ulm university universität ulm

WiMa-Praktikum (Matlab 4/8)

Einführung in LaTeX und Matlab

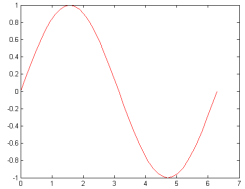
Prof. Dr. Stefan Funken, Dipl.-Ing. Christoph Erath
6. Juni 2009

2D Plots

Es gibt mehrere Möglichkeiten in Matlab zweidimensionale Graphen von Funktionen zu Plotten:

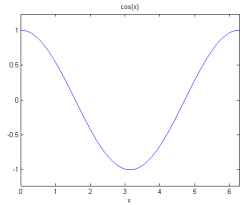
- Erzeugen von Punktemengen $(x, f(x))$ und Plotten mit der Funktion `plot`:

```
>> x=0:pi/20:2*pi;
>> fx=sin(x);
>> plot(x,fx, 'r')
```



- Plotten einer inline-Funktion mit der Funktion `fplot`:

```
>> fplot('cos(x)', [0, 2*pi]);
```



Zur Berechnung eines Graphen kann ein Gitter des Urbildbereichs erzeugt werden.

- `linspace(x1,x2,N)` oder `x1:h:x2` erzeugt eine äquidistante Unterteilung eines Intervalls,

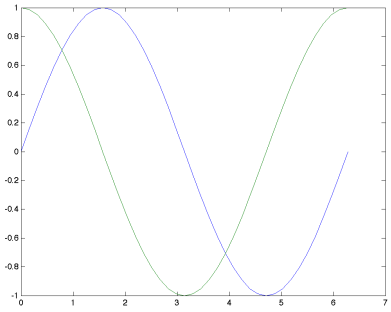
```
>> linspace(1,5,5)
ans =
     1     2     3     4     5
>> 10.^linspace(1,5,5)    % == logspace(1,5,5)
ans =
          10         100        1000       10000      100000
```

2D Plots mehrerer Graphen

Soll mehr als ein Graph in ein Koordiantensystem geplottet werden, so kann dies auf verschiedene Arten erreicht werden:

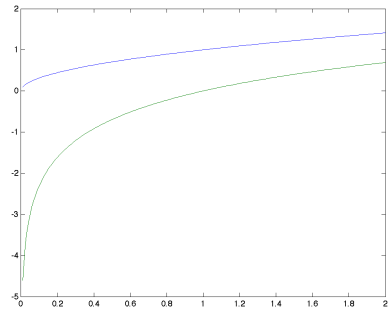
- Man erstellt einen Spaltenvektor von x-Werten, erzeugt eine Matrix mit den Funktionswerten und plottet die Spalten der Matrix:

```
>> x=[0:pi/20:2*pi]';
>> plot(x, [sin(x) cos(x)]);
```



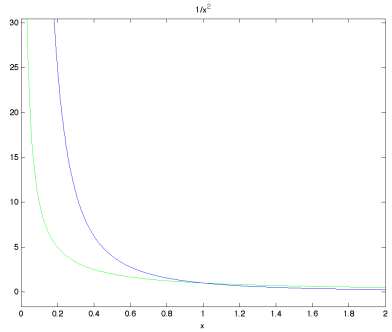
- Die Graphen werden einzeln aufgelistet:

```
>> x=[0.01:.01:2];
>> y=[0.01:.02:2];
>> plot(x, sqrt(x), y, log(y))
```



- Mit `hold` werden Graphen in einem Figure nicht durch neue Plotbefehle überschrieben. Neue Graphen werden ins Koordinatensystem eingefügt.

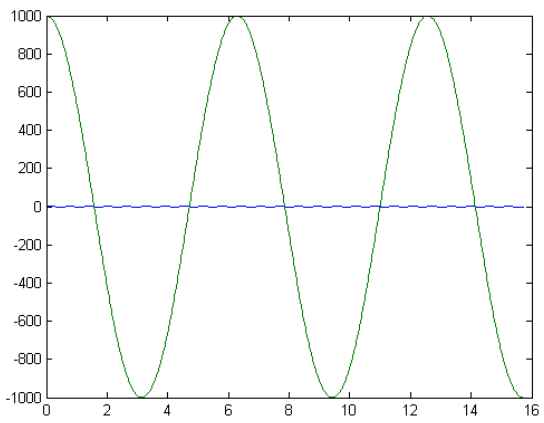
```
>> x=[0.01:.01:2];
>> plot(x, 1./x, 'g');
>> hold on
>> fplot('1./x.^2', [0, 2]);
>> hold off
```

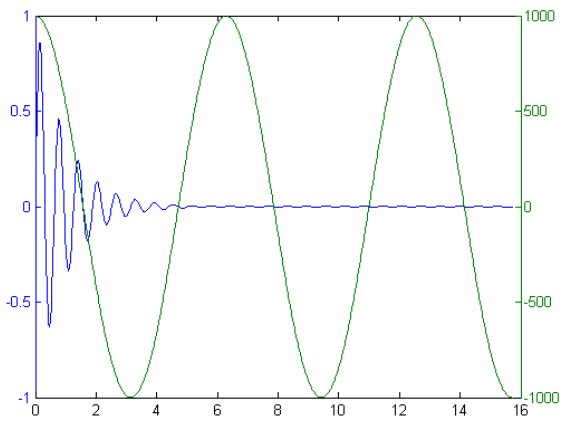


2D Plots meherer Graphen

- Sollen mehrere Graphen mit unterschiedlicher Skalierung der y-Achsen geplottet werden, so kann der Befehl `plotyy` verwendet werden:

```
>> t=linspace(0, 5*pi, 1000);
>> x=exp(-t).*sin(10.*t);
>> y = 1000*cos(t);
>> plot(t,x,t,y);
>> plotyy(t,x,t,y);
```



