Inhaltsverzeichnis

Vorwort

Der vorliegenden Band enthält die Lösungen der Aufgaben des in derselben Reihe *Programm Praxis* erschienenen Lehrbuches 'Computermathematik'. Die Kapiteleinteilung wurde beibehalten, die Aufgabenstellung wiederholt, um dem Leser ein gleichzeitiges Blättern in beiden Büchern zu ersparen. Für die zur Lösung erforderliche Theorie wird auf das Lehrbuch verwiesen.

Bei den Lösungen der Aufgaben handelt es sich meistens um Programme, die in TURBO PASCAL angegeben sind. Bei Programmierübungen ist es sehr wichtig, dass man nicht nur lernt, selbst Programme zu schreiben, sondern auch, andere zu lesen, um eventuelle Anpassungen vornehmen zu können. Die vorliegenden Programme sollen diesem Zweck dienen. Der Leser sollte seine eigenen Lösungen mit den hier angeführten vergleichen. Bei grösseren Programmen gehen wir meistens in zwei Schritten vor: Zuerst wird ein 'Traktor' und erst anschliessend die 'Luxuslimusine' gebaut. Damit ist gemeint, dass man zuerst ein kurzes, lauffähiges Programm erstellt, das eine korrekte Implementation des Algorithmus darstellt, jedoch eventuell nur für Spezialfälle funktioniert. Erst danach wird das Programm für den allgemeinen Fall erweitert. Als Beispiel sei auf den Algorithmus 4.7 hingewiesen, mit dem Polynomnullstellen mittels Newtonverfahren und Deflation berechnet werden. Der Algorithmus funktioniert nur für reelle Arithmetik und ist zwar kurz und übersichtlich, aber im allgemeinen unbrauchbar. Es ist aber nicht schwierig, diesen Algorithmus für komplexe Arithmetik umzuschreiben, wie in Aufgabe 4.17 verlangt wird. Das neue Programm ist wesentlich länger und wäre ohne den vorher konstruierten 'Traktor' auch schwieriger zu programmieren.

Programme können immer geändert, verbessert und erweitert werden. Der Autor freut sich über jede Anregung aus dem Leserkreis. Es wurde darauf geachtet, dass weitgehend nur Elemente des Standard PASCAL verwendet werden und dass dadurch die Programme maschinenunabhängig, d.h. mit wenig Änderungen auf einen anderen Computer übertragbar sind. Das Buch wurde mit MicroTEX geschrieben. Alle Programme konnten dadurch direkt in den Text hineinkopiert werden. So besteht grosse Hoffnung, dass sich keine Druckfehler in ihnen eingeschlichen haben.

Als nichtstandard Element benüzen wir den Typus **string**, der aber doch in den meisten Implementationen von PASCAL vorhanden ist. Ferner erlaubt TURBO PASCAL nicht, Prozeduren als Parameter zu verwenden. Wir verzichten daher auf diese Möglichkeit und verwenden nur globale Funktionen oder Prozeduren. Dagegen machen wir oft davon Gebrauch, dass ein File *text.pas* durch 'Include' – Komentare (∗$I *text* ∗) vom Compiler in ein anderes Programm hineinkopiert werden kann. Dadurch können oft gebrauchte

Prozeduren in einer Bibliothek abgelegt werden mit dem Vorteil, dass bei einer eventuellen Änderung automatisch alle Programme die neue Version benützen. Diese Include–Kommentare sind auch kein Element von Standard PASCAL, aber in den meisten Implementationen vorhanden.

Bei einigen Aufgaben besteht die Lösung aus einer Funktion (beispielsweise bei der Splineinterpolation oder bei den Differentialgleichungen). Es ist unbefriedigend, wenn man sich mit einer Wertetabelle als Lösung zufrieden geben muss. In TURBO PASCAL sind Prozeduren für graphischen Output vorhanden. Wir beschränken uns auf das *'basic graphics'*, das bei allen Versionen von TURBO PASCAL vorhanden ist und geben in Kapitel 8 einige Prozeduren an, welche es ermöglichen, sehr bequehm Funktionen auf dem Bildschirm darzustellen. Zum Beispiel zeichnet die Prozedur *achsen* die Koordinatenachsen und beschriftet sie automatisch vernünftig für alle möglichen Fälle.

Für den Input/Output Fluss benützen wir einen Mechanismus, der auch nicht Standard, aber häufig anzutreffen ist. Oft möchte man *erst zur Ausführungszeit bestimmen*, ob die Resultate auf den Bildschirm, den Drucker oder auf ein Diskfile geschrieben werden sollen. Logisch werden alle diese Geräte als 'Files' betrachtet. Die logischen Filenamen für TURBO PASCAL und UCSD PASCAL sind aus der folgenden Tabelle ersichtlich:

I/O Gerät	TURBO PASCAL	UCSD PASCAL
Diskfile	A:res.txt	#4:res.text
Drucker	LST:	#6: *oder* PRINTER:
Bildschirm	CON:	#1: *oder* CONSOLE:

Im Program wird eine Filevariable vom Typus *text* vereinbart: **var** *aus* : *text*. Bei Beginn der Ausführung des Programms kann ihr in TURBO PASCAL, z.B. durch die Anweisung $assign(aus,$'LST:') der Drucker zugewiesen werden und die Schreibanweisung $writeln(aus, x)$ bewirkt, dass der Wert der Variablen x auf dem Drucker ausgegeben wird. Wenn man als zweiten Parameter von *assign* eine **string** Variable verwendet, kann der Name des Outputfiles eingelesen werden. Das folgende Programmfragment möge als Beispiel dienen:

```
program xxx;
var aus, ein : text; stri : string [20];
begin
   writeln('wohin mit dem Output ?');
   readln(stri); (* Output–Filenamen einlesen *)
   assign(aus, stri);
      (* Der File Variablen 'aus' wird der aktuelle Name zugewiesen *)
   rewrite(aus); (* zurueckspulen des Outputfiles *)
```

```
writeln('woher kommt der Input ?');
readln(stri); (* Filenamen fuer den Input *)
assign(ein, stri);
   (* Der File Variablen 'ein' wird der aktuelle Name zugewiesen *)
reset(ein); (* zurueckspulen des Inputfiles *)
      ..................
read(ein, ...); (* vom File 'ein' lesen *)
      ..................
writeln(aus, ...) ; (* auf das File 'aus' schreiben *)
      ..................
close(aus);
      (* abschliessen des Outputfiles, EOF Marke schreiben *)
end.
```

In UCSD PASCAL kann dieses Programmschema übernommen werden, wenn man die Anweisungen für das Outputfile (und analog für das Inputfile) $assign(aus, stri)$; $rewrite(aus)$ durch $rewrite(aus, stri)$ und die Anweisung $close(aus)$ durch $close(aus, lock)$ ersetzt.

Bei Verweisen auf Gleichungen des Lehrbuches 'Computermathematik' wird an der Gleichungsnummer ein '$*$' angehängt: Gleichung (2.1$*$) bedeutet also die Gleichung (2.1) aus dem Lehrbuch, die Gleichung (2.1) dagegen eine aus dem vorliegenden Lösungsbuch.

Die Programmfiles werden auf der Diskette wie folgt benannt: *A2.1* ist der **Algorithmus 2.1** und das Programm, welches Lösung der **Aufgabe 2.6** ist, hat den Filenamen *AUFG2.6*. Files, die als Include - Files in andere Programme geladen werden, haben meistens ihren Programmnamen als Filenamen z.B. *topolar.pas*.

Einige der vorliegenden Programme hat mein Bruder Maurice Gander während seines Weiterbildungsurlaubes geschrieben. Ich möchte ihm dafür und für die anregenden Diskussionen herzlich danken. Ferner gilt mein Dank auch Philipp Schwizer, der mir bei der Druckfehlersuche behilflich war. Dem Institut für Astronomie an der ETHZ danke ich für die Untestützung bei der Herstellung der Druckvorlage.

Werdenberg, Frühling 1986 Walter Gander

Kapitel 1

Rechnen mit Computer

Aufgabe 1.2 *Gegeben sei die folgende endliche dezimale Arithmetik: Mantisse: 2 Dezimalstellen, Exponent : 1 Dezimalstelle.*
a) Wieviele normalisierte Maschinenzahlen gibt es ?
b) Über– und Unterlaufbereich ?
c) Maschinengenauigkeit ?
d) Welches ist der grösste (kleinste) Abstand zweier aufeinanderfolgender Maschinenzahlen ?

Die Maschinenzahlen haben in dieser Arithmetik die Form $\pm Z.Z\text{E}\pm Z$, wobei für Z eine der Ziffern $0, 1, 2, \ldots, 9$ steht.

a) Es sind 19 Exponenten möglich $-9, -8, \ldots, 9$ und wenn wir von der Zahl 0 absehen, gibt es wegen der Normalisierung nur 9 Möglichkeiten für die Ziffer vor dem Dezimalpunkt. Es ergeben sich daher $2*9*10*19 = 3420$ Zahlen, wobei wir die Zahl $0 = 0.0\text{E}0$ noch nicht mitgezählt haben. Total gibt es also 3421 Maschinenzahlen.

b) $M = 9.9\text{E}9 = 9'900'000'000$ und $m = 1.0\text{E}{-9}$. Der Überlaufbereich ist $|x| > 9.9\text{E}9$ und der Unterlaufbereich ist das Intervall $|x| < 1.0\text{E}{-9}$.

c) Maschinengenauigkeit: Es ist

$$1.0\text{E}0 + 5\text{E}{-2} = 1.05 \overset{\text{gerundet}}{=} 1.1\text{E}0$$

also ist $\tilde{\varepsilon} = 5\text{E}{-2}$.

d) Der Abstand zwischen zwei Maschinenzahlen ist am grössten, wenn der Exponent 9 beträgt: z.B. ist $9.9\text{E}9 - 9.8\text{E}9 = 1\text{E}8$. Der kleinste Abstand zwischen zwei aufeinanderfolgenden Maschinenzahlen beträgt $1.1\text{E}{-9} - 1.0\text{E}{-9} = 1\text{E}{-10}$.

Aufgabe 1.3 *Man simuliere auf einem Taschenrechner die in Aufgabe 1.2 definierte Arithmetik, indem nach jeder Operation das Resultat auf 2 Dezimalstellen gerundet wird und löse die folgenden Aufgaben:*
a) Man löse die quadratische Gleichung $x^2 - 64x + 1 = 0$ mit der üblichen Formel (1.2$*$) und mit der stabilen Version.
b) Man berechne die Seite c eines Dreiecks nach dem Cosinussatz und mit der stabilisierten Version davon, wenn $a = 5.6$, $b = 5.7$ und $\gamma = 5^\circ$.
Man vergleiche die Resultate mit den exakten Lösungen.

a) Die exakten Lösungen sind

$$x_{1,2} = 32 \pm \sqrt{1023} = \begin{cases} 63.98437118 \\ 0.01562881656 \end{cases}$$

Mit der üblichen Formel ist

$$\begin{aligned} \frac{p}{2} &= 6.4\text{E}1/2.0\text{E}0 = 3.2\text{E}1 \quad \text{(ohne Fehler)} \\ \left(\frac{p}{2}\right)^2 &= (3.2\text{E}1)^2 = 1024 \overset{\text{gerundet}}{=} 1.0\text{E}3 \\ \left(\frac{p}{2}\right)^2 - q &= 1.0\text{E}3 - 1.0\text{E}0 = 999 \overset{\text{gerundet}}{=} 1.0\text{E}3 \\ \sqrt{\left(\frac{p}{2}\right)^2 - q} &= \sqrt{1.0\text{E}3} = 31.6227 \overset{\text{gerundet}}{=} 3.2\text{E}1 \end{aligned}$$

somit ist

$$\begin{aligned} x_1 &= 3.2\text{E}1 + 3.2\text{E}1 = 6.4\text{E}1 = 64 \\ x_2 &= 3.2\text{E}1 - 3.2\text{E}1 = 0 \end{aligned}$$

Mit der stabilisierten Formel berechnet man zuerst $x_1 = 6.4\text{E}1 = 64$ wie oben. Anschliessend erhält man

$$x_2 = q/x_1 = 1.0\text{E}0/6.4\text{E}1 = 0.015625 \overset{\text{gerundet}}{=} 1.6\text{E}{-2},$$

was besser mit dem exakten Wert übereinstimmt.

b) Nach dem Cosinussatz berechnet sich die Seite c durch

$$c = \sqrt{a^2 + b^2 - 2ab\cos\gamma}$$

Mit $a = 5.6$, $b = 5.7$ und $\gamma = 5°$ ergibt sich der exakte Wert $c = 0.50292\ldots$. Mit der definierten endlichen Arithmetik wird

$$\begin{aligned} a^2 &= 5.6^2 = 31.36 \overset{\text{gerundet}}{=} 31 \\ b^2 &= 5.7^2 = 32.49 \overset{\text{gerundet}}{=} 32 \\ 2a &= 11.2 \overset{\text{gerundet}}{=} 11 \\ (2a)b &= 62.7 \overset{\text{gerundet}}{=} 63 \\ (2ab)\cos\gamma &= 62.7602 \overset{\text{gerundet}}{=} 63 \\ c &= \sqrt{31 + 32 - 63} = 0, \end{aligned}$$

und wir erhalten ein ganz falsches Resultat. Schlimmer wäre es sogar, wenn direkt

$$2ab\cos\gamma = 63.597 \overset{\text{gerundet}}{=} 64$$

gerechnet würde, was eine imaginäre Seite $c = \sqrt{-1}$ zur Folge hätte ! Mit der stabilisierten Formel

$$c = \sqrt{(a-b)^2 + 4ab\left(\sin\frac{\gamma}{2}\right)^2}$$

wird

$$\begin{aligned}
a - b &= -0.1 \\
(a-b)^2 &= 0.01 \\
4a &= 22.4 \overset{\text{gerundet}}{=} 22 \\
(4a)b &= 125.4 \overset{\text{gerundet}}{=} 1.3\text{E}2 \\
\sin\frac{\gamma}{2} &= 0.043619 \overset{\text{gerundet}}{=} 4.4\text{E}{-2} \\
\left(sin\frac{\gamma}{2}\right)^2 &= 0.001936 \overset{\text{gerundet}}{=} 1.9\text{E}{-3} \\
4ab\left(sin\frac{\gamma}{2}\right)^2 &= 0.247 \overset{\text{gerundet}}{=} 2.5\text{E}{-1} \\
(a-b)^2 + 4ab\left(sin\frac{\gamma}{2}\right)^2 &= 0.26 = 2.6\text{E}{-1} \\
c = \sqrt{(a-b)^2 + 4ab\left(sin\frac{\gamma}{2}\right)^2} &= 0.509901 \overset{\text{gerundet}}{=} 5.1\text{E}{-1},
\end{aligned}$$

was sehr gut mit dem exakten Resultat übereinstimmt.

Aufgabe 1.4 *Man schreibe ein Programm, mit welchem die Maschinenkonstanten $\tilde{\varepsilon}$, m und M berechnet werden.*

Während die Konstante m leicht erhältlich ist (man dividiert solange durch 2 bis wegen Unterlauf das Resultat 0 wird), kann M nicht auf analoge Weise berechnet werden, da bei Überlauf auf dem Computer die Rechnung mit einer Fehlermeldung abgebrochen wird. Nimmt man diesen unschönen Abbruch in Kauf, so muss nach jeder Verdoppelung der Wert ausgedruckt werden, damit man neben der Fehlermeldung 'Überlauf' den zuletzt berechneten Wert vom Bildschirm ablesen kann.

Man kann M näherungsweise aus der Beziehung $M \approx 1/m$ zu rechnen versuchen. Häufig ist der Zahlenbereich aber nicht symmetrisch, so dass dennoch dabei ein Überlauf auftreten kann. Dies ist z.B. in TURBO PASCAL der Fall und man muss daher $M = 1/(\alpha m)$ setzen. Ein paar Experimente zeigen, dass das kleinste α in TURBO PASCAL $\alpha = 2.00000000001$ ist.

```
(*$B- Aufgabe 1.4*)
program machconst;
var h, eps, km, gm : real;
begin
   eps := 1;
   while 1 + eps <> 1 do eps := eps/2;
   writeln('eps = ',2 * eps);
   h := 1;
   repeat
      km := h; h := h/2
   until h = 0;
   writeln('m =', km, ' M = ', 1/2.00000000001/km);
end.
```

Dieses Programm liefert den Output

```
eps = 1.8189894035E-12
m = 2.9387358771E-39 M = 1.7014118346E+38
```

Kapitel 2

Elementare Algorithmen

Aufgabe 2.1 *Man schreibe ein Programm, das narrensicher die quadratische Gleichung*

$$ax^2 + bx + c = 0 \tag{2.1}$$

löst. Man beachte auch die Spezialfälle, wenn die Koeffizienten null sind.

Die Probleme, die hier auftreten können, werden am besten an einem Beispiel gezeigt:

$$10^{-10}x^2 + 10^{-20}x + 10^{20} = 0 \tag{2.2}$$

Wir machen davon Gebrauch, dass die Lösungen der Gleichung (2.1) bzw. (2.2) nicht geändert werden, wenn man die ganze Gleichung durch einen Faktor dividiert. Man normiert die Koeffizienten nun so, dass sie durch $max = \max\{|a|, |b|, |c|\}$ dividiert werden. Dabei können kleine Koeffizienten durch Unterlauf zu null werden. Bei der Gleichung (2.2) wird $max = 10^{20}$ und die Division ergibt

$$10^{-30}x^2 + 10^{-40}x + 1 = 0.$$

Wegen des Unterlaufbereichs von TURBO PASCAL ist $b = 10^{-40} = 0$ und damit

$$10^{-30}x^2 + 1 = 0 \tag{2.3}$$

numerisch aequivalent zu Gleichung (2.2). Nach der Prüfung, ob die Gleichung nicht entartet ist, d.h. ob $a = 0$ wird oder sogar $a = 0$ und $b = 0$ werden, kann man die Gleichung durch den Koeffizienten a dividieren

$$x^2 + 10^{30} = 0$$

und anschliessend den Algorithmus 2.1 benützen. Man erhält damit das Programm:

```
(*$B- Aufgabe2.1*)
program qgleich;
type zustand=(beliebig, keine, eine, komplex, doppel, reell);
var a, b, c, x1, x2 : real;
    info : zustand;

procedure quad(a, b, c : real; var x1, x2 : real; var info: zustand);
var p, q, diskr, fak, max : real;
begin
    max := abs(a);
    if abs(b) > max then max := abs(b);
    if abs(c) > max then max := abs(c);
    if max = 0 then info:= beliebig
    else
    begin
        a := a/max; b := b/max; c := c/max;
        if a = 0 then
        if b = 0 then info:= keine
        else begin info:= eine; x1 := -c/b end
        else
        begin
            p := b/a; q := c/a;
            if abs(p) > 1 then
            begin fak := abs(p); diskr := 0.25 - q/p/p end
            else
            begin fak := 1; diskr := sqr(p/2) - q end ;
            if diskr < 0 then
            begin
                info:= komplex; x1 := -p/2; x2 := fak * sqrt(-diskr)
            end
            else
            begin
                x1 := abs(p/2) + fak * sqrt(diskr);
                if p > 0 then x1 := -x1;
                if x1 = 0 then x2 := 0 else x2 := q/x1;
                if diskr = 0 then info:= doppel
                else info:= reell
            end
        end
    end
end ;

begin
    repeat
```

```
    writeln('Koeffizienten a,b,c?'); read(a, b, c);
    writeln(a,' x **2 + ',b,' x + ',c,' = 0 hat ');
    quad(a, b, c, x1, x2,info);
    case info of
      keine:    writeln('keine Loesung');
      eine:     writeln('die Loesung x=',x1);
      beliebig: writeln('jedes x als Loesung');
      reell:    writeln('die beiden Loesungen ', x1,' und ',x2);
      doppel:   writeln('die Doppelloesung x1 = x2 =',x1);
      komplex: writeln('die komplexen Loesungen ',x1,'+- i* ',x2)
    end ;
  until eof
end.
```

Aufgabe 2.2 *Wenn $a >> b$ ist, arbeitet der Algorithmus 2.2 nicht sehr effizient, da sehr viele Subtraktionen $a-b$ auftreten. Man kann den Rest viel schneller durch eine Division bestimmen. Man schreibe eine entsprechend verbesserte Funktion für den ggT.*

Anstelle der Subtraktionen berechnet man den Rest durch die Anweisung $a := a$ **mod** b. Man muss darauf achten, dass $a \geq b$ ist. Im folgenden Algorithmus werden dazu bei Bedarf die Variabeln a und b vertauscht. Beim Durchlaufen der **repeat**–Schleife wird alternierend vertauscht und ein neuer Rest gerechnet. Durch Einfügen der Druckanweisung $writeln(a, b)$ vor dem **until** kann der Ablauf des Algorithmus durchsichtig gemacht werden.

```
(*$B- Aufgabe 2.2*)
program gemteiler;
type longint = integer;
var a,b : longint;
function ggt(a,b : integer) : integer;
var h : integer;
begin
  repeat
    if a < b then
    begin h := a; a := b; b := h end
    else a := a mod b;
  until b = 0;
  ggt := a
end ;
begin
  writeln('Berechnung des ggT: a,b eingeben'); read(a,b);
```

```
    writeln('ggT(',a,',',b,') = ',ggt(a,b))
end.
```

Aufgabe 2.3 *Man schreibe eine Prozedur, um Brüche zu kürzen.*

Um einen Bruch darzustellen verwenden wir wegen der besseren Leserlichkeit die Datenstruktur

```
type longint = integer;
   var bruch = record zaehler, nenner : longint end
```

Bei einigen PASCAL Implementationen ist es möglich mit grossen Integerzahlen zu rechnen (z.B. existiert im UCSD PASCAL ein Typus *integer*[36] von ganzzahligen Variabeln mit bis zu 36 Dezimalstellen). Deswegen führen wir den Typus *longint* ein, der bei diesen Gelegenheiten entsprechend deklariert werden sollte. Aus dem Programm der Aufgabe 2.2 machen wir eine Prozedur um Brüche zu kürzen:

```
procedure kuerzen(var res: bruch);
var a, b, h : longint;
begin
   a := abs(res.zaehler); b := abs(res.nenner) ;
   repeat
      if a < b then
      begin h := a; a := b; b := h end
      else a := a mod b;
   until b = 0;
   if res.nenner < 0 then a := -a;
   res.zaehler := res.zaehler div a;
   res.nenner := res.nenner div a
end
```

Ferner möchten wir den zu kürzenden Bruch als Folge von Characters wie etwa 176/36 eingeben. Dazu benötigen wir eine Prozedur *liesbruch*, welche Zähler und Nenner des Bruches unter Verwendung einer Prozedur *lieszahl* aus der Folge von Characters aufbaut. In der ersten **repeat** Schleife werden alle Zeichen überlesen, die nicht Ziffern sind und man merkt sich ein eventuell auftretendes Vorzeichen. Die zweite **repeat** Schleife baut dann aus den Ziffern die Zahl auf.

```
procedure liesbruch(var a : bruch; var ch : char);

procedure lieszahl(var x : longint; var c : char);
var ziffer : boolean; vor : integer;
begin
```

```
    vor := 1;
    repeat
        read(c);
        if c ='-' then vor := -vor;
        ziffer := ('0'<= c) and (c <='9');
    until ziffer;
    x := 0;
    repeat
        x := x * 10 + ord(c) - ord('0');
        read (c); ziffer := ('0'<= c) and (c <='9');
    until not ziffer;
    x := x * vor
end ;
begin
    lieszahl(a.zaehler, ch);
    lieszahl(a.nenner, ch)
end ;
```

Das nachfolgende Hauptprogramm wird damit ganz kurz. Die Prozedur *schreibbruch* dient dazu die Zahl ohne Bruchstrich zu schreiben, falls der Nenner 1 oder der Zähler 0 ist.

```
(*$B- Aufgabe2.3*)
program bruchkuerzen;
type longint = integer;
     bruch = record
                 zaehler, nenner : longint
             end ;
var a, b, res : bruch;
    ch, op : char;
(*$I kuerzen *)
(*$I liesbruch *)
procedure schreibbruch(res: bruch);
begin
    if (res.nenner = 1) or (res.zaehler = 0) then
        write( res.zaehler)
    else write( res.zaehler,'/',res.nenner )
end ;
begin
    writeln('Bruch eingeben ');
    liesbruch(a, ch); kuerzen(a);
    write('gekuerzter Bruch =');
    schreibbruch(a)
```

end.

Aufgabe 2.4 *Man schreibe ein Programm, das die vier Bruchoperationen* $\{+, -, \times, /\}$ *exakt ausführt. Der Input lautet etwa*

$$4/7 + 12/37 =$$

und als Output soll das Resultat $232/259$ *ausgedruckt werden.*

Für dieses Programm verwenden wir dieselbe Datenstruktur wie in Aufgabe 2.3 und übernehmen auch die Prozeduren *liesbruch* und *kürzen*. Da wir den Output nicht nur auf dem Bildschirm haben wollen, ergänzen wir die Prozedur *schreibbruch* durch eine File-Variable. Neu dazu kommen Prozeduren für die Bruchoperationen. Es ist wichtig, dass nach jeder Operation das Resultat gekürzt wird, damit Zähler und Nenner möglichst nicht aus dem Integerbereich überlaufen. Im TURBO PASCAL besteht dieser Bereich nur aus dem Intervall $[-32'768, 32'767]$ und man stösst leider damit sehr schnell an die Grenze, die nicht als Fehlermeldung angezeigt wird.

In UCSD PASCAL gibt es ganzzahlige Variabeln mit 36 Dezimalstellen. Durch die Deklaration **type** *longint* = *integer*[36] kann dieser grosse Bereich ausgenützt werden. Allerdings gibt es für diese Zahlen die **mod** – Operation nicht, man muss sie durch

$$a \ \mathbf{mod}\ b = a - b * (a \ \mathbf{div}\ b)$$

umschreiben.

Die folgenden Prozeduren für die Grundoperationen mit Brüchen sind selbsterklärend, sie werden als Include-File unter dem Namen *bruchop* in das Hauptprogramm geladen.

```
procedure add(a, b : bruch; var res : bruch);
begin
    res.zaehler := a.zaehler * b.nenner + b.zaehler * a.nenner;
    res.nenner := a.nenner * b.nenner;
    kuerzen(res)
end ;
procedure sub(a, b : bruch; var res : bruch);
begin
    res.zaehler := a.zaehler * b.nenner − b.zaehler * a.nenner;
    res.nenner := a.nenner * b.nenner;
    kuerzen(res)
end ;
procedure mal(a, b : bruch; var res : bruch);
```

```
begin
    res.zaehler := a.zaehler * b.zaehler;
    res.nenner := a.nenner * b.nenner;
    kuerzen(res)
end ;
procedure teil(a, b : bruch; var res : bruch);
begin
    res.zaehler := a.zaehler * b.nenner;
    res.nenner := a.nenner * b.zaehler;
    kuerzen(res)
end ;
procedure gemzahl(a : bruch; var c : longint; var d : bruch);
begin
    c := a.zaehler div a.nenner;
    d.zaehler := a.zaehler mod a.nenner;
    d.nenner := a.nenner
end ;
```

Wie schon erwähnt, wird in TURBO PASCAL ein *integer*-Überlauf nicht als Fehler angezeigt. Eine Rechnung kann daher mit einem falschen Resultat beendet werden, das vom einem unkritischen Benützer des Programms nicht unbedingt erkannt wird. In TURBO PASCAL wird aber ein Programm mit einer Fehlermeldung abgebrochen, wenn das Argument der **function** *round* ausserhalb des *integer* Bereichs liegt. Man kann dies ausnützen, um den Überlauf zu entdecken. Der *integer* Ausdruck muss dabei *real* gemacht werden (etwa durch Multiplikation mit der *real* Zahl 1.0) und danach mittels *round* gerundet werden. Als Beispiel zeigen wir eine so umgeformte Prozedur:

```
procedure add(a, b : bruch; var res : bruch);
begin
    res.zaehler := round(a.zaehler * 1.0 * b.nenner + b.zaehler * a.nenner);
    res.nenner := round(a.nenner * 1.0 * b.nenner);
    kuerzen(res)
end ;
```

Solche 'Tricks' gehören aber zu schlechtem Programmierstil und sollten deshalb nicht gepflegt werden. Die richtige Abhilfe wäre hier eindeutig eine Anpassung des TURBO PASCAL Compilers.

Um Brüche auszuschreiben, verwenden wir eine leicht geänderte Prozedur *schreibbruch*: es wird auf das Textfile *tf* geschrieben und etwas Abstand nach dem Bruch eingeschoben.

```
procedure schreibbruch(res: bruch);
begin
```

```
    if (res.nenner = 1) or (res.zaehler = 0) then
        write(tf,' ',res.zaehler,'    ')
    else write(tf,res.zaehler,'/',res.nenner,'    ')
end ;
```

Das Hauptprogramm hat dieselbe Struktur wie das nachfolgende Taschenrechnerprogramm: es wird jeweils ein Operationszeichen und ein Operand eingelesen und mit dem zuletzt berechneten Resultat verrechnet. Es wird abgebrochen, falls das Operationszeichen ein Gleichheitszeichen ist. Bei diesem Vorgehen werden die Prioritäten der Operatoren nicht berücksichtigt.

```
(*$B- Taschenrechner *)
program taschenrechner;
var op : char; res,zahl : real;
begin
    writeln('Rechnung eingeben:');
    writeln('Zahlen und Operatoren mit Leerschlag trennen');
    res := 0; op :='+';
    repeat
        read(zahl);
        case op of
            '+'   : res := res + zahl;
            '-'   : res := res - zahl;
            '*'   : res := res * zahl;
            ':','/': res := res/zahl
        end ;
        repeat read(op) until op <>' '
    until op ='=';
    writeln(res)
end.
```

Das nachfolgende Hauptprogramm für die Bruchoperationen ist nun leicht zu lesen.

```
(*$B- Aufgabe2.4*)
program bruchrechnen;
type longint = integer;
    bruch = record
                zaehler,nenner : longint
            end ;
var a,b,res: bruch; x : longint; ch,op: char;
tf: text; stri: string [20];
(*$I kuerzen ( siehe Aufgabe2.4)*)
(*$I liesbruch ( siehe Aufgabe2.4)*)
(*$I schreibbruch *)
```

```
(*$I bruchop *)
begin
  writeln('Wohin mit dem Output?');
  readln(stri); assign(tf,stri); rewrite(tf);
  repeat
    writeln('Rechnung eingeben'); writeln;
    res.zaehler := 0; res.nenner := 1;
    op :='+'; write(tf,'   ');
    repeat
      liesbruch(a,ch);
      if (stri <>'CON:') and (stri <>'con:') then schreibbruch(a);
      writeln(tf); kuerzen(a);
      case op of
        '+'   : add(res,a,res);
        '-'   : sub(res,a,res);
        '*'   : mal(res,a,res);
        '/',':' : teil(res,a,res);
      end ;
      while not ( ch in ['*','/','+','-',':','='] ) do read(ch);
      op := ch;
      if (stri <>'CON:') and (stri <>'con:') then
      if op in ['/',':'] then write(tf,' : ') else write(tf,' ',op,' ')
    until op = '=';
    schreibbruch(res); gemzahl(res,x,res);
    if res.zaehler < 0 then op :=' ' else op :='+';
    if (x <> 0) and (res.zaehler <> 0) then
    writeln(tf,' = ',x,' ',op,' ',res.zaehler,'/',res.nenner);
    writeln(tf); writeln(tf);
    writeln('weiterfahren (RET) fertig (CTRL-Z) '); readln
  until eof(input); close(tf)
end.
```

Aufgabe 2.5 *Man schreibe ein Programm, das alle Primzahlen, die kleiner als 1000 sind, berechnet und ausdruckt.*

Im nachfolgenden Programm berechnen wir die Primzahlen, indem wir alle ungeraden Zahlen betrachten und versuchen, diese durch schon gefundene Primzahlen zu teilen. Dazu müssen die gefundenen Primzahlen in einem **array** $p[1..300]$ abgespeichert werden. Wenn die Primzahlen bis zur Zahl n berechnet werden sollen, kann man mittels des Primzahlensatzes

$$(\text{Anzahl Primzahlen } \leq n) \sim \frac{n}{\ln n}$$

die Grösse des **array** 's p abschätzen. Bei der Prüfung, ob j eine Primzahl sei, genügt es, *durch die schon gefundenen Primzahlen bis* $\sqrt{j}$ zu teilen.

```
(*$B- Aufgabe 2.5A*)
program primzahlen;
var i,j,k,n,logn: integer; gehtauf: boolean;
    p : array [1..300] of integer;
    tf : text; stri : string[20];
function log(x : real) : real; begin log := ln(x)/ln(10) end ;
begin
    write('Wohin mit der Ausgabe?');
    readln(stri); assign(tf, stri); rewrite(tf);
    writeln('Bis wohin Primzahlen rechnen ? (n <= 1000)');
    read (n); i := 1; j := 1; p[j] := 2; logn:= round(log(n)) + 2;
    repeat
        i := i + 2; k := 0;
        repeat
            k := k + 1; gehtauf:= (i mod p[k]) = 0;
        until gehtauf or (sqr(p[k]) >= i);
        if not gehtauf then begin j := j + 1; p[j] := i end ;
    until (i >= n);
    for k := 1 to j do
    if (k mod 10) = 0 then writeln(tf, p[k] : logn)
    else write(tf, p[k] : logn);
    close(tf);
end.
```

Eine ganz andere Möglichkeit, Primzahlen zu berechnen ist das *Sieb des Erastostenes*. Hier wird nicht durch einen Teilbarkeitstest entschieden, ob eine Primzahl vorliegt, sondern es werden in der Liste der Zahlen {2, 3, ..., n} die Vielfachen von Primzahlen gestrichen, sodass nur die Primzahlen übrigbleiben. Diese Methode erfordert einen kleineren Rechen– dafür aber einen grösseren Speicheraufwand als das Programm *primzahlen*. Man verwendet einen **array** $p[1..n]$ **of** *boolean*, der angibt, ob i prim ist ($p[i] =$ *true*).

```
(*$B- Aufgabe 2.5B*)
program sieb ;
var i,j,n,k,logn : integer;
    p : array [1..1000] of boolean;
    tf : text; stri : string [20];
function log(x :real) : real; begin log := ln(x)/ln(10) end ;
begin
    writeln('Wohin mit der Ausgabe');
```

```
   readln(stri); assign(tf,stri); rewrite(tf);
   writeln ('Bis wohin Primzahlen rechnen');
   read (n); logn := round(log(n)) + 2; k := 0;
   for i := 2 to n do p[i] := true; i := 2;
   repeat
      if p[i] then
      begin
         write (tf, i : logn); k := k + 1;
         if k mod 10 = 0 then writeln(tf);
         j := 2 * i;
         while j <= n do begin p[j] := false; j := j + i end ;
      end ;
      i := i + 1;
   until i = n;
   close(tf)
end.
```

Aufgabe 2.6 *Man schreibe ein Programm, welches die Primfaktorenzerlegung einer Zahl berechnet und ausdruckt.*

Diese Aufgabe kann einfach durch ein rekursives Programm gelöst werden. Wenn man durch einen Algorithmus einen Faktor j der Zahl i bestimmt hat, d.h. wenn

$$i = j \times (i \textbf{ div } j)$$

gilt, so muss derselbe Algorithmus auf die beiden Faktoren j und $(i$ **div** $j)$ angewendet werden bis man nicht mehr faktorisieren kann.

Die Prozedur *zerlege* im nachfolgenden Programm ist rekursiv und ruft sich selbst wieder auf für die Zerlegung der beiden Faktoren von i. Wie bei Aufgabe 2.5 wird versucht, einen Teiler von i zu finden, indem durch $j = round(\sqrt{i}), \ldots, 3, 2$ geteilt wird.

```
(*$B- Aufgabe2.6*)
program primfaktoren;
var i : integer;
procedure zerlege(i : integer);
var j,rest : integer;
begin
   j := round(sqrt(i)) + 1 ;
   repeat
      j := j - 1 ; rest := i mod j
   until ( rest = 0) or (j = 1);
   if j > 1 then begin zerlege(i div j); zerlege(j) end
```

```
   else write(i,'*')
end ;
begin
   writeln('ganze Zahl eingeben'); read(i);
   write(' = '); zerlege(i); writeln('1')
end.
```

Aufgabe 2.7 *Die goniometrische Gleichung*

$$a \sin x + b \cos x = c \tag{2.5*}$$

kann nach der Hilfswinkelmethode *wie folgt gelöst werden: Man berechnet die Polarkoordinaten des Punktes* (a, b)

$$a = r \cos \varphi$$
$$b = r \sin \varphi$$

und setzt die Ausdrücke in die Gleichung $(2.5*)$ *ein*

$$r \cos \varphi \sin x + r \sin \varphi \cos x = c.$$

Nach der Division durch r *und wegen des Additionstheorems für die trigonometrischen Funktionen erhält man die Gleichung (*φ *ist der Hilfswinkel)*

$$\sin(x + \varphi) = \frac{c}{r}, \tag{2.6*}$$

welche nun leicht gelöst werden kann.

Man schreibe ein Programm, welches nach dieser Methode alle Lösungen der Gleichung $(2.5*)$ *berechnet. Da in PASCAL die* arcsin*-Funktion nicht vorhanden ist, forme man die Gleichung* $(2.6*)$ *so um, dass die* arctan *- Funktion benützt werden kann.*

Aus der Gleichung $y = \arcsin x$ folgt

$$\sin y = x \quad \Rightarrow \cos y = \sqrt{1 - x^2} \quad \Rightarrow \tan y = \frac{x}{\sqrt{1 - x^2}}$$

und damit erhalten wir für y den Ausdruck

$$y = \arctan \frac{x}{\sqrt{1 - x^2}},$$

der für die Lösung der Gleichung $(2.6*)$ benützt wird. Für $x \approx 1$ tritt Auslöschung auf. Wir verzichten auf einen Versuch, sie zu vermeiden. Sie stört hier nicht.

```
(*$B- Aufgabe 2.7*)
program hiwi;
var a,b,c,x1,x2,r,phi,pi: real;
(*$I topolar Algorithmus 2.3 *)
function arcsin(x : real) : real;
var y : real;
begin
   if abs(x) > 1 then
      writeln('Argument des arcsin ausserhalb des Definitionsbereichs')
   else
      if abs(x) = 1 then y := x* arctan(1) * 2
      else y := arctan(x/sqrt(1 - sqr(x)));
   arcsin := y
end ;
begin
   pi := 4 * arctan(1);
   writeln('Hilfswinkelmethode');
   writeln('a * sinx + b * cosx = c ');
   writeln('a,b,c eingeben');
   read(a, b, c);
   topolar(a, b, r, phi);
   if abs(c) > abs(r) then writeln('keine Loesung')
   else
   begin
      x1 := arcsin(c/r); x2 := pi - x1 - phi; x1 := x1 - phi;
      writeln('Loesungen:'); writeln(x1,' , ', x2,' +k* ',2 * pi)
   end ;
   writeln('Kontrolle');
   writeln(a * sin(x1) + b * cos(x1) - c);
   writeln(a * sin(x2) + b * cos(x2) - c);
end.
```

Aufgabe 2.8 *Man schreibe eine Funktionsprozedur, um das skalare Produkt*

$$s = \mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i$$

der beiden Vektoren $\mathbf{x}$ *und* $\mathbf{y}$ *zu berechnen.*

Man wende die Funktionsprozedur an, um den Winkel zwischen den Vektoren $\mathbf{x}$ *und* $\mathbf{y}$ *nach der Formel*

$$\cos\alpha = \frac{\mathbf{x}^T\mathbf{y}}{\sqrt{\mathbf{x}^T\mathbf{x}}\sqrt{\mathbf{y}^T\mathbf{y}}}$$

zu berechnen.

Analog wie in Aufgabe 2.7 muss man diesmal die arccos Funktion durch den arctan ersetzen. Aus $y = \arccos x$ folgt $\cos y = x$, $\sin y = \sqrt{1-x^2}$ und damit

$$y = arctan\frac{\sqrt{1-x^2}}{x}$$

```
(*$B- Aufgabe 2.8*)
program skalarprodukt;
const nn = 10;
type vektor = array [1..nn] of real;
var n,i : integer; x,y : vektor; alpha : real;
function arccos(x : real) : real;
begin
   if x = 0 then arccos := arctan(1) * 2
   else arccos := arctan(sqrt(1 - x * x)/x)
end ;
function skalprod(n : integer; x,y : vektor) : real;
var i : integer; s : real;
begin
   s := 0;
   for i := 1 to n do s := s + x[i] * y[i];
   skalprod := s
end ;
begin
   writeln('Skalarprodukt von 2 Vektoren. Dimension n=?');
   read(n); writeln;
   write('Vektor x eingeben '); for i := 1 to n do read(x[i]); writeln;
   write('Vektor y eingeben '); for i := 1 to n do read(y[i]); writeln;
   alpha := arccos(skalprod(n,x,y)/sqrt(skalprod(n,x,x)*skalprod(n,y,y)));
   writeln('Der Zwischenwinkel betraegt :', alpha)
end.
```

Aufgabe 2.9 *Es ist*

$$\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}. \qquad (2.14*)$$

Man prüfe diese Gleichung nach, indem die ersten 10'000 Summanden der Reihe addiert werden. Bei der Reihe

$$\sum_{i=1}^{\infty} \frac{1}{i^3}$$

ist der Wert analytisch nicht bekannt. Man berechne auch hier die Summe der ersten 10'000 Summanden.

Hier das Programm sehr einfach. Wir wollen daran einige theoretische Überlegungen anstellen. Aus der Euler–Mac Laurinschen Summenformel (s. Gleichung (7.32*)) ergibt sich für $h = 1$ und $f(x) = 1/x^m$ die asymptotische Entwicklung

$$\sum_{i=1}^{n} \frac{1}{i^m} = \zeta(m) - \frac{\frac{1}{2}}{n^m} - \frac{1}{m-1} \sum_{k=0}^{\infty} \binom{1-m}{2k} B_{2k} \frac{1}{n^{m-1+2k}}, \qquad (2.4)$$

wobei B_{2k} die Bernoullizahlen sind [vgl. Gleichung (7.34∗)]. Insbesondere ergibt sich für $m = 2$

$$\sum_{i=1}^{n} \frac{1}{i^2} = \frac{\pi^2}{6} - \frac{B_0}{n} - \frac{1}{2n^2} - \frac{B_2}{n^3} - \frac{B_4}{n^5} - \frac{B_6}{n^7} - \dots$$

Aus der letzten Gleichung kann man sehen, dass wenn 10'000 Summanden aufsummiert werden, der Fehler $\approx \frac{B_0}{10'000}$ ist. Man erhält daher nur ca. 4 Stellen der Summe der unendlichen Reihe. Besser konvergiert die zweite Reihe. Hier folgt aus Gleichung (2.4), dass der Fehler $\approx \frac{1}{n^2}$ ist, sodass ca. 8 Dezimalstellen richtig sein dürften.

```
(*$B-  Aufgabe 2.9*)
program reihe;
var i : integer; s : real;
begin
   s := 0 ;
   for i := 1 to 10000 do s := s + 1/i/i ;
   writeln('1. Reihe =', s, ' sollte gleich ', sqr(4 * arctan(1))/6, ' sein');
   s := 0;
   for i := 1 to 10000 do s := s + 1/i/i/i ;
   writeln('2. Reihe = ', s)
end.
```

Als Output des Programms *reihe* erhält man

```
1.  Reihe = 1.6448340629 sollte gleich 1.6449340668 sein
2.  Reihe = 1.2020568887
```

Aufgabe 2.10 *Die Folge* $a_n = \ln(\cos\frac{1}{n})$ *lässt sich für* $n > 512$ *auf dem Taschenrechner nicht genau berechnen. Man entwickle die Funktion*

$$f(x) = \ln(\cos(x))$$

in die MacLaurinreihe und konstruiere damit eine einfache Formel für die Berechnung von a_n auf Taschenrechnergenauigkeit (ca. 10 Dezimalstellen).

Die Reihenentwicklung von $f(x)$ erhält man am einfachsten, wenn zunächst die Ableitung berechnet wird: $f'(x) = -\tan x$. Aus der bekannten Reihenentwicklung des $\tan x$ erhält man

$$f'(x) = -\left(x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \cdots\right).$$

Gliedweise Integration liefert

$$f(x) = C - \frac{x^2}{2} - \frac{x^4}{12} - \frac{2x^6}{90} - \cdots$$

Wegen $f(0) = \ln 1 = 0$, ist die Integrationskonstante $C = 0$. Für

$$x \leq \frac{1}{512} \quad \Rightarrow \frac{2x^6}{90} \leq 1.2336\text{E}{-}18,$$

so dass dieser und die nächsten Summanden für Taschenrechnergenauigkeit vernachlässigt werden können. Man erhält damit die Formel

$$\ln(\cos x) \simeq -\frac{x^2}{2}\left(1 + \frac{x^2}{6}\right) \text{ für } x \leq \frac{1}{512}.$$

Aufgabe 2.11 *Man schreibe ein Programm, welches das Argument α der Funktion $\sin\alpha$ auf das Intervall $[0°, 90°]$ reduziert. Beispiel: Für $\alpha = 1265°$, soll das Programm den Text*

$$\sin(1265 \text{ GRAD}) = -\sin(5 \text{ GRAD})$$

ausdrucken.

Im nachfolgenden Programm führen wir die Reduktion in mehreren Schritten durch. Zuerst merkt man sich das Vorzeichen von α. Anschliessend reduziert man α Modulo 360°: $\beta := \alpha$ **mod** 360. Nun liegt der Winkel β zwischen 0° und 360°. Wenn $\beta > 180°$ ist, kann die Beziehung $\sin(\beta) = -\sin(\beta - 180°)$ benützt werden, um β weiter auf das Intervall $[0°, 180°]$ zu reduzieren. Schliesslich benützt man noch die Beziehung $\sin(\beta) = \sin(180° - \beta)$, um den gewünschten Funktionswert im Intervall $[0°, 90°]$ zu berechnen.

```
(*$B- Aufgabe 2.11*)
program winkel;
var pi : real; alpha, beta, vorzeichen : integer;
begin
   pi := 4*arctan(1);
   writeln('Winkel im Gradmass eingeben');
   read(alpha); beta := alpha;
   if beta < 0 then
   begin vorzeichen:= -1; beta := - beta end
   else vorzeichen:= 1;
   beta := beta mod 360;
   if beta> 180 then
   begin beta := beta-180; vorzeichen:= -vorzeichen end ;
   if beta> 90 then beta:= 180-beta;
   write('SIN(',alpha,' GRAD) =');
   if vorzeichen= -1 then write('-');
   writeln('SIN(',beta,' GRAD)');
   writeln('Kontrolle:');
   write(sin(alpha * pi/180));
   if vorzeichen= -1 then
   write(-sin(beta * pi/180)) else write(sin(beta * pi/180))
end.
```

Aufgabe 2.12 *Man schreibe ein Programm zur Berechnung der Funktion* $\sin x$ *mittels der MacLaurinreihe. Da die Reihe alternierend ist, muss man das Argument* x *jeweils auf das Intervall* $[0, \frac{\pi}{2}]$ *(Bogenmass!) reduzieren. Man verwende dazu die Idee von Aufgabe 2.11*

Die MacLaurinreihe der Funktion $\sin x$ lautet:

$$\sin x = \sum_{j=0}^{\infty} (-1)^j \frac{x^{2j+1}}{(2j+1)!}. \tag{2.5}$$

Bezeichnet t_j den Summanden

$$t_j = (-1)^j \frac{x^{2j+1}}{(2j+1)!},$$

so lässt sich der nächste Summand t_{j+1} aus t_j rekursiv berechnen

$$t_j = -t_{j-1} \frac{x^2}{2j(2j+1)}.$$

Dies wird im nachfolgenden Programm in der Funktion *sinreihe* benützt. Da die Zählvariable $i = 2j$ vor der Berechnung von t_j erhöht wird ($i := i + 2$), lautet der Nenner $(i - 1) * i$. Wie beim Algorithmus 2.7 werden solange Summanden aufsummiert, bis sie numerisch nichts mehr zur Summe beitragen. Dadurch wird die Sinusfunktion auf Maschinengenauigkeit berechnet. Die Funktion *reduziere* entspricht genau dem Programm *winkel* von Aufgabe 2.11. Wegen der Verwendung der PASCAL-Funktion *trunc* ist der Anwendungsbereich durch die grösste darstellbare Integerzahl beschränkt, d.h. das nachfolgende Programm kann die Sinusfunktion nur für $|x| < 2 * \pi * 32767 \simeq 205'881$ berechnen.

```
(*$B- Aufgabe2.12*)
program sinus;
var x,f : real;
function reduziere( beta : real) : real;
var pi, zweipi : real; vorzeichen : integer;
begin
   pi := 4* arctan(1); zweipi := 2*pi;
   if beta < 0 then
      begin vorzeichen s:= -1; beta:= -beta end
   else vorzeichen := 1;
   beta := beta - trunc(beta/zweipi) * zweipi;
   if beta > pi then
   begin beta := beta - pi; vorzeichen:= -vorzeichen end ;
   if beta > pi/2 then beta := pi - beta;
   if vorzeichen= -1 then beta := - beta;
   reduziere := beta
end ;
function sinreihe(x : real) : real;
var s,salt, t : real; i : integer;
begin
   s := x; t := x ; i := 1;
   repeat
      salt := s; i := i + 2 ;
      t := -t * x/(i - 1) * x/i; s := salt + t
   until s = salt;
   sinreihe := s
end ;
begin
   repeat
      writeln('x =?'); read(x);
      x := reduziere(x); f := sinreihe(x);
      writeln('Reihe =',f,'  Fehler =',f - sin(x));
```

until *eof*
end.

Aufgabe 2.13 *Man berechne das Integral*

$$I = \int_0^{0.5} \frac{\sin x}{x}\, dx$$

indem der Integrand in seine MacLaurinreihe entwickelt und anschliessend die Reihe gliedweise integriert und aufsummiert wird.

Verwendet man die MacLaurinreihe der Sinusfunktion (s. Gleichung (2.5)) und dividiert diese durch x, so ergibt sich

$$\frac{\sin x}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} \pm \cdots$$

Gliedweise Integration liefert

$$\int_0^x \frac{\sin x}{x} dx = x - \frac{x^3}{3 \cdot 3!} + \frac{x^5}{5 \cdot 5!} - \frac{x^7}{7 \cdot 7!} \pm \cdots, \tag{2.6}$$

und der gesuchte Wert I ergibt sich für $x = 0.5$. Im nachfolgenden Programm kann die obere Grenze x eingegeben werden. Da die Reihe alternierend ist, darf x wegen numerischer Auslöschung nicht zu gross gewählt werden. Wiederum berechnet man soviele Summanden der Reihe (2.6), bis sie numerisch nichts mehr zur Summe beitragen können. Für $x = 0.5$ erhält man den Wert $I = 0.49310741804$.

```
(*$B- Aufgabe2.13*)
program integral;
var x : real;

function reihe(x : real) : real;
var s,salt, t : real; i : integer;
begin
    s := x; t := x ; i := 1;
    repeat
        salt := s; i := i + 2; t := -t * x/(i - 1) * x/i;
        s := salt + t/i
    until s = salt;
    reihe := s
```

```
end ;
begin
   repeat
      writeln('x =?'); read(x);
      writeln( reihe(x));
   until eof
end.
```

Aufgabe 2.14 *Man schreibe eine Funktionsprozedur, welche die MacLaurinreihe der Funktion* $\arctan x$ *berechnet und wende sie an, um* π *nach der Formel*

$$\pi = 24 \arctan \frac{1}{8} + 8 \arctan \frac{1}{57} + 4 \arctan \frac{1}{239} \qquad (2.15*)$$

zu berechnen.

Die Ableitung von $f(x) = \arctan(x)$ ist

$$f'(x) = \frac{1}{1+x^2}.$$

Setzt man in der geometrischen Reihe

$$\frac{1}{1-q} = 1 + q + q^2 + q^3 + \cdots$$

$q = -x^2$, so ist

$$f'(x) = 1 - x^2 + x^4 - x^6 \pm \cdots$$

Gliedweise Integration und Berücksichtigung, dass $f(0) = \arctan(0) = 0$, ergibt die gesuchte MacLaurinreihe von arctan:

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \pm \cdots \qquad (2.7)$$

Der Konvergenzradius der Reihe ist $r = 1$. Wir können wieder maschinenunabhängig aufsummieren, bis numerisch der neue Summand nichts mehr zur Summe beiträgt.

```
(*$B- Aufgabe2.14*)
program berechnungvonpi;
var x, pi : real;
function atan(x : real) : real;
var s,salt, t : real; i : integer;
begin
   s := x; t := x ; i := 1;
   repeat
      salt := s; i := i + 2 ; t := -t * x * x; s := salt + t/i
   until s = salt;
   atan := s
end ;
begin
   pi := 24 * atan(1/8) + 8 * atan(1/57) + 4 * atan(1/239);
   writeln( pi,' exakter Wert =', 4 * arctan(1))
end.
```

Aufgabe 2.15 *Die Binomialreihe ist*

$$(1+z)^\alpha = \sum_{i=0}^{\infty} \binom{\alpha}{i} z^i = \binom{\alpha}{0} + \binom{\alpha}{1} z + \binom{\alpha}{2} z^2 + \cdots, \qquad (2.16*)$$

wobei der verallgemeinerte Binomialkoeffizient durch

$$\binom{\alpha}{i} = \frac{\alpha(\alpha-1)\cdots(\alpha-i+1)}{i!} \qquad (2.8)$$

für beliebiges reelles α definiert ist. Man schreibe ein Programm zur Berechnung der Binomialreihe und wende es an, um das Integral

$$\int_0^{0.8} \frac{dx}{\sqrt[3]{1+x^2}} \qquad (2.9)$$

zu berechnen. Hinweis: Der Integrand lässt sich für $z = x^2$ und $\alpha = -\frac{1}{3}$ in eine Binomialreihe entwickeln!

Bei dieser Aufgabe hat sich im Lehrbuch ein Fehler eingeschlichen: die obere Grenze des Integrals (2.9) kann nicht 1 sein, da der Konvergenzradius der Binomialreihe $r = 1$ beträgt und die Reihe für das Integral nicht konvergiert. Aus der Beziehung

$$\binom{\alpha}{i} = \binom{\alpha}{i-1} \frac{\alpha - i + 1}{i}$$

erhält man eine Rekursionsformel für die Berechnung der Summanden der Binomialreihe. Damit ist

```
(*$B- Aufgabe2.15A*)
program binomialreihe;
var alpha,x : real;
function binreihe( alpha,x : real) : real;
var s,salt, t : real; i : integer;
begin
   s := 1; t := 1; i := 0;
   repeat
      salt := s; i := i + 1;
      t := t * x/i * (alpha - i + 1); s := salt + t
   until s = salt;
   binreihe := s
end ;
begin
   repeat
      writeln('(1 + x) * *alpha : alpha, x =?'); read(alpha, x);
      write(binreihe(alpha, x));
      writeln('   Kontrolle:', exp(alpha * ln(abs((1 + x)))));
   until eof
end.
```

Für $sign(\alpha) \neq sign(x)$ ist die Reihe für $|\alpha| >> 1$ numerischer Auslöschung unterworfen.

Zur Berechnung des Integrals integriert man die Binomialreihe gliedweise

$$\int_0^x (1+x^2)^\alpha \, dx = \int_0^x \left(1 + \binom{\alpha}{1} x^2 + \binom{\alpha}{2} x^4 + \cdots\right) dx$$
$$= x + \binom{\alpha}{1}\frac{x^3}{3} + \binom{\alpha}{2}\frac{x^5}{5} + \cdots$$

und erhält das Programm

```
(*$B- Aufgabe 2.15B*)
program integral;
var x : real;
function int(x : real) : real;
var s,salt, t, alpha,z : real; i : integer;
begin
   alpha := -1/3;
   z := sqr(x);
   s := x; t := x ; i := 0;
   repeat
```

```
        salt := s; i := i + 1 ; t := t * z/i * (alpha - i + 1);
        s := salt + t/(2 * i + 1)
    until s = salt;
    int := s
end ;
begin
    repeat
        write('obere Grenze ( muss < 1 sein)x = ?'); read(x);
        writeln(int(x));
    until eof
end.
```

Für $x = 0.8$ erhält man den Wert:

$$\int_0^{0.8} \frac{dx}{\sqrt[3]{1+x^2}} = 0.75398330456$$

Aufgabe 2.16 *Man schreibe ein Programm, welches die Funktion* $\ln x$ *berechnet und dabei nur die 4 Grundoperationen* $\{+, -, \times, /\}$ *verwendet.*

Hinweis: Man löse die Gleichung $e^y = x$ *mittels des Newtonverfahrens (Siehe Kap. 3) und berechne* e^y *mit dem Algorithmus 2.7.*

Wenn wir das Newtonverfahren auf die Gleichung $f(y) = e^y - x = 0$ anwenden, so berechnen wir die Folge $\{y_n\}$

$$\begin{aligned} y_{n+1} &= y_n - \frac{f(y_n)}{f'(y_n)} = y_n - \frac{e^{y_n} - x}{e^{y_n}} \\ &= y_n + xe^{-y_n} - 1, \end{aligned}$$

welche quadratisch gegen $y = \ln x$ konvergiert. Es bleibt nur noch die Wahl eines günstigen Startwertes. Dies kann durch Zählen der Dezimalziffern des Argumentes x geschehen. Es ist $e^5 \simeq 148$. Daher ergeben für $x > 1$ die Anweisungen

$$y := 0; \textbf{ while } z > 1 \textbf{ do begin } y := y + 5; \; z := z/150 \textbf{ end } ;$$

einen vernünftigen Startwert y, der *rechts* von der Lösung $\ln x$ liegt. Betrachtet man die Funktion $f(y) = e^y - x$, so ist es geometrisch klar, dass mit einem solchen Startwert die Newtonfolge *monoton fallend* gegen die gesuchte Nullstelle konvergiert. Numerisch kann natürlich $y_{n+1} < y_n$ nicht ständig gelten, weil es nur endlich viele Maschinenzahlen gibt. Es muss daher der Fall eintreten, dass $y >= yalt$ wird, und dies ist der Moment, wo die Lösung auf Maschinengenauigkeit berechnet ist.

Die Gleichung $e^y = x$ ist gleichbedeutend mit $e^{-y} = \frac{1}{x}$. Falls nun $x < 1$ ist, können wir die Gleichung $e^z = \frac{1}{x}$ nach z lösen und anschliessend daraus $y = -z$ erhalten. So brauchen wir nur den Fall $x \geq 1$ zu betrachten. Wir erhalten somit folgendes Programm, bei dem der Algorithmus 2.7 als Include-File zugeladen wird:

```
(*$B- Aufgabe 2.16*)
program logarithmus;
var x : real;
(*$I e Algorithmus 2.7 *)
function log(x : real) : real;
var y,yalt,z : real;
begin
   (* Berechnung des Startwertes durch Zaehlen der Ziffern *)
   if x > 1 then z := x else z := 1/x;
   y := 0; while z > 1 do begin y := y + 5; z := z/150 end ;
   if x > 1 then z := x else z := 1/x;
   repeat
      yalt := y; y := yalt + z * e(-yalt) - 1
   until yalt <= y ;
   if x < 1 then y := -y;
   log := y
end ;
begin
   repeat
      write('x = '); read(x);
      writeln('  ln = ',log(x),' exakt = ',ln(x))
   until eof
end.
```

Aufgabe 2.17 *Die Exponentialfunktion e^x lässt sich für reelles x durch die Reihe* (2.11∗)

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

berechnen. Es gilt auch

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

Man schreibe ein Programm, welches die Euler'sche Relation (2.18∗) *nachprüft. Das Programm soll für*

$$\varphi = 0, 0.2, 0.4, \ldots, 3.0$$

die folgende Tabelle berechnen und ausdrucken:

$$\varphi \quad \lim_{n\to\infty}\left(1+\frac{i\varphi}{n}\right)^n \quad \sum_{k=0}^{\infty}\frac{(i\varphi)^k}{k!} \quad \cos\varphi \quad \sin\varphi.$$

Die Folge $(1+i\varphi/n)^n$ konvergiert sehr langsam. Wir betrachten daher ihre Teilfolge

$$a_m = \left(1+\frac{(i\varphi)}{2^m}\right)^{2^m},$$

bei der das Glied a_m durch Quadrieren schnell wie folgt berechnet werden kann:

$z := 1 + i\varphi/2^m$;
for $k := 1$ **to** m **do** $z := sqr(z)$

Wir werden m solange erhöhen, bis für eine vorgegebene Fehlerschranke ε $|a_{m+1} - a_m| < \varepsilon$ gilt und dann a_{m+1} als 'Grenzwert' der Folge betrachten.

Bei der Reihe $\sum \frac{(i\varphi)^k}{k!}$ liefern jeweils die Summanden mit geradem Index einen Beitrag zum Realteil und jene mit ungeradem Index bilden den Imaginärteil der Summe. Da zudem die Vorzeichen alternieren, können wir die Summe am einfachsten mit einer **case** - Anweisung berechnen.

```
(*$B- Aufgabe 2.17*)
program ehochiphi;
var i : integer; phi,fre,fim : real;

procedure mult(ar,ai,br,bi : real; var cr,ci : real);
begin
  cr := ar*br - ai*bi; ci := ar*bi + ai*br
end ;

procedure folge(phi : real; var re, im : real);
var rea,ima, zhm : real;
  i,m : integer;
begin
  m := 1; zhm := 2; re := 1; im := 1;
  repeat
    rea := re; ima := im; re := 1; im := phi/zhm;
    for i := 1 to m do mult(re,im,re,im,re,im);
    m := m + 1; zhm := 2*zhm;
  until (abs(re - rea) <= abs(re) * 1e - 6) and
    (abs(im - ima) <= abs(im) * 1e - 6)
end ;

procedure reihe(phi : real; var re,im : real);
```

```
var k : integer; rea,ima, t : real;
begin
   k := 0; re := 1; im := 0; t := 1;
   repeat
      rea := re; ima := im; k := k + 1; t := t * phi/k;
      case k mod 4 of
         0 : re := re + t;
         1 : im:= im + t;
         2 : re := re - t;
         3 : im:= im - t
      end ;
   until (abs(re - rea) <= abs(re) * 1e - 6) and
      (abs(im - ima) <= abs(im) * 1e - 6)
end ;
begin
   writeln('phi    Folge',' ': 16,' Reihe',' ': 17,'cos(phi)   sin(phi)');
   for i := 0 to 15 do
   begin
      phi := i * 2/10; write(phi : 3 : 1,'   ');
      folge(phi, fre, fim);
      write(fre : 10 : 7, fim : 10 : 7,'   ');
      reihe(phi, fre, fim);
      write(fre : 10 : 7, fim : 10 : 7,'   ');
      writeln(cos(phi) : 10 : 7, sin(phi) : 10 : 7);
   end
end.
```

Aufgabe 2.18 *Man schreibe ein Programm, um eine der vier Grundoperationen mit zwei komplexen Zahlen durchzuführen. Man lese dabei zuerst Real- und Imaginärteil des ersten Operanden ein, dann ein Operationszeichen* $\{+, -, *, /\}$ *und schliesslich den zweiten Operanden.*

Wir übernehmen bei diesem Programm die Grundstruktur des Programms *taschenrechner* von Aufgabe 2.4. Es werden solange abwechslungsweise Operanden und Operationszeichen gelesen, bis ein Gleichheitszeichen eingegeben wird. Dieses bewirkt den Ausdruck des Resultates. Die Algorithmen *topolar* (Algorithmus 2.3), *csqrt* (Algorithmus 2.10) und *mult* (Algorithmus 2.8) werden als Include-Files dazugeladen. Neu kommen die Prozeduren *add, sub* und *teil* für die entsprechenden komplexen Operationen hinzu.

```
(*$B- Aufgabe 2.18*)
program komplexrechnen;
type komplex = record re,im : real end ;
```

```
var x,y,res,a : komplex; ch,op : char;
    tf : text; stri : string [20];
(*$I topolar Algorithmus 2.3*)
(*$I csqrt Algorithmus 2.10*)
(*$I mult Algorithmus 2.8*)
procedure add(a,b : komplex; var res : komplex);
begin
    res.re := a.re + b.re; res.im := a.im + b.im
end ;
procedure sub(a,b : komplex; var res : komplex);
begin
    res.re := a.re - b.re; res.im := a.im - b.im
end ;
procedure teil(a,b : komplex; var res : komplex);
var ra,rb,phia,phib,r,phi : real;
begin
    topolar( a.re,a.im,ra,phia); topolar( b.re,b.im,rb,phib);
    r := ra/rb; phi := phia - phib;
    res.re := r * cos(phi); res.im := r * sin(phi)
end ;
procedure schreibzahl( res : komplex);
begin
    write(tf,res.re : 10 : 5);
    if res.im >= 0 then write( tf,' +I* ')
    else write( tf,' -I* ');
    write(tf,abs(res.im) : 10 : 5)
end ;
begin
    writeln('Wohin mit dem Output');
    readln(stri); assign(tf,stri); rewrite(tf);
    repeat
        res.re := 0; res.im := 0;
        op := '+'; write( tf,'    ');
        repeat
            writeln('Real- und Imaginaerteil eingeben');
            readln(a.re,a.im);
            if stri <> 'CON:' then schreibzahl(a); writeln(tf);
            case op of
                '+'     : add(res,a,res);
                '-'     : sub(res,a,res);
                '*'     : mult(res,a,res);
```

```
            '/','*'    : teil(res, a, res);
        end ;
        writeln('Operationszeichen eingeben'); ch := ' ';
        while not ( ch in ['*','/','+','-',':','=']) do read(ch);
        op := ch;
        if stri <> 'CON:' then
        if op in [ '/',':'] then write( tf,' : ')
        else write( tf,' ',op,' ')
      until op = '=';
      schreibzahl(res);
      writeln(tf); writeln(tf);
      writeln('Weiterfahren (ret) fertig (CTRL-Z) '); readln
  until eof;
  close(tf)
end.
```

Aufgabe 2.19 *Man schreibe ein Programm, das die n-ten Wurzeln w_i einer komplexen Zahl berechnet und ausdruckt. Ferner berechne man als Kontrolle für jede Wurzel w_i^n.*

Wir berechnen die Wurzeln nach ihrer Definitionsgleichung (2.20∗). Man benötigt dazu die Prozedur *topolar* und für die Kontrolle den Algorithmus 2.8 *mult*.

```
(*$B- Aufgabe 2.19*)
program komplexewurzeln;
const nn = 50;
type komplex = record re,im : real end ;
   wurzeln = array [0..nn] of komplex;
var x,y : komplex; w : wurzeln; k,n : integer;
(*$I topolar Algorithmus 2.3*)
procedure wurzel(n : integer; a : komplex; var w : wurzeln);
var r,phi,zpin : real; k : integer;
begin
   zpin := 8* arctan(1)/n;
   topolar(a.re, a.im, r, phi);
   r := exp(ln(r)/n); phi := phi/n;
   for k := 0 to n - 1 do
   begin
      w[k].re := r * cos(phi + k * zpin);
      w[k].im := r * sin(phi + k * zpin)
   end
```

```
end ;
procedure hochn(x : komplex; var y : komplex);
var k : integer;
(*$I mult Algorithmus 2.8*)
begin
   y.re := 1; y.im := 0;
   for k := 1 to n do mult(y, x, y)
end ;
begin
   repeat
      writeln('Berechnung der n komplexen Wurzeln von a');
      writeln('n, Re(a) und Im(a) eingeben');
      read(n, x.re, x.im);
      wurzel(n, x, w);
      for k := 0 to n − 1 do
      begin
         write(k : 2,'.Wurzel =',w[k].re : 10 : 7,' +I* ',w[k].im : 10 : 7);
         write(' w**n = '); hochn(w[k], y);
         writeln(y.re : 10 : 7,' +I* ',y.im : 10 : 7);
      end ;
   until eof
end.
```

Aufgabe 2.20 *Die quadratische Gleichung*

$$az^2 + bz + c = 0 \qquad (2.21*)$$

mit den komplexen Koeffizienten a, b *und* c *hat die beiden Lösungen*

$$z_{1,2} = \frac{-b + \sqrt[2]{b^2 - 4ac}}{2a},$$

wobei unter der Wurzel die beiden komplexen Quadratwurzeln gemeint sind. Unter Verwendung der Prozedur csqrt schreibe man ein Programm für die Lösung von (2.21∗).

```
(*$B− Aufgabe2.20*)
program quadrgleichung;
type komplex = record re, im : real end ;
var a, b, c, wu, z1, z2, h, diskr : komplex;
(*$I topolar Algorithmus 2.3*)
(*$I csqrt Algorithmus 2.10*)
```

```
(*$I mult Algorithmus 2.8*)
procedure add(a,b : komplex; var res : komplex);
begin
   res.re := a.re + b.re; res.im := a.im + b.im
end ;
procedure sub(a,b : komplex; var res : komplex);
begin
   res.re := a.re - b.re; res.im := a.im - b.im
end ;
procedure teil(a,b : komplex; var res : komplex);
var ra, rb, phia, phib, r, phi : real;
begin
   topolar(a.re, a.im, ra, phia); topolar(b.re, b.im, rb, phib);
   r := ra/rb; phi := phia - phib;
   res.re := r * cos(phi); res.im := r * sin(phi)
end ;
procedure schreibzahl(res : komplex);
begin
   write(res.re);
   if res.im >= 0 then write(' +I*') else write(' -I*');
   write(abs(res.im))
end ;
begin
   repeat
      writeln('Loesen von quadratischen Gleichungen mit komplexen');
      writeln('Koeffizienten : ax**2 + b*x + c = 0 ');
      writeln(' Real- und Imaginaerteil von a,b und c eingeben');
      read(a.re,a.im,b.re,b.im,c.re,c.im);
      h.re := 4*a.re; h.im := 4*a.im;
      mult(h,c,h);
      mult(b,b,diskr); sub(diskr,h,diskr);
      csqrt(diskr,wu);
      h.re := 2*a.re; h.im := 2*a.im;
      sub(wu,b,z1); teil(z1,h,z1);
      wu.re := -wu.re; wu.im := -wu.im;
      sub(wu,b,z2); teil(z2,h,z2);
      write('1.Loesung ='); schreibzahl(z1); writeln;
      write('2.Loesung ='); schreibzahl(z2);
      writeln; writeln('Kontrolle');
      mult(a,z1,h); add(h,b,h); mult(h,z1,h); add(h,c,h);
      schreibzahl(h); writeln;
      mult(a,z2,h); add(h,b,h); mult(h,z2,h); add(h,c,h);
```

```
      schreibzahl(h); writeln; writeln
   until eof
end.
```

Aufgabe 2.21 *Man kann die komplexen Quadratwurzeln einer Zahl $a+ib$ auch ohne die Exponentialform zu verwenden durch den Ansatz*

$$(x+iy)^2 = a+ib$$

berechnen. Multipliziert man aus und vergleicht auf beiden Seiten Real- und Imaginärteil, so erhält man zwei reelle Gleichungen für die Unbekannten x und y. Man schreibe ein Programm, das die Wurzeln auf diese Weise berechnet und verwende es, um die quadratische Gleichung (2.21∗) *zu lösen.*

Multipliziert man den Ansatz aus und vergleicht Real- und Imaginärteil, so erhält man die Gleichungen:

$$\begin{aligned} x^2 - y^2 &= a \\ 2xy &= b. \end{aligned}$$

Löst man die zweite Gleichung nach x auf und setzt den Ausdruck in die erste ein, so ergibt sich eine biquadratische Gleichung für y:

$$4y^4 + 4ay^2 - b^2 = 0.$$

Man erhält

$$y^2 = \frac{-a \pm \sqrt{a^2+b^2}}{2},$$

also scheinbar 4 Lösungen für den Imaginärteil ?! Es sind aber nur 2 Lösungen, denn y ist eine reelle Zahl und wegen $|a| \leq \sqrt{a^2+b^2}$ kommt nur das '+' Zeichen vor der Wurzel in Frage. Wir erhalten somit

$$y = \pm\sqrt{\frac{-a+\sqrt{a^2+b^2}}{2}}. \tag{2.10}$$

Für $a > 0$ kann im Ausdruck (2.10) numerische Auslöschung auftreten und dadurch y ungenau berechnet werden. Erweitert man den Bruch mit $a + \sqrt{a^2+b^2}$, so wird

$$\frac{-a+\sqrt{a^2+b^2}}{2} = \frac{b^2}{2(a+\sqrt{a^2+b^2})}$$

und damit berechnet sich y durch

$$y = \begin{cases} \dfrac{|b|}{\sqrt{2(a+\sqrt{a^2+b^2})}} & \text{für } a > 0 \\ \sqrt{\dfrac{-a+\sqrt{a^2+b^2}}{2}} & \text{für } a < 0 \end{cases}$$

Den Realteil x erhält man aus $2xy = b$, wobei noch darauf geachtet werden muss, dass $y \neq 0$ ist. y wird 0, wenn b null oder sehr klein ist. Wenn aber $b = 0$ ist, so ist die komplexe Zahl, aus der wir die Wurzel berechnen wollen reell. Deshalb

```
if y = 0 then
   if a < 0 then begin x := 0; y := sqrt(-a) end
   else x := sqrt(a)
else x := b/2/y ;
```

Oft möchte man jene Quadratwurzel, bei der der Realteil ≥ 0 ist. Man kann dies durch Einfügen von

```
if x < 0 then begin x := -x; y := -y end
```

als letzte Anweisung erreichen. Das nachfolgende Programm ist lediglich ein Testprogramm, um die Prozedur *csqrt* zu prüfen. Um die Aufgabe zu lösen, ersetzt man damit den Algorithmus 2.10 in Aufgabe 2.20.

```
(*$B- Aufgabe 2.21*)
program komplexewurzel
type komplex = record re,im : real end ;
var x,y : komplex;
procedure csqrt(a : komplex; var x : komplex);
begin
   if a.re > 0 then
      x.im := abs(a.im)/sqrt(2 * (a.re + sqrt(sqr(a.re) + sqr(a.im))))
   else x.im := sqrt((-a.re + sqrt(sqr(a.re) + sqr(a.im)))/2);
   if x.im = 0 then
      if a.re < 0 then begin x.re := 0; x.im := sqrt(-a.re) end
                  else x.re := sqrt(a.re)
   else x.re := a.im/2/x.im;
end ;
(*$I mult Algorithmus 2.8*)
begin
```

```
  repeat
    writeln('Berechnung der Quadratwurzel aus a');
    writeln('Re(a) und Im(a) eingeben');
    read(x.re, x.im); csqrt(x, y);
    writeln('Wurzel = ',y.re,' +I* ',y.im);
    writeln('Kontrolle');
    mult(y, y, y); writeln(x.re − y.re, x.im − y.im)
  until eof
end.
```

Aufgabe 2.22 *Man schreibe Prozeduren, um die Matrixoperationen* $\mathbf{A}+\mathbf{B}$, $\mathbf{A}-\mathbf{B}$ *und* $\mathbf{AB}$ *durchzuführen.*

Um eine Matrix in PASCAL darzustellen, verwendet man üblicherweise die Datenstruktur

```
type matrix = array [1..nn, 1..nn] of real
```

wobei die Konstante nn genügend gross gewählt wird, damit der Platz für die auftretenden Matrizen ausreicht. Zur genauen Beschreibung einer Matrix gehören auch die *aktuellen* Dimensionen, m = Anzahl Zeilen und n = Anzahl Kolonnen, so dass als Datenstruktur besser der **record**

```
type matrix =  record
                   m,n : integer;
                   a : array [1..nn, 1..nn] of real
               end                                    (2.11)
```

verwendet wird. Der Nachteil der Darstellung (2.11) ist, dass bei der Verwendung der Matrixelemente eine etwas schwerfälligere Notation in Kauf genommen werden muss. Statt etwa $s := s + a[i,k] * b[k,j]$ muss man $s := s + a.a[i,k] * b.a[k,j]$ verwenden. Wir werden uns daher nicht stur auf einen Datentyp festlegen, sondern beide benützen. Bei dieser und der nächsten Aufgabe verwenden wir die **record** Variante.

Als erstes schreiben wir 5 kleine Prozeduren für Matrizenoperationen sowie für das Einlesen und Ausdrucken einer Matrix:

```
procedure matadd(a, b : matrix; var c : matrix);
var i,j : integer;
begin
  c.m := a.m; c.n := a.n;
  for i := 1 to c.m do for j := 1 to c.n do
  c.a[i, j] := a.a[i, j] + b.a[i, j]
```

```
end.

procedure matsub(a,b : matrix; var c : matrix);
var i,j : integer;
begin
   c.m := a.m; c.n := a.n;
   for i := 1 to c.m do for j := 1 to c.n do
   c.a[i,j] := a.a[i,j] - b.a[i,j]
end.

procedure matmult(a,b : matrix; var c : matrix);
var i,j,k : integer; s : real;
begin
   c.m := a.m; c.n := b.n;
   for i := 1 to c.m do for j := 1 to c.n do
   begin
      s := 0;
      for k := 1 to a.n do s := s + a.a[i,k] * b.a[k,j];
      c.a[i,j] := s
   end
end.

procedure eingabe( var a : matrix);
var i,j : integer;
begin
   writeln('Anzahl Zeilen der Matrix='); read(a.m);
   writeln('Anzahl Kolonnen der Matrix ='); read(a.n);
   writeln('Matrix zeilenweise eingeben');
   for i := 1 to a.m do for j := 1 to a.n do
   read(a.a[i,j])
end.

procedure matdruck( var a : matrix);
var i,j : integer;
begin
   for i := 1 to a.m do
   begin
      for j := 1 to a.n do write(a.a[i,j] : 11 : 5);
      writeln
   end ;
end.
```

Das Hauptprogramm, bei dem die obigen Prozeduren als Include-Files dazugeladen werden, wird damit ganz kurz:

```
(*$B- Aufgabe 2.22*)
```

```
program matrixoperationen;
const nn = 10;
type matrix =   record
                  m,n : integer;
                  a : array [1..nn, 1..nn] of real;
                end ;
var m,n,p : integer; op : char; a,b,c : matrix;
(*$I matmult *)
(*$I matadd *)
(*$I matsub *)
(*$I eingabe *)
(*$I matdruck *)
begin
  writeln('Erste Matrix eingeben');
  eingabe(a);
  writeln('Operationszeichen +, -, * eingeben');
  readln; readln(op);
  writeln('Zweite Matrix eingeben');
  eingabe(b);
  case op of
    '+' : matadd(a, b, c);
    '-' : matsub(a, b, c);
    '*' : matmult(a, b, c)
  end ;
  writeln('Resultat');
  matdruck(c)
end.
```

Aufgabe 2.23 [*Golub*] *Es seien* $\mathbf{A} = \mathbf{A}_r + i\mathbf{A}_i$ *und* $\mathbf{B} = \mathbf{B}_r + i\mathbf{B}_i$ *zwei komplexe Matrizen. Gesucht ist das Produkt*

$$\mathbf{C} = \mathbf{AB} \tag{2.24*}$$

Wenn man den Ansatz $\mathbf{C} = \mathbf{C}_r + i\mathbf{C}_i$ *macht, in* (2.24∗) *ausmultipliziert und Real- und Imaginärteil vergleicht, erhält man*

$$\begin{aligned} \mathbf{C}_r &= \mathbf{A}_r\mathbf{B}_r - \mathbf{A}_i\mathbf{B}_i \\ \mathbf{C}_i &= \mathbf{A}_r\mathbf{B}_i + \mathbf{A}_i\mathbf{B}_r \end{aligned}$$

Es scheint, dass 4 reelle Multiplikationen zur Berechnung einer komplexen

Matrixmultiplikation nötig sind. Führt man aber die Matrizen

$$\begin{aligned}\mathbf{S} &= (\mathbf{A}_r + \mathbf{A}_i)(\mathbf{B}_r - \mathbf{B}_i)\\ &= \mathbf{A}_r\mathbf{B}_r - \mathbf{A}_r\mathbf{B}_i + \mathbf{A}_i\mathbf{B}_r - \mathbf{A}_i\mathbf{B}_i\\ \mathbf{T} &= \mathbf{A}_i\mathbf{B}_r\\ \mathbf{U} &= \mathbf{A}_r\mathbf{B}_i\end{aligned}$$

ein, so ist

$$\mathbf{C}_r = \mathbf{S} - \mathbf{T} + \mathbf{U} \quad \textit{und} \quad \mathbf{C}_i = \mathbf{T} + \mathbf{U}$$

und man kann damit $\mathbf{C}$ *mit nur 3 Matrizenmultiplikationen berechnen. Man schreibe ein Programm für diese Methode.*

Wir benützen die Prozeduren von Aufgabe 2.22 und erhalten das folgende Programm:

```
(*$B- Aufgabe 2.23*)
program komplmatmult;
const nn = 10;
type matrix =   record
                    m,n : integer;
                    a : array [1..nn, 1..nn] of real;
                end ;
var n : integer; ar,ai,br,bi,cr,ci,s,t,u : matrix;
(*$I matmult ( siehe Aufgabe 2.22)*)
(*$I matadd ( siehe Aufgabe 2.22)*)
(*$I matsub ( siehe Aufgabe 2.22)*)
(*$I matdruck ( siehe Aufgabe 2.22)*)
procedure eingabe( var a : matrix);
var i,j : integer;
begin
   writeln('Matrix zeilenweise eingeben');
   for i := 1 to a.m do for j := 1 to a.n do read(a.a[i, j])
end ;
begin
   writeln('Multiplikation von 2 komplexen Matrizen');
   writeln(' (AR + I * AI) * (BR + I * BI) ');
   writeln('Anzahl Zeilen der Matrix A ='); read(ar.m);
   writeln('Anzahl Kolonnen der Matrix A ='); read(ar.n);
   ai.m := ar.m; ai.n := ar.n;
   writeln('Realteile AR eingeben'); eingabe(ar);
   writeln('Imaginaerteile AI eingeben'); eingabe(ai);
   writeln('Anzahl Kolonnen der Matrix B ='); read(br.n);
```

```
    writeln('Realteile BR eingeben');
    br.m := ar.n; bi.m := br.m; bi.n := br.n; eingabe(br);
    writeln('Imaginaerteile BI eingeben'); eingabe(bi);
    matadd(ar, ai, t); matsub(br, bi, u); matmult(t, u, s);
    matmult(ai, br, t); matmult(ar, bi, u);
    matsub(s, t, s); matadd(s, u, cr); matadd(t, u, ci);
    writeln('Resultat');
    writeln('Realteil'); matdruck(cr);
    writeln('Imaginaerteil'); matdruck(ci);
end.
```

Aufgabe 2.24 *Beim Lösen von linearen homogenen Differentialgleichungssystemen mit konstanten Koeffizienten*

$$\mathbf{y}' = \mathbf{A}\mathbf{y}$$

wird die Exponentialmatrix

$$e^{\mathbf{A}} = \mathbf{I} + \frac{\mathbf{A}}{1!} + \frac{\mathbf{A}^2}{2!} + \cdots$$

oft benützt. Man schreibe ein Programm, welches die Exponentialmatrix für gegebenes $\mathbf{A}$ *berechnet.*

Hinweis: Man übernehme den Algorithmus 2.7 und ersetze die Operationen durch Matrizenoperationen. Anstelle des Betrages, verwende man eine Matrixnorm, z.B. die Frobeniusnorm

$$\|\mathbf{A}\| = \sqrt{\sum_{i,j=1}^{n} a_{ij}^2}.$$

Der Kern des Algorithmus 2.7 lautet

```
m := 0; z := x;                          Hier wird die Potenz
while abs(z) > 1 do                      m bestimmt, so dass
begin m := m + 1; z := z/2 end ;         |x|/2^m < 1 ist.

sn := 1; a := 1; i := 0;                 Initialisierung
repeat
    i := i + 1; s := sn; a := a * z/i;   Neuer Summand a
    sn := s + a                          Neue Teilsumme sn
until sn = s;

for i := 1 to m do sn := sqr(sn);        Quadrieren
```

```
e := sn
```

Für Matrizen geschrieben, lautet dasselbe Programmstück:

```
m := 0; zhm := 1; f := frob(a);                Hier wird m
while zhm <= f do                              bestimmt, so dass
begin m := m + 1; zhm := 2 * zhm end ;         ||a||/2^m < 1 ist.
adivk(a, zhm, z);
einheitsmatrix(sn); einheitsmatrix(term); i := 0;
repeat
   i := i + 1; s := sn;
   matmult(term, z, term);
   adivk(term, i, term);                       Neuer Summand term
   matadd(s, term, sn)                         Neue Teilsumme sn
until gleich(sn, s);
for i := 1 to m do matmult(sn, sn, sn);        Quadrieren
e := sn;
```

Auch bei den Matrizen berechnen wir die Reihe solange, bis der neue Summand so klein geworden ist, dass er nichts mehr zur Summe beiträgt. Wir brauchen eine *boolean* **function** *gleich*, welche angibt, ob die beiden Partialsummenmatrizen s und sn elementweise übereinstimmen. Die Prozedur *adivk* berechnet die Division einer Matrix durch eine Zahl. Wir brauchen sie, um die Summanden durch die Fakultäten zu dividieren.

```
(*$B- Aufgabe 2.24*)
program expa;
type matrix = array [1..10,1..10] of real;
     vektor = array [1..10] of real;
     wort = string [30];
var a,e : matrix; z,y : vektor; k,n,i,j : integer;
   tf : text; stri : wort;
procedure matdruck(s : wort; n : integer; a : matrix);
var i,j : integer;
begin
   writeln(tf, s);
   for i := 1 to n do
   begin
      writeln(tf);
      for j := 1 to n do write(tf, a[i, j] : 18);
   end ;
   writeln(tf); writeln(tf)
end ;
```

```
procedure ehoch(n : integer; a : matrix; var e : matrix);
var s,sn,x,term,z : matrix;
    zhm,m,k,i : integer;
    f : real;
procedure matmult(a,b : matrix;var c : matrix);
var i,j,k : integer;
s : real;
begin
    for i := 1 to n do for j := 1 to n do
    begin
        s := 0;
        for k := 1 to n do s:=s+a[i,k]*b[k,j];
        c[i,j] := s;
    end ;
end ;
function frob( var a : matrix) : real;
var s : real; i,j : integer;
begin
    s := 0;
    for i := 1 to n do
    for j := 1 to n do s := s + sqr(a[i,j]);
    frob := sqrt(s);
end ;
procedure adivk(a : matrix; k : real;var c : matrix);
var i,j : integer;
begin
    for i := 1 to n do
    for j := 1 to n do c[i,j] := a[i,j]/k;
end ;
procedure einheitsmatrix( var a : matrix);
var i,j : integer;
begin
    for i := 1 to n do for j := 1 to n do
    if i = j then a[i,j] := 1 else a[i,j] := 0;
end ;
procedure matadd(a,b : matrix; var c : matrix);
var i,j : integer;
begin
    for i := 1 to n do for j := 1 to n do
    c[i,j] := a[i,j] + b[i,j];
end ;
```

```
function gleich(a,b : matrix) : boolean;
var i,j : integer; g : boolean;
begin
   g := true; i := 1;
   repeat
      j := 1;
      repeat
         g := g and (a[i,j] = b[i,j]); j := j + 1
      until not g or (j > n);
      i := i + 1
   until not g or (i > n);
   gleich := g
end ;

begin
   m := 0; zhm := 1; f := frob(a);
   while zhm <= f do begin m := m + 1; zhm := 2 * zhm end ;
   adivk(a, zhm, z);

   einheitsmatrix(sn); einheitsmatrix(term); i := 0;
   repeat
      i := i + 1;
      matdruck('Term',n,term); matdruck('sn',n,sn);
      s := sn; matmult(term, z, term); adivk(term, i, term);
      matadd(s, term, sn)
   until gleich(sn, s);
   for i := 1 to m do matmult(sn, sn, sn);
   e := sn;
end ;

begin
   writeln('Wohin mit dem Output');
   readln(stri); assign(tf, stri); rewrite(tf);
   writeln('N=?'); readln(n);
   writeln('Matrix A zeilenweise eingeben');
   for i := 1 to n do
   for j := 1 to n do read(a[i,j]);
   matdruck('Gegebene Matrix',n,a);
   ehoch(n, a, e);
   matdruck('exp(A)',n,e);
   close(tf)
end.
```

Beispiel: wir berechnen die Exponentialmatrix von

$$\mathbf{A} = \begin{pmatrix} 4 & 2 & 1 \\ 2 & -1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Es werden hier nur 13 Summanden der Exponentialreihe benötigt und man erhält zunächst:

```
sn
1.7040157624E+00 3.2030137941E-01 1.8266912547E-01
3.2030137941E-01 9.2578074963E-01 1.3763225395E-01
1.8266912547E-01 1.3763225395E-01 1.0183761320E+00
```

Dieses Resultat muss noch 3 mal quadriert werden und ergibt

```
exp(A)
1.2553370888E+02 4.7199158894E+01 3.4084090822E+01
4.7199158894E+01 1.8020323026E+01 1.3115068071E+01
3.4084090822E+01 1.3115068071E+01 1.0166368345E+01
```

Aufgabe 2.25 *Um einen Tisch sitzen 7 Zwerge. Vor jedem steht ein Becher. Einige Becher enthalten Milch, insgesamt 3 Liter. Einer der Zwerge verteilt seine Milch gleichmässig auf die anderen Becher. Danach tut sein rechter Nachbar dasselbe. Genauso verfährt der nächste rechts herum u.s.w. Nachdem der 7. Zwerg seine Milch verteilt hat, ist in jedem Becher soviel Milch wie anfangs. Wieviel Milch war anfangs in jedem Becher ?*

Diese Aufgabe stammt aus einer deutschen Mathematikolympiade und kann analytisch gelöst werden. Man kann sie aber auch mittels Matrizenrechnung wie folgt lösen: Seien

$$x_1^{(0)}, \ldots, x_7^{(0)} \text{ die Milchmengen am Anfang}$$

und

$$x_1^{(1)}, \ldots, x_7^{(1)} \text{ nach dem ersten Ausschank.}$$

Es gilt

$$\mathbf{x}^{(1)} = \mathbf{T}^{(1)}\mathbf{x}^{(0)},$$

wobei

$$\mathbf{T}^{(1)} = \begin{pmatrix} 1/7 & 0 & \cdots & & 0 \\ 1/7 & 1 & 0 & \cdots & 0 \\ 1/7 & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 1/7 & 0 & \cdots & 0 & 1 \end{pmatrix} \qquad (2.25*)$$

ist. Allgemein wird der i-te Ausschank durch die Transformation

$$\mathbf{x}^{(i)} = \mathbf{T}^{(i)}\mathbf{x}^{(i-1)} \tag{2.12}$$

beschrieben, wobei $\mathbf{T}^{(i)}$ *ähnlich wie* $\mathbf{T}^{(1)}$ *aussieht: Die Kolonne mit* $1/7$ *ist nun die i-te statt die erste. Für den Endzustand haben wir*

$$\mathbf{x}^{(7)} = \underbrace{\mathbf{T}^{(7)} \dots \mathbf{T}^{(1)}}_{\mathbf{A}} \mathbf{x}^{(0)}$$

und da $\mathbf{x}^{(7)} = \mathbf{x}^{(0)} =: \mathbf{x}$ *ist, erhält man das homogene lineare Gleichungssystem*

$$(\mathbf{A} - \mathbf{I})\mathbf{x} = \mathbf{0}. \tag{2.26*}$$

Man kann zeigen, dass nichttriviale Lösungen existieren. Die Lösung wird eindeutig, wenn wir die Gesamtmilchmenge von 3 Litern berücksichtigen, also die Gleichung

$$x_1 + x_2 + \cdots + x_7 = 3$$

dem System (2.26) zufügen.*
Man schreibe ein Programm, um die Matrix $\mathbf{A}$ *zu berechnen, stelle das Gleichungssystem auf und löse es mittels des Givensverfahrens (siehe Kap. 5) als überbestimmtes Gleichungssystem.*

Als erstes wollen wir uns vergewissern, dass das Problem eine Lösung hat. Was wir suchen, ist der *Eigenvektor der Matrix* $\mathbf{A}$ zum *Eigenwert* $\lambda = 1$: $\mathbf{A}\mathbf{x} = 1 \cdot \mathbf{x}$. Es ist nicht klar, dass die Matrix $\mathbf{A}$ diesen Eigenwert besitzt.

Betrachten wir aber die transponierte Matrix $\left(\mathbf{T}^{(1)}\right)^T$, so ist es offensichtlich, dass der Vektor $\mathbf{e} = (1, 1, \dots, 1)^T$ Eigenvektor dieser Matrix zum Eigenwert $\lambda = 1$ ist. Ebenso ist dieser Vektor auch Eigenvektor für alle übrigen Matrizen $\left(\mathbf{T}^{(i)}\right)^T$ und wegen

$$\begin{aligned}\mathbf{A}^T &= \left(\mathbf{T}^{(7)} \dots \mathbf{T}^{(1)}\right)^T \\ &= \left(\mathbf{T}^{(1)}\right)^T \dots \left(\mathbf{T}^{(7)}\right)^T\end{aligned}$$

ist $\mathbf{e}$ auch Eigenvektor von $\mathbf{A}^T$ zum Eigenwert $\lambda = 1$. Da die Transponierte einer Matrix die gleichen Eigenwerte hat, können wir schliessen, dass $\mathbf{A}$ auch den Eigenwert 1 haben muss.

Nachdem die Existenz der Lösung gesichert ist, wollen wir diese berechnen. Wir beschränken uns nicht auf 7, sondern schreiben das Programm für n Zwerge. Die Prozedur *tk* erzeugt die Matrix $\mathbf{T}^{(k)}$. Wir benötigen ferner eine Prozedur *matmult*, um die Matrix A zu berechnen und eine Prozedur, um

das Gleichungssystem aufzulösen. Wir verwenden die Prozedur *givenselimination* von Algorithmus 5.6, wobei die obere Grenze der k-Schleife geändert wird, damit die $n+1$ Gleichungen verarbeitet werden (vgl. Aufgabe 5.12).

Zur Kontrolle simulieren wir am Schluss die Milchverteilung, indem die Folge (2.12) mit dem gefundenen Lösungsvektor berechnet wird. Dazu benötigen wir eine Prozedur *mult*, um einen Vektor mit einer Matrix zu multiplizieren.

```
(*$B- Aufgabe 2.25*)
program zwerge;
const nn = 20;
type vektor = array [1..nn] of real;
   matrix = array [1..nn] of vektor;
var n,k,i : integer; a,t : matrix; b,x : vektor;

procedure tk(k : integer; var t : matrix);
var i,j : integer;
begin
   for i := 1 to n do for j := 1 to n do
   if i = j then t[i,j] := 1 else t[i,j] := 0;
   for i := 1 to n do t[i,k] := 1/n
end ;

procedure matmult(a,b : matrix; var c : matrix);
var i,j,k : integer; s : real;
begin
   for i := 1 to n do for j := 1 to n do
   begin
      s := 0 ;
      for k := 1 to n do s := s + a[i,k] * b[k,j];
      c[i,j] := s
   end
end ;

procedure rueckwaertseinsetzen;
var s : real; i,j : integer;
begin
   for i := n downto 1 do
   begin
      s := b[i];
      for j := i + 1 to n do s := s - a[i,j] * x[j];
      x[i] := s/a[i,i]
   end
end ;

procedure givenselimination;
```

```
(* leicht abgeaenderter Algorithmus 5.6, vgl. auch Aufgabe 5.12 *)
var i,j,k : integer; cot,co,si,h : real;
begin
   for i := 1 to n do
   begin
      for k := i + 1 to n + 1 do    (* n+1 Gleichungen ! *)
      if a[k, i] <> 0 then
      begin
         cot := a[i, i]/a[k, i];
         si := 1/sqrt(1 + cot * cot); co := si * cot;
         a[i, i] := a[i, i] * co + a[k, i] * si;
         for j := i + 1 to n do
         begin
            h := a[i, j] * co + a[k, j] * si;
            a[k, j] := −a[i, j] * si + a[k, j] * co;
            a[i, j] := h
         end ;
         h := b[i] * co + b[k] * si;
         b[k] := −b[i] * si + b[k] * co;
         b[i] := h
      end
   end
end ;

procedure mult(a : matrix; x : vektor; var y : vektor);
var i,j : integer; s : real;
begin
   for i := 1 to n do
   begin s := 0;
      for j := 1 to n do s := s + a[i, j] * x[j];
      y[i] := s
   end ;
end ;
begin
   writeln('Anzahl Zwerge n=?'); read(n);
   tk(1, a);
   for k := 2 to n do
   begin tk(k, t); matmult(t, a, a) end ;
   for i := 1 to n do
   begin
      a[i, i] := a[i, i] − 1;    (* Matrix A − I berechnen *)
      a[n + 1, i] := 1;    (* letzte Gleichung *)
      b[i] := 0    (* rechte Seite *)
```

```
    end ;
    b[n + 1] := 3;    (* Gesamtmilchmenge *)
    givenselimination; rueckwaertseinsetzen;
    writeln('Loesung');
    for i := 1 to n do writeln(x[i]);
    writeln('Kontrolle');
    for i := 1 to n do
    begin tk(i, t); mult(t, x, x) end ;
    for i := 1 to n do writeln(x[i]);
end.
```

Für 7 Zwerge erhält man das Resultat

```
7.5000000000E-01
6.4285714286E-01
5.3571428571E-01
4.2857142857E-01
3.2142857143E-01
2.1428571428E-01
1.0714285714E-01
```

Aufgabe 2.26 *Man berechne mittels Algorithmus 2.14* $1'000$ *Dezimalstellen von e. Man verbessere den Output so, dass, wenn* $a[i] < c = 10^m$ *ist, keine Lücke, sondern entsprechend viele Nullen gedruckt werden. Ferner drucke man eine schöne Tabelle z.B. 80 Ziffern pro Zeile*

Um den Output so zu verbessern, dass statt der Lücken Nullen gedruckt werden, müssen wir von einer *integer* Zahl, die eine Zifferngruppe enthält, die einzelnen Ziffern von links nach rechts herauslesen und ausdrucken.

Wir benützen dazu einen Algorithmus für Zahlenumwandlungen (vgl. Lehrbuch Kapitel 4.2). Die rekursive Prozedur *stelle* von Aufgabe 4.6 leistet gerade das Gewünschte, wenn wir sie für unsere Zwecke anpassen, sodass sie statt der Dualziffern die Dezimalziffern berechnet. Naheliegender wäre eine Lösung, bei welcher die Ziffern in einem **array** abgespeichert werden und dieser mit Nullen auf die Länge m aufgefüllt wird. Die vorgeschlagene Lösung ist aber programmiertechnisch einfacher und eleganter.

Bei der folgenden Prozedur *drucken* wird jeweils nach 10 Dezimalziffern eine Lücke gedruckt (dies ist eine wirkliche Lücke und nicht eine verkappte Null!), damit wird die Leserlichkeit erhöht. Damit der Output auf dem Bildschirm Platz hat, drucken wir nur 70 Ziffern pro Zeile (auf das Textfile tf):

```
procedure drucken(s : megezahl);
var k,i : integer;
procedure stelle(n : integer);
(* Man vergleiche dazu Aufgabe 4.6*)
begin
   k := k + 1;
   if k <= m then
   begin stelle(n div 10); write(tf, n mod 10 : 1) end ;
end ;
begin
   for i := 1 to n do
   begin
      k := 0; stelle(s[i]);
      if (i * m) mod 10 = 0 then write(tf,' ');
      if (i * m) mod 70 = 0 then writeln(tf)
   end ;
end ;
```

Leider ist der Integerbereich in TURBO PASCAL zu klein, als dass von dieser Prozedur wirklich Gebrauch gemacht werden könnte. Für $n = 30$ und $m = 2$ erhalten wir:

0271828182 8459045235 3602874713 5266249775 7247093699 9595749645

Für grösseres n entsteht bei der Operation $rest := (\ rest$ **mod** $h) * c + a[i + 1]$ sehr bald ein Integer-Überlauf. Man muss sich also im TURBO PASCAL auf die Packungsdichte $m = 1$ beschränken.

Die Prozedur *drucken* und die folgenden Prozeduren werden als Include-Files zugeladen. Wir werden diese auch bei der Berechnung von π (Aufgabe 2.29) benützen.

```
procedure add(n,imin : integer; var a,b,res : megezahl);
var i : integer;
begin
   for i := imin to n do res[i] := a[i] + b[i]
end ;
(* Algorithmus 2.12*)
procedure uebertrag(n,c : integer; var a : megezahl);
var i : integer;
begin
   for i := n downto 1 do
   while a[i] >= c do
   begin a[i] := a[i] - c; a[i - 1] := a[i - 1] + 1 end
end ;
```

```
(* Algorithmus 2.13*)
procedure teil(n,c : integer; var imin : integer;
          k : integer; var a : megezahl);
var rest,i : integer; null : boolean;
begin
   null := true; rest:= a[imin];
   for i := imin to n − 1 do
   begin
      a[i] := rest div k; rest := (rest mod k) ∗ c + a[i + 1];
      if null then
      if a[i] = 0 then imin := i else null := false;
   end ;
   a[n] := rest div k
end ;
```

Das restliche Programm lautet:

```
(*$B− Aufgabe 2.26*)
program emege;
const n = 1000; m = 1;
(*m = Packungsdichte. Bei TURBO PASCAL fuehrt m > 1 zu einem
       integer Ueberlauf fuer 1000 Stellen ! *)
(* Es werden m ∗ n Stellen von e berechnet *)
type megezahl = array [0..n] of integer;
var a,s : megezahl; i,imin,c,k : integer;
   tf : text; stri : string [10];
(*$I add *)
(*$I uebertrag Algorithmus 2.12*)
(*$I teil Algorithmus 2.13*)
(*$I drucken *)
begin
   writeln('wohin mit dem Output?');
   readln(stri); assign(tf, stri); rewrite(tf);
   c := 1; for i := 1 to m do c := c ∗ 10;
   for i := 0 to n do
   begin a[i] := 0; s[i] := 0 end ;
   (* Initialisierung *)
   s[1] := 2; a[1] := 1; k := 1; imin := 0;
   repeat
      k := k + 1;
      teil(n, c, imin, k, a);     (* neuer Summand *)
      add(n, imin, s, a, s);     (* neue Partialsumme *)
      uebertrag(n, c, s);
```

```
  until imin = n − 1;
  drucken(s); close(tf)
end.
```

Mit dem Programm *emege* erhält man den Output von Tabelle 2.1.

2718281828 4590452353 6028747135 2662497757 2470936999 5957496696 7627724076
6303535475 9457138217 8525166427 4274663919 3200305992 1817413596 6290435729
0033429526 0595630738 1323286279 4349076323 3829880753 1952510190 1157383418
7930702154 0891499348 8416750924 4761460668 0822648001 6847741185 3742345442
4371075390 7774499206 9551702761 8386062613 3138458300 0752044933 8265602976
0673711320 0709328709 1274437470 4723069697 7209310141 6928368190 2551510865
7463772111 2523897844 2505695369 6770785449 9699679468 6445490598 7931636889
2300987931 2773617821 5424999229 5763514822 0826989519 3668033182 5288693984
9646510582 0939239829 4887933203 6250944311 7301238197 0684161403 9701983767
9320683282 3764648042 9531180232 8782509819 4558153017 5671736133 2069811250
9961818815 9304169035 1598888519 3458072738 6673858942 2879228499 8920868058
2574927961 0484198444 3634632449 6848756023 3624827041 9786232090 0216099023
5304369941 8491463140 9343173814 3640546253 1520961836 9088870701 6768396424
3781405927 1456354906 1303107208 5103837505 1011574770 4171898610 6873969655
2126715468 8957034808

Tabelle 2.1: Eulersche Zahl e

Aufgabe 2.27 *Man berechne eine Tabelle der Potenzen von 2:*

$$2^i, \quad i = 1, 2, \ldots, 300$$

Im folgenden Programm berechnen wir die Zweierpotenzen nur bis $i = 162$, dadurch wird gerade die Breite des Bildschirms ausgenützt. Da $2^{300} \simeq 10^{90}$ ist, muss bei einer Vergrösserung auf $i = 300$ auch die Konstante nn auf etwa $nn = 100$ gesetzt werden, da sonst der **array** a zu klein ist. Solange es möglich ist (Überlauf!), berechnen wir die Zweierpotenz als Kontrolle zusätzlich als *real* Zahl.

```
(*$B- Aufgabe 2.27*)
program zwei;
const nn = 50;
type megezahl = array [1..nn] of integer;
var a : megezahl; i,j,n,imax : integer; z : real;
begin
   a[1] := 1; for i := 2 to nn do a[i] := 0;
   imax := 1; n := 0; z := 1;
   repeat
      write('2**', n:3,' = ',' ': nn - imax);
      for j := imax downto 1 do write(a[j]:1);
      if z/10 = 0 then writeln else writeln(1/z);
      n := n + 1; z := z/2; for j := 1 to imax do a[j] := a[j] * 2;
      (* Uebertraege nachfuehren: Es ist hier 0 <= a[j] < 20 *)
      for j := 1 to imax do
      if a[j] >= 10 then
      begin a[j] := a[j] - 10; a[j + 1] := a[j + 1] + 1 end ;
      if a[imax + 1] > 0 then imax := imax + 1
   until nn=imax
end.
```

Aufgabe 2.28 *Man schreibe ein Programm, welches $n!$ für beliebig grosses n exakt berechnet.*

Das nachfolgende Programm ist ähnlich aufgebaut wie dasjenige von Aufgabe 2.27. Der **array** *nfak* enthält die Ziffern des Resultates und wird fortlaufend durch Multiplikation mit der *integer* Schleifenvariable i, $i = 1, 2, \ldots, n$ aufgebaut. Die Länge $nn = 300$ von *nfak* reicht aus, um $166!$ zu rechnen.

```
(*$B- Aufgabe 2.28*)
program fakultaet;
const nn = 300;
type megezahl = array [1..nn] of integer;
var nfak : megezahl; i,j,n,imax : integer;
begin
   writeln('Berechnung von n!, n=?'); read(n);
   nfak[1] := 1; for i := 2 to nn do nfak[i] := 0; imax := 2;
   for i := 1 to n do
   begin
      for j := 1 to imax do nfak[j] :=nfak[j] * i;
      (* Uebertraege nachfuehren *)
      j := 1;
```

```
      repeat
        while nfak[j] >= 10 do
        begin
          nfak[j] := nfak[j] - 10; nfak[j+1] := nfak[j+1] + 1;
        end ;
        j := j + 1;
        if nfak[imax] > 0 then imax := imax + 1;
        if imax > nn then writeln('nn vergroessern !')
      until j = imax;
    end ;
    writeln(n,' ! = '); j := 0;
    for i := imax - 1 downto 1 do
    begin
      j := j + 1; write(nfak[i] : 1);
      if (i > 1) and ((i - 1) mod 10 = 0) then
      begin
        write(''''); j := j + 1;
        if j > 60 then begin writeln; j := 0 end
      end ;
    end
end.
```

Aufgabe 2.29 *Man berechne π auf 1000 Stellen. Hinweis: Man verwende die Reihe der Funktion* arctan *und die Gleichung* (2.15).

Es ist eine Tradition geworden, den Computer für die Berechnung von vielen Stellen von π einzusetzen. Letztes Jahr (Mai 1984) wurde ein Rekord von Wissenschaftlern in Tokyo dadurch aufgestellt, dass sie *16 Millionen Dezimalziffern* dieser misteriösen Zahl berechneten.

Viele Formeln, die dabei verwendet werden, beruhen auf Addition von Vielfachen von kleinen Winkeln, deren Tangenswerte rationale Zahlen sind. In der folgenden Tabelle sind 3 solche Identitäten angegeben:

$$\text{C. Störmer} \quad \pi = 24\arctan\frac{1}{8} + 8\arctan\frac{1}{57} + 4\arctan\frac{1}{239} \qquad (2.15*)$$

$$\text{C.F.Gauss} \quad \pi = 48\arctan\frac{1}{18} + 32\arctan\frac{1}{57} - 20\arctan\frac{1}{239} \qquad (2.13)$$

$$\text{Machin} \quad \pi = 16\arctan\frac{1}{5} - 4\arctan\frac{1}{239} \qquad (2.14)$$

Um die arctan Funktion zu berechnen, verwenden wir ihre MacLaurinreihe. Diese wurde bereits in Aufgabe 2.14 programmiert. Wir müssen jene **function** *atan* nur noch für mehrfache Genauigkeit umschreiben. Als Argument

tritt jetzt nicht x sondern ein echter Bruch $1/k$ auf, wie z.B. $1/57$. Die folgende Prozedur ist eine leicht angepasste Version von *atan* und berechnet $s = \arctan(1/k)$:

```
procedure arcustan(k : integer; var s : real);
var salt, t,h : real; i : integer;
begin
   t := 1/k ; s := t; i := 1;
   repeat
      salt := s; i := i + 2 ; t := −t/k/k;
      h := t/i;
      s := salt + h
   until s = salt
end ;
```

Aus den *real* Zahlen werden Zahlen vom Typus *megezahl* und die Operationen müssen durch Prozeduraufrufe ersetzt werden. Die alternierenden Vorzeichen der Summanden bewirken, dass abwechselnd die Prozeduren *add* und *sub* ausgeführt werden müssen. Beim Berechnen des neuen Summanden t merkt man sich mit der Variabeln *imin*, wo die ersten Ziffern $\neq 0$ beginnen. Die Summation wird abgebrochen, wenn der nächste Summand aus lauter Nullen besteht. Im folgenden Programm berechnen wir π mit allen 3 Varianten. Durch Vergleich der Ergebnisse kann man den Einfluss der Rundungsfehler feststellen. Die Prozeduren *add, uebertrag, teil* und *drucken* übernehmen wir von der Aufgabe 2.26.

```
(*$B− Aufgabe 2.29*)
program pimege;
const n = 140; m = 1;
(* m=Packungsdichte. Es werden m * n Stellen von pi berechnet *)
type megezahl = array [0..n] of integer;
var a,b,cc,s1,s2,s3 : megezahl; i,imin,c,k : integer;
    stri : string [30]; tf : text;
($I add ( siehe Aufgabe 2.26)*)
($I uebertrag Algorithmus 2.12*)
($I sub*)
($I teil Algorithmus 2.13*)
($I drucken ( siehe Aufgabe 2.26)*)
procedure sub(n, c, imin: integer; a, b : megezahl; var r : megezahl);
var i : integer;
begin
   i := n + 1; b[0] := 0;
   repeat
      i := i − 1;
```

```
        while a[i] < b[i] do
        begin
            a[i] := a[i] + c; b[i − 1] := b[i − 1] + 1
        end ;
        r[i] := a[i] − b[i];
    until ((i <=imin) and (b[i − 1] = 0))
end ;

procedure mal(n, k : integer; var a : megezahl);
var i : integer;
begin
    for i := 1 to n do a[i] := a[i] ∗ k
end ;

procedure arcustan(n, c, k : integer; var s : megezahl);
var t,h : megezahl; i,j,vorzeichen,ia : integer;
(* Berechnet s = arctan(1/k). Man vergleiche dazu die Aufgabe 2.14∗)
begin
    t[1] := 1; for j := 2 to n do t[j] := 0; imin := 1;
    teil(n, c, imin, k, t); s := t; i := 1; vorzeichen := 1;
    repeat
        i := i + 2; teil(n, c, imin, k, t); teil(n, c, imin, k, t);
        ia := imin; h := t; teil(n, c, imin, i, h); imin := ia;
        vorzeichen := −vorzeichen;
        case vorzeichen of
              1 : begin add(n, imin, s, h, s); uebertrag(n, c, s) end ;
             −1 : sub(n, c, imin, s, h, s)
        end ;
    until imin = n − 1
end ;

begin
    writeln('Berechnung von pi. Wohin mit dem Output ?');
    readln(stri); assign(tf, stri); rewrite(tf);
    c := 1; for i := 1 to m do c := c ∗ 10;
    arcustan(n, c, 8, s1); mal(n, 24, s1);
    arcustan(n, c, 57, s2); mal(n, 8, s2);
    arcustan(n, c, 239, s3); mal(n, 4, s3);
    add(n, 1, s1, s2, s1); add(n, 1, s1, s3, s1);
    uebertrag(n, c, s1);
    writeln( tf, 'Nach Stoermer 24 arctan(1/8) ',
                '+8 arctan(1/57) + 4 arctan(1/239) ');
    drucken(s1);
    writeln(tf); writeln(tf);
    arcustan(n, c, 18, s1); mal(n, 48, s1);
```

```
  arcustan(n, c, 57, s2); mal(n, 32, s2);
  arcustan(n, c, 239, s3); mal(n, 20, s3);
  add(n, 1, s1, s2, s1); sub(n, c, 1, s1, s3, s1);
  uebertrag(n, c, s1);
  writeln(tf,'Nach Gauss 48 arctan(1/18)',
          '+32 arctan(1/57) - 20 arctan(1/239)');
  drucken(s1);
  writeln(tf); writeln(tf);
  arcustan(n, c, 5, s1); mal(n, 16, s1);
  arcustan(n, c, 239, s2); mal(n, 4, s2);
  sub(n, c, 1, s1, s2, s1); uebertrag(n, c, s1);
  writeln(tf,'Nach Machin 16 arctan(1/5) - 4 arctan(1/239)');
  drucken(s1);
  writeln(tf); writeln(tf);
  close(tf);
end.
```

Als Output erhalten wir mit dem Programm *pimege* für $n = 140$ und $m = 1$ die Werte von Tabelle 2.2. Zur Lösung der gestellten Aufgabe muss im Programm die Konstante $n = 1000$ gesetzt werden.

```
Nach Stoermer 24 arctan(1/8) + 8 arctan(1/57) + 4 arctan(1/239)
3141592653 5897932384 6264338327 9502884197 1693993751 0582097494 4592307816
4062862089 9862803482 5342117067 9821480865 1328230664 7093844609 5505822296
Nach Gauss 48 arctan(1/18) + 32 arctan(1/57) - 20 arctan(1/239)
3141592653 5897932384 6264338327 9502884197 1693993751 0582097494 4592307816
4062862089 9862803482 5342117067 9821480865 1328230664 7093844609 5505821888
Nach Machin 16 arctan(1/5) - 4 arctan(1/239)
3141592653 5897932384 6264338327 9502884197 1693993751 0582097494 4592307816
4062862089 9862803482 5342117067 9821480865 1328230664 7093844609 5505822320
```

Tabelle 2.2: Berechnung von π

Aufgabe 2.30 *Viele Leser haben in ihrer Schulzeit noch ein Rechenverfahren kennengelernt, um die Quadratwurzel einer gegebenen Zahl von Hand zu berechnen. Dieser Algorithmus soll hier programmiert werden.*

Theorie dazu: Sei a der Radikand und w eine Näherung für die Wurzel mit $w^2 < a$. Gesucht wird eine Korrektur z so, dass

$$a = (w + z)^2$$

gilt, d.h.

$$rest = a - w^2 = 2wz + z^2 = (2w + z)z.$$

Falls nun $2w >> z$ *ist, so lässt sich die Korrektur* z *näherungsweise durch*

$$z = \frac{a - w^2}{2w} \qquad (2.28*)$$

schätzen. Beim Handrechnen besteht die Korrektur nur aus einer einzigen neuen Ziffer von w*. Das folgende Beispiel diene als Illustration:*

$$\begin{array}{lll}
\sqrt{2}\ 1\ 4\ 3\ 6\ 9 & = & 463 \\
\underline{1\ 6} & & \\
\quad 5\ 4\ 3 & : & (80{+}6)6 \\
\quad \underline{5\ 1\ 6} & & \\
\quad\quad 2\ 7\ 6\ 9 & : & (920{+}3)3 \\
\quad\quad \underline{2\ 7\ 6\ 9} & & \\
\quad\quad\quad\ 0 & &
\end{array}$$

Betrachten wir bei diesem Beispiel den zweiten Iterationsschritt, d.h. wie die Ziffer $z = 6$ *der Wurzel erhalten wird. Der bisherige Näherungswert der Wurzel ist* $w = 4$ *und der Rest* $r = 21 - 16 = 5$*. Nach der Gleichung* $(2.28*)$ *ergibt sich die nächste Stelle durch Division des Restes durch* $2w$*. Da aber an den Rest* r *die beiden nächsten Stellen des Radikanden zugefügt werden, rechnet man*

$$r := r * 100 + 4 * 10 + 3 = 543.$$

Dieser Rest muss durch $20{*}w = 80$ *geteilt werden. Man erhält als Schätzung für die nächste Ziffer* $z = 6$*. Für den neuen Rest wird die Ziffer an* $20 * w$ *angefügt, d.h. man rechnet*

$$(20 * w + z)z = 516$$

und erhält damit

$$r_{neu} = r - (20 * w + z)z = 543 - 516 = 27.$$

Es kann vorkommen, dass der Rest r_{neu} *negativ wird. In diesem Fall wurde* z *zu gross geschätzt und man probiert mit* $z := z - 1$*.*

Wenn $a[k], a[k+1]$ *die nächsten Ziffern des Radikanden bedeuten und die Ziffer der Wurzel in* $w[l]$ *gespeichert wird, kann ein Iterationsschritt wie folgt zusammengefasst werden*

rest := *rest* $*100 + a[k] * 10 + a[k+1]$*;*
$k := k + 2$*; zweia* := 20$*$ *wurzel;*

ziffer := *rest* **div** *zweia;*
hilf := (*zweia* + *ziffer*)∗ *ziffer;*
while *rest* < *hilf* **do**
begin
ziffer := *ziffer* −1*;*
hilf := (*zweia* + *ziffer*)∗ *ziffer;*
end *;*
rest := *rest* − *hilf; l* := *l* + 1*; w*[*l*] := *ziffer;*

Dieses Programmstück bildet den Kern des zu erstellenden Programms. Die Variabeln a, hilf, rest, wurzel und zweia sind mehrfachgenaue Zahlen. Die Variable ziffer ist aber nur eine integer Zahl. Für alle Operationen sind entsprechende Prozeduren zu programmieren.

Diese Aufgabe ist etwas komplizierter, wir wollen sie deshalb in zwei Schritten lösen. Zuerst machen wir ein Programm, das in einfacher Genauigkeit rechnet, bei dem wir aber schon den in der Aufgabenstellung beschriebenen Algorithmus benützen. Anschliessend werden wir dieses Programm für mehrfachgenaues Rechnen erweitern.

Beim dem oben beschriebenen Iterationsschritt werden für eine Dezimalstelle der Wurzel jeweils 2 Dezimalstellen des Radikanden verwendet. Man muss dazu die Dezimalstellen des Radikanden in Zweiergruppen einteilen. So erhält man etwa für die sechsstellige Zahl $a = 63'42'50$

$$\sqrt{634250} = 796 \quad \text{Rest} \quad 634,$$

also nur eine dreistellige Wurzel $w = 796$. Es ist $w^2 = 796^2 = 633616$. Wenn man gleichviele signifikante Stellen von w wie a haben möchte, *muss man durch Anfügen von Nullen die Stellenzahl des Radikanden a auf das doppelte, bei gerader Anzahl Stellen, bzw. auf das doppelte* −1*, bei ungerader Stellenzahl, erhöhen.* Für das obige Beispiel müsste man 6 Nullen anhängen und könnte damit 3 weitere Stellen von w berechnen:

$$\sqrt{634250.000000} = 796.398.$$

Jetzt ist $w^2 = 796.398^2 = 634249.774404$, also gerundet gleich $a = 634250$. Somit ist es sinnvoll, die Stellenzahl des Radikanden durch Anhängen von Nullen zu erweitern.

Den Radikanden a stellen wir ziffernweise als **array** $[0..n]$ **of** *integer* dar. Die erste Ziffer ist in $a[1]$ und es wird immer $a[0] = 0$ gesetzt. Für das obige Beispiel ist

$$a[0] = 0,\ a[1] = 6,\ a[2] = 3,\ a[3] = 4,\ a[4] = 2,\ a[5] = 5,\ a[6] = 0.$$

Es bezeichne *zifftot* die Anzahl der Dezimalziffern des Radikanden vor dem Dezimalpunkt. Durch die Anweisung k := (zifftot+1) **mod** 2 wird der

Radikand in Zweiergruppen eingeteilt. Es ist

$$k = \begin{cases} 0, & \text{für } \textit{zifftot} \text{ ungerade} \\ 1, & \text{für } \textit{zifftot} \text{ gerade} \end{cases}$$

und $a[k]$ und $a[k+1]$ sind die ersten beiden Ziffern, d.h. für

$$\textit{zifftot} = \begin{cases} \text{ungerade :} & a[0] = 0 \text{ und } a[1] \\ \text{gerade :} & a[1] \text{ und } a[2]. \end{cases}$$

Die Anzahl der Stellen der Wurzel vor dem Dezimalpunkt ist dann durch *wvorkomma* := *zifftot* **div** $2 + 1 - k$ gegeben. Nach der Berechnung der ersten Ziffer der Wurzel wird k erhöht $k := k + 2$ und zwei neue Ziffern werden an den Rest angefügt:

if $k <=$*zifftot* **then** *rest* := *rest*$*100 + a[k] * 10 + a[k+1]$
else *rest* := *rest*$*100;

wobei für $k >$*zifftot* einfach 2 Nullen angehängt werden. Die neue Ziffer wird durch

ziffer := *rest* **div** *zweia*

geschätzt, wobei $zweia := 20 * w$ ist. Dies funktioniert beim *ersten* Schritt nicht, da noch keine Ziffern von w vorhanden sind und somit $w = zweia = 0$ ist. Am einfachsten setzt man in diesem Fall die Ziffer = 9, sie wird dann von selbst auf die richtige Grösse in der **while**-Schleife korrigiert. Somit lautet die Anweisung:

if *zweia*<> 0 **then** *ziffer* := *rest* **div** *zweia* **else** *ziffer* := 9*;*

Nach diesen Bemerkungen können wir das erste Programm schreiben:

```
(*$B- Quadratwurzel erste Version *)
program w1;
const n = 5;
type longint = integer;
   zahl = array [0..n] of integer;
var a,w : zahl; hilf, rest, wurzel, zweia, ziffer : longint;
   i,wvorkomma,zifftot,k,l : integer;
procedure lieszahl(var a : zahl; var zifftot : integer);
begin
   zifftot := 0;
   writeln('Radikand ? Nach letzter Ziffer CTRL-Z');
   repeat zifftot := zifftot +1; read(a[zifftot])
   until eof
end ;
```

```
begin
   for i := 0 to n do a[i] := 0; rest := 0 ; wurzel := 0;
   lieszahl(a,zifftot);
   k := (zifftot+1) mod 2 ; l := 0;
   wvorkomma := zifftot div 2 + 1 - k ;
   while k < 2*zifftot do
   begin
      if k <=zifftot then rest := rest*100 + a[k] * 10 + a[k + 1]
      else rest := rest*100;
      k := k + 2;
      zweia := 20*wurzel;
      if zweia<> 0 then ziffer := rest div zweia else ziffer := 9;
      hilf :=(zweia+ziffer)*ziffer;
      while rest < hilf do
      begin
         ziffer := ziffer-1;
         hilf :=(zweia+ziffer)*ziffer;
      end ;
      rest := rest- hilf; l := l + 1; w[l] := ziffer;
      wurzel := wurzel*10+ziffer;
      writeln('Rest,Wurzel =',rest: 10, wurzel: 10);
   end ;
   writeln; writeln('Gegebene Zahl');
   for i := 1 to zifftot do write(a[i]);
   writeln; write('Wurzel = ');
   for i := 1 to l do
   begin
      write(w[i]); if i =wvorkomma then write('.');
   end
end.
```

Wegen des beschränkten Integerbereichs von TURBO PASCAL kann dieses Programm nur für kleine Zahlen funktionieren. Für den Radikanden 1205, der natürlich mit Leerschlägen zwischen den Ziffern eingegeben werden muss, erhält man den folgenden Ausdruck auf dem Bildschirm:

```
Radikand ?  Nach letzter Ziffer CTRL-Z
1 2 0 5
Rest,Wurzel =         3         3
Rest,Wurzel =        49        34
Rest,Wurzel =        91       347
Rest,Wurzel =      2159      3471
Gegebene Zahl
1205
```

```
Wurzel = 34.71
```

Damit der Output gedeutet werden kann geben wir das zugehörige Handrechenschema in Tabelle 2.3 an. Die entsprechenden Reste sind fettgedruckt.

$$
\begin{array}{lll}
\sqrt{1}\ 2\ 0\ 5 & = & 34.71 \\
\underline{9} & & \\
3\ 0\ 5 & : & (60{+}4)4 \\
\underline{2\ 5\ 6} & & \\
\mathbf{4\ 9}\ 0\ 0 & : & (680{+}7)7 \\
\underline{4\ 8\ 0\ 9} & & \\
\mathbf{9\ 1}\ 0\ 0 & : & (6940{+}1)1 \\
\underline{6\ 9\ 4\ 1} & & \\
\mathbf{2\ 1\ 5\ 9} & &
\end{array}
$$

Tabelle 2.3: Handrechenschema

Nun wollen wir das Programm $w1$ für mehrfache Genauigkeit umschreiben. Wir haben bereits durch den Typus *longint* unterschieden, welche Variablen mehrfachgenaue Zahlen sind. In UCSD PASCAL könnte man wieder durch die Deklaration

$$\textbf{type}\ longint = integer[36]$$

mit dem Programm $w1$ über 30–stellige Wurzeln berechnen. Die Operationen wie etwa

$$rest := rest - hilf$$

werden wie früher durch Prozeduraufrufe

$$sub(rest,\ hilf,\ rest)$$

ersetzt. Um die neue Ziffer zu ermitteln, ist es nicht nötig, die volle Division

$$ziffer := rest\ \textbf{div}\ zweia$$

durchzuführen. Es genügt, die ersten paar Ziffern von Dividend und Divisor zu betrachten. Dies wurde in der Prozedur *teil* berücksichtigt. Der Radikand wird mittels der Prozedur *lieszahl* als Folge von Charakter eingelesen. Dadurch müssen die einzelnen Ziffern nicht mehr durch Leerschläge getrennt werden. Es ist neu auch möglich, einen Dezimalpunkt und Nachkommastellen einzugeben. Wenn dieser fehlt, wird die Zahl als ganze Zahl betrachtet.

Neu wird die Prozedur *kontrolle* verwendet, welche die Multiplikation $hilf = wurzel \times wurzel$ durchführt. Da die Wurzel l Dezimalstellen aufweist, hat $hilf$ im allgemeinen $2 \times l$ Stellen. Wir berechnen aber nur die ersten $l + 5$ Stellen, da nur diese interessieren. Die Stellen von *hilf* werden durch

Polynommultiplikation berechnet. Die mehrfachgenaue Zahl

$$wurzel = \sum_{i=1}^{l} w_i 10^{l-i}$$

kann als Funktionswert eines Polynoms mit Koeffizienten w_i an der Stelle $x = 10$ angesehen werden. Seien ganz allgemein

$$p(x) = \sum_{i=0}^{n} a_i x^i, \quad q(x) = \sum_{j=0}^{m} b_j x^j$$

zwei Polynome. Dann berechnet sich das Produktpolynom $r(x) = p(x)q(x)$ als

$$r(x) = \sum_{k=0}^{n+m} c_k x^k \tag{2.15}$$

$$= \sum_{i=0}^{n} a_i x^i \sum_{j=0}^{m} b_j x^j = \sum_{i=0}^{n} \sum_{j=0}^{m} a_i b_j x^{i+j} \tag{2.16}$$

Durch Koeffizientenvergleich in Gleichungen (2.15) und (2.16) erhält man die bekannte Rekursionsformel

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}. \tag{2.17}$$

Man kann die Gleichung (2.17) auch erhalten, wenn man die beiden Klammern

$$(a_0 + a_1 x + \cdots + a_n x^n)(b_0 + b_1 x + \cdots + b_m x^m)$$

ausmultipliziert und die Beiträge zur Potenz x^k summiert. Anstatt einen Koeffizienten c_k nach dem anderen so zu berechnen, kann man die Beiträge erzeugen und simultan beim entsprechenden Koeffizienten aufsummieren. Dies ergibt eine programmiertechnisch etwas einfachere Formel. Man initialisiert zuerst $c_k = 0$ und summiert anschliessend alle möglichen Beiträge auf durch

for $i := 0$ **to** n **do**
for $j := 0$ **to** m **do**
$c_{i+j} := c_{i+j} + a_i * b_j;$

Will man nicht alle Koeffizienten berechnen, kann dies durch Beschränken der j-Schleife erreicht werden: wenn z.B. nur die ersten n Koeffizienten c_k gesucht sind, verwendet man

for $j := 0$ **to** $n - i$ **do**

Dies wurde in der Prozedur *kontrolle* so durchgeführt. Man beachte, dass die Koeffizienten w_i dort die Indices $1, 2, \ldots l$ aufweisen. Damit lautet das vollständige Programm:

```
(*$B- Aufgabe 2.30*)
program quadratwurzel;
const n = 300;
type zahl =array [0..n] of integer;
var a,hilf,rest,wurzel,zweia,zero : zahl;
    ziffer,zifftot,wvorkomma,avorkomma,
    imin,k,j,i,l : integer;
    tf : text; stri : string [10];
procedure lieszahl(var a : zahl; var anzstellen,avorkomma : integer);
var c : char; ziffer : boolean;
begin
    avorkomma:= 0; anzstellen:= 0;
    writeln('Radikand ? Nach letzter Ziffer CTRL-Z');
    repeat
        read(c);
        ziffer:= ('0'<= c) and (c <='9');
        if ziffer then
        begin
            anzstellen:=anzstellen+1;
            a[anzstellen] := ord(c) - ord('0') ;
        end ;
        if c ='.' then avorkomma:=anzstellen;
    until eof;
    if avorkomma= 0 then avorkomma:=anzstellen
end ;
procedure teil(a,b : zahl; var z : integer);
var f,g : real;
begin
    f := 0; g := 0; i :=imin;
    while (g < 10) and (i < k) do
    begin
        i := i + 1; f := f * 10 + a[i]; g := g * 10 + b[i];
    end ;
    if g <> 0 then z := round(f/g) else z := 9;
end ;
procedure sub(a,b : zahl; var c : zahl);
var i : integer;
begin
    for i := k downto imin do
    begin
        c[i] := a[i] - b[i];
        if c[i] < 0 then
```

```
        begin
            c[i] := c[i] + 10; a[i − 1] := a[i − 1] − 1;
        end
    end
end ;
procedure uebertrag( var c : zahl);
var i : integer;
begin
    for i := k downto imin do
    while c[i] >= 10 do
    begin
        c[i] := c[i] − 10; c[i − 1] := c[i − 1] + 1;
    end
end ;
procedure mult(a : zahl; b : integer; var c : zahl);
var i : integer;
begin
    for i := k downto imin do c[i] := a[i] ∗ b;
    uebertrag(c)
end ;
function kleiner(a, b : zahl) : boolean;
var i : integer; ungl : boolean;
begin
    i := imin;
    repeat
        i := i + 1; ungl:= a[i] <> b[i];
    until ungl or (i = k);
    kleiner := a[i] < b[i]
end ;
procedure printwurzel;
begin
    for i := 1 to l do
    begin
        write(tf,wurzel[i]); if i =wvorkomma then write(tf,'.');
    end ; writeln(tf)
end ;
procedure kontrolle;
var i,j : integer;
begin
    hilf := zero;
    for i := 1 to l do
    begin
```

```
        for j := 1 to l − i + 5 do
        hilf[i + j] := hilf[i + j]+wurzel[j]∗wurzel[i]
    end ;
    for i := l + 5 downto 2 do
    while hilf[i] >= 10 do
    begin
        hilf[i] :=hilf[i] − 10;
        hilf[i − 1] :=hilf[i − 1] + 1
    end ;
    writeln(tf); writeln(tf); writeln(tf,'Wurzel∗ ∗ 2 =');
    for i := 1 to l + 1 do write(tf,hilf[i])
end ;

begin
    writeln('wohin mit dem Output?');
    readln(stri); assign(tf, stri); rewrite(tf);
    for i := 0 to n do zero[i] := 0;
    rest:=zero; wurzel:=zero; hilf:=zero; a:=zero;
    lieszahl(a,zifftot,avorkomma);
    k := (avorkomma +1) mod 2;
    wvorkomma:= avorkomma div 2 + 1 − k;
    l := 0;

    while k < 2∗zifftot do
    begin
        if k <=zifftot then
        begin
            rest[k + 1] := a[k]; rest[k + 2] := a[k + 1]
        end
        k := k + 2; imin:= k − l − 3; if imin < 0 then imin:= 0;
        zweia := zero;
        for j := 1 to l do zweia[k − l − 1 + j] :=wurzel[j] ∗ 2;
        teil(rest,zweia,ziffer);
        zweia[k] :=ziffer;
        mult(zweia,ziffer,hilf);
        while kleiner(rest,hilf) do
        begin
            ziffer:=ziffer−1; zweia[k] :=ziffer;
            mult(zweia,ziffer,hilf);
        end ;
        sub(rest,hilf,rest);
        l := l + 1; wurzel[l] :=ziffer;
        writeln(tf);writeln(tf,'Rest, Wurzel');
        for i := k − l − 1 to k do write(tf,rest[i]); writeln(tf);
```

```
    printwurzel;
  end ;
  writeln(tf); writeln(tf,'Gegebene Zahl');
  for i := 1 to zifftot do
  begin
    write(tf, a[i]); if i = avorkomma then write(tf,'.');
  end ;
  writeln(tf); writeln(tf); writeln(tf,'Wurzel');
  printwurzel;
  kontrolle; writeln(tf); close(tf)
end.
```

Als Beispiel berechnen wir damit ca. 50 Dezimalstellen von $\sqrt{2}$. Die Kontrolle zeigt, dass richtig gerechnet wurde: das Quadrat der Wurzel ist $1.99\ldots$.

```
Radikand ?  Nach letzter Ziffer CTRL-Z
2.00000000000000000000000000000000000000000000000000
Rest, Wurzel
001
1.
Rest, Wurzel
0004
1.4
Rest, Wurzel
00119
1.41
```

Hier überspringen wir Zwischenoutput. Der Schluss lautet dann:

```
Rest, Wurzel
00135971476893686922599760809891043208165715482946 39
1.41421356237309504880168872420969807856967187537769
Rest, Wurzel
00228343919038393186956257119542673618801417329162364
1.41421356237309504880168872420969807856967187537694
Gegebene Zahl
2.00000000000000000000000000000000000000000000000000
Wurzel
1.41421356237309504880168872420969807856967187537694
Wurzel**2 =
01999999999999999999999999999999999999999999999997
```

Kapitel 3

Nichtlineare Gleichungen

Aufgabe 3.2 *Man schreibe ein Programm zur Lösung von Gleichungen mittels Bisektion. Man modifiziere dabei den Algorithmus 3.2 so, dass er auch funktioniert, wenn $f(a) > 0$ und $f(b) \leq 0$ ist.*

Es gibt hier zwei Möglichkeiten, den allgemeinen Fall auf den Fall $f(a) < 0$ und $f(b) \geq 0$ zurückzuführen. Man kann das Vorzeichen von $f(b)$ am Anfang berechnen: **if** $f(b) > 0$ **then** $sigb := 1$ **else** $sigb := -1$. Nun wird bei $f(b) < 0$ durch

$$\textbf{if } sigb * f(x) > 0 \textbf{ then } b := x \textbf{ else } a := x$$

die Funktion mit dem Faktor -1 multipliziert und somit auf den ersten Fall zurückgeführt. Bei der zweiten Möglichkeit wird eine *boolean* Variable verwendet. Man setzt etwa $fbpos := f(b) > 0$. Der Test, welche Grenze zu verschieben sei, lautet damit

$$\textbf{if } fbpos = (f(x) > 0) \textbf{ then } b := x \textbf{ else } a := x;$$

wobei die Gleichheit zweier *boolean* Ausdrücke geprüft wird. Diese Variante ist etwas schneller, da keine Multiplikation durchgeführt werden muss. Man erhält so folgende Funktion, um eine Nullstelle von $f(x)$ im Intervall (a, b) auf Maschinengenauigkeit zu berechnen:

```
function bisekt(a, b : real) : real;
var fbpos : boolean; x : real;
begin
   fbpos:= f(b) > 0; x := (a + b)/2;
   while (a < x) and (x < b) do
   begin
      if fbpos= (f(x) > 0) then b := x else a := x;
      x := (a + b)/2;
   end ;
   bisekt:= x
end ;
```

Das Hauptprogramm dazu lautet:

```
(*$B- Aufgabe 3.2*)
program bis;
var a,x,b : real;
    tf : text; stri : string [20];
function f(x : real):real;
begin f := (2 + x)/2 * sqrt(2 * x) - 12 end ;
(*$I bisekt*)
begin
    writeln('Output wohin ?'); readln(stri);
    assign(tf, stri); rewrite(tf);
    writeln('a,b eingeben');
    read(a, b); writeln(tf,'Startintervall =',a, b);
    x := bisekt(a, b);
    writeln(tf,'Nullstelle =',x,' f(x) =',f(x));
    close(tf)
end.
```

Aufgabe 3.3 *Wie kann man mittels des Bisektionsalgorithmus die Mantissenlänge eines Computers bestimmen ? Man gebe ein Programm dafür an.*

Pro Iterationsschritt berechnet man beim Lösen von Gleichungen mit der Bisektion eine Dualstelle. Wenn man nun zum Beispiel die Gleichung $f(x) = x - 1 = 0$ löst und die Startwerte $a = 0.8$ und $b = 1.3$ verwendet, muss man nur noch die Anzahl der Iterationsschritte zählen und diese ausdrucken. Das Resultat ist nur dann richtig, wenn die *real* Darstellung der beiden Startwerte a und b *denselben Exponenten* aufweist.

```
(*$B- Aufgabe 3.3*)
program mantisse;
var a,b,x,z : real; i,j : integer;
begin
    a := 0.8; b := 1.3; i := 1; x := (a + b)/2;
    while (a < x) and (x < b) do
    begin
        i := i + 1;
        if x - 1 < 0 then a := x else b := x;
        x := (a + b)/2
    end ;
    writeln('Mantissenlaenge = ',i);
end.
```

Man erhält hier das Resultat: Mantissenlaenge = 40, was gerade mit dem im TURBO PASCAL Manual angegebenen Wert übereinstimmt.

Aufgabe 3.4 *Man löse die Gleichungen*

$$a)\quad x^x = 50 \qquad b)\quad \ln(x) = \cos(x) \qquad c)\quad x + e^x = 0$$

Im folgenden Programm wird durch ein Menue abgefragt, welche Aufgabe gelöst werden soll:

```
(*$B- Aufgabe 3.4*)
program bis;
var a, x, b : real; fall : integer;
    tf : text; stri : string [20];
function f(x : real) : real;
begin
   case fall of
      1 : f := exp(x * ln(x)) - 50;
      2 : f := ln(x) - cos(x);
      3 : f := x + exp(x);
   end
end ;
(*$I bisekt*)
begin
   writeln('Output wohin'); readln(stri); assign(tf, stri); rewrite(tf);
   repeat
      writeln('fertig          -> 0       Aufg. 3.4A-> 1 ');
      writeln('Aufg. 3.4B-> 2       Aufg. 3.4C-> 3 ?');
      writeln('waehle Fall: 0, 1, 2, 3 :'); read(fall);
      if fall <> 0 then
      begin
         case fall of
            1 : begin a := 3; b := 4 end ;
            2 : begin a := 1; b := 2 end ;
            3 : begin a := -1; b := 0 end
         end ;
         writeln(tf,'Startintervall =',a,b);
         x := bisekt(a, b);
         writeln(tf,'Nullstelle =',x,' f(x)=', f(x));
      end
   until fall= 0;
   close(tf)
end.
```

Man erhält damit folgende Lösungen:
a) $x = 3.2872621954$ b) $x = 1.3029640012$ c) $x = -0.56714329041$

Aufgabe 3.5 *Eine Geiss weidet auf einer kreisförmigen Wiese vom Radius r. Der Bauer bindet sie am Rande an einen Pflock an. Man bestimme die Schnurlänge R so, dass die Geiss gerade die Hälfte des Weideplatzes abgrasen kann. Man stelle eine Gleichung für das Verhältnis $x = R/r$ auf und löse sie mittels des Bisektionsalgorithmus.*

Figur 3.1: Geissweide

Verbindet man die Schnittpunkte der beiden Kreise (s. Fig. 3.1), so lautet die Aufgabe, den Winkel α so zu bestimmen, dass die Summe der Flächen der beiden Kreisabschnitte F_1 und F_2 gleich der halben Kreisfläche $\frac{\pi}{2}r^2$ ist. Es ist

$$h = R\sin\alpha = r\sin(\pi - 2\alpha)$$

und daraus

$$\frac{R}{r} = 2\cos\alpha \tag{3.1}$$

Aus der bekannten Formel für den Kreisabschnitt (Sektor − Dreieck) ergibt sich unmittelbar:

$$\begin{aligned} F_1 &= \frac{R^2 2\alpha}{2} - R^2\cos\alpha\sin\alpha \\ &= R^2\left(\alpha - \frac{\sin 2\alpha}{2}\right) \\ F_2 &= \frac{r^2 2(\pi - 2\alpha)}{2} - r^2\frac{\sin(2\pi - 4\alpha)}{2} \\ &= r^2\left(\pi - 2\alpha + \frac{\sin(4\alpha)}{2}\right). \end{aligned}$$

Aus der Gleichung $F_1 + F_2 = r^2 \frac{\pi}{2}$ folgt

$$\frac{R^2}{r^2}\left(\alpha - \frac{\sin 2\alpha}{2}\right) + \pi - 2\alpha + \frac{\sin 4\alpha}{2} = \frac{\pi}{2}$$

und unter Verwendung von Gleichung (3.1) erhält man

$$4\cos^2\alpha\left(\alpha - \frac{\sin 2\alpha}{2}\right) + \frac{\pi}{2} - 2\alpha + \frac{\sin 4\alpha}{2} = 0. \tag{3.2}$$

Mit den Startwerten $a = 0$ und $b = \pi/2$ erhält man als Lösung dieser Gleichung mit dem Bisektionsalgorithmus $\alpha = 0.9528478647$ und daraus $R = 2\cos\alpha\, r = 1.158728473\, r$.

Falls man direkt die Grösse $x = \frac{R}{r}$ berechnen möchte, ersetzt man in der Gleichung (3.2) $\cos\alpha = \frac{x}{2}$ und erhält nach einigen Umformungen die Gleichung

$$\frac{x}{2} = \cos\left(\frac{2x\sqrt{1 - \left(\frac{x}{2}\right)^2} - \pi}{2x^2 - 4}\right)$$

welche mit den Startwerten $a = 1$ und $b = 2$ direkt das gesuchte Verhältnis $x = \frac{R}{r} = 1.158728473$ liefert.

Aufgabe 3.6 *Man bestimme x so, dass*

$$\int_0^x e^{-t^2}\,dt = 0.5$$

ist. Hinweis: Da das Integral analytisch nicht berechnet werden kann, entwickle man den Integranden in seine MacLaurinreihe, integriere gliedweise und suche mittels Bisektion die Nullstelle der entstehenden Potenzreihe.

Setzt man in der MacLaurinreihe von e^x, Gleichung (2.11$*$), $x = -t^2$ und integriert gliedweise, so ergibt sich:

$$\int_0^x e^{-t^2}\,dt = x - \frac{x^3}{1!3} + \frac{x^5}{2!5} - \frac{x^7}{3!7} + \frac{x^9}{4!9} \mp \cdots \tag{3.3}$$

Bei der Berechnung des Funktionswertes der Funktion

$$f(x) = \int_0^x e^{-t^2}\,dt - 0.5$$

muss jedesmal die Reihe (3.3) aufsummiert werden. Bezeichnet man mit

$$ta := (-1)^{i-1}\frac{x^{2i-1}}{(i-1)!} \qquad t := (-1)^{i}\frac{x^{2i+1}}{i!}$$

zwei aufeinanderfolgende Summanden von (3.3), so berechnet sich die neue Partialsumme s der Reihe (3.3) aus der alten sa wie folgt:

$$t := -ta * x * x/i; \quad s := sa + t/(2 * i + 1)$$

Das ganze Programm lautet damit:

```
(*$B- Aufgabe 3.6*)
program integral;
var a,x,b : real;
    tf : text; stri : string [20];
function f(x : real) : real;
var s,t,sa,ta : real; i : integer;
begin
    t := x; s := x; i := 0;
    repeat
        i := i + 1; sa := s; ta := t;
        t := -ta * x * x/i;
        s := sa + t/(2 * i + 1)
    until s = sa;
    f := s - 0.5
end ;
(*$I bisekt*)
begin
    writeln('Output wohin ?');
    read(stri); assign(tf, stri); rewrite(tf);
    a := 0; b := 5; writeln(tf,'Startintervall =',a,b);
    x := bisekt(a, b);
    writeln(tf,'Nullstelle =',x,'f(x) =',f(x));
    close(tf)
end.
```

Als Resultat erhält man hier $x = 0.55103942761$.

Aufgabe 3.7 *Binärer Suchprozess: Gegeben sei eine sortierte Zahlenfolge:*

$$x_1 \leq x_2 \leq \cdots \leq x_n$$

und eine neue Zahl z. Man schreibe ein Programm, das einen Index i bestimmt so, dass entweder $x_{i-1} < z \leq x_i$ oder $i = 1$ oder $i = n+1$ gilt. Diese Aufgabe kann gelöst werden, wenn man die Funktion

$$f(i) = x_i - z$$

betrachtet und deren 'Nullstelle' mittels Bisektion sucht.

In folgenden Programm wird zuerst die geordnete Zahlenfolge x_i eingelesen, anschliessend die neue Zahl z. Da die Anzahl der x_i nicht vorgegeben wird und die Eingabe der Folge durch CTRL-Z beendet wird, muss das Input File mittels *reset(input)* wieder initialisiert werden, damit z eingelesen werden kann. Danach bestimmt man mittels Bisektion das Intervall, wo z hineinpasst.

```
(*$B- Aufgabe 3.7*)
program binaersuch;
const nn = 20;
var x : array [1..nn] of real; z : real;
    a, b, i, j, n : integer;
begin
    writeln('Geordnete Zahlenfolge x1 <= x2 <= x3 ... eingeben,',
            ' mit CTRL-Z aufhoeren ');
    n := 0; repeat n := n + 1; read(x[n]) until eof; reset(input); writeln;
    writeln('gegebene Folge'); for i := 1 to n do writeln(x[i]);
    writeln; writeln('Neue Zahl z eingeben'); read(z);

    if z < x[1] then i := 1
    else
    if z > x[n] then i := n + 1
    else
    begin
        a := 1; b := n;
        repeat
            i := (a + b) div 2;
            if x[i] < z then a := i else b := i;
        until a + 1 = b;
        i := b;
    end ;
    for j := 1 to i - 1 do writeln(x[j]);
    writeln('       ',z);
    for j := i to n do writeln(x[j]);
end.
```

Aufgabe 3.8 *Man bestimme x so, dass das folgende Maximum angenommen wird:*

$$\max_{0<x<\frac{\pi}{2}} \left(\frac{1}{4\sin x} + \frac{\sin x}{2x} - \frac{\cos x}{4x} \right)$$

Es ist

$$g(x) = \frac{1}{4\sin x} + \frac{\sin x}{2x} - \frac{\cos x}{4x}$$
$$\Rightarrow g'(x) = \frac{\cos x}{4x}\left(2 - \frac{x}{\sin^2 x} + \frac{1}{x}\right) + \frac{\sin(x)}{4x}\left(1 - \frac{2}{x}\right)$$

und die Lösung von $g'(x) = 0$ ist die gesuchte Stelle des Maximums. Mit den Startwerten $a = 0.001$ (0 geht nicht, weil $\sin(x)$ im Nenner ist!) und $b = 1.5$ erhält man mit dem folgenden Programm

```
(*$B- Aufgabe 3.8*)
program max;
var x:real;
function f(x : real):real;
begin
   f := cos(x)/4/x * (2 - x/sqr(sin(x)) + 1/x) + sin(x)/4/x * (1 - 2/x)
end ;
(*$I bisekt*)
begin
   x := bisekt(0.001, 1.5);
   writeln('xmax =',x,'f(xmax) =',1/4/sin(x) + sin(x)/2/x - cos(x)/4/x)
end.
```

das Resultat $x_{max} = 1.0311580967$ und $f(x_{max}) = 0.58282216245$.

Aufgabe 3.9 *Man bringe die folgenden Gleichungen auf eine Iterationsform und versuche damit die Lösungen zu berechnen. Um geeignete Startwerte zu ermitteln, bringe man die Gleichung auf eine Form $h(x) = g(x)$ wo h und g bekannte, leicht zu skizzierende Funktionen sind und lese die x-Koordinate der Schnittpunkte ab.*

a) $3x - \cos x = 0$
b) $2\sin x + e^x = 0$
c) $x \ln x - 1 = 0$
d) $x + \sqrt{x} = 1 + x^2$
e) $3x^2 + \tan x = 0$

a) Aus dem Graphen der beiden Seiten der Gleichung $3x = \cos x$ ist ersichtlich, dass nur ein Schnittpunkt bei $x \approx 0.3$ existiert. Zwei Iterationsformen sind daher naheliegend:

$$1.\quad x = \frac{\cos x}{3} \qquad 2.\quad x = \arccos(3x).$$

Die zweite Form liefert eine divergente Folge und mit der ersten erhält man die Lösung $x = 0.3167508288$.

b) Hier bringt man die Gleichung auf die Form $\sin x = -0.5e^x$ und aus den Graphen der beiden Seiten liest man für die Schnittpunkte folgende Näherungswerte ab:

$$s_1 \approx -0.3, \qquad s_2 \approx -3, \qquad s_k \approx -k\pi, \quad k = -2, -3, \ldots$$

Wird die Gleichung nach dem x auf der rechten Seite aufgelöst, so ergibt sich die Iterationsform $x = F(x) = \ln(-2\sin x)$, welche jedoch nicht geeignet ist, weil $|F'(s_k)| > 1$ ist.

Dagegen liefert die Auflösung nach dem x auf der linken Seite gute Iterationsformen. Man muss hier aber sorgfältig vorgehen:

$$\begin{aligned} sin(x) &= -\frac{e^x}{2} \\ \Rightarrow x &= \left\{ \begin{array}{l} \arcsin\left(-\dfrac{e^x}{2}\right) \\ \pi - \arcsin\left(-\dfrac{e^x}{2}\right) \end{array} \right\} + 2\pi k, \quad k = 0, \pm 1, \pm 2, \ldots \end{aligned}$$

Für die Bestimmung von s_1 muss $x = \arcsin\left(-\frac{e^x}{2}\right)$ verwendet werden. Man erhält damit $s_1 = -0.35732741132$. Für $k = -1$ erhält man ferner die Iterationsform $x = -\pi - \arcsin\left(-\frac{e^x}{2}\right)$, welche eine gegen $s_2 = -3.119501258$ konvergente Folge liefert.

c) Aus $\ln x = \frac{1}{x}$ sieht man, dass nur eine Lösung bei $s \approx 1.8$ vorhanden ist. Die Iterationsform $x = 1/\ln x$ liefert eine divergente Folge. Mit der Umkehrfunktion

$$x = e^{\frac{1}{x}}$$

lässt sich jedoch $s = 1.763222834$ berechnen.

d) Von den Schnittpunkten der Graphen der beiden Seiten der Gleichung

$$\sqrt{x} = 1 - x + x^2 = \left(x - \frac{1}{2}\right)^2 + \frac{3}{4}$$

liest man die exakte Lösung $x_1 = 1$ sowie $x_2 \approx 0.5$ ab. Die Iterationsform

$$x = x^2 - \sqrt{x} + 1$$

eignet sich für die Berechnung von x_2 und liefert den Wert $x_2 = 0.569840291$.

e) Aus den Graphen von $\tan x = -3x^2$ sieht man, dass wieder unendlich viele Lösungen vorhanden sind, nämlich

$$x_0 = 0, \quad x_1 \approx -0.5, \quad \text{und } s_k \approx \pm\frac{\pi}{2} + k\pi,\ k = 0, \pm1, \pm2, \ldots$$

Die Auflösung der Gleichung nach dem Argument des Tangens ergibt

$$x = \arctan(-3x^2) + k\pi, \quad k = 0, \pm1, \pm2, \ldots$$

und liefert Iterationsformen, mit welchen die Lösungen s_k berechnet werden können : zum Beispiel ist

$$\text{für } k = 0\ s_0 = -1.403060421 \text{ und für } k = 1\ s_1 = 1.687342883.$$

Die Lösung x_1 erhält man durch Auflösen der Gleichung nach dem x auf der rechten Seite. Die Iterationsform

$$x = -\sqrt{-\frac{\tan x}{3}}$$

liefert die Lösung $x_1 = -0.3474257645$.

Aufgabe 3.10 *Was wird mit den folgenden Iterationsformen berechnet und wie konvergiert die Folge ?*

a) $x_0 = 1,\ x_{k+1} = 0.2(4x_k + \frac{a}{x_k})$

b) $x_0 = 1,\ x_{k+1} = 0.5(x_k + \frac{a}{x_k})$

c) $x_0 = 1,\ x_{k+1} = x_k(x_k^2 + 3a)/(3x_k^2 + a)$

a) Die Gleichung $x = F(x) = 0.2(4x + \frac{a}{x})$ lässt sich analytisch lösen und ergibt $x = \pm\sqrt{a}$. Mit dem Startwert $x_0 = 1$ konvergiert die Folge gegen $\sqrt{a}$. Zur Untersuchung der Konvergenz berechnen wir

$$F'(x) = 0.2\left(4 - \frac{a}{x^2}\right) \quad \Rightarrow F'(\sqrt{a}) = 0.6.$$

Die Folge konvergiert damit *linear* gegen $\sqrt{a}$.

b) Die Gleichung $x = F(x) = 0.5(x + \frac{a}{x})$ lässt sich auch analytisch lösen und ergibt dieselbe Lösung $x = \pm\sqrt{a}$. Wegen

$$F'(x) = 0.5\left(1 - \frac{a}{x^2}\right) \quad \Rightarrow F'(\sqrt{a}) = 0$$
$$F''(x) = \frac{a}{x^3} \qquad \Rightarrow F''(\sqrt{a}) \neq 0,$$

ist die Konvergenz *quadratisch*.

c) Die Gleichung $x = F(x) = x(x^2 + 3a)/(3x^2 + a)$ hat auch die Lösung $x = \pm\sqrt{a}$. Wegen

$$F'(x) = 3\left(\frac{x^2 - a}{3x^2 + a}\right)^2 \quad \Rightarrow F'(\sqrt{a}) = 0$$

und

$$F''(x) = 48ax\frac{x^2 - a}{(3x^2 + a)^3} \quad \Rightarrow F''(\sqrt{a}) = 0$$

ist die Konvergenz mindestens *kubisch*.

Aufgabe 3.11 *Was soll unter den folgenden Ausdrücken verstanden werden ?*

$$a)\ 1 + \frac{1}{\sqrt{1 + \frac{1}{\sqrt{1+\cdots}}}} \qquad b)\ \sqrt[3]{1 + \sqrt[3]{1 + \sqrt[3]{1 + \cdots}}}$$

a) Bezeichnet man mit x den angegebenen Ausdruck, so erfüllt x die Gleichung

$$x = 1 + \frac{1}{\sqrt{x}}. \tag{3.4}$$

Fasst man die Gleichung (3.4) als Iterationsform auf, so ist

$$x_1 = 1 + \frac{1}{\sqrt{x}_0}$$
$$x_2 = 1 + \frac{1}{\sqrt{x}_1} = 1 + \frac{1}{\sqrt{1 + \frac{1}{\sqrt{x}_0}}}$$

und es ist $lim_{k\to\infty} x_k = x$. Die Folge konvergiert tatsächlich und man erhält $x = 1.7548776662$.

b) Bezeichnet man hier auch wieder mit x den angegebenen Ausdruck, so erfüllt x die Gleichung

$$x = \sqrt[3]{1 + x}.$$

Diese Gleichung liefert als Iterationsform auch eine konvergente Folge mit dem Grenzwert $x = 1.3247179572$.

Aufgabe 3.12 *Man bestimme graphisch Näherungswerte für die drei Lösungen der Gleichung*

$$2x^3 - 5.2x^2 - 4.1x + 3.1 = 0. \tag{3.5}$$

Indem die Gleichung nach x, x^2 bzw. x^3 aufgelöst wird, erhält man drei Iterationsformen. Man untersuche, welche Form für welche Lösung verwendet werden kann und berechne die Lösungen auf Maschinengenauigkeit.

Die drei Lösungen der Gleichung sind $s_1 = -1$, $s_2 = 0.5$ und $s_3 = 3.1$. Auflösung der Gleichung (3.5) nach $4.1x$ und Division durch den Koeffizienten 4.1 liefert die Iterationsform

$$x = F(x) = \frac{(2x - 5.2)x^2 + 3.1}{4.1},$$

welche nur für die Berechnung von $s_2 = 0.5$ verwendet werden kann. Die Konvergenz ist allerdings sehr schlecht. Wegen

$$F'(x) = \frac{6x^2 - 10.4x}{4.1} \quad \Rightarrow F'(0.5) = -0.9024$$

hat man lineare Konvergenz und man braucht etwa 22 Iterationsschritte um eine Dezimalstelle zu berechnen.

Wenn wir die Gleichung (3.5) nach x^2 auflösen, erhalten wir

$$x = \pm\sqrt{\frac{(2x^2 - 4.1)x + 3.1}{5.2}}. \tag{3.6}$$

Wählt man das positive Vorzeichen, so kann mit der Iterationsform die Lösung $s_2 = 0.5$ berechnet werden. Die Konvergenz ist wegen $|F'(0.5)| = 0.5$ besser.

Wählt man in der Iterationsform (3.6) das negative Vorzeichen, so konvergiert die Folge linear ($F'(-1) = -0.183$) gegen die Lösung $s_1 = -1$.

Löst man schliesslich die Gleichung (3.5) nach x^3 auf, ergibt sich

$$x = \sqrt[3]{(2.6x + 2.05)x - 1.55}$$

und man kann mit dieser Iterationsform die Lösung $s_3 = 3.1$ berechnen. Die Konvergenz ist wieder linear mit dem Faktor $F'(3.1) = 0.630$.

Aufgabe 3.13 *Die Iteration*

$$x_0 = 1, \quad x_{k+1} = \cos x_k, \quad k = 0, 1, \ldots \tag{3.7}$$

konvergiert gegen $s = 0.7390851332$. *Wieviele Schritte wären nötig, um* s *auf 100 Dezimalstellen zu berechnen ?*

Die Konvergenz der Folge (3.7) ist linear mit dem Faktor

$$F'(0.7390851332) = -0.6736120292.$$

Aus dem Fehlergesetz (3.16*) kann k, die erforderliche Anzahl Schritte, abgeschätzt werden:

$$|e_k| \sim |F'(s)|^k |e_0| = 0.6736120292^k \times 0.2609148668 < 10^{-100}$$

$$\Rightarrow k > \frac{-100 - \log 0.2609148668}{\log 0.6736120292} = 579.38$$

Also muss $k \geq 580$ sein.

Aufgabe 3.14 *Die Funktion* $f(x) = xe^x - 1$ *hat eine einzige Nullstelle, etwa bei* $x \approx 0.5$. *Man bilde die Iterationsform*

$$x = x + kf(x) =: F(x) \tag{3.21*}$$

und bestimme k *so, dass* $F'(0.5) = 0$ *wird. Anschliessend berechne man die Nullstelle mittels Iteration (3.21*).*

Die Iterationsfunktion von Gleichung (3.21*) ist $F(x) = x + kf(x)$. Aus $F'(x) = 1 + kf'(x) = 0$ kann durch Einsetzen eines Näherungswertes die Konstante k bestimmt werden. Hier wird $k = -1/f'(0.5) = -0.4043537731$. Es genügt etwa $k = -0.4$ zu setzen und man erhält die Iterationsform

$$x = x - 0.4(xe^x - 1)$$

mit welcher die Lösung $x = 0.56714329041$ mit guter linearer Konvergenz berechnet werden kann.

Aufgabe 3.15 *Sei $x = F(x)$ eine Iterationsform, bei der die Folge $\{x_k\}$ divergiert, weil $|F'(s)| > 1$ ist. Man zeige, dass dann die Iterationsform*

$$x = F^{[-1]}(x)$$

eine gegen den Fixpunkt von $x = F(x)$ konvergente Folge liefert. Anwendung: Es soll die erste positive Lösung von $x - \tan x = 0$ berechnet werden. Die naheliegende Iterationsform:

$$x = \tan x$$

konvergiert nicht, aber die Umkehrfunktion

$$x = \arctan(x) + \pi$$

liefert eine konvergente Folge.

Wenn man die Identität

$$F^{[-1]}(F(x)) = x$$

nach x differenziert erhält man

$$\Big(F^{[-1]}(F(x))\Big)' F'(x) = 1$$

und daraus für den Fixpunkt $s = F(s)$ die Beziehung:

$$F^{[-1]}(s)' = \frac{1}{F'(s)}. \tag{3.8}$$

Wenn nun $|F'(s)| > 1$ ist, folgt aus dieser Gleichung, dass $|F^{[-1]}(s)'| < 1$ ist und dass somit die Iteration konvergiert.

Aufgabe 3.16 *Die Gleichung von Aufgabe 3.6*

$$\int_0^x e^{-t^2} dt = 0.5$$

kann mit der Iterationsform

$$x = x + 0.5 - \int_0^x e^{-t^2} dt$$

gelöst werden. Warum haben wir Konvergenz ?

Die Ableitung der Iterationsfunktion $F(x)$ ergibt

$$F(x) = x + 0.5 - \int_0^x e^{-t^2} dt \quad \Rightarrow F'(x) = 1 - e^{-x^2}.$$

Für $x = 0$ ist $F'(0) = 0$ und für $x > 0$ liegt die Ableitung im Intervall $0 < F'(x) < 1$. Somit konvergiert die Iteration linear.

Aus $F(0) = 0.5$ folgt, dass genau ein Fixpunkt $s > 0$ der Iterationsform $x = F(x)$ existiert. Beginnt man die Iteration mit dem Startwert $x_0 = 0$, so konvergiert die Folge $\{x_k\}$ *monoton wachsend gegen* s. Wir können daher im nachfolgenden Programm maschinenunabhängig die Iteration dann abbrechen, wenn numerisch $x_{k+1} \leq x_k$ gilt.

```
(* Aufgabe 3.16*)
program itera;
var xa,x : real;
function integral(x : real) : real;
var s,t,sa,ta : real; i : integer;
begin
   t := x; s := x; i := 0;
   repeat
      i := i + 1; sa := s; ta := t; t := −ta ∗ x ∗ x/i;
      s := sa + t/(2 ∗ i + 1)
   until s = sa;
   integral:= s
end ;
begin
   x := 0;
   repeat
      xa := x; x := x + 0.5− integral(x);
      writeln(x)
   until xa >= x
end.
```

Mit diesem Programm erhält man die Lösung $x = 0.55103942761$.

Aufgabe 3.17 *Man zeige, dass für Startwerte im 'nahen Konvergenzbereich' das folgende maschinenunabhängige Abbruchkriterium benützt werden kann: Iteration abbrechen, falls* $|x_{k+1} - x_k| \geq |x_k - x_{k-1}|$.

Der 'nahe Konvergenzbereich' ist jene Umgebung von s, in der $|F'(x)| < 1$ ist. Es ist

$$x_{k+1} = F(x_k) \quad \text{und} \quad x_k = F(x_{k-1}).$$

Durch Subtraktion der beiden Gleichungen erhält man unter Benützung des Mittelwertsatzes

$$x_{k+1} - x_k = F(x_k) - F(x_{k-1}) = F'(\xi)(x_k - x_{k-1}),$$

wobei ξ zwischen x_k und x_{k-1} liegt. Im 'nahen Konvergenzbereich' ist $|F'(\xi)| < 1$ und damit gilt theoretisch

$$|x_{k+1} - x_k| < |x_k - x_{k-1}|. \tag{3.9}$$

Wegen der endlichen Arithmetik kann das $<$-Zeichen nicht für immer bestehen. Es muss der Fall eintreten, wo infolge der Rundungsfehler die linke Seite von Gleichung (3.9) grösser gleich der rechten wird.

Dieses an und für sich schöne Abbruchkriterium hat einen Haken. Die Schwierigkeit besteht natürlich darin, zu erkennen, wann die Iteration im 'nahen Konvergenzbereich' ist und eine solche monotone Abnahme stattfindet. Wenn man es anwendet, muss es durch weitere Kriterien ergänzt werden.

Aufgabe 3.18 *Man löse die folgenden Gleichungen mit der Newtonmethode*
a) $x + e^x = 0$
b) $\ln(x) = \cos(x)$
In beiden Fällen schreibe man die Iterationswerte auf und beobachte die Konvergenz.

Mit den Startwerten $x_0 = 0$ für Aufgabe a) und $x_0 = 1$ für Aufgabe b) erhält man die Werte

Aufgabe a)	Aufgabe b)
|0.0000000000E+00	1|.0000000000E+00
−5|.0000000000E−01	1.2|934079930E+00
−5.6|631100320E−01	1.3029|554729E+00
−5.67143|16503E−01	1.3029640012|E+00
−5.6714329041|E−01	1.3029640012|E+00
−5.6714329041|E−01	

Durch den senkrechten Strich sind die richtigen Dezimalstellen der Näherungswerte angezeigt. Man sieht an der ungefähren Stellenverdoppelung pro Iterationsschritt die quadratische Konvergenz.

Aufgabe 3.19 *Gleich wie Aufgabe 3.6. Man bestimme x so, dass*

$$f(x) = \int_0^x e^{-t^2} dt - 0.5 = 0.$$

Da einerseits eine Funktionsauswertung von f viel Rechenaufwand erfordert (Taylorreihe aufsummieren), andrerseits aber Ableitungen von f leicht berechenbar sind, lohnt es sich, das Newton- oder Halleyverfahren anzuwenden.

Im folgenden Programm berechnen wir die Lösung mit beiden Verfahren. Es ist

$$f'(x) = e^{-x^2} \quad \text{und} \quad f''(x) = -2xe^{-x^2}.$$

Aus den Ungleichungen $f'(x) > 0$ und $f''(x) < 0$ für $x > 0$ folgt, dass mit dem Startwert $x_0 = 0$ die Folge der Näherungswerte $\{x_k\}$ *monoton wachsend* gegen die Lösung konvergiert. Also kann das maschinenunabhängige Abbruchkriterium: Aufhören, wenn numerisch $x_{k-1} \geq x_k$ wird, benützt werden.

```
(*$B- Aufgabe 3.19*)
program newtonhalley;
var xa, x, f0, f1, f2 :real;
function f(x : real) : real;
var s,t,sa,ta : real; i : integer;
begin
   t := x; s := x; i := 0;
   repeat
      i := i + 1; sa := s; ta := t; t := -ta * x * x/i;
      s := sa + t/(2 * i + 1)
   until s = sa;
   f := s - 0.5
end ;
function fs(x : real) : real;
begin fs:= exp(-sqr(x)) end ;
function fss(x : real):real;
begin fss := -exp(-sqr(x)) * 2 * x end ;
begin
   x := 0; writeln('mit Newton');
   repeat
      xa := x; x := x - f(x)/fs(x);
      writeln(x)
   until xa >= x;
   x := 0; writeln('mit Halley');
```

```
   repeat
      xa := x;
      f0 := f(x); f1 :=fs(x); f2 :=fss(x);
      x := x − f0/f1/(1 − 0.5 ∗ f0/f1 ∗ f2/f1);
      writeln(x)
   until xa >= x
end.
```

Dieses Programm liefert den Output:

```
mit Newton
5.0000000000E-01
5.4971617187E-01
5.5103846475E-01
5.5103942761E-01
5.5103942761E-01
mit Halley
5.0000000000E-01
5.5098352471E-01
5.5103942761E-01
5.5103942761E-01
5.5103942761E-01
```

Aufgabe 3.20 *Von einem Dreieck weiss man, dass der Winkel β doppelt so gross ist wie der Winkel α. Ferner ist die Höhe $h_c = 5$ und der Inkreisradius $\rho = 2$ gegeben. Man bestimme die Seiten des Dreiecks.*

Wir berechnen hier die Seite c des Dreiecks auf zwei Arten: Unter Benützung von $\beta = 2\alpha$ und der Höhe h_c folgt

$$c = \frac{h}{\tan\alpha} + \frac{h}{\tan 2\alpha}.$$

Wenn wir andrerseits den Inkreisradius ρ benützen und davon Gebrauch machen, dass der Inkreismittelpunkt Schnittpunkt der Winkelhalbierenden ist, ergibt sich

$$c = \frac{\rho}{\tan\alpha} + \frac{\rho}{\tan\frac{\alpha}{2}}.$$

Setzt man nun die beiden Ausdrücke für c einander gleich, so ergibt sich eine Gleichung für den unbekannten Winkel α:

$$f(\alpha) = \frac{h}{\tan\alpha} + \frac{h}{\tan 2\alpha} - \frac{\rho}{\tan\alpha} - \frac{\rho}{\tan\frac{\alpha}{2}} = 0 \tag{3.10}$$

Die Ableitung der Funktion $f(\alpha)$ ist ein schon etwas komplizierter Ausdruck, deswegen ist es am einfachsten, die Lösung der Gleichung (3.10) mit der Bisektion zu berechnen. Man erhält folgendes Programm:

```
(*$B- Aufgabe 3.20*)
program dreieck;
const h=5;ro=2;
var alpha,beta,gamma,a,b,c,pi:real;
function f(x :real):real;
begin
   f := h * (cos(2 * x)/sin(2 * x) + cos(x)/sin(x))
      -ro * (cos(x/2)/sin(x/2) + cos(x)/sin(x));
end ;
(*$I bisekt*)
begin
   pi := 4*arctan(1); alpha:= bisekt(0.01, 1.5);
   beta:= 2*alpha; gamma:= pi-alpha-beta;
   writeln('Winkel in Grad: Alpha =',alpha*180/pi : 6 : 3,
   ' Beta =',beta*180/pi : 6 : 3,' Gamma =',gamma*180/pi : 6 : 3);
   c := h * (cos(2*alpha)/sin(2*alpha) + cos(alpha)/sin(alpha));
   a := sin(alpha)/sin(gamma) * c; b := sin(beta)/sin(gamma) * c;
   writeln('Seiten: a =',a : 6 : 3,' b =',b : 6 : 3,' c =',c : 6 : 3);
end.
```

Das Programm liefert die Lösung:

```
Winkel in Grad:  Alpha =33.557 Beta =67.115 Gamma =79.328
Seiten:  a = 5.427  b = 9.045  c = 9.648
```

Aufgabe 3.21 *Ein Öltank hat die Gestalt eines liegenden Zylinders mit Radius $r = 1.2m$ und Länge $l = 5m$. Wie hoch steht das Öl, wenn der Tank zu einem Viertel gefüllt ist ?*

Das Volumen des Öls berechnet sich (s. Figur 3.2) als Fläche des Kreisabschnitts mal Länge l:

$$V = \frac{r^2}{2}(\alpha - \sin\alpha)l$$

Setzt man diesen Audruck gleich einem Viertel des Zylindervolumens ($\frac{\pi}{4}r^2l$), so erhält man nach der Division durch r^2 und l die Gleichung

$$f(\alpha) = \alpha - \sin\alpha - \frac{\pi}{2} = 0.$$

Figur 3.2: Öltank

Wir lösen diese Gleichung mit der Newtonmethode, da die Ableitung einfach erhältlich ist. Die gesuchte Höhe ergibt sich aus α durch

$$h = r - r\cos\frac{\alpha}{2}.$$

```
(*$B- Aufgabe 3.21*)
program oeltank;
const r = 1.2; (*L = 5 wird nicht gebraucht*)
var alphaa,alpha,pi,h,voll : real;
function f(alpha : real) : real;
begin f :=alpha-sin(alpha) - pi * 2*voll end ;
function fs(alpha:real) :real;
begin fs:= 1 - cos(alpha) end ;
begin
   pi := 4*arctan(1);
   writeln('Wie voll ist der Tank (0.25) ?'); read(voll);
   writeln('Itearationen fuer den Winkel alpha:');
   alpha:= 3;
   repeat
      alphaa:=alpha;
      alpha:=alpha-f(alpha)/fs(alpha);
      writeln(alpha*180/pi);
   until abs(alphaa-alpha) <abs(alpha) * 1E-9;
   h := r * (1 - cos(alpha/2));
   writeln('Bei',100*voll: 6 : 1,
      '% Fuellung steht das Oel',h : 6 : 3,' m hoch');
end.
```

Aufgabe 3.22 *Ein Rohr vom Radius $r = 4cm$ wird an einer Schnur der Länge $L = 30cm$ aufgehängt (Siehe Figur 3.3). Wie gross ist der Abstand h des Rohrs von der Decke.*

Figur 3.3: Rohraufgabe

Ausgedrückt durch den Winkel α erhält man für die Schnurlänge

$$L = 2\pi r - 2\alpha r + 2r\tan\alpha$$

oder etwas vereinfacht die folgende Gleichung für den Winkel α:

$$f(\alpha) = \alpha - \tan\alpha + \frac{L}{2r} - \pi = 0.$$

Da die Ableitung $f'(\alpha)$ leicht berechenbar ist, verwenden wir wieder das Newtonverfahren mit Startwert $\alpha = 1$ zur Lösung. Nach der Berechnung von α erhält man die gesuchte Höhe durch

$$h = r\left(\frac{1}{\cos\alpha} - 1\right).$$

Für $L = 30$cm und $r = 4$cm wird $\alpha = 1.020104471$ und $h = 3.644$cm.

Aufgabe 3.23 *Man bestimme a so, dass $\int_0^1 e^{at}dt = 2$.*

Es ist

$$\int_0^1 e^{at}dt = \frac{1}{a}e^{at}\Big]_0^1 = \frac{1}{a}e^a - \frac{1}{a}$$

Somit ist die Lösung der Gleichung

$$f(a) = \frac{e^a - 1}{a} - 2 = 0$$

gesucht. Wieder kann man das Newtonverfahren anwenden und erhält $a = 1.2564312086$.

Aufgabe 3.24 *Es sei* $p = 0.9$ *und*

$$\{a_j\} = \{0.1, 0.5, 1.0, 0.2, 5.0, 0.3, 0.8\}.$$

Gesucht ist ein $x > 0$*, so dass*

$$\prod_{j=1}^{n}(1 + xa_j) = 1 + p.$$

Hinweis: Durch Logarithmieren erhält man die Gleichung

$$f(x) = \sum_{j=1}^{n} \ln(1 + xa_j) - \ln(1 + p) = 0,$$

welche gut mittels des Newtonverfahrens gelöst werden kann. Aus der Diskussion von $f''(x)$ *ergibt sich ein auf Monotonie beruhendes gutes Abbruchkriterium !*

Es ist hier

$$f'(x) = \sum_{j=1}^{n} \frac{a_j}{1 + xa_j} > 0$$

$$f''(x) = -\sum_{j=1}^{n} \frac{a_j^2}{(1 + xa_j)^2} < 0$$

und aus diesen Ungleichungen folgt, dass die vom Newtonverfahren erzeugte Folge mit dem Startwert $x = 0$ monoton wachsend gegen die gesuchte Lösung konvergiert.

```
(*$B- Aufgabe 3.24*)
program produkt;
const nn = 20;
var xa, x, p : real;
    n,i : integer; a : array [1..nn] of real;
    tf : text; stri : string [10];
function f(x : real) : real;
var s : real; j : integer;
begin
    s := -ln(1 + p);
    for j := 1 to n do s := s + ln(1 + x * a[j]);
    f := s
end ;
function fs(x :real):real;
```

```
var s :real; j :integer;
begin
   s := 0;
   for j := 1 to n do s := s + a[j]/(1 + x * a[j]);
   fs:= s
end ;
begin
   writeln('Wohin mit dem Output?');
   readln(stri); assign(tf,stri); rewrite(tf);
   writeln('p=?'); read(p);
   writeln('a[1],...,a[n] eingeben mit CTRL-Z aufhoeren');
   n := 0; repeat n := n + 1; read(a[n]) until eof;
   writeln(tf,'p=',p : 5 : 2,' Koeffizienten:');
   for i := 1 to n do write(tf, a[i] : 4 : 1);
   writeln(tf); writeln(tf,'Iterationen fuer x :');
   x := 0;
   repeat
      xa := x; x := x - f(x)/fs(x); writeln(tf,x)
   until xa>= x;
   writeln(tf); writeln(tf,'x = ',x,'  f(x) = ',f(x));
   close(tf)
end.
```

Mit den in der Aufgabenstellung angegebenen Daten liefert das Programm den Output:

```
p= 0.90 Koeffizienten:
0.1 0.5 1.0 0.2 5.0 0.3 0.8
Iterationen fuer x:
8.1247327364E−02
9.2632232122E−02
9.2780846830E−02
9.2780870979E−02
9.2780870980E−02
9.2780870980E−02
x =   9.2780870980E−02  f(x) =  −1.8189894035E−12
```

Aufgabe 3.25 *Man verwende das Halley-Verfahren, um eine Funktionsprozedur zu schreiben, mit welcher man Quadratwurzeln berechnen kann.*

Die Halley-Iterationsfunktion für $f(x) = 0$ lautet

$$x = F(x) = x - \frac{f}{f'}\frac{1}{1 - \frac{ff''}{2f'^2}} = x - \frac{ff'}{f'^2 - \frac{1}{2}ff''}.$$

Setzt man nun $f(x) = x^2 - a$, $f'(x) = 2x$ und $f''(x) = 2$ ein, so ergibt sich

$$\begin{aligned} F(x) &= x - \frac{(x^2 - a)2x}{4x^2 - \frac{1}{2}(x^2 - a)2} \\ &= x - \frac{2x^3 - 2ax}{3x^2 + a} \\ &= \frac{3ax + x^3}{3x^2 + a} = x\left(\frac{x^2 + 3a}{3x^2 + a}\right) \end{aligned}$$

Beginnt man wie bei Algorithmus 3.4 mit dem Startwert $x = (1 + a)/2$, so konvergiert die Folge der Näherungen monoton fallend gegen $x = \sqrt{a}$. Dies kann wieder für ein maschinenunabhängiges Abbruchkriterium benützt werden.

```
(*$B- Aufgabe 3.25*)
program wurzel;
var x : real;

function quadratwurzel(a :real) : real;
var xneu,xalt: real;
begin
   xneu:= (1 + a)/2;
   repeat
      xalt:=xneu; xneu:=xalt*(sqr(xalt) + 3 * a)/(3*sqr(xalt) + a)
   until xneu>=xalt;
   quadratwurzel:=xneu
end ;
begin
   repeat
      writeln('Berechnung der Quadratwurzel, x eingeben'); read(x);
      writeln('exakt',sqrt(x),'  mit Halley ',quadratwurzel(x))
   until eof
end.
```

Aufgabe 3.26 *Das Newtonverfahren konvergiert quadratisch für einfache Nullstellen einer Funktion f. Wie steht es mit der Konvergenz, wenn $f'(s) = 0$ und $f''(s) \neq 0$? Man analysiere die Iterationsfunktion*

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

für diesen Fall.

Es ist

$$F'(x) = 1 - \frac{f'(x)^2 - f''(x)f(x)}{f'(x)^2} = \frac{f''(x)f(x)}{f'(x)^2} \tag{3.11}$$

und für $x = s$ ist F' unbestimmt, weil Zähler und Nenner verschwinden. Wendet man die Regel von de l'Hospital an, um den Grenzwert für $x \to s$ zu berechnen, so wird

$$F'(s) = \lim_{x \to s} \frac{f''(x)f'(x) + f'''(x)f(x)}{2f'(x)f''(x)}.$$

Immer noch sind Zähler und Nenner für $x = s$ null, so dass nochmals die Regel von de l'Hospital angewendet werden muss:

$$\begin{aligned} F'(s) &= \lim_{x \to s} \frac{f'''(x)f'(x) + f''(x)^2 + f^{(4)}(x)f(x) + f'''(x)f'(x)}{2f''(x)^2 + 2f'(x)f'''(x)} \\ &= \frac{f''(s)^2}{2f''(s)^2} = \frac{1}{2}. \end{aligned}$$

Wir haben somit keine quadratische Konvergenz mehr, sondern bloss lineare mit dem Faktor $\frac{1}{2}$, wie beim Bisektionsalgorithmus.

Aufgabe 3.27 *Die Gleichung*

$$x^4 - 6x^3 + 12x^2 - 10x + 3 = 0$$

hat eine Lösung $x = 1$. Wie konvergiert das Newtonverfahren gegen diese Lösung ?

Durch Division mit dem Linearfaktor $x - 1$ stellt man fest, dass 1 eine dreifache Nullstelle ist:

$$g(x) = x^4 - 6x^3 + 12x^2 - 10x + 3 = (x-1)^3(x-3).$$

Wir wollen die Newtoniterationsfunktion gleich für den allgemeinen Fall analysieren. Sei also s eine m-fache Nullstelle der Funktion $g(x)$, d.h.

$$g(x) = (x-s)^m h(x) \quad \text{wobei} \quad h(s) \neq 0 \quad \text{ist.}$$

Für die Newtoniterationsfunktion erhält man dann

$$\begin{aligned} F(x) &= x - \frac{g(x)}{g'(x)} = x - \frac{(x-s)^m h}{m(x-s)^{m-1}h + (x-s)^m h'} \\ &= x - \frac{(x-s)h}{mh + (x-s)h'}, \end{aligned}$$

wobei wir das Argument bei der Funktion h weglassen. Es ist

$$F'(x) = 1 - \frac{(mh+(x-s)h')[h+(x-s)h']-[mh'+h'+(x-s)h'']](x-s)h}{(mh+(x-s)h')^2}$$

Für $x = s$ vereinfacht sich dieser Ausdruck zu

$$F'(s) = 1 - \frac{1}{m} \tag{3.12}$$

und wir haben somit *lineare Konvergenz mit dem Faktor* $1 - \frac{1}{m}$. Im vorliegenden Fall ($m = 3$) ist der Faktor $\frac{2}{3}$.

Aufgabe 3.28 *Die Iteration*

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

(Newtoniteration mit dem falschen Vorzeichen) konvergiert seltsamerweise gegen Pole von f. Warum ?

Wenn man die Pole einer Funktion f berechnen will, kann man die Nullstellen der reziproken Funktion

$$g(x) = \frac{1}{f(x)}$$

suchen. Wendet man dafür das Newtonverfahren an, ergibt sich die Iterationsfunktion

$$F(x) = x - \frac{g(x)}{g'(x)} = x - \frac{\frac{1}{f(x)}}{\left(\frac{1}{f(x)}\right)'} = x - \frac{\frac{1}{f(x)}}{-\frac{f'(x)}{f(x)^2}} = x + \frac{f(x)}{f'(x)}.$$

Aufgabe 3.29 *Diese Aufgabe zeigt, was unter 'geeigneten Startwerten' gemeint sein kann. Man berechne mit dem Newtonverfahren die einzige Nullstelle der Funktion*

$$f(x) = \sqrt[11]{x^{11} - 1} + 0.5 + 0.05\sin(x/100).$$

Das Newtonverfahren konvergiert hier nur, wenn man sehr gute Startwerte x_0 wählt.

Die Funktion f hat für $x = 1$ einen vertikalen Wendepunkt und die Ableitung f' existiert nicht an dieser Stelle. Es ist

$$f'(x) = \left|x^{11} - 1\right|^{-\frac{10}{11}} x^{10} + \frac{0.05}{100} \cos\left(\frac{x}{100}\right).$$

Man muss beim Programmieren etwas aufpassen und bei der Berechnung von f die Fallunterscheidung $x^{11} - 1 > 0$ oder < 0 machen:

```
(*$B-,U+ Aufgabe 3.29*)
program startwerte;
var xa,x :real;

function f(x :real) :real;
var s,d : real; i,vor : integer;
begin
   s := 1; for i := 1 to 11 do s := s * x;
   d := s - 1;
   if d > 0 then vor:= 1 else vor:= -1;
   d := exp(ln(abs(d))/11);
   f := vor*d + 0.5 + 0.05 * sin(x/100)
end ;

function fs(x :real) :real;
var s,d : real; i,vor : integer;
begin
   s := 1; for i := 1 to 10 do s := s * x;
   d := s * x - 1;
   d := exp(-10/11 * ln(abs(d))) * s;
   fs:= d + 0.0005 * cos(x/100)
end ;

begin
   writeln('Startwert eingeben');
   writeln('Die Newtonfolge konvergiert z.B. fuer x0 = 0.9999 ');
   writeln('Iteration Abbrechen mit CTRL-C');
   read(x);
   repeat
      xa:= x; x := x - f(x)/fs(x); writeln(x);
   until abs(xa - x) < abs(x) * 1E-9;
end.
```

Für ‘genügend nahe’ Startwerte erhält man die Nullstelle $s = 0.99995511002$.

Aufgabe 3.30 *Indem die Funktion $f(x)$ lokal durch das Taylorpolynom 2. Grades approximiert wird, leite man eine Iterationsformel (die* Euler'sche Iterationsformel*) zur Berechnung von Nullstellen her. Man schreibe die resultierende Iterationsform in der Form von Satz 3.5 und beweise damit gleichzeitig, dass die Folge kubisch konvergiert.*

Das Taylorpolynom 2. Grades von f an der Entwicklungsstelle x_k lautet

$$T_2(x) = c + b(x - x_k) + a(x - x_k)^2$$

wobei

$$c = f(x_k), \quad b = \frac{f'(x_k)}{1!} \quad \text{und} \quad a = \frac{f''(x_k)}{2!}$$

ist. Löst man die Gleichung $T_2(x) = 0$ nach x auf und bezeichnet die Lösung mit $x = x_{k+1}$, so wird

$$x_{k+1} - x_k = -\frac{b}{2a}\left(1 \pm \sqrt{1 - \frac{4ac}{b^2}}\right).$$

Von den beiden möglichen Korrekturen $x_{k+1} - x_k$ wählen wir die betragsmässig kleinere, d.h.

$$x_{k+1} - x_k = -\frac{b}{2a}\left(1 - \sqrt{1 - \frac{4ac}{b^2}}\right).$$

Der Ausdruck in der Klammer kann nun durch Auslöschung ungenau werden, wir bilden daher

$$x_{k+1} - x_k = -\frac{b}{2a}\frac{\left(1 - \sqrt{1 - \frac{4ac}{b^2}}\right)\left(1 + \sqrt{1 - \frac{4ac}{b^2}}\right)}{1 + \sqrt{1 - \frac{4ac}{b^2}}}$$

und erhalten nach Ausmultiplizieren und Vereinfachung

$$\begin{aligned} x_{k+1} - x_k &= -\frac{b}{2a}\left(\frac{\frac{4ac}{b^2}}{1 + \sqrt{1 - \frac{4ac}{b^2}}}\right) \\ \Rightarrow x_{k+1} &= x_k - \frac{c}{b}\left(\frac{2}{1 + \sqrt{1 - \frac{4ac}{b^2}}}\right) \\ &= x_k - \frac{f(x_k)}{f'(x_k)}\frac{2}{1 + \sqrt{1 - 2\frac{f(x_k)f''(x_k)}{f'(x_k)^2}}} \end{aligned}$$

die *Euler'sche Iterationsformel*. Es ist ferner

$$G(t) = \frac{2}{1 + \sqrt{1 - 2t}} = 1 + \frac{1}{2}t + \frac{1}{2}t^2 + \frac{5}{8}t^3 + \cdots$$

also $G(0) = 1$ und $G'(0) = 1/2$, woraus wegen Satz 3.5 die kubische Konvergenz folgt.

Kapitel 4
Polynome

Aufgabe 4.1 *Man dividiere das Polynom*

$$P_4(x) = 2x^4 - 24x^3 + 100x^2 - 168x + 93$$

durch den Linearfaktor $(x - z)$*. Wie gross werden die Reste* r *für* $z = 1, 2, 3, 4, 5$ *und wie lauten die Koeffizienten des Polynoms* $P_3(x)$*, welches durch die Gleichung* $(4.7*)$

$$P_4(x) = (x - z)P_3(x) + r$$

bestimmt ist ?

Man benützt hier das einfache Hornerschema $(4.5*)$ und erhält z.B. für $z = 1$

$$\begin{array}{rrrrrrl} & 2 & -24 & 100 & -168 & 93 & \\ & & 2 & -22 & 78 & -90 & \\ \hline x = 1 & 2 & -22 & 78 & -90 & \mid\ 3 & = r \end{array}$$

woraus man ablesen kann, dass

$$P_4(x) = (x - 1)(2x^3 - 22x^2 + 78x - 90) + 3$$

ist. Analog erhält man für

$$\begin{array}{lll} z = 2 & P_3(x) = 2x^3 - 20x^2 + 60x - 48 & r = -3 \\ z = 3 & P_3(x) = 2x^3 - 18x^2 + 46x - 30 & r = \ \ 3 \\ z = 4 & P_3(x) = 2x^3 - 16x^2 + 36x - 24 & r = -3 \\ z = 5 & P_3(x) = 2x^3 - 14x^2 + 30x - 18 & r = \ \ 3 \end{array}$$

Aufgabe 4.2 *Man ermittle mittels des Hornerschemas die Funktionswerte des Polynoms*

$$P_6(x) = 5x^6 - 3x^3 + x^2 + 4x - 12$$

für $x = 2, -2, 5, i, 1 + i$*.*

Die Funktionswerte sind

$$P_6(2) = 296, \quad P_6(-2) = 328, \quad P_6(5) = 77'783, \quad P_6(i) = -18 + 7i$$

und für $z = 1 + i$ erhält man das Hornerschema

$$\begin{array}{rrrrrrr} 5 & 0 & 0 & -3 & 1 & 4 & -12 \\ & 5 + 5i & 10i & -10 + 10i & -23 - 3i & -19 - 25i & 10 - 40i \\ \hline 5 & 5 + 5i & 10i & -13 + 10i & -22 - 3i & -15 - 25i & \mid -2 - 40i \end{array}$$

d.h. es ist $P_6(1 + i) = -2 - 40i$.

Aufgabe 4.3 *Man schreibe ein Programm, das eine Wertetabelle für das Polynom*

$$P_6(x) = 1 - 5x^2 - 0.5x^3 + x^4 + 0.1x^5 - x^6$$

für $x = -2, -1.5, -1, -0.5, \ldots, 2$ *mittels des Hornerschemas berechnet und ausdruckt.*

```
(*$B- Aufgabe 4.3*)
program polynom;
type vektor= array [0..20] of real;
var a : vektor; n,i : integer; z : real;
function p(z : real; a : vektor) : real;
var s : real; i : integer;
begin
   s := 0;
   for i := n downto 0 do s := s * z + a[i];
   p := s
end ;
begin
   writeln('n und a[0],...,a[n] eingeben');
   read(n); for i := 0 to n do read(a[i]);
   for i := -4 to 4 do
   begin
      z := i/2; writeln('z=',z : 5 : 1,'  P(z) =', p(z,a) : 12 : 6)
   end
end.
```

Aufgabe 4.4 *Man schreibe ein Programm, das Zahlen im Zahlensystem der Basis* b_1 *einliest und in Zahlen des Zahlensystems mit Basis* b_2 *umwandelt. Hinweis:* $b_1 \longrightarrow$ *dezimal* $\longrightarrow b_2$.

Die Ziffern einer Zahl im Zahlensystem mit Basis b werden durch die Zeichen $0, 1, \ldots, 9, \mathrm{A}, \mathrm{B}, \mathrm{C} \ldots$ dargestellt. Zum Beispiel sind für $b = 2$ nur die beiden Ziffern $\{0, 1\}$ vorhanden, während im Hexadezimalsystem $b = 16$ die Ziffern durch die Zeichen $\{0, 1, 2, \ldots, 9, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$ dargestellt werden. Es ist etwa

$$\begin{aligned} 2\mathrm{B1C}_{16} &= 2 \cdot 16^3 + \mathrm{B} \cdot 16^2 + 1 \cdot 16^1 + \mathrm{C} \\ &= 2 \cdot 16^3 + 11 \cdot 16^2 + 1 \cdot 16^1 + 12 = 11'036_{10} \end{aligned}$$

Im nachfolgenden Programm werden die Ziffern Variabeln vom Typus *char* sein. Wir brauchen daher eine Funktion $wert(c : char) : integer$, mit welcher der Zahlenwert einer Ziffer berechnet werden kann:

$$wert(c) = \begin{cases} ord(c) - ord('0'), & \text{wenn } '0' \leq c \leq '9' \\ ord(c) - ord('A') + 10, & \text{wenn } c = A, B, C \ldots \end{cases}$$

Um die Zahl im neuen Zahlensystem auszudrucken, benötigen wir ferner auch die Umkehrfunktion von *wert*, welche wir mit *ch* bezeichnen.

Die Funktion *bnachdez*(*d* : *bzahl*) berechnet nach Algorithmus 4.3 die Umwandlung von *d* ins Dezimalsystem. Ferner besorgt die Prozedur *deznachb* nach Algorithmus 4.4 die Umwandlung einer Dezimalzahl ins Zahlensystem mit Basis *b*. Es ergibt sich damit das folgende Programm:

```
(*$B-,U+ Aufgabe 4.4*)
program zahum;
const anzziffern = 50;
type bzahl =  record
                  basis, laenge : integer;
                  ziffer : array [0..anzziffern] of char
              end ;
var zahlb : bzahl; h : char; k,i,b,zahl10 : integer;

function wert(c : char) : integer;
(* Die Ziffern werden durch die Charakters 0,1,...,9,A,B,...
dargestellt. Wert berechnet den dezimalen Wert einer Ziffer *)
begin
   if ('0'<= c) and (c <='9') then wert:= ord(c)- ord('0')
   else wert:= ord(upcase(c))-ord('A') + 10
end ;

function ch(i : integer) : char;
(* Umkehrfunktion von Wert *)
begin
   if (0 <= i) and (i <= 9) then ch:= chr(ord('0') + i)
   else ch:= chr(ord('A') + i - 10)
end ;

function bnachdez(d : bzahl) : integer;
(* Algorithmus 4.3*)
var i,u : integer;
begin
   u := 0;
   for i :=d.laenge downto 0 do
   u := u*d.basis+wert(d.ziffer[i]);
   bnachdez :=u
end ;

procedure deznachb(u,b : integer; var d : bzahl);
(* Algorithmus 4.4*)
var i : integer;
begin
   i := 0;
```

```
    while u <> 0 do
    begin
        d.ziffer[i] := ch(u mod b); u := u div b; i := i + 1
    end ;
    d.laenge := i − 1; d.basis:= b
end ;
begin
    writeln('Basis der umzuwandelnden Zahl ?'); readln(b);
    writeln('damit moegliche Ziffern :');
    for i := 0 to b − 1 do write(ch(i));
    writeln; writeln; writeln('Zahl eingeben');
    i := −1;
    while not eoln do
    begin i := i + 1; read(zahlb.ziffer[i]) end ;
    zahlb.laenge:= i; zahlb.basis:= b;
    (*umordnen*)
    with zahlb do
    for k := 0 to laenge div 2 do
    begin
        h := ziffer[k];
        ziffer[k] := ziffer[laenge−k];
        ziffer[laenge−k] := h
    end ;
    zahl10:=bnachdez(zahlb);
    writeln('im Zehnersystem: ',zahl10); writeln;
    writeln('neue Basis eingeben'); read(b);
    deznachb(zahl10, b, zahlb);
    with zahlb do
    for i := laenge downto 0 do write(ziffer[i]); writeln
end.
```

Da wir die Ziffern einer Zahl von links nach rechts einlesen, sie aber von rechts nach links im **array** *ziffer* speichern, muss man sie nach dem Einlesen umordnen. Die dezimale Darstellung der Zahl wird ebenfalls ausgedruckt. Man beachte, dass die Grösse der Zahlen durch den integer Bereich von TURBO PASCAL ($\leq 32'767$) begrenzt ist. Nach der Eingabe der Basis b werden die möglichen Ziffern $0, 1, 2, \ldots, b-1$ ausgedruckt.

Als Beispiel soll die Hexadezimalzahl $2B1C_{16}$ in das Siebnersystem umgewandelt werden. Mit dem Programm *zahum* erhält man folgenden Ausdruck auf dem Bildschirm:

```
Basis der umzuwandelnden Zahl?
```

```
16
damit moegliche Ziffern :
0123456789ABCDEF
Zahl eingeben
2B1C
im Zehnersystem: 11036
neue Basis eingeben
7
44114
```

Aufgabe 4.5 *Man verallgemeinere den Algorithmus für Zahlenumwandlungen für gebrochene Zahlen.*

Das zu erstellende Programm soll Zähler z und Nenner n eines Bruchs einlesen und eine Basis b. Danach sollen die Ziffern a_i in der Darstellung

$$\frac{z}{n} = a_m b^m + \cdots + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \cdots \qquad (4.1)$$

berechnet werden. Zum Beispiel ist im Hexadezimalsystem

$$\frac{77}{32} = 2 \cdot 16^0 + 6 \cdot 16^{-1} + 8 \cdot 16^{-2} = 2.68_{16}$$

ein abbrechender Hexadezimalbruch.

In der Gleichung (4.1) bilden die Ziffern $a_m, a_{m-1}, \ldots, a_0$ den *ganzzahligen Teil*, welchen man durch z **div** n erhält. Der *gebrochene Teil* ist ein echter Bruch und wird von den übrigen Ziffern gebildet. Allgemein gilt die Zerlegung

$$\frac{z}{n} = z \;\mathbf{div}\; n + \frac{z \;\mathbf{mod}\; n}{n}$$

und die Ziffern der beiden Teile werden unabhängig voneinander berechnet.

Die Umwandlung des ganzzahligen Teils kann mit dem Algorithmus 4.4 erfolgen oder einfacher mit der rekursiven Prozedur *stelle*, welche entsprechend geändert, aus Aufgabe 4.6 übernommen wurde. Um die Ziffern des gebrochenen Teils zu bestimmen, multiplizieren wir die Gleichung

$$\frac{r}{n} = a_{-1} b^{-1} + a_{-2} b^{-2} + a_{-3} b^{-3} + \cdots$$

mit dem Faktor b und erhalten

$$\frac{br}{n} = a_{-1} + a_{-2} b^{-1} + a_{-3} b^{-2} + \cdots \qquad (4.2)$$

Der ganze Teil des Ausdrucks von Gleichung (4.2) ist gerade die Ziffer a_{-1} und die übrigen Ziffern bilden den Rest :

$$a_1 := (b * r) \textbf{ div } n \quad \text{und} \quad r_{neu} := (b * r) \textbf{ mod } n,$$

wobei

$$\frac{r_{neu}}{n} = a_{-2}b^{-1} + a_{-3}b^{-2} + \cdots$$

Um also k Ziffern $a_{-1}, \ldots, a_{-k}$ des echten Bruchs $\frac{r}{n}$ zu berechnen und auszudrucken, führt man folgende Operationen aus:

```
for i := 1 to k do
begin
   r := b * r;
   write(r div n : 1);
   r := r mod n;
end
```

Im folgenden Programm berechnen wir soviele Ziffern des gebrochenen Teils, bis die Periode erkannt ist. Dazu muss man neben den Ziffern auch die Reste speichern. Tritt ein Rest auf, der schon vorhanden ist, so ist das Periodenende erreicht, denn von nun an wiederholt sich dieselbe Ziffernfolge wie beim erstmaligen Auftreten des Restes. Die Reste könnten in einer Menge *reste* (vom Typus **set**) gespeichert werden, dann wäre die Überprüfung, ob der neue Rest r schon in der Menge vorhanden ist einfach der bool'sche Ausdruck: r **in** *reste*. Wir möchten aber noch den Periodenanfang mit einer Leerstelle kennzeichnen, sodass nicht nur die Menge der Reste, sondern auch die Reihenfolge derselben bekannt sein muss. Wir verwenden daher einen *integer* **array** für die Reste. Die Funktion *ch*, die den Wert einer Ziffer in ein Zeichen $0, 1, \ldots, 9$,A,B,... umwandelt, wurde von Aufgabe 4.4 übernommen.

```
(*$B-,U+ Aufgabe 4.5*)
program bruch;
(* Der Bruch zaehler/nenner wird im System mit Basis b dargestellt *)
const nn = 100;
var zaehler,nenner,b,k,i : integer; fertig : boolean;
   ziffer,r : array [1..nn] of integer;

function ch(i : integer) : char;
begin
   if (0 <= i) and (i <= 9) then ch:= chr(ord('0') + i)
   else ch:= chr(ord('A') + i - 10)
end ;

procedure stelle(b, n : integer);
begin
```

```
    if n > 0 then
    begin
        stelle(b, n div b);
        write(ch(n mod b))
    end
end ;
function restvorhanden : boolean;
var ja : boolean; i : integer;
begin
    i := 0; ja := false;
    while (i < k) and not ja do
    begin i := i + 1; ja := zaehler= r[i] end ;
    restvorhanden := ja
end ;
begin
    writeln('Zaehler und Nenner des umzuwandelnden Bruchs eingeben');
    read(zaehler,nenner);
    writeln('Basis des Zahlensystems eingeben '); read(b);
    stelle(b,zaehler div nenner);
    write('.'); zaehler := zaehler mod nenner;
    k := 0;
    repeat
        k := k + 1; r[k] :=zaehler;
        zaehler:=zaehler*b;
        ziffer[k] := zaehler div nenner;
        zaehler:= zaehler mod nenner;
        fertig:= restvorhanden;
    until fertig or (k =nn);
    if not fertig then writeln('Periode zu gross! nn erhoehen');
    for i := 1 to k do
    begin
        if r[i] =zaehler then write(' ');
        write(ch(ziffer[i]))
    end ;
end.
```

Als Beispiel wandeln wir mit diesem Programm den Bruch $\frac{77}{32}$ in das Hexadezimalsystem um:

```
Zaehler und Nenner des umzuwandelnden Bruchs eingeben
77    32
Basis des Zahlensystems eingeben
16
```

2.68 0

Die Periode ist 0, d.h. die Entwicklung bricht ab. Eine Entwicklung mit langer Periode erhält man mit dem Nenner 47. Hier die Resultate für den Bruch $\frac{120}{47}$ im Zehner– und im Hexadezimalsystem:

Zaehler und Nenner des umzuwandelnden Bruchs eingeben
120 47
Basis des Zahlensystems eingeben
10
2. 553191489361702127659574468085106382978723404 2

Zaehler und Nenner des umzuwandelnden Bruchs eingeben
120 47
Basis des Zahlensystems eingeben
16
2. 8D9DF51B3BEA3677D46CEFA

Aufgabe 4.6 *Man studiere das folgende rekursive Programm.*

```
program was(input,output);
var z : integer;
procedure stelle(n : integer);
begin
   if n > 0 then
   begin
      stelle(n div 2);
      write(n mod 2)
   end
end ;

begin
   writeln('Zahl eingeben'); read(z);
   stelle(z)
end.
```

Dieses Programm berechnet die Darstellung einer ganzen Zahl im Zweiersystem. Die Dualstellen werden durch n **mod** 2 von rechts nach links berechnet aber wegen der Rekursivität in umgekehrter Reihenfolge, wie es sein soll, ausgedruckt. Durch die Rekursivität erspart man sich damit das explizite Abspeichern der Ziffern in einem **array** .

Aufgabe 4.7 *Man berechne für $x = -1$ die Taylorentwicklung von*

$$P_5(x) = x^5 + x^4 + 2x^3 + 2x^2 + x + 1. \qquad (4.21*)$$

Wir ergänzen Algorithmus 4.5 zu einem vollständigen Programm und drukken das ganze Hornerschema aus:

```
(*$B- Algorithmus 4.5 *)
program vollsthorner;
var a : array [0..20] of real; z : real; i,j,n : integer;
    tf : text; stri : string [10];
begin
    writeln('wohin mit dem Output?'); readln(stri);
    assign(tf,stri); rewrite(tf);
    writeln('z,n und a[0],...,a[n] eingeben');
    read(z,n); for i := 0 to n do read(a[i]);
    writeln(tf,'Vollstaendiges Hornerschema fuer z =',z : 7 : 2);
    for i := n downto 0 do write(tf, a[i] : 9 : 2); writeln(tf);
    for j := 0 to n do
    begin write(tf, a[n] : 9 : 2);
       for i := n - 1 downto j do
       begin
          a[i] := a[i + 1] * z + a[i];
          write(tf, a[i] : 9 : 2)
       end ;
       writeln(tf)
    end ;
    writeln(tf); writeln(tf,'neue Koeffizienten : b[',n,'],...,b[0] ');
    for i := n downto 0 do write(tf, a[i] : 9 : 2); writeln(tf);
    close(tf)
end.
```

Für das Polynom (4.21∗) erhalten wir für $z = -1$ den Output

Vollstaendiges Hornerschema fuer $z = -1.00$

1.00	1.00	2.00	2.00	1.00	1.00
1.00	0.00	2.00	0.00	1.00	0.00
1.00	−1.00	3.00	−3.00	4.00	
1.00	−2.00	5.00	−8.00		
1.00	−3.00	8.00			
1.00	−4.00				
1.00					

neue Koeffizienten : $b[5], \ldots, b[0]$

1.00	−4.00	8.00	−8.00	4.00	0.00

d.h. es gilt

$$P_5(x) = (x+1)^5 - 4(x+1)^4 + 8(x+1)^3 - 8(x+1)^2 + 4(x+1) + 0$$

und somit ist $x = -1$ eine Nullstelle von P_5.

Aufgabe 4.8 *Man berechne $P_5'''(2)$ von Aufgabe 4.7 durch gewöhnliches Ableiten und mittels vollständigem Hornerschema.*

Es ist $P_5'''(x) = 60x^2 + 24x + 12$ und somit $P_5(2) = 300$. Wenn wir aber P_5 an der Stelle $z = 2$ umentwickeln, erhalten wir

$P_5(x) = (x-2)^5 + 11(x-2)^4 + 50(x-2)^3 + 118(x-2)^2 + 145(x-2) + 75$

und es ist

$$\frac{P_5'''(2)}{3!} = 50 \quad \Rightarrow P_5(2) = 300.$$

Aufgabe 4.9 *Man entwickle das Polynom*

$$P_5(x) = x^5 + 4x^4 + 3x^3 - 2x + 1$$

nach Potenzen von $(x-2)$ und nach Potenzen von $(x+3)$.

Mit dem Programm von Aufgabe 4.7 erhält man

$$P_5(x) = h^5 + 14h^4 + 75h^3 + 194h^2 + 242h + 117 \quad \text{mit} \quad h = x - 2$$

$$P_5(x) = (x+3)^5 - 11(x+3)^4 + 45(x+3)^3 - 81(x+3)^2 + 52(x+3) + 7$$

Aufgabe 4.10 *Man berechne für die Entwicklungsstelle $x = -2$ das Taylorpolynom 6. Grades von*

$$P_6(x) = x^6 + 12x^5 + 60x^4 + 160x^3 + 240x^2 + 192x^+64.$$

Hier erhält man mit dem Programm:

Vollstaendiges Hornerschema fuer $z = -2.00$

1.00	12.00	60.00	160.00	240.00	192.00	64.00
1.00	10.00	40.00	80.00	80.00	32.00	0.00
1.00	8.00	24.00	32.00	16.00	0.00	
1.00	6.00	12.00	8.00	0.00		
1.00	4.00	4.00	0.00			
1.00	2.00	0.00				
1.00	0.00					
1.00						

neue Koeffizienten :$b[6], \ldots, b[0]$

1.00	0.00	0.00	0.00	0.00	0.00	0.00

d.h. das Taylorpolynom ist gerade $(x+2)^6$.

Aufgabe 4.11 *Man schätze die Lage der Nullstellen der folgenden Polynome ab:*

a) $x^4 - 8x^3 - 2x^2 + 24x - 15$

b) $x^2 - 4x + 4 - i$

Die Anwendung von Satz 4.2 ergibt für

a)

$$\sqrt[1]{\left|\frac{a_3}{a_4}\right|} = 8 \quad \sqrt[2]{\left|\frac{a_2}{a_4}\right|} = \sqrt{2} \quad \sqrt[3]{\left|\frac{a_1}{a_4}\right|} = 2.88 \quad \sqrt[4]{\left|\frac{a_0}{a_4}\right|} = 1.968\,,$$

somit liegen alle Nullstellen im Kreis mit Radius $r = 16$.

b)

$$\sqrt[1]{\left|\frac{a_1}{a_2}\right|} = 4 \quad \sqrt[2]{\left|\frac{a_0}{a_2}\right|} = 2.03$$

also ist hier $r = 8$.

Aufgabe 4.12 *Man bestimme durch Erraten und mittels Abspalten mit dem Hornerschema alle Nullstellen des Polynoms*

$$P_4(x) = 10 - 35x + 45x^2 - 25x^3 + 5x^4.$$

Wie lautet die Faktorzerlegung von P?

$x = 1$ ist eine Nullstelle von P_4. Wenn wir sie abspalten, erhalten wir ein Polynom 3. Grades, welches ebefalls die Nullstelle $x = 1$ hat, wie man aus dem folgendem Hornerschema ersehen kann

	5	−25	45	−35	10	
		5	−20	25	−10	
$x = 1$	5	−20	25	−10	| 0	$= P_4(1)$
		5	−15	10		
$x = 1$	5	−15	10	| 0	$= P_3(1)$	
		5	−10			
$x = 1$	5	−10	| 0	$= P_2(1)$		

Das Polynom $P_2(x)$, das nach dem Abspalten von $x = 1$ von P_3 auftritt, hat auch die Nullstelle $x = 1$, so dass man ein drittes Hornerschema anfügen konnte.

Aus dem obigen Schema lesen wir die folgenden Zerlegungen ab:

$$P_4(x) = P_3(x)(x-1) = (5x^3 - 20x^2 + 25x - 10)(x-1)$$
$$P_3(x) = P_2(x)(x-1) = (5x^2 - 15x + 10)(x-1)$$
$$P_2(x) = P_1(x)(x-1) = (5x - 10)(x-1)$$

Also lautet die gesuchte Faktorzerlegung:

$$P_4(x) = (5x - 10)(x - 1)^3 = 5(x - 2)(x - 1)^3.$$

Aufgabe 4.13 *Das Polynom*

$$P_5(x) = x^5 + x^4 + 2x^3 + 2x^2 + x + 1$$

hat die Nullstelle $x = i$. Man bestimme die weiteren Nullstellen und stelle die Faktorzerlegung auf. Hinweis: ein reelles Polynom hat immer konjugiert komplexe Nullstellen !

Wenn wir Die Nullstellen $x = i$ und $x = -i$ abspalten, erhalten wir ein Polynom 3. Grades, bei dem man die Nullstelle $x = -1$ erraten kann, wie aus folgendem Schema ersichtlich ist:

	1	1	2	2	1	1	
		i	$-1+i$	$-1+i$	$-1+i$	-1	
$x = i$	1	$1+i$	$1+i$	$1+i$	i	$\mid 0$	$= P_5(i)$
		$-i$	$-i$	$-i$	$-i$		
$x = -i$	1	1	1	1	$\mid 0$		$= P_4(-i)$
		-1	0	-1			
$x = -1$	1	0	1	$\mid 0$			$= P_3(-1)$
		i	-1				
$x = i$	1	i	$\mid 0$				$= P_2(i)$

Das verbleibende Polynom 2. Grades ist $x^2 + 1$ und hat die Nullstelle $x = i$, die im obigen Schema auch abgespalten wurde. Die Faktorzerlegung kann damit abgelesen werden:

$$P_5(x) = (x + i)^2(x - i)^2(x + 1).$$

Aufgabe 4.14 *$x = 1$ ist Nullstelle des Polynoms $P_{10}(x) = x^{10} - 1$. Man spalte diese Nullstelle mittels des Hornerschemas ab. Wie lautet das Restpolynom ? Man bestimme alle Nullstellen des Polynoms P_{10}.*

	1	0	0	0	0	0	0	0	0	0	-1
		1	1	1	1	1	1	1	1	1	1
$x = 1$	1	1	1	1	1	1	1	1	1	1	$\mid 0$

$= P_{10}(1)$

Das Restpolynom ist

$$P_9(x) = x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1.$$

Die Nullstellen von P_{10} sind die 10-ten Einheitswurzeln

$$x_k = e^{i\frac{2\pi}{10}k}, \quad k = 0, 1, \ldots, 9$$

Aufgabe 4.15 *Das Polynom*

$$P_5(x) = x^5 - x^4 - 8x^3 + 20x^2 - 17x + 5$$

hat eine Nullstelle mehrfach. Man bestimme ihre Vielfachheit.

$x = 1$ ist Nullstelle von P_5. Wir spalten diese ab und erhalten das Schema:

$$\begin{array}{rrrrrrrl}
 & 1 & -1 & -8 & 20 & -17 & 5 & \\
 & & 1 & 0 & -8 & 12 & -5 & \\
\hline
x=1 & 1 & 0 & -8 & 12 & -5 & \mid 0 & = P_5(1) \\
 & & 1 & 1 & -7 & 5 & & \\
\hline
x=1 & 1 & 1 & -7 & 5 & \mid 0 & & = P_4(1) \\
 & & 1 & 2 & 5 & & & \\
\hline
x=1 & 1 & 2 & -5 & \mid 0 & & & = P_3(1) \\
 & & 1 & 3 & & & & \\
\hline
x=1 & 1 & 3 & \mid -2 & & & & = P_2(1)
\end{array}$$

Die Nullstelle $x = 1$ ist *dreifach* und man liest die folgende Zerlegung ab:

$$P_5(x) = (x^2 + 2x - 5)(x - 1)^3.$$

Aufgabe 4.16 *Die Gleichung*

$$x^4 + ax^3 + bx^2 - 26x + 120 = 0 \tag{4.3}$$

hat die Lösungen $x_1 = 2$ und $x_2 = -3$. Man bestimme die Koeffizienten a und b, sowie die restlichen Lösungen.

Setzt man in der Gleichung (4.3) $x = 2$ bzw. $x = 3$ ein, so erhält man zwei lineare Gleichungen für die Unbekannten a und b:

$$\begin{aligned} 8a + 4b &= -84 \\ -27a + 9b &= -279 \end{aligned}.$$

Durch Elimination erhält man als Lösungen $a = 2$ und $b = -25$. Um die restlichen Lösungen der Gleichung (4.3) zu bestimmen, spalten wir die beiden Nullstellen ab:

$$\begin{array}{rrrrrrl}
 & 1 & 2 & -25 & -26 & 120 & \\
 & & 2 & 8 & -34 & -120 & \\
\hline
x = 2 & 1 & 4 & -17 & -60 & \mid 0 & = P_4(2) \\
 & & -3 & -3 & 60 & & \\
\hline
x = -3 & 1 & 1 & -20 & \mid 0 & & = P_3(-3)
\end{array}$$

Die gesuchten Lösungen sind Lösungen der quadratischen Gleichung

$$x^2 + x - 20 = 0 \quad \Rightarrow x_3 = -5 \quad \text{und} \quad x_4 = 4.$$

Aufgabe 4.17 *Man ändere die beiden Newtonprogramme Algorithmus 4.7 und 4.9 so ab, dass komplex gerechnet wird, und dass sie auch für Polynome mit komplexen Koeffizienten funktionieren. Man breche die Iteration ab, wenn*

$$|P_n(s)| < 10 * \varepsilon * \sum_{i=0}^{n} |a_i||s|^i, \tag{4.4}$$

wobei ε die Maschinengenauigkeit bedeutet.

In Aufgabe 1.4 wurde für die Maschinengenauigkeit als $\tilde{\varepsilon} = 1.818\text{E}{-12}$ berechnet. Wir benützen in den folgenden Programmen dafür die Konstante *eps*= 2E−12. Für das Abbruchkriterium benötigen wir die Beträge der Koeffizienten a_i, welche wir am Anfang der Prozedur *newtoniteration* berechnen und im Vektor *c* speichern.

Als allererster Startwert für die Iteration benützen wir $x_0 = 1+i$, danach wird jeweils der konjugiert komplexe Wert der eben berechneten Nullstelle als neuer Startwert benützt. Bei reellen Polynomen ist es wichtig, die Iteration mit einer komplexen Zahl zu beginnen, weil mit einer reellen Zahl die Newtonfolge auch reel bleibt und man keine komplexen Nullstellen erreichen kann. Da bei einem reellen Polynom komplexe Nullstellen immer konjugiert komplex auftreten, ist die oben angegebene Wahl des neuen Startwertes geeignet.

Die jeweils zuletzt berechnete Nullstelle kann bei der Deflation an Stelle des Höchstkoeffizienten gespeichert werden. Dadurch überschreibt man den Koeffizientenvektor a_i, $i = 1, 2 \ldots, n$ mit den Nullstellen.

Es ergibt sich damit folgende Übersetzung des Algorithmus 4.7 in komplexe Arithmetik:

```
(*$B-,U+ Aufgabe 4.17A*)
program cnewton;
const nn = 20; eps= 2e - 12;
type koeff = array [0..nn] of real;
var ar,ai,br,bi,c : koeff; n,grad,i : integer;
    yb,yr,yi,xr,xi,pr,pi,psr,psi,h : real;
    tf : text; stri : string [20];
procedure newtoniteration;
var fertig : boolean;
begin
    for i := 0 to n do c[i] := sqrt(sqr(ar[i]) + sqr(ai[i]));
    xr := yr; xi := -yi;
    repeat
        pr := 0; pi := 0; psr := 0; psi := 0;
        for i := n downto 0 do
```

```
        begin
            h := pr + psr * xr - psi * xi;
            psi := pi + psi * xr + psr * xi; psr := h;
            h := ar[i] + pr * xr - pi * xi;
            pi := ai[i] + pi * xr + pr * xi; pr := h;
        end ;
        writeln(tf,'x =',xr : 14 : 7, xi : 14 : 7,' P(x) = ',pr,pi);
        yb := sqrt(sqr(xr) + sqr(xi)); h := 0;
        for i := n downto 0 do h := h * yb + c[i];
        fertig:= sqrt(sqr(pr) + sqr(pi)) < 10 * eps * h;
        h := sqr(psr) + sqr(psi);
        yr := xr - (pr * psr + pi * psi)/h;
        yi := xi - (pi * psr - pr * psi)/h;
        xr := yr; xi := yi;
    until fertig;
end ;

procedure deflation;
(* Algorithmus 4.8*)
begin
    br[n - 1] := ar[n]; bi[n - 1] := ai[n];
    for i := n - 1 downto 1 do
    begin
        br[i - 1] := ar[i] + br[i] * yr - bi[i] * yi;
        bi[i - 1] := ai[i] + br[i] * yi + bi[i] * yr;
    end ;
    pr := ar[0] + br[0] * yr - bi[0] * yi;
    pi := ai[0] + br[0] * yi + bi[0] * yr;
    writeln(tf,'Nst Nr. ',grad - n + 1,'  ',yr,yi);
    br[n] := yr; bi[n] := yi; ar := br; ai := bi;
end ;

begin
    writeln('Wohin mit dem output ?');
    readln(stri); assign(tf, stri); rewrite(tf);
    writeln('Grad, a[0], . . . , a[n] ?');
    read(grad); for i := 0 to grad do read(ar[i], ai[i]);
    (* Startwert*)
    yr := 1; yi := -1;
    for n := grad downto 1 do
    begin
        newtoniteration; deflation
    end ;
    writeln(tf,'Nullstellen');
```

for $n := 1$ **to** *grad* **do** *writeln*(*tf*, *ar*[*n*],'+*I*∗ ',*ai*[*n*]);
writeln(*tf*); *close*(*tf*);
end.

Die folgenden drei Beispiele zeigen einige Eigenschaften dieses Algorithmus: Zuerst berechnen wir die Nullstellen von

$$P_3(x) = 3x^3 + x^2 - 5x + 1.$$

Nach 14 Iterationen wird die erste Nullstelle $x_1 = 1$ gefunden und abgespalten. Für die nächste Nullstelle benötigt man nur noch 6 Iterationen und man erhält das Resultat:

```
Nullstellen
−1.5485837704E+00   +I∗   −7.3604380899E−25
 2.1525043702E−01   +I∗    2.3904050341E−25
 1.0000000000E+00   +I∗   −1.6543612251E−24
```

Die Nullstellen sind bis auf Maschinengenauigkeit richtig gerechnet: es ist $x_{2,3} = (-2 \pm \sqrt{7})/3$.

Der *boolean* Variabeln *fertig* wird der Ausdruck (4.4) zugewiesen. Wenn man zu einem Iterationswert x_k die Funktionswerte $P_n(x_k)$ und $P'_n(x_k)$ gerechnet hat, ist es schade, diese Information nicht auszunützen und auf den nächsten Iterationsschritt, der nicht viel Rechenzeit kostet, zu verzichten. Wenn das Abbruchkriterium erfüllt ist (*fertig* = *true*), führen wir daher noch den nächsten Iterationsschritt aus. Bei einfachen Nullstellen ist an diesem Vorgehen sicher nichts auszusetzen, bei mehrfachen Nullstellen kann jedoch dieser Schritt ein gutes Resultat zerstören (vgl. Aufgabe 4.20). Bei der Berechnung der Korrektur $-P_n(x_k)/P'_n(x_k)$ werden bei mehrfachen Nullstellen Zähler und Nenner durch Auslöschung null und der Quotient kann einen beliebigen Wert annehmen, sodass man sich wieder von der gefundenen Nullstelle entfernt.

Als zweites Beispiel betrachten wir das Polynom von Aufgabe 4.11 b:

$$P_2(x) = x^2 - 4x + 4 - i.$$

Hier benötigt man für die Berechnung der ersten Nullstelle mit dem Startwert $x = 1 + i$ 48 Iterationen ! Man sieht bei diesem Beispiel deutlich, dass ein grosser Unterschied zwischen globaler und lokaler (quadratischer) Konvergenz besteht. Das Programm liefert die beiden Nullstellen $2.7071067812 + 0.70710678118i$ und $1.2928932188 - 0.70710678118i$, welche nicht konjugiert komplex sind, weil das Polynom nicht reell ist.

Das dritte Beispiel ist das Polynom von Aufgabe 4.10, welches die sechsfache Nullstelle $x = -2$ hat:

$$P_6(x) = x^6 + 12x^5 + 60x^4 + 160x^3 + 240x^2 + 192x + 64.$$

Nach 23 Iterationen erhält man die erste Nullstelle und danach in jeweils etwa 5-7 Iterationen die restlichen Nullstellen. Der Output lautet:

Nullstellen	
−2.0452325659E+00	$+I*$ −1.4860170931E−02
−2.0355019572E+00	$+I*$ 3.1790218374E−02
−2.0097663673E+00	$+I*$ −4.6556926495E−02
−1.9901733208E+00	$+I*$ 4.6629121825E−02
−1.9645907692E+00	$+I*$ −3.1773943727E−02
−1.9547350198E+00	$+I*$ 1.4771700955E−02

Hier ist man sicherlich von der Genauigkeit überrascht (oder vielmehr von der Ungenauigkeit!). Die Ursache ist hier nicht die oben beschriebene Auslöschung, welche bei der Berechnung der Korrektur auftritt, sondern liegt in der Kondition des Problems. Statt der sechsfachen reellen Nullstelle bei -2 erhalten wir sechs einfache komplexe Nullstellen, die sich auf einem kleinen Kreis um -2 befinden. Wie schon im Lehrbuch erwähnt, ist eine mehrfache Nullstelle immer schlecht konditioniert und die numerisch berechneten Nullstellen haben die Tendenz zu 'explodieren'. Man kann sich das leicht verständlich machen, wenn man das Prinzip von Wilkinson beachtet, nach dem das durch Rundungsfehler verfälschte Resultat einer numerischen Rechnung gleich dem exakten Resultat eines benachbarten Problems ist.

Die Aufgabe war, die Nullstellen des Polynoms $P_6(x) = (x+2)^6$ zu berechnen. Ein dazu benachbartes Problem ist die Aufgabe, die Nullstellen von

$$Q_6(x) = (x+2)^6 - \varepsilon \quad \text{mit} \quad \varepsilon = \text{Maschinengenauigkeit}$$

zu berechnen. Diese sind aber

$$x = -2 + \sqrt[6]{\varepsilon},$$

wobei mit der Wurzel die 6 komplexen Wurzeln gemeint sind. Für

$$\varepsilon = 2\text{E}{-12} \quad \Rightarrow \sqrt[6]{\varepsilon} = 0.011225,$$

was die Störung der Grössenordnung 10^{-2} und die Lage der 6 Nullstellen auf einem kleinen Kreis vollauf erklärt. Dieses Beispiel zeigt ferner, dass man das Abbruchkriterium sorgfältig wählen muss: hier würde es nicht genügen, die relative Differenz zweier Näherungen zu prüfen. Diese stimmen nämlich kaum auf zwei Dezimalstellen überein, wie im Ausdruck der Zwischenwerte ersichtlich ist, so dass man mit einem solchen Abbruchkriterium eine unendliche Programmschleife erhalten würde.

Nun soll noch der zweite Algorithmus (Newton mit Umentwickeln) für komplexe Arithmetik umgeschrieben werden. Das vorgeschlagene Abbruchkriterium (4.4) kann nicht angewendet werden, weil wir die Koeffizienten des Polynoms laufend durch die neuen Koeffizienten der Taylorentwicklung überschreiben. Der konstante Koeffizient a_0 ist der Funktionswert im jeweiligen Näherungswert x_k und wir brauchen ein Mass, um festzustellen, ob $|a_0|$ 'klein' ist. Am einfachsten berechnet man dazu einmal am Anfang die Beträge der Koeffizienten

$$ab := \sum_{i=0}^{n} |a_i|$$

und prüft danach ob

$$|P_n(x_k)| = |a_0| < \varepsilon \times ab$$

ist. Falls dies der Fall ist, wird x_k als Nullstelle akzeptiert. Man kann die Grösse ab als

$$ab := \sum_{i=0}^{n} |a_i||x|^i \quad \text{für} \quad x = 1$$

deuten. Mit diesem Kriterium erhält man das folgende Programm als Übersetzung von Algorithmus 4.9

```
(*$B- Aufgabe 4.17B*)
program cnewum;
(* Komplexe Newtoniteration mit Umentwickeln *)
const nn = 20; eps= 2e - 12;
type koeff = array [0..nn] of real;
var ar,ai : koeff; n,grad,i,j : integer;
    xr,xi,hr,hi,y,ab : real; tf : text; stri : string [20];
procedure newtoniteration;
begin
  ab := 0;
  for i := 0 to n do ab := ab + sqrt(sqr(ar[i]) + sqr(ai[i]));
  while sqrt(sqr(ar[0]) + sqr(ai[0])) > eps * ab do
  begin
    for j := 0 to n - 1 do
    for i := n - 1 downto j do
    begin
      ar[i] := hr * ar[i + 1] - hi * ai[i + 1] + ar[i];
      ai[i] := hi * ar[i + 1] + hr * ai[i + 1] + ai[i];
    end ;
    xr := xr + hr; xi := xi + hi;
    writeln(tf,'x =',xr : 12 : 7, xi : 12 : 7,'  P(x) =',ar[0], ai[0]);
    y := sqr(ar[1]) + sqr(ai[1]);
```

```
        hr := (−ar[0] ∗ ar[1] − ai[0] ∗ ai[1])/y;
        hi := (ar[0] ∗ ai[1] − ai[0] ∗ ar[1])/y;
    end ;
end ;
procedure deflation;
begin
    for i := 1 to n do
    begin ar[i − 1] := ar[i]; ai[i − 1] := ai[i] end ;
    writeln(tf,'Nst Nr. ',n,' ',xr, xi);
    ar[n] := xr; ai[n] := xi; hr := 0; hi := −2 ∗ xi;
end ;
begin
    writeln('Wohin mit dem Output ?');
    readln(stri); assign(tf, stri); rewrite(tf);
    writeln('Grad a[0], . . . , a[n] ');
    read(grad); for i := 0 to grad do read(ar[i], ai[i]);
    xr := 0; xi := 0; hr := 1; hi := 1;
    for n := grad downto 1 do
    begin
        newtoniteration; deflation;
    end ;
    writeln(tf,'Nullstellen');
    for i := 1 to grad do writeln(tf, ar[i],'+I∗',ai[i]);
    close(tf)
end.
```

Als Startwert wurde hier die Korrektur $h = 1 + i$ gewählt. Damit erhält man theoretisch dieselbe Newtonfolge wie beim Algorithmus 4.17A. Die Rundungsfehler bewirken aber, dass sich die Folgen mit der Zeit unterscheiden.

Aufgabe 4.18 *Man schreibe ein Programm für das Nickelverfahren und verbessere das Abbruchkriterium wie in Aufgabe 4.17.*

Im wesentlichen müssen wir zur Lösung dieser Aufgabe die Prozedur *newtoniteration* von Aufgabe 4.17B durch die Nickeliteration ersetzen, welche im Lehrbuch ausführlich beschrieben ist. Das Abbruchkriterium ist dasselbe und die Prozedur *deflation* kann übernommen werden. Die Prozedur *topolar* (Algorithmus 2.3) wird als Include-File dazugeladen. Man erhält damit

```
(∗$B−, U+ Aufgabe 4.18∗)
program nickel;
const nn = 20; eps = 2e − 12;
type koeff = array [0..nn] of real;
```

```
var ar,ai : koeff; grad,n,i,j : integer;
    xr,xi,h1,h2,rmin,r0,phi0,rj,phij,r,phi,ab : real;
    tf : text; stri : string [20];
(*$I topolar *)
procedure eingabe;
begin
    writeln('Wohin mit dem Output ?');
    readln(stri); assign(tf,stri); rewrite(tf);
    writeln('Grad a[0],...,a[Grad] ');
    read(grad); ab := 0;
    for i := 0 to grad do
    begin
        read(ar[i],ai[i]);
        ab := ab + sqrt(sqr(ar[i]) + sqr(ai[i]));
    end ;
end ;
procedure deflation;
begin
    writeln(tf);
    writeln(tf,'Nst Nr.  ',grad + 1 - n,'  ',xr,xi);
    for i := 1 to n do
    begin ar[i - 1] := ar[i]; ai[i - 1] := ai[i] end ;
    ar[n] := xr; ai[n] := xi;
end ;
procedure nickeliteration;
begin
topolar(-ar[0],-ai[0],r0,phi0);
while r0 > eps * ab do
begin
    rmin := 1e10;
    for j := 1 to n do
    begin
        topolar(ar[j],ai[j],rj,phij);
        if rj <> 0 then
        begin
            r := exp(ln(r0/rj)/j);
            if r <= rmin then
            begin
                rmin := r; phi := (phi0 - phij)/j ;
            end
        end
    end ;
```

```
        h1 := rmin * cos(phi); h2 := rmin * sin(phi);
        for j := 0 to n do
        for i := n - 1 downto j do
        begin
            ar[i] := h1 * ar[i + 1] - h2 * ai[i + 1] + ar[i];
            ai[i] := h2 * ar[i + 1] + h1 * ai[i + 1] + ai[i];
        end ;
        xr := xr + h1; xi := xi + h2;
        writeln(tf,'x = ',xr : 12 : 7, xi : 12 : 7,'  P(x) = ',ar[0], ai[0]);
        topolar(-ar[0], -ai[0], r0, phi0);
    end
end ;
begin
    eingabe; xr := 0; xi := 0;
    for n := grad downto 1 do
    begin
        nickeliteration; deflation;
    end ;
    writeln(tf); writeln(tf,'Nullstellen');
    for i := 1 to grad do writeln(tf, ar[i],' + I*',ai[i]);
    close(tf);
end.
```

Wie man beim Benützen dieses Algorithmus bemerkt, ist die globale Konvergenz wesentlich besser als bei den beiden vorangehenden Newtonprogrammen. Beispielsweise benötigt man nur 6 Iterationen, um die Gleichung $x^2 - 4x + 4 - i = 0$ zu lösen, während mit *cnewton* dafür 48 Iterationen durchgeführt werden mussten.

Aufgabe 4.19 *Man beweise, dass die durch die Laguerre-Iteration erhaltene Folge kubisch konvergiert. Hinweis: Man verwende den Satz 3.5.*

Die Laguerre-Iterationsfunktion in Gleichung (4.44∗) hat die Form

$$F(x) = x - \frac{P_n(x)}{P_n'(x)} G(t(x)), \quad \text{mit} \quad t(x) = \frac{P_n(x) P_n''(x)}{{P_n'}^2(x)}$$

und

$$G(t) = \frac{n}{1 + \sqrt{(n-1)^2 + n(n-1)t}}. \tag{4.5}$$

Es ist $G(0) = 1$ für $n \geq 1$ und

$$G'(t) = -n \frac{\frac{n(n-1)}{2\sqrt{(n-1)^2+n(n-1)t}}}{\left(1 + \sqrt{(n-1)^2 + n(n-1)t}\right)^2} \quad \Rightarrow G'(0) = -\frac{1}{2}.$$

Die Voraussetzungen von Satz 3.5 sind also erfüllt und die Konvergenz ist kubisch.

Aufgabe 4.20 *Man schreibe ein Programm für die Laguerre-Iteration. Man verwende dabei*

a) Die Prozedur csqrt (vgl. Kap 2), um die komplexe Wurzel zu berechnen.

b) Die benötigte zweite Ableitung von P ergibt sich aus den beiden ersten Zeilen des Hornerschemas. Wir geben hier das Programmstück für reelle Arithmetik an:

```
pss:= 0; ps:= 0; p := 0;
for i := n downto 0 do
begin
    pss:= pss*x + 2*ps;
    ps:=ps*x + p;
    p := p * x + a[i];
end ;
(* Es ist p = P(x), ps= P'(x) und pss= P''(x) *)
```

Der Programmaufbau ist hier gleich wie bei *cnewton* (Aufgabe 4.17A). Das Abbruchkriterium und die Deflation können übernommen werden. Für die Berechnung der Laguerrekorrektur lohnt es sich, Prozeduren für die komplexe Multiplikation und Division zu schreiben. Zur Berechnung der Wurzel aus einer komplexen Zahl, kann man Algorithmus 2.10 *csqrt* verwenden. Im folgenden Programm verwenden wir aber dafür die etwas kürzere Prozedur von Aufgabe 2.21.

```
(*$B- Aufgabe 4.20*)
program laguerre;
const nn= 20; eps= 2e - 12;
type koeff = array [0..nn] of real;
var ar,ai,br,bi,c : koeff; n,i,grad : integer;
    yr,yi,xr,xi,f1,f2,fs1,fs2,fss1,fss2,ffs1,ffs2,fssf1,fssf2,h,yb : real;
    tf : text; stri : string [20];
procedure csqrt(a, b : real; var x, y : real);
(* von Aufgabe 2.21*)
begin
    if a > 0 then y := b/sqrt(2 * (a + sqrt(sqr(a) + sqr(b))))
    else y := sqrt((-a + sqrt(sqr(a) + sqr(b)))/2);
    if y = 0 then
    if a < 0 then begin x := 0; y := sqrt(-a) end
    else x := sqrt(a)
```

```
    else
    begin
        if (b < 0) and (y > 0) then y := -y;
        x := b/2/y;
    end ;
end ;
procedure cmul(a,b,c,d : real; var x,y : real);
begin x := (a*c - b*d); y := (b*c + a*d) end ;
procedure cdiv(a,b,c,d : real; var x,y : real);
begin
    x := sqr(c) + sqr(d);
    y := (b*c - a*d)/x; x := (a*c + b*d)/x;
end ;
procedure deflation(yr,yi : real);
(* Algorithmus 4.8*)
begin
    br[n-1] := ar[n]; bi[n-1] := ai[n];
    for i := n-1 downto 1 do
    begin
        br[i-1] := ar[i] + br[i]*yr - bi[i]*yi;
        bi[i-1] := ai[i] + br[i]*yi + bi[i]*yr;
    end ;
    f1 := ar[0] + br[0]*yr - bi[0]*yi;
    f2 := ai[0] + br[0]*yi + bi[0]*yr;
    writeln(tf ,'Nst Nr.  ',grad+1-n,'   ',yr,yi,f1,f2);
    br[n] := yr; bi[n] := yi; ar := br; ai := bi
end ;
procedure laguerreiteration;
var fertig : boolean;
begin
    for i := 0 to n do c[i] := sqrt(sqr(ar[i]) + sqr(ai[i]));
    repeat
        fss1 := 0; fss2 := 0; fs1 := 0; fs2 := 0; f1 := 0; f2 := 0;
        for i := n downto 0 do
        begin
            h := 2*fs1 + fss1*xr - fss2*xi;
            fss2 := 2*fs2 + fss2*xr + fss1*xi; fss1 := h;
            h := f1 + fs1*xr - fs2*xi;
            fs2 := f2 + fs2*xr + fs1*xi; fs1 := h;
            h := ar[i] + f1*xr - f2*xi;
            f2 := ai[i] + f2*xr + f1*xi; f1 := h;
        end ;
```

```
        writeln(tf ,'x =',xr : 12 : 7, xi : 12 : 7,'  P(x) =', f1, f2);
        yb := sqrt(sqr(xr) + sqr(xi)); h := 0;
        for i := n downto 0 do h := h * yb + c[i];
        fertig:= sqrt(sqr(f1) + sqr(f2)) < 10 * eps * h;
        if not fertig then
        begin
            cdiv(f1, f2, fs1, fs2, ffs1, ffs2);
            cdiv(fss1, fss2, fs1, fs2, fssf1, fssf2);
            cmul(ffs1, ffs2, fssf1, fssf2, yr, yi);
            yr := sqr(n − 1) − n * (n − 1) * yr; yi := −n * (n − 1) * yi;
            csqrt(yr, yi, yr, yi);
            cdiv(n * ffs1, n * ffs2, 1 + yr, yi, yr, yi);
            yr := xr − yr; yi := xi − yi; xr := yr; xi := yi;
        end
    until fertig
end ;

begin
    writeln('Wohin mit dem Output ?');
    readln(stri); assign(tf, stri); rewrite(tf);
    writeln('Grad a[0], . . . , a[n] '); read(grad);
    for i := 0 to grad do read(ar[i], ai[i]);
    yr := 1; yi := −1;
    for n := grad downto 1 do
    begin
        xr := yr; xi := −yi;
        laguerreiteration; deflation(xr, xi);
    end ;
    writeln(tf,'Nullstellen');
    for i := 1 to grad do writeln(tf, ar[i],' +I*',ai[i]);
    close(tf);
end.
```

Dieses Programm löst die quadratische Gleichung von Aufgabe 4.11 b), welche dem Newtonprogramm so viel Mühe macht, in nur 2 Iterationsschritten! Für die Berechnung der ersten Nullstelle des Polynoms $P_3(x) = 3x^3 + x^2 - 5x + 1$ werden 4 Iterationen gebraucht. Die globale Konvergenz ist bei diesem Verfahren sehr gut.

Für das Polynom von Aufgabe 4.10 mit der sechsfachen Nullstelle bei -2 ist schon nach nur 2 Iterationsschritten das Abbruchkriterium erfüllt. Es

ist

$$\begin{aligned} x_2 &= -1.9999846312 + i \times 9.2316695373\text{E}{-6} \\ P_6(x_2) &= -1.9944729759\text{E}{-10} \\ P_6'(x_2) &= 2.6620351126\text{E}{-10} + i \times 1.7763568394\text{E}{-15} \\ P_6''(x_2) &= 1.0825124448\text{E}{-10} - i \times 6.4392935428\text{E}{-15}, \end{aligned}$$

sodass x_2 (zufällig) eine sehr genaue Näherung der Nullstelle ist. Lokal wird bei der Laguerre-Iteration das Polynom P_n durch $g(x) = a(x-x_1)(x-x_2)^{n-1}$ ersetzt, was gerade bei mehrfachen Nullstellen eine gute Ersatzfunktion ist. Wenn wir nun aber einen weiteren Iterationsschritt ausführen und mit den obigen Funktionswerten x_3 berechnen, erhält man die Laguerrekorrektur

$$h = -0.65693384026 - i \times 1.0850159689\text{E}{-6},$$

welche von dem guten Näherungswert x_2 wegführt zu $x_3 = -1.34305 + i \times 1.031668\text{E}{-5}$. Die Funktionswerte für P, P' und P'' sind durch Auslöschung ungenau berechnet und ihr Quotient, der in der Laguerreiterationsfunktion (4.5) auftritt, dadurch beliebig falsch. Mathematisch ergibt sich ein unbestimmter Ausdruck $0/0$. Es ist deswegen wichtig, die Iteration bei mehrfachen Nullstellen nach der Erfüllung des Abbruchkriteriums abzubrechen. Bei einfachen Nullstellen würde die zusätzliche Iteration nicht schaden, wir verzichten aber im obigen Programm auf diese Fallunterscheidung.

Aufgabe 4.21 *Die Gleichung*

$$\tan(x) = \sin(x) - \cos(x) \tag{4.5}$$

kann durch die Substitution $t = \sin(x)$ *und* $\cos(x) = \sqrt{1-t^2}$ *bzw.* $\tan(x) = t/\sqrt{1-t^2}$ *auf eine Gleichung 4. Grades in* t *umgeformt werden. Man löse diese Gleichung mit einem der Verfahren für Polynomnullstellen.*

In der neuen Variabeln t lautet die gegebene Gleichung

$$\frac{t}{\sqrt{1-t^2}} = t - \sqrt{1-t^2}.$$

Multipliziert man mit der Wurzel, isoliert sie und quadriert schliesslich die entstehende Gleichung, so erhält man

$$t^4 - t^3 - t^2 + t + \frac{1}{2} = 0.$$

Die Lösungen dieser Gleichung, berechnet mit dem Laguerreverfahren, sind:

$$\begin{array}{rcr} -8.1281478667E-01 & +I* & 5.0022208598E-12 \\ -4.2878157444E-01 & +I* & -5.0022208598E-12 \\ 1.1207981806E+00 & +I* & -4.2243262130E-01 \\ 1.1207981806E+00 & +I* & 4.2243262130E-01 \end{array}$$

Die ersten beiden Nullstellen sind reell und betragsmässig kleiner 1 und kommen damit als Sinuswerte in Frage:

$$\begin{aligned} \sin x = t_1 &= -8.1281478667E-01 \\ \Rightarrow x &= \begin{cases} x_1 &= -0.9489680204 + 2\pi k \\ x_2 &= 4.090560674 + 2\pi k \end{cases}, k = 0, \pm 1 \pm 2 \ldots \\ \sin x = t_2 &= -4.2878157444E-01 \\ \Rightarrow x &= \begin{cases} x_3 &= -0.4431436458 + 2\pi k \\ x_4 &= 3.584736299 + 2\pi k \end{cases}, k = 0, \pm 1 \pm 2 \ldots \end{aligned}$$

Man verifiziert durch Einsetzen, dass nur x_1 und x_4 Lösungen der Gleichung (4.5) sind. Die andern beiden Lösungsscharen wurden durch das Quadrieren der Gleichung eingeschleppt.

Aufgabe 4.22 *Die Gleichung*

$$1 + \cos x + \cos^2 x + \cos^3 x + \cos^4 x = 2$$

hat eine Lösung $x \in (0, \frac{\pi}{2})$. *Man löse die Gleichung durch Substitution* $z = \cos x$.

Die Gleichung $-1 + z + z^2 + z^3 + z^4 = 0$ kann mit einem der Programme für Polynomnullstellen gelöst werden. Man erhält z.B. mit dem Laguerreverfahren die Lösungen:

$$\begin{array}{rcr} -1.2906488013E+00 & +I* & 0.0000000000E+00 \\ -1.1407063116E-01 & +I* & -1.2167460040E+00 \\ -1.1407063116E-01 & +I* & 1.2167460040E+00 \\ 5.1879006368E-01 & +I* & 0.0000000000E+00 \end{array}$$

Als Cosinuswert kommt nur eine in Frage: $\cos(x) = 0.51879006368$. Die gesuchte Lösung ist damit $x = 1.025361278$.

Aufgabe 4.23 *Die goniometrische Gleichung*

$$\cos(x) + \sin(2x) = -0.5$$

kann mittels $\sin(2x) = 2\sin x \cos x$ *und* $\cos x = \sqrt{1 - \sin^2 x}$ *und der Substitution* $z = \sin x$ *auf eine Gleichung 4. Grades umgeformt werden. Man löse sie mittels eines der behandelten Verfahren.*

Mit der angegebenen Substitution erhält man die Gleichung

$$\sqrt{1 - z^2} + 2z\sqrt{1 - z^2} = -0.5,$$

welche man quadriert, um die Wurzeln wegzuschaffen. Geordnet nach Potenzen von z ergibt sich:

$$0.75 + 4z + 3z^2 - 4z^3 - 4z^4 = 0.$$

Die Lösungen davon sind:

$$\begin{array}{rcr} -8.7171070299E-01 & +I* & -1.5837405764E-01 \\ -8.7171070299E-01 & +I* & 1.5837405764E-01 \\ -2.4232019664E-01 & +I* & 0.0000000000E+00 \\ 9.8574160263E-01 & +I* & 0.0000000000E+00 \end{array}$$

Wieder kommt nur eine Lösung in Frage: $z = 0.98574160263$ und daraus folgt

$$x = \arcsin(z) = \begin{cases} x_1 &= 1.401725865 + 2\pi k \\ x_2 &= 1.739866789 + 2\pi k \end{cases}, k = 0, \pm 1 \pm 2 \ldots$$

Durch Einsetzen in die ursprüngliche Gleichung stellt man fest, dass nur x_2 eine Lösung ist. Durch das Quadrieren der Gleichung wurden die Lösungen x_1 eingeschleppt.

Aufgabe 4.24 *Bei einem rechtwinkligen Dreieck* ($\gamma = 90°$) *ist die Seite* $c = 5cm$. *Die Winkelhalbierende von* α *schneidet die Seite* b *im Punkte* P. *Die Strecke* $\overline{PC}$ *beträgt 1 cm. Man berechne die Seiten* a *und* b. *Man führe die entstehende Gleichung auf eine Polynomgleichung zurück (eventuell durch eine geeignete Substitution).*

Die Seite b kann auf zwei Arten ausgedrückt werden:

$$b = 5\cos\alpha \quad \text{und} \quad b = \frac{1}{\tan\frac{\alpha}{2}}.$$

Unter Verwendung der Identität

$$\tan\frac{\alpha}{2} = \sqrt{\frac{1-\cos\alpha}{1+\cos\alpha}}$$

erhält man die Gleichung

$$5\cos\alpha = \sqrt{\frac{1+\cos\alpha}{1-\cos\alpha}}. \tag{4.6}$$

Quadriert man diese Gleichung und führt die Unbekannte $z = \cos\alpha$ ein, so ergibt sich die Polynomgleichung

$$1 + z - 25z^2 + 25z^3 = 0.$$

Diese Gleichung hat 3 reelle Lösungen:

$$\begin{aligned} z_1 &= -0.16868089747 \;\Rightarrow \alpha_1 = 99.711^\circ \\ z_2 &= \;\; 0.26135497323 \;\Rightarrow \alpha_2 = 14.974^\circ \\ z_3 &= \;\; 0.90732592424 \;\Rightarrow \alpha_3 = 24.862^\circ \end{aligned}$$

Man verifiziert durch Einsetzen in Gleichung (4.6), dass nur α_2 und α_3 Lösungen sind. Durch das Quadrieren der Gleichung wurde die dritte Lösung eingeschleppt.

Kapitel 5

Lineare Gleichungssysteme

Aufgabe 5.1 *Unter Verwendung der Funktion* det *schreibe man ein vollständiges Programm zur Lösung von linearen Gleichungssystemen mittels der Cramerregel. Man zeige an Beispielen, dass es für grössere n instabil ist und schätze den Rechenaufwand ab.*

Im nachfolgenden Programm lesen wir zuerst die Koeffizientenmatrix **A** und die rechte Seite **b** ein. Anschliessend berechnen wir die Determinanten von **A** und der Matrizen $\mathbf{A_i}$. Zum Schluss wird noch als Kontrolle der Residuenvektor $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$ gerechnet und ausgedruckt. Der Algorithmus 5.2 (die **function** *det*) wird als Include-Flile dazugeladen.

```
(*$B- Aufgabe 5.1*)
program cramer;
type vektor = array [1..10] of real;
     matrix = array [1..10] of vektor
var i,j,n :integer; s,d :real;
     ai,a : matrix; b,x :array [1..10] of real;
     stri : string [20]; tf : text;

(*$I A5.2 *)

procedure print(n : integer; a : matrix);
var i,j : integer;
begin
  for i := 1 to n do
  begin
    writeln(tf);
    for j := 1 to n do write(tf, a[i, j] : 12 : 7)
  end ;
  writeln(tf)
end ;

begin
  writeln('Wohin mit dem Output ? ');
  read(stri); assign(tf,stri); rewrite(tf);
  writeln('n eingeben'); read(n);
  writeln('Matrix A zeilenweise eingeben');
  for i := 1 to n do for j := 1 to n do read(a[i, j]);
  writeln('rechte Seite eingeben');
```

```
    for i := 1 to n do read(b[i]);
    writeln(tf,'Gegebene Matrix'); print(n,a); writeln(tf);
    writeln(tf,'rechte Seite');
    for i := 1 to n do writeln(tf,b[i] : 12 : 7);
    d := det(n,a);
    writeln(tf,'Determinante von A = ',d);
    for i := 1 to n do
    begin
        ai := a;
        for j := 1 to n do ai[j,i] := b[j];
        x[i] := det(n,ai)/d;
        writeln(tf,'x[',i : 2,'] = ',x[i]);
    end ;
    writeln(tf); writeln(tf, 'Residuenvektor');
    for i := 1 to n do
    begin
        s := -b[i];
        for j := 1 to n do s := s + a[i,j] * x[j];
        writeln(tf,s);
    end ;
    close(tf)
end.
```

Eine für Testzwecke beliebte Matrix ist die *Hilbertmatrix*, welche für jedes n wie folgt definiert ist:

$$H_n = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & & \cdots & \frac{1}{2n-1} \end{pmatrix} \tag{5.1}$$

Diese Matrizen haben eine schlechte Kondition und die Auflösung von linearen Gleichungssystemen mit H_n als Koeffizientenmatrix ergibt numerisch ungenaue Resultate.

Die inverse Matrix von H_n ist ganzzahlig und hat auch eine schlechte Kondition. Es sind zum Beispiel

$$H_4^{-1} = \begin{pmatrix} 16 & -120 & 240 & -140 \\ -120 & 1200 & -2700 & 1680 \\ 240 & -2700 & 6480 & -4200 \\ -140 & 1680 & -4200 & 2800 \end{pmatrix} \tag{5.2}$$

$$H_5^{-1} = \begin{pmatrix} 25 & -300 & 1050 & -1400 & 630 \\ -300 & 4800 & -18900 & 26880 & -12600 \\ 1050 & -18900 & 79380 & -117600 & 56700 \\ -1400 & 26880 & -117600 & 179200 & -88200 \\ 630 & -12600 & 56700 & -88200 & 44100 \end{pmatrix}$$

$$H_6^{-1} = \begin{pmatrix} 36 & -630 & 3360 & -7560 & 7560 & -2772 \\ -630 & 14700 & -88200 & 211680 & -220500 & 83160 \\ 3360 & -88200 & 564480 & -1411200 & 1512000 & -582120 \\ -7560 & 211680 & -1411200 & 3628800 & -3969000 & 1552320 \\ 7560 & -220500 & 1512000 & -3969000 & 4410000 & -1746360 \\ -2772 & 83160 & -582120 & 1552320 & -1746360 & 698544 \end{pmatrix}$$

Wenn man nun das Gleichungssystem $\mathbf{H}_n^{-1}\mathbf{x} = \mathbf{e}_1$ auflöst, wobei $\mathbf{e}_1$ den ersten Einheitsvektor bedeutet, so muss die Lösung $\mathbf{x}$ die erste Kolonne von $\mathbf{H}_n$ sein. Man kann damit die numerisch erhaltenen Resultate sehr gut mit den exakten überprüfen.

In den folgenden Tabellen sind die Lösungen für $n = 4$, 5 und 6 angegeben, welche mit den Programmen *cramer* (Algorithmus 5.1) und mit Algorithmus 5.6 (Elimination mittels Givensrotationen) berechnet wurden.

	Cramerregel	Givenselimination
$n = 4$		
$x_1 =$	1.0000000000E+00	9.9999999959E−01
$x_2 =$	5.0000000000E−01	4.9999999970E−01
$x_3 =$	3.3333333333E−01	3.3333333310E−01
$x_4 =$	2.5000000000E−01	2.4999999982E−01
$n = 5$		
$x_1 =$	1.0000000000E+00	9.9999999927E−01
$x_2 =$	5.0000000000E−01	4.9999999988E−01
$x_3 =$	3.3333333333E−01	3.3333333346E−01
$x_4 =$	2.5000000000E−01	2.5000000023E−01
$x_5 =$	2.0000000000E−01	2.0000000027E−01

Überrascht stellt man fest, dass die Cramerregel für $n = 4$ und 5 genauere Resultate liefert als das Givensverfahren, das numerisch als sehr stabil gilt. Dies ist dadurch zu erklären, dass keine Rundungsfehler bei der Berechnung der Determinanten entstehen, *weil die Matrixelemente von H_4 und H_5 kleine*

ganze Zahlen sind. Dies ändert sofort, wenn man den Fall $n = 6$ betrachtet:

	Cramerregel	Givenselimination
$n = 6$		
$x_1 =$	1.0018657169E+00	1.0000001007E+00
$x_2 =$	5.0064753418E−01	5.0000008608E−01
$x_3 =$	3.3363885405E−01	3.3333340792E−01
$x_4 =$	2.5020720314E−01	2.5000006580E−01
$x_5 =$	2.0018464463E−01	2.0000005886E−01
$x_6 =$	1.6685253484E−01	1.6666671991E−01

Jetzt ist der Rechenbereich zu klein, um die bei der Berechnung der Determinanten auftretenden Produkte exakt darzustellen. Die Rundungsfehler sind deutlich bemerkbar und die Givensmethode liefert wesentlich bessere Resultate, deren Genauigkeit nur durch die Kondition der Matrix bestimmt sind. Zudem bemerkt man auch, dass die Resultate mit der Cramerregel viel langsamer berechnet werden.

Wenn man durch Skalieren der Gleichungen die schönen ganzen Zahlen zerstört, sieht man, dass die Cramerregel auch schon für kleineres n ungenauere Werte liefert. Löst man etwa

$$\left(7777.7\ \mathbf{H}_5^{-1}\right)\mathbf{y} = \mathbf{e}_1$$

und berechnet anschliessend $\mathbf{x} = 7777.7\ \mathbf{y}$, so ergibt sich beim Givensverfahren kein wesentlicher Unterschied in der Genauigkeit, aber die mit der Cramerregel erhaltenen Werte sind schlechter:

	Cramerregel	Givenselimination
$x_1 =$	9.9999556275E−01	1.0000000021E+00
$x_2 =$	4.9999728289E−01	5.0000000251E−01
$x_3 =$	3.3333146889E−01	3.3333333574E−01
$x_4 =$	2.4999862457E−01	2.5000000223E−01
$x_5 =$	1.9999893312E−01	2.0000000205E−01

Der Rechenaufwand für die Cramerregel kann wie folgt abgeschätzt werden. Durch die Laplace'sche Entwicklung (5.14∗) wird die Berechnung einer n–reihigen Determinante auf n Berechnungen von jeweils einer $n-1$–reihigen Determinante zurückgeführt. Sei d_n die Anzahl der Flops um eine Determinante zu berechnen. Es gilt daher nach Gleichung (5.14∗)

$$d_n = n\, d_{n-1}$$

und wegen $d_2 = 2$ folgt unmittelbar

$$d_n = n!$$

Um nun ein Gleichungssystem mit n Gleichungen aufzulösen, muss man $n+1$ Determinanten berechnen, nämlich

$$\det \mathbf{A} \quad \text{und} \quad \det \mathbf{A}_1, \ldots, \det \mathbf{A}_n$$

somit ist der Rechenaufwand für die Auflösung eines linearen Gleichungssystems mit dem Programm *cramer* proportional zu $(n+1)\,!$, was sehr schnell zu astronomischen Rechenzeiten führt.

