

# Wissenschaftliches Rechnen mit Matlab/Python

## Einheit 3 - Rekursionen, Grafik

Jochen Schulz

Georg-August Universität Göttingen 

## 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:  $n! = \text{fak}(n)$

$$\begin{aligned}n! &= n(n-1)! = n \text{ fak}(n-1) \\&= n(n-1) \text{ fak}(n-2) \\&= \dots = n(n-1) \dots 1\end{aligned}$$

```
function res = fak(n)
if (n == 1)
    res = 1;
else
    res = n*fak(n-1);
end
```

```
def fak(n):
    if (n == 1):
        res = 1
    else:
        res = n*fak(n -1)
    return res
```

```
function fak = fak_it(n)
fak = 1;
for i = 1:n
    fak = fak*i;
end;
```

```
def fak_it(n):
    fak = 1
    for i in range(1,n+1):
        fak = fak*i
    return fak
```

# 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);
```

```
%timeit fak_it(200)
```

# rekursive Implementierung GGT

```
function [a,b] = ggt_rekursiv(a,b)
if a~=b
    if a>b
        a = a-b;
    else
        b = b-a;
    end;
    [a,b] = ggt_rekursiv(a,b);
end;
```

```
def ggt_rekursiv(a,b):
    if a != b:
        if a>b:
            a -= b
        else:
            b -= a
        return ggt_rekursiv(a,b)
    return a
```

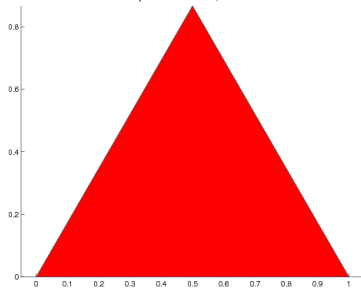
# Sierpinski Dreieck

- 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.

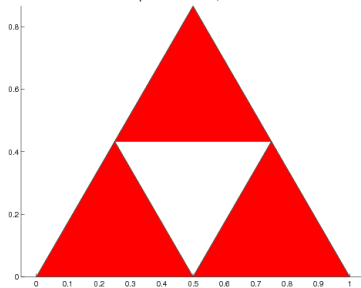


# Sierpinski Dreieck

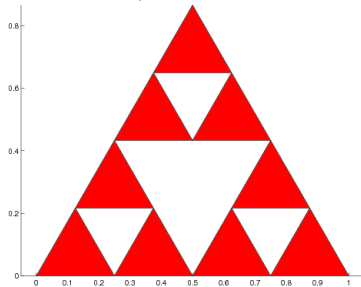
Sierpinski Dreieck, Level =0



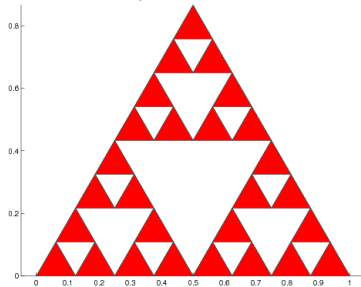
Sierpinski Dreieck, Level =1



Sierpinski Dreieck, Level =2



Sierpinski Dreieck, Level =3



# Matlab: Implementierung

```
%  sierpinski_plot.m
level=10;

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);
```

# Matlab: 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);
    %pause;
    sierpinski(ecke12,ecke2,ecke23,level-1);
    %pause;
    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'`.

# Python: Implementierung

```
level = 1
ecke1 = array([0,0])
ecke2 = array([1,0])
ecke3 = array([0.5, sqrt(3)/2])
figure
sierpinski (ecke1 ,ecke2 ,ecke3 , level)
title ('Sierpinski Dreieck , Level ={}'.format(level))
show()
```

# Python: Implementierung

```
def sierpinski(ecke1,ecke2,ecke3,level):  
    """Teilt das Dreieck auf in 3 Dreiecke (level >0)  
    Plotten des Dreiecks ( level =0)"""  
    if level == 0:  
        fill ([ecke1[0],ecke2[0],ecke3[0]],[ecke1[1],  
            ecke2[1],ecke3[1]], '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)
```

## 1 Rekursionen

## 2 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

## 1 Rekursionen

## 2 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation



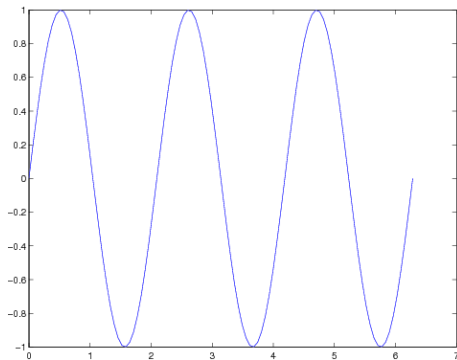
# Standard-Plot

```
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)
```



**Hinweis:** in Python muss häufig `show()` ausgeführt werden um die Grafiken darzustellen.

# Erweiterungen

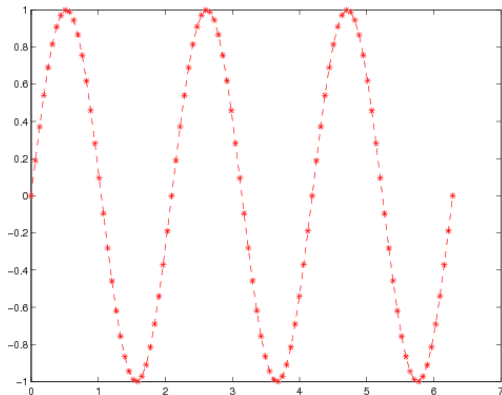
```
plot(<x>,<y>,<string>)
```

**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.



**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), : (ge-  
punktete Linie), -. (Strich-Punkt Linie)

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

# Matlab: Optionen II

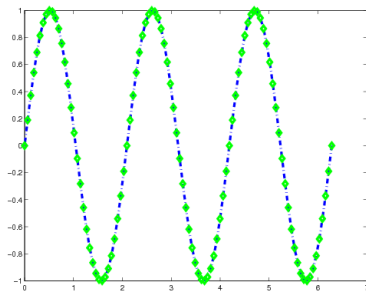
```
plot(<x>,<y>,<string>,<Eigenschaft>,<Spez.>)
```

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,...)
```

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

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

bzw.

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

- Logarithmische Skalierung beider Achsen:

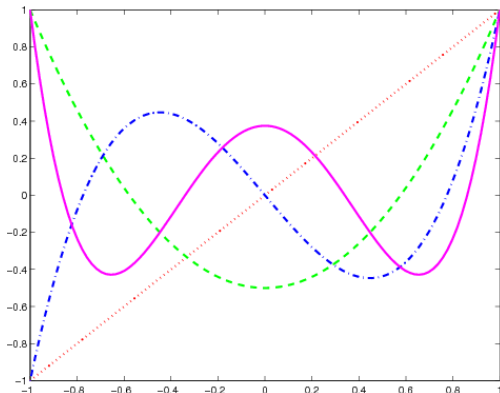
```
loglog(x1,y1)
```

- 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])  
axis auto| axis('auto')  
axis equal| axis('equal')
```

Setzen der x- und y-Achsen Grenzen  
Rückkehr zu Default Achsen Grenzen  
Gleiche Dateneinheiten auf allen Achsen

```
axis off| axis('off')  
axis square|  
axis('square')  
axis tight| axis('tight')
```

Entfernen der Achsen  
quadratische Achsen-Box

