

# Kapitel 9

## Graphische Ausgabe

### 9.1 Grundlagen

### 9.2 Beispiele

#### 9.2.1 Zweidimensionale Plots

Es gibt eine Reihe von Befehlen zur Darstellung zweidimensionaler Graphiken.

Tabelle 9.1: MATLAB Befehle zum Erzeugen einfacher zweidimensionaler Graphiken

<code>fplot('fun', [x<sub>min</sub>, x<sub>max</sub>])</code>	9.2.1.1	Zeichnet 'fun' im Bereich von $x_{min}$ bis $x_{max}$
<code>plot(x,y)</code>	9.2.1.2	Zeichnet y als Funktion von x
<code>ezplot('fun', [x<sub>min</sub>, x<sub>max</sub>])</code>	9.2.1.3	erstellt u.a. implizite Funktionen, automatische Achsenbeschriftung
<code>comet(x,y,p)</code>	9.2.1.4	Zeichnet 2D Funktion in Form eines animierten 'Kometen'
<code>semilogx(x,y)</code>	9.2.1.5	Zeichnet 2D Funktion mit (10er-) logarithmischer x-Achse
<code>semilogy(x,y)</code>	9.2.1.6	Zeichnet 2D Funktion mit (10er-) logarithmischer y-Achse
<code>loglog(x,y)</code>	9.2.1.7	Zeichnet 2D Funktion mit (10er-) logarithmischer x- und y-Achse
<code>plotyy(x<sub>1</sub>, y<sub>1</sub>, x<sub>2</sub>, y<sub>2</sub>, 'f<sub>1</sub>', 'f<sub>2</sub>')</code>	9.2.1.8	Erstellt 2 Graphen mit den Plotbefehlen $f_1$ und $f_2$ mit getrennten y-Achsen
<code>polar(phi,r)</code>	9.2.1.9	Zeichnet die Funktion r(phi) in Polarkoordinaten.

### 9.2.1.1 Fplot

Einfachste Möglichkeit, eine Funktion (in String - Schreibweise) innerhalb eines Intervalls zu plotten.

---

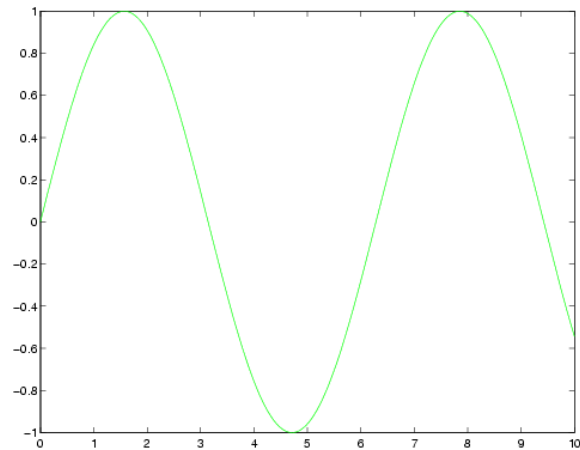
`fplot`

`graph_fplot.m`

---

Plot einer grünen Sinuskurve im Bereich von  $x = 0$  bis 10

```
fplot('sin',[0,10],'g')
```



---

Als weitere Farbkürzel neben 'g' (grün) sind 'k' (schwarz), 'm' (violett), 'r' (rot), 'c' (türkis), 'b' (blau), 'w' (weiß) und 'y' (gelb) erlaubt, siehe auch [linespec](#).

### 9.2.1.2 Plot

Einfacher 2D Plot, zeichnet die Funktion  $y = f(x)$  bei Vorgabe des Vektors  $x$

---

`plot`

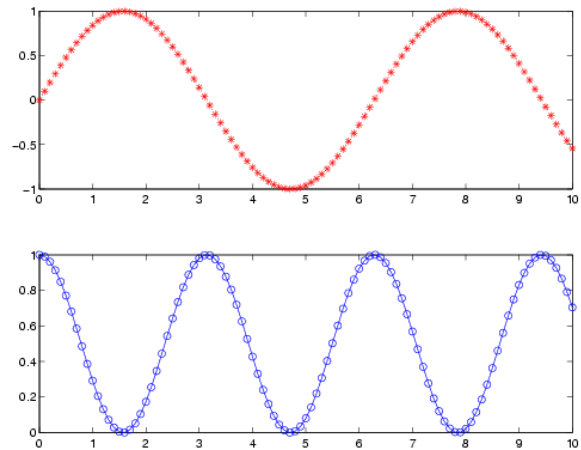
`graph_plot.m`

---

Mit Hilfe von `subplot` werden 2 Achsen geschaffen, die Zeichen zwischen den ' ' in `plot` symbolisieren Farbe, 'Marker Style' und 'Line Style'.

```
x=0:0.1:10;  
y1=sin(x);  
y2=cos(x).^2;
```

```
figure  
subplot(2,1,1)  
plot(x,y1,'r*:')  
  
subplot(2,1,2)  
plot(x,y2,'bo-')
```



---

Eine vollständige Auflistung der verfügbaren Symbole der erwähnten 'Styles' finden sich in der Hilfe von `linespec`

### 9.2.1.3 Ezplot

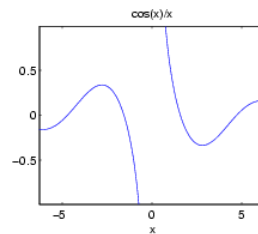
Erstellt 2 dimensionale, unter anderem auch implizite Funktionen mit automatischer Achsenbeschriftung und wenn erwünscht, mit automatischen Intervallgrenzen.

`ezplot`

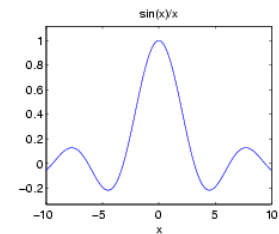
`graph_ezplot.m`

Der Befehl `axis square` stellt jede Achse mit derselben Länge dar und verhindert, dass Kreise als Ellipsen wirken.

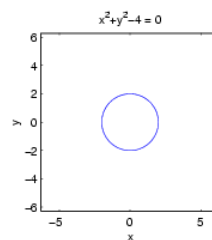
```
subplot(2,2,1)
ezplot('cos(x)/x')
```



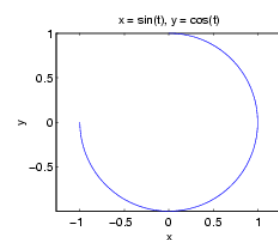
```
subplot(2,2,2)
ezplot('sin(x)/x', [-10,10])
```



```
subplot(2,2,3)
ezplot('x^2+y^2-4')
axis square
```



```
subplot(2,2,4)
ezplot('sin', 'cos', [0,1.5*pi])
```



### 9.2.1.4 Comet

Erstellt eine 2 dimensionale Funktion in Form eines sich bewegenden 'Kometen', dessen Schweif bzw. Spur den Graphen darstellt.

---

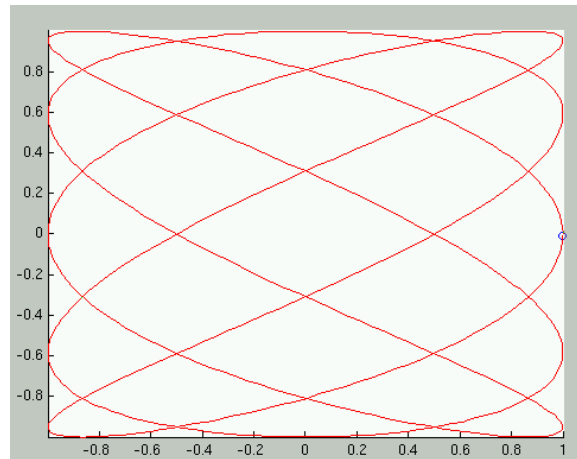
`comet`

`graph_comet.m`

---

Der letzte Parameter in `comet` gibt die Schweiflänge relativ zur Gesamtlänge des Graphen an.

```
t=0:0.01:2*pi;  
x=cos(5*t);  
y=sin(3*t);  
  
comet(x,y,0.2)
```



---

Achtung, die Erstellung des Graphen erfolgt im `erasemode none`, wird das Graphikfenster vergrößert, verschwindet der Graph, er kann daher auch nicht gedruckt werden.

### 9.2.1.5 Semilogx

Erstellt eine 2 dimensionale Funktion mit logarithmischer x - Achse.

`semilogx`

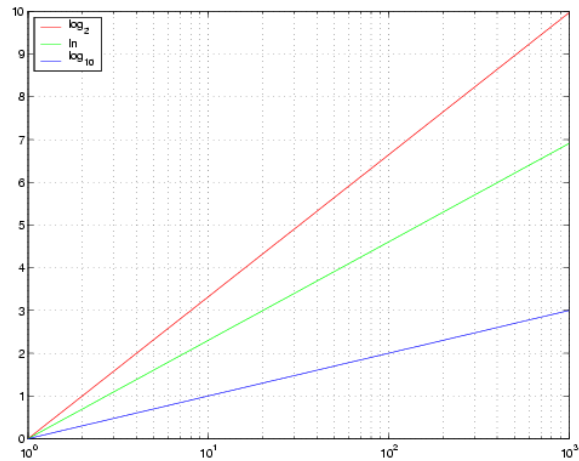
`graph_semilogx.m`

Der Befehl `legend` fügt dem Plot an einer wählbaren Position eine Legende der Lines hinzu, `grid` fügt der Graphik Gitterlinien hinzu.

```
x=logspace(0,3,30);  
y1=log2(x);  
y2=log(x);  
y3=log10(x);
```

```
semilogx(x,y1,'r',...  
         x,y2,'g',...  
         x,y3,'b')
```

```
grid on  
legend('log_2','ln','log_{10}',2)
```



Sollen mehrere Lines in eine Achse gezeichnet werden, so können die Koordinaten und Style Eigenschaften der Lines hintereinandergefügt werden.

### 9.2.1.6 Semilogy

Erstellt eine 2 dimensionale Funktion mit logarithmischer y - Achse.

`semilogy`

`graph_semilogy.m`

Die Befehle `xlabel` und `ylabel` ermöglichen die Beschriftung der x - und der y - Achse.

```
x=0:0.5:10;
```

```
y1=2.^x;
```

```
y2=exp(x);
```

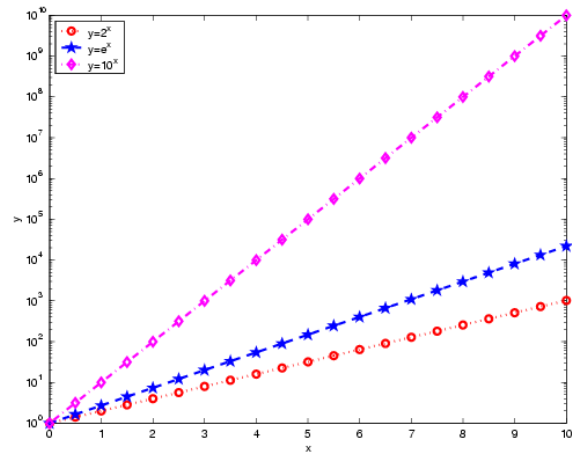
```
y3=10.^x;
```

```
semilogy(x,y1,'r:o',...  
          x,y2,'b--p',...  
          x,y3,'m-.d',...  
          'linewidth',2)
```

```
xlabel('x')
```

```
ylabel('y')
```

```
legend('y=2^x','y=e^x','y=10^x',2)
```



Die Dicke der Linien lässt sich mit der Line - Eigenschaft `linewidth` verändern, im Beispiel beträgt sie 2 Punkte.



### 9.2.1.7 Loglog

Erstellt eine 2 dimensionale Funktion mit logarithmischer x - und y - Achse.

`loglog`

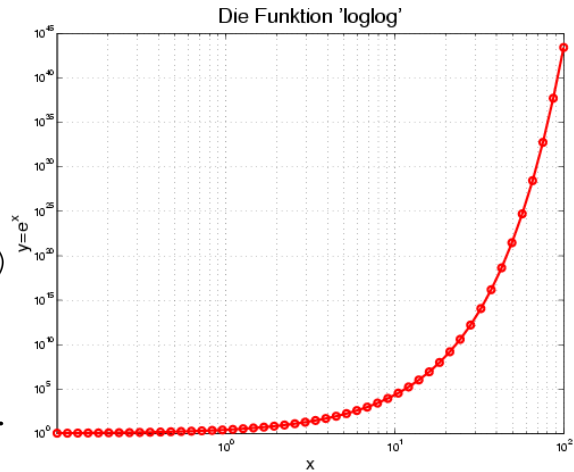
`graph_loglog.m`

Um die Achse mit einer Überschrift zu versehen, kann der Befehl `titel` verwendet werden.

```
x=logspace(-1,2);  
y=exp(x);
```

```
loglog(x,y,'ro-','linewidth',2)
```

```
xlabel('x','fontsize',16)  
ylabel('y=e^x','fontsize',16)  
title('Funktion ''loglog''',...  
      'fontsize',18)
```



Die Größe der Schrift wird mit `fontsize` gesteuert, dies ist jedoch nur eine von vielen Texteneigenschaften. Werden in einem String `''` - Symbole verwendet, so muss man, wie im Beispiel der Überschrift, zwei statt nur eines der `''` Symbole verwenden.

