Anhang B.

Einführung in MATLAB

2.1. Grundlegende MATLAB-Befehle

Ruft man das Programm MATLAB mit dem Befehl `matlab` auf, so erscheinen auf dem Monitor einige Fenster. Auf den Linux-Rechnern des KIZ müssen Sie vorher die notwendigen Pfade ergänzen. Geben Sie dazu in einem Konsole-Fenster den Befehl option matlab ein. Von diesen ist das Befehl-Fenster der primäre Ort um Befehle einzugeben und Fehler oder Ergebnisse abzulesen! Das Prompt-Zeichen ≫ ist im Befehl-Fenster dargestellt und dort findet man üblicherweise einen blinkenden Cursor. Der blinkende Cursor und der MATLAB-Prompt zeigen einem, dass MATLAB eine Eingabe erwartet.

2.1.1. Einfache mathematische Operationen Genauso wie mit einem simplen Taschenrechner kann man auch mit MATLAB einfache mathematische Operationen ausführen, z.B. ergibt die Eingabe

```
≫ 3 + 4
```

die Ausgabe

```
ans =
     7
```

Man beachte, dass MATLAB im Allgemeinen keine Zwischenräume benötigt, um Befehle eindeutig zu verstehen. Alternativ zu dem obigen Beispiel können in MATLAB auch Variablen verwendet werden.

```
≫ a = 3
a =
     3
≫ b = 4
b =
     4
≫ c = a + b
c =
     7
```

MATLAB besitzt folgende einfache arithmetische Operationen

Operation	Symbol	Beispiel
Addition, $a + b$	+	5 + 3
Subtraktion, $a - b$	−	23−12
Multiplikation, $a \cdot b$	*	13.3 * 63.13
Division, $a$div b	/ or \	17/4 = 4\17
Potenz, a^b	^	3^4

Die Reihenfolge, in der eine Folge von Operationen abgearbeitet wird, lässt sich wie folgt beschreiben. Ausdrücke werden von links nach rechts ausgeführt, wobei die Potenzierung die höchste Priorität besitzt gefolgt von Punktoperation, sprich Multiplikation und Division. Die geringste Priorität haben Addition und Subtraktion. Mit Hilfe von Klammern kann diese Vorgehensweise geändert werden, wobei innere Klammern vor äußeren Klammern berechnet werden.

2.1.2. Variablen Wie in anderen Programmiersprachen hat auch MATLAB Regeln für Variablennamen. Eine Variable repräsentiert ein Datenelement, dessen Wert während der Programmausführung – gegebenenfalls mehrfach – geändert werden kann. Variablen werden anhand ihrer „Namen“ identifiziert. Namen bestehen aus ein bis neunzehn Buchstaben, Ziffern oder Unterstrichen, wobei das erste Zeichen ein Buchstabe sein muss. M an beachte, dass MATLAB Groß- und Kleinschreibung unterscheidet. (Windows ist im Gegensatz zu Linux nicht so restriktiv, da Sie jedoch Programme austauschen wollen, sollten Windows-Benutzer besondere Aufmerksamkeit walten lassen.)

Einer Variablen ist Speicherplatz zugeordnet. Wenn man eine Variable verwendet, dann meint man damit entweder den zugeordneten Speicherplatz oder den Wert, der dort augenblicklich abgespeichert ist. Einen Überblick über alle Variablen erhält man mit dem Befehl `who` oder `whos`, wobei letzterer die Angabe des benutzten Speicherplatzes beinhaltet.

Zusätzlich zu selbstdefinierten Variablen gibt es in MATLAB verschiedene spezielle Variablen. Diese lauten

spezielle Variablen	Wert
ans	standard Variablenname benutzt für Ergebnisse
pi	3.1415 . . .
eps	Maschinengenauigkeit
flops	Zähler für die Anzahl der Fließkommaoperationen
inf	steht für Unendlich (eng. infinity). z.B. 1/0
NaN	eng. Not a Number, z.B. 0/0
i (und) j	$i = j = \sqrt{-1}$

In MATLAB kann der Speicherplatz, der durch Variablen belegt ist, durch den Befehl `clear` wieder freigegeben werden, z.B.

```
≫ clear a b c
```

2.1.3. Kommentare und Punktion Der Text, der nach einem Prozentzeichen % folgt, wird in MATLAB als Kommentar verstanden

```
≫ dummy = 4 % Wert von dummy
dummy =
     4
```

Mehrere Befehle können in eine Zeile geschrieben werden, wenn sie durch Kommata oder Semikola getrennt werden. Kommata veranlassen MATLAB, die Ergebnisse anzuzeigen. Bei einem Semikolon wird die Ausgabe unterdrückt. Durch eine Sequenz von drei Punkten kann man einen Befehl in der folgenden Zeile fortsetzen.

2.1.4. Spezielle Funktionen Eine unvollständige Liste von Funktionen, die MATLAB bereitstellt, ist im folgenden dargestellt. Die meisten Funktionen sind so definiert, wie man sie üblicherweise benutzt.

```
≫ y = cos(pi)
y =
     -1
```

MATLAB bezieht sich im Zusammenhang mit Winkelfunktionen auf das Bogenmaß.