```
xlim([x1 x2])  
ylim([y1 y2])  
grid on| grid('on')  
box on, box off | box('on')  
box('off')
```

Achsen Grenzen werden passend zu den Daten gewählt.  
Setzen der x-Achse  
Setzen der y-Achse  
Gitter aktivieren  
Box um die Grafik legen, Box entfernen

## 1 Rekursionen

## 2 Einführung Grafik

- einfache zweidimensionale Grafiken
- **Beschriftungen**
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation



# Beschriften der Grafik

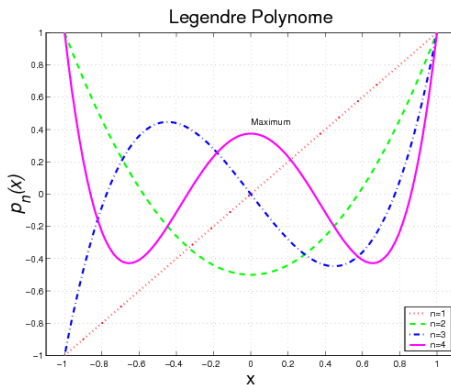
- Titel:  
`title ( ' Titel ' )`
- Achsenbeschriftung:  
`xlabel ( ' Text ' )`, `ylabel ( ' Text ' )`
- Legende(Matlab):  
`legend ( ' Text1 ' , ' Text2 ' , ..., nr )`  
`nr` gibt die Position der Legendengbox in der Grafik an: -1 (rechts vom Plot), 0 'bester' Ort, 1 oben rechts (default), 2 oben links, 3 unten links, 4 unten rechts.
- Legende(Python):  
`legend ( ( ' Text1 ' , ' Text2 ' , ... ) , loc = '<locationstr>' )`
- 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

- **Matlab:** In den strings kann direkt eine abgespeckte  $\text{\LaTeX}$ -Notation verwendet werden. (nahezu vollständige  $\text{\LaTeX}$ -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')`  $\Rightarrow$   
 $f(x) = \frac{1}{x^2+a}$
- **Python:** der Mathemodus in den üblichen `$$` kann benutzt werden. Für die PDF-Ausgabe kann man vollständigen  $\text{\LaTeX}$ -Umfang nutzen.
- Ändern der Schriftgröße, z.B. `title ('Titel', 'FontSize', 20)|`  
`title ('Titel', fontsize=20).`
- Auflistung aller modifizierbaren Texteingenschaften:  
`doc text_props(Matlab)`

# Beispiel - Legendre Polynome II

```
title('Legendre Polynome','FontSize', 20)
xlabel('x','FontSize', 20); ylabel('p_n(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])
```



## 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)

```
fill(<x>,<y1>)+fill_between(<x>,<y2>,<y1>)
```

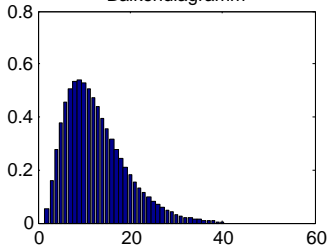
- Tortengrafik:

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

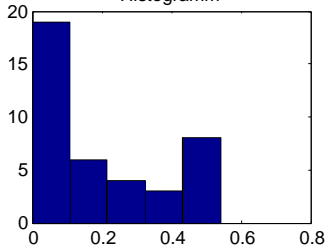
```
pie([<anteil1>,<anteil2> .. <anteilx>])
```

# Matlab: Darstellung von Daten

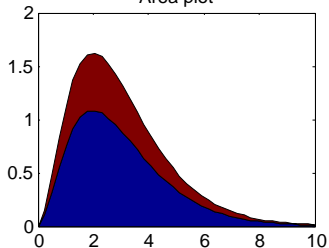
Balkendiagramm



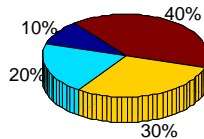
Histogramm



Area plot



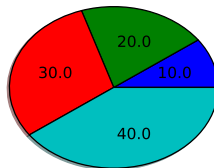
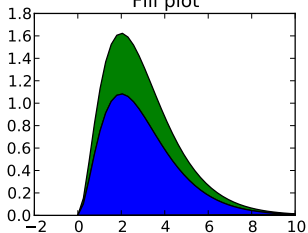
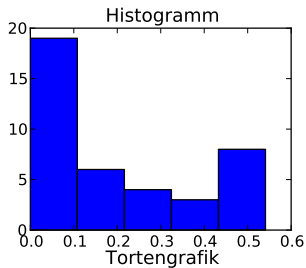
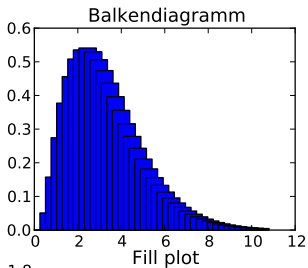
Tortengrafik



# Matlab: 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');
```

