

Einführung in Matlab

Einheit 3

Jochen Schulz

Georg-August Universität Göttingen 

3. September 2009

1 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Anwendung: polynomiale Interpolation
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

2 Polynome

1 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Anwendung: polynomiale Interpolation
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

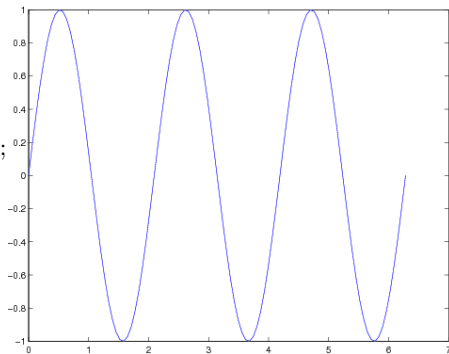
2 Polynome

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.

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

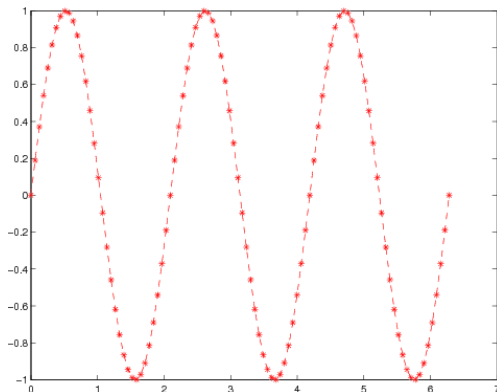


Erweiterungen

```
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.



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.)
```

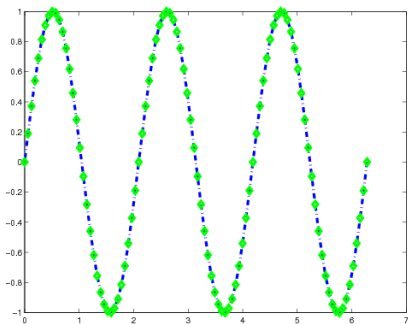
Eigenschaften:

'MarkerSize' (Default 6), 'LineWidth' (Default 0.5),

'MarkerEdgeColor', 'MarkerFaceColor'

Beispiel:

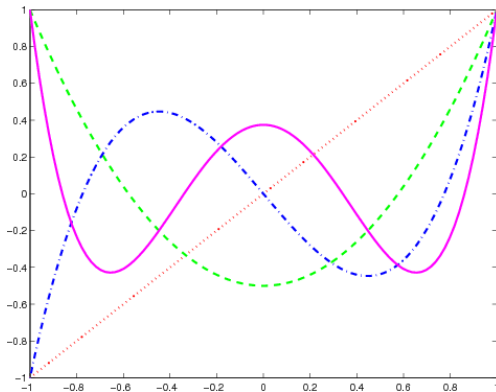
```
>>plot(x,y1,'b-.d','LineWidth',3,...  
'MarkerEdgeColor','g')
```



- 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

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

Exkurs: Characters (char)

Characters:

Characters (Zeichen) sind einzelne Zeichen.

Intern werden Characters in MATLAB durch Integer dargestellt. Die Werte zwischen 0 und 128 entsprechen den ASCII Werten. Insgesamt wird zur Speicherung eines Zeichens 2 Bytes benötigt. Es wird also jedem Zeichen eine Zahl zwischen 0 und $2^{16} - 1$ zugeordnet.

```
> s='d'
s =
d
>> s1=double(s)
s1 =
    100
>> s2=char(100)
s2 =
d
```

String:

Ein String ist ein Vektor von character (Zeichen). Intern werden sie durch die ASCII Werte dargestellt.

```
>> s='AB6de*'
s =
AB6de*
>> sd=double(s)
sd =
    65    66    54   100   101    42
>> s2=char(sd)
s2 =
AB6de*
```

Exkurs: Befehle für Strings

- Durch `strcat` werden Strings verbunden, z.B.
`strcat('Hello', ' world')`.
- `num2str(x,n)` konvertiert x in einen String mit n signifikanten Stellen.
(Default: $n = 4$)
- `int2str(x)` rundet x und konvertiert es in einen String.
- `strcmp(s,t)` vergleicht die Strings s und t .
- Durch `help strfun` erhält man eine Liste aller Befehle im Zusammenhang mit Strings.

