Aufgabe 1.

a) Numerisch können Lösungen quadratischer Gleichungen als Nullstellen (`roots`) von Polynomen ermittelt werden. Exakte Lösungen liefert die *Symbolic Toolbox*. Die Variable (oft x) muss zuvor einmal mit `syms x` als solche deklariert werden.

```
roots([1 3 -4])
syms x; solve(x^2 + 3*x - 4 == 0)
```

Im vorliegenden Fall werden als Antwort auf beide Weisen die Zahlen -4 und 1 ausgegeben. In der nachstehenden Tabelle zeigt die erste Zeile immer die exakten Ergebnisse von `solve`, die zweite Zeile die Ergebnisse von `roots`. Die letzte Spalte gibt die Anzahl verschiedener reeller Lösungen an.

Gleichung	Lösung 1	Lösung 2	Anzahl in $\mathbb{R}$
$x^2+3x-4=0$	-4	1	
	-4	1	2
$x^2+3x+4=0$	$-\frac{i}{2}\sqrt{7}-\frac{3}{2}$	$\frac{i}{2}\sqrt{7}-\frac{3}{2}$	
	$-1.5000+1.3229i$	$-1.5000-1.3229i$	0
$x^2-4x+2=0$	$2-\sqrt{2}$	$\sqrt{2}+2$	
	3.4142	0.5858	2
$2x^2-4x+2=0$	1	1	
	1	1	1

b) Die vier Bruch- und Wurzelgleichungen können problemlos gelöst werden:

```
syms x
solve(1/(x+3)+1/(x-3) == 6/(x^2-9))
solve(1/(x+3)+1/(x-3) == 5/(x^2-9))
solve(2+sqrt(3*x*(x-2)) == x)
solve(2-sqrt(3*x*(x-2)) == x)
```

Die erste Gleichung ist unlösbar, was Matlab etwas kryptisch mit `Empty sym: 0-by-1` quittiert. Bei der zweiten und dritten Gleichungen gibt es je eine Lösung, $\frac{5}{2}$ und 2. Die vierte Gleichung besitzt die beiden Lösungen -1 und 2.

c) Auch die einfachen Exponentialgleichungen löst Matlab vollständig. Der ln wird dabei als `log` angegeben. Bei der Interpretation der Antwort ist zudem zu beachten, dass Matlab auch komplexe Lösungen sucht. Diese sind für uns bedeutungslos.

Gleichung	Antwort von Matlab	reelle Lösungen	Anzahl in $\mathbb{R}$
$e^x - 4e^{-x} = 0$	`log(2)`	$\ln 2$	1
$e^x + 4e^{-x} = 4$	`log(2)`	$\ln 2$	1
$e^x + 2e^{-x} = 3$	`log(2)`	$\ln 2$	
	0	0	2
$e^{2x} + e^x = 0$	`pi*1i`		0
$e^{2x} - e^x = 2$	`pi*1i`		
	`log(2)`	$\ln 2$	1

d) Zu lösen ist für verschiedene Werte des Parameters a und $t \in [0, \frac{\pi}{2}]$ die trigonometrische Gleichung

$$a - \sin t = \cos^2 t$$

Am geschicktesten definiert man zunächst

```
syms t
a = (cos(t))^2 + sin(t)
```

Dann können leicht die verschiedenen Werte von a eingesetzt werden. Bei der Interpretation der Antwort ist zu beachten, dass durch komplexe Rechnung für uns sinnlose Ergebnisse entstehen und wir zudem nur an Ergebnissen im Intervall $[0, \frac{\pi}{2}]$ interessiert sind.

Eingabe	Antwort	Interpretation	Anzahl in $\mathbb{R}$
`solve(a == 1/4)`	`-pi/6`	nicht in $[0, \frac{\pi}{2}]$	
	`(7*pi)/6`	nicht in $[0, \frac{\pi}{2}]$	
	`asin(3/2)`	für uns sinnlos	
	`pi - asin(3/2)`	für uns sinnlos	0
`solve(a == 1)`	`0`	1. Lösung	
	`pi/2`	2. Lösung	2
`solve(a == 19/16)`	`asin(1/4)`	1. Lösung	
	`asin(3/4)`	2. Lösung	
	`pi - asin(1/4)`	nicht in $[0, \frac{\pi}{2}]$	
	`pi - asin(3/4)`	nicht in $[0, \frac{\pi}{2}]$	2
`solve(a == 5/4)`	`pi/6`	1. Lösung	
	`(5*pi)/6`	nicht in $[0, \frac{\pi}{2}]$	1

Aufgabe 2.

a) Das Rechnen mit komplexen Zahlen funktioniert genau wie bei reellen Zahlen. Insbesondere rechnet Matlab dabei nicht exakt. Symbolisches Rechnen muss - etwa durch die Funktion `sym` - explizit angefordert werden. Die imaginäre Einheit kann als `i` (mathematisch) oder als `j` (elektrotechnisch) eingegeben werden.

