Ein Schnellkurs in Matlab*

Oliver Stein
Lehrstuhl C für Mathematik
RWTH Aachen
E-Mail: stein@mathC.rwth-aachen.de

Überarbeitete Version, Oktober 2001

1 Einführung

Diese Kurzeinführung in grundlegende Elemente von MATLAB basiert im wesentlichen auf Kapitel 2 des sehr empfehlenswerten Buches [1] von Günter Gramlich und Wilhelm Werner. Der Originaltext dieses Kapitels ist im Internet unter `www.mathematik.uni-kl.de/~gramlich/matlab_kap2.pdf` zur Zeit frei erhältlich. Dort findet man weitergehende Erklärungen zu Struktur und Befehlen in MATLAB, die wir im hier vorliegenden Text unterschlagen wollen. Vielmehr soll dieser Schnellkurs den Leser in kurzer Zeit in die Lage versetzen, einfache Berechnungen, Programme und Visualisierungen selber zu erstellen. Die vielfältigen und weitreichenden Fähigkeiten von MATLAB kann er von diesem Grundstock aus dann problemlos selber erkunden.

MATLAB ist ein Matrix-orientiertes Softwaresystem – der Name MATLAB steht für *Matrix Laboratory*. Es ist aus den Mitte der 70er Jahre entwickelten Softwarepaketen LINPACK und EISPACK zur Lösung linearer Gleichungssysteme bzw. algebraischer Eigenwertprobleme hervorgegangen. Seitdem hat es sich durch Beiträge vieler Benutzer zu einem Umfang entwickelt, der es nicht nur an Hochschulen zu einem bevorzugten Werkzeug in Forschung und Lehre werden ließ, sondern durch den es auch in der Industrie vermehrt für Forschung, Entwicklung, Datenauswertung und Visualisierungen aller Art Einsatz findet.

MATLAB lässt sich in drei Hauptkomponenten untergliedern: Berechnung, Programmierung und Visualisierung. Für Berechnungen verfügt MATLAB über

*Unter www.mathC.rwth-aachen.de/~stein/lit/matlab ist dieser Text frei erhältlich. Disclaimer: Für die Richtigkeit der Informationen in diesem Text wird keine Gewähr übernommen. Insbesondere kann keine Haftung für mittelbare oder unmittelbar verursachte Schäden übernommen werden.

eine qualitativ hochwertige Programmsammlung, die es dem Benutzer erspart, Standardalgorithmen selber zu implementieren. Aufbauend auf den vorliegenden Programmen kann der Benutzer auf einfache Weise selber neue Programme schreiben und damit die Funktionalität von MATLAB erweitern. Nicht zum Standardumfang von MATLAB gehören die sogenannten *Toolboxen*, die weitere Programme für Spezialgebiete beinhalten, etwa die *Optimization Toolbox*, die *Partial Differential Equation Toolbox* oder die *Statistics Toolbox*. Für die Visualisierung von Daten stellt MATLAB moderne und leicht zu handhabende Algorithmen zur Verfügung.

Auf viele Fähigkeiten von MATLAB werden wir im folgenden nicht näher eingehen: Etwa bietet MATLAB mit der *(Extended) Symbolic Math Toolbox* die Möglichkeit zum symbolischen Rechnen, durch das *Application Program Interface (API)* kann man C- oder FORTRAN-Programme einbinden, und hinter *Handle Graphics* verbirgt sich eine Grafik-Hochsprache zur Bildverarbeitung und Erstellung von Präsentationen. Schließlich lassen sich mit SIMULINK, einem Partnerprogramm von MATLAB, diskrete und kontinuierliche dynamische Systeme blockorientiert modellieren und simulieren.

2 Arbeiten mit Matlab

Wir werden nun zunächst die grundlegenden Prinzipien des interaktiven MATLAB-Modus vorstellen und danach sehen, wie diese auf sehr einfache Weise zur Programmierung in MATLAB führen. Das Kapitel endet mit einer kurzen Darstellung wichtiger Visualisierungsinstrumente.

2.1 Starten und Beenden

Je nachdem, ob man unter einem kommando-orientierten oder grafischen Betriebssystem arbeitet, startet man MATLAB durch Eingabe des Befehls `matlab` oder durch Anklicken des MATLAB-Symbols. Im Kommandofenster erscheint dann die folgende Begrüßung:

```
                    < 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.

>>
```

In der Studentenversion ist der MATLAB-Prompt `>>`, an dem die Eingabe erwartet wird, durch `EDU>>` ersetzt. Man beendet MATLAB durch Eingabe von `exit`.

