# Programmierpraktikum im WS 2009/2010

Denny Otten, V5-134, dotten@math.uni-bielefeld.de

Jens Rottmann–Matthes, V5-142, jrottman@math.uni-bielefeld.de

## 5. Grafik

### Zeichnen in 2D

Du hast auf den bisherigen Zetteln schon einige Methoden zum Zeichnen von Funktionen $\mathbb{R} \to \mathbb{R}$ kennengelernt. Die einfachsten sind `ezplot` und `plot` ein weiterer einfacher Befehl ist `fplot` (siehe `help fplot`). Zunächst schauen wir uns noch einmal den Befehl `ezplot` (amerikanische Aussprache: easy-plot) an:

```
>> f=@(x) abs(x).*sin(x);             % vektorisierte, anonyme Funktion erstellen
>> ezplot(f);                         % und zeichnen;
>> ezplot('abs(x)*sin(x)', [-1, 10]) % Formelstring direkt eingegeben
                                      % und y-Achse auf [-1,10] eingeschraenkt;
>> ezplot(@(x) sin(1./x), [0.01 0.1]) % anonyme Funktion direkt eingegeben;
```

Hierbei ist es wichtig in der ersten und dritten Version die Funktion zu vektorisieren (siehe Übungszettel 3), da dann die Funktion mit `ezplot` wesentlich schneller gezeichnet werden kann.

### Parametrische und implizite Plots

Mit `ezplot` ist es auch möglich parametrische Kurven zu zeichnen, d.h. Kurven $\mathbb{R} \ni t \mapsto (x(t), y(t)) \in \mathbb{R}^2$.

```
>> ezplot(@sin,@cos, [0,2*pi])              % ein Kreis
>> ezplot('2*sin(x)','2*cos(x)', [-pi,pi]) % und noch einer
```

Diese Kurven können auch implizit durch eine Gleichung, etwa $x^2 + y^2 = 1$, gegeben sein.

```
>> ezplot('x^2+y^2-1', [-2, 2, -2, 2])      % noch ein Kreis
>> ezplot(@(x,y) x.^2+3*y.^3-1,[-2 2 -2 2]) % und zur Abwechslung ein Tafelberg
```

**Aufgabe 1.** *Zeichne eine Ellipse mit dem Durchmesser 2 auf der x-Achse und dem Durchmesser 1 auf der y-Achse mit* `ezplot`.

Die Benutzung von `ezplot` hat den Nachteil, dass sich nur wenige Einstellung vornehmen lassen: die Einteilung in $x$ Richtung liegt fest und der Linienstil sowie die Farbe sind nur nachträglich änderbar. Wesentlich flexibler ist die Funktion `plot`, die Du bereits kennst. Sie zeichnet Punkte $(x, y)$ und verbindet sie (gegebenenfalls) der Reihe nach mit Linien. Um Funktionen mit `plot` visualisieren zu können, müssen also "Wertetabellen" erzeugt werden.

```
>> ptx = [0 0 1 0.5 0 1 1 0 1];    % x-Koordinaten
>> pty = [0 1 1 1.5 1 0 1 0 0];    % y-Koordinaten
>> plot(ptx,pty,'ro-','LineWidth',4); title('Haus vom Nikolaus')
>> t = -4:0.1:4;                   % x-Werte erstellen (Alternative: linspace)
>> y = t.^2+sin(t)./sqrt(t);       % y-Werte (y=f(x)) erstellen
>> plot(t,y, 'r+-','Markersize',5) % Wertepaare (x_i,y_i) zeichnen und verbinden
>> grid on                         % Gitter einblenden
>> x=cos(t); y = sin(t);
>> plot(x, y, 'k:')                % Kreis zeichnen
>> axis equal                      % damit es auch wie ein Kreis aussieht
```

**Aufgabe 2.**

- *Was bedeutet die Warnung, die Du beim Zeichnen von* `y=t.^2+sin(t)./sqrt(t)` *in obigem Beispiel bekommen hast?*
- *Zeichne den Graph der Funktion* $r(x) = \begin{cases} 1, & |x| < 0.5 \\ 0, & \text{sonst} \end{cases}$ *im Intervall* $[-3, 3]$.
- *Erstelle anonyme Funktionen* $x(t) = \cos(3t)\cos(t)$, $y(t) = \cos(3t)\sin(t)$ *und zeichne die Kurve* $(x(t), y(t))$ *für* $t \in [0, 2\pi]$ *mit* `ezplot` *und* `plot`.

### Achsenskalierungen

Manche Beziehungen lassen sich besser mit logarithmischen Skalen erkennen, dazu kann man entweder die Skalierung der Achsen in einem Plot nachträglich über die Eigenschaften `xscale, yscale, zscale` ändern, oder die Grafik gleich mit `semilogx, semilogy` oder `loglog` erstellen.

```
>> x = linspace(1,20,39); y = 2*sqrt(x);       % "Wertetabellie" fuer die Wurzel
>> plot(x,y); grid on;                         % Werte mit linearer Skalierung plotten
>> set(gca,'xscale','log')                     % x-Achse logarithmisch setzen
>> semilogx(x,y, 'r')                          % zeichnet gleich mit dieser Skalierung
>> set(gca,'yscale','log', 'xscale', 'linear') % nur y-Achse logarithmisch setzen
>> grid on;                                    % Gitter einblenden
>> set(gca,'xscale','log')                     % beide Achsen logarithmisch setzen
>> loglog(x,y,'r')                             % tut das gleiche direkt beim plotten
```

