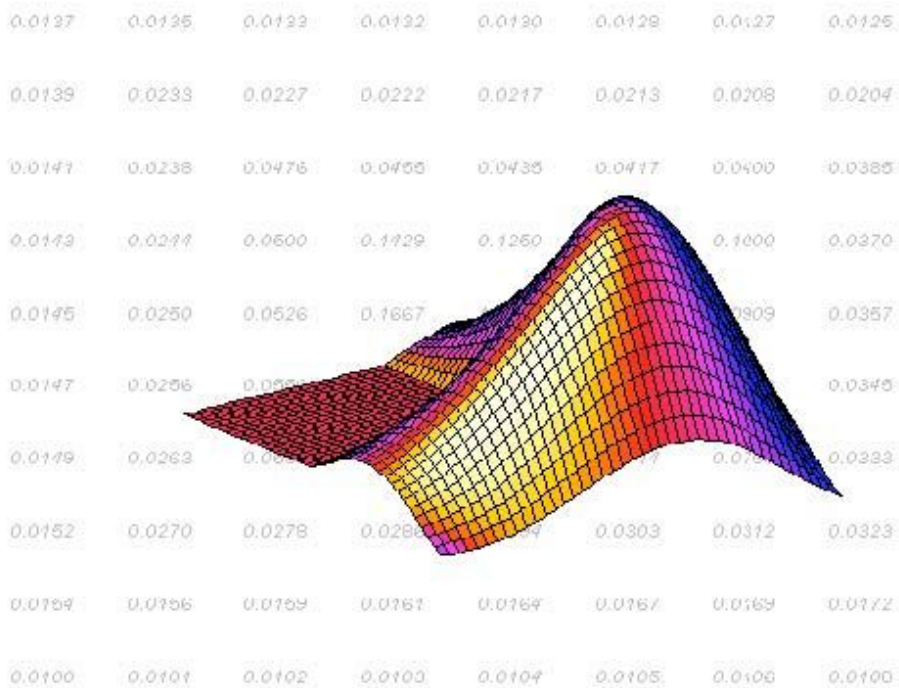


MATLAB - Eine Einführung

**Frederik Beuth
Susanne Teschl**



Dieses Skriptum soll Ihnen einen schnellen Einstieg in MATLAB (ab R2007a) ermöglichen. Es ist als kurze Einführung gedacht, die an Hand von Beispielen die Arbeitsweise von MATLAB demonstriert. Das Skriptum enthält zahlreiche Übungen mit Lösungen und kann daher auch zum Selbststudium verwendet werden.

Aktuelle Informationen zu MATLAB finden Sie unter anderem auf der offiziellen Webpage

<http://www.mathworks.com>

Das Skriptum besteht aus den folgenden Kapiteln:

Inhaltsverzeichnis

0. Was ist MATLAB?	3
1. Erste Schritte	4
Einfache Rechenoperationen	4
Abspeichern und Laden von Daten	5
Online Hilfe: help oder HELP-Menü	8
Eingebaute Funktionen	9
Darstellung von numerischen Ergebnissen	10
Spezielle Zahlenwerte: 'Inf',	10
2. Vektoren und Matrizen – die Grundbausteine von MATLAB	12
Vektoren	12
Elementweise Operationen	13
Polynome	15
Matrizen	16
Matrixoperationen	19
3. Grafik	21
'plot' veranschaulicht numerische Daten	21
Wie kann ich eine Grafik beschriften?	23
Dreidimensionale Plots	28
4. m-Files: MATLAB als Programmiersprache	31
Script-Files	31
Interaktive Eingabe	33
Function-Files	35
IF-Anweisungen und Schleifen:.....	37
Höhere Datenstrukturen.....	39
5. Verzeichnis nützlicher Funktionen.....	40

Über Anregungen oder Bemerkungen würde ich mich freuen!

Ursprungsskript:

Wien, Oktober 2001 Susanne Teschl

Aktuelle Version v0.8:

Chemnitz, Oktober 2009 Frederik Beuth beuth@hrz.tu-chemnitz.de

0. Was ist MATLAB?

Kurz gesagt: MATLAB ist ein Softwarepaket für **numerische Berechnungen** und für die **Visualisierung von Daten** im technisch-wissenschaftlichen Bereich.

MATLAB wurde in den 70er Jahren an der University of New Mexico und der Stanford University entwickelt um Kurse aus Lineare Algebra und Numerische Analysis zu unterstützen. Der Name ("**Matrix Laboratory**") erinnert noch daran. Heute ist MATLAB ein universelles Werkzeug, das in weiten Bereichen der angewandten Mathematik eingesetzt wird.

Bei MATLAB stehen numerische Rechnungen und die Darstellung von Zahlenmaterial im Vordergrund. Der **Grundbaustein** ist eine **Matrix**, deren Dimensionen nicht explizit definiert werden müssen. Dadurch können numerische Probleme innerhalb kürzester Zeit gelöst werden.

Man kann MATLAB auf zwei Arten verwenden:

- Bei der interaktiven Verwendung werden Anweisungen direkt über die Tastatur eingegeben und sofort ausgeführt.
- Für umfangreiche Probleme ist es empfehlenswert, MATLAB als Programmiersprache einzusetzen. Dabei werden mehrere Anweisungen als sogenannte m-Files abgespeichert. m-Files sind ASCII-Files und werden mit einem Texteditor geschrieben. Sobald sie im Commandfenster aufgerufen werden, führt sie der MATLAB-Interpreter wie ein Programm aus.

Darüber hinaus können auch **C- und Fortran-Programme** in Form von sog. *mex-Files* von MATLAB exekutiert werden.

Für spezielle Anwendungen gibt es **Toolboxes**, das sind Bibliotheken von m-Files zu bestimmten Aufgabenbereichen. So sind zum Beispiel elementare Befehle zum symbolischen Rechnen in der *Symbolic Math Toolbox* enthalten.

Berechnungen mit MATLAB können mithilfe eines MATLAB Notebooks (M-book) übersichtlich dokumentiert werden. Ein M-book ist ein Word-Dokument, das neben Text auch MATLAB-Anweisungen und deren Output enthält.

Die **farbliche Kennzeichnung** im Skript haben diese Bedeutungen:

- Matlabeingabe
- Matlab-ausgabe
- Matlab Fehlermeldung
- Übung
- Übung[opt]: zusätzliches Übungsbeispiel

1. Erste Schritte

Einfache Rechenoperationen

Nach dem Start von MATLAB öffnet sich das Commandfenster (*command window*) und es erscheint das *Prompt* » . In das Commandfenster werden MATLAB-Anweisungen eingegeben und numerische Ergebnisse ausgegeben. Zur Darstellung von Grafiken wird ein eigenes Grafikfenster geöffnet.

Alle Anweisungen werden nach dem Prompt eingegeben und **mit ↵ (Return) bestätigt**. MATLAB nennt das Ergebnis **ans** (kurz für *answer*):

```
12/3 +7*5 -1
```

```
ans =  
    38
```

Addition (+), Subtraktion (-), Multiplikation (*) und Division (/) werden wie gewohnt bezeichnet und verwendet.

Tipp: Wenn Sie im Menüpunkt FILE / PREFERENCES / COMMAND WINDOW "Numeric display" auf "compact" (statt "loose") stellen, so werden Leerzeilen in der Ausgabe vermieden.

Braucht man einen Ausdruck öfters, so kann man ihn als Variable definieren. MATLAB speichert den Wert der Variablen automatisch im sogenannten **Workspace**:

```
a = 48/3 - 3^2
```

```
a =  
    7
```

Folgende **Regeln** sind bei der **Definition von Variablen** zu beachten:

- Ein Variablenname darf keine Sonderzeichen außer dem Unterstrich enthalten.
- Das erste Zeichen muß ein Buchstabe sein.

```
Variable_1 = 4*2
```

```
Variable_1 =  
    8
```

Ein **Semikolon** am Ende der Eingabezeile bewirkt, dass MATLAB das Ergebnis zwar auswertet, aber nicht ausgibt:

```
A = 3*a^2 - Variable_1;
```

MATLAB unterscheidet zwischen **Groß- und Kleinbuchstaben** (*upper case* und *lower case letters*). Daher sind a und A verschiedene Variablen (man sagt, die Variablennamen sind *case sensitive*).

Beachten Sie, dass Potenzieren (z.B. a²) vor einer Multiplikation oder Division ausgewertet wird. Danach kommen Addition oder Subtraktion.

Sie können auch mehrere Anweisungen in eine Zeile eingeben: sind sie durch einen Komma getrennt, so folgt eine Ausgabe, werden sie durch einen Semikolon getrennt, so folgt keine Ausgabe:

```
b = (3+5)*6; c = (b/3)^2, d = A/c^2
```

```
c =  
    256
```

```
d =  
    0.0021
```

Das Komma einer Dezimalzahl wird, wie im Englischen allgemein üblich, als Punkt (*decimal point*) geschrieben.

Wichtig: Bei der Multiplikation darf das **Multiplikationszeichen (*) nicht weggelassen** werden:

3a

```
??? format compact;3a  
|  
Error: Missing operator, comma, or semicolon.
```

3*a

```
ans =  
    21
```

Übung 1:

(a) Sie geben nacheinander folgende Anweisungen ein. Was wird MATLAB ausgeben?

```
u = 2,v = 5;
```

```
(u+6)/4
```

```
y = x+1
```

```
y = 3u
```

(b) Welche der folgenden Variablennamen sind nicht zulässig?
anzahl, Summe_a+b, 5_Tageskarte, dauer_phase3

Abspeichern und Laden von Daten

Mit dem Befehl **who** kann man herausfinden, welche Variablen momentan im Workspace gespeichert sind:

```
who
```

```
Your variables are:
```

```
A          a          b          d  
Variable_1 ans          c
```

Um diese Daten zu sichern, speichern wir den Workspace mit dem Befehl **save**:

```
save ~/Matlab/uebung1
```

Der Workspace (d.h., alle bisher definierten Variablen) wird nun als File uebung1.mat im angegebenen Verzeichnis abgespeichert. MATLAB hängt die Endung **.mat** automatisch an. Zur Speicherung kann auch der Menüpunkt FILE / SAVE WORKSPACE AS... verwendet werden.

Mit der Anweisung **pwd** erhalten wir das Verzeichnis, in dem wir uns momentan befinden. Der Inhalt des aktuellen Verzeichnisses wird mit **dir** aufgelistet. Mit **cd Pfad** kann ins Verzeichnis mit dem Pfad *Pfad* gewechselt werden:

```
cd ~/Matlab; pwd
```

```
ans =  
~/Matlab
```

```
dir
```

```
. .. uebung1.mat
```

Ein Verzeichnis höher gelangt man mit **cd ..** (Achtung: zwischen "cd" und den Punkten ".." ist ein Leerzeichen notwendig!)

```
cd ..; pwd
```

```
ans =  
~/
```

Tipp: Fügen Sie das Verzeichnis, in dem Sie Ihre Files abspeichern, im sog. **MATLAB-Suchpfad** (*search path*) hinzu. Alle Verzeichnisse im Suchpfad werden bei MATLAB-Anweisungen wie z.B. **load** oder **which** (siehe später) automatisch durchsucht. Der Suchpfad wird mit **path** ausgegeben.

```
path
```

```
MATLABPATH
```

```
/opt/matlabR2009a/toolbox/matlab/general  
/opt/matlabR2009a/toolbox/matlab/ops  
.....  
/opt/matlabR2009a/toolbox/wavelet/compression
```

In unserem Fall wurde *uebung1.mat* im Verzeichnis *~/Matlab* abgespeichert. Wir fügen also dieses Verzeichnis dem Suchpfad hinzu mit

```
addpath ~/Matlab
```

Variablen werden wieder **gelöscht** mit der Anweisung

```
clear a           % löscht die Variable a  
oder  
clear           % löscht alle Variablen im Workspace
```

Kommentare

Prozentzeichen nach einem eingegebenen Befehl bedeuten, dass ein **Kommentar folgt**. MATLAB ignoriert alles, was in der Zeile nach dem Prozentzeichen steht. Kommentare sind besonders für m-Files (siehe Kapitel 4) von Bedeutung, da sie deren Lesbarkeit deutlich verbessern.

Nach Auswertung des clear-Befehls sind keine Variablen mehr im Workspace gespeichert:

who

Geben wir nun zwei neue Variable a und g ein und laden wir danach den Workspace uebung1 wieder. Das **Laden** erfolgt mit der Anweisung **load** (oder dem Menüpunkt FILE / LOAD WORKSPACE...):

a=1; g=2; who

Your variables are:

a g

load uebung1

Achtung: Das File **uebung1.mat** wird **nur dann gefunden** und geladen, wenn

- Sie sich entweder bereits im Verzeichnis von uebung1.mat befinden (mit pwd und dir herausfinden) oder
- wenn das Verzeichnis, das uebung1.mat enthält, im MATLAB-Suchpfad enthalten ist. Ansonsten muß nach load der gesamte Pfad von uebung1.mat angegeben werden.

who

Your variables are:

A	a	b	d
Variable_1	g		

Alle damals gespeicherten Variablen, aber auch die gerade neu eingegebenen Variablen stehen uns zur Verfügung!

Wichtig: Bereits im Workspace vorhandene Variablen werden durch die in uebung1 gespeicherten Variablen **ergänzt** und, wenn zweimal derselbe Buchstabe verwendet wurde, **überschrieben**:

a

a =
7

Das ist der Wert für a aus uebung1.

MATLAB wird mit **quit** oder mit dem Menüpunkt FILE / EXIT MATLAB beendet:

quit

Online Hilfe: help oder HELP-Menü

Der Befehl **help** ist sehr hilfreich, wenn man Näheres zu einer MATLAB-Anweisung wissen möchte. Wir erhalten zum Beispiel weitere Informationen zum Befehl **who** (und zu verwandten Befehlen) mit

help who

```
WHO List current variables.  
WHO lists the variables in the current workspace.  
WHOS lists more information about each variable.  
.....  
See also WHOS.
```

Eine **Liste aller Hilfe-Themen** (*help topics*) erhält man über den HELP-Menüpunkt oder **F1** oder einfach durch Eingabe von

help

HELP topics:

```
matlab/general -General purpose commands.  
matlab/ops -Operators and special characters.  
For more help on directory/topic, type "help topic".
```

Nun können wir zu einem der angeführten Themen nähere Informationen verlangen, zum Beispiel zum Thema "general":

help general

```
General purpose commands.  
  
MATLAB Version 7.8 (R2009a) 15-Jan-2009
```

....

Wir erhalten eine Liste der allgemeinen Befehle mit kurzer Beschreibung. Neuere Versionen von MATLAB stellen auch den sogenannten *help desk* (Hilfe im HTML-Format) zur Verfügung.

Übung 2:

Mit welchem Befehl wird das Commandfenster gelöscht (dh., der Bildschirm wird gelöscht, es bleiben aber trotzdem noch alle Variablenwerte gespeichert) ?

Übung 3:

Sie möchten die Variablen $a=2$ und $b=1$ zugleich aus dem Workspace löschen. Bisher kennen Sie die Möglichkeit:

clear a; clear b

Sie versuchen nun, sich Tipparbeit zu sparen, und geben ein:

clear a,b

Warum mißversteht MATLAB Ihren Befehl? Wie lautet die richtige Verwendung von clear, um mehrere Variable auf einmal zu löschen?

Eingebaute Funktionen

Es gibt - wie auch in jedem Taschenrechner - in MATLAB bereits "**eingebaute Funktionen**". Ein Beispiel ist die Wurzelfunktion (*square root*)

```
a = sqrt(2)/2
```

```
a =  
    0.7071
```

Hier haben wir $a = \frac{\sqrt{2}}{2}$ berechnet. Nähere Informationen zur Funktion **sqrt()** erhält man mit

help sqrt

```
SQRT Square root.
```

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

```
See also SQRTM.
```

```
Overloaded methods  
help sym/sqrt.m
```

Hinweis: Beachten Sie, dass die Anweisung $y = \text{sqrt}(x)$ nur dann von MATLAB ausgewertet werden kann, wenn x einen Zahlenwert hat (oder allgemein, wenn x eine numerische Matrix ist)! MATLAB ist vor allem auf die *Verarbeitung von numerischen Daten* ausgerichtet. Symbolische Ausdrücke können nur mit der *Symbolic Math Toolbox* verarbeitet werden (siehe Originalskript von S. Teschl, Kapitel 5). Ist x noch nicht definiert, so erhält man eine Fehlermeldung:

```
y = sqrt(x)
```

```
??? Undefined function or variable 'x'.
```

Eine Liste aller eingebauten elementaren Funktionen mit kurzer Beschreibung erhalten wir mit

help elfun

```
Elementary math functions.  
Trigonometric.
```

```
sin      - Sine.
```

```
....
```

```
sign     - Signum.
```

Tipp: Die **vorangehende Eingabe** bekommt man mit der Cursor-Taste \uparrow . Das ist zum Beispiel praktisch, wenn man einen Tippfehler ausbessern möchte. Wiederholtes Tippen von \uparrow oder \downarrow ermöglicht ein *Scrollen* in den vorangehenden Eingaben.

Übung 4:

Berechnen Sie den natürlichen Logarithmus von 1.36.

Übung 5:

Berechnen Sie $\cos(\pi)$ und $\cos(\pi/2)$. Das Argument der Kosinusfunktion wird von MATLAB immer im Bogenmaß interpretiert. Die Konstante π ist in MATLAB bereits eingebaut und wird mit **pi** bezeichnet.

Darstellung von numerischen Ergebnissen

MATLAB gibt standardmäßig ein ganzzahliges Ergebnis als Zahl ohne Nachkommastellen und ein nicht-ganzzahliges Ergebnis als Dezimalzahl mit 4 Nachkommastellen aus:

9.0/3.0

ans =

3

13/5

ans =

2.6000

1. Erste Schritte

Man kann das **Ausgabeformat** numerischer Resultate **ändern**. Zum Beispiel erhält man mehr Nachkommastellen durch Änderung des Formats von **short** auf **long**. Das geschieht durch direkte Eingabe von

format long; pi

ans =

3.14159265358979

oder über den Menüpunkt File / Preferences / Command Window.

Sehr große / kleine Zahlen werden automatisch in Gleitkommadarstellung ausgegeben (wir stellen hier zuvor wieder auf das Format shortum):

format short; 1/1000000

ans =

1.0000e-006

Eine Beschreibung der verschiedenen numerischen Formate erhalten Sie zum Beispiel mit `help format`.

Wichtig: Bei Änderung des Formats ändert sich **nur die Ausgabe**, nicht aber die interne Darstellung (und damit die Genauigkeit) der Zahlen.

(Relative) Genauigkeit der Darstellung numerischer Daten:

eps

ans =

2.2204e-016

eps gibt den Abstand von 1 zur nächst größeren (Gleitkomma-)Zahl an, die darstellbar ist.

Spezielle Zahlenwerte: 'Inf', ..

Es existieren einige mathematisch fest definierte Zahlenwerte. Diese können auch ab geprüft werden.

Unendlich:

Inf + 1 oder **1/0**

ans =

Inf

Ungültige Zahlenwerte:

0/0

```
ans =  
NaN
```

Prüfung:

`isnan(NaN)` `%ohne ''`

```
ans =  
1
```

Übung 6:

Berechnen Sie den natürlichen Logarithmus von 1.45 auf 14 Nachkommastellen genau.

Übung 7:

Finden Sie nur durch Wahl eines geeigneten Ausgabeformats eine möglichst einfache Bruchdarstellung von 0.784544.

2. Vektoren und Matrizen – die Grundbausteine von MATLAB

Vektoren

Nehmen wir an, wir möchten die Werte von x berechnen, z.B. für die Werte $x = 0, 2, 4, 6, 8, 10, 12$. Hier kommt uns MATLAB sehr entgegen. Wir fassen die x -Werte zu einem **Vektor** (*list* oder auch *array*) zusammen:

```
x = [0 2 4 6 8 10 12]
```

```
x =  
    0    2    4    6    8   10   12
```

Vektoren werden immer in **eckigen Klammern** (*brackets*) eingegeben. Die einzelnen Elemente des Vektors (*elements*) sind durch **Beistriche oder Leerzeichen** (*spaces*) zu trennen. Die Funktionswerte für die einzelnen Werte von x müssen nun nicht einzeln berechnet werden, sondern sie können alle auf einmal mit:

```
y = sqrt(x)
```

```
y =  
    0    1.4142    2.0000    2.4495    2.8284    3.1623    3.4641
```

erhalten werden! Der Befehl **sqrt** angewendet auf einen Vektor x gibt nämlich die Anweisung: Nimm **von jedem Element von x die Wurzel** und schreibe das Ergebnis als entsprechendes Element eines neuen Vektors (hier y genannt).

Diese unkomplizierte Art, einen Befehl (hier das Wurzelziehen) elementweise anzuwenden, ist eine große Stärke von MATLAB. Sie ermöglicht eine rasche Verarbeitung von großen Datenmengen.

Tipp: Der Vektor $x = [0 2 4 6 8 10 12]$ kann kürzer eingegeben werden:

```
x = 0:2:12
```

```
x =  
    0    2    4    6    8   10   12
```

Die Schreibweise $0:2:12$ bedeutet: **beginne mit 0, mit Schrittweite 2 und zähle bis die Grenze 12 erreicht ist**. Diese Anweisung ist sehr nützlich bei der Eingabe von Vektoren mit vielen Elementen. Man kann $0:2:12$ auch in runden oder eckigen Klammern schreiben: $(0:2:12)$ oder $[0:2:12]$.