2.2 Der interaktive Modus

Nach dem Starten von MATLAB kann man am Prompt `>>` mit dem MATLAB-Interpreter kommunizieren. Für eine einfache Rechnung geben Sie zum Beispiel

```
>> 2 + 1.5
```

ein und drücken die Enter-Taste. MATLAB antwortet

```
ans =
    3.5000
```

wobei `ans` eine Hilfsvariable bezeichnet und für *answer* steht. Durch

```
>> x = 5
```

weist man der Variable `x` den Wert 5 zu, und MATLAB antwortet

```
x =
     5
```

Möchte man diese Antwort unterdrücken, so schließt man die Eingabe mit einem Semikolon ab: auf

```
>> y = 3 + x;
```

antwortet MATLAB zwar nicht, aber Eingabe von

```
>> y
```

liefert die Antwort

```
y =
     8
```

Mit dem Kommando `who` verschafft man sich einen Überblick über die im sogenannten *Workspace* angelegten Variablen:

```
>> who

Your variables are:

ans       x         y
```

Mit `whos` erhält man umfangreichere Informationen über die Variablen, und gelöscht werden sie mit `clear`. Nach dem Verlassen von MATLAB steht der Workspace nicht mehr zur Verfügung. Daher kann man mit `save` den Workspace oder einzelne Variablen sichern:

```
>> save test1
```

legt eine Datei `test1.mat` im aktuellen Verzeichnis an, in der alle Variablen des Workspace gespeichert sind. In einer späteren MATLAB-Sitzung oder nach der Eingabe von `clear` werden diese Variablen durch

```
>> load test1
```

wieder in den Workspace geladen.

```
>> save test2 x y
```

speichert nur die Variablen `x` und `y` in `test2.mat`.

Der Befehl

```
>> dir

.          ..         test1.mat test2.mat
```

zeigt den Inhalt des aktuellen Verzeichnisses an,

```
>> delete test1.mat
```

löscht die Datei `test1.mat`,

```
>> mkdir tests
>> cd tests
```

legt ein neues Unterverzeichnis `tests` an und wechselt dorthin. Benötigt man andere Befehle des zugrundeliegenden Betriebssystems, so öffnet `!` eine *Shell*. Beispielsweise öffnet unter Linux der Befehl

```
>> !emacs &
```

den EMACS-Texteditor und gibt danach den MATLAB-Prompt wieder frei.

Wird eine Eingabezeile zu lang, so schließt man die erste Zeile mit drei Punkten ab und schreibt nach Drücken der Enter-Taste in der nächsten Zeile weiter:

```
>> h = 1 + 1/2 + 1/3 + 1/4 + 1/5 ...
       + 1/6 + 1/7;
```

Ein laufendes Programm oder eine Berechnung lassen sich jederzeit durch Eingabe von `Ctrl-C` abbrechen, ohne dass MATLAB beendet werden muss. Alte Eingaben kann man mit der Cursortaste ↑ zurückholen und editieren. Das Kommando `diary` legt eine Datei an, in der sämtliche Bildschirmausgaben der laufenden Sitzung gespeichert werden - man beendet die Aufzeichnung mit `diary off`.

Zu jedem MATLAB-Befehl gibt es einen Hilfetext, der durch `help` aufgerufen wird.

```
>> help save
```

erläutert zum Beispiel Funktionsweise und Optionen des `save`-Befehls.

Wenn man einen Befehl sucht, ohne seinen Namen zu kennen, bietet `lookfor` die Möglichkeit einer Stichwortsuche. Dieser Befehl durchforstet die jeweils erste Zeile aller MATLAB-Hilfetexte und gibt passende Befehle aus. `helpwin` öffnet ein Hilfefenster, `helpdesk` verbindet per Internet zu einer Hilfe-Seite. Schließlich bietet MATLAB unter `demo` eine Kurzeinführung an.

2.3 Grundlegende mathematische Elemente

Operationen, elementare Funktionen, Konstanten. Die üblichen arithmetischen Operationen Addition, Subtraktion, Multiplikation, Division und Potenzieren besitzen die MATLAB-Syntax `a+b, a-b, a*b, a/b` bzw. `a^b`. Als elementare Funktionen stehen die trigonometrischen und hyperbolischen Funktionen sowie ihre Inversen (`cos, sinh, atan` etc.), die Exponentialfunktion `exp`, die Logarithmusfunktion `log` und die Wurzelfunktion `sqrt` zur Verfügung. Eine komplette Liste liefert `help elfun`. Ferner kann man auf wichtige Konstanten wie `pi` und die komplexe Einheit `i` zugreifen:

```
>> exp(i*pi)

ans =

  -1.0000 + 0.0000i
```

Zu beachten ist hierbei, dass `i` ohne Warnung überschrieben wird, wenn man `i` zum Beispiel als Laufindex benutzt. Rechnungen mit komplexen Zahlen können danach falsche Ergebnisse liefern. Weitere wichtige Konstanten sind die Maschinengenauigkeit `eps`, die „unendlich große Zahl“ `Inf` und der Ausdruck „Not-a-Number“ `NaN`.

Variablen. Zur Definition von Variablen sind keine Typerklärungen oder Dimensionsanweisungen erforderlich. Beim Auftreten eines neuen Variablennamens richtet MATLAB automatisch die entsprechende Variable ein und weist ihr den erforderlichen Speicherplatz zu. Variablennamen bestehen aus einem Buchstaben mit nachfolgend beliebig vielen weiteren Buchstaben, Ziffern oder Unterstrichen (keine Leerzeichen!). MATLAB wertet aber nur die ersten 31 Zeichen eines Namens aus und unterscheidet dabei zwischen Groß- und Kleinschreibung.