### 9.2.1.8 Plotyy

Erstellt zwei durch  $x_1$  und  $y_1$  bzw.  $x_2$  und  $y_2$  definierte Graphen mit eigenen y-Achsen. Es ist erlaubt, beide Funktionen mit unterschiedlichen Plot-Befehlen darzustellen.

`plotyy`

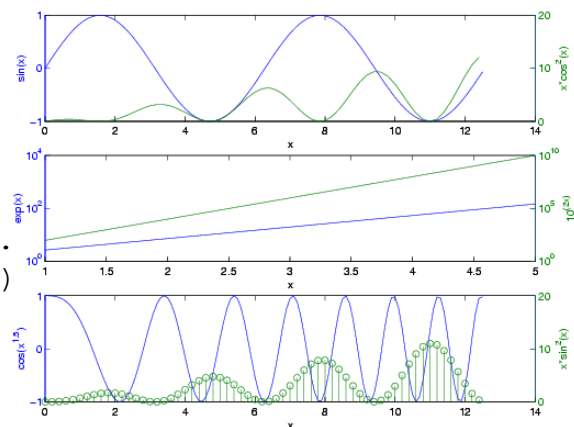
`graph_plotyy.m`

Die linke y-Achse gehört zur ersten, die rechte hingegen zur zweiten Funktion. Stellvertretend für die 3 Subplots sei hier nur der 3. angeführt.

```
subplot(3,1,3)
x1=0:0.1:4*pi;
y1=cos((x1.^1.5));
x2=0:.2:4*pi;
y2=x2.*sin(x2).^2;

[AX,H1,H2]=plotyy(x1,y1,x2,y2, .
    'plot','stem')

set(get(AX(1),'xlabel'),...
    'String','x')
set(get(AX(2),'xlabel'),...
    'String','x')
set(get(AX(1),'ylabel'),...
    'String','cos(x^{1.5})')
set(get(AX(2),'ylabel'),...
    'String','x*sin^2(x)')
```



In diesem Beispiel tritt erstmals das sehr wichtige 'Graphik-Handle' Konzept auf. Ein Graphik-Handle ist ein Code, der die gesamte Information von Achsen, Figures und anderen Graphik-Objekten beinhaltet. Mit dem Befehl `get` können alle Eigenschaften des Objekts abgefragt und mit `set` gesetzt werden. In diesem Beispiel etwa werden die 'String' Eigenschaften von x- und ylabel gesetzt. AX beinhaltet die Handles beider Achsen, H1 und H2 sind die Handles der beiden 'Line' Objekte. So bekommt man beispielsweise mit `get(H1)` die gesamte Information über den blau gezeichneten Graphen, mit `set(H1,'linewidth',4)` verändert man die Liniendicke auf 4 Punkte.

Für die Darstellungsarten der Funktionen sind folgende Varianten erlaubt: `plot`, `semilogx`, `semilogy`, `loglog` sowie `stem`.

Tabelle 9.2: MATLAB Befehle zum Erzeugen von Balken- und Kreisdiagrammen

<code>hist(y,x)</code>	9.2.1.10	Erstellt ein Histogramm der Werte in y über jenen von x
<code>bar(x,y,'width','style')</code>	9.2.1.11	Stellt die Datenpaare [x,y] als vertikale Balken dar
<code>barh(x,y,'width','style')</code>	9.2.1.12	Stellt die Datenpaare [x,y] als horizontale Balken dar
<code>pie(x,'explode')</code>	9.2.1.13	Zeichnet ein 2D Kreisdiagramm der Daten von x

### 9.2.1.9 Polardiagramm

Zeichnet die Funktion  $r=f(\text{phi})$  im Polardiagramm.

`polar`

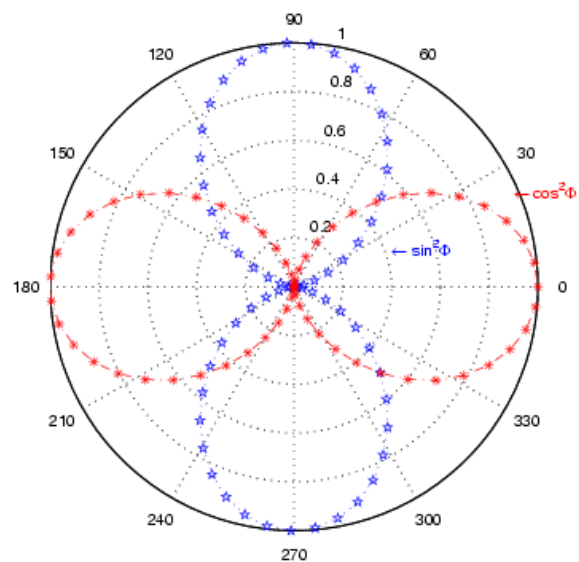
`graph_polar.m`

Text in der Spalte

```
phi=0:0.1:2*pi;
r1=sin(phi).^2;
r2=cos(phi).^2;

polar(phi,r1,'b:p')
hold on
polar(phi,r2,'r-.*')

text(phi(5),r1(5),...
      '\leftarrow sin^2\Phi',...
      'color','blue')
text(phi(10),r2(10),...
      '\leftarrow cos^2\Phi',...
      'color','red')
hold off
```



Nach dem Befehl `hold on` werden alle weiteren Graphiken in das aktuelle Achsensystem gezeichnet, ohne die vorigen Graphiken zu löschen, erst mit `hold off` werden alten Graphiken durch neue ersetzt.

`text(x,y,'string')` gestattet die Positionierung eines Texts 'string' bei den Koordinaten (x,y) im Achsensystem.

### 9.2.1.10 Histogramm

Die Daten von `y` werden in Form von Histogrammen dargestellt.

`hist`

`graph_hist.m`

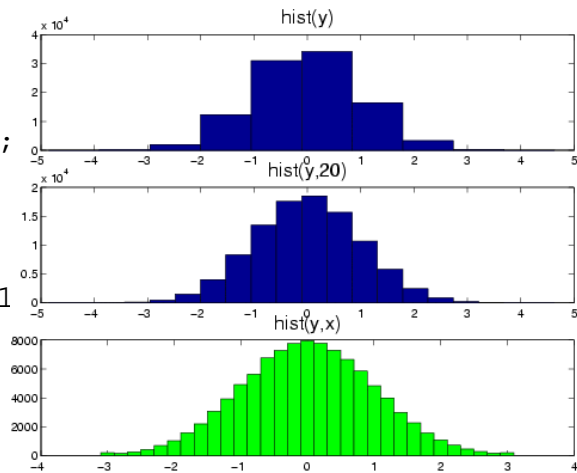
Die unterschiedlichen Aufrufe des Histogramm - Befehls anhand eines Beispiels normalverteilter Daten:

```
y=randn(1,100000);
subplot(3,1,1)
hist(y)
title('hist(y)', 'fontsize', 16);

subplot(3,1,2)
hist(y, 20)
title('hist(y, 20)', 'fontsize', 1

subplot(3,1,3)
x=-3:0.2:3;
hist(y,x)
title('hist(y,x)', 'fontsize', 16);

h = findobj(gca, 'Type', 'patch');
set(h, 'facecolor', 'g')
```



Die letzten beiden Zeilen färben die Balken des Histogramms grün ein, dabei wird mit `findobj` nach allen Graphik-Objekten der mit `gca` abgefragten aktuellen Achsen gesucht, die vom Typ `patch` sind. Der resultierende Handle wird von `set` zum Verändern der Patch-Eigenschaft herangezogen.

### 9.2.1.11 Bar

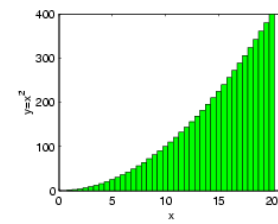
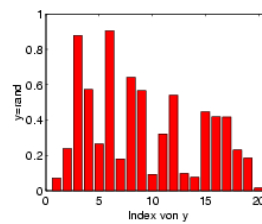
Erstellt an den Positionen von  $x$  vertikale Balken der Höhe  $y$  mit der relativen Balkenbreite 'width'. Die Balkengruppierung wird mit der Option 'style' gesteuert.

`bar`

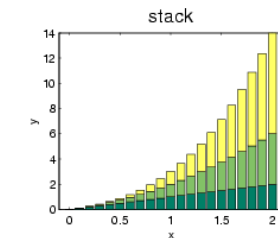
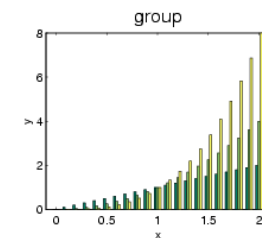
`graph_bar.m`

$y$  kann sowohl ein Vektor, als auch eine  $n * m$  Matrix sein, wobei  $n=length(x)$  und  $m$  die Anzahl der dargestellten Datensätze entspricht.

```
subplot(2,2,1)
y=rand(20,1);
bar(y,'r')
```



```
subplot(2,2,2)
x=1:0.5:20;
y=x.^2;
bar(x,y,1,'g')
```



```
subplot(2,2,3)
x=[0:0.1:2]';
y=[x,x.^2,x.^3];
colormap summer
bar(x,y,1,'group')
```

```
subplot(2,2,4)
bar(x,y,'stack')
```

Der Style 'grouped' positioniert die Balken der  $m$  Datensätze nebeneinander, mit 'stack' werden sie übereinander angeordnet. Mit `colormap` lassen sich sowohl vordefinierte, als auch selbst entworfene Farbskalen für die Darstellung der Graphiken verwenden.

### 9.2.1.12 Barh

Die Datenpaare  $(x,y)$  werden in Form von horizontalen Balken des Stiles 'style' mit der relativen Breite 'width' veranschaulicht.

`barh`

`graph_barh.m`

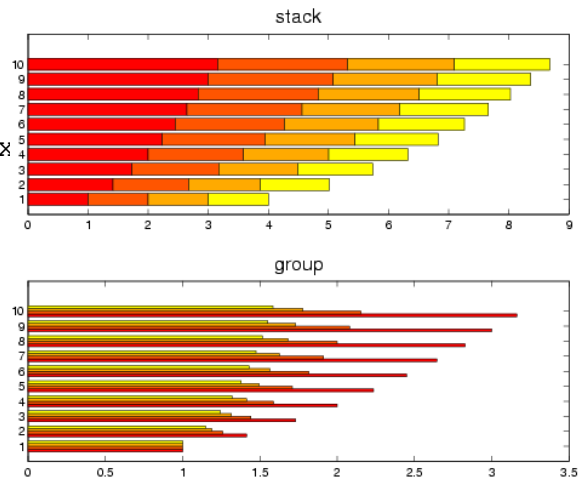
Wie im Beispiel 9.2.1.11 kann  $y$  eine Matrix sein.

```
x=(1:1:10)';  
y=[x.^(1/2),x.^(1/3),x.^(1/4),x
```

```
subplot(2,1,1)  
barh(x,y,'stack')
```

```
subplot(2,1,2)  
barh(x,y,1,'group')
```

```
colormap autumn  
set(gcf,'color','w')
```



Für die Darstellungsmöglichkeiten gruppierter Daten kann man zwischen 'grouped' und 'stack' wählen.

Der Befehl `gcf` ermittelt den Handle der aktuellen Figure, im Beispiel wird er benutzt, um die Farbe des Fensters auf weiß zu setzen.

### 9.2.1.13 Pie

Erstellt aus den Daten von `x` ein 2D Kreisdiagramm.