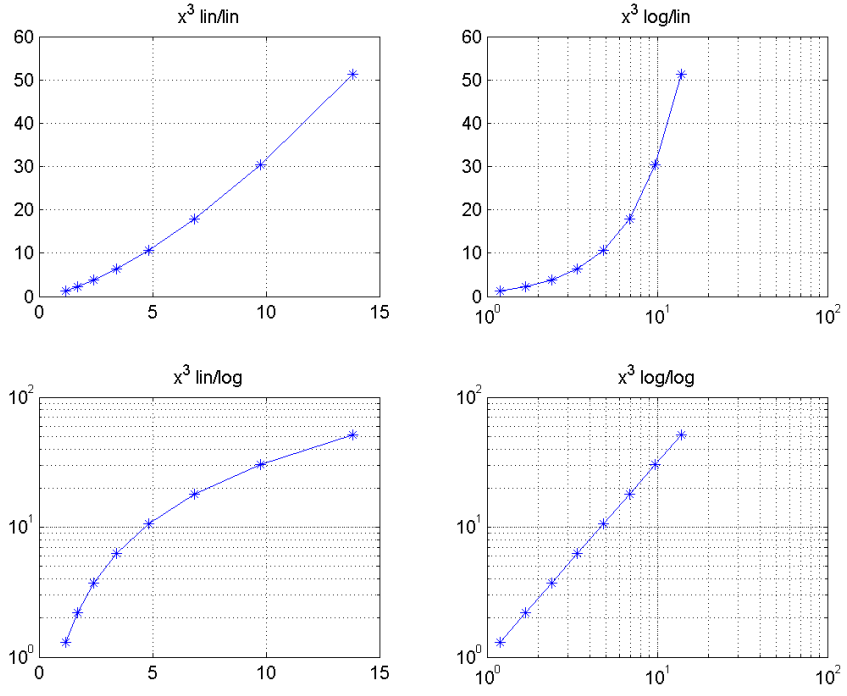


Graphen mit logarithmischer Skalierung

Oft ist es sinnvoll Graphen mit logarithmisch skalierten Achsen zu verwenden. Diese können mit `semilogx`, `semilogy` bzw. `loglog` für eine logarithmisch skalierte x-Achse, y-Achse bzw. logarithmische Skalierung beider Achsen erzeugt werden.

```
>> x= 2.4.^(0.2:.4:3.2);
>> fx = x.^(3/2)
>> plot(x,fx,'-*')
>> semilogx(x,fx,'-*')
```

```
>> x= 2.4.^(0.2:.4:3.2);
>> fx = x.^(3/2)
>> semilogy(x,fx,'-*')
>> loglog(x,fx,'-*')
```

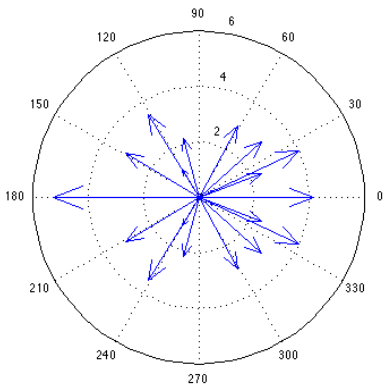


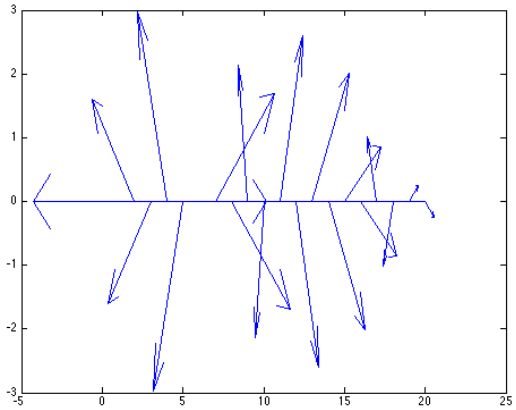
Plots komplexer Zahlen

Komplexe Zahlen lassen sich in 2D Plots veranschaulichen.

- `compass` stellt komplexe Zahlen in Polarkoordinaten dar
- `feather` stellt zweidimensionale Vektoren bzw. komplexe Zahlen als Vektoren entlang einer Geraden dar. Dabei wird bei $x = 1$ der 1. Vektor angetragen, bei $x = 2$ der 2. Vektor,...

```
>> Z=eig(randn(20));
>> compass(Z)
>> feather(Z)
```



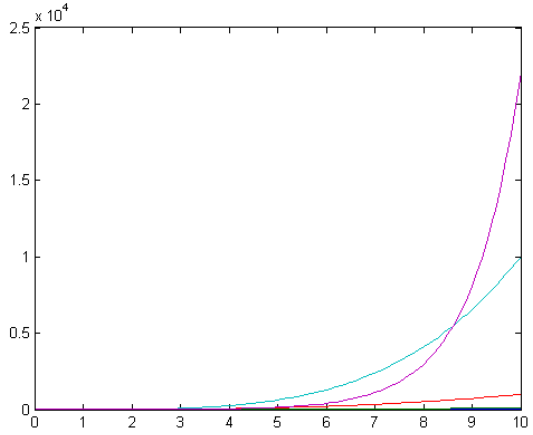


Achsengestaltung

Zur Gestaltung können bei Plots die verschiedenen Elemente der Grafik wie Achsen, Beschriftungen, Legende, Farben und später bei dreidimensionalen Plots Ansichtswinkel angepasst werden.

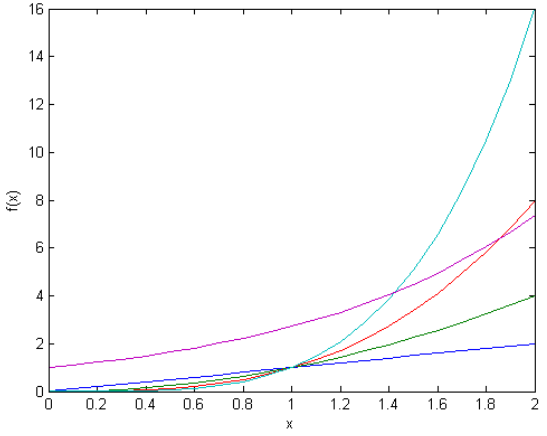
Als Beispiel nehmen wir einen Plot von verschiedenen Potenzfunktionen und der Exponentialfunktion.

```
>> x=[0:.1:10]';
>> fx=[x x.^2 x.^3 x.^4 exp(x)];
>> plot(x,fx);
```



Mit `axis` kann die Skalierung der Achsen angepasst werden. Die Beschriftung der Achsen kann mit `xlabel` und `ylabel` geändert werden.

```
>> axis([0 2 0 16])
>> xlabel('x')
>> ylabel('f(x)')
```

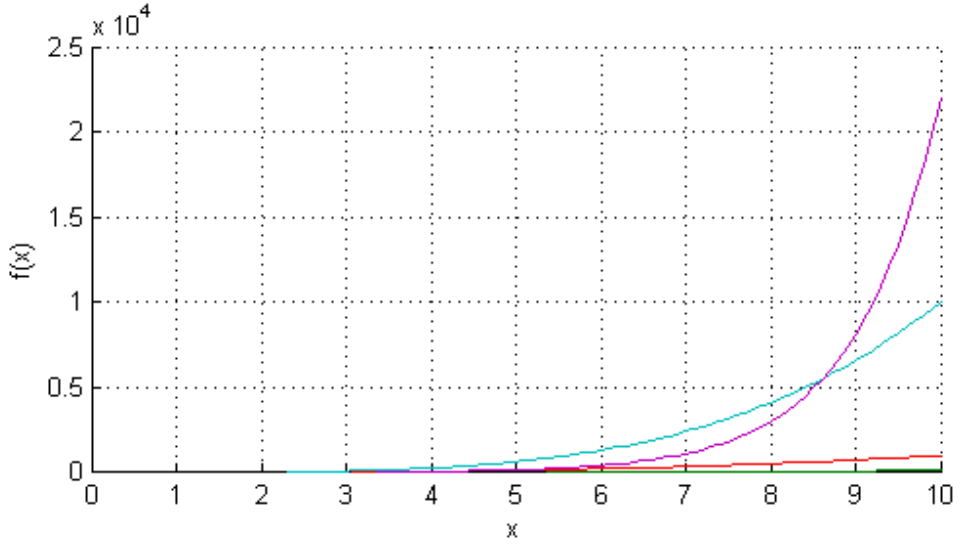


Die Grenzen der Achsen können auch mit `xlim`, `ylim` und `zlim` einzeln festgelegt werden, z.B. setzt `xlim`$([0\ 2])$ die Grenzen der x-Achse.

Achsengestaltung (cont'd)

Mit `box` kann die Umrahmung des Bildes an und ausgeschaltet werden, ebenso kann mit `axis` die Achsen angezeigt oder unterdrückt werden. `grid` stellt ein Gitter hinter dem Graphen dar.

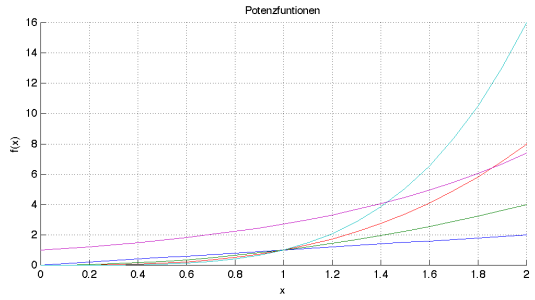
```
>> box off
>> grid on
```



Beschriftungen

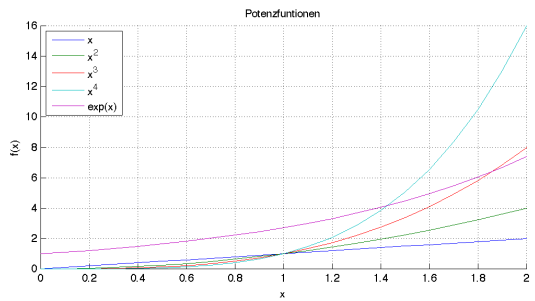
Eine Überschrift zu einem Graphen kann mit `title` festgelegt werden.

```
>> title('Potenzfuntionen')
```



Hilfreich ist es auch bei Plots mit mehreren Graphen eine Legende zu haben. Diese lässt sich mit `legend` hinzufügen. Mit der Option `'location'` kann dabei der Ort der Legende festgelegt werden (z.B. `'NW'` für North West).

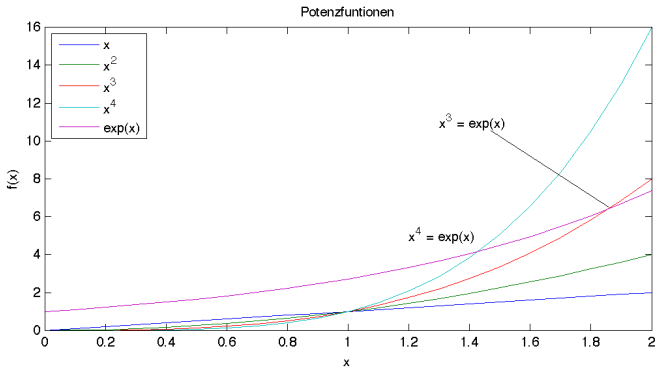
```
>> legend('x', 'x^2', ...
'x^3', 'x^4', 'exp(x)', ...
'location', 'NW')
```



Textmarken

Besondere Stellen im Graphen können mit `text` mit einem Textfeld markiert werden. Mit `annotation` können auch Linien, Pfeile und andere Graphikelemente eingefügt werden. Dabei müssen jeweils die Koordinaten angegeben werden (Vorsicht: bei annotation sind die Koordinaten auf das ganze Fenster bezogen).

```
>> text(1.2, 5, 'x^4 = exp(x)')
>> ann=annotation('line', ...
[0.85 0.7],[0.47 0.6]);
>> t2=text(1.3, 11, 'x^3 = exp(x)')
```



Als Rückgabe liefern die Funktionen jeweils ein Handle auf das erstellte Objekt. Dieses Handle kann dazu benutzt werden um Eigenschaften des Objekts zu verändern oder mit `delete` das Objekt zu löschen.

```
>> delete(t2)
>> delete(ann)
```

Aufbau eines Grafikfensters

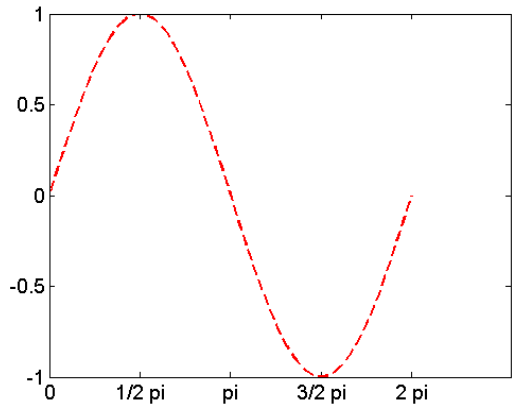
- Optionen der Grafikobjekte können mit `get` aus dem Handle ausgelesen und mit `set` gesetzt werden. Dabei gibt `get` eine Struktur mit Objekteigenschaften zurück.
- Mit `reset` können Objekteigenschaften zurückgesetzt werden;
- Grafikhandles können mit
  - `gco` für das aktive Objekt (get handle to current object),
  - `gcf` für das aktive Grafikfenster (get handle to current figure),
  - `gca` für das aktive Achsensystem und (get handle to current axis),
  - `findobj` für ein Objekt mit vorgegebenen Objekteigenschaften

  abgerufen werden.

