, so dass die Approximation das geometrische Verhalten richtig wiedergeben kann. In unserer Tabelle liegt nur eine Stützstelle in dieser Rundung. □

Beispiel A.5 (Newtonsche Interpolation, mit Script-File)

Wir nehmen nun einen Punkt zusätzlich dazu und wählen die sieben Stützstellen wie folgt aus:

Oberseite		Unterseite	
x	y	x	y
0	0	0	0
1,158	1,954	1,342	−1,830
14,833	5,951	15,617	−4,733
40,000	6,803	40,000	−4,803
60,051	5,453	59,949	−3,675
80,058	3,178	79,942	−2,066
100,000	0,126	100,000	0,126

Das Ergebnis des Worksheets überrascht dann. Statt einer besseren Lösung liefert es oszillierende Kurven, wie dies in der Abb. A.4 aufgezeichnet ist.

Hier zeigt sich das in der Bemerkung angesprochene Verhalten: Das Interpolationspolynom kommt ins Schwingen und liefert eine sehr ungünstige Approximation. Durch die Vorgabe von anderen Stützpaaren kann man diese Oszillationen vermeiden. Für den praktischen Einsatz ist ein solches Verfahren natürlich nicht anwendbar. □

Die Polynominterpolation tendiert zur Instabilität, falls der Grad des Polynoms größer als fünf wird. Aus diesem Grunde haben sich in den letzten Jahren andere Methoden durch-

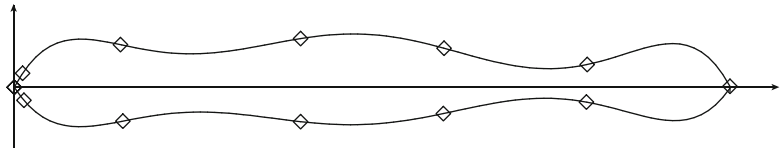


Abb. A.4 Interpolation eines NACA-Profils mit sieben Wertepaaren

gesetzt, die dieses Oszillationsverhalten nicht zeigen. Die wesentliche Idee dahinter ist einfach: Statt ein Interpolationspolynom hoher Ordnung zu benutzen, setzt man Polynome niedriger Ordnung möglichst stetig zusammen. Das nennt man Spline-Interpolation.

A.3 Spline-Interpolation

Der Nachteil der Polynome hoher Ordnung wird bei der Spline-Interpolation vermieden, indem nur Polynome niedriger Ordnung eingesetzt und diese stetig aneinander gehängt werden. Der Begriff Spline oder übersetzt Strak oder Straklatte stammt ursprünglich aus dem Schiffsbau. Es ist eine lange biegsame Holzlatte, die als Kurvenlineal für das Zeichnen von Schiffsrümpfen verwandt wird. Wird die Latte an einzelnen Punkten durch Nägel fixiert, so biegt sie sich so, dass die Krümmung minimal wird. Wir werden hier die Idee der Spline-Interpolation an den **kubischen Splines** erläutern. Zwischen zwei Stützpunkten wird dabei jeweils ein Polynom vom Grad $n = 3$ also ein kubisches Polynom gelegt. Durch Anschlussbedingungen wird garantiert, dass die zusammengesetzte Funktion zweimal stetig differenzierbar ist. Die Situation sieht dann wie folgt aus.

Gegeben sind $n + 1$ Stützwerte f_i an den Stützstellen $x_0, x_1, \cdots, x_n$, die der Größe nach sortiert sind:

$$a = x_0 < x_1 < x_2 < \cdots < x_n = b \ .$$

Die Näherungsfunktion $S = S(x)$ mit $x \in [a, b]$ setzt sich stückweise aus Polynomen zusammen, d. h.

$$S(x) = S_i(x) \text{ für } x \in [x_i, x_{i+1}] \ ,$$

mit $i = 0, 1, \ldots, n-1$, wobei wir kubische Polynome

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{A.28}$$

betrachten wollen.

Jedes kubische Polynom hat vier unbekannte Koeffizienten, also insgesamt gibt es $4n$ freie Parameter. Um diese zu bestimmen wird gefordert, dass die folgenden Bedingungen gelten:

1. S erfüllt die Interpolationsbedingungen

$$\begin{aligned} S(x_i) &= f_i, \quad i = 0, \cdots, n \text{ , d. h.} \\ S_i(x_i) &= f_i, \quad i = 0, \cdots, n-1 \\ S_{n-1}(x_n) &= f_n \ . \end{aligned} \tag{A.29}$$

2. S ist im Innern von $[a,b]$ zweimal stetig differenzierbar, d. h.

$$\begin{aligned} S_i(x_i) &= S_{i-1}(x_i), \quad i = 1,2,\cdots,n-1\ , \\ S_i'(x_i) &= S_{i-1}'(x_i), \quad i = 1,2,\cdots,n-1\ , \\ S_i''(x_i) &= S_{i-1}''(x_i), \quad i = 1,2,\cdots,n-1\ . \end{aligned} \tag{A.30}$$

Damit haben wir $4n-2$ Gleichungen für die $4n$ Koeffizienten. Die fehlenden zwei Bedingungen erhält man aus zusätzlichen Randbedingungen an S. Gilt

$$S_0''(a) = S_{n-1}''(b) = 0\ ,$$

so verlangt man den stetigen Übergang von S am Rand in eine Gerade. S wird dann der **natürliche kubische Spline** genannt.

Die Berechnung der ersten und zweiten Ableitung der kubischen Polynome S_i liefert

$$\begin{aligned} S_i'(x) &= b_i + 2c_i(x-x_i) + 3d_i(x-x_i)^2\ , \\ S_i''(x) &= 2c_i + 6d_i(x-x_i)\ . \end{aligned}$$

Aus den Bedingungen (A.29) und (A.30) erhält man damit

1. $a_i = f_i, i = 0,1,\ldots,n-1, a_n = 0,$
2. $c_0 = c_n = 0,$
3. $h_{i-1}c_{i-1}+2(h_{i-1}+h_i)c_i+h_ic_{i+1} = \frac{3}{h_i}(a_{i+1}-a_i)-\frac{3}{h_{i-1}}(a_i-a_{i-1}), i = 1,2,\cdots,n-1$ mit $h_i := x_{i+1}-x_i, i = 0,1,\ldots,n-1,$
4. $b_i = \frac{1}{h_i}(a_{i+1}-a_i)-\frac{h_i}{3}(c_{i+1}+2c_i), i = 0,1,\ldots,n-1,$
5. $d_i = \frac{1}{3h_i}(c_{i+1}-c_i), i = 0,1,\ldots,n-1.$

Die Gleichungen in 3. bilden ein System linearer Gleichungen

$$\mathbf{Ac} = \mathbf{r}$$

mit der Matrix

$$\mathbf{A} = \begin{pmatrix} 2(h_0+h_1) & h_1 & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & h_{n-2} & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix},$$

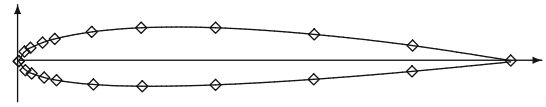


Abb. A.5 Spline-Interpolation eines NACA-Profils

dem Vektor **c**, in dem die gesuchten Koeffizienten stehen und der rechten Seite **r**:

$$\mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix}, \quad \mathbf{r} = \begin{pmatrix} \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \end{pmatrix}.$$

Man nennt dieses lineare Gleichungsystem ein tridiagonales Gleichungssystem, da nur die mittleren drei Diagonalen, Haupt- und die angrenzenden zwei Nebendiagonalen, besetzt sind und alle anderen Koeffizienten der Matrix Null sind. Die Matrix A ist stark diagonal dominant, streng regulär und positiv definit. Bei äquidistanten Schrittweiten h_i ist sie symmetrisch.

Das Gleichungssystem ist damit eindeutig lösbar. Am besten wendet man ein modifiziertes Gauß-Verfahren für tridiagonale Matrizen an, den sogenannten Thomas-Algorithmus an. Dieser wird im Kapitel über Randwertprobleme bei der Anwendung von Differenzenverfahren genauer beschrieben.

Die Vorgabe anderer Randwerte ist möglich, wie periodische Randwerte oder die Vorgabe der ersten Ableitungen der Spline-Funktionen am Rand. Die starke Forderung, dass die Krümmung der kubischen Polynome an den Stützstellen identisch ist, garantiert den glatten Verlauf der Interpolationsfunktion. Sie ist besonders für solche Anwendungen geeignet, von denen man weiß, dass sie durch sehr glatte Kurven beschrieben werden. Der kubische Spline beschreibt im Wesentlichen das Straken und minimiert die Biegungsenergie.

Beispiel A.6 (Spline-Interpolation, mit Script-File)

Wir greifen noch einmal das NACA-Profil aus Tab. A.1 auf. Diesmal verwenden wir allerdings alle Stützstellen und wenden die Spline-Interpolation darauf an. Das Ergebnis in Abb. A.5 entspricht einem realen Profil. □

A.4 Bemerkungen und Entscheidungshilfen

Wir haben uns in diesem Anhang auf die Interpolation durch Polynome und stückweise Polynome, speziell die kubischen Splines beschränkt. Dies ist nur ein kleiner Ausschnitt der Interpolation und Approximation, den wir im Kapitel über die numerische Lösung von Integralen und von Differenzialgleichungen benötigen.

Die Interpolation mit Hilfe von Polynomen kann mit Polynomen bis zum Grad fünf ausgeführt werden. Insofern ist der praktische Einsatz der reinen Polynominterpolation beschränkt. Es ist das klassische Verfahren für die Interpolation zwischen einigen wenigen Werten. Werden bei einer Wertetabelle an Zwischenstellen Werte benötigt und ist die Genauigkeitsanforderung nicht sehr hoch, dann kann dort lokal eine solche Interpolationsformel eingesetzt werden. Es werden dann nur die benachbarten Werte als Stützwerte benutzt und keine Interpolation über alle vorliegenden Werte erzeugt. Bei Näherungswerten an diskreten Punkten im Rahmen einer numerischen Simulation tritt die gleiche Situation auf. Bei der Konstruktion von Näherungsverfahren wie bei der numerischen Integration und Differenziation spielt sie eine wichtige Rolle und kommt in diesen Kapiteln entsprechend vor.

