FORTRAN und MATLAB
ein Versuch einer Einführung

Manuskript für eine Kompaktvorlesung
täglich 30. 9. bis zum 14. 10. 2002

Gerhard Opfer
Universität Hamburg
Fachbereich Mathematik

15. Dezember 2002

In memoriam Elsbeth Bredendiek

1937–1999, Professorin am Fachbereich Mathematik der Universität Hamburg. Sie hat jahrelang den Programmiersprachkurs geleitet und wesentlich dazu beigetragen, daß dieser Kurs eine feste Institution am Fachbereich Mathematik der Universität Hamburg wurde.

Vorwort

Es ist ein ungewöhnliches Experiment, FORTRAN, eine alte, und MATLAB, eine neue Programmiersprache nebeneinander zu unterrichten. FORTRAN (1954) zeichnet sich allein durch ein enormes Beharrungsvermögen aus. Es gibt vermutlich keine ähnlich alte Programmiersprache für die man auf einem modernen Rechner noch einen Compiler erwerben kann. Aber FORTRAN war nie eine moderne Sprache. Die Betreiber von FORTRAN haben immer versucht, auch bei Modernisierungsversuchen, Kompatibilität zu den alten Formen zu bewahren. Daher besteht FORTRAN heute aus einem Konglomerat von alten und neuen Sprachelementen, die das Erlernen erschweren. Bei FORTRAN steht das Rechnen umfangreicher Probleme und visualisieren der Ergebnisse in Tabellenform im Vordergrund. Graphische Elemente fehlen.

MATLAB (1987) ist darauf angelegt, mit einfachen Mitteln viele numerische Probleme zu lösen (lineare Gleichungssysteme z. B. mit Hilfe von `x=A\b`, also einer Folge von nur fünf Zeichen), und die Ergebnisse auf mannigfache Weisen in Graphiken darzustellen.

Das hier vorgelegte Manuskript soll eine kleine Hilfestellung leisten, um in den beiden Sprachen, einfache Programme, hauptsächlich zur Benutzung in den Vorlesungen über Numerische Mathematik herstellen zu können. Dies Manuskript ist aber weit von jeder Vollständigkeit entfernt.

Aufgaben zu dieser Vorlesung mit Kriterien zur Erlangung eines Übungsscheins und Hinweise zur Rechnerbenutzung werden an einer anderen Stelle formuliert.

Hamburg, im September 2002 G. O.

Seit dem Ende des Kurses sind einige Fehler im vorliegenden Manuskript berichtigt und einige Ergänzugen vorgenommen worden. Insgesamt ist diese neue Version um 6 Seiten länger als die ursprüngliche Version.

Zur Orientierung benutze man das Inhaltsverzeichnis mit den angegebenen Listen und das Stichwortverzeichnis. Die in Tabelle 8.2, S. 24 aufgelisteten, eingebauten FORTRAN-Prozeduren sind nur zu einem geringen Teil auch im Stichwortverzeichnis aufgeführt. Demonstrationsprogramme zu MATLAB findet man auch unter

`http://www.math.uni-hamburg.de/home/opfer/aufgaben01w.html`

Die in diesem Manuskript vorhandenen FORTRAN-Programme sind gesammelt unter

`http://www.math.uni-hamburg.de/home/opfer/aufgaben02w.html`

Hamburg, im Dezember 2002 G. O.

Inhaltsverzeichnis

Vorwort v
Inhalt vi
Liste der Beispiele vii
Liste der Tabellen vii
Liste der Figuren vii
Liste der Programme vii

1 FORTRAN und MATLAB, ein prinzipieller Unterschied 1
1.1 FORTRAN 1
1.2 MATLAB 3

2 Trennung von Anweisungen und Anbringung von Kommentaren 6
2.1 Trennung von Anweisungen und Kommentare in FORTRAN 6
2.2 Trennung von Anweisungen und Kommentare in MATLAB 7

3 Rahmenbedingungen 8
3.1 Rahmenbedingungen für FORTRAN 8
3.2 Rahmenbedingungen für MATLAB 8

4 Namen von Variablen 9

5 Deklaration von Variablen 10
5.1 Deklaration von Variablen in FORTRAN 10
5.2 Deklaration von Variablen in MATLAB 14

6 Unbedingte und bedingte Anweisungen 16
6.1 Unbedingte Anweisungen 16
6.2 Bedingte Anweisungen 18

7 Wiederholungen von Programmteilen 21
7.1 Wiederholungen in FORTRAN mit fester Anzahl von Wiederholungen 21
7.2 Wiederholungen in FORTRAN mit variabler Anzahl von Wiederholungen 21
7.3 Wiederholungen in MATLAB mit fester Anzahl von Wiederholungen 21
7.4 Wiederholungen in MATLAB mit variabler Anzahl von Wiederholungen 22

8 Unterprogramme in FORTRAN und MATLAB 23
8.1 Unterprogramme in FORTRAN 23
8.2 Unterprogramme in MATLAB 26

9 Ausgabeformate in FORTRAN und MATLAB 28
9.1 Ausgabeformate in FORTRAN 28
9.2 Ausgabeformate in MATLAB 31

10 Graphik in FORTRAN und MATLAB 32
10.1 Graphik in FORTRAN 32
10.2 Graphik in MATLAB 32

11 Entstehung der Programmiersprachen FORTRAN und MATLAB **35**
11.1 Entstehung von FORTRAN 35
11.2 Entstehung von MATLAB 35

Stichwortverzeichnis **36**

Liste der Beispiele

1.4 Ein mit `help` angezeigter MATLAB-Kommentar 4
1.5 Ein mit `type` angezeigtes MATLAB-Programm 5
1.6 Ein mit `type` nicht anzeigbares MATLAB-Programm 5
3.1 FORTRAN-Rahmen 8
3.2 FORTRAN-Rahmen mit Unterprogramm 8
6.1 FORTRAN-Programm mit Division durch Null 16
6.2 MATLAB-Operation `.^2` im Vergleich zu `^2` 17

Liste der Tabellen

5.5 Identifizierung von Konstanten in FORTRAN 12
6.3 Logische Operatoren in FORTRAN und MATLAB 19
6.4 Logische Ausdrücke in FORTRAN und MATLAB als Ergebnis von Vergleichen 20
8.2 FORTRAN IV und einige neuere Funktionen 24

Liste der Figuren

6.6 Mit `find` bestimmte lokale Extrema (MATLAB), lokale Extrema markiert 20
10.2 Geradlinige Verbindung von zwei Punkten (MATLAB) 33
10.3 Geradlinige Verbindung von mehreren Punkten (MATLAB) 34

Liste der Programme

1.1 Beispiel eines FORTRAN77-Programms 2
1.2 Dasselbe Programm in FORTRAN90/95 2
1.3 Ein erstes MATLAB-Programm 3
4.1 Groß-und Kleinschreibung in FORTRAN-Programmen 9
5.1 Einfache und doppelte Genauigkeit in FORTRAN 10
5.2 Einfache Variable in einem FORTRAN-Programm 11
5.3 Feld-Variable in einem FORTRAN-Programm 11
5.4 Selbst definierte Typ-Deklarationen in einem FORTRAN-Programmen 12
5.6 String-Array mit Gedicht als Text (FORTRAN) 13
5.7 Ausdruck des Gedicht-Programms (FORTRAN) 13
5.8 FORTRAN-Datentyp `type` 14
6.5 Mit `find` bestimmte lokale Extrema (MATLAB) 20
7.1 Taylorreihe für Sinus, Benutzung von `while` (MATLAB) 22
8.1 FORTRAN-Programm als Beispiel zur Benutzung von Unterprogrammen 23
8.3 Drei verschiedene Zeitmessungen (FORTRAN) 26
8.4 MATLAB-Programm zur Benutzung von Unterprogrammen 27
8.5 MATLAB-Beispiel zur rekursiven Definition 27
9.1 Zuordnung beim Lesen von indizierten Variablen (FORTRAN) 30
10.1 Das erste Bild (MATLAB) 32

1 FORTRAN und MATLAB, ein prinzipieller Unterschied

Computer-Programme (im weiteren nur noch *Programme* genannt) bestehen aus Zeichensequenzen, die unter Beachtung der jeweiligen (Syntax-) Regeln Folgen von *Anweisungen* ergeben. Diese sollen den Computer befähigen, das Programm mit Hilfe eines *Compilers* entsprechend zu interpretieren. Eine Anweisung (auch *Befehl* oder *Kommando* genannt) ist dabei ein Text, der dem Computer gewisse Aufgaben zuweist.

FORTRAN-Programme müssen bei Übergabe an den Compiler bereits vollständig formuliert sein und in Form einer Text-Datei vorliegen. Wir nennen dieses Programm *Quellprogramm*. Das Schreiben eines Quellprogramms geschieht mit Hilfe eines *Editors*. Der Umgang mit einem Editor sollte beim Besuch dieses Kurses bereits bekannt sein. Das fertige Quellprogramm wird mit Hilfe des Compilers in ein *Maschinenprogramm* übersetzt.

Ein MATLAB-Programm kann dagegen direkt am Computer komponiert werden. Jedes Zeichen bzw. jede Zeichenfolge kann sofort entsprechend ausgeführt werden. Ein vollständiges Quellprogramm muß also für MATLAB nicht vorliegen. Das erlaubt interaktives Arbeiten. Insbesondere ist es möglich, direkt bei der MATLAB-Benutzung Informationen über MATLAB einzuholen.

Wir werden an verschiedenen Stellen davon sprechen, daß ein Programm *gestartet* werden muß. Bei windows-basierten Rechnern bedeutet das im Regelfall, daß ein Mausklick an einer geeigneten Stelle auszuführen ist. Bei unix-basierten Rechnern (auch linux) erfolgt der Start entweder durch Tippen eines passenden Wortes in ein bestimmtes Fenster oder auch durch einen Mausklick. Programme in Apple-Rechnern sind im Regelfall über Menüleisten ebenfalls durch Mausklick zu starten.

Für Programme gilt generell: Sie werden genau so ausgeführt wie sie unter Beachtung der Regeln geschrieben worden sind und nicht so, wie sie vielleicht gewünscht werden.

1.1 FORTRAN

Ein FORTRAN-Programm besteht wie beschrieben aus einem vollständigen Quellprogramm, zusammengefaßt und untergebracht in einer Datei (manchmal sind das auch mehrere Dateien), das von einem *Compiler* in ein für den betreffenden Computer verständliches Maschinenprogramm übersetzt werden muß. Erst nach Fertigstellung des Maschinenprogramms kann dieses mit einem besonderen Startbefehl ausgeführt werden. Im Regelfall ist das geschriebene FORTRAN-Quellrogramm (bei den ersten Versuchen) syntaktisch (formal) nicht richtig, und man erhält nach dem Compiliervorgang nur eine Liste von Fehlermeldungen auf dem Bildschirm, die man oft schwer entziffern kann. Das ist eine uralte Krankheit von FORTRAN. Noch schwerer sind im Regelfall Fehler zu ermitteln, die erst nach Starten des compilierten Programms auftreten, sog. *run-time*-Fehler.

Über formale Regeln nach denen Quellprogramme aufzubauen sind, werden wir noch sprechen. FORTRAN-Programmieren besteht also aus den folgenden Schritten:

1. Schreiben des Quellprogramms in eine Datei, z. B. mit Namen `Datei.ext`.
2. Compilieren von `Datei.ext` und Erzeugen eines Maschinenprogramms z. B. mit dem Namen `Datei.out` oder einer Fehlerliste auf dem Bildschirm.
3. Starten des Maschinenprogramms `Datei.out` oder Beheben der Fehler und Neubeginn bei 2.

Es gibt historisch gesehen alte FORTRAN-Formen, die letzte dieser Formen, die etwa bis 1990 verwendet wurde, heißt FORTRAN77 und neue FORTRAN-Formen. Alte Formen benötigen ein strenges an Lochkartenformate angelehntes Korsett, die neuen Formen, können i. w. frei, d. h. formatungebunden in eine Datei geschrieben werden. Mehr darüber im letzten Abschnitt, S. 35.

Compilierprogramme haben z. B die entsprechenden Namen `f77, g77, f90, f95`. Dabei ist die oben im Dateinamen `Datei.ext` gewählte *Erweiterung* (auch *Extension* genannt) `.ext` des Namens weitgehend festgelegt. FORTRAN77-Programme, die mit `f77` oder `g77` gestartet werden sollen, benötigen die Erweiterung `.f`, `.ftn` oder

`.for`. FORTRAN90- und FORTRAN95-Programme, gestartet mit `f90` oder `f95` benötigen die Erweiterung `.f90` oder `.f95`. Den Namen des produzierten Maschinenprogramms kann man beim Start vorgeben in der Form

```
f90 -o Name_Maschinenprogramm Name_Quellprogramm
```

Da man aus Beispielen am besten lernt, geben wir ein FORTRAN-Programm in der alten (bis FORTRAN 77) und neuen Form (ab FORTRAN 90) an. Die angegebenen Zeilennummern sind nicht Bestandteil des Programms. Wir werden im weiteren Text nur auf die neuen Formen eingehen.

Programm 1.1. Beispiel eines FORTRAN77-Programms

```
1       PROGRAMME MAIN
2 C     From - Wed Aug 28 18:25:35 2002
3 C     Spaltenzaehlung in den folgenden zwei  Zeilen
4 C23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
5 C        |10       |20       |30       |40       |50       |60         |72
6 C ****************************************************************
7 C     Bildschirm- Ein- und Ausgabe
8 C     Iterative Annaeherung an die Kreiszahl pi
9 C     pi/4 = 1/1 -1/3 +1/5 -1/7 ... (konvergiert sehr langsam)
10 C
11 C     Autor:  Wolfgang Loebnitz  (Wed Aug 28 18:25:35 2002)
12 C ****************************************************************
13      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
14      PI=3.1415
15      PIEXAKT=4*ATAN(1.0)
16      WRITE(*,100)
17      READ(*,110) I
18      PV = 0.D0
19      V  = -1.D0
20      II = 2*I-1
21      DO 120 J=1,II,2
22      V  = -V
23      Q  = V/J
24      PV = PV+Q
25 C     WRITE(*,290) J, Q, PV
26   120 CONTINUE
27      PI = PV*4
28      FEHLER=PIEXAKT-PI
29      WRITE(*,300) I
30      WRITE(*,310) PI
31      WRITE(*,310) PIEXAKT
32      WRITE(*,310) FEHLER
33   9999 CONTINUE
34    100 FORMAT("Anzahl der Iterationen zur Berechnung von pi?")
35    110 FORMAT(I8)
36    290 FORMAT(I8,2X,F12.5,2X,F12.5)
37    300 FORMAT("Kreiszahl pi nach",2X,I8,1X," Iterationen:")
38    310 FORMAT(1X,F18.15)
39 C ----------------------------------------------------------------
40      STOP
41      END
```

Programm 1.2. Dasselbe Programm in FORTRAN90/95

```
1  program zur_berechnung_von_pi
2    ! ****************************************************************
3    !     Bildschirmein- und -ausgabe
4    !     iterative Annaeherung an die Kreiszahl pi
5    !     pi/4 = 1/1 -1/3 +1/5 -1/7 ... (konvergiert sehr langsam)
6    !
7    !     Autor:  Wolfgang Loebnitz  (Mittwoch Aug 28 18:25:35 2002)
8    !     umgeschrieben in FORTRAN90 von Gerhard Opfer
9    ! ****************************************************************
10   implicit none
11   double precision :: pi_exakt, pv, v, q, pi_approximativ, fehler
12   integer :: i, ii, j
13   pi_exakt=4*datan(1.d0) !FORTRAN-Form von arctan (arcus tangens) double precision
14   write(*,fmt="('Anzahl der Iterationen zur Berechnung von pi: ')",advance="no")
15   read(*,fmt="(i8)") i                !advance="no" heisst: kein Zeilenvorschub
16   pv = 0.d0; v  = -1.d0  !d signalisiert double precision bei Konstanten
17   ii = 2*i-1
18   do j=1,ii,2
19     v  = -v;  q  = v/j;  pv = pv+q
20     if(j>2*i-10) then !nur die letzten 5 Zeilen werden gezeigt
21       write(*,fmt="(i8,2x,f12.5,2x,f12.5)") j, q, pv
22     end if
```

```
23   end do
24   pi_approximativ = pv*4; fehler = pi_exakt-pi_approximativ
25   write(*,fmt="('Kreiszahl pi nach',2x,i8,1x,' Iterationen:',1x,f18.15)")&
26         i, pi_approximativ !& bedeutet Fortsetzung in der naechsten Zeile
27   write(*,fmt="('Exakt und Fehler: ',23x,2(1x,f18.15))") pi_exakt, fehler
28 end program zur_berechnung_von_pi
29 !Ergebnis:
30 !Anzahl der Iterationen zur Berechnung von pi: 200
31 !      391      -0.00256       0.78412
32 !      393       0.00254       0.78667
33 !      395      -0.00253       0.78414
34 !      397       0.00252       0.78665
35 !      399      -0.00251       0.78415
36 !Kreiszahl pi nach       200  Iterationen:  3.136592684838816
37 !Exakt und Fehler:                          3.141592653589793  0.004999968750977
```

