Matlab

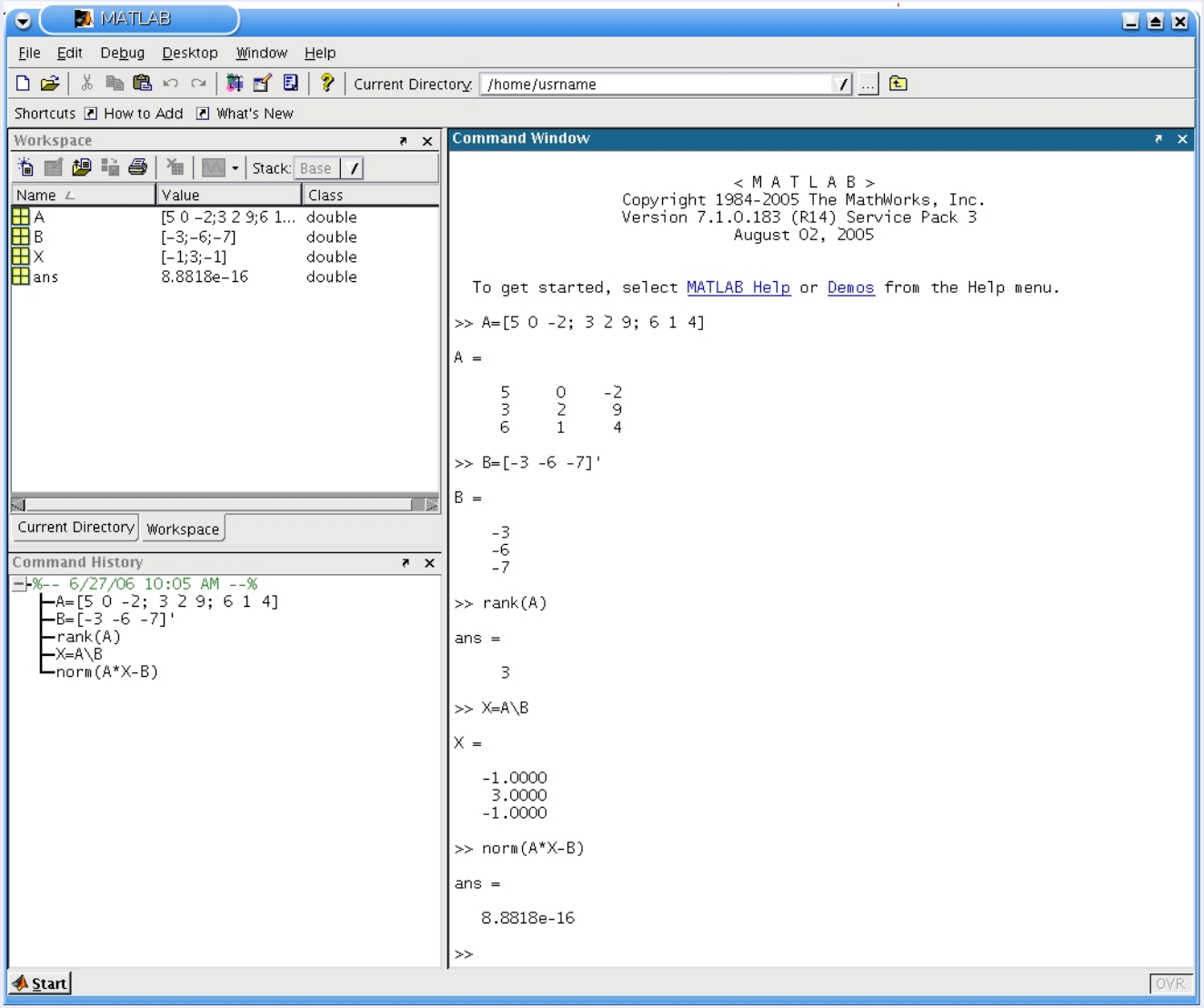
Das Handout ist Bestandteil der Vortragsfolien zur Höheren Mathematik; siehe die Hinweise auf der Internetseite vhm.mathematik.uni-stuttgart.de für Erläuterungen zur Nutzung und zum Copyright.

Benutzeroberfläche

dreiteiliger Desktop

- Command Window
  Matlab-Operationen und -Befehle.
  Ausführung von Programmen und Skripten.
- Command History
  zuletzt eingegebene Befehle (Wiederholung durch Mausklick).
- Current Directory
  Browser für aktuelles Verzeichnis.

Aufruf von der Konsole: `matlab -nodesktop &`


MATLAB
File Edit Debug Desktop Window Help
Current Directory: /home/usrname
Shortcuts How to Add What's New
Workspace
Stack: Base
Name Value Class
A [5 0 -2;3 2 9;6 1... double
B [-3;-6;-7] double
X [-1;3;-1] double
ans 8.8818e-16 double
Current Directory Workspace
Command History
%-- 6/27/06 10:05 AM --%
A=[5 0 -2; 3 2 9; 6 1 4]
B=[-3 -6 -7]'
rank(A)
X=A\B
norm(A*X-B)
Command Window
< M A T L A B >
Copyright 1984-2005 The MathWorks, Inc.
Version 7.1.0.183 (R14) Service Pack 3
August 02, 2005
To get started, select MATLAB Help or Demos from the Help menu.
>> A=[5 0 -2; 3 2 9; 6 1 4]
A =
5 0 -2
3 2 9
6 1 4
>> B=[-3 -6 -7]'
B =
-3
-6
-7
>> rank(A)
ans =
3
>> X=A\B
X =
-1.0000
3.0000
-1.0000
>> norm(A*X-B)
ans =
8.8818e-16
>>
Start
OVR

Beispiel

```
user@host:~> matlab -nodesktop

                                  < M A T L A B >
                      Copyright 1984-2005 The MathWorks, Inc.
                      Version 7.1.0.183 (R14) Service Pack 3
                                  August 02, 2005


To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

  >>
```

```
>> A=[5 0 -2; 3 2 9; 6 1 4]
A =

     5     0    -2
     3     2     9
     6     1     4

>> A\[-3; -6; -7]
ans =

   -1.0000
    3.0000
   -1.0000

>> exit
user@host:~>
```

Hilfe

- `help`, `help` *Thema*, `help` *Funktionsname*
  Übersicht bzw. Hilfe zu einem Thema oder einer Funktion.
- `lookfor` *Text*
  sucht Funktionen mit *Text* in ihrer Kurzbeschreibung.
- `helpbrowser`
  Dokumentation in HTML-Format.
- `doc` *Thema*, `doc` *Funktionsname*
  Beschreibungen zu *Themen* oder *Funktionen*.

Beispiel

```
>> lookfor rank
CHOLUPDATE Rank 1 update to Cholesky factorization.
QRUPDATE Rank 1 update to QR factorization.
RANK   Matrix rank.
SPRANK Structural rank.
GFRANK Compute the rank of a matrix over a Galois field.
FITSCALINGRANK Rank based fitness scaling.
GANGSTR Zero out 'small' entries subject to structural rank.
FRANKE Franke's bivariate test function.
...
```

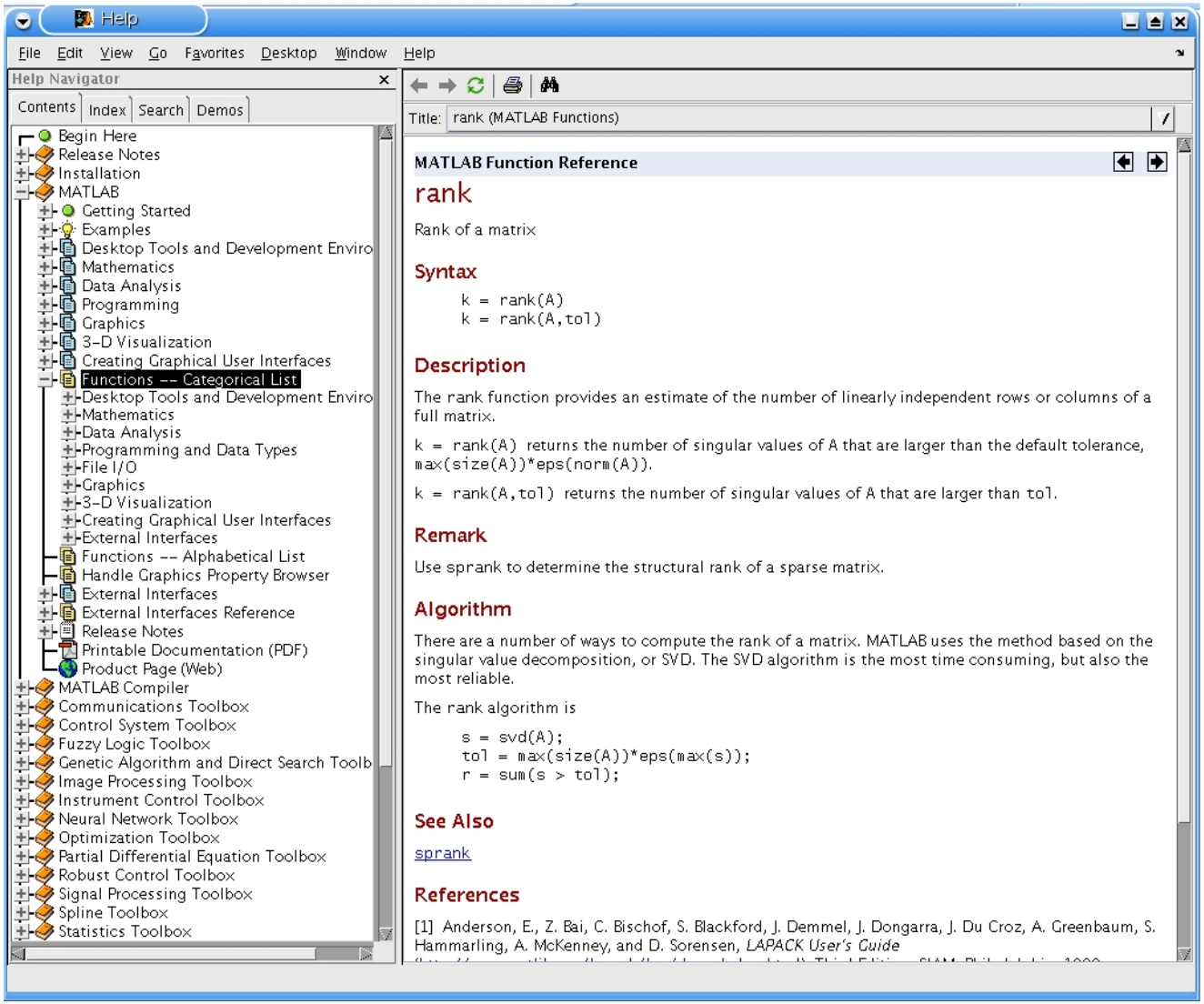
```
>> help rank
 RANK   Matrix rank.
    RANK(A) provides an estimate of the number of linearly
    independent rows or columns of a matrix A.
    RANK(A,tol) is the number of singular values of A
    that are larger than tol.
    RANK(A) uses the default tol = max(size(A)) * eps(norm(A)).

    Class support for input A:
       float: double, single

    Overloaded functions or methods (ones with the same name in
         other directories)
       help gf/rank.m
       help sym/rank.m

    Reference page in Help browser
       doc rank
```

doc rank


Help
File Edit View Go Favorites Desktop Window Help
Help Navigator
Contents Index Search Demos
Begin Here
Release Notes
Installation
MATLAB
Getting Started
Examples
Desktop Tools and Development Enviro
Mathematics
Data Analysis
Programming
Graphics
3-D Visualization
Creating Graphical User Interfaces
Functions -- Categorical List
Desktop Tools and Development Enviro
Mathematics
Data Analysis
Programming and Data Types
File I/O
Graphics
3-D Visualization
Creating Graphical User Interfaces
External Interfaces
Functions -- Alphabetical List
Handle Graphics Property Browser
External Interfaces
External Interfaces Reference
Release Notes
Printable Documentation (PDF)
Product Page (Web)
MATLAB Compiler
Communications Toolbox
Control System Toolbox
Fuzzy Logic Toolbox
Genetic Algorithm and Direct Search Toolb
Image Processing Toolbox
Instrument Control Toolbox
Neural Network Toolbox
Optimization Toolbox
Partial Differential Equation Toolbox
Robust Control Toolbox
Signal Processing Toolbox
Spline Toolbox
Statistics Toolbox
Title: rank (MATLAB Functions)
MATLAB Function Reference
rank
Rank of a matrix
Syntax
k = rank(A)
k = rank(A,tol)
Description
The rank function provides an estimate of the number of linearly independent rows or columns of a full matrix.
k = rank(A) returns the number of singular values of A that are larger than the default tolerance, max(size(A))*eps(norm(A)).
k = rank(A,tol) returns the number of singular values of A that are larger than tol.
Remark
Use sprank to determine the structural rank of a sparse matrix.
Algorithm
There are a number of ways to compute the rank of a matrix. MATLAB uses the method based on the singular value decomposition, or SVD. The SVD algorithm is the most time consuming, but also the most reliable.
The rank algorithm is
s = svd(A);
tol = max(size(A))*eps(max(s));
r = sum(s > tol);
See Also
sprank
References
[1] Anderson, E., Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, LAPACK User's Guide