### Plots beschriften

Mit den Funktionen `title`, `xlabel`, `ylabel`, `zlabel` und `legend` können Beschriftungen (nachträglich) zum Plot hinzugefügt werden.

```
>> x = linspace(1,20,39); y = [2*sqrt(x);sin(x)];
>> plot(x,y)
>> title('Achsenueberschrift mit \lambda und \pi')
>> xlabel('x - Achse'), ylabel({'y - Achse','zweizeilig'})
>> legend({'2 x^{1/2}', 'Kurve 2'})  % Legende einfuegen (erlaubt einfachstes TeX)
```

Dasselbe ist auch per Maus über die Menüs im Grafikfenster möglich (siehe insbesondere das Menü `Insert`). Matlab bietet noch viele weitere Möglichkeiten, einen Plot zu kontrollieren – ähnlich wie Funktionshandles gibt es auch Grafikhandles, die Zugriffsnummern auf Grafikobjekte wie `figure`, `axis` etc. sind. Ein erstes Beispiel hast Du bei `set(gca,'xscale','log')` kennengelernt. Es ist nämlich `gca` das Handle auf die aktive Achse. Mit `gcf` bekommt man das Handle auf die aktuelle Figur.
Mit `figure` lässt sich ein neues Grafikfenster erstellen und das Handle darauf zurückgeben. Die Eigenschaften des `figure` Objekts lassen sich mit `get` lesen und mit `set` verändern. Mit dem Befehl `shg` (show graph window) kann das aktive Grafikfenster in den Vordergrund geholt werden, wenn es verborgen ist.

```
>> hfig1 = gcf  % hfig1 ist der Handle auf die aktuelle Figur
                % falls keine existiert, wird eine erzeugt
>> hfig2 = figure('Name', 'Mein Titel', 'position',[50,100,400,200])
                % erzeugt ein Fenster in x=50,y=100 mit Breite 400, Hoehe 200
                % x,y sind die Bildschirm - Koordinaten!
>> set(hfig2, 'Name','Neuer Titel')   % anderen Titel setzen
>> set(hfig1, 'Name','Altes Fenster') % auch das alte koennen wir noch aendern
>> figure(1)                          % holt das 1. Fenster in den Vordergrund
>> plot([0],[0], '+','Markersize',8)  % zeichnet etwas darauf
>> cla                                % und loescht es gleich wieder (siehe auch: clf)
```

Innerhalb eines Grafikfensters kann es eine oder mehrere Achsenobjekte geben, deren Eigenschaften man ebenfalls mit `get` bzw. `set` lesen und bearbeiten kann.

```
>> hplot = plot(x,y)           % der plot Befehl erzeugt (falls noetig) eine neue Achse
                               % und gibt ein Handle auf den Plot zurueck
>> set(hplot, 'Marker', 'x');  % Eigenschaft des Objektes hplot aendern
>> hax = gca                   % Handle auf die aktuelle Achse (gca = get current axis)
>> get(hplot,'Parent')         % das kann man auch aus der Eigenschaft parent
                               % des hplot Objektes  herauslesen
>> set(hax, 'xlim', [-2 2], 'ylim', [-9, 9]) % x und y-Bereich aendern
```

**Aufgabe 3.** *Ändere Eigenschaften des Plots (Achsenbeschriftung/Font, Überschriften, Legende etc.) mit Hilfe der Funktionen im Menü des Grafikfensters.*
*(**Zusatz:** Versuche das gleiche per Code durch Setzen der entsprechenden Eigenschaften mit* `set`*. Verschaffe Dir dazu einen groben Überblick über die verschiedenen Grafikobjekte (siehe* `>> helpwin` *und dann unter* `MATAB->User Guide->Graphics->Handle Graphics Objects`*). Experimentiere mit den Eigenschaften von Grafikfenstern und Achsen (suche in der Online Hilfe nach:* `axes properties` *und* `figure properties`*).)*

**Aufgabe 4.** *Experimentiere mit den verschiedenen Optionen für* `axis` *(*`help axis`*).*

### Mehrere Plots in eine Achse und mehrere Achsen in eine Figur

Es lassen sich auch mehrere Funktionen direkt mit `plot` in ein Fenster zeichnen

```
>> x = linspace(0,pi,101);
>> plot(x, sin(x), x, sin(2*x), x, sin(3*x))  % Methode 1
>> figure                                     % neues Grafikfenster erzeugen
>> plot(x, [sin(x); sin(2*x); sin(3*x)])      % Methode 2
```

Mehrere Plots lassen sich mit `hold on` nacheinander in ein Grafikfenster zeichnen ohne die vorherigen zu löschen (siehe: `help hold`).

```
>> hold on                          % Zeichne die folgende Grafik hinzu
>> plot(x, sin(x).*cos(x), 'k:')
>> hold off                         % Grafik mit der naechsten Zeichnung ersetzen
>> plot(x, abs(10-x), 'm')
```

**Aufgabe 5** (**Zusatz-/Weihnachtsaufgabe**)**.** *Lade die Daten aus*
`www.math.uni-bielefeld.de/~jrottman/programmierpraktikum/rt.txt` *mit* `uiimport` *und zeichne die Spalten in einen Plot. Je zwei Spalten sind einmal x- und einmal y-Koordinaten. (Wichtig: benutze* `hold on`*!)*

Wie Du schon vorher gesehen hast, kannst Du mehrere Achsen mit `subplot` in eine Figur (ein Fenster) zeichnen.

```
>> x = 0:0.2:5; y = 2*x.^3;
>> h1 = subplot(2, 1, 1)     % 1. Subplot von 2 Plots in 2 Zeilen, 1 Spalte
>> h2 = subplot(2, 1, 2)     % 2. Subplot von 2 Plots in 2 Zeilen, 1 Spalte
>> plot(x, y, 'c+-')         % auf aktuelle Achse plotten
>> axes(h1)                  % waehle Achse h1
>> plot(y, x.^2)             % auf Achse h1 plotten
```

**Aufgabe 6.** *Schreibe ein Matlab-Skript, das folgende Aufgabe erledigt:*
*Es wird eine zufällige* **symmetrische** *Matrix $A \in \mathbb{R}^{6,6}$ erzeugt und dann der Graph des charakteristischen Polynoms $\xi_A(\lambda) = \det(A - \lambda I)$ in einem geeigneten Intervall gezeichnet.*
*Ferner sollen die Eigenwerte $\lambda_i$ der Matrix $A$ mit* `lm = eig(A)` *berechnet, und die Punkte $(\lambda_i, 0)$ mit $*$ mit in den vorherigen Plot eingetragen werden.*

### Komplexe Werte in 2d mit `plot` darstellen

Falls man die Funktion `plot` nur mit einem Argument aufruft (also `plot(x)`), so hängt das Verhalten der Funktion davon ab, was für Daten übergeben wurden. Sind die Daten reellwertig, so zeichnet `plot` den Wert gegen den Index
`>> x=(5:-1:0).^2;`
`>> plot(x);` entspricht `plot(1:length(x),x)`.
Falls es sich bei `x` um eine Matrix handelt, so wird jede Spalte gegen den Spaltenindex gezeichnet.
Handelt es sich dagegen um komplexwertige Daten, so wird $\mathbb{R}^2$ als $\mathbb{C}$ interpretiert und der Befehl entspricht `plot(Re(x),Im(x))`

```
>> x=exp(1i*(1:10).*pi./5); plot(x,'.');
>> title('Komplexe 10te Einheitswurzeln');
```

## Zeichnen in 3D

### Kurven in $\mathbb{R}^3$

Auch für das Zeichnen in drei Dimensionen gibt es einen einfachen Zeichenbefehl: `ezplot3`. Dieser funktioniert ganz ähnlich wie der entsprechende Befehl `ezplot`. Er ist jedoch nur für das Zeichnen parametrischer Kurven geeignet:

```
>> f_x=@(t)2*t; f_y=@(t)sin(t); f_z=@(t)t.^2;
>> ezplot3(f_x,f_y,f_z);                % standardmaessig 0<t<2*pi
>> ezplot3(f_x,f_y,f_z,[-pi,pi]);       % jetzt -pi<t<pi
>> ezplot3('t','t^(1/2)','t^2',[0,1]); % wie fuer ezplot auch mit Strings
```

Wie auch schon im 2D Fall ist dieser Befehl sehr einfach, man kann aber bei weitem nicht soviele Einstellungen vornehmen, wie mit dem etwas komplizierteren `plot3`–Befehl. Dieser ist das 3d-Äquivalent zum `plot`-Befehl und wird auch genau so aufgerufen. Es werden also Datenpunkte gezeichnet und (eventuell) mit Linien verbunden.

```
>> figure;
>> x = rand(5,1), y=rand(5,1), z=rand(5,1);
>> plot3(x,y,z,'r-p');
>> xlabel('x'); ylabel('y'); zlabel('z');
>> grid on;
>> box on;
```

Entsprechend einfach ist es Kurven in 3d mit `plot3` zu zeichnen, die in Parameterdarstellung

$$I \ni t \mapsto (x(t), y(t), z(t)) \in \mathbb{R}^3$$

vorliegen.

**Aufgabe 7.** *Zeichne die folgenden in Parameterdarstellung gegebenen Raumkurven jeweils zusammen mit ihren drei Projektionen in die x–y, x–z und y–z Ebene. (Eventuell bietet es sich an* `deal` *zu benutzen.)*

1. $t \mapsto (t, t^2, t^3), \quad t \in [-1, 1]$
2. $t \mapsto ((t/\pi)^2 \sin(20t), t^2, t), \quad t \in [0, \pi]$
3. $t \mapsto (\sin(t), \sin(t)\cos(t), t), \quad t \in [0, 2\pi]$

### Animation von Kurven

Eine nette Fähigkeit von `ezplot3` ist die Möglichkeit Kurven zu animieren. Die Kurve

$$x(t) = (1 + t^2)\sin(2t),\ y(t) = (1 - t^2)\cos(2t),\ z(t) = t, \quad t \in [-10, 10]$$

kann einfach durch

```
>> ezplot3('(1+t.^2)*sin(2*t)','(1-t.^2)*cos(2*t)','t',[-10,10], 'animate')
```

mit einer Animation des Kurvendurchlaufs gezeichnet werden. Durch Klick auf `Repeat` kannst Du die Animation wiederholen.

**Aufgabe 8.** *Zeichne die folgenden ebenen Kurven*

$$x(t) = \cos(4t)\cos(2t), \quad y(t) = \cos(3t)\sin(t), \quad t \in [0, 2\pi]$$

*sowie die Zykloide*

$$x(t) = t + r\sin(-t), \quad y(t) = 1 - r\cos(-t), \quad t \in [0, 4\pi]$$

*für verschiedene r (z. B. $r = 1/2$, $r = 1$ und $r = 3$) mit* `ezplot3` *mit der Option* `'animate'` *in eine Ebene.*

Ein anderer Befehl um Animationen zu erzeugen ist `comet` in 2D und `comet3` in 3D. Wie `plot` zeichnet auch `comet` nicht Funktionen sondern Datenpaare (oder Datenvektoren). Zum Beispiel für einen Kreis

```
>> t=0:0.001:2*pi; x=sin(t); y=cos(t);
>> comet(x,y);
```

**Aufgabe 9.** *Zeichne für verschiedene $r$ die Epizykloiden $(r\cos(t)-\cos(rt), r\sin(t)-\sin(rt))$ für $t \in [0, 2*pi]$ mit* `comet`

Eine Animation wie mit der Option `'animate'` von `ezplot3` kannst Du auch selbst erstellen, indem Du die einzelnen Bilder (frames) in einer Schleife hintereinander plottest. Wichtig ist der Befehl `drawnow`, der veranlasst, dass das erstellte Bild angezeigt wird. Am Besten speicherst Du das folgende in einem Matlab-Skript.

```
N=input('Wieviele Stuetzstellen? ')
figure; shg;
x=linspace(0,2*pi,N); y=1./x.*sin(x.^2);
for j = 1:N
   plot(x,y,'k');       % Funktion zeichnen
   hold on;
   plot(x(j),y(j),'ro','Markerfacecolor','red');    % Marke zeichnen
   hold off;
   drawnow;
end
```

Wesentlich effizienter ist es nicht die ganze Grafik, sondern nur die Position des Markers zu verändern, indem man die Eigenschaften `xdata` und `ydata` des Linienobjektes, dass den Marker beschreibt, verändert:

```
N = input('Wieviele Stuetzstellen? ');
figure;
x=linspace(0,2*pi,N); y=1./x.*sin(x.^2);
plot(x,y,'k'); hold on;
h=plot(x(1),y(1),'ro','Markerfacecolor','red');
for j = 2:N
   set(h,'xdata',x(j),'ydata',y(j));
   drawnow;
end
```

**Aufgabe 10** (Zusatzaufgabe)**.** *Vergleiche mit Hilfe des Profilers die unterschiedlichen Aufwände der beiden oben vorgestellten Möglichkeiten eine Animation zu erstellen.*

**Aufgabe 11.** *Erstelle eine Funktion* `ezanim`*, die eine parametrische Funktion $(x(t), y(t)), t \in I$ nach Eingabe der Funktionen $x(\cdot)$, $y(\cdot)$ und des Intervalls $I$ plottet und eine Animation wie oben anzeigt.*

**Aufgabe 12.** *Ein Vektor $(x, y) \in \mathbb{R}^2$ lässt sich um den Winkel $\phi$ drehen, indem die Drehmatrix*

$$R_\phi = \begin{pmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix}$$

*mit ihm multipliziert wird. Auf diese Weise lassen sich graphische Objekte, wie z.B. Quadrate drehen, indem alle ihre Ecken gedreht werden. Erstelle eine Animation in der das Haus vom Nikolaus von Seite 1 einmal um sich selbst gedreht wird.*
*Versuche es auch mit der Methode das Grafikobjekt nur zu ändern und nicht neu zu erzeugen. Ist dies schneller?*

### Flächen in $\mathbb{R}^3$

Flächen, die in Parameterdarstellung gegeben sind, d.h. die als Abbildung $f : \mathbb{R}^2 \to \mathbb{R}$ vorliegen, kann man mit `mesh`, `surf`, und in manchen Fällen auch besser mit `trimesh` zeichnen.
Das Vorgehen ist dabei stets dasselbe, man muss zunächst eine "Wertetabelle" erzeugen, also jedem Paar von $(x, y)$-Werten einen $z$-Wert zuordnen. Da man häufig die $x$-Achse und die $y$-Achse in gewisse Abschnitte eingeteilt hat, muss man zunächst das Gitter der $(x, y)$-Paare erzeugen. Dieses liefert ein strukturiertes Gitter. Erst dann kann man die Wertetabelle ausfüllen, dafür ist es sinnvoll eine Funktion in vektorisierter Form vorliegen zu haben, da man dann sehr einfach auswerten kann. Ein einfaches Beispiel:

```
>> clear;
>> f = @(x,y)x.^3-3*x.*y.^2; % ein sogenannter Affensattel soll gezeichnet werden
>> x = -1:0.2:1;            % Wir moechten fuer jedes x aus -1, -0.8, -0.6, ..., 0.8, 1
>> y = -1:0.2:1;            % und jedes y aus der selben Menge auswerten
>> [X,Y] = meshgrid(x,y);   % Gitter (X(i),Y(i)) ist die Menge aller Gitterpunkte
>> Z = f(X,Y);              % hier wichtig, dass f vektorisiert ist:
                            % Z(i) = f(X(i),Y(i)), fuer alle i
>> mesh(X,Y,Z);             % Zeichnet die Funktion auch x,y statt X,Y erlaubt
>> meshc(x,y,Z);            % mit Konturlinien (hier x,y statt X,Y benutzt)
>> meshz(x,y,Z);            % mit Vorhang
>> hs = surf(X,Y,Z);        % wirklich eine Oberflaeche
>> set(hs,'edgecolor','none');  % bitte ohne Kanten
>> shading interp           % interpoliert den Farbverlauf
>> shading flat             % stellt die urspruenglichen Farben wieder her
```