`pie`

`graph_pie.m`

Wird der aus 0 und 1 bestehende Vektor 'explode' angegeben, so werden jene Segmente hervorgehoben, die in `explode` (muß dieselbe Länge wie `x` haben) den Wert 1 aufweisen.

```
einwohner=[278,562.7,1545.3,...  
          1380.5,518.6,1202.3,  
          672.2,350.3,1611.4];  
explode=[0,1,0,0,0,1,0,0,0];
```

```
pie(einwohner,explode)
```

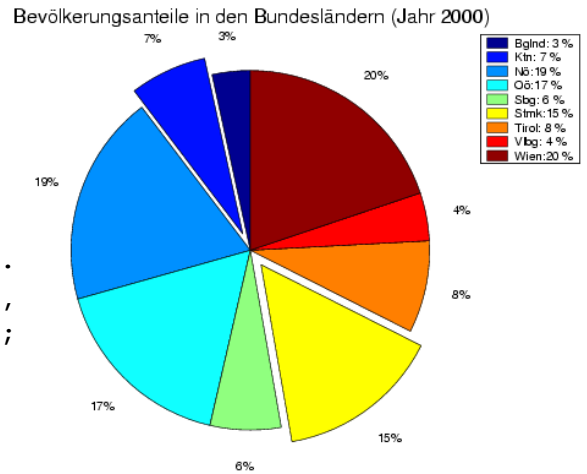


Tabelle 9.3: MATLAB Befehle zum Erzeugen von speziellen zweidimensionalen Graphiken

<code>stem(x,y)</code>	9.2.1.14	Zeichnet $y=f(x)$ und verbindet Punkte mit x-Achse
<code>stairs(x,y)</code>	9.2.1.15	Erstellt Funktion $y=f(x)$ in Form eines Stufendiagramms
<code>errorbar(x,y,e)</code>	9.2.1.16	Zeichnet $y$ als Funktion von $x$ samt Fehlerbalken der Länge $e$
<code>compass(x,y)</code>	9.2.1.17	Zeichnet $y=f(x)$ und verbindet die Punkte durch Vektorpfeile mit dem Ursprung
<code>feather(u,v)</code>	9.2.1.18	Zeichnet die relativen Koordinaten $u$ und $v$ und verbindet die Punkte mit den jeweiligen Koordinatenursprüngen entlang der Abszisse
<code>scatter(x,y,r,c)</code>	9.2.1.19	Zeichnet Punkte an den Stellen $(x,y)$ der Größe $r$ sowie der Farbe $c$
<code>pcolor(x,y,c)</code>	9.2.1.20	Erstellt einen 'Pseudocolorplot' der Elemente $c$ an den von den Punkten $(x,y)$ definierten Positionen
<code>area(x,y)</code>	9.2.1.21	Füllt den Bereich zwischen $y=f(x)$ und der Abszisse mit einer Farbe
<code>fill(x,y,c)</code>	9.2.1.22	Malt die durch $(x,y)$ definierten Polygone mit der Farbe $c$ aus
<code>contour(x,y,z)</code>	9.2.1.23	Zeichnet durch $z=f(x,y)$ definierte Konturlinien
<code>contourf(x,y,z)</code>	9.2.1.24	Zeichnet durch $z=f(x,y)$ definierte Konturlinien und füllt die Flächen dazwischen aus
<code>quiver(x,y,u,v)</code>	9.2.1.25	Erstellt von den Punkten $(x,y)$ ausgehende Vektoren mit den Komponenten $(u,v)$
<code>plotmatrix(x,y)</code>	9.2.1.26	Streudiagramm, die Spalten von $x$ werden über jenen von $y$ aufgetragen



### 9.2.1.14 Stem

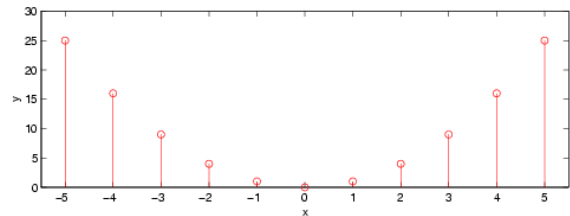
Zeichnet  $y$  als eine Funktion von  $x$  und verbindet zusätzlich die Punkte  $(x,y)$  durch senkrechte Linien mit der Abszisse.

`stem`

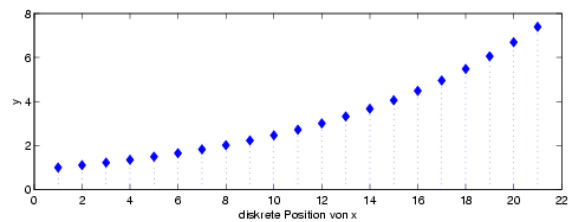
`graph_stem.m`

Mit der Option 'filled' werden die Datenpunkte ausgefüllt.

```
subplot(2,1,1)
x=-5:5;
y=x.^2;
stem(x,y,'r')
axis([-5.5,5.5,0,30])
```



```
subplot(2,1,2)
x=0:0.1:2;
stem(exp(x),'fill','b:d')
xlim([0,length(x)+1])
```



Im ersten Subplot werden die Achsengrenzen durch `axis([xmin,xmax, ymin, ymax])` geregelt, im zweiten Subplot mit dem Befehl `xlim`, wobei der Wertebereich der y-Achse unberührt bleibt.

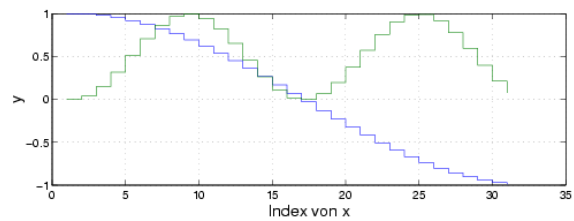
### 9.2.1.15 Stairs

Erstellt ein 2D Stufendiagramm von  $y$  als Funktion von  $x$

`stairs`

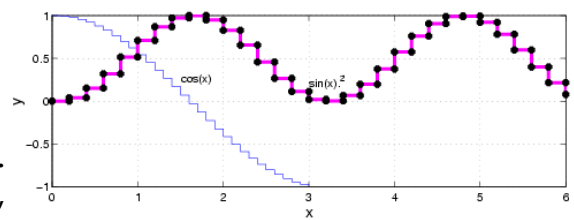
`graph_stairs.m`

```
subplot(2,1,1)
x1=[0:0.1:3]';x2=[0:0.2:6]';
y1=cos(x1);y2=sin(x2).^2;
y=[y1,y2];
stairs(y)
```



```
subplot(2,1,2)
x=[x1,x2];
handle=stairs(x,y);
```

```
set(handle(2),'linewidth',3,...
      'color','m','marker','* ',
      'markeredgecolor','k')
```



Mit Hilfe des Handle-Konzepts werden Liniendicke, Malfarbe, Datensymbole sowie die Umrandung dieser Datensymbole verändert.

### 9.2.1.16 Errorbar

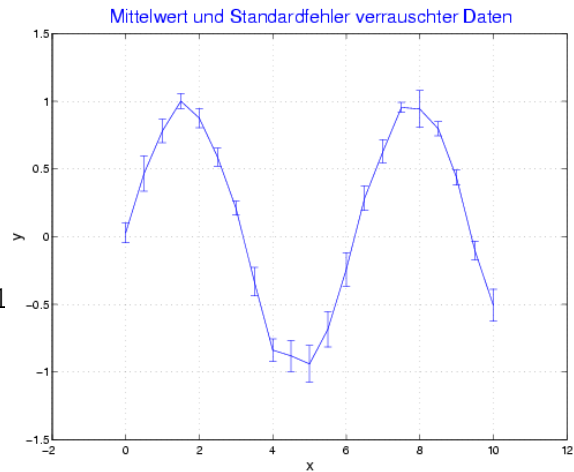
Zeichnet  $y$  als Funktion von  $x$  und fügt Fehlerbalken hinzu, die nach unten und oben durchaus unterschiedlicher Länge sein können.

`errorbar`

`graph_errorbar.m`

Mit `Errorbar` lassen sich elegant Mittelwerte und Standardabweichungen abbilden.

```
x=0:0.5:10;  
y= repmat(sin(x),[5,1]);  
zufalls_fehler=randn(size(y))/1  
y = y + zufalls_fehler;  
  
errorbar(x,mean(y),std(y));
```



### 9.2.1.17 Compass

Zeichnet  $y$  als Funktion von  $x$  und verbindet die Punkte mit dem Koordinatenursprung durch Vektorpfeile.

---

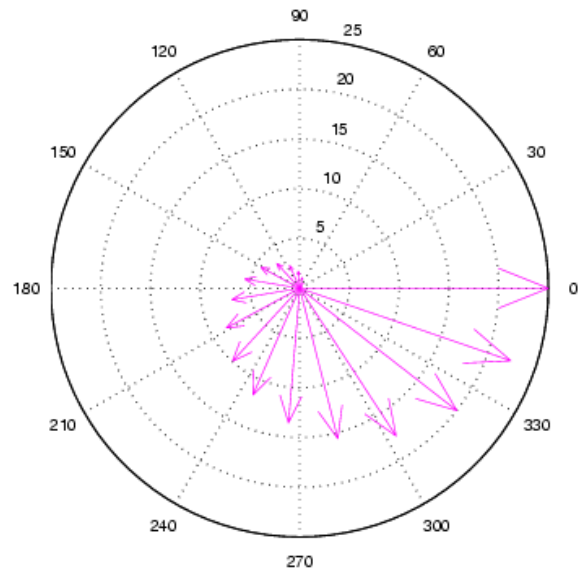
`compass`

`graph_compass.m`

---

Bei den Daten  $(x,y)$  handelt es sich um kartesische Koordinaten.

```
phi=linspace(0,2*pi,20);  
r=linspace(0,5,20);  
[x,y]=pol2cart(phi,r);  
  
compass(r.*x,r.*y,'m')
```



---

Mit `[x,y]=pol2cart(phi,r)` lassen sich die Polarkoordinaten  $(\phi,r)$  in die kartesischen Koordinaten  $(x,y)$  umwandeln.

### 9.2.1.18 Feather

Zeichnet die Punkte  $(u, v)$  relativ zu äquidistanten, auf der Abszisse liegenden Koordinatenursprüngen und verbindet sie mit Vektorpfeilen. Statt der reellen Werte  $(u, v)$  können auch komplexe Werte  $(z)$  verwendet werden, wobei auf der Abszisse die Real- und auf der Ordinate die Imaginärteile aufgetragen werden.

---

`feather`

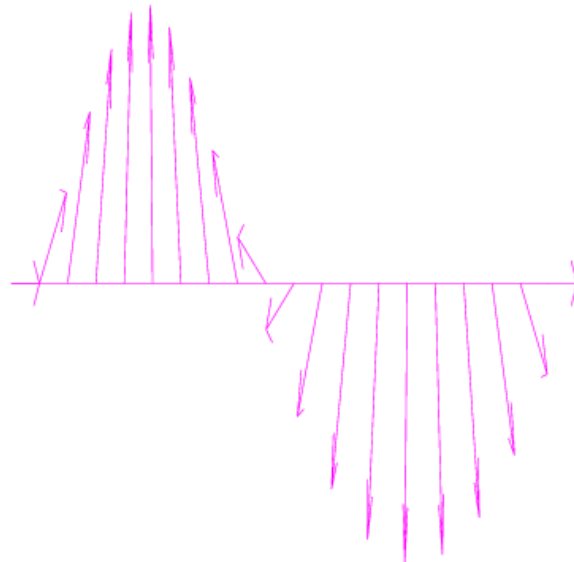
`graph_feather.m`

---

Normalerweise ist  $i$  auch in Matlab die imaginäre Einheit, das Symbol 'i' wird jedoch häufig als Laufindex verwendet und verliert dadurch den Wert  $\sqrt{-1}$ .

```
phi=linspace(0,2*pi,20);  
i=sqrt(-1);  
z=exp(i*phi);  
  
feather(z,'m')  
  
axis off
```

Die Funktion feather



---

Mit `axis off` werden die Achsenbeschriftungen sowie -ticks entfernt.

### 9.2.1.19 scatter

Zeichnet Daten durch Angabe der Positionen (x,y). Die Größe r sowie die Farbe c ist für alle Punkte getrennt einstellbar. Zusätzlich kann die Form der Datenpunkte ausgewählt und bei Bedarf durch die Option 'filled' gefüllt werden.

---

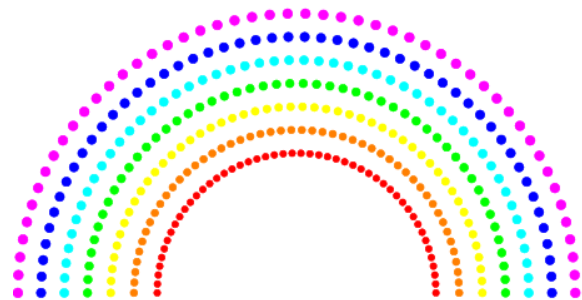
scatter

graph\_scatter.m

---

```
t=linspace(0,pi,50);
x= repmat(cos(t),[7,1]);
y= repmat(sin(t),[7,1]);
r=[6:12]';
r= repmat(r,[1,50]);
farbe=[1:7]';
farbe= repmat(farbe,[1,50]);

xx= reshape(r.*x,[],1);
yy= reshape(r.*y,[],1);
rr= 5* reshape(r,[],1);
farbe= reshape(farbe,[],1);
```



```
scatter(xx,yy,rr,farbe,'o','filled')
axis equal off
```

---

`axis equal` paßt das Achsensystem einem Quadrat an, sodass Kreise wirklich kreisförmig und nicht elliptisch aussehen.

