Introduction to Biomedical Engineering

Section 4: Basics of High-level programming: Matlab

Lecture 4.2 Visualisation, serial interface, filters

POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

OPEN
POLYTECH

Basic Plotting

- You can create a **basic line graph** in MATLAB using the function '**plot**'
- **'plot(y)'** will plot your data (**y**) against an arbitrary (**numbered**) x-axis
- **'plot(x, y)'** will plot your data (**y**) against the corresponding x-axis locations in **x**
- If y is a **matrix**, the '**plot**' function will display all **columns** of y as **separate lines**
- For data is **arranged in rows, use the apostrophe** shortcut to **transpose the matrix**
- You can create a **basic line graph** with **error bars** using the function '**errorbar**'
- '**errorbar(x, y, SE)**' will plot data with **error bars** whose **length is specified by SE**
- **Scatter plots** can be created in MATLAB using the '**scatter**' function
- The '**scatter**' function requires a minimum of **two inputs** (i.e. both **x** and **y** data)

Basic Figure Handling

- The '**plot**' function automatically generates a **new figure window**, if one is not open
- The '**plot**' function automatically overwrites an **existing figure window**, if one is open
- To **open** an **additional figure window**, use the '**figure**' command
- To **add lines** to an **existing figure**, use the '**hold on**' / '**hold off**' commands
- You can **close figures** individually with the '**close'** command
- Note that this will **close** the **most recently active figure**
- You can **close all figures** with the '**close all**' command

Axis Limits

- MATLAB **automatically scales** figure axes to **accommodate the data** being plotted
- Axes limits can be **manually controlled** using '**xlim([min max])**' and '**ylim([min max])**'
- The '**axis**' function can also be used adjust axes in several different ways
- '**axis tight**' fits the axes limits to the data exactly
- '**axis square**' makes the figure axes square, for display purposes
- '**axis off**' removes the axes, labels and background, for display purposes
- Each of these axis limit commands can be used on **any MATLAB plot**

Figure Text

- There are **several functions** for adding **labels**, **titles** and **text** to figures
- Axes can be labelled using the functions '**xlabel(*'string'*)**' and '**ylabel(*'string'*)**'
- Titles can be added to a figure using the function '**title(*'string'*)**'
- Text can be added to a figure using the function '**text(x_position, y_position, *'string'*)**'
- Text added by these functions can also be **formatted directly from the command line**
- This is achieved by passing **additional inputs** to the function
- For example: **title(*'string'*, 'FontSize', 24)** or **title(*'string'*, 'FontName', 'Times')**

Editing Figure Properties I

- There are **two ways** to **edit** the **properties of a figure**
- The first is through the **figure window toolbar**, by clicking **Edit -> Figure Properties…**
- This brings up an **additional panel** at the **bottom** of the **figure window**
- **Clicking** on **various parts** of the figure then allows you to **edit properties directly**
- For example, **click on or near the axes**, and you can edit the **labels, font and limits**
- Clicking **on the plotted line** allows you to edit the **colour, line style and marker style**

Editing Figure Properties II

- The **second way** to **edit a figure** is **directly from the command line**
- **Basic properties** can be defined **when first creating the plot**
- These include **LineWidth, LineStyle**, and **Color** for lines…
- …and **Marker**, **MarkerEdgeColor**, **MarkerFaceColor** and **MarkerSize** for markers
- For example: **plot(x, y, 'LineWidth', 3, 'Color', 'r', 'LineStyle', '--')**
- The **colour**, **line** and **marker style alone** can be defined in a **single set of apostrophes**
- For example: **plot(x, y, 'r--*')**

Editing Figure Properties II

MATLAB 'Color' Code

'b' = Blue (default)
'g' = Green
'r' = Red
'k' = Black
'y' = Yellow
'w' = White
'c' = Cyan
'm' = Magenta

MATLAB 'LineStyle' Code

'-' = Solid line (default for plot)
'--' = Dashed line
'-.' = Dash-dot line
':' = Dotted line
'none' = No line

MATLAB 'Marker' Code

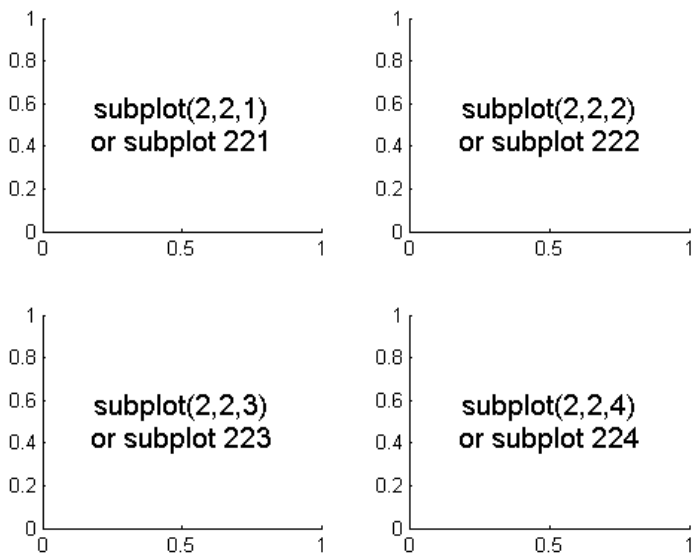
'o' = Circle (default for scatter)
'.' = Point
'x' = Cross
'+' = Plus sign
'*' = Asterisk
's' = Square
'd' = Diamond
'v' = Down triangle
'^' = Up triangle
'>' = Right triangle
'<' = Left triangle
'p' = Pentagram
'h' = Hexagram
'none' = No marker (default for plot)

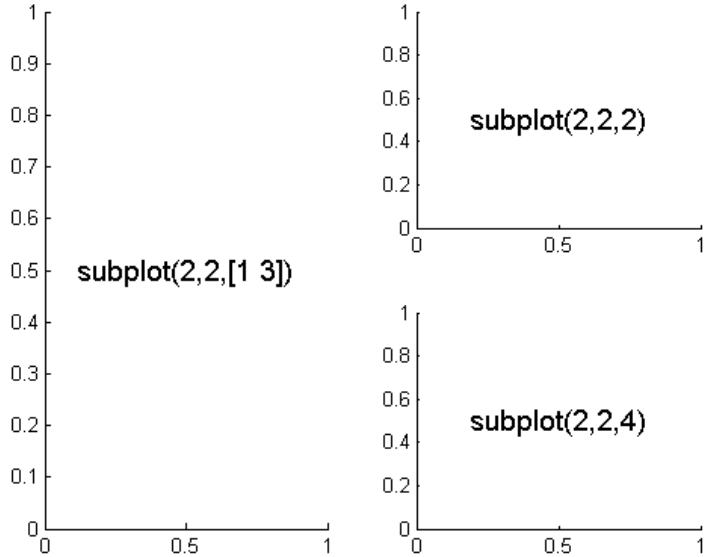
Editing Figure Properties III

- More **complex properties** can be adjusted using the '**set**' function
- The general syntax for this is '**set (FigureHandle, 'Property_Name', Value)**'
- For example: **set(gca, 'TickDir', 'out', 'Xscale', 'log')**
- There are **a number** of **different properties** that can be **changed** in this way
- These can be **viewed** by **searching** for '**Axes Properties**' in MATLAB help
- The '**figure handle**' **identifies the figure** that we wish to **edit**
- **For now**, we will use the **shortcut** '**gca**', which **specifies the 'current' figure**

Subplots

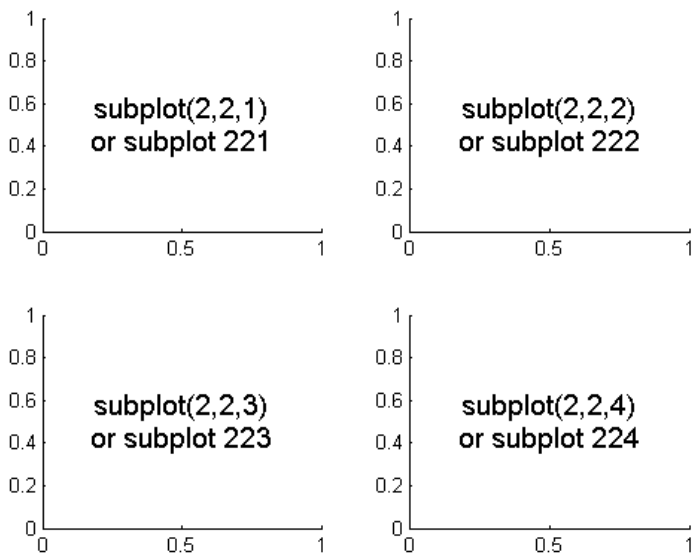
- Several **different panels** can be plotted **within a single figure** using '**subplot**'
- The syntax is: **subplot(Panel_Rows, Panel_Columns, Panels_to_be_used)**
- Panels are **numbered along each row**
- For example, **subplot(2,2,1)** sets up a **2 x 2 square of panels**, and **plots in the first**
- Plots can also be **spread across multiple panels**

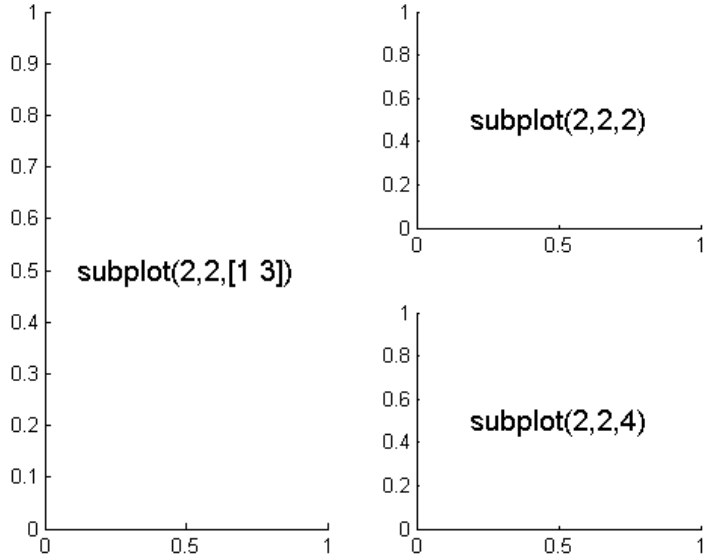




Advanced Figure Handling

- When drawing **subplots**, **figure handles** can be used to **control each panel** separately
- This means **assigning a variable** to **each panel**
- This is achieved by **providing** the '**subplot**' function with an **output variable**
- For example, '**h1 = subplot(2,2,1)**'
- **Properties** of that **specific panel** can then be **controlled** using '**set(h1, …)**' etc.





Bar Charts

- **Vertical** and **horizontal bar charts** can be created using the '**bar**' and '**barh**' functions
- Like plot, '**bar(y)**' plots the data in **y** on an **arbitrary x-axis** of 1 to length(y)…
- …while '**bar(x, y)**' plots the data in **y** at the x-axis **positions defined in x**
- The width of the bars can be defined by a third input, i.e. '**bar(x, y, width)**'
- **Other properties** can be **defined** using the **'PropertyName', Value** syntax from above
- For example: **bar(x, y, 'FaceColor', 'r', 'EdgeColor', 'k')**
- A **list** of **bar chart properties** can be found by searching for '**Barseries properties**'

Histograms

- **Histograms** can be created using the '**hist**' function
- Providing a **single input** will plot a histogram with **10 evenly spaced bins**, i.e. '**hist(y)**'
- The **second input** can be used to define the **centre of the x-axis bins**, i.e. '**hist(y,x)**'
- …or to define the **total number** of **evenly spaced bins**, i.e. '**hist(y,n)**'
- Note that the **syntax differs** from '**plot**' – input the **data first**, **x-axis positions second**

Contour Plots and Images

- **Two dimensional plots** can be created using '**contourf**' and '**imagesc**'
- Passing a **single (matrix) input** defines the **colour** of **each point** on the graph
- For example, '**contourf(c)**' or '**imagesc(c)**'
- Alternatively, **x** and **y axis locations** for **each colour point** can be provided
- For example, '**contourf(x, y, c)**' or '**imagesc(x, y, c)**'
- The '**contourf**' function also allows you to **specify how many z gradations** are plotted
- For example, '**contourf(x, y, c, n)**'
- Note that the '**imagesc**' function **automatically inverts** the **y-axis**
- This can be corrected using **set(gca, 'Ydir', 'normal')**

Other Useful Plotting Functions

- There are **many different plotting functions** within MATLAB
- **Each** of these **can be controlled** using the **general syntax** discussed above
- **Information** on the **specific syntax** used for each function can be found in the '**Help**' file
- Some **examples include**:
- '**plotyy**' – allows you to plot **multiple lines** on one figure with **different y-axes**
- '**pie**' – allows you to plot a **pie chart** (with one **emerging segment**!)
- '**loglog**', '**semilogx**', '**semilogy**' – allows you plot one or more **logarithmic axes**
- '**rose**' – allows you to plot a **circular histogram**
- '**surf**' – allows you to plot a **3D surface** (like the **MATLAB logo!**)

Other Useful Plotting Tools

- There are several buttons at the top of the Figure window that may be useful
- The '**Data Cursor**' returns the **x** and **y co-ordinates** of any point clicked on
- The '**Insert Legend**' button inserts an **editable legend** to annotate your figure
- The '**Insert Colorbar**' button adds a '**colorbar**' (i.e. **z-axis scale**) to your figure
- There are also buttons for **zooming in** and **out** of your figure
- Finally, it is important to note that you can **save figures** in **several formats**
- These include *.**bmp**, *.**jpg**, various **Adobe Illustrator formats** etc.
- Figures **saved** in the **native** MATLAB *.**fig** format can be **re-opened and edited**

Programming practices: Basic visual aspects

- Use meaningful file/variable names (within reason)
- Use spacing and white-space to write clean and easy to parse code
- Alignment (within reason)
- Indentation – Matlab will allow you to break the conventions – Don’t!
- Take advantage of Matlab editor features like cells and code-folding

-> Let’s look at some examples

Developing and organizing your code 1

- Assign variables early, make your code easily extendable/adaptable

- Go from simple to complex: If you are faced with a complex issue, develop a toy model first.

- Modular programming: create functions when possible
  -> cleaning up your code
  -> creating reusable scripts/functions
  -> separating conceptually distinct problems

- Backup your code, with a meaningful naming conventions (insurance against computer failure)

- Backup working code before trying out new things (insurance against stupidity)

Developing and organizing your code 2

- Make extensive use of comments !
  -> Making your code understandable to others that might use it
  -> And to your future self!

- Matlab does not necessarily return errors when it does not do what you intended
  -> These are usually the most insidious bugs.
  -> Don't have faith! Always test your code with simple examples

-> Let’s look at some code

Memory and Performance

- Optimizing your code is important for large projects. A very wide topic.
- But even for smaller projects a few basic guidelines should be followed.
- Pre-assign arrays and don’t let them grow inside of loops
- Avoid loops when possible (are slow), favor matrix algebra
- Favor logical operators over the built-in functions “find” and “nonzeros”
- Consider using sparse matrices (but sparse indexing is slow)

-> code examples

Matlab serial interface – convenient communication to your microcontroller

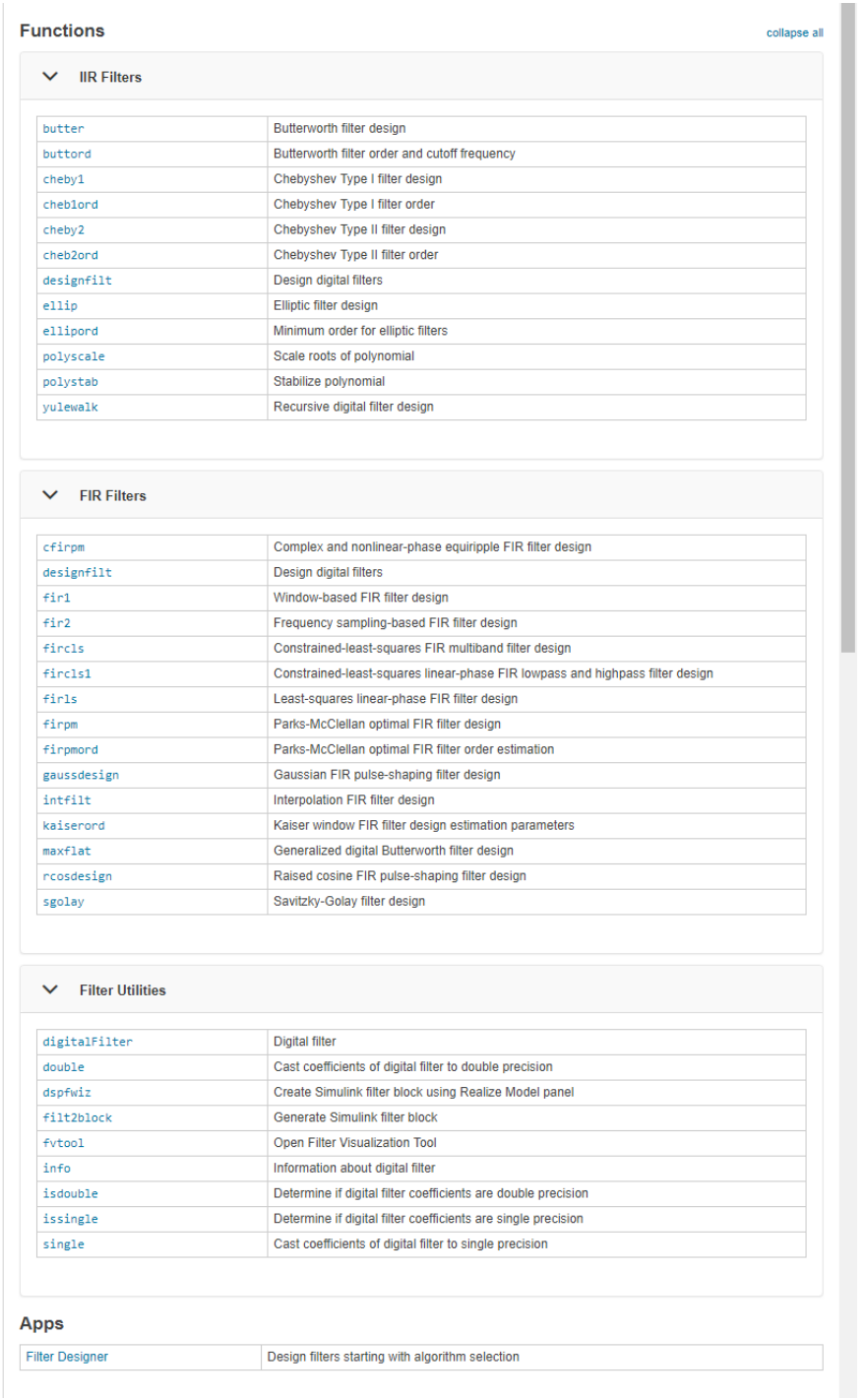
```
s = serial('COM1');                 % Creates serial object
set(s,'BaudRate',4800);             % sets Baud Rate to 4800
fopen(s);                           % opens serial port
fprintf(s,'*IDN?')                  % sends ‘who are you?’ command
out = fscanf(s);                    % reads from buffer into variable  out
fclose(s)                           % closes the port
delete(s)                           % deletes the object
clear s                             % clears the variable
```

Filters

F = designfilt;

Does the job for most cases

If this is not enough, then you already know what you are doing!



Functions collapse all

IIR Filters

butter	Butterworth filter design
buttord	Butterworth filter order and cutoff frequency
cheby1	Chebyshev Type I filter design
cheb1ord	Chebyshev Type I filter order
cheby2	Chebyshev Type II filter design
cheb2ord	Chebyshev Type II filter order
designfilt	Design digital filters
ellip	Elliptic filter design
ellipord	Minimum order for elliptic filters
polyscale	Scale roots of polynomial
polystab	Stabilize polynomial
yulewalk	Recursive digital filter design

FIR Filters

cfirpm	Complex and nonlinear-phase equiripple FIR filter design
designfilt	Design digital filters
fir1	Window-based FIR filter design
fir2	Frequency sampling-based FIR filter design
fircls	Constrained-least-squares FIR multiband filter design
fircls1	Constrained-least-squares linear-phase FIR lowpass and highpass filter design
firls	Least-squares linear-phase FIR filter design
firpm	Parks-McClellan optimal FIR filter design
firpmord	Parks-McClellan optimal FIR filter order estimation
gaussdesign	Gaussian FIR pulse-shaping filter design
intfilt	Interpolation FIR filter design
kaiserord	Kaiser window FIR filter design estimation parameters
maxflat	Generalized digital Butterworth filter design
rcosdesign	Raised cosine FIR pulse-shaping filter design
sgