Matlab for PhD students – Advanced Topics 7 – WS 2010/11

"Graphics"

General consideration:

- Creating graphics with Matlab GUIs:
  - It is possible to create great figures with Matlab without knowing a single Matlab command. Ctrl-Click on a variable name in the workspace window opens a menu offering several different ways to graphically display the data. If you click on "more plots" you will get a full list of possible plotting options with nice pictograms of example plots**.** A single click on a plot name opens a help window, double click issues the corresponding plot command.
  - A click on the little arrow in the menu of the figure window allows to modify the figure properties by clicking on the figure:
  - Double clicking on the outer part of a figure window (outside of the axes) opens a menu in which e.g. the figure name and background color can be set. A click on "more properties" opens an inspector window with a long list of figure properties that can be set to different values by the user, controlling e.g. the size of the graphics window and the behavior when the figure is printed or saved to a file.
  - Double clicking in a graphics window inside of the axes opens a different menu in which e.g. axes labels and color of the background within the axes can be changed. The inspector offers a huge number of "axes properties" that can be controlled by the user.
  - Double clicking on the data graph in a graphics window opens a different menu to change the properties of the plot, e.g. line style, marker and color. The inspector offers the complete list of plot properties that can be changed.
  - However, if you want to produce graphics in a reproducible way it is highly advisable to use scripts or functions instead of GUIs. A very convenient way to produce such a script is to generate a nice plot using GUIs and then choose `File -> Generate M-File` from the menu of the graphics window. The resulting function opened in the Editor can be used as basis for your default program to produce plots like this.

- The Matlab graphics objects hierarchy:
  - In Matlab, graphics is organized as a hierarchy of objects. A `figure` is an object, which has specific properties (e.g. `Name, Colormap, OuterPosition`) and methods (e.g. `set, get, refreshdata, clf, close`). It can contain sub-objects, so-called children.
  - Figures are used to display data graphs or to contain graphical user interfaces. In this script I will only discuss figures to display data.
  - For a data graph `figure` object the direct child is an `axes` object. The properties of `axes` provide the "frame" of a graph plot, e.g. the limits and tics displayed on x- and y-axis, the background color of the figure, and the camera position used for 3D plotting. Many operations used for graphics are axes methods, e.g. `gca, axis, box, grid`
  - The data itself is contained in objects such as `line` or `surface`, which are children of `axes`.

- Matlab functions that draw graphics (e.g. `plot` or `surf`) create a `figure` automatically if none exist. If there are multiple figures open, one figure is always designated as the "current" figure, and is the target for graphics output.

- Handle Graphics:
  - Matlab graphics objects are handle objects. When an object is generated, it receives a specific number, the so-called handle, which Matlab uses to address the object. (In other languages this would be called a pointer). Graphics commands have these handle numbers as optional output arguments.
  - Graphics objects behave like all other handle objects in Matlab: Functions which get a handle of a graphics object as input argument can change the properties of this object (not of a copy of the object!) in the function.
  - Using the handle of a graphics objects, you can get and set the values of its properties. Matlab graphics objects have very many properties as you can see in the graphics inspector.
  - Some properties are structures or cell arrays.
  - Properties have default values that are used at creation if no other values are specified.

Manipulation of graphics objects:

- Handles: When a graphics object is created, it receives a handle as unambiguous identifier. The handle can be assigned to a variable. Alternatively, you can ask Matlab for the handle of the currently active graphics object. E.g.

```
Hfig=figure           % gives back a figure handle. Figures are
                      % consecutively numbered, the number is
                      % displayed in the header line
figure(Hfig)          % makes the figure with handle Hfig the
                      % current figure and displays it on top
                      % of other figures.
Hline=plot(x,y,x2,y2) % gives back a vector of handles, one for
                      % each individual line. The first line
                      % would be addressed by Hline(1).
Hfig=gcf              % get current figure handle
Haxes=gca             % get current axes handle
Hobject=gco           % get current object handle
get(H,'type')         % gets the type of object with handle H
```

  - Matlab numbers figure objects according to their sequence of creation. The number displayed in the title line of the figure window is the figure handle.
  - The handles of other graphics objects are more complicated numbers which do not have a meaning to the user.

- Current objects: Which object is the current object depends initially on the sequence of object generation. The most recently generated object is the current object. However, if you click on a graphics object it becomes the active object. Matlab uses stacks of objects, the topmost object in the stack is the current object. To find e.g. which object is the current one in a given figure with a handle named `Hfig` use

```
Hfig_kids= get(Hfig,'Children')    % axes objects
Hfig_kids_kids=get(Hfig_kids(1),'Children')   % Children of
                                   % the first child
```

- Display and set properties:
  - Inspector: If you double click on a graphics object, the property editor will open. In the property editor, the button more properties opens a window with all properties of the object.
  - Get:

```
get(H)          % displays a list of all object
                % properties of the object with handle
                % H, including the ones which cannot
                % be changed.
n=get(H,PropertyName)      % gets the value of the
                % specific property
```

  - Set:

    If more than one property is changed at a time, Matlab interprets the properties from left to right. Alternative to the set command you could use the property editor.

```
set(H)          % displays a list of all object
                % properties of the object with handle
                % H, which can be changed with set.
set(H,PropertyName)  % displays a list of possible values
                % for this property of the object with
                % handle H.
set(H,PropertyName,value)  % sets the value of the
                % specified property. Several
                % properties can be changed with one
                % function call of set.
```

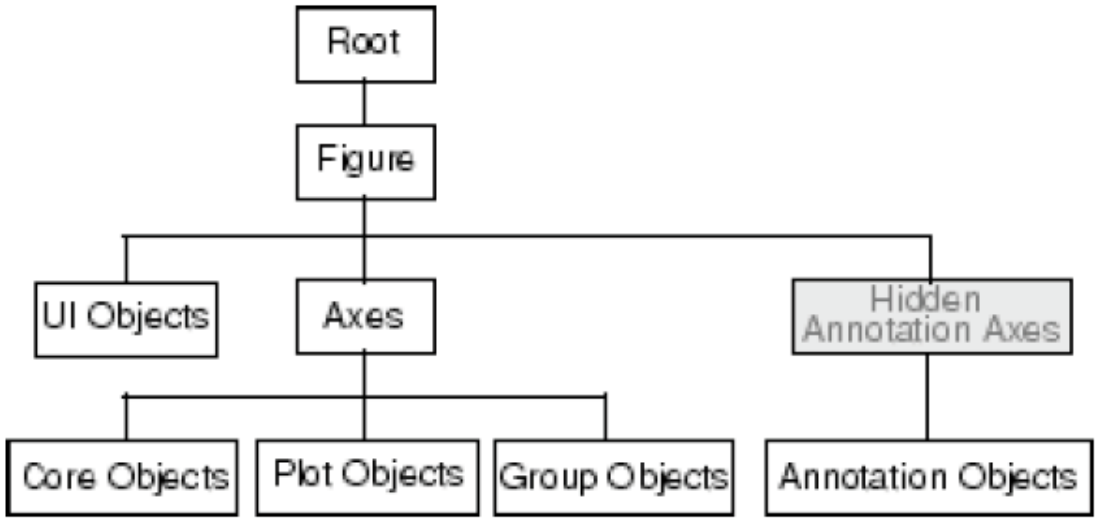
  - Set properties at generation: Most plot commands can be used with specified properties as additional optional input parameters. E.g.

```
plot(x,y,'LineWidth',3)    % sets line width to 3 points
                           % while plotting the line
```

- Property names: By convention, Matlab documentation capitalizes the first letter of each word that makes up a property name, such as `LineStyle` or `XTickLabelMode`. While this makes property names easier to read, Matlab does not check for uppercase letters. In addition, you need to use only enough letters to identify the name uniquely, so you can abbreviate most property names. In your code, however, using the full property name can prevent problems with futures releases of Matlab if a shortened name is no longer unique because of the addition of new properties.