### 9.2.1.20 Pseudocolor

Erstellt einen 'Pseudocolorplot' der Elemente  $c$  an den von den Punkten  $(x,y)$  definierten Positionen. Wird nur die Farben  $c$  angegeben, so werden die Farbe auf einer Matrix der Größe  $\text{size}(c)$  abgebildet.

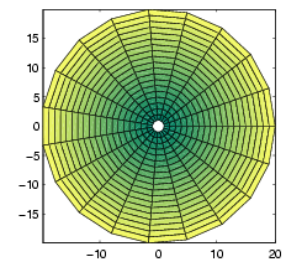
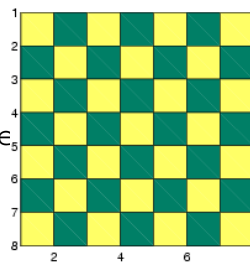
`pcolor`

`graph_pcolor.m`

Der Befehl `eye(2)` erzeugt eine  $2 \times 2$  Diagonalmatrix, mit `repmat` wird diese Diagonalmatrix zu einem Schachbrettmuster aneinanderkopiert.

```
x=eye(2);
X=repmat(x,[4,4]);
pcolor(X)
colormap summer; axis ij square;

t=linspace(0,2*pi,20);
x=cos(t); y=sin(t); r=[1:20]';
X=repmat(x,[20,1]);
Y=repmat(y,[20,1]);
R=repmat(r,[1,20]);
axis square; pcolor(R.*X,R.*Y,R)
```



`axis ij` wählt für das Achsensystem den Matrixmodus, wodurch die Indizierung in der linken oberen Ecke der dargestellten Matrix beginnt und jede Zelle die Länge 1 besitzt.

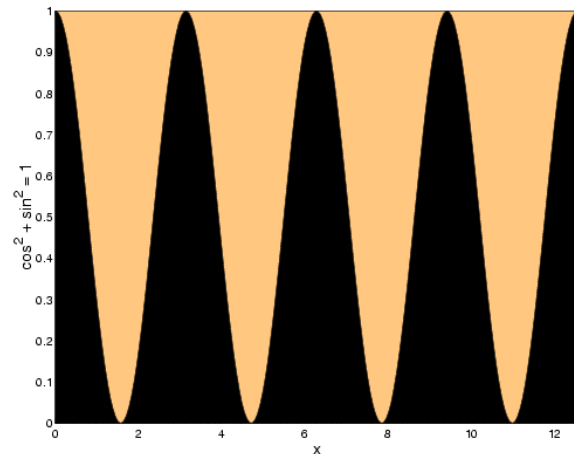
### 9.2.1.21 Area

Füllt den Bereich zwischen 2 Graphen (wenn y eine Matrix ist) bzw. zwischen einem Graphen und der Abszisse (wenn y ein Vektor ist) mit Farben aus.

`area`

`graph_area.m`

```
t=linspace(0,4*pi,200);  
y1=cos(t).^2;  
y2=sin(t).^2;  
y=[y1;y2]';  
  
area(t,y)  
  
axis tight  
colormap copper
```



`axis tight` wählt die Achsengrenzen derart, dass sie nur den Bereich der Graphik abdecken.



### 9.2.1.22 Fill

Malt die durch die Punkte  $(x,y)$  definierten Polygone mit der Farbe  $c$  aus.

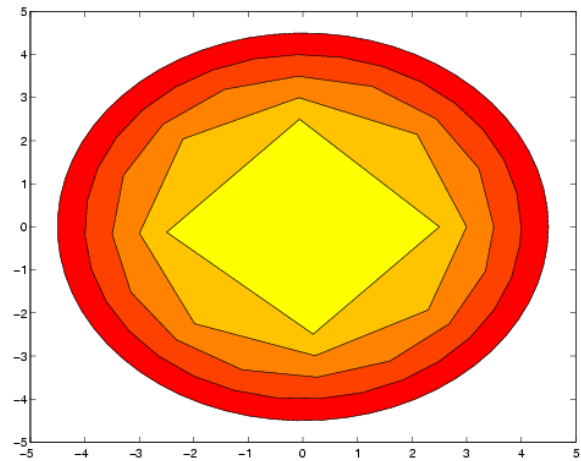
`fill`

`graph_fill.m`

Von dem Kreis (eigentlich 64-Eck) werden in einer Schleife jeder, jeder 2., 4., 8. und 16. Punkt herausgegriffen und durch Linien zu einem Polygon verbunden und mit der  $i$ . Farbe der aktuellen `colormap` ausgemalt.

```
t=linspace(0,2*pi,64);
x=cos(t);
y=sin(t);

for i=1:5
    r=5-i/2;
    index=2^(i-1);
    fill(r*x(1:index:end),...
        r*y(1:index:end),i)
    hold on
end
```



### 9.2.1.23 Contour

Zeichnet  $z$  als Funktion von  $x$  und  $y$  in Form von Konturlinien (Höhenlinien), die je nach Aufruf von `contour` äquidistant sind oder bei bestimmten Werten von  $z$  liegen.

`contour`

`graph_contour.m`

Der sehr wichtige und vorallem bei 3D Plots unabkömmliche Befehl `meshgrid` erzeugt eine Matrix für die  $x$ - sowie eine für die  $y$ - Komponente des Gitters, über dem  $z$  definiert ist

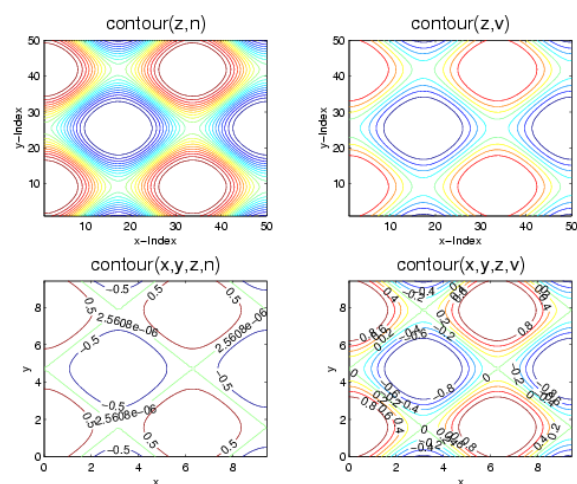
```
x=linspace(0,3*pi,50);  
y=linspace(0,3*pi,50);  
[xx,yy]=meshgrid(x,y);  
z=(sin(cos(xx)+sin(yy)));  
v=linspace(min(min(z)),...  
          max(max(z)),10);
```

```
subplot(2,2,1)  
contour(z,20)
```

```
subplot(2,2,2)  
contour(z,v)
```

```
subplot(2,2,3)  
[c,h]=contour(xx,yy,z,3);  
clabel(c,h)
```

```
subplot(2,2,4)  
v=[-1:0.2:1];  
[c,h]=contour(xx,yy,z,v);  
clabel(c,h)
```



Mit `clabel` werden die Konturlinien mit den entsprechenden  $z$ -Werten beschriftet.

### 9.2.1.24 Contourf

Ähnliche Wirkung wie `contour` in 9.2.1.23, allerdings werden die Flächen zwischen den Konturlinien ausgemalt.

`contourf`

`graph_contourf.m`

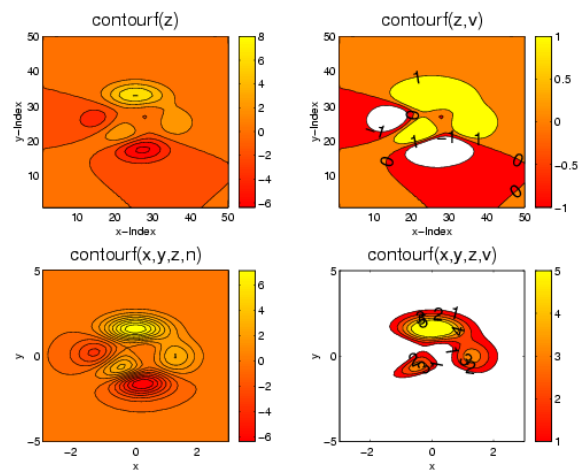
```
x=linspace(-3,3,50);
y=linspace(-5,5,50);
[xx,yy]=meshgrid(x,y);

subplot(2,2,1)
zz=peaks(xx,yy);
contourf(zz);

subplot(2,2,2);
v=[-1,0,1];
[c,h]=contourf(zz,v);
clabel(c,h,'fontsize',16)

subplot(2,2,3)
contourf(xx,yy,zz,15)

subplot(2,2,4)
v=[1,2,3,4,5];
[c,h]=contourf(xx,yy,zz,v);
clabel(c,h,'fontsize',16)
```



`colorbar`

Der Befehl `colorbar` fügt am rechten Rand der Achse eine Farbskala mit einer Zuordnung der Farben zu den z-Werten hinzu.

### 9.2.1.25 Quiver

Erstellt von den Punkten  $(x,y)$  ausgehende Vektoren mit den Komponenten  $(u,v)$ .

`quiver`

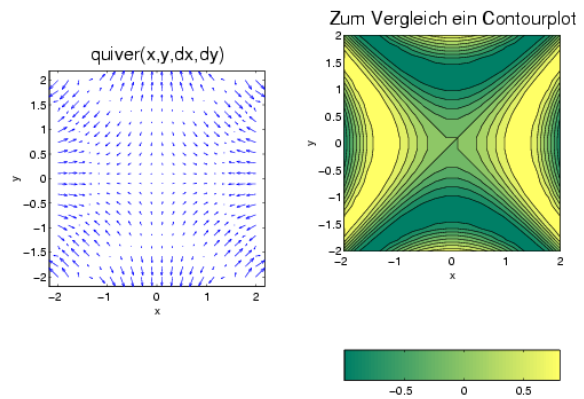
`graph_quiver.m`

Die linke Abbildung wurde mit dem Befehl `quiver` erzeugt, rechts davon befindet sich zum besseren Verständnis seiner Funktionsweise ein Contourplot

```
x=linspace(-2,2,20);  
y=linspace(-2,2,20);  
[xx,yy]=meshgrid(x,y);  
  
zz=sin(xx.^2-yy.^2);  
[dx,dy]= gradient(zz);
```

```
subplot(1,2,1)  
quiver(xx,yy,dx,dy)
```

```
subplot(1,2,2)  
contourf(xx,yy,zz)  
colorbar('horiz')
```



Mit Hilfe von `gradient` erhält man die x- und y- Komponenten des numerischen Gradienten.

Tabelle 9.4: MATLAB Befehle zum Erzeugen einfacher dreidimensionaler Graphiken

<code>plot3(x,y,z)</code>	9.2.2.1	3D Daten werden durch Angabe von x, y und z dargestellt
<code>ezplot3(x(t),y(t),z(t))</code>	9.2.2.2	Erstellt parametrischen 3D Plot durch Angabe der Funktionen als Strings und des Wertebereichs für t
<code>comet3(x,y,z,p)</code>	9.2.2.3	Zeichnet 3D Funktion in Form eines animierten 'Kometen'
<code>fill3(x,y,z,c)</code>	9.2.2.4	Malt die durch (x,y,z) definierten 3D-Polygone mit der Farbe c aus

### 9.2.1.26 Plotmatrix

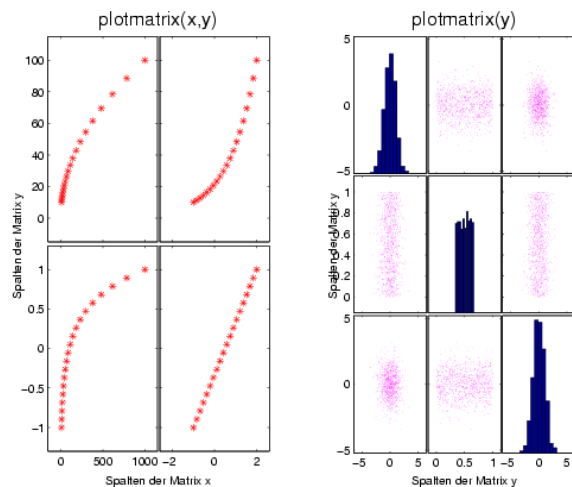
Erstellung eines Streudiagramms, die Spalten der Matrix x werden über jenen der Matrix y aufgetragen.

`plotmatrix`

`graph_plotmatrix.m`

```
subplot(1,2,1)
x1=logspace(1,3,20)';
x2=linspace(-1,2,20)';
y1=logspace(1,2,20)';
y2=linspace(-1,1,20)';
x=[x1,x2];
y=[y1,y2];

plotmatrix(x,y,'r*')
subplot(1,2,2)
y = randn(1000,3);
y(:,2)=rand(1000,1);
plotmatrix(y,'m.')
```



Wird nur eine Matrix übergeben, dann werden in den Diagonalen der Subplots Histogramme der betreffenden Spalten eingezeichnet.

