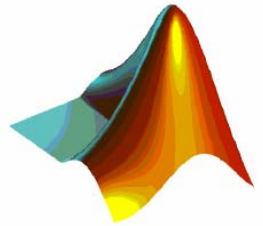
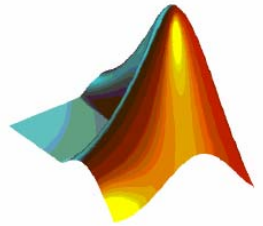


Mathematik (Teil II)

- Einführung in MATLAB -

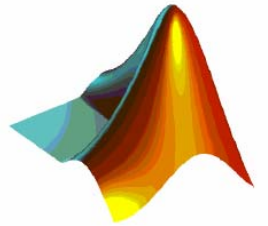
Harald Loose
FH Brandenburg

Inhalt



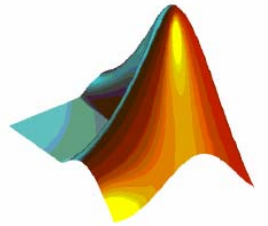
- Einführung in MATLAB
  - Überblick und Grundlagen
  - Ein- und Ausgabe
- Mathematik
  - Matrizen (und Vektoren)
  - Lineare Gleichungssysteme
  - Interpolation und Extrapolation
  - Approximation
  - Numerisch Differenzieren und Integrieren
  - Fourier-Transformation (in MATLAB)

Ablauf



- Einführung in MATLAB 21.04. / 9-14 Uhr
  - Hausaufgaben
- Mathematik I 28.04. / 9-14 Uhr
  - Hausaufgaben
- Mathematik II 02.05. / 9-12 Uhr
  - Hausaufgaben

Literatur (kleine Auswahl)



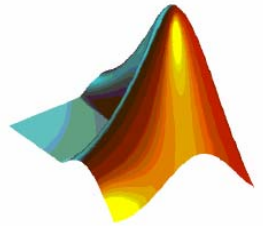
- W. Schweizer: MATLAB kompakt, Oldenbourg, 2005
- A. Angermann, M. Beuschel, M.Rau, U. Wohlfarth: Matlab-Simulink-Stateflow, Oldenbourg, 2005 (Folien- und Übungssatz)
- MATLAB®: The Language of Technical Computing

  Version 7.1

  The MathWorks

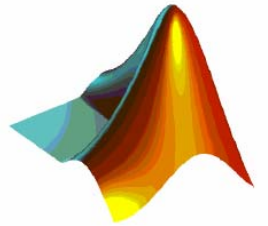
  – Getting Started with MATLAB
  – Mathematics
  – Data Analysis
  – Programming
  – Graphics

What is MATLAB?



- high-performance language for technical computingcomputation, visualization, and programming
- Typical application fields:
  - Math and computation
  - Algorithm development
  - Data acquisition
  - Modeling, simulation, and prototyping
  - Data analysis, exploration, and visualization
  - Scientific and engineering graphics
  - Application development, including graphical user interface building

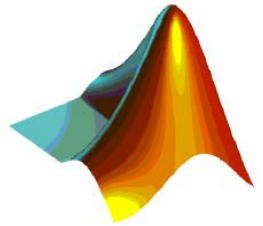
What is MATLAB?



- The name MATLAB stands for *matrix laboratory*.

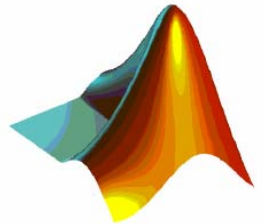
- originally to provide easy access to matrix software developed by the LINPACK and EISPACK projects
- Today engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation

History (wikipedia)



- MATLAB was invented in the late 1970s by Cleve Moler, then chairman of the computer science department at the University of New Mexico. He designed it to give his students access to LINPACK and EISPACK without having to learn Fortran. It soon spread to other universities and found a strong audience within the applied mathematics community. Jack Little, an engineer, was exposed to it during a visit Moler made to Stanford University in 1983. Recognizing its commercial potential, he joined with Moler and Steve Bangert. They rewrote MATLAB in C and founded The MathWorks in 1984 to continue its development. These rewritten libraries were lovingly known as JACKPAC. MATLAB was first adopted by control design engineers, Little's specialty, but quickly spread to many other domains. It is now also used in education, in particular the teaching of linear algebra and numerical analysis.

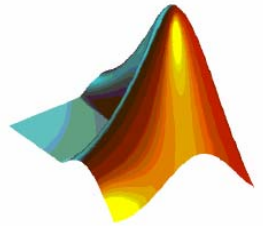
Main parts



- Desktop Tools and Development Environment
- MATLAB Mathematical Function Library
- MATLAB Language
- Graphics
- MATLAB External Interfaces/API

- and ….

Toolboxes I



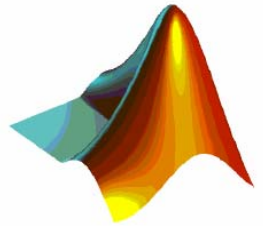
MATLAB Products

- MATLAB
- MATLAB Compiler &
- C/C++ Math Library
- MATLAB C/C++ Graphics Library
- Database Toolbox
- Data Acquisition Toolbox
- Excel Link
- MATLAB Report Generator
- MATLAB Runtime Server
- MATLAB Web Server
- MatrixVB

Design Automation Products

- Simulink
- Real-Time Workshop
- Stateflow
- Stateflow Coder
- CDMA Reference Blockset
- DSP Blockset
- Communications Toolbox
- Motorola DSP Developer's Kit
- Dials & Gauges Blockset
- Power System Blockset
- Fixed Point Blockset
- Simulink Report Generator
- Real-Time Workshop ADA Coder
- Real-Time Windows Target
- Requirements Management Interface
- xPC Target