Die Schrittweite (*increment*) darf auch negativ sein:

```
u = 29:-2:0
```

```
u =  
    29    27    25    23    21    19    17    15    13    11  
     9     7     5     3     1
```

Ist die **Schrittweite gleich 1**, so kann man die Angabe der Schrittweite **weglassen**:

```
z = 7:11
```

```
z =  
    7    8    9   10   11
```

Zugriff:

Wir erhalten z.B. das 4. Element von z mit:

```
z(4)
```

```
ans =  
    10
```

Übung 1:

Erzeugen Sie einen Vektor y, der die Funktionswerte des natürlichen Logarithmus an den Stellen x = 1,3,5,7,9 enthält. Was gibt MATLAB aus, wenn Sie y(1) eingeben?

Übung 2:

Geben Sie die Vektoren a und b mit den Elementen -10,-8,-6,...,6,8,10 bzw. 10,9,8,...,0 mit kurzen Anweisungen ein.

Übung 3:

x = [1 3 -2 6 0 7 11 -8]. Finden Sie mit **help paren** (von *parentheses*; man kommt darauf mit **help ops**) und **help colon** heraus, wie man aus x einen Vektor bildet, der

- das 1., 4. und 7. Element von x enthält
- aus den ersten 4 Elementen von x besteht
- jedes zweite Element von x enthält.

Elementweise Operationen

Vektoraddition und die Multiplikation eines Vektors mit einem Skalar sind wie üblich definiert, nämlich **elementweise**:

```
w = [1 2 3]*2 % Vektor * Skalar
```

```
w =  
    2  4  6
```

Hier wird jedes Element des Vektors [1 2 3] mit 2 multipliziert und der so entstandene Vektor wird w genannt.

```
[2 -1 9] + [1 3 6] % Vektor + Vektor
```

```
ans =  
    3  2 15
```

Zu jedem Element von [2 -1 9] wird das entsprechende Element von [1 3 6] addiert. Die Differenz zweier Vektoren wird analog gebildet.

Achtung: Es können **nur Vektoren bzw. Matrizen mit gleichen Dimensionen** addiert oder subtrahiert werden! Man sagt auch, sie müssen dieselbe Länge (*length* oder *dimension*) haben:

```
w + [1 4 6 7]
```

```
??? Error using ==> +  
Matrix dimensions must agree.
```

Die Länge eines Vektors (dh. die Anzahl seiner Elemente) erhalten wir mit

length(w)

```
ans =  
    3
```

Neben diesen aus der Mathematik bekannten Vektoroperationen sind in MATLAB **weitere elementweise Vektoroperationen** definiert:

[2 5 7] + 3 % Vektor + Zahl

```
ans =  
    5    8   10
```

Zu jedem Element von [2 5 7] wird 3 addiert.

Neu ist auch die in MATLAB definierte **elementweise Multiplikation**. Dazu benötigen wir zwei Vektoren der **gleichen Länge**:

a = 1:2:10, b = 1:5

```
a =  
    1    3    5    7    9
```

```
b =  
    1    2    3    4    5
```

Nun wollen wir einen neuen Vektor bilden, indem wir das erste Element von a mit dem ersten Element von b multiplizieren, das zweite Element von a mit dem zweiten von b... Der MATLAB-Befehl für diese Operation ist allerdings nicht, wie vielleicht erwartet, $a*b$, sondern

a .* b % elementweise Multiplikation

```
ans =  
    1    6   15   28   45
```

Das Ergebnis ist wieder ein Vektor der Länge 5.

Achtung: Der **Punkt vor dem Stern** (*asterisk*) ist **notwendig!** Die Anweisung **a*b** bedeutet die in der Mathematik übliche **Matrixmultiplikation**. Sie ist völlig verschieden von der elementweisen Multiplikation und kann nur durchgeführt werden, wenn die Anzahl der Spalten von a gleich der Anzahl der Zeilen von b ist:

a*b

```
??? Error using ==> *  
Inner matrix dimensions must agree.
```

Die **elementweise Division** in MATLAB ist ganz analog definiert:

a./b

```
ans =  
    1.0000  1.5000  1.6667  1.7500  1.8000
```

Jedes Element von a wird durch das entsprechende Element von b dividiert. Auch ein **elementweises Potenzieren** ist möglich:

a.^2 % elementweises Potenzieren

```
ans =  
    1  9  25  49  81
```

Jedes Element von a wird quadriert.

Polynome

Polynome sind besonders einfache mathematische Funktionen, die vielfach Anwendung finden. Ein Polynom, etwa $y = -x^3 + 2 \cdot x^2 + 5$, kann über Punktoperationen für einen bestimmten Vektor x errechnet werden. Einfacher kann ein Polynom in Matlab durch seine Koeffizienten angegeben werden. Im folgenden soll das Polynom $y = -x^3 + 3 \cdot x^2 + 1$ für die Werte $x = -1, 0, 1, 2, 3, 4$ und 5 berechnet werden. x wird als Vektor definiert; dazu werden die Polynomkoeffizienten $-1, 3, 0$ und 1 ebenfalls als Vektor, etwa **koef**, zusammengefasst. Die Berechnung erfolgt mit der **polyval**-Anweisung.

```
x = -1:5;  
koef = [-1 3 0 1]; % Koeffizient der höchsten Potenz zuerst  
polyval(koef, x)
```

```
ans =  
    5  1  3  5  1 -15 -49
```

Übung 4:

$a = [1 \ 4 \ 6]$ und $b = [-1 \ 2 \ 1]$. Was gibt MATLAB aus, wenn Sie eingeben: $a+b$, a^*2 , $a/2$, $a+3$, $a*b$, $a.*b$, $a./b$? Geben Sie die Antwort, bevor Sie mit MATLAB rechnen!

Übung 5:

Sie möchten einen Vektor x mit den Elementen $0, 0.5\pi, \dots, 2\pi$ (d.h. Schrittweite 0.5π) definieren. Dazu können Sie zum Beispiel die elementweise Multiplikation mit π

```
x = (0:0.5:2)*pi
```

verwenden. Warum sind hier die Klammern notwendig?

Übung 6 [opt]:

a) Berechnen Sie die Werte der Polynomfunktion

$y = -0,01 \cdot x^3 + 0,02 \cdot x^2 + 1,2$ für $x = 0, 1, 2, 3, 4, 5$ und 6

b) Die Sinusfunktion kann näherungsweise in der Form $\sin x \approx x - \frac{x^3}{6} + \frac{x^5}{120}$

(Winkel x in Bogenmaß) geschrieben werden. Vergleichen Sie den genauen Sinuswert und den Näherungswert für $x = 0^\circ, 10^\circ, 20^\circ, \dots, 50^\circ$.

Matrizen

Matrizen sind wertvolle Hilfsmittel bei der Verarbeitung von Daten. Betrachten wir zum Beispiel das lineare Gleichungssystem

$$\begin{aligned}3x + 4y - 2z &= 4 \\ -x + 2y + 8z &= -1 \\ 2x + -5z &= 3\end{aligned}$$

Es kann auch in der Form $A*x = b$ geschrieben werden, mit einer Matrix A und zwei Spaltenvektoren x und b. Matrizen werden in MATLAB zwischen eckigen Klammern eingegeben. Das Ende **einer Zeile** wird **durch ein Semikolon gekennzeichnet**:

$$A = [3\ 4\ -2; -1\ 2\ 8; 2\ 0\ -5]$$

A =

$$\begin{array}{ccc}3 & 4 & -2 \\ -1 & 2 & 8 \\ 2 & 0 & -5\end{array}$$

Die **Transponierte** einer (reellen) Matrix erhalten wir, indem wir ein **Apostroph** ' anhängen

$$C = A'$$

C =

$$\begin{array}{ccc}3 & -1 & 2 \\ 4 & 2 & 0 \\ -2 & 8 & -5\end{array}$$

Hat die Matrix **auch komplexe Elemente**, so wird sie durch Anhängen von ' transponiert und die Elemente werden komplex konjugiert.

Geben wir den Vektor b in der gewohnten Form ein,

$$b = [4\ -1\ 3];$$

so erhalten wir eine einzeilige Matrix, d.h., einen **Zeilenvektor**. Die Schreibweise $A*x = b$ verlangt aber, dass b ein **Spaltenvektor** ist. Mit **b'** entsteht aus dem Zeilenvektor b durch Transponieren ein Spaltenvektor. Wir definieren den Vektor b neu durch

$$b = b'$$

b =

$$\begin{array}{c}4 \\ -1 \\ 3\end{array}$$

(Der Ausdruck links von einem Gleichheitszeichen wird immer durch den Ausdruck rechts definiert.)

Tipp: Einen Spaltenvektor erhalten Sie auch mit **b(:)**.

Zugriff

Wie bei Vektoren können wir auch ein bestimmtes Element einer Matrix herausgreifen:

C(3,2)

```
ans =  
      8
```

C(3,2) bedeutet das Element in der **dritten Zeile und zweiten Spalte** der Matrix C. Die ganze zweite Zeile von C wird ausgegeben mit:

C(2,:)

```
ans =  
      4  2  0
```

C(2,:) bedeutet: nimm die **2. Zeile** und **alle Spalten** (der **Doppelpunkt** steht hier für "*alle Spalten*"). Entsprechend erhalten wir zum Beispiel die ganze erste Spalte von C durch

C(:,1)

```
ans =  
      3  
      4  
     -2
```

Hier steht der Doppelpunkt für "*alle Zeilen*".

Möchten wir **ein bestimmtes Element der Matrix C ändern**, zum Beispiel C(3,2) auf den Wert 7, so geben wir ein

C(3,2) = 7

```
C =  
      3  -1  2  
      4   2  0  
     -2   7 -5
```

Soll eine ganze Spalte oder Zeile von C gelöscht werden, so setzen wir diese Spalte (bzw. Zeile) gleich []. So wird etwa die **erste Spalte** von C **gelöscht**, indem wir eingeben:

C(:,1) = []

```
C =  
     -1   2  
      2   0  
      7  -5
```

Die **Dimension einer Matrix** (d.h., ihre Zeilen- und Spaltenanzahl), gibt der Befehl

size (C)

```
ans =  
      3  2
```

Als Ergebnis erhält man den Vektor [*Zeilenanzahl, Spaltenanzahl*], hier [3 2].

Tipp: Die Anweisung **whos** gibt ausführliche Informationen über die im Workspace gespeicherten Variablen.

Einige **spezielle** Matrizen werden oft gebraucht: Alle Matrizen sind in von quadratisch oder auch rechteckig verfügbar.

```
zeros(2)           % 2x2 Nullmatrix
ans =              0 0
                  0 0

zeros(2,4)         % 2x4 Nullmatrix
ans =              0 0 0 0
                  0 0 0 0

eye(3)             % 3x3 Einheitsmatrix (3-by-3 identity matrix )
ans =              1 0 0
                  0 1 0
                  0 0 1

ones(3)            %3x3 Einermatrix
ans =              1 1
                  1 1

ones(3) * 1.7      % 3x3 Matrix mit 1.7
ans =              1.7 1.7
                  1.7 1.7

rand(2, 3)         % 2x3 Matrix mit Zufallszahlen
ans =              0.9058 0.9134 0.0975
                  0.1270 0.6324 0.2785
```

Übung 7:

A = [1 3 5; 2 0 1; 2 4 6]; Was ist das Ergebnis der folgenden Anweisungen? A', A(1,:), A(:,3), A(2,2), size(A). Überlegen Sie, bevor Sie mit MATLAB rechnen.