Zahlen. MATLAB verarbeitet Zahlen in der üblichen Dezimalschreibweise mit Dezimalpunkt. In wissenschaftlicher Notation bezeichnet der Buchstabe `e` eine Skalierung um Zehnerpotenzen. Zulässige Zahlen sind zum Beispiel

```
4   0.0001   1.5e-12   3i
```

Alle Zahlen werden intern im `double precision`-Format (gemäß der Spezifikation durch die Gleitpunktnorm der IEEE) abgespeichert. Die Zahlenausgabe unterliegt folgenden Regeln: Eine ganze Zahl wird als ganze Zahl ausgegeben. Eine reelle Zahl wird auf vier Dezimalen gerundet ausgegeben. Ist eine Zahl größer als 10^3 oder kleiner als 10^{-3}, so wird sie in exponentieller Form dargestellt. Zieht man die Ausgabe in anderen Formaten vor, so hilft das Kommando `format`.

2.4 Matrizen

In MATLAB existieren mehrere Datentypen, die durch ein mehrdimensionales Feld dargestellt werden. Im folgenden werden wir uns auf die Diskussion von Matrizen mit numerischen oder mit Text-Einträgen beschränken.

Eingabe von Vektoren. Zeilenvektoren werden durch eckige Klammern definiert, wobei die Vektorkomponenten wahlweise durch Leerzeichen oder durch Kommata getrennt werden:

```
>> a = [1 2 3]

a =

     1     2     3

>> b = [sqrt(2),cos(pi)]

b =

    1.4142    -1.0000
```

Bei Spaltenvektoren trennt man die Komponenten durch Semikolons oder gibt sie als Zeilenvektor mit nachfolgendem Transponiertzeichen `'` ein:

```
>> c = [-1;2]

c =

    -1
     2
```

```
>> d = [0.1 3.1]'

d =

    0.1000
    3.1000
```

Zeilenvektoren mit Schrittweite 1 oder einer allgemeineren Schrittweite können ebenfalls einfach erzeugt werden:

```
>> x = 3:8

x =

     3     4     5     6     7     8

>> y = 3:0.5:4.8

y =

    3.0000    3.5000    4.0000    4.5000
```

Mit `linspace` ist es möglich, Vektoren zu erzeugen, indem man den Wert der ersten, letzten, sowie die Anzahl der Komponenten vorgibt:

```
>> z = linspace(3,4.8,4)

z =

    3.0000    3.6000    4.2000    4.8000
```

Auf Vektorkomponenten greift man in der üblichen mathematischen Schreibweise zu:

```
>> z(3)

ans =

    4.2000
```

Eingabe von Matrizen. Matrizen gibt man ähnlich wie Vektoren ein: in eckigen Klammern werden zeilenweise die Elemente aufgeführt und dabei jede Zeile mit einem Semikolon abgeschlossen:

```
>> A = [1 2; 3 4]

A =

     1     2
     3     4
```

Einheitsmatrizen, Einsermatrizen und Null-Matrizen liefern folgende Kommandos:

```
>> eye(3)

ans =

     1     0     0
     0     1     0
     0     0     1

>> ones(2,3)

ans =

     1     1     1
     1     1     1

>> zeros(3,1)

ans =

     0
     0
     0
```

Die folgenden Operationen zeigen, wie man auf Elemente, Zeilen, Spalten und Untermatrizen einer Matrix zugreifen kann.

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

A =

     1     2
     3     4
     5     6

>> A(3,1)

ans =

     5

>> A(2,:)

ans =

     3     4

>> A(:,2)'

ans =

     2     4     6

>> A([1 3],:)

ans =

     1     2
     5     6
```

Die Eingabe sehr großer Matrizen kann im interaktiven Modus aufwendig werden. Bequemer ist es, hierfür ein MATLAB-Script-File anzulegen (siehe Abschnitt 2.6). Hilfreich ist hier auch der Befehl `sparse`, mit dem MATLAB dünnbesetzte Matrizen speichersparend verwaltet.

2.5 Matrixoperationen

Addition, Subtraktion, Multiplikation und Potenzieren besitzen als Matrixoperationen dieselbe Syntax wie für Skalare (siehe Abschnitt 2.3). Zusätzlich existieren eine rechte und eine linke Division: `A/B` löst $XA = B$ nach X, während `A\B` das System $AX = B$ nach X löst. Ein lineares Gleichungssystem $Ax = b$ löst MATLAB also durch den Befehl

```
>> x = A\b;
```

Während Addition und Subtraktion für Matrizen elementweise ausgeführt werden, sind Multiplikation und Potenzieren echte Matrixoperationen. Auch sie können jedoch elementweise ausgeführt werden, und zwar mit der Syntax `A.*B` bzw. `A.^p`. Um zum Beispiel die Funktion x^3 auf Gitterpunkten des Intervalls $[-1, 1]$ auszuwerten, gibt man folgendes ein:

```
>> [-1:0.1:1].^3
```

Addition und Subtraktion zwischen einer Matrix und einem Skalar sind ebenfalls definiert, und zwar elementweise:

```
>> [1 2 3] - 2

ans =

    -1     0     1
```

MATLAB stellt eine große Zahl von Standard-Matrixoperationen zur Verfügung, zum Beispiel Determinante, Inverse oder Bestimmung von Eigenwerten und Eigenvektoren:

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

A =

     1     2     3
     2     4     5
     3     5     6
```

```
>> B = inv(A)

B =

    1.0000   -3.0000    2.0000
   -3.0000    3.0000   -1.0000
    2.0000   -1.0000   -0.0000

>> det(A)*det(B)

ans =

    1.0000

>> eig(A)

ans =

   -0.5157
    0.1709
   11.3448

>> [V,e] = eig(B)

V =

   -0.5910    0.7370    0.3280
    0.7370    0.3280    0.5910
   -0.3280   -0.5910    0.7370

e =

    5.8509         0         0
         0   -1.9390         0
         0         0    0.0881
```

In der letzten Berechnung bilden die Eigenvektoren von B die Spalten der Matrix V, während auf der Diagonale von e die zugehörigen Eigenwerte stehen.