1 Einführung Grafik

- einfache zweidimensionale Grafiken
- **Beschriftungen**
- Anwendung: polynomiale Interpolation
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

2 Polynome

Beschriften der Grafik

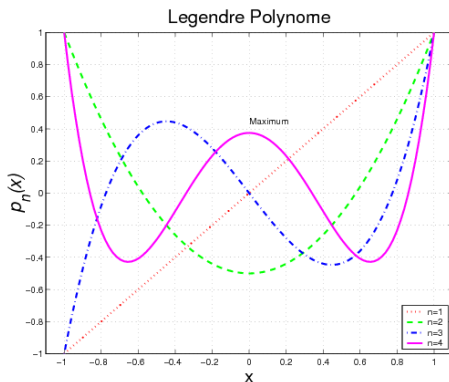
- Titel: `title('Titel')`
- Achsenbeschriftung: `xlabel('Text')`, `ylabel('Text')`
- Legende: `legend('Text1','Text2', ... ,nr)`
`nr` gibt die Position der Legendenbox in der Grafik an: -1 (rechts vom Plot), 0 'bester' Ort, 1 oben rechts (default), 2 oben links, 3 unten links, 4 unten rechts.
- zusätzlicher 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 L^AT_EX-Notation verwendet werden. Eine nahezu vollständige L^AT_EX-Unterstützung ist mit dem latex-interpreter möglich. 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 Texteingenschaften: `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])
```



- 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`

1 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Anwendung: polynomiale Interpolation
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

2 Polynome

Polynomiale Interpolation

Suche ein Polynom vom Grad 3

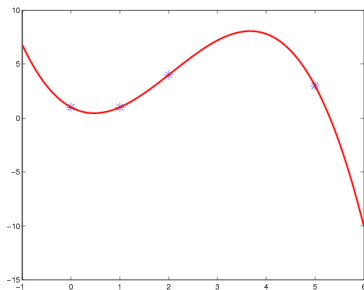
$$p(x) = p_0 + p_1x + p_2x^2 + p_3x^3,$$

dass durch die vier Punkte $(0, 1)$, $(1, 1)$, $(2, 4)$, $(5, 3)$ verläuft.

$$\Rightarrow p(0) = 1, p(1) = 1, p(2) = 4, p(5) = 3$$

\Rightarrow Lineares GLS $Ap = b$ mit

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 2^2 & 2^3 \\ 1 & 5 & 5^2 & 5^3 \end{pmatrix}, \quad p = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 4 \\ 3 \end{pmatrix},$$



Polynomiale Interpolation II

Suche ein Polynom vom Grad n

$$p(x) = p_0 + p_1x + p_2x^2 + p_3x^3 + \cdots + p_nx^n,$$

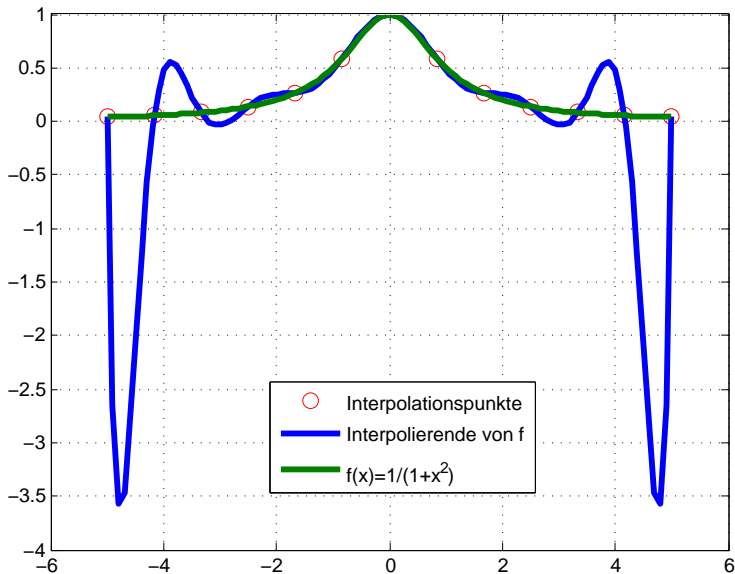
dass durch die $n + 1$ Punkte $(x_i, y_i)_{i=0}^n$ verläuft.

Beispiel: Interpolation von

$$(x_i, y_i)_{i=0}^{12}$$

mit $x=\text{linspace}(-5,5,13)$ und $y_i = \frac{1}{1+x_i^2}$.

Polynomiale Interpolation: Beispiel



Programm 1

