Studienbücherei

H. Kaiser

Numerische Mathematik und Rechentechnik I

Mathematik für Lehrer
Band 9

Herausgegeben von:
W. Engel, S. Brehmer, M. Schneider, H. Wussing

Unter Mitarbeit von:
G. Asser, J. Böhm, J. Flachsmeyer, G. Geise, T. Glocke,
K. Härtig, G. Kasdorf, O. Krötenheerdt, H. Lugowski,
P. H. Müller, G. Porath

Studienbücherei

Numerische Mathematik und Rechentechnik I

H. Kaiser

Mit 60 Abbildungen
und 22 Tabellen

VEB Deutscher Verlag
der Wissenschaften
Berlin 1977

Verlagslektor: Dipl.-Math. B. Mai
Verlagshersteller: B. Winterstein
Umschlaggestaltung: R. Wendt

Printed in the German Democratic Republic
Lizenz-Nr. 206 · 435/101/77
Gesamtherstellung: IV/2/14 VEB Druckerei »Gottfried Wilhelm Leibniz«,
445 Gräfenhainichen · 4809
LSV 1084
Bestellnummer: 570 224 1
DDR 16,80 M

Vorwort

Dieser Band behandelt die im derzeit gültigen Lehrprogramm ausgewiesenen Themenkomplexe der Numerischen Mathematik und Rechentechnik unter Berücksichtigung damit zusammenhängender Gegenstände der Kybernetik. Auf diesen Gebieten ist die Meinungsbildung hinsichtlich der inhaltlichen Abgrenzung, des Abstraktionsgrades und der methodischen Gestaltung des Lehrgegenstandes noch sehr im Fluß. Die vorliegende Darstellung ist aus Vorlesungen entstanden, die der Verfasser an der Pädagogischen Hochschule „Karl Liebknecht" Potsdam gehalten hat, und berücksichtigt Erfahrungen, die sich bei der Einarbeitung in Forschungsaufträge der Industrie ergaben. Während der Arbeit am Manuskript sind ausgewählte Stoffgebiete in Schülerarbeitsgemeinschaften erprobt worden; eine erweiterte Fassung lag 1972–1974 einem WPA-Kurs[1]) an der Heinrich-Hertz-Schule Berlin zugrunde.

Das erste Kapitel soll zu einer prinzipiellen Auseinandersetzung mit den Fragen anregen, die entstehen, wenn man Praxisprobleme mit dem Ziel ihrer automatischen Lösung auf einer EDV-Anlage bearbeitet. Ein umfassendes Verständnis für die Schwierigkeiten der Modellbildung, algorithmischen Aufbereitung und Programmierung wird man allerdings erst in Verbindung mit einer hinreichend komplexen Anwendungsaufgabe gewinnen. Die ins einzelne gehende Erörterung aller Schritte der Analyse eines Problems und deren Ausführung mit solcher Präzision, daß schließlich ein leistungsfähiges Programm für seine Lösung resultiert, kann eine lohnende Thematik für einen Kurs der wahlobligatorischen Ausbildung sein.

Im zweiten Kapitel werden ausgewählte Gegenstände des digitalen Rechnens unter besonderer Berücksichtigung der für die Schule grundsätzlich wichtigen Stoffgebiete behandelt. Im Mittelpunkt stehen der algorithmische Aspekt der Arithmetik in Positionssystemen, deren technische Realisierung und damit zusammenhängende Fragen der Fehleranalyse. Auswahl und Darstellung der Themen-

[1]) Die Kurse der Wissenschaftlich-Praktischen Arbeit im Umfang von eineinhalb Jahren wurden 1969 in der Abiturstufe der erweiterten Oberschule eingeführt.

komplexe erfolgte auch in der Absicht, einen Beitrag zur Gestaltung des fakultativen mathematisch-naturwissenschaftlichen Unterrichts in der erweiterten Oberschule zu leisten.

Das dritte Kapitel soll an Hand von Beispielen in die wichtigsten Strukturen der Programmiersprache ALGOL 60 einführen. Unter Berücksichtigung dessen, daß zur Vorbereitung des einwöchigen Grundkurspraktikums nur acht Stunden Vorlesungen und vier Stunden Übungen zur Verfügung stehen, wurde versucht, ALGOL-Elemente in Verbindung mit bekannten Strukturen in Programmablaufplänen zu entwickeln. Die den inhaltlichen Erklärungen beigefügten Definitionen metalinguistischer Begriffe dienen im allgemeinen der nachträglichen Präzisierung und brauchen nur im Zweifelsfall zur Klärung der Zulässigkeit einer Sprachkonstruktion herangezogen zu werden. Der Abschnitt 3.4. über die R 300-Implementation von ALGOL soll eine Hilfe für die Durchführung des Praktikums an diesem Rechner sein. Methodische Bedeutung hat dieser Text für das Studium des fakultativen Studienabschnitts 3.5. über das Verhältnis von Syntax und Semantik einer Programmiersprache.

Das letzte Kapitel behandelt einige elementare Verfahren der Numerischen Mathematik. Dabei wird der algorithmische Aspekt hervorgehoben und eine Vertiefung der Kenntnisse über den Gebrauch von Prozeduren angestrebt.

Der Verfasser dankt allen, die durch konstruktive Kritik zur Verbesserung der Darstellung beigetragen haben, insbesondere den Herren des Herausgeberkollegiums, namentlich Prof. Dr. Schneider und Prof. Dr. Wussing, für wertvolle methodische bzw. philosophisch-historische Hinweise.

Meine Frau hat das Manuskript geschrieben und zusammen mit meinem Sohn Hans-Christoph eine Fülle wissenschaftsorganisatorischer Aufgaben, wie etwa die Erprobung von Programmen und Literaturrecherchen, besorgt. Sie haben wesentlichen Anteil an der Bewältigung einer für mich schwierigen Arbeitssituation. Dem Verlag danke ich herzlich für das Verständnis, das der verzögerten Fertigstellung des Textes entgegengebracht wurde, und den Mitarbeitern der Druckerei für die Sorgfalt, die sie dem schwierigen Satz angedeihen ließen.

Berlin, Sommer 1977 H. Kaiser

Inhalt

1. Arbeitsstufen der Problemanalyse

1.1. Einführende Beispiele

Die wissenschaftliche Bearbeitung eines Praxisproblems ist ihrem Wesen nach eine methodisch gestaltete Veränderung der Wirklichkeit. Im weitesten Sinne soll darunter auch die Ausdehnung der Naturerkenntnis und im besonderen die nach geeigneten Gütekriterien zu bestimmende optimale Anpassung an gegebene Anforderungen und Bedingungen verstanden werden. Voraussetzung dafür ist eine der Fragestellung angemessene Einsicht in den Sachverhalt und die Aufdeckung der darin maßgebenden Beziehungen. Wir betrachten dazu einige Beispiele und untersuchen anschließend die diesen Beispielen gemeinsamen Ansätze zur Auffindung von Problemlösungen.

1. Das im Vorderasiatischen Museum Berlin aufbewahrte Bruchstück eines Keilschrifttextes aus der Zeit 1800–1600 v. u. Z. enthält die Aufgabe, die Diagonale d eines rechteckigen Tores zu berechnen. Höhe und Weite sind mit $h = \frac{40}{60}$ GAR bzw. $w = \frac{10}{60}$ GAR angegeben; ein GAR entspricht etwa 6 m. Die babylonischen Mathematiker benutzten zur Zahldarstellung ein Sexagesimalsystem (Grundzahl 60), benötigten also 60 Ziffernzeichen (vgl. MfL Bd. 2, Anhang S. 148). Wir drücken diese dezimal aus und müssen dann bei der positionellen Niederschrift einer Zahl zwischen die so gebildeten Sexagesimalziffern ein Trennzeichen setzen. O. Neugebauer [54] folgend verwenden wir dafür das Komma und lassen auf den ganzen Teil einer Zahl in dieser Darstellung ein Semikolon folgen. Damit kann man zum Beispiel $w = 0;10$ und $h = 0;40$ schreiben. Die erwähnte Tontafel enthält die Lösung der Aufgabe in folgender übersetzter Form:

> Du: 0;10 Weite quadriere. 0;1,40 als Fläche siehst Du.
> Das Reziproke von 0;40 GAR bilde, mit 0;1,40 der Fläche multipliziere.
> 0;2,30 siehst Du. $\frac{1}{2}$ von 0;2,30 brich ab. 0;1,15 siehst Du.
> 0;1,15 zu 0;40 Höhe addiere.
> 0;41,15 ist seine Diagonale.

Der Berechnung liegt vermutlich der bereits den Babyloniern bekannte Satz des PYTHAGORAS (580–496 v. u. Z.) zugrunde. Für

$$d = \sqrt{h^2 + w^2}$$

bestimmt das Verfahren den Näherungswert

$$d \approx h + \frac{1}{2}\frac{1}{h} w^2.$$

Beachtet man die unterschiedliche Größenordnung von h und w, so mag man daraus folgern, daß der Wurzelwert zum Radikanden A, ausgehend von einem etwa einer Quadrattafel entnommenen Näherungswert a, nach der Formel

$$\sqrt{A} = \sqrt{a^2 + \Delta A} \approx a + \frac{1}{2}\frac{1}{a}\Delta A = \frac{1}{2}\left(a + \frac{A}{a}\right)$$

verbessert wurde. Wir werden diese noch heute benutzte Methode in Verbindung mit dem Newtonschen Verfahren zur Lösung von Gleichungen in 4.1.2. erörtern.

2. Als GALILEI um 1590 den freien Fall untersuchte, verfügte er nicht über Uhren, mit denen Zeiten in der Größenordnung von Sekundenbruchteilen hätten gemessen werden können. Es mußte ein Weg gefunden werden, den Vorgang zu verlangsamen und gewissermaßen in Zeitlupe zu betrachten. Dieser eröffnete sich durch die Idee, die Fallbewegung als Grenzsituation des Abrollens einer Kugel auf schiefen Ebenen wachsender Neigung zu betrachten. In seinen berühmten „Discorsi" [22] beschreibt GALILEI die mit Hilfe einer Wasseruhr als Zeitmesser durchgeführten Versuche. Die wesentlichen Textstellen lauten, frei übersetzt ([22], Bd. 2, S. 25): „Auf einer gut geglätteten Rinne von 12 Ellen Länge ließen wir bei verschiedener Neigung derselben eine polierte Messingkugel abrollen. Dieser Versuch wurde mit verkürzten Weglängen häufig wiederholt und ergab . . . für jede Neigung der Rinne, daß sich die Strecken wie die Quadrate der zur Durchlaufung benötigten Zeiten verhalten."

Durch Vergleiche mit den Ergebnissen von Pendelversuchen wird ferner plausibel gemacht ([22], Bd. 2, S. 15), daß – gleiche Höhe h (vgl. Abb. 1.1) bei den

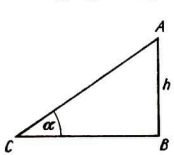


Abb. 1.1

schiefen Ebenen vorausgesetzt – die bei C erreichte Endgeschwindigkeit einen vom Neigungswinkel α unabhängigen Wert v_c annimmt. Auf Grund dieser beiden Feststellungen können die Probleme des freien Falls an der Bewegung auf der schiefen

Ebene für ein zur Beobachtung günstiges α studiert werden: Durch Differentiation [1]) erhält man aus dem Weg(s)-Zeit(t)-Gesetz

$$s = a\,t^2 \tag{1}$$

die Geschwindigkeit

$$v = \frac{ds}{dt} = 2at = 2\sqrt{as} \tag{2}$$

und speziell im Punkt C für $s_c = \frac{h}{\sin\alpha}$

$$v_c = 2\sqrt{\frac{ah}{\sin\alpha}},$$

also

$$a = a(\alpha) = \frac{v_c^2 \sin\alpha}{4h}. \tag{3}$$

Für den freien Fall durch die Strecke AB $\left(\alpha = \frac{\pi}{2}\right)$ ergibt sich aus (3)

$$a\left(\frac{\pi}{2}\right) = \frac{a(\alpha)}{\sin\alpha}. \tag{4}$$

Man kann also durch eine Weg-Zeit-Messung bei geeignet gewähltem α aus (1) den Wert $a(\alpha)$ und in den benutzten Einheiten gemäß (4) die Fallbeschleunigung zu

$$g = 2\,\frac{a(\alpha)}{\sin\alpha}$$

bestimmen.

Es ist aufschlußreich, damit die experimentellen Vorkehrungen zu vergleichen, die für eine unmittelbare Untersuchung des freien Falls erforderlich sind, wenn man einigermaßen genaue Messungen erhalten will (vgl. etwa [59], S. 15–16).

3. Zu untersuchen sind die gedämpften Schwingungen einer Masse m und die zeitliche Änderung der Ladung q eines Kondensators der Kapazität C in einem elektrischen Schwingkreis, in dem sich noch eine Spule der Induktivität L und ein Ohmscher Widerstand R befinden; $U(t)$ sei die eingeprägte Spannung. Die beiden physikalischen Systeme sind im Prinzip durch Abb. 1.2 beschrieben. Als Lagekoordinate für das mechanische System führen wir gemäß Abb. 1.2 die Variable x ein, deren Werte an einer geeignet zugeordneten Skala abgelesen werden. Die Aufgabe besteht darin, x in seiner Abhängigkeit von der Zeit t zu bestimmen. Wir nehmen noch an, daß von außen die zeitlich variable Kraft $F(t)$ einwirkt. Nach den Grundgesetzen der Mechanik ist diese beim Schwingungsvorgang im

[1]) Die Hilfsmittel der Differentialrechnung standen GALILEI nicht zur Verfügung. Mit seinen Studien zur gleichförmig beschleunigten Bewegung hat er jedoch wesentlich zur Herausbildung der Newtonschen Ideen beigetragen.

Gleichgewicht mit der Summe aus Trägheitskraft $m\ddot{x}(t)$, Reibungskraft $d\dot{x}(t)$ (d bedeutet die Dämpfungskonstante) und einer der Auslenkung x mit der Federkonstanten k proportionalen Kraft $kx(t)$. Auf diese Weise ergibt sich folgende Be-

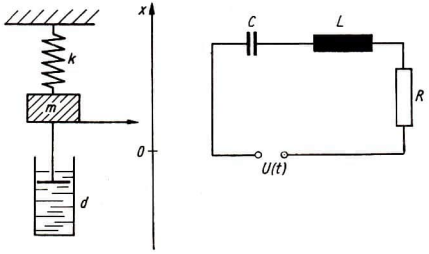


Abb. 1.2

ziehung zwischen der gesuchten Funktion und ihrer ersten und zweiten Ableitung:

$$m\ddot{x}(t) + d\dot{x}(t) + kx(t) = F(t). \tag{5}$$

Für den Schwingkreis erhält man auf Grund des Kirchhoffschen Maschensatzes Gleichheit zwischen der eingeprägten Spannung $U(t)$ und der Summe aus dem Spannungsabfall $\frac{q(t)}{C}$ am Kondensator, $R\dot{q}(t)$ am Ohmschen Widerstand und $L\ddot{q}(t)$ an der Spule. Man findet so die Gleichung

$$L\ddot{q}(t) + R\dot{q}(t) + \frac{1}{C}q(t) = U(t), \tag{6}$$

die bis auf die Bezeichnungen mit (5) übereinstimmt. Im wesentlichen handelt es sich also bei (5) und (6) um das gleiche mathematische Objekt, nämlich um eine lineare Differentialgleichung zweiter Ordnung. In dieser spiegelt sich das uns interessierende physikalische Geschehen wider, und Systeme, bei denen m, d, k, $F(t)$ und L, R, $\frac{1}{C}$, $U(t)$ entsprechend übereinstimmen, werden – wenn noch in gewissem Sinne gleiche Anfangsbedingungen gegeben sind – durch die gleichen Funktionen x und q beschrieben. Mit anderen Worten: Man kann Untersuchungen, die das eine betreffen, am anderen durchführen. Wegen der leichten Veränderbarkeit der Größen C, L, R ist es bequemer, mit dem elektrischen als mit dem mechanischen System zu experimentieren.

4. Als letztes Beispiel betrachten wir die optimale Organisation eines Produktionsprozesses unter folgenden Bedingungen (nach [8]): In einem Betriebsteil seien

m Maschinen zur Herstellung von n Erzeugnissen eingesetzt;

t_{ij} bezeichnet die Zeit, die an der i-ten Maschine zur Fertigung einer Mengeneinheit des j-ten Erzeugnisses benötigt wird;

c_{ij} sind die entsprechenden Kosten.

Die i-te Maschine kann nach den gegebenen Betriebsbedingungen monatlich maximal b_i Zeiteinheiten in Anspruch genommen werden. Der Plan schreibt vor: Von jedem Erzeugnis müssen monatlich mindestens a_j Einheiten produziert werden.

Die Organisationsaufgabe besteht darin, den Plan mit minimalen Kosten zu erfüllen, d. h., es sind unter dieser Zielsetzung die von der i-ten Maschine monatlich zu produzierenden Mengen x_{ij} des j-ten Erzeugnisses zu bestimmen. Die mathematische Formulierung der Aufgabe ist naheliegend: Da das gesamte Produktionsaufkommen am j-ten Erzeugnis durch $\sum_{i=1}^{m} x_{ij}$ gegeben ist, hat man

$$\sum_{i=1}^{m} x_{ij} \geqq a_j \quad (j=1, 2, \ldots, n) \tag{7}$$

zu fordern. Die zeitliche Inanspruchnahme der i-ten Maschine unterliegt der Einschränkung

$$\sum_{j=1}^{n} t_{ij} x_{ij} \leqq b_i \quad (i=1, 2, \ldots, m), \tag{8}$$

und natürlich ist

$$x_{ij} \geqq 0 \quad (i=1, \ldots, m;\ j=1, \ldots, n). \tag{9}$$

Die Produktionskosten werden durch die *Zielfunktion*

$$Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{10}$$

erfaßt; sie ist unter den genannten Nebenbedingungen zum Minimum zu machen. Diese Nebenbedingungen haben die Form linearer Ungleichungen, und auch die Zielfunktion ist linear. Aufgaben dieser Art sind Gegenstand der *linearen Optimierung*, deren Theorie in Teil II genauer untersucht wird.

1.2. Methodologische Betrachtung der Beispiele

Die in den Beispielen interessierenden Fragen betreffen Gegenstandsbereiche, die durch gewisse Beziehungen ihrer Elemente strukturiert sind. Im ersten handelt es sich um Lageverhältnisse eines Gebäudeteils, im zweiten und dritten um physikalische Objekte, deren Beziehungen zueinander durch Gesetze der Mechanik und des Elektromagnetismus bestimmt sind. Die Elemente des letzten Beispiels sind die Maschinen und die auf ihnen herzustellenden Erzeugnisse; Kopplungen zwischen diesen kommen durch Festlegung der oben eingeführten Größen x_{ij} zustande. In allen Fällen haben wir es mit einer Gesamtheit von aufeinander bezogenen, d. h. in

gewissen Relationen befindlichen Elementen zu tun; in der Kybernetik bezeichnet man diese mit dem oben schon umgangssprachlich benutzten Begriffsnamen *System*.

Den Beispielen ist gemeinsam, daß Problemstudien nicht an den betroffenen Objektbereichen, sondern an diesen in einer wesentlichen Hinsicht ähnlichen Abbildern durchgeführt werden. Im ersten und vierten Beispiel wird das zu lösende Problem unmittelbar einer mathematischen Frage zugeordnet: der Berechnung der Diagonalen in einem Rechteck bzw. der Lösung einer Aufgabe der linearen Optimierung. Wie in diesen Beispielen erfolgt die Abbildung von Realem auf mathematische Strukturen bei Sachverhalten, die sinnfällige Lage- oder Mengenverhältnisse beinhalten, mehr oder weniger spontan. Das folgende Zitat aus [1] mag als Versuch einer Begründung dafür gelten:

„Die erste allgemeine Struktur, die ein Gegenstand der praktischen Aneignung und der Widerspiegelung in allerersten Abstraktionen wurde, war die Struktur der endlichen Mengen mit ihren Beziehungen . . . Gleichzeitig vollzog sich die praktische und theoretische Aneignung einer anderen allgemeinen Struktur – der geometrischen, welche die räumlichen Beziehungen der Körper und ihrer Teile und damit deren räumliche Formen umfaßte."

Im zweiten und dritten Beispiel sind es materielle Systeme, die in einem bestimmten Verhalten mit den Originalen vergleichbar sind: Aus schon genannten experimentellen Gründen wird der freie Fall an der schiefen Ebene und eine mechanische Schwingung mit Hilfe eines elektrischen Schwingkreises untersucht. In beiden Fällen verhilft die mathematische Beschreibung der Sachverhalte zu der Einsicht, wie das zu geschehen hat, d. h. welche Zuordnungen zwischen Original und Bildbereich zu beachten sind. Diese kann aber auch selbst als Abbild betrachtet und zur Erkenntnisgewinnung über das Original herangezogen werden. So ist es etwa möglich, statt der Untersuchung einer physikalischen Schwingung durch eine andere die Differentialgleichung 1.1.(5) bzw. (6) zu lösen.

Aus den bisherigen Betrachtungen wollen wir die Beschreibung der folgenden Problemsituation und Lösungsmethode ableiten: (1)

a) Für einen gegebenen Objektbereich (ein System) O sind bestimmte Fragen zu beantworten. Diese können zweckorientiert auf die Beeinflussung des Verhaltens eines Subjektes S gegenüber O gerichtet sein.

b) An Stelle von O wird ein dazu in bestimmter Hinsicht ähnlicher Bildbereich M untersucht, dessen *Analogie* zu O darin besteht, daß Fragen, die O betreffen, in solche für M übertragen und dazu gefundene Antworten bezüglich O interpretiert werden können.

Ein derartiges Bildsystem heißt ein *Modell* von O. Da die Analogie im allgemeinen nur bezüglich eines bestimmten Sachverhaltes gegeben ist, muß auch auf dieses „wozu" bei der Charakterisierung von M eingegangen werden. Abb. 1.3 gibt die in (1) a), b) beschriebene *Modellmethode* schematisch wieder.

Da O und M Systeme, also strukturierte Elementmengen sind, ist der Versuch naheliegend, die bestehenden Analogien in Anlehnung an die in der Theorie mathe-

matischer Strukturen verwendeten Begriffe *Homomorphie* und *Isomorphie* zu beschreiben. Eine methodologische Bestimmung der Sachlage in diesem Sinne wurde bereits von HEINRICH HERTZ (1857–1894) in der Einleitung seiner „Prinzipien der Mechanik" formuliert: Das Wesen eines Modells drückt sich darin aus, daß „die

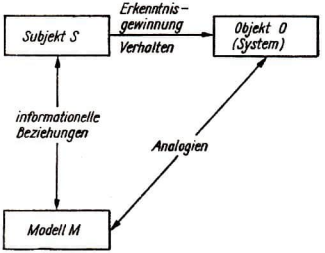


Abb. 1.3

denknotwendigen Folgen der Bilder" sich stets erweisen als die Bilder „von den naturnotwendigen Folgen der abgebildeten Gegenstände".

Neben realen Modellen sind in unseren Erörterungen mathematische Strukturen als Widerspiegelung objektiver Sachverhalte aufgetreten, die im Erkenntnisprozeß die Rolle von M gemäß dem Schema der Abb. 1.3 übernehmen können. Beispielsweise läßt sich die Untersuchung des freien Falls an der Differentialgleichung 1.1. (5) durchführen. Bei Vernachlässigung des Luftwiderstandes erhält man bezüglich des Koordinatensystems von Abb. 1.2 für $m=1$, $d=0$, $k=g$, $F(t)=0$ ein *mathematisches Modell* des Bewegungsablaufs. Im Fall der optimalen Organisation des betrachteten Produktionsprozesses ist ein solches Modell durch die Zielfunktion 1.1. (10) in Verbindung mit den linearen Ungleichungen (7) bis (9) gegeben.

Wie das Beispiel der mechanischen und der elektrischen Schwingungen zeigt, können verschiedene physikalische Systeme durch das gleiche mathematische Modell – im vorliegenden Fall eine lineare Differentialgleichung zweiter Ordnung – dargestellt werden. Das Verhältnis von Modell zum Modellierten ist hier also das des Allgemeinen zum Besonderen.

Einer anderen, in dieser Hinsicht umgekehrten Auffassung des Begriffs des mathematischen Modells begegnet man in den Grundlagen der Mathematik, wo dieses als Interpretation einer axiomatisch charakterisierten Struktur auftritt. Berühmte Beispiele hierfür sind die Modelle von F. KLEIN und H. POINCARÉ für die ebene hyperbolische Geometrie, mit deren Hilfe es u. a. gelang, die Unabhängigkeit des Parallelenaxioms von den übrigen Axiomen der euklidischen Geometrie zu beweisen.

Im Rahmen der vorliegenden Analogie sind die Rolle von Modell und Modelliertem austauschbar. So kann man etwa den elektrischen Schwingkreis als Modell der Differentialgleichung 1.1. (6) auffassen. Diese Betrachtungsweise liegt der Konzeption des *Analogrechners* zugrunde.

Die mit der Modellmethode durchzuführenden Systemuntersuchungen betreffen in mathematischer Hinsicht wesentlich zwei Fragen:

Wie verhält sich ein vorgegebenes System unter bestimmten Bedingungen? (2)

Wie muß ein System bestimmter Art beschaffen sein, damit es ein vorgegebenes Verhalten (möglichst gut) realisiert? (3)

Ein einfaches Beispiel für die *Analyse*aufgabe (2) ist die Bestimmung der Fallbewegung eines Körpers, der zu einem Zeitpunkt $t = t_0$ die Anfangslage x_0 und Anfangsgeschwindigkeit v_0 hat.

Das *Synthese*problem (*system design*) (3) setzt eine Klasse gleichartiger, von gewissen Parametern abhängender Systeme voraus, aus der ein im Verhalten optimales ausgewählt werden soll. Als Beispiel mag das Produktionssystem dienen, in dessen mathematischem Modell die Größen x_{ij} die Rolle solcher Parameter spielen. Wie hier wird auch allgemein die Güte eines Systems durch den Wert einer von den Parametern abhängenden Zielfunktion bestimmt, und optimale Systeme sind durch absolute Extrema derselben charakterisiert. Durch ihre Bedeutung und eventuelle Beschränkungen bei der technischen Realisierung ergeben sich für die Parameter im allgemeinen Nebenbedingungen (*Restriktionen*), die bei der Optimierung zu beachten sind. Dafür geeignete Methoden werden in der *Operationsforschung* (*operations research*) entwickelt und untersucht.

Die Formulierung der Syntheseaufgabe gemäß (3) entspricht der Problemsituation in der Technik und im Organisationswesen. In den angewandten Naturwissenschaften hat man es demgegenüber mit der Erkennung eines real gegebenen Systems zu tun. Man denke z. B. an die Erkundung von Lagerstätten durch Auswertung von Anomalien des Schwerefeldes. Solche Aufgaben können mit Hilfe von Hypothesen über die Struktur des aufzuklärenden Systems auf ein Problem des Typs (3) zurückgeführt werden, wobei die Zielvorgaben der Technik durch Meßwerte aus Experimenten mit dem Erkundungsobjekt zu ersetzen sind (Fragen an die Natur). Der Erkenntnisprozeß vollzieht sich dann nach einer *Trial-and-error*-(Versuch und Irrtum)-*Methode*, bei der nach erfolgter Parameterbestimmung weitere Erfahrung vorausberechnet und mit der wirklichen verglichen wird. Hinreichende Übereinstimmung spricht für die Wahl der Hypothesen, Abweichungen erfordern ihre Veränderung. Der in seinen philosophischen Positionen materialistische Physiker L. Boltzmann [13] hat diese Entwicklung in den Naturwissenschaften mit der Evolution der Organismen verglichen und im Sinne eines Darwinismus von Theorien interpretiert.

Weiterhin interessieren uns ausschließlich mathematische Modelle. Ihre Auffindung kann naheliegend, andererseits aber auch geniale Vollendung einer langen wissenschaftlichen Entwicklung sein. Als Beispiel für das letzte sei etwa die zusammenfassende Beschreibung der elektromagnetischen Erscheinungen durch die Maxwellschen Gleichungen genannt. Das methodisch Wesentliche der Modellierung kommt dann zum Tragen, wenn die einem Praxisproblem entsprechende mathematische Aufgabe mit Hilfe eines endlichen Verfahrens konstruktiv gelöst werden kann. Ein solches ist allerdings kaum von Interesse, wenn es nur zur Beantwortung einer speziellen, in jeder Hinsicht festgelegten Frage geeignet ist. Vielmehr wird

man erwarten, daß dieses die Probleme einer gewissen Klasse löst. Wir sprechen dann von einem *Algorithmus* und charakterisieren diesen vorläufig als

> *ein durch einen Text endlicher Länge beschreibbares Verfahren zur schematischen Beantwortung von Fragen eines gewissen Problemkreises.*[1] (4)

Man könnte meinen, daß es sich bei dem Keilschrifttext des ersten Beispiels in 1.1. um eine der ausgeschlossenen uninteressanten Verfahrensvorschriften handelt. Tatsächlich liegt hier aber ein Grundmuster vor, das man in der mesopotamischen Mathematik wiederkehrend mit anderen Zahlwerten „aktualisiert" findet und von dieser als ein allgemeines Verfahren im Sinne von (4) zur näherungsweisen Berechnung von Rechteckdiagonalen erkannt wurde.

Die Erklärung (4) ist keine Definition im eigentlichen Sinne, sondern eine Beschreibung inhaltlicher Vorstellungen, wie sie sich bei der Bildung des Begriffs „Algorithmus" entwickelt haben. Zu einer auch für die Schule brauchbaren Präzisierung gelangt man durch den Umgang mit einer Darstellungstechnik für solche Verfahren und daran anschließende Abstraktionen. Davon handelt Abschnitt 1.3.

1.3. Zum Algorithmusbegriff

Das Wort Algorithmus wird von dem Namen des arabischen Mathematikers Muhamed Ibn Musa al-Hwârazmî abgeleitet, der um 825 u. Z. in Bagdad wirkte und dessen Bücher einen bedeutenden Einfluß auf die Entwicklung der abendländischen Mathematik hatten.[2] Er wurde in der Oasenstadt Choresm geboren, die heute Chiwa heißt und südlich des Aralsees in der Usbekischen SSR liegt. al-Hwârazmî war ein Antagonist der griechischen Schule und in seinem Schaffen der orientalischen Mathematik verhaftet. Diese kann in einem gewissen Sinne als eine (vgl. [1], S. 254) praxisbezogene „ideale Technik" beschrieben werden, in der empirische und logisch-deduktive Elemente in einer noch weithin unbekannten Weise miteinander verknüpft sind. Sie ist uns in Form von Anweisungen zum Lösen gewisser Aufgaben von der Art des ersten Beispiels in 1.1. überliefert und läßt sich an Hand von Keilschrifttexten bis in sumerische Zeit (3. Dynastie von Ur, etwa 2000 v. u. Z.) zurückverfolgen. Die Vorstellung von einem Algorithmus im Sinne der Erklärung von 1.2. (4) ist also schon sehr alt. Wir wollen zunächst die darin zum Ausdruck gebrachte Formulierung von Antworten auf gewisse Fragen präzisieren.

---

[1] Knuth beschreibt das Wesen eines Algorithmus in [41], S. 607, so: algorithms are „precise rules for transforming specified inputs into specified outputs in a finite number of steps".

[2] Viele mittelalterliche europäische Texte zur Mathematik beginnen unter Berufung auf ihn mit den Worten „Dixit Algorismi . . ." (So spricht Algorismi . . .). (Diesen Hinweis verdanke ich einer Mitteilung von Herrn Professor Dr. Wussing.)

Fragen und Antworten setzt eine Sprache und damit die Benutzung eines *Alphabets* voraus. Darunter sei hier eine endliche Menge Σ von Zeichen

$$x_1 | x_2 | \ldots | x_n \text{[1]}$$

verstanden, die man auch *Buchstaben* nennen kann. Endliche Symbolfolgen

$$x_{i_1} x_{i_2} x_{i_3} \ldots x_{i_k}$$

heißen *Wörter* über Σ; k nennt man die Länge des Wortes. Zwei Wörter sollen gleich sein, wenn sie gleiche Länge haben und buchstäblich übereinstimmen. Σ^* bedeute die Menge der Wörter über Σ, welche auch das mit ε bezeichnete leere Wort enthalten soll. Die Hinzunahme dieses fiktiven Wortes ohne Buchstaben ist für die Zwecke der Algorithmentheorie bequem. Das Wesentliche der Erklärung 1.2. (4) kann nun so ausgedrückt werden:

Ein Algorithmus ordnet gewissen als Aufgaben oder Fragen zu interpretierenden Wörtern über einem Alphabet Σ ein(e) (Ant)wort aus Σ^* zu.

Über der als Problemklasse[2]) ausgezeichneten Wortmenge W realisiert der zu betrachtende Algorithmus eine Abbildung (*Alphabetoperator*)

$$A: W \to \Sigma^*. \tag{1}$$

Jede Teilmenge $W \subseteq \Sigma^*$ heißt eine *Sprache über* Σ. Im Zusammenhang mit der Erörterung von ALGOL 60 in Kapitel 3 werden wir dafür eine Methode der Beschreibung kennenlernen.

Den mit (1) zusammenhängenden Aspekt wollen wir am Beispiel des Euklidischen Algorithmus zur Bestimmung des größten gemeinsamen Teilers zweier positiver natürlicher Zahlen (vgl. MfL Bd. 1, S. 155) m, n betrachten. Zuvor erläutern wir daran noch eine graphische Darstellung für Algorithmen, die u. a. auch zur Vorbereitung der automatischen Abarbeitung nützlich ist und deshalb als *Programmablaufplan* (PAP, Flußdiagramm, Flußbild) bezeichnet wird. Eine solche Figur dient vor allem der menschlichen Verständigung über einen Algorithmus und ist meist noch mit mehr oder weniger Kommentar versehen. Ihr Entwurf enthält daher einige Willkür hinsichtlich des Grades an kontextfreier Verständlichkeit. Unter *Programmierung* wollen wir uns in erster Näherung die Formulierung eines Algorithmus in einer für eine Maschine verständlichen Sprache vorstellen. Im Ergebnis ist ein *Programm* immer eine Zeichenfolge, die diese zur Realisierung der Abbildung (1) befähigt.

Nach dem Euklidischen Algorithmus wird die nicht kleinere der beiden Zahlen (x) durch die andere (y) mit Rest geteilt:

$$x = q \cdot y + r, \tag{2}$$

wobei $0 \leqq r < y$ ist. Beim nächsten Schritt übernehmen y und r die Rollen von x

[1]) Das Symbol | wird als Trennzeichen benutzt und gehört ebenso wie . . . nicht zu Σ.

[2]) Gelegentlich bezeichnet man eine solche Problemklasse auch als das „allgemeine Problem", welches der Algorithmus löst.

bzw. y, und man gewinnt

$$y = q_1 \cdot r + r_1 \qquad \text{mit} \qquad 0 \leqq r_1 < r;$$

so fortfahrend ergibt sich

$$\begin{aligned} r &= q_2 \cdot r_1 + r_2, & 0 &\leqq r_2 < r_1, \\ r_1 &= q_3 \cdot r_2 + r_3, & 0 &\leqq r_3 < r_2, \\ &\cdots\cdots\cdots & &\cdots\cdots \end{aligned}$$

Die Folge dieser Gleichungen bricht nach endlich vielen Schritten ab, da die Reste r, r_1, r_2, ... sukzessive kleiner werden und einmal der Fall eintritt, daß die Division aufgeht. Bekanntlich ist der letzte von Null verschiedene Rest der gesuchte größte gemeinsame Teiler von m und n. Bei der Durchführung des Verfahrens sind nur die für jeweils den nächsten Schritt erforderlichen Operanden wesentlich, alle sonstigen Zwischenresultate können „gelöscht" werden. Das führt dazu, nur mit den Variablen der Gleichung (2) zu arbeiten und diese bei jedem Rechenschritt mit den jeweils *aktuellen* Werten zu belegen. Die Zuweisung des Wertes a etwa an die Variable x wird mit Hilfe eines *dynamischen Gleichheitszeichens* durch

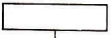
$$x := a^{1}) \tag{3}$$

ausgedrückt. Man denke sich dabei x etwa als den Namen eines Speicherplatzes, auf dem (unter Löschen eines eventuell schon dort stehenden Wertes) vermöge der Wertzuweisung (3) a deponiert wird. An Stelle von a kann ein *arithmetischer Ausdruck* in Erscheinung treten, der vor der Wertzuweisung mit den darin gegebenen Operanden zu berechnen ist. Speziell wäre beispielsweise

$$x := x + 1 \tag{4}$$

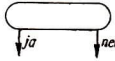
möglich. In (4) würde der momentane Wert von x um 1 zu erhöhen und das Resultat der Variablen x als neuer aktueller Wert zuzuweisen sein.

Abb. 1.4 zeigt einen Programmablaufplan für den Euklidischen Algorithmus. Dieser enthält die Symbole 

zur Kennzeichnung des Programmanfangs bzw. -endes. In die Operationskästchen  mit einem Ausgang wird die Anweisung eingetragen, die an der betreffenden Programmstelle auszu-

[1]) Das Zeichen := wird in dieser Lehrbuchreihe auch zur Definition von Termen benutzt. Es ist als Wertzuweisungszeichen in ALGOL 60 und andere Programmiersprachen eingegangen und daher hier nicht vermeidbar. Aus dem Zusammenhang wird klar werden, welche dieser Bedeutungen zutreffend ist.

führen ist; als Spezialfall davon erscheint das Symbol  für Datenein-

gabe und -ausgabe. Den mit zwei Ausgängen versehenen Symbolen 

sind Aussagen zugeordnet, deren Wahrheitswert den Programmablauf steuert: Je nachdem, ob die Frage nach der Wahrheit der in einem solchen „Entscheidungs-

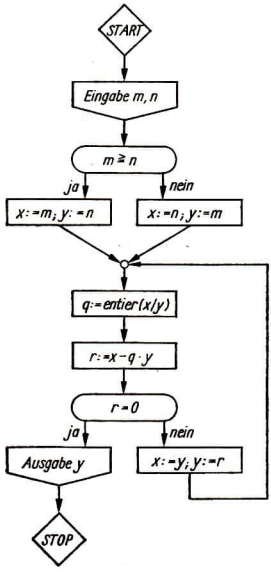


Abb. 1.4

kästchen" stehenden Aussage mit ja oder nein zu beantworten ist, verläßt man dieses über den entsprechend gekennzeichneten Zweig. Häufig beziehen sich solche Entscheidungsfragen auf arithmetische Vergleiche.

Beim Durchlaufen der *Programmlinie* in Pfeilrichtung erfolgt nach dem Start zunächst die Eingabe der beiden ganzen Zahlen, für die der größte gemeinsame Teiler ermittelt werden soll. In einer EDV-Anlage müßten dazu entsprechende

Speicherplätze bereitgestellt werden. Sodann wird getestet, ob $m \equiv n$ ist, und das Programm *verzweigt* sich: Falls die Testfrage mit „ja" zu beantworten ist (die im Entscheidungskästchen stehende Aussage wahr ist), wird die Wertzuweisung

$$x := m; \quad y := n,$$

sonst

$$x := n; \quad y := m$$

veranlaßt. Auch für x, y und alle noch in Erscheinung tretenden Variablen sind Speicherplätze bereitzustellen. Man beachte, daß trotz der Programmverzweigung bei der auf einen konkreten Fall bezogenen Abarbeitung des Algorithmus nur ein bestimmter Zweig durchlaufen wird.

Die nächste Anweisung belegt die Variable q mit dem ganzen Anteil des Quotienten x/y. Das wird hier mit der von A. M. LEGENDRE (1752–1833) in seiner „Théorie des nombres" eingeführten Funktion entier ( ) ausgedrückt, für die C. F. GAUSS (1777–1855) das Klammersymbol [ ] benutzte.

Weiterhin der Programmlinie folgend, erhält r den Wert des arithmetischen Ausdrucks $x - q \cdot y$ zugewiesen.

Damit sind die aus der ersten Division mit Rest resultierenden Größen des Euklidischen Algorithmus ermittelt. Nunmehr wird festgestellt, ob $r = 0$ ist und der Algorithmus abbricht. Falls ja, ist y der größte gemeinsame Teiler und wird als solcher ausgedruckt; falls nein, erfolgt die Wertzuweisung

$$x := y; \qquad y := r$$

(man achte auf die Reihenfolge), und die Rechnung beginnt mit den also aktualisierten Werten von neuem an der Stelle des Ablaufplanes, in welche die zurückführende Programmlinie einmündet. Auf diese Weise entsteht ein Zyklus, der erst verlassen wird, wenn $r = 0$ ist. Allgemein bezeichnet man ein Programm als *zyklisch*, wenn ein Teil desselben mehrfach durchlaufen wird.

Bei der Programmierung eines Algorithmus sind Irrtümer möglich. Da diese bei einer automatischen Problembearbeitung erhebliche Störungen und Kosten verursachen können, ist eine *Programmprüfung* unerläßlich. Diese vollzieht sich in der Regel in Form eines sogenannten *Trockentests*, bei dem der Bearbeiter einen oder mehrere konkrete Fälle auf Grund der programmierten Anweisungen und nur dieser durchspielt und die Resultatgrößen auf ihre Richtigkeit überprüft. Der Test ist so anzulegen, daß man bei einem verzweigten Programm sämtliche Zweige durchläuft. Zu diesem Zweck ist ein Rechenblatt gemäß dem reservierten Speicherraum so zu organisieren, daß eine fortlaufende Aktualisierung sich bei der Programmabarbeitung verändernder Größen möglich wird. Zur genaueren Erläuterung testen wir den Euklidischen Algorithmus für die Werte

$$m = 64 \qquad \text{und} \qquad n = 28.$$

Auf einem Rechenblatt werden dazu für die Variablen m, n, x, y, q und r sechs Spalten eingerichtet, in die untereinander diejenigen Werte eingetragen werden, die

sich bei der Abarbeitung für diese Größen ergeben (der letzte in einer Spalte erscheinende Wert ist der an der jeweiligen Programmstelle aktuelle, derjenige also, der im Computer dann auf dem für diese Variable reservierten Speicherplatz stehen würde):

m	n	x	y	q	r
64	28	64	28	2	8
		28	8	3	4
		8	4	2	0

Der beim Abbruch des Algorithmus aktuelle Wert 4 von y ist der gesuchte größte gemeinsame Teiler von 64 und 28.

Die schriftliche Fixierung eines Trockentests in der beschriebenen Weise bezeichnet man als ein *Protokoll* des betreffenden Algorithmus.

Unabhängig von der Programmprüfung hat die Niederschrift solcher Protokolle methodische Bedeutung. U. a. können auf diese Weise die erfahrungsgemäß auftretenden Schwierigkeiten beim Umgang mit dem Zuweisungszeichen := ausgeräumt und die Neubewertung von Variablen (vgl. (4)) an Stelle der Einführung neuer Variablen geübt werden. Wesentlich ist jedoch, daß sich darin der durch einen Algorithmus erzeugte Alphabetoperator ausdrückt. – Beim Euklidischen Algorithmus wird jede Aufgabe der zugehörigen Problemklasse (Bestimmung des größten gemeinsamen Teilers zweier natürlicher Zahlen) in der Symbolik von MfL Bd. 1, S. 151, durch

$$m \sqcap n \tag{5}$$

wiedergegeben. Betrachten wir die natürlichen Zahlen in ihrer dezimalen Ziffterndarstellung, so erscheint (5) und die Problemlösung als Wort über dem Alphabet

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \sqcap\}.$$

Für den bezüglich Σ zum Euklidischen Algorithmus gehörenden Alphabetoperator (1) gilt auf Grund des für ein spezielles Problem geschriebenen Protokolls

$$A: 64 \sqcap 28 \mapsto 4.$$

Da verschiedene Algorithmen den gleichen Alphabetoperator erzeugen können, muß man zwischen beiden Begriffen unterscheiden. Der Alphabetoperator ist als eindeutige Abbildung von einer Wortmenge in eine andere hinreichend bestimmt; bezüglich des erzeugenden Verfahrens, das wir Algorithmus genannt haben, soll im weiteren eine Präzisierung erfolgen. Dazu betrachten wir noch einmal den Programmablaufplan der Abb. 1.4, sehen aber von der besonderen Bedeutung der Programmierkästchen ab. Ersetzen wir diese durch einheitliche Symbole – etwa Kreise – so entsteht eine Figur von der Art der Abb. 1.5, die man als *Graph* bezeichnet. Die von den Kreisen und orientierten Verbindungen repräsentierten

Elemente heißen *Knoten* bzw. *gerichtete Kanten* oder *Bögen*. Wesentlich ist dabei nur die *Inzidenz* von Bögen und Knoten, d. h. die Zuordnung geordneter Paare von Elementen einer bestimmten Menge zu den Elementen einer anderen Menge. Die figürliche Darstellung ist Anschauungshilfsmittel und bis auf die richtige Wiedergabe der Inzidenz willkürlich. Der zu Abb. 1.5 gehörende Graph heißt

- *endlich*, weil nur endlich viele Knoten und Kanten in Erscheinung treten,
- *gerichtet*, weil seine Kanten einen Durchlaufssinn haben,
- *zusammenhängend*, weil irgend zwei Knoten durch einen Weg aus Bögen (ohne Berücksichtigung der Orientierung) verbunden werden können.

Wir bemerken, daß genau ein *Eingangsknoten* und genau ein *Ausgangsknoten* vorhanden ist, in den kein Bogen einmündet bzw. von dem keiner austritt. In alle

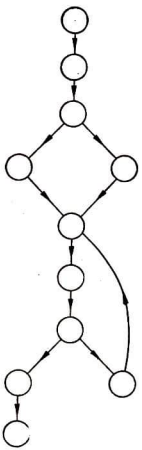


Abb. 1.5

übrigen Knoten münden ein oder mehrere Bögen ein; die Anzahl der austretenden ist 1 oder 2. Um von dem Graphen zum Programmablaufplan des Euklidischen Algorithmus zurückzukommen, muß man diesen geeignet interpretieren; dabei sind den Knoten gewisse Operationen oder Entscheidungen zuzuordnen und die Bögen als Programmlinienstücke zu deuten. Von dieser Vorstellung ist L. A. KALOUJNINE ausgegangen, um zu einer Präzisierung des Algorithmusbegriffs zu gelangen. Zunächst werden zwei endliche Mengen

$$\mathfrak{A} = \{\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_n\} \quad \text{und} \quad \Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_m\} \tag{6}$$

bestimmt, deren Elemente *Operatoren* bzw. *Testoperatoren* heißen und eine gewisse mengentheoretische Interpretation erfahren. Ein Algorithmus im Sinne von KALOUJNINE kommt nun dadurch zustande, daß man einen Graphen mit den Eigenschaften wählt, die im Anschluß an die Betrachtung von Abb. 1.5 erwähnt wurden, und jedem vom Ein- und Ausgang verschiedenen Knoten P ein Element aus $\mathfrak{P}$ bzw. Φ zuordnet. Letzteres in Abhängigkeit davon, ob aus P ein oder zwei Bögen austreten. Je nach Art der bezüglich (6) vorgenommenen Interpretation gelangt man zu speziellen Algorithmenbegriffen, etwa dem von A. A. MARKOV oder A. M. TURING. Auf eine systematische Behandlung von Fragen der Algorithmentheorie wird hier nicht eingegangen; der Leser sei für ein weiterführendes Studium auf die bequem zugängliche Arbeit von L. A. KALOUJNINE [36] und auf [30] verwiesen.

Abschließend wollen wir die mehr grundsätzlichen Betrachtungen dieses Kapitels zu einer Methode zusammenfassen, die man bei der mathematischen Bearbeitung eines Praxisproblems mit Hilfe von Rechenautomaten zu befolgen hat. Diese besteht im wesentlichen aus drei Schritten, welche gewöhnlich als *Arbeitsstufen der Problemanalyse* bezeichnet werden.

I. Aufstellung eines mathematischen Modells und Formulierung einer dem Praxisproblem entsprechenden mathematischen Aufgabe.

Zum Beispiel:

Praxisproblem: Optimale Organisation eines Produktionsprozesses.

Mathematisches Modell: Beschreibung der Produktionsbedingungen und -anforderungen durch ein System linearer Ungleichungen. Erfassung der Kosten in Form einer linearen Funktion.

Die *mathematische Aufgabe* besteht in der Minimierung dieser Zielfunktion unter der Nebenbedingung, daß die linearen Ungleichungen erfüllt sind.

II. Bestimmung eines *Lösungsalgorithmus* für das Modellproblem, d.h. Angabe eines konstruktiven Verfahrens im Sinne von 1.2. (4) zur Gewinnung der Lösung. Für das Beispiel einer Aufgabe der linearen Optimierung leistet der von G. B. DANTZIG 1947 veröffentlichte Simplexalgorithmus[1]) das Gewünschte.

III. Bearbeitung der Fragen, die mit der Abarbeitung des Algorithmus auf einem Rechenautomaten zusammenhängen. Dazu gehört nicht nur dessen Programmierung, z. B. in einer sogenannten problemorientierten Sprache wie ALGOL 60, sondern auch eine Untersuchung hinsichtlich der erforderlichen Rechenzeit und numerischen Stabilität. Das letzte betrifft das Verhalten eines für einen mathematischen Zahlbereich konzipierten Verfahrens auf der in einem Rechner gegebenen endlichen Menge von Maschinenzahlen und erfordert eine Einschätzung der auftretenden Rundungsfehler. Da über derartige Fragen der Bewertung und Optimierung von Algorithmen laufend publiziert wird, ist es geboten, die Arbeitsstufe II mit einer sorgfältigen Literaturrecherche zu verbinden.

[1]) Man vergleiche dazu MfL Bd. 10, Kap. 6.

2. Datenverarbeitung in Digitalrechnern

2.1. Programmgesteuerte Digitalrechner

2.1.1. Die folgenden Betrachtungen betreffen Aufbau und Funktionen *universeller programmgesteuerter Digitalrechner*. Universalität bedeutet die Eigenschaft, einen beliebigen Algorithmus realisieren zu können; „digital" bezieht sich auf das Rechnen mit Ziffern (digit [engl.]: Finger, Zehe). Die Arbeitsweise einer solchen Maschine vollzieht sich in diskreten Schritten, im Unterschied zur analogen Rechentechnik, wo die in einem Problem auftretenden Operanden durch kontinuierlich veränderbare physikalische Größen (z. B. Längen, Spannungen, Ströme) dargestellt und die Ergebnisse durch gewisse Experimente mit diesen gefunden werden. Nicht automatisch arbeitende Analogiegeräte werden als *mathematische Instrumente* bezeichnet; ein einfaches Beispiel ist der Rechenstab.

Mit Rücksicht auf die Behandlung des Gegenstandes in der Schule (vgl. [58]) erörtern wir die Funktionsgruppen eines Digitalrechners und die Darstellung ihres Zusammenwirkens in einem Blockschaltbild im Vergleich mit der Analyse der Tätigkeit eines Menschen, der bei der Abarbeitung eines Algorithmus eine Tischrechenmaschine benutzt. Dabei wird zunächst ein Formular angelegt, welches Informationen über das durchzuführende Verfahren enthält und außerdem so gestaltet ist, daß Zwischenergebnisse übersichtlich eingetragen und zur weiteren Verwendung entnommen werden können. Auch Nebenrechnungen sollten dort im Rahmen geeigneter Schemata und nicht unkontrollierbar auf Zetteln durchgeführt werden. In den Ablauf des Verfahrens können darüber hinaus noch externe Wissensspeicher – wie Formelsammlungen und Tafelwerke – einbezogen sein. Eingangsdaten und die gesuchten Endergebnisse denke man sich je auf einem besonderen Blatt notiert. Die Wechselbeziehungen in diesem informationsverarbeitenden System können etwa durch das in Abb. 2.1 angegebene und in seiner Bedeutung wohl unmittelbar verständliche Blockschaltbild beschrieben werden.

In einem Rechenautomaten entspricht dem Menschen das *Steuerwerk*, der Tischrechenmaschine das *Rechenwerk* und dem Formular der *Hauptspeicher* (*Arbeitsspeicher*) der Anlage. Diese Funktionsgruppen sind nach dem Blockschaltbild in Abb. 2.2 zur *Zentraleinheit* zusammengeschlossen, die ihrerseits mit den peripheren

Geräten der Ein- und Ausgabe in Verbindung steht. Abb. 2.2 stellt die mögliche Grobstruktur eines Digitalrechners dar, die im konkreten Fall unter Umständen zu modifizieren ist und einer weiteren Differenzierung der Verbindungen zwischen den Gerätesystemen bedarf.

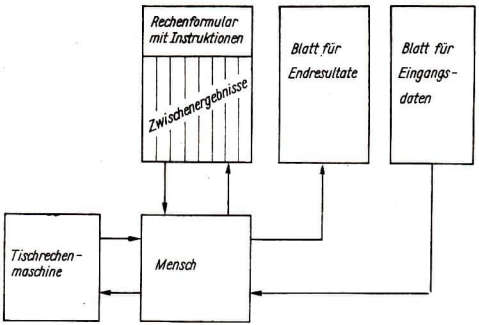


Abb. 2.1

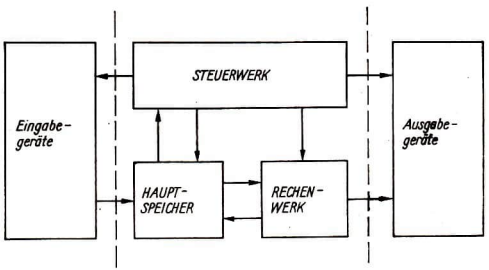


Abb. 2.2

Der Hauptspeicher ist in Zellen eingeteilt, die mit Nummern, sogenannten *Adressen*, versehen sind. Sowohl das Programm als auch die zu verarbeitenden Daten (vgl. 2.3.1.) sind dort zu speichern, und zwar als Wörter über einem sogenannten *internen Alphabet*. Dieses umfaßt z. B. bei der EDVA R 300 die 64 Zeichen der Tabelle 2.1 (S. 52). Die technischen Charakteristika des Hauptspeichers sind seine *Kapazität*, d. h. die Anzahl der speicherbaren Zeichen, und die *Zugriffszeit*,

d. h. die (gemittelte) Zeit für das Aufsuchen eines Speicherplatzes und die Datenentnahme. Zum Vergleich geben wir die Werte für die EDVA ZRA1, R 300 und R 21 an:

	Anzahl der adressierbaren Speicherplätze	Zugriffszeit
ZRA 1	2^{12}	2,5 ms ($1\ \text{ms} = 10^{-3}$ s; Millisekunde)
R 300	40000	5 µs ($1\ \mu\text{s} = 10^{-6}$ s; Mikrosekunde)
R 21	2^{16}	520 ns ($1\ \text{ns} = 10^{-9}$ s; Nanosekunde)

Die Hochleistungsrechner der ESER-Serie (Einheitliches System der Elektronischen Rechentechnik) verfügen über eine Kapazität von 2^{20} adressierbaren Plätzen im Hauptspeicher (vgl. [39]).

Vom Automaten werden bei der Verarbeitung einer Information nach einem Algorithmus die Operationen ausgeführt, welche durch die Befehle des in den Hauptspeicher übernommenen Programms vorgeschrieben sind. Ein Befehl ist eine bestimmte Instruktion, die durch ein Wort über dem internen Maschinenalphabet ausgedrückt wird. Beim R 300 umfaßt dieses sechs Zeichen. Die Lokalisierung des Befehls im Hauptspeicher erfolgt durch eine Adresse, die sogenannte *Steueradresse* oder *äußere Adresse*. In der Ordnung der Numerierung ist die Befehlsfolge eines Programms in aufeinanderfolgenden Speicherplätzen abgelegt. Ein R 300-Befehl wird durch die Adresse des ersten Zeichens des Befehlswortes erfaßt. Die normale Befehlsfolge erfordert also eine sukzessive Adressenerhöhung um 6. Die verfügbaren Befehle und ihre Darstellung als Wörter über dem internen Alphabet sind in der *Befehlsliste* des Automaten festgelegt. Jeder Befehl setzt sich aus einem *Operations-* und einem *Adressenteil* zusammen (Abb. 2.3). Man darf den Inhalt des Adressenteils nicht mit der äußeren Adresse (Steueradresse) des Befehls verwechseln und spricht hier deshalb auch von *inneren Adressen*.

Art und Struktur der Befehle sind vom Maschinentyp abhängig, der die Anzahl der im Adressenteil auftretenden Operandenadressen bestimmt. Bei einer *n-Adreß-*

OPERATIONSTEIL	*ADRESSENTEIL*
enthält in verschlüsselter Form die auszuführenden Operationen	*enthält in verschlüsselter Form die Adresse(n) der beteiligten Größe(n)*

Abb. 2.3

maschine enthält kein Befehl mehr als n, mindestens einer jedoch genau n Adressen. Um die Wirkung eines solchen Befehls zu erklären, sind einige Bemerkungen zum Aufbau des Rechenwerks nötig. Dieses ist – wie in 2.2. begründet wird – bezüglich seiner arithmetischen Funktionen wesentlich ein Addierwerk. Darüber hinaus können gewisse logische Operationen installiert sein. Möglichkeiten der technischen Realisierung solcher Systeme werden in 2.4. erörtert. Zur Aufnahme der zu verknüpfenden Operanden verfügt das Rechenwerk über Ein-Wort-Speicher[1]) extrem kurzer Zugriffszeit, sogenannte *Register*. Eines derselben wird vielfach zugleich für die Aufnahme des Resultats verwendet und dann als *Akkumulator* bezeichnet. Das Blockschaltbild eines solchen Rechenwerks nach [25] ist in Abb. 2.4 wiedergegeben.

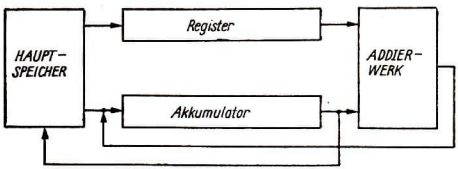


Abb. 2.4

Von der Funktion her lassen sich die Befehle in drei Hauptgruppen einteilen:

1. *Transportbefehle* für Datenübertragung in der Zentraleinheit und zwischen Hauptspeicher und peripheren Geräten.
2. *Arithmetische* und *logische Befehle* für die Verknüpfung von Registerinhalten im Rechenwerk. In einer EDVA mit festverdrahteten logischen Operationen können über das Rechnen hinaus allgemein Umwandlungen codierter Informationen erfolgen.
3. *Sprungbefehle* zur Realisierung von Programmverzweigungen, die unbedingt oder in Abhängigkeit vom Wahrheitswert von Entscheidungsaussagen auftreten können. Beispielsweise erfordert die Abarbeitung des Euklidischen Algorithmus (vgl. den PAP der Abb. 1.4) einen bedingten Sprung in Verbindung mit der Entscheidung $r = 0$ und den unbedingten Rücksprung zu einer vorangegangenen Anweisung nach den Wertzuweisungen

$$x := y \quad \text{und} \quad y := r.$$

Wir wollen nun kurz auf die sich aus der Struktur des Adressenteils ergebenden Konsequenzen für die Programmierung arithmetischer Operationen eingehen. Bei einer Dreiadreßmaschine würde die Addition zweier Zahlen und die Speiche-

[1]) Damit soll gesagt werden, daß darin keine Teilbereiche adressierbar sind.

rung des Resultates in einem Befehl zu erfassen sein:

Add	x, y, z
Operationsteil	Adressenteil

Darin steht Add für die laut Befehlsliste zu verschlüsselnde Addition; x, y für die Adressen der Operanden und z für die Adresse des Resultats, das auf Grund der Hardware[1])-Gegebenheiten auf diesen Platz des Hauptspeichers gelangt. In einer Zweiadreßmaschine ist vom Vollzug der gleichen Anweisung ein Transportbefehl erforderlich. Diese könnte so konstruiert sein, daß

Add	x, y

die Bildung der Summe der Inhalte von x und y im Akkumulator des Rechenwerks veranlaßt. Ein weiterer Befehl, etwa

TRANS	z

wäre nötig, um dieses von dort nach dem mit z bezeichneten Speicherplatz zu bringen.

Die Ausführung einer arithmetischen Operation in einer Einadreßmaschine wie dem R 300 erfordert, daß sich ein Operand vor ihrer Einleitung bereits in einem Register des Rechenwerks befindet. Im Prinzip braucht man also drei Befehle, um die betrachtete Addition auszuführen:

- einen Transportbefehl, um den Inhalt der Speicherzelle mit der symbolischen Adresse x (man schreibt dafür $\langle x \rangle$) in ein Register des Rechenwerks zu bringen;
- einen Befehl, der die Addition von $\langle y \rangle$ zu $\langle x \rangle$ im Rechenwerk veranlaßt, wozu $\langle y \rangle$ in ein zweites Register (den Akkumulator) gebracht wird;
- einen Transportbefehl, der das im Akkumulator stehende Resultat nach der Speicherzelle mit der Adresse z bringt.

Bei einer Vieradreßmaschine ist die vierte Adresse die des nächsten Befehls, während die ersten drei wie bei einer Dreiadreßmaschine zu interpretieren sind.

Die vierte und fünfte Adresse einer Fünfadreßmaschine beziehen sich ebenfalls auf die Speicherplatznummern des nächsten Befehls. Bei einer Programmverzweigung ist dafür die vierte maßgebend, wenn die Sprungbedingung erfüllt ist, sonst die fünfte. In allen anderen Fällen steht auf Platz 4 und 5 die Steueradresse des nächsten Befehls.

Aufgabe des Steuerwerks ist es, die im Hauptspeicher deponierten Befehle eines Programms in der richtigen Reihenfolge aufzurufen, zu entschlüsseln und den

[1]) Hardware, wörtlich harte Ware, meint in der Terminologie der Rechentechnik etwas zur Maschinenausrüstung Gehöriges; hardwaremäßig bedeutet bei einer EDVA meist so viel wie „fest verdrahtet".

Ablauf der entsprechenden Operation zu kontrollieren. Wir wollen kurz die Arbeitsweise des Steuerwerks einer Einadreßmaschine erörtern, dessen wesentliche Bestandteile ein *Befehlszähler* und ein *Befehlsregister* sind. Im Befehlszähler werden die Adressen der sukzessive aufzurufenden Befehle gebildet. Das Befehlsregister nimmt den jeweils auszuführenden Befehl auf, und zwar dessen Operations- und Adressenteil in einem Operations- bzw. Adreßregister. Im Operationsregister wird der Befehl entschlüsselt und die für seine Abarbeitung notwendige Schaltung veranlaßt. Speziell wird festgestellt, ob ein Sprungbefehl vorliegt, und dann dessen innere Adresse, welche die Steueradresse des nächsten Befehls angibt, zur Neueinstellung des Befehlszählers benutzt. In jedem anderen Fall erfolgt bei diesem ein „normales" Weiterrücken, was bei der EDVA R 300 eine Erhöhung der Steueradresse um 6 bedeutet. Die Information im Adreßregister wird bei einem Nicht-Sprungbefehl zur Steuerung der Operandenauswahl benutzt; die detaillierte Darstellung dieses Vorgangs würde ein genaueres Eingehen auf die Maschinenprogrammierung erfordern. Der Zyklus der Befehlsabarbeitung in einer Einadreßmaschine könnte dem Flußbild in Abb. 2.5 entsprechen.

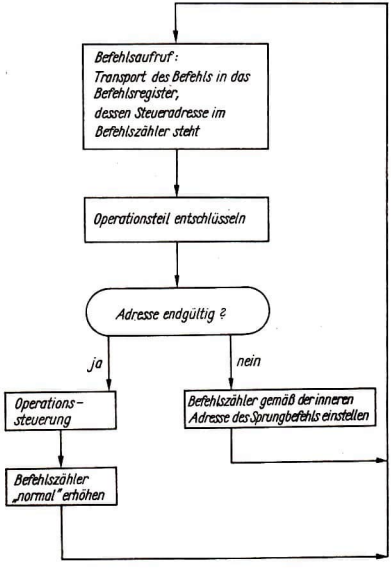


Abb. 2.5

Die Verbindung zwischen dem Automaten und seiner Außenwelt wird durch die *peripheren Geräte der Ein- und Ausgabe* hergestellt. Deren Aufbau und Wirkungsweise hängt von der Beschaffenheit der Informationsträger ab, die nach der beabsichtigten Verwendung des Datenmaterials ausgewählt werden. Als externe Informationsträger kommen Lochkarten, Lochbänder, magnetisierbare Schichten (zum Beispiel Magnetbänder) und Klartextbelege in Betracht. Die Übertragung der Daten wird eingabeseitig entsprechend durch Lochkarten-, Lochband- und Magnetbandleser vermittelt. Die Forschungen auf dem Gebiet der automatischen Zeichenerkennung zielen u. a. auf die Entwicklung von Geräten, die manuell angefertigte Urbelege auf Grund wahrgenommener Kontraste in codierte Zeichenfolgen eines Maschinenalphabets umwandeln. Es befinden sich bereits leistungsfähige Magnetschrift-Leser im Einsatz, die mit ferrithaltiger Druckfarbe materialisierte Symbole erkennen und maschinengerecht verschlüsseln. Die Ausgabe kann über Magnetband sowie vermittels Lochkarten- und Lochbandstanzer erfolgen. Klartextbelege erhält man über sogenannte Schnelldrucker, die Informationen zeilenweise simultan auf randgelochte Papierbahnen drucken, welche in Faltstapeln eingelegt und ausgegeben werden. Der Schnelldrucker des R 300 erreicht eine Geschwindigkeit von 360 Zeilen je 156 Zeichen pro Minute.

Diese Aufzählung von Geräten der Ein- und Ausgabe beschränkt sich auf die zur Zeit am häufigsten benutzten. Als Beispiel für eine speziellere Ausrüstung seien etwa Kartiergeräte – sogenannte Plotter – genannt, die es u. a. ermöglichen, die Graphen berechneter Funktionsverläufe automatisch zu zeichnen.

Die vorangegangenen Betrachtungen wollten einen Überblick über die Grobstruktur und das Zusammenwirken der Funktionsgruppen eines Digitalrechners vermitteln. Bewußt wurde dabei auf die Beschreibung technischer Einzelheiten verzichtet. Einer Vertiefung der Einsichten in prinzipielle, d. h. mathematisierbare Probleme der Erfassung und Verarbeitung von Informationen in Digitalrechnern einschließlich der Präzisierung damit zusammenhängender Begriffe sind die folgenden Abschnitte dieses Kapitels gewidmet. Zur Abrundung wird diese Einleitung noch durch eine Darstellung der historischen Entwicklung der Rechenhilfsmittel ergänzt.

2.1.2. Der Prozeß der begrifflichen Analyse endlicher Mengen hat vermutlich in der älteren Steinzeit eingesetzt und dürfte – bedingt durch die Entwicklung und Differenzierung der Produktivkräfte und Produktionsverhältnisse – in der jüngeren Steinzeit einen gewissen Höhepunkt mit der Herausarbeitung von Zahlbegriffen erreicht haben (vgl. MfL Bd. 2, S. 146–153). Fast gleichzeitig wurden räumliche Beziehungen erfaßt und geometrische Formen von figürlichen Erscheinungen abstrahiert. Die Ornamentik jener Epoche liefert dafür reiches Belegmaterial. Mit der Herausbildung arithmetischer und geometrischer Grundvorstellungen wurden Hilfsmittel zur Lösung von Aufgaben der Praxis entwickelt, auf welche sich diese Keimelemente einer mathematischen Theorie anwenden ließen. Für arithmetische Operationen waren Finger und Zehen die ersten Rechenhilfsmittel. Das ist durch

den Umstand belegt, daß die meisten Zahlsysteme der Urgesellschaft die 5, 10 oder 20 zur Grundzahl hatten. Geometrische Aufgaben der Vermessung und Reproduktion von Regelmäßigkeiten wurden offenbar durch das Hantieren mit Seilen gelöst. Man erkennt in solchen Hilfsmitteln frühe Formen digitaler und analoger Instrumentarien.

Die Durchführung von Additionen nach einem schematischen Verfahren wurde möglich, als man die Finger durch geeignete Gegenstände ersetzte, die auf eingeschnittenen Linien eines *Rechenbretts* verschoben werden konnten. Die Römer nannten dieses *Abakus* und benutzten von ihnen als calculi bezeichnete kleine Kalksteine, von denen sich der Name Kalkül ableitet. Die materiale Beschaffenheit der Rechenhilfsmittel findet sich auch in einigen anderen mathematischen Begriffsbezeichnungen wieder.

Das Prinzip der Addition auf dem Abakus zeigt Abb. 2.6. Die eingeschnittenen Linien kann man im unteren Teil des Rechenbretts den Zahlwerten I, X, C, M, . . ., im oberen Teil den Halbwerten V, L, D, . . . zuordnen. Die Konfiguration aus Abb. 2.6a ist als

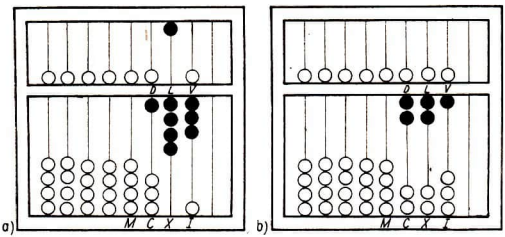


Abb. 2.6

Darstellung von 193 zu interpretieren. Wir wollen dazu etwa 28 addieren. Man beginnt mit dem höchsten Stellenwert, versucht also im Beispiel zwei Zehnersteine nach oben zu schieben. Da dies nicht möglich ist, erfolgt ein Übertrag, d. h., ein Hunderter-Stein wird verschoben, die X-Spalte in die Grundstellung gebracht und der verbleibende Zehner des Addenden dort durch Verschieben eines Steines eingestellt. Das Hinzufügen der 8 erfordert nun einen Übertrag in der X-Spalte und die Darstellung des verbleibenden Einers. Abb. 2.6b veranschaulicht das Ergebnis CCXXI =221.

