Karl Hainer

Numerische Algorithmen auf programmierbaren Taschenrechnern

B·I·
Hochschultaschenbücher
Band 805

B.I.-Hochschultaschenbücher
Band 805

Numerische Algorithmen auf programmierbaren Taschenrechnern

von
Dr. Karl Hainer
Akademischer Oberrat an der Universität Frankfurt

Bibliographisches Institut Mannheim/Wien/Zürich
B. I.-Wissenschaftsverlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Hainer, Karl:
Numerische Algorithmen auf programmierbaren Taschenrechnern / von Karl Hainer. - Mannheim, Wien, Zürich : Bibliographisches Institut, 1980.
(BI-Hochschultaschenbücher; Bd. 805)
ISBN 3-411-00805-9

Druck und Bindearbeit: Hain-Druck GmbH, Meisenheim/Glan
Printed in Germany
ISBN 3-411-00805-9

Vorwort

Die heute verfügbaren, außerordentlich leistungsfähigen wissenschaftlichen Taschenrechner gestatten die Lösung von Problemen der numerischen Datenverarbeitung. Zu diesem Zweck gibt dieses Buch eine Einführung in die Grundaufgaben der Numerischen Mathematik sowie die Aufstellung der zugehörigen Lösungsalgorithmen, und es wird gezeigt, wie diese Algorithmen auf die UPN-Rechner HP-67/97 und die AOS-Rechner TI-58/59 übertragen werden können.

Die Abschnitte des Buches sind einheitlich gegliedert. Zunächst werden die Aufgabenstellung und das jeweilige Lösungsverfahren erläutert. Dann wird unter der Bezeichnung "Algorithmus" ein Flußdiagramm für die Lösungsmethode angegeben. Hierbei werden nur lineare Flußdiagramme verwendet, so daß die Übersicht über den Rechenablauf bereits in der Form vorliegt, wie sie für die Programmierung benötigt wird. Es folgt die Übertragung des Algorithmus in je ein Programm für die Rechner HP-67/97 und die Rechner TI-58/59 mit einander entsprechender Programmstruktur. Eine Anleitung zur Verwendung der Programme schließt sich an. Häufig sind Anmerkungen zum Programmaufbau angefügt, in denen auch auf die Eigenheiten der umgekehrt polnischen Notation (UPN) und des algebraischen Operations-Systems (AOS) eingegangen wird. Jeder Abschnitt endet mit numerischen Beispielen.

Die Programme sind ausführlich dokumentiert: Vorangestellt ist jeweils eine vollständige Übersicht über die Belegung des Datenspeichers; die Programme selbst sind nicht nur als Auflistung der Taschenrechner-Anweisungen wiedergegeben, sondern abweichend von den Programmformularen der Herstellerfirmen Hewlett-Packard und Texas Instruments wird eine übersichtliche Notation verwendet, mit der auch bei dem Taschenrechnerprogramm die Struktur des Lösungsalgorithmus deutlich wird. Damit kann der Leser die Wirkungsweise jedes Programms nachvollziehen, und er erhält gleichzeitig Anleitungen zum Erstellen und Über-

prüfen eigener Programme. Auch gibt die Gegenüberstellung der Programme in umgekehrt polnischer Notation und im algebraischen Operations-System Hinweise für die Übertragung von Programmen aus einer Rechner-Logik in die andere. Viele der angegebenen Programme könnten noch kürzer gefaßt werden hinsichtlich der Anzahl der benötigten Programmschritte, der Datenregister oder im Blick auf die erforderliche Rechenzeit; mit Rücksicht auf die angestrebte Übersichtlichkeit wurde davon jedoch Abstand genommen.

Die Programme können übertragen werden auf andere programmierbare Taschenrechner mit Unterprogrammtechnik und indirekter Adressierung. Insbesondere die Programme für die Rechner HP-67/97 können über Magnetkarten unverändert auf dem alphanumerischen Rechner HP-41C gerechnet werden.

In diesem Buch wird davon ausgegangen, daß der Leser mit den wesentlichen Teilen der Bedienungsanleitung für einen der Rechner HP-67/97 oder TI-58/59 vertraut ist. Als Zeichen für die Division dient in den Programmen für die Taschenrechner das darauf gebräuchliche Symbol ÷, sonst treten nur Bruchstriche auf; der Doppelpunkt : wird in diesem Buch nirgends als Divisionszeichen verwendet, vielmehr kennzeichnet er in den Flußdiagrammen den Vergleich zweier Zahlen bei Verzweigungen.

Kapitel 1 führt an Beispielen ein in die Methoden beim Aufstellen von Algorithmen und zugehörigen Programmen für die Rechner HP-67/97 und TI-58/59. Abschnitt 1.3 enthält Ausführungen zum Schneiden und Runden von Zahlen sowie ein Programm zur 13-stelligen Anzeige von Ergebnissen der Rechner TI-58/59.

Ausgehend von einfachen Algorithmen und Programmen werden ab Kapitel 2 die Grundaufgaben der Numerischen Mathematik behandelt. Nach dem Horner-Schema für Polynome und der näherungsweisen Berechnung unendlicher Reihen in Kapitel 2 folgen in Kapitel 3 Methoden zur Ermittlung von Nullstellen reeller Funktionen. Interpolationspolynome und dividierte Differenzen

werden in Kapitel 4 aus einem Dreieckschema berechnet, das auch bei der numerischen Integration in Kapitel 5 Verwendung findet. Im Hinblick auf die beschränkten Speichermöglichkeiten bei Taschenrechnern ist es von Bedeutung, daß von diesem Zahlenschema jeweils nur eine Spalte gespeichert werden muß. Die Methoden zur Ermittlung der Ausgleichsgeraden in Kapitel 6 sind Beispiele gegliederter Algorithmen, wo Rechnungen bereits vor Abschluß der Dateneingabe ausgeführt werden. Die Ausgleichsgerade wird zunächst mit Hilfe der Tastenfunktion Σ+ berechnet, wie es von den Herstellern der Taschenrechner vorgeschlagen wird; da an Beispielen jedoch gezeigt wird, daß dieses Vorgehen anfällig gegen Auslöschungseffekte ist, wird eine andere, numerische stabile Berechnungsmöglichkeit für die Ausgleichsgerade erläutert. Kapitel 7 bringt Ein- und Mehrschrittverfahren für Anfangswertaufgaben gewöhnlicher Differentialgleichungen erster Ordnung. Mit diesen Methoden können auch gekoppelte Systeme von zwei Differentialgleichungen erster Ordnung auf den Taschenrechnern behandelt werden, ebenso Anfangswertaufgaben für gewöhnliche Differentialgleichungen zweiter Ordnung, wenn sie als ein solches System geschrieben werden; aus Platzgründen mußte darauf jedoch verzichtet werden.

In Kapitel 8 wird ausführlich auf die Organisation des Datenspeichers der Rechner eingegangen, wenn ein oder mehrere Zahlenfelder gespeichert werden sollen. Nach einem Rückblick auf die Methoden, die bereits in den vorangehenden Kapiteln bei der Speicherung eines Zahlenfeldes eingesetzt wurden, wird die Speicherung von Vektoren und Matrizen vom Ende des verfügbaren Datenspeichers her besprochen. Programmschleifen treten bei Rechnungen mit Vektoren und Matrizen häufig auf; sie werden von nun an mit Hilfe der Dsz-Anweisung und des komplementären Schleifenindex kontrolliert. Zur Erläuterung dieser Vorgehensweisen dient in Abschnitt 9.1 die Berechnung von Defektvektoren linearer Gleichungssysteme; dort wird bei der Eingabe der Zahlenfelder vor dem Eintasten jeder Komponente die Nummer des zugehörigen Datenregisters angezeigt. Von

Abschnitt 9.2 an sind dann die Einleseprogramme für Zahlenfelder durchgehend so gestaltet, daß eine einfache Überprüfung und Berichtigung der eingegebenen Werte möglich ist durch die Wiederholung der Anweisungsfolge der Eingabe. Auch wird bei Programmen mit Zahlenfeldern jeweils für die Rechner HP-67/97, TI-58 und TI-59 angegeben, wie groß die Zahlenfelder auf Grund des zur Verfügung stehenden Datenspeichers gewählt werden können. So können die Programme für das Gaußsche Eliminationsverfahren mit maximalen Spaltenpivots aus Abschnitt 9.2 verwendet werden, um lineare Gleichungssysteme mit bis zu 4 Unbekannten auf den Rechnern HP-67/97 zu lösen und mit bis zu 7 Unbekannten auf dem Rechner TI-59. Von den iterativen Verfahren zur Lösung linearer Gleichungssysteme wird in Abschnitt 9.3 das Jacobische Gesamtschrittverfahren kurz besprochen, ausführlich behandelt werden das Einzelschrittverfahren von Gauß-Seidel und das zugehörige Relaxationsverfahren. In Kapitel 10 wird das Einzelschrittverfahren für nichtlineare Gleichungssysteme am Beispiel der Differenzenapproximation nichtlinearer Randwertaufgaben erläutert. Hier ließe sich auch das Newtonsche Verfahren für nichtlineare Gleichungssysteme auf den Taschenrechnern einsetzen, worauf jedoch nicht näher eingegangen werden kann. Eigenwertaufgaben bei Matrizen werden in Kapitel 11 besprochen; mit der Potenzmethode für symmetrische Matrizen können auf den Rechnern HP-67/97 und TI-58 Matrizen mit bis zu 4 Zeilen behandelt werden, auf dem Rechner TI-59 Matrizen bis zur Zeilenzahl 9. Die explizite Differenzenapproximation für die Anfangs-Randwert-Aufgabe der Wärmeleitungsgleichung in Kapitel 12 dient als Beispiel für die numerischen Methoden bei partiellen Differentialgleichungen. Den Abschluß bilden in Kapitel 13 die Damen-Aufgabe und das d'Hondtsche Höchstzahlverfahren, zwei Beispiele nichtnumerischer Datenverarbeitung.

Die numerischen Beispiele am Ende eines jeden Abschnitts wurden alle sowohl auf einem Rechner HP-97 wie auf einem Rechner TI-59 mit Drucker gerechnet. Sie dienen zur Veranschaulichung der Algorithmen; zur Charakterisierung der Verfahren in Bezug auf ihre numerischen Eigenschaften reichen sie jedoch nicht aus.

Dieses Buch ist entstanden aus einem Vorlesungsskriptum, das ich zu meiner Vorlesung "Praktische Mathematik auf Taschenrechnern" im Wintersemester 1979/80 ausgegeben habe, und aus einer Programmsammlung, die im Wintersemester 1978/79 von Herrn Professor Dr. F. Stummel und mir gemeinsam erstellt worden war zu unseren Vorlesungen über Numerische Mathematik.

Herrn Professor Dr. F. Stummel bin ich dankbar für die Ermunterung, dieses Buch zu schreiben, für seine Unterstützung und für wertvolle Anregungen. Ebenso danke ich Herrn Professor Dr. K.H. Müller für zahlreiche Hinweise und ganz besonders Frau H. Meßner für das sorgfältige Schreiben des reproduktionsreifen Manuskripts.

Frankfurt am Main, im August 1980 Karl Hainer

INHALT

1. Einführung 1
   1.1. Algorithmen und Programme 1
   1.2. Hinweise 9
   1.3. Schneiden und Runden 12

2. Berechnung von Funktionen 22
   2.1. Polynome 22
   2.2. Unendliche Reihen 27

3. Berechnung von Nullstellen 33
   3.1. Die Methode der Intervallhalbierung 33
   3.2. Die Methode der sukzessiven Approximation 38
   3.3. Das Newtonsche Verfahren 44
   3.4. Regula falsi 50
   3.5. Quadratische Interpolation 54

4. Interpolationspolynome und Differenzenschema 61

5. Numerische Integration 74
   5.1. Trapezformeln und Simpsonsche Formel 74
   5.2. Romberg-Integration 79

6. Ausgleichsgerade 89

7. Anfangswertaufgaben gewöhnlicher Differentialgleichungen 101
   7.1. Einschrittverfahren 102
   7.2. Mehrschrittverfahren 114

8. Rechnen mit Vektoren und Matrizen 131

9. Lineare Gleichungssysteme 135
   9.1. Defektvektoren 135
   9.2. Das Gaußsche Eliminationsverfahren 144
   9.3. Iterative Verfahren 167

10 Nichtlineare Gleichungssysteme 183

11. Eigenwertaufgaben bei Matrizen:
Die Potenzmethode 200

12. Die Wärmeleitungsgleichung 216

13. Beispiele nichtnumerischer Datenverarbeitung 226
13.1. Die Damen-Aufgabe 226
13.2. Das d'Hondtsche Verfahren 234

Literatur 246

Sachverzeichnis 248

1. Einführung

Wir erklären in diesem Kapitel unsere Notation für Algorithmen und Taschenrechnerprogramme sowie die Methodik bei der Programmentwicklung. Es folgen Hinweise über den Aufbau und die Verwendung der angegebenen Programme. Schließlich zeigen wir mit Hilfe der Funktion INT Möglichkeiten zum Schneiden und Runden von Zahlen, die wir bei den Rechnern TI-58/59 zur 13-stelligen Anzeige verwenden.

1.1. Algorithmen und Programme

Wir beginnen mit einfachen Beispielen, um daran die Schreibweisen und das methodische Vorgehen zu erläutern, womit in den weiteren Kapiteln dieses Buches kompliziertere Aufgabenstellungen behandelt werden.

Zunächst beschäftigen wir uns damit, die Summe der Stammbrüche $1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{n}$ zu ermitteln. Die Rechnungen zur Auswertung der Summe

$$s_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

zerlegen wir in solche aufeinanderfolgende Einzelschritte, wie sie mit einem Taschenrechner ausgeführt werden können. Die Vorschrift, welche die einzelnen Rechenschritte sowie ihre Reihenfolge genau festlegt, nennen wir einen Algorithmus. Der folgende Algorithmus beruht darauf, daß mit den Teilsummen

$$s_k = 1 + \frac{1}{2} + \ldots + \frac{1}{k}, \quad k<n,$$

die Beziehungen gelten

$$s_0 = 0, \quad s_k = s_{k-1} + \frac{1}{k}, \quad k = 1,2,\ldots,n.$$

Damit können die Summen rekursiv berechnet werden.

Algorithmus Voraussetzung: n ist eine natürliche Zahl.
Eingabe: n
Anzeige: $s_1, s_2, s_3, \ldots, s_n$

i	n	s

$s \leftarrow 0$
$i \leftarrow 1$
$s \leftarrow s + \frac{1}{i}$
Anzeige s
$i \leftarrow i + 1$
$i \ : \ n \ \leq$
Stop

Hier ist als Voraussetzung die Eigenschaft genannt, die für die Gültigkeit des Algorithmus erfüllt sein muß. Nach "Eingabe" ist n als diejenige Größe aufgeführt, deren Wert festliegen muß, bevor Rechnungen gemäß den einzelnen Anweisungen des Algorithmus ausgeführt werden. In der Zeile "Anzeige" sind die Größen angegeben, die als Ergebnisse angezeigt werden. Den einzelnen Anweisungen des Algorithmus vorangestellt ist jeweils eine Auflistung der vorkommenden Variablen, womit man eine Übersicht der benötigten Speicherplatzreservierungen für die zugehörigen Taschenrechnerprogramme erhält. Eingabegrößen sind hierbei unterstrichen.

In den Anweisungen tritt der Pfeil ← auf. Er kennzeichnet Wertzuweisungen der allgemeinen Gestalt

$$v \leftarrow a.$$

Diese Symbolik bedeutet, daß der Wert der Variablen v ersetzt wird durch den Wert des Ausdrucks a. Hierzu wird zunächst a berechnet, wobei die zu diesem Zeitpunkt gültigen Werte von Variablen verwendet werden; nach Abschluß der Berechnung von a wird dann der berechnete Wert nach v gespeichert. Die ersten Beispiele s←0 und i←1 könnten auch durch s = 0 und i = 1 richtig beschrieben werden. Da es sich bei einer Wertzuweisung um einen zeitlichen Prozeß handelt,

ist jedoch im allgemeinen die Verwendung des Gleicheitszeichens unzutreffend. Dies zeigen die beiden nächsten Beispiele. Durch die Anweisung s ← s+$\frac{1}{i}$ wird der Wert der Variablen s verändert; der bisherige Wert von s dient zur Berechnung des rechts stehenden Ausdrucks, und der neue Wert wird anschließend wieder unter dem Namen s gespeichert. Entsprechend wird durch die Anweisung i ← i+1 der Wert von i um 1 erhöht.

Die sechste Zeile des Algorithmus enthält eine Entscheidung über die weitere Abfolge der Anweisungen auf Grund eines Vergleichs. Hier sollen die jeweils gültigen Werte von i und n miteinander verglichen werden. Ist die Bedingung $i \leq n$ erfüllt, so folgt die Ausführung des Algorithmus dem Pfeil; dadurch entsteht hier ein Rücksprung. Ist die angegebene Bedingung verletzt, so wird der Verzweigungspfeil nicht beachtet und es wird zu der Anweisung in der nächsten Zeile übergegangen. Als Bedingungen bei entsprechenden Vergleichen sind =, $\neq$, $\geq$, >, $\leq$, < möglich.

In dem angegebenen Algorithmus durchläuft i die Werte i = 1,2,...,n; für jeden dieser Werte werden dieselben Anweisungen ausgeführt. Eine solche Situation bezeichnen wir als Schleife, und wir schreiben zur Abkürzung anstelle des obigen Algorithmus auch

```
s ← 0
i = 1(1)n
|  s ← s + 1/I
|  Anzeige s
Stop
```

In der Titelzeile i = a(w)e einer Schleife ist der Schleifenparameter i angegeben sowie sein Anfangswert a, die Schrittweite w und der Endwert e. Für i = a, a + w, a + 2w,... werden die Anweisungen des Wiederholungsbereichs der Schleife ausgeführt; das sind die Anweisungen, die durch einen linken seitlichen Winkel gekennzeichnet sind. Jeweils nach

der letzten Anweisung des Wiederholungsbereichs findet die Wertzuweisung i ← i+w statt, und die Ausführung des Algorithmus verzweigt zur ersten Anweisung des Wiederholungsbereichs der Schleife. Diese Wiederholungen werden vorgenommen, solange der so erhaltene Wert des Schleifenparameters i noch zwischen a und e liegt.

Nun geben wir die Übertragung dieses Algorithmus in Programme für die Rechner HP-67/97 und TI-58/59 an.

Programm für HP-67/97

Datenspeicher

R_1	R_2		R_I
n	s		i

```
001   STO 1 0 STO 2 1 STO I          R1←n, s←0, i←1
006   LBL 0 I 1/x STO + 2            s ← s+1/i   ←
010   RCL 2 PRTx ISZ I               Anzeige s, i ← i+1
013   RCL 1 I x≤y? GTO 0             wenn i≤n  ——
017   R/S                            Stop
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}
i	n	s

```
000   STO 01 0 STO 02 1 STO 0        R01←n, s←0, i←1
008   Lbl INV RCL 0 1/x SUM 2        s ← s+1/i   ←
015   RCL 2 Prt Op 20                Anzeige s, i ← i+1
020   RCL 0 x↔t RCL 1 x≥t INV        wenn n≥i  ——
027   R/S                            Stop
```

Die Wiedergabe der Taschenrechnerprogramme ist jeweils in drei Teile gegliedert. In der Mitte sind Taschenrechneranweisungen in Zeilen zusammengefaßt, deren Wirkung rechts durch die entsprechenden Teile des vorhergehenden Algorithmus erläutert werden. Die dreistellige Zahl in der linken Spalte nennt die Nummer derjenigen Programmspeicherzeile, die die

erste Taschenrechneranweisung dieser Zeile aufnimmt. Zum leichteren Auffinden sind die Sprungmarken (Labels) unterstrichen.

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der natürlichen Zahl n in das Anzeigeregister.
3. Zum Start der Rechnung und Drucken der Ergebnisse $s_1, s_2, \ldots, s_n$ betätigt man
   bei den Rechnern HP-67/97 die Tasten RTN R/S,
   bei den Rechnern TI-58/59 die Tasten RST R/S.

Mögliche Wiederholungen: ab Schritt 2.

Die Eingabe des Programms in den Programmspeicher kann im Programmiermodus über das Tastenfeld erfolgen; falls bei den Rechnern HP-67/97 oder TI-59 das Programm bereits auf eine Magnetkarte aufgezeichnet ist, so kann es auch durch Einlesen der Magnetkarte gemäß der jeweiligen Bedienungsanleitung eingegeben werden.

Die Druckanweisung PRTx gilt für den Rechner HP-97; die analoge Anweisung Prt gilt bei den Rechnern TI-58/59, falls sie an einen Drucker angeschlossen sind. Für den Rechner HP-67 und im Fall der TI-Rechner ohne angeschlossenen Drucker ersetzt man die Druckanweisungen entweder durch eine Pausenanweisung zur kurzzeitigen Anzeige der entsprechenden Werte oder durch die Anweisung R/S, so daß die Programmausführung unterbrochen wird zur Anzeige des jeweiligen Ergebnisses. In diesem Fall muß dann zur Fortsetzung der Rechnung die Taste R/S betätigt werden.

Mit dem Rechner HP-97 ergab sich für n = 10 unter Verwendung des Anzeigeformats FIX DSP 9 das folgende Ergebnis.

```
1.000000000  ***
1.500000000  ***
1.833333333  ***
2.083333333  ***
2.283333333  ***
2.450000000  ***
2.592857143  ***
2.717857143  ***
2.828968254  ***
2.928968254  ***
```

Als weiteres Beispiel behandeln wir einen Algorithmus und zugehörige Programme, um von gegebenen Zahlen $a_1, a_2, \ldots, a_n$ die größte zu bestimmen.

Algorithmus Voraussetzung: $n \geq 2$
Eingabe: $n, a_1, a_2, \ldots, a_n$
Anzeige: $m = \max(a_1, a_2, \ldots, a_n)$

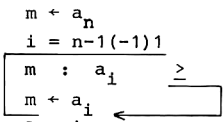
a_1	a_2	...	a_n	i	m	n

$m \leftarrow a_1$
$i = 2(1)n$
  $m : a_i \quad \geq$
  $m \leftarrow a_i$
Anzeige m
Stop

Während der Ausführung des Algorithmus ist m die größte der bereits untersuchten Zahlen. Beginnend mit $m = a_1$ wird m mit den Zahlen $a_2, \ldots, a_n$ verglichen; tritt dabei ein Wert auf, der größer ist als der augenblickliche Wert von m, so wird m durch diesen größeren Wert ersetzt.

In den nachfolgenden Taschenrechnerprogrammen wird davon ausgegangen, daß die Zahl a_i auf dem Datenspeicherplatz R_i gespeichert ist für $i = 1, \ldots, n$. Die Zahlen a_i werden durch indirekte Adressierung von ihren Speicherplätzen abgerufen. Um zur Kontrolle der Programmschleife die Dsz-Anweisung ver-

wenden zu können, soll die Schleife rückwärts durchlaufen werden. Zur Programmierung wird also der obige Algorithmus in die folgende gleichwertige Form umgewandelt.



Programm für HP-67/97

Datenspeicher

R_1	R_2	...	R_n		R_I
a_1	a_2	...	a_n		n
					i

$n \leq 24$

```
001   RCL (i) ENTER DSZ I          m←a_n, i ← n-1
004   LBL O CLx RCL (i) x>y?       wenn a_i>m,
008   x↔y                          dann m←a_i
009   DSZ I GTO O                  i ← i-1, wenn i>O
011   R↓ R/S                       Anzeige m, Stop
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	...	R_n
n	a_1	a_2	...	a_n
i				

```
000   RCL Ind O x↔t Op 30          m←a_n, i ← n-1
005   Lbl INV RCL Ind O            R_X←a_i
009   INV x≥t lnx                  wenn a_i<m
012   x↔t                          m←a_i
013   Lbl lnx DSZ O INV            i ← i-1, wenn i>O
018   x↔t R/S                      Anzeige m, Stop
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der Zahlen n, a_1, a_2,...,a_n
   bei den Rechnern HP-67/97: n STO I a_1 STO 1
   a_2 STO 2 ... a_n STO n,
   bei den Rechnern TI-58/59: n STO 00 a_1 STO 01 ... a_n STO n.
3. Start des Programms durch RTN R/S bzw. RST R/S.
   Die Ausführung des Programms endet damit, daß m in der Anzeige wiedergegeben wird.

Mögliche Wiederholungen: ab Schritt 2.

Im Unterschied zu dem vorher aufgestellten Algorithmus zur Ermittlung von max(a_1, a_2,...,a_n) ist in den Programmen für die Taschenrechner kein Datenspeicherplatz für m reserviert. In dem Programm für die Rechner HP-67/97 wird m in R_Y geführt, der zweiten Ebene des Stapelspeichers; in dem Programm für die Rechner TI-58/59 dient das Testregister R_T zur Aufnahme von m.

Der Vergleich der beiden Taschenrechnerprogramme zeigt, daß die Verzweigungen unterschiedlich angelegt sind. Grund dafür ist die verschiedenartige Verzweigungslogik der beiden Fabrikate. Beiden gemeinsam ist, daß die Programmspeicherzeile nach der Abfrage übersprungen wird, falls die gefragte Bedingung verletzt ist. Ist bei den HP-Rechnern die Bedingung x>y erfüllt, so wird die nächste Anweisung ausgeführt. Da für x>y hier nur die Vertauschung x↔y vorgenommen werden soll, steht die zugehörige Anweisung unmittelbar nach der Abfrage; man kommt also ohne einen Sprungbefehl aus. Es wäre jedoch auch möglich, nach der Abfrage einen Sprungbefehl einzufügen, ähnlich wie im ersten Beispiel dieses Abschnitts. Im Gegensatz dazu verzweigt bei den TI-Technern das Programm bei Bestehen der Bedingung x<t zu der Sprungadresse, die in der folgenden Anweisung genannt ist; hier ist also grundsätzlich nur ein Sprung möglich.

1.2. Hinweise

Wir geben nun Bemerkungen und Hinweise zum Aufstellen von Algorithmen und zugehörigen Programmen für Taschenrechner sowie zu ihrer Verwendung.

ALGORITHMEN

Den Algorithmen vorangestellt haben wir jeweils die Ein- und Ausgabegrößen sowie die Liste der auftretenden Variablen, häufig auch Voraussetzungen zur Gültigkeit des Algorithmus. Es hat sich als günstig erwiesen, bei dem Aufstellen eines neuen Algorithmus diese Größen während der Formulierung des Flußdiagramms zu notieren.

DATENSPEICHER

Wenn der Algorithmus vollständig vorliegt, kann man mit der Übertragung in ein Taschenrechnerprogramm beginnen. In diesem Stadium sollte die Belegung des Datenspeichers festgelegt werden. Die Liste der Variablen des Algorithmus dient jetzt als Übersicht über die benötigten Datenspeicherplätze. Eine zweckmäßige Belegung des Datenspeichers sowie auch schon vorher eine geeignete Benennung der Variablen trägt wesentlich zur Übersichtlichkeit und Durchschaubarkeit der Programme bei. Gleichartige Größen wie beispielsweise Schleifenindizes oder Vektorkomponenten können durch gleichartige Namen gekennzeichnet werden, und man verwendet zu ihrer Speicherung nach Möglichkeit benachbarte Datenspeicherplätze.

ANZEIGEANWEISUNGEN

In den Taschenrechnerprogrammen lassen wir Ergebnisse jeweils durch Druckanweisungen anzeigen. Verwendet man ein Gerät ohne Drucker, so kann man zur kurzfristigen Anzeige von Zwischenergebnissen die zugehörigen Druckanweisungen durch "Pause"-Anweisungen ersetzen. Möchte man Ergebnisse notieren, so verwendet man statt dessen die Stop-Anweisung R/S. Zur Fortsetzung der Rechnung muß dann das Programm mit der Taste

R/S jeweils erneut gestartet werden. Druckanweisungen in HP-97-Programmen wirken auf dem Rechner HP-67 als 5-Sekunden-Pause und umgekehrt.

PROGRAMMAUFLISTUNGEN

Die Auflistungen der Programme enthalten jeweils nur die erforderlichen Tastenfunktionen, auch wenn sie bei der Wiedergabe des Programms mit Hilfe eines Druckers ausführlicher wiedergegeben werden. So dient zum Beispiel bei den HP-Rechnern die Tastenfunktion B als Kurzform der Anweisung GSB B; bei den TI-Rechnern ist es beispielsweise möglich, für RCL OO die Kurzform RCL O einzugeben, falls anschließend keine Zifferntaste betätigt wird.

PROGRAMMABBRUCH

Insbesondere in den ersten Kapiteln sind Algorithmen in Endlosform aufgestellt, etwa wenn eine Folge von Näherungen für einen Grenzwert angezeigt wird. Die Ausführung der zugehörigen Programme wird dann im Rechenmodus mit der Taste R/S gestoppt. Bei den Rechnern TI-58/59 ist es möglich, daß danach noch arithmetische Operationen offenstehen. Vor Beginn einer neuen Rechnung müssen diese erst durch "=" abgeschlossen oder durch CLR gelöscht werden.

Bei dem Versuch, durch ein Programm unzulässige Operationen ausführen zu lassen, entstehen Fehlermeldungen, zum Beispiel bei Division durch Null oder wenn die Quadratwurzel einer negativen Zahl berechnet werden soll. Bei den Rechnern HP-67/97 führt eine Fehlermeldung zur sofortigen Unterbrechung der Programmausführung und zu der Anzeige ERROR. Im Fall der Rechner TI-58/59 wird bei einer Fehlerbedingung die Rechnung nur unterbrochen, wenn zuvor Flag 8 gesetzt war; andernfalls wird die Rechnung weitergeführt, und Ergebnisse werden mit Fehlermeldung angegeben, indem die Werte blinkend angezeigt werden und bei gedruckten Zahlen zusätzlich Fragezeichen angefügt sind.

MARKEN

In den Programmauflistungen stehen Sprungmarken (Labels) jeweils am Anfang einer Zeile und sind unterstrichen, so daß sie beim Lesen leicht gefunden werden können. Auch ist dadurch bei der Erstellung von zusätzlichen Programmteilen sichtbar, welche Marken bereits verwendet sind. In der Regel verwenden wir die Marken LBL 0, LBL 1, LBL 2,... in Programmen für die HP-Rechner an den entsprechenden Stellen wie die Marken Lbl INV, Lbl lnx, Lbl CE,... in dem zugehörigen Programm für die TI-Rechner. Bei den TI-Rechnern ist es möglich, absolut zu adressieren, indem für Sprungbefehle und Unterprogrammaufrufe anstelle von Marken die Nummern der betreffenden Programmspeicherzeilen verwendet werden. Ähnlich kann absolute Adressierung bei den HP-Rechnern mit Hilfe des Indexregisters R_I erreicht werden. Obwohl die absolute Adressierung weniger Rechenzeit beansprucht, haben wir auf diese Möglichkeit verzichtet, um bei der Programmerstellung die Flexibilität zu erhöhen. Bei der ausschließlichen Verwendung von Marken lassen sich Programmteile einfacher einfügen oder entfernen, und man erhält eine bessere Übersicht über die Programmstruktur. Absolute Adressierung ist nur zweckmäßig bei ausgetesteten Programmen, die nicht mehr modifiziert werden sollen.

UNTERPROGRAMME

Häufig sind angegebene Programme noch durch ein weiteres Programmstück zu ergänzen, bevor sie verwendet werden können; wenn etwa Nullstellen einer Funktion bestimmt werden sollen, so muß ein Unterprogramm eingegeben werden, das die Werte der speziellen Funktion berechnet. Zum einfacheren Programmieren solcher Programmstücke endet jeweils das hier angegebene Taschenrechnerprogramm mit derjenigen Marke, mit der das noch fehlende Unterprogramm beginnt. Zur Eingabe des zusätzlichen Programmteils wird der Programmschrittzeiger durch "GTO Marke" im Rechenmodus so eingestellt, daß man nach Umschalten in den Programmiermodus dann die Anweisungen unmittelbar eingeben

kann. In der Anleitung zur Verwendung der Programme ist angegeben, auf welchen Datenspeicherplätzen die Größen zur Verfügung stehen, die für das Unterprogramm benötigt werden; ebenso wird genannt, auf welchen Speicherplätzen die berechneten Werte nach dem Rücksprung in das rufende Programm erwartet werden.

Ruft man in einem Programm für die HP-Rechner ein Unterprogramm auf, so muß man darauf achten, daß über das Stapelregister verfügt werden kann. Sollen jedoch für das aufrufende Programm im Stapelspeicher Daten aufbewahrt werden, so ist zu prüfen, bis zu welcher Ebene das Stapelregister in dem Unterprogramm benötigt wird; nur so weiß man sicher, welche Größen im Stapelregister auch nach der Beendigung des Unterprogramms noch verfügbar sind.

Bei den TI-Rechnern kann die =-Taste in Unterprogrammen nur verwendet werden, wenn nicht gleichzeitig im aufrufenden Programm noch arithmetische Operationen offenstehen; denn mit dem Gleichheitszeichen werden alle offenstehenden arithmetischen Operationen abgeschlossen.

1.3. Schneiden und Runden

Die Ergebnisanzeige elektronischer Taschenrechner kann so eingestellt werden, daß Resultate mit einer bestimmten Anzahl von Nachkommastellen angezeigt werden; die angezeigte Zahl entsteht durch Rundung des wirklichen Wertes auf die gewünschte Stellenzahl. Intern ist weiterhin der genaue Wert vorhanden, und er wird auch bei der weiteren Rechnung mit der vollen Genauigkeit verwendet.

In diesem Zusammenhang besprechen wir zunächst kurz die Gaußsche Klammer [x] und die Funktion INT(x). Anschließend beschäftigen wir uns damit, wie Dezimalzahlen in ihrem Wert auf eine gewünschte geringere Anzahl von Nachkommastellen abgeschnitten werden können. Mit dieser Überlegung wird es möglich, Zahlen nicht nur für die Anzeige, sondern auch in

ihrem internen Wert zu runden; außerdem werden wir für die Rechner TI-58/59 die intern verwendeten 13-stelligen Zahlen anzeigen können.

[X] UND INT(X)

An verschiedenen Stellen benutzen wir die Gaußsche Klammer [x]. Ist x eine gegebene Zahl, so ist [x] die größte ganze Zahl $\leq x$. Daher ist eine Zahl x ganz, genau wenn $x = [x]$. Diese Eigenschaft verwenden wir zum Beispiel, wenn wir in Flußdiagrammen ermitteln, ob ein Schleifenindex i Vielfaches einer festen Zahl n ist; wir vergleichen dazu die beiden Zahlen $\frac{i}{n}$ und $[\frac{i}{n}]$.

Auf Taschenrechnern steht jedoch die Gaußsche Klammer nicht zur Verfügung, sondern die Funktion INT(x), die mit [x] eng verwandt ist. INT(x) ist der ganzzahlige Teil von x; bei Dezimalzahlen werden also die Nachkommastellen abgeschnitten. Für positive Zahlen x und für ganze Zahlen x stimmen [x] und INT(x) überein, im Fall von negativen Zahlen, die nicht ganz sind, unterscheiden sie sich jedoch. Es besteht der Zusammenhang

$$\mathrm{INT}(x) = \begin{cases} [x] & x>0, \\ [x] \quad \text{für} & x \text{ ganz}, \\ [x]+1 & x<0 \text{ und } x \text{ nicht ganz}. \end{cases}$$

ABSCHNEIDEN NACH D NACHKOMMASTELLEN

Eine gegebene Dezimalzahl mit mehr als $d \geq 0$ Nachkommastellen soll nach der d-ten Stelle hinter dem Komma abgeschnitten werden.

Multiplikation mit 10^d bringt alle diejenigen Ziffern, die erhalten bleiben sollen, vor das Komma; mit der Funktion INT werden die weiteren Dezimalstellen abgeschnitten; Division durch 10^d setzt das Dezimalkomma wieder an seinen ursprünglichen Platz. Die gegebene Zahl x wird also ersetzt durch $y = \mathrm{INT}(10^d x)10^{-d}$.

Algorithmus Eingabe: d; x
Anzeige: $y = INT(10^d x) \cdot 10^{-d}$

d	x

$x \leftarrow 10^d x$
$x \leftarrow INT(x)$
$x \leftarrow x \cdot 10^{-d}$
Anzeige x
Stop

Programm für HP-67/97

Datenspeicher

R_0
10^d

001	RCL 0 * INT RCL 0 ÷	$x \leftarrow INT(10^d x) \cdot 10^{-d}$
008	R/S	Stop

Programm für TI-58/59

Datenspeicher

R_{00}
10^d

000	* RCL 0 = Int ÷ RCL 0 =	$x \leftarrow INT(10^d x) \cdot 10^{-d}$
009	R/S	Stop

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der Stellenzahl d
   bei HP-67/97: EEX d STO 0,
   bei TI-58/59: 1 EE d STO 00.
3. Die Zahl x soll auf d Nachkommastellen abgeschnitten werden: x RTN R/S bzw. x RST R/S.

Mögliche Wiederholungen:
ab Schritt 3 für eine andere Zahl x,
ab Schritt 2 für eine andere Stellenzahl d.

Bei den Rechnern TI-58/59 kann im Anschluß an Schritt 2 die Exponentialschreibweise der Anzeige wieder aufgehoben werden durch INV EE.

RUNDEN AUF D NACHKOMMASTELLEN

Es sonn in der üblichen Weise auf d Stellen gerundet werden: Ist bei der gegebenen Zahl die (d+1)-te Dezimalstelle ≥ 5 so soll bei dem Rundungsvorgang die d-te Dezimalstelle um eine Einheit erhöht werden.

Durch Multiplikation der gegebenen Zahl mit 10^d wird wieder erreicht, daß alle gewünschten Ziffern vor dem Komma stehen. Addition von $\frac{1}{2}$ und anschließende Anwendung der Funktion INT ergibt die richtigen Ziffern für die Rundung; Division durch 10^d schiebt das Komma wieder an seinen ursprünglichen Platz zurück. Runden auf d Nachkommastellen kann also erreicht werden, indem die Zahl x ersetzt wird durch $z = \mathrm{INT}(10^d x + .5) \cdot 10^{-d}$. Der zugehörige Rechenablauf stimmt mit dem vorherigen überein bis auf die erste Anweisung, die dort $x \leftarrow 10^d x$ lautete, hier nun $x \leftarrow 10^d x + .5$ heißen muß.

Mit derselben Speicherbelegung und der entsprechenden Anleitung zur Verwendung lauten jetzt die Programme

für HP-67/97

001 RCL 0 * .5 + INT RCL 0 ÷ R/S,

für TI-58/59

000 * RCL 0 + .5 = Int ÷ RCL 0 = R/S.

Die Rundung auf d Nachkommastellen ist auch ohne diese Programme möglich durch Verwendung eingebauter Funktionen der Rechner.

Ist bei den Rechnern HP-67/97 die Anzeige durch DSP d auf $d \leq 8$ Nachkommastellen eingestellt, so wird durch die Funktionstaste RND eine in der Anzeige befindliche und gerundet angezeigte Zahl durch ihren gerundeten Wert ersetzt. Dies kann auch innerhalb einer Programmausführung geschehen, indem d im Indexregister R_I gespeichert wird; die Anweisungsfolge DSP (i) RND bewirkt dann die Rundung der Zahl im Anzeigeregister auf d Nachkommastellen.

Ähnlich kann bei den Rechnern TI-58/59 vorgegangen werden: eine in der Anzeige befindliche und gerundet angezeigte Zahl wird durch ihren gerundeten Wert ersetzt, falls die Taste EE betätigt wird.

REDUKTION AUF DIE ERSTEN D WESENTLICHEN ZIFFERN

Wir wollen bei einer gegebenen Zahl die Ziffern nach der d-ten wesentlichen Ziffer durch Nullen ersetzen.

Dazu gehen wir aus von der normalisierten Dezimaldarstellung der gegebenen Zahl a, mit der Mantisse m und dem Exponenten k,

$$a = m \cdot 10^k, \quad 1 \leq |m| < 10, \quad k \text{ ganz}.$$

Die Zahl $b = a \cdot 10^{-k-1}$ besitzt dieselben Ziffern wie a, wobei nun bei b die wesentlichen Ziffern direkt hinter dem Komma beginnen. Jetzt können die Überlegungen über das Abschneiden nach d Nachkommastellen angewandt werden, und anschließend führt Multiplikation mit 10^{k+1} das Dezimalkomma wieder an die Ausgangsstelle zurück. Daher entsteht aus der gegebenen Zahl a die reduzierte Zahl a', die mit a nur die ersten d wesentlichen Ziffern gemeinsam hat und anschließend nur noch Nullen enthält, indem betrachtet wird

$$a' = \mathrm{INT}(a \cdot 10^{d-k-1}) \cdot 10^{-d+k+1}.$$

Bei gegebener Zahl a berechnet sich der Exponent k ihrer normalisierten Dezimaldarstellung gemäß

$$k = [\lg|a|] = \begin{cases} \text{INT}(\lg|a|) & \lg|a| \geq 0, \\ \text{INT}(\lg|a|) & \text{für } \lg|a| < 0 \text{ und } \lg|a| \text{ ganz}, \\ \text{INT}(\lg|a|)-1 & \text{sonst.} \end{cases}$$

In dem folgenden Algorithmus ist s = k + 1.

Algorithmus Eingabe: d; a
Anzeige: a'

a	d	r	s

s ← lg|a|
s : [s] =
< s : 0
s ← s + 1
s ← INT(s)
r ← 10^{s-d}
a ← INT($\frac{a}{r}$)·r
Anzeige a

Programm für HP-67/97

Datenspeicher

R_0
d

```
001   LBL A ENTER ABS LOG
005   FRAC x=0? GTO 0
008   CLx LASTx x<0? GTO 1
012   LBL 0 CLx LASTx 1 +
017   LBL 1 INT RCL 0 - 10^x
022   ÷ LASTx x↔y INT *
027   RTN
```

```
s←lg|a|
wenn s = [s]
wenn s<0
s ← s + 1
s←INT(s), r←10^(s-d)
R_X ← INT(a/r)·r
Rücksprung
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}
d	a	r	(s)

```
000   Lbl A STO 1                    R01←a
004   |x| log STO 3                  s←lg|a|, R03←s
008   Int x↔t RCL 3 x=t INV          wenn s=[s]
014   CP INV x≥t lnx                 wenn s<0
018   Lbl INV ( CE + 1 )             s ← s + 1
025   Lbl lnx Int                    s←INT(s)
028   ( CE - RCL 0 ) INV lg STO 2    r←10^(s-d)
038   ( ( RCL 1 ÷ RCL 2 )            RX ← a/r
046   Int * RCL 2 )                  RX←INT(a/r)·r
051   INV SBR                        Rücksprung
```

Damit diese Programmeinheit als Unterprogramm verwendet werden kann, wurde es vermieden, die Taste "=" zu verwenden. Auf diese Weise ist die Verwendung als Unterprogramm auch möglich, wenn in dem rufenden Programm zum Zeitpunkt des Aufrufs unvollständige Operationen offenstehen. Bei den verwendeten Klammeroperationen tritt auch die Blindoperation mit Klammern "(CE" auf; hierdurch hat man direkt hinter der Klammer denjenigen Wert zur Verfügung, der sich vor Öffnen der Klammer im Anzeigeregister befand.

In der folgenden Programmversion wird mit dem Testregister R_T als Hilfsspeicher und der arithmetischen Hierarchie in der Weise gearbeitet, daß für a, r und s keine Datenspeicherplätze reserviert werden müssen.

Datenspeicher

R_{00}
d

```
000   Lbl A ( ( CE ÷ ( ( CE |x| log   s←lg|a|
011   ( CE + x↔t 0 )                  R_X←s, R_T←s
017   Int x↔t x=t INV                 wenn s=[s]
021   CP INV x≥t lnx                  wenn s<0
025   Lbl INV ( CE + 1 )              s ← s + 1
032   Lbl lnx Int                     s←INT(s)
035   - RCL 0 ) INV log               r←10^(s-d)
041   + x↔t 0 )                       R_X←r, R_T←r
045   ) Int * x↔t )                   R_X←INT(a/r)·r
050   INV SBR                         Rücksprung
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der Stellenzahl d
   bei HP-67/97: d STO 0,
   bei TI-58/59: d STO 00.
3. Eingabe der Zahl a, Berechnung und Anzeige der reduzierten Zahl a': a A.

Mögliche Wiederholungen:
Schritt 3 für eine andere Zahl a;
ab Schritt 2 für eine andere Stellenzahl d.

13-STELLIGE ANZEIGE

Das Programm A verwenden wir nun, um bei den Rechnern TI-58/59 die interne Genauigkeit der Zahlen sichtbar zu machen: über die 10 Stellen, die üblicherweise angezeigt werden können, sind noch zusätzlich 3 Schutzstellen vorhanden. Eine entsprechende Untersuchung für die Rechner HP-67/97 entfällt, da bei diesen Geräten die intern verwendeten Zahlen bereits mit ihrer vollen 10-stelligen Genauigkeit angezeigt werden, wenn man das Anzeigeformat durch DSP 9 festlegt.

Wir gehen folgendermaßen vor. Mit d = 5 wird eine im Anzeigeregister befindliche Zahl a zunächst auf 5 wesentliche Ziffern reduziert zu der Zahl a' und ausgedruckt; damit sind die fünf ersten wesentlichen Ziffern von a sichtbar. Anschließend wird die Differenz a - a' erneut reduziert und ausgedruckt; hierdurch werden die sechste bis zehnte wesentliche Ziffer der Ausgangszahl a angezeigt. Schließlich wird noch die Differenz aus a - a' und der zweiten Reduktion gedruckt, was die restlichen drei wesentlichen Ziffern der ursprünglichen Zahl a liefert.

Wir verwenden die zweite Version des vorangehenden Programms A mit d = 5, so daß kein Datenspeicherplatz reserviert wird.

Programm für TI-58/59

Unterprogramm: Reduktion auf die ersten fünf wesentlichen Ziffern

```
000   Lbl A ( ( CE ÷ ( ( CE |x| log    s←lg|a|
011   ( CE + x↔t 0 )                   R_X←s, R_T←s
017   Int x↔t x=t INV                  wenn s = [s]  ┐
021   CP INV x>t lnx                   wenn s<0      │
025   Lbl INV ( CE + 1 )               s ← s + 1  ←──┘
032   Lbl lnx Int                      s←INT(s)
035   - 5 ) INV log                    r←10^(s-5)
040   + x↔t 0 )                        R_X←r, R_T←r
044   ) Int * x↔t )                    R_X←INT(a/r)·r
049   INV SBR                          Rücksprung
```

Hauptprogramm

```
050   Lbl B ( CE - A Prt               Anzeige a'
057   ) ( CE - A Prt                   Anzeige (a-a')'
063   ) Prt                            Anzeige a-a'-(a-a')'
065   INV SBR                          Rücksprung
```

Anleitung zur Verwendung des Programms

1. Eingabe des Programms
2. EE Fix 4
3. Drucken einer Zahl a mit ihren 13 wesentlichen Ziffern:
   a B.

Mögliche Wiederholungen: Schritt 3.

Beispiele

π	$\sin 45^0$
3.1415 00	7.0710-01
9.2653-05	6.7811-06
5.9400-10	8.8200-11

e	$\cos 45^0$
2.7182 00	7.0710-01
8.1828-05	6.7811-06
4.6200-10	8.7500-11

$\sqrt{2}$	$\frac{1}{3}$
1.4142 00	3.3333-01
1.3562-05	3.3333-06
3.7500-10	3.3700-11

$\sqrt{3}$	$\frac{1}{9}$
1.7320 00	1.1111-01
5.0807-05	1.1111-06
5.7000-10	1.1300-11

2. Berechnung von Funktionen

In den praktischen Anwendungen der Mathematik tritt im Zusammenhang mit größeren Problemstellungen häufig die Aufgabe auf, Funktionswerte spezieller Funktionen zu ermitteln. Für solche Funktionen sind oft Reihenentwicklungen oder approximierende Polynome bekannt. Wir behandeln daher in diesem Kapitel die Berechnung von Polynomen mit Hilfe des Horner-Schemas sowie die näherungsweise Berechnung unendlicher Reihen durch die Folge der zugehörigen Partialsummen.

2.1. Polynome

Zur Berechnung von Funktionswerten eines Polynoms

$$p_n(x) = a_0 + a_1 x + \ldots + a_n x^n$$

verwendet man diese Darstellung nicht als Rechenvorschrift; denn mit dem anschließend geschilderten Horner-Schema liegt ein Algorithmus vor, der hinsichtlich der Anzahl der Rechenoperationen und der Rundungsfehler wesentlich günstiger ist. Über die Berechnung von Funktionswerten hinaus erläutern wir, wie mit Hilfe des Horner-Schemas Linearfaktoren von einem Polynom abgespalten werden und wie Funktionswerte der Ableitung des Polynoms berechnet werden können.

Wir gehen aus von der Darstellung

$$p_n(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \ldots + x_0(a_{n-2}+x_0(a_{n-1}+x_0 a_n))\ldots)).$$

Die Klammern werden berechnet gemäß der Vorschrift des Horner-Schemas,

$$a_n' = a_n \,,$$

$$a_k' = a_k + x_0 a_{k+1}', \quad k = n-1, n-2, \ldots, 1, 0.$$

Der gesuchte Funktionswert entsteht dann in der Form $p_n(x_0) = a_0'$.

Mit den Koeffizienten a'_k des Horner-Schemas kann ein Polynom (n - 1)-ten Grades erklärt werden durch

$$p_{n-1}(x) = a'_1 + a'_2 x + \ldots + a'_n x^{n-1} \quad .$$

Dieses Polynom erfüllt die Beziehung $p_n(x)=p_n(x_0)+(x-x_0)p_{n-1}(x)$ (s.[9] 1.1.), so daß also p_{n-1} durch Abspaltung des Linearfaktors $x - x_0$ von p_n entsteht. Folglich gilt $p'_n(x_0) = p_{n-1}(x_0)$. Damit kann man den Funktionswert der Ableitung des Ausgangspolynoms p_n jetzt dadurch ausrechnen, daß man das Horner-Schema erneut anwendet bei der Auswertung des Polynoms p_{n-1}. Wir werden diese Eigenschaft benutzen bei der Bestimmung von Nullstellen von Polynomen mit Hilfe des Newtonschen Verfahrens.

Der folgende Algorithmus mit den zugehörigen Programmen dient zur Berechnung des Funktionswertes $p_n(x_0) = a'_0$ nach Horner.

Algorithmus Voraussetzung: $n \geq 1$

Eingabe: $n, a_0, a_1, \ldots, a_n ; x_0$

Anzeige: $p_n(x_0)$

a_0	a_1	...	a_n	k	n	p	x_0
—	—	—	—		—		—

$p \leftarrow a_n$

$k = n-1(-1)0$

$p \leftarrow a_k + x_0 p$

Anzeige p

Stop

Die eingegebenen Größen bleiben während der Rechnung unverändert. Daher können weitere Funktionswerte von p_n erhalten werden durch

Eingabe: x_i, Anzeige: $p_n(x_i)$, $i = 1,2,\ldots$.

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	...	R_{n+1}		R_I
n	a_0	a_1	...	a_n		Adr a_k =k+1

$n \leq 23$

Einleseprogramm

```
001   STO 0 1 STO I                 R0←n, k←0
004   LBL 0 R/S                     Stop
006   STO (i) ISZ I GTO 0           Rk+1←ak, k ← k + 1,
```

Hauptprogramm

```
009   LBL A ENTER ENTER ENTER       RY←x0, RZ←x0, RT←x0
013   RCL 0 STO I ISZ I CLx         k←n
017   RCL (i) DSZ I                 p←an, k ← n - 1
019   LBL 1 * RCL (i) +             p←ak + x0p
023   DSZ I GTO 1                   k ← k - 1,wenn k+1>0
025   PRTx R/S                      Anzeige p, Stop
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	...	R_{n+4}
Adr a_k =k+4	n	p	x_0	a_0	a_1	...	a_n

Einleseprogramm

```
000   STO 01 4 STO 0                R01←n, k←0
005   Lbl INV R/S                   Stop
008   STO Ind 0 Op 20 GTO INV       Rk+4←ak, k ← k + 1,
```

Hauptprogramm

```
014   Lbl A STO 3                   R03←x0
018   RCL 1 + 4 = STO 0             k←n
025   RCL Ind 0 STO 2               p←an
029   Lbl lnx Op 30                 k ← k - 1
033   RCL 3 Prd 2 RCL Ind 0 SUM 2   p ← ak + x0p
041   RCL 0 x↔t 4 INV x≥t lnx       wenn 4<k + 4
048   RCL 2 Prt R/S                 Anzeige p, Stop
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der Koeffizienten des Polynoms p_n:
   n, RTN bzw. RST, R/S a_0 R/S a_1 R/S ... R/S a_n R/S.
3. Eingabe der Stelle x_0, Start der Berechnung sowie Anzeige von $p_n(x_0)$: x_0 A.

Mögliche Wiederholungen:

ab Schritt 3 für einen anderen Wert x_0,
ab Schritt 2 für ein anderes Polynom.

In dem Programm für die Rechner HP-67/97 ist es infolge der Stapelregisterlogik möglich, auf die Reservierung von eigenen Datenspeicherplätzen für x_0 und p zu verzichten. Die eigentliche Berechnung des Funktionswertes $p_n(x_0)$ kann auch auf einem nicht programmierbaren UPN-Rechner ausgeführt werden. Es handelt sich dann um die Tastenfolge

x_0 ↑↑↑ a_n * a_{n-1} + * a_{n-2} + ... * a_0 + .

Nach dem Anfang mit x_0 ↑↑↑ a_n besteht also das Kernstück des Hauptprogramms aus der Wiederholung der Tastenfolge * a_k + für k = n - 1, n - 2,..., 1, 0. Das Stapelregister ist dabei wie folgt belegt.

T				x_0	x_0	x_0	x_0	x_0		x_0	x_0	x_0	
Z			x_0	x_0	x_0	x_0	x_0	x_0		x_0	x_0	x_0	
Y		x_0	x_0	x_0	x_0	x_0	x_0a_n'	x_0		x_0	x_0a_1'	x_0	
X	x_0	x_0	x_0	x_0	a_n'	x_0a_n'	a_{n-1}	a_{n-1}'		x_0a_1'	a_0	$a_0'=p_n(x_0)$	
Tasten													
	x_0	↑	↑	↑	a_n	*	a_{n-1}	+	...	*	a_0	+	

Beispiel

Mit den obigen Programmen wurden die Funktionswerte der Polynome

$$p_5(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} \text{ und}$$

$$q_5(x) = x - 0.16605x^3 + 0.00761x^5$$

berechnet für x = 0.0(0.1)2.0 und mit den Funktionswerten von sin x verglichen.

x	$p_5(x)$	$q_5(x)$	sin x
0.0	0.000000000	0.000000000	0.000000000
0.1	0.099833417	0.099834026	0.099833417
0.2	0.198669333	0.198674035	0.198669331
0.3	0.295520250	0.295535142	0.295520207
0.4	0.389418667	0.389450726	0.389418342
0.5	0.479427083	0.479481563	0.479425539
0.6	0.564648000	0.564724954	0.564642473
0.7	0.644233917	0.644323863	0.644217687
0.8	0.717397333	0.717476045	0.717356091
0.9	0.783420750	0.783443173	0.783326910
1.0	0.841666667	0.841560000	0.841470985
1.1	0.891587583	0.891243431	0.891207360
1.2	0.932736000	0.932001715	0.932039086
1.3	0.964774417	0.963443547	0.963558185
1.4	0.987485333	0.985287206	0.985449730
1.5	1.000781250	0.997369658	0.997494987
1.6	1.004714667	0.999655834	0.999573603
1.7	0.999488083	0.992247468	0.991664811
1.8	0.985464000	0.975352525	0.973847631
1.9	0.963174916	0.949454184	0.946300088
2.0	0.933333333	0.915120000	0.909297427

p_5 ist Partialsumme der Potenzreihenentwicklung für sin x, q_5 ist eine Polynom-Approximation für sin mit der Eigenschaft $\sin x = q_5(x) + x \cdot \varepsilon(x)$, $|\varepsilon(x)| \leq 2 \cdot 10^{-4}$, $0 \leq x \leq \frac{\pi}{2}$ (Tschebyscheff-Approximation). Es zeigt sich, daß die Partialsumme sehr gute Näherungswerte liefert in der Nähe des Nullpunktes, des Entwicklungspunktes der Potenzreihe; mit zunehmendem Abstand vom Nullpunkt wächst ihr Fehler. Im Gegensatz dazu approximiert q_5 gleichmäßig in dem Intervall $[0, \frac{\pi}{2}]$.

2.2. Unendliche Reihen

Der Wert einer unendlichen Reihe

$$a_0 + a_1 + a_2 + \ldots = \sum_{k=0}^{\infty} a_k$$

wird approximiert durch die Partialsummen

$$s_n = a_0 + a_1 + \ldots + a_n = \sum_{k=0}^{n} a_k .$$

Mit den Gleichungen $s_0 = a_0$ und $s_n = s_{n-1} + a_n$, $n = 1,2,3,\ldots$, ist es möglich, die Partialsummen rekursiv zu bestimmen. Daher verläuft ihre Berechnung im wesentlichen jeweils nach der folgenden Struktur.

Algorithmus Anzeige: $s_n = a_0 + a_1 + \ldots + a_n$ für $n = 0,1,2,\ldots$

n	s

```
n ← 0
s ← a_n
Anzeige s
n ← n + 1
s ← s + a_n
```

Bei der Umsetzung in Taschenrechnerprogramme sind in der zweiten und fünften Zeile des Algorithmus die Anweisungen zur Berechnung des Summanden a_n einzufügen, oder es erfolgt ein entsprechender Unterprogrammaufruf. Wir erläutern dies am Beispiel der alternierenden harmonischen Reihe und an Potenzreihen.

Beispiel

Wir berechnen Näherungswerte für die alternierende harmonische Reihe

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots = \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} = \ln 2.$$

Algorithmus Anzeige: $s_n = \sum_{k=1}^{n} \frac{(-1)^{k-1}}{k}$ für n = 1,2,...

n	s	v

```
n ← 1
s ← 1
v ← 1
Anzeige s
n ← n + 1
v ← - v
s ← s + v/n
```

```
                1.
              0.5
.8333333333
.5833333333
.7833333333
.6166666667
.7595238095
.6345238095
.7456349206
.6456349206
.7365440115
.6532106782
.7301337551
.6587051837
.7253718504
```

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2
n	s	v

```
001   1 STO 0 STO 1 STO 2              n←1, s←1, v←1
005   LBL 0 RCL 1 PRTx                 Anzeige s
008   1 STO + 0 CHS STO * 2            n ← n + 1, v ← - v
012   RCL 2 RCL 0 ÷ STO + 1 GTO 0      s ← s + v/n ,
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}
n	s	v

```
000   1 STO 0 STO 1 STO 2              n←1, s←1, v←1
007   Lbl INV RCL 1 Prt                Anzeige s
012   1 SUM 0 +/- Prd 2                n ← n + 1 , v ← - v
018   RCL 2 ÷ RCL 0 = SUM 1 GTO INV    s ← s + v/n ,
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Start der Rechnung und Anzeige der Folge der Partialsummen durch RTN R/S bzw. RST R/S.

Bei zahlreichen unendlichen Reihen erweist es sich als günstig, zunächst einen analytischen Ausdruck für die Quotienten

$$q(n) = \frac{a_{n+1}}{a_n}, \quad n = 0,1,2,\ldots,$$

zu bestimmen. Ausgehend von dem ersten Summanden a_0 werden dann die weiteren Summanden rekursiv berechnet gemäß $a_{n+1} = q(n)a_n$, $n = 0,1,2,\ldots$. Für Potenzreihen

$$\sum_{k=0}^{\infty} c_k x^k$$

haben die Quotienten die Gestalt

$$q(n) = \frac{c_{n+1}}{c_n}\, x.$$

Algorithmus Eingabe: Unterprogramm für q(n);
a = a_0; bei Potenzreihen zusätzlich x.
Anzeige: $s_n = a_0 + a_1 + \ldots + a_n$ für
$n = 0,1,2,\ldots$.

x	a	n	s

```
s ← a
n ← 0
Anzeige s
a ← q(n)a
s ← s + a
n ← n + 1
```

Beispiel

Der Integralsinus $Si(x) = \int_0^x \frac{\sin t}{t}\,dt$ besitzt die Potenzreihenentwicklung

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

Hier ist $q(n) = -\frac{2n+1}{2n+3} \frac{x^2}{(2n+2)(2n+3)}$ und $a_0 = x$.

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	R_3
x	a	n	s

Hauptprogramm

```
001   STO 0 STO 1 STO 3              R0←x, a←a0, s←a
004   0 STO 2                        n←0
006   LBL 0 RCL 3 PRTx               Anzeige s
009   GSB 1 STO * 1 RCL 1 STO + 3    a←q(n)a, s ← s + a
013   1 STO + 2 GTO 0                n ← n + 1 ,
```

Unterprogramm für q(n)

```
016   LBL 1 RCL 2 .5 + ENTER ENTER   RX ← n + 0.5
023   1 + ÷                          RX ← (n+0.5)/(n+1.5)
026   RCL 2 2 * 2 + ENTER ENTER      RX ← 2n + 2
033   1 + * ÷                        RX ← (n+0.5)/(n+1.5) · 1/((2n+2)(2n+3))
037   RCL 0 x² * CHS                 RX ← q(n)
041   RTN                            Rücksprung
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}
x	a	n	s

Hauptprogramm

```
000   STO 0 STO 1 STO 03             R00←x, a←a0, s←a
006   0 STO 2                        n←0
009   Lbl INV RCL 3 Prt              Anzeige s
014   SBR lnx Prd 1 RCL 1 SUM 3      a←q(n)a, s ← s + a
022   Op 22 GTO INV                  n ← n + 1 ,
```

Unterprogramm für q(n)

026	Lbl lnx ((RCL 2 + .5)	$R_X \leftarrow n + 0.5$
036	÷ (RCL 2 + 1.5) ÷	$R_X \leftarrow \frac{n+0.5}{n+1.5}$
046	(2 * RCL 2 + 2) ÷	$R_X \leftarrow \frac{n+0.5}{n+1.5} \frac{1}{2n+2}$
055	(2 * RCL 2 + 3) *	$R_X \leftarrow \frac{n+0.5}{n+1.5} \frac{1}{(2n+2)(2n+3)}$
064	RCL 0 x^2 +/-)	$R_X \leftarrow q(n)$
069	INV SBR	Rücksprung

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Start der Rechnung und Anzeige der Partialsummen zur Approximation von Si(x) durch x RTN R/S bzw. x RST R/S.