2.6 Scripts und Funktionen

Bislang haben wir den interaktiven Modus von MATLAB benutzt. Diese Arbeitsweise ist unzweckmäßig für Algorithmen, die mehrere Programmzeilen benötigen oder mehrfach verwendet werden sollen. Hierfür eignen sich sogenannte *m-Files*, die mit einem Editor erzeugt und unter einem Filenamen mit dem Anhang `.m` abgespeichert werden. Es gibt zwei Arten von m-Files: *Script-Files* und *Function-Files*. Bevor man ein neues m-File `filename.m` anlegt, empfiehlt es sich zu überprüfen, ob in MATLAB schon ein File dieses Namens existiert, z.B. mit `help filename`.

Script-Files. Ein Script-File ist eine Folge von gewöhnlichen MATLAB-Anweisungen. Die Anweisungen des Script-Files `dateiname.m` werden durch

```
>> dateiname
```

ausgeführt. Variablen in einem Script-File sind global. Zum Beispiel kann man eine große Matrix dadurch eingeben, dass man in das Script-File `data.m` die Anweisungen

```
A = [ 2.5465  3.4688 -5.2345;
      3.0113 -9.8894  0.1111;

           ⋮

      8.4994  0.4943 -1.9553];
```

schreibt und am MATLAB-Prompt `data` eingibt. Die Variable A steht dann im Workspace zur Verfügung. Script-Files dürfen ihrerseits m-Files aufrufen.

Kommentare in m-Files beginnen mit `%`, genauer: der Text in einer Zeile nach `%` wird als Kommentar aufgefasst. Für die Ein- und Ausgabe von Informationen kann man etwa die Befehle `input` und `disp` benutzen, wie in folgendem Script-File:

```
x = input('Eingabe von x: '); % Input
y = x^3;                      % Berechnung
disp(['x^3 = ',num2str(y)])   % Output
```

Texte, sogenannte *Strings*, werden in MATLAB in Apostrophen eingeschlossen. Man kann Textteile als Elemente von String-wertigen Vektoren auffassen und damit längere Textzeilen zusammenstellen. Im obigen `disp`-Befehl werden zum Beispiel der String `'x^3 = '` und der per `num2str` in einen String umgewandelte Wert von y zu einem einzigen String zusammengefasst.

Function-Files. Damit ein m-File ein Function-File ist, muss es mit dem Schlüsselwort `function` beginnen. An solch ein File kann man Eingabeargumente `In_1,...,In_m` übergeben, aus denen Ausgabeargumente `Out_1,...,Out_n` berechnet werden. Die allgemeine Form eines Function-Files lautet

```
function [Out_1,...,Out_n] = funktionsname(In_1,...,In_m)
% Hilfetext
   Irgendwelche Anweisungen
```

Der Hilfetext wird durch Eingabe von `help funktionsname` ausgegeben. Wichtig ist, dass dieses m-File unter dem Namen `funktionsname.m` abgespeichert werden muss.

Variablen in Function-Files sind lokal, d.h. sie werden nicht gemeinsam mit dem allgemeinen MATLAB-Workspace verwaltet, sondern im jeweiligen Function-Workspace. Ein Function-File kann ein Script-File aufrufen, wobei dieses Script-File im Function-Workspace ausgewertet wird, und nicht im MATLAB-Workspace. Ein Function-File kann andere Function-Files aufrufen. Es muss weder Ein- noch Ausgabeargumente besitzen. Erwähnt sei, dass Funktionen mit dem MATLAB-Workspace über globale Variablen kommunizieren können (siehe `help global`), womit man lange Übergabelisten vermeiden kann.

Die folgende Funktion mit dem Dateinamen `dyad.m` berechnet das dyadische Produkt zweier Vektoren.

```
function C = dyad(a,b)
% DYAD berechnet das dyadische Produkt zweier Spaltenvektoren

   C = a * b';
```

Man erhält zum Beispiel folgende Ausgaben:

```
>> x = [1 2 3]';
>> y = [8 0 1]';
>> dyad(x,y)

ans =

     8     0     1
    16     0     2
    24     0     3

>> help dyad

  DYAD berechnet das dyadische Produkt zweier Spaltenvektoren
```

Besitzt ein Function-File mehrere Ausgabeargumente, so liefert der Aufruf `funktionsname(In_1,...,In_m)` lediglich das erste Ausgabeargument `Out_1`. Die Werte weiterer Ausgabeargumente erhält man durch eine Zuweisung der Form

```
>> [a b c] = funktionsname(In_1,...,In_m)
```