# Python: Darstellung von Daten





# Python: Darstellung von Daten

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

#Balkendiagramm
subplot(2,2,1)
bar(n,y),title('Balkendiagramm')

#Histogramm
subplot(2,2,2)
hist(y,5),title('Histogramm')

#Area plot
subplot(2,2,3)
fill(n,2*y),fill_between(n,3*y,2*y,facecolor='green')
title('Fill plot')

#Tortengrafik
subplot(2,2,4)
pie([ 1,2,3,4],autopct='%1.1f', shadow=True)
title('Tortengrafik')
```

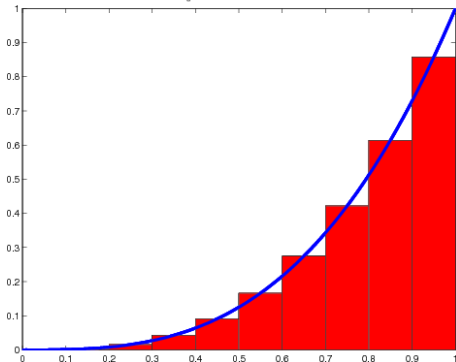
# Approximation von Integralen

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

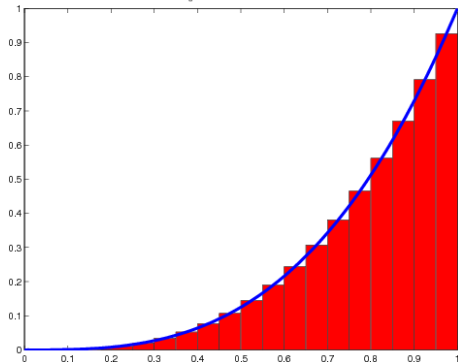
$$\int_0^1 f(x)dx \approx \sum_{i=1}^N \frac{1}{N} f\left(\frac{i - \frac{1}{2}}{N}\right)$$

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

$\int_0^1 x^3 = 0.24875$  fuer  $N=10$



$\int_0^1 x^3 = 0.24969$  fuer  $N=20$



# Matlab: Integral - Implementation

```
% integral.m
% berechnet approximativ ein Integral
% ueber (0,1) durch die Mittelpunkregel
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) ],'FontSize',20);
```

# Python: Integral - Implementation

```
# berechnet approximativ ein Integral
# ueber (0,1) durch die Mittelpunkregel
N = 10# Anzahl Unterteilungen
x = arange(0+1./(2*N),1-1./(2*N),1./N)
y = x**3
# Berechnung des Integrals
result = sum(y)*(1/N)