Bei der Subtraktion werden die Rechensteine der Darstellung des Minuenden entsprechend den im Subtrahenden auftretenden Einern, Zehnern usw. nach unten verschoben, wobei wieder an der höchsten Stelle zu beginnen ist. Reichen dabei an einer Position die verfügbaren Steine nicht aus, so erfolgt der Abzug einer Einheit in der nächst höheren Position, und in der betrachteten werden soviel Steine nach oben gerückt, wie Einheiten im Subtrahenden bis 10 fehlen.

Multiplikation und Division lassen sich auf eine wiederholte Addition bzw. Subtraktion zurückführen.

Rechenbretter nach dem Prinzip des Abakus waren und sind in vielen Kulturvölkern verbreitet. Mit je zehn auf Drähten aufgezogenen Kugeln sind sie als sogenannte

Kinderrechenmaschinen bekannt. Bezüglich der Verwendung des russischen Rechenbretts für alle vier Grundrechenarten vergleiche man etwa [60].

Die Funktion des Steuerwerks für das Rechenwerk Abakus liegt mit zahlreichen bedienenden Eingriffen beim Menschen. Vom jeweiligen Training im Kopfrechnen bzw. im Umgang mit einem Rechenbrett wird es abhängen, ob eine Steigerung der Rechengeschwindigkeit durch Verzicht auf dieses Hilfsmittel möglich ist. Der Sieg des „Rechnens auf der Feder" über das „Rechnen auf den Linien" der Abakisten in Europa wird in einer Illustration aus dem Jahre 1504 dargestellt (vgl. MfL Bd. 2, S. 155).

Mehr als 2000 Jahre mußten vergehen, bis eine *Automatisierung des Zehnerübertrags* gelang und die Weiterentwicklung des Rechenbretts zur mechanischen Tischrechenmaschine möglich wurde. Dieser Vorgang zeigt, daß die Entwicklung von Datenverarbeitungsgeräten nicht nur von den aus dem gesellschaftlichen Status resultierenden Bedürfnissen, sondern wesentlich auch vom Umfang der naturwissenschaftlichen und technischen Kenntnisse und Fähigkeiten abhängt. Die erste mechanische Rechenmaschine mit automatischem Zehnerübertrag scheint auf Anregung Keplers von dem Tübinger Professor für Mathematik und Astronomie Wilhelm Schickart in Zusammenarbeit mit dem Mechaniker Pfister 1624 gebaut worden zu sein. Nachweisbar funktionstüchtig war die 1641 von dem französischen Mathematiker und Philosophen Blaise Pascal konstruierte Maschine, mit der man Additionen und Subtraktionen ausführen konnte. Ein Exemplar derselben wird im Mathematisch-Physikalischen Salon im Dresdner Zwinger aufbewahrt. G. W. Leibniz stellte 1672 in London einen noch mit technischen Mängeln behafteten von ihm entworfenen Vier-Spezies-Rechner vor. Die weitere Entwicklung ist vor allem durch die Erfindung neuer mechanischer Schaltelemente bestimmt. Die fabrikmäßige Herstellung handgetriebener mechanischer Rechenmaschinen setzte um 1820 in Frankreich ein. Durch den Einbau eines elektrischen Antriebs wurde später das menschliche Eingreifen auf das Eintasten von Operanden, Auslösen der Operation durch Tastendruck und Ablesen der Resultate reduziert. Die zunehmende Verbreitung solcher elektromechanischer Maschinen beeinflußte schon merklich die Methoden des praktischen Rechnens: Die Verwendung von Logarithmen geht zurück, und Tafelwerke entstehen, die der neuen Rechentechnik angepaßt sind.

Parallel zu dieser Entwicklung wurde durch Triebkräfte der beginnenden Industrialisierung im technologischen Bereich die programmgesteuerte Abarbeitung von Algorithmen vorbereitet und eingeleitet. Im Jahre 1805 erfand der Franzose J. M. Jacquard den nach ihm benannten automatischen Webstuhl, der Muster erzeugte, die in gelochten Stahlplatten vorprogrammiert waren. Die Übertragung solcher Ideen auf die Bearbeitung numerischer Probleme wurde am Ende des 19. Jahrhunderts durch die Notwendigkeit stimuliert, umfängliches, aus demoskopischen Erhebungen stammendes Datenmaterial zur Gewinnung von Planungsunterlagen statistisch auszuwerten. Nach 1880 machte der Amerikaner Herrmann Hollerith die grundlegenden Erfindungen der Lochkartentechnik; die von ihm

konstruierte Zähl- und Sortiermaschine wurde 1890 bei der 11. amerikanischen Volkszählung eingesetzt und ermöglichte, die Dauer der Auswertung auf ein Sechstel der bislang benötigten Zeit zu reduzieren. Bereits um 1820 versuchte der Cambridger Mathematikprofessor CHARLES BABBAGE – angeregt durch die Jacquardsche Erfindung – einen Vier-Spezies-Rechner mit einer Programmsteuerung auf Lochkartenbasis zu bauen. Er scheiterte an den technischen Unzulänglichkeiten seiner Zeit; die Konzeption berücksichtigte jedoch schon die Prinzipien wesentlicher Funktionsgruppen moderner Digitalrechner. Der Einsatz von Lochkartenmaschinen ist wegen der Begrenztheit der ausführbaren Operationen zurückgegangen; sie haben heute noch eine eingeschränkte Bedeutung in Bereichen der Statistik und des Rechnungswesens.

Die Entwicklung von universellen Digitalrechnern, wie sie in 2.1.1. beschrieben wurden, setzte während des zweiten Weltkrieges etwa gleichzeitig in Deutschland und den USA ein. Pionierarbeit leisteten KONRAD ZUSE mit dem 1941 vorgestellten Automaten Z3 und HOWARD H. AIKEN mit dem für die Harvard University gebauten MARK I. Beide Rechner arbeiteten elektromechanisch, d. h., ihre Schaltelemente waren Relais. In die Klasse dieser Rechner ist auch die 1953/54 von H. KORTUM und W. KÄMMERER entwickelte und im VEB Carl Zeiss Jena gebaute OPREMA einzuordnen. Die Trägheit der mechanisch arbeitenden Teile solcher Anlagen setzt der Rechengeschwindigkeit natürliche Grenzen; die Zeit für die Umschaltung eines Relais liegt im Bereich einer Hundertstelsekunde.

Die Schaltzeiten elektronischer Bauelemente sind um Größenordnungen kleiner; sie liegen im Bereich von Mikro- und Nanosekunden. Mit der Konstruktion des ersten Elektronenrechners – des 1946 an der Universität von Pennsylvania fertiggestellten ENIAC – wurde daher ein neuer Abschnitt in der Entwicklung universeller Digitalrechner eingeleitet. Der ENIAC war mit 18000 Elektronenröhren ausgestattet und in seinen Ausmaßen entsprechend groß. Die Elektronenröhren als Schaltelemente benutzenden Automaten werden *Rechner der ersten Generation* genannt; bei ihnen liegt die für eine Operation benötigte Zeit im Bereich einer Millisekunde. Alle bis Anfang der fünfziger Jahre gebauten Anlagen waren Einzelanfertigungen und wurden meist von Arbeitsgruppen an Universitäten entwickelt. Die Serienanfertigung von EDV-Anlagen wurde zuerst von der amerikanischen Firma IBM aufgenommen. Der wirtschaftliche Erfolg ihrer ersten Anlage IBM 650 gründete sich wesentlich auf eine gute Einsatzvorbereitung der von dem Unternehmen anfangs vermieteten Anlagen. Mit diesen wurden zugleich Wartungsteams, Bedienungskräfte, Programmierer und Mathematiker für die Problemanalyse angeboten. Auch unter den Bedingungen der sozialistischen Wirtschaftsführung ist die erfolgreiche Gestaltung dieser komplexen Aufgabe Voraussetzung für den aus der EDV zu ziehenden Nutzen. Da die EDV in stürmischer Entwicklung immer größere Bereiche unseres Lebens erfaßt, muß die Behandlung ihrer Bedeutung für die Praxis ein wichtiges Bildungsanliegen der polytechnischen Schule sein.

Die *Rechner der zweiten Generation* – Vertreter dieser Klasse sind die sowjetische Anlage BESM 6 und die in der DDR entwickelte und gebaute EDVA R 300 –

haben Transistoren und Dioden als Schaltelemente. Das wirkt sich in einer Miniaturisierung der Funktionsgruppen, Reduzierung des Energiebedarfs und Vereinfachung der Kühlprobleme aus. Die für eine Operation erforderliche Zeit liegt im Mikrosekundenbereich. Die gegenüber den Anlagen der ersten Generation bedeutend gesteigerte Rechengeschwindigkeit ist wesentlich auch durch die Beschaffenheit des Arbeitsspeichers bestimmt. Während jene einen rotierenden *Magnettrommelspeicher* besitzen, sind die Rechner der zweiten Generation mit einem *Ferritkernspeicher* ausgestattet. Zum Vergleich: Der Trommelspeicher des bis in die sechziger Jahre vom VEB Carl Zeiss Jena gefertigten Röhrenrechners ZRA 1 hatte eine mittlere Zugriffszeit von 2,5 ms; die Zugriffszeit des Ferritkernspeichers des R 300 beträgt 5 µs.

Die *Rechner der dritten Generation* verwenden Festkörperschaltkreise, die auf Plättchen von wenigen Quadratmillimetern die Funktionen mehrerer Transistoren vereinigen. Mit dieser Technik läßt sich eine kaum noch vorstellbare Miniaturisierung erreichen. Die Operationen laufen in der Größenordnung von Nanosekunden ab. Durch zusätzliche Verwendung sogenannter *Dünnschichtspeicher* wird die Zugriffszeit weiter verringert. Rechner der dritten Generation sind die Anlagen der ESER-Serie und die in der DDR gefertigte EDVA R 21. Die Zugriffszeit dieses Automaten liegt bei 520 ns.

Abschließend sei darauf hingewiesen, daß die Verwendung der modernen elektronischen Bauelemente zu den Operationsgeschwindigkeiten führte, die eine neue Qualität der Rechentechnik bewirkten, daß aber das kybernetische Prinzip eines universellen Digitalrechners von der Elektronik durchaus unabhängig ist.

2.2. Zahldarstellung in Positionssystemen

Die natürlichen Zahlen werden genetisch als Kardinalzahlen endlicher Mengen eingeführt (vgl. MfL Bd. 1, 3.1.). Als solche sind sie in der Praxis nicht zu handhaben. Arithmetische Operationen können automatisch nur nach einem Algorithmus ausgeführt werden, und das erfordert die symbolische Verschlüsselung der Operanden, nämlich ihre Darstellung durch ein aus Zeichen zusammengesetztes Wort (vgl. 1.3.). Zahldarstellungen gewinnt man mit Hilfe eines *Positionssystems* zu einer *Basis* g ($g \in \mathbb{N}$, $g \geq 2$); die benutzten Symbole heißen *Ziffern* und bezeichnen die natürlichen Zahlen, die kleiner als g sind. Für das *Dezimalsystem* sind das

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.

Im *Dual-* oder *Binärsystem* ($g = 2$) verwendet man im Bereich der Rechentechnik meist

0 | L. (1)

Offenbar sind in jedem *g-adischen* Positionssystem die natürlichen Zahlen 0 und 1 zu beziffern; mit Ausnahme des Dualsystems, wo gemäß (1) verfahren wird, verwenden wir dafür **0 | 1.** Die Ausführungen dieses Abschnitts sind auf MfL Bd. 1, 3.8.,

bezogen und beachten die dort eingeführte Unterscheidung zwischen Zahlen und Zahlzeichen, die im weiteren allerdings aus Gründen der Typographie nur noch begrifflich aufrechterhalten werden kann. $\boldsymbol{a}$ bedeutet in einem g-adischen Positionssystem die der natürlichen Zahl a, $0 \leqq a < g$, entsprechende Ziffer.

Bei gegebener Basis betrachten wir die Menge W aller Wörter

$$\mathfrak{w} := \boldsymbol{w}_k \boldsymbol{w}_{k-1} \ldots \boldsymbol{w}_0, \qquad \boldsymbol{w}_k \neq \boldsymbol{0},$$

über dem Alphabet der Ziffern und die Abbildung

$$f\colon \boldsymbol{w}_k \boldsymbol{w}_{k-1} \ldots \boldsymbol{w}_0 \mapsto w_k \cdot g^k + w_{k-1} \cdot g^{k-1} + \cdots + w_1 \cdot g + w_0 \tag{2}$$

von W in N^*. Der Satz zu MfL Bd. 1, 3.8.(3), besagt:

$$f\colon W \to \mathsf{N}^* \textit{ ist eine eineindeutige Abbildung von } W \textit{ auf } \mathsf{N}^*. \tag{3}$$

Die einer natürlichen Zahl n entsprechende Zifferndarstellung $\mathfrak{w}$ kann in einer gewissen Analogie zum Euklidischen Algorithmus durch eine Folge von Divisionen mit Rest gefunden werden, deren Divisor die Basis g ist. Diese beginnt mit

$$n = q_1 \cdot g + w_0, \qquad 0 \leqq w_0 \leqq g-1.$$

Im folgenden bildet man

$$q_1 = q_2 \cdot g + w_1, \qquad 0 \leqq w_1 \leqq g-1,$$
$$q_2 = q_3 \cdot g + w_2, \qquad 0 \leqq w_2 \leqq g-1,$$

usw. Wegen $n > q_1 > q_2 > q_3 > \cdots$ ergibt sich nach endlich vielen Schritten zum ersten Mal eine Gleichung der Form

$$q_k = 0 \cdot g + w_k \quad \text{mit} \quad 0 \leqq w_k \leqq g-1,$$

bei welcher das Verfahren abgebrochen wird. Offenbar ist $w_k \neq 0$, und durch sukzessives Eliminieren von $q_1, q_2, \ldots, q_{k-1}$ ergibt sich

$$n = w_k \cdot g^k + w_{k-1} \cdot g^{k-1} + \cdots + w_0, \tag{4}$$

also in der Schreibweise von MfL Bd. 1, 3.8.,

$$n \triangleq_g \boldsymbol{w}_k \boldsymbol{w}_{k-1} \ldots \boldsymbol{w}_1 \boldsymbol{w}_0.$$

Der Algorithmus zur Konstruktion dieser Darstellung läßt sich durch das Flußbild in Abb. 2.7 beschreiben. Dabei wird angenommen, daß die Ausgabe der Ziffern in linearer Ordnung von links nach rechts erfolgt. $\mathfrak{w} \Leftarrow \tilde{\mathfrak{w}}$ bedeutet den Übergang zum gespiegelten Wort:

$$\tilde{\mathfrak{w}} := \boldsymbol{w}_0 \boldsymbol{w}_1 \ldots \boldsymbol{w}_{k-1} \boldsymbol{w}_k,$$
$$\mathfrak{w} := \boldsymbol{w}_k \boldsymbol{w}_{k-1} \ldots \boldsymbol{w}_1 \boldsymbol{w}_0.$$

Das Begreifen einer Zahl als Abstraktum und ihre Darstellung in einem Positionssystem war das Ergebnis einer langen Entwicklung.[1]) Das älteste uns bekannte

[1]) Für ein eingehenderes Studium sei auf den Artikel von I. G. Baschmakowa und A. P. Juschkewitsch in [9] und den Anhang von H. Wussing in MfL Bd. 2 verwiesen.

Positionssystem ist das um 2000 v. u. Z. aufkommende *Sexagesimalsystem* ($g=60$) der Babylonier, anfangs insofern unvollständig, als es noch kein Zeichen für die Null enthielt. Die übrigen Ziffern wurden als Wortsymbole[1]) aus den Keilschriftzeichen ▼ für 1 und ≺ für 10 gebildet; die Struktur der Wortsymbole war – wie auch bei der ägyptischen Hieroglyphenschreibweise – streng additiv, wobei links mit den höchsten Einheiten begonnen wurde. Der Zahl 23 zum Beispiel entsprach als Ziffer die Zeichenfolge ≺≺▼▼▼.

Das heute allgemein benutzte Dezimalsystem entstand in Indien und gelangte im 11. Jahrhundert über das maurische Spanien nach Mitteleuropa.

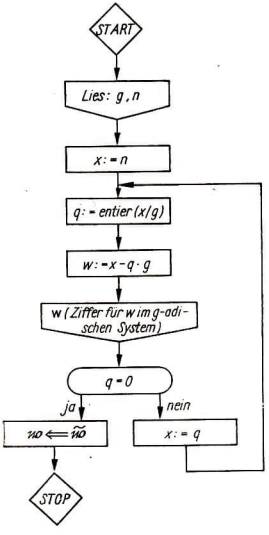


Abb. 2.7

Mit dem Algorithmus des PAP von Abb. 2.7 hat man prinzipiell die Möglichkeit, die Darstellung einer Zahl in einem Positionssystem in die in einem anderen umzuwandeln. Der Übergang vom Dezimalsystem in ein anderes g-adisches System wird

[1]) Man vergleiche dieses Vorgehen mit der Symbolbildung in den Alphabeten von Programmiersprachen (Kapitel 3).

als *Konvertierung*[1]), der umgekehrte Prozeß als *Rekonvertierung* bezeichnet. Als Beispiel betrachten wir die Umwandlung der Dezimalzahl[2]) **3875** in die oktale Darstellung; Ziffern des *Oktalsystems* ($g=8$) seien die entsprechenden des Dezimalsystems. Als Protokoll erhält man

g	n	x	q	w	Darstellung im Oktalsystem w
8	3875	3875	484	3	**3447**
		484	60	4	**7443**
		60	7	4	
		7	0	7	

Die Abarbeitung des Algorithmus gestaltet sich schwieriger, wenn man nicht von einer Dezimalzahl ausgeht, weil das „Einmaleins" in dem benutzten Positionssystem nicht genügend beherrscht wird. Es empfiehlt sich, hier erst zur Dezimaldarstellung überzugehen und diese in die Darstellung des gewünschten Positionssystems zu übertragen. Die Rekonvertierung erfordert gemäß (4) die Auswertung eines Polynoms, dessen Koeffizienten die Zahlwerte der Ziffern der betrachteten positionellen Darstellung sind. Dazu bedient man sich zweckmäßigerweise eines als *Hornersches Schema* bezeichneten Algorithmus, der schon von dem chinesischen Mathematiker CHHIN CHIU-SHAO um 1250 und in der ersten Hälfte des 15. Jahrhunderts von dem Perser AL-KAŠI zur numerischen Lösung algebraischer Gleichungen benutzt wurde. W. G. HORNER (1786–1837) publizierte das Verfahren 1819 (vgl. [68]); da es von elementarer Bedeutung für die numerische Mathematik ist, gehen wir an dieser Stelle genauer darauf ein.

Vorgelegt sei die ganze rationale Funktion P, deren Wert

$$P(x)=a_n x^n+a_{n-1}x^{n-1}+\cdots+a_1 x+a_0 \tag{5}$$

für ein reelles Argument $x=x_0$ zu berechnen ist. Auch die Koeffizienten a_ν, $\nu=0,1,\ldots,n$, sollen reell sein, und ferner sei $a_n\neq 0$. Rekursiv werden zu P Polynome $P_0, P_1, \ldots, P_{n-1}, P_n$ gemäß

$$\begin{array}{l|l} P_0(x)=a_n & x^n \\ P_1(x)=a_{n-1}+xP_0(x) & x^{n-1} \\ \quad\ldots\ldots\ldots & \quad\ldots\ldots \\ P_{n-1}(x)=a_1+xP_{n-2}(x) & x \\ P_n(x)\ \ =a_0+xP_{n-1}(x) & 1 \end{array} \tag{6}$$

[1]) Gelegentlich wird auch der Übergang von einer positionellen Darstellung zu einer anderen als Konvertierung schlechthin bezeichnet.

[2]) Hierbei handelt es sich um eine verkürzte Ausdrucksweise. Korrekt müßte man von der Zahl 3875 in ihrer Darstellung im Dezimalsystem und deren Umwandlung in die Darstellung im Oktalsystem reden. 3875 wird also hier als ein Zeichen aufgefaßt, dessen Zuordnung zu dem Bezeichneten als bekannt angenommen wird.

konstruiert. Dann gilt für alle x

$$P_n(x) = P(x). \tag{7}$$

Zum Beweis multipliziere man, wie in (6) angegeben, die erste Gleichung mit x^n, die zweite mit x^{n-1} usw. und addiere. Dann heben sich der Term der linken Seite der i-ten Gleichung ($i = 0, 1, \ldots, n-1$) und der zweite Summand der rechten Seite der $(i+1)$-ten Gleichung auf, und es bleibt

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0.$$

Die Bestimmung von $P_n(x)$ aber kann nach (6) schematisch so erfolgen: Man notiert die Koeffizienten von (5) (auch die verschwindenden), beginnend mit a_n, in einer Zeile, multipliziert a_n (das aus formalen Gründen auch mit a'_n bezeichnet wird) mit dem Argumentwert x_0 und addiert zu a_{n-1}. Das Resultat wird mit x_0 multipliziert, zu a_{n-2} addiert und liefert a'_{n-2}. So fortfahrend erhält man am Schluß der Rechnung $P(x_0)$, da für die Zwischenresultate gemäß (6) und (7)

$$\begin{aligned} a'_n &= a_n = P_0(x_0), \\ a'_{n-1} &= P_1(x_0), \\ &\cdots\cdots \\ a'_0 &= P_n(x_0) \end{aligned}$$

gilt. Für die Rechnung von Hand empfiehlt sich folgende Anordnung:

$$\begin{array}{ccccccc} a_n & a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 & \\ & a'_n x_0 & a'_{n-1} x_0 & \cdots & a'_2 x_0 & a'_1 x_0 & \\ \hline a'_n & a'_{n-1} & a'_{n-2} & \cdots & a'_1 & a'_0 = P(x_0) & \end{array} \tag{8}$$

Die Bestimmung von $P(x_0)$ gemäß (8) hat gegenüber dem Vorgehen, zunächst die erforderlichen Potenzen von x_0 zu bilden und diese mit den Polynomkoeffizienten linear zu kombinieren, u. a. den Vorteil, daß wiederholt ein und derselbe Rechenschritt

$$a'_{n-i} := a_{n-i} + a'_{n-i+1} x_0, \quad i = 1(1)n,^{1})$$

auszuführen ist. Von den Größen a'_i braucht man nur die für den jeweils nächsten Rechenschritt erforderliche aufzubewahren. Das legt nahe, eine Variable p einzuführen und diese mit dem jeweils aktuellen a'_i zu belegen (a'_i auf dem p zugeordneten Speicherplatz zu deponieren). Man vergleiche diese Bemerkung mit der entsprechenden bezüglich des Euklidischen Algorithmus in 1.3. Das Verfahren (8) kann dann in dem PAP der Abb. 2.8 zusammengefaßt werden:

Am Anfang steht die Eingabe des Polynomgrades, der Koeffizienten und der Argumentstelle x_0. Sodann wird der Variablen p der Anfangswert a_n zugewiesen.

---

[1]) Damit ist gemeint: i läuft mit der Schrittgröße 1 von 1 bis n.

Während beim Euklidischen Algorithmus in 1.3. der Abbruch des Verfahrens, also das Herausführen aus dem Zyklus, durch den Schritt für Schritt zu testenden Wert von r veranlaßt wird, ist im vorliegenden Fall die Anzahl der Zyklen von

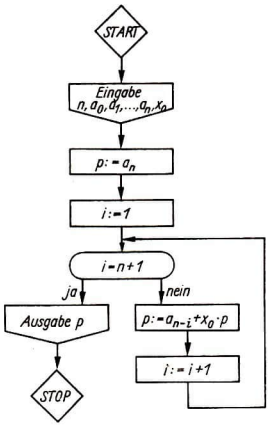


Abb. 2.8

vornherein durch den Polynomgrad n bestimmt. Man baut daher einen Zähler in das Programm ein, der mit Hilfe der Variablen i organisiert wird. Zu Beginn erhält diese den Wert 1 zugewiesen. Danach wird geprüft, ob i den Wert $n+1$ erreicht hat. Das ist (im Fall $n \geqq 1$) beim ersten Eintritt in das Entscheidungskästchen natürlich zu verneinen und führt zum Verlassen desselben über den entsprechenden Zweig. Der Programmlinie folgend wird nun p aktualisiert, und zwar gemäß der Anweisung des Hornerschen Schemas, die für $i = 1$

$$p := a_{n-1} + x_0 \cdot p$$

lautet, wobei p auf der rechten Seite des Zuweisungszeichens den Wert a_n hat. Nunmehr erhöht man die Zählvariable um 1[1]) und kehrt zum Test der Abbruchbedingung zurück. Wenn i auf diese Weise den Wert $n+1$ erreicht hat, ist der Horner-Algorithmus abgearbeitet, und es wird die Ausgabe von p $(= P(x_0))$ veranlaßt.

Als Beispiel betrachten wir mit zwischengeschalteter Rekonvertierung die Umwandlung der Oktalzahl **7 4 4 3** in die Darstellung im *Duodezimalsystem* $(g = 12)$.

[1]) Man vergleiche die Bemerkungen zur Formel (4) in 1.3.

In diesem benutzt man die Dezimalziffern, denen noch zwei Zeichen für 10 und 11 hinzuzufügen sind. MfL Bd. 1, 3.8., folgend, nehmen wir dafür **A** und **B**. Die Einheit der zweiten Stelle wird im deutschen Sprachgebrauch als *Dutzend*, die der dritten als *Gros* bezeichnet. Beginnend mit der höchsten Stelle notiert man die Zahlwerte der Ziffern der umzuwandelnden positionellen Darstellung als Koeffizienten eines Polynoms und berechnet dieses für die Basis g. Im Beispiel erhält man so

	7	4	4	3
		56	480	3872
8				
	7	60	484	3875

also wieder die oben konvertierte Dezimalzahl **3875**. Die Duodezimaldarstellung ergibt sich daraus mit Hilfe des Konvertierungsalgorithmus gemäß dem folgenden Protokoll:

g	n	x	q	w	Darstellung im Duodezimalsystem $\mathfrak{D}$
12	3875	3875	322	11	**AB22**
		322	26	10	**22BA**
		26	2	2	
		2	0	2	

Besonders einfach ist die Umwandlung einer Oktalzahl in die duale (oder dyadische oder binäre) Darstellung ($g=2$) und umgekehrt. Hier gelangt man unmittelbar zum Ziel durch *duale Verschlüsselung* der einzelnen Oktalziffern. Damit ist gemeint, daß jede derselben durch eine *Triade* von Dualziffern ersetzt wird, die bis auf eventuell vorangehende Nullen die Dualdarstellung der Oktalziffer ist:

Oktalziffer	Triade der dualen Verschlüsselung
0	**000**
1	**00L**
2	**0L0**
3	**0LL**
4	**L00**
5	**L0L**
6	**LL0**
7	**LLL**

Beispielsweise erhält man so für die Oktalzahl **7 4 4 3** die Dualdarstellung

LLL L00 L00 0LL.

Die Umkehrbarkeit des Vorgehens liegt auf der Hand. Zum Beweis braucht man

nur in der dyadischen Darstellung, mit den Einern beginnend, jeweils drei Glieder zusammenzufassen und eine entsprechende Potenz von 8 auszuklammern:

$$\begin{aligned} n &= z_0 + z_1 \cdot 2 + z_2 \cdot 2^2 + z_3 \cdot 2^3 + z_4 \cdot 2^4 + z_5 \cdot 2^5 + z_6 \cdot 2^6 + \cdots \\ &= (z_0 + z_1 \cdot 2 + z_2 \cdot 2^2) + (z_3 + z_4 \cdot 2 + z_5 \cdot 2^2)\, 8^1 + (z_6 + \cdots)\, 8^2 + \cdots; \end{aligned}$$

da die Werte w_k $(k = 0, 1, 2, \ldots)$ in den Klammern der Abschätzung

$$0 \leqq w_k < 8$$

genügen, folgt die Behauptung aus (3).

Analog kann man bei allen Positionssystemen verfahren, deren Basis g eine Potenz von 2 ist. Ausgehend von der Darstellung im *Hexagesimalsystem* ($g = 16 = 2^4$), dessen Ziffern mit

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (9)

bezeichnet seien, ergibt sich danach die entsprechende Darstellung im Binärsystem, indem man die Ziffern (9) durch **0, L**-*Tetraden* gemäß der folgenden Tabelle ersetzt:

Hexagesimalziffer	Tetrade der dualen Verschlüsselung
0	**0000**
1	**000L**
2	**00L0**
3	**00LL**
4	**0L00**
5	**0L0L**
6	**0LL0**
7	**0LLL**
8	**L000**
9	**L00L**
A	**L0L0**
B	**L0LL**
C	**LL00**
D	**LL0L**
E	**LLL0**
F	**LLLL**

Beispielsweise lautet die Dualdarstellung der Hexagesimalzahl **7 A 1 E**

0LLL L0L0 000L LLL0.

Die Umwandlung der Dualzahl **LL00LLLLL0L0L** in die hexagesimale Darstellung ergibt sich mit Hilfe der Klammerung und Ergänzung

(000L)(L00L)(LLLL)(0L0L)

zu **1 9 F 5.**

Wie wir in 2.3. begründen werden, spielen in der Rechentechnik das Binärsystem und positionelle Darstellungen, die sich wie im Fall des Oktal- und des

Hexagesimalsystems leicht in die duale Darstellung überführen lassen, eine besondere Rolle.

Zur Darstellung von Brüchen benötigt man Potenzen der Basis g mit negativen ganzzahligen Exponenten. Wir erörtern einen Algorithmus zur Entwicklung einer Zahl m $(0 \leqq m < 1)$ in einen g-adischen Bruch. Um in der Darstellung

$$m = w_{-1} \cdot g^{-1} + w_{-2} \cdot g^{-2} + \cdots$$

zunächst w_{-1} zu ermitteln, wird mit g multipliziert:

$$g \cdot m = w_{-1} + w_{-2} \cdot g^{-1} + \cdots .$$

Danach ist w_{-1} wegen $0 \leqq g \cdot m < g$ als die der natürlichen Zahl *entier*$(g \cdot m)$ in dem g-adischen Positionssystem entsprechende Ziffer bestimmbar. Nunmehr ergibt sich aus

$$(g \cdot m - w_{-1}) \cdot g = w_{-2} + w_{-3} \cdot g^{-1} + \cdots$$

in analoger Weise w_{-2} usw.

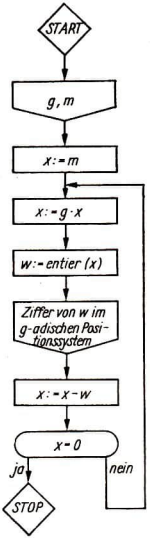


Abb. 2.9

Das Verfahren läßt sich mit dem PAP der Abb. 2.9 beschreiben. Beispielsweise erhält man für den Dezimalbruch $m = 0{,}4$ und $g = 2$ folgendes Protokoll:

g	m	x	w	Darstellung im Dualsystem (Ziffernfolge nach dem Komma)
2	0,4	0,4	0	**0LL00...**
		0,8	1	
		0,8	1	
		1,6	0	
		0,6	0	
		1,2		
		0,2		
		0,4		
		0,4		
		0,8		

Der Algorithmus bricht nicht ab; der Dezimalbruch 0,4 wird in den periodischen Dualbruch $\mathbf{0{,}0\overline{LL0}}$ übergeführt. Zur Probe rekonvertieren wir über die Summation der geometrischen Reihe

$$m=\frac{3}{8}\left(1+\frac{1}{2^4}+\frac{1}{2^8}+\cdots\right)=\frac{3}{8}\cdot\frac{1}{1-\frac{1}{2^4}}$$

$$=\frac{3}{8}\cdot\frac{16}{15}=\frac{2}{5}=0{,}4.$$

Aus spätbabylonischer Zeit (nach 600 v. u. Z.) sind uns umfängliche numerische Rechnungen mit bis zu 17 Sexagesimalstellen überliefert. Nach O. NEUGEBAUER [53] war folgender Näherungswert für $\sqrt{2}$ bekannt:[1])

60^0 60^{-1} 60^{-2} 60^{-3}

Um diesen Wert dezimal zu bestimmen, rekonvertieren wir mit Hilfe des Hornerschen Schemas:

	10	51	24	1
1		$0{,}1\overline{6}$	$0{,}85\overline{27}$	$0{,}4142\overline{129}$
$\overline{60}$	10	$51{,}1\overline{6}$	$24{,}85\overline{27}$	$1{,}4142\overline{129}$

Auf sechs Stellen nach dem Komma gerundet ergibt sich also der dezimale Näherungswert 1,414213.

[1]) Bei mindestens vier Zehnern oder Einern in einem Wortsymbol für eine Sexagesimalziffer werden mit diesen in der angedeuteten Weise gewisse Bündelungen vorgenommen. Ein „Sexagesimalkomma" wurde nicht benutzt; die Zuordnung der Ziffern zu den Potenzen von 60 mußte aus dem Zusammenhang entnommen werden.

Die Durchführung der vier Grundrechenoperationen in einem Positionssystem wurde in MfL Bd. 1, 3.8., erörtert. Danach läßt sich die Multiplikation auf wiederholte Additionen, die Division auf Subtraktionen zurückführen, wobei gewisse *Operandenverschiebungen* vorzunehmen sind. Im Hinblick auf die maschinelle Ausführung wollen wir die Subtraktion in einem g-adischen System weiter analysieren und dabei annehmen, daß Operanden a, b (a, $b \in \mathbf{N}$) mit maximal n Stellen zu verarbeiten sind. Die Differenz $a-b$ gestattet die Darstellung

$$\begin{aligned} a-b &= a+(0-b) = a+[(g^n-1)-(g^n-1)-b] \\ &= a+[(g^n-1)-b]-g^n+1, \end{aligned} \tag{10}$$

in welcher $(g^n-1)-b$ als das $(g-1)$-*Komplement von* b bezeichnet wird. Dazu einige Beispiele in verschiedenen g-adischen Systemen für $n=6$:

g	g-adische Darstellung von b	des $(g-1)$-Komplements
10	**157231**	**999999 − 157231 = 842768** (Neunerkomplement)
8	**237415**	**777777 − 237415 = 540362** (Siebenerkomplement)
2	**0LL0L0**	**LLLLLL − 0LL0L0 = L00L0L** (Einerkomplement)

Man beachte, daß im Fall des Dualsystems die Ziffern des Einerkomplements aus denen von b durch Ersetzung von **0** durch **L** und **L** durch **0** („bitweise Negation"; vgl. 2.3.2.) gewonnen werden.

Auf Grund von (10) läßt sich nun folgendes Verfahren für die Subtraktion g-adisch dargestellter Operanden aus **N** ableiten: Ist $a > b$, so liefert die Addition von a und des $(g-1)$-Komplements von b einen **1**-Übertrag in der $(n+1)$-ten Stelle, der durch die Subtraktion von g^n annulliert wird. Aus der verbleibenden Zahl erhält man durch Addition von 1 das Resultat $a-b$. Ist $a \leqq b$, so ergibt sich bei der ersten Addition – das Resultat sei mit s bezeichnet – kein Überlauf in der $(n+1)$-ten Stelle. Indem man (10) nun zu

$$-\{(g^n-1)-[a+[(g^n-1)-b]]\}$$

umformt, erkennt man, daß die Ziffernfolge von $|a-b|$ als das $(g-1)$-Komplement von s zu erhalten ist.

Da die Fälle $a > b$ und $a \leqq b$ durch den Überlauf in der $(n+1)$-ten Stelle charakterisiert sind, läßt sich der Algorithmus der Subtraktion in **N** durch das in Abb. 2.10 angegebene Flußbild darstellen; darin bedeuten Operandennamen die jeweiligen g-adischen Darstellungen, und mit $a+k$ ist z. B. die g-adische Darstellung des Resultats gemeint.

Unsere Betrachtungen haben gezeigt, daß sich die vier Grundrechenarten in g-adischen Positionssystemen auf die folgenden Operationen zurückführen lassen:

Addition, Bildung des $(g-1)$-Komplements und Operandenverschiebung.

Damit ist es unter Beachtung der Erkenntnisse aus 2.3. zu begründen, daß die Funktion des Rechenwerks wesentlich die einer Addiervorrichtung ist.

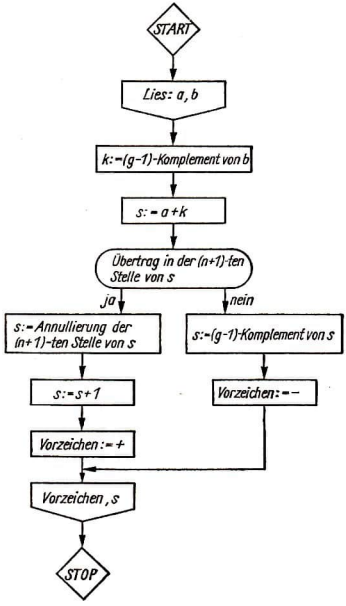


Abb. 2.10

2.3. Informationen und Signale in einer EDVA

2.3.1. Unter einer *Information* (Nachricht) wollen wir in erster Näherung eine Mitteilung verstehen, die einen Empfänger zu einem bestimmten Verhalten veranlaßt. Informationen können gespeichert, übertragen oder umgewandelt (verarbeitet) werden; dabei sind sie stets an Zustände oder Veränderungen materieller Systeme gebunden. Diese heißen *Signale* und sind als Träger der Information an der Beeinflussung des Empfängers unmittelbar beteiligt. Signale, die das gleiche Verhalten bewirken, nennt man äquivalent, und Informationen können schließlich als Äquivalenzklassen in der Menge der von einem bestimmten Empfänger interpretierbaren Signale erklärt werden.

Bei der Speicherung, Übertragung und Verarbeitung von Informationen in einer EDVA hat man von der Grundtatsache auszugehen, daß deren *Informationsträger* zwei physikalische Zustände stabil realisieren: In einer Leitung wird ein Impuls übertragen oder nicht; ein Relais kann „anziehen" oder „abfallen", eine Kontrollampe brennen oder nicht; an einer bestimmten Position eines Papierstreifens befindet sich eine Lochung, oder das ist nicht der Fall; eine magnetisierbare Schicht ist lokal polarisiert oder nicht; ein Transistor führt Strom oder sperrt; ein Ferritkern kann sich in einem positiven oder negativen Remanenzzustand befinden; ein als *Flipflop* bezeichnetes Rückkopplungssystem fungiert als bistabile Schaltung usw. Das legt nahe, alle zu verarbeitenden Informationen durch Wörter des Dualalphabets auszudrücken und **0, L** je einem der Zustände eines solchen *bistabilen Elements* zuzuordnen. Man nennt das eine *binäre Codierung* der Informationen und bezeichnet deren kleinste im Rechner speicherbare Einheit als ein *Bit*[1]). Gespeicherte und auf diese Weise gegenständlich repräsentierte Informationen (gelegentlich mit der Einschränkung: soweit sie nicht Bestandteile eines Programms sind) heißen *Daten*.

Die Mitteilung einer Zahl könnte man beispielsweise dadurch in ein Datum verwandeln, daß man sie im Dualsystem darstellt und der Ziffernfolge so unmittelbar eine Bitfolge zuordnet. Diese Konzeption benutzen die sogenannten *Dualrechner*, z. B. der vom VEB Kombinat Zentronik gefertigte Kleinrechner C 8205. Da Dualzahlen die dreieinhalbfache Länge der entsprechenden Dezimaldarstellung haben können, werden bei diesen Rechnern im allgemeinen 30 bis 40 Bit für das interne Zahlwort (ein materielles Abbild der Zahl z) veranschlagt.

2.3.2. Außerhalb einer EDVA wird heute fast überall dezimal gerechnet. Das erfordert bei einem Dualrechner die Konvertierung der Eingabedaten und Rekonvertierung der Ergebnisse. Dieser Umstand mindert den Vorteil der einfachen technischen Realisierbarkeit von Rechenoperationen im Dualsystem (vgl. 2.4.3.) und hat zur Konstruktion von in gewissem Sinne dezimal arbeitenden Maschinen geführt. Der erste elektronische Digitalrechner ENIAC (vgl. 2.1.2.) war eine Dezimalmaschine, die zur Speicherung von Ziffern sogenannte Ringzähler aus zehn bistabilen Elementen (Flipflops) benutzte. Von diesen war – der Ziffer entsprechend – genau eines „eingestellt", mit anderen Worten: Jede Dezimalziffer erforderte zehn Bitstellen, die gemäß den Dualwörtern[2])

0 ≙ 000000000L
1 ≙ 00000000L0
2 ≙ 0000000L00
3 ≙ 000000L000

[1]) Abgeleitet vom engl. binary digit für Binärziffer. Bei Verwendungs als Maßeinheit schreibt man bit.

[2]) Abweichend von MfL Bd. 1, 3.8., erscheint hier links von ≙ an Stelle eines Zahlzeichens das entsprechende dekadische Ziffernsymbol.

4 ≙ 00000L0000
5 ≙ 0000L00000
6 ≙ 000L000000
7 ≙ 00L0000000
8 ≙ 0L00000000
9 ≙ L000000000

(*Eins-aus-zehn-Code*) zu belegen waren. Da Flipflops relativ teure Bauelemente sind, stellte sich die Frage nach Codierungen der Dezimalziffern, die mit weniger Bitstellen auskommen. Eine solche ergibt sich z. B. aus der Dualdarstellung der durch die Dezimalziffer bezeichneten Zahl, für die maximal vier Stellen benötigt werden. Mit vorangehenden Nullen kann man so jede Dezimalziffer durch eine Tetrade von Binärziffern verschlüsseln und erhält für diesen sogenannten *direkten dezimal-binären Code* die Zuordnungstabelle

0 ≙ 0000
1 ≙ 000L
2 ≙ 00L0
3 ≙ 00LL
4 ≙ 0L00
5 ≙ 0L0L
6 ≙ 0LL0
7 ≙ 0LLL
8 ≙ L000
9 ≙ L00L .

Diese lexikographische Anordnung der Vier-Zeichen-Wörter des Binäralphabets kann durch die Tetraden

L0L0
L0LL
LL00
LL0L
LLL0
LLLL

vervollständigt werden, denen keine Ziffern entsprechen und die deshalb *Pseudotetraden* genannt werden.

Gegenüber dem reinen Binärcode der Dualrechner bietet die ziffernweise Verschlüsselung in Tetraden die Möglichkeit, das Erfordernis des dezimalen Rechnens außerhalb des Automaten mit den Vorteilen der technischen Realisierung der Rechenoperationen im Dualsystem zu verbinden: Konvertierung und Rekonvertierung beziehen sich nur auf die Korrespondenz zwischen Binärtetraden und Dezimalziffern; die Verknüpfung der Operanden erfolgt ziffernweise dual.

Dabei entsteht das Problem, bei den Resultaten Pseudotetraden zu vermeiden und gegebenenfalls einen Übertrag in die nächsthöhere Stelle zu veranlassen. Diesem

Erfordernis wird der direkte dezimal-binäre Code nicht in übersichtlicher Weise gerecht. Beispielsweise stellt sich die Addition von 6 und 7 in dem folgenden Schema von Codewörtern dar:

```
  0LL0
  0LLL
  LL0L
```

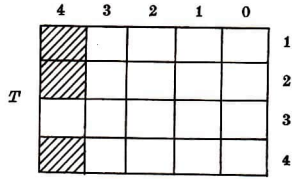
Das Resultat ist eine Pseudotetrade, der man nicht ohne weiteres ansieht, nach welchem Verfahren diese auf die 3 entsprechende Tetrade **00LL** und den Übertrag zu reduzieren ist. Bei einer abgewandelten Verschlüsselung – dem sogenannten *Dreiexzeßcode* – ist das leicht zu beschreiben. Jede Dezimalziffer z wird hierbei durch die Dualdarstellung von $z+3$ erfaßt. Die Tetradenzuordnung ist also folgende:

```
    0000
    000L   Pseudotetraden
    00L0
0 ≙ 00LL
1 ≙ 0L00
2 ≙ 0L0L
3 ≙ 0LL0
4 ≙ 0LLL
5 ≙ L000
6 ≙ L00L
7 ≙ L0L0
8 ≙ L0LL
9 ≙ LL00
    LL0L
    LLL0   Pseudotetraden
    LLLL
```

Am Anfang und Ende der lexikographisch geordneten Liste stehen je drei Pseudotetraden, denen keine Ziffern entsprechen.

Wie wir in 2.2. sahen, ist für die Subtraktion im Dezimalsystem die Bildung des Neunerkomplements wesentlich. Im Dreiexzeßcode gestaltet sich diese besonders einfach: Durch Vergleich der Tetraden für **0** und **9**, **1** und **8**, . . ., **4** und **5** erkennt man, daß diese durch Austausch von **0** und **L** oder – wie man mit Rücksicht auf die Interpretierbarkeit dieser Zeichen als Wahrheitwerte sagt – durch bitweise Negation auseinander hervorgehen. Einen solchen Code nennt man *symmetrisch*.

Bei der Addition von im Dreiexzeßcode verschlüsselten Zahlen ergeben sich in jeder Dezimalstelle Dualzahlen, die um 6 zu groß sind. Diesen Umstand kann man für die Bildung des Übertrags ausnutzen. Überschreitet nämlich die tatsächliche Ziffernsumme 9, so überschreitet die Summe der codierten Ziffern 15 und zeigt durch das **L** am Anfang des fünfziffrigen Dualwortes den Übertrag an. Nun muß nur noch dafür gesorgt werden, daß der an der betrachteten Position verbleibende Ziffernwert durch die ihm entsprechende Tetrade im Dreiexzeßcode wiedergegeben wird. Dabei sind zwei Fälle zu unterscheiden: Ist die Summe der unverschlüsselten Dezimalziffern kleiner als 10, so muß der Überschuß von 6 durch Subtraktion von 3 reduziert werden; ist hingegen ein Übertrag erfolgt, bei Verschlüsselung der 6-Überschuß also in die folgende Stelle abgewandert, so muß eine 3 addiert werden. Der bei der Addition an jeder Stelle abzuarbeitende Algorithmus kann demnach durch das Flußbild in Abb. 2.11 beschrieben werden, in welchem z_1, z_2 die zu addierenden Dezimalziffern und T ein Rechentableau der Form



bedeutet. Dessen Plätze werden gemäß den angeschriebenen Zeilen- und Spaltennummern mit T_{ik} $(i=1, 2, 3, 4;\ k=0, 1, 2, 3, 4)$ bezeichnet und erfahren durch den Algorithmus eine Belegung mit Binärziffern.

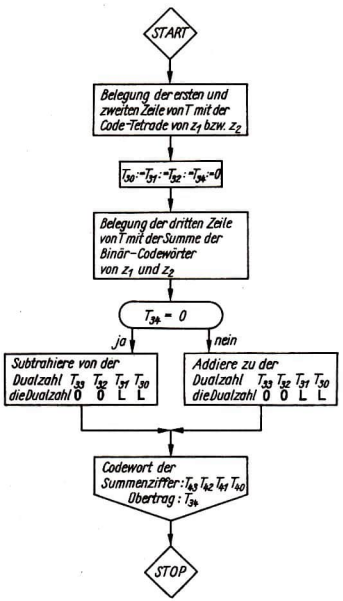
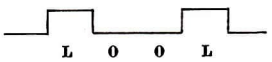


Abb. 2.11

Die Ausfüllung von T in einem konkreten Fall liefert im wesentlichen ein Protokoll des Algorithmus, z. B. ergibt sich für $z_1=7$, $z_2=6$:

	L	0	L	0
	L	0	0	L
L	0	0	L	L
	0	L	L	0

Daten werden in einer EDVA durch Folgen von Impulsen übertragen, die je nach ihrem Auftreten oder Ausbleiben die Binärzeichen **L** bzw. **0** vermitteln. Symbolisch kann man etwa den einer im Dreiexzeßcode verschlüsselten **6** entsprechenden Impulszug durch



wiedergeben. Hierbei können Störungen eintreten, die z. B. die Intensität eines Impulses so mindern, daß dieser von einem Schaltelement nicht mehr wahrgenommen wird. Eine Einschätzung der Wahrscheinlichkeit solcher Fehler läßt erkennen, daß es oft genügt, zur *Datensicherung* den Ausfall eines Bits zu kontrollieren. Betrachten wir zunächst den Fall, daß die Zeichenmenge, aus denen sich die zu verschlüsselnden Informationen zusammensetzen, die Ziffern **0, 1, . . ., 9** sind, dann würde der oben betrachtete Eins-aus-zehn-Code einen Fehler bei der Übermittlung eines Zeichens zu erkennen geben, wenn die Anzahl der mit **L** belegten Bits gerade ist. Die Kontrolle kann also in Form einer sogenannten *Paritätsprüfung* (*parity check*) durchgeführt werden.

Bei der Verschlüsselung in Tetraden ist diese Möglichkeit nicht gegeben, da es bei diesen Codes Zifferndarstellungen mit gerader und ungerader Anzahl von **L**-Bits gibt. Hier ist es naheliegend, zur Datensicherung ein fünftes Bit, das sogenannte *Prüfbit* zu verwenden.

2.3.3. Die von einer universell einsetzbaren EDVA zu verarbeitenden Informationen beziehen sich nicht nur auf Zahlen. Auszudruckende Ergebnisse müssen unter Umständen mit Kommentaren versehen werden, und spezielle Symbole benötigt man für die Steuerfunktionen innerhalb der Maschine. Das erfordert die Speicherung und Übertragung – also die Signalisierung – von Buchstaben und Sonderzeichen. Codes, die außer Ziffern auch solche Symbole zu verschlüsseln gestatten, werden *alphanumerisch* genannt. Beispielsweise können bei der EDVA R 300 zur Bildung von Informationen die in Tabelle 2.1 angegebenen **64** Zeichen des internen Alphabets (vgl. 2.1.1.) benutzt werden. Ihre Verschlüsselung durch Codewörter gleicher Länge des Dualalphabets (1) aus 2.2. würde mindestens sechs Bit-Stellen erfordern. Tatsächlich werden deren acht nach folgendem Prinzip belegt: Die ersten vier sogenannten *numerischen Bits* enthalten die direkte duale Ver-

schlüsselung der Spaltennummer, die als *Überbits* bezeichneten der Position 6 und 7 die der Zeilennummer des obigen Schemas. Zur Datensicherung wird an fünfter Stelle ein Prüfbit eingefügt, dessen Belegung noch von einem achten Bit, dem sogenannten *Wortmarkenbit* abhängt:

Wortmarkenbit	w
Überbits	v
	u
Prüfbit	p
numerische Bits	8
	4
	2
	1

Die angegebenen Bezeichnungen dienen zur kurzen Beschreibung der maschineninternen Zeichendarstellung, wobei die numerischen Bits durch die entsprechenden Stellenwerte des Dualsystems unterschieden sind.

Spalte / Zeile	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	⌑	#	(	:	▼	]
1	+	a	b	c	d	e	f	g	h	i	~	.	;	!	▼▼	„
2	−	j	k	l	m	n	o	p	q	r	≈	)	*	=	≺	?
3	´	/	s	t	u	v	w	x	y	z	≋	,	%	△	≻	[

Tabelle 2.1

Die Wortmarke und die Sonderzeichen ~ (Satzmarke), ≈ (Gruppenmarke) ≋ (Blockmarke) dienen der Abgrenzung von Informationen. Dazu geben wir folgende Erläuterung: Die kleinste Bit-Struktur des Hauptspeichers, die adressierbar ist, heißt *Byte*; in 2.1.1. sprachen wir in diesem Zusammenhang von „Zellen". Beim R 300 ist jedes Zeichen adressierbar, d. h., ein Byte umfaßt hier acht Bit. 1024 Byte werden üblicherweise zur Einheit K für die Kapazitätsbestimmung des Hauptspeichers zusammengefaßt. Zum Beispiel besitzt der Trommelspeicher des oben erwähnten C 8205 eine Kapazität von $4K$. Die von einer EDVA zu verarbeitenden Informationen sind Wörter über dem internen Alphabet. Es gibt Automaten (z. B. C 8205), wo diese eine *feste Länge* haben müssen, andere – wie der R 300 – gestatten die Verarbeitung von Informationen *variabler Wortlänge*. Bei diesen wird mit dem Zeichen begonnen, dessen Adresse im Befehl angegeben ist; danach werden die Zeichen mit den folgenden Adressen einbezogen, bis eine Wort-, Satz-, Gruppen- oder Blockmarke das Ende der Information bestimmt. Welches Zeichen Endmarke ist, muß im Befehl angegeben werden. Die Wortmarke kann dabei durch Belegung des w-Bits mit L auf jedes Zeichen gesetzt werden, während die übrigen Marken je ein Zeichen für sich beanspruchen. Unter Einbeziehung des Wortmarkenbits wird das Prüfbit so belegt, daß die Anzahl der L-Bits des Zeichens

ungerade ist. Beispielsweise würde der internen Darstellung der Zahl 1975 als Wort die 8-Bit-Codierung

1	9	7	5
L	0	0	0
0	0	0	0
0	0	0	0
L	L	0	L
0	L	0	0
0	0	L	L
0	0	L	0
L	L	L	L

entsprechen. Dabei hat man sich vorzustellen, daß für eine Operation, welche diese Zahl verarbeiten soll, die Einerstelle „5" zu adressieren ist und die folgenden Ziffern auf den Speicherplätzen mit den folgenden Nummern stehen. Die Unterstreichung der höchsten Stelle bezieht sich auf die Setzung des Wortmarkenbits, wodurch das Ende der Information angezeigt wird.

Abschließend wollen wir an Hand der betrachteten Beispiele das Problem der *Codierung* allgemein beschreiben. In jedem Fall handelte es sich um die Darstellung der Zeichen eines Alphabets Σ durch Wörter über einem Alphabet Φ. In den Beispielen war Φ stets das Binäralphabet (1) aus 2.2., und die Codewörter hatten alle gleiche Länge. Allgemein ist eine Codierung eine eindeutige Abbildung aus Φ^* (vgl. 1.3.) auf Σ, der *Code* selbst deren Urbildmenge. Eine Folge von Symbolen des Alphabets Σ, das auch *Inputmenge* genannt wird, geht bei der Codierung in ein Wort aus Φ^* über. Wenn aus diesem die Inputfolge eindeutig rekonstruierbar ist, heißt der Code *decodierbar*. Offensichtlich haben Codes mit gleicher Wortlänge diese Eigenschaft.

Man kann eine Codierung übersichtlich mit Hilfe eines als *Codebaum* bezeichneten speziellen Graphen darstellen. Abb. 2.12 zeigt diesen für den in 2.3.2. betrachteten Dreiexzeßcode. Je ein den Zeichen 0, L des Dualalphabets entsprechender Zweig führt von der Wurzel des Baumes zu den Knoten der ersten Stufe, von dort zu denen der zweiten und so fort bis hin zu Endknoten einer gewissen Stufe. Jeder Knoten ist von der Wurzel durch genau einen Weg erreichbar, dessen Bögen den Zeichen 0 und L zugeordnet sind und so ein Wort des Dualalphabets bestimmen. Wenn dieses Wort Codewort eines Zeichens aus Σ ist, ordnet man es dem Knoten zu, in welchem der betrachtete Weg endet. Analog verfährt man, wenn das Alphabet Φ D Zeichen enthält, indem man einen Baum mit D Verzweigungen konstruiert. Dieser enthält, bis zur k-ten Stufe ausgeführt, D^k Knoten.

Für ein eingehenderes Studium von Codierungsproblemen vgl. [34], Kapitel VII.

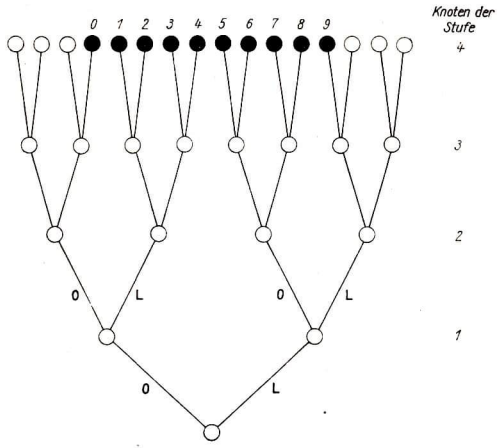


Abb. 2.12

2.4. Informationsverarbeitung durch binäre Schaltsysteme

2.4.1. Dieser Abschnitt ist dem Entwurf von Systemen zur Umwandlung binär codierter Informationen gewidmet. Zur Einführung in das Problem betrachten wir Beispiele, zunächst die Addition von Zahlen auf Grund ihrer Darstellung im Dualsystem. An der k-ten Stelle ist mit drei Binärwerten zu rechnen: den Ziffern dieser Stelle beider Operanden a, b und dem Übertrag $\ddot{u}$ von der $(k-1)$-ten Stelle. Als Ergebnis sind die k-te Ziffer der Summe s und der nächste Übertrag festzuhalten. Man vergleiche dazu die Tabelle 2.2, wo die Indizes die Stellennummer der zu betrachtenden Binärgrößen angeben.

a_k	b_k	$\ddot{u}_{k-1}$	s_k	$\ddot{u}_k$
L	L	L	L	L
L	L	0	0	L
L	0	L	0	L
L	0	0	L	0
0	L	L	0	L
0	L	0	L	0
0	0	L	L	0
0	0	0	0	0

Tabelle 2.2

Im weiteren wird uns die technische Lösung der Aufgabe interessieren, die Zuordnung der 0, L-Triaden des linken Teils der Tabelle zu den Dualziffern der Spalte von s_k bzw. $\ddot{u}_k$ zu realisieren.

Als ein Beispiel des täglichen Lebens betrachten wir eine Beleuchtung, die von n Stellen beliebig ein- und ausgeschaltet werden kann (Treppenhausbeleuchtung). Den Schaltstellen ordnen wir die Variablen $x_1, x_2, \ldots, x_n$ zu, die gemäß den Zuständen „eingeschaltet"/„ausgeschaltet" mit L bzw. 0 belegt werden sollen. Es existieren 2^n solcher Belegungen, die man sich in einer Tabelle lexikographisch geordnet notiert denke. Neben jedem dieser n-Tupel soll in einer weiteren y-Spalte durch L bzw. 0 der jeweilige Beleuchtungseffekt „hell" bzw. „dunkel" ausgedrückt werden; steht dabei am Anfang der Tabelle L, so ist die weitere Zeichenfolge zwangsläufig: Bei Übergang von einem Schalt-n-Tupel zum folgenden ändert sich der Beleuchtungswert oder bleibt erhalten, je nachdem, ob sich die beiden n-Tupel auf einer ungeraden bzw. geraden Anzahl von Plätzen unterscheiden. Tabelle 2.3

x_1	x_2	x_3	y
L	L	L	L
L	L	0	0
L	0	L	0
L	0	0	L
0	L	L	0
0	L	0	L
0	0	L	L
0	0	0	0

Tabelle 2.3

gibt die Verhältnisse für $n=3$ wieder. Die Aufgabe besteht darin, ein solches Beleuchtungssystem zu konstruieren.

Schließlich befassen wir uns mit der technischen Realisierung der binären Codierung und Decodierung von Dezimalziffern bezüglich des Dreiexzeßcodes. Diese lassen sich auf Grund der Darstellung im Eins-aus-zehn-Code unmittelbar signalisieren: Man verwendet zehn Leitungen und übermittelt der Ziffer entsprechend auf genau einer einen Impuls. Diese Information ist gemäß der Tabelle 2.4 automatisch in eine bestimmte Tetrade aus den Binärzeichen 0, L umzuwandeln.

Zeilennummer	Dezimalziffer	Eins-aus-zehn-Code										Dreiexzeßcode			
		x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_0	y_1	y_2	y_3
0	9	L	0	0	0	0	0	0	0	0	0	L	L	0	0
1	8	0	L	0	0	0	0	0	0	0	0	L	0	L	L
2	7	0	0	L	0	0	0	0	0	0	0	L	0	L	0
3	6	0	0	0	L	0	0	0	0	0	0	L	0	0	L
4	5	0	0	0	0	L	0	0	0	0	0	L	0	0	0
5	4	0	0	0	0	0	L	0	0	0	0	0	L	L	L
6	3	0	0	0	0	0	0	L	0	0	0	0	L	L	0
7	2	0	0	0	0	0	0	0	L	0	0	0	L	0	L
8	1	0	0	0	0	0	0	0	0	L	0	0	L	0	0
9	0	0	0	0	0	0	0	0	0	0	L	0	0	L	L

Tabelle 2.4

Den betrachteten Beispielen ist folgendes gemeinsam: Gesucht ist ein informationsverarbeitendes System, das n-Tupeln binärer Eingangsgrößen (*Inputs*) x_i in vorgeschriebener Weise m binäre Ausgangsgrößen (*Outputs*) y_j in Form entsprechender Signale zuordnet.

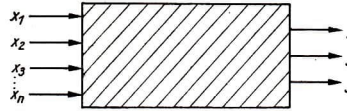


Abb. 2.13

Abb. 2.13 veranschaulicht das Problem in einer *Black-box*-Darstellung, da uns bislang nur die externe Beschreibung des Systems in Form von Zuordnungstabellen für die x_i, y_j, nicht aber dessen innere Struktur bekannt ist.

Jede Outputgröße y ist eine auf dem n-fachen kartesischen Produkt der Menge $\{0, L\}$ definierte Funktion f, deren Werte in dieser Menge liegen:

$$f\colon \{0, L\}^n = \{0, L\} \times \{0, L\} \times \cdots \times \{0, L\} \to \{0, L\}. \tag{1}$$

Eine Abbildung der Form (1) heißt eine *Boolesche Funktion.*[1]) Das uns in diesem Abschnitt interessierende Problem kann daher endgültig als das der technischen Realisierung Boolescher Funktionen charakterisiert werden. In 2.4.2. sind Hinweise enthalten, wie diese mit zahlreichen Fragestellungen der Praxis zusammenhängende Aufgabe im Rahmen der Aussagenlogik modelliert und gelöst werden kann.

2.4.2. Wir machen uns zunächst klar, daß sich die bei Belegung von Ausdrücken der Aussagenlogik ergebenden Wahrheitswerttabellen („wahr" und „falsch" mögen L bzw. 0 entsprechen) als Boolesche Funktionen interpretieren lassen. Beispielsweise erhält man für die

Negation $\neg\, x$, *Konjunktion* $x_1 \wedge x_2$, *Alternative* $x_1 \vee x_2$,
Implikation $x_1 \Rightarrow x_2$, *Äquivalenz* $x_1 \Leftrightarrow x_2$

die in Tabelle 2.5 wiedergegebenen Werteverläufe. Verschiedene Ausdrücke können

x	$\neg x$
L	0
0	L

x_1	x_2	$x_1 \wedge x_2$	$x_1 \vee x_2$	$x_1 \Rightarrow x_2$	$x_1 \Leftrightarrow x_2$
L	L	L	L	L	L
L	0	0	L	0	0
0	L	0	L	L	0
0	0	0	0	L	L

Tabelle 2.5

[1]) Im Beispiel der Codierung und Decodierung von Dezimalziffern sind die den Outputs entsprechenden Booleschen Funktionen durch das Problem zunächst nur auf einer Teilmenge von $\{0, L\}^n$ definiert.

in dieser Weise dieselbe Boolesche Funktion erzeugen; sie heißen dann *semantisch äquivalent.*

Für die Theorie der informationsverarbeitenden Systeme ist wesentlich, daß sich der hier festgestellte Sachverhalt umkehren läßt:

Satz 1. *Zu jeder Booleschen Funktion von n Veränderlichen gibt es einen aussagenlogischen Ausdruck in diesen Variablen, der durch Belegung derselben mit* 0 *bzw.* L *die Werte der Booleschen Funktion erzeugt.*

Beim Beweis geht man von der tabellarischen Darstellung der Booleschen Funktion aus und achtet entweder auf die Belegungen, denen der Funktionswert L, oder auf diejenigen, denen 0 entspricht. Wir betrachten zunächst den ersten Fall. Jeder dieser L-Belegungen $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ ordnet man die *Elementarkonjunktion*

$$x_1^{(\alpha_1)} \wedge x_2^{(\alpha_2)} \wedge \cdots \wedge x_n^{(\alpha_n)}$$

zu, wobei

$$x_i^{(\alpha_i)} = \begin{cases} x_i & \text{für } \alpha_i = \mathrm{L}, \\ \neg\, x_i & \text{für } \alpha_i = 0 \end{cases} \qquad (i = 1, 2, \ldots, n) \tag{2}$$

bedeutet. Die alternative Verbindung aller dieser Elementarkonjunktionen liefert einen Ausdruck, der das Gewünschte leistet. An Stelle von $\neg\, x$ notieren wir dabei zur Vereinfachung der Schreibweise $\bar{x}_i$. Wir erläutern das Vorgehen an der in Tabelle 2.2 gegebenen Funktion zur Bestimmung von s_k bei der Addition im Dualsystem (vgl. Tabelle 2.6).

x_1	x_2	x_3	y	Elementarkonjunktionen	Elementaralternativen
L	L	L	L	$x_1 \wedge x_2 \wedge x_3$	
L	L	0	0		$\bar{x}_1 \vee \bar{x}_2 \vee x_3$
L	0	L	0		$\bar{x}_1 \vee x_2 \vee \bar{x}_3$
L	0	0	L	$x_1 \wedge \bar{x}_2 \wedge \bar{x}_3$	
0	L	L	0		$x_1 \vee \bar{x}_2 \vee \bar{x}_3$
0	L	0	L	$\bar{x}_1 \wedge x_2 \wedge \bar{x}_3$	
0	0	L	L	$\bar{x}_1 \wedge \bar{x}_2 \wedge x_3$	
0	0	0	0		$x_1 \vee x_2 \vee x_3$

Tabelle 2.6

Durch die alternative Verbindung der Elementarkonjunktionen gewinnt man den Ausdruck

$$(x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \bar{x}_2 \wedge \bar{x}_3) \vee (\bar{x}_1 \wedge x_2 \wedge \bar{x}_3) \vee (\bar{x}_1 \wedge \bar{x}_2 \wedge x_3),$$

der bei jeder der zu betrachtenden L-Belegungen den Wert L ergibt, da jeweils die aus dieser Belegung abgeleitete Elementarkonjunktion dabei den Wert L annimmt. Für eine 0-Belegung ist in jeder Elementarkonjunktion mindestens ein Konjunktionsglied (Variable oder negierte Variable) vorhanden, das zu 0 wird und

so der Elementarkonjunktion insgesamt den Wert 0 erteilt. Diese Schlußweise kann man natürlich auf den allgemeinen Fall übertragen und darauf den Beweis der Behauptung gründen. Der so gewonnene Ausdruck ist die *kanonische alternative Normalform* jedes Ausdrucks, der die betrachtete Boolesche Funktion erzeugt. Der Ausartungsfall einer Booleschen Funktion, die konstant den Wert 0 annimmt, wird durch die Fiktion der *leeren* kanonischen alternativen Normalform einbezogen.

Bei der Betrachtung der 0-Belegungen $(\beta_1, \beta_2, \ldots, \beta_n)$ erzeugt man daraus zunächst durch bitweise Negation Belegungen $(\alpha_1, \alpha_2, \ldots, \alpha_n)$, ordnet diesen gemäß (2) die *Elementaralternativen*

$$x_1^{(\alpha_1)} \vee x_2^{(\alpha_2)} \vee \cdots \vee x_n^{(\alpha_n)}$$

zu und gewinnt durch deren konjunktive Verbindung einen Ausdruck, der die Boolesche Funktion erzeugt. Im Beispiel ergibt sich so

$$(\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3).$$

Für eine L-Belegung nimmt jede dieser Elementaralternativen den Wert L an, da mindestens eine der darin vertretenen Variablen oder negierten Variablen diesen Wert produziert. Damit ergibt sich auch für die konjunktive Verbindung L. Bei einer 0-Belegung nimmt diejenige Elementaralternative den Wert 0 an, die aus dieser abgeleitet wurde; das hat den Wert 0 auch für die konjunktive Verbindung zur Folge. Allgemein ergibt sich auf diese Weise die *kanonische konjunktive Normalform* aller Ausdrücke, welche die Boolesche Funktion erzeugen. Der Ausartungsfall einer solchen, die konstant den Wert L annimmt, wird durch die Fiktion der *leeren* kanonischen konjunktiven Normalform einbezogen.

2.4.3. Satz 1 ist Grundlage für den Entwurf gewisser informationsverarbeitender Systeme, die sich in einer speziellen Verwirklichung folgendermaßen beschreiben lassen:

Sie setzen sich aus „Schaltern" zusammen, die zweier Stellungen fähig sind und so binäre Signale realisieren können. Technische Beschaffenheit, speziell die Art der Betätigung, dürfen zunächst außer acht bleiben. Es kann sich z. B. im Fall elektrischer Schaltungen um von einem Relais bediente Federkontakte oder um von Hand betätigte Schalter handeln. Man unterscheidet *Arbeitskontakte* (Symbol —o/ o—) und *Ruhekontakte* (Symbol —o o—):

Ein {*Arbeitskontakt* / *Ruhekontakt*} *stellt genau dann eine leitende Verbindung her, wenn die betätigende Vorrichtung entsprechend dem Input* {L *eingeschaltet* / 0 *ausgeschaltet*} *ist.*

Abb. 2.14 veranschaulicht im Prinzip die Funktion eines Relais mit einem Arbeits- bzw. Ruhekontakt. Durch einen Erregerstromkreis, der eine Spule mit Weicheisen-

kern enthält, wird über einen Federkontakt ein zweiter Stromkreis gesteuert. Beim Arbeitskontakt ist dieser im erregten Zustand geschlossen, im nicht erregten Zustand geöffnet; der Ruhekontakt verhält sich umgekehrt. Offenbar kann man mittels geeigneter technischer Konstruktionen durch eine betätigende Vorrichtung

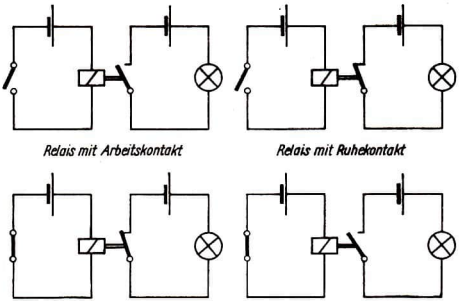
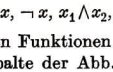


Abb. 2.14

– z. B. eine Relaisspule – zugleich mehrere Arbeits- und Ruhekontakte bedienen. Diesen zentralgesteuerten Kontaktgruppen ordnet man je eine Variable zu, die bei der Realisierung einer Booleschen Funktion ein Argument derselben vertritt. Ist deren Name x, so bezeichnet man jeden Arbeitskontakt der Gruppe mit x, jeden Ruhekontakt mit $\bar{x}$. Ein n-Tupel binärer Inputsignale wird n solcher Gruppen von Arbeits- und Ruhekontakten steuern, wobei L und 0 bezüglich der zugeordneten betätigenden Vorrichtung als „eingeschaltet" bzw. „ausgeschaltet" zu interpretieren sind.

Die zu entwerfenden Schaltungen sind hinsichtlich ihres Gesamtverhaltens *Zweipole* mit zwei nach außen führenden Leitungen, die wir gelegentlich durch das Symbol

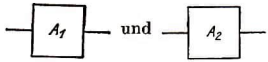


darstellen; je nachdem, ob sie durchgängig sind oder nicht, realisieren sie Outputsignale L bzw. 0. Die in Abb. 2.15 angegebenen Schaltungen von Arbeits- und Ruhekontakten zur Verwirklichung der durch die Ausdrücke

$$x,\ \neg x,\ x_1 \wedge x_2,\ x_1 \vee x_2$$

gegebenen Funktionen von einer bzw. zwei Veränderlichen sind naheliegend (die dritte Spalte der Abb. 2.15 lasse man zunächst unberücksichtigt). Nehmen wir ferner an, daß uns zu den von Ausdrücken A_1 und A_2 erzeugten Booleschen Funk-

tionen Zweipole



aus Arbeits- und Ruhekontakten bekannt sind, dann realisieren offenbar die Schaltungen aus Abb. 2.16 die durch $A_1 \wedge A_2$ bzw. $A_1 \vee A_2$ definierten Funktionen.

Auf Grund der in Abb. 2.15 und 2.16 dargestellten Sachverhalte ist es nun möglich, zu jeder Booleschen Funktion einen Zweipol in Form einer Reihen-Parallel-

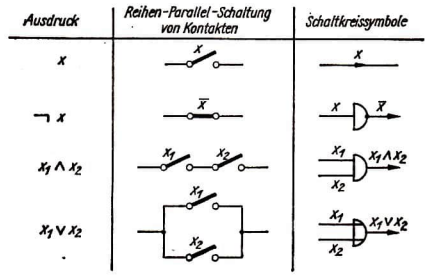


Abb. 2.15

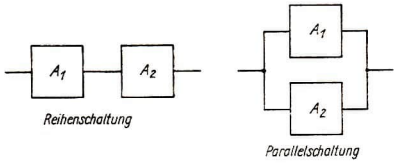


Abb. 2.16

Schaltung von Arbeits- und Ruhekontakten zu entwerfen, der mit seinen Outputsignalen deren Werte anzeigt. Dazu geht man von der im Beweis von Satz 1 betrachteten kanonischen alternativen oder konjunktiven Normalform aus. Im ersten Fall wird jeder Elementarkonjunktion entsprechend dem Auftreten von Variablen und negierten Variablen eine Reihenschaltung entsprechender Arbeits- bzw. Ruhekontakte zugeordnet. Jeder derselben würde die durch die Elementarkonjunktion dargestellte Boolesche Funktion realisieren. Die Parallelschaltung aller dieser Zweipole liefert ein Schaltsystem, dessen Outputs die Werte der vorgelegten Funktion signalisieren. Abb. 2.17 veranschaulicht dies für das Beispiel der Tabelle 2.6.

Geht man von der kanonischen konjunktiven Normalform aus, so sind zunächst den Elementaralternativen entsprechende Parallelschaltungen von Arbeits- und Ruhekontakten herzustellen, die, in Reihe geschaltet, einen Zweipol liefern, der das Gewünschte leistet. Für das in Tabelle 2.6 dargestellte Beispiel ist dies in Abb. 2.18 illustriert.

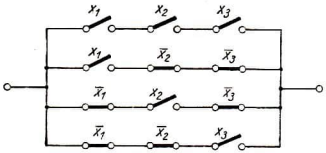


Abb. 2.17

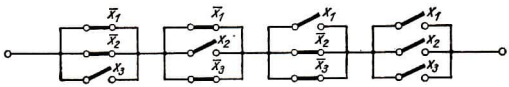


Abb. 2.18

Um eine möglichst einfache Schaltung zu erhalten, wird man den Entwurf nach der alternativen oder konjunktiven Normalform vornehmen, je nachdem, in welcher Darstellung weniger Elementarterme auftreten. Weitere Optimierung im Hinblick auf die Einsparung von Kontakten ist möglich durch semantisch äquivalenten Übergang zu noch einfacheren Ausdrücken. Wir wollen das an Hand eines Beispiels erläutern; die systematische Erörterung dafür geeigneter Methoden ist Gegenstand der *Schaltalgebra*. Betrachtet werde die den Übertrag bei Addition im Dualsystem charakterisierende Boolesche Funktion der Tabelle 2.2. Wir gehen von der kanonischen alternativen Normalform aus, wobei die Variablen der Übersichtlichkeit halber mit a, b, $\ddot{u}$ bezeichnet werden:

$$(a \wedge b \wedge \ddot{u}) \vee (a \wedge b \wedge \bar{\ddot{u}}) \vee (a \wedge \bar{b} \wedge \ddot{u}) \vee (\bar{a} \wedge b \wedge \ddot{u}). \tag{3}$$

Die folgenden Umformungen führen stets zu Ausdrücken, die zu (3) semantisch äquivalent sind. Zweimaliges Hinzufügen der ersten Elementarkonjunktion ergibt

$$(a \wedge b \wedge \ddot{u}) \vee (a \wedge b \wedge \bar{\ddot{u}}) \vee (a \wedge b \wedge \ddot{u}) \vee (a \wedge \bar{b} \wedge \ddot{u}) \vee (a \wedge b \wedge \ddot{u}) \vee (\bar{a} \wedge b \wedge \ddot{u})$$

und auf Grund von Gesetzen der Kommutativität und Distributibität

$$a \wedge b \wedge (\ddot{u} \vee \bar{\ddot{u}}) \vee a \wedge \ddot{u} \wedge (b \vee \bar{b}) \vee b \wedge \ddot{u} \wedge (a \vee \bar{a}).^{1})$$

Die aussagenlogischen Identitäten $\ddot{u} \vee \bar{\ddot{u}}$, $b \vee \bar{b}$ und $a \vee \bar{a}$ können im Sinne einer semantisch äquivalenten Umformung weggelassen werden, und man erhält

$$a \wedge b \vee a \wedge \ddot{u} \vee b \wedge \ddot{u} \tag{4}$$

[1]) Man beachte, daß $\vee$ stärker trennt als $\wedge$.

als einen die vorgelegte Boolesche Funktion darstellenden Ausdruck. Die entsprechende Reihen-Parallel-Schaltung von Kontakten zeigt Abb. 2.19.

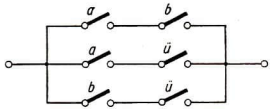


Abb. 2.19

An das in 2.4.1. betrachtete Beispiel der Treppenhausbeleuchtung knüpfen wir an, um zu zeigen, daß wesentliche Vereinfachungen erzielt werden können, wenn man außer den Reihen-Parallel-Schaltungen von Kontakten etwa noch sogenannte *Brückenschaltungen* zuläßt. Abb. 2.20 zeigt die auf der Basis der kanonischen alternativen Normalform zur Booleschen Funktion der Tabelle 2.3 konstruierte Reihen-Parallel-Schaltung, Abb. 2.21a einen Zweipol mit gleichem Outputverhalten in Brückenschaltung mit weniger Kontakten. Die Teilschaltung *ABCD* findet man in den handelsüblichen Kreuzschaltern vereinigt, deren Prinzip in Abb. 2.21b dargestellt ist. Zur

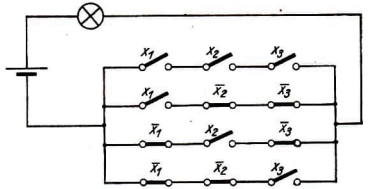


Abb. 2.20

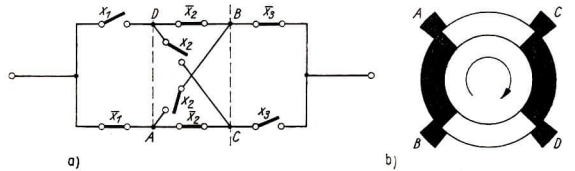


Abb. 2.21

Übung übertrage man das Prinzip dieser Brückenschaltung auf beliebig viele Schaltstellen. Im Fall $n=2$ vereinfacht sich der Zweipol in Abb. 2.20 zu dem in Abb. 2.22 gemäß der die Input-Output-Beziehungen bestimmenden Booleschen Funktion

x_1	x_2	y
L	L	L
L	0	0
0	L	0
0	0	L

die durch den Ausdruck $(x_1 \wedge x_2) \vee (\bar{x}_1 \wedge \bar{x}_2)$ dargestellt wird. Üblicherweise ersetzt man diesen Zweipol äquivalent durch die Wechselschaltung aus Abb. 2.23. Abschließend sei bemerkt, daß man mit Hilfe handelsüblicher Mehrkontaktschalter, die an Buchsen-

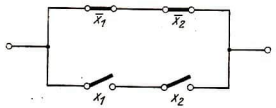


Abb. 2.22

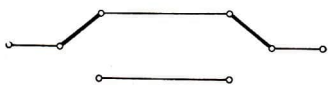


Abb. 2.23

felder angeschlossen werden, leicht Schülerarbeitsgeräte und Demonstrationsmodelle herstellen kann, auf denen sich durch Steckverbindungen Boolesche Funktionen realisieren lassen.

Die bisher benutzte graphische Darstellung von Reihen-Parallel-Schaltungen orientierte sich an deren technischer Realisierung durch Relaiskontakte. Wir wollen jetzt zu einer allgemeineren symbolischen Beschreibung übergehen, die allein ausdrückt, daß eine solche Schaltung Übertragungsglied zwischen binären Input- und Outputsignalen ist. Diese Symbolik ist mehr der Verwirklichung einer Booleschen Funktion mit Hilfe elektronischer Elementarglieder angepaßt, auf die wir hier allerdings nicht näher eingehen können. In Abb. 2.15 sind die allgemeinen Schaltkreissysmbole den bisher benutzten Darstellungen von Kontaktschaltungen gegenübergestellt. Wie bei der Wiedergabe von Schaltungen zu einer alternativen oder konjunktiven Normalform mit den allgemeinen Symbolen vorzugehen ist, sei am Beispiel der Booleschen Funktion aus Tabelle 2.6 erläutert. Wir gehen von der kanonischen alternativen Normalform aus, die auch der Darstellung in Abb. 2.17 zugrunde lag. Der Elementarkonjunktion $x_1 \wedge x_2 \wedge x_3$ würde in der Assoziation $(x_1 \wedge x_2) \wedge x_3$ die Teilschaltung aus Abb. 2.24 entsprechen. Die nächste Elementar-

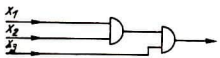


Abb. 2.24

konjunktion $(x_1 \wedge \bar{x}_2) \wedge \bar{x}_3$ wird mit Berücksichtigung zweier Negationen an die Signalleitungen der Variablen x_1, x_2, x_3 angeschlossen. Dabei sind Linienüberschneidungen, wenn sie nicht durch Punktierung als tatsächliche Verbindungen kenntlich gemacht wurden, bedeutungslos. Auf diese Weise erhält man die Teilschaltung aus Abb. 2.25, die, für sich betrachtet, zwei von den Variablen x_1, x_2, x_3 abhängende Boolesche Funktionen realisiert. So fortfahrend ergibt sich für die vier Elementarkonjunktionen die Darstellung aus Abb. 2.26. Die Parallelschaltung dieser Teil-

schaltungen liefert einen Zweipol, dessen Outputs die Werte der Booleschen Funktion aus Tabelle 2.6, d. h. die Ziffern bei der Addition im Dualsystem bestimmen (Abb. 2.27).

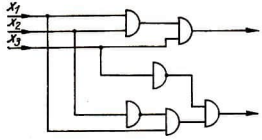


Abb. 2.25

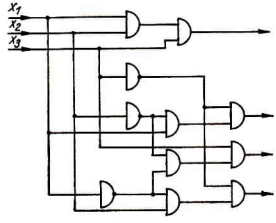


Abb. 2.26

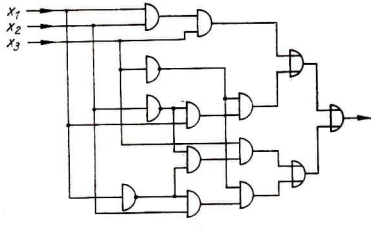


Abb. 2.27

Beim Entwurf der Schaltung wurden die Elementarkonjunktionen in einer bestimmten Zusammenfassung der Variablen berücksichtigt. Da aber für mehrgliedrige Konjunktionen und Alternativen das assoziative Gesetz gilt, d. h. Ausdrücke dieser Art, die sich nur durch die Assoziation der logischen Operanden unterscheiden, semantisch äquivalent sind, verwirklichen auch die dazu konstruierten Schaltungen die gleiche Boolesche Funktion. Zum Beispiel trifft das für die Schaltungen aus Abb. 2.28a und 2.28b zu, was die Verwendung des Symbols aus Abb. 2.28c nahe-

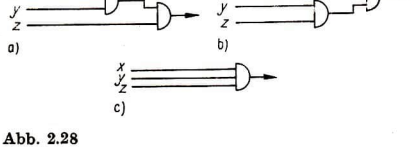


Abb. 2.28

legt, das entsprechend für mehr als drei Variable und die Alternative zu bilden ist. In diesem Zusammenhang sei darauf hingewiesen, daß es gerätetechnisch möglich ist, sogenannte UND- bzw. ODER-Glieder mit n Eingängen herzustellen. In der neuen Darstellungsweise würde Abb. 2.27 die in Abb. 2.29 gezeigte Vereinfachung erfahren.

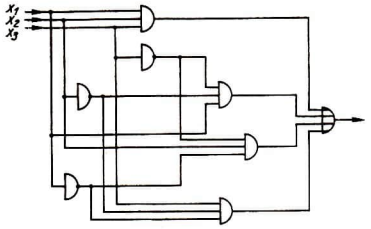


Abb. 2.29

Abb. 2.30 zeigt eine daraus abgeleitete Schaltung, deren zweiter Ausgang die der $\ddot{u}_k$-Spalte in Tabelle 2.2 entsprechenden Signale liefert. Dem Entwurf der zugehörigen Teilschaltung liegt die Formel (4) zugrunde. Das Schaltsystem der Abb. 2.30 wird als *Adder* bezeichnet. Wegen anderer Realisierungsmöglichkeiten und der Erörterung technischer Aspekte muß auf die Spezialliteratur verwiesen werden (etwa [7], [72]). Für die abschließende Betrachtung sehen wir von der besonderen Verwirklichung ab und

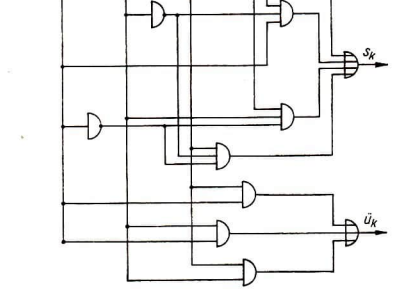


Abb. 2.30

symbolisieren den Adder durch das Blockschaltbild aus Abb. 2.31. Mehrere gemäß Abb. 2.32 geschaltete Adder ergeben ein binäres Addierwerk entsprechender Stellenzahl. Da der Übertrag in der Einerstelle stets 0 ist, könnte man dort ein einfacheres, als *Halbadder* bezeichnetes Schaltsystem verwenden.

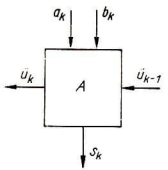


Abb. 2.31

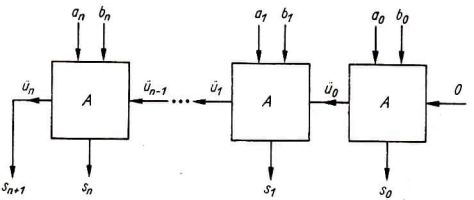


Abb. 2.32

2.5. Probleme des Rechnens mit beschränkter Stellenzahl

2.5.1. In 2.2. wurde darauf hingewiesen, daß die Ausführung arithmetischer Operationen nach einem Algorithmus die Darstellung der beteiligten Zahlen in einem Positionssystem voraussetzt. Notwendigerweise muß dabei eine Beschränkung der Stellenzahl, allgemein gesprochen also die Ersetzung eines Operanden durch einen Näherungswert erfolgen. Neben der üblichen *Festkommadarstellung* wie zum Beispiel 3,14159 werden sogenannte *Gleitkommadarstellungen* oder *halblogarithmische Darstellungen* mit einem Skalenfaktor benutzt, die man kurz (obzwar nicht ganz korrekt) auch Festkomma- bzw. Gleitkommazahlen[1]) nennt. Der betrachtete Näherungswert von π ließe sich auf diese Weise u. a. in der Form

$$0{,}0314159 \cdot 10^2 \text{ oder } 31{,}4159 \cdot 10^{-1}$$

[1]) Bedingt durch die Syntax der Programmiersprachen setzt sich auch bei uns im täglichen Leben der Dezimalpunkt an Stelle des Kommas mehr und mehr durch. Sinngemäß wäre dann die Bezeichnung Festpunkt- bzw. Gleitpunktdarstellung von Zahlen zu verwenden.

schreiben. Alle in einer Zahldarstellung angegebenen Ziffern mit Ausnahme führender Nullen (bei Gleitkommazahlen natürlich ohne Berücksichtigung des Skalenfaktors) und solcher, die man nur zur Festlegung des Dezimalkommas benötigt, werden als *wesentlich* bezeichnet; die erste wesentliche Ziffer (mit dem höchsten Stellenwert) heißt *führende Ziffer*. Wollte man zum Beispiel die Loschmidtsche Zahl $6{,}025 \cdot 10^{23}$ der in einem Mol enthaltenen Moleküle ohne Skalenfaktor notieren, so müßten 20 Nullen angefügt werden, die alle nicht wesentlich sind. Die halblogarithmische Darstellung einer Zahl kann stets so gewählt werden, daß alle Ziffern wesentlich sind. Alle Formen des oben angegebenen Näherungswertes von π enthalten sechs wesentliche Ziffern. 31 ist ein Näherungswert von π^3 mit zwei, 31,00 ein solcher mit vier wesentlichen Ziffern.

Zu einer eindeutigen halblogarithmischen Darstellung für Zahlen, die verschieden von Null sind, gelangt man durch Festlegungen über den Exponenten des Skalenfaktors oder die Stellung des Kommas bezüglich der führenden Ziffer. Beispielsweise könnte man fordern, das Komma unmittelbar vor die führende Ziffer zu setzen. In dieser *Normierung*, die im folgenden beibehalten werden soll, würde die Angabe des betrachteten Näherungswertes $0{,}314159 \cdot 10^1$ lauten. Die hier zur Zahldarstellung eingeführten Begriffe lassen sich sinngemäß auf beliebige Positionssysteme übertragen. Bei normierten Gleitkommazahlen heißt der Faktor vor der Basispotenz – und auch die diesen darstellende Ziffernfolge – *Mantisse*. Zusammen mit dem *Exponenten* des Skalenfaktors bildet diese ein geordnetes Paar vorzeichenbehafteter Zahlen, das in einem Rechner, der mit einer Gleitkommaarithmetik arbeitet, in Form einer gewissen Bytefolge gespeichert wird. Es gibt Automaten, wo dieses Zahlwort eine feste Anzahl von Stellen mit einer bestimmten Aufteilung für die Ziffern der Mantisse und des Exponenten umfaßt; bei anderen (zum Beispiel R 300) ist die Länge des Zahlwortes in bestimmten Grenzen variabel. In jedem Fall steht nur eine endliche Menge von *Maschinenzahlen* zur Verfügung; wesentlich darauf ist zurückzuführen, daß im Körper der reellen Zahlen konzipierte Rechenalgorithmen bei der Übertragung in den Bereich der Maschinenzahlen zu qualitativ anderen Ergebnissen führen können.

Manche Rechner gestatten nur oder auch die Verarbeitung von Zahlen mit fester Kommastellung. Wir wollen annehmen, daß diese dem Bereich $|z| < 1$ angehören. Derartige Einschränkungen erschweren die Programmierung von Festkommarechnungen; es muß dafür gesorgt werden, daß alle sich dabei ergebenden Werte (also auch die Zwischenresultate) in dem engen Bereich der für die Maschine verfügbaren Festkommazahlen liegen. Zur Fixierung der Anschauung nehmen wir an, daß Fest- und Gleitkommazahlen konstanter Wortlänge verarbeitet werden, wobei man sich bei den letzten je ein Wort für die Mantisse und den Exponenten vorstellen mag. Die Anzahl der Ziffern nach dem Komma bei einer Festkommazahl und die der Mantissenstellen sei mit t bezeichnet.

2.5.2. Vor einer Analyse der Fehlerfortpflanzung bei arithmetischen Operationen führen wir Begriffe ein, die mit der Ersetzung von Zahlen durch Näherungswerte zusammenhängen, wobei es unerheblich ist, wie solche Approximationen zustande gekommen sind. Es sei darauf hingewiesen, daß deren Definition in der Literatur unterschiedlich ist. Wir bezeichnen eine Approximation der Zahl z mit z^* und erklären

$$\begin{aligned} \Delta z &:= z^* - z && \text{als } \textit{absoluten Fehler}, \\ \delta z &:= \frac{\Delta z}{z} \quad (z \neq 0) && \text{als } \textit{relativen Fehler} \end{aligned} \tag{1}$$

und

$$100\,\delta z \quad (z \neq 0) \qquad \text{als } \textit{prozentualen Fehler}$$

bei der Ersetzung von z durch den Näherungswert z^*.[1]) Der absolute Fehler ist häufig nicht angebbar, insbesondere dann, wenn z eine Meßgröße ist. In solchen Fällen muß man sich mit oberen Schranken $\square z$ für seinen Betrag begnügen; dann gilt

$$|z^* - z| \leqq \square z \tag{2}$$

und bezüglich des relativen Fehlers

$$|\delta z| = \left| \frac{z^* - z}{z} \right| \leqq \frac{\square z}{|z|}. \tag{3}$$

Die Abschätzung (3) ist unbrauchbar, wenn z nicht bekannt ist. Wegen

$$|z| = |z^* - \Delta z| \geqq |z^*| - |\Delta z| \geqq |z^*| - \square z$$

ist unter der Voraussetzung $|z^*| - \square z > 0$

$$|\delta z| \leqq \frac{\square z}{|z^*| - \square z}, \tag{4}$$

und man erhält in $\dfrac{\square z}{|z^*| - \square z}$ eine zwar gröbere, aber tatsächlich berechenbare obere Schranke für den Betrag des relativen Fehlers. Ist $\square z$ klein gegenüber $|z^*|$, so wird diese in praxi meist durch $\dfrac{\square z}{|z^*|}$ ersetzt.

Aufgabe 1. Für die Lichtgeschwindigkeit im Vakuum wurde 1941 von R. T. Birge $c = 299776 \pm 4$ km/s ermittelt. Man bestimme eine Schranke für den Betrag des relativen Fehlers dieser Messung.

Hat man einen Näherungswert z^* und eine obere Schranke $\square z$ für den absoluten Fehler, so läßt sich der exakte Wert z in das Intervall $z^* - \square z \leqq z \leqq z^* + \square z$ einschließen. Eine derartige Erfassung von z wäre bei der Tabulierung größerer Zahlenmengen unbequem; aus diesem Grunde trifft man eine Vereinbarung, die

[1]) Die Definitionen (1) werden z. B. von K. Samelson und F. L. Bauer in [10] und R. H. Pennington in [57] benutzt. Davon und voneinander abweichend sind die Begriffsbestimmungen in [17] und [11], Bd. 1.

ein solches Einschließungsintervall bereits an der positionellen Zahldarstellung erkennen läßt. Wir bezeichnen in einer g-adischen Zahldarstellung von z^* die bei der Potenz g^k stehende Ziffer ζ_k als *gültig* (auch sicher), wenn

$$|z^* - z| \leqq \frac{1}{2} g^k \tag{5}$$

gilt.[1]) Ist $l > k$ und ζ_k gültig, so natürlich auch ζ_l. In Tafelwerken sind in der Regel alle angegebenen Ziffern gültig. Ein Näherungswert für π mit vier gültigen Ziffern nach dem Komma wäre 3,1416, denn es ist

$$|3{,}1416 - \pi| \leqq \frac{1}{2} \cdot 10^{-4}.$$

Die Definition der gültigen Ziffer ist so gefaßt, daß beim Runden exakter Werte nach den gebräuchlichen Vorschriften alle verbleibenden Ziffern gültig sind. In Rechenautomaten erfolgt das Runden auf t Stellen meist durch Addition von $\frac{1}{2} \cdot 10^{-t}$ $\left(\text{bzw. } \frac{1}{2} \cdot 2^{-t} \text{ bei Dualrechnern}\right)$ und Weglassen der auf die t-te Stelle folgenden Ziffern. Dabei stelle man sich etwa vor, daß der Akkumulator des Rechenwerks ein Register doppelter Wortlänge ist und zu rundende Resultate dort mit $2t$ Stellen erscheinen. Beispielsweise würde bei Festkommarechnung mit $t = 4$ Stellen das Produkt

$$z = 0{,}6417 \cdot 0{,}8396 = 0{,}53877132$$

folgendermaßen auf den vierstelligen Wert z^* gerundet:

$$z + \frac{1}{2} \cdot 10^{-4} = 0{,}53882132$$

$$z^* = 0{,}5388$$

$$|z^* - z| \leqq 0{,}00002868, \quad \text{also } |z^* - z| \leqq \frac{1}{2} \cdot 10^{-4}.$$

Offenbar gilt allgemein für den Betrag des *absoluten Rundungsfehlers* bei Festkommarechnung

$$|z^* - z| \leqq \begin{cases} \frac{1}{2} \cdot 10^{-t} & \text{im Dezimalsystem,} \\ \frac{1}{2} \cdot 2^{-t} & \text{im Dualsystem,} \end{cases} \tag{6}$$

und ein Vergleich mit (5) zeigt, daß alle Stellen von z^* gültig sind.

Für den Übergang von z zu z^* ist bei diesem Vorgehen nur die $(t+1)$-te Stelle maßgebend: Es wird *abgerundet*, wenn die erste überschüssige Stelle 0, 1, 2, 3, 4

[1]) Gelegentlich wird die Anforderung an die Gültigkeit einer Ziffer zu

$$|z^* - z| \leqq \frac{1}{2} \omega g^k \quad \text{mit} \quad 0 < \omega \leqq 1$$

verschärft. (Vgl. [11], Bd. 1, S. 34ff.)

(Dezimalsystem) bzw. 0 (Dualsystem) ist, *aufgerundet*, wenn diese 5, 6, 7, 8, 9 bzw. 1 (L) ist. Das starre Aufrunden im Fall, daß die $(t+1)$-te Ziffer eine 5 (dezimal) bzw. 1 (dual) gefolgt von lauter Nullen ist, kann zu systematischen Fehlern führen. Dieser Nachteil wird wegen der technischen Einfachheit des Verfahrens in Kauf genommen. Bei Handrechnungen im Dezimalsystem rundet man meist nach dem Algorithmus des Flußbildes in Abb. 2.33; darin ist

$$r := \zeta_{t+1} + \zeta_{t+2} \cdot 10^{-1} + \zeta_{t+3} \cdot 10^{-2} + \cdots,$$

wobei

$$\zeta_{t+1},\ \zeta_{t+2}, \cdots$$

die Folge der nach der Rundung wegzunehmenden Ziffern bedeutet.

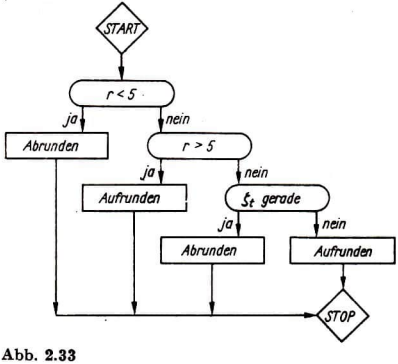


Abb. 2.33

Aufgabe 2. Man runde nach dem PAP der Abb. 2.33

$$0{,}1499\ldots = 0{,}14\bar{9}$$

auf eine Stelle nach dem Komma.

Zwischen der Zahl der gültigen Ziffern eines Näherungswertes und dem relativen Fehler besteht folgender Zusammenhang:

$$z^* = \zeta_m g^m + \zeta_{m-1} g^{m-1} + \cdots + \zeta_{m-n+1} g^{m-n+1} + \cdots$$

sei eine positive Näherung von z mit n gültigen Ziffern ($\zeta_m \neq 0$). *Dann* (7)

ist $\frac{1}{\zeta_m} g^{1-n}$ *eine obere Schranke für den Betrag des relativen Fehlers.*

Diese Schranke wird also durch Division von g^{1-n} durch den Wert der führenden Ziffer der Näherung gewonnen. Im Beweis folgen wir der Darstellung in [17].

Beweis. Auf Grund der Definition des Begriffs der gültigen Ziffer ist nach (5)

$$|z^* - z| \leqq \frac{1}{2} g^{m-n+1}$$

und daher

$$z \geqq z^* - \frac{1}{2} g^{m-n+1}.$$

Diese Abschätzung für z gilt erst recht, wenn z^* durch den nicht größeren Wert $\zeta_m g^m$ ersetzt wird, so daß

$$z \geqq \zeta_m g^m - \frac{1}{2} g^{m-n+1} = \frac{1}{2} g^m (2\zeta_m - g^{1-n}) \geqq \frac{1}{2} g^m (2\zeta_m - 1) \tag{8}$$

ist. Für das letztere beachte man, daß g^{1-n} seinen größten Wert für $n = 1$ annimmt. Wegen

$$2\zeta_m - 1 = \zeta_m + (\zeta_m - 1) \geqq \zeta_m$$

ergibt sich weiter $z \geqq \frac{1}{2} g^m \zeta_m$ und schließlich

$$\frac{|z^* - z|}{|z|} \leqq \frac{\frac{1}{2} g^{m-n+1}}{\frac{1}{2} g^m \zeta_m} = \frac{1}{\zeta_m} g^{1-n},$$

was zu beweisen war.

Als Beispiel zu (7) soll der prozentuale Fehler der Näherung $z^* = 3{,}14$ von π betragsmäßig abgeschätzt werden:

3,14 hat drei gültige Ziffern, und somit ist $\frac{1}{3} \cdot 10^{-2}$ eine Schranke für den Betrag des relativen und $\frac{1}{3}$ für den Betrag des prozentualen Fehlers. Verzichtet man auf die Majorisierung von $2\zeta_m - 1$ durch ζ_m in (8), so ergibt sich die schärfere Schranke $\frac{1}{5}$. In der Praxis kann für $n > 1$ im Dezimalsystem die Potenz g^{1-n} der Abschätzung (8) vernachlässigt werden, und man gewinnt dann die Schranke

$$\frac{1}{2\zeta_m} g^{1-n} \tag{9}$$

für den Betrag des relativen Fehlers.

Aufgabe 3. Im Tafelwerk für die Klassen 7 bis 12 wird der Wert des Planckschen Wirkungsquantums mit

$$h = 6{,}625 \cdot 10^{-34}\ \mathrm{W \cdot s^2} = 6{,}625 \cdot 10^{-27}\ \mathrm{erg \cdot s}$$

angegeben. Die Ziffer 2 dieses Näherungswertes ist gültig. Man bestimme eine Schranke für den Betrag des relativen Fehlers der Messung und vergleiche die Größenordnung des relativen und des absoluten Fehlers in Aufgabe 1 und 3.

Der Satz (7) gestattet in gewissem Sinne eine Umkehrung. In einem g-adischen Positionssystem sei z^* eine Näherung für z, und $n \in \mathbb{N}^*$. Dann gilt

$$\textit{Ist } \frac{|z^*-z|}{|z^*|} \leqq \frac{1}{2} \cdot g^{-n} \textit{ , so sind in der mit der führenden Ziffer beginnenden } g\textit{-adischen Darstellung von } z^* \textit{ die ersten } n \textit{ Ziffern gültig.} \tag{10}$$

Beweis. Ist

$$|z^*| = \zeta_m g^m + \zeta_{m-1} g^{m-1} + \cdots + \zeta_{m-n+1} g^{m-n+1} + \cdots \quad (\zeta_m \neq 0),$$

so folgt aus

$$\frac{|z^*-z|}{|z^*|} \leqq \frac{1}{2} g^{-n}$$

die Relation

$$|z^*-z| \leqq \frac{1}{2} g^{m-n+1}(\zeta_m g^{-1} + \zeta_{m+1} g^{-2} + \cdots) \leqq \frac{1}{2} g^{m-n+1}$$

und in Verbindung mit (5) die Behauptung.

In praxi wird $\frac{|z^*-z|}{|z^*|}$ in (10) meist durch eine obere Schranke für den Betrag des relativen Fehlers ersetzt.

2.5.3. Werden in einer mit z_1 und z_2 auszuführenden arithmetischen Operation Näherungswerte z_1^* und z_2^* benutzt, so erhält man eine mehr oder minder gute Näherung des exakten Ergebnisses. Wir befassen uns zunächst mit der Auswirkung von Fehlern bei den Operanden auf das Resultat einer der vier Grundverknüpfungen; diesen Effekt bezeichnet man als *lokale Fehlerfortpflanzung*. Untersuchungen, die alle bei der Abarbeitung eines numerischen Algorithmus auszuführenden Operationen in Betracht ziehen und Aussagen über die Genauigkeit des Endresultats machen, gehören zur Theorie der *globalen Fehlerfortpflanzung*.

Addition. Bezüglich des absoluten Fehlers gilt

$$\square(z_1+z_2) = \square z_1 + \square z_2, \tag{11}$$

d. h., die Summe von zwei oberen Schranken für den Betrag des absoluten Fehlers bei der Ersetzung von z_i durch z_i^* $(i=1, 2)$ ist eine solche Schranke für die Ersetzung von z_1+z_2 durch $z_1^*+z_2^*$.

Im Sinne dieses Kommentars sollen auch im folgenden Gleichungen der Form (11) interpretiert werden.

Beweis.

$$|z_1^*+z_2^*-(z_1+z_2)| = |\Delta z_1 + \Delta z_2| \leqq |\Delta z_1| + |\Delta z_2| \leqq \square z_1 + \square z_2.$$

Aus (11) folgt, daß beim Rechnen mit Näherungswerten die Genauigkeit einer Summe durch den Operanden mit der geringsten Genauigkeit bestimmt wird.

Aufgabe 4. Mit Bezug auf die letzte Bemerkung begründe man für Handrechnung im Dezimalsystem die Praktikerregel: Ist k die niedrigste gültige Ziffer des Summanden mit der geringsten Genauigkeit, so runde man vor Ausführung der Addition alle Operanden auf die Position $k-1$ (oder auch $k-2$) und das Resultat schließlich auf die vorletzte Dezimale.

Für den relativen Fehler einer Summe gilt

$$\delta(z_1+z_2)=\frac{z_1}{z_1+z_2}\,\delta z_1+\frac{z_2}{z_1+z_2}\,\delta z_2 \quad (z_1, z_2, z_1+z_2\neq 0). \tag{12}$$

Es ist nämlich

$$\begin{aligned}\delta(z_1+z_2)&=\frac{z_1^*+z_2^*-(z_1+z_2)}{z_1+z_2}\\&=\frac{z_1}{z_1+z_2}\,\frac{\Delta z_1}{z_1}+\frac{z_2}{z_1+z_2}\,\frac{\Delta z_2}{z_2}\\&=\frac{z_1}{z_1+z_2}\,\delta z_1+\frac{z_2}{z_1+z_2}\,\delta z_2.\end{aligned}$$

Im Fall $z_i>0$ $(i=1, 2)$ sind

$$\theta:=\frac{z_1}{z_1+z_2} \quad \text{und} \quad \frac{z_2}{z_1+z_2}=1-\theta$$

positiv. Ein Vergleich von (12) mit dem Streckenbegriff der analytischen Geometrie zeigt dann, daß $\delta(z_1+z_2)$ innerer Punkt des von δz_1 und δz_2 berandeten Intervalls ist.

Unter der Voraussetzung, daß z_1, z_2 gleiches Vorzeichen haben, folgt aus (12) die Abschätzung

$$|\delta(z_1+z_2)|\leqq\max(|\delta z_1|, |\delta z_2|). \tag{13}$$

Aufgabe 5. Man beweise die den Beziehungen (11), (12) und (13) entsprechenden Aussagen für k $(k>2)$ Summanden.

Subtraktion. Für den absoluten Fehler gilt

$$\square(z_1-z_2)=\square z_1+\square z_2. \tag{14}$$

Der Beweis entspricht vollständig dem von (11).

Bemerkenswert und von weitreichenden Konsequenzen für das praktische Rechnen ist das Verhalten des relativen Fehlers einer Differenz annähernd gleicher Operanden. z_1, z_2 mögen gleiches Vorzeichen haben, und es sei $z_1\neq z_2$. In diesem Fall gilt

$$\begin{aligned}\delta(z_1-z_2)&=\frac{z_1^*-z_2^*-(z_1-z_2)}{z_1-z_2}=\frac{z_1+z_2}{z_1-z_2}\left(\frac{\Delta z_1}{z_1+z_2}-\frac{\Delta z_2}{z_1+z_2}\right)\\&=\frac{z_1+z_2}{z_1-z_2}\left(\frac{z_1}{z_1+z_2}\,\delta z_1-\frac{z_2}{z_1+z_2}\,\delta z_2\right).\end{aligned} \tag{15}$$

Aus (15) folgt für annähernd gleiche Werte von z_1 und z_2, daß betragsmäßig die

Größenordnung des relativen Fehlers der Differenz die der relativen Fehler der Operanden um ein Vielfaches übersteigen kann.

Betrachten wir etwa die Berechnung der trigonometrischen Funktionen $\sin z$, $\cos z$ für große Argumentwerte z, wobei es erforderlich ist, diese periodisch auf das Intervall $-\frac{\pi}{2} \leqq z \leqq \frac{\pi}{2}$ zu reduzieren. Beispielsweise[1]) soll der Wert von $\sin z$ für ein Argument bestimmt werden, von dem die Näherung $z^* = 314159{,}3$ gegeben ist. Dabei sei $|z^* - z| \leqq 10^{-1}$; als Näherung für die zur Argumentreduktion benötigte Größe $p := 10^5 \cdot \pi$ benutzen wir den Wert $p^* := 314159{,}27$ mit acht gültigen Ziffern. Es ist

$$|\delta z| \leqq \frac{10^{-1}}{3 \cdot 10^5} = \frac{1}{3} \cdot 10^{-6}$$

und auf Grund von (7)

$$|\delta p| \leqq \frac{1}{3} \cdot 10^{-7}.$$

In dem Näherungswert $d^* := z^* - p^* = 0{,}03$ von $d := z - p$ ist die Ziffer 3 nicht sicher und deshalb

$$|d^* - d| > \frac{1}{2} \cdot 10^{-2}.$$

Ferner ist

$$|d| = |z - p| = |z - z^* + z^* - p| \leqq |z - z^*| + |z^* - p| \leqq 10^{-1} + 10^{-1}$$

und folglich

$$|\delta d| > \frac{\frac{1}{2} \cdot 10^{-2}}{2 \cdot 10^{-1}} = \frac{1}{4} \cdot 10^{-1}.$$

Der relative Fehler der Differenz unterscheidet sich also um Größenordnungen vom relativen Fehler der Operanden. Die folgende Tabelle zeigt, welche Ungenauigkeit der Verlust an gültigen Ziffern im Wert von d^* bei der Berechnung von $\sin z$ verursacht:

z	$\sin z = \sin d$
$z^* = 314159{,}3$	0,03
$z^* + 10^{-1}$	0,13
$z^* - 10^{-1}$	−0,7

Während also z mit sechs gültigen Ziffern gegeben ist, sind im Wert von $\sin z$ alle Ziffern unsicher. Aus der Analyse dieses Beispiels kann man die Faustregel ableiten, daß man bei der Berechnung von $\sin z$ und $\cos z$ nur so viele gültige Ziffern erhält, als in z Ziffern nach dem Komma gesichert sind.

[1]) Dieses Beispiel wurde [69] entnommen.

Der Verlust an gültigen Ziffern bei der Subtraktion nahezu gleicher Zahlen kann in einer Rechnung mit Näherungswerten zu sinnlosen Resultaten führen. Die Vermeidung dieses Effektes ist eine der wichtigsten Aufgaben der algorithmischen Organisation einer Abfolge arithmetischer Operationen in einem Computerprogramm.

Multiplikation. Das Produkt zweier Zahlen z_1, z_2 soll durch Multiplikation der Näherungswerte z_1^*, z_2^* approximativ bestimmt werden. Dann gilt für den absoluten Fehler

$$\begin{aligned}\Delta(z_1 \cdot z_2) &= z_1^* \cdot z_2^* - z_1 \cdot z_2 = (z_1 + \Delta z_1)(z_2 + \Delta z_2) - z_1 \cdot z_2 \\ &= z_1 \Delta z_2 + z_2 \Delta z_1 + \Delta z_1 \cdot \Delta z_2. \end{aligned} \tag{16}$$

Mit oberen Schranken $\square z_1$, $\square z_2$ für den absoluten Betrag von Δz_1 bzw. Δz_2 folgt aus (16)

$$|z_1^* \cdot z_2^* - z_1 \cdot z_2| \leqq |z_1| \square z_2 + |z_2| \square z_1 + \square z_1 \cdot \square z_2.$$

Ist $\square z_1$, $\square z_2$ klein gegenüber $|z_1|$ bzw. $|z_2|$, so wird das Glied $\square z_1 \cdot \square z_2$ meist vernachlässigt und

$$\square(z_1 \cdot z_2) := |z_1| \square z_2 + |z_2| \square z_1 \tag{17}$$

als eine obere Schranke für den absoluten Betrag des Produkts genommen. Unter dieser Annahme ist $|\delta z_1|$ und $|\delta z_2|$ klein gegenüber 1, und für den relativen Fehler des Produkts folgt aus (16) nach Division durch $z_1 \cdot z_2$

$$\delta(z_1 \cdot z_2) = \delta z_1 + \delta z_2 + \delta z_1 \cdot \delta z_2, \tag{18}$$

bei Vernachlässigung des Produkts $\delta z_1 \cdot \delta z_2$ also

$$\delta(z_1 \cdot z_2) \approx \delta z_1 + \delta z_2 . \tag{19}$$

Beispiel. Für die Seiten eines Rechtecks wurden durch Messung 5,23 m und 3,37 m ermittelt. Der Betrag des absoluten Meßfehlers sei höchstens 5 mm. Der Wert des Flächeninhalts und die darin gültigen Ziffern sind zu bestimmen.

Die Voraussetzungen zur Anwendung von (19) sind offenbar gegeben. Für δz_1 und δz_2 benutzen wir gemäß (3) die Schranken

$$|\delta z_1| \leqq \frac{5 \cdot 10^{-3}}{5{,}23} < 1{,}0 \cdot 10^{-3},$$

$$|\delta z_2| \leqq \frac{5 \cdot 10^{-3}}{3{,}37} < 1{,}5 \cdot 10^{-3}$$

und erhalten

$$|\delta(z_1 \cdot z_2)| \leqq 0{,}25 \cdot 10^{-2}.$$

Auf Grund von (10) sind im Produkt

$$5{,}23 \cdot 3{,}37 = 17{,}6251$$

der Meßwerte nur die ersten beiden Ziffern gesichert; die unmittelbare Rechnung zeigt jedoch, daß auch noch die folgende 6 gültig ist. Der Flächeninhalt ist also mit

$$F = 17{,}6 \text{ m}^2$$

anzugeben.

Auf Grund von (9), (19) und (10) gewinnt man folgende Regel[1]) zur Beurteilung der Gültigkeit von Ziffern in einem Produkt:

In den Näherungen z_1^* und z_2^* seien n ($n>1$) Ziffern gültig; ζ_1, ζ_2 bedeuten die führenden Ziffern von z_1^* bzw. z_2^*. Dann können $n-2$ Ziffern in dem Produkt $z_1^* \cdot z_2^*$ als gültig betrachtet werden. (20)

Begründung. Auf Grund von (9) und (19) betrachten wir

$$\frac{1}{2}\left(\frac{1}{\zeta_1}+\frac{1}{\zeta_2}\right)g^{1-n}$$

als eine Schranke für den Betrag des relativen Fehlers $\delta(z_1 \cdot z_2)$, die durch

$$\frac{1}{2}g \cdot g^{1-n}\left(=\frac{1}{2}g^{-(n-2)}\right)$$

majorisierbar ist. Damit folgt aus der Bemerkung nach Satz (10), daß im Näherungswert $z_1^* \cdot z_2^*$ für das Produkt $z_1 \cdot z_2$ die ersten $n-2$ wesentlichen Ziffern gültig sind. Im Dezimalsystem wird die Schranke $\frac{1}{2}g^{-(n-2)}$ meist zu grob und – wie im zuvor betrachteten Beispiel – noch eine Ziffer mehr gültig sein.

In diesem Zusammenhang bearbeite man die

Aufgabe 6. Es ist zu zeigen: Die Regel (20) ist im Dezimalsystem noch für ein Produkt von etwa zehn Faktoren anwendbar.

Division. Bei der Berechnung des Quotienten $\frac{z_1}{z_2}$ ($z_1, z_2 \neq 0$) mit Hilfe der Näherungen z_1^* und z_2^* ($z_2^* \neq 0$) erhält man für den relativen Fehler

$$\delta\left(\frac{z_1}{z_2}\right)=\frac{\frac{z_1^*}{z_2^*}-\frac{z_1}{z_2}}{\frac{z_1}{z_2}}=\frac{(z_1+\Delta z_1)\cdot z_2-(z_2+\Delta z_2)\cdot z_1}{z_1 \cdot z_2^*}$$

$$=\frac{\Delta z_1 \cdot z_2-\Delta z_2 \cdot z_1}{z_1 \cdot z_2^*}=(\delta z_1-\delta z_2)\cdot\frac{z_2}{z_2^*} \tag{21}$$

$$=\frac{\delta z_1-\delta z_2}{\frac{z_2+(z_2^*-z_2)}{z_2}}=\frac{\delta z_1-\delta z_2}{1+\delta z_2}.$$

Für kleine Werte von $|\Delta z_2|$ resultiert daraus die Näherungsformel

$$\delta\left(\frac{z_1}{z_2}\right)\approx\delta z_1-\delta z_2\,. \tag{22}$$

[1]) Wir reden hier von einer Regel statt eines Theorems und von einer Begründung statt eines Beweises, weil verschiedene Vernachlässigungen bei den zu betrachtenden Größen statthatten, deren Auswirkung nicht genauer untersucht wurde. Das entspricht dem Vorgehen in der Praxis.

Aus (21) ergibt sich für den absoluten Fehler des Quotienten

$$\Delta\left(\frac{z_1}{z_2}\right)=\frac{1}{z_2^2}\left(\Delta z_1-\frac{z_1}{z_2}\Delta z_2\right). \tag{23}$$

Für die Beurteilung der gültigen Ziffern in einem Quotienten resultiert aus (22) die Regel:

Sind n Ziffern in den Näherungen z_1^*, z_2^* gültig, so kann man im Quotienten $\frac{z_1^*}{z_2^*}$ die ersten $n-2$ Ziffern als gültig betrachten, im Dezimalsystem sogar die ersten $n-1$, sofern ζ_1 und ζ_2 (führende Ziffern in z_1^* bzw. z_2^*) größer als 1 sind. (24)

Die Begründung von (24) sei dem Leser als Übung empfohlen.

2.5.4. Wir knüpfen an 2.5.1. an und erörtern auf Grund der Ergebnisse von 2.5.3. die *Rundungsfehler*, die bei arithmetischen Operationen in einer EDVA auftreten können.

Bei *Festkommarechnungen* verursachen Addition und Subtraktion – wenn man die Operanden als exakt ansieht – keine Rundungsfehler, wohl aber kann der zulässige Zahlbereich überschritten werden. Sind die Operanden Näherungswerte z_1^* und z_2^*, so gewinnt man nach (11) und (14) durch Addition von Betragsschranken für deren absoluten Fehler eine Schranke für den absoluten Fehler von Summe bzw. Differenz.

Bei der Multiplikation ergeben sich $2t$-stellige Produkte; wie in 2.5.2. wollen wir annehmen, daß der Akkumulator des Rechenwerks ein Zahlspeicher doppelter Wortlänge ist und diese $2t$ Stellen dort tatsächlich verfügbar sind. Bei Rundung auf t Stellen werden auf Grund von (16) die bei den Operanden kumulierten Rundungsfehler unter Umständen gedämpft. Man bedenke, daß die Beträge der Operanden kleiner als Eins sind.

Bei der Division kann wieder eine Überschreitung des Zahlbereichs eintreten; ist der Betrag des Divisors klein, so muß dies auch für den Dividenden gelten, wenn $\left|\frac{z_1}{z_2}\right|<1$ sein soll. Diese gelegentlich als 0/0-Fall charakterisierte Situation ist numerisch kritisch: Bereits akkumulierte Rundungsfehler erfahren auf Grund von (23) bei kleinem $|z_2^*|$ eine entsprechende Vervielfachung.

Zur Erläuterung einiger Effekte bei *Gleitkommarechnung* betrachten wir (ohne Berücksichtigung des Vorzeichens) beispielsweise vierstellige Mantissen- und einstellige Exponentenwerte. Bei der Produktbildung sind die Mantissen der Operanden zu multiplizieren, die Exponenten zu addieren. Auf Grund der in 2.5.1. gewählten Normierung können im Mantissenprodukt führende Nullen auftreten, die bei der Rundung auf t Stellen einen Verlust an wesentlichen Ziffern verursachen würden. Um dies zu vermeiden, wird eine sogenannte *Normalisierung* vorgenommen, bei welcher die höchste Mantissenstelle mit der führenden Ziffer des

im Akkumulator stehenden Produkts besetzt wird und die folgenden dort verfügbaren wesentlichen Ziffern nachlaufen. Der Exponent wird dieser Verschiebung angepaßt. Zur Erläuterung folgendes Beispiel:

	Mantisse	Exponent
3,142	3 1 4 2	1
×		
2,718	2 7 1 8	1
=		
8,540	8 5 4 0	1
	(nach Normalisierung)	

Die Quotientenbildung führt man auf die Division der Mantissen und Subtraktion der Exponenten zurück. Dabei kann der Mantissenquotient m zunächst außerhalb des Bereichs $|m|<1$ liegen. In diesem Fall findet wieder eine Normalisierung in der Weise statt, daß die führende Ziffer von m die höchste Stelle der Resultatmantisse besetzt.

Addition und Subtraktion werden bei Gleitkommaarithmetik in einem Computer nach Angleichung der Exponenten auf den Wert des größeren mit den Mantissen ausgeführt; die Operanden sind dabei als normalisiert anzunehmen.

Beispiel:

	Mantisse	Exponent
314,2	3 1 4 2	3
+		
2,718	2 7 1 8	1
	0 0 2 7	3
	(nach Exponentenerhöhung)	
=		
316,9	3 1 6 9	3

Bei der Addition und Subtraktion von Zahlen ungleichen bzw. gleichen Vorzeichens können im Ergebnis zunächst führende Nullen auftreten, die wieder durch Normalisierung beseitigt werden. Dieser automatisch ablaufende Vorgang verdeckt nur den Effekt der Auslöschung gültiger Ziffern bei der Subtraktion annähernd gleicher Zahlen; an der in 2.5.3. erörterten Erhöhung des relativen Fehlers der Differenz ändert sich nichts.

In 2.5.3. wurde untersucht, mit welchem Fehler das Resultat einer mit Näherungswerten z_1^*, z_2^* ausgeführten arithmetischen Operation[1]) $z_1^* \circ z_2^*$ gegenüber dem Wert $z_1 \circ z_2$ behaftet ist. Dabei haben wir angenommen, daß $z_1^* \circ z_2^*$ exakt gebildet wird. Wenn die Berechnung maschinell erfolgt, muß man zusätzlich beachten, daß beim Runden auf t Mantissenstellen $z_1^* \circ z_2^*$ durch einen Wert z_{12}^{**} ersetzt wird, so daß etwa für den relativen Fehler

$$\delta_r(z_1 \circ z_2) = \frac{z_{12}^{**} - (z_1 \circ z_2)}{z_1 \circ z_2} \tag{25}$$

anzusetzen ist; der Index r soll dabei die Berücksichtigung des Rundungsfehlers zum Ausdruck bringen, während wir weiterhin mit $\delta(z_1 \circ z_2)$ die Größe

$$\frac{z_1^* \circ z_2^* - (z_1 \circ z_2)}{z_1 \circ z_2}$$

bezeichnen. Es ist

$$\begin{aligned} \delta_r(z_1 \circ z_2) &= \frac{z_{12}^{**} - (z_1^* \circ z_2^*) + (z_1^* \circ z_2^*) - (z_1 \circ z_2)}{z_1 \circ z_2} \\ &= \varepsilon_r + \delta(z_1 \circ z_2) \end{aligned} \tag{26}$$

mit

$$\varepsilon_r = \frac{z_{12}^{**} - (z_1^* \circ z_2^*)}{z_1 \circ z_2}.$$

Im Dezimalsystem hat man die Abschätzung

$$|\varepsilon_r| \leqq \frac{\frac{1}{2} \cdot 10^{-t}}{|m|},$$

wobei m die im Bereich $|m| < 1$ liegende, aus den ersten t Dezimalziffern von $z_1 \circ z_2$ gebildete Mantisse dieser Größe ist. Durch Erweiterung mit 10 erhält man

$$|\varepsilon_r| \leqq 5 \cdot 10^{-t}.$$

Damit ist

$$|\delta_r(z_1 \circ z_2)| \leqq |\delta(z_1 \circ z_2)| + 5 \cdot 10^{-t}. \tag{27}$$

Wir wollen die Auswirkung der Rundung auf Grund von (27) am Beispiel der Berechnung einer Summe untersuchen.[2]) Betrachten wir etwa $s = (a + (b + (c + d)))$ in der durch die Klammerung bestimmten Abfolge der Rechenschritte; die Daten a, b, c, d seien positiv und der Einfachheit halber als exakt angenommen. Dann ist (vgl. (12))

$$\delta(c + d) = \frac{c}{c+d}\,\delta c + \frac{d}{c+d}\,\delta d = 0$$

[1]) $\circ$ steht für das Operationszeichen einer der vier Speziesrechnungen.

[2]) Diese Betrachtung schließt sich der Darstellung in [57] an.

und

$$|\delta_r(c+d)| \leqq 5 \cdot 10^{-t}.$$

Weiter hat man nun in (27) auf der rechten Seite

$$\delta(b+(c+d)) = \frac{b}{b+(c+d)}\,\delta b + \frac{c+d}{b+(c+d)}\,\delta_r(c+d)$$
$$= \frac{c+d}{b+(c+d)}\,\delta_r(c+d)$$

zu betrachten, so daß

$$|\delta_r(b+(c+d))| \leqq \frac{c+d}{b+(c+d)}\,5 \cdot 10^{-t} + 5 \cdot 10^{-t}$$

ist. Schließlich ergibt sich

$$|\delta_r(a+(b+(c+d)))| \leqq \frac{b+(c+d)}{a+(b+(c+d))}\left[\frac{c+d}{b+(c+d)}\,5 \cdot 10^{-t} + 5 \cdot 10^{-t}\right] + 5 \cdot 10^{-t}$$
$$= \frac{a+2b+3c+3d}{a+b+c+d}\,5 \cdot 10^{-t}.$$

Sind allgemein $a_1, a_2, \ldots, a_n$ positive, mit t Mantissenstellen exakt gegebene Operanden und wird deren Summe s nach dem Algorithmus des PAP der Abb. 2.34 gebildet, so gilt

$$|\delta_r(s)| \leqq \frac{a_1+2a_2+3a_3+\cdots+(n-1)a_{n-1}+(n-1)a_n}{s}\,5 \cdot 10^{-t}. \tag{28}$$

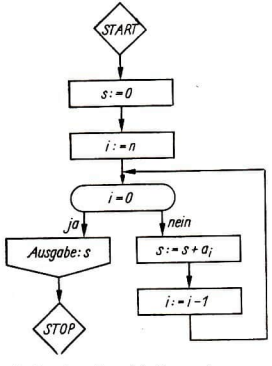


Abb. 2.34

Der Beweis ist im Anschluß an den zuvor betrachteten Spezialfall leicht durch Induktion zu führen. Man erkennt, daß die zuerst in die Rechnung einbezogenen Summanden mit den höchsten Gewichten zu versehen sind, so daß es zweckmäßig

ist, bei der Summation positiver Zahlen mit den kleinsten zu beginnen. Die in Tabelle 2.7 angegebenen Werte wurden mit achtstelliger Gleitkommaarithmetik nach dem Algorithmus der Abb. 2.34 in der angegebenen und der umgekehrten Reihenfolge summiert. In den Ergebnissen kommt der Einfluß der Summandenanordnung zum Ausdruck.

i	a_i Mantisse	Exponent
0	0,59768412	6
1	0,47132908	5
2	0,83271870	3
3	0,20213314	3
4	0,11778855	2
5	0,92163165	0
6	0,72121981	0
7	0,61827317	0
8	0,12345678	0

$$\sum_{i=0}^{8} a_i = 0{,}64586602 \cdot 10^6$$

$$\sum_{j=0}^{8} a_{8-j} = 0{,}64586603 \cdot 10^6$$

Tabelle 2.7

Aufgabe 7. Man zeige, daß bei der Bildung einer Summe aus vier positiven, mit t Mantissenstellen exakt gegebenen Operanden nach der Vorschrift $s=(a+b)+(c+d)$ in Gleitkommarechnung der durch Rundung auf t Mantissenstellen verursachte relative Fehler des Resultats durch

$$|\delta_r(s)| \leqq \frac{2a+2b+2c+2d}{a+b+c+d}\, 5 \cdot 10^{-t}$$

abgeschätzt werden kann.

Die letzten Betrachtungen lassen erkennen, daß für das Rechnen mit beschränkter Stellenzahl die Gesetze der Arithmetik nicht mehr uneingeschränkt gelten und der Analyse numerischer Verfahren unter diesem Gesichtspunkt erhebliche Bedeutung zukommt. Beispielsweise zeigt schon eine oberflächliche Betrachtung, daß die Bestimmung von Polynomwerten nach dem Hornerschen Schema Vorteile gegenüber der unmittelbar durch den Polynomausdruck gegebenen Berechnungsweise bietet. Zunächst ist festzustellen, daß diese bereits hinsichtlich der Anzahl der auszuführenden arithmetischen Operationen ungünstiger ist: $2n-1$ Multiplikationen bei einem Polynom n-ten Grades gegenüber n beim Hornerschen Schema. Oft wird der Horner-Algorithmus zur Partialsummenberechnung von Potenzreihenentwicklungen benutzt. Wenn – wie es häufig der Fall ist – die Beträge der Koeffizienten monoton gegen Null streben, wird bei der Abarbeitung des Schemas auch der oben empfohlenen Summandenanordnung entsprochen.

Neben den in diesem Abschnitt erörterten Rundungsfehlern, die auch als *numerische Fehler* bezeichnet werden, hat das bei der mathematischen Modellierung einer Aufgabe gewählte Verfahren wesentlichen Einfluß auf die Genauigkeit der Lösung. Zum Beispiel werden wir in Kapitel 4 Methoden zur näherungsweisen numerischen Differentiation und Integration betrachten, die darauf beruhen, daß man

die Funktion durch ein Interpolationspolynom ersetzt und für dieses die gesuchten Ableitungen bzw. Integrale bestimmt. Dabei entsteht im allgemeinen auch bei exakter Auswertung der einschlägigen Formeln eine Abweichung von der Lösung der ursprünglich gestellten Aufgabe, die gemeinhin als *Verfahrensfehler* bezeichnet wird.

Natürlich kommt es auch auf die Genauigkeit der Eingangsdaten einer Rechnung an, die etwa in Form von Meßwerten gegeben sein mögen. Die hierbei auftretenden sogenannten *eingangsbedingten Fehler* können jedoch in vielen Betrachtungen technisch wie Rundungsfehler behandelt werden.

3. Einführung in die Programmiersprache ALGOL 60

3.1. Grundwissen

3.1.1. Zur syntaktischen Beschreibung der Sprachstruktur

Der Erschließung eines großen Benutzerkreises der maschinellen Rechentechnik und der Entlastung von geistiger Routinearbeit beim Programmieren war die um das Jahr 1960 einsetzende Verbreitung sogenannter problemorientierter Programmiersprachen sehr förderlich. Diese ermöglichen eine vom Automaten weitgehend unabhängige Formulierung von Programmen und erleichtern deren Austausch zwischen verschiedenen Rechenzentren. Die Übertragung in die dem speziellen Rechner verständliche Maschinensprache erfolgt automatisch mittels eines *Compiler* genannten Übersetzungsprogramms.

Eine für das wissenschaftliche Rechnen wichtige problemorientierte Sprache ist ALGOL 60. Sie wurde von einem internationalen Gremium im wesentlichen während der Jahre 1958–1960 erarbeitet. Wir schicken ihrer Beschreibung einige allgemeine Bemerkungen zur Theorie formaler Sprachen voraus.

Eine (formale) Sprache L über einem Alphabet Σ ist eine Teilmenge von Σ^*, der Menge aller Wörter über Σ. Um eine solche auszuzeichnen, bedienen wir uns einer von J. W. Backus [6] stammenden Definitionstechnik, bei welcher die in L zulässigen Texte (= Wörter) mit Hilfe sogenannter *metalinguistischer Begriffe* beschrieben werden. Dabei werden folgende Zeichen benutzt, von denen zu fordern ist, daß sie nicht zum Alphabet gehören:

Symbole der Backus-Notation	Symbolbedeutung
::=	Definitionszeichen
|	Oder-Zeichen
⟨ ⟩	Klammern

Das Wesentliche der *Backussche Normalform* genannten Sprachbeschreibung wollen wir uns zunächst an zwei einfachen Beispielen klar machen:

1. Es sei $\Sigma = \{a, b, c, +, -, \times, (,)\}$ und L die Menge der Wörter über Σ, die sich als ganzrationale Ausdrücke in den Variablen a, b und c interpretieren lassen.

Nennen wir jede derselben kurz ⟨Formel⟩, dann ist das ein solcher metalinguistischer Begriff, der hier zur Charakterisierung von L ausreicht. Genauer gesagt wird mit Hilfe der Klammerung ein *Symbol für diesen Begriff* aus Buchstaben des lateinischen Alphabets gebildet. Wir werden auf die Syntax formaler Sprachen in MfL Bd. 10 zurückkommen und hier mit ⟨. . .⟩ weiterhin den zugehörigen metalinguistischen Begriff ansprechen. Welche Wörter über Σ den Begriffsumfang von ⟨Formel⟩ ausmachen, wird in der Backusschen Normalform so ausgedrückt:

⟨Formel⟩ ::= *a*|*b*|*c*|(⟨Formel⟩+⟨Formel⟩)|(⟨Formel⟩−⟨Formel⟩)
|(⟨Formel⟩×⟨Formel⟩)

Hiernach ist *a* oder *b* oder *c* eine Formel, und wenn Wörter w_1, w_2 über Σ schon als Formeln erkannt sind, ist auch die Zeichenreihe (w_1+w_2) eine Formel; Entsprechendes gilt für die Verknüpfung mit − und ×. Offenbar werden auf diese Weise alle richtig gebildeten ganzrationalen Ausdrücke in *a*, *b* und *c* und nur diese bestimmt, wenn wir von den üblichen Vereinbarungen über das Einsparen von Klammern einmal absehen. Wie im Beispiel sind die Definitionen metalinguistischer Begriffe meist induktiv, d. h., der zu definierende Begriff erscheint auf beiden Seiten des Zeichens ::= (man vergleiche die Definition der Fakultät).

2. Über dem Alphabet $\Sigma = \{x, \blacktriangledown, (,), \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$ wollen wir diejenigen Zeichenreihen charakterisieren, die sich als Ausdrücke der Aussagenlogik interpretieren lassen. Zunächst definieren wir den Begriff ⟨Aussagenvariable⟩ durch

⟨Aussagenvariable⟩ ::= *x*|⟨Aussagenvariable⟩ ▼.

Danach sind die Wörter x, $x\blacktriangledown$, $x\blacktriangledown\blacktriangledown$, . . . Aussagenvariable, für die man abkürzend $x, x_1, x_2, \ldots$ schreiben könnte. Nun ist der Begriff ⟨Ausdruck⟩ durch

⟨Ausdruck⟩ ::= ⟨Aussagenvariable⟩|¬⟨Ausdruck⟩
|(⟨Ausdruck⟩∧⟨Ausdruck⟩)|(⟨Ausdruck⟩∨⟨Ausdruck⟩)
|(⟨Ausdruck⟩⇒⟨Ausdruck⟩)|(⟨Ausdruck⟩⇔⟨Ausdruck⟩)

bestimmbar. Auf Grund dieser induktiven Definition kann in endlich vielen Schritten entschieden werden, ob ein Wort über Σ ein (vollständig geklammerter) Ausdruck der Aussagenlogik ist. Beispiele dazu findet man etwa in [3].

Das ALGOL-Alphabet setzt sich aus vier Arten von Symbolen zusammen: *Buchstaben*, *Ziffern*, *logischen Werten* und sogenannten *Begrenzern*, die als einelementige Wörter in der Backusschen Normalform wie folgt definiert sind:

⟨Buchstabe⟩ ::= *A*|*B*|*C*|*D*|*E*|*F*|*G*|*H*|*I*|*J*|*K*|*L*|*M*|*N*|*O*|*P*|*Q*|*R*|*S*|*T*|*U*|*V*|*W*|*X*|*Y*|*Z*|
a|*b*|*c*|*d*|*e*|*f*|*g*|*h*|*i*|*j*|*k*|*l*|*m*|*n*|*o*|*p*|*q*|*r*|*s*|*t*|*u*|*v*|*w*|*x*|*y*|*z*|

⟨logischer Wert⟩ ::= **true**|**false**

⟨Ziffer⟩ ::= 0|1|2|3|4|5|6|7|8|9

Der Fettdruck soll **true** und **false** als je ein Zeichen charakterisieren, dessen Gestalt prinzipiell belanglos ist. Die Bezugnahme auf Wörter der englischen Sprache

gibt diesen Symbolen jedoch eine gewisse Signifikanz, die ihren Gebrauch erleichtert. Deshalb werden noch weitere Zeichen des ALGOL-Alphabets auf diese Weise gebildet. Statt des Fettdrucks verwendet man auch andere Schreibungen zur Erzeugung solcher Wortsymbole.