Aufgabe Ergebnis	Eingabe	Matlab-Ergebnis
$z_1 = 5i \cdot e^{\frac{\pi}{6}i}$	`z1=5i*exp(pi/6*i)`	`-2.5000 + 4.3301i`
$= \frac{5}{2}(-1+i\sqrt{3})$	`w1=sym(z1)`	`(3^(1/2)*5i)/2 - 5/2`
$z_2 = \frac{1-3i}{1-i}$	`w2=sym((1-3i)/(1-i))`	`2 - 1i`
$= 2 - i$	`z2=double(w2)`	`2.0000 - 1.0000i`
$z_3 = \frac{z_1}{z_2}$	`z3=z1/z2`	`-1.8660 + 1.2321i`
	`w3=sym(z3)`	`- 8403831313147477/4503599627370496 +`
$= (-1-\frac{1}{2}\sqrt{3})+$		`+ 5548663557868715i/4503599627370496`
$+i(-\frac{1}{2}+\sqrt{3})$	`w3=w1/w2`	`- 3^(1/2)*(1/2 - 1i) - 1 - 1i/2`
$z_4 = \frac{\bar{z}_1}{\bar{z}_2}$	`z4=conj(z1)/conj(z2)`	`-1.8660 - 1.2321i`
$= (-1-\frac{1}{2}\sqrt{3})+$	`w4=conj(w1)/conj(w2)`	`- 3^(1/2)*(1/2 + 1i) - 1 + 1i/2`
$+i(\frac{1}{2}-\sqrt{3})$	`w4=conj(w3)`	`- 3^(1/2)*(1/2 + 1i) - 1 + 1i/2`

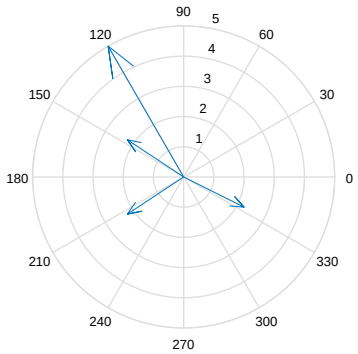
Erkennbar ist, dass die Vermischung von numerischen Variablen und symbolischen Variablen in `w3=sym(z3)` zu merkwürdigen und untauglichen Ergebnissen führt.

b) Durch `help compass` erfährt man, dass komplexe Zahlen mittels `compass` als Zeiger dargestellt werden können.

```
compass([z1 z2 z3 z4])
```

Die Zahlen müssen dabei in numerischer Form vorliegen. Zu einem Fehler führt demnach

```
compass([w1 w2 w3 w4])
```



Falls die numerische Form nicht vorhanden ist, kann sie mit der Funktion `double` aus der symbolischen gewonnen werden.

```
compass(double([w1 w2 w3 w4]))
```

c) Für den Betrag von `w3` und `w4` liefert Matlab

```
((3^(1/2) - 1/2)^2 + (3^(1/2)/2 + 1)^2)^(1/2)
```

Mit `simplify` muss eine Vereinfachung explizit verlangt werden.

Eingabe	Antwort
`betraege = abs([w1 w2 w3 w4])`	`betraege =` `[ 5, 5^(1/2), ((3^(1/2) - ...`
`simplify(betraege)`	`ans =` `[ 5, 5^(1/2), 5^(1/2), 5^(1/2)]`
`angle([z1 z2 z3 z4])*(180/pi)`	`ans =` `120.0000 -26.5651 146.5651 -146.5651`

Winkel werden von Matlab im Bereich $(-\pi, \pi]$ angegeben. Die berechneten Gradangaben liegen demnach im Bereich $(-180, 180]$.

Aufgabe 3.

