KURZE EINFÜHRUNG IN MATLAB[1]

Peter Arbenz
Institut für Computational Science
ETH Zürich

November 2005/Januar 2006

[1] http://people.inf.ethz.ch/arbenz/MatlabKurs/

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Was ist Matlab [14]

MATLAB ist eine Hoch-Leistungs-Sprache für technisches Rechnen (Eigenwerbung). MATLAB integriert Berechnung, Visualisierung und Programmierung in einer leicht zu benützenden Umgebung (graphisches Benützer-Oberflächen, GUI). Probleme und Lösungen werden in bekannter mathematischer Notation ein- und ausgegeben. Typische Verwendungen von MATLAB sind:

- Technisch-wissenschaftliches Rechnen.
- Entwicklung von Algorithmen.
- Datenaquisition.
- Modellierung, Simulation und Prototyping.
- Datenanalyse und Visualisierung.
- Graphische Darstellung von Daten aus Wissenschaft und Ingenieurwesen.
- Entwicklung von Anwendungen, einschliesslich graphischer Benützer-Oberflächen.

MATLAB ist ein interaktives System dessen grundlegender Datentyp das Array (oder Matrix) ist, das nicht dimensioniert werden muss. Es erlaubt viele technische Probleme (vor allem jene, die in Matrix- / Vektornotation beschrieben sind) in einem Bruchteil der Zeit zu lösen, die es brauchen würde, um dafür ein Program in C oder FORTRAN zu schreiben.

MATLAB steht für MATrizen-LABoratorium. MATLAB wurde ursprünglich als interaktives Programm (in FORTRAN) geschrieben, um bequemen Zugriff auf die bekannte Software für Matrixberechnungen aus den LINPACK- and EISPACK-Projekten zu haben. Heutzutage umfasste die MATLAB-Maschine die LAPACK und BLAS-Bibliotheken, welche den *state of the art* der Matrixberechungen sind.

MATLAB hat sich aber sehr stark entwickelt. Es ist nicht mehr nur auf die Basis-Algorithmen der numerischen linearen Algebra beschränkt. Mit sogenannte *Toolboxen* kann MATLAB durch anwendungsspezifischen Lösungsverfahren erweitert werden. Toolboxen sind Sammlungen von MATLAB-Funktionen

(M-Files). Gebiete für die es Toolboxen gibt sind z.B. Signalverarbeitung, Regelungstechnik, Neuronale Netzwerke, 'fuzzy logic', Wavelets, Simulation und viele andere.

1.2 Geschichtliches [8]

Die lineare Algebra, insbesondere Matrix-Algebra, ist beim wissenschaftlichen Rechnen von grosser Bedeutung, weil die Lösung vieler Probleme sich aus Grundaufgaben aus diesem Gebiet zusammensetzt. Diese sind im wesentlichen Matrixoperationen, Lösen von linearen Gleichungssystemen und Eigenwertprobleme.

Diese Tatsache wurde früh erkannt und es wurde deshalb schon in den 60-er Jahren an einer Programmbibliothek für lineare Algebra gearbeitet. Damals existierten für wissenschaftliches Rechnen nur die beiden Programmiersprachen ALGOL 60 und FORTRAN. Eine Reihe "Handbook for Automatic Computation" wurde im Springer-Verlag begonnen mit dem Ziel, eines Tages eine vollständige Bibliothek von Computerprogrammen zu enthalten. Man einigte sich als Dokumentationssprache auf ALGOL, denn

> indeed, a correct ALGOL program is the *abstractum* of a computing process for which the necessary analyses have already been performed. [1]

Band 1 des Handbuches besteht aus zwei Teilen: in Teil A beschreibt H. Rutishauser die Referenzsprache unter dem Titel "Description of ALGOL 60" [17], in Teil B "Translation of ALGOL 60" geben die drei Autoren Grau, Hill und Langmaack [9] eine Anleitung zum Bau eines Compilers.

Der zweite Band des Handbuches, redigiert von Wilkinson und Reinsch, erschien 1971. Er enthält unter dem Titel "Linear Algebra" [21] verschiedene Prozeduren zur Lösung von linearen Gleichungssystemen und Eigenwertproblemen.

Leider wurde die Handbuchreihe nicht mehr fortgesetzt, weil die stürmische Entwicklung und Ausbreitung der Informatik eine weitere Koordination verunmöglichte.

Wegen der Sprachentrennung Europa – USA:

> The code itself has to be in FORTRAN, which is the language for scientific programming in the United States.[2]

wurde Ende der siebziger Jahren am Argonne National Laboratory das LINPACK Projekt durchgeführt. LINPACK enthält Programme zur Lösung von vollbesetzten linearen Gleichungssystemen. Sie stützen sich auf die Prozeduren des Handbuchs ab, sind jedoch neu und systematisch in FORTRAN programmiert. Dies äussert sich in einheitlichen Konventionen für Namengebung, Portabilität und Maschinenunabhängigkeit (z.B. Abbruchkriterien), Verwendung von elementaren Operationen mittels Aufruf der BLAS (Basic linear Algebra Subprograms). Der LINPACK Users' Guide erschien 1979 [2]. LINPACK steht auch für den Namen Name eines Benchmarks zur Leistungsmessung eines Rechners im Bereich der Fliesskommaoperationen. Früher bestand dieser Benchmark

[1] Rutishauser in [17]

[2] aus dem Vorwort des LINPACK users guide [9]

aus zwei Teilen: Einerseits musste ein vorgegebenes FORTRAN Programm zur Lösung eines voll besetzten 100×100 linearen Gleichungssystem kompiliert und ausgeführt werden, andererseits musste ein 1000×1000 Gleichungssystem möglichst schnell (mit beliebig angepasstem Programm) gelöst werden. Dieser Benchmark wird heute in veränderter Form zur Bestimmung der 500 leistungsfähigsten Computer auf der Welt benützt, die in die halbjährlich nachgeführte `top500`-Liste aufgenommen werden, siehe `http://www.top500.org`.

Auch die Eigenwertprozeduren aus [21] wurden in FORTRAN übersetzt und sind unter dem Namen EISPACK erhältlich [20, 6]. EISPACK und LINPACK sind vor einigen Jahren von LAPACK [1] abgelöst worden. Elektronisch kann man LINPACK-, EISPACK- und LAPACK-Prozeduren (und viele mehr) von der on-line Software-Bibliothek NETLIB [22] erhalten, siehe `http://www.netlib.org`.

Ende der siebziger Jahre entwickelte Cleve Moler das interaktive Programm MATLAB (MATrix LABoratory), zunächst nur mit der Absicht, es als bequehmes Rechenhilfsmittel in Vorlesungen und Übungen einzusetzen. Grundlage dafür waren Programme aus LINPACK und EISPACK. Weil Effizienzüberlegungen nicht im Vordergrund standen, wurden nur acht Prozeduren aus LINPACK und fünf aus EISPACK für Berechnungen mit vollen Matrizen verwendet. MATLAB hat sich nicht nur im Unterricht als sehr gutes Hilfsmittel etabliert, sondern wird entgegen der ursprünglichen Absicht heute auch in Industrie und Forschung eingesetzt. Das ursprüngliche in Fortran geschriebene public domain MATLAB [13] wurde von der Firma MathWorks vollständig neu überarbeitet, erweitert und zu einem effizienten Ingenieurwerkzeug gestaltet [14]. Es ist jetzt in C geschrieben.

Diese Philosophie beim Entwickeln von MATLAB hat dazu geführt, dass laufend neue Funktionen-Pakete (sog. toolboxes) für verschiedene Anwendungsgebiete geschrieben werden. Diese Pakete sind zu einem kleinen Teil öffentlich (erhältlich via netlib) zum grössten Teil werden sie aber von The MathWorks selber bietet verschiedene sog. toolboxes an. Die neueste Information findet man immer auf der WWW-Homepage von The MathWorks [23]. Natürlich kann auch jeder Benützer MATLAB durch eigene Funktionen nach seinen Anforderungen erweitern.

Vergleiche von MATLAB mit anderen ähnlichen Systemen findet man z.B. bei Higham [10] oder bei Simon und Wilson [19].

Alternativen zu MATLAB im *public domain* sind Scilab (`www.scilab.org`), welches sich recht nahe an MATLAB anlehnt. Ebenfalls gibt es Octave (`www.octave.org`), welches aber anscheinend nicht weiterentwickelt wird. Im Statistik-Bereich gibt es die Umgebung R, siehe `http://www.r-project.org/`.

Kapitel 2

Grundlegendes

2.1 Das Matlab-Fenster

Nach dem Start von MATLAB "durch Anklicken eines Icons oder durch Eintippen des Befehls `matlab` in einem Konsolenfenster (shell) wird ein Fenster wie in Abbildung 2.1 geöffnet[1].

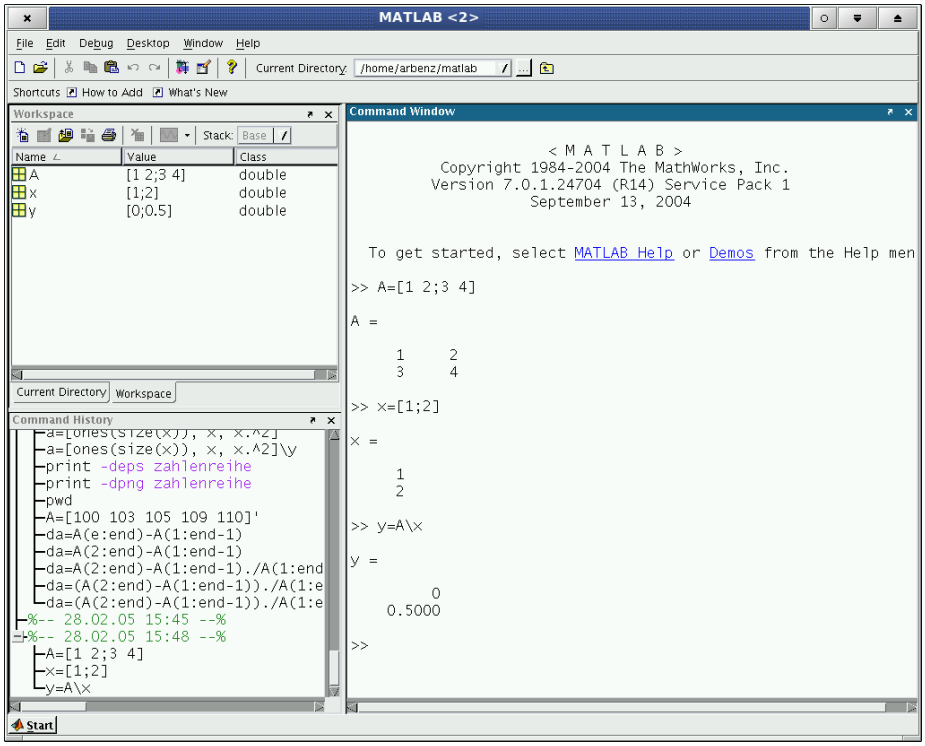

Abbildung 2.1: MATLAB-Fenster in der Default-Anordnung

[1] Durch den Befehl `matlab -nodesktop` in einem Konsolenfenster wird das Kommandofenster von MATLAB im Konsolenfenster selber gestartet.

Das Befehls-Fenster (*Command Window*) ist der Ort, an dem Sie MATLAB-Befehle eingeben werden. Ein Befehl wird rechts vom Doppelpfeil eingetippt und mit der `<Enter>`-Taste abgeschlossen. Er wird dann von MATLAB ausgeführt. Eine ganze Reihe von Befehlen können zusammen in einer Datei mit der Endung `.m` abgespeichert werden. Solche Dateien werden M-Files genannt. Die Datei wird im Befehls-Fenster mit ihrem Namen (ohne Endung) aufgerufen. Der Benutzer kann die Datei `bucky.m` also mit dem Befehl `bucky` starten. Dazu muss sich die Datei im aktuellen Verzeichnis (*current directory*) befinden, welches in der Befehlsleiste oben ausgewählt wird, oder in einem Verzeichnis des `matlabpath` befinden[2].

Im Teilfenster links unten (*command history*) werden Befehle gespeichert, die Sie bereits ausgeführt haben. Durch (Doppel-)Klick auf einen Befehl in diesem Fenster wird der Befehl ins Befehlsfenster kopiert und (noch einmal) ausgeführt. Im Teilfenster links oben werden (default-mässig) die Files des gegenwärtige Verzeichnisses (*current directoy*) angezeigt und MATLAB's Arbeitsspeicher (*workspace*). Im Arbeitsspeicher befinden sich die Variablen, die sie angelegt haben. Durch einen Doppelklick auf den Variablennamen wird ein Arrayeditor geöffnet, mit welchem man die Matrixelemente editieren kann.

Die Organisation des MATLAB-Fensters kann mit der Maus via `Desktop -> Desktop Layout` verändert werden. Man kann zum Beispiel alle Fenster ausser dem Befehlsfenster löschen. Die Teilfenster können auch vom MATLAB-Fenster getrennt (*undock*) werden. (Das Fenster hat zu diesem Zweck einen kleinen gebogenen Pfeil.)

MATLAB bietet vielfältige Hilfen an. `Help` an der oberen Befehlsleiste eröffnet ein neues Teilfenster, welches sämtliche Hilfsangebote von MATLAB auflistet. Neben der Dokumentation einzelner Befehle finden sich hier auch ein MATLAB-Einführungskurs (*Getting Started*), ein Benützer-Handbuch (*User Guide*), Demos, pdf-Files der Dokumentation, und vieles mehr.

MATLAB wird durch eintippen von

```
>> quit
```

oder

```
>> exit
```

wieder verlassen. Natürlich kann auch das einfach das MATLAB-Fenster geschlossen werden.

2.2 Hilfe aus dem Befehlsfenster

Der erste Teil des Kurses beschäftigt sich mit der Bedienung des Befehlsfensters. Nach dem Start von MATLAB sieht das Befehlsfenster so aus:

```
                    < M A T L A B >
        Copyright 1984-2004 The MathWorks, Inc.
      Version 7.0.1.24704 (R14) Service Pack 1
                  September 13, 2004
```

[2]Via `pathtool` kann der Pfad, den MATLAB bei der Befehlssuche durchläuft, modifiziert werden. `path`, `addpath`, u.ä. sind die entsprechenden Befehle, die im Befehlsfenster abgegeben werden können.

```
 To get started, select MATLAB Help or Demos from the Help menu.

>>
```

Wir wollen zunächst noch eine weitere Art erwähnen, wie Sie Hilfe zu Befehlen erhalten können: mit dem MATLAB-Befehl `help` der als Argument einen Befehlsnamen (den Sie natürlich kennen müssen) hat. Der Befehl

```
>> help help
```

gibt eine Beschreibung der vielfältigen Möglichkeiten des Befehls `help`. Da die Ausgabe dieses Befehls lang ist, sei hier ein kürzeres Beispiel gegeben.

```
>> help tic
 TIC Start a stopwatch timer.
    TIC and TOC functions work together to measure elapsed time.
    TIC saves the current time that TOC uses later to measure
    the elapsed time. The sequence of commands:

              TIC
              operations
              TOC

    measures the amount of time MATLAB takes to complete the one
    or more operations specifed here by "operations" and displays
    the time in seconds.

    See also toc, cputime.


    Reference page in Help browser
       doc tic
```

Auf die letzte Zeile (`doc tic`) kann geklickt werden[3]. Dann öffnet ein ausführliches Hilfe-Fenster im html-Format.

`lookfor` sucht in allen Files, die via `matlabpath` zureifbar sind nach einer vorgegebenen Zeichenfolge in der *ersten* Kommentarzeile. Dies ist nützlich, wenn man den Namen einer Funktion nicht mehr genau kennt.