Graphic objects hierarchy:

The generation and manipulation of graphics are organized as a hierarchy of graphics objects:



- `figure`: A figure object consists of the figure window and its children (menus, toolbars, user-interface objects, context menus and axes).
    - General properties of the figure object like `Position, Color, Units`,... are inherited by its children.
    - Specific figure properties are e.g. the name of the figure, properties concerning printing like `PaperSize` or `Renderer`, or properties concerning the window like `WindowStyle` or `WindowScroll`.

- `axes`: Axes objects define a frame of reference in a figure window for the display objects that are generally defined by data. The axes determines the location of each data point in the figure by defining axis scales (x, y, and z, or radius and angle, etc.)
    - A `figure` can contain several axes objects as children, e.g. if `subplot` was used. The `axes` of a figure can be arranged in various locations within the figure and can be of various sizes.
    - Axes inherit properties of the parent figure, e.g. the axes' position is measured relative to the figure position.
    - Specific properties of axes are e.g. `XTick, YLim, CameraPosition, GridLineStile`... Some properties which are inherited by children of axes are e.g. `FontSize` and `FontName`.
    - All functions that draw graphics (e.g., `plot`, `surf`, `mesh`, and `bar`) create an axes object if no axes objects exist.
    - If there are multiple axes within the figure, one `axes` object is always designated as the "current" `axes`, and is the target for display of graphics objects.

- Core objects: Core objects are generated directly by plotting commands. An `axes` object can have several core objects as children. They are the basic building blocks of graphics. Each of them has specific properties. Inherited properties like `FontSize` can be specifically changed for individual children. List of core objects:

```
axes            % Axes objects define the coordinate system for
                % displaying graphs. Axes are always contained
                % within a figure.
image           % 2-D representation of a matrix where numeric
                % values are mapped to colors. Images can also
```

```
                % be 3-D arrays of RGB values.
light           % Directional light source located within the
                % axes. Lights affect patches and surfaces, but
                % cannot themselves be seen.
line            % A line is drawn by connecting the data points
                % that define it.
patch           % Filled polygons with separate edge
                % properties. A single patch can contain
                % multiple faces, each colored independently
                % with solid or interpolated colors.
rectangle       % 2-D object that has settable edge and face
                % color, and variable curvature (can draw
                % ellipses).
surface         % 3-D grid of quadrilaterals created by
                % plotting the value of each element in a
                % matrix as a height above the x-y plane.
text            % Character strings positioned in the
                % coordinate system defined by the axes.
```

- Plot Objects: Plot objects contain several core objects as children, which were generated together by a specific plot command. By using the plot object, the properties of all its core objects can be changed together

```
areaseries      % generated by area
barseries       % generated by bar, bar3, barh, bar3h
contourseries   % generated by countour, countour3, countourf
errorbarseries  % generated by errorbar
lineseries      % generated by plot, plot3, semilogx, semilogy,
                % loglog
quivergroup     % generated by quiver, quiver3
scattergroup    % generated by scatter, scatter3
stairseries     % generated by stairs
stemseries      % generated by stem, stem3
surfaceplot     % generated by surf, mesh
```

- Group Objects: You can group several axes objects into one group object with the commands `hggroup` and `hgtransform`. Groups of axes generated with `hgtransform` can be scaled together by changing the `Matrix` property to the desired transform matrix (see help) which is then applied to all axes in the group. The command `makehgtform` facilitates the construction of such a matrix.