Zahlen

Gleitpunktformat, IEEE-Standard double

$$\text{Betrag} \in [\underbrace{2^{-1022}}_{realmin}, \underbrace{(1-2^{-53})2^{1024}}_{realmax}] = [2.2251e-308, 1.7977e+308]$$

Spezielle Zahlen

- eps:
  Abstand zwischen eins und nächstgrößerer darstellbarer Zahl (entspricht doppelter relativer Genauigkeit)
- inf:
  Überlauf (∞)
- nan:
  Resultat mathematisch nicht definierter Operationen
- i,j:
  Komplexe Einheit

Beispiel

Eingabe von (komplexen) Zahlen:

```
>> 123
ans =
   123
```

```
>> 1.23
ans =
   1.2300
```

```
>> .000123
ans =
   1.2300e-04
```

```
>> 1.23e-4
ans =
   1.2300e-04
```

```
>> 12i
ans =
   0 +12.0000i
```

```
>> 12+3i
ans =
   12.0000 + 3.0000i
```

```
>> 1.23+4.56e-3i
ans =
   1.2300 + 0.0046i
```

```
>> 4j-123
ans =
  -1.2300e+02 + 4.0000e+00i
```

Überlauf und mathematisch nicht definierte Operationen:

```
>> 1e400
ans =
   Inf

>> 1-inf
ans =
  -Inf

>> -1+inf
ans =
   Inf

>> inf+inf
ans =
   Inf
```

```
>> inf-inf
ans =
   NaN

>> 1/0
Warning: Divide by zero.
ans =
   Inf
```

```
>> 0/0
Warning: Divide by zero.
ans =
   NaN
```

Darstellungsformate

`format` *Stil*: Wählt Ausgabeformat

- `short`: skalierte Festpunktdarstellung mit 5 Stellen
- `long`: skalierte Festpunktdarstellung mit 15 Stellen
- `short e`: Gleitpunktdarstellung mit 5 Stellen
- `long e`: Gleitpunktdarstellung mit 15 Stellen
- `short g`: optimale Darstellung mit 5 Stellen
- `long g`: optimale Darstellung mit 15 Stellen
- `short eng`: mindestens 5 Stellen und einen durch drei teilbaren Exponenten
- `long eng`: 16 signifikante Stellen und einen durch drei teilbaren Exponenten
- `rat`: Näherung in Bruchdarstellung

Formatierte Ausgabe

Ausgabe in eine Datei

`file_id = fopen('`*Dateiname*`','w')`

`fprintf(file_id,'`*Format*`',`*Variablen*`)`

`fclose(file_id)`

einige Formate

- %e: Gleitpunktformat
- %f: Festkommadarstellung
- %s: Zeichenkette

Sonderzeichen

`\n`: neue Zeile, `\t`: horizontaler Tab, `\\`: Backslash

Beispiel

Programmsegment

```
f = fopen('Winkel.txt','w');
fprintf(f,'Winkelumrechnung\n');
fprintf(f,'%i Grad \t%e%s\n',30,30*pi/180,'(Bogenmaß)');
fclose(f);
```

→ Datei `Winkel.txt` mit zwei Textzeilen:

```
Winkelumrechung
30 Grad       5.235998e-001 (Bogenmaß)
```

Elementare Operatoren und Funktionen

- **arithmetische Operatoren**

  `+`, `-`, `*`, `/`, `\` (`a\b` entspricht `b/a`), `^` (Potenz)

- **trigonometrische Funktionen** (Winkelangaben in Bogenmaß)

  `cos, sin, tan, cot, acos, asin,`
  `atan, sinh, cosh, ...`

- **Exponentialfunktionen**

  `exp, pow2, log, log10, log2, sqrt, realsqrt, ...`

- **Rechnen mit komplexen Zahlen**

  `abs, angle, conj, real, imag`

- **Sonstige**

  `round, floor, ceil, mod, rem, sign`

- **Konstanten**

  `pi, exp(1), i, j`

Beispiel

Beispiele zur Reihenfolge bei der Auswertung von Operatoren:

```
>> 3^3^3
ans =
       19683
```

```
>> (3^3)^3
ans =
       19683
```

```
>> 3^(3^3)
ans =
   7.6256e+12
```

```
>> 12/3/4
ans =
     1
```

```
>> 3\12/4
ans =
     1
```

```
>> 16^1/2-16^(1/2)
ans =
     4
```

Fehler aufgrund von endlicher Präzision bei der Gleitpunktarithmetik:

```
>> exp(1)*exp(1)/exp(2)-1
ans =
   2.2204e-16
```

Komplexe Funktionen:

```
>> i^i
ans =
    0.2079
```

```
>> sqrt(-1)
ans =
          0 + 1.0000i
```

```
>> realsqrt(-1)
??? Error using ==> realsqrt
Realsqrt produced complex result.
```

Rechnungen mit dem speziellen Wert bzw. Resultat `inf` (∞)

```
>> sqrt(inf)
ans =
   Inf
>> sqrt(-inf)
ans =
   0 +    Infi
>> log(0)
Warning: Log of zero.
ans =
   -Inf
```

Variablen

Name aus Buchstaben, Ziffern und dem Zeichen _ (case sensitive)
Wertzuweisung:

Variable = Ausdruck

Befehle

- `who` bzw. `whos`:
  Liste der definierten *Variablen*
- `clear` *Variablennamen*, `clear`:
  löscht *Variablen* bzw. *alle Variablen*
- `save` *Dateiname* [*Variablen*]:
  speichert alle [bzw. die angegebenen *Variablen*]
  in Datei *Dateiname*.mat
- `load` *Dateiname* [*Variablen*]:
  lädt alle [bzw. die angegebenen *Variablen*]
  von Datei *Dateiname*.mat

Beispiel

```
>> clear; Radius=4;
>> Flaeche=pi*radius^2
??? Undefined function or variable 'radius'.

>> Flaeche=pi*Radius^2;
>> Flaeche
Flaeche =
  50.2655
>> who

Your variables are:

Flaeche  Radius
```

Eingabe von Matrizen

- *Variable* $= [a_{11}, a_{12}, \ldots, a_{1n}; a_{21}, a_{22}, \ldots, a_{2n}, \ldots]$
  Elemente durch Komma oder Leerzeichen,
  Zeilen durch Semicolon oder Return getrennt.
- Mehrzeilige Eingabe einer Zeile mit Fortsetzungspunkten $''\ldots''$
- Schachtelung möglich
- Skalare und Vektoren als spezielle Matrizen

Beispiel

```
>> v_col = [1;2;3], v_row = [1 2 3]
v_col =
     1
     2
     3

v_row =
     1     2     3

>> A = [v_col,...
[v_row; 0,0,0; 4 5 6] v_col]
A =
     1     1     2     3     1
     2     0     0     0     2
     3     4     5     6     3
```

Beispiel

Definition einer aus fünf Blöcken bestehenden Matrix:

```
>> A11=[1 1 1; 1 1 1];
>> A12=[2 2; 2 2];
>> A21=[3 3; 3 3; 3 3];
>> A22=[4 4 4; 4 4 4; 4 4 4];
>> [[A11 A12;A21 A22],[5;5;5;5;5]]
ans =
     1     1     1     2     2     5
     1     1     1     2     2     5
     3     3     4     4     4     5
     3     3     4     4     4     5
     3     3     4     4     4     5
```

Spezielle Matrizen und Vektoren

- `ones(n,m)`: (`n` $\times$ `m`)-Matrix bei der alle Einträge 1 sind
- `zeros(n,m)`: (`n` $\times$ `m`)-Matrix bei der alle Einträge 0 sind
- `eye(n,m)`: (`n` $\times$ `m`)-Matrix bei der alle Einträge auf der Hauptdiagonalen 1 und sonst 0 sind
- `rand(n,m)`: (`n` $\times$ `m`)-Matrix deren Einträge Pseudozufallszahlen zwischen 0 und 1 sind
- `randn(n,m)`: (`n` $\times$ `m`)-Matrix mit normalverteilten Pseudozufallszahlen mit Mittelwert 0 und Standardabweichung 1
- `[a:d:b]`: Ergibt den Vektor `[a a+d a+2d ... a+m*s]` mit `m=fix((b-a)/d)`
  (`[a:b]`: Kurzform von `[a:1:b]`)
- `linspace(a,b,n)`: Erzeugt einen äquidistant unterteilten $(1 \times n)$-Vektor mit erstem Element a und letztem Element b.