Funktion	**Bedeutung**
abs(x)	Absolutbetrag
cos(x)	Kosinus
exp(x)	Exponentialfunktion: e^x
fix(x)	rundet auf die nächste, vom Betrag her kleinere ganze Zahl
floor(x)	rundet auf die nächste, kleinere ganze Zahl
gcd(x,y)	größter gemeinsamer Teiler von x und y
lcm(x,y)	kleinstes gemeinsames Vielfaches von x und y
log(x)	natürlicher Logarithmus
rem(x,y)	Modulo (eng. remainder of division), z.B. rem(5,2)=1
sign(x)	Signum Funktion, z.B. sign(2.3) = 1, sign(0) = 0, sign(-.3) = -1
sin(x)	Sinus
sqrt(x)	Quadratwurzel
tan(x)	Tangens

2.1.5. Skript-Dateien Für einfache Probleme ist es schnell und effizient, die Befehle am MATLAB-Prompt einzugeben. Für größere und umfangreichere Aufgabenstellungen bietet MATLAB die Möglichkeit, sogenannte Skript-Dateien zu verwenden, in denen die Befehle in Form einer Textdatei aufgeschrieben sind und die man am Prompt übergibt. MATLAB öffnet dann diese Dateien und führt die Befehle so aus, als hätte man sie am Prompt eingegeben. Die Datei nennt man Skript-Datei oder M-Datei, wobei der Ausdruck M-Datei daher rührt, dass diese Dateien das Suffix `.m` haben, z.B. `newton.m`. Um eine M-Datei zu erstellen, ruft man einen Editor auf und speichert die Datei in dem Verzeichnis, von dem aus man MATLAB gestartet hat oder starten wird. Die Datei, z.B. `newton.m`, wird in MATLAB dann durch Eingabe von `newton` am Prompt aufgerufen.

Für die Benutzung von Skript-Dateien hat MATLAB unter anderem folgende hilfreichen Befehle

M-Datei-Funktionen	
disp(ans)	zeigt den Wert der Variablen `ans`, ohne ihren Namen auszugeben
input	erwartet vom Benutzer eine Eingabe
keyboard	übergibt zeitweise die Kontrolle an die Tastatur
pause	hält das Programm an, bis eine Taste betätigt wird

Die folgende Skript-Datei `beispiel1.m`

```
% beispiel1.m
% Beispiel fuer eine Skript-Datei
tmp = input(' Geben Sie bitte eine Zahl an >' );
3 * tmp;
```

führt zu der Ausgabe

```
≫ beispiel1
Geben Sie bitte eine Zahl an > 6
ans =
     18
```

2.1.6. Dateiverwaltung MATLAB unterstützt eine Vielzahl von Dateiverwaltungsbefehlen, welche es einem ermöglichen Dateien zu listen, Skript-Dateien anzusehen oder zu löschen und Verzeichnisse zu wechseln.

Datei-Managment-Funktionen	
cd path	wechselt in das Verzeichnis `path`
delete beispiel	löscht die Datei `beispiel.m`
ls	zeigt alle Dateien im aktuellen Verzeichnis an
pwd	zeigt den aktuellen Verzeichnispfad an
type beispiel	zeigt den Inhalt der Datei `beispiel.m` im Befehl-Fenster
what	zeigt alle M- und MAT-Dateien im aktuellen Verzeichnis an

2.1.7. Hilfe

Online-Hilfe: Da sich nicht jeder Benutzer alle MATLAB-Befehle merken kann oder auch von einigen auch nur die Syntax unklar ist, bietet MATLAB die Möglichkeit der Online-Hilfe. Dabei gibt es prinzipiell mehrere Möglichkeiten. Ist einem ein Befehl bekannt und man sucht Informationen über die Syntax, so gibt es den Befehl `help`.

```
≫ help sqrt
  SQRT   Square root.
         SQRT(X) is the square root of the elements of X. Complex
         results are produced if X is not positive.

         See also SQRTM.
```

Als Beispiel haben wir uns hier die Hilfe zu dem Befehl `sqrt` ausgeben lassen.

Die andere Möglichkeit der von MATLAB gelieferten Hilfe ist durch den Befehl `lookfor` gegeben. Hier durchsucht das Programm alle ersten Zeilen der MATLAB Hilfe-Kennwörter und Skript-Dateien die im MATLAB Suchpfad zu finden sind. Das Bemerkenswerte dabei ist, dass dieser Begriff kein Befehl zu sein braucht.