## 9.2.2 Dreidimensionale Plots

Matlab bietet auch eine Fülle von Befehlen, 3D Graphiken eindrucksvoll darzustellen

### 9.2.2.1 Plot3

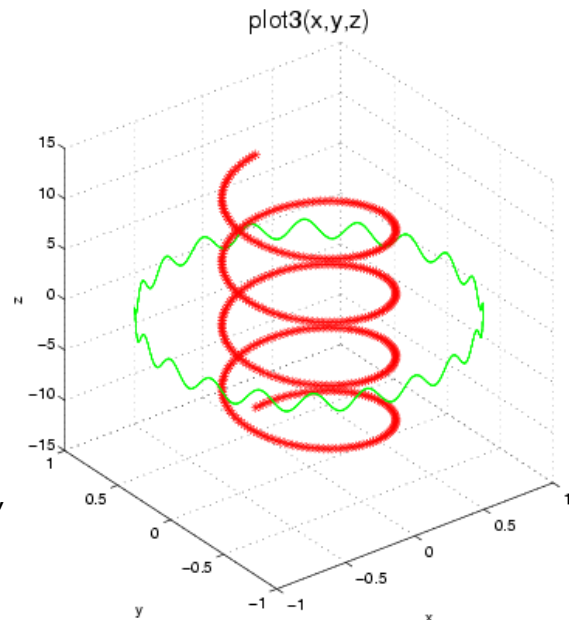
Zeichnet die Daten  $(x,y,z)$  in einem 3D-Koordinatensystem ein und verbindet sie gegebenenfalls durch Linien.

`plot3`

`graph_plot3.m`

Informationen zu den möglichen Farben und Stilen der 3D-Linien findet man unter [linespec](#)

```
t=linspace(-4*pi,4*pi,500);  
x1=0.5*sin(t);  
y1=0.5*cos(t);  
z1=t;  
x2=cos(t);  
y2=sin(t);  
z2=cos(20*t);  
  
plot3(x1,y1,z1,'r*-',x2,y2,z2,'  
  
rotate3d
```



Der Befehl `rotate3d` ermöglicht eine Drehung des Achsensystems mit Hilfe der Maus.

### 9.2.2.2 Ezplot3

Die 'Easy to Plot' Version von `plot3` zeichnet die durch  $x(t)$ ,  $y(t)$  und  $z(t)$  definierte parametrische 3D-Kurve, wobei  $x$ ,  $y$  und  $z$  von  $t$  abhängige Funktionen sind.

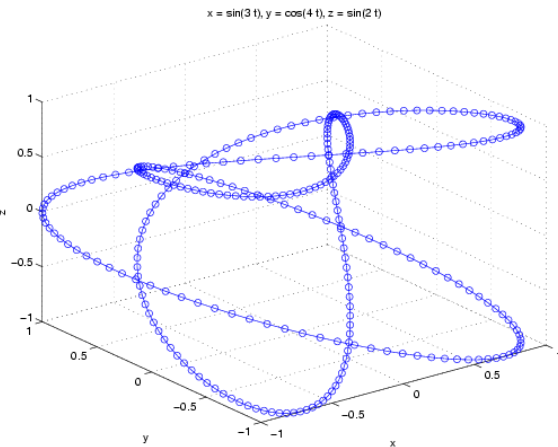
---

`ezplot3`

`graph_ezplot3.m`

---

```
h=ezplot3('sin(3*t)','cos(4*t)'  
          'sin(2*t)',[0,2*pi]);  
  
set(h, 'marker','o')  
rotate3d
```



---

Die Grenzen von  $t$  sind, wenn nicht anders festgelegt, 0 und  $2\pi$ , die Achsenbeschriftung erfolgt automatisch.

### 9.2.2.3 Comet3

Erstellt eine 3 dimensionale Funktion in Form eines sich bewegenden 'Kometen', dessen Schweif bzw. Spur den Graphen darstellt.

---

`comet3`

`graph_comet3.m`

---

Optional kann in `comet3` die Schweiflänge relativ zur Gesamtlänge des Graphen angegeben werden.

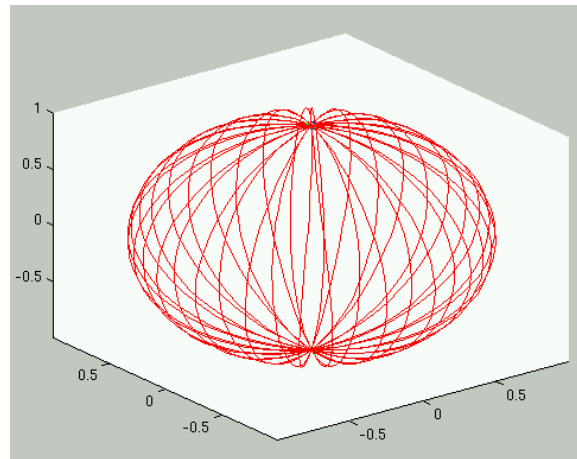
```
t=linspace(0,2*pi,1000);
```

```
x=cos(t).*sin(20*t);
```

```
y=sin(t).*sin(20*t);
```

```
z=cos(20*t);
```

```
comet3(x,y,z);
```



---

Achtung, die Erstellung des Graphen erfolgt im `erasemode none`, wird das Graphikfenster vergrößert, verschwindet der Graph, er kann daher auch nicht gedruckt werden.



Tabelle 9.5: MATLAB Befehle zum Erzeugen von 3D-Balken- und Kreisdiagrammen

<code>bar3(x,y,w,'style')</code>	9.2.2.5	Stellt die 2D Daten als vertikale 3D Balken dar
<code>bar3h(x,y,w,'style')</code>	9.2.2.6	Stellt die 2D Daten als horizontale 3D Balken dar
<code>pie3(x,'explode')</code>	9.2.2.7	Zeichnet ein 3D Kreisdiagramm von x

### 9.2.2.4 Fill3

Zeichnet dreidimensionale Polygone durch Angabe der Eckpunkte sowie der Füllfarben. Die Punkte werden in Form von Vektoren für die x-, y- und z- Komponenten angegeben, die Farbe c als Index in der aktuellen `colormap`.

`fill3`

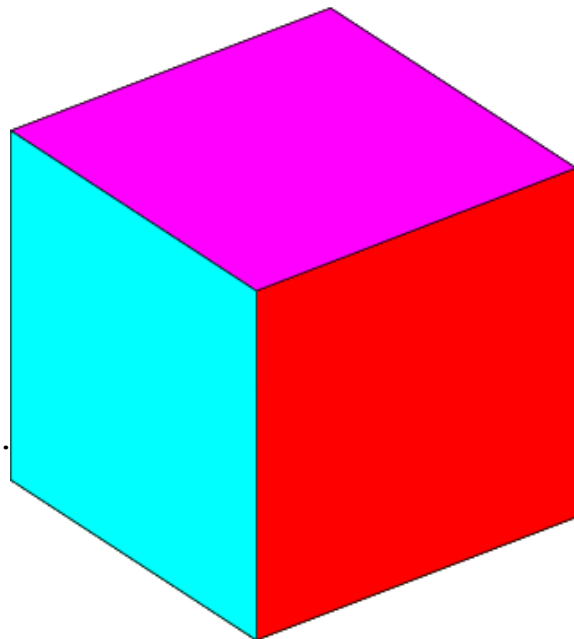
`graph_fill3.m`

Definition der 6 Flächen eines Würfels:

```
x=[0,1,1,0;0,1,1,0;1,1,1,1;...
    0,1,1,0;0,1,1,0;0,0,0,0]';
y=[0,0,0,0;0,0,1,1;0,1,1,0;...
    1,1,1,1;0,0,1,1;0,1,1,0]';
z=[0,0,1,1;0,0,0,0;0,0,1,1;...
    0,0,1,1;1,1,1,1;0,0,1,1]';
```

```
colormap([1,0,0;0,1,0;0,0,1;...
          1,1,0;1,0,1;0,1,1]);
```

```
fill3(x,y,z,1:6)
```



### 9.2.2.5 Bar3

Daten von  $y$  werden entlang der Abszisse als vertikale Säulen der Breite  $w$  dargestellt.

`bar3`

`graph_bar3.m`

Wird der Vektor  $x$  angegeben, so werden die Säulen an den Positionen von  $x$  aufgetragen, sonst bei den Werten von 1 bis  $\text{length}(n)$

```
y=sort(rand(3,5))';  
x=linspace(12,14,size(y,1));  
colormap([0,0,1;1,0,0;0,1,0]);
```

```
subplot(2,2,1)
```

```
bar3(y,0.5)
```

```
subplot(2,2,2)
```

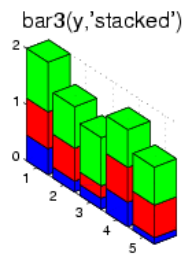
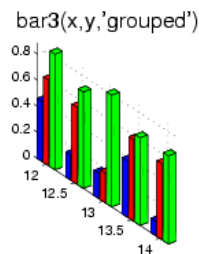
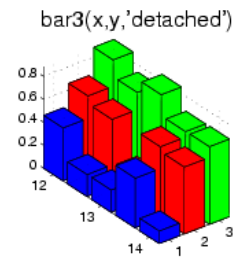
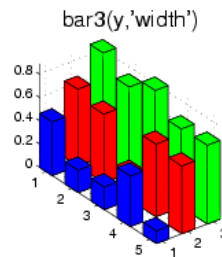
```
bar3(x,y,'detached')
```

```
subplot(2,2,3)
```

```
bar3(x,y,'grouped')
```

```
subplot(2,2,4)
```

```
bar3(y,'stacked')
```



Man beachte die unterschiedliche Darstellung der Säulendiagramme bei der Verwendung der Stile 'detached', 'grouped' und 'stacked'.

### 9.2.2.6 Bar3h

Daten von  $y$  werden als horizontale Säulen der Breite  $w$  gezeichnet.

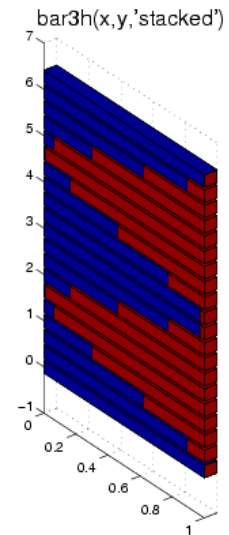
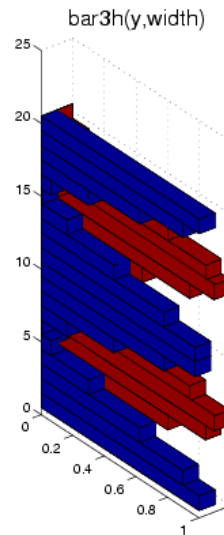
`bar3h`

`graph_bar3h.m`

```
x=linspace(0,2*pi,20)';  
y=[cos(x).^2,sin(x).^2];
```

```
subplot(1,2,1)  
bar3h(y,1);
```

```
subplot(1,2,2);  
bar3h(x,y,'stacked');
```



Hier gilt dasselbe wie bei `bar3` mit dem Unterschied, dass hier Ordinate und Abszisse vertauscht sind.

### 9.2.2.7 Pie3

Die Daten des Vektors  $x$  werden als 3D-Kreisdiagramme dargestellt, wobei die Segmente optional mit Hilfe des Vektors 'explode' hervorgehoben werden können.

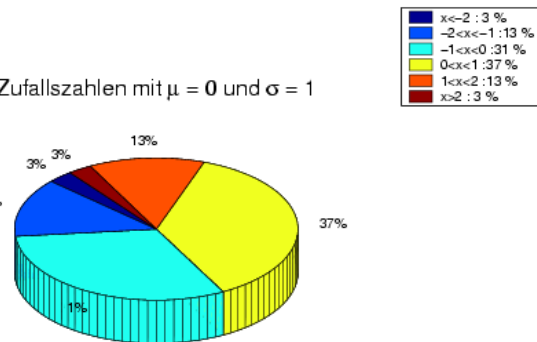
`pie3`

`graph_pie3.m`

Anteile normalverteilter Daten innerhalb bestimmter Intervalle (siehe Legende)

```
x=randn(1000,1);
y1=length(x(find(x<-2)));
y2=length(x(find(x<-1 & x>-2)));
y3=length(x(find(x<0 & x>-1)));
y4=length(x(find(x<1 & x>0)));
y5=length(x(find(x<2 & x>1)));
y6=length(x(find(x>2)));
y=[y1,y2,y3,y4,y5,y6];
```

Zufallszahlen mit  $\mu = 0$  und  $\sigma = 1$



```
h=pie3(y);
```

Der Vektor 'explode' muß die selbe Länge wie  $x$  aufweisen, Einträge des Wertes 1 führen zur Betonung des entsprechenden Segments.