Der Cursor kann nicht mit der Maus auf eine vorangegengene Zeile bewegt werden, um z.B. einen falschen Befehl zu korrigieren. Ein Befehl muss neu eingegeben werden. Allerdings kann ein früher eingegebener Befehl mit den Pfeiltasten (↑, ↓) auf die aktuelle Kommandozeile kopiert werden, wo er editiert werden kann[4]. Der Cursor kann unter Benutzung der Maus oder der Pfeiltasten (←, →) auf der Zeile vor- und zurückbewegt werden. Wie schon erwähnt kann ein früherer Befehl auch aus der *command history* kopiert werden.

[3] Das gilt nicht, wenn Sie MATLAB in einem Konsolenfenster mit dem Parameter `nodesktop` gestartet haben. Dort müssen Sie `doc tic` eingeben.

[4] Wenn man eine Buchstabenfolge eingibt werden beim Drücken der ↑-Taste nur die Befehle durchlaufen, die mit den angegebenen Buchstaben anfangen.

2.3 Matrizen als grundlegender Datentyp

MATLAB ist ein auf Matrizen basierendes Werkzeug. Alle Daten, die in MATLAB eingegeben werden werden von MATLAB als Matrix oder als mehrdimensionales Array abgespeichert. Sogar eine einzige Zahl wird als aine Matrix (in diesem Fall eine 1×1-Matrix) abgespeichert.

```
>> A = 100

A =

   100

>> whos
  Name      Size                    Bytes  Class

  A         1x1                         8  double array

Grand total is 1 element using 8 bytes
```

Der Befehl `whos` zeigt den Arbeitsspeicher an. Man beachte auch das Teilfenster *workspace*. Unabhängig davon, welcher Datentyp gebraucht wird, ob numerische Daten, Zeichen oder logische Daten, MATLAB speichert die Daten in Matrixform ab. Zum Beispiel ist in MATLAB die Zeichenfolge (*string*) "Hello World" ein 1×11 Matrix von einzelnen Zeichen.

```
>> b='hello world'

b =

hello world

>> whos
  Name      Size                    Bytes  Class

  A         1x1                         8  double array
  b         1x11                       22  char array

Grand total is 12 elements using 30 bytes
```

Bemerkung:

Man kann auch Matrizen aufbauen, deren Elemente komplizierterer Datentypen sind, wie Strukturen oder *cell arrays*. In diesem Kurs werden wir darauf nicht eingehen.

Noch eine Bemerkung:

> Indizes von Vektor- und Matrixelementen haben wie in FORTRAN *immer* die Werte $1, 2, \ldots$ Das erste Element eines Vektors `x` ist somit `x(1)`. Das Elementeiner Matrix `y` 'links oben' ist `y(1,1)`.

2.4 Zuweisungen

Das Grundkonstrukt der MATLAB-Sprache ist die *Zuweisung*:

$$[\mathit{Variable} =]\ \mathit{Ausdruck}$$

Ein *Ausdruck* ist zusammengesetzt aus Variablennamen, Operatoren und Funktionen. Das Resultat des Ausdrucks ist eine Matrix, welche der angeführten Variable auf der linken Seite des Gleichheitszeichens zugeordnet wird. Wenn keine Variable angegeben wird, wird das Resultat in die Variable `ans` gespeichert. Wenn die Zuweisung mit einem Semicolon endet, wird das Resultat nicht auf den Bildschirm geschrieben, die Operation wird aber ausgeführt! Wenn eine Zuweisungen länger als eine Zeile wird, müssen die fortzusetztenden Zeilen mit *drei* Punkten beendet werden.

```
>> s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
   - 1/8 + 1/9 - 1/10;
>> sin(pi/4)

ans =

    0.7071
```

Man beachte, dass der Variable auf der linken Seite des Gleichheitszeichens ein beliebiges Resultat (eine beliebige Matrix) zugewiesen werden kann. MATLAB kümmert sich um die Zuordnung des nötigen Speicherplatzes.

2.5 Namen

Namen von Variablen und Funktionen beginnen mit einem Buchstaben gefolgt von einer beliebigen Zahl von Buchstaben, Zahlen oder Unterstrichen (_). Matlab unterscheidet zwischen Gross- und Kleinschreibung!

```
>> I
??? Undefined function or variable 'I'.

>> minus1 = i^2

minus1 =

    -1

>> a_b_c = 1*2*3*pi

a_b_c =

   18.8496

>> 1minus
??? 1minus
|
Error: Unexpected MATLAB expression.
```

```
>> t=i/0
Warning: Divide by zero.

t =

      NaN +    Infi

>>
```

MATLAB rechnet nach dem IEEE Standard für Fliesskommazahlen [16].

2.6 Eingabe von Matrizen

In MATLAB können Matrizen auf verschiedene Arten eingegeben werden:

1. durch explizite Eingabe der Matrixelemente,
2. durch Aufruf einer Matrix-generierenden Funktion,
3. durch Aufbau einer Matrix durch ein eigenes M-File, oder
4. durch Laden aus einem externen File.

2.6.1 Matrixaufbau durch explizite Eingabe der Matrixelemente

Die Grundregeln bei der Eingabe von Matrixelementen sind:

- Elemente einer (Matrix-)Zeile werden durch Leerzeichen oder Komma getrennt.
- Das Ende einer Zeile wird durch einen Strichpunkt (`;`) angegeben.
- Die ganze Liste von Zahlen wird in eckige Klammern (`[ ]`) geschlossen.

Da Vektoren $n \times 1$- oder $1 \times n$-Matrizen sind gilt nachfolgendes in analoger Weise für Vektoren.

Das magische Quadrat von Dürer, siehe Abb. 2.2 erhält man durch Eingabe von

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]

A =

    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1

>>
```

Ein Spaltenvektor ist eine $n \times 1$-Matrix, ein Zeilenvektor ist eine $1 \times n$-Matrix und ein Skalar ist eine 1×1-Matrix. Somit ist nach obigem

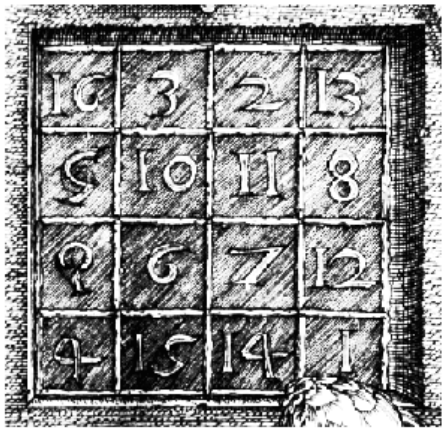

Abbildung 2.2: Das magische Quadrat aus Dürers Zeichnung “Melancholie”.

```
>> u = [3; 1; 4], v = [2 0 -1], s = 7

u =

     3
     1
     4


v =

     2     0    -1


s =

     7

>>
```

Wenn klar ist, dass ein Befehl noch nicht abgeschlossen ist, so gibt MATLAB keine Fehlermeldung. Man kann eine Matrix so eingeben:

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

B =

     1     2
     3     4
     5     6
```

```
>>
```

Auch Matrizen können als Elemente in neuen Matrizen verwendet werden. Hier ist ein Beispiel in welchem eine Matrix "rekursiv" definiert wird.

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> A = [A 2*A -A]
A =
     1     2     3     2     4     6    -1    -2    -3
     4     5     6     8    10    12    -4    -5    -6
     7     8     9    14    16    18    -7    -8    -9
>>
```

Dies ist ein schönes Beispiel fuer die dynamische Speicherplatzallokation von MATLAB. Zunächst wurden neun Fliesskommazahlen (72 Byte) für A reserviert. Nach der Ausführung des Ausdrucks belegt A dreimal mehr Speicherplatz, d.h., 216 Byte.

2.6.2 Matrixaufbau durch Aufruf einer matrixbildenden Funktion

In MATLAB gibt es verschiedene Matrix generierenden Funktionen. Die m.E. wichtigsten sind in Tabelle 2.1 aufgelistet.

`eye`	Einheitsmatrix
`zeros`	Nullmatrix
`ones`	Einser-Matrix
`diag`	siehe `help diag`
`triu`	oberes Dreieck einer Matrix
`tril`	unteres Dreieck einer Matrix
`rand`	Zufallszahlenmatrix

Tabelle 2.1: Matrixbildende Funktionen

`zeros(m,n)` produziert eine $m \times n$-Matrix bestehend aus lauter Nullen; `zeros(n)` ergibt eine $n \times n$-Nuller-Matrix. Wenn A eine Matrix ist, so definiert `zeros(A)` eine Nuller-Matrix mit den gleichen Dimensionen wie A.

Hier einige Beispiele

```
>> A = [ 1 2 3; 4 5 6; 7 8 9 ];
>> diag(A)  % speichert die Diagonale einer Matrix als Vektor
ans =
     1
     5
     9

>> diag([1 2 3])  % macht aus einem Vektor eine Diagonalmatrix
ans =
```

```
     1     0     0
     0     2     0
     0     0     3

>> eye(size(A))   % erzeugt die Einheitsmatrix von
                  % gleicher Groesse wie A
ans =
     1     0     0
     0     1     0
     0     0     1

>> ones(size(A))  % erzeugt eine Matrix mit Einselementen
ans =
     1     1     1
     1     1     1
     1     1     1

>> zeros(size(A))   %  erzeugt eine Nullmatrix
ans =
     0     0     0
     0     0     0
     0     0     0
>> triu(A)
ans =

     1     2     3
     0     5     6
     0     0     9

>> M = magic(4)
M =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

2.6.3 Matrixaufbau durch Aufbau einer Matrix durch ein eigenes M-File

Wir werden die Struktur von M-Files später eingehend behandeln. Hier erwähnen wir nur die sogenannten *Script-Files*: Diese ASCII-Files enthalten einfach eine Reihe von MATLAB-Befehlen, die beim Aufruf des Files einer nach dem anderen ausgeführt wird als wären sie auf der Kommandozeile eingegeben worden. Sei `matrix.m` ein File bestehend aus einer einzigen Zeile mit folgendem Text:

```
A = [ 1 2 3; 4 5 6; 7 8 9 ]
```

Dann bewirkt der Befehl `matrix`, dass der Variable `A` ebendiese 3×3-Matrix zugeordnet wird.

```
>> matrix
```

```
A     =

     1     2     3
     4     5     6
     7     8     9
```

2.6.4 Matrixaufbau durch Laden aus einem externen File

Die Funktion `load` liest binäre Files, die in einer früheren MATLAB-Sitzung erzeugt wurden oder liest Textfiles, die numerische Daten enthalten. Das Textfile sollte so organisiert sein, dass es eine rechteckige Tabelle von Zahlen enthält, die durch Leerzeichen getrennt sind, eine Textzeile pro Matrixzeile enthalten. Jede Matrixzeile muss gleichviele Elemente haben. Wenn z.B. ausserhalb von MATLAB eine Datei `mm.m` mit den vier Zeilen

```
16.0     3.0     2.0    13.0
 5.0    10.0    11.0     8.0
 9.0     6.0     7.0    12.0
 4.0    15.0    14.0     1.0
```

generiert wurde, so liest der Befehl `load mm` das File und kreiert eine Variable `mm`, die die 4×4-Matrix enthält.

Eine weitere Möglichkeit, Daten aud Files zu lesen, bietet der Befehl `dlmread`. Der Import Wizard (`File -> Import Data...`) erlaubt es, Daten in verschiedenen Text und binären Formaten in MATLAB zu laden.

2.6.5 Aufgaben

1. Überlegen Sie sich Varianten, wie man in MATLAB die Matrix

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

   definieren kann.

2. Wie würden Sie vorgehen, wenn Sie zu vorgegebenem n eine $n \times n$ Matrix mit analoger Struktur wie vorher (lauter Einsen ausser auf der Diagonale, wo die Elemente null sind) aufbauen wollen.

3. Geben Sie die Matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

   ein. Überlegen Sie sich, was passiert, wenn Sie den Befehl

   ```
   A(4,6) = 17
   ```

   eintippen. Danach geben Sie den Befehl ein.

4. Finden Sie heraus, was der Befehl `fliplr` (oder `flipud`) macht. Konstruieren Sie danach die Matrix

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

5. Wie wäre es mit

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}?$$

6. Definieren Sie die Matrizen O und E

```
n = 5; O=zeros(n), E=ones(n)
```

Dann erzeugen Sie das Schweizerkreuz von Figur 2.3, d.h. eine 15×15 Matrix A. Ein Punkt bedeutet dabei eine Eins, kein Punkt eine Null.

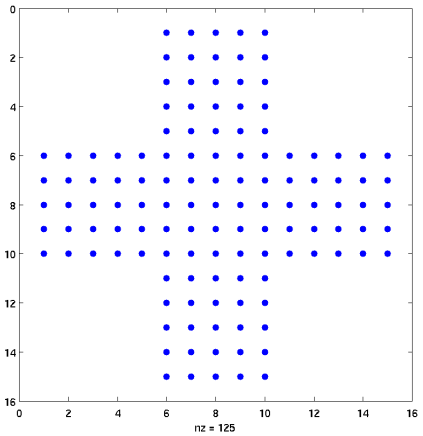

Abbildung 2.3: Einfaches Schweizerkreuz

Abbildung 2.3 können Sie mit

```
spy(A)
```

erzeugen.

2.7 Operationen mit Matrizen

2.7.1 Addition und Subtraktion von Matrizen

Addition und Subtraktion von Matrizen geschieht elementweise. Matrizen, die addiert oder voneinander subtrahiert müssen dieselben Dimensionen haben.

```
>> A=ones(3)

A =

     1     1     1
     1     1     1
     1     1     1

>> B = round(10*rand(3))  % rand: Zufallszahlenmatrix

B =

    10     5     5
     2     9     0
     6     8     8

>> B-A

ans =

     9     4     4
     1     8    -1
     5     7     7

>>
```

Falls die Dimensionen nicht zusammenpassen bring MATLAB eine Fehlermeldung.

```
>> A + eye(2)
??? Error using ==> plus
Matrix dimensions must agree.

>>
```

2.7.2 Vektorprodukte and Transponierte

Ein Zeilenvektor und ein Spaltenvektor können miteinander multipliziert werden. Das Resultat ist entweder ein Skalar (das innere oder Skalarprodukt) oder eine Matrix, das äussere Produkt.

```
>> u = [3; 1; 4];
>> v = [2 0 -1]; x = v*u

x =

     2

>> X = u*v

X =
```

```
     6     0    -3
     2     0    -1
     8     0    -4

>>
```

Bei reellen Matrizen spiegelt die Transposition die Elemente an der Diagonale. Seinen die Elemente der Matrix $m \times n$ Matrix A bezeichnet als $a_{i,j}$, $1 \leq i \leq m$ und $1 \leq j \leq n$. Dann ist die Transponierte von A die $n \times m$ Matrix B mit den Elementen $b_{j,i} = ai, j$. MATLAB benützt den Apostroph für die Transposition.

```
>> A = [1 2 3;4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>> B=A'

B =

     1     4     7
     2     5     8
     3     6     9

>>
```

Transposition macht aus einem Zeilenvektor einen Spaltenvektor.

```
>> x=v'

x =

     2
     0
    -1

>>
```

Das Produkt zweier reeller Spaltenvektoren gleicher Länge ist nicht definiert, jedoch sind die Produkte `x'*y` und `y'*x` gleich. Sie heissen Skalar- oder inneres Produkt.