Mögliche Wiederholungen: Schritt 2.

Diese Programme können bei der Berechnung anderer unendlicher Reihen verwendet werden; hierzu ist das Unterprogramm zur Berechnung der Quotienten q(n) geeignet zu ersetzen, und zu Beginn des Hauptprogramms wird die Wertzuweisung $a \leftarrow a_0$ dadurch den jeweiligen Erfordernissen angepaßt, daß der Wert a_0 zwischen STO 0 und STO 1 eingefügt wird, falls er von x verschieden ist.

x = 0.5	x = 1	x = 1.5
0.500000000	1.000000000	1.500000000
0.493055556	0.944444445	1.312500000
0.493107639	0.946111111	1.325156250
0.493107418	0.946082767	1.324671955
0.493107418	0.946083073	1.324683726
0.493107418	0.946083070	1.324683525
	0.946083070	1.324683531
	0.946083070	1.324683531
		1.324683531

x = 2	x = 3	x = 10
2.000000000	3.000000000	10.00000000
1.555555556	1.500000000	-45.55555555
1.608888889	1.905000000	121.1111112
1.605260771	1.643010204	-162.3356069
1.605417542	1.849036990	143.8568351
1.605412878	1.848633544	-83.68960490
1.605412979	1.848653239	35.64150160
1.605412977	1.848652567	-11.33956997
1.605412977	1.848652528	5.198393890
1.605412577	1.848652527	0.871743757
	1.848652527	1.803788576
	1.848652527	1.635607258
		1.661395059
		1.657993692
		1.658383691
		1.658344461
		1.658347951
		1.658347674
		1.658347694
		1.658347693
		1.658347693
		1.658347693

Konvergiert eine unendliche Reihe $a_0 + a_1 + a_2 + \ldots$ auf Grund des Leibnizschen Konvergenzkriteriums oder nach dem Quotientenkriterium, so wird der Fehler der Partialsumme s_n gegenüber dem Reihenwert abgeschätzt durch $\delta|a_{n+1}|$ mit einer von n unabhängigen Zahl δ; für das Leibnizsche Kriterium ist $\delta = 1$ (s. [9] 1.2.). Man kann dann die obigen Algorithmen dadurch erweitern, daß man die Berechnung weiterer Partialsummen abbricht, sobald $|a_{n+1}|$ eine vorgegebene Schranke unterschreitet.

Für größere Werte von x verhalten sich Potenzreihen häufig numerisch ungünstig. Treten nämlich zu Beginn der Summation sehr große Summanden auf, so wirkt sich schließlich die spätere Addition der dann kleiner werdenden Summanden numerisch nicht mehr aus, so daß keineswegs die erhoffte Genauigkeit erreicht wird. Im Fall alternierender Reihen können zusätzlich Auslöschungseffekte bei der Subtraktion nahezu gleich großer Zahlen zu erheblichen Genauigkeitsverlusten führen.

3. Berechnung von Nullstellen

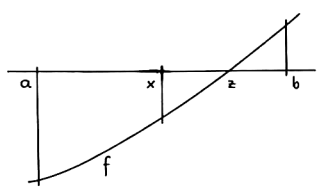
Wir befassen uns in diesem Kapitel mit einigen einfachen Verfahren zur Berechnung von Nullstellen reeller Funktionen. Die Methode der Intervallhalbierung und die quadratische Interpolation beruhen auf einem Vorzeichenwechsel der betrachteten Funktion und sind daher auf reellwertige Funktionen beschränkt. Die anderen Verfahren können auf größeren Rechenanlagen auch bei komplexwertigen Funktionen verwendet werden. Verallgemeinerungen der hier betrachteten Methode der sukzessiven Approximation dienen in den Kapiteln 9 und 10 zur iterativen Lösung linearer und nichtlinearer Gleichungssysteme. Entsprechend ist es auch möglich, das Newtonsche Verfahren auf nichtlineare Gleichungssysteme zu übertragen.

3.1. Die Methode der Intervallhalbierung

Ist f eine stetige Funktion, die an den Punkten a und b verschiedene Vorzeichen hat,

$$f(a)f(b)<0,$$

so gibt es mindestens eine Nullstelle z von f zwischen a und b (Zwischenwertsatz). Eine solche Nullstelle kann durch Intervallhalbierung näherungsweise ermittelt werden. Man berechnet dazu das Vorzeichen des Funktionswertes f(x) im Mittelpunkt $x = \frac{1}{2}(a + b)$. Ist $f(x) \neq 0$, so hat eines der beiden Teilintervalle mit den Endpunkten a und x bzw. x und b wieder die Eigenschaft, daß f an den Endpunkten verschiedene Vorzeichen hat, und es enthält somit eine Nullstelle von f. Dieses Vorgehen wird wiederholt, bis die Intervallänge und damit die Kenntnis über die Lage der Nullstelle eine gewünschte Genauigkeit δ



unterschreitet. Bei dieser Methode wird also durch fortgesetzte Intervallhalbierung eine Nullstelle eingeschachtelt. Daher sind auch die Bezeichnungen Intervallschachtelungsverfahren sowie Bisektion gebräuchlich.

Im n-ten Schritt des Verfahrens dient die Intervallmitte x_n als Näherung für die Nullstelle z, und es gilt die Ungleichung

$$|x_n - z| \leq |x_n - x_{n-1}| = \frac{|b-a|}{2^n}$$

Diese Ungleichung ist eine sogenannte a-priori-Fehlerabschätzung. Mit ihr ist man in der Lage, bereits vor Beginn der Rechnung die Anzahl der Schritte des Verfahrens zu ermitteln, die höchstens erforderlich sind, um eine Näherung der Nullstelle z mit gewünschter Genauigkeit zu erreichen.

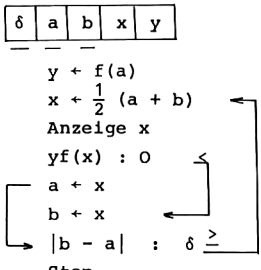
Algorithmus 1

Voraussetzung: $f(a)f(b)<0$

Eingabe: Unterprogramm für f;
δ, a, b.

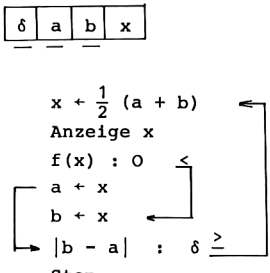
Anzeige: $x_0, x_1, x_2, \ldots, x_n$.

n ist der kleinste Index mit $|x_n - x_{n-1}| < \delta$.



Algorithmus 2

Voraussetzung: $f(a)>0, f(b)<0$



Für beide Algorithmen ist das Bestehen der Ungleichung a<b nicht erforderlich. Vor Beginn der Rechnung muß die Gültigkeit des Vorzeichenwechsels f(a)f(b)<0 überprüft werden. In der Rechnung selbst tritt f(b) nicht auf, so daß dafür kein Speicherplatz reserviert wird.

Bei dem Vergleich von yf(x) mit 0 in Algorithmus 1 ist von y nur das Vorzeichen von Bedeutung; deshalb wird die anfängliche Wertzuweisung y←f(a) während der gesamten Rechnung nicht geändert.

In Algorithmus 2 ist a derjenige Punkt, für den der Funktionswert positiv ist. Durch diese Festlegung vereinfacht sich der Algorithmus gegenüber Algorithmus 1; es wird kein Speicherplatz für y benötigt, und es entfällt die Berechnung des Produkts yf(x) in jedem Schritt des Algorithmus. Außerdem tritt bei Algorithmus 2 nur an einer Stelle die Berechnung eines Wertes der Funktion f auf. Daher ist diese Form des Algorithmus auch für programmierbare Taschenrechner ohne Unterprogrammtechnik möglich.

In beiden Algorithmen wird nicht überprüft, ob eine erhaltene Näherung x für die gesuchte Nullstelle bereits selbst Nullstelle ist, also f(x) = 0 erfüllt. Infolge der unvermeidlichen Rundungsfehler bei der Auswertung der Funktion f ist nämlich eine Abfrage "f(x)=0 ?" nicht sinnvoll. Eine derartige Abfrage kann jedoch numerisch dadurch realisiert werden, daß man bei Kenntnis der Größenordnung der Funktionswerte von f und der Rechengenauigkeit die Berechnung weiterer Näherungen für die Nullstelle dann abbricht, wenn f(x) dem Betrag nach eine vorgegebene Schranke ε unterschreitet. Die zugehörige Verzweigung würde in den obigen Algorithmen eingefügt werden zwischen Anzeige x und der Verzweigung in Abhängigkeit des Vorzeichens von f(x).

Im folgenden geben wir die Taschenrechnerprogramme zu Algorithmus 1 an.

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	R_3	R_4
δ	a	b	x	y

Hauptprogramm

```
001   LBL A RCL 1 B STO 4                y ← f(a)
005   LBL O RCL 1 RCL 2 + 2 ÷ STO 3      x ← 1/2 (a + b)
012   PRTx                               Anzeige x
013   B RCL 4 * x<O? GTO 1               wenn yf(x)<O
018   RCL 3 STO 1 GTO 2                  a←x
021   LBL 1 RCL 3 STO 2                  b←x
024   LBL 2 RCL 2 RCL 1 - ABS            R_X←|b-a|
029   RCL O x≤y? GTO O                   wenn δ≤|b-a|
032   R/S                                Stop
```

Unterprogramm f(x)

```
033   LBL B
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}
δ	a	b	x	y

Hauptprogramm

```
000   Lbl A RCL 1 B STO 4                y ← f(a)
007   Lbl INV RCL 1 + RCL 2 =            R_X ← a + b
015   ÷ 2 = STO 3                        x ← 1/2 (a + b)
020   Prt                                Anzeige x
021   B * RCL 4 = CP INV x≥t lnx         wenn yf(x)<O
030   RCL 3 STO 1 GTO CE                 a←x
036   Lbl lnx RCL 3 STO 2                b←x
042   Lbl CE RCL O x↔t                   R_T←δ
047   RCL 2 - RCL 1 = |x| x≥t INV        wenn |b-a|≥δ
056   R/S                                Stop
```

Unterprogramm f(x)

057 Lbl B

Anleitung zur Verwendung der Programme

1. Eingabe des Hauptprogramms.
2. Eingabe des Unterprogramms B für die Funktion f; es wird der Funktionswert von f für denjenigen Punkt berechnet, der sich bei Aufruf des Unterprogramms im Anzeigeregister befindet; bei Rücksprung in das aufrufende Programm wird der berechnete Funktionswert im Anzeigeregister übergeben.
3. Eingabe der Schranke δ für die Beendigung der Rechnung sowie der Anfangspunkte a und b
   bei den Rechnern HP-67/97: δ STO 0 a STO 1 b STO 2,
   bei den Rechnern TI-58/59: δ STO 00 a STO 01 b STO 02.
4. Berechnung und Anzeige der Näherungen $x_0, x_1, \ldots, x_n$ für z: A.

Mögliche Wiederholungen:

ab Schritt 3 für eine andere Auswahl von δ, a, b;
ab Schritt 2 für eine andere Funktion f.

Beispiel

Die Funktion $f(x) = \sin x$, $2 \leq x \leq 4$, besitzt die Nullstelle $z = \pi$. z wird durch Intervallhalbierung näherungsweise berechnet.

Das Unterprogramm B lautet
für HP-67/97:

033 LBL B SIN RTN $\quad R_X \leftarrow \sin x$, Rücksprung,

für TI-58/59:

057 Lbl B sin INV SBR $\quad R_X \leftarrow \sin x$, Rücksprung.

Vor Start der Programmausführung muß der Rechner auf das Bogenmaß (Radiant) eingestellt werden: RAD. Mit $\delta = 10^{-5}$, a = 4, b = 2 entstand das folgende Ergebnis.

```
          3.
        3.5
       3.25
      3.125
     3.1875
    3.15625
   3.140625
  3.1484375
 3.14453125
3.142578125
3.141601563
3.141113281
3.141357422
3.141479492
3.141540527
3.141571045
3.141586304
3.141593933
```

In dem Beispiel ist der Wert 3.141601563 eine wesentlich genauere Approximation für $z = \pi$ als die nachfolgenden Näherungen; jedoch kann diese Eigenschaft von dem Verfahren nicht ausgenutzt werden. Das Verfahren konvergiert langsam aber sicher. Zufällig erreichte sehr gute Näherungen für z können höchstens durch a-posteriori-Fehlerabschätzungen erkannt werden, nicht jedoch auf Grund der hier verwendeten a-priori-Fehlerabschätzung.

3.2. Die Methode der sukzessiven Approximation

FIXPUNKTE

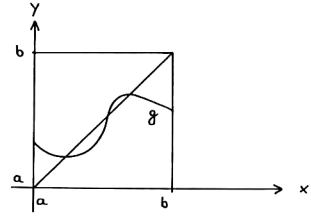
Ist eine Funktion g gegeben, so heißt eine Zahl z ihres Definitionsbereichs ein Fixpunkt von g, wenn sie Lösung der folgenden Gleichung ist:

$$z = g(z).$$

In der graphischen Darstellung der Funktion g sind Fixpunkte gegeben als Schnittpunkte der Geraden $y = x$ mit der Kurve $y = g(x)$. Für eine stetige Funktion g auf einem Intervall $I = [a,b]$, die I in sich abbildet,

$g(x) \in I$ für $x \in I$,

gibt es mindestens einen Fixpunkt in I.



Zur rechnerischen Ermittlung von Fixpunkten dient das Verfahren der sukzessiven Approximation. Ausgehend von einem Anfangswert x_0 wird eine Folge von Näherungen berechnet nach der Vorschrift

$$x_{t+1} = g(x_t), \quad t = 0,1,2,\ldots .$$

Unter der obigen Voraussetzung $g: I \to I$ kann diese Folge für beliebige Anfangsnäherungen x_0 aus I gebildet werden.

Die Funktion g wird kontrahierend genannt, wenn sie eine Lipschitz-Bedingung

$$|g(x) - g(y)| \leq q|x - y|, \quad x,y \in I, \text{ mit } 0 \leq q < 1$$

erfüllt. Für differenzierbare Funktionen ist diese Bedingung gleichwertig zu

$$|g'(x)| \leq q < 1, \quad x \in I.$$

Ist nun $g: I \to I$ kontrahierend, so gibt es eine eindeutig bestimmte Lösung z der Gleichung $z = g(z)$ in I, und für jedes $x_0 \in I$ konvergiert die Folge $x_0, x_1, x_2, \ldots$ gegen z mit den Fehlerabschätzungen

$$|x_t - z| \leq \frac{q}{1-q}|x_t - x_{t-1}| \leq \frac{q^t}{1-q}|x_1 - x_0|, \quad t = 1,2,\ldots$$

(s. [9] 2.2.). Die hierin enthaltene a-priori-Fehlerabschätzung $|x_t - z| \leq q^t/(1 - q) \cdot |x_1 - x_0|$ erlaubt die folgende Anwendung: schon zu Beginn der Rechnung kann aus der Kenntnis von q, x_0 und x_1 festgelegt werden, wie viele Schritte höchstens erforderlich sind, um den Fixpunkt z mit einer gewünschten Genauigkeit näherungsweise zu berechnen. Die a-posteriori-Fehlerabschätzung $|x_t - z| \leq q/(1 - q) \cdot |x_t - x_{t-1}|$ kann erst während des Ablaufs der Rechnung eingesetzt werden; sie liefert im allgemeinen eine genauere Abschätzung des Fehlers. Aus der a-posteriori-Ungleichung folgt, daß hier z mit entsprechender Genauigkeit approximiert ist, wenn sich zwei aufeinanderfolgende Näherungen nur noch wenig unterscheiden.

Die Methode der sukzessiven Approximation wird bereits durch den folgenden einfachen Algorithmus 1 erfaßt, in welchem die aufeinanderfolgenden Näherungen angezeigt werden. In Algorithmus 2 wird die obige Folgerung aus der a-posteriori-Fehlerabschätzung berücksichtigt, indem die Rechnung beendet wird, sobald zwei aufeinanderfolgende Näherungen die Ungleichung $|x_n - x_{n-1}| < \varepsilon$ erfüllen.

Algorithmus 1

Eingabe: Unterprogramm für g
$x = x_0$
Anzeige: $x_0, x_1, x_2, \ldots$

x

Anzeige x
$x \leftarrow g(x)$

Algorithmus 2

Eingabe: Unterprogramm für g
ε, $x = x_0$
Anzeige: $x_0, x_1, \ldots, x_n$.

ε	x	y

Anzeige x
$y \leftarrow g(x)$
$|x - y|$: $\varepsilon <$
$x \leftarrow y$
Anzeige y
Stop

Interessiert man sich im Fall des Algorithmus 2 nur für die letzte Näherung x_n, so entfällt die erste Zeile mit "Anzeige x". Nachfolgend sind Taschenrechnerprogramme zu Algorithmus 2 wiedergegeben.

Programm für HP-67/97

Datenspeicher

R_0	R_1
ε	x

Hauptprogramm

```
001   LBL A STO 1 PRTx
004   B ENTER ENTER
007   RCL 1 - ABS RCL 0 x>y? GTO 0
013   R↓ R↓ GTO A
016   LBL 0 R↓ R↓ PRTx R/S
```

$R_1 \leftarrow x$, Anzeige x
$y \leftarrow g(x)$
wenn $|x - y| < \varepsilon$
$x \leftarrow y$
Anzeige y, Stop

Unterprogramm g(x)

021 LBL B

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}
ε	x	y

Hauptprogramm

```
000   Lbl A STO 1 Prt
005   B STO 2 RCL 0 x↔t
011   RCL 1 - RCL 2 STO 1 =
019   |x| INV x≥t INV
023   RCL 1 GTO A
027   Lbl INV RCL 2 Prt R/S
```

```
→ R01←x, Anzeige x
  y←g(x), RT←ε
  RX ← x - y und x←y
  wenn |x - y|<ε
  RX←x
  Anzeige y, Stop
```

Unterprogramm g(x)

033 Lbl B

Anleitung zur Verwendung der Programme

1. Eingabe des Hauptprogramms.
2. Eingabe des Unterprogramms B für die Funktion g; aus x im Anzeigeregister wird der Funktionswert g(x) berechnet; x ist gleichzeitig auch in R_1 gespeichert. Bei Rücksprung in das aufrufende Programm wird der berechnete Funktionswert im Anzeigeregister übergeben.
3. Speichern von ε für die Abbruchbedingung: ε STO 0 bzw. ε STO 00.
4. Berechnung und Anzeige von $x_0, x_1, \ldots, x_n$: x_0 A.

Mögliche Wiederholungen:

ab Schritt 4 für eine andere Anfangsnäherung x_0;
ab Schritt 3 für eine andere Genauigkeitsabfrage;
ab Schritt 2 für eine andere Funktion g.

In dem Programm für die Rechner HP-67/97 ist kein Datenregister für y reserviert, da es ausreicht, y im Stapelregister zu führen. In dem Programm für die Rechner TI-58/59 dient das Vergleichsregister R_T hier nur zur Aufnahme von ε. Wird R_T in dem Unterprogramm B nicht verwendet, so kann für die TI-Rechner Schritt 3 der Anleitung zur Verwendung des Programms ersetzt werden durch ε x↔t, und die Anweisungen 008 bis 010 können entfallen. Das Datenregister R_{00} wird in diesem Fall dann nicht zur Aufnahme von ε benötigt.

Beispiel

Die Funktion g(x) = cos x erfüllt die obigen Voraussetzungen zum Beispiel in dem Intervall I = [0,1.5]. Daher kann die Lösung von z = cos z durch sukzessive Approximation ermittelt werden.

Das Unterprogramm B lautet
für HP-67/97

```
021   LBL B COS RTN          Rx←cos x, Rücksprung,
```

für TI-58/59

```
033   Lbl B cos INV SBR      Rx←cos x, Rücksprung.
```

Vor dem Beginn der Programmausführung werden die Rechner auf das Bogenmaß eingestellt: RAD.

$\varepsilon = 10^{-2}$, $x_0 = 1.5$

```
1.500000000  ***
0.070737202  ***
0.997499167  ***
0.542484992  ***
0.856469709  ***
0.655108802  ***
0.792981646  ***
0.701724168  ***
0.763730311  ***
0.722261082  ***
0.750312886  ***
0.731475556  ***
0.744189587  ***
0.735637098  ***
```

$\varepsilon = 10^{-3}$, $x_0 = 0.5$

```
0.500000000  ***
0.877582562  ***
0.639012494  ***
0.802685101  ***
0.694776027  ***
0.768195831  ***
0.719165446  ***
0.752355760  ***
0.730081063  ***
0.745120341  ***
0.735006309  ***
0.741826523  ***
0.737235725  ***
0.740329652  ***
0.738246236  ***
0.739649963  ***
0.736704539  ***
```

NULLSTELLEN

Das Verfahren der sukzessiven Approximation kann auch zur Berechnung von Nullstellen verwendet werden.Hierzu müssen die Nullstellengleichungen der Form f(x) = 0 in gleichwertige Fixpunktgleichungen x = g(x) umgeformt werden. Eine einfache Möglichkeit ist beispielsweise g(x) = x + f(x). Die Fixpunkte von g sind dann die Nullstellen von f.

Beispiel

Wir betrachten wieder die Funktion f(x) = sin x, $2 \leq x \leq 4$, mit der Nullstelle z = π. Für g(x) = x + sin x, $2 \leq x \leq 4$, sind die obigen Voraussetzungen erfüllt. Das Unterprogramm lautet jetzt

für HP-67/97

021	LBL B ENTER SIN +	R_X ← x + sin x
025	RTN	Rücksprung,

für TI-58/59

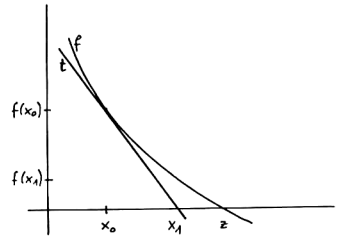
033	Lbl B (CE + CE sin)	R_X ← x + sin x
041	INV SBR	Rücksprung.

Nach Einstellen auf Bogenmaß durch die Anweisung RAD ergaben sich mit $\varepsilon = 10^{-3}$ und $x_0 = 2$ die folgenden Näherungen.

```
                2.
2.909297427
3.139509133
3.141592652
3.141592654
```

3.3. Das Newtonsche Verfahren

Bei dem Newtonschen Verfahren zur näherungsweisen Bestimmung von Nullstellen geht man aus von einer Näherung x_0 für die Nullstelle z von f. Die Tangente in dem Kurvenpunkt $(x_0, f(x_0))$ ist gegeben durch



$$t(x) = f(x_0) + f'(x_0)(x - x_0).$$

Ihre Nullstelle x_1 wird als neue Näherung für z betrachtet,

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} .$$

Durch Wiederholen dieses Vorgehens mit der jeweils erhaltenen Näherung entsteht eine Folge von Punkten (x_t) nach der Vorschrift des Newtonschen Verfahrens,

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)} , \quad t = 0,1,2,\ldots .$$

Hier darf die Ableitung f' nirgends in dem betrachteten Bereich verschwinden. Ist f' stetig, so kann $f'(x) \neq 0$ in einer Umgebung der Nullstelle z durch die Voraussetzung $f'(z) \neq 0$ sichergestellt werden. Das bedeutet, daß f in z nur eine einfache Nullstelle haben soll, was als Bedingung für günstiges Konvergenzverhalten der Folge (x_t) auch aus der geometrischen Bedeutung des Verfahrens verständlich ist.

Das Newtonsche Verfahren kann aufgefaßt werden als eine spezielle Methode der sukzessiven Approximation $x_{t+1} = g(x_t)$ mit der Funktion g,

$$g(x) = x - \frac{f(x)}{f'(x)} .$$

Auf Grund von $g'(x) = f(x)f''(x)/f'(x)^2$ ist g eine Kontraktion in einer geeigneten Umgebung der Nullstelle z von f. Liegt die Näherung x_0 hinreichend nahe bei der gesuchten Nullstelle,

so zeigt sich hier jedoch ein wesentlich besseres Konvergenzverhalten: Ist f eine zweimal stetig differenzierbare Funktion mit $f'(x) \neq 0$, dann gibt es eine Konstante q, $0<q<1$, so daß Konvergenz herrscht mit der a-priori-Fehlerabschätzung

$$|x_t - z| \leq \text{const} \cdot q^{(2^t)}$$

sowie mit der a-posteriori-Fehlerabschätzung

$$|x_t - z| \leq \text{const} \cdot |x_t - x_{t-1}|^2$$

(s. [9] 2.3.). Die a-priori-Fehlerabschätzung zeigt die schnelle Konvergenz des Newtonschen Verfahrens in der Nähe der Nullstelle. Mit der a-posteriori-Fehlerabschätzung ist man in der Regel berechtigt, aus einem nur noch geringen Unterschied zweier aufeinanderfolgender Näherungen zu schließen, daß z bereits gut approximiert ist.

Liegt die Anfangsnäherung nicht dicht genug bei der gesuchten Nullstelle, so kann das Newtonsche Verfahren versagen. Das ist insbesondere in der Nähe von lokalen Extremalstellen oder Wendepunkten der Fall.

Häufig ist es zweckmäßig, zur numerischen Ermittlung der Nullstelle einer Funktion zunächst mit einem sicher konvergenten Verfahren zu beginnen, zum Beispiel mit der Methode der Intervallhalbierung; man berechnet damit Näherungen für eine Nullstelle z, jedoch konvergieren solche Verfahren langsam. Hat man die Nullstelle z mit mäßiger Genauigkeit approximiert, so liefern dann noch wenige Schritte des Newtonschen Verfahrens sehr gute Approximationen für z.

Beispiele

Wir betrachten zunächst wieder $f(x) = \sin x$, $2 \leq x \leq 4$. Es können die Programme für die sukzessive Approximation aus dem vorigen Abschnitt angewandt werden für $g(x) = x - f(x)/f'(x) = x - \tan x$. Das zugehörige Unterprogramm B für den dortigen Algorithmus lautet

für HP-67/97:

```
O21   LBL B ENTER TAN -          R_X ← x - tan x
O25   RTN                        Rücksprung,
```

für TI-58/59:

```
O33   Lbl B ( CE - CE tan )      R_X ← x - tan x
O41   INV SBR                    Rücksprung.
```

Mit $\varepsilon = 10^{-4}$ und den Anfangsnäherungen x_0 = 3, 2, 1.95, 1.9, 1.8 ergaben sich die folgenden Ergebnisse.

3.000000000	2.000000000	1.950000000	1.900000000	1.800000000
3.142546543	4.185039863	4.459475468	4.827097515	6.086261675
3.141592653	2.467893675	0.590220869	13.50657490	6.285770917
3.141592654	3.266186277	-0.079654710	12.13675297	6.283185301
	3.140943912	0.000168895	12.59491130	6.283185307
	3.141592654	-1.600000006-12	12.56636286	
	3.141592654	0.000000006	12.56637061	

Für $x_0 = 3$ zeigt sich die schnelle Konvergenz, wie sie für das Newtonsche Verfahren erwartet wird. Mit $x_0 = 2$ herrscht langsamere Konvergenz als bei der Lösung derselben Aufgabe mit der Methode der sukzessiven Approximation $x_{t+1} = x_t + \sin x_t$ im vorangehenden Abschnitt; x_0 liegt nicht nahe genug bei der Nullstelle $z = \pi$, so daß die schnelle Konvergenz des Newtonschen Verfahrens nicht von Anfang an gilt. Die Anfangsnäherungen 1.95, 1.9 und 1.8 zeigen, daß das Newtonsche Verfahren in der Nähe von Extremalstellen von der benachbarten Nullstelle wegführen kann.

Die positive Nullstelle von $f(x) = x^2 - a$, $a>0$, ist $z = \sqrt{a}$. Für diese Funktion f lautet das Newtonsche Verfahren

$$x_{t+1} = x_t - \frac{x_t^2-a}{2x_t} = \frac{1}{2}\left(x_t + \frac{a}{x_t}\right).$$

Für jedes $x_0>0$ herrscht Konvergenz gegen $\sqrt{a}$, für jedes $x_0<0$ konvergieren die Näherungen $x_1, x_2, \ldots$ gegen $-\sqrt{a}$. Dies ist das historische Verfahren von Heron zur Berechnung von

$\sqrt{a}$. Es ordnet sich hier als Spezialfall des Newtonschen Verfahrens ein, wodurch seine schnelle Konvergenz verständlich wird.

Entsprechend liefert das Newtonsche Verfahren für $f(x) = x^\nu - a$, $a>0$, $\nu \neq 0$, bei geeigneter Anfangsnäherung x_0 die Lösung $z = a^{1/\nu}$. Die Näherungsfolge wird berechnet gemäß

$$x_{t+1} = x_t - \frac{x_t^\nu - a}{\nu \cdot x_t^{\nu-1}} = \frac{1}{\nu}\left((\nu-1)x_t + \frac{a}{x_t^{\nu-1}}\right).$$

DAS NEWTONSCHE VERFAHREN FÜR POLYNOME

Im Fall von Polynomen

$$f(x) = a_0 + a_1 x + \ldots + a_n x^n$$

ist die Berechnung der Funktionswerte von f und f' besonders einfach. Man erhält sie rekursiv mit Hilfe des Horner-Schemas,

$$b_n = a_n,\ b_k = a_k + x b_{k+1},\quad k = n-1,\ldots,0\colon\ b_0 = f(x),$$
$$c_n = b_n,\ c_k = b_k + x c_{k+1},\quad k = n-1,\ldots,1\colon\ c_1 = f'(x).$$

Algorithmus Voraussetzung: x_0 sei geeignete Anfangsnäherung
Eingabe: $n, a_0, a_1, \ldots, a_n$;
$x = x_0$
Anzeige: $x_0, x_1, x_2, \ldots$

a_0	a_1	...	a_n	b	c	k	n	x

Anzeige x
b ← a_n
c ← b
k = n-1(-1)1
  b ← a_k + xb
  c ← b + xc
b ← a_0 + xb
x ← x - $\frac{b}{c}$

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	R_3	R_4	R_5	...	R_{n+4}		R_I
b	c	n	x	a_0	a_1	...	a_n		Adr a_k =k+4

$n \leq 20$

Einleseprogramm

```
001   STO 2 4 STO I                  R2←n, k←0
004   LBL 0 R/S                      Stop
006   STO (i) ISZ I GTO 0            Rk+4←ak, k ← k + 1,
```

Hauptprogramm

```
009   LBL A STO 3                    R3←x
011   LBL 1 RCL 3 PRTx               Anzeige x
014   RCL 2 4 + STO I                k←n
018   RCL (i) STO 0 STO 1 DSZ I      b←an, c←b, k ← n - 1
022   LBL 2 RCL 3 STO * 0 STO * 1    b←xb, c←xc
026   RCL (i) STO + 0 RCL 0 STO + 1  b ← ak + b, c ← b + c
030   DSZ I 4 I x>y? GTO 2           k ← k - 1, wenn k>0
035   RCL 3 STO * 0 RCL (i) STO + 0  b ← a0 + xb
039   RCL 0 RCL 1 ÷ STO - 3 GTO 1    x ← x - b/c,
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	...	R_{n+5}
b	c	n	x	Adr a_k =k+5	a_0	a_1	...	a_n

Einleseprogramm

```
000   STO 02 5 STO 4                 R02←n, k←0
005   Lbl INV R/S                    Stop
008   STO Ind 4 Op 24 GTO INV        Rk+5←ak, k ← k + 1,
```

Hauptprogramm

014	Lbl A STO 3	$R_{03} \leftarrow x$
018	Lbl lnx RCL 3 Prt	Anzeige x
023	RCL 2 + 5 = STO 4	k←n
030	RCL Ind 4 STO 0 STO 1 Op 34	$b \leftarrow a_n$, c←b, k ← n - 1
038	Lbl CE RCL 3 Prd 0 Prd 1	b←xb, c←xc
046	RCL Ind 4 SUM 0 RCL 0 SUM 1	$b \leftarrow a_k + b$, c ← b + c
054	Op 34 RCL 4 x↔t 5 INV x≥t CE	k ← k - 1, wenn k>0
063	RCL 3 Prd 0 RCL Ind 4 SUM 0	$b \leftarrow a_0 + xb$
071	RCL 0 ÷ RCL 1 = INV SUM 3	x ← x - b/c
080	GTO lnx	

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der Koeffizienten des Polynoms:
   n, RTN bzw. RST, R/S a_0 R/S a_1 R/S ... R/S a_n R/S.
3. Start der Rechnung und Anzeige der Näherungen
   $x_0, x_1, x_2, \ldots$: x_0 A.

Mögliche Wiederholungen:

ab Schritt 3 für eine andere Anfangsnäherung x_0,
ab Schritt 2 für ein anderes Polynom.

Beispiel

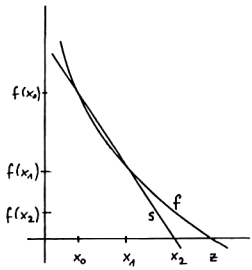
Durch Wahl verschiedener Anfangsnäherungen wurden die Nullstellen von $f(x) = -0.09 + 0.68\,x - 1.5x^2 + x^3$ bestimmt.

0.	0.6	0.7	2.
.1323529412	0.45	0.82	1.510479042
.2037627159	0.504	.7762900506	1.189406948
.2309793406	.4999981702	.7652804128	.9833332327
.2353170502	0.5	.7645779338	.8579773382
0.235424803	0.5	.7645751312	.7917791269
.2354248689	0.5	.7645751311	.7679605133
.2354248689		.7645751311	.7646382246
.2354248689		.7645751311	.7645751537
			.7645751311
			.7645751311
			.7645751311

3.4. Regula falsi

Die Regula falsi oder Sekantenmethode zur näherungsweisen Bestimmung von Nullstellen beruht auf der folgenden geometrischen Überlegung. Kennt man zwei Näherungen x_0 und x_1 für die Nullstelle z von f, so bildet man die Sekante an die Kurve $(x,f(x))$ durch die Punkte $(x_0,f(x_0))$ und $(x_1,f(x_1))$; sie ist gegeben durch

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



Ihre Nullstelle x_2 dient als neue Näherung für die gesuchte Nullstelle z,

$$x_2 = x_1-f(x_1)\frac{x_1 - x_0}{f(x_1)-f(x_0)} .$$

Durch Wiederholung dieser Überlegung mit den Näherungen x_1 und x_2 usw. erhält man eine Folge von Punkten x_t nach der Vorschrift

$$x_{t+1} = x_t - f(x_t)\frac{x_t - x_{t-1}}{f(x_t)-f(x_{t-1})} .$$

Wenn das Verfahren nicht mit einer Näherung $x_t = z$ abbricht, so konvergiert die Folge der Näherungen $x_0,x_1,x_2,\ldots$ gegen eine Nullstelle z von f. Unter denselben Voraussetzungen wie für das Newtonsche Verfahren herrscht hier ähnlich gute Konvergenz: Ist f zweimal stetig differenzierbar, gilt $f'(x) \neq 0$, und sind x_0, x_1 nahe genug bei z, dann gibt es eine Konstante q, $0<q<1$, so daß die a-priori-Fehlerabschätzung

$$|x_t - z| \leq \text{const}\cdot q^{k_t}, \quad t = 0,1,2,\ldots,$$

und die a-posteriori-Fehlerabschätzung

$$|x_t - z| \leq \text{const}\cdot|x_t - x_{t-1}|\cdot|x_t - x_{t-2}|, \quad t = 2,3,\ldots,$$

gelten (s. [9] 2.4). Hierbei sind k_t die Fibonaccischen Zahlen

$$k_0 = 1,\ k_1 = 1,\ k_t = k_{t-1} + k_{t-2},\quad t = 2,3,\ldots .$$

Mit ihrer Eigenschaft $k_t \doteq 0.723\cdot(1.618)^t$ für $t>>1$ erkennt man aus der a-priori-Fehlerabschätzung bei Vergleich mit dem Newtonschen Verfahren die etwas langsamere Konvergenz der Regula falsi. Auch die a-posteriori-Fehlerabschätzung zeigt, daß hier $|x_t-z|$ mit dem Produkt $|x_t-x_{t-1}|\cdot|x_t-x_{t-2}|$ abgeschätzt wird, was im allgemeinen größer ist als das Quadrat $|x_t-x_{t-1}|^2$ bei dem Newtonschen Verfahren. Dafür ist es hier jedoch im Gegensatz zum Newtonschen Verfahren nicht erforderlich, daß für jede Näherung x_t die Funktionswerte $f(x_t)$ und $f'(x_t)$ berechnet werden; denn an die Stelle von $f'(x_t)$ im Newtonschen Verfahren treten hier die Differenzenquotienten $(f(x_t) - f(x_{t-1}))/(x_t-x_{t-1})$. Da $f(x_{t-1})$ bereits in dem vorherigen Schritt als $f(x_t)$ verwendet wurde, erlaubt es das Umspeichern dieses Wertes in dem folgenden Algorithmus, daß die Anzahl der Funktionswertberechnungen im Vergleich zu dem Newtonschen Verfahren auf die Hälfte reduziert wird.

Algorithmus

Voraussetzung: x_0, x_1 seien geeignete Anfangsnäherungen

Eingabe: Unterprogramm für f, $x = x_0$, $y = x_1$.

Anzeige: x_0, x_1, x_2,... .

u	v	x	y	z

Anzeige x
u ← f(x)
Anzeige y
v ← f(y)
z ← $y - v\cdot\frac{y-x}{v-u}$
Anzeige z
u ← v
x ← y
y ← z

Programm für HP-67/97

R_0	R_1	R_2	R_3
u	v	x	y

Hauptprogramm

```
001   LBL A RCL 2 PRTx B STO 0        Anzeige x, u ← f(x)
006   RCL 3 PRTx                      Anzeige y
008   LBL 0 B STO 1                   v ← f(y)   ←┐
011   RCL 2 RCL 3 STO 2 -             R_X←x-y und x←y   │
015   RCL 0 RCL 1 STO 0 -             R_X←u-v und u←v   │
019   ÷ * STO - 3                     z←y-v (x-y)/(u-v) und y←z   │
022   RCL 3 PRTx GTO 0                Anzeige z,  ──────┘
```

Unterprogramm f(x)

```
025   LBL B
```

Programm für TI-58/59

Datenspeicher	R_{00}	R_{01}	R_{02}	R_{03}
	u	v	x	y

Hauptprogramm

```
000   Lbl A RCL 2 Prt B STO 0         Anzeige x, u ← f(x)
008   RCL 3 Prt                       Anzeige y
011   Lbl INV B STO 1                 v ← f(y)   ←┐
016   * ( RCL 2 - RCL 3 STO 2 ) ÷     R_X←v(x-y) und x←y   │
027   ( RCL 0 - RCL 1 STO 0 ) =       R_X←v (x-y)/(u-v) und u←v   │
037   INV SUM 3                       z←y - v (x-y)/(u-v) und y←z   │
040   RCL 3 Prt GTO INV               Anzeige z,  ──────┘
```

Unterprogramm f(x)

```
045   Lbl B
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe des Unterprogramms B für die Funktion f. Bei Aufruf dieses Unterprogramms ist die Stelle, an der die

Funktion f berechnet werden soll, im Anzeigeregister gespeichert; bei Rücksprung in das aufrufende Programm befindet sich dann der berechnete Funktionswert im Anzeigeregister.

3. Eingabe der Anfangsnäherungen gemäß
   x_0 STO 2 , x_1 STO 3 für HP-67/97,
   x_0 STO 02, x_1 STO 03 für TI-58/59.
4. Berechnung der Näherungen und Anzeige von $x_0, x_1, x_2, \ldots$: A.

Mögliche Wiederholungen:

ab Schritt 3 für andere Anfangsnäherungen;
ab Schritt 2 für eine andere Funktion f.

Im Unterschied zu dem vorherigen Flußdiagramm kommen wir in diesen Programmen ohne einen eigenen Datenspeicherplatz für z aus. Dies wird dadurch ermöglicht, daß der reziproke Differenzenquotient $\frac{y-x}{v-u}$ in der Form $\frac{x-y}{u-v}$ berechnet wird und daß gleichzeitig während dieser Berechnung v nach u und y nach x umgespeichert wird.

Beispiel

Mit der Funktion f(x) = sin x folgte mit denselben Unterprogrammen B wie bei der Methode der Intervallhalbierung für $x_0 = 2$, $x_1 = 4$:

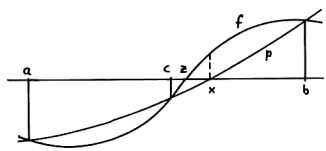
```
                RND
2.000000000 STO2
4.000000000 STO3
                GSBA
2.000000000  ***
4.000000000  ***
3.091528863  ***
3.147874957  ***
3.141596358  ***
3.141592654  ***
3.141592654  ***
                ERROR
```

Wenn hier bei den Beispielen zwei aufeinanderfolgende Näherungen x und y im Rahmen der Rechengenauigkeit übereinstimmen, so gilt dann auch u = f(x) = v = f(y). Daher ergibt sich im nächsten Schritt bei der Division durch u-v eine Fehlermeldung. Sie zeigt also hier das Ende des Näherungsverfahrens an.

3.5. Quadratische Interpolation

Wir betrachten nun wieder eine stetige Funktion f mit Vorzeichenwechsel, $f(a)f(b)<0$. Es existiert also mindestens eine Nullstelle z von f zwischen a und b. Vom Intervallhalbierungsverfahren her ist uns die Methode bekannt, eine Nullstelle z durch Intervalle einzuschachteln, in denen f das Vorzeichen wechselt. In Erweiterung dieses Verfahrens und des Vorgehens bei der Regula falsi wird hier nun das Interpolationspolynom zweiten Grades durch die Punkte (a,f(a)), (b,f(b)) und (c,f(c)) verwendet, mit $c = \frac{1}{2}(a+b)$. Seine Nullstelle x dient als Näherung für z. Ist $f(x) \neq 0$, so wählt man aus den Punkten a, b, c, x das kleinste Intervall, das einen Vorzeichenwechsel enthält, und wiederholt das Vorgehen.

Das Interpolationspolynom zweiten Grades durch die Punkte (a,f(a)), (b,f(b)), (c,f(c)) ist eindeutig bestimmt; mit der Abkürzung $h = \frac{1}{2}(b-a)$ lautet es



$$p(t) = f(c) + \frac{t-c}{2h}(f(b)-f(a)) + \frac{(t-c)^2}{2h^2}(f(a)+f(b)-2f(c)),\ t\in R.$$

p besitzt zwischen a und b genau eine Nullstelle. Zu ihrer Berechnung ist die Gestalt

$$p(c+hs) = \alpha s^2 + \beta s + \gamma, \quad s = \frac{1}{h}(t-c), \quad |s| \leq 1,$$

zweckmäßig, mit den Koeffizienten

$$\alpha = \frac{1}{2}(f(a)+f(b)-2f(c)), \quad \beta = \frac{1}{2}(f(b)-f(a)), \quad \gamma = f(c).$$

Es ist $\beta \neq 0$, während $\alpha = 0$ möglich ist. In diesem Fall wird p ein Polynom ersten Grades, die Sehne durch die Punkte (a,f(a)), (b,f(b)) der Kurve (x,f(x)), mit der Nullstelle $x = c - h\gamma/\beta$. Für $\alpha \neq 0$ besitzt die quadratische Gleichung $\alpha s^2 + \beta s + \gamma = 0$ zwei reelle Lösungen, wovon nur die betragsmäßig kleinere,

$$s_0 = \frac{1}{2\alpha}\left(-\beta + \operatorname{sgn}\beta \sqrt{\beta^2-4\alpha\gamma}\right),$$

der Ungleichung $|s_0| \leq 1$ genügt. Diese Darstellung ist für kleine Werte von α jedoch numerisch ungünstig. Wegen $\beta \neq 0$ kann s_0 auch geschrieben werden in der Form

$$s_0 = -\frac{2\gamma/\beta}{1+\sqrt{1-4\alpha\gamma/\beta^2}} .$$

Für $\alpha \to 0$ entsteht hieraus $s_0 = -\gamma/\beta$, die Lösung von $\beta s+\gamma = 0$. Die Nullstelle $x = c+hs_0$ des Polynoms p dient dann als Näherung für eine Nullstelle von f zwischen a und b,

$$x = c - \frac{2h\gamma/\beta}{1+\sqrt{1-4\alpha\gamma/\beta^2}} .$$

Das somit erhaltene Näherungsverfahren zur Berechnung von Nullstellen bei stetigen Funktionen mit Vorzeichenwechsel konvergiert, da es sich um eine Intervallschachtelung handelt. Es ist schnell konvergent, wie die Untersuchung numerischer Beispiele zeigt.

In dem folgenden Algorithmus wird in der beschriebenen Weise verfahren. In dem eingerahmten Teil wird dabei aus den Punkten a, b, c, x das kleinste Teilintervall bestimmt, das einen Vorzeichenwechsel von f enthält.

Algorithmus

Voraussetzung: $f(a)f(b)<0$

Eingabe: Unterprogramm für f, a, b.

Anzeige: Die Näherungen $x_0, x_1, x_2, \ldots$ für eine Nullstelle z von f.

a	b	c	p	q	r	s	x

$p \leftarrow f(a)$

$q \leftarrow f(b)$

$c \leftarrow \frac{1}{2}(a+b)$

$r \leftarrow f(c)$

$$r \leftarrow c + \frac{2(b-a)r/(p-q)}{1+\sqrt{1+8r(2r-p-q)/(p-q)^2}}$$

Anzeige x

$s \leftarrow f(x)$

pr : 0 <

rs : 0 <

$a \leftarrow x$ (4)

$p \leftarrow s$

$a \leftarrow c$ (3)

$p \leftarrow r$

$b \leftarrow x$

$q \leftarrow s$

ps : 0 <

$b \leftarrow c$ (2)

$q \leftarrow r$

$a \leftarrow x$

$p \leftarrow s$

$b \leftarrow x$ (1)

$q \leftarrow s$

In diesem Algorithmus wird nicht benötigt, daß beispielsweise a<b gilt.

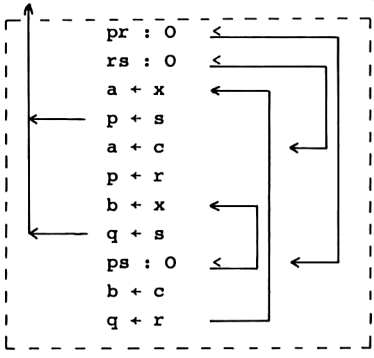
Die Speicherplätze p, q, r, s nehmen berechnete Funktionswerte von f auf, die im weiteren Verlauf umgespeichert werden. Dadurch ist es möglich, in jedem Schritt des Verfahrens mit nur zwei Funktionswertberechnungen auszukommen. Soll aus Gründen der Speicherplatzersparnis auf die Reservierung der zugehörigen Datenspeicherplätze verzichtet werden, so kann man dies dadurch erreichen, daß wiederholt dieselben Funktionswerte berechnet werden.

Bei der Ermittlung des kleinsten Teilintervalls mit Vorzeichenwechsel wird in dem eingerahmten Teil wie folgt verfahren. Auf Grund der Ausgangssituation ist $f(a)f(b)<0$. Zunächst wird nach dem Vorzeichen von f(c) entschieden. Im ersten Teil ist $pr = f(a)f(c)<0$, im zweiten Fall ist $pr = f(a)f(c)\geq 0$.

Erster Fall: x liegt zwischen a und c. Gilt die Möglichkeit (1): $ps = f(a)f(x)<0$, so wechselt f zwischen a und x das Vorzeichen; a bleibt für den nächsten Schritt des Verfahrens als Intervallendpunkt erhalten, b wird durch x ersetzt, die zugehörigen Funktionswerte entsprechend. Im Fall der Möglichkeit (2), $ps = f(a)f(x)\geq 0$, wechselt f das Vorzeichen zwischen x und c; daher wird b durch c sowie a durch x ersetzt, und die zugehörigen Funktionswerte werden analog umgespeichert.

Zweiter Fall: x liegt zwischen c und b. Im Fall (3), $rs = f(c)f(x)<0$, wechselt f das Vorzeichen zwischen c und x; daher werden a durch c und b durch x ersetzt mit den zugehörigen Funktionswerten. Ist (4): $rs = f(c)f(x)\geq 0$, so wechselt f das Vorzeichen zwischen x und b; b bleibt unverändert, a wird durch x ersetzt, der zugehörige Funktionswert entsprechend.

Die Berechnung des neuen Teilintervalls läßt sich kürzer fassen, was wir in den anschließenden Programmen verwenden.



Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7
a	b	c	p	q	r	s	x

Hauptprogramm

001	LBL A RCL 0 B STO 3	p ← f(a)
005	RCL 1 B STO 4	q ← f(b)
008	LBL 0 RCL 0 RCL 1 + 2 ÷ STO 2	c ← $\frac{1}{2}$(a+b)
015	B STO 5	r ← f(c)
017	RCL 1 RCL 0 - * 2 *	R_X ← 2(b-a)r
023	RCL 3 RCL 4 - ÷	R_X ← 2(b-a)r/(p-q)
027	RCL 5 2 * RCL 3 - RCL 4 -	R_X ← 2r-p-q
034	RCL 5 * 8 * RCL 3 RCL 4 - x^2 ÷	R_X←8r(2r-p-q)/$(p-q)^2$
043	1 + $\sqrt{x}$ 1 +	R_X←1+$\sqrt{1+8r...}$
048	÷ RCL 2 + STO 7	x←c+ $\frac{2(b-a)r/(p-q)}{1+\sqrt{1+8r...}}$
052	PRTx B STO 6	Anzeige x, s ← f(x)

```
055   RCL 3 RCL 5 * x<0? GTO 4          ┌wenn pr<0
060   RCL 5 RCL 6 * x<0? GTO 2          ┌wenn rs<0
065   LBL 1 RCL 7 STO 0                 │ a ← x  ←──┐
068   RCL 6 STO 3 GTO 0                 │ p ← s, ──┐│
071   LBL 2 RCL 2 STO 0 RCL 5 STO 3     └►a ← c, p ← r
076   LBL 3 RCL 7 STO 1                   b ← x  ←──┐
079   RCL 6 STO 4 GTO 0                   q ← s, ──┐│
082   LBL 4 RCL 3 RCL 6 * x<0? GTO 3    └►wenn ps<0 ──┐
088   RCL 2 STO 1 RCL 5 STO 4 GTO 1       b ← c, q ← r, ┘
```

Unterprogramm f(x)

```
093   LBL B
```

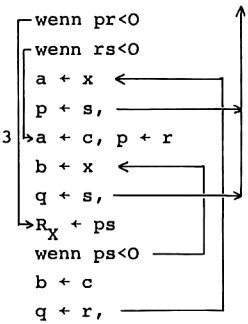
Programm für TI-58/59

Datenspeicher	R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	R_{07}
	a	b	c	p	q	r	s	x

Hauptprogramm

```
000   Lbl A RCL 0 B STO 3               p ← f(a)
007   RCL 1 B STO 4                     q ← f(b)
012   Lbl INV RCL 0 + RCL 1 =           R_X ← a+b  ←──┐
020   ÷ 2 = STO 2 B STO 5               c ← ½(a+b), r←f(c)
028   * ( RCL 1 - RCL 0 ) * 2 ÷         R_X ← 2(b-a)r
039   ( RCL 3 - RCL 4 ) ÷               R_X ← 2(b-a)r/(p-q)
047   ( ( 1 + RCL 5 *                   R_X ← 1 + r
054   ( CE * 2 - RCL 3 - RCL 4 ) *                ·(2r-p-q)
066   8 ÷ ( RCL 3 - RCL 4 ) x²  )       R_X←1+8r(2r-p-q)/(p-q)²
077   √x + 1 ) + RCL 2 = STO 7          x←c+ 2(b-a)r/(p-q)
                                                1+√1+8r...
087   Prt B STO 6                       Anzeige x, s ← f(x)
091   CP                                R_T ← 0 ──┘
```

```
092   RCL 3 * RCL 5 = INV x≥t x↔t
101   RCL 5 *  RCL 6 = INV x≥t CE
110   Lbl lnx RCL 7 STO 0
116   RCL 6 STO 3 GTO INV
122   Lbl CE RCL 2 STO 0 RCL 5 STO 3
132   Lbl CLR RCL 7 STO 1
138   RCL 6 STO 4 GTO INV
144   Lbl x↔t RCL 3 * RCL 6 =
152   INV x≥t CLR
155   RCL 2 STO 1
159   RCL 5 STO 4 GTO lnx
```



Unterprogramm f(x)

```
165   Lbl B
```

Anleitung zur Verwendung der Programme

1. Eingabe des Hauptprogramms.
2. Eingabe des Unterprogramms B für die Funktion f (wie bei der Methode der Intervallhalbierung und der Regula falsi).
3. Eingabe der Punkte a und b
   bei HP-67/97: a STO 0, b STO 1;
   bei TI-58/59: a STO 00, b STO 01.
4. Berechnung und Anzeige der Näherungen x_0, x_1, x_2,... für die Nullstelle z: A.

Mögliche Wiederholungen:
ab Schritt 3 für andere Ausgangspunkte a und b;
ab Schritt 2 für eine andere Funktion f.

Beispiel

Für f(x) = sin x ist das Unterprogramm B dasselbe wie in dem Beispiel zur Methode der Intervallhalbierung. Mit a = 4, b = 2 ergaben sich die folgenden Näherungen.

```
 3.16722397
3.141627659
3.141592663
3.141592654
3.141592654
3.141592654
```

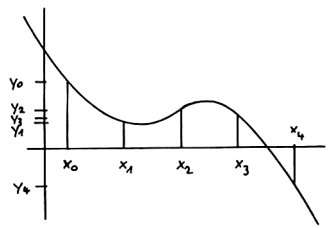
4. Interpolationspolynome und Differenzenschema

Die numerische Berechnung der Funktionswerte von Interpolationspolynomen ist auf verschiedene Arten möglich. Wir verwenden hier das Schema der iterativen linearen Interpolation. Ähnlich günstig ist die Newtonsche Darstellung von Interpolationspolynomen, für die wir das Differenzenschema berechnen. Wesentliches Hilfsmittel ist in beiden Fällen die Berechnung der Spalten eines Dreieckschemas, das auch in Kapitel 5 bei der Romberg-Integration Verwendung findet.

Es seien $m + 1$ paarweise verschiedene Punkte $x_0,\ldots,x_m$ gegeben und Zahlen $y_0,\ldots,y_m$. Dann gibt es ein eindeutig bestimmtes Interpolationspolynom p höchstens m-ten Grades,

$$p(x) = a_0+a_1x+\ldots+a_mx^m,$$

das an den Stützstellen x_j die Funktionswerte y_j annimmt,



$$p(x_j) = y_j,\ j = 0,\ldots,m.$$

Es ist gegeben durch die Lagrange-Darstellung

$$p(x) = \sum_{k=0}^{m} y_k \prod_{\substack{l=0 \\ l\neq k}}^{m} \frac{x-x_l}{x_k-x_l}\ .$$

Bei der numerischen Berechnung erweist sich jedoch diese Darstellung als nicht zweckmäßig; insbesondere ist es bei Hinzunahme einer weiteren Stützstelle x_{m+1} nicht möglich, bereits berechnete Ergebnisse weiter zu verwenden.

Wir bezeichnen das Interpolationspolynom zu den Punkten $x_j, x_{j+1},\ldots,x_{j+l}$ mit $p_{j,\ldots,j+l}$; dann ist für $l = 0$ das Polynom p_j konstant,

$$p_j(x) = y_j,\ j = 0,\ldots,m.$$

Für $l = 1,\ldots,m$ genügen die Interpolationspolynome $p_{j,\ldots,j+l}$ der Rekursionsformel (s. [9] 3.1.)

$$p_{j,\ldots,j+l}(x) = \frac{1}{x_j - x_{j+l}}\{(x - x_{j+l})p_{j,\ldots,j+l-1}(x) - (x - x_j)p_{j+1,\ldots,j+l}(x)\},\ j=0,\ldots,m-l.$$

Daher können Funktionswerte des Interpolationspolynoms $p_{0,\ldots,m}$ durch fortlaufende Berechnung der Spalten des folgenden Schemas der iterativen linearen Interpolation erhalten werden.

x_0	x_1	x_2	$\cdots$	x_{m-1}	x_m
y_0	y_1	y_2	$\cdots$	y_{m-1}	y_m
	$p_{0,1}(x)$	$p_{1,2}(x)$	$\cdots$	$p_{m-2,m-1}(x)$	$p_{m-1,m}(x)$
		$p_{0,1,2}(x)$	$\cdots$	$p_{m-3,m-2,m-1}(x)$	$p_{m-2,m-1,m}(x)$
				$\vdots$	$\vdots$
				$p_{0,\ldots,m-1}(x)$	$p_{1,\ldots,m}(x)$
					$p_{0,\ldots,m}(x)$

Hieran zeigt sich der Vorteil dieser rekursiven Darstellung der Interpolationspolynome; ist nämlich für einen Punkt x der Funktionswert des Interpolationspolynoms $p_{0,\ldots,m}(x)$ nach diesem Schema ermittelt und möchte man die weitere Stützstelle x_{m+1} berücksichtigen, so erhält man $p_{0,\ldots,m+1}(x)$ durch Ergänzung des Schemas um eine weitere Spalte.

Wir betrachten nun die Newtonsche Darstellung des Interpolationspolynoms $p = p_{0,\ldots,m}$,

$$p(x) = y[x_0] + (x-x_0)y[x_0,x_1] + \ldots + (x-x_0)\cdot\ \cdots\ \cdot(x-x_{m-1})y[x_0,\ldots,x_m],$$

mit den dividierten Differenzen

$$y[x_j] = y_j, \quad j = 0,\dots,m,$$

$$y[x_j,\dots,x_{j+l}] = \frac{1}{x_j - x_{j+l}} \left(y[x_j,\dots,x_{j+l-1}] - y[x_{j+1},\dots,x_{j+l}]\right), \quad j = 0,\dots,m-l,$$

für $l = 1,\dots,m$. Man ordnet die dividierten Differenzen $y_{j,\dots,j+l} = y[x_j,\dots,x_{j+l}]$ in dem sogenannten Differenzenschema an.

$$\begin{array}{cccccc}
x_0 & x_1 & x_2 & \cdots & x_{m-1} & x_m \\
y_0 & y_1 & y_2 & \cdots & y_{m-1} & y_m \\
 & y_{0,1} & y_{1,2} & \cdots & y_{m-2,m-1} & y_{m-1,m} \\
 & & y_{0,1,2} & \cdots & y_{m-3,m-2,m-1} & y_{m-2,m-1,m} \\
 & & & & \vdots & \vdots \\
 & & & & y_{0,\dots,m-1} & y_{1,\dots,m} \\
 & & & & & y_{0,\dots,m}
\end{array}$$

Für Interpolationspolynome höchstens m-ten Grades und für dividierte Differenzen m-ter Ordnung haben die beiden entstehenden Zahlenschemata eine gemeinsame Struktur. Es werden jeweils die Spalten eines (m+1)-zeiligen Schemas

$$\begin{array}{ccccccccc}
u_{0,0} & u_{0,1} & u_{0,2} & \cdots & u_{0,m-1} & u_{0,m} & u_{0,m+1} & \cdots \\
 & u_{1,1} & u_{1,2} & \cdots & u_{1,m-1} & u_{1,m} & u_{1,m+1} & \cdots \\
 & & u_{2,2} & \cdots & u_{2,m-1} & u_{2,m} & u_{2,m+1} & \cdots \\
 & & & & \vdots & \vdots & \vdots & \\
 & & & & u_{m-1,m-1} & u_{m-1,m} & u_{m-1,m+1} & \cdots \\
 & & & & & u_{m,m} & u_{m,m+1} & \cdots
\end{array}$$

ermittelt; auf Grund der obigen Rekursionsformeln berechnet sich für $i>0$ das Element u_{ik} nur aus den beiden oberhalb und links oberhalb stehenden Elementen $u_{i-1,k}$ und $u_{i-1,k-1}$. Setzen wir zur Vereinfachung und Speicherplatzersparnis voraus, daß die Stützstellen äquidistant sind,

$$x_k = x_0 + kh, \quad k = 0,1,2,\dots,$$

so gilt für die Interpolationspolynome

$$u_{ik} = p_{k-i,\ldots,k}(x)$$

$$= u_{i-1,k} + \frac{s-k}{i}\,(u_{i-1,k} - u_{i-1,k-1}),\quad s = \frac{1}{h}\,(x - x_0),$$

und im Fall der dividierten Differenzen ist

$$u_{ik} = y_{k-i,\ldots,k} = \frac{1}{ih}\,(u_{i-1,k} - u_{i-1,k-1}).$$

Die letzte Zeile des so berechneten Schemas enthält dann die Funktionswerte $u_{m,k+m} = p_{k,\ldots,k+m}(x)$ des Interpolationspolynoms höchstens m-ten Grades zu den Punkten $x_k, x_{k+1}, \ldots, x_{k+m}$ bzw. die dividierten Differenzen m-ter Ordnung $u_{m,k+m} = y_{k,\ldots,k+m}$ zu diesen Punkten für $k = 0,1,2,\ldots$.

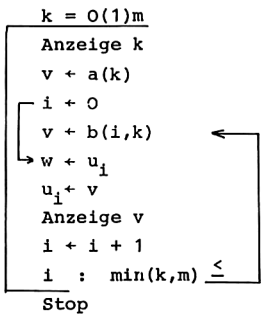
In dem folgenden Algorithmus zur Berechnung des Zahlenschemas sind in den Speicherplätzen $u_0,\ldots,u_m$ jeweils nur die entsprechenden Elemente der k-ten Spalte des Schemas gespeichert. Ist u_{ik} neu berechnet, so wäre der Speicherplatz u_i der geeignete Ort zur Aufnahme dieses Wertes; jedoch wird der dort gespeicherte Wert $u_{i,k-1}$ aus der vorherigen Spalte noch für die anschließende Berechnung von $u_{i+1,k}$ benötigt. Daher wird in dem folgenden Algorithmus u_{ik} berechnet und zunächst nach v gespeichert; sodann wird der Wert $u_{i,k-1}$ vom Speicherplatz u_i nach w gebracht, und dann erst wird u_{ik} aus v nach u_i übertragen. Auf diese Weise stehen dann u_{ik} in v und $u_{i,k-1}$ in w zur Verfügung, wenn im anschließenden Schritt $u_{i+1,k}$ berechnet werden soll.

Algorithmus

Eingabe: Unterprogramm a(k) zur Berechnung von u_{0k};
Unterprogramm b(i,k) zur Berechnung von u_{ik} aus $v = u_{i-1,k}$ und $w = u_{i-1,k-1}$;
m

Anzeige: für k = 0,1,...,m Nummer und Inhalt der k-ten Spalte des Schemas:
k, u_{ik} für i = 0,..., min(k,m).

i	k	m	u_0	u_1	...	u_m	v	w



In den Taschenrechnerprogrammen zu diesem Algorithmus ist die Schleife k = 0(1)m durch die zugehörigen einzelnen Anweisungen ersetzt. Daher kann nach Beendigung des Algorithmus eine weitere (m+1)-zeilige Spalte des Schemas erhalten werden, indem die Berechnung bei "Anzeige k" weitergestartet wird; dieser Schritt kann mehrfach wiederholt werden.

Sollen die Daten u_{0k} für die oberste Zeile des Zahlenschemas eingegeben werden, also nicht durch ein Unterprogramm a(k) berechnet werden, so ersetzt man die Wertzuweisung v←a(k) durch die Anweisungen Stop, Eingabe v = u_{0k}.

Ist man nur an der Anzeige des letzten berechneten Wertes u_m interessiert, so entfernt man "Anzeige k" und "Anzeige v" aus dem Wiederholungsbereich der obigen Programmschleife und fügt am Schluß des Algorithmus vor Stop ein: Anzeige u_m.

Sind $y_k = f(x_k)$ die Werte einer Funktion f an äquidistanten Stützstellen $x_k = x_0 + kh$, zu denen Interpolationspolynome oder dividierte Differenzen berechnet werden sollen, so lautet das Unterprogramm a(k) zur Berechnung von $u_{0k} = y_k$ dann folgendermaßen.

h	k	x_0

$a(k) \leftarrow f(x_0 + kh)$
Rücksprung

Das Unterprogramm b(i,k) zur Berechnung von u_{ik} besteht für die lineare iterative Interpolation mit der Abkürzung $s = \frac{1}{h}(x - x_0)$ aus der Wertzuweisung

i	k	s	v	w

$b(i,k) \leftarrow v + \frac{s-k}{i}(v - w)$
Rücksprung.

Für das Differenzenschema gilt

h	i	v	w

$b(i,k) \leftarrow \frac{v-w}{ih}$
Rücksprung.

In den folgenden Taschenrechnerprogrammen speichern wir die Größen $u_0, u_1, \ldots, u_m$ vom Ende des Datenspeichers her. Auf diese Weise ist es möglich, daß für die Unterprogramme b(i,k) und a(k) weitere Datenspeicherplätze benutzt werden können ohne daß durch das Programm bereits eine Begrenzung für m festgelegt wird. In dem Programm für die Rechner HP-67/97 mit den hier verwendeten Unterprogrammen für a(k) und b(i,k) bedeutet dies, daß $m \leq 16$ möglich ist; die Größen $u_0, \ldots, u_m$ sind in den Datenregistern $R_E, R_D, \ldots, R_A, R_{s9}, \ldots$ gespeichert, und sie werden aufgerufen durch indirekte Adressierung mit Adr u_i = 24-i. In dem Programm für die Rechner TI-58/59 ist Adr u_i = ω - i mit der Nummer ω des letzten Datenregisters R_ω; die Zahl ω hängt ab von dem verwendeten Gerät und von der jeweiligen Aufteilung des Speichers zwischen Datenregistern und Programmspeicher.

Programm für HP-67/97

Belegung des Datenspeichers für das Hauptprogramm

R_0	R_1	R_2	R_3	R_4		R_{24-m}	...	R_{23}	R_{24}	R_I
i	k	m	v	w		u_m	...	u_1	u_0	Adr u_i =24-i

$m \leq 16$

Hauptprogramm

```
001   0 STO 1                          k←0
003   LBL 0 PRTSPC RCL 1 PRTx          Anzeige k
007   A STO 3                          v←a(k)
009   0 STO 0 24 STO I GTO 2           i←0, R_I←Adr u_0
015   LBL 1 B STO 3                    v←b(i,k)
018   LBL 2 RCL (i) STO 4              w←u_i
021   RCL 3 STO (i) PRTx               u_i←v, Anzeige v
024   1 STO + 0 DSZ I                  i←i + 1, R_I←Adr u_i
027   RCL 2 RCL 1 x>y? x↔y             R_X←min(k,m)
031   RCL 0 x≤y? GTO 1                 wenn i≤min(k,m)
034   1 STO + 1                        k ← k + 1
036   RCL 2 RCL 1 x<y? GTO 0           wenn k≤m
040   R/S                              Stop
```

Zusätzliche Belegung des Datenspeichers für die Unterprogramme a(k) und b(i,k)

R_5	R_6	R_7
h	s	x_0

Unterprogramm b(i,k)

```
041   LBL B RCL 3 RCL 4 - RCL 0 ÷      R_X ← (v-w)/i
047   F? 0 GTO 3                       wenn Flag 0 gesetzt
049   RCL 6 RCL 1 - * RCL 3 +          R_X ← v + ((s-k)/i) (v-w)
055   RTN                              Rücksprung
056   LBL 3 RCL 5 ÷                    R_X ← (v-w)/(ih)
059   RTN                              Rücksprung
```

Unterprogramm a(k)

```
060   LBL A
```

Programm für TI-58/59

Belegung des Datenspeichers für das Hauptprogramm

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}		$R_{\omega-m}$	...	$R_{\omega-1}$	R_{ω}
i	k	m	v	w	ω	Adr u_i =ω-i		u_m	...	u_1	u_0

Hauptprogramm

```
000   Lbl E Op 16 INV Int * 100 =      R_X←ω
011   STO 05 0 STO 1                   R_05←ω, k←0
016   Lbl INV Adv RCL 1 Prt            Anzeige k
022   A STO 03                         v←a(k)
025   0 STO 0 RCL 5 STO 6 GTO CE       i←0, R_06←Adr u_0
034   Lbl lnx B STO 3                  v←b(i,k)
039   Lbl CE RCL Ind 6 STO 4           w←u_i
045   RCL 3 STO Ind 6 Prt              u_i←v, Anzeige v
050   Op 20 Op 36                      i ← i + 1, R_06←Adr u_i
054   RCL 2 x↔t RCL 1 x≥t CLR x↔t      R_T←min(k,m)
062   Lbl CLR RCL 0 x↔t x≥t lnx        wenn min(k,m)≥i
069   Op 21 RCL 1 x↔t RCL 2 x≥t INV    k ← k + 1, wenn m≥k
078   R/S                              Stop
```

Zusätzliche Belegung des Datenspeichers für die Unterprogramme a(k) und b(i,k)

R_{07}	R_{08}	R_{09}
h	s	x_0

Unterprogramm b(i,k)

```
079   Lbl B ( ( RCL 3 - RCL 4 )        R_X ← v - w
089   ÷ RCL 0                          ÷ i
092   Ifflg 0 x↔t                      wenn Flag 0 gesetzt
095   *( RCL 8 - RCL 1 ) + RCL 3 )     R_X ← v + (s-k)/i (v - w)
107   INV SBR                          Rücksprung
108   Lbl x↔t ÷ RCL 7 )                R_X ← (v-w)/(ih)
114   INV SBR                          Rücksprung
```

Unterprogramm a(k)

```
115   Lbl A
```

Anleitung zur Verwendung der Programme

1. Eingabe des Hauptprogramms und des Unterprogramms b(i,k).
2. Eingabe des Unterprogramms A für a(k) zur Berechnung der Elemente u_{0k} in der obersten Zeile des Schemas.
   In den anschließenden Anwendungen ist $u_{0k} = y_k = f(x_k) = f(x_0 + kh)$. Im Fall der HP-Rechner wird aus x_0 in R_7, k in R_1 und h in R_5 der Funktionswert $f(x_0 + kh)$ berechnet; bei den TI-Rechnern stehen x_0 in R_{09}, k in R_{01} und h in R_{07} zur Verfügung.
   Der berechnete Funktionswert wird bei Rücksprung in das rufende Programm im Anzeigeregister übergeben.
3. Eingabe der Schrittweite h der äquidistanten Punkte $x_0, x_1, x_2, \ldots$
   für HP-67/97: h STO 5, für TI-58/59: h STO 07.
4. Eingabe des Anfangspunktes x_0
   bei den Rechnern HP-67/97: x_0 STO 7,
   bei den Rechnern TI-58/59: x_0 STO 09.
5. Soll das Schema der iterativen linearen Interpolation an der Stelle x berechnet werden, so muß $s = \frac{1}{h}(x - x_0)$ eingegeben werden
   für HP-67/97: s STO 6,
   für TI-58/59: s STO 08.
6. Ist Flag 0 gelöscht, so wird das Schema der iterativen linearen Interpolation berechnet; ist Flag 0 gesetzt, so wird das Differenzenschema berechnet.
7. Es werden die Zeilen 0 bis m des Schemas berechnet. Eingabe m: m STO 2 bzw. m STO 02.
8. Start der Rechnung und für k = 0,1,...,m Anzeige von Nummer und Inhalt der k-ten Spalte
   bei den Rechnern HP-67/97 durch RTN R/S,
   bei den Rechnern TI-58/59 durch E.
9. Soll nach Beendigung des Algorithmus eine weitere (m+1)-zeilige Spalte des Schemas berechnet werden, so wird die Rechnung und Anzeige fortgesetzt mit GTO 0 R/S bzw. GTO INV R/S.

Mögliche Wiederholungen:

Schritt 9 für eine weitere Spalte;
ab Schritt 7 für eine andere Zeilenzahl m;
zur Veränderung von h, x_0, x oder der Wahl Interpolationspolynom/Differenzenschema führt man Schritt 3, 4, 5 oder 6 aus.
Zur Verwendung einer anderen obersten Zeile bzw. einer anderen Funktion f geht man gemäß Schritt 2 vor.

Das Hauptprogramm für die Rechner TI-58/59 beginnt mit Lbl E und wird daher durch Betätigung der Programmtaste E gestartet. Im Gegensatz dazu wurde das HP-67/97-Programm ohne Anfangsmarke angegeben, und es wurde dort die Ausführung mit RTN R/S begonnen. Die entsprechende Struktur (keine Anfangsmarke, Start mit RST R/S) erweist sich für die Rechner TI-58/59 hier als unzweckmäßig, weil durch RST nicht nur der Programmzeiger an den Anfang gerückt wird, sondern gleichzeitig auch alle Flags gelöscht werden.

Insbesondere bei der Berechnung der Funktionswerte von Interpolationspolynomen wird man häufig nur an dem zuletzt berechneten Wert u_m interessiert sein. Wie oben bei Aufstellung des Algorithmus bereits erwähnt, ersetzt man zu diesem Zweck in dem Programm für HP-67/97 die Schlußanweisung 040 des Hauptprogramms durch die Anweisungen zur Anzeige von u_m,

040	24 RCL 2 - STO I	$R_I \leftarrow$Adr u_m
045	RCL (i) PRT x	Anzeige u_m
047	R/S	Stop

und entfernt danach die Anzeigeanweisungen 023, 006, 005 und 004. Entsprechend ersetzt man in dem Programm für TI-58/59 die Schlußanweisung 078 des Hauptprogramms mit GTO 078 LRN Ins ... Ins (elfmal) durch

078	RCL 5 - RCL 2 = STO 6	$R_{06} \leftarrow$ Adr u_m
086	RCL Ind 6 Prt	Anzeige u_m
089	R/S	Stop

und entfernt anschließend die Anzeigeanweisungen 049 und 018 bis 021.

Beispiele

(1) Für $f(x) = x^5$ wurde das Differenzenschema bis zu den dividierten Differenzen fünfter Ordnung berechnet zu der Stelle $x_0 = 0$ mit der Schrittweite $h = 1$.

Das Unterprogramm a(k) lautet
für HP-67/97

```
060   LBL A RCL 1 RCL 5 * RCL 7 +     R_X ← x_0 + kh
066   5 y^x                           R_X ← (x_0 + kh)^5
068   RTN                             Rücksprung,
```

für TI-58/59

```
115   Lbl A ( ( RCL 1 * RCL 7 +       R_X ← kh
125   RCL 9 ) y^x 5 )                 R_X ← (x_0 + kh)^5
131   INV SBR                         Rücksprung.
```

Mit $h = 1$, $x_0 = 0$, Flag 0 gesetzt, $m = 5$ sowie Start der Rechnung gemäß Schritt 8 der Anleitung zur Verwendung des Programms wurden die Spalten 0 bis 5 des Differenzenschemas berechnet. Anschließend wurden die Spalten 6 und 7 nach Schritt 9 der Anleitung erhalten.

0	1	2	3	4	5	6	7
0	1	32	243	1024	3125	7776	16807
	1	31	211	781	2101	4651	9031
		15	90	285	660	1275	2190
			25	65	125	205	305
				10	15	20	25
					1	1	1

(2) Zu Beispiel (1) wurde mit Hilfe der iterativen linearen Interpolation der Wert des Interpolationspolynoms $p_{0,\ldots,5}$ an der Stelle $\frac{1}{2}$ ermittelt.

Mit demselben Unterprogramm a(k) und mit $h = 1$, $x_0 = 0$, $s = x = 0.5$, Flag 0 gelöscht und $m = 5$ wurde die Berechnung gemäß Schritt 8 der Programmanleitung ausgeführt. Als letzter Wert wurde $p_{0,\ldots,5}\,(\frac{1}{2}) = 0.03125 = (\frac{1}{2})^5$ angezeigt. $f(x) = x^5$

ist selbst ein Polynom fünften Grades und stimmt daher mit seinem Interpolationspolynom $p_{0,\ldots,5}$ überein, $f = p_{0,\ldots,5}$.

(3) Für die Funktion $f(x) = \cosh x = \frac{1}{2}(e^x + e^{-x})$ wurden die Funktionswerte des Interpolationspolynoms $p_{0123}(x)$ zu der Stelle $x_0 = 0$ und der Schrittweite $h = 0.1$ berechnet für $x = -0.1(0.02)0.5$ mit Hilfe der iterativen Interpolation.

Das Unterprogramm a(k) lautet
für HP-67/96

060	LBL A RCL 1 RCL 5 * RCL 7 +	$R_X \leftarrow x_0 + kh$
066	e^x ENTER 1/x + 2 ÷	$R_X \leftarrow \cosh(x_0 + kh)$
072	RTN	Rücksprung,

für TI-58/59

115	Lbl A (((RCL 1 * RCL 7 +	$R_X \leftarrow kh$
126	RCL 9) INV lnx + 1/x) ÷ 2)	$R_X \leftarrow \cosh(x_0 + kh)$
137	INV SBR	Rücksprung.

Mit $h = 0.1$, $x_0 = 0$, Flag 0 gelöscht, $m = 3$ wurde jeweils zur Berechnung von $p_{0123}(x)$ der Wert $s = \frac{1}{h}(x - x_0) = 10x$ gemäß Punkt 5 der Anleitung eingegeben und die Berechnung entsprechend Punkt 8 gestartet. Hierbei wurde das Hauptprogramm in der oben erwähnten Modifikation verwendet, so daß nur der zuletzt berechnete Wert $p_{0123}(x)$ angezeigt wurde.

In diesem Zusammenhang erweist es sich als zweckmäßig, bei der Tabellierung von Funktionswerten $p_{0,\ldots,m}(x)$ den Anfang des Hauptprogramms so zu ändern, daß vor dem Start die Stelle x in das Anzeigeregister eingegeben werden kann; das Hauptprogramm berechnet damit zuerst $s = \frac{1}{h}(x - x_0)$ und speichert diesen Wert nach R_6 bzw. R_{08}. In der Anleitung zur Verwendung des Programms entfällt dann Punkt 5, und Punkt 8 lautet für die Rechner HP-67/97: Eingabe x RTN R/S, für die Rechner TI-58/59: Eingabe x E.

x	cosh x	$p_{0123}(x)$
-0.10	1.005004168	1.004503502
-0.08	1.003201707	1.002137424
-0.06	1.001800541	1.001152629
-0.04	1.000800107	1.000768532
-0.02	1.000200067	1.000152911
0.00	1.000000000	1.000000000
0.02	1.000200067	1.000203396
0.04	1.000800107	1.000804305
0.06	1.001800540	1.001803533
0.08	1.003201707	1.003203484
0.10	1.005004166	1.005004166
0.12	1.007206644	1.007207156
0.14	1.009810017	1.009813751
0.16	1.012827330	1.012835653
0.18	1.016243788	1.016242328
0.20	1.020066756	1.020066756
0.22	1.024297765	1.024299556
0.24	1.028936566	1.028941917
0.26	1.033990837	1.033955063
0.28	1.039456777	1.039468132
0.30	1.045338514	1.045338514
0.32	1.051636401	1.051631232
0.34	1.058358557	1.058339555
0.36	1.065502871	1.065464683
0.38	1.073072999	1.073007826
0.40	1.081072372	1.080976192
0.42	1.089504188	1.089352934
0.44	1.098371820	1.098157409
0.46	1.107678816	1.107384673
0.48	1.117428897	1.117035980
0.50	1.127625966	1.127112540

Der Vergleich der Werte des Interpolationspolynoms mit den Funktionswerten von cosh x zeigt, daß an den Stützstellen 0, 0.1, 0.2, 0.3 Übereinstimmung herrscht; dort interpoliert das Polynom. Die Fehlerfunktion cosh x - $p_{0123}(x)$ wechselt an den Stützstellen das Vorzeichen; zwischen den Stützstellen ist der Betrag des Fehlers klein, außerhalb - bei Extrapolation - nimmt er rasch zu.

5. Numerische Integration

Den Wert bestimmter Integrale ermittelt man in der Praxis mit Hilfe von Integraltafeln. Jedoch kann die Auswertung der darin auftretenden höheren transzendenten Funktionen selbst sehr kompliziert sein, oder eine geschlossene Darstellung eines gewissen Integrals kann nicht gefunden werden. Daher ist es häufig erforderlich, mit numerischen Methoden bestimmte Integrale näherungsweise zu berechnen.

5.1. Trapezformeln und Simpsonsche Formel

Einfache Methoden zur Approximation bestimmter Integrale

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

beruhen darauf, daß das Integrationsintervall [a,b] durch äquidistante Punkte unterteilt wird:

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

mit einer natürlichen Zahl N. Bei der Sehnentrapezformel dient

$$S = h(\tfrac{1}{2}f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{N-1}) + \tfrac{1}{2}f(x_N))$$

als Näherung für I; bei der Tangententrapezformel wird

$$T = h(f(\tfrac{1}{2}(x_0 + x_1)) + f(\tfrac{1}{2}(x_1 + x_2)) + \ldots + f(\tfrac{1}{2}(x_{N-2} + x_{N-1})) + f(\tfrac{1}{2}(x_{N-1} + x_N)))$$

verwandt.

Algorithmus	Algorithmus
Sehnentrapezformel	Tangententrapezformel
Eingabe: Unterprogramm für f; a, b, N.	
Anzeige: S	Anzeige: T

a	b	h	j	N	S

$h \leftarrow \frac{b-a}{N}$
$S \leftarrow f(a)$
$S \leftarrow S + f(b)$
$S \leftarrow \frac{1}{2}S$
j = 1(1)N-1
S ← S + f(a + jh)
$S \leftarrow hS$
Anzeige S
Stop

a	b	h	j	N	T

$h \leftarrow \frac{b-a}{N}$
$T \leftarrow 0$
$a \leftarrow a - \frac{h}{2}$
j = 1(1)N-1
T ← T + f(a + jh)
$T \leftarrow hT$
Anzeige T
Stop

Die Trapezformeln entstehen dadurch, daß für k = 0,1,...,N-1 in jedem Intervall $[x_k, x_{k+1}]$ an Stelle von f eine lineare Näherung für f integriert wird. Verwendet man in Teilintervallen der Länge 2h Interpolationspolynome höchstens zweiten Grades zur Approximation von f, so entsteht die Simpsonsche Formel

$$Q = \frac{h}{3}(f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+2f(x_4)+\ldots +2f(x_{N-2})+4f(x_{N-1})+f(x_N));$$

hierbei muß N eine gerade Zahl sein.

Während für zweimal stetig differenzierbare Funktionen f auf [a,b] für die Trapezformeln die Fehlerabschätzungen

$$|I - S| \leq (a - b)\frac{h^2}{12} \max_{a \leq x \leq b} |f''(x)|, \quad |I - T| \leq (b - a)\frac{h^2}{24} \max_{a \leq x \leq b} |f''(x)|$$

gelten, besteht bei der Simpsonschen Formel für Funktionen f, die viermal stetig differenzierbar auf [a,b] sind, die Abschätzung

$$|I - Q| \leq (b - a)\frac{h^4}{180} \max_{a \leq x \leq b} |f^{(4)}(x)|$$

(s. [9] 4.2.1.). Verkleinerung der Schrittweite $h = \frac{b-a}{N}$ bewirkt hier eine wesentlich stärkere Verbesserung der Approximation.

Algorithmus **Simpsonsche Formel**

Voraussetzung: $N \geq 2$ sei eine gerade Zahl

Eingabe: Unterprogramm für f;
a, b, N

Anzeige: Q

a	b	h	j	N	Q	v

$h \leftarrow \frac{b-a}{N}$

$Q \leftarrow f(a)$

$Q \leftarrow Q + f(b)$

$v \leftarrow 4$

$j = 1(1)N-1$

$Q \leftarrow Q + vf(a + jh)$

$v \leftarrow 6 - v$

$Q \leftarrow \frac{h}{3} Q$

Anzeige Q

Stop

Einer Anregung aus [2]2.4. folgend wird durch die Wertzuweisung v←4 vor der Schleife j = 1(1)N-1 sowie durch v←6-v im Wiederholungsbereich dieser Schleife erreicht, daß in der Summe für Q die Faktoren 4 und 2 abwechseln.

In den anschließenden Taschenrechnerprogrammen für die Simpsonsche Formel durchlaufen wir die Schleife j = 1(1)N-1 wieder rückwärts in der Form j = N-1(-1)1, so daß wir die Dsz-Anweisung verwenden können.

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	R_3	R_4	R_5		R_I
a	b	h	N	Q	v		j

Hauptprogramm

001	RCL 1 RCL 0 - RCL 3 ÷ STO 2	$h \leftarrow \frac{b-a}{N}$
007	RCL 0 C STO 4	$Q \leftarrow f(a)$
010	RCL 1 C STO + 4	$Q \leftarrow Q + f(b)$
013	4 STO 5 RCL 3 STO I DSZ I	$v \leftarrow 4$, $j \leftarrow N - 1$
018	LBL 0 I RCL 2 * RCL 0 +	$R_X \leftarrow a + jh$
024	C RCL 5 * STO + 4	$Q \leftarrow Q + vf(a + jh)$
028	6 RCL 5 - STO 5	$v \leftarrow 6 - v$
032	DSZ I GTO 0	$j \leftarrow j - 1$, wenn $j>0$
034	RCL 2 3 ÷ STO * 4	$Q \leftarrow \frac{h}{3}Q$
038	RCL 4 PRTx R/S	Anzeige Q, Stop

Unterprogramm f(x)

041 LBL C

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}
a	b	h	j	N	Q	v

Hauptprogramm

000	RCL 1 - RCL 0 =	$R_X \leftarrow b - a$
006	÷ RCL 4 = STO 2	$h \leftarrow \frac{b-a}{N}$
012	RCL 0 C STO 5	$Q \leftarrow f(a)$
017	RCL 1 C SUM 05 4 STO 6	$Q \leftarrow Q + f(b)$, $v \leftarrow 4$
025	RCL 4 STO 3 Op 33	$j \leftarrow N - 1$
031	Lbl INV RCL 3 * RCL 2 +	$R_X \leftarrow jh$
039	RCL 0 = C * RCL 6 = SUM 05	$Q \leftarrow Q + vf(a + jh)$
049	6 - RCL 6 = STO 6	$v \leftarrow 6 - v$
056	Dsz 3 INV	$j \leftarrow j - 1$, wenn $j>0$
059	RCL 2 ÷ 3 = Prd 5	$Q \leftarrow \frac{h}{3}Q$
066	RCL 5 Prt R/S	Anzeige Q, Stop

Unterprogramm f(x)

070 Lbl C

Anleitung zur Verwendung der Programme

1. Eingabe des Hauptprogramms.
2. Eingabe des Unterprogramms C zur Berechnung von f(x).
   x steht bei Aufruf des Unterprogramms im Anzeigeregister;
   bei Rücksprung in das rufende Programm wird f(x) im Anzeigeregister übergeben.
3. Eingabe der Integrationsgrenzen a und b
   bei HP-67/97: a STO 0 b STO 1,
   bei TI-58/59: a STO 00 b STO 01.
4. Eingabe der Zahl N
   bei HP-67/97: N STO 3,
   bei TI-58/59: N STO 04.
5. Berechnung und Anzeige von Q: RTN R/S bzw. RST R/S.

Mögliche Wiederholungen:

ab Schritt 4 für einen anderen Wert von N,
ab Schritt 3 für ein anderes Integrationsintervall,
ab Schritt 2 für eine andere Funktion.

Beispiele

Es wurden Näherungswerte ermittelt für die Integrale

$$I_1 = \int_0^1 \frac{4}{x^2+1}\,dx = 4 \arctan 1 = \pi,$$

$$I_2 = \int_0^2 e^{-x^2}dx = \frac{\sqrt{\pi}}{2}\,\mathrm{erf}(2).$$

Im ersten Fall lautet das Unterprogramm C
für HP-67/97

041	LBL C x^2 1 + 1/x 4 *	$R_X \leftarrow 4/(x^2+1)$
048	RTN	Rücksprung,

für TI-58/59

070	Lbl C ((CE x^2 + 1)	$R_X \leftarrow x^2+1$
079	1/x * 4)	$R_X \leftarrow 4/(x^2+1)$
083	INV SBR	Rücksprung.

Für das zweite Integral lautet das Unterprogramm C
für HP-67/97

041	LBL C x^2 CHS e^x	$R_X \leftarrow e^{-x^2}$
045	RTN	Rücksprung,

für TI-58/59

070	Lbl C x^2 +/- INV lnx	$R_X \leftarrow e^{-x^2}$
076	INV SBR	Rücksprung.

Für N = 2, 4, 8, 16, 32 lieferte die Simpsonsche Formel die folgenden Näherungswerte.

N	I_1	I_2
2.	3.133333333	.8299444679
4.	3.141568627	.8818124253
8.	3.141592502	.8820655104
16.	3.141592651	.8820803966
32.	3.141592654	.8820813286

5.2. Romberg-Integrartion

Ausgehend von der Sehnentrapezformel werden durch fortgesetzte Halbierung der Schrittweite und geeignete Linearkombination die Rombergschen Näherungsformeln erhalten, die zu einer wesentlichen Verbesserung der Konvergenzordnung führen. Grundlage hierfür ist die folgende genauere Kenntnis des Fehlers der Sehnentrapezformel:
Für 2n-mal stetig differenzierbare Funktionen f auf [a,b] gilt

$$I - S = c_1 h^2 + c_2 h^4 + \ldots + c_{n-1} h^{2n-2} + E$$

mit $E = c_n h^{2n} f^{(2n)}(y)$, wobei y ein Punkt zwischen a und b ist; die Größen $c_1, \ldots, c_n$ hängen von f, a und b ab, aber nicht von der Schrittweite h (s. [9] 4.2.1.).

Wir wollen die Sehnentrapezformel S für verschiedene Schrittweiten verwenden, indem wir die Anzahl N der Teilintervalle des Integrationsintervalls [a,b] jeweils verdoppeln. Für $k = 0,1,2,\ldots$ sei $N_k = 2^k$, die Schrittweite $h_k = (b-a)/N_k$, und die Teilpunkte seien $x_i^{(k)} = a + ih_k$, $i = 0,1,\ldots,N_k$. Die zugehörigen Sehnentrapezformeln lauten dann

$$S_0 = \tfrac{1}{2}h_0(f(a) + f(b)),$$

$$S_k = h_k(\tfrac{1}{2}f(x_0^{(k)}) + f(x_1^{(k)}) + f(x_2^{(k)}) + \ldots + f(x_{N_k-1}^{(k)}) + \tfrac{1}{2}f(x_{N_k}^{(k)})), k = 1,2,\ldots .$$

Mit der obigen Darstellung des Fehlers der Sehnentrapezformel erhalten wir

$$\int_a^b f(x)dx = S_k + c_1h_k^2 + c_2h_k^4 + \ldots ,$$

$$\int_a^b f(x)dx = S_{k+1} + c_1(\tfrac{1}{2}h_k)^2 + c_2(\tfrac{1}{2}h_k)^4 + \ldots ,$$

und daraus durch Linearkombination

$$\int_a^b f(x)dx = \tfrac{1}{3}(4S_{k+1} - S_k) + d_2h_k^4 + \ldots .$$

Die neu erhaltene Näherungsformel $R_{1k} = \frac{1}{3}(4S_{k+1} - S_k)$ besitzt also eine Fehlerentwicklung, die mit der vierten Potenz in h_k beginnt. Es handelt sich hier um die Simpsonsche Formel Q zur Schrittweite h_{k+1}. Durch eine ähnliche Linearkombination dieser Näherungsformel mit der entsprechenden für die halbe Schrittweite kann man den Fehlersummanden mit h_k^4 zum Verschwinden bringen. Dieses Vorgehen wird systematisch fortgesetzt: Beginnend mit den Sehnentrapezformeln $R_{0k} = S_k$, $k = 0,1,2,\ldots$, werden Rombergsche Näherungsformeln erhalten gemäß

$$R_{ik} = \frac{4^iR_{i-1,k} - R_{i-1,k-1}}{4^i-1}$$

$$= R_{i-1,k} + \frac{R_{i-1,k} - R_{i-1,k-1}}{4^i-1} , \quad i=1,\ldots,k, \quad k=1,2,3,\ldots .$$

Für jede auf [a,b] 2n-mal stetig differenzierbare Funktion f konvergieren daher die Summen R_{ik} gegen das Integral gemäß

$$\left|\int_a^b f(x)dx - R_{ik}\right| \leq \text{const.} h_k^{2(i+1)}, \quad h_k = \frac{b-a}{2^k},$$

für $i = 0,1,\ldots,n-1$, $k = 0,1,2,\ldots$.

Ordnen wir die entsprechenden Romberg-Näherungen in einem Dreieckschema an,

$$\begin{array}{lllll} R_{00} & R_{01} & R_{02} & R_{03} & R_{04} \cdots \\ & R_{11} & R_{12} & R_{13} & R_{14} \cdots \\ & & R_{22} & R_{23} & R_{24} \cdots \\ & & & R_{33} & R_{34} \cdots \end{array} \qquad \begin{array}{l} R_{0k} = S_k \\ R_{1k} = R_{0k} + \frac{1}{3}(R_{0k} - R_{0,k-1}) \\ R_{2k} = R_{1k} + \frac{1}{15}(R_{1k} - R_{1,k-1}) \\ R_{3k} = R_{2k} + \frac{1}{63}(R_{2k} - R_{2,k-1}) \end{array}$$

so bedeutet die Konvergenzaussage, daß für $i = 0,1,\ldots,n-1$ in der i-ten Zeile das Schema Konvergenz herrscht für $k \to \infty$. Gleichzeitig erkennt man, daß für $i>0$ der Wert R_{ik} nur berechnet wird aus den beiden oberhalb und links oberhalb stehenden Werten $R_{i-1,k}$ und $R_{i-1,k-1}$. Für die Berechnung des Romberg-Schemas kann daher der Algorithmus aus Kapitel 4 verwendet werden.

In der obersten Zeile des Dreieckschemas werden die Werte der Sehnentrapezformeln $R_{0k} = S_k$ benötigt. Würde man für $k = 1,2,\ldots$ den Wert von S_k nach der obigen Definition berechnen, so würden $\frac{1}{2}N_k+1$ Funktionswerte von f nochmals berechnet werden, die bereits in der Summe für S_{k-1} benötigt wurden. Neu hinzu kommen nur Funktionswerte an denjenigen Punkten, die in der Tangententrapezformel T_{k-1} zur Schrittweite h_{k-1} auftreten. Es gilt

$$S_k = \frac{1}{2}(S_{k-1} + T_{k-1}), \quad k = 1,2,\ldots,$$

mit

$$T_k = h_k \sum_{j=1}^{N_k} f(a - \tfrac{1}{2}h_k + jh_k), \quad k = 0,1,2,\ldots .$$

Das Unterprogramm a(k) zur Berechnung der Zahlen in der nullten Zeile des Dreieckschemas ergibt sich daher aus

$$R_{Ok} = S_k = \begin{cases} \frac{b-a}{2}(f(a) + f(b)) \text{ für } k = 0, \\ \frac{1}{2}(S_{k-1} + T_{k-1}) \text{ für } k = 1,2,\dots . \end{cases}$$

a	b	h	j	k	N	s	u_0	z

```
z ← 1
k : 0   ≠ ——————————→  a ← a + h/2
s ← f(a)               s ← 0
s ← s + f(b)           j = 1(1)N
N ← 1                  s ← s + f(a + jh)
h ← b - a              s ← hs
a ← a - h              N ← 2N
a(k) ← ½hs             h ← ½h
Rücksprung             a(k) ← ½(u0 + s)
                       Rücksprung
```

Dieses Unterprogramm wird während der Ausführung des Algorithmus aus Kapitel 4 immer nur dann aufgerufen, wenn i = 0 ist. Deshalb wird für die weitere Berechnung der Zahlen R_{ik} in der k-ten Spalte des Romberg-Schemas hier $z = 4^i = 1$ gesetzt; dadurch ist es möglich, bei den nachfolgenden Aufrufen des Unterprogramms b(i,k) für i = 1,2,..., min(k,m) dort den Wert 4^i mit der Anweisung z←4z zu erhalten.

Das Unterprogramm b(i,k) berechnet R_{ik} aus $v = R_{i-1,k}$ und $w = R_{i-1,k-1}$. Es lautet hier nun

i	v	w	z

```
z ← 4z
b(i,k) ← v + (v-w)/(z-1)
Rücksprung.
```

Programm für HP-67/97

Belegung des Datenspeichers für das Hauptprogramm

R_0	R_1	R_2	R_3	R_4		R_{24-m}	...	R_{23}	R_{24}	R_I	$m \leq 14$
i	k	m	v	w		u_m	...	u_1	u_0	Adr u_i =24-i	

Hauptprogramm (Berechnung des Dreieckschemas, s. Kapitel 4)

```
001   0 STO 1                          k ← 0
003   LBL 0 PRTSPC RCL 1 PRTx          Anzeige k
007   A STO 3                          v ← a(k)
009   0 STO 0 24 STO I GTO 2           i ← 0, R_I ← Adr u_0
015   LBL 1 B STO 3                    v ← b(i,k)
018   LBL 2 RCL (i) STO 4              w ← u_i
021   RCL 3 STO (i) PRTx               u_i ← v, Anzeige v
024   1 STO + 0 DSZ I                  i ← i + 1, R_I ← Adr u_i
027   RCL 2 RCL 1 x>y? x↔y             R_X ← min(k,m)
031   RCL 0 x≤y? GTO 1                 wenn i≤min(k,m)
034   1 STO + 1                        k ← k + 1
036   RCL 2 RCL 1 x≤y? GTO 0           wenn k≤m
040   R/S                              Stop
```

Zusätzliche Belegung des Datenspeichers für die Unterprogramme a(k) und b(i,k)

R_5	R_6	R_7	R_8	R_9	R_{80}		R_I
h	N	s	z	a	b		j

Unterprogramm b(i,k)

```
041   LBL B 4 STO * 8                  z ← 4z
044   RCL 3 RCL 4 -                    R_X ← v - w
047   RCL 8 1 - ÷ RCL 3 +              R_X ← v + (v-w)/(z-1)
053   RTN                              Rücksprung
```

Unterprogramm a(k)

```
054  LBL A 1 STO 8                    z ← 1
057  RCL 1 x≠0? GTO 3                 wenn k≠0
060  RCL 9 C STO 7                    s ← f(a)
063  P↔S RCL 0 P↔S C STO + 7          s ← s + f(b)
068  1 STO 6                          N ← 1
070  P↔S RCL 0 P↔S RCL 9 - STO 5      h ← b - a
076  STO - 9 RCL 7 * 2 ÷              a ← a - h, R_X ← 1/2 hs
081  RTN                              Rücksprung
082  LBL 3 RCL 5 2 ÷ STO + 9          a ← a + h/2
087  0 STO 7 RCL 6 STO I              s ← 0, j ← N
091  LBL 4 I RCL 5 * RCL 9 +          R_X ← a + jh
097  C STO + 7                        s ← s + f(a + jh)
099  DSZ I GTO 4                      j ← j - 1, wenn j>0
101  RCL 5 STO * 7                    s ← hs
103  2 STO * 6 STO ÷ 5                N ← 2N, h ← 1/2 h
106  RCL E RCL 7 + 2 ÷                R_X ← 1/2 (u_0 + s)
111  RTN                              Rücksprung
```

Unterprogramm f(x)

```
112  LBL C
```

Programm für TI-58/59

Belegung des Datenspeichers für das Hauptprogramm

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}		$R_{\omega-m}$	...	$R_{\omega-1}$	R_{ω}
i	k	m	v	w	ω	Adr u_i =ω-i		u_m	...	u_1	u_0

Hauptprogramm (Berechnung des Dreieckschemas, s. Kapitel 4)

```
000  Op 16 INV Int * 100 = STO 05     R_05 ← ω
011  0 STO 1                          k ← 0
```

```
014  Lbl INV Adv RCL 1 Prt          Anzeige k
020  A STO 03                       v ← a(k)
023  0 STO 0 RCL 5 STO 6 GTO CE     i ← 0, R06 ← Adr u0
032  Lbl lnx B STO 3                v ← b(i,k)
037  Lbl CE RCL Ind 6 STO 4         w ← ui
043  RCL 3 STO Ind 6 Prt            ui ← v, Anzeige v
048  Op 20 Op 36                    i ← i + 1, R06 ← Adr ui
052  RCL 2 x↔t RCL 1 x≥t CLR x↔t    RT ← min(k,m)
060  Lbl CLR RCL 0 x↔t x≥t lnx      wenn min(k,m)≥i
067  Op 21 RCL 1 x↔t RCL 2 x≥t INV  k ← k + 1, wenn m≥k
076  R/S                            Stop
```

Zusätzliche Belegung des Datenspeichers für die Unterprogramme a(k) und b(i,k)

R_{06}	R_{07}	R_{08}	R_{09}	R_{10}	R_{11}	R_{12}
j	a	b	h	N	s	z

Unterprogramm b(i,k)

```
077  Lbl B 4 Prd 12                 z ← 4z
082  RCL 3 - RCL 4 =                RX ← v - w
088  ÷ ( RCL 12 - 1 ) + RCL 3 =     RX ← v + (v-w)/(z-1)
099  INV SBR                        Rücksprung
```

Unterprogramm a(k)

```
100  Lbl A 1 STO 12                 z ← 1
105  CP RCL 1 INV x=t x↔t           wenn k≠0
111  RCL 7 C STO 11                 s ← f(a)
116  RCL 8 C SUM 11                 s ← s + f(b)
121  1 STO 10 RCL 8 - RCL 7 = STO 9 N ← 1, h ← b - a
132  INV SUM 7 * RCL 11 ÷ 2 =       a ← a - h, RX ← 1/2 hs
141  INV SBR                        Rücksprung
142  Lbl x↔t RCL 9 ÷ 2 = SUM 07     a ← a + h/2
151  0 STO 11 RCL 10 STO 6          s ← 0, j ← N
```

```
158  Lbl x²  RCL 6 * RCL 9 + RCL 7 =   R_X ← a + jh ←
169  C SUM 11                          s ← s + f(a + jh)
172  Dsz 6 x²                          j ← j - 1, wenn j>0 ┘
175  RCL 9 Prd 11                      s ← hs
179  2 Prd 10 INV Prd 9                N ← 2N, h ← ½h
185  RCL 5 STO 6                       R_06 ← Adr u_0
189  RCL Ind 6 + RCL 11 = ÷ 2 =        R_X ← ½(u_0 + s)
198  INV SBR                           Rücksprung
```

Unterprogramm f(x)

```
199  Lbl C
```

Anleitung zur Verwendung der Programme

1. Eingabe des Hauptprogramms und der Unterprogramme B und A.
2. Eingabe des Unterprogramms C zur Berechnung der Funktion f, die integriert werden soll. Aus x im Anzeigeregister wird f(x) berechnet und bei Rücksprung in das rufende Programm im Anzeigeregister übergeben.
3. Eingabe der Integrationsgrenzen a und b
   bei HP-67/97: a STO 9 P↔S b STO 0 P↔S,
   bei TI-58/59: a STO 07 b STO 08.
4. Berechnung und Anzeige von k und der k-ten Spalte des Romberg-Schemas für k = 0,1,...,m: m STO 2 RTN R/S bzw. m STO 02 RST R/S.
5. Soll eine weitere Spalte aus m + 1 Elemente berechnet werden:
   GTO 0 R/S bzw. GTO INV R/S.

Mögliche Wiederholungen:
Schritt 5,
ab Schritt 3 für ein anderes Integrationsintervall,
ab Schritt 2 für eine andere Funktion.

Es ist nicht möglich, ab Schritt 4 für eine andere Wahl von m zu wiederholen, ohne die Integrationsgrenze a (sowie in dem anschließend geschilderten Fall auch b) erneut einzugeben; denn während der Berechnung der Tangententrapezformel wird a verändert.

In dem Programm für die Rechner HP-67/97 ist $m \leq 14$ zulässig. Denn b auf dem Speicherplatz R_{50} wird nur für k = 0 in dem Unterprogramm a(k) benötigt; im Fall m = 14 wird dieser Wert schließlich durch u_m überspeichert.

Die Aufteilung des Speichers zwischen Programmspeicherplätzen und Datenspeicherregistern ist für den Rechner TI-58 bei Einschalten des Gerätes so, daß die Programmspeicherplätze 000-239 und die Datenspeicher R_{00} - R_{29} belegt werden können. Benötigt das Unterprogramm C für den Integranden f(x) nicht mehr als 41 Programmspeicherplätze, so kann daher $m \leq 17$ gewählt werden. Andernfalls stehen nach 2 Op 17 die Programmspeicherplätze 000-319 zur Verfügung, als Datenspeicher sind dann nur noch R_{00} - R_{19} zugelassen; in diesem Fall kann $m \leq 7$ gewählt werden.

Für den Rechner TI-59 ist der Speicher bei Einschalten des Gerätes so aufgeteilt, daß die Programmspeicherplätze 000-479 und die Datenspeicher R_{00} - R_{59} benutzt werden können. Daher ergeben sich im allgemeinen keine Einschränkungen.

In dem vorstehenden Programm für TI-58/59 wurde von dem Grundsatz abgewichen, in Unterprogrammen keine Gleichheitszeichen zu verwenden. Dies ist hier zur Einsparung zahlreicher Klammeroperationen möglich, da bei Aufruf der Unterprogramme A und B in dem zugehörigen Hauptprogramm keine unvollständigen Operationen offenstehen.

Beispiele

Für die Integrale $I_1 = \int_0^1 \frac{4}{x^2+1}\,dx = \pi$ und $I_2 = \int_0^2 e^{-x^2}\,dx = \frac{\sqrt{\pi}}{2}\,\mathrm{erf}(2)$

wurde das Romberg-Schema berechnet. Die Unterprogramme C lauten hier wie bei den Beispielen zur Simpsonschen Formel. Mit m = 5 wurden die nachfolgenden Ergebnisse erhalten. Die zweiten Zeilen enthalten jeweils die Simpsonschen Näherungen.

I_1

0.	1.	2.	3.	4.	5.
3.000000000	3.100000000	3.131176471	3.138988495	3.140941612	3.141429893
	3.133333333	3.141568626	3.141592503	3.141592651	3.141592653
		3.142117648	3.141594095	3.141592661	3.141592653
			3.141585765	3.141592638	3.141592653
				3.141592665	3.141592653
					3.141592653

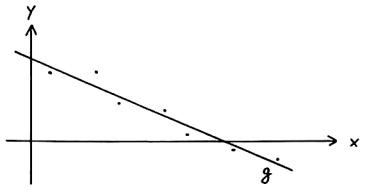
I_2

0.	1.	2.	3.	4.	5.
1.018315639	0.877037261	0.880618634	0.881703792	0.881986246	0.882057558
	0.829944468	0.881812425	0.882065511	0.882080337	0.882081329
		0.885270285	0.882082383	0.882081389	0.882081351
			0.882031781	0.882081373	0.882081391
				0.882081568	0.882081391
					0.882081391

6. Die Ausgleichsgerade

Zunächst befassen wir uns mit der Berechnung der Ausgleichsgeraden unter Verwendung der Tastenfunktion Σ+, wie sie von Taschenrechner-Herstellern vorgeschlagen wird. Beispiele nach [18] zeigen, daß diese Methode anfällig gegen Auslöschungseffekte ist. Im Anschluß daran erläutern wir dann ein numerisch stabiles Berechnungsverfahren.

Zu paarweise verschiedenen Punkten $x_1,\dots,x_k$ und Zahlen $y_1,\dots,y_k$ ist die Ausgleichsgerade g gesucht. Die Ausgleichsgerade approximiert an den Stellen x_i die Funktionswerte y_i optimal im Sinne des Fehlerquadrates:



$$\sum_{i=1}^{k} (y_i - g(x_i))^2 = \min_{\alpha,\beta\in\mathbb{R}} \sum_{i=1}^{k} (y_i - (\alpha x_i + \beta))^2.$$

Die rechts stehende Summe hat als Funktion von α und β eine eindeutig bestimmte Minimalstelle (a,b),

$$a = \frac{1}{d}\left(k \sum_{i=1}^{k} x_i y_i - \sum_{i=1}^{k} x_i \sum_{i=1}^{k} y_i\right) \text{ mit } d = k \sum_{i=1}^{k} x_i^2 - \left(\sum_{i=1}^{k} x_i\right)^2,$$

$$b = \frac{1}{d}\left(\sum_{i=1}^{k} x_i^2 \sum_{i=1}^{k} y_i - \sum_{i=1}^{k} x_i \sum_{i=1}^{k} x_i y_i\right) = \frac{1}{k}\left(\sum_{i=1}^{k} y_i - a \sum_{i=1}^{k} x_i\right).$$

Damit ist die Ausgleichsgerade g dann gegeben durch $g(x) = ax + b$.

Zur Berechnung von a und b werden die Summen

$$s = \sum_{i=1}^{k} y_i,\quad u = \sum_{i=1}^{k} x_i,\quad v = \sum_{i=1}^{k} x_i^2,\quad w = \sum_{i=1}^{k} x_i y_i$$

benötigt. In dem folgenden Algorithmus berechnen wir diese Summen nicht erst dann, wenn alle Wertepaare (x_i,y_i) eingegeben sind; vielmehr wird jeweils nach der Eingabe eines Wertepaares zu jeder Summe der entsprechende neue Summand addiert. Die Wertepaare (x_i,y_i) für $i = 1,\ldots,k$ werden daher nicht vor Beginn des Algorithmus eingegeben, sondern im Laufe der Ausführung. Nach der Eingabe des letzten Wertepaares (x_k,y_k) werden dann die Koeffizienten a und b der Ausgleichsgeraden berechnet. Anschließend können zu eingegebenen Werten x zugehörige Funktionswerte g(x) der Ausgleichsgeraden angezeigt werden, und es wird auch die umgekehrte Fragestellung behandelt: zu gegebenem y wird die Abszisse x ermittelt mit g(x) = y.

Algorithmus Voraussetzung: $k \geq 2$

a	b	i	k	s	u	v	w	x	y

$i \leftarrow 0$
$s \leftarrow 0$
$u \leftarrow 0$
$v \leftarrow 0$
$w \leftarrow 0$

(1) Vorbereitungsschritt: Löschen der Summationsspeicher

Eingabe $x = x_i$, $y = y_i$ ←
$s \leftarrow s + y$
$u \leftarrow u + x$
$v \leftarrow v + x^2$
$w \leftarrow w + xy$
$i \leftarrow i + 1$
$i : k$ (<)

(2) Eingabe der Wertepaare, Berechnung der Summen

$a \leftarrow \frac{kw-su}{kv-u^2}$

$b \leftarrow \frac{s-ua}{k}$

(3) Berechnung von a und b

Eingabe x
Anzeige ax + b

(4) Funktionswert der Ausgleichsgeraden an der Stelle x

Eingabe y
Anzeige $\frac{y-b}{a}$

(5) Zu gegebenem y wird die Stelle x mit g(x) = y berechnet.

Nachdem die Teile (1) bis (3) des Algorithmus durchlaufen sind, können die Teile (4) und (5) unabhängig voneinander ausgeführt und wiederholt werden. Der Algorithmus ist so angelegt, daß weitere Wertepaare (x_i,y_i) hinzugenommen werden können, ohne daß die früher eingegebenen Werte und Rechnungen wiederholt werden müssen: neue Punkte zur Ausgleichung werden dadurch berücksichtigt, daß für sie der Algorithmus ab Teil (2) fortgesetzt wird.

Wir übertragen diesen Algorithmus in ein Programm für die Rechner HP-67/97 und verwenden dabei die Taste Σ+ zur simultanen Berechnung der Summen s,u,v,w. Bei den Rechnern TI-58/59 sind darüber hinaus alle erforderlichen Anweisungsfolgen bereits in speziellen Steueroperationen enthalten, so daß für diese Rechner anschließend nur die Anleitung zur Verwendung aufgeführt ist.

Programm für HP-67/97

Datenspeicher	R_A	R_B		R_{S4}	R_{S5}	R_{S6}	R_{S7}	R_{S8}	R_{S9}
	a	b		s	-	u	v	w	k

(1) Vorbereitungsschritt

001	LBL a P↔S 0 STO 4 STO 6	s ← 0, u ← 0
006	STO 7 STO 8 STO 9 P↔S	v ← 0, w ← 0, k ← 0
010	RTN	Rücksprung

(3) Berechnung von a und b

011	LBL A P↔S RCL 9 RCL 8 *	$R_X \leftarrow kw$
016	RCL 6 RCL 4 * -	$R_X \leftarrow kw - su$
020	RCL 9 RCL 7 * RCL 6 x^2 -	$R_X \leftarrow kv - u^2$
026	÷ STO A	$a \leftarrow (kw-su)/(kv-u^2)$
028	RCL 6 * CHS RCL 4 +	$R_X \leftarrow s - ua$
033	RCL 9 ÷ STO B P↔S	$b \leftarrow (s - ua)/k$
037	RTN	Rücksprung

(4)

038	LBL B RCL A * RCL B +	$R_X \leftarrow ax + b$
043	RTN	Rücksprung

(5)

044	LBL C RCL B - RCL A ÷	$R_X \leftarrow (y - b)/a$
049	RTN	Rücksprung

Programm für TI-58/59

Datenspeicher

R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}
s	-	k	u	v	w

Statt eines eigenen Programms können hier die speziellen Steueroperationen Op 12, Op 14 und Op 15 verwendet werden.

Anleitung zur Verwendung

Eingabe des Programms für HP-67/97; dieser Schritt entfällt für die Rechner TI-58/59.
Im weiteren ist die Gliederung des obigen Algorithmus beibehalten.

(1) Vorbereitungsschritt zur Eingabe
bei HP-67/97: a (Löschen der Summationsspeicher),
bei TI-58/59: Pgm 1 SBR CLR (Löschen von R_{01} bis R_{06} und R_T).

(2) Eingabe der Wertepaare
bei HP-67/97: x_i ENTER y_i $\Sigma+$ für i = 1,...,k,
bei TI-58/59: x_i x↔t y_i $\Sigma+$ für i = 1,...,k.

(3) Berechnung der Koeffizienten der Ausgleichsgeraden bei HP-67/97: A;
dieser Schritt entfällt bei TI-58/59.

(4) Berechnung und Anzeige des Funktionswertes g(x) der Ausgleichsgeraden zu gegebenem x
bei HP-67/97: x B,
bei TI-58/59: x Op 14.

(5) Berechnung und Anzeige der Stelle x mit g(x) = y zu gegebenem y
für HP-67/97: y C,
für TI-58/59: y Op 15.

Mögliche Wiederholungen:

Schritt (4) und Schritt(5) in beliebiger Reihenfolge,
ab Schritt (2) zur zusätzlichen Verwendung weiterer Wertepaare,
ab Schritt (1) für eine andere Ausgleichsaufgabe.

Die Koeffizienten a und b der verwendeten Ausgleichsgeraden können angezeigt werden:
bei HP-67/97 im Anschluß an (3) oder später durch RCL A zur Anzeige von a, RCL B zur Anzeige von b,
bei TI-58/59 im Anschluß an (2) oder später durch Op 12 zur Anzeige von b und anschließend $x \leftrightarrow t$ zur Anzeige von a.

Die oben angegebene Belegung des Datenspeichers durch k, s, u, v, w wird durch die Funktionstaste $\Sigma+$ vorgenommen; gleichzeitig wird noch R_{85} bzw. R_{02} mit $t = \Sigma y_i^2$ belegt, jedoch hat diese Summe für die Ausgleichsgerade keine Bedeutung.

Beispiele

(i) Für i = 1,...,21 wurden die Punkte $x_i = -1 + 0.1(i-1)$ und die Funktionswerte $y_i = e^{x_i}$ gewählt. Die zugehörige Ausgleichsgerade hat die Koeffizienten a = 1.114036447 und b = 1.193651807. Hiermit ist $\max_{1 \le i \le 21} |e^{x_i} - g(x_i)| = e - g(1) = 0.410593574$.

(ii) Wir wählen nach [18] k = 3 und die Werte $x_1 = 665999$, $y_1 = -1$, $x_2 = 666000$, $y_2 = 0$, $x_3 = 666001$, $y_3 = 1$. Die Wertepaare liegen auf einer Geraden. Mit dieser Geraden fällt die Ausgleichsgerade zusammen, und das Minimum in der Definition der Ausgleichsgeraden wird Null. Wir erwarten daher a = 1 und b = -666000.

Die Berechnung mit TI-59 liefert die erwarteten Ergebnisse. Bei der Durchführung der Rechnung mit HP-97 tritt eine Unterbrechung des Programmablaufs infolge Division durch Null auf. Dies kommt daher, daß bei der Berechnung des Nenners $d = kv - u^2$ von a Null entsteht. Während $v = \Sigma x_i^2 = 1330668000002$ sein müßte, kann infolge der zehnstelligen Rechengenauigkeit nur v = 1330668000000 wiedergegeben werden; damit wird numerisch $3v = 3992004 \cdot 10^6$ und $u^2 = (3.666 \cdot 10^3)^2 = 3992004 \cdot 10^6$, so daß numerisch d = 0 folgt im Unterschied zu dem exakten Wert d = 6. In diesem kritischen Beispiel wirkt sich die höhere Rechengenauigkeit der TI-Rechner aus. Jedoch wird die entsprechende Genauigkeitsgrenze bei den TI-Rechnern zum Beispiel bei der Berechnung der Ausgleichsgeraden für die Wertepaare $x_1 = 2665999$, $y_1 = -1$, $x_2 = 2666000$, $y_2 = 0$, $x_3 = 2666001$, $y_3 = 1$ erreicht; hier ist dann entsprechend der Nenner zur Berechnung von a numerisch Null.

Diese Erscheinungen treten dadurch auf, daß mit der Funktionstaste Σ+ sowohl bei den Rechnern HP-67/97 wie auch bei den Rechnern TI-58/59 die Summen s, u, v, w gemäß ihrer Definition berechnet werden. In den Beispielen entstehen bei der Summe $v = \Sigma x_i^2$ Genauigkeitsverluste, und bei der Subtraktion $k\Sigma x_i^2 - (\Sigma x_i)^2$ erfolgt dann die Auslöschung aller wesentlichen Ziffern der Differenz.

Nach [18] geben wir anschließend eine rekursive Berechnungsmethode an, mit der solche Auslöschungseffekte vermieden werden können.

Mit den arithmetischen Mitteln

$$L_k = \frac{1}{k}\sum_{i=1}^{k} x_i, \quad M_k = \frac{1}{k}\sum_{i=1}^{k} y_i$$

lassen sich die Koeffizienten a und b der Ausgleichsgeraden auch schreiben als

$$a = \frac{\sum_{i=1}^{k} x_i y_i - kL_kM_k}{\sum_{i=1}^{k} x_i^2 - kL_k^2}, \quad b = M_k - aL_k.$$

Die arithmetischen Mittel können rekursiv berechnet werden gemäß

$$L_1 = x_1,\ L_k = L_{k-1} + \frac{1}{k}(x_k - L_{k-1}),\ k = 2,3,\ldots,$$

$$M_1 = y_1,\ M_k = M_{k-1} + \frac{1}{k}(y_k - M_{k-1}),\ k = 2,3,\ldots .$$

Für die Nenner N_k der letzten Darstellung von a gilt die Beziehung

$$N_k = \sum_{i=1}^{k} x_i^2 - k\,L_k^2 = \sum_{i=1}^{k} (x_i - L_k)^2.$$

Daraus folgt zusammen mit der Rekursionsformel für L_k auch für N_k eine rekursive Berechnungsmöglichkeit,

$$N_1 = 0,\ N_k = N_{k-1} + \frac{k-1}{k}(x_k - L_{k-1})^2,\ k = 2,3,\ldots .$$

Auf Grund dieser Überlegungen erhält der Algorithmus zur Berechnung der Ausgleichsgeraden nun die folgende Form.

Algorithmus Voraussetzung: $k \geq 2$

a	b	i	k	l	m	n	w	x	y

i ← 0
l ← 0
m ← 0
n ← 0
w ← 0

(1) Vorbereitungsschritt: Löschen der Summationsspeicher

Eingabe $x=x_i$, $y=y_i$ ← (2) Eingabe der Wertepaare, rekursive Rechnung

$w \leftarrow w + xy$

$i \leftarrow i + 1$

$n \leftarrow n + \frac{i-1}{i}(x - l)^2$

$l \leftarrow l + \frac{1}{i}(x - l)$

$m \leftarrow m + \frac{1}{i}(y - m)$

$i : k$ <

$a \leftarrow \frac{1}{n}(w - klm)$ (3) Berechnung von a und b

$b \leftarrow m - al$

Eingabe x (4) Funktionswert der Ausgleichsgeraden an der Stelle x

Anzeige ax + b

Eingabe y (5) Abszisse x, an der die Ausgleichsgerade den Funktionswert y annimmt.

Anzeige $\frac{y-b}{a}$

Programm für HP-67/97

Datenspeicher	R_0	R_1	R_2	R_3	R_4	R_5	R_6		R_A	R_B
	i	l	m	n	w	x	y		a	b
	k									

(1) Vorbereitungsschritt

```
001   LBL b 0 STO 0 STO 1          i ← 0, l ← 0
005   STO 2 STO 3 STO 4            m ← 0, n ← 0, w ← 0
008   RTN                          Rücksprung
```

(2) Eingabe der Wertepaare, rekursive Rechnung