Die Newtonsche und Lagrangesche Interpolationsformel liefert natürlich dasselbe Polynom, da dieses eindeutig bestimmt ist. Allerdings haben die verschiedenen Darstellungen verschiedene Eigenschaften. Bei der Hinzunahme einer zusätzlichen Interpolationsstelle müssen bei der Lagrangeschen Formel alle Stützpolynome L_i neu berechnet werden. Die Berechnung der Lagrangeschen Stützpolynome, alle vom Grad n, ist im Vergleich zur Newton-Interpolation aufwändiger. Insofern ist die Newtonsche Interpolationsformel für praktische Rechnungen vorzuziehen. Die Lagrange Interpolation hat vor allem durch den einfachen Aufbau theoretische Bedeutung. Für praktische Probleme mit vielen Werten oder im Bereich CAD hat die Interpolation mit Splines die reine Polynominterpolation abgelöst. Die Spline-Interpolation ist robuster und zeigt kein Oszillationsverhalten.

Es gibt eine ganze Reihe weiterer Ansätze für die Interpolation. Eng verwandt mit der Polynominterpolation ist die **Hermite-Interpolation**. Hier wird das Interpolationspolynom erzeugt, indem neben den Werten der Funktion auch Werte der Ableitungen mit in die Berechnung eingehen. Dies wird gerne dort angewandt, wo auch Werte von Ableitungen vorliegen. Ein typische Anwendung ist die Interpolation von Näherungslösungen von Differenzialgleichungen. Mit der Hermite-Interpolation können sehr genaue Interpolationspolynome konstruiert werden. Allerdings ist bei vielen Stützstellen auch ein oszillierendes Verhalten möglich. Interpolation mit gebrochen rationalen Funktionen wird insbesondere dort eingesetzt, wo bekannt ist, dass die interpolierte Funktion eine Singularität besitzt. Dagegen wird die trigonometrische Interpolation gerne für periodische Probleme benutzt.

Wir haben hier nur die Interpolation von Funktionen einer Raumdimension betrachtet. Anwendungen in mehreren Dimensionen, wie Kurven und Flächen im Raum sind zum Beispiel im Bereich der CAD-Systeme sehr wichtig. Weitergehende Darstellungen der Interpolation findet man in verschiedenen Büchern zur numerische Mathematik, z. B. [2–6]. Ein Standardwerk zu der Spline-Interpolation ist [7]. Unterprogramme zur Interpolation finden sich in [8] oder [9] und in den großen numerischen Software-Bibliotheken IMSL [10] und NAG [11].

B Lösen nichtlinearer Gleichungen

In diesem Anhang geht es um die numerische Lösung von nichtlinearen algebraischen Gleichungen, wie z. B.

$$x^2 + 7x + 9 = 0 \,,$$
$$e^x + \cos x = \sin x \,,$$
$$g(x) = h(x) + 7, \ldots \,.$$

Die Approximation einer nichtlinearen Differenzialgleichung wird auf ein solches Problem führen. So ergeben sich bei den Differenzenverfahren ein System von nichtlinearen Differenzengleichungen, welches dann numerisch gelöst werden muss. Ein anderes Beispiel ist das Schießverfahren in Kap. 4.2 zur numerischen Lösung von gewöhnlichen Randwertproblemen. Hier ist es so, dass die vorkommende Funktion nicht unbedingt durch eine geschlossene Formel gegeben ist.

Ganz allgemein betrachten wir in diesem Kapitel des Anhangs die numerische Lösung einer Gleichung der Form

$$f(x) = 0 \,. \tag{B.1}$$

Gesucht ist somit eine Nullstelle der im Allgemeinen nichtlinearen Funktion von $f = f(x)$. Ist die nichtlineare Funktion nicht gerade eine quadratische Funktion, dann ist man darauf angewiesen, dies näherungsweise auszuführen. Vorausgesetzt sei im Folgenden immer, dass die Funktion $f = f(x)$ im betrachteten Bereich zumindest **stetig** ist.

Die Stetigkeit von Funktionen, die durch Formelausdrücke gegeben sind, kann man entsprechend überprüfen. Beim Schießverfahren in Kap. 4.2 ist diese Überprüfung allerdings nicht so einfach. Wir gingen beim Schießverfahren davon aus, dass nach Definition in Abschn. 3.2.1 dies immer der Fall ist, wenn das Problem sachgemäß gestellt ist.

Im Folgenden werden wir verschiedene Verfahren zur Lösung nichtlinearer Gleichungen vorstellen.

B.1 Bisektion

Aus der Analysis kennt man den Satz von Rolle, der besagt, dass eine stetige Funktion, die an einer Stelle einen positiven und an einer anderen Stelle einen negativen Wert hat, dazwischen mindestens eine Nullstelle besitzt. Diese Tatsache kann man als Ausgangspunkt eines numerischen Verfahrens zum Auffinden von Nullstellen benutzen.

Gibt es zwei Punkte x_0, x_1 mit der Eigenschaft, dass

$$f(x_0)\, f(x_1) < 0\ ,$$

d. h. die beiden zugehörigen Funktionswerte $f(x_0)$ und $f(x_1)$ haben unterschiedliche Vorzeichen, dann gibt es nach dem Satz von Rolle zwischen x_0 und x_1 mindestens eine Nullstelle von f.

Wir nehmen an, dass $x_0 < x_1$ und halbieren als nächstes das Intervall $[x_0, x_1]$, nehmen den Mittelpunkt des Intervalls und prüfen das Vorzeichen des Funktionswerts an dieser Stelle. Ist es das gleiche Vorzeichen wie das des Funktionswertes bei x_0, so ersetzen wir x_0 durch den neuen Punkt x_{neu} und wir haben erneut ein Intervall, jetzt $[x_{\text{neu}}, x_1]$, welches die gleichen Eigenschaften wie das Ausgangsintervall $[x_0, x_1]$ hat. Es befindet sich somit eine Nullstelle in diesem Intervall.

Im anderen Fall hat die Funktion bei x_{neu} das gleiche Vorzeichen wie der Funktionswert bei x_1, so ersetzen wir x_1 durch den neuen Punkt. Das neue Intervall ist dann $[x_0, x_{\text{neu}}]$. Wir sehen, wie dieses Verfahren weiter läuft. Man fängt wieder von vorne an. Das betrachtete Intervall wird sukzessive halbiert und die Hälften weiter untersucht, in welcher der Vorzeichenwechsel stattfindet. War der Funktionswert schon Null oder aber sein Betrag so klein, dass er uns als numerische Näherung reicht, dann sind wir bereits fertig.

Die einzelnen Schritte lassen sich in dem folgenden Algorithmus formulieren:

- Berechne $x_{\text{neu}} = \dfrac{x_0 + x_1}{2}$.
- Falls $|f(x_{\text{neu}})|$ klein genug, brich ab und verwende x_{neu} als numerische Näherung der Nullstelle.
- sonst:
  - Falls $f(x_0)\, f(x_{\text{neu}}) < 0$, ersetze x_1 durch x_{neu} ($x_1 := x_{\text{neu}}$).
  - Andernfalls ersetze x_0 durch x_{neu} ($x_0 := x_{\text{neu}}$).
  - Beginne von vorne.

In Abb. B.1 sind die Punkte ihrer Entstehung nach nummeriert. Der Algorithmus ist in Abb. B.2 zusätzlich als Struktogramm dargestellt.

Bemerkung Haben beim Start des Verfahrens die Funktionswerte an den Intervallgrenzen x_0 und x_1 die gleichen Vorzeichen, so muss man das Intervall entsprechend vergrößern bis unterschiedliche Vorzeichen auftreten.

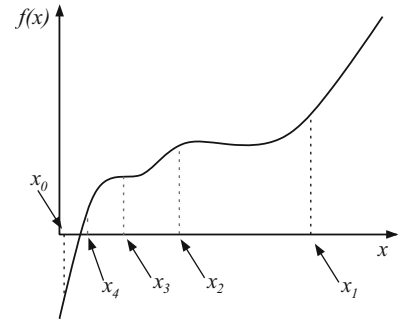


Abb. B.1 Prinzip des Bisektionsverfahrens

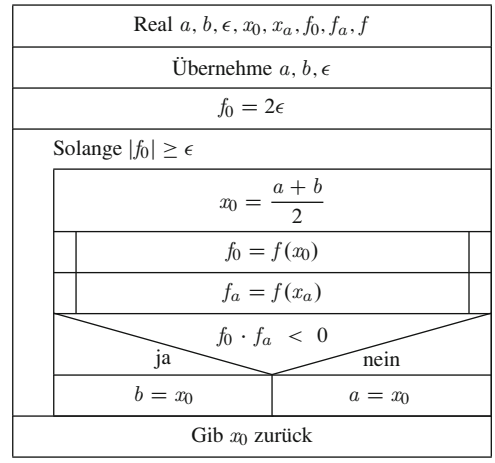
B.2 Regula Falsi

Schon Adam Riese kannte eine Verbesserung der Bisektion. Die Wahl des neuen Punktes in der Mitte erscheint oft nicht optimal gewählt, und das Verfahren kann selbst bei sehr einfachen Funktionen wie z. B. linearen sehr lange brauchen. Da liegt es nahe, statt der Halbierung eine Gerade durch die Punkte $(x_0, f(x_0))$ und $(x_1, f(x_1))$ zu ziehen. Dort, wo diese die x-Achse schneidet, wählt man dann den neuen Wert x_{neu}. Damit hat man bei linearen Funktionen in einem Schritt die exakte Lösung.