# Plot
for i in arange(1.,N+1):
    fill_between([(i-1)/N, i/N],[((i-0.5)/N)**3, ((i-0.5)/N)**3],facecolor='r')
plot(arange(0,1,0.01),array(arange(0,1,0.01))**3,
     linewidth=3)
title('\int_0^1 x^3 = {} fuer N = {}'.format(result,N),
     fontsize=20)
```

## 1 Rekursionen

## 2 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

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

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

$(X(i,j), Y(i,j), Z(i,j))$  dar.

# Python: Dreidimensionale Grafiken

3D Grafiken können auf sehr verschiedene Weise erzeugt werden:

- **Mayavi mlab** (gute 3D Beschleunigung, GUI möglich, Empfohlen)
- **mplot3d** (keine 3D Beschleunigung, am einfachsten zu Benutzen)
- **easyviz** aus den scitools (Frontend zu diversen Tools)

Allen Tools ist es gemein, dass sie versuchen Matlab-Syntax nahe zu kommen. Aufgrund der Vielfalt wird hier aber auf einer Auflistung der Befehle verzichtet.

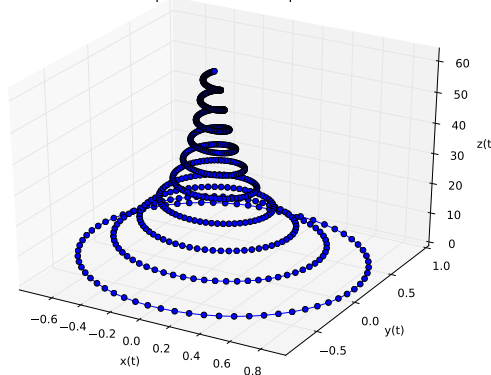
Wir benutzen **mplot3d**, es sei denn es etwas Anderes angegeben.

Import-Zeile: `from mpl_toolkits.mplot3d import Axes3D`

# 3D Kurven

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: 3D Kurve mit plot





# Beispiel 3D Kurven

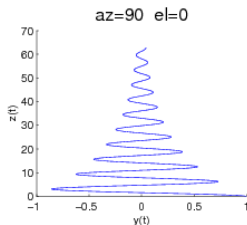
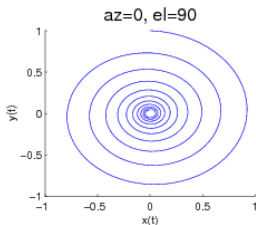
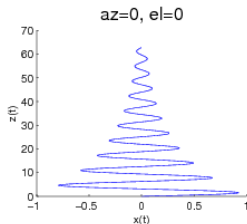
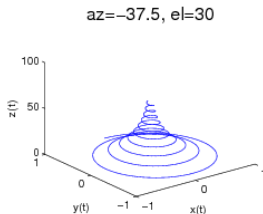
```
t = 0:0.1:20*pi;
x = exp(-t/20).*sin(t);
y = exp(-t/20).*cos(t);
z = t;
plot3(x,y,z,'b-o','LineWidth',1);
xlabel('x(t)'), ylabel('y(t)');
    zlabel('z(t)');
title('Beispiel: 3D Kurve mit plot3','FontSize',15);
```