Umfasst ein Function-File nur wenige Anweisungen, so kann es günstiger sein, stattdessen eine *Inline-Funktion* zu definieren. Das Abspeicheren als Datei entfällt hierbei. Ist zum Beispiel

```
>> f = inline('sqrt(x.^2+y.^2)','x','y')

f =
     Inline function:
     f(x,y) = sqrt(x.^2+y.^2)
```

so kann man diese Funktion in gewohnter Weise auswerten:

```
>> f(4,5)

ans =
    6.4031
```

Ein bequemes Programmierwerkzeug stellt die MATLAB-Funktion `feval` dar, die den Namen der zu benutzenden Funktion durch Auswerten einer Zeichenkette bestimmt:

```
>> a = feval(['fun',int2str(nummer)],x)
```

wertet zum Beispiel die Funktion `fun1` an x aus, wenn `nummer` den Wert 1 hat, oder die Funktion `fun2`, wenn `nummer` den Wert 2 hat, etc. Allgemein kann man sogar jede Folge von Anweisungen zunächst zu einem String `s` zusammenfügen und dann mit

```
>> eval(s)
```

ausführen.

2.7 Vergleichsoperatoren und logische Operatoren

Die Vergleichsoperatoren $<, \leq, >, \geq, =$ und $\neq$ haben die MATLAB-Syntax `<, <=, >, >=, ==` bzw. `~=`. Sie dienen dazu, zwei Matrizen elementweise miteinander zu vergleichen und für jedes Element das Ergebnis wahr/falsch in der Codierung 1/0 auszugeben:

```
>> x = 1:5

x =

     1     2     3     4     5

>> x >= 3

ans =

     0     0     1     1     1
```

Die Vergleichsfunktion `any` liefert 1, wenn mindestens ein Element des Argumentes ungleich 0 ist, während `all` nur 1 erzeugt, wenn alle Elemente ungleich 0 sind. Im obigen Beispiel ist also

```
>> c = [any(ans) all(ans)]

c =

     1     0
```

Die logischen Operatoren *und, oder* und *nicht* dienen zum Verknüpfen von Wahrheitswerten und besitzen als MATLAB-Syntax die Sonderzeichen `&`, `|` bzw. `~`. Zum Beispiel gilt für obigen Vektor c:

```
>> [c(1) & c(2), c(1) & ~c(2), c(1) | c(2)]

ans =

     0     1     1
```

2.8 Steuerstrukturen

Steuerstrukturen erlauben es, den Ablauf eines Programmes zu steuern. MATLAB bietet vier Möglichkeiten, den sequentiellen Ablauf durch *Schleifen* und *Verzweigungen* zu ändern, nämlich `for`-Schleifen, `while`-Schleifen, `if`-Verzweigungen und `switch`-Verzweigungen.

For-Schleifen. Ein For-Schleife wiederholt eine Gruppe von Anweisungen nach Maßgabe eines festen Zählers. Die allgemeine Syntax lautet

```
for spalte = Matrix
   Irgendwelche Anweisungen
end
```

Hier nimmt die Variable `spalte` als Werte nacheinander die Spalten von `Matrix` an, und mit jedem dieser Werte wird einmal der Anweisungsblock durchlaufen. Daher legt die Spaltenanzahl von `Matrix` die Anzahl der Schleifendurchläufe fest. Oft ist `Matrix` ein Zeilenvektor und `spalte` entsprechend ein Skalar. Eine typische For-Schleife tritt bei der folgenden Berechnung der Fakultät von 5 auf:

```
>> n = 5;
>> fak = 1;
>> for j = 1:n
fak = fak * j;
end
>> fak

fak =

   120
```

While-Schleifen. Eine While-Schleife wiederholt eine Gruppe von Anweisungen so lange, bis ein Testausdruck wahr wird. Ihre allgemeine Syntax lautet

```
while testausdruck
   Irgendwelche Anweisungen
end
```

Das folgende Script berechnet in der Variable `sum` die Summe der ersten hundert natürlichen Zahlen:

```
>> sum = 0;
>> k = 1;
>> while k <= 100
sum = sum + k;
k = k + 1;
end
```

If-Verzweigungen. Eine If-Verzweigung wertet einen Testausdruck aus und lässt eine Gruppe von Anweisungen ausführen, wenn der Ausdruck wahr ist. Die allgemeine Struktur lautet

```
if testausdruck
   Irgendwelche Anweisungen
end
```

Eine zweiseitige Auswahl ist mit der `else`-Anweisung möglich:

```
if testausdruck
   Irgendwelche Anweisungen
else
   Irgendwelche andere Anweisungen
end
```

Für mehrseitige Auswahlen kann man `elseif` benutzen:

```
if testausdruck_1
   Irgendwelche Anweisungen
elseif testausdruck_2
   Irgendwelche andere Anweisungen
elseif testausdruck_3
   Irgendwelche weitere andere Anweisungen

            ⋮

else
   Irgendwelche ganz andere Anweisungen
end
```

Beispiel:

```
>> x = 1;
>> if x >= 5
      disp('x ist zu gross')
   elseif x < -3
      disp('x ist zu klein')
   elseif x == 0
      disp('x verschwindet')
   else
      disp('x ist okay')
   end

x ist okay
```