1.2 MATLAB

Um MATLAB benutzen zu können, muß man vorher ein (kommerziell zu erwerbendes) MATLAB-Programm starten. Man erhält ein neues Fenster (ggf. auch mehrere) mit einem Text und einem sogenannten *Prompt* der Form >>. Das relevante Fenster sieht etwa so aus:

```
                    < M A T L A B >
          Copyright 1984-1999 The MathWorks, Inc.
             Version 5.3.0.10183 (R11)
                    Jan 21 1999


  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, type tour or visit www.mathworks.com.

>>
```

Jetzt kann man direkt hinter den Prompt Anweisungen schreiben, die nach Betätigen der Enter-Taste, s. S. 6, sofort ausgeführt werden. Ein Beispiel:

```
>> x=3; y=4;
>> z=x+y
z =
     7
```

Für umfangreichere Rechnungen ist das aber zu mühsam. Man schreibt also die auszuführenden Befehle in eine Datei, die die Erweiterung `.m` haben muß, und schreibt dann den Dateinamen (ohne das `.m`) hinter den Prompt. Dann wird das Programm (also die in der Datei zusammengefaßten Anweisungen) von links nach rechts und von oben nach unten, wie in der Datei angegeben, ausgeführt. Wir schreiben die angegebenen Befehle zum Beispiel in eine Datei mit dem Namen `erster_test.m`, die den folgenden Inhalt hat (die angegebenen Zeilennummern sind nicht Bestandteil des Programms):

Programm 1.3. Ein erstes MATLAB-Programm

```
1 x=3; y=4;
2 x+y      %Das Weglassen des trennenden Semikolons ; bewirkt eine
3          %Sichtbarmachung des Ergebnisses (der Operation x+y)
4          %auf dem Bildschirm. Wir sehen auch, dass Kommentare
5          %zeilenweise mit einem Prozentzeichen % eingeleitet werden.
```

Die Ausführung sieht dann so aus:

```
>> erster_test
ans =
     7
>>
```

Wir sehen hier einen kleinen Unterschied zu oben. Das Ergebnis lautet hier

```
ans =
     7
```

weil wir in dem Programm direkt `x+y` und nicht `z=x+y` geschrieben haben. Ein Ergebnis muß also nicht einer Variablen zugeordnet werden. Jeder syntaktische Fehler führt sofort mit einer Fehlermeldung zum Abbruch, ein einfaches Beispiel:

```
>> a=sin(b)
??? Undefined function or variable 'b'.

>>
```

Ein MATLAB-Programm führt also jede Anweisung sofort aus und wartet nicht auf ein besonderes, formales Ende. Es gibt verschiedene Möglichkeiten um sich innerhalb des MATLAB-Programms über MATLAB zu informieren. Dazu gehören die vier Kommandos

(1.1) 1. `help Name` 2. `lookfor Stichwort` 3. `type Name` 4. `demo`

1. `help Name`: Mit diesem Befehl kann man sich über ein Unterprogramm mit dem *bekannten* Namen `Name` informieren, wenn man Einzelheiten der Benutzung vergessen hat. Ein Beispiel:

   Beispiel 1.4. *Ein mit* `help` *angezeigter MATLAB-Kommentar*

   ```
   >> help quad

    QUAD   Numerically evaluate integral, low order method.
       Q = QUAD('F',A,B) approximates the integral of F(X) from A to B to
       within a relative error of 1e-3 using an adaptive recursive
       Simpson's rule.  'F' is a string containing the name of
       the\index{String!MATLAB}
       function.  Function F must return a vector of output values if given
       a vector of input values.  Q = Inf is returned if an excessive
       recursion level is reached, indicating a possibly singular integral.

       Q = QUAD('F',A,B,TOL) integrates to a relative error of TOL.  Use
       a two element tolerance, TOL = [rel_tol abs_tol], to specify a
       combination of relative and absolute error.

       Q = QUAD('F',A,B,TOL,TRACE) integrates to a relative error of TOL and
       for non-zero TRACE traces the function evaluations with a point plot
       of the integrand.

       Q = QUAD('F',A,B,TOL,TRACE,P1,P2,...) allows parameters P1, P2, ...
       to be passed directly to function F:   G = F(X,P1,P2,...).
       To use default values for TOL or TRACE, you may pass in the empty
       matrix ([]).

       See also QUAD8, DBLQUAD.

   >>
   ```

   Zu beachten ist, daß in dieser `help`-Funktion, alle Programmnamen zur besseren Hervorhebung groß geschrieben werden, sie aber tatsächlich bei Benutzung klein geschrieben werden müssen.

2. `lookfor Stichwort`: Hier wird in allen vorhandenen Programmen in der ersten Kommentarzeile des Kommentarteils (s. auch Abschnitt 2.2, S. 7) dieser Programme nach dem angegebenen Stichwort `Stichwort` gesucht und angegeben in welchen Programmen es vorkommt. Ein Beispiel:

```
>> lookfor integral
ELLIPKE Complete elliptic integral.
EXPINT Exponential integral function.
DBLQUAD Numerically evaluate double integral.
INNERLP Used with DBLQUAD to evaluate inner loop of integral.
QUAD   Numerically evaluate integral, low order method.
QUAD8  Numerically evaluate integral, higher order method.
>>
```

3. `type Name` Mit diesem Befehl erscheint das Programm `Name`, wenn es als Datei mit Erweiterung `.m` vorliegt vollständig auf dem Bildschirm. Wir geben zwei Beispiele an:

   Beispiel 1.5. *Ein mit* `type` *angezeigtes MATLAB-Programm*

```
>> type quad

function [Q,cnt] = quad(funfcn,a,b,tol,trace,varargin)
%QUAD   Numerically evaluate integral, low order method.
%   Q = QUAD('F',A,B) approximates the integral of F(X) from A to B to
%   within a relative error of 1e-3 using an adaptive recursive
%   Simpson's rule.  'F' is a string containing the name of the
%   function.  Function F must return a vector of output values if given
%   a vector of input values.  Q = Inf is returned if an excessive
%   recursion level is reached, indicating a possibly singular integral.
%
%   Q = QUAD('F',A,B,TOL) integrates to a relative error of TOL.  Use
%   a two element tolerance, TOL = [rel_tol abs_tol], to specify a
%   combination of relative and absolute error.
%
%   Q = QUAD('F',A,B,TOL,TRACE) integrates to a relative error of TOL and
%   for non-zero TRACE traces the function evaluations with a point plot
%   of the integrand.
%
%   Q = QUAD('F',A,B,TOL,TRACE,P1,P2,...) allows parameters P1, P2, ...
%   to be passed directly to function F:   G = F(X,P1,P2,...).
%   To use default values for TOL or TRACE, you may pass in the empty
%   matrix ([]).
%
%   See also QUAD8, DBLQUAD.

%   C.B. Moler, 3-22-87.
%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.16 $  $Date: 1997/12/02 15:35:23 $

% [Q,cnt] = quad(F,a,b,tol) also returns a function evaluation count.

if nargin < 4,   (...Rest weggelassen)
```

   Ein nicht in MATLAB geschriebenes Programm erzeugt die folgenden Zeilen:

   Beispiel 1.6. *Ein mit* `type` *nicht anzeigbares MATLAB-Programm*

```
>> type sin
sin is a built-in function.
```

4. `demo`: Hier kann man sich über eine Menüsteuerung sehr viele Programme (besonders graphische) auch aus den vorhandenen `toolboxes` vorführen lassen. Das sind Programmsammlungen mit besonderen Themen.

2 Trennung von Anweisungen und Anbringung von Kommentaren

Ein Programm besteht unabhängig von der gewählten Sprache aus einer Folge von Anweisungen, die auch *Befehle* oder *Kommandos* heißen. Eine Anweisung sagt dem Programm was es tun soll. Daneben kann und sollte das Programm Kommentare enthalten, die den Programmablauf nicht beeinträchtigen, aber das Programm für einen menschlichen Leser besser verständlich machen und zum Beispiel Informationen über den Namen des Programmierers und das Datum der Programmherstellung (Beginn und letzte Änderung) enthalten. Da wir heutzutage dem Computer Programme aller Art in Form von Dateien zur Verarbeitung übergeben und zum Beispiel nicht mehr in Form von Lochkarten, Lochstreifen oder Magnetbändern müssen wir wissen, wie wir die Anweisungen in einer Datei anordnen und voneinander trennen. Auch müssen wir wissen, wie wir Kommentare von Anweisungen unterscheiden können. Wir unterstellen hier, daß Sie wissen, wie man eine Datei in einem Computer (mit Hilfe eines *Editors*) anlegt, verändert, kopiert, mit e-mail verschickt, und auch zu einem späteren Zeitpunkt wiederfindet, etc. Wichtig ist zu wissen, daß eine Datei aus *Zeilen* aufgebaut ist, wobei das Zeilenende durch ein unsichtbares aber doch vorhandenes Zeichen gekennzeichnet wird und durch Drücken der Enter-Taste auf dem Computer erzeugt wird. Diese Taste ist meistens größer als eine Buchstabentaste und rechts auf der Tastatur angeordnet und meistens mit einem Pfeil gekennzeichnet. Manche Editoren beginnen scheinbar eine neue Zeile, wenn man an das Ende des Dateifensters kommt, ein Zeilenendezeichen wird aber trotzdem nicht eingesetzt. Unter einer Anweisung verstehen wir im Moment nur eine nach noch zu lernenden Regeln aufgebaute Zeichenfolge. Diese Zeichenfolge kann sehr kurz aber auch sehr lang sein.

2.1 Trennung von Anweisungen und Kommentare in FORTRAN

Wir beschreiben nur FORTRAN ab Version 90. Alle Versionen bis FORTRAN 77 sind lochkartenorientiert und erfordern eine genaue Anpassung der Anweisungen auf die 80 Spalten einer Lochkarte. Davon wollen wir hier aber absehen.

1. Regel: Anweisungen werden durch das Zeilenende getrennt.

2. Regel: Wollen wir mehrere Anweisungen in eine einzige Zeile schreiben, so sind diese Anweisungen durch ein Semikolon zu trennen. Der letzten Anweisung folgt nicht notwendig ein Semikolon.

3. Regel: Will man eine einzige Anweisung sich über mehrere Zeilen erstrecken lassen, so beende man die Zeile mit einem & (*Ampersand*) und setze die Anweisung in der nächsten Zeile fort. So kann sich eine einzige Anweisung über mehrere Zeilen erstrecken. Beispiele kommen in Programm 1.2, Zeile 25 vor. Bei der Fortsetzung von Texten (Strings) über mehrere Zeilen muß auch die Anzahl der Leerzeichen am Anfang einer Zeile durch & am Anfang der Zeile gesteuert werden. Beispiel:

   ```
   langer_Text=                          &
   'This book is a systematic exposition&
   & of the part of general topology&
   & which has  proven useful in several&
   & branches of mathematics.&
   &   (John L. Kelley [1955]: General Topology)'
   ```

   Ein Beipiel (in Form eines Gedichts) haben wir bei der ersten Einführung von Textvariablen (Strings), Programm 5.6, Seite 13 angegeben.

Kommentare beginnen mit einem Ausrufezeichen ! und gelten von da an bis zum Zeilenende. Beispiel:

```
!Alles was hier steht, ist ein FORTRAN-Kommentar.
```

2.2 Trennung von Anweisungen und Kommentare in MATLAB

Die entsprechenden Regeln für MATLAB lauten:

1. Regel: Anweisungen werden durch das Zeilenende getrennt.
2. Regel: Wollen wir mehrere Anweisungen in eine einzige Zeile schreiben, so sind diese Anweisungen durch ein Semikolon oder durch ein Komma zu trennen. Der letzten Anweisung in einer Zeile können ein Semikolon, Komma oder Leerzeichen folgen.
3. Regel: Soll sich eine einzige Anweisung über mehrere Zeilen erstrecken, so ist sie (an geeigneter Stelle) durch drei Punkte ... zu unterbrechen und auf der nächsten Zeile fortzusetzen.

Neben der trennenden Wirkung von Komma und Semikolon gibt es einen wesentlichen Unterschied. Wird eine Anweisung durch ein Komma oder durch ein Zeilenende abgeschlossen, so wird das Ergebnis dieser Anweisung auch (soweit sinnvoll) auf dem Bildschirm angezeigt (es findet ein *Echo* auf dem Bildschirm statt), wie in dem folgenden Beispiel.

```
>> x=pi, y=sin(x)
x =
    3.1416
y =
   1.2246e-16
>>
```

Die Anweisungen `x=pi; y=sin(x);` würden dieselben Zuweisungen bewirken, aber die Ergebnisse nicht auf dem Bildschirm anzeigen. Gleichzeitig lernen wir, daß es eine vorgefertigte Variable `pi` mit dem Wert π gibt, daß aber $\sin \pi$ in MATLAB nur näherungsweise mit Null übereinstimmt. Auch sehen wir, daß die Zahlen in einem vorgegebenen, festen *Format* ausgegeben werden. Über andere Formate sprechen wir an anderer Stelle.

Kommentare werden prinzipiell wie in FORTRAN, aber durch ein %-Zeichen (Prozentzeichen) eingeleitet und gelten nur für eine Zeile. Beispiel:

```
%Alles was hier steht, ist ein MATLAB-Kommentar.
```

Die in MATLAB vorkommenden Kommentare haben aber eine weitere, sehr nützliche Funktion. Bei Verwendung der `help Name`-Funktion (s. S. 4) wird unter allen Programmen (auch den selbst geschriebenen) nach dem Programm mit dem Namen `Name` gesucht und der gesamte Kommentar am Anfang der Programms, der in der ersten Spalte mit einem Kommentarzeichen % beginnt als Information ausgegeben. Darüberhinaus sucht die `lookfor Stichwort`-Funktion in allen ersten Kommentarzeilen nach diesem `Stichwort`.

3 Rahmenbedingungen

Die *Rahmenbedingungen* sagen, was minimal gebraucht wird, um ein syntaktisch richtiges Programm zu schreiben.

3.1 Rahmenbedingungen für FORTRAN

Ein FORTRAN-Programm besteht mindestens aus einer Kopfzeile und einer Schlußzeile nach folgendem Muster:

Beispiel 3.1. *FORTRAN-Rahmen*

```
program Name
  ...(Deklarations- und Anweisungsteil)
end program Name
```

Das Wort `Name` ist ein Platzhalter für einen beliebigen Namen. Er hat keinen Einfluß auf den Programmablauf. Ein Programm kann auch so aufgebaut sein:

Beispiel 3.2. *FORTRAN-Rahmen mit Unterprogramm*

```
program Name1
  ...(Deklarations- und Anweisungsteil)
end program Name1

subroutine Name2(Parameterliste)
  ...(Deklarations- und Anweisungsteil)
end subroutine Name2

function Name3(Parameterliste) result(Parameter)
  ...(Deklarations- und Anweisungsteil)
end function Name3
```

Es können mehrere Unterprogramme vorkommen. Es gibt neben `function` und `subroutine` noch weitere Unterprogrammtypen, auf die wir hier aber nicht eingehen können.

3.2 Rahmenbedingungen für MATLAB

Ein MATLAB-Programm braucht keinen Rahmen. Es kann also sofort in das MATLAB-Fenster irgendeine Anweisung geschrieben werden, die auch sofort (nach Betätigen der Enter-Taste, s. S. 6) ausgeführt wird. Unterprogramme müssen in separate Dateien (mit Erweiterung `.m`) geschrieben werden. Sie können in einem Hauptprogramm durch ihren Dateinamen (ohne Erweiterung) aufgerufen werden. Es gilt folgende Sonderregel: Eine separat zu schreibendes Unterprogramm in der Form einer Funktion (darauf kommen wir zurück) darf weitere derartige Unterprogramme (also in Form einer Funktion) enthalten, die aber nur in dem ersten Unterprogramm aufgerufen werden können und von außen nicht zugänglich sind.

4 Namen von Variablen

Variable sind Platzhalter (also Bezeichnungen für Speicherplätze), die durch besondere *Namen* gekennzeichnet werden. Namen von Variablen sind Folgen von Buchstaben und Ziffern mit der zusätzlichen Regel, daß an der ersten Stelle ein Buchstabe stehen muß. Ein *Buchstabe* ist eines der folgenden 52 Zeichen:

`a,b,...,z,A,B,...,Z.`

Unter *Ziffer* verstehen wir ein Zeichen aus der Menge von 10 Zeichen:

`0,1,...,9.`

Zusätzlich zu den Buchstaben und Ziffern kann noch ein Unterstreichungszeichen _ benutzt werden um die Namen etwas besser lesbar zu machen. Beispiele für Namen von Variablen:

`a, X, bb2, bB2, Bb2, BB2, langer_Name, noch_etwas_laenger`

In beiden Programmsprachen sind die *Wortlängen* für Variablennamen begrenzt mit dem folgenden Unterschied: In FORTRAN gibt es bei Überschreitung der maximalen Wortlänge eine Fehlermeldung, in MATLAB-Programmen dagegen werden die überzähligen Zeichen schlicht ignoriert. FORTRAN- und MATLAB-Variable erlauben die gleiche maximale Wortlänge von 31 Zeichen. In FORTRAN jedoch werden große und kleine Buchstaben identifiziert, in MATLAB werden sie unterschieden. Ein Beispiel für ein fehlerhaftes FORTRAN-Programm, verursacht durch Benutzung der gleichen Variablen in Klein- und Großschreibung ist Programm 4.1. Alle Zeichen, die von Ziffern und Buchstaben verschieden sind, heißen *Sonderzeichen*. Außer dem Unterstreichungszeichen _ dürfen Sonderzeichen, insbesondere das *Leerzeichen* (=Zwischenraum) und *Umlaute* in Namen nicht vorkommen. Bei Namen von Dateien gibt es eine Sonderregel. Sie dürfen eine durch einen Punkt abgetrennte *Erweiterung* (*Extension*) tragen. Beispiele für Dateinamen:

`x, x1.y1, FORTRAN_Programm.f, v_w.x_y, MATLAB_Programm.m.`

Bei manchen FORTRAN-Installationen werden die angegebenen Regeln nicht vollständig geprüft. Wechselt man mit derartigen Programmen (oder Dateien) den Rechner, kann man böse Überraschungen erleben.

Programm 4.1. Groß-und Kleinschreibung in FORTRAN-Programmen

```
1  program Gross_und_Kleinschreibung
2    implicit none
3    integer :: s, x, X
4    x=1; X=2
5    s=x+X
6    write(*,fmt=*) s
7    !Fehlermeldung:
8    !  integer :: s, x, X
9    !                 ^
10   !cf90-554 f90comp: ERROR GROSS_UND_KLEINSCHREIBUNG, File = gr_u_klein.f90,
11   !Line = 3, Column = 17
12   !  "X" has the INTEGER attribute.
13   !It must not be given the INTEGER attribute again.
14 end program Gross_und_Kleinschreibung
```

5 Deklaration von Variablen

In den verschiedenen Programmiersprachen muß entweder explizit oder implizit mitgeteilt werden, welche Bedeutung verwendete Variablennamen haben. Es muß z. B. mitgeteilt werden, ob eine Variable eine reelle Zahl, eine ganze Zahl, eine komplexe Zahl, eine logische Variable, ein Vektor, eine Matrix, oder ein Text, usw. ist. Dieser Teil ist besonders für FORTRAN schwierig, weil es eine große Anzahl von Möglichkeiten gibt.

5.1 Deklaration von Variablen in FORTRAN

In FORTRAN kann zwischen ganzzahligen *Festkommazahlen*, sog. `integer` und *Gleitkommazahlen* unterschieden werden. Gleitkommazahlen gibt es darüberhinaus in einfacher und doppelter Genauigkeit, manchmal auch in vierfacher Genauigkeit. Für Festkommazahlen kann - compilerabhängig - ggf. auch eine Genauigkeit eingestellt werden. Will man beispielsweise nur mit sehr kleinen Zahlen arbeiten, kann es einen Sinn haben, z. B. nur mit zweistelligen ganzen Zahlen zu arbeiten. Wir kommen darauf zurück, S. 12. Es ist zu beachten, daß *doppelt genau* in anderen Programmiersprachen (Pascal, MATLAB) die übliche, voreingestellte Genauigkeit ist. Werden Variable nicht in einer besonderen Anweisung deklariert, so wird die Bedeutung aus dem ersten Buchstaben des Namens abgelesen. Alle Variablen, die mit den Buchstaben

```
i, j, k, l, m, n, I, J, K, L, M, N
```

beginnen, werden - wenn nichts anderes gesagt wird - als ganze Festkommazahlen verstanden, alle übrigen als Gleitkommazahlen in einfacher Genauigkeit. Um Klarheit über die Typvereinbarung zu haben, ist es jedoch zweckmäßig, in *jedem* FORTRAN-Programm von dieser Voreinstellung abzusehen. Das geschieht in der Form

```
implicit none
```

Damit ist man gezwungen, *jeder* im Programm vorkommenden Variablen einen Typ zuzuordnen. Das `implicit none`-statement gehört auch in Unterprogramme hinein. Man kann das `implicit`-statement auch zur Globalvereinbarung gewisser Variablennamen benutzen. Diese Technik wird jedoch nicht empfohlen. Mit dem Befehl `implicit double precision (a-h,o-z)` werden z. B. alle Variablen, die mit den Buchstaben `a` bis `h` und `o` bis `z` beginnen zu doppelt genauen Variablen (also reelle Gleitkommazahlen mit etwa 16 Stellen). Den Unterschied zwischen einfach- und doppelt genauen Gleitkommazahlen kann man aus dem nachfolgenden kleinen Programm 5.1 gut erkennen.

Programm 5.1. Einfache und doppelte Genauigkeit in FORTRAN

```
1  program Test_mehrfache_Genauigkeit
2    implicit none
3    real :: x;   double precision :: y
4    x=4.0/3.0; y=10.0d0/7.0d0  !das Anhaengen von d0 signalisiert
5                               !doppelte Genauigkeit (~16 Stellen)
6    write(*,fmt=*) x  !Der erste Stern * heisst Bildschirmausgabe
7                      !fmt=* heisst Ausgabe in Standardformat
8    write(*,fmt=*) x*3.d0
9    write(*,fmt=*) 4/3
10   write(*,fmt=*) y
11   ! Die Ergebnisse sind:
12   !-------------------------------------------
13   ! 1.3333337          ( 7 richtige Stellen)
14   ! 4.0000001192092896 ( 7 richtige Stellen)
15   ! 1 (ganzzahlige Division, der Rest faellt weg)
16   ! 1.4285714285714286 (17 richtige Stellen)
17   !-------------------------------------------
18   ! 1 2345678901234567 (zur Numerierung)
19 end program Test_mehrfache_Genauigkeit
```

Explizite Deklarationen, die am Programmanfang stehen müssen, kann man aus den Beispielprogrammen 5.2 bis 5.4 entnehmen. Kommen Deklarationen vom Typ

```
real, dimension(4) :: x
```

vor, so kann eine Besetzung im Programm in der Form

```
x=(/2.4,-5.2,13.2,-0.8/)
```

vorgenommen werden. Der Parameter `x` darf auch direkt in `write`-Befehlen erscheinen. Ist ein Feld zweidimensional also z. B. `real, dimension(m,n) :: A`, so kann man `A` in der folgenden Form besetzen:

```
A(1,1:n)=(/a11,a12,...,a1n/)
A(2,1:n)=(/a21,a22,...,a2n/) ...

A(m,1:n)=(/am1,am2,...,amn/)
```

In FORTRAN sind auch negative Indizes erlaubt. Die Vereinbarung muss dann die folgende Form haben:

```
real, dimension(-3:4) :: x
```

Entsprechendes gilt für mehrdimensionale Felder.

Texte (Strings) werden in den folgenden Formen deklariert:

`character(len=30) :: wort` oder `character(len=15), dimension(5) :: text_feld`

Im ersten Fall wird eine Textvariable `wort` deklariert, die (maximal) 30 Zeichen enthält. Im zweiten Fall wird ein eindimensionales Feld der Länge 5 deklariert, wobei jede der 5 Komponenten einen Text bis zu 15 Zeichen aufnehmen kann. Bei der Besetzung müssen jedoch alle 5 Komponenten genau gleich viele Zeichen (≤ 15) enthalten. Die einzelnen Zeichen der Variablen `wort` können in der Form `wort(3:8)` benutzt werden. Die mittleren drei Zeichen der dritten Komponente von `text_feld` kann man mit `text_feld(3)(7:9)` extrahieren. Die erste Klammer enthält die Komponentennummer, die zweite Klammer die Nummern der auszuwählenden Zeichen.

Programm 5.2. Einfache Variable in einem FORTRAN-Programm

```
1  program Typ_Deklaration_einf_Variable
2    implicit none
3    integer :: i
4    real :: x
5    double precision :: x2
6    complex :: c
7    logical :: l
8    character(len=14) :: Text
9    i=3; x=1.0/7.0; x2=1.0d0/7.0d0; c=(3.2,-4.1); l=.true.; Text='FORTRANuMATLAB'
10   write(*,fmt=*) i,x,x2,c
11   write(*,fmt=*) l,' ',Text
12   !Ergebnis:  3,  0.142857149,  0.14285714285714285,  (3.20000005,-4.0999999)
13   !           T FORTRANuMATLAB
14 end program Typ_Deklaration_einf_Variable
```

Programm 5.3. Feld-Variable in einem FORTRAN-Programm

```
1  program Typ_Deklaration_dim_Variable
2    implicit none
3    integer, dimension(4) :: i
4    double precision, dimension(3,2) :: x
5    character(len=6) :: ch1, ch2
6    character(len=10), dimension(3) :: wort
7   !-----------------------------------------------------
8    wort=(/'langweilig','kurzweilig','zweiteilig'/)
9    ch1=wort(2)(1:4)   !Zeichen 1 bis  4 der 2. Komponente
10   ch2=wort(3)(5:10)  !Zeichen 5 bis 10 der 3. Komponente
11   write(*,fmt=*) ch1,' ',ch2  ! kurz   teilig
12   !-----------------------------------------------------
13   i=(/-1,-2,-3,-4/)
14   write(*,fmt=*) ' '  !leere Zeile
15   write(*,fmt=*) i    !-1,  -2,  -3,  -4
16   !-----------------------------------------------------
```

```
  x(1,1:2)=(/1.0d0,2.0d0/)
  x(2,1:2)=(/3.0d0,4.0d0/)
  x(3,1:2)=(/5.0d0,6.0d0/)
  !-------------------------------------------------------
  write(*,fmt=*) ' '  !leere Zeile
  write(*,fmt=*) x
  !1.,  3.,  5.,  2.,  4.,  6. !spaltenweise, ohne Zeilenumbruch
  !-------------------------------------------------------
  write(*,fmt=*) ' '  !leere Zeile
  write(*,fmt="(2(f4.2,' '))") x(1,1:2)
  write(*,fmt="(2(f4.2,' '))") x(2,1:) !von 1 bis zum Ende
  write(*,fmt="(2(f4.2,' '))") x(3,:2) !vom Anfang bis 2
  !1.00 2.00
  !3.00 4.00
  !5.00 6.00     !gibt Originalmatrix
  !-------------------------------------------------------
  write(*,fmt=*) ' '  !leere Zeile
  write(*,fmt="(2(f4.2,' '))") transpose(x)
  !1.00 2.00
  !3.00 4.00
  !5.00 6.00     !gibt Originalmatrix
end program Typ_Deklaration_dim_Variable
```

Über die Zuordnung von Daten zu den entsprechenden Feldelementen vergleiche man den Abschnitt 9, auch Programm 9.1 auf Seite 30.

Programm 5.4. Selbst definierte Typ-Deklarationen in einem FORTRAN-Programmen

```
program Typ_Deklaration_eigene_Typen
  implicit none
  !integer, parameter :: dr = selected_real_kind(14) !7Bytes=56Binaerstellen
  integer, parameter :: dr = selected_real_kind(2*precision(1.0)) !2-fach
  real(kind=dr) :: x1, x2
  complex(kind=dr) :: c
  x1=1.0_dr/7.0_dr; x2=3.0_dr/7.0_dr
  c=dcmplx(x1,x2) !das d ist wichtig!
  write(*,fmt=*) c ,x1, x2, c**2 !alles in eine Zeile
  !(0.14285714285714285,0.42857142857142855),  !c
  ! 0.14285714285714285,0.42857142857142855,   !x1, x2
  !(-0.16326530612244897,0.12244897959183672)  !c**2
end program Typ_Deklaration_eigene_Typen
```

Da Konstanten nicht explizit definiert werden, muß man ihnen ansehen können, welche Bedeutung sie im Rahmen der möglichen Deklarationsformen haben. Wir machen dazu eine kleine Tabelle:

Tabelle 5.5. Identifizierung von Konstanten in FORTRAN

1. `21`: ganze Zahl, Typ `integer`,
2. `21.3`: reelle, einfach genaue Zahl, Typ `real`,
3. `1.3d0`: reelle, doppelt genaue Zahl, Typ `double precision`. Die Endung `d0` ist voreingestellt,
4. `21.3_qr`: reelle, vierfach genaue Zahl, Typ `real(kind=qr)`. Die Endung `_qr` ist nicht voreingestellt, sondern muß im Programm definiert werden,
5. `(21.4,-13.8)`: komplexe Zahl, jede Komponente ist einfach genau, Typ `complex`,
6. `(21.4_dr,-13.8_dr)`: komplexe Zahl, doppelt genau, Typ `complex (kind=dr)`. Die Endung `_dr` ist nicht voreingestellt, sondern muß im Programm definiert werden,
7. `'Wie ist das Wetter heute?'`, alternativ `"Wie ist das Wetter heute?"`: Typ `character`,
8. `.true., .false.`: logische Konstanten, Typ `logical`.

Um die Endungen in 4. und 6. zu verstehen, muß man die folgende Deklarationsart verstehen, die allerdings erst ab FORTRAN95 funktioniert. Sie hat große Ähnlichkeit mit der `type`-Deklaration von Pascal. Man schreibt in den Deklarationsteil:

```
integer, parameter :: ns = selected_int_kind(n)
```

```
integer, parameter :: xs = selected_real_kind(n)
```

und definiert damit nicht eine Variable, sondern einen neuen, ganzzahligen, bzw. reellen Variablentyp mit `4n` Binärstellen bzw $n/2$ Bytes. Unter einem Byte versteht man 8 Binärstellen. Dabei ist für `n` eine ganzzahlige, positive Konstante (mit gewissen Beschränkungen) einzusetzen und `ns`, bzw. `xs` ist ein frei zu wählender Name. Da Rechner meistens in Bytes rechnen, ist es naheliegend, daß `n` eine gerade Zahl sein sollte. Mit `4n` Binärstellen kann man z. B. die 2^{4n} ganzen Zahlen von $-(2^{4n-1}-1)$ bis 2^{4n-1} darstellen. Bei $n=2$ sind das die 256 Zahlen $-127,-126,\ldots,-1,0,1,\ldots 128$. Um mit dieser Deklaration arbeiten zu können, deklariert und besetzt man ganzzahlige, bzw. reelle Variablen `u, v` in der Form.

```
integer(kind=ns) :: u
real(kind=xs) :: v
  ...
u=1_ns; v=3.4_xs
```

Konstante dieses Typs werden entsprechend der Tabelle 5.5 durch Anhängen von `_ns, _xs` als ganzzahlige, bzw reelle Konstante vom oben definierten Typ definiert. Für reelle Konstanten mit doppelter Genauigkeit (`double precision`) gibt es auch die voreingestellte Standard-Endung `d0`. Die FORTRAN-Zahl `2.34d0` ist z. B. die doppelt genaue Konstante 2.34. Für die obige Typ-Deklaration `selected_real_kind` existiert auch die alternative mit FORTRAN95 funktionierende Form

```
integer, parameter :: xn = selected_real_kind(n*precision(1.0))
```

(auch `n*precision(1.0d0)` ist möglich). Textvariable können in FORTRAN in der Form `Text='Wort'` und in der Form `Text="Wort"` besetzt werden. Das erlaubt z. B. den Text `"Wie geht's"` mit einem Apostroph im Text.