Toolboxes II



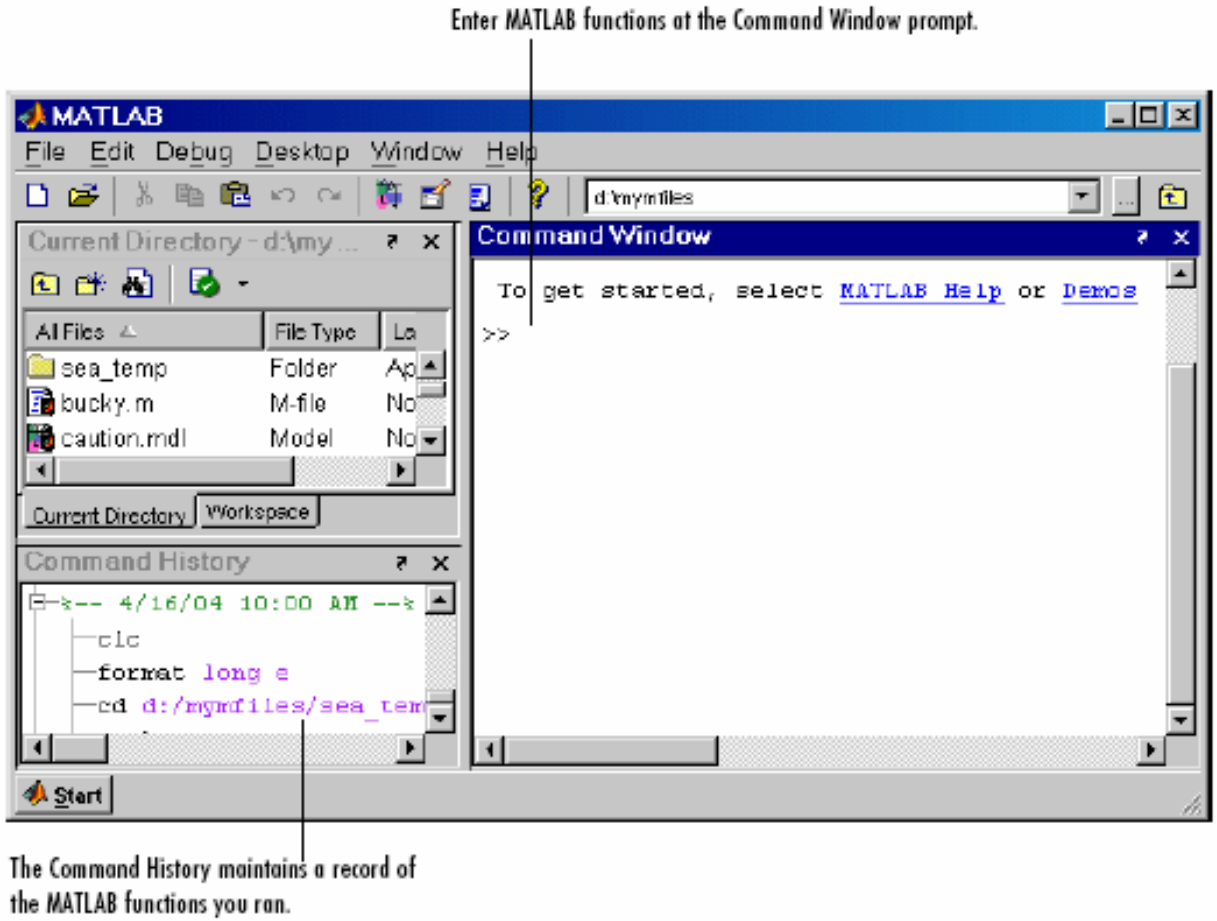
Application Toolboxes

Signal & Image Processing
- Signal Processing
- Image Processing
- Wavelet
- Higher-Order Spectral Analysis
- Quantized Filtering

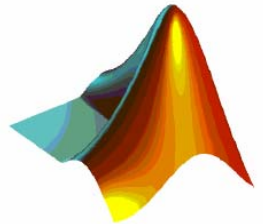
Control Design
- Control System
- Fuzzy Logic
- Robust Control
- Nonlinear Control Design Blockset
- System Identfication
- μ-Analysis and Synthesis
- LMI Control
- Model Predictive Control
- QFT Control Design

General
- Optimization
- Statistics
- Neural Network
- Symbolic/Extended Symbolic Math
- Partial Differential Equation
- Financial
- Financial Derivatives
- Financial Time Series
- GARCH
- Mapping
- Spline
- NAG Foundation
- Datafeed

MATLAB Desktop

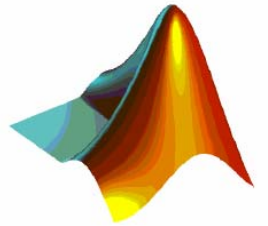


Integrierte Entwicklungsumgebung



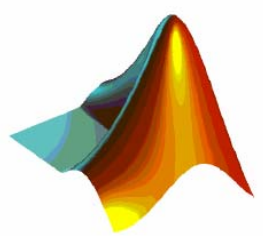
- Command Window
- Editor
- Command History
- Workspace Browser
- Launch Pad
- Current Directory Browser
- Profiler
- Cell Mode

Online-Hilfe



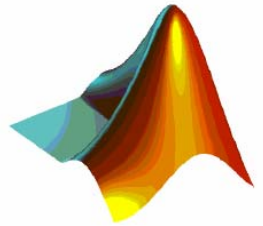
- Aufruf der Hilfe zu einem Befehl:
  help [befehl]
- Eigener Hilfe-Browser:
  helpwin [befehl]
  doc [befehl]
- Suche nach Ausdruck *suchstring*:
  lookfor *suchstring*