```
t = arange(0,20*pi,0.1)
x = exp(-t/20)*sin(t)
y = exp(-t/20)*cos(t)
z = t
fig=figure()
ax = Axes3D(fig)
plot(x,y,z,'b-o',linewidth=1)
ax.set_xlabel('x(t)'),ax.set_ylabel('y(t)'),ax.set_zlabel
    ('z(t)')
title('Beispiel: 3D Kurve mit plot',fontsize=15)
```

# Blickwinkel

`view(az, el)`(Matlab) | `Axes3D.view_init(el, az)`(Python)

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



# 3D-Funktionenplots

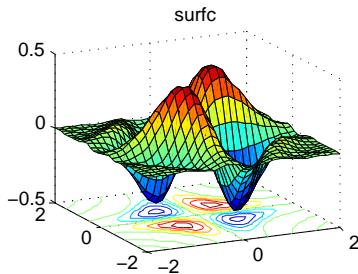
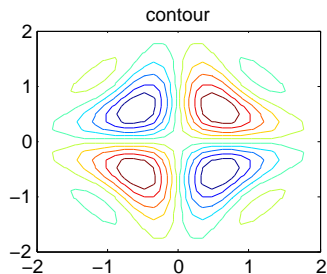
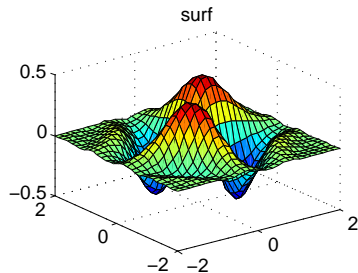
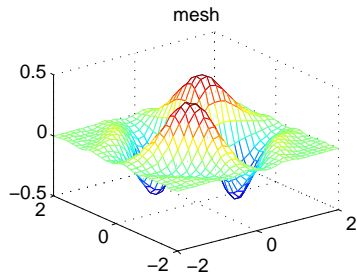
Darstellung von Funktionen

$$f: \mathbb{R}^2 \rightarrow \mathbb{R}$$

**Beispiel:**

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

# Matlab: Funktionenplot

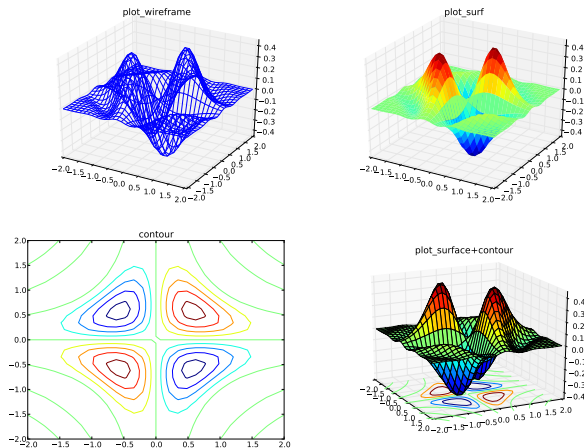


# Matlab: Funktionenplot - Implementation

```
% 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');
```

# Python: Funktionenplot



# Python: Funktionenplot - Implementation

```
x = linspace(-2,2,30)
y = linspace(-2,2,30)
[X,Y] = meshgrid(x,y)
Z = exp(-X**2-Y**2)*sin(pi*X*Y)
fig=figure()
ax = fig.add_subplot(2, 2, 1, projection='3d')
ax.plot_wireframe(X,Y,Z),title('plot_wireframe')

ax = fig.add_subplot(2, 2, 2, projection='3d')
ax.plot_surface(X,Y,Z,rstride=1,cstride=1,cmap=cm.jet,
               linewidth=0),title('plot_surface')

subplot(2, 2, 3)
contour(X,Y,Z,10), title('contour')

ax = fig.add_subplot(2, 2, 4, projection='3d')
ax.plot_surface(X,Y,Z,rstride=1,cstride=1,cmap=cm.jet)
ax.contour(X, Y, Z, zdir='z', offset=-0.5)
ax.view_init(20,-26),title('plot_surface + contour')
```

```
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  $(1, 1), (1, 2), \dots$

**Python-Zusatz (mplot3d):** Für die mplot3d-Achsen braucht man folgenden subplot:

```
fig.add_subplot(<n>, <m>, <k>, projection='3d')
```



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

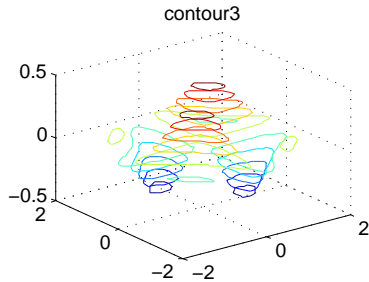
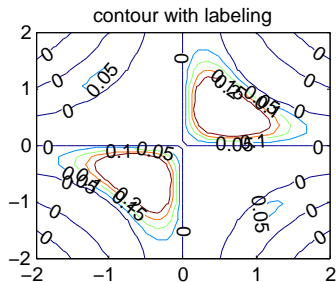
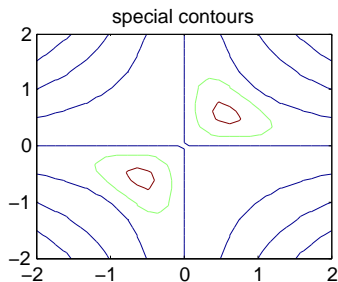
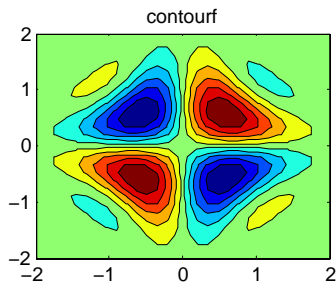
```
[X,Y]=meshgrid(x,y)
```