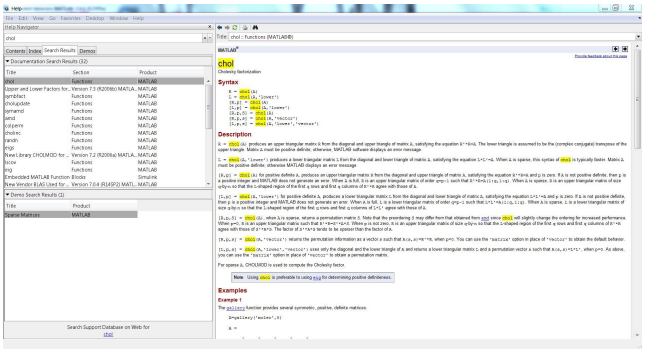
```
≫ lookfor cholesky
  CHOL    Cholesky factorization
≫  CHOL   Cholesky factorization.
  CHOL(X) uses only the diagonal and upper triangle of X.
  The lower triangular is assumed to be the (complex conjugate)
  transpose of the upper.  If X is positive definite, then
  R = CHOL(X) produces an upper triangular R so that R'*R =X.
  If X is not positive definite, an error message is printed.

  With two output arguments, [R,p] = CHOL(X) never produces an
  error message.  If X is positive definite, then p is 0 and R
  is the same as above.   But if X is not positive definite, then
  p is a positive integer and R is an upper triangular matrix of
  order q = p-1 so that R'*R = X(1:q,1:q).
```

```
≫ lookfor  factorization

CHOL    Cholesky factorization.
QRDELETE Delete a column from the QR factorization.
QRINSERT Insert a column in the QR factorization.
SYMBFACT Symbolic factorization analysis.
```

Eine weitere Möglichkeit, sich Hilfe zu verschaffen, besteht darin, das Helpdesk aufzurufen. Wenn Sie helpdesk am Prompt eingeben, öffnet sich die folgende Hilfsumgebung



2.2. Mathematik mit Matrizen

2.2.1. Matrixkonstruktion und Adressierung Beschäftigen wir uns nun mit Möglichkeiten Matrizen in MATLAB zu definieren.

einfache Matrix Konstruktionen	
x=[1 4 2*pi 4]	erstelle einen Zeilenvektor x mit Einträgen
x=$anfang$:$ende$	erstelle einen Zeilenvektor x beginnend mit $anfang$, Inkrement 1 und endend mit $ende$
x=$anfang$:$inkrement$:$ende$	Ähnliches wie oben mit dem Inkrement $inkrement$
x=linspace($anfang$,$ende$,n)	erzeugt einen Zeilenvektor der Dimension n mit $x(i) = \frac{(n-i)\cdot anfang+(i-1)\cdot ende}{n-1}$

Im Folgenden sind einige charakteristische Beispiele aufgeführt.

```
≫ B = [1 2 3 4; 5 6 7 8]
B =
     1 2 3 4
     5 6 7 8
```

Der Operator ′ liefert für reelle Matrizen die Transponierte.

```
≫ C = B'
C =
     1 5
     2 6
     3 7
     4 8
```

Der Doppelpunkt `:` in der zweiten Komponente spricht alle vorhandenen Spalten an, d.h. er ist ein zu `1:4` äquivalenter Ausdruck.

```
≫ C = B(1,:)
C =
     1 2 3 4
≫C = B(:,3)'
C =
     3 7
```

Es lassen sich auch einzelne Komponenten neu definieren.

```
≫A = [1 2 3; 4 5 6; 7 8 9]
A =
     1 2 3
     4 5 6
     7 8 9
≫ A(1,3) = 9
A =
     1 2 9
     4 5 6
     7 8 9
```

Ist ein Eintrag noch nicht definiert, so verwendet MATLAB die minimale Erweiterung dieser Matrix und setzt undefinierte Einträge zu Null.

```
≫ A(2,5) = 4
A =
     1 2 9 0 0
     4 5 6 0 4
     7 8 9 0 0
```

Im Folgenden werden die Vektoren `(3,2,1)` und `(2,1,3,1,5,2,4)` dazu verwendet, die Matrix `C` zu indizieren, d.h. `C` hat die Struktur

```
A(3,2) A(3,1) A(3,3) A(3,1) A(3,5) A(3,2) A(3,4)
A(2,2) A(2,1) A(2,3) A(2,1) A(2,5) A(2,2) A(2,4)
A(1,2) A(1,1) A(1,3) A(1,1) A(1,5) A(1,2) A(1,4)
```

In MATLAB erhält man nun

```
≫ C=A(3:-1:1,[2 1 3 1 5 2 4])
C =
     8 7 9 7 0 8 0
     5 4 6 4 4 5 0
     2 1 9 1 0 2 0
```

Ein weiteres Beispiel für Indizierung ist

```
≫ C=C(1:2,2:3)
C =
     7 9
     4 6
```

Im nächsten Beispiel wird ein Spaltenvektor dadurch konstruiert, dass alle Elemente aus der Matrix `C` hintereinander gehängt werden. Dabei wird spaltenweise vorgegangen.

```
≫ b=C(:)'
b =
     7 4 9 6
```

Das Löschen einer ganzen Zeile oder Spalte kann durch das Umdefinieren in eine 0×0-Matrix geschehen, z.B.

```
≫ C(2,:)=[ ]
C =
     7 9
```

2.2.2. Skalar-Matrix-Operationen In MATLAB sind Skalar-Matrix-Operationen in dem Sinne definiert, dass Addition, Subtraktion, Division und Multiplikation mit einem Skalar elementweise durchgeführt werden. Es folgen zwei erklärende Beispiele.

```
≫ B - 1
ans =
     0 1 2 3
     4 5 6 7
≫ 9 + 3 * B
ans =
     12 15 18 21
     24 27 30 33
```