Beispiel

```
>> eye(2,3)
ans =
     1     0     0
     0     1     0

>> zeros(2)
ans =
     0     0
     0     0

>> rand(3)
ans =
    0.3046    0.3028    0.3784
    0.1897    0.5417    0.8600
    0.1934    0.1509    0.8537
```

```
>> [2:5]
ans =
   2   3   4   5

>> v = [2:-3:-5]
v =
   2   -1   -4

>> linspace(0,1,5)
   0   0.25   0.5   0.75   1

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

Indizierung von Matrixelementen

Zugriff auf Matrixblöcke mit Indexvektoren

- `A(Z,S)`: Teilmatrix mit den Elementen der durch die Zeilenvektoren bzw. Spaltenvektoren `Z` bzw. `S` indizierten Elementen
- `A(K)`: Vektor der durch den Indexvektor `K` bei spaltenweiser Nummerierung indizierter Elemente. Stehender Vektor, falls `K` ein stehender Vektor ist, andernfalls liegend.

alle Zeilen/Spalten mit $Z = (1 : end)/S = (1 : end)$ oder abgekürzt mit ″ :″ als Index.
Zuweisungsmöglichkeiten:

- Teilmatrix = *Skalar*: Alle indizierten Elemente werden durch den *Skalar* ersetzt
- Teilmatrix = *Matrix*: Ersetzen der indizierten Teilmatrix durch die gegebene *Matrix* gleicher Größe
- Teilmatrix = `[ ]`: Löschen der indizierten Spalten und/oder Zeilen

Beispiel

```
>> A=[11 12 13 14 15 16 17 18 19
      21 22 23 24 25 26 27 28 29];
>> A(2,7)
ans =
    27
>> A(1,[3 7 1])
ans =
    13    17    11
>> A([5 9 end])
ans =
    13    15    29
>> A(:,2:2:end)
ans =
    12    14    16    18
    22    24    26    28
```

```
>> A(:,1:3:end)=[]
A =
    12    13    15    16    18    19
    22    23    25    26    28    29

>> A(:,3:4)=0
A =
    12    13     0     0    18    19
    22    23     0     0    28    29

>> A(:,[1:2,end-1:end])=ones(2,4)
A =
     1     1     0     0     1     1
     1     1     0     0     1     1
```

Matrix-Operationen

- $A \pm B$: Addition bzw. Subtraktion
- $A * B$: Matrixmultiplikation
- $A \wedge n$: n-te Potenz
- $X = A \setminus B \quad (X = B \setminus A)$: Ausgleichslösung des linearen Gleichungssystems $AX = B \quad (XA = B)$
- $A. * B$, $A. \wedge B$, ...: punktweise Operationen

Beispiel

```
% Lösen eines linearen Gleichungssystems

>> A = [1 2; 3 4]; b = [5; -6];
>> x = A\b
x =
   -16.0000
    10.5000
>> format long
>> A*x
ans =
   4.999999999999998
  -6.000000000000000
```

Beispiel

```
% Matrix-Operationen

>> A = [0 1 2; 3 4 5]; x = [-1; -2];
>> A + [x x x]
ans =
   -1    0    1
    1    2    3
>> B = x*x'
B =
    1    2
    2    4
>> [B^2 B.^2]
ans =
    5   10    1    4
   10   20    4   16
```

Matrixumwandlung

- `A.'` bzw. `A'`:
  transponiert bzw. komplex konjugiert transponieren von `A`
- `fliplr, flipud`:
  Spalten- bzw. Zeilenspiegelung
- `tril, triu`:
  untere bzw. obere Dreiecksmatrix selektieren
- `diag`:
  (Neben-)Diagonale selektiert bzw. setzen
- `reshape`:
  Änderung der Matrixdimension unter Beibehaltung der Elemente
- `repmat`:
  Matrix mehrfach in einer Blockmatrix anordnen.

Beispiel

```
>> A=[1 2 2-i; i 3 -1+i]
A =
   1.0000                  2.0000                  2.0000 - 1.0000i
        0 + 1.0000i        3.0000                 -1.0000 + 1.0000i
>> A'
ans =
   1.0000                       0 - 1.0000i
   2.0000                  3.0000
   2.0000 + 1.0000i       -1.0000 - 1.0000i
>> A.'
ans =
   1.0000                       0 + 1.0000i
   2.0000                  3.0000
   2.0000 - 1.0000i       -1.0000 + 1.0000i
```

```
>> A = [11 12 13; 21 22 23];
```

Umordnen der Einträge

```
>> B = reshape(A,3,2)
B =
    11    22
    21    13
    12    23
```

Die Matrix `B` wird spaltenweise gefüllt. Dabei werden die Elemente spaltenweise der Matrix `A` entnommen.
Selektion bzw. Setzen der Hauptdiagonalen

```
>> d=diag(B)
d =
     11
     13
```

```
>> diag(d)
ans =
     11     0
      0    13
```

Darstellung von Funktionen und Kurven

- `plot`: Zeichnen von Polygonzügen
- `semilogx`, `semilogy`: logarithmische Skalierung in x- bzw. y-Richtung
- `loglog`: logarithmische Skalierung in x- und y-Richtung

Aufrufvarianten:

`plot(X,Y)`, `plot(X,Y,S)`, `plot(X1,Y1,S1,X2,Y2,S2,...)`

Daten: $(x_1, y_1), (x_2, y_2), \ldots$
optionaler Formatstring S mit maximal 4 Zeichen für
Farbe: `b` (Blau), `g` (Grün), `r` (Rot), `y` (Gelb), `k` (Schwarz), ...
Kennzeichner: `.`, `o`, `x`, `+`, `*`, `<`, `>`, ...
Linienstil: `-`, `--`, `:`, `-.`

analog: Zeichnen räumlicher Polygonzüge mit `plot3`

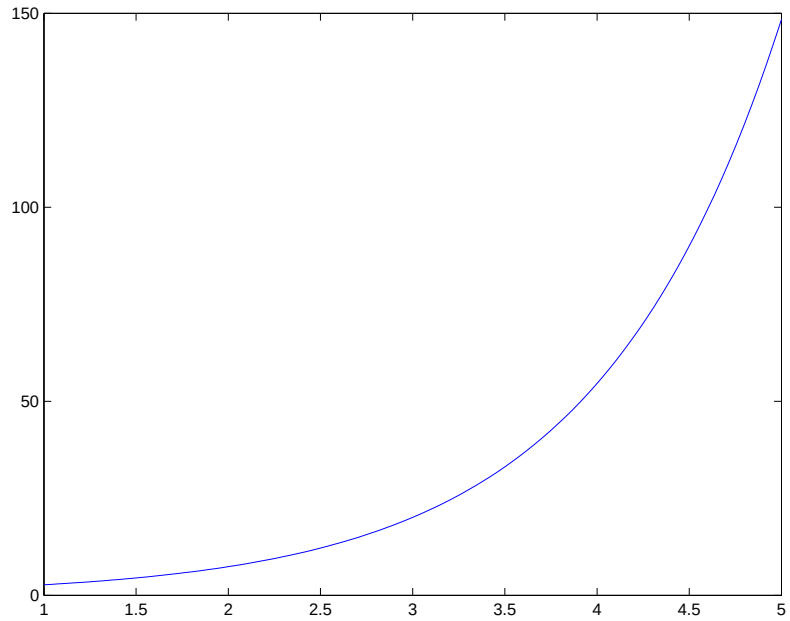
Beispiel

Exponentialfunktion

$$x \mapsto y = \exp(x)$$

auf dem Intervall $[1, 5]$

```
>> x=linspace(1,5);
>> y=exp(x);
>> plot(x,y);
```



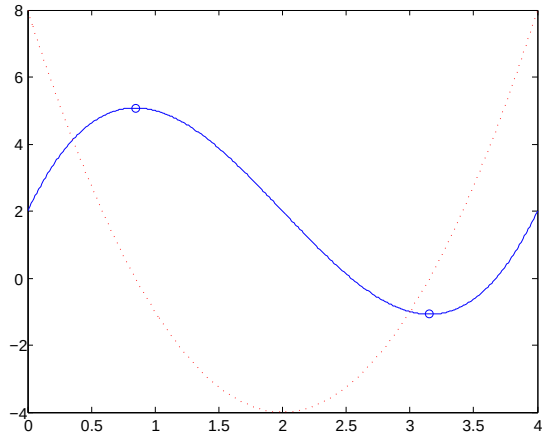
Beispiel

Darstellung eines Polynoms

$$x \mapsto y = x^3 - 6x^2 + 8x + 2$$

mit Ableitung und Extremstellen

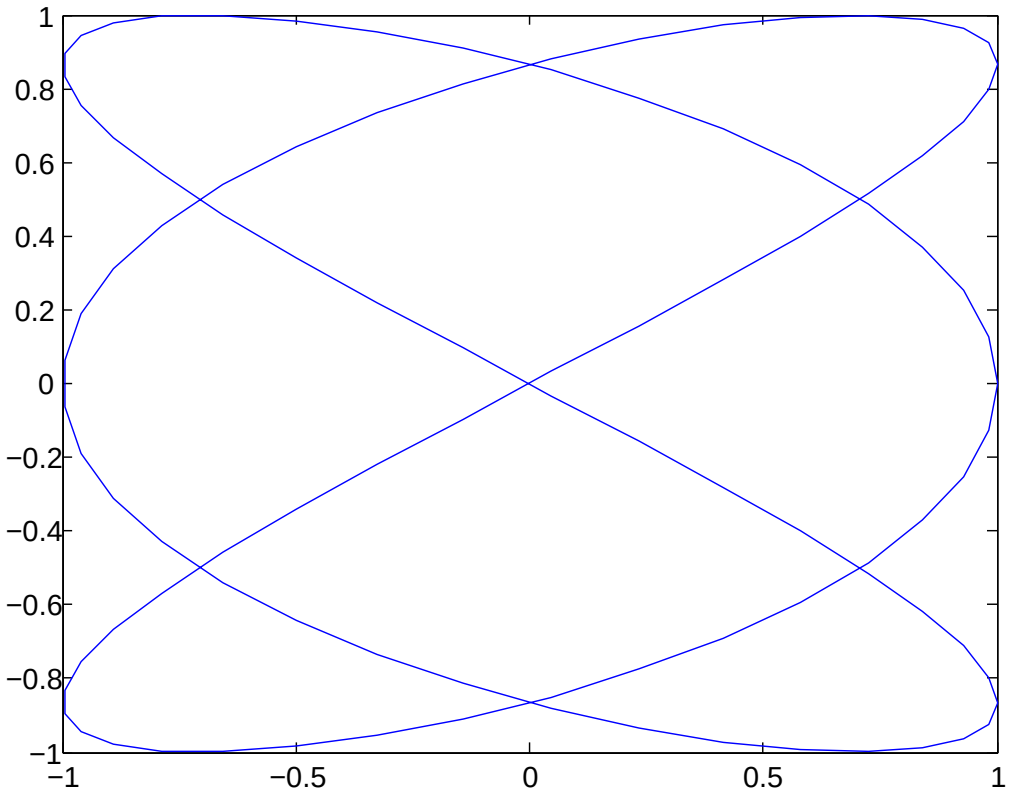
```
>> x = [0:0.01:4];
>> c = [1 -6 8 2];
>> y = polyval(c,x);
>> dc = polyder(c);
>> dy = polyval(dc,x);
>> xe = roots(dc);
>> ye = polyval(c,xe);
>> plot(x,y,'b',x,dy,':r',xe,ye,'o')
```



Beispiel

Lissajous-Kurve