Aufgabe 5.2 *Man schreibe ein Programm, das lineare Gleichungssysteme mit rationalen Koeffizienten ohne Rundungsfehler exakt löst.*
Vorgehen: Die Koeffizienten haben den Typus

type *bruch* = **record**
zaehler, nenner : *integer*
end

Das Programm gauss kann übernommen und modifiziert werden. Alle Operationen müssen durch entsprechende Prozeduraufrufe ersetzt werden, z.B. wird aus

$$faktor := a[k,i]/a[i,i]$$

jetzt

$$division(a[k,i], a[i,i], faktor);$$

Die Prozedur division führt die Division der beiden Brüche in integer Arithmetik aus. Um einen Überlauf möglichst zu vermeiden, muss nach jeder Operation das Resultat gekürzt werden. Man verwende dazu den Euklid'schen Algorithmus (siehe Kap. 2).

Da in TURBO PASCAL ein Sprung aus einem Block heraus nicht erlaubt ist, müssen wir zuerst das Programm *gauss* (den Algorithmus 5.5) umschreiben. Wir führen dazu eine boolean Variable *sing* ein, welche angibt, ob die Koeffizientenmatrix singulär ist. Die Hauptschleife in der Prozedur *elimination* wird als **repeat** -Schleife umgeschrieben:

$$i := 0;\ \textbf{repeat}\ i := i+1;\ \ldots\ldots\ \textbf{until}\ (i = n)\ \textbf{or}\ sing$$

Als weitere Modifikation übernehmen wir den Algorithmus 5.3 (Rückwärtseinsetzen) als Include-File. Wir erhalten damit folgende Neufassung von Algorithmus 5.5:

```
(*$B- Algorithmus 5.5*)
program gauss ;
const nn = 20;
type vektor = array [1..nn] of real;
     matrix = array [1..nn] of vektor;
var a : matrix; b, x : vektor; i, n : integer; sing : boolean;

(*$I rueckwae ( Algorithmus 5.3)*)

procedure eingabe(var a : matrix; var b : vektor);
var i,j : integer;
begin
  writeln('eine Gleichung nach der andern eingeben');
  for i := 1 to n do
  begin
    writeln('Gleichung Nr.',i : 3);
    for j := 1 to n do read(a[i, j]); readln(b[i]);
  end
end ;

procedure singulaer;
begin
  writeln('Matrix ist singulaer');
end ;

procedure elimination;
var i,j,k, jmax : integer; h : vektor; z, faktor : real;
begin
  i := 0;
  repeat
    i := i + 1; jmax:= i;
    for j := i + 1 to n do
    if abs(a[j, i]) > abs(a[jmax, i]) then jmax:= j;
    sing:= a[jmax, i] = 0;
    if sing then singulaer
    else
    begin
      (* Gleichungen tauschen *)
      if jmax <> i then
      begin
        h := a[i]; a[i] := a[jmax]; a[jmax] := h;
        z := b[i]; b[i] := b[jmax]; b[jmax] := z
      end ;

      (* Elimination *)
```

```
        for k := i + 1 to n do
        begin
            faktor:= a[k, i]/a[i, i];
            for j := i + 1 to n do
            a[k, j] := a[k, j]−faktor∗a[i, j];
            b[k] := b[k]−faktor∗b[i]
        end
    end
  until (i = n) or sing
end ;

begin
  writeln('Wieviele Unbekannte');read(n);
  eingabe(a, b);
  elimination;
  if not sing then
  begin
    rueckwaertseinsetzen(n, a, b, x);
    writeln('Loesung');
    for i := 1 to n do write(x[i]);
  end
end.
```

Wendet man diesen Algorithmus auf das Gleichungssystem $\mathbf{H}_4^{-1}\mathbf{x} = \mathbf{e}_1$ (vgl. Aufgabe 5.1) an, so erhält man die Lösung

$$\mathbf{x} = \begin{pmatrix} 1.0000000003\text{E}{+00} \\ 5.0000000020\text{E}{-01} \\ 3.3333333349\text{E}{-01} \\ 2.5000000013\text{E}{-01} \end{pmatrix}$$

welche so genau ist wie jene, die mit der Givenselimination berechnet wurde. Löst man aber das System

$$\begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 4 \\ 8 \\ 12 \end{pmatrix} \tag{5.3}$$

so erhält man eine exakte Lösung $\mathbf{x} = (0, -1, 2)^T$, *obwohl die Koeffizientenmatrix singulär ist*. Die Rundungsfehler haben bewirkt, dass das Pivotelement nicht genau 0 wurde und, weil das System kompatibel ist, wurde eine der unendlich vielen Lösungen korrekt berechnet.

Wählt man aber beim letzten Gleichungssystem als rechte Seite den Einheitsvektor $\mathbf{e}_1 = (1, 0, 0)^T$, so wird das System inkompatibel und man

erhält die Lösung

$$\mathbf{x} = \begin{pmatrix} 9.1625968981\text{E+}10 \\ -1.8325193796\text{E+}11 \\ 9.1625968983\text{E+}10 \end{pmatrix}$$

welche durch ihre Grösse die Singularität der Matrix anzeigt.

Der Algorithmus 5.5 lässt sich nun leicht für das exakte Rechnen mit Brüchen übersetzen. Wir übernehmen die Bruchoperationen von Aufgabe 2.4, welche mit dem Include-File *bruchop* dazugeladen werden. Ebenso übernehmen wir die Prozeduren *liesbruch, schreibbruch* und *kuerzen*. Man erhält damit

```
(*$B- Aufgabe 5.2*)
program bruchgauss;
const nn = 20;
type longint = integer; (* integer[36] fuer UCSD PASCAL *)
   bruch = record
              zaehler,nenner: longint
           end ;
   vektor = array [1..nn] of bruch;
   matrix = array [1..nn] of vektor;

var a : matrix; c,b,x,y : vektor;
   i, k, l, m, n, imax : integer; ch : char;
   h,s : bruch; sing : boolean;
   ind : array [1..nn] of integer;
   tf : text; stri : string [20];

(*$I kuerzen *)
(*$I liesbruch *)
(*$I bruchop *)
(*$I schreibbruch *)

procedure matdruck;
var i,k : integer;
begin
   writeln(tf);
   for i := 1 to n do
   begin
      for k := 1 to n do schreibbruch(a[i,k]);
      write(tf,' '); schreibbruch(b[i]);
      writeln(tf);
   end ;
end ;
```

```
procedure rueckwaertseinsetzen;
var s, z : bruch; i,j : integer;
begin
   for i := n downto 1 do
   begin
      s := b[i];
      for j := i + 1 to n do
      begin
         mal(a[i, j], x[j], z); sub(s, z, s)
      end ;
      division(s, a[i, i], x[i]);
   end ;
end ;

procedure elimination(var sing: boolean);
var i, j, k, jmax : integer; h : vektor; z, faktor : bruch;
begin
   i := 0;
   repeat
      i := i + 1; jmax := 0; j := i;
      repeat
         if a[j, i].zaehler <> 0 then jmax := j;
         j := j + 1
      until (j > n) or ( jmax<> 0);
      (* es genuegt a[ jmax,i] <> 0 zu finden *)
      sing := jmax = 0;
      if notsing then
      begin
         if jmax <> i then
         begin
            h := a[i]; a[i] := a[jmax]; a[jmax] := h;
            z := b[i]; b[i] := b[jmax]; b[jmax] := z
         end ;
         for k := i + 1 to n do
         begin
            division(a[k, i], a[i, i],faktor);
            for j := i + 1 to n do
            begin
               mal( faktor, a[i, j], z); sub(a[k, j], z, a[k, j])
            end ;
            mal( faktor, b[i], z); sub(b[k], z, b[k])
         end ;
end ;
```

```
    until (i = n) or sing
end (* Elimination *);

begin
    writeln('wohin mit dem Output');
    readln(stri); assign(tf,stri); rewrite(tf);
    writeln('wieviele Unbekannte'); readln(n);
    writeln('Gleichungen zeilenweise eingeben');
    for i := 1 to n do
    begin
        for k := 1 to n do liesbruch(a[i,k],ch);
        liesbruch(b[i],ch);
    end ;
    writeln(tf,'gegebenes Gleichungssystem:');
    matdruck;
    elimination(sing);
    writeln(tf); writeln(tf,'nach der Elimination');
    matdruck; writeln(tf);
    if sing then writeln(tf,'Matrix ist singulaer')
    else
    begin
        rueckwaertseinsetzen;
        writeln(tf,'Loesung:'); writeln(tf);
        for i := 1 to n do schreibbruch(x[i]);
        writeln(tf);
    end ;
    close(tf)
end.
```

Leider stösst man in TURBO PASCAL sehr rasch an die Überlaufgrenze für die *integer* Zahlen, man kann also nur sehr kleine Gleichungssysteme lösen. Zum Beispiel erhält man die exakte Lösung für das Gleichungssystem $\mathbf{H}_4^{-1}\mathbf{x} = \mathbf{e}_1$ (vgl. (5.2)) und beim Gleichungssystem (5.3), das vom Algorithmus *gauss* nicht als singulär erkannt wurde, wird die Rechnung abgebrochen wie das folgende Bildschirmprotokoll zeigt:

```
Wieviele Unbekannte
3
Gleichungen zeilenweise eingeben

1/1     2/1     3/1      4/1
5/1     6/1     7/1      8/1
9/1    10/1    11/1     12/1
```

```
gegebenes Gleichungssystem:
1      2      3          4
5      6      7          8
9     10     11         12
nach der Elimination
1      2      3          4
5     -4     -8        -12
9     -8      0          0
Matrix ist singulaer
```

Beim Gleichungssystem $\mathbf{H}_4\mathbf{x} = \mathbf{e}_1$ mit der Hilbertmatrix $\mathbf{H}_4$ (5.1) entsteht bei der Elimination ein *integer* Überlauf und man erhält das falsche Resultat

$$-10512/13615 \quad 9672/841 \quad 816/29 \quad 36/29.$$

Wenn man die Bruchoperationen abändert und die Brüche als mehrfachgenaue Zahlen darstellt (s. Lehrbuch Kap. 2.7), kann dieser Fehler behoben werden. Dies sei dem interessierten Leser als weitere Aufgabe überlassen.

Aufgabe 5.3 *Man schreibe ein Programm, das lineare Gleichungssysteme mit komplexen Koeffizienten löst.*
Vorgehen: Man könnte analog wie bei der Aufgabe 5.2 alle Operationen durch entsprechende Prozeduren für komplexe Arithmetik ersetzen. Wir wollen aber hier einen anderen Weg zeigen, der darin besteht, das komplexe Gleichungssystem auf ein reelles mit doppelt so vielen Unbekannten zurückzuführen. Es genügt dann, aus den komplexen Koeffizienten das reelle System aufzustellen, welches dann mittels des Programms gauss *gelöst werden kann. Sei also*

$$\mathbf{C}\,\mathbf{z} = \mathbf{d} \tag{5.38$*$}$$

das gegebene komplexe System, wobei $\mathbf{C}$ *eine komplexe* $n \times n$ *Matrix und* $\mathbf{d}$ *ein Vektor mit komplexen Komponenten ist. Wir trennen* $\mathbf{C}$ *und* $\mathbf{d}$ *in Real- und Imaginärteil:*

$$\mathbf{C} = \mathbf{A} + i\,\mathbf{B} \quad \textit{und} \quad \mathbf{d} = \mathbf{e} + i\,\mathbf{f}$$

Auch für den Unbekanntenvektor machen wir den Ansatz

$$\mathbf{z} = \mathbf{x} + i\,\mathbf{y}.$$

Die Gleichung (5.38$*$) *lautet dann*

$$(\mathbf{A} + i\,\mathbf{B})(\mathbf{x} + i\,\mathbf{y}) = \mathbf{e} + i\,\mathbf{f}.$$

Wenn wir ausmultiplizieren und ordnen, ist

$$(\mathbf{A}\,\mathbf{x} - \mathbf{B}\,\mathbf{y}) + i(\mathbf{A}\,\mathbf{y} + \mathbf{B}\,\mathbf{x}) = \mathbf{e} + i\,\mathbf{f}. \tag{5.39*}$$

Die Gleichung (5.39∗) *kann nur gelten, wenn Real- und Imaginärteile auf beiden Seiten übereinstimmen, d.h. wenn*

$$\begin{aligned} \mathbf{A}\,\mathbf{x} - \mathbf{B}\,\mathbf{y} &= \mathbf{e} \\ \mathbf{B}\,\mathbf{x} + \mathbf{A}\,\mathbf{y} &= \mathbf{f} \end{aligned} \tag{5.40*}$$

Die Gleichungen (5.40∗) *stellen aber ein* $(2n) \times (2n)$ *lineares Gleichungssystem mit reellen Koeffizienten für den Unbekanntenvektor*

$$\mathbf{w} := \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$$

dar, nämlich

$$\begin{pmatrix} \mathbf{A} & -\mathbf{B} \\ \mathbf{B} & \mathbf{A} \end{pmatrix} \mathbf{w} = \begin{pmatrix} \mathbf{e} \\ \mathbf{f} \end{pmatrix}$$

Wir können den Algorithmus 5.5 grösstenteils übernehmen. Die Prozedur *eingabe* wird durch eine neue Prozedur *lies* ersetzt, welche die Real- und Imaginärteile der Koeffizientenmatrix $\mathbf{C}$ einliest und gleichzeitig die reelle Koeffizientenmatrix des Gleichungssystems (5.40∗) erzeugt. Im Hauptprogramm wird n verdoppelt, da das reelle System doppelt so gross wie das komplexe ist:

```
(*$B- Aufgabe 5.3*)
program lincompl;
const nn = 20;
type vektor = array [1..nn] of real;
   matrix = array [1..nn] of vektor;
var a : matrix; b,x : vektor; i,n : integer; sing : boolean;

procedure lies(var a : matrix; var b : vektor);
var i,j : integer;
begin
   writeln('Gleichungen zeilenweise eingeben');
   for i := 1 to n do
   begin
      for j := 1 to n do
      begin
         write('Re A[',i : 2,',',j : 2,'] =');
         read(a[i, j]); a[n + i, n + j] := a[i, j];
```

```
            write('Im A[',i : 2,',',j : 2,'] =');
            read(a[n + i, j]); a[i, n + j] := −a[n + i, j];
            writeln;
        end ;
        write('Re b[',i : 2,'] =');
        read(b[i]);
        write('Im b[',i : 2,'] =');
        read(b[n + i]);
    end ;
end ;

procedure singulaer;
begin
    writeln('Matrix ist singulaer');
end ;

(*$I rueckwae ( Algorithmus 5.3)*)

procedure elimination;
var i,j,k, jmax : integer; h : vektor; z, faktor : real;
begin
    i := 0;
    repeat
        i := i + 1; jmax:= i;
        for j := i + 1 to n do
        if abs(a[j, i]) > abs(a[jmax, i]) then jmax:= j;
        sing:= a[jmax, i] = 0;
        if sing then singulaer
        else
begin
    (* Gleichungen tauschen *)
    if jmax <> i then
    begin
        h := a[i]; a[i] := a[jmax]; a[jmax] := h;
        z := b[i]; b[i] := b[jmax]; b[jmax] := z
    end ;

    (* Elimination *)
    for k := i + 1 to n do
    begin
        faktor:= a[k, i]/a[i, i];
        for j := i + 1 to n do
        a[k, j] := a[k, j]−faktor∗a[i, j];
        b[k] := b[k]−faktor∗b[i]
    end
```

```
      end
   until (i = n) or sing
end ;

begin
   writeln('Wieviele Unbekannte'); read(n);
   lies(a, b);
   n := 2 * n;
   elimination;
   rueckwaertseinsetzen(n, a, b, x);
   n:=n div 2;
   for i := 1 to n do
   writeln(' z[',i : 2,'] =', x[i],' +i* ', x[n + i]);
end.
```

Aufgabe 5.4 *Für das lineare Gleichungssystem*

$$\begin{pmatrix} 2 & 1 & 3 & -1 \\ -6 & 0 & 1 & -1 \\ 4 & 2 & 0 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 5 \\ -5 \\ 7 \\ 6 \end{pmatrix} \tag{5.4}$$

bestimme man die drei Eliminationsmatrizen $\mathbf{C}^{(i)}$ *sowie die Dreieckszerlegung. Die Elimination kann ohne Vertauschungen durchgeführt werden.*

Nach Gleichung (5.24∗) wird

$$\mathbf{C}^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}$$

und die Multiplikation von (5.4) von links damit bewirkt den ersten Eliminationsschritt

$$\begin{pmatrix} 2 & 1 & 3 & -1 \\ 0 & 3 & 10 & -4 \\ 0 & 0 & -6 & 3 \\ 0 & 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 5 \\ 10 \\ -3 \\ 1 \end{pmatrix} \tag{5.5}$$

Daraus ergibt sich die zweite Eliminationsmatrix:

$$\mathbf{C}^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{3} & 0 & 1 \end{pmatrix}$$

Wenn man damit das System (5.5) von links multipliziert erhält man

$$\begin{pmatrix} 2 & 1 & 3 & -1 \\ 0 & 3 & 10 & -4 \\ 0 & 0 & -6 & 3 \\ 0 & 0 & -\frac{13}{3} & \frac{7}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 5 \\ 10 \\ -3 \\ -\frac{7}{3} \end{pmatrix} \tag{5.6}$$

woraus man schliesslich die Matrix

$$\mathbf{C}^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{13}{18} & 1 \end{pmatrix}$$

abliest, welche (5.6) auf ein System mit oberer Dreiecksmatrix transformiert:

$$\begin{pmatrix} 2 & 1 & 3 & -1 \\ 0 & 3 & 10 & -4 \\ 0 & 0 & -6 & 3 \\ 0 & 0 & 0 & \frac{1}{6} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 5 \\ 10 \\ -3 \\ -\frac{1}{6} \end{pmatrix} \tag{5.7}$$

Die untere Dreiecksmatrix $\mathbf{L}$ ergibt sich aus (5.33$*$)

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 1 & \frac{1}{3} & \frac{13}{18} & 1 \end{pmatrix}$$

Aufgabe 5.5 *Man zeige, dass das Produkt von zwei oberen (bzw. unteren) Dreiecksmatrizen wieder eine obere (bzw. untere) Dreiecksmatrix ist.*

Wenn $\mathbf{A}$ und $\mathbf{B}$ zwei obere Dreiecksmatrizen sind, so gilt

$$a_{ij} = b_{ij} = 0 \quad \text{für} \quad i > j.$$

Für das Element c_{ij} mit $i > j$ der Produktmatrix $\mathbf{C}$ gilt

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Da $\mathbf{A}$ eine obere Dreiecksmatrix ist, muss die Summation nur für $k \geq i$ erfolgen, d. h. es ist

$$c_{ij} = \sum_{k=i}^{n} a_{ik} b_{kj} = 0,$$

weil $k \geq i > j$ und somit $b_{kj} = 0$. Somit ist $\mathbf{C}$ auch eine obere Dreiecksmatrix.

Analog schliesst man für untere Dreiecksmatrizen aus

$$a_{ij} = b_{ij} = 0 \quad \text{für} \quad i < j,$$

dass

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = \sum_{k=1}^{i} a_{ik} b_{kj} = 0$$

wird, weil $b_{kj} = 0$ ist für $k \leq i < j$.

Aufgabe 5.6 *Wenn ein Gleichungssystem eine singuläre Koeffizientenmatrix besitzt, kann es infolge Rundungsfehler vorkommen, dass das Gleichungssystem numerisch nicht singulär ist, weil die Rundungsfehler verhindern, dass bei der Elimination ein Kolonnenvektor des reduzierten Systems exakt null wird. Die Lösung, die man erhält, ist entweder unsinnig gross (in diesem Fall war das System inkompatibel) oder man merkt ihr gar nichts an (das System ist lösbar und wir erhalten eine der unendlich vielen Lösungen des inhomogenen Systems).*

Sollte einmal der Fall auch numerisch auftreten, dass die Prozedur singulaer *aufgerufen wird, zeigt es sich, dass es oft besser ist, künstlich dem Pivotelement a_{ii} einen kleinen Wert, etwa*

$$a_{ii} := \max_{i,j} |a_{ij}| \varepsilon,$$

wobei ε die Maschinengenauigkeit bedeutet, zuzuweisen und die Rechnung fortzusetzen. Erhält man eine grosse Lösung $\mathbf{x}$, so kann man diese normieren (etwa durch die grösste Komponente dividieren) und erhält so eine Lösung des homogenen Systems. Man ändere die Prozedur singulaer *entsprechend ab und führe einige Experimente mit singulären Gleichungssystemen durch.*

Wenn wir wie vorgeschlagen bei der Elimination die Matrixelemente leicht ändern, treten keine singulären Matrizen mehr auf. Man kann immer n Eliminationsschritte durchführen und daher ersetzen wir die **while** -Schleife in *elimination* wieder durch eine **for** -Schleife. Beim Einlesen der Matrixelemente wird gleichzeitig der Wert

$$amax := \max_{1 \leq i,j \leq n} |a_{ij}|$$

berechnet. Um die erhaltene Lösung in das ursprüngliche Gleichungssystem einsetzen zu können, verwenden wir die Prozedur *ax*, welche das Produkt $\mathbf{Ax}$

berechnet und ausdruckt. In der Prozedur *singulaer* wird das Pivotelement $a[i,i] := amax * 1E{-}11$ gesetzt. Da i eine *lokale* Variable der Prozedur *elimination* ist, muss die Prozedur *singulaer* auch als lokale Prozedur in *elimination* verwendet werden.

Damit bei gleicher Koeffizientenmatrix mit mehreren rechten Seiten experimentiert werden kann, wird die Matrix **A** von einem File eingelesen. Um die Singularität der Matrix besser zu erkennen, wird das reduzierte System $\mathbf{Rx} = \mathbf{y}$ ausgedruckt. Man erhält damit folgendes Programm.

```
(*$B- Aufgabe 5.6*)
program gauss ;
const nn = 20;
type vektor = array [1..nn] of real;
   matrix = array [1..nn] of vektor;
var aa,a : matrix; b,x : vektor; i,j,n : integer; amax : real;
   tf : text; stri : string [20];

(*$I rueckwae ( Algorithmus 5.3)*)

procedure eingabe(var a : matrix; var b : vektor);
var i,j : integer;
begin
   amax := 0;
   writeln('eine Gleichung nach der anderen eingeben');
   for i := 1 to n do
   begin
      writeln('Gleichung Nr.',i : 3);
      for j := 1 to n do
      begin
         read(tf, a[i, j]);
         if amax< abs(a[i, j]) then amax:= abs(a[i, j])
      end ;
      readln(b[i]);
   end ;
   aa := a
end ;

procedure ax;
var i,j : integer;s : real;
begin
   writeln('A * x=');
   for i := 1 to n do
   begin s := 0;
      for j := 1 to n do s := s + aa[i, j] * x[j];
      writeln(s);
```

```
    end ;
    writeln
end ;

procedure elimination;
var i,j,k, jmax : integer; h : vektor; z, faktor : real;

procedure singulaer;
begin
    writeln('Matrix ist singulaer');
    a[i, i] := amax * 1e-11;
end ;

begin
    for i := 1 to n do
    begin
        jmax:= i;
        for j := i + 1 to n do
        if abs(a[j, i]) > abs(a[jmax,i]) then jmax:= j;
        if a[jmax,i] = 0 then singulaer;
        (* Gleichungen tauschen *)
        if jmax<> i then
        begin
            h := a[i]; a[i] := a[jmax]; a[jmax] := h;
            z := b[i]; b[i] := b[jmax]; b[jmax] := z
        end ;

        (* Elimination *)
        for k := i + 1 to n do
        begin
            faktor:= a[k, i]/a[i, i];
            for j := i + 1 to n do
            a[k, j] := a[k, j]-faktor*a[i, j];
            b[k] := b[k]-faktor*b[i]
        end
    end
end ;

begin
    writeln('von welchem File kommt die Matrix');
    readln(stri); assign(tf,stri); reset(tf);
    writeln('Wieviele Unbekannte'); read(n);
    repeat
        eingabe(a, b);
        writeln('gegebene Matrix ');
```

```
    for i := 1 to n do
    begin
      for j := 1 to n do write(a[i,j]:8:3);
      writeln;
    end ;
    elimination;
    writeln('Reduziertes System');
    for i := 1 to n do
    begin
      for j := 1 to i-1 do write(' ':12);
      for j := i to n do write(a[i,j]:11,' ');
      writeln(' ':5,b[i]:11);
    end ;
    rueckwaertseinsetzen(n,a,b,x);
    writeln('Loesung');
    for i := 1 to n do writeln(x[i]);
    ax;
    writeln('Durch welche Unbekannte skalieren?');
    readln(i); amax:= x[i];
    for j := 1 to n do
    begin x[j] := x[j]/amax; writeln(x[j]) end ;
    ax;
    writeln; reset(tf);
    writeln('nochmals ? RET, fertig: CTRL-Z')
  until eof;
end.
```

Mit diesem Programm erhalten wir für das Gleichungssystem

$$\begin{pmatrix} 2 & 4 & 2 & 5 \\ 4 & -2 & 0 & -8 \\ 6 & 2 & 2 & -3 \\ 10 & 5 & 4 & -2 \end{pmatrix} x = \begin{pmatrix} 13 \\ -6 \\ 7 \\ 17 \end{pmatrix} \tag{5.8}$$

den Output:

Matrix ist singulaer

Reduziertes System

```
 1.00000E+01  5.00000E+00  4.00000E+00 -2.0000E+00    1.70000E+01
              -4.0000E+00 -1.6000E+00 -7.2000E+00   -1.2800E+01
                           1.00000E-10 -7.2760E-12    0.00000E+00
                                       -2.1828E-11    0.00000E+00
```

Loesung

$$\begin{matrix} 1.0000000000\text{E}{-}01 \\ 3.2000000000\text{E}{+}00 \\ 0.0000000000\text{E}{+}00 \\ 0.0000000000\text{E}{+}00 \end{matrix}$$

Die Fehlermeldung '*Matrix ist singulaer*' zeigt, dass hier ein Pivotelement exakt Null geworden ist. Die Matrix hat daher numerisch nicht den Rang 4, sondern höchstens 3. Andrerseits sieht man der Matrix $\mathbf{R}$ des reduzierten Systems an, dass der Rang 2 sein muss: die beiden letzten Zeilen enthalten nur Elemente der Grössenordnung der Rundungsfehler.

Wie man leicht verifiziert, ist die Lösung eine exakte Lösung des Gleichungssystems. Das System war somit kompatibel und das Programm *gauss* liefert *eine* der unendlich vielen Lösungen. Die rechte Seite $\mathbf{b}$ wurde so konstruiert, dass $\mathbf{x} = (1,1,1,1)^T$ eine Lösung ist. Wir erhalten diese Lösung jedoch nicht, sondern eine andere partikuläre Lösung.

Die allgemeine Lösung des Systems (5.8) ist gegeben durch

$$\mathbf{x} = \mathbf{x}_i + \lambda \mathbf{x}_h^{(1)} + \mu \mathbf{x}_h^{(2)} \quad \text{mit} \quad -\infty < \lambda, \mu < \infty,$$

wobei $\mathbf{x}_i$ eine partikuläre Lösung des inhomogenen Systems ist und $\mathbf{x}_h^{(1)}$ und $\mathbf{x}_h^{(2)}$ zwei linear unabhängige Lösungen des homogenen Systems sind.

Mit der oben gefundenen Lösung haben wir $\mathbf{x}_i = (0.1, 3.2, 0, 0)^T$ bestimmt. Wenn man eine andere rechte Seite für das Gleichungssystem (5.8) wählt, etwa $\mathbf{b} = (1,0,0,0)^T$, so erhält man als Lösung

$$\mathbf{x} = \begin{pmatrix} -4.9727616273\text{E}{+}10 \\ 8.3796705416\text{E}{+}10 \\ -3.3333333333\text{E}{+}09 \\ -4.5812984491\text{E}{+}10 \end{pmatrix} \text{ mit } \mathbf{A}\mathbf{x} = \begin{pmatrix} 5.0000000000\text{E}{-}01 \\ 0.0000000000\text{E}{+}00 \\ 2.5000000000\text{E}{-}01 \\ 5.0000000000\text{E}{-}01 \end{pmatrix} \neq \mathbf{b}$$

Bei einer so grossen Lösung $\mathbf{x}$ kann infolge der Rundungsfehler nicht erwartet werden, dass $\mathbf{A}\mathbf{x} = \mathbf{b}$ ist. Der Vektor $\mathbf{b}$ ist im Verhältmis dazu so klein, dass man ihn ebenso als Nullvektor betrachten kann. Multipliziert man das Gleichungssystem mit 1E−10, so folgt aus

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \Rightarrow \mathbf{A}(\mathbf{x} \times 1\text{E}{-}10) &= \mathbf{b} \times 1\text{E}{-}10 \approx \mathbf{0}, \end{aligned}$$

dass $\mathbf{x}$ oder auch $\mathbf{x} \times 1\text{E}{-}10$ eine Lösung des homogenen Systems sein muss.

Skaliert man also den Vektor $\mathbf{x}$, indem er durch $x[2]$ dividiert wird, so erhält man

$$\mathbf{x}_i^{(1)} = \begin{pmatrix} -5.9343163942\text{E}{-}01 \\ 1.0000000000\text{E}{+}00 \\ -3.9778811312\text{E}{-}02 \\ -5.4671581971\text{E}{-}01 \end{pmatrix} \text{ mit } \mathbf{A}\mathbf{x}_i^{(1)} = \begin{pmatrix} 7.2759576142\text{E}{-}12 \\ 0.0000000000\text{E}{+}00 \\ 5.4569682106\text{E}{-}12 \\ 9.0949470177\text{E}{-}12 \end{pmatrix}$$

d.h. wir haben eine Lösung des homogenen Systems gefunden!

Für die rechte Seite $\mathbf{b} = (0, 0, 1, 0)^T$ erhält man die Lösung

$$\mathbf{x} = \begin{pmatrix} -2.0000000000\text{E+}09 \\ -4.0000000000\text{E+}09 \\ 1.0000000000\text{E+}10 \\ 0.0000000000\text{E+}00 \end{pmatrix} \quad \text{oder skaliert} \quad \mathbf{x}_h^{(2)} = \begin{pmatrix} -0.2 \\ -0.4 \\ 1.0 \\ 0.0 \end{pmatrix} \tag{5.9}$$

und man verifiziert, dass $\mathbf{x}_h^{(2)}$ eine zweite Lösung des homogenen Systems ist, welche von $\mathbf{x}_h^{(1)}$ linear unabhängig ist. Somit lautet die allgemeine Lösung des Systems

$$\mathbf{x} = \begin{pmatrix} 0.1 \\ 3.2 \\ 0.0 \\ 0.0 \end{pmatrix} + \lambda \begin{pmatrix} -5.9343163942\text{E}{-}01 \\ 1.0000000000\text{E+}00 \\ -3.9778811312\text{E}{-}02 \\ -5.4671581971\text{E}{-}01 \end{pmatrix} + \mu \begin{pmatrix} -0.2 \\ -0.4 \\ 1.0 \\ 0.0 \end{pmatrix}$$

Abschliessend sei noch bemerkt, dass die Bestimmung des Ranges einer Matrix numerisch ein schwieriges Problem ist. Die zur Zeit zuverlässigste Methode dafür ist die *Singulärwertzerlegung* .

Aufgabe 5.7 *Man schreibe ein Programm für die Auflösung von linearen Gleichungssystemen mittels Gauss'scher Dreieckszerlegung. Das Programm gauss kann dabei so geändert werden, dass zuerst nur die Dreieckszerlegung*

$$\mathbf{PA} = \mathbf{LR}$$

berechnet wird. Anschliessend können beliebig viele rechte Seiten eingelesen werden und die zugehörigen Lösungen werden durch Vorwärts- und Rückwärtseinsetzen berechnet. Die Zeilenvertauschungen merkt man sich am einfachsten mittels eines Permutationsvektors

$$zeile : \textbf{array } [1..n] \textbf{ of } integer;$$

Zu Beginn wird der Vektor zeile durch

$$\textbf{for } i := 1 \textbf{ to } n \textbf{ do } zeile[i] := i;$$

initialisiert. Bei jeder Vertauschung werden auch die entsprechenden Komponenten von zeile mitgetauscht

$$j := zeile[i];\ zeile[i] := zeile[jmax];\ zeile[jmax] := j$$

Die Umordnung des Vektors **b** *erfolgt dann durch*

$$\textbf{for } i := 1 \textbf{ to } n \textbf{ do } \tilde{b}[i] := b[zeile[i]]$$

Die Matrix **A** *kann durch die beiden Matrizen* **L** *und* **R** *überschrieben werden. An Stelle der in* **A** *entstehenden Nullen speichert man die Elemente der Matrix* **L** *ab. Die Diagonale von* **L** *braucht nicht gespeichert zu werden, da sie immer* 1 *ist.*

Die Lösung eines linearen Gleichungssystems erfolgt bei dieser Aufgabe in drei Schritten:

1. Dreieckszerlegung von **A**.
2. Vertauschen von **b** und Vorwärtseinsetzen (vgl. Gleichung (5.36∗)
3. Rückwärtseinsetzen nach Gleichung (5.37∗)

Wir können den Algorithmus 5.5 von Aufgabe 5.2 im wesentlichen übernehmen. Am Anfang wird nur die Matrix **A** eingelesen. Für die Dreieckszerlegung müssen bei der Elimination an Stelle der entstehenden Nullen in $a[k,i]$ die Faktoren abgespeichert werden. Dies geschieht durch die zusätzliche Anweisung

$$a[k,i] := \mathit{faktor};$$

Ferner wird bei der Elimination die rechte Seite nicht mitbehandelt, dafür merkt man sich jede Vertauschung wie in der Aufgabenstellung beschrieben. Falls der Algorithmus infolge Singularität der Matrix nicht fortgesetzt werden kann, wird nur eine partielle Dreieckszerlegung der Matrix geliefert.

Beim zweiten Schritt muss nach dem Einlesen einer neuen rechten Seite **b** diese zunächst entsprechend den erfolgten Zeilenvertauschungen permutiert werden:

$$\tilde{\mathbf{b}} = \mathbf{P}\mathbf{b}.$$

Anschliessend löst man das System mit der oberen Dreiecksmatrix **L**

$$\mathbf{L}\mathbf{y} = \tilde{\mathbf{b}}$$

durch Vorwärtseinsetzen: Die erste Gleichung enthält nur die Unbekannte y_1, es ist daher $y_1 = \tilde{b}_1/l_{11}$. Die zweite Gleichung enthält nur die Unbekannten y_1 und y_2. Da aber y_1 schon berechnet ist, kann man den Wert für y_1 einsetzen und nach y_2 auflösen u. s. w. Allgemein, mit den Bezeichnugen $\mathbf{A} := \mathbf{L}$ und $\mathbf{b} := \tilde{\mathbf{b}}$ und mit Berücksichtigung, dass $l_{ii} = 1$ ist, wird dieser Prozess durch die Gleichungen

$$y_i = b_i - \sum_{j=1}^{i-1} a_{ij} y_j, \quad i = 1, 2, \ldots, n$$

beschrieben. Im Programm wird der Vektor **x** durch den Vektor **y** überschrieben. Für den dritten Schritt benützen wir den Algorithmus 5.3, der im nachfolgenden Programm als Include File dazugeladen wird.

```
(*$B- Aufgabe 5.7*)
program gauss ;
const nn = 20;
type vektor = array [1..nn] of real;
   matrix = array [1..nn] of vektor;
var a : matrix; b, x : vektor; i, j, n : integer;
   zeile : array [1..nn] of integer; sing : boolean;

(*$I rueckwae ( Algorithmus 5.3)*)

procedure eingabe(var a : matrix);
var i,j : integer;
begin
   writeln('Matrix zeilenweise eingeben');
   for i := 1 to n do
   begin
      writeln('Zeile Nr.',i : 3);
      for j := 1 to n do read(a[i, j])
   end ;
   for i := 1 to n do zeile[i] := i;
end ;

procedure singulaer;
begin
   writeln('Matrix ist singulaer');
end ;

procedure vorwaertseinsetzen;
var i,j : integer; s : real;
begin
   for i := 1 to n do x[i] := b[zeile[i]];
   for i := 1 to n do
   begin
      s := x[i];
      for j := 1 to i - 1 do s := s - a[i, j] * x[j];
      x[i] := s
   end
end ;

procedure elimination;
var i,j,k, jmax : integer; h : vektor; faktor : real;
begin
   i := 0;
   repeat
      i := i + 1; jmax:= i;
```

```
      for j := i + 1 to n do
      if abs(a[j, i]) > abs(a[jmax, i]) then jmax := j;
      sing:= a[jmax, i] = 0;
      if sing then singulaer
      else
      begin
         (* Gleichungen tauschen *)
         if jmax <> i then
         begin
            h := a[i]; a[i] := a[jmax]; a[jmax] := h;
            (* merken, welche Gleichungen getauscht werden: *)
            j :=zeile[i]; zeile[i] := zeile[jmax]; zeile[jmax] := j
         end ;

         (* Elimination *)
         for k := i + 1 to n do
         begin
            faktor:= a[k, i]/a[i, i];
            a[k, i] := faktor; (* Faktoren fuer L abspeichern *)
            for j := i + 1 to n do
            a[k, j] := a[k, j]−faktor∗a[i, j]
         end
      end
   until (i = n) or sing
end ;

begin
writeln('Wieviele Unbekannte'); read(n);
eingabe(a);
elimination;
if sing then write('partielle ');
writeln('Dreieckszerlegung'); writeln('Matrix L:');
for i := 1 to n do
begin
   for j := 1 to i − 1 do write(a[i, j] : 8 : 4); writeln(1.0 : 8 : 4)
end ;
writeln; writeln('Matrix R:');
for i := 1 to n do
begin
   write(' ': 8 ∗ i); for j := i to n do write(a[i, j] : 8 : 4); writeln;
end ;
writeln; write('Permutationsvektor = ');
for i := 1 to n do write(zeile[i] : 3); writeln;
```

```
  if not sing then
  repeat
    writeln('neue rechte Seite b eingeben');
    for i := 1 to n do read(b[i]);
    vorwaertseinsetzen;
    rueckwaertseinsetzen(n, a, x, x);
    writeln('Loesung');
    for i := 1 to n do write(x[i]); writeln; writeln;
  until eof
end.
```

Als Beispiel betrachten wir die Matrix

$$\mathbf{A} = \begin{pmatrix} 1 & -1 & 1 & -1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \end{pmatrix} \tag{5.10}$$

Wenn wir als rechte Seiten die Einheitsvektoren $\mathbf{b} = \mathbf{e}_i$ eingeben, so erhält man jeweils eine Kolonne der inversen Matrix von $\mathbf{A}$. Das obige Programm liefert den Output:

```
Dreieckszerlegung
Matrix L:
 1.0000
 1.0000  1.0000
 1.0000  0.6667  1.0000
 1.0000  0.3333  1.0000  1.0000

Matrix R:
 1.0000  -1.0000   1.0000  -1.0000
         3.0000   3.0000   9.0000
                 -2.0000  -4.0000
                           2.0000

Permutationsvektor = 1 4 3 2
neue rechte Seite b eingeben
1 0 0 0
Loesung
   6.8212102633E-13
  -3.3333333333E-01
   5.0000000000E-01
  -1.6666666667E-01
```

```
neue rechte Seite b eingeben
0 1 0 0
Loesung
   1.0000000000E+00
 −5.0000000000E−01
 −1.0000000000E+00
   5.0000000000E−01
neue rechte Seite b eingeben
0 0 1 0
Loesung
   4.5474735089E−12
   1.0000000000E+00
   5.0000000000E−01
 −5.0000000000E−01
neue rechte Seite b eingeben
0 0 0 1
Loesung
   −2.2737367544E−13
   −1.6666666667E−01
     4.5474735089E−13
     1.6666666667E−01
```

Wir lesen daraus die inverse Matrix ab:

$$\mathbf{A}^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{2} & 1 & -\frac{1}{6} \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 \\ -\frac{1}{6} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{6} \end{pmatrix}$$

Man erhält die Matrix $\mathbf{A}$, wenn man die Koeffizienten a_i eines Polynoms 3. Grades

$$P_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

berechnen will, das durch die Punkte

$$(-1, y_0),\ (0, y_1),\ (1, y_2) \text{ und } (2, y_3)$$

verlaufen soll. Wenn statt der rechten Seite

$$\mathbf{b} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

die 4 Einheitsvektoren gewählt werden, erhält man als Lösung die Koeffizienten der *Lagrangepolynome* (s. Kap. 6). Es ist also auch

$$L_0(x) = -\frac{1}{3}x + \frac{1}{2}x^2 - \frac{1}{6}x^3$$
$$L_1(x) = 1 - \frac{1}{2}x - x^2 + \frac{1}{2}x^3$$
$$L_2(x) = x + \frac{1}{2}x^2 - \frac{1}{2}x^3$$
$$L_3(x) = -\frac{1}{6}x + \frac{1}{6}x^3$$

Aufgabe 5.8 *Man schreibe ein Programm, um numerisch stabil die Determinante einer Matrix* $\mathbf{A}$ *zu berechnen.*
Vorgehen: Man berechne die Dreieckszerlegung von $\mathbf{A}$

$$\mathbf{PA} = \mathbf{LR} \qquad (5.41*)$$

Weil die Determinante einer Dreiecksmatrix gleich dem Produkt der Diagonalelemente und die Determinante einer Permutationsmatrix ± 1 *ist, folgt aus* (5.41∗)

$$\det(\mathbf{A}) = (-1)^{Anz.\ Vertauschungen} \prod_{i=1}^{n} r_{ii}$$

Die Lösung dieser Aufgabe ist schon als Algorithmus 8.7 in 'Computermathematik' vorhanden. Hier machen wir aus diesem Algorithmus eine **function** *det*, um diese in Aufgabe 5.1 verwenden und die Resultate vergleichen zu können. Die Prozedur *elimination* aus Algorithmus 5.5 (vgl. Aufgabe 5.2) muss dazu übernommen und geändert werden. Das Produkt der Diagonalelemente der oberen Dreiecksmatrix $\mathbf{R}$ wird der Variabeln d zugewiesen. Die Initialisierung ist $d := 1$ und bei jedem Zeilentausch wird das Vorzeichen mit $d := -d$ nachgeführt. Man erhält damit:

```
function det(n : integer; a : matrix) : real;
var i,j,k,jmax: integer;h : vektor; d, faktor : real;

begin
   d := 1; i := 1;
   while (i <= n) and (d <> 0) do
   begin
      (* Pivotsuche *)
      jmax:= i;
```

```
    for j := i + 1 to n do
    if abs(a[j, i]) > abs(a[jmax, i]) then jmax:= j;
    if a[jmax, i] = 0 then d := 0
    else
    begin
      if jmax<> i then
      begin
        h := a[i]; a[i] := a[jmax]; a[jmax] := h; d := −d
      end ;
      (* Elimination *)
      d := d * a[i, i];
      for k := i + 1 to n do
      begin
        faktor:= a[k, i]/a[i, i];
        for j := i + 1 to n do
        a[k, j] := a[k, j]−faktor*a[i, j];
      end
    end ;
    i := i + 1;
  end ;
  det:= d;
end;
```

Wird diese **function** anstelle des Include-Files A5.2 in das Programm *cramer* von Aufgabe 5.1 eingesetzt, so kann man damit Gleichungssysteme lösen. Natürlich ist dies nicht eine effiziente Methode, weil die Berechnung einer Determinante asymptotisch gleichviele Rechenoperationen erfordert, wie die Lösung des Gleichungssystems. Dafür aber werden jetzt die Determinanten numerisch stabil berechnet. Zum Beispiel erhält man als Lösung des Gleichungssystems $\mathbf{H}_6^{-1}\mathbf{x} = \mathbf{e}_1$ (vgl. Aufgabe 5.1) jetzt

Determinante von A = 1.8631341116E+17 und

$$\begin{aligned} x_1 &= 9.9999973039\text{E}{-}01 \\ x_2 &= 5.0000002587\text{E}{-}01 \\ x_3 &= 3.3333334689\text{E}{-}01 \\ x_4 &= 2.5000001190\text{E}{-}01 \\ x_5 &= 2.0000001060\text{E}{-}01 \\ x_6 &= 1.6666667623\text{E}{-}01 \end{aligned}$$

Die Lösung ist so genau, wie man es von der Kondition des Systems erwarten kann.

Aufgabe 5.9 *Man zeige, dass das Produkt zweier orthogonaler Matrizen wieder eine orthogonale Matrix ergibt.*

Seien $\mathbf{P}$ und $\mathbf{Q}$ zwei orthogonale Matrizen, d. h. es gilt $\mathbf{P}^T\mathbf{P} = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}$. Dann gilt für die Produktmatrix $\mathbf{C} = \mathbf{PQ}$

$$\mathbf{C}^T\mathbf{C} = \big(\mathbf{PQ}\big)^T\mathbf{PQ} = \mathbf{Q}^T\big(\mathbf{P}^T\mathbf{P}\big)\mathbf{Q} = \mathbf{Q}^T\mathbf{I}\,\mathbf{Q} = \mathbf{I}$$

Somit ist $\mathbf{C}$ auch orthogonal.

Aufgabe 5.10 *Man schreibe ein Programm, welches die QR-Zerlegung einer Matrix* $\mathbf{A}$ *berechnet und die Matrizen* $\mathbf{Q}$ *und* $\mathbf{R}$ *ausdruckt. Hinweis: Man kann dazu die Prozedur* givenselimination *abändern, so dass die Givensrotationsmatrizen zusammenmultipliziert werden. Man kann aber auch das Programm* gauss *zusammen mit der Prozedur* givenselimination *unverändert benützen und durch geeignete Wahl der rechten Seiten* $\mathbf{b}$ *und dem Ausdrucken von* $\mathbf{b}$ *vor dem Rückwärtseinsetzen die Matrix* $\mathbf{Q}$ *kolonnenweise berechnen.*

Wir geben beide Lösungen für diese Aufgabe an. Nach Gleichung (5.54∗) berechnet man

$$\mathbf{G}^{(n-1,n)} \cdots \mathbf{G}^{(13)}\mathbf{G}^{(12)}\mathbf{A} = \mathbf{R}$$

und wenn $\mathbf{I}$ die Einheitsmatrix bezeichnet, ist

$$\mathbf{Q}^T = \mathbf{G}^{(n-1,n)} \cdots \mathbf{G}^{(13)}\mathbf{G}^{(12)}\,\mathbf{I}. \tag{5.11}$$

Die Gleichung (5.11) bedeutet, dass man $\mathbf{Q}^T$ berechnen kann, indem die einzelnen Givensmatrizen von links zuerst mit der Einheitsmatrix und danach mit der entstehenden Produktmatrix multipliziert werden. Wir ändern dazu den Algorithmus 5.6 so ab, dass zuerst die globale Matrix $qt[1..n, 1..n]$ mit der Einheitsmatrix initialisiert wird. Die Givensrotationen werden danach nicht mehr auf den Vektor b sondern auf jeden Kolonnenvektor der Matrix qt ausgeübt. Dies erfordert eine **for** -Schleife über alle Kolonnenvektoren. Die Prozedur *kontrolle* multipliziert die Matrizen $\mathbf{Q} \times \mathbf{R}$ und $\mathbf{Q}^T \times \mathbf{Q}$ und druckt die Resultate, die $\mathbf{A}$ und $\mathbf{I}$ ergeben müssen. Man erhält damit

```
(*$B- Aufgabe 5.10A*)
program qr ;
const nn = 20;
type vektor = array [1..nn] of real;
   matrix = array [1..nn] of vektor;
var qt,a : matrix; i,j,k,n : integer; s : real;

procedure eingabe(var a : matrix);
```

```
var i,j : integer;
begin
    writeln('Matrix A zeilenweise eingeben');
    for i := 1 to n do
    begin
        writeln('Zeile Nr.',i : 3);
        for j := 1 to n do read(a[i, j])
    end
end ;

procedure givenselimination;
(* Die Givensmatrizen werden zusammenmultipliziert und ergeben Q^T*)
var i,j,k : integer; cot,co,si,h : real;
begin
    (* Initialisierung von Q^T = I*)
    for i := 1 to n do for j := 1 to n do
    if i = j then qt[i, i] := 1 else qt[i, j] := 0;

    for i := 1 to n do
    begin
        for k := i + 1 to n do
        if a[k, i] <> 0 then
        begin
            cot:= a[i, i]/a[k, i];
            si:= 1/sqrt(1+cot*cot); co:= si* cot;
            a[i, i] := a[i, i] * co + a[k, i] * si;
            for j := i + 1 to n do
            begin
                h := a[i, j] * co + a[k, j] * si;
                a[k, j] := −a[i, j] * si + a[k, j] * co;
                a[i, j] := h
            end ;
            (* Q^T := G*Q^T *)
            for j := 1 to n do
            begin
                h := qt[i, j] * co + qt[k, j] * si;
                qt[k, j] := −qt[i, j] * si + qt[k, j] * co;
                qt[i, j] := h
            end
        end
    end
end ;

procedure kontrolle;
```

```
begin
   for i := 2 to n do
   for j := 1 to i − 1 do a[i, j] := 0;
   writeln('Kontrolle Q∗R=');
   for i := 1 to n do
   begin
      for j := 1 to n do
      begin
         s := 0;
         for k := 1 to n do s := s + qt[k, i] ∗ a[k, j];
         write(s : 10 : 5);
      end ;
      writeln;
   end ;
   writeln('Q^T∗Q =');
   for i := 1 to n do
   begin
      for j := 1 to n do
      begin
         s := 0;
         for k := 1 to n do s := s + qt[i, k] ∗ qt[j, k];
         write(s : 10 : 5);
      end ;
      writeln;
   end
end ;
```

```
begin
   writeln('Ordnung der Matrix'); read(n);
   eingabe(a);
   givenselimination;
   writeln('Matrix R');
   for i := 1 to n do
   begin
      for j := 1 to i − 1 do write(' ': 10);
      for j := i to n do write(a[i, j] : 10 : 5);
      writeln
   end ;
   writeln('Matrix Q');
   for i := 1 to n do
   begin
      for j := 1 to n do write(qt[j, i] : 10 : 5);
      writeln
```

end *;*
kontrolle;
end.

Als Beispiel berechnen wir die QR-Zerlegung der Matrix (5.10). Man erhält den Output:

```
Matrix R
   2.00000     1.00000     3.00000     4.00000
               2.23607     2.23607     6.26099
                           2.00000     3.00000
                                       1.34164
 Matrix Q
   0.50000    −0.67082     0.50000    −0.22361
   0.50000    −0.22361    −0.50000     0.67082
   0.50000     0.22361    −0.50000    −0.67082
   0.50000     0.67082     0.50000     0.22361
Kontrolle Q∗R=
   1.00000    −1.00000     1.00000    −1.00000
   1.00000     0.00000    −0.00000     0.00000
   1.00000     1.00000     1.00000     1.00000
   1.00000     2.00000     4.00000     8.00000
Q^T∗Q =
   1.00000     0.00000     0.00000     0.00000
   0.00000     1.00000     0.00000     0.00000
   0.00000     0.00000     1.00000    −0.00000
   0.00000     0.00000    −0.00000     1.00000
```

Die zweite Lösung ergibt sich durch folgende Überlegungen: Löst man das Gleichungssystem

$$\mathbf{Ax} = \mathbf{b} \quad \text{mit } \mathbf{b} = \mathbf{e}_i \quad \text{Einheitsvektor,}$$

so ist nach der Givenselimination vor dem Rückwärtseinsetzen

$$\mathbf{Rx} = \mathbf{y} = \mathbf{G}^{(n-1,n)} \cdots \mathbf{G}^{(13)}\mathbf{G}^{(12)}\mathbf{e}_i = \mathbf{Q}^T\mathbf{e}_i = \mathbf{c}\mathbf{Q}_i^T = \mathbf{r}\mathbf{Q}_i.$$

Wenn wir also n Gleichungssysteme mit den rechten Seiten

$$\mathbf{b} = \mathbf{e}_i, \quad i = 1, \ldots, n$$

lösen, erhalten wir als Zwischenvektor $\mathbf{y}$ jeweils *eine Zeile von* $\mathbf{Q}$. Natürlich ist diese Lösung nicht sehr wirtschaftlich, weil für jede neue rechte Seite dieselben Eliminationsschritte wieder durchgeführt werden. Die erste Lösung, Aufgabe 5.10A, ist daher besser. Damit die Matrix nicht immer wieder eingegeben werden muss, speichern wir diese zusätzlich im **array** *aa* ab.

```
(*$B- Aufgabe 5.10B*)
program qr ;
const nn = 20;
type vektor = array [1..nn] of real;
   matrix = array [1..nn] of vektor;
var a,aa : matrix; b,x : vektor; i,j,n : integer;

procedure eingabe(var a : matrix);
var i,j : integer;
begin
   writeln('Matrix zeilenweise eingeben');
   for i := 1 to n do
   begin
      writeln('Zeile Nr.',i : 3);
      for j := 1 to n do read(a[i, j]);
   end
end ;

procedure singulaer;
begin
   writeln('Matrix ist singulaer');
end ;

procedure givenselimination;
var i,j,k : integer; cot,co,si,h : real;
begin
   for i := 1 to n do
   begin
   for k := i + 1 to n do
   if a[k, i] <> 0 then
   begin
      cot:= a[i, i]/a[k, i];
      si:= 1/sqrt(1+cot*cot); co := si*cot;
      a[i, i] := a[i, i] * co + a[k, i] * si;
      for j := i + 1 to n do
      begin
         h := a[i, j] * co + a[k, j] * si;
         a[k, j] := -a[i, j] * si + a[k, j] * co;
         a[i, j] := h
      end ;
      h := b[i] * co + b[k] * si;
      b[k] := -b[i] * si + b[k] * co;
      b[i] := h
end ;
```

```
        if a[i,i] = 0 then singulaer
    end
end ;
begin
    writeln('Ordnung der Matrix'); read(n);
    eingabe(aa);
    writeln('Matrix Q =');
    for i := 1 to n do
    begin
        a := aa;
        (* b = i-ter Einheitsvektor *)
        for j := 1 to n do b[j] := 0; b[i] := 1;
        givenselimination;
        (* b enthaelt jetzt die i-te Kolonnen von Q^T
           d.h. die i-te Zeile von Q *)
        for j := 1 to n do write(b[j] : 10 : 5); writeln
    end ;
    writeln('Matrix R');
    for i := 1 to n do
    begin
        for j := 1 to i - 1 do write(' ': 10);
        for j := i to n do write(a[i,j] : 10 : 5);
        writeln
    end
end.
```

Aufgabe 5.11 *Man zeige, dass die lineare Abbildung* $\mathbf{y}$=$\mathbf{Qx}$ *mit der orthogonalen Matrix* $\mathbf{Q}$ *längentreu ist.*

$\mathbf{Q}$ ist orthogonal, also gilt $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$. Es ist

$$||\mathbf{y}||^2 = \mathbf{y}^T\mathbf{y} = (\mathbf{Qx})^T\mathbf{Qx} = \mathbf{x}^T\mathbf{Q}^T\mathbf{Qx} = \mathbf{x}^T\mathbf{x} = ||\mathbf{x}||^2,$$

was zu zeigen war.

Aufgabe 5.12 *Man setze die Prozedur givenselimination in das Programm gauss anstelle der Prozedur elimination ein und ändere sie so ab, dass überbestimmte Gleichungssysteme gelöst werden können.*

Wir bezeichnen mit n die Anzahl der Unbekannten und mit m die Anzahl der Gleichungen. In Algorithmus 5.6 *givenselimination* muss nur die obere Grenze der k-Schleife geändert werden:

for $k := i + 1$ **to** m **do**

Ebenso ändert die obere Grenze der i-Schleife bei der Prozedur *eingabe* und man muss neben der Anzahl n der Unbekannten auch die Anzahl m der Gleichungen einlesen:

```
(*$B- Aufgabe 5.12*)
program ausgleich ;
const nn = 20;
type vektor = array [1..nn] of real;
   matrix = array [1..nn] of vektor;
var a : matrix; b,x : vektor; i,n,m : integer;
      (* n=Anzahl Unbekannte, m = Anzahl Gleichungen *)

(*$I rueckwae ( Algorithmus 5.3)*)

procedure eingabe(var a : matrix; var b : vektor);
var i,j : integer;
begin
   writeln('eine Gleichung nach der andern eingeben');
   for i := 1 to m do
   begin
      writeln('Gleichung Nr.',i : 3);
      for j := 1 to n do read(a[i, j]); readln(b[i]);
   end
end ;

procedure singulaer;
begin
   writeln('Matrix ist singulaer');
end ;
```

```
procedure givenselimination;
var i,j,k : integer; cot,co,si,h : real;
begin
   for i := 1 to n do
   begin
      for k := i + 1 to m do
      if a[k, i] <> 0 then
      begin
         cot:= a[i, i]/a[k, i];
         si:= 1/sqrt(1+cot*cot); co := si * cot;
         a[i, i] := a[i, i] * co + a[k, i] * si;
         for j := i + 1 to n do
         begin
            h := a[i, j] * co + a[k, j] * si;
            a[k, j] := −a[i, j] * si + a[k, j] * co;
```

```
            a[i, j] := h
        end ;
        h := b[i] ∗ co + b[k] ∗ si;
        b[k] := −b[i] ∗ si + b[k] ∗ co;
        b[i] := h
    end ;
    if a[i, i] = 0 then singulaer
  end
end ;

begin
  writeln('Wieviele Unbekannte ?'); read(n);
  writeln('Wieviele Gleichungen ?'); read(m);
  eingabe(a, b);
  givenselimination;
  rueckwaertseinsetzen(n, a, b, x);
  writeln('Loesung x =');
  for i := 1 to n do writeln(x[i])
end.
```

Als Beispiel lösen wir das Gleichungssystem (5.59∗). Man erhält den folgenden Output auf dem Bildschirm

```
Wieviele Unbekannte ?
3
Wieviele Gleichungen ?
4
eine Gleichung nach der andern eingeben
Gleichung Nr. 1
1 1 1 5
Gleichung Nr. 2
4 2 1 7
Gleichung Nr. 3
9 3 1 5
Gleichung Nr. 4
16 4 1 1
Loesung x =
−1.5000000000E+00
 6.1000000000E+00
 5.0000000001E−01
```

Aufgabe 5.13 *Die Givenselimination für Taschenrechner.*

Man schreibe ein Programm für die zeilenweise Elimination mittels Givensrotationen. Das PASCAL Programm kann dann leicht für Taschenrechner übersetzt werden (Siehe Kap.8).

Taschenrechner haben meistens kleine Speicher. Man kann die Elimination so anordnen, dass nicht alle Gleichungen am Anfang in den Speicher eingelesen werden müssen. Man kann dabei eine Gleichung nach der anderen verarbeiten, d.h. soviele Unbekannte eliminieren als schon Gleichungen eingelesen wurden.Man speichert lediglich die Rechtsdreiecksmatix $\mathbf{R}$ *und die rechte Seite* $\mathbf{c}$ *ab. Um wirklich nur die Elemente der Rechtsdreiecksmatrix zu speichern, kann durch die Vorschrift*

$$r_{ij} = v[i + j * (j-1)/2] \tag{5.12}$$

die Matrix $\mathbf{R}$ *auf den Vektor* $\mathbf{v}$ *abgebildet werden. Damit werden die Nullen von* $\mathbf{R}$ *nicht gespeichert. Bei Ausgleichsproblemen kann nach jeder neuen Gleichung eine Zwischenlösung berechnet werden und auch, falls gewünscht, das neue* $\|\mathbf{r}\|^2$ *nachgeführt werden.*

Wir zeigen die Idee der zeilenweisen Elimination für $n = 4$*. Wir nehmen an, wir hätten schon 2 Gleichungen eingelesen und verarbeitet, d.h.* x_1 *wurde in der 2. Gleichung schon eliminiert. Wenn wir die rechte Seite als* $n+1$*-ste Kolonne auch in der Matrix* $\mathbf{R}$ *abspeichern, so haben wir bisher die Matrix*

$$\mathbf{R} = \left(\begin{array}{cccc|c} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} \\ 0 & r_{22} & r_{23} & r_{24} & r_{25} \end{array}\right) \tag{5.68*}$$

Nun lesen wir die nächste Gleichung ein, deren Koeffizienten wir mit d_i *bezeichnen:*

$$d_1x_1 + d_2x_2 + d_3x_3 + d_4x_4 = d_5.$$

Fügen wir sie zum System (5.68∗) *hinzu, so erhalten wir die neue Matrix:*

$$\mathbf{A} = \left(\begin{array}{cccc|c} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} \\ 0 & r_{22} & r_{23} & r_{24} & r_{25} \\ d_1 & d_2 & d_3 & d_4 & d_5 \end{array}\right) \tag{5.69*}$$

Wir wollen nun in der dritten Zeile die Unbekannten x_1 *und* x_2 *eliminieren. Dazu wählen wir eine Matrix* $\mathbf{G}^{(13)}$ *so, dass bei der Multiplikation mit* (5.69∗) *von links* $d_1 = 0$ *wird. Dabei ändert in* (5.69∗) *nur die erste und die letzte Zeile. Wenn wir der Einfachheit halber die neuen Koeffizienten wieder mit* r_{ik} *und* d_i *bezeichnen, erhält man*

$$\mathbf{G}^{(13)}\mathbf{A} = \left(\begin{array}{cccc|c} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} \\ 0 & r_{22} & r_{23} & r_{24} & r_{25} \\ 0 & d_2 & d_3 & d_4 & d_5 \end{array}\right) \tag{5.70*}$$

Bei der nächsten Multiplikation mit $\mathbf{G}^{(23)}$ *wird* $d_2 = 0$ *und man erhält*

$$\mathbf{G}^{(23)}\mathbf{G}^{(13)}\mathbf{A} = \left(\begin{array}{cccc|c} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} \\ 0 & r_{22} & r_{23} & r_{24} & r_{25} \\ 0 & 0 & d_3 & d_4 & d_5 \end{array}\right) = \mathbf{R}^{neu} \qquad (5.71*)$$

Wenn noch nicht n *Gleichungen eingelesen worden sind, ergibt die neue Gleichung nach der Elimination eine neue Zeile in der Matrix* $\mathbf{R}$*. Wenn wir aber ein Ausgleichsproblem haben und die* k*-te Gleichung mit* $k > n$ *einlesen, so können wir alle* n *Unbekannten davon eliminieren und die bestehende Dreiecksmatrix* $\mathbf{R}$ *wird nur nachgeführt. Einzig das Element auf der rechten Seite* d_{n+1} *bleibt übrig. Ist das System kompatibel, so muss* $d_{n+1} = 0$ *werden. Ist* $d_{n+1} \neq 0$*, so ist es eine Komponente, die zur Länge des Residuenvektors gehört, und man kann die Residuenquadratsumme wegen* (5.66∗) *durch*

$$residuum := residuum + sqr(d[n+1])$$

nachführen.