Für einen komplexen Vektor `z` bedeutet `z'` *komplex-konjugierte* Transposition:

```
>> z = [1+2i 3+4i]

z =
```

```
   1.0000 + 2.0000i   3.0000 + 4.0000i

>> z'

ans =

   1.0000 - 2.0000i
   3.0000 - 4.0000i

>>
```

Für komplexe Vektoren sind die beiden Skalarprodukte `x'*y` und `y'*x` komplex konjugiert. Das Skalarprodukte `x'*x` ist reell. Es gibt aber auch den *punktierten* Operator `.'`

```
>> z=z.', w = [4+6i;7+8i]

z =

   1.0000 + 2.0000i
   3.0000 + 4.0000i

w =

   4.0000 + 6.0000i
   7.0000 + 8.0000i

>> z'*w, w'*z

ans =

  69.0000 - 6.0000i

ans =

  69.0000 + 6.0000i
```

2.7.3 Zugriff auf Matrixelemente

Das Element der Matrix A in Zeile i und Spalte j wird durch `A(i,j)` bezeichnet.

```
>> % Hier kommt Duerers magisches Quadrat
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
>> A(2,1) + A(2,2) + A(2,3) + A(2,4)
ans =
    34
```

```
>> A(5,1)
??? Index exceeds matrix dimensions.

>> v = [1;2;3;4;5];
>> v(4)
ans =
     4
>>
```

Bemerkung: Matrixelemente können auch mit einem einzelnen Index zugegriffen werden. MATLAB interpretiert die Matrix so, als wären alle Spalten übereinandergestapelt.

```
>> A(8)
ans =
    15
>>
```

In der 4×4-Matrix A ist des achte Element also das letzte der zweiten Spalte.

2.7.4 Der Doppelpunkt-Operator

Der Doppelpunkt ist ein äusserst wichtiger und nützlicher MATLAB-Operator. Der Ausdruck

```
1:10
```

ist ein Zeilenvektor, der die Elemente 1 bis 10 enthält. Der Ausdruck

```
1:3:10
```

liefert den Vektor `[1 4 7 10]`. Allgemein kann man schreiben

```
[i : j : k]
```

was einen Vektor mit erstem Element i erzeugt, gefolgt von $i + j$, $i + 2j$, bis zu einem Element welches $\geq k$ ist, falls $j > 0$ und $\leq k$ falls $j < 0$ ist..

```
>> 1:10
ans =
     1     2     3     4     5     6     7     8     9    10
>> 1:3:10
ans =
     1     4     7    10
>> 100:-7:50
ans =
   100    93    86    79    72    65    58    51
>> 1:3:12
ans =
     1     4     7    10
>> 100:-7:50
ans =
   100    93    86    79    72    65    58    51
>> x=[0:.25:1]
x =
         0    0.2500    0.5000    0.7500    1.0000
```

Der Doppelpunkt-Operator ist sehr bequem, wenn man Teile von Matrizen zugreiffen will.

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
>> A(2,1:4)        % 2-te Zeile von A
ans =
     5    10    11     8
>> A(2,1:3)        % Teil der 2-te Zeile von A
ans =
     5    10    11
>> A(2:3,2:3)
ans =
    10    11
     6     7
>> A(3:end,3)
ans =
     7
    14
>> A(4,:)
ans =
     4    15    14     1
>>
```

2.7.5 Arithmetische Operationen

Matrixoperationen		Elementweise Operationen	
`+`	Addition	`+`	Addition
`-`	Subtraktion	`-`	Subtraktion
`*`	Multiplikation	`.*`	Multiplikation
`/`	Division von rechts	`./`	Division von rechts
`\`	Division von links	`.\`	Division von links
`^`	Potenz	`.^`	Potenz
`'`	Hermitesch Transponierte	`.'`	Transponierte

Tabelle 2.2: Arithmetische Operationen

Mit Matrizen können die in Tabelle 2.2 aufgeführten *arithmetischen Operationen* durchgeführt werden. Unter Matrixoperationen werden dabei die aus der Matrixalgebra bekannten Operationen verstanden. Durch Voranstellen eines Punkts vor die Operatoren erhält man Operationen, welche *elementweise* ausgeführt werden.

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

a =
```

```
     1     2
     3     4

>> a^2

ans =

     7    10
    15    22

>> a.^2

ans =

     1     4
     9    16
```

Vergleichsoperatoren		logische Operatoren	
$<$	kleiner als	&	and
$<=$	kleiner oder gleich	|	or
$>$	grösser als	$\sim$	not
$>=$	grösser oder gleich		
$==$	gleich		
$\sim=$	ungleich		

Tabelle 2.3: Vergleichende und logische Operatoren

2.7.6 Vergleichsoperationen

Weiter können die in Tabelle 2.3 aufgelisteten *Vergleichsoperationen* mit Matrizen durchgeführt werden. Als Resultat einer Vergleichsoperation erhält man eine 0-1-Matrix. Erfüllen die (i, j)-Elemente der zwei verglichenen Matrizen die Bedingung, so wird das (i, j)-Element der Resultatmatrix 1, andernfalls 0 gesetzt.

```
>> b=ones(2)   %  a  wie im vorigen Beispiel

b =

     1     1
     1     1

>> a > b

ans =

     0     1
     1     1
```

2.7.7 Logische Operationen

Die *logischen Operationen* in Tabelle 2.3 werden elementweise auf 0-1-Matrizen angewandt

```
>> ~(a>b)

ans =

     1     0
     0     0
```

2.7.8 Aufgaben

1. Konstruieren Sie die 15×15-Matrix mit den Einträgen von 1 bis 225 ($= 15^2$). Die Einträge sollen lexikographisch (zeilenweise) angeordnet sein. Verwenden Sie neben dem Doppelpunkt-Operator die Funktion `reshape`.

2. Multiplizieren Sie die entstehende Matrix mit der Schweizerkreuzmatrix. Verwenden Sie das 'normale' Matrixprodukt und das elementweise. Nennen Sie die letztere Matrix C.

3. Filtern Sie jetzt aus C die Elemente zwischen 100 und 200 heraus. Dazu definieren Sie eine logische Matrix, die Einsen an den richtigen Stellen hat. Danach multiplizieren Sie diese logische Matrix elementweise mit C.

2.8 Ausgabeformate

Zahlen können in verschiendener Weise am Bildschirm angezeigt werden.

```
>> s = sin(pi/4)

s =

    0.7071

>> format long
>> s

s =

    0.70710678118655

>> format short e
>> s

s =

   7.0711e-01

>> format short
```

In Tabelle 2.4 auf Seite 24 ist eine Übersicht über die möglichen Parameter von `format` gegeben. Man beachte, dass man mit dem aus C bekannten Befehl `fprintf` die Ausgabe viel feiner steuern kann als mit `format`.

Befehl	Beschreibung
`format short`	Scaled fixed point format with 5 digits.
`format long`	Scaled fixed point format with 15 digits for double and 7 digits for single.
`format short e`	Floating point format with 5 digits.
`format long e`	Floating point format with 15 digits for double and 7 digits for single.
`format short g`	Best of fixed or floating point format with 5 digits.
`format long g`	Best of fixed or floating point format with 15 digits for double and 7 digits for single.
`format hex`	Hexadecimal format.
`format +`	The symbols +, - and blank are printed for positive, negative and zero elements. Imaginary parts are ignored.
`format bank`	Fixed format for dollars and cents.
`format rat`	Approximation by ratio of small integers.
`format compact`	Suppresses extra line-feeds.
`format loose`	Puts the extra line-feeds back in.

Tabelle 2.4: MATLAB-Formate, siehe `help format`

Kapitel 3

Funktionen

3.1 Skalare Funktionen

Viele MATLAB-Funktionen sind skalare Funktionen und werden *elementweise* ausgeführt, wenn sie auf Matrizen angewandt werden. Die wichtigsten Funktionen dieser Art sind in Tab. 3.1 gegeben. Alle Funktionen dieser Art können mit dem Befehl `help elfun` aufgelistet werden. Der Befehl `help specfun` gibt eine Liste von speziellen mathematischen Funktionen wie Gamma-, Bessel-, Fehlerfunktion, aber auch ggT oder kgV aus. Diese Funktionen werden oft auf Vektoren angewandt, wie folgendes Beispiel zeigt.

```
>> x=[0:pi/50:2*pi];
>> y = cos(x.^2);
>> plot(x,y)
>> xlabel('x-axis')
>> ylabel('y-axis')
>> title('Plot of y = cos(x^2), 0 < x < 2\pi')
```

Diese Befehlssequenz generiert Abbildung 3.1

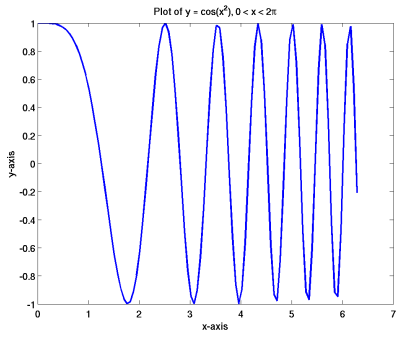

Abbildung 3.1: Plot der Funktion $\cos(x^2)$

Kategorie	Funktion	Beschreibung (gemäss MATLAB)
Trigonometrische	`sin`	Sine
	`sinh`	Hyperbolic sine
	`asin`	Inverse sine
	`cos`	Cosine
	`cosh`	Hyperbolic cosine
	`acos`	Inverse cosine
	`acosh`	Inverse hyperbolic cosine
	`tan`	Tangent
	`tanh`	Hyperbolic tangent
	`atan`	Inverse tangent
	`atanh`	Inverse hyperbolic tangent
	`cot`	Cotangent
	`coth`	Hyperbolic cotangent
	`acot`	Inverse cotangent
	`acoth`	Inverse hyperbolic cotangent
Exponentiell	`exp`	Exponential
	`expm1`	Compute exp(x)-1 accurately
	`log`	Natural logarithm
	`log1p`	Compute log(1+x) accurately
	`log10`	Common (base 10) logarithm
	`log2`	Base 2 logarithm and dissect floating point number
	`sqrt`	Square root
	`nthroot`	Real n-th root of real numbers
Komplex	`abs`	Absolute value
	`angle`	Phase angle
	`complex`	Construct complex data from real and imaginary parts
	`conj`	Complex conjugate
	`imag`	Complex imaginary part
	`real`	Complex real part
Runden und Rest	`fix`	Round towards zero
	`floor`	Round towards minus infinity
	`ceil`	Round towards plus infinity
	`round`	Round towards nearest integer
	`mod`	Modulus (signed remainder after division)
	`rem`	Remainder after division
	`sign`	Signum

Tabelle 3.1: Elementare Funktionen

3.2 Spezielle Konstanten

MATLAB hat einige spezielle Funktionen, die nützliche Konstanten zur Verfügung stellen, siehe Tab. 3.2.

Funktion	Beschreibung
`pi`	$3.14159265\ldots$
`i`	Imaginäre Einheit ($\sqrt{-1}$)
`j`	Wie `i`
`eps`	Relative Genauigkeit der Fliesskomma-Zahlen ($\varepsilon = 2^{-52}$)
`realmin`	Kleinste Fliesskomma-Zahl (2^{-1022})
`realmax`	Grösste Fliesskomma-Zahl ($(2-\varepsilon)^{1023}$)
`Inf`	Unendlich (∞)
`NaN`	Not-a-number

Tabelle 3.2: Spezielle Funktionen

Man beachte, dass Funktionennamen hinter Variablennamen versteckt werden können! Z.B. kann `i` mit einer Zahl (Matrix, etc.) überschrieben werden.

```
    >> i=3

    i =

         3

>> 3+5*i

ans =

    18
```

Wenn hier `i` für die imaginäre Einheit $\sqrt{-1}$ stehen soll, muss man die Variable `i` löschen.

```
     >> clear i
     >> i

     ans =

             0 + 1.0000i

     >> 3+5*i

     ans =

        3.0000 + 5.0000i

     >>
```

3.3 Vektorfunktionen

Eine zweite Klasse von MATLAB-Funktionen sind Vektorfunktionen. Sie können mit derselben Syntax sowohl auf Zeilen- wie auf Spaltenvektoren angewandt werden. Solche Funktionen operieren *spaltenweise*, wenn sie auf Matrizen angewandt werden. Einige dieser Funktionen sind

Funktion	Beschreibung
`max`	Largest component
`mean`	Average or mean value
`median`	Median value
`min`	Smallest component
`prod`	Product of elements
`sort`	Sort array elements in ascending or descending order
`sortrows`	Sort rows in ascending order
`std`	Standard deviation
`sum`	Sum of elements
`trapz`	Trapezoidal numerical integration
`cumprod`	Cumulative product of elements
`cumsum`	Cumulative sum of elements
`cumtrapz`	Cumulative trapezoidal numerical integration
`diff`	Difference function and approximate derivative

Tabelle 3.3: Übersicht Vektorfunktionen

```
>> z=[-3 -1 4 7 7 9 12]

z =

    -3    -1     4     7     7     9    12

>> [min(z), max(z)]

ans =

    -3    12

>> median(z)

ans =

     7

>> mean(z)

ans =

     5

>> mean(z), std(z)
```

```
ans =

     5

ans =

   5.38516480713450

>> sum(z)

ans =

    35

>> trapz(z)

ans =

  30.5000

>> (z(1) + z(end) + 2*sum(z(2:end-1)))/2

ans =

  30.5000

>> u=[1 2 3;4 5 6]

u =

     1     2     3
     4     5     6

>> max(u)

ans =

     4     5     6

>> max(max(u))

ans =

     6
```

Um das grösste Element einer Matrix zu erhalten, muss man also die Maximumfunktion zweimal anwenden.

3.4 Matrixfunktionen

Die Stärke von MATLAB sind die *Matrixfunktionen*. In Tabelle 3.4 finden Sie die wichtigsten.

Kategorie	Funktion	Beschreibung (gemäss MATLAB)
Matrixanalysise	`norm`	Matrix or vector norm.
	`normest`	Estimate the matrix 2-norm.
	`rank`	Matrix rank.
	`det`	Determinant.
	`trace`	Sum of diagonal elements.
	`null`	Null space.
	`orth`	Orthogonalization.
	`rref`	Reduced row echelon form.
	`subspace`	Angle between two subspaces.
Lineare Gleichungen	`\ and /`	Linear equation solution.
	`inv`	Matrix inverse.
	`cond`	Condition number for inversion.
	`condest`	1-norm condition number estimate.
	`chol`	Cholesky factorization.
	`cholinc`	Incomplete Cholesky factorization.
	`linsolve`	Solve a system of linear equations.
	`lu`	LU factorization.
	`luinc`	Incomplete LU factorization.
	`qr`	Orthogonal-triangular decomposition.
	`lsqnonneg`	Nonnegative least-squares.
	`pinv`	Pseudoinverse.
	`lscov`	Least squares with known covariance.
Eigen- und singuläre Werte	`eig`	Eigenvalues and eigenvectors.
	`svd`	Singular value decomposition.
	`eigs`	A few eigenvalues.
	`svds`	A few singular values.
	`poly`	Characteristic polynomial.
	`polyeig`	Polynomial eigenvalue problem.
	`condeig`	Condition number for eigenvalues.
	`hess`	Hessenberg form.
	`qz`	QZ factorization.
	`schur`	Schur decomposition.
Matrixfunktionen	`expm`	Matrix exponential.
	`logm`	Matrix logarithm.
	`sqrtm`	Matrix square root.
	`funm`	Evaluate general matrix function.

Tabelle 3.4: Übersicht Matrixfunktionen

Um die meisten dieser Funktionen zu verstehen, sind einige Kenntnisse in Linearer Algebra nötig. In dieser kurzen Einführung beschränken wir uns auf das Lösen von linearen (überbestimmten) Gleichungssystemen.

3.5 Lineare Gleichungssysteme

Das wichtigste Verfahren zur Lösung von linearen Gleichungssystemen ist die *Gauss-Elimination* oder *LU-Faktorisierung*. MATLAB stellt dafür den **Backslash-Operator** (\) zur Verfügung.

