MATLAB

Autorenkollektiv der PPM

Projektgruppe Praktische Mathematik
Fachbereich Mathematik
TU Berlin

12. April 2001

Inhaltsverzeichnis

1 Preliminarien und Start **1**

2 Interaktives Arbeiten mit Matlab **2**

3 Grundlagen **4**

3.1 Unix-Kommandos 4

3.2 Von Datei lesen, auf Datei schreiben 5

4 Vektoren und Matrizen **7**

4.1 Matrixelemente 7

4.2 Intervalldarstellung 8

4.3 Teilmatrizen 8

5 M-Files **9**

5.1 Script Files 9

5.2 Function Files 10

6 Programmieren in Matlab **11**

6.1 Kontrollstrukturen 11

6.1.1 Alternation 11

6.1.2 Iteration 11

6.2 Logische Operationen 13

6.3 Fehlersuche in Matlab - Debugging 17

6.3.1 Interaktives Programmieren 17

6.3.2 Script Files 18

6.3.3 Function Files 18

7 Graphik **19**

7.1 Erstellen von Plots 19

7.2 Einstellungsmöglichkeiten 21

7.3 Abspeichern und Ausdrucken 22

8 Hilfe **23**

Literaturverzeichnis **24**

1 Preliminarien und Start

Matlab ist die Abkürzung von *matrix laboratory* und die Bezeichnung für eine mathematische Software, die besonders geeignet ist für Matrizenoperationen. Durch die Eingabe des Kommandos

```
matlab
```

wird Matlab gestartet. Es erscheint ein Prompt an dem Befehle eingegeben werden können, die dann sofort ausgeführt (interpretiert) werden. Matlab ist also ein Interpreter und damit langsamer als ein kompiliertes z.B. in `C` oder `Fortran` geschriebenes Programm.

Variablen müssen nicht deklariert werden, wie sonst bei höheren Programmiersprachen üblich. In Matlab sind alle Variablen Matrizen mit möglicherweise komplexen Elementen. Ein Vektor ist eine Matrix mit nur einer Zeile oder Spalte und ein skalarer Wert eine 1x1 Matrix. Eine Zeichenkette wird ebenfalls als Vektor behandelt, dessen Elemente entsprechend der ASCII-Zeichensatz-Tabelle abgespeichert werden.[1]
Standardmäßig wird jede Variable als doppelt genaue Fließkommazahl gespeichert, auf dem Bildschirm jedoch nur mit vier Nachkommastellen dargestellt.

```
>> help format
```

gibt Auskunft über die möglichen Formate.

Man kann Matlab-Befehlsfolgen in den sogenannten M-Files zusammenfassen. Es gibt zwei Arten von M-Files: Script-Dateien und Funktionen. Auch solche Funktionen kann man selber schreiben. Neben elementaren mathematischen Funktionen bietet Matlab viele anerkannt gute numerische Algorithmen in Form von solchen Funktionen.

```
>> help elfun
```

Matlab kennt auch Kontrollstrukturen, also Abfragen und Iterationen, so daß eine Art Programmierung möglich ist.

[1] **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

```
>> help lang
```

In einzelnen Toolboxen sind weitere Funktionen zu bestimmten Themen oder Aufgaben zusammengefaßt (PDE[2] Toolbox, Optimization Toolbox, Simulink, etc).

Man kann von Matlab auch nur die grafischen Fähigkeiten nutzen und es als Plot Programm verwenden, dessen Ergebnisse sich leicht in LaTeX2e -Dokumente einbinden lassen.

2 Interaktives Arbeiten mit Matlab

Dieses Kapitel ist ein ausschließlich praktischer Einstieg in die interaktive Arbeit mit Matlab.

Erstelle eine Matrix A mit der folgenden Eingabe am Matlab-Prompt (am Zeilenende ein ENTER eingeben):

```
>> A = [ 1 2 3
         4 4 4
         5 6 9 ]
```

Das geht auch in der kompakteren Schreibweise:

```
>> A = [ 1 2 3 ; 4 4 4 ; 5 6 9]
```

Ein Semikolon am Ende einer Anweisung verhindert die Ausgabe auf dem Bildschirm:

```
>> A = [ 1 2 3 ; 4 4 4 ; 5 6 9];
```

Berechne die Tansponierte der Matrix A mit:

```
>> A'
```

Die Inverse der Matrix A soll in der Variablen B gespeichert werden:

```
>> B = inv(A)
```

[2]Partial Differential Equations

Das Ergebnis müsste sein:

```
-1.5000    0.0000    0.5000
 2.0000    0.7500   -1.0000
-0.5000   -0.5000    0.5000
```

Zur Überprüfung dient die Matrizenmultiplikation von A mit B:

```
>> erg = A*B
```

Das Ergebnis (wird der Variablen `erg` zugewiesen):

```
 1.0000    0.0000         0
      0    1.0000         0
 0.0000         0    1.0000
```

Subtrahiert man von `erg` die 3x3-Einheitsmatrix:

```
>> erg = erg - eye(3)
```

erhält man:

```
1.0e-14 *
-0.0999   -0.0555    0.0666
-0.2220   -0.1332    0.1332
-0.3886   -0.1887    0.2442
```

Es gelten die üblichen arithmetischen Operatoren (`+,-,*,/,^`) und Rangfolgeregeln. Die Operationen und Funktionen sind für ein Objekt vom Typ Matrix definiert und damit natürlich auch für Vektoren und Skalare.

Bisher wurden zwei Matlab-Funktionen verwendet: `inv` und `eye`. Mit

```
>> help functionname
```

kann man sich Informationen zu Funktionen anzeigen lassen; was diese Funktionen tun, mit welcher Syntax sie korrekt verwendet werden (oft gibt es mehrere Möglichkeiten) und welche Funktionen es noch zu diesem Thema gibt.

Die Matlab-Online-Hilfe kann mal also entweder mit einem thematischen Begriff oder mit einem Funktionsnamen aufrufen. Gibt man nur

```
>> help
```

ein, werden die möglichen Themenbereiche aufgelistet.

Weitere Funktionen:

`ones(n)`	erzeugt eine nxn Matrix mit Einsen
`zeros(n)`	erzeugt eine nxn Matrix mit Nullen
`min(x)`	findet das kleinste Element eines Vektors (– jeder Spalte einer Matrix)
`max(x)`	findet das größte Element eines Vektors (– jeder Spalte einer Matrix)
`sum(x)`	summiert die Elemente eines Vektors auf
`sort(x)`	sortiert einen Vektor in aufsteigender Reihenfolge
`mean(x)`	bildet den Mittelwert eines Vektors (oder der Vektoren einer Matrix)
`std(x)`	bildet die Standardabweichung enes Vektors (oder der ...)
`char(x)`	konvertiert numerische Vektoren entsprechend ASCII in Buchstaben

Übung: Was ist das Skalarprodukt eines Vektors $x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ mit seiner Transponierten?

3 Grundlagen

3.1 Unix-Kommandos

Matlab *kennt* einige Unix-Kommandos, wie zum Beispiel

```
ls, pwd, mkdir, cd, dir,
```

d.h. es gibt Matlab m-files, die den gleichen Namen und annähernd die gleiche Funktion haben wie entsprechende Unix-Kommandos. Zusätzlich ist es möglich, alle weiteren Unix-Kommandos an eine Shell[3] weiterzureichen, indem ein Ausrufungszeichen davor gesetzt

[3]Unix-Kommandointerpreter

wird. Dann wird das *echte* Unix-Kommando ausgeführt.

Es ist sinnvoll in einem eigenen Verzeichnis zu arbeiten. Wer nicht schon vorher eins angelegt hat, sollte dies nun tun.

```
>>  mkdir matlab
>>  cd matlab
```

Kopiert alle Dateien aus dem Verzeichnis `~ppm-soft/pub/vorlagen/matlab/` in dieses Verzeichnis:

```
>>  !cp ~ppm-soft/pub/vorlagen/matlab/*.* .
```

Hier wird der Kopierbefehl `cp` mit zwei Abkürzungen (Wildcards) verwendet. `*.*` steht für alle Dateien und der Punkt `.` steht für das aktuelle Verzeichnis.
Der Befehl

```
>>  ls
```

müßte das Ergebnis ergeben:

```
ans =

messen.m
tTemp.dat
```

3.2 Von Datei lesen, auf Datei schreiben

Daten von Datei einlesen ist in Matlab denkbar einfach, vor allem, wenn die Datei im ASCII-Format vorliegt und die Daten in Form von Zeilen und Spalten enthält. Mit

```
>>  load tTemp.dat
```

wird der Inhalt der Datei `tTemp.dat` einer Variablen (Matrix) mit dem Namen `tTemp` zugewiesen. Durch Eingabe des Variablennames – ohne Semikolon dahinter – bekommt man eine Terminalausgabe der Variablen.

Um diese Matrix wieder zu speichern (unter einem anderen Namen, z.B. `A`), wird das Kommando `save` verwendet.

```
>>  save A.dat tTemp -ascii
```

Die Datei soll ein lesbares Textfile sein, deshalb ist die Angabe `A.dat` und die Option `-ascii` erforderlich. Ohne diese Angaben wird der Inhalt der Variablen `tTemp` in die Datei `tTemp.mat` geschrieben und zwar im Matlab-Binärformat.

```
>>  save tTemp
```

Weitere Informationen zu den Befehlen erhält man wieder mit `help` *befehlsname*.

Die Datei `tTemp.dat` enthält Meßdaten. Neun Spalten und 403 Zeilen. In der ersten Spalte stehen Zeitwerte von 0 bis 804 Sekunden und die restlichen acht Spalten enthalten jeweils die dazugehöhrigen Temperaturwerte von acht Meßfühlern. Die Datei `messen.m` wird in Kapitel 5 besprochen.

Die bisher erstellten Variablen werden im Matlab Arbeitsspeicher (*Workspace*) gespeichert. Verändert man die Werte einer Datei, z.B. in einem Editor, und möchte mit den veränderten Werten weiterarbeiten, muß die Variable wieder neu in den Arbeitsspeicher geladen werden. Der Befehl

```
>>  whos
```

listet alle Variablen im aktuellen Arbeitsspeicher auf.

```
>>  clear
```

löscht alle Variablen im aktuellen Workspace.

Weitere Befehle:

`quit, exit`	beendet Matlab
`more on`	zum seitenweisen Lesen der Bildschirmausgabe (`more off`)
`which` *function*	zeigt den Pfad zu der Datei *function.m* an
`lookfor` *keyword*	gibt aus allen Matlab-Dateien die Sätze aus, die den Begriff *keyword* enthalten
`fprintf`	ähnlich, aber nicht genauso wie die entsprechenden ANSI C Funktionen zur formatierten Ausgabe (siehe auch: `fscanf, fopen`)

4 Vektoren und Matrizen

In Kapitel 2 wurde eine Matrix erstellt. Hier werden weitere Möglichkeiten angewendet, mit denen Vektoren erzeugt oder aus vorhandenen Vektoren und Matrizen Teile extrahiert und wieder neu zusammengesetzt werden können. Dabei wird die matrizenorientierte Arbeitsweise von Matlab deutlich. Vektor- und Matrixelemente werden über Indizes angesprochen. Das Element der i-ten Zeile und der j-ten Spalte einer Matrix $\mathbf{A}$ erhält man mit der Anweisung `A(i,j)`.

4.1 Matrixelemente

Die Dimension einer Matrix oder eines Vektors muß in Matlab nicht extra angegeben werden. Mit der folgenden Anweisung wird ein Vektor mit drei Elementen erzeugt.

```
>> x = [ 1/2 pi sqrt(4)-1 ]
```

So ist es z.B. möglich, den Vektor x wie folgt zu erweitern:

```
>> x(5) = x(1)
```

Über das vierte Element wurde hier keine Aussage gemacht. Deshalb bekommt es bei Matlab den Defaultwert Null. Die Anweisung

```
>> x(4) = []
```

weist dem vierten Element die leere Menge zu. Damit wird dieses Element eliminiert und die Dimension von x um eins runtergesetzt.

Vektoren und Matrizen können wie einzelne Elemente wiederum in Matrizen zusammengefaßt werden. Aus

```
>>  A = [1 2 ; 3 4]
>>  B = ones(2)
```

sollen neue Matrizen gebildet werden. Dabei können die Elemente nebeneinander

```
>>  C = [A B]
```

```
C =

     1     2     1     1
     3     4     1     1
```

oder untereinander angeordnet werden.

```
>>  D = [A; B]

D =

     1     2
     3     4
     1     1
     1     1
```

4.2 Intervalldarstellung

Man kann Vektoren auch bilden, indem man das Intervall und den Zuwachs (increment) angibt. Mit der folgenden Anweisung wird ein Vektor mit den Elementen $\{0, \frac{1}{3}, \frac{2}{3}, 1\}$ erzeugt, gemäß der Schreibweise: `Startwert:Schrittweite:Endwert`. Solange die Schrittweite 1 ist, kann die Schreibweise auch abgekürzt werden zu: `Startwert:Endwert`.

```
>>  y = 0:1/3:1
```

4.3 Teilmatrizen

Zu Anfang des Abschnitts wurde gesagt, daß einzelne Matrixelemente $\mathbf{A}(i, j)$ mit der üblichen Indexschreibweise angesprochen werden. Nun können i und j aber auch Vektoren sein.

```
>>  i = 1:2
>>  E = C(i,1)
```

Die Matrix $\mathbf{E}$ enthält dann die Elemente der ersten Spalte: $\mathbf{C}(i(1), 1), ..., \mathbf{C}(i(n), 1)$. Die Anweisung

```
>>  E = C(1:2,1)
```

ist ebenfalls möglich.

Mit Hilfe des *colon*-Operators (:) können alle Elemente einer Spalte oder Zeile angesprochen werden.

```
>>  E = C(:,1)
```

Weitere Befehle:

`linspace(x1,x2,n)`	erzeugt einen Vektor mit n Elementen in gleichen Abständen zwischen $x1$ und $x2$ (*linearly equally spaced*)
`[m,n]=size(x)`	berechnet die Dimensionen der mxn Matrix x
`length(x)`	berechnet die Anzahl der Elemente des Vektors x
`diag(x)`	erzeugt einen Vektor der Diagonalelemente der Matrix x

5 M-Files

Wie in den vorhergehenden Kapiteln gesehen, führt Matlab einzelne nach dem Prompt eingegebene Kommandos aus. In diesem Kapitel soll gezeigt werden, daß Matlab eine Folge von Befehlen, die in einer Datei gespeichert sind, abarbeiten kann. Solche Dateien werden M-Files genannt; die im Editor erstellte Datei erhält die Endung `.m`. Es gibt zwei verschiedene Arten von M-Files, die im folgenden vorgestellt werden: Script Files und Function Files.

5.1 Script Files

Script Files, auch Batch-Files genannt, führen lange oder häufig verwendete Befehlsfolgen aus, deren interaktive Eingabe mit der Zeit ermüdend wären. Dabei verwenden die im File enthaltenen Anweisungen globale Daten aus dem Arbeitsspeicher.

Bei der in Kapitel 3 ins Arbeitsverzeichnis kopierten Datei handelt es sich um ein Script File, dessen Inhalt ihr euch mit

```
>> type messen.m
```

auf dem Bildschirm anzeigen lassen könnt. Zur Ausführung wird lediglich

```
>> messen
```

eingegeben. Danach verbleiben die in der Datei verwendeten Variablen im Arbeitsspeicher.

5.2 Function Files

An Function Files, die mitunter auch als M-Functions bezeichnet werden, können Parameter übergeben werden. Variablen, die innerhalb des Function Files definiert und verändert werden, gehören lokal zu der Funktion und arbeiten nicht global im Arbeitsspeicher. Der Kopf eines Function Files besteht aus dem Schlüsselwort `function`, der Liste der Rückgabewerte, dem Namen der Funktion und der Input-Parameter.
Die Datei `mean.m` ist ein Beispiel für ein von Matlab zur Verfügung gestelltes Function File, dessen Quellcode ihr euch anschauen könnt. Dieser Funktion kann als Parameter z.B. ein Vektor oder eine Matrix übergeben werden; wird der zweite Parameter weggelassen, so berechnet ihn die Funktion selbst. In diesem Fall lautet der Aufruf lediglich

```
>> mean(z)
```

falls z vorher als Vektor oder Matrix beschrieben wurde. Der Aufruf eines vordefinierten oder selbstgeschriebenen Function Files kann natürlich auch in ein Script File eingebunden werden.
Abschließend noch einige Anmerkungen zu `mean.m`:

- Die erste Zeile legt den Funktionsnamen, sowie die Input- und Output-Parameter fest. Ohne diese Zeile wäre die Datei ein Script File und kein Function File.
- Das %-Symbol kennzeichnet einen Kommentar, der vom Interpreter ignoriert werden soll; dies gilt für alle Arten von M-Files.
- Die Kommentarzeilen bis zur ersten Leerzeile dokumentieren das M-File und erscheinen, wenn

  ```
  >> help mean
  ```

  eingegeben wird. Auf diese Weise können für die eigenen M-Files hilfreiche Online-Hilfe erstellt werden.

Übung: Verbessert das Script File `messen.m`.

a. Fügt eine Online-Hilfe der Datei ein, die mit `help messen` angezeigt wird.
b. Berechnet die Anzahl der Temperatursensoren aus den Matrizengrößen. (siehe Kap. 4)
c. Erzeugt einen Indexvektor s und verwendet ihn bei der Berechnung der Matrix `temp`.

6 Programmieren in Matlab

6.1 Kontrollstrukturen

Bisher wurde gezeigt, wie Matlab sequentiell, d.h. in unverzweigten Strukturen, Anweisungen abarbeitet. Sollen jedoch kompliziertere Sachverhalte programmiert werden, sind verzweigte Strukturen unerläßlich. Diese werden durch die beiden Grundbausteine *Alternation* für die Auswahl zwischen zwei oder mehreren unterschiedlichen Operationen und *Iteration* für die Wiederholung der gleichen Operation in einer Schleife mit Abbruchbedingung realisiert.
Der vorliegende Abschnitt zeigt, wie diese sogenannten Kontrollstrukturen in Matlab implementiert werden.

6.1.1 Alternation

Die Alternation wird durch folgende klassische Konstruktion umgesetzt:

`if` *Bedingung 1*,
Anweisungsfolge 1
`elseif` *Bedingung 2*,
Anweisungsfolge 2
`else` *Bedingung 3*,
Anweisungsfolge 3
`end`

Die `if`-Anweisung kann in jeglicher Form verwendet werden, d.h. als einfaches `if`, als `if`, `else` oder als `if`, `elseif`, `else` und wird stets mit `end` abgeschlossen.

6.1.2 Iteration

Die Iteration kann als Schleife mit Eintrittsbedingung

`while` *Bedingung*
Anweisungen
`end`

oder als Zählschleife

```
for i = 1:n,
Anweisungen
end
```

ausgeführt werden.

Hinweis: Schlüsselwörter wie `while`, `for` und `end` werden stets kleingeschrieben, auch wenn sie in den `help`-Informationen bisweilen in Großbuchstaben auftauchen.

In Kapitel 4 wurden Vektoren durch die Angabe des Intervalls und der Schrittweite erzeugt. Damit entspricht der Zähler der obigen `for`-Schleife gerade einem Vektor, der als Elemente die Zahlen von `1` bis `n` enthält, wobei mit Schrittweite 1 hochgezählt wird. Auf diese Weise können ganze Schleifen vektoriell abgearbeitet werden. Das folgende Beispiel soll die Vorgehensweise verdeutlichen:
Die gewöhnliche `for`-Schleife

```
x = 0;
for i = 0:0.1:2*pi,
    x = x+1;
    y(x) = sin(i);
end
```

lautet als *vektorisierte for-Schleife*

```
i = 0:0.1:2*pi;
y = sin(i);
```

Der Vorteil der letzteren Variante besteht darin, daß die Matlab-eigenen Vektor- und Matrixoperationen um ein Vielfaches schneller sind als die Compiler/Interpreter-Operationen. Um die höchste Geschwindigkeit bei Matlab-Anwendungen zu erzielen, sollten daher die Algorithmen in den M-files nach Möglichkeit in vektorisierter Form vorliegen.

Um in einer strukturierten Programmierung die logischen Bedingungen zu formulieren, unter denen bestimmte Rechenoperationen ausgeführt werden sollen, ist es mitunter hilfreich, auf in Matlab vordefinierte Variablen zurückzugreifen. Einige dieser Variablen sind in der folgenden Tabelle aufgeführt:

`eps`	Floating point relative accuracy: relative Fließkomma-Genauigkeit; gibt den aktuellen Abstand von 1.0 zur nächstgrößeren Fließkommazahl zurück
`flops`	Floating point operation count: Anzahl der Fließkomma-Operationen; gibt die kumulierte Anzahl an Fließkomma-Operationen zurück
`Inf`	Infinity: unendlich; entsteht z.B. bei der Division durch Null
`NaN`	Not-a-Number: keine Zahl; entsteht z.B. bei der Division von Null durch Null
`cputime`	CPU time: CPU-Zeit; gibt die CPU-Zeit in Sekunden zurück, die vom Matlab-Prozeß seit dem Start verbraucht wurde
`nargin`	Number of function input arguments: Anzahl der Funktionseingangsargumente eines Function Files
`nargout`	Number of function output arguments: Anzahl der Funktionsausgangsargumente eines Function Files

6.2 Logische Operationen

In Matlab gibt es die Möglichkeit logische Operatoren auf Skalare und auf Matrizen anzuwenden. Als Rückgabewert erhält man bei den meisten Operatoren 1 oder 0, wobei 0 für FALSE und 1 für TRUE steht.
Die wichtigsten vergleichenden Operatoren sind:

$<$	kleiner als
$<=$	kleiner oder gleich
$>$	größer als
$>=$	größer oder gleich
$==$	gleich
$\tilde{}=$	ungleich

außerdem gibt es noch die logischen Operatoren:

&	und
|	oder
`xor`	ausschließendes oder
~	nicht

Beispiele:

Es sollen die Elemente einer Matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

gefunden werden, die größer als 3 sind:

```
>>  P = A>3
```

ergibt als Lösung:

```
P =

     0     0     0     1
     1     1     1     1
```

und noch ein Beispiel; mit den logischen Operatoren & und | soll bei zwei Matrizen

$$\mathbf{A} = \begin{pmatrix} 0 & 3 & 0 \\ 1 & 1 & 2 \end{pmatrix}$$

$$\mathbf{B} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

verglichen werden, ob die gleichen Elementplätze belegt sind:

```
>>  C = A&B
```

```
C =

     0     0     0
     0     1     1


>> C = A|B

C =

     0     1     1
     1     1     1
```