Der Begriff ⟨Begrenzer⟩ erfordert eine induktive Definition. Dabei treten rechts vom Definitionszeichen metalinguistische Begriffe auf, die ihrerseits wieder rekursiv zu bestimmen sind. Das Schema der Backusschen Normalform lautet:

⟨Begrenzer⟩ ::= ⟨Operator⟩|⟨Trennzeichen⟩|⟨Klammer⟩|⟨Vereinbarungszeichen⟩|⟨Spezifikationszeichen⟩
⟨Operator⟩ ::= ⟨arithmetischer Operator⟩|⟨Vergleichsoperator⟩|⟨logischer Operator⟩|⟨Folgeoperator⟩
⟨arithmetischer Operator⟩ ::= +|−|×|/|÷|↑
⟨Vergleichsoperator⟩ ::= <|≦|=|≧|>|≠
⟨logischer Operator⟩ ::= ≡|⊃|∨|∧|¬
⟨Folgeoperator⟩ ::= **goto|if|then|else|for|do**
⟨Trennzeichen⟩ ::= ,|.|₁₀|:|;|:=|⌴ **|step|until|while|comment**
⟨Klammer⟩ ::= (|)|[|]|'|'**|begin|end**
⟨Vereinbarungszeichen⟩ ::= **Boolean|integer|real|array|switch|procedure|own**
⟨Spezifikationszeichen⟩ ::= **string|label|value**

Damit ist die Zeichenmenge des ALGOL-Alphabets vollständig beschrieben. Charakterisieren wir ein Element desselben durch den Begriff ⟨Grundsymbol⟩, so gilt per definitionem

⟨Grundsymbol⟩ ::= ⟨Buchstabe⟩|⟨Ziffer⟩|⟨logischer Wert⟩|⟨Begrenzer⟩

Man kann sich davon überzeugen, daß es 116 Grundsymbole gibt. Die *Syntax* der damit aufzubauenden Programmiersprache, d. h. die Beschreibung der diese ausmachenden Wortmenge (der zulässigen ALGOL-Texte), ist im ALGOL-Bericht [61] in der Backusschen Normalform niedergelegt.

Dieses Kapitel soll an Hand von Beispielen in die wichtigsten Strukturen einführen. Die beigefügten Definitionen metalinguistischer Begriffe dienen im allgemeinen der nachträglichen Präzisierung und sollen dann nur im Zweifelsfall zur Klärung der Zulässigkeit einer Sprachkonstruktion herangezogen werden. Wegen des rekursiven Aufbaus solcher Definitionen treten gelegentlich metalinguistische Begriffe auf, die an der betreffenden Stelle noch nicht erklärt sind oder in diesem Buch überhaupt nicht behandelt werden.

Zunächst sei noch folgendes zum Alphabet bemerkt: Einige Zeichen, wie etwa die Vergleichsoperatoren, entstammen der gebräuchlichen mathematischen Notation und bedürfen keiner Erklärung. Die Zeichen $+, -, \times$ und / sind die arithmetischen Operatoren der Addition, Subtraktion, Multiplikation bzw. Division. Mit $\div$ wird die sogenannte ganzzahlige Division bezeichnet, die nur auf ganzzahlige Operanden angewandt wird. $a \div b$ bedeutet den ganzzahligen Anteil von $\frac{a}{b}$ und

kann mit Hilfe der in 1.3. betrachteten Gauß- bzw. Legendre-Symbolik in der Form

$$a \div b = \operatorname{sgn} \frac{a}{b} \left[\left| \frac{a}{b} \right| \right] = \operatorname{sgn} \frac{a}{b} \operatorname{entier} \left(\left| \frac{a}{b} \right| \right)$$

ausgedrückt werden. ↑ ist das Operationszeichen der Potenzierung; in ALGOL-Texten ist a^b in der Form $a \uparrow b$ zu schreiben.

Die logischen Operatoren unterscheiden sich nur bei ≡ und ⊃ von den sonst in diesem Lehrwerk verwendeten Funktoren: ≡ entspricht dem Äquivalenzzeichen ⇔, und ⊃ entspricht dem Implikationspfeil ⇒. Das Trennzeichen := wird in der Bedeutung gebraucht, die wir in 1.3. erläutert haben. Ausdrücklich sei hier noch einmal auf die Fußnote auf S. 19 hingewiesen.

Die in ALGOL-Programmen auftretenden *Zahlkonstanten* sind im Dezimalsystem auszudrücken; an Stelle des Kommas ist dabei der *Dezimalpunkt* zu verwenden. Neben der allgemein üblichen ist auch die halblogarithmische Schreibweise mit einem Skalenfaktor möglich. Zum Beispiel kann man einen Näherungswert für die Lichtgeschwindigkeit in km/s in der Form $2.99776 \cdot 10^5$ angeben. Die entsprechende ALGOL-Notierung müßte mit Benutzung des Trennzeichens 10 lauten: $2.99776_{10}5$. Den Skalenfaktor bezeichnet man in ALGOL als *Exponententeil*. Gewisse Vereinfachungen der Darstellung sind zulässig. So darf man zum Beispiel die Dezimalzahl 0.791 in der Form .791 schreiben; $_{10}-2$ bedeutet $1 \cdot 10^{-2}$, und $.2_{10}-3$ bedeutet $0.2 \cdot 10^{-3}$. Die vollständige Syntax der Zahldarstellung lautet:

⟨Zahl⟩ ::= ⟨Zahl ohne Vorzeichen⟩|+⟨Zahl ohne Vorzeichen⟩|−⟨Zahl ohne Vorzeichen⟩

⟨Zahl ohne Vorzeichen⟩ ::= ⟨Dezimalzahl⟩|⟨Exponententeil⟩|⟨Dezimalzahl⟩ ⟨Exponententeil⟩

⟨Dezimalzahl⟩ ::= ⟨ganze Zahl ohne Vorzeichen⟩|⟨Dezimalbruch⟩|⟨ganze Zahl ohne Vorzeichen⟩ ⟨Dezimalbruch⟩

⟨Exponententeil⟩ ::= 10 ⟨ganze Zahl⟩

⟨Dezimalbruch⟩ ::= . ⟨ganze Zahl ohne Vorzeichen⟩

⟨ganze Zahl⟩ ::= ⟨ganze Zahl ohne Vorzeichen⟩|+ ⟨ganze Zahl ohne Vorzeichen⟩| − ⟨ganze Zahl ohne Vorzeichen⟩

⟨ganze Zahl ohne Vorzeichen⟩ ::= ⟨Ziffer⟩|⟨ganze Zahl ohne Vorzeichen⟩ ⟨Ziffer⟩

Auf Grund dieser rekursiven Bestimmungsweise kann man von jeder Zeichenreihe des ALGOL-Alphabets in endlich vielen Schritten entscheiden, ob sie eine Zahl darstellt. $314._{10}-2$ ist zum Beispiel keine Zahl, da 314. keine Dezimalzahl ist.

Offensichtlich bedingen die kapazitiven Eigenschaften eines konkreten Rechners, daß nicht jedes ALGOL-Programm übersetzbar ist. Es können nicht beliebig viele Variable, Felder[1]) beliebiger Ausdehnung, Zahlen mit beliebig vielen Stellen und Programme beliebiger Länge in Betracht gezogen werden. Davon abgesehen ist auch eine Einschränkung der Ausdrucksmittel von ALGOL (Bezugssprache) unter Umständen nötig und möglich. Zum Beispiel erfordert der Zeichenvorrat

[1]) Felder werden in 3.1.3. eingeführt.

des Eingabegerätes meist die Bildung eines Grundsymbol-Subsets; andererseits hat sich gezeigt, daß die Programmierung aller praktisch interessierenden Algorithmen mit reduzierten Ausdrucksmitteln möglich ist. Es ist daher in gewissem Umfang vertretbar, daß die mit Rücksicht auf die Leistungsfähigkeit der Maschine zu entwickelnden Compiler einige Sprachkonstruktionen in ALGOL nicht zulassen. Diese Umstände haben dazu geführt, daß eine gegenüber ALGOL 60 eingeschränkte Sprache IFIP SUBSET ALGOL 60 (vgl. [62]) geschaffen wurde, aus der sich verschiedene maschinengebundene Versionen ableiten. Bezüglich der Variante für die EDVA ROBOTRON 300 wird dazu noch etwas in 3.4. gesagt.

3.1.2. Orientierende Betrachtungen zum Aufbau eines ALGOL-Programms

Um eine erste Vorstellung von der Struktur eines ALGOL-Programms zu vermitteln, betrachten wir den folgenden Text dieser Sprache, der den Euklidischen Algorithmus beschreibt:

```
begin
  integer m, n, x, y, q, r;
  Lies (m, n);
  if m ≧ n then begin x:=m; y:=n end
           else begin x:=n; y:=m end;
L:q:=entier(x/y);
  r:=x−q×y;
  if r>0 then begin x:=y; y:=r;
  goto L      end;
  Drucke(y)
end
```

Jeder ALGOL-Text ist ein Wort über dem Alphabet der Grundsymbole, dessen Zeichen zeilenweise von links nach rechts aufeinanderfolgen. Es kommt allein auf die Reihenfolge der Symbole an; eine Gliederung bei ihrer Niederschrift hat nur für die externe Lesbarkeit der Programme Bedeutung.

Wie im Beispiel stehen am Anfang und Ende jedes ALGOL-Programms die Grundsymbole **begin** bzw. **end**, die darin allgemein die Funktion von Anweisungsklammern haben und die Abarbeitung in analoger Weise beeinflussen wie Klammern in arithmetischen Ausdrücken deren Berechnung.

Alle in einem ALGOL-Programm auftretenden Größen müssen bezeichnet sein. Als Bezeichnung sind Buchstaben zugelassen, denen weitere Buchstaben oder Ziffern folgen können. Die exakte Definition in der Backusschen Normalform lautet

⟨Bezeichnung⟩ ::= ⟨Buchstabe⟩|⟨Bezeichnung⟩ ⟨Buchstabe⟩|⟨Bezeichnung⟩ ⟨Ziffer⟩

Für die Übersetzung in die Maschinensprache ist es erforderlich, ALGOL-Größen in gewisse Kategorien einzuteilen und diese (mit Ausnahme der sogenannten Mar-

ken) entsprechend zu *vereinbaren*. In einem ALGOL-Programm erscheint jede Bezeichnung, die keine Marke ist, erstmalig in einer solchen Vereinbarung.

In unserem Beispiel treten Variable auf, deren in 1.3. eingeführte Bezeichnung in ALGOL zulässig ist und übernommen wurde. Genauer gesagt, handelt es sich dabei um sogenannte *einfache Variable* gemäß folgender syntaktischer Bestimmung in der Backusschen Normalform:

⟨einfache Variable⟩ ::= ⟨Variablenbezeichnung⟩
⟨Variablenbezeichnung⟩ ::= ⟨Bezeichnung⟩

m, n, x, y, q und r nehmen nur ganzzahlige Werte an und werden daher als vom Typ **integer** vereinbart. Dies erfolgt am Anfang des Programms im *Vereinbarungsteil* und veranlaßt den Compiler, für jede vereinbarte Variable Speicherraum zu reservieren, der zur Aufnahme von Werten derselben geeignet ist. Im Beispiel entspricht dem die Struktur

integer m, n, x, y, q, r;

Hinter einem Vereinbarungssymbol können also die Bezeichnungen mehrerer Größen des betreffenden Typs, durch Komma getrennt, in einer Liste erscheinen. Das Komma ist in ALGOL nur als Listentrennzeichen zu verwenden; Dezimalzahlen sind – wie schon bemerkt – mit einem Punkt zu notieren. Hinsichtlich des Typs verschiedene Vereinbarungen werden durch Semikolon getrennt, desgleichen Anweisungen voneinander und Vereinbarungen von Anweisungen.

Der *Anweisungsteil* beginnt im Beispiel mit dem Einlesen der Werte von m und n, die dabei über ein peripheres Eingabegerät dem Rechner vermittelt und auf den für m und n reservierten Plätzen gespeichert werden. Zur Fixierung der Anschauung stelle man sich etwa vor, daß m und n in codierter Form auf einem Lochstreifen erfaßt sind. Ein- und Ausgabevorgänge sind automatenspezifisch und werden im ALGOL-Bericht nicht behandelt. Wir verzichten hier auf die Erörterung später erfolgter Standardisierungen und verwenden eine Anweisung *Lies* bzw. *Drucke*, der, in runde Klammern eingeschlossen, die Liste der ein- bzw. auszugebenden Größen folgt. Es sei dabei vereinbart, die Reihenfolge dem realen Vorgang anzupassen.

Der dann folgenden Verzweigung des Flußdiagramms entspricht im ALGOL-Programm die *bedingte Anweisung* zweiter Art

if $m \geqq n$ **then begin** $x := m$; $y := n$ **end**
else begin $x := n$; $y := m$ **end**

Diese enthält eine Wenn-Klausel, das ist die in die Grundsymbole **if** und **then** eingeschlossene Aussage $m \geqq n$. Ist diese wahr, so wird die auf **then** folgende Anweisung abgearbeitet und die auf **else** folgende übergangen, sonst in der umgekehrten Weise verfahren. Da die ALGOL-Syntax nach **then** und **else** nur jeweils eine Anweisung zuläßt, war es im Beispiel erforderlich, die tatsächlich auszuführenden Befehle

$x := m; y := n$ bzw. $x := n; y := m$

durch die Klammern **begin** und **end** zu einer *Verbundanweisung* zusammenzuschließen.

Die nächste Anweisung des PAP konnte (von der Marke L abgesehen; siehe unten) unverändert in das ALGOL-Programm übernommen werden. *entier* ist eine von neun Standardfunktionen, die fester Bestandteil des Compilers sind und in Programmen keiner Vereinbarung bedürfen. Ihre Bezeichnungen sind reserviert und können nicht für andere ALGOL-Größen benutzt werden. Tabelle 3.1 enthält die ALGOL-Notation der Funktionswerte und erläutert ihre Bedeutung in der üblichen mathematischen Bezeichnungsweise. Das Argument der Standardfunktionen ist vom Typ **real** und mit Ausnahme von *entier* und *sign* auch der Funktionswert. Wir merken hier an, daß einer Variablen vom Typ **real** ein Wert vom Typ **integer** zugewiesen werden darf. Vor Zuweisung eines **real**-Wertes a an eine Variable vom Typ **integer** findet eine Approximation durch die nächstgelegene ganze Zahl statt. Dies geschieht mit Hilfe der Standardfunktion *entier*, und zwar durch Bildung des Wertes $entier(a+0.5)$.

ALGOL-Notation	übliche mathematische Bezeichnungsweise
$abs(x)$	$\|x\|$
$arctan(x)$	arctan x (Hauptwert)
$cos(x)$	cos x
$entier(x)$	$[x]$ (größte ganze Zahl, die kleiner oder gleich x ist)
$exp(x)$	e^x
$ln(x)$	ln x
$sign(x)$	sgn x ($+1$ für $x>0$, 0 für $x=0$, -1 für $x<0$)
$sin(x)$	sin x
$sqrt(x)$	$\sqrt{x}$

Tabelle 3.1

Durch die folgende Anweisung erhält r den Wert des arithmetischen Ausdrucks $x-q\times r$ zugewiesen, in dem $\times$ den Multiplikationsoperator in ALGOL bezeichnet. Die Berechnung von arithmetischen Ausdrücken erfolgt in linearer Anordnung von links nach rechts fortschreitend, wobei die üblichen Klammer- und Vorrangregeln bezüglich der Operationen beachtet werden. Als Klammern in arithmetischen Ausdrücken sind nur die Symbole (und) zulässig. Ausdrücklich sei darauf hingewiesen, daß alle Operatoren gesetzt werden müssen. Multiplikationszeichen können also nicht – wie gelegentlich sonst üblich – eingespart werden; ab wird vom Compiler als Name einer ALGOL-Größe und nicht als Produkt $a\times b$ interpretiert. Es ist zu beachten, daß in einem arithmetischen Ausdruck arithmetische Operatoren nicht unmittelbar aufeinanderfolgen dürfen. Deshalb müssen beim Auftreten von Operanden mit einem Minuszeichen erforderlichenfalls runde Klammern gesetzt werden. Für das Produkt von $-a$ und b kann man $-a\times b$ schreiben; bei Vertauschung der Faktoren muß jedoch $b\times(-a)$ notiert werden. Dies folgt aus der Syntax arithmetischer Ausdrücke, die unsere inhaltlichen Vorstellungen von diesem Begriff, auf die bisher allein Bezug genommen wurde, präzisiert:

⟨arithmetischer Ausdruck⟩ ::= ⟨einfacher arithmetischer Ausdruck⟩|⟨Wenn-Klausel⟩ ⟨einfacher arithmetischer Ausdruck⟩ **else** ⟨arithmetischer Ausdruck⟩

⟨Wenn-Klausel⟩ ::= **if** ⟨logischer Ausdruck⟩ **then**

⟨einfacher arithmetischer Ausdruck⟩ ::= ⟨Term⟩|⟨Summationsoperator⟩ ⟨Term⟩|⟨einfacher arithmetischer Ausdruck⟩ ⟨Summationsoperator⟩ ⟨Term⟩

⟨Summationsoperator⟩ ::= +|−

⟨Term⟩ ::= ⟨Faktor⟩|⟨Term⟩ ⟨Multiplikationsoperator⟩ ⟨Faktor⟩

⟨Multiplikationsoperator⟩ ::= ×|/|÷

⟨Faktor⟩ ::= ⟨Elementarausdruck⟩|⟨Faktor⟩↑⟨Elementarausdruck⟩

⟨Elementarausdruck⟩ ::= ⟨Zahl ohne Vorzeichen⟩|⟨Variable⟩|⟨Funktion⟩|(⟨arithmetischer Ausdruck⟩)

Dieses Schema ist insofern unvollständig, als wir die Begriffe ⟨logischer Ausdruck⟩, ⟨Variable⟩ und ⟨Funktion⟩ noch nicht erklärt haben. Variable können einfache Variable sein (siehe oben) oder indizierte Variable. Die letzten werden in 3.1.3. zusammen mit den Feldern eingeführt. Den Begriff ⟨Funktion⟩ können wir erst in Verbindung mit den Prozeduren (vgl. 3.3.) erörtern. Diese verwirklichen in ALGOL eine Unterprogrammtechnik und ermöglichen auch die Einbeziehung von Programmen in der Maschinensprache in ALGOL-Programme. Spezielle Funktionen sind die Standardfunktionen, d. h., Elementarausdrücke können zum Beispiel sein *abs*(*x*), *entier*(*x*), *sin*(*x*) usw.

Der Begriff ⟨logischer Ausdruck⟩ deckt sich im Spezialfall des einfachen logischen Ausdrucks wesentlich mit dem des Ausdrucks der Aussagenlogik; als Aussagenvariable stelle man sich dabei die Menge der Symbole vor, die nach der ALGOL-Syntax Variable sind. Diese nehmen zwei – inhaltlich als „wahr" bzw. „falsch" zu interpretierende – Werte an, die in ALGOL durch **true** und **false** ausgedrückt werden; sie sind als vom Typ **Boolean** zu vereinbaren. Solche Variable können auch indiziert sein, d. h., es sind Felder vom Typ **Boolean** möglich (vgl. 3.1.3.).

Im ALGOL-Programm erfolgt die Wertbestimmung eines logischen Ausdrucks gemäß 2.4.1. nach vorheriger Aktualisierung der Variablen durch logische Werte. Dabei können für Variable gewisse Aussagen – sogenannte *Vergleiche* – substituiert werden, die mit **true** bzw. **false** in die Wertbestimmung eingehen, je nachdem, ob die darin erfaßten arithmetischen Sachverhalte zutreffen oder nicht. Die Definition des Vergleichs ist:

⟨Vergleich⟩ ::= ⟨einfacher arithmetischer Ausdruck⟩ ⟨Vergleichsoperator⟩ ⟨einfacher arithmetischer Ausdruck⟩

(einfacher arithmetischer Ausdruck siehe oben). Die ALGOL-Syntax läßt in logischen Ausdrücken auch **true** und **false** als „logische Konstanten" zu.

Wie bei den arithmetischen Ausdrücken erfolgt die Wertbestimmung in der Reihenfolge von links nach rechts unter Berücksichtigung von Klammern. Diese können auf Grund bekannter Konventionen teilweise eingespart werden. Das Defi-

nitionsschema des Begriffs ⟨logischer Ausdruck⟩ ist:

⟨logischer Ausdruck⟩ ::= ⟨einfacher logischer Ausdruck⟩ | ⟨Wenn-Klausel⟩ ⟨einfacher logischer Ausdruck⟩ **else** ⟨logischer Ausdruck⟩
⟨einfacher logischer Ausdruck⟩ ::= ⟨Implikation⟩|⟨einfacher logischer Ausdruck⟩ ≡⟨Implikation⟩
⟨Implikation⟩ ::= ⟨logischer Term⟩|⟨Implikation⟩ ⊃⟨logischer Term⟩
⟨logischer Term⟩ ::= ⟨logischer Faktor⟩|⟨logischer Term⟩ ∨⟨logischer Faktor⟩
⟨logischer Faktor⟩ ::= ⟨logischer Elementarausdruck 2. Art⟩|⟨logischer Faktor⟩ ∧⟨logischer Elementarausdruck 2. Art⟩
⟨logischer Elementarausdruck 2. Art⟩ ::= ⟨logischer Elementarausdruck 1. Art⟩| ¬⟨logischer Elementarausdruck 1. Art⟩
⟨logischer Elementarausdruck 1. Art⟩ ::= ⟨logischer Wert⟩|⟨Variable⟩|⟨Funktion⟩| ⟨Vergleich⟩|(⟨logischer Ausdruck⟩)

(Variable siehe S. 94, Funktion S. 113, Wenn-Klausel S. 90). Der Begriff ⟨Ausdruck⟩ umfaßt den des arithmetischen und des logischen; seine Syntax ist:

⟨Ausdruck⟩ ::= ⟨arithmetischer Ausdruck⟩|⟨logischer Ausdruck⟩|⟨Zielausdruck⟩

Die einfachsten *Zielausdrücke* sind die *Marken*, vergleichbar den Zahlen und logischen Werten in den arithmetischen bzw. logischen Ausdrücken; ihre Verwendung wird bei der weiteren Erörterung unseres Beispiels erklärt.

Bei der Berechnung arithmetischer Ausdrücke für aktuelle Werte der darin vorkommenden Variablen, die nur vom Typ **integer** oder **real** sein können, gelten natürlich die aus der mathematischen Bedeutung der auszuführenden Operationen herzuleitenden Einschränkungen. Darüber hinaus muß man wissen, von welchem Typ die Resultate arithmetischer Operationen sind. Addition, Subtraktion und Multiplikation liefern ein Ergebnis vom Typ **integer**, wenn beide Operanden vom Typ **integer** sind, sonst vom Typ **real**. Bei der Division erhält man stets ein Resultat vom Typ **real**, bei der nur für Operanden vom Typ **integer** erklärten ganzzahligen Division ein Resultat vom Typ **integer**.

Die Potenz $a \uparrow b$ wird in ALGOL auf eine wiederholte Multiplikation zurückgeführt, wenn b vom Typ **integer** ist, und für b vom Typ **real** mit Hilfe der Standardfunktionen *exp* und *ln* gemäß

$$a \uparrow b = exp(b \times ln(a))$$

gebildet. Das entspricht dem üblichen methodischen Vorgehen beim schrittweisen Erweitern des Potenzbegriffs bezüglich der zugelassenen Exponentenbereiche. Damit ist klar, daß

$a \uparrow b$ nicht definiert ist für $a = 0$, $b < 0$ vom Typ **integer** und $a \leqq 0$, b vom Typ **real**.

Mit der Typbestimmung der Werte der Standardfunktionen und der Ergebnisse bei Multiplikation und Division ist so auch der Typ von $a \uparrow b$ festgelegt. Beispielsweise ist $a \uparrow b$ vom Typ **real**, wenn $a \neq 0$ und $b < 0$ vom Typ **integer**, da im vorliegenden Fall $a \uparrow b$ der Wert von $1/(a \times a \times a \times \cdots \times a)$ ($-b$ Faktoren im Nenner) ist.

Darüber hinaus findet in ALGOL eine Wertbestimmung der Potenz noch für folgende Grenzfälle statt, die nicht in den bisherigen Erklärungen enthalten sind:

a	b	$a \uparrow b$
$a \neq 0$	$b = 0$ vom Typ **integer**	1 Typ von a
$a = 0$	$b > 0$ vom Typ **real**	0 Typ **real**

In einem arithmetischen Ausdruck kann eine Zahl als Elementarausdruck enthalten sein. Diese wird als **integer**-Größe behandelt, wenn sie im Sinne der ALGOL-Syntax ganze Zahl ist; sonst stellt sie eine Größe vom Typ **real** dar.

Wir wenden uns wieder dem PAP der Abb. 1.4 zu und betrachten die durch die Bedingung $r = 0$ veranlaßte Verzweigung. Am Alphabetoperator des Algorithmus würde sich nichts ändern, wenn man in das Entscheidungskästchen $r > 0$ einträgt und gleichzeitig den Ja- und Nein-Zweig vertauscht. Diese Auffassung liegt der im ALGOL-Programm an entsprechender Stelle erscheinenden bedingten Anweisung erster Art zugrunde:

Die auf **then** folgende Verbundanweisung wird ausgeführt, wenn die Aussage $r > 0$ der Wenn-Klausel wahr ist, sonst übergangen, und es erfolgt der Übergang zur nächsten Anweisung des Programms, die im Beispiel die letzte ist und die Ausgabe des größten gemeinsamen Teilers veranlaßt.

In der erwähnten Verbundanweisung tritt der Sprungbefehl

goto *L*

auf, der den Zyklus des Flußbildes realisiert. *L* ist dabei eine Marke, welche, durch : getrennt, vor die Anweisung zu setzen ist, die angesprungen und als nächste abgearbeitet werden soll. Von da an wird im PAP wieder der Programmlinie – im ALGOL-Text der linearen Anordnung der Anweisungen – folgend fortgefahren, bis erneut ein Sprungbefehl diese Abfolge unterbricht. Wir erinnern an eine frühere Bemerkung: Marken sind die einzigen ALGOL-Größen, die nicht vereinbart werden müssen; ihre Definition in der Backusschen Normalform lautet:

⟨Marke⟩ ::= ⟨Bezeichnung⟩|⟨ganze Zahl ohne Vorzeichen⟩.

Bei der Erörterung unseres Beispiels haben wir zunächst darauf verzichtet, die Anweisungen betreffenden metalinguistischen Begriffe zu definieren. Der Leser präge sich aber die PAP-Strukturen ein, die im ALGOL-Programm zu einer bedingten Anweisung 1. bzw. 2. Art Anlaß gegeben haben. Im übrigen sei wieder an den in 3.1.1. gegebenen Hinweis zum Gebrauch der Syntax-Schemata erinnert.

Zur Festigung der bisher erworbenen ALGOL-Kenntnisse seien folgende Übungen empfohlen (die größeren Abstände zwischen den auftretenden Zeichenreihen sind signifikant):

Aufgabe 1. Welche der folgenden Zeichenreihen sind Bezeichnungen?

real array 0-8-15 MP71/1 C2H5OH

Welche der folgenden Zeichenreihen sind zulässige Zahldarstellungen?

3.14 10 2.5 27.1828 ×10 − 1 27.1828 10 − 1 π 3,14

Aufgabe 2. Man entscheide, welche der folgenden Zeichenreihen arithmetischer Ausdruck ist:

$-a \times b$	$b \times -a$	H2O + NaCl
$2 \uparrow -3$	$2 \uparrow (-3)$	$-.314\ 10\ 1$

Aufgabe 3. Man schreibe die folgenden arithmetischen Ausdrücke so in der allgemein üblichen mathematischen Notation, daß die Reihenfolge der Rechenoperationen erkennbar ist:

$a / b \times 0$	$a / b - c$	$a / b / c$
$a / (a - b - c)$	$a + b \times c$	$-b \uparrow c$
$a / c \uparrow b$	$a \uparrow b \uparrow c$	

Aufgabe 4. Die aktuellen Werte von a, b, c seien bzw. 1 1 0. Welche der arithmetischen Ausdrücke in Aufgabe 3 sind dafür definiert, wenn man a, b, c einmal als vom Typ **integer**, dann als vom Typ **real** annimmt. Gegebenenfalls bestimme man ihren Wert und dessen Typ.

Aufgabe 5. Man drücke die folgenden trigonometrischen Terme in ALGOL-Notation aus:

$$\sqrt{\frac{1-\cos 2\alpha}{1+\cos 2\alpha}} \qquad \sqrt{2}\sin(45° + \alpha)$$

$$b^2 + c^2 - 2bc\cos\alpha \qquad \frac{\cot\alpha\cot\beta - 1}{\cot\beta + \cot\alpha}$$

Aufgabe 6. Man verifiziere, daß die folgende Zeichenreihe logischer Ausdruck ist, und bestimme dessen logischen Wert unter der Annahme, daß a, b Variable vom Typ **Boolean** mit den aktuellen logischen Werten **false** bzw. **true** sind:

$(6 \times 4 \div 3 > 6 \times (4 \div 3)) \wedge \neg a) \supset (b \supset a)$

3.1.3. Einführung weiterer Sprachelemente

Wir knüpfen an die Berechnung von Polynomwerten nach dem Hornerschen Schema (vgl. 2.2.) an. Häufig entspricht es dem Wesen eines mathematischen Sachverhalts, endlich viele Variable zu einem Ganzen zusammenzufassen und die Position einer bestimmten Variablen in diesem durch Indizes zu kennzeichnen. Ein Beispiel dafür sind die Koeffizienten eines Polynoms. Neben einfach indizierten Größen ist die Verwendung solcher mit mehreren Zeigern zweckmäßig. So entspricht es etwa der Natur der Sache, die Koeffizienten eines linearen Gleichungssystems mit zwei, die Koeffizienten der Gleichung einer Fläche zweiter Ordnung mit drei Indizes auszustatten; ihre Anzahl mag als Dimension des Größensystems verstanden werden. Dem in der Mathematik aus dieser Vorstellung heraus entwickelten Begriff der Matrix ist in ALGOL der des Feldes zuzuordnen. Felder können als Ganzheiten in ALGOL-Programmen auftreten und müssen als solche eine Bezeichnung erhalten und vereinbart werden. Dies geschieht mit dem Grundsymbol **array**, dem der Typ der in dem Feld zusammengefaßten Variablen vorangestellt wird. Zur Abkürzung kann man für **real array** einfach **array** schreiben. In bestimmter Weise sind dann Dimension und Variabilitätsbereiche der Indizes zu kennzeichnen. Zum Beispiel bedeutet

array A[0:5]

ein eindimensionales Feld (einen Vektor) von **real**-Variablen, deren Index von 0 bis 5

läuft, und

integer array *B*[1:3,2:5]

eine Matrix ganzzahliger Variabler, deren Zeilenindex von 1 bis 3 und deren Spaltenindex von 2 bis 5 variiert. Bei mehrfach indizierten Variablen treten zwischen den eckigen Klammern durch Komma getrennt soviel Laufbereiche *i:k* auf, wie Indizes vorhanden sind.

Will man eine bestimmte indizierte Variable eines Feldes in einem Programm – etwa bei der Auswertung eines arithmetischen Ausdrucks – benutzen, so ist diese mit der Feldbezeichnung zu notieren, der, in eckige Klammern eingeschlossen, die sogenannte *Indexliste* zu folgen hat. Diese besteht aus der durch Komma getrennten Folge der Indizes, welche die Position der Variablen im Feld charakterisieren. Bezüglich der Beispiele ist etwa *A*[1] bzw. *B*[3,4] möglich. An Stelle der Indizes können auch arithmetische Ausdrücke auftreten, die zur Bestimmung derselben automatisch ausgewertet werden. Wir beschränken uns darauf, das bezüglich der indizierten Variablen Gesagte syntaktisch zu präzisieren:

⟨indizierte Variable⟩ ::= ⟨Feldbezeichnung⟩ [⟨Indexliste⟩]
⟨Feldbezeichnung⟩ ::= ⟨Bezeichnung⟩
⟨Indexliste⟩ ::= ⟨Indexausdruck⟩|⟨Indexliste⟩, ⟨Indexausdruck⟩
⟨Indexausdruck⟩ ::= ⟨arithmetischer Ausdruck⟩

Wie schon erwähnt, kann der Begriff ⟨Variable⟩ nun wie folgt definiert werden:

⟨Variable⟩ ::= ⟨einfache Variable⟩|⟨indizierte Variable⟩

Die Verwendung der neu eingeführten Sprachelemente in einem Programm sei am Beispiel des Horner-Algorithmus erläutert. Dabei müssen wir uns zunächst auf Polynome eines festen Maximalgrades, etwa $n=5$, beschränken. Für das System der Koeffizienten $a_0, a_1, a_2, \ldots, a_5$ wird das Feld **array** *A*[0:5] vereinbart. Wie in 2.2. dargelegt, benötigen wir weiterhin eine Zählvariable *i* vom Typ **integer** und zwei **real**-Variable *p* und *x*0. Die Bezeichnung der letzteren wurde mit der ALGOL-Syntax in Einklang gebracht. Unter Verwendung von zwei Marken läßt sich der PAP der Abb. 2.8 aus 2.2. ($n=5$) in folgendes ALGOL-Programm übertragen:

```
begin
  integer i; real p,x0; array A[0:5];
  Lies(A,x0);
  p:=A[5];
  i:=1;
LA: if i=6 then goto LB;
  p:=A[5-i]+x0×p;
  i:=i+1;
  goto LA;
LB:Drucke(p)
end
```

Nach den Vereinbarungen beginnt der Anweisungsteil mit dem Einlesen der Polynomkoeffizienten auf das Feld A und des von $x0$ aufzunehmenden Argumentwerts. Nach der Anfangsstellung der Zählvariablen verzweigt sich der PAP. Wenn i den Wert $n+1=6$ erreicht hat, ist die Aussage der Wenn-Klausel wahr, und es erfolgt der Sprung zur letzten Anweisung des Programms, welche die Ausgabe des Polynomwerts p veranlaßt. Sie wurde zu diesem Zweck mit der Marke LB versehen, die zugleich in der Sprunganweisung innerhalb der bedingten Anweisung erster Art

if $i=6$ **then goto** LB

auftritt. Ist $i=6$ nicht zutreffend, so wird der Sprungbefehl übergangen und die folgende Anweisung

$$p:=A[5-i]+x0\times p$$

ausgeführt. Hier tritt der Fall auf, daß die Indexliste der indizierten Variablen einen arithmetischen Ausdruck, nämlich $5-i$ enthält. Weiter erfolgt dann die Erhöhung der Zählvariablen und die Rückkehr zur Verzweigungsstelle durch die Sprunganweisung

goto LA

deren Marke vor die bedingte Anweisung zu setzen ist.

Der Gebrauch des Semikolons in ALGOL-Programmen wurde bereits in 3.1.2. beschrieben. Danach ist weder in einem Programm noch in einer Verbundanweisung vor dem schließenden Grundsymbol **end** ein Semikolon nötig. Die damit zusammenhängenden syntaktischen Fragen werden endgültig mit der Definition des Programmbegriffs geklärt werden. Bereits hier soll aber auf eine Präzisierung bzw. Modifikation der erwähnten Regel hingewiesen werden, die aus der Einbeziehung der *leeren Anweisung* in die ALGOL-Syntax resultiert. Das ist zweckmäßig, weil es auf diese Weise möglich wird, zum Beispiel das Ende eines Programms zu markieren.

L: **end**

würde dann so zu deuten sein: Vor **end** steht die mit der Marke L versehene leere Anweisung. Da eine markierte Anweisung nach der ALGOL-Syntax eine Anweisung ist, muß zwischen L und einer vorangehenden Anweisung bei dieser Sachlage unbedingt ein Semikolon stehen. In jedem Fall kann man sich die leere Anweisung als letzte des Programms denken und vor das schließende Grundsymbol **end** ein Semikolon setzen; beim praktischen Programmieren ist es unter Umständen vorteilhaft, so zu verfahren.

Offensichtlich ist, daß die Benutzung des Programms für Polynome mit einem Grad n kleiner als 5 erfordert, die Koeffizienten $a_5, a_4, \ldots, a_{n+1}$ als Nullen einzugeben.

Wir wollen genauer den eingerahmten Teil des ALGOL-Textes und den in Abb. 3.1 dargestellten entsprechenden Teil des PAP der Abb. 2.8 betrachten. Derartige Flußdiagrammuster treten in Programmablaufplänen sehr häufig auf. Typisiert

sind sie von der in Abb. 3.2 gezeigten Form. Die in Abb. 3.2 dargestellte Programmstruktur kann in einer sogenannten *Laufanweisung* mit einem **step-until**-*Element* zusammengefaßt werden. Diese würde, bezogen auf das Beispiel des Hornerschen Schemas (Abb. 3.1), lauten:

for $i := 1$ **step 1 until** n **do** $p := A[n-i] + x0 \times p$

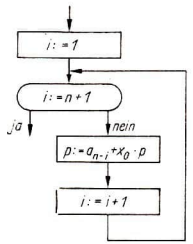


Abb. 3.1

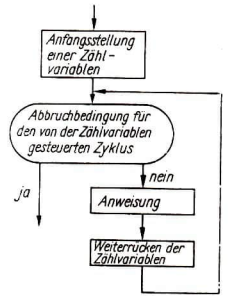


Abb. 3.2

Nach dem Grundsymbol **for** erscheint die Bezeichnung i der Zählvariablen, gefolgt von dem Trennzeichen := und dem **step-until**-Element

1 step 1 until n

in dem die erste 1 den Anfangswert von i, der Wert nach **step** die Schrittgröße des Weiterrückens und der nach **until** im Beispiel dasjenige i bezeichnet, für welches zum letzten Mal die auf **do** folgende Anweisung ausgeführt wird. Ist allgemein a der Anfangs-, z der Endwert der den Zyklendurchlauf kontrollierenden Variablen und h die Schrittgröße, so hat die betrachtete Laufanweisung die Gestalt

for $i := a$ **step** h **until** z **do** | Anweisung | (1)

Soll diese auf das Programmstück der Abb. 3.2 passen, so muß die Testaussage in der Form

$$(i - z) \times sign(h) > 0$$

ausdrückbar sein. In Abb. 3.1 kann man zum Beispiel die Abbruchbedingung $i = n + 1$ durch

$$(i - n) \times sign(1) > 0$$

ersetzen, ohne daß eine Veränderung im Zyklendurchlauf eintritt.

Verschiedene Verallgemeinerungen treten auf: An Stelle von z, a und h sind arithmetische Ausdrücke möglich, und die Laufvariable, ihre Grenzen und die Schrittgröße können vom Typ **real** sein. Besonders im letzten Fall nimmt i in (1)

unter Umständen (Rundungseffekte) den Wert z nicht an. [38] folgend sollte man daher bezüglich z lieber vom Endwert des Laufbereichs statt der Laufvariablen reden. Die Bezeichnung der Laufvariablen ist natürlich beliebig, sofern sie nur in Übereinstimmung mit den im Programm getroffenen Vereinbarungen erfolgt.

Nunmehr kann das Programm zur Berechnung von Polynomwerten nach dem Hornerschen Schema so formuliert werden:

```
begin
  integer i; real p, x0; array A[0:5];
  Lies(A,x0);
  p := A[5];
  for i := 1 step 1 until 5 do p := A[5 − i] + x0 × p;
  Drucke(p);
end
```

Der Übergang zur Anweisung *Drucke*(p) findet statt, ohne daß jetzt noch eine Marke im Programm zu setzen ist. Innerhalb der Laufanweisung ist bei jedem Schritt die Indexrechnung $5-i$ auszuführen, die vermeidbar wird, wenn man i mit der Schrittgröße -1 von 4 bis 0 laufen läßt. Die betrachtete Programmzeile würde dann lauten:

```
for i := 4 step −1 until 0 do p := A[i] + x0 × p;
```

Vor einer syntaktischen Präzisierung des Begriffs der Anweisung sollen weitere Programmbeispiele mit den vermittelten Sprachelementen vertraut machen.

Die bisher betrachteten *Ergibtanweisungen* ordneten – allgemein gesprochen – einer Variablen den Wert eines arithmetischen Ausdrucks zu. Dabei mag man es als eine gewisse Inkonsequenz empfinden, daß nach dessen Berechnung in der Reihenfolge von links nach rechts das Resultat links vom dynamischen Gleichheitszeichen (Ergibtzeichen) := erscheint. Analog kann man Variablen vom Typ **Boolean** den Wert eines logischen Ausdrucks zuweisen. Zur Erläuterung betrachte man folgendes Programm zur Berechnung der reellen Wurzeln der quadratischen Gleichung $x^2+ax+b=0$ (a, b reell):

```
begin
  real a,b,d,w1,w2;
  Boolean test;
  Lies(a,b);
  d := a × a − 4 × b;
  test := d < 0;
  if test then goto l;
  if d = 0 then w1 := w2 := −a/2 else begin
  d := sqrt(d);
  w1 := (−a + d)/2;
  w2 := (−a − d)/2                    end;
  Drucke(w1,w2);
l:end
```

Neben den für die Rechnung benötigten **real**-Größen wird eine Variable *test* als vom Typ **Boolean** vereinbart. Auf das Einlesen von a und b folgen zwei Ergibtanweisungen: Die erste ordnet der Variablen d den Zahlwert des arithmetischen Ausdrucks der Diskriminante, die zweite der Variablen *test* den logischen Wert des Vergleichs $d<0$ zu. *test* erscheint in der Wenn-Klausel der bedingten Anweisung erster Art und veranlaßt den Sprung zum Programmende, wenn der aktuelle Wert dieser Variablen **true**, d. h. die Diskriminante negativ ist. Sonst wird zur nächsten Anweisung übergegangen, die eine bedingte zweiter Art ist und in der Wenn-Klausel den Vergleich $d=0$ enthält. Ist dieser zutreffend, so wird in einer Anweisung den Variablen $w1$ und $w2$ der Wert der jetzt vorliegenden Doppelwurzel zugewiesen. Wir haben hier ein Beispiel dafür, daß durch eine Ergibtanweisung der Wert eines Ausdrucks mehreren Variablen zugeordnet werden kann. Dem Leser sei empfohlen, das an Hand des untenfolgenden Syntaxschemas der Ergibtanweisung zu rechtfertigen. Es ist hinzuzufügen, daß alle in einer mehrfachen Ergibtanweisung links von := (Liste des linken Teils) vorkommenden Variablen vom gleichen Typ sein müssen. Ist $d=0$ nicht erfüllt, so wird in dem auf **else** folgenden Anweisungsverbund d mit dem Wert seiner Quadratwurzel aktualisiert, und mit Hilfe von Ergibtanweisungen erhalten $w1$ und $w2$ die beiden Wurzelwerte der quadratischen Gleichung. Am Schluß erfolgt die Ausgabe von $w1$ und $w2$. Man beachte das Semikolon vor der Marke *l*.

Die Einführung der Variablen *test* ist natürlich vermeidbar, da man den Vergleich $d<0$ unmittelbar in die erste Wenn-Klausel des Programms eintragen kann.

Wir wenden uns nun dem recht umfangreichen, die verschiedenen Arten von Anweisungen erfassenden Syntaxkomplex zu. Darin sind mehr Sprachstrukturen enthalten, als in dieser Darstellung Verwendung finden. Wegen des bereits erwähnten rekursiven Aufbaues von ALGOL ist es jedoch kaum möglich, den tatsächlich benutzten Teil der Sprache für sich zu definieren. Wir beginnen mit der Syntax der *Ergibtanweisung*:

⟨Ergibtanweisung⟩ ::= ⟨Liste des linken Teils⟩ ⟨arithmetischer Ausdruck⟩|
⟨Liste des linken Teils⟩ ⟨logischer Ausdruck⟩

⟨Liste des linken Teils⟩ ::= ⟨linker Teil⟩|⟨Liste des linken Teils⟩ ⟨linker Teil⟩

⟨linker Teil⟩ ::= ⟨Variable⟩:=|⟨Prozedurbezeichnung⟩:=

Neben Ergibtanweisungen traten in den Programmbeispielen die leere Anweisung und spezielle Sprunganweisungen der Form

goto ⟨Marke⟩

auf. Diese fallen sämtlich unter den Begriff der nicht markierten Grundanweisung gemäß der Definition

⟨nicht markierte Grundanweisung⟩ ::= ⟨Ergibtanweisung⟩|⟨Sprunganweisung⟩|
⟨leere Anweisung⟩|⟨Prozeduranweisung⟩

Prozeduranweisungen werden genauer in 3.3. erörtert; ein Beispiel dafür ist die im ALGOL-Text zum Euklidischen Algorithmus auftretende Struktur

entier(x/y).

Die allgemeine Form der Sprunganweisung ist

⟨Sprunganweisung⟩ ::= **goto** ⟨Zielausdruck⟩

Der Begriff der *Grundanweisung* ist nun durch

⟨Grundanweisung⟩ ::= ⟨nicht markierte Grundanweisung⟩|⟨Marke⟩ :
⟨Grundanweisung⟩

definiert. Als Beispiele zitieren wir aus den ALGOL-Texten des Euklidischen Algorithmus und der Wurzelberechnung

L: q:= *entier*(x/y) bzw. l:

Die erste Struktur ist eine markierte Ergibtanweisung, die zweite stellt die mit der Marke l versehene leere Anweisung dar.

Als nächste Verallgemeinerung definiert man die *unbedingten Anweisungen*:

⟨unbedingte Anweisung⟩ ::= ⟨Grundanweisung⟩|⟨Verbundanweisung⟩|⟨Block⟩

Der Begriff des *Blocks* wird in 3.2. erörtert.

Beispiele für Verbundanweisungen sind im ALGOL-Text zum Euklidischen Algorithmus und im Programm zur Lösung quadratischer Gleichungen aufgetreten:

begin $x := m$; $y := n$ **end**
begin $x := y$; $y := r$; **goto** L **end**
begin $d := sqrt(d)$; $w1 := (-a+d)/2$; $w2 := (-a-d)/2$ **end**

Die Definition in der Backusschen Normalform lautet

⟨Verbundanweisung⟩ ::= ⟨nicht markierter Verbund⟩|⟨Marke⟩: ⟨Verbundanweisung⟩
⟨nicht markierter Verbund⟩ ::= **begin** ⟨Verbundschluß⟩
⟨Verbundschluß⟩ ::= ⟨Anweisung⟩ **end**|⟨Anweisung⟩; ⟨Verbundschluß⟩

(Marke siehe S. 92, Anweisung siehe unten). Man beachte die in der Definition des Begriffs ⟨Verbundschluß⟩ enthaltene Regelung der Verwendung des Semikolons in Verbundanweisungen.

Der allgemeine Begriff der *Anweisung* wird durch

⟨Anweisung⟩ ::= ⟨unbedingte Anweisung⟩|⟨bedingte Anweisung⟩|
⟨Laufanweisung⟩

bestimmt. *Bedingte Anweisungen* sind in allen der bisher betrachteten Programmbeispiele aufgetreten, wobei im erläuternden Text zwischen solchen erster bzw. zweiter Art unterschieden wurde. Sie sind wesentlich den folgenden Strukturen im PAP zuzuordnen, die insofern noch zu speziell sind, als sie von der Annahme ausgehen, daß die Anweisungen keine Sprüngeenthalten (vgl. Abb. 3.3 und 3.4). Die Syntax der bedingten Anweisung lautet:

⟨bedingte Anweisung⟩ ::= ⟨Wenn-Anweisung⟩|⟨Wenn-Anweisung⟩
else ⟨Anweisung⟩|⟨Wenn-Klausel⟩
⟨Laufanweisung⟩|⟨Marke⟩ : ⟨bedingte Anweisung⟩
⟨Wenn-Anweisung⟩ ::= ⟨Wenn-Klausel⟩ ⟨unbedingte Anweisung⟩

(Wenn-Klausel siehe S. 90, logischer Ausdruck S. 91, Laufanweisung siehe unten).

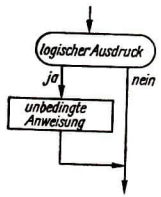


Abb. 3.3

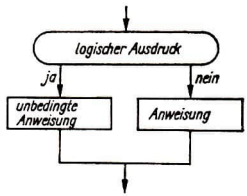


Abb. 3.4

Daraus ergeben sich einige Konsequenzen, die man sich zur Vermeidung von Programmierfehlern als Faustregeln einprägen sollte:

Regel 1. Auf die Wenn-Klausel einer bedingten Anweisung kann nur eine unbedingte Anweisung oder eine Laufanweisung folgen. In einem syntaktisch richtigen ALGOL-Text darf daher **then** nicht vor **if** stehen.

Regel 2. Laufanweisungen sind keine unbedingten Anweisungen. Nach der Definition der Wenn-Anweisung ist somit eine Laufanweisung vor **else** nicht zulässig.

Regel 3. Sollte der zu programmierende Algorithmus die Benutzung von Anweisungen erfordern, die nach Regel 1 oder 2 verboten sind, so können diese durch Einschließen in die Klammern **begin** und **end** (Übergang zu einer Verbundanweisung) in zulässige verwandelt werden.

Den speziellen Typ der Laufanweisung mit einem **step-until**-Element haben wir bei der Erörterung des Horner-Algorithmus einschließlich der entsprechenden PAP-Struktur beschrieben. Die Syntax des allgemeinen Begriffs lautet:

⟨Laufanweisung⟩ ::= ⟨Laufklausel⟩ ⟨Anweisung⟩|⟨Marke⟩ : ⟨Laufanweisung⟩
⟨Laufklausel⟩ ::= **for** ⟨Variable⟩ :=⟨Laufliste⟩ **do**
⟨Laufliste⟩ ::= ⟨Lauflistenelement⟩|⟨Laufliste⟩, ⟨Lauflistenelement⟩
⟨Lauflistenelement⟩ ::= ⟨arithmetischer Ausdruck⟩|⟨arithmetischer Ausdruck⟩ **step** ⟨arithmetischer Ausdruck⟩ **until** ⟨arithmetischer Ausdruck⟩|⟨arithmetischer Ausdruck⟩ **while** ⟨logischer Ausdruck⟩

(Variable siehe S. 94). Auf dieses Schema werden wir im Zusammenhang mit Programmbeispielen zurückkommen. Hier sei abschließend nur darauf hingewiesen, daß sich die im ALGOL-Text des Horner-Algorithmus auftretende Zeichenreihe

for $i :=$ **4 step** -1 **until** 0 **do** $p := A[i] + x0 \times p$

als von der Struktur

for ⟨Variable⟩ := ⟨arithmetischer Ausdruck⟩ **step** ⟨arithmetischer Ausdruck⟩
until ⟨arithmetischer Ausdruck⟩ **do** ⟨Anweisung⟩

einordnet.

Aufgabe 7. Man stelle den in ALGOL formulierten Algorithmus zur Lösung einer quadratischen Gleichung (ohne die Boolesche Variable *test*) in einem PAP dar.

Aufgabe 8. Man schreibe ein ALGOL-Programm, das die Plätze des
integer array *A*[1:5]
mit 0 beginnend alternierend mit 0 und 1 belegt.

Aufgabe 9. SL sei der Stundenlohn eines Arbeiters, A die Anzahl der in einem Monat geleisteten Stunden. Man berechne den Bruttolohn B und den von ihm zu entrichtenden Sozialversicherungsbeitrag SV mit Hilfe eines ALGOL-Programms.

Aufgabe 10. Man interpretiere die folgende bedingte Anweisung, in der x eine Variable vom Typ **real** bedeutet:

if $x \leqq 0 \supset x = 0$ **then else** $x := -x$

und wandele sie in eine die Standardfunktion *abs* benutzende Ergibtanweisung um.

3.2. Blockstruktur von ALGOL-Programmen

Die von uns bisher betrachteten ALGOL-Programme hatten die in Abb. 3.5. gezeigte Struktur, d. h., alle im Programm benutzten Größen (mit Ausnahme der Marken) waren an dessen Anfang zu vereinbaren. Das galt auch für Felder, die dabei

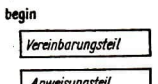


Abb. 3.5

hinsichtlich ihrer Dimension und Indexbereiche konkret ausgewiesen sein mußten. Wir wollen damit sagen, daß es nicht möglich ist, diese Parameter nur symbolisch ins Programm zu schreiben und etwa durch eine *Lies*-Anweisung bei dessen Abarbeitung später „von außen" zu bestimmen. Auf diese Weise ergeben sich hinsichtlich der universellen Benutzbarkeit solcher Programme starke Einschränkungen. Beispielsweise konnten wir das Programm zur Berechnung von Polynomwerten nach dem Hornerschen Schema nicht für einen beliebigen Grad formulieren. Die jetzt zu erörternde *Blockstruktur* von ALGOL, die es ermöglicht, Größen nicht nur am Anfang eines Programms, sondern auch innerhalb desselben zu vereinbaren, beseitigt diese Hindernisse.

Ein *Block* ist eine von den Grundsymbolen **begin** und **end** eingeschlossene Struktur, an deren Anfang Vereinbarungen stehen, denen Anweisungen und möglicherweise weitere Blöcke folgen. Diese werden durch Semikolon voneinander getrennt,

und zwar gemäß den in 3.1.2. und 3.1.3. getroffenen Festlegungen. Nach der ALGOL-Syntax ist ein Block eine spezielle Anweisung (vgl. 3.1.3., S. 99) und kann daher in einem Programm überall dort stehen, wo eine Anweisung auftreten darf. Aus dieser Beschreibung geht der rekursive Charakter der Blockstruktur hervor, der in der Backusschen Normalform so zu präzisieren ist:

⟨Block⟩ ::= ⟨nicht markierter Block⟩|⟨Marke⟩ : ⟨Block⟩
⟨nicht markierter Block⟩ ::= ⟨Blockkopf⟩ ; ⟨Verbundschluß⟩
⟨Blockkopf⟩ ::= **begin** ⟨Vereinbarung⟩|⟨Blockkopf⟩; ⟨Vereinbarung⟩
⟨Verbundschluß⟩ ::= ⟨Anweisung⟩ **end**|⟨Anweisung⟩; ⟨Verbundschluß⟩

(Anweisung siehe S. 99, Vereinbarung S. 106).

Zur Erläuterung einer Blockstruktur betrachten wir das folgende ALGOL-Programm zur Berechnung der Werte eines Polynoms von beliebigem Grad nach dem Hornerschen Schema:

```
begin
  integer n;
  Lies(n);
    begin
      integer i; real x0,p; array A[0:n];
      Lies(A, x0);
      p := A[n];
      for i := n−1 step −1 until 0 do p := A[i]+x0 ×p;
      Drucke(p)
    end
end
```

Die Blockstruktur ist typisiert in Abb. 3.6 dargestellt; der syntaktische Aufbau entspricht dem Schema

begin ⟨Vereinbarung⟩; ⟨Anweisung⟩; ⟨Block⟩ **end**

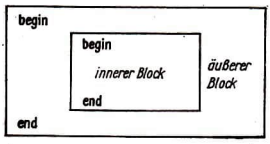
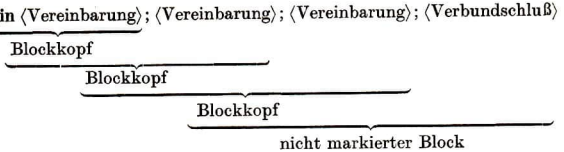


Abb. 3.6

Daraus resultiert, daß das Programm als Ganzes ein nicht markierter Block ist: **begin** ⟨Vereinbarung⟩ ist nämlich Blockkopf und ⟨Anweisung⟩; ⟨Block⟩ **end** Verbundschluß.

Daß der im inneren Rahmen stehende Programmteil tatsächlich ein Block ist, folgt aus seiner Syntaxstruktur:



Im äußeren Block wird nur der Polynomgrad n vereinbart und eingelesen. Dieser ist also vor Eintritt in den inneren Block, der zusammen mit *Lies*(n) den Anweisungsteil des äußeren ausmacht, konkret verfügbar und wird im inneren Block zur Vereinbarung des Feldes A benutzt. Dieser umfaßt im übrigen mit den weiteren Vereinbarungen und Anweisungen für dieses n den schon für $n=5$ programmierten Algorithmus des Hornerschen Schemas. Der Datenstreifen (vgl. dazu die Bemerkung auf S. 88) wäre folgendermaßen zu organisieren:

$$n,\ a_0,\ a_1,\ \ldots,\ a_n,\ x_0.$$

Beim Einlesen von Feldern wollen wir von der Annahme ausgehen, daß die Komponenten gemäß der lexikographischen Anordnung der Indextupel von der EDVA übernommen werden, und diese wie im Beispiel entsprechend notieren.

Die im inneren Block auftretende Größe n ist dort nicht vereinbart und wird deshalb in diesem als *global* bezeichnet. So wird es möglich, darin das Feld $A[0:n]$ mit beliebigem n (oder, wie man sagt, *dynamisch*) zu vereinbaren. Alle in einem Block vereinbarten Größen heißen bezüglich desselben *lokal*; sie haben nur in dem Block Bedeutung, in dem sie vereinbart wurden. *Drucke*(p) kann deshalb nicht als letzte Anweisung des äußeren Blocks unseres Beispiels auftreten.

Die automatische Bearbeitung eines Praxisproblems erfordert, daß alle Aktivitäten, die der Mensch als Rechner ausführen würde, in den Lösungsalgorithmus und dessen Programm einbezogen sind. Dazu gehören auch Handlungen, die in weitestem Sinne mit Organisationsfragen zusammenhängen. Als Beispiel dazu betrachten wir die Aufgabe, über einem eindimensionalen Zahlenfeld A zwischen zwei Platznummern p, q ($p \leqq q$) den größten und den kleinsten Wert MA bzw. MI der Feldkomponenten zu ermitteln; Grenzen des Indexbereichs von A seien die ganzen Zahlen k und l. Das ALGOL-Programm bietet Gelegenheit, die meisten der bisher entwickelten Sprachelemente einzusetzen.

Die Lösung der Aufgabe erfordert die vergleichende Betrachtung der Größen $A[p]$, $A[p+1]$, ..., $A[q]$, die mit Hilfe einer Zählvariablen i organisiert wird. Der zugehörige PAP ist in Abb. 3.7 dargestellt. Wir erkennen darin eine Teilstruktur, die durch eine Laufanweisung mit **step-until**-Element erfaßt werden kann. Sie beginnt mit der Anfangsstellung von i auf den Wert $p+1$. Der folgenden Abbruchbedingung für den Durchlauf des Zyklus entnimmt man, daß dieser zum letzten Mal für den Wert $i=q$ erfolgt. Die vor dem Weiterrücken von i aufzuführende Anweisung ist im PAP durch das in Abb. 3.8 gezeigte Stück beschrieben. Denkt

man sich die eingerahmte komplexe Anweisung zunächst als ein Ganzes und bezeichnet sie zur Abkürzung mit ANW, so entspricht der Abb. 3.8 die bedingte Anweisung zweiter Art

if $MA < A[i]$ **then** $MA := A[i]$ **else** ANW

ANW selbst kann als bedingte Anweisung erster Art formuliert werden:

if $MI > A[i]$ **then** $MI := A[i]$

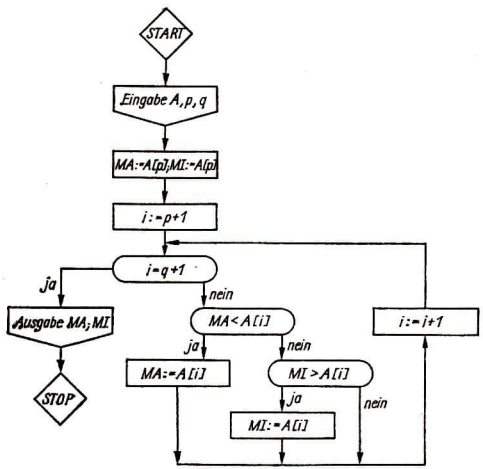


Abb. 3.7

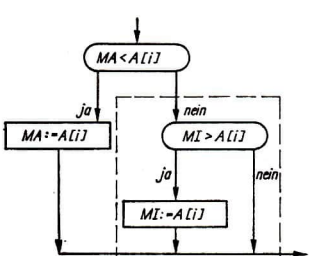


Abb. 3.8

Die betrachtete Laufanweisung lautet nunmehr

```
for i := p+1 step 1 until q do
if MA < A[i] then MA := A[i]
                else
 if MI > A[i] then MI := A[i]
```

Wir schreiben nun das vollständige ALGOL-Programm zur Maximum-Minimum-Bestimmung, wobei – dem PAP der Abb. 3.7 folgend – am Anfang A, p, q einzulesen sind. Das erfordert eine dynamische Vereinbarung des Feldes A, die durch einen äußeren Block verwirklicht wird, in dem man die Grenzen k, l vereinbart:

```
begin
  integer k, l;
  Lies (k, l);
    begin
      integer i, p, q; real MA, MI; array A[k: l];
      Lies (A, p, q);
      MA := A[p]; MI := A[p];
      for i := p+1 step 1 until q do
      if MA < A[i] then MA := A[i]
                   else
      if MI > A[i] then MI := A[i];
      Drucke (MA, MI);
    end
end
```

Bei der Einführung der lokalen Größen eines Blocks wurde darauf hingewiesen, daß diese außerhalb desselben keine Bedeutung haben. Das gilt auch für den Fall, daß solche Größen dort noch einmal mit der gleichen Bezeichnung vereinbart werden. Beispielsweise würde die Programmstruktur in Abb. 3.9 genauso interpretiert, als wenn in dem eingerahmten Block an allen Stellen von n etwa m stünde,

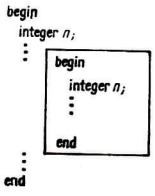


Abb. 3.9

sofern diese Bezeichnung nicht sonst schon in diesem Block – etwa als Name einer globalen Größe – vorkommt. Beim Verlassen desselben erfolgt keine Zuweisung des Wertes von n an die außen vereinbarte Variable gleichen Namens. Eine solche

Duplizität der Bezeichnung sollte man jedoch vermeiden, da sie leicht zu Irrtümern und daraus resultierenden Programmfehlern führen kann.

Es ist zweckmäßig, Marken als lokale Größen des Blockes zu interpretieren, in dem sie auftreten. So ergibt sich eine Begründung für das Verbot, von außen in einen Block „hineinzuspringen", da die betreffende Marke als ALGOL-Größe in der Sprung anweisung nicht existent ist. Hingegen ist der Sprung aus einem Block zu einer in einem übergeordneten Block befindlichen markierten Anweisung möglich.

Die syntaktische Beschreibung des Blocks erfordert die Definition des Begriffs ⟨Vereinbarung⟩. Das folgende Schema in der Backusschen Normalform umfaßt die bisher dazu gegebenen Erklärungen:

⟨Vereinbarung⟩ ::= ⟨Typvereinbarung⟩|⟨Feldvereinbarung⟩|
⟨Verteilervereinbarung⟩|⟨Prozedurvereinbarung⟩
⟨Typvereinbarung⟩ ::= ⟨Typ⟩ ⟨Typenliste⟩|**own** ⟨Typ⟩ ⟨Typenliste⟩
⟨Typ⟩ ::= **real**|**integer**|**Boolean**
⟨Typenliste⟩ ::= ⟨einfache Variable⟩|⟨einfache Variable⟩, ⟨Typenliste⟩
⟨Feldvereinbarung⟩ ::= **array** ⟨Feldliste⟩|⟨Typ⟩ **array**
⟨Feldliste⟩| **own** ⟨Typ⟩ **array** ⟨Feldliste⟩
⟨Feldliste⟩ ::= ⟨Feldsegment⟩|⟨Feldliste⟩, ⟨Feldsegment⟩
⟨Feldsegment⟩ ::= ⟨Feldbezeichnung⟩ [⟨Grenzenliste⟩]|
⟨Feldbezeichnung⟩, ⟨Feldsegment⟩
⟨Grenzenliste⟩ ::= ⟨Grenzenpaar⟩|⟨Grenzenliste⟩, ⟨Grenzenpaar⟩
⟨Grenzenpaar⟩ ::= ⟨untere Grenze⟩ : ⟨obere Grenze⟩
⟨obere Grenze⟩ ::= ⟨arithmetischer Ausdruck⟩
⟨untere Grenze⟩ ::= ⟨arithmetischer Ausdruck⟩
(arithmetischer Ausdruck siehe Seite 90, Variable S. 94).

Die Syntax der Feldvereinbarung wollen wir noch etwas erläutern:

1. Die Bestimmung des Begriffs ⟨Feldsegment⟩ gestattet die listenmäßige Vereinbarung mehrerer Felder gleichen Typs, gleicher Dimension und gleicher Indexbereiche mit nur einer in die Klammern [und] eingeschlossenen *Grenzenliste*. Möglich ist zum Beispiel

array *A, B, C* [−2:3, 0:5]

2. Die Grenzen der Indexbereiche von Feldern sind allgemein gesprochen arithmetische Ausdrücke, die (vgl. S. 90) speziell auch ganze Zahlen sein können. Arithmetische Ausdrücke, die Variable enthalten, treten wie im Beispiel des Horner-Algorithmus im Zusammenhang mit einer dynamischen Feldvereinbarung auf. Ihre Auswertung liefert unter Umständen Resultate vom Typ **real**. In diesem Fall erfolgt eine Resultatersetzung durch die nächste ganze Zahl, und zwar so, wie das bei der Zuweisung eines **real**-Wertes an eine Variable vom Typ **integer** erklärt wurde.

3. Bei der Übersetzung werden gewisse syntaktisch richtige Feldvereinbarungen nicht akzeptiert. Die Reservierung von Speicherraum erfolgt nicht, wenn eine untere Indexgrenze größer als die entsprechende obere ist.

Nach der Definition des Blocks kann nun auch die syntaktische Struktur eines Programms bestimmt werden:

ALGOL-Programm ist jeder Block und jede Verbundanweisung, die nicht in einer anderen Anweisung enthalten sind oder von einer solchen Gebrauch machen.

Aufgabe 1. Man prüfe, ob der folgende Text ein ALGOL-Programm ist, und interpretiere ihn gegebenenfalls:

```
begin
  integer i, j; real SU1,SU2;
  SU 1:=S U2:=
  0; for i:=2 step 2 until 50 do
  begin j:=i+i; SU1:=SU1+1/(j-1);
  SU2:=SU2+1/j end; end
```

Aufgabe 2. Im Programm zur Berechnung von Polynomwerten nach dem Hornerschen Schema von S. 102 ersetze man die Feldvereinbarung **array** $A[0{:}n]$ durch

$$\textbf{array}\ A[3\times(n-1)\times(n-2)/2{:}(n-3)\times(2-3\times n)/2]$$

Man prüfe die syntaktische Richtigkeit des modifizierten Textes und interpretiere diesen für die Eingangsdaten eines Polynoms zweiten bzw. dritten Grades.

Aufgabe 3. Man schreibe ein dem PAP der Abb. 2.7 entsprechendes ALGOL-Programm zur Konvertierung einer natürlichen Zahl im Dezimalsystem in eine g-adische Darstellung.

3.3. Prozeduren

Gewisse mathematische Methoden, wie etwa Verfahren zur Lösung von Gleichungen, zur Bestimmung der Extrema von Funktionen oder zur Lösung von Approximationsaufgaben haben Basischarakter und sind, den verschiedensten Praxissituationen angepaßt, immer wieder einzusetzen. Es ist daher notwendig, diese algorithmisch sorgfältig zu studieren und optimal (hinsichtlich der Rechenzeit und auftretenden Rundungsfehler) zu programmieren. Das erfordert aus Gründen der Arbeitsteilung, der vielseitigen Verwendbarkeit und damit verbundenen Kostensenkung und bequemen programmiertechnischen Handhabung eine gewisse Verselbständigung solcher Verfahren, nämlich ihre Behandlung als *Unterprogramme*. Diese *Unterprogrammtechnik* wird in ALGOL 60 durch die Konzeption der *Prozeduren* in sehr vollkommener Weise verwirklicht.

Prozeduren sind ALGOL-Größen, die als solche zu bezeichnen sind und mit dem Grundsymbol **procedure** vereinbart werden. Im einfachsten Fall macht das den sogenannten *Prozedurkopf* aus, dem, durch ein Semikolon getrennt, der *Prozedurhauptteil* (auch *Prozedurkörper* genannt) folgt. Dieser ist eine Anweisung, meistens ein Verbund. Darin sind die Anweisungen zusammengefaßt, welche das als Unterprogramm zu fixierende Verfahren ausmachen. Speziell kann der Prozedurhauptteil ein Block sein und beginnt dann mit der Vereinbarung für dessen Abarbeitung benötigter lokaler Größen.

Wir wollen in dieser Weise das Verfahren der Maximum-Minimum-Bestimmung über einem Zahlenfeld (vgl. S. 103–105) in einer mit *MINIMAX* bezeichneten Prozedur erfassen:

```
procedure MINIMAX;
  begin
    integer i;
    MA := A[p]; MI := A[p];
    for i := p+1 step 1 until q do
    if MA < A[i]then MA := A[i]
                 else
    if MI > A[i] then MI := A[i]
  end
```

In dieser Prozedur ist die **integer**-Größe i lokal. p, q, MA, MI und das Feld A werden in *MINIMAX* nicht vereinbart und heißen deshalb global. Sie müssen in einem Programm, das diese Prozedur benutzt, anderweitig vereinbart werden. Das Beispiel von S. 105 könnte bei Benutzung von *MINIMAX* folgende Gestalt haben:

```
begin
  integer k, l;
  Lies (k, l);
    begin
      integer p, q; real MA, MI; array A[k : l];
      procedure MINIMAX;
        begin
          integer i;
          MA := A[p]; MI := A[p];
          for i := p+1 step 1 until q do
          if MA < A[i] then MA := A[i]
                       else
          if MI > A[i] then MI := A[i]
        end;
      Lies (A, p, q);
      MINIMAX;
      Drucke (MA, MI);
    end
end
```

Die Prozedur *MINIMAX* wurde im inneren Block vereinbart und nach dem Einlesen der Größen A, p, q mit ihrem Namen aufgerufen. Das bedeutet: Der Prozedurkörper wird an dieser Stelle als Block in das Programm eingefügt. Bei nur einmaligem Prozeduraufruf erscheint es einfacher, die Prozedurbildung zu vermeiden und die darin zusammengefaßten Anweisungengleich in dieser Weise in das Programm zu schreiben. Bezüglich der in einer Prozedur eventuell auftretenden globalen Größen beachte man: *Globale Größen einer Prozedur müssen spätestens in dem*

Block vereinbart werden, in dem die Prozedur vereinbart wird. Dieser Forderung wurde in dem betrachteten Beispiel genügt, da wir *MINIMAX* zusammen mit *MA*, *MI*, *A*, *p* und *q* auf dem niedrigsten Blockniveau vereinbart haben.