- Handbücher als PDF-Dateien vom Helpdesk aus
  (www.mathworks.com Q:\LOOSE\MATLAB)



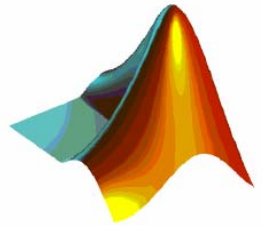
Grundlagen

Variablen



- Namen:
  - Maximal 32 Zeichen
  - Buchstaben, Unterstrich _ und Zahlen
  - Erstes Zeichen ein Buchstabe
  - Unterscheidung von Groß- und Kleinschreibung
- Zuweisung von Werten an Variable:
  v1 = 25, a = [2 5 7; 1 8 3],
- Standardergebnisvariable ans
- Variablen i.d.R. global im Workspace definiert
- Typvereinbarung ist nicht notwendig

Variablen-Typen



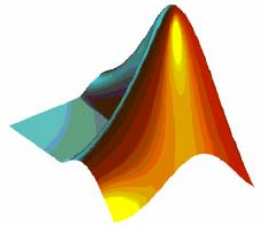
verschiedene Typen (Klassen) von Variablen möglich:

- double, boolean, char, sparse, storage, cell, struct
- double und char am meisten verwendet

für spezielle Aufgaben:

- sparse schwachbesetzte zweidimensionale Matrizen
- storage int8, int16, int32, uint8, uint16, uint32
- struct Zusammenfassung von Daten
- cell Programmierung großer Systeme

Ausgeben, Grundrechenarten und Konstanten



- Komma oder Leerzeichen nach der Operation oder Anweisung lassen die Ausgabe zu, der Strichpunkt unterdrückt sie
- Rechnen:

+	Addition	–	Subtraktion
*	Multiplikation	/	Division
^	Potenzieren		

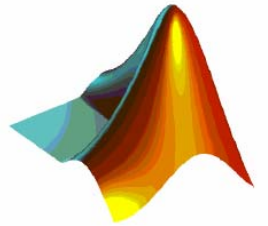
- Konstanten:

pi	Kreiszahl Pi	eps	Fließkomma-Genauigkeit
inf	Unendlich	NaN	Not-a-Number

- Komplexe Zahlen:

i, j	Imaginäre Einheit

Vektoren und Matrizen



- Vektoren: Trennung der Elemente durch Komma oder Leerzeichen:

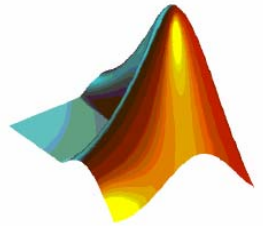
  vektor = [1 2 3]

- Matrizen: Trennung der Zeilen durch Strichpunkt:

  matrix = [1 2 3 ; 5 6 7]

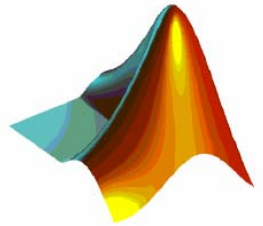
- Erstes Element hat Index 1
- Doppelpunkt für Zeile/Spalte: zeile 1 = matrix(1,:)
- Zusammensetzen: matrix = [vektor ; 5 6 7]
- Letztes Element: matrix(:,end)
- Reshape array: B = reshape(A,m,n)
  A(:) – matrix->vektor

Spezielle Vektoren und Matrizen



- Elemente mit gleicher Schrittweite: fort = 1:2:100
- Bestimmte Anzahl *anzahl* von Elementen innerhalb eines Intervalls mit Grenzen *start* und *ziel*:
  - Linear: linspace(start,ziel,anzahl)
  - Logarithmisch: logspace(start,ziel,anzahl)
- Spezielle Matrizen(m Zeilen, n Spalten):
  - Einheitsmatrix: eye(m)
  - Einsermatrix: ones(m,n)
  - Nullmatrix: zeros(m,n)
  - Zufallswertmatrix: rand(m,n)

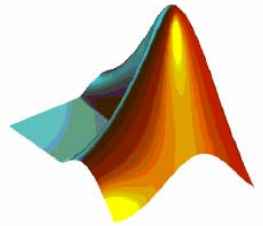
Mathematische Funktionen



- Viele mathematische & trigonometrische Funktionen

`sqrt`(x)	Quadratwurzel	`rem`(x,y)	Rest nach Division x/y
`exp`(x)	Exponentialfunktion	`round`(x)	Runden
`log`(x)	Natürlicher Logarithmus	`ceil` (x)	Rundet nach oben
`log10`(x)	Zehner–Logarithmus	`floor`(x)	Runden nach unten
`abs`(x)	Betrag	`sum`(v)	Summe der Vektorelemente
`sign`(x)	Signum (Vorzeichen)	`prod`(v)	Produkt der Vektorelemente
`real`(x)	Realteil	`min`(v)	kleinstes Vektorelement
`imag`(x)	Imaginärteil	`max`(v)	größtes Vektorelement
`phase`(x)	Phase einer komplexen Zahl	`mean`(v)	Arithmetisches Mittel
`sin`(x)	Sinus	`atan`(x)	Arcus–Tangens $\pm 90°$
`cos`(x)	Cosinus	`atan2`(x,y)	Arcus–Tangens $\pm 180°$
`tan`(x)	Tangens	`sinc`(x)	Spaltfunktion $\sin(\pi x)/(\pi x)$

Einige Beispiele



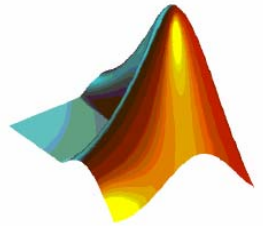
Vektor-Operationen

```
v = [ 3 4 2 9 5 ];
sum(v)          Summe der Elemente
prod(v)         Produkt der Elemente
mean(v)         Mittelwert der Elemente
```