Außer diesen Standartoperatoren gibt es noch weitere interessante logische Operatoren:

`any`	true wenn mindestens ein Element ungleich Null ist
`all`	true wenn **alle** Elemente ungleich Null sind
`find`	sucht nach den Indizes der Elemente, die einer bestimmten Bedingung genügen
`exist`	prüft die Existenz einer Variablen
`isnan`	prüft auf Elemente, die NaN sind
`finite`	sucht nach endlichen (finiten) Elementen
`isinf`	sucht nach unendlichen (infiniten) Elementen
`isempty`	true wenn die Matrix leer ist
`isstr`	true wenn eine string-Variable vorliegt

Beispiele:

Die folgenden Operationen werden, falls nicht anders definiert auf die Matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 3 & 0 \\ 0 & 5 & 1 \end{pmatrix}$$

angewendet.

```
>>  P = any(A)

P =

     0     1     1


>>  P = all(A)

P =

     0     1      0
```

Man sieht, daß die Operatoren `any` und `all` spaltenweise arbeiten.

Bei dem Operator `find` erhält man als Rückgabewert zwei Vektoren mit den Indizes der Elemente, die das gegebene Kriterium erfüllen.

```
>>  [i,j] = find(A>2)

i  =

      1     2

j  =

      2     2
```

Also sind die Matrixelemente a_{12} und a_{22} größer als 2.

Der Rückgabewert des Operators `exist` kann unterschiedliche Werte annehmen (siehe `help exist`).

Die Überprüfung unerlaubter Rechenoperationen geschieht mit `isnan` und `infinite`.

```
>>  B = A/0
```

```
>>  P = isnan(B)

P =

     1     0     1
     1     0     0

>>  P = infinite(B)

P =

     0     1     0
     0     1     1
```

6.3 Fehlersuche in Matlab - Debugging

Wie in jeder Programmiersprache können sich auch beim Programmieren in Matlab Fehler einschleichen. Um diese zu finden, bietet sich die Arbeit mit einem Debugger an. Ein Debugger ermöglicht es, ein Programm zeilenweise abzuarbeiten und sich z.B. Werte von Variablen ausgeben zu lassen. Allerdings steht nur bei Function Files der komplette Debugger-Befehlssatz zur Verfügung. Deshalb betrachten wir zur Fehlersuche in Matlab die folgenden Bereiche voneinander getrennt:

- Interaktives Programmieren
- Programmieren von Script Files
- Programmieren von Function Files

6.3.1 Interaktives Programmieren

Bei dieser Programmierart ist es am leichtesten, Fehler zu finden. Man kann sich Zwischenergebnisse auf dem Bildschirm ausgeben lassen (wenn das Semikolon am Ende einer Anweisung wegelassen wird), um so die Rechnung nachzuvollziehen und eventuelle Fehler zu entdecken. Da Matlab das Kommando und die Zeile, in dem der Fehler auftrat, benennt,

ist es oft hilfreich, die Online-Hilfe zu benutzen. Hier kann man z.B. erfahren, welche Parameter eine Funktion benötigt.
Darum solte man auch bei einer selbst geschriebenen Funktion unbedingt an eine Online-Hilfe denken!

6.3.2 Script Files

Ein Script File ist letztlich nichts anders als eine Folge von Befehlen, die in einer Datei gespeichert sind. Beim Aufruf der Datei werden die Befehle dann nacheinander ausgeführt. Das Fehlersuchen gleicht also dem des interaktiven Programmierens. Allerdings muß hier gewartet werden bis auch die letzte Zeile ausgeführt worden ist - erst dann kann man einen Fehler korrigieren. Das laufende Script File kann aber auch unterbrochen werden um die im globalen Arbeitsspeicher bekannten Variablen auszugeben. Danach kann das Programm wieder fortgesetzt werden. Dazu muß folgendes getan werden:

- In das Script File den Befehl `keyboard` an geeigneter Stelle aufnehmen.
- Bei der Ausführung des Programms unterbricht es an dieser Stelle und es erscheint ein Prompt: `K>>`
- Jetzt können z.B. Variablen zur Überprüfung ausgegeben werden.
- Um das Programm wieder zu starten, einfach das Wort `return` eintippen.

6.3.3 Function Files

Fortgeschrittenes Matlab Programmieren bedeutet meistens Function Files zu schreiben. Da Function Files lokale Variablen verwenden, die nicht im globalen Arbreitsspeicher gespeichert sind, ist es hierbei schwierig Laufzeitfehler zu finden. Deshalb verwendet man erst beim Programmieren von Function Files einen Debugger im herkömmlichen Sinn. Um eine Funktion zu debuggen, muß zuerst ein Breakpoint gesetzt werden, damit die Funktion nach dem Aufruf auch unterbricht. Befindet man sich im Debugging-Mode, sieht man folgenden Prompt:
`K>>`.
Dort können spezielle Debug- Kommandos eingegeben und z.B. Variablen geändert werden. Es folgt eine Tabelle der Befehle und dazu jeweils eine kurze Erläuterung.

`dbstop in test`	Setzen eines Breakpoints an den Anfang der Funktion `test.m`
`dbstop in test at 10`	Setzen eines Breakpoints in die 10'te Zeile
`dbtype`	listet den Quelltext mit Zeilennummern auf
`dbcont`	die Funktion läuft weiter
`dbstep`	Ausführen einer Zeile
`dbstep in`	Sprung in eine Unterfunktion, die in der Zeile aufgerufen wird
`who`	Anzeige der Variablen im Arbeitsspeicher
`dbup`	zeigt die Variablen des vorherigen Arbeitsspeichers
`dbdown`	Rückkehr zum aktuellen Arbeitsspeicher
`dbstack`	zeigt die aktuelle Position und den Weg dorthin
`dbstatus` *FunktionsName*	listet die Breakpoints auf
`dbclear all in` *FunktionsName*	löscht die Breakpoints
`dbclear all`	löscht alle Breakpoints
`dbquit`	Verlassen des Debugging-Modes

7 Graphik

Hier geht es um Graphikausgaben. Wir haben versucht, die wichtigsten Befehle zusammenzutragen. Diese Liste hat nicht den Anspruch *alles* darzustellen, deshalb raten wir euch auch die Matlab-Hilfe zu verwenden.
Hinweis: Bei den Befehlen werden unterschiedliche Darstellungen, abhängig von welche und wieviele Argumenten eingegebenen werden, erzielt. Deshalb sind sie hier zur verallgemeinerung meistens weggelassen, aber wenn sie vorhanden sind, sind bei den Beispielen Skalare und Vektoren klein, und Matrizen groß geschrieben.

7.1 Erstellen von Plots

Zweidimensionale Darstellung

`plot(y)`	trägt die Elemente des Vektors y über den jeweiligen Index von y auf
`plot(x,y)`	trägt Vektor y über Vektor x auf
`plot(t,y1,t,y2,t,y3)`	drei Graphen werden mit einem Aufruf gezeichnet
`plotyy(x,y1,x,y2,'function')`	ein Graph mit zwei y-Achsen; Beispiel für eine Funktion ist `loglog`
`loglog( )`	doppeltlogarithmische Auftragung der Argumente (Argumente wie beim Befehl `plot`)
`fplot('functionfile', [xmin xmax ymin ymax])`	plottet den Graph eines Function Files

Beispiel

```
>>  t = 0:pi/100:2*pi;       % Vektor t definieren
>>  y = sin(t);              % Vektor y erzeugen
>>  plot(t,y);
```

Dreidimensionale Darstellung

`plot3( )`	plottet Linien und Punkte in 3-D (Argumente wie beim Befehl `plot`)
`[X,Y]=meshgrid(x,y)`	generiert zwei Matrizen die für die 3D-Darstellung erforderlich sind
`mesh()`	legt ein Gitter über den Plot
`surf()`	füllt die Oberfläche des Plots aus (zusätzlich 3D)
`quiver(x,y)`	zeichnet Pfeile mit den Komponenten x,y
`contour(x)`	erzeugt die Höhenlinien zu x

Hinweis: Der Befehl `help graph3d` zeigt euch mehrere interessante Befehle auf einem Blick.

Beispiel (für quiver)

```
>>  x = [0:.2:1];            % Vektor x definieren
>>  y=x.^2;                  % Vektor y erzeugen
```

```
>>  u=gradient(x);          % bildet den Gradienten von x
>>  v=gradient(y);
>>  s=.5;                   % Normierung der Pfeile
>>  quiver(x,y,u,v,s);
>>  hold on                 % Die folgenden Befehle werden in den
                            % aktuellen Graph hinzugeplottet

>>  y = .1+x.^2;            % Erweiterung
>>  v=gradient(y);          % bildet den Gradienten von y
>>  s=.2;
>>  quiver(x,y,u,v,s);
>>  hold off;
```

7.2 Einstellungsmöglichkeiten

Achseneinstellungen

`axis([xmin xmax ymin ymax])`	Skalierung der Achsen
`grid on/off`	Gitterlinien werden ein bzw. ausgeschaltet
`hold on/off`	Graphikfenster bleibt für weitere Plots

Graphenbeschriftung

`xlabel('Name')`	x-Achsenbeschriftung
`ylabel('Name')`	y-Achsenbeschriftung
`title('Titel')`	Titel
`text(x,y,'Text')`	beliebiger Text wird an die Stelle x,y im Graph geschrieben
`gtext('Text')`	beliebiger Text wird per Maus an gewünschte Stelle gesetzt
`legend('Text1','Text2',...)`	Legende für alle geplotteten Graphen

Beispiel

```
>>  t = 0:pi/100:2*pi;
>>  y = sin(t);
```

```
>> plot(t,y);
>> xlabel('0 \leq t \leq \pi')        % 0<= t <=pi
>> ylabel('y')
>> title('Sinuskurve')
>> text(4,0.2,'Matlab macht Spass!')
>> legend('sin(t)')
```

Weitere Einstellungen

`shading interp`	interpoliert die diskreten Werte
`colorbar`	zeichnet eine Farbskala
`colormap()`	Auswahl eines Farbschemas(siehe helpdesk colormap)
`pcolor(Z)`	farbiges plot wo die Werte der Matrix die Farbe bestimmen
`view(`α, β`)`	setzt den Blickpunkt auf die Graphik fest
`pause(5)`	das momentane Bild wird 5 Sekunden gehalten, bevor das Programm weiterarbeitet
`pause`	das Programm erwartet eine enter-Eingabe

7.3 Abspeichern und Ausdrucken

Nun kann man die Graphik natürlich auch noch ausdrucken oder auf eine Datei schreiben. Alle Optionen finden sich wie immer in der Hilfe.

Die allgemeine Form ist: `print -`*devicetype* `-`*option filename*

```
>> print -dps2 filename              % Postscript(s/w)
>> print -dpsc2 filename             % farbiges Postscript
>> print -deps2 filename             % encapsulated Postscript
>> print -depsc2 filename            % farbiges EPS
>> print -Pdruckername               % ausdrucken
```

Dies alles geht auch im Graphikfenster. Im Graphikfenster wählt man im Menü File die Option Print aus. Es poppt ein neues Fenster auf. Dort wählt man dann entweder Print (ausdrucken) oder Save (schreiben auf Datei) aus. Auch hier müssen die obenstehenden Optionen eingetragen werden.

8 Hilfe

Wie schon mehrmals erwähnt, kann man mit der Eingabe

```
>> help begriff
```

viele intressante Hinweise zu dem gesuchten Begriff bekommen.
Mit

```
>>  helpdesk
```

wird ein Browser (Netscape) mit einer komfortablen Hilfe gestartet.[4] Dort gibt es u.a. die Möglichkeit Informationen über Matlab-Funktionen nach Thema oder nach Index zu suchen. Über den Link `Solution Search` wird auch Hilfe zu dem allgemeinen Problem `How To Find Answers` angeboten.

Außerdem gibt es noch die WWW-Seite des Europäischen Matlab-Servers,
`http://www-europe.mathworks.com`
auf dem auch eine Support-Seite mit Fragendatenbank angeboten wird, in der nach Begriffen gesucht werden kann.

Auf der Homepage der PPM gibt es den Primer (eine 35-seitige Kurzeinführung in Matlab).

Weitere Befehle:

`demo`	Öffnet ein Demo-Fenster mit vielen Beispielen zum anschauen
`doc`	wie `helpdesk`

Übung: Man gebe bei `Go to MATLAB function:` „`inv`“ ein und suche in den angebotenen Informationen nach Hinweisen darauf, ob es sinnvoll ist, ein Lineares Gleichungssystem mit der Berechnung der Inversen oder doch lieber mit dem Gauß-Algorithmus zu lösen.

Noch Fragen ?: NEWSGROUP: comp.soft-sys.matlab

[4] Die angezeigten HTML-Dateien liegen local auf einem Server und müssen also nicht von weither übertragen werden.

Literatur

[1] Cavallo, Alberto, Roberto Setola und Francesco Vasca: *Using MATLAB, SIMULINK and Control System Toolbox. A practical approach.* The MATLAB Curriculum Series. Prentice Hall, London, New York, Toronto, Sydney, Tokyo, Singapore, Madrid, Mexico City, Munich, 1996.

[2] Sigmon, Kermit: *MATLAB Primer.* CRC Press, Boca Raton, Ann Arbor, London, Tokyo, vierte Auflage, 1994.

[3] The MathWorks, Inc. (Herausgeber): *MATLAB High-Performance Numeric Computation and Visualization Software. User's Guide for UNIX Workstations.* The MathWorks, Inc., 24 Prime Park Way, Natick, Mass., vierte Auflage, 1996.