Zur Übung wollen wir noch den Euklidischen Algorithmus als Prozedur mit dem Namen *ggT* formulieren:

```
procedure ggT;
  begin
    integer x,q,r;
    if m≧n then begin x:=m; y:=n end
           else begin x:=n; y:=m end;
  L:q:=entier(x/y);
    r:=x−q×y;
    if r>0 then begin x:=y; y:=r; goto L end
  end
```

x, *q*, *r* sind lokal und haben daher außerhalb der Prozedur keine Bedeutung; *y*, *m*, *n* sind globale Größen, die in einem Programm, das *ggT* benutzt, anderweitig vereinbart werden müssen. Wir betrachten dazu als Beispiel die Aufgabe, zwei ganze Zahlen *m*, *n* einzulesen, ihren größten gemeinsamen Teiler zu bestimmen und diesen Wert auszudrucken. Ein dafür geeignetes Programm ist:

```
begin
  integer m, n, y;
  procedure ggT;
    begin
      integer x,q,r;
      if m≧n then begin x:=m; y:=n end
             else begin x:=n; y:=m end;
    L: q:=entier(x/y);
       r:=x−q×y;
      end; if r>0 then begin x:=y; y:=r; goto L end
  Lies(m,n);
  ggT;
  Drucke(y);
end
```

Wie schon gesagt, ist es der Zweck der Prozeduren, häufig auftretende Algorithmen so zu erfassen, daß man sie bausteinartig in Programmen zusammenfügen kann. Wir wollen in diesem Sinne beispielsweise folgende Aufgabe lösen:

Auf einem Zahlenfeld **integer array** *A*[*k*:*l*] (*A*[*i*]>0) soll zwischen zwei Platznummern *p* und *q* das Maximum und das Minimum bestimmt und für diese beiden Werte der größte gemeinsame Teiler ermittelt werden. Es liegt nahe, für die Lösung an das Programm auf S. 108 anzuschließen und zusätzlich darin noch die Prozedur *ggT* zu vereinbaren, um sie nach der Prozedur *MINIMAX* aufzurufen.

Diese Absicht scheitert jedoch an dem Umstand, daß *ggT* nur bezüglich der globalen Größen *m*, *n*, nicht aber für *MA* und *MI* in Anspruch genommen werden kann. *m*, *n* sind aber in keinem der beiden Blöcke vereinbart und schon gar nicht mit Werten belegt. Diese Sachlage macht deutlich, daß die bisher betrachteten Prozeduren wenig flexibel sind.

Die gewünschte Anpassungsfähigkeit besitzen *Prozeduren mit formalen Parametern*, deren Betrachtung wir uns nun zuwenden. Die sogenannten formalen Parameter erscheinen als Liste, in Klammern eingeschlossen, hinter dem Prozedurnamen. Beim Aufruf der Prozedur werden für sie die Bezeichnungen der Größen eingetragen, an die man Anschluß haben möchte. Man sagt: Die formalen Parameter werden beim Aufruf *aktualisiert*.

In der Prozedur *ggT* wäre es aus den genannten Gründen zweckmäßig, *m* und *n* als formale Parameter zu verwenden:

```
procedure ggT(m,n);
  begin
    integer x,q,r;
    if m ≧ n then begin x := m; y := n end
             else begin x := n; y := m end;
  L: q := entier(x/y);
    r := x − q × y;
    if r > 0 then begin x := y; y := r; goto L end
  end
```

Das Programm zu der oben formulierten Aufgabe könnte dann lauten:

```
begin
  integer k,l;
  Lies(k,l);
  begin
    integer p,q,y,MA,MI; integer array A[k:l];
    procedure MINIMAX;
      begin
        integer i;
        MA := A[p]; MI := A[p];
        for i := p + 1 step 1 until q do
        if MA < A[i] then MA := A[i]
                     else
        if MI > A[i] then MI := A[i]
      end;
    procedure ggT(m,n);
      begin
        integer x,q,r;
        if m ≧ n then begin x := m; y := n end
                 else begin x := n; y := m end;
```

```
        L: q := entier(x/y);
           r := x - q × y;
           if r > 0 then begin x := y; y := r; goto L end
        end;
      Lies(A,p,q);
      MINIMAX;
      ggT(MA,MI);
      Drucke(y);
    end
  end
```

Die Bezeichnung der formalen Parameter ist weitgehend willkürlich; es ist erlaubt und wird oft so gehalten, ihnen die Namen aktueller Parameter zu geben.

Für formale Parameter, die Variable vertreten, können beim Aufruf arithmetische Ausdrücke eingesetzt werden. Das hätte zur Konsequenz, daß nach dem Aufruf an allen Stellen des Prozedurkörpers, wo diese Variablen vorkommen, die entsprechenden Ausdrücke eingesetzt und an allen diesen Positionen berechnet werden müßten. Es wäre ökonomischer, das nur einmal zu tun und statt der Ausdrücke die entsprechenden Werte zu substituieren. Das kann erreicht werden, wenn man diese Variablen in einen sogenannten *Werteteil* des Prozedurkopfs aufnimmt. Dieser beginnt mit dem Grundsymbol **value**, dem in einer Liste die Bezeichnungen der betroffenen Parameter folgen, und schließt mit einem Semikolon. Beim Aufruf werden den Vereinbarungen am Anfang des Prozedurkörpers weitere für die Parameter des Werteteils hinzugefügt, so daß diese nunmehr den Charakter lokaler Größen haben. Dazu ist die Kenntnis ihres Typs erforderlich, der in einem dem Werteteil folgenden *Benennungs-* oder *Spezifikationsteil* festgelegt sein muß. Formal hat dieser die Gestalt einer durch ein Semikolon abgeschlossenen Vereinbarung bezüglich der im Werteteil erscheinenden Variablen. Es ist syntaktisch zulässig, auch alle übrigen formalen Parameter in dieser Weise zu spezifizieren, wobei Felder nur mit ihren Bezeichnungen, also ohne die in eckige Klammern eingeschlossene Indexliste, anzugeben sind. Da die meisten Maschinenversionen von ALGOL – auch die in 3.4. behandelte Variante für die EDVA R 300 – die Spezifikation aller formalen Parameter `erfordern`, sollte man sich daran gewöhnen, so zu verfahren. Je nachdem, ob ein formaler Parameter im Werteteil erscheint oder nicht, spricht man beim Aufruf der Prozedur bezüglich desselben von einem Aufruf mit Wert bzw. Namen (call by value bzw. call by name). Die Prozedur *ggT* zum Beispiel hat mit *m* und *n* im Werteteil die Gestalt

```
procedure ggT(m,n); value m,n; integer m,n;
 begin
  integer x,g,r;
  if m ≧ n then begin x := m; y := n end
           else begin x := n; y := m end;
L: q := entier(x/y);
```

```
    r := x − q × y;
    if r > 0 then begin x := y; y := r; goto L end
  end
```

Wir gehen nun noch einen Schritt weiter und führen in ALGOL jenen Typ von Prozeduren ein, der eine Erfassung von Funktionen in der Weise ermöglicht, daß man deren Werte wie üblich (etwa in arithmetischen Ausdrücken) benutzen kann.

Eine *Funktionsprozedur* berechnet – den Anweisungen des Prozedurkörpers folgend – einen Funktionswert und weist diesen dem Prozedurnamen zu. Dem Grundsymbol **procedure** im Prozedurkopf muß die für die Funktionswerte maßgebende Typangabe vorangehen.

Wir wollen als Beispiel die Prozedur *ggT* betrachten. Die durch den größten gemeinsamen Teiler zwei natürlichen Zahlen *m*, *n* zugeordneten Funktionswerte sind vom Typ **integer**. Als Funktionsprozedur hat *ggT* die Gestalt

```
integer procedure ggT(m,n); value m,n; integer m,n;
  begin
    integer x,y,q,r;
    if m ≧ n then begin x := m; y := n end
             else begin x := n; y := m end;
  L: q := entier(x/y);
    r := x − q × y;
    if r > 0 then begin x := y; y := r; goto L end;
    ggT := y
  end
```

Der größte gemeinsame Teiler wird zunächst auf dem jetzt lokalen Speicherplatz *y* gebildet und schließlich dem Namen der Prozedur *ggT* zugeordnet.

Schließlich soll noch das Hornersche Schema als Funktionsprozedur programmiert werden. Als formale Parameter wählen wir den Polynomgrad, das Feld der Koeffizienten und das Argument. Diese Größen seien mit *m*, *B* bzw. *t*, die Prozedur selbst mit *HORNER* bezeichnet. Es empfiehlt sich, *t* in den Werteteil aufzunehmen:

```
real procedure HORNER(m,B,t);
  value m,t; integer m; real t; array B;
  begin
    integer i; real p;
    p := B[m];
    for i := m − 1 step −1 until 0 do
    p := B[i] + t × p; HORNER := p
  end
```

In einem Block, in dem diese Funktionsprozedur vereinbart ist, darf man *HORNER*(*n*,*A*,*x*) so verwenden wie den Wert des durch die Koeffizienten des Feldes *A* bestimmten Polynoms vom Grad *n* an der Stelle *x*.

Das diesen Abschnitt abschließende Syntaxschema enthält die zum Prozedurbegriff gegebenen Erklärungen, die vorab noch durch eine Erörterung des Begriffs der *Zeichenkette* ergänzt werden sollen. Das ist eine beliebige Folge von Grundsymbolen, die im allgemeinen keinen syntaktisch richtigen ALGOL-Text darstellt und Kommentarzwecken dient. Zeichenketten lassen sich nach bestimmten Regeln in Programme einfügen, um deren Lesbarkeit zu erleichtern, oder können bei der Ausgabe zur Erläuterung der Resultate gedruckt werden. Um sie gewissermaßen als Fremdkörper in einem ALGOL-Text zu kennzeichnen, sind sie mit den *Kettenanführungszeichen* ‘ und ’ einzuklammern. Ihre Struktur leitet sich aus diesem Syntaxschema ab:

⟨Zeichenkette⟩ ::= ‘⟨offene Zeichenkette⟩’
⟨offene Zeichenkette⟩ ::= ⟨echte Zeichenkette⟩|‘⟨offene Zeichenkette⟩’|⟨offene Zeichenkette⟩ ⟨offene Zeichenkette⟩
⟨echte Zeichenkette⟩ ::= ⟨jede beliebige Folge von Grundsymbolen, die nicht ‘oder’ enthält⟩|⟨leere Zeichenkette⟩
⟨leere Zeichenkette⟩ ::=

Weitere Einzelheiten zur Handhabung von Zeichenketten werden an Beispielen erklärt.

Für den Aufruf einer Prozedur sind folgende Definitionen maßgebend:

⟨Prozeduranweisung⟩ ::= ⟨Prozedurbezeichnung⟩ ⟨aktueller Parameterteil⟩
⟨Funktion⟩ ::= ⟨Prozedurbezeichnung⟩ ⟨aktueller Parameterteil⟩
⟨Prozedurbezeichnung⟩ ::= ⟨Bezeichnung⟩
⟨aktueller Parameterteil⟩ ::= ⟨leere Zeichenkette⟩|(⟨aktuelle Parameterliste⟩)
⟨aktuelle Parameterliste⟩ ::= ⟨aktueller Parameter⟩|⟨aktuelle Parameterliste⟩ ⟨Parameterbegrenzer⟩ ⟨aktueller Parameter⟩
⟨Parameterbegrenzer⟩ ::= ,|⟨Buchstabenkette⟩ : (
⟨Buchstabenkette⟩ ::= ⟨Buchstabe⟩|⟨Buchstabenkette⟩ ⟨Buchstabe⟩
⟨aktueller Parameter⟩ ::= ⟨Ausdruck⟩|⟨Feldbezeichnung⟩|⟨Verteilerbezeichnung⟩|⟨Prozedurbezeichnung⟩|⟨Zeichenkette⟩

(Bezeichnung siehe S. 87, Ausdruck S. 91).

Die Syntax der Prozedurvereinbarung ist:

⟨Prozedurvereinbarung⟩ ::= **procedure** ⟨Prozedurkopf⟩ ⟨Prozedurhauptteil⟩|⟨Typ⟩ **procedure** ⟨Prozedurkopf⟩ ⟨Prozedurhauptteil⟩
⟨Prozedurhauptteil⟩ ::= ⟨Anweisung⟩|⟨Code⟩
⟨Prozedurkopf⟩ ::= ⟨Prozedurbezeichnung⟩ ⟨formaler Parameterteil⟩; ⟨Werteteil⟩ ⟨Benennungsteil⟩
⟨formaler Parameterteil⟩ ::= ⟨leere Zeichenkette⟩|(⟨formale Parameterliste⟩)
⟨formale Parameterliste⟩ ::= ⟨formaler Parameter⟩|⟨formale Parameterliste⟩ ⟨Parameterbegrenzer⟩⟨formaler Parameter⟩
⟨formaler Parameter⟩ ::= ⟨Bezeichnung⟩
⟨Werteteil⟩ ::= ⟨leere Zeichenkette⟩|**value** ⟨Bezeichnungsliste⟩;

⟨Bezeichnungsliste⟩ ::= ⟨Bezeichnung⟩|⟨Bezeichnungsliste⟩, ⟨Bezeichnung⟩
⟨Benennungsteil⟩ ::= ⟨leere Zeichenkette⟩|⟨Benennung⟩ ⟨Bezeichnungsliste⟩;
⟨Benennungsteil⟩ ⟨Benennung⟩ ⟨Bezeichnungsliste⟩;
⟨Benennung⟩ ::= ⟨Typ⟩|**array**|⟨Typ⟩ **array**|**procedure**|⟨Typ⟩ **procedure**|**label**|
switch|**string**

(Typ siehe S. 106).

Aufgabe 1. Ist es zulässig, in der Funktion *HORNER* den Prozedurkopf durch folgende Struktur zu ersetzen?

real procedure *HORNER*(*m*) Grad des Polynoms mit dem Koeffizientenfeld: (*B*) das Argument ist: (*t*);
value *m*, *t*; **integer** *m*; **real** *t*; **array** *B*;

Aufgabe 2. Mit *a*, *b*, *c* seien die Seitenlängen eines Dreiecks, *F* der Flächeninhalt, *r*, *R* der Radius des In- bzw. Umkreises bezeichnet. Man schreibe die Vereinbarung einer Prozedur, welche diese sechs Größen als formale Parameter enthält und *F*, *r*, *R* aus *a*, *b*, *c* berechnet. Falls *a*, *b*, *c* nicht der Dreiecksungleichung genügen, soll ein Sprung zum Ende des Prozedurkörpers erfolgen. Man lege der Programmierung folgende Formeln zugrunde:

$$2s = a+b+c, \qquad F = \sqrt{s(s-a)\,(s-b)\,(s-c)},$$

$$R = \frac{abc}{4F}, \qquad r = \frac{F}{s}.$$

Aufgabe 3. (x, y) und (r, φ) seien kartesische Koordinaten bzw. Polarkoordinaten eines Punktes der Ebene. Man schreibe die Vereinbarung einer Prozedur, welche diese vier Größen (mit geeigneten Bezeichnungen) als formale Parameter enthält und r, φ aus x, y berechnet.

Aufgabe 4. Man schreibe die Vereinbarung der Funktion *tanh*.

Aufgabe 5. Gesucht ist eine Funktionsprozedur $V(n,X)$ zur Bestimmung des Wertes der Vandermondeschen Determinante

$$V_n = \begin{vmatrix} x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \dots & x_2 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{vmatrix},$$

$$V_n = (x_1-x_2)\,(x_1-x_3)\cdots(x_1-x_n)$$
$$(x_2-x_3)\cdots(x_2-x_n)$$
$$\dots\dots\dots$$
$$(x_{n-1}-x_n)$$

Die Werte $x_1, x_2, \dots, x_n$ denke man sich auf einem Feld **array** *X*[1:*n*] gespeichert.

Aufgabe 6. Betrachtet werden die Permutationen der natürlichen Zahlen 1, 2, ..., *n*; eine bestimmte denke man sich auf dem Feld **integer array** *X*[1:*n*] gespeichert. Es ist die Vereinbarung einer Funktionsprozedur *GU*(*n*,*X*) zu schreiben, die den Wert 0 oder 1 liefert, je nachdem, ob die betrachtete Permutation gerade bzw. ungerade ist.

3.4. R 300-Variante von ALGOL 60

3.4.1. Einschränkungen und Erweiterungen der Bezugssprache

Wir kommen auf die Bemerkungen am Schluß von 3.1.1. zurück und erörtern hier die vom Compiler der EDVA Robotron 300 übersetzbare Variante von ALGOL 60. Ein in dieser Sprache geschriebenes Programm muß dem Rechner mit den zugehörigen Eingabedaten in codierter Form übermittelt werden. Das kann über Achtkanal-Lochstreifen erfolgen. Nach der Eingabe erscheint der ALGOL-Text als Wort über dem internen Alphabet des R 300, also in Form einer gewissen Bytestruktur, im Hauptspeicher der Anlage. Die externe Verschlüsselung auf dem Lochband entspricht in der Mehrzahl der Fälle der Bitbelegung eines Zeichens gemäß dem in 2.3.3. angegebenen Schlüssel mit zusätzlicher Beachtung eines Prüfbits in der fünften Spur. Beispielsweise erscheint die Zeichenfolge algol 60 auf dem Streifen mit den Lochungen der Abb. 3.10. Das Lochband kann etwa auf dem Organisationsautomaten Optima 528 oder Daro 1413 hergestellt werden. Die Zeichen des R 300-Alphabets, deren externe Darstellung von der internen ab-

Abb. 3.10

weicht, sind in Tabelle 3.2 mit den entsprechenden Verschlüsselungen angegeben. Um sie von anderen Zeichen mit derselben Codierung unterscheiden zu können, ist vorher die Umschalttaste GB der Schreibmaschine des Organisationsautomaten zu betätigen. Die davon erzeugte Lochung veranlaßt die Interpretation der folgenden Lochkombination als Zeichen der Tabelle. Soll der Lesevorgang wieder entsprechend der internen Darstellung erfolgen, so muß zuvor die Taste KB betätigt werden.

Bei der Codierung eines ALGOL-Textes entsteht das Problem, eine Zuordnung zwischen den 116 Grundsymbolen von ALGOL und den 64 Zeichen des internen Alphabets des R 300 herzustellen. Diese Aufgabe wird zunächst dadurch vereinfacht, daß man auf einige Grundsymbole und darauf aufbauende Sprachkonstruktionen verzichtet. Nicht zugelassen sind alle Großbuchstaben und **own**. Die Anzahl der übrigbleibenden Zeichen ist aber größer als 64, so daß eine eineindeutige Zuordnung zwischen diesen und denen des internen Alphabets unmöglich ist. Aus diesem Grunde müssen einige ALGOL-Zeichen (Grundsymbole) durch Wörter über dem internen Alphabet ausgedrückt werden.[1]) Das geschieht bei den aus Wörtern der englischen Sprache abgeleiteten Grundsymbolen nach dem Prinzip, das buch-

[1]) Es handelt sich hier um ein Codierungsproblem, wie es allgemein am Ende von 2.3.3. beschrieben wurde.

	Codierung								
	1	2	3		4	5	6	7	8
GB	o	o	o	•	o		o		
KB	o	o	o	•	o			o	
R 300-Zeichen	nach Umschalten								
≺	o			•					
▼▼		o		•					
≈	o	o		•		o			
[			o	•					
≻	o		o	•		o			
▼		o	o	•		o			
≋	o	o	o	•					
~				•	o				
]	o			•	o	o			
=				•			o		
)	o			•			o	o	
?	o	o		•	o		o	o	
;				•		o	o	o	
△				•				o	
!	o	o		•	o	o	o		
(	o		o	•	o				
#				•		o			
*			o	•	o	o	o	o	
Wagenrücklauf/ Zeilenvorschub		o		•	o	o	o		o

Tabelle 3.2

stäblich gleiche Wort über dem internen Alphabet zu bilden und mit dem Zeichen ' einzuklammern. Beispielsweise entsprechen so den Grundsymbolen

begin true Boolean

die Worte

'begin' 'true' 'boolean'.

Die folgende Liste vermittelt die Zuordnung zwischen ALGOL-Symbolen der R 300-Variante und Wörtern des internen Alphabets; in einigen Fällen sind zwei

Ersetzungen möglich. Zeichen des Subset, die darin nicht enthalten sind, werden durch die gleichen Zeichen des internen Alphabets repräsentiert:

:=	:= [1])
×	*
÷	'div'
↑	'power' oder **
=	= oder 'equal'
<	< oder 'less'
>	> oder 'greater'
≦	'notgreater'
≧	'notless'
≠	'notequal'
≡	'equiv'
⊃	'impl'
∨	'or'
∧	'and'
¬	'not'
‘	'('
’	')'
␣	⌑
10	#

In diesem Abschnitt wird jeder ALGOL-Text durch den entsprechenden über dem R 300-Alphabet ausgedrückt, also so formuliert, wie er für die Herstellung des Programmlochbandes vorgelegt werden muß. Zur Unterscheidung des Buchstabens O von der Zahl Null ist deren Ziffernsymbol durchzustreichen. Beim Ablochen wird die äußere Gestalt des Programms, wie sie im Schreibmaschinenprotokoll sichtbar ist, auf dem Lochband reproduziert, d. h., Zeilenschaltungen und Zwischenräume, die durch Betätigung der Taste ¬ und der Leertaste erzeugt werden, erscheinen mit den entsprechenden Lochkombinationen. Diese werden jedoch beim Einlesen des Programms übergangen; man vergleiche dazu die Bemerkung am Anfang von 3.1.2. Um die Beendigung der Eingabe eines Programms anzuzeigen, ist zum Schluß die Kombination Wagenrücklauf/Zeilenvorschub zu lochen.

Die zugehörigen Daten werden auf einem besonderen Streifen erfaßt, und zwar gemäß ihrer in ALGOL zulässigen Darstellung. Nach jedem Eingabewert ist die Kombination Wagenrücklauf/Zeilenvorschub zu lochen. Die Organisation des Datenstreifens, d. h. die lineare Anordnung der Eingabewerte, wird im Zusammenhang mit den Prozeduren für Ein- und Ausgabe besprochen.

Bei der R 300-Variante bezüglich der Menge der Grundsymbole von einem Subset

[1]) Hier ist das aus den Zeichen : und = zusammengesetzte Wort gemeint.

zu sprechen ist insofern nicht ganz korrekt, als auch zwei Symbole, nämlich **code** ('code') und **wait** ('wait') hinzukommen.

'code' steht im ALGOL-Programm für den Körper solcher Prozeduren, die in Maschinensprache programmiert vorliegen und unmittelbar in die als *Objektprogramm* bezeichnete Übersetzung eingefügt werden. Auf diese Weise lassen sich die Vorzüge der ALGOL-Programmierung weitgehend mit denen der Programmierung in der Maschinensprache verbinden. Die einen betreffen die bequeme Handhabung, die anderen die durch Ausnutzung der Besonderheiten der Maschine mögliche Laufzeitoptimierung. Bei häufiger Benutzung eines Programms wird man die bei der Rechenzeit wesentlich ins Gewicht fallenden Algorithmen als Code-Prozeduren bausteinartig in dessen ALGOL-Text einfügen und diesem die Funktion eines Rahmens von Maschinenprogrammteilen zuweisen. Solches Vorgehen entlastet den Übersetzer und gestattet die Einbeziehung von Beiträgen speziell ausgebildeter Programmierer.

'wait' ist ein Symbol, das für Organisationszwecke verwendet wird und die Übersetzung zeitweilig zu unterbrechen gestattet.

Im folgenden werden Einschränkungen und Erweiterungen der in ALGOL allgemein möglichen Sprachkonstruktionen betrachtet.

1. Bezeichnungen sind aus Buchstaben und Ziffern gemäß ALGOL-Syntax zu bilden. Diese werden bei der Übersetzung jedoch nur als verschieden erkannt, wenn sie nicht in den ersten sechs Zeichen übereinstimmen.

2. Die in der R 300-Variante von ALGOL zugelassenen Zahlen z ergeben sich aus ihrer maschineninternen Darstellung und der Forderung

$$|z| \leqq 10^{99}.$$

'real'-Werte sind zehnstellige Gleitkommazahlen mit achtstelliger ganzzahliger Mantisse und zweistelligem Exponenten, die beide vorzeichenbehaftet sind. Die Normalisierung der Gleitkommazahlen ist also so getroffen, daß die höchste der acht Mantissenstellen von Null verschieden ist. Als betragsmäßig kleinste dieser Zahlen ergibt sich

$$10^{7} \cdot 10^{-99} = 10^{-92};$$

ihre Gesamtheit ist in der Menge

$$M = \{z\colon 10^{-92} \leqq |z| \leqq 10^{99}\}$$

enthalten. Null hat als 'real'-Größe eine besondere Darstellung mit der Mantisse 0.

'integer'-Zahlen werden als Gleitkommazahlen mit dem Exponenten Null und maximal achtstelliger Mantisse erfaßt; ihr Betrag ist also kleiner als 10^{8}. Ganze Zahlen, welche dieser Einschränkung nicht genügen, können als zehnstellige Gleitkommazahl nur gerundet mit einem positiven Exponenten dargestellt werden. Sie verlieren ihren Charakter als ganze Zahlen und werden als 'real'-Größen weiterbehandelt.

Aus diesen Bemerkungen über die unterschiedliche maschineninterne Erfassung von 'real'- und 'integer'-Größen erkennt man die Bedeutung der Typvereinbarung.

3. Zeichenketten erfordern bezüglich des Begriffs ⟨echte Zeichenkette⟩ (vgl. 3.3.) eine Modifikation. Außer der leeren Zeichenkette kann dies jede Folge aus R 300-Symbolen (vgl. 2.3.3.) sein, die von

$$\sim \quad \approx \quad \approx \quad \blacktriangledown \quad \blacktriangledown\blacktriangledown \quad \triangle$$

verschieden sind. Die Kettenanführungszeichen '(' und ')' dürfen in einer echten Zeichenkette nicht vorkommen.

Beispiele:

'(' ('*das␣ist␣eine␣zeichenkette*: 'begin 123' ')'
'('('*das␣ist␣keine␣zeichenkette*: 'begin 123'')')'

Eine Kette darf höchstens 120 Zeichen enthalten.

4. Besonders bei den Feldern wirken sich die kapazitativen Beschränkungen einer konkreten Maschine aus. In der hier betrachteten ALGOL-Version muß der Absolutbetrag jedes in einem Grenzenpaar auftretenden Wertes kleiner als 10^5 sein.

5. Als Marken (vgl. 3.1.2., S. 92) sind nur Bezeichnungen zugelassen.

6. In einer Laufklausel darf nach 'for' nur eine einfache Variable stehen (vgl. 3.1.3.). Nach Abarbeitung der Laufanweisung behält diese ihren letzten Wert.

7. Zu den in 3.1.2. beschriebenen Standardfunktionen von ALGOL 60 kommen in der R 300-Variante weitere hinzu. Wir erwähnen nur die für uns wesentlichen in Tabelle 3.3. Die Werte von *div* und *res* sind vom Typ 'integer', die aller übrigen hier genannten Standardfunktionen vom Typ 'real'.

8. Weitere *Standardprozeduren* stehen für Ein- und Ausgabe zur Verfügung. Von diesen erwähnen wir nur *read* für das Einlesen von Daten und *print* für die Ausgabe über Schnelldrucker. *read* und *print* dürfen mit beliebig vielen Parametern aufgerufen werden, die sich auf Variable oder Felder beziehen können. Zum Beispiel müßten die Lies-Anweisungen im Programm zur Berechnung von Polynomwerten nach dem Hornerschen Schema (3.2.) in der Form

read (n) und *read* $(a, x\emptyset)$

ausgedrückt werden, wenn mit Rücksicht auf das unter Punkt 1. Gesagte für das Koeffizientenfeld die Bezeichnung a vereinbart wird. Bei der Datenniederschrift sind die Komponenten eines Feldes als Folge gemäß der lexikographischen Ordnung der Indextupel zu notieren; bei einer Matrix bedeutet das ein Aneinanderfügen der Zeilen. Das entspricht der in 3.2. getroffenen Konvention und würde bei dem betrachteten Programm die auf S. 103 angegebene Datenorganisation zur Folge haben. Beim Eingabevorgang wird geprüft, ob die in der ALGOL-Notation

ausgedrückten Werte in ihrem Typ (vgl. die Bemerkung im Anschluß an die Typbestimmung der Potenz in 3.1.2.) mit dem Typ der Variablen, denen sie zugewiesen werden, übereinstimmen. Ist das nicht der Fall, so erfolgt eine Fehlermeldung; diese unterbleibt bei Zuweisung von 'integer'-Werten an Variable vom Typ 'real'.

R 300-ALGOL-Notation	Argumenttyp	übliche mathematische Bezeichnungsweise bzw. Bedeutung
tan(x)	'real'	tan x
arcsin(x)	'real'	arcsin x (Hauptwert)
arccos(x)	'real'	arccos x (Hauptwert)
arc(x,y)	'real'	im Bogenmaß ausgedrückter Polarwinkel eines Punktes mit den kartesischen Koordinaten x, y: $0 \leqq \text{arc}\,(x, y) < 2\pi$; arc (0, 0) nicht definiert
div(i,k)	'integer'	$\text{sgn}\left(\frac{i}{k}\right) \cdot \left[\left\|\frac{i}{k}\right\|\right]$
res(i,k)	'integer'	$i - \text{sgn}\left(\frac{i}{k}\right)\left[\left\|\frac{i}{k}\right\|\right] \cdot k$ $(= i - div(i,k) * k)$
max(x1, x2,...,xn)	'real'	Maximum der Größen $x_1, x_2, \ldots, x_n$
min(x1, x2,...,xn)	'real'	Minimum der Größen $x_1, x_2, \ldots, x_n$

Tabelle 3.3

Mit Hilfe von *print* können außer Daten, die etwa als Resultate anfallen, auch Zeichenketten gedruckt werden. Beispielsweise wäre es informativ, wenn im ALGOL-Programm zur Bestimmung der reellen Wurzeln einer quadratischen Gleichung (3.1.3.), falls die logische Variable *test* den Wert 'true' hat, nicht nur zum Programmende gesprungen, sondern auch noch der Text

keine reellen Wurzeln

ausgegeben würde. In der R 300-Version ist das über Schnelldrucker durch die Anweisung

print('('*keine*␣*reellen*␣*wurzeln*')')

realisierbar. Allgemein erscheint beim Aufruf von *print* eine Liste von Argumenten, die sich auf Variable, Felder und Zeichenketten beziehen können. Variable werden in der Reihenfolge ihres Auftretens mit ihren Werten nebeneinander auf eine Zeile gedruckt, sofern nicht aus Gründen des Platzbedarfs eine zweite folgen muß. Die Redeweise, daß Zeichenfolgen „auf eine Zeile" gedruckt werden, soll hier immer in diesem Sinne verstanden werden. Tritt der Name eines Feldes auf, wird bei dessen Ausgabe stets eine neue Zeile begonnen. Die Werte der darin enthaltenen indizierten Variablen erscheinen in lexikographischer Ordnung der Indextupel, und zwar diejenigen auf einer Zeile, welche in den ersten $n-1$ Zeigern übereinstimmen. Diese werden, falls $n > 1$, in einer Zeile vor der entsprechenden Gruppe indizierter Vari-

abler gedruckt. Bei einer Matrix würde sich diese Information auf die Angabe des Zeilenindex reduzieren.

Bei jedem Aufruf von *print* wird eine neue Zeile begonnen. Die Anweisung

print('('')') oder *print*('('␣')')

veranlaßt die Einfügung einer Leerzeile in das Druckbild.

'real'-Größen werden in einem normierten ALGOL-Format mit 14 Zeichen ausgegeben; zum Beispiel der Näherungswert 3.1415927 von π in der unter dem Strich angegebenen Form:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		.	3	1	4	1	5	9	2	7	#		0	1

In dieser halblogarithmischen Darstellung gilt also für die Mantisse m

$$0.1 \leqq |m| \leqq 1 - 10^{-8};$$

für den Exponenten sind wegen der unter Punkt 2. genannten Einschränkung zwei Stellen (Position 13 und 14) vorgesehen. Position 1 und 12 sind für das Vorzeichen von Mantisse bzw. Exponent reserviert; sie erscheinen als Leerstellen, wenn diese positiv sind, sonst wird ein Minuszeichen gedruckt. Auf Platz 2 befindet sich der Dezimalpunkt, auf Platz 11 das Basissymbol #. Zwei hintereinander auf einer Zeile zu druckende Werte sind durch drei Leerstellen getrennt.

Wir schließen die Bemerkungen zu *print* mit einem Beispiel ab. In dem betrachteten Programm zur Berechnung der reellen Wurzeln einer quadratischen Gleichung sollen – falls solche vorhanden sind – die Resultate mit folgenden Textdruck ausgegeben werden:

Wurzeln der Gleichung $x*x+ax+b=0$
$w1=\ldots$
$w2=\ldots$

Dabei sollen an Stelle von a und b sowie rechts vom Gleichheitszeichen bei $w1$ und $w2$ die jeweiligen aktuellen Werte erscheinen. Die im Programm notierte Anweisung *Drucke*($w1$,$w2$) müßte dann durch die folgenden ersetzt werden:

```
print('('wurzeln der gleichung x*x+')',a,'('x+')',b,'('=0')');
print('('w1 =')',w1);
print('('w2 =')',w2);
```

Bei der Ausgabe über Schnelldrucker sind zwei Druckbreiten zu 146 bzw. 70 Zeichen pro Zeile wählbar. Alle Buchstaben erscheinen dabei als Majuskeln; um den Buchstaben O von der Ziffer Null unterscheiden zu können, wird diese als Sechseck stilisiert wiedergegeben.

9. Die folgenden Bemerkungen betreffen Vereinbarungen und Aufruf von Prozeduren in der R 300-Version, im besonderen die Aktualisierung formaler Parameter. Zunächst sei noch einmal darauf hingewiesen, daß diese bei der Vereinbarung sämtlich zu spezifizieren sind. Beim Aufruf müssen Art und Typ der formalen und

aktuellen Parameter entsprechend übereinstimmen mit der Ausnahme, daß Parameter vom Typ 'real' durch Größen vom Typ 'integer' ersetzt werden können. Wegen des Umstands, daß Variable, die im Werteteil auftreten, in der Prozedur wie lokale Größen behandelt werden, ist es bei diesen auch zulässig, zugehörige Parameter vom Typ 'integer' mit 'real'-Größen zu aktualisieren.

Formale Parameter, die im Benennungsteil mit dem Grundsymbol 'procedure' spezifiziert wurden, dürfen nicht mit Namen von Standardprozeduren für Ein- und Ausgabevorgänge aufgerufen werden. Ein formaler Parameter, der nicht im Werteteil erscheint, darf – wenn er eine Variable vertritt – nur durch eine Variable (vgl. die entsprechende Definition in 3.1.3.), sonst nur durch eine Bezeichnung ersetzt werden. Eine Ausnahme macht der bisher noch nicht aufgetretene Fall, daß ein formaler Parameter *p* für eine Zeichenkette steht und dann im Benennungsteil mit dem Spezifikationszeichen 'string' erscheinen muß. Für *p* kann beim Aufruf der Prozedur eine beliebige Zeichenkette gemäß der Bestimmung in Punkt 3. eingesetzt werden.

Formale Parameter des Werteteils dürfen mit Ausdrücken aktualisiert werden. Dieses gilt auch für die Parameter von *print*.

Die Bemerkungen dieses Abschnitts zur R 300-Variante von ALGOL sind nicht vollständig, dürften aber ausreichen, um die in den Praktika gestellten Programmieraufgaben zu lösen. Der folgende Abschnitt beschreibt kurz das Einfahren der Programme, speziell die Fehlererkennung.

3.4.2. Fehlererkennung

Bei der Übersetzung wird ein in einer problem- oder maschinenorientierten Programmiersprache formuliertes Programm ganz oder in Teilen in den Hauptspeicher der EDVA übernommen. Der ebenfalls dort gespeicherte Compiler analysiert dieses sogenannte Quellenprogramm und baut dabei das Objektprogramm auf, dessen Befehle im Hauptspeicher abgelegt werden. Unter anderem ist es erforderlich, den in den Vereinbarungen auftretenden Größen Speicherplätze zuzuweisen, d. h. die symbolischen Adressen der Bezeichnungen in Speicheradressen umzuwandeln. Dazu wird ein Adreßbuch angelegt, dem bei der Erzeugung der Maschinenbefehle die einem Namen entsprechende Speicheradresse entnommen wird.

Die Übersetzung eines R 300 ALGOL-Programms erfolgt in zwei Durchgängen, und zwar beim ersten Paß sequentiell durch Einlesen des Lochbandes bis zur jeweils nächsten Lochkombination Wagenrücklauf/Zeilenvorschub. Vor der Verarbeitung wird jede Zeile, mit einer Nummer versehen, über den Schnelldrucker ausgegeben. Das Ergebnis dieser Übersetzungsphase ist ein Pseudoprogramm und unter Umständen die Feststellung gewisser Fehler. Diese werden nach der Nomenklatur der folgenden Übersicht zwischen den Zeilen der Schnelldruckerprotokolls angezeigt, und zwar dort, wo sie sich zum ersten Mal bemerkbar gemacht haben.

Fehler, die beim ersten Paß erkannt werden (nach [45])

Fehlermeldung	*Erläuterung*
UEBERLAUF 01	Speicherplatz zur Aufnahme der Listen nicht ausreichend, Abbruch der Übersetzung.
UEBERLAUF 02	Programmstruktur zu kompliziert, Abbruch der Übersetzung.
UEBERLAUF 03	Abbruch der Übersetzung als Nachwirkung früher aufgetretener Fehler, eventuell Maschinenfehler.
FEHLER 04	Zahlenaufbau falsch oder Zeichen . bzw. # in falschem Zusammenhang.
FEHLER 05	Unzulässiges ALGOL-Zeichen verwendet.
FEHLER 06	Marke ist kein Name oder wurde im Block mehrfach zur Markierung verwendet.
FEHLER 07	Klammerstruktur falsch oder 'begin' bzw. 'end' fehlt.
FEHLER 08	Name in unzulässigem Zusammenhang.
FEHLER 09	Syntaktischer Fehler in Ausdruck oder Wertzuweisung.
FEHLER 10	In Laufanweisung keine einfache Variable zwischen 'for' und := oder kein := da.
FEHLER 11	Aufbau einer Prozedurvereinbarung falsch oder Prozedurname bereits vereinbart.
FEHLER 12	Wortsymbol 'code' in falschem Zusammenhang.
FEHLER 13	Spezifikationssymbol 'procedure' in falschem Zusammenhang.
FEHLER 14	Formaler Parameter ist nicht spezifiziert.
FEHLER 15	Vereinbarung nicht am Blockanfang.
FEHLER 16	Nachwirkung früher aufgetretener Fehler, evtl. Maschinenfehler, Weiterlauf.
FEHLER 17	Wie Fehler 16, Abbruch der Übersetzung.
FEHLER 18	Wortsymbol 'comment' in falschem Zusammenhang.
FEHLER 19	Wortsymbol 'else' in falschem Zusammenhang.
ALGOL-TEXT FALSCH	Kette länger als 120 Zeichen oder Parametertrennzeichen schließt nicht mit :(oder nicht verwertbares Zeichen vom Lochband gelesen. Abbruch der Übersetzung.

Mit der Meldung des Fehlers wird eine vierstellige Adresse ausgegeben, welche

die Stelle im Compiler angibt, wo dieser ermittelt wurde. Der erste Paß kann mit der Fehleranzeige

DEKL? . . .

enden, in der die Punkte für einen Namen stehen, der an einer Stelle des Programms auftrat, aber dort nicht vereinbart war.

Als Beispiel konstruieren wir aus dem ALGOL-Text des in 3.1.2. behandelten Euklidischen Algorithmus ein fehlerhaftes Programm, in dem wir x nicht vereinbaren, in der bedingten Anweisung zweiter Art vor 'else' ein Semikolon setzen und das Multiplikationszeichen in dem Produkt $q \times y$ auslassen. Über den Schnelldrucker erhalten wir das Protokoll der Abb. 3.11a. Die Übersetzung wurde vorzeitig mit der Anzeige des falschen Semikolons vor 'else' und einer unzulässigen Wertzuweisung abgebrochen. Diese bezieht sich auf die nicht vereinbarte Variable x. Nach Korrektur der beiden gemeldeten Fehler erfolgte der Ausdruck der Abb. 3.11b, dem zu entnehmen ist, daß qy vom Compiler als nicht vereinbarte Variable (und nicht als Produkt $q \times y$; vgl. S. 89) interpretiert wurde. Die Korrektur dieses Fehlers liefert das funktionstüchtige Programm der Abb. 3.11c.

Wenn sich im ersten Paß keine Fehler ergeben haben, wird aus dem nunmehr vorliegenden Pseudoprogramm im zweiten Paß ein Objektprogramm erzeugt und dabei über den Schnelldrucker eine Liste ausgegeben, die jedem Semikolon des ALGOL-Programms die folgende interne Adresse des Objektprogramms zuordnet. Auf diese Liste beziehen sich die Fehlermeldungen des zweiten Passes; sie sind von der Form

ZEILE . . . SEMIKOLON . . . FEHLER . . . BEI ADRESSE . . .

```
01  'BEGIN'
02    'INTEGER'M,N,Y,Q,R;
03     READ(M,N);
04    'IF'M'NOTLESS'N'THEN''BEGIN'X:=M;Y:=N'END';
05                      'ELSE''BEGIN'X:=N;Y:=M'END';
    FEHLER 19                                  V150
    FEHLER 09                                  '932
                                                    PROGRAMM FEHLERHAFT
```

Abb. 3.11a

```
01  'BEGIN'
02    'INTEGER'M,N,X,Y,Q,R;
03     READ(M,N);
04    'IF'M'NOTLESS'N'THEN''BEGIN'X:=M;Y:=N'END'
05                      'ELSE''BEGIN'X:=N;Y:=M'END';
06  L:Q:=ENTIER(X/Y);
07     R:=X-QY;
08    'IF'R>0'THEN''BEGIN'X:=Y; Y:=R;
09    'GOTO'L       'END';
10     PRINT(Y);
11  'END'
    DEKL.?  QY       32  00  0  9963
                                                    PROGRAMM FEHLERHAFT
```

Abb. 3.11b

```
01  'BEGIN'
02   'INTEGER'M,N,X,Y,Q,R;
03    READ(M,N);
04   'IF'M'NOTLESS'N'THEN''BEGIN'X:=M;Y:=N'END'
05                      'ELSE''BEGIN'X:=N;Y:=M'END';
06  L:Q:=ENTIER(X/Y);
07    R:=X-Q*Y;
08   'IF'R>0'THEN''BEGIN'X:=Y; Y:=R;
09   'GOTO'L        'END';
10    PRINT(Y);
11  'END'

                        ZEILE/SEMIKOLON  < >  ADRESSE          E025      1

   2  1  0457     3  1  0475     4  1  0505     5  1  0535     5  2  0547
   6  1  0571     7  1  0601     8  1  0625     8  2  0637     9  1  0649
  10  1  0661

                      ARBEITSBEREICH VON 0691 BIS R999

  DATEN:
  113     91


       1
```

Abb. 3.11c

Hinter ZEILE erscheint die dreistellige Nummer der Zeile des vom Schnelldrucker ausgegebenen ALGOL-Programms, in welcher der Fehler auftritt. Dieser wird weiter lokalisiert durch die folgende Zahl, welche die Nummer des Semikolons dieser Zeile bedeutet, nach dem er zu suchen ist. Nach FEHLER wird der Fehlertyp gemäß der Nomenklatur der folgenden Übersicht angezeigt, und die letzte Angabe enthält die Adresse der Stelle des Objektprogramms, bei welcher der Fehler festgestellt wurde.

Fehler, die beim zweiten Paß erkannt werden (nach [45])

Fehlergruppe	*Erläuterung*
0	Hauptspeicher reicht nicht aus, um Objektprogramm aufzunehmen.
1	Syntaktischer Fehler im ALGOL-Text.
2	Typ von Variablen oder Teilausdrücken ist mit dem programmierten Operationszeichen nicht verträglich.
3	Bei Speicherung vorkommende Typen sind nicht verträglich.
4	Art und Typ aktueller Parameter sind mit Standardfunktion nicht verträglich.

5	Unzulässige Namen in Verteilerliste.
6	Anzahl aktueller Parameter falsch.
7	Unzulässige aktuelle Parameter für Eingabeanweisungen.
8	Ausdruck tritt als linke Seite auf.
9	Syntaktischer Fehler im ALGOL-Text (in dem zur Übersetzung benutzten Kellerspeicher fehlen Informationen).

Sind die beim ersten und beim zweiten Paß erkannten Fehler eliminiert, so können weitere noch beim *aktuellen Lauf* des Objektprogramms in Erscheinung treten und zum Abbruch der Rechnung führen. Das würde zum Beispiel der Fall sein, wenn der für eine Variablenbelegung erforderliche Wert auf dem Datenband nicht vorhanden ist oder Typ bzw. Größenanforderung nicht entsprechen. Über den Schnelldrucker wird dann eine der folgenden Fehlermeldungen ausgegeben:

Fehlermeldung beim aktuellen Lauf (nach [45])

Fehlermeldung	*Erläuterung*
ZELLE LEER	Die zur Rechnung benutzte Variable ist unbewertet.
ADRESSE > 39999	Eine berechnete Adresse ist im Hauptspeicher nicht realisierbar.
DIVISOR = 0	Division durch Null versucht.
HS VOLL	Speicherplatz reicht zur Fortsetzung der Rechnung nicht aus.
SPRUNG IN BLOCK	Ein unzulässiger Sprung ins Innere eines Blocks sollte ausgeführt werden.
FELDGRENZEN	Berechnete Feldgrenzen sind unzulässig (links größer als rechts).
FELDLAENGE	Ein vereinbartes Feld hat mehr als 3000 Komponenten.
PARAMETER TYP	Art und Typ formaler Parameter und zugeordneter aktueller Parameter sind nicht verträglich.
AUSDRUCK ALS PARAMETER	Der einem Namensparameter zugeordnete aktuelle Parameter ist ein Ausdruck; nach den hier geforderten Einschränkungen muß der aktuelle Parameter ein Name sein.
PARAMETERZAHL	Die Anzahl der aktuellen Parameter und der formalen Parameter stimmt nicht überein.
INDEXANZAHL	Die Anzahl der Indizes einer indizierten Variablen stimmt nicht mit der durch die zugeordnete Feldvereinbarung bestimmten Dimension überein.
INDEX ZU KLEIN	Ein Index einer indizierten Variablen liegt unterhalb der unteren Feldgrenze oder ein Verteilerindex ist nicht positiv.

INDEX ZU GROSS	Ein Index einer indizierten Variablen liegt oberhalb der oberen Feldgrenze oder ein Verteilerindex ist zu groß.
WERT UNBESTIMMT	Das Resultat einer Rechnung ist nicht definiert.
ARGUMENT < 0	Ein als Argument einer Standardfunktion auftretender aktueller Parameter ist negativ, obwohl die Funktion für negative Argumente nicht definiert ist.
ZAHL ZU GROSS	Der Absolutbetrag eines Resultats ist zur Speicherung zu groß.
KANAL NR.	Die Kanalnummer einer Ein- oder Ausgabeanweisung ist nicht verwendbar.
SEL.NR.	Die angegebene Selektornummer ist nicht definiert.

3.4.3. Beispiele und Übungen

1. Das Programm

```
'begin'
  'integer'i;
  read(i); print(i);
'end'
```

wurde fehlerfrei übersetzt. Beim aktuellen Lauf mit dem Eingabewert

```
123456789
```

erfolgte eine Fehlermeldung. Warum?

2. Man vergleiche und kommentiere die Schnelldruckerprotokolle der Abb. 3.12 und 3.13. Der aktuelle Lauf erfolgte mit dem Eingabewert

```
87654321
```

```
01  'BEGIN'
02   'INTEGER' I;
03    READ(I);
04    I:=2*I;  PRINT(I);
05  'END'


                    ZEILE/SEMIKOLON  <  >  ADRESSE

   2  1  0381     3  1  0393     4  1  0411     4  2  0423

                 ARBEITSBEREICH VON 0453 BIS R999

   DATEN:
   87654321


          FEHLER BEI ADRESSE Y743                ZAHL ZU GROSS
```

Abb. 3.12

```
01  'BEGIN'
02   'INTEGER'I; 'REAL' X;
03    READ(I);
04    X:=2*I;  PRINT(X);
05  'END'

                    ZEILE/SEMIKOLON  < >  ADRESSE

  2  1  0397    2  2  0397    3  1  0409    4  1  0427    4  2  0439

                  ARBEITSBEREICH VON 0469 BIS R999

  DATEN:
  87654321


 ·17530864# 09
```

Abb. 3.13

3. Beim aktuellen Lauf des Programms

```
'begin'
print(1/0.00000001)
'end'
```

wurde über Schnelldrucker die Fehlermeldung

DIVISOR = 0

ausgegeben. Das Programm

```
'begin'
print(1/# − 8)
'end'
```

lieferte den Ergebnisdruck

.10000000 #09

Warum?

```
01  'BEGIN'
02   'INTEGER'I; 'REAL'X;
03    I:=X:=3.14;
04    PRINT(I,X);
05  'END'

                    ZEILE/SEMIKOLON  < >  ADRESSE          E018     1

 ZEILE   2     SEMIKOLON  2     FEHLER 03  BEI ADRESSE 0403

   2  1  0397    2  2  0397    3  1  0415    4  1  0433

                                                  PROGRAMM FEHLERHAFT
```

Abb. 3.14

4. Das Schnelldruckerprotokoll der Abb. 3.14 weist einen Programmfehler aus, der im zweiten Paß erkannt wurde, und zwar zwischen dem zweiten Semikolon der zweiten Zeile und dem ersten der dritten. Gegen welche syntaktische Vorschrift wurde verstoßen?

5. Das Schnelldruckerprotokoll der Abb. 3.15 betrifft einen aktuellen Lauf des in 3.1.3. betrachteten Programms zur Lösung einer quadratischen Gleichung. Ferner wurden die in 3.4.1. unter Punkt 8. formulierten Textkommentare berücksichtigt.

6. Das Protokoll der Abb. 3.16 enthält die Berechnung eines Polynomwertes nach dem Hornerschen Schema. Das Programm ist so angelegt, daß mehrere aus Polynomgrad, -koeffizienten und Argumentstelle bestehende Datensätze nacheinander eingelesen und bearbeitet werden können. Der letzte Eingabewert ist eine Null. Wird dieser der Variablen n zugewiesen, so erfolgt Sprung zum Programmende. Der aktuelle Lauf bezieht sich auf die Berechnung der Legendreschen Polynome ersten, zweiten und dritten Grades an der Stelle 0.5. Das Programm ist syntaktisch einwandfrei und liefert für

```
01  'BEGIN'
02    'REAL' A,B,D,W1,W2;
03    'BOOLEAN' TEST;
04     READ(A,B);
05     PRINT('('WURZELN DER QUADRATISCHEN GLEICHUNG')');
06     PRINT('('X*X+('),A,'('),*X+('),B,'(')=0')');
07      D:=A*A-4*B;
08      TEST:= D<0;
09     'IF'TEST'THEN''BEGIN'
10       PRINT('('SIND NICHT REELL')'); 'GOTO'L
11                    'END';
12     'IF'D=0'THEN' W1:=W2:= -A/2
13           'ELSE''BEGIN'          D:=SQRT(D);
14       W1:=(-A+('IF'A>0'THEN'-1'ELSE'1)*D)/2;
15       W2:= B/W1   'END';
16      PRINT('('W1=')',W1,'('W2=')',W2);
17  L:'END'

                    ZEILE/SEMIKOLON  <  >  ADRESSE          E010      1

   2   1  0561     3   1  0561     4   1  0579     5   1  0591     6   1  0627
   7   1  0669     8   1  0699    10   1  0717    11   1  0729    13   1  0801
  14   1  0885    15   1  0903    16   1  0933

                  ARBEITSBEREICH VON 0973 BIS R999

   DATEN:
   -5.2     1

WURZELN DER QUADRATISCHEN GLEICHUNG
X*X+(   -.52000000# 01   )*X+(    .10000000# 01    )=0
W1=     .50000000# 01    W2=     .20000000# 00
```

Abb. 3.15

den Grad 1 und 2 die gewünschten Polynomwerte. Für $n=3$ erfolgt eine die Feldgrenzen von A betreffende Fehlermeldung. Diese wird dadurch verursacht, daß von den durch arithmetische Ausdrücke gegebenen Feldgrenzen dann die untere größer als die obere ist. Wir werden auf dieses Programmbeispiel in 3.5. zurückkommen.

```
01  'BEGIN'
02    'INTEGER'   N;
03  F1: READ(N);  'IF'N=0'THEN''GOTO'F2;
04        'BEGIN'
05          'INTEGER'I;    'REAL' X0,P;
06          'ARRAY' A[3*(N-1)*(N-2)/2:(N-3)*(2-3*N)/2];
07           READ(A,X0);
08            P:=A[N];
09           'FOR'I:=N-1'STEP'-1'UNTIL'0'DO' P:=A[I]+X0*P;
10            PRINT(P) ;     PRINT('(' ')');
11           'GOTO'F1
12        'END';
13  F2:'END'
```

```
                    ZEILE/SEMIKOLON  < >  ADRESSE          E206      1

  2   1   0546     3   1   0558     3   2   0588     5   1   0606     5   2   0606
  6   1   0756     7   1   0774     8   1   0810     9   1   1002    10   1   1014
 10   2   1026    12   1   1044

                ARBEITSBEREICH VON 1122 BIS R999

 DATEN:
 1     0     1     0.5     2     -0.5     0     1.5     0.5     3     0
 -1.5     0     2.5     0.5     0

.50000000₈ 00

-.12500000₈ 00

          FEHLER BEI ADRESSE 0P44                    FELDGRENZEN

                      VARIABLENLISTE                          E206

          *B0528     0000000001      0368
          N          0000000300      0384
          *B0528     0000000000      0400
          I          00 LEER 00      0416
          P          00 LEER 00      0432
          X0         00 LEER 00      0448
          *FELD*     0000000000      0464
          000000     0000000003      0480
          ,,         00 LEER 00      0496
          ::B0528    0000000000      0512
```

Abb. 3.16

3.5. Syntax und Semantik

In den vorangegangenen Betrachtungen war gelegentlich von der Interpretation eines ALGOL-Textes die Rede und davon, daß gewisse syntaktisch richtige Sprachstrukturen von einer EDVA nicht akzeptiert werden. Solche Fragen gehören zur *Semantik* einer Programmiersprache, deren Verhältnis zur Syntax hier kurz erörtert werden soll.

In 3.1.1. haben wir eine formale Sprache über einem Alphabet Σ als eine Teilmenge L von Σ^* und die Gesamtheit von Aussagen, welche die Bestimmung von L betreffen, als deren Syntax erklärt. Die Syntax von ALGOL 60 ist bezüglich des in 3.1. beschriebenen Alphabets der Grundsymbole im wesentlichen durch die Definitionen der Backusschen Normalform gegeben. Die in dieser Formulierung enthaltene Einschränkung bezieht sich darauf, daß weitere Syntaxregeln zu beachten sind, beispielsweise die, daß jede in einem Programm vorkommende Variable in diesem auch vereinbart sein muß. Im weiteren verstehen wir unter L bei der formalen Sprache ALGOL 60 die Menge der syntaktisch richtigen Programme im Sinne bisheriger Begriffsbestimmung (vgl. 3.2., S. 107).

Eine formale Sprache heißt allgemein eine *Programmiersprache*, wenn jedem Element von L eine Bedeutung zugeordnet ist. Die Elemente von L werden dann *Programme* genannt und sind somit per definitionem syntaktisch richtig. In welchem Sinne ist ALGOL 60 eine Programmiersprache? Um das Wesen der Bedeutungszuordnung sinnfällig zu machen, erörtern wir die Frage bezüglich der R 300-Variante und betrachten noch einmal das Programmbeispiel 6 von 3.4.3. Um das Wesentliche hervorzuheben, reduzieren wir es auf das folgende ALGOL-Programm zur Berechnung eines Polynomwertes nach dem Hornerschen Schema:

```
'begin'
  'integer'n;
  read(n);
  'begin'
    'integer'i; 'real' xØ, p;
    'array'a[3*(n-1)*(n-2)/2:(n-3)*(2-3*n)/2];
    read(a, xØ);
    p:=a[n];
    'for'i:=n-1'step'-1'until'Ø'do'
    p:=a[i]+xØ*p;
    print(p)
  'end'
'end'
```

Dieses Programm ist syntaktisch richtig und wird nacheinander mit folgenden Eingabedaten gerechnet:

$n := 1,\ a[Ø] := Ø,\ a[1] := 1,\ xØ := Ø.5$
$n := 2,\ a[Ø] := -Ø.5,\ a[1] := Ø,\ a[2] := 1.5,\ xØ := Ø.5$
$n := 3,\ a[Ø] := Ø,\ a[1] := -1.5,\ a[2] := Ø,\ a[3] := 2.5,\ xØ := Ø.5$

Diese Datensätze kann man als Wörter eines Eingabealphabets Δ betrachten, denen bei Ausgabe über Schnelldrucker folgende Wörter über einem Alphabet Φ zugeordnet sind:

Eingabewort	Ausgabewort
1, Ø, 1, Ø.5	.50000000 # 00
2, −Ø.5, Ø, 1.5, Ø.5	−.12500000 # 00
3, Ø, −1.5, Ø, 2.5, Ø.5	FEHLER FELDGRENZEN

Beim aktuellen dritten Lauf erfolgte eine Fehlermeldung, die in der Übersicht (S. 126) verkürzt wiedergegeben wurde. Ihre Ursache haben wir in 3.4.3. erörtert. Es lassen sich beliebig viele Wörter des Eingabealphabets bilden, die aus diesem oder anderem Grunde zur Anzeige eines Fehlers führen. Wir wollen in diesem Fall sagen: Das Programm hat für das Eingabewort keine Bedeutung. Ansonsten sei diese durch das Ausgabewort gegeben. Die Festlegung der Bedeutung eines Programms macht die Semantik einer Programmiersprache aus. Dieser Begriff erfährt in unserer Betrachtung durch Einbeziehung einer konkreten EDVA eine gegenständliche Interpretation, die sich abbildungstheoretisch so erfassen läßt:

Zugrunde gelegt wird eine formale Sprache L, $\mathsf{L} \subseteq \Sigma^*$. Neben dem Alphabet Σ wird ein *Eingabealphabet* Δ und ein *Resultatalphabet* Φ nebst den darüber bildbaren Wortmengen Δ^* bzw. Φ^* betrachtet. x sei ein Element von L (ein Programm) und $y \in \Delta^*$. Wenn x für y eine Bedeutung hat, so drückt sich diese in einem Wort über Φ, d. h. in einem Element von Φ^* aus. Die Semantik reduziert sich also auf eine Funktion f, die auf einer Teilmenge M von $\mathsf{L} \times \Delta^*$ definiert ist und deren Wertebereich in Φ^* liegt:

$$M \subseteq \mathsf{L} \times \Delta^*; \quad f: M \to \Phi^*.$$

Bei der Konzipierung einer Programmiersprache ist die Abgrenzung von Syntax und Semantik bis zu einem gewissen Grade willkürlich. Man könnte zum Beispiel, ohne Wesentliches zu ändern, auch ALGOL-Programme als syntaktisch richtig bezeichnen, in denen nicht alle Variablen vereinbart sind. Die auf diese Weise zu L hinzukommenden Programme könnte man dann durch eine geeignete Bestimmung von f als bedeutungslos wieder aussondern.

4. Ausgewählte Gegenstände der Numerischen Mathematik

In diesem Kapitel erörtern wir einige Methoden der Numerischen Mathematik zur Lösung von Elementaraufgaben, die sich häufig bei der Bearbeitung von Praxisproblemen ergeben. Neben der inhaltlichen Klärung der Sachverhalte findet der algorithmische Aspekt besondere Beachtung. Wichtige Verfahren werden in ALGOL-Prozeduren zusammengefaßt.

4.1. Lösung von Gleichungen

Wir betrachten nur Gleichungen in einer Unbekannten. Diese sind uns durch

$$g(x) = 0 \tag{1}$$

gegeben, wobei g eine auf einer Teilmenge von R definierte reellwertige Funktion bedeutet. Sehen wir von den Fällen ab, wo eine „explizite" Lösung von (1) möglich ist, so haben wir unser Interesse auf die Feststellung der Lösbarkeit und gegebenenfalls die angenäherte Berechnung von Lösungen (Nullstellen) durch ein *schrittweise vorgehendes Verfahren* zu richten. Daraus leiten sich folgende Teilaufgaben ab:

1. Lokalisierung von Lösungen, d. h. Bestimmung von Intervallen, in denen jeweils genau eine derselben enthalten ist.

2. Berechnung einer Lösung mit vorgegebener Genauigkeit und Formulierung dafür geeigneter Abbruchkriterien. Hierbei handelt es sich wesentlich um *Fehlerabschätzungen* bei der Konstruktion von Näherungslösungen.

Das erste dieser beiden Probleme erweist sich als das im allgemeinen schwierigere. Ist eine Nullstelle schon hinreichend gut lokalisiert, so stehen zahlreiche Verfahren zur schrittweise beliebig genauen Berechnung zur Verfügung. Vielfach beruhen diese auf der Annahme, daß es möglich ist, (1) in die Gestalt

$$f(x) = x \tag{2}$$

unter gewissen Voraussetzungen über die Funktion f zu transformieren. Man bezeichnet die *Gleichungen* (1) und (2) als von *erster* bzw. *zweiter Art*. Die Lösung von (2) ist mit der Bestimmung der Fixpunkte der Abbildung f äquivalent. Da eine weit entwickelte Theorie bezüglich der Existenz, Unität und Konstruierbarkeit solcher Fixpunkte zur Verfügung steht, sind Gleichungen zweiter Art im allgemeinen der Lösung zugänglicher als solche vom Typ (1).

4.1.1. Kontrahierende Abbildungen

Wir knüpfen an den in MfL Bd. 4, S. 168, bewiesenen Banachschen Fixpunktsatz an und erörtern ein auf dem Prinzip der kontrahierenden Abbildung beruhendes Verfahren zur iterativen Lösung von (2). Zunächst beweisen wir eine Variante des Fixpunktsatzes, bei welcher die Forderung, daß f eine abgeschlossene Menge in sich abbildet, durch eine in den Anwendungen leicht prüfbare Bedingung ersetzt ist.

Satz 1. *Für $r>0$ und $z\in \mathbf{R}$ sei*

$$\bar{U}_r(z) := [z-r,\, z+r]^1) \tag{3}$$

die abgeschlossene r-Umgebung des Punktes z (vgl. MfL Bd. 4, 1.5.) *und $f\colon \bar{U}_r(z) \to \mathbf{R}$ kontrahierende Abbildung zum Kontraktionsfaktor q $(0 \leqq q < 1)$, d. h., es ist für alle $x_1, x_2 \in \bar{U}_r(z)$*

$$|f(x_1)-f(x_2)| \leqq q\,|x_1-x_2|; \tag{4}$$

ferner gelte

$$|z-f(z)| \leqq (1-q)\,r. \tag{5}$$

Dann besitzt f genau einen Fixpunkt ξ in $\bar{U}_r(z)$, und die mit einem beliebigen Startpunkt $x_0 \in \bar{U}_r(z)$ gebildete Iterationsfolge

$$x_{j+1} := f(x_j) \qquad (j\in \mathbf{N}) \tag{6}$$

konvergiert gegen diesen:

$$\xi = \lim_{j\to\infty} x_j. \tag{7}$$

Es gelten die Fehlerabschätzungen

$$|\xi - x_j| \leqq \frac{q^j}{1-q}\,|x_1-x_0| \tag{8}$$

und

$$|\xi - x_j| \leqq \frac{q}{1-q}\,|x_j - x_{j-1}|. \tag{9}$$

---

[1]) In diesem Band wird, abweichend von den anderen Bänden der MFL, für das abgeschlossene (bzw. offene) Intervall die Klammer [ ] (bzw.] [) verwendet.

Beweis. Zunächst ist die Bildbarkeit der Folge (x_j) gemäß (6) zu zeigen. Da f nur als auf $\bar{U}_r(z)$ definiert angenommen wurde, muß nachgewiesen werden, daß jedes x_j dieser Umgebung angehört.

Nach Voraussetzung ist $x_0 \in \bar{U}_r(z)$. Ferner gilt

$$\bigwedge_x (x \in \bar{U}_r(z) \Rightarrow f(x) \in \bar{U}_r(z)), \tag{10}$$

denn auf Grund von (5) und (4) ist für $x \in \bar{U}_r(z)$

$$|z - f(x)| \leqq |z - f(z)| + |f(z) - f(x)| \leqq (1-q)r + q\,|z - x| \leqq (1-q)r + qr = r,$$

also

$$f(x) \in \bar{U}_r(z).$$

Die Behauptung $x_j \in \bar{U}_r(z)$ für alle $j \in \mathbf{N}$ folgt nunmehr durch Induktion.

Im übrigen entspricht der Beweis von Satz 1 vollständig dem des Satzes in MfL Bd. 4, S. 168; wir rekapitulieren diesen der Vollständigkeit halber.

I. Es wird gezeigt, daß die x_j eine Fundamentalfolge bilden. Wegen (4) ist für $j \geqq 1$

$$|x_{j+1} - x_j| = |f(x_j) - f(x_{j-1})| \leqq q|x_j - x_{j-1}|,$$

und durch mehrfache Anwendung dieser Abschätzung gewinnt man

$$|x_{j+1} - x_j| \leqq q\,|x_j - x_{j-1}| \leqq q^2\,|x_{j-1} - x_{j-2}| \leqq \ldots \leqq q^j\,|x_1 - x_0|.$$

Wir betrachten nun den Abstand $|x_j - x_k|$, wobei ohne Beschränkung der Allgemeinheit $k > j$ angenommen werden kann. Auf Grund der zuletzt gefundenen Abschätzung gewinnt man nach mehrmaliger Anwendung der Dreiecksungleichung

$$\begin{aligned}
|x_j - x_k| &\leqq |x_j - x_{j+1}| + |x_{j+1} - x_k| \\
&\leqq |x_j - x_{j+1}| + |x_{j+1} - x_{j+2}| + |x_{j+2} - x_k| \leqq \ldots \\
&\leqq |x_j - x_{j+1}| + |x_{j+1} - x_{j+2}| + \cdots + |x_{k-1} - x_k| \\
&\leqq (q^j + q^{j+1} + \cdots + q^{k-1})\,|x_1 - x_0| \\
&\leqq q^j |x_1 - x_0| \sum_{\nu=0}^{\infty} q^\nu = \frac{q^j}{1-q} |x_1 - x_0|.
\end{aligned}$$

Wegen $0 \leqq q < 1$ strebt die rechte Seite der Abschätzung gegen Null, wenn j die Folge der natürlichen Zahlen durchläuft, und wir haben $|x_j - x_k| \to 0$ mit $j, k \to \infty$.

II. Wegen der Vollständigkeit von $\mathbf{R}$ konvergiert die Folge (x_j) gegen eine reelle Zahl ξ:

$$\xi = \lim_{j \to \infty} x_j .$$

Offensichtlich ist $\xi \in \bar{U}_r(z)$, da aus $|x_j - z| \leqq r$

$$|\xi - z| = \lim_{j \to \infty} |x_j - z| \leqq r$$

folgt. Wir behaupten: Es ist $f(\xi)=\xi$, d. h., ξ ist Fixpunkt von f. In der Tat hat man wegen (4)

$$|f(\xi)-x_{j+1}|=|f(\xi)-f(x_j)|\leqq q\,|\xi-x_j|,$$

woraus für $j\to\infty$

$$|f(\xi)-\xi|=0,$$

also $f(\xi)=\xi$ folgt.

III. Um zu zeigen, daß es in $\bar{U}_r(z)$ nur einen Fixpunkt gibt, gehen wir von der Annahme $f(\xi_1)=\xi_1$ und $f(\xi_2)=\xi_2$ für Punkte $\xi_1, \xi_2\in\bar{U}_r(z)$ aus und betrachten den Abstand $|\xi_1-\xi_2|$. Es ist

$$|\xi_1-\xi_2|=|f(\xi_1)-f(\xi_2)|=q\,|\xi_1-\xi_2|,$$
$$(1-q)\,|\xi_1-\xi_2|=0,$$

woraus wegen $0\leqq q<1$ sofort $\xi_1=\xi_2$ folgt.