Matrizen-Operationen

```
A = pascal(5)
sum(A)          Summe der Elemente je Spalte
prod(A)
mean(A)
sum(A(:))       Summe aller Elemente
prod(A(:))
mean(A(:))
```

Rechnen mit Vektoren und Matrizen



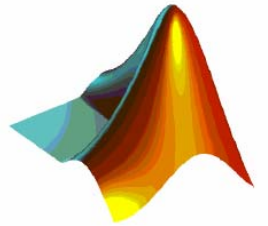
- Viele Operationen können auf Vektoren und Matrizen angewendet werden
- Element weises Ausführen mit dem Punktoperator

  a * b vs. a .* b

- Spezielle Vektoren- und Matrixfunktionen:

.'	Transposition	inv(x)	Inversion
det(x)	Determinante	rank(x)	Rang
eig(x)	Eigenwerte	'	Transposition (komplex konj.)

Strukturen und Cell Arrays



- Strukturen zum Verwalten von Daten verschiedenen Typs: Skalare, Matrizen, Strings, etc.

- Felder einer Struktur enthalten einen Wert:

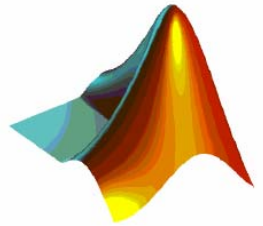
  str = struct ('name 1',wert 1,'name 2',wert 2,...)

- Zugriff auf Werte mit dem . Operator:

  str.name

- Cell Arrays: Multidimensionale Strukturen

Verwalten von Variablen



Dimension:

- eines Vektors: length(vektor)
- einer Matrix: size(matrix)

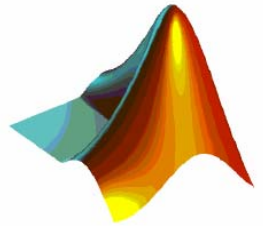
Anzeigen:

- who [variable]: Nur Namen
- whos [variable]: Namen, Größe, Bytes und Klasse

Löschen:

- einer Variablen: clear variable
- aller Variablen: clear, clear all

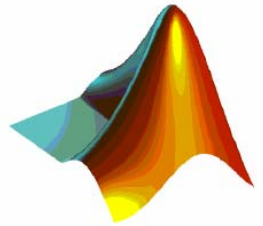
Vergleichsoperatoren



- Vergleichsoperatoren: ==, ~=, <, <=, >, >=,
- Test auf Existenz einer Variable x: exist(x)
- Vergleichsoperatoren können auf Skalare, Vektoren und Matrizen angewendet werden.
- Werte:

  Null (logisch falsch) bei Nichterfüllung der Operation, andernfalls Eins (logisch wahr)
- Auswerte-Reihenfolge:
  1. Mathematische Ausdrücke
  2. Vergleichsoperatoren von links nach rechts

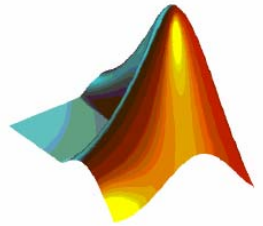
Logische Operatoren



- Logische Operatoren:
  ~ NOT, & AND, | OR, xor XOR

- Logische Operatoren können auf Skalare, Vektoren und Matrizen angewendet werden.
- Werte:
  Null ist logisch falsch, alle anderen Werte sind logisch wahr. Ergebnisse immer 0 oder 1.
- Auswerte-Reihenfolge:
  1. Mathematische Ausdrücke
  2. NOT
  3. UND und ODER von links nach rechts

Ablaufsteuerung: Verzweigungen



IF–Verzweigung

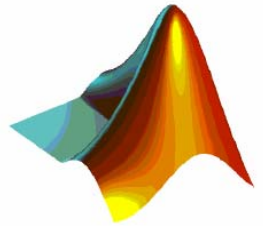
```
if ausdruck
      befehle
elseif ausdruck
      befehle
else
      befehle
end
```

SWITCH–Verzweigung

```
switch ausdruck
   case ausdruck
      befehle
   case ausdruck ausdruck ...
      befehle
   otherwise
      befehle
end
```

- Trennen der Ausdrücke und Befehle sinnvoll (, oder ,)
  Verschachtelungen von if und switch möglich

Ablaufsteuerung: Schleifen



- FOR:

```
for variable = ausdruck
    befehle
end
```

```
for n = 1:1:10 ,
    f = n^2,
end
```

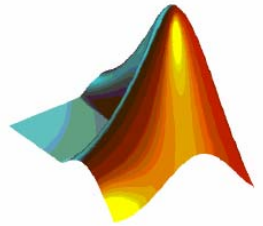
- WHILE:

```
while ausdruck
    befehle
end
```

```
while w > 1 ,
    w = w + 1 ;
end
```

- Schleife:
  - Überspringen: `continue`
  - Abbrechen: `break`

MATLAB-Skripts

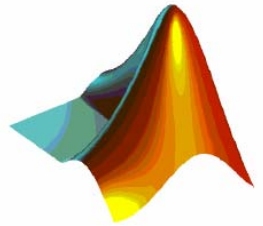


- M-File: Befehle in ASCII-Datei mit Endung .m

- Kommentar beginnt mit Prozentzeichen:
  % Kommentar
- *Cell* beginnt mit 2 Prozentzeichen:
  %% Dokumentations-Kommentar
- Umbruch innerhalb eines Befehls: ...