Sei ein lineares Gleichungssystem

$$A\mathbf{x} = \mathbf{b} \tag{3.1}$$

gegeben mit regulärer (nicht-singulärer) $n \times n$ Matrix

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}.$$

Dann berechnet der Befehl

```
x = A\b
```

die Lösung $\mathbf{x}$ von (3.1) vermittels Gauss-Zerlegung von A mit sog. Spaltenpivotsuche.

```
>> A

A =

     1     2     4
     3     5     4
     4     6     9

>> b

b =

     1
     1
     1

>> x=A\b

x =

   -1.6667
    1.1111
    0.1111

>> norm(b-A*x)

ans =

   2.2204e-16
```

Man kann auch Gleichungssysteme mit mehreren rechten Seiten lösen.

```
>> b=[1 1;1 2;1 3]

b =

     1     1
     1     2
     1     3

>> x=A\b

x =

   -1.6667    0.3333
    1.1111    0.1111
    0.1111    0.1111

>> norm(b-A*x)

ans =

   3.1402e-16
```

Wenn man in MATLAB den Backslash-Operator braucht, wird zwar die LU-Zerlegung der involvierten Matrix berechnet. Diese Zerlegung geht aber nach der Operation verloren. Deshalb kann es sinvoll sein die Funktion `lu` zu benützen und die Faktorisierung explizit abzuspeichern, insbesondere wenn die Gleichungssysteme mit derselben Systemmatrix hintereinander gelöst werden müssen.

3.5.1 Aufgaben

1. *Polynominterpolation.* Vorgegeben sei eine Funktion $f(x)$. Gesucht ist das Polynom
$$p(x) = a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1} x + a_n$$
vom Grad $< n$ so, dass
$$p(x_i) = f(x_i), \quad i = 1, \ldots, n.$$
Diese Interpolation ist in Matrixschreibweise
$$\begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \ldots & x_2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}$$
oder
$$V\mathbf{a} = \mathbf{f}$$
Man verwende
   - Funktionen `vander` und `polyval`

- Stützstellen `x=[0:.5:3]'`.
- Funktionswerte `f=[1 1 0 0 3 1 2]'`

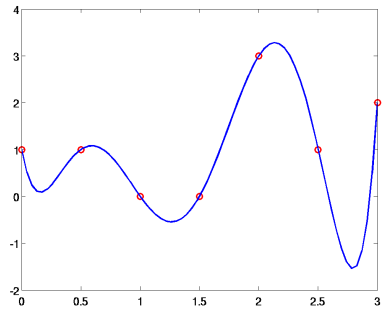

Abbildung 3.2: Polynominterpolation

Die Lösung sollte etwa so wie in Abbildung 3.2 aussehen.

2. *Was leistet Ihr Computer?*

   In dieser Aufgabe wollen wir sehen, wie schnell Ihr Computer rechnen kann.

   Die Idee ist, Gleichungssysteme mit immer höherer Ordnung mit dem Backslash-Operator zu lösen und die Ausführungszeit zu messen.

   Ihr Programm soll eine `for`-Schleife ausführen, welche für jeden der Werte in `N`,

   ```
   N = [10:10:100,150:50:500,600:100:1000];
   ```

   ein Gleichungssystem der entsprechenden Grösse löst. Ein Grundgerüst für Ihr Programm könnte so aussehen

   ```
   T = [];
   for n = ...

     % Hier wird eine Matrix der Ordung n und eine rechte Seite
     % erzeugt.  Verwenden Sie dazu die Funktion rand.
     % Reservieren Sie Platz f"ur den L"osungsvektor (zeros).

     tic

     % Hier wird das Gleichungssystem geloest.

     t = toc; T = [T,t];
   end
   ```

   Sie können nun `N` gegen die zugehörige Ausführungszeiten `T` plotten. Instruktiver ist es aber, wenn Sie zu allen Problemgrössen die Mflop/s-Rate berechnen und plotten. (Aus der linearen Algebra wissen Sie sicher,

dass das Auflösen eines Gleichungssystems der Ordnung n etwa $2/3n^3$ Fliesskomma-Operationen $(+, -, \times, /)$ kostet.) Wie gross ist die höchste Mflop/s-Rate, die Sie erhalten haben und wie gross ist MHz-Rate des Prozessors Ihres Computers? (Die erhaltene Kurve kann geglättet werden, indem man mehrere Messungen ausführt und jeweils die besten Resultate nimmt.)

3.6 Die Methode der kleinsten Quadrate (least squares)

Die sogenannte "Methode der kleinsten Quadrate" (Least Squares) ist eine Methode, um überbestimmte lineare Gleichungssysteme

$$A\mathbf{x} = \mathbf{b}, \qquad A \in \mathbb{R}^{m\times n}, \mathbf{x} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^m \tag{3.2}$$

zu lösen. Die $m \times n$-Matrix A hat *mehr* Zeilen als Spalten ($m > n$). Wir haben also mehr Gleichungen als Unbekannte. Deshalb gibt es im allgemeinen kein $\mathbf{x}$, das die Gleichung (3.2) erfüllt. Die *Methode der kleinsten Quadrate* bestimmt nun ein $\mathbf{x}$ so, dass die Gleichungen "möglicht gut" erfüllt werden. Dabei wird $\mathbf{x}$ so berechnet, dass der *Residuenvektor* $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ minimale Länge hat. Dieser Vektor $\mathbf{x}$ ist Lösung der *Gauss'schen Normalgleichungen*

$$A^T A\mathbf{x} = A^T\mathbf{b}.$$

(Die Lösung ist eindeutig, wenn A linear unabhängige Spalten hat.) Die Gaussschen Normalgleichungen haben unter Numerikern einen schlechten Ruf, da für die Konditionszahl $\mathrm{cond}(A^T A) = \mathrm{cond}(A)^2$ gilt und somit die Lösung $\mathbf{x}$ durch die verwendete Methode ungenauer berechnet wird, als dies durch die Konditionszahl der Matrix A zu erwarten wäre.

Deshalb wird statt der Normalgleichungen die *QR-Zerlegung* für die Lösung der Gleichung (3.2) nach der Methode der kleinsten Quadrate vorgezogen. In MATLAB kann man mit dem Befehl `qr` die QR-Zerlegung leicht berechnen. Noch bequemer ist aber wieder die Benützung des Backslash-Operators,

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

3.6.1 Aufgabe (Regressionsgerade)

1. Vorgegeben sei wieder eine Funktion $f(x)$. Gesucht ist nun die Gerade, d.h. das lineare Polynom

   $$p(x) = a_1 + a_2 x$$

   so, dass

   $$p(x_i) \approx f(x_i), \quad i = 1, \ldots, n.$$

   im Sinne der kleinsten Quadrate gilt.

$$\left\| \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} - \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix} \right\|_2 = \text{minimal}$$

Verwenden Sie die Daten:

```
>> n=20;
>> x = [1:n]';
>> y = x + (2*rand(size(x))-1);
```

Die Systemmatrix hat als erste Spalte die Stützstellen x_i und als zweite Spalte 1. Natürlich könnte man wieder wie bei der Polynominterpolation die Funktion `vander` benützen und dann die richtigen Spalten aus der Matrix extrahieren. Hier ist es aber einfacher, die Systemmatrix direkt aufzustellen.

Die Lösung des überbestimmten Gleichungssystems ist ein Vektor mit zwei Komponenten. Werten Sie das Polynom an den Stützstellen x_i aus, direkt oder mit `polyval`. Wenn Sie diese Werte in einem Vektor `Y` speichern und den Befehl

```
plot(x,y,'ro',x,Y,'b')
```

eingeben, dann erhalten Sie (im wesentlichen) die Abbildung 3.3.

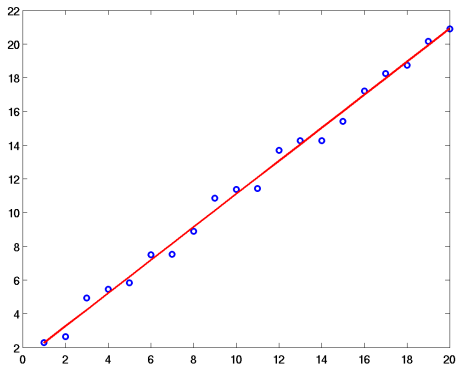

Abbildung 3.3: Regressionsgerade

Kapitel 4

Konstruktionen zur Programmsteuerung

MATLAB hat verschiedenen Konstrukte, um den Programmablauf steuern zu können. In diesem Abschnitt behandeln wir `for`-, `while`-, `if`- und `switch-case`-Konstrukte. Diese Elemente der MATLAB-Programmiersprache kann man direkt im Befehlsfenster eingeben. Ihre Verwendung finden sie in M-Files, MATLAB-Funktionen, die im nächsten Abschnitt behandelt werden.

4.1 Das `if`-Statement

Die einfachste Form des `if`-Statements ist

```
>>  if logischer Ausdruck
       Zuweisungen
    end
```

Die Zuweisungen werden ausgeführt, falls der logische Ausdruck wahr ist. Im folgenden Beispiel werden x und y vertauscht, falls x grösser als y ist

```
>> x=3;y=1;
>> if x > y
     tmp = y;
     y = x;
     x = tmp;
   end
>> [x y]

ans =

     1     3
```

Ein `if`-Statement kann auch auf einer einzigen Zeile geschrieben werden (falls es nicht zu lange ist). Kommata müssen aber an Stellen eingesetzt werden, an denen neue Zeilen anfangen und vorher keine Strichpunkte waren.

```
>> x=3;y=5;
```

```
>> if x > y, tmp = y; y = x; x = tmp; end
>> [x y]

ans =

     3     5
```

Zuweisungen, die nur ausgeführt werden sollen, wenn der logische Ausdruck falsch ist, können nach einem `else` plaziert werden.

```
e = exp(1);
if 2^e > e^2
  disp('2^e ist gr"osser')
else
  disp('e^2 ist gr"osser')
end
```

Mit `elsif` kann die Struktur noch verkompliziert werden.

```
if abs(i-j) > 1,
   t(i,j) = 0;
elseif i == j,
   t(i,j) = 2;
else
   t(i,j) = -1;
end
```

4.2 Die `switch-case`-Konstruktion

Wenn eine Variable eine feste Anzahl von Werten annehmen kann und für jeden von ihnen eine bestimmte Befehlsfolge ausgeführt werden soll, so bietet sich die `switch-case`-Konstruktion zur Implementation an. Diese hat die Form

```
>>  switch switch-Ausdruck (Skalar oder String)
       case case-Ausdruck
            Befehle
       case case-Ausdruck,
            Befehle
       ...
       otherwise,
            Befehle
    end
```

Beispiele:

```
>> switch grade   % grade soll eine integer Variable
                  % mit Werten zwischen 1 und 6 sein
   case {1,2}
      disp 'bad'
   case {3,4}
      disp 'good'
   case {5,6}
```

```
    disp 'very good'
  end

>> day_string = 'FrIday'

day_string =

FrIday

>> switch lower(day_string)
    case 'monday'
     num_day = 1;
    case 'tuesday'
     num_day = 2;
    case 'wednesday'
     num_day = 3;
      case 'thursday'
     num_day = 4;
      case 'friday'
     num_day = 5;
      case 'saturday'
     num_day = 6;
      case 'sunday'
     num_day = 7;
    otherwise
     num_day = NaN;
   end
>> num_day

num_day =

     5
```

4.3 Die `for`-Schleife

Die `for`-Schleife ist eine der nützlichsten Konstrukte in MATLAB. Man beachte aber die Bemerkung zur Effizienz in Abschnitt 4.5.

Die Form des `for`-Statements ist

```
>>  for Variable = Ausdruck
      Zuweisungen
    end
```

Sehr oft ist der Ausdruck ein Vektor der Form `i:s:j`, siehe Abschnitt 2.7.4. Die Zuweisungen werden ausgeführt, wobei die Variable einmal gleich jedem Element des Ausdruckes gesetzt wird. Die Summe der ersten 25 Terme der Harmonischen Reihe $1/i$ können folgendermassen berechnet werden.

```
>> s=0;
>> for i=1:25, s = s + 1/i; end
>> s
```

```
s =

    3.8160
```

Hier wird die Schleife also 25 Mal durchlaufen, wobei `i` in jedem Schleifendurchlauf den Wert ändert, angefangen bei $i = 1$ bis $i = 25$.

Der Ausdruck kann eine Matrix sein,

```
>> for i=A, i, end

i =

     1
     3

i =

     2
     4
```

Das nächste Beispiel definiert eine tridiagonale Matrix mit 2 auf der Diagonale und -1 auf den Nebendiagonalen.

```
>> n = 4;
>> for i=1:n,
      for j=1:n,
         if abs(i-j) > 1,
            t(i,j) = 0;
         elseif i == j,
            t(i,j) = 2;
         else
            t(i,j) = -1;
         end
      end
   end
>> t

t =

     2    -1     0     0
    -1     2    -1     0
     0    -1     2    -1
     0     0    -1     2
```

4.4 Die `while`-Schleife

Die allgemeine Form des `while`-Statements ist

```
>>  while Relation
       Zuweisungen
```

```
    end
```

Die Zuweisungen werden solange ausgeführt wie die Relation wahr ist.

```
>> e = 1; j = 0;
>> while 1+e>1, j = j + 1; e = e/2; end
>> format long
>> j = j - 1, e = 2*e

j =

    52

e =

     2.220446049250313e-16

>> eps     %  Zum Vergleich: die permanente Variable  eps

eps =

     2.220446049250313e-16
```

In MATLAB ist eine Relation natürlich eine Matrix. Wenn diese Matrix mehr als ein Element hat, so werden die Statements im `while`-Körper genau dann ausgeführt, wenn jede einzelne Komponente der Matrix den Wert 'wahr' (d.h. 1) hat.

4.5 Eine Bemerkung zur Effizienz

In MATLAB werden die Befehle *interpretiert*. D.h., eine Zeile nach der anderen wird gelesen und augeführt. Dies geschieht z.B. auch in Schleifen. Es ist deshalb von Vorteil, zumindest bei zeitaufwendigeren Programmen vektorisiert zu progammieren.

```
>> x=[0:pi/100:10*pi];
>> y=zeros(size(x));
>> tic, for i=1:length(x), y(i)=sin(x(i)); end, toc

elapsed_time =

    0.0161

>> tic, y=sin(x); toc

elapsed_time =

   9.7100e-04

>> 0.0161 / 9.7100e-04
```

```
ans =

   16.5808

>>
```

4.6 Aufgaben

1. Konstruieren Sie unter Verwendung von `for`-Schleife und `if`-Verzweigung die Matrix
$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 6 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 12 \\ 0 & 0 & 0 & 13 & 14 \\ 0 & 0 & 0 & 0 & 15 \end{bmatrix}.$$

2. Berechnen Sie den Wert des Kettenbruchs
$$1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1+1}}}}}}}}}}$$
für variable Länge des Bruchs. Um den Kettenbruch auszuwerten, müssen Sie von unten nach oben auswerten. Das Resultat muss den goldenen Schnitt $(1 + \sqrt{5})/2$ geben.

3. *Collatz-Iteration.* Ausgehend von einer *natürlichen* Zahl n_1 berechnet man eine Folge von natürlichen Zahlen nach dem Gesetz $n_{k+1} = f(n_k)$ wobei
$$f(n) = \begin{cases} 3n+1, & \text{falls } n \text{ ungerade} \\ n/2, & \text{falls } n \text{ gerade} \end{cases}$$
Collatz vermutete, dass diese Folge immer zum Wert 1 führt. Bewiesen ist diese Vermutung aber nicht. Schreiben Sie eine `while`-Schleife, die ausgehend von einem vorgegebenen n_0 eine Collatz-Folge berechnet. Wenn Sie wollen, können alle Zahlen bis zur Eins speichern und danach die Folge plotten [11]. Zur Bestimmung, ob eine Zahl gerade oder ungerade ist, können Sie (wenn Sie wollen) eine der Funktionen `rem` oder `mod` verwenden.

   Bemerkung: Sie können sich die Aufgabe erleichtern, wenn Sie zuerst Abschnitt 5.1 über Scriptfiles lesen.

