Einführung in Matlab - Einheit 3

Rekursionen, Grafik

Jochen Schulz

Georg-August Universität Göttingen

Aufbau

1. **Rekursionen**

2. **Einführung Grafik**
   - einfache zweidimensionale Grafiken
   - Beschriftungen
   - Weitere zweidimensionale Darstellungsmöglichkeiten
   - Dreidimensionale Grafiken
   - Animation

Rekursive Funktionen

Rekursive Funktionen sind Funktionen, die sich selbst aufrufen.
Bei jedem Aufruf wird ein neuer lokaler Workspace erzeugt.

Beispiel: Fakultät: $n! = \mathrm{fak}(n)$

$$
\begin{aligned}
n! &= n(n-1)! = n\,\mathrm{fak}(n-1) \\
&= n(n-1)\,\mathrm{fak}(n-2) \\
&= \cdots = n(n-1)\cdots 1
\end{aligned}
$$

Fakultät - rekursiv

```
function res = fak(n)
% fakultaet   berechnet zu einer gegebenen natuerlichen
   Zahl n
%             die Fakultaet n!:=1*2*...*n (rekursiv)
%
%              INPUT: natuerliche Zahl n
%              OUTPUT: Fakultaet fak
%   Jochen Schulz 3.9.2010
if (n == 1)
    res = 1;
else
    res = n*fak(n-1);
end
```

Fakultät - direkt

```
function fak = fak_it(n)
% fakultaet   berechnet zu einer gegebenen natuerlichen
   Zahl n
%             die Fakultaet n!:=1*2*...*n
%
%              INPUT: natuerliche Zahl n
%              OUTPUT: Fakultaet fak
%   Gerd Rapin  10.11.

fak = 1;
for i = 1:n
  fak = fak*i;
end;
```

Fakultät - Zeitvergleich

```
% fak_vergleich.m
% iterativ
tic
for i = 1:100
  fak_it(20);
end
time1 = toc;
fprintf('\nZeitverbrauch direktes Verfahren: %f',time1);
% rekursiv
tic
for i = 1:100
  fak(20);
end
time2 = toc;
fprintf('\nZeitverbrauch rekursives Verfahren: %f\n',
    time2);
```

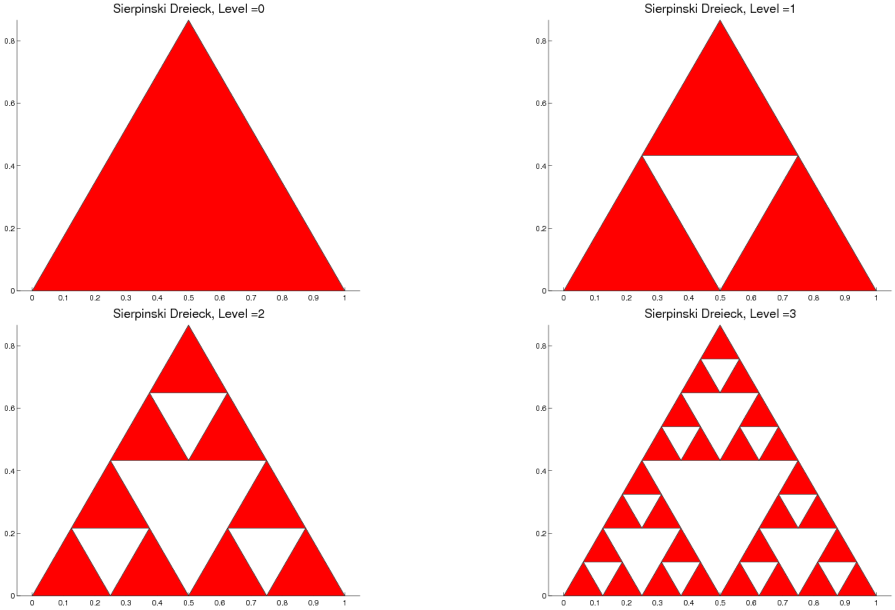
rekursive Implementierung GGT

```
function [a,b] = ggt_rekursiv(a,b)
% ggt_rekursiv berechnet den groessten
% gemeinsamen Teiler (ggT)
if a~=b
  if a>b
    a = a-b;
  else
    b = b-a;
  end;
  [a,b] = ggt_rekursiv(a,b);
end;
```

Sierpinski Dreieck

- Wir beginnen mit einem Dreieck mit Eckpunkten P_a, P_b und P_c.
- Wir entfernen daraus das Dreieck, das durch die Mittelpunkte der Kanten entsteht.
- Die verbliebenden drei Dreiecke werden der gleichen Prozedur unterzogen.
- Diesen Prozess können wir rekursiv wiederholen.
- Das Ergebnis ist das Sierpinski Dreieck.