Die vorgelegte komplexe Zahl $w = \frac{1}{2}(i-1)\sqrt{2}$ wird als symbolische Variable definiert:

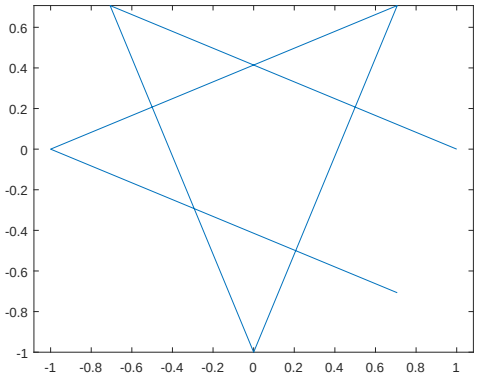
```
w=sym(sqrt(2)/2*(j-1))
```

a) Am besten werden die Hochzahlen als Vektor gespeichert:

```
hochzahlen=0:5
potenzen=w.^hochzahlen
betraege=abs(potenzen)
winkel=angle(potenzen)
```

Die Beträge sind alle 1, die Winkel sind

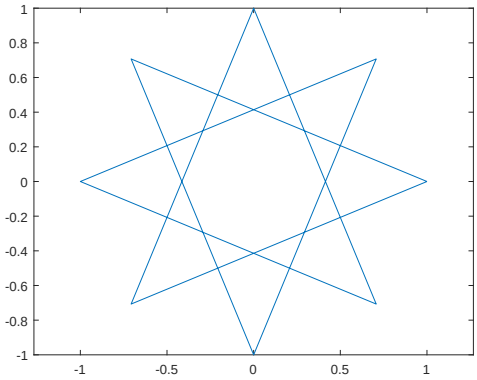
```
[ 0, (3*pi)/4, -pi/2, pi/4, pi, -pi/4]
```



Die Zahl $w = w^1$ hat also den Betrag 1 und das Argument $\varphi := \frac{3\pi}{4}$. Daraus ergibt sich, dass tatsächlich die Beträge von allen Potenzen 1 sein müssen. Auch die Winkel sind korrekt. Sie sind modulo 2π Vielfache von φ:

$$-\frac{\pi}{2} = 2\cdot\varphi - 2\pi, \quad \frac{\pi}{4} = 3\cdot\varphi - 2\pi, \quad \pi = 4\cdot\varphi - 2\pi, \quad -\frac{\pi}{4} = 5\cdot\varphi - 4\pi$$

b) Das Polygon beginnt bei $w^0 = 1$ und endet bei $w^5 = e^{-i\frac{\pi}{4}}$, siehe Aufgabenteil a). Wegen $w^8 = 1 = w^0$ erhält man mit

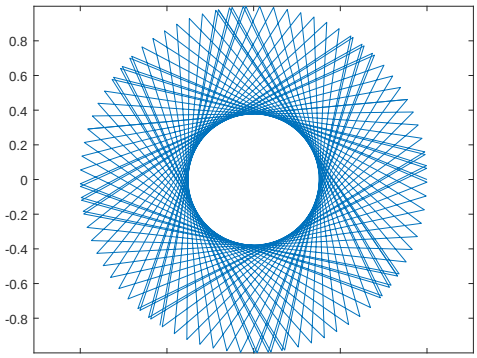


```
hochzahlen=0:8
potenzen=w.^hochzahlen
realTeile=real(potenzen)
imagTeile=imag(potenzen)
plot(realTeile,imagTeile)
axis equal
```

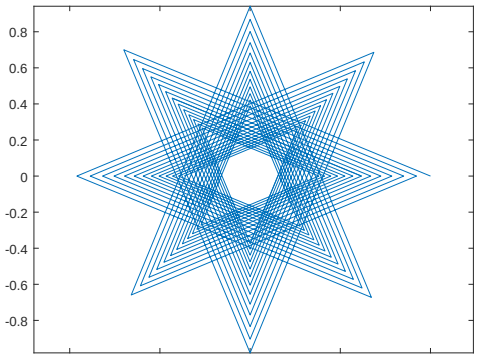
ein geschlossenes Polygon. Das zugehörige Bild zeigt demnach sämtliche Potenzen w^n mit $n \in \mathbb{N}_0$.

c) Es wird also einfach `hochzahlen=0:8` durch `hochzahlen=0:100` ersetzt. Das Bild ändert sich dadurch nicht. Interessant ist das Verhalten, wenn die Zahl w durch eine oder beide der folgenden Aktionen leicht verändert wird:

```
w=w*exp(i/sym(100))     % verdrehen
w=w*0.99                % verkleinern
```

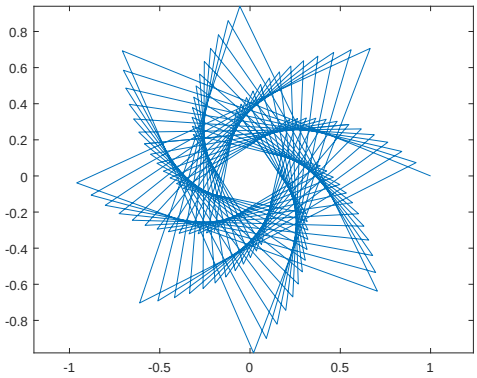


Das obere der drei Bilder zeigt alleiniges Verdrehens, das mittlere Bild alleiniges Verkleinern. Beim unteren Bild wird sowohl gedreht, als auch verkleinert.



Es bietet sich an, die Befehle in einer Skript-Datei (**m-File**) zu speichern und als ersten Befehl `clx` einzufügen. Dann können Sie leicht Änderungen vornehmen und das Bild mit einem Knopfdruck neu erstellen.

Hinweis: Interessant sind auch Experimente mit anderen Zahlen, insbesondere das Beispiel $\frac{1}{5}(3+4i)$ aus der Vorlesung.



Aufgabe 4.

a) Die folgenden Zeilen müssen in einer Datei namens `argument.m` gespeichert werden und zwar entweder im aktuellen Verzeichnis oder in einem Verzeichnis, das sich im

Matlab-Suchpfad befindet. Unter dem ersten Reiter (**HOME**) finden Sie die Möglichkeit (**Set Path**), den Suchpfad zu erweitern.

```
function arg = argument( z )
% bestimmt das Argument der angegebenen komplexen Zahl
%   das Argument ist >= 0 und < 2pi
winkel = angle(z);
if (winkel < 0)
    winkel = winkel + 2*pi;
end
arg = winkel
end
```

b) Die folgenden Zeilen müssen entsprechend in einer Datei namens `polar.m` gespeichert werden.

```
function [ r phi ] = polar( z )
% liefert die Polarkoordinaten [r, phi] der angegebenen komplexen Zahl
r = abs(z);
phi = argument(z);
end
```

c) Wie die Funktion `angle` kann auch `argument` sowohl numerisch, als auch symbolisch verwendet werden.

Eingabe	Antwort	Eingabe	Antwort
`angle(sym(-1-1i))`	`-(3*pi)/4`	`angle(-1-1i)`	`-2.3562`
`argument(sym(-1-1i))`	`(5*pi)/4`	`argument(-1-1i)`	`3.9270`
`angle(sym(-1+1i))`	`(3*pi)/4`	`argument(sym(3+4i))`	`atan(4/3)`
`argument(sym(-1+1i))`	`(3*pi)/4`	`argument(sym(-3-4i))`	`pi + atan(4/3)`
`argument(sym(2))`	`0`	`argument(sym(2i))`	`pi/2`

Aufgabe 5. Um ableiten zu können, müssen die unabhängigen Variablen zuvor als symbolisch deklariert werden. Bei a) geschieht dies am einfachsten durch `syms x`, bei b) entsprechend mit `syms x y`.

a) In den meisten Fällen genügt der Befehl `diff`, wie etwa in *Aufgabe 1a*:

Eingabe	Antwort
`a1=diff((1-2*x^2)^9)`	`a1 =` `-36*x*(2*x^2 - 1)^8`

Es kommt aber durchaus vor, dass Vereinfachungen nicht automatisch vorgenommen werden. Häufig hilft dann ein anschließendes `simplify`, etwa in *Aufgabe 3a*:

Eingabe	Antwort

`c1=diff(log(x*exp(-sin(2*x))))`	`c1 =`
	`(exp(sin(2*x))*(exp(-sin(2*x)) -`
	`- 2*x*cos(2*x)*exp(-sin(2*x))))/x`

`c1=simplify(c1)`	`c1 =` `-(2*x*cos(2*x) - 1)/x`

Wie beim manuellen Ableiten macht es manchmal Sinn, vor dem Ableiten auszuklammern. Zu sehen ist dies etwa in *Aufgabe 1d*:

Eingabe	Antwort
`a4=diff((1-x^2)/sqrt(x))`	`a4 =` `(x^2 - 1)/(2*x^(3/2)) - 2*x^(1/2)`
`simplify(a4)`	`ans =` `-(3*x^2 + 1)/(2*x^(3/2))`
`expand(a4)`	`ans =` `- (3*x^(1/2))/2 - 1/(2*x^(3/2))`
`expand((1-x^2)/sqrt(x))`	`ans =` `1/x^(1/2) - x^(3/2)`
`a4=diff(ans)`	`a4 =` `- (3*x^(1/2))/2 - 1/(2*x^(3/2))`

Gelegentlich ist das Ergebnis einfach schwer lesbar. Dann hilft meist `pretty`, etwa in *Aufgabe 1g*:

Eingabe	Antwort
`a7=diff(sqrt(1+sqrt(x)))`	`a7 =` `1/(4*x^(1/2)*(x^(1/2) + 1)^(1/2))`

`pretty(a7)`	`1`
	`--------------------------`
	`4 sqrt(x) sqrt(sqrt(x) + 1)`

Vereinzelt treten Situationen auf, wo auch vorherige oder nachbereitende Verwendungen von `simplify` oder `expand` keine wirklich guten Lösungen liefern. Ein Beispiel hierfür ist *Aufgabe 1c*. Anschließendes `simplify` ist eine deutliche Verbesserung, stellt aber nicht ganz zufrieden. Die beiden Potenzen von $(1-x)$ werden einfach nicht zusammengefasst. Mit manueller Vorarbeit klappt es:

Eingabe	Antwort

`a3=diff(8*(1-x)*((1-x)^3)^(1/4))`	`a3 =`
	`(3*(8*x - 8)*(x - 1)^2)/(4*(-(x - 1)^3)^(3/4)) -`
	`- 8*(-(x - 1)^3)^(1/4)`

`simplify(a3)`	`a3 =` `(14*(x - 1)^3)/(-(x - 1)^3)^(3/4)`
`a3b=diff(8*(1-x)^(1+3/4))`	`a3b =` `-14*(1 - x)^(3/4)`

b) Bei mehreren unabhängigen Variablen muss natürlich angegeben werden, nach welcher davon abgeleitet werden soll. Höhere Ableitungen erhält man durch erneutes Ablei-

ten einer Ableitung. Alternativ können sie aber auch in einem Schritt aus der Funktion bestimmt werden.

$f(x,y)$	`f=2*y^2-x*(x-1)^2`	Alternative 1	Alternative 2
$f_x(x,y)$	`fx=diff(f,x)`		
$f_y(x,y)$	`fy=diff(f,y)`		
$f_{xx}(x,y)$	`fxx=diff(fx,x)`	`fxx=diff(f,x,x)`	`fxx=diff(f,x,2)`
$f_{xy}(x,y)$	`fxy=diff(fx,y)`	`fxy=diff(f,x,y)`	
$f_{yx}(x,y)$	`fyx=diff(fy,x)`	`fxx=diff(f,y,x)`	
$f_{yy}(x,y)$	`fyy=diff(fy,y)`	`fyy=diff(f,y,y)`	`fxx=diff(f,y,2)`

Häufig wird der gesamte Gradient $\nabla f(x,y)$ einer Funktion $f(x,y)$ benötigt. Bei Bedarf können aus diesem die einzelnen Ableitungen gewonnen werden.

Eingabe	Antwort
`f=x*exp(-(x^2+y^2))`	`f=` `x*exp(- x^2 - y^2)`

`nabla=jacobian(f)`	`nabla=`
	`[ exp(- x^2 - y^2) - 2*x^2*exp(- x^2 - y^2),`
	`-2*x*y*exp(- x^2 - y^2)]`

`fx=nabla(1)`	`fx=` `exp(- x^2 - y^2) - 2*x^2*exp(- x^2 - y^2)`
`fy=nabla(2)`	`fy=` `-2*x*y*exp(- x^2 - y^2)`

Der Gradient kann damit besonders leicht an einzelnen Stellen ausgewertet werden. Dazu werden mittels `subs` für die Variable `[x,y]` die gewünschten Werte eingesetzt.

Eingabe	Antwort
`subs(nabla,[x,y],[0,0])`	`ans=` `[ 1, 0]`
`subs(nabla,[x,y],[1,1])`	`ans=` `[ -exp(-2), -2*exp(-2)]`
`subs(nabla,[x,y],[0,sqrt(log(sym(86)))])`	`ans=` `[ 1/86, 0]`

Besonders elegant ist damit die Berechnung der zweiten Ableitungen. Der gesamte Vektor `nabla` kann durch `hesse=jacobian(nabla)` auf einmal abgeleitet werden, wodurch eine Matrix mit den zweiten Ableitungen - die sogenannte **Hesse-Matrix** - entsteht.

```
hesse =
[   4*x^3*exp(- x^2 - y^2) - 6*x*exp(- x^2 - y^2), 4*x^2*y*exp(- x^2 - y^2) - 2*y*exp(- x^2 - y^2)]
[ 4*x^2*y*exp(- x^2 - y^2) - 2*y*exp(- x^2 - y^2), 4*x*y^2*exp(- x^2 - y^2) - 2*x*exp(- x^2 - y^2)]
```