4. *Tschebyscheff-Polynome.* Tschebyscheff-Polynome sind rekursiv definiert:
$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), \qquad T_0(x) = 1, \quad T_1(x) = x.$$

Tschebischeff-Polynome oszillieren im Interval $[-1, 1]$ zwischen den Werten -1 und 1. Berechnen Sie einige der Polynome in diesem Interval uns plotten Sie sie, siehe Fig. 4.1. Benützen Sie dabei eine `for`-Schleife. Setzen Sie

```
x=linspace(-1,1,101)';
```

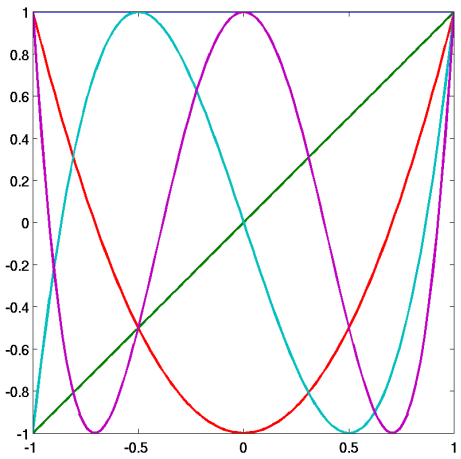

Abbildung 4.1: Tschebyscheff-Polynome $T_0(x)$ bis $T_5(x)$

Kapitel 5

M-Files

Obwohl man viele nützliche Arbeit im MATLAB-Befehlsfenster ausführen kann, wird man früher oder später M-Files schreiben wollen oder müssen. M-Files entsprechen Programmen, Funktionen, Subroutinen oder Prozeduren in anderen Programmiersprachen. M-Files bestehen zu einem grossen Teil aus einer Folge von MATLAB-Befehlen. (Sie heissen M-Files, da ihr Filetyp d.h. der letzte Teil ihres Filenamens '.m' ist.)

M-Files eröffnen eine Reihe von Möglichkeiten, die das Befehlsfenster nicht bietet:

- Experimentieren mit einem Algorithmus indem ein File editiert wird, anstatt eine lange Befehlssequenz wieder und wieder eintippen.
- Eine permanente Aufzeichnung eines Experiments machen.
- Der Aufbau von Dienstprogrammen zur späteren Benützung.
- Austausch von Programmen mit Kollegen. Viele M-Files sind von MATLAB-Enthusiasten geschrieben worden und ins Internet gestellt worden.

Es gibt zwei Arten von M-Files, *Scriptfiles* und *Funktionenfiles.*

5.1 Scriptfiles

Scriptfiles werden einfach durchlaufen und die darin enthaltenen Befehle so ausgeführt (interpretiert) wie wenn sie am Bildschirm eingetippt worden wären. Sie haben keine Eingabe oder Ausgabeparameter. Scriptfiles werden benützt, wenn man lange Befehlsfolgen z.B. zum Definieren von grossen Matrizen hat oder wenn man oft gemachte Befehlsfolgen nicht immer wieder eintippen will. Der Gebrauch von Scriptfiles ermöglicht es, Eingaben im wesentlichen mit dem Editor[1] anstatt im MATLAB-Befehlsfenster zu machen.

Der folgende MATLAB-Auszug zeigt zwei Beispiele von M-Files. Mit dem Befehl `what` kann man die M-Files im gegenwärtigen Directory auflisten lassen. Der Befehl `type` listet den Inhalt eines Files aus.

[1] MATLAB hat einen eingebauten Editor, den man via Fensterbalken oder dem Befehl `edit filename` oder nur `edit` aufrufen kann.

```
>> type init

A = [  7     3     4   -11    -9    -2 ;
      -6     4    -5     7     1    12 ;
      -1    -9     2     2     9     1 ;
      -8     0    -1     5     0     8 ;
      -4     3    -5     7     2    10 ;
       6     1     4   -11    -7    -1 ]

>> type init_demo

   %  Script-File fuer Matlab-Kurs

   disp(' Erzeugt 2 Zufallszahlenvektoren der Laenge n')

   n = input(' n = ? ');

   rand('state',0);
   x = rand(n,1);
   y = rand(n,1);

>> init_demo
 Erzeugt 2 Zufallszahlenvektoren der Laenge n

  n = ? 7
>> who

Your variables are:

n         x         y

>> [x,y]'

ans =

  0.9501  0.2311  0.6068  0.4860  0.8913  0.7621  0.4565
  0.0185  0.8214  0.4447  0.6154  0.7919  0.9218  0.7382
```

Bemerkung: Im Scriptfile `startup.m` im MATLAB-Heimverzeichnis kann man Befehle eingeben, die MATLAB am Anfang ausführen soll. Hier kann man z.B. angeben, in welchem Directory MATLAB starten soll, oder den Verzeichnis-Pfad erweitern (`addpath`).

5.2 Funktionen-Files

Funktionen-Files erkennt man daran, dass die erste Zeile des M-Files das Wort 'function' enthält. Funktionen sind M-Files mit Eingabe- und Ausgabeparameter. Der Name des M-Files und der Funktion sollten gleich sein. Funktionen arbeiten mit eigenem Speicherbereich, unabhängig vom Arbeitsspeicher, der vom

MATLAB-Befehlsfenster sichtbar ist. Im File definierte Variablen sind *lokal*. Die Parameter werden *by address* übergeben. Lokale Variable wie Parameter gehen beim Rücksprung aus der Funktion verloren.

Als erstes Beispiel eines Funktionen-Files habe ich das obige Script-File `init_demo.m` in eine Funktionen-File `init1` umgeschrieben. Dadurch wird es viel einfacher zu gebrauchen.

```
function  [x,y] = init1(n)
%INIT1           [x,y] = INIT(n) definiert zwei
%                Zufallszahlenvektoren der Laenge n

%  Demo-File fuer Matlab-Kurs

rand('state',0);
x = rand(n,1);
y = rand(n,1);
```

Neben dem Stichwort `function` erscheinen auf der ersten Zeile des Files die Ausgabeparameter (`x`, `y`) und die Eingabeparameter. MATLAB-Funktionen können mit einer variablen Anzahl von Parametern aufgerufen werden.

```
>> norm(A)
ans =
   16.8481
>> norm(A,'inf')
ans =
    24
```

Wenn das zweite Argument in `norm` fehlt, berechnet die Funktion einen Defaultwert. Innerhalb einer Funktion stehen zwei Grössen, `nargin` und `nargout` zur Verfügung, die die Zahl der beim aktuellen Aufruf verwendeten Parameter angibt. Die Funktion `norm` braucht `nargin` aber nicht `nargout`, da sie immer nur einen Wert liefert.

Bemerkung: MATLAB sucht beim Eintippen nicht das `init1` der ersten Zeile des Funktionen-Files, sondern das File mit Name `init1.m`! Der auf der ersten Zeile angegebene Funktionenname ist unwesentlich! Wenn in MATLAB ein Name z.B. `xyz` eingegeben wird, so sucht der MATLAB-Interpreter

1. ob eine Variable `xyz` im Arbeitsspeicher existiert,
2. ob `xyz` eine eingebaute MATLAB-Funktion ist,
3. ob ein File namens `xyz.m` im gegenwärtigen Verzeichnis (Directory) existiert,
4. ob es ein File namens `xyz.m` im einem der in der Unix-Umgebungsvariable MATLABPATH eingetragenen Verzeichnisse gibt.

Ein eigenes Funktionen-File wird wie eine eingebaute Funktion behandelt.

Wird in MATLAB der Befehl `help init1` eingetippt, werden die im File abgespeicherte Kommentarzeilen bis zur ersten Nichtkommentarzeile auf den Bildschirm geschrieben.

```
>> help init1

 INIT1               [x,y] = INIT(n) definiert zwei
                     Zufallszahlenvektoren der Laenge n

>> init1(4)

ans =

    0.9501
    0.2311
    0.6068
    0.4860

>> [a,b]=init1(4)

a =

    0.9501
    0.2311
    0.6068
    0.4860

b =

    0.8913
    0.7621
    0.4565
    0.0185
```

Das zweite Beispiel zeigt ein M-File, welches $n!$ rekursive berechnet. Wenn die Eingabe nicht korrekt ist, wird die MATLAB-Funktion 'error' aufgerufen, die eine Fehlermeldung auf den Bildschirm schreibt und dann die Ausführung abbricht.

```
>> type fak

function y=fak(n)
%
%FAK     fak(n) berechnet die Fakultaet von  n.
%
%            fak(n) = n * fak(n-1),  fak(0) = 1

%      P. Arbenz   27.10.89

if n < 0 | fix(n) ~= n,
   error(['FAK ist nur fuer nicht-negative',...
          ' ganze Zahlen definiert!'])
end

if n <= 1,
   y = 1;
```

```
else
   y = n*fak(n-1);
end

>> fak(4)

ans =

    24

>> fak(4.5)
??? Error using ==> fak
FAK ist nur fuer nicht-negative ganze Zahlen definiert!
```

In diesem Beispiel sind die Variablen n und y lokal. Sie werden durch `who` nicht aufgelistet, wenn sie nicht schon vor dem Aufruf von `fak` definiert wurden. In letzterem Fall stimmen ihre Werte im Allg. nicht mit den in der Funktion zugewiesenen Werten überein.

5.3 Arten von Funktionen

In MATLAB gibt es verschiedene Arten Funktionen, auch solche, die kein eigenes File haben.

5.3.1 Anonyme Funktionen

Eine anonyme Funktion besteht aus einem einzigen MATLAB-Ausdruck hat aber beliebig viele Ein-und Ausgabeparameter. Anonyme Funktionen können auf der MATLAB-Kommandozeile definiert werden. Sie erlauben, schnell einfache Funktioen zu definieren, ohne ein File zu editieren.

Die Syntax ist

```
f = @(arglist)expression
```

Der Befehl weiter unten kreiert eine anonyme Funktion, die das Quadrat einer Zahl berechnet. Wenn die Funktion aufgerufen wird, weist MATLAB der Variable `x` zu. Diese Variable wird dann in der Gleichung `x.^2` verwendet.

```
>> sqr = @(x) x.^2

sqr =

    @(x) x.^2

>> a = sqr(7)

a =

    49

>> clear i
```

```
>> sqr(i)

ans =

    -1

>> % rest: ganzzahliger Teiler und Rest
>> rest=@(x,y) [floor(x/y), x-y*floor(x/y)]

rest =

    @(x,y) [floor(x/y), x-y*floor(x/y)]

>> h=rest(8,3)

h =

     2     2

>> 8 - (h(1)*3 + h(2))

ans =

     0
```

5.3.2 Primäre und Subfunktionen

Alle Funktionen, die nicht anonym sind, müssen in M-Files definiert werden. Jedes M-File muss eine primäre Funktion enthalten, die auf der ersten Zeile des Files definiert ist. Weitere Subfunktionen können dieser primären Funktion folgen. Primäre Funktionen haben einen witeren Definitionsbereich (*scope*) als Subfunktionen. Primäre Funktionen können von ausserhalb des M-Files aufgerufen werden, z.B. von der MATLAB-Kommandozeile oder aus einer anderen Funktion. Subfunktionen sind nur in dem File sichtbar, in welchem sie definiert sind.

5.3.3 Globale Variable

Wenn mehr als eine Funktion auf eine einzige Variable Zugriff haben soll, so muss die Variable in allen fraglichen Funktionen als `global` deklariert werden. Wenn die Variable auch vom Basis-Arbeitsspeicher zugegriffen werden soll, führt man das `global`-Statement auf der Kommandozeile aus. Eine Variable muss als `global` deklariert werden *bevor* sie das erste Mal gebraucht wird.

```
>> type myfun

function f = myfun(x)
%MYFUN  myfun(x) = 1/(A + (x-B)^2)
%       A, B are global Variables
```

```
  global A B
  f  = 1/(A + (x-B)^2);

>> global A B
>> A = 0.01; B=0.5;
>> fplot(@myfun,[0 1])
```

Da MATLAB Variablen, insbesondere Matrizen, lokal abspeichert, kann der Speicherbedarf bei rekursiven Funktionsaufrufen sehr gross werden. In solchen Fällen müssen Variablen als globale erklärt werden, um Speicherplatzüberlauf zu verhindern.

5.3.4 Funktionenfunktionen

Eine Klasse von Funktionen, sog. Funktionenfunktionen, arbeitet mit nicht-linearen Funktionen einer skalaren Variable. Eine Funktion arbeitet mit einer anderen. Dazu gehören

- Nullstellensuche
- Optimierung (Minimierung/Maximierung)
- Integration (Quadratur)
- Gewöhnliche Differentialgleichungen (ODE)

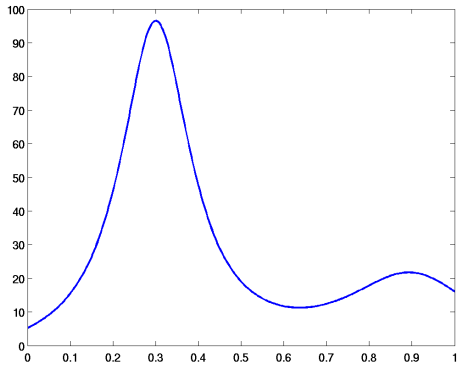

Abbildung 5.1: Graph der Funktion in `humps.m`

In MATLAB wird die nicht-lineare Funktion in einem M-File gespeichert. In der Funktion `humps` ist die Funktion programmiert

$$y(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6,$$

die ausgeprägte Maxima bei $x = 0.3$ und $x = 0.9$ hat, siehe Abb. 5.1. Aus dieser Abbildung sieht man, dass die Funktion bei $x = 0.6$ ein Minimum hat.

```
>> p = fminsearch(@humps,.5)

p =

    0.6370

>> humps(p)

ans =

   11.2528
```

Das bestimmte Integral

$$\int_0^1 \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6\; dx$$

nähert man numerisch an durch die eingebaute MATLAB-Funktion `quadl` an

```
>> Q = quadl(@humps,0,1)

Q =

   29.8583
```

Schliesslich, kann man eine Nullstelle bestimmen durch

```
>> z = fzero(@humps,.5)

z =

   -0.1316
```

Die Funktion hat keine Nullstelle im abgebildeten Interval. MATLAB findet aber eine links davon.

5.3.5 Funktionen mit variabler Zahl von Argumenten

Es gibt Situationen, in denen man die Zahl der Ein- oder Ausgabe-Parameter variabel haben will. Nehmen wir an, wir wollen die direkte Summe einer unbestimmten Anzahl von Matrizen bilden:

$$A_1 \oplus A_2 \oplus \cdots \oplus A_m = \begin{pmatrix} A_1 & O & \cdots & O \\ O & A_2 & \cdots & O \\ \vdots & \vdots & \ddots & \vdots \\ O & O & \cdots & A_m \end{pmatrix}$$

Wenn nur zwei Matrizen da wären, wäre die Lösung naheliegend.

```
function C = dirsum2(A,B)
%DIRSUM2 Direct sum of two matrices.
%
%     C = dirsum (A,B):   C = [A 0]
```

```
%                               [0 B]

% Peter Arbenz, Sep 8, 1989.

[n,m] = size(A);
[o,p] = size(B);
C = [A zeros(n,p); zeros(o,m) B];
```

Das Benützen einer variable Anzahl von Argumenten unterstützt MATLAB durch die Funktionen `varargin` und `varargout` an. Diese kann man sich als eindimensionale Arrays vorstellen, deren Elemente die Eingabe-/Ausgabeargumente sind. In MATLAB werden diese Arrays, deren Elemente verschiedene Typen haben können cell arrays genannt. Hier müssen wir nur wissen, dass die Elemente eines cell arrays mit geschweiften statt mit runden Klammern ausgewählt werden. Die Lösung unseres kleinen Problems könnte somit so aussehen.

```
function A = dirsum(A,varargin)
%DIRSUM Direct sum of matrices
%       C = dirsum(A1,A2,...,An)

% Peter Arbenz, May 30, 1997.

for k=1:length(varargin)
   [n,m] = size(A);
   [o,p] = size(varargin{k});
   A = [A zeros(n,p); zeros(o,m) varargin{k}];
end
```

Hier wird das erste Argument als `A` bezeichnet, alle weiteren werden mit der Funktion `varargin` behandelt.

```
>> a=[1 2 3]; b=ones(3,2);
>> dirsum(a,b,17)

ans =

     1     2     3     0     0     0
     0     0     0     1     1     0
     0     0     0     1     1     0
     0     0     0     1     1     0
     0     0     0     0     0    17
```

5.3.6 Aufgaben

1. Schreiben Sie eine Funktion, die die ersten n Tschebyscheff-Polynome an den Stützstellen gegeben durch den Vektor **x** auswertet, siehe Seite 41. Die erste Zeile des Funktionenfiles soll die Form

   ```
   function T = tscheby(x,n)
   ```

   haben.