Grafikeigenschaften Beispiel

Weitere Eigenschaften von Grafiken können mit den Funktionen get und set ausgelesen und geändert werden. Für das Auslesen und Ändern von Eigenschaften in Graphiken können jeweils Handels auf die Grafikob-
jekte verwendet werden. Diese werden beim Erstellen von Plots von den Funktionen zurückgegeben.

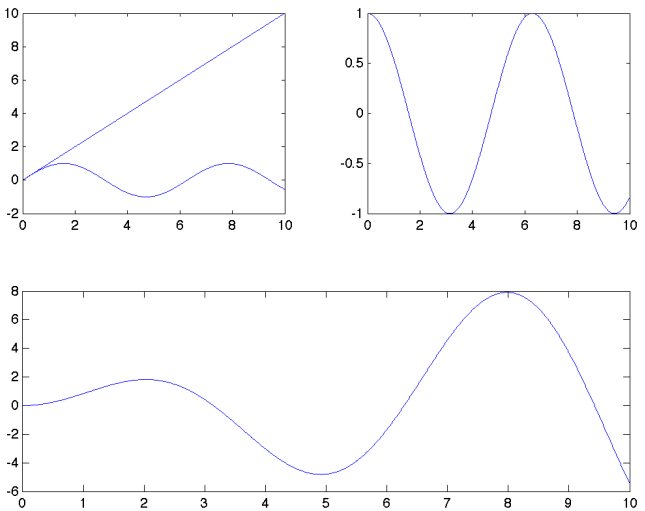
```
>> t=linspace(0,2*pi,100)
>> h=plot(t, sin(t))
>> p=get(h);
>> p.LineStyle
ans =
-
>> set(h, 'LineStyle', '--', 'Color', 'r',...
       'LineWidth', 2)
>> a=gca;
>> set(a, 'FontSize',14)
>> set(a, 'XTick', [0 pi/2, pi, 3/2*pi, 2*pi], ...
       'XTickLabel', ...
     {'0'; '1/2 pi'; 'pi'; '3/2 pi'; '2 pi'})
```



Verschiedene Graphen in einem Bild

In einem Figure Objekt können auch mehrere verschiedene Plots dargestellt werden. Dazu kann mit `subplot` der Bildbereich des Plotfensters rechteckig unterteilt werden. Die ersten beiden Werte geben dabei an in wie viele Bereiche in vertikaler bzw. horizontaler Richtung unterteilt wird. Der 3. Wert bestimmt welcher Bereich aktuell verwendet wird, wobei die Nummerierung zeilenweise erfolgt. Dabei kann beliebig zwischen den Teilfenstern hin und her gesprungen werden und auch Bereiche zusammengefasst werden. Änderungen auf Graphiken wirken sich jeweils nur auf das aktive Teilfenster aus.

```
>> x=0:.1:10;
>> subplot(2,2,1)
>> plot(x,sin(x))
>> subplot(2,2,2)
>> plot(x,cos(x))
>> subplot(2,2,[3, 4])
>> plot(x,x.*sin(x))
>> subplot(2,2,1)
>> hold on
>> plot(x, x)
>> print -dpng 'subplot.png'
```



Speichern und Drucken von Graphiken

Zum Speichern und Drucken einer Graphik steht die Funktion

- `print` ohne weitere Argumente zum drucken des aktuellen Graphikfensters mit dem Standarddrucker oder mit
- `print -d<Format> <Dateiname>` zur Ausgabe in eine Graphikdatei zur Verfügung. Gebräuchliche Grafikformate sind dabei `bmp`, `epsc`, `jpeg`, `pdf`, `png`, `tiff`.

3D Plots

Dreidimensionale Objekte können auf verschiedene Arten mathematisch ausgedrückt werden, z.B.

- als Graph einer reelwertigen Funktion von zwei Variablen,
- als parametrisierte vektorwertige Funktion,
- implizit über eine Menge die gewissen Bedingungen genügt.

Eine Möglichkeit ist, wie im zweidimensionalen, vektorwertige inline Funktionen mit `ezplot3` darzustellen.
Andere Möglichkeiten bestehen darin 3D Punktmengen zu Plotten. Hierzu gibt es die Funktionen

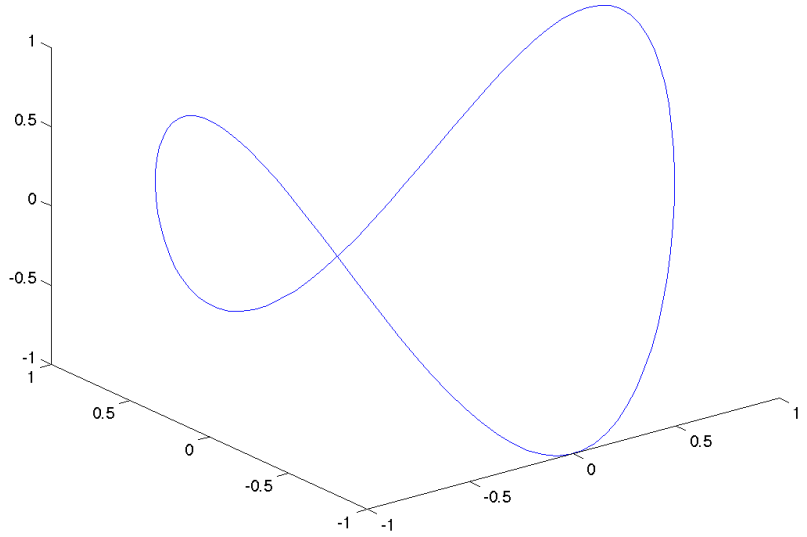
- `plot3`: Plottet räumliche Punkte und zeichnet Verbindungslinien zwischen den Punkten
- `surf`: Zeichnet eine Oberfläche durch eine Menge von 3D Punkten mit und ohne Höhenlinien.
- `mesh`: Zeichnet Punkte und verbindet die Punkte durch ein Gitter mit und ohne Höhenlinien
- `waterfall`: Zeichnet Punkte und verbindet die Punkte in einer Koordinatenrichtung.