Aber auch bei nichtlinearen Funktionen sollte dieses Verfahren schneller zum Ziel kommen. In Abb. B.3 zeigt sich dies an unserem Beispiel deutlich. Nun benötigen wir noch die zugehörigen Formeln, nach der sich x_{neu} berechnen lässt. Dazu setzen wir einfach eine allgemeine lineare Funktion an, die in x_0 den Wert $f(x_0)$ hat, in x_1 den Wert $f(x_1)$ und in x_{neu} verschwindet. Nach dem Lagrangschen Interpolationspolynom lautet

Abb. B.2 Bisektionsverfahren, es wird dabei vorausgesetzt, dass die Funktionswerte an den Grenzen des Ausgangsintervalls unterschiedliches Vorzeichen besitzen

Bisektion als Unterprogramm



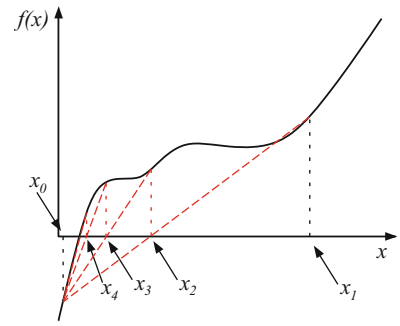


Abb. B.3 Prinzip der Regula Falsi

die Geradengleichung

$$y = f(x_0)\frac{x - x_1}{x_0 - x_1} + f(x_1)\frac{x - x_0}{x_1 - x_0}.$$

Setzen wir $x = x_{neu}$ in diese Gerade ein, ergibt sich als Funktionswert Null. Löst man weiter nach x_{neu} auf, so ergibt sich die gewünschte Formel:

$$x_{\text{neu}} = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)} \,. \tag{B.2}$$

Der Algorithmus für die Regula Falsi lautet also:

- Berechne x_{neu} nach Formel (B.2).
- Falls $|f(x_{\text{neu}})|$ klein genug, brich ab und verwende x_{neu} als numerische Näherung der Nullstelle.
- sonst:
  - Falls $f(x_0) f(x_{\text{neu}}) < 0$, ersetze x_1 durch x_{neu} ($x_1 := x_{\text{neu}}$).
  - Andernfalls ersetze x_0 durch x_{neu} ($x_0 := x_{\text{neu}}$).
  - Beginne von vorne.

In der Abb. B.3 hierzu sind die Punkte wieder in der Reihenfolge ihrer Entstehung nummeriert.

B.3 Sekantenverfahren

War die Regula Falsi eine Variation der Bisektion, so ist das Sekantenverfahren eine Variation der Regula Falsi. Es ergibt sich, wenn man sich von Anfang an nicht darum kümmert,

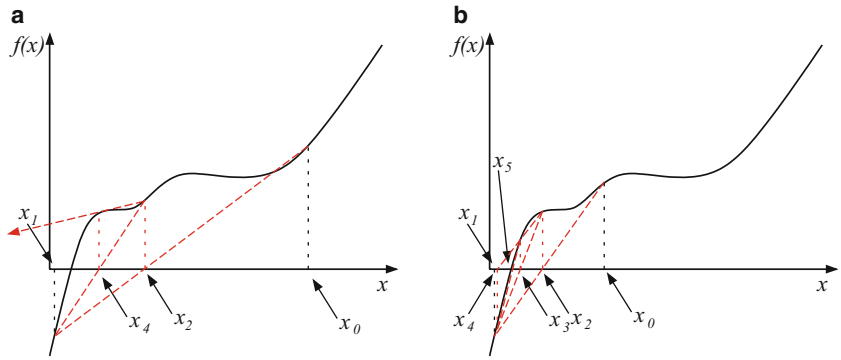


Abb. B.4 Sekantenverfahren; **a** x_0, x_1 vertauscht, **b** Startwerte näher zusammen

ob die jeweiligen zwei Punkte die Nullstelle einschließen oder nicht. Man verwirft einfach, nachdem man den neuen Punkt ausgerechnet hat, den ältesten Punkt. Sind x_0 und x_1 gegeben, so lautet die Iterationsvorschrift:

$$x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})} \,. \tag{B.3}$$

Die Abb. B.4 zeigen, wo die Stärken und die Schwächen liegen. Im linken Bild wurden die Startwerte wie im Beispiel für die Bisektion und Regula Falsi gewählt. Das Verfahren führt sehr schnell aus dem Definitionsbereich von f heraus. Sind dagegen die Startwerte wie im rechten Bild näher beieinander, so kann man sehr schnelle Konvergenz bekommen.

Möchte man dieses Verfahren zur Approximation einer Nullstelle verwenden, so muss man sicher stellen, dass das Verfahren abbricht, sobald man in eine Situation wie in den beiden linken Bildern kommt, und mit zwei anderen Werten neu startet. Diese kann man sich durch ein paar Schritte des Bisektionsverfahrens besorgen.

Eine wichtige Anwendung des Sekantenverfahrens ist die Suche nach einer Anfangseinschließung für die Bisektion oder Regula Falsi. Hat man nämlich am Anfang zwei Punkte, deren Funktionswert das gleiche Vorzeichen hat, so kann man einige Schritte des Sekantenverfahrens durchführen, bis man auf einen Punkt kommt, dessen Funktionswert das umgekehrte Vorzeichen besitzt. Für das Schießverfahren wird das in Abschn. 4.2.2 beschrieben.

B.4 Das Newton-Verfahren

Für differenzierbare Funktionen kann man aus dem Sekantenverfahren eine Methode herleiten, die oft noch um einiges schneller konvergiert: das Newton-Verfahren. Die Iterati-

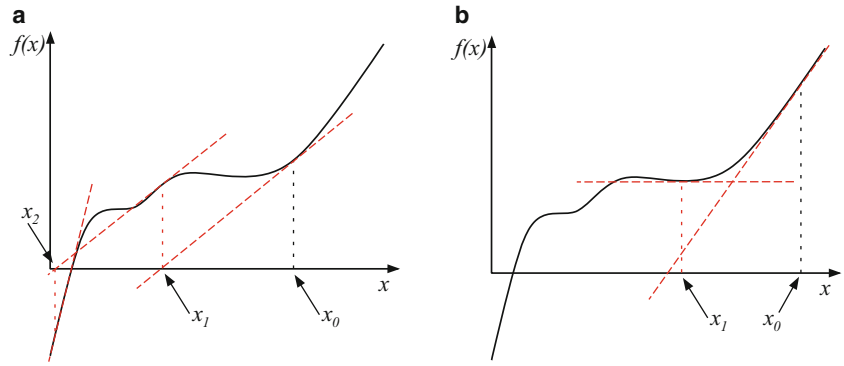


Abb. B.5 Newton-Verfahren; **a** Guter Startwert, **b** Startwert leicht verschoben

onsvorschrift des Sekantenverfahrens (B.3) lässt sich umschreiben in

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} .$$

Für differenzierbares f kann man hier den Grenzübergang $x_{k-1} \to x_k$ durchführen. Der Bruch auf der rechten Seite ist gerade der Kehrwert des Differenzenquotienten. Durch den Grenzübergang geht er demnach in den Kehrwert der Ableitung über. Damit ergibt sich als Iterationsvorschrift für das Newton-Verfahren:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} . \tag{B.4}$$

Man sucht nun nicht mehr nach dem Schnittpunkt der Sekante mit der x-Achse sondern nach dem Schnittpunkt der Tangente mit der x-Achse.

In der rechten Abbildung in Abb. B.5 findet man ein Beispiel dafür, dass dies nicht immer gut gehen muss. Wird die Ableitung Null, so ist die Tangente parallel zur x-Achse und schneidet sie nicht. In der Formel (B.4) drückt sich das darin aus, dass der Nenner des Bruchs Null wird. Hier versagt das Verfahren und bricht ab. Hat man jedoch wie in der linken Abbildung einen guten Startwert erwischt, so bekommt man eine besonders schnelle Konvergenz. Man muss also auch hier Abfragen einbauen, die kontrollieren, ob der Startwert gut war. Man kann z. B. dann, wenn der Betrag des Funktionswertes der Iterierten $|f(x_k)|$ steigt statt zu fallen, für ein paar Schritte auf die Regula Falsi übergehen und mit dem dort erreichten Wert wieder in die Newton-Iteration einsteigen.

B.5 Bemerkungen und Entscheidungshilfen

Das Newton-Verfahren ist sicherlich das Schnellste der Verfahren. Allerdings muss man es, wie gesehen, oft mit einem anderen Verfahren kombinieren, weil man einem gegebenen Startwert nicht direkt ansehen kann, ob er zur Konvergenz führt oder nicht. Ist die Funktion nicht differenzierbar oder die Ableitung nicht durch eine geeignete Formel gegeben, so ist das Verfahren nicht anwendbar. Manchmal ist die Berechnung der Ableitung auch so aufwändig, dass sie die schnellere Konvergenz im Vergleich zum Sekantenverfahren wieder aufwiegt. In einem solchen Fall wird auch ein modifiziertes Newton-Verfahren angewandt, bei dem die Ableitung nicht in jedem Iterationsschritt neu berechnet wird.

Ein Vorteil des Sekantenverfahrens gegenüber dem Newton-Verfahren ist der wesentlich geringere Rechenaufwand für einen Iterationsschritt. Das Problem, dass man den Startwerten die Konvergenz oder Divergenz nicht direkt ansehen kann, teilt es allerdings mit diesem. So muss man beide Verfahren oft in Kombination mit der Bisektion oder Regula Falsi benutzen.

Die Bisektion und die Regula Falsi garantieren, dass das Verfahren konvergiert, sobald man eine Einschließung einer Nullstelle durch Punkte mit Funktionswerten unterschiedlichen Vorzeichens hat. Allerdings erkauft man sich diese Sicherheit durch eine – vor allem bei der Bisektion – langsamere Konvergenz. Oft ist diese aber ausreichend, insbesondere dann, wenn keine allzu hohe Genauigkeit gefordert wird. Bei der Regula Falsi kann es vorkommen, dass einer der beiden Punkte sich nur sehr langsam ändert, obwohl der andere der Nullstelle schon sehr nahe kommt. Dieses ist für die Konvergenz natürlich nicht ideal. In einem solchen Fall kann man einen Schritt mit dem Bisektionsverfahren dazwischen ausführen oder einen Schritt mit der Hälfte des Funktionswerts an diesem Punkt in der Formel (B.2) ausführen. Auf diese Art und Weise kann man den neuen Punkt auf die andere Seite der Nullstelle bringen. Der alte Punkt, der sich so langsam bewegt hatte, wird durch eine bessere Näherung ersetzt. Für die Anfangseinschließung brauchen beide Verfahren, wie oben gesehen, oft ein paar Schritte der Sekantenmethode, so dass auch diese beiden mit den anderen kombiniert werden sollten.