2. *Wurzelberechnung.* Das Newton-Verfahren zur Berechnung der Wurzel einer positiven Zahl a ist gegeben durch

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right).$$

Schreiben Sie eine eigene Funktion

`[x, iter] = wurzel(a, tau)`,

die $\sqrt{a}$ auf eine gewisse Genauigkeit τ berechnet:

$$|x_{k+1} - x_k| \leq \tau.$$

Wenn nur ein Eingabeparameter vorgegeben wird, so soll $\tau = \varepsilon =$ `eps` gesetzt werden.

Sehen Sie auch einen Notausgang vor, wenn Folge $\{x_k\}$ nicht (genügend schnell) konvergiert.

3. Lösen Sie die gewöhnliche Differentialgleichung

$$\frac{dy}{dt} = -2ty(t) + 4t, \qquad y(0) = 0. \tag{5.1}$$

Die analytische Lösung ist

$$y(t) = ce^{-t^2} + 2, \qquad c = y(0) - 2.$$

Wir wollen die Funktion `ode23` zur Lösung dieser Differentialgleichung verwenden. `ode23` geht von einer Differentialgleichung der Form

$$y'(t) = f(t, y(t)) \qquad \text{plus Anfangsbedingungen}$$

aus. Deshalb schreiben Sie zunächst eine Funktion, z.B. `fun_ex1.m`, die *zwei* Eingabeargumente, t und $y(t)$, und ein Ausgabeargument, $f(t, y)$, hat. Danach lösen Sie die Gleichung (5.1) im Interval $[0, 3]$. `ode23` hat zwei Ausgabeparameter, Vektoren (`t`, `y`) der selben Länge, die die (approximativen) Werte der Lösung $y(t)$ and gewissen Stellen t enthält. Die Lösung kann so leicht geplottet werden.

4. *Räuber-Beute-Modell.* Im Räuber-Beute-Modell von Lotka–Volterra geht man von zwei Populationen aus, einer Beutepopulation y_1 und einer Räuberpopulation y_2. Wenn ein Räuber ein Beutetier trifft frisst er es mit einer bestimmten Wahrscheinlichkeit c. Wenn die Beutepopulation sich alleine überlassen wird, so wächst sie mit einer Rate a. (Das Nahrungsangebot ist unendlich gross.) Die Räuber haben als einzige Nahrung die Beutetiere. Wenn diese nicht mehr vorhanden sind, so sterben auch die Räuber (mit einer Rate b) aus.

Hier betrachten wir das Beispiel von Hasen und Füchsen, deren Bestände durch $y_1(t)$ und $y_2(t)$ bezeichnet seien. Das Räuber-Beute-Modell von Lotka-Voltarra hat die Form

$$\begin{aligned} \frac{dy_1}{dt}(t) &= ay_1 - cy_1y_2, & y_1(0) &= y_1^{(0)}, \\ \frac{dy_2}{dt}(t) &= -by_2 + dy_1y_2, & y_2(0) &= y_2^{(0)}. \end{aligned} \tag{5.2}$$

Alle Koeffizienten a, b, c und d sind positiv. Man rechen das Modell mit folgenden Parametern durch:

- Zuwachsrate Hasen: $a = 0.08$,
- Sterberate Füchse: $b = 0.2$,
- Wahrscheinlichkeit, bei Treffen gefressen zu werden: $c = 0.002$,
- Beutewahrscheinlichkeit der Füchse: $d = 0.0004$,
- Anfangsbedingungen: Anfangsbestand Hasen 500, Anfangsbestand Füchse 20.

Die Simulation ergibt eine periodische Schwingung, wobei die Maxima der Räubertiere jeweils eine kurze Weile nach den Maxima der Beutetiere auftreten.

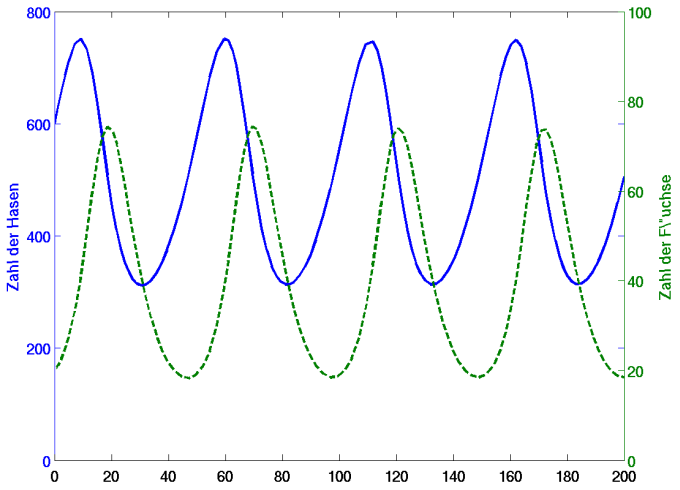

Abbildung 5.2: Simulation eines Räuber-Beute-Modell

Verwenden Sie `ode34` und lösen Sie die Differentialgleichung im Interval $[0, 200]$. Das M-File, das Sie schreiben müssen, sieht wie in der vorigen Aufgabe aus, hat aber einen 2-komponentigen Vektor als Ausgabeargument. Zur Darstellung der beiden Lösungskomponenten können Sie den Befehl `plotyy` verwenden.

5. Die nicht-lineare gewöhnliche Differentialgleichung dritter Ordnung

$$f'''(t) + f''(t)f(t) = 0, \qquad f(0) = f'(0) = 0, \lim_{t\to\infty} f'(t) = 1 \tag{5.3}$$

hat keine analytische Lösung. Da MATLABs ODE-Löser Anfangswertprobleme erster Ordnung lösen, müssen wir zunächst (5.3) in ein solches um-

schreiben. Wir setzen

$$
\begin{aligned}
y_1(t) &= f(t) \\
y_2(t) &= \frac{dy_1(t)}{dt} = f'(t), \\
y_3(t) &= \frac{dy_2(t)}{dt} = \frac{d^2y_1(t)}{dt^2} = f''(t), \\
\frac{dy_3(t)}{dt} &= f'''(t).
\end{aligned}
$$

Somit erhält die ursprüngliche gewöhnliche Differentialgleichung dritter Ordnung die Form

$$
\begin{aligned}
\frac{dy_1(t)}{dt} &= y_2(t), \\
\frac{dy_2(t)}{dt} &= y_3(t), \\
\frac{dy_3(t)}{dt} &= -y_1(t)y_3(t).
\end{aligned}
$$

Die Anfangsbedingungen für y_1 und y_2 sind bekannt: $y_1(0) = 0$ und $y_2(0) = 0$. Wir haben keine Anfangsbedingung für y_3. Wir wissen aber, dass $y_2(t)$ mit $t \to \infty$ gegen 1 konvergiert. Deshalb kann die Anfangsbedingung für y_3 verwendet werden, um $y_2(\infty) = 1$ zu erhalten. Durch versuchen erhält man $y_3(0) \approx 0.469599$. Die Lösung sieht wie in Abb. 5.3 aus.

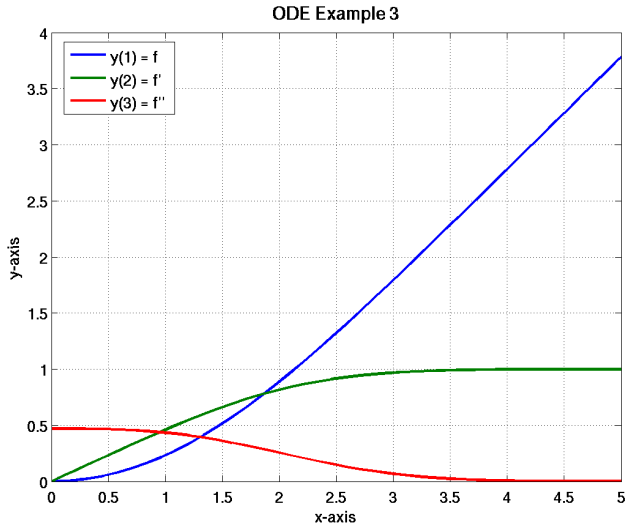

Abbildung 5.3: ODE 3-ter Ordnung

6. Versuchen Sie die Konstante in der Anfangsbedingung für y_3 zu bestimmen. Man muss dabei nicht bis $t = \infty$ gehen; $t = 6$ reicht. Versuchen Sie

also mit `fzero` eine Nullstelle der Funktion

$$g(a) = y_2(6, a) - 1 = 0$$

zu berechnen. Hier bedeutet $y_3(6, a)$ der Funktionswert von y_3 an der Stelle $t = 6$ wenn die Anfangsbedingung $y_3(0) = a$ gewählt wurde.

Kapitel 6

Graphik in Matlab

MATLAB hat Möglichkeiten Linien und Flächen graphisch darzustellen. Es ist auch möglich, graphische Benützeroberflächen oder 'Filme' herzustellen. Für eine eingehende Behandlung sei auf [12] verwiesen.

In dieser Einführung wollen wir uns auf das Wesentliche beschränken. Es soll nicht alle möglichen graphischen Darstellungen behandelt werden. Es sollte aber verständlich werden, wie das Graphiksystem aufgebaut ist.

6.1 Darstellung von Linien

MATLAB erlaubt das Plotten von Linien in 2 und 3 DimensionenEine erste Übersicht erhält man, wenn man mit `demo` im Befehlsfenster das MATLAB-Demosfenster eröffnet und dort unter 'MATLAB Visualization' die '2-D Plots' laufen lässt.

Linien in zwei Dimensionen kann man auf verschiedene Arten darstellen. Hier einige Beispiele.

```
>> x=[1:.5:4]';
>> y1=5*x; y2=8*x.^2;
>> plot(x,y1)
>> plot(x,[y1 y2])
>> plot(x,y1,x,y2)
>> bar(x,[y1 y2])
>> stairs(x,[y1 y2])
```

Der erste Plot-Befehl, hat zwei Vektoren der gleichen Länge, x und y, als Eingabeparameter. Der erste Vektor wird für die Abszisse (x-Achse) der zweite für die Ordinate (y-Achse). Dabei werden die vorgegebenen Koordinatenpaare interpoliert.

Eine Legende kann beigefügt werden:

```
>> plot(x,y1,x,y2)
>> legend('erste Linie','zweite Linie')
```

Die Legende kann mit der Maus verschoben werden (linke Maustaste drücken) und mit `legend off` wieder entfernt werden. Ähnlich in drei Dimensionen

```
>> z=[0:.1:20]';
>> x=sin(z);
>> y=cos(z);
>> plot3(x,y,z)
```

Sowohl Farbe wie Art der Darstellung der einzelnen Linien kann bestimmt werden, indem nach den x-, y- und (eventuell) z-Werten eine entsprechende Zeichenkette angefügt wird.

```
>> plot3(x,y,z,'g')
>> plot3(x,y,z,'g:')
>> plot3(x,y,z,'rv')
>> hold on                % Plot wird nun ueberschrieben
>> plot3(x,y,z,'g')
>> hold off
>> plot3(x,y,z,'rv',x,y,z,'g')
```

Einige mögliche Werte können folgender Zusammenstellung entnommen werden.

`y`	yellow	`.`	point	`>`	triangle (right)
`m`	magenta	`o`	circle	`-`	solid
`c`	cyan	`x`	x-mark	`:`	dotted
`r`	red	`+`	plus	`-.`	dashdot
`g`	green	`*`	star	`--`	dashed
`b`	blue	`v`	triangle (down)		
`w`	white	`^`	triangle (up)		
`k`	black	`<`	triangle (left)		

Eine Übersicht über alle Möglichkeiten der Kurvendarstellung findet man unter `doc plot`. Es können leicht Beschriftungen angebracht werden.

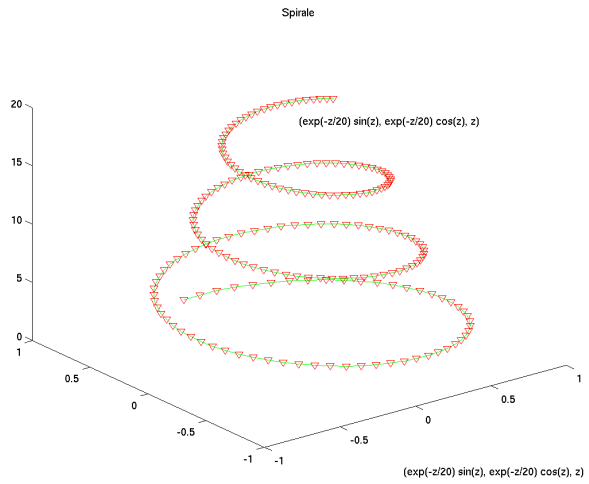

Abbildung 6.1: Beispiel eines Linien-Plot

```
>> title('Spirale')
>> xlabel(' (sin(z), cos(z), z) ' )
>> text(0,0,22,' (sin(z), cos(z), z) ' )
```

Am Ende dieser Befehlssequenz hat das Graphikfenster den in Figur 6.1 gezeigten Inhalt.

6.1.1 Aufgaben

1. Führen Sie den Befehl `plot(fft(eye(17)))` aus. Verschönern Sie die Darstellung wieder mit dem Befehl `axis`: Beide Achsen sollen gleich lang sein, da die gezeichneten und durch Linien verbundenen Punkte auf dem Einheitskreis liegen. Stellen Sie die Achsen überhaupt ab.

2. Plotten Sie die Funktion

$$\frac{1}{(x-1)^2} + \frac{3}{(x-2)^2}$$

   auf dem Intervall $[0, 3]$. Verwenden Sie die MATLAB-Funktion `linspace`, um die x-Koordinaten zu definieren.

   Da die Funktion Pole im vorgegebenen Interval hat, wählt MATLAB einen ungünstigen Bereich für die y-Achse. Verwenden Sie den Befehl `axis` oder `ylim`, um die obere Grenzen für die y-Achse auf 50 zu setzen.

   Verwenden Sie den Graphik-Editor, um die Achsen-Beschriftung grösser und die Linie dicker zu machen. Beschriften Sie die Achsen.

6.2 Das Matlab-Graphiksystem

`plot`, `bar`, etc., sind 'high-level' MATLAB-Befehle. MATLAB hat aber auch low-level Funktionen, welche es erlauben, Linien, Flächen und andere Graphik-Objekte zu kreieren oder zu ändern. Dieses System heisst *Handle Graphics*. Seine Bausteine sind hierarchisch strukturierte Graphik-Objekte.

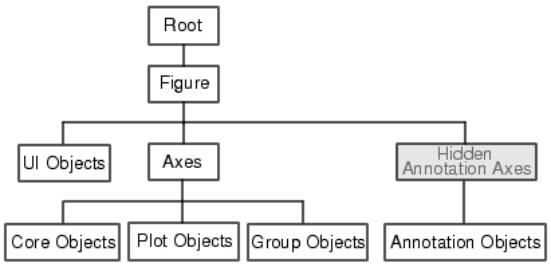

Abbildung 6.2: Objekt Hierarchie [15]

- An der Wurzel des Baums ist der Bildschirm (*root-Objekt*).
- *Figure-Objekte* sind die individuellen Fenster. Sie sind Kinder von `root`. Alle anderen Graphik-Objekte stammen von Figure-Objekten ab. Alle Objekte kreierenden Funktionen wie z.B. die höheren Graphikbefehle eröffnen

eine `figure`, falls noch keine existiert. Ein Figure-Objekt kann auch mit `figure` gestartet werden.