```
009   LBL D STO 6 R↓ STO 5 R↑      R6 ← y, R5 ← x
014   * STO + 4 1 STO + 0          w ← w + xy, i ← i + 1
018   RCL 5 RCL 1 - ENTER x²       RX ← (x - l)²
023   RCL 0 1 - RCL 0 ÷ * STO + 3  n ← n + (x - l)²(i - 1)/i
030   CLx RCL 0 ÷ STO + 1          l ← l + (x - l)/i
034   RCL 6 RCL 2 - RCL 0 ÷ STO + 2  m ← m + (y - m)/i
040   RTN                          Rücksprung
```

(3) Berechnung von a und b

041	LBL E RCL 4 RCL 0 RCL 1 *	$R_X \leftarrow kl$
046	RCL 2 * - RCL 3 ÷ STO A	$a \leftarrow (w - klm)/n$
052	RCL 1 * CHS RCL 2 + STO B	$b \leftarrow m - al$
058	RTN	Rücksprung

(4)

059	LBL B RCL A * RCL B +	$R_X \leftarrow ax + b$
064	RTN	Rücksprung

(5)

065	LBL C RCL B - RCL A ÷	$R_X \leftarrow (y - b)/a$
070	RTN	Rücksprung

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	R_{07}	R_{08}
i	l	m	n	w	x	y	a	b
k								

(1) Vorbereitungsschritt

000	0 STO 0 STO 1 STO 2	$i \leftarrow 0$, $l \leftarrow 0$, $m \leftarrow 0$
007	STO 3 STO 4	$n \leftarrow 0$, $w \leftarrow 0$
011	INV SBR	Rücksprung

(2) Eingabe der Wertepaare, rekursive Rechnung

012	Lbl D STO 6 * x↔t STO 5	$R_{06} \leftarrow y$, $R_{05} \leftarrow x$
020	= SUM 4	$w \leftarrow w + xy$
023	RCL 5 - RCL 1 = x^2 * RCL 0 ÷	$R_X \leftarrow (x - l)^2 i$
034	Op 20 RCL 0 = SUM 3	$i \leftarrow i+1$, $n \leftarrow n+\frac{i-1}{i}(x-l)^2$
041	RCL 5 - RCL 1 =	$R_X \leftarrow x - l$
047	÷ RCL 0 = SUM 1	$l \leftarrow l + (x - l)/i$
053	RCL 6 - RCL 2 =	$R_X \leftarrow y - m$
059	÷ RCL 0 = SUM 2	$m \leftarrow m + (y - m)/i$
065	INV SBR	Rücksprung

(3) Berechnung von a und b

066	Lbl E RCL 4 -	$R_X \leftarrow w$
071	RCL 0 * RCL 1 * RCL 2 =	$R_X \leftarrow w - klm$
080	÷ RCL 3 = STO 7	$a \leftarrow (w - klm)/n$
086	* RCL 1 +/- + RCL 2 = STO 8	$b \leftarrow m - al$
096	INV SBR	Rücksprung

(4)

097	Lbl B * RCL 7 + RCL 8 =	$R_X \leftarrow ax + b$
106	INV SBR	Rücksprung

(5)

107	Lbl C - RCL 8 = ÷ RCL 7 =	$R_X \leftarrow (y - b)/a$
117	INV SBR	Rücksprung

Anleitung zur Verwendung der Programme

Eingabe des Programms.
Es gilt wieder die Gliederung des obigen Algorithmus.

(1) Vorbereitungsschritt (Löschen der Summationsspeicher)
bei HP-67/97: b,
bei TI-58/59: RST R/S.

(2) Eingabe der Wertepaare
bei HP-67/97: x_i ENTER y_i D für i = 1,...,k,
bei TI-58/59: x_i x↔t y_i D für i = 1,...,k.

(3) Berechnung der Koeffizienten der Ausgleichsgeraden: E.

(4) Berechnung und Anzeige eines Funktionswertes g(x) der Ausgleichsgeraden: x B.

(5) Berechnung und Anzeige des Abszissenwertes x mit g(x) = y zu gegebenem y: y C.

Mögliche Wiederholungen:
(4) und (5) in beliebiger Reihenfolge,
ab Schritt (2) zur Berücksichtigung weiterer Wertepaare,
ab Schritt (1) für eine andere Ausgleichsaufgabe.

Die Koeffizienten a und b der berechneten Ausgleichsgeraden können angezeigt werden
bei HP-67/97 durch RCL A bzw. RCL B,
bei TI-58/59 durch RCL 07 bzw. RCL 08.

Beispiele

Bei der Wiederholung der obigen Beispiele liefern nun beide Rechnerarten die genauen Ergebnisse. Ebenso wird die Ausgleichsgerade zu den Wertepaaren $x_1 = 666665999$, $y_1 = -1$, $x_2 = 666666000$, $y_2 = 0$, $x_3 = 666666001$, $y_3 = 1$ richtig ermittelt. Erst für das Beispiel $x_1 = 6666665999$, $y_1 = -1$, $x_2 = 6666666000$, $y_2 = 0$, $x_3 = 6666666001$, $y_3 = 1$ zeigen sich wieder Unterschiede. Mit den Rechnern HP-67/97 werden unzutreffende Koeffizienten a = 1.714285714, $b = -1.142857028 \cdot 10^{10}$ ermittelt; dagegen werden durch die höhere Rechengenauigkeit der Rechner TI-58/59 von diesen Geräten die richtigen Ergebnisse a = 1, b = -6666666000 geliefert. Die Berechnungsfehler der HP-Rechner in diesem Beispiel stammen von der natürlichen Genauigkeitsgrenze, da intern mit zehn geltenden Ziffern gearbeitet wird. Mit der Abkürzung $z = 6666666 \cdot 10^3$ ist hier der Nenner N_3 bei exakter Rechnung $N_3 = x_1^2 + x_2^2 + x_3^2 - 3L_3^2 = (z - 1)^2 + z^2 + (z + 1)^2 - 3z^2 = 2$. Da die eingegebenen Zahlenwerte x_i jedoch bereits die volle Genauigkeit von zehn geltenden Ziffern ausschöpfen, ist die Mittelwertbildung zur Berechnung von L_2 numerisch nicht mehr wirksam ausführbar, denn es würde eine halbe Einheit der zehnten geltenden Stelle benötigt. Daher wird insbesondere N_3 fehlerhaft berechnet.

Diese Beispiele zeigen, daß die Berechnung der Ausgleichsgeraden mit Hilfe der Funktionstaste Σ+ für Daten mit vielen signifikanten Stellen numerisch nicht angemessen ist; durch Auslöschung entstehen dann künstliche Grenzen für den Einsatzbereich der Rechner. Bei der zuletzt dargestellten rekursiven Berechnungsmethode wird dagegen der Anwendungsbereich der Rechner in natürlicher Weise erst dann begrenzt, wenn schon die eingegebenen Werte die volle Rechengenauigkeit des Gerätes ausschöpfen.

7. Anfangswertaufgaben gewöhnlicher Differentialgleichungen

Wir betrachten Anfangswertaufgaben für gewöhnliche Differentialgleichungen in der Gestalt

$$u'(x) = f(x,u(x)),\ a \le x \le b,$$
$$u'(a) = \alpha.$$

Ist eine geschlossene Lösung nicht auffindbar, so ist man darauf angewiesen, Näherungslösungen zu berechnen. In diesem Kapitel erläutern wir, wie Funktionswerte von Näherungslösungen durch Einschrittverfahren und durch Mehrschrittverfahren erhalten werden können, und wir geben die zugehörigen Algorithmen und Programme an.

Diese Methoden sind auch anwendbar auf Anfangswertaufgaben für Systeme gewöhnlicher Differentialgleichungen

$$u_j'(x) = f_j(x,u_1(x),u_2(x),\ldots,u_n(x)),\ a \le x \le b,$$
$$u_j(a) = \alpha_j,\quad j = 1,\ldots,n.$$

Hierzu überträgt man die Näherungsverfahren auf vektorwertige Funktionen $u = (u_1,u_2,\ldots,u_n)$. Anfangswertaufgaben für eine Differentialgleichung höherer Ordnung,

$$u^{(n)}(x) = g(x,u(x),u'(x),\ldots,u^{(n-1)}(x)),\ a \le x \le b,$$
$$u^{(j-1)}(a) = \alpha_j,\quad j = 1,\ldots,n,$$

lassen sich hier ebenfalls einordnen, wenn man sie als Aufgabe für ein Differentialgleichungssystem erster Ordnung formuliert mit $f_j = u_{j+1}$, $j = 1,\ldots,n-1$ und $f_n = g$. Insbesondere für den Fall $n = 2$ sind Anfangswertaufgaben

$$u''(x) = g(x,u(x),u'(x)),\ a \le x \le b,$$
$$u(a) = \alpha_1,\ u'(a) = \alpha_2,$$

damit auf Taschenrechnern näherungsweise lösbar, worauf wir jedoch nicht ausführlicher eingehen können.

Neben den hier behandelten Methoden gibt es noch zahlreiche weitere Ein- und Mehrschrittverfahren, die bei höherem Rechenaufwand bessere Näherungen liefern (s. [4]).

7.1. Einschrittverfahren

Die Anfangswertaufgabe

$$u'(x) = f(x,u(x)),\ a \leq x \leq b,$$
$$u(a) = \alpha,$$

wird numerisch gelöst, indem mit Hilfe von Einschrittverfahren für eine Schrittweite $h \neq 0$ Funktionswerte von Näherungslösungen u_h berechnet werden für $x = a,\ a + h,\ a + 2h, \ldots$.

Wir befassen uns mit m-stufigen Einschrittverfahren vom Runge-Kutta-Typ und geben dafür die allgemeine Struktur des Lösungsalgorithmus an. Im Hinblick auf die Programme für Taschenrechner beschränken wir uns dann auf das Polygonzugverfahren, das verbesserte Polygonzugverfahren und das klassische Runge-Kutta-Verfahren.

Bei m-stufigen Verfahren vom Runge-Kutta-Typ geht man folgendermaßen vor. Beginnend mit $x = a$ und $u_h(a) = \alpha$ (oder einer Näherung α_h für α) wird gerechnet

$$u_h(x+h) = u_h(x) + h(\gamma_1 k_1 + \ldots + \gamma_m k_m)(x, u_h(x))$$

mit

$$k_1(x,y) = f(x,y)$$
$$k_2(x,y) = f(x + \alpha_2 h,\ y + h\beta_{21} k_1(x,y))$$
$$\vdots$$
$$k_m(x,y) = f(x+\alpha_m h,\ y+h(\beta_{m1} k_1 + \ldots + \beta_{m,m-1} k_{m-1})(x,y)).$$

Die Koeffizienten solcher Verfahren werden angegeben in der Form

α_2	β_{21}				
α_3	β_{31}	β_{32}			
.	.	.			
.	.	.			
.	.	.			
α_m	β_{m1}	β_{m2}	...	$\beta_{m,m-1}$	
	γ_1	γ_2	...	γ_{m-1}	γ_m

Beispiele

m = 1

Polygonzugverfahren

	1

m = 2

verbessertes Polygonzugverfahren

$\frac{1}{2}$	$\frac{1}{2}$	
	0	1

verbessertes Verfahren von Euler-Cauchy

1	1	
	$\frac{1}{2}$	$\frac{1}{2}$

m = 3

Verfahren von Kutta

$\frac{1}{2}$	$\frac{1}{2}$		
1	-1	2	
	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$

m = 4

klassisches Runge-Kutta-Verfahren

$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

3/8-Regel

$\frac{1}{3}$	$\frac{1}{3}$			
$\frac{2}{3}$	$-\frac{1}{3}$	1		
1	1	-1	1	
	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$

Verfahren von Kuntzmann

$\frac{2}{5}$	$\frac{2}{5}$			
$\frac{3}{5}$	$-\frac{3}{20}$	$\frac{3}{4}$		
1	$\frac{19}{44}$	$-\frac{15}{44}$	$\frac{40}{44}$	
	$\frac{55}{360}$	$\frac{125}{360}$	$\frac{125}{360}$	$\frac{55}{360}$

Mit $0<h\leq b-a$, $N_h = [(b-a)/h]$ und den Abkürzungen

$$I_h = \{x\in[a,b] \mid x = a + jh,\ j = 0,\dots,N_h\}$$
$$I_h' = \{x\in[a,b] \mid x = a + jh,\ j = 0,\dots,N_h-1\}$$

gilt für diese Beispiele m-stufiger Einschrittverfahren der folgende Konvergenzsatz: Ist die rechte Seite f der Differentialgleichung stetig und erfüllt sie bezüglich der zweiten Veränderlichen eine Lipschitzbedingung, dann konvergieren für $\alpha_h\to\alpha\ (h\to 0)$ die Näherungslösungen u_h gegen die exakte Lösung u der Anfangswertaufgabe für die Differentialgleichung gemäß

$$\max_{x\in I_h} |u_h(x) - u(x)| \to 0 \quad (h\to 0),$$

$$\max_{x\in I_h'} \left|\frac{1}{h}(u_h(x+h)-u_h(x)) - u'(x)\right| \to 0 \quad (h\to 0).$$

Mit $p = m$ gilt darüberhinaus für die Konvergenzordnung die folgende Aussage: Ist die Funktion f zusätzlich p-mal stetig differenzierbar und konvergieren die Anfangswerte gemäß $|\alpha_h - \alpha| \leq \text{const} \cdot h^p$, so gilt

$$|u_h(x) - u(x)| \leq \text{const} \cdot h^p, \quad x \in I_h, \text{ für } h \to 0$$

(s. [4]; insbesondere gilt für $m>4$ nur $p<m$).

Der folgende Algorithmus ermittelt mit Hilfe eines Einschrittverfahrens eine Näherungslösung $u_h(x)$ für $x = a(h)a+jh$. Angezeigt wird jeder n-te Ort x und der zugehörige Funktionswert $u_h(x)$.

Algorithmus

Eingabe: Unterprogramm für f;
h, j, n, x = a, u = α.
Anzeige: h; x, $u_h(x)$ für x = a(nh)a+jh.

Hauptprogramm

h	i	j	n	u	x

Anzeige h
Anzeige x
Anzeige u
i = 1(1)j
  Aufruf "Einschrittverfahren"
  [i/n] : i/n ±
    Anzeige x
    Anzeige u
Stop

Unterprogramm m-stufiges Einschrittverfahren

h	k_1	...	k_m	m	s	u	v	x	x'
—				—		—		—	

```
s = 1(1)m
Aufruf "Unterprogramm U_s"
k_s ← f(x',v)
Aufruf "Unterprogramm U_{m+1}"
x ← x + h
Rücksprung
```

Unterprogramm U_s

```
für s = 1:          x' ← x
                    v ← u
                    Rücksprung
für s = 2,...,m:    x' ← x + α_s h
                    v ← u + h(β_{s1} k_1 + ... + β_{s,s-1} k_{s-1})
                    Rücksprung
für s = m + 1:      u ← u + h(γ_1 k_1 + ... + γ_m k_m)
                    Rücksprung
```

Durch diese Gestalt des Algorithmus wird die allgemeine Struktur der Einschrittverfahren vom Runge-Kutta-Typ wiedergegeben. Die Aufgabe des Unterprogramms "m-stufiges Einschrittverfahren" ist es, die Wertzuweisungen $u \leftarrow u_h(x+h)$, $x \leftarrow x+h$ durchzuführen. Die dabei aufgerufenen Unterprogramme U_s stellen für $s = 1,\ldots,m$ jeweils die beiden Argumente der Funktion f bereit, mit denen k_s berechnet wird. Das Unterprogramm U_{m+1} berechnet $u_h(x+h)$. Die Koeffizienten α_j, β_{jk}, γ_j des Verfahrens treten nur in den Unterprogrammen U_s für $s = 2,\ldots,m+1$ auf.

Für die nachfolgenden Beispiele von Einschrittverfahren verwenden wir jedoch nicht diese allgemeine Struktur. In diesen Fällen kann man nämlich ohne Speichern der Größen $k_1,\ldots,k_m$ auskommen, so daß man bei der Realisierung dieser Verfahren durch Taschenrechnerprogramme weniger Datenspeicherplätze be-

nötigt. Ebenfalls im Hinblick auf die Erstellung von Programmen für Taschenrechner sind die Funktionsaufrufe für die Funktion f so formuliert, daß die beiden Argumente jeweils zuerst berechnet werden und dann auf den festen Plätzen x und v zur Verfügung stehen. Wir geben daher die Unterprogramme für spezielle Einschrittverfahren an.

Unterprogramm Polygonzugverfahren

h	u	v	x

$v \leftarrow u$	
$u \leftarrow u + hf(x,v)$	$u = u_h(x+h) = u_h(x)+hf(x,u_h(x))$
$x \leftarrow x + h$	
Rücksprung	

Unterprogramm verbessertes Polygonzugverfahren

h	u	v	x

$v \leftarrow u$	
$v \leftarrow u + \frac{1}{2}hf(x,v)$	$v = u_h(x)+\frac{1}{2}hf(x,u_h(x))$
$x \leftarrow x + \frac{h}{2}$	$x' = x+\frac{h}{2}$
$u \leftarrow u + hf(x,v)$	$u = u_h(x+h) = u_h(x)+hf(x',v)$
$x \leftarrow x + \frac{h}{2}$	$x = x'+\frac{h}{2}$
Rücksprung	

Unterprogramm klassisches Runge-Kutta-Verfahren

c	h	k	u	v	x

$v \leftarrow u$	
$k \leftarrow hf(x,v)$	$k = hk_1 = hf(x,u_h(x))$
$c \leftarrow k$	$c = hk_1$
$v \leftarrow u + \frac{1}{2}k$	
$x \leftarrow x + \frac{1}{2}h$	$x' = x+\frac{1}{2}h$
$k \leftarrow hf(x,v)$	$k = hk_2 = hf(x',u_h(x)+\frac{h}{2}k_1)$
$c \leftarrow c + 2k$	$c = h(k_1+2k_2)$
$v \leftarrow u + \frac{1}{2}k$	

$k \leftarrow hf(x,v)$	$k = hk_3 = hf(x', u_h(x)+\frac{h}{2}k_2)$
$c \leftarrow c + 2k$	$c = h(k_1+2k_2+2k_3)$
$v \leftarrow u + k$	
$x \leftarrow x + \frac{1}{2}h$	$x'' = x'+\frac{1}{2}h$
$k \leftarrow hf(x,v)$	$k = hk_4 = hf(x'', u_h(x)+hk_3)$
$u \leftarrow u + \frac{1}{6}(c + k)$	$u_h(x+h) = u_h(x)+\frac{h}{6}(k_1+2k_2+2k_3+k_4)$
Rücksprung	

In den nachfolgenden Taschenrechnerprogrammen kommen wir bei dem klassischen Runge-Kutta-Verfahren ohne die Reservierung eines Datenspeicherplatzes für k aus, da wir mit der Speicherarithmetik arbeiten.

Programm für HP-67/97

Belegung des Datenspeichers für das Hauptprogramm

R_0	R_1	R_2	R_3	R_4	R_5
h	i	j	n	u	x

Hauptprogramm

001	LBL A RCL 0 FIX PRTx	Anzeige h
005	RCL 5 PRTx RCL 4 SCI PRTx	Anzeige x=a, Anzeige u=α
010	0 STO 1	i ← 0
012	LBL 0 E	u ← $u_h(x+h)$, x ← x + h
014	1 STO + 1	i ← i + 1
016	RCL 1 RCL 3 ÷ FRAC x≠0? GTO 1	wenn $[\frac{i}{n}] \neq \frac{i}{n}$
022	RCL 5 FIX PRTx RCL 4 SCI PRTx	Anzeige x, Anzeige u
028	LBL 1 RCL 1 RCL 2 x>y? GTO 0	wenn j>i
033	R/S	Stop
034	LBL E	

Zusätzliche Belegung des Datenspeichers für die speziellen Einschrittverfahren (c nur für das Runge-Kutta-Verfahren)

R_6	R_7
v	c

Unterprogramm Polygonzugverfahren

```
034  LBL E RCL 4 STO 6                v ← u
037  B RCL O STO + 5 * STO + 4        u ← u+hf(x,v), x←x+h
042  RTN                              Rücksprung
043  LBL B
```

Unterprogramm verbessertes Polygonzugverfahren

```
034  LBL E RCL 4 STO 6                v ← u
037  B RCL O 2 ÷ STO + 5 * STO + 6    v ← v + h/2 f(x,v), x ← x+h/2
044  B RCL O * STO + 4                u ← u + hf(x,v)
048  RCL O 2 ÷ STO + 5                x ← x + h/2
052  RTN                              Rücksprung
053  LBL B
```

Unterprogramm klassisches Runge-Kutta-Verfahren

```
034  LBL E RCL 4 STO 6                v ← u
037  B RCL O * STO 7                  k ← hf(x,v), c ← k
041  2 ÷ STO + 6 RCL O 2 ÷ STO + 5    v ← v + 1/2 k, x←x+1/2 h
048  B RCL O * STO + 7 STO + 7        k ← hf(x,v), c ← c+2k
053  2 ÷ RCL 4 + STO 6                v ← u + 1/2 k
058  B RCL O * STO + 7 STO + 7        k ← hf(x,v), c ← c+2k
063  RCL 4 + STO 6                    v ← u + k
066  RCL O 2 ÷ STO + 5                x ← x + 1/2 h
070  B RCL O *                        k ← hf(x,v)
073  RCL 7 + 6 ÷ STO + 4              u ← u + 1/6 (c + k)
078  RTN                              Rücksprung
079  LBL B
```

Programm für TI-58/59

Belegung des Datenspeichers für das Hauptprogramm

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}
h	i	j	n	u	x

Hauptprogramm

```
000   Lbl A RCL 0 Prt RCL 5 Prt        Anzeige h, Anzeige x = a
008   RCL 4 Prt 0 STO 1                Anzeige u = α, i←0
014   Lbl INV E                        u ← u_h(x+h), x ← x+h
017   Op 21 RCL 1 ÷ RCL 3 =            i ← i+1, R_X ← i/n
025   INV Int CP INV x=t lnx           wenn [i/n] ≠ i/n
031   RCL 5 Prt RCL 4 Prt              Anzeige x, Anzeige u
037   Lbl lnx RCL 2 x↔t RCL 1          R_T ← j, R_X ← i
044   INV x>t INV                      wenn i<j
047   R/S                              Stop
048   Lbl E
```

Zusätzliche Belegung des Datenspeichers für die speziellen Einschrittverfahren (c nur für das Ruge-Kutta-Verfahren)

R_{06}	R_{07}
v	c

Unterprogramm Polygonzugverfahren

```
048   Lbl E RCL 4 STO 6                v ← u
054   B * RCL 0 SUM 5 = SUM 4          u ← u+hf(x,v), x ← x+h
063   INV SBR                          Rücksprung
064   Lbl B
```

Unterprogramm verbessertes Polygonzugverfahren

```
048   Lbl E RCL 4 STO 6                v ← u
054   B * ( RCL 0 ÷ 2 )                R_X ← h/2
062   SUM 5 = SUM 6                    v ← v+(h/2)f(x,v), x ← x+h/2
067   B * RCL 0 = SUM 4                u ← u+hf(x,v)
074   RCL 0 ÷ 2 = SUM 5                x ← x + h/2
081   INV SBR                          Rücksprung
082   Lbl B
```

Unterprogramm klassisches Runge-Kutta-Verfahren

048	Lbl E RCL 4 STO 6	$v \leftarrow u$
054	B * RCL O = STO 7	$k \leftarrow hf(x,v)$, $c \leftarrow k$
061	÷ 2 = SUM 6 RCL O ÷ 2 = SUM 5	$v \leftarrow v + \frac{k}{2}$, $x \leftarrow x + \frac{h}{2}$
073	B * RCL O = SUM 7 SUM 7	$k \leftarrow hf(x,v)$, $c \leftarrow c + 2k$
082	÷ 2 + RCL 4 = STO 6	$v \leftarrow u + \frac{k}{2}$
090	B * RCL O = SUM 7 SUM 7	$k \leftarrow hf(x,v)$, $c \leftarrow c + 2k$
099	+ RCL 4 = STO 6	$v \leftarrow u + k$
105	RCL O ÷ 2 = SUM 5	$x \leftarrow x + \frac{h}{2}$
112	B * RCL O + RCL 7 =	$k \leftarrow hf(x,v)$, $R_X \leftarrow c + k$
120	÷ 6 = SUM 4	$u \leftarrow u + \frac{1}{6}(c + k)$
125	INV SBR	Rücksprung
126	Lbl B	

Anleitung zur Verwendung der Programme

1. Eingabe des Hauptprogramms und eines Unterprogramms für ein Einschrittverfahren.
2. Eingabe des Unterprogramms B zur Berechnung von f(x,v) mit x aus R_5 und v aus R_6; bei Rücksprung in das aufrufende Programm wird der berechnete Funktionswert im Anzeigeregister übergeben.
3. Eingabe der Schrittweite h, der Zahlen j und n sowie der Anfangsstelle a und der Anfangsbedingung u(a) = α
   bei den Rechnern HP-67/97: h STO O j STO 2 n STO 3
   a STO 5 α STO 4,
   bei den Rechnern TI-58/59: h STO OO j STO O2 n STO O3
   a STO O5 α STO O4.
4. Berechnung von x und $u_h(x)$ für x = a+h(h)a+jh, Anzeige von h und Anzeige von x, $u_h(x)$ für x = a(nh)a+jh: A.
5. Sollen zusätzliche Werte für weitere Punkte x = a+(j+1)h, a+(j+2)h,..., a+Jh berechnet und angezeigt werden, so gibt man den Wert für J ein: J STO 2 bzw. J STO O2; die weitere Programmausführung wird dann gestartet durch GTO O R/S bzw. GTO INV R/S.

Mögliche Wiederholungen:

Schritt 5,
ab Schritt 3 für eine andere Wahl der Eingabegrößen,
ab Schritt 2 für eine andere Differentialgleichung.

Nach Abschluß der Rechnung in Schritt 4 und 5 sind die eingegebenen Werte von h, n und j bzw. J unverändert. Bei einer Wiederholung des Programmablaufs müssen die Größen a und α immer wieder neu eingegeben werden; die anderen Eingabegrößen nur, soweit sie verändert werden sollen.

Beispiel

Für die Anfangswertaufgabe $u'(x) = 1 + u(x)^2$, $0 \le x < \frac{\pi}{2}$, $u(0) = 0$, wurden Näherungslösungen ermittelt mit Hilfe des Polygonzugverfahrens, des verbesserten Polygonzugverfahrens und des klassischen Runge-Kutta-Verfahrens. Mit h = 0.1, j = 15, n = 1, a = 0, α = 0 wurden Näherungswerte $u_h(x)$ für x = 0(0.1)1.5 erhalten und angezeigt; für h = 0.05, j = 30, n = 2, a = 0, α = 0 wurden Näherungswerte berechnet für x = 0(0.05)1.5 und angezeigt für x = 0(0.1)1.5. Das Unterprogramm B zur Berechnung der Funktion f lautet

für die Rechner HP-67/97

LBL B RCL 6 x^2 1 + RTN $\qquad R_X \leftarrow 1+v^2$, Rücksprung,

für die Rechner TI-58/59

Lbl B (1 + RCL 6 x^2) INV SBR $\qquad R_X \leftarrow 1+v^2$, Rücksprung.

Mit veränderter Ausgabeanweisung entstanden die folgenden Ergebnisse. Zum Vergleich ist noch die exakte Lösung u(x) = tan x der Anfangswertaufgabe tabelliert für x = 0(0.1)1.5.

x	u_h für h=0.1	u_h für h=0.05	tan x

Polygonzugverfahren

0.0	0.00000000	0.00000000	0.00000000
0.1	0.10000000	0.10012500	0.10033467
0.2	0.20100000	0.20176066	0.20271004
0.3	0.30504010	0.30701665	0.30933625
0.4	0.41434505	0.41827203	0.42279322
0.5	0.53151323	0.53839699	0.54630249
0.6	0.65976386	0.67106441	0.68413681
0.7	0.80329269	0.82122640	0.84228838
0.8	0.96782061	0.99589350	1.02963856
0.9	1.16148828	1.20548791	1.26015822
1.0	1.39639379	1.46634682	1.55740772
1.1	1.69138535	1.80570080	1.96475966
1.2	2.07746378	2.27249182	2.57215162
1.3	2.60904936	2.96370411	3.60210245
1.4	3.38976322	4.09900065	5.79788372
1.5	4.63881269	6.28364241	14.10141995

verbessertes Polygonzugverfahren

0.0	0.00000000	0.00000000	0.00000000
0.1	0.10025000	0.10031320	0.10033467
0.2	0.20252263	0.20266219	0.20271004
0.3	0.30900339	0.30925068	0.30933625
0.4	0.42223680	0.42264918	0.42279322
0.5	0.54538743	0.54606389	0.54630249
0.6	0.68262914	0.68374057	0.68413681
0.7	0.83977189	0.84162102	0.84228838
0.8	1.02534044	1.02848641	1.02963856
0.9	1.25255777	1.25809351	1.26015822
1.0	1.54327465	1.55350103	1.55740772
1.1	1.93649215	1.95675519	1.96475966
1.2	2.50911638	2.55364334	2.57215162
1.3	3.43504629	3.54996860	3.60210245
1.4	5.19562790	5.58565872	5.79788372
1.5	9.64549913	11.93794474	14.10141995

klassisches Runge-Kutta-Verfahren

x	u_h für h=0.1	u_h für h=0.05	tan x
0.0	0.00000000	0.00000000	0.00000000
0.1	0.10033459	0.10033467	0.10033467
0.2	0.20270988	0.20271003	0.20271004
0.3	0.30933604	0.30933624	0.30933625
0.4	0.42279299	0.42279321	0.42279322
0.5	0.54630231	0.54630248	0.54630249
0.6	0.68413676	0.68413681	0.68413681
0.7	0.84228857	0.84228840	0.84228838
0.8	1.02963906	1.02963861	1.02963856
0.9	1.26015878	1.26015831	1.26015822
1.0	1.55740644	1.55740776	1.55740772
1.1	1.96474656	1.96475908	1.96475966
1.2	2.57207175	2.57214704	2.57215162
1.3	3.60156340	3.60206730	3.60210245
1.4	5.79197480	5.79743653	5.79788372
1.5	13.83665092	14.07013099	14.10141995

7.2. Mehrschrittverfahren

In diesem Abschnitt beschäftigen wir uns damit, für Anfangswertaufgaben

$$u'(x) = f(x,u(x)),\ a \le x \le b,$$
$$u(a) = \alpha,$$

Näherungslösungen $u_h(x)$ für x = a, a+h, a+2h,... zu ermitteln mit Hilfe von Mehrschrittverfahren. Im Gegensatz zu den Einschrittverfahren hängt bei den Mehrschrittverfahren die Berechnung von $u_h(x+h)$ von der Kenntnis mehrerer zurückliegender Funktionswerte ab, zum Beispiel von $u_h(x)$, $u_h(x-h)$, $u_h(x-2h)$. Dadurch wird es möglich, mit geringerem Rechenaufwand hohe Konvergenzordnungen zu erreichen. Andererseits ist von der Differentialgleichungsaufgabe her nur die Anfangsbedingung u(a) = α gegeben. Bevor die Mehrschrittverfahren

zur Ermittlung von Funktionswerten von Näherungslösungen u_h eingesetzt werden können, müssen deshalb erst mehrere aufeinanderfolgende Funktionswerte von u_h bereitgestellt sein. Dies geschieht in einer sogenannten Anlaufrechnung, in der wir Einschrittverfahren verwenden.

Beispiele für Mehrschrittverfahren sind

das Extrapolationsverfahren von Adams-Bashforth

$$u_h(x+h) = u_h(x) + \frac{h}{12}(23f_0 - 16f_{-1} + 5f_{-2}),$$

das Prädiktor-Korrektor-Verfahren von Adams

$$\tilde{u}_h(x+h) = u_h(x) + \frac{h}{12}(23f_0 - 16f_{-1} + 5f_{-2})$$

$$u_h(x+h) = u_h(x) + \frac{h}{24}(9\tilde{f}_1 + 19f_0 - 5f_{-1} + f_{-2}),$$

das Interpolationsverfahren von Adams-Moulton

$$u_h(x+h) = u_h(x) + \frac{h}{24}(9f_1 + 19f_0 - 5f_{-1} + f_{-2}),$$

sowie

das Extrapolationsverfahren von Nyström

$$u_h(x+h) = u_h(x-h) + \frac{h}{3}(7f_0 - 2f_{-1} + f_{-2}),$$

das Prädiktor-Korrektor-Verfahren von Milne

$$\tilde{u}_h(x+h) = u_h(x-h) + \frac{h}{3}(7f_0 - 2f_{-1} + f_{-2})$$

$$u_h(x+h) = u_h(x-h) + \frac{h}{3}(\tilde{f}_1 + 4f_0 + f_{-1}),$$

das Interpolationsverfahren von Milne-Simpson

$$u_h(x+h) = u_h(x-h) + \frac{h}{3}(f_1 + 4f_0 + f_{-1}).$$

Hierbei sind die Abkürzungen verwendet

$$f_j = f(x+jh, u_h(x+jh)),\ j = -2,-1,0,1, \text{und } \tilde{f}_1 = f(x+h, \tilde{u}_h(x+h)).$$

Im Fall der Interpolationsverfahren liegt für $z = u_h(x+h)$ eine nichtlineare Gleichung $z = g(z)$ vor. Sie wird hier näherungsweise gelöst mit dem Verfahren der sukzessiven Approximation, indem für $s = 0,1,2,3$ die Schritte der Iteration

$z^{(s+1)} = g(z^{(s)})$ ausgeführt werden; der Funktionswert $u_h(x)$ im vorhergehenden Punkt dient dabei als Startwert $z^{(0)}$. In entsprechender Weise können die Prädiktor-Korrektor-Verfahren aufgefaßt werden. Dort wird nur ein Schritt der sukzessiven Approximation zur Lösung der Gleichung $z = g(z)$ ausgeführt; jedoch dient als Startwert die Näherung für $u_h(x+h)$, die mit Hilfe des zugehörigen Extrapolationsverfahrens erhalten wurde.

Für die hier angegebenen Mehrschrittverfahren gilt ähnlich wie bei den Einschrittverfahren die folgende Konvergenzaussage: Die rechte Seite f der Differentialgleichung sei stetig und erfülle bezüglich der zweiten Veränderlichen eine Lipschitzbedingung; bei konvergenter Anlaufrechnung konvergieren dann die Näherungslösungen u_h gegen die exakte Lösung u der Anfangswertaufgabe für die Differentialgleichung gemäß

$$\max_{x \in I_h} |u_h(x) - u(x)| \to 0 \text{ für } h \to 0.$$

Die Ableitung u' wird durch geeignete Differenzenquotienten approximiert. Mit p = 3 für die angegebenen Extrapolationsverfahren und mit p = 4 für die angegebenen Prädiktor-Korrektor-Verfahren und die Interpolationsverfahren gilt darüber hinaus für die Konvergenzordnung: Ist zusätzlich die Lösung u der Anfangswertaufgabe (p+1)-mal stetig differenzierbar, so gilt

$$|u_h(x) - u(x)| \leq \text{const.} \cdot h^p, \quad x \in I_h, \text{ für } h \to 0,$$

wenn auch die Anlaufrechnung mit dieser Konvergenzordnung durchgeführt worden ist.

Die Konvergenzordnung dieser Verfahren wird also nur erreicht, wenn die Anlaufrechnung für die angegebenen Extrapolationsverfahren mit der Ordnung h^3 ausgeführt wird; bei den hier betrachteten Prädiktor-Korrektor-Verfahren und den Interpolationsverfahren wird in der Anlaufrechnung die Ordnung h^4 benötigt.

Mit dem folgenden Algorithmus wird eine Näherungslösung $u_h(x)$ berechnet für x = a(h)a+jh. Angezeigt wird jeder n-te Ort x mit dem zugehörigen Funktionswert $u_h(x)$. In der Anlaufrechnung verwenden wir das klassische Runge-Kutta-Verfahren, so daß für die anschließend untersuchten Mehrschrittverfahren die Konvergenzordnung gesichert ist.

Algorithmus Eingabe: Unterprogramm für f;
h, j, n, x=a, u=α.
Anzeige: h; x, $u_h(x)$ für x = a(nh)a+jh.

Hauptprogramm

h	i	j	n	u	x

```
Anzeige h
Anzeige x
Anzeige u
Aufruf "Anlaufrechnung"
i = 2(1)j
  Aufruf "Mehrschrittverfahren"
  [i/n] : i/n  ≠
    Anzeige x
    Anzeige u
Stop
```

Dieses Hauptprogramm entspricht demjenigen für die Einschrittverfahren; hinzugekommen ist die Anlaufrechnung. Für die hier betrachteten Mehrschrittverfahren hat die Anlaufrechnung die anschließend angegebene gemeinsame Gestalt. Es werden aus $u_h(a)$ unter Verwendung des klassischen Runge-Kutta-Verfahrens Funktionswerte $u_h(x)$ für x = a+h und x = a-h berechnet. Mit diesen Werten werden für x = a+h die Größen f_{-2} = $f(a-h,u_h(a-h))$, $f_{-1} = f(a,u_h(a))$ und $f_0 = f(a+h,u_h(a+h))$ bereitgestellt. Ausgehend von $u = u_h(a+h)$ - sowie $w = u_h(a)$ für die Verfahren von Nyström und Milne - kann dann im Anschluß

an die Anlaufrechnung eines der obigen Mehrschrittverfahren zur Berechnung von $u_h(a+2h)$ und den nachfolgenden Funktionswerten der Näherungslösung angewandt werden.

Unterprogramm Anlaufrechnung

f_0	f_{-1}	f_{-2}	h	n	u	v	w	x

	Erläuterungen
w ← u	w = $u_h(a)$
v ← u	
f_{-1} ← f(x,v)	f_{-1} = f(a,$u_h(a)$)
h ← -h	
Aufruf "klassisches Runge-Kutta-Verfahren"	u=$u_h(a-h)$,x=a-h
h ← -h	
v ← u	
f_{-2} ← f(x,v)	f_{-2}=f(a-h,$u_h(a-h)$)
u ← w	u = $u_h(a)$
x ← x+h	x = a
Aufruf "klassisches Runge-Kutta-Verfahren"	u=$u_h(a+h)$,x=a+h
n : 1 ∔	
Anzeige x	
Anzeige u	
v ← u ←	
f_0 ← f(x,v)	f_0=f(a+h,$u_h(a+h)$)
Rücksprung	

Das jeweilige Unterprogramm "Mehrschrittverfahren" nimmt die Wertzuweisungen u←$u_h(x+h)$, x←x+h vor. Dies geschieht dadurch, daß mit u = $u_h(x)$ (und w = $u_h(x-h)$ bei den Verfahren von Nyström und Milne) der Wert v = $u_h(x+h)$ berechnet wird. Außerdem werden f_0, f_{-1} und v (sowie bei den Verfahren von Nyström und Milne zuvor u) so umgespeichert, daß sie für den nächsten Aufruf dieses Unterprogramms bereits in der richtigen Anordnung zur Verfügung stehen; zu diesem Zweck wird auch der dann benötigte neue Wert f_0 = f(x+h,$u_h(x+h)$) berechnet.

Unterprogramm Extrapolationsverfahren von Adams-Bashforth

f_0	f_{-1}	f_{-2}	h	u	v	x

Erläuterungen

$u \leftarrow u + \frac{h}{12}(23f_0 - 16f_{-1} + 5f_{-2})$

$v \leftarrow u$ — $u = v = u_h(x+h)$

$x \leftarrow x + h$

$f_{-2} \leftarrow f_{-1}$

$f_{-1} \leftarrow f_0$

$f_0 \leftarrow f(x,v)$ — $f_0 = f(x+h, u_h(x+h))$

Rücksprung

Unterprogramm Prädiktor-Korrektor-Verfahren von Adams

f_0	f_{-1}	f_{-2}	h	u	v	x

Erläuterungen

$v \leftarrow u + \frac{h}{12}(23f_0 - 16f_{-1} + 5f_{-2})$ — $v = \tilde{u}_h(x+h)$

$x \leftarrow x + h$

$u \leftarrow u + \frac{h}{24}(9f(x,v) + 19f_0 - 5f_{-1} + f_{-2})$

$v \leftarrow u$ — $u = v = u_h(x+h)$

$f_{-2} \leftarrow f_{-1}$

$f_{-1} \leftarrow f_0$

$f_0 \leftarrow f(x,v)$ — $f_0 = f(x+h, u_h(x+h))$

Rücksprung

Unterprogramm Interpolationsverfahren von Adams-Moulton

f_0	f_{-1}	f_{-2}	h	s	u	v	x

Erläuterungen

$v \leftarrow u$ — $v = z^{(0)} = u_h(x)$

$x \leftarrow x + h$

$u \leftarrow u + \frac{h}{24}(19f_0 - 5f_{-1} + f_{-2})$ — $\bar{u} = u + \frac{h}{24}(19f_0 - 5f_{-1} + f_{-2})$

$s = 1(1)4$

$v \leftarrow u + \frac{9}{24}hf(x,v)$ — $v = z^{(s)} = \bar{u} + \frac{9}{24}hf(x+h, z^{(s-1)})$

$u \leftarrow v$ — $u = v = z^{(4)} = u_h(x+h)$

$f_{-2} \leftarrow f_{-1}$

$f_{-1} \leftarrow f_0$

$f_0 \leftarrow f(x,v)$ — $f_0 = f(x+h, u_h(x+h))$

Rücksprung

Unterprogramm Extrapolationsverfahren von Nyström

f_0	f_{-1}	f_{-2}	h	u	v	w	x

	Erläuterungen
$v \leftarrow w + \frac{h}{3}(7f_0 - 2f_{-1} + f_{-2})$	$v = u_h(x+h)$
$w \leftarrow u$	$w = u_h(x)$
$u \leftarrow v$	$u = u_h(x+h)$
$x \leftarrow x + h$	
$f_{-2} \leftarrow f_{-1}$	
$f_{-1} \leftarrow f_0$	
$f_0 \leftarrow f(x,v)$	$f_0 = f(x+h, u_h(x+h))$
Rücksprung	

Unterprogramm Prädiktor-Korrektor-Verfahren von Milne

f_0	f_{-1}	f_{-2}	h	u	v	w	x

	Erläuterungen
$v \leftarrow w + \frac{h}{3}(7f_0 - 2f_{-1} + f_{-2})$	$v = \tilde{u}_h(x+h)$
$x \leftarrow x + h$	
$v \leftarrow w + \frac{h}{3}(f(x,v) + 4f_0 + f_{-1})$	$v = u_h(x+h)$
$w \leftarrow u$	$w = u_h(x)$
$u \leftarrow v$	$u = u_h(x+h)$
$f_{-2} \leftarrow f_{-1}$	
$f_{-1} \leftarrow f_0$	
$f_0 \leftarrow f(x,v)$	$f_0 = f(x+h, u_h(x+h))$
Rücksprung	

Unterprogramm Interpolationsverfahren von Milne-Simpson

f_0	f_{-1}	f_{-2}	h	s	u	v	w	x

	Erläuterungen
$v \leftarrow u$	$v = z^{(0)} = u_h(x)$
$x \leftarrow x + h$	
$w \leftarrow w + \frac{h}{3}(4f_0 + f_{-1})$	$\bar{w} = w + \frac{h}{3}(4f_0 + f_{-1})$
$s = 1(1)4$	
$v \leftarrow w + \frac{1}{3}hf(x,v)$	$v = z^{(s)} = \bar{w} + \frac{h}{3}f(x+h, z^{(s-1)})$
$w \leftarrow u$	$w = u_h(x)$
$u \leftarrow v$	$u = v = z^{(4)} = u_h(x+h)$
$f_{-2} \leftarrow f_{-1}$	
$f_{-1} \leftarrow f_0$	
$f_0 \leftarrow f(x,v)$	$f_0 = f(x+h, u_h(x+h))$
Rücksprung	

Die aufgeführten Variablen der einzelnen Algorithmen dienen als Übersicht über den Bedarf an Datenspeicherplätzen für die zugehörigen Programme. Über die bereits genannten Variablen hinaus wird noch ein Datenspeicherplatz für die Größe c benötigt, die in dem klassischen Runge-Kutta-Verfahren während der Anlaufrechnung benötigt wird. Wir speichern im folgenden c gemeinsam mit f_0, da f_0 in der Anlaufrechnung erst nach dem letzten Aufruf des Unterprogramms für das klassische Runge-Kutta-Verfahren berechnet und gespeichert wird.

Bei den Interpolationsverfahren durchlaufen wir die Schleife s = 1(1)4 wieder rückwärts in der Form s = 4(-1)1 und steuern sie mit der Dsz-Anweisung. In dem Programm für die Rechner HP-67/97 wird deshalb s im Indexregister R_I gespeichert. Da bei den TI-Rechnern die Dsz-Anweisung für die Datenregister R_{00} bis R_{09} möglich ist, verwenden wir für s den Datenspeicherplatz R_{09}, auf dem auch f_{-2} gespeichert wird; dies ist möglich, weil in den Programmen für die Rechner TI-58/59 entsprechend den Algorithmen für die Interpolationsverfahren der bisherige Wert von f_{-2} letztmalig vor der Schleife s = 4(-1)1 verwendet wird, während der neue Wert für f_{-2} erst nach Beendigung der Schleife gespeichert wird.

Programm für HP-67-97

Datenspeicher

R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{s0}		R_I
h	i	j	n	u	x	v	f_0	f_{-1}	f_{-2}	w		s
							c					

Hauptprogramm

```
001   LBL A RCL 0 FIX PRTx              Anzeige h
005   RCL 5 PRTx RCL 4 SCI PRTx         Anzeige x=a, Anzeige u=α
010   a 1 STO 1                         Anlaufrechnung, i ← 1
```

013	LBL O D	$u \leftarrow u_h(x+h)$, $x \leftarrow x+h$
015	1 STO + 1	$i \leftarrow i + 1$
017	RCL 1 RCL 3 ÷ FRAC x≠0? GTO 1	wenn $[\frac{i}{n}] \neq \frac{i}{n}$
023	RCL 5 FIX PRTx RCL 4 SCI PRTx	Anzeige x, Anzeige u
029	LBL 1 RCL 1 RCL 2 x>y? GTO O	wenn j>i
034	R/S	Stop

Unterprogramm Anlaufrechnung

035	LBL a RCL 4 P↔S STO O P↔S	w ← u
040	STO 6 B STO 8 1 CHS STO * O	$v \leftarrow u, f_{-1} \leftarrow f(x,v), h \leftarrow -h$
046	E	$u \leftarrow u_h(a-h)$, $x \leftarrow a-h$
047	1 CHS STO * O RCL 4 STO 6	h ← -h, v ← u
052	B STO 9 P↔S RCL O P↔S STO 4	$f_{-2} \leftarrow f(x,v)$, u ← w
058	RCL O STO + 5	x ← x + h
060	E	$u \leftarrow u_h(a+h)$, x ← a+h
061	RCL 3 1 x≠y? GTO 2	wenn n ≠ 1
065	RCL 5 FIX PRTx RCL 4 SCI PRTx	Anzeige x, Anzeige u
071	LBL 2 RCL 4 STO 6 B STO 7	$v \leftarrow u$, $f_O \leftarrow f(x,v)$
076	RTN	Rücksprung

Unterprogramm klassisches Runge-Kutta-Verfahren

077	LBL E RCL 4 STO 6	v ← u
080	B RCL O * STO 7	k ← hf(x,v), c ← k
084	2 ÷ STO + 6 RCL O 2 ÷ STO + 5	$v \leftarrow v + \frac{1}{2}k$, $x \leftarrow x + \frac{1}{2}h$
091	B RCL O * STO + 7 STO + 7	k ← hf(x,v), c ← c + 2k
096	2 ÷ RCL 4 + STO 6	$v \leftarrow u + \frac{1}{2}k$
101	B RCL O * STO + 7 STO + 7	k ← hf(x,v), c ← c + 2k
106	RCL 4 + STO 6	v ← u + k
109	RCL O 2 ÷ STO + 5	$x \leftarrow x + \frac{h}{2}$
113	B RCL O *	k ← hf(x,v)
116	RCL 7 + 6 ÷ STO + 4	$u \leftarrow u + \frac{1}{6}(c + k)$
121	RTN	Rücksprung
122	LBL D	

Unterprogramme für die speziellen Mehrschrittverfahren

Unterprogramm Extrapolationsverfahren von Adams-Bashforth

```
122  LBL D RCL 9 5 *                    }
126  RCL 8 STO 9 16 * -                 } u←u+ h/12(23f0-16f-1+5f-2)
132  RCL 7 STO 8 23 * +                 } und f-2←f-1, f-1←f0
138  RCL 0 * 12 ÷ STO + 4               }
144  RCL 4 STO 6 RCL 0 STO + 5          v ← u, x ← x + h
148  B STO 7                            f0 ← f(x,v)
150  RTN                                Rücksprung
151  LBL B
```

Unterprogramm Prädiktor-Korrektor-Verfahren von Adams

```
122  LBL D RCL 9 5 * RCL 8 16 * -       RX ← -16f-1 + 5f-2
131  RCL 7 23 * + RCL 0 * 12 ÷          RX ← h/12(23f0-16f-1+5f-2)
141  RCL 4 + STO 6                      v ← u+ h/12(23f0-16f-1+5f-2)
144  RCL 0 STO + 5                      x ← x + h
146  B 9 * RCL 9 + RCL 8 STO 9 5 *      }
155  - RCL 7 STO 8 19 * +               } u ← u+ h/24(9f(x,v)+19f0
162  RCL 0 * 24 ÷ STO + 4               }          -5f-1+f-2)
168  RCL 4 STO 6 B STO 7                und f-2←f-1, f-1←f0
172  RTN                                v ← u, f0 ← f(x,v)
173  LBL B                              Rücksprung
```

Unterprogramm Interpolationsverfahren von Adams-Moulton

```
122  LBL D RCL 4 STO 6                  v ← u
125  RCL 0 STO + 5                      x ← x + h
127  RCL 9 RCL 8 STO 9 5 * -            }
133  RCL 7 STO 8 19 * +                 } u←u+ h/24(19f0-5f-1+f-2)
139  RCL 0 * 24 ÷ STO + 4               } und f-2←f-1, f-1←f0
145  4 STO I                            s ← 4
147  LBL 3 B RCL 0 * 9 * 24 ÷           RX ← 9/24hf(x,v)   <-----+
156  RCL 4 + STO 6                      v ← u + 9/24hf(x,v)      |
159  DSZ I GTO 3                        s ← s-1, wenn s>0  ------+
161  STO 4 B STO 7                      u ← v, f0 ← f(x,v)
164  RTN
165  LBL B
```

Unterprogramm Extrapolationsverfahren von Nyström

```
122  LBL D RCL 9 RCL 8 STO 9 2 * -
129  RCL 7 STO 8 7 * + RCL O * 3 ÷
138  P↔S RCL O P↔S + STO 6
143  RCL 4 P↔S STO O P↔S
147  RCL 6 STO 4 RCL O STO + 5
151  B STO 7
153  RTN
154  LBL B
```

$v \leftarrow w+\frac{h}{3}(7f_0-2f_{-1}+f_{-2})$ und $f_{-2}\leftarrow f_{-1}, f_{-1}\leftarrow f_0$

$w \leftarrow u$

$u \leftarrow v$, $x \leftarrow x + h$

$f_0 \leftarrow f(x,v)$

Rücksprung

Unterprogramm Prädiktor-Korrektor-Verfahren von Milne

```
122  LBL D RCL 9 RCL 8 2 * -
128  RCL 7 7 * + RCL O * 3 ÷
136  P↔S RCL O P↔S + STO 6
141  RCL O STO + 5
143  B RCL 8 STO 9 +
147  RCL 7 STO 8 4 * + RCL O * 3 ÷
156  P↔S RCL O P↔S + STO 6
161  RCL 4 P↔S STO O P↔S
165  RCL 6 STO 4 B STO 7
169  RTN
170  LBL B
```

$R_x \leftarrow -2f_{-1} + f_{-2}$

$R_x \leftarrow \frac{h}{3}(7f_0-2f_{-1}+f_{-2})$

$v \leftarrow w+\frac{h}{3}(7f_0-2f_{-1}+f_{-2})$

$x \leftarrow x + h$

$v \leftarrow w+\frac{h}{3}(f(x,v)+4f_0+f_{-1})$ und $f_{-2}\leftarrow f_{-1}, f_{-1}\leftarrow f_0$

$w \leftarrow u$

$u \leftarrow v$, $f_0 \leftarrow f(x,v)$

Rücksprung

Unterprogramm Interpolationsverfahren von Milne-Simpson

```
122  LBL D RCL 4 STO 6
125  RCL O STO + 5
127  RCL 8 STO 9 RCL 7 STO 8 4 * +
134  RCL O * 3 ÷ P↔S STO + O P↔S
141  4 STO I
143  LBL 3 B RCL O * 3 ÷
149  P↔S RCL O P↔S + STO 6
154  DSZ I GTO 3
156  RCL 4 P↔S STO O P↔S
160  RCL 6 STO 4 B STO 7
164  RTN
165  LBL B
```

$v \leftarrow u$

$x \leftarrow x + h$

$w \leftarrow w+\frac{h}{3}(4f_0+f_{-1})$ und $f_{-2}\leftarrow f_{-1}, f_{-1}\leftarrow f_0$

$s \leftarrow 4$

$R_x \leftarrow \frac{h}{3}f(x,v)$

$v \leftarrow w + \frac{h}{3}f(x,v)$

$s \leftarrow s - 1$, wenn $s>0$

$w \leftarrow u$

$u \leftarrow v$, $f_0 \leftarrow f(x,v)$

Rücksprung

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	R_{07}	R_{08}	R_{09}	R_{10}
h	i	j	n	u	x	v	f_0	f_{-1}	f_{-2}	w
							c		s	

Hauptprogramm

```
000   Lbl A RCL 0 Prt RCL 5 Prt        Anzeige h, Anzeige x=a
008   RCL 4 Prt                        Anzeige u=α
011   A' 1 STO 1                       Anlaufrechnung, i ← 1
015   Lbl INV D                        u←u_h(x+h),x←x+h  ←
018   Op 21 RCL 1 ÷ RCL 3 =            i←i+1, R_X←i/n
026   INV Int CP INV x=t lnx           wenn [1/n] ≠ 1/n  ——
032   RCL 5 Prt RCL 4 Prt              Anzeige x, Anzeige u
038   Lbl lnx RCL 2 x↔t RCL 1          R_T ← j, R_X ← i  ←
045   INV x≥t INV                      wenn i<j  ————————
048   R/S                              Stop
```

Unterprogramm Anlaufrechnung