- Anzeigen der Befehle und Kommentare: echo on
- Seitenweise Ausgabe: more on
- Anzeigen der Datei: type datei

MATLAB-Funktionen I



- Sonderform der M-Files:
  - Übergabe von Parametern
  - Rückgabe von Werten
  - Lokale Variablen

- Definition (Dateiname == Funktionsname):

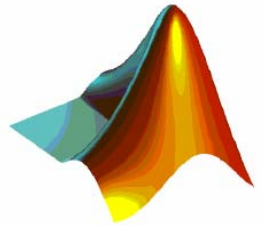
```
function [var] = functionname (par)
…….
end
```

- Aufruf:

```
[var] = functionname (par)
```

- Interne Hilfsgrößen: nargin, nargout
- Hilfetext in Funktionen: % Hilfetext

MATLAB-Funktionen II

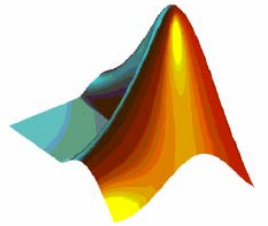


- Interne und private Funktionen sind möglich
- Nur in m-Files, die Funktionen definieren
- Variablen, die in Funktionen deklariert werden, sind lokal

- Datenübergabe an Funktionen:
  - Übergabe von Parametern
  - Rückgabe von Werten
  - Über applikationsspezifische Daten:

```
setappdata(0,'Daten',daten)
daten = getappdata(0,'Daten')
```

MATLAB-Funktionen III

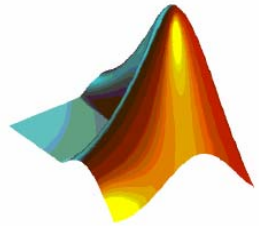


- Function Handle: f_handle = @funktion
- Informationen: functions(f_handle)
- Aufruf:

  [y1, …, ym] = feval (f_handle, x1,…, xn)

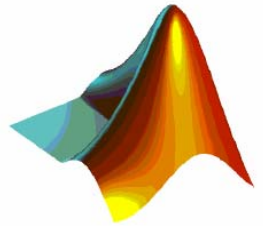
- Inline Functions: f = inline (funktion,var)
- Pseudo-Code: pcode (funktion)

- Entfernen aller Funktionen: clear functions



Ein- und Ausgabe

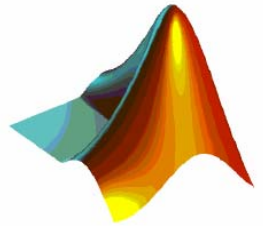
Steuerung der Bildschirmausgabe



Allgemein: *on* aktiviert, *off* deaktiviert Befehls-Funktion,
der Befehl alleine wechselt den Zustand

- Protokolldatei erstellen: diary, diary datei
- seitenweise Bildschirmausgabe: more, more(n)
- Befehle anzeigen/verbergen: echo
- Bildschirmausgabe anhalten: pause, pause(n)
- Bildschirm rücksetzen: clc

Eingabe über Benutzerdialoge

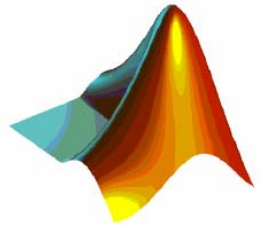


- Strings:
  - Zeilenvektoren aus Zeichen (char):
    text = ['Das ist',' ','ein Text!']
  - Funktionen für Strings: help strfun

- Eingabe von:
  - Daten: variable = input(string)
  - Text: string = input(string,'s')

- Sonderzeichen:
  \n Zeilenumbruch, \\ Backslash \, '' Anführungszeichen '

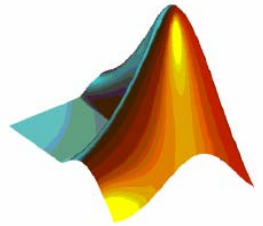
Formatierte Ausgabe



- Ausgabe des Strings string: disp(string)

- Formatierung:
  - string = sprintf(string,variable)
  - Syntax entspricht weitgehend C
  - Auch vektorisierte Daten möglich
  - Umwandlung von Zahlen in Strings mittels des Befehls
    num2str(variable[,format])

Import und Export von Daten

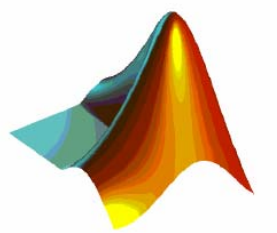


- Daten laden: load datei [-mat | -ascii] [variable …]
- Daten speichern: save datei [options] [variable …]
  - options:

-mat	Binäre MAT-Datei
-ascii	8-digit ASCII Format
-append	Daten anhÄangen (MAT)
-ascii -double	16-digit ASCII Format
-ascii -tabs	Mit Tabulator getrennt

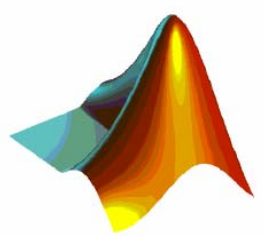
- Formatiertes Schreiben in Textdatei:
  - Datei öffnen: fid = fopen(datei.endung,zugriff)
  - Schreiben: fprintf(fid,string,variable)
  - Datei schließen: fclose(fid)

Betriebssystemaufruf und Dateiverwaltung



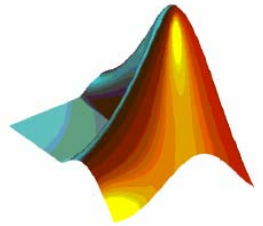
- Pfad:
  - aktuell: pwd
  - anzeigen: path
- Verzeichnis:
  - wechseln: cd verzeichnis
  - erstellen: mkdir verzeichnis
  - Inhalt anzeigen: dir [verzeichnis]
ls [verzeichnis]
- Datei:
  - kopieren: copyfile quelle ziel
  - löschen: delete datei