Die Bestandteilen können leicht abgefragt werden. Beispielsweise ist `hesse(2,1)` der Wert in der zweiten Zeile und der ersten Spalte, also die Ableitung f_{yx}. Mit `hesse(2,:)` bekommt man die ganze zweite Zeile, mit `hesse(:,1)` die gesamte erste Spalte. Auch das Einsetzen von Werten ist bequem:

Eingabe	Antwort

`subs(hesse,[x,y],[1,1])`	`ans=`
	`[ -2*exp(-2), 2*exp(-2)]`
	`[  2*exp(-2), 2*exp(-2)]`

Aufgabe 6. Mittels `plot` können Polygonzüge gezeichnet werden. Diese können durch die mächtige Matrizen-Funktionalität von Matlab auch leicht aus Funktionen oder Relationen gewonnen werden. Für symbolische Funktionen ist aber oftmals die Verwendung von `fplot` geschickter.

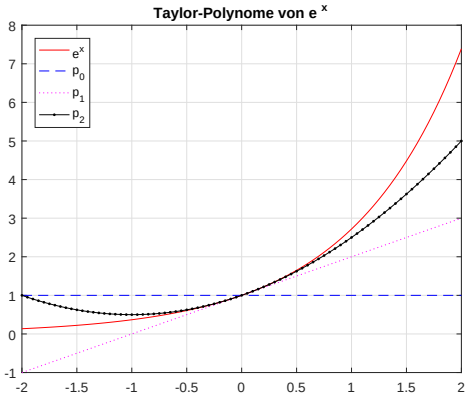
a) Zunächst werden zur Funktion e^x die Taylorpolynome p_0 bis p_2 berechnet (nachfolgend linke Spalte). Falls `plot` verwendet wird, muessen daraus Polygonzüge gebastelt werden (nachfolgend rechte Spalte). Die x-Werte werden in einem Vektor (`X`) vorgegeben.

```
syms x
f=exp(x)
p0=taylor(f,x,'Order',1)
p1=taylor(f,x,'Order',2)
p2=taylor(f,x,'Order',3)
```

```
X=-2:0.05:2
F=subs(f,x,X)
P0=subs(p0,x,X)
P1=subs(p1,x,X)
P2=subs(p2,x,X)
```

Damit kann die Darstellung erfolgen. Die Funktion soll rot dargestellt werden, das nullte Taylorpolynom blau strichliert, das erste magenta punktiert und das zweite schwarz strichpunktiert. Das Bild soll durch ein Gitter, einen Titel und eine Legende ergänzt werden.

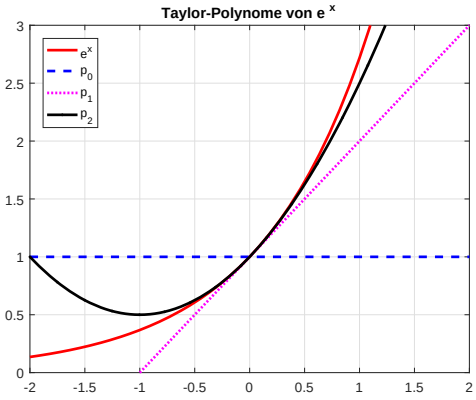
```
plot(X,F,'r', X,P0,'--b', ...
             X,P1,':m ', X,P2,'.-k')
grid
title('Taylor-Polynome von e^x')
legend('e^x','p_0','p_1','p_2', ...
            'Location', 'Northwest')
```



Wie Sie sehen, kann durch `...` am Ende einer Zeile der Quellcode eines Befehls in zwei oder mehr Zeilen umgebrochen werden.

b) Um die geforderte leichte Änderbarkeit zu erreichen, muss neben der Funktion mindestens noch der Grad des ersten Taylorpolynoms als Variable (`von`) angelegt werden.

```
syms x
f=exp(x);
von=0;
```



Der `;` am Ende einer Befehlszeile bewirkt, dass keine Ausgabe im *Command Window* erfolgt. Sinnvoll ist, auch die Darstellungsarten in einer Variable zu speichern. Dazu eignet sich eine Matrix aus Zeichen.

```
darst=['r  '; '--b'; ':m '; '.-k'];
```

