Matlab Tutorials

Das *LiveScript* zu dem Tutorial 3.

Lösung: Aufgabe aus Tutorial 2

```
% Script to analyse some data
%
% This script contains some very usefull calculations from the
% "Matlab Tutorials".

% Author: Julian Kahnert (c) IHA @ Jade Hochschule

clear
close all

load carsmall.mat

figure
plot(Model_Year, Horsepower, 'xk')
ha = gca;                                   % handle der axis speichern, um später auf die
                                            % Eigenschaften zugreifen zu können
xlabel('Baujahr',         'FontSize', 13)
ylabel('PS',              'FontSize', 13)
title('Beispielplot 2',   'FontSize', 16)

set(ha, ...
    'XLim', [65 85], ...
    'XTick', 65:5:85, ...
    'YTick', 40:40:240, ...
    'YGrid', 'on')
```

Falls mehrere Plots vorhanden sind, kann nach jedem Plot-Befehl ein *ha1 = gca;* bzw. *ha2 = gca;* verwendet werden, um die Handles für jeden Plot abzuspeichern (Beispiel hierfür bei *subplot* in Tutorial 3.1) und nachträglich verändern zu können .

Tutorial 3.1

- Übersicht von verschiedenen Darstellungsformen

Funktion: **figure()**

Der Befehl *figure* gibt hat ein Output Argument, welches ein *handle* ist. Hiermit kann man auf die Eigenschaften der Figure zugreifen. Dies ist nicht mit dem *handle*, welches *gca* ausgibt zu verwechseln. *gca* gibt ein handle einer Axis zurück, mit dem man auf die Eigenschaften dieser Axis zugreifen kann.

Hier sind zwei Möglichkeiten, um Eigenschaften direkt zu setzten:

```
hFig1 = figure;
set(hFig1, 'Units', 'pixels', 'Position', [100 100 500 300])
hFig2 = figure('Units', 'pixels', 'Position', [100 100 500 300]);
```

Eine Figure wird automatisch erzeugt, wenn der Befehl *plot* ausgeführt wird. Es kann allerdings sinnvoll sein *figure* zu verwenden, wenn man bestimmte Eigenschaften definieren möchte oder mehrere Grafiken hat, sodass *gcf* nicht zielführend ist.

Funktion: **plot()**

Beispiel für die grafische Darstellung eines Rauschvektors mit 20 Elementen:

```
x = rand(20,1);
plot(x)
```

Rauschvektor mit einer rot gestrichelten Linie und Diamanten als Marker:

```
x = rand(20,1);
plot(x,'rd--')
```

Für weitere Informationen: *doc LineSpec*

Andere Darstellungsformen:

```
% Säulendiagramm
bar(rand(20,1));

% Tortendiagramm
pie([2 4 3 5],{'North', 'South', 'East', 'West'})

% für diskrete Werte
stem(linspace(-1, 1, 20))

% Polarplot
theta = 0:0.01:2*pi;
rho = sin(2*theta) .* cos(2*theta);
polarplot(theta, rho)
```

Funktion: **subplot()**

Darstellung von mehreren Plots in einer Figure. Bei dem Befehl *subplot(M, N, O)* gibt das *M* die Anzahl der Zeilen und *N* die Anzahl der Spalten an. *O* gibt die Nummer(n) des Subplots an, in der der Plot dargestellt werden soll.

```
figure('Name', 'Beispiel: subplot')
subplot(3, 2, 1);
plot(rand(20, 1))
ha1 = gca;

subplot(3, 2, 2);
plot(rand(20, 1))
ha2 = gca;

subplot(3, 2, 3);
plot(rand(20, 1))
ha3 = gca;
```

```
subplot(3, 2, 4);
plot(rand(20, 1))
ha4 = gca;

subplot(3, 2, 5:6);
plot(rand(20, 1))
ha5 = gca;
```

Mit *ha1*, *ha2*, *ha3*, *ha4* und *ha5* können später die Eigenschaften verändert werden.