- Aufruf des Betriebssystems: ! os - befehl



Graphik

Graphik

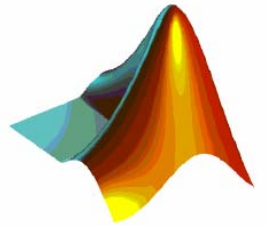


- 2- und 3-dimensionales Plotten von Daten möglich

- Erzeugen einer Graphik (Figure): figure
- Aktuelle Figurenummer (handle): gcf
- Unterplots in einer Figure: subplot (z,s,n)

- Figure:
  - rücksetzen: clf
  - löschen: delete (figure (nummer))
  - schließen: close (nummer)

Eigenschaften einer Graphik



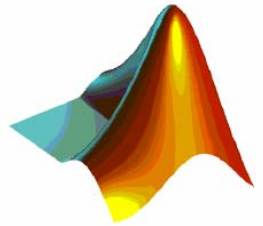
- Jedes Objekt einer Figure hat einen Handle h mit einer Eigenschaft Eig und dessen Wert EigWert

  Wichtige Objekte: Figure und Achsen

- Auslesen einer Eigenschaft: get(h,'Eig')
- Setzen einer Eigenschaft: set(h,'Eig',EigWert)
- Löschen des Wertes: delete(h)

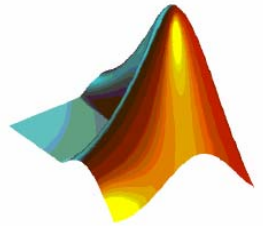
- Alternativ: Property Editor

2D-Graphik: Achsen, Skalierung und Beschriftung



- Aktuelle Achsennummer (handle): gca
- Skalierung: axis ([x_min,x_max,y_min,y_max])
  - Automatisch: axis ('auto')
  - Gitternetz einblenden: grid [on | off]
  - Zoomfunktion aktivieren: zoom [on | off]
- Beschriftung:
  - Achse: xlabel (string), ylabel (string)
  - Überschrift: title (string)
  - Text platzieren: text (x_wert,y_wert,string)
  - Legende: legend (str_1,str_2,… [, position])

2D-Graphik: Plotbefehle



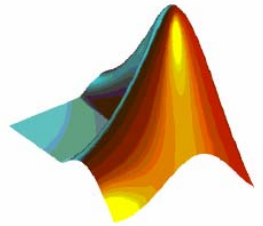
- Plot-Befehl: plot (x_werte,y_werte,… [, *plotstil*])
  x_werte, y_werte: gleich lange Vektoren
- Vorhandene Objekte beibehalten: hold [on| off]
- Farben und Stil der Linien mit *plotstil* einstellbar:

Farben			
k	schwarz	r	rot
b	blau	m	mangenta
c	cyan	y	gelb
g	grün	w	weiß

Linien und Punkte			
-	durchgezogen	o	Kreise
--	gestrichelt	*	Sterne
:	gepunktet	+	Kreuze
.	Punkte	x	Diagonalkreuze

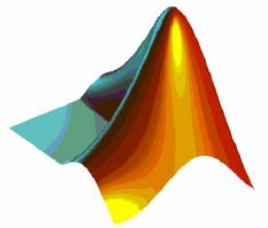
- Beispiel: plot(1:0.1:2*pi,sin(1:0.1:2*pi),'r-.')

2D-Graphik: Spezielle Plotbefehle

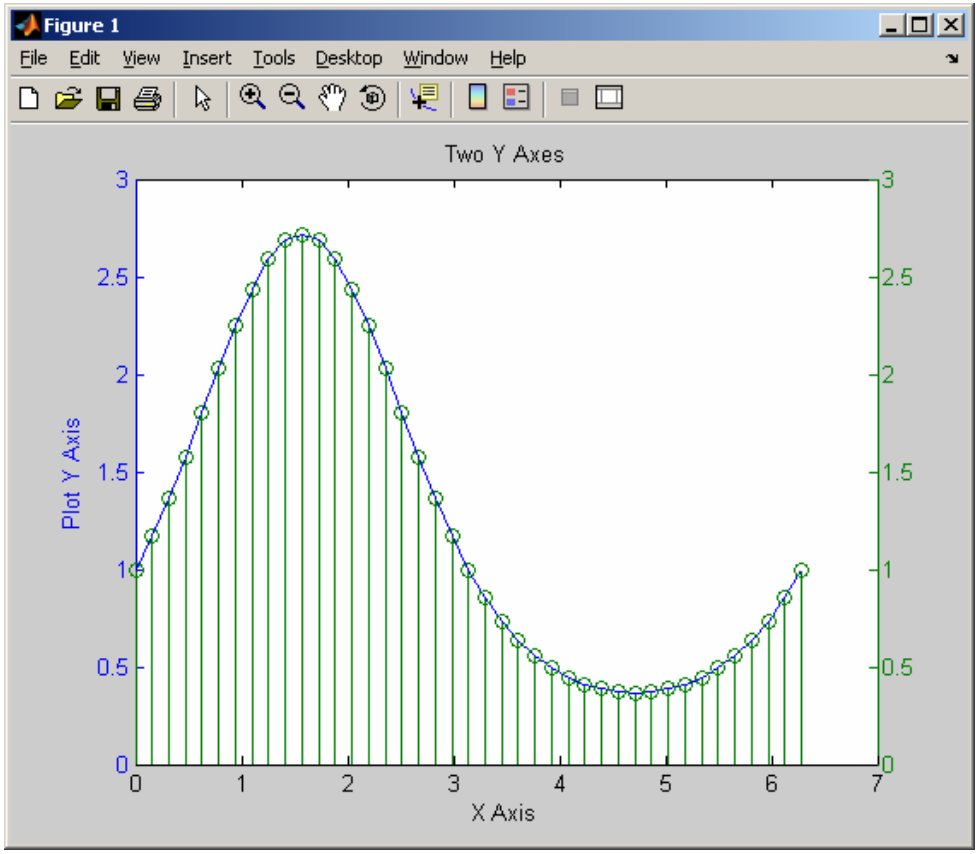


- Treppenförmig: stairs ([x,] y... [, plotstil])
- Diskret: stem ([x,] y... [, plotstil])
- Logarithmisch:
  - loglog (x, y... [, plotstil])
  - x-Achse: semilogx (x, y... [, plotstil])
  - y-Achse: semilogy (x, y... [, plotstil])
- Funktionen:
  - explizite Funktion: fplot (f, bereich)
  - implizite Funktion: ezplot (f(x, y), bereich)
  - Parameterkurve: ezplot (f_1,f_2, bereich)