Axes and multiple figures:

- Figure windows:

```
figure          % opens new figure window
H=figure        % opens new figure window and gives back handle
figure(n)       % makes figure window n the current figure
clf             % clears the current figure window
clf(H)          % clears the figure window with handle H
clf reset       % clears the current figure window and sets all
                % properties to default
close(n)        % closes figure window n (number displayed as
                % title in figure window)
close all       % closes all figure windows
hold on         % keep same current figure, just add new line
```

```
                % with next plot command
hold off        % overwrite current figure with next plot
                % command
drawnow         % forces Matlab to update current figure
                % window, which would usually be done only
                % after several processing steps in a script or
                % function
```

- Multiple figures in one window: In subplots, the current axes are specified by numbering. E.g. `subplot(2,3,3)` has 2 rows and 3 columns of axes, the rightmost plot in the first row is active. All axes properties can be applied to individual subplots.

- Modify properties of current axes: The `axis` command is a more convenient way to change the properties of the current axes than using `set` and `get`. It has very many features, only some of them are listed here

```
axis([xmin xmax ymin ymax])   % sets axis limits on current plot
axis auto         % returns to automatic default
axis manual       % freeze axis to current limits when hold is on
axis equal        % sets aspect ratio so that tick mark
                  % increments are equal on all axes
axis square       % makes axis box square
axis ij           % matrix mode: vertical axis increases from top
                  % to bottom
axis xy           % Cartesian mode: vertical axis increases from
                  % bottom to top
axis off          % turns off background, tick marks and labels
axis on           % turns on background, tick marks and labels
```

- Colormap: Similar to the command `axis`, the command `colormap` provides a more convenient way to change the colormap of a figure and all of its children.

2D-plotting:

- The plot command: The `plot` command creates line objects and displays them in the current axes.
    - A plot command by default erases all line objects from the current axes and rescales the axes limits.
    - The command can be used with a great variety of input arguments and one optional output argument:

```
plot(y)                   % plots vector y versus index of y
plot(M)                   % plots for each column of Matrix M a
                          % line into the same figure
                          % (convenient way also to plot several
                          % vectors of the same length by
                          % combining them into a matrix)
plot(x,y)                 % plots vector y versus vector x
plot(x1,y1,x2,y2)         % plots two lines in one figure: y1
                          % versus x1 and y2 versus x2
plot(x,y,'r:-')           % plots y versus x with specified line
                          % style (see line specifiers below)
plot(x,y,'LineWidth',3)   % plots with specified properties
plot(axes_handle,x,y)     % plots into specified axes instead of
                          % gca
h=plot(x,y)               % returns a vector of handles, one for
                          % each line
```

- More plotting commands creating line objects:

```
plotyy(x1,y1,x2,y2)        % linear plot with two separate y-axes
loglog(x,y)                % double-logarithmic plot
semilogx(x,y)              % plot with logarithmic x-axes
semilogy(x,y)              % plot with logarithmic y-axes
errorbar(x,y,e)            % plots y versus x with symmetric
                           % errorbars e
errorbar(x,y,l,u)          % plots y versus x with lower
                           % errorbars l and upper errorbars u
line(x,y)                  % creates a line object like plot but
                           % does NOT erase other line objects
                           % created with the line command or
                           % rescale the axes.
```

- Line specifiers: Short way to specify line properties when using the plot command or some of the other 2D and 3D plotting commands. Sequence of specifiers for up to three properties: `Color, LineStyle` and `Marker`. The order of the properties does not matter.
  - Color: `r (red), g (green), b (blue), c (cyan), m (magenta), y (yellow), k (back), w (white)`
  - LineStyle: `- (solid), -- (dashed), : (dotted), -. (dash-dotted)`
  - Marker: `+ (plus sign), o (circle), * (asterisk), . (point), x (cross), s (square), d (diamond), ^ (upward-pointing triangle), v (downward-pointing triangle), < (left-pointing triangle), > (right-pointing triangle), p (pentagram), h (hexagram)`