**Aufgabe 13.** *Schaue Dir die Online-Hilfe zu den Funktionen* `mesh`, `surf` *und ihren Varianten an. Starte auch die Demo* `graf3d` *um die verschiedenen Optionen von* `mesh` *und* `surf` *kennenzulernen.*

Die Konturen, die mit `meshc` zusätzlich zum Gitter erzeugt werden, können auch mit `contour`, `contour3`, oder `contourf` einzeln erstellt werden (siehe `help contour`).

```
>> f=@(x,y)sin(x.*y)+exp(-(x.^2+1/2*y.^2));
>> x=linspace(-pi,pi,200); y=x;         % Ein Rechtecksgitter
>> [X,Y]=meshgrid(x,y);                 % wird erzeugt
>> Z=f(X,Y);                            % die Funktion an den Gitterpunkten
>> subplot(2,2,1), surf(X,Y,Z), shading flat
>> axis tight, colorbar
>> subplot(2,2,2), contour3(X,Y,Z,6), colorbar
>> subplot(2,2,3), contour(X,Y,Z), colorbar
>> subplot(2,2,4), contourf(X,Y,Z,[-1:0.5:1]), colorbar
```

**Aufgabe 14** (Zusatz)**.** *Zeichne das Polynom* $p(z) = z^3 - 2z^2 - 2z - 7$ *in* $\mathbb{C}$ *mit* `mesh` *oder* `surf`*. Was kannst Du sinnvollerweise darstellen? Wie kann man die komplexen Nullstellen sehen?*

Falls man ein unstrukturiertes Gitter hat, d.h. die Punkte, an denen die Funktion $f$ ausgewertet wird liegen nicht in einem Rechtecksgitter vor, so muss man vor dem Zeichnen noch eine Gitterstruktur erzeugen. Dies geht mit der Funktion `delaunay`, die aus den ungeordneten Punkten ein Dreiecksgitter erzeugt:

```
>> clf
>> x = 2*rand(100,1)-1;     % zufaellige Stuetzstellen x und y erzeugen
>> y = 2*rand(100,1)-1;
>> z = x.^3-y.^2;           % wollen x^3-y^2 zeichnen
>> plot3(x,y,z);            % dieses bringt gar nichts
>> plot3(x,y,z,'kx');       % wenn man nur die Punkte zeichnet, wird es besser
>> mesh(x,y,z);             % mesh kann mit solchen Daten gar nicht umgehen
>> tri = delaunay(x,y);     % baut fuer die unstrukturierten Daten ein Dreiecksgitter
>> plot(x,y,'ro'); hold on  % Zeichne Datenpunkte (2d)
>> triplot(tri,x,y);        % Zeichne ein Dreiecksgitter hinzu
```

```
>> hold off
>> trimesh(tri,x,y,z);      % damit ist es jetzt auch in 3d moeglich
>> trisurf(tri,x,y,z);      % anstelle eines Gitters eine Oberflaeche
```

**Aufgabe 15** (Zusatzaufgabe)**.** *Versuche mit der Onlinehilfe zu verstehen, was* `delaunay` *macht. Hilfreich kann auch die Grafik sein, die der folgende Code liefert:*

```
>> x = rand(12,1); y = rand(12,1);
>> tri = delaunay(x,y)
>> plot(x,y,'k.','MarkerSize',12), hold on
>> for i = 1:12
    text(x(i), y(i), num2str(i));
  end
>> tri(4,:)
>> plot(x(tri(4,[1:end,1])),y(tri(4,[1:end,1])),'r-o','MarkerSize',22)
```

### Rotationskörper

Ein Rotationskörper mit dem Profil $r(z)$ wird durch die Parameterdarstellung von $(x, y, z)$ mit

$$x(z,\varphi) = r(z)\sin(\varphi), y(z,\varphi) = r(z)\cos(\varphi), z = z$$

beschrieben:

```
>> phi=linspace(0,2*pi,180);        % Winkel
>> z = -1:0.1:1;
>> r = @(z)cosh(z);
>> x = @(z,phi) r(z).*sin(phi);
>> y = @(z,phi) r(z).*cos(phi);
>> [Z,Phi]=meshgrid(z,phi);
>> surf(x(Z,Phi),y(Z,Phi),Z);
```

**Aufgabe 16.** *Zeichne einen Rotationskörper mit dem Profil $r(z) = 2 + \cos(z)$ im Bereich $z = [0, 6\pi]$. Zeichne einen Doppelkegel im Bereich $z \in [-1, 1]$.*

Wenn man als dritte Koordinate nicht den Wert $z$, sondern den Winkel $\varphi$ zeichnet, so erhält man statt Rotationskörpern Wendelflächen (Helikoide):

```
>> surf(Z.*sin(Phi),Z.*cos(Phi),Phi);
```

**Aufgabe 17.** *Zeichne eine Wendelfläche mit innerem Radius 1 und äußerem Radius 2, die sich dreimal um die z-Achse windet.*

Es ist auch nicht schwer ein Möbiusband zu zeichnen:

```
>> x=@(u,v)(1+1/2.*v.*cos(1/2*u)).*cos(u);
>> y=@(u,v)(1+1/2.*v.*cos(1/2*u)).*sin(u);
>> z=@(u,v)1/2*v.*sin(1/2*u);
>> u=linspace(0,2*pi,180);v=-0.5:0.05:0.5;
>> [u,v]=meshgrid(u,v);
>> surf(x(u,v),y(u,v),z(u,v));
```