```
>> t = linspace(0,2*pi);
>> x = cos(3*t); y = sin(2*t);
>> plot(x,y)
```



Diagramme

Balkendiagramme

`bar(x,Y)`: Werte `Y(k,:)` als Balkengruppe über `x(k)`
Default `x = [1:size(Y,1)]`

Histogramme

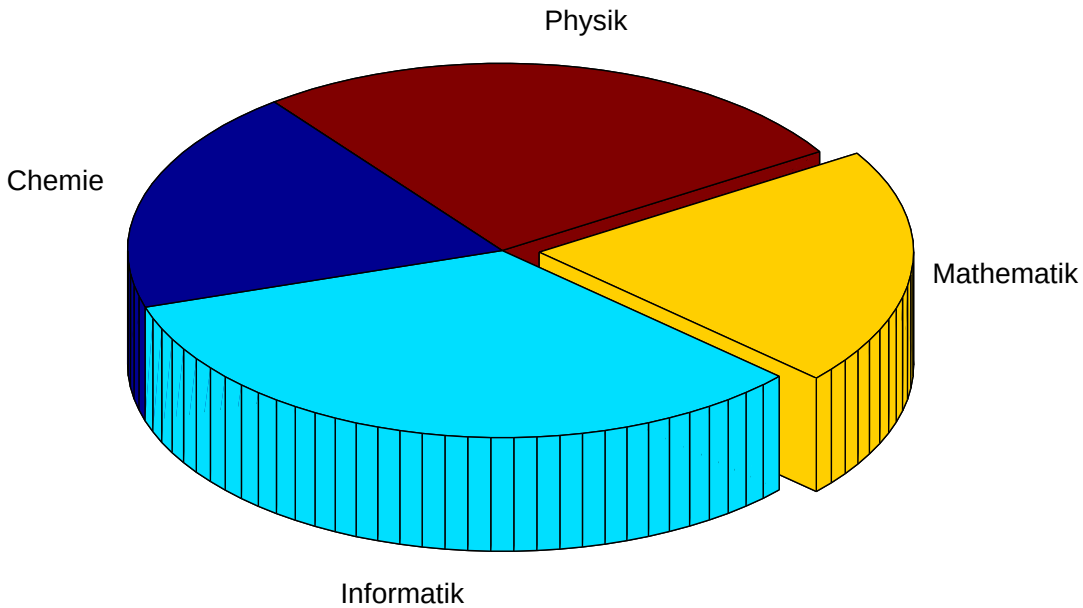
`n = histc(x,v)`: Anzahl `n(k)` der `x`-Werte in `[v(k),v(k+1))`
bzw. gleich `v(end)`
Balkendiagramm bei fehlendem Ausgabewert

Tortendiagramme

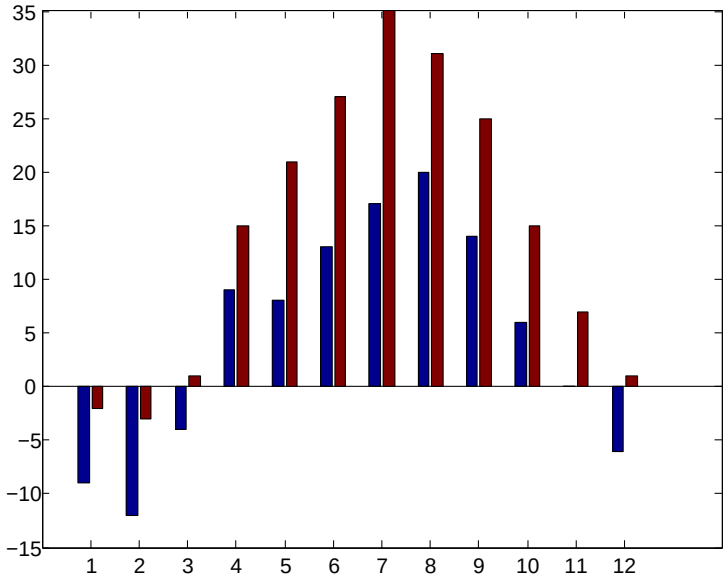
`pie(n,{'Anteil_1', ...})`: Anteile `n(k)/sum(n)` als Sektoren
`pie(n,[0,1,...])`: Flags für hervorgehobene Teile

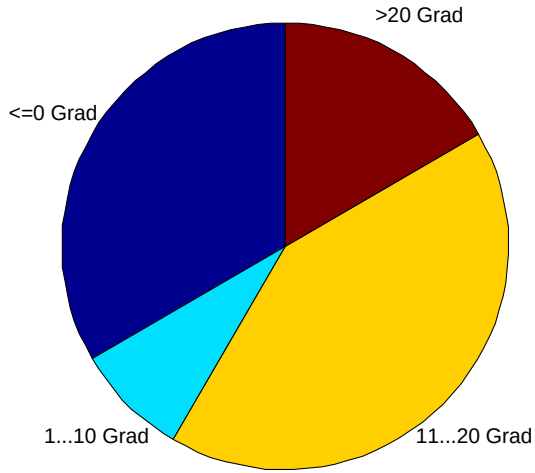
Beispiel

```
>> studierende=[485 831 513 661];
>> faecher={'Chemie','Informatik',...
            'Mathematik','Physik'};
>> pie3(studierende,[0 0 1 0],faecher)
```



```
>> % Tages- und Nachttemperaturen im monatlichen Mittel
>> bar(degrees);
>> average = sum(degrees')/2;
>> months = histc(average,[-inf 1 11 21 inf]);
>> pie(months(1:end-1),{'<=0 Grad', '1...10 Grad', ...
      '11...20 Grad', '>20 Grad'})
```





Darstellung bivariater Funktionen und Flächen

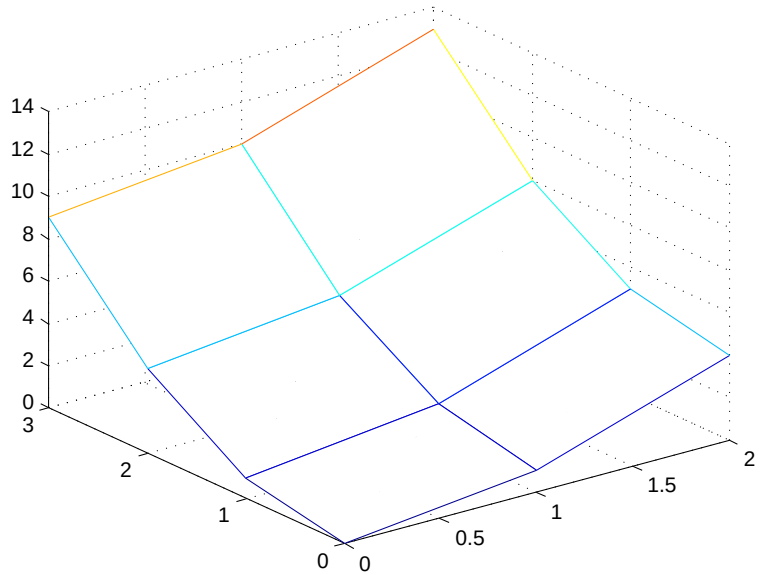
- `mesh(X,Y,Z,C)`: Flächennetz
- `surf(X,Y,Z,C)`, `surfl(X,Y,Z,V)`: Fläche bzw. beleuchtete Fläche

Argumente:
Matizen X, Y, Z, C von Koordinaten und optionalen Farbwerten
$\rightarrow$ schachbrettförmiges Viereckgitter mit Index in Farbtabelle
Richtung V der Lichtquelle
Steuerung der Darstellung:
`colormap jet, gray, autumn, cool, hsv, ...`
`shading flat, interp, faceted`
`lighting flat, gouraud, phong`
Gittererzeugung: `[X,Y] = meshgrid(x,y)`

Beispiel

Funktion $z = x^2 + y^2$ auf $[0, 2] \times [0, 3]$

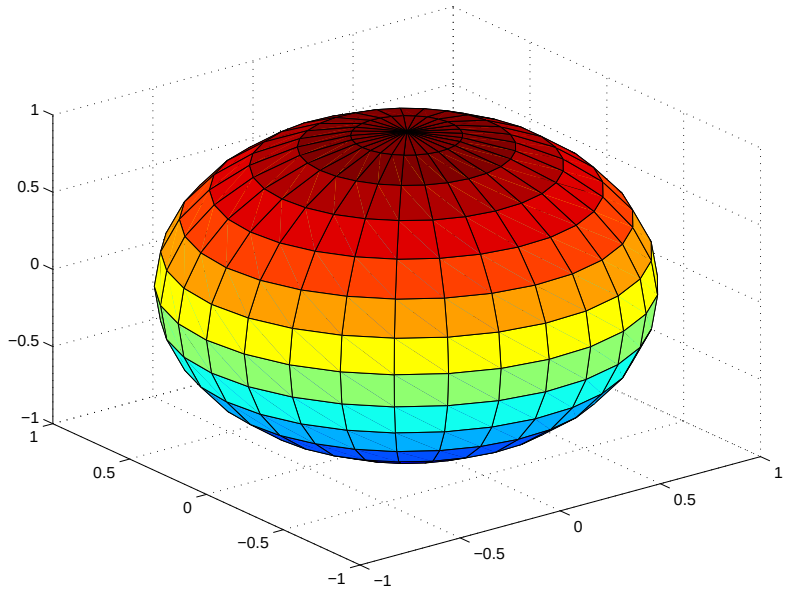
```
>> [X,Y] = meshgrid(0:2,0:3)
   X =
        0     1     2
        0     1     2
        0     1     2
        0     1     2
   Y =
        0     0     0
        1     1     1
        2     2     2
        3     3     3
>> Z = X.^2+Y.^2;
mesh(X,Y,Z)
```



Beispiel

Einheitsphäre, parametrisiert mit Hilfe von Kugelkoordinaten

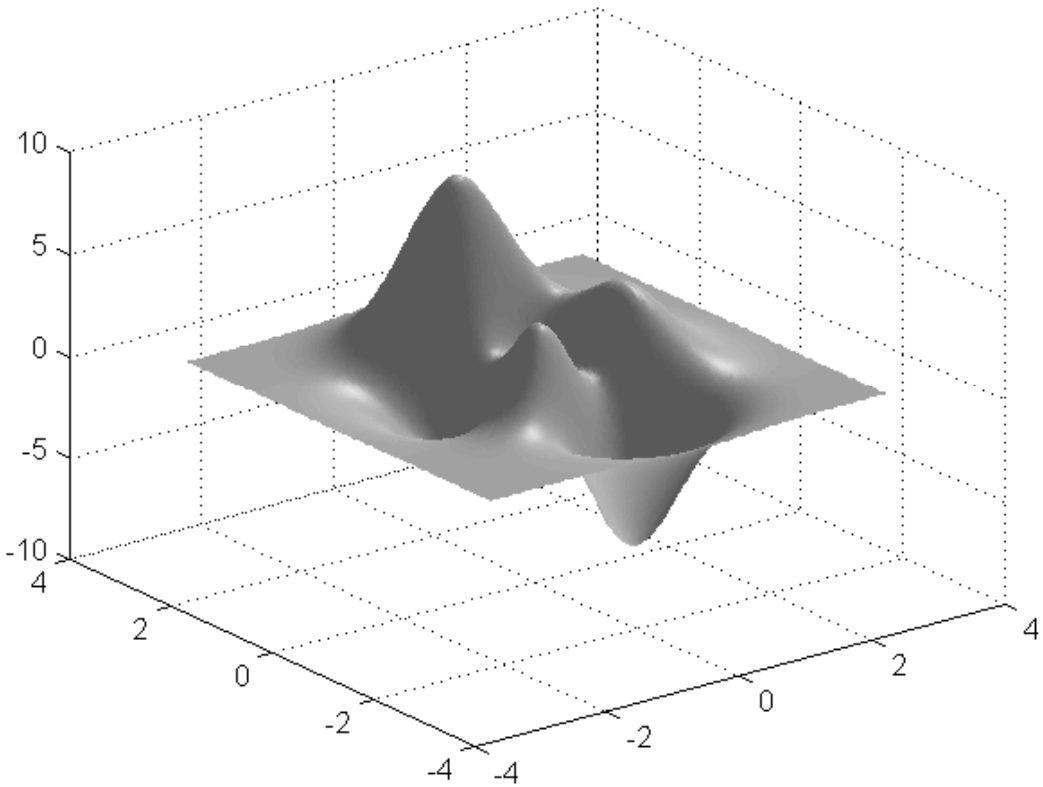
```
>> [p,t]=meshgrid(...
      linspace(-pi,pi,30),...
      linspace(0,pi,15));
>> X=cos(p).*sin(t);
>> Y=sin(p).*sin(t);
>> Z=cos(t);
>> surf(X,Y,Z);
```



Beispiel

beleuchtetes Geländeprofil