Ein Beipiel für einen String-Array, der ein Gedicht enthält, haben wir im folgenden FORTRAN-Programm 5.6 angegeben.[1]

Programm 5.6. String-Array mit Gedicht als Text (FORTRAN)

```
1  program Gedicht_schreiben
2    implicit none
3    character(len=50), dimension(15) :: Gedicht
4    Gedicht=(/                                          &
5    "                                                  ",&
6    "                 Die Trichter                     ",&
7    "                                                  ",&
8    "   Zwei Trichter wandeln durch die Nacht.         ",&
9    "    Durch ihres Rumpfs verengten Schacht          ",&
10   "         fliesst weisses Mondlicht                ",&
11   "              still und heiter                    ",&
12   "                 auf ihren                        ",&
13   "                  Waldweg                         ",&
14   "                   u. s.                          ",&
15   "                     w.                           ",&
16   "                                                  ",&
17   " Christian Morgenstern, Galgenlieder (1905)      ",&
18   "               Serie Piper, Muenchen, 1982        ",&
19   "                                                  "/)
20   write(*,fmt="(a43)") Gedicht
21 end program Gedicht_schreiben
```

Programm 5.7. Ausdruck des Gedicht-Programms (FORTRAN)

```
1
2                 Die Trichter
3
4    Zwei Trichter wandeln durch die Nacht.
5     Durch ihres Rumpfs verengten Schacht
6          fliesst weisses Mondlicht
7               still und heiter
8                  auf ihren
9                   Waldweg
10                   u. s.
11                     w.
12
13  Christian Morgenstern, Galgenlieder (1905)
14                Serie Piper, Muenchen, 1982
```

[1] In Zeile 10 des Programms 5.6 bzw. in Zeile 6 des Programms 5.7 steht natürlich „fließt weißes Mondlicht“ aber das „ß“ wird in der angegebenen Umgebung nicht richtig wiedergegeben, dasselbe gilt für das „ü“ in „München“.

Als Analogon zu dem Pacal-Datentyp `record` gibt es auch in FORTRAN eine Möglichkeit eigene Datentypen über das Schlüsselwort `type` zu erfinden. Wir zeigen das an einem Programmbeispiel 5.8. Zuerst wird dort ein Datentyp `Person` eingeführt, dann werden Variable als `Person` deklariert.

Programm 5.8. FORTRAN-Datentyp `type`

```
 1 program type_testen
 2   integer, parameter :: kurz=selected_int_kind(2)
 3   integer :: summe
 4   type Person
 5     character(len=50) :: Name
 6     integer           :: Alter
 7     real              :: Groesse
 8     integer(kind=kurz):: Geschlecht
 9   end type Person
10   type (Person) :: Er, Sie
11   Er=Person('Fritz Meyer',12,1.79,2_kurz)
12   Sie=Person('Marianne Schultz',29,1.81,1_kurz)
13   summe=Er%Alter+Sie%Alter !gibt 41
14   write(*,fmt=*) summe
15   write(*,fmt=*) Er
16   write(*,fmt="(a16,i3,f5.2,i2)") Sie
17 end program type_testen
18 !Ergebnis:
19 ! 41
20 ! Fritz Meyer                                        12,  1.78999996,  2
21 !Marianne Schultz 29 1.81 1
```

5.2 Deklaration von Variablen in MATLAB

In MATLAB sind alle Variablen Gleitkommazahlen mit derselben *Wortlänge* (gleiche Anzahl von Ziffern in der Gleitkomma-Darstellung), und die Deklaration geschieht implizit durch die Benutzung. Es können nicht nur einzelne Zahlen als Variable vorkommen, sondern auch ganze Felder von Zahlen, insbesondere *Vektoren und Matrizen*. Felder werden in Programmiersprachen auch gern *arrays* genannt. Da eine explizite Deklaration von Feldern (im Regelfall) nicht vorkommt, gibt es folgende generelle Vereinbarung:

MATLAB-Regel: Alle Laufvariablen (Indizes) starten mit der Nummer Eins.

Es gibt also keine negativen Indizes, und es gibt auch nicht den Index Null. Das ist in manchen Situationen unbequem.

Beispiele:

```
>> x1=1; x2=[1.2,2.3]; x3=[1 2 3;4 5 6]; x4=2+3i;
>> x1, x2, x3, x4
x1 =
     1
x2 =
    1.2000    2.3000
x3 =
     1     2     3
     4     5     6
x4 =
   2.0000 + 3.0000i
```

Im obigen Beispiel ist also `x3(1,1)=1,x3(2,3)=6`. Hier *muß* zur Trennung der Indizes ein Komma stehen. In der Definition von Feldern (z. B. Vektoren, Matrizen, Tensoren) haben die Trennzeichen *Komma* und *Leerzeichen* dieselbe Bedeutung, sie trennen *neben*einander liegende Zahlen voneinander ab, wohingegen das *Semikolon* und das *Zeilenende unter*einander stehende Zahlen voneinander abtrennen. Zu beachten ist, daß die Felder immer rechteckig sein müssen. Bei der Deklaration von `x3` beispielsweise müssen also in jeder Zeile gleichviele Zahlen stehen. Zur Deklaration von Feldern werden rechteckige Klammern verwendet, zum Aufrufen dagegen runde. Deklarationen sind auch von folgendem Typ möglich:

```
>> x5=[1:4]; x6=[1:0.5:2;2:4]; Text='Dies ist ein String';
>> x5, x6, x6(1,:), Text
x5 =
     1     2     3     4
x6 =
    1.0000    1.5000    2.0000
    2.0000    3.0000    4.0000
ans =
    1.0000    1.5000    2.0000
Text =
Dies ist ein String
>>
```

Ein String ist also ein (fast) beliebiger Text eingeschlossen in ' ', wobei das Zeichen ' nicht vorkommen darf. Der String ist aufzufassen als ein Feld (*array*). Die Länge dieses Feldes ist die Anzahl der Zeichen, die im String vorkommt. Im angegebenen Beispiel sind das 19 Zeichen. Die einzelnen Zeichen des Strings können dann in der Form `Text(5)` adressiert werden (im Beipiel ist das ein Leerzeichen). Vektoren können also auch in der Form `v1=[m:n]` oder in der Form `v2=[x:step:y]` deklariert werden. Die Bedeutung ergibt sich aus den Beispielen. Die Größe `step` in der Deklaration von `v2` darf auch negativ und nicht ganzzahlig sein. Ist in `v1=[m:n]` die Zahl `m` größer als `n`, so wird ein leeres Feld definiert, mit dem man auch gewisse Operationen ausführen kann. Eine sehr bequeme Technik Vektoren zu erzeugen, geschieht in der Form

`x=linspace(a,b,m)` oder `x=linspace(a,b)`.

Ist `m<=1`, so ist `x=b`. Sei also `m>1`. Unter der Voraussetzung, daß `a ~= b` (ungleich) werden im Intervall von `a` nach `b` (oder umgekehrt) `m` äquidistante (gleichabständige) Punkte definiert, d. h. `x` besteht aus den Komponenten

`x(j)=a+(j-1)*(b-a)/(m-1), j=1,2,...,m.`

Ist `a=b`, so sind alle `m` Komponenten von `x` gleich `a`. Kommt `m` im Aufruf nicht vor, so wird `m=100` gesetzt.

Die Benutzung der leeren Matrix `[]` ergibt sich aus dem folgenden Beispiel:

```
A=[];B=[];
for j=1:3
  b=j;
  A=[A,b];
  B=[B;b];
end; %for
```

Es werden also sukzessive die Vektoren `A=[],B=[],A=[1],B=[1],A=[1,2],B=[1;2],A=[1,2,3],B=[1;2;3]` erzeugt. `A` ist also ein Zeilen- und `B` ein Spaltenvektor. Hat man in MATLAB ein Feld vom Typ `a(2,3,4)` deklariert (z. B. durch direkte Besetzung wie im Programm 9.1, S. 30), so ist die Standardanzeige :

```
a(:,:,1) =
     1     3     5
     2     4     6
a(:,:,2) =
     7     9    11
     8    10    12
a(:,:,3) =
    13    15    17
    14    16    18
a(:,:,4) =
    19    21    23
    20    22    24
```

Unabhängig von dem Anzeigeformat auf dem Bildschirm wird in MATLAB immer mit der gleichen Anzahl von Stellen (ca. 16) gerechnet. Eine doppelte oder höhere (oder auch niedrigere) Genauigkeit gibt es in MATLAB nicht. Die voreingestellte Genauigkeit entspricht der doppelten Genauigkeit von FORTRAN.

6 Unbedingte und bedingte Anweisungen

Wegen der geringfügigen Unterschiede behandeln wir FORTRAN und MATLAB in einem Kapitel.

6.1 Unbedingte Anweisungen

Unbedingte Anweisungen haben (in FORTRAN und in MATLAB) formal die Gestalt

`a=b` in MATLAB auch `b` alleine möglich.

Dabei ist `a` eine Variable und `b` ein Ausdruck, also entweder eine Konstante, eine Variable oder eine Rechenvorschrift, die schließlich einen Wert liefert, der auf `a` abgelegt wird. Das Gleichheitszeichen hat die Funktion eines Zuordnungszeichens, nicht die Funktion eines mathematischen Gleichheitszeichens. Der Wert von `b` wird auf den Speicher, der durch `a` bezeichnet wird, abgelegt. Damit die Zuordnung ausgeführt werden kann, muß `b` vom gleichen Datentyp wie `a` sein. Steht in MATLAB nur `b`, also kein Gleichheitszeichen, so wird b ausgerechnet und einer temporären Variablen `ans` zugeordnet. Für `b` können Ausdrücke eingesetzt werden die mit Hilfe der vier Grundrechenarten

$+,\ -,\ *,\ /$ (in dieser Schreibweise für FORTRAN und MATLAB)

und dem Zeichen für Exponentiation aus anderen Ausdrücken zusammengesetzt werden können:

Exponentiation in FORTRAN : `**` (Beispiele `c**d`, `3.0**1.33`),
Exponentiation in MATLAB : `^` (Beispiele `c^d`, `3^1.33`).

In den Ausdrücken können auch Funktionsaufrufe vorkommen,

```
y=4*sin(1+a*b/c)
```

ist ein Beispiel für eine Anweisung in FORTRAN und in MATLAB mit einem Aufruf der Sinus-Funktion `sin`. Falls eine Division / im mathematischen Sinn nicht ausgeführt werden kann, weil durch Null geteilt wird, gibt es in FORTRAN und MATLAB verschiedene Reaktionen. Bei FORTRAN führt das zu einem Fehler mit Fehlermeldung und zum Abbruch, wie man an dem folgenden Beispiel sieht.

Beispiel 6.1. *FORTRAN-Programm mit Division durch Null*

```
1 program durch_Null_teilen
2   implicit none
3   real :: a,b,c,d
4   a=1.0; b=0.0; c=a/b; d=b/b;
5   write(*,fmt=*) c, d
6   !Fehlermeldung: Arithmetic Exception(coredump)
7   !keine Lokalisierung des Fehlers
8 end program durch_Null_teilen
```

Bei MATLAB gibt es eine Warnung: `Warning: Divide by zero.`, aber es wird weitergerechnet und dem Ergebnis ein neuer Datentyp `Inf` zugeordnet, falls die zu teilende Größe von Null verschieden ist. Falls auch die zu teilende Größe Null ist, gibt es dieselbe Warnung und das Ergebnis erhält den neuen Datentyp `NaN` (*not a number*). Wir sehen das auch an dem folgenden Beispiel:

```
>> a=1; b=0;
>> c=a/b, d=b/b
Warning: Divide by zero.
c =
   Inf
```

```
Warning: Divide by zero.
d =
   NaN
>>
```

Die beiden Vektorraumoperationen $\alpha\mathbf{A}$ und $\mathbf{A}+\mathbf{B}$ können in beiden Sprachen FORTRAN und MATLAB in gleicher Weise ausgeführt werden. Ist also α eine Zahl und `A` ein Vektor oder eine Matrix, so hat `B=`α`*A` die gleiche Größe wie `A`, und `B` wird durch elementweise Multiplikation von α mit den Elementen von `A` gewonnen. Sind `A,B` zwei gleich große Vektoren oder Matrizen, so ist `C=A+B` die elementweise gebildete Summe.

Bei Verwendung der vier Grundrechenarten und der Exponentiation in einem Ausdruck gibt es für MATLAB die Prioritätenregel: Zuerst werden die Exponentiationen ausgeführt, dann $*$ und $/$, dann $+$ und $-$, immer von links nach rechts. Beispiele, gerechnet mit MATLAB:

```
2^3^2=64, 2^(3^2)=512, 2^3*4=32, 2^(3*4)=4096, 3*4^2*5=240, 120/5/4/3=2, 120/5/(4/3)=18.
```

In FORTRAN erhält man dagegen für dieselben Rechnungen:

```
512          512          32          4096          240          2          24.
```

Wir sehen Unterschiede. Bei Verwendung der vier Grundrechenarten und der Exponentiation in einem Ausdruck gilt in FORTRAN: Zuerst werden die Exponentiationen von rechts nach links ausgeführt, dann $*$ und $/$, dann $+$ und $-$, von links nach rechts. Kommen Exponentiationen vor, so wird in FORTRAN anders als in MATLAB gerechnet. FORTRAN-Beispiel: `2**2**2**2=2**2**(2**2)=2**2**4=2**(2**4)=2**16=65536`, in MATLAB kommt dagegen 256 heraus. Der Unterschied im letzten Ergebnis (MATLAB: 18, FORTRAN: 24) hat eine andere Erklärung: Bei der Division wird in FORTRAN ganzzahlig gerechnet, wenn durch die Form der auftretenden Konstanten ganzzahlige Rechnung verlangt wird, also (4/3)=1.

Da MATLAB auch mit Matrizen, insbesondere mit Vektoren rechnen kann, werden die Grundrechenarten, wenn sinnvoll, übertragen. Ist also `A` ein (m,n)-Feld und `B` ein (n,p)-Feld, so ist `C=A*B` als gewöhnliche Matrixmultiplikation definiert, und für `C` kommt ein (m,p)-Feld heraus. Sind `x,y` zwei gleich lange Vektoren der Länge `n` als Spalten geschrieben, mit den Komponenten `x_1,x_2,...,x_n, y_1,y_2,...,y_n`, formal also ein (n,1)-Feld, so hat das *Skalarprodukt* `s:= x_1*y_1+x_2*y_2+`$\cdots$`+x_n*y_n` die einfache MATLAB-Form

```
s=x'*y
```

Dabei beschreibt `x'` den Übergang von der Spaltenform zur Zeilenform, d. h. `x'` als Matrix aufgefaßt ist ein (1,n)-Feld und damit ist `s=x'*y` ein gewöhnliches Matrixprodukt von zwei Matrizen der Größen (1,n) und (n,1), das Ergebnis also ein (1,1)-Feld, also eine Zahl. Ist `A` irgend ein (m,n)-Feld mit den Elementen a_{jk} (wir verwenden vorübergehend mathematische Symbole), so ist B:=A' ein (n,m)-Feld mit den Elementen $b_{kj} := \overline{a_{jk}}$, $j=1,2,\ldots,m,\ k=1,2,\ldots,n$. Ist a_{jk} reell, so stimmt $\overline{a_{jk}}$ mit a_{jk} überein, ist aber $a_{jk} := \alpha_{jk}+\mathrm{i}\beta_{jk}$ komplex, so ist $\overline{a_{jk}} := \alpha_{jk}-\mathrm{i}\beta_{jk}$. Bei reellen Matrizen nennt man den Übergang `A`$\rightarrow$`A'` *Stürzen* (oder Transponieren) der Matrix `A`. Der Übergang von einer komplexen Zahl a zur komplexen Zahl $\overline{a}$ nennt man *Konjugieren*, und $\overline{a}$ heißt auch die zu a komplex konjugierte Zahl.

Will man aber eine Matrix `A` *elementweise* quadrieren, so geht das mit `A.^2`. Entsprechend sind auch `.*` und `./` als elementweise Operationen definiert. Es gibt also in MATLAB auch noch die elementweise zu verstehenden Punkt-Operationen

```
.*      ./      .^
```

Beispiel 6.2. *MATLAB-Operation* `.^2` *im Vergleich zu* `^2`

```
A = 1.1000    2.2000   A.^2=  1.2100    4.8400   A^2=A*A=   8.4700   12.1000
    3.3000    4.4000          10.8900   19.3600              18.1500   26.6200
```

Die MATLAB-Operationen mit dem Punkt werden auch *Kindergartenoperationen* genannt, weil sie so einfach zu erklären und auszuführen sind. Das Lösen eines linearen Gleichungssystems `Ax=b` kann man formal auffassen als Linksdivision von `b` durch `A`, in Zeichen

```
x=A\b.
```

Das funktioniert auch, wenn `A, b` komplexe Einträge haben.