2.2.3. Matrix-Matrix-Operationen Die Operationen zwischen Matrizen sind nicht so kanonisch zu definieren wie die zwischen Skalar und Matrix, insbesondere sind Operationen zwischen Matrizen unterschiedlicher Dimension schwer zu definieren. Des Weiteren sind die Operationen $*$ und $.*$, bzw. / und ./ sowie \ und .\ zu unterscheiden. In nachfolgender Tabelle sind die Matrixoperationen beschrieben.

	komponentenweise Matrixoperationen
Beispieldaten	$a = [a_1, a_2, \ldots, a_n]$, $b = [b_1, b_2, \ldots, b_n]$, c ein Skalar
komp. Addition	$a + c = [a_1 + c\ a_2 + c \ldots a_n = c]$
komp. Multiplikation	$a * c = [a_1 \cdot c\ a_2 \cdot c\ \ldots\ a_n \cdot c]$
Matrix–Addition	$a + b = [a_1 + b_1\ a_2 + b_2 \ldots a_n + b_n]$
komp. Matrix-Multiplikationen	$a. * b = [a_1 \cdot b_1\ a_2 \cdot b_2 \ldots a_n \cdot b_n]$
komp. Matrix-Div. von rechts	$a./b = [a_1/b_1\ a_2/b_2 \ldots a_n/b_n]$
komp. Matrix-Div. von links	$a.\backslash b = [b_1/a_1\ b_2/a_2 \ldots b_n/a_n]$
komp. Matrix-Potenz	$a.\hat{}c = [a_1^c\ a_2^c \ldots a_n^c]$
	$c.\hat{}a = [c^{a_1}\ c^{a_2} \ldots c^{a_n}]$
	$a.\hat{}b = [a_1^{b_1}\ a_2^{b_2} \ldots a_n^{b_n}]$

Es folgen nun einige Beispiele zu Matrixoperationen

```
≫ g=[1 2 3; 4 5 6]; % zwei neue Matrizen
≫ h=[2 2 2; 3 3 3];
≫ g+h   % addiere g und h komponentenweise
ans =
     3  4  5
     7  8  9
≫ ans-g   % subtrahiere g von der vorherigen Antwort
ans =
     2  2  2
     3  3  3
≫ h.*g   % multipliziere g mit h komponentenweise
ans =
      2   4   6
     12  15  18
≫ g*h'   % multipliziere g mit h'
ans =
     12  18
     30  45
```

2.2.4. Matrix-Operationen und -Funktionen Nun einige Operationen die sich auf Matrizen anwenden lassen.

Matrixfunktionen	
reshape(A,m,n)	erzeugt aus den Eintägen der Matrix A eine $m \times n$-Matrix, wobei die Einträge spaltenweise aus A gelesen werden.
diag(A)	ergibt die Diagonale von A als Spaltenvektor
diag(v)	erzeugt Diagonalmatrix mit dem Vektor v in der Diagonalen
tril(A)	extrahiert den unteren Dreiecksanteil der Matrix A
triu(A)	extrahiert den oberen Dreiecksanteil der Matrix A

Es folgen einige Beispiele

```
≫ g = linspace(1,9,9)   % ein neuer Zeilenvektor
g =
     1     2     3     4     5     6     7     8     9
≫ B = reshape(g,3,3)   % macht aus g eine 3 x 3 Matrix
B =
     1     4     7
     2     5     8
     3     6     9
≫ tril(B)
ans =
     1     0     0
     2     5     0
     3     6     9
```

Funktion	Bedeutung
R=chol(A)	Choleskyzerlegung
cond(A)	Konditionszahl der Matrix A
d=eig(A)	Eigenwerte und -vektoren
[V,d]=eig(A)	
det(A)	Determinante
hess(A)	Hessenbergform
inv(A)	Inverse
[L,U]=lu(A)	Zerlegung gegeben durch Gauss-Algorithmus
norm(A)	euklidische-Norm
rank(A)	Rang der Matrix A

Bemerkung: Der \ Operator ist auch für Matrizen definiert und liefert in Kombination mit Vektoren für reguläre Matrizen ihre Inverse, d.h. `A^(-1)x=A\x` .

2.2.5. Spezielle Matrizen Einige häufig auftretende spezielle Matrizen sind im folgenden aufgelistet.

spezielle Matrizen	
eye(n)	erzeugt eine Einheitsmatrix der Dimension n
ones(m,n)	erzeugt eine $m \times n$-Matrix mit den Einträgen 1
zeros(m,n)	erzeugt eine $m \times n$-Matrix mit den Einträgen 0

2.2.6. Spezielle Funktionen für schwachbesetzte Matrizen Bei vielen numerischen Anwendungen treten schwachbesetzte Matrizen auf. MATLAB hat für solche Matrizen besondere Sparse-Funktionen, die dieser Eigenschaft Rechnung tragen.

Funktion	Bedeutung
find(A)	findet Indizes von Nichtnulleinträgen
nnz(A)	Anzahl an Nichtnulleinträgen
spdiags(v)	erzeugt eine Sparse-Diagonalmatrix mit Vektor v als Diagonale
speye(n)	erzeugt eine Sparse-Einheitsmatrix
spy(A)	visualisiert die Struktur der Matrix A

Kurze Illustration der Funktionsweise obiger Befehle anhand einiger Beispiele.

```
≫ E=eye(100);    % vollbesetzte 100 x 100 Einheitsmatrix
≫ Es=sparse(E);  % Sparse-Version von E
≫ whos
      Name      Size    Elements  Bytes  Density  Comple
         E   100 by 100    10000  80000     Full      No
        Es   100 by 100      100   1600   0.0100      No
 Grand total is 10100 elements using 81600 bytes
≫ A=spdiags([7*ones(4,1),ones(4,1),2*ones(4,1)],[-1,0,1],4,4);
≫ nnz(A)
ans =
     10
≫ full(A)
ans =
      1   2   0   0
      7   1   2   0
      0   7   1   2
      0   0   7   1
```

Als Beispiel für `spy` sei hier die Besetzungstruktur einer 3D-FEM Steifigskeitsmatrix in Abb. B.1 gezeigt.

2.3. Datenverwaltung

Für die meisten Anwendungen genügt es, Datenfelder in einem Format abzuspeichern und wieder laden zu können. Die Befehle `load` und `save` setzen voraus, dass die Daten in einem

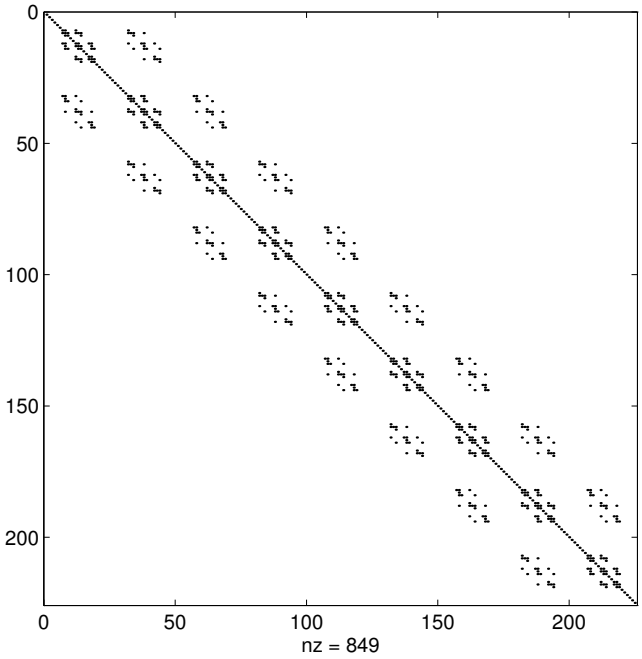


Abbildung B.1.: Besetzungstruktur einer 3D-FEM Steifigskeitsmatrix

System unabhängigen, binären Format in einer Datei mit dem Suffix `.mat` gespeichert sind oder in einem einfachen ASCII-Format vorliegen.

2.3.1. Daten speichern Im Folgenden wird eine 3×5 -Matrix im binär-Format in der Datei `A.mat` gespeichert. Diese Daten sind sehr kompakt gespeichert.

```
≫ A=zeros(3,5);
≫ save A
```

Gibt man sich aber den Inhalt dieser Datei auf dem Bilschirm aus, so gibt er wenig Sinn. Möchte man sich also z.B. einen Lösungsvektor sichern um ihn später „per Hand zu analysieren“ so speichere man die Daten als ASCII-Datei, dabei kann man wählen zwischen einer 8-stelligen oder 16-stelligen Abspeicherung.

```
≫ save mat1.dat A -ascii          %  8-stellige Speicherung
≫ save mat2.dat A -ascii -double  % 16-stellige Speicherung
```

Hier wurde die Matrix mit 8-stelligem Format in der Datei `mat1.dat` gespeichert, bzw. 16-stellig in der Datei `mat2.dat`.

2.3.2. Daten laden Mit dem Befehl `load A` versucht MATLAB die in `A.mat` gespeicherten Daten in einem Datenfeld `A` zu speichern. Auch ASCII-Dateien kann MATLAB lesen. Da es hier jedoch keine Standardendung gibt, ist die Datei inklusive Endung anzugeben. Es ist darauf zu achten, dass ein rechteckiges Feld an Daten vorliegt, d.h. dass m Zeilen

mit jeweils n numerischen Werten vorliegen. MATLAB erstellt dann eine $m \times n$-Matrix mit dem Namen der Datei ohne Suffix.

```
≫ load  mat1.dat
≫ whos
    Name        Size    Elements  Bytes  Density  Complex
    mat1       3 by 5         15    120     Full    No
 Grand total is 15 elements using 120 bytes
```

2.4. Ausgabe von Text

Mit dem Befehl `fprintf` lassen sich Strings, d.h. Zeichenfolgen, auf dem Bildschirm ausgeben.

```
≫ fprintf('\n Hello world %12.3e\n',4);
  Hello world    4.000e+00
```

Man sieht, dass der auszugebende Text zusätzliche Zeichen enthält, die nicht mit ausgedruckt werden. Diese Zeichen nennt man Escape-Sequenzen. In obigem Beispiel ist die Sequenz `\n` eingebaut. `\n` steht für „newline“ und sorgt dafür, dass bei der Textausgabe an diesen Stellen eine neue Zeile begonnen wird. Der Ausdruck `%12.3e` dient als Platzhalter für einen reellen Wert, der durch Komma getrennt hinter der Zeichenkette folgt. Dabei sei die Zahl in Exponentialdarstellung auszugeben, wofür 12 Stellen mit 3 Nachkommastellen bereitgestellt. Im Beispiel ist dies der Wert 4. Anstatt eines expliziten Wertes können auch Variablen oder Ausdrücke, z.B. `3 * 4` stehen. Ein Platzhalter kann mehrfach in einem `printf`-Befehl vorkommen. In diesem Fall müssen hinter der Zeichenkette genau so viele Werte folgen, wie Platzhalter angegeben sind. Die Reihenfolge der Werte muss mit der Reihenfolge der Platzhalter übereinstimmen, da die Ausdrücke von links nach rechts bewertet werden. Die Escape-Sequenzen dürfen im Text an beliebiger Stelle stehen.