Sierpinski Dreieck



Implementierung

```
%  sierpinski_plot.m
level=2;

ecke1=[0;0];
ecke2=[1;0];
ecke3=[0.5; sqrt(3)/2];
figure; axis equal;
hold on;
sierpinski(ecke1,ecke2,ecke3,level);
hold off;
title(['Sierpinski Dreieck, Level =' ...
        num2str(level)],'FontSize',16);
```

Implementierung

```
function sierpinski(ecke1,ecke2,ecke3,level)
% Teilt das Dreieck auf in 3 Dreiecke (level>0)
% Plotten des Dreiecks (level=0)

if level == 0
    fill([ecke1(1),ecke2(1),ecke3(1)],...
     [ecke1(2),ecke2(2),ecke3(2)],'r');
else
    ecke12 = (ecke1+ecke2)/2;
    ecke13 = (ecke1+ecke3)/2;
    ecke23 = (ecke2+ecke3)/2;
    sierpinski(ecke1,ecke12,ecke13,level-1);
    sierpinski(ecke12,ecke2,ecke23,level-1);
    sierpinski(ecke13,ecke23,ecke3,level-1);

end;
```

Zeichnen von Polygonen

Ein Polygon sei durch die Eckpunkte $(x_i, y_i)_{i=1}^{n}$ gegeben. Dann kann durch den Befehl

```
fill(x,y,char)
```

dargestellt werden. `char` gibt die Farbe des Polygons an, z.B. rot wäre `'r'`.

Aufbau

1. Rekursionen
2. **Einführung Grafik**
   - einfache zweidimensionale Grafiken
   - Beschriftungen
   - Weitere zweidimensionale Darstellungsmöglichkeiten
   - Dreidimensionale Grafiken
   - Animation

Aufbau

1. Rekursionen
2. **Einführung Grafik**
   - einfache zweidimensionale Grafiken
   - Beschriftungen
   - Weitere zweidimensionale Darstellungsmöglichkeiten
   - Dreidimensionale Grafiken
   - Animation

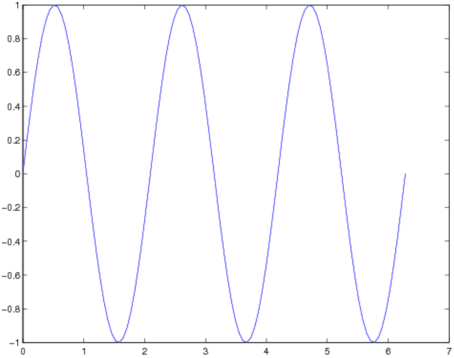
Standard-Plot

```
plot(<x>,<y>)
```

zeichnet für Vektoren $x = (x_1, \ \dots \ , x_N)$ und $y = (y_1, \dots \ , y_N)$ eine Grafik, die die Punkte (x_i, y_i) und (x_{i+1}, y_{i+1}) miteinander verbindet.

Beispiel:

```
x = linspace(0,2*pi,100);
y1 = sin(3*x);
plot(x,y1)
```



Erweiterungen

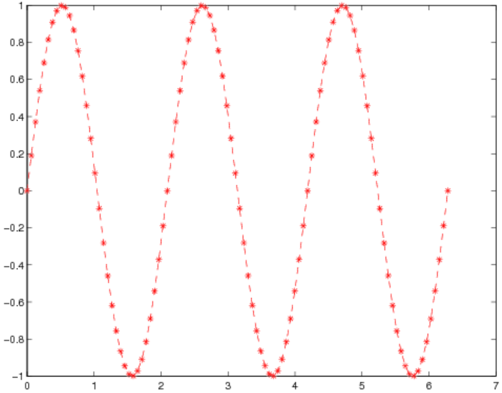
```
plot(<x>,<y>,<string>)
```

String besteht aus drei Elementen, die die Farbe, Linienstil und die Markierung der Punkte kontrollieren. Die Reihenfolge der drei Elemente ist beliebig.

Beispiel: Durch

```
plot(x,y,'r*--')
```

wird die Linie gestrichelt (- -) in rot (r) gezeichnet und die Punkte durch * markiert.