Als weiterführende Literatur sei auf die allgemeinen Bücher über numerische Methoden verwiesen, so z. B. [2–6].

C Iterative Methoden zur numerischen Lösung von linearen Gleichungssytemen

Bei Näherungsverfahren für partielle Differenzialgleichungen treten oft große lineare Gleichungssystem (LGS) auf, welche zur Ermittlung der Werte der Nährungslösung gelöst werden müssen. Diese Gleichungssysteme nennt man schwach besetzt, da nur sehr wenige Elemente der Matrix von Null verschieden sind. So hat man bei einem Differenzenverfahren für die Poisson-Gleichung mit dem 5-Punkte-Stern höchstens 5 Elemente einer Zeile der Matrix von Null verschieden. Schwach besetzte Matrizen treten analog auch bei Finite-Volumen- und Finite-Element-Verfahren auf. Insofern ist die effiziente Lösung von schwach besetzten LGS eine wichtige Grundaufgabe bei der numerischen Lösung von partiellen Differenzialgleichungen.

Iterative Methoden haben bei schwach besetzten Gleichungssystemen wesentliche Vorteile. Die Matrix des Gleichungssystems mit den vielen Nullen muss nicht abgespeichert werden, es werden nur die Elemente ungleich Null benötigt. Weiter muss ein System von Differenzengleichungen, welches schon eine diskrete Approximation der Differenzialgleichung darstellt, nicht unbedingt exakt gelöst werden. Eine Näherungslösung des LGS, deren Fehler etwas kleiner als der schon gemachte Verfahrensfehler ist, reicht hier aus. Der Rechenaufwand eines Iterationsverfahrens wird insbesondere bei großen Systemen sehr viel kleiner sein als der des Gaußschen Eliminationsverfahrens.

Eine iterative Methode wird durch eine Iterationsvorschrift definiert. Auf der linken Seite steht die Iterierte im neuen Iterationsschritt, auf der rechten Seite eine Berechnungsvorschrift, in welche die alten Iterierten eingehen. Durch wiederholtes Ausführen werden sukzessive Näherungen der Lösung erhalten. Die im Kap. 7 im Abschnitt über die Differenzenverfahren für elliptische Differenzialgleichungen vorgestellten iterativen Verfahren sind bei einer kleinen Anzahl von Unbekannten meist effizient. In der Praxis erfordert allerdings die Auflösung mancher physikalischer Phänomene sehr kleine Schrittweiten und somit eine sehr große Zahl von Gitterpunkten. Hier sind schnellere Methoden nötig, die im Folgenden in einem Überblick vorgestellt werden. Für weitere Informationen wird auf die entsprechende Spezialliteratur verwiesen. Begonnen wird mit einer etwas allgemeineren Darstellung der in Kap. 7 vorgestellten klassischen Iterationsverfahren, danach werden Mehrgitter-, CG- und Krylov-Unterraum-Methoden angesprochen.

Wir betrachten ein lineares Gleichungssystem der From

$$\mathbf{Au} = \mathbf{b}$$

mit der $(n \times n)$-Matrix $\mathbf{A}$, der rechten Seite $\mathbf{b}$ und dem gesuchten Lösungsvektor $\mathbf{u}$:

$$\mathbf{A} = (a_{ij})_{i=1,2,..,n;j=1,2,..,n}; \quad \mathbf{b} = (b_1, b_2, .., b_n)^T; \quad \mathbf{u} = (u_1, u_2, .., u_n)^T \ .$$

Das obige LGS besteht aus n Zeilen der Form

$$\sum_{j=1}^{n} a_{ij} u_j = b_i \qquad \text{mit} \quad i = 1, 2, \ldots, n \ .$$

C.1 Die klassischen Iterationsmethoden

In Kap. 7 haben wir die i-te Gleichung des Systems von Differenzengleichungen für die Poisson-Gleichung nach u_i aufgelöst (d. h. Gleichung 1 nach u_1, Gleichung 2 nach u_2, ..., Gleichung n nach u_n):

$$u_i = \frac{1}{a_{ii}} (b_i - \sum_{j=1}^{i-1} a_{ij} u_j - \sum_{j=i+1}^{n} a_{ij} u_j), \qquad i = 1, 2, \ldots, n \ .$$

Dieses System wird dann iterativ gelöst, indem zunächst eine Anfangsschätzung für die Lösung des LGS $(u_1^{(0)}, u_2^{(0)}, \ldots, u_n^{(0)})$ vorgegeben wird und dann neue Iterierte durch die Vorschrift

$$u_i^{(m+1)} = \frac{1}{a_{ii}} (b_i - \sum_{j=1}^{i-1} a_{ij} u_j^{(m)} - \sum_{j=i+1}^{n} a_{ij} u_j^{(m)}), \qquad i = 1, 2, \ldots, n$$

bestimmt werden.

Dieses Iterationsverfahren heißt **Jacobi-Verfahren** und ist im Worksheet **Jacobi** sowohl direkt als auch in Form einer Prozedur programmiert.

Bei dem Jacobi-Verfahren beginnt man bei der $(m + 1)$-ten Iteration mit der 1. Gleichung, um $u_1^{(m+1)}$ durch die „alten" Werte $u_1^{(m)}, u_2^{(m)} \ldots, u_n^{(m)}$ zu berechnen. Anschließend wählt man Gleichung 2 und berechnet $u_2^{(m+1)}$ aus den „alten" Daten $u_1^{(m)}, u_2^{(m)}, \ldots, u_n^{(m)}$. Dabei könnte man im Prinzip schon ausnutzen, dass $u_1^{(m+1)}$ zu diesem Zeitpunkt schon berechnet wurde und voraussichtlich auch schon eine bessere Approximation darstellt. Man sollte also ein verbessertes Verfahren erhalten, indem man bei der Berechnung von $u_i^{(m+1)}$ die schon aktualisierten Werte $u_1^{(m+1)}, u_2^{(m+1)}, \ldots, u_{i-1}^{(m+1)}$ berücksichtigt:

Gauß-Seidel-Verfahren

$$u_i^{(m+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} u_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} u_j^{(m)}), \qquad i = 1, 2, \ldots, n \ .$$

Dieses Verfahren nennt man das **Gauß-Seidel-Verfahren** und ist im MAPLE-Worksheet **Gauß-Seidel** sowohl direkt als auch in Form einer Prozedur programmiert.

Das **SOR-Verfahren** (Successive Overrelaxation) beruht auf der Tatsache, dass die Konvergenz des Gauß-Seidel-Verfahrens verbessert werden kann, wenn man eine Linearkombination des alten Wertes $u_i^{(m)}$ und des aktualisierten Wertes $u_i^{(m+1)}$ wählt

SOR-Verfahren

$$u_i^{(\text{neu})} = w \cdot u_i^{(m+1)} + (1 - w) \cdot u_i^{(m)} \ .$$

Der Relaxationsparameter w liegt üblicherweise im Bereich $1 \leq w \leq 2$. Bei $w = 1$ ist das Verfahren identisch mit dem Gauß-Seidel-Verfahren. Werte des Relaxationsparameters w über 1 bedeuten, dass der neue iterierte Wert stärker gewichtet wird, daher der Name Überrelaxation. Es muss $0 < w < 2$ sein, da sonst keine Konvergenz erfolgt.

Das SOR-Verfahren ist im MAPLE-Worksheet **SOR** sowohl direkt als auch in Form einer Prozedur programmiert. Es kann gezeigt werden, dass der optimale Relaxationsparameter durch $w_{\text{opt}} = \frac{2}{1+\sqrt{1-\rho^2}}$ gegeben ist, wenn ρ der größte Eigenwert der Jacobi-Matrix ist.

Den Iterationsverfahren ist eines gemeinsam, dass die Iteration bei Erreichen der gewünschten Genauigkeit abgebrochen werden muss. Um ein **Abbruchkriterium** zu erhalten, bestimmt man nach jeder Iteration das Residuum. Im m-ten Iterationsschritt ist es durch

$$R_i^{(m)} = \sum_{j=1}^{n} a_{ij} u_j^{(m)} - b_i \qquad (i = 1, \ldots, n)$$

gegeben. Von diesem Residuumvektor nehmen wir die maximale Komponente

$$R^{(m)} = \max_{i=1,\ldots,n} |R_i^{(m)}| \ ,$$

die Alternative wäre das quadratische Mittel.

Als Abbruchkriterium fordert man in der Regel, dass sowohl

1. das Residuum kleiner einer gewissen Vorgabe ist, $R^{(m)} < \delta_1$, als auch dass
2. die Differenz der Werte zweier aufeinander folgender Iterationen kleiner einer vorgegebenen Schranke sind

$$\max_{i=1,\ldots,n} |\mathbf{u}_i^{(m)} - \mathbf{u}_i^{(m-1)}| < \delta_2.$$

Im Kapitel der Differenzenverfahren für elliptische Differenzialgleichungen werden diese Verfahren so in Rechenprogramme umgesetzt, indem die Differenzengleichungen geeignet umgestellt wurden. Allgemeine theoretische Aussagen und Informationen über die Struktur dieser Iterationsverfahren erhält man über die Matrix-Formulierung: Dazu wird die Matrix **A** in die Summe

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U} \tag{C.1}$$

zerlegt. Dabei ist **L** (lower) die Matrix, welche die Elemente von A im unteren Dreieck enthält, **U** (upper) ist die Matrix, welche die Elemente im oberen Dreieck enthält und **D** die Matrix mit den Diagonalelementen. Die Matrizen haben somit die Gestalt

$$\mathbf{L} = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ * & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ * & \cdots & * & 0 \end{pmatrix}, \quad \mathbf{D} = \begin{pmatrix} * & & & \\ & \ddots & & \\ & & \ddots & \\ & & & * \end{pmatrix},$$

$$\mathbf{U} = \begin{pmatrix} 0 & * & \cdots & * \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & * \\ 0 & \cdots & \cdots & 0 \end{pmatrix}, \tag{C.2}$$

wobei „$*$“ für ein Matrixelement steht, welches ungleich Null sein darf. Es wird hier vorausgesetzt, dass die Diagonalmatrix D regulär ist und somit die Inverse D^{-1} existiert.