**Aufgabe 18.** *Ein Torus wird mit Hilfe der Winkelvariablen $\theta, \varphi \in [0, 2\pi]$ wie folgt parametrisiert*

$$x(\theta,\varphi) = (a + b\cos(\theta))\cos(\varphi), \quad y(\theta,\varphi) = (a + b\cos(\theta)\sin(\varphi), \quad z(\theta) = b\sin(\theta).$$

*Zeichne die Torusfläche mit $a = 10$, $b = 4$ mit* `mesh`*. Setze die Achsen mit* `axis equal` *auf gleiche Skalierung.*

### Zusatz: Sonstige Plots in zwei Dimensionen

#### Histogramme, Flächen und Polardarstellung

Neben dem normalen 2d Plot gibt es noch weitere plot Funktionen wie `bar, barh, pie, stem`, die insbesonder für die Analyse von Daten interessant sind. Ein einfaches (unsinniges) Beispiel für die Benutzung von `bar`:

```
>> Y=[7 6 5; 8 4 6; 1 2 3; 9 7 8];
>> subplot(2,2,1);
>> bar(Y)

>> subplot(2,2,2);
>> bar(-10:2:-2,Y)

>> subplot(2,2,3);
>> bar(Y,'stacked')

>> subplot(2,2,4);
>> barh(Y)
```

**Aufgabe 19.** *Mache Dich mit Hilfe der Online Hilfe mit den oben genannten Funktionen vertraut und erstelle jeweils ein Beispiel zu jeder Funktion ähnlich wie oben für* `bar` *gemacht.*

Eine spezielle Version eines `bar`-Plots ist ein Histogramm, das auf einfache Weise mit der Funktion `hist` (siehe: `help hist`) erstellt werden kann. Mit ihrer Hilfe lassen sich die Unterschiede zwischen den Zufallszahlengeneratoren `rand` und `randn` visualisieren.

```
>> N=2000; bins = 50;            % N Versuchsausgaenge in bins Kaestchen
>> r = rand(N, 1);               % Zufallzahlen mit rand erzeugen
>> figure('name', 'rand vs. randn'); % neues Fenster erzeugen
>> subplot(2,1,1);               % Teilplot erzeugen
>> hist(r, bins);                % mit hist die Verteilung der Zahlen in r darstellen
>> title('Verteilung von rand'); % Beschriftung nicht vergessen
>> rn = randn(N,1);              % Zufallszahlen mit randn erzeugen
>> subplot(2,1,2);               % andere Achse waehlen
>> hist(rn, bins);               % ebenfalls mit hist plotten
>> title('Verteilung von randn');
```

**Aufgabe 20.** *Experimentiere mit verschiedenen Werten für* `N` *und* `bins`*. Kontrolliere auch, wie fair der von Dir auf dem zweiten Zettel programmierte Würfel ist. Alternativ kannst Du auch die folgende Funktion Würfel ausprobieren, in diesem Fall benutze die Online-Hilfe um den Code zu verstehen.*

```
function wurf = dice(varargin)
%WURF = DICE(M,N,...) generates the results of a fair dice
% the syntax is the same as for the usual matrix functions, i.e.
% DICE(M) generates a M x M - Matrix with random entries between 1 and 6
% DICE(M,N,L) generates a M x N x L - Matrix with dice-entries
   wurf = ceil(6*rand(varargin{:}));
end
```

Listing 1: `dice.m`

Um Flächen zu zeichnen kann man die Befehle `area` und `fill` benutzen.

```
>> x = linspace(0,2*pi,31);
>> area(x,sin(x))               % Zeichnet die Flaeche unterhalb des Graphen
>> area(x,[sin(x);-1/2*sin(2*x);1/3*sin(3*x);-1/4*sin(4*x)]');
```

```
>> t = (1:2:15)'*pi/8;
>> x = sin(t); y = cos(t);
>> fill(x,y,'r'); axis equal; hold on;
>> text(0,0,'STOP','Color',[1 1 1],'HorizontalAlignment','center',...
    'FontSize',80,'FontWeight','bold');

>> figure
>> t = -2:0.2:2; y = t.^2;
>> fill(-y,t,'k');
```

Mit der Funktion `polar` (siehe: `help polar`) lassen sich Funktionen $f = f(\theta, r)$, die in Polarkoordinaten gegeben sind, einfach zeichnen.

```
>> r = 0:0.001:1; theta = 4*pi*r;
>> polar(theta, r, 'g');
>> hold on;
>> t=linspace(0,2*pi);
>> polar(t,sin(2*t).*cos(2*t));
```

**Aufgabe 21.** *Zeichne zwei Kreise, einen mit Radius* 1 *und einen mit Radius* 2 *mit* `polar` *in eine Achse.*

### Vektorfelder

Mit der Funktion `quiver` lassen sich Vektorfelder $f : \mathbb{R}^2 \to \mathbb{R}^2$ durch kleine Pfeile visualisieren. Dies ist im Zusammenhang mit Differentialgleichungen besonders nützlich.
Das Vektorfeld $f : \mathbb{R}^2 \to \mathbb{R}^2, (x, y) \mapsto (-y, x)$ kann wie folgt visualisiert werden

```
>> x = [0,1,2,3,0,1,2,3,0,1,2,3];      % x-Koordinaten
>> y = [0,0,0,0,1,1,1,1,2,2,2,2];      % y-Koordinaten
>> quiver(x,y,-y,x)
```