```
>> [X,Y,Z]=peaks(100);
>> surfl(X,Y,Z,[1 0 1]);
>> colormap gray; shading interp; lighting phong;
```



Modifikation des Koordinatensystems und dessen Darstellung

Kontrolle des Achsensystems:

`axis`	*Grenzen, Skalierung und Darstellung der Achsen*
`grid, box`	*Darstellung von Gitterlinien*
`zoom`	*Vergrößerung von Bereichen*
`pbaspect`	*Verhältnis der Achsenlängen festlegen*

Kontrolle des Blickwinkels:

`view`	*Einstellung des Blickwinkels*
`rotate3d`	*interaktives Drehen der Grafik*

Beispiele:

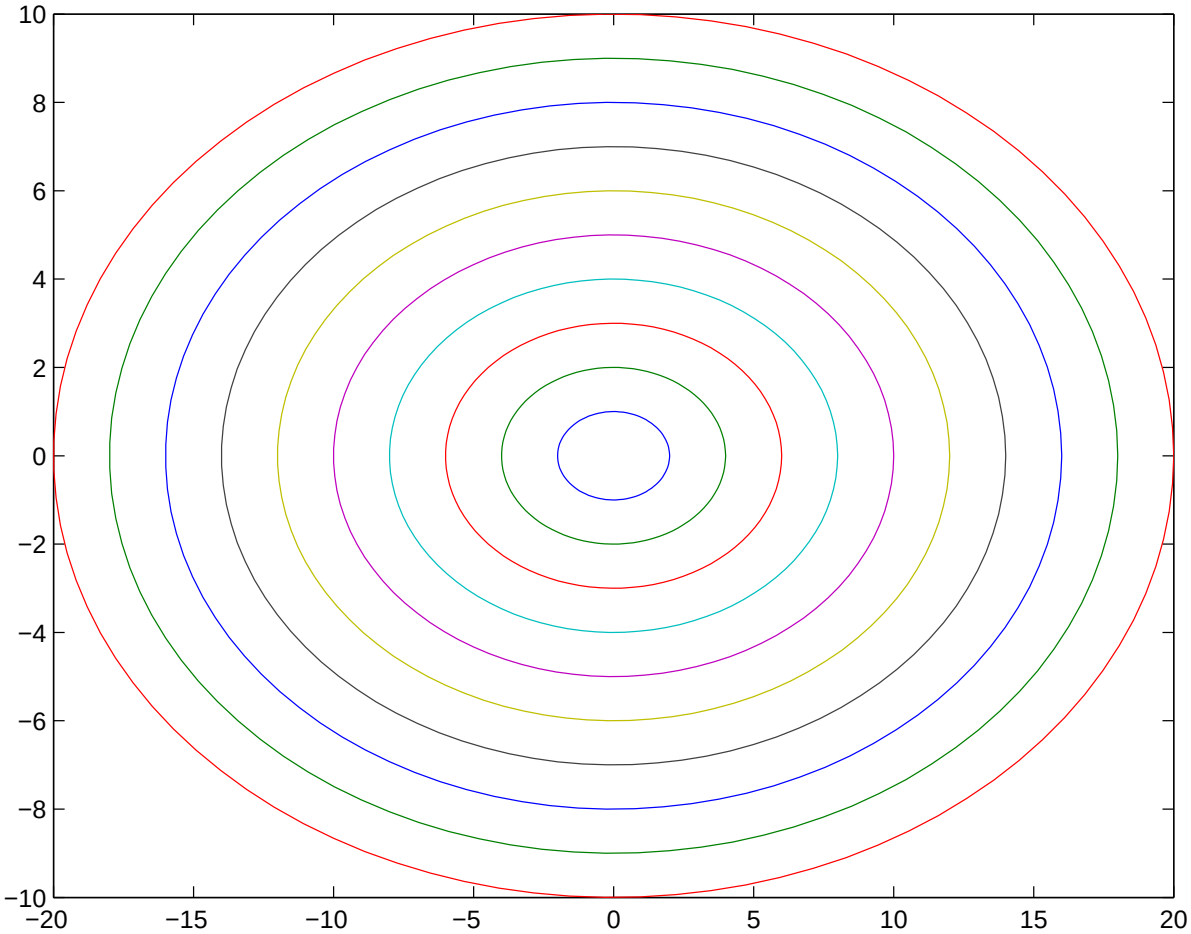
`axis([xmin,xmax,ymin,ymax])`, `axis` *modus*

modus: `auto, manual, tight, equal, square, on, off`

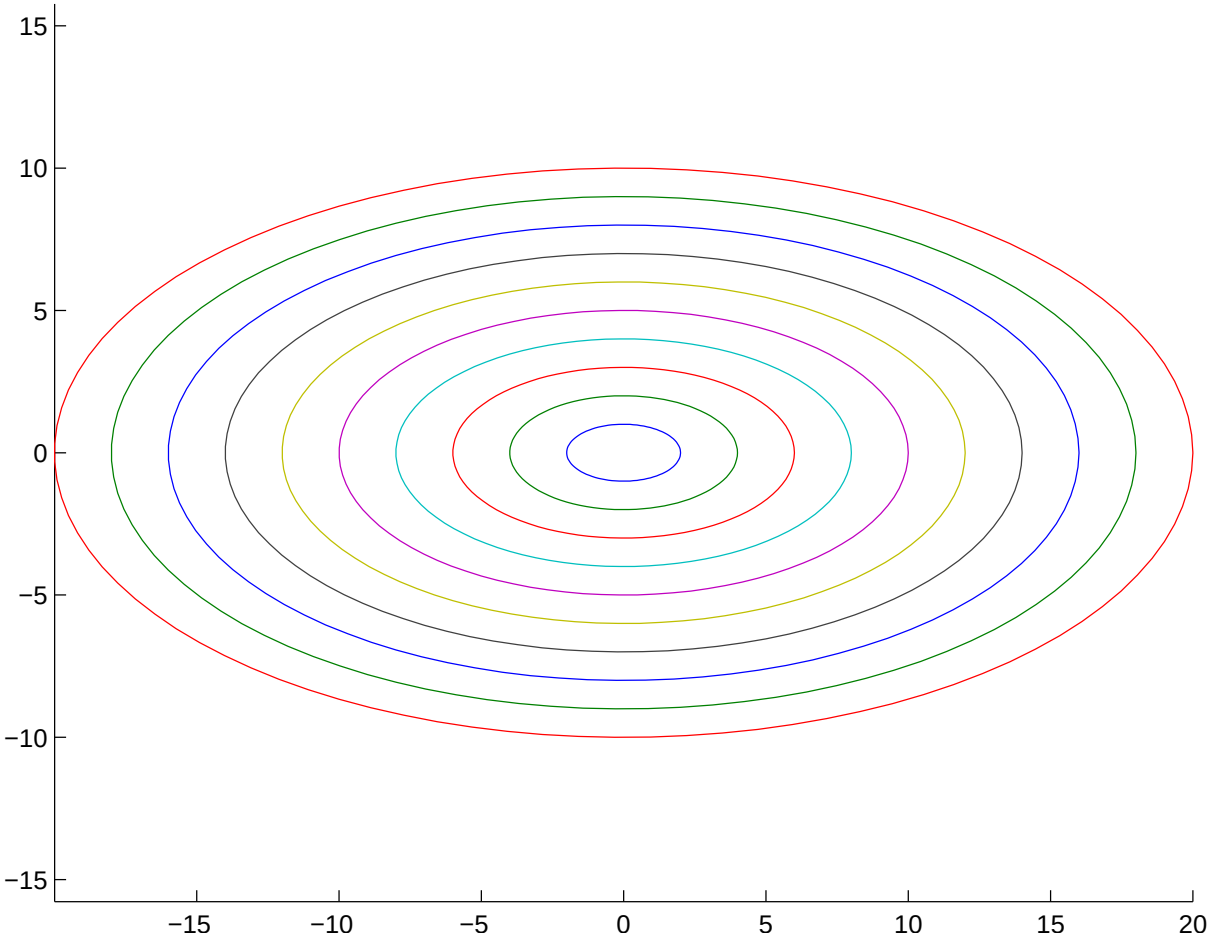
`view([x,y,z])`, `view(`*az*`,`*el*`)`

Beispiel

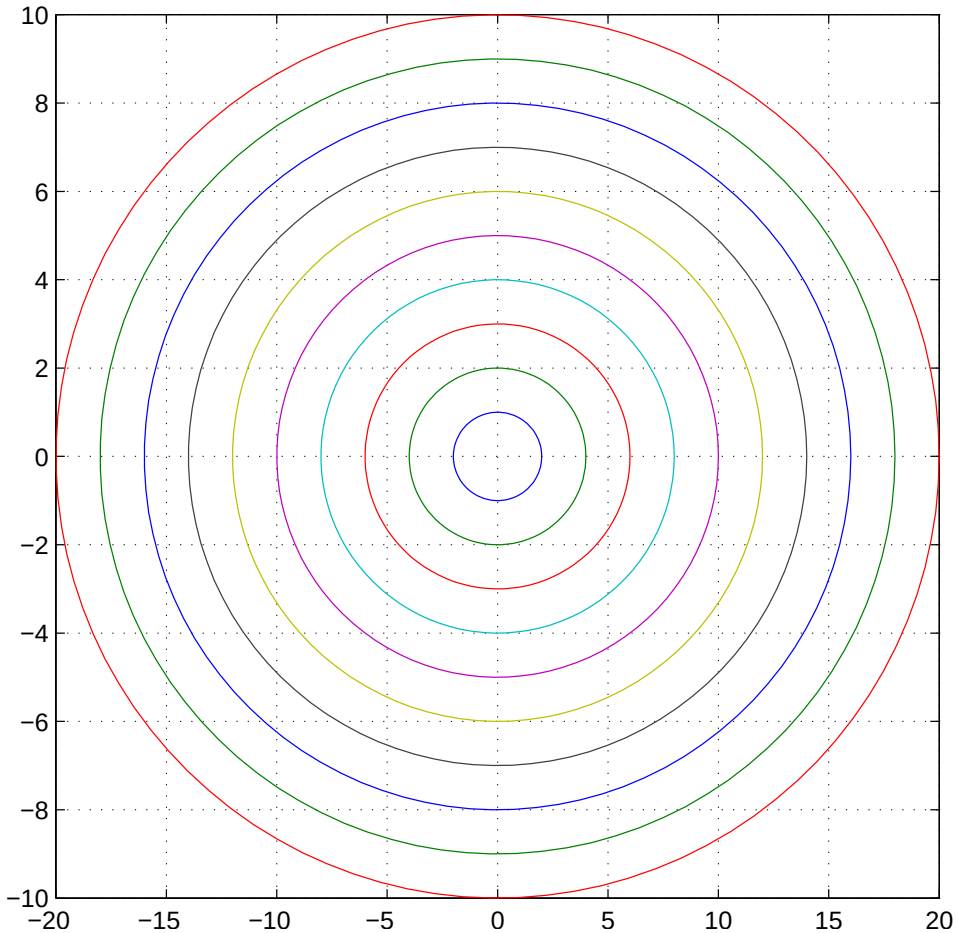
```
>> t=linspace(0,2*pi)';
>> r=1:10;
>> plot(cos(t)*2*r,sin(t)*r)
```