In FORTRAN werden Matrixmultiplikationen nicht unterstützt. Es gibt zwei Ersatzmöglichkeiten. Die beiden FORTRAN-Befehle

`matmul(A,B), dot_product(a,b)`

führen die Matrixmultiplikation `C=A*B` und die Skalarmultiplikation `c=a*b` gleichlanger Vektoren `a,b` aus. Das Multiplikationszeichen $*$ und das Exponentiationszeichen $**$ wirken in FORTRAN elementweise und haben so dieselbe Wirkung wie die mit Punkten versehenen Operationen `.*, .^` in MATLAB.

Warnungen und Fehlermeldungen können in MATLAB auch selbst programmiert werden in der Form

`warning(Text)` oder `error(Text)`.

Der Text `Text` muß die Form eines `strings` haben, also in der Form `Text='Fehler wegen ...'` definiert werden. Im Warnungsfall `warning(Text)` wird der Text `Warning: Text` angezeigt, weiter passiert nichts, im Fehlerfall `error(Text)` dagegen gibt es einen Abbruch mit einer akustischen Warnung und einer Meldung

```
??? Error using ==> Dateiname (in dem der Fehler passiert ist (ohne .m))
Text
```

Die Strings können in MATLAB noch für einen ungewöhnlichen Zweck benutzt werden in den beiden Befehlen (`Text` soll einen String repräsentieren)

`eval(Text), y=feval(Text,x).`

Der Befehl `eval(Text)` führt genau das aus, was in `Text` ohne die Striche ' ' steht. Ist also z. B. `Text='y=3+x'`, so führt `eval(Text)` die Anweisung `y=3+x` aus. Ist `Text='sin'`, so wird mit `y=feval(Text,1.5)` der Befehl `y=sin(1.5)` ausgeführt. Anwendungen werden wir noch kennenlernen. In MATLAB gibt es eine Reihe nützlicher *Transferfunktionen* , die z. B. Umrechnungen von ganzen Zahlen in Strings vornehmen. Beispiele sind

`s=int2str(n), s=num2str(x),`

mit denen eine ganze Zahl `n` bzw. eine reelle Zahl `x` in einen String `s` umgewandelt wird. Ein Anwendungsbeispiel ist: `s=['Die Temperatur betraegt ',num2str(g),' Grad']` und `g=23.4` ist dabei eine reelle Größe.

6.2 Bedingte Anweisungen

Wir kommen jetzt zu bedingten Anweisungen. Eine bedingte Anweisung hat in FORTRAN die aus den nächsten Beispielen ersichtliche Form.

```
if (l) then

  ...(Anweisung)

end if
```

oder

```
if (l) then

  ...(Anweisung 1)
  else
  ...(Anweisung 2)

end if
```

Hat man viele Fälle zu unterscheiden, so ist die folgende Technik sehr bequem:

```
select case(Fall) !(Fall ist ganzzahlig oder String)
  case(:0) !alle Faelle bis Null
  ...(Anweisung 1)
```

```
  case(1)
  ...(Anweisung 2)
  case(2:5) !alle Faelle von 2 bis 5
  ...(Anweisung 3)
  case(6)
  ...(Anweisung 4)
  case(7:) !alle Faelle ab 7
  ...(Anweisung 5)
end select
```

Eine bedingte Anweisung hat in MATLAB die Form

```
if l

  ...(Anweisung)

end
```

oder

```
if l

  ...(Anweisung 1)
  else
  ...(Anweisung 2)

end
```

oder

```
if l1

  ...(Anweisung 1)
  elseif l2
  ...(Anweisung 2)
  elseif l3
  ...(Anweisung 3)
  else
  ...(Anweisung 4)

end
```

Bedingte Anweisungen von diesem Typ heißen auch `if`-statements. Dabei wird in MATLAB der Ausdruck `l` (entsprechend `l1,l2,l3`) nur dann als falsch angesehen, wenn er den Wert Null hat. Ist l ein Vektor oder eine Matrix, so ist `l` nur dann falsch, wenn *alle* Komponenten Null sind. Ist `l` komplex, so wird nur der Realteil betrachtet. Logische Ausdrücke können durch logische Operatoren miteinander verknüpft werden, dazu Tabelle 6.3.

Tabelle 6.3. Logische Operatoren in FORTRAN und MATLAB

Sprache	oder	und	äquivalent	nicht äquivalent	nicht
FORTRAN	.or.	.and.	.equiv.	.nequiv.	.not.
MATLAB	|	&			~

In MATLAB gibt es auch noch die logischen Funktionen `a=and(b,c), a=or(b,c), a=xor(b,c),`

`a=isempty(b), a=isfinite(b), a=isnan(b), a=isinf(b), a=all(b), a=any(b).`

Die Bedeutung möge man mit `help` ergründen. Auch für die logischen Operationen gibt es eine Prioritätenliste, wenn mehrere derartige Operatoren in einem Ausdruck vorkommen. Die Reihenfolge ist: nicht, und, oder, äquivalent, nicht äquivalent, von links nach rechts. Bei MATLAB gibt es in manchen Fällen Abweichungen, die man in `help precedence` nachlesen kann. Logische Ausdrücke werden oft durch Vergleiche gewonnen, s. Tabelle 6.4.

Tabelle 6.4. Logische Ausdrücke in FORTRAN und MATLAB als Ergebnis von Vergleichen

Sprache	gleich	ungleich	größer	größer gleich	kleiner	kleiner gleich
FORTRAN	==	/=	>	>=	<	<=
MATLAB	==	~=	>	>=	<	<=

Besteht in MATLAB ein logischer Ausdruck aus einer Verkettung von logischen Operatoren, so sind Klammern zu benutzen wie im Beispiel `if (x<y) | ((z~=4) & (x+y<z))`. Man konsultiere ggf. `help precedence`. Ein sehr praktischer, aus logischen Komponenten zusammengesetzter MATLAB-Befehl ist

`J=find(X), K=find(Y>10), L=find(Z==3).`

In allen Fällen sind `X`, `Y`, `Z` Matrizen. In J werden diejenigen Indizes der Komponenten von `X` gespeichert, die nicht Null sind. Entsprechend in den anderen Fällen. Wir zeigen an einem einfachen Beispiel, wie man mit `find` alle lokalen Extrema einer Funktion finden kann. Gleichzeitig machen wir einen kleinen Vorgriff auf das Herstellen von Funktionsgraphen.

Programm 6.5. Mit `find` bestimmte lokale Extrema (MATLAB)

```
%Wir suchen mit Hilfe von find lokale Maxima und Minima einer gegebenen Funktion.
x=linspace(0,1,1001);
y=sin(3*pi*x)+0.5*cos(5*pi*x)+0.3*sin(7*pi*x);
%hat 3 lokale Maxima und 2 lokale Minima in [0,1]
y1=y;  y2=y;
y1(1)=[]; y1=[y1,y(length(y))];
y2(1:2)=[]; y2=[y2,y(length(y)),y(length(y))];
%alle drei Vektoren y,y1,y2 sind gleich lang, sie haben den folgenden Inhalt:
%y =(y1,y2,y3   ,...,   yn)
%y1=(y2,y3,y4 ,...,  yn,yn)
%y2=(y3,y4,y5,...,yn,yn,yn)
%Ist y(j-1)<yj und y(j+1)<yj, so liegt bei yj ein lokales Maximum vor.
%Ist y(j-1)>yj und y(j+1)>yj, so liegt bei yj ein lokales Minimum vor.
%Wir vergleichen y1 mit y und mit y2:
jmax=find(y2-y1<0&y-y1<0); jmax=jmax+1; %da y1 erst bei Index 2 beginnt.
jmin=find(y2-y1>0&y-y1>0); jmin=jmin+1;
plot(x,y,[0 1],[0 0]); hold on
for j=jmax
  plot(x(j),y(j),'ro',x(j),y(j),'r*',x(j),y(j),'rs');  %lokale Maxima rot
end;
for j=jmin
  plot(x(j),y(j),'go',x(j),y(j),'g*',x(j),y(j),'gs');  %lokale Minima gruen
end;
hold off
```

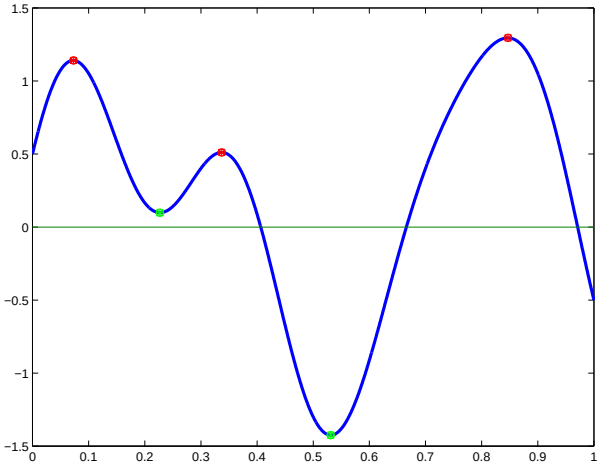

Abb. 6.6. Mit `find` bestimmte lokale Extrema (MATLAB), lokale Extrema markiert

7 Wiederholungen von Programmteilen

Eine wesentliche Fähigkeit aller Programmiersprachen besteht darin, gewisse Anweisungen wiederholen zu können. Dabei gibt es i. w. zwei Fälle: Die Anzahl der Wiederholungen ist von vornherein bekannt oder die Anzahl hängt von einer Bedingung ab und ist nicht vorher bekannt.

7.1 Wiederholungen in FORTRAN mit fester Anzahl von Wiederholungen

Das typische Muster ist

```
do j=anfang,ende,schrittweite

  ...(Anweisungsteil)

end do
```

Der obige Anweisungsteil wird wiederholt für `j=anfang`, `j=anfang+schrittweite`, `j=anfang+2*schrittweite`, `...`, `j=anfang+k*schrittweite`, wobei `k` so gewählt wird, daß `anfang+k*schrittweite` $\leq$ `ende` und `anfang+(k+1)*schrittweite` $>$ `ende`. Die obige Konstruktion nennt man auch einen `do-loop`. Es ist möglich, daß der Anweisungsteil selbst weitere `do-loops` enthält. Kommt `schrittweite` nicht vor, so wird mit der Schrittweite Eins gerechnet. Ist `anfang > ende` und die Schrittweite positiv, so ist der `do-loop` eine *leere Anweisung* (nichts wird getan). Dasselbe tritt ein bei negativer Schrittweite und `anfang < ende`. Beispiele für `do-loops` stehen bereits in den Programmen 1.2, ab Zeile 16.

7.2 Wiederholungen in FORTRAN mit variabler Anzahl von Wiederholungen

Das typische Muster ist hier

```
do while(logische_Bedingung)

  ...(Anweisungsteil)

end do
```

Hier wird der Anweisungsteil solange ausgeführt, wie die `logische_Bedingung` richtig ist. Ist die Bedingung gleich am Anfang falsch wird der Anweisungsteil gar nicht ausgeführt, der Programmteil hat die Form einer leeren Anweisung. Im Anweisungsteil muß daher sinnvollerweise diese Bedingung geändert werden.

7.3 Wiederholungen in MATLAB mit fester Anzahl von Wiederholungen

Wir zeigen verschiedene typische Muster. Die Variable j läuft von $1, 2, \ldots,$ bis 10.

```
for j=1:10

   ...(Anweisungsteil)

end
```

Die Variable j läuft in Schritten von 3 von 1 bis (höchstens) 10, also $j = 1, j = 4, j = 7, j = 10$.

```
for j=1:3:10

   ...(Anweisungsteil)

end
```

Die Variable `j` übernimmt nacheinander die Werte aus dem Vektor `x`.

```
x=0.6:0.1:1.5; %also x=[0.6, 0.7, ..., 1.5]
for j=x

   ...(Anweisungsteil)

end
```

Die obige Wiederholungsform nennt man auch `for`-Schleife.

7.4 Wiederholungen in MATLAB mit variabler Anzahl von Wiederholungen

Das typische Muster ist hier

```
while l

  ...(Anweisungsteil)

end
```

Der Anweisungsteil wird solange ausgeführt bis `l` falsch ist. Dabei ist `l` ein Ausdruck mit dem Wert wahr oder falsch. Was in MATLAB wahr oder falsch ist, wird beim `if`-statement, S. 19 erklärt. Wir geben ein kleines Beispiel bei der eine Summe solange gebildet wird bis der letzte Summand eine gegebene Schranke unterschreitet.

Programm 7.1. Taylorreihe für Sinus, Benutzung von `while` (MATLAB)

```
1  %Die Taylorreihe fuer Sinus ist sin(x)=x-x^3/3!+x^5/5!-x^7/7!+...
2  %Fuer ein gegebenes x (reell oder komplex) addieren wir die Summe solange,
3  %bis der letzte Summand (dem Betrage nach) <0.001 ist. Funktioniert sehr gut.
4  epsilon=0.001;   x=1+i;   s=x;   Term=x;   n=0;
5  while abs(Term) >= epsilon
6    n=n+1;
7    Term=-Term*x^2/(2*n*(2*n+1));
8    s=s+Term;
9  end;
10 [n,s,sin(x)]
11 %n                    s                 sin(x)
12 %4             1.2985 + 0.6350i   1.2985 + 0.6350i
```

8 Unterprogramme in FORTRAN und MATLAB

Sollen in einem Programm größere Programmteile wiederholt verwendet werden, so ist es zweckmäßig, diese Teile separat als *Unterprogramme* zu formulieren. Die Techniken in den beiden Programmiersprachen dazu sind verschieden. Trotzdem gibt es ein einheitliches Muster. Unterprogramme sind Programme, die vom Hauptprogramm aufgerufen werden und folgende Funktionen ausführen können. Sie können Daten, die vom Hauptprogramm geliefert werden (in Form von Variablennamen oder Konstanten) lesen, sie können Daten (in Form von Variablennamen) erzeugen, und sie können Daten von Variablen lesen und nach dem Lesen andere Daten auf diese Variablen speichern. Daneben können sie weitere Funktionen ausführen, wie das Erzeugen von Bildern und Tönen. Man spricht von einem *Wertaufruf* (engl. *call by value*) einer zu lesenden Variablen in einem Unterprogramm, wenn diese am Beginn des Unterprogrammaufrufs ausgewertet, intern auf eine Hilfsvariable abgelegt wird und im Unterprogramm nur mit dem festen Wert dieser Hilfsvariable gerechnet wird. So ist sichergestellt, daß nach Beendigung des Unterprogramms die zu lesende Variable unverändert zurückkommt. Zu beachten ist, daß diese Aufrufform zusätzlichen Speicherplatz kostet.

8.1 Unterprogramme in FORTRAN

In FORTRAN werden die Unterprogramme entweder als `subroutine` oder als `function` formuliert, und direkt an das bestehende Hauptprogramm angehängt. Die als `subroutine name(...)` formulierten Unterprogramme werden mit `call name(...)` aufgerufen, die als `function name(...) result(...)` definierten Funktionsunterprogramme, werden mit ihrem Namen aufgerufen, s. Musterprogramme 3.2, S. 8 und 8.1. Abgesehen von den Unterprogrammrahmen sind innerhalb der Unterprogramme alle FORTRAN-Regeln gültig.