Ausgabeformate

Befehl	**Ausgabe**
fprintf('%.0e\n',1.234567)	1e00+
fprintf('%.2e\n',1.234567)	1.23e00+
fprintf('%.5e\n',1.234567)	1.23456e00+
fprintf('%10.0e\n',1.234567)	1e00+
fprintf('%10.2e\n',1.234567)	1.23e00+
fprintf('%10.5e\n',1.234567)	1.2346e00+
fprintf('%10.2f\n',1.234567)	1.23
fprintf('%10.5f\n',1.234567)	1.23457

MATLAB rundet numerische Werte bei der Ausgabe, wenn nötig!

2.5. Kontrollbefehle

2.5.1. For-Schleifen **1.** Mit jeglicher gültigen Matrix-Darstellung lässt sich eine FOR-Schleife definieren, z.B.

```
≫ data = [1 7 3 2; 5 4 7 2]
data =
     1 7 3 2
     5 4 7 2
≫ for n=data
   x=n(1)-n(2)
  end
x =
     -4
x =
      3
x =
     -4
x =
      0
```

2. FOR-Schleifen können nach Belieben geschachtelt werden.

```
≫ for k=3:5
   for l=4:-1:2
    A(k,l)=k^2-l;
   end
  end
≫ A
A =
      0  0  0  0
      0  0  0  0
      0  7  6  5
      0 14 13 12
      0 23 22 21
```

3. FOR-Schleifen sollten vermieden werden, wann immer sie durch eine äquivalente Matrix- Darstellung ersetzt werden können. Der folgende Ausdruck ist darunten in optimierter Version aufgeführt.

```
≫ for n=1:10
   x(n)=sin(n*pi/10);
  end
≫x
x =
     Columns 1 through 7
       0.3090  0.5878  0.8090  0.9511  1.0000  0.9511  0.8090
     Columns 8 through 10
       0.5878  0.3090  0.0000
```

```
≫ n=1:10;
≫ x=sin(n*pi/10);
≫ x
x =
     Columns 1 through 7
       0.3090   0.5878   0.8090   0.9511   1.0000   0.9511   0.8090
     Columns 8 through 10
       0.5878   0.3090   0.0000
```

4. Um die Ausführungsgeschwindigkeit zu maximieren, sollte benötigter Speicherplatz vor Ausführung der FOR-Schleife alloziert werden.

```
≫ x=zeros(1,10);
≫ x(1)=0.5;
≫ for n=2:10
    x(n)=x(n-1)*sin(n*pi/10);
  end
≫ x
x =
     Columns 1 through 7
       0.5000   0.2939   0.2378   0.2261   0.2261   0.2151   0.1740
     Columns 8 through 10
       0.1023   0.0316   0.0000
```

2.5.2. WHILE-Schleifen

WHILE-Schleifen sind wie folgt aufgebaut.

```
while Aussage
      Anweisungen
end
```

Die Anweisungen zwischen `while` und `end` werden so lange ausgeführt, wie `Aussage` wahr ist, z.B.

```
≫ num=0; EPS=1;
≫ while (1+EPS) > 1
    EPS=EPS/2;
    num=num+1;
  end
≫ num
num =
      53
```

2.5.3. IF-ELSE-END Konstrukte Eine IF-ELSE-END-Schleife enthält nach dem IF eine Aussage, die daraufhin überprüft wird, ob sie wahr oder falsch ist. Ist sie wahr, so werden die in den folgenden Zeilen stehenden Anweisungen ausgeführt und die Schleife beendet. Ist sie falsch, so erfolgen die Anweisungen, die dem ELSE folgen (ELSE ist optional). Solch ein Konstrukt kann erweitert werden um beliebig viele ELSIF Befehle, die dieselbe Funktion haben wie der am Anfang stehende IF Befehl, aber nur beachtet werden, falls alle vorher überprüften Aussagen falsch sind.

```
if Aussage1
      Anweisungen, wenn Aussage1 wahr
elseif Aussage2
      Anweisungen, wenn Aussage1 falsch und Aussage2 wahr
else
      Anweisungen, wenn Aussage1 und Aussage2 falsch
end
```

2.5.4. Relationen und logische Operatoren Einige Relationen und logische Operatoren sind in den folgenden Tabellen gelistet.