Optionen

Farben	r (rot), g (grün), b (blau), c (hellblau), m (magenta), y (gelb), k (schwarz), w (weiß)
Marker	o (Kreis), * (Stern), . (Punkt), + (Plus), x (Kreuz), s (Quadrat), d (Raute),...
Linien-Stil	- (durchgezogene Linie), -- (gestrichelte Linie), : (gepunktete Linie), -. (Strich-Punkt Linie)

Läßt man den Linien-Stil weg, so werden die Punkte nicht verbunden.

Optionen II

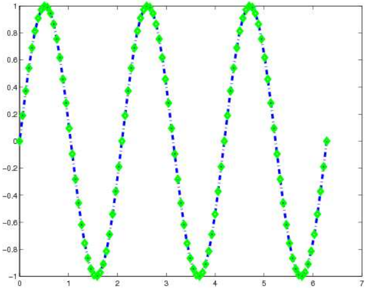
```
plot(<x>,<y>,<string>,<Eigenschaft>, <Spez.>)
```

Eigenschaften:

`'MarkerSize'` (Default 6), `'LineWidth'` (Default 0.5), `'MarkerEdgeColor'`, `'MarkerFaceColor'`

Beispiel:

```
plot(x,y1,'b-.d','LineWidth',...
3,'MarkerEdgeColor','g')
```



Alternativen

- Mehrere Plots in eine Grafik:

```
plot(x1,y1,string1,x2,y2,string2,...)
```

- Logaritmische Skalierung in x- bzw in y-Richtung:

```
semilogx(x1,y1)
```

bzw.

```
semilogy(x1,y1)
```

- Logarithmische Skalierung beider Achsen:

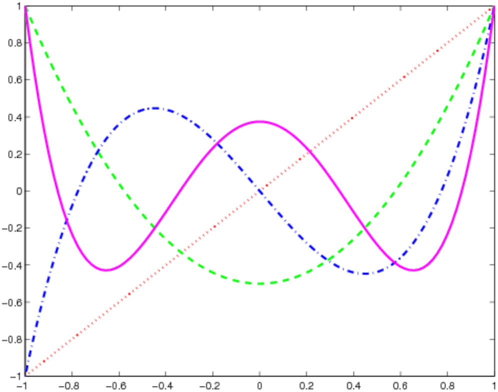
```
loglog(x1,y1)
```

- Ist X ein Vektor mit komplexen Einträgen, so ergibt `plot(X)`

```
plot(real(X),imag(X))
```

Beispiel - Legendre Polynome

```
x = linspace(-1,1,100);
p1 = x;
p2 = (3/2)*x.^2-1/2;
p3 = (5/2)*x.^3-(3/2)*x;
p4 = (35/8)*x.^4 - (15/4)*x.^2+3/8;
plot(x,p1,'r:',x,p2,'g--',x,p3,'b-.',x,p4,'m-','LineWidth',2)
```



Achseneinstellungen

`axis([x1 x2 y1 y2])`	Setzen der x- und y-Achsen Grenzen
`axis auto`	Rückkehr zu Default Achsen Grenzen
`axis equal`	Gleiche Dateneinheiten auf allen Achsen
`axis off`	Enfernen der Achsen
`axis square`	quadratische Achsen-Box
`axis tight`	Achsen Grenzen werden passend zu den Daten gewählt.
`xlim([x1 x2])`	Setzen der x-Achse
`ylim([y1 y2])`	Setzen der y-Achse
`grid on`	Gitter aktivieren
`box on`, `box off`	Box um die Grafik legen, Box entfernen

Aufbau

1. Rekursionen

2. **Einführung Grafik**
   - einfache zweidimensionale Grafiken
   - Beschriftungen
   - Weitere zweidimensionale Darstellungsmöglichkeiten
   - Dreidimensionale Grafiken
   - Animation

Beschriften der Grafik

- Titel: `title('Titel')`
- Achsenbeschriftung: `xlabel('Text')`, `ylabel('Text')`
- Legende: `legend('Text1','Text2',...,nr)`

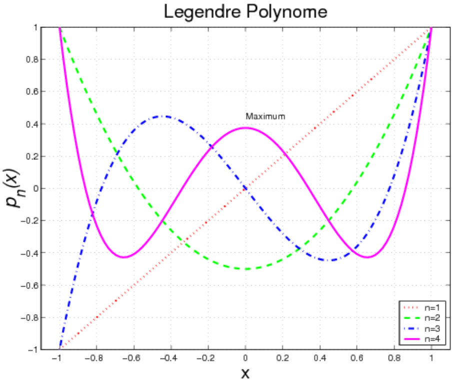
  nr gibt die Position der Legendenbox in der Grafik an: -1 (rechts vom Plot), 0 'bester' Ort, 1 oben rechts (default), 2 oben links, 3 unten links, 4 unten rechts.
- zusätzlicher Text: `text(x,y,'Text')` Plaziert 'Text' an die Position (x, y) bzgl. der Werte auf der x- bzw. y-Achse.