Bezeichnet k *die Nummer der Gleichung, die gerade eingelesen wurde, und ist* $min = \min\{n, k-1\}$*, so kann die Verarbeitung der* k*-ten Gleichung wie folgt symbolisch dargestellt werden:*

for $i := 1$ **to** *min* **do**
begin

$$\begin{pmatrix} r_{ii} & r_{i,i+1} & \dots & r_{in} \\ 0 & d_{i+1} & \dots & c_n \end{pmatrix} := \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} r_{ii} & \cdots & r_{in} \\ d_i & \cdots & d_n \end{pmatrix}$$

end

Die Winkelfunktionen $\cos\alpha$ *und* $\sin\alpha$ *werden dabei wie bei* (5.46∗) *so gewählt, dass*

$$\cos\alpha \cdot r_{ii} - \sin\alpha \cdot d_i = 0.$$

Zunächst formen wir den für überbestimmte Gleichungssysteme abgeänderten Algorithmus 5.6 (s. Aufgabe 5.12) so um, dass die Unbekannten *zeilenweise* eliminiert werden. Dazu muss man die k– und die i–Schleife vertauschen. Die obere Grenze der i-Schleife ist *min*, wie in der Aufgabenstellung beschrieben. Den Test, ob ein Diagonalelement null wird lassen wir weg; wir werden dies beim Rückwärtseinsetzen prüfen. Man erhält damit:

```
procedure givenselimination;
var i,j,k,min : integer; cot,co,si,h : real;
begin
   for k := 1 to m do
   begin
      if k − 1 < n then min:= k − 1 else min:= n;
      for i := 1 to min do
      if a[k, i] <> 0 then
      begin
         cot:= a[i, i]/a[k, i];
         si:= 1/sqrt(1+cot∗cot); co := si∗cot;
         a[i, i] := a[i, i] ∗ co + a[k, i] ∗ si;
         for j := i + 1 to n do
         begin
            h := a[i, j] ∗ co + a[k, j] ∗ si;
            a[k, j] := −a[i, j] ∗ si + a[k, j] ∗ co;
            a[i, j] := h;
         end ;
         h := b[i] ∗ co + b[k] ∗ si;
         b[k] := −b[i] ∗ si + b[k] ∗ co;
         b[i] := h;
      end
   end
end ;
```

Es ist nun nicht mehr nötig, alle Gleichungen am Anfang einzulesen. Man kann bei jeder Erhöhung von k eine weitere Gleichung einlesen. Daher ist es sinnvoll, die **for** -Schleife durch eine **repeat** -Schleife zu ersetzen, welche aufhört, wenn keine Gleichungen mehr vorhanden sind:

```
k := 0;
repeat
   k := k + 1;
   writeln('Gleichung Nr.',k : 3);
   for j := 1 to n do read(a[k, j]); read(b[k]);
   if k − 1 < n then min:= k − 1 else min:= n;
   .........
until eof;
```

Die Prozedur *eingabe* entfällt und ebenso muss die Anzahl Gleichungen im Hauptprogramm nicht mehr eingelesen werden. Als weitere Vereinfachung im Hinblick auf die Implementation auf einen Taschenrechner ersetzen wir den Aufruf von *givenselimination* durch den Prozedurkörper und ebenso eliminieren wir den Aufruf der Prozedur *rueckwaertseinsetzen*. Für $k \geq n$

berechnen wir jedesmal eine neue Lösung, indem das Rückwärtseinsetzen in die k-Schleife genommen wird. Die rechte Seite **b** wird als $n+1$-ste Kolonne der Matrix **A** angefügt. Um die Givensrotationen darauf auszuüben, muss die j-Schleife bis $n+1$ gehen.

```
(*$B- *)
program zeilengivens ;
const nn = 20;
type vektor = array [1..nn] of real;
   matrix = array [1..nn] of vektor;
var a : matrix; b,x : vektor;
   n,i,j,k,min : integer; cot,co,si,h : real;

begin
   writeln('Wieviele Unbekannte ?'); read(n);
   k := 0;
   repeat
      k := k + 1;
      writeln('Gleichung Nr.',k : 3);
      for j := 1 to n + 1 do read(a[k, j]);
      if k − 1 < n then min:= k − 1 else min:= n;
      for i := 1 to min do
      if a[k, i] <> 0 then
      begin
         cot:= a[i, i]/a[k, i];
         si:= 1/sqrt(1+cot∗cot); co := si∗cot;
         a[i, i] := a[i, i] ∗ co + a[k, i] ∗ si;
         for j := i + 1 to n + 1 do
      begin
         h := a[i, j] ∗ co + a[k, j] ∗ si;
         a[k, j] := −a[i, j] ∗ si + a[k, j] ∗ co;
         a[i, j] := h;
      end
end ;

if k >= n then
begin
   (∗ Rueckwaertseinsetzen ∗)
   for i := n downto 1 do
   begin
      h := a[i, n + 1];
      for j := i + 1 to n do h := h − a[i, j] ∗ x[j];
      if a[i, i] <> 0 then x[i] := h/a[i, i]
      else writeln('singulaer');
```

```
        end ;
        writeln('Loesung x =');
        for i := 1 to n do writeln(x[i])
      end ;
  until eof;
end.
```

Nun muss nur noch die Abbildung der Matrix **A** auf den Vektor **v** nach Gleichung (5.12) durchgeführt werden, damit nur die Elemente der oberen Dreiecksmatrix gespeichert werden. Es treten komplizierte Indices in **v** auf, welche wenn möglich nur einmal berechnet und in der Variabeln *ind* abgespeichert werden.

Die neue Gleichung wird im Vektor **d** gespeichert, wie in der Aufgabenstellung vorgeschlagen. Falls ein Diagonalelement von **R** null ist, wird im Sinne von Aufgabe 5.6 eine grosse Lösung erzeugt, welche die lineare Abhängigkeit der Kolonnenvektoren aufzeigt. Als Kontrolle dazu werden die Diagonalelemente r_{ii} neben der Lösung ausgedruckt. Ferner wird das Residuenquadrat in der Variabeln *residuum* bei der Verarbeitung einer neuen Gleichung nachgeführt. Man erhält damit folgendes Programm, das nun leicht für einen Taschenrechner verwendet werden kann:

```
(*$B- Aufgabe 5.13*)
program zeilengivens;
const nd = 20; (*n < nd*)
var v : array [1..200] of real; d,x : array [1..nd] of real;
   n,i,j,k,min,ind : integer; co,si,h,residuum : real ;

begin
writeln('Anzahl Unbekannte ?'); read(n);
k := 0; residuum:= 0;
repeat
   k := k + 1;
   writeln('naechste Gleichung eingeben');
   for j := 1 to n + 1 do read(d[j]);
   if k <= n then min:= k - 1 else min:= n;
   for i := 1 to min do
   begin
      if d[i] <> 0 then
      begin
         ind:= i * (i + 1) div 2;
         h := v[ind]/d[i]; si := 1/sqrt(1 + h * h); co := si * h;
         v[ind] := v[ind] * co + d[i] * si;
         for j := i + 1 to n + 1 do
         begin
```

```
            ind:= i + j * (j - 1) div 2;
            h := v[ind] * co + d[j] * si;
            d[j] := -v[ind] * si + d[j] * co; v[ind] := h;
          end
        end
      end ;
      if k <= n then
        for j := k to n + 1 do v[k + j * (j - 1) div 2] := d[j]
      else residuum:= residuum + sqr(d[n + 1]);

      if k >= n then
      begin
        (* Rueckwaetrseinsetzen *)
        for i := n downto 1 do
        begin
          h := v[i + (n + 1) * n div 2];
          for j := i + 1 to n do
            h := h - v[i + j * (j - 1) div 2] * x[j];
          ind:= i * (i + 1) div 2;
          if v[ind] <> 0 then x[i] := h/v[ind]
          else begin writeln('singulaer'); x[i] := h * 1e11 end ;
        end ;
        writeln('Loesung x, Diagonalelemente von R: r[i,i] ');
        for i := 1 to n
          do writeln(x[i] : 12 : 8, v[i * (i + 1) div 2] : 20);
        writeln('Residuenquadrat ',residuum);
      end ;
    until eof
  end.
```

Als Beispiel berechnen wir mit diesem Programm nochmals die Lösung von (5.59∗) und erhalten den Output:

```
Anzahl Unbekannte ?
3
naechste Gleichung eingeben
1 1 1 5
naechste Gleichung eingeben
4 2 1 7
naechste Gleichung eingeben
9 3 1 5
Loesung x, Diagonalelemente von R: r[i,i]
−2.00000000    9.8994949366E+00
 8.00000000 −8.8063057185E−01
```

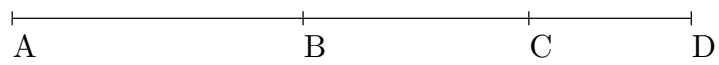
```
−1.00000000   2.2941573387E−01
Residuenquadrat    0.0000000000E+00
naechste Gleichung eingeben
16 4 1 1
Loesung x, Diagonalelemente von R: r[i,i]
−1.50000000   1.8814887722E+01
  6.10000000 −1.3234093960E+00
  0.50000000   3.5921060405E−01
Residuenquadrat    1.9999999999E−01
```

Wir sehen, dass wir nach dem Einlesen der ersten drei Gleichungen als Zwischenlösung die Koeffizienten der Parabel erhalten, welche durch die ersten drei Punkte verläuft.

Aufgabe 5.14 *Von einem Strassenstück (siehe Figur 5.1) werden die folgenden Teilstücke gemessen:*

$$AD = 89m,\ AC = 67m,\ BD = 53m,\ AB = 35m \text{ und } CD = 20m$$

Wie gross sind die ausgeglichenen Strecken AB, BC und CD ?



Figur 5.1: Strassenstück

Wir setzen

$$x_1 = AB,\ x_2 = BC \text{ und } x_3 = CD$$

Die angegebenen Messungen lauten dann

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & + & x_3 & = & 89 \\ x_1 & + & x_2 & & & = & 67 \\ & & x_2 & + & x_3 & = & 53 \\ x_1 & & & & & = & 35 \\ & & & & x_3 & = & 20 \end{array}$$

Die Auflösung dieses überbestimmten linearen Gleichungssystems mittels des Givensverfahrens (s. Aufgabe 5.12 oder 5.13) liefert die Lösung:

$$x_1 = 35.125,\ x_2 = 32.5 \text{ und } x_3 = 20.625$$

Aufgabe 5.15 *Durch die Punkte*

x	2	4	6	8
y	0.350	0.573	0.725	0.947

lege man eine Regressionsgerade der Gestalt $y = ax + b$*, so dass*

$$\sum_{i=1}^{4}(ax_i + b - y_i)^2 = min.$$

Für die beiden Unbekannten a und b erhält man nach der Methode der kleinsten Quadrate $\big($s. Gleichung $(5.63*)$ $\big)$ das Gleichungssystem

$$\begin{pmatrix} 2 & 1 \\ 4 & 1 \\ 6 & 1 \\ 8 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 0.350 \\ 0.573 \\ 0.725 \\ 0.947 \end{pmatrix}$$

welches mit dem Givensverfahren gelöst, die Werte $a = 0.09715$ und $b = 0.163$ liefert.

Aufgabe 5.16 *Man lege eine Funktion der Gestalt*

$$y = ae^{bx} \qquad (5.72*)$$

durch die Punkte

x	30.0	64.5	74.5	86.7	94.5	98.9
y	4	18	29	51	73	90

Durch Logarithmieren der Gleichung $(5.72*)$ *erhält man*

$$\ln y = \ln a + bx,$$

und wenn man statt a *die Unbekannte* $c = \ln a$ *einführt, so kann man wie bei Aufgabe 5.15 vorgehen.*

Für die Unbekannten c und b erhalten wir hier

$$\begin{pmatrix} 1 & 30.0 \\ 1 & 64.5 \\ 1 & 74.5 \\ 1 & 86.7 \\ 1 & 94.5 \\ 1 & 98.9 \end{pmatrix} \begin{pmatrix} c \\ b \end{pmatrix} = \begin{pmatrix} \ln 4 \\ \ln 18 \\ \ln 29 \\ \ln 51 \\ \ln 73 \\ \ln 90 \end{pmatrix}$$

Der Givensalgorithmus liefert

$$b = 0.04524310648 \text{ und } c = 0.00789262406 \Rightarrow a = 1.00792752.$$

Aufgabe 5.17 *Durch die Punkte*

x	5	6	7	9
y	11	9	8	7

lege man eine Funktion der Gestalt

$$y = 1 + \frac{a}{x+b}. \tag{5.73$*$}$$

Durch Umformung der Gleichung (5.73$*$) *führe man wiederum die Aufgabe auf ein lineares Ausgleichsproblem zurück.*

Es gibt verschiedene Möglichkeiten, die Gleichung (5.73$*$) umzuformen. Multipliziert man mit dem Nenner $x+b$ und schafft die unbekannten Parameter a und b auf die linke Seite, so ergibt sich

$$a + (1-y)b = xy - x. \tag{5.13}$$

Setzt man nun die gegebenen Punkte in die Gleichung (5.13) ein, so erhält man das Gleichungssystem

$$\begin{pmatrix} 1 & -10 \\ 1 & -8 \\ 1 & -7 \\ 1 & -6 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 50 \\ 48 \\ 49 \\ 54 \end{pmatrix}$$

welches, gelöst mit dem Givensverfahren, die Parameter $a = 56.228571428$ und $b = 0.77142857142$ liefert.

Eine andere Umformung von Gleichung (5.73$*$) ist

$$\frac{1}{y-1} = \frac{1}{a}x + \frac{b}{a}. \tag{5.14}$$

Führt man als neue Unbekannte

$$\alpha = \frac{1}{a} \quad \text{und} \quad \beta = \frac{b}{a}$$

ein, so erhält man das Gleichungssystem:

$$\begin{pmatrix} 5 & 1 \\ 6 & 1 \\ 7 & 1 \\ 9 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 1/10 \\ 1/8 \\ 1/7 \\ 1/6 \end{pmatrix}$$

Diesmal erhält man

$$\alpha = 0.016224489797 = \frac{1}{a} \quad \Rightarrow a = 61.63522011$$

$$\beta = 0.024115646262 = \frac{b}{a} \quad \text{d.h.} \quad b = a\beta = 1.486373166$$

Man kann nicht erwarten, dass die beiden Umformungen gleiche Resultate ergeben, da man mit verschiedenen Massen minimiert.

Aufgabe 5.18 *Man lege eine Funktion der Form*

$$y = at + \frac{b}{t} + c\sqrt{t}$$

durch die Punkte

t	1	2	3	4	5
y	2.1	1.6	1.9	2.5	3.1

Man erhält hier das folgende Gleichungssystem für die Unbekannten a, b und c:

$$\begin{aligned} 1a + b/1 + c\sqrt{1} &= 2.1 \\ 2a + b/2 + c\sqrt{2} &= 1.6 \\ 3a + b/3 + c\sqrt{3} &= 1.9 \\ 4a + b/4 + c\sqrt{4} &= 2.5 \\ 5a + b/5 + c\sqrt{5} &= 3.1 \end{aligned}$$

Die Auflösung ergibt

$$\begin{aligned} x_1 &= \quad 1.003967387 \\ x_2 &= \quad 2.13669432 \\ x_3 &= -1.042430842 \end{aligned}$$

Aufgabe 5.19 *Man bestimme Radius r und Mittelpunkt M des Kreises, der möglichst gut durch die Punkte*

x	0.7	3.3	5.6	7.5	6.4	4.4	0.3	-1.1
y	4.0	4.7	4.0	1.3	-1.1	-3.0	-2.5	1.3

geht. Vorgehen: Mit $M = (m_1, m_2)$ lautet die Kreisgleichung:

$$(x - m_1)^2 + (y - m_2)^2 = r^2.$$

Wenn wir ausmultiplizieren und umordnen, erhalten wir

$$2xm_1 + 2ym_2 + (r^2 - m_1^2 - m_2^2) = x^2 + y^2. \qquad (5.74*)$$

Wenn wir die neue Unbekannte $c = r^2 - m_1^2 - m_2^2$ einführen, ist (5.74$*$) eine lineare Gleichung in den drei Unbekannten m_1, m_2 und c und führt auf ein lineares Ausgleichsproblem.

Das Gleichungssystem lautet hier

$$\begin{aligned}
1.4\,m_1 + \quad 8\,m_2 + c &= 16.49\\
6.6\,m_1 + 9.4\,m_2 + c &= 32.98\\
11.2\,m_1 + \quad 8\,m_2 + c &= 47.36\\
15\,m_1 + 2.6\,m_2 + c &= 57.94\\
12.8\,m_1 - 2.2\,m_2 + c &= 42.17\\
8.8\,m_1 - \quad 6\,m_2 + c &= 28.36\\
0.6\,m_1 - \quad 5\,m_2 + c &= \quad 6.34\\
-2.2\,m_1 + 2.6\,m_2 + c &= \quad 2.90
\end{aligned}$$

Mit dem Givensverfahren erhält man die Lösung

$$\begin{aligned}
m_1 &= 3.0603035657\\
m_2 &= 0.74360732104\\
c &= 6.9665974189 \quad \Rightarrow r = \sqrt{c + m_1^2 + m_2^2} = 4.109137036
\end{aligned}$$

Kapitel 6

Interpolation

Aufgabe 6.1 *Von der Funktion $y = f(x) = 2^x$ seien die drei Stützpunkte*

x	-1	1	3
y	0.5	2	8

gegeben. Wie gross ist der Interpolationsfehler im Intervall $[1, 3]$, wenn diese Punkte durch ein Polynom interpoliert werden? Man schätze den Fehler mittels (6.7∗) und berechne anschliessend das Polynom und den exakten Fehler.

Für die Fehlerabschätzung wird die 3. Ableitung von f benötigt:

$$f(x) = 2^x = e^{x \ln 2} \quad \Rightarrow \quad f'''(x) = \ln^3 2 \, e^{x \ln 2} = 2^x \ln^3 2$$

Im Intervall $[1, 3]$ wird $f(\xi)$ am grössten für $\xi = 3$. Damit ist

$$|f(x) - P_2(x)| \leq \frac{1}{3!} |\underbrace{(x+1)(x-1)(x-3)}_{g(x)}| 2^3 \ln^3 2. \tag{6.1}$$

Da die Abschätzung für alle $x \in [1, 3]$ gelten soll, muss noch das Maximum der Funktion g bestimmt werden. Es ist

$$\begin{aligned} g(x) &= (x^2 - 1)(x - 3) \\ g'(x) &= 2x(x - 3) + (x^2 - 1) = 3x^2 - 6x - 1 = 0 \\ \Rightarrow x_{1,2} &= 1 \pm 2/\sqrt{3} \quad \Rightarrow \max |g(x_1)| = 3.079201436 \end{aligned}$$

Setzt man diesen Wert in die Gleichung (6.1) ein, so ergibt sich

$$|f(x) - P_2(x)| \leq 1.36726649. \tag{6.2}$$

Nach der Lagrangeinterpolationsformel (6.5∗) lautet das Interpolationspolynom hier

$$P_2(x) = (x-1)(x-3)\frac{1}{16} - (x+1)(x-3)\frac{1}{2} + (x+1)(x-1).$$

Der exakte Interpolationsfehler ist

$$d(x) = 2^x - P_2(x)$$

und für die Abschätzung muss das Maximum der Funktion d im Intervall $[1, 3]$ bestimmt werden. Es ist

$$\begin{aligned} d'(x) &= 2^x \ln 2 - (x - 1 + x - 3)\frac{1}{16} + (x + 1 + x - 3)\frac{1}{2} - (x + 1 + x - 1) \\ &= 2^x \ln 2 - 1.125x - 0.75 = 0. \end{aligned} \tag{6.3}$$

Mit dem Bisektionsalgorithmus (s. Aufgabe 3.2) und den Startwerten $a = 1.1$ und $b = 2.9$ erhält man die Nullstelle $x = 2.2361397274$ von $d'(x)$ und damit die Schranke

$$|d(x)| \leq 0.4659378527.$$

Aufgabe 6.2 *Von der nicht einfach zu berechnenden Funktion*

$$C(x) = -\int_x^\infty \frac{\cos t}{t} dt$$

kennt man die Funktionswerte

x	5.0	5.2	5.5	5.6
y	-0.19002	-0.17525	-0.14205	-0.12867

Die Tabellenwerte sind korrekt gerundet. Man interpoliere den Wert $C(5.3)$ und schätze den Interpolationsfehler ab.

Wir wenden für die Interpolation die Lagrangeformel an. Algorithmus 6.1 besteht bloss aus der **function** p, wir schreiben dazu ein Hauptprogramm, das zuerst die Stützpunkte einliest und darauf für jeweils eine Neustelle z den interpolierten Wert $P_n(z)$ berechnet und ausdruckt. Die **function** p wird als Include-File dazugeladen.

```
(*$B- A6.1 *)
program lagran;
type vektor = array [0..20] of real;
var n,i,k : integer; z : real; x,y : vektor;
(*$I p Algorithmus 6.1*)
begin
   writeln('Punkte x,y eingeben mit CTRL-Z abbrechen');
   n := -1;
   while not eof do
   begin
      n := n + 1; read(x[n], y[n])
   end ;
   reset(input);
   writeln; writeln('eingelesene Stuetzpunkte:');
   for i := 0 to n do writeln(x[i] : 10 : 6, y[i] : 10 : 6);
   writeln('Neustellen eingeben mit CTRL-Z aufhoeren');
   while not eof do
   begin
      read(z); writeln('z=',z : 10 : 3,' P(z) =', p(n, x, y, z))
   end
end.
```

Für die Neustelle $z = 5.3$ und den 4 angegebenen Stützpunkten von $C(x)$ erhält man mit diesem Programm den interpolierten Wert:

$$P_3(5.3) = -0.165498. \tag{6.4}$$

Um die Genauigkeit dieses Wertes zu beurteilen, benützen wir die Fehlerabschätzung (6.7∗). Es ist

$$\begin{aligned}
C'(x) &= \frac{\cos x}{x} \\
C''(x) &= -\frac{1}{x}\sin x - \frac{1}{x^2}\cos x \\
C'''(x) &= \frac{2}{x^2}\sin x + \left(\frac{2}{x^3} - \frac{1}{x}\right)\cos x \\
C''''(x) &= \left(\frac{1}{x} - \frac{6}{x^3}\right)\sin x + \left(\frac{3}{x^2} - \frac{6}{x^4}\right)\cos x
\end{aligned}$$

Durch eine Wertetabelle überzeugt man sich, dass

$$\max_{\xi\in[5,5.6]} |C''''(xi)| = |C''''(5)| = 0.11444$$

ist, und daher ergibt die Fehlerabschätzung für $n = 3$ und $x = 5.3$

$$|C(5.3) - P_3(5.3)| \leq 8.583\text{E}{-6}.$$

Der interpolierte Wert (6.4) hat also etwa dieselbe Genauigkeit wie die tabellierten Werte.

Aufgabe 6.3 *Von der Funktion f kennt man die Werte*

x	1.7	2.0	2.3	2.6
y	0.996	0.483	−0.192	−0.926

Man berechne die Nullstelle von f so genau wie möglich.

Die naheliegende Lösung besteht darin, die Nullstelle des Interpolationspolynoms 3. Grades durch die vier Punkte zu berechnen. Dies geschieht am einfachsten, mittels Bisektion. Man ersetzt dazu die Funktion $f(x)$ in Aufgabe 3.2 durch das Interpolationspolynom

$$\begin{aligned} f(x) = P_3(x) = &-(x-2)(x-2.3)(x-2.6)\ 0.996/0.162 \\ &+(x-1.7)(x-2.3)(x-2.6)\ 0.483/0.054 \\ &+(x-1.7)(x-2)(x-2.6)\ 0.192/0.054 \\ &-(x-1.7)(x-2)(x-2.3)\ 0.926/0.162 \end{aligned}$$

und erhält mit den Startwerten $a = 2$ und $b = 2.3$ die Nullstelle

$$x = 2.2191545111.$$

Eine etwas einfachere Lösung erhält man mittels *inverser Interpolation* (vgl. Aufgabe 6.5).

Aufgabe 6.4 *Von der Funktion*

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{z^2}{2}}\, dz$$

kennt man die Werte

x	0.5	0.6	0.7	0.8
$f(x)$	0.6915	0.7257	0.7580	0.7881

Man interpoliere für $x = 0.52$.

Mit dem Algorithmus 6.2 erhält man für $z = 0.52$ das Aitken-Nevilleschema

0.5	0.6915			
0.6	0.7257	0.6983400		
0.7	0.7580	0.6998600	0.69849200	
0.8	0.7881	0.7038200	0.69827600	0.69847760

also das Resultat $P_3(0.52) = 0.69847760$.

Aufgabe 6.5 *Bei vielen Interpolationsproblemen ist in einer Tabelle der Funktionswert f* gegeben *und das Argument x gesucht. Als Beispiel betrachten wir die Tabelle in Aufgabe 6.4 und suchen jenes x, für welches*

$$f(x) = 0.7 \qquad (6.13*)$$

ist. Wenn wir in der üblichen Weise f durch ein Polynom P_n interpolieren, muss man x aus der Gleichung

$$P_n(x) = 0.7$$

berechnen, d.h. für unser Beispiel eine Gleichung dritten Grades lösen. Ein einfacheres Vorgehen besteht darin, die Umkehrfunktion zu interpolieren. *Man sucht somit den Wert $f^{[-1]}(0.7)$. Man muss dazu nur die Tabellenwerte für x und y vertauschen und normal interpolieren. Man löse die Gleichung* (6.13$*$) *auf beide Arten.*

Zuerst berechnen wir die Lösung der Gleichung $P_3(x) = 0.7$ mittels Bisektion wie in Aufgabe 6.3. Es ist

$$\begin{aligned} P_3(x) = &-(x-0.6)(x-0.7)(x-0.8)\ 0.6915/0.006 \\ &+(x-0.5)(x-0.7)(x-0.8)\ 0.7257/0.002 \\ &-(x-0.5)(x-0.6)(x-0.8)\ 0.7580/0.002 \\ &+(x-0.5)(x-0.6)(x-0.7)\ 0.7881/0.006. \end{aligned}$$

Für $f(x) = P_3(x) - 0.7 = 0$ erhält man mit den Startwerten $a = 0.5$ und $b = 0.6$ die Lösung $x = 0.52438890398$.

Interpoliert man die Umkehrfunktion für $z = 0.7$, indem die x- und y-Werte vertauscht werden, so erhält man mit Algorithmus 6.2 das Aitken-Neville-Schema

0.6915	0.5			
0.7257	0.6	0.524853801		
0.7580	0.7	0.520433437	0.524288792	
0.7881	0.8	0.507308970	0.525838866	0.524425186

und das Resultat $x = 0.524425186$.

Aufgabe 6.6 *Lösen von Gleichungen mittels inverser Interpolation. Gegeben sei eine nichtlineare Gleichung $f(x) = 0$. Ausgehend von 2 gegebenen Funktionswerten $f(x_0)$ und $f(x_1)$, welche am besten links und rechts von der Nullstelle gewählt werden, berechnet man für die Neustelle 0 das folgende Aitken-Neville-Schema:*

$$\begin{array}{l|llllll} f(x_0) & x_0 & & & & & \\ f(x_1) & x_1 & x_2 := T_{11} & & & & \\ f(x_2) & x_2 & T_{21} & x_3 := T_{22} & & & \\ f(x_3) & x_3 & T_{31} & T_{32} & x_4 := T_{33} & & \\ \cdots & \cdots & \text{u.s.w} & \cdots & \cdots & & \end{array}$$

Der extrapolierte Wert in der Diagonalen $x_{i+1} := T_{ii}$ wird als neuer Wert $T_{i+1,0}$ in die erste Kolonne des Schemas geschrieben. Hierauf wird damit der Funktionswert $f(x_{i+1})$ berechnet und dann kann die $i+1$-ste Zeile, d.h. die Elemente $T_{i+1,1}, \ldots, T_{i+1,i+1}$, berechnet werden. Bei Konvergenz (Startwerte genügend nahe wählen!) konvergiert die Diagonalfolge quadratisch *gegen die Nullstelle von f.*

Man schreibe ein Programm und löse damit die Gleichungen

$$a)\ x - \cos x = 0 \qquad b)\ x = e^{\sqrt{\sin x}}.$$

Zur Lösung dieser Aufgabe muss der Algorithmus 6.2 abgeändert werden. Die ersten beiden Näherungswerte für die Nullstelle werden eingelesen, danach wird jeweils der extrapolierte Wert als neuer Näherungswert verwendet. Die Extrapolation wird abgebrochen, wenn 7 Zeilen des Aitken-Neville-Schemas berechnet worden sind oder wenn die beiden letzten Elemente auf einer Zeile bis auf einen relativen Fehler von 1E−7 übereinstimmen. Man kann beweisen, dass das Verfahren für einfache Nullstellen *quadratisch* konvergiert. Dies ist recht attraktiv, denn es wird pro Iterationsschritt nur die Funktion $f(x)$ einmal ausgewertet.

```
(*$B- Aufgabe 6.6*)
program inverseinterpolation;
var n,i,j : integer; x,y : array [0..20] of real;
    z : real; tf : text; stri : string [20]; ch : char;

function f(x : real) : real;
begin
   case ch of
      'A','a' : f := x - cos(x) ;
      'B','b' : f := x - exp(sqrt(sin(x)))
   end
end ;
```

```
begin
  writeln('Wohin mit dem Output');
  readln(stri); assign(tf,stri); rewrite(tf);
  writeln(tf,'Loesen von Gleichungen mit inverser Interpolation');
  writeln('Welche Aufgabe: a oder b? ');
  readln(ch);
  i := 0;
  repeat
    if i < 2 then
    begin write('Startwert x[',i,'] =?'); read(y[i]) end
    else y[i] := y[0];
    x[i] := f(y[i]);
    write(tf,x[i] : 10 : 5, y[i] : 10 : 7);
    for j := i - 1 downto 0 do
    begin
      y[j] := (x[i] * y[j] - x[j] * y[j + 1])/(x[i] - x[j]);
      write(tf, y[j] : 10 : 7);
    end ; writeln(tf);
    i := i + 1;
  until (i = 7) or (abs(y[0] - y[1]) < abs(y[0]) * 1e - 7);
  writeln(tf);
  writeln(tf,'x =',y[0],'  f(x) =', f(y[0]));
  close(tf);
end.
```

Mit den Startwerten $x_0 = 0$ und $x_1 = 1$ erhält man für die Aufgabe a) das Schema:

−1.00000	0.0000000				
0.45970	1.0000000	0.6850734			
−0.08930	0.6850734	0.7362990	0.7413220		
0.00375	0.7413220	0.7390577	0.7390804	0.7390888	
0.00001	0.7390888	0.7390851	0.7390851	0.7390851	0.7390851

und die Lösung $x = 7.3908513322\text{E}{-}01$ mit $f(x) = 1.0004441720\text{E}{-}11$.

Für die Aufgabe b) und den Startwerten $x_0 = 2$ und $x_1 = 3$ resultiert

−0.59496	2.0000000				
1.54405	3.0000000	2.2781473			
−0.11313	2.2781473	2.3274271	2.3389981		
0.00395	2.3389981	2.3369467	2.3369711	2.3369845	
0.00001	2.3369845	2.3369798	2.3369798	2.3369798	2.3369798

mit der Lösung $x = 2.3369797663\text{E}{+}00$ und $f(x) = 0.0000000000\text{E}{+}00$ (zufällig!).

Aufgabe 6.7 *Man berechne $f'(1)$ für*

$$f(x) = x^2 \ln\left(\frac{\sqrt{x^3+1}\, e^x \left(x^3 + \sin x^2 + 1\right)}{2\left(\sin x + \cos^2 x + 3\right) + \ln x}\right)$$

Für diese Aufgabe kann man den Algorithmus 6.3 verwenden. Bei den Deklarationen wird zusätzlich die Funktion $f(x)$ aufgeführt und die **function** $t(h)$ entsprechend abgeändert:

```
function f(x : real) : real;
begin
   f := sqr(x) * ln((sqrt(x * x * x + 1) * exp(x)*
      (x * x * x + sin(sqr(x)) + 1))/(2 * (sin(x) + sqr(cos(x)) + 3) + ln(x)))
end ;

function t(h : real) : real;
begin
   t := (f(1 + h) - f(1 - h))/(2 * h)
end ;
```

Mit den Werten $h_0 = 0.4$, *faktor* $= 4$ und $n = 4$ erhält man das Schema:

```
4.036810328
3.791214401  3.709349093
3.731463359  3.711546345  3.711692829
3.716627785  3.711682594  3.711691677  3.711691659
3.712925266  3.711691094  3.711691660  3.711691660  3.711691660
```

und somit $f'(1) = 3.711691660$.

Aufgabe 6.8 *Extrapolation von π. Der Umfang des dem Einheitskreis einbeschriebenen regelmässigen n-Ecks beträgt*

$$U_n = 2n \sin\left(\frac{\pi}{n}\right). \qquad (6.24*)$$

Wir führen die Variable

$$h = \frac{1}{n}$$

und die Funktion

$$T(h) = \frac{U_n}{2} = n \sin\left(\frac{\pi}{n}\right) = \frac{\sin(h\pi)}{h} \qquad (6.25*)$$

ein. Entwickelt man $T(h)$ *in eine Reihe, so erhält man*

$$T(h) = \pi - \frac{\pi^3}{3!}h^2 + \frac{\pi^5}{5!}h^4 \mp \cdots \qquad (6.26*)$$

Wegen $\lim_{h\to 0} T(h) = \pi$, *kann* π *aus den halben Umfängen durch Extrapolation berechnet werden. Es kommen nur gerade Potenzen vor, also muss mit* (6.20∗) *extrapoliert werden. Man schreibe ein Programm dafür und benütze die folgende Tabelle der Umfänge einiger regelmässiger Vielecke, die sich elementargeometrisch berechnen lassen:*

n	2	3	4	5	6	8	10
$\frac{U_n}{2}$	2	$\frac{3}{2}\sqrt{3}$	$2\sqrt{2}$	$\frac{5}{4}\sqrt{10-2\sqrt{5}}$	3	$4\sqrt{2-\sqrt{2}}$	$\frac{5}{2}\left(\sqrt{5}-1\right)$

Da bei dieser Aufgabe der Extrapolationsparameter $h = 1/n$ *nicht immer halbiert* wird, muss man nach der Formel (6.16∗) extrapolieren. Man kann dazu den Algorithmus 6.2 übernehmen und abändern. Die gegebenen Umfänge lesen wir nicht ein, sondern weisen die Werte direkt den Variabeln im Programm zu. Damit der Ausdruck auf dem Bildschirm Platz hat, drucken wir das Extrapolationsschema nicht in einem festen Format, sondern mit zunehmenden Dezimalstellen für die extrapolierten Werte.

```
(*$B− Aufgabe 6.8*)
program pi;
var x,y : array [0..10] of real; i,j,k : integer;
begin
  x[0] := 1/2; y[0] := 2;
  x[1] := 1/3; y[1] := 3/2 * sqrt(3);
  x[2] := 1/4; y[2] := 2 * sqrt(2);
  x[3] := 1/5; y[3] := 5/4 * sqrt(10 - 2 * sqrt(5));
  x[4] := 1/6; y[4] := 3;
  x[5] := 1/8; y[5] := 4 * sqrt(2 - sqrt(2));
  x[6] := 1/10; y[6] := 5/2 * (sqrt(5) - 1);
  (* Wegen Gleichung (6.26* ): Extrapolation mit sqr(x[i])*)
  for i := 0 to 6 do x[i] := sqr(x[i]);
  for i := 0 to 6 do
  begin
    k := 6;
    write(x[i] : 5 : 3, y[i] : 13 - k : 10 - k);
    for j := i - 1 downto 0 do
    begin
      k := k - 1;
      y[j] := (x[i] * y[j] - x[j] * y[j + 1])/(x[i] - x[j]);
```

```
        write(y[j] : 13 − k : 10 − k)
      end ;
      writeln
   end
end.
```

Mit diesem Programm erhält man das Schema

0.250	2.0000					
0.111	2.5981	3.07654				
0.063	2.8284	3.12459	3.140611			
0.040	2.9389	3.13537	3.141431	3.1415872		
0.028	3.0000	3.13880	3.141552	3.1415920	3.14159264	
0.016	3.0615	3.14050	3.141582	3.1415926	3.14159265	3.141592654
0.010	3.0902	3.14120	3.141590	3.1415926	3.14159265	3.141592653	3.1415926535

Die Extrapolation wirkt sehr gut: aus dem 2-stelligen Resultat für $n = 10$ erhalten wir 11 richtige Dezimalstellen von π.

Aufgabe 6.9 *Die Folge*

$$s_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} - \ln n$$

ist konvergent. Der Grenzwert wurde schon von Euler berechnet und mit c bezeichnet. Man berechne c durch Extrapolation. Hinweis: Wie in der letzten Aufgabe führe man die Variable $h = 1/n$ ein.

Für diese Aufgabe kann man den Algorithmus 6.3 verwenden. Man muss dabei die **function** $t(h)$ anpassen. Wir ersetzen sie durch eine **function** $sum(n)$, welche s_n berechnet. Da das Hauptprogramm dasselbe wie bei der Aufgabe 6.10 ist, geben wir ein Programm an für die Lösung beider Aufgaben.

Die *integer* Variable *aufg* erhält den Wert 1,2 oder 3, je nachdem ob die Aufgabe 6.9, 6.10 a) oder 6.10 b) gelöst werden soll. In der **function** *sum* wird mittels einer **case** Anweisung darauf die entsprechende Summe berechnet. In allen 3 Fällem wird mit dem *faktor* = 2 extrapoliert, da ohne asymptotische Entwicklung nicht bekannt ist, ob nur gerade Potenzen von $h = 1/n$ auftreten.

```
(*$B- Aufgaben 6.9 und 6.10 *)
program summe;
var n,i,j,k,faktor, aufg : integer;
    x,y : array [0..10] of real; zhj,h : real;

function sum(n :integer) :real;
var k : integer; s :real;
begin
   case aufg of
      (*6.9*)
         1 : begin s := 0;
                for k := n downto 1 do s := s + 1/k; s := s - ln(n)
             end ;
      (*6.10a)*)
         2 : begin
                s := 0; for k := n downto 1 do s := s + sqr(1/k);
             end ;
      (*6.10b)*)
         3 : begin s := 0; (* aufpassen wegen integer Ueberlauf ! *)
                for k := n downto 1 do
                   s := s + (1 + 1/k + 1/k/k)/(3 + 1/k/k/k/k)/k/k;
                s := s + 1;
             end ;
   end ;
   sum:= s
end ;

begin
repeat
   writeln; writeln('1 : Aufgabe 6.9');
   writeln('2 : Aufgabe 6.10a)');
   writeln('3 : Aufgabe 6.10b)'); writeln;
   writeln('Waehle: 1,2 oder 3 ? (0: fertig)');
   read(aufg);
   if (0 <aufg) and (aufg< 4) then
   begin
      y[0] := sum(1);
      writeln(y[0] : 7 : 4); i := 0; n := 1; faktor:= 2;
      repeat
         n := 2 * n; i := i + 1;
         y[i] := sum(n); zhj := 1; k := 6;
         write(y[i] : 13 - k : 10 - k);
         for j := i - 1 downto 0 do
         begin
```

```
            k := k − 1;
            zhj := zhj/faktor;
            y[j] := (y[j + 1] − zhj ∗ y[j])/(1 − zhj);
            write(y[j] : 13 − k : 10 − k)
          end ;
          writeln
        until abs(y[0] − y[1]) <= 1e − 8 ∗ abs(y[0]);
        writeln('extrapolierter Wert:',y[0])
      end
  until aufg= 0
end.
```

Für die Euler'sche Konstante erhält man mit diesem Programm das Extrapolationsschema:

```
 1.0000
 0.8069  0.61371
 0.6970  0.58723   0.578398
 0.6384  0.57979   0.577315   0.5771598
 0.6081  0.57786   0.577223   0.5772094   0.57721266
 0.5928  0.57738   0.577216   0.5772152   0.57721558   0.577215673
 0.5850  0.57726   0.577216   0.5772156   0.57721566   0.577215666   0.5772156657
extrapolierter Wert: 5.7721566574E-01
```

Aufgabe 6.10 *Man berechne die Reihensumme der folgenden Reihen durch Extrapolation:*

$$a)\quad \sum_{k=1}^{\infty} \frac{1}{k^2} \qquad b)\quad \sum_{k=0}^{\infty} \frac{k^2+k+1}{3k^4+1}$$

Man mache die Variablentransformation $h = 1/n$ und extrapoliere die Summe aus den Partialsummen

$$T(h) = s_n = \sum_{k=0}^{n} a_k.$$

Man wähle die Folge (6.17∗) *für die h_i.*

Für die Lösung dieser Aufgabe wird wieder der Algorithmus 6.3 verwendet. Die Partialsummen werden als **function** *sum* berechnet. Bei der Aufgabe b) muss man mit dem *integer* Bereich aufpassen, da bei der Berechnung von

$3 * k^4$ schon für $k = 11$ ein Überlauf auftritt ! Es ist daher nötig, den k-ten Summanden wie folgt umzuformen. Zuerst kürzt man den Bruch mit k^2:

$$\frac{k^2 + k + 1}{3k^4 + 1} = \frac{1 + \frac{1}{k} + \frac{1}{k^2}}{3k^2 + \frac{1}{k^2}}$$

Um auch einen Überlauf bei der Berechnung von k^2 zu vermeiden, berechnet man den Bruch schliesslich durch

$$(1 + 1/k + 1/k/k)/(3 + 1/k/k/k/k)/k/k.$$

Man beachte, dass dabei die Divisionen von links nach rechts durchgeführt werden müssen (gleich wie in PASCAL).

Mit dem Programm von Aufgabe 6.9 erhält man für Aufgabe 6.10 a) den bekannten Wert $\pi^2/6$:

```
1.0000
1.2500  1.50000
1.4236  1.59722  1.629630
1.5274  1.63123  1.642570  1.6444185
1.5843  1.64127  1.644617  1.6449095  1.64494219
1.6142  1.64399  1.644894  1.6449332  1.64493475  1.644934510
1.6294  1.64469  1.644929  1.6449340  1.64493410  1.644934074  1.6449340673
extrapolierter Wert: 1.6449340673E+00
```

und für Aufgabe 6.10 b):

```
1.7500
1.8929  2.03571
1.9734  2.05403  2.060136
2.0149  2.05631  2.057072  2.0566343
2.0357  2.05651  2.056577  2.0565068  2.05649829
2.0461  2.05653  2.056530  2.0565237  2.05652482  2.056525677
2.0513  2.05653  2.056527  2.0565264  2.05652655  2.056526601  2.0565266156
extrapolierter Wert: 2.0565266156E+00
```

Aufgabe 6.11 *Man berechne eine Tabelle der Funktion*

$$f(x) = \prod_{n=1}^{\infty} \cos(\frac{x}{n})$$

für $x = 0, 0.1, \ldots, 1$. *Man extrapoliere jeden Funktionswert aus den Teilprodukten.*

Wir können wiederum den Algorithmus 6.3 mit *faktor* = 2 verwenden. Wir lassen diesmal den Ausdruck des Extrapolationsschemas weg und machen eine **function** daraus, welche die Funktion $f(x)$ berechnet. An die Stelle der **function** $t(h)$ tritt nun $prod(n)$:

```
(*$B- Aufgabe 6.11*)
program produkt;
var i : integer;

function f(z : real) : real;
var n,i,j,faktor : integer; x,y : array [0..10] of real; zhj : real;

function prod(z : real; n : integer) : real;
var k : integer; s : real;
begin
   s := 1; for k := 1 to n do s := s * cos(z/k);
   prod:= s
end ;

begin
   y[0] := 1; i := 0; n := 1; faktor:= 2;
   repeat
      n := 2 * n; i := i + 1; y[i] := prod(z, n); zhj:= 1;
      for j := i - 1 downto 0 do
      begin
         zhj:= zhj/ faktor;
         y[j] := (y[j + 1] - zhj * y[j])/(1 - zhj);
      end ;
   until abs(y[0] - y[1]) <= 1e - 9 * abs(y[0]);
   f := y[0]
end ;

begin
   writeln(' x       f(x)');
   for i := 1 to 10 do
   writeln(i/10 : 5 : 2, f(i/10) : 13 : 9)
end.
```

Man erhält damit die Tabelle:

x	$f(x)$
0.10	0.991800090
0.20	0.967495547
0.30	0.927957317
0.40	0.874589219
0.50	0.809259296
0.60	0.734209513
0.70	0.651949125
0.80	0.565137899
0.90	0.476465845
1.00	0.388536152

Aufgabe 6.12 *Man verifiziere die Gleichung* (6.33∗). *Hinweis: Man mache den Ansatz*

$$Q_i(t) = a + bt + ct^2 + dt^3$$

und bestimme die Koeffizienten a, b, c *und* d *durch Auflösen der Gleichungen* (6.32∗).

Es ist

$$\begin{aligned} Q_i(t) &= a + bt + ct^2 + dt^3 \\ Q_i'(t) &= b + 2ct + 3dt^2 \\ Q_i''(t) &= 2c + 6dt \end{aligned}$$

und es folgt aus $Q_i(0) = y_i$ und $Q_i'(0) = h_i y_i'$, dass

$$a = y_i \quad \text{und} \quad b = h_i y_i'. \tag{6.5}$$

Setzt man diese Werte in den beiden weiteren Gleichungen

$$\begin{aligned} Q_i(1) = \quad y_{i+1} &= a + b + c + d \\ Q_i'(1) = h_i y_{i+1}' &= b + 2c + 3d \end{aligned}$$

ein, so erhält man für c und d das Gleichungssystem

$$\begin{aligned} c + \quad d &= y_{i+1} - y_i - h_i y_i' \\ 2c + 3d &= h_i y_{i+1}' - h_i y_i' \end{aligned}$$

mit den Lösungen

$$c = 3y_{i+1} - 3y_i - 2h_i y_i' - h_i y_{i+1}' \tag{6.6}$$

$$d = h_i y_{i+1}' + h_i y_i' - 2y_{i+1} + 2y_i. \tag{6.7}$$

Somit ist

$$\begin{aligned} Q_i(t) &= y_i + h_i y_i' t + (3y_{i+1} - 3y_i - 2h_i y_i' - h_i y_{i+1}')t^2 \\ &\quad + (h_i y_{i+1}' + + h_i y_i' - 2y_{i+1} + 2y_i)t^3 \\ &= y_i(1 - 3t^2 + 2t^3) + y_{i+1}(3t^2 - 2t^3) \\ &\quad + h_i y_i'(t - 2t^2 + t^3) + h_i y_{i+1}'(-t^2 + t^3), \end{aligned}$$

was zu zeigen war.

Aufgabe 6.13 *Man berechne das Differenzenschema* (6.34∗) *algebraisch und verifiziere durch Umordnen, dass der entstehende Ausdruck in Gleichung* (6.35∗) *gleich dem in Gleichung* (6.33∗) *ist.*

Bildet man das Differenzenschema (6.34∗), so ergeben sich die Koeffizienten

$$\begin{aligned} a_0 &= y_i \qquad a_1 = y_{i+1} - y_i \\ a_2 &= y_{i+1} - y_i - h_i y_i' \\ b &= h_i y_{i+1}' - y_{i+1} + y_i \\ a_3 &= h_i y_{i+1}' + h_i y_i' - 2y_{i+1} + 2y_i \end{aligned}$$

Multipliziert man die Gleichung (6.35∗) aus, so wird

$$Q_i(t) = a_0 + (a_1 - a_2)t + (a_2 - a_3)t^2 + a_3 t^3.$$

Die Koeffizienten a_0 und a_3 stimmen mit den Koeffizienten a und d der Gleichungen (6.5) und (6.7) überein. Für die Koeffizienten von t und t^2 ergibt sich

$$\begin{aligned} a_1 - a_2 &= h_i y_i' = b \quad \text{und} \\ a_2 - a_3 &= 3y_{i+1} - 3y_i - 2h_i y_i' - h_i y_{i+1}' = c \end{aligned}$$

und damit ist die Gültigkeit des Differenzenschemas gezeigt.

Aufgabe 6.14 *Die Funktion $f(x) = \sin x$ werde durch ein kubisches Polynom $P_3(x)$ so approximiert, dass Funktionswert und erste Ableitung für $x = 0$ und $x = \pi$ übereinstimmen. Wie lautet das Polynom und wie gross ist der maximale Interpolationsfehler im Intervall $[0, \pi]$?*

Wir benützen die Darstellung (6.33∗) des Splinepolynoms für diese Aufgabe. Es ist $h = \pi$ und wegen

$$f(0) = f(\pi) = 0, \quad f'(0) = 1 \quad \text{und} \quad f'(\pi) = -1$$

ist

$$Q_i(t) = \pi(t - 2t^2 + t^3) - \pi(-t^2 + t^3) = \pi(t - t^2).$$

Wir erhalten also ein Polynom 2. Grades! In der Variabeln x lautet es wegen $t = x/\pi$:

$$P(x) = x - \frac{1}{\pi}x^2.$$

Der Interpolationsfehler ist

$$d(x) = \sin x - x + \frac{x^2}{\pi}$$

und wird maximal, wenn

$$d'(x) = \cos x - 1 + \frac{2x}{\pi} = 0 \tag{6.6}$$

ist. Es scheint, dass man diese Gleichung mit einem Nullstellenverfahren lösen muss. Auf Grund der Symmetrie des Problems wird man erwarten, dass der Fehler am grössten in der Intervallmitte bei $x = \pi/2$ ist. Tatsächlich ist dieser Wert eine Lösung und damit ist

$$|d(x)| \leq 1 - \pi/4 = 0.2146018 \quad \text{für} \quad x \in [0, \pi].$$

Aufgabe 6.15 *Man berechne das Polynom dritten Grades, welches die folgenden Daten interpoliert:*

x	2	3
$f(x)$	1	2
$f'(x)$	0.5	-2

Man berechne das Polynom auf zwei Arten:

1. *Man mache einen Ansatz mit unbekannten Koeffizienten und löse das entstehende lineare Gleichungssystem.*
2. *Man verwende die Formel* (6.33$*$) *und ordne nach Potenzen, so dass das Resultat mit 1. verglichen werden kann.*

Es sei $P(x) = a+bx+cx^2+dx^3$ das zu bestimmende Polynom. Die verlangte Interpolation führt auf das Gleichungssystem

$$\begin{aligned} a+ 2b + 4c + 8d &= 1 \\ b + 4c + 12d &= 0.5 \\ a+ 3b + 9c + 27d &= 2 \\ b + 6c + 27d &= -2 \end{aligned}$$

Aufgelöst mit einem Verfahren für lineare Gleichungssysteme erhält man

$$a = 44,\ b = -57.5,\ c = 25 \text{ und } d = -3.5,$$

somit das Polynom

$$P(x) = -3.5x^3 + 25x^2 - 57.5x + 44. \tag{6.7}$$

Andrerseits ist nach Gleichung (6.33∗)

$$\begin{aligned} Q_1(t) &= (1 - 3t^2 + 2t^3) + 2(3t^2 - 2t^3) + 0.5(t - 2t^2 + t^3) - 2(-t^2 + t^3) \\ &= 1 + 0.5t + 4t^2 - 3.5t^3. \end{aligned} \tag{6.8}$$

Nach Gleichung (6.29∗) ist $t = x - 2$. Wenn wir den Ausdruck (6.8) mit (6.7) vergleichen wollen, müssen wir das Polynom $Q_1(t)$ an der Stelle -2 umentwickeln. Wir benützen dazu das vollständige Hornerschema:

	-3.5	4	0.5	1
		7	-22	43
$t=-2$	-3.5	11	-21.5	$\vert 44$
		7	-36	
$t=-2$	-3.5	18	$\vert -57.5$	
		7		
$t=-2$	-3.5	$\vert 25$		

und erhalten dieselben Koeffizienten wie in Gleichung (6.7).

Aufgabe 6.16 *Von der Funktion $f(x)$ kennt man n* gleichabständige *Funktionswerte und deren erste Ableitungen:*

x	h	$2h$	$\ldots$	nh
$f(x)$	y_1	y_2	$\ldots$	y_n
$f'(x)$	y'_1	y'_2	$\ldots$	y'_n

Man berechne angenähert das Integral

$$\int_h^{nh} f(x)dx,$$

indem durch die Punkte eine Splinefunktion gelegt wird und über dieser integriert wird. Was für eine Quadraturformel erhält man?

Wir teilen das gesuchte Integral in $n-1$ Teilintegrale auf:

$$\int_h^{nh} f(x)\,dx = \sum_{i=1}^{n-1} \int_{ih}^{(i+1)h} f(x)\,dx.$$

Führt man $t = (x - ih)/h$ als neue Integrationsvariable ein, so wird $dx = h\,dt$ und

$$\int_{ih}^{(i+1)h} f(x)\,dx \simeq h \int_0^1 Q_i(t)\,dt.$$

Unter Benützung des Ausdrucks (6.33$*$) ist

$$\begin{aligned} h\int_0^1 Q_i(t)\,dt = h\Bigg[& y_i\left(t - t^3 + \frac{t^4}{2}\right) + y_{i+1}\left(t^3 - \frac{t^4}{2}\right) \\ & + hy_i'\left(\frac{t^2}{2} - \frac{2t^3}{3} + \frac{t^4}{4}\right) + hy_{i+1}'\left(-\frac{t^3}{3} + \frac{t^4}{4}\right)\Bigg]_0^1 \\ = {} & \frac{h}{2}(y_i + y_{i+1}) + \frac{h^2}{12}(y_i' - y_{i+1}'). \end{aligned}$$

Werden nun alle Teilintegrale aufsummiert, heben sich die Werte der Ableitungen bis auf den ersten und den letzten auf und man erhält die Formel

$$\int_h^{nh} f(x)\,dx \simeq= h\left(\frac{1}{2}y_1 + y_2 + \cdots + y_{n-1} + \frac{1}{2}y_n\right) + \frac{h^2}{12}(y_1' - y_n'), \qquad (6.9)$$

welche unter dem Namen *verbesserte Trapezregel* bekannt ist.

Aufgabe 6.17 *Man schreibe ein Programm zur Interpolation mit einer defekten Splinefunktion. Man interpoliere damit die Punkte*

x	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
y	0.0	0.4	1.0	0.4	0.0	-0.4	-1.0	-0.4	0.0

Im nachfolgenden Programm werden zuerst die Stützpunkte von einem File eingelesen. Die Ableitungen für die inneren Punkte werden nach der Gleichung (6.36$*$) berechnet und für die Randableitungen wird die gewünschte Variante zur Auswahl präsentiert. Für die angegebenen Stützpunkte passen

die periodischen Randbedingungen am besten. Die Interpolation geschieht mit dem Algorithmus 6.4, der als Include-File dazugeladen wird.

Es können einzelne Punkte interpoliert werden und anschliessend wird mit den Plotprozeduren (s. Kap. 8) eine Zeichnung auf dem Bildschirm erzeugt. Um die Zeichnung mittels der Taste *scrprt* auzudrucken, muss vor der Generierung der Zeichnung das DOS-Programm *graphics* aufgerufen werden.

```
(*$B− Aufgabe 6.17*)
program spline;
type vektor = array [1..20] of real;
var n,i,fall,xh,yh, xg,yg,xt,yt,oldx,oldy, farbe,keine : integer;
    h, xi,yi, xmin,xmax,ymin,ymax : real; x,y,ys : vektor;
    tf : text; stri : string [20];

(*$I A6.4 Algorithmus 6.4 : function g *)
(*$I plotproz *)

begin
    writeln('Inputfilenamen fuer die Punkte?');
    readln(stri); assign(tf,stri); reset(tf);
    if ( stri='CON : ') or ( stri='con : ') then
    writeln('Punkte (x, y) eingeben. Mit CTRL-Z abbrechen');
    n := 0;
    while not eof(tf) do
    begin
        n := n + 1; writeln( 'x,y'); read(tf, x[n], y[n]);
    end ;
    reset(tf);
    for i := 2 to n − 1 do
    ys[i] := (y[i + 1] − y[i − 1])/(x[i + 1] − x[i − 1]);
    repeat
        writeln('Welche Randbedingungen ?');
        writeln(' 1 : Steigung durch erste bzw. letzte beiden Pte.');
        writeln(' 2 : Natuerliche RB');
        writeln(' 3 : Periodische RB');
        read(fall);
    until fall in [1, 2, 3];
    case fall of
        1 : begin
                ys[1] := (y[2] − y[1])/(x[2] − x[1]);
                ys[n] := (y[n] − y[n − 1])/(x[n] − x[n − 1])
            end ;
        2 : begin
```

```
            ys[1] := 3/2 * (y[2] - y[1])/(x[2] - x[1]) - ys[2]/2;
            ys[n] := 3/2 * (y[n] - y[n - 1])/(x[n] - x[n - 1]) - ys[n - 1]/2
        end ;
    3 : begin
            ys[1] := (y[2] - y[n - 1])/(x[2] - x[1] + x[n] - x[n - 1]);
            ys[n] := ys[1]
        end
  end ;

  writeln('Neustelle eingeben, mit CTRL-Z aufhoeren');
  repeat
    write('x = '); read(xi); yi := g(n, xi, x, y, ys);
    writeln('  g(x) = ',yi);
  until eof; reset(input);
  writeln('xmin,xmax,ymin,ymax eingeben');
  readln( xmin,xmax,ymin,ymax);
  plotbegin( xg,yg,xt,yt,farbe,keine);
  achsen(xmin,xmax,ymin,ymax,xg,yg,xt,yt,farbe, keine);
  for i := 1 to n do kreuz(x[i], y[i], 5);
  gotoxy(1, 24);

  pl(xmin, g(n, xmin, x, y, ys),keine);
  h := ( xmax- xmin)/xg;
  for i := 0 to xg do
  begin
    xi := xmin+i * h; pl(xi, g(n, xi, x, y, ys), farbe)
  end
end.
```

Aufgabe 6.18 *Für die Punkte*

x	0	1	3	4	6
y	2	4	4	2	1

berechne man die Ableitungen nach folgenden Methoden:

a) defekter Spline mit natürlichen Randbedingungen

b) echter Spline mit natürlichen Randbedingungen

Unter Benützung der Gleichung (6.36∗) erhält man für die Ableitungen der defekten Splinefunktion

$$\mathbf{y}' = \left(\frac{8}{3}, \quad \frac{2}{3}, -\frac{2}{3}, -1, -\frac{1}{4} \right)^T$$

Für die Ableitungen der echten Splinefunktion erhält man nach (6.41∗) und (6.43∗) das tridiagonale Gleichungssystem

$$\begin{pmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 3 & 0.5 & 0 & 0 \\ 0 & 0.5 & 3 & 1 & 0 \\ 0 & 0 & 1 & 3 & 0.5 \\ 0 & 0 & 0 & 0.5 & 1 \end{pmatrix} \mathbf{y}' = \begin{pmatrix} 6 \\ 6 \\ -6 \\ -6.75 \\ -0.75 \end{pmatrix}$$

mit der Lösung

$$\mathbf{y}' = \begin{pmatrix} 2.2311827957 \\ 1.5376344086 \\ -1.6881720430 \\ -1.7043010753 \\ 0.1021505376 \end{pmatrix}$$

Aufgabe 6.19 *Wie lautet das Gleichungssystem für die Ableitungen einer echten Splinefunktion mit natürlichen Randbedingungen, wenn die Stützstellen gleichabständig* ($x_{i+1} - x_i = h = const$) *sind ?*

Für konstantes h kann man jede Gleichung des Gleichungssystems (6.41∗) mit h multiplizieren und erhält eine tridiagonale Matrix mit ganzzahligen Koeffizienten. Die rechte Seite vereinfacht sich auch, wenn man die Definition der Koeffizienten d_i (6.42∗) einsetzt. Man erhält

$$\begin{pmatrix} 2 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & \ddots & & \\ & & \ddots & \ddots & 1 & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{pmatrix} \mathbf{y}' = \frac{3}{h} \begin{pmatrix} y_2 - y_1 \\ y_3 - y_1 \\ y_4 - y_2 \\ \vdots \\ y_n - y_{n-2} \\ y_n - y_{n-1} \end{pmatrix}$$

Aufgabe 6.20 *Man beweise durch Ausmultiplizieren der beiden Matrizen*

$$\left(\mathbf{I} - \frac{\alpha}{\alpha \mathbf{q}^T \mathbf{p} + 1} \mathbf{p}\mathbf{q}^T \right) \cdot \left(\mathbf{I} + \alpha \mathbf{p}\mathbf{q}^T \right)$$

die Gleichung (6.57).

Multipliziert man die beiden Klammern aus, so wird

$$= \mathbf{I} - \frac{\alpha}{\alpha \mathbf{q}^T \mathbf{p} + 1} \mathbf{p}\mathbf{q}^T + \alpha \mathbf{p}\mathbf{q}^T - \frac{\alpha^2}{\alpha \mathbf{q}^T \mathbf{p} + 1} \mathbf{p} \underbrace{(\mathbf{q}^T \mathbf{p})}_{\text{Zahl}} \mathbf{q}^T$$

$$= \mathbf{I} + \left(\alpha - \frac{\alpha}{\alpha \mathbf{q}^T \mathbf{p} + 1} - \frac{\alpha^2 \mathbf{q}^T \mathbf{p}}{\alpha \mathbf{q}^T \mathbf{p} + 1} \right) \mathbf{p}\mathbf{q}^T$$

und der Ausdruck in der Klammer verschwindet.

Aufgabe 6.21 *Man erweitere die Prozedur tridia so, dass die beiden Gleichungssysteme* **Au**=**e** *und* **Av**=**d** *gleichzeitig gelöst werden können. Hinweis: Die Givensrotationen gleichzeitig auf die beiden rechten Seiten ausüben.*

In Algorithmus 6.5 enthält der Parameter x vor dem Aufruf von *tridia* die rechte Seite des Gleichungssystems, und nach dem Aufruf wird er durch die Lösung überschrieben. Wir ersetzen x jetzt durch die beiden rechten Seiten u und v. Alle Operationen, die auf x ausgeübt werden, müssen jetzt auf die beiden neuen Vektoren übertragen werden. Man erhält

```
procedure tridiazwei(n : integer; var c,a,b,u,v : vektor);
var i : integer; co,si,h,t : real;
begin
  b[n] := 0;
  for i := 1 to n - 1 do
  if c[i] <> 0 then
  begin
    t := a[i]/c[i]; si := 1/sqrt(1 + t * t); co := t * si;
    a[i] := a[i] * co + c[i] * si; h := b[i];
    b[i] := h * co + a[i + 1] * si; a[i + 1] := -h * si + a[i + 1] * co;
    c[i] := b[i + 1] * si; b[i + 1] := b[i + 1] * co;
    h := u[i]; u[i] := h * co + u[i + 1] * si;
    u[i + 1] := -h * si + u[i + 1] * co;
    h := v[i]; v[i] := h * co + v[i + 1] * si;
    v[i + 1] := -h * si + v[i + 1] * co;
end ;
(* Rueckwaertseinsetzen *)
u[n] := u[n]/a[n]; u[n - 1] := (u[n - 1] - b[n - 1] * u[n])/a[n - 1];
v[n] := v[n]/a[n]; v[n - 1] := (v[n - 1] - b[n - 1] * v[n])/a[n - 1];
for i := n - 2 downto 1 do
begin
```

```
      u[i] := (u[i] − b[i] ∗ u[i + 1] − c[i] ∗ u[i + 2])/a[i];
      v[i] := (v[i] − b[i] ∗ v[i + 1] − c[i] ∗ v[i + 2])/a[i];
   end
end ;
```

Aufgabe 6.22 *Die inverse Matrix von*

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{pmatrix}$$

ist

$$\mathbf{A}^{-1} = \begin{pmatrix} 4 & -6 & 4 & -1 \\ -6 & 14 & -11 & 3 \\ 4 & -11 & 10 & -3 \\ -1 & 3 & -3 & 1 \end{pmatrix}$$

Die Matrix

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{pmatrix}$$

kann als Rang 1 Modifikation von $\mathbf{A}$ *dargestellt werden. Man benütze diese Darstellung um das Gleichungssystem*

$$\mathbf{Bx} = (1, 0, 0, 0)^T$$

unter Verwendung von $\mathbf{A}^{-1}$ *zu lösen.*

Es ist

$$\mathbf{B} = \mathbf{A} - \mathbf{e}_1\mathbf{e}_4^T = \mathbf{A}(\mathbf{I} - \mathbf{A}^{-1}\mathbf{e}_1\mathbf{e}_4^T).$$

Die gesuchte Lösung ist

$$\mathbf{x} = \mathbf{B}^{-1}\mathbf{e}_1 = (\mathbf{I} - \mathbf{A}^{-1}\mathbf{e}_1\mathbf{e}_4^T)^{-1}\mathbf{A}^{-1}\mathbf{e}_1.$$

Mit der Bezeichnung $\mathbf{u} := \mathbf{A}^{-1}\mathbf{e}_1 = \mathbf{c}\mathbf{A}_1^{-1}$ und der Benützung von Satz 6.4 ist

$$\mathbf{x} = (\mathbf{I} + \frac{1}{1 - u_4}\ \mathbf{u}\mathbf{e}_4^T)\mathbf{u} = \mathbf{u} + \frac{1}{2}\mathbf{u}(\mathbf{e}_4^T\mathbf{u}).$$

Setzt man im letzten Ausdruck die Werte der ersten Kolonne der inversen Matrix von $\mathbf{A}$ für $\mathbf{u}$ ein, so erhält man die Lösung

$$x = (\ \ 2, -3,\ \ 2, -0.5)^T.$$

Aufgabe 6.23 *Man berechne die inverse Matrix von*

$$\mathbf{A} = \begin{pmatrix} \alpha+1 & 1 & 1 & 1 \\ 1 & \alpha+1 & 1 & 1 \\ 1 & 1 & \alpha+1 & 1 \\ 1 & 1 & 1 & \alpha+1 \end{pmatrix}$$

mit $\alpha > 0$ mittels der Technik der Rang 1 Modifikation.