Zu beachten ist dabei, dass jede Zeichenkette gleich lang sein muss. Kürzere Strings müssen mit Leerzeichen aufgefüllt werden. Die Darstellungsart für die Funktion ist die erste Zeile der Matrix, also `darst(1,:)`. Wir verwenden diesmal `fplot` und wählen eine größere Linienbreite. Mittels `axis` wählen wir einen horizontalen Darstellungsbereich von -2 bis 2 und einen vertikalen von 0 bis 3. Schließlich werden ein Gitter und ein Titel zugefügt.

```
fplot(f,darst(1,:),'Linewidth',2)
axis([-2 2 0 3])
grid
title('Taylor-Polynome von e^x')
```

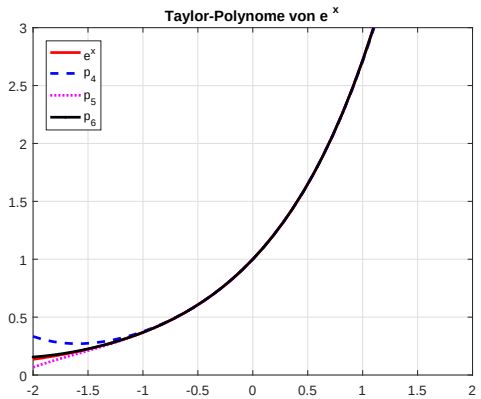
Die Taylorpolynome selbst werden in einer Schleife berechnet und gezeichnet. Damit sie ins gleiche Bild eingetragen werden, muss die Schleife von den Befehlen `hold on` und `hold off` umschlossen werden.

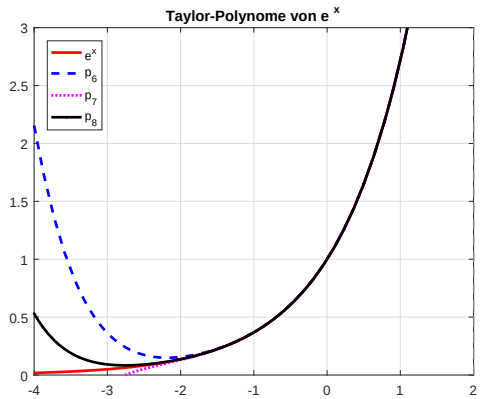
```
hold on
for i=1:3
    p=taylor(f,x,'Order',von+i);
    fplot(p,darst(i+1,:),'Linewidth',2)
end
hold off
```

Der Befehl `sprintf` erlaubt, in der Legende die richtigen Indizes für die Polynome einzusetzen.

```
legend('e^x',sprintf('p_{%i}',von),sprintf('p_{%i}',von+1),...
             sprintf('p_{%i}',von+2),'Location', 'Northwest');
```

c) Das nachstehend linke Diagramm (`von=4`) zeigt, dass p_6 (schwarze Kurve) im Bereich $[-2, 1]$ kaum mehr von e^x abweicht. Im rechten Diagramm (`von=6`) wird aber deutlich, dass sich das für $x < -2$ deutlich ändert. p_4 (blau strichliert) hat hier einen ganz anderen Verlauf als die e-Funktion.





d) Eine von 0 verschiedene Entwicklungsmitte kann als `'ExpansionPoint'` vorgegeben werden.

```
taylor(exp(x),x,'ExpansionPoint',-1,'Order',3)
taylor(sin(x),x,'ExpansionPoint',pi,'Order',7)
taylor(sin(x),x,'ExpansionPoint',pi,'Order',6)
taylor(log(x),x,'ExpansionPoint',1,'Order',4)
taylor(2+log(x),x,'ExpansionPoint',1,'Order',4)
```

Die Ergebnisse sind

```
exp(-1) + exp(-1)*(x + 1) + (exp(-1)*(x + 1)^2)/2
pi - x + (x - pi)^3/6 - (x - pi)^5/120
pi - x + (x - pi)^3/6 - (x - pi)^5/120
x - (x - 1)^2/2 + (x - 1)^3/3 - 1
x - (x - 1)^2/2 + (x - 1)^3/3 + 1
```

Bei der Entwicklung von e^x um $x = -1$ ist das Ergebnis genau wie erwartet. Bei $\sin x$ um $x = \pi$ ist auffällig, dass das Glied vom Grad 1 als $\pi - x$ und nicht als $-(x - \pi)$ angegeben wird. Bei der Entwikclung von $\ln x$ wird das Glied $(x - 1)$ vom Grad 1 sogar gespalten, das x am Anfang, das -1 am Ende. Das ist dann definitiv schon nicht mehr korrekt. Noch schlimmer wird es bei $2 + \ln x$, wo die Glieder vom Grad 0 und Grad 1 zusammengefasst werden. Richtig wäre $2 + (x - 1) - \frac{1}{2}(x - 1)^2 + \frac{1}{2}(x - 1)^3$.