Bemerkungen zur Beschriftung

- In den strings kann direkt eine abgespeckte LaTeX-Notation verwendet werden. (nahezu vollständige LaTeX-Unterstützung: latex-interpreter). Beispiele:
  - `\alpha` $\Rightarrow \alpha$
  - `sin^{3/2}(x)` $\Rightarrow \sin^{3/2}(x)$.
  - `title('f(x)= \frac{1}{x^2+a}','interpreter','latex')` $\Rightarrow$ $f(x) = \frac{1}{x^2+a}$
- Ändern der Schriftgröße, z.B. `title('Titel','FontSize', 20)`.
- Auflistung aller modifizierbaren Texteigenschaften: `doc text_props`

Beispiel - Legendre Polynome II

```
title('Legendre Polynome','FontSize', 20)
xlabel('x','FontSize', 20)
text(0,0.45,'Maximum')
legend('n=1','n=2','n=3','n=4',4)
grid on, box on;
xlim([-1.1,1.1])
```



Umgang mit Grafikfenster

- Öffnen eines (weiteren) Grafikfensters: `figure`. Eine Grafik wird immer im aktuellen Fenster erzeugt. Ist noch kein Fenster geöffnet, so wird ein Fenster erzeugt.
- Durch den Befehl `hold on` werden bestehende Grafiken im aktuellen Fenster erhalten. Neue Grafiken werden den bestehenden hinzugefügt.
- `hold off` (default) überschreibt Grafiken im aktuellen Fenster
- Schliessen: `close`, `close all`

Aufbau

1. **Rekursionen**

2. **Einführung Grafik**
   - einfache zweidimensionale Grafiken
   - Beschriftungen
   - Weitere zweidimensionale Darstellungsmöglichkeiten
   - Dreidimensionale Grafiken
   - Animation

Darstellung von Daten

- Balkendiagramm:

```
bar(<Daten>)
```

- Histogramm:

```
hist(<Daten>,<Anzahl Bars>)
```

- einfacher ausgefüllter Plot:

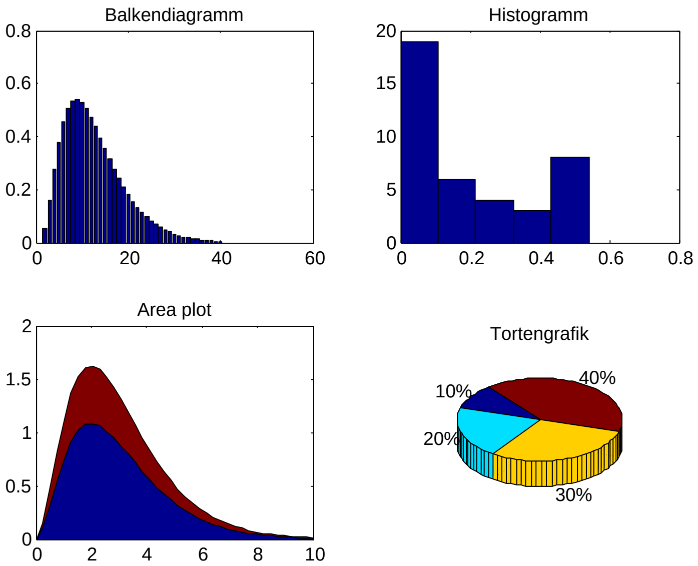
```
area(<x>,[<y1>,<y2>])
```

  (y1 und y2 werden addiert)