Wir lösen die Aufgabe für allgemeines n. Es ist

$$\mathbf{A} = \alpha\mathbf{I} + \mathbf{e}\mathbf{e}^T = \alpha(\mathbf{I} + \frac{1}{\alpha}\mathbf{e}\mathbf{e}^T) \quad \text{wobei} \quad \mathbf{e} = (1, 1, \ldots, 1)^T.$$

Nach Satz 6.4 ist wegen $\mathbf{e}^T\mathbf{e} = n$

$$\begin{aligned} \mathbf{A}^{-1} &= \left(\mathbf{I} - \frac{1/\alpha}{n/\alpha+1}\,\mathbf{e}\mathbf{e}^T\right)\frac{1}{\alpha} \\ &= \frac{1}{\alpha}\mathbf{I} - \frac{\alpha}{n+\alpha}\mathbf{e}\mathbf{e}^T \\ &= \frac{\alpha}{n+\alpha}\left(\frac{n+\alpha}{\alpha^2}\mathbf{I} - \mathbf{e}\mathbf{e}^T\right) \\ &= \frac{\alpha}{n+\alpha}\begin{pmatrix} \beta & -1 & \cdots & -1 \\ -1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \beta & -1 \\ -1 & \cdots & -1 & \beta \end{pmatrix} \text{ mit } \beta = \frac{n+\alpha}{\alpha^2} - 1 \end{aligned}$$

Für $n = 4$ und $\alpha = 1$ erhält man zum Beispiel

$$\begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix} = \frac{1}{5}\begin{pmatrix} 4 & -1 & -1 & -1 \\ -1 & 4 & -1 & -1 \\ -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 \end{pmatrix}$$

Aufgabe 6.24 *Man schreibe eine Prozedur zur Lösung von tridiagonalen linearen Gleichungssystemen nach der Methode der Gauss'schen Dreieckszerlegung (siehe Kap. 5). Man mache dabei den Ansatz*

$$\begin{pmatrix} a_1 & b_1 & & \\ c_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & c_{n-1} & a_n \end{pmatrix} = \begin{pmatrix} 1 & & & \\ l_1 & 1 & & \\ & \ddots & \ddots & \\ & & l_{n-1} & 1 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 & \beta_1 & & \\ & \alpha_2 & \ddots & \\ & & \ddots & \beta_{n-1} \\ & & & \alpha_n \end{pmatrix} \tag{6.10}$$

Man multipliziere die beiden Matrizen auf der rechten Seite aus und stelle durch Vergleich mit der linken Seite Rekursionsformeln für die l_i, α_i und β_i auf. Die Auflösung des Gleichungssystems erfolgt durch Vorwärts- und Rückwärtseinsetzen. Man vergleiche die Anzahl der Rechenoperationen mit denjenigen der Prozedur tridia.

Wenn man die beiden Bidiagonalmatrizen auf der rechten Seite von Gleichung (6.10) ausmultipliziert und den Koeffizientenvergleich auf der i-ten Zeile macht, so ergeben sich die Gleichungen

$$c_i = l_i\alpha_i, \quad a_{i+1} = l_i\beta_i + \alpha_{i+1} \quad \text{und} \quad b_{i+1} = \beta_{i+1}.$$

Die Koeffizienten β_i stimmen also mit den b_i überein. Die Koeffizienten l_i und α_i lassen sich iterativ berechnen:

$$\begin{aligned}
& & \alpha_1 &= a_1 \\
l_1 &= \frac{c_1}{\alpha_1} & \alpha_2 &= a_2 - l_1 b_1 \\
& \cdots & & \cdots \\
l_i &= \frac{c_i}{\alpha_i} & \alpha_{i+1} &= a_{i+1} - l_i b_i \\
& \cdots & & \cdots \\
l_{n-1} &= \frac{c_{n-1}}{\alpha_{n-1}} & \alpha_n &= a_n - l_{n-1} b_{n-1}.
\end{aligned}$$

Man kann wie bei Aufgabe 5.7 die gegebene Matrix durch die Zerlegung überschreiben, d.h. c_i durch l_i und a_i durch α_i. Nach der Zerlegung wird die Lösung durch Vorwärts- und Rückwärtseinsetzen erhalten:

```
procedure tridiagauss(n : integer; c, a, b : vektor; var y : vektor);
(* Loest ein tridiagonales Gleichungssystem der Form (ci, ai, bi)x = y *)
var i : integer;
begin
   (* Dreieckszerlegung *)
   for i := 1 to n − 1 do
   begin
      c[i] := c[i]/a[i];
      a[i + 1] := a[i + 1] − c[i] ∗ b[i]
   end ;
   (* Vorw. und Rueckw.einsetzen *)
   for i := 2 to n do y[i] := y[i] − c[i − 1] ∗ y[i − 1];
   y[n] := y[n]/a[n];
   for i := n − 1 downto 1 do y[i] := (y[i] − b[i] ∗ y[i + 1])/a[i];
end ;
```

Beim Vergleich des Rechenaufwandes zählt man üblicherweise nur die Anzahl der Multiplikationen, bzw. Divisionen. Bei *tridiagauss* erfordert die Dreieckszerlegung $\sim 2n$, das Vorwärtseinsetzen $\sim n$ und das Rückwärtseinsetzen

$\sim 2n$ Operationen. Total benötigt man also $\sim 5n$ Operationen. Beim Algorithmus 6.5 *tridia* erfordert die Transformation auf die obere tridiagonale Dreiecksmatrix $\sim 17n$ Operationen, wenn man die n Quadratwurzeln gleich wie die Multiplikationen und Divisionen zählt. Das Rückwärtseinsetzen benötigt noch weitere $\sim 3n$ Operationen, sodass hier ein Rechenaufwand von $\sim 20n$ resultiert, *also etwa 4 mal soviel wie mit der Gauss'schen Dreieckszerlegung.*

Aufgabe 6.25 *Man schreibe ein Programm für die Interpolation mittels einer echten Splinefunktion.*

Im folgenden Programm werden mit der Prozedur *ableitungen* die Ableitungen $\mathbf{y}'$ berechnet, je nach der gewünschten Variante, die in der Prozedur *menue* gewählt wurde. Als Include-Files werden die **function** *g* (Algorithmus 6.4), die Prozedur *tridia* (Algorithmus 6.5), die Prozedur *tridiazwei* von Aufgabe 6.21 und die Plotprozeduren (s. Kap. 8) dazugeladen.

Nach dem Berechnen der Ableitungen können einzelne Punkte interpoliert werden und danach wird eine Zeichnung der Splinefunktion mit den Stützpunkten auf dem Bildschirm erzeugt. Um die Zeichnung mittels der Taste *scrprt* auzudrucken, muss vor dem Erzeugen der Zeichnung das DOS-Programm *graphics* aufgerufen werden.

```
(*$B− Aufgabe 6.25*)
program spline;
type vektor = array [1..100] of real;
var n,i,fall,xh,yh, xg,yg,xt,yt,oldx,oldy, farbe,keine : integer;
    h, xi,yi, xmin,xmax,ymin,ymax : real; x, y, ys : vektor;
    f : text; stri : string [20];

(*$I A6.4 Algorithmus 6.4 : function g*)
(*$I plotproz *)
(*$I a6.5 Algorithmus 6.5 : tridia *)
(*$I tridiazwei von Aufgabe 6.21*)

procedure ableitungen(fall,n : integer; var x, y, ys : vektor);
var i : integer; fak : real; a, b, c, h, d, u, v : vektor;
begin
    for i := 1 to n − 1 do h[i] := x[i + 1] − x[i];
    for i := 1 to n − 1 do d[i] := (y[i + 1] − y[i])/sqr(h[i]);
    for i := 2 to n − 1 do
    begin
        a[i] := 2/h[i − 1] + 2/h[i]; b[i] := 1/h[i];
        c[i] := b[i]; ys[i] := 3 ∗ (d[i] + d[i − 1]);
    end ;
```

```
case fall of
1 : (* natuerlich *)
    begin
        b[1] := 1/h[1]; c[1] := b[1]; a[1] := 2 * b[1];
        a[n] := 2/h[n − 1];
        ys[1] := 3 * d[1]; ys[n] := 3 * d[n − 1];
        tridia(n, c, a, b, ys);
    end ;
2 : (* de Boor*)
    begin
        a[1] := 1/h[1]; c[1] := a[1]; b[1] := a[1] + 1/h[2];
        a[n] := 1/h[n − 1]; c[n − 1] := a[n] + 1/h[n − 2];
        ys[1] := 2 * d[1] + (d[1] + d[2])/(h[1] + h[2]) * h[1];
        ys[n] := 2 * d[n − 1] + (d[n − 1] + d[n − 2])/
                 (h[n − 1] + h[n − 2]) * h[n − 1];
        tridia(n, c, a, b, ys);
    end ;
3 : (* periodisch*)
    begin
        b[1] := 1/h[1]; c[1] := b[1]; a[1] := (2 * b[1] + 1/h[n − 1]);
        a[n − 1] := 2/h[n − 2] + 1/h[n − 1];
        for i := 2 to n − 1 do
        begin u[i] := 0; v[i] := 3 * (d[i] + d[i − 1]) end ;
        u[1] := 1; u[n − 1] := 1; v[1] := 3 * (d[1] + d[n − 1]);
        tridiazwei(n − 1, c, a, b, u, v);
        fak:= (v[1] + v[n − 1])/(u[1] + u[n − 1] + h[n − 1]);
        for i := 1 to n − 1 do ys[i] := v[i]− fak∗u[i];
        ys[n] := ys[1];
    end ;
end ;

procedure ini;
begin
    writeln('Inputfilenamen ?'); readln(stri);
    assign(f, stri); reset(f);
    writeln('Punkte x,y eingeben. Die x in aufsteigender Reihenfolge');
    writeln('Nach letztem y mit ctrl-z abbrechen');
    n := 0;
    while not eof(f)do
    begin
        n := n + 1; writeln('x,y?'); read(f, x[n], y[n])
    end ;
```

```
   for i := 1 to n do writeln(x[i], y[i]);
end ;

procedure menue;
begin
   repeat
      writeln('Fall fuer Ableitungen :');
      writeln('1 : Natuerliche Randbedingungen');
      writeln('2 : Not a knot condition von de Boor');
      writeln('3 : Periodischer Spline');
      readln(fall)
   until fall in [1..3]
end ;

begin
   ini; reset(input); menue;
   ableitungen(fall, n, x, y, ys);
   writeln('Ableitungen');
   for i := 1 to n do writeln(ys[i]);
   writeln('Neustelle eingeben, mit CTRL-Z aufhoeren');
   repeat
      write('x = '); read(xi); yi := g(n, xi, x, y, ys);
      writeln('  g(x) = ',yi);
   until eof; reset(input);
   writeln('xmin,xmax,ymin,ymax eingeben');
   readln(xmin,xmax,ymin,ymax);
   plotbegin( xg,yg,xt,yt,farbe,keine);
   achsen(xmin,xmax,ymin,ymax,xg,yg,xt,yt,farbe, keine);
   for i := 1 to n do kreuz(x[i], y[i], 5);
   gotoxy(1, 24);
   pl(xmin,g(n, xmin, x, y, ys),keine);
   h := (xmax− xmin)/xg;
   for i := 0 to xg do
   begin
      xi := xmin+i ∗ h; pl(xi, g(n, xi, x, y, ys), farbe)
   end
end.
```

Aufgabe 6.26 *Die Gleichungen*

$$x = \sin t$$
$$y = \sin(2t - \frac{\pi}{4}) \qquad (6.61*)$$

stellen eine geschlossene Kurve dar. Man schreibe ein Programm für die Splineinterpolation von Kurven, berechne n Punkte von (6.61∗)*, interpoliere sie und vergleiche die Interpolation mit der exakten Kurve (durch Berechnen einer Wertetabelle oder durch eine Zeichnung).*

Im folgenden Programm kann man für die Interpolation defekte oder echte Splinefunktionen wählen. Die Prozedur *ableitungen* ist aus den entsprechenden Prozeduren von Aufgaben 6.17 und 6.25 zusammengesetzt. Sie wird im Hauptprogramm zweimal aufgerufen: zuerst, um aus den (s_i, x_i) die Ableitungen x'_i und darauf, um aus (s_i, y_i) die y'_i zu berechnen. Die Parameterwerte s_i werden beim Einlesen nach (6.60∗) berechnet.

Die Splinekurve wird gezeichnet, indem jeweils 20 Zwischenpunkte in einem Intervall $[t_i, t_{i+1}]$ berechnet werden. Wenn man bei der angegebenen Kurve (6.61∗) 30 Punkte berechnet, so unterscheiden sich die beiden Graphen bei periodischen Randbedingungen praktisch nicht mehr auf dem Bildschirm (s. Figuren 6.1 und 6.2).

```
(*$B− Aufgabe 6.26*)
program kurve;
type vektor = array [1..100] of real;
var n,i,j,fall,xh,yh, xg,yg,xt,yt,oldx,oldy, farbe,keine : integer;
    h, si, xmin,xmax,ymin,ymax : real; s,x,y,xs,ys : vektor;
    f : text; stri : string [20];

(*$I A6.4 Algorithmus 6.4 : function g *)
(*$I plotproz *)
(*$I A6.5 Algorithmus 6.5 : tridia *)
(*$I tridiazwei von Aufgabe 6.21*)
procedure ableitungen(fall,n : integer; var x,y,ys : vektor);
var i : integer; fak : real; a,b,c,h,d,u,v : vektor;
begin
   for i := 1 to n − 1 do h[i] := x[i + 1] − x[i];
   for i := 1 to n − 1 do d[i] := (y[i + 1] − y[i])/sqr(h[i]);
   case fall of
      1, 2, 3 : (* defekter Spline *)
      begin
         for i := 2 to n − 1 do
         ys[i] := (y[i + 1] − y[i − 1])/(x[i + 1] − x[i − 1]);
         case fall of
            1 : begin
                   ys[1] := d[1] ∗ h[1]; ys[n] := d[n − 1] ∗ h[n − 1]
                end ;
            2 : begin
                      ys[1] := 1.5 ∗ d[1] ∗ h[1] − 0.5 ∗ ys[2];
```

Figur 6.1: die Kurve (6.61∗)

Figur 6.2: echter Spline mit 30 Punkten

```
                ys[n] := 1.5 * d[n-1] * h[n-1] - 0.5 * ys[n-1];
            end ;
        3 : begin
                ys[1] := (y[2] - y[n-1])/(h[1] + h[n-1]);
                ys[n] := ys[1]
            end ;
    end ;
end ;

21, 22, 23 : (* echter Spline *)
begin
    for i := 2 to n - 1 do
    begin
        a[i] := 2 * (1/h[i-1] + 1/h[i]);
        b[i] := 1/h[i];
        c[i] := b[i];
        ys[i] := 3 * (d[i] + d[i-1]);
    end ;
    case fall of
        21 : (* natuerlich *)
            begin
                b[1] := 1/h[1]; c[1] := b[1]; a[1] := 2 * b[1];
                a[n] := 2/h[n-1];
                ys[1] := 3 * d[1]; ys[n] := 3 * d[n-1];
                tridia(n, c, a, b, ys);
            end ;
        22 : (* de Boor *)
    begin
        a[1] := 1/h[1] c[1] := a[1]; b[1] := a[1] + 1/h[2];
        a[n] := 1/h[n-1]; c[n-1] := a[n] + 1/h[n-2];
        ys[1] := 2 * d[1] + (d[1] + d[2])/(h[1] + h[2]) * h[1];
        ys[n] := 2 * d[n-1] + (d[n-1] + d[n-2])/
                 (h[n-1] + h[n-2]) * h[n-1];
        tridia(n, c, a, b, ys);
    end ;
23 : (* periodisch *)
    begin
        b[1] := 1/h[1]; c[1] := b[1];
        a[1] := (2 * b[1] + 1/h[n-1]);
        a[n-1] := 2/h[n-2] + 1/h[n-1];
        for i := 2 to n - 1 do
        begin u[i] := 0; v[i] := 3 * (d[i] + d[i-1]) end ;
```

```
                u[1] := 1; u[n − 1] := 1; v[1] := 3 ∗ (d[1] + d[n − 1]);
                tridiazwei(n − 1, c, a, b, u, v);
                fak:= (v[1] + v[n − 1])/(u[1] + u[n − 1] + h[n − 1]);
                for i := 1 to n − 1 do ys[i] := v[i]−fak∗u[i];
                ys[n] := ys[1];
            end ;
        end ;
    end ;
end ;

procedure ini;
begin
    writeln('Inputfilenamen ? Fuer diese Aufgabe: DAT6.26');
    readln(stri); assign(f, stri); reset(f);
    writeln('Punkte x,y: mit CTRL-Z aufhoeren');
    n := 0;
    while not eof(f)do
    begin
        n := n + 1;
        writeln('x,y'); read(f, x[n], y[n])
    end ;
    s[1] := 0;
    for i := 2 to n do
    s[i] := s[i − 1] + sqrt(sqr(x[i] − x[i − 1]) + sqr(y[i] − y[i − 1]));
    for i := 1 to n do writeln(s[i], x[i], y[i]);
end ;

procedure menue;
begin
    writeln('Fall fuer Ableitungen eingeben :');
    writeln('1:lineare Interpolation ');
    writeln('2:lineare Interpolation, natuerliche RB');
    writeln('3:lineare Interpolation, periodische RB');
    writeln('21:echter Spline, natuerliche RB');
    writeln('22:echter Spline, de Boor RB');
    writeln('23:echter Spline, periodische RB');
    readln(fall);
end ;

begin
    ini; menue;
    ableitungen( fall,n, s, x, xs);
    ableitungen( fall,n, s, y, ys);
```

```
    writeln('xmin,xmax,ymin,ymax eingeben');
    readln(xmin,xmax,ymin,ymax);
    plotbegin( xg,yg,xt,yt,farbe,keine);
    achsen(xmin,xmax,ymin,ymax,xg,yg,xt,yt,farbe, keine);
    for i := 1 to n do kreuz(x[i], y[i], 5);
    gotoxy(1, 24);
    pl(g(n, 0, s, x, xs), g(n, 0, s, y, ys), keine);
    for i := 1 to n − 1 do
    begin
        h := (s[i + 1] − s[i])/20;
        for j := 0 to 20 do
        begin
            si := s[i] + j ∗ h;
            pl(g(n, si, s, y, ys), g(n, si, s, x, xs), farbe);
        end ;
    end ;
end.
```

Kapitel 7

Numerische Integration

Für dieses Kapitel schreiben wir zuerst ein Hauptprogramm, mit dem man mittels Menuetechnik verschiedene Funktionen mit den 4 Integrationsmethoden integrieren kann. Die einzelnen Methoden *trapez, simpson, romberg* und *adapt* werden als Include-Files dazugeladen. In TURBO PASCAL kann man keine Funktionen als Parameter übergeben. Wir müssen daher einen globalen Funktionsnamen f verwenden und in der **function** f eine **case** Anweisung benützen, um die verschiedenen Aufgaben zu lösen.

```
function f(x : real) : real;
begin
   case fall of
      1 : f := x * exp(x)/sqr(x + 1);
      2 : f := sqrt(x);
      3 : f := ln(x);
      4 : f := x * ln(x);
   end ;
end ;
```

Die *integer* Variable *fall* ist global und erhält beim Aufruf der Prozedur *menue* einen Wert zugewiesen. Die Prozedur *menue* muss die in f aufgeführten Funktionen auf dem Bildschirm zur Auswahl präsentieren. Wenn man eine weitere Funktion in die Liste aufnehmen will, muss man *sowohl die Prozedur menue als auch die* **function** f *abändern*. Dies ist nicht nur lästig, sondern kann auch zu Fehlern führen.

Die Möglichkeit Include-Files zu verwenden, schafft hier eine elegante Lösung. Wir schreiben die Liste der Funktionen auf das File *f.pas* und laden es in f als Include-File dazu:

```
function f(x : real) : real;
begin
   case fall of
      (*$I f.pas *)
   end ;
end ;
```

Bei einer Änderung der Funktionen muss nur das File *f.pas* editiert werden. Die Prozedur *menue* kann nun das File *f.pas* als Text-File lesen, auf den Bildschirm schreiben und dem Benützer zur Auswahl stellen. So muss am

Hauptprogramm nichts mehr geändert werden: bei Veränderungen des Files *f.pas* wird das Menue automatisch nachgeführt. Das Hauptprogramm wird damit:

```
(*$B-*)
program integral;
var a, b, eps, int : real; i, z, fall : integer; ;
    tf : text; stri : string [20];

function f(x : real) : real;
begin
    case fall of
        (*$I f.pas*)
    end ;
    z := z + 1;
end ;

(*$I trapez Algorithmus 7.1*)
(*$I simpson Algorithmus 7.2*)
(*$I romberg Algorithmus 7.3*)

function ada(a, b, eps :real) : real;
var fa,fb,fm, is : real; v : integer;
(*$I adapt Algorithmus 7.4*)
begin
    fa := f(a); fm := f((a + b)/2); fb := f(b);
    is := (b - a)/6 * (fa + 4 * fm + fb) ;
    v := 1; if is < 0 then v := -1;
    is := v * (abs(is) + b - a)/2 * eps/1e-11;
    writeln(tf,'a': 5,'b - a': 15,'i1': 16);
    ada:= adapt(a, b, fa, fm, fb, is);
end ;

procedure menue;
var f :text; stri : string [100];
begin
    assign(f,'f.pas'); reset(f);
    while not eof(f) do
    begin
        readln(f, stri); writeln( stri)
    end ;
end ;

begin
    writeln('Wohin mit dem Output ? ');
    read(stri); assign(tf,stri); rewrite(tf);
```

```
  menue;
  writeln('Fall'); read(fall);
  writeln('a,b und eps eingeben'); read(a,b,eps);

  repeat
    z := 0;
    writeln; writeln('Welche Integrationsmethode ?');
    writeln('0 : fertig ');
    writeln('1 : Trapezregel   2 : Simpson ');
    writeln('3 : Romberg       4 : Adaptive Quadratur ');
    read(i);
    if i in [1,2,3,4] then
    begin
      case i of
        1 : int:= trapez(a,b, eps);
        2 : int:= simpson(a,b, eps);
        3 : int := romberg(a,b, eps);
        4 : int := ada(a,b, eps);
      end ;
      write(tf,'Integral = ',int,'  Anzahl Fkt.Aufr.  ', z);
    end ;
  until i = 0;

  close(tf)
end.
```

Die Algorithmen zum Integrieren 7.1 bis 7.4 wurden durch *write*-Anweisungen ergänzt, so dass aufeinanderfolgende Näherungswerte ausgedruckt werden. Die **function** *ada* ruft den Algorithmus 7.4, die **function** *adapt*, auf. Zusätzlich wird nach Gleichung (7.48$*$) der Schätzwert is automatisch berechnet, so dass nur noch die Integrationsgrenzen und die gewünschte Genauigkeit *eps* als Parameter übergeben werden müssen.

Infolge eines TURBO PASCAL Compiler Fehlers (Version 3.01A) muss man das Abbruchkriterium in *adapt* (mathematisch und numerisch aequivalent) umformen. Anstatt **if** $is + i1 = is + i2$ **then** muss man

$$\textbf{if } is + (i1 - i2) = is \textbf{ then}$$

verwenden. In der jetzigen Version von TURBO PASCAL gibt es Zahlen, die das erste Kriterium nicht erfüllen, wohl aber das zweite. Zum Beispiel gilt für die Zahlen

$$\begin{aligned} i1 &:= \quad 0.0048947334290; \\ i2 &:= \quad 0.0048947334289; \\ is &:= 458333.33333; \end{aligned}$$

$(is+i1)-(is+i2) = 4.7683715820\text{E}{-}07$, was nicht richtig ist. Da die beiden Klammern bis auf 10^{-11} übereinstimmen, müsste das richtige Resultat 0 sein.

Es gibt aber ähnliche Zahlen, bei denen der Compiler den Fehler nicht macht. Betrachtet man statt $i2$ die Zahl $i3 := 0.0048947334291$, welche sich nur um 2 Einheiten der letzten Stelle von $i2$ unterscheidet, so ist jetzt $(is + i1) - (is + i3) = 0$, wie es sein sollte.

Aufgabe 7.1 *Man berechne mit der Trapezregel für $h = 0.5$ und $h = 0.25$ das Integral $\int_2^4 \ln x\, dx$. Man berechne den Integrationsfehler in beiden Fällen. Wieviel nimmt der Fehler ab bei der Halbierung der Schrittweite ?*

Es ist $I = \int_2^4 \ln x\, dx = [x \ln x - x]_2^4 = 2.158883083$ der exakte Wert. Für $h = 0.5$ liefert die Trapezregel

$$T(0.5) = 0.5 \left(\frac{1}{2} \ln 2 + \ln 2.5 + \ln 3 + \ln 3.5 + \frac{1}{2} \ln 4 \right) = 2.15369338$$

und für $h = 0.25$ erhält man analog $T(0.25) = 2.157582181$. Damit ergibt sich für den Fehler:

$$I - T(0.5) = 5.189703 \times 10^{-3} \text{ und } I - T(0.25) = 1.300902 \times 10^{-3},$$

also

$$\frac{I - T(0.5)}{I - T(0.25)} = 3.9893.$$

Der Fehler sinkt bei Halbierung der Schrittweite wie von der Fehlerabschätzung zu erwarten ist etwa auf den 4. Teil, wie man es auch schon am Beispiel 7.1 sehen konnte.

Aufgabe 7.2 *Wie klein muss die Schrittweite h sein, damit das Integral $\int_2^5 x \ln x\, dx$ bis auf einen Fehler kleiner als 10^{-8} mit der Trapezregel berechnet werden kann ? Man schätze h mittels Gleichung (7.5$*$) und prüfe das Resultat durch Integration mit dem Programm trapez nach.*

Aus $f(x) = x \ln x$ folgt $f''(x) = 1/x$ und das Maximum von f'' im Intervall $[2, 5]$ ist bei $x = 2$. Um die Abschätzung mit dem numerisch erhaltenen Resultat vergleichen zu können, prüfen wir in beiden Fällen den relativen Fehler. Mit der groben Schrittweite $h = 3$ ergibt sich für das Integral mit der Trapezregel der Näherungswert 14. Nach Gleichung (7.5$*$) berechnet sich die Schrittweite damit aus

$$\frac{(b-a)h^2}{12} \frac{1}{2} < 14 \times 10^{-8} \quad \Rightarrow h < \sqrt{112} \times 10^{-4} = 1.058 \times 10^{-3}$$

Es werden somit ca. 3'000 Integrationsschritte benötigt.

Mit dem Programm *integral* und Algorithmus 7.1 erhält man die aufeinanderfolgenden Näherungswerte:

Anzahl Intervalle	$T(h_i)$
2	13.652118527
4	13.524540402
8	13.492411591
16	13.484363630
32	13.482350633
64	13.481847320
128	13.481721488
256	13.481690029
512	13.481682164
1024	13.481680195
2048	13.481679700
4096	13.481679568

Integral = 13.481679568 Anzahl Fkt.Aufr. 4097

Es sind also für die verlangte Genauigkeit nur ca. 2000 Integrationsschritte nötig. Um das festzustellen muss beim Algorithmus 7.1 die Schrittweite nochmals halbiert werden, daher erhalten wir 4097 Funktionsaufrufe. Wir haben also gute Übereinstimmung mit der Abschätzung.

Aufgabe 7.3 *Integrale über eine volle Periode bei stetig differenzierbaren periodischen Funktionen (welche bei den Fourierreihen vorkommen können) werden am besten mit der Trapezregel integriert. Um das zu illustrieren berechne man das Integral* $\int_0^T \sqrt{1+\cos^2 x}\,dx$ *für* $T = \pi$ *(volle Periode) und* $T = 4$ *mittels des Algorithmus trapez, drucke aufeinanderfolgende Näherungen aus und beobachte die Konvergenz.*

Mit dem Programm *integral* erhalten wir für *eps* = 1E−8 und der Trapezregel die Werte in Tabelle 7.1 (in der ersten Kolonne ist die Anzahl Teilintervalle angegeben).

Es ist beeindruckend, wie gut die Trapezregel für die Integration über die ganze Periode funktioniert. Um dieses Verhalten zu erklären ist eine genauere Untersuchung des Diskretisationsfehlers notwendig. Einen Hinweis dafür erhält man bei der Betrachtung der Euler'schen Summenformel (7.32∗): alle Koeffizienten c_i verschwinden und der Fehler ist für jedes h nur im Restglied der asymptotischen Entwicklung vorhanden.

a=0 und b=4		a=0 und b=π	
	$T(h_i)$		$T(h_i)$
2	4.7751562534	2	3.7922377964
4	4.9313625917	4	3.8199436437
8	4.9578449249	8	3.8201977159
16	4.9644503391	16	3.8201977896
32	4.9660761329	32	3.8201977896
64	4.9664810176	Integral = 3.8201977896	
128	4.9665821412	Anzahl Fkt.Aufr. 33	
256	4.9666074158		
512	4.9666137337		
1024	4.9666153124		
2048	4.9666157056		
4096	4.9666158010		
8192	4.9666158189		
Integral = 4.9666158189			
Anzahl Fkt.Aufr. 8193			

Tabelle 7.1: Aufgabe 7.3

Aufgabe 7.4 *Man beweise die Fehlerabschätzung* (7.21∗) *für die Simpsonregel.*

Wir betrachten zunächst die Simpsonregel für nur 2 Intervalle der Länge $h = x_2 - x_1 = x_1 - x_0$:

$$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{h}{3}\,(y_0 + 4y_1 + y_2).$$

Ähnlich wie beim entsprechenden Beweis für die Trapezregel bilden wir die Funktion

$$F(z) = \int_{x_1-z}^{x_1+z} f(x)\,dx - \frac{z}{3}\,(f(x_1 - z) + 4f(x_1) + f(x_1 + z))\,.$$

Es ist $F(h)$ der gesuchte Fehler. Bildet man die Ableitungen von F, so ergibt sich

$$F'(z) = -\frac{4}{3}f(x_1) + \frac{2}{3}\,(f(x_1 + z) + f(x_1 - z))$$

$$-\frac{z}{3}\,(f'(x_1 + z) - f'(x_1 - z))$$

$$F''(z) = \frac{1}{3}f'(x_1+z) - \frac{1}{3}f'(x_1-z) - \frac{z}{3}\left(f''(x_1+z) + f''(x_1-z)\right)$$

$$F'''(z) = -\frac{z}{3}\left(f'''(x_1+z) - f'''(x_1-z)\right)$$

$$= -\frac{2z^2}{3}\Bigg(\underbrace{\frac{f'''(x_1+z) - f'''(x_1-z)}{2z}}_{f^{(4)}(\xi)}\Bigg)$$

wobei für die letzte Umformung der Mittelwertsatz der Differentialrechnung benützt wurde. Es gilt somit

$$F'''(z) = -\frac{2z^2}{3}\, f^{(4)}(\xi) \quad \text{mit} \quad \xi \in [x_1 - z, x_1 + z] \subset [x_0, x_2]. \tag{7.1}$$

Man bemerkt ferner, dass $F(0) = F'(0) = F''(0) = 0$ ist. Daher ist zunächst

$$F''(z) = F''(z) - F''(0) = \int_0^z F'''(t)\, dt = -\frac{2}{3}\int_0^z t^2 f^{(4)}(\xi_t)\, dt, \tag{7.2}$$

wobei wir den Ausdruck (7.1) im Integranden verwendet haben. Nach dem Mittelwertsatz der Integralrechnung lässt sich mit einem gewissen $\xi_1 \in [x_0, x_2]$ das Integral (7.2) umformen zu

$$F''(z) = -\frac{2}{3}f^{(4)}(\xi_1)\int_0^z t^2 dt = -\frac{2}{9}z^3 f^{(4)}(\xi_1). \tag{7.3}$$

Unter Verwendung von $F'(0) = 0$ machen wir nun mit Gleichung (7.3) dieselbe Überlegung:

$$F'(z) = \int_0^z F''(t)\, dt = -\frac{2}{9}\int_0^z t^3 f^{(4)}(\xi_t)$$

$$= -\frac{2}{9}f^{(4)}(\xi_1)\int_0^z t^3\, dt = -\frac{1}{18}z^4 f^{(4)}(\xi_1) \tag{7.4}$$

Dabei haben wir wiederum den Mittelwertsatz angewendet und das neue ξ_1 der Einfachheit halber gleich bezeichnet. Schliesslich folgt aus $F(0) = 0$ mit nochmaliger Anwendung des Mittelwertsatzes

$$F(z) = \int_0^z -\frac{1}{18}t^4 f^{(4)}(\xi_t)\, dt = -\frac{1}{90}z^5 f^{(4)}(\xi_1)\, dt \tag{7.5}$$

d.h. man erhält für $z = h$ und $\xi_1 \in [x_0, x_2]$ die gesuchte Fehlerabschätzung für 2 Intervalle.

Für den gesamten Fehler müssen alle Fehler über jeweils 2 Intervalle addiert werden. Mit $h = (b-a)/(2n)$ ist

$$\int_a^b f(x)\,dx - S(h) = -\sum_{k=1}^{n} \frac{h^5}{90} f^{(4)}(\xi_k), \tag{7.6}$$

wobei ξ_k im Intervall $[x_{2k-2}, x_{2k}]$ liegt. Wenn $f^{(4)}$ stetig ist, wird der Mittelwert von Funktionswerten angenommen, d.h. es gibt ein $\xi \in [a, b]$ mit

$$\frac{1}{n}\sum_{k=1}^{n} f^{(4)}(\xi_k) = f^{(4)}(\xi).$$

Benützen wir diese Tatsache in Gleichung (7.6) und beachten, dass $2nh = b - a$ ist, so folgt

$$\int_a^b f(x)\,dx - S(h) = -\frac{h^5}{90} n f^{(4)}(\xi) = -\frac{(b-a)h^4}{180} f^{(4)}(\xi)$$

was zu beweisen war.

Aufgabe 7.5 *Man berechne das Integral*

$$I = \int_0^2 (x^3 + 2x^2 + 1)dx$$

a) analytisch b) mittels Simpsonregel ($h = 1$).

Wie gross ist bei b) der Integrationsfehler ? Welche Regel lässt sich daraus ableiten und warum ?

Es ist einerseits

$$\int_0^2 (x^3 + 2x^2 + 1)\,dx = \left[\frac{x^4}{4} + 2\frac{x^3}{3} + x\right]_0^2 = \frac{34}{3},$$

andrerseits erhält man mit $f(x) = x^3 + 2x^2 + 1$

$$S(1) = \frac{1}{3}(f(0) + 4f(1) + f(2)) = \frac{1}{3}(1 + 4 \cdot 4 + 17) = \frac{34}{3}$$

dasselbe Resultat! Bei Betrachtung der Fehlerabschätzung (7.21$*$) kann man dies leicht erklären. Die vierte Ableitung eines Polynoms 3. Grades verschwindet und daher integriert die Simpsonformel nicht nur Polynome 2. Grades exakt, für welche sie konstruiert wurde, sondern auch dritten Grades.

Aufgabe 7.6 *Wie gross muss die Integrationsschrittweite h sein, damit das Integral*

$$\int_0^{100} \frac{dx}{(1+x)^2}$$

mittels der Simpsonregel bis auf einen Fehler kleiner als 10^{-6} berechnet werden kann ? Man schätze h mittels (7.21$*$) ab und prüfe es durch Integration mit dem Programm simpson nach.

Mit $f(x) = 1/(1+x)^2$ ist

$$f^{(4)}(x) = \frac{120}{(1+x)^6} \quad \text{und} \quad \max_{x\in[0,100]} |f^{(4)}(x)| = f^{(4)}(0) = 120.$$

Damit ergibt sich für die Integrationsschrittweite

$$\frac{100}{180} h^4 120 < 10^{-6} \quad \Rightarrow h^4 < 1.5\text{E}{-8} \quad \Rightarrow h < 0.0110668.$$

Die Abschätzung ergibt also ca. 9'000 Integrationsschritte.

Andrerseits liefert das Programm *integral* mit *eps* = 1E−6 und Algorithmus 7.2 die Werte

Anzahl Intervalle	$T(h_i)$
4	8.3956387097
8	4.2930041238
16	2.3177580107
32	1.4347968445
64	1.1017007812
128	1.0090724153
256	0.99222480787
512	0.99027110363
1024	0.99011074454
2048	0.99009976089
4096	0.99009905607

Integral = 0.99009905607 Anzahl Fkt.Aufr. 4097

Die Fehlerabschätzung ist hier zu pessimistisch, weil die 4. Ableitung in diesem grossen Intervall sehr verschiedene Werte annimmt.

Aufgabe 7.7 *Man berechne mittels Programm simpson das elliptische Integral*

$$A(k,\phi) = \int_0^{\phi} \frac{dx}{\sqrt{1-k^2\sin^2 x}}$$

für $k = 0.4$ *und* $\phi = \dfrac{\pi}{2}$.

Mit dem Programm *integral* und der Simpsonmethode erhält man für $a = 0$ und $b = \pi/2$ den Output:

Anzahl Intervalle	$S(h_i)$
4	1.6399987892
8	1.6399998661
16	1.6399998661

Integral = 1.6399998661 Anzahl Fkt.Aufr. 17

Es werden hier sehr wenig Funktionsauswertungen gebraucht. Noch schneller geht es mit der Trapezregel bei diesem Beispiel, da wir wiederum eine periodische Funktion über eine volle Periode integrieren.

Aufgabe 7.8 *Man beweise, dass in der zweiten Kolonne des Rombergschemas die entsprechenden Werte der Simpsonregel stehen.*

Wir betrachten zwei aufeinanderfolgende Trapezsummen für $2h$ und h:

$$T(2h) = 2h\left(\frac{1}{2}y_0 + y_2 + y_4 + \cdots + y_{n-2} + \frac{1}{2}y_n\right) \tag{7.7}$$

$$T(h) = h\left(\frac{1}{2}y_0 + y_1 + y_2 + y_3 + \cdots + y_{n-2} + y_{n-1} + \frac{1}{2}y_n\right) \tag{7.8}$$

Bildet man die Linearkombination $4 \times (7.8) - (7.7)$ so wird

$$4 \times T(h) - T(2h) = h(y_0 + 4y_1 + 2y_2 + 4y_3 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n)$$

und nach der Division durch 3 erhält man $S(h)$.

Aufgabe 7.9 *Man berechne mit dem Algorithmus romberg das Integral*

$$I = \int_2^5 \frac{e^x}{x} dx$$

und drucke das ganze Rombergschema aus.

Wir verwenden das Programm *integral* und erhalten für $eps = 10^{-6}$ eine schöne Konvergenzbeschleunigung

```
50.065739804
39.225206456
36.250318915  35.258689735
35.487212151  35.232843230  35.231120130
35.295169204  35.231154888  35.231042332  35.231041097
35.247078404  35.231048137  35.231041021  35.231041000  35.231041000
Integral = 3.5231041000     Anzahl Fkt.Aufr. 33
```

Aufgabe 7.10 *Man berechne mit dem Verfahren von Romberg das Integral*

$$I = \int_0^\pi \frac{dx}{1 + \sin^2 x}.$$

Man drucke das ganze Rombergschema und beobachte, dass die Trapezwerte am schnellsten konvergieren. Warum ?

Wiederum liefert das Programm *integral* für $eps = 10^{-4}$ die Werte

```
3.141592654
2.356194491
2.225294797  2.181661565
2.221444806  2.220161475  2.222728136
2.221441469  2.221440357  2.221525616  2.221506529
2.221441469  2.221441469  2.221441544  2.221440209  2.221439949
Integral = 2.2214399490     Anzahl Fkt.Aufr. 33
```

Da wir eine periodische Funktion über eine volle Periode integrieren, konvergieren die Trapezwerte in der ersten Kolonne am schnellsten.

Aufgabe 7.11 *Man berechne mittels adapt* $\int_0^\pi \sqrt{1+\cos^2 x}\,dx$ *und vergleiche die Anzahl Funktionsauswertungen mit derjenigen von trapez (Integration einer periodischen Funktion !).*

Wenn man für *eps* = 1E−8 mittels Programm *integral* und *adapt* das Integral berechnet, so erhält man den Wert 3.8201977890. Dafür werden 121 Funktionsaufrufe benötigt. Das gleiche Integral mit der Trapezregel berechnet (s. Aufgabe 7.3) liefert 3.8201977896 mit nur 33 Funktionsauswertungen.

Aufgabe 7.12 *Wenn ein Integrand einen Pol im Integrationsintervall hat und das Integral existiert, kann man durch* Ignorieren des Pols *das Integral berechnen. Als Beispiel berechne man*

$$I = \int_0^1 \frac{e^x}{\sqrt{x}}\,dx$$

indem für $x = 0$ *der Funktionswert des Integranden einfach* 0 *gesetzt wird.*

Verwendet man als Funktion f wie vorgeschlagen

$$\textbf{if } x = 0 \textbf{ then } f := 0 \textbf{ else } f := exp(x)/sqrt(x),$$

so erhält man mittels Programm *integral* und *adapt* für verschiedene Genauigkeiten *eps* folgende Tabelle:

eps	Integral	Anzahl Fkt. Auswertungen
10^{-7}	2.9253031003	365
10^{-8}	2.9253034430	553
10^{-9}	2.9253034857	833
10^{-10}	2.9253034914	1441
10^{-11}	2.9253034917	2185

Beim Pol muss die Integrationschrittweite sehr klein gewählt werden und kann dann bis $x = 1$ zunehmen. *adapt* wählt z.B. für $eps = 10^{-11}$ Schrittweiten von 6.7×10^{-21} bis 1.56×10^{-5}. Dabei werden bei diesem Beispiel auch die Teilintegrale grösser, und weil beim Aufsummieren mit den kleinsten begonnen wird, bleiben die Rundungsfehler trotz den 2185 Funktionsauswertungen beschränkt.

Bei Polen im Integrationsbereich sollte man versuchen, diese durch eine geeignete Transformation der Integrationsvariabeln zu eliminieren. Dies ist beim vorliegenden Integral durch folgende Substitution leicht möglich:

$$\sqrt{x} = t \quad \Rightarrow 2\,dt = \frac{dx}{\sqrt{x}} \quad \text{und} \quad \int_0^1 \frac{e^x}{\sqrt{x}}\,dx = \int_0^1 2e^{t^2}\,dt.$$

Das neue Integral hat keinen Pol mehr und man erhält mit *adapt* die Tabelle

eps	Integral	Anzahl Fkt. Auswertungen
10^{-7}	2.9253034921	53
10^{-8}	2.9253034918	89
10^{-9}	2.9253034918	133
10^{-10}	2.9253034918	209
10^{-11}	2.9253034918	361

Man sieht, dass schon für $eps = 10^{-8}$ mit nur 89 Funktionsauswertungen die volle Maschinengenauigkeit erreicht wird.

Aufgabe 7.13 *Man kann mittels adapt auch unstetige Funktionen integrieren. Als Beispiel dazu berechne man* $\int_0^5 f(x)dx$*, wobei* $f(x)$ *durch Figur* 7.2∗ *gegeben ist.*

Figur 7.2∗: unstetige Funktion

Mittels des Programms *integral* und *adapt* erhalten wir die Werte von Tabelle 7.2. Man beachte, wie klein die Schrittweite bei $x = 1$ (Spitze) und $x = 3$ (Unstetigkeit) gewählt wird.

Aufgabe 7.14 *Man löse die Gleichung*

$$f(x) = \int_0^1 e^{xt^2}\, dt - 2 = 0$$

mit dem Newtonverfahren. Man berechne f *und* f' *mittels adapt.*

Intervallanfang	Schrittweite	Teilintegral
0.00000	6.2500000000E−01	8.2031250000E−01
0.62500	3.1250000000E−01	5.5664062500E−01
0.93750	3.9062500000E−02	7.6446533203E−02
0.97656	1.9531250000E−02	3.8795471191E−02
0.99609	9.7656250000E−03	1.9505606757E−02
1.00586	9.7656250000E−03	1.9426345825E−02
1.01563	7.8125000000E−02	1.5197753906E−01
1.09375	1.5625000000E−01	2.8564453125E−01
1.25000	1.2500000000E+00	1.4062500000E+00
2.50000	3.1250000000E−01	1.0742187500E−01
2.81250	1.5625000000E−01	1.7089843750E−02
2.96875	1.9531250000E−02	4.1961669922E−04
2.98828	9.7656250000E−03	6.6757202148E−05
2.99805	1.2207031250E−03	1.6391277313E−06
2.99927	6.1035156250E−04	2.6077032089E−07
2.99988	7.6293945313E−05	6.4028427005E−09
2.99995	3.8146972656E−05	1.0186340660E−09
2.99999	3.8146972656E−05	7.0359994425E−05
3.00003	1.5258789063E−04	3.0517578125E−04
3.00018	3.0517578125E−04	6.1035156250E−04
3.00049	2.4414062500E−03	4.8828125000E−03
3.00293	4.8828125000E−03	9.7656250000E−03
3.00781	3.9062500000E−02	7.8125000000E−02
3.04688	7.8125000000E−02	1.5625000000E−01
3.12500	6.2500000000E−01	1.2500000000E+00
3.75000	1.2500000000E+00	2.5000000000E+00
Integral =	7.5000084771	Anzahl Fkt.Aufr. 105

Tabelle 7.2: Aufgabe 7.13

In TURBO PASCAL macht diese Aufgabe Mühe, weil man *keine Prozeduren als Parameter* übergeben kann. Für die Berechnung der beiden Funktionen

$$f(x) = \int_0^1 e^{xt^2}\,dt - 2 \quad \text{und} \quad f'(x) = \int_0^1 t^2 e^{xt^2}\,dt$$

sollte im gleichen Programm *adapt* mit *verschiedenen* Funktionsnamen aufgerufen werden. Dies ist aber nicht möglich, wenn immer der globale Name f verwendet wird.

Wir betrachten im folgenden zwei Lösungen, welche den Namenskonflikt lösen, ohne dass *adapt* geändert werden muss. Die erste Lösung verwendet

dieselbe Idee wie das Programm *integral*: mittels der *integer* Variable *abl* wird in einer **case**-Anweisung in f für $abl = 0$ die Funktion f und für $abl = 1$ die Ableitung f' berechnet. Die Integrationsvariable ist t, daher wird der Parameter x als globale Variable gewählt und der formale Parameter in der Deklaration von f ist t.

Für die Lösung x ist der Wert der Integrals 2, dadurch ist ein guter Schätzwert bekannt und wir verwenden deshalb für $is = eps/1\text{E}{-}11$, wobei *eps* gleichzeitig auch die gewünschte Genauigkeit für die Newtoniteration bedeutet. Für die Ableitung können wir denselben Schätzwert verwenden, weil sie auch von der Grössenordnung 1 ist. Damit erhalten wir folgendes Programm, bei dem *adapt* als Include-File dazugeladen wird:

```
(*$B- Aufgabe 7.14A*)
program newton;
var xn,x,eps,is,fa,fb,fm : real; abl : integer; a : array [0..1] of real;

function f(t : real) : real;
begin
   case abl of
      0 : f := exp(x * sqr(t));
      1 : f := sqr(t) * exp(x * sqr(t))
   end
end ;

(*$I A7.4 function adapt *)

begin
   writeln('Startwert eingeben'); read(xn);
   writeln('eps =?'); read(eps); is := eps/1e-11;
   repeat
      x := xn;
      for abl := 0 to 1 do
      begin
         fa := f(0); fm := f(0.5); fb := f(1);
         a[abl] := adapt(0, 1, fa, fm, fb, is);
         if abl = 0 then write('x=',x,'f(x) = ', a[0] - 2)
         else writeln(' f''(x) =', a[1]);
      end ;
      xn := x - (a[0] - 2)/a[1];
   until abs(xn - x) < eps * abs(x);
   writeln('Loesung =', xn);
end.
```

Für zwei verschiedene Genauigkeiten *eps* erhält man mit diesem Programm

die folgenden Näherungswerte der Newtoniteration:

Startwert 1, $eps = 1\text{E}{-6}$

x_k	$f(x_k)$	$f'(x_k)$
1.0000000000	−5.3734825245E−01	6.2781504265E−01
1.8559021622	1.9249613625E−01	1.1328644270E+00
1.6859823274	1.1161897128E−02	1.0043369809E+00
1.6748686302	4.3550193368E−05	9.9651108114E−01
1.6748249275	6.6938810050E−10	9.9648044094E−01

Loesung = 1.6748249268

Startwert 1, $eps = 1\text{E}{-10}$

x_k	$f(x_k)$	$f'(x_k)$
1.0000000000	−5.3734825410E−01	6.2781504127E−01
1.8559021667	1.9249613875E−01	1.1328644275E+00
1.6859823298	1.1161897783E−02	1.0043369798E+00
1.6748686319	4.3550211558E−05	9.9651107970E−01
1.6748249292	6.6938810050E−10	9.9648043949E−01
1.6748249285	0.0000000000E+00	9.9648043902E−01

Loesung = 1.6748249285

Man sieht, dass 9 richtige Dezimalstellen schon für $eps = 1\text{E}{-6}$ berechnet werden.

Eine zweite Lösung erhalten wir, indem wir für f und f' zwei Funktionsprozeduren g und gs deklarieren. Sowohl g als auch gs enthalten als lokale Funktionen die **function** *adapt* und unter dem Funktionsnamen f die Funktion respektive ihre Ableitung. Wir erhalten mit dem nachfolgenden Programm dieselben numerischen Werte wie die oben angegebenen. Unschön ist die zweimalige Deklaration von *adapt*. Man kann aber diese **function** wieder als Include-File an beiden Stellen dazuladen.

```
(*$B− Aufgabe 7.14B*)
program newton;
var xn,x,eps,fa,fb,fm,is : real;

function g(x : real) : real;

function f(t : real) : real;
begin f := exp(x * sqr(t)) end ;
(*$I A7.4 function adapt *)
begin
   fa := f(0); fb := f(1); fm := f(0.5);
   g := adapt(0, 1, fa, fm, fb, is) − 2;
end ;
```

```
function gs(x : real) : real;
function f(t : real) : real;
begin f := sqr(t) * exp(x * sqr(t)) end ;
(*$I A7.4 function adapt *)
begin
   fa := f(0); fb := f(1); fm := f(0.5);
   gs := adapt(0, 1, fa, fm, fb, is);
end ;
begin
   writeln('Startwert eingeben'); read(xn);
   writeln('eps = ?'); read(eps); is := eps/1e-11;
   repeat
      x := xn; xn := x - g(x)/gs(x); writeln(xn);
   until abs(xn - x) < eps * abs(x);
   writeln('x =', xn,'  f(x) =', g(xn))
end.
```

Aufgabe 7.15 *Berechnung von uneigentlichen Integralen. Wenn das Integrationsintervall unendlich ist, kann man durch eine geeignete Variabelnsubstitution, etwa*

$$t = \frac{1}{x}, \quad t = \frac{1}{x+1} \quad \text{oder} \quad t = e^{-x}$$

das Integral auf eines mit endlichen Grenzen zurückführen. Treten im Integrationsintervall Pole auf, so sind diese zu ignorieren, d.h. man setzt dort den Funktionswert einfach gleich null.
Man berechne mittels dieser Technik die Integrale

$$a) \quad \int_0^\infty \frac{x}{e^x+1}\,dx \qquad b) \quad \int_0^\infty \frac{\arctan x}{(x+1)^2}\,dx$$

$$c) \quad \int_0^\infty \frac{x^3+1}{1+x^2+x^5}\,dx \quad d) \quad \int_1^\infty \frac{1}{x} e^{-\frac{x}{2}}\,dx$$

a) Mit der Substitution $t = e^{-x}$ transformiert sich das erste Integral in

$$\int_0^\infty \frac{x}{e^x+1}\,dx = -\int_0^1 \frac{\ln t}{1+t}\,dt \tag{7.9}$$

und der Integrand wird durch Ignorieren des Pols wie folgt programmiert

$$\textbf{if } x = 0 \textbf{ then } f := 0 \textbf{ else } f := -ln(x)/(1+x)$$

b) Hier verwenden wir die Substitution $t = 1/(x+1)$ und erhalten für das zweite Integral

$$\int_0^\infty \frac{\arctan x}{(x+1)^2}\, dx = \int_0^1 \arctan\left(\frac{1}{t} - 1\right) dt. \tag{7.10}$$

Auch hier programmieren wir den Integranden, indem der Pol ignoriert wird.:

$$\textbf{if } x = 0 \textbf{ then } f := 0 \textbf{ else } f := arctan(1/x - 1)$$

c) Bei diesem Integral ist auch die Substitution $t = 1/(x+1)$ geeignet und nach dem Erweitern des Bruchs enthält der neue Integrand keinen Pol:

$$\int_0^\infty \frac{x^3+1}{1+x^2+x^5}\, dx = \int_0^1 \frac{(1-t)^3+t^3}{t^5+(1-t)^2t^3+(1-t)^5}\, dx \tag{7.11}$$

Mit den Hilfsvariabeln $h1$, $h2$ und $h3$ lautet der Integrand hier:

```
begin
  h1 := 1 - x; h2 := sqr(h1); h3 := h1 * h2;
  f := (h3 + x * x * x)/(sqr(sqr(x)) * x + h2 * x * x * x + h2 * h3)
end ;
```

d) Für das letzte Integral ist $t = 1/x$ eine mögliche Substitution:

$$\int_1^\infty \frac{1}{x} e^{-\frac{x}{2}}\, dx = \int_0^1 \frac{1}{t} e^{-\frac{1}{2t}}\, dt \tag{7.12}$$

Hier muss der Pol wieder ignoriert werden und man muss wegen des Unterlaufs der Exponentialfunktion aufpassen, der im TURBO PASCAL als Fehlermeldung angezeigt wird:

$$\textbf{if } (1/160 > x) \textbf{ then } f := 0 \textbf{ else } f := exp(-1/2/x)/x$$

Für diese Aufgabe können wir das Programm *integral* übernehmen. Einzig das Include-File *f.pas* ändert, oder müsste ergänzt werden. Wir geben trotzdem ein neues Programm an, weil wir nur die beiden Prozeduren *simpson* und *adapt* benützen und vergleichen wollen. Das File *F* 7.15 enthält die 4 Integranden und wird als Include-File in die **function** *f* zugeladen und von der Procedure *menue* als Text File gelesen. Das Hauptprogramm lautet:

```
(*$B-,U+ Aufgabe 7.15*)
program uneigentlich;
var a,b,eps,int :real; z,i :integer; fall : char;
    tf : text; stri : string [20];
function f(x :real) :real;
var h1, h2, h3 : real;
begin
    case fall of
        (*$IF 7.15*)
    end ;
    z := z + 1;
end ;
(*$I simpson Algorithmus 7.2*)
function ada(a, b, eps :real) :real;
var fa,fb,fm, is :real; v :integer;
(*$I adapt Algorithmus 7.4 *)
begin
    fa := f(a); fm := f((a + b)/2); fb := f(b);
    is := (b − a)/6 ∗ (fa + 4 ∗ fm + fb) ;
    v := 1; if is < 0 then v := −1;
    is := v ∗ (abs(is) + b − a)/2 ∗ eps/1e−11;
    writeln(tf,'a': 5,'b-a': 15,'i1': 16);
    ada:= adapt(a, b, fa, fm, fb, is);
end ;
procedure menue;
var f :text; stri : string [120];
begin
    assign(f,'F 7.15 '); reset(f);
    while not eof(f) do
    begin
        readln(f,stri); writeln(stri)
    end
end ;
begin
    writeln('wohin mit der Ausgabe ?');
    readln(stri); assign(tf,stri); rewrite(tf);
    menue;
    writeln('Fall eingeben : 1, 2, 3, 4 oder a,b,c,d'); read(fall);
    writeln('Grenzen a,b eingeben'); read(a, b);
    writeln('eps?');read(eps);
    writeln('Integrationsmethode : (1) = adapt, (2) =simpson');
    read(i); z := 0;
```

```
    case i of
        1 : int:= ada(a, b,eps);
        2 : int:= simpson(a, b,eps);
    end ;
    writeln(tf,'Integral =',int,'  Anz.Fkt.Aufr.  ',z);
    close(tf);
end.
```

Für die Aufgabe a) erhält man mit *adapt* die Resultate:

eps	Integral	Anzahl Fkt.Ausw.
1E−5	8.2245814733E−01	93
1E−6	8.2246638286E−01	149
1E−8	8.2246702093E−01	321
1E−10	8.2246703336E−01	849

Mit *simpson* kommt man bei diesem Integral mit fester Schrittweite nirgends hin: bei 16'385 Funktionsaufrufen weist der Näherungswert 0.822218 erst etwa 3 richtige Dezimalstellen auf.

Beim Integral (7.10) von Aufgabe b) erhält man mittels *adapt* die Werte

eps	Integral	Anzahl Fkt.Ausw.
1E−5	7.8539066829E−01	73
1E−6	7.8539723097E−01	89
1E−8	7.8539815611E−01	177
1E−10	7.8539816334E−01	337
1E−12	7.8539816339E−01	613

Auch hier versagt der Simpsonalgorithmus mit fester Schrittweite: bei 16'385 Schritte hat man erst den 4-stelligen Wert 7.8536620329E−01.

Beim Integral (7.11) von Aufgabe c) arbeiten beide Algorithmen etwa gleich. Man erhält mit *adapt*:

eps	Integral	Anzahl Fkt.Ausw.
1E−5	1.7866280528	41
1E−6	1.7866313851	53
1E−8	1.7866314569	161
1E−10	1.7866314571	377

und mit *simpson*

eps	Integral	Anzahl Fkt.Ausw.
1E−5	1.7866312992	33
1E−6	1.7866314472	65
1E−8	1.7866314565	129
1E−10	1.7866314570	513

Bei Aufgabe d) schliesslich ist *simpson* überlegen. Man erhält für das Integral (7.12) mittels *adapt*

eps	Integral	Anzahl Fkt.Ausw.
1E−5	5.5977355195E−01	41
1E−6	5.5977359113E−01	61
1E−8	5.5977359719E−01	149
1E−10	5.5977359477E−01	389
1E−12	5.5977359477E−01	913

Der Algorithmus *simpson* benötigt aber deutlich weniger Funktionsaufrufe:

eps	Integral	Anzahl Fkt.Ausw.
1E−5	5.5977359479E−01	129
1E−6	5.5977359479E−01	129
1E−8	5.5977359476E−01	257
1E−10	5.5977359476E−01	257

Aufgabe 7.16 *Man führe die folgenden linearen Differentialgleichungen*

a) $y^{(4)} + 1.1y''' - 0.1y'' + y' - 0.3y = \sin x + 5$
$y(0) = y''(0) = y'''(0) = 0, \quad y'(0) = 2$

b) $x^2y'' + xy' + (x^2 - n^2)y = 0$
$y(1) = y'(1) = 2$
(n ist ein Parameter)

auf ein System erster Ordnung zurück.

a) Mit $z_i = y^{(i-1)}$, $i = 1, \ldots, 4$ lautet das System in Matrixschreibweise

$$\mathbf{z}' = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0.3 & -1 & 0.1 & -1.1 \end{pmatrix} \mathbf{z} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \sin x + 5 \end{pmatrix} \quad \text{mit} \quad \mathbf{z}(0) = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 0 \end{pmatrix}$$

b) Mit $z_1 = y$ und $z_2 = y'$ ist hier

$$\mathbf{z}' = \begin{pmatrix} 0 & 1 \\ \left(\frac{n}{x}\right)^2 - 1 & -\frac{1}{x} \end{pmatrix} \mathbf{z} + \quad \text{mit} \quad \mathbf{z}(1) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

In beiden Fällen ist die Differentialgleichung linear, bei Aufgabe b) mit nichtkonstanten Koeffizienten.

Aufgabe 7.17 *Man führe die Differentialgleichung*

$$y'' - \frac{1 - y^2}{10} y' + y = 0 \quad \textit{Van der Pol Dgl.}$$

mit der Anfangsbedingung $y(0) = 1$ *und* $y'(0) = 0$ *auf ein System erster Ordnung zurück.*

Mit $z_1 = y$ und $z_2 = y'$ lautet das nichtlineare System

$$\begin{aligned} z_1' &= z_2 \\ z_2' &= -z_1 + \frac{1 - z_1^2}{10} z_2 \end{aligned} \quad \text{mit} \quad \mathbf{z}(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Aufgabe 7.18 *Man führe das folgende System von Differentialgleichungen für die gesuchten Funktionen* $r(t)$ *und* $\Theta(t)$

$$r'' - r\Theta^2 = -\frac{2}{r^2}$$

$$r\Theta'' + 2r'\Theta' = 0$$

auf ein System erster Ordnung zurück:

Mit $z_1 = r$, $z_2 = r'$, $z_3 = \Theta$ und $z_4 = \Theta'$ erhält man das nichtlineare System

$$\begin{aligned} z_1' &= z_2 \\ z_2' &= -\frac{2}{z_1^2} + z_1 z_3^2 \\ z_3' &= z_4 \\ z_4' &= -\frac{2 z_2 z_4}{z_1} \end{aligned}$$

Aufgabe 7.19 *Die Differentialgleichung des Pendels lautet*

$$\varphi'' + \frac{g}{l}\sin\varphi = 0$$

dabei ist φ der Auslenkwinkel, g die Erdbeschleunigung und l die Fadenlänge. In der Physik ersetzt man für 'kleine Schwingungen' den $\sin\varphi$ durch φ:

$$\varphi'' + \frac{g}{l}\varphi = 0.$$

Wenn man den Luftwiderstand berücksichtigt (proportional der Geschwindigkeit φ'), so ergibt sich

$$\varphi'' + a\varphi' + \frac{g}{l}\sin\varphi = 0$$

Man führe alle drei Modelle auf Systeme erster Ordnung zurück.

Mit den neuen Funktionen $z_1 = \varphi$ und $z_2 = \varphi'$ erhält man das nichtlineare System

$$z_1' = z_2$$

$$z_2' = -\frac{g}{l}\sin z_1$$

für die Pendelgleichung. Für 'kleine Schwingungen' wird das System linear:

$$\mathbf{z}' = \begin{pmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{pmatrix} \mathbf{z}$$

Mit Berücksichtigung des Lufwiderstands erhält man das nichtlineare System

$$z_1' = z_2$$

$$z_2' = -\frac{g}{l}\sin z_1 - az_2.$$

Aufgabe 7.20 *Gegeben ist die Differentialgleichung*

$$y'(x) = 1 + xy^2(x), \quad y(0) = -8$$

Man berechne $y(0.2)$ durch Integration nach Heun. Man führe 2 Schritte mit $h = 0.1$ durch.

Es ist

$$\begin{aligned} y^* &= y_0 + hf(x_0, y_0) = -8 + 0.1 = -7.9 \\ y_1 &= y_0 + \frac{h}{2}(f(x_0, y_0) + f(x_1, y^*)) = -8 + 0.05\,(1 + 7.214) = -7.58795 \\ y^* &= -7.58795 + 0.1 \times 6.75769852 = -6.912180148 \\ y_2 &= -7.58795 + 0.05\,(6.75769852 + 10.55564688) = -6.72228273 \approx y(0.2) \end{aligned}$$

Aufgabe 7.21 *Man integriere die Differentialgleichung*

$$y'' - y' + 4x^2 y = 0 \quad \textit{mit} \quad y(1) = 1,\ y'(1) = -1$$

nach der Methode von Euler-Cauchy. Man führe einen Schritt der Länge $h = 0.2$ durch.