- Other types of plotting commands creating plot objects:

```
scatter(x,y)      % scatter plot of y versus x
scatter(x,y,s,c) % scatter plot of y versus x with specified
                  % size of each point (in vector s) and color of
                  % each point (specified in vector c according
                  % to current color map, matrix c with rgb
                  % values or string c with color specifiers)
plotmatrix(M)     % plots a matrix of scatter plots displaying
                  % each column of M versus each other column and
                  % a histogram on the diagonal.
plotmatrix(M1,M2)% plots a matrix of scatter plots displaying
                  % each column of M2 versus each column of M1.
hist(y)           % histogram of values in y, divided in 10 bins
hist(y,n)         % histogram of values in y, divided in n bins
hist(y,centers)   % histogram of values in y, divided in bins
                  % specified by vector of bin centers
bar(y)            % draws one bar for each element in vector y
bar(M)            % groups the bars produced by the elements in
                  % each row of M
bar(x,y)          % draws values of y as bars at positions
                  % specified by x
bar(…, width)     % specifies the width of the bars (default 0.8)
barh              % like bar, but with horizontal bars
bar3              % like bar, but 3 dimensional bars
bar3h             % like bar, but 3 dimensional horizontal bars
pie(x)            % pie chart of vector x
pie(x,explode)    % pie chart with specified segments offset from
                  % center, explode is vector of 1 and 0.
```

```
pie(x,{'text1','text2','text3'})  % pie chart with labels
pie3              % like pie, but 3 dimensional
rose(theta)       % round graph showing distribution of angles
rose(theta,n)     % round graph showing distribution of angles in
                  % n bins
rose(theta,centers)  % round graph showing distribution of
                  % angles in bins specified by vector of bin
                  % centers
compass(u,v)      % round graph with arrows with base at origin
                  % and tip at point [u(i),v(i)] relative to base
polar(theta,rho) % polar plot of Cartesian coordinates
area(x,y)         % plots y versus x and fills the area between
                  % 0 and the curve
fill(x,y,c)       % fills polygon specified by points x(i),y(i)
                  % with color c.
```

3D plotting:

- 3D line plotting: `plot3` is the 3D line plot corresponding to `plot`. It expects a triplet of vectors for each line: `plot3(x1,y1,z1,x2,y2,z2)` All axes properties apply also to the z-axis and can be modified accordingly. By changing the camera position (either by mouse or by handle), the line object can be moved and turned in 3D space.

- 3D surface plotting: If a function of two variables `z=f(x,y)` is plotted, it usually is desirable to produce a surface plot. `z` is a matrix of the dependent variables and is given by `z(i,:)=f(x,y(i))` and `z(:,j)=f(x(j),y)`.
  - Produce convenient matrices for plotting: x- and y-axis matrices can be created by the function `meshgrid`. E.g.
    ```
    x=-3:3; y=1:5; [X,Y]=meshgrid(x,y); Z=(X+Y).^2
    ```
  - Commands for surface plotting:
    ```
    mesh(X,Y,Z)         % meshgrid plotting
    meshz(X,Y,Z)        % meshgrid with zero plane
    surf(X,Y,Z)         % surface plot, like mesh with filled
                        % spaces
    shading flat        % varies appearance of surf-plot
    shading interp      % interpolated shading of surface plot
    surfc(X,Y,Z)        % surf plot with contours
    ```

Text in figures:

- Add texts:
```
xlabel('text')            % x-axis label
ylabel('text')            % y-axis label
zlabel('text')            % z-axis label in 3D plot
title('text')             % title at top of plot
legend('y1','y2')         % legend (see below for properties)
text(x,y,'text')          % places text at coordinates (x,y)
gtext('text')             % places text at position of mouse
                          % click in figure
gtext('text1','text2')    % places multiple lines of text at
                          % position of mouse click in figure
[x,y]=ginput              % get coordinates of button click in
                          % figure (not really a command to
                          % produce graphics, but convenient e.g.
                          % to position text)
```