Einfacher lassen sich Koordinaten eines rechteckigen Gitters mit der Funktion `meshgrid` (siehe: `help meshgrid`) erstellen.

```
>> [x,y]=meshgrid(0:3,0:2)           % erzeugt die obigen Koordinaten
>> quiver(x,y,-y,x)                  % zeichnet das gleiche wie oben
>> [x,y]=meshgrid(-3:0.5:3,-3:0.5:3) % jetzt ist es einfach, groessere Bereiche auszuwaehlen
>> fx = -y; fy = x;
>> quiver(x,y,fx,fy)                 % und die Funktion f=(fx; fy) dort zu zeichnen
```

**Aufgabe 22.** *Zeichne das Vektorfeld*

$$f : \mathbb{R}^2 \to \mathbb{R}^2, \quad (x, y) \mapsto \frac{1}{\sqrt{x^2 + y^2}}(-y, x)$$

*im Bereich* $[-2, 2] \times [-2, 2]$ *(Hinweis: Vektorisierung beachten!).*

### Weitere Darstellungen der Komplexen Zahlen

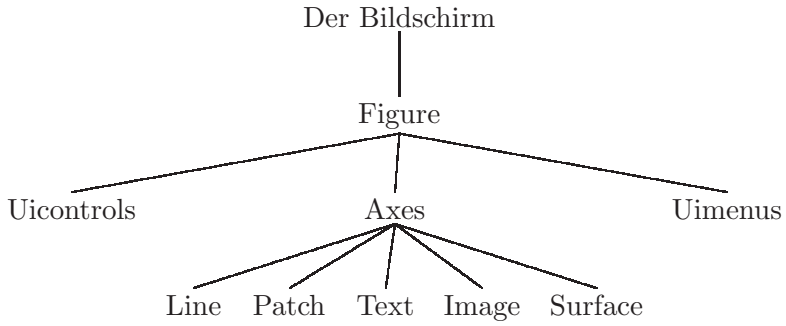
Wie Du weißt kann man komplexe Zahlen auch in Polarkoordinaten als $z = \rho e^{i\varphi}$ darstellen, wobei $\rho \geq 0$ und $\varphi \in [0, 2\pi)$ sind. Dieses kann mit Matlab auf verschiedene Weisen direkt (das heißt ohne den Umweg über `polar`) visualisiert werden.

```
>> z = eig(randn(20));      % erzeugt 20 "zufaellige" komplexe Zahlen
>> subplot(2,2,1)
>> compass(z)               % Komplexe Zahlen als Pfeile in der kompl.  Ebene
>> subplot(2,2,2)
>> quiver(zeros(20,1),zeros(20,1),real(z),imag(z)), axis equal
                            % liefert dasselbe mit Real- und Imaginaertail
>> subplot(2,2,[3,4])       % man kann auch subplots zusammenfassen
>> feather(z)               % zeichnet die Richtungen der kompl. Zahlen als Pfeile
```

**Aufgabe 23.** *Zeichne alle 11ten komplexen Einheitswurzeln (d.h. alle $z \in \mathbb{C}$ mit $z^{11} = 1$) mit den Grafikfunktionen* `plot`*,* `compass`*,* `quiver` *und* `feather`*.*

### Zusatz: Handle Graphics

Die Grafikobjekte in Matlab sind in einer strikt hierarchischen Ordnung aufgebaut, siehe dazu `doc` und dort `MATLAB -> Handle Graphics Property Browser`. Es stehen dort die "Elternobjekte" oben und die "Kinderobjekte" darunter. Also:




Ein einfaches Beispiel:
`>> plot(1:10,'x-');`
Mit der Funktion `findobj` (siehe: `help findobj`) kannst Du Dir alle Grafikobjekte ausgeben lassen:
`>> h = findobj`
`>> get(h,'type')` liefert dann den Typ der verschiedenen Grafikobjekte. Wie Du siehst, ist `h(4)` der Verweis (Handle) auf das Linienobjekt (es sei denn, Du hast vorher nicht alle Figuren geschlossen ;-)).
`>> set(h(4))`
Hiermit erhälst Du alle Einstellungsmöglichkeiten der Linie, unter anderem kannst Du auf die Daten der Linie mit
`>> get(h(4),'YData')`
zugreifen. Du kannst daher die Daten des Handles einfach ändern:
`>> set(h(4),'YData',[1 3 2 4 3 5 4 6 5 7]);` dieses ist insbesondere im Zusammenhang mit Animationen interessant, aber dazu später mehr.
Jetzt ein etwas umfangreicheres Beispiel:

```
>> x=linspace(0,2*pi);
>> subplot(2,2,1)
>> lo = plot(x,sin(x),'rx');
>> subplot(2,2,2)
>> ro = plot(x,sin(x),'bx');
>> subplot(2,2,[3,4])
>> u = plot(x,sin(x).*cos(x),'ko');
>> get(lo,'type')
>> axes(get(lo,'parent'))
>> title('sin(x)')
>> title(get(ro,'parent'),'cos(x)')
>> set(get(get(u,'parent'),'Title'),'Color',[0 0.6 0],'FontWeight','bold',...
'String','sin(x)cos(s)')
>> h = findobj(gcf,'type','axes')
>> set(h,'Xlim',[0 2*pi])
>> set(h,'Box','off')
>> set(h,'XTick',pi*(0:0.5:2))
>> set(h,'XTickLabel','0|pi/2|pi|3 pi/2|2pi')
>> set(h,'XGrid','on')
>> set(u,'FontSize',12)
>> set(get(u,'Parent'),'Fontsize',12)
```

```
>> figure(2)
>> set(get(lo,'Parent'),'Parent',2)
```