Mit $z_1 = y$ und $z_2 = y'$ lautet das System

$$\mathbf{z}' = \begin{pmatrix} 0 & 1 \\ -4x^2 & 1 \end{pmatrix} \mathbf{z} \quad \text{mit} \quad \mathbf{z}(1) = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

und ein Euler-Schritt berechnet sich damit

$$\mathbf{z}(1.2) \approx \begin{pmatrix} 1 \\ -1 \end{pmatrix} + 0.2 \begin{pmatrix} 0 & 1 \\ -4 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0.8 \\ -2 \end{pmatrix}$$

Also ist $y(1.2) \approx 0.8$ und $y'(1.2) \approx -2$.

Aufgabe 7.22 *Man schreibe ein Programm, um Systeme von Differentialgleichungen erster Ordnung nach der Methode von Heun zu integrieren. Als Testbeispiel wähle man die Differentialgleichung*

$$xy'' - y' + 4x^3 y = 0, \tag{7.13}$$

welche die allgemeine Lösung

$$y(x) = A \sin(x^2) + B \cos(x^2)$$

mit beliebigen Integrationskonstanten A und B hat. Man integriere die Differentialgleichung unter der Anfangsbedingung

$$y(1) = 1, \quad y'(1) = 0 \tag{7.14}$$

für x im Intervall $[1, 5]$.

Wir berechnen zuerst die exakte spezielle Lösung, welche durch die Anfangsbedingungen (7.14) bestimmt ist.

$$\begin{array}{l} y(x) = A\sin x^2 + B\cos x^2 \\ y'(x) = 2Ax\cos x^2 - 2Bx\sin x^2 \end{array} \quad \Rightarrow \quad \begin{array}{r} y(1) = A\sin 1 + B\cos 1 = 1 \\ y'(1) = 2A\cos 1 - 2B\sin 1 = 0 \end{array} \tag{7.15}$$

Dividiert man die 2. Gleichung von (7.15) durch 2, so kann man die Lösung ablesen, weil das Gleichungssystem für A und B eine *orthogonale* Koeffizientenmatrix hat: es ist $A = \sin 1$ und $B = \cos 1$. Damit wird

$$y(x) = \sin 1 \sin x^2 + \cos 1 \cos x^2 = \cos(1 - x^2). \tag{7.16}$$

Um die Differentialgleichung numerisch mittels Heun zu integrieren, führen wir sie auf ein System erster Ordnung zurück. Mit $y_1 = y$ und $y_2 = y'$ ergibt sich

$$\mathbf{y}' = \begin{pmatrix} 0 & 1 \\ -4x^2 & \dfrac{1}{x} \end{pmatrix} \mathbf{y} \quad \text{mit} \quad \mathbf{y}(1) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Das durch die Gleichungen (7.61∗) definierte Heunverfahren muss nun für Vektoren **y** angewendet werden. Wir deklarieren dazu die Prozedur

```
procedure f(x : real; y : vektor; var ys : vektor);
begin
  ys[1] := y[2];
  ys[2] := y[2]/x − 4 ∗ sqr(x) ∗ y[1]
end ;
```

welche das System beschreibt, d.h. zu gegebenem x und **y** die Ableitungen $\mathbf{y}' = \mathbf{ys}$ berechnet.

Im nachfolgenden Programm wird dieselbe Technik angewendet wie im Programm *integral*: das File *fdgl* wird als Include-File im Prozedurkörper von *f* dazugeladen, es beschreibt mehrere Differentialgleichungssysteme, welche durch eine **case** -Anweisung ausgewählt werden. Gleichzeitig kann dieses File von der Prozedur *menue* gelesen und auf dem Bildschirm präsentiert werden.

Das Programm sieht die Möglichkeit vor, entweder eine Wertetabelle auszugeben oder eine Funktion graphisch auf dem Bildschirm darzustellen. Es werden die Graphikprozeduren von Kapitel 8 dazu verwendet und als Include-File *plotproz* dazugeladen.

```
(*$B− Aufgabe 7.22*)
program systemheun;
const nn = 10;
type vektor = array [1..nn] of real;
var x, x0, h, xend, xmax, xmin, ymin, ymax : real ;
    m,n,i,k,schritte, fall, xg, yg, xt, yt, oldx, oldy, farbe, keine : integer;
    zeich : boolean;
    y, y0, ys, k1, k2 : vektor; tf : text; line : string [120];

(*$I plotproz *)

procedure f(x : real; y : vektor; var ys : vektor);
begin
   case fall of
      (*$I fdgl.pas*)
   end ;
end ;

begin
   assign(tf,'fdgl.pas'); reset(tf);
   while not eof(tf) do
   begin readln(tf,line); writeln(line) end ;
   writeln('Fall eingeben : '); read(fall);
   writeln('Anzahl Gleichungen ?'); read(n);
   writeln('Anfangsbedingungen eingeben x='); read(x0);
   for i := 1 to n do begin writeln('y[',i,'] ='); read(y0[i]) end ;
   writeln('bis wohin integrieren? xend='); read(xend);
   writeln('wieviele Integrationsschritte ?'); read(schritte);
   writeln('Wertetabelle (1) oder Zeichnung (2) ?');
   read(i); zeich:= i = 2;
   if zeich then
   begin
      writeln('welche Funktion plotten ?'); read(m);
      writeln('ymin, ymax eingeben'); readln( ymin, ymax);
      xmin:= x0; xmax:= xend;
      plotbegin( xg, yg, xt, yt, farbe, keine);
      achsen(xmin, xmax, ymin, ymax, xg, yg, xt, yt, farbe, keine);
   end ;
   h := (xend−x0)/ schritte; x := x0; y := y0;
   for k := 1 to schritte do
   begin
      f(x, y, k1); for i := 1 to n do ys[i] := y[i] + h ∗ k1[i];
      x := x0 + k ∗ h;
      f(x, ys, k2);
```

```
      for i := 1 to n do y[i] := y[i] + h/2 * (k1[i] + k2[i]);
      if zeich then pl(x, y[m], farbe)
      else
      begin
         writeln('x=', x : 7 : 5);
         for i := 1 to n do writeln('  y[',i : 2,'] =', y[i]);
      end
   end
end.
```

Wenn wir mit diesem Programm die Differentialgleichung (7.13) integrieren, so erhält man mit 300 Integrationsschritten im Intervall $[1, 5]$ auf dem Bildschirm eine Lösung, welche man wegen der gegebenen Auflösung nicht mehr von der exakten Funktion (7.16) unterscheiden kann. Druckt man dagegen eine Wertetabelle aus, so sieht man, dass die Werte noch ungenau sind und einen absoluten Fehler von ungefähr 10^{-1} aufweisen. Zum Beispiel erhält man bei

$x = 4.96$	exakt
$y[1] = 0.0752070593$	0.039644706
$y[2] = 9.9407638275$	9.912201289

und bei

$x = 5.00$	
$y[1] = 0.4591828355$	0.42417900
$y[2] = 8.9400525723$	9.05578362

Aufgabe 7.23 *Ein numerisches Verfahren liefert für $x = 1$ folgende Werte für die Lösung $y(x)$ einer Differentialgleichung:*

h=0.04	y(1)= 0.30151340
h=0.02	y(1)= 0.30145185

Der exakte Wert beträgt $y(1) = 0.30145781$. Welche Fehlerordnung hat das Verfahren wahrscheinlich ?

Nach der Faustregel sollte der Fehler bei Halbierung der Schrittweite ungefähr um den Faktor 2^{-p} abnehmen. Bilden wir die Differenzen der Näherungswerte und des exakten Werts, so erhalten wir

$$\text{für h=0.04: } 5559 \times 10^{-8} \text{ und für h = 0.02: } 596 \times 10^{-8}.$$

Es ist $5559 \times 2^{-3} = 694.8$, also *ist die Fehlerordnung wahrscheinlich* 3.

Aufgabe 7.24 *Man beweise, dass das Verfahren von Heun die Fehlerordnung $p = 2$ hat. Hinweis: Man benötigt die Taylorentwicklung einer Funktion von zwei Variabeln.*

Für eine Funktion f in mehreren Variabeln $\mathbf{x} = (x_1, \ldots, x_n)^T$ lautet die Taylorentwicklung

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \frac{1}{1!}\,\mathbf{grad} f(\mathbf{x})^T \mathbf{h} + \frac{1}{2!}\mathbf{h}^T \mathbf{hess} f(\mathbf{x})\,\mathbf{h} + O\big(||\mathbf{h}||^3\big) \quad (7.17)$$

Dabei ist für zwei Variable $\mathbf{x} = \binom{x}{y}$

$$\mathbf{grad} f(\mathbf{x}) = \begin{pmatrix} f_x \\ f_y \end{pmatrix} \quad \text{der Gradientenvektor}$$

und

$$\mathbf{hess} f(\mathbf{x}) = \begin{pmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{pmatrix} \quad \text{die Hesse Matrix von } f$$

Um die Fehlerordnung des Verfahren von Heun zu berechnen, müssen wir den Wert y_1 als Funktion der Schrittweite h in eine Reihe entwickeln und mit der Reihe des exakten Wertes $y(x_1)$ vergleichen. Beim Verfahren von Heun berechnet sich y_1 nach der Formel

$$y_1 = y_0 + \frac{h}{2}\left(f(x_0, y_0) + f\big(x_0 + h, y_0 + hf(x_0, y_0)\big)\right). \quad (7.18)$$

Da wir bei x_0 von der exakten Anfangsbedingung ausgehen, ist

$$y_0' = y'(x_0) = f(x_0, y_0) \quad (7.19)$$

und damit vereinfacht sich der Ausdruck (7.18) zu

$$y_1 = y_0 + \frac{h}{2} y_0' + \frac{h}{2} f(x_0 + h, y_0 + hy_0'). \quad (7.20)$$

Nun benützen wir die Taylorentwicklung (7.17) und erhalten

$$\begin{aligned} f(x_0 + h, y_0 + hy_0') = f(x_0, y_0) &+ \frac{h}{1!}\,\mathbf{grad} f(x_0, y_0)^T \begin{pmatrix} 1 \\ y_0' \end{pmatrix} \\ &+ \frac{h^2}{2!}\big(1, y_0'\big)\,\mathbf{hess} f(x_0, y_0) \begin{pmatrix} 1 \\ y_0' \end{pmatrix} + O\big(h^3\big). \end{aligned} \quad (7.21)$$

Setzt man dies in Gleichung (7.20) ein und beachtet (7.19), so folgt

$$\begin{aligned} y_1 = y_0 + hy_0' &+ \frac{h^2}{2}\,\mathbf{grad} f(x_0, y_0)^T \begin{pmatrix} 1 \\ y_0' \end{pmatrix} \\ &+ \frac{h^3}{4}\big(1, y_0'\big)\,\mathbf{hess} f(x_0, y_0) \begin{pmatrix} 1 \\ y_0' \end{pmatrix} + O\big(h^4\big). \end{aligned} \quad (7.22)$$

Andrerseits gilt für den exakten Wert die Taylorreihe

$$y(x_1) = y_0 + \frac{y'(x_0)}{1!}h + \frac{y''(x_0)}{2!}h^2 + \frac{y'''(x_0)}{3!}h^3 + O(h^4). \tag{7.23}$$

Durch Differentiation der Differentialgleichung kann man die Glieder der beiden Reihen (7.22) und (7.23) miteinander vergleichen:

$$\begin{aligned} y' &= f(x,y) \\ y'' &= f_x + f_y y' = \mathbf{grad} f^T \begin{pmatrix} 1 \\ y'_0 \end{pmatrix} \\ y''' &= f_{xx} + 2f_{xy}y' + f_{yy}{y'}^2 + f_y y'' = \big(1, y'_0\big)\mathbf{hess} f(x_0,y_0) \begin{pmatrix} 1 \\ y'_0 \end{pmatrix} + f_y y'' \end{aligned}$$

Bildet man die Differenz der Reihen $d_1(h) = y_1 - y(x_1)$, so heben sich die Glieder bis und mit h^2 auf und man erhält für den lokalen Diskretisationsfehler

$$d_1(h) = \left(\frac{1}{12}\big(1, y'_0\big)\, \mathbf{hess} f(x_0,y_0) \begin{pmatrix} 1 \\ y'_0 \end{pmatrix} - f_y(x_0,y_0)y''(x_0) \right) h^3 + O\big(h^4\big),$$

woraus man ablesen kann, dass die Fehlerordnung $p = 2$ ist.

Aufgabe 7.25 *Die spezielle Differentialgleichung*

$$y'(x) = f(x), \quad y(x_0) = 0$$

hat die Lösung

$$y(x) = \int_{x_0}^{x} f(t)\, dt.$$

Was für Quadraturformeln ergeben sich, wenn man die Differentialgleichung nach den Verfahren von Euler, Heun und Runge-Kutta integriert?

Wir führen einen Integrationsschritt der Länge h aus. Mit dem Verfahren von Euler erhält man wegen $y_0 = 0$

$$y_1 = y_0 + hf(x_0) = hf(x_0) \approx \int_{x_0}^{x_1} f(t)\, dt.$$

Man approximiert das Integral somit durch ein *Rechteck*.

Mit dem Verfahren von Heun wird

$$y_1 = \frac{h}{2}\big(f(x_0) + f(x_1)\big) \approx \int_{x_0}^{x_1} f(t)\,dt,$$

also erhält man hier die *Trapezregel*.

Mit Runge-Kutta schliesslich ist $k_1 = f(x_0)$, $k_2 = f(x_0 + \frac{h}{2}) = k_2$ und $k_4 = f(x_1)$. Daher wird

$$\begin{aligned} y_1 &= y_0 + \frac{h}{3}\left(\frac{1}{2}f(x_0) + 2f(x_0 + \frac{h}{2}) + \frac{1}{2}f(x_1)\right) \\ &= \frac{\tilde{h}}{3}\Big(f(x_0) + 4f(x_0 + \tilde{h}) + f(x_1)\Big) \quad \text{mit} \quad \tilde{h} = \frac{h}{2} \end{aligned}$$

und man erhält die *Simpsonregel*.

Aufgabe 7.26 *Das System* $\mathbf{y}' = \mathbf{A}\mathbf{y}$ *mit* $\mathbf{y}(0) = \mathbf{y}_0$ *und konstanter Matrix* $\mathbf{A}$ *hat bekanntlich die Lösung*

$$\mathbf{y}(x) = e^{\mathbf{A}x}\mathbf{y}_0. \qquad (7.67*)$$

Man führe einen Integrationsschritt der Länge h *mit jedem der drei Verfahren Euler, Heun und Runge-Kutta durch. Man schreibe das Resultat in der Form*

$$\mathbf{y}_1 = (\textit{Ausdruck in } \mathbf{A})\mathbf{y}_0$$

und vergleiche die Approximation mit der exakten Lösung $(7.67*)$ *für* $x = h$.

Nach der Methode von Euler ist

$$\mathbf{y}_1 = \mathbf{y}_0 + h\,\mathbf{A}\mathbf{y}_0 = (\mathbf{I} + h\mathbf{A})\mathbf{y}_0.$$

Mit Heun erhält man

$$\begin{aligned} \mathbf{y}^* &= \mathbf{y}_0 + h\mathbf{A}\mathbf{y}_0 = (\mathbf{I} + h\mathbf{A})\mathbf{y}_0 \\ \Rightarrow \mathbf{y}_1 &= \mathbf{y}_0 + \frac{h}{2}\left(\mathbf{A}\mathbf{y}_0 + \mathbf{A}(\mathbf{I} + h\mathbf{A})\mathbf{y}_0\right) \\ &= \left(\mathbf{I} + h\mathbf{A} + \frac{h^2}{2}\mathbf{A}^2\right)\mathbf{y}_0 \end{aligned}$$

Mit Runge Kutta ist

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{A}\mathbf{y}_0 \\
\mathbf{y}_a &= \mathbf{y}_0 + \frac{h}{2}\mathbf{A}\mathbf{y}_0 \\
\mathbf{k}_2 &= \mathbf{A}\mathbf{y}_a = \mathbf{A}\mathbf{y}_0 + \frac{h}{2}\mathbf{A}^2\mathbf{y}_0 \\
\mathbf{y}_b &= \mathbf{y}_0 + \frac{h}{2}\mathbf{A}\mathbf{y}_0 + \frac{h^2}{4}\mathbf{A}^2\mathbf{y}_0 \\
\mathbf{k}_3 &= \mathbf{A}\mathbf{y}_b = \mathbf{A}\mathbf{y}_0 + \frac{h}{2}\mathbf{A}^2\mathbf{y}_0 + \frac{h^2}{4}\mathbf{A}^3\mathbf{y}_0 \\
\mathbf{y}_c &= \mathbf{y}_0 + h\mathbf{A}\mathbf{y}_0 + \frac{h^2}{2}\mathbf{A}^2\mathbf{y}_0 + \frac{h^3}{4}\mathbf{A}^3\mathbf{y}_0 \\
\mathbf{k}_4 &= \mathbf{A}\mathbf{y}_c = \mathbf{A}\mathbf{y}_0 + h\mathbf{A}^2\mathbf{y}_0 + \frac{h^2}{2}\mathbf{A}^3\mathbf{y}_0 + \frac{h^3}{4}\mathbf{A}^4\mathbf{y}_0 \\
\Rightarrow \mathbf{y}_1 &= \mathbf{y}_0 + \frac{h}{3}\left(\frac{1}{2}\mathbf{k}_1 + \mathbf{k}_2 + \mathbf{k}_3 + \frac{1}{2}\mathbf{k}_4\right)
\end{aligned}$$

Setzt man die Ausdrücke für $\mathbf{k}_i$ ein und klammert $\mathbf{y}_0$ nach rechts aus, so ergibt sich das Resultat

$$\mathbf{y}_1 = \left(\mathbf{I} + \frac{h}{1!}\mathbf{A} + \frac{h^2}{2!}\mathbf{A}^2 + \frac{h^3}{3!}\mathbf{A}^3 + \frac{h^4}{4!}\mathbf{A}^4\right)\mathbf{y}_0.$$

Man sieht, dass bei Euler die ersten 2, bei Heun die ersten 3 und bei Runge-Kutta die ersten 5 Glieder der Exponentialreihe

$$e^{h\mathbf{A}} = \sum_{k=0}^{\infty} \frac{(h\mathbf{A})^k}{k!}$$

berechnet werden.

Aufgabe 7.27 *Mit dem Programm Runge-Kutta löse man die Differentialgleichung*

$$y'y + x = 0, \quad y(0) = 4$$

von $x = 0$ bis $x = 4$. Wie lautet die analytische Lösung (Separation der Variabeln!)?

Durch Integration der Differentialgleichung $yy' = -x$ ergibt sich

$$\frac{y^2}{2} = C - \frac{x^2}{2} \quad \Rightarrow y = \pm\sqrt{C - \frac{x^2}{2}}.$$

Bei Berücksichtigung der Anfangsbedingung wird die Integrationskonstante $C = 16$ und $y > 0$, somit stellt der Graph der exakten Lösung einen Viertelskreis mit Mittelpunkt $(0, 0)$ und Radius $r = 4$ im ersten Quadranten dar.

Für die numerische Lösung dieser und der nächsten Aufgabe verwenden wir ein etwas luxuriöseres Programm für das Runge-Kutta Verfahren als der spartanische Algorithmus 7.5 des Lehrbuchs. Analog wie bei Aufgabe 7.22 kann man mit dem folgenden Programm entweder eine Wertetabelle ausdrucken oder eine Zeichnung auf dem Bildschirm erzeugen. Die Wertetabelle wird auf ein File geschrieben, so dass man die Zahlen später in einem anderen Programm, zum Beispiel für Splineinterpolation, wieder einlesen und verwenden kann.

```
(*$B− Algorithmus 7.5*)
program runge;
const nn = 10;
type vektor = array [1..nn] of real;
var x, x0, h, xend,xmax, xmin, ymin, ymax : real ;
    m,n,i,k,schritte, fall, xg, yg, xt, yt, oldx, oldy, farbe, keine : integer;
    zeich : boolean; y, y0, ys, z, k1, k2, k3, k4 : vektor;
    tf,aus : text; line : string [120];

(*$I plotproz *)

procedure f(x : real; y : vektor; var ys : vektor);
begin
   case fall of
      (*$I fdgl.pas*)
   end ;
end ;

begin
   assign(tf,'fdgl.pas'); reset(tf);
   while not eof(tf) do
   begin readln(tf,line); writeln(line) end ;
   writeln('Fall eingeben : '); read(fall);
   writeln('Anzahl Gleichungen ?'); read(n);
   write('Anfangsbedingungen eingeben x='); read(x0);
   for i := 1 to n do begin write('y[',i,'] ='); read(y0[i]) end ;
   writeln('bis wohin integrieren? xend='); read(xend);
   writeln('wieviele Integrationsschritte ?'); read(schritte);
   writeln('Wertetabelle (1) oder Zeichnung (2) ?');
   readln(i); zeich:= i = 2;
```

```
if zeich then
begin
    writeln('welche Funktion plotten ?'); read(m);
    writeln('ymin, ymax eingeben'); readln( ymin, ymax);
    xmin:= x0; xmax:= xend;
    plotbegin( xg, yg, xt, yt, farbe, keine);
    achsen(xmin, xmax, ymin, ymax, xg, yg, xt, yt, farbe, keine);
end
else
begin
    writeln('Filenamen fuer die Wertetabelle ?');
    readln(line); assign(aus,line); rewrite(aus);
end ;

h := (xend−x0)/schritte; x := x0; y := y0;
for k := 1 to schritte do
begin
    f(x, y, k1);
    for i := 1 to n do z[i] := y[i] + h/2 ∗ k1[i];
    f(x + h/2, z, k2);
    for i := 1 to n do z[i] := y[i] + h/2 ∗ k2[i];
    f(x + h/2, z, k3);
    for i := 1 to n do z[i] := y[i] + h ∗ k3[i];
    f(x + h, z, k4);
    for i := 1 to n do
    y[i] := y[i] + h/3 ∗ (0.5 ∗ k1[i] + k2[i] + k3[i] + 0.5 ∗ k4[i]);
    x := x0 + k ∗ h;
        if zeich then pl(x, y[m], farbe)
        else
        begin
            write(aus, x); for i := 1 to n do write(aus, y[i]);
            writeln(aus)
        end
    end ;
    if not zeich then close(aus)
end.
```

Mit diesem Programm erhält man für $h = 0.2$ (d.h. mit 20 Integrationsschritten) die Werte von Tabelle 7.3.

Man sieht, wie der Fehler gegen $x = 4$ zunimmt, wo die y' einen Pol hat.

x_i	y_i	$y_i - y(x_i)$
0.20	3.9949968704	−6.5119820647E−10
0.40	3.9799497458	−2.6484485716E−09
0.60	3.9547439805	−6.1409082264E−09
0.80	3.9191835771	−1.1394149624E−08
1.00	3.8729833274	−1.8855644157E−08
1.20	3.8157567765	−2.9202055885E−08
1.40	3.7469987556	−4.3484760681E−08
1.60	3.6660604926	−6.3373590820E−08
1.80	3.5721141284	−9.1509718914E−08
2.00	3.4641014828	−1.3235330698E−07
2.20	3.3406584241	−1.9364597392E−07
2.40	3.1999997103	−2.8974318411E−07
2.60	3.0397363815	−4.4920670916E−07
2.80	2.8565706372	−7.3423871072E−07
3.00	2.6457500134	−1.2976597645E−06
3.20	2.3999974188	−2.5812259992E−06
3.40	2.1071245469	−6.2036742747E−06
3.60	1.7435387368	−2.0840616344E−05
3.80	1.2488562646	−1.4333508625E−04
4.00	0.1389684226	1.3896302778E−01

Tabelle 7.3: $y'y + x = 0$, $y(0) = 4$

Aufgabe 7.28 *Mittels Runge-Kutta löse man die Pendelgleichung*

$$\varphi'' + a\varphi' + b\sin\varphi = 0$$

für $a = 0.2$ *und* $b = 10$ *unter der Anfangsbedingung*

$$\varphi(0) = 0.3, \quad \varphi'(0) = 0.$$

Wie gross ist die Winkelgeschwindigkeit beim ersten Durchgang durch die Ruhelage ?

Das System erster Ordnung lautet für diese Differentialgleichung (siehe Aufgabe 7.19):

$$\begin{aligned} z_1' &= z_2 \\ z_2' &= -10\sin z_1 - 0.3z_2 \end{aligned} \quad \text{mit} \quad \mathbf{z}(0) = \begin{pmatrix} 0.3 \\ 0 \end{pmatrix}$$

Mit Algorithmus 7.5 erhält man mit der Schrittweite $h = 0.01$ beim Nulldurchgang die Werte von Tabelle 7.4, die eine absolute Genauigkeit von

t_i	φ_i	φ_i'
0.450	5.3791782134E−02	−8.9291817963E−01
0.460	4.4846145530E−02	−8.9605919456E−01
0.470	3.5873600034E−02	−8.9829975014E−01
0.480	2.6883154856E−02	−8.9963908494E−01
0.490	1.7883821963E−02	−9.0007740927E−01
0.500	8.8846063604E−03	−8.9961590464E−01
0.510	−1.0550360967E−04	−8.9825672165E−01
0.520	−9.0775459400E−03	−8.9600297607E−01
0.530	−1.8022594585E−02	−8.9285874333E−01
0.540	−2.6931769022E−02	−8.8882905127E−01
0.550	−3.5796243733E−02	−8.8391987128E−01

Tabelle 7.4: Pendelgleichung

mindestens 10^{-8} haben, wie man sich durch Vergleich entsprechender Werte, die mit halber Integrationsschrittweite berechnet wurden, überzeugen kann. Man kann die Geschwindigkeit beim Nulldurchgang nun durch Interpolation erhalten. Wir verwenden das Aitken-Neville-Schema mit $x_i = \varphi_i$ und $y_i = \varphi_i'$ und interpolieren für $x = 0$, indem wir abwechslungsweise einen neuen Stützpunkt links und rechts vom Nulldurchgang wählen. Man erhält mit Algorithmus 6.2 das Schema:

$$\begin{array}{cccc} 0.008885 & -0.89961590 & & \\ -0.000106 & -0.89825672 & -0.89827267 & \\ 0.017884 & -0.90007741 & -0.89826740 & -0.89827788 \\ -0.009078 & -0.89600298 & -0.89737479 & -0.89827790 \quad -0.89827789 \end{array}$$

Es genügen schon 4 Punkte, um das Resultat $\varphi' = -0.89827789$ zu bestimmen.

Aufgabe 7.29 *Man schreibe für das Runge-Kutta-Fehlberg Verfahren ein Programm mit Schrittweitensteuerung.*

Wir übernehmen für dieses Programm den abgeänderten Algorithmus 7.5 von Aufgabe 7.27. Die Prozedur *schritt* führt neu einen Integrationsschritt durch. Aus den beiden Näherungswerten $\mathbf{y}_{k+1} = \mathbf{y1}$ und $\mathbf{y}^*_{k+1} = \mathbf{y2}$ berechnet man die beiden Grössen

$$\mathit{fehler} := |\mathbf{y1} - \mathbf{y2}|_\infty \text{ und } \mathit{norm} := |\mathbf{y2}|_\infty.$$

Die Variable *eps* enthält die gewünschte Integrationsgenauigkeit. Wenn

$$\mathit{fehler} > \mathit{eps} * \mathit{norm}, \quad \text{d.h.} \quad \frac{|\mathbf{y1} - \mathbf{y2}|_\infty}{|\mathbf{y2}|_\infty} > \mathit{eps}$$

ist, muss der Schritt als misslungen betrachtet und mit einem kleineren $h := 0.8 * h$ wiederholt werden. Gelingt der Schritt, so wird nach Gleichung (7.70∗) die nächste Schrittweite h geschätzt. Die Anzahl der gelungen und misslungen Schritte wird gezählt und am Ende der Rechnung ausgedruckt.

```
(*$B-,U+ Aufgabe 7.29*)
program fehlberg;
const nn=10;
type vektor = array [1..nn] of real;
var x,x0,h, xend,xmax, xmin, ymin, ymax,
    man,max,norm,fehler,eps : real ;
    m,n,i,k, fall, xg, yg, xt, yt, oldx, oldy, farbe, keine,miss,gel : integer;
    zeich,misslungen : boolean;
    y,y0,ys,y1,y2,z,k1,k2,k3,k4,k5,k6 : vektor;
    tf,aus : text; line : string [120];

(*$I plotproz *)

procedure f(x : real; y : vektor; var ys : vektor);
begin
   case fall of
      (*$I fdgl.pas*)
   end ;
end ;

procedure schritt;
begin
   repeat
      f(x,y,k1);
for i := 1 to n do z[i] := y[i] + h/4 * k1[i];
f(x + h/4,z,k2);
for i := 1 to n do z[i] := y[i] + h/32 * (3 * k1[i] + 9 * k2[i]);
f(x + 3 * h/8,z,k3);
for i := 1 to n do
z[i] := y[i] + h/2197 * (1932 * k1[i] - 7200 * k2[i] + 7296 * k3[i]);
f(x + 12 * h/13,z,k4);
for i := 1 to n do
z[i] := y[i] + h * (439 * k1[i]/216 - 8 * k2[i]
                + 3680 * k3[i]/513 - 845 * k4[i]/4104);
f(x + h,z,k5);
for i := 1 to n do
z[i] := y[i] + h * (-8 * k1[i]/27 + 2 * k2[i] - 3544 * k3[i]/2565
                + 1859 * k4[i]/4104 - 11 * k5[i]/40);
f(x + h/2,z,k6);
norm:= 0; fehler:= 0;
```

```
      for i := 1 to n do
      begin
         y1[i] := y[i] + h * (25 * k1[i]/216 + 1408 * k3[i]/2565
                        + 2197 * k4[i]/4104 - k5[i]/5);
         y2[i] := y[i] + h * (16 * k1[i]/135 + 6656 * k3[i]/12825.0
               + 28561.0 * k4[i]/56430.0 - 9 * k5[i]/50 + 2 * k6[i]/55);
         max:= abs(y1[i] - y2[i]);
         if max > fehler then fehler:= max;
         max:= abs(y2[i]);
         if max > norm then norm:= max;
      end ;
      misslungen := fehler> eps* norm;
      if misslungen then
      begin miss:= miss+1; h := 0.8 * h end ;
   until not misslungen;
   gel:= gel+1;
end ;

begin
   assign(tf,'fdgl.pas'); reset(tf);
   while not eof(tf) do
   begin readln(tf,line); writeln(line) end ;
   writeln('Fall eingeben:'); read(fall);
   writeln('Anzahl Gleichungen ?'); read(n);
   write('Anfangsbedingungen eingeben x='); read(x0);
   for i := 1 to n do begin write('y[',i,'] ='); read(y0[i]) end ;
   writeln('bis wohin integrieren? xend='); read(xend);
   writeln('eps =?'); read(eps);
   writeln('Wertetabelle (1) oder Zeichnung (2) ?');
   readln(i); zeich:= i = 2;
   if zeich then
   begin
      writeln('welche Funktion plotten ?'); read(m);
      writeln('ymin, ymax eingeben'); readln( ymin, ymax);
      xmin:= x0; xmax:= xend;
      plotbegin( xg, yg, xt, yt, farbe, keine);
      achsen(xmin, xmax, ymin, ymax, xg, yg, xt, yt, farbe, keine);
   end
   else
   begin
      writeln('Filenamen fuer die Wertetabelle ?');
      readln(line); assign(aus,line); rewrite(aus);
   end ;
```

```
    miss:= 0; gel:= 0; h := (xend−x0)/100; x := x0; y := y0;
    repeat
        schritt; y := y2; x := x + h;
        if zeich then pl(x, y[m], farbe)
        else
        begin
            write(aus, x);
            for i := 1 to n do write(aus, y[i]);
            writeln(aus)
        end ;
        if fehler= 0 then h := 1.2 ∗ h
        else h := 0.8 ∗ h ∗ exp(ln( eps∗ norm/ fehler)/5) ;
    until x >= xend;
    if zeich then gotoxy(1, 23);
    writeln('misslungen : ',miss,' gelungen : ',gel,' Schritte');
    if not zeich then close(aus)
end.
```

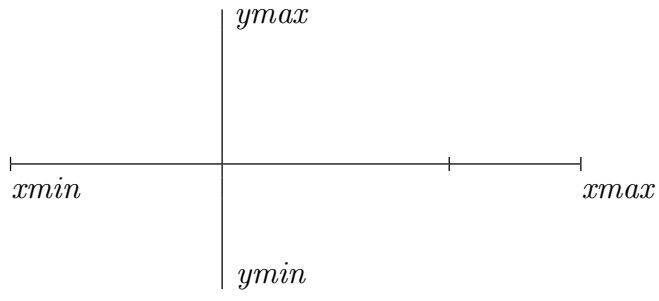
Kapitel 8

Hinweise zur Programmierung

Plotprozeduren zum Zeichnen von Funktionen

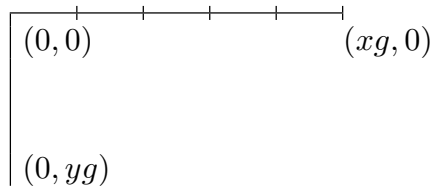
In diesem Abschnitt stellen wir einige TURBO PASCAL Prozeduren zusammen,welche es ermöglichen, unter Verwendung von nur *Basic Graphics* Funktionen zu zeichnen.

Zunächst befassen wir uns damit, Punkte in den verschiedenen Koordinatensystemen umzurechnen. Der Benützer der Graphikprozeduren arbeitet in einen Koordinatensystem, das vom Problem her gegeben ist (siehe Figur 8.1).



Figur 8.1: Koordinatensystem des Benützers

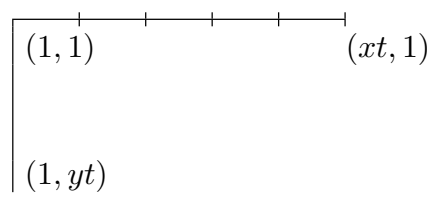
Auf dem Bildschirm des Computers werden 2 Koordinatensysteme verwendet, eines für Textoutput und eines für Graphik. Wenn man Text und Graphik mischen will, muss man sich beide Systeme aufeinander gelegt denken. Im Graphik Modus liegt der Nullpunkt des Systems in der linken oberen Ecke des Schirms, die x-Achse verläuft nach rechts und die y-Achse senkrecht nach unten (siehe Figur 8.2).



Figur 8.2: Koordinatensystem des Bildschirms für Graphik

Alle Punkte auf dem Bildschirm haben ganzzahlige Koordinaten und die Achsenlängen betragen im Graphikmodus $yg = 199$ und $xg = 319$ für *graphmode* und $xg = 619$ für *hires*.

Im Textmodus haben die Koordinatenachsen auf dem Bildschirm die gleiche Richtung wie im Graphikmodus, ihr Schnittpunkt links oben ist aber nicht $(0,0)$ sondern $(1,1)$. Ferner sind ihre Längen verschieden, nämlich $yt = 25$ und $xt = 40$ bzw. $xt = 80$ entsprechend dem dazu gewählten Graphikmodus (siehe Figur 8.3).



Figur 8.3: Koordinatensystem des Bildschirms für Text

Die Umrechnung der Koordinaten eines Punktes (x_1, x_2) im Benützersystem in den entsprechenden Punkt (y_1, y_2) des Bildschirmkoordinatensystem geschieht mit einer linearen Abbildung

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}. \tag{8.1}$$

Um die Matrix $\mathbf{A}$ und den Vektor $\mathbf{b}$ zu berechnen genügt es, die Gleichungen aufzustellen, die man erhält, wenn die Eckpunkte des Benützerkoordinatensystems auf die entsprechenden des Bildschirms abgebildet werden. Wir beginnen mit der Abbildung für den Graphikschirm. Die Rechnung wird etwas einfacher, wenn man zuerst die Umkehrabbildung

$$\mathbf{x} = \mathbf{B}\mathbf{y} + \mathbf{c} \tag{8.2}$$

bestimmt. Es soll gelten:

$$\textit{Bildschirmpunkt } \mathbf{y} \longmapsto \textit{Punkt } \mathbf{x} \textit{ im Benützersystem}$$

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \longmapsto \quad \begin{pmatrix} xmin \\ ymax \end{pmatrix} \tag{8.3}$$

$$\begin{pmatrix} 0 \\ yg \end{pmatrix} \quad \longmapsto \quad \begin{pmatrix} xmin \\ ymin \end{pmatrix} \tag{8.4}$$

$$\begin{pmatrix} xg \\ 0 \end{pmatrix} \quad \longmapsto \quad \begin{pmatrix} xmax \\ ymax \end{pmatrix} \tag{8.5}$$

Setzt man (8.3) in den Ansatz (8.2) ein, so erhält man

$$\mathbf{c} = \begin{pmatrix} xmin \\ ymax \end{pmatrix}.$$

Die Beziehung (8.4) liefert die beiden Gleichungen

$$\begin{aligned} xmin &= b_{12}\ yg + xmin \quad &\Rightarrow b_{12} = 0 \\ ymin &= b_{22}\ yg + ymax \quad &\Rightarrow b_{22} = \frac{ymin - ymax}{yg} \end{aligned}$$

und die erste Kolonne der Matrix **B** erhält man mittels (8.5):

$$\begin{aligned} xmax &= b_{11}\ xg + xmin \quad &\Rightarrow b_{11} = \frac{xmax - xmin}{xg} \\ ymax &= b_{21}\ xg + ymax \quad &\Rightarrow b_{21} = 0. \end{aligned}$$

Die Matrix **B** ist somit eine Diagonalmatrix und man kann die Gleichung (8.2) leicht nach **y** auflösen:

$$\mathbf{y} = \begin{pmatrix} \frac{1}{b_{11}} & 0 \\ 0 & \frac{1}{b_{22}} \end{pmatrix} (\mathbf{x} - \mathbf{c}) \tag{8.6}$$

Die Gleichung (8.6) ist die gesuchte lineare Abbildung (8.1) und lautet in Komponenten:

$$\begin{aligned} y_1 &= xg\ \frac{x_1 - xmin}{xmax - xmin} \\ y_2 &= yg\ \frac{x_2 - ymax}{ymin - ymax} \end{aligned} \tag{8.7}$$

Für den Textbildschirm kann man die Beziehung (8.7) übernehmen, wenn man zuerst eine Verschiebung mit dem Vektor (1,1) durchführt, damit der linke obere Bildpunkt wieder die Koordinaten (0,0) besitzt. Man erhält dadurch die Abbildung:

$$\begin{aligned} y_1 &= (xt - 1)\ \frac{x_1 - xmin}{xmax - xmin} + 1 \\ y_2 &= (yt - 1)\ \frac{x_2 - ymax}{ymin - ymax} + 1 \end{aligned} \tag{8.8}$$

Da die Bildschirmkoordinaten ganzzahlig sind, müssen bei der Transformation die Werte *gerundet* werden. Die Standardfunktion *round* unterbricht den Ablauf des Programms mit einer Fehlermeldung, wenn das Argument grösser als die grösste darstellbare *integer* Zahl ist. Ein solcher Unterbruch ist aber hier unerwünscht und unberechtigt, da beim Zeichnen Punkte, die ausserhalb des Bildschirmbereichs liegen, einfach ignoriert werden. Wir verwenden deshalb die folgende Funktion für das Runden, welche für zu grosse Argumente die konstante Zahl 30'000 liefert:

```
function rd(x :real) :integer;
var si : integer;
begin
   si := 1; if x < 0 then si := −1;
   if abs(x) > 3e4 then rd := si ∗ 30000 else rd :=round(x);
end ;
```

Mit dieser Funktion kann nun die Transformation für den Textbildschirm nach Gleichung (8.8) wie folgt durchgeführt werden:

```
procedure trafo(x, y: real; var xi, yi: integer);
(∗ global: xmin,xmax,ymin,ymax : real; xt,yt : integer ∗)
(∗ transformiert die Koordinaten fuer Textschirm ∗)
begin
   xi := rd((xt − 1)/(xmax − xmin) ∗ (x − xmin) + 1);
   yi := rd((yt − 1)/(ymin − ymax) ∗ (y − ymax) + 1);
end ;
```

Mit den Anweisungen

```
trafo(1, f(1), xi, yi); gotoxy(xi, yi + 2); write('f(x)');
```

kann nun z.B. der Graph der Funktion $f(x)$ bei $x = 1$ beschriftet werden. Um die Parameterliste nicht schwerfällig zu verlängern, werden die Variabeln *xmin, xmax, ymin, ymax, xt* und *yt* global übernommen. Bei Namenskonflikten bleibe es dem Leser überlassen, diese zu ändern.

Die Grenzen des Benützerkoordinatensystems *xmin, xmax, ymin* und *ymax* werden eingelesen. Die Schirmparameter werden mit dem Aufruf der folgenden Prozedur zu Beginn des Zeichnungsvorganges gesetzt:

```
procedure plotbegin(var xg, yg, xt, yt, farbe, keine: integer);
var i : integer;
begin
   writeln('hires 1=ja ?'); read(i);
   if i <> 1 then
   begin
      graphcolormode; graphbackground(6);
      palette(0); farbe:= 2; keine:= 3;
      xg := 319; yg := 199; xt := 40; yt := 25;
   end
   else
   begin
      hires; graphbackground(10); palette(0);
      xg := 619; yg := 199; xt := 80; yt := 25;
      farbe:= white; keine:= black;
```

```
   end ;
end ;
```

Der vorliegende Vorschlag ist für einen schwarz-weiss Bildschirm gedacht. Es wird nur mit einer Farbe (*farbe*) gezeichnet. Die Variable *keine* enthält die Hintergrundfarbe, welche zum löschen von Zeichnungen gebraucht werden kann. Auch hier bleibe es dem Leser überlassen, aus der Vielfalt der möglichen Parameterkombinationen die ihm passende auszuwählen. Es sei dabei auf das TURBO PASCAL Handbuch verwiesen.

Um eine Funktion zu zeichnen benützen wir die folgende Prozedur $pl(x, y,farbe)$, welche eine Gerade von der letzten Position zum Punkt (x, y) zeichnet. Um wiederum die Parameterliste nicht unnötig zu verlängern sind viele Variabeln global gewählt worden. Die letzte Cursor-Position auf dem Graphikschirm ist jeweils in den ganzzahligen Variabeln (*xold, yold*) gespeichert. Bei Namenskonflikten bleibt es wieder dem Leser überlassen, die Namen entsprechend zu ändern.

```
procedure pl( x,y : real; farbe : integer);
(* global xmin,xmax,ymin,ymax : real,
         xg,yg,xold,yold,keine : integer und function rd *)
var xi, yi : integer;
begin
   xi := rd(xg/( xmax− xmin) ∗ (x−xmin));
   yi := rd(yg/( ymin− ymax) ∗ (y−ymax));
   if farbe<> keine then draw( oldx,oldy,xi,yi,farbe);
   oldx:= xi; oldy:= yi
end ;
```

Die folgende Prozedur $kreuz(x, y, l)$ zeichnet im Punkt (x, y) ein Kreuz der Länge und Breite von l Bildschirmpunkten. Sie wird vor allem bei der Interpolation gebraucht, um die Stützpunkte zu markieren.

```
procedure kreuz(x, y :real; l : integer);
(* global xmin,xmax,ymin,ymax, xg,yg,xold,yold
         und function rd *)
var xi, yi :integer;
begin
   xi := rd(xg/(xmax − xmin) ∗ (x − xmin));
   yi := rd(yg/(ymin − ymax) ∗ (y − ymax));
   draw(xi − l, yi, xi + l, yi, farbe);
   draw(xi, yi − l, xi, yi + l, farbe);
end ;
```

Die letzte und komplizierteste Prozedur schliesslich zeichnet das Achsenkreuz. Diese Prozedur enthält keine globalen Variabeln, der Benützer kann

sie daher in allen seinen Programmen verwenden und braucht sie auch bei den oben erwähnten Namenskonflikten nicht zu verändern. Die lange Parameterliste, die sich als Folge ergibt, kann man leicht in Kauf nehmen, da in einem Programm *achsen* nur einmal aufgerufen wird.

Es wird versucht für alle Situationen ein Achsenkreuz mit automatischer vernünftiger Beschriftung der Achsen zu zeichnen. Diese Aufgabe ist gar nicht so einfach, wie es vielleicht im ersten Augenblick scheint. Um das Bild nicht zu überlasten, werden 5 bis 10 Markierungsstriche auf den Achsen gezeichnet und angeschrieben. Naürlich sollten diese Markierungen möglichst bei ungebrochenen Zahlen stehen, selbst dann, wenn die Grenzen des Benützerfensters, das durch *xmin, xmax, ymin* und *ymax* definiert ist, als Dezimalbrüche eingegeben werden. Wenn bei der Beschriftung der Markierungen so grosse Zahlen auftreten, dass man sie als *real* Zahlen mit Exponenten schreiben muss, dann wird der Platz auf dem Bildschirm zu klein. In solchen Fällen wird eine Skalierung mit einer Zehnerpotenz vorgenommen.

Wenn der Koordinatenursprung nicht im Benützerfenster enthalten ist, werden die Achsen verschoben. Der neue Achsenschnittpunkt hat die Koordinaten ox und oy.

Liest man nun als Koordinaten eines Punktes auf dem Bildschirm die Werte x_a und y_a ab, so besteht wegen der Nullpunktverschiebung (ox, oy) und den Skalierungsfaktoren 10^{ex} bzw. 10^{ey} die folgende Beziehung zwischen diesen und den wahren Koordinaten (x, y):

$$x = ox + x_a \times 10^{ex}$$
$$y = oy + y_a \times 10^{ey}$$

Diese Transformationsgleichungen werden unten rechts auf den Bildschirm geschrieben und damit können die Koordinaten von Punkten leicht umgerechnet werden.

Damit die nachfolgende Prozedur *achsen* unabhängig ist, enthält sie die oben angegebenen Prozeduren *pl* und *trafo*, als lokale Prozeduren. Wegen der Skalierung treten nur kleine *integer* Zahlen auf und deshalb wird die Prozedur *rd* nicht gebraucht. Die Prozedur *zerlege* wird bei der Skalierung verwendet. Sie zerlegt eine *real* Zahl x in Mantisse und Exponent (zur Basis 10). Die **function** *skala* bestimmt eine ganzzahlige Schrittweite für die Markierungsstriche auf einer Achse.

```
procedure achsen(xmin,xmax,ymin,ymax: real;
                 xg,yg,xt,yt,farbe,keine : integer);
var i,k,ex,ey,xold,yold,xi,yi : integer;
    ox,oy, mx,my : real; stri : string [20] ;
function zehnhoch(e : integer) : real;
begin zehnhoch:= exp(e * ln(10)) end ;
```

```
procedure trafo(x, y : real; var xi,yi : integer);
begin
   xi := round((xt − 1)/( xmax−xmin) ∗ (x−xmin) + 1);
   yi := round((yt − 1)/( ymin−ymax) ∗ (y−ymax) + 1);
end ;
procedure pl(x, y : real; farbe : integer);
var xi, yi : integer;
begin
   xi := round(xg/( xmax− xmin) ∗ (x−xmin));
   yi := round(yg/( ymin− ymax) ∗ (y−ymax));
   if farbe<> keine then draw(oldx,oldy,xi,yi,farbe);
   oldx:=xi; oldy:=yi
end ;
procedure tick(x, y : real; ch : char);
(∗ Produziert die Tickmarks auf Achsen ∗)
var xi,yi : integer;
begin
   xi := round(xg/( xmax−xmin) ∗ (x−xmin));
   yi := round(yg/( ymin−ymax) ∗ (y−ymax));
   case ch of
      'x','v' : (∗ auf x–Achse∗) draw(xi, yi − 3, xi, yi + 3, farbe);
      'y','h' : (∗ auf y–Achse∗) draw(xi − 3, yi, xi + 3, yi, farbe);
   end
end ;
procedure zerlege(x : real; var mantisse : real; var exponent : integer);
(∗ zerlegt eine real Zahl in Mantisse und Exponent ∗)
var e : integer;
begin
   e := trunc(ln(x)/ln(10)); x := x/exp(e ∗ ln(10));
   (∗ anpassen ∗)
   if x < 1 then begin x := x ∗ 10; e := e − 1 end ;
   if x > 10 then begin x := x/10 ; e := e + 1 end ;
   exponent:= e; mantisse:= x;
end ;
function skala(a, b : real) : integer;
(∗ bestimmt eine einen integer Skalenschritt für die Tickmarks ∗)
var k : real;
begin
   k := 1;
   while (b − a)/k >= 10 do k := k ∗ 10;
   while (b − a)/k < 5 do k := k/2;
   if k > 1 then skala:= round(k) else skala:= 1;
```

```
end ;
begin
  ox := 0; oy := 0;
  (* Nullpunktverschiebung *)
  if xmin > 0 then
  begin
    zerlege(xmax, my, ey); ey := ey − 1; my := my ∗ 10;
    ox := round(xmin∗zehnhoch(−ey))∗zehnhoch(ey);
    (* Korrektur, falls ox auuserhalb des Intervalls liegt *)
    if ox > xmax then ox := xmax
    else if ox < xmin then ox := xmin;
  end
  else
  if xmax < 0 then
  begin
    zerlege(−xmin, my, ey); ey := ey − 1; my := my ∗ 10;
    ox := −round(−xmax∗zehnhoch(−ey))∗zehnhoch(ey);
    if ox < xmin then ox := xmin
    else if ox > xmax then ox := xmax
  end ;
  xmax := xmax − ox; xmin := xmin − ox;
  if ymin > 0 then
  begin
    zerlege(ymax, my, ey); ey := ey − 1; my := my ∗ 10;
    oy := round(ymin∗zehnhoch(−ey))∗zehnhoch(ey);
    if oy > ymax then oy := ymax
    else if oy < ymin then oy := ymin
  end
  else
  if ymax < 0 then
  begin
    zerlege(−ymin, my, ey); ey := ey − 1; my := my ∗ 10;
    oy := −round(−ymax∗zehnhoch(−ey))∗zehnhoch(ey);
    if oy < ymin then oy := ymin
    else if oy > ymax then oy := ymax
  end ;
  ymax := ymax − oy; ymin := ymin − oy;
  (* Skalierung *)
  zerlege(xmax − xmin, mx, ex); zerlege(ymax − ymin, my, ey);
  ex := ex − 1; ey := ey − 1;
  xmax := xmax∗zehnhoch(−ex); xmin := xmin∗zehnhoch(−ex);
  ymax := ymax∗zehnhoch(−ey); ymin := ymin∗zehnhoch(−ey);
```

```
(* x-Achse *)
pl(xmin,0,keine); pl(xmax,0,farbe);
k :=skala(xmin,xmax);
for i :=round(xmin) to round(xmax) do
begin
   if i mod k = 0 then
   begin
      str(i,stri); if i > 0 then stri:=' '+stri;
      trafo(i,0,xi,yi);
      xi := xi - 1; if xi < 1 then xi := 2;
      yi := yi + 1; if yi >= yt then yi := yt - 1;
      gotoxy(xi,yi);
      if (xi+length(stri) < xt) then writeln(stri);
      tick(i,0,'x');
   end ;
end ;
(* y-Achse *)
pl(0,ymin,keine); pl(0,ymax,farbe);
k :=skala(ymin,ymax);
for i :=round(ymin) to round(ymax) do
begin
   if i mod k = 0 then
   begin
      str(i,stri); if i > 0 then stri:=' '+stri;
      trafo(0,i,xi,yi);
      xi := xi + 1;
      if xi+length(stri) > xt then xi := xt-length(stri) - 2;
      gotoxy(xi,yi);
      if (yi < yt) and (xi > 0) then writeln(stri);
      tick(0,i,'y');
   end ;
end ;
(* Einheiten beschriften *)
gotoxy(xt - 22,yt - 3);
if ox <> 0 then write(ox : 12,'+x') else write(' ': 14) ;
if ex <> 0 then
begin
   if ox = 0 then write('x'); write('*10^',ex)
end ;
gotoxy(xt - 22,yt - 2);
if oy <> 0 then write(oy : 12,'+y') else write(' ': 14) ;
if ey <> 0 then
```

```
   begin
      if oy = 0 then write('y'); write('*10^',ey)
   end
end ;
```

Mit diesen Prozeduren, die als Include-File *plotproz* dazugeladen werden, kann man nun mit wenig Befehlen ein Programm schreiben, um Funktionen zu zeichnen:

```
(*$B-*)
program fktplt;
var i, xg, yg, xt, yt, oldx, oldy, farbe, keine : integer;
   xmin, xmax, ymin, ymax, x, h : real ;

(*$I plotproz *)

function f(x : real) : real;
begin
   f := sin(x) + x * exp(x)
end ;

begin
   writeln('xmin, xmax, ymin, ymax eingeben');
   readln(xmin, xmax, ymin, ymax);
   plotbegin( xg, yg, xt, yt, farbe, keine);
   achsen(xmin, xmax, ymin, ymax, xg, yg, xt, yt, farbe, keine);
   pl(xmin, f(xmin), keine);
   h := ( xmax-xmin)/xg;
   for i := 0 to xg do
   begin
      x :=xmin+i * h;
      pl(x, f(x), farbe)
   end ;
end.
```

Aufgabe 8.1 *Gegeben ist die Funktion*

$$f(x) = \frac{e^{\sqrt{1-x^2}}\sqrt{1-x^2}\sin\sqrt{1-x^2}}{\arctan(1-x^2)\sin(1-x^2)-\sqrt{1-x^2}}$$

Das folgende Programm berechnet einen Funktionswert von f:

```
program funktion(input,output);
var h,w,zae,nen,f,x,
```

```
    hs,ws,zaes,nens,fs : real;
begin
    read(x);
    h := 1 − sqr(x);
    w := sqrt(h) ;
    zae := exp(w) ∗ w;
    zae := zae ∗ sin(w);
    nen :=arctan(h) ∗ sin(h) − w;
    f := zae/nen ;
    writeln(x, f);
end.
```

Man modifiziere dieses Programm durch algorithmische Differentiation so, dass neben f auch $f' = fs$ berechnet und ausgedruckt wird und berechne damit Funktionswert und Ableitung für $x = 0.5$.

Wir deklarieren für die Ableitungen neue Variable, wenden auf jede Anweisung die üblichen Differentiationsregeln an und schieben diese neue Anweisung *vor* der abzuleitenden Anweisung ein.

```
(∗$B− Aufgabe 8.1∗)
program funktion(input,output);
var h,w,zae,nen,f,x,
    hs,ws,zaes,nens,fs : real;
begin
    read(x);
        hs := −2 ∗ x;
    h := 1 − sqr(x);
        ws := 1/2/sqrt(h) ∗ hs;
    w := sqrt(h) ;
        zaes := exp(w) ∗ ws ∗ (w + 1);
    zae := exp(w) ∗ w;
        zaes := zaes ∗ sin(w) + zae ∗ cos(w) ∗ ws;
    zae := zae ∗ sin(w);
        nens := 1/(1 + sqr(h)) ∗ hs ∗ sin(h)+ arctan(h) ∗ cos(h) ∗ hs − ws;
    nen :=arctan(h) ∗ sin(h) − w;
        fs := (nen ∗ zaes − zae ∗ nens)/sqr(nen);
    f := zae/nen ;
    writeln(x, f, fs);
end.
```

Mit diesem Programm erhält man den Output

x	$f(x)$	$f'(x)$
0.49	−3.7618448291	9.2227205161
0.50	−3.6697322223	9.1984138516
0.51	−3.5779038359	9.1659226195

Als Kontrolle bilden wir den Differenzenquotienten

$$\frac{f(0.51) - f(0.49)}{0.02} = 9.19704966 \approx f'(0.50).$$

Er stimmt mit dem exakten Wert $f'(0.5)$ auf 3 Dezimalstellen überein.

Aufgabe 8.2 *Bei der Laguerreiteration (siehe Kap.4) wird die 2. Ableitung eines Polynoms benützt. Man berechne diese durch algorithmische Differentiation von Algorithmus 8.6 in Beispiel 8.3.*

Die Differentiation von Algorithmus 8.6 liefert

```
pss:= 0; ps:= 0; p := 0;
for i := n downto 0 do
begin
    pss:= pss*x + 2*ps;
    ps:=ps*x + p;
    p := p * x + a[i];
end ;
```

Dieses Programmstück wurde in Aufgabe 4.20 verwendet.

Aufgabe 8.3 *Mit dem Algorithmus determinante schreibe man ein Programm, um die Eigenwerte λ_i des allgemeinen Eigenwertproblems*

$$P_n(\lambda) = det(\mathbf{A} - \lambda\mathbf{B}) = 0 \qquad (8.8*)$$

zu berechnen, wobei $\mathbf{A}$ *und* $\mathbf{B}$ *symmetrisch sind und* $det(\mathbf{B}) \neq 0$ *ist (alle Eigenwerte sind in diesem Fall reell). Man verwende das Newtonverfahren für* $det(\mathbf{C}(\lambda)) = 0$, *wobei*

$$\mathbf{C}(\lambda) = \mathbf{A} - \lambda\mathbf{B}$$

und

$$\mathbf{C}'(\lambda) = -\mathbf{B}$$

ist.

Wenn man zur Lösung der Gleichung (8.8∗) *zuerst die Koeffizienten* a_i *des charakteristischen Polynoms* $P_n(\lambda)$ *berechnet*

$$P_n(\lambda) = a_0 + a_1\lambda + \cdots + a_n\lambda^n,$$

so ist bekanntlich dieser Weg numerisch instabil. Berechnet man aber den Funktionswert von P_n *mittels Gauss'scher Dreieckszerlegung und die Ableitung mit algorithmischer Differentiation, so resultiert ein numerisch stabiler Algorithmus. Allerdings ist dieser Algorithmus sehr rechenaufwendig. Er kann aber wiederum für beliebige nichtlineare* λ*-Matrizen* $\mathbf{C}(\lambda)$ *verwendet werden.*

Im folgenden Programm wird der Algorithmus 8.8 als Include-File dazugeladen. Da die Matrix $\mathbf{C}'(\lambda)$ konstant ist, berechnen wir sie nur einmal vor der Iteration. Durch Wahl von verschiedenen Startwerten kann man versuchen, alle Eigenwerte zu erhalten. Man könnte gefundene Eigenwerte deflatieren, wir verzichten darauf und überlassen diese Erweiterung dem interessierten Leser.

```
(*$B− Aufgabe 8.3*)
program eigenwert;
type vektor = array [1..10] of real;
   matrix = array [1..10] of vektor;
var i,j,n : integer; s,d,f,fs,l,ln : real;
   b,a,c,cs : matrix; y,x : vektor;
   stri : string [20]; tf : text;

(*$IA8.8 Algorithmus 8.8*)
begin
   writeln('wohin mit dem Output ? ');
   read(stri); assign(tf,stri); rewrite(tf);
   writeln('n eingeben'); read(n);
   writeln('Matrix A zeilenweise eingeben');
   for i := 1 to n do for j := 1 to n do read(a[i,j]);
   writeln('Matrix B eingeben');
   for i := 1 to n do for j := 1 to n do read(b[i,j]);
   for i := 1 to n do for j := 1 to n do cs[i,j] := -b[i,j];

   repeat
      write('Startwert lambda ='); read(ln);
      writeln(tf,' ': 6, 'lambda ',' ': 10, 'P(lambda)',' ': 8, 'P''(lambda)');
      repeat
         l := ln;
         for i := 1 to n do for j := 1 to n do
```

```
            c[i,j] := a[i,j] - l * b[i,j];

          determinante(c,cs,f,fs);

          ln := l - f/fs; writeln(tf,l,f,fs);
        until (abs(ln - l) < 1e - 5 * abs(ln)) or (abs(ln) < 1e-10);
        writeln(tf, 'Loesung = ', ln);
        writeln('nochmals ? (1) =ja'); read(i);
    until (i <> 1);
    close(tf);
end.
```

Mit diesem Programm erhält man für die beiden Matrizen

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{pmatrix} \quad \text{und} \quad \begin{pmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{pmatrix}$$

und dem Startwert $\lambda_0 = 7$ die folgenden Iterationswerte:

λ_i	$P(\lambda_i)$	$P'(\lambda_i)$
7.0000000000	−1.4146000000E+04	−6.7600000000E+03
4.9073964497	−4.1679845059E+03	−3.0216838654E+03
3.5280382205	−1.2180603929E+03	−1.3621048399E+03
2.6337895862	−3.4836686570E+02	−6.2775942270E+02
2.0788526898	−9.3741546253E+01	−3.0715911337E+02
1.7736638083	−2.0971149719E+01	−1.7494434024E+02
1.6537905406	−2.6477801526E+00	−1.3157271389E+02
1.6336664581	−6.8728878982E−02	−1.2476488215E+02
1.6331155909	−5.0806140763E−05	−1.2458044114E+02

Loesung = 1.6331151831

Es ist also $\lambda = 1.6331151831$ ein Eigenwert des Eigenwertproblems

$$det(\mathbf{A} - \lambda\mathbf{B}) = 0.$$

Programmindex

Nach dem Stichwort folgt in Grossbuchstaben der Filename des Programms auf der Diskette, danach kursiv der Programmname und anschliessend die Seitenzahl(en). Die mit '*' bezeichneten Seitenzahlen beziehen sich auf das Lehrbuch.

Errata des Lehrbuchs Computermathematik

Seite 36:

$$\frac{20^{20}}{20!} = \frac{20^{19}}{19!} \simeq 4.3 \times 10^7$$

Seite 42, obere Integrationsgrenze von:

$$\int_0^{0.8} \frac{dx}{\sqrt[3]{1+x^2}}$$

Seite 66:

$$x = \frac{24}{\sqrt{2x}} - 2 =: F(x) \tag{3.9}$$

Seite 71, fehlender Betragsstrich: Man spricht hier, falls $F'(s) \neq 0$ aber $|F'(s)| < 1$ ist, ...

Seite 96:

$$\begin{array}{ccccccccccc} 1 & & 1 & & 1 & & 0 & & 0 & & 1 \\ 0 & & 2 & & 6 & & 14 & & 28 & & 56 \\ & \swarrow & & \swarrow & & \swarrow & & \swarrow & & \swarrow & \uparrow \\ 1 & & 3 & & 7 & & 14 & & 28 & & 57 \end{array} \tag{4.11}$$

Seite 103: Aufgabe 4.12

$$P_4 = 10 - 35x + 45x^2 - 25x^3 + 5x^4$$

Seite 108:

$$|P_n(y)| \approx \varepsilon \sum_{i=0}^{n} |a_i y^i| \tag{4.27}$$

Seite 130: Strichpunkt vor dem Label 1:

```
    for i := 1 to n do write(x[i]) ;
1 :
end.
```

Seite 165: Die Variable k ist unnötig: die Deklaration von k und die Anweisung $k := 1$; ist zu streichen

Seite 175: In Figur 6.4 ist die Beschriftung der beiden Funktionen g' und g'' zu vertauschen.

Seite 188:

$$s_{i+1} = s_i + \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{6.60}$$

Seite 211, erster Abschnitt: ... (wobei $eps \geq macheps$ ist) ...

Seite 222 Bei Aufgabe 7.21 fehlt die Anfangsbedingung:

$$y'' - y' + 4x^2 y = 0 \quad \text{mit} \quad y(1) = 1,\ y'(1) = -1$$

Seite 225:

$$k_4 = f(x_k + h, y_c)$$

Seite 241, die Variabeln x, h, t und u sollten Grossbuchstaben sein: X, H, T und U.

Seite 244, hier ist eine Anweisung verlorengegangen. Beim Holen eines Eintrags des Stacks, muss die Variable i erniedrigt werden:

```
(* Eintrag holen *)
a := stack[i].a; b := stack[i].b; fa := stack[i].fa;
fm := stack[i].fm; fb := stack[i].fb;
i := i − 1; (* = fehlende Anweisung *)
repeat
    m := (a + b)/2; h := (b − a)/4;
```

Seite 252 und Seite 253, die j-Schleife muss in beiden Fällen erst von $i + 1$ an laufen:

for $j := i + 1$ **to** n **do**