In vernünftigen Programmiersprachen, die Eingabe- und Ausgabeparameter bei der Definition von Unterprogrammen unterscheiden, werden die Eingabeparameter durch Aufruf des Unterprogramms nicht verändert (Wertaufruf), selbst, wenn im Unterprogramm Manipulationen mit den Eingabeparametern vorgenommen werden. Das wird dadurch bewirkt, daß die Eingabeparameter vor Übergabe an das Unterprogramm gesondert gespeichert und nach Ende des Unterprogrammablaufs zurückgespeichert werden. In diesem Sinne ist FORTRAN nicht vernünftig. Zu den vernünftigen Programmiersprachen gehören Pascal, MATLAB, ALGOL60. [1] In den neueren FORTRAN-Varianten ist es jedoch möglich durch zusätzliches Anbringen eines Attributs `intent` im Deklarationsteil mit den drei möglichen Parametern `in, out, inout` zu bestimmen wie sich die Parameter einer subroutine verhalten sollen. Dazu ein FORTRAN-Beispiel:

Programm 8.1. FORTRAN-Programm als Beispiel zur Benutzung von Unterprogrammen

```
1  program subroutine_und_function
2    implicit none
3    integer :: l, q, quadratsumme
4    double precision :: x, ll
5    !quadratsumme ist als Funktion ganzzahlig definiert.
6    !Trotzdem wird der Name "quadratsumme" ohne Deklaration
7    !als "real" interpretiert, und das Ergebnis ist immer Null.
8    !Es ist zu beachten, dass ganzzahlige Variablen nur
9    !bis 2^31 (etwa 2.15e9) gehen. Die Quadratsumme wird also falsch
10   !fuer groessere l (ab l=1861)
11   call biswieweit(l)  !<=== Aufruf der Subroutine
12   q=quadratsumme(l)   !<=== Aufruf des Funktions-Unterprogramms
13   write(*,fmt=*) 'Quadratsumme ist: ',q
14   ll=l; if (ll > 0.0d0) then
15     x=ll*(ll+1)*(2*ll+1)/6
16     else; x=0.0d0
17   end if
18   write(*,fmt=*) 'Quadratsumme doppelt genau zum Vergleich: ',x
19 end program subroutine_und_function
20
```

[1] J. W. BACKUS et al.: Report on the algorithmic language ALGOL60, Numer. Math.**2** (1960), 106–136.

```
21 subroutine biswieweit(n)
22   implicit none
23   integer, intent(out) :: n
24   write(*,fmt="('Bitte ganzzahliges n angeben: ')",advance="no")
25   read(*,fmt=*) n
26 end subroutine biswieweit
27
28 function quadratsumme(x) result(y)
29   implicit none
30   integer :: j,x,y,s
31   !Die Summe der Quadrate y=1+2^2+...+x^2 ist
32   !bis zu einem vorgegebenen x auszurechnen
33   !Das x wird ueber die Subroutine "biswieweit" eingeschleust.
34   s=0
35   do j=1,x
36     s=s+j*j
37   end do
38   y=s;
39 end function quadratsumme
```

Es gibt eine große Anzahl vorgefertigter, fest eingebauter Unterprogramme (*intrinsic procedures*). Dazu gehören die schon seit FORTRAN IV vorhandenen Funktionen. Wir geben in Tabelle 8.2 im ersten, größeren Teil bis zur Trennlinie eine Liste dieser Funktionen aus dem Buch von MCCRACKEN, [1965, S. 136–137], s. Fußnote Nr. 4, S. 35. Aus dem Buch von METCALF & REID [2002, S. 169ff.] (s. Fußnote Nr. 5, S. 35) kann man sich über neuere eingebaute Funktionen informieren. Aber Vollständigkeit liegt auch dort nicht vor.

Tabelle 8.2. FORTRAN IV und einige neuere Funktionen

Math. Name	Math. Bez.	FORTRAN	Argument(e)	Wert
Exponentialfunktion	e^a	`exp`	Real	Real
		`dexp`	Double	Double
		`cexp`	Complex	Complex
Natürlicher Logarithmus	$\log_e(a)$	`alog`	Real	Real
		`dlog`	Double	Double
		`clog`	Complex	Complex
Zehnerlogarithmus	$\log_{10}(a)$	`alog10`	Real	Real
		`dlog10`	Double	Double
Sinus	$\sin(a)$	`sin`	Real	Real
		`dsin`	Double	Double
		`csin`	Complex	Complex
Kosinus	$\cos(a)$	`cos`	Real	Real
		`dcos`	Double	Double
		`ccos`	Complex	Complex
Hyperbolischer Tangens	$\tanh(a)$	`tanh`	Real	Real
Quadratwurzel	$(a)^{1/2}$	`sqrt`	Real	Real
		`dsqrt`	Double	Double
		`csqrt`	Domplex	Complex
Arcus Tangens	$\arctan(a)$	`atan`	Real	Real
		`datan`	Double	Double
	$\arctan(a_1/a_2)$	`atan2`	Real	Real
		`datan2`	Double	Double
Modulus	$a_1 \pmod{a_2}$	`dmod`	Double	Double
		`amod`	Real	Real
		`mod`	Integer	Integer
Betrag	$\lvert a \rvert$	`cabs`	Complex	Real
		`abs`	Real	Real
		`iabs`	Integer	Integer
		`dabs`	Double	Double

Math. Name	Math. Bez.	FORTRAN	Argument(e)	Wert
Abschneiden	Vorzeichen von a	`aint`	Real	Real
	mal größte ganze	`int`	Real	Integer
	Zahl $\leq \lvert a \rvert$	`idint`	Double	Integer
Maximum	Max $(a_1, a_2, \ldots)$	`amax0`	Integer	Real
		`amax1`	Real	Real
		`max0`	Integer	Integer
		`max1`	Real	Integer
		`dmax1`	Double	Double
Minimum	Min $(a_1, a_2, \ldots)$	`amin0`	Integer	Real
		`amin1`	Real	Real
		`min0`	Integer	Integer
		`min1`	Real	Integer
		`dmin1`	Double	Double
integer $\rightarrow$ real	Transfer			
	integer $\rightarrow$ real	`float`	Integer	Real
real $\rightarrow$ integer	Transfer			
	real $\rightarrow$ integer	`ifix`	Real	Integer
Signum (a_1, a_2)	Vorzeichen von	`sign`	Real	Real
	a_2 mal $\lvert a_1 \rvert$	`isign`	Integer	Integer
		`dsign`	Double	Double
Differenz ≥ 0	a_1 - Min (a_1, a_2)	`dim`	Real	Real
		`idim`	Integer	Integer
double precision $\rightarrow$ real		`sngl`	Double	Real
real $\rightarrow$ double precision		`dble`	Real	Double
Realteil	Re	`real`	Complex	Real
Imaginärteil	Im	`aimag`	Complex	Real
konjugiert komplex	$\overline{a}$	`conjg`	Complex	Complex
(real,real) $\rightarrow$ complex	$a_1 + a_2\sqrt{-1}$	`cmplx`	Real	Complex
ab hier neuere Funktionen:				
(Double,Double) $\rightarrow$ complex, dbl prec	$a_1 + a_2$i	`dcmplx`	Double	Complex, Double
Decke	$\lceil x \rceil$	`ceiling`	Real	Real
Fußboden	$\lfloor x \rfloor$	`floor`	Real	Real
Produkt aller Feldelemente	$\prod$	`product`	Feld	Feldtyp
Summe aller Feldelemente	$\sum$	`sum`	Feld	Feldtyp
Maximum eines Feldes	max	`maxval`	Feld	Feldtyp
Minimum eines Feldes	min	`minval`	Feld	Feldtyp
ascii-Nummer eines Zeichens		`ichar`	character(len=1)	integer
Zeichen mit ascii-Nummer		`char`	integer	character(len=1)

Bei Simulationen werden oft Zufallszahlen gebraucht, entsprechende (Pseudo-)Zufallszahlengeneratoren sind in den meisten Programmiersprachen vorhanden. In FORTRAN ergibt der Aufruf

```
call random_number(x)
```

eine in $]0,1[$ liegende Zufallszahl `x` vom Typ `real`, die bei vielen Aufrufen als gleich verteilt erscheinen. Eine Zeitabfrage ist mit

`call system_clock(...)` `call data_and_time(...)` `call cpu_time(...)` (ab FORTRAN95)

möglich. Ein Beispiel steht in Programm 8.3, das auch die hohe Rechengeschwindigkeit von FORTRAN zeigt.

Programm 8.3. Drei verschiedene Zeitmessungen (FORTRAN)

```
1  program Zeiten_Testen
2    integer, parameter :: m=1000, n=800
3    character(len=20) :: Datum,Zeit1,Zeit2
4    integer :: Zeler1,Zeler2,Zelerrate
5    real :: x,x1,x2,t
6    real, dimension(m,n) :: A
7    integer :: j,k
8    call cpu_time(x1)
9    call system_clock(Zeler1,Zelerrate)
10   call date_and_time(Datum,Zeit1)
11   write(*,fmt="(a12,i12)") Zeit1,Zeler1
12   do j=1,m
13     do k=1,n
14       call random_number(x)
15       A(j,k)=x
16     end do
17   end do
18   call date_and_time(Datum,Zeit2)
19   call system_clock(Zeler2)
20   call cpu_time(x2)
21   write(*,fmt="(a12,i12)") Zeit2,Zeler2
22   t=(float(Zeler2)-float(Zeler1))/float(Zelerrate)
23   write(*,fmt=*) m,n,A(m,n),t,x2-x1
24   !braucht auf unix1 nur 0.72 Sek.
25   !alle drei Zeiten (mit cpu_time, system_clock, date_and_time)
26   !stimmen i. w. ueberein. cpu_time nur FORTRAN95.
27 end program Zeiten_Testen
```

In gewissen Situationen ist es zweckmäßig, innerhalb der Erklärung eines Funktionsunterprogramms, die gerade zu erklärende Funktion im Unterprogramm selbst aufzurufen. Man spricht dann von einer *rekursiven* Funktionsdefinition oder kurz von einer rekursiven Funktion. Ein Standardbeispiel ist die Berechnung der Fakultät

$$n! := 1 \cdot 2 \cdots n = (1 \cdot 2 \cdots (n-1)) \cdot n = n(n-1)!, \quad 1! := 1. \tag{8.1}$$

In FORTRAN müssen jedoch besondere Weichen gestellt werden, um rekursive Funktionen zu definieren. Wir gehen daher darauf nicht ein, s. METCALF & REID [2002, S. 91–93], Fußnote Nummer 5, S. 35. Im nächsten Abschnitt benutzen wir die Formel (8.1) und konstruieren daraus ein rekursiv aufgebautes MATLAB-Programm 8.5.

8.2 Unterprogramme in MATLAB

In MATLAB müssen alle Unterprogramme in der Form

`[outputparameterliste]=function Name(inputparameterliste)`

eingeleitet werden und in einer Datei mit dem Namen `Name.m` stehen. Aufgerufen werden sie dann in der Form

`[outputparameterliste]=Name(inputparameterliste)`

Der Aufruf bewirkt das Suchen nach einer Datei `Name.m` und Ausführen der dort formulierten Funktion, selbst dann wenn der Funktionsname vom Dateinamen verschieden ist. Zu beachten ist, daß die `inputparameterliste` nach Aufruf unverändert ist (Wertaufruf). MATLAB ist also vernünftig im eben angegebenen Sinne. Es ist möglich, weitere Funktionen in derselben Datei zu formulieren, wenn man sie in der zuerst benutzten Funktion benutzen will. Außen sind diese Funktionen aber nicht zugänglich. Im angegebenen Beispiel, Programm 8.4, ist das in den Zeilen 15 und 16 formulierte Programm `quadrat` im Hauptprogramm nicht benutzbar. Dazu ein einfaches MATLAB-Beispiel:

Programm 8.4. MATLAB-Programm zur Benutzung von Unterprogrammen

```
function y=quadratsumme(x);
%y=quadratsumme(x);
%Berechnung der Quadratsumme y = 1 + 2^2 + ... + x^2
if x ~= round(x) | x ~= real(x)
  error('Argument in quadratsumme nicht ganzzahlig!')
end;
X=[1:x];
if isempty(X)
  y=0;
  else
  X=quadrat(X);
  y=sum(X);
end; %keine Schleife wurde benutzt.

function v=quadrat(u)
v=u.*u;
```

In MATLAB ist es sehr leicht, Funktionen rekursiv zu definieren. Wir behandeln das Beispiel aus Formel (8.1), S. 26. Das Ergebnis steht in Programm 8.5. Ein wesentlich interessanteres Beispiel ist unter dem Namen *Turm von Hanoi* bekannt, s. S. 34.

Programm 8.5. MATLAB-Beispiel zur rekursiven Definition

```
function nfak=fakultaet(n);
%Rekursive Definition der Fakultaet.
%function nfak=fakultaet(n);
  if n<=1
    nfak=1;
    else
    nfak=n*fakultaet(n-1);
  end
return %return nicht notwendig
```

Verwendet man in einem MATLAB-Programm Unterprogramme, so kann man nicht auf die lokalen Variablen dieser Unterprogramme zugreifen. Es gibt eine Möglichkeit, über die explizite Deklaration

```
global x
```

im Hauptprogramm und in allen Unterprogrammen, in denen Kommunikation mit `x` gewünscht wird, auf `x` zuzugreifen.

Beim Aufruf eines Unterprogramms kann man von der vordefinierten Zahl der Parameter abweichen. Das kann bei der Formulierung des Unterprogramms über die beiden Hilfsfunktionen

```
nargin    nargout
```

gesteuert werden. Beides sind parameterlose Unterprogramme, die bei Aufruf die tatsächliche Anzahl der Parameter, von vorne gezählt, die in der inputparamerterliste, bzw. in der outputparameterliste stehen, angeben. Ist z. B. `x` ein eindimensionaler Vektor, so gibt es die beiden Aufrufe

```
xmin=min(x)    [xmin,jmin]=min(x)
```

Im ersten Fall wird der minimale Wert unter den Komponenten von `x` als `xmin` ausgegeben, im zweiten Fall zusätzlich `jmin`, der erste Index für den das Minimum `xmin` angenommen wird. Ist also `x=[-1 -2 -3 -3 -2 -1]`, so ist `xmin=-3` und `jmin=3`. Läßt man also Parameter weg, so kann man das nur von hinten nach vorne tun.

In MATLAB gibt es ein Riesenarsenal von vorgefertigten Unterprogrammen. Ein Zufallszahlengenerator mit in $]0,1[$ gleich-verteilten Zufallszahlen hat die Form

```
y=rand(m,n).
```

Der Aufruf bewirkt, daß `y` eine (m,n)-Matrix ist, die an jeder Position mit einer entsprechenden Zufallszahl besetzt ist. Eine Stoppuhr wird durch die beiden Befehle

```
tic    toc
```

ausgelöst. Mit `tic` wird die Uhr gestartet, mit `toc` wird sie gestoppt, und eine Anzeige der Form

```
elapsed_time =
    2.3304
```

erscheint auf dem Schirm. Die Zahl ist eine Angabe in Sekunden.

9 Ausgabeformate in FORTRAN und MATLAB

Wir sprechen hier von den Fomaten der Ausgabe von Zahlen, nicht von den Formaten der internen Benutzung. Da man jedenfalls bei der Ausgabe größerer Zahlenmengen mehr an graphischen Repräsentationen interessiert ist, sind die Formatierungsmöglichkeiten nicht mehr so wichtig. Daher gibt es in MATLAB i. w. einige wenige Standardausgabeformate, in FORTRAN aus historischen Gründen viele.

9.1 Ausgabeformate in FORTRAN

Die FORTRAN-Zahl-Formate werden untergebracht in den Formen

```
write(A,B,C) x1, x2, ..., xn  oder  read(A,B,C) y1, y2, ..., yn
```

wobei der Buchstabe `A` das Medium (Bildschirm, Datei, etc.), der Buchstabe `B` Zahl-Formate aller in der Liste angegebenen Variablen, und der Buchstabe `C` ggf. noch Zusatzinformationen enthält. Der Buchstabe `B` muß also durch eine Liste von Formatangaben ersetzt werden, die für jede der auszugebenden Größen `x1, x2, ..., xn` ein Format bereitstellt. Enthält `B` weniger als `n` Formatangaben, so werden die Formatangaben in `B` zyklisch wiederholt. Allerdigs wird nach Beendigung der Formatliste eine neue Zeile begonnen. Gibt man also alle Ergebnisse in dem obigen `write`-Befehl mit einem einzigen Format aus (so überhaupt möglich), so werden alle Ergebnisse untereinander geschrieben. Ist `A` ein Stern `*`, so wird damit das Standardformat bezeichnet (Bildschirm, Tastatur). Ist `A` eine positive, ganze Zahl, so wird in einer weiteren Anweisung der Form

```
open(A,file='Dateiname',status='unknown')
```

eine Datei bezeichnet, in die geschrieben, oder aus der gelesen werden kann. Es gibt verschiedene Angaben zum `status`.

1. 'unknown': Eine Datei wird angelegt. Gibt es sie schon, wird sie überschrieben, aber compilerabhängig.
2. 'scratch': Diese Dateien werden im Fehlerfall und am Programmende, wenn also kein `close` kommt, gelöscht.
3. 'replace': Datei wird gelöscht und überschrieben, wenn vorhanden.
4. 'old': Eine vorhandene Datei wird geöffnet. Ist sie nicht vorhanden, gibt es eine Fehlermeldung.
5. 'new': Eine Datei wird neu angelegt und ihr status auf 'old' gesetzt. Gibt es sie schon, gibt es eine Fehlermeldung.