Die Funktionen `mesh` und `surf` gibt es jeweils noch als Befehle mit angehängtem `c` und `z` bzw. angehängtem `c`, bei denen zusätzlich Höhenlinien oder vertikale Linien zu den Punkten gezeichnet werden.

3D Plot Beispiel: Parametrisierte Kurve

Zum Plotten von räumlichen Kurven $\gamma : \mathbb{R} \longrightarrow \mathbb{R}^3$ wird eine Unterteilung für den Kurvenparameter berechnet, die Koordinaten zu den Bogenparametern berechnet und die Werte mit der Funktion `plot3` dargestellt.

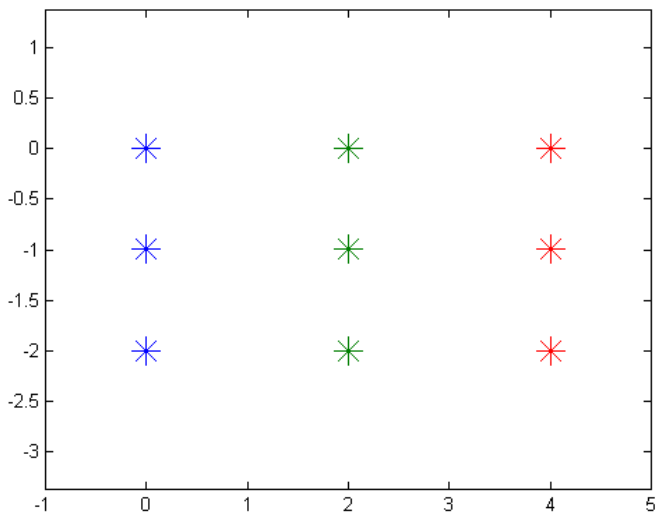
```
>> t=[0:pi/40:2*pi];
>> x=cos(t);
>> y=sin(t);
>> z=cos(2*t);
>> plot3(x,y,z);
```



Gittererzeugung

- Mit dem Befehl [X,Y]=meshgrid(x,y) können Punktegitter erzeugt werden um Oberflächen auszuwerten.
- Dazu wird mit x ein Vektor der Datenpunkte im x-Bereich und mit y ein Vektor der Datenpunkte im y-Bereich übergeben.
- Die Zeilen der Ausgabematrix X sind Kopien des Vektors x und die Spalten der Ausgabematrix Y sind Kopien des Vektors y. Damit kann jeder x-Wert mit jedem y-Wert kombiniert werden und durch plot(X,Y) als Gitter dargestellt werden.

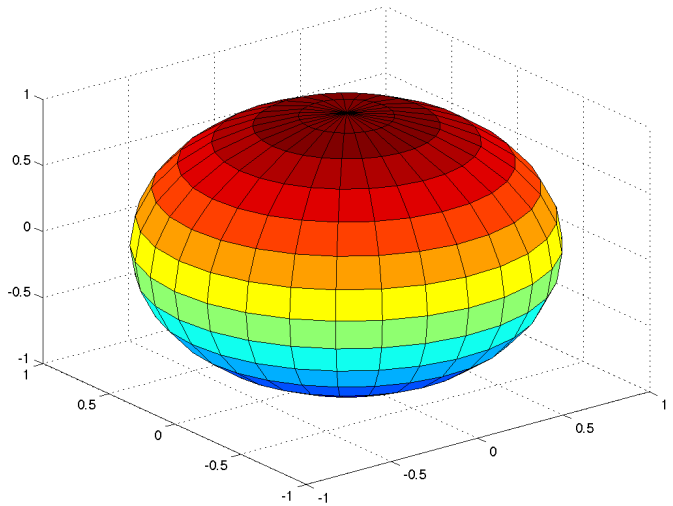
```
>> [X,Y] = meshgrid(0:2:4, -2:1:0);
>> X
X =
     0     2     4
     0     2     4
     0     2     4
>> Y
Y =
    -2    -2    -2
    -1    -1    -1
     0     0     0
>> plot(X,Y,'*')
```



3D Plot Beispiel: Parametrisierte Fläche

- Analog zur räumlichen Kurve kann auch eine parametrisierte Oberfläche $s : \mathbb{R}^2 \longrightarrow \mathbb{R}^3$ dargestellt werden.
- Hierbei wird mit `meshgrid` ein Gitter erzeugt, die Koordinaten auf dem Gitter berechnet und die Oberfläche mit `surf` dargestellt.

```
>> [s, t]=meshgrid([-pi:pi/30:pi]);
>> x=cos(s).*sin(t);
>> y=sin(s).*sin(t);
>> z=cos(t);
>> surf(x,y,z);
```

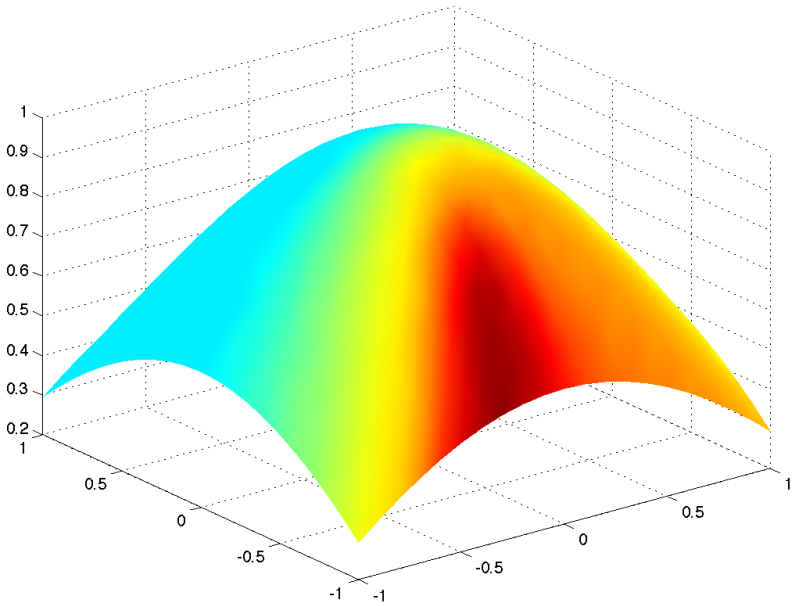


3D Plot Beispiel: Beleuchteter Graph einer Funktion

Um Funktionen $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$ darzustellen kann wiederum die Funktion `surf` verwendet werden. Im folgenden Beispiel wurde der Plotbefehl `surfl` verwendet, bei dem der Graph als schattierte Oberfläche dargestellt wird.