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

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

# Matlab: Contour Plots



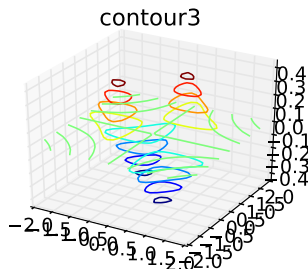
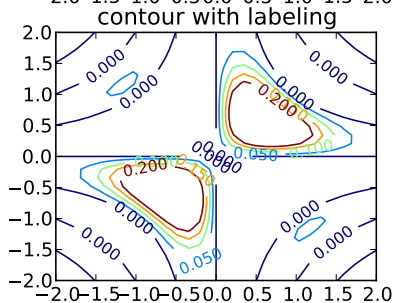
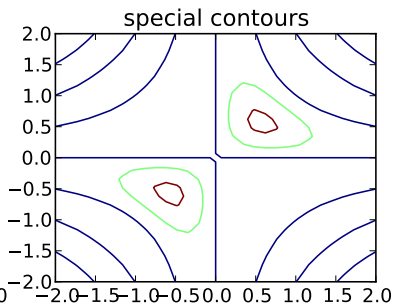
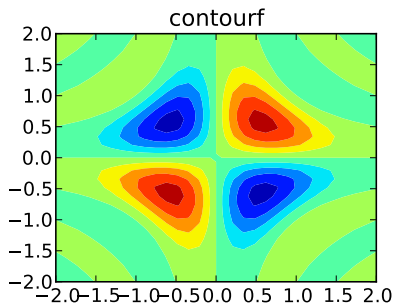
# Matlab: Contour Plots - Listing

```
% 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')
```

- `[C,h] = contour(X,Y,Z,n)` | `C = 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. ( $C$  sind die Konturlinien und  $h$  ist das Grafik-handle).
- `contourf(X,Y,Z,n)`  
funktioniert wie `contour` nur das die Flächen zwischen den Konturlinien ausgefüllt werden.
- `clabel(C,h)` (Matlab) | `clabel(C, inline=1)` (Python)  
beschriftet die Konturlinien, deren Werte in  $C$  gespeichert sind und die zum Grafik-Handle  $h$  gehören.
- `contour3(X,Y,Z,n)` | `Axes3D.contour(X,Y,Z,n)` (Python)  
zeichnet jede Konturlinie auf einer anderen Höhe.

# Python: Contour Plots



# Python: 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
fig=figure()
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)
CS = contour(X,Y,Z,[0,0.05,0.1,0.15,0.2])
plt.clabel(CS, inline=1, fontsize=10)
title('contour with labeling')
ax = fig.add_subplot(2, 2, 4, projection='3d')
ax.contour(X,Y,Z,10),title('contour3')
```

# Matlab: Slice

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

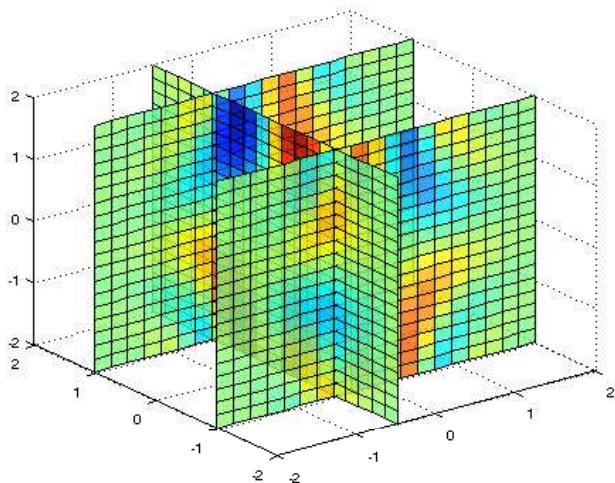
zeichnet Schnitte zu den Funktionswerten  $V(i)$  zu  $(X(i), Y(i), Z(i))$ .  
Schnitte sind durch die Vektoren  $sx$ ,  $sy$  und  $sz$  gegeben.

**Beispiel:**

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