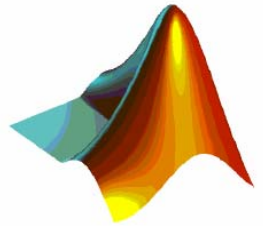
Beispiel:



```
t = 0:pi/20:2*pi;
y = exp(sin(t));
plotyy(t,y,t,y,'plot','stem')
xlabel('X Axis')
ylabel('Plot Y Axis')
title('Two Y Axes')
```



3D-Graphik: Plotbefehle



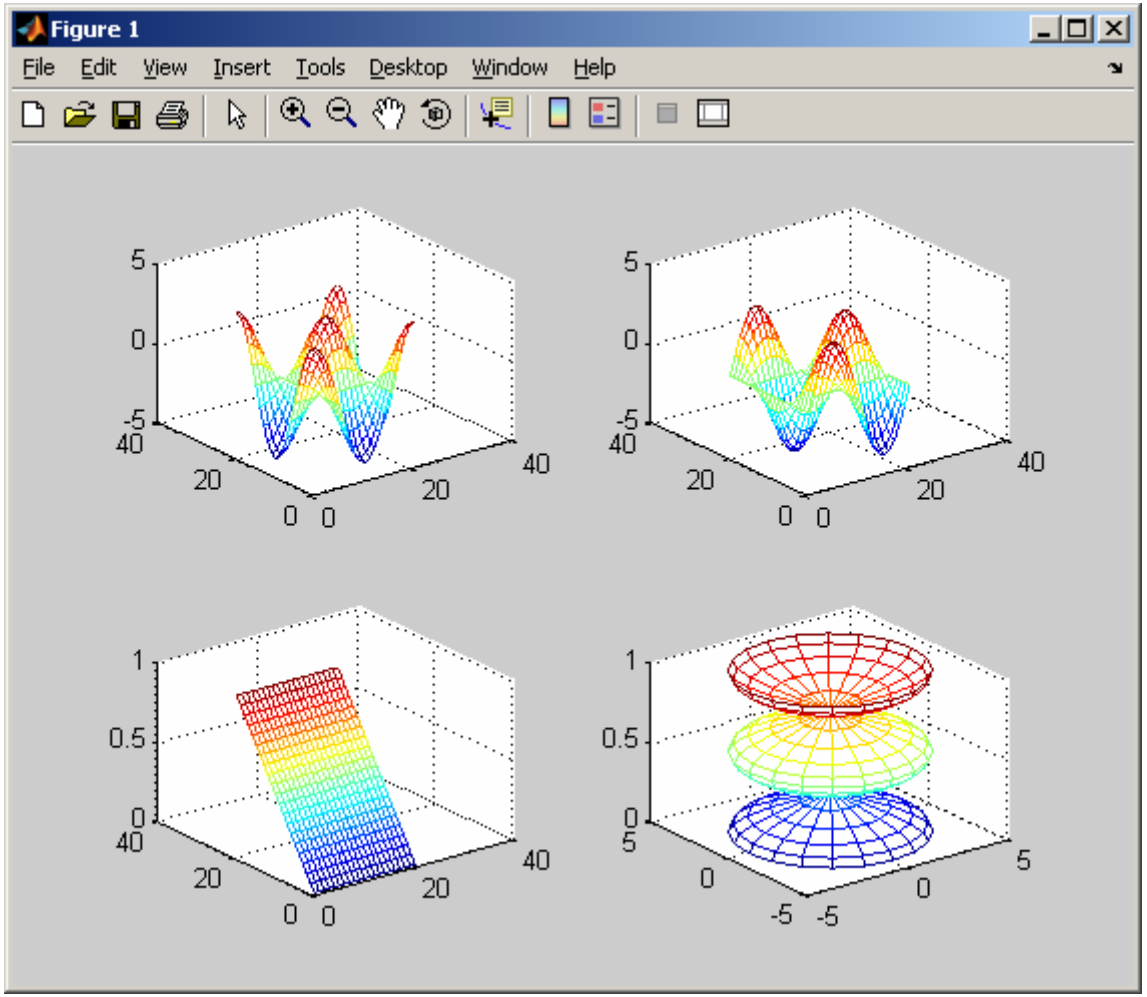
- Punkt/Linien: plot3 (x, y, z... [, plotstil])
- Fläche: surf (x, y, z... [, farbe])
- Gitter: mesh (x, y, z... [, farbe])
- Wasserfall: waterfall (x, y, z... [...])
- Höhenlinien: contour (x, y, z... [...])

  x, y, z: Matrizen gleicher Dimension

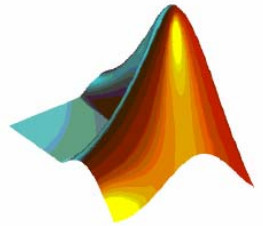
- [X,Y] = meshgrid(x_vek, y_vek) erzeugt aus Vektoren x_vek und y_vek Koordinatenmatrizen richtiger Größe

Beispiel:

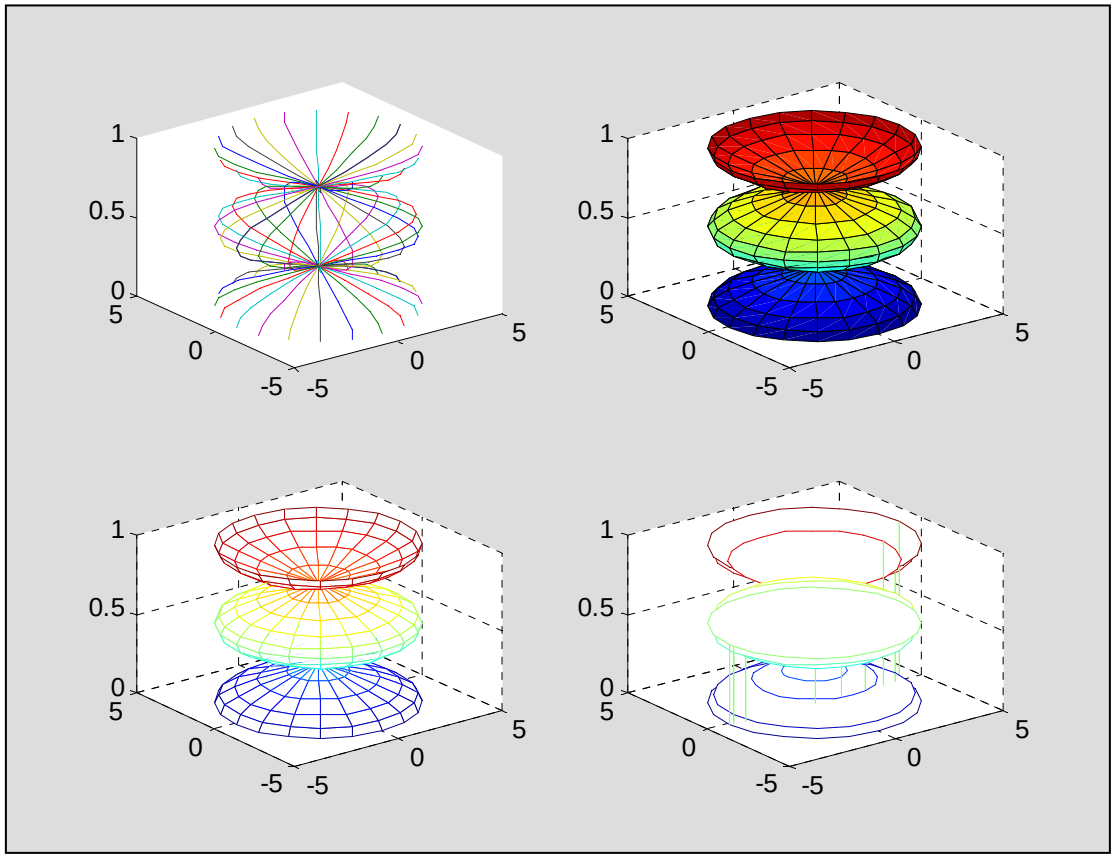
```
t = 0:pi/10:2*pi;
[X,Y,Z] =
    cylinder(4*cos(t));
subplot(2,2,1);
    mesh(X)
subplot(2,2,2);
    mesh(Y)
subplot(2,2,3);
    mesh(Z)
subplot(2,2,4);
    mesh(X,Y,Z)
```



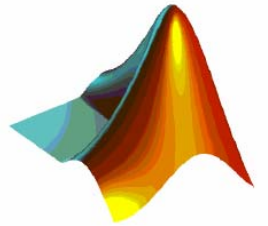
Beispiel:



```
t = 0:pi/10:2*pi;
[X,Y,Z] =
    cylinder(4*cos(t));
subplot(2,2,1);
    plot3(X,Y,Z)
subplot(2,2,2);
    surf(X,Y,Z)
subplot(2,2,3);
    mesh(X,Y,Z)
subplot(2,2,4);
    waterfall(X,Y,Z)
```



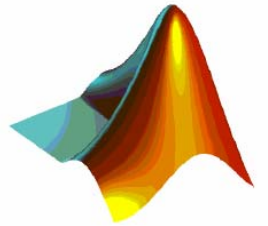
3D-Graphik: Beschriftung, Perspektive und Farben



- Skalierung: axis ([x...,y...,z_min,z_max])
- Beschriftung der z-Achse: zlabel (string)
- Box einblenden: box [on | off]
- Perspektive ändern: view (azimuth, elevation)
- Farben:
  - Farbtabelle: colormap (name)
  - Skalierung: caxis (farbe_min, farbe_max)

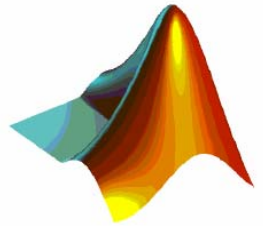
- Online-Hilfe: graph2d, graph3d, specgraph

Importieren, exportieren und drucken von Graphiken



- Figure
  - drucken: print -f*nummer*
  - speichern: print -f*nummer* -d*device* datei
    saveas(h,'datei','fig')
- Ausgabeoptionen (Format, Treiber): -d*device*
  - Formate: PS, EPS, TIFF, HPGL, JPEG, . . .
  - Windows: EMF, BMP, Druckertreiber, Clipboard
- Graphik:
  - einlesen: A = imread(datei,format)
  - erzeugen: imwrite(A,datei,format)
  - plotten: image(A)

GUI - Graphical User Interface



- Graphische Benutzerschnittstelle
- Volle Ausnutzung der Objekt-Eigenschaften von
- MATLAB-Graphiken
- Programmieren mittels GUIDE oder von Hand