Mit der Funktion `shading` kann die Art der Schattierung bei 3D Plotts festgelegt werden.

```
>> [x,y]=meshgrid(-1:.1:1);
>> z=cos(x).*cos(y);
>> surfl(x,y,z);
>> shading interp;
```

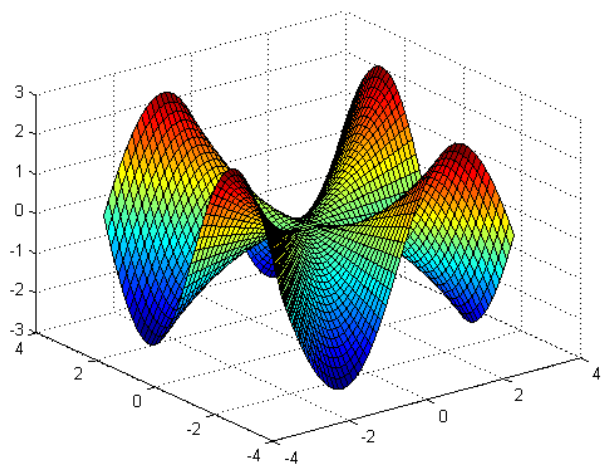


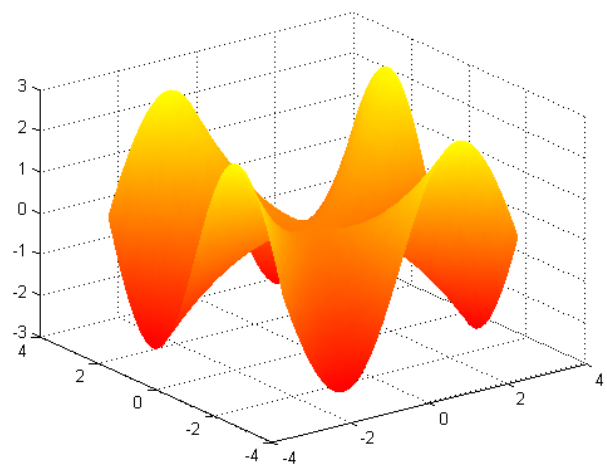
Farbgestaltung von 3D Plots

Bei Plots in 3D ist oft die farbliche Gestaltung wichtig um Details zu erkennen. Zur Änderung der Farbgestaltung können z.B. folgende Funktionen benutzt werden:

- `shading`: legt die Farbgestaltung fest
- `colormap`: legt die Farbcodierung fest
- `spinmap`: Animiert die Farbgebung eines Graphen

```
>> x=linspace(-3,3,50);
>> y=x;
>> [x,y]=meshgrid(x,y);
>> z=x.*y.*(x.^2-y.^2)./(x.^2+y.^2);
>> surf(x,y,z)
>> figure
>> shading interp
>> colormap autumn
```

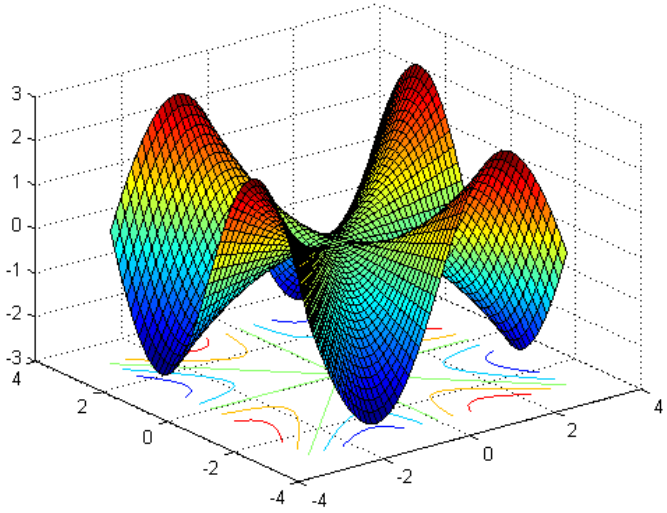


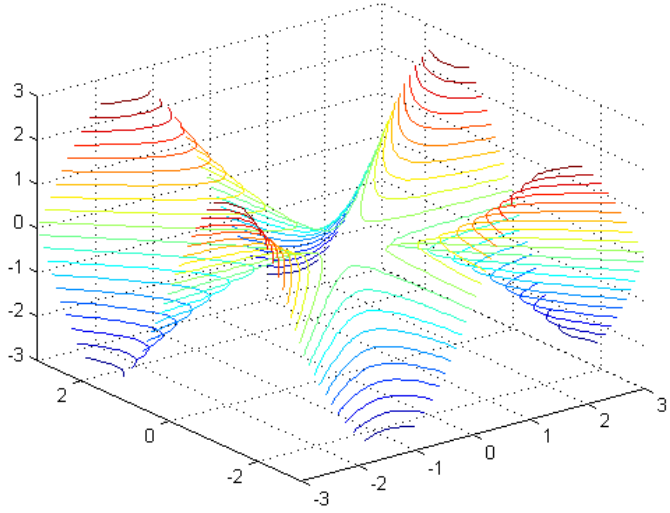


Schnittbilder

Eine weitere Möglichkeit reelwertigen Funktionen $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$ darzustellen besteht darin, Schnitte der Funktionen darzustellen, z. B. Höhenlinien bzw. farbcodierte Schnitte. Höhenlinenen können z.B. mit der Funktion `surfc` zusammen mit der Oberfläche dargestellt werden. Sollen nur Höhenlinien zu bestimmten Niveaus dargestellt werden, so kann die Funktion `contour3` verwendet werden.

```
>> x=linspace(-3,3,50);
>> y=x;
>> [x,y]=meshgrid(x,y);
>> z=x.*y.*(x.^2-y.^2)./(x.^2+y.^2);
>> surfc(x,y,z);
>> contour3(x,y,z,20);
```