Funktion: **hold()**

Darstellung von mehreren Säulendiagrammen innerhalb eines Plots:

```
figure;
bar(rand(20, 1));
hold('on')
bar(rand(20, 1),'r');
hold('off')
```

Funktion: **xlabel() / ylabel()**

```
xlabel('X-Achsen Beschriftung \rightarrow')
ylabel('Formel: \pi \int_{0}^{1/pi} \rightarrow')
```

Funktion: **title()**

```
title('Überschrift')
```

Funktion: **grid()**

Darstellung eines Gitternetzes in der Grafik bzw. entfernen des Gitters:

```
grid('on')
grid('off')
```

Funktion: **legend()**

Darstellung einer Legende. Für weitere Attribute wie z.B. den Ort der Legende, siehe: *doc legend*

```
figure;
plot(rand(20, 1));
hold;
plot(rand(20, 1), 'r');
legend('Erstes Rauschen', 'Zweites Rauschen')
```

Funktion: **text()**

Darstellung von Text/Annotation in einem Plot:

```
text(2, 0.5, 'Text an Punkt x,y -> hier x=2;y=0.5')
annotation('arrow', [0.2 0.4], [0.4 0.4])
```

Tutorial 3.2

- Zugriff auf Eigenschaften

Um Veränderungen an den Eigenschaften einer Axis zu verdeutlichen, erzeugen wir uns zunächst einen Plot:

```
plot(rand(20, 1))   % Plot erzeugen
ha = gca;           % handle abspeichern
```

Nun kann man sich die Eigenschaften der Axis mit Hilfe von *get* und dem handle *ha* ansehen:

```
get(ha)             % alle Eigenschaften
get(ha, 'XScale')   % nur Eigenschaft: XScale

% neuer
ha.XScale
```

Die Werte der Eigenschaften können mit *set* verändert werden. Wenn *set* mit zwei Input-Argumenten verwendet wird, werden die möglichen Werte der Eigenschaft ausgegeben.

```
set(ha, 'XScale')   % zulässige Werte der Eigenschaft XScale
set(ha, 'XGrid')    % zulässige Werte der Eigenschaft XGrid

% verändern der Werte
set(ha, 'XScale', 'log')
set(ha, 'XGrid', 'on')

% neuer
ha.XScale = 'log'
ha.XGrid = 'on'
```

Tutorial 3.3

- Speichern von Grafiken

Wenn Grafiken später abgespeichert werden sollen, empfiehlt es sich schon bei der Erzeugung der figure auf die richtige Größe zu achten. Hierfür sollte die Einheit auf Zentimeter und eine feste Position definiert werden. Der Positionsvektor enthält die folgenden vier Werte:

- Abstand zum linken Bildschirmrand
- Abstand zum unteren Bildschirmrand
- Breite der *figure*

- Höhe der *figure*

```
szFontName = 'Palatino';
%szFontName = 'Arial';
hFig = figure(...
    'Units', 'centimeters', ...
    'Position', [3 3 10 8], ...
    'PaperPositionMode', 'auto')


...
    PLOTBEFEHLE
...


% finde alle Handles, die die Eigenschaft "FontName" haben:
hTemp = findall(hFig, '-property', 'FontName');
% bei diesen Handles die Schriftart setzen
set(hTemp, 'FontName', szFontName);

print(hFig, '-depsc2', 'grafik.eps');     % speichern als EPS
print(hFig, '-dpng', 'grafik.png');       % speichern als PNG
```

Allgemein sind eps-Dateien zu bevorzugen, da es sich hierbei um Vektorgrafiken handelt. Solche Grafiken können auch im Nachhinein noch vergrößert werden, ohne das sie unscharf werden.

Tutorial 3.4

- Laden von Daten
- Speichern von Daten

Den Befehl zum laden von Daten haben wir bereits verwendet:

```
load carsmall.mat
% oder
load('carsmall.mat')
```

Es lassen sich aber auch Variablen aus dem Workspace abspeichern:

```
save('daten.mat', 'VariablenName1', 'VariablenName2', 'VariablenName3')
```

Beispiel:

```
x = 0:0.01:2*pi;
y = sin(x);
save('sinus_daten.mat', 'x', 'y')
```

Trick:

Um bspw. sicher zu stellen, dass ein Vektor, der abgespeichert werden soll, ein Spaltenvektor (N x 1) ist, kann dieser Befehl verwendet werden:

```
data1 = data1(:);     % erzeugt Spaltenvektor
```

Entsprechend wird hier ein Zeilenvektor erzeugt:

```
data2 = data2(:).';   % erzeugt Zeilenvektor
```

Achtung: Auch Matrizen werden zu einem Vektor mit einer Spalte bzw. Zeile!

Tutorial 3: Aufgabe

Erzeugt eine Matrix, diesen Output zur Folge hat:

```
test =

     0     0     0     0     0     0     0
     0     1     1     1     1     1     0
     0     1     2     2     2     1     0
     0     1     2     3     2     1     0
     0     1     2     2     2     1     0
     0     1     1     1     1     1     0
     0     0     0     0     0     0     0
```