- Tortengrafik:

```
pie3([<anteil1> <anteil2> .. <anteilx>])
```

Darstellung von Daten



Darstellung von Daten

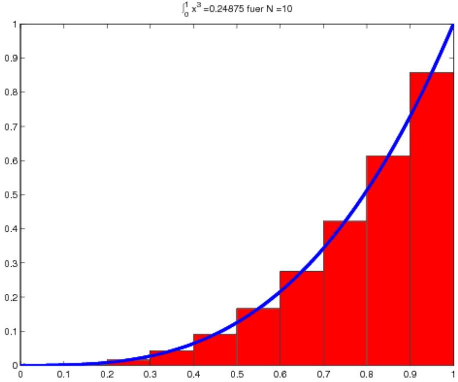
```
n = linspace(0,10,40);
y = n.^2.*exp(-n);

% Balkendiagramm
subplot(2,2,1),
bar(y); title('Balkendiagramm');

% Histogramm
subplot(2,2,2),
hist(y,5); title('Histogramm');

% Area plot
subplot(2,2,3),
area(n,[y',2*y']); title('Area plot');

% Tortengrafik
subplot(2,2,4),
pie3([ 1 2 3 4]); title('Tortengrafik');
```

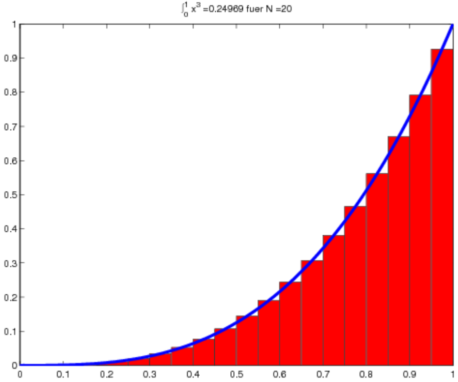
Approximation von Integralen

Approximiere $\int_0^1 f(x)dx$ durch (Mittelpunktsregel)

$$\int_0^1 f(x)dx \approx \sum_{i=1}^{N} \frac{1}{N} f\left(\frac{i-\frac{1}{2}}{N}\right)$$

für gegebenes $N \in \mathbb{N}$. **Beispiel**: $f(x) = x^3$





Integral - Implementation

```
%   integral.m
% berechnet approximativ ein Integral
% ueber  (0,1) durch die Mittelpunktregel
N = 5; % Anzahl Unterteilungen

x = (0+1/(2*N)):(1/N):(1-1/(2*N));
y = x.^3;
% Berechnung des Integrals
result = sum(y)*(1/N);

% Plot
for i = 1:N
    fill([(i-1)/N (i-1)/N i/N i/N],...
        [0 ((i-0.5)/N).^3  ((i-0.5)/N).^3 0], 'r');
    hold on;
end;
plot(0:0.01:1,(0:0.01:1).^3,'LineWidth',3);
title(['\int_0^1 x^3 = ',num2str(result),' fuer N =',
   num2str(N) ]);
```

Aufbau

1. Rekursionen

2. **Einführung Grafik**
   - einfache zweidimensionale Grafiken
   - Beschriftungen
   - Weitere zweidimensionale Darstellungsmöglichkeiten
   - Dreidimensionale Grafiken
   - Animation

Dreidimensionale Grafiken

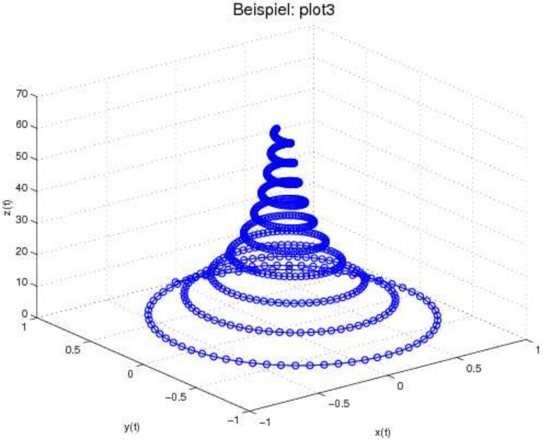
- Dreidimensionale Version von `plot`: `plot3`
- Darstellung von Funktionen $f: \mathbb{R}^2 \to \mathbb{R}$:
  - Contourplot (zeichnet die Niveaulinien): `contour`, `contourf`, `contour3`
  - Darstellung des Graphen mit Gitterlinien: `mesh` , `meshc`
  - Flächige Darstellung des Graphen: `surf`, `surfc`
- Darstellung von Funktionen $f: \mathbb{R}^3 \to \mathbb{R}$:
  - Streifenansichten `slice`

`mesh(X,Y,Z)` z.B. stellt für Matrizen $X, Y, Z \in \mathbb{R}^{n\times k}$ die Punkte

$$(X(i,j), Y(i,j), Z(i,j)) \quad \text{dar.}$$

plot3

Bei gegebenen Vektoren $x = (x_i)_{i=1}^n$, $y = (y_i)_{i=1}^n$, $z = (z_i)_{i=1}^n$ erzeugt `plot3(x,y,z)` einen Plot der die Punkte (x_i, y_i, z_i) und $(x_{i+1}, y_{i+1}, z_{i+1})$ miteinander verbindet.