Übung 8:

Ersetzen Sie in einer 3x4 Matrix B mit Zufallszahlen das erste Element in der ersten Zeile durch 0 und streichen Sie die gesamte zweite Spalte von B.

Übung 9:

Erzeugen Sie eine 2x5 Matrix A mit Zufallszahlen. Bilden Sie daraus:

- eine Matrix B, die aus den ersten drei Spalten von A besteht
- eine Matrix C, die aus der zweiten und vierten Spalte von A besteht.

Finden Sie die dazu nötigen Anweisungen mit help colon und help paren.

Matrixoperationen

$A+B$, $A-B$, $A*B$ bezeichnen die üblichen Matrizenoperationen.

Wichtig: Der Stern '*' allein bedeutet die übliche **Matrizenmultiplikation** (die nur definiert ist, wenn die Anzahl der Spalten von A gleich der Anzahl der Zeilen von B ist). Darüber hinaus ist, wie bei den Vektoren, auch die **elementweise Multiplikation** definiert, die mit dem Symbol '.*' bezeichnet wird:

```
[1 2; -2 5] .* [3 6; 0 -1]
```

```
ans =  
     3    12  
     0    -5
```

Jedes Element von [1 2; -2 5] wird mit dem entsprechendem Element von [3 6; 0 -1] multipliziert.

Mehr Informationen zur Matrizeneingabe und –manipulation erhalten Sie zum Beispiel mit **help elmat**, **help matfun** oder auch **help colon** und **help paren**.

Existiert die Inverse A^{-1} , so ist die Lösung x von $A*x = b$ gegeben durch $x = A^{-1}*b$. A ist invertierbar genau dann, wenn die Determinante von A ungleich Null ist:

```
det(A) % Determinante der Matrix A
```

```
ans =  
     22
```

Wir erhalten die Inverse A^{-1} mit **inv(A)**:

```
x = inv(A)*b
```

```
x =  
     2.1818  
    -0.5000  
     0.2727
```

MATLAB bietet noch eine andere Schreibweise für $x = A^{-1}*b$:

```
x = A\b
```

```
x =  
     2.1818  
    -0.5000  
     0.2727
```

Das Symbol '\' bezeichnet die sogenannte **Linksdivision (left division)**. Der Name kommt daher, dass nun der **Nenner links** steht und der **Bruchstrich nach links geneigt** ist. Hier steht A\ sozusagen für die Inverse von A. Im Fall von Skalaren ist die Linksdivision

```
2\3
```

```
ans =  
     1.5000
```

gleich der Rechtsdivision $3/2$ (die Multiplikation von Skalaren ist ja kommutativ). Bei Matrizen aber ist $inv(A)*b$ (Linksdivision) ungleich $b*inv(A)$ (Rechtsdivision), der letzte Ausdruck ist nämlich **gar nicht definiert**. Daher erhalten wir eine Fehlermeldung, wenn wir eingeben:

```
b/A
```

??? Error using ==> /
Matrix dimensions must agree.

Hinweis: Die Verwendung von $x = A \backslash b$ anstelle von $x = \text{inv}(A) * b$ hat verschiedene Vorteile:

- Erstens werden bei Eingabe von $A \backslash b$ weniger **interne Rechenschritte** durchgeführt (es wird das Gaußsche Eliminationsverfahren verwendet). Daher erhält man so vor allem bei größeren Problemen **schneller eine Lösung**.
- Zweitens liefert die Eingabe von $x = A \backslash b$ auch dann eine Lösung, wenn das **Gleichungssystem nicht eindeutig lösbar** ist (und daher die Inverse A^{-1} nicht existiert). Im Fall eines **überbestimmten** Systems erhält man zum Beispiel eine Lösung, die **den quadratischen Fehler von $A * x - b = 0$ minimiert**.

Für nähere Informationen siehe Handbuch Teil 3 (*Reference*), Stichwort "*Arithmetic Operators*" oder z.B. **help slash**. Wie ein Gleichungssystem symbolisch gelöst wird, kann im Kapitel 5 des Originalskriptes von S. Teschl nachgeschlagen werden.

Übung 10:

$A = \text{eye}(3,3)$, $B = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 9]$. Was ist das Ergebnis von $A+B$, A^2 , $A/2$, $B.^2$, $A.*B$, $A*B$? Überlegen Sie, bevor Sie mit MATLAB rechnen!

Übung 11:

Lösen Sie das Gleichungssystem

$$-x + 4y + 6z = 5$$

$$3x - z = 7$$

$$2x + 7y - 3z = -1$$

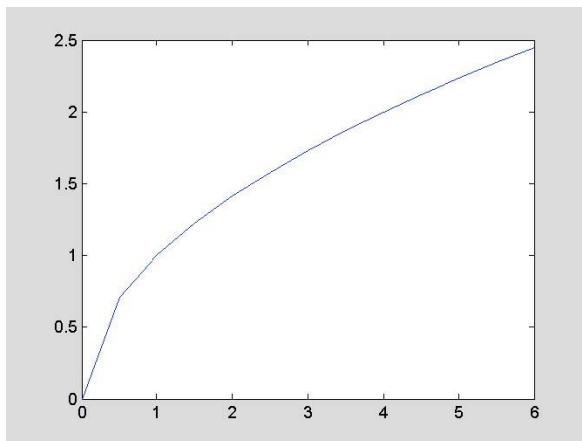
3. Grafik

'plot' veranschaulicht numerische Daten

In MATLAB kann man mit wenigen Befehlen sehr leicht Daten veranschaulichen. Die Anweisung **plot(x,y)** erzeugt ein **Grafikfenster**. (Achtung: es ist oft hinter den anderen geöffneten Fenstern versteckt!). x und y sind Vektoren, sie enthalten die x- und y-Koordinaten der zu zeichnenden Datenpunkte.

Wir wollen zum Beispiel die Wurzelfunktion im Intervall [0,6] zeichnen:

x = 0:0.5:6; y = sqrt(x); plot(x,y)



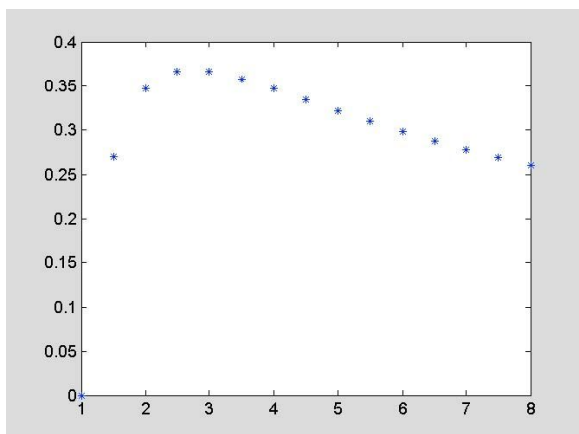
Es öffnet sich ein Grafikfenster und die Funktionswerte von $y = \sqrt{x}$ werden für die Werte $x = 0, 0.5, 1, \dots, 5.5, 6$ gezeichnet. MATLAB wählt die Achsenbegrenzungen automatisch, beschriftet die Achsen und **verbindet die Datenpunkte durch gerade Linien**.

Tipp: Mithilfe des Menüpunktes WINDOW (bei manchen MATLAB-Versionen auch mit der Tastenkombination Alt + Tab) kann man zwischen Commandfenster und Grafikfenster wechseln.

Übung 1: Zeichnen Sie die Gerade $y = \frac{3}{4}x - 1$ im Intervall [0,4].

Das nächste Beispiel zeigt, wie praktisch die in MATLAB möglichen elementweisen Operationen sind. Es ist die Funktion $y = \ln(x)/x$ im Intervall [1,8] zu zeichnen:

**x = 1:0.5:8; y = log(x)./x;
plot(x,y,'*')**



Der Stern (*) unter **einfachen Anführungszeichen** bewirkt, dass **nur die Datenpunkte gezeichnet** werden, und zwar in Form von Sternen (anstelle der Sterne sind zum Beispiel auch Kreise (o) oder Kreuze (+) möglich).

Achtung: Es darf nur das Anführungszeichen ' verwendet werden. Verwendet man ´ oder ` oder ", so erhält man eine Fehlermeldung.

Übung 2:

Wie sieht der Graph von $y=1-e^{-2t}$ im Intervall $[0,3]$ aus?

Übung 3:

Zeichnen Sie folgende Kurve in Parameterdarstellung: $x(t) = \sin(2t)$, $y(t) = \cos(t)$, $0 \leq t \leq 2\pi$.

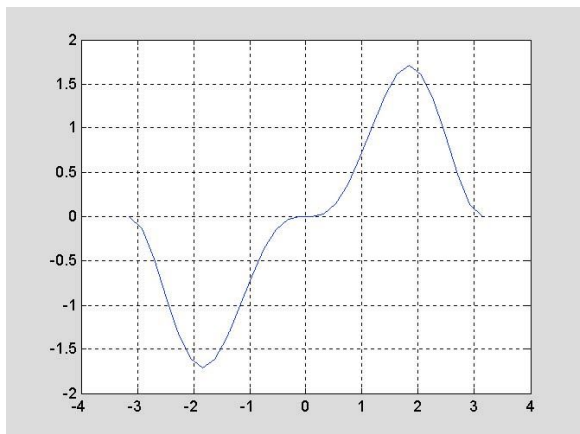
Möchte man die **Anzahl der gezeichneten Datenpunkte vorgeben**, so erzeugt man den Vektor x am besten mit **linspace**:

```
x = linspace(-pi,pi,30);
```

Hier besteht der Vektor x aus 30 Werten im gleichen Abstand, wobei der erste Wert $-\pi$ und der letzte Wert π ist. **Merkhilfe: linspace (erster Wert, letzter Wert, Anzahl der Werte)**. Fehlt die Angabe für die Anzahl der Werte, so wird dafür 100 genommen.

```
y = x.*sin(x).^2;
```

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



Die Anweisung **grid** erzeugt **Gitterlinien**.

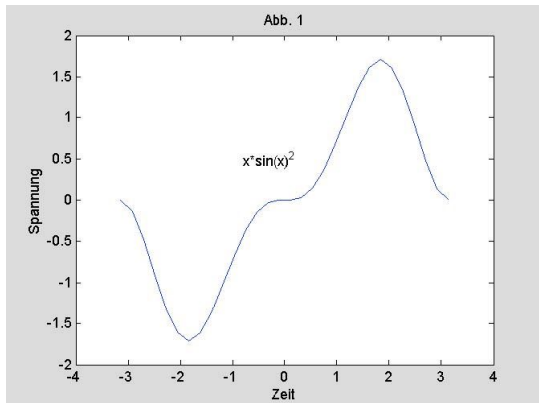
Tipp: Die Kurve kann mit **verschiedenen Linienarten** (linestyles) und **Farben** gezeichnet werden.

Einen Überblick über die verschiedenen Möglichkeiten erhalten Sie z.B. mit der Anweisung **help plot**.