Wir setzen diese Aufspaltung in das Gleichungssystem ein und erhalten

$$(\mathbf{L} + \mathbf{D} + \mathbf{U})\mathbf{u} = \mathbf{b}\,. \tag{C.3}$$

Behält man hier **D** auf der linken Seite und schafft alles andere auf die rechte, ergibt sich

$$\mathbf{D}\mathbf{u} = -(\mathbf{L} + \mathbf{U})\mathbf{u} + \mathbf{b}\,. \tag{C.4}$$

Mit der inversen Matrix $\mathbf{D}^{-1}$ multipliziert folgt

$$\mathbf{u} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{u} + \mathbf{D}^{-1}\mathbf{b} \tag{C.5}$$

und damit der Ausgangspunkt für eine Iterationsvorschrift:

$$\mathbf{u}^{(m+1)} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{u}^{(m)} + \mathbf{D}^{-1}\mathbf{b} \tag{C.6}$$

mit $m = 0, 1, \ldots$. Schaut man sich diese Gleichung komponentenweise an, dann sieht man schnell, dass dies gerade die Matrix-Schreibweise des Jacobi-Verfahrens ist.

Allgemein können wir ein lineares Iterationsverfahren damit durch die Vorschrift

$$\mathbf{u}^{(m+1)} = \mathbf{M}\mathbf{u}^{(m)} + \mathbf{N}\mathbf{b} \tag{C.7}$$

definieren. Durch die Wahl

$$\mathbf{M}_J := -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U}), \quad \mathbf{N}_J = \mathbf{D}^{-1} \tag{C.8}$$

erhält man das **Jacobi-Verfahren**.

Beim **Gauß-Seidel-Verfahren** werden die schon berechneten Werte im neuen Iterationslevel schon benutzt. Die zugehörigen Koeffizienten stehen in der unteren Dreiecksmatrix. Man erhält somit für dieses Verfahren zunächst

$$(\mathbf{L}+\mathbf{D})\mathbf{u} = -\mathbf{U}\mathbf{u} + \mathbf{b} \tag{C.9}$$

und daraus die Iterationsvorschrift

$$\mathbf{u}^{(m+1)} = -(\mathbf{L}+\mathbf{D})^{-1}\,\mathbf{U}\mathbf{u}^{(m)} + (\mathbf{L}+\mathbf{D})^{-1}\,\mathbf{b} \tag{C.10}$$

mit den Iterationsmatrizen

$$\mathbf{M}_{\mathbf{GS}} := -(\mathbf{L}+\mathbf{D})^{-1}\,\mathbf{U}, \quad \mathbf{N}_{\mathbf{GS}} := (\mathbf{L}+\mathbf{D})^{-1}\,. \tag{C.11}$$

Das SOR-Verfahren lässt sich analog in die Gestalt (C.7) bringen. Die Matrizen der Iterationsvorschrift lauten hier

$$\mathbf{M}_{\mathrm{SOR}} = -(\boldsymbol{\omega}\,\mathbf{L}+\mathbf{D})^{-1}\,((\boldsymbol{\omega}-\mathbf{1})\,\mathbf{D}+\boldsymbol{\omega}\,\mathbf{U}), \tag{C.12}$$

und

$$\mathbf{N}_{\mathrm{SOR}} = \boldsymbol{\omega}\,(\boldsymbol{\omega}\,\mathbf{L}+\mathbf{D})^{-1}\,. \tag{C.13}$$

Für diese Verfahren kann man Konvergenzaussagen ableiten. Das theoretische Werkzeug ist der Banachsche Fixpunktsatz oder besser der endlich dimensionale Spezialfall dieses Satzes. Man muss zeigen, dass die Iterationsmatrix eine Kontraktion definiert. Darauf werden wir hier nicht weiter eingehen, sondern auf die Spezialliteratur verweisen. Es gibt die folgende Aussage:

Die Iteration (C.7) ist genau dann für jeden Startvektor $\mathbf{u}^0$ und jede rechte Seite $\mathbf{b}$ konvergent, wenn

$$\boldsymbol{\rho}(\mathbf{M}) < \mathbf{1} \tag{C.14}$$

gilt. Dabei ist

$$\boldsymbol{\rho}(\mathbf{M}) = \max_{i=1,\ldots,n} |\lambda_i(\mathbf{M})| \tag{C.15}$$

der Spektralradius von $\mathbf{M}$, λ_i bezeichnet den i-ten Eigenwert von $\mathbf{M}$.

Aus diesem Satz lassen sich dann verschiedene Bedingungen ableiten, bei denen die Konvergenz der Iterationsverfahren unabhängig von dem Startvektor und der rechten Seite gesichert ist. Kriterien dieser Art sind:

Bemerkungen

1. Ist $\mathbf{A}$ strikt diagonal dominant, das heißt es gilt

$$|a_{ii}| > \sum_{\substack{i=1 \\ j \neq i}}^{n} |a_{ij}|, \quad i = 1, 2, \ldots, n \ , \tag{C.16}$$

   dann ist das Jacobi- und das Gauß-Seidel-Verfahren konvergent.
2. Ist $\mathbf{A}$ symmetrisch und positiv definit, dann ist das Jacobi- und das SOR-Verfahren mit $\omega \in (0, 2)$ konvergent.
3. Das SOR-Verfahren ist höchstens für $0 < \omega < 2$ konvergent.

Weitere Konvergenzsätze sind in der Spezialliteratur zu finden. Ebenso gibt es hier weitere Aussagen über die optimale Wahl des Relaxationsparameters ω.

Bei der numerischen Lösung der Poisson-Gleichung mit dem Differenzenverfahren in Kap. 7 ergibt sich

$$\omega_{\text{opt}} = \frac{2}{1 + \sqrt{1 - \mu^2}}, \quad \mu := \rho(\mathbf{M}_J) \ . \tag{C.17}$$

Bei der Approximation von elliptischen Randwertproblemen zeigt sich, dass der Spektralradius der Iterationsmatrizen $\mathbf{M}$ das Verhalten

$$\rho(\mathbf{M}) = 1 - 0(h^P) \tag{C.18}$$

besitzt. Dabei ist h die Diskretisierungsschrittweite und der Exponent p positiv. Je kleiner die Schrittweite wird, desto mehr strebt der Spektralradius gegen Eins und das Konvergenzverhalten wird schlechter. Das Problem wird schlecht konditioniert. Speziell bei sehr großen, schwach besetzten Gleichungssystemen ist dann das SOR-Verfahren selbst bei optimaler Wahl von w langsam.

C.2 Mehrgitterverfahren

Die genaue Betrachtung der klassischen Iterationsverfahren zur Lösung der großen schwach besetzten Gleichungssysteme ist der Ausgangspunkt der Mehrgitterverfahren. Es zeigt sich, dass ein klassisches Iterationsverfahren kleinskalige Anteile in der Lösung sehr schnell sehr gut wiedergibt und die große Anzahl der Iterationen auf die großskaligen Lösungsanteile zurückgehen. Wenn wir an das Differenzenverfahren für die Poisson-Gleichung denken, dann ist dieses Verhalten einfach zu motivieren. Setzen wir den 5-Punkte-Stern ein, dann ist jeder Gitterpunkt mit den umliegenden Punkten verbunden. In einer Iteration wird damit Information über diesen 5-Punkte-Stern ausgetauscht. Information mit der Punkten, die in der Nähe liegen, geschieht somit recht schnell, während der Informationsaustausch von weit entfernten Punkten dann sehr viele Iterationen erfordert. Man denke nur daran, wenn die Information von einem Eck des Rechengebiets zu dem anderen Eck transportiert werden muss. Man benötigt über den 5-Punkte-Stern dann eine Anzahl von Iterationen in der Größenordnung der Anzahl der Gitterzellen. Kurz: die hochfrequenten Anteile in der Lösung sind sehr schnell sehr gut approximiert, während eine gute Approximation der niederfrequenten Anteile viele Iterationen erfordert.

Die Definition, was hoch- und niederfrequent oder kurz- und langwellig ist, hängt allerdings von der Gitterschrittweite ab. So sind die langwelligen Anteile von dem feinen Gitter – wir definieren diese z. B. als die Änderungen gemittelt über 20 Gitterpunkte – kurzwellig auf dem groben Gitter, da sie hier z. B. über 5 Gitterpunkte approximiert werden. Jetzt liegt die Idee auf der Hand. Man führt verschieden feine Gitter ein und approximiert immer nur die kurzwelligen Anteile auf dem entsprechenden Gitter. Dies geht jeweils in einigen wenigen Iterationen recht schnell. Das Ganze muss dann nur noch richtig zusammengesetzt werden.