- *Axes-Objekte* definieren Bereiche in einem `figure`-Fenster und orientieren ihre Kinder im Gebiet. Axes-Objekte sind Kinder von Figure-Objekten und Eltern von Lines-, Surfaces-, Text- und anderen Objekten.

Wenn Sie den Graphik-Editor aufrufen (`Edit` im Menu-Balken des Graphik-Fensters), dann können Sie die wichtigsten Eigenschaften der verschiedenen Objekte leicht ändern. Wenn Sie im Property Editor auf `Inspector` klicken, erhalten Sie die gesamte Liste der Eigenschaften eines Objekts. Zu beachten ist, dass diese Änderungen verloren gehen, sobald Sie einen neuen `plot`-Befehl von der Kommandozeile absetzten.

Jedes Graphik-Objekt kann mit einem sog. Handle auch von der Kommandozeile oder von einem M-File zugegriffen werden. Der Handle wird bei der Erzeugung des Objekts definiert. Befehle wie `plot`, `title`, `xlabel`, und ähnliche habe als Ausgabewerte einen oder mehrere (`plot`) handles. Wir wollen zunächst mit dem Befehl `gcf` (get current figure) das gegenwärtige (einzige) Fenster ergreifen:

```
>> gcf
>> figure
>> gcf
>> close(2)
>> gcf
```

Mit dem Befehl `gca` (get handle to current axis) können wir die gegenwärtigen Koordinatenaxen mit Zubehör (z.B. Linien) ergreifen.

```
>> gca
```

Wir sehen uns nun die Eigenschaften dieses Objekts an:

```
>> get(gca)
```

Achtung: Der Output ist lang! Diese Information kann man aber auch via 'property editor' im File-Menu auf dem Rahmen der Figur erhalten.

Wir wollen uns einige der (alphabetisch angeordneten) Attribute genauer ansehen. Einige Attribute enthalten Werte, andere sind wiederum handles.

```
>> th = get(gca,'title')
>> get(th)
```

ordnet `th` den handle zum Title zu. Mit dem Befehl `set` können Werte von Attributen geändert werden. Z.B.

```
>> set(th,'String','S P I R A L E')
```

Die wesentlichen Attribute von axis, z.B. `XTick`, `XTickLabel`, sind selbsterklärend:

```
>> get(gca,'XTick')
>> get(gca,'XTickLabel')
```

Wir können die Figur mit etwas unsinnigem beschriften:

```
>> set(gca,'XTickLabel',['a';'b';'c';'d';'e'])
```

6.2.1 Aufgaben

1. Führen Sie zunächst folgende Befehle aus

   ```
   z=[0:.1:20]';
   x=sin(z);
   y=cos(z);
   h=plot3(x,y,z,'rv',x,y,z,'g')
   ```

   Dann machen sie aus den Dreiecken der ersten Kurve Kreise und wechseln die Farbe der zweiten Kurve zu blau.

2. Führen Sie das folgende script aus.

   ```
   x = [1:.5:16]';
   y = x.^2+2*(rand(size(x)).*x-.5);
   plot(x,y,'o')
   ```

   (a) Berechnen Sie via Methode der kleinsten Quadrate das beste quadratische Polynom durch die Punkte. Zeichnen Sie beide Kurven uebereinander und beschriften Sie sie (`legend`).

   (b) Das gleiche in doppelt-logarithmischer Skala. Dabei sollen die beiden Achsen mit 1,2,4,8,16 beschriftet werden. Hier kann man den Befehl `loglog` verwenden.

6.3 Darstellung von Flächen

MATLAB kennt verschiedene Möglichkeiten, um 3D Objekte graphisch darzustellen. Den Befehl `plot3` haben wir schon kennen gelernt.

Mir der Funktion `mesh` können 3D-Maschenflächen geplottet werden. Zu diesem Zweck ist es von Vorteil, zunächst eine weitere Funktion `meshgrid` einzuführen. Diese Funktion generiert aus zwei Vektoren x (mit n Elementen) und y (mit m Elementen) ein Rechtecksgitter mit $n \times m$ Gitterpunkten.

```
>> x = [0 1 2];
>> y = [10 12 14];
>> [xi yi] = meshgrid(x,y)

xi =

     0     1     2
     0     1     2
     0     1     2


yi =

    10    10    10
    12    12    12
    14    14    14
```

Der Befehl `meshgrid` angewandt auf die beiden Arrays x und y erzeugen zwei Matrizen, in welchen die x- und y-Werte repliziert sind. So erhält man die Koordinaten aller Gitterpunkte gebildet mit den x_i und y_j.

Wir können jetzt z.B. die Sattelfläche $z = x^2 - y^2$ über dem Quadrat $[-1, 1] \times [-1, 1]$ zeichnen.

```
x = -1:0.05:1;
y = x;
[xi, yi] = meshgrid(x,y);
zi = yi.^2 - xi.^2;
mesh(xi, yi, zi)
axis off
```

Mit

```
meshc(xi, yi, zi)
axis off
```

erhält man dasselbe noch mit einem Kontour-Plot.

Die Maschenlinien werden als Flächen dargestellt, wenn man `mesh(c)` durch `surf(c)` ersetzt. Die Figur 6.3 ist erhalten worden durch die Befehle

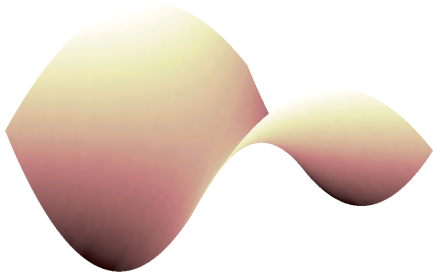

Abbildung 6.3: Die Sattelfläche dargestellt mit `surf`

```
x = -1:.05:1;  y = x;
[xi,yi] = meshgrid(x,y);
zi = yi.^2 - xi.^2;
surf(xi, yi, zi)
axis off
colormap pink
shading interp     % Interpolated shading
```

Der Befehl `colormap` wird verwendet um der Fläche eine bestimmte Farbtönung zu geben. MATLAB stellt folgende Farbpaletten zur Verfügung:

hsv	Hue-saturation-value color map (default)
hot	Black-red-yellow-white color map
gray	Linear gray-scale color map
bone	Gray-scale with tinge of blue color map
copper	Linear copper-tone color map
pink	Pastel shades of pink color map
white	All white color map
flag	Alternating red, white, blue, and black color map
lines	Color map with the line colors
colorcube	Enhanced color-cube color map
vga	Windows colormap for 16 colors
jet	Variant of HSV
prism	Prism color map
cool	Shades of cyan and magenta color map
autumn	Shades of red and yellow color map
spring	Shades of magenta and yellow color map
winter	Shades of blue and green color map
summer	Shades of green and yellow color map

Eine Farbpalette ist eine 3-spaltige Matrix mit Elementen zwischen 0 und 1. Die drei Werte einer Zeile geben die Intensität der Farben Rot, Grün und Blau an.

Sei `m` die Länge der Palette und seien `cmin` und `cmax` zwei vorgegebene Zahlen. Wenn z.B. eine Matrix mit `pcolor` (pseudocolor (checkerboard) plot) graphisch dargestellt werden soll, so wird ein Matrixelement mit Wert `c` die Farbe der Zeile `ind` der Palette zugeordnet, wobei

$$\texttt{ind} = \begin{cases} \texttt{fix((c-cmin)/(cmax-cmin)*m)+1} & \text{falls } \texttt{cmin} \leq \texttt{c} < \texttt{cmax} \\ \texttt{m} & \texttt{c == cmax} \\ \text{unsichtbar} & \texttt{c<cmin | c>cmax | c==NaN} \end{cases}$$

Mit dem Befehl `view` könnte man den Blickwinkel ändern.

Isolinien (Konturen) erhält man mit `contour` oder 'gefüllt' mit `contourf`.

```
x = -1:.05:1;  y = x;
[xi,yi] = meshgrid(x,y);
zi = yi.^2 - xi.^2;
contourf(zi), hold on,
shading flat                % flat = piecewise constant
[c,h] = contour(zi,'k-');
clabel(c,h)                 % adds height labels to
                            % the current contour plot
title('The level curves of z = y^2 - x^2.')
ht = get(gca,'Title');
set(ht,'FontSize',12)
```

Es können ein Vektorfelder gemacht werden. Wir nehmen den Gradienten der Funktion, die in `Z` gespeichert ist, vgl. Figur 6.4.

```
>> x = [0:24]/24;
>> y = x;
>> for i=1:25
```

```
>>    for j=1:25
>>       hc = cos((x(i) + y(j) -1)*pi);
>>       hs = sin((x(i) + y(j) -1)*pi);
>>       gx(i,j) = -2*hs*hc*pi + 2*(x(i) - y(j));
>>       gy(i,j) = -2*hs*hc*pi - 2*(x(i) - y(j));
>>    end
>> end
>> quiver(x,y,gx,gy,1)
```

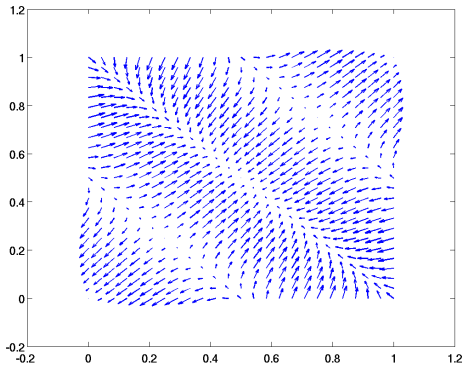

Abbildung 6.4: Das Vektorfeld **grad**$(\sin(x+y)\cos(x+y))$

6.4 Darstellung von Daten

MATLAB hat vier Funktionen, um 2- und 3-dimensionale *Balkengraphiken* zu produzieren: `bar`, `barh`, `bar3`, `bar3h`. Die 3 steht für 3-dimensionale Balken, das h für horizontal. Wir gehen hier nur auf die einfachste Art der Benützung ein. Sei eine Matrix mit m Zeilen und n Spalten vorgegeben. Mit dem Befehl `bar` werden die Zahlen in Gruppen von n Balken dargestellt. Die n Zahlen der i-ten Matrixzeilen gehören in die i-te von m Gruppen. Die folgende Befehlssequenz produziert die Graphicken in Abb. 6.5.

```
>> B=round(10*rand(6,4)+0.5)

B =

     7     9     3     2
     9     5     6     2
     7     8     6     5
     6     9     2     1
     1     5     8     9
     3     3     2     1

>> bar(2:3:17,B,'grouped')
>> bar(B,'stacked')
```

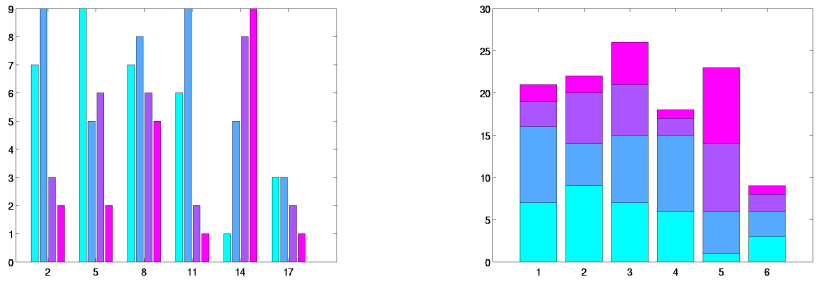

Abbildung 6.5: Darstellung einer Zahlenreihe `grouped` (links) und `stacked` (rechts).

6.4.1 Aufgaben

1. Berechnen Sie mit `rand` n im Interval $[0, 1]$ gleichverteilte Zufallszahlen, und schauen Sie, ob sie auch wirklich gleichverteilt sind. Wenden Sie den Befehl `hist` an.

2. Im Jahr 2000 war die Sprachverteilung in der Schweiz folgendermassen:

Sprache	Deutsch	Französisch	Italienisch	Rätoromanisch
Bewohner	4'640'000	1'480'000	470'000	35'000

   Stellen Sie die prozentuale Sprachverteilung durch ein Tortendiagramm dar. Dazu verwenden Sie den MATLAB-Befehl `pie`. Die Beschriftung machen Sie mit `legend`.

Kapitel 7

Symbolisches Rechnen in Matlab

Computeralgebrasysteme rechnen symbolisch, d.h. sie manipulieren mathematische Formeln nach den Regeln von Algebra, Differential- und Integralrechnung, etc. Bekannte Beispiele sind die Systeme Maple und Mathematica. In Computeralgebrasystemen sind die Werte der Variablen exakt. Daten müssen in beliebiger Genauigkeit (Länge) dargestellt werden können. Da Operationen mit beliebig langen Worten nicht Hardware-mässig implementiert sind, können sie sehr lange dauern.

Werkzeuge wie MATLAB speichern ihre Daten (Zahlen) im IEEE-Format ab. Da nur eine vorgegebene Präzision bei der Darstellung der Zahlen existiert, sind Resultate im allgemeinen nur approximativ. Operationen können sehr schnell ausgeführt werden. Allerdings können Rundungsfehler auftreten.

In MATLAB kann man aber auch symbolisch rechnen, wenn die *Symbolic Math Toolbox* installiert ist. Die Symbolic Math Toolbox ist Teil der Studentenversion von MATLAB. Sie basiert auf dem Kern von Maple V. Mit der Symbolic Math Toolbox kann man symbolische Variablen definieren, Differenzieren, Integrieren symbolische Ausdrücke umformen, und Gleichungssysteme oder Differentialgleichungen lösen.[1]

Viele Funktionen in the Math Toolbox haben dieselben Namen wie ihre numerischen Gegenstücke. MATLAB wählt diejenige aus, welche dem Typ der Eingabeparameter entspricht. Mit den Befehlen `help eig` respektive `help sym/eig` kann man Hilfe für den numerischen respektive symbnolischen Eigenlöser anzeigen.

7.0.2 Symbolische Variable

Mit dem MATLAB-Befehl `sym` kann man symbolische Variable deklarieren:

```
>> syms x y
```

[1] Auf gewissen Linux-Rechnern muss eine Umbebungsvariable gesetzt werden im Fenster in dem MATLAB gestartet wird:

```
setenv LD_ASSUME_KERNEL 2.4.1
```

```
>> syms z real
>> syms w positive
```

Die letzten beiden Deklarationen schränken den Wertebereich der Variable ein. Der Befehl `sym` wird gebraucht, um einer Variable einen symbolischen Wert zuzuweisen. (Damit wird sie eine symbolische Variable.)

```
>> s = sym('13/17+17/23')
s =
13/17+17/23
>> s = simple(sym('13/17+17/23'))
s =
588/391
```

Mit dem MATLAB-Befehl `vpa` kann die *variable precision arithmetic* angesprochen werden.

```
>> vpa('pi')
ans =
3.14159265358979323384626433832795
>> vpa('pi',50)
ans =
3.1415926535897932384626433832795028841971693993751
>> pi
ans =
3.1416e+00
>> sin(pi)
ans =
1.2246e-16
>> sin(sym('pi'))
ans =
0
```

Der zweite Parameter von `vpa` ist die Zahl der gewünschten Ziffern. Diese Zahl kann 'beliebig' gross sein. Die Rechenzeiten wachsen allerdings (mindestens) proportional zu dieser Zahl.

Weshalb will man überhaupt numerisch (mit MATLAB) rechnen, wenn symbolisches Rechnen so leistungsfähig ist? Dafür gibt es mindestens zwei Gründe:

- Die analytischen Ausdrücke können sehr kompliziert werden.

  Zum Beispiel kann die Ableitung einer Funktion wesentlich komplizierter als die Funktion selber sein. Dies bedingt einerseits eine langsame Auswertung, andererseits erhält man dadurch keine neue Einsicht.