Wie kann ich eine Grafik beschriften?

Die Koordinatenachsen werden mit **xlabel** bzw. **ylabel** beschriftet. Die Anweisung **title** versieht die Grafik mit einer Überschrift, und mit **text** kann man eine Beschriftung in die Grafik setzen.

```
plot(x,y);  
xlabel('Zeit'), ylabel('Spannung')  
title('Abb. 1')  
text(-0.8,0.5,'x*sin(x)^2')
```



In der aktuellen Grafik wird die x-Achse mit 'Zeit' beschriftet, die y-Achse mit 'Spannung'. Die ganze Grafik wird mit 'Abb.1' betitelt. Im Punkt mit den **Koordinaten (-0.8,0.5)** beginnt der Text 'x*sin(x)^2'.

Achtung: Es ist wichtig, **zuerst** die Anweisung **plot(x,y)** einzugeben, und dann erst die Befehle zur Beschriftung der Grafik. Die Beschriftung wird nämlich immer in die **aktuelle Grafik** eingefügt.

Übung 4:

Beschriften Sie den Graphen von $y = 1 - e^{-2t}$ im Intervall $[0,3]$.

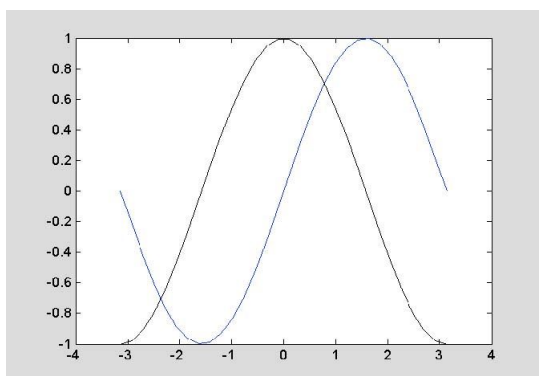
Übung 5:

Beschriften Sie den Plot: $x(t) = \sin(2t)$, $y(t) = \cos(t)$, $0 \leq t \leq 2\pi$.

Wie kann ich mehrere Funktionen gleichzeitig zeichnen?

Die Anweisung **plot(x,y,x,z)** zeichnet die Funktionen $y(x)$ und $z(x)$ in das gleiche Koordinatensystem.

```
x = linspace(-pi,pi,30); y = sin(x); z = cos(x);  
plot(x,y,x,z)
```

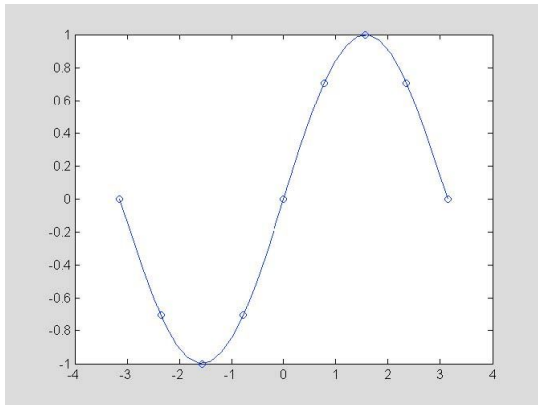


Die Sinus- und Kosinusgraphen werden automatisch in verschiedenen Farben in ein-und dasselbe Grafikfenster gezeichnet.

Tipp: Eine andere Möglichkeit bietet die Anweisung **hold**. Sie bewirkt, dass die aktuelle Grafik im Grafikfenster festgehalten wird. Anschließende plot-Anweisungen **fügen die Grafiken in dasselbe Achsenkreuz** hinzu. **hold off** beendet diese Funktion.

```
x1=linspace(-pi,pi,30);  
x2=-pi:pi/4:pi;  
y1=sin(x1); y2=sin(x2);  
plot(x1,y1); hold; plot(x2,y2,'o');
```

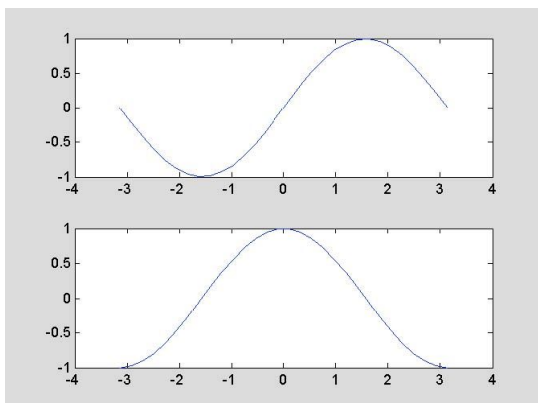
Current plot held



Tipp: Möchte man die nächste Grafik in ein **weiteres Grafikfenster** gezeichnet haben, so muss man zuvor ein neues Grafikfenster mit dem Menüpunkt FILE/NEW.../FIGURE öffnen oder **figure** eingeben.

Matlab ermöglicht es auch, das Grafikfenster mit Hilfe des Befehls **subplot (m,n,p)** in **mehrere Teilfenster** zu zerlegen.

```
x=linspace(-pi,pi,30); y = sin(x); z = cos(x);  
subplot(2,1,1); plot(x,y);  
subplot(2,1,2); plot(x,z);
```



Die erste Zahl m sagt, dass in vertikaler Richtung m Grafikteilfenster, die zweite Zahl n, dass in horizontaler Richtung n Grafikteilfenster vorgesehen sind. Die Zahl p gibt an, in welches der m mal n Teilfenster die nächste Grafik gezeichnet werden soll. Gezählt wird zeilenweise von links oben nach rechts unten.

Übung 6:

Zeichnen Sie $y = x^2$ und $z = 2 + |x|$ im Intervall $[-3,3]$ und lesen Sie aus der Grafik ab, für welche x die Beziehung $2 + |x| < x^2$ gilt!

Tipp: Falls Sie `linspace` verwenden, wählen Sie eine ungerade Anzahl an Datenpunkten (warum?).

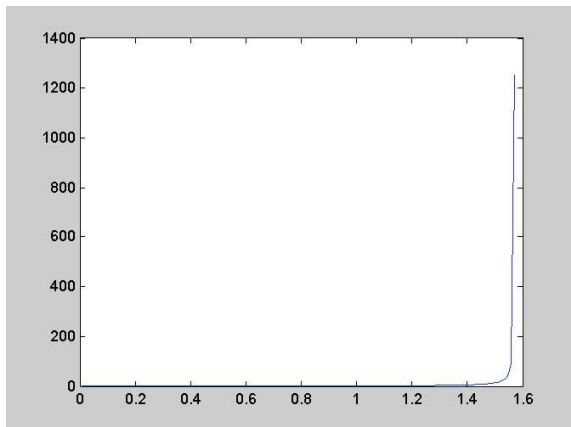
Übung 7:

Zeichnen Sie die Funktionen $y = x$, $w = x^2$ und $z = \sqrt{x}$ in eine Grafik.

Wie kann ich die Skalierung verändern?

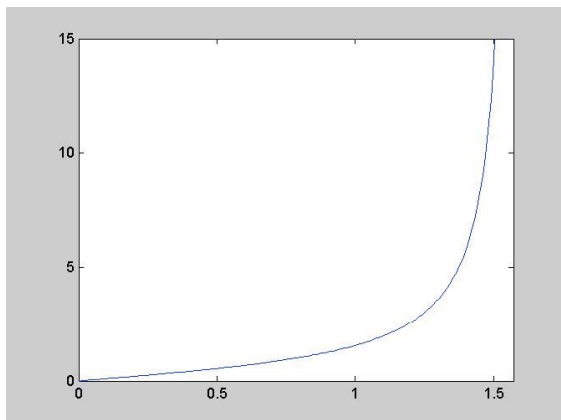
Angenommen, wir möchten die Funktion $y = \tan(x)$ im Intervall $[0, \pi/2]$ zeichnen:

`x=0:0.01:pi/2; plot(x,tan(x));`



Die Funktion wird nicht gut dargestellt, denn der von MATLAB automatisch gewählte y-Bereich ist zu groß! Wir können den x- und y-Bereich, in dem die Funktion gezeichnet wird, mit der Anweisung `axis([xmin, xmax, ymin, ymax])` ändern. Die Funktion wird dann im x-Bereich $x_{\min} \leq x \leq x_{\max}$ und im y-Bereich $y_{\min} \leq y \leq y_{\max}$ dargestellt.

`axis([0, pi/2, 0, 15]);`



Gleiche Skalierung auf der x- und y-Achse erreichen wir mit `axis equal`. Die ursprüngliche (von MATLAB gewählte) Skalierung wird mit `axis normal` wiederhergestellt.

Tipp: Um einen bestimmten Ausschnitt des Graphen zu vergrößern, kann auch `zoom` verwendet werden.

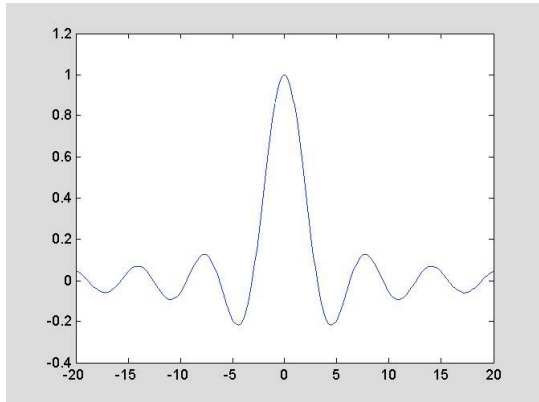
Übung 9:

Zeigen Sie graphisch, dass die Funktion $y = x^3 - x$ nicht monoton wachsend ist.

Kann ich eine Funktion zeichnen, ohne Datenpunkte anzugeben?

Das ist mit der Anweisung **fplot** möglich. Mit ihr kann man Funktionen zeichnen, ohne vorher einen Vektor x definieren zu müssen. MATLAB wählt die Stellen, an denen die Funktionswerte berechnet werden, automatisch.

```
fplot('sin(x)./x', [-20,20,-0.4,1.2])
```



Die zu zeichnende Funktion wird **zwischen Anführungszeichen** eingegeben, danach der **gewünschte x-Bereich** (hier: $-20 \leq x \leq 20$) und der gewünschte y-Bereich (hier: $-0.4 \leq y \leq 1.2$). Der y-Bereich kann auch weggelassen werden, dann wählt MATLAB ihn automatisch.

Wichtig: x und y sind **intern nach wie vor Vektoren**, daher müssen elementweise Operationen verwendet werden (also zum Beispiel **.** / anstelle von /)!

Tipp: Auch mit der Anweisung **ezplot** ("easyplot") können einfache Grafiken erzeugt werden. Für nähere Informationen siehe `help ezplot` oder `help plotxy`.

Wie kann ich Grafiken schließen ?

Der Befehl **close** schließt die aktuelle Grafik, während der Befehl **close all** alle offenen Fenster schließt.