```
function p = interpol2(x,y)
% interpol2 berechnet zu n+1 Punkten (x_i,y_i)
%           das Polynom n-ten Grades, das durch die
%           n+1 Punkte verläuft
%           INPUT: Vektoren x,y
%           OUTPUT: Koeffizientenvektor p
%   Gerd Rapin    23.11.2003

% Aufstellen des lin. GLS
A = vandermonde(x);

% Lösen des lin GLS
p = A\y';
```

Programm 2

```
% berechnet die polynomiale
% Interpolation fuer 1/(1+x^2)
%
% Gerd Rapin 23.11.2003

% Stuetzstellen
x = linspace(-5,5,13);
y = 1./(1+x.*x);
plot(x,y, 'or', 'Markersize',8);
hold on;

% Berechnen der Koeffizienten
p = interp2(x,y);

% Plotten
x1 = linspace(-5,5,100);
y1 = ausw_poly2(p',x1);
y2 = 1./(1+x1.*x1);
plot(x1,y1,x1,y2, 'Linewidth',3);
xlim([-6,6]);grid on; box on;
legend('Interpolationspunkte',...
'Interpolierende von f','f(x)=1/(1+x^2)');
hold off
```


1 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Anwendung: polynomiale Interpolation
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

2 Polynome

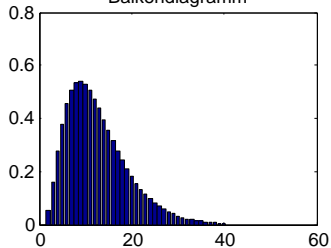
Daten:

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

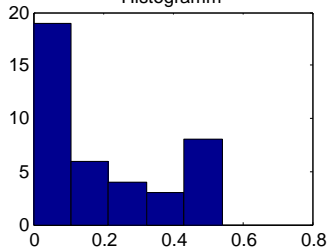
- Balkendiagramm: `bar(y)`
- Histogramm: `hist(y,5)`
- einfacher Plot: `area(n,[y',2*y'])`
- Tortengrafik: `pie3([1 2 3 4])`

Darstellung von Daten

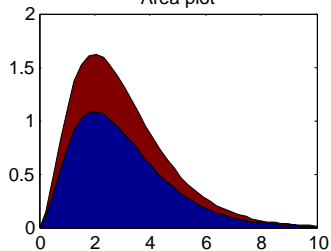
Balkendiagramm



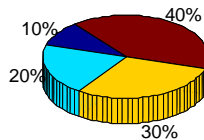
Histogramm



Area plot



Tortengrafik



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');
```

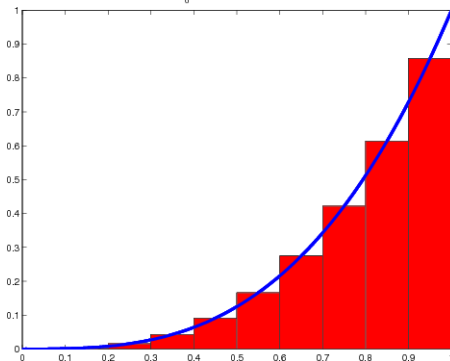
Approximation von Integralen

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

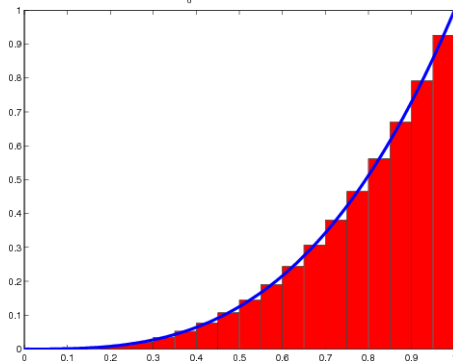
$$\int_0^1 f(x) dx \approx \sum_{i=1}^N \frac{1}{N} f\left(\frac{i - \frac{1}{2}}{N}\right)$$

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

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



$\int_0^1 x^3 = 0.24989$ fuer $N=20$



Integral - Implementation

```
% integral.m

N = 20; % 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)]);
```

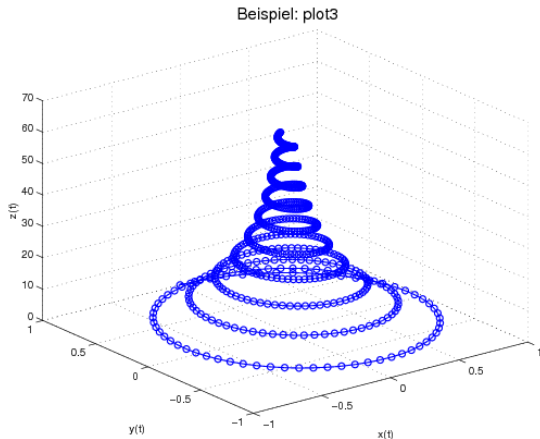
1 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Anwendung: polynomiale Interpolation
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

2 Polynome

- 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`

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.

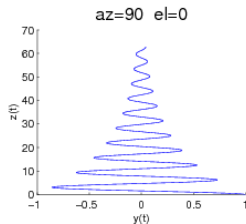
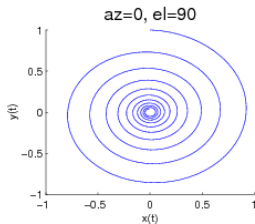
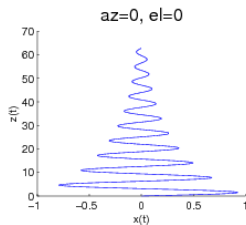
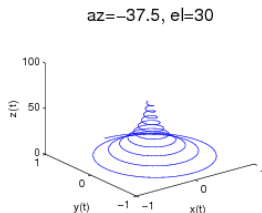


Beispiel plot3

```
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);
```

$\text{view}(\text{az}, \text{el})$

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



3D-Funktionenplots

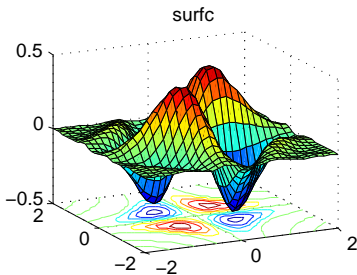
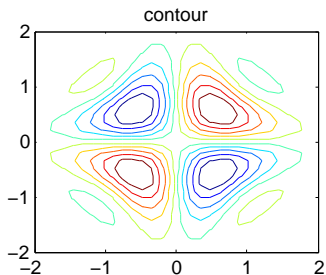
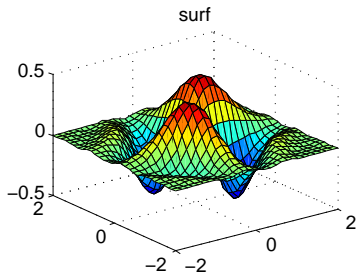
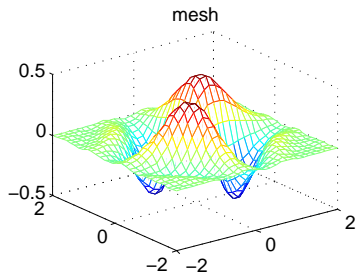
Darstellung von Funktionen