- Text formatting:
  - for multiline texts use string arrays, e.g.
    `xlabel({'first line','second line'})`
  - special characters like Greek letters and mathematical symbols are included by using latex formatting strings e.g. `'\alpha', '\Sigma','\circ', '\leq', '\infty'`. For the full list see "text properties" in the Matlab help. (German Umlaute are still not supported as far as I know)
  - For subscripts and superscripts also use latex commands: `'A_{ij}', 'x^2'`
  - Texts can be formatted with latex commands, e.g.
    `text(0.2,0.4,'\fontsize{30}\Sigma\fontsize{16}\fontname{courier}\bf x_i')`

- Text properties: specifications of text properties can be included for all text commands, some text commands have additional special properties. E.g. `title('text','FontSize',30)`. For more properties, see help or property editor. Some important properties and their ranges of possible values are:
  - `'FontSize',pixels`
  - `'FontWeight','light' / 'normal' / 'bold'`
  - `'FontName', 'FixedWidth' / 'Courier' / 'Helvetica'` (others depending on system used)
  - `'FontAngle','normal' / 'italic'`
  - `For text command: 'HorizontalAlignment', 'left' / 'center' / 'right'`
  - `For legend command: 'Location', 'North' / 'SouthEast' / 'WestOutside' / … / 'Best' / 'BestOutside'`

Exporting figures:
Exporting figures from Matlab is often difficult, because what you see is usually not what you get, because the export results depend on many properties of the graphics objects that only become relevant for the export…

- From the command line, a script or function, the most convenient way to export a figure is the `print` command. Please refer to the help page for the many, many options. It supports several graphic file formats, including the vector graphics eps and ill (Adobe illustrator) and the bitmap formats bmp and jpeg. Unfortunately, pdf is not supported (at least up to version 2010a).
- Many properties of the exported figure can be different from on the screen if the object properties (e.g. `XTickMode` or `PaperPositionMode`) are set to `auto` instead of `manual`. In these cases, the renderer evaluates the properties.
- There are three renderers: `painters, zbuffer` and `OpenGL`. If `RendererMode` is set to `auto` Matlab will choose the renderer depending on the file format of the exported figure. HPDGL and Adobe Illustrator output formats use `painters`, JPEG and TIFF use `zbuffer`, if transparency is included in the figure, `OpenGL` should be used.
- A helpful function `savefigure.m` with its sub-function `parseArgs.m` written by Aslak Grinsted can be downloaded from:
  http://www.mathworks.com/matlabcentral/fileexchange/6854
  It e.g. supports exporting a figure in a given size, which is required by many publishers. (There might be more up-to-date versions for newer Matlab

releases by now.)

Graphics Callbacks

- A callback is a function that executes when a specific event occurs on a graphics object. You specify a callback by setting the appropriate property of the object.

- Graphics Object Callbacks: All graphics objects have three properties for which you can define callback routines:

```
ButtonDownFcn          % Executes when you click the left mouse
                       % button while the cursor is over the
                       % object or within a 5-pixel border around
                       % the object.
CreateFcn              % Executes during object creation after you
                       % set all properties.
DeleteFcn              % Executes just before deleting the object.
```

- Figure Callbacks: Figures have additional properties that execute callback routines with the appropriate user action. Only the CloseRequestFcn property has a callback defined by default:

```
CloseRequestFcn        % Executes when a request is made to close
                       % the figure (by a close command, by the
                       % window manager menu, or by quitting
                       % Matlab).
KeyPressFcn            % Executes when you press a key while the
                       % cursor is within the figure window.
ResizeFcn              % Executes when you resize the figure
                       % window.
WindowButtonDownFcn    % Executes when you click a mouse button
                       % while the cursor is over the figure
                       % background.
WindowButtonMotionFcn  % Executes when you move the mouse button
                       % within the figure window (but not over
                       % menus or title bar).
WindowButtonUpFcn      % Executes when you release the mouse
                       % button, after having pressed the mouse
                       % button within the figure.
```

Homework:

- Try to export a figure from your own work into a convenient data format and import it into your favorite program for text processing.
- If you use standard plots to evaluate your data, write scripts or functions to generate them in a convenient way.