Aufgabe 7. Grenzwerte berechnet Matlab sehr zuverlässig. Bei den *Aufgaben 1 bis 3* und auch der schweren *Aufgabe 8* von *Übungsblatt 8* gibt es keinerlei Probleme. Gleiches gilt für sämliche Folgen-Grenzwerte von *Zusatzblatt 3*. Beispielhaft seien gezeigt:

Nr	Aufgabe	Eingabe	Antwort
1c	$\lim\limits_{x\to 1} \dfrac{\sin^2 \pi x}{1 - x + \ln x}$	`limit((sin(pi*x))^2/(1-x+log(x)),1)`	`-2*pi^2`
1d	$\lim\limits_{x\to \infty} \dfrac{\ln(1 + e^x)}{x}$	`limit(log(1+exp(x))/x,inf)`	`1`
1e	$\lim\limits_{x\to 0+} \sqrt[3]{x} \cdot \ln x$	`limit(x^(1/3)*log(x),x,0,'right')`	`0`
2a	$\lim\limits_{n\to \infty} \sqrt[n]{3}$	`limit(3^(1/n),inf)`	`1`
3a	$\lim\limits_{x\to \infty} \dfrac{x \cdot (\sin x^4 + 2)}{x^3 + 1}$	`limit((x*(sin(x^4)+2))/(x^3+1),inf)`	`0`
8	$\lim\limits_{n\to \infty} \dfrac{1}{n} \sqrt[n]{n!}$	`limit((1/n)*factorial(n)^(1/n),inf)`	`exp(-1)`
2b	$\lim\limits_{n\to \infty} \dfrac{(-3)^n + 2^n}{3^n + 2^{n+1}}$	`limit(((-3)^n+2^n)/(3^n+2^(n+1)),inf)`	`NaN`
2c	$\lim\limits_{n\to \infty} \dfrac{3^{n+1} + 2^{2n}}{3^n + 2^{n+1}}$	`limit(((-3)^n+2^n)/(3^n+2^(n+1)),inf)`	`Inf`

Selbst der Parameter a in *Aufgabe 4* bereitet Matlab keine Schwierigkeiten. Die Eingabe

```
syms a; limit((x-1)^2/(a*(x-1)+log(x)),x,1)
```

wird von Matlab mit

```
ans =
piecewise([a == -1, -2], [a ~= -1, 0])
```

beantwortet. Damit ist gemeint

$$\lim_{x\to 1} \frac{(x-1)^2}{a\cdot(x-1)+\ln x} = \begin{cases} -2 & \text{für } a = -1 \\ 0 & \text{für } a \neq -1 \end{cases}$$

Bei *Aufgabe 5* liefert Matlab auch ein korrektes Ergebnis, schafft es aber nicht, dieses in der einfachen Weise

$$f(x) := \lim_{n\to\infty} \frac{1}{1+(x^2)^n} = \frac{1}{1+\lim\limits_{n\to\infty}(x^2)^n} = \begin{cases} 1 & |x| < 1 \\ \frac{1}{2} & |x| = 1 \\ 0 & |x| > 1 \end{cases}$$

darzustellen. Vielmehr gibt Matlab als Ergebnis (in einer Zeile!) an

```
piecewise([~abs(x)^2 in Dom::Interval(0, 1) & x in {-1, 1},
limit(1/2, n, Inf)], [~x in {-1, 1} & (in(x, 'real') | 1 < x^2
| abs(x) ~= 1 | abs(x)^2 in Dom::Interval(0, 1)),
limit(1/((x^2)^n + 1), n, Inf)])
```

Auch `simplify` verbessert das nur sehr unwesentlich. Das Problem von Matlab liegt hier darin, dass er x prinzipiell nicht als reell ansieht. Durch explizite Kennzeichnung von x als reelle Variable

```
assume(x,'real')
limit(1/(1+(x^2)^n),n,inf)
```

entsteht ein akzeptables Ergebnis:

```
piecewise([0 < x^2 - 1, 0], [x^2 - 1 < 0, 1], [x in {-1, 1}, 1/2])
```