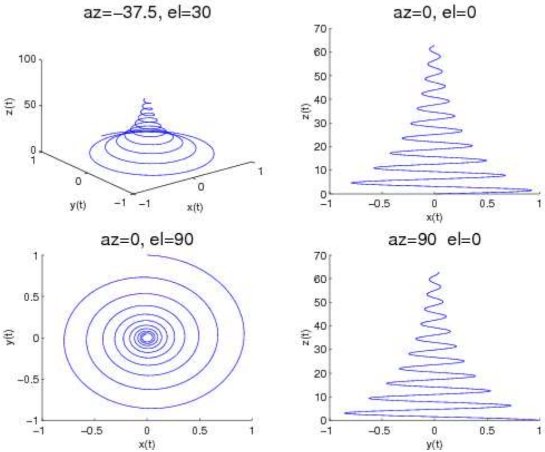
Beispiel plot3

```
t = 0:0.1:20*pi;
x = exp(-t/20).*sin(t);
y = exp(-t/20).*cos(t);
z = t;

plot3(x,y,z,'b-o','LineWidth',1);
grid on
xlabel('x(t)'), ylabel('y(t)');
 zlabel('z(t)');
title('Beispiel: plot3','FontSize',15);
```

Blickwinkel

`view(az,el)`

- `az` ist die horiz. Rotation in Grad (Def. -37.5)
- `el` ist die vertikale Rotation in Grad (Def. 30)



3D-Funktionenplots

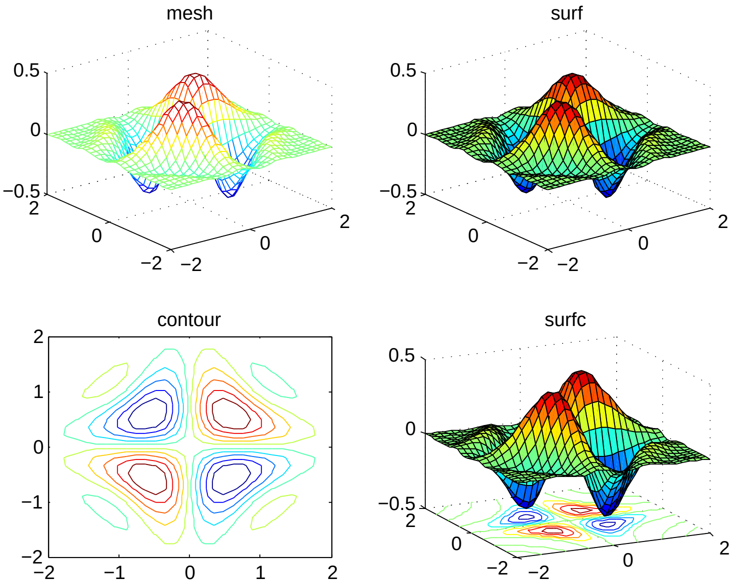
Darstellung von Funktionen

$$f : \mathbb{R}^2 \quad \to \quad \mathbb{R}$$

Beispiel:

$$f(x, y) := \exp(-x^2 - y^2)\,\sin(\pi x y)$$

Beispiel: Funktionenplot



Funktionenplot - Implementation

```
% Erzeugen des Gitters
x = linspace(-2,2,30);
y = linspace(-2,2,30);
[X,Y] = meshgrid(x,y);
% Funktionswerte
Z = exp(-X.^2-Y.^2).*sin(pi*X.*Y);

% verschiedenen Darstellungen
subplot(2,2,1),
 mesh(X,Y,Z), title('mesh');
subplot(2,2,2),
 surf(X,Y,Z), title('surf');
subplot(2,2,3),
 contour(X,Y,Z,10), title('contour');
subplot(2,2,4),
 surfc(X,Y,Z);
 view(-26,20), title('surfc');
```

subplot

```
subplot(<n>,<m>,<k>)
```

zerlegt das Grafikfenster in $n \times m$ Teilfenster.
Die Zahl $1 \leq k \leq nm$ gibt an, welches Teilfenster gerade aktiv ist.
Durchnumeriert wird zeilenweise, also $(1, 1), (1, 2), \ldots$.