Wie kann ich Grafiken im Hintergrund erzeugen und abspeichern

Wenn viele Grafiken automatisch erzeugt werden sollen und man nebenbei weiterarbeiten möchte, so sollte dies im Hintergrund geschehen. Andernfalls wird der Tastatur/Mausfokus immer der aktuellen Grafik zugeordnet. Um dies zu verhindern, können Grafiken unsichtbar gezeichnet werden. Das abspeichern als Bild funktioniert trotzdem.

```
h=figure;  
set( h, 'visible', 'off' );  
....%Grafik zeichnen  
saveas(gcf, 'test.bmp');  
close (h);  
%schließt das Grafikenfenster
```

Übung 10:

Zeichnen Sie die Funktion $y = \sin(x)$ im Intervall $[0, 2\pi]$ mit fplot.

Übung 11:

Zeichnen Sie die Funktion $y = x + 1/(10000x)$ im Intervall $[-1, 1]$ mit fplot. Warum gibt die Grafik das Verhalten der Funktion nicht richtig wieder?

(Tipp: Was kann man über das Verhalten von f am Nullpunkt aussagen?). Stellen Sie die Grafik richtig, indem Sie den y -Bereich einschränken und die Anzahl der gezeichneten Kurvenpunkte erhöhen.

Übung 12: Wurfparabel

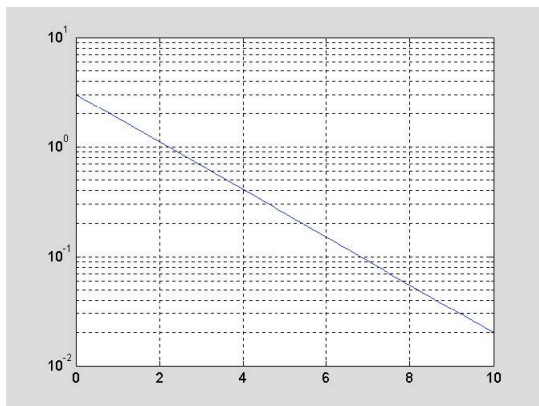
Zeichnen Sie die Wurfparabel: $x(t) = v t \cos(\phi)$, $y(t) = v t \sin(\phi) - g t^2 / 2$.

Anfangsgeschwindigkeit: $v = 100\text{km/h}$, Abwurfwinkel: $\phi = 60^\circ$. (Achtung: Einheiten in m/s bzw. Radiant umwandeln!)

Logarithmische Skalierung einer Achse

Es kann vorteilhaft sein, eine oder beide Achsen eines rechtwinkligen Koordinatensystems logarithmisch zu skalieren. So wird eine Exponentialfunktion $y = a \cdot e^{b \cdot x}$ in einem "ordinatenlogarithmischen Papier" (d.h. die y -Achse ist logarithmisch geteilt) als Gerade dargestellt.

```
x = linspace(0,10);  
y = 3*exp(-0.5*x);  
semilogy(x,y); grid
```



semilogy wird wie plot verwendet, die y -Achse wird jedoch nun **logarithmisch skaliert**. Analog bewirkt **semilogx** eine logarithmische Teilung der x -Achse, **loglog** eine logarithmische Skalierung beider Achsen. Zu beachten ist, dass eine logarithmische Skalierung nie von 0 beginnen kann, da $\log(0)$ nicht definiert ist.

Übung 13: Doppellogarithmische Skalierung

Zeichnen Sie die Funktion $y = \sqrt{x}$ in einem doppellogarithmischen Papier zwischen 1 und 1000.

Dreidimensionale Plots

Will man eine Funktion $z = f(x,y)$ graphisch veranschaulichen, so kann man den Graph dieser Funktion, eine Fläche, über der (x,y) -Ebene zeichnen. Dazu ist anzugeben, über welchem x -Bereich und y -Bereich gezeichnet werden soll. Wir wollen die Funktion $z = x \cdot y$ zuerst sehr grobflächig als *Maschenplot* (*Drahtgitter*) über den folgenden 15 Gitterpunkten $(-2/-1)$, $(-2/0)$, ..., $(2/1)$ zeichnen, die aus den fünf x -Werten -2 , -1 , 0 , 1 , 2 sowie den drei y -Werten -1 , 0 und 1 gebildet werden können.

```
x = -2:1:2
y = -1:1:1
[xx, yy] = meshgrid(x,y)
```

```
x =
    -2  -1  0  1  2
```

```
y =
    -1  0  1
```

```
xx =
    -2  -1  0  1  2
    -2  -1  0  1  2
    -2  -1  0  1  2
```

```
yy =
    -1  -1  -1  -1  -1
     0   0   0   0   0
     1   1   1   1   1
```

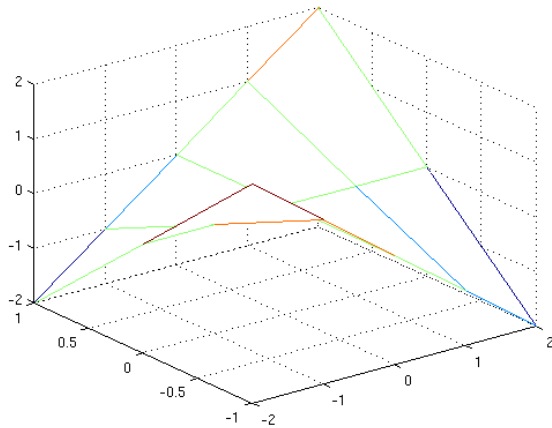
Von besonderer Bedeutung ist hier die Anweisung **meshgrid** (*mesh* = *Masche*, *grid* = *Gitter*). Es erzeugt zwei Matrizen, hier *xx* und *yy* genannt. Einander entsprechende Elemente dieser beiden Matrizen bilden gerade die beiden Koordinaten unserer Gitterpunkte. Anwendung der Punktmultiplikation ergibt:

```
z = xx.*yy
```

```
z =
     2   1  0  -1  -2
     0   0  0   0   0
    -2  -1  0   1   2
```

Dies sind die gewünschten Funktionswerte. Zuletzt soll jeder dieser Funktionswerte über "seinem" Gitterpunkt als Punkt dargestellt und in x - und in y -Richtung durch Strecken mit den Nachbarpunkten verbunden werden. Wir erhalten den *Maschenplot* oder das *Drahtmodell* der Funktion. Dies erfolgt mit Hilfe der Anweisung **mesh**:

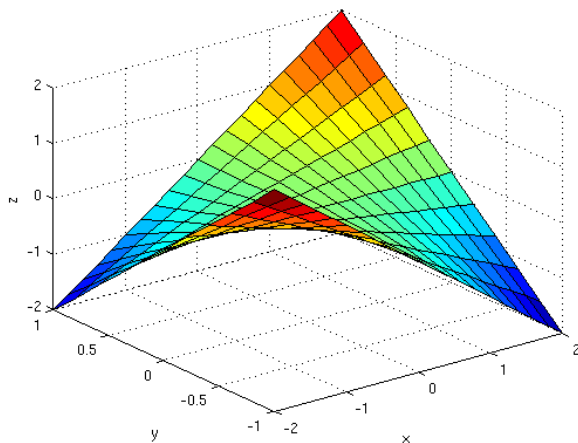
mesh(x,y,z)



x und y geben die x- bzw. die y-Werte der Gitterpunkte an, z die Funktionswerte über den Gitterpunkten an.

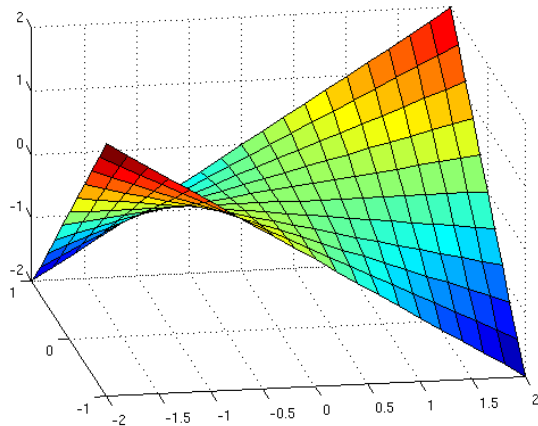
Statt die Punkte durch Strecken zu verbinden, können auch kleine Flächenstücke zwischen den Punkten verwendet werden. Man erhält so ein *Kachelmodell* der Funktion, indem man mesh durch **surf** (von *surface*) ersetzt. Im Folgenden ist die Fläche außerdem durch mehr Gitterpunkte feinmaschiger gezeichnet.

```
x = -2:0.2:2; y = -1:0.2:1;  
[xx,yy] = meshgrid(x,y,z);  
z = xx.*yy;  
surf(x,y,z); xlabel('x'); ylabel('y'); zlabel('z')
```



Mit **view(az,el)** kann die Blickrichtung auf den Plot gewählt werden: az gibt den sogenannten *Azimuthwinkel* (=Winkel in der xy-Ebene zur x-Achse), el den *Höhenwinkel* (Winkel über der xy-Ebene) an. Beide Winkel werden in Grad angegeben. Voreinstellung: az = -37,5°, el = 30°. view(0,90) ergibt beispielsweise einen zweidimensionalen Plot. Zur Grundeinstellung kehrt man durch view(3) zurück.

`surf(x,y,z); view(-10, 25);`



Es gibt noch weitere Anweisungen für dreidimensionale Grafiken, auf die hier aber nicht mehr eingegangen werden kann.

Übung 14: Lineare Funktion

Stellen Sie die lineare Funktion $z = -0,4 \cdot x - 0,8 \cdot y + 3$ für $0 \leq x \leq 5$ und $0 \leq y \leq 5$ graphisch dar. Um welche Art von Fläche handelt es sich dabei?

Übung 15: Dreidimensionaler Plot

Stellen Sie die Funktion $z = x^2 + y^2$ graphisch für $-3 \leq x \leq 3$ und $-3 \leq y \leq 3$ graphisch dar.

4. m-Files: MATLAB als Programmiersprache

Man unterscheidet zwei Arten von **m-Files**: *Script-Files* und *Function-Files*. Beide werden mit der Endung `.m` abgespeichert.

Script-Files

Im Übungsbeispiel 12 des vorangehenden Kapitels haben wir eine Wurfparabel gezeichnet. Angenommen, wir möchten nun weitere Wurfparabeln mit neuen Anfangsbedingungen (verschiedene Anfangsgeschwindigkeiten und Abwurfwinkel) zeichnen. Um nicht jedesmal von Neuem alle Befehle im Commandfenster eingeben zu müssen, wollen wir ein Script-File schreiben.

Ein **Script-File** (auch: *Script m-File* oder kurz *Script*) ist eine Aneinanderreihung von MATLAB-Anweisungen, die **in einen Texteditor eingegeben** und **mit der Endung `.m` abgespeichert** werden. Wird der Filename im Commandfenster nach dem Prompt eingegeben, so werden die Befehle im Script-File der Reihe nach ausgeführt, so, als ob sie einzeln nach dem Prompt eingegeben werden.

Bei seinem Aufruf stehen dem Script-File **alle im Workspace gespeicherten Variablen** zur Verfügung. Umgekehrt sind alle Variablen, die im Script-File definiert werden, nach dessen Ablauf noch im Workspace definiert (dh., die Variablen des Script-Files sind **global**).