```
>> axis equal
>> box off
>> grid off
```

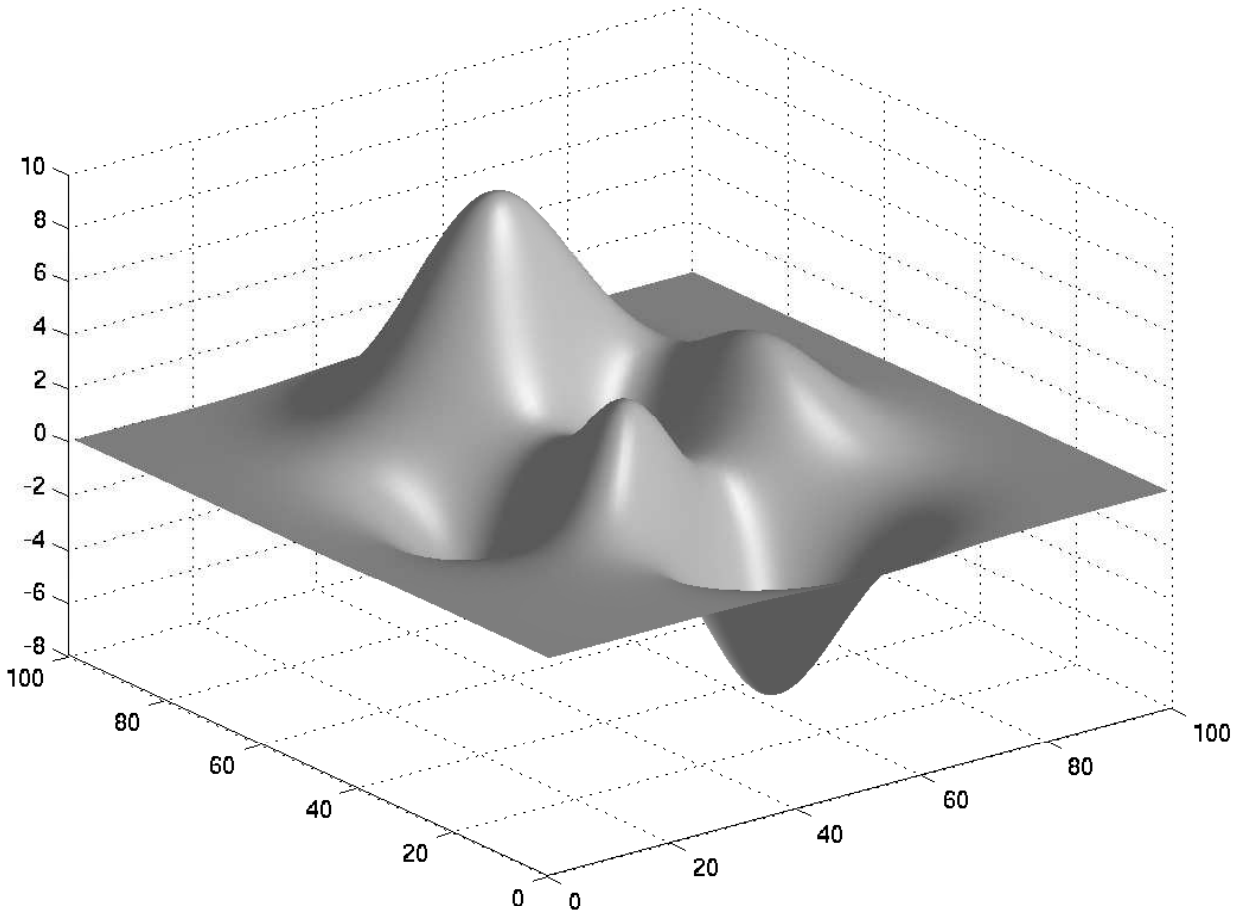


```
>> axis square
>> box on
>> grid on
```

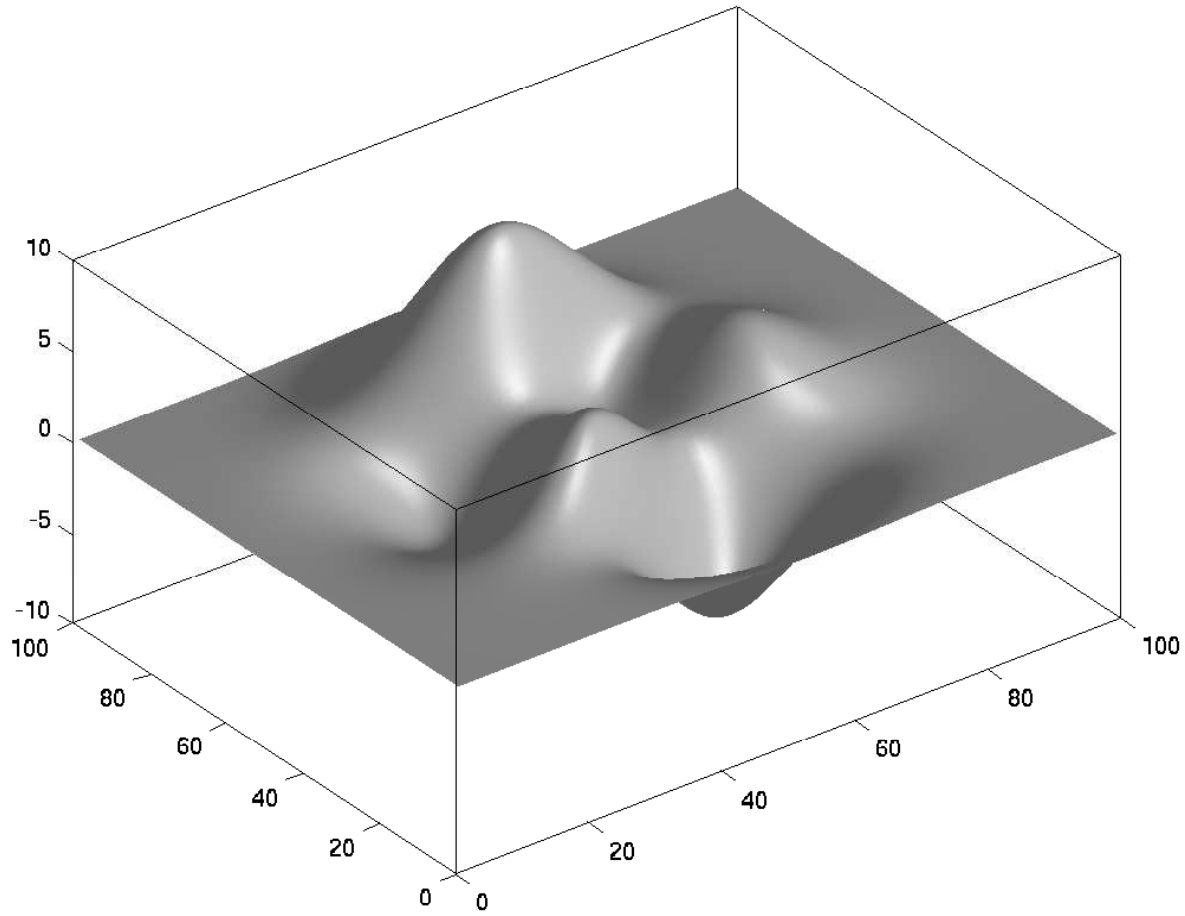


Beispiel

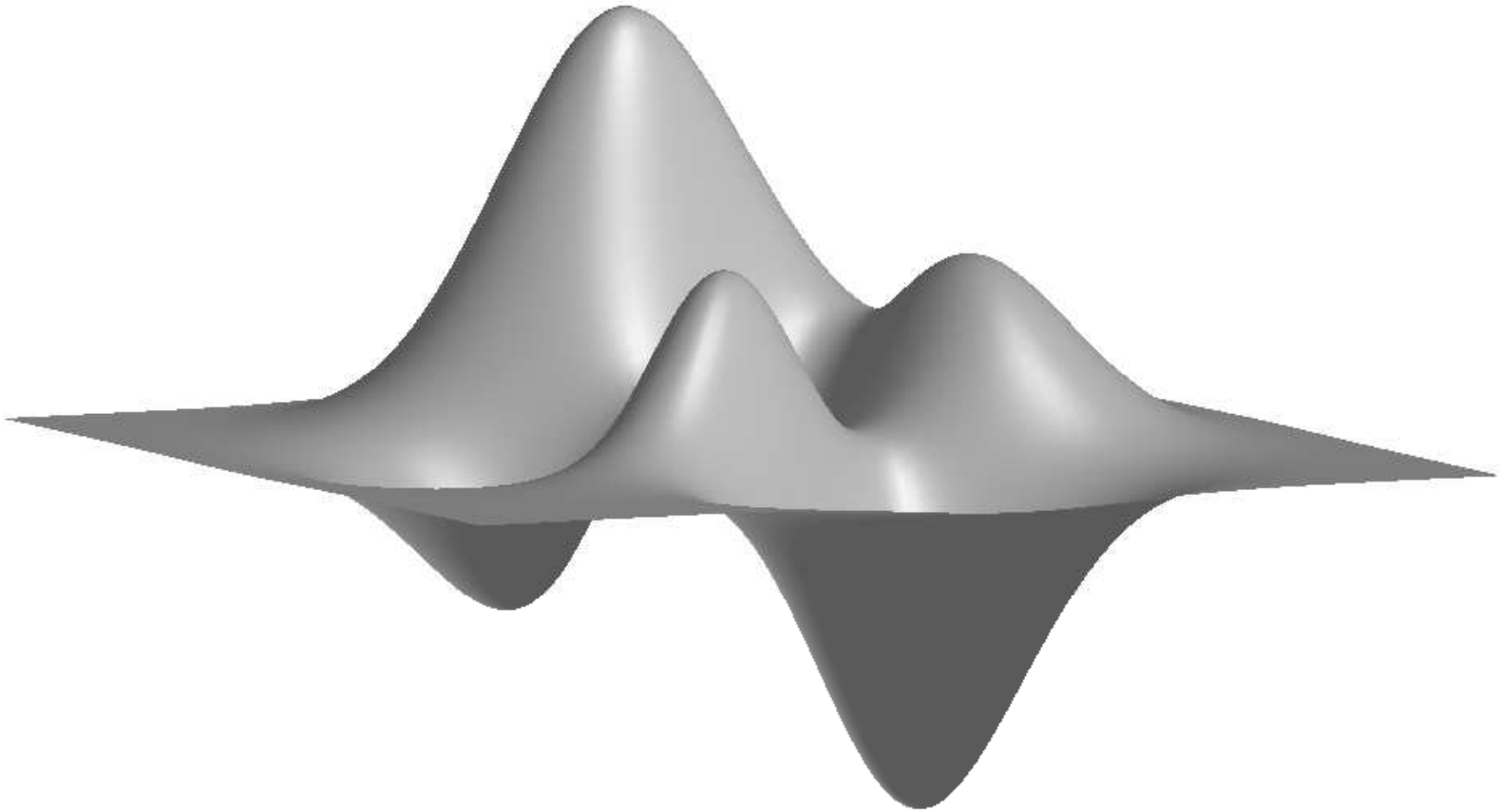
```
>> surfl(peaks(100))
>> colormap(gray(1000))
>> shading interp
```



