Tag 9: Handles und Wiederholung noch nicht gelöster Aufgaben.

Ein “handle” ist in Matlab eine Referenz auf ein Objekt oder ein Event. Der Hauptunterschied zwischen einem Objekt und einem handle ist:

- Bei der Übergabe einer „normalen“ Variablen an eine Funktion wird im Workspace der Funktion eine Kopie der Variablen angelegt. Diese wird wieder gelöscht, wenn die Funktion beendet wird. Wenn man möchte, dass innerhalb der Funktion vorgenommene Änderungen des Objekts sich auf die Variable im übergeordneten Workspace auswirken, muss die Funktion die Variable als Ausgabeparameter zurückgeben.
- Wenn man dagegen ein handle an eine Funktion übergibt, wird nicht das Objekt selber übergeben, sondern nur die Referenz darauf. Es wird keine Kopie angelegt, sondern innerhalb der Funktion direkt das „Original“ im übergeordneten Workspace bearbeitet. Entsprechend muss ein durch handle übergebenes, in einer Funktion verändertes Objekt nicht von der Funktion zurückgegeben werden.

Eine typische Anwendung für handles sind deshalb Datenbankanwendungen, in denen von verschiedenen Funktionen aus auf den gleichen, stets konsistent gehaltenen Datensatz zugegriffen werden und dieser Datensatz aus den Funktionen heraus verändert werden kann. Mit dieser (sehr schnell zum Thema objektorientierte Programmierung hinführenden) Anwendung selbst definierter handles werden wir uns heute aber nicht beschäftigen, sondern lediglich zwei wichtige und häufig benutzte von Matlab vordefinierte Arten von handles behandeln Grafik-handles und Funktions-handles.

Generelle Funktionen zum Umgang mit handles:

```
l = isa(h,'handle') % Allgemeiner Test, ob h ein handle ist
l = ishandle (h)    % Test, ob h ein Graphics handle ist
set(h,'Eigenschaftsname', Eigenschaftswert) % belegt die
                    % Eigenschaft des referenzierten Objekts
                    % auf den angegebenen Wert
v=get(h,'Eigenschaftsname') % gibt den Wert der Eigenschaft
                    % des referenzierten Objekts zurück.
delete(h)           % Löscht das Objekt, auf das handle h
                    % referenziert, aber NICHT die Variable h
clear h             % löscht das handle h aber NICHT das
                    % Objekt, auf das h referenziert.
l=isvalid(h)        % Überprüft, ob das Objekt, auf das handle
                    % h referenziert, (noch) existiert.
```

handle graphics

In Matlab wird die Graphic objektorientiert erzeugt. Eine Abbildung besteht aus einer Hierarchie von (mindestens) drei miteinander interagierenden Objekten:
Figure -> Axes -> Data Objects

Normalerweise werden diese Objekte nicht einzeln, sondern gemeinsam durch einen spezifischen Grafikbefehl wie `plot, line, scatter, bar` etc. erzeugt. Das Data Object (z.B. eine Linie) benötigt die ihm übergeordneten Achsen und das

Grafikfenster, in dem es erscheint. Wenn diese vorher nicht vorhanden waren, werden sie erzeugt. Wenn bereits ein aktives Grafikfenster vorhanden ist, wird es benutzt, ebenso die bereits eingestellten speziellen Eigenschaften der Achsen (z.B. größere Beschriftung).

data object (z.B. Linienobjekt zur Darstellung der Daten):

- Die eigentliche Darstellung der Daten erfolgt ebenfalls über ein Objekt vom Typ `line`. Diese Objekte werden durch die Funktionen `line` und `plot` erzeugt. Beide Befehle können ein handle auf das Objekt zurückgeben, wenn sie vom Benutzer entsprechend aufgerufen werden.

T9A1) Probieren Sie schrittweise aus:

```
figure
x=1:100;  y1=sqrt(x);  y2=x/5;
l1=plot(x,y1)
l2=line(x,y2)
```

Was ist der Unterschied zwischen den Befehlen `line` und `plot`? Tipp: Drehen Sie mal die Reihenfolge um...
Wenn Sie die Ausgaben nicht mit Semikolon unterdrücken, bekommen Sie einige der Eigenschaften angezeigt. Klicken Sie in der Ausgabe auf „show all properties“ um sich die Eigenschaften des data objects anzusehen.

- Ein data object (z.B. Linie) hat sehr viele Eigenschaften. Um diese Eigenschaften zu beeinflussen, gibt es mehrere Möglichkeiten:
  - Sie können (so wie sie das bisher schon gemacht haben) beim `plot`- bzw. `line`-Befehl die Eigenschaftsnamen und Werte als Argumente mit übergeben werden , z.B.
    `l3=plot(x,y,'LineWidth',3,'Color',[0,1,0.8])`
  - Sie können handles benutzen und wiederum mit dem Befehl set auch nachträglich die Eigenschaften des Objekts verändern:
    `set(l3,'LineStyle',':')`
  - Wenn Sie eine Abbildung nur einmalig verschönern wollen, können Sie dies auch interaktiv per Maus tun: Klicken Sie auf den Pfeil in der Menüleiste des Abbildungsfensters, klicken Sie dann doppelt auf die Linie, die sie verändern wollen. Dadurch bekommen Sie ein Menü, in dem die Linieneigenschaften eingestellt werden können. Allerdings ist es dann nicht mehr möglich, die genau gleiche Art der Abbildung automatisch noch einmal zu erzeugen.

T9A2) Ändern Sie einige Linieneigenschaften der beiden eben erstellten Linien auf die verschiedenen Arten. (Plotten Sie auch gerne noch zusätzliche Linien dazu, um die jeweiligen Auswirkungen zu sehen.)

axes object (Achsen):

- Die x-, y- und eventuell z-Achsen einer Abbildung stellen gemeinsam mit den Beschriftungen, den ‚tics' etc ein gemeinsames Objekt dar.
- Die Achsen (insbesondere ihre Skalierung) werden automatisch dem dargestellten data object angepasst.
- Ein Achsenobjekt kann auch durch den Befehl `axes` explizit erstellt und mit einem handle versehen werden:
  `axes  % Erzeugt Achsen`

```
h=axes          % Erzeugt Achsen und ein handle h darauf
axes(h)         % Macht die Achsen mit dem existierenden
                % handle h zu den aktuell aktiven Achsen.
h=gca           % erstellt ein handle zu den aktuell
                % aktiven Achsen
```

- Eigenschaften der Achsen können wie beim data object auch auf mehrere Arten verändert werden:
  - Sie können bei der Erzeugung von Achsen Eigenschaftsnamen und Werte als Argumente mit übergeben werden , z.B.
    `ha=axes('XLim',[0,100],'YLim,[1,5])`
  - Für die aktuellen Achsen können ebenfalls mit dem Befehl `axes` Eigenschaften verändert werden:
    ```
    axes('Eigenschaftsname',Wert) % setzt bei aktuellen
                % Achsen die Eigenschaft auf den Wert.
    ```
  - Sie können handles benutzen und wiederum mit dem Befehl set auch nachträglich die Eigenschaften des Objekts verändern:
    `set(ha,'FontSize',20)`
  - Für die aktuellen Achsen bekommen Sie ein handle mit dem Befehl `gca` (get current axes):
    ```
    ha2=gca     % erzeugt ein handle auf die aktuellen
                % Achsen
    ```
  - Außerdem gibt es für die Achsen spezielle Funktionen zur Einstellung der wichtigsten Achsenparameter, z.B. `xlim, ylim, grid, axis`
  - Wenn Sie eine Abbildung nur einmalig verschönern wollen, können Sie dies auch interaktiv per Maus tun: Klicken Sie auf den Pfeil in der Menüleiste des Abbildungsfensters, klicken Sie dann doppelt auf die Achse. Dadurch bekommen Sie ein Menü, in dem die Achseneigenschaften eingestellt werden können. Allerdings ist es dann nicht mehr möglich, die genau gleiche Art der Abbildung automatisch noch einmal zu erzeugen.