Beispiel : Wurfparabel

Wir wollen ein Script-File schreiben, das bei Aufruf die Wurfparabel zeichnet. Die Anfangsgeschwindigkeit `v_kmh` und den Abwurfwinkel `phi_grad` soll das Script-File **aus dem Workspace übernehmen**.

Um ein Script-File zu erzeugen, wählt man in der Menüzeile NEW und dann SCRIPT-FILE. Es öffnet sich daraufhin ein Texteditor-Fenster. (Man kann natürlich auch einen beliebigen anderen Texteditor verwenden.) Hier geben wir die Befehle ein, die MATLAB ausführen soll:

```
% Plot einer Wurfparabel
% übernimmt aus Workspace: Abwurfgeschwindigkeit v_kmh,
% Abwurfwinkel phi_grad
g = 9.81;
v_ms = v_kmh/3.6; % Umrechnung in m/s
phi_rad = phi_grad*pi/180; % Umrechnung ins Bogenmaß
wurfzeit = 2*v_ms*sin(phi_rad)/g;
t = 0:0.01:wurfzeit;
x = v_ms*t*cos(phi_rad);
y = v_ms*t*sin(phi_rad) - 0.5*g*t.^2;
plot(x,y); grid
```

Nun speichern wir dieses File mit der Endung `.m` ab, zum Beispiel als **wurf.m**. Jetzt kehren wir ins Commandfenster zurück. Bevor wir das Script-File aufrufen, müssen wir noch `v_kmh` und `phi_grad` im Workspace definieren:

```
v_kmh = 100; phi_grad = 60;
```

Geben wir nun `wurf` ein, so führt MATLAB die Befehle in `wurf.m` nacheinander aus, so, als ob sie im Commandfenster nach dem Prompt eingegeben werden:

`wurf`

Achtung: Bei Eingabe von `wurf` **durchsucht** MATLAB das aktuelle Verzeichnis und alle Verzeichnisse im **MATLAB-Suchpfad** (dieser kann mit `path` ausgegeben werden). Wenn Sie die Fehlermeldung

`wurf`

??? Undefined function or variable 'wurf'.

erhalten, so müssen Sie

- entweder in das Verzeichnis, in dem sich `wurf.m` befindet, wechseln (Anweisungen dazu: `pwd`, `dir`, `cd`) oder
- -den vollständigen Pfad von `wurf.m` eingeben oder
- -das Verzeichnis, in dem `wurf.m` sich befindet, dem Suchpfad hinzufügen (verwenden Sie `help path`)

Eine Liste aller im aktuellen Verzeichnis enthaltenen m-Files (sowie mex-Files und mat-Files) erhält man mit `what`. Kennt man den Namen des gewünschten m-Files, weiß aber nicht, wo es sich befindet, so hilft der Befehl `which` (allerdings nur, wenn das Verzeichnis des Files im Suchpfad enthalten ist):

`which wurf`

`~/Matlab/wurf.m`

Wir erhalten den Pfad zum m-File `wurf.m`.

Tipp: Zum besseren Verständnis sollten die ersten paar Zeilen eines Script-Files **Kommentarzeilen** sein, die den Zweck und die Arbeitsweise des Script-Files beschreiben. Die Kommentarzeilen haben noch eine andere nützliche Funktion: sie **werden ausgegeben, wenn der help-Befehl verwendet wird**. So erhalten wir die ersten Kommentarzeilen von `wurf.m` mit

`help wurf`

`Plot einer Wurfparabel
übernimmt aus Workspace: Abwurfgeschwindigkeit v_kmh,
Abwurfwinkel phi_grad`

Übung 1: Lissajous-Figuren

Lissajous-Figuren entstehen, wenn sich zwei Schwingungen überlagern, die *senkrecht aufeinander* stehen und deren Frequenzen *ein rationales Verhältnis* zueinander haben. Werden z.B. gleichzeitig eine Spannung $x = A \sin(\omega_1 t)$ in x-Richtung und eine Spannung $y = B \cos(\omega_2 t)$ in y-Richtung angelegt, so wird die augenblickliche Lage eines Punktes (bzw. Elektronenstrahls) in der xy-Ebene beschrieben durch die Parametergleichungen:

$$x = A \sin(\omega_1 t), y = B \cos(\omega_2 t), t \geq 0.$$

Schreiben Sie ein Script-File, das nach Eingabe der Amplituden A und B sowie der Frequenzen ω_1 und ω_2 die zugehörige Lissajous-Figur zeichnet!

Interaktive Eingabe

Die Anweisungen **input**, **disp**, und **num2string** ermöglichen es, **Daten interaktiv einzugeben**.

Möchten wir einfach einen **Text** im Commandfenster **ausgeben**, so geben wir (im Script-file oder direkt ins Commandfenster) **disp('text')** ein:

```
disp('Guten Tag')
```

```
Guten Tag
```

Auch der **Wert einer Variablen** kann auf diese Weise ausgegeben werden:

```
a = 65/5; disp(a)
```

```
13
```

Die Eingabe von **disp(' ')** erzeugt keine Ausgabe (im Commandfenster) bzw. eine **Leerzeile** (im m-File).

Möchten wir den Benutzer auffordern, einer Variablen einen Wert zuzuordnen, so verwenden wir **input**. Die Anweisung

```
w = input('Geben Sie die Kreisfrequenz w ein: ')
```

```
Geben Sie die Kreisfrequenz w ein: 2
```

```
w =  
2
```

verwandelt das Prompt in den Satz "Geben Sie die Kreisfrequenz ein". Dieses Prompt **wartet auf die Eingabe einer Zahl**, welche der Variablen *w* zugeordnet wird.

Soll keine Zahl, sondern ein **Buchstabe** oder allgemein eine Zeichenkette (*string*) **ingegeben** werden, so ist der **Zusatz 's'** notwendig:

```
antwort = input('Wollen Sie abbrechen? (j/n) ','s')
```

```
Wollen Sie abbrechen? (j/n) j
```

```
antwort =  
j
```

Hier wird das Prompt in den Satz "Wollen Sie abbrechen? (j/n)" verwandelt. Es wartet auf die **Eingabe irgendeines Buchstaben**. Diese Zeichenkette wird der Variablen-antwort zugeordnet.

Beispiel zur interaktiven Eingabe: Überlagerung von zwei Sinusschwingungen

Wir wollen ein Script-File *schwing.m* schreiben, das zwei gleichfrequente Sinusschwingungen im Intervall $[-\pi, \pi]$ graphisch überlagert. Bei Aufruf von *schwing.m* soll zunächst nach der gemeinsamen Frequenz, den Amplituden und den Nullphasenwinkeln der beiden Schwingungen gefragt werden.

```
% Plot zweier gleichfrequenter Sinusschwingungen  
clc % löscht das Commandfenster  
disp('Graphische Überlagerung zweier gleichfrequenter Schwingungen ');  
disp(' ')  
w = input('Kreisfrequenz = ');  
A1 = input('Amplitude der ersten Schwingung = ');  
A2 = input('Amplitude der zweiten Schwingung = ');  
phi1 = input('Phasenkonstante der ersten Schwingung = ');
```

```

phi2 = input('Phasenkonstante der zweiten Schwingung = ');
x = (-1:0.01:1)*pi;
y1 = A1*sin(w*x + phi1);
y2 = A2*sin(w*x + phi2);
y = y1 + y2;
plot(x,y1,'g', x,y2,'b',x,y,'r');
title(['Überlagerung zweier Schwingungen der Frequenz w = ', num2str(w)])
grid

```

Wir haben die Anweisung **title('text')** schon kennengelernt. Sie gibt den Text *text* als Titel einer Grafik aus. Besteht der **Titel aus mehreren Strings**, so müssen sie in einem **Vektor** (d.h. zwischen eckigen Klammern) zusammengefaßt werden: **title(['text1', 'text2',...])**.

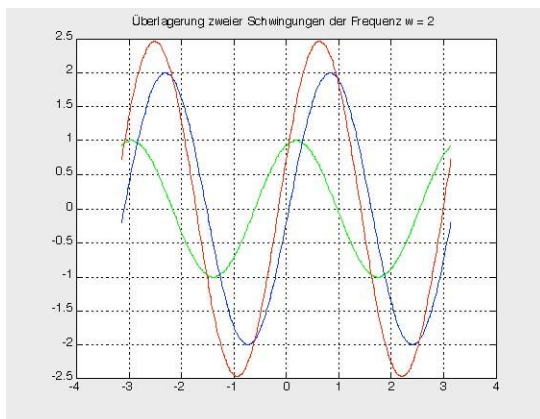
Die Anweisung **num2str(3)** (Abkürzung für *number to string*) wandelt die Zahl 3 in den String 3 um. Da das Argument von title nur aus Strings bestehen kann, muß hier zum Beispiel der numerische Wert von w mit num2str in einen String umgewandelt werden.

Bei Aufruf von schwing.m passiert nun folgendes:

schwing

Graphische Überlagerung zweier gleichfrequenter Schwingungen

Kreisfrequenz = 2 Amplitude der ersten Schwingung = 1 Amplitude der zweiten Schwingung = 2 Phasenkonstante der ersten Schwingung = 1.2 Phasenkonstante der zweiten Schwingung = -0.1



Tipp: Die **Ausführung eines Script-Files** im Commandfenster kann mit der Tastenkombination **Strg-C unterbrochen** werden.

Übung 2[opt]: Lissajous-Figuren (interaktiv)

Schreiben Sie ein Script-File, das nach interaktiver Eingabe Lissajous-Figuren zeichnet! Es sollen die Amplituden und Kreisfrequenzen der beiden Schwingungen eingegeben werden. Im Titel der Grafik soll das Frequenzverhältnis der beiden Schwingungen angegeben sein.

Übung 3: Darstellung von Funktionsgraphen

Schreiben Sie ein Script-File, das einen beliebigen Funktionsgraphen zeichnet. Es sollen interaktiv eingegeben werden: die untere und obere Grenze des Definitionsbereiches, die Funktion, die zu zeichnen ist, und der Titel der Grafik.

Function-Files

Über *Function-Files* (auch: *Function m-Files*) können wir in MATLAB **neue Funktionen definieren**. Ein Function-File wird, wie auch ein Script-File, mit der Endung `.m` abgespeichert. Der Unterschied zu einem Script-File besteht darin, dass die im Function-File definierten Variablen **lokal** sind.

Beim Aufruf eines Function-Files wird ihm ein Wert (oder mehrere) als Argument übergeben (wie bei `sin(x)`, hier wird der Wert von `x` übergeben). Nach Auswertung der Befehle im Function-File wird ein Wert (oder mehrere) ausgegeben, der Funktionswert.

Wir wollen zum Beispiel eine Funktion definieren, die einen Winkel `x` vom Bogenmaß ins Gradmaß umrechnet. Die Funktion soll etwa mit `r2d(x)` (**radiant to degree**) bezeichnet werden:

```
function d = r2d(x)
% r2d(x) rechnet einen Winkel x vom Bogenmaß ins Gradmaß um
d = x*180/pi;
```

Wir speichern das File unter dem Namen `r2d.m` ab.