Switch-Verzweigungen. Die Switch-Anweisung lässt Gruppen von Anweisungen entsprechend dem Wert einer Variablen oder eines Ausdrucks ausführen. Die Schlüsselworte `case` und `otherwise` begrenzen diese Gruppen. Es wird nur die erste Übereinstimmung ausgeführt. Die allgemeine Syntax lautet

```
switch testausdruck
case konstante_1
   anweisung_1
case konstante_2
   anweisung_2

           ⋮

case konstante_N
   anweisung_N
otherwise
   anweisung_N+1
end
```

Beispiel:

```
>> x = 1;
>> switch x
   case -1
      disp('x ist -1')
   case 0
      disp('x verschwindet')
   case 1
      disp('x ist 1')
   otherwise
      disp('x ist weder -1, 0 noch 1')
   end

x ist 1
```

Weitere Sprunganweisungen. Die `break`-Anweisung dient dazu, aus einer Schleife herauszuspringen. Liegt eine verschachtelte Schleife vor, so springt MATLAB in die nächstübergeordnete.

Die `return`-Anweisung beendet eine Funktion und springt in das aufrufende Programm zurück.

Die `error`-Anweisung beendet den Programmablauf mit einer Fehlermeldung und gibt die Kontrolle an das Kommandofenster zurück:

```
error('Fehler 404: ...')
```

Im Gegensatz dazu gibt die `warning`-Anweisung nur einen Warnhinweis aus, setzt das Programm dann aber fort:

```
warning('Achtung: ...')
```

Die Ausgabe von Warnungen kann man mit

```
>> warning off
```

unterdrücken.

2.9 Visualisierung

MATLAB verfügt über eine Vielzahl von modernen und leicht zu handhabenden Visualisierungsmöglichkeiten. Drei von ihnen werden im folgenden behandelt: die Darstellung eindimensionaler Daten mit `plot`, die Darstellung zweidimensionaler Daten mit `mesh` sowie die Erstellung von animierten Grafiken mit `movie`.

Darstellung eindimensionaler Daten. Der `plot`-Befehl benötigt als Eingabe zwei Vektoren gleicher Länge, etwa x und y. Die Werte in y werden als Daten über den Stützstellen in x aufgefasst, und durch lineare Interpolation zwischen diesen Datenpunkten entsteht ein Funktionsgraph:

```
>> plot([3 4 5],[8 7 9])
```

öffnet ein Grafikfenster mit dem in Abbildung 1 dargestellten Inhalt:

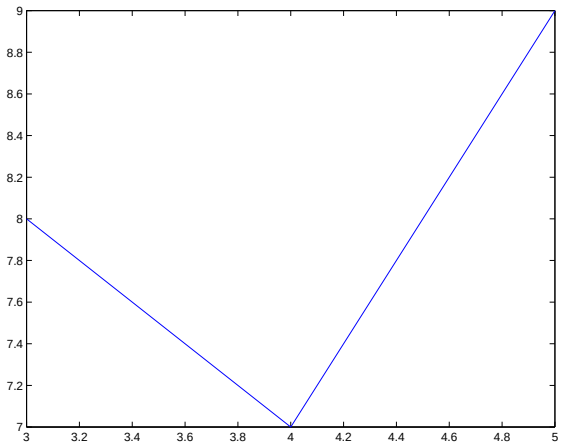

Abbildung 1: Ein eindimensionaler Plot von drei Datenpunkten

Achsen kann man mit `xlabel` und `ylabel` beschriften, und mit `axis` skalieren. Einen Titel fügt das Kommando `title` hinzu, und ein Gitter wird mit `grid` eingeblendet:

```
>> x = [-1:.1:1];
>> plot(x,x.^3)
>> xlabel('x')
>> ylabel('x^3')
>> title('Funktionsgraph von x^3')
>> grid
```

erzeugt die Grafik in Abbildung 2.

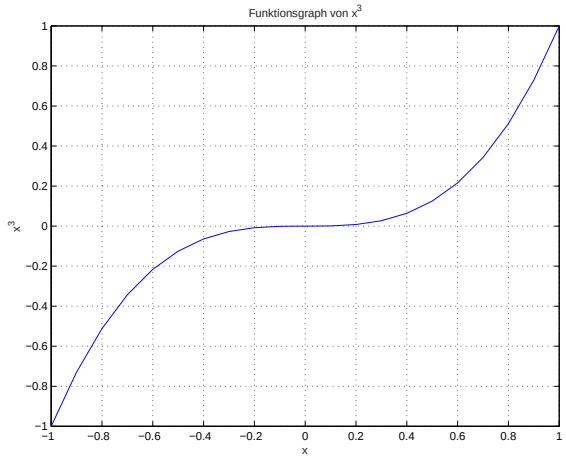

Abbildung 2: Ein Plot von x^3

Darstellung zweidimensionaler Daten. Der Befehl `mesh` stellt die Einträge einer Matrix Z mit m Zeilen und n Spalten als Funktionsgraph einer reellwertigen Funktion von zwei Variablen dar. Als „diskretisierte Koordinatenachsen“ gibt man einen Vektor x der Länge n und einen Vektor y der Länge m an:

```
>> x = 1:4;
>> y = 1:3;
>> Z = [1 2 3 4; 5 6 7 8; 9 10 11 12];
>> mesh(x,y,Z)
```

erzeugt die Grafik in Abbildung 3.