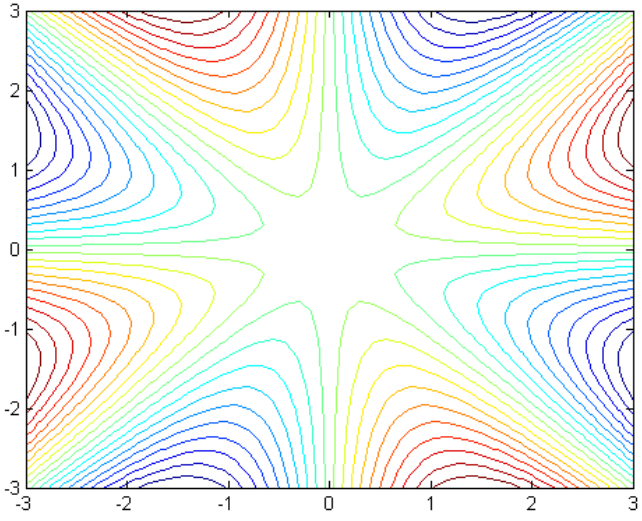


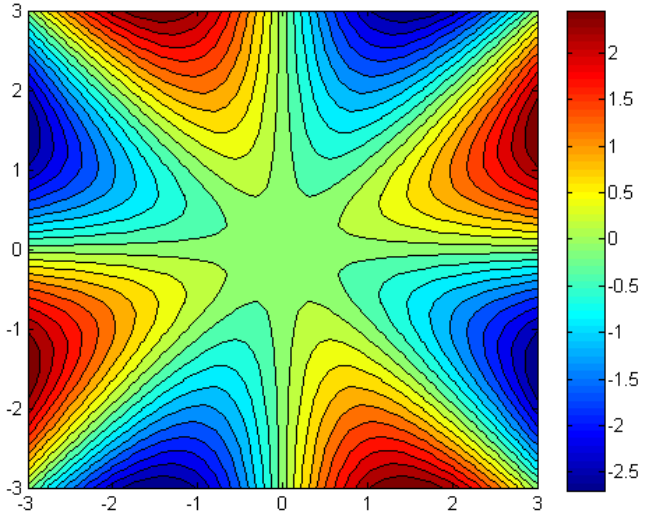


2D Schnittbilder

Sollen die Höhenlinien als 2D Bild dargestellt werden (Kartendarstellung), so können die Funktionen `contour` oder `contourf` verwendet werden. Für diese Darstellung ist es meist sinnvoll mit `colorbar` die verwendete Farbcodierung mit anzugeben.

```
>> x=linspace(-3,3,50);
>> y=x;
>> [x,y]=meshgrid(x,y);
>> contour(x,y,z,20);
>> contourf(x,y,z,20);
>> colorbar
```

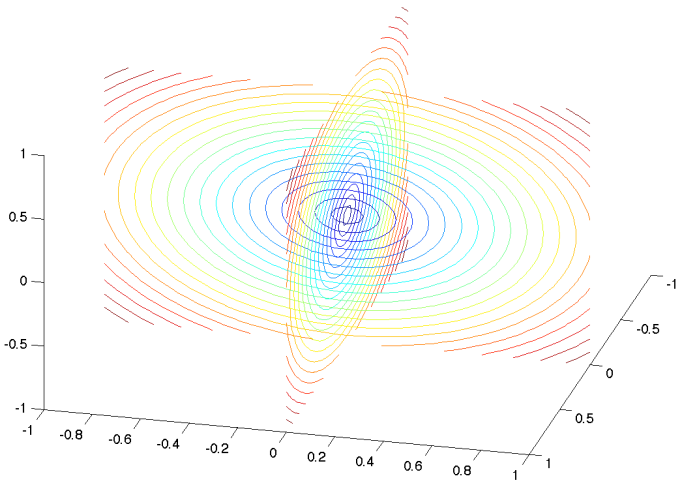


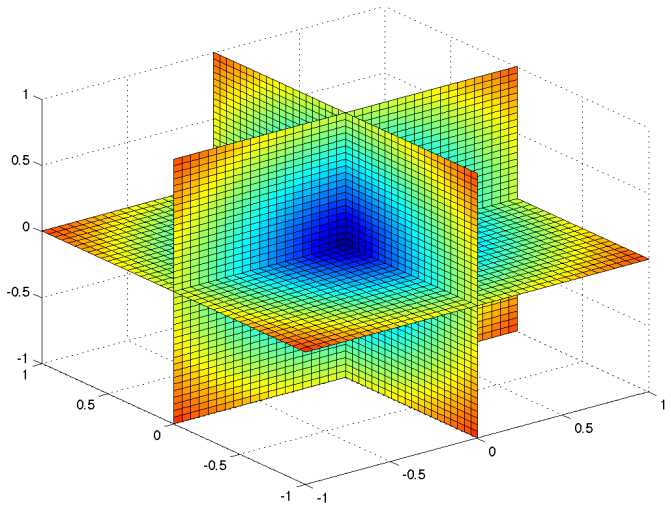


Höhenlinien auf Schnittebenen: Beispiel

Sollen Funktionen $f : \mathbb{R}^3 \longrightarrow \mathbb{R}$ dargestellt werden, so geht dies nur indem man sich Niveaus auf Flächen anschaut. Die Funktionen `slice` und `countourslice` erzeugen eine solche Darstellung. Dazu werden auf einem 3D-Gitter die Funktionswerte berechnet und zusammen mit den Achsabschnitten der Niveauflächen an die Funktion übergeben. Der Ansichtswinkel kann, wie bei den anderen Funktionen zur räumlichen Darstellung mit dem Befehl `view` festgelegt werden.

```
>> [x,y,z]=meshgrid(-1:.05:1);
>> v=sqrt(x.^2+y.^2+z.^2);
>> contourslice(x,y,z,v,0,0,[], 20);
>> view([4,1,3]);
>> slice(x,y,z,v,0,0,0);
```