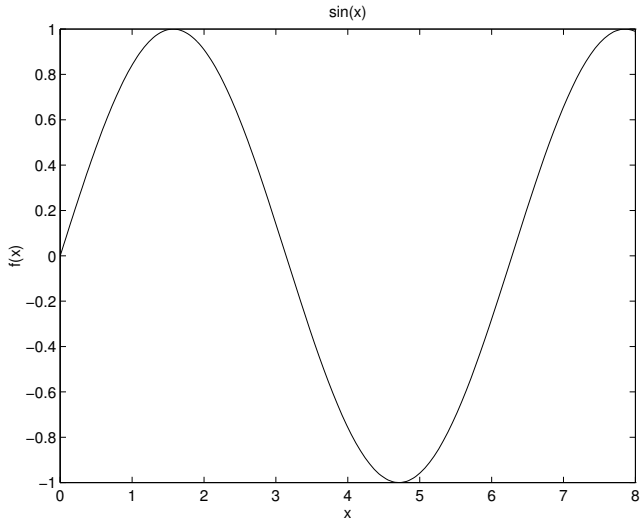
Relationen	
<	kleiner als
<=	kleiner als oder gleich
>	größer als
>=	größer als oder gleich
==	gleich
~=	ungleich

logische Operatoren	
&	UND
|	ODER
~	NICHT

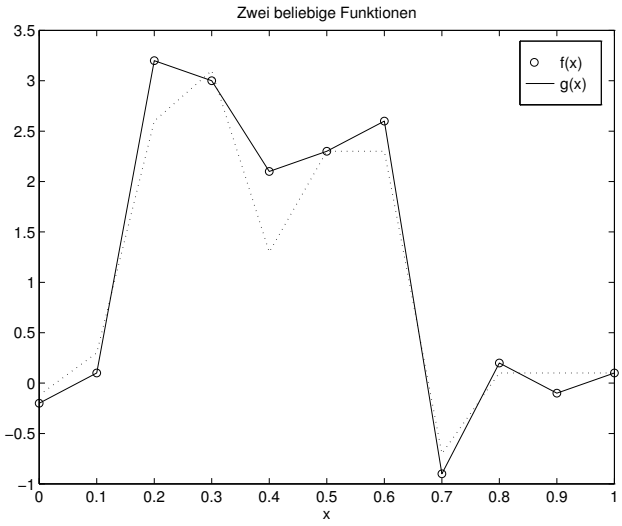
2.6. Graphische Darstellung

2.6.1. Zweidimensionale Graphiken

```
≫ f='sin(x)';
≫ fplot(f,[0 8]);
≫ title(f),xlabel('x'),ylabel('f(x)');
```




```
≫ x=0:0.1:1;
≫ y=[-0.2 0.1  3.2  3  2.1  2.3  2.6 -0.9 0.2 -.1 .1];
≫ z=[-0.12 0.3  2.6  3.1  1.3  2.3  2.3 -0.7 0.1 .1 .1];
≫ plot(x,y,'o',x,y,x,z,':');
≫ title('Zwei beliebige Funktionen'),xlabel('x');
≫ legend('f(x)','g(x)')
```



Linientypen und Farben

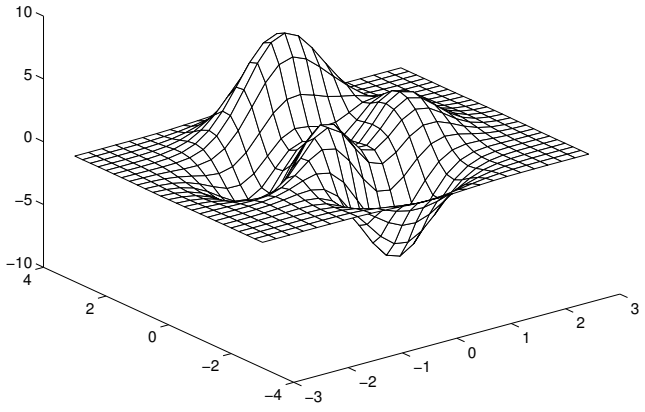
Symbol	Farbe	Symbol	Linientyp
y	gelb	·	Punkt
m	magenta	○	Kreis
c	cyan	x	x-Markierung
r	rot	+	+-Markierung
g	grün	∗	Sternchen
b	blau	−	durchgezogene Linie
w	weiß	:	gepunktete Linie
k	schwarz	−.	Strichpunkt-Linie
		−−	gestrichelte Linie

2-D Graphikanweisung

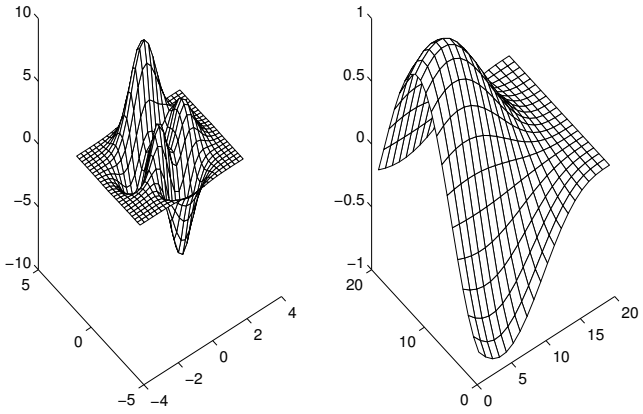
axis	modifiziert die Axen-Proportionen
clf	löscht die Graphik im Graphik-Fenster
close	schließt das Graphik-Fenster
grid	erzeugt ein achsenparalleles Gitter
hold	ermöglicht das Überlagern von Graphiken
subplot	erstellt mehrere Teilgraphiken in einem Fenster
text	gibt Text an vorgegebener Stelle aus
title	zeigt einen Titel an
xlabel	beschriftet die x-Achse
ylabel	beschriftet die y-Achse
colormap(white)	wechselt die Farbtabelle, für S/W-Monitore