```
049   Lbl A' RCL 4 STO 10 STO 6        w ← u, v ← u
057   B STO 08 1 +/- Prd 0             f_-1 ← f(x,v), h ← -h
064   E                                u ← u_h(a-h), x ← a-h
065   1 +/- Prd 0 RCL 4 STO 6          h ← -h, v ← u
073   B STO 9 RCL 10 STO 4             f_-2 ← f(x,v), u ← w
080   RCL 0 SUM 5                      x ← x + h
084   E                                u ← u_h(a+h), x ← a+h
085   RCL 3 x↔t 1 INV x=t CE           wenn n≠1  ————————
092   RCL 5 Prt RCL 4 Prt              Anzeige x, Anzeige u
098   Lbl CE RCL 4 STO 6 B STO 7       v ← u, f_0 ← f(x,v)  ←
107   INV SBR                          Rücksprung
```

Unterprogramm klassisches Runge-Kutta-Verfahren

108	Lbl E RCL 4 STO 6	$v \leftarrow u$
114	B * RCL O = STO 7	$k \leftarrow hf(x,v)$ $c \leftarrow k$
121	÷ 2 = SUM 6 RCL O ÷ 2 = SUM 5	$v \leftarrow v+\frac{1}{2}k$, $x \leftarrow x+\frac{1}{2}h$
133	B * RCL O = SUM 7 SUM 7	$k \leftarrow hf(x,v)$, $c \leftarrow c + 2k$
142	÷ 2 + RCL 4 = STO 6	$v \leftarrow u + \frac{1}{2}k$
150	B * RCL O = SUM 7 SUM 7	$k \leftarrow hf(x,v)$, $c \leftarrow c + 2k$
159	+ RCL 4 = STO 6	$v \leftarrow u + k$
165	RCL O ÷ 2 = SUM 5	$x \leftarrow x + \frac{h}{2}$
172	B * RCL O + RCL 7 =	$k \leftarrow hf(x,v)$, $R_X \leftarrow c+k$
180	÷ 6 = SUM 4	$u \leftarrow u + \frac{1}{6}(c+k)$
185	INV SBR	Rücksprung
186	Lbl D	

Unterprogramme für die speziellen Mehrschrittverfahren

Unterprogramm Extrapolationsverfahren von Adams-Bashforth

186	Lbl D 23 * RCL 7 -	$R_X \leftarrow 23f_0$
194	16 * RCL 8 + 5 * RCL 9 =	$R_X \leftarrow 23f_0-16f_{-1}+5f_{-2}$
205	* RCL O ÷ 12 = SUM 4	$u \leftarrow u+\frac{h}{12}(23f_0-16f_{-1}+5f_{-2})$
214	RCL 4 STO 6 RCL O SUM 5	$v \leftarrow u$, $x \leftarrow x + h$
222	B Exc 7 Exc 8 STO 9	$f_{-2} \leftarrow f_{-1}, f_{-1} \leftarrow f_0, f_0 \leftarrow f(x,v)$
229	INV SBR	Rücksprung
230	Lbl B	

Unterprogramm Prädiktor-Korrektor-Verfahren von Adams

186	Lbl D 23 * RCL 7 -	$R_X \leftarrow 23f_0$
194	16 * RCL 8 + 5 * RCL 9 =	$R_X \leftarrow 23f_0-16f_{-1}+5f_{-2}$
205	* RCL O ÷ 12 + RCL 4 = STO 6	$v \leftarrow u+\frac{h}{12}(23f_0-16f_{-1}+5f_{-2})$
217	RCL O SUM 5	$x \leftarrow x + h$
221	B * 9 + 19 * RCL 7 -	$R_X \leftarrow 9f(x,v) + 19f_0$
231	5 * RCL 8 + RCL 9 =	$R_X \leftarrow 9f(x,v)+19f_0-5f_{-1}+f_{-2}$
239	* RCL O ÷ 24 = SUM 4	$u \leftarrow u+\frac{h}{24}(9f(x,v)+\ldots)$
248	RCL 4 STO 6	$v \leftarrow u$
252	B Exc 7 Exc 8 STO 9	$f_{-2} \leftarrow f_{-1}, f_{-1} \leftarrow f_0, f_0 \leftarrow f(x,v)$
259	INV SBR	Rücksprung
260	Lbl B	

Unterprogramm Interpolationsverfahren von Adams-Moulton

186	Lbl D RCL 4 STO 6 RCL 0 SUM 05	$v \leftarrow u$, $x \leftarrow x + h$
196	19 * RCL 7 - 5 * RCL 8 +	$R_X \leftarrow 19f_0 - 5f_{-1}$
207	RCL 9 = * RCL 0 ÷ 24 = SUM 04	$u \leftarrow u + \frac{h}{24}(19f_0 - 5f_{-1} + f_{-2})$
219	4 STO 9	$s \leftarrow 4$
222	Lbl CLR B * RCL 0 * 9 ÷ 24 +	$R_X \leftarrow \frac{9}{24}hf(x,v)$ ⟵
234	RCL 4 = STO 6	$v \leftarrow u + \frac{9}{24}hf(x,v)$
239	Dsz 9 CLR	$s \leftarrow s - 1$, wenn $s>0$ ⟵
242	STO 4	$u \leftarrow v$
244	B Exc 7 Exc 8 STO 9	$f_{-2} \leftarrow f_{-1}, f_{-1} \leftarrow f_0, f_0 \leftarrow f(x,v)$
251	INV SBR	Rücksprung
252	Lbl B	

Unterprogramm Extrapolationsverfahren von Nyström

186	Lbl D 7 * RCL 7 - 2 * RCL 8 +	$R_X \leftarrow 7f_0 - 2f_{-1}$
198	RCL 9 = * RCL 0 ÷ 3 +	$R_X \leftarrow \frac{h}{3}(7f_0 - 2f_{-1} + f_{-2})$
207	RCL 10 = STO 6	$v \leftarrow w + \frac{h}{3}(7f_0 - 2f_{-1} + f_{-2})$
212	Exc 4 STO 10 RCL 0 SUM 5	$w \leftarrow u$, $u \leftarrow v$, $x \leftarrow x+h$
220	B Exc 7 Exc 8 STO 9	$f_{-2} \leftarrow f_{-1}, f_{-1} \leftarrow f_0, f_0 \leftarrow f(x,v)$
227	INV SBR	Rücksprung
228	Lbl B	

Unterprogramm Prädiktor-Korrektor-Verfahren von Milne

186	Lbl D 7 * RCL 7 - 2 * RCL 8 +	$R_X \leftarrow 7f_0 - 2f_{-1}$
198	RCL 9 = * RCL 0 ÷ 3 +	$R_X \leftarrow \frac{h}{3}(7f_0 - 2f_{-1} + f_{-2})$
207	RCL 10 = STO 6	$v \leftarrow w + \frac{h}{3}(7f_0 - 2f_{-1} + f_{-2})$
212	RCL 0 SUM 5	$x \leftarrow x + h$
216	B + 4 * RCL 7 + RCL 8 =	$R_X \leftarrow f(x,v) + 4f_0 + f_{-1}$
226	* RCL 0 ÷ 3 + RCL 10 = STO 6	$v \leftarrow w + \frac{h}{3}(f(x,v) + 4f_0 + f_{-1})$
237	Exc 4 STO 10	$w \leftarrow u$, $u \leftarrow v$
241	B Exc 7 Exc 8 STO 9	$f_{-2} \leftarrow f_{-1}, f_{-1} \leftarrow f_0, f_0 \leftarrow f(x,v)$
248	INV SBR	Rücksprung
249	Lbl B	

Unterprogramm Interpolationsverfahren von Milne-Simpson

186	Lbl D RCL 4 STO 6 RCL O SUM O5	$v \leftarrow u,\ x \leftarrow x + h$
196	4 * RCL 7 + RCL 8 =	$R_X \leftarrow 4f_O + f_{-1}$
204	* RCL O ÷ 3 = SUM 1O	$w \leftarrow w+\frac{h}{3}(4f_O+f_{-1})$
212	4 STO 9	$s \leftarrow 4$
215	Lbl CLR B * RCL O ÷ 3 +	$R_X \leftarrow \frac{h}{3}f(x,v)$
224	RCL 1O = STO 6	$v \leftarrow w + \frac{h}{3}f(x,v)$
229	Dsz 9 CLR	$s \leftarrow s - 1$, wenn $s>0$
232	Exc 4 STO 1O	$w \leftarrow u,\ u \leftarrow v$
236	B Exc 7 Exc 8 STO 9	$f_{-2}+f_{-1}, f_{-1}+f_O, f_O+f(x,v)$
243	INV SBR	Rücksprung
244	Lbl B	

Anleitung zur Verwendung der Programme

1. Eingabe des Hauptprogramms mit Anlaufrechnung und klassischem Runge-Kutta-Verfahren sowie Eingabe eines Unterprogramms für ein Mehrschrittverfahren.
2. Die weiteren Punkte aus der Anleitung zur Verwendung der Programme für die Einschrittverfahren gelten unverändert.

Beispiele

(1) Wir betrachten wieder die Anfangswertaufgabe $u'(x) = 1+u(x)^2$, $0 \le x < \frac{\pi}{2}$, $u(0) = 0$, mit der exakten Lösung $u(x) = \tan x$. Hierzu wurden Näherungswerte ermittelt mit den Verfahren von Adams. Mit der Schrittweite h = 0.05 und mit j = 30, n = 2, a = 0, α = 0 wurden Funktionswerte der Näherungslösungen berechnet für x = 0(0.05)1.5 und angezeigt für x = 0(0.1)1.5. Das Unterprogramm B zur Berechnung von f(x,v) lautet

für die Rechner HP-67/97

LBL B RCL 6 x^2 1 + RTN $\quad R_X \leftarrow 1+v^2$, Rücksprung,

für die Rechner TI-58/59

Lbl B (1 + RCL 6 x^2) INV SBR $\quad R_X \leftarrow 1+v^2$, Rücksprung.

Mit veränderter Ausgabeanweisung wurden die folgenden Ergebnisse erhalten.

x	Extrap.	Präd.-Korr.	Interp.	tan x
0.0	0.000000000	0.000000000	0.000000000	0.000000000
0.1	0.100333859	0.100334600	0.100334804	0.100334672
0.2	0.202716!616	0.202710414	0.202710471	0.202710036
0.3	0.309310362	0.309336852	0.309337096	0.309336250
0.4	0.422735526	0.422793986	0.422794691	0.422793219
0.5	0.546185579	0.546303276	0.546304995	0.546302490
0.6	0.683927855	0.684137250	0.684141128	0.684136808
0.7	0.841907926	0.842287576	0.842296084	0.842288381
0.8	1.028937690	1.029634203	1.029653009	1.029638557
0.9	1.258621235	1.260144153	1.260187278	1.260158218
1.0	1.554700239	1.557365872	1.557471934	1.557407725
1.1	1.956751799	1.964629824	1.964921419	1.964759657
1.2	2.556832629	2.571690324	2.572645453	2.572151622
1.3	3.553374700	3.599570195	3.604148126	3.602102448
1.4	5.568624674	5.781328858	5.812698964	5.797883715
1.5	11.48457346	13.62892399	14.52960659	14.10141995

(2) Die Anfangswertaufgabe $u'(x) = u(x) - \frac{2x}{u(x)}$, $0 \le x \le 1.5$, $u(0) = 1$ besitzt die exakte Lösung $u(x) = \sqrt{2x+1}$. Mit den Verfahren von Nyström und Milne wurden Näherungswerte berechnet mit h = 0.1, j = 15, n = 1, a = 0, α = 1. Das Unterprogramm B, mit dem f(x,v) berechnet wird, lautet

für die Rechner HP-67/97

LBL B RCL 6 RCL 5 2 * RCL 6 ÷ - RTN $\quad R_X \leftarrow v - \frac{2x}{v}$ Rücksprung,

für die Rechner TI-58/59

Lbl B (RCL 6 - 2 * RCL 5 ÷ RCL 6) INV SBR $\quad R_X \leftarrow v - \frac{2x}{v}$ Rücksprung.

x	Extrap.	Präd.-Korr.	Interp.	$\sqrt{2x+1}$
0.0	1.000000000	1.000000000	1.000000000	1.000000000
0.1	1.095445532	1.095445532	1.095445532	1.095445115
0.2	1.183604453	1.183237975	1.183221269	1.183215957
0.3	1.265239971	1.264927938	1.264914661	1.264911064
0.4	1.342228360	1.341675615	1.341648424	1.341640786
0.5	1.414808036	1.414244212	1.414219714	1.414213562
0.6	1.484046340	1.483287860	1.483249490	1.483239697
0.7	1.550086477	1.549240269	1.549202126	1.549193338
0.8	1.613564244	1.612517187	1.612464095	1.612451550
0.9	1.674600532	1.673388676	1.673332512	1.673320053
1.0	1.733597276	1.732140853	1.732067203	1.732050808
1.1	1.790667985	1.788953152	1.788871455	1.788854382
1.2	1.846077179	1.844033751	1.843930796	1.843908891
1.3	1.899932544	1.897507966	1.897390476	1.897366596
1.4	1.952419517	1.949533714	1.949386712	1.949358869
1.5	2.003639474	2.000202175	2.000033591	2.000000000

8. Rechnen mit Vektoren und Matrizen

Die Algorithmen der nachfolgenden Kapitel werden mit Hilfe mehrerer Vektoren formuliert. In den zugehörigen Taschenrechnerprogrammen ist es erforderlich, dementsprechend mehrere Zahlenfelder zu speichern. Daher stellen wir jetzt Überlegungen zum Speichern von Vektoren und Matrizen an.

Im Zusammenhang mit Berechnungen bei Vektoren und Matrizen treten in den Algorithmen häufig Schleifen auf. Wir gehen deshalb hier auch darauf ein, wie wir diese Schleifen in den Taschenrechnerprogrammen steuern mit Hilfe der komplementären Schleifenindizes und der Dsz-Anweisung.

In manchen Algorithmen der vorangehenden Kapitel trat bereits ein Vektor $(x_1,\ldots,x_n)$ auf. Dabei war auch die Anzahl n der Vektorkomponenten eine Variable, die nicht durch das Programm festgelegt war sondern eingegeben werden konnte.

In Kapitel 1 speicherten wir $x_1,\ldots,x_n$ in den Datenregistern $R_1,\ldots,R_n$, so daß die Adresse des Datenspeicherplatzes für die Komponente x_j des Vektors bestimmt war durch

$$\text{Adr } x_j = j, \quad j=1,\ldots,n.$$

Dadurch gestaltete sich das indirekte Aufrufen und Speichern dieser Größen sehr einfach.

Hätten wir diese Speicherung der Vektorkomponenten auch in den Kapiteln 2 bis 5 beibehalten, so wäre es erforderlich geworden, die weiteren auftretenden Variablen zum Teil im Anschluß an R_n zu speichern. Dann wären entweder alle diese Variablen nur über indirekte Adressierung zugänglich gewesen oder bei direkter Adressierung dieser Variablen hätte ihre Belegung des Datenspeichers eine Begrenzung für n nach sich gezogen.

Wir vermieden dies, indem wir in den Kapiteln 2 und 3 die Vektorkomponenten erst im Anschluß an die anderen Variablen speicherten. In den Kapiteln 4 und 5 war das jedoch nicht möglich, denn dort wurden zu einem gemeinsamen Hauptprogramm

verschiedene Unterprogramme mit unterschiedlichem Speicherplatzbedarf verwendet; deshalb speicherten wir dort den auftretenden Vektor vom Ende des Datenspeichers aus, während die anderen Variablen die Datenspeicherplätze vom Anfang her belegten. Dieses Vorgehen ermöglicht es auch, das zugehörige Programm nachträglich durch Verwendung weiterer Größen zu ergänzen, ohne daß die Speicherbelegung und die Adressierung der Vektorkomponenten verändert werden muß.

Aus dem zuletzt geschilderten Grund entscheiden wir uns im folgenden dafür, Vektoren vom Ende des verfügbaren Datenspeichers her zu speichern. Da die Komponenten x_j in aufeinanderfolgenden Datenregistern plaziert werden sollen, gilt dann

$$\text{Adr } x_j = \text{Adr } x_1 - (j - 1), \quad j=1,\ldots,n.$$

Matrizen $(a_{ik})_{\substack{i=1,\ldots,m \\ k=1,\ldots,n}}$ speichern wir entsprechend, indem wir sie als Vektoren (a_j) auffassen durch spaltenweise Anordnung der Matrizenelemente,

$$a_j = a_{ik} \text{ mit } j = i + (k - 1)m$$

für $i=1,\ldots,m$, $k=1,\ldots,n$. Die Funktion j von i und k nennen wir die Speicherabbildungsfunktion. Die Matrix (a_{ik}) wird daher auf mn Speicherplätzen mit den Adressen

$$\text{Adr } a_{ik} = \text{Adr } a_{11} - (i - 1 + (k - 1)m)$$

für $i=1,\ldots,m$, $k=1,\ldots,n$ gespeichert.

Bezeichnen wir mit ω die Adresse des letzten Datenspeicherplatzes und wählen wir Adr $a_{11} = \omega$, so wird also

(1) $$\text{Adr } a_{ik} = \omega - (i - 1 + (k - 1)m).$$

Sind außer der Matrix (a_{ik}) etwa noch die Vektoren $(b_i)_{i=1,\ldots,m}$ und $(c_k)_{k=1,\ldots,n}$ zu speichern, so kann man wählen

$$\text{Adr } b_1 = \text{Adr } a_{mn} - 1 = \omega - mn,$$

so daß gilt

(2) $\quad \text{Adr } b_i = \omega - mn - (i - 1),$

sowie

$$\text{Adr } c_1 = \text{Adr } b_m - 1 = \omega - m(n + 1)$$

und damit

(3) $\quad \text{Adr } c_k = \omega - m(n + 1) - (k - 1).$

Für symmetrische Matrizen gilt $a_{ik} = a_{ki}$, $i,k=1,\ldots,n$. Daher ist das Zahlenfeld bereits durch die Elemente in den Spalten ab der Hauptdiagonalen bestimmt,

$$\begin{array}{ccccc} a_{11} & & & & \\ a_{21} & a_{22} & & & \\ a_{31} & a_{32} & a_{33} & & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{array}$$

und es genügt, nur diese Elemente zu speichern. Im Vergleich zur Speicherabbildungsfunktion der vollen Matrix entfallen hier in der s-ten Spalte jeweils s-1 Elemente über der Hauptdiagonalen,

$$j = i + (k - 1)n - \sum_{s=1}^{k} (s - 1).$$

Folglich wird die Speicherabbildungsfunktion für die Teilmatrix $(a_{ik})_{1\leq k\leq i\leq n}$

$$j = i + \frac{1}{2}(k - 1)(2n - k);$$

für die zugehörigen Adressen gilt

(4) $\quad \text{Adr } a_{ik} = \text{Adr } a_{11} - (i - 1 + \frac{1}{2}(k - 1)(2n - k)), \quad 1\leq k\leq i\leq n.$

In den Programmen der weiteren Abschnitte speichern wir die Zahlenfelder in der beschriebenen Art. Die Adressen der einzelnen Elemente lassen wir jeweils durch Unterprogramme berechnen und nicht durch konkrete Bezugnahme auf die Plazierung der Elemente im Datenspeicher. Dadurch werden die

Hauptprogramme nicht durch die Berechnung von Adressen belastet, und es wird gewährleistet, daß eine andere Speicherung der Zahlenfelder bereits allein durch Anpassen der zugehörigen Adressenunterprogramme berücksichtigt werden kann.

Bei Rechnungen mit Vektoren und Matrizen treten immer wieder Programmschleifen j = 1(1)n auf. Wir haben schon mehrmals solche Schleifen rückwärts in der Form j = n(-1)1 durchlaufen und dabei am Ende des Wiederholungsbereichs der Schleife mit Hilfe der Dsz-Anweisung den Schleifenparameter j vermindert und die zugehörige Verzweigung überwacht. Manchmal ist es jedoch erforderlich, die natürliche Reihenfolge j = 1(1)n bei einer Schleife einzuhalten, etwa wenn Vektorkomponenten x_j in der Schleife berechnet und in der natürlichen Reihenfolge angezeigt werden sollen. Die Dsz-Anweisung läßt sich in diesem Fall trotzdem anwenden, wenn die Schleife mit Hilfe des komplementären Index j' = n + 1 - j gesteuert wird. Die beiden Schleifen

```
j = 1(1)n                    j' = n(-1)1
|Berechnung und             |Berechnung und
|Anzeige von x_j             |Anzeige von x_(n+1-j')
```

sind gleichwertig, auch hinsichtlich der Reihenfolge der angezeigten Werte.

In den komplementären Indizes i' = m + 1 - i, k' = n + 1 - k lauten dann die Adressen

(1') Adr a_{ik} = ω - m(n+1) + i' + k'm
(2') Adr b_i = ω - m(n+1) + i'
(3') Adr c_k = ω - m(n+1) - n + k'

für i=1,...,m, k=1,...,n. Für Teilmatrizen $(a_{ik})_{1\leq k\leq i\leq n}$ sind mit m = n die Adressen

(4') Adr a_{ik} = Adr a_{11} - $\frac{n}{2}(n+1)$ + i' + $\frac{1}{2}$k'(k'-1)

für $1\leq k\leq i\leq n$ und $1\leq i'\leq k'\leq n$.

9. Lineare Gleichungssysteme

Wir betrachten in diesem Kapitel lineare Gleichungssysteme mit n Gleichungen für n Unbekannte in der Form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n. \end{aligned}$$

Mit der Koeffizientenmatrix $A = (a_{ik})_{i,k=1,\ldots,n}$ und dem Vektor $b = (b_1,\ldots,b_n)$ der rechten Seiten lautet das Gleichungssystem für den Vektor $x = (x_1,\ldots,x_n)$ der Unbekannten dann

$$Ax = b.$$

Die Methoden zur Lösung solcher linearer Gleichungssysteme stellen ein umfangreiches Gebiet der Numerischen Mathematik dar. Wir beschäftigen uns hier mit dem Gaußschen Eliminationsverfahren, das zu den direkten Lösungsmethoden gehört; bei ihnen wir die Lösung in endlich vielen Schritten ermittelt. Im Unterschied dazu liefern iterative Verfahren die Lösung als Grenzwert einer unendlichen Folge von Näherungsvektoren; zu diesen Methoden gehört das anschließend behandelte Einzelschrittverfahren mit dem zugehörigen Relaxationsverfahren. Zuvor befassen wir uns noch mit der Berechnung von Defektvektoren bei linearen Gleichungssystemen, um die Überlegungen des vorherigen Kapitels über das Rechnen mit Vektoren und Matrizen zuerst an einem einfachen Beispiel anzuwenden.

9.1. Defektvektoren

Ist $A = (a_{ik})_{i,k=1,\ldots,n}$ eine invertierbare Matrix, so ist für jeden Vektor $b = (b_i)_{i=1,\ldots,n}$ das lineare Gleichungssystem $Ax = b$ eindeutig lösbar. Kennt man eine Näherung c

für die exakte Lösung x des Gleichungssystems, so gibt der Defektvektor $d = Ac - b$ an, bis auf welche Fehler die Gleichungen des Gleichungssystems erfüllt sind. Mit $d = A(c - x)$ folgt für den Unterschied zwischen der exakten Lösung x und der Näherung c die Gleichung $c - x = A^{-1}d$, mit der inversen Matrix A^{-1}. Daher kann in der Regel die Größe des Defektvektors als ein Maß für den absoluten Fehler der Näherungslösung c angesehen werden.

Wir betrachten im folgenden zu einer Norm (Betragsfunktion) $||y||$ für Vektoren eine dazu gehördende Norm $||B||$ für Matrizen, die die Eigenschaft besitzt, daß für beliebige Vektoren y und beliebige Matrizen B die Ungleichung $||By|| \leq ||B|| \cdot ||y||$ gilt (mit der Vektornorm verträgliche Matrizennorm). Für den absoluten Fehler c - x folgt dann

$$||c - x|| \leq ||A^{-1}|| \cdot ||d||.$$

Mit der Beziehung $||b|| \leq ||A|| \cdot ||x||$ gilt daher noch die Abschätzung des relativen Fehlers

$$\frac{||c - x||}{||x||} \leq ||A|| \cdot ||A^{-1}|| \cdot \frac{||d||}{||b||} .$$

Beispiele für Vektornormen sind die Betragssummennorm $||x||_1 = \sum_{i=1}^{n} |x_i|$, die euklidische Norm $||x||_2 = (\sum_{i=1}^{n} x_i^2)^{1/2}$ und die Maximumnorm $||x||_\infty = \max_{i=1,\ldots,n} |x_i|$.

Dazu gehörende verträgliche Matrizennormen sind die maximale Spaltensumme $||B||_1 = \max_{k=1,\ldots,n} \sum_{i=1}^{n} |b_{ik}|$ bzw. die Quadratsummennorm $||B||_2 = (\sum_{i,k=1}^{n} b_{ik}^2)^{1/2}$ bzw. die maximale Zeilensumme $||B||_\infty = \max_{i=1,\ldots,n} \sum_{k=1}^{n} |b_{ik}|$.

In dem folgenden Algorithmus berechnen wir den Defektvektor $d = (d_1,\ldots,d_n)$ und die Defektnorm $||d||_\infty$ gemäß

$$d_i = \sum_{k=1}^{n} a_{ik}c_k - b_i, \quad i = 1,\ldots,n,$$

$$||d||_\infty = \max_{i=1,\ldots,n} |d_i|.$$

Algorithmus Eingabe: n, (a_{ik}), (b_i), (c_k)
Anzeige: $d_1, d_2,\ldots,d_n$; $||d||_\infty$.

a_{ik}	b_i	c_k	d	i	k	n	s

```
s ← 0
i = 1(1)n
  d ← 0
  k = 1(1)n
    d ← d + a_ik c_k
  d ← d - b_i
  Anzeige d
  s ← max(s,|d|)
Anzeige s
Stop
```

Soll an Stelle der Defektnorm $||d||_\infty$ die Zahl $||d||_1$ angezeigt werden, so ist die Anweisung s←max(s,|d|)zu ersetzen durch s←s+|d|. Die Defektnorm $||d||_2$ wird angezeigt, falls statt s←max(s,|d|) die Anweisung $s \leftarrow s+d^2$ gewählt wird und falls außerdem die letzte Anweisung vor Stop lautet: Anzeige $\sqrt{s}$.

Für das Speichern der Matrix und der Vektoren wenden wir die Überlegungen des vorherigen Kapitels an. Mit der spaltenweisen Speicherung der Matrix wird wegen m = n nun

$$\begin{aligned} \text{Adr } a_{ik} &= \omega - (i-1 + (k-1)n) &&= i' - (n+1 - k')n + \omega, \\ \text{Adr } b_i &= \omega - n^2 - (i-1) &&= i' - (n+1)n + \omega, \\ \text{Adr } c_k &= \omega - n(n+1) - (k-1) &&= k' - (n+2)n + \omega, \end{aligned}$$

für i,k = 1,...,n und mit den komplementären Indizes $i' = n + 1 - i$, $k' = n + 1 - k$.

Ist die Koeffizientenmatrix A symmetrisch und entscheiden wir uns dafür, nur die Teilmatrix $(a_{ik})_{k\le i}$ zu speichern, so wird

$$\begin{aligned}
\text{Adr } a_{ik} &= \omega - (i - 1 + \tfrac{1}{2}(k - 1)(2n - k)) , && k\le i,\\
&= \tfrac{1}{2}k'(k' - 1) + i' - (n + 1)\tfrac{n}{2} + \omega, && k'\ge i',\\
\text{Adr } b_i &= \omega - \tfrac{1}{2}n(n + 1) - (i - 1) = i' - (n + 3)\tfrac{n}{2} + \omega,\\
\text{Adr } c_k &= \omega - \tfrac{1}{2}n(n + 3) - (k - 1) = k' - (n + 5)\tfrac{n}{2} + \omega.
\end{aligned}$$

Programm für HP-67/97

Belegung des Datenspeichers bei Speicherung der vollen Matrix

R_0	R_1	R_2	R_3	R_4		$R_{25-(n+2)n}$	...	$R_{24-(n+1)n}$
d	i'	k'	n	s		c_n	...	c_1

$R_{25-(n+1)n}$	...	R_{24-n^2}	R_{25-n^2}	...	R_{23}	R_{24}	R_I
b_n	...	b_1	a_{nn}	...	a_{21}	a_{11}	Adr

$n\le 3$

Belegung des Datenspeichers bei Speicherung der Teilmatrix $(a_{ik})_{k\le i}$

R_0	R_1	R_2	R_3	R_4		$R_{25-(n+5)n/2}$	...	$R_{24-(n+3)n/2}$
d	i'	k'	n	s		c_n	...	c_1

$n\le 4$

$R_{25-(n+3)n/2}$	...	$R_{24-(n+1)n/2}$	$R_{25-(n+1)n/2}$	...	R_{23}	R_{24}	R_I
b_n	...	b_1	a_{nn}	...	a_{21}	a_{11}	Adr

Einleseprogramme

```
001   LBL A RCL 3 STO 1 STO 2        i' ← n, k' ← n
005   a GTO 0                        R_I ← Adr a_11,
007   LBL B RCL 3 STO 1 b GTO 0      i' ← n, R_I ← Adr b_1,
012   LBL C RCL 3 STO 2 c            k' ← n, R_I ← Adr c_1,
016   LBL 0 I R/S                    Anzeigen der Adresse, Stop
019   STO (i) DSZ I GTO 0            Speichern, Adr ← Adr - 1,
```

Hauptprogramm

```
022  LBL D 0 STO 4 RCL 3 STO 1        s ← 0, i' ← n
027  LBL 1 RCL 3 STO 2 0 STO 0        k' ← n, d ← 0          ←
032  LBL 2 a RCL (i)                  R_X ← a_ik             ←
035  c CLx RCL (i) * STO + 0          d ← d + a_ik c_k
040  2 STO I DSZ (i) GTO 2            k' ← k' - 1, wenn k'>0 ┘
044  b RCL (i) STO - 0 RCL 0 PRTx     d ← d - b_i, Anzeige d
049  ABS RCL 4 x≤y? x↔y STO 4         s ← max(s,|d|)
054  1 STO I DSZ (i) GTO 1            i' ← i' - 1, wenn i'>0 ┘
058  PRTSPC RCL 4 PRTx                Anzeige s
061  R/S                              Stop
```

Adressenunterprogramme bei Speicherung der vollen Matrix

```
062  LBL a RCL 1 RCL 2 CHS GTO 3      i'-(-k' ──→ Adr a_ik
067  LBL b RCL 1 0 GTO 3              i'-(0   ──→ Adr b_i
071  LBL c RCL 2 1                    k'-(1       Adr c_k
074  LBL 3 1 + RCL 3 +                +1+n)  ←┘
079  RCL 3 * - 24 + STO I             ·n+24, R_I ← Adr
086  RTN                              Rücksprung
```

Adressenunterprogramme bei Speicherung der Teilmatrix $(a_{ik})_{k\leq i}$

```
062  LBL a RCL 1 RCL 2 x≤y? x↔y       wenn k'≤i', dann R_X↔R_Y
067  ENTER ENTER 1 - * 2 ÷            k'(k'-1)/2
074  + 1 GTO 3                        +i'-(1 ──→ Adr a_ik
077  LBL b RCL 1 3 GTO 3              i'-(3  ──→ Adr b_i
081  LBL c RCL 2 5                    k'-(5      Adr c_k
084  LBL 3 RCL 3 + RCL 3 * 2 ÷ -      +n)n/2 ←┘
092  24 + STO I                       +24, R_I ← Adr
096  RTN                              Rücksprung
```

Programm für TI-58/59

Belegung des Datenspeichers bei Speicherung der vollen Matrix

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}		$R_{\omega-(n+2)n+1}$	$R_{\omega-(n+1)n}$
Adr	i'	k'	n	s	ω		c_n	c_1

$R_{\omega-(n+1)n+1}$	···	$R_{\omega-n^2}$	$R_{\omega-n^2+1}$	···	$R_{\omega-1}$	R_ω
b_n	···	b_1	a_{nn}	···	a_{21}	a_{11}

Belegung des Datenspeichers bei Speicherung der Teilmatrix $(a_{ik})_{k\le i}$

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}		$R_{\omega-(n+5)n/2+1}$	...	$R_{\omega-(n+3)n/2}$
Adr	i'	k'	n	s	ω		c_n	...	c_1

$R_{\omega-(n+3)n/2+1}$	...	$R_{\omega-(n+1)n/2}$	$R_{\omega-(n+1)n/2+1}$	...	$R_{\omega-1}$	R_{ω}
b_n	...	b_1	a_{nn}	...	a_{21}	a_{11}

Einleseprogramme

```
000   STO 3                              R03 ← n
002   Op 16 INV Int * 100 = STO 5        R05 ← ω
013   R/S                                Stop
014   Lbl A RCL 3 STO 1 STO 2            i' ← n, k' ← n
022   A' GTO INV                         R00 ← Adr a11' ──────────┐
025   Lbl B RCL 3 STO 1 B' GTO INV       i' ← n, R00 ← Adr b1' ──┐│
034   Lbl C RCL 3 STO 2 C'               k' ← n, R00 ← Adr c1    ││
041   Lbl INV RCL 0 R/S                  Anzeigen der Adresse,Stop◄┘
046   STO Ind 0 Dsz 0 INV                Speichern, Adr←Adr-1, ──┘
```

Hauptprogramm

```
051   Lbl D 0 STO 4 RCL 3 STO 1          s ← 0, i' ← n
060   Lbl lnx RCL 3 STO 02 0             k' ← n, d ← 0 ◄──────┐
067   Lbl CE + A' RCL Ind 0 *          } d ← d + a_ik c_k ◄──┐│
074   C' RCL Ind 0                     }                     ││
077   Dsz 2 CE                           k' ← k'-1, wenn k'>0 ─┘│
080   - B' RCL Ind 0 = Prt               d ← d-b_i, Anzeige d   │
086   |x| x↔t RCL 4 x≥t CLR            } s ← max(s,|d|)         │
092   x↔t STO 4                        }                        │
095   Lbl CLR Dsz 1 lnx                  i'←i'-1, wenn i'>0 ─────┘
100   Adv RCL 4 Prt                      Anzeige s
104   R/S                                Stop
```

Adressenunterprogramme bei Speicherung der vollen Matrix

```
105  Lbl A' ( RCL 1 -                  i'-
111  ( RCL 2 +/- + GTO x↔t             ( - k' + → Adr a_ik
118  Lbl B' ( RCL 1 - ( GTO x↔t        i' - (   → Adr b_i
127  Lbl C' ( RCL 2 - ( 1 +            k' - (1+   Adr c_k
136  Lbl x↔t 1 + RCL 3 )               1 + n)  ←┘
143  * RCL 3 + RCL 5 ) STO 0           ·n + ω, R_00 ← Adr
152  INV SBR                           Rücksprung
```

Adressenunterprogramme bei Speicherung der Teilmatrix $(a_{ik})_{k \leq i}$

```
105  Lbl A' ( RCL 1 x↔t RCL 2          R_X ← k', R_T ← i'
113  x≥t x↔t                           wenn k'≥i' ──┐
115  x↔t                               R_X↔R_T      │
116  Lbl x↔t * ( CE - 1 ) ÷ 2 +        k'(k' - 1)/2 ←┘
127  x↔t - ( 1 GTO x²                  +i' -(1 → Adr a_ik
133  Lbl B' ( RCL 1 - ( 3 GTO x²       i' -(3  → Adr b_i
143  Lbl C' ( RCL 2 - ( 5              k' -(5    Adr c_k
151  Lbl x² + RCL 3 ) * RCL 3 ÷ 2 +    +n)n/2  ←┘
163  RCL 5 ) STO 0                     +ω, R_00 ← Adr
168  INV SBR                           Rücksprung
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms (Einleseprogramme, Hauptprogramm und eine Art der Adressenunterprogramme).
2. Eingabe von n
   bei HP-67/97: n STO 3,
   bei TI-58/59: n RST R/S.
3. Eingabe der Matrix A (spaltenweise) und der Vektoren b und c:

   A a_{11} R/S a_{21} R/S ... a_{nn} R/S
   b_1 R/S b_2 R/S ... b_n R/S
   c_1 R/S c_2 R/S ... c_n R/S.

   Gleichwertig zu Schritt 3 können die Schritte 3a, 3b, 3c einzeln und unabhängig voneinander ausgeführt werden.

3a Eingabe der Matrix A (spaltenweise):
A a_{11} R/S a_{21} R/S ... a_{nn} R/S.

3b Eingabe des Vektors b:
B b_1 R/S b_2 R/S ... b_n R/S.

3c Eingabe des Vektors c:
C c_1 R/S c_2 R/S ... c_n R/S.

4. Berechnung und Anzeige der Komponenten $d_1,\ldots,d_n$ des Defektvektors d und der Defektnorm $||d||_\infty$: D.

Mögliche Wiederholungen:
ab Schritt 3c bzw. 3b bzw. 3a für einen anderen n-komponentigen Vektor c, b bzw. für eine andere nxn-Matrix A;
ab Schritt 2 für eine andere Wahl von n.

In den Schritten 3 sowie 3a, 3b und 3c wird jeweils vor dem Eintasten eines Elements die Adresse des Datenspeicherplatzes angezeigt, auf den dieser Wert dann durch R/S gespeichert wird; danach wird die Adresse des nächsten Datenspeicherplatzes angezeigt.

In Schritt 2 der Anleitung zur Verwendung des Programms für die Rechner TI-58/59 wird n gespeichert und gleichzeitig auch die Datenspeicherverteilung durch Ermitteln des letzten Datenspeicherplatzes festgelegt. Ändert man die Aufteilung der Speicherregister zwischen Programmspeicher und Datenspeicher, so wird dies bei Wiederholung ab Schritt 2 ebenfalls berücksichtigt.

In dem Programm für die TI-Rechner wird die Größe d zur Berechnung der Summe für d_i nur im Rechenregister geführt, so daß dort für d kein Datenspeicherplatz reserviert wird. Da jedoch die Nummer ω des letzten Datenspeicherplatzes gespeichert wird, benötigen sowohl das TI-Programm wie das HP-Programm n(n+2)+6 Datenspeicherplätze, falls die volle Matrix A gespeichert wird; bei Speicherung der Teilmatrix $(a_{ik})_{k\le i}$ einer symmetrischen Matrix A werden nur $\frac{n}{2}(n+5)+6$ Datenspeicherplätze benötigt.

Bei den Rechnern HP-67/97 ist die Größe des Datenspeichers mit 26 Datenregistern fest vorgegeben. Im ersten Fall ist daher nur $n \leq 3$ zulässig, während im zweiten Fall $n \leq 4$ möglich ist. Bei der hier gewählten Form der Adressenunterprogramme c wird der Stapelspeicher bis zur dritten Ebene belegt. Wenn in Programmzeile 035 durch Aufruf des Unterprogramms c die Adresse von c_k berechnet wird, kann daher nur noch der Faktor a_{ik} gleichzeitig im Stapelspeicher gehalten werden. Deshalb ist es im Unterschied zu dem Programm für die TI-Rechner hier erforderlich, für d einen Datenspeicherplatz zu reservieren.

Für den Rechner TI-58 stehen mit der Standardaufteilung des Speichers zwischen Datenspeicher und Programmspeicher 240 Programmschritte und 30 Datenspeicherplätze zur Verfügung, so daß $n \leq 4$ möglich ist. Bei dem Rechner TI-59 führt 9 Op 17 zur Aufteilung des Speichers in 240 Programmschritte und 90 Datenregister; bei Speicherung der vollen Koeffizientenmatrix können also Defektvektoren bis zur Reihenzahl $n \leq 8$ berechnet werden, bei Speicherung der Teilmatrix $(a_{ik})_{k \leq i}$ einer symmetrischen Koeffizientenmatrix ist $n \leq 10$ zulässig.

In dem Programm für die TI-Rechner wird das Testregister R_T benötigt für Vergleichsoperationen im Zusammenhang mit s sowie im Adressenunterprogramm A' für den Fall, daß nur die Teilmatrix $(a_{ik})_{k \leq i}$ gespeichert ist. Beschränkt man sich darauf, nur das Programm für vollständig gespeicherte Matrix zu verwenden, so kann s in R_T gespeichert werden. Nach den daraus resultierenden Änderungen des Programms wird für den Rechner TI-58 schließlich $n \leq 5$ zulässig, wenn man durch 4 Op 17 dann 40 Datenspeicherplätze und 160 Programmspeicherplätze zur Verfügung hat.

Beispiel

Mit n = 4,

$$A = \begin{pmatrix} 4.99 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad c = \begin{pmatrix} 212.5 \\ -128.1 \\ -53.1 \\ 31.2 \end{pmatrix}$$

wurden die Komponenten des Defektvektors und die Defektnorm in der nebenstehenden Form angezeigt.

```
 0.075
  0.1
   0.
 -0.1

  0.1
```

9.2. Das Gaussche Eliminationsverfahren

Es sei $A = (a_{ik})_{i,k=1,\ldots,n}$ eine invertierbare Matrix. Dann ist das lineare Gleichungssystem Ax = b für beliebige Vektoren $b = (b_i)_{i=1,\ldots,n}$ eindeutig lösbar. Für $n \geq 2$ ermitteln wir die Lösung mit Hilfe des Gaußschen Eliminationsverfahrens mit maximalen Spaltenpivots. Im Anschluß daran verwenden wir das Eliminationsverfahren auch zur Lösung mehrerer Gleichungssysteme mit derselben Koeffizientenmatrix A, womit wir auch die Inverse A^{-1} berechnen können.

Das lineare Gleichungssystem Ax = b lautet ausführlich geschrieben

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n. \end{aligned}$$

Der Grundgedanke des Eliminationsverfahrens besteht darin, in n-1 Gleichungen eine Unbekannte zu eliminieren. Mit den entstehenden n-1 Gleichungen für die restlichen n-1 Unbekannten wird dieses Vorgehen wiederholt. Auf diese Weise wird das Gleichungssystem schließlich in Dreiecksgestalt gebracht; anschließend wird die Lösung durch Rückwärtseinsetzen bestimmt.

Um diese Methode durch einen einfachen Algorithmus realisieren zu können, werden wir die Unbekannten $x_1,\ldots,x_n$ in ihrer natürlichen Reihenfolge eliminieren. Da $a_{11} = 0$ möglich ist, kann zur Elimination von x_1 nicht grundsätzlich die erste Gleichung herangezogen werden. Da aber nicht in jeder Gleichung der Koeffizient von x_1 Null sein kann, nehmen wir eine Vertauschung von Zeilen des Gleichungssystems vor. Für das numerische Verhalten des Verfahrens erweist es sich als günstig, die Zeilen so zu vertauschen, daß ein betragsgrößter Koeffizient von x_1 in der ersten Zeile steht. Die nach der Zeilenvertauschung entstandene Matrix nennen wir $A^{(0)} = (a_{ik}^{(0)})$, die neue rechte Seite $b^{(0)} = (b_i^{(0)})$, und es ist

$$|a_{11}^{(0)}| = \max_{i=1,\ldots,n} |a_{i1}^{(0)}| = \max_{i=1,\ldots,n} |a_{i1}|.$$

Wir wählen hierbei für $a_{11}^{(0)}$ den obersten betragsgrößten Koeffizienten von x_1 in der Matrix A.

Die erste Gleichung des Gleichungssystems $A^{(0)}x = b^{(0)}$ lautet

$$a_{11}^{(0)}x_1 + a_{12}^{(0)}x_2 + \ldots + a_{1n}^{(0)}x_n = b_1^{(0)}.$$

Subtrahieren wir das $a_{i1}^{(0)}/a_{11}^{(0)}$-fache dieser Gleichung von der i-ten Gleichung des Gleichungssystems $A^{(0)}x = b^{(0)}$ für $i = 2,\ldots,n$, so entstehen die Gleichungen

$$\begin{aligned}
&\tilde{a}_{22}^{(1)}x_2 + \ldots + \tilde{a}_{2n}^{(1)}x_n = \tilde{b}_2^{(1)},\\
&\quad\vdots\\
&\tilde{a}_{n2}^{(1)}x_2 + \ldots + \tilde{a}_{nn}^{(1)}x_n = \tilde{b}_n^{(1)},
\end{aligned}$$

mit den Koeffizienten

$$\tilde{a}_{ik}^{(1)} = a_{ik}^{(0)} - a_{i1}^{(0)}a_{1k}^{(0)}/a_{11}^{(0)},\quad \tilde{b}_i^{(1)} = b_i^{(0)} - a_{i1}^{(0)}b_1^{(0)}/a_{11}^{(0)},$$

für $i,k = 2,\ldots,n$.

Damit ist x_1 aus diesen Gleichungen eliminiert. Durch Wiederholung des beschriebenen Verfahrens für diese n-1 Gleichungen kann nun x_2 eliminiert werden. Hierzu nehmen wir wieder eine Zeilenvertauschung nach einem betragsgrößten Koeffizienten von x_2 vor. Die nach dieser Vertauschung entstandene Matrix nennen wir $A^{(1)}$. Bei der fortgesetzten Wiederholung werden für jedes t = 1,2,...,n-1 die Koeffizienten $\tilde{a}_{ik}^{(t)}$ und $\tilde{b}_i^{(t)}$ für i,k = t+1,...,n berechnet gemäß

$$\tilde{a}_{ik}^{(t)} = a_{ik}^{(t-1)} - a_{it}^{(t-1)} a_{tk}^{(t-1)} / a_{tt}^{(t-1)},$$

$$\tilde{b}_i^{(t)} = b_i^{(t-1)} - a_{it}^{(t-1)} b_t^{(t-1)} / a_{tt}^{(t-1)}.$$

Aus $\tilde{A}^{(t)} = (\tilde{a}_{ik}^{(t)})_{i,k=t+1,\ldots,n}$ entsteht die Matrix $A^{(t)} = (a_{ik}^{(t)})_{i,k=t+1,\ldots,n}$ durch eine Zeilenvertauschung, die in der Spalte mit der Nummter t+1 eine betragsgrößte der Zahlen $\tilde{a}_{i,t+1}^{(t)}$, i = t+1,...,n, in die Hauptdiagonale bringt, so daß gilt

$$|a_{t+1,t+1}^{(t)}| = \max_{i=t+1,\ldots,n} |a_{i,t+1}^{(t)}| = \max_{i=t+1,\ldots,n} |\tilde{a}_{i,t+1}^{(t)}|.$$

Wir wählen jeweils für $a_{t+1,t+1}^{(t)}$ den obersten betragsgrößten Koeffizienten von x_{t+1} in der Matrix $\tilde{A}^{(t)}$.

Die Nenner $a_{tt}^{(t-1)}$ sind von Null verschieden, wenn die Matrix A invertierbar ist. Andernfalls enthält das Gleichungssystem Ax = b oder eines der entstehenden Untersysteme eine Spalte, die ab der Hauptdiagonalen nur aus Nullen besteht, so daß das Gleichungssystem nicht oder nicht eindeutig lösbar ist; die Matrix A ist dann nicht invertierbar.

Werden die beschriebenen Eliminationsschritte ausgeführt für t = 1,...,n-1, so entsteht schließlich ein gestaffeltes Gleichungssystem

$$\begin{aligned}
a_{11}^{(0)} x_1 + a_{12}^{(0)} x_2 + \ldots + a_{1n}^{(0)} x_n &= b_1^{(0)},\\
a_{22}^{(1)} x_2 + \ldots + a_{2n}^{(1)} x_n &= b_2^{(1)},\\
\ddots \qquad\quad &\vdots\\
a_{nn}^{(n-1)} x_n &= b_n^{(n-1)}.
\end{aligned}$$

Hieraus kann die Lösung des Gleichungssystems Ax = b rekursiv berechnet werden durch Rückwärtseinsetzen,

$$x_n = b_n^{(n-1)}/a_{nn}^{(n-1)},$$

$$x_i = (b_i^{(i-1)} - \sum_{k=i+1}^{n} a_{ik}^{(i-1)} x_k)/a_{ii}^{(i-1)}, \quad i = n-1,\ldots,1.$$

Die Umformungen, die während des Eliminationsprozesses mit der Matrix A vorgenommen werden, lassen mit Ausnahme der Zeilenvertauschungen den Wert det A der Determinante von A unverändert. Bei jeder Zeilenvertauschung ändert sich das Vorzeichen der Determinante. Da die Determinante der schließlich entstandenen Dreiecksmatrix gleich dem Produkt ihrer Hauptdiagonalelemente ist, folgt

$$\det A = \sigma_0 \cdot \sigma_1 \cdot \cdots \cdot \sigma_{n-1} \cdot a_{11}^{(0)} \cdot a_{22}^{(1)} \cdot \cdots \cdot a_{nn}^{(n-1)},$$

wobei $\sigma_t = -1$, wenn zur Bildung der Matrix $A^{(t)}$ aus $\tilde{A}^{(t)}$ bzw. zur Bildung von $A^{(0)}$ aus A eine Zeilenvertauschung vorgenommen worden ist, sonst $\sigma_t = +1$, für $t = 0,1,\ldots,n-1$.

Wir fassen nun das beschriebene Eliminationsverfahren zur Lösung eines linearen Gleichungssystems Ax = b zu einem Algorithmus zusammen. Hierbei werden für jedes t die Elemente $\tilde{a}_{ik}^{(t)}$, $a_{ik}^{(t)}$ und $b_i^{(t)}$ auf die Plätze der Elemente a_{ik} und $a_{i,n+1} = b_i$ der Ausgangsmatrix und der rechten Seite gespeichert, ebenso die Quotienten $c_{it} = a_{it}^{(t-1)}/a_{tt}^{(t-1)}$ auf die Plätze von a_{it} für $i = t+1,\ldots,n$.

Algorithmus Voraussetzung: $n \geq 2$

Eingabe: n;

a_{ik} für $i = 1,\ldots,n$, $k = 1,\ldots,n+1$

mit $a_{i,n+1} = b_i$.

Anzeige: det A;

$x_1, x_2, \ldots, x_n$.

a_{ik}	d	i	i_t	k	n	t	u	x	x_1	x_2	...	x_n

C

```
d ← 1
t = 1(1)n-1
    u ← 0
    i = t(1)n
        u : |a_it| ≥
        u ← |a_it|
        i_t ← i
    i_t : t   =
    d ← -d
    k = t(1)n+1
        a_tk ↔ a_i_t k
    d ← d·a_tt
    i = t+1(1)n
        a_it ← a_it / a_tt
    k = t+1(1)n+1
    i = t+1(1)n
        a_ik ← a_ik - a_it a_tk
d ← d·a_nn
Anzeige d
x_n ← a_n,n+1 / a_nn
i = n-1(-1)1
    x ← a_i,n+1
    k = n(-1)i+1
        x ← x - a_ik x_k
    x_i ← x / a_ii
Anzeige x_1, x_2, ..., x_n
Stop
```

Erläuterungen

(1) u wird das oberste betragsgrößte Element in der t-ten Spalte, i_t der zugehörige Zeilenindex.

(2) Für i_t=t werden keine Zeilen vertauscht. Für $i_t \neq t$ wird das Vorzeichen von det A geändert, und die t-te Zeile der Matrix $(a_{ik})_{i\geq t, k\geq t}$ wird mit der i_t-ten Zeile vertauscht.

(3) Berechnung der Quotienten c_{it} und Speichern nach a_{it}.

(4) Eliminieren:
$$a_{ik}^{(t)} = a_{ik}^{(t-1)} - c_{it} a_{tk}^{(t-1)}$$

d = det A

(5) Lösung des gestaffelten Gleichungssystems

In den nachfolgenden Programmen speichern wir die Elemente a_{ik}, i=1,...,n, k=1,...,n+1, der erweiterten Koeffizientenmatrix spaltenweise vom Ende des Datenspeichers her. Mit der Adresse ω des letzten Datenspeicherplatzes gilt dann

$$\text{Adr } a_{ik} = \omega - (i-1 + (k-1)n),\ i=1,...,n,\ k=1,...,n+1.$$

Bei der Lösung des gestaffelten Gleichungssystems in Teil (5) überspeichern wir mit den Komponenten $x_1,...,x_n$ des Lösungsvektors die rechten Seiten $b_1,...,b_n$ des Gleichungssystems. Es ist daher

$$\text{Adr } x_k = \omega - n^2 + 1 - k,\ k=1,...,n.$$

Da bei der Lösung des gestaffelten Gleichungssystems bereits durch die Vorschrift des Algorithmus die Schleifen rückwärts durchlaufen werden, verwenden wir dort nicht die komplementären Indizes, sondern wir arbeiten an dieser Stelle der Programme mit den Schleifenindizes i und k selbst. Deshalb berechnen die Adressenunterprogramme d bzw. D' sowie e bzw. E' die Adressen von a_{ik} und x_k aus i und k.

In den Teilen (1) bis (4) ermitteln wir die Adressen der Matrizenelemente für

$$i=1,...,n,\ k=1,...,n+1,\ t=1,...,n-1$$

aus den komplementären Indizes

$$i' = n+1-i,\ k' = n+2-k,\ t' = n-t$$

mit den Adressenunterprogrammen

a bzw. A': $\text{Adr } a_{ik} = \omega - (i-1+(k-1)n)$
$= i' + (k'-n-2)n + \omega,$

b bzw. B': $\text{Adr } a_{it} = \omega - (i-1+(t-1)n)$
$= i' + (t'-n)n + \omega,$

c bzw. C': $\text{Adr } a_{tk} = \omega - (t-1+(k-1)n)$
$= t' + 1 + (k'-n-2)n + \omega.$

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	R_3	R_4	
d	i'	i'_t	t'	n	
x	i	k'			
		k			

$n \leq 4$

$R_{25-(n+1)n}$	...	R_{24-n^2}	R_{25-n^2}	...	R_{23}	R_{24}	R_I
$a_{n,n+1}=b_n$	...	$a_{1,n+1}=b_1$	a_{nn}	...	a_{21}	a_{11}	Adr
x_n	...	x_1					

Hauptprogramm

```
001   1 STO 0                          d ← 1
003   RCL 4 1 - STO 3                  t' ← n-1 (t ← 1)
(1)
007   LBL 0 0                          u ← 0          ←
009   RCL 3 1 + STO 1                  i' ← t'+1 (i←t)
013   LBL 1 R↓ b CLx                   R_I ← Adr a_it ←
017   RCL (i) ABS x≤y? GOTO 2          wenn |a_it|≤u  |
021   RCL 1 STO 2                      u←|a_it|,i'_t←i'(i_t←i)
023   LBL 2 CLx 1 STO I                R_X←1, R_I←1   ←
027   DSZ (i) GOTO 1                   i'←i'-1, wenn i'>0 ┘
(2)
029   RCL 3 + RCL 2 x=y? GOTO 4        wenn i'_t=t'+1 (i_t=t)┐
034   STO 1 R↓ STO 2 1 STO + 2         i'←i'_t,k'←t'+2(k←t)  |
039   CHS STO * 0                      d ← -d                |
041   LBL 3 c RCL (i)                  R_X ← a_tk     ←      |
044   a CLx RCL (i) x↔y STO (i)        a_tk↔a_i_tk    |      |
049   R↓ c R↓ STO (i)                                 |      |
053   2 STO I DSZ (i) GOTO 3           k'←k'-1, wenn k>0 ┘    |
057   LBL 4 RCL 3 1 + STO 1            i'←t'+1 (i←t)  ←      |
062   b RCL (i) STO * 0                d ← d·a_tt             |
(3)                                                           |
065   1 STO - 1 R↓                     i' ← t' (i ← t+1)      |
068   LBL 5 b R↓ STO ÷ (i)             a_it←a_it/a_tt ←       |
072   1 STO I R↓ DSZ (i) GOTO 5        i'←i'-1, wenn i'>0 ┘   |
                                                              |
                                                              ↑
```

```
(4)
077   RCL 3 1 + STO 2                 k'←t'+1 (k←t+1)
081   LBL 6 RCL 3 STO 1               i'←t' (i←t+1)
084   LBL 7 b RCL (i)                 R_X ← a_it
087   c CLx RCL (i)*                  R_X ← a_it a_tk
091   a R↓ STO - (i)                  a_ik←a_ik-a_it a_tk
094   1 STO I DSZ (i) GTO 7           i'←i'-1, wenn i'>0
098   STO 1                           i' ← 1 (i ← n)
099   2 STO I DSZ (i) GTO 6           k'←k'-1, wenn k'>0
103   STO 2                           k' ← 2 (k ← n)
104   3 STO I DSZ (i) GTO 0           t'←t'-1, wenn t'>0
108   a RCL (i) STO * 0 RCL 0 PRTx    d←d.a_nn, Anzeige d
113   PRTSPC                          (Papiervorschub)
(5)
114   RCL 4 STO 1 1 + STO 2           i ← n, k ← n + 1
119   d 1 STO - 2 RCL (i)             R_X←a_n,n+1 und k←n
123   d R↓ RCL (i) ÷ e R↓ STO (i)     x_n←a_n,n+1/a_nn
130   1 STO - 1                       i ← n - 1
132   LBL 8 RCL 4 1 + STO 2           k ← n + 1
137   d RCL (i) STO 0 1 STO - 2       x←a_i,n+1, k←n
142   LBL 9 d RCL (i)                 R_X ← a_ik
145   e R↓ RCL (i)* STO - 0           x ← x-a_ik x_k
150   1 STO - 2                       k ← k - 1
152   RCL 1 RCL 2 x>y? GTO 9          wenn k>i
156   d RCL (i) STO ÷ 0               x ← x/a_ii
159   e RCL 0 STO (i)                 x_i ← x
162   1 STO I DSZ (i) GTO 8           i←i-1, wenn i>0
166   LBL E e RCL (i) PRTx 1 STO + 2  Anzeige x_k,k←k+1
172   RCL 4 RCL 2 x≤y? GTO E          wenn k≤n
176   R/S                             Stop
```

Adressenunterprogramme

```
177  LBL a RCL 1 RCL 2 GTO D          i'+(k'  ──→ Adr a_ik
181  LBL b RCL 1 RCL 3 2 + GTO D      i'+(t'+2 ─→ Adr a_it
187  LBL c RCL 3 1 + RCL 2            t'+1+(k'    Adr a_tk
192  LBL D RCL 4 - 2 - RCL 4 * +      -n-2)n ←┘
200  24 + STO I                       +24, R_I←Adr
204  RTN                              Rücksprung
205  LBL d RCL 1 RCL 2 1 -            (i+(k-1)    Adr a_ik
210  RCL 4 * GTO B                    ·n)
213  LBL e RCL 2 RCL 4 x^2            (k+n^2)     Adr x_k
217  LBL B + CHS 25 +                 ·(-1)+25 ←┘
223  STO I                            R_I ← Adr
224  RTN                              Rücksprung
```

Programm für TI-59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	
Adr	i'	i'_t	t'	n	ω	d	
	i	k'					
		k					

$R_{\omega+1-(n+1)n}$	···	$R_{\omega-n^2}$	$R_{\omega+1-n^2}$	···	$R_{\omega-1}$	R_ω
$a_{n,n+1}=b_n$	···	$a_{1,n+1}=b_1$	a_{nn}	···	a_{21}	a_{11}
x_n		x_1				

Einleseprogramme

```
000  STO 4                            R_04 ← n
002  Op 16 INV Int * 100 = STO 5      R_05 ← ω
013  R/S                              Stop
014  Lbl A 1 STO 1 STO 2              i ← 1, k ← 1
021  D' GTO (                         R_00 ← Adr a_11 , ──┐
024  Lbl B 1 STO 2 E'                 k ← 1, R_00 ← Adr b_1 ┘
030  Lbl ( RCL Ind 0 R/S              Anzeige a_j, Stop ←┐
035  STO Ind 0                        Speichern von a_j  │
037  Op 30 GTO (                      Adr ← Adr - 1, ────┘
```

Hauptprogramm

```
041  Lbl C 1 STO 6                    d ← 1
046  RCL 4 STO 3 Op 33                t' ← n-1 (t ← 1)

(1)
052  Lbl INV O x↔t                    u ← O
056  RCL 3 STO 1 Op 21                i' ← t'+1 (i ← t)
062  Lbl lnx B' RCL Ind O |x|         R_X ← |a_it|
068  x↔t x≥t CE                       wenn u≥|a_it|
071  RCL 1 STO 2 x↔t                  i'_t ← i'(i_t←i), u←|a_it|
076  Lbl CE x↔t Dsz 1 lnx             i'←i'-1, wenn i'>O

(2)
082  RCL 3 + 1 = x↔t                  R_T ← t' + 1
088  RCL 2 x=t x↔t                    wenn i'_t=t'+1(i_t=t)
092  STO 1 x↔t STO 2 Op 22            i'←i'_t,k'←t'+2(k←t)
099  1 +/- Prd 6                      d ← -d
103  Lbl CLR C' RCL Ind O x↔t         R_T ← a_tk
109  A' x↔t Exc Ind O x↔t             a_tk↔a_i_t k
114  C' x↔t STO Ind O
118  Dsz 2 CLR                        k'←k'-1, wenn k'>O
121  Lbl x↔t RCL 3 STO 1 Op 21        i'←t'+1 (i←t)
129  B' RCL Ind O Prd 6 x↔t           d←d·a_tt, R_T←a_tt

(3)
135  Op 31                            i'←t' (i←t+1)
137  Lbl x² B' x↔t INV Prd Ind O      a_it←a_it/a_tt
144  x↔t Dsz 1 x²                     i'←i'-1, wenn i'>O

(4)
148  RCL 3 STO 2 Op 22                k'←t'+1 (k←t+1)
154  Lbl √x RCL 3 STO 1               i'←t' (i←t+1)
160  Lbl 1/x B' RCL Ind O *           R_X ← a_it
166  C' RCL Ind O = x↔t               R_T ← a_it a_tk
171  A' x↔t INV SUM Ind O             a_ik←a_ik-a_it a_tk
176  Dsz 1 1/x                        i'←i'-1, wenn i'>O
179  Dsz 2 √x                         k'←k'-1, wenn k'>O
182  Dsz 3 INV                        t'←t'-1, wenn t'>O
185  1 STO O1 2 STO 2                 i'←1,k'←2(i←n,k←n)
191  A' RCL Ind O Prd 6 RCL 6 Prt     d←d·a_nn, Anzeige d
199  Adv                              (Papiervorschub)
```

(5)

```
200  RCL 4 STO 1 STO 2 Op 22          i ← n, k ← n + 1
208  D' RCL Ind O ÷ Op 32             R_X ← a_{n,n+1}, k←n
214  D' RCL Ind O = x↔t               R_T ← a_{n,n+1}/a_nn
219  E' x↔t STO Ind O Op 31           x_n ← a_{n,n+1}/a_nn, i←n-1
225  Lbl STO RCL 4 STO 2 Op 22        k ← n + 1
233  D' RCL Ind O - Op 32             x ← a_{i,n+1}, k ← n
239  Lbl RCL D' RCL Ind O *           x←x-a_ik x_k, k←k-1
245  E' RCL Ind O - Op 32
251  RCL 2 x↔t RCL 1 INV x=t RCL      wenn k≠i
259  O = ÷ D' RCL Ind O = x↔t         R_T ← x/a_ii
267  E' x↔t STO Ind O                 x_i ← x/a_ii
271  Dsz 1 STO                        i←i-1, wenn i>O
274  Lbl SUM E' RCL Ind O Prt Op 22   Anzeige x_k, k←k+1
282  RCL 2 x↔t RCL 4 x≥t SUM          wenn n≥k
289  R/S                              Stop
```

Adressenunterprogramme

```
290  Lbl A' ( RCL 1 + ( RCL 2 GTO y^x  i'+(k'        Adr a_ik
301  Lbl B' ( RCL 1 +                  i'+           Adr a_it
307  ( RCL 3 + 2 GTO y^x               (t' + 2
314  Lbl C' ( RCL 3 + 1 + ( RCL 2      t'+1+(k'      Adr a_tk
325  Lbl y^x - RCL 4 - 2 ) * RCL 4     -n-2)n
336  + RCL 5 ) STO O                   + ω, R_OO←Adr
342  INV SBR                           Rücksprung
343  Lbl D' ( ( RCL 1 +                (i+           Adr a_ik
350  ( RCL 2 - 1 ) GTO EE              (k-1)
358  Lbl E' ( ( RCL 2 + RCL 4          (k+n          Adr x_k
367  Lbl EE * RCL 4 ) +/- +            ·n)·(-1)+
375  1 + RCL 5 ) STO O                 1+ω, R_OO←Adr
382  INV SBR                           Rücksprung
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der Zeilenzahl n des Gleichungssystems
   bei HP-67/97: n STO 4,
   bei TI-58/59: n RST R/S.
3. Eingabe der Elemente a_{ik} der Koeffizientenmatrix (spaltenweise) sowie der rechten Seiten b_i des Gleichungssystems
   bei HP-67/97 entsprechend der oben angegebenen Belegung des Datenspeichers,
   bei TI-59 durch Verwendung des Einleseprogramms:
   A a_{11} R/S a_{21} R/S ... a_{nn} R/S
   b_1 R/S b_2 R/S ... b_n R/S.
   Für den Rechner TI-59 können gleichwertig zu Schritt 3 die Schritte 3a und 3b unabhängig voneinander ausgeführt werden.

3a Eingabe der Matrix A (spaltenweise):
   A a_{11} R/S a_{21} R/S ... a_{nn} R/S.

3b Eingabe des Vektors b:
   B b_1 R/S b_2 R/S ... b_n R/S.

4. Durchführung der Rechnung, Anzeige von det A sowie der Lösung $x_1,\ldots,x_n$
   für HP-67/97: RTN R/S,
   für TI-59: C.

Mögliche Wiederholungen:

ab Schritt 3 für ein anderes Gleichungssystem mit derselben Zeilenzahl n,
ab Schritt 2 für eine andere Zeilenzahl n.

Mit der obigen Gestalt des Einleseprogramms für den TI-Rechner wird jeweils vor dem Eintasten eines Elements in Schritt 3 bzw. 3a, 3b der bisherige Inhalt des Speicherplatzes angezeigt, auf den der neu eingegebene Wert dann mit R/S gespeichert wird; nach R/S wird der Inhalt des nächsten Datenspeicherplatzes angezeigt. Damit ist es möglich, nach Beendigung der Eingabe die gespeicherten Werte nochmals sichtbar zu machen und zu überprüfen durch die Tastenfolge A R/S ... R/S entsprechend Schritt 3 bzw. durch Tastenfolgen entsprechend den Schritten 3a, 3b. Fehlerhaft eingegebene

Werte werden dann durch Überschreiben im Anzeigeregister auch im Datenregister berichtigt. Im Anschluß an Schritt 4 kann man entsprechend die Koeffizientenmatrix des gestaffelten Gleichungssystems überprüfen durch die Anweisungen nach Schritt 3a. Durch Einfügen einer zusätzlichen Stop-Anweisung in Programmschritt 199 können entsprechend, nachdem det A angezeigt ist, auch die rechten Seiten des gestaffelten Gleichungssystems sichtbar gemacht werden gemäß Schritt 3b. In dem darauffolgenden Teil (5) wird das gestaffelte Gleichungssystem gelöst, wobei diese rechten Seiten durch die Lösung überspeichert werden.

Das Programm für die TI-Rechner umfaßt die Schritte 000 bis 382, und es benötigt n(n+1)+7 Datenspeicherplätze. Das Programm kann zwar in den Rechner TI-58 eingegeben werden, wenn man die Speicherbereichsverteilung mit 1 Op 17 auf 400 Programmschritte einstellt; jedoch stehen dann nur noch 10 Datenspeicherplätze zur Verfügung, so daß das Programm auf diesem Rechner nicht gerechnet werden kann. Auf dem Rechner TI-59 stehen nach 7 Op 17 ebenfalls 400 Programmschritte zur Verfügung sowie 70 Datenspeicherplätze; daher ist es möglich, hier Gleichungssysteme mit der Zeilenzahl $n \leq 7$ zu lösen.

In den Rechnern HP-67/97 ist die Speicheraufteilung mit 26 Datenspeicherplätzen und 224 Programmschritten fest vorgegeben. Daher ist bei den dort benötigten n(n+1)+6 Datenspeicherplätzen $n \leq 4$ zulässig. Das Hauptprogramm und die Adressenunterprogramme belegen den Programmspeicher der HP-Rechner vollständig; daher mußte auf Einleseprogramme verzichtet werden.

In beiden Programmen ist kein Datenspeicherplatz für die Größe u reserviert, die nur in Teil (1) auftritt. In dem Programm für die HP-Rechner wird u im Stapelregister geführt. Das ist möglich, weil das gleichzeitig verwendete Adressenunterprogramm b zur Berechnung von Adr a_{it} das Stapelregister nur bis zur dritten Ebene belegt. Hierbei wird in den Programmzeilen 009 und 021 von der automatischen Anhebung des Stapels

bei Rückruf gespeicherter Größen Gebrauch gemacht, so daß die Wertzuweisungen u←0 und u←$|a_{it}|$ keine gesonderten Anweisungen erfordern. In dem Programm für die TI-Rechner wird u im Testregister R_T gespeichert, denn R_T wird weder bei der Berechnung von Adr a_{it} benötigt noch zur Steuerung der Schleife i=t(1)n, die in der Form i'=t'+1(-1)1 durch die Dsz-Anweisung kontrolliert wird.

Ebenso kommen wir in dem Programm für TI-59 ohne einen Datenspeicherplatz für die Größe x aus Teil (5) des Algorithmus aus, denn x wird im Rechenregister und im Testregister geführt. In ähnlicher Weise wäre es auch möglich, in dem Programm für die Rechner HP-67/69 die Größe x bei den Berechnungen für die Wertzuweisungen x←x-$a_{ik}x_k$ im Stapelregister zu halten; denn das Unterprogramm d zur Berechnung von Adr a_{ik} belegt das Stapelregister bis zur dritten Ebene, so daß x noch erhalten bliebe, und das Unterprogramm e für Adr x_k ist so angelegt, daß es das Stapelregister nur bis zur zweiten Ebene belegt; bei seinem Aufruf könnten also noch a_{ik} und x im Stapelspeicher aufbewahrt werden. Wir mußten jedoch zur Einsparung von Programmschritten auf diese Möglichkeit verzichten und speichern daher in dem Programm für HP-67/97 die Größe x gemeinsam mit d auf dem Speicherplatz R_0; nachdem d = det A angezeigt ist, steht nach der Lösung des gestaffelten Gleichungssystems in Teil (5) der Wert d also im Speicher nicht mehr zur Verfügung.

Beispiele

(i)
$$A = \begin{pmatrix} 2 & 3 & -1 & 0 \\ -6 & -5 & 0 & 2 \\ 2 & -5 & 6 & -6 \\ 4 & 6 & 2 & -3 \end{pmatrix}, \quad b = \begin{pmatrix} 20 \\ -45 \\ -3 \\ 58 \end{pmatrix}$$

det A = 40 und die Lösung
$x_1 = 1$, $x_2 = 7$, $x_3 = 3$,
$x_4 = -2$ wurden in der nebenstehenden Form angezeigt.
Die Rechnungszeit betrug
4 Minuten bei HP-97,
7 Minuten bei TI-59. Das gestaffelte Gleichungssystem, aus dem durch Rückwärtseinsetzen die Lösung erhalten wurde, lautete