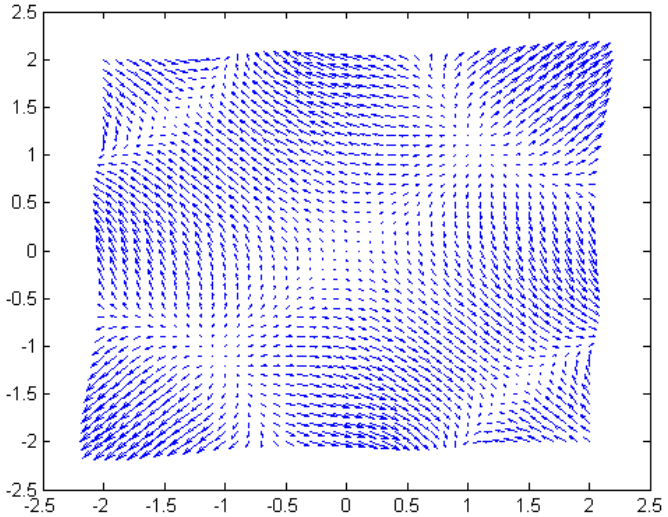


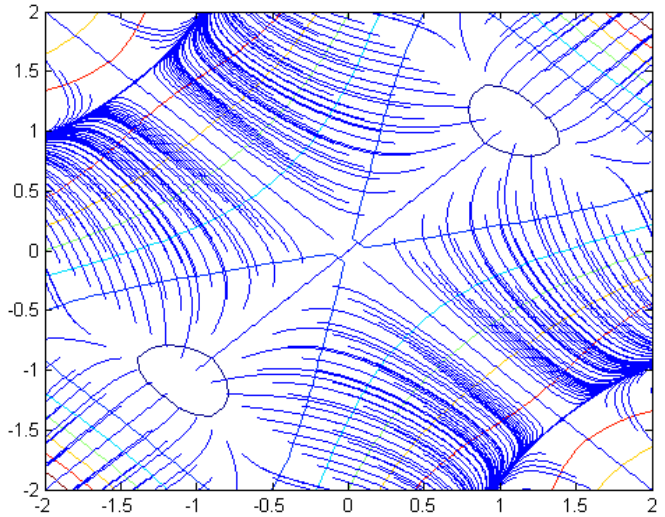
Vektorfelder in 2D

Vektorfelder $f : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ können als ebene Vektorfelder oder mit Hilfe von Stromlinien dargestellt werden. Für die Darstellung als Vektorfeld wird auf einem Gitter die Funktion ausgewertet und mit der Funktion `quiver` dargestellt.
Stromlinien können mit der Funktion `streamline` unter Angabe von Startpunkten gezeichnet werden.

```
>> [x,y]=meshgrid(-2:0.1:2);
>> z = x.^2-5*sin(x.*y)+y.^2;
>> [dx,dy]=gradient(z,0.2,0.2);
>> quiver(x, y , dx, dy, 2);
```

```
>> xs=x(1:4:end);
>> ys=y(1:4:end);
>> contour(x,y,z);
>> streamline(x,y,gx,gy,xs,ys);
```





Speichern von Variablen und IO mittels `load/save`

- Variablen eines Workspace können mit `save <Dateiname> <Variablenname>` gespeichert und mit `load <Dateiname> <Variablenname>` wieder geladen werden.
- Ebenso kann mit `load` eine Textdatei mit einer Liste von Werten als Matrix eingelesen werden.
- Mit `save <Dateiname>` und `load <Dateiname>` werden alle Variablen des Workspace gespeichert bzw. alle Variablen der Datei geladen.

```
save -ascii a.dat m              % schreibt Matrix m in Datei a.dat
                                 % (einfach genau)
save -ascii -double a.dat m      % schreibt Matrix m in Datei a.dat
                                 % (doppelt genau)

load -ascii a.dat                % belegt Matrix a mit Daten aus a.dat
m = load('-ascii', 'a.dat')      % belegt Matrix m mit Daten aus a.dat
```

fopen / fclose

- `fid = fopen('datei', 'w')` öffnet Datei und definiert fid (file identifier)
- `fprintf(fid, 'format', x, ..)` schreibt formatiert in Datei
- `fscanf(fid, 'format', x, ..)` liest formatierte Daten aus Datei
- `fclose(fid)` schliesst Datei

```
fid=fopen('daten.txt','r');
while 1
    tline = fgetl(fid);
    if ~ischar(tline), break, end
    disp(tline)
end
fclose(fid);

A = rand(2,3);
fid=fopen('daten.txt','w');
fprintf(fid,'%10.4f␣%10.4f\n',A'); % a wird transponiert
fclose(fid);
```

File-Positionierung

- Wurde eine Datei mit `fopen` geöffnet und in diese bereits geschrieben oder herausgelesen, dann lässt sich mit `frewind` die Fileposition auf den Dateianfang zurückstellen.
- `status = fseek(fid,offset,origin)` setzt die Fileposition zurück. `origin` kann die Werte `bof` (Begin Of File), `cof` (Current position Of File) oder `eof` (End Of File) haben. Betrachten wir eine Datei (`fid`) mit der ersten Zeile `ABCDEFGHIJKLM`, so würde wir mit `fseek(fid,4)` auf `D` rücken.
- `ftell(fid)` gibt die aktuelle Position in der Datei wieder.

String-Verarbeitung

- `strfind('text','muster')` findet Strings `muster` in einem anderen String `text`

```
S = 'Find␣the␣starting␣indices␣of␣the␣pattern␣string';
strfind(S, 'in')
ans =
     2    15    19    45
strfind('in', S)
ans =
     []
```

- `findstr('text','muster')` findet Strings `muster` in einem anderen String `text`

```
s = 'How␣much␣wood␣would␣a␣woodchuck␣chuck?';
findstr(s,'a')
ans =
     21
findstr('a',s)
ans =
     21
```

- `strcmp(s1,s2)` liefert den Wert 1, falls s1 und s2 identisch sind, ansonsten 0.

Excel-Dateien

- `xlsread` liest Daten von Excel-Dateien aus.
  Beispiel:

```
A = xlsread('testdata2.xls', 1, 'A4:B5')

A =
    4     9
    5   NaN
```

- `xlswrite` schreibt Daten in eine Excel-Datei.
  Beispiel:

```
xlswrite('testdata', [12.7 5.02 -98 63.9 0 -.2 56])
% Fängt bei A1 an (hier bis G1)

d = {'Time', 'Temp'; 12 98; 13 99; 14 97};
xlswrite('tempdata.xls', d, 'Temperatures', 'E1')
```

Internet

- Mit `s=urlread('url')` greift man auf die URL-Adresse zu und liest deren Inhalt in den string `s` ein.
  Beispiel:

```
s = urlread('http://www.mathworks.com')
```

- Mit `sendmail('empf','sub','nachricht','anhang')` können an die Email-Adresse empf mit dem Betreff `sub` die Informationen `nachricht` und ggf. ein Anhang `anhang` gesandt werden.
  Bevor eine E-mail versandt werden kann, müssen zunächst die Präferenzen entsprechend dem vorliegenden E-Mail-System mit `setpref` gesetzt werden.

  Beispiel:

```
setpref('Internet','SMTP_Server','myserver.myhost.com');
setpref('Internet','E_mail','myaddress@example.com');
```

  Man beachte, die `sendmail` Funktion unterstützt keine Email-Server, die Kennwortauthentifizierung (SPA) erfordern!