**Aufgabe 24.** *Versuche das Beispiel zu verstehen und kommentiere den Code! Es ist nicht alles korrekt, warum?*

Wie Du an dem obigen Beispiel siehst, ist es zwar möglich, aber zum Teil recht mühsam die Eigenschaften von Objekten nachträglich zu ändern. An dieser Stelle kommt die hierarchische Struktur zum Vorschein. Das folgende Beispiel ist gut für den Ausdruck in schwarz-weiß geeignet. Zunächst werden einige Defaultangaben gemacht und dann die Zeichnung erstellt:

```
>> set(0,'DefaultAxesFontSize',12);
>> set(0,'DefaultAxesColorOrder',[0 0 0],'DefaultAxesLineStyleOrder','-|--|:|-.',...
'DefaultLineLineWidth',2);
>> set(0,'DefaultAxesFontWeight','bold');
>> x=0:0.01:pi;
>> plot(x,[sin(x);sin(2*x);sin(3*x);sin(4*x)]);
>> title('Sinus(k*x), k=1,2,3,4');
```

**Aufgabe 25.** *Erzeuge zwei Figuren und gib ihnen unterschiedliche Defaultoptionen. Zeichne dann jeweils verschiedene Dinge in die Figuren hinein und schaue, ob sie sich unterscheiden.*

Zum Schluss ist es noch möglich auf Objekte zuzugreifen, die bestimmte Eigenschaften besitzen:

```
h=findobj(gca, 'LineStyle', '--') % in aktueller Achse gestricheltes Objekt finden
set(h,'color','red')              % und auf rot setzen
delete(h)                         % oder gleich ganz loeschen
```

**Aufgabe 26.** *Versuche, einzelne Grafikobjekte zu löschen sowie zwischen verschiedenen Grafikfenstern zu kopieren (**Hinweis:**siehe* `help copyobj`*).*

**Aufgabe 27.** *Schreibe eine Funktion, die als Eingabe einen Handle auf eine Linie hat, sowie eine Punkt als Eingabe nimmt. Dann soll die Linie um diesen Punkt erweitert werden.*
*Überlege, wie Du die Funktion erweitern kannst, z. B. falls es noch keine Linie gibt, oder falls sie ohne ein Handle auf eine Linie aufgerufen wird,...*
*Vielleicht möchtest Du auch einen Pfad durch interaktive Benutzereingab zeichnen, ähnlich wie die Funktion* `interaktiv` *vom zweiten Zettel?*

Dieses ist nur ein winziger Einblick in die Möglichkeiten von Grafikhandels. Mehr findest Du in der Online-Hilfe und insbesondere in den Handbüchern, die Du online als PDF-Versionen unter `doc` finden kannst. Insbesondere liefert Kapitel 8 des Handbuchs zur Grafik eine ausführliche Erklärung von Handle-Graphics.

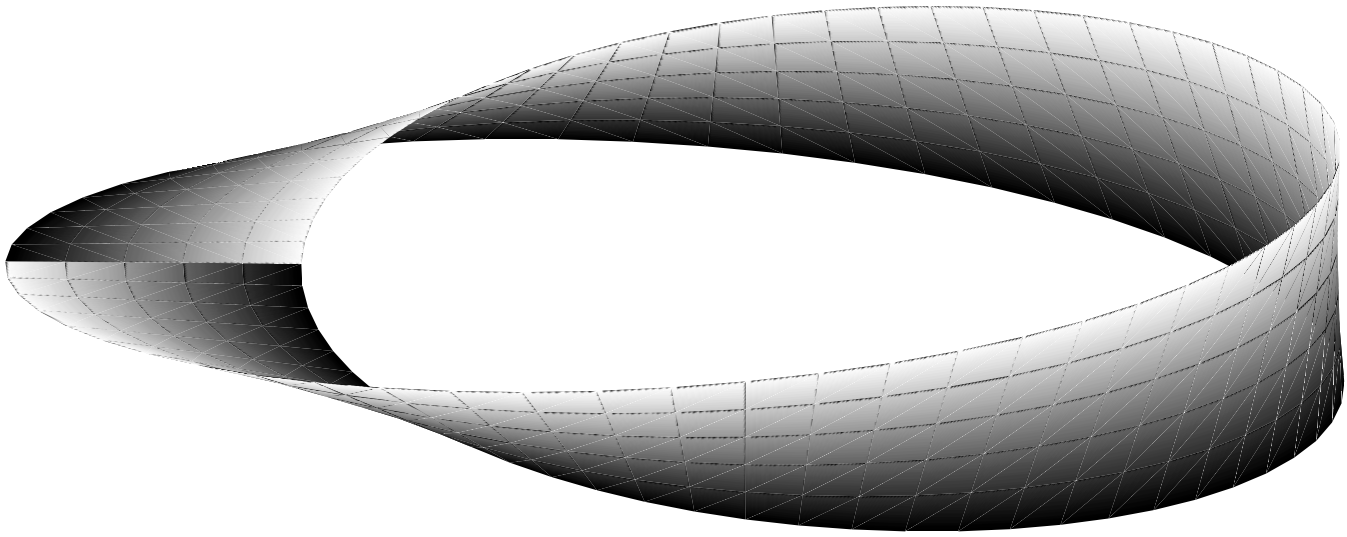


Abbildung 1: Möbiusband mit Matlab erzeugt. Der Farbverlauf soll die Orientierung des Bandes visualisieren. Man kann die Nahtstelle gut erkennen. Es wurden folgende Einstellungen benutzt: `colormap gray` und `axis tight`.