2.6.2. Dreidimensionale Graphiken

```
≫ [X,Y,Z] = peaks(25)
≫ mesh(X,Y,Z)
```



```
≫ subplot(1,2,1);
≫ [X,Y,Z] = peaks(25);
≫ mesh(X,Y,Z);
≫ subplot(1,2,2);
≫ X=1:20;
≫ Y=1:20;
≫ Z(X,Y)=(-(cos(X/4)))'*(sin((20-Y)/10).^3);
≫ mesh(X,Y,Z);
```



2.6.3. Graphiken drucken

Graphik-Druckbefehl		
print [-dAusgabetyp] [-Optionen] [Dateiname]		
Ausgabetyp	-dps	Postscript für Schwarzweißdrucker
	-dpsc	Postscript für Farbdrucker
	-deps	Encapsulated Postcript
	-depsc	Encapsulated Color Postcript
Optionen	-P<Drucker>	Spezifiziert den zu benutzenden Drucker

Die Eingabe

```
≫ print fig4 -deps
```

erzeugt die Datei `fig4.eps`.

2.7. Fortgeschrittenes

2.7.1. MATLAB-Skripte Wie schon in Abschnitt 2.1 zu Skript-Dateien erwähnt, lässt sich eine Abfolge von MATLAB-Befehlen auch in einer Datei speichern. Diese kann man

dann am Befehl-Fenster aufrufen und die gespeicherte Folge von Befehlen wird ausgeführt. Solche Dateien werden als MATLAB-Skripte oder M-Files bezeichnet. Alle o.g. Befehle lassen sich in einer Datei z.B. mit dem Namen `abc.m` zusammenstellen. Für die Wahl des Dateinamens gelten dabei die folgenden Regeln:

- das erste Zeichen ist ein Buchstabe und
- die Datei hat die Endung **.m** .

Um ein solches M-File zu erstellen, ruft man den Editor auf, der es erlaubt Text einzugeben und diesen als Datei zu speichern. Wenn man den Editor gestartet hat, gebe man die folgenden Befehle ein und speichere die Datei unter dem Namen `abc.m`

```
a = 1;
b = 3;
c = -5;
x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Um nun die Befehle aus der Datei `abc.m` auszuführen, gibt man im MATLAB-Befehlsfenster

```
≫ abc
```

ein. MATLAB sucht im aktuellen Pfad nach der Datei `abc.m` und führt die darin enthaltenen Befehle aus. Das aktuelle Verzeichnis wird angezeigt, wenn man

```
≫ pwd
```

eingibt (pwd, engl. print working directory).

2.7.2. Erstellen eigener Funktionen Es wird sicherlich etwas komfortabler sein, eine eigene Funktion zu haben, der man a,b und c als Argument übergibt un die beiden Wurzeln als Eregebnis erhält, als jedesmal erneut ein eigenes M-File anzufertigen. Ein solches Programm könnte z.B. die folgende Datei `root.m` sein.

```
%-----------------
modified "abc.m"
%-----------------

a = input('Enter a: ');
b = input('Enter b: ');
c = input('Enter c: ');

x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Die Prozentzeichen in den erstn Zeilen dienen der Dokumentation. Alles was einem solchen Zeichen folgt, wird von MATLAB nicht ausgewertet, sprich interpretiert. Die Argumente werden in deieser Funktion jedoch nur bedingt übergeben und ausgegeben. Eine Funktion, die diese Aufgabe erfüllt, ist die folgende.

```
%-----------------
modified "abc.m"
%-----------------

function [x1, x2] = quadroot(a,b,c)

radical = sqrt(b^2 - 4*a*c);

x1 = (-b + radical)/(2*a)
x2 = (-b - radical)/(2*a)
```

Wenn eine Funktion keine Rückgabewerte hat, können die eckigen Klammern mit den Variablen und das Gleichheitszeichen fehlen. Fehlen die Eingabeparameter, so kann auch der Klammerausdruck nach `quadroot` fehlen. Auf die in der Funktion verwendeten Variablen, hier `radical`, kann man vom Befehlsfenster nicht zugreifen. Ist die Funktion in der Datei `quadroot.m` gespeichert, so erhält man die Wurzeln, indem man

```
[x1, x2] = quadroot(1,3,-5);
```

eingibt. Es kann vom Befehlsfenster oder einer anderen Datei immer nur die erste Funktion in einer Datei aufgerufen werden. Dies bedeutet, in einer Datei können mehrere Funktionen stehen, aber nur die erste Funktion kann extern aufgerufen werden. Alle weiteren Funktionen können nur von Funktionen in der gleichen Datei aufgerufen werden. Für den Anfang ist es einfacher, wenn jede Datei nur eine Funktion enthält. Entscheidend für den Funktionsnamen ist der Name der Datei.