Tabelle 9.6: MATLAB Befehle zum Erstellen von 3D - Oberflächen

<code>contour3(x,y,z)</code>	9.2.2.8	Zeichnet durch $z=f(x,y)$ definierte 3D-Konturlinien
<code>mesh(x,y,z)</code>	9.2.2.9	Stellt die Matrix $z=f(x,y)$ in Form eines 'Drahtgitters' dar
<code>ezmesh('f(x,y)')</code>	9.2.2.10	'Easy to use' Variante von mesh, $f(x,y)$ wird als String eingegeben
<code>meshc(x,y,z)</code>	9.2.2.11	Zeichnet ein 3D-Drahtgitter und einen 2D-Contourplot der Funktion $z=f(x,y)$
<code>meshz(x,y,z)</code>	9.2.2.12	Zeichnet ein 3D-Drahtgitter der Funktion $z=f(x,y)$ mit zusätzlichen seitlichen Referenzlinien
<code>trimesh(tri,x,y,z)</code>	9.2.2.13	Zeichnet ein aus Dreiecken bestehendes 3D-Drahtgitter der Funktion $z=f(x,y)$
<code>surf(x,y,z)</code>	9.2.2.14	Erstellt eine 3D-Oberflächengraphik der Funktion $z=f(x,y)$
<code>ezsurf('f(x,y)')</code>	9.2.2.15	'Easy to use' Variante von surf, $f(x,y)$ wird als String eingegeben
<code>surfc(x,y,z)</code>	9.2.2.16	Zeichnet eine 3D-Oberflächengraphik und einen 2D-Contourplot der Funktion $z=f(x,y)$
<code>ezsurfc(x,y,z)</code>	9.2.2.17	'Easy to use' Variante von surfc, $f(x,y)$ wird als String eingegeben
<code>surf1(x,y,z)</code>	9.2.2.18	Erstellt eine 3D-Oberflächengraphik der Funktion $z=f(x,y)$ mit wählbarer Beleuchtung
<code>trisurf(tri,x,y,z)</code>	9.2.2.19	Zeichnet eine 3D-Oberfläche der Funktion $z=f(x,y)$ aus Dreiecken
<code>waterfall(x,y,z)</code>	9.2.2.20	Zeichnet die Reihen der Matrix $z=f(x,y)$ als 3D-Linien entlang der x-Achse

### 9.2.2.8 Contour3

Zeichnet  $z$  als Funktion von  $x$  und  $y$  in Form von 3D-Konturlinien (Höhenlinien), die je nach Aufruf von `contour3` äquidistant sind oder bei bestimmten Werten von  $z$  liegen.

---

`contour3`

`graph_contour3.m`

---

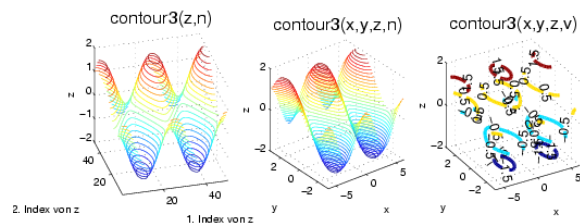
Text in Spalten

```
x=linspace(-2*pi,2*pi,50);  
y=linspace(-pi,pi,50);  
[xx,yy]=meshgrid(x,y);  
zz=cos(xx)+sin(yy);
```

```
subplot(2,2,1)  
contour3(zz,20);
```

```
subplot(2,2,2)  
contour3(xx,yy,zz,30);
```

```
subplot(2,2,4)  
v=[-1.5,-0.5,0.5,1.5];  
[c,h]=contour3(xx,yy,zz,v);  
clabel(c,h,'fontsize',12);
```



---

Mit `clabel` werden die Konturlinien mit den entsprechenden  $z$ -Werten beschriftet.

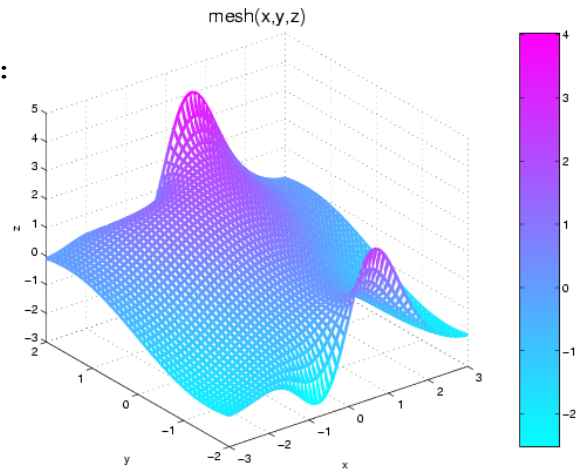
### 9.2.2.9 Mesh

Zeichnet die Funktion  $z=f(x,y)$  in Form eines Drahtgittermodells.

`mesh`

`graph_mesh.m`

```
[x,y]=meshgrid(-3:0.1:3,-2:0.1:2);  
z1=x.*exp(-x.^2+y.^2);  
z2=10+cos(x)+sin(y);  
z=z1./z2;  
  
h=mesh(x,y,z);  
  
set(h,'linewidth',2.5);  
colormap cool  
colorbar
```



Zur Erinnerung: mit `get(h)` können alle Eigenschaften des mit dem Handle `h` verknüpften Graphik-Objekts ausgegeben und mit `set(h,'Eigenschaft','Wert')` gesetzt werden.

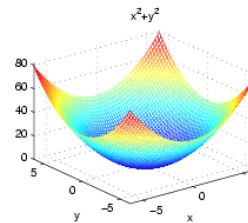
### 9.2.2.10 Ezmesh

'Easy to use' Variante von mesh, die als String eingegebene Funktion  $f(x,y)$  wird als Drahtgittermodell gezeichnet, Achsenbeschriftung und Titel werden automatisch hinzugefügt.

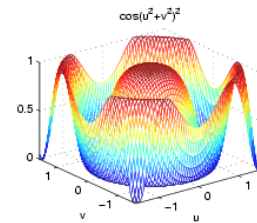
`ezmesh`

`graph_ezmesh.m`

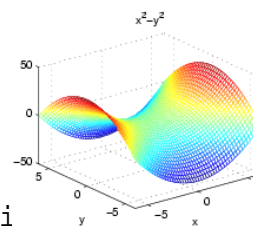
```
subplot(2,2,1)
ezmesh('x^2+y^2')
```



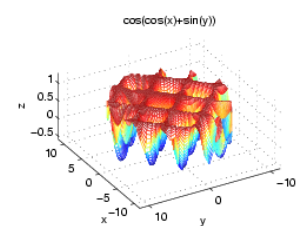
```
subplot(2,2,2)
ezmesh('cos(u^2+v^2)^2', ...
       [-pi/2,pi/2])
```



```
subplot(2,2,3)
ezmesh('x^2-y^2', 50)
```



```
subplot(2,2,4)
ezmesh('cos(cos(x)+sin(y))', 'ci')
```



Neben der Funktion  $f(x,y)$  können optional die Grenzen von  $x$  und  $y$ , die Anzahl der Gitterelemente oder der Ausdruck 'circ' (zeichnet Graphik über kreisförmigen Definitionsgebiet) angegeben werden.



### 9.2.2.11 Meshc

Die Funktion  $z=f(x,y)$  wird als 'Drahtgittermodell' inklusive 2D-Konturlinien in der Ebene  $z = 0$  gezeichnet.

---

meshc

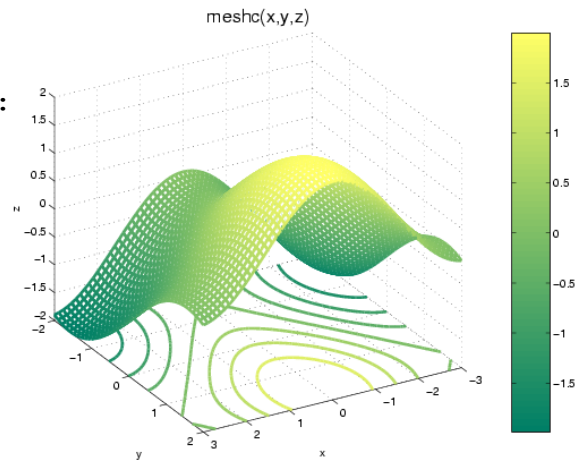
graph\_meshc.m

---

```
[x,y]=meshgrid(-3:0.1:3,-2:0.1:2);
z1=x.*exp(-x.^2-y.^2)
z2=10+cos(x)+sin(y);
z=z1./z2;

h=meshc(x,y,z);

set(h,'linewidth',2.5);
```



---

Die Dicke der Konturlinien kann nur gemeinsam mit jenen des Drahtgitters verändert werden.

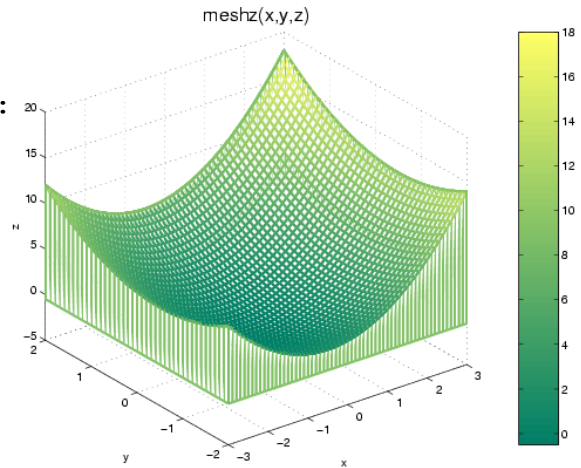
### 9.2.2.12 Meshz

Zeichnet die Funktion  $z=f(x,y)$  als 'Drahtgittermodell', wobei die Ränder des Gitters mit der durch  $z=0$  definierten Ebene verbunden sind.

`meshz`

`graph_meshz.m`

```
[x,y]=meshgrid(-3:0.1:3,-2:0.1:2);  
z=x+y+x.^2+y.^2;  
  
h=meshz(x,y,z);  
  
colorbar  
set(h,'linewidth',2.0);  
colormap summer
```



### 9.2.2.13 Trimesh

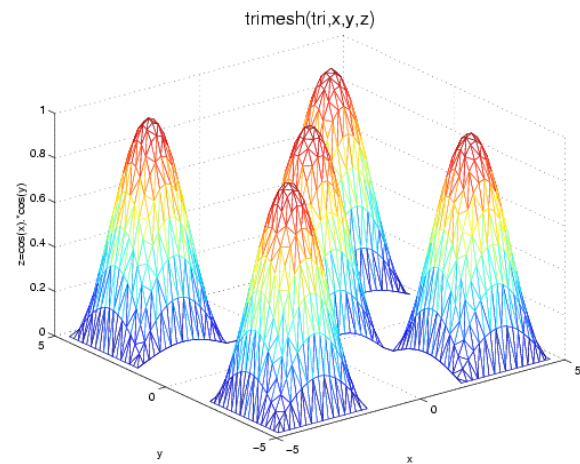
Zeichnet ein aus Dreiecken bestehendes 3D-Drahtgitter der Funktion  $z=f(x,y)$ .

`trimesh`

`graph_trimesh.m`

Die Koordinaten (`tri`) der Dreiecke werden mit der `delaunay` Triangulation aus den  $(x,y)$  Daten gewonnen.

```
t=linspace(-1.5*pi,1.5*pi,50);  
[x,y]=meshgrid(t,t);  
z=cos(x).*cos(y);  
z(z<0)=nan;  
tri = delaunay(x,y);  
  
trimesh(tri,x,y,z)
```



Elemente der Matrix `z` mit dem Eintrag `nan` werden nicht gezeichnet.

### 9.2.2.14 Surf

Erstellt eine 3D-Oberflächengraphik der Funktion  $z=f(x,y)$  mit dem in `shading` spezifizierten Schattiermodus.