T9A3) Sehen Sie sich die Achsenparameter an und probieren Sie die verschiedenen Methoden aus, um diese zu ändern.

figure object (Grafikfenster):

- kann mit dem Befehl `figure` erzeugt werden.
- Das Fenster bekommt eine Nummer, entweder automatisch aufsteigend nummeriert in der Reihenfolge der Erzeugung oder vom Benutzer festgelegt, z.B. durch `figure(27)`.
- Um in ein bestimmtes bereits bestehendes Grafikfenster zu plotten, kann man dieses mit dem Aufruf von `figure(Nummer)` vor dem `plot`-Befehl zum aktiven Fenster machen.
- Interaktiv wird dasjenige Grafikfenster zum aktiven Fenster, auf das der Benutzer zuletzt geklickt hat.
- Das Grafikfenster hat zusätzlich zu seiner Nummer mehrere weitere Eigenschaften, z.B. seine Position und seine Farbe
- Wenn Befehl figure kann mit einer Ausgabe aufgerufen wird, gibt er ein handle auf das Graphikfenster-Objekt zurück: `h27=figure(27)`

T9A4) Erzeugen Sie ein neues Graphikfenster und lassen Sie sich das handle dafür zurückgeben. Ändern Sie einige der Eigenschaften des Graphikfensters mit dem Befehl `set(handlename,'Eigenschaftsname',Eigenschaftswert)`
z.B. `set(h27,'Color', [0.5 0 0.5])`
und schauen Sie sich die Auswirkung auf das Graphikfenster an.

function handles
Handles können auch für Funktionen benutzt werden, um diese indirekt aufzurufen. Dadurch ist es möglich, Funktionen an Funktionen zu übergeben und Funktionen in Strukturen abzulegen. Die Schlüsselbezeichnung, um ein function handle zu erzeugen ist das @ Zeichen

```
fh1=@functionsname        %Erzeugt ein function handle zu einer
                          %benannten Funktion, z.B.
fh1=@sqrt
fh2=@(eingebeargumente)anonymous_function   %Erzeugt ein
                          % function handle zu einer anonymen
                          % Funktion, z.B.
fh2=@(x)x.^2              % Funktion handle auf punktweise
                          % quadrieren
```

Funktionen können über function handles direkt oder mit `feval` aufgerufen werden:

```
[y1, y2, ...] = fhandle(x1, ..., xn)  % wendet die
                % Funktion, auf die fhandle referenziert, auf die
                % Eingaben x1 bis xn an.

[y1, y2, ...] = feval(fhandle, x1, ..., xn)     % wendet die
                % Funktion, auf die fhandle referenziert, auf die
                % Eingaben x1 bis xn an.
```

T9A5) Probieren Sie beide Arten des Aufrufs für die oben definierten function handles `fh1` und `fh2` aus. Erzeugen Sie entsprechend andere function handles und probieren Sie diese aus. Erzeugen Sie auch ein function handle für eine von Ihnen selber geschriebene Funktion in ihrem aktuellen Arbeitsverzeichnis.

Eine extrem praktische Anwendung von function handles ist der Befehl `cellfun`. Er wendet eine mit function handle referenzierte Funktion auf alle Zellen eines cell arrays an (das funktioniert allerdings normalerweise nur, wenn die Zellen alle mit Werten des gleichen Typs gefüllt sind).

```
[A1,...,Am] = cellfun(fhandle,C1,...,Cn) % wendet die
                % Funktion, auf die fhandle referenziert, auf jede
                % Zelle der cell arrays C1 bis Cn an. Dabei ist n
                % die Anzahl der von der Funktion vorgesehenen
                % Eingabeparameter. (Die Werte für den ersten
                % Eingabeparameter stehen also in C1, für den
                % zweiten in C2 etc.)

[A1,...,Am] = cellfun(fhandle,C1,...,Cn,'UniformOutput',false)
                % mit dieser zusätzlichen Angabe werden A1 bis Am
                % als cell arrays zurückgegeben, so dass auch
                % unterschiedliche Typen oder Matrixdimensionen als
                % Funktionsausgaben zurückgegeben werden können.
```

T9A6) Erzeugen Sie ein cell array C mit mehreren verschiedenen (auch verschieden langen) Vektoren. Wenden Sie darauf an:

```
averages = cellfun(@mean, C)
```

T9A7) Probieren Sie aus:

```
days = {'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'};
abbrev = cellfun(@(x) x(1:3), days, 'UniformOutput', false)
```

Was passiert hier?

Probieren Sie aus, ob das auch auf zweidimensionalen Cell Arrays funktioniert und was passiert, wenn einige der Zellen Vektoren enthalten. Was passiert, wenn man die Argumente `'UniformOutput', false` weglässt?

T9A8) Überlegen Sie sich ein Beispiel für eine Anwendung von cellfun auf eine von Ihnen selber geschriebene Funktion. Was passiert, wenn diese Funktion zwei Ausgabeargumente hat?