Entsprechend gibt es auch `close`-Anweisungen der beiden Formen

```
close(A,status='keep')    close(A,status='delete')
```

wobei die Wirkung aus der Wortbedeutung klar ist. Fehlt die `close`-Anweisung wird die entsprechende Datei am Programmende mit `'keep'` abgeschlossen, es sei denn, sie hat den Status `scratch`, dann wird sie gelöscht.

Das Format `B` besteht aus einer durch Kommata getrennten Liste, und jedes Einzelformat darin hat i. w. eine der folgenden Formen:

```
wKm.n     wKm
```

Dabei ist `K` eine *Formatkennung*, engl. *data edit descriptor* meistens aus einem kleinen Buchstaben bestehend, wobei die in der folgenden Liste vorkommenden Formatkennungen die häufigsten sind:

1. `a` Kennung für Textformate, mit `character` deklariert: Max und Moritz,
2. `i` Kennung für ganze Zahlen, mit integer deklariert: -123,

3. `f` Kennung für Festkommaformate: 23.45,
4. `d` Kennung für Gleitkommaformate: 0.123E-02 (vorne Null)
5. `es` Kennung für Gleitkommaformate: 1.23E-03 (vorne einstellig ≥ 1),
6. `l` Kennung für logische Konstanten: `T` oder `F`.

Formate für komplexe Zahlen werden aus Paaren reeller Formate zusammengesetzt. Der Buchstabe `w` steht für Wiederholung und bedeutet, daß das Format `w`-mal verwendet werden soll. Ist `w=1`, so kann diese Angabe entfallen. Die direkt hinter der Formatkennung angegebene Zahl `m` ist die Gesamtlänge der auszugebenden Größe. Die Angabe `m.n` liefert eine Ausgabe mit `n` Nachkommastellen. Zwischen die Formatangaben kann durch Kommata getrennt Text (z. B. `'a = '`) und `wx` (z. B. `4x`) und `w/` geschrieben werden. Der Buchstabe `w` steht wieder für die Anzahl der Wiederholungen, und `x` steht für einen Zwischenraum, und `/` steht für einen neuen Zeilenbeginn. Mehrere Formate können durch eine Klammer mit einer davorstehenden Wiederholungsangabe zusammengefasst werden. Beispiel (enthält 7 Formatangaben plus Zwischenraum, Text und neuen Zeilenbeginn):

```
2(2a5,3x,' y = ',f17.2),/, l7
```

Ist die auszugebende Größe kürzer als das vorgegebene Format, so wird sie rechtsbündig eingesetzt. Ist sie länger, so werden die vorgesehenen Formatstellen durch Sterne `*` aufgeüllt oder (bei Textformaten) die auszugebende Größe wird auf die ersten `m` Stellen reduziert. Passen die Formate nicht zum auszugebenden Typ, so erfolgt eine Fehlermeldung. Wie man das Format `B` besetzen kann, ergibt sich auch aus der nachfolgenden Beispielliste. Insbesondere gibt es noch das Standardformat, das immer funktioniert.

1. Standardformat: `B=fmt=*` Mit diesem Format kann man alles ein- und ausgeben, sehr empfehlenswert. Die Ergebnisse eines `write`-Befehls werden nebeneinander geschrieben.
2. Ganzzahlige Formate:
   (a) `B=fmt="(' ',i5)"`: Luft, eine 5-stellige ganze Zahl,
   (b) `B=fmt="(10(' ',i6))"`: 10-mal: Luft und eine 6-stellige Zahl,
   (c) `B=fmt="(12i4)"`: 12 vierstellige ganze Zahlen.
3. Festkomma-Formate:
   (a) `B=fmt="(' ',f5.1)"`: Luft, eine 5-stellige reelle Zahl, davon eine Stelle nach dem Komma, alle Stellen, Vorzeichen, Komma etc werden mitgezählt.
   (b) `B=fmt="(10(2x,f10.2))"`: 10-mal: 2xLuft und eine 10-stellige Zahl davon 2 nach dem Komma, insgesamt werden also 120 Stellen benutzt,
   (c) `B=fmt="(2f10.5,4f12.6)"`: Zwei 10-stellige und vier 12-stellige reelle Zahlen.
4. Gleitkomma-Formate: Diese Zahlen enthalten in der ersten Stelle (vor dem Komma) eine Null (bei Formatkennung `d`) oder eine 1 (bei Formatkennung `es`) und am Schluß ein Exponentenzeichen `E` oder `D`.
   (a) `B=fmt="(' ',d5.1)"`: Luft, eine 5-stellige reelle Zahl, davon eine Stelle nach dem Komma, alle Stellen, Vorzeichen, Komma etc werden mitgezählt,
   (b) `B=fmt="(10(2x,d10.2))"`: 10-mal: 2xLuft und eine 10-stellige Zahl, davon 2 nach dem Komma, insgesamt werden also 120 Stellen benutzt,
   (c) `B=fmt="(2es10.5,4es12.6)"`: Zwei 10-stellige und vier 12-stellige Gleitkommazahlen mit ≥ 1 beginnend.
5. Formate für Texte (Strings)
   (a) `B=fmt="(a10,2x,a12)"`: 10-stelliger Text, 2 Leerstellen, 12-stelliger Text.
6. Formate für logische Variable
   (a) `B=fmt="(l1,2x,l1)"`: 2 logische Werte (`T` oder `F`) dazwischen 2 Leerstellen.

Der Zusatz `C` kann z. B `advance="no"` enthalten. Das bedeutet, daß der nächste `write`-Befehl nach einem `read`-Befehl nicht eine Zeile weiterrückt.

Beim Lesen einer Datei mit `read` gilt, daß die in der Datei befindlichen Daten sequentiell, also von links nach rechts und von oben nach unten gelesen werden ohne Rücksicht auf etwa vorkommende Zeilenumbrüche oder Leerzeilen. Die in der Datei vorkommenden Daten werden gegeneinander durch Leerzeichen oder Kommata abgetrennt. Kommentare dürfen nicht vorkommen. Ist `a` ein Feld, deklariert mit `dimension(d1,d2,...,dn)`, so bewirkt

```
read(1,fmt=*) a
```

eine Zuordnung zu den einzelnen Elementen von `a` in der Form, daß zuerst der erste Index `d1` läuft, dann der zweite `d2` etc. Ein Beispiel steht im Programm 9.1.

Programm 9.1. Zuordnung beim Lesen von indizierten Variablen (FORTRAN)

```
program matrix_lesen_testen
  integer :: j,k,l
  integer, dimension(2,3,4) :: a
  open(1,file='matrix_lesen.dat',status='old')
  read(1,fmt=*) a
  do l=1,4
    do k=1,3
      do j=1,2
        write(*,fmt=*) 'a(',j,k,l,') = ',a(j,k,l)
      end do
    end do
  end do
  !Ergebnis der Zuordnung:
  ! a(1,  1,  1) = 1
  ! a(2,  1,  1) = 2
  ! a(1,  2,  1) = 3
  ! a(2,  2,  1) = 4
  ! a(1,  3,  1) = 5
  ! a(2,  3,  1) = 6
  ! a(1,  1,  2) = 7
  ! a(2,  1,  2) = 8
  ! a(1,  2,  2) = 9
  ! a(2,  2,  2) = 10
  ! a(1,  3,  2) = 11
  ! a(2,  3,  2) = 12
  ! a(1,  1,  3) = 13
  ! a(2,  1,  3) = 14
  ! a(1,  2,  3) = 15
  ! a(2,  2,  3) = 16
  ! a(1,  3,  3) = 17
  ! a(2,  3,  3) = 18
  ! a(1,  1,  4) = 19
  ! a(2,  1,  4) = 20
  ! a(1,  2,  4) = 21
  ! a(2,  2,  4) = 22
  ! a(1,  3,  4) = 23
  ! a(2,  3,  4) = 24
  write(*,fmt=*) ' '      !Leerzeile
  write(*,fmt=*) a        !alle Zahlen in eine Zeile, kein Zeilenumbruch
  close(1,status='keep')
end program matrix_lesen_testen

!Die zu lesenden Daten einer Datei werden sequentiell,
!also ohne Verwendung von Zeilenumbruechen oder Leerzeilen
!von links nach rechts und von oben nach unten gelesen.
!Die Zuordnung zu einem mehrdimensionalen Feld erfolgt dann so,
!dass zuerst der erste, dann der zweite dann der dritte Index,
!etc. laeuft. Kommentare duerfen nicht vorkommen. Die Daten koennen
!durch Leerzeichen oder durch Komma voneinander getrennt werden.

!Die zu lesende Datei hatte (ohne Ausrufezeichen)
!den folgenden Inhalt:
!1 2 3 4 5 6 7 8 9
!10 11 12 13 14 15 16 17 18 19 20
!
!21 22 23 24
```

9.2 Ausgabeformate in MATLAB

Das Zauberwort heißt hier `format`, mit `help format` kann man sich die ganze Palette von Möglichkeiten ansehen. Die wesentlichen Formate sind

1. `format compact`: nicht so viel Luft zwischen den Zeilen,
2. `format short e`: 3.1416e+00 (alle Beispiele mit `pi`),
3. `format long`: 3.14159265358979,
4. `format rat`: Brüche: 355/113,
5. `format bank`: für Geldleute, immer 2 Stellen nach dem Komma: 3.14,
6. `format short`: das ist das Standardformat: 3.1416.

Will man in MATLAB unbedingt FORTRAN-ähnliche Formate, so ist dafür `fprintf` zuständig, ggf. auch `sprintf`. Man informiere sich mit `help`. Wir können darauf aber nicht eingehen.

Will man alles, was man im Laufe einer Sitzung am Rechner gemacht hat, nochmal sehen, oder aufheben, so schreibe man

```
diary on
```

Die gesamte Sitzung wird von da an in einer Datei (mit Namen diary) protokolliert. Will man das wieder abschalten, so ist

```
diary off
```

der richtige Befehl.

10 Graphik in FORTRAN und MATLAB

Hier geht es darum, in einfacher Form, Daten zu visualisieren. Die einfachsten Aufgaben bestehen im Plotten (Zeichnen) von Funktionsgraphen oder Meßdaten, z. B. meteorologischen Daten oder statistischen Daten (Stichwort: Balkendiagramm, Tortendiagramm). Dazu stellt MATLAB eine riesige Palette von Möglichkeiten bereit, auf die wir nur sehr obeflächlich eingehen können.

10.1 Graphik in FORTRAN

Graphik ist in FORTRAN nicht existent.

10.2 Graphik in MATLAB

Die Aufgabe, nämlich einen Funktionsgraphen zu zeichnen, erledigen wir in der einfachen Form

```
plot(x,y);
```

Dabei sind `x,y` zwei gleich lange Vektoren `x=(x_1,x_2,...,x_n)`, `y=(y_1,y_2,...,y_n)` und `plot` stellt eine jeweils geradlinige Verbindung her zwischen den Paaren `(x_j,y_j)` und `(x_j+1,y_j+1)`, $j = 1, 2, \ldots, n-1$. Wir beginnen zur Einführung mit der einfachsten Aufgabe, nämlich dem geradlinigen Verbinden von nur zwei Punkten, die in mathematischer Form gegeben seien als $(x_1, y_1), (x_2, y_2)$. Setzen wir jetzt `x=[x_1,x_2], y=[y_1,y_2]` in den `plot`-Befehl ein, und zeichnen wir nach demselben Muster auch noch eine x- und eine y-Achse ein, so erhalten wir das Bild aus Abbildung 10.2, wobei wir noch eine Beschriftung (kennen Sie den beschrifteten Sessel von Ernst Jandl [1925-2000]?) [1] mit

```
xlabel('Unser erstes Bild')
```

angebracht haben. Das kleine Programm mit dem wir das Bild gemacht haben ist:

Programm 10.1. Das erste Bild (MATLAB)

```
1 x=[-1,1]; y=[-1,2]; %die zu verbindenen Punkte.
2 xachsex=[-1 1]; xachsey=[0 0];
3 yachsex=[0 0];  yachsey=[-1 2];
4 plot(x,y, xachsex,xachsey,yachsex,yachsey)
5 xlabel('Unser erstes Bild')
6 %print -dpsc bild1.ps
```

[1] der beschriftete sessel

für harry & angelika

ich habe einen sessel
stehn JANDL groß hinten drauf
wenn ich mal nicht wissen
sein ich's oder sein ich's nicht
ich mich nur hinsetzen müssen
und warten bis von hinten wer
kommen und mir's flüstern [Reclam-Verlag, Leipzig, 1991]

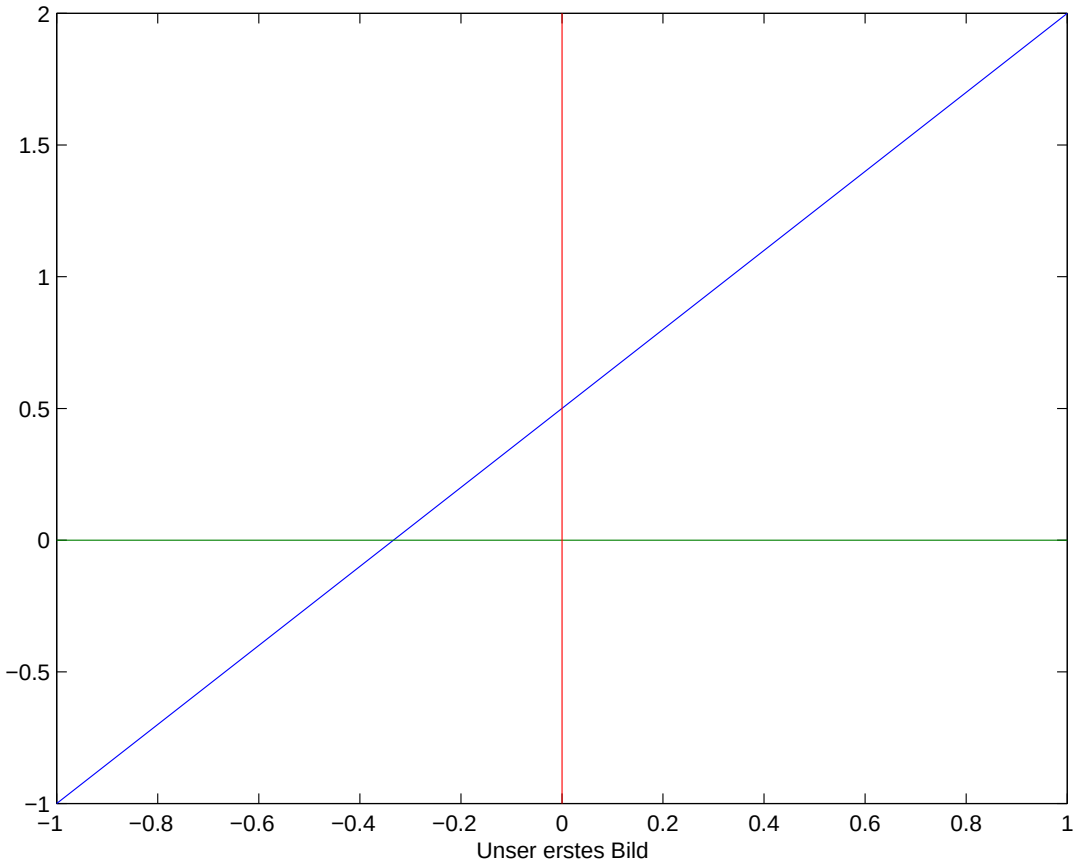

Abb. 10.2. Geradlinige Verbindung von zwei Punkten (MATLAB)

Wir sehen, daß wir einen Bilderrahmen mitgeliefert bekommen. Will man ihn nicht haben, oder die Beschriftung ändern, oder nur einen Ausschnitt sehen, so passiert das mit dem Befehl `axis`, s. `help axis`. Das Bild kann man auch über die Menüleiste manipulieren. Der letzte (ausgeblendete) Befehl `print -dpsc bild1.ps` sagt, daß das gerade zu sehende Bild in postscript, farbig (das `c`) unter dem Namen `bild1.ps` abgespeichert werden soll. Unter `help print` wird eine große Palette von Abspeichermöglichkeiten gezeigt. In diesen Text (mit Textverarbeitungssystem LaTeX) wird das Bild dann mit der folgenden Befehlessequenz eingefügt.

```
\vbox{
$$\hbox{\psfig{file=\pfadmatlab/bild1.ps,width=0.9\hsize}}$$

\medskip
\figur{Geradlinige Verbindung von zwei Punkten (MATLAB)}{GeradM}
}
```

Die Einsperrung in eine `vbox` bewirkt, daß Bild und Unterschrift nicht getrennt werden, die Einsperrung in $$`hbox` $$ bewirkt Zentrierung des Bildes.