- Die analytische Lösung existiert gar nicht.

  Dies ist typisch für Integrale (Stammfunktionen), höherdimensionale Probleme insbesondere partielle Differentialgleichungen mit unregelmässige Gebiete.

7.0.3 Differenzieren

Mit der Funktion `diff` können (partielle) Ableitungen von symbolisch definierten Funktionen berechnet werden. Letztere erhält man, indem man Funktionen mit symbolischen Variablen definiert.

```
>> syms x y
>> diff(sqrt(5*x^2 - 7*x + 4))
ans =
1/2/(5*x^2-7*x+4)^(1/2)*(10*x-7)
>> pretty(ans)
                                  10 x - 7
                        1/2 ------------------
                                 2           1/2
                             (5 x  - 7 x + 4)
>> g = x*y + x^2
g =
x*y+x^2
>> diff(g)
ans =
y+2*x
>> diff(g,x)
ans =
y+2*x
>> diff(g,y)
ans =
x
```

Der zweite Parameter von `diff` gibt an, nach welcher Variable abgeleitet werden soll. Fehlt dieser zweite Parameter, mach MATLAB eine Annahme: abgeleitet wirde nach demParameter, der am 'nächsten' bei x steht (vgl. die Funktion `findsym`).

Die Funktion `pretty` zeigt die Lösung in vielen Situationen besser lesbar an.

7.0.4 Integrieren

MATLAB kann bestimmte Integrale sowie Stammfunktionen berechnen, vorausgesetzt, dass sie existieren.

```
>>syms a b t x y z theta
>> int (sin(a*t + b))
ans =
-1/a*cos(a*t+b)
>> int (sin(a*theta + b))
ans =
-1/a*cos(a*theta+b)
>> int (sin(a*theta + b),theta)
ans =
-1/a*cos(a*theta+b)
>> int(x/(x^2+1))
ans =
1/2*log(x^2+1)
>> f=(2*x^2-1)/(x+1)^2/(x+3); pretty(f)
                                   2
                                2 x  - 1
                             ----------------
                                   2
```

```
                                (x + 1)  (x + 3)
>> int(f,x,0,1)
ans =
25/4*log(2)+1/4-17/4*log(3)
>> int(exp(-a*t)*sin(b*t),0,inf)
Definite integration: Can't determine if the integral is convergent.
Need to know the sign of --> a Will now try indefinite

>> syms a b positive
>> int(exp(-a*t)*sin(b*t),0,inf)
ans =
(a^2+b^2)^(1/2)*b/a/(1+b^2/a^2)^(1/2)/(1/(1+b^2/a^2)*a^2+2/(1+b^2/a^2)*b^2+b^4/a^2/(1+

>> res = simple(ans)
res =
1/(a^2+b^2)*b
```

Man beachte, dass das Integral

$$\int_0^\infty a^{-at} \sin(bt)\, dt$$

nur für positive a existiert. Dies muss bei der Deklaration von `a` angegeben werden. (Es ist nicht nötig, dass b positiv ist.)

7.0.5 Umformungen von symbolischen Ausdrücken

Die Umformung und Vereinfachung von symbolischen Ausdrücken ist oft nötig. Der einfachste Befehl ist `simplify`. Oft ist das Resultat dieses Befehls nicht das erwünschte. MATLAB stellt noch weitere Befehle zur Verfügung.

```
>> syms a b x y z
>> f = x^2 - 4*x + 4;
>> factor(f)
ans =
(x-2)^2
>> expand((a + b)^5)
ans =
a^5+5*a^4*b+10*a^3*b^2+10*a^2*b^3+5*a*b^4+b^5
>> factor(ans)
ans =
(a+b)^5
>> expand(exp(x+y))
ans =
exp(x)*exp(y)
>> expand(sin(x+2*y))
ans =
2*sin(x)*cos(y)^2-sin(x)+2*cos(x)*sin(y)*cos(y)
>> factor(x^6 - 1)
ans =
(x-1)*(x+1)*(x^2+x+1)*(x^2-x+1)
```

```
>> collect((x+y+z)*(x-y-z))
ans =
x^2+(y+z)*(-y-z)
>> collect((x+y+z)*(x-y-z),y)
ans =
-y^2-2*z*y+(x+z)*(x-z)
>> collect((x+y+z)*(x-y-z),z)
ans =
-z^2-2*z*y+(x+y)*(x-y)
>> simplify(sin(x)^2 + cos(x)^2)
ans =
1
>> simplify(exp(5*log(x) +1))
ans =
x^5*exp(1)
>> d = diff((x^2+1)/(x^2-1))
d =
2*x/(x^2-1)-2*(x^2+1)/(x^2-1)^2*x
>> simplify(d)
ans =
-4*x/(x^2-1)^2
```

Wenn unklar ist, welche Vereinfachungsregel die erfolgversprechendste ist, kann man mit dem Befehl `simple` ohne Ausgabeparameter (!) auf einen Schlag alle Möglichkeiten ausprobieren.

7.0.6 Substitutionen

Die Funktion `subs` erlaubt Variable in symbolischen Ausdrücken durch andere Ausdrücke zu ersetzen, insbesondere auch durch Zahlwerte.

```
>> syms a b x s t g
>> f = a+b;
>> subs(f,a,4)
ans =
4+b
>> subs(f,[a b],[4 -4])
ans =
     0
>> subs(sin(x),x,pi/3)
ans =
    0.8660
>> subs(sin(x),x,sym(pi)/3)
ans =
1/2*3^(1/2)
>> double(ans)
ans =
    0.8660
>> subs(g*t^2/2,t,sqrt(2*s))
ans =
g*s
```

7.0.7 Graphen von Funktionen

Die MATLAB-Funktion `fplot` erlaubt es, Funktionen bequem zu plotten. Da aber ein Funktionenhandle als Parameter für `fplot` benötigt wird, ist es oft nötig, zunächst eine Funktion in einem m-File zu definieren. In der Symbolic Math Toolbox gibt es die Funktion `ezplot`, mit der eine symbolisch definierte Funktion graphisch dargestellt werden kann.

```
>> ezplot(x*exp(x))
>> ezplot(x*exp(x),[-1 4])
>> ezplot(1/(1+30*exp(-x)))
```

7.0.8 Matrizen

Mit der Symbolic Math Toolbox lassen sich einfach Matrizen definieren, deren Elemente symbolische Ausdrücke sind. Mit diesen Matrizen lässt ähnlich rechnen wie mit numerischen Matrizen. Hier einige Beispiele

```
>> syms a b s
>> K = [a+b, a-b; b-a, a+b]
K =
[ a+b, a-b]
[ b-a, a+b]
>> G=[cos(s), sin(s); - sin(s), cos(s)];
>> K*G
ans =
[ (a+b)*cos(s)-(a-b)*sin(s), (a+b)*sin(s)+(a-b)*cos(s)]
[ (b-a)*cos(s)-(a+b)*sin(s), (b-a)*sin(s)+(a+b)*cos(s)]
>> inv(G)
ans =
[  cos(s)/(cos(s)^2+sin(s)^2), -sin(s)/(cos(s)^2+sin(s)^2)]
[  sin(s)/(cos(s)^2+sin(s)^2),  cos(s)/(cos(s)^2+sin(s)^2)]
>> simplify(ans)
ans =
[  cos(s), -sin(s)]
[  sin(s),  cos(s)]
>> L=K^2;
>> collect(L)
ans =
[        4*a*b, -2*b^2+2*a^2]
[  2*b^2-2*a^2,        4*a*b]
>> factor(L)
ans =
[          4*a*b,  2*(a+b)*(a-b)]
[ -2*(a+b)*(a-b),          4*a*b]
```

7.0.9 Algebraische Gleichungen

Der Befehl `solve(s)` versucht Nullstellen des symbolischen Ausdrucks `s` zu finden.

```
>> solve('3*x + 4 = 17')
ans =
13/3
>> f = '5^(x-1) = 10'; solve (f)
ans =
(log(5)+log(10))/log(5)
>> solve('tan(2*x) = sin(x)')
ans =
                                                    pi
                                                     0
    atan(1/2*(-2*3^(1/2))^(1/2),1/2+1/2*3^(1/2))
   atan(-1/2*(-2*3^(1/2))^(1/2),1/2+1/2*3^(1/2))
  atan(1/2*2^(1/2)*3^(1/4)/(1/2-1/2*3^(1/2)))+pi
 -atan(1/2*2^(1/2)*3^(1/4)/(1/2-1/2*3^(1/2)))-pi
>> solve('sin(3*x + 1.2) = 0')
ans =
-.400000000000000000000000000000000
```

Wenn Zahlen mit Dezimalpunkt eingegeben werden (z.B. 1.2 statt 6/5), so wird das Resultat numerisch statt symbolisch ausgegeben. Auch lineare Gleichungssysteme können natürlich gelöst werden:

```
>> g1 = 'x + y + z = 0';
>> g2 = '4*x + 5*y + z = 3';
>> g3 = '-2*x + y - 3*z -5 = 0';
>> [x y z] = solve(g1,g2,g3)
x =
-1
y =
3/2
z =
-1/2
```

7.0.10 Differentialgleichungen

Die Funktion `dsolve` versucht gewöhnliche Differentialgleichungen zu lösen. Wenn nicht genügend viele Randbedingungen gegeben werden, so enthält die Lösung freie Parameter.

```
>> Y = dsolve('Dy = x^2*y','x')
Y =
C1*exp(1/3*x^3)
>> Y = dsolve('Dy = x^2*y','y(0)=4','x')
Y =
4*exp(1/3*x^3)
>> dsolve('D2y + y = x^2', 'x')
ans =
sin(x)*C2+cos(x)*C1-2+x^2
>> dsolve('D2y + y = x^2', 'y(0)=4', 'Dy(0)=1','x')
ans =
sin(x)+6*cos(x)-2+x^2
```

Kapitel 8

Einführungen in Matlab, Resourcen auf dem Internet

Die Firma The Mathworks, die MATLAB produziert und vertreibt, hat eine gute Web-Seite an der URL

http://www.mathworks.com/.

Via den MATLAB Help Navigator kann man bequem auf Informationen von MathWorks zugreiffen, vorausgesetzt der Rechner ist am Internet angeschlossen. Nach dem Start von MATLAB Help, folge man den Links

`MATLAB ▷ Printable Documentation (PDF)`

Auf der Seite

http://www.mathworks.com/support/books/

findet man eine Liste von Hunderten von Büchern zu MATLAB und SimuLink, insbesondere zu Büchern über Numerische Methoden mit MATLAB in verschiedenen Fachgebieten.

Empfehlenswerte Einführungen in MATLAB sind die Bücher

- Kermit Sigmon & Timothy A. Davis: MATLAB Primer, 6th edition. Chapman & Hall/CRC, 2002.
- Desmond J. Higham & Nicholas J. Higham: MATLAB Guide, 2nd edition, SIAM, Philadelphia, 2005.
- Walter Gander & Jirí Hřebíček: Solving Problems in Scientific Computing Using Maple and MATLAB, 4th edition. Springer, 2004.

Das erste Buch ist (bei entsprechenden Zugriffsberechtigung) elektronisch verfügbar, siehe http://www.nebis.ch/. Das letzte ist eine Sammlung von interessanten und recht realistischen Problemen, deren Lösung mit Maple and MATLAB vorgeführt wird.

8.1 Tutorials

- http://www.math.mtu.edu/~msgocken/intro/intro.html
  'A Practical Introduction to Matlab' von Mark S. Gockenbach (Michigan Technological University). Sehr einfach.

- http://www.math.siu.edu/matlab/tutorials.html
  'MATLAB Tutorials' von Edward Neuman (Southern Illinois University). Fünf PDF Files. Ausfühliche Beschreibung zum Programmieren, zur linearen Algebra und zu numerischer Analysis mit MATLAB.

- http://math.ucsd.edu/~driver/21d-s99/matlab-primer.html
  'MATLAB Primer' (Second Edition) von Kermit Sigmon (Department of Mathematics, University of Florida) Gute Übersicht über die ältere MATLAB-Version 5.

8.2 Software

- Die Home page von The MathWorks ist

  http://www.mathworks.com/

  Auf dieser Seite gibt es auch public domain MATLABSoftware.

- Stimuliert durch das NA-NET, ein "Netzwerk" für und von numerischen Mathematikern, entstand 1986 NETLIB [3],

  http://www.netlib.org/

  von wo eine Unmenge frei verfügbarer (Public Domain) Software vor allem aus dem Gebiet der Numerischen Analysis gespeichert wird.

8.3 Alternativen zu Matlab

Als Alternativen zum (teuren) Matlab können aufgefasst werden

- SciLab: http://www.scilab.org/
- Octave: http://www.octave.org/

Beide Programme arbeiten ähnlich und mit ähnlicher Syntax wie MATLAB. m-Files sind teilweise auch in SciLab oder Octave benützbar.

Literaturverzeichnis

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide - Release 2.0*, SIAM, Philadelphia, 1994.

[2] J. J. Dongarra, C. Moler, J. Bunch, G. W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, 1979.

[3] J. Dongarra and E. Grosse, *Distribution of Mathematical Software via Electronic Mail*, Comm. ACM, **30**, 5, pp. 403–407, 1987.

[4] T. F. Coleman und Ch. van Loan, *Handbook For Matrix Computations*, SIAM, Philadelphia, 1988.

[5] W. Gander and J. Hřebíček, ed., *Solving Problems in Scientific Computing*, 3rd edition, Springer, Berlin, 1997.

[6] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, *Matrix Eigensystem Routines – EISPACK Guide Extension*, Springer Lecture notes in computer science 51, Springer, Berlin, 1977

[7] G. H. Golub and C. F. van Loan, *Matrix Computations*, 2nd edition, Johns Hopkins University Press, Baltimore, 1989.

[8] W. Gander, Matlab-*Einführung*. Neue Methoden des Wissenschaftliches Rechnen, Departement Informatik, 1991.

[9] A. A. Grau, U. Hill, H. Langmaack, *Translation of* ALGOL 60, Springer, Berlin, 1967. (Handbook of automatic computation ; vol. 1, part b)

[10] N. Higham, *Matrix Computations on a PC*, SIAM News, January 1989.

[11] D. J. Higham and N. J. Higham, *MATLAB Guide*, 2nd edition. SIAM, Philadelphia, 2005.

[12] P. Marchand, *Graphics and GUIs with MATLAB*, 2nd edition. CRC Press, Boca Raton, FL, 1999.

[13] C. Moler, *MATLAB Users' Guide*, University of New Mexico Report, Nov. 1980

[14] Matlab, *the Language of Technical Computing*, The MathWorks, South Natick MA, 2004. Available from URL http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/graphg.pdf.

[15] *Using* MATLAB *Graphics*, Version 7, The MathWorks, South Natick MA, 2004. Available from URL http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf.

[16] MICHAEL L. OVERTON *Numerical computing with IEEE floating point arithmetic*, SIAM, Philadelphia PA, 2001.

[17] HEINZ RUTISHAUSER, *Description of* ALGOL 60, Springer, Berlin, 1967 (Handbook of automatic computation ; vol. 1, part b)

[18] K. SIGMON AND T. A. DAVIS, *MATLAB Primer*, 6th edition. Chapman & Hall/CRC, 2002.

[19] B. SIMON UND R. M. WILSON, *Supercalculators on the PC*, Notices of the AMS, 35 (1988),e pp. 978–1001.

[20] B. T. SMITH, J. M. BOYLE, J. J. DONGARRA, B. S. GARBOW, Y. IKEBE, V. C. KLEMA, AND C. B. MOLER, *Matrix Eigensystem Routines – EISPACK Guide*, Springer Lecture notes in computer science 6, Springer, Berlin, 1976

[21] J. WILKINSON AND C. REINSCH, *Linear Algebra*, Springer, Berlin, 1971

[22] URL von Netlib: http://www.netlib.org/.

[23] URL von The Mathworks: http://www.mathworks.com/.