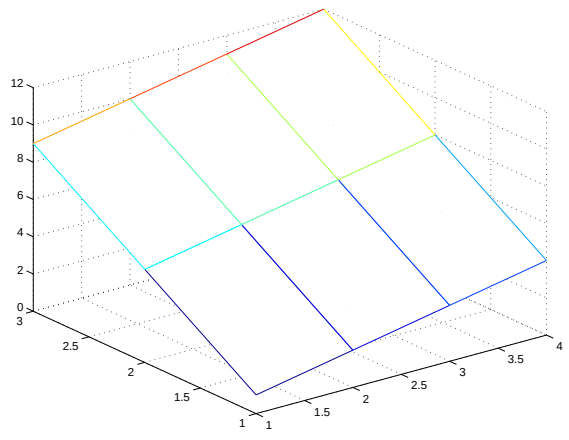

Abbildung 3: Ein zweidimensionaler Plot von zwölf Datenpunkten

Die Darstellungsfarben orientieren sich am Wert der Einträge von Z und können mit `colormap` geändert werden, etwa mit den Argumenten `gray`, `pink`, `hot` oder `cool`.

Möchte man nun mit Koordinatenvektoren x und y eine Funktion auswerten und das Ergebnis in die Matrix Z schreiben, so stellt sich das Problem, aus den Einträgen von x und y ein zweidimensionales Gitter zu konstruieren. Dies übernimmt der Befehl `meshgrid`.

```
>> x = -4:.1:4;
>> y = -4:.05:0;
>> [X,Y] = meshgrid(x,y);
>> mesh(x,y,cos(X.*Y))
>> xlabel('x')
>> ylabel('y')
>> zlabel('cos(x \cdot y)')
>> title('Funktionsgraph von cos(x \cdot y)')
```

resultiert in der Grafik aus Abbildung 4.

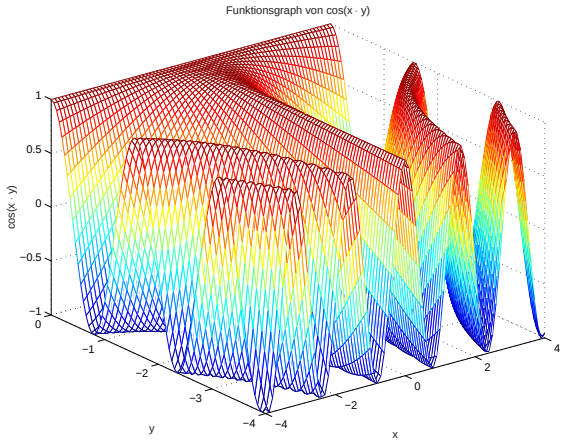

Abbildung 4: Ein Plot von $\cos(x \cdot y)$

Bequem ist, dass man für die Grafikbeschriftungen gängige LaTeX-Befehle verwenden darf.

Animierte Grafiken. Mit dem Kommando `movie` stellt MATLAB die Möglichkeit bereit, Trickfilme zu erzeugen. Dazu erstellt man zunächst in einer Schleife eine Reihe von Grafiken und speichert diese per `getframe` als Spalten einer Matrix M. Der Befehl `movie(M,number,speed)` spielt dann die gespeicherten Grafiken nacheinander ab, und zwar so oft wie `number` angibt, und mit `speed` Bildern pro Sekunde (bis zu einer rechnerbedingten Obergrenze). Obwohl es in MATLAB 5.3 laut Hilfetext nicht mehr nötig ist, empfiehlt es sich dennoch, die Matrix M zunächst mit `moviein` zu initialisieren.

Einen Film mit Schnitten der Grafik aus Abbildung 4 für y zwischen 0 und -4 liefern die folgenden Anweisungen:

```
>> x = -4:.1:4;
>> y = -4:.1:0;
>> M = moviein(41);
>> for j = 1:41;
      plot(x,cos(x*y(j)))
      title(['Die Funktion cos(x \cdot y) mit y = ',num2str(y(j))])
      xlabel('x')
      ylabel('cos(x \cdot y)')
      axis([-4 4 -1 1])
      M(:,j) = getframe;
   end
>> movie(M,3,20)
```

2.10 Einige Anmerkungen

Da sich die Syntax von MATLAB sehr stark an mathematische Schreibweisen anlehnt, lässt sich Programmcode produzieren, der leicht lesbar ist, und den man auch selber noch auf Anhieb versteht, wenn man ihn einige Monate beiseite gelegt hatte... Erstes Instrument hierzu ist der Hilfetext, den man jeder Funktion und auch jedem Script voranstellen sollte (auch die ersten Kommentarzeilen eines Scriptes werden mit `help` ausgegeben) und der eine kurze Erklärung des m-Files enthält.