Nun ist es naheliegend, nicht nur zwei Punkte zu verbinden, sondern mehrere. Der Effekt kann aus den nächsten Bildern entnommen werden in denen wir einige Punkte des Graphen der Sinus-Funktion geradlinig verbinden. Schon bei 64 Punkten haben wir nicht mehr den Eindruck einer stückweise, geradlinigen Verbindung.

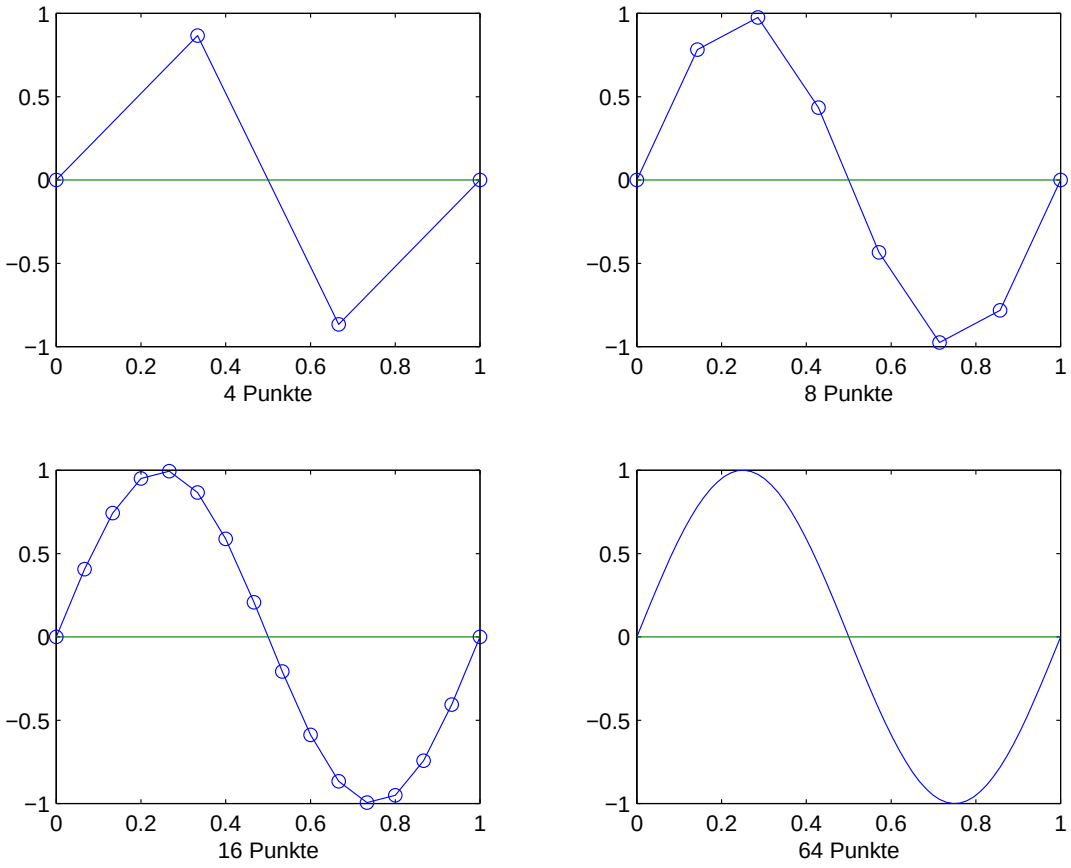

Abb. 10.3. Geradlinige Verbindung von mehreren Punkten (MATLAB)

Will man mit verschiedenen `plot`-Befehlen in dieselbe Figur zeichen, so verwende man *nach* dem ersten `plot`-Befehl, der auf dasselbe Fenster zugreifen soll das Kommando `hold on` und nach dem letzten `plot`-Befehl den Befehl `hold off`.

Will man in ein einziges Bild mehrere kleine Bildchen zeichnen, so wie in Abbildung 10.3, so geschieht dies mit den beiden Befehlen

```
subplot(m,n,pos)
plot(x,y)
```

Mit `subplot(m,n,pos)` wird das Bild in m mal n kleine Rechtecke aufgeteilt und mit `pos` wird festgelegt in welches der m mal n Rechtecke gezeichnet wird. Dabei werden die Rechtecke zeilenweise von links nach rechts und von oben nach unten durchnumeriert.

Die Balkendiagramme kann man mit `bar, hist` herstellen, die Tortendiagramme mit `pie`. Mehrdimensionale Graphiken kann man u. a. mit `plot3, surf` herstellen. Demonstrationsprogramme findet man mit `demo`, und auf der Homepage des Verfassers:

`http://www.math.uni-hamburg.de/home/opfer/aufgaben01w.html`

Zu den dort angegebenen Beispielen gehört auch der Turm von Hanoi ein nichttriviales Beispiel zur rekursiven Funktionsdefinition, mit der bunten Visualisierung.

11 Entstehung der Programmiersprachen FORTRAN und MATLAB

Wir schreiben die beiden Programmiersprachen mit großen Buchstaben weil sie aus Abkürzungen abgeleitet sind. Die Programmiersprache Pascal [1] dagegen ist nach BLAISE PASCAL (* Clermont-Ferrand 1623 – †Paris 1662) benannt, und wird daher nicht durchgehend groß geschrieben.

11.1 Entstehung von FORTRAN

Die Programmiersprache FORTRAN (*formula translation*) ist 1954 entstanden, entwickelt von einem IBM-Team unter der Leitung von JOHN BACKUS. [2] Nach GERD GROTEN: Programmieren in Fortran 90/95, Forschungszentrum Jülich, FZJ-ZAM-BHB-0124, 5. Aufl. 1999[3] hat FORTRAN die folgende Entwicklung:

- 1954 FORTRAN I, erstes Handbuch 1957, erster Compiler 1957 bei IBM,
- 1966 FORTRAN IV, eine Einführung gab es bereits 1965 von MCCRACKEN[4]
- 1977 FORTRAN 77, letztes Lochkartenformat
- 1991 FORTRAN 90, erste Fassung mit freier Formatierung
- 1997 FORTRAN 95, gute Information in METCALF & REID [5]

11.2 Entstehung von MATLAB

Die Programmiersprache MATLAB (*matrix laboratory*) ist neueren Datums, geht zurück auf CLEVE MOLER, [6] der eine erste Version 1987 entworfen hat. Sie wird kommerziell hergestellt von der Firma The MathsWorks, USA. Zur Zeit (Oktober 2002) ist die Version 6.5 (Release 13) aktuell. Informationen über die Version 6 von MATLAB sind enthalten in einem Buch der Brüder Higham.[7]

[1] N. WIRTH, The Programming Language PASCAL, Acta Informatica, **1**, 1971, 35–63.
[2] The Man behind FORTRAN, Computing Report, II (4), 1966.
[3] S. 3 von http://www.fz-juelich.de/zam/docs/bhb/bhb_html/d0124/d0124.html
[4] D. D. MCCRACKEN, A guide to FORTRAN IV programming, Wiley, New York, 1965, 151 S.
[5] M. METCALF & J. REID, FORTRAN 90/95 explained, 2nd ed., Oxford University Press, Oxford, 2002, 341 S.
[6] C. MOLER, M. ULLMAN, J. LITTLE, & S. BANGERT, 386-MATLAB for 80386 Personal Computer, The MathsWorks, Sherborn, MA, 1987.
[7] D. J. HIGHAM & N. J. HIGHAM: MATLAB Guide, siam, Philadelphia, 2000, 283 S.

Stichwortverzeichnis

Abbruch (bei Fehler), 16
`advance="no"`, 30
ALGOL60, 23
Ampersand &, 6
`.and.`, 19
`ans`, 16
Anweisung, 1, 6, 16
    bedingt, 18
    unbedingt, 16
Anzeigeformat, 15
Apostroph in FORTRAN-Texten, 13
äquidistant (gleichabständig), 15
äquivalent (logisch), 19
Arithmetik
    Diskrepanz MATLAB-FORTRAN, 17
Arithmetik mit Punkt: `.*, ./, .^`, 17
array, 14
Ausgabeformate, 28
Ausrufezeichen !, 6
`axis`, 33

John Backus, 23, 35
Balkendiagramm, 32, 34
`bar`, 34
Bedingte Anweisung, 18
Befehl, 1, 6
Besetzung eines Feldes, 11
Elsbeth Bredendiek, iv
Buchstabe, 9
Byte, 13

`call`, 23
call by value, 23
`case`, 18
`character`, 11, 12
`close`, 28
Compiler, 1
`complex`, 11, 12
    `(kind=dr)`, 12
Computer-Programm, 1
`cpu_time` (FORTRAN95), 26

`d0` Standard-Endung von `double precision`-Konstanten, 12, 13
data edit descriptor, 28
`date_and_time` (FORTRAN), 26
Dateinamen, 9
`demo`, 4, 34
`diary`, 31
`dimension`, 12
    Besetzung, 11
    Deklaration, 11
Division durch Null, 16
`do-loop`, 21
doppelt genau, 10, 12
    Endung `d0`, 13
`double precision`, 11, 12

Echo (MATLAB), 3, 7
Editor, 1, 6
Enter-Taste, 3, 6
`.equiv.`, 19
Erweiterung (Dateiname), 1, 3, 8, 9
`eval`, 18
explizite Variablendeklaration, 10
Extension (Dateiname), 1, 9

`f77`, FORTRAN-Compilierprogramm, 1
`f90`, FORTRAN-Compilierprogramm, 1
`f95`, FORTRAN-Compilierprogramm, 1
Fakultät, 26
`.false.`, 12
Fehler
    FORTRAN, 16
    FORTRAN-Programme, 1
    MATLAB-Programme, 4
    run-time (FORTRAN), 1
Feld, 14
    Besetzung, 11, 30
    mehr als 2 Indizes (FORTRAN), 30
    mehr als 2 Indizes (MATLAB), 15
Festkommazahl, 10
`feval`, 18
`find` (MATLAB), 20
`format`
    `bank`, 31
    `compact`, 31
    `long`, 31
    `rat`, 31
    `short e`, 31
    `short`, 31
Format (FORTRAN), 29
Format (MATLAB), 7
Formatkennung (FORTRAN), 28
`for`-Schleife, 22
FORTRAN, 1, 35
FORTRAN IV, 35
FORTRAN77, 35
FORTRAN90, 35
FORTRAN95, 35
FORTRAN=*formula translation*, 35
`fprintf` MATLAB-Format, 31
function
    FORTRAN, 8, 23

MATLAB, 26

g77, FORTRAN-Compilierprogramm, 1
Gedicht=String-Array, 13
Genauigkeit
FORTRAN, 10
MATLAB, 15
Gleichheitszeichen, 16
Gleitkommazahl (FORTRAN), 10
doppelt genau, 10
einfach genau, 10
vierfach genau, 10
`global`, 27
Globalvereinbarung, 10
Groß- und Kleinschreibung, 9
Gerd Groten, 35

`help`
`Name`, 4
`format`, 31
`precedence`, 19, 20
Desmond J. Higham, 35
Nicholas J. Higham, 35
`hist`, 34
`hold on - hold off`, 34

`if`-statement, 19
`implicit`
`double precision`, 10
`none`, 10
implizite Variablendeklaration, 10
`inf`, 16
`int2str` (MATLAB), 18
`integer`, 10–12
`(kind=ns)`, 12, 13
`integer, parameter`, 11, 12
`intent`
`intent(in)`, 23
`intent(inout)`, 23
`intent(out)`, 23
interaktive Arbeit, 1
intrinsic procedures, 24

Ernst Jandl, 32

John L. Kelley, 6
Kindergartenoperationen, 17
Klammern (MATLAB)
eckige, 14
runde, 14
Komma, 14
MATLAB, 7
Kommando, 1, 6
Kommentar, 6
Zeichen % (MATLAB), 5, 7
Zeichen ! (FORTRAN), 6
komplex konjugiert, 17
Konjugieren
komplexe Zahl, 17
Konstante, 12

LaTeX, 33
Laufvariable (MATLAB), 14
leere Anweisung, 21
leere Matrix `[]`, 15
leeres Feld, 15
Leerzeichen, 9, 14
Lesen
FORTRAN-Datei, 30
lineares Gleichungssystem, 17
Linksdivision, 17
`linspace`, 15
`logical`, 11, 12
logische Ausdrücke (FORTRAN und MATLAB), 20
lokale Variable, 27
`lookfor Stichwort`, 4

Maschinenprogramm, 1
MATLAB, 1, 23, 35
MATLAB-Format, 31
MATLAB=*matrix laboratory*, 35
Matrix, 14, 15
Matrixmultiplikation, 17, 18
Daniel D. McCracken, 35
mehrfache Genauigkeit (FORTRAN), 13
mehrzeilige Anweisung
FORTRAN: &, 6
MATLAB: ..., 7
Michael Metcalf, 35
Cleve Moler, 35
Christian Morgenstern, 13

Name
von Dateien, 9
von Variablen, 9
`NaN` (*not a number*), 16
`nargin`, 27
`nargout`, 27
negative Indizes
FORTRAN, 11
MATLAB, 14
`.nequiv.`, 19
nicht (logisch), 19
nicht äquivalent (logisch), 19
`.not.`, 19
`num2str` (MATLAB), 18

oder (logisch), 19
`open`, 28
`.or.`, 19

Pascal, 23, 35
`pi`=π, 7
`pie`, 34
`plot`, 20, 32
`plot3`, 34

Plotten (Zeichnen), 32
`precedence`, 19
`print`, 33
Prioritätsregeln
  Arithmetik
    FORTRAN, 17
    MATLAB, 17
  logische Operatoren, 19
Programm, 1
Programmwiederholungen, 21
  mit fester Anzahl, 21
  mit variabler Anzahl, 21
Prompt >>, 3
Prozentzeichen %, 7
Punkt-Operationen `.* ./ .^`(MATLAB), 17

Quellprogramm, 1

Rahmenbedingungen, 8
`rand`, 27
`random_number`, 26
`real`, 11, 12
  `(kind=xs)`, 12, 13
`record` (Pascal), 14
John Reid, 35
rekursiv, 26
result
  FORTRAN, 8
run-time-Fehler, 1

`select case`, 18
`selected_int_kind`, 12
`selected_real_kind`, 11, 12
Semikolon, 14
  FORTRAN, 6
  MATLAB, 7
`sin`, 16
Skalarprodukt, 17
Sonderzeichen, 9
Spaltenvektor, 15
`sprintf` MATLAB-Format, 31
Standardformat (FORTRAN), 29
Starten (eines Programms), 1
status (Datei, FORTRAN), 28
Stoppuhr: `tic toc` (MATLAB), 27
String
  FORTRAN, 6, 11, 13, 29
  MATLAB, 15, 18
Stürzen
  Matrix, 17
`subplot`, 34
`subroutine`, 8
`surf`, 34
syntaktisch richtig, 8
syntaktischer Fehler, 1
Syntax, 1
`system_clock` (FORTRAN), 26

Tensor, 14
`tic toc` Stoppuhr, 27
`toolbox`, 5
Tortendiagramm, 32, 34
Transferfunktion, 18
Transponieren
  Matrix, 17
Trennzeichen, 14
`.true.`, 12
Turm von Hanoi, 27, 34
Typ-Deklaration, 12
`type`
  FORTRAN, 14
  MATLAB, 4
  Pascal, 12

Umlaute, 9
Unbedingte Anweisung, 16
und (logisch), 19
Unterprogramm, 23
  FORTRAN, 23
  MATLAB, 26
Unterprogramme (fest eingebaut), 24
Unterstreichungszeichen `_`, 9

Variable, 9
  ganzzahlig, 10
  global, 27
  komplex, 10
  logisch, 10
  lokal, 27
  Matrix, 10
  reell, 10
  Vektor, 10
Variablendeklaration
  explizit, 10, 27
  implizit, 10
Vektor, 14, 15
Vektorraumoperationen, 17
vernünftige Programmiersprache, 23, 26

Warnung (vor Fehlern), 16
Wertaufruf, 23, 26
Niklaus Wirth, 35
Wortlänge, 9, 14
  maximale Länge von Variablen, 9

Zahlformat (FORTRAN), 29
Zahlformat (MATLAB), 7
Zeilenbeginn, 29
Zeilenende, 6, 14
Zeilenvektor, 15
Zeitabfrage
  FORTRAN, 26
  MATLAB, 27
Ziffer, 9
Zufallszahlen, 25, 27
Zwischenraum, 9, 29