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

Beispiel:

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

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

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

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

- Contourplot (zeichnet die Niveaulinien): `contour`
- Darstellung des Graphen mit Gitterlinien: `mesh` , `meshc`
- Flächige Darstellung des Graphen: `surf`, `surfc`

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

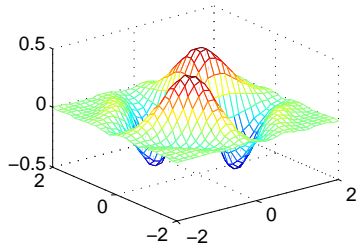
$(X(i,j), Y(i,j), Z(i,j))$ dar.

Weitere Möglichkeiten

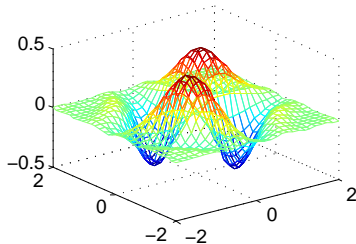
- Darstellung versteckter Linien (bei mesh): `hidden off`, Default: `hidden on`
- Verschmieren des Gitters: `shading('interp')`
- Blickwinkel: `view(az,el)`
- ähnlich wie mesh; nur mit 'Vorhang': `meshz(X,Y,Z)`

Beispiel: Funktionenplot

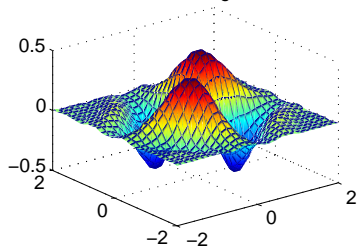
Default



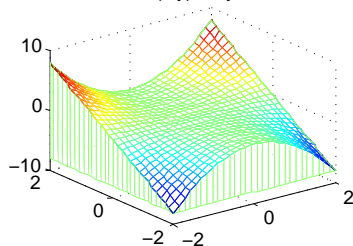
keine versteckten Linien



Shading



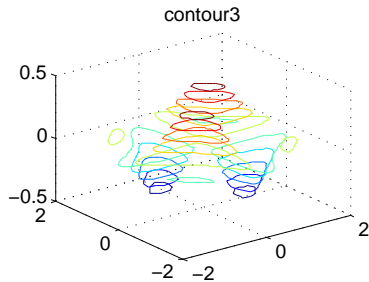
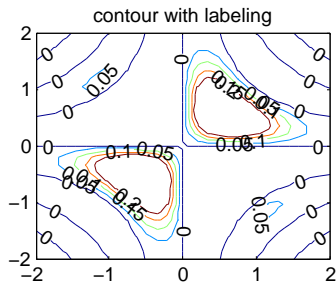
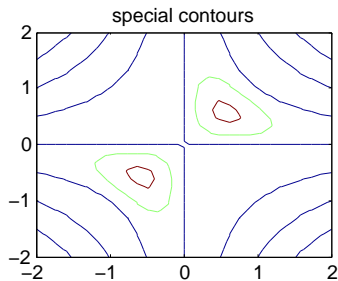
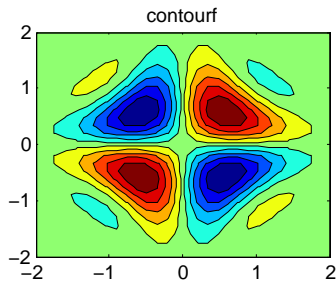
$f(x,y)=x^2 y$



```
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