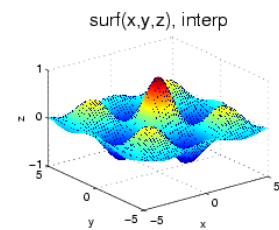
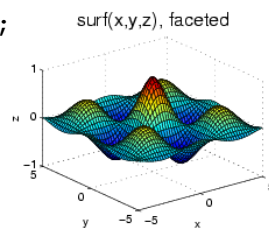
`surf`

`graph_surf.m`

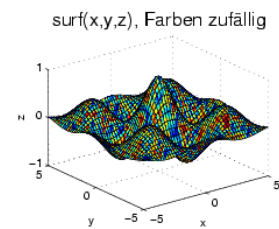
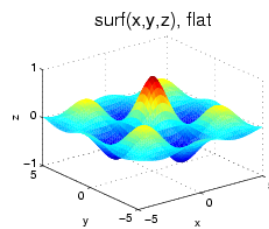
Die Farbgebung im 4. Subplot erfolgt zufällig.

```
x=linspace(-5,5,50);  
y=linspace(-5,5,50);  
[xx,yy]=meshgrid(x,y);  
z1=cos(xx).*cos(yy);  
z2=exp(-0.2*sqrt(xx.^2+yy.^2));  
zz=z1.*z2;
```

```
subplot(2,2,1)  
surf(xx,yy,zz);  
shading faceted
```



```
subplot(2,2,2)  
surf(xx,yy,zz);  
shading interp
```



```
subplot(2,2,3)  
surf(xx,yy,zz);  
shading flat
```

```
subplot(2,2,4)  
h=surf(xx,yy,zz);  
shading interp  
set(h,'cdata',rand(size(zz)),'edgecolor','k')
```

Werden im Aufruf von `surf` die x- und y- Matrizen weggelassen, so werden auf den x- und y- Achsen die beiden Indizes der Matrix z aufgetragen.

### 9.2.2.15 Ezsurf

Die 'Easy to use' Variante von `surf` mit automatischer Achsenbeschriftung und Überschrift.

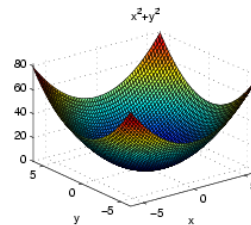
---

`ezsurf`

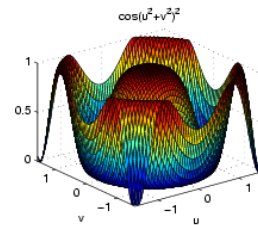
`graph_ezsurf.m`

---

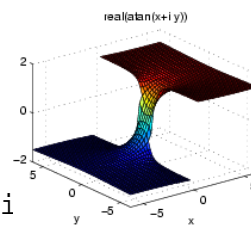
```
subplot(2,2,1)
ezsurf('x^2+y^2')
```



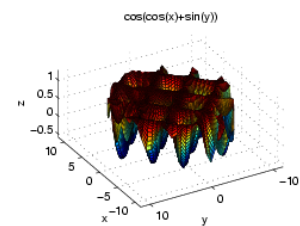
```
subplot(2,2,2)
ezsurf('cos(u^2+v^2)^2',...
      [-pi/2,pi/2])
```



```
subplot(2,2,3)
i=sqrt(-1);
ezsurf('real(atan(x+i*y))',50)
```



```
subplot(2,2,4)
ezsurf('cos(cos(x)+sin(y))', 'ci
view(-120,50)
```



---

Neben der Funktion  $f(x,y)$  können optional die Grenzen von  $x$  und  $y$ , die Anzahl der Gitterelemente oder der Ausdruck 'circ' (zeichnet Graphik über kreisförmigen Definitionsgebiet) angegeben werden. Mit Hilfe des Befehls `view` stellt man den Blickwinkel auf das Achsensystem ein. Die erste Komponente ist der Azimuthwinkel in Grad (Rotation der  $x,y$  Ebene), die zweite Komponente ist der Kippwinkel aus der horizontalen Lage der  $x,y$  Ebene.

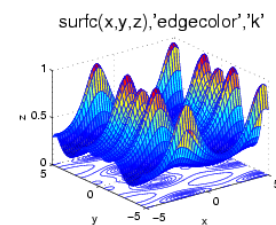
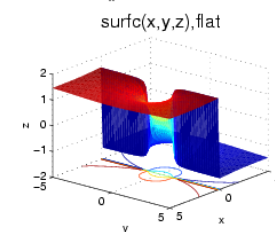
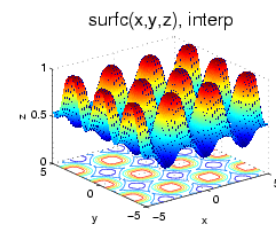
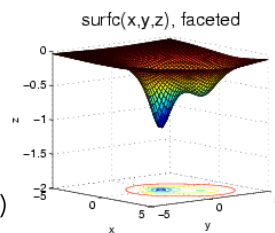
### 9.2.2.16 Surfz

Erstellt eine 3D-Oberflächengraphik der Funktion  $z=f(x,y)$  mit dem in `shading` spezifizierten Schattiermodus und fügt 2D-Konturlinien in der Ebene  $z = 0$  hinzu.

`surfz`

`graph_surfz.m`

```
x=linspace(-5,5,50);  
[xx,yy]=meshgrid(x,x);  
  
subplot(2,2,1)  
zz=-1./(xx.^2+yy.^2+1)-1./...  
    ((xx-2).^2+(yy-2).^2+2);  
surfz(xx,yy,zz)  
shading faceted  
  
subplot(2,2,2)  
zz=1./(cos(xx).^4+sin(yy).^4+1)  
surfz(xx,yy,zz)  
shading interp  
  
subplot(2,2,3)  
zz=real(atan(xx+sqrt(-1)*yy));  
surfz(xx,yy,zz);  
shading flat  
  
subplot(2,2,4)  
zz=1./(sin(xx)+2+abs(yy).*cos(yy).^2);  
h=surfz(xx,yy,zz);  
set(h,'edgecolor','b')
```



### 9.2.2.17 Ezsurf

Die 'Easy to use' Variante von `surf` mit automatischer Achsenbeschriftung und Überschrift.

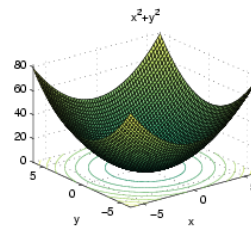
---

`ezsurf`

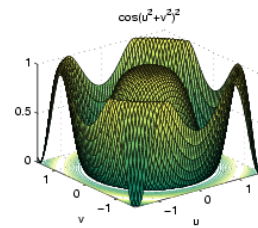
`graph_ezsurf.m`

---

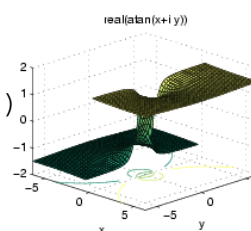
```
subplot(2,2,1)
ezsurf('x^2+y^2')
```



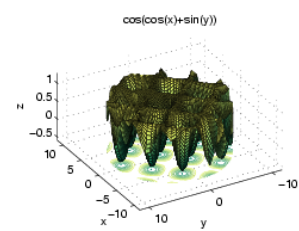
```
subplot(2,2,2)
ezsurf('cos(u^2+v^2)^2',...
      [-pi/2,pi/2])
```



```
subplot(2,2,3)
i=sqrt(-1);
ezsurf('real(atan(x+i*y))',50)
view(45,25)
```



```
subplot(2,2,4)
ezsurf('cos(cos(x)+sin(y))','circ')
```



### 9.2.2.18 Surf1

Erstellt beleuchtete 3D Oberflächenplots einer Funktion  $z=f(x,y)$ .

`surf1`

`graph_surf1.m`

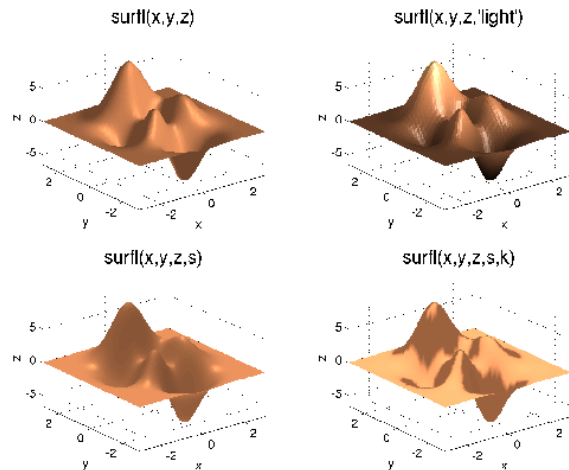
```
[x,y] = meshgrid(-3:1/8:3);  
z = peaks(x,y);
```

```
subplot(2,2,1)  
surf1(x,y,z);
```

```
subplot(2,2,2)  
surf1(x,y,z,'light')
```

```
subplot(2,2,3)  
s=[0,90];  
surf1(x,y,z,s)
```

```
subplot(2,2,4)  
s=[0,90];  
k=[1,0.1,1,0.1];  
surf1(x,y,z,s,k)
```



Der Vektor `s` beinhaltet die `x`-, `y`- und `z`-Komponenten der Einfallsrichtung des Lichts und `k` die relativen Intensitäten des Umgebungslichtes, der diffusen Reflexion, der spiegelnden Reflexion sowie des spiegelnden Glanzes.



### 9.2.2.19 Trisurf

Zeichnet eine aus Dreiecken bestehende Oberflächengraphik der Funktion  $z=f(x,y)$ .

`trisurf`

`graph_trisurf.m`

Die Koordinaten der Dreiecke werden mittels `delaunay` aus den x- und y-Werten des Gitters gewonnen.

```
t=linspace(-1.5*pi,1.5*pi,25);  
[x,y]=meshgrid(t,t);  
z=cos(x+cos(y));  
z(z<0)=0;  
tri = delaunay(x,y);
```

```
h=trisurf(tri,x,y,z);
```

```
shading interp  
set(h,'edgecolor','k')
```

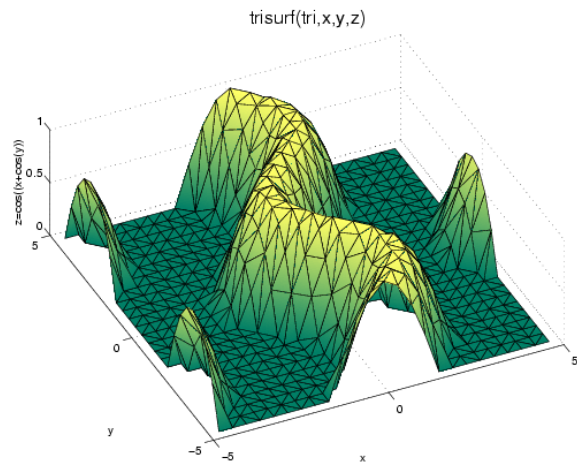


Tabelle 9.7: MATLAB Befehle zum Erstellen von 3D - volumetrischen Graphiken

<code>quiver3(x,y,z,u,v,w)</code>	9.2.2.21	Zeichnet an den Punkten $(x,y,z)$ Vektorpfeile mit den Komponenten $(u,v,w)$
<code>slice(x,y,z,d,sx,sy,sz)</code>	9.2.2.22	Veranschaulicht die volumetrische Funktion $d=f(x,y,z)$ durch senkrecht durch die Achsen gelegte Schnittflächen

### 9.2.2.20 Waterfall

Zeichnet die Reihen der Matrix  $z=f(x,y)$  als 3D-Linien entlang der x-Achse

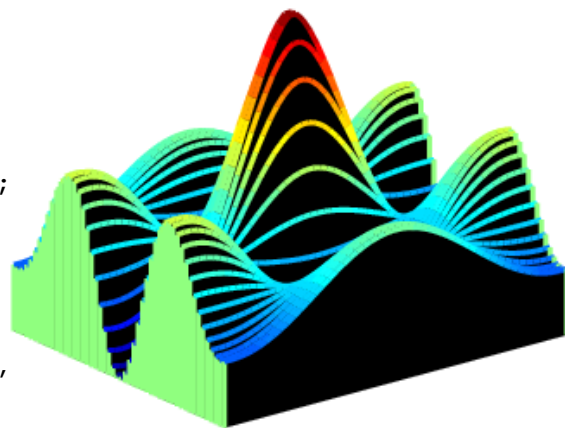
`waterfall`

`graph_waterfall.m`

```
x=linspace(-pi,pi,50);
y=linspace(-2*pi,2*pi,50);
[xx,yy]=meshgrid(x,y);
z1=cos(xx).*cos(yy);
z2=exp(-(sqrt(xx.^2+yy.^2))./4);
zz=z1.*z2;
```

```
h=waterfall(xx,yy,zz);
```

```
set(h,'linewidth',3,'facecolor'
set(gcf,'color','k');
```



### 9.2.2.21 Quiver3

Zeichnet an den Punkten  $(x,y,z)$  Vektorpfeile mit den Komponenten  $(u,v,w)$ .