Wir schauen uns die einfachste Situation an, wo man von dem gröbsten Gitter startet und bis zum feinsten Gitter fortschreitet, was gerne mit geschachtelter Iteration bezeichnet wird. Neben den verschiedenen Gittern $G_1, G_2, \ldots, G_q$ benötigt man noch eine Interpolation von einem Gitter zu dem feineren Gitter. Man nennt dies Prolongation und definiert die Prolongationsoperatoren

$$P_{i,i+1} : \; G_i \rightarrow G_{i+1}$$

vom Gitter G_i nach dem Gitter G_{i+1}. Man startet auf dem gröbsten Gitter G_1 mit einer direkten Lösung des Gleichungssystems. Dies kann man sich erlauben, das hier die exakte Lösung, etwa mit dem Gauß-Algorithmus, noch schnell zu berechnen ist. Diese Lösung nennen wir $\mathbf{u}_1$ und die Prolongation auf das nächst feinere Gitter dann mit $\bar{\mathbf{u}}_2$. Mit dieser Startlösung wird dann das Problem iterativ auf diesem Gitter gelöst. Man benötigt nur einige wenige Schritte, da nur noch die auf diesem Gitter kurzwelligen Anteile eingefangen werden müssen. Dann geht man zum nächsten feineren Gitter weiter. Das gesamte

Verfahren sieht dann wie folgt aus:

$$
\begin{aligned}
&\text{Löse auf } G_1 \text{ exakt} \\
&\quad \text{Resultat: } \mathbf{u}_1 \\
&\Downarrow \qquad\qquad \bar{\mathbf{u}}_2 = P_{1,2}\,\mathbf{u}_1 \\
&\text{Löse auf } G_2 \text{iterativ, Start: } \bar{\mathbf{u}}_2 \\
&\quad \text{Resultat: } \mathbf{u}_2 \\
&\Downarrow \qquad\qquad \bar{\mathbf{u}}_3 = P_{2,3}\,\mathbf{u}_2 \\
&\vdots \\
&\Downarrow \qquad\qquad \bar{\mathbf{u}}_q = P_{q-1,q}\,\mathbf{u}_{q-1} \\
&\text{Löse auf } G_q \text{iterativ, Start: } \bar{\mathbf{u}}_q \\
&\quad \text{Resultat: } \mathbf{u}_q
\end{aligned}
$$

Dieses Vorgehen nennt man dann geschachtelte Iteration oder „Nested Iteration“. Bei der iterativen Lösung auf jedem Gitterlevel benötigt man nur einige wenige Iterationen, da nur die auf diesem Gitter kleinskaligen Komponenten eingefangen werden müssen.

Bei einem Mehrgitterverfahren wird diese geschachtelte Iteration mit einem Vorgang in umgekehrter Richtung kombiniert, vom feinsten Gitter auf das gröbste Gitter. Neben den Prolongationsoperatoren benötigt man nun noch den **Restriktionsoperator**

$$R_{i,i-1} : \; G_i \to G_{i-1} \; ,$$

der eine Näherung auf G_i nach G_{i-1} bringt und man benötigt die Koeffizientenmatrizen $\mathbf{A}_i$ der Gleichungssysteme für die entsprechenden Gitter G_i. Starten wir zunächst auf dem feinsten Gitter G_q. Nach einigen wenigen Iterationen mit dem Jakobi-Verfahren sind die auf diesem Gitter kleinskaligen Anteile eingefangen. Die Lösung zu diesem Zeitpunkt nennen wir $\tilde{\mathbf{u}}_q$. Das Residuum

$$\mathbf{r}_q = b - \mathbf{A}_q\,\tilde{\mathbf{u}}_q$$

enthält somit keine nennenswerten kleinskaligen Anteile mehr. Dann kann man aber auf das vergröberte Gitter gehen und mittelt das Residuum:

$$\tilde{\mathbf{r}}_{q-1} = R_{q,q-1}\,\mathbf{r}_q \; .$$

Die Korrektur der Lösung ergibt sich durch die Lösung des Gleichungssystems

$$\mathbf{A}_{q-1}\,\mathbf{e} = \tilde{\mathbf{r}}_{q-1}\ .$$

Die Lösung nach einigen wenigen Iterationen nennen wir $\mathbf{e}_{q-1}$ und berechnen das zugehörige Residuum nach

$$\mathbf{r}_{q-1} = \tilde{\mathbf{r}}_{q-1} - \mathbf{A}_{q-1}\,\mathbf{e}_{q-1}\ .$$

Jetzt gilt das gleiche Argument wie oben. Die kleinskaligen Anteile jetzt auf dem Gitter G_{q-1} sind eingefangen, so dass man dieses Residuum wieder auf das gröbere Gitter mittelt und dort einige wenige Iterationen durchführt. Schrittweise kommt man dann an bei

$$\tilde{\mathbf{r}}_1 = R_{2,1}\,\mathbf{r}_2\ .$$

Die Gleichung

$$\mathbf{A}_1\,\mathbf{e} = \tilde{\mathbf{r}}_1\ .$$

wird dann exakt gelöst. Man erhält die Korrektur $\mathbf{e}_1$.

Um zu der Lösung auf dem feinsten Gitter zu kommen, muss dann der Vorgang wie bei der „Nested Iteration“ von dem gröbsten zu dem feinsten Gitter ablaufen, indem man die Korrekturen aufaddiert. Dabei muss man die Anteile auf dem gröberen Gitter in jedem Schritt auf das feinere interpolieren, d. h. prolongieren. Von $i = 1, 2, \ldots, q-2$ ergibt sich dann

$$\hat{\mathbf{e}}_{i+1} = \mathbf{e}_{i+1} + P_{i,i+1}\,\tilde{\mathbf{e}}_i\ .$$

Dabei wurde hier die Korrektur auf dem i-ten Gitter $\mathbf{e}_i$ durch $\tilde{\mathbf{e}}_i$ ersetzt. Diese Modifikation ergibt sich wegen der Tatsache, dass durch die Prolongation kleinskalige Fehler erneut eingeschleppt werden können. Diese werden eliminiert, indem man wieder einige wenige Schritte des Iterationsverfahrens durchgeführt. Es ergibt sich $\tilde{\mathbf{e}}_i$ nach einigen wenigen Iterationen des LGS

$$\mathbf{A}_{i+1}\,\mathbf{e} = \tilde{\mathbf{r}}_{i+1}$$

mit Start bei $\hat{\mathbf{e}}_{i+1}$.

Ist man bei $i = q-2$ angekommen, ergibt sich

$$\hat{\mathbf{e}}_q = P_{q-1,q}\,\tilde{\mathbf{e}}_{q-1}, \quad \mathbf{u}_q = \tilde{\mathbf{u}}_q + \hat{\mathbf{e}}_q$$

als Lösung auf dem Gitter G_q.

Das Mehrgitterverfahren enthält sehr viele verschiedene Elemente, welche optimal an das vorgegebene Problem angepasst werden können. Das sind zum einen die Prolongation und zum anderen die Restriktion, aber auch das iterative Verfahren zur Lösung des Gleichungssystems. Da dies dazu eingesetzt wird, die kleinskaligen Änderungen schnell zu approximieren und das Residuum dann nur noch die großskaligen Fehler enthält, spricht man in diesem Zusammenhang auch von einem Glätter. Als sogenannte Glättungsverfahren kommen die klassischen Iterationsverfahren, allen voran das Jacobi- oder das Gauß-Seidel-Verfahren zum Einsatz. Um die Reduktion der hochfrequenten Fehleranteile zu erreichen, wird das sogenannte gedämpfte Jacobi-Verfahren benutzt mit einem Relaxationsparameter $\omega < 1$.

Oben haben wir einen sogenannten V-Zyklus angegeben: Von dem feinsten Gitter wird zum gröbsten Gitter gegangen und dann wieder zurück. Ein W-Zyklus ergibt sich, wenn man auf dem feinsten Gitter startet und nach Erreichen des gröbsten Gitters nicht bis zum feinsten Gitter zurück geht, sondern nach einigen Schritten nochmals auf das gröbste Gitter geht und dann erst auf das feinste zurück. Die Struktur dieses Ablaufes sieht dann wie ein W aus. Eine Kombination von W- und V-Zyklus nennt man den F-Zyklus.

C.3 Das Verfahren der konjugierten Gradienten

1. Das Richardson-Iterationsverfahren Die klassischen Iterationsverfahren des vorherigen Abschnitts lassen sich auch in der Form

$$\mathbf{u}^{(m+1)} = \mathbf{u}^{(m)} + \alpha \mathbf{B}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{u}^{(m)}) \tag{C.19}$$

mit einer regulären Matrix $\mathbf{B}$ und einem Relaxationsparameter $\boldsymbol{\alpha}$ darstellen. So ergibt sich das Jacobi- und das SOR-Verfahren mit den folgenden Definitionen:

$$\mathbf{B}_J = \mathbf{D}, \quad \alpha_j = 1 \quad \text{bzw.} \quad \mathbf{B}_{\text{SOR}} = \omega \mathbf{L} + \mathbf{D}, \quad \alpha_{\text{SOR}} = \omega \, . \tag{C.20}$$

Im Falle der Konvergenz ergibt sich aus (C.19)

$$\mathbf{u} = \mathbf{u} + \alpha \mathbf{B}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{u}) \, . \tag{C.21}$$

Man löst somit als Ausgangsgleichung eigentlich

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{u} = \mathbf{B}^{-1}\mathbf{b} \, . \tag{C.22}$$

Man nennt die Matrix $\mathbf{B}$ auch die Vorkonditionierungsmatrix oder kurz den Vorkonditionierer im Englischen „Preconditioner“. Die zugehörige Iterationsvorschrift wird als **vorkonditioniertes Richardson-Verfahren** bezeichnet.

Der Vorkonditionierer $\mathbf{B}$ sollte so gewählt werden, dass das LGS $\mathbf{B}\,\mathbf{s} = \mathbf{r}$ einfacher als $\mathbf{A}\,\mathbf{u} = \mathbf{b}$ berechnet werden kann. Bei den klassischen Iterationsverfahren ist dies nach

(C.20) erfüllt. Bei der Wahl $\mathbf{B} = \mathbf{A}$ und $\alpha = 1$ hätte man die exakte Lösung in einem Schritt. Die Matrix $\mathbf{B}$ sollte $\mathbf{A}$ somit möglichst gut approximieren.

Oft eingesetzt wird die unvollständige Cholesky-Faktorisierung. Die Cholesky-Faktorisierung zerlegt $\mathbf{A}$ in das Produkt $\mathbf{A} = \mathbf{LU}$. Bei den schwach besetzten Matrizen wird dies nur unvollständig ausgeführt, indem nur Elemente berechnet werden, für die $a_{ij} \neq 0$ gilt. Andernfalls würde die Matrix aufgefüllt werden. Wir verweisen hier auf die Spezialliteratur.

Bemerkung Beim Richardson-Verfahren wird die Matrix nur bei der Berechnung des Residuums benötigt. Es genügt im Programm, das Matrix-Vektor-Produkt mit der schwach besetzten Matrix $\mathbf{A}$ zu berechnen. In dem numerischen Verfahren zur Lösung von elliptischen Randwertproblemen liegt dies mit dem System der Differenzengleichungen vor.