Das Programm selber lässt sich nicht nur durch Kommentarzeilen lesbar gestalten, sondern auch indem man aussagekräftige und suggestive Variablen- und Script-Namen benutzt. Dazu lassen sich beispielsweise auch *structs* einsetzen, das sind Variablen mit „Untervariablen“, die durch Dezimalpunkte eingeleitet werden, etwa

```
>> parameter.epsilon = 1e-6;
>> parameter.delta = 1e-3;
>> parameter.anzahl = 20;
>> parameter.name = 'test';
>> parameter

parameter =

    epsilon: 1.0000e-06
      delta: 1.0000e-03
     anzahl: 20
       name: 'test'
```

Als Nebeneffekt vermeidet man lange Übergabelisten an Funktionen, da man dort nur den Hauptnamen der Variable (hier also `parameter`) anzugeben braucht.

MATLAB kann Vektoren und Matrizen schnell verarbeiten, ist dagegen bei der Ausführung von Schleifen aber weniger effizient. „Vektorisieren“ Sie daher so viele Schleifen wie möglich. Dabei hilft, dass viele skalare MATLAB-Funktionen auch Vektoren verarbeiten. Ersetzen Sie zum Beispiel

```
>> for j = 1:21;
      y(j) = cos(0.2*(j-1));
   end
```

durch

```
y = cos(0:.2:4)
```

Lässt sich eine Schleife dennoch nicht vermeiden, so sollte man jede Variable, die darin nach und nach belegt wird, vor dem Start der Schleife initialisieren, im obigen Beispiel etwa mit

```
>> y = zeros(1,21)
```

Damit kann MATLAB der Variable vor Beginn des Schleifendurchlaufs genügend Speicherplatz zur Verfügung stellen, während ohne Initialisierung der Speicherbedarf in jedem Durchlauf neu ermittelt werden muss, was unnötigen Zeitaufwand bedeutet.

Abschließend noch ein Wort zu den MATLAB-Suchpfaden. Das sind Unterverzeichnisse, in denen MATLAB bei Eingabe eines Befehls nach ausführbaren Scripts sucht. Das Kommando `path` zeigt alle gesetzten Suchpfade. Eigene Suchpfade fügt man hinzu, indem man etwa

```
path(path,'/home/username/MyMatlabFiles')
```

(hier in UNIX-Schreibweise) eingibt. Meist verwaltet MATLAB die Pfade allerdings so, dass man sich nicht selber um die Definitionen zu kümmern braucht.

3 Übungsaufgaben

Die nachfolgenden Übungen trainieren den Umgang mit den grundlegenden MATLAB-Kommandos. Greifen Sie dabei auch auf Befehle zurück, die in diesem Kurs nicht besprochen wurden. In Übung 4 können Sie etwa das Kommando zur Größenbestimmung einer Matrix benutzen, in Übung 5 die Sortierfunktion für einen Vektor. Suchen Sie diese Kommandos zum Beispiel mit `lookfor` und `help`.

Übung 1: Schreiben Sie ein Script-File zur Berechnung des Volumens eines Quaders. Länge, Breite und Höhe sollen dabei von der Tastatur eingelesen und das Ergebnis auf den Bildschirm ausgegeben werden.

Übung 2: Schreiben Sie ein Function-File zur Berechnung von Umfang und Flächeninhalt eines Kreises. Input-Argument soll der Kreisradius sein, Outputargumente Umfang und Flächeninhalt.

Übung 3: Schreiben Sie ein Function-File für die stückweise definierte Funktion

$$f(x) \;=\; \begin{cases} 0\,, & x < 0 \\ \frac{1}{2}\,x^2\,, & 0 \le x \le 1 \\ x\,, & x \ge 1\ . \end{cases}$$

Übung 4: Schreiben Sie ein Function-File zur Berechnung des dyadischen Produktes zweier Spaltenvektoren, das bei Eingabe von Zeilenvektoren eine Warnung ausgibt und die Zeilen zur Berechnung des Produktes in Spalten umwandelt.

Übung 5: Schreiben Sie ein Function-File zur Bestimmung des zweitgrößten Eintrags eines Vektors.

Übung 6: Schreiben Sie ein Function-File das zu gegebener natürlicher Zahl n den Wert

$$\min\{k \in I\!N \,|\, \sin(k) \ge \cos(n)\}$$

bestimmt.

Übung 7: Schreiben Sie ein Programm zur Berechnung der Quersumme einer natürlichen Zahl.

Übung 8: Plotten Sie die Funktion aus Übung 3.

Übung 9: Plotten Sie die Funktion

$$f(x) \;=\; (x_1^2 - x_2) \cdot (x_1^2 - 3x_2)$$

mit Argumenten aus einer Umgebung des Nullpunktes.

Übung 10: Erstellen Sie für die Funktion f aus Übung 9 eine animierte Grafik, die illustriert, dass die Einschränkung von f auf jede beliebige Ursprungsgerade eine lokales Minimum am Koordinatenursprung besitzt.

4 Internet-Ressourcen

Im Internet findet sich eine große Zahl von Matlab-Einführungen, die man am besten über eine Suchmaschine erkundet. An weiterführenden Texten möchten wir hier nur die umfangreiche Online-Dokumentation von The MathWorks (`www.mathworks.com/access/helpdesk/help/helpdesk.shtml`) sowie die Matlab-Linksammlung von Mathtools.net (`www.mathtools.net`) empfehlen.

Literatur

[1] G. Gramlich, W. Werner, *Numerische Mathematik mit* Matlab, dpunkt-Verlag, 2000.