`quiver3`

`graph_quiver3.m`

Es ist sinnvoll, diesen Graphikbefehl gemeinsam mit `mesh` oder `surf` zu verwenden.

```
subplot(1,2,1)
[x,y]=meshgrid(-2:0.5:2,-2:0.5:2);
z=x.^2+y.^2;
[u,v,w] = surfnorm(x,y,z);
```

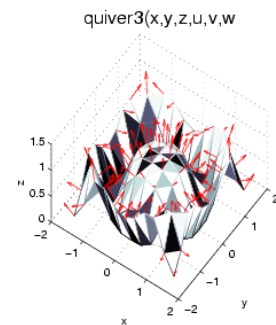
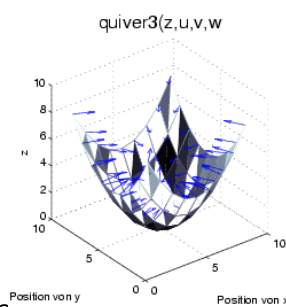
```
quiver3(z,u,v,w)
```

```
hold on
mesh(z)
```

```
subplot(1,2,2)
[x,y]=meshgrid(-pi/2:pi/10:pi/2, .
z=cos(x.^2+y.^2).^2;
[u,v,w] = surfnorm(x,y,z);
```

```
quiver3(x,y,z,u,v,w,'r')
```

```
hold on
mesh(x,y,z)
```



Die Komponenten der Normalvektoren auf die Oberfläche  $z=f(x,y)$  werden mit dem Befehl `[u,v,w]=surfnorm(x,y,z)` berechnet.

Tabelle 9.8: Weitere spezielle 3D Graphik-Befehle

<code>stem3(x,y,z)</code>	9.2.2.23	Zeichnet 3D Funktion und verbindet Datenpunkte mit der Ebene $z=0$
<code>sphere(n)</code>	9.2.2.24	Erstellt eine durch $n^2$ Flächen angenäherte Kugel
<code>cylinder(r,n)</code>	9.2.2.25	Erstellt einen durch ein n-seitiges Prisma angenäherten Zylinder mit Radius $r$
<code>scatter3(x,y,z,r,c)</code>	9.2.2.26	Zeichnet Daten an den Positionen $(x,y,z)$ der Größe $r$ sowie der Farbe $c$
<code>ribbon(y,z,w)</code>	9.2.2.27	Zeichnet die Spalten von $z$ über jenen von $y$ als 3D Bänder der Breite $w$

### 9.2.2.22 Slice

Veranschaulicht die volumetrische Funktion  $d=f(x,y,z)$  durch senkrecht durch die Achsen gelegte Schnittflächen. Dabei wird die  $x$ -Achse an den Stellen des Vektors `xslice` geschnitten, analog für die beiden anderen Achsen.

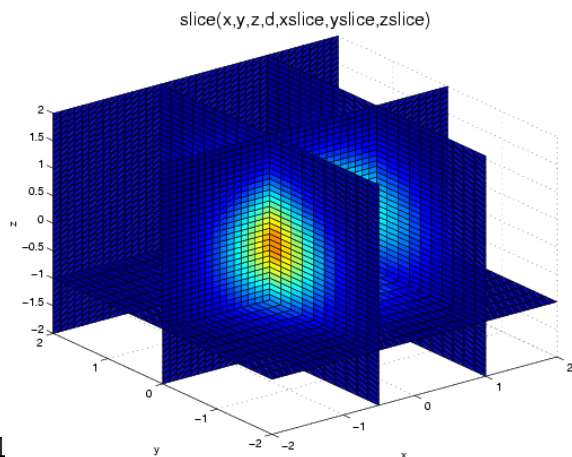
`slice`

[graph\\_slice.m](#)

Wie zu den Achsen geneigte Schnittflächen erstellt werden, findet man in in der Hilfe von `slice`

```
[x,y,z] = meshgrid(-2:.1:2,...
                  -2:.2:2,-2:.1:2);
d=exp(-x.^2-y.^2-z.^2);
xslice = [-0.5,1];
yslice = [0,2];
zslice = [-1];
```

```
slice(x,y,z,d,xslice,yslice,zsl
```



Mit `meshgrid` lassen sich auch die  $x$ -,  $y$ - und  $z$ - Koordinaten dreidimensionaler Gitter berechnen.

### 9.2.2.23 Stem3

Zeichnet dreidimensionale Daten und verbindet Datenpunkte mit der Ebene  $z=0$ .

`stem3`

`graph_stem3.m`

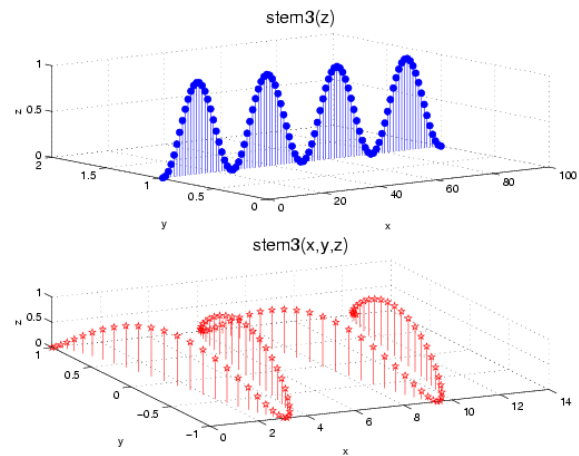
Die in `linespec` definierten Datensymbole können mit der Option 'filled' ausgefüllt werden.

```
t=linspace(0,4*pi,100);  
x=t;  
y=cos(t);  
z=sin(t).^2;
```

```
subplot(2,1,1)  
stem3(z,'filled')
```

```
subplot(2,1,2);  
stem3(x,y,z,'rp')
```

```
view(-25,60)
```



Wird `stem3` nur der Vektor  $z$  übergeben, dann wird  $z$  über  $x=1$  bis  $\text{size}(z,1)$  und  $y=1$  bis  $\text{size}(z,2)$  aufgetragen.

### 9.2.2.24 Kugel

Erstellt eine durch  $n \times n$  Segmenten angenäherte Kugel mit dem Radius 1.

`sphere`

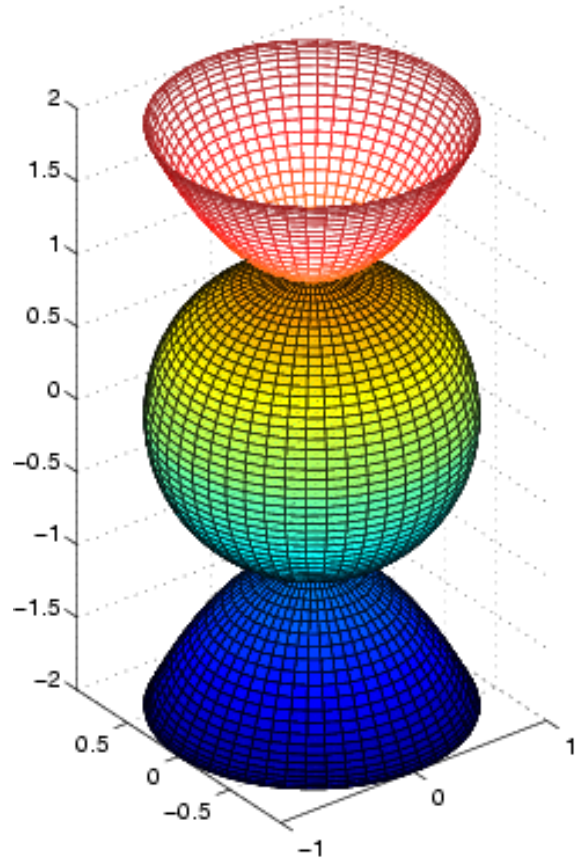
`graph_sphere.m`

Einheitskugel mit vertikal angrenzenden paraboloid-ähnlichen Objekten.

```
sphere(50)
[x,y,z]=sphere(50);

hold on
mesh(x,y,-z.^2+2)

surf(x,y,z.^2-2)
axis equal
```



Wird der Befehl in Form von `[x,y,z]=sphere(n)` verwendet, so können wie im Beispiel mit `surf(x,y,z)` oder `mesh(x,y,z)` ebenfalls Kugeln und kugelähnliche Objekte gezeichnet werden. Der Vorteil liegt darin, dass auf diese Weise Eigenschaften wie Größe, Position und Farben beeinflusst werden können.

### 9.2.2.25 Zylinder

Erstellt Zylinder (bzw. n-seitige Prismen) und allgemeine um die z-Achse symmetrische Körper der Höhe 1 mit der Profilkurve  $r(h)$ .

`cylinder`

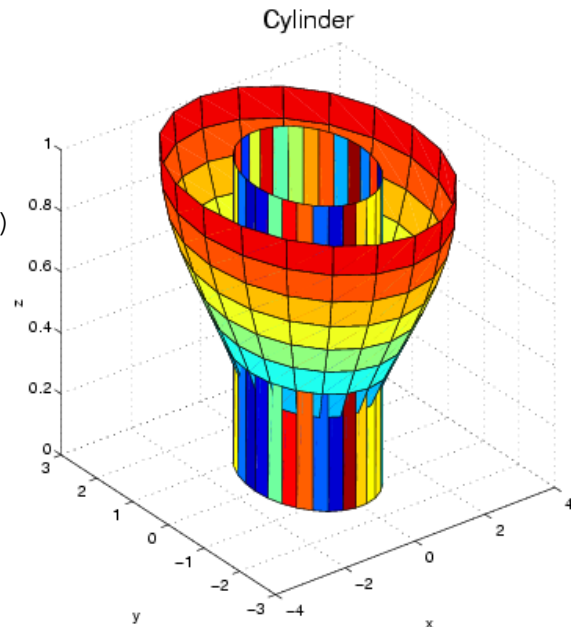
`graph_cylinder.m`

Zylinder und Rotationskörper mit der Profilkurve  $r(t)=2+\cos(t)$

```
t = pi:pi/10:2*pi;
[X1,Y1,Z1] = cylinder(2+cos(t))
[X2,Y2,Z2] = cylinder(1.5,30);

h1=surf(X1,Y1,Z1);
hold on
h2=surf(X2,Y2,Z2);
c=rand(size(get(h2,'cdata')));

set(h2,'cdata',c)
axis square
```



Wird der Befehl in Form von  $[x,y,z]=\text{cylinder}(r,n)$  verwendet, so können wie im Beispiel mit `surf(x,y,z)` oder `mesh(x,y,z)` ebenfalls Rotationskörper gezeichnet werden. Der Vorteil liegt wie im Beispiel 9.2.2.24 darin, dass auf diese Weise unter anderem Größe, Position und Farbeigenschaften beeinflusst werden können.

### 9.2.2.26 Scatter3

Zeichnet Daten an den Positionen  $(x,y,z)$  der Größe  $r$  sowie der Farbe  $c$ , wobei im Gegensatz zu `plot3` die Attribute Größe und Farbe für jeden Punkt getrennt eingestellt werden können. Allen Punkten gemeinsam ist das Datensymbol (siehe `linespec`) sowie die Option `'filled'`, wodurch Datensymbole ausgemalt werden.

`scatter3`

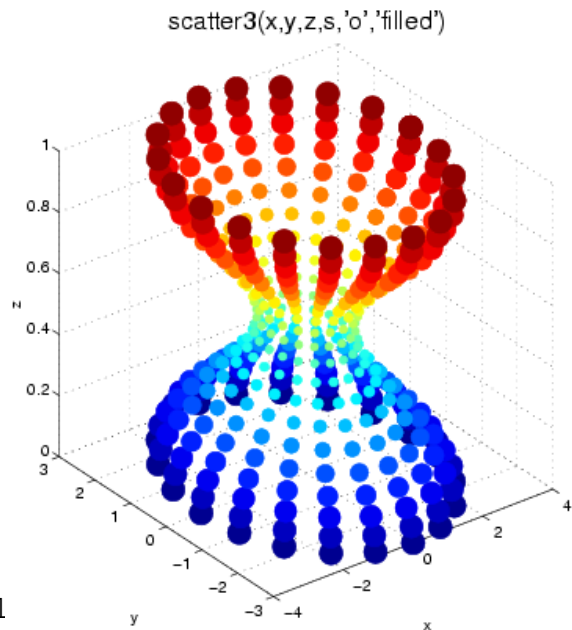
`graph_scatter3.m`

Mit Hilfe des Graphikbefehls `cylinder` erhaltene Koordinaten des Rotationskörpers der Profilkurve  $r(t)=2+\cos(t)$ . Farbe und Punktgröße hängen von den Koordinaten ab.

```
t = 0:pi/10:2*pi;
[x,y,z] = cylinder(2+cos(t));

vx=reshape(x,[],1);
vy=reshape(y,[],1);
vz=reshape(z,[],1);
r=25*((vx.^2)+(vy).^2)
c=vz;;

scatter3(vx,vy,vz,r,c,'o','fill
```





### 9.2.2.27 Ribbon

Zeichnet die Spalten von z über jenen von y als 3D Bänder der Breite w

`ribbon`

`graph_ribbon.m`

```
x=linspace(-5,5,50);  
y=linspace(-5,5,50);  
[xx,yy]=meshgrid(x,y);  
z1=cos(xx).*cos(yy);  
z2=exp(-0.2*sqrt(xx.^2+yy.^2));  
zz=z1.*z2;  
  
ribbon(yy,zz,1)
```