Der Einfachheit halber betrachten wir im folgenden die Richardson-Iteration ohne Vorkonditionierung:

$$\mathbf{u}^{(m+1)} = \mathbf{u}^{(m)} + \alpha(\mathbf{b} - \mathbf{A}\mathbf{u}^{(m)}) \; . \tag{C.23}$$

Betrachtet man die ersten zwei Iterierten, so erhält man

$$\mathbf{u}^{(1)} = \mathbf{u}^{(0)} + \alpha\mathbf{r}^{(0)} \; , \tag{C.24}$$

$$\mathbf{u}^{(2)} = \mathbf{u}^{(1)} + \alpha(\mathbf{b} - \mathbf{A}\mathbf{u}^{(1)}) \tag{C.25}$$

$$= \mathbf{u}^{(0)} + \alpha\mathbf{r}^{(0)} + \alpha(\mathbf{b} - \mathbf{A}\mathbf{u}^{(0)}) \tag{C.26}$$

$$= \mathbf{u}^{(0)} + \alpha\mathbf{r}^{(0)} + \alpha(\mathbf{b} - \mathbf{A}\mathbf{u}^{(0)} - \alpha\mathbf{A}\mathbf{r}^{(0)}) \tag{C.27}$$

$$= \mathbf{u}^{(0)} + 2\alpha\mathbf{r}^{(0)} - \alpha^2\mathbf{A}\mathbf{r}^{(0)} \; . \tag{C.28}$$

Allgemein ergibt sich für m-te Iterierte eine Darstellung in der Form

$$\mathbf{u}^{(m)} = \mathbf{u}^{(0)} + \gamma_1^{(m)}\mathbf{r}^{(0)} + \gamma_2^{(m)}\mathbf{A}\mathbf{r}^{(0)} + \cdots + \gamma_m^{(m)}\mathbf{A}^{m-1}\mathbf{r}^{(0)} \; , \tag{C.29}$$

mit den Koeffizienten $\gamma_i = \gamma_i(\alpha)$. Die m-te Iterierte ist somit korrigiert durch eine Linearkombination der Vektoren $\mathbf{A}^j\mathbf{r}^{(0)}, \quad j = 0, 1, \ldots, m-1$.

Man nennt den Teilraum, welcher durch die Vektoren $\mathbf{A}^j\mathbf{r}^{(0)}$ aufgespannt wird:

$$K_{\mathbf{m}}(\mathbf{r^{(0)}}; \mathbf{A}) = \text{span}\,\{\mathbf{r^{(0)}}, \mathbf{A}\mathbf{r^{(0)}}, \ldots, \mathbf{A}^{m-1}\mathbf{r}^{(0)}\} \; , \tag{C.30}$$

den **Krylov-Teilraum** der Ordnung m. Kurz schreibt man für diesen Sachverhalt

$$\mathbf{u^{(m)}} \in \mathbf{u^{(0)}} + K_{\mathbf{m}}(\mathbf{r^{(0)}}, \mathbf{A}) \; . \tag{C.31}$$

In der Iterationsvorschrift (C.23) werden die Koeffizienten $\gamma_i^{(m)}$ durch das fest vorgegebene α bestimmt. Es ist jetzt die Frage, wie dieses α am besten gewählt wird, dass das Verfahren schnell konvergiert. Man benötigt ein Kriterium für den Fehler und versucht diesen durch die Wahl von α zu minimieren.

1. Das CG-Verfahren

Ein Kriterium für die Minimierung des Fehlers wird im Folgenden für den Spezialfall eines symmetrischen, positiv definiten Systems ausgeführt. Das Verfahren nennt man das **Verfahren der konjugierten Gradienten (CG-Verfahren)**.

Die Matrix $\mathbf{A}$ sei symmetrisch und positiv definit. Wir betrachten das quadratische Funktional

$$F(\mathbf{v}) := \frac{1}{2}(\mathbf{v}, \mathbf{Av}) - (\mathbf{b}, \mathbf{v}) \ , \tag{C.32}$$

wobei (.,.) das Skalarprodukt bezeichnet:

$$(\mathbf{v}, \mathbf{Av}) = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} v_i v_k \ , \tag{C.33}$$

$$(\mathbf{b}, \mathbf{v}) = \sum_{i=1}^{n} b_i v_i \ . \tag{C.34}$$

Es zeigt sich, dass das Auffinden der Lösung $\mathbf{u}$ des LGS $\mathbf{Au} = \mathbf{b}$ äquivalent mit der Minimierungsaufgabe

$$\mathbf{u} = \min_{\mathbf{v}} \mathbf{F}(\mathbf{v}) \tag{C.35}$$

ist. Dies sieht man folgendermaßen.

Eine notwendige Bedingung für ein lokales Extremum ist, dass der Gradient $\nabla F(\mathbf{v})$ Null wird. Mit

$$\frac{\partial F}{\partial v_i} = \sum_{k=1}^{n} a_{ik} v_k - b_i \tag{C.36}$$

ergibt sich dieser zu

$$\nabla F(\mathbf{v}) = \mathbf{Av} - \mathbf{b} =: \mathbf{r} \ . \tag{C.37}$$

Da die zweite Ableitung, die Hesse-Matrix von $F(\mathbf{v})$, gerade $\mathbf{A}$ ist und diese Matrix nach Voraussetzung positiv definit, ist die Lösung $\mathbf{u}$ des LGS die Stelle des Minimums. Es zeigt sich, dass diese eindeutig ist und es sich um das globale Minimum handelt. Die Lösung des Gleichungssystems $\mathbf{Au} = \mathbf{b}$ ist somit äquivalent zu dem Finden des Minimums von $F(\mathbf{v})$. Damit ist ein Kriterium zur Minimierung des Fehlers gefunden.

Beim CG-Verfahren berechnet man eine Basis $\{\mathbf{q^1}, \mathbf{q^2}, \ldots, \mathbf{q^m}\}$ des Krylov-Unterraums $K_{\mathbf{m}}$, für die

$$(\mathbf{q}^i, \mathbf{Aq}^j) = 0 \quad \text{für} \quad i \neq j \tag{C.38}$$

gilt. Diese Eigenschaft heißt **A**-orthogonal oder konjugiert bezüglich **A**. Der Ansatz im m-ten Iterationsschritt ist

$$\mathbf{q}^{(\mathbf{m})} = -\mathbf{r}^{(\mathbf{m-1})} + \boldsymbol{\beta}\mathbf{q}^{(\mathbf{m-1})} . \tag{C.39}$$

Der Wert von β ergibt sich aus der Bedingung der Konjugiertheit von $\mathbf{q}^{(m)}$ und $\mathbf{q}^{(m-1)}$ bezüglich **A**. Den Wert von α erhält man dann aus der Minimierungseigenschaft. Sind **v** und **q** fest, wird $\mathbf{v}' = \mathbf{v} + \boldsymbol{,}\mathbf{q}$ gesucht mit

$$F(\mathbf{v}') = \min_{\alpha} F(\mathbf{v} + \boldsymbol{\alpha}\mathbf{q}) . \tag{C.40}$$

Nach kurzer Rechnung erhält man

$$F(\mathbf{v} + \boldsymbol{\alpha}\,\mathbf{q}) = \frac{1}{2}(\mathbf{v} + \boldsymbol{\alpha}\,\mathbf{q}, \mathbf{A}(\mathbf{v} + \boldsymbol{\alpha}\,\mathbf{q})) - (\mathbf{b}, \mathbf{v} + \boldsymbol{\alpha}\,\mathbf{q}) \tag{C.41}$$

$$= \frac{1}{2}(\mathbf{v}, \mathbf{A}\mathbf{v}) + \alpha\,(\mathbf{q}, \mathbf{A}\mathbf{v}) + \frac{1}{2}\alpha^2\,(\mathbf{q}, \mathbf{A}\mathbf{q}) - (\mathbf{b}, \mathbf{v}) \tag{C.42}$$

$$- \alpha\,(\mathbf{b}, \mathbf{q}) \tag{C.43}$$

$$= \frac{1}{2}\alpha^2\,(\mathbf{q}, \mathbf{A}\mathbf{q}) + \alpha\,(\mathbf{q}, \mathbf{r}) + F(\mathbf{v}) =: F^*(\alpha) . \tag{C.44}$$

Das Nullsetzen der Ableitung nach α:

$$\frac{d}{d\alpha} F^*(\alpha) = \alpha\,(\mathbf{q}, \mathbf{A}\mathbf{q}) + (\mathbf{q}, \mathbf{r}) = 0 \tag{C.45}$$

liefert für α den Wert

$$\alpha_{\min = -\frac{(q,r)}{(q,Aq)}} \tag{C.46}$$

Die zweite Ableitung nach α ist positiv, da **A** positiv definit ist.

Der Rechenaufwand für einen Iterationsschritt ergibt sich aus der schon erwähnten Matrix-Vektor-Multiplikation, aus zwei Skalarprodukten und drei skalaren Multiplikationen von Vektoren. Neben der Matrix **A** werden nur $4n$ Speicherplätze für $\mathbf{q}^{(m)}$, $\mathbf{r}^{(m)}$, $\mathbf{u}^{(m)}$ und **s** benötigt. Eine Vorkonditionierung beim CG-Verfahren ist möglich, indem **A** und **b** durch $\mathbf{B^{-1}A}$ und $\mathbf{B^{-1}b}$ ersetzt werden.

3. Methode der verallgemeinerten Residuen

Wenn **A** unsymmetrisch ist, versagt das CG-Verfahren. Es gibt hier die Möglichkeit, das unsymmetrische LGS äquivalent in ein System mit einer symmetrischen, positiv definiten Matrix umzuformulieren:

$$\mathbf{A}^T\mathbf{A}\mathbf{v} = \mathbf{A}^T\mathbf{b} \tag{C.47}$$

und das CG-Verfahren auf dieses System anzuwenden. Allerdings verschlechtert sich hier die Konditionszahl – man erhält das Quadrat der Konditionszahl von **A**. Dieser Ansatz ist somit nur für gut konditionierte LGS sinnvoll.