```
% verschiedenen 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 dass die Flächen zwischen den Konturlinien ausgefüllt werden.
- `label(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(X,Y,Z,V,sx,sy,sz)
```

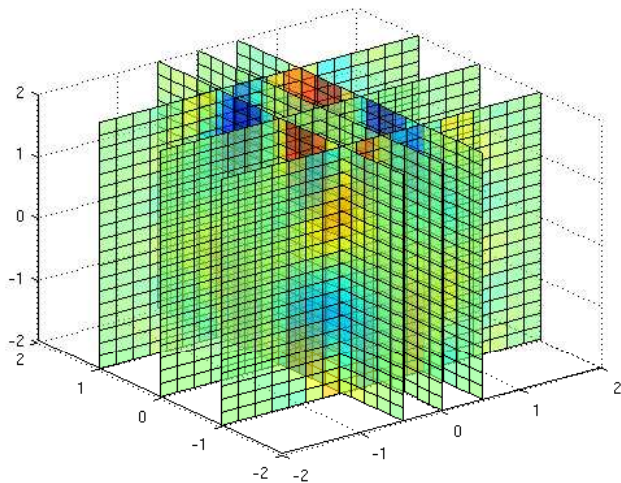
zeichnet Schnitte zu den Funktionswerten $V(i)$ zu $(X(i), Y(i), Z(i))$.
Schnitte sind durch die Vektoren sx , sy und sz gegeben.

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 = [-0.5,0,0.5]; sy = [-1,0,1];  
sz = [];  
slice(X,Y,Z,V,sx,sy,sz)  
alpha(0.6) % Transparency
```


Beispiel: slice



1 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Anwendung: polynomiale Interpolation
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

2 Polynome

Animation-Beispiel

```
% animation.m

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

for j = 1:30
    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 $F(j)=\text{getframe}$ wird die aktuelle Grafik in das Array F gespeichert.
- Nach dem alle Bilder zu F hinzugefügt worden sind, kann man die Sequenz der Bilder F darstellen durch $\text{movie}(F,n,\text{fps})$, wobei n die Anzahl der Wiederholungen angibt und fps der gezeigten Frames pro Sekunde entspricht (Default: $n = 1$, $\text{fps} = 12$).
- Speichern des Movies in AVI Format: $\text{movie2avi}(F,\text{Dateiname})$

1 Einführung Grafik

- einfache zweidimensionale Grafiken
- Beschriftungen
- Anwendung: polynomiale Interpolation
- Weitere zweidimensionale Darstellungsmöglichkeiten
- Dreidimensionale Grafiken
- Animation

2 Polynome

In MATLAB werden Polynome

$$p(x) = p_1 x^n + p_2 x^{n-1} + \dots + p_{n+1}$$

repräsentiert durch einen Zeilenvektor $p = [p(1) \ p(2) \ \dots \ p(n+1)]$.

Vorsicht: Normalerweise werden Polynome in der Form $\sum_{i=0}^n p_i x^i$ dargestellt. In MATLAB dagegen ist die Darstellung invers und beginnt bei 1.

1. **Auswerten:** Bei gegebenen Koeffizienten, das zugehörige Polynom an bestimmten Stellen auswerten.
2. **Nullstellenbestimmung:** Bestimme zu gegebenen Koeffizienten die Nullstellen des zugehörigen Polynoms.
3. **Interpolation:** Bestimme zu einer gegebenen Menge von Punkten $(x_i, y_i)_{i=0}^n$ ein Polynom n -ten Grades, das durch diese Punkte verläuft.

Auswerten

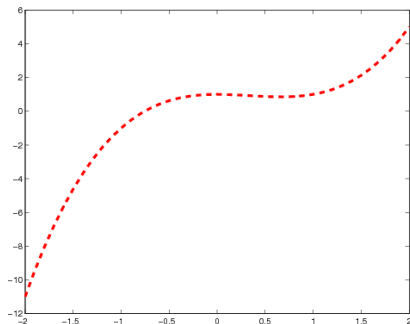
Durch

```
y=polyval(p,x)
```

werden aus einem vorgegebenen Koeffizientenvektor p und entsprechenden Stellen x die zugehörigen Funktionswerte y berechnet. x kann eine Matrix sein. y ist dann von der gleichen Dimens. wie x .

Beispiel: $p(x) := x^3 - x^2 + 1$