meshgrid

Zu Vektoren $x = (x_i)_{i=1}^k$, $y = (y_j)_{j=1}^n$ erzeugt

```
[X,Y]=meshgrid(x,y)
```

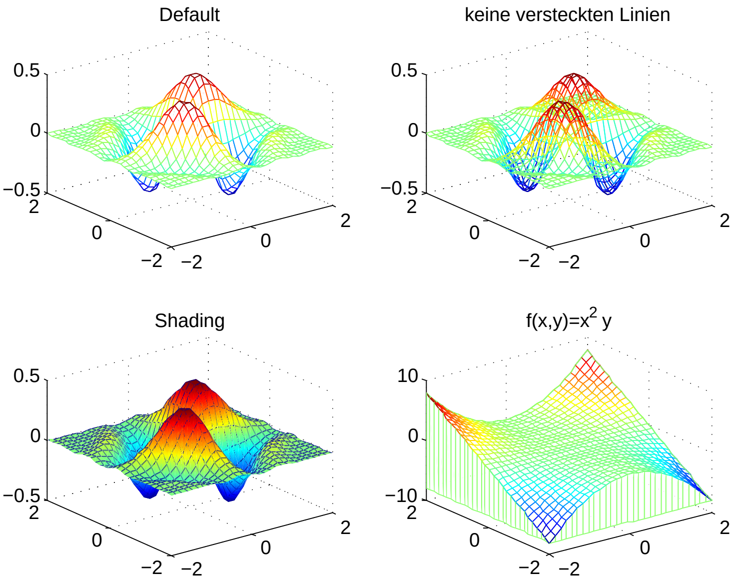
Matrizen $X, Y \in \mathbb{R}^{n\times k}$, wobei jede Zeile von X eine Kopie des Vektors x ist und Y als Spalten den Vektor y enthält.
Dann hat `Z=X.*Y` die Komponenten

$$Z(i,j) = x(j) * y(i).$$

Weitere Möglichkeiten

- Darstellung versteckter Linien (bei `mesh`): `hidden off`, Default: `hidden on`
- Verschmieren des Gitters: `shading('interp')`
- Blickwinkel: `view(az,el)`
- ähnlich wie `mesh`; nur mit 'Vorhang': `meshz(X,Y,Z)`

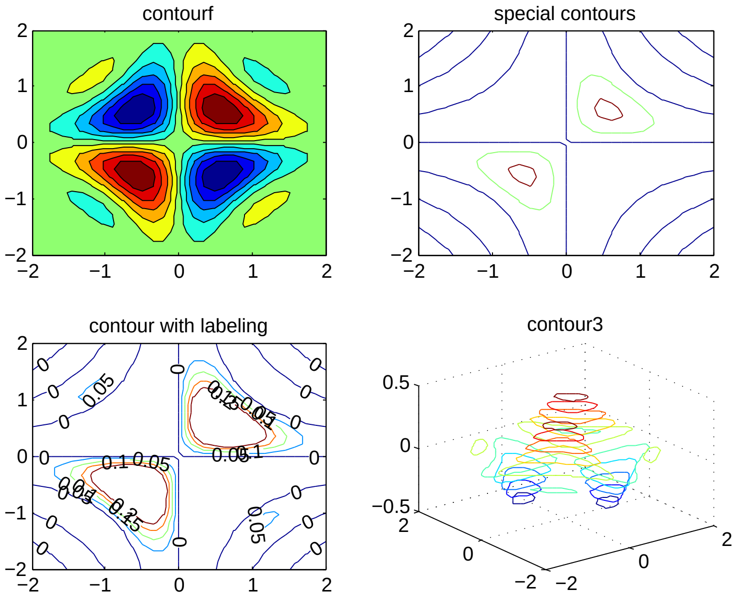
Beispiel: Funktionenplot



Programm

```
x = linspace(-2,2,30);
y = linspace(-2,2,30);
[X,Y] = meshgrid(x,y);
% Funktionswerte
Z = exp(-X.^2-Y.^2).*sin(pi*X.*Y);

% verschiedenen Darstellungen
subplot(2,2,1),
 mesh(X,Y,Z), title('Default');
subplot(2,2,2),
 mesh(X,Y,Z), hidden off,
 title('keine versteckten Linien');
subplot(2,2,3), surf(X,Y,Z);
 shading('interp'), title('Shading');
subplot(2,2,4), Z=X.^2.*Y;
 meshz(X,Y,Z), title('f(x,y)=x^2 y');
```