Wichtig: Der **Funktionsname** (Name in der ersten Zeile des Function-Files) muß immer **gleich** sein wie der **Name, unter dem das File abgespeichert** wird.

Nun können wir die Funktion `r2d(x)` bereits verwenden. Um zum Beispiel `x = 3.14` vom Bogenmaß ins Gradmaß umzurechnen, brauchen wir nur im Commandfenster einzugeben:

```
r2d(3.14)
ans =
    179.9087
```

Natürlich kann das Argument `x` auch ein Vektor sein:

```
x = (0:0.5:2)*pi; r2d(x)
ans =
    0 90 180 270 360
```

Ein Function-File hat also die folgende Struktur:

```
function d = Funktionsname(Argumentliste)
% Kommentare
Anweisungen
d =...;
```

Es beginnt immer mit dem Wort **'function'**. Danach folgt eine Variable (hier `d`), der (irgendwo) im Anweisungsteil der **Funktionswert** zugewiesen wird (hier: `d = x*180/pi`).

Der Funktionswert kann auch ein Vektor sein:

```
function d = Funktionsname(Argumentliste)
% Kommentare
Anweisungen
a =...; b =...; ...
d = [a,b,..];
```

Die auf die erste Zeile folgenden **Kommentarzeilen** haben dieselbe Bedeutung wie bei einem Script-File. Sie werden ausgegeben, wenn man die `help`-Anweisung verwendet:

```
help r2d
r2d(x) rechnet einen Winkel x vom Bogenmaß ins Gradmaß um
```

Die **allererste** Kommentarzeile wird auch von der **lookfor**-Anweisung durchsucht.

Wie schon zu Beginn erwähnt, sind alle im Function-File definierten Variablen **lokal**. Nach Auswertung der Anweisungen im Function-File sind sie also im Workspace unbekannt. Umgekehrt kann das Function-File auch auf keine Variablenwerte aus dem Workspace zugreifen (außer auf die im Argument übergebenen).

Am besten sehen wir uns das anhand eines Beispiels an. Definieren wir etwa die Funktion f.m

```
function y = f(x)
% Parabel um c nach oben/unten verschoben
c = 3;
y = x^2 + c;
```

Werten wir nun die Funktion an der Stelle x = 2 aus:

```
f(2)
ans =
    7
```

Es wird - wie erwartet - der Funktionswert mit dem Wert c = 3 berechnet. Wir können aber mit der Variablen c im Workspace nicht weiterarbeiten, sie ist unbekannt:

```
c
```

??? Undefined function or variable 'c'.

Geben wir nun der Variablen c im Workspace den Wert c = 1 und werten wir die Funktion wieder an der Stelle x = 2 aus:

```
c = 1; f(2)
ans =
    7
```

Für die Auswertung der Funktion wird also nach wie vor der Wert c = 3 verwendet! Das "c" im Workspace hat mit dem "c" im Function-File nichts zu tun!

Um c vom Workspace aus ändern zu können, müssen wir es **als globale Variable definieren**. Das müssen wir **sowohl im Workspace als auch im Function-File** tun. Die hier notwendige MATLAB Anweisung heißt **global**. Wir schreiben also einerseits das Function-File um,

```
function y = f(x)
% Parabel um c nach oben/unten verschoben
global c
y = x^2 + c;
```

und definieren auch im Commandfenster c als globale Variable:

```
global c;
c = 1; f(2)
ans =
    5
```

Nun verwenden Workspace und Function-File dasselbe "c".

Achtung, globale Variablen dürfen wegen ihrer Fehleranfälligkeit nur in reinen Skripten und nicht in Function-m Files verwendet werden!

Tipp: Matlab unterstützt ebenso wie andere Programmiersprachen Debugging. Unter

dem Menü „Debug“ können Breakpoints, bedingte Breakpoints, Einzelschritt, Variablenüberwachung, etc. benutzt werden.

Viele von MATLAB zur Verfügung gestellten Funktionen sind selbst m-Files (im Gegensatz zu bereits im MATLAB-Prozessor eingebauten Funktionen wie sin(x)). Ein Beispiel ist die Funktion **angle(x)**. Den **Pfad des** zugehörigen **m-Files** erhalten wir mit der Anweisung **which**

which angle

</opt/matlabR2009a/toolbox/matlab/elfun/angle.m>

Den Code von angle können wir mit

edit angle

lesen (es öffnet sich ein Editor-Fenster mit den Anweisungen zu angle.m).

IF-Anweisungen und Schleifen:

Sie können in Ihrem m-File auch Kontrollstrukturen einbauen. MATLAB stellt dazu die **IF-Anweisung** sowie **FOR- und WHILE -Schleifen** zur Verfügung. Für nähere Informationen zu ihrer Verwendung siehe Handbuch oder help if, help for, help while... Man kann die Anweisungen genauso wie in jeder anderen Programmiersprache verwenden.

Tipp: Man sollte nach Möglichkeit Matrizen anstelle von Schleifen verwenden, da damit Berechnungen bis um den Faktor 10 schneller ablaufen. Das folgende Beispiel ist der Einfachheit halber trotzdem als Schleife programmiert. Die Programmierung als Matrix befinden sich in der Datei `ministatFast.m`, vergleichen sie einfach einmal die Programmierweise.

Beispiel: Mittelwert und Standardabweichung einer Stichprobe

Das folgende Programm ermittelt den Mittelwert \bar{x} und die Standardabweichung s einer Stichprobe:

$$\bar{x} = \frac{1}{n} * \sum_{i=1}^n x_i, \quad s = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^n (x_i^2 - n \cdot \bar{x}^2)} = \sqrt{\frac{1}{n-1} \cdot \left(\sum_{i=1}^n x_i^2 - \frac{1}{n} \cdot \left(\sum_{i=1}^n x_i \right)^2 \right)}$$

```
function [mittel, sdev] = ministat(x)
%Mittelwert und Standardabweichung einer Stichprobe x
n = length(x); % Anzahl der Komponenten des Vektors x
sum = 0; sumq = 0;
for k = 1:n

    sum = sum+x(k);
    sumq = sumq + x(k)*x(k);

end;
mittel = sum/n;
sdev = sqrt(1/(n-1)*(sumq-sum*sum/n));
```

Wir speichern das m-File unter ministat.m ab. Im Commandfenster definieren wir zunächst einen Vektor v und rufen danach das m-File mit dem Argument v auf:

```
v = [2 1 0 3 2];  
[m s] = ministat(v)
```

```
m =  
    1.6000  
s =  
    1.1402
```

Übung 4[opt]: Hypotenuse eines rechtwinkligen Dreiecks

Schreiben Sie eine Funktion hyp(a,b), die die Länge der Hypotenuse eines rechtwinkligen Dreiecks nach dem Lehrsatzes des Pythagoras berechnet. Es sollen die Längen der Katheten a und b eingegeben werden. Schreiben Sie die Funktion so, dass a und b auch Vektoren sein können.

Übung 5: Winkel zwischen zwei Vektoren

Schreiben Sie eine Funktion winkel(u,v), die den Winkel zwischen zwei Vektoren u und v im Gradmaß berechnet. Die Vektoren sollen als Zeilenvektoren eingegeben werden.

Der Winkel berechnet sich aus: $\phi = \arccos\left(\frac{u \cdot v}{|u| \cdot |v|}\right)$. Die Länge |u| eines Vektors u

erhalten Sie in MATLAB mit der Anweisung norm(u), die Arcuskosinusfunktion mit acos(x). Beachten Sie, dass acos(x) den Winkel im Bogenmaß ausgibt.

Übung 6[opt]: Polarkoordinaten einer komplexen Zahl

Schreiben Sie eine Funktion pol(a,b), die die Polarkoordinaten (r,phi) der komplexen Zahl $z = a + b \cdot j$ ausgibt. Dabei ist r der Betrag von z und phi das Argument im Gradmaß.

Tipp: Batchbetrieb

Matlab kann auch ohne grafische Oberfläche und ohne JavaEngine gestartet werden. Ohne beide Programmteile laufen Berechnungen bis zu doppelt so schnell ab. Dieser Modus ist somit ideal um lange Berechnungen zu starten. Für die Entwicklung von Algorithmen ist er dagegen nicht geeignet.

Der Befehl um Matlab so zu starten (unter Linux):

```
matlab -nojvm -nodesktop
```

Höhere Datenstrukturen

Neben Werte, Vektoren und Matrizen gibt es noch zwei weitere Datenstrukturen in Matlab. Da sie nicht so oft benötigt werden, werden wir nur kurz auf sie eingehen und ansonsten auf die Matlab-Hilfe verweisen.

Strukturen

Eine Struktur speichert eine Ansammlung von verschiedenen Datentypen. Alle Inhalte sind mittels eines festen Namens ansprechbar.

```
network = struct('M', {10}, ...      % Number of features
                'C', {5}, ...       % Number of Clusters
                'NodeList', {[]}, ... % Nodeslist (like in a graph) as a cell arry
                'maxNumCategories', {100})
```

```
network =
  M: 10
  C: 5
  NodeList: []
  maxNumCategories: 100
```

```
network.M
```

```
ans =
  10
```

Cellarray

Zellen können gemischte Daten abspeichern, ohne das ihre Inhalte einen explizieten Namen besitzen. Cellarrays werden oft u.a. für die Abspeicherung von Strings verwendet.

```
X = cell(1, 4);
```

```
X(1:3) = {'SimpleCluster', 'K-Means', 'SVM'};
```

```
X
```

```
X =
  'SimpleCluster'    'K-Means'    'SVM'    []
```

```
X{1}
```

```
ans =
  SimpleCluster
```

```
X(1)
```

```
ans =
  'SimpleCluster'
```

Bei der Indizierung $X(1)$ wird die erste Zeile von C in Form einer Cell zurückgegeben. Dagegen gibt $X\{1\}$ die Elemente der ersten Zeile von C aus.

5. Verzeichnis nützlicher Funktionen

Im folgenden eine kurze Auflistung nützlicher Funktionen. Für die Benutzung wird die Matlab-Hilfe empfohlen. Der erste Teil enthält allgemeine Funktionen, danach für Bildverarbeitung und abschließend Funktionen speziell für KI/Gehirnforschung.

Funktion	Bezeichnung
set	Setzt Attribute von Grafiken, Achsen, Plots, etc.
print	Textausgabe auf die Konsole
sprintf	Formatierte Ausgabe in einen String
reshape	Wandelt die Form einer Matrix um
squeeze	Entfernt Dimensionen der Größe 1
imread, imresize	Funktion um Bilder einzulesen
imread	Funktion um Bilder zu manipulieren
lsqnonlin	Nicht lineare Datenapproximation
normpdf	Erzeugt eine Normalverteilung
princomp	PCA (Hauptkomponentenanalyse)
fastica	ICA (Independent Component Analysis) unter Benutzung der externen FastICA Bibliothek
gmmem	EM-Algorithmus. Die externe Bibliothek NetLab enthält eine Sammlung von Optimierungsalgorithmen, zB. EM.