Ein anderer Ansatz ist die Euklidische Norm des Residuums

$$\| \mathbf{b} - \mathbf{A}\mathbf{v} \|_2^2 \tag{C.48}$$

über dem Krylov-Teilraum K_m zu minimieren. Dieses Verfahren wird GMRES (Generalized Minimum Residual) genannt. Man bestimmt in K_m eine Basis mit dem Gram-Schmidt-Verfahren. Ein Nachteil des GMRES-Verfahrens ist, dass die ganze Basis $\{\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(m)}\}$ abgespeichert werden muss und im Laufe der Iteration der Speicherplatz anwächst. Um dies zu vermeiden, bricht man nach N Schritten ab und startet mit der letzten Iterierten erneut. Dieser Restart verschlechtert natürlich die Eigenschaften des Verfahrens. Sinnvolle Werte für N liegen zwischen 6 und 20.

Es gibt eine ganze Reihe weiterer Krylov-Teilraum-Verfahren. Wir verweisen hier auf die Literatur, welche im nächsten Abschnitt aufgeführt wird.

C.4 Bemerkungen und Entscheidungshilfen

Weitere Informationen zu den Iterationsverfahren für große schwach besetzte Gleichungssysteme findet man in den Büchern über numerische Methoden wie [6] und Stoer [12], ausführlicher dann in der Spezialliteratur über die numerische Lösung von Gleichungssystemen, z. B. bei Hackbusch [13], Meister [14] und Golub und van Loan [15]. Den Aspekt der effizienten Ausführung dieser Verfahren auf Parallelrechnern wird von Schwandt [16] behandelt.

Die klassischen Iterationsverfahren, allen voran das SOR-Verfahren, werden auch heute noch für kleine Probleme eingesetzt. Ist die Anzahl der Gitterpunkte groß, dann steigt die Anzahl der Iterationen beträchtlich an und diese Verfahren werden zu langsam. Die Bücher von Varga [17] und Young [18] sind die klassischen Werke über diese iterativen Verfahren.

Die klassischen Iterationsverfahren, allen voran das SOR-Verfahren, werden auch heute noch für kleine Probleme eingesetzt. Ist die Anzahl der Gitterpunkte groß, dann steigt die Anzahl der Iterationen beträchtlich an und diese Verfahren werden zu langsam. Die Bücher von Varga [17] und Young [18] sind die klassischen Werke über diese iterativen Verfahren.

Für sehr große Probleme schneiden die Mehrgitterverfahren bezüglich der Rechenzeit oft am besten ab. Wie beschrieben gibt es bei diesen Verfahren eine ganze Reihe von Freiheitsgraden: Restriktion, Prolongation, Iterations-Verfahren und Art des Zyklus: V, W oder F, welche auf das entsprechende Problem optimiert werden können. Für einen Nicht-Fachmann ergibt sich daraus die Schwierigkeit, dass die beste Auswahl dieser Komponenten einiges an Erfahrung erfordert. Für die Standardprobleme, wie die Poisson-Gleichung,

gibt es hier optimierte Versionen in der Literatur oder in Programmbibliotheken, die in Bezug auf Rechenzeit meist nicht zu schlagen sind. Für Nicht-Standardprobleme können aber die Schwierigkeiten auftreten, dass der Einsatz dieser Verfahren und die Optimierung etwas Zeit erfordert. Von der Programmstruktur her ist ein Mehrgitterverfahren recht kompliziert. Eine eigene Programmentwicklung setzt hier schon eine gute Kenntnis und Erfahrung im Programmieren voraus. Ein Tutorial zur Entwicklung eines Mehrgitterverfahrens ist das Buch von Briggs, Henson und McCormick [19]. Weitere Beschreibungen dieser Verfahren finden sich in der oben angeführten Literatur. Es gibt darüber hinaus Monographien speziell über Mehrgitterverfahren, z. B. von Hackbusch [20] und Wesseling [21].

Das CG-Verfahren wurde als direktes Verfahren entwickelt. Insbesondere durch die Vorkonditionierung des Gleichungssystems hat es sich dann später als ein sehr robustes und schnelles iteratives Verfahren gezeigt. Es braucht im Allgemeinen weniger Rechenaufwand als das SOR-Verfahren. Die Matrix $\mathbf{A}$ muss bei dem CG-Verfahren symmetrisch und positiv definit sein. Wie schon oben erwähnt wurden in den letzten Jahren ähnliche Methoden angegeben, bei denen diese Voraussetzungen abgeschwächt sind. Bei sehr großen Systemen und den Standardproblemen ist ein Mehrgitterverfahren in Bezug auf Rechenzeiten einer Krylov-Unterraum-Methode im Allgemeinen überlegen. Allerdings ist die Krylov-Unterraum-Methode mit einer guten Vorkonditionierung sehr robust und unproblematisch bei dem Einsatz auf ein Nicht-Standardproblem. Es muss kein Parameter eingestellt werden. Mehrgitterverfahren können hier auch zur Vorkonditionierung eingesetzt werden. Möchte man sehr große Probleme lösen und dies öfters, dann wird sich der Aufwand lohnen, ein Mehrgitterverfahren geeignet anzupassen. Eine Übersicht über Krylov-Unterraum-Methoden geben die Bücher von Hackbusch [13], Meister [14] und Axelsson [22].

Bei den klassischen Iterationsverfahren wurde diskutiert, wie es vermieden werden kann, dass die Koeffizientenmatrix des Gleichungssystems abgespeichert wird. Bei den anderen Iterationsverfahren wird als zentraler Baustein die Matrix-Vektor-Multiplikation benötigt. Insofern geht es im Allgemeinen darum, diese Operation geschickt abzuspeichern. Es werden hier auf einen Vektor nacheinander nur die Elemente $\neq 0$ abgespeichert. Es werden dann Hilfsfelder eingeführt, welche die Position der Nicht-Null-Koeffizienten, eindeutig beschreiben. Dadurch wird der Speicherplatzbedarf proportional zu der Anzahl der Unbekannten.

Dadurch, dass viele Probleme auf die Lösung großer linearer Gleichungssysteme zurückgeführt werden können, gibt es in diesem Bereich eine ganze Reihe von Programmen. Die Computer-Algebra-Systeme wie MAPLE oder das Programmpaket MATLAB haben die entsprechende Software zur Verfügung. Die Programmpakete wie IMSL und NAG bieten diesbezüglich viele Routinen an. Berühmt ist die Programmesammlung LAPACK zur linearen Algebra. Viele der IMSL- und NAG-Programme gehen auf diese Sammlung zurück.

Da viele Probleme auf die Lösung großer linearer Gleichungssysteme zurückgeführt werden können, gibt es in diesem Bereich eine ganze Reihe von Programmen. MATLAB

stellt hier z. B. entsprechende Software zur Verfügung. Die Programm-Pakete wie IMSL [10] und NAG [11] bieten diesbezüglich viele Routinen an. Berühmt ist die Programmesammlung LAPACK zur linearen Algebra. Viele der IMSL- und NAG-Programme gehen auf diese Sammlung zurück. Das Suchen von frei erhältlichen Programmen unterstützt z. B. http://www.netlib.org.

Literatur

1. K. Förster, Numerische Behandlung von Differentialgleichungen, RUS-26, Rechenzentrum Universität Stuttgart, Stuttgart, 1995
2. P. Deuflhard und A. Hohmann, Numerische Mathematik I, De Gruyter Verlag, Berlin, 3. Aufl., 2002
3. G. Engeln-Müllges und F. Reutter, Numerische Mathematik
4. H.-G. Roos und H. Schwetlick, Numerische Mathematik, Teubner-Verlag, Stuttgart, Leipzig, 1999
5. J. Stoer, Einführung in die Numerische Mathematik I, Springer-Verlag, Berlin, Heidelberg, New York, 9. Aufl., 2002
6. H.-R. Schwarz und N. Köckler, Numerische Mathematik, Teubner-Verlag, Stuttgart, Leipzig, Wiesbaden, 5. Aufl., 2004
7. C. DeBoor, A Practical Guide to Splines, Springer-Verlag, Berlin, Heidelberg, New York, 2. Aufl., 2001
8. W. H. Press, S. A. Teukolsky, W. T. Vetterling und B. P. Flannery, Numerical Recipes in Fortran 77: The Art of Scientific Computing, Cambridge University Press, Cambridge, 2001
9. G. Engeln-Müllges und F. Reutter, Numerik Algorithmen. Entscheidungshilfe zur Auswahl und Nutzung, VDI-Verlag, 8. Aufl., 1996
10. Visual Numerics, IMSL Numerical Libraries, www.roguewave.com
11. NAG Numerical Algorithms Group, NAG Numerical Components, www.nag.com
12. J. Stoer und R. Bulirsch, Einführung in die Numerische Mathematik II, Springer-Verlag, Berlin, Heidelberg, New York, 4. Aufl., 2000
13. W. Hackbusch, Iterative Löser großer schwach besetzter Gleichungssysteme, Teubner-Verlag, Stuttgart, 1991
14. A. Meister, Numerik linearer Gleichungssysteme, Vieweg-Verlag, Braunschweig, Wiesbaden, 2. Aufl., 1999
15. G. H. Golub und C. van Loan, Matrix Computations, John Hopkins University Press, Baltimore, 3. Aufl., 1996
16. H. Schwandt, Parallele Numerik, Teubner-Verlag, Stuttgart, Leipzig, Wiesbaden, 2003
17. R. S. Varga, Matrix Iterative Analysis, Springer-Verlag, Berlin, Heidelberg, New York, 2. Aufl., 2000
18. D. M. Young, Iterative Solutions of Large Linear Systems, Academic Press, New York, 1971
19. W. Briggs, V. Henson und S. McCormick, A Multigrid Tutorial, SIAM, Philadelphia, 2000
20. W. Hackbusch, Multigrid Methods and Applications, Springer-Verlag, Berlin, Heidelberg, New York, 1985
21. P. Wesseling, An Introduction to Multigrid Methods, Wiley, Chichester, 1992
22. O. Axelsson, Iterative Solution Methods, Cambridge University Press, Cambridge, 1996

Sachverzeichnis