Contour Plots



Contour Plots - Listing

```
% Erzeugen des Gitters
x=linspace(-2,2,30);
y=linspace(-2,2,30);
[X,Y]=meshgrid(x,y);
% Funktionswerte
Z=exp(-X.^2-Y.^2).*sin(pi*X.*Y);

% verschiedene Darstellungen
subplot(2,2,1),
 contourf(X,Y,Z,10), title('contourf')
subplot(2,2,2),
 contour(X,Y,Z,[0 0.2 0.4]), title('special contours');
subplot(2,2,3),
 [C,h]=contour(X,Y,Z,[0 0.05 0.1 0.15 0.2 ]);
 title('contour with labeling');
 clabel(C,h)
subplot(2,2,4),
  contour3(X,Y,Z,10), title('contour3')
```

Erläuterungen zu Contour-Befehlen

- `contour(X,Y,Z,n)` zeichnet für $n \in \mathbb{N}$ n-Konturlinien. Ist n ein Vektor, werden Konturlinien zu den Werten in dem Vektor n geplottet.
- `contourf` funktioniert wie `contour` nur das die Flächen zwischen den Konturlinien ausgefüllt werden.
- `clabel(C,h)` beschriftet die Konturlinien, deren Werte in C gespeichert sind und die zum Grafik-Handle h gehören.
- `contour3` zeichnet jede Konturlinie auf einer anderen Höhe.

Slice

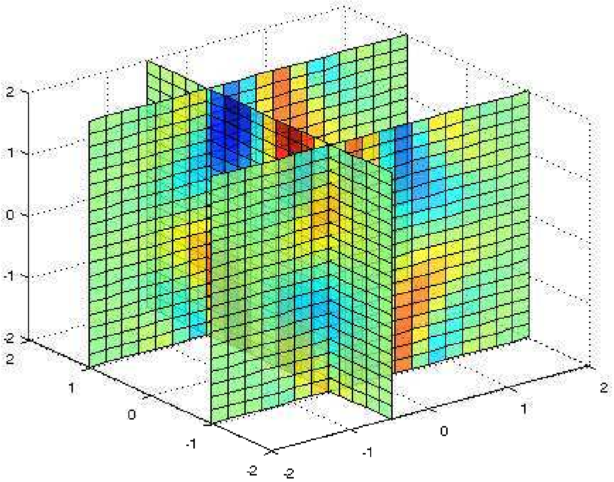
```
slice(X,Y,Z,V,sx,sy,sz)
```

zeichnet Schnitte zu den Funktionswerten $V(i)$ zu $(X(i), Y(i), Z(i))$.
Schnitte sind durch die Vektoren *sx*, *sy* und *sz* gegeben.
Beispiel:

$$f(x, y, z) := \exp(-x^2 - y^2) \sin(\pi xyz)$$

```
x = linspace(-2 ,2 ,20);
[X,Y,Z] = meshgrid (x, x, x) ;
V = exp(-X.^2-Y.^2) .* sin(pi*X.*Y.*Z);
sx = [-0.5  ] ; sy = [ -1, 1 ] ;
sz = [ ] ;
slice (X,Y, Z,V ,sx ,sy ,sz );
alpha (0.8 ) % Transparency
```

Beispiel: slice



Aufbau

Animation-Beispiel

```
% animation.m

clear all;
[X,Y] = meshgrid(-1:0.05:1,-1:0.05:1);

for j = 1:60
    Z = cos(j^0.5*pi*exp(-X.^2-Y.^2));
    mesh(X,Y,Z);
    %surf(X,Y,Z);
    shading interp
    F(j) = getframe;
end
% Abspielen des Movies
movie(F,1);
```

Erstellen einer Animation

- Mit `F(j)=getframe` wird die aktuelle Grafik in das Array F gespeichert.
- Sequenz der Bilder F darstellen: `movie(F,n,fps)`, wobei n die Anzahl der Wiederholungen angibt und *fps* der gezeigten Frames pro Sekunde entspricht (Default: $n = 1$, $fps = 12$).
- Speichern des Movies in AVI Format: `movie2avi(F,Dateiname)`