```
x = linspace(-2, 2, 180);  
[X,Y,Z] = meshgrid(x, x, x);  
V = exp(-X.^2-Y.^2) .* sin(pi*X.*Y.*Z);  
sx = [-0.5, 0]'; sy = [-1, 1]';  
sz = [0, 0, 1]';  
%slice(X,Y,Z,V,sx,sy,sz);  
alpha(0.8) % Transparency  
shading interp  
isosurface(X,Y,Z,V)
```

# Matlab: Beispiel-slice





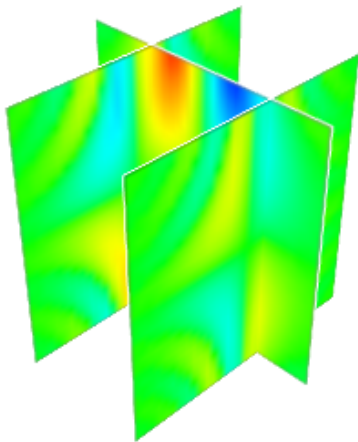
# Python: slice (Mayavi mlab)

```
ml.pipeline.image_plane_widget(ml.pipeline.scalar_field(V
    ), plane_orientation='x_axes' | 'y_axes' | 'z_axes',
    slice_index=<idx>)
```

- V: Funktionswerte  $V(i)$  zu  $(X(i), Y(i), Z(i))$ .
- plane\_orientation: Schnitte durch x-/y-/z-Achse
- slice\_index: Index in den Matrizen (keine direkte Koordinaten-Angabe)

```
X, Y, Z = np.ogrid[-2:2:20j, -2:2:20j, -2:2:20j]
V = exp(-X**2-Y**2) * sin(pi*X*Y*Z)
ml.pipeline.image_plane_widget(ml.pipeline.scalar_field(V
    ), plane_orientation='x_axes', slice_index=8)
ml.pipeline.image_plane_widget(ml.pipeline.scalar_field(V
    ), plane_orientation='y_axes', slice_index=5)
ml.pipeline.image_plane_widget(ml.pipeline.scalar_field(V
    ), plane_orientation='y_axes', slice_index=15)
```

# Python: Beispiel-slice



## 1 Rekursionen

## 2 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

# Matlab: Animation-Beispiel

```
% animation.m

clear all;
[X,Y] = meshgrid(-1:0.01:1,-1:0.01:1);

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);
```

# Matlab: Erstellen einer Animation/Videos

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

# Python: Animation-Beispiel

```
[X,Y] = meshgrid(arange(-1,1,0.05),arange(-1,1,0.05))
fig = figure()
ax = Axes3D(fig)

for j in arange(1,50):
    Z = cos(j**0.5*pi*exp(-X**2-Y**2))
    ax.plot_surface(X,Y,Z,rstride=1,cstride=1,cmap=cm.jet,
                    ,linewidth=0)
    ax.grid(b=False)
    fname = '/scratch/jschulz1/_tmp{:05d}.png'.format(j)
    fig.savefig(fname)
    ax.cla()

print ("creating movie")
CreateMovie('/scratch/jschulz1/', 'course', 10)
```