```
>> x=-2:0.1:2;  
>> y=polyval([1 -1 0 1],x);  
>> plot(x,y, 'r—', 'Linewidth',3);
```



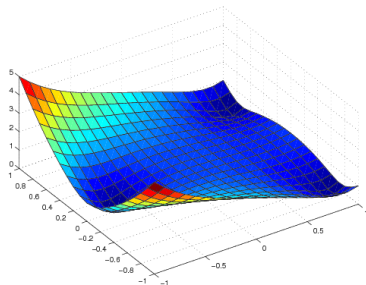
Bestimmung von Nullstellen

Ist p der obige Koeffizientenvektor, so können die Nullstellen z durch $z = \text{roots}(p)$ berechnet werden.

Beispiel:

Nullstellen von $p(x) := x^3 - x^2 + 1$

```
>> roots([1 -1 0 1])  
ans =  
    0.8774 + 0.7449i  
    0.8774 - 0.7449i  
   -0.7549  
>> x=-1:0.1:1; [X,Y]=meshgrid(x,x);  
>> Z=abs(polyval([1 -1 0 1],X+i*Y));  
>> surf(X,Y,Z)
```



Interpolation

Suche zu gegebenen Punkten $(x_i, y_i)_{i=0}^n$ ein Polynom p n -ten Grades, so dass $p(x_i) = y_i$ gilt für $i = 0, \dots, n$.

In MATLAB: `p=polyfit(x,y,n)`

Ruft man `p=polyfit(x,y,m)` mit $m < n$ auf, so sucht MATLAB die Least Square Lösung, d.h. das Polynom p der Ordnung m , welches $\sum_{i=0}^n (p(x_i) - y_i)^2$ minimiert.

Ein weiterer Befehl zur Interpolation ist

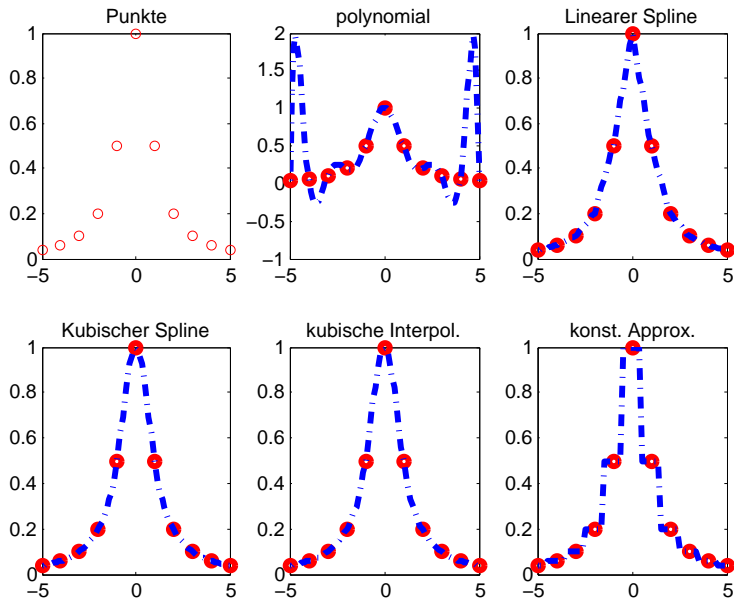
```
yi=interp1(x,y,xi,'method').
```

Dabei sind (x, y) die gegebenen Punkte, x_i sind die Stellen, an die die Interpolante berechnet wird und y_i sind die entsprechenden

Funktionswerte. Als 'method' gibt es

- | | |
|-----------|--|
| 'nearest' | stückweise konstante Approximation |
| 'linear' | Lineare Interpolation |
| 'spline' | stückweise kubischer Spline u ($u \in C^2$, $u _{[x_i, x_{i+1}]} \in \mathbb{P}_3$) |
| 'cubic' | kubische Hermite Interpolation |

Beispiel



- Nur für die Spline-Methoden können bei `interp1` auch Stellen außerhalb des Interpolationsintervalls berechnet werden.
- Data Fitting kann auch über die Oberfläche durchgeführt werden. Plotten Sie die Daten und wählen Sie Basic Fitting im Menü Tools.