```
 40.00000004  ***

  0.999999992 ***
  7.000000008 ***
  2.999999978 ***
 -2.000000026 ***
```

$$
\begin{aligned}
-6x_1 - 5x_2 + 2x_4 &= -45\\
-6.\overline{6}x_2 + 6x_3 - 5.\overline{3}x_4 &= -18\\
4.4x_3 - 3.8x_4 &= +20.8\\
-0.2\overline{27}x_4 &= +0.\overline{45}
\end{aligned}
$$

(ii)
$$
A = \begin{pmatrix} 2 & 3 & -1 & 0\\ 4 & 6 & 2 & -3\\ 2 & -5 & 6 & -6\\ -6 & -5 & 0 & 2 \end{pmatrix},\quad b = \begin{pmatrix} 20\\ 58\\ -3\\ -45 \end{pmatrix}
$$

Es handelt sich um dasselbe Gleichungssystem wie in Beispiel (i), jedoch sind zwei Zeilen vertauscht. Jetzt wurde det A = -40 und dieselbe Lösung x ermittelt.

(iii)
$$
A = \begin{pmatrix} 5 & 7 & 6 & 5\\ 7 & 10 & 8 & 7\\ 6 & 8 & 10 & 9\\ 5 & 7 & 9 & 10 \end{pmatrix},\quad b = \begin{pmatrix} 1\\ 0\\ 0\\ 0 \end{pmatrix}
$$

```
  1.000000006  ***

 67.99999956  ***
-40.99999974  ***
-16.99999988  ***
  9.999999930 ***
```

LÖSUNG MEHRERER GLEICHUNGSSYSTEME

Löst man für eine invertierbare Matrix A das Gleichungssystem $Ax = b$ mit $b = e^{(j)}$,

$$e_i^{(j)} = \begin{cases} 1 & \text{für } i = j \\ 0 & \text{für } i \neq j \end{cases} \qquad i,j = 1,\ldots,n,$$

so ist die Lösung x die j-te Spalte der inversen Matrix A^{-1}. In Beispiel (iii) ist also x die erste Spalte von A^{-1}. In diesem Zusammenhang erweist sich die obige Form des Algorithmus als schwerfällig, wenn man mit derselben Matrix A mehrere Gleichungssysteme $Ax = b$ lösen möchte, insbesondere wenn mit $b = e^{(j)}$, $j = 1,\ldots,n$, alle Spalten der inversen Matrix berechnet werden sollen. Bei dem obigen Algorithmus müssen dann zur Lösung jedes Gleichungssystems die Matrizenelemente a_{ik} für $i,k = 1,\ldots,n$ erneut eingegeben werden, da sie während der Rechnung verändert werden. Wir geben deshalb anschließend eine Fassung des Eliminationsverfahrens an, in der die wiederholte Eingabe und Elimination derselben Matrix überflüssig wird.

Gegeben seien eine invertierbare Matrix $A = (a_{ik})_{i,k=1,\ldots,n}$ und rechte Seiten $f^{(j)}$, $j = 1,\ldots,m$. Für die Gleichungssysteme $Ax = f^{(j)}$ ermitteln wir die zugehörigen Lösungsvektoren $x^{(j)}$ mit Hilfe des Gaußschen Eliminationsverfahrens in der folgenden Weise. Zunächst wird mit $f^{(1)}$ das obige Verfahren vollständig durchgeführt und man erhält die Lösung $x^{(1)}$. Nun ist auf den Speicherplätzen der Elemente a_{ik} für $1 \leq i \leq k \leq n$ die Matrix des gestaffelten Gleichungssystems gespeichert. Für $j \geq 2$ unterwerfen wir daher nur noch die rechte Seite $f^{(j)}$ den Transformationen, die bei der Elimination vorgenommen wurden, und lösen dann sogleich das gestaffelte Gleichungssystem mit Teil (5) des vorigen Algorithmus.

Die nachträgliche Transformation der rechten Seiten ermöglichen wir dadurch, daß in dem obigen Algorithmus auch die Indizes i_t der Zeilenvertauschungen gespeichert werden.

Algorithmus Voraussetzung: $n \geq 2$

Erster Schritt: Lösung von $Ax = f^{(1)}$

Eingabe: n;
a_{ik} für i=1,...,n, k = 1,...,n+1,
mit $a_{i,n+1} = f_i^{(1)}$.

Anzeige: det A;
$x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)}$.

j-ter Schritt: Lösung von $Ax = f^{(j)}$, $j \geq 2$

Eingabe: $a_{i,n+1} = f_i^{(j)}$, i = 1,...,n.

Anzeige: $x_1^{(j)}, x_2^{(j)}, \ldots, x_n^{(j)}$.

a_{ik}	d	i	i_1	i_2	...	i_{n-1}	k	n	t	u	x	x_1	x_2	...	x_n

Rechnung für den ersten Schritt:

Es wird der oben beschriebene Algorithmus C ausgeführt, wobei jetzt auch $i_1, \ldots, i_{n-1}$ gespeichert werden.

Rechnung für den j-ten Schritt, $j \geq 2$:

D
t = 1(1)n-1
i_t : t =
$a_{t,n+1} \leftrightarrow a_{i_t,n+1}$
i = t+1(1)n
$a_{i,n+1} \leftarrow a_{,in+1} - a_{it}a_{t,n+1}$
Sprung nach Teil (5) von C

Erläuterungen
Teil (2) für die neue rechte Seite
Teil (4) für die neue rechte Seite

Für die nachträgliche Transformation der neuen rechten Seite stehen die benötigten Elemente in der richtigen Anordung zur Verfügung, weil bei der Zeilenvertauschung in Teil (2) des Algorithmus C für t = 2,...,n-1 die erste bis (t-1)-te Spalte nicht betroffen waren.

Im Hinblick auf die erforderlichen Programmspeicherplätze geben wir nur ein Programm für den Rechner TI-59 an. Vom Ende des verfügbaren Datenspeichers aus speichern wir spaltenweise die Elemente a_{ik}, i=1,...,n, k=1,...,n+1, der erweiterten Koeffizientenmatrix, deren letzte Spalte durch die Lösung $x_1,...,x_n$ des Gleichungssystems überspeichert wird; daran anschließend werden die Zeilenindizes i_1, $i_2,...,i_{n-1}$ gespeichert. Die obigen Adressen bleiben daher unverändert. Zusätzlich ist

E: Adr $i_t = \omega - (n+1)n + 1 - t$
$= t' + 1 - (n+2)n + \omega$
für t = 1,...,n-1 und mit t' = n - t.

Programm für TI-59

Belegung des Datenspeichers

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}		$R_{\omega+2-(n+2)n}$	...	$R_{\omega-(n+1)n}$
Adr	i'	i'_t	t'	n	ω	d		i'_{n-1}	...	i'_1
	i	k'								
		k								

$R_{\omega+1-(n+1)n}$	...	$R_{\omega-n^2}$	$R_{\omega+1-n^2}$	...	$R_{\omega-1}$	R_ω
$a_{n,n+1}=b_n$	...	$a_{1,n+1}=b_1$	a_{nn}	...	a_{21}	a_{11}
x_n		x_1				

Einleseprogramme

000	STO 4	$R_{04} \leftarrow$ n
002	Op 16 INV Int * 100 = STO 5	$R_{05} \leftarrow \omega$
013	R/S	Stop
014	Lbl A 1 STO 1 STO 2	i ← 1, k ← 1
021	D' GTO (	$R_{00} \leftarrow$ Adr a_{11},
024	Lbl B 1 STO 1 E'	k ← 1, $R_{00} \leftarrow$ Adr b_1
030	Lbl (RCL Ind 0 R/S	Anzeige a_j, Stop
035	STO Ind 0	Speichern von a_j
037	Op 30 GTO (	Adr ← Adr - 1,

Hauptprogramm für den ersten Schritt

```
041  Lbl C 1 STO 6                    d ← 1
046  RCL 4 STO 3 Op 33                t' ← n-1 (t ← 1)

(1)
052  Lbl INV 0 x↔t                    u ← 0
056  RCL 3 STO 1 Op 21                i' ← t'+1 (i ← t)
062  Lbl lnx B' RCL Ind 0 |x|         R_X ← |a_it|
068  x↔t x≥t CE                       wenn u≥|a_it|
071  E RCL 1 STO 2 STO Ind 0 x↔t      i_t←i' (i_t←i),u←|a_it|
079  Lbl CE x↔t Dsz 1 lnx             i'←i'-1, wenn i'>0

(2)
085  RCL 3 + 1 = x↔t                  R_T ← t' + 1
091  RCL 2 x=t x↔t                    wenn i_t=t'+1 (i_t=t)
095  STO 1 x↔t STO 2 Op 22            i'←i_t,k'←t'+2(k←t)
102  1 +/- Prd 6                      d ← -d
106  Lbl CLR SBR sin                  a_tk↔a_i_tk
110  Dsz 2 CLR                        k'←k'-1, wenn k'>0
113  Lbl x↔t RCL 3 STO 1 Op 21        i'←t'+1 (i←t)
121  B' RCL Ind 0 Prd 6 x↔t           d←d·a_tt, R_T←a_tt

(3)
127  Op 31                            i' ← t' (i ← t + 1)
129  Lbl x² B' x↔t INV Prd Ind 0      a_it ← a_it/a_tt
136  x↔t Dsz 1 x²                     i'←i'-1, wenn i'>0

(4)
140  RCL 3 STO 2 Op 22                k' ← t'+1 (k ← t+1)
146  Lbl √x RCL 3 STO 1               i' ← t' (i ← t + 1)
152  Lbl 1/x SBR cos                  a_ik←a_ik - a_ita_tk
156  Dsz 1 1/x                        i'←i'-1, wenn i'>0
159  Dsz 2 √x                         k'←k'-1, wenn k'>0
162  Dsz 3 INV                        t'←t'-1, wenn t'>0
165  1 STO 01 2 STO 2                 i'←1,k'←2(i←n,k←n)
171  A' RCL Ind 0 Prd 6 RCL 6 Prt     d←d·a_nn, Anzeige d
179  Adv                              (Papiervorschub)
```

```
(5)
180  Lbl tan RCL 4 STO 1              i ← n
186  STO 2 Op 22                      k ← n + 1
190  D' RCL Ind O ÷ Op 32             R_X ← a_{n,n+1}, k ← n
196  D' RCL Ind O = x↔t               R_T ← a_{n,n+1}/a_nn
201  E' x↔t STO Ind O Op 31           x_n ← a_{n,n+1}/a_nn, i←n-1
207  Lbl STO RCL 4 STO 2 Op 22        k ← n + 1   ←┐
215  D' RCL Ind O - Op 32             x←a_{i,n+1}, k←n
221  Lbl RCL D' RCL Ind O *           x←x-a_ik x_k, k←k-1 ←┐
227  E' RCL Ind O - Op 32             }                    │
233  RCL 2 x↔t RCL 1 INV x=t RCL      wenn k≠i ────────────┘
241  O = ÷ D' RCL Ind O = x↔t         R_T ← x/a_ii
249  E' x↔t STO Ind O                 x_i ← x/a_ii
253  Dsz 1 STO                        i←i-1, wenn i>O ─────┘
256  Lbl SUM E' RCL Ind O Prt Op 22   Anzeige x_k, k←k+1 ←┐
264  RCL 2 x↔t RCL 4 x≥t SUM          wenn n≥k ────────────┘
271  R/S                              Stop
```

Adressenunterprogramme

```
272  Lbl E ( RCL 3 +                  t'+          Adr i_t
278  1 + ( O GTO y^x                  1+(O ──→┐
284  Lbl A' ( RCL 1 + ( RCL 2 GTO y^x i'+(k' ──→ Adr a_ik
295  Lbl B' ( RCL 1 +                 i'+          Adr a_it
301  ( RCL 3 + 2 GTO y^x              (t'+2 ──→┐
308  Lbl C' ( RCL 3 + 1 + ( RCL 2     t'+1+(k'     Adr a_tk
319  Lbl y^x - RCL 4 - 2 ) * RCL 4    -n-2)n ←─┘
330  + RCL 5 ) STO O                  +ω,R_OO←Adr
336  INV SBR                          Rücksprung
337  Lbl D' ( ( RCL 1 +               (i+          Adr a_ik
344  ( RCL 2 - 1 ) GTO EE             (k - 1) ──┐
352  Lbl E' ( ( RCL 2 + RCL 4         (k + n)      Adr x_k
361  Lbl EE * RCL 4 ) +/- +           ·n)·(-1)+ ←┘
369  1 + RCL 5 ) STO O                1+ω,R_OO←Adr
376  INV SBR                          Rücksprung
```

Hauptprogramm für den j-ten Schritt, $j \geq 2$

```
377  Lbl D RCL 4 STO 3 Op 33          t' ← n - 1 (t ← 1)
385  1 STO 2                          k' ← 1 (k ← n + 1)
388  Lbl log E RCL Ind O STO 1        i'←i'_t (i←i_t)
395  x↔t RCL 3 + 1 = x=t CP           wenn t'+1=i'_t(t=i_t)
403  SBR sin                          a_t,n+1 ↔ a_i_t,n+1
405  Lbl CP RCL 3 STO 1               i'←t' (i←t+1)
411  Lbl P→R SBR cos                  a_i,n+1 ← a_i,n+1 - a_it a_t,n+1
415  Dsz 1 P→R                        i'←i'-1, wenn i'>O
418  Dsz 3 log                        t'←t'-1, wenn t'>O
421  GTO tan                          Sprung nach Teil (5)
```

Unterprogramm Zeilenvertauschung

```
423  Lbl sin C' RCL Ind O x↔t         R_T ← a_tk
429  A' x↔t Exc Ind O x↔t             a_tk ↔ a_i_t,k
434  C' x↔t STO Ind O
438  INV SBR                          Rücksprung
```

Unterprogramm Eliminieren

```
439  Lbl cos B' RCL Ind O *           R_X ← a_it
445  C' RCL Ind O = x↔t               R_T ← a_it a_tk
450  A' x↔t INV SUM Ind O             a_ik ← a_ik - a_it a_tk
455  INV SBR                          Rücksprung
```

Anleitung zur Verwendung des Programms

1. Eingabe des Programms.
2. Eingabe der Zeilenzahl n des Gleichungssystems: n RST R/S.
3. Eingabe der Elemente a_{ik} der Koeffizientenmatrix (spaltenweise) sowie der Elemente $f_i^{(1)}$ der ersten rechten Seite des Gleichungssystems:
   A a_{11} R/S a_{21} R/S ... a_{nn} R/S
   $f_1^{(1)}$ R/S $f_2^{(1)}$ R/S ... $f_n^{(1)}$ R/S.
   Gleichwertig zu Schritt 3 können die Schritte 3a und 3b unabhängig voneinander ausgeführt werden.

3a Eingabe der Matrix A (spaltenweise):
   A a_{11} R/S a_{21} R/S ... a_{nn} R/S.

3b Eingabe des Vektors $f^{(1)}$:
B $f_1^{(1)}$ R/S $f_2^{(1)}$ R/S ... $f_n^{(1)}$ R/S.
4. Berechnung der Lösung von $Ax = f^{(1)}$, Anzeige von det A sowie der Lösungskomponenten $x_1^{(1)},\ldots,x_n^{(1)}$: C.
5. Eingabe einer neuen rechten Seite $f^{(j)}$, $j\geq 2$:
B $f_1^{(j)}$ R/S $f_2^{(j)}$ R/S ... $f_n^{(j)}$ R/S.
6. Rechnung zur Lösung von $Ax = f^{(j)}$ mit $j\geq 2$ und Anzeige der Lösungskomponenten $x_1^{(j)},\ldots,x_n^{(j)}$: D.

Mögliche Wiederholungen:
ab Schritt 5 für eine weitere rechte Seite,
ab Schritt 3 für eine andere Koeffizientenmatrix mit derselben Zeilenzahl n,
ab Schritt 2 für eine andere Zeilenzahl n.

Der Inhalt des Datenspeichers kann wie bei dem Programm zur Lösung eines einzigen Gleichungssystems sichtbar gemacht werden.

Es werden (n+2)n+6 Datenspeicherplätze benötigt und die Programmschritte 000 bis 455. Daher kann mit der Standardaufteilung zwischen Programmspeicher und Datenspeicher des Rechners TI-59 gearbeitet werden, bei der 480 Programmschritte und 60 Datenspeicherplätze verfügbar sind, und es ist $n\leq 6$ möglich.

Beispiele

Es wurden die folgenden Matrizen invertiert.

$$A = \begin{pmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{pmatrix},\quad B = \begin{pmatrix} 4.99 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{pmatrix},$$

$$M = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{pmatrix}$$

6.5 Minuten nach Betätigen der Taste C wurde im ersten Schritt die Determinante angezeigt, nach weiteren 1.5 Minuten die erste Spalte der inversen Matrix. Die Berechnung der weiteren Spalten mit D benötigte jeweils 3 Minuten.

Ergebnis: $\det A = 1$, $\det B = 0.32$, $\det M = 1.653 \cdot 10^{-7}$.

$$A^{-1} = \begin{pmatrix} 68. & -41. & -17. & 10. \\ -41. & 25. & 10. & -6. \\ -17. & 10. & 5. & -3. \\ 10. & -6. & -3. & 2. \end{pmatrix}$$

$$B^{-1} = \begin{pmatrix} 212.5 & -128.125 & -53.125 & 31.25 \\ -128.125 & 77.53125 & 31.78125 & -18.8125 \\ -53.125 & 31.78125 & 14.03125 & -8.3125 \\ 31.25 & -18.8125 & -8.3125 & 5.125 \end{pmatrix}$$

$$M^{-1} = \begin{pmatrix} 15.99999999 & -119.9999999 & 239.9999998 & -139.9999999 \\ -119.9999999 & 1199.999999 & -2699.999998 & 1679.999999 \\ 239.9999998 & -2699.999998 & 6479.999996 & -4199.999997 \\ -139.9999999 & 1679.999999 & -4199.999997 & 2799.999998 \end{pmatrix}$$

Bei der Invertierung der Matrix N wurde im ersten Schritt knapp 19 Minuten nach Betätigen der Taste C der Wert det N = -6 angezeigt, nach weiteren 3 Minuten die erste Spalte von N^{-1}. Die Berechnung der weiteren Spalten der inversen Matrix mit D dauerte jeweils etwas mehr als 6 Minuten.

$$N = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 2 & 3 & 4 & 5 & 6 \\ 3 & 3 & 3 & 4 & 5 & 6 \\ 4 & 4 & 4 & 4 & 5 & 6 \\ 5 & 5 & 5 & 5 & 5 & 6 \\ 6 & 6 & 6 & 6 & 6 & 6 \end{pmatrix}, N^{-1} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 & -0.8\overline{3} \end{pmatrix}$$

Die von Null verschiedenen Elemente von N^{-1} wurden durch die Unterdrückung der Schutzstellen mit der angegebenen Genauigkeit angezeigt; die als Null wiedergegebenen Elemente waren dem Betrag nach $\leq 2 \cdot 10^{-12}$, also Null im Rahmen der Rechengenauigkeit.

Das gestaffelte Gleichungssystem bei $Nx = e^{(1)}$ zur Bestimmung der ersten Spalte von N^{-1} lautete

$$
\begin{aligned}
6\,x_1 + 6\,x_2 + 6\,x_3 + 6\,x_4 + 6\,x_5 + 6\,x_6 &= 0\\
x_2 + 2\,x_3 + 3\,x_4 + 4\,x_5 + 5\,x_6 &= 1\\
x_3 + 2\,x_4 + 3\,x_5 + 4\,x_6 &= -10^{-12}\\
x_4 + 2\,x_5 + 3\,x_6 &= 0\\
x_5 + 2\,x_6 &= -10^{-12}\\
x_6 &= -10^{-12}
\end{aligned}
$$

9.3. Iterative Verfahren

Zur Lösung linearer Gleichungssysteme $Ax = b$ dienen neben direkten Verfahren wie dem obigen Eliminationsverfahren auch iterative Methoden. Ausgehend von einer Näherung $x^{(0)}$ für die Lösung z des Gleichungssystems wird eine Folge von Vektoren $x^{(1)}, x^{(2)}, x^{(3)}, \ldots$ bestimmt, die unter geeigneten Voraussetzungen gegen z konvergiert.

Iterative Verfahren sind besonders geeignet bei großen Gleichungssystemen, wie sie zum Beispiel bei der Differenzenapproximation partieller Differentialgleichungen auftreten. Häufig ist dann die Koeffizientenmatrix A dünn besetzt, und man kann ihre Elemente a_{ik} in Abhängigkeit von i und k als Unterprogramm realisieren statt sie zu speichern. Mit einem derartigen virtuellen Speicher wird die Behandlung der großen Zahlenfelder auch möglich bei vergleichsweise geringem Umfang des Datenspeichers, wie ihn programmierbare Taschenrechner aufweisen.

Im folgenden gehen wir jedoch zur Erläuterung der Verfahren davon aus, daß die Koeffizientenmatrix des Gleichungssystems in den Datenregistern gespeichert ist. Wir stellen die Algorithmen zur Berechnung der Näherungslösungen auf für das Gesamtschrittverfahren und das Einzelschrittverfahren, beschränken uns dann aber im Hinblick auf die Realisierung durch Taschenrechnerprogramme auf das Einzelschrittverfahren und das zugehörige Relaxationsverfahren.

DAS GESAMTSCHRITTVERFAHREN

Die Methode der sukzessiven Approximation $x^{(t+1)} = g(x^{(t)})$ zur Bestimmung von Fixpunkten einer Funktion g führt bei Übertragung auf lineare Gleichungssysteme auf das Jacobische Gesamtschrittverfahren.

Besitzt die Matrix A von Null verschiedene Diagonalelemente,

$$a_{ii} \neq 0, \quad i = 1,\ldots,n,$$

so ist zu Ax = b die folgende Beziehung gleichwertig,

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{k=1 \\ k\neq i}}^{n} a_{ik}x_k\right), \quad i = 1,\ldots,n.$$

Damit ist das Gleichungssystem Ax=b in eine Fixpunktgleichung für den Vektor der Unbekannten $x = (x_1,\ldots,x_n)$ umformuliert. Das Verfahren der sukzessiven Approximation ergibt hier nun die Iterationsvorschrift des Gesamtschrittverfahrens:

$$x_i^{(t+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{k=1 \\ k\neq i}}^{n} a_{ik}x_k^{(t)}\right), \quad i = 1,\ldots,n,$$

für t = 0,1,2,... .

Unter der Voraussetzung des schwachen Zeilensummenkriteriums,

$$\sum_{\substack{k=1\\k\neq i}}^{n} |a_{ik}| \leq |a_{ii}|, \quad \sum_{k=i+1}^{n} |a_{ik}| < |a_{ii}|, \quad i = 1,\ldots,n,$$

ist die Matrix A invertierbar und somit das Gleichungssystem $Ax = b$ für jede rechte Seite b eindeutig lösbar (s. [9] 8.1); weiter konvergiert für jede Anfangsnäherung $x^{(0)}$ das Gesamtschrittverfahren gegen die Lösung z des Gleichungssystems mit einer a-posteriori-Fehlerabschätzung

$$||x^{(t)} - z|| \leq \text{const.} ||x^{(t)} - x^{(t-1)}||, \quad t = 1,2,\ldots .$$

Algorithmus Voraussetzung: Die Matrix A erfülle das schwache Zeilensummenkriterium.

Eingabe: n, $(a_{ik})_{i,k=1,\ldots,n}$, $(b_i)_{i=1,\ldots,n}$,
$x_i = x_i^{(0)}$ für $i=1,\ldots,n$.

Anzeige: $t, x^{(t)}$ für $t = 1,2,\ldots$.

a_{ik}	b_i	d	i	k	n	t	x	x_i	y_i

```
t ← 1
Anzeige t                    ←
i = 1(1)n
x ← b_i
k = 1(1)n
≠  k : i
   d ← a_ii
   x ← x - a_ik x_k          ←
y_i ← x/d
Anzeige y_i
i = 1(1)n
x_i ← y_i
t ← t + 1                    ┘
```

Hier werden die Komponenten der neu berechneten Näherung auf den Speicherplätzen $y_1,\ldots,y_n$ gespeichert. Nach Beendigung eines Iterationsschrittes dienen sie dann im folgenden Schritt als vorherige Näherung; dazu werden sie auf die Speicherplätze $x_1,\ldots,x_n$ umgespeichert.

DAS EINZELSCHRITTVERFAHREN

Unter der Voraussetzung $a_{ii} \neq 0$, $i = 1,\ldots,n$ an die Diagonalelemente der Matrix A betrachten wir wieder das lineare Gleichungssystem $Ax = b$ in der Darstellung

$$x_i = \frac{1}{a_{ii}} \Big(b_i - \sum_{\substack{k=1 \\ k\neq i}}^{n} a_{ik}x_k\Big), \quad i = 1,\ldots,n.$$

Die Iterationsvorschrift des Einzelschrittverfahrens von Gauß-Seidel zur Lösung dieser Gleichungen lautet

$$x_i^{(t+1)} = \frac{1}{a_{ii}} \Big(b_i - \sum_{k=1}^{i-1} a_{ik}x_k^{(t+1)} - \sum_{k=i+1}^{n} a_{ik}x_k^{(t)}\Big)$$

für $i = 1,\ldots,n$, $t = 0,1,2,\ldots$. Der Unterschied zu dem Gesamtschrittverfahren besteht darin, daß hier die schon berechneten Komponenten $x_1^{(t+1)},\ldots,x_{i-1}^{(t+1)}$ der neuen Näherung $x^{(t+1)}$ verwendet werden, um die Komponenten $x_i^{(t+1)}$ für $i = 2,\ldots,n$ zu ermitteln.

Sowohl unter der obigen Voraussetzung des schwachen Zeilensummenkriteriums für die Matrix A wie auch für positiv definite Matrizen A gibt es eine Konstante q, $0<q<1$, so daß das Einzelschrittverfahren mit beliebiger Anfangsnäherung $x^{(0)}$ gegen die eindeutig bestimmte Lösung z des Gleichungssystems $Ax = b$ konvergiert (s. [9] 8.2) mit der a-priori-Fehlerabschätzung

$$||x^{(t)} - z|| \leq \text{const.} \cdot ||x^{(1)} - x^{(0)}|| \cdot q^t, \quad t = 1,2,\ldots,$$

und der a-posteriori-Fehlerabschätzung

$$||x^{(t)} - z|| \leq \text{const.} \cdot ||x^{(t)} - x^{(t-1)}||, \quad t = 1,2,\ldots .$$

Algorithmus Voraussetzung: Die Matrix A erfüllt das schwache Zeilensummenkriterium, oder A sei positiv definit.

Eingabe: n, $(a_{ik})_{i,k=1,\ldots,n}$, $(b_i)_{i=1,\ldots,n}$,

$x_i = x_i^{(0)}$ für $i=1,\ldots,n$.

Anzeige: $t, x^{(t)}$ für $t = 1,2,\ldots$.

a_{ik}	b_i	d	i	k	n	t	x	x_i

```
t ← 1
Anzeige t                ←
i = 1(1)n
  x ← b_i
  k = 1(1)n
  ≠  k : i
     d ← a_ii
     x ← x - a_ik x_k
  x_i ← x/d
  Anzeige x_i
  t ← t + 1              ←
```

Bei dem Gesamtschrittverfahren war es erforderlich, die Komponenten der neu berechneten Näherung gesondert zu speichern und am Ende eines jeden Iterationsschrittes nach $x_1,\ldots,x_n$ umzuspeichern. Im Gegensatz dazu überspeichern hier die schon berechneten Komponenten der neuen Näherung sofort die vorherigen Werte. Daher benötigt man bei der Programmierung des Einzelschrittverfahrens weniger Datenspeicherplätze als bei dem Gesamtschrittverfahren. Da außerdem für zahlreiche numerische Beispiele das Einzelschrittverfahren schneller konvergiert als das Gesamtschrittverfahren, beschränken wir uns im folgenden auf das Einzelschrittverfahren.

In den Programmen für das Einzelschrittverfahren speichern wir die Elemente der Matrix A spaltenweise vom Ende des Datenspeichers her, anschließend die Komponenten der rechten Seite b und des Näherungsvektors $x^{(t)}$. Mit der Adresse ω des letzten Datenspeicherplatzes und mit den komplementären Indizes $i' = n+1-i$, $k' = n+1-k$ lauten die Adressen dieser Speicherplätze für $i,k = 1,\ldots,n$:

$$\text{Adr } a_{ik} = \omega - (i-1 + (k-1)n) = i' - (n+1-k')n + \omega,$$

$$\text{Adr } b_i = \omega - n^2 - (i-1) = i' - (n+1)n + \omega,$$

$$\text{Adr } x_k = \omega - (n+1)n - (k-1) = k' - (n+2)n + \omega.$$

Außerdem bedienen wir uns auch der Möglichkeit, für symmetrische Matrizen nur die Teilmatrix $(a_{ik})_{1\le k\le i\le n}$ zu speichern, welche die Matrizenelemente in den Spalten ab der Hauptdiagonalen enthält; in diesem Fall lauten dann die Speicherplatzadressen für $i,k = 1,\ldots,n$:

$$\text{Adr } a_{ik} = \omega - (i-1 + \tfrac{1}{2}(k-1)(2n-k)), \qquad k\le i,$$

$$= \tfrac{1}{2}k'(k'-1) + i' - (n+1)\tfrac{n}{2} + \omega, \quad k'\ge i',$$

$$\text{Adr } b_i = \omega - \tfrac{1}{2}n(n+1) - (i-1) = i' - (n+3)\tfrac{n}{2} + \omega,$$

$$\text{Adr } x_k = \omega - \tfrac{1}{2}n(n+3) - (k-1) = k' - (n+5)\tfrac{n}{2} + \omega.$$

Programm für HP-67/97

Belegung des Datenspeichers bei Speicherung der vollen Matrix

R_0	R_1	R_2	R_3	R_4	R_5		$R_{25-(n+2)n}$	...	$R_{24-(n+1)n}$
x	d	i'	k'	n	t		x_n	...	x_1

$R_{25-(n+1)n}$	...	R_{24-n^2}	R_{25-n^2}	...	R_{23}	R_{24}	R_I
b_n	...	b_1	a_{nn}	...	a_{21}	a_{11}	Adr

$n\le 3$

Belegung des Datenspeichers bei Speicherung der Teilmatrix $(a_{ik})_{k\leq i}$

R_0	R_1	R_2	R_3	R_4	R_5		$R_{25-(n+5)n/2}$	...	$R_{24-(n+3)n/2}$	$n\leq 4$
x	d	i'	k'	n	t		x_n	...	x_1	

$R_{25-(n+3)n/2}$	...	$R_{24-(n+1)n/2}$	$R_{25-(n+1)n/2}$	...	R_{23}	R_{24}	R_I
b_n	...	b_1	a_{nn}	...	a_{21}	a_{11}	Adr

Einleseprogramme

```
001   LBL A RCL 4 STO 2 STO 3          i' ← n, k' ← n
005   a GTO 0                          R_I ← Adr a_11,
007   LBL B RCL 4 STO 2 b GTO 0        i' ← n, R_I ← Adr b_1,
012   LBL C RCL 4 STO 2 c              i' ← n, R_I ← Adr x_1
016   LBL 0 RCL (i) R/S                Anzeigen, Stop
019   STO (i) DSZ I GTO 0              Speichern,Adr←Adr-1,
```

Hauptprogramm

```
022   LBL D 1 STO 5                    t ← 1
025   LBL 1 PRTSPC RCL 5 PRTx          Anzeige t
029   RCL 4 STO 2                      i' ← n
031   LBL 2 b RCL (i) STO 0            x ← b_i
035   RCL 4 STO 3                      k' ← n
037   LBL 3 RCL 2 RCL 3 x≠y? GTO 4     wenn i' ≠ k'
042   a RCL (i) STO 1 GTO 5            d ← a_ii,
046   LBL 4 a RCL (i)                  R_X ← a_ik
049   d CLx RCL (i) * STO - 0          x ← x - a_ik x_k
054   LBL 5 3 STO I DSZ (i) GTO 3      k'←k'-1, wenn k'>0
059   c RCL 0 RCL 1 ÷ STO (i) PRTx     x_i←x/d, Anzeige x_i
065   2 STO I DSZ (i) GTO 2            i'←i'-1, wenn i'>0
069   1 STO + 5 GTO 1                  t ← t + 1,
```

Adressenunterprogramme bei Speicherung der vollen Matrix

```
072  LBL a RCL 2 RCL 3 CHS GTO 6      i'-(-k' ──→ Adr a_ik
077  LBL b RCL 2 0 GTO 6              i'-(0   ──→ Adr b_i
081  LBL c RCL 2 1 GTO 6              i'-(1   ──→ Adr x_i
085  LBL d RCL 3 1                    k'-(1       Adr x_k
088  LBL 6 1 + RCL 4 + RCL 4 * -      +1+n)n ←──┘
096  24 + STO I                       +24, R_I←Adr
100  RTN                              Rücksprung
```

Adressenunterprogramme bei Speicherung der Teilmatrix $(a_{ik})_{k\leq i}$

```
072  LBL a RCL 2 RCL 3 x≤y? x↔y       wenn k'≤i', dann R_X↔R_Y
077  ENTER ENTER 1 - * 2 ÷            (k'-1)k'/2
084  + 1 GTO 6                        +i'-(1  ──→ Adr a_ik
087  LBL b RCL 2 3 GTO 6              i'-(3   ──→ Adr b_i
091  LBL c RCL 2 5 GTO 6              i'-(5   ──→ Adr x_i
095  LBL d RCL 3 5                    k'-(5       Adr x_k
098  LBL 6 RCL 4 + RCL 4 * 2 ÷ -      +n)n/2 ←──┘
106  24 + STO I                       +24, R_I←Adr
110  RTN                              Rücksprung
```

Programm für TI-58/59

Belegung des Datenspeichers bei Speicherung der vollen Matrix

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}		$R_{\omega-(n+2)n+1}$	...	$R_{\omega-(n+1)n}$
Adr	d	i'	k'	n	t	ω		x_n	...	x_1

$R_{\omega-(n+1)n+1}$	...	$R_{\omega-n^2}$	$R_{\omega-n^2+1}$	...	$R_{\omega-1}$	R_{ω}
b_n	...	b_1	a_{nn}	...	a_{21}	a_{11}

Belegung des Datenspeichers bei Speicherung der Teilmatrix $(a_{ik})_{k \le i}$

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}		$R_{\omega-(n+5)n/2+1}$	...	$R_{\omega-(n+3)n/2}$
Adr	d	i'	k'	n	t	ω		x_n	...	x_1

$R_{\omega-(n+3)n/2+1}$	...	$R_{\omega-(n+1)n/2}$	$R_{\omega-(n+1)n/2+1}$	...	$R_{\omega-1}$	R_{ω}
b_n	...	b_1	a_{nn}	...	a_{21}	a_{11}

Einleseprogramme

```
000   STO 4                          R04 ← n
002   Op 16 INV Int * 100 = STO 6    R06 ← ω
013   R/S                            Stop
014   Lbl A RCL 4 STO 2 STO 3        i' ← n, k' ← n
022   A' GTO INV                     R00 ← Adr a11',
025   Lbl B RCL 4 STO 2 B' GTO INV   i'←n, R00←Adr b1',
034   Lbl C RCL 4 STO 2 C'           i'←n, R00←Adr x1
041   Lbl INV RCL Ind 0 R/S          Anzeigen, Stop
046   STO Ind 0 Dsz 0 INV            Speichern, Adr←Adr-1,
```

Hauptprogramm

```
051   Lbl D 1 STO 5                  t ← 1
056   Lbl lnx Adv RCL 5 Prt          Anzeige t
062   RCL 4 STO 2                    i' ← n
066   Lbl CE RCL 4 STO 3             k' ← n
072   B' RCL Ind 0                   x ← bi
075   Lbl CLR - RCL 2 x↔t            RT ← i'
081   RCL 3 INV x=t x↔t              wenn k'≠i'
086   A' RCL Ind 0 STO 1 0 GTO x²    d ← aii',
094   Lbl x↔t A' RCL Ind 0 *         x←x-aikxk
100   D' RCL Ind 0
103   Lbl x² Dsz 3 CLR               k'←k'-1, wenn k'>0
108   = ÷ C' RCL 1 = STO Ind 0 Prt   xi←x/d, Anzeige xi
117   Dsz 2 CE                       i'←i'-1, wenn i'>0
120   Op 25 GTO lnx                  t ← t + 1,
```

Adressenunterprogramme bei Speicherung der vollen Matrix

```
124  Lbl A' ( RCL 2 -                    i'-          Adr a_ik
130  ( RCL 3 +/- + GTO √x                (-k'+   ──┐
137  Lbl B' ( RCL 2 - ( GTO √x           i'-(    ──┼─ Adr b_i
146  Lbl C' ( RCL 2 - (1 + GTO √x        i'-(1+  ──┼─ Adr x_i
157  Lbl D' ( RCL 3 - ( 1 +              k'-(1+  ──┼─ Adr x_k
166  Lbl √x 1 + RCL 4 ) * RCL 4 +        1+n)n+  ←─┘
177  RCL 6 ) STO 0                       ω, R00←Adr
182  INV SBR                             Rücksprung
```

Adressenunterprogramme bei Speicherung der Teilmatrix $(a_{ik})_{k \leq i}$

```
124  Lbl A' ( RCL 2 x↔t RCL 3            R_X ← k', R_T ← i'
132  x≥t 1/x                             wenn k'≥i'  ──┐
134  x↔t                                 R_X↔R_T       │
135  Lbl 1/x * ( CE - 1 ) ÷ 2 +          k'(k'-1)/2+ ←─┘
146  x↔t - ( 1 GTO √x                    i'-(1   ──┐
152  Lbl B' ( RCL 2 - ( 3 GTO √x         i'-(3   ──┼─ Adr a_ik
162  Lbl C' ( RCL 2 - ( 5 GTO √x         i'-(5   ──┼─ Adr b_i
172  Lbl D' ( RCL 3 - ( 5                k'-(5   ──┼─ Adr x_i
180  Lbl √x + RCL 4 ) * RCL 4 ÷ 2 +      +n)n/2+ ←─┘  Adr x_k
192  RCL 6 ) STO 0                       ω, R00 ← Adr
197  INV SBR                             Rücksprung
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms (Einleseprogramme, Hauptprogramm und eine Art der Adressenunterprogramme).
2. Eingabe von n
   bei HP-67/97: n STO 4,
   bei TI-58/59: n RST R/S.
3. Eingabe der Matrix A (spaltenweise), der rechten Seite b und der Anfangsnäherung $x^{(0)}$:

   A a_{11} R/S a_{21} R/S ... a_{nn} R/S
   b_1 R/S b_2 R/S ... b_n R/S
   $x_1^{(0)}$ R/S $x_2^{(0)}$ R/S ... $x_n^{(0)}$ R/S.

Gleichwertig zu Schritt 3 können die Schritte 3a, 3b, 3c einzeln und unabhängig voneinander ausgeführt werden.

3a Eingabe der Matrix A (spaltenweise):
A a_{11} R/S a_{21} R/S ... a_{nn} R/S.

3b Eingabe des Vektors b:
B b_1 R/S b_2 R/S ... b_n R/S.

3c Eingabe der Anfangsnäherung $x^{(0)}$:
C $x_1^{(0)}$ R/S $x_2^{(0)}$ R/S ... $x_n^{(0)}$ R/S.

4. Berechnung der Näherungslösungen und Anzeige von t, $x_1^{(t)}$, $x_2^{(t)}$,...,$x_n^{(t)}$ für t = 1,2,...: D.

Mögliche Wiederholungen:

ab Schritt 3c bzw. 3b bzw. 3a für eine andere Anfangsnäherung $x^{(0)}$, für eine andere rechte Seite b bzw. für eine andere nxn-Matrix A;
ab Schritt 2 für eine andere Zeilenzahl n.

Verwendet man das Programm für symmetrische Matrizen bei Speicherung der Teilmatrix $(a_{ik})_{k\leq i}$, so sind in Schritt 3 bzw. 3a die Matrizenelemente jeweils nur ab der Hauptdiagonalen aufeinanderfolgend einzugeben.

Wie in den Programmen zum Gaußschen Eliminationsverfahren wird auch hier vor dem Eintasten eines Elements in Schritt 3 bzw. 3a, 3b, 3c der bisherige Inhalt des Speicherplatzes angezeigt, auf den der neu eingegebene Wert dann mit R/S gespeichert wird; nach R/S wird der Inhalt des nächsten Datenspeicherplatzes angezeigt. Dadurch ist es möglich, nach Beendigung der Eingabe die gespeicherten Werte zu überprüfen durch die Tastenfolge A R/S ... R/S entsprechend Schritt 3 bzw. durch die Tastenfolgen entsprechend den Schritten 3a, 3b, 3c. Fehlerhaft eingegebene Werte werden hierbei durch Überschreiben im Anzeigeregister auch im Datenspeicher berichtigt.

In Schritt 2 wird für die Rechner TI-58/59 die Zeilenzahl n gespeichert sowie die Datenspeicherverteilung durch den letzten Datenspeicherplatz festgelegt.

Bei Speicherung der vollständigen Koeffizientenmatrix werden $n(n+2)+7$ Datenspeicherplätze benötigt, bei Speicherung der Teilmatrix $(a_{ik})_{k\leq i}$ einer symmetrischen Matrix sind $\frac{1}{2}n(n+5)+7$ Datenspeicherplätze erforderlich.

Auf den Rechnern HP-67/97 ist im ersten Fall daher $n\leq 3$ zulässig, im zweiten Fall kann $n\leq 4$ gewählt werden.

Für den Rechner TI-58 stehen bei der Standardaufteilung des Speichers 240 Programmschritte und 30 Datenspeicherplätze zur Verfügung. Daher ist bei Speicherung der vollen Matrix nur $n\leq 3$ zulässig, bei Speicherung der Teilmatrix $(a_{ik})_{k\leq i}$ einer symmetrischen Matrix ist $n\leq 4$ möglich. Ändert man jedoch das obige Programm so ab, daß ω nicht gespeichert wird, sondern bei Bedarf jeweils mit Hilfe eines Unterprogramms (entsprechend den obigen Programmschritten 002 bis 010) berechnet wird, so kann auch bei Speicherung der vollen Matrix $n\leq 4$ gewählt werden. Verzichtet man auf die Einleseprogramme A, B, C, so werden weniger als 160 Programmschritte benötigt; nach 4 Op 17 stehen dann auf dem Rechner TI-58 insgesamt 40 Datenspeicherplätze zur Verfügung, und es ist ohne sonstige Änderungen des obigen Programms $n\leq 4$ bei Speicherung der vollen Matrix und $n\leq 6$ bei Speicherung der Teilmatrix möglich.

Der Rechner TI-59 bietet nach 9 Op 17 ebenfalls 240 Programmschritte, gleichzeitig 90 Datenspeicherplätze. Daher kann man bei Speicherung der vollen Matrix Gleichungssysteme mit bis zu n = 8 Unbekannten lösen, bei Speicherung der Teilmatrix bis zu n = 10.

Beispiel

Das lineare Gleichungssystem Ax = b mit

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

besitzt die Lösung z = (0.8, 0.6, 0.4, 0.2). Mit dem Einzelschrittverfahren und $x^{(0)}$ = (1, 1, 1, 1) als Anfangsnäherung wurden die folgenden Näherungsvektoren $x^{(t)}$ für z errechnet.

t = 1	t = 2	t = 3	t = 4	t = 5
1.000000000	1.000000000	1.000000000	0.937500000	0.890625000
1.000000000	1.000000000	0.875000000	0.781250000	0.718750000
1.000000000	0.750000000	0.625000000	0.546875000	0.496093750
0.500000000	0.375000000	0.312500000	0.273437500	0.248046875

t = 10	t = 20	t = 30	t = 40	t = 50
0.810897828	0.800157215	0.800002268	0.800000033	0.800000001
0.614265442	0.600285797	0.600002969	0.600000043	0.600000001
0.411540965	0.400165493	0.400002402	0.400000035	0.400000001
0.205770493	0.200083247	0.200001201	0.200000017	0.200000006

Mit dem Rechner HP-97 wurde dieses Beispiel mit Speicherung der Teilmatrix $(a_{ik})_{k<i}$ gerechnet; die Berechnung eines Näherungsvektors $x^{(t+1)}$ aus $x^{(t)}$ dauerte 73 Sekunden. Bei der Rechnung des entsprechenden Programms mit dem Rechner TI-59 wurden 103 Sekunden je Iteration benötigt. Bei Speicherung der vollen Matrix waren mit TI-59 jeweils 87 Sekunden für eine Iteration erforderlich. In diesem Fall ist das Adressenunterprogramm A' kürzer.

RELAXATIONSVERFAHREN ZUM EINZELSCHRITTVERFAHREN

Für positiv definite Matrizen A geben wir eine weitere Methode zur Lösung linearer Gleichungssysteme Ax = b an. Mit ihrer Eigenschaft

$$a_{ii} \neq 0,\ i = 1,\ldots,n,$$

bildet man mit einer Zahl $\sigma \neq 0$ die Näherungsvektoren nach der Vorschrift

$$x_i^{(t+1)} = (1-\sigma)x_i^{(t)} + \frac{\sigma}{a_{ii}}\left(b_i - \sum_{k=1}^{i-1} a_{ik}x_k^{(t+1)} - \sum_{k=i+1}^{n} a_{ik}x_k^{(t)}\right)$$

für i = 1,...,n und t = 0,1,2,... . Zur Berechnung des neuen Näherungsvektors $x^{(t+1)}$ wird also jeweils eine Linearkombination der vorherigen Näherung $x^{(t)}$ und der rechten Seite der Iterationsvorschrift des Einzelschrittverfahrens gebildet. Für beliebige Anfangsnäherung $x^{(0)}$ und für $0<\sigma<2$ herrscht Konvergenz gegen die eindeutig bestimmte Lösung z des linearen Gleichungssystems Ax = b (s. [9] 8.3.2). Für $0<\sigma<1$ spricht man von Unterrelaxation, für $1<\sigma<2$ nennt man dieses Näherungsverfahren ein Überrelaxationsverfahren zur Lösung des Gleichungssystems. Für $\sigma = 1$ entsteht das obige Einzelschrittverfahren. In Abhängigkeit von der gegebenen positiv definiten Matrix A kann σ optimal gewählt werden, so daß eine wesentliche Beschleunigung der Konvergenz gegenüber dem Einzelschrittverfahren erreicht wird.

Im folgenden stellen wir einen Algorithmus für das Relaxationsverfahren auf; für die zugehörige Taschenrechnerprogramme geben wir nur die Änderungen an, die bei den obigen Programmen für das Einzelschrittverfahren vorzunehmen sind.

Algorithmus Eingabe: n, $(a_{ik})_{i,k=1,\ldots,n}$, $(b_i)_{i=1,\ldots,n}$,
σ, $x_i = x_i^{(0)}$ für $i=1,\ldots,n$.
Anzeige: t, $x^{(t)}$ für $t = 1,2,\ldots$.

a_{ik}	b_i	d	i	k	n	σ	t	x	x_i

```
t ← 1
Anzeige t                                  ←
i = 1(1)n
x ← b_i
k = 1(1)n
≠ k : i
d ← a_ii
x ← x - a_ik x_k                           ←
x_i ← (1-σ)x_i + (σ/d)x
Anzeige x_i
t ← t + 1
```

In dem obigen Programm für die Rechner HP-67/97 benutzt man zusätzlich den Datenspeicherplatz R_6 für σ. An die Stelle der Programmschritte 059 bis 064 treten die Anweisungen

c 1 RCL 6 - STO * (i)	$x_i \leftarrow (1-\sigma)x_i$
RCL 6 RCL 0 * RCL 1 ÷ STO + (i)	$x_i \leftarrow x_i + \frac{\sigma}{d}x$
RCL (i) PRTx	Anzeige x_i

Bei dem obigen Programm für die Rechner TI-58/59 kann das Datenregister R_{07} zur Aufnahme von σ dienen. Die Programmschritte 108 bis 116 werden dann ersetzt durch die folgenden Anweisungen.

= ÷ C' (1 - RCL 7) Prd Ind 0	$x_i \leftarrow (1-\sigma)x_i$
RCL 1 * RCL 7 = SUM Ind 0	$x_i \leftarrow x_i + \frac{\sigma}{d}x$
RCL Ind 0 Prt	Anzeige x_i

Jeweils vor Beginn der Rechnung ist σ zu speichern. Im übrigen gilt die obige Anleitung zur Verwendung der Programme unverändert. Es werden n(n+2)+8 Datenspeicherplätze benötigt, falls die volle Koeffizientenmatrix gespeichert wird; für die Rechner HP-67/97 und TI-58 ist $n \leq 3$ möglich, bei TI-59 kann nach 9 Op 17 wieder $n \leq 8$ gewählt werden. Speichert man nur die Teilmatrix $(a_{ik})_{k \leq i}$, so sind $\frac{1}{2}n(n+5)+8$ Datenspeicherplätze erforderlich und als Zeilenzahl n ist wie oben $n \leq 4$ bzw. $n \leq 10$ zulässig.

10. Nichtlineare Gleichungssysteme

Wir betrachten nun nichtlineare Gleichungssysteme mit n Gleichungen für n Unbekannte. Die Methoden zur Bestimmung von Lösungen stellen Verallgemeinerungen der Verfahren für eine nichtlineare Gleichung mit einer Unbekannten dar. Wie schon im Fall linearer Gleichungssysteme ergibt die Methode der sukzessiven Approximation ein Gesamtschrittverfahren und ein zugehöriges Einzelschrittverfahren. Wir befassen uns einführend mit diesen beiden Verfahren und erläutern dann das Einzelschrittverfahren an nichtlinearen Gleichungssystemen, die bei der Differenzenapproximation von Randwertaufgaben gewöhnlicher Differentialgleichungen zweiter Ordnung auftreten.

GESAMT- UND EINZELSCHRITTVERFAHREN

Das nichtlineare Gleichungssystem liege in der Gestalt $x = g(x)$ vor, mit n-komponentigen Vektoren x und g. Ausführlich geschrieben lautet es

$$\begin{aligned} x_1 &= g_1(x_1, x_2, \ldots, x_n), \\ x_2 &= g_2(x_1, x_2, \ldots, x_n), \\ &\vdots \\ x_n &= g_n(x_1, x_2, \ldots, x_n). \end{aligned}$$

Als Lösungen sucht man also Fixpunkte der Abbildung g. Für das Gesamtschrittverfahren $x^{(t+1)} = g(x^{(t)})$ berechnet man eine Folge von Näherungsvektoren gemäß

$$x_j^{(t+1)} = g_j(x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)}), \quad j=1,\ldots,n, \quad t=0,1,2,\ldots .$$

Bei dem Einzelschrittverfahren lautet die Iterationsvorschrift

$$x_1^{(t+1)} = g_1(x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)}),$$
$$x_j^{(t+1)} = g_j(x_1^{(t+1)}, \ldots, x_{j-1}^{(t+1)}, x_j^{(t)}, \ldots, x_n^{(t)}), \quad j=2,\ldots,n,$$

für $t = 0,1,2,\ldots$. Die im vorangegangenen Abschnitt behandelten Gesamt- und Einzelschrittverfahren für lineare Gleichungssysteme ordnen sich hier ein, wenn man wählt

$$g_j(x_1,\ldots,x_n) = \frac{1}{a_{jj}}(b_j - \sum_{\substack{k=1 \\ k\neq j}}^{n} a_{jk}x_k), \quad j = 1,\ldots,n.$$

Bildet g eine abgeschlossene Menge G des n-dimensionalen Zahlenraumes in sich ab, so kann für jeden Vektor $x^{(0)}$ aus G die Folge der iterierten Vektoren $x^{(t+1)} = g(x^{(t)})$ gebildet werden. Ist weiter die Abbildung g kontrahierend,

$$||g(x) - g(y)|| \leq q||x - y||, \quad x,y \in G, \text{ mit } 0 \leq q < 1,$$

dann gibt es eine eindeutig bestimmte Lösung z des nichtlinearen Gleichungssystems $x = g(x)$ in G, und für jeden Anfangsvektor $x^{(0)}$ aus G konvergiert die Folge $x^{(0)}$, $x^{(1)}$, $x^{(2)}$,... des Gesamtschrittverfahrens gegen z mit der a-posteriori- und der a-priori-Fehlerabschätzung

$$||x^{(t)} - z|| \leq \frac{q}{1-q}||x^{(t)} - x^{(t-1)}|| \leq \frac{q^t}{1-q}||x^{(1)} - x^{(0)}||, \quad t=1,2,\ldots .$$

Diese Konvergenzaussage entspricht derjenigen für den Fall einer Unbekannten; an die Stelle des Betrags von reellen Zahlen ist jetzt eine beliebige Norm für Vektoren getreten.

Für das Einzelschrittverfahren gilt dieselbe Aussage über die eindeutige Lösbarkeit des Gleichungssystems und die Konvergenz und Fehlerabschätzung, falls G ein abgeschlossenes n-dimensionales Intervall ist und falls als Vektornorm die Maximumnorm verwendet wird (s. [9] 9.1).

Für eine Abbildung g mit einmal partiell differenzierbaren Komponenten g_j bezeichnen wir die Matrix der partiellen Ableitungen $\frac{\partial g_j}{\partial x_k}$ als Funktionalmatrix g',

$$g'(x) = \frac{\partial g}{\partial x}(x) = \left(\frac{\partial g_j}{\partial x_k}(x) \right)_{j,k=1,\ldots,n}.$$

Ist die Punktmenge G konvex und die Abbildung g auf G erklärt mit stetigen und beschränkten partiellen Ableitungen erster Ordnung, dann gilt

$$||g(x)-g(y)|| \leq L||x-y||, \; x,y \in G, \text{ mit } L = \sup_{w \in G} ||g'(w)||.$$

Hierbei ist auf die Funktionalmatrix g'(w) eine Matrizennorm angewandt, die mit der verwendeten Vektornorm verträglich ist; für L<1 ist dann g eine Kontraktion mit q = L<1. Da im Fall des Einzelschrittverfahrens die Maximumnorm als Vektornorm dient, ist eine verträgliche Matrizennorm zum Beispiel die maximale Zeilensumme.

NICHTLINEARE RANDWERTAUFGABE

Wir behandeln nun nichtlineare Randwertaufgaben bei gewöhnlichen Differentialgleichungen und setzen zur Berechnung von Näherungslösungen das Einzelschrittverfahren ein.

Gegeben sei eine Funktion f, die erklärt ist für (x,y,z) mit $a \leq x \leq b$ und y,z beliebig reell. Gesucht ist eine zweimal stetig differenzierbare Funktion y, die die Differentialgleichung

$$y''(x) = f(x,y(x),y'(x)), \; a \leq x \leq b,$$

erfüllt und gleichzeitig den Randbedingungen

$$y(a) = \alpha, \; y(b) = \beta$$

genügt.

Mit einer natürlichen Zahl N, $h = \frac{b-a}{N}$ und den Gitterpunkten $x_j = a + jh$, $j = 0,\ldots,N$, ermitteln wir Näherungswerte für die Funktionswerte $y(x_j)$ als Lösung $(u_0, u_1, \ldots, u_N)$ der Differenzenapproximation

$$u_0 = \alpha,$$

$$\frac{1}{h^2}(u_{j+1}-2u_j+u_{j-1}) = f(x_j,u_j, \frac{1}{2h}(u_{j+1}-u_{j-1})), \; j = 1,\ldots,N-1,$$

$$u_N = \beta.$$

Hiermit liegt ein nichtlineares Gleichungssystem für die Unbekannten $u_1,\ldots,u_{N-1}$ vor.

Dieses Gleichungssystem lösen wir durch das Einzelschrittverfahren

$$u_0^{(t+1)} = \alpha,$$

$$u_j^{(t+1)} = \frac{1}{2}(u_{j+1}^{(t)}+u_{j-1}^{(t+1)})-\frac{1}{2}h^2 f(x_j,u_j^{(t)},\frac{1}{2h}(u_{j+1}^{(t)}-u_{j-1}^{(t+1)})),$$

$$u_N^{(t+1)} = \beta, \qquad j = 1,\ldots,N-1,$$

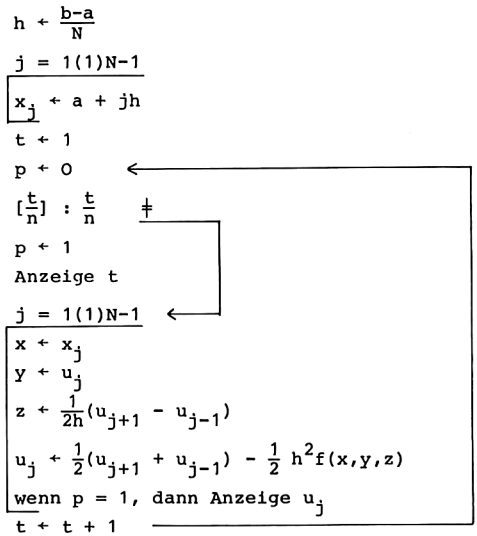
$t = 0,1,2,\ldots$. Für hinreichend kleine h können die obigen Voraussetzungen über die Konvergenz dieses Verfahrens erfüllt werden.

Bei dem Einzelschrittverfahren für lineare Gleichungssysteme hatten wir den Lösungsalgorithmus so gestaltet, daß jeder Näherungsvektor angezeigt wurde. Im Unterschied dazu wird nun in dem folgenden Algorithmus nur jeder n-te Näherungsvektor angezeigt, und die Zahl n wird zu Beginn mit eingegeben.

Algorithmus Eingabe: Unterprogramm für f(x,y,z);
a, b, N;
$u_0^{(0)} = \alpha$, $u_1^{(0)},\ldots,u_{N-1}^{(0)}$, $u_N^{(0)} = \beta$;
n.

Anzeige: t, $u_1^{(t)},\ldots,u_{N-1}^{(t)}$ für t = n, 2n,... .

a	b	h	j	N	n	p	t	u_0	u_1	...	u_N	x_1	...	x_{N-1}	x	y	z



Der Parameter p dient als Steuergröße für die Anzeigeanweisungen. Ist t ein Vielfaches von n, so wird p = 1 gesetzt und es werden die berechneten Komponenten des Näherungsvektors angezeigt. Andernfalls werden die Anzeigeanweisungen für p = 0 übersprungen. In den Programmen verwenden wir hierzu Flag 0. Für die Rechner HP-67/97 setzen wir Flag 0 =p; mit der Abfrage F? 0 wird die Anweisung "Anzeige u_j" ausgeführt, falls Flag 0 gesetzt ist (p = 1), und die Anzeigeanweisung wird übersprungen, falls Flag 0 nicht gesetzt ist (p = 0). In Abweichung davon wird bei den Rechnern TI-58/59 mit der Abfrage Ifflg 0 zu der in der nachfolgenden Anweisung genannten Marke verzweigt, falls Flag 0 gesetzt ist; deshalb setzen wir für diese Rechner Flag 0 = 1-p und erreichen auf diese Weise, daß die Anzeigeanweisungen für p = 0 übersprungen werden.

Für die beiden Schleifen des Algorithmus wollen wir in den Programmen auf die Dsz-Anweisung zurückgreifen. Deshalb arbeiten wir mit dem komplementären Index j' = N-j. Um bei der Berechnung der Gitterpunkte $x_j = a + jh$ in der ersten Schleife nicht außerdem noch den Index j selbst verwenden zu müssen, formulieren wir die Schleife zur Berechnung der Gitterpunkte um in die gleichwertige Form

```
j' = N-1(-1)1
x_j' ← b - j'h
```

Wir berechnen und speichern diese Gitterpunkte bereits im Einleseprogramm, nachdem a,b und N eingegeben sind und h berechnet ist. Dadurch wird erreicht, daß bei Wiederholung des Hauptprogramms mit einer anderen Anfangsnäherung und sonst unveränderten Eingabedaten nicht nochmals dieselben Gitterpunkte ermittelt werden.

Die Größen $u_0, u_1, \ldots, u_N$ werden vom Ende des Datenspeichers her gespeichert, anschließend die Gitterpunkte $x_1, \ldots, x_{N-1}$. Mit der Platzziffer ω des letzten Datenspeicherplatzes R_ω lauten dann die Adressen

$$\text{Adr } u_j = \omega - j = -N + j' + \omega,$$
$$\text{Adr } x_j = \omega - N - j = -2N + j' + \omega.$$

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7	
z	y	x	t	n	N	j'	h	
a	b							

R_{25-2N}	...	R_{23-N}	R_{24-N}	...	R_{23}	R_{24}	R_I
x_{N-1}	...	x_1	u_N	...	u_1	u_0	Adr

$N \leq 8$

Einleseprogramm

001	STO 0 R/S	$R_0 \leftarrow a$, Stop
003	STO 1 - CHS R/S	$R_1 \leftarrow b$, $R_X \leftarrow b-a$, Stop
007	STO 5 STO 6 ÷ STO 7	$R_5 \leftarrow N$, $j' \leftarrow N$, $h \leftarrow \frac{b-a}{N}$
011	1 STO - 6	$j' \leftarrow N-1$
013	LBL 0 b	$R_I \leftarrow$ Adr $x_{j'}$, ⟵
015	RCL 1 RCL 7 RCL 6 * - STO (i)	$x_{j'} \leftarrow b-j'h$
021	6 STO I DSZ (i) GTO 0	$j' \leftarrow j'-1$, wenn $j'>0$ ⟵
025	LBL A RCL 5 STO 6 a	$j' \leftarrow N$, $R_I \leftarrow$ Adr u_0
029	LBL 1 RCL (i) R/S	Anzeigen, Stop ⟵
032	STO (i) DSZ I GTO 1	Speichern, Adr←Adr-1, ⟵

Hauptprogramm

035	LBL B 1 STO 3	$t \leftarrow 1$
038	LBL 2 CLF 0	Flag 0 ← 0 ⟵
040	RCL 3 RCL 4 ÷ FRAC x≠0? GTO 3	wenn $[\frac{t}{n}] \neq \frac{t}{n}$ ⟵
046	STF 0 PRTSPC RCL 3 PRTx	Flag 0 ← 1, Anzeige t
050	LBL 3 RCL 5 1 - STO 6	$j' \leftarrow N-1$ ⟵
055	LBL 4 b RCL (i) STO 2	$x \leftarrow x_j$ ⟵
059	a RCL (i) STO 1	$y \leftarrow u_j$
062	DSZ I RCL (i) ISZ I STO (i)	$u_j \leftarrow u_{j+1}$
066	STO 0	$z \leftarrow u_{j+1}$
067	ISZ I RCL (i) DSZ I STO + (i)	$u_j \leftarrow u_j+u_{j-1}$
071	STO - 0 RCL 7 2 * STO ÷ 0	$z \leftarrow (z-u_{j-1})/(2h)$
076	C RCL 7 x^2 * STO - (i)	$u_j \leftarrow u_j-h^2 f(x,y,z)$
081	2 STO ÷ (i)	$u_j \leftarrow u_j/2$
083	RCL (i) F? 0 PRTx	wenn Flag 0=1:Anzeige u_j
086	6 STO I DSZ (i) GTO 4	$j' \leftarrow j'-1$, wenn $j'>0$ ⟵
090	1 STO + 3 GTO 2	$t \leftarrow t+1$, ⟵

Adressenunterprogramme

```
093   LBL a 1 GTO 5                1 (  ┐    Adr u_j
096   LBL b 2                      2 (  ┘    Adr x_j
098   LBL 5 RCL 5 * CHS            -N)  ←
102   RCL 6 + 24 + STO I           +j'+24, R_I←Adr
108   RTN                          Rücksprung
```

Unterprogramm f(x,y,z)

```
109   LBL C
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	R_{07}	R_{08}	R_{09}
z	y	x	t	n	N	j'	ω	Adr	h
a	b								

$R_{\omega-2N+1}$	...	$R_{\omega-N-1}$	$R_{\omega-N}$	...	$R_{\omega-1}$	R_{ω}
x_{N-1}	...	x_1	u_N	...	u_1	u_0

Einleseprogramm

```
000   STO 0 - R/S                  R_00 ← a, Stop
004   STO 1 = +/- ÷ R/S            R_01←b, R_X←b-a, Stop
010   STO 5 STO 6 = STO 9          R_05←N, j'←N, h←(b-a)/N
017   Op 16 INV Int * 100 = STO 7  R_07 ← ω
028   Op 36                        j' ← N-1
030   Lbl INV B' RCL 1 -           R_08←Adr x_j', R_X←b  ←
036   RCL 6 * RCL 9 = STO Ind 8    x_j' ← b-j'h
044   Dsz 6 INV                    j'←j'-1, wenn j'>0  ┘
047   Lbl A RCL 5 STO 6 A'         j'←N, R_08←Adr u_0
054   Lbl lnx RCL Ind 8 R/S        Anzeigen, Stop  ←
059   STO Ind 8 Dsz 8 lnx          Speichern, Adr←Adr-1,  ┘
```

Hauptprogramm

```
064  Lbl B 1 STO 3                    t ← 1
069  Lbl CE Stflg 0 RCL 3 ÷           Flag 0←1, Rx←t
076  RCL 4 = INV Int CP INV x=t CLR   wenn [t/n] ≠ x/n
085  INV Stflg 0 Adv RCL 3 Prt        Flag 0←0, Anzeige t
092  Lbl CLR RCL 5 STO 6 Op 36        j' ← N-1
100  Lbl x↔t B' RCL Ind 8 STO 2       x ← xj
107  A' RCL Ind 8 STO 1               y ← uj
112  Op 38 RCL Ind 8 Op 28            Rx ← uj+1
118  STO Ind 8 - Op 28 RCL Ind 8      uj←uj+1, Rx←uj-1
125  Op 38 SUM Ind 8                  uj ← uj+uj-1
129  = ÷ RCL 9 ÷ 2 = STO 0            z←(uj+1-uj-1)/(2h)
138  C * RCL 9 x² = INV SUM Ind 08     uj←uj-h²f(x,y,z)
147  2 INV Prd Ind 8                  uj ← uj/2
151  Ifflg 0 x²                       wenn Flag 0 = 1
154  RCL Ind 8 Prt                    Anzeige uj
157  Lbl x² Dsz 6 x↔t                 j'←j'-1, wenn j'>0
162  Op 23 GTO CE                     t ← t+1,
```

Adressenunterprogramme

```
166  Lbl A' ( 1 GTO √x                1 (        Adr uj
172  Lbl B' ( 2                       2 (        Adr xj
176  Lbl √x * RCL 5 +/- +             -N)+
183  RCL 6 + RCL 7 ) STO 8            j'+ω, R08←Adr
191  INV SBR                          Rücksprung
```

Unterprogramm f(x,y,z)

```
192  Lbl C
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe des Unterprogramms C zur Berechnung der rechten Seite der Differentialgleichung: aus x in R_2, y in R_1 und z in R_0 wird f(x,y,z) berechnet und im Anzeigeregister an das rufende Programm übergeben.

3. Eingabe von a, b, N und der Startnäherung $u^{(0)}$:
   RTN bzw. RST, a R/S b R/S N R/S
   $u_0^{(0)} = \alpha$ R/S $u_1^{(0)}$ R/S ... $u_{N-1}^{(0)}$ R/S $u_N^{(0)} = \beta$ R/S.
   Gleichwertig zu Schritt 3 können erst Schritt 3a und danach Schritt 3b ausgeführt werden.

3a Eingabe von a, b und N:
   RTN bzw. RST, a R/S b R/S N R/S.

3b Eingabe der Anfangsnäherung $u^{(0)}$:
   A $u_0^{(0)} = \alpha$ R/S $u_1^{(0)}$ R/S ... $u_{N-1}^{(0)}$ R/S $u_N^{(0)} = \beta$ R/S.

4. Eingabe der Zahl n:
   n STO 4 bzw. n STO 04.

5. Berechnung der Näherungen $u^{(t)}$ und Anzeige von
   t, $u_1^{(t)}, \ldots, u_{N-1}^{(t)}$ für t = n, 2n,...: B.

Mögliche Wiederholungen:

für eine andere Anfangsnäherung $u^{(0)}$ ab Schritt 3b,
für einen anderen Wert von N ab Schritt 3,
für ein anderes Intervall $a \leq x \leq b$ ab Schritt 3,
für eine andere Differentialgleichung ab Schritt 2.
Bei Wiederholungen kann Schritt 4 entfallen, wenn n nicht geändert werden soll.

Wie in den vorherigen Abschnitten kann auch bei diesen Programmen durch die Tastenfolge A R/S ... R/S die Anfangsnäherung $u^{(0)}$ angezeigt werden zur Überprüfung der Eingabe; fehlerhaft eingegebene Werte werden hierbei durch Überschreiben im Anzeigeregister auch im Datenspeicher berichtigt.

Bei den Rechnern HP-67/97 werden 2N+9 Datenspeicherplätze belegt; daher kann man $N \leq 8$ wählen. Wird für die Berechnung der Funktion f durch das Unterprogramm C noch ein Zwischenspeicher benötigt, so kann R_8 dazu dienen.

Für die Rechner TI-58/59 sind bei den obigen Programmen 2N+10 Datenspeicherplätze erforderlich. Bei dem Rechner TI-58 umfaßt nach Einschalten des Gerätes der Programmspeicher 240 Programmschritte und der Datenspeicher 30 Datenregister;

es stehen also 46 Programmspeicherplätze für das Unterprogramm C zur Verfügung, und es ist $N \leq 10$ möglich. Für den Rechner TI-59 ergeben sich keine wesentlichen Beschränkungen an N; nach 9 Op 17 kann $N \leq 40$ gewählt werden.

Für die obige Differenzenapproximation der Randwertaufgabe geben wir anschließend noch die Fassung des Algorithmus an für den Fall a = 0, b = 1. Für diese speziellen Randpunkte vereinfacht sich die Berechnung; es wird $h = \frac{1}{N}$, und es ist nicht mehr erforderlich, die Gitterpunkte x_j, $j = 1,\ldots,N-1$, zu speichern. Dadurch sind in den anschließenden Taschenrechnerprogrammen größere Werte für N zulässig.

Algorithmus

Eingabe: Unterprogramm für f(x,y,z);
N;
$u_0^{(0)} = \alpha$, $u_1^{(0)},\ldots, u_{N-1}^{(0)}$, $u_N^{(0)} = \beta$;
n.

Anzeige: t, $u_1^{(t)},\ldots, u_{N-1}^{(t)}$ für $t = n, 2n, \ldots$.

j	N	n	p	t	u_0	u_1	...	u_N	x	y	z

$t \leftarrow 1$
$p \leftarrow 0$
$[\frac{t}{n}] : \frac{t}{n}$ $\neq$
$p \leftarrow 1$
Anzeige t
$j = 1(1)N-1$
$x \leftarrow \frac{j}{N}$
$y \leftarrow u_j$
$z \leftarrow \frac{N}{2}(u_{j+1} - u_{j-1})$
$u_j \leftarrow \frac{1}{2}(u_{j+1} + u_{j-1}) - \frac{1}{2N^2}f(x,y,z)$
wenn p = 1, dann Anzeige u_j
$t \leftarrow t + 1$

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	R_3	R_4	R_5	R_6	
z	y	x	t	n	N	j'	

R_{24-N}	...	R_{23}	R_{24}	R_I
u_N	...	u_1	u_0	Adr

$N \leq 17$

Einleseprogramm

```
001   LBL A RCL 5 STO 6 a          j'←N, R_I←Adr u_0
005   LBL 1 RCL (i) R/S            Anzeigen, Stop
008   STO (i) DSZ I GTO 1          Speichern, Adr←Adr-1,
```

Hauptprogramm

```
011   LBL B 1 STO 3                t ← 1
014   LBL 2 CLF 0                  Flag 0 ← 0
016   RCL 3 RCL 4 ÷ FRAC x≠0? GTO 3  wenn [t/n] ≠ t/n
022   STF 0 PRTSPC RCL 3 PRTx      Flag 0 ← 1, Anzeige t
026   LBL 3 RCL 5 1 - STO 6        j' ← N-1
031   LBL 4 1 RCL 6 RCL 5 ÷ - STO 2  x←j/N=(N-j')/N
038   a RCL (i) STO 1              y ← u_j
041   DSZ I RCL (i) ISZ I STO (i)  u_j ← u_j+1
045   STO 0                        z ← u_j+1
046   ISZ I RCL (i) DSZ I STO + (i)  u_j ← u_j+u_j-1
050   STO - 0 RCL 5 2 ÷ STO * 0    z←(z-u_j-1)N/2
055   C RCL 5 x^2 ÷ STO - (i)      u_j←u_j-f(x,y,z)/N^2
060   2 STO ÷ (i)                  u_j←u_j/2
062   RCL (i) F? 0 PRTx            wenn Flag 0=1:Anzeige u_j
065   6 STO I DSZ (i) GTO 4        j'←j'-1, wenn j'>0
069   1 STO + 3 GTO 2              t ← t+1,
```

Adressenunterprogramm

```
072   LBL a RCL 6 RCL 5 -          j'-N          Adr u_j
076   24 + STO I                   +24, R_I←Adr
080   RTN                          Rücksprung
```

Unterprogramm f(x,y,z)

```
081   LBL C
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	R_{07}	R_{08}	
z	y	x	t	n	N	j'	ω	Adr	

$R_{\omega-N}$	...	$R_{\omega-1}$	R_{ω}
u_N	...	u_1	u_0

Einleseprogramm

```
000   STO 5                                R_05 ← N
002   Op 16 INV Int * 100 = STO 7          R_07 ← ω
013   Lbl A RCL 5 STO 6 A'                 j'←N, R_08←Adr u_0
020   Lbl lnx RCL Ind 8 R/S                Anzeigen, Stop
025   STO Ind 8 Dsz 8 lnx                  Speichern, Adr←Adr-1,
```

Hauptprogramm

```
030  Lbl B 1 STO 3                    t ← 1
035  Lbl CE Stflg 0 RCL 3 ÷           Flag 0←1, R_X←t
042  RCL 4 = INV Int CP INV x=t CLR   wenn [t/n] ≠ t/n
051  INV Stflg 0 Adv RCL 3 Prt        Flag 0←0, Anzeige t
058  Lbl CLR RCL 5 STO 6 Op 36        j' ← N-1
066  Lbl x↔t 1 - RCL 6 ÷ RCL 5 =      R_X←j/N=(N-j')/N
076  STO 2 A' RCL Ind 8 STO 1         x←j/N, y←u_j
083  Op 38 RCL Ind 8 Op 28            R_X ← u_j+1
089  STO Ind 8 - Op 28 RCL Ind 8      u_j←u_j+1, R_X←u_j-1
096  Op 38 SUM Ind 8                  u_j ← u_j+u_j-1
100  = * RCL 5 ÷ 2 = STO 0            z←(u_j+1-u_j-1)N/2
109  C ÷ RCL 5 x² = INV SUM Ind 08     u_j←u_j-f(x,y,z)/N²
118  2 INV Prd Ind 8                  u_j ← u_j/2
122  Ifflg 0 x²                       wenn Flag 0=1
125  RCL Ind 8 Prt                    Anzeige u_j
128  Lbl x² Dsz 6 x↔t                 j'←j'-1, wenn j'>0
133  Op 23 GTO CE                     t ← t+1,
```

Adressenunterprogramm

```
137  Lbl A' ( RCL 6 - RCL 5 +         j'-N+        Adr u_j
146  RCL 7 ) STO 8                    ω, R_08←Adr
151  INV SBR                          Rücksprung
```

Unterprogramm f(x,y,z)

```
152  LbL C
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe des Unterprogramms C, das aus x in R_2, y in R_1 und z in R_0 den Wert f(x,y,z) berechnet und ihn im Anzeigeregister an das aufrufende Programm übergibt.
3. Eingabeschritte:

3a Eingabe von N
bei HP-67/97: N STO 5,
bei TI-58/59: N RST R/S.

3b Eingabe der Anfangsnäherung $u^{(0)}$:
A $u_0^{(0)}$ = α R/S $u_1^{(0)}$ R/S ... $u_{N-1}^{(0)}$ R/S $u_N^{(0)}$ = β R/S.

4. Eingabe der Zahl n:
n STO 4 bzw. n STO 04.

5. Berechnung der Näherungen $u^{(t)}$ und Anzeige von t, $u_1^{(t)}$,..., $u_{N-1}^{(t)}$ für t = n, 2n,...: B.

Mögliche Wiederholungen:
für eine andere Anfangsnäherung ab Schritt 3b,
für eine andere Wahl von N ab Schritt 3,
für eine andere Differentialgleichung ab Schritt 2.

Wie bei den obigen Programmen ist die Überprüfung der eingegebenen Ausgangsnäherung $u^{(0)}$ möglich, und Schritt 4 kann bei Wiederholungen entfallen, falls n unverändert bleiben soll.

Das Programm für die Rechner HP-67/97 benötigt N+9 Datenspeicherplätze, so daß jetzt $N \leq 17$ möglich wird. Bei den Rechnern TI-58/59 belegen wir N+10 Datenregister. Falls das Unterprogramm C zur Berechnung von f(x,y,z) nicht mehr als 30 Programmschritte umfaßt, so kommt man auf den TI-Rechnern mit 160 Programmschritten aus; mit 4 Op 17 werden in dem Rechner TI-58 gerade 160 Programmschritte und dazu 40 Datenspeicherplätze bereitgestellt, so daß $N \leq 30$ gewählt werden kann. Entsprechend führt 10 Op 17 bei dem Rechner TI-59 zu 160 Programmschritten und 100 Datenregistern, womit vom verfügbaren Datenspeicherplatz her $N \leq 90$ zulässig wird; die Behandlung so großer Gleichungssysteme scheitert jedoch am Rechenzeitbedarf.

Die beiden ersten Programme dieses Abschnitts lassen mit beliebigen Randpunkten a und b eine größere Klasse von Aufgabenstellungen zu. Zuletzt wurden durch die spezielle Wahl a=0 und b=1 sowohl Datenspeicherplätze wie auch Programmschritte eingespart. Zusätzlich könnte auf weitere Datenspeicherplätze verzichtet werden, wenn beispielsweise Eigenschaften der Funktion f bekannt wären. In dem nachfolgenden ersten Beispiel

hängt f nur von z ab, in dem zweiten Beispiel nur von y. Die Berechnung und Bereitstellung der jeweils anderen Argumente für f wäre dann nicht erforderlich. Um die Programme allgemein verwendungsfähig zu halten, haben wir sie jedoch nicht auf spezielle Aufgabenstellungen zugeschnitten.

Beispiele

(i) Die Randwertaufgabe $y''(x) = \sqrt{1 + y'(x)^2}$, $0 \le x \le 1$, $y(0) = 0$, $y(1) = 1$, besitzt als Lösung die Kettenlinie $y(x) = \cosh(x-c)+d$; c und d sind durch die Randbedingungen festgelegt.

Das Unterprogramm C zur Berechnung von $f(x,y,z) = \sqrt{1 + z^2}$ lautet für die Rechner HP-67/97

LBL C RCL 0 x^2 1 + $\sqrt{x}$	$R_X \leftarrow \sqrt{1+z^2}$
RTN	Rücksprung,

für die Rechner TI-58/59

Lbl C (RCL 0 x^2 + 1) $\sqrt{x}$	$R_X \leftarrow \sqrt{1+z^2}$
INV SBR	Rücksprung.

Näherungswerte für die Funktionswerte der Kettenlinie wurden berechnet mit N=5 und der Anfangsnäherung $u_0^{(0)} = 0$, $u_1^{(0)} = 0.2$, $u_2^{(0)} = 0.4$, $u_3^{(0)} = 0.6$, $u_4^{(0)} = 0.8$, $u_5^{(0)} = 1$ und mit n=5 gedruckt.

5.	10.	15.	20.
.1119988864	.0960729188	.0941041203	.0938607231
.2592472625	.2371101278	.2343793804	.2340418688
.4480665397	.4288682573	0.42650083	.4262082349
.6899513632	0.679570413	.6782896764	.6781313784

25.	30.	35.	40.
.0938306316	.0938269114	.0938264514	.0938263946
.2340001432	.2339949847	.2339943469	.2339942681
.4261720623	.4261675903	.4261670374	.4261669691
.6781118085	0.678109389	.6781090899	.6781090529

45.	50.	55.	60.
.0938263875	.0938263867	.0938263866	.0938263866
.2339942583	.2339942571	0.233994257	.2339942569
.4261669606	.4261669596	.4261669594	.4261669594
.6781090483	.6781090478	.6781090477	.6781090477

(ii) Die Randwertaufgabe $y''(x) = \frac{3}{2}y(x)^2$, $0 \le x \le 1$, $y(0) = 4$, $y(1) = 1$, besitzt u.a. die Lösung $y(x) = 4/(1+x)^2$.

Das Unterprogramm C lautet

für die Rechner HP-67/97

LBL C RCL 1 x^2 3 * 2 ÷	$R_X \leftarrow \frac{3}{2}y^2$
RTN	Rücksprung,

für die Rechner TI-58/59

Lbl C (RCL 1 x^2 * 3 ÷ 2)	$R_X \leftarrow \frac{3}{2}y^2$
INV SBR	Rücksprung.

Näherungswerte für die Funktionswerte einer Lösung wurden erhalten mit N=5 und $u_0^{(0)} = 4$, $u_1^{(0)} = \ldots = u_4^{(0)} = 2.5$, $u_5^{(0)} = 1$, sowie mit n=5 für die Druckanweisungen.

5.	10.	15.
2.814233907	2.79483954	2.794632535
2.080327963	2.05809753	2.057860631
1.590790362	1.575340831	1.57517441
1.247948884	1.241428298	1.241358028

20.	25.	30.
2.794630293	2.794630269	2.794630268
2.057858065	2.057858037	2.057858037
1.575172607	1.575172587	1.575172587
1.241357267	1.241357258	1.241357258

11. Eigenwertaufgaben bei Matrizen: Die Potenzmethode

In diesem Kapitel befassen wir uns mit Eigenwertaufgaben bei Matrizen. Die verschiedenen Näherungsmethoden zur Ermittlung von Eigenwerten und Eigenvektoren weisen einen umfangreichen Programm- und Speicherplatzbedarf auf. Im Hinblick auf die Möglichkeiten bei programmierbaren Taschenrechnern beschränken wir uns hier auf die Potenzmethode bei symmetrischen Matrizen. Zuvor erwähnen wir kurz einige Grundlagen, die wir zur Formulierung der Potenzmethode benötigen.

GRUNDLAGEN

Für Vektoren $x = (x_1,\ldots,x_n)$, $y = (y_1,\ldots,y_n)$ mit reellen Komponenten verwenden wir das euklidische Skalarprodukt

$$(x,y) = x_1y_1 + \ldots + x_ny_n$$

und die euklidische Norm

$$||x|| = (x,x)^{1/2} = (x_1^2 + \ldots + x_n^2)^{1/2}.$$

Ist eine Matrix $A = (a_{ik})_{i,k=1,\ldots,n}$ gegeben, so ist für jeden Vektor $x \neq 0$ der Rayleigh-Quotient erklärt als

$$\rho(x) = \frac{(Ax,x)}{(x,x)} .$$

Eine Zahl λ heißt Eigenwert der Matrix $A = (a_{ik})_{i,k=1,\ldots,n}$, wenn es einen Vektor $w \neq 0$ gibt mit der Eigenschaft

$$Aw = \lambda w.$$

w wird dann Eigenvektor zum Eigenwert λ genannt. Ist ein Eigenvektor w der Matrix A gegeben, so ist $\rho(w) = \lambda$ der zugehörige Eigenwert.

Eigenwerte von Matrizen sind gleichwertig charakterisiert durch die Gleichung $\det(A - \lambda E) = 0$ als Nullstellen eines Polynoms n-ten Grades, des sogenannten charakteristischen Polynoms. Jede reelle Matrix A besitzt daher n Eigenwerte $\lambda_1,\ldots,\lambda_n$, die komplexe Zahlen sein können.

Ist die Matrix A symmetrisch, $a_{ik} = a_{ki}$, $i,k = 1,\ldots,n$, so sind alle Eigenwerte von A reell; gleichzeitig gibt es eine orthonormierte Basis des Vektorraumes, die aus zugehörigen Eigenvektoren besteht:

$$Aw^{(i)} = \lambda_i w^{(i)},$$

$$(w^{(i)},w^{(k)}) = \begin{cases} 1 \text{ für } i = k \\ 0 \text{ für } i \neq k \end{cases} \qquad i,k = 1,\ldots,n.$$

Jeder Vektor x kann dann geschrieben werden als Linearkombination

$$x = \sum_{i=1}^{n} (x,w^{(i)})w^{(i)}.$$

DIE POTENZMETHODE

Bei der Potenzmethode wird ein Eigenvektor der gegebenen Matrix durch eine Folge von Näherungsvektoren approximiert, und es werden zugehörige Eigenwertnäherungen bestimmt. Dieses Verfahren wird auch als Vektoriteration oder als von-Mises-Verfahren bezeichnet.

Wir schildern zunächst das Vorgehen bei der Potenzmethode und stellen den zugehörigen Algorithmus auf unter Verwendung der Vektorschreibweise. Anschließend bringen wir die ausführliche Formulierung des Algorithmus, in der also die einzelnen Komponenten der Vektoren aufgeführt sind, sowie die Übertragung in Taschenrechnerprogramme und Beispiele.

Wir betrachten die Potenzmethode für symmetrische Matrizen mit nichtnegativen Eigenwerten. Unter dieser Voraussetzung gelten die folgenden Eigenschaften (s. [9] 10.1).

Für jeden Vektor $x^{(0)}$ mit $Ax^{(0)} \neq 0$ konvergiert die Folge der normierten iterierten Vektoren,

$$y^{(0)} = \frac{1}{||x^{(0)}||} x^{(0)}, \quad y^{(t+1)} = \frac{1}{||Ay^{(t)}||} Ay^{(t)}, \quad t=0,1,2,\ldots,$$

gegen einen normierten Eigenvektor w von A. w ist Eigenvektor zum größten vorkommenden Eigenwert $\lambda>0$ in dem Sinn, daß mit der Anordnung $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \geq 0$ und den zugehörigen Eigenvektoren $w^{(1)},\ldots,w^{(n)}$ die Beziehung besteht

$$w = w^{(k)} \text{ mit } k = \min \{i \mid (x^{(0)},w^{(i)}) \neq 0\}.$$

Mit einer Zahl q, $0 \leq q < 1$, gilt

$$||y^{(t)} - w|| \leq \text{const.} \cdot q^t, \quad t = 0,1,2,\ldots,$$

und die Folge der Rayleigh-Quotienten konvergiert gemäß

$$|\lambda - \rho(y^{(t)})| \leq \text{const.} \cdot q^{2t}, \quad t = 0,1,2,\ldots .$$

Außerdem konvergieren die Zahlen $||Ay^{(t)}||$ monoton gegen λ,

$$||Ay^{(t)}|| \leq ||Ay^{(t+1)}||, \quad ||Ay^{(t)}|| \to \lambda \text{ für } t \to \infty.$$

Ist eine symmetrische Matrix A gegeben, für die die Eigenschaft "nichtnegative Eigenwerte" nicht erfüllt oder nicht bekannt ist, so kann man das genannte Verfahren auf die Matrix A^2 anwenden, da diese Matrix die geforderten Voraussetzungen erfüllt. In diesem Fall erhält man einen Eigenvektor w von A^2 mit zugehörigem Eigenwert μ, für den mit einem Eigenwert λ von A gilt: $\mu = \lambda^2$. Bildet man die Vektoren

$$w + Aw = z_+, \quad w - Aw = z_-,$$

so ist im Fall $z_+ \neq 0$ die Zahl $|\lambda|$ Eigenwert von A zum Eigenvektor z_+, für $z_- \neq 0$ ist $-|\lambda|$ Eigenwert von A zum Eigenvektor z_-.

Die Potenzmethode ist auch für nichtsymmetrische Matrizen anwendbar, wenn andere geeignete Voraussetzungen erfüllt sind. Gibt es zum Beispiel eine Basis des Vektorraumes, die aus Eigenvektoren von A besteht, ist der betragsgrößte Eigenwert λ von A positiv und besitzt der Startvektor $x^{(0)}$ eine Komponente in Richtung des Eigenvektors w zum Eigenwert λ, so konvergiert die Folge der normierten iterierten Vektoren gegen w (s.[6]5.3).

Mit dem folgenden Algorithmus berechnen wir die Folge der normierten iterierten Vektoren und die zugehörigen Rayleigh-Quotienten als Eigenwertnäherungen. Der Algorithmus endet, wenn sich zwei aufeinanderfolgende normierte iterierte Vektoren in der euklidischen Norm um weniger als ε unterscheiden.

Algorithmus

Voraussetzung: A sei symmetrische Matrix mit nichtnegativen Eigenwerten, $x^{(0)} \neq 0$

Eingabe: n, $(a_{ik})_{i,k=1,\ldots,n}$, $x_i = x_i^{(0)}$ für $i=1,\ldots,n$, ε

Anzeige: $t, y^{(t)}, \rho(y^{(t)})$ für $t = 0$; dann (a) oder (b).

(a) Fehlermeldung und Stop, falls $Ax^{(0)} = 0$; dieser Fall wird angenommen, falls $||Ax^{(0)}|| < \varepsilon ||x^{(0)}||$. Der Algorithmus kann mit einem anderen Startvektor wiederholt werden.

(b) $t, y^{(t)}, \rho(y^{(t)})$ für $t=1,2,\ldots,j$; j ist die kleinste Zahl mit $||y^{(j)} - y^{(j-1)}|| < \varepsilon$.

In der Vektorschreibweise hat der Algorithmus die folgende Gestalt.

a_{ik}	ε	n	r	s	t	x_i	y_i	z_i

	Erläuterungen
$t \leftarrow 0$	
Anzeige t	
$s \leftarrow \|\|x\|\|$	$s = \|\|x^{(0)}\|\|$
$y \leftarrow \frac{1}{s} x$	$y = \frac{1}{\|\|x^{(0)}\|\|} x^{(0)} = y^{(0)}$
Anzeige y	
$x \leftarrow Ay$	$x = Ay^{(0)}$
$r \leftarrow (x,y)$	$r = (Ay^{(0)}, y^{(0)}) = \rho(y^{(0)})$
Anzeige r	
$s \leftarrow \|\|x\|\|$	$s = \|\|Ay^{(0)}\|\| = \frac{1}{\|\|x^{(0)}\|\|} \|\|Ax^{(0)}\|\|$
$s : \varepsilon \quad \geq$	
Fehlermeldung	
Stop	
$t \leftarrow t + 1$	
Anzeige t	
$z \leftarrow \frac{1}{s} x$	$z = \frac{1}{\|\|Ay^{(t-1)}\|\|} Ay^{(t-1)} = y^{(t)}$
Anzeige z	
$x \leftarrow z - y$	$x = y^{(t)} - y^{(t-1)}$
$y \leftarrow z$	$y = y^{(t)}$
$s \leftarrow \|\|x\|\|$	$s = \|\|y^{(t)} - y^{(t-1)}\|\|$
$x \leftarrow Ay$	$x = Ay^{(t)}$
$r \leftarrow (x,y)$	$r = (Ay^{(t)}, y^{(t)}) = \rho(y^{(t)})$
Anzeige r	
$< s : \varepsilon$	$\|\|y^{(t)} - y^{(t-1)}\|\| : \varepsilon$
$s \leftarrow \|\|x\|\|$	$s = \|\|Ay^{(t)}\|\|$
Stop	

Nun wird der Algorithmus ausführlich formuliert mit Hilfe der Unterprogramme $s \leftarrow ||x||$, $x \leftarrow Ay$ und $r \leftarrow (x,y)$. Hierbei zeigt es sich, daß an die Stelle des Hilfsvektors $z = (z_1,\ldots,z_n)$ eine einzelne Variable treten kann, die wir u nennen.

Hauptprogramm Potenzmethode

a_{ik}	ε	i	k	n	r	s	t	u	v	x_i	y_i

```
t ← 0
Anzeige t
s ← ||x||
i = 1(1)n
  y_i ← (1/s) x_i
  Anzeige y_i
x ← Ay
r ← (x,y)
Anzeige r
s ← ||x||
s : ε  ≥
Fehlermeldung
Stop
t ← t + 1
Anzeige t
i = 1(1)n
  u ← (1/s) x_i
  Anzeige u
  x_i ← u - y_i
  y_i ← u
s ← ||x||
x ← Ay
r ← (x,y)
Anzeige r
< s : ε
  s ← ||x||
→Stop
```

Unterprogramm s ← ||x||

i	n	s	x_i

```
s ← 0
i = 1(1)n
  s ← s + x_i^2
s ← √s
Rücksprung
```

Unterprogramm x ← Ay

a_{ik}	i	k	n	v	x_i	y_k

```
i = 1(1)n
  v ← 0
  k = 1(1)n
    v ← v + a_ik y_k
  x_i ← v
Rücksprung
```

Unterprogramm r ← (x,y)

i	n	r	x_i	y_i

```
r ← 0
i = 1(1)n
  r ← r + x_i y_i
Rücksprung
```

Soll nach der Anzeige der Endergebnisse $j, y^{(j)}, \rho(y^{(j)})$ die Rechnung für eine kleinere Genauigkeitsabfrage ε fortgesetzt werden, so startet man nach Abänderung von ε die

weitere Rechnung ab der vorletzten Zeile des Algorithmus. Ist man nur daran interessiert, die Endergebnisse angezeigt zu erhalten, so entfernt man die obigen Anzeigeanweisungen sowie die Anweisungen r←(x,y) zur Berechnung der Rayleigh-Quotienten aus dem Algorithmus; anstelle der letzten Stop-Anweisung muß es dann heißen

```
Anzeige t
i = 1(1)n
Anzeige y_i
r ← (x,y)
Anzeige r
Stop
```

In den folgenden Taschenrechnerprogrammen speichern wir die Teilmatrix $(a_{ik})_{k\le i}$ der symmetrischen Matrix A spaltenweise vom Ende des Datenspeichers her, anschließend die Komponenten der Vektoren x und y. Mit der Adresse ω des letzten Datenspeicherplatzes und mit den komplementären Indizes $i' = n + 1 - i$, $k' = n + 1 - k$ lauten dann die Adressen dieser Speicherplätze für $i,k = 1,\ldots,n$:

$$\begin{aligned}
\text{Adr } a_{ik} &= \omega - (i-1 + \tfrac{1}{2}(k-1)(2n-k)), \qquad k\le i,\\
&= \tfrac{1}{2}k'(k'-1) + i' - (n+1)\tfrac{n}{2} + \omega, \quad k'\ge i',\\
\text{Adr } x_i &= \omega - \tfrac{1}{2}n(n+1) - (i-1) = i' - (n+3)\tfrac{n}{2} + \omega,\\
\text{Adr } y_i &= \omega - \tfrac{1}{2}n(n+3) - (i-1) = i' - (n+5)\tfrac{n}{2} + \omega.
\end{aligned}$$

Es werden Unterprogramme zur Berechnung der Adressen von a_{ik}, x_i, y_i und y_k benötigt.

Programm für HP-67/97

Datenspeicher

R_0	R_1	R_2	R_3	R_4	R_5	R_6	
i'	k'	n	r	s	t	ε	
			v				

$n \leq 4$

$R_{25-(n+5)n/2}$	...	$R_{24-(n+3)n/2}$	$R_{25-(n+3)n/2}$	...
y_n	...	y_1	x_n	...

$R_{24-(n+1)n/2}$	$R_{25-(n+1)n/2}$	...	R_{23}	R_{24}	R_I
x_1	a_{nn}	...	a_{21}	a_{11}	Adr

Einleseprogramme

```
001   STO 2 R/S                    R2 ← n, Stop
003   STO 6                        R6 ← ε
004   LBL A RCL 2 STO O STO 1      i' ← n, k' ← n
008   a GTO O                      RI ← Adr a11'
010   LBL B RCL 2 STO O b          i' ← n, RI ← Adr x1
014   LBL O RCL (i) R/S            Anzeigen, Stop
017   STO (i) DSZ I GTO O          Speichern, Adr←Adr-1,
```

Hauptprogramm

```
020   LBL C 0 STO 5 PRTx                t ← 0, Anzeige t
024   D RCL 2 STO 0                     s ← ||x||, i' ← n
027   LBL 1 b RCL (i) RCL 4 ÷           R_X ← x_i/s
032   c R↓ STO (i) PRTx                 y_i←x_i/s, Anzeige y_i
036   0 STO I DSZ (i) GTO 1             i'←i'-1, wenn i'>0
040   E e RCL 3 PRTx                    x←Ay,r←(x,y),Anzeige r
044   D RCL 6 x≤y? GTO 2                s←||x||, wenn ε≤s
048   CHS √x                            Fehlermeldung, Stop
050   LBL 2 PRTSPC 1 STO + 5            t ← t + 1
054   RCL 5 PRTx RCL 2 STO 0            Anzeige t, i' ← n
058   LBL 3 b RCL (i) RCL 4 ÷ PRTx      u←x_i/s, Anzeige u
064   c R↓ RCL (i) x↔y STO (i)          x_i←u-y_i und y_i←u
069   R↓ - b R↓ STO (i)
074   0 STO I DSZ (i) GTO 3             i'←i'-1, wenn i'>0
078   D E                               s←||x||, x←Ay
080   e RCL 3 PRTx                      r←(x,y), Anzeige r
083   RCL 4 RCL 6 x>y? R/S               wenn ε>s, dann Stop
087   D GTO 2                           s ← ||x||,
```

Unterprogramm s ← ||x||

```
089   LBL D 0 STO 4 RCL 2 STO 0         s ← 0, i' ← n
094   LBL 4 b RCL (i) x^2 STO + 4       s ← s + x_i^2
099   0 STO I DSZ (i) GTO 4             i'←i'-1, wenn i'>0
103   RCL 4 √x STO 4                    R_X ← √s, s ← √s
106   RTN                               Rücksprung
```

Unterprogramm x ← Ay

```
107   LBL E RCL 2 STO 0                 i' ← n
110   LBL 5 0 STO 3 RCL 2 STO 1         v ← 0, k' ← n
115   LBL 6 a RCL (i)                   R_X ← a_ik
118   d R↓ RCL (i) * STO + 3            v ← v + a_ik y_k
123   1 STO I DSZ (i) GTO 6             k'←k'-1, wenn k'>0
127   b RCL 3 STO (i)                   x_i ← v
130   0 STO I DSZ (i) GTO 5             i'←i'-1, wenn i'>0
134   RTN                               Rücksprung
```

Unterprogramm r ← (x,y)

```
135   LBL e 0 STO 3 RCL 2 STO 0        r ← 0, i' ← n
140   LBL 7 b RCL (i)                  R_X ← x_i   ←──┐
143   c R↓ RCL (i) * STO + 3           r ← r + x_i y_i │
148   0 STO I DSZ (i) GTO 7            i'←i'-1, wenn i'>0 ─┘
152   RTN                              Rücksprung
```

Adressenunterprogramme

```
153   LBL a RCL 0 RCL 1 x≤y? x↔y       wenn k'≤i', dann R_X↔R_Y
158   ENTER ENTER 1 - * 2 ÷            k'(k'-1)/2
165   + 1 GTO 8                        +i'-(1  ──→  Adr a_ik
168   LBL b RCL 0 3 GTO 8              i'-(3   ──→  Adr x_i
172   LBL c RCL 0 5 GTO 8              i'-(5   ──→  Adr y_i
176   LBL d RCL 1 5                    k'-(5        Adr y_k
179   LBL 8 RCL 2 + RCL 2 * 2 ÷ -      +n)n/2  ←──┘
187   24 + STO I                       +24, R_I ← Adr
191   RTN                              Rücksprung
```

Programm für TI-58/59

Datenspeicher	R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	R_{07}	
	i'	k'	n	Adr	s	t	ε	ω	

$R_{\omega-(n+5)n/2+1}$	...	$R_{\omega-(n+3)n/2}$	$R_{\omega-(n+3)n/2+1}$	...
y_n	...	y_1	x_n	...

$R_{\omega-(n+1)n/2}$	$R_{\omega-(n+1)n/2+1}$	...	$R_{\omega-1}$	R_{ω}
x_1	a_{nn}	...	a_{21}	a_{11}

Einleseprogramme

```
000   STO 2 R/S                          R02 ← n, Stop
003   STO 6                              R06 ← ε
005   Op 16 INV Int * 100 = STO 7        R07 ← ω
016   Lbl A RCL 2 STO 0 STO 1            i' ← n, k' ← n
024   A' GTO INV                         R03 ← Adr a11'
027   Lbl B RCL 2 STO 0 B'               i' ← n, R03 ← Adr x1
034   Lbl INV RCL Ind 3 R/S              Anzeigen, Stop
039   STO Ind 3 Dsz 3 INV                Speichern, Adr←Adr-1,
```

Hauptprogramm

```
044   Lbl C 0 STO 5 Prt                  t ← 0, Anzeige t
050   D RCL 2 STO 0                      s ← ||x||, i' ← n
055   Lbl lnx B' RCL Ind 3 ÷             RX ← xi
061   C' RCL 4 = STO Ind 3 Prt           yi←xi/s, Anzeige yi
068   Dsz 0 lnx                          i'←i'-1, wenn i'>0
071   E E' Prt                           x←Ay, r←(x,y), Anzeige r
074   RCL 6 x↔t D x≥t CE                 s←||x||, wenn s≥ε
080   +/- √x R/S                         Fehlermeldung, Stop
083   Lbl CE Adv Op 25 RCL 5 Prt         t←t+1, Anzeige t
091   RCL 2 STO 0                        i' ← n
095   Lbl CLR B' RCL Ind 3 ÷             RX ← xi
101   RCL 4 = Prt                        u←xi/s, Anzeige u
105   - x↔t C' x↔t Exc Ind 3             xi←u-yi und yi←u
111   + B' 0 = STO Ind 3
117   Dsz 0 CLR                          i'←i'-1, wenn i'>0
120   D E                                s←||x||, x←Ay
122   E' Prt                             r←(x,y), Anzeige r
124   RCL 6 x↔t RCL 4 x≥t x↔t            wenn s≥ε
131   R/S                                Stop
132   Lbl x↔t D GTO CE                   s ← ||x||,
```

Unterprogramm s ← ||x||

```
137  Lbl D RCL 2 STO OO O          i' ← n, s ← O
144  Lbl x² + B' RCL Ind 3 x²      s ← s + x_i²   ←┐
151  Dsz O x²                      i'←i'-1, wenn i'>O ┘
154  = √x STO 4                    R_X ← √s, s ← √s
158  INV SBR                       Rücksprung
```

Unterprogramm x ← Ay

```
159  Lbl E RCL 2 STO O             i' ← n
165  Lbl √x RCL 2 STO 1            (v←O), k'←n   ←┐
171  Lbl 1/x A' RCL Ind 3 *        R_X ← a_ik    ←┐
177  D' RCL Ind 3 +                v←v+a_ik y_k   │
181  Dsz 1 1/x                     k'←k'-1, wenn k'>O ┘
184  B' O = STO Ind 3              x_i ← v        │
189  Dsz O √x                      i'←i'-1, wenn i'>O ┘
192  INV SBR                       Rücksprung
```

Unterprogramm r ← (x,y)

```
193  Lbl E' RCL 2 STO OO O         i' ← n, r ← O
200  Lbl STO + B' RCL Ind 3 *      R_X ← x_i     ←┐
207  C' RCL Ind 3                  r ← r + x_i y_i │
210  Dsz O STO                     i'←i'-1, wenn i'>O ┘
213  = INV SBR                     Rücksprung
```

Adressenunterprogramme

```
215  Lbl A' ( RCL O x↔t RCL 1      R_X ← k', R_T ← i'
223  x≥t RCL                       wenn k'≥i'  ──┐
225  x↔t                           R_X↔R_T       │
226  Lbl RCL * ( CE - 1 ) ÷ 2 +    k'(k'-1)/2+  ←┘
237  x↔t - ( 1 GTO SUM             i'-(1  →  Adr a_ik
243  Lbl B' ( RCL O - ( 3 GTO SUM  i'-(3  →  Adr x_i
253  Lbl C' ( RCL O - ( 5 GTO SUM  i'-(5  →  Adr y_i
263  Lbl D' ( RCL 1 - ( 5          k'-(5     Adr y_k
271  Lbl SUM + RCL 2 ) * RCL 2 ÷ 2 +n)n/2 ←┘
282  + RCL 7 ) STO 3               +ω, R_O3 ← Adr
288  INV SBR                       Rücksprung
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der Zeilenzahl n, der Genauigkeitsschranke ε, der Matrix $(a_{ik})_{k\leq i}$ und des Startvektors $x^{(0)} \neq 0$:
   RTN bzw. RST, n R/S ε R/S
   a_{11} R/S a_{21} R/S ... a_{nn} R/S
   $x_1^{(0)}$ R/S $x_2^{(0)}$ R/S ... $x_n^{(0)}$ R/S.
   Gleichwertig zu Schritt 2 können erst Schritt 2a und danach unabhängig voneinander die Schritte 2b und 2c ausgeführt werden.

2a Eingabe von n und ε:
   RTN bzw. RST, n R/S ε R/S.

2b Eingabe der Matrix $(a_{ik})_{k\leq i}$:
   A a_{11} R/S a_{21} R/S ... a_{nn} R/S.

2c Eingabe des Startvektors $x^{(0)} \neq 0$:
   B $x_1^{(0)}$ R/S $x_2^{(0)}$ R/S ... $x_n^{(0)}$ R/S.

3. Start der Rechnung: C. Es folgt Anzeige t=0, $y^{(0)}$, $\rho(y^{(0)})$; dann (a) oder (b).
   (a) Fehlermeldung und Stop, falls $||Ax^{(0)}|| < \varepsilon ||x^{(0)}||$, was als $Ax^{(0)} = 0$ gewertet wird.
   (b) Anzeige t, $y^{(t)}$, $\rho(y^{(t)})$ für t = 1,2,...,j; hierbei ist j die kleinste Zahl mit $||y^{(j)}-y^{(j-1)}|| < \varepsilon$.
4. Soll nach Beendigung von Schritt 3(b) die Rechnung für eine kleinere Genauigkeitsschranke fortgesetzt werden, so speichert man den neuen Wert für ε gemäß
   ε STO 6 bzw. ε STO 06
   und betätigt dann die Taste R/S.

Mögliche Wiederholungen:
ab Schritt 2c für einen anderen Startvektor $x^{(0)}$,
ab Schritt 2b für eine andere n-zeilige Matrix,
ab Schritt 2a für eine andere Wahl von n.

Auch hier ist wieder die Kontrolle und Berichtigung der eingegebenen Werte möglich durch die Wiederholung der Tastenfolgen gemäß Schritt 2b und 2c.

Auf den Rechnern HP-67/97 werden $\frac{1}{2}n(n+5)+8$ Datenspeicherplätze belegt, so daß die Potenzmethode für Matrizen mit bis zu 4x4 Elementen gerechnet werden kann. r und v sind gemeinsam gespeichert, da die Werte dieser Variablen nicht gleichzeitig verwendet werden. Die Adressenunterprogramme b, c, d belegen den Stapelspeicher jeweils nur bis zur dritten Ebene; daher kann bei Aufruf dieser Unterprogramme noch eine weitere Zahl im Stapelspeicher aufbewahrt werden, was in den Unterprogrammen E und e und an mehreren Stellen des Hauptprogramms benutzt wird. Insbesondere bei den Programmspeicherzeilen 064 bis 070 wird durch den Aufruf des Adressenunterprogramms c die Größe u bis in die vierte Ebene des Stapelspeichers angehoben, so daß nach dem Rücksprung aus dem Unterprogramm c die zweite bis vierte Ebene des Stapelspeichers mit u belegt sind; dadurch braucht für u kein eigenes Datenregister reserviert zu werden.

Bei den Rechnern TI-58/59 ist es infolge der algebraischen Rechenlogik nicht erforderlich, Datenspeicherplatz für r, u oder v zu reservieren. Trotz der Speicherung von ω werden daher auch bei diesen Rechnern nur $\frac{1}{2}n(n+5)+8$ Datenspeicherplätze benötigt. Mit 8 Op 17 stehen auf dem Rechner TI-59 insgesamt 320 Programmschritte und 80 Datenregister bereit, so daß $n \leq 9$ gewählt werden kann.

Für den Rechner TI-58 führt 2 Op 17 zu 320 Programmschritten und 20 Datenspeicherplätzen, womit $n \leq 3$ zulässig ist. Verzichtet man jedoch auf das Einleseprogramm und auf die Berechnung und Anzeige der Rayleigh-Quotienten, so entfallen die Programmschritte 000 bis 043 und 193 bis 214. Man kann dann anstelle der Rayleigh-Quotienten $\rho(y^{(t)})$ die Normen $||Ay^{(t)}||$ als Eigenwertnäherungen verwenden; hierzu ersetzt

man die Programmzeilen 120 bis 123 durch D STO 8 E D Prt und 127 bis 128 durch RCL 8, so daß $||x|| = ||Ay||$ berechnet und angezeigt wird. Die Anweisung 134 D kann dann wegfallen; entsprechend werden die Programmzeilen 072 durch D und 077 durch RCL 4 ersetzt. Mit diesen Änderungen wird auf dem Rechner TI-58 die Potenzmethode mit $n \leq 4$ möglich, wenn man die Standardaufteilung des Speichers in 240 Programmschritte und 30 Datenregister verwendet.

Beispiele

Die Matrix $A = \begin{pmatrix} 2.2 & 0.2 & 1.2 & 1.4 \\ 0.2 & 1.4 & 0.4 & -0.6 \\ 1.2 & 0.4 & 2.8 & 0.8 \\ 1.4 & -0.6 & 0.8 & 2.6 \end{pmatrix}$

besitzt die Eigenwerte 4.8, 2.4, 1.2, 0.6; der Vektor (1, 0, 1, 1) ist Eigenvektor zum Eigenwert 4.8. Mit $x^{(0)} = (1, 1, 1, 1)$ als Startvektor wurden Rechnungen durchgeführt. Für $\varepsilon = 10^{-2}$ wurde

$$j = 6, \quad y^{(6)} = \begin{matrix} 0.577388687 \\ 0.003054710 \\ 0.580302444 \\ 0.574336179 \end{matrix}, \quad \rho(y^{(6)}) = 4.799934873 \ ;$$

Fortsetzung der Rechnung mit $\varepsilon = 10^{-4}$ gemäß Punkt 4 der Anleitung zur Verwendung der Programme ergab

$$j = 12, \ y^{(12)} = \begin{matrix} 0.577350279 \\ 0.000646996 \\ 0.577397241 \\ 0.577303283 \end{matrix}, \quad \rho(y^{(12)}) = 4.799999987 \ .$$

Die Matrix $B = \begin{pmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{pmatrix}$

besitzt die Eigenwerte 10, 5, 2, 1, mit dem Vektor (2, 2, 1, 1) als Eigenvektor zum Eigenwert 10. Mit dem Startvektor $x^{(0)} = (1, 2, 3, 4)$ und $\varepsilon = 10^{-2}$ entstand

$$j = 7, \quad y^{(7)} = \begin{matrix} 0.630351295 \\ 0.630351323 \\ 0.320400105 \\ 0.320403218 \end{matrix}, \quad \rho(y^{(7)}) = 9.999781512 ;$$

Fortsetzung der Rechnung mit $\varepsilon = 10^{-4}$ führte zu

$$j = 14, \quad y^{(14)} = \begin{matrix} 0.632439200 \\ 0.632439200 \\ 0.316260429 \\ 0.316260429 \end{matrix}, \quad \rho(y^{(14)}) = 9.999999986 .$$

12. Die Wärmeleitungsgleichung

Die Differenzenapproximation für die Anfangs-Randwert-Aufgabe der Wärmeleitungsgleichung dient uns in diesem Kapitel als Beispiel für die numerischen Methoden bei partiellen Differentialgleichungen. Wir behandeln eine Approximation, bei der die Näherungsfunktionswerte jeder Zeitzeile durch explizite Gleichungen gegeben werden. Dieses Verfahren ist nur für verhältnismäßig kleine Zeit-Schrittweiten stabil. Andere Näherungsverfahren, die ohne Einschränkungen an die Wahl der Schrittweiten stabil sind, entstehen bei impliziten Approximationen; dort sind dann die Funktionswerte in den Zeitzeilen Lösungen linearer Gleichungssysteme.

Bei der Anfangs-Randwert-Aufgabe für die Wärmeleitungsgleichung in einer Ortsveränderlichen ist zu gegebenen Funktionen f, g_0, g_1 eine zweimal stetig differenzierbare Funktion u gesucht, die der Differentialgleichung

$$\frac{\partial^2}{\partial x^2}u(x,t) = \frac{\partial}{\partial t}u(x,t),\ 0\leq x\leq 1,\ t\geq 0,$$

genügt und gleichzeitig die Anfangsbedingung

$$u(x,0) = f(x),\ 0<x<1,$$

und die Randbedingungen

$$u(0,t) = g_0(t),\quad u(1,t) = g_1(t),\ t\geq 0,$$

erfüllt.

Sind M und N natürliche Zahlen, so können mit den Schrittweiten $h = \frac{1}{M}$ in x-Richtung und $k = \frac{1}{N}$ in t-Richtung Differenzenquotienten zur Approximation der partiellen Ableitungen verwendet werden. An die Stelle der Differentialgleichung tritt dann ein System von Differenzengleichungen, dessen Lösung als Näherungen für die Werte der gesuchten Funktion u an den Gitterpunkten dient.

Wir befassen uns mit den Differenzengleichungen

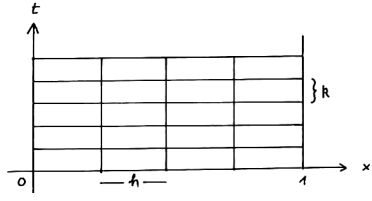
$$\frac{1}{h^2}(v(x+h,t) - 2v(x,t) + v(x-h,t)) = \frac{1}{k}(v(x,t+k) - v(x,t))$$

für $x = ih$, $i = 1,\ldots,M-1$, und $t = jk$, $j = 0,1,2,\ldots$. Bei Kenntnis der Werte einer Näherungslösung zur Zeit t können die Funktionswerte für die Zeit t+k berechnet werden gemäß

$$v(x,t+k) = qv(x+h,t) + (1-2q)v(x,t) + qv(x-h,t),$$

mit der Abkürzung $q = k/h^2$.

Für die Funktionswerte $v_{ij} = v(ih,jk)$ an den Gitterpunkten entsteht zusammen mit der Anfangsbedingung und den Randbedingungen das lineare Gleichungssystem



$$v_{i,0} = f(ih), \qquad i = 1,\ldots,M-1,$$

und

$$v_{0,j} = g_0(jk),$$

$$v_{i,j} = qv_{i+1,j-1} + (1-2q)v_{i,j-1} + qv_{i-1,j-1}, \quad i=1,\ldots,M-1,$$

$$v_{M,j} = g_1(jk),$$

für $j = 1,2,\ldots$.

Diese explizite Differenzenapproximation der Anfangs-Randwert-Aufgabe für die Differentialgleichung ist numerisch stabil für $q \leq \frac{1}{2}$, d.h. für $k \leq \frac{1}{2}h^2$. In der Zeit-Richtung sind also nur verhältnismäßig kleine Schrittweiten zugelassen.

Der anschließende Algorithmus gibt an, wie mit Hilfe gegebener Funktionen f für die Anfangsbedingung und g_0, g_1 für die Randbedingungen die Funktionswerte der zugehörigen Näherungslösung berechnet werden. Aus den Näherungswerten $v_0,\ldots,v_M$ für eine Zeitzeile werden die Werte $w_0,\ldots,w_M$ für die nächste Zeitzeile berechnet. Für jede n-te Zeitzeile werden die Werte angezeigt.

Algorithmus

Eingabe: Unterprogramme
für die Anfangsbedingung f(x)
und die Randbedingungen $g_0(t)$, $g_1(t)$;
M, N, n.

Anzeige: j, v_{0j}, v_{1j},...,v_{Mj} für
$j = 0, n, 2n, \ldots$.

i	j	M	N	n	p	q	v_0	v_1	...	v_M	w_0	w_1	...	w_M

$q \leftarrow \frac{M^2}{N}$
$j \leftarrow 0$
Anzeige j
$v_0 \leftarrow g_0(0)$
Anzeige v_0
i = 1(1)M-1
$v_i \leftarrow f(\frac{i}{M})$
Anzeige v_i
$v_M \leftarrow g_1(0)$
Anzeige v_M
$j \leftarrow j + 1$
$p \leftarrow 0$
$[\frac{j}{n}] : \frac{j}{n}$ ≠
$p \leftarrow 1$
Anzeige j
$w_0 \leftarrow g_0(\frac{j}{N})$
wenn p=1, dann Anzeige w_0
i = 1(1)M-1
$w_i \leftarrow q(v_{i-1}+v_{i+1}-2v_i)+v_i$
wenn p=1, dann Anzeige w_i
$w_M \leftarrow g_1(\frac{j}{N})$
wenn p=1, dann Anzeige w_M
i = 0(1)M
$v_i \leftarrow w_i$

In diesem Algorithmus dient wieder p als Steuergröße für die Anzeige-Anweisungen. Ist die Nummer j der Zeitzeile ein Vielfaches von n, so wird p=1 gesetzt, und es werden die Funktionswerte der Näherungslösung für diese Zeitzeile angezeigt; andernfalls werden die Anzeige-Anweisungen für p=0 übersprungen. Wie in den Programmen in Kapitel 10 wählen wir Flag 0=p für die Rechner HP-67/97 und Flag 0=1-p bei den Rechnern TI-58/59.

Die Programmschleifen i = 1(1)M-1 steuern wir mit Hilfe des komplementären Index i' = M-i und der Dsz-Anweisung.

In den Rechnern HP-67/97 speichern wir die Funktionswerte der Näherungslösung $v_0,\ldots,v_M$ für eine Zeitzeile in den Datenregistern $R_M,\ldots,R_0$; für die daraus berechneten Funktionswerte $w_0,\ldots,w_M$ für die nächste Zeitzeile benutzen wir die Sekundärspeicher $R_{sM},\ldots,R_{s0}$. Das Umspeichern $v_i \leftarrow w_i$ für i = 0,1,...,M am Ende des Wiederholungsbereichs erreichen wir dann durch die Anweisung P↔S zum Vertauschen der Inhalte der Primär- und Sekundärspeicher. Bei den Rechnern TI-58/59 speichern wir die Komponenten $v_0,\ldots,v_M$, $w_0,\ldots,w_M$ vom Ende des Datenspeichers her, so daß mit der Platzziffer ω des letzten Datenspeicherplatzes die Adressen gelten

$$\text{Adr } v_i = \omega - i = -M + i' + \omega,$$

$$\text{Adr } w_i = \omega - (M+1) - i = -(2M+1) + i' + \omega.$$

Programm für HP-67/97

Datenspeicher

R_A	R_B	R_C	R_D	R_E	R_I
j	M	N	n	q	i'=Adr

$M \leq 9$

R_0	...	R_{M-1}	R_M		R_{s0}	...	$R_{s,M-1}$	R_{sM}
v_M	...	v_1	v_0		w_M	...	w_1	w_0

Einleseprogramm

```
001   STO B x^2 R/S                    R_B ← M, Stop
004   STO C ÷ STO E R/S                R_C ← N, R_E ← q, Stop
008   STO D R/S                        R_D ← n, Stop
```

Hauptprogramm

```
010   LBL A RCL B STO I 0 STO A        i' ← M,R_X ← 0, j ← 0
015   FIX DSP 0 PRTx SCI DSP 9         Anzeige j
020   C STO (i) PRTx                   v_0←g_0(0), Anzeige v_0
023   DSZ I                            i' ← M - 1
024   LBL 0 1 I RCL B ÷ -              R_X←1-i'/M=i/M
030   B STO (i) PRTx                   v_i←f(i/M), Anzeige v_i
033   DSZ I GTO 0                      i'←i'-1, wenn i'>0
035   0 D STO (i) PRTx                 v_M←g_1(0), Anzeige v_M
039   LBL 1 CLF 0 RCL A 1 + STO A      Flag 0←0, j←j+1
045   RCL D ÷ FRAC x≠0? GTO 2          wenn [j/n] ≠ j/n
050   STF 0 PRTSPC                     Flag 0 ← 1
052   RCL A FIX DSP 0 PRTx SCI DSP 9   Anzeige j
058   LBL 2 RCL B STO I                i' ← M
061   RCL A RCL C ÷ C                  R_X ← g_0(j/N)
065   P↔S STO (i) P↔S                  w_0 ← g_0(j/N)
068   F? 0 PRTx                        wenn Flag 0=1: Anzeige w_0
070   DSZ I                            i' ← M - 1
071   LBL 3 I 1 - STO I RCL (i)        R_X ← v_{i+1}
077   ISZ I RCL (i) 2 * -              R_X←v_{i+1}-2v_i
082   RCL (i) x↔y ISZ I RCL (i) +      R_X←v_{i+1}-2v_i+v_{i-1},R_Y←v_i
087   RCL E * + DSZ I P↔S STO (i)      w_i←q(v_{i+1}-2v_i+v_{i-1})+v_i
093   P↔S F? 0 PRTx                    wenn Flag 0=1: Anzeige w_i
096   DSZ I GTO 3                      i'←i'-1, wenn i'>0
098   RCL A RCL C ÷ D P↔S STO (i)      w_M ← g_1(j/N)
104   F? 0 PRTx                        wenn Flag 0=1: Anzeige w_M
106   GTO 1
```

Unterprogramm f(x)

107 LBL B

Unterprogramm $g_0(t)$

108 LBL C

Unterprogramm $g_1(t)$

109 LBL D

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	R_{07}	
Adr	i'	j	M	N	n	q	ω	

$R_{\omega-(2M+1)}$	...	$R_{\omega-(M+2)}$	$R_{\omega-(M+1)}$	$R_{\omega-M}$	...	$R_{\omega-1}$	R_{ω}
w_M	...	w_1	w_0	v_M	...	v_1	v_0

Einleseprogramm

000	STO 3 x^2 ÷ R/S	$R_{03} \leftarrow M$, Stop
005	STO 4 = STO 6 R/S	$R_{04} \leftarrow N$, $R_{06} \leftarrow q$, Stop
011	STO 5	$R_{05} \leftarrow n$
013	Op 16 INV Int * 100 = STO 7	$R_{07} \leftarrow \omega$
024	R/S	Stop

Hauptprogramm

025	Lbl A RCL 3 STO 1 A'	$i' \leftarrow M$, $R_{00} \leftarrow$ Adr v_0
032	0 STO 2 Prt	$R_X \leftarrow 0$, $j \leftarrow 0$, Anzeige j
036	C STO Ind 0 * 1 EE = Prt	$v_0 \leftarrow g_0(0)$, Anzeige v_0
044	Op 31	$i' \leftarrow M - 1$
046	Lbl INV A'	$R_{00} \leftarrow$ Adr v_i
049	1 - RCL 1 ÷ RCL 3 =	$R_X \leftarrow 1-i'/M=i/M$
057	B STO Ind 0 Prt	$v_i \leftarrow f(i/M)$, Anzeige v_i
061	Dsz 1 INV	$i' \leftarrow i'-1$, wenn $i'>0$
064	A' 0 D STO Ind 0 Prt	$v_M \leftarrow g_1(0)$, Anzeige v_M

```
070  Lbl lnx Op 22 Stflg 0          j←j+1, Flag 0←1
076  RCL 2 ÷ RCL 5 =                R_X ← j/n
082  INV Int CP INV x=t CE          wenn [j/n] ≠ j/n
088  INV Stflg 0 Adv                Flag 0 ← 0
092  RCL 2 INV EE Prt * 1 EE =      Anzeige j
101  Lbl CE RCL 3 STO 1 B'          i'←M, R_00←Adr w_0
108  RCL 2 ÷ RCL 4 = C STO Ind 0    w_0 ← g_0(j/N)
117  Op 31 Ifflg 0 CLR              i'←M-1, wenn Flag 0=1
122  Prt                            Anzeige w_0
123  Lbl CLR A' RCL Ind 0 +         R_X ← v_i
129  ( CE * 2 +/- +                     +(-2v_i
135  Op 31 A' RCL Ind 0 +                 +v_i+1
141  Op 21 Op 21 A' RCL Ind 0 )           +v_i-1)
149  * Op 31 B' RCL 6 = STO Ind 0    w_i←v_i+(v_i-1+v_i+1-2v_i)q
158  Ifflg 0 x↔t                    wenn Flag 0 = 1
161  Prt                            Anzeige w_i
162  Lbl x↔t Dsz 1 CLR              i'←i'-1, wenn i'>0
167  B' RCL 2 ÷ RCL 4 = D STO Ind 0 w_M ← g_1(j/N)
177  Ifflg 0 x^2                    wenn Flag 0 = 1
180  Prt                            Anzeige w_M
181  Lbl x^2 x↔t A' x↔t STO Ind 0   v_M ← w_M
188  RCL 3 STO 1                    i' ← M
192  Lbl √x B' RCL Ind 0 x↔t        R_T ← w_i
198  A' x↔t STO Ind 0               v_i ← w_i
202  Dsz 1 √x                       i'←i'-1, wenn i'>0
205  GTO lnx
```

Adressenunterprogramme

```
207  Lbl B' ( ( 2 * RCL 3 + 1 )     (2M+1)        Adr w_i
218  GTO 1/x
220  Lbl A' ( RCL 3                 M             Adr v_i
225  Lbl 1/x +/- + RCL 1 +          ·(-1)+i'+
232  RCL 7 ) STO 0                  ω,R_00←Adr
237  INV SBR                        Rücksprung
```

Unterprogramm f(x)

238 Lbl B

Unterprogramm $g_0(t)$

Lbl C

Unterprogramm $g_1(t)$

Lbl D

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der Funktionsunterprogramme für die Anfangsbedingung f und die Randbedingungen g_0, g_1: aus x bzw. t im Anzeigeregister berechnet das Unterprogramm B den Funktionswert f(x), C berechnet $g_0(t)$, D berechnet $g_1(t)$; der berechnete Funktionswert wird bei Rücksprung in das rufende Programm im Anzeigeregister übergeben.
3. Eingabe von M, N und n:
   RTN bzw. RST, M R/S N R/S n R/S.
4. Start der Rechnung und Anzeige von j, v_{0j}, v_{1j},...,v_{Mj} für j = 0, n, 2n,...: A.

Mögliche Wiederholungen:
für andere Schrittweiten h = 1/M, k = 1/N ab Schritt 3,
für andere Anfangsbedingung oder Randbedingungen ab Schritt 2.

Auf Grund der gewählten Anzeigeform in den Programmen wird jeweils die ganze Zahl j ohne Nachkommastelle angezeigt, während die Werte in den Zeitzeilen v_{0j}, v_{1j},...,v_{Mj} in der Exponentialdarstellung wiedergegeben werden.

In dem Programm für die Rechner HP-67/97 werden 2M+8 Datenspeicherplätze belegt, so daß $M \leq 9$ gewählt werden kann.

Für das Programm zu den Rechnern TI-58/59 benötigen wir 2M+10 Datenregister. Bei dem Rechner TI-58 stehen nach 2 Op 17 dann 320 Programmspeicherplätze und 20 Datenregister bereit,

so daß $M \leq 5$ möglich ist; für die Funktionsunterprogramme B, C und D können insgesamt 80 Programmspeicherplätze verwendet werden. Verzichtet man auf das Einleseprogramm und gibt M, N, n, q und ω direkt in den Datenspeicher, so verringert sich der Bedarf an Programmspeicherplätzen um 25. Für das nachfolgende Beispiel reichen dann 240 Programmschritte aus, um neben dem Hauptprogramm und den Adressenunterprogrammen noch die Funktionsunterprogramme aufzunehmen; es kann dann mit der Standardaufteilung zwischen Programm- und Datenspeicher gearbeitet werden, wo 240 Programmschritte und 30 Datenspeicherplätze belegt werden können, so daß $M \leq 10$ zulässig wird. Nach dem Verzicht auf das Einleseprogramm sind weitere Einsparungen im Hauptprogramm zugunsten längerer Funktionsunterprogramme möglich: Lbl A wird dann nicht mehr benötigt, und es können die obigen Programmzeilen 039 bis 042, 094, 095 und 097 bis 100 weggelassen werden, mit denen die Form der Anzeige im Programm kontrolliert wird.

Für den Rechner TI-59 stehen nach 8 Op 17 ebenfalls 320 Programmspeicherplätze zur Verfügung sowie 80 Datenregister, so daß $M \leq 35$ möglich wird.

Beispiel

Wir betrachten die Abkühlung eines Drahtes mit gekühlten Enden mit der Anfangsbedingung $u(x,0) = f(x) = 4x(1-x), 0<x<1$, und den Randbedingungen $u(0,t) = g_0(t) = 0$, $u(1,t) = g_1(t) = 0$, $t \geq 0$.

Die Funktionsunterprogramme lauten
für die Rechner HP-67/97

```
107   LBL B 4 x↔y * 1 LASTx - *      R_X ← 4x(1-x)
      RTN                            Rücksprung
116   LBL C
117   LBL D 0 RTN                    R_X ← 0, Rücksprung,
```

für die Rechner TI-58/59

238	Lbl B (CE * (CE - 1)	$R_X \leftarrow x(x-1)$
248	+/- * 4)	$R_X \leftarrow 4x(1-x)$
252	INV SBR	Rücksprung
253	Lbl C	
255	Lbl D 0 INV SBR	$R_X \leftarrow 0$, Rücksprung.

Mit M=5 und N=50 wurde $h = \frac{1}{5}$, $k = \frac{1}{50}$ und damit $q = \frac{1}{2}$, so daß Stabilität vorliegt. Um jede zehnte Zeitzeile anzuzeigen, wurde n=10 gesetzt. So entstand das folgende Ergebnis.

0.	20.	40.
0. 00	0. 00	0. 00
6.4-01	8.7454224-03	1.2616349-04
9.6-01	1.4150391-02	2.0413681-04
9.6-01	1.4150391-02	2.0413681-04
6.4-01	8.7454224-03	1.2616349-04
0. 00	0. 00	0. 00

10.	30.	50.
0. 00	0. 00	0. 00
7.28125-02	1.0504061-03	1.5153401-05
1.178125-01	1.6995928-03	2.4518718-05
1.178125-01	1.6995928-03	2.4518718-05
7.28125-02	1.0504061-03	1.5153401-05
0. 00	0. 00	0. 00

Bei einer weiteren Rechnung wurde M=5, N=25 gewählt, so daß die Schrittweiten $h = \frac{1}{5}$, $k = \frac{1}{25}$ waren. In diesem Fall ist q=1, und es zeigt sich die Instabilität des zugehörigen Näherungsverfahrens: Für die Näherungslösung ergeben sich oszillierende Funktionswerte, die zum Teil negativ sind und betragsmäßig anwachsen. Für diese Schrittweitenkombination ist durch die größere Schrittweite in der Zeitrichtung die explizite Differenzenapproximation nicht geeignet, das Abnehmen der Temperatur wiederzugeben.

13. Beispiele nichtnumerischer Datenverarbeitung

Zum Abschluß zeigen wir an zwei Beispielen die Leistungsfähigkeit programmierbarer Taschenrechner auch auf Gebieten der nichtnumerischen Datenverarbeitung. Wir beschäftigen uns mit der Damen-Aufgabe und mit dem d'Hondtschen Verfahren. Diese Aufgabenstellungen gehören nicht zum Gebiet der Numerischen Mathematik, jedoch können wir auch hier Lösungsalgorithmen aufstellen und sie auf den Rechnern HP-67/97 und TI-58/59 rechnen.

13.1. Die Damen-Aufgabe

Wir wollen die Frage untersuchen, ob es möglich ist, auf einem Schachbrett acht Damen so aufzustellen, daß keine von ihnen eine andere schlagen kann. Wir beantworten diese Frage, indem wir alle Aufstellmöglichkeiten ermitteln.

Zunächst analysieren wir die Aufgabenstellung nach den Regeln des Schachspiels und entwickeln eine Lösungsstrategie.

Damen vereinen in sich die Eigenschaften von Turm und Läufer. Folglich kann bei dieser Aufgabe in jeder Spalte, Zeile und Diagonale des Schachbretts nur eine Dame stehen. Mögliche Lösungen der gestellten Aufgabe beschreiben wir daher durch einen Vektor $(d_1,\ldots,d_8)$, dessen j-te Komponente d_j angibt, daß in der j-ten Spalte des Schachbretts die Dame in der d_j-ten Zeile steht. Für $1\leq k<j\leq 8$ muß gelten

$$\begin{aligned} d_j &\neq d_k && \text{(Turmeigenschaft)},\\ |d_j - d_k| &\neq j - k && \text{(Läufereigenschaft)}. \end{aligned}$$

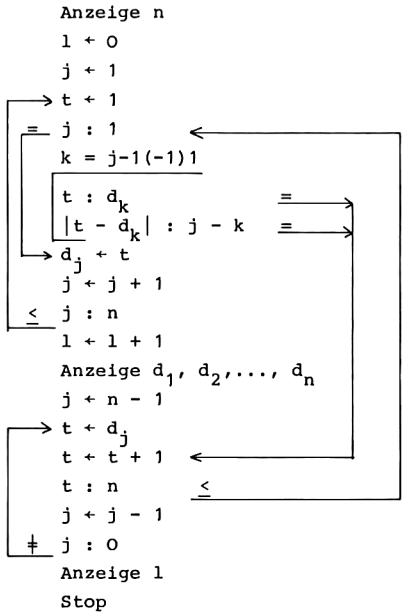
Wir wollen die zulässigen Positionen der Damen spaltenweise bestimmen. In jeder Spalte wird für die Dame zunächst die kleinstmögliche Zeilennummer gewählt. $d_1 = 1$ ist der Ausgangspunkt. Seien nun für $j\geq 2$ bereits $d_1,\ldots,d_{j-1}$ bestimmt. Wir wählen dann für d_j den kleinsten Zeilenindex t, der mit

den obigen Ausschlußbedingungen verträglich ist, falls ein solcher Index existiert; sodann verfahren wir für die nächste Spalte entsprechend (j←j+1). Wurde schließlich ein zulässiger Wert für d_8 ermittelt, so ist eine Aufstellung der acht Damen erreicht und der Ergebnisvektor $(d_1,\ldots,d_8)$ wird angezeigt. Gibt es jedoch keinen solchen Index d_j oder war j = 8, so gehen wir eine Spalte zurück (j←j-1) und stellen die dortige Dame auf den nächsten zulässigen Platz. Wird hierbei schließlich j = 0, so sind alle Aufstellmöglichkeiten in lexikographischer Anordnung ermittelt.

In dem nachfolgend angegebenen Algorithmus wird nach dieser Methode verfahren. Die Aufgabenstellung ist jedoch verallgemeinert. Es werden auf einem Schachbrett aus nxn Feldern alle Möglichkeiten bestimmt, n Damen so aufzustellen, daß sie sich gegenseitig nicht schlagen können.

Algorithmus Voraussetzung: $n \geq 2$ sei eine natürliche Zahl.
Eingabe: n
Anzeige: n;
alle Möglichkeiten $d_1, d_2, \ldots, d_n$;
die Anzahl l der ermittelten Möglichkeiten.

d_1	d_2	...	d_n	j	k	l	n	t



Da bei Taschenrechnern jeweils nur eine Zahl angezeigt bzw. in eine Zeile gedruckt wird, machen wir in den Taschenrechnerprogrammen für $n \leq 9$ die Zahlen $d_1,\ldots,d_n$ in der folgenden Weise gleichzeitig sichtbar: wir berechnen die Zahl

$$z = 10^{n-1}d_1 + 10^{n-2}d_2 + \ldots + 10 \cdot d_{n-1} + d_n$$

und lassen sie anzeigen; ihre Ziffern sind gerade die Zahlen $d_1,\ldots,d_n$. Die Zahl z wird berechnet aus der Gestalt

$$z = (\ldots((10 \cdot d_1 + d_2) \cdot 10 + d_3) \cdot 10 + \ldots + d_{n-1}) \cdot 10 + d_n$$

gemäß dem Horner-Schema:

d_1	d_2	...	d_n	i	n	z

$z \leftarrow d_1$
$i = 2(1)n$
$z \leftarrow 10z + d_i$

Der Lösungsalgorithmus für die Damen-Aufgabe kommt mit einem einzigen Zahlenfeld aus, dessen Komponenten d_i wir wie in Kapitel 1 im Datenregister R_i speichern für i = 1,...,n.

Programm für HP-67/97

Datenspeicher	R_0	R_1	R_2	...	R_n		R_C	R_D	R_E	R_I
	j	d_1	d_2	...	d_n		l	n	t	Adr
	z									i
										k

```
001   STO D PRTx                          R_D ← n, Anzeige n
003   0 STO C 1 STO 0                     l ← 0, j ← 1
007   LBL 0 1 STO E                       t ← 1
010   LBL 1 1 RCL 0 x=y? GTO 3            wenn j = 1
015   STO I DSZ I                         k ← j - 1
017   LBL 2 RCL E RCL (i) x=y? GTO 6      wenn t = d_k
022   - ABS RCL 0 I - x=y? GTO 6          wenn |t-d_k| = j-k
029   DSZ I GTO 2                         k←k-1, wenn k>0
031   LBL 3 RCL 0 STO I                   R_I ← Adr d_j
034   RCL E STO (i) 1 STO + 0             d_j ← t, j ← j + 1
038   RCL D RCL 0 x≤y? GTO 0              wenn j≤n
042   RCL C 1 + STO C                     l ← l + 1
046   1 STO I RCL (i) STO 0               i ← 1, z ← d_1
050   LBL 4 ISZ I                         i ← i + 1
052   10 STO * 0 RCL (i) STO + 0          z ← 10z + d_i
057   I RCL D x>y? GTO 4                  wenn n>i
061   RCL 0 PRTx                          Anzeige d_1,...,d_n
063   RCL D 1 - STO 0                     j ← n - 1
067   LBL 5 RCL 0 STO I RCL (i) STO E     t ← d_j
072   LBL 6 RCL D RCL E 1 + STO E         R_Y ← n, t ← t + 1
078   x≤y? GTO 1                          wenn t≤n
080   0 STO I DSZ (i) GTO 5               j←j-1, wenn j>0
084   RCL C PRTx                          Anzeige l
086   R/S                                 Stop
```

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	...	R_n		R_{26}	R_{27}	R_{28}	R_{29}
i	d_1	d_2	...	d_n		l	n	t	j
k									z

```
000   STO 27 Prt                          R27 ← n, Anzeige n
003   0 STO 26 1 STO 29                   l ← 0, j ← 1
009   Lbl INV 1 STO 28                    t ← 1
014   Lbl lnx 1 x↔t RCL 29 x=t CLR        wenn j = 1
022   STO 0 Op 30                         k ← j - 1
026   Lbl CE RCL 28 - x↔t                 RT ← t
032   RCL Ind 0 x=t √x                    wenn dk = t
036   = |x| x↔t                           RT ← |t - dk|
039   RCL 29 - RCL 0 = x=t √x              wenn j-k = |t-dk|
047   Dsz 0 CE                            k←k-1, wenn k>0
050   Lbl CLR RCL 28 STO Ind 29           dj ← t
056   1 SUM 29                            j ← j + 1
059   RCL 29 x↔t RCL 27 x≥t INV           wenn n≥j
066   1 SUM 26                            l ← l + 1
069   1 STO 0 RCL Ind 0 STO 29            i ← 1, z ← d1
076   Lbl x↔t Op 20                       i ← i + 1
080   10 Prd 29 RCL Ind 0 SUM 29          z ← 10z + di
088   RCL 27 x↔t RCL 0 INV x≥t x↔t        wenn i<n
096   RCL 29 Prt                          Anzeige d1,...,dn
099   RCL 27 - 1 = STO 29                 j ← n - 1
106   Lbl x² RCL Ind 29 STO 28            t ← dj
112   Lbl √x CLR 1 SUM 28                 t ← t + 1
118   RCL 28 x↔t RCL 27 x≥t lnx           wenn n≥t
125   1 INV SUM 29                        j ← j - 1
129   RCL 29 CP x↔t INV x≥t x²            wenn 0<j
136   RCL 26 Prt                          Anzeige l
139   R/S                                 Stop
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Wahl des Anzeigeformats bei den Rechnern HP-67/97:
   FIX DSP O (entfällt für die Rechner TI-58/59).
3. Eingabe von n, $2 \leq n \leq 9$, und Start der Rechnung:
   n RTN R/S bzw. n RST R/S.
   Angezeigt werden n, alle zulässigen Aufstellmöglichkeiten von n Damen auf dem nxn-Schachbrett und zum Schluß die Anzahl der ermittelten Möglichkeiten.

Mögliche Wiederholungen: Schritt 3.

Da wir $d_1,\ldots,d_n$ in den Datenregistern $R_1,\ldots,R_n$ gespeichert haben, ist die indirekte Adressierung dieser Größen einfach.

Bei den Rechnern HP-67/97 dient das Indexregister R_I als Adreßregister zur indirekten Adressierung. Neben den Schleifenparametern i und k wird daher dort auch Adr d_k = k und Adr d_j = j verwendet. Darüberhinaus wird in den Anweisungen O8O bis O83 mit Null in R_I ermöglicht, daß die Wertzuweisung j←j-1 und die anschließende Verzweigung auf Grund des Vergleichs von j mit Null durch eine indirekte DSZ-Anweisung realisiert werden können.

Bei den Rechnern TI-58/59 ist es möglich, jedes Datenregister als Adreßregister zur indirekten Adressierung zu verwenden. Bei der gewählten Speicherbelegung kann daher d_k über R_O und d_j über R_{29} indirekt adressiert werden. Andererseits ist mit j in R_{29} eine Dsz-Anweisung für j nicht möglich.

Bei der Untersuchung, ob ein gewählter Wert t als d_j zulässig ist, wird in dem Programm für die TI-Rechner während der Berechnung von $t-d_k$ die Turmeigenschaft ($t \neq d_k$) überprüft; bei Bedarf wird bereits vor Abschluß der Differenz $t-d_k$ verzweigt.Die dann noch offenstehenden arithmetischen Operationen werden daher in Programmspeicherzeile 114 durch CLR gelöscht.

Ergebnisse

```
     2. ***
     0. ***

     3. ***
     0. ***

     4. ***
  2413. ***
  3142. ***
     2. ***

     5. ***
 13524. ***
 14253. ***
 24135. ***
 25314. ***
 31425. ***
 35241. ***
 41352. ***
 42531. ***
 52413. ***
 53142. ***
    10. ***

     6. ***
246135. ***
362514. ***
415263. ***
531642. ***
     4. ***
```

```
     7. ***
1357246. ***
1473625. ***
1526374. ***
1642753. ***
2417536. ***
2461357. ***
2514736. ***
2531746. ***
2574136. ***
2637415. ***
2753164. ***
3162574. ***
3164275. ***
3572461. ***
3625147. ***
3724615. ***
3741526. ***
4136275. ***
4152637. ***
4275316. ***
4613572. ***
4736251. ***
4752613. ***
5147362. ***
5164273. ***
5263741. ***
5316427. ***
5724613. ***
5726314. ***
6135724. ***
6251473. ***
6314752. ***
6357142. ***
6374152. ***
6427531. ***
6471352. ***
7246135. ***
7362514. ***
7415263. ***
7531642. ***
    40. ***
```

```
     8. ***
15863724. ***
16837425. ***
17468253. ***
17582463. ***
24683175. ***
25713864. ***
25741863. ***
26174835. ***
26831475. ***
27368514. ***
27581463. ***
28613574. ***
31758246. ***
35281746. ***
35286471. ***
35714286. ***
35841726. ***
36258174. ***
36271485. ***
36275164. ***
36418572. ***
36428571. ***
36814752. ***
36815724. ***
36824175. ***
37285146. ***
37286415. ***
38471625. ***
41582736. ***
41586372. ***
42586137. ***
42736815. ***
42736851. ***
42751863. ***
42857136. ***
42861357. ***
46152837. ***
46827135. ***
46831752. ***
47185263. ***
47382516. ***
47526138. ***
47531682. ***
48136275. ***
48157263. ***
48531726. ***
```

```
51468273. ***
51842736. ***
51863724. ***
52468317. ***
52473861. ***
52617483. ***
52814736. ***
53168247. ***
53172864. ***
53847162. ***
57138642. ***
57142863. ***
57248136. ***
57263148. ***
57263184. ***
57413682. ***
58413627. ***
58417263. ***
61528374. ***
62713584. ***
62714653. ***
63175824. ***
63184275. ***
63185247. ***
63571428. ***
63581427. ***
63724815. ***
63728514. ***
63741825. ***
64158273. ***
64285713. ***
64713528. ***
64718253. ***
68241753. ***
71386425. ***
72418536. ***
72631485. ***
73168524. ***
73625164. ***
74258136. ***
74286135. ***
75316624. ***
82417536. ***
82531746. ***
83162574. ***
84136275. ***
    92. ***
```

Für n = 2 und n = 3 gibt es keine Aufstellmöglichkeiten; in diesen Fällen wird daher nur n und 1 = 0 angezeigt.

Für n = 8 beträgt die Rechenzeit bis zur Ermittlung der ersten Aufstellmöglichkeit 39 Minuten für HP-97, 52 Minuten für TI-59. Die zweite Möglichkeit wurde nach weiteren 11 bzw. 16 Minuten angezeigt. Die Rechenzeit zur Ermittlung aller 92 Lösungen der Damen-Aufgabe beträgt 11 Stunden 42 Minuten bzw. 15 Stunden 48 Minuten.

Wie bereits erwähnt, muß wegen der gleichzeitigen Anzeige der Komponenten $d_1,\ldots,d_n$ einer zulässigen Position bei den obigen Programmen die Bedingung $n\leq 9$ erfüllt sein. Durch Umstellung auf Einzelanzeige der Lösungskomponenten $d_1,\ldots,d_n$ ließen sich die Programme auch noch für größere Werte von n verwenden. Auf den Rechnern HP-67/97 wäre dann $n\leq 21$ möglich, für die Rechner TI-58/59 könnte infolge der obigen Belegung des Datenspeichers $n\leq 25$ gewählt werden. Jedoch wächst die Rechenzeit für größere n stark an. So wurden zur Ermittlung der 352 Lösungen der Aufgabe auf einem 9x9-Schachbrett mit dem Rechner HP-97 rund 58 Stunden benötigt, auf dem Rechner TI-59 betrug die Rechenzeit 77,5 Stunden.

In diesem Zusammenhang zeigt es sich, daß es zweckmäßig gewesen ist, die ursprüngliche Aufgabenstellung zu verallgemeinern. Statt für 8 Damen auf einem 8x8-Schachbrett wurde der obige Algorithmus für n Damen auf einem nxn-Schachbrett formuliert, mit einer natürlichen Zahl $n\geq 2$. Dadurch war es möglich, das entstehende Programm für n = 2, n = 3, n = 4 zu testen, ohne daß für den Testlauf die umfangreiche Rechenzeit des ursprünglichen Falles n = 8 benötigt wurde.

13.2. Das d'Hondtsche Verfahren

Wird ein Parlament nach dem Verhältniswahlrecht gewählt, so setzt man für die Mandatszuteilung häufig das d'Hondtsche Höchstzahlverfahren ein. Wir erläutern diese Methode zunächst an einem Beispiel. Dann beschreiben wir das Verfahren ausführlich, so daß wir anschließend einen zugehörigen Algorithmus aufstellen können, den wir in Taschenrechnerprogramme übertragen.

Für das d'Hondtsche Verfahren werden die von den einzelnen Listen erreichten Stimmenzahlen nacheinander durch 1,2,3,... dividiert; die entstehenden Quotienten werden der Größe nach geordnet, und die Mandate werden nach der Reihenfolge dieser "Höchstzahlen" den Listen zugeteilt.

Wurden zum Beispiel bei einer Wahl für ein Gremium mit 10 Mitgliedern von 1000 Stimmen 338 Stimmen für Liste A abgegeben, 416 Stimmen für Liste B und 246 Stimmen für Liste C, so entsteht die folgende Tabelle der Höchstzahlen.

	A	Mandat	B	Mandat	C	Mandat
÷ 1	338	Nr. 2	416	Nr. 1	246	Nr. 3
÷ 2	169	5	208	4	123	7
÷ 3	$112,\overline{6}$	8	$138,\overline{6}$	6	82	
÷ 4	84,5	10	104	9	61,5	
÷ 5	67,6		83,2		49,2	
÷ 6	$56,\overline{3}$		$69,\overline{3}$		41	

Die Listen A und B erhalten je 4 Mandate, Liste C erhält 2 Mandate.

Entsprechend könnte man für die Vergabe von m Mandaten an n Listen mit mn Quotienten arbeiten, davon die m größten ermitteln und die Mandate den zugehörigen Listen zuteilen. Ein solches Vorgehen ist jedoch sehr aufwendig hinsichtlich des Bedarfs an Datenspeicherplätzen; auch läßt das Beispiel erkennen, daß es nur selten erforderlich ist, alle Quotienten bis zur Division durch m zu berechnen.

Wir betrachten nun die allgemeine Situation und beschreiben das d'Hondtsche Verfahren ausführlich. Hierbei findet die Zuteilung der Mandate während der Berechnung der Quotienten statt, so daß für jede Liste immer nur ein Quotient verfügbar sein muß. Auch wird das Zuteilungsverfahren für den Fall gleicher Höchstzahlen geregelt.

Für ein Parlament mit m Mitgliedern hat eine Wahl nach dem Verhältniswahlrecht stattgefunden. n Listen $L_1,\ldots,L_n$ haben mit ihrem Wahlergebnis die Bedingungen erfüllt, die für eine Mandatszuteilung erforderlich sind. Dabei wurden s_i Stimmen für die Liste L_i abgegeben, i=1,...,n. Gesucht sind die Anzahlen m_i der Mandate, welche den Listen $L_1,\ldots,L_n$ auf Grund des Wahlergebnisses zustehen.

Bei der Mandatsverteilung nach dem d'Hondtschen Verfahren geht man folgendermaßen vor. Das erste Mandat erhält die Liste mit der höchsten Stimmenzahl; bei Stimmengleichheit mehrerer Listen geht das erste Mandat an diejenige Liste, welche die niedrigere Nummer hat. Hat nun die Liste L_i bereits $m_i \geq 0$ Mandate zugewiesen erhalten, i=1,...,n, und ist $m_1 + m_2 + \ldots + m_n < m$, so ist noch mindestens ein weiteres Mandat zu vergeben; hierzu ermittelt man die größte der Teilstimmenzahlen $s_i/(m_i+1)$, i=1,...,n, und teilt der zugehörigen Liste das nächste Mandat zu, bei Auftreten mehrerer maximaler Teilstimmenzahlen der Liste mit der niedrigeren Nummer. Man verfährt in dieser Weise, bis alle Mandate vergeben sind.

Für den Fall, daß bei der Zuteilung des letzten Mandats das Maximum der Teilstimmenzahlen mehrmals angenommen wird, erhält also nur die Liste mit der niedrigeren Nummer ein Mandat; die anderen Listen mit maximaler Teilstimmenzahl im letzten Zuteilungsschritt erhalten keine weiteren Mandate (Form I). In diesem Zusammenhang werden jedoch häufig noch zwei andere Verfahrensweisen praktiziert.

Form II: Die Zuteilung des m-ten Mandats wird aufgehoben; es findet ein Losentscheid um das letzte Mandat statt unter allen Listen mit maximaler Teilstimmenzahl im letzten Zuteilungsschritt.

Form III: Es erhalten alle diejenigen Listen ein zusätzliches Mandat, für die bei Vergabe des m-ten Mandats die Teilstimmenzahl maximal ist; hierdurch sind bis zu n-1 zusätzliche Mandate möglich.

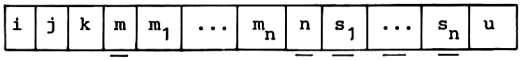
Anschließend geben wir einen Algorithmus an für das d'Hondtsche Höchstzahlverfahren gemäß Form I. Hierbei wird die Reihenfolge des Mandatserhalts mit den Teilstimmenzahlen angezeigt und zum Schluß die Sitzverteilung. Aussagen über die Mandatsverteilung mit Abschluß nach Form II oder III kann man damit erhalten, wenn man die Verteilung von m+n-1 Mandaten ermittelt und auf Grund der Teilstimmenzahlen des m-ten bis (m+n-1)-ten Zuteilungsschrittes über den Abschluß des Verteilungsverfahrens entscheidet. In dem Algorithmus werden im Hinblick auf die Realisierung mit Taschenrechnern zur Einsparung von Datenspeicherplätzen für jede Liste das Wahlergebnis und die daraus entstehenden Teilstimmenzahlen auf jeweils demselben Speicherplatz geführt. Das ist deshalb möglich, weil aus jeder Teilstimmenzahl durch Multiplikation mit ihrem Nenner das zugehörige Wahlergebnis wiederhergestellt werden kann.

Algorithmus

Eingabe: m, n, $s_1, s_2, \ldots, s_n$.

Anzeige: (a) für $j = 1,\ldots,m$: Nummer j des Zuteilungsschritts; Nummer k der Liste, die in diesem Schritt ein Mandat erhält; zugehörige Teilstimmenzahl s_k.

(b) Endergebnis: Nummer i der Liste und Anzahl m_i der erhaltenen Mandate, für $i = 1,\ldots,n$.




```
i = 1(1)n                          Erläuterungen
  m_i ← 0
j = 1(1)m
  Anzeige j
  u ← 0
  i = 1(1)n
    u : s_i   >
    u ← s_i                        u = max s_i = s_k
    k ← i                               i
  Anzeige k                        mit k = min {i|s_i = u}
  Anzeige u
  m_k ← m_k + 1
  s_k ← s_k m_k/(m_k+1)
  i = 1(1)n
    Anzeige i                      Anzeige (b)
    Anzeige m_i
  Stop
```

Ist man nur an der Anzeige des Endergebnisses interessiert, so entfernt man alle Anzeigeanweisungen im Wiederholungsbereich der Schleife j = 1(1)m; dadurch entfällt Anzeige (a).

Mit der Anweisung $s_k \leftarrow s_k m_k/(m_k+1)$ des Algorithmus wird nach der Zuteilung eines Mandats an die Liste L_k aus der bisherigen Teilstimmenzahl s_k durch das Produkt $s_k m_k$ das Wahlergebnis der Liste zurückerhalten; anschließend ergibt die Division durch m_k+1 die neue Teilstimmenzahl für L_k. Bei diesen wiederholten Divisionen und Multiplikationen können Rundungsfehler zu Ungenauigkeiten führen, die in den folgenden Zuteilungsschritten bei der Ermittlung der größten Teilstimmenzahl u eine unzutreffende Reihenfolge nach sich ziehen können; insbesondere bei der Zuteilung des m-ten Mandats kann dies im Fall gleicher Teilstimmenzahlen zu einer Abweichung von der Zuteilungsvorschrift führen. Wir vermeiden die Fort-

pflanzung von Rundungsfehlern, indem wir in den nachfolgenden Programmen das Produkt $s_k m_k$ durch Rundung ganzzahlig machen, so daß immer wieder das exakte Wahlergebnis hergestellt wird.

Bei den Programmen für die Rechner TI-58/59 steuern wir die Schleifen i = 1(1)n und j = 1(1)m mit Hilfe der komplementären Indizes i' = n+1-i, j' = m+1-j und der Dsz-Anweisung; vom Ende des Datenspeichers her speichern wir die erhaltenen Stimmen und Teilstimmenzahlen $s_1,\ldots,s_n$, anschließend die Mandatszahlen $m_1,\ldots,m_n$. Mit der Platzziffer ω des letzten Datenspeicherplatzes lauten daher die zugehörigen Adressen

$$\text{Adr } s_i = \omega - (i-1) = -n + i' + \omega,$$
$$\text{Adr } m_i = \omega - n - (i-1) = -2n + i' + \omega .$$

Für die Rechner HP-67/97 weichen wir davon ab und berücksichtigen die spezielle Gestalt des Datenspeichers mit der Aufteilung in Primär- und Sekundärspeicher.

Programm für HP-67/97

Datenspeicher

R_A	R_B	R_C	R_D	R_E	R_I
j	k-1	m	n	u	i-1 =Adr

R_0	R_1	...	R_{n-1}		R_{s0}	R_{s1}	...	$R_{s,n-1}$
s_1	s_2	...	s_n		m_1	m_2	...	m_n

$n \leq 10$

Einleseprogramm

```
001   CLREG P↔S                    m_i←0 für i=1,...,n
003   LBL e RCL C R/S STO C        Anzeige m, Stop, R_C←m
007   RCL D R/S STO D              Anzeige n, Stop, R_D←n
010   0 STO I                      i ← 1
012   LBL 0 RCL (i) R/S            Anzeige s_i, Stop  ←┐
015   STO (i) ISZ I GTO 0          Speichern von s_i, i←i+1, ┘
```

Hauptprogramm

018	LBL A 1 STO A DSP O	j ← 1
022	LBL 1 RCL A F? O PRTx	wenn Flag O=1: Anzeige j
026	O STO E STO I	u ← O, i ← 1
029	LBL 2 RCL E RCL (i) x≤y? GTO 3	wenn $s_i \leq u$
034	STO E I STO B	u ← s_i, k ← i
037	LBL 3 ISZ I I RCL D x>y? GTO 2	i←i+1, wenn n>i-1
043	RCL B STO I	i ← k
045	1 + F? O PRTx	wenn Flag O=1: Anzeige k
049	RCL E DSP 9 F? O PRTx	wenn Flag O=1: Anzeige u
053	P↔S ISZ (i)	$m_k \leftarrow m_k + 1$
055	RCL (i) ENTER P↔S RCL (i) *	$R_X \leftarrow s_k m_k$, $R_Y \leftarrow m_k$
060	DSP O RND x↔y 1 + ÷ STO (i)	$s_k \leftarrow RND(s_k m_k)/(m_k+1)$
067	RCL C RCL A 1 + STO A	j←j+1, $R_X \leftarrow j$, $R_Y \leftarrow m$
072	x≤y? GTO 1	wenn j≤m

Anzeige des Endergebnisses

074	LBL B PRTSPC O STO I P↔S	i ← 1
079	LBL 4 RCL (i) ISZ I I	Anzeige i,
083	DSP O PRTx R↓ DSP 1 PRTx	Anzeige m_i und i←i+1
088	R↑ RCL D x>y? GTO 4	wenn n>i-1
092	P↔S R/S	Stop

Programm für TI-58/59

Datenspeicher

R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	R_{07}	
i'	j'	k'	m	n	u	Adr	ω	

$R_{\omega-2n+1}$	...	$R_{\omega-n}$	$R_{\omega-n+1}$	...	R_{ω}
m_n	...	m_1	s_n	...	s_1

Einleseprogramm

```
000   Op 16 INV Int * 100 =          R_X ← ω
009   STO 7 STO 6                    R_07←ω, R_06←Adr s_1
013   RCL 3 R/S STO 3                Anzeige m, Stop,R_03←m
018   RCL 4 R/S STO 4                Anzeige n, Stop,R_04←n
023   Lbl INV RCL Ind 6 R/S          Anzeigen, Stop
028   STO Ind 6 Dsz 6 INV            Speichern, Adr←Adr-1,
```

Hauptprogramm

```
033   Lbl A RCL 4 STO 0              i' ← n
039   Lbl lnx B' 0 STO Ind 6         m_i ← 0
045   Dsz 0 lnx                      i'←i'-1, wenn i'>0
048   RCL 3 STO 1                    j' ← m
052   Lbl CE INV Ifflg 0 CLR         wenn Flag 0 = 0
058   RCL 3 + 1 - RCL 1 = Prt        Anzeige j = m+1-j'
067   Lbl CLR 0 STO 5 RCL 4 STO 0    u ← 0, i' ← n
076   Lbl x↔t A' RCL Ind 6 x↔t       R_T ← s_i
082   RCL 5 x≥t x^2                  wenn u≥s_i
086   x↔t STO 5 RCL 0 STO 2          u ← s_i, k ← i
093   Lbl x^2 Dsz 0 x↔t              i'←i'-1, wenn i'>0
098   RCL 2 STO 0 INV Ifflg 0 √x     i←k, wenn Flag 0 = 0
106   +/- + RCL 4 + 1 = Prt          Anzeige k = n+1-k'
114   RCL 5 Fix 8 Prt INV Fix        Anzeige u
121   Lbl √x B' 1 SUM Ind 6          m_k ← m_k + 1
127   RCL Ind 6 * x↔t                R_X←m_k, R_T←m_k
131   A' RCL Ind 6 = EE INV EE       R_X←RND(s_k m_k)
138   ÷ ( x↔t + 1 ) = STO Ind 6      s_k←RND(s_k m_k)/(m_k+1)
147   Dsz 1 CE                       j'←j'-1, wenn j'>0
```

Anzeige des Endergebnisses

```
150   Lbl B Adv RCL 4 STO 0          i' ← n
157   Lbl 1/x RCL 4 + 1 - RCL 0 =    R_X←n+1-i' = i  ←┐
167   Prt                            Anzeige i        │
168   B' RCL Ind 6 Fix 1 Prt         Anzeige m_i      │
174   INV Fix Dsz 0 1/x              i'←i'-1, wenn i'>0 ┘
179   R/S                            Stop
```

Adressenunterprogramme

```
180   Lbl A' ( 1 GTO STO             1 ──┐   Adr s_i
186   Lbl B' ( 2                     2   │   Adr m_i
190   Lbl STO +/- * RCL 4 +          ·(-n)+ ←┘
197   RCL 0 + RCL 7 ) STO 6          i'+ω, R_06←Adr
205   INV SBR                        Rücksprung
```

Anleitung zur Verwendung der Programme

1. Eingabe des Programms.
2. Eingabe der Mandatsanzahl m, der Anzahl n der Listen und der erhaltenen Stimmen:
   RTN bzw. RST, R/S m R/S n R/S
   s_1 R/S ... s_n R/S.
3. Möglicher Zwischenschritt zur Überprüfung der in Schritt 2 eingegebenen Werte:
   bei HP-67/97 durch
   e Anzeige m R/S Anzeige n R/S
   Anzeige s_1 R/S ... Anzeige s_n R/S,
   bei TI-58/59 mit der Anweisungsfolge von Schritt 2.
   Fehlerhaft eingegebene Werte werden hierbei durch Überschreiben im Anzeigeregister auch im Datenspeicher berichtigt.
4. Falls bei Schritt 5 während der Rechnung die Zuteilungsschritte angezeigt werden sollen: STF 0 bzw. Stflg 0; falls sie nicht angezeigt werden sollen: CLF 0 bzw. INV Stflg 0.
5. Berechnung der Mandatsverteilung und Anzeige des Endergebnisses: A.

Weitere zusätzlich mögliche Schritte:

6. Soll nach Schritt 5 die Mandatsverteilung für M Mandate ermittelt werden mit M>m, so ist dies durch Fortsetzung der Rechnung möglich
   bei den Rechnern HP-67/97 durch
   M STO C GTO 1 R/S,
   bei den Rechnern TI-58/59 durch
   M STO 03 M-m STO 01 GTO CE R/S.
7. Wiederholung der Anzeige des Endergebnisses: B.

Mögliche Wiederholungen:

Schritt 7,
Schritt 6,
ab Schritt 2 für andere Eingabedaten.

Mit Schritt 4 und 6 können die Zuteilungsschritte angezeigt werden in der folgenden Form: Nummer des Zuteilungsschritts; Nummer der Liste, die ein Mandat erhält; zugehörige Teilstimmenzahl (mit größtmöglicher Zahl von Nachkommastellen).

Bei Schritt 5 und 6 wird nach Beendigung der Rechnung die Mandatsverteilung angezeigt: die Listennummer i und die Anzahl m_i der erhaltenen Mandate für i = 1,...,n. Zur besseren Unterscheidung werden hierbei die Mandatszahlen mit einer Nachkommastelle angegeben, alle anderen ganzzahligen Werte ohne Nachkommastellen.

Das Programm für die Rechner HP-67/97 benötigt 2n+6 Datenspeicherplätze, so daß man $n \leq 10$ wählen kann. In dem Programm für die Rechner TI-58/59 werden 2n+8 Datenregister belegt. Für den Rechner TI-58 stehen nach dem Einschalten des Gerätes Speicherplätze für 240 Programmschritte und 30 Datenregister bereit, so daß $n \leq 11$ möglich ist. Auf dem Rechner TI-59 führt 9 Op 17 zur Aufteilung des Speichers in 240 Programmschritte und 90 Datenspeicherplätze, womit $n \leq 41$ zulässig wird.

Beispiele

(1) $m = 10$, $n = 3$, $s_1 = 338$, $s_2 = 416$, $s_3 = 246$.

Vor Beginn der Rechnung wurde Flag 0 gesetzt, so daß die Zuteilungsschritte mit ausgedruckt sind. Es ergibt sich die Mandatsverteilung des obigen Beispiels.

```
          1.              6.
          2.              2.
416.0000000     138.6666667
          2.              7.
          1.              3.
338.0000000     123.0000000
          3.              8.
          3.              1.
246.0000000     112.6666667
          4.              9.          1.
          2.              2.        4.0
208.0000000     104.0000000           2.
          5.             10.        4.0
          1.              1.          3.
169.0000000      84.50000000         2.0
```

(2) $m = 5$, $n = 5$,
$s_1 = 618$, $s_2 = 412$, $s_3 = 1236$, $s_4 = 404$, $s_5 = 719$.

Mit Flag 0 gesetzt folgte die Anzeige der Zuteilungsschritte und das Endergebnis. Mit M = 9 wurde die Rechnung fortgesetzt.

```
          1.
          3.
1236.000000
          2.                              6.
          5.                              3.
719.0000000     1.      412.0000000       1.
          3.   1.0                        7.   2.0
          1.     2.                       4.     2.
618.0000000     1.0     404.0000000       1.0
          4.     3.                       8.     3.
          3.   2.0                        5.   3.0
618.0000000     4.      359.5000000       4.
          5.   0.0                        9.   1.0
          2.     5.                       1.     5.
412.0000000     1.0     309.0000000       2.0
```

Man erkennt, daß im 5. Zuteilungsschritt die Listen L_2 und L_3 gleiche Teilstimmenzahlen besitzen. Bei Vergabe von 5 Mandaten und Abschluß des Zuteilungsverfahrens nach Form II werden daher nur 4 Mandate fest vergeben: L_1 und L_5 erhalten je 1 Mandat, L_3 erhält 2 Mandate, L_4 erhält kein Mandat; die Listen L_2 und L_3 nehmen an einem Losentscheid um das 5. Mandat teil. Für m = 5 und Abschluß des Zuteilungsverfahrens nach Form III erhalten sowohl L_2 wie L_3 im letzten Zuteilungsschritt ein Mandat, so daß das Endergebnis nach Form III lautet: L_1, L_2 und L_5 erhalten je 1 Mandat, L_3 erhält 3 Mandate, L_4 erhält kein Mandat.

(3) m = 12, n = 4,
$s_1 = 400$, $s_2 = 1000$, $s_3 = 800$, $s_4 = 600$.

Um die Zuteilungsschritte mit auszudrucken, wurde wieder Flag 0 gesetzt. Die Rechnung wurde anschließend mit M = 15 fortgesetzt.

```
          1.              7.
          2.              2.
1000.0000000    333.3333333
          2.              8.
          3.              4.
 800.0000000    300.0000000
          3.              9.
          4.              3.
 600.0000000    266.6666667
          4.             10.
          2.              2.              1.
 500.0000000    250.0000000             2.0
          5.             11.              2.
          1.              1.             5.0
 400.0000000    200.0000000              3.
          6.             12.             3.0
          3.              2.              4.
 400.0000000    200.0000000             2.0
```

```
          13.
           3.                    1.
200.0000000                    2.0
          14.                   2.
           4.                   6.0
200.0000000                    3.
          15.                  4.0
           2.                   4.
166.6666667                    3.0
```

Auf Grund des speziellen Wahlergebnisses in diesem Beispiel sind im 11. Zuteilungsschritt alle Quotienten gleich. Da m = 12 Mandate vergeben werden sollen, erhält die Liste L_1 das 11. Mandat; wird die Mandatsvergabe nach der oben besprochenen Form I abgeschlossen, so erhält die Liste L_2 das 12. Mandat, und die Listen L_3 und L_4 erhalten für dieselbe Höchstzahl kein Mandat. Bei Abschluß nach Form III durch die Zuteilung zusätzlicher Mandate erhalten auch die Listen L_3 und L_4 je ein Mandat.

In diesem Zusammenhang zeigt sich die volle Bedeutung der oben formulierten Mandatsvergabe mit Abschluß nach Form II, wo bei gleichen Teilstimmenzahlen im letzten Zuteilungsschritt das letzte Mandat verlost wird. Da m = 12 Mandate vergeben werden sollen, erhält die Liste L_1 das 11. Mandat fest zugeteilt, da sie es nicht im letzten Zuteilungsschritt erwirbt. Die Listen L_2, L_3 und L_4 haben im letzten Zuteilungsschritt gleiche Teilstimmenzahlen, so daß zwischen ihnen das 12. Mandat verlost wird.

LITERATUR

NUMERISCHE MATHEMATIK

[1] Becker, J.; Dreyer, H.-J.; Haacke, W.; Nabert, R.: Numerische Mathematik für Ingenieure. Stuttgart: Teubner 1977.

[2] Engel, A.: Elementarmathematik vom algorithmischen Standpunkt. Stuttgart: Klett 1977.

[3] Finckenstein, K. Graf Finck v.: Einführung in die Numerische Mathematik Band 1, Band 2. München: Hanser 1977, 1978.

[4] Grigorieff, R.D.: Numerik gewöhnlicher Differentialgleichungen, Band 1: Einschrittverfahren, Band 2: Mehrschrittverfahren. Stuttgart: Teubner 1972, 1977.

[5] Henrici, P.: Elemente der numerischen Analysis Band 1, Band 2. Mannheim: Bibliographisches Institut 1972.

[6] Jordan-Engeln, G.; Reutter, F.: Numerische Mathematik für Ingenieure. Mannheim: Bibliographisches Institut 1978.

[7] Selder, H.: Einführung in die Numerische Mathematik für Ingenieure. München: Hanser 1973.

[8] Stoer, J.: Einführung in die Numerische Mathematik I, II. Berlin: Springer 1972, 1973 (Band II gemeinsam mit R. Bulirsch).

[9] Stummel F.; Hainer, K.: Praktische Mathematik. Stuttgart: Teubner 1971.

[10] Törnig, W.: Numerische Mathematik für Ingenieure und Physiker Band 1, Band 2. Berlin: Springer 1979.

[11] Werner, H.: Praktische Mathematik I, II. Berlin: Springer 1975, 1979 (Band II gemeinsam mit R. Schaback).

PROGRAMMIERBARE TASCHENRECHNER

[12] Alt, H.: Angewandte Mathematik, Finanzmathematik, Statistik, Informatik für UPN-Rechner. Braunschweig: Vieweg 1979.

[13] Bromm, K.U.: Programmierbare Taschenrechner in Schule und Ausbildung. Braunschweig: Vieweg 1979.

[14] Eisberg, R.M.: Mathematische Physik für Benutzer programmierbarer Taschenrechner. Oldenbourg 1978.

[15] Gloistehn, H.H.: Lehr- und Übungsbuch für den TI-58 und TI-59. Braunschweig: Vieweg 1978.

[16] Henrici, P.: Analytische Rechenverfahren für den Taschenrechner HP-25. München: Oldenbourg 1978.

[17] Hoyer, K.; Schnell, G.: Differentialgleichungen der Elektrotechnik; Lösung mittels Theorie der Differentialgleichungen, Laplace-Transformation und programmierbarer Taschenrechner. Braunschweig: Vieweg 1978.

[18] Kahan, W.; Parlett, B.N.: Können Sie sich auf Ihren Rechner verlassen? In: Jahrbuch Überblicke Mathematik 1978, S. 199-216. Mannheim: Bibliographisches Institut 1978.

[19] Ludwig, H.-J.: Programmoptimierung für Taschenrechner (AOS). Braunschweig: Vieweg 1979.

[20] Schauer, H.; Barta, G.: Methoden der Programmerstellung für Tisch- und Taschenrechner. Wien: Springer 1979.

[21] Thießen, P.: Lehr- und Übungsbuch für die Rechner HP-29C/HP-19C und HP-67/HP-97. Braunschweig: Vieweg 1980.

[22] Venz, G.: Lösung von Differentialgleichungen mit programmierbaren Taschenrechnern; Verfahren für gewöhnliche und partielle Differentialgleichungen. München: Oldenbourg 1978.

SACHVERZEICHNIS

A

Abkühlung 224
Ableitung von Polynomen 23
Abschneiden von
  Nachkommastellen 13
Abspaltung von
  Linearfaktoren 23
Adams-Bashforth-Extrapola-
  tionsverfahren 115f, 128
Adams-Moulton-Interpola-
  tionsverfahren 115f, 128
Adams-Prädiktor-Korrektor-
  verfahren 115f, 128
Adressenunterprogramme 134
Adressierung
-, indirekte 6, 131f
-, absolute 11
äquidistant 63, 74
Algorithmus 1, 9
-, in Endlosform 10
Anfangs-bedingung 101, 114,
  216
- marke 70
- Randwert-Aufgabe 216
- wert 3, 101, 114
-- aufgaben 101f
Anlaufrechnung 115, 117f
Anweisung 2
-, Druck- 5
-, Taschenrechner- 4
Anzeige-anweisungen 9
--, Steuergrößen für 187,
  219
- format 5, 19, 21, 108,
  121f, 129, 220f, 223,
  231, 239f, 242
- n von Ergebnissen 5
---, 13-stelliges 19
Approximation, sukzessive
  38f, 44, 115, 168, 183
arithmetische Hierarchie
  18, 213
- Operation, offen-
  stehende 10, 12, 231
Auflistung von Programmen 10
Aufruf von Unterprogrammen 11
Aufstellmöglichkeiten 226
Ausdruck 2
Ausgabegrößen 9
Ausgleichsgerade 89f
Auslöschung 32, 94

B

Bedingungen bei Vergleichen 3
Betragssummennorm 136
Bisektion 34
Blindoperation mit Klammern 18

C

charakteristisches Polynom 200

D

Damen-Aufgabe 226f
Datenspeicher-Belegung 9, 132
- platz 6
Defekt-normen 136, 137
- vektoren 135f
Determinanten 147
Dezimaldarstellung,
  normalisierte 16
Differentialgleichungen
  gewöhnliche 101f, 185f
  partielle 216
-, Systeme von 101
Differenzen,
  dividierte 63, 71
- approximation 185, 216
- gleichungen 217
- quotienten 51, 216
--, reziproke 53
- schema 61f
Drei-Achtel-Regel 104
Dreieckschema 63, 81
Dreiecksmatrix 147
Dreizehnstellige Anzeige 19
Druckanweisung 5, 9
Dsz-Anweisung 6

E

Eigenvektor 200
- näherungen 201
Eigenwert 200
- näherungen 202, 203
- aufgaben bei Matrizen 200f
Eingabe eines Programms 5
- größen 9
Einschrittverfahren 102f

Einzelschrittverfahren
für lineare Gleichungssysteme 170f, 180
für nichtlineare Gleichungssysteme 184f, 186f
Eliminationsverfahren 144f
für mehrere Gleichungssysteme 159f
Endlosform von Algorithmen 10
Endwert 3
euklidische Norm 136, 200
- s Skalarprodukt 200
Euler-Cauchy, verbessertes Verfahren von 103
Exponent 16
Extrapolation 73
- sverfahren 115f, 128f

F

Fehlerabschätzung 32, 75, 82, 105, 116, 136, 202
-, a-posteriori- 38, 45, 50, 169, 170, 184
-, a-priori- 34, 39, 45, 50, 170, 184
Fehler-funktion 78, 87
- meldung 10, 54, 94, 203f
- quadrat 89
Fibonaccische Zahlen 50
Fixpunkt 38, 43, 168, 183
Flag 69, 70, 187, 219, 241
Funktionalmatrix 184
Funktionen, Berechnung 22f

G

ganzzahliger Teil 13
Gaußsche Klammer 13
- s Eliminationsverfahren 144f
--, für mehrere Gleichungssysteme 159f
Gauß-Seidel-Verfahren für lineare Gleichungssysteme 170f
Genauigkeit, interne 19, 99
- sverluste 32, 94
Gesamtschrittverfahren
für lineare Gleichungssysteme 168f
für nichtlineare Gleichungssysteme 183f
Gitterpunkte 102, 114, 185, 217
Gleichheitszeichen 12
Gleichungssysteme
-, gestaffelte 146, 158, 167
-, lineare 135f
-, nichtlineare 183f

H

Heron, Verfahren von 46
Höchstzahlen 234
d'Hondtsches Verfahren 234f
Horner-Schema 22, 47, 228

I

INT 13
Integralsinus 30
Integration, numerische 74f
Interpolation 73
-, quadratische 54
-, iterative lineare 62, 71f
Interpolationspolynome 54, 61f, 75
-, Lagrange-Darstellung 61
-, Newtonsche Darstellung 62
-, Rekursionsformeln 62
Interpolationsverfahren 115f, 128f
Intervall-halbierung 33f
- schachtelungsverfahren 34
iterative lineare Interpolation 62, 71f
Iterative Verfahren
für lineare Gleichungssysteme 167f
für nichtlineare Gleichungssysteme 183f
für Nullstellen 33f
iterierte Vektoren 202

J

Jacobisches Gesamtschrittverfahren für lineare Gleichungssysteme 168f

K

Kettenlinie 198
Klammer [x] 13
- operationen 18, 87
komplementärer Index 134
kontrahierend 39, 184
Konvergenzordnung 105, 116

Kuntzmann, Verfahren 104
Kutta, Verfahren von 103

L
Leibnizsches Konvergenz-
kriterium 32
Linearfaktor 23
Lipschitz-Bedingung 39, 184

M
Mandatszuteilung 235
Mantisse 16
Marken 3, 5, 8, 11, 70
Matrix 131f
-, Dreiecks- 147
-, inverse 159, 165f
-, symmetrische 133, 201
-, positiv definite 170, 180
Matrizennorm
verträgliche 136, 185
Maximum 6
- norm 136, 184
Mehrschrittverfahren 114f
Milne-Prädiktor-Korrektor-
verfahren 115f, 129
- Simpson-Interpolations-
verfahren, 115f, 129
v.-Mises-Verfahren 201

N
Nachkommastellen
Abschneiden 13
Runden 15
Newtonsches Verfahren 44f
für Polynome 47f
Normen 136
normierte iterierte
Vektoren 202
Nullstellen-Berechnung 33f
-, einfache 44
Nyström-Extrapolations-
verfahren 115f, 129

O
optimal 89

P
Partialsummen 1, 26, 27f, 32
Pfeil bei Wertzuweisung 2
Polygonzugverfahren 103,
107, 112
-, verbessertes 103, 107, 112
Polynom, charakteristisches
200
Polynome 22f, 61f
-, Newtonsches Verfahren 47f
Potenz-methode 201f
- reihen 26, 29, 32
Prädiktor-Korrektor-
Verfahren 115f, 128f
Programm-abbruch 10, 53, 94
- auflistung 10
- schleifen 134
- speicherzeile 4

Q
Quadratische Interpolation 54
Quadratsummennorm 136
Quotientenkriterium 32

R
Randbedingungen 185, 216
Randwertaufgaben 185f, 198f
Rayleigh-Quotient 200
Rechenzeit 11, 158, 166,
179, 233
Reduktion von Ziffern 16
Regula falsi 50f
Reihen, unendliche 27f
Rekursionsformeln 1, 22,
27, 62f, 95
Relaxationsverfahren 180f
Romberg-Integration 79f
- Schema 81f
Rücksprung 3, 12
Rückwärtseinsetzen 147
Runden 15f
Rundungsfehler 35, 237
Runge-Kutta-Typ 102f
-- Verfahren 104, 107,
112, 118

S
Schach-Aufgabe 226f
Schema, iterative lineare
Interpolation 62
-, Differenzen- 61f
-, Horner- 22, 47, 228

Schema, Romberg- 81
Schleifen 3, 134
- index 3, 9, 134
--, komplementärer 134
- parameter 3, 9, 134
Schneiden 12f
Schrittweite 3, 69, 75, 80, 185, 193, 216
Schutzstellen 19, 167
Sehnentrapezformel 74f, 79f
Sekantenmethode 50
Simpsonsche Formel 74f, 80
Skalarprodukt
euklidisches 200
Spaltensumme, maximale 136
Speicher, virtueller 167
- abbildungsfunktion 132f
Speichern von Vektoren und Matrizen 131f
Sprung 3, 8, 11
- marken 5, 11
Stapelspeicher 8, 12, 25, 42, 143, 156f, 213
Steueroperationen, spezielle 91
Stützstellen 61f
-, äquidistante 63
sukzessive Approximation 38f, 44, 115, 168, 183

T

Tangententrapezformel 74f, 81
Tastenfunktion-Kurzform 10
-, Σ+ 92f, 100
Testregister 8, 18, 42
Trapezformeln 74f
Tschebyscheff-Approximation 26

U

Überrelaxation 180
Unterprogramme 11
Unterrelaxation 180

V

Vektoren 131f
-, normierte iterierte 202
Vektoriteration 201
Vergleiche 3
Verzweigungen 3, 8
Vorzeichenwechsel 33, 35, 54, 57

W

Wärmeleitungsgleichung 216f
Wertepaare 90
Wertzuweisung 2
Wiederholungsbereich 3

Z

Zahlenfelder 131, 167
Zeilensumme, maximale 136, 185
Zeilensummenkriterium, schwaches 169
Zeilenvertauschung 145f
Zeitbedarf 158, 166, 179, 233
Zeitzeile 217
Zuteilungsverfahren 235, 244f

Karl Hainer

Geboren 15. Dezember 1942 in Offenbach am Main. Ab 1962 Studium der Mathematik an der Universität Frankfurt, Diplom 1966, Promotion 1968. Von 1966 bis 1970 wissenschaftlicher Mitarbeiter am Lehrstuhl für Angewandte und Instrumentelle Mathematik an der Universität Frankfurt. 1971 Akademischer Rat, 1975 Akademischer Oberrat am Mathematischen Seminar der Universität Frankfurt.

Für Mathematiker, Informatiker, Naturwissenschaftler, Techniker, Studenten naturwissenschaftlicher und ingenieurwissenschaftlicher Fachrichtungen an Universitäten, technischen Hochschulen und Fachhochschulen, Benutzer programmierbarer Taschenrechner. Das Buch bringt eine Einführung in die Grundaufgaben der numerischen Mathematik und die praktische Durchführung der Lösungsalgorithmen auf programmierbaren Taschenrechnern. Mit Flußdiagrammen, übersichtlich gestalteten Programmen für die Rechner HP-67/97 und TI-58/59 und numerischen Beispielen.

ISBN 3-411-00805-9