IV. Wir befassen uns noch mit dem Fehler, der bei der Approximation des Grenzwerts ξ durch ein Folgenelement x_j auftritt. Dazu lassen wir k in der Abschätzung

$$|x_j-x_k|\leqq\frac{q^j}{1-q}\,|x_1-x_0|$$

des Beweisteiles I. gegen ∞ streben und erhalten (8). Da die rechte Seite nur von den Anfangselementen x_0, x_1 der Folge (x_j) abhängt, kann die Abschätzung dazu benutzt werden, vor deren weitergehender Berechnung zu entscheiden, wie viele Glieder hinreichend sind, um mit dem zuletzt bestimmten eine vorgegebene Annäherung zu gewährleisten. Man nennt (8) daher eine *Fehlerabschätzung a priori*. Majorisiert man in der Abschätzung

$$|x_j-x_k|\leqq|x_j-x_{j+1}|+|x_{j+1}-x_{j+2}|+\cdots+|x_{k-1}-x_k|$$

von I. die Abstandwerte auf der rechten Seite durch

$$|x_j-x_{j+1}|\leqq q\,|x_{j-1}-x_j|,$$
$$|x_{j+1}-x_j|\leqq q^2|x_{j-1}-x_j|,$$
$$\cdots\cdots\cdots\cdots$$
$$|x_{k-1}-x_k|\leqq q^{k-j}|x_{j-1}-x_j|,$$

so ergibt sich

$$|x_j-x_k|\leqq q\,|x_{j-1}-x_j|\;(1+q+q^2+\cdots+q^{k-j-1})\leqq\frac{q}{1-q}|x_{j-1}-x_j|$$

und für $k\to\infty$ mit Beachtung von (7) die Abschätzung (9). Ob mit x_j eine gewünschte Approximationsgüte erreicht wird, kann hiernach erst entschieden werden, wenn die Iterationsfolge bis zu diesem Element gebildet wurde. (9) heißt aus diesem Grunde eine *Fehlerabschätzung a posteriori*. Es ist plausibel und wird

noch durch Beispiele belegt, daß im allgemeinen (9) als Abbruchbedingung effektiver ist als (8).

Der Satz 1 bleibt unverändert gültig, wenn wir **R** durch einen vollständigen metrischen Raum X und $|x-y|$ durch die darin definierte Entfernung $\varrho(x, y)$ ersetzen. $\bar{U}_r(z)$ bedeutet dann die Menge

$$\{x: x \in X \wedge \varrho(x, z) \leqq r\}.$$

Das hat zur Folge, daß die auf Grund von Satz 1 gegebene Methode zur Lösung der Gleichung (2) auf allgemeinere Gleichungen zweiter Art und insbesondere auch auf Systeme linearer und nichtlinearer Gleichungen in n Veränderlichen ausgedehnt werden kann.

Der konstruktive Teil des Satzes 1 ist die Charakterisierung des Fixpunktes ξ von f, also der Lösung von (2), in $\bar{U}_r(z)$ als Limes der gemäß (6) zu bildenden Iterationsfolge (x_j). Man nennt die näherungsweise Bestimmung von ξ durch Berechnung der x *Methode der sukzessiven Approximationen* oder auch *gewöhnliches Iterationsverfahren*. Der Algorithmus des Verfahrens ist bei vorgegebener Fehlerschranke $\varepsilon > 0$ unter Verwendung von (9) als Abbruchbedingung im PAP der Abb. 4.1 dargestellt. In diesem PAP kommt die indexfreie Wiedergabe der

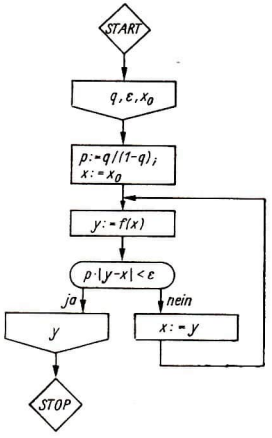


Abb. 4.1

Iterationsfolge in der fortlaufenden Aktualisierung der Variablen x zum Ausdruck: Nach Berechnung der in (9) benötigten Hilfsgröße p wird x zunächst der Startwert x_0 zugewiesen und $y = f(x) = x_1$ bestimmt. Gemäß (9) ist dann zu entscheiden, ob $|\xi - x_1|$ kleiner als ε ist. Wenn ja, wird $y(=x_1)$ als Näherung akzeptiert

und ausgegeben, ansonsten erhält x den Wert x_1, und man tritt mit der Berechnung von $y = x_2$ erneut in den zuvor beschriebenen Rechnungsgang ein.

Dieses Verfahren kann in der folgenden ALGOL-Prozedur *SAP* zusammengefaßt werden:

```
procedure SAP (f, x0, y, q, eps); value q, eps;
real q, eps, x0, y; real procedure f;
  begin
    real p, x;
    p := q/(1 - q); x := x0;
L:  y := f(x);
    if q × abs(y - x) ≧ eps then
    begin x := y; goto L end
  end
```

Die formalen Parameter in *SAP* vertreten in ihrer Abfolge Bezeichnungen für die Funktion in Gleichung (2), den Startpunkt der Iterationsfolge (6), die Variable, welcher schließlich der Näherungswert zugewiesen wird, den Kontraktionsfaktor und die Fehlerschranke. Die letzten beiden Größen werden in den Werteteil aufgenommen und alle formalen Parameter ihrem Typ entsprechend spezifiziert (vgl. 3.3., S. 111). Im Prozedurkörper werden die nur für Zwischenrechnungen benötigten Variablen p und x lokal vereinbart; der Anweisungsteil entspricht vollständig dem PAP der Abb. 4.1 bis auf die Ersetzung der Bedingung durch ihre Negation und die Vertauschung des ja- und des nein-Zweiges.

Wir betrachten ein Beispiel und erläutern daran die Verwendung der Prozedur *SAP* in einem ALGOL-Programm. Es soll der Berührungspunkt (ξ, η) der Parabel $y = 1 + x^2$ und eines von der Abszissenachse tangierten Kreises vom Radius 1 ge-

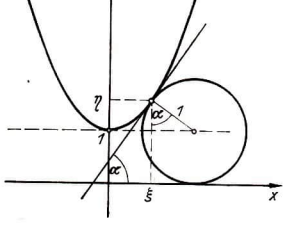


Abb. 4.2

mäß Abb. 4.2 bestimmt werden.[1]) Dieser entnimmt man die Beziehung

$$\tan\alpha = \frac{\xi+\sin\alpha-\xi}{\eta-1} = \frac{\sin\alpha}{\eta-1}$$
$$= \frac{1}{\eta-1}\cdot\frac{\tan\alpha}{\sqrt{1+\tan^2\alpha}}.$$

Andererseits folgt aus der Parabelgleichung für den Tangentenanstieg im Berührungspunkt

$$\tan\alpha = 2\xi \quad \text{und somit} \quad 2\xi = \frac{1}{\xi^2}\frac{2\xi}{\sqrt{1+4\xi^2}}.$$

Für $u := \xi^2$ resultiert daraus die kubische Gleichung

$$u^2(1+4u) = 1.$$

Diese gestattet in der Umgebung $\bar{U}_{1/4}\left(\frac{1}{2}\right)$ die äquivalente Umformung

$$f(u) = \frac{1}{4^{1/3}}(1-u^2)^{1/3} = u.$$

Die Kontraktivität von f auf $\bar{U}_{1/4}\left(\frac{1}{2}\right)$ ergibt sich mit Hilfe des Mittelwertsatzes der Differentialrechnung: Für $\left|u-\frac{1}{2}\right| \leqq \frac{1}{4}$ ist

$$|f'(u)| = \frac{1}{3\cdot 4^{1/3}}\,\frac{2u}{(1-u^2)^{2/3}} \leqq 0{,}55$$

und daher

$$|f(v)-f(u)| \leqq 0{,}55\,|v-u| \quad \left(u, v \in \bar{U}_{1/4}\left(\frac{1}{2}\right)\right).$$

Mit dem Kontraktionsfaktor $q = 0{,}55$ prüfen wir die Bedingung (5) für $z = \frac{1}{2}$ und $r = \frac{1}{4}$. Diese ist erfüllt, denn es ist

$$|z-f(z)| = \left|\frac{1}{2}-f\left(\frac{1}{2}\right)\right| < 0{,}1$$

und

$$r(1-q) = \frac{1}{4}\cdot 0{,}45.$$

Man wird bemerken, daß die wesentliche Vorarbeit für die Anwendung des Satzes 1 darin besteht, die gesuchte Lösung hinreichend gut zu lokalisieren. Um diese mit vorgebbarer Genauigkeit zu bestimmen, schreiben wir ein R 300-ALGOL-Programm, welches die Prozedur *SAP* benutzt. Dazu ist erforderlich, die linke

[1]) Diese Aufgabe wurde R. Rothe, Höhere Mathematik, Teubner, Leipzig 1959, entnommen.

Seite der Gleichung vor dem Aufruf von *SAP* als Funktionsprozedur zu vereinbaren.

```
'begin'
  'real' p, eps, x0, y;
  'real' 'procedure' f(x); 'value' x; 'real' x;
    'begin' f := (1 - x * x) * * 0.33333333/1.5874011 'end';
  'procedure' sap(f, x0, y, p, eps);
      . . ;
  read (p, eps, x0);
  sap (f, x0, y, p, eps); print(y);
'end'
```

Am Anfang des Programms werden außer den Variablen *p, eps, x0, y* die Funktionsprozedur *f* und die das Verfahren der sukzessiven Approximationen realisierende Prozedur *sap* vereinbart. Diese müßte entsprechend ihrer obigen Formulierung vollständig in R 300-ALGOL ausgedrückt werden. Der Anweisungsteil besteht nur aus dem Aufruf von *sap* und dem Ausdruck des Näherungswertes *y*. Bei der Aktualisierung der formalen Parameter der Prozedur bedeuten *f* die vereinbarte Funktion, *x0*, *p* und *eps* die mit der *read*-Anweisung eingelesenen Werte und *y* die im Vereinbarungsteil des Programms so bezeichnete Variable. Das Programm wurde mit den Daten

$$p = 0{,}55; \quad eps = 10^{-6} \quad \text{und} \quad x0 = 0{,}5$$

eingegeben und durch Einfügen einer Druckanweisung in *sap* die Ausgabe sämtlicher bis zum Abbruch berechneten Iterationen veranlaßt. Es ergeben sich die 12 Werte der Tabelle 4.1 (bezüglich der Zahldarstellung vgl. S. 121). Mit Benutzung von $u = \xi^2$ erhält man folgende Näherungswerte für die Koordinaten des Berührungspunktes von Kreis und Parabel:

$$\xi = 0.746119 \qquad \eta = 1.556693$$

i	u
0	.50000000 # 00
1	.57235751 # 00
2	.55189655 # 00
3	.55811965 # 00
4	.55626550 # 00
5	.55682141 # 00
6	.55665505 # 00
7	.55670486 # 00
8	.55668995 # 00
9	.55669441 # 00
10	.55669308 # 00
11	.55669347 # 00

Tabelle 4.1 Iterative Lösung der kubischen Gleichung $u^2(1+4u) = 1$

Bemerkungen

1. Ist f auf der Umgebung $\bar{U}_r(z)$ stetig und in ihrem Inneren differenzierbar, so kann die Kontraktivität der Funktion wie in dem betrachteten Beispiel mit Hilfe des Mittelwertsatzes gezeigt werden, sofern in $U_r(z)$ [1])

$$|f'(x)| \leqq q < 1$$

ist. Falls f in dieser Umgebung sein Vorzeichen nicht ändert, folgt aus

$$\xi - x_j = f(\xi) - f(x_j) = f'(\xi + \vartheta(\xi - x_j))\,(\xi - x_j) \qquad (0 < \vartheta < 1),$$

daß die Differenzen $\xi - x_{j+1}$ und $\xi - x_j$ gleiches oder entgegengesetztes Vorzeichen haben, je nachdem, ob f' positiv oder negativ ist. Die Iterationsfolge konvergiert dann entsprechend einseitig monoton oder oszillierend gegen ξ. Abb. 4.3 gibt dafür eine anschauliche Erklärung. Die Übertragung von $f(x_j)$ als Iteration x_{j+1} auf die x-Achse erfolgt durch Projektion des Schnittpunkts der Geraden $y = f(x_j)$ und $y = x$, wobei es genügt, die dargestellten Streckenzüge zu konstruieren.

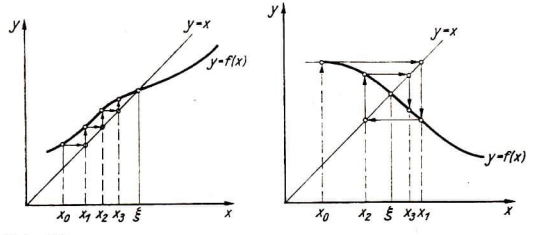


Abb. 4.3

2. In Satz 1 sei $q \leqq \frac{1}{2}$. Dann folgt aus der Abschätzung (9)

$$|\xi - x_j| \leqq |x_j - x_{j-1}|,$$

d. h., die Entfernung zweier aufeinanderfolgender Näherungen x_{j-1}, x_j majorisiert die Entfernung des Punktes x_j von ξ. Man beachte hier die Voraussetzung und schließe nicht fälschlich aus der Übereinstimmung der Werte x_{j-1} und x_j in den ersten Dezimalstellen, daß x_j ein Näherungswert von ξ mit entsprechend vielen gültigen Ziffern ist.

3. Unter den Voraussetzungen von Satz 1 folgt aus den Gleichungen $f(\xi) = \xi$ und $f(x_j) = x_{j+1}$

$$|\xi - x_{j+1}| = |f(\xi) - f(x_j)| \leqq q\,|\xi - x_j| \leqq |\xi - x_j|,$$

[1]) $U_r(z)$ bedeutet das offene Intervall $]z - r, z + r[$.

d. h., der Betrag des absoluten Fehlers der Näherungswerte x_j strebt monoton gegen Null.

4. Die Voraussetzung (5) des Satzes 1 hat zur Konsequenz, daß die Iterationsfolge (x_j) ausgehend von jedem Startelement $x_0 \in \mathring{U}_r(z)$ gebildet werden kann. Das läßt sich auch auf Grund anderer Gegebenheiten sichern. Nehmen wir an, daß f auf einem abgeschlossenen Intervall $[\![a, b]\!]$ definiert ist. In $[\![a, b]\!]$ existiere f', und es sei dort

$$|f'(x)| \leqq q < 1 .$$

Die Existenz einer Lösung ξ der Gleichung (2) wird vorausgesetzt, und es wird angenommen, daß diese Lösung im Intervall

$$I = [\![a + \frac{1}{3}(b-a),\ b - \frac{1}{3}(b-a)]\!]$$

lokalisiert ist. Dann sind die Elemente (6) ausgehend von jedem Startwert $x_0 \in I$ bildbar. Dazu zeigen wir:

$$x \in [\![a, b]\!] \wedge |x - \xi| < \frac{b-a}{3} \Rightarrow f(x) \in [\![a, b]\!] \wedge |f(x) - \xi| < \frac{b-a}{3} .$$

Auf Grund des Mittelwertsatzes ergibt sich unter Beachtung von $f(\xi) = \xi$

$$|f(x) - \xi| = |f(x) - f(\xi)| \leqq q\,|x - \xi| < \frac{b-a}{3}$$

und daraus wegen der Lage von ξ

$$f(x) \in [\![a, b]\!] .$$

Der Satz besagt, daß sich die in der Voraussetzung genannten Eigenschaften von x auf $f(x)$ vererben. Da sie für x_0 zutreffen, folgt durch Induktion ihre Gültigkeit für alle x_j. Nunmehr kann wie beim Beweis von Satz 1 unter Beachtung der Bemerkung 1 gezeigt werden, daß f auf $[\![a, b]\!]$ genau einen Fixpunkt besitzt, $\lim_{j \to \infty} x_j = \xi$ ist und die Abschätzungen (8) und (9) gelten.

Im Sinne der Bemerkung 4 betrachte man auch die

Aufgabe 1. Man beweise: Hat die Gleichung (2) eine Lösung ξ und genügt f in einer gewissen Umgebung $U_r(\xi)$ der Bedingung

$$|f(x) - f(\xi)| \leqq q\,|x - \xi| \quad \text{mit} \quad 0 \leqq q < 1 ,$$

dann ist die Iterationsfolge (6) mit jedem $x_0 \in U_r(\xi)$ bildbar und konvergiert gegen ξ. Weitere Lösungen der Gleichung (2) existieren in $U_r(\xi)$ nicht. Man beachte, daß die an f gestellte Forderung schwächer ist als die Kontraktionsbedingung (4).

4.1.2. Das Newtonsche Verfahren[1])

Der Methode von NEWTON, die auf Gleichungen der Form (1) angewandt wird, liegt folgende Konzeption zugrunde:

Es sei g eine auf einem Intervall I differenzierbare Funktion, deren Ableitung dort nicht verschwindet, und $x_0 \in I$ Näherung für eine in I gelegene Nullstelle ξ von g. Die Tangente an den Graphen von g in x_0 schneidet die x-Achse im Punkt

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

Liegt x_1 in I, so kann man die Konstruktion mit x_1 wiederholen und erhält

$$x_2 = x_1 - \frac{g(x_1)}{g'(x_1)},$$

allgemein mit entsprechenden Voraussetzungen und Bezeichnungen

$$x_{j+1} = x_j - \frac{g(x_j)}{g'(x_j)} \tag{11}$$

(vgl. Abb. 4.4). Es zeigt sich, daß die Folge der Elemente (11) unter gewissen Voraussetzungen uneingeschränkt bildbar ist und gegen eine Nullstelle von g konvergiert. Die Erörterung des Sachverhalts kann wie in MfL Bd. 5, 3.5.2., auf der Grundlage des Banachschen Fixpunktsatzes erfolgen. Wir knüpfen dazu an den Satz 1 an und ersetzen die Forderung (3) des zitierten Textes durch eine der Voraussetzung (5) in Satz 1 entsprechende.

Unter Beibehaltung der g betreffenden Differenzierbarkeitseigenschaften verstehen wir weiterhin unter I die abgeschlossene r-Umgebung einer Stelle z. Dann ist auf $\bar{U}_r(z)$

$$x - \frac{g(x)}{g'(x)} = x \tag{12}$$

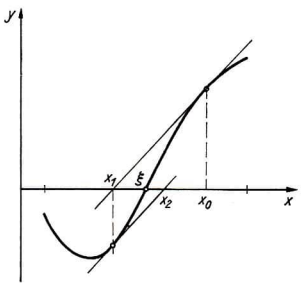


Abb. 4.4

[1]) Auch als Verfahren von NEWTON-RAPHSON bezeichnet.

eine äquivalente Umformung der Gleichung (1) in eine Gleichung vom Typ (2), wobei

$$f(x) = x - \frac{g(x)}{g'(x)} \tag{13}$$

ist. Die Folge der Elemente (11) erweist sich als die mit der Funktion f in (13) gebildete Iterationsfolge (6). Besitzt g in $U_r(z)$ eine zweite Ableitung, so kann die Kontraktivität von f auf Grund der Bemerkung 1 am Schluß von 4.1.1. geprüft werden: Hinreichend ist, daß in $U_r(z)$

$$|f'(x)| = \left|1 - \frac{g'(x)^2 - g(x)g''(x)}{g'(x)^2}\right| = \left|\frac{g(x)\,g''(x)}{g'(x)^2}\right| \leqq q < 1$$

gilt. Der Forderung (5) des Satzes 1 würde in dem betrachteten Spezialfall

$$|z - f(z)| = \left|\frac{g(z)}{g'(z)}\right| \leqq (1-q)r$$

entsprechen. Wir fassen die Ergebnisse in dem folgenden Satz zusammen:

Satz 2. *g'' existiert in $U_r(z)$* $\wedge \bigwedge_{x \in U_r(z)} (g'(x) \neq 0) \wedge$

$$\bigvee_q \left(0 \leqq q < 1 \wedge \bigwedge_{x \in U_r(z)} \left(\frac{|g(x)g''(x)|}{g'(x)^2} \leqq q\right) \wedge \left|\frac{g(z)}{g'(z)}\right| \leqq (1-q)\,r\right)$$

$$\Rightarrow \bigvee_{x \in U_r(z)}^{q} (g(x) = 0).$$

Ist die Voraussetzung des Satzes 2 erfüllt, so kann die auf $U_r(z)$ wohlbestimmte Lösung der Gleichung $g(x) = 0$ nach der Methode der sukzessiven Approximationen gewonnen werden. Dabei gelten die Fehlerabschätzungen (8) und (9).

Als Beispiel betrachten wir die Gleichung

$$g(x) = 3 - e^{x/2} - x \ln x = 0 \quad (x > 0).$$

Um eine erste Orientierung über vorhandene Lösungen zu gewinnen, interpretieren wir diese als Schnittpunktabszisse der Graphen von

$$g_1(x) = 3 - e^{x/2} \quad \text{und} \quad g_2(x) = x \ln x$$

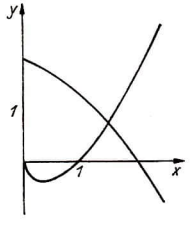


Abb. 4.5

(vgl. Abb. 4.5). Offensichtlich existiert genau eine Lösung im Intervall $1 \leqq x \leqq 2$. Wir versuchen, die Voraussetzungen des Satzes 2 bezüglich der Umgebung $\bar{U}_{1/2}\left(\frac{3}{2}\right)$ zu verifizieren. Es ist

$$g'(x) = -\frac{1}{2}e^{x/2} - \ln x - 1,$$

$$g''(x) = -\frac{1}{4}e^{x/2} - \frac{1}{x};$$

g und g' sind über dem betrachteten Intervall monoton fallend, so daß

$$|g(x)| \leqq 1{,}4 \quad \text{und} \quad |g'(x)| \geqq 1{,}83$$

ist; ferner ist

$$|g''(x)| \leqq 1{,}5 \quad \text{und folglich} \quad \frac{|g(x)\, g''(x)|}{g'(x)^2} \leqq 0{,}7.$$

Wegen $\left|\frac{g(z)}{g'(z)}\right| \leqq 0{,}12$ ist mit $q := 0{,}7$ die Bedingung

$$\left|\frac{g(z)}{g'(z)}\right| \leqq (1-q)\, r$$

erfüllt. Wir benutzen die Prozedur *SAP* zur Lösung der Gleichung. Mit $\varepsilon := 10^{-6}$ und $x_0 := \frac{3}{2}$ erhält man als Folge der sukzessiven Approximationen die in Tabelle 4.2 angegebenen Werte.

j	x_j
0	.15000000 # 01
1	.16115285 # 01
2	.16086781 # 01
3	.16086763 # 01
4	.16086763 # 01

Tabelle 4.2 Iterative Lösung der Gleichung $3 - e^{\frac{x}{2}} - x \ln x = 0$ nach dem Newtonschen Verfahren

Zur Bildung der Iterationsfolge beim Newtonschen Verfahren müssen die Werte $g(x)$ und $g'(x)$ berechnet werden. Im Fall einer Polynomgleichung $P(x) = 0$ bedient man sich dazu des sogenannten *erweiterten Hornerschen Schemas*, das auch darüber hinaus vielfältig eingesetzt werden kann. Es stellt einen Algorithmus dar zur Umordnung eines Polynoms

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

nach Potenzen einer beliebigen Entwicklungsgröße $(x - x_0)$. Auf Grund des Taylorschen Satzes ist das auf genau eine Weise möglich, und für die Koeffizienten in der neuen Darstellung

$$P(x) = b_n(x-x_0)^n + b_{n-1}(x-x_0)^{n-1} + \cdots + b_1(x-x_0) + b_0$$

gilt

$$b_j = \frac{1}{j!} P^{(j)}(x_0), \quad j = 0(1)n. \tag{14}$$

Die b_j ergeben sich durch mehrmalige Anwendung des in 2.2. behandelten einfachen Hornerschen Schemas mit dem Argument x_0, indem man die nach 2.2. (8) ermittelten Größen

$$a_n', a_{n-1}', \ldots, a_2', a_1'$$

als Koeffizienten eines Polynoms Q_{n-1} vom Grad $n-1$ auffaßt und zunächst mit diesem den Horner-Algorithmus für x_0 abarbeitet. Die „unter dem Strich" erscheinenden Werte seien mit a_i'', $i = 1(1)n$, bezeichnet. Von diesen werden entsprechend a_n'', a_{n-1}'', …, a_2'' zur Bildung eines Polynoms Q_{n-2} vom Grad $n-2$ benutzt, mit dem für x_0 ein weiterer Horner-Schritt zu rechnen ist, und so fort. Auf diese Weise erhält man folgendes Schema, dessen Diagonalterme – wie anschließend gezeigt wird – die Taylorkoeffizienten (14) sind:

	a_n	a_{n-1}	a_{n-2}	…	a_1	a_0	
x_0		$a_n' x_0$	$a_{n-1}' x_0$	…	$a_2' x_0$	$a_1' x_0$	
	a_n'	a_{n-1}'	a_{n-2}'	…	a_1'	a_0' $(=b_0)$	nullter Schritt
x_0		$a_n'' x_0$	$a_{n-1}'' x_0$	…	$a_2'' x_0$		
	a_n''	a_{n-1}''	a_{n-2}''	…	a_1'' $(=b_1)$		erster Schritt
				⋮			
x_0	$a_n^{(j)}$	$a_{n-1}^{(j)}$	…	$a_{j-1}^{(j)}$ $(=b_{j-1})$			$(j-1)$-ter Schritt (15)
				⋮			
x_0	$a_n^{(n)}$	$a_{n-1}^{(n)}$ $(=b_{n-1})$					$(n-1)$-ter Schritt
	$a_n^{(n+1)}$ $(=b_n)$						

Zum Beweis bezeichnen wir aus formalen Gründen P mit Q_n und haben dann nach Definition

$$\begin{aligned}
Q_n(x) &= a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 \\
Q_{n-1}(x) &= a_n' x^{n-1} + a_{n-1}' x^{n-2} + \cdots + a_2' x + a_1' \\
Q_{n-2}(x) &= a_n'' x^{n-2} + a_{n-1}'' x^{n-3} + \cdots + a_2'' \\
&\cdots\cdots\cdots\cdots \\
Q_1(x) &= a_n^{(n-1)} x + a_{n-1}^{(n-1)} \\
Q_0(x) &= a_n^{(n)} = a_n^{(n+1)}.
\end{aligned} \tag{16}$$

Es wird behauptet:

$$Q_n(x) = P(x) = (x - x_0)Q_{n-1}(x) + a_0'. \tag{17}$$

Zur Verifikation von (17) notiere man die Beziehungen zwischen den Größen des ersten Horner-Schrittes

$$\begin{array}{l|l}
a_n' = a_n & x^{n-1} \\
a_{n-1}' = a_{n-1} + a_n' x_0 & x^{n-2} \\
a_{n-2}' = a_{n-2} + a_{n-1}' x_0 & x^{n-3} \\
\cdots\cdots\cdots & \cdots \\
a_1' = a_1 + a_2' x_0 & 1
\end{array}$$

und multipliziere sie in der angegebenen Art der Reihe nach mit $x^{n-1}, x^{n-2}, \ldots, 1$. Addiert man alles, so resultiert auf der linken Seite $Q_{n-1}(x)$; die erste Spalte der rechten Seite liefert $\frac{Q_n(x) - a_0}{x}$, die zweite Spalte $\frac{x_0}{x}(Q_{n-1}(x) - a_1')$. Daraus folgt die Behauptung. Da zwei der Polynome $Q_\nu(x)$ mit aufeinanderfolgenden Indizes in derselben Weise miteinander verknüpft sind wie $Q_n(x)$ und $Q_{n-1}(x)$, ergibt sich auf Grund des Bewiesenen

$$\begin{array}{l|l}
Q_n(x) = (x - x_0)\, Q_{n-1}(x) + a_0' & 1 \\
Q_{n-1}(x) = (x - x_0)\, Q_{n-2}(x) + a_1'' & (x - x_0) \\
Q_{n-2}(x) = (x - x_0)\, Q_{n-3}(x) + a_2''' & (x - x_0)^2 \\
\cdots\cdots\cdots & \cdots \\
Q_1(x) = (x - x_0)\, Q_0(x) + a_{n-1}^{(n)} & (x - x_0)^{n-1} \\
Q_0(x) = a_n^{(n+1)} & (x - x_0)^n
\end{array}$$

Multipliziert man diese Gleichungen in der angegebenen Weise mit Potenzen von $(x - x_0)$ und addiert, so folgt unter Beachtung der wechselseitigen Kompensation der Glieder $Q_{n-i}(x)\,(x - x_0)^i$, $i = 1(1)n$,

$$\begin{aligned} P(x) &= Q_n(x) \\ &= a_n^{(n+1)}(x - x_0)^n + a_{n-1}^{(n)}(x - x_0)^{n-1} + \cdots + a_1''(x - x_0) + a_0', \end{aligned} \tag{18}$$

und das ist die gesuchte Umordnung des Polynoms nach Potenzen von $(x - x_0)$. Die in (18) auftretenden Koeffizienten

$$a_j^{(j+1)} = b_j = \frac{1}{j!} P^{(j)}(x_0)$$

erscheinen in der Diagonalen des Schemas (15). Dessen algorithmische Struktur wird mit dem PAP der Abb. 4.6 erfaßt. Dieser benutzt eine die Zeilen von (15) zählende Variable j; die Variable i kontrolliert den in Abb. 2.8 im PAP dargestellten Horner-Algorithmus und variiert in der j-ten Zeile rückläufig von $n-1$ bis j. Zur Aufnahme der Ergebnisgrößen b_j wird ein Feld $B[0:n]$ bereitgestellt. Würde man unmittelbar mit dem Feld der Koeffizienten a_ν rechnen, so stünden diese nach Abarbeitung des erweiterten Hornerschen Schemas nicht mehr zur Verfügung.

Vor Übertragung des PAP in ein ALGOL-Programm betrachten wir darin die beiden ineinander geschachtelten Zyklen, die durch Hinweisungspfeile herausgehoben wurden. Vergleicht man mit dem in Abb. 3.2 dargestellten Schema einer

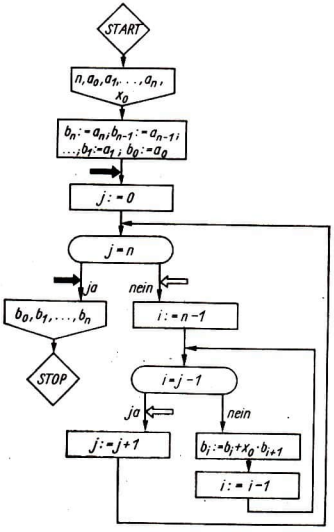


Abb. 4.6

Laufanweisung, so ergibt sich, daß die von j gesteuerte Anweisung selbst wieder eine Laufanweisung bezüglich i ist. A sei der Name des Feldes für die Koeffizienten, das wir vor Ausführung der Rechnung auf ein Feld B umstapeln. Beide Felder werden dynamisch vereinbart.

```
begin
  integer n; Lies(n);
  begin
    integer i, j; real x0; array A, B[0 : n]; Lies(A, x0);
    for i := 0 step 1 until n do B[i] := A[i];
    for j := 0 step 1 until n − 1 do
    for i := n − 1 step −1 until j do
```

```
    B[i] := B[i] + x0 × B[i + 1]; Drucke(B)
  end
end
```

Wir wollen noch das erweiterte Hornersche Schema als Prozedur – etwa mit der Bezeichnung *HORNER1* – programmieren und dabei die Felder A, B der Koeffizienten a_ν bzw. b_ν sowie den Polynomgrad n und die die Entwicklungsgröße bestimmende Variable x_0 als formale Parameter verwenden:

```
procedure HORNER1(A, B, n, x0);
value n, x0; integer n; real x0; array A, B;
begin
  integer i, j;
  for i := 0 step 1 until n do B[i] := A[i];
  for j := 0 step 1 until n − 1 do
  for i := n − 1 step − 1 until j do
    B[i] := B[i] + x0 × B[i + 1]
end
```

Zur Nullstellenbestimmung von Polynomen läßt sich das Newtonsche Verfahren mit dem erweiterten Hornerschen Schema kombinieren. Wir nehmen dabei an, daß die Konvergenz gesichert ist, und konzentrieren uns auf den algorithmischen Aspekt der Aufgabe. Startwert der bezüglich $f(x) = x - \frac{P(x)}{P'(x)}$ zu bildenden Iterationsfolge sei x_0; mit x_0 und den Koeffizienten des vorgelegten Polynoms

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

wird das erweiterte Hornersche Schema durchgerechnet. Dann ist

$$x_1 := f(x_0) = x_0 - \frac{a_0'}{a_1''} = x_0 + u_0$$

mit

$$u_0 := -\frac{a_0'}{a_1''}.$$

Wir können jetzt $P(x)$ nach Potenzen von $u := x - x_0$ umordnen und bezeichnen das Polynom P in seiner Abhängigkeit von u mit Q; es ist also $P(x) = Q(x - x_0) = Q(u)$ und $P(x_1) = Q(u_0)$, was als eine Verschiebung des Koordinatenursprungs in den Punkt x_0 der Abszissenachse gedeutet werden kann. Danach läuft es auf dasselbe hinaus, ob man nach dem Newtonschen Verfahren die Näherung x_0 bezüglich $P(x)$ oder u_0 bezüglich $Q(u)$ verbessert. Wählt man die zweite Möglichkeit und bildet mit Hilfe der Umordnung von

$$Q(u) = b_n u^n + b_{n-1} u^{n-1} + \cdots + b_1 u + b_0$$

nach der Entwicklungsgröße $v := u - u_0$

$$u_1 := u_0 - \frac{b_0'}{b_1''},$$

so erhält man mit $v_0 := -\frac{b_0'}{b_1''}$

$$u_1 = u_0 + v_0$$

und entsprechend in x die korrigierte Näherung

$$x_2 := x_0 + u_1 = x_0 + u_0 + v_0.$$

So fortfahrend ergibt sich die aus x_0 durch f erzeugte Iterationsfolge $x_0, x_1, x_2, \ldots$.

Aufgabe 2. Man fasse das kombinierte Newton-Hornersche Verfahren dieser Version in einer ALGOL-Prozedur zusammen, in der als formale Parameter das Koeffizientenfeld A des Polynoms, dessen Grad n, ferner x_0, y als Start- bzw. Endwert der Iterationsfolge sowie q und *eps* als Kontraktionsfaktor bzw. Genauigkeitsschranke auftreten. Lokal soll die Prozedur *HORNER*1 benutzt werden.

Im Unterschied zu Aufgabe 2 bildet die folgende Prozedur *NEWHO* die sukzessiven Approximationen gemäß

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

und benötigt demzufolge von *HORNER*1 nur die ersten beiden Schritte; diese sind in der Prozedur *HORNER*2 zusammengefaßt. Außerdem wird darauf verzichtet, eine Abbruchbedingung mit Hilfe eines Kontraktionsfaktors zu formulieren, und statt dessen durch einen Parameter k die Anzahl der zu bildenden Iterationen festgelegt; im übrigen haben die formalen Parameter die gleiche Bedeutung wie in Aufgabe 2:

```
procedure NEWHO(A, x0, y, k, n); value x0;
integer k, n; real x0, y; array A;
  begin
    integer i; real b0, b1, q, x;
    procedure HORNER2(A, n, b0, b1, x0); value x0;
    integer n; real b0, b1, x0; array A;
      begin
        integer i; array B[0 : n];
        for i := 0 step 1 until n do B[i] := A[i];
        for i := n − 1 step −1 until 0 do B[i] := B[i] + x0 × B[i + 1];
        for i := n − 1 step −1 until 1 do B[i] := B[i] + x0 × B[i + 1];
        b0 := B[0]; b1 := B[1];
      end;
    i := 0; x := x0;
```

```
L: HORNER2(A, n, b0, b1, x);
   i := i+1; y := x−b0/b1;
   if i<k then begin x := y; goto L end
end
```

Ein R 300-Programm zur näherungsweisen Bestimmung einer Polynomnullstelle, das *NEWHO* benutzt, könnte lauten:

```
'begin'
  'integer' k, n; read(k, n);
    'begin'
      'real' xØ, y; 'array' a[Ø:n];
      'procedure' newho(a, xØ, y, k, n);
        ⋮ ;
      read(xØ,a); newho(a, xØ, y, k, n); print(y);
    'end'
'end'
```

Beispiel. Es sollen sämtliche reellen Wurzeln der oben betrachteten kubischen Gleichung

$$g(u)=u^3+\frac{1}{4}u^2-\frac{1}{4}=0$$

nach der Methode von NEWTON bestimmt werden.

Der Verlauf der Graphen von

$$g_1(u)=u^3 \quad \text{und} \quad g_2(u)=\frac{1}{4}(1-u^2)$$

(vgl. Abb. 4.7) läßt erkennen, daß außer der früher schon ermittelten Wurzel

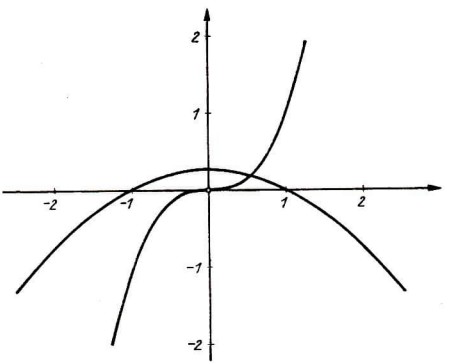


Abb. 4.7

keine weitere existiert. Auf der dabei betrachteten Umgebung $U_{1/4}\left(\frac{1}{2}\right)$ ist die Funktion f in (13) zwar erklärt, erfüllt jedoch dort nicht die Voraussetzungen des Satzes 2. Dennoch erzeugen wir mit f eine Iterationsfolge, als deren Startelement aber der rechte Endpunkt des Umgebungsintervalls $x_0 = 0{,}75$ gewählt

j	x_j
0	.75000000 # 00
1	.59848485 # 00
2	.55924042 # 00
3	.55670335 # 00
4	.55669309 # 00
5	.55669310 # 00
6	.55669310 # 00

Tabelle 4.3 Iterative Lösung der Gleichung $u^2(1+4u)=1$ nach der Methode von NEWTON-HORNER

wird (vgl. Tabelle 4.3). Man erkennt, daß die Iterationsfolge konvergiert, und zwar erheblich schneller als die der Tabelle 4.1. Wir wollen diesen Sachverhalt analysieren.

Die bisher erörterten Gleichungsprobleme wurden als Fixpunktaufgaben für gewisse Funktionen f interpretiert und einheitlich unter dem Gesichtspunkt des Banachschen Satzes bzw. des Satzes 1 betrachtet. Diese enthalten als konstruktives Element das Verfahren der sukzessiven Approximationen, dessen Konvergenz auf Grund der Kontraktivität von f nachgewiesen wurde. Die Iterationsfolge (6) kann jedoch auch aus anderen Gründen, z. B. wegen gewisser Monotonieeigenschaften von f konvergieren. Im Anschluß an das zuletzt betrachtete Beispiel erörtern wir einen speziellen Sachverhalt dieser Art für die gemäß (13) gebildete Funktion des Newtonschen Verfahrens.

Satz 3. *Die Funktion g in* (13) *besitze auf dem Intervall* $[a, b]$ *Ableitungen erster und zweiter Ordnung; ferner sei*

$$g(a) \cdot g(b) < 0 \tag{19}$$

und

$$g', g'' \neq 0 \quad \text{auf} \quad [a, b]. \tag{20}$$

Dann hat die Gleichung $g(x) = 0$ auf dem abgeschlossenen Intervall genau eine Lösung ξ. Ist $x_0 \in [a, b]$ so gewählt, daß

$$g(x_0) g''(x_0) > 0 \tag{21}$$

ist, so konvergiert die mit der in (13) *definierten Funktion f aus x_0 erzeugte Iterationsfolge (x_j) monoton gegen ξ:*

$$\lim_{j \to \infty} x_j = \xi. \tag{22}$$

Beweis. Zur Vereinfachung des Beweises nehme man zunächst noch an, daß g'' auf $[a, b]$ stetig ist. Dann lassen g' und g'' keinen Zwischenwert aus und haben daher wegen (20) einheitliches Vorzeichen auf $[a, b]$. Die Funktion g ist folglich streng monoton und nimmt wegen (19) Werte unterschiedlichen Vorzeichens in den Intervallenden an. Nach dem Zwischenwertsatz hat g daher genau eine Nullstelle ξ in $]a, b[$. Nun wird x_0 gemäß (21) gewählt und damit die Iterationsfolge (x_j) nach dem Newtonschen Verfahren erzeugt. Wir behaupten, daß diese monoton gegen ξ konvergiert. Für den Beweis sei etwa angenommen, daß $g(a) < 0$, $g(b) > 0$ sowie $g'(x) > 0$ und $g''(x) > 0$ auf $[a, b]$ ist. Das entspricht den Gegebenheiten in dem zuletzt betrachteten Beispiel bezüglich des Intervalls $[\frac{1}{4}, \frac{3}{4}]$. Durch Induktion kann man sich zunächst davon überzeugen, daß $\xi < x_j$ $(j = 0, 1, 2, \ldots)$ ist. Für x_0 folgt das aus (21) mit Beachtung von $g''(x_0) > 0$: Man erhält $g(x_0) > 0$, und wegen des monotonen Wachstums von g ergibt sich daraus $\xi < x_0$; es wird nun $\xi < x_j$ als gültig angenommen und gemäß

$$\xi = x_j + (\xi - x_j)$$

g an der Stelle x_j nach der Taylorschen Formel entwickelt:

$$0 = g(\xi) = g(x_j) + g'(x_j)\,(\xi - x_j) + \frac{1}{2} g''(\eta)\,(\xi - x_j)^2, \tag{23}$$

wobei $\xi < \eta < x_j$ ist. Mit Beachtung von $g''(\eta) > 0$ folgt daraus

$$x_{j+1} := x_j - \frac{g(x_j)}{g'(x_j)} > \xi,$$

was zu zeigen war. Wegen $x_j > \xi$ ist auf Grund der Vorzeichenverhältnisse bei g und g'

$$x_{j+1} < x_j$$

und somit (x_j) eine durch ξ nach unten beschränkte monoton fallende Folge, die einen Limes $\hat{\xi}$ in $[a, b]$ besitzt. Durch den Grenzübergang $j \to \infty$ folgt nun aus $f(x_j) = x_{j+1}$ wegen der hier gegebenen Stetigkeit von f

$$f(\hat{\xi}) = \hat{\xi},$$

also mit Beachtung der schon gezeigten Einzigkeit der Lösung

$$\hat{\xi} = \xi.$$

Damit ist der Satz 3 bis auf die zusätzlich eingeführte Annahme der Stetigkeit von g'' bewiesen. Diese wurde nur in Anspruch genommen, um bequem zeigen zu können, daß g'' keinen Zwischenwert ausläßt. Das aber folgt allein schon daraus, daß diese Funktion Ableitung einer anderen ist (auf $[a, b]$ eine Stammfunktion besitzt). Zur Begründung wird auf das folgende Lemma verwiesen.

Eine Funktion h sei auf $[a,b]$ differenzierbar. Dann nimmt h' alle Werte zwischen $h'(a)$ und $h'(b)$ an.

Beweis. Die Differenzenquotienten

$$\varphi(x)=\frac{h(x)-h(a)}{x-a} \quad \text{und} \quad \psi(x)=\frac{h(b)-h(x)}{b-x}$$

definieren auf $[a, b]$ stetige Funktionen, wenn man diese in den Nullstellen der Nenner durch $h'(a)$ bzw. $h'(b)$ erklärt. Die Funktionen φ, ψ nehmen auf Grund des Zwischenwertsatzes alle Werte zwischen

$$\frac{h(b)-h(a)}{b-a} \quad \text{und} \quad h'(a) \quad \text{bzw.} \quad \frac{h(b)-h(a)}{b-a} \quad \text{und} \quad h'(b)$$

an. Da jeder der betrachteten Differenzenquotienten nach dem Mittelwertsatz einem Wert von h' gleich ist, nimmt $h'(x)$ jeden Wert zwischen $h'(a)$ und $h'(b)$ an.

Man verifiziert leicht, daß für das Beispiel der iterativen Lösung einer kubischen Gleichung nach dem Newtonschen Verfahren die Voraussetzungen des Satzes 3 erfüllt sind. Auf die in Tabelle 4.3 sichtbar gewordene schnelle Konvergenz der Folge (x_j) werden wir im nächsten Abschnitt zurückkommen.

Unter gewissen weiteren Voraussetzungen (vgl. dazu etwa [17], S. 129) kann man auf die Einschränkung (21) bei der Wahl des Startelements verzichten. Wir betrachten dazu die Quadratwurzelberechnung nach der Methode von NEWTON als Beispiel. Die Gleichung

$$g(x)=x^2-A=0, \quad A>0, \tag{24}$$

geht dabei in

$$f(x)=\frac{1}{2}\left(x+\frac{A}{x}\right)=x \tag{25}$$

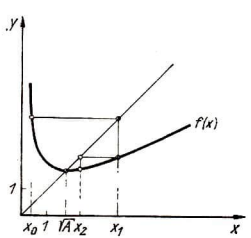


Abb. 4.8

(vgl. Abb. 4.8) über, und für $x \neq 0$ sind (24) und (25) äquivalent.

Aus $f'(x)=\frac{1}{2}\left(1-\frac{A}{x^2}\right)$ folgt

$$f'(x)>0 \quad \text{für} \quad x>\sqrt{A},$$

$$f'(x)<0 \quad \text{für} \quad x<\sqrt{A},$$

$$f'(\sqrt{A})=0,$$

d. h., f hat bei $\sqrt{A}$ sein absolutes Minimum, und es gilt somit

$$\bigwedge_x (0 < x < \infty \Rightarrow f(x) \geqq \sqrt{A}). \tag{26}$$

Wenn man also ein beliebiges $x_0 > 0$ wählt und damit $x_1 := f(x_0)$ bildet, ist dieser Wert die gesuchte Quadratwurzel von A oder größer als $\sqrt{A}$. Dann ist für x_1 (21) erfüllt und – da sich die Punkte x_0, x_1 gewiß in ein Intervall $[a, b]$ einschließen lassen, für welches die Funktion g in (24) den Voraussetzungen (19) und (20) genügt – die Konvergenz der Iterationsfolge auf Grund des Satzes 3 gesichert. Diese erfolgt (zumindest von x_1 ab) theoretisch monoton fallend gegen $\sqrt{A}$ (vgl. Abb. 4.8); praktisch wird jedoch wegen des beschränkten Zahlbereichs des Automaten einmal $x_{j+1} = x_j$ oder – durch Rundungsfehler bedingt – sogar $x_{j+1} \cong x_j$ eintreten. In der folgenden Funktionsprozedur *WURZ* wird dieser Effekt zur Bildung der Abbruchbedingung bei der Berechnung einer Quadratwurzel ausgenutzt (Startelement x_0 ist der Radikand):

```
real procedure WURZ(t); value t; real t;
  begin real x, y;
    x := t;
L:  y := 0.5 × (x + t/x);
    if y < x then begin x := y; goto L end;
    WURZ := x
  end
```

Aufgabe 3. Man schreibe mit Benutzung der Prozedur *WURZ* ein R 300-ALGOL-Programm zur Berechnung einer Quadratwurzeltafel mit äquidistanten Argumenten.

4.1.3. Fehlerbetrachtungen

Wir wollen genauer die Konvergenz einer Iterationsfolge (x_j) des Newtonschen Verfahrens erörtern, und zwar weiterhin unter der Annahme, daß g'' auf einem diese einschließenden Intervall $[a, b]$ existiert und beschränkt ist. Dabei ist es unerheblich, ob deren Bildbarkeit und Konvergenz auf Grund des Satzes 2 oder 3 gesichert ist. Es sei $\xi^* \in [a, b]$ Näherung für die Lösung ξ der Gleichung $g(x) = 0$. *Dann gilt bezüglich des absoluten Fehlers*

$$|g'(x)| \geqq m_1 > 0 \text{ auf } [a, b] \Rightarrow |\xi - \xi^*| \leqq \frac{|g(\xi^*)|}{m_1}. \tag{27}$$

Beweis. Auf Grund des Mittelwertsatzes ist

$$g(\xi) - g(\xi^*) = (\xi - \xi^*)\, g'(\eta), \quad \xi \ldots \eta \ldots \xi^*,$$

also wegen $g(\xi) = 0$

$$|g(\xi^*)| \geqq m_1 |\xi - \xi^*|.$$

Wir wenden (27) auf die sukzessiven Approximationen $\xi^* = x_j$ des Newtonschen Verfahrens an. Man beachte, daß hier $|g'(x)|$ als stetige und von Null verschiedene

Funktion auf $[a, b]$ eine positive untere Schranke besitzt. $|g(x_j)|$ wird mit Hilfe des Taylorschen Satzes abgeschätzt. Danach ist

$$g(x_j) = g(x_{j-1} + (x_j - x_{j-1}))$$
$$= g(x_{j-1}) + g'(x_{j-1})\,(x_j - x_{j-1}) + \frac{1}{2}\,g''(\eta_{j-1})\,(x_j - x_{j-1})^2,$$
$$x_{j-1} \ldots \eta_{j-1} \ldots x_j.$$

Wegen (11) annullieren sich die ersten beiden Terme der rechten Seite, so daß

$$|g(x_j)| \leqq \frac{1}{2}\,M_2 (x_j - x_{j-1})^2$$

ist, wenn auf $[a, b]$

$$|g''(x)| \leqq M_2$$

gilt. (27) liefert damit die Abschätzung

$$|\xi - x_j| \leqq \frac{M_2}{2m_1}\,(x_j - x_{j-1})^2. \tag{28}$$

Da die Folge (x_j) als konvergent angenommen wurde, strebt $x_j - x_{j-1}$ gegen Null, und es gibt eine Nummer p so, daß für $j \geqq p$

$$|\xi - x_j| \leqq |x_j - x_{j-1}|$$

ist, d. h., der Betrag der Differenz zweier aufeinanderfolgender Näherungen majorisiert für hinreichend großes j den Betrag des absoluten Fehlers der j-ten Näherung.

Der Vergleich des absoluten Fehlers zweier aufeinanderfolgender Näherungen des Newtonschen Verfahrens zeigt, daß dieses im Konvergenzfall rasch konvergiert. In der Tat ist nach (23)

$$\xi = x_j - \frac{g(x_j)}{g'(x_j)} - \frac{1}{2}\,\frac{g''(\eta)}{g'(x_j)}\,(\xi - x_j)^2$$

und mit Beachtung von (11)

$$\xi - x_{j+1} = -\frac{1}{2}\,\frac{g''(\eta)}{g'(x_j)}\,(\xi - x_j)^2, \tag{29}$$

also

$$|\xi - x_{j+1}| \leqq \frac{M_2}{2m_1}\,|\xi - x_j|^2. \tag{30}$$

Ist $\frac{M_2}{2m_1} \leqq 1$, so folgt nach (30) aus $|\xi - x_j| \leqq 10^{-k}$ $(k \geqq 1)$

$$|\xi - x_{j+1}| \leqq 10^{-2k},$$

d. h., die Zahl der gültigen Ziffern nach dem Komma verdoppelt sich ungefähr bei jedem Verfahrensschritt (vgl. Tabelle 4.3).

Die zuletzt gewonnenen Einsichten nehmen wir zum Anlaß, ein Maß für die

Konvergenzgeschwindigkeit einer gegen ξ konvergierenden Folge (x_j) einzuführen. Dabei interessieren gewisse mit dem Betrag des absoluten Fehlers

$$d_j := |x_j - \xi| \tag{31}$$

gebildete Verhältnisse für zwei aufeinanderfolgende Näherungen. Wir definieren: (x_j) heißt *linear konvergent*, wenn

$$\bigvee_q \left(0 < q < 1 \wedge \lim_{j \to \infty} \frac{d_{j+1}}{d_j} = q\right), \tag{32}$$

und *überlinear konvergent* vom *Konvergenzgrad* γ $(\gamma > 1)$, wenn

$$\bigvee_Q \left(0 < Q < \infty \wedge \lim_{j \to \infty} \frac{d_{j+1}}{d_j^\gamma} = Q\right). \tag{33}$$

Offensichtlich ist der Konvergenzgrad ($\gamma = 1$ bei linearer Konverenz) eindeutig bestimmt.

Wir betrachten als Beispiel eine nach dem Newtonschen Verfahren erzeugte Iterationsfolge unter der Voraussetzung des Satzes 2 oder 3; g'' sei auf dem betreffenden Intervall stetig und beschränkt. Auf Grund von (29) ist

$$\lim_{j \to \infty} \frac{d_{j+1}}{d_{j+1}^2} = \frac{1}{2} \frac{g''(\xi)}{g'(\xi)},$$

d. h., wir erhalten $\gamma = 2$, sofern $g''(\xi) \neq 0$ ist. Wie am Beispiel des Newtonschen Verfahrens gezeigt wurde, drückt sich der numerische Effekt des Konvergenzgrades allgemein in einer Gesetzmäßigkeit bezüglich der sukzessive wachsenden Zahl gültiger Ziffern bei den x_j aus.

Oft ist man genötigt, an Stelle der Fehlerbeträge d_j geeignete Schranken e_j gemäß

$$d_j = |x_j - \xi| \leqq e_j \tag{34}$$

in den Definitionen (32) und (33) zu benutzen. Unter den Voraussetzungen des Satzes 1 gewinnt man solche z. B. aus der a-priori-Abschätzung (8) in der Form

$$e_j := \frac{q^j}{1-q} |x_0 - x_1|. \tag{35}$$

Geht man damit an Stelle der d_j in (32) ein, so folgt

$$\lim_{j \to \infty} \frac{e_{j+1}}{e_j} = q.$$

Allgemein wird man also von einer nach dem gewöhnlichen Iterationsverfahren auf Grund der Voraussetzungen des Satzes 1 konstruierten Folge (6) nur lineare Konvergenz erwarten können. Wir bemerken dazu noch, daß für die Funktion $f(x) = qx$, $0 < q < 1$, die Größen (35) mit den Werten d_j übereinstimmen: In diesem Fall ist $\xi = 0$ und $x_j = q^j x_0$, also

$$d_j = q^j |x_0| = \frac{q^j}{1-q} |1-q| \cdot |x_0| = \frac{q^j}{1-q} |x_0 - x_1| = e_j.$$

Von erheblichem Interesse sind Methoden, die es gestatten, die Konvergenz einer gegen eine Lösung ξ von (2) konvergierenden Folge (x_j) von Näherungen zu beschleunigen, wobei außer Betracht bleiben kann, wie diese erzeugt wurden. Beim *Aitkenschen δ^2-Verfahren* betrachtet man neben x_j den Wert $\hat{x}_{j+1} := f(x_j)$ und die Abweichungen

$$f(x_j) - x_j \quad \text{bzw.} \quad f(\hat{x}_{j+1}) - \hat{x}_{j+1}.$$

Die Verbesserung von x_j wird durch die Schnittpunktabszisse z_{j+1} der durch die Punkte

$$(x_j, \hat{x}_{j+1} - x_j) = (x_j, f(x_j) - x_j)$$

und

$$(\hat{x}_{j+1}, f(\hat{x}_{j+1}) - \hat{x}_{j+1})$$

laufenden Geraden mit der x-Achse bestimmt. Es ist also

$$\frac{z_{j+1} - x_j}{x_j - \hat{x}_{j+1}} = \frac{\hat{x}_{j+1} - x_j}{f(\hat{x}_{j+1}) - 2\hat{x}_{j+1} + x_j}$$

oder

$$z_{j+1} = x_j - \frac{(\hat{x}_{j+1} - x_j)^2}{f(\hat{x}_{j+1}) - 2\hat{x}_{j+1} + x_j}. \tag{36}$$

Falls (x_j) die mit einem Startelement x_0 gebildete Iterationsfolge (6) ist, nimmt (36) die Form

$$z_{j+1} = x_j - \frac{(x_{j+1} - x_j)^2}{x_{j+2} - 2x_{j+1} + x_j}$$

an. Dafür kann man auch schreiben:

$$z_{j+1} = x_{j+2} - \frac{(x_{j+2} - x_{j+1})^2}{x_{j+2} - 2x_{j+1} + x_j}. \tag{37}$$

Man erkennt, daß die Näherung z_{j+1} aus drei aufeinanderfolgenden Iterationen zu ermitteln ist.

Mit Benutzung der Funktion f aus (2) ist

$$\begin{aligned} z_{j+1} &= x_j - \frac{(f(x_j) - x_j)^2}{f(f(x_j)) - 2f(x_j) + x_j} \\ &= \frac{x_j f(f(x_j)) - f(x_j)^2}{f(f(x_j)) - 2f(x_j) + x_j}. \end{aligned} \tag{38}$$

Danach ergibt sich z_{j+1} als Wert der Funktion

$$y = \frac{xf(f(x)) - f(x)^2}{f(f(x)) - 2f(x) + x} \tag{39}$$

für das Argument x_j, d. h., die fortlaufende Anwendung des Aitken-Algorithmus (37) auf die Folge (x_j) erzeugt die Werte der mit (39) zum Startelement x_0 gebildeten Iterationsfolge (6). Auf dieser Grundlage wird die Konvergenz des δ^2-Verfahrens eingehend in

[33], 3.2.4., untersucht. Seine Bezeichnung nimmt auf den Nenner in (37) Bezug, der ein Ausdruck der Differenzenrechnung ist (vgl. 4.2.).

Die bisher zu Iterationsverfahren durchgeführten Fehlerbetrachtungen waren insofern unvollständig, als sie von der Annahme exakt berechenbarer x_j ausgingen. Tatsächlich werden dafür im allgemeinen nur Näherungen x_j^* ermittelt, weil mit Rundungsfehlern behaftete Daten in die Rechnung eingehen oder f schon zu Beginn derselben durch eine approximierende Funktion ersetzt wird.

Wir betrachten den Sachverhalt weiter unter der Annahme, daß f eine abgeschlossene Menge $F \subseteq \mathsf{R}$ in sich abbildet und auf dieser kontrahierend mit dem Kontraktionsfaktor q ist. Nach dem Banachschen Satz konvergiert jede Iterationsfolge gegen den wohlbestimmten Fixpunkt $\xi \in F$. Für die auf Grund der zu Anfang geschilderten Fehlereinflüsse tatsächlich konstruierten Näherungen x_j^* setzen wir

$$x_{j+1}^* = f(x_j^*) + h_j \quad (j = 0, 1, 2, \ldots) \tag{40}$$

und fordern, daß diese zu F gehören; von den Korrekturtermen h_j wird vorausgesetzt, daß ihre Beträge durch eine positive Zahl majorisierbar sind:

$$|h_j| \leqq \delta \quad (j = 0, 1, 2, \ldots).$$

Man findet dann für den absoluten Fehler

$$\begin{aligned} |x_{j+1}^* - \xi| &= |f(x_j^*) - f(\xi) + h_j| \leqq |f(x_j^*) - f(\xi)| + |h_j| \\ &\leqq q|x_j^* - \xi| + \delta \end{aligned}$$

und durch wiederholte Anwendung dieser Schlußweise

$$\begin{aligned} |x_{j+1}^* - \xi| &\leqq q|x_j^* - \xi| + \delta \leqq q^2 |x_{j-1}^* - \xi| + q\delta + \delta \leqq \cdots \\ &\leqq q^{j+1} |x_0^* - \xi| + (q^j + q^{j-1} + \cdots + 1)\delta \\ &\leqq q^{j+1} |x_0^* - \xi| + \frac{\delta}{1-q}. \end{aligned} \tag{41}$$

Aus (41) kann nicht mehr auf die Konvergenz der Folge (x_j^*) gegen ξ geschlossen werden, wohl aber gilt

$$\limsup_{j \to \infty} |x_j^* - \xi| \leqq \frac{\delta}{1-q}, \tag{42}$$

und dies besagt, daß man mit der Folge (x_j^*) den Wert ξ angenähert bestimmen kann, sofern keine höhere Genauigkeit als $\frac{\delta}{1-q}$ gefordert ist. Das muß man bei der Formulierung von Abbruchbedingungen beachten. Durch weitere Voraussetzungen kann man sichern, daß die Elemente (40) stets im Definitionsbereich von f liegen (vgl. [33], S. 96).

4.1.4. Mehrstufige Verfahren

Die algorithmische Struktur der schrittweisen Konstruktion von Näherungen für eine Gleichungslösung kann komplizierter als beim gewöhnlichen Iterationsverfahren sein. In letzter Verallgemeinerung ergibt sich das Element x_{j+1} einer Folge von Näherungen mit Hilfe eines von j abhängenden Algorithmus A_j, der auf alle vorangegangenen Approximationen anzuwenden ist (vgl. [28]); wir bringen diesen Sachverhalt durch

$$x_{j+1} := A_j\{x_0, x_1, \ldots, x_j\} \tag{43}$$

zum Ausdruck. Gelegentlich wird das Verfahren (43) als *stationär* bezeichnet, wenn A_j für alle j ein fester Algorithmus A ist. Ein stationäres Verfahren heißt *m-stufig*, wenn in (43) nur die letzten m vorangehenden Näherungen auftreten:

$$x_{j+1} := A\{x_{j-m+1}, x_{j-m+2}, \ldots, x_j\}. \tag{44}$$

Zwei sehr einfache Beispiele für Verfahren des Typs (43) sind die Methode der *Bisektion* und der *regula falsi*, die wir hier für die Gleichung (1) unter der Annahme, daß g stetig ist, erörtern.

Die erste Methode beruht auf der Tatsache, daß eine auf dem abgeschlossenen Intervall $[x_0, x_1]$ stetige Funktion g, die in x_0, x_1 Werte unterschiedlichen Vorzeichens annimmt, auf dem Intervall mindestens eine Nullstelle ξ besitzt. Zur approximativen Bestimmung einer solchen betrachtet man den Mittelpunkt $x_2 := \frac{x_1 + x_2}{2}$. Falls nun $g(x_2) \neq 0$ ist, wird dasjenige der beiden Intervalle $[x_0, x_2]$, $[x_2, x_1]$ gewählt, für dessen Endpunkte a, b $(a < b)$

$$g(a)\, g(b) < 0 \tag{45}$$

gilt, und wie beim ersten Schritt fortgefahren. Die Endpunkte a_n, b_n der sukzessive ausgewählten Intervalle bilden je eine monoton nicht abnehmende bzw. nicht wachsende, nach oben bzw. unten beschränkte Zahlenfolge, die beide wegen $\lim_{n\to\infty}(b_n - a_n) = 0$ gegen einen gemeinsamen Grenzwert ξ konvergieren. Wegen (45) ist aus Stetigkeitsgründen $g(\xi)^2 \leqq 0$, also $g(\xi) = 0$. Bedeutet k den größten Index unterhalb j, für den

$$g(x_j)\, g(x_k) < 0 \tag{46}$$

gilt, so konvergiert auch die nach dem Algorithmus

$$x_{j+1} := \frac{1}{2}(x_j + x_k) \quad (j = 1, 2, \ldots) \tag{47}$$

konstruierte Folge der Intervallmitten gegen die Lösung ξ der Gleichung (1). Für den absoluten Fehler gilt die Abschätzung

$$e_j := \frac{|x_1 - x_0|}{2^{j-1}} \geqq |\xi - x_j| \quad (j > 1). \tag{48}$$

Der PAP der Abb. 4.9 veranschaulicht den Algorithmus des Halbierungsverfahrens. Die Abbruchbedingung wird mit der Länge des jedem Schritt zugeordneten Einschließungsintervalls bzw. dem Erreichen einer Nullstelle von g gebildet. Bezüglich

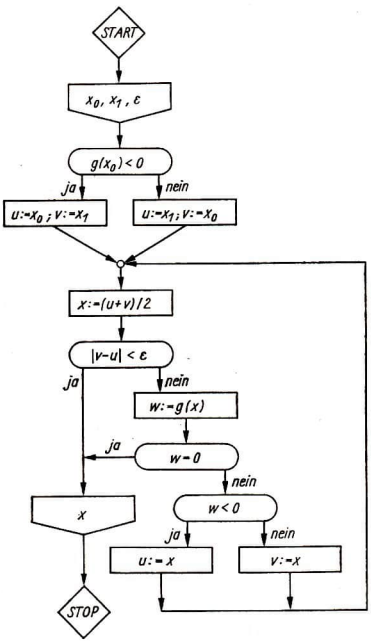


Abb. 4.9

der Abschätzungen (48) erweist sich die Folge (x_j) als linear konvergent, denn es ist

$$\lim_{j\to\infty} \frac{e_{j+1}}{e_j} = \frac{1}{2}.$$

Wir fassen das Verfahren noch in einer ALGOL-Prozedur *BIS* zusammen, die in ihrem Aufbau vollständig dem PAP entspricht:

```
procedure BIS(g, x0, x1, x, eps); value x0, x1, eps;
real x0, x1, x, eps; real procedure g;
  begin real u, v, w;
    if g(x0) < 0 then begin u := x0; v := x1 end
                 else begin u := x1; v := x0 end;
L:  x := (u + v)/2;
    if abs(v - u) < eps then goto M;
    w := g(x);
    if w = 0 then goto M;
    if w < 0 then u := x else v := x; goto L;
M:end
```

Die Bisektionsmethode ist für Computerprogramme zumindest als *Anlaufverfahren* für Verfahren höherer Konvergenzgeschwindigkeit (z. B. NEWTON-RAPHSON) gut geeignet wegen des einfachen Algorithmus und der Gewißheit, ohne zusätzliche Bedingungen eine Näherungslösung der Gleichung mit gewünschter Genauigkeit zu liefern.

Die regula falsi führen wir durch *Diskretisierung* des Newtonschen Verfahrens ein, d. h., wir ersetzen in der Iterationsvorschrift

$$x_{j+1} := x_j - \frac{g(x_j)}{g'(x_j)} \,,$$

die Ableitung $g'(x_j)$ durch den mit x_{j-1} und x_j gebildeten Differenzenquotienten und erhalten so

$$x_{j+1} := x_j - g(x_j)\frac{x_j - x_{j-1}}{g(x_j) - g(x_{j-1})} \quad (j = 1, 2, \ldots); \tag{49}$$

danach ist x_{j+1} der Schnittpunkt der Sekante durch die Punkte $(x_{j-1}, g(x_{j-1}))$ und $(x_j, g(x_j))$ mit der Abszissenachse. Man erkennt die Iterationsvorschrift als ein zweistufiges Verfahren im Sinne von (44). Wir befassen uns nicht mit Bedingungen für die Konvergenz der Folge (49); unter gewissen Voraussetzungen kann gezeigt werden, daß der Konvergenzgrad $\gamma = \frac{1+\sqrt{5}}{2}$ ist (vgl. [33]). Dieser merkwürdige Zahlwert resultiert aus einer Eigenschaft der Folge der Fibonaccischen Zahlen.

Als Primitivform der regula falsi bezeichnet man gelegentlich die Iterationsvorschrift

$$x_{j+1} := x_j - g(x_j)\frac{x_j - x_k}{g(x_j) - g(x_k)} \,, \tag{50}$$

wobei k den größten Index unterhalb j bezeichnet, für den $g(x_j)\, g(x_k) < 0$ ist; die Startelemente x_0, x_1 sind ebenfalls so zu wählen, daß $g(x_0)\, g(x_1) < 0$ gilt (vgl. Abb. 4.10). Das Verfahren (50) konvergiert unter den schwachen Voraussetzungen der Methode der Bisektion gegen eine Lösung der Gleichung $g(x) = 0$, allerdings wie dieses nur linear. Das ist ein Nachteil gegenüber dem Newtonschen Verfahren. Die Werte x_j, x_k bestimmen in jedem Schritt ein Einschließungsintervall für die

Lösung ξ, und diese bilden insgesamt eine ξ erfassende Intervallschachtelung. Abgesehen von den Konvergenzbedingungen besitzt die regula falsi sowohl in der Version (49) wie auch (50) gegenüber dem Newtonschen Verfahren den Vorteil

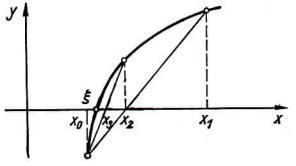


Abb. 4.10

eines wesentlich geringeren Rechenaufwands. Während bei diesem in jedem Schritt ein Funktions- und ein Ableitungswert zu bestimmen ist, erfordert jene nur die Berechnung von $g(x_j)$, während $g(x_k)$ $(k=j-1$ in (49)) gespeichert übernommen werden kann.

Wir fassen das Verfahren (50) in einer ALGOL-Prozedur *Refa* zusammen, die analog zur Prozedur *BIS* konzipiert ist:

```
procedure REFA(g, x0, x1, x, eps); value x0, x1, eps;
real x0, x1, x, eps; real procedure g;
  begin
    real u, v, wu, wv, w;
    if g(x0)<0 then begin u := x0, v := x1 end
               else begin u := x1; v := x0 end;
    wu := g(u); wv := g(v);
L:  x := v-(v-u)/(wv-wu)×wv;
    if abs(v-u)<eps then goto M;
    w := g(x);
    if w=0 then goto M;
    if w<0 then begin u := x; wu := w end
           else begin v := x; wv := w end;
    goto L;
M:end
```

4.2. Interpolation mit ganzrationalen Funktionen

4.2.1. Interpolationsaufgabe. Lagrangesches Polynom

Das Interpolationsproblem ist in MfL Bd. 4, 2.7., als eine spezielle Aufgabe der Approximationstheorie umrissen worden. In dem hier zu erörternden Fall, daß die zur Interpolation zugelassenen Funktionen Polynome sind, kann man diese in geometrischer Ausdrucksweise so formulieren:

Gegeben sind $n+1$ Punkte $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ mit paarweise verschiedenen Abszissen. Gesucht ist ein Polynom P, dessen Graph diese Punkte verbindet (Abb. 4.11). (1)

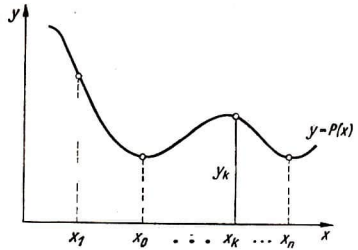


Abb. 4.11

Setzt man $P(x) = \sum_{j=0}^{m} c_j x^j$, so führt (1) auf das lineare Gleichungssystem

$$\sum_{j=0}^{m} c_j x_k^j = y_k \quad (k = 0, 1, \ldots, n). \tag{2}$$

Dieses besitzt für $m = n$ genau eine Lösung, da die Koeffizientendeterminante eine Vandermondesche Determinante und als solche von Null verschieden ist. Für $m > n$ gibt es offenbar unendlich viele Lösungen von (2) und damit der Interpolationsaufgabe (1). Im Fall $m < n$ ist das Problem im allgemeinen unlösbar. Auf Grund dieser Betrachtungen wurde in MfL Bd. 4, 2.7.2., der folgende Satz bewiesen:

Satz 1. *Zu jeder reellen (oder komplexen) Funktion f und zu jedem System von n verschiedenen Stützstellen x_k $(k = 0, 1, \ldots, n; x_k \in D(f))$ gibt es genau eine ganzrationale Funktion P von höchstens n-tem Grade derart, daß*

$$P(x_k) = f(x_k) \quad (k = 0, 1, \ldots, n) \tag{3}$$

ist.

Die Funktion P werden wir weiterhin als das durch die Punkte (x_k, y_k) $(k = 0, 1, \ldots, n)$ bestimmte Interpolationspolynom ansprechen. Die Abszissen x_k heißen *Stützstellen* oder *Interpolationsknoten*.

Häufig betrachtet man das gemäß (2) eindeutig bestimmte Interpolationspolynom in einer expliziten, von J. L. Lagrange angegebenen Form, der wir uns jetzt zuwenden. Mit $l_i (i = 0, 1, \ldots, n)$ seien Polynome n-ten Grades bezeichnet, die an allen Stützstellen x_j mit Ausnahme der i-ten verschwinden; bei x_i soll $l_i(x_i) = 1$ gelten. Wir fordern also

$$l_i(x_j) = \delta_{ij} \quad (i, j = 0, 1, \ldots, n). \tag{4}$$

Auf Grund von Satz 1 sind die l_i durch (4) eindeutig bestimmt; wie man leicht sieht, gilt explizit

$$l_i(x) = \frac{(x-x_1)\,(x-x_2)\cdots(x-x_{i-1})\,(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_1)\,(x_i-x_2)\cdots(x_i-x_{i-1})\,(x_i-x_{i+1})\cdots(x_i-x_n)}. \tag{5}$$

Wir behaupten:

$$L(x) = \sum_{i=1}^{n} y_i l_i(x) \tag{6}$$

ist Lösung des Problems (1).

Beweis. Als Linearkombination von Polynomen n-ten Grades ist (6) höchstens vom Grade n. Ferner gilt für $j = 0, 1, \ldots, n$

$$L(x_j) = \sum_{i=1}^{n} y_i l_i(x_j) = \sum_{i=1}^{n} y_i \delta_{ij} = y_j. \tag{7}$$

Der Ausdruck (6) ist das *Lagrangesche Interpolationspolynom*, das man häufiger im Zusammenhang mit theoretischen Untersuchungen als zur numerischen Auswertung der Lösung von (1) benutzt. Dabei ist beachtenswert, daß in den l_i nur die Abszissen der Interpolationspunkte vorkommen; die Ordinaten y_i erscheinen in (6) linear außerhalb der l_i. Für Berechnungen ist das Lagrangesche Polynom dann weniger geeignet, wenn man – etwa um den Hornerschen Algorithmus einzusetzen – nach Potenzen von x ordnet. In diesem Fall würde die Einbeziehung weiterer Interpolationspunkte eine Wiederholung aller Umformungen von Anfang an erfordern. Es ist aber darauf hinzuweisen, daß für äquidistante Stützstellen $x_j = x_0 + jh$ $(j = 0, 1, \ldots, n)$ eine Normierung der l_i möglich ist, die deren vom Stützstellensystem unabhängige Tabulierung gestattet (vgl. [35]).

Aufgabe 1. Die zwischen 1 und 2 gelegenen Nullstellen von

$$g(x) = x^3 - 7x + 7$$

sind näherungsweise durch quadratische Interpolation zu bestimmen, d. h., es ist ein Interpolationspolynom h zweiten Grades zu ermitteln, das im Intervall $1 \leqq x \leqq 2$ die Funktion g approximiert und dessen Nullstellen als Näherungen für die Wurzeln von $g(x) = 0$ genommen werden können. Man benutze $x_0 = 1$, $x_1 = \frac{3}{2}$ und $x_2 = 2$ als Stützstellen und kontrolliere die Güte der Näherungsnullstellen durch Berechnung der entsprechenden Funktionswerte $g(x)$.

Aufgabe 2. Bestimmung eines Extremwerts aus drei Messungen (z. B. Kulminationshöhe eines Gestirns). Von einer zwei physikalische Größen verbindenden Funktion mögen auf Grund eines Experiments drei zusammengehörige Wertepaare (x_1, y_1), (x_2, y_2), (x_3, y_3) mit $x_1 < x_2 < x_3$ vorliegen. Ferner sei bekannt, daß die Funktion im Intervall (x_1, x_3) genau ein absolutes Extremum besitzt. Man bestimme dieses näherungsweise unter Benutzung eines Interpolationspolynoms.

Anleitung. Man setze mit unbestimmten Koeffizienten

$$y(x) = a_2 x^2 + a_1 x + a_0$$

an und eliminiere die a_r aus den Gleichungen

$$\begin{aligned} a_2x^2+a_1x\ +a_0-y(x)&=0,\\ a_2x_1^2+a_1x_1+a_0-y_1&=0,\\ a_2x_2^2+a_1x_2+a_0-y_2&=0,\\ a_2x_3^2+a_1x_3+a_0-y_3&=0 \end{aligned}$$

mit Hilfe einer Determinantenbedingung.

Die Lösung dieser Aufgabe findet auch Anwendung in der nichtlinearen Optimierung bei sogenannten linearen Suchprozessen. Dabei handelt es sich darum, für eine im $\mathbf{R}_p$ definierte Funktion Extrema auf einer Halbgeraden zu bestimmen.

4.2.2. Steigungen (Dividierte Differenzen)

Für die numerische Bestimmung von Werten eines Interpolationspolynoms wird allgemein eine von NEWTON stammende Form desselben bevorzugt. Diese benutzt an Stelle der in (6) auftretenden Funktionswerte y_i gewisse mit diesen gebildete Differenzen. Im folgenden wird ein Abriß der Theorie dieser Größen dargestellt.

Es sei f eine Funktion einer reellen Veränderlichen, die zumindest an den weiterhin betrachteten Argumentstellen definiert ist. Soweit diese Stellen verschieden bezeichnet sind, sollen sie zunächst auch tatsächlich verschieden sein.

Als *Steigung* (*dividierte Differenz*) $f[x_0]$ nullter Ordnung von f an der Stelle x_0 definiert man

$$f[x_0] := f(x_0). \tag{8}$$

Die Steigung $f[x_0, x_1]$ erster Ordnung von f bei x_0, x_1 ist durch

$$f[x_0, x_1] := \frac{f[x_1]-f[x_0]}{x_1-x_0} = \frac{f(x_1)-f(x_0)}{x_1-x_0} \tag{9}$$

erklärt, und allgemein bestimmt man die Steigung $f[x_0, x_1, \ldots, x_k]$ k-ter Ordnung von $f(x)$ bei $x_0, x_1, \ldots, x_k$ induktiv durch

$$f[x_0, x_1, \ldots, x_k] := \frac{f[x_1, \ldots, x_k]-f[x_0, \ldots, x_{k-1}]}{x_k-x_0}. \tag{10}$$

Wenn aus dem Zusammenhang zweifelsfrei erkennbar ist, mit welcher Funktion Steigungen zu bilden sind, kann das Funktionszeichen vor dem Klammersymbol weggelassen werden.

Bereits auf Grund der induktiven Definition ergibt sich, daß $f[x_0, x_1, \ldots, x_k]$ eine Linearkombination der Funktionswerte $f(x_0), f(x_1), \ldots, f(x_k)$ ist. Genauer gilt

$$\begin{aligned} f[x_0, x_1, \ldots, x_k] = &\frac{f(x_0)}{(x_0-x_1)\,(x_0-x_2)\cdots(x_0-x_k)}\\ &+\frac{f(x_1)}{(x_1-x_0)\,(x_1-x_2)\cdots(x_1-x_k)}+\cdots\\ &+\frac{f(x_i)}{(x_i-x_0)\,(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_k)}+\cdots\\ &+\frac{f(x_k)}{(x_k-x_0)\,(x_k-x_1)\cdots(x_k-x_{k-1})}. \end{aligned} \tag{11}$$

Beweis (Vollständige Induktion nach k). Für $k=1$ folgt aus (9)

$$f[x_0, x_1] = \frac{f(x_0)}{x_0 - x_1} + \frac{f(x_1)}{x_1 - x_0},$$

was der Formel (11) entspricht.

Möge nun (11) für $k=n$ gelten. Für $k=n+1$ ergibt sich auf Grund von (10) und der Induktionsannahme

$$\begin{aligned}
f[x_0, x_1, \ldots, x_n, x_{n+1}] &= \frac{f[x_1, \ldots, x_n, x_{n+1}] - f[x_0, x_1, \ldots, x_n]}{x_{n+1} - x_0} \\
&= \frac{-f(x_0)}{(x_0 - x_1)(x_0 - x_2) \cdots (x_0 - x_n)} \cdot \frac{1}{x_{n+1} - x_0} \\
&\quad + \sum_{i=1}^{n} \frac{f(x_i)}{x_{n+1} - x_0} \left\{ \frac{1}{(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_{n+1})} \right. \\
&\quad \left. - \frac{1}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)} \right\} \\
&\quad + \frac{1}{x_{n+1} - x_0} \cdot \frac{f(x_{n+1})}{(x_{n+1} - x_1)(x_{n+1} - x_2) \cdots (x_{n+1} - x_n)} \\
&= \frac{-f(x_0)}{(x_0 - x_1)(x_0 - x_2) \cdots (x_0 - x_n)} \cdot \frac{1}{x_{n+1} - x_0} \\
&\quad + \sum_{i=1}^{n} \frac{f(x_i)}{x_{n+1} - x_0} \cdot \frac{(x_i - x_0) - (x_i - x_{n+1})}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_n)(x_i - x_{n+1})} \\
&\quad + \frac{1}{x_{n+1} - x_0} \cdot \frac{f(x_{n+1})}{(x_{n+1} - x_1)(x_{n+1} - x_2) \cdots (x_{n+1} - x_n)} \\
&= \sum_{i=0}^{n+1} \frac{f(x_i)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)(x_i - x_{n+1})},
\end{aligned}$$

was zu beweisen war.

Aus (11) ergibt sich:

Die Steigung k-ter Ordnung ist eine symmetrische Funktion ihrer $k+1$ Argumente. (12)

Damit ist gemeint, daß $f[x_0, x_1, \ldots, x_k]$ invariant ist gegenüber einer Permutation der $x_0, x_1, \ldots, x_k$. Der Beweis ergibt sich sofort aus der Bemerkung, daß jede Permutation als eineindeutige Abbildung einer endlichen Menge auf sich als Produkt von endlich vielen Transpositionen (Vertauschung von zwei Elementen) dargestellt werden kann und es somit genügt, die Invarianz der Steigung für eine Transposition zu zeigen. Bei einer Transposition der Elemente x_i und x_j bleiben aber in (11) alle Terme, die Funktionswerte $f(x_r)$ mit $r \neq i, j$ enthalten, als Ganzes ungeändert, während sich die Summanden mit $f(x_i)$ und $f(x_j)$ vertauschen.

Bezeichnet S_i die mit $f(x)$ gebildete Steigung $(k-1)$-ter Ordnung bezüglich der Argumentstellen $x_0, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k$ (ihre Reihenfolge ist nach (12) belanglos), so kann jetzt $f[x_0, x_1, \ldots, x_k]$ in der Form

$$f[x_0, x_1, \ldots, x_k] = \frac{S_i - S_j}{x_j - x_i} \tag{13}$$

ausgedrückt werden. Die rekursive Bestimmung (10) ist ein Spezialfall von (13). Andererseits läßt sich (13) auf (10) zurückführen, indem man die Argumente in $f[x_0, x_1, \ldots, x_k]$ so anordnet, daß die Anwendung von (10) die Formel (13) liefert.

Wir schließen mit einer Bemerkung über Steigungen mit wiederholtem Argument. Es seien $x_0, x_1, \ldots, x_k, \xi$ paarweise verschiedene Argumente, die dem Definitionsbereich einer Funktion $f(x)$ angehören; $f(x)$ sei ferner an der Stelle ξ differenzierbar. Dann ist auch $f[x_0, x_1, \ldots, x_k, x]$ in einer gewissen Umgebung von ξ definiert und bei ξ differenzierbar. Dies folgt aus (11), da alle dort auftretenden Nenner für hinreichend nahe bei ξ gelegene x von Null verschieden sind und die einzelnen Summanden somit bei ξ eine Ableitung besitzen.

Wir wollen eine sinnvolle Definition für

$$f[x_0, x_1, \ldots, \xi, \ldots, x_i, \ldots, \xi, \ldots, x_k] \tag{14}$$

finden, wenn in dem Klammersymbol die Argumente $x_0, x_1, \ldots, x_k$ auftreten und zweimal an irgendwelchen Positionen ξ vorkommt. Zu diesem Zweck wird eines der Argumente ξ durch $\xi + h$ ersetzt, wobei das Inkrement $h \neq 0$ und so klein sei, daß $x_0, x_1, \ldots, x_k, \xi, \xi + h$ ein System paarweise verschiedener Punkte ist und im übrigen $\xi + h$ dem Definitionsbereich von f angehört. Für die derart modifizierte Klammer (14) kann gemäß (12) und (13)

$$f[\xi, x_0, x_1, \ldots, x_k, \xi + h] = \frac{f[x_0, \ldots, x_k, \xi + h] - f[x_0, \ldots, x_k, \xi]}{h}$$

geschrieben werden, woraus für $h \to 0$

$$\lim_{h \to 0} f[x_0, x_1, \ldots, \xi + h, \ldots, x_i, \ldots, \xi, \ldots, x_k] \tag{15}$$

$$= \frac{d}{dx} f[x_0, x_1, \ldots, x_k, x]\Big|_{x=\xi}$$

folgt. (15) ergibt sich in jeder Position der beiden Argumente ξ, und durch das Resultat dieses Grenzprozesses soll (14) als Steigung $(k+1)$-ter Ordnung mit einmal wiederholtem Argument definiert werden.

Aufgabe 3. a) Man zeige, daß $f[x_0, x_1]$ genau dann von x_0 und x_1 unabhängig ist, wenn $f(x)$ eine lineare Funktion ist $(f(x) = ax + b)$.

b) Es sei $f(x) = u(x)v(x)$. Man zeige:

$$f[x_0, x_1] = u[x_0]v[x_0, x_1] + u[x_0, x_1]v[x_1]$$

und allgemein

$$f[x_0, \ldots, x_n] = \sum_{k=0}^{n} u[x_0, \ldots, x_k] v[x_k, \ldots, x_n].$$

4.2.3. Newtonsche Interpolationsformel

Es sei f eine Funktion, die zumindest an den im folgenden genannten Argumentstellen definiert ist. Diese selbst seien als paarweise verschieden vorausgesetzt. Dann gilt definitionsgemäß für die mit f zu bildenden Steigungen:

$$\begin{aligned}
(x - x_0)\,[x, x_0] &= f(x) - f(x_0), \\
(x - x_1)\,[x, x_0, x_1] &= [x, x_0] - [x_0, x_1], \\
(x - x_2)\,[x, x_0, x_1, x_2] &= [x, x_0, x_1] - [x_0, x_1, x_2], \\
&\ldots \\
(x - x_n)\,[x, x_0, x_1, x_2, \ldots, x_n] &= [x, x_0, x_1, \ldots, x_{n-1}] - [x_0, x_1, \ldots, x_n].
\end{aligned} \tag{16}$$

Durch Einsetzen von $[x, x_0]$ aus der zweiten Gleichung in die erste erhält man

$$f(x) = f(x_0) + (x - x_0)\,[x_0, x_1] + (x - x_0)\,(x - x_1)\,[x, x_0, x_1].$$

Durch Elimination von $[x, x_0, x_1]$ mit Hilfe der dritten Gleichung von (16) ergibt sich weiter

$$\begin{aligned}
f(x) = f(x_0) &+ (x - x_0)\,[x_0, x_1] + (x - x_0)\,(x - x_1)\,[x_0, x_1, x_2] \\
&+ (x - x_0)\,(x - x_1)\,(x - x_2)\,[x, x_0, x_1, x_2]
\end{aligned}$$

und so fortfahrend schließlich

$$\begin{aligned}
f(x) = f(x_0) &+ (x - x_0)\,[x_0, x_1] + (x - x_0)\,(x - x_1)\,[x_0, x_1, x_2] + \cdots \\
&+ (x - x_0)\,(x - x_1) \cdots (x - x_{n-1})\,[x_0, x_1, x_2, \ldots, x_n] + R_n
\end{aligned} \tag{17}$$

mit

$$R_n = (x - x_0)\,(x - x_1) \cdots (x - x_{n-1})\,(x - x_n)\,[x, x_0, x_1, x_2, \ldots, x_n]. \tag{18}$$

(17) ist eine Identität, die wir *Newtonsche Darstellungsformel* nennen wollen; sie wird uns in Verbindung mit Aussagen über das *Restglied* R_n nützlich sein. Man vergleiche die Sachlage mit der Anwendung der Taylorschen Formel. Die rechte Seite von (17) ohne das Restglied R_n ist ein Polynom höchstens n-ten Grades, das wir mit $N(x)$, falls erforderlich deutlicher mit $N_f(x)$, bezeichnen; entsprechend wird R_n durch $R_{f,n}$ ersetzt. Es gilt

$$N_f(x_j) = f(x_j) \quad \text{für} \quad j = 0, 1, \ldots, n. \tag{19}$$

Beweis. Wir wenden die Newtonsche Darstellungsformel (17) auf das mit den Werten $f(x_i)$ an den Stellen x_i $(i = 0, 1, \ldots, n)$ gebildete Lagrangesche Interpola-

tionspolynom $L(x)$ an. Dann ist $L(x) = N_L(x) + R_{L,n}(x)$. In dieser Gleichung lassen wir x gegen x_j streben. Da L als Polynom differenzierbar ist, existiert

$$\lim_{x \to x_j} L\,[x, x_0, x_1, \ldots, x_n]$$

und ist gleich dem am Schluß des vorigen Abschnitts ermittelten Wert (15). Wegen des in (18) auftretenden Linearfaktors $(x - x_j)$ erhält man also $\lim_{x \to x_j} R_{L,n}(x) = 0$. Die Polynome L und N_L streben aus Stetigkeitsgründen gegen $L(x_j)$ bzw. $N_L(x_j)$, so daß $L(x_j) = N_L(x_j)$ und wegen $L(x_j) = f(x_j)$ auch

$$N_L(x_j) = f(x_j) \tag{20}$$

gilt. In den Ausdruck von $N_f(x)$ gehen aber von der Funktion f nur deren Werte an den Stellen x_i ein, woraus $N_L = N_f$ und schließlich wegen (20) $N_f(x_j) = f(x_j)$ resultiert.

Das Polynom N_f ist auf Grund von Satz 1 und (19) eine weitere Form der Lösung von (1), die *Newtonsches Interpolationspolynom* genannt wird. Für die Berechnung der Steigungen, die als Koeffizienten in N_f auftreten, werden wir einen Algorithmus entwickeln, der bei Einbeziehung weiterer Interpolationsstellen die Fortsetzung der Rechnung erlaubt (vgl. die Bemerkung am Schluß von 4.2.1.). Für die Auswertung braucht N_f nicht nach Potenzen von x geordnet zu werden, da auch für Polynome in dieser sogenannten Newtonschen Produktform ein hornerartiger Algorithmus zur Verfügung steht. Bevor wir auf diese Sachverhalte eingehen, soll aus (17) noch eine Folgerung bezüglich der Steigungen höherer Ordnung bei einem Polynom gezogen werden:

Bei einem Polynom von höchstens n-tem Grade verschwinden sämtliche Steigungen $(n+1)$-ter und höherer Ordnung. (21)

Beweis. Es sei P ein der Voraussetzung von (21) genügendes Polynom. Wir betrachten $n+1$ paarweise verschiedene Argumentstellen x_i $(i = 0, 1, \ldots, n)$ und wenden darauf bezüglich P die Newtonsche Darstellungsformel (17) an:

$$P(x) = N_P(x) + (x - x_0)\,(x - x_1) \cdots (x - x_n)\,P[x, x_0, x_1, \ldots, x_n].$$

$N_P(x) - P(x)$ ist ein Polynom höchstens n-ten Grades, das an den Stellen x_i $(i = 0, 1, \ldots, n)$ verschwindet. Auf Grund des Fundamentalsatzes der Algebra ist also $P(x) = N_P(x)$ und damit, falls $x \neq x_i$,

$$P[x, x_0, x_1, \ldots, x_n] = 0.$$

Da die Stellen $x, x_0, x_1, \ldots, x_n$, abgesehen von der Forderung, paarweise verschieden zu sein, beliebig gewählt werden können, ist das Verschwinden aller Steigungen $(n+1)$-ter Ordnung für P gezeigt. Dann müssen aber a fortiori auch alle Steigungen noch höherer Ordnung verschwinden.

Die Berechnung der im Ausdruck von $N_f(x)$ vorkommenden dividierten Differenzen $[x_0, x_1]$, $[x_0, x_1, x_2]$, $\ldots$, $[x_0, x_1, \ldots x_n]$ besorgt man von Hand zweckmäßig mit Hilfe des in Tabelle 4.4 angegebenen Schemas, das auch die Grundlage für die

x	$f(x)$					
x_0	y_0					
		y_1-y_0				
x_1	y_1		$[x_1,x_0]$			
		y_2-y_1		$[x_2,x_1]-[x_1,x_0]$		
x_2	y_2		$[x_2,x_1]$		$[x_2,x_1,x_0]$	
		y_3-y_2		$[x_3,x_2]-[x_2,x_1]$		$[x_3,x_2,x_1,x_0]$
x_3	y_3		$[x_3,x_2]$		$[x_3,x_2,x_1]$	
		x_1-x_0				
			x_2-x_0			
		x_2-x_1		x_3-x_0		
			x_3-x_1			
		x_3-x_2				

Tabelle 4.4

anschließende Programmierung bildet. In zwei zentralen Spalten stehen zu Anfang die x_i, y_i $(i=0, 1, \ldots, n)$. Schrittweise wird dann das Schema nach beiden Seiten entwickelt (j-ter Schritt): Auf der linken Seite bildet man die Differenzen $x_i - x_{i-j}$, $i=n(-1)j$, und notiert sie in einer Spalte; auf der rechten Seite entsprechen diesen Differenzen zwei Spalten, in die man $z_i - z_{i-1}$ bzw. $\frac{z_i - z_{i-1}}{x_i - x_{i-j}}$, $i=n(-1)j$, einträgt, wobei die z_i die in der zweiten Spalte des $(j-1)$-ten Schrittes stehenden Werte bedeuten; für $j=1$ ist $z_i := y_i - y_{i-1}$ zu definieren. Es erhöht die Übersichtlichkeit, wenn man die Zeilen des Schemas schrittweise versetzt; sein Aufbau läßt erkennen, daß sich eine mit gewissen Stellen x_i durchgeführte Berechnung der Klammerterme fortsetzen läßt, wenn weitere Stellen in die Interpolation einbezogen werden sollen. Im Newtonschen Polynom sind entsprechend die weiteren Linearfaktorterme additiv hinzuzufügen. Man beachte, daß bisher keine Forderungen bezüglich der Anordnung der Stützstellen erhoben wurden.

Wir fassen die Berechnung der Steigungskoeffizienten in einer Prozedur *NEWKO*(X, Y, Z, n) zusammen. X, Y bedeuten eindimensionale Felder, welche die x_i bzw. y_i $(i=0, 1, \ldots, n)$ aufnehmen. Z ist ebenfalls ein Feld vom Typ **array** $Z[0{:}n]$, das während der Rechnung mit den z_i aktualisiert wird und am Ende die gesuchten Steigungen enthält. Lokal vereinbaren wir ein Feld **array** $X1[1{:}n]$ und belegen dieses sukzessive mit den Abszissendifferenzen des linken Teils unseres Schemas. Der zu programmierende Algorithmus entspricht im übrigen vollkommen dessen oben beschriebenen Aufbau.

```
procedure NEWKO(X, Y, Z, n); value n;
integer n; array X, Y, Z;
  begin integer i, j; array X1[1 : n];
    for i := 0 step 1 until n do Z[i] := Y[i];
    for j := 1 step 1 until n do
    for i := n step −1 until j do begin
    X1[i] := X[i] − X[i − j];
    Z[i] := Z[i] − Z[i − 1];
    Z[i] := Z[i]/X1[i]                    end
  end
```

Für die Auswertung des Newtonschen Interpolationspolynoms entwickeln wir einen hornerartigen Algorithmus zur Berechnung von Polynomen der Form

$$N(x) = b_0 + b_1(x-x_0) + b_2(x-x_0)\,(x-x_1) + \ldots + b_n(x-x_0)\,(x-x_1)\cdots(x-x_{n-1}). \tag{22}$$

Wie im Anschluß an 1.3.1.(1) definieren wir rekursiv Polynome $N_0, N_1, \ldots, N_{n-1}, N_n$ gemäß

$$\begin{array}{l|l} N_0(x) = b_n & (x-x_{n-1})\,(x-x_{n-2})\cdots(x-x_0) \\ N_1(x) = b_{n-1} + (x-x_{n-1})N_0(x) & (x-x_{n-2})\cdots(x-x_0) \\ N_2(x) = b_{n-2} + (x-x_{n-2})N_1(x) & \\ \quad\vdots & \quad\vdots \\ N_{n-1}(x) = b_1 + (x-x_1)N_{n-2}(x) & (x-x_0) \\ N_n(x) = b_0 + (x-x_0)N_{n-1}(x) & 1 \end{array} \tag{23}$$

Dann gilt für alle x

$$N_n(x) = N(x). \tag{24}$$

Zum Beweis multipliziere man die Gleichungen mit den in (23) angegebenen Linearfaktoren und addiere. Wie in 2.2.(6) heben sich dabei Terme auf der rechten und linken Seite gegenseitig auf, und es verbleibt

$$N_n(x) = b_0 + b_1(x-x_0) + \cdots + b_n(x-x_0)\,(x-x_1)\cdots(x-x_{n-1}).$$

(24) ist die Grundlage für folgendes Schema zur Berechnung der Werte von N, das man im Sinne der 2.2.(8) beigefügten Erläuterungen zu lesen hat:

$$\begin{array}{lllll} b_n & b_{n-1} & b_{n-2} & \ldots\ b_1 & b_0 \\ & b'_n(x-x_{n-1}) & b'_{n-1}(x-x_{n-2}) & \ldots\ b'_2(x-x_1) & b'_1(x-x_0) \\ \hline b'_n & b'_{n-1} & b'_{n-2} & \ldots\ b'_1 & b'_0 = N(x) \end{array} \tag{25}$$

Wir fassen die Berechnung von $N(x)$ gemäß (25) in einer Funktionsprozedur *NEWPOL*(n, B, x) zusammen, die sich bei entsprechender Zuordnung der formalen Parameter mit der Prozedur *HORNER*(m, B, t) aus 3.3. vergleichen läßt.

```
real procedure NEWPOL(n, B, x); value n, x;
integer n; real x; array B;
  begin integer i; real p;
    p := B[n];
    for i := n − 1 step −1 until 0 do p := B[i] + (x − X[i]) × p;
    NEWPOL := p
  end
```

X bedeutet das in *NEWPOL* globale Feld der Größen

$$X[j] := x_j \quad (j = 0, 1, \ldots, n-1).$$

Aufgabe 4. Man schreibe ein R 300-ALGOL-Programm, das die Prozeduren *NEWKO*, *NEWPOL* und *MINIMAX* (vgl. 3.3.) benutzt und folgendes leistet: Zu den Punkten (x_i, y_i), $i=0, 1, 2, \ldots$, mit paarweise verschiedenen Abszissen wird das Newtonsche Interpolationspolynom bestimmt und mit äquidistanten Schritten der Weite h zwischen der kleinsten und der größten Stützstelle tabuliert. Konkret betrachte man etwa folgendes Beispiel: Die größte zulässige Belastung B einer eisernen Kette in Abhängigkeit vom Kettenglieddurchmesser d ist im folgenden Schema angegeben:

i	0	1	2	3	4	5	6	7
$x=d$ [mm]	5	8	10	16	20	26	30	36
$y=B$ [kp]	160	550	950	2500	3800	6400	8500	12500

Man benutze die Werte x_i mit $i=0, 2, 3, 5, 7$ zur Interpolation und vergleiche die an den übrigen Stützstellen berechneten Werte mit den gemessenen.

4.2.4. Fehlerbetrachtungen

Für den verfahrensbedingten Fehler bei der Approximation einer Funktion f durch das an den Stellen x_i $(i=0, 1, \ldots, n)$ bezüglich der Ordinaten $y_i=f(x_i)$ gebildete Interpolationspolynom P wurde in 4.2.3. der Ausdruck

$$R_n(x)=f[x, x_0, x_1, \ldots, x_n]\,\omega_n(x), \tag{26}$$

$$\omega_n(x) := (x-x_0)\,(x-x_1)\cdots(x-x_n)$$

gefunden. Wir wollen hier eine andere Darstellung des Restgliedes erörtern, zu deren Herleitung jedoch weitergehende Voraussetzungen über die Funktion f erforderlich sind. Es werde angenommen, daß die zu interpolierende Funktion auf einem Intervall $[\![a, b]\!]$, das sämtliche Stützstellen enthält, n-mal stetig differenzierbar ist und die $(n+1)$-te Ableitung im Innern desselben existiert. Die Funktion

$$\varphi(x) := f(x) - P(x) - \varkappa\omega_n(x) \tag{27}$$

verschwindet bei beliebiger Wahl der reellen Konstanten $\varkappa$ an den Stellen x_i $(i=0, 1, \ldots, n)$. Durch geeignete Verfügung über $\varkappa$ soll erreicht werden, daß noch für ein weiteres $\bar{x}$ $(\bar{x}\neq x_i)$

$$\varphi(\bar{x})=0$$

ist. Das ist offenbar auf genau eine Weise möglich. $\varkappa$ bedeute im weiteren den (von $\bar{x}$ abhängigen) durch diese Forderung bestimmten Wert

$$\varkappa(\bar{x}) := \frac{f(\bar{x})-P(\bar{x})}{\omega_n(\bar{x})} \quad (\bar{x}\neq x_i\,;\ i=0, 1, \ldots, n); \tag{28}$$

$\bar{I}$ sei das von $\min(\bar{x}, x_0, x_1, \ldots, x_n)$ und $\max(\bar{x}, x_0, x_1, \ldots, x_n)$ berandete abgeschlossene Teilintervall von $[\![a, b]\!]$. Auf $\bar{I}$ verschwindet die durch (27) definierte Funktion φ an mindestens $n+2$ Stellen, und φ' hat daher nach dem Satz von Rolle im Innern von $\bar{I}$ mindestens $n+1$ Nullstellen. In Fortführung dieser Schluß-

weise erkennt man, daß φ'' wenigstens n-mal und schließlich $\varphi^{(n+1)}$ wenigstens einmal im Innern von I, etwa bei $\xi_{\tilde{x}}$, verschwindet. Wir haben ferner

$$P^{(n+1)}(x) \equiv 0,$$

da P ein Polynom von höchstens n-tem Grade ist, und

$$\omega_n^{(n)}(x) \equiv (n+1)!.$$

$(n+1)$-malige Differentiation von (27) an der Stelle $\xi_{\tilde{x}}$ liefert damit

$$\varkappa(\tilde{x}) = \frac{f^{(n+1)}(\xi_{\tilde{x}})}{(n+1)!}$$

als eine aus der Definition (28) resultierende Darstellung von $\varkappa$ durch die $(n+1)$-te Ableitung von f. Da für $\tilde{x}$ ein beliebiger, von den Stützstellen x_i verschiedener Wert x des Intervalls $[a, b]$ gewählt werden kann, gilt allgemein

$$x \in \bigwedge_{[a,b]} \bigvee_{\xi} \left(a < \xi < b \wedge R_n(x) := f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\, \omega_n(x) \right). \tag{29}$$

An den Stützstellen x_i gilt (29) trivialerweise, da dort ω_n verschwindet. Aus dem Beweis des letzten Satzes ergibt sich, daß die Lage eines (29) genügenden, von x abhängigen ξ genauer durch

$$\min(x, x_0, \ldots, x_n) < \xi < \max(x, x_0, \ldots, x_n) \tag{30}$$

bestimmt werden kann. Für die Klasse der auf $[a, b]$ n-mal stetig differenzierbaren Funktionen, deren $(n+1)$-te Ableitung im Inneren des Intervalls existiert und beschränkt ist, gilt bei fixierten Stützstellen x_i generell

$$|R_n(x)| \leqq \frac{M_{n+1}}{(n+1)!} |\omega_n(x)| \leqq \frac{M_{n+1}}{(n+1)!} \max_{a \leqq x \leqq b} |\omega_n(x)|, \tag{31}$$

wobei

$$M_{n+1} := \sup_{a < x < b} |f^{(n+1)}(x)|.$$

Die Abschätzung (31) ist insofern scharf, als bei einem Polynom $(n+1)$-ten Grades für mindestens ein x aus $[a, b]$

$$|R_n(x)| = \frac{M_{n+1}}{(n+1)!} \max_{a \leqq x \leqq b} |\omega_n(x)|$$

gilt.

Durch einen Vergleich der Restglieddarstellung in (26) und (29) gewinnt man für die Steigungen einer genügend oft differenzierbaren Funktion f den Satz:

Es sei f auf $[a, b]$ n-mal differenzierbar und besitze im Innern dieses Intervalls eine $(n+1)$-te Ableitung. Dann gilt für jedes System $x, x_0, x_1, \ldots, x_n$ paarweise verschiedener Punkte aus $[a, b]$

$$f[x_0, x_1, \ldots, x_n, x] = \frac{1}{(n+1)!}\, f^{(n+1)}(\xi_x) \tag{32}$$

für ein gewisses ξ_x mit $\min(x, x_0, \ldots, x_n) < \xi_x < \max(x, x_0, \ldots, x_n)$.

Durch die Restgliedabschätzung (31) lasse man sich nicht zu dem falschen Schluß verleiten, der Interpolationsfehler könne in jedem Fall über dem gesamten Intervall $[a, b]$ dadurch beliebig klein gemacht werden, daß man die Anzahl der Stützstellen erhöht. Als Gegenbeispiel (nach [63], Bd. I, S. 19) betrachten wir die rationale Funktion $y = \frac{1}{1+x^2}$ über dem Intervall $[-8, 8]$. Abb. 4.12 zeigt den Graphen der Funktion und der mit den Stützstellen 0, ± 2, ± 4, ± 6, ± 8 (linker Teil) bzw. 0, $\pm 1, \pm 2, \pm 3, \pm 4, \pm 5, \pm 6, \pm 7, \pm 8$ (rechter Teil) gebildeten Interpolationspoly-

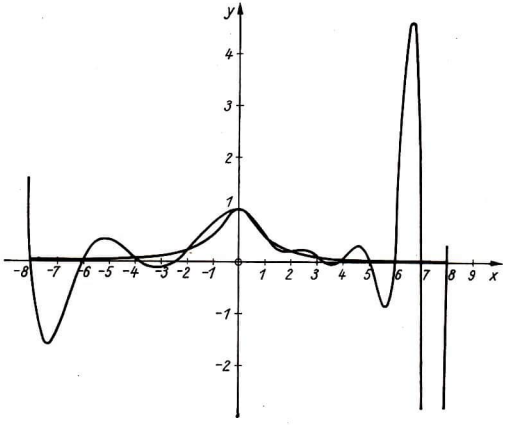


Abb. 4.12

nome. Die Maximalabweichung des zweiten ist beträchtlich größer als die des ersten. Für eine gewisse Funktionenklasse läßt sich das Verhalten des Restgliedes mit Hilfsmitteln der Funktionentheorie klären (vgl. [16]).

Wesentlich geringere Abweichungen ergeben sich, wenn man zur Interpolation an Stelle von Polynomen sogenannte *Splinefunktionen* verwendet, die zwischen den Knoten abteilungsweise aus (im allgemeinen verschiedenen) Polynomen eines gewissen Maximalgrades m zusammengesetzt sind. Ist $x_0 < x_1 < \cdots < x_n$, so nennt man S eine Splinefunktion m-ten Grades mit den Knoten x_j $(j = 0, 1, \ldots, n)$, wenn

a) S in jedem der Intervalle $]-\infty, x_0[$, $]x_j, x_{j+1}[$ $(j = 1, 2, \ldots, n-1)$, $]x_n, \infty[$ ein Polynom höchstens m-ten Grades ist und

b) S (falls $m > 0$) für $-\infty < x < \infty$ stetige Ableitungen bis zur Ordnung $m-1$ einschließlich besitzt.

Es gilt der Satz:

Jede Splinefunktion S vom Grad m mit den Knoten $x_0, x_1, \ldots, x_n$ läßt sich genau auf eine Weise in der Form

$$S(x) = P_m(x) + \sum_{j=0}^{n} c_j (x - x_j)_+^m \tag{33}$$

darstellen, wobei P_m ein Polynom vom Grad $\leqq m$, $c_j \in \mathbf{R}$ $(j = 1, \ldots, n)$ und $z_+^m = z^m$ für $z > 0$, $z_+^m = 0$ für $z \leqq 0$ ist.

Von besonderem Interesse für die Praxis sind Splinefunktionen dritten Grades; der mit ihrer Handhabung verbundene Rechenaufwand übersteigt nicht wesentlich den, der bei der Interpolation mit ganzrationalen Funktionen zu bewältigen ist. Ein Hilfsmittel sind dabei die in 4.2.2. eingeführten Steigungen (vgl. [26], [27]).

Wir werden auf Theorie und Anwendungen von Splinefuntion in MfL Bd. 10 zurückkommen.

Neben dem im Restglied R_n seinen Ausdruck findenden verfahrensbedingten Fehler bei der Interpolation sind die Rundungsfehler zu berücksichtigen[1]), wobei es wesentlich auf den Algorithmus ankommt, welcher der Berechnung der Polynomwerte zugrunde liegt. Wir machen dazu eine Bemerkung bezüglich der Newtonschen Formel. Wenn die Interpolationspunkte (x_i, y_i), $i = 0, 1, \ldots, n$, fest vorgegeben sind, kann der sie repräsentierende arithmetische Ausdruck dadurch modifiziert werden, daß man die Punkte umnumeriert. Dabei ist es übrigens nicht erforderlich, für jede dieser Anordnungen ein besonderes Differenzenschema aufzustellen. Die

		Steigungen der Ordnung			
x	y	1	2	3	4
x_0	y_0				
		$[x_1, x_0]$			
x_1	y_1		$[x_2, x_1, x_0]$		
		$[x_2, x_1]$		$[x_3, x_2, x_1, x_0]$	
x_2	y_2		$[x_3, x_2, x_1]$		$[x_4, x_3, x_2, x_1, x_0]$
		$[x_3, x_2]$		$[x_4, x_3, x_2, x_1]$	
x_3	y_3		$[x_4, x_3, x_2]$		
		$[x_4, x_3]$			
x_4	y_4				
⋮					

Tabelle 4.5

[1]) Ungenauigkeiten in den Ausgangsdaten (x_i, y_i), $i = 0, 1, 2, \ldots, n$, können technisch wie Rundungsfehler behandelt werden.

Verhältnisse lassen sich übersichtlich beschreiben, wenn wir zunächst diejenige Indizierung betrachten, welche der Größenanordnung der Abszissen entspricht:

$$x_0 < x_1 < x_2 < \cdots < x_n.$$

Tabelle 4.5 zeigt das damit gebildete (vereinfachte) Differenzenschema der Tabelle 4.4 ($n=4$). Das zugehörige Newtonsche Polynom entnimmt man (17); die als Koeffizienten benötigten Differenzen erscheinen auf einem bestimmten in das Schema der Tabelle 4.5 gelegten Weg: auf der von links oben nach rechts unten fallenden Berandung des Steigungsdreiecks. Wir betrachten nun beispielsweise das Interpolationspolynom P in der Form, die der Abszissenfolge x_3, x_2, x_4, x_1, x_0 entspricht. Die Newtonsche Formel liefert dafür

$$\begin{aligned} P(x) = y_3 &+ (x-x_3)\,[x_3, x_2] + (x-x_3)\,(x-x_2)\,[x_3, x_2, x_4] \\ &+ (x-x_3)\,(x-x_2)\,(x-x_4)\,[x_3, x_2, x_4, x_1] \\ &+ (x-x_3)\,(x-x_2)\,(x-x_4)\,(x-x_1)\,[x_3, x_2, x_4, x_1, x_0]. \end{aligned}$$

Unter Beachtung, daß die Steigungen symmetrische Funktionen ihrer Argumente sind, findet man die Koeffizienten jetzt auf dem mit einer unterbrochenen Linie gezogenen Weg. Es läßt sich zeigen, daß jeder Anordnung der Interpolationspunkte mit Bezug auf die Newtonsche Formel ein solcher Zickzackweg (der Stufenhöhe 1) im Differenzenschema entspricht, der immer an der Spitze des Steigungsdreiecks endet, und umgekehrt gibt es zu jedem solchen Weg genau eine korrespondierende Punktanordnung. Die eingeklammerte Bemerkung soll zum Ausdruck bringen, daß beim Fortschreiten in die Spalte der nächst höheren Differenz nur die unmittelbar über oder unter dem Ausgangspunkt stehende Differenz angegangen wird.

Wie wird man nun die Punktanordnung wählen, wenn mit Hilfe eines Interpolationspolynoms näherungsweise $f(x)$ berechnet werden soll? Es erscheint plausibel, für x_0 und x_1 in der Newtonschen Formel gemäß (17) die der Abszisse x nächstgelegenen Stützstellen zu wählen und nach wachsender Entfernung weitere x_i einzubeziehen. Dabei wird man möglichst symmetrisch zum Bezugspunkt x vorgehen. Zur Rechtfertigung kann zunächst gesagt werden, daß die auf diese Weise gebildeten Linearfaktorenprodukte, mit denen man die Steigungen zu multiplizieren hat, kleiner sind als die bei einer anderen Punktanordnung. Daher ist die Auswirkung der bei den Steigungen kumulierten Rundungsfehler nach 2.5.3(16) entsprechend geringer; sie können unter Umständen sogar gedämpft werden. Bei fest vorgegebenen Stützstellen ist dieser Effekt jedoch meist von untergeordneter Bedeutung. Wichtiger ist der Einfluß des Verfahrensfehlers, wenn man – etwa in einer Tafel von Funktionswerten – die Stützstellen bezüglich x in der geschilderten Weise auswählen kann. Beginnt man dabei mit linearer Interpolation und bildet durch Einbeziehung weiterer benachbarter Stützstellen sukzessive Interpolationspolynome höherer Ordnung, so läßt sich die Approximationsgüte merklich steigern. Wie weit man dabei gehen kann, hängt vom Verhalten der Werte im Steigungsschema ab. Bei einem Polynom sind die dividierten Differenzen (exakte Berechnung vorausgesetzt) von einer gewissen Ordnung an gleich Null (vgl. (21)). Ist die

zu interpolierende Funktion hinreichend glatt, so tritt statt dessen häufig folgendes ein: Zunächst werden die Differenzen mit wachsender Ordnung klein, verhalten sich dann unregelmäßig und schaukeln sich infolgedessen weiterhin dem Betrage nach auf. Die Ursache für dieses Phänomen ist in den Ungenauigkeiten der Ausgangsdaten und den sich kumulierenden Rundungsfehlern beim Aufstellen des Differenzenschemas zu suchen. Offensichtlich hat es keinen Sinn, Differenzen zu benutzen, die im Bereich der unregelmäßigen Wertestreuung liegen.

Auf Grund des geschilderten Sachverhalts ist die Berechnung Newtonscher Interpolationspolynome nach dem Algorithmus (23) aus den gleichen Gründen vorteilhaft, die am Schluß von 2.5.4. bezüglich des gewöhnlichen Hornerschen Schemas genannt wurden.

4.2.5. Interpolation bei äquidistanten Stützstellen. Spezielle Interpolationsformeln

Besonders für Anwendungen von Interpolationspolynomen bei der Benutzung von Tafeln ist der Fall äquidistanter Stützstellen x_i von Interesse. Für das Weitere sei vereinbart, diese ihrer Größenanordnung entsprechend zu indizieren; die konstante Schrittweite zweier aufeinanderfolgender Stützstellen sei mit h bezeichnet. Diese Spezialisierung gestattet es, einen Differenzenkalkül zu entwickeln, mit dessen Hilfe sich die Berechnung der mit den x_i zu bildenden Steigungen und daraus abgeleiteter Größen wesentlich vereinfachen läßt.

Ist f Funktion der reellen Veränderlichen x und an allen betrachteten Stellen erklärt, so sei bezüglich der Schrittweite h

$$\Delta f(x) := f(x+h) - f(x) = h\, f[x, x+h]. \tag{34}$$

Mit Hilfe des in (34) erklärten Differenzenoperators definieren wir induktiv für f sogenannte *vordere Differenzen* Δ^k beliebiger Ordnung k ($k \in \mathbb{N}^*$):

$$\begin{aligned} \Delta^1 f(x) &:= \Delta f(x), \\ \Delta^{k+1} f(x) &:= \Delta \Delta^k f(x) = \Delta^k f(x+h) - \Delta^k f(x). \end{aligned} \tag{35}$$

Speziell an den Stützstellen x_i würde für $k = 1, 2$ gelten:

$$\begin{aligned} \Delta^1 f(x_i) &= h\, f[x_i, x_{i+1}], \\ \Delta^2 f(x_i) &= h\{\Delta^1 f(x_{i+1}) - \Delta^1 f(x_i)\} \\ &= h\{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]\} \\ &= 2h^2 f[x_i, x_{i+1}, x_{i+2}]. \end{aligned}$$

Durch vollständige Induktion zeigt man leicht

$$\Delta^k f(x_i) = k!\, h^k f[x_i, x_{i+1}, \ldots, x_{i+k}]. \tag{36}$$

Aufgabe 5. Man beweise (36).

Mit Hilfe der vorderen Differenzen kann man die Koeffizienten des mit

$$x_0,\quad x_1=x_0+h,\quad x_2=x_0+2h,\quad \ldots,\quad x_n=x_0+nh$$

gebildeten Newtonschen Interpolationspolynoms ausdrücken. Auf Grund von (36) ist nämlich, wenn zur Abkürzung $\Delta_i^k := \Delta^k f(x_i)$ gesetzt wird,

$$N_f(x) = f(x_0) + \frac{1}{1!h}\,\Delta_0^1 (x-x_0) + \frac{1}{2!h^2}\,\Delta_0^2 (x-x_0)\,(x-x_1) + \cdots + \frac{1}{n!h^n}\,\Delta_0^n (x-x_0)\,(x-x_1)\cdots(x-x_{n-1}). \tag{37}$$

Die in (37) benötigten Δ_i^k kann man dem Schema der Tabelle 4.6 entnehmen, dessen k-te Spalte die Differenzen aufeinanderfolgender Werte in der $(k-1)$-ten enthält $(k>1)$ und ausgehend von den Funktionswerten f_i an den Stützstellen x_i aufgebaut wird. Die interessierenden Δ_i^k liegen auf dem in das Differenzenschema eingetragenen oberen Weg.

x	$f(x)$	Δ^1	Δ^2	Δ^3	$\ldots$
$\vdots$	$\vdots$				
x_0	f_0	Δ_0^1			
x_1	f_1	Δ_1^1	Δ_0^2	Δ_0^3	
x_2	f_2	Δ_2^1	Δ_1^2	$\vdots$	
x_3	f_3	$\vdots$	$\vdots$	Δ_{n-3}^3	
$\vdots$	$\vdots$	Δ_{n-1}^1	Δ_{n-2}^2		
x_n	f_n				
$\vdots$	$\vdots$				

Tabelle 4.6

Es ist zweckmäßig, durch eine lineare Transformation der Form

$$x = a + hs \tag{38}$$

die Interpolationsaufgabe so zu normieren, daß die Schrittweite den Wert 1 annimmt. Wählt man in (38) für a den Wert x_0, so sind die den x_i entsprechenden

s-Werte deren Indizes:

$$x_i = x_0 + hi. \tag{39}$$

Wir bezeichnen das mit (38) ($a = x_0$) transformierte Polynom (37) mit $\hat{N}_f(s)$, so daß

$$\hat{N}_f(s) = N_f(x_0 + hs) = f_0 + \frac{1}{1!}\Delta_0^1 s + \frac{1}{2!}\Delta_0^2 s(s-1) + \cdots + \frac{1}{n!}\Delta_0^n s(s-1)\cdots(s-n+1). \tag{40}$$

Offenbar ist das transformierte Polynom (37) identisch mit dem an den Stellen $s = 0, 1, \ldots, n$ gebildeten Interpolationspolynom zur Funktion $\hat{f}(s) = f(x_0 + hs)$. Nehmen wir f und damit auch $\hat{f}$ als $(n+1)$-mal differenzierbar an, so gilt nach (29)

$$\hat{f}(s) = N_{\hat{f}}(s) + \frac{\hat{f}^{(n+1)}(\xi_s)}{(n+1)!}\, s(s-1)\,(s-2)\cdots(s-n),$$

und daher ist wegen $N_{\hat{f}}(s) = \hat{N}_f(s)$

$$\hat{R}_n(s) = \frac{\hat{f}^{(n+1)}(\xi_s)}{(n+1)!}\, s(s-1)\,(s-2)\cdots(s-n) \tag{41}$$

das transformierte Restglied. Wegen

$$\hat{f}'(s) = hf'(x), \quad \hat{f}''(s) = h^2 f''(x), \quad \ldots, \quad \hat{f}^{(n+1)}(s) = h^{n+1} f^{(n+1)}(x) \tag{42}$$

kann man (41) auch in der Form

$$\hat{R}_n(s) = \frac{h^{n+1}}{(n+1)!}\, f^{(n+1)}(\xi_x) s(s-1)\,(s-2)\cdots(s-n) \tag{43}$$

ausdrücken, wobei die Lage von ξ_x durch (30) bestimmt ist. Die normierende Transformation (38) bietet die Möglichkeit, die im Interpolationspolynom auftretenden Linearfaktorenprodukte unabhängig von der speziellen Lage der äquidistanten Stützstellen zu tabulieren.

Im Hinblick auf das am Schluß von 4.2.4. bezüglich der Auswahl der Stützstellen Gesagte eignet sich das Polynom (37) bzw. (40) zur Interpolation am Anfang einer Funktionstafel. Am Ende einer solchen würde man die Stützstellen hingegen in der Anordnung $x_n, x_{n-1}, \ldots, x_0$ zum Aufbau der Newtonschen Formel benutzen. Diese hätte dann nach (17) die Gestalt

$$N_f(x) = f_n + (x - x_n)\,[x_n, x_{n-1}] + (x - x_n)\,(x - x_{n-1})\,[x_n, x_{n-1}, x_{n-2}] + \cdots + (x - x_n)\,(x - x_{n-1})\cdots(x - x_1)\,[x_n, x_{n-1}, \ldots, x_1, x_0],$$

woraus mit Beachtung von (36)

$$N_f(x) = f_n + \frac{1}{1!\,h}\,\Delta_{n-1}^1\,(x - x_n) + \frac{1}{2!\,h^2}\,\Delta_{n-2}^2\,(x - x_n)\,(x - x_{n-1}) + \cdots + \frac{1}{n!\,h^n}\,\Delta_0^n (x - x_n)\,(x - x_{n-1})\cdots(x - x_1) \tag{44}$$

folgt. Setzt man in (38) $a = x_n$, so entsprechen den Stützstellen s-Werte gemäß der Zuordnung

$$x_i = x_n + h(i-n), \tag{45}$$

d. h., es ist

$$s_i = i - n \quad (i = 0, 1, \ldots, n).$$

Analog zu (40) findet man durch Transformation mit $x = x_n + hs$

$$\hat{N}_f(s) = f_n + \frac{1}{1!}\Delta^1_{n-1}\, s + \frac{1}{2!}\Delta^2_{n-2}\, s(s+1) + \cdots + \frac{1}{n!}\Delta^n_0\, s(s+1)\cdots(s+n-1) \tag{46}$$

und für das Restglied entsprechend (41) und (43)

$$\hat{R}_n(s) = \frac{\hat{f}^{(n+1)}(\xi_s)}{(n+1)!}\, s(s+1)\cdots(s+n-1) = \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(\xi_x) s(s+1)\cdots(s+n-1). \tag{47}$$

Bei Interpolation in der Mitte einer Tafel würde es sich empfehlen, abwechselnd die vor und nach x jeweils nächstgelegenen Stützstellen zu benutzen. Es ist zweckmäßig, die erste mit x_0 und die weiteren ihrer Lage zu x_0 entsprechend mit $x_1, x_{-1}, x_2, x_{-2}, \ldots$ usw. zu bezeichnen. Die resultierenden Interpolationspolynome werden nach Gauss benannt; wir bezeichnen diese mit $G_f^{(\mathrm{I})}$ bzw. $G_f^{(\mathrm{II})}$, je nachdem, ob die Stützstellenanordnung

$$\begin{aligned} &\text{(I)} \quad x_0, x_{-1}, x_1, x_{-2}, x_2, \ldots \\ &\text{oder} \\ &\text{(II)} \quad x_0, x_1, x_{-1}, x_2, x_{-2}, \ldots \end{aligned} \tag{48}$$

zugrunde liegt. Die dazu nach der Newtonschen Formel gebildeten Polynome sind

$$\text{(I)} \quad \begin{cases} G_f^{(\mathrm{I})}(x) = f_0 + (x-x_0)\,[x_0, x_{-1}] + (x-x_0)(x-x_{-1})\,[x_0, x_{-1}, x_1] + \cdots \\ \qquad + (x-x_0)(x-x_{-1})\cdots(x-x_{-m})\,[x_0, x_{-1}, \ldots, x_{-m}, x_m] \\ \qquad + (x-x_0)(x-x_{-1})\cdots(x-x_{-m})(x-x_m)\,[x_0, x_{-1}, \ldots, x_{-m}, x_m, x_{-m-1}], \end{cases} \tag{49}$$

$$\text{(II)} \quad \begin{cases} G_f^{(\mathrm{II})}(x) = f_0 + (x-x_0)\,[x_0, x_1] + (x-x_0)(x-x_1)\,[x_0, x_1, x_{-1}] + \cdots \\ \qquad + (x-x_0)(x-x_1)\cdots(x-x_m)\,[x_0, x_1, \ldots, x_m, x_{-m}] \\ \qquad + (x-x_0)(x-x_1)\cdots(x-x_m)(x-x_{-m})\,[x_0, x_1, \ldots, x_m, x_{-m}, x_{m+1}]. \end{cases}$$

Bricht man mit dem vorletzten Glied ab, so werden $2m+1$ der in (48) (I) bzw. (48) (II) angegebenen Stützstellen benutzt, und zwar insgesamt die gleichen, so daß

$G_f^{(\mathrm{I})}$ und $G_f^{(\mathrm{II})}$ verschiedene Formen ein und desselben Interpolationspolynoms sind. Bei Mitführung des letzten Gliedes erfaßt man $2m+2$ dieser Stützstellen, wobei sich die Systeme (I) und (II) in den Punkten x_{-m-1} und x_{m+1} unterscheiden; $G_f^{(\mathrm{I})}$ und $G_f^{(\mathrm{II})}$ stellen dann im allgemeinen verschiedene Interpolationspolynome dar. Die mit $2m$ bzw. $2m+1$ zu indizierenden Restglieder sind, falls f genügend oft differenzierbar ist,

$$
\text{(I)}\quad\begin{cases} R_{2m}^{(\mathrm{I})}(x) = \dfrac{f^{(2m+1)}(\xi_x)}{(2m+1)!}(x-x_0)(x-x_{-1})\cdots(x-x_{-m})(x-x_m), \\ R_{2m+1}^{(\mathrm{I})}(x) = \dfrac{f^{(2m+2)}(\xi_x)}{(2m+2)!}(x-x_0)(x-x_{-1})\cdots(x-x_{-m})(x-x_m)(x-x_{-m-1}), \end{cases}
$$

$$
\text{(II)}\quad\begin{cases} R_{2m}^{(\mathrm{II})}(x) = \dfrac{f^{(2m+1)}(\xi_x)}{(2m+1)!}(x-x_0)(x-x_1)\cdots(x-x_m)(x-x_{-m}), \\ R_{2m+1}^{(\mathrm{II})}(x) = \dfrac{f^{(2m+2)}(\xi_x)}{(2m+2)!}(x-x_0)(x-x_1)\cdots(x-x_m)(x-x_{-m})(x-x_{m+1}). \end{cases}
\tag{50}
$$

Auf Grund von (36) lassen sich die Steigungen in (49) wieder durch gewisse Differenzen Δ_i^k ausdrücken, die im Schema der Tabelle 4.7 für $G_f^{(\mathrm{I})}$ und $G_f^{(\mathrm{II})}$ durch einen unterbrochenen bzw. ausgezogenen Weg herausgehoben wurden. Man findet

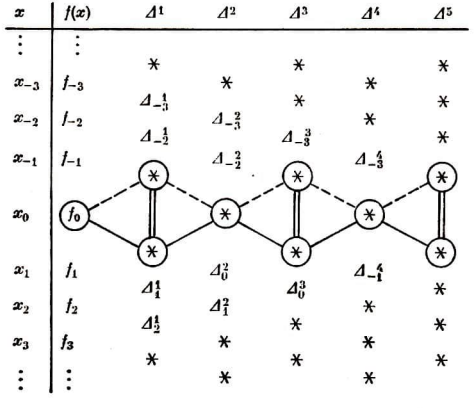


Tabelle 4.7

$$\text{(I)}\quad \begin{cases} G_f^{(\mathrm{I})}(x) = f_0 + \dfrac{1}{1!\,h}\,\Delta^1_{-1}\,(x-x_0) + \dfrac{1}{2!\,h^2}\,\Delta^2_{-1}(x-x_0)\,(x-x_{-1}) + \cdots \\ \qquad + \dfrac{1}{(2m)!\,h^{2m}}\,\Delta^{2m}_{-m}(x-x_0)\,(x-x_{-1})\cdots(x-x_{-m}) \\ \qquad + \dfrac{1}{(2m+1)!\,h^{2m+1}}\,\Delta^{2m+1}_{-m-1}\,(x-x_0)\,(x-x_{-1})\cdots(x-x_{-m})\,(x-x_m), \end{cases}$$

(51)

$$\text{(II)}\quad \begin{cases} G_f^{(\mathrm{II})}(x) = f_0 + \dfrac{1}{1!\,h}\,\Delta^1_{0}(x-x_0) + \dfrac{1}{2!\,h^2}\,\Delta^2_{-1}\,(x-x_0)\,(x-x_1) + \cdots \\ \qquad + \dfrac{1}{(2m)!\,h^{2m}}\,\Delta^{2m}_{-m}(x-x_0)\,(x-x_1)\cdots(x-x_m) \\ \qquad + \dfrac{1}{(2m+1)!\,h^{2m+1}}\,\Delta^{2m+1}_{-m}(x-x_0)\,(x-x_1)\cdots(x-x_m)\,(x-x_{-m}). \end{cases}$$

Bei der linearen Substitution $x = x_0 + hs$ gehen die Stützstellen (48) in

(I) $\quad 0, -1, 1, -2, 2, \ldots$

bzw. (52)

(II) $\quad 0, 1, -1, 2, -2, \ldots$

über, und man erhält aus (51) und (50)

$$\text{(I)}\quad \begin{cases} G_f^{(\mathrm{I})}(s) = f_0 + \dfrac{1}{1!}\,\Delta^1_{-1}\,s + \dfrac{1}{2!}\,\Delta^2_{-1}\,s(s+1) + \cdots \\ \qquad + \dfrac{1}{(2m)!}\,\Delta^{2m}_{-m}\,s(s+1)\cdots(s+m) \\ \qquad + \dfrac{1}{(2m+1)!}\,\Delta^{2m+1}_{-m-1}\,s(s+1)\cdots(s+m)\,(s-m), \end{cases}$$

(53)

$$\text{(II)}\quad \begin{cases} G_f^{(\mathrm{II})}(s) = f_0 + \dfrac{1}{1!}\,\Delta^1_{0}\,s + \dfrac{1}{2!}\,\Delta^2_{-1}\,s(s-1) + \cdots \\ \qquad + \dfrac{1}{(2m)!}\,\Delta^{2m}_{-m}\,s(s-1)\cdots(s-m) \\ \qquad + \dfrac{1}{(2m+1)!}\,\Delta^{2m+1}_{-m}\,s(s-1)\cdots(s-m)\,(s+m); \end{cases}$$

$$\text{(I)}\quad \begin{cases} \hat{R}^{(\mathrm{I})}_{2m}(s) = \dfrac{h^{2m+1}}{(2m+1)!} f^{(2m+1)}(\xi_x)\, s(s^2-1)\,(s^2-4)\cdots(s^2-m^2), \\ \hat{R}^{(\mathrm{I})}_{2m+1}(s) = \dfrac{h^{2m+1}}{(2m+2)!} f^{(2m+2)}(\xi_x)\, s(s^2-1)\,(s^2-4)\cdots(s^2-m^2)\,(s+m+1), \end{cases} \tag{54}$$

$$\text{(II)}\quad \begin{cases} \hat{R}^{(\mathrm{II})}_{2m}(s) = \dfrac{h^{2m+1}}{(2m+1)!} f^{(2m+1)}(\xi_x)\, s(s^2-1)\,(s^2-4)\cdots(s^2-m^2), \\ \hat{R}^{(\mathrm{II})}_{2m+1}(s) = \dfrac{h^{2m+2}}{(2m+2)!} f^{(2m+2)}(\xi_x)\, s(s^2-1)\,(s^2-4)\cdots(s^2-m^2)\,(s-m-1). \end{cases}$$

Die Gaußschen Formeln weisen bezüglich der Stelle x_0 noch eine gewisse Unsymmetrie auf, die sich durch Mittelbildung beseitigen läßt. Auf diese Weise erhält man die *Formel von Stirling*, welche nur bei Verwendung einer ungeraden Anzahl der Argumente (48) Interpolationspolynom zu eben diesen Stützstellen und ansonsten das arithmetische Mittel von zwei verschiedenen Interpolationspolynomen ist. Bezeichnen wir dieses in jedem Fall mit S, so gilt in der Variablen s

$$\begin{aligned} S_f(s) :=& \frac{\hat{G}^{(\mathrm{I})}_f(s)+\hat{G}^{(\mathrm{II})}_f(s)}{2} \\ =& f_0 + \frac{1}{1!}\frac{\Delta^1_{-1}+\Delta^1_0}{2}\, s + \frac{1}{2!}\Delta^2_{-1}\, s^2 + \frac{1}{3!}\frac{\Delta^3_{-2}+\Delta^3_{-1}}{2}\, s(s^2-1) \\ &+ \cdots + \frac{1}{(2m)!}\Delta^{2m}_{-m}\, s^2(s^2-1)\,(s^2-4)\cdots(s^2-(m-1)^2) \\ &+ \frac{1}{(2m+1)!}\frac{\Delta^{2m+1}_{-m-1}+\Delta^{2m+1}_{-m}}{2}\, s(s^2-1)\,(s^2-4)\cdots(s^2-m^2). \end{aligned} \tag{55}$$

Man beachte, daß sich die Mittelbildung abwechselnd auf die in den Gaußschen Formeln auftretenden Differenzen und Linearfaktorenprodukte erstreckt; die zu mittelnden Δ^k_i sind in Tabelle 4.7 durch zwei Verbindungsstriche herausgehoben.

Im Fall, daß eine ungerade Anzahl der ersten Stützstellen (48) benutzt wird, ist das Restglied der Stirlingschen Formel (55) das von $\hat{G}^{(\mathrm{I})}_f$ oder $\hat{G}^{(\mathrm{II})}_f$. Ansonsten wird es aus diesen durch Mittelbildung gewonnen:

$$\begin{aligned} \hat{R}^{(S)}_{2m+1}(s) = & \frac{h^{2m+2}}{2(2m+2)!}\, s(s^2-1)\,(s^2-4)\cdots(s^2-m^2) \\ & \times[(s-m-1)\, f^{(2m+2)}(\xi_1) + (s+m+1)\, f^{(2m+2)}(\xi_2)]. \end{aligned} \tag{56}$$

Aus Symmetriegründen eignet sich die Stirlingsche Formel zur approximativen Berechnung von Funktionswerten in der Nachbarschaft von x_0 und zur Herleitung von Näherungsverfahren für damit auszuführende Operationen, wie etwa die Differentiation bei x_0. Für Interpolationsaufgaben, die sich auf das ganze Intervall zwischen zwei aufeinanderfolgenden Stützstellen x_i, x_{i+1} beziehen, verwendet man

die *Besselsche Formel*, die mit Differenzen gebildet wird, welche im Schema symmetrisch zur Mittellinie der Positionen von x_i und x_{i+1} angeordnet sind. Man gewinnt diese als arithmetisches Mittel von $G_f^{(\mathrm{I})}$, begonnen mit x_{i+1}, und $G_f^{(\mathrm{II})}$, begonnen mit x_i. Wir notieren die Formel für $i=0$ und können dann $G_f^{(\mathrm{II})}$ und das zugehörige Restglied wie in (51) (II) und (50) (II) angegeben benutzen. Bezüglich $G_f^{(\mathrm{I})}$ muß in (51) (I) und (50) (I) eine Erhöhung der Indizes um 1 erfolgen. Mit der Substitution $x=x_0+hs$ resultiert dann für das transformierte Besselpolynom B

$$\begin{aligned} B_f(s) = &\frac{f_0+f_1}{2} + \frac{1}{1!}\Delta_0^1\left(s-\frac{1}{2}\right) + \frac{1}{2!}\frac{\Delta_{-1}^2+\Delta_0^2}{2}s(s-1)+\cdots \\ &+\frac{1}{(2m)!}\frac{\Delta_{-m}^{2m}+\Delta_{-m+1}^{2m}}{2}s(s-1)\,(s+1)\cdots(s-m) \\ &+\frac{1}{(2m+1)!}\Delta_{-m}^{2m+1}\,s(s-1)\,(s+1)\cdots(s-(m-1))\,(s+(m-1)) \\ &\times(s-m)\left(s-\frac{1}{2}\right). \end{aligned} \tag{57}$$

Bezüglich der Mittelbildung und der beteiligten Stützstellen liegt bei (57) die zur Stirlingschen Formel komplementäre Situation vor: Benutzt man (57) bis zu dem letzten angegebenen Glied, so sind die Stützstellen

$$x_0, x_1, x_{-1}, \ldots, x_m, x_{-m}, x_{m+1}$$

beteiligt, und die Besselsche Formel stellt das mit diesen gebildete Interpolationspolynom dar; das Restglied kann in der Form (54) (II) (zweite Formel) angegeben werden. Bricht man mit dem vorletzten Glied ab, so treten in $G_f^{(\mathrm{I})}$ die Stützstellen

$$x_0, x_{\pm1}, x_{\pm2}, \ldots, x_{\pm(m-1)}, x_m, x_{m+1}$$

und in $G_f^{(\mathrm{II})}$ die Stützstellen

$$x_0, x_{\pm1}, x_{\pm2}, \ldots, x_{\pm(m-1)}, x_{\pm m}$$

auf. (57) ist dann das arithmetische Mittel zweier verschiedener Interpolationsformeln, und für das Restglied findet man durch entsprechende Mittelung

$$\begin{aligned} \hat{R}_{2m}^{(B)}(s) = &\frac{h^{2m+1}}{2(2m+1)!}\,s(s^2-1)\,(s^2-4)\cdots(s^2-(m-1)^2)\,(s-m) \\ &\times[(s-m-1)\,f^{(2m+1)}(\xi_1)+(s+m)\,f^{(2m+1)}(\xi_2)]. \end{aligned} \tag{58}$$

Im Schema der Tabelle 4.8 sind die in der Besselschen Formel auftretenden Differenzen kenntlich gemacht; Doppelstriche verbinden die zu mittelnden Differenzen.

Häufig unterwirft man (57) noch der Transformation

$$s=t+\frac{1}{2}; \tag{59}$$

$t=0$ entspricht dann dem Mittelpunkt des Intervalls $[x_0, x_1]$. In der Variablen t stellen die Linearfaktorenprodukte der Besselschen Formel abwechselnd gerade und ungerade Funktionen dar.

Aufgabe 6. Man führe in der Besselschen Formel die Variablentransformation (59) durch.

Wir weisen noch auf weitere formale Sachverhalte bei den Interpolationspolynomen mit äquidistanten Stützstellen hin: Die in einem Formelterm auftretenden Linearfaktoren und Fakultäten werden gelegentlich zu Binomialkoeffizienten zusammengefaßt. Beispielsweise könnte man (53) in der Form

$$G_f^{(1)}(s) = f_0 + \Delta^1_{-1}\binom{s}{1} + \Delta^2_{-1}\binom{s+1}{2} + \Delta^3_{-2}\binom{s+1}{3} + \cdots$$
$$+ \Delta^{2m}_{-m}\binom{s+m}{2m} + \Delta^{2m+1}_{-m-1}\binom{s+m}{2m+1}$$

schreiben.

An Stelle der vorderen Differenzen (35) werden auch sogenannte *hintere* und *zentrale Differenzen* benutzt, die miteinander austauschbar, aber in einem bestimmten Zusammenhang mehr oder weniger signifikant sind. Sie werden analog zu (35)

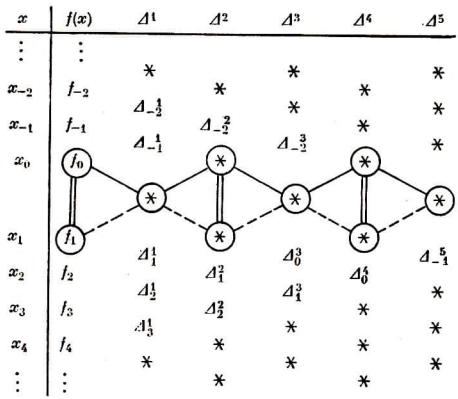


Tabelle 4.8

mit Hilfe der Operatoren

$$\nabla f(x) := f(x) - f(x-h)$$

bzw. (60)

$$\delta f(x) := f\left(x+\frac{1}{2}h\right) - f\left(x-\frac{1}{2}h\right)$$

eingeführt.

Bei der Bildung des Differenzenschemas für die Δ_i^k treten die gleichen numerischen Effekte auf, die wir am Schluß von 4.2.4. bezüglich der Steigungen beschrieben haben. Zur weiteren Verdeutlichung wollen wir den Fall betrachten, daß die bei den x_i gegebenen Funktionswerte bis auf einen exakt sind; der absolute Fehler des Ausnahmewertes sei ε. Dem mit durchweg exakten Funktionswerten gebildeten Differenzenschema wird sich dann das Schema der Tabelle 4.9 überlagern. Die

y	Δ^1	Δ^2	Δ^3	Δ^4	...
⋮					
0					
	0		0		
0		0		ε	
	0		ε		
0		ε		$-4\,\varepsilon$	
	ε		$-3\,\varepsilon$		
ε		$-2\,\varepsilon$		$6\,\varepsilon$	
	$-\varepsilon$		$3\,\varepsilon$		
0		ε		$-4\,\varepsilon$	
	0		$-\varepsilon$		
0		0		ε	
	0		0		
0					
⋮					

Tabelle 4.9

Gesetzmäßigkeit des Aufschaukelns eines solchen Fehlers kann man dazu benutzen, in einer Tafel Fehler aufzuspüren und zu korrigieren, deren Größenordnung diejenige der durch Rundung auf die angegebenen gültigen Ziffern bedingten übersteigt.

Aufgabe 7. Die folgende Logarithmentabelle enthält einen Tafelfehler. Dieser ist mit Hilfe des Differenzenschemas zu korrigieren.

x	$f(x)$	x	$f(x)$
40	1,60206	47	1,67210
41	1,61278	48	1,68124
42	1,62325	49	1,69020
43	1,63347	50	1,69897
44	1,64345	51	1,70757
45	1,65312	52	1,71600
46	1,66276	53	1,72428

4.3. Numerische Differentiation und Integration

Die Differentiation bzw. Integration einer Funktion f nach den formalen Regeln der Infinitesimalrechnung ist nur beschränkt möglich und wird jedenfalls dann in Frage gestellt, wenn diese Operationen automatisch von einem Computer auszuführen sind. Würde man sich zum Beispiel bemühen, die Potenzreihenentwicklung der Funktion $y = \arctan x$ in der MacLaurinschen Form dadurch zu gewinnen, daß man deren Ableitungen bei Null bestimmt, so erhielte man sehr bald komplizierte und in ihrer Struktur schwer übersehbare Ausdrücke, und der Versuch, mit dem Hauptsatz der Differential- und Integralrechnung praktisch Integrale auszurechnen, scheitert oft an der Unmöglichkeit, für den Integranden eine Stammfunktion in „geschlossener" Form anzugeben (vgl. [48], Bd. 3, S. 29). Ein Beispiel dafür ist $\int_a^b e^{-x^2}\,dx$. Von vornherein ist nach numerischen und im allgemeinen approximativen Verfahren zu suchen, wenn von f nur endlich viele Werte an diskreten Argumentstellen x_i bekannt sind. Eine Methode zur näherungsweisen Berechnung von Ableitungs- und Integralwerten beruht auf der Interpolation mit ganzrationalen Funktionen:

Man approximiert f durch ein Interpolationspolynom P und benutzt Ableitungs- bzw. Integralwerte von P als Näherungen für die entsprechenden mit f zu bildenden Größen.

Die numerische Integration von Funktionen einer Veränderlichen wird auch als *Quadratur* bezeichnet. Offensichtlich ist die als Leitprinzip unserer Betrachtungen formulierte Verfahrensweise bei der Differentiation problematisch, da eine kleine Abweichung der Werte von f und P nicht notwendig auch eine solche für f' und P' zur Folge hat.

Je nachdem, ob man von der Lagrangeschen oder der Newtonschen Form des Interpolationspolynoms ausgeht, erhält man Formeln, welche unmittelbar die Werte von f an den Stützstellen oder gewisse daraus zu bildende Differenzen enthalten.

Eine im Verhältnis zum Aufwand beträchtliche Steigerung an Genauigkeit wird erzielt, wenn man an Stelle von Polynomen Splinefunktionen zur Interpolation benutzt (vgl. MfL Bd. 10).

4.3.1. Numerische Differentiation

In jedem Fall wollen wir annehmen, daß die zu differenzierende Funktion f an endlich vielen äquidistanten Argumentstellen der Schrittweite h gegeben ist. Der Verfahrensfehler bei der Berechnung von Ableitungen nach der in der Einleitung formulierten Methode ergibt sich aus dem Interpolationsfehler gemäß

$$RD_n(x) := f'(x) - P'(x) \tag{1}$$

und ist hinsichtlich des oben ausgedrückten Vorbehalts zu diskutieren. Der Index n deutet auf die benutzten $n+1$ Stützstellen $x_0<x_1<\cdots<x_n$ hin. Bei der angenäherten Berechnung höherer Ableitungen hat man eine (1) entsprechende Fehlerformel zu betrachten.

Wir benutzen zunächst das Polynom von LAGRANGE und erhalten mit den Bezeichnungen von 4.2.1. ($x_0<x_1<\cdots<x_n$)

$$y'(x)=f'(x)\approx\sum_{i=0}^{n}y_i l_i'(x).$$

Führt man gemäß $x=x_0+hs$ die Variable s ein, so ergibt sich wegen $\frac{d}{dx}=\frac{1}{h}\frac{d}{ds}$

$$y'(x)=f'(x)\approx\frac{1}{h}\sum_{i=0}^{n}\frac{(-1)^{n-i}}{i!\,(n-i)!}y_i\frac{d}{ds}\frac{\bar{\omega}_n(s)}{s-i}, \tag{2}$$

wobei

$$\bar{\omega}_n(s)=s(s-1)\,(s-2)\cdots(s-n). \tag{3}$$

Für das Restglied (1) resultiert bei hinreichenden Differentiationsannahmen über f aus 4.2.4.(29)

$$RD_n(x)=\frac{1}{(n+1)!}\left[\omega_n'(x)f^{(n+1)}(\xi_x)+\omega_n(x)\frac{d}{dx}f^{(n+1)}(\xi_x)\right]. \tag{4}$$

An einer Stützstelle x_j ist

$$\omega_n'(x_j)=(x_j-x_0)\cdots(x_j-x_{j-1})\,(x_j-x_{j+1})\cdots(x_j-x_n).$$

Setzt man voraus, daß $\frac{d}{dx}f^{(n+1)}(\xi_x)$ beschränkt ist, so folgt damit aus (2) und (4)

$$y'(x_j)=\frac{1}{h}\sum_{i=0}^{n}\frac{(-1)^{n-i}}{i!\,(n-i)!}y_i\left[\frac{d}{ds}\frac{\bar{\omega}_n(s)}{s-i}\right]_{s=j}$$
$$+\frac{(-1)^{n-j}}{(n+1)!}h^n j!\,(n-j)!\,f^{(n+1)}(\xi)\quad(x_0\leqq\xi\leqq x_n). \tag{5}$$

Nach (5) berechnen wir beispielsweise die Ableitungen für drei Stützstellen ($n=2$); die sich durch Vernachlässigung des Restgliedes für $y'(x_j)$ ergebenden Näherungswerte seien mit y_j' bezeichnet:

$$\frac{d}{ds}\frac{\bar{\omega}_2(s)}{s}=2s-3,\quad\frac{d}{ds}\frac{\bar{\omega}_2(s)}{s-1}=2(s-1),\quad\frac{d}{ds}\frac{\bar{\omega}_2(s)}{s-2}=2s-1;$$

$$y_0'=\frac{1}{2h}(-3y_0+4y_1-y_2),$$

$$y_1'=\frac{1}{2h}(-y_0+y_2), \tag{6}$$

$$y_2'=\frac{1}{2h}(y_0-4y_1+3y_2);$$

die verfahrensbedingten Fehler $r_j := RD_2(x_j)$ sind

$$r_0 = \frac{1}{3}h^2y'''(\xi_0),$$
$$r_1 = -\frac{1}{6}h^2y'''(\xi_1), \tag{7}$$
$$r_2 = \frac{1}{3}h^2y'''(\xi_2).$$

Wegen ihrer Einfachheit (Bildung des „übersprungenen Differenzenquotienten") und der vergleichsweise höheren Genauigkeit wird man nach Möglichkeit die mittlere Formel (6) benutzen. Entsprechende Ausdrücke für $n = 3, 4, 5, 6$ und die zweiten Ableitungen findet man in [11], 3.2.3.

Formeln zur näherungsweisen Berechnung von Ableitungen, die Differenzen benutzen, leiten wir aus dem Stirlingschen Interpolationspolynom her, auf dessen Eignung dafür bereits hingewiesen wurde. Durch Differentiation von Formel (55) aus 4.2.5. gewinnt man für die erste und die zweite Ableitung von $y(x) = S_f(x)$ mit der Abkürzung

$$\Delta^{2m+1}_{m+\frac{1}{2}} := \frac{\Delta^{2m+1}_{-m-1} + \Delta^{2m+1}_{-m}}{2} \tag{8}$$

die Beziehungen

$$\begin{aligned} y'(x) &= \frac{1}{h}\frac{d}{ds}S_f(s) \\ &= \frac{1}{h}\Big[\Delta^1_{-\frac{1}{2}} + \Delta^2_{-1}s + \frac{1}{6}\Delta^3_{-\frac{3}{2}}(3s^2-1) + \frac{1}{12}\Delta^4_{-2}(2s^3-s) \\ &\quad + \frac{1}{120}\Delta^5_{-\frac{5}{2}}(5s^4-15s^2+4) + \frac{1}{360}\Delta^6_{-3}(3s^5-10s^3+4s) + \cdots\Big]; \\ y''(x) &= \frac{1}{h^2}\frac{d^2}{ds^2}S_f(s) \\ &= \frac{1}{h^2}\Big[\Delta^2_{-1} + \Delta^3_{-\frac{3}{2}}s + \frac{1}{12}\Delta^4_{-2}(6s^2-1) + \frac{1}{12}\Delta^5_{-\frac{5}{2}}(2s^3-3s) \\ &\quad + \frac{1}{360}\Delta^6_{-3}(15s^4-30s^2+4) + \cdots\Big]. \end{aligned} \tag{9}$$

Näherungswerte für die erste und zweite Ableitung der interpolierten Funktion an der Stelle x_0 ergeben sich, indem man $s = 0$ setzt:

$$\begin{aligned} y'(x_0) &= \frac{1}{h}\Big[\Delta^1_{-\frac{1}{2}} - \frac{1}{6}\Delta^3_{-\frac{3}{2}} + \frac{1}{30}\Delta^5_{-\frac{5}{2}} - \cdots\Big], \\ y''(x_0) &= \frac{1}{h^2}\Big[\Delta^2_{-1} - \frac{1}{12}\Delta^4_{-2} + \frac{1}{90}\Delta^6_{-3} - \cdots\Big]. \end{aligned} \tag{10}$$

Wir erläutern den Gebrauch der Formel (9) an einem Beispiel, das auch zeigen soll, wie man zu Abschätzungen der in den Restgliedformeln auftretenden Ableitungen gelangt. Für die in dem Differenzenschema Tabelle 4.10 tabulierte Funktion soll zwischen den Argumenten 5,52 und 5,56 eine Untertafelung mit der Schrittweite $h = 0{,}0025$ vorgenommen werden. Neben den auf fünf Stellen nach dem Komma zu bestimmenden Funktionswerten sind die Ableitungswerte mit drei gültigen Ziffern nach dem Komma zu ermitteln.

x	$f(x)$	Δ^1	Δ^2	Δ^3	Δ^4	Δ^5
5,50	−86,31857					
		0,01253				
5,51	−86,30604		0,01768			
		0,03021		0,00037		
5,52	−86,27583		0,01805		−0,00003	
		0,04826		0,00034		0,00006
5,53	−86,22757		0,01839		0,00003	
		0,06665		0,00037		−0,00005
5,54	−86,16092		0,01876		−0,00002	
		0,08541		0,00035		0,00004
5,55	−86,07551		0,01911		0,00002	
		0,10452		0,00037		−0,00003
5,56	−85,97099		0,01948		−0,00001	
		0,12400		0,00036		
5,57	−85,84699		0,01984			
		0,14384				
5,58	−85,70315					

Tabelle 4.10

Für die Untertafelung zwischen zwei aufeinanderfolgenden der im Schema angegebenen Argumentstellen benutzen wir die Besselsche Interpolationsformel und untersuchen den Verfahrensfehler für das Polynom dritten Grades. Dieser ist gemäß 4.2.5.(54) durch $\hat{R}^{(\mathrm{II})}_{2m+1}(s)$ für $m = 1$ gegeben und erfordert die Abschätzung der vierten Ableitung von f. Da die Funktion nur an diskreten Stellen gegeben ist, muß dabei auf die (sich noch regelmäßig verhaltenden) Differenzen vierter Ordnung zurückgegriffen werden. Unter der Annahme, daß die betrachteten Ableitungen existieren, hat man ($x = x_0 + hs$, $f(x) = \hat{f}(s)$)

$$h^4 f^{(\mathrm{IV})}(\xi_x) = \frac{d^4}{ds^4}\hat{f}(\sigma),$$

wobei σ ein Punkt des durch die beteiligten Interpolationsstellen $s = 0, 1, -1, 2$ bestimmten Intervalls ist. Ersetzen wir die erste Ableitung in der Mitte des von zwei Stützstellen berandeten Intervalls näherungsweise durch den mit den Endordinaten gebildeten Differenzenquotienten, so ist dort

$$\frac{d\hat{f}}{ds} = h\frac{df}{dx} \approx h\frac{\Delta f}{h} = \Delta f.$$

Analog kann man die Werte in der Δ^2-Spalte als Näherungen für $\frac{d^2\hat{f}}{ds^2}$ in den Stützstellen interpretieren und so fort. In diesem Sinne approximieren wir die Werte von $\frac{d^4\hat{f}}{ds^4}$ an den Stützstellen durch die in der entsprechenden Zeile des Schemas stehenden Differenzen vierter Ordnung und können damit den Betrag von $\frac{d^4\hat{f}}{ds^4}$ über dem s-Intervall $[-1, 2]$ durch $3 \cdot 10^{-5}$ abschätzen. Da auf $[-1, 2]$

$$|s(s^2-1)(s-2)| \leqq 1$$

ist (vgl. Abb. 4.13), ergibt sich auf Grund von Formel (54) (II) aus 4.2.5. als Ab-

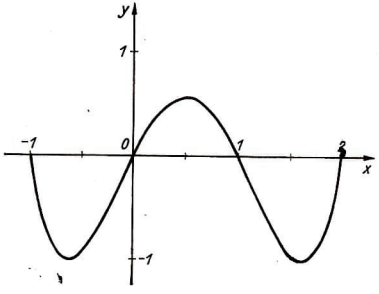


Abb. 4.13

schätzung des Verfahrensfehlers bei der Interpolation

$$|\hat{R}_3^{(\mathrm{II})}| \leqq \frac{3 \cdot 10^{-5}}{24} = 0{,}125 \cdot 10^{-5},$$

was der geforderten Genauigkeit entspricht. Die zwischen aufeinanderfolgenden Stützstellenabszissen der Tabelle 4.10 zur Untertafelung für $s = 0{,}25$; 0,50; 0,75 auszuwertenden Besselpolynome sind

$$\hat{B}_{52;53}(s) := -\left[86{,}27583 - 0{,}04826s - 0{,}00911s(s-1) - 0{,}000056s(s-1)\left(s-\frac{1}{2}\right)\right],$$

$$\hat{B}_{53;54}(s) := -\left[86{,}22757 - 0{,}06665s - 0{,}009288s(s-1) - 0{,}000062s(s-1)\left(s-\frac{1}{2}\right)\right],$$

$$\hat{B}_{54;55}(s) := -\left[86{,}16092 - 0{,}08541s - 0{,}009468s(s-1) - 0{,}000058s(s-1)\left(s-\frac{1}{2}\right)\right],$$

$$\hat{B}_{55;56}(s) := -\left[86{,}07551 - 0{,}10452s - 0{,}009648s(s-1) - 0{,}000062s(s-1)\left(s-\frac{1}{2}\right)\right].$$

Die Ergebnisse sind in der Tabelle 4.11 zusammen mit den ersten und zweiten Differenzen der untertafelten Funktion verzeichnet.

x	$f(x)$	Δ^1	Δ^2	$f'(x)$
5,5200	−86,27583			
		0,01036		
	−86,26547		0,00113	4,370
		0,01149		
50	−86,25398		0,00114	4,824
		0,01263		
	−86,24135		0,00116	5,282
		0,01378		
5,5300	−86,22757		0,00114	5,740
		0,01492		
	−86,21265		0,00116	6,200
		0,01608		
50	−86,19657		0,00116	6,664
		0,01724		
	−86,17933		0,00117	7,130
		0,01841		
5,5400	−86,16092		0,00117	7,598
		0,01958		
	−86,14234		0,00118	8,068
		0,02076		
50	−86,12058		0,00118	8,540
		0,02194		
	−86,09864		0,00119	9,014
		0,02313		
5,5500	−86,07551		0,00119	9,490
		0,02432		
	−86,05119		0,00121	9,970
		0,02553		
50	−86,02566		0,00120	10,452
		0,02673		
	−85,99893		0,00121	10,934
		0,02794		
5,5600	−85,97099			

Tabelle 4.11

Auf Grund der Tabelle 4.11 bestimmen wir nun mit Hilfe der mittleren Formel (6) die gesuchten Ableitungswerte. Für den verfahrensbedingten Fehler folgt nach (7), wenn wieder der Betrag von

$$h^2 f''(x) = \hat{f}''(s) \quad \text{(jetzt ist } h = 0{,}0025\text{)}$$

durch eine obere Schranke für die Beträge der zweiten Differenzen der Tabelle 4.11 majorisiert wird,

$$|r_1| \leqq \frac{0{,}0012}{6} = 0{,}0002.$$

Das entspricht der geforderten Genauigkeit. Benutzen wir die bei den Interpolationsformeln von Gauss, Bessel und Stirling eingeführte Indizierung der Abszissen, wobei 0 der jeweiligen Differentiationsstelle zugeordnet sei, so ist (6)

in der Form

$$y_0' = \frac{1}{2h}(y_1 - y_{-1}) = \frac{\Delta_0^1 + \Delta_{-1}^1}{2h} = 2(\Delta_0^1 + \Delta_{-1}^1) \cdot 10^2$$

zu notieren. Das stimmt mit der Näherung überein, die man aus (10) durch Abbruch nach dem ersten Glied gewinnt. Die in Tabelle 4.11 angegebenen Ableitungswerte sind danach berechnet worden.

4.3.2. Numerische Quadratur

Wir betrachten das Problem der approximativen Berechnung des Riemannschen Integrals $\int_a^b f(x)\,dx$ einer auf dem Intervall $[a, b]$ stetigen Funktion f. Dafür werden wir Näherungsausdrücke der Form

$$\sum_{i=0}^{n} A_i f(x_i) \tag{11}$$

herleiten, in denen die x_i $n+1$ Punkte des Intervalls $[a, b]$ und die A_i gewisse von f unabhängige Konstanten sind. Je nachdem, ob beide Integrationsgrenzen zu den Stützstellen x_i gehören oder nicht, heißt (11) eine *geschlossene* bzw. *offene* Quadraturformel. Von dieser werden wir jedenfalls fordern, daß die Funktion $f(x) = 1$ exakt integriert wird, was für die A_i

$$\int_a^b 1\,dx = b - a = \sum_{i=0}^{n} A_i \tag{12}$$

zur Konsequenz hat. Ansonsten ist die Untersuchung des Verfahrensfehlers

$$RI := \int_a^b f(x)\,dx - \sum_{i=0}^{n} A_i f(x_i) \tag{13}$$

wesentliches Element der Theorie. Auf Grund von (12) ergibt sich für den durch

$$f_m := \frac{1}{b-a}\int_a^b f(x)\,dx \tag{14}$$

definierten *mittleren Funktionswert* von f auf $[a, b]$

$$f_m \approx \frac{\sum_{i=0}^{n} A_i f(x_i)}{\sum_{i=0}^{n} A_i}, \tag{15}$$

d. h., f_m ist näherungsweise das mit den A_i gewichtete arithmetische Mittel der Funktionswerte $f(x_i)$. Wegen dieses Sachverhaltes nennt man (11) gelegentlich eine *Mittelwertquadraturformel* und bezeichnet die A_i als deren *Gewichte*. *Typen von Quadraturformeln* werden dadurch bestimmt, daß man für alle oder gewisse n ein Verfahren zur Konstruktion von (11) angibt. Beispielsweise könnte (11) eine spezielle *Interpolationsquadraturformel* sein, die durch Integration des mit den

Punkten $(x_i, f(x_i))$ gebildeten Lagrangeschen Polynoms erzeugt wird. Eine weitere Differenzierung ergibt sich dann aus Festlegungen bezüglich der Lage der x_i im Integrationsintervall. So erhielte man etwa die sogenannten Formeln von NEWTON-COTES bei äquidistanter Verteilung der x_i über das Intervall $[\![a, b]\!]$ unter Einbeziehung der Integrationsgrenzen. Über die Stützstellen und Gewichte vieler Typen von Quadraturformeln gibt es umfangreiche Tafelwerke (vgl. etwa [44]). Um diese Werte tabulieren zu können, ist zuvor eine Normierung des Problems hinsichtlich des Integrationsintervalls erforderlich. Meist bezieht man sich dabei auf das Intervall $[\![0, 1]\!]$ oder $[\![-1, 1]\!]$. Wir betrachten das letzte und unterwerfen dazu die Integrationsvariable x der linearen Transformation

$$x = \frac{b-a}{2}\, t + \frac{a+b}{2}\,. \tag{16}$$

(16) führt das x-Intervall $[\![a, b]\!]$ in das t-Intervall $[\![-1, 1]\!]$ über, und bezüglich der durch

$$\hat{f}(t) := f(x) = f\left(\frac{b-a}{2}\, t + \frac{a+b}{2}\right) \tag{17}$$

definierten Funktion $\hat{f}$ gilt auf Grund der Substitutionsregel

$$\int_a^b f(x)\, dx = \frac{b-a}{2} \int_{-1}^{1} \hat{f}(t)\, dt. \tag{18}$$

Es bedeutet also keine wesentliche Einschränkung der Allgemeinheit, wenn man von vornherein den Integranden als auf dem Intervall $[\![-1, 1]\!]$ definiert annimmt. Im folgenden verfahren wir so und bezeichnen $t, \hat{f}$ wieder mit x bzw. f. Eine unter dieser Annahme abgeleitete Formel (11) geht auf Grund von (17) und (18) bezüglich eines Integrals $\int_a^b f(x)\, dx$ in die Näherungsquadraturformel

$$\int_a^b f(x)\, dx \approx \sum_{i=0}^{n} \frac{b-a}{2}\, A_i f\left(\frac{b-a}{2}\, x_i + \frac{a+b}{2}\right) \tag{19}$$

über, und der Verfahrensfehler ist das $\left(\frac{b-a}{2}\right)$-fache des für den normierten Fall bestimmten Wertes.

Wir betrachten genauer die mit Hilfe des Lagrangeschen Polynoms konstruierten Interpolationsquadraturformeln. x_i $(i = 0, 1, \ldots, n)$ seien $n+1$ paarweise verschiedene Stellen des Intervalls $[\![-1, 1]\!]$, und L bedeutet das bezüglich der Punkte $(x_i, y_i := f(x_i))$ gebildete Polynom (6) aus 4.2.1. Dieses stimmt für alle x mit dem entsprechenden Newtonschen Interpolationspolynom von f überein, so daß auf Grund von 4.2.3.(17) die Beziehung

$$\begin{aligned} \int_{-1}^{1} f(x)\, dx &= \int_{-1}^{1} L(x)\, dx + \int_{-1}^{1} R_n(x)\, dx \\ &= \sum_{i=0}^{n} y_i \int_{-1}^{1} l_i(x)\, dx + \int_{-1}^{1} R_n(x)\, dx \end{aligned} \tag{20}$$

gilt. Die Gewichte der Formel sind

$$A_i := \int_{-1}^{1} l_i(x)\,dx, \qquad i=0, 1, 2, \ldots, n, \tag{21}$$

und

$$RI := \int_{-1}^{1} R_n(x)\,dx \tag{22}$$

ist der Verfahrensfehler. Daß die Gewichte, wie eingangs gefordert, von der zu integrierenden Funktion unabhängig sind, folgt aus der entsprechenden Eigenschaft der Polynome l_i (vgl. die Bemerkung im Anschluß an den Beweis zu (7) aus 4.2.1.). Die mit (21) gebildeten Quadraturformeln (11) liefern den exakten Integralwert, wenn f ein Polynom maximal n-ten Grades ist. Dann verschwindet nämlich nach 4.2.3.(18), (21) der Interpolationsfehler R_n identisch und damit auch RI. Das trifft speziell auf die Potenzen x^k $(k=0, 1, \ldots, n)$ zu, so daß

$$\begin{aligned}
\int_{-1}^{1} 1\,dx &= 2 &&= A_0 + A_1 + \cdots + A_n,\\
\int_{-1}^{1} x\,dx &= 0 &&= x_0A_0 + x_1A_1 + \cdots + x_nA_n,\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
\int_{-1}^{1} x^{2m}\,dx &= \frac{2}{2m+1} &&= x_0^{2m}A_0 + x_1^{2m}A_1 + \cdots + x_n^{2m}A_n,\\
\int_{-1}^{1} x^{2m+1}\,dx &= 0 &&= x_0^{2m+1}A_0 + x_1^{2m+1}A_1 + \cdots + x_n^{2m+1}A_n,\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
\int_{-1}^{1} x^n\,dx &= \frac{1+(-1)^n}{n+1} &&= x_0^nA_0 + x_1^nA_1 + \cdots + x_n^nA_n.
\end{aligned} \tag{23}$$

gilt. (23) enthält (12) und stellt ein lineares Gleichungssystem dar, dem die Gewichte (21) genügen. Die Koeffizientendeterminante

$$\begin{vmatrix} 1 & 1 & \ldots & 1 \\ x_0 & x_1 & \ldots & x_n \\ \ldots & \ldots & \ldots & \ldots \\ x_0^n & x_1^n & \ldots & x_n^n \end{vmatrix}$$

ist eine Vandermondesche Determinante und als solche von Null verschieden. Die A_i lassen sich also nicht nur nach der Definition (21), sondern auch eindeutig als Lösungskomponente von (23) bestimmen. Innerhalb eines bestimmten Formeltyps sind die Bezeichnungen der Stützstellen, Gewichte und des Verfahrensfehlers durch einen Hinweis auf die Anzahl der Stützstellen zu unterscheiden. Ist diese $n+1$, so schreiben wir $x_i^{(n)}$, $A_i^{(n)}$ bzw. RI_n.

Bei den Formeln von Newton-Cotes wie auch in den meisten anderen Fällen liegen die $x_i^{(n)}$ im Intervall $[-1, 1]$ symmetrisch zum Nullpunkt. Es ist dann signi

fikant, die Stützstelle 0 mit x_0 und im übrigen Stützstellen gleichen Betrages paarweise mit x_{-i} bzw. x_i zu bezeichnen und entsprechend bei den Gewichten zu verfahren.

Wir betrachten einige Spezialfälle der Formeln von NEWTON-COTES und notieren für $n=1, 2, 3$ in den neuen Bezeichnungen die Stützstellen sowie die zugehörigen Gleichungssysteme (23):

$$
\begin{aligned}
n=1:\quad & x_{-1}^{(1)} = -1, \quad x_1^{(1)} = 1;\\
& 2 = A_{-1}^{(1)} + A_1^{(1)},\\
& 0 = -A_{-1}^{(1)} + A_1^{(1)};\\
& A_1^{(1)} = A_{-1}^{(1)} = 1;\\
n=2:\quad & x_{-1}^{(2)} = -1, \quad x_0^{(2)} = 0, \quad x_1^{(2)} = 1;\\
& 2 = A_{-1}^{(2)} + A_0^{(2)} + A_1^{(2)},\\
& 0 = -A_{-1}^{(2)} + A_1^{(2)},\\
& \frac{2}{3} = A_{-1}^{(2)} + A_1^{(2)};\\
& A_1^{(2)} = A_{-1}^{(2)} = \frac{1}{3}, \quad A_0^{(2)} = \frac{4}{3};\\
n=3:\quad & x_{-2}^{(3)} = -1, \quad x_{-1}^{(3)} = -\frac{1}{3}, \quad x_1^{(3)} = \frac{1}{3}, \quad x_2^{(3)} = 1;\\
& 2 = A_{-2}^{(3)} + A_{-1}^{(3)} + A_1^{(3)} + A_2^{(3)},\\
& 0 = -A_{-2}^{(3)} - \frac{1}{3} A_{-1}^{(3)} + \frac{1}{3} A_1^{(3)} + A_2^{(3)},\\
& \frac{2}{3} = A_{-2}^{(3)} + \frac{1}{9} A_{-1}^{(3)} + \frac{1}{9} A_1^{(3)} + A_2^{(3)},\\
& 0 = -A_{-2}^{(3)} - \frac{1}{27} A_{-1}^{(3)} + \frac{1}{27} A_1^{(3)} + A_2^{(3)};
\end{aligned}
$$

durch Subtraktion der ersten und dritten bzw. zweiten und vierten Gleichung gewinnt man

$$\frac{3}{2} = A_{-1}^{(3)} + A_1^{(3)}, \quad 0 = A_{-1}^{(3)} - A_1^{(3)},$$

also

$$A_1^{(3)} = A_{-1}^{(3)} = \frac{3}{4};$$

aus der zweiten Gleichung folgt dann

$$A_2^{(3)} = A_{-2}^{(3)}$$

und schließlich aus der ersten

$$A_2^{(3)} = A_{-2}^{(3)} = \frac{1}{4}.$$ [1])

Die (19) entsprechenden Quadraturformeln sind

$n=1$:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\,[f(a)+f(b)] \quad \text{(Trapezregel)},$$

$n=2$: (24)

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right] \quad \text{(Keplersche Faßregel)},$$

$n=3$:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{8}\left[f(a)+3f\left(\frac{2a+b}{3}\right)+3f\left(\frac{a+2b}{3}\right)+f(b)\right]$$

(Newtonsche Drei-Achtel-Regel).

Aufgabe 1. Man berechne einige Gewichte dieser Formeln auf Grund der Definition (21).

Bei der Untersuchung des Quadraturfehlers RI wollen wir uns auf die Keplersche Faßregel beschränken und betrachten zunächst den Spezialfall $a=-1$, $b=1$. Nach (13) ist

$$RI_2(-1,1) := \int_{-1}^{1} f(x)\,dx - \frac{1}{3}[f(-1)+4f(0)+f(1)]$$

und, wenn man die auf dem Intervall $-1 \leqq u \leqq 1$ definierte Funktion

$$r(u) := \int_{-u}^{u} f(x)\,dx - \frac{u}{3}[f(-u)+4f(0)+f(u)] \tag{25}$$

einführt,

$$RI_2(-1,1) = r(1). \tag{26}$$

Für das Folgende sei angenommen, daß f auf $[-1,1]$ eine stetige vierte Ableitung besitzt. Durch Differentiation gewinnt man aus (25)

$$\begin{aligned} r'(u) &= f(u)+f(-u)-\frac{1}{3}[f(-u)+4f(0)+f(u)] \\ &\quad +\frac{u}{3}[f'(-u)-f'(u)] \\ &= \frac{2}{3}f(u)+\frac{2}{3}f(-u)-\frac{4}{3}f(0)+\frac{u}{3}[f'(-u)-f'(u)]. \end{aligned}$$

[1]) Auf Grund von (21) kann man allgemein zeigen, daß bei symmetrisch zum Nullpunkt gelegenen Stützstellen die Gewichte bei x_i und x_{-i} gleich sind.

Weiteres Differenzieren führt zu

$$r''(u) = \frac{1}{3} f'(u) - \frac{1}{3} f'(-u) - \frac{u}{3} [f''(-u) + f''(u)],$$

$$r'''(u) = -\frac{u}{3} [f'''(u) - f'''(-u)];$$

die Anwendung des Mittelwertsatzes der Differentialrechnung liefert schließlich

$$r'''(u) = -\frac{2}{3} u^2 f^{(\mathrm{IV})}(\xi), \quad -u \ldots \xi \ldots u,$$

und mit $\hat{M}_4$ als einer oberen Schranke für den Betrag der vierten Ableitung

$$|r'''(u)| \leqq \frac{2}{3} u^2 \hat{M}_4, \quad -1 \leqq u \leqq 1. \tag{27}$$

Durch sukzessive Integration dieser Ableitungen wird nun $r(u)$ rekonstruiert, was eine Betragsabschätzung dieser Größe ermöglicht. Mit einer zunächst noch unbestimmten Konstanten C gilt

$$r''(u) = \int_0^u r'''(w)\, dw + C.$$

Wegen $r''(0) = 0$ ist $C = 0$ zu setzen, und man erhält mit Beachtung von (27)

$$|r''(u)| \leqq \frac{2}{3} \hat{M}_4 \int_0^{|u|} w^2\, dw = \frac{2}{9} |u|^3 \hat{M}_4, \quad -1 \leqq u \leqq 1. \tag{28}$$

Weiter folgt

$$r'(u) = \int_0^u r''(w)\, dw + D;$$

da $r'(0)$ verschwindet, ist auch die Integrationskonstante D gleich Null zu setzen, so daß mit Beachtung von (28)

$$|r'(u)| \leqq \frac{1}{18} u^4 \hat{M}_4, \quad -1 \leqq u \leqq 1. \tag{29}$$

Schließlich ist wegen $r(0) = 0$

$$r(u) = \int_0^u r'(w)\, dw$$

– und auf Grund von (29) –

$$|r(u)| \leqq \frac{1}{90} |u|^5 \hat{M}_4, \quad -1 \leqq u \leqq 1. \tag{30}$$

Nunmehr folgt aus (26)

$$|RI_2(-1, 1)| = |r(1)| \leqq \frac{1}{90} \hat{M}_4.$$

Bei der Bestimmung des Verfahrensfehlers $RI_2(a, b)$ der Keplerschen Faßregel zur Berechnung des Integrals $\int_a^b f(x)\,dx$ im allgemeinen Fall hat man nach der Bemerkung im Anschluß an (19) zunächst $RI_2(-1, 1)$ mit $\frac{b-a}{2}$ zu multiplizieren. Sodann muß der Zusammenhang zwischen der Größe $\hat{M}_4$ der mit (17) auf das Intervall $[-1, 1]$ transformierten Funktion f und der entsprechenden Schranke M_4 für den Betrag der vierten Ableitung des Integranden auf $[a, b]$ überlegt werden. Nach (17) erhält man $\hat{M}_4 = \left(\frac{b-a}{2}\right)^4 M_4$. Damit ergibt sich zusammengefaßt

$$RI_2(a, b) \leqq \frac{1}{90}\left(\frac{b-a}{2}\right)^5 M_4 = \frac{(b-a)^5}{2880} M_4 \qquad (|f^{(\mathrm{IV})}(x)| \leqq M_4 \text{ auf } [a, b]). \quad (31)$$

Bemerkenswert ist, daß nach (31) sogar noch Polynome bis zum Grad 3 durch die Keplersche Faßregel exakt integriert werden, da für diese $M_4 = 0$ gewählt werden kann. Das folgt auch direkt aus dem Umstand, daß die betrachtete Quadraturformel den genauen Wert des Integrals $\int_{-1}^{1} x^3\,dx$ liefert.

Ein allgemeiner Ausdruck für das Restglied RI_n der Formeln von Newton-Cotes wurde von Steffensen auf Grund von (22) hergeleitet (vgl. etwa [74]).

Aufgabe 2. Nach der bei der Keplerschen Faßregel benutzten Methode zeige man für den Verfahrensfehler $RI_1(a, b)$ der Trapezregel

$$|RI_1(a, b)| \leqq \frac{(b-a)^3}{12} M_2.$$

Dabei werde angenommen, daß der Integrand f auf $[a, b]$ zweimal stetig differenzierbar und M_2 eine Schranke für $|f''|$ auf diesem Intervall ist.

Aufgabe 3. Man bestimme bezüglich des Intervalls $[-1, 1]$ die Gewichte der (offenen) Interpolationsquadraturformel

$$\int_{-1}^{1} f(x)\,dx \approx A_{-1} f\left(-\frac{1}{2}\right) + A_0 f(0) + A_1 f\left(\frac{1}{2}\right)$$

und zeige, daß diese für Polynome bis zum Grad 3 exakt ist.

Das im Anschluß an 4.2.4.(53) über den Interpolationsfehler Gesagte ist wegen (22) auch für den Verfahrensfehler einer Interpolationsquadraturformel zu beachten. Zur Erläuterung haben wir mit den Formeln von Newton-Cotes für $n+1 = 3(2)13$ das Integral $\int_{-8}^{8} \frac{dx}{1+x^2} = 2 \arctan 8 = 2.8928826$ „näherungsweise" berechnet.[1]) Die Ergebnisse sind in Tabelle 4.12 angegeben und zeigen, daß die beste Approximation des Integralwertes für $n+1 = 5$ erreicht wird, während die Rechnungen mit höherer Ordinatenzahl Ergebnisse liefern, die, um den wahren Wert oszillierend, immer mehr von diesem abweichen.

[1]) Die dazu erforderlichen Gewichte wurden den Tafeln [44] entnommen. Über ein numerisches Experiment dieser Art wird in [31] berichtet.

Zahl der Stützstellen	Integralwert
3	10.7487179
5	2.8409050
7	5.6112528
9	−0.0576087
11	8.8446749
13	−8.4528835

Tabelle 4.12

Dieser Effekte wegen ist es naheliegend, den Integranden nicht – wie bei der Herleitung der Formeln von NEWTON-COTES – durch ein Interpolationspolynom zu approximieren, sondern das Integrationsintervall geeignet zu zerlegen und über jedem Teilintervall gesondert Interpolationspolynome zu konstruieren. Wenn man für diese einheitlich einen Maximalgrad festlegt und wenn die entsprechende Ableitung im Interpolationsfehler (29) aus 4.2.4. vorhanden und über dem Integrationsintervall beschränkt ist, kann der Verfahrensfehler bei der Integration über diese stückweise Polynomapproximation durch Verfeinerung der Intervalleinteilung beliebig klein gemacht werden.

Wir erläutern die Methode genauer am Beispiel der *verallgemeinerten Trapezregel* und der *Simpsonschen Regel*. Bei der ersten wird das Integrationsintervall $[a, b]$ in k gleichlange Teile zerlegt und bezüglich eines jeden die Formel (24) für $n = 1$ angewandt. Es sei

$$h := \frac{b-a}{k}$$

und

$$\xi_0 := a, \quad \xi_1 := a+h, \ldots, \quad \xi_j := a+jh, \ldots, \xi_k := a+kh = b. \tag{32}$$

Dann ergibt sich auf diese Weise

$$\begin{aligned} \int_a^b f(x)\,dx &= \sum_{j=1}^{k} \int_{\xi_{j-1}}^{\xi_j} f(x)\,dx \\ &\approx \frac{h}{2}\left[f(\xi_0) + 2f(\xi_1) + \cdots + 2f(\xi_j) + \cdots + 2f(\xi_{k-1}) + f(\xi_k)\right] \qquad (33) \\ &= \frac{b-a}{k}\left[\frac{1}{2}f(a) + \sum_{j=1}^{k-1} f(a+jh) + \frac{1}{2}f(b)\right]. \end{aligned}$$

Für den Verfahrensfehler $RI_k^{(T)}$ bei der Anwendung von (33) erhält man mit Beachtung des Resultats von Aufgabe 2

$$|RI_k^{(T)}| = \left| \sum_{j=1}^{k} \left[\int_{\xi_{j-1}}^{\xi_j} f(x)\,dx - \frac{h}{2}\left(f(\xi_{j-1}) + f(\xi_j)\right) \right] \right|$$

$$\leqq \sum_{j=1}^{k} \left| \int_{\xi_{j-1}}^{\xi_j} f(x)\,dx - \frac{h}{2}\,(f(\xi_{j-1}) - f(\xi_j)) \right|$$

$$\leqq \sum_{j=1}^{k} \frac{h^3}{12} M_2^{(j)} \leqq \frac{M_2}{12}\,\frac{(b-a)^3}{k^2}. \tag{34}$$

Dabei bedeuten $M_2^{(j)}$ und M_2 Schranken für $|f''|$ auf dem Intervall $[\![\xi_{j-1}, \xi_j]\!]$ bzw. $[\![a, b]\!]$.

Bei der Herleitung der Simpsonschen Regel geht man von einer Zerlegung (32) mit geradzahligem $k = 2m$ aus (vgl. Abb. 4.14) und wendet bezüglich jedes der entstehenden Doppelintervalle der Länge $2h$ die Keplersche Faßregel an. Dann ist

$$\begin{aligned}
\int_a^b f(x)\,dx &= \sum_{j=1}^{m} \int_{\xi_{2(j-1)}}^{\xi_{2j}} f(x)\,dx \\
&\approx \sum_{j=1}^{m} \frac{h}{3}[f(\xi_{2(j-1)}) + 4f(\xi_{2j-1}) - f(\xi_{2j})] \\
&= \frac{h}{3}\left[f(a) + 2 \sum_{j=1}^{m-1} f(\xi_{2j}) + 4 \sum_{j=1}^{m} f(\xi_{2j-1}) + f(b) \right] \\
&= \frac{h}{3}\left[f(a) + 2 \sum_{j=1}^{m-1} f(a+2jh) + 4 \sum_{j=1}^{m} f(a+(2j-1)h) + f(b) \right].
\end{aligned} \tag{35}$$

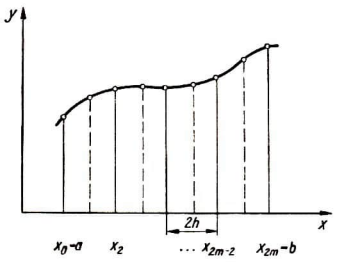


Abb. 4.14

Für den Verfahrensfehler $RI_{2m}^{(S)}$ bei der Anwendung von (35) erhält man auf Grund von (31)

$$|RI_{2m}^{(S)}| = \left| \sum_{j=1}^{m} \left\{ \int_{\xi_{2(j-1)}}^{\xi_{2j}} f(x)\,dx - \frac{h}{3}[f(\xi_{2(j-1)}) + 4f(\xi_{2j-1}) + f(\xi_{2j})] \right\} \right|$$

$$\leqq \sum_{j=1}^{m} \left| \int_{\xi_{2(j-1)}}^{\xi_{2j}} f(x)\,dx - \frac{h}{3}[f(\xi_{2(j-1)}) + 4f(\xi_{2j-1}) + f(\xi_{2j})] \right|$$

$$\leqq \sum_{j=1}^{m} \frac{(2h)^5}{2880} M_4^{(j)} \leqq \frac{M_4(b-a)^5}{2880m^4}. \tag{36}$$

In (36) bedeuten $M_4^{(j)}$, M_4 obere Schranken für $|f^{(\mathrm{IV})}|$ auf dem Intervall $[\xi_{2(j-1)}, \xi_{2j}]$ bzw. $[a, b]$. Ist ε eine gegebene obere Schranke für den Betrag des Verfahrensfehlers, so genügt nach (36) die Schrittweite dieser Genauigkeitsforderung, wenn

$$h < \sqrt[4]{\frac{180\,\varepsilon}{M_4(b-a)}}. \tag{37}$$

Bei der Herleitung der Trapezregel wird als stückweise polygonale Approximation – geometrisch gesprochen – der dem Graphen von f über der Zerlegung (32) einbeschriebene Sehnenzug benutzt. Im Fall der Simpsonschen Formel erfolgt über den m Doppelintervallen eine Ersetzung durch Parabelbögen. Wir wollen die Auswertung von (33) und (35) in einer Funktionsprozedur *TRAPEZ* bzw. *SIMPSON* zusammenfassen. Als formale Parameter verwenden wir die Integrationsgrenzen a, b, die Bezeichnung f der zu integrierenden Funktion und die Anzahl k der Teilintervalle in (32) bzw. den halben Wert m. Lokale Variable, welche Bezeichnungen bisher benutzter Größen tragen, haben auch deren Bedeutung.

```
real procedure TRAPEZ(a, b, f, k); value a, b;
integer k; real a, b; real procedure f;
  begin integer j; real h, x, s;
    h := (b−a)/k; s := (f(a)+f(b))×0.5; x := a;
    for j := 1 step 1 until k−1 do
    begin x := x+h; s := s+f(x) end;
    s := h×s;
    TRAPEZ := s
  end
```

In *SIMPSON* benutzen wir zur Summation der Ordinaten mit dem Gewicht 2 und 4 je eine Variable *sg* bzw. *su*.

```
real procedure SIMPSON(a, b, f, m); value a, b;
integer m; real a, b; real procedure f;
  begin
    integer j; real x, sg, su, h, h2;
    h2 := (b−a)/m; h := h2×0.5; x := a; sg := su := 0;
    for j := 1 step 1 until m do                    begin
    x := x+h2; sg := sg+f(x); su := su+f(x−h) end;
    sg := (4×su+2×sg+f(a)−f(b))/3×h;
    SIMPSON := sg
  end
```

Für die Tabulierung von Integralen mit variabler oberer Grenze zwischen a und b mit der Schrittweite $2h$ eignet sich folgende Modifikation von *SIMPSON*, die mit Rücksicht auf die automatenspezifische Ausgabe in R 300-ALGOL formuliert wird. Diese Prozedur *tabu* enthält einige rechentechnische Verbesserungen, die auch in *SIMPSON* genutzt werden können.

```
'procedure' tabu(a, b, f, m, ); 'value' a, b;
'integer' m; 'real' a, b; 'real' 'procedure' f;
    'begin' 'integer' j; 'real' x, y, y1, s, h, h2, h3;
            h2 := (b−a)/m; h := h2 * ø.5; h3 := h/3;
            x := a; s := ø; y1 := f(a);
            'for' j := 1 'step' 1 'until' m 'do'          'begin'
            y := y1; x := x+h2; y1 := f(x);
            s := s+(y+4 * f(x−h) +y1) * h3; print (x,s)   'end'
            'end'
```

Wir erläutern den Gebrauch dieser Prozedur an einem Beispiel:

Für $t = 1{,}1(0{,}1)2{,}0$ sei $\ln t$ auf fünf Dezimalen genau zu berechnen. Wegen $\ln t = \int_1^t \frac{1}{x}\,dx$ kann diese Aufgabe als ein Quadraturproblem behandelt werden. Wir benutzen die Simpsonsche Regel und haben zunächst auf Grund einer Betrachtung des Verfahrensfehlers eine für die geforderte Genauigkeit hinreichende Schrittgröße h zu bestimmen. Auf dem Intervall $[1, 2]$ ist $M_4 := 24$ kleinste obere Schranke des Betrages von

$$\frac{d^4}{dx^4}\frac{1}{x} = \frac{24}{x^5},$$

so daß man für die rechte Seite von (37) mit $\varepsilon = 10^{-5}$ den Wert $0{,}1 \cdot \sqrt[4]{0{,}75}$ erhält. Der Wert $h := 0{,}05$ entspricht also der Genauigkeitsforderung bei der Berechnung von $\int_1^t \frac{1}{x}\,dx$ für $t = 2$ und a fortiori für die übrigen t-Werte. Folgendes R 300-Programm wäre für die automatische Bearbeitung einschließlich Tabulierung geeignet:

```
'begin'
  'integer' m; 'real' a, b;
  'real' 'procedure' f(x); 'value' x; 'real' x;
  f := 1/x;
  'procedure' tabu(a, b, f, m);
     :
  ;
  read(a, b, m); tabu(a, b, f, m);
'end'
```

Dem Programm sind die Daten 1, 2, 1 ø beizufügen.

In dem betrachteten Beispiel wurde eine für die geforderte Genauigkeit hinreichende Schrittgröße vorher bestimmt und bei der Abarbeitung des Programms eingegeben. Wir gehen kurz auf eine Möglichkeit der automatischen Bestimmung der Schrittweite bei der Simpsonschen Formel ein. Aus (37) folgt

$$\frac{h}{2} < \frac{1}{2}\sqrt[4]{\frac{180\,\varepsilon}{M_4(b-a)}} < \sqrt[4]{\frac{180\cdot 10^{-1}\varepsilon}{M_4(b-a)}},$$

d. h., erfüllt h die Bedingung (37) bezüglich ε, so $\frac{h}{2}$ bezüglich der Fehlerschranke $10^{-1}\varepsilon$. Auf diese Bemerkung gründet sich folgende Regel:

Wenn bei der zweimaligen Berechnung eines Integrals nach der Simpsonschen Formel mit der Schrittweite h bzw $\frac{h}{2}$ l Dezimalstellen übereinstimmen, so betrachtet man diese als gültig.

Aufgabe 4. Man programmiere die Berechnung eines Integrals nach der Simpsonschen Formel mit automatischer Bestimmung der Schrittweite.

Bisher wurde nur der Verfahrensfehler bei Anwendung einer Quadraturformel untersucht. Numerische Fehler resultieren wesentlich aus der Berechnung der darin auftretenden Ordinaten. Nehmen wir an, daß deren absolute Fehler dem Betrage nach kleiner als ε, die Gewichte A_i hingegen exakt sind, so läßt sich der absolute Fehler R_{num} bei der als exakt angenommenen Auswertung von (11) durch

$$|R_{\text{num}}| \leqq \sum_{i=0}^{n} |A_i|\varepsilon$$

majorisieren. Bei einer Quadraturformel mit positiven Gewichten folgt daraus mit Beachtung von (12)

$$|R_{\text{num}}| \leqq (b-a)\varepsilon. \tag{38}$$

Die Theorie der angenäherten Berechnung von Integralen ist ein sehr entwickeltes Teilgebiet der numerischen Analysis, das im Rahmen dieser Darstellung nur andeutungsweise umrissen werden konnte. Neben den hier behandelten Quadraturformeln existiert eine große Zahl weiterer, die nach bestimmten Approximationsgesichtspunkten konzipiert wurden. Erwähnt seien etwa die *Formeln von Gauß*, die unter denen des Typs (11) dadurch ausgezeichnet sind, daß sie noch für Polynome des Grades $N = 2n+1$ den exakten Integralwert liefern. Durch diese Forderung sind die Gewichte und Knoten eindeutig bestimmt. Speziell müßten nämlich die Potenzen x^k, $k = 0, 1, \ldots, 2n+1$, durch die Formeln von Gauss exakt integriert werden, und das führt an Stelle von (23) zu

$$\int_{-1}^{1} x^k dx = x_0^k A_0 + x_1^k A_1 + \cdots + x_n^k A_n, \quad k = 0, 1, \ldots, 2n+1. \tag{39}$$

Man kann zeigen, daß dieses System von $2(n+1)$ Gleichungen für die Unbekannten $x_0, x_1, \ldots, x_n$; $A_0, A_1, \ldots, A_n$ genau eine Lösung besitzt. Die Stützstellen x_i erweisen sich als die im Intervall $]-1, 1[$ gelegenen Nullstellen der sogenannten

Legendreschen Polynome P_n, welche etwa durch

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n (x^2-1)^n}{dx^n} \tag{40}$$

definiert werden können.

Eine wichtige Frage ist die Auswahl der für Integranden einer bestimmten Funktionenklasse günstigsten Quadraturformel. Diesbezüglich mag der Hinweis genügen, daß man zu jeder Quadraturformel Integranden konstruieren kann, für welche diese zur Integralapproximation nicht geeignet sind. Beispielsweise würde die Keplersche Faßregel für die in Abb. 4.15 dargestellte Funktion f_n ($n \in \mathbf{N}$, $n \geqq 2$) den Wert -2 liefern, während

$$\int_{-1}^{1} f_n(x)\, dx = 2 - \frac{8}{n}$$

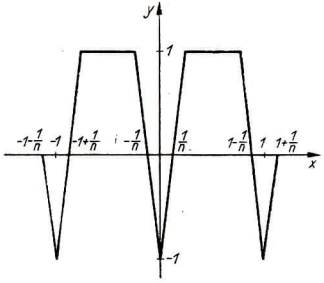


Abb. 4.15

ist. Für $n = 2$ stimmt das mit dem Wert der Quadraturformel überein, aber für $n \geqq 4$ wird nicht einmal das Vorzeichen des Integrals reproduziert. Das Beispiel läßt erkennen, daß die numerische Quadratur bei Integranden mit großer Schwankung der Funktionswerte problematisch ist. Zur Entwicklung effektiver universeller automatischer Quadraturverfahren (sogenannter automatischer Integratoren) erscheinen gegenwärtig viele Veröffentlichungen.

Von den modernen Verfahren sei noch die mit Zufallszahlen arbeitende sogenannte *Monte-Carlo-Methode* erwähnt, die sich auch zur Berechnung mehrdimensionaler Integrale eignet.

Wir schließen mit einem Hinweis auf die Möglichkeit, funktionalanalytische Hilfsmittel zur Untersuchung des Verfahrensfehlers und der *Konvergenz einer Quadraturformel* einzusetzen. Letzteres betrifft die Frage, unter welchen Voraussetzungen die von einer Quadraturformel bestimmten Typs gelieferten Werte mit wachsender Stützstellenzahl gegen den exakten Integralwert streben. Für ein erstes Studium solcher Probleme sei etwa auf [77], Bd. 2, 9.7., verwiesen. Ein schönes Resultat stammt von dem bedeutenden russischen Mathematiker W. A. Steklow (1863–1926):

Wenn die Gewichte einer Quadraturformel (11) *bestimmten Typs sämtlich nichtnegativ sind und diese bei* $n+1$ *Stützstellen den Integralwert eines beliebigen Polynoms von*

höchstens n-tem Grade exakt bestimmt, dann konvergiert die Formel für jeden stetigen Integranden.

Im Hinblick auf das oben untersuchte Beispiel der Berechnung von

$$\int_{-8}^{8} \frac{dx}{1+x^2}$$

wird man (zu Recht) vermuten, daß in den Formeln von NEWTON-COTES auch negative Gewichte auftreten müssen.

Literatur

[1] Alexandrow, A., Mathematik und Dialektik, Ideen des exakten Wissens 4 (1971), 251–257.

[2] Andersen, Ch., ALGOL 60 – Eine Sprache für Rechenautomaten, VEB Verlag Technik, Berlin 1966.

[3] Asser, G., Einführung in die mathematische Logik, Teil I (4. Aufl.), II, BSB B. G. Teubner Verlagsgesellschaft, Leipzig 1972.

[4] Bachmann, K. H., Programmierung für Digitalrechner, 4. Aufl., VEB Deutscher Verlag der Wissenschaften, Berlin 1970.

[5] Bachmann, K. H., ALGOL-Programmierung mit Variante für Robotron 300, 5. Aufl., VEB Deutscher Verlag der Wissenschaften, Berlin 1972.

[6] Backus, J. W., The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference, in: Proc. Intern. Conf. Inf. Proc. UNESCO, Paris 1959.

[7] Bär, D., Einführung in die Schaltalgebra, VEB Verlag Technik, Berlin 1966.

[8] Barsow, A. S., Was ist lineare Programmierung?, B. G. Teubner, Leipzig 1964 (Übersetzung aus dem Russischen).

[9] Baschmakowa, I. G., und A. P. Juschkewitsch, Die Entstehung der Bezeichnungssysteme für Zahlen, in: Enzyklopädie der Elementarmathematik, Bd. I, 7. Aufl., VEB Deutscher Verlag der Wissenschaften, Berlin 1978, S. 1–60 (Übersetzung aus dem Russischen).

[10] Bauer, F. L., J. Heinhold, K. Samelson und R. Sauer, Moderne Rechenanlagen, B. G. Teubner, Stuttgart 1965.

[11] Beresin, I. S., und N. P. Shidkow, Numerische Methoden, Bd. 1, 2, VEB Deutscher Verlag der Wissenschaften, Berlin 1970, 1971 (Übersetzung aus dem Russischen).

[12] Блехман, И. И., А. Д. Мышкис и Я. Г. Пановко, Правдоподобность и доказательность в прикладной математике, Механика твердого тела 2 (1967), 196–202.

[13] Boltzmann, L., Populäre Schriften, J. A. Barth, Leipzig 1905.

[14] Church, A., Introduction to Mathematical Logic, Vol. 1, University Press, Princeton 1956.

[15] Collatz, L., Funktionalanalysis und numerische Mathematik, Springer-Verlag, Berlin-Göttingen-Heidelberg 1964.

[16] Davis, Ph., Errors of numerical approximation for analytic functions, J. Rat. Mech. Anal. 2 (1953), 303–313, oder in: J. Todd (ed.), Survey of Numerical Analysis, McGraw-Hill, New York-San Francisco-Toronto-London 1962, p. 468 – 484.

[17] Demidowitsch, B. P., I. A. Maron und E. S. Schuwalowa, Numerische Methoden der Analysis, VEB Deutscher Verlag der Wissenschaften, Berlin 1968 (Übersetzung aus dem Russischen).
[18] Döring, B., Das Fixpunktprinzip in der Analysis, in: D. Laugwitz (Hrsg.), Überblicke Mathematik II, Bibliographisches Institut, Mannheim 1969, S. 151 bis 210.
[19] Einführung in die sozialistische Produktion, Lehrbuch für Klasse 10, Verlag Volk und Wissen, Berlin 1973.
[20] Elektronische Datenverarbeitung, Broschüre zum Fernsehkurs unter Verwendung der Autorenmanuskripte von G. Voigt und G. Huhn, Verlag Die Wirtschaft, Berlin 1969.
[21] Faddejew, D. K., und W. N. Faddejewa, Numerische Methoden der linearen Algebra, 4. Aufl., VEB Deutscher Verlag der Wissenschaften, Berlin / R. Oldenbourg, München 1976 (Übersetzung aus dem Russischen).
[22] Galilei, G., Unterredungen und mathematische Demonstrationen, Ostwald's Klassiker der exakten Wissenschaften, Bd. 11, 24, 25, Teubner, Leipzig 1890.
[23] Gluschkow, V. M., Einführung in die technische Kybernetik, VEB Verlag Technik, Berlin 1970 (Übersetzung aus dem Russischen).
[24] Göttner, R., Was ist – was soll Operationsforschung?, 3. Aufl., Urania-Verlag, Leipzig-Jena-Berlin 1972.
[25] Götzke, H., Programmgesteuerte Rechenautomaten, VEB Fachbuchverlag, Leipzig 1969.
[26] Greville, T. N. E., Data fitting by spline functions, MRC Technical Summary Report # 893, Madison (Wisc.) 1968.
[27] Greville, T. N. E., Spline functions, interpolation, and numerical quadrature, in: A. Ralston and H. S. Wilf, Mathematical Methods for Digital Computers, Vol. II, John Wiley & Sons, New York-London-Sidney 1967, Chapter 8.
[28] Heinrich, H., Numerische Behandlung nichtlinearer Gleichungen, in: D. Laugwitz (Hrsg.), Überblicke Mathematik II, Bibliographisches Institut, Mannheim 1969, S. 99–129.
[29] Henrici, P., Elemente der numerischen Analysis, Bibliographisches Institut, Mannheim-Wien-Zürich 1972.
[30] Hermes, H., Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit, 2. Aufl., Springer-Verlag, Berlin-Heidelberg-New York 1971.
[31] Hildebrand, F. B., Introduction to Numerical Analysis, McGraw-Hill, New York-London 1956.
[32] Huhn, G., Grundlagen, Aufbau, Wirkungsweise und Nutzung elektronischer Datenverarbeitungsanlagen, Verlag Die Wirtschaft, Berlin 1969.
[33] Isaacson, E., und H. B. Keller, Analyse numerischer Verfahren, Edition, Leipzig 1972 (Übersetzung aus dem Englischen).
[34] Kämmerer, W., Einführung in mathematische Methoden der Kybernetik, Akademie-Verlag, Berlin 1971.
[35] Kaiser, H., Studienmaterial zur Praktischen Analysis, Teil 1, Lehrbriefe für das Fernstudium der Lehrer, Potsdam 1969.
[36] Kaloujnine, L. A., Über die Algorithmierung mathematischer Aufgaben, in: Probleme der Kybernetik, Bd. 2, Akademie-Verlag, Berlin 1963, S. 54–74 (Übersetzung aus dem Russischen).
[37] Kern, L., K. Ober und J. Schumann, ESER-Programmierung im Betriebssystem DO/ES, VEB Verlag Technik, Berlin 1973.
[38] Kerner, I. O., und G. Zielke, Einführung in die algorithmische Sprache ALGOL, B. G. Teubner, Leipzig 1966.
[39] Kiesewetter, H., und G. Maess, Elementare Methoden der numerischen Mathematik, Akademie-Verlag, Berlin 1974.

[40] KIESEWETTER, H., Vorlesungen über lineare Approximation, VEB Deutscher Verlag der Wissenschaften, Berlin 1973.
[41] KNUTH, D. E., The Art of Computer Programming, Vol. 2, Addison-Wesley Publ. Comp., Reading (Mass.)-Palo Alto-London 1969.
[42] KRASNOSELSKI, M. A., u. a., Näherungsverfahren zur Lösung von Operatorgleichungen, Akademie-Verlag, Berlin 1973 (Übersetzung aus dem Russischen).
[43] KREIFELTS, T., Über vollautomatische Erfassung und Abschätzung von Rundungsfehlern in arithmetischen Prozessen, Bonner Berichte der Gesellschaft für Mathematik und Datenverarbeitung, Bd. 50, Bonn 1971.
[44] Крылов, В. И., и Л. Т. Шульгина, Справочная книга по численному интегрированию, Наука, Москва 1966.
[45] KUBISCH, W., und R. WITSCHURKE, ALGOL 60 für Robotron 300, Verlag Die Wirtschaft, Berlin 1970.
[46] LASDON, L. S., Optimization Theory for Large Systems, The Mac Millan Company, London 1970.
[47] LERNER, A., Grundzüge der Kybernetik, VEB Verlag Technik, Berlin 1971.
[48] MANGOLDT, H. v., und K. KNOPP, Einführung in die höhere Mathematik, Bd. III, 13. Aufl., Hirzel, Leipzig 1970.
[49] SCHRÖDER, K. (Hrsg.), Mathematik für die Praxis I–III, 3. Aufl., VEB Deutscher Verlag der Wissenschaften, Berlin 1966.
[50] MAURER, H., Theoretische Grundlagen der Programmiersprachen – Theorie der Syntax, Bibliographisches Institut, Mannheim-Wien-Zürich 1969.
[51] MURPHY, J. S., Elektronische Ziffernrechner, VEB Verlag Technik, Berlin 1965 (Übersetzung aus dem Englischen).
[52] NEUGEBAUER, O., Vorlesungen über Geschichte der Mathematik, Bd. 1, Springer, Berlin 1934.
[53] NEUGEBAUER, O., The Exact Sciences in Antiquity, 2. Aufl., Providence, R. I., 1957.
[54] NEUGEBAUER, O. (Hrsg.), Mathematische Keilschrifttexte, in: Quellen und Studien zur Geschichte der Mathematik, Springer, Berlin 1931–1933, A, 3.
[55] Autorenkollektiv, Operationsforschung in der sozialistischen Wirtschaft, Dietz Verlag, Berlin 1969.
[56] PAULIN, G., Grundzüge des Programmierens, VEB Verlag Technik, Berlin 1972.
[57] PENNINGTON, R. H., Introductory Computer Methods and Numerical Analysis, The Mac Millan Company, New York 1970.
[58] Pläne für den fakultativen mathematisch-naturwissenschaftlichen Unterricht in der EOS, Lehrgang Grundlagen der Rechentechnik.
[59] POHL, R. W., Einführung in die Mechanik, Akustik und Wärmelehre, Springer, Berlin 1944.
[60] POLOSHI, G. N., Mathematisches Praktikum, B. G. Teubner, Leipzig 1963 (Übersetzung aus dem Russischen).
[61] NAUR, P. (ed.), Report on the algorithmic language ALGOL 60, ALGOL Bulletin Supplement No. 2, oder Numer. Math. **2** (1960), 106–136. (Deutsche Übersetzung: Berichte über die algorithmische Sprache ALGOL 60, Elektronisches Rechnen und Regeln, Sonderband 1, Akademie-Verlag, Berlin 1962).
[62] Report on SUBSET ALGOL 60 (IFIP), Numer. Math. **6** (1964), 454–458.
[63] RICE, J. R., The Approximation of Functions, Addison-Wesley Publ. Comp., Reading (Mass.)-Palo Alto-London 1964.
[64] ROHLEDER, H., Die Verwendung von Aussagenkalkülen zur Beschreibung elektrischer Schaltungen, Z. Math. Logik und Grundl. Math. **1** (1955), 304–309.
[65] RUTISHAUSER, H., Description of ALGOL 60, in: Handbook for Automatic Computation, Vol. 1, Springer-Verlag, Berlin-Heidelberg-New York 1967.
[66] SACHS, H., Einführung in die Theorie der endlichen Graphen I–II, BSB B. G. Teubner Verlagsgesellschaft, Leipzig 1970, 1972.

[67] Southworth, R. W., Digital Computation and Numerical Methods, McGraw-Hill, New York 1965.
[68] Struik, D. J., Abriß der Geschichte der Mathematik, 6. Aufl., VEB Deutscher Verlag der Wissenschaften, Berlin 1976 / 4. Aufl., Verlag Vieweg & Sohn, Braunschweig 1967 (Übersetzung aus dem Amerikanischen).
[69] Stegun, I. A., und M. Abramowitz, Pitfalls in computation, J. SIAM 4 (1956), 207–219.
[70] Trachtenbrot, B. A., Wieso können Automaten rechnen?, 6. Aufl., VEB Deutscher Verlag der Wissenschaften, Berlin 1971 (Übersetzung aus dem Russischen).
[70a] Trachtenbrot, B. A., Algorithmen und Rechenautomaten, VEB Deutscher Verlag der Wissenschaften, Berlin 1977 (Übersetzung aus dem Russischen).
[71] Weizsäcker, C. F. Frh. v., Die philosophische Interpretation der modernen Physik, Nova Acta Leopoldina, Neue Folge, Nr. 207, Bd. 37/2.
[72] Weyh, U., Elemente der Schaltungsalgebra, R. Oldenbourg, München 1960.
[73] Wilkinson, J. H., Rounding Errors in Algebraic Processes, Her Majesty's Stationery Office, London 1963.
[74] Willers, F. A., Methoden der praktischen Analysis, 3. Aufl., W. de Gruyter, Berlin 1957.
[75] Klaus, G. (Hrsg.), Wörterbuch der Kybernetik, Dietz-Verlag, Berlin 1967.
[76] Klaus, G., und M. Buhr (Hrsg.), Philosophisches Wörterbuch, Bibliographisches Institut, Leipzig 1971.
[77] Wulich, B. S., Einführung in die Funktionalanalysis I–II, BSB B. G. Teubner Verlagsgesellschaft, Leipzig 1961, 1962 (Übersetzung aus dem Russischen).
[78] Wussing, H., Mathematik in der Antike, BSB B. G. Teubner Verlagsgesellschaft, Leipzig 1965.

Namen- und Sachverzeichnis