```
>> pbaspect([4 3 2])
>> box on
>> grid off
```



```
>> pbaspect([3 3 2])
>> axis off
>> view(-25,6)
```



Niveaulinien

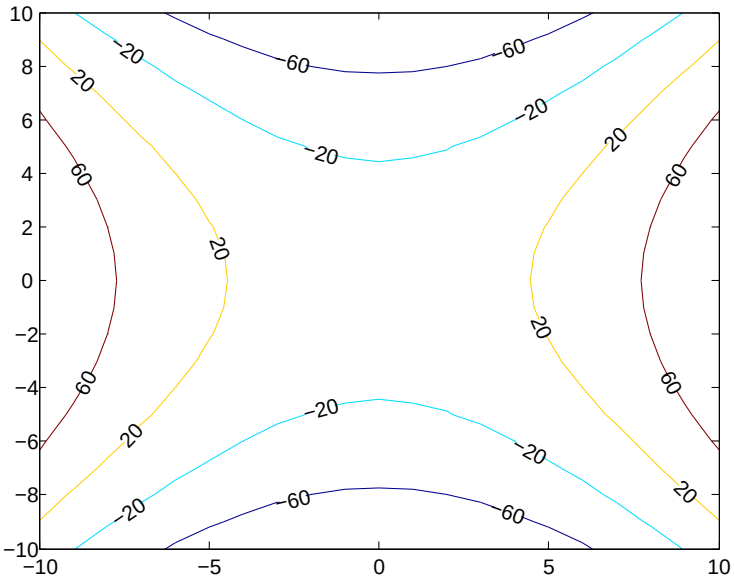
- `[c,h] = contour(X,Y,Z)`: Niveaulinen
- `contourf`: Darstellung mit eingefärbten Bereichen
- `contour3`: dreidimensionale Darstellung

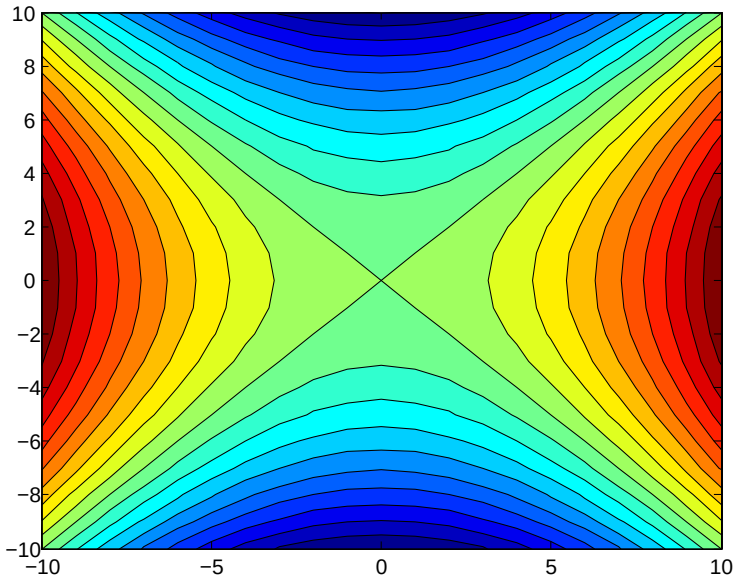
optionale Anzahl `n` oder z-Werte `v` der Niveaulinien
Beschriftung:

- `clabel(c,h)`: z-Werte entlang der Niveaulinien

Beispiel

```
>> [X,Y] = meshgrid(-10:10); Z = X.^2-Y.^2;
>> [c,h] = contour(X,Y,Z,4); clabel(c,h);
>> contourf(X,Y,Z,[-100:10:100]);
```





Visualisierung von Vektorfeldern

`quiver`, `quiver3`	Richtungsfeld
`streamline`	Strömungslinien

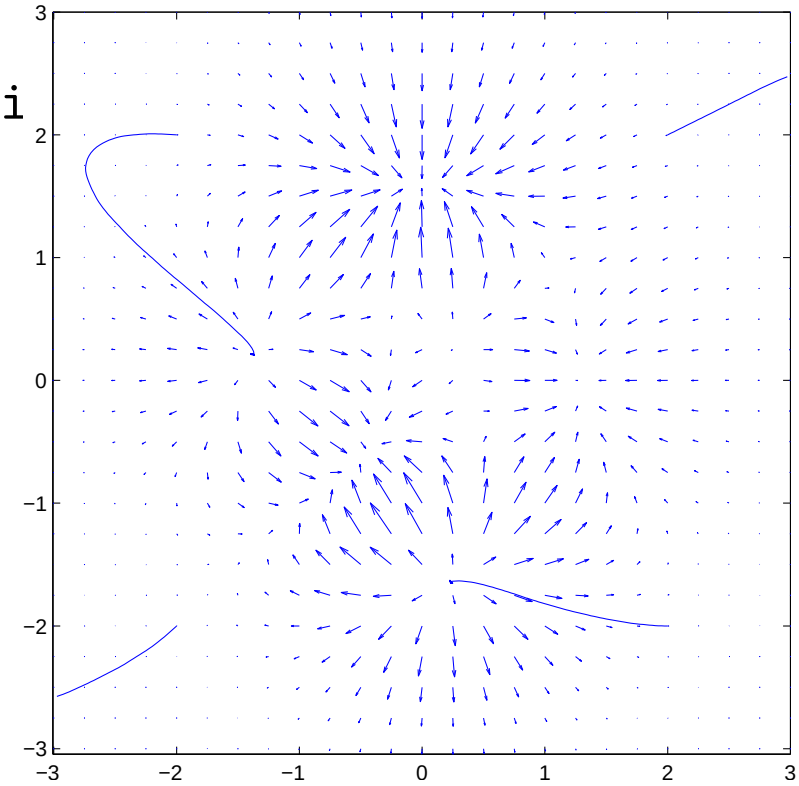
Operatoren

`gradient`	Gradient
`curl`	Rotation
`surfnorm`	Flächennormale

Beispiel

Gradientenfeld und Strömungslinien eines Geländeprofils

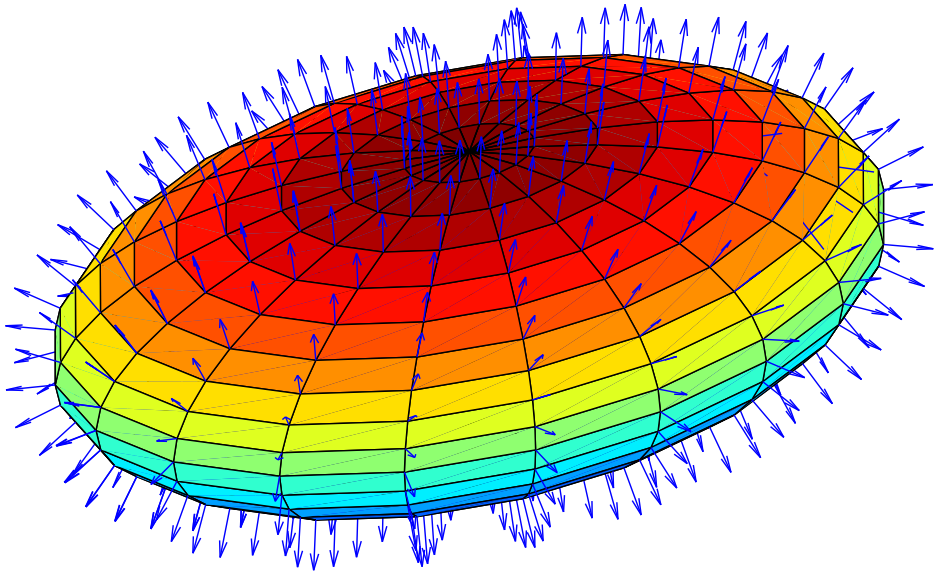
```
>> Gradientenfeld und Strömungsli
>> [xx,yy,zz]=peaks(25);
>> hx=xx(1,2)-xx(1,1);
>> hy=yy(2,1)-yy(1,1);
>> [gx,gy]=gradient(zz,hx,hy);
>> quiver(xx,yy,gx,gy)
>> streamline(xx,yy,-gx,-gy,...
      [-2 2 2 -2],[-2 -2 2 2])
>> axis equal
>> axis tight
```



Beispiel

Ellipsoid mit Normalen

```
>> [xx,yy,zz]=ellipsoid(0,0,0,3,2,1,20);
>> [nx,ny,nz]=surfnorm(xx,yy,zz);
>> surf(xx,yy,zz)
>> hold on
>> quiver3(xx,yy,zz,nx,ny,nz)
>> axis equal, axis off
```



Bilder und Animationen

Bildverarbeitung:

`imread, imwrite`	Lesen und Schreiben von Grafikdateien
`image`	Darstellung von Bildern
`imfinfo`	Ausgabe von Informationen zu einem Bild

Animationen:

`getframe`	Speichern von Animationsframes
`movie`	Abspielen gespeicherter Frames
`im2frame, frame2im`	Konvertierung zwischen Bildern und Movie-Frames
`movie2avi`	Konvertiert MATLAB-Movie in eine AVI-Datei
`aviread`	Einlesen eines AVI-Films

Beispiel

Resonanz bei Schwingungen

```
>> clf; axis([0 1 -2 2]); hold on;
>> t = [0:0.001:1];

>> for k=0:100
>>    w = 10-k/10;
>>    p = sin(2*pi*(10-w)*t);
>>    q = sin(2*pi*(10+w)*t);
>>    h = plot(t,p,t,q,t,p+q);
>>    M(k+1) = getframe;
>>    delete(h);
>> end

>> hold off;
>> % 1 mal abspielen mit Geschwindigkeit 10 frames per s
>> movie(M,1,10);
```

Speichern und Drucken von Grafiken

`print`	*Drucken und Speichern von Grafiken*
`print -d` *Format Dateiname*	*Speichern im angegebenen Format*

(z.B. `bmp`, `epsc`, `jpeg`, `pdf`, `png`, `tiff`*)*

Steuerung der Ausgabe

`orient`	*Einstellen der Papierorientierung*
`printdlg`	*öffnet Dialogfenster mit Druckeinstellungen*
`pagesetupdlg`	*öffnet Dialogfenster mit Seiteneinstellungen*
`printpreview`	*Vorschau der Druckausgabe*

Skripte

Speicherung von Matlab-Befehlen in Textdatei `Dateiname.m`
Ausführung:

```
>> Dateiname
```

Ablaufsteuerung:

`pause`	*Unterbrechung der Programmausführung*
`echo on, echo off`	*Anzeige der Befehle ein bzw. ausschalten*

Kommentare:

% *Text*
%{ *mehrzeiliger Text* %}

Beispiel

Gauss-Elimination

```
>> echo on
>> A = [2 6 4; 1 5 9; 3 7 8]
A = 2   6   4
    1   5   9
    3   7   8
>> pause
>> A(2:3,:) = A(2:3,:) - A(2:3)*A(1,:)/A(1,1)
A = 2   6   4
    0   2   7
    0  -2   2
>> pause
>> A(3,2:3) = A(3,2:3) + A(2,2:3)
A = 2   6   4
    0   2   7
    0   0   9
```

if-Abfrage

Syntax:

```
if logischer Ausdruck
   Befehle
elseif logischer Ausdruck
   Befehle
else
   Befehle
end
```

Indikatorfunktionen:

`isempty`, `isstr`, `ischar`, `isinf`, `isnan`, `isfinite`.

Beispiel

Signum s einer Zahl x

$$s(x) = \begin{cases} 1 & \text{für } x > 0, \\ 0 & \text{für } x = 0, \\ -1 & \text{für } x < 0 \end{cases}$$

Berechnung in MATLAB

```
if x>0
  s=1;
elseif x<0
  s=-1;
else
  s=0;
end
```

alternativ

```
s=(x>0)-(x<0);
```

switch-Anweisung

Syntax:

```
switch Ausdruck
  case Wert
    Befehle
  case {Wert1,Wert2,...,Wertn}
    Befehle
  otherwise
    Befehle
end
```

Wert von *Ausdruck*: skalare Größe oder Zeichenkette

Beispiel

Switch-Anweisung zur Ausgabe von Informationen über eine Zahl *n*:

```
switch n
  case {1,4,9}
    fprintf('%d ist eine Quadratzahl\n',n);
  case {2,3,5,7}
    fprintf('%d ist eine Primzahl\n',n);
  case 6
    fprintf('%d hat zwei Primfaktoren: 2 und 3\n',n);
  case 8
    fprintf('%d ist eine Kubikzahl\n',n);
  case {1,7}
    % Dieser Zweig wird nie erreicht, da die Fälle
    % 1 und 7 bereits zuvor definiert wurden
  otherwise
    disp('n muss natürliche Zahl zwischen 1 und 9 sein.');
end
```

for-Schleife

Syntax:

```
for Variable = Matrix/Cell/Feld
    Befehle
end
```

Variable durchläuft erste Spalte von *Matrix/Cell/Feld*

Unterbrechung der Befehlssequenz

`break`	*Abbruch der Schleife*
`continue`	*nächste Iteration*

Beispiel

Monte-Carlo-Schätzung der Kreiszahl $\pi \approx 3.1416$

- wähle zufällig n Punkte $(p_1, p_2) \in [0, 1)^2$
- bestimme die Anzahl z der Punkte mit $p_1^2 + p_2^2 < 1$
- $\pi/4 \approx z/n$

Vergleich der Laufzeiten unterschiedlicher Implementierungen
`tic` (Start einer Stoppuhr) `toc` (Anhalten der Stoppuhr)

Implementierung mit einer for-Schleife über die Anzahl der Tests:

```
tic
n=10^6;
z=0;
for k=1:n
  p=rand(2,1);
  z=z+(p(1).^2+p(2).^2<1);
end
pi=4*z/n
toc
```

Ausgabe:

```
pi =
    3.1432
Elapsed time is 18.367863 seconds.
```

Implementierung mit einer for-Schleife über die Spalten einer Zufallspunktematrix:

```
tic
n=10^6;
z=0;
for p=rand(2,n)
  z=z+(p(1).^2+p(2).^2<1);
end
pi=4*z/n
toc
```

Ausgabe:

```
pi =
    3.1453
Elapsed time is 11.304552 seconds.
```

Implementierung ohne eine for-Schleife:

```
tic
n=10^6;
P=rand(2,n);
z=sum(P(1,:).^2+P(2,:).^2<1);
pi=4*z/n
toc
```

Ausgabe:

```
pi =
    3.1403
Elapsed time is 0.388576 seconds.
```

while-Schleife

Syntax:

```
while logischer Ausdruck
   Befehle
end
```

Unterbrechung der Befehlssequenz:

`break`	*Abbruch der Schleife*
`continue`	*nächste Iteration*

Beispiel

Pfaffs Approximation von π durch $6 * 2^n$-Ecke

```
% Startwerte:
% dreifache Kantenlängen des ein- und umbeschriebenen 6-Ecks
n=0; a=3; b=2*sqrt(3);

% Iteration von Pfaff
while b-a > 1e-4
  n=n+1;
  if n > 100
     disp('Abbruch nach 100 Iterationen'); break;
  end
  b= 2*a*b/(a+b); a=sqrt(a*b);
end
```

Resultat

$$a = 3.141584 < \pi < 3.141610 = b$$

Funktionen

gespeichert in Datei *Funktionsname.m*

`function` [*Rückgabevariable*, . . .]=*Funktionsname*(*Parameter*, . . .)
  % Kurzbeschreibung für Stichwortsuche
  % . . . Beschreibung . . .

  . . . *Befehle*
end

lokale Funktionen

Verlassen der Funktion durch `return`

Beispiel

Fibonacci-Zahlen z_n mit Rekursion

$$z_1 = z_2 = 1, \quad z_n = z_{n-1} + z_{n-2},\ n > 2$$

```
function z = fibonacci(n)
% n-te Fibonacci-Zahl z

z_last = 0; z = 1;
for k=2:n
   z_save = z_last;
   z_last = z;
   z = z + z_save;
end
```

```
  >> z = fibonacci(10)
        z = 55
```

Speicherung aller berechneten Fibonacci-Zahlen
$\rightsquigarrow$ elementares Programm

```
z(k) = z(k-1) + z(k-2)
```

einzeilige Programm-Version

```
z = [1; 0] * ([0 1; 1 1]^n) * [0; 1];
```

Rekursive Programmversion:

```
function z = fibonacci(n)
% FIBONACCI n-te Fibonacci-Zahl z

if n<3
   z = 1;
else
   z = fibonacci(n-1) + fibonacci(n-2);
end
```

Beispiel

Binomialkoeffizient

```
function c = binomial(n,k)
% BINOMIAL binomial coefficient
% c = binomial(n,k)
% n,k: nonnegative integers with k <= n
% c: n choose k

if k < 0 | k > n | n < 0
   disp('invalid input'); return;
end

c = 1;
for m = 1:k
   c = c * (n+1-m) / m;
end
```

Aufruf der Funktion

```
function c = binomial(n,k)

>> c = binomial(5,2)
>> c
   10
>> binomial(3,4)
>> invalid input
>> binomial(4,3)
>> ans =
   4
```

Darstellung der Kommentare im Funktionskopf mittels `lookfor` und `help`:

```
>> lookfor binomial
 BINOMIAL binomial coefficient

>> help binomial
 BINOMIAL binomial coefficient
 c = binomial(n,k)
 n,k: nonnegative integers with k <= n
 c: n choose k
```

Programmierung mit Hilfe einer Unterfunktion:

```
function c = binomial(n,k)
c = product(n)/(product(n-k)*product(k));
end
function p = product(m)
p = 1;
for k = 2:m, p = p*k; end
end
```

Programmierung mit Rekursion:

```
function c = binomial(n,k)
if n == 0 | k == 0 | k == n
   c = 1;
else
   c = binomial(n-1,k-1) + binomial(n-1,k);
end
```

Beispiel

Newton-Verfahren:

```
function x = newton(f, df, x)
% Newtonverfahren zur Bestimmung einer Nullstelle
% einer Funktion f mit Ableitung df nahe bei x
max_iter = 100; tol = 1.0e-10;
for k=1:max_iter
   fx = f(x); dfx = df(x);
   if abs(fx) < tol
      display('Nullstelle bestimmt'); return;
   elseif abs(dfx) < tol
      display('waagrechte Tangente'); return;
   end;
   x = x - fx/dfx;
end
display('keine Konvergenz');
```

Übergabe der Funktion und ihrer Ableitung als function-handles

```
>> f = @(x) x^3-x; df = @(x) 3*x^2-1;
>> x = newton(f, df, 4)
x =
  1.0000
```

kurze Programmvariante bei gesicherter Konvergenz

```
...

while abs(f(x)) > eps
   x = x - f(x)/df(x);
end

...
```

Ein- und Ausgabeparameter von Funktionen

Funktion zur Kontrolle der Ein- und Ausgabeparameter:

`nargin`	*Anzahl der Eingabeparameter*
`nargout`	*Anzahl der Ausgabeparameter*
`exist`	*prüft, ob eine Variable existiert*
`varargin`	*Eingabeparameterliste unbestimmter Länge (cell-array)*
`varargout`	*Ausgabeparameterliste unbestimmter Länge*
`nargchk`	*prüfen der Eingabeparameteranzahl*
`nargoutchk`	*prüfen der Ausgabeparameteranzahl*

Übergabe von Funktionen als:

`string`	*Name der m-Datei, Aufruf mit feval*
`functionhandle`	

Beispiel

```
function [Vol, Ob] = zylinder(h, R, r)
% Volumen und Oberfläche (optional) eines Hohl- oder Vollzylinders
% mit Höhe h, Aussenradius R und Innenradius r (optional)

if nargin < 2
   error('zu wenige Parameter')
end

if nargin == 2
   r = 0
elseif r >= R
   error('Aussenradius nicht größer als Innenradius')
end

Vol = pi*h*(R^2 - r^2);

if nargout == 2
   Ob = 2*pi*(h*(r+R) + R^2-r^2);
end
```

Beispiel

```
function [x,fx] = steffensen(f,x,tol,max_it)
% STEFFENSEN zero of f(x) by Steffensen's method
% function [x,fx] = steffensen(f,x,tol,max_it)
% f: function handle
% x: approximation of a zero, updated
% tol: bound for |fx|, fx = f(x), for termination
% max_it: maximal number of iterations

% set defaults
if nargin < 4
   max_it = 100;
   if nargin < 3
      tol = 1e-10;
      if nargin < 2 disp('too few arguments'), return; end
   end
end
```

```
for n = 1:max_it
    fx = f(x);
    % Abbruch bei erreichter Genauigkeit
    if abs(fx) < tol
        return
    end
    d = f(x+fx)-fx;
    if abs(d) < eps*fx
        disp('nearly zero denominator');
        return
    end
    % Iterationsschritt
    x = x - fx*fx/d;
end
disp('no convergence');

end
```

```
>> x = steffensen(@cos,0)
   x =
       1.5708

>> fct = @(x) x^2-2
>> [x,fx] = steffensen(fct,1,1.0e-15,100)
   x =
       1.4142
   fx =
       4.4409e-16
```

Datenpfad

Verzeichnisbefehle:

`pwd`	*Ausgabe des aktuellen Arbeitsverzeichnisses*
`cd`	*Wechseln des Arbeitsverzeichnisses*
`dir`	*Ausgabe des Verzeichnisinhalts*
`rmdir`	*Verzeichnis löschen*
`mkdir`	*Verzeichnis erstellen*

Pfadbefehle:

`path`	*Ausgabe bzw. Durchsuchen des* MATLAB*-Pfads*
`addpath`	*Verzeichnis in den Pfad aufnehmen*
`rmpath`	*Verzeichnis aus dem Pfad löschen*
`savepath`	*aktuellen Pfad speichern*
`pathtool`	*Interaktive Bearbeitung des Pfads*
`which`	*Angabe des Pfads zu einer Funktion*

Globale und persistente Variablen

`global` *Variablenname(n)*

deklariert Variablen mit globaler Gültigkeit
funktionsübergreifend

`persistent` *Variablenname(n)*

globaler Charakter innerhalb der deklarierten Funktion
nicht sichtbar für andere Funktionen

Beispiel

Funktionen `fa`, `fb` mit persistenter Variable `PVAR` und globaler Variable `GVAR`

```
function fa
persistent PVAR
global GVAR
GVAR=GVAR+1;
if isempty(PVAR)
  PVAR=1;
else
  PVAR=PVAR+1;
end
fprintf('Funktion fa: PVAR=%d, GVAR=%d\n',PVAR,GVAR);

function fb ...
```

Testlauf:

```
>> global GVAR
>> GVAR=0;
>> fa, fa, fb, fa, fb
Funktion fa: PVAR=1, GVAR=1
Funktion fa: PVAR=2, GVAR=2
Funktion fb: PVAR=1, GVAR=3
Funktion fa: PVAR=3, GVAR=4
Funktion fb: PVAR=2, GVAR=5
```

Befehle zur Benutzer Interaktion

`input`	*Eingabeaufforderung an den Benutzer*
`inputdlg`	*Eingabefeld in einem Dialogfenster*
`keyboard`	*zweitweilig Übergabe der Kontrolle an den Benutzer*
`uigetfile`	*Standarddialog zur Auswahl einer Eingabedatei*
`uigetdir`	*Standarddialog zur Auswahl eines Verzeichnisses*
`uiputfile`	*Standarddialog zur Auswahl einer Ausgabedatei*
`msgbox`	*Meldungsfenster*
`errordlg`	*Dialogfenster für Fehlermeldungen*
`helpdlg`	*Dialogfenster für Hilfestellungen*
`questdlg`	*Dialogfenster für Abfragen*
`warndlg`	*Dialogfenster für Warnmeldungen*
`ginput`	*graphische Eingaben mit der Maus*

Beispiel

Beispiel: input-Werte

```
>> A = input('2x2-Matrix eingeben: ')
2x2-Matrix eingeben: [1 2; 3 4]
A =
  1   2
  3   4
```

Beispiel: input-Zeichenkette

```
>> fct = input('Funktionsnamen eingeben: ');
Funktionsnamen eingeben: 'sin'
fct =
    sin
```

Beispiel

```
function ellipse
% zeichnet eine Ellipse

% Grafikfenster öffnen
clf; hold on;
axis([-10 10 -10 10]);

% Eingabe von Mittelpunkt und Halbachsenlängen
[x,y] = ginput(1);
plot(x, y,'ro');
H = inputdlg({'a:', 'b:'});
a = str2num(H{1}); b = str2num(H{2});

% Zeichnen der Ellipse
t = linspace(0,2*pi);
plot(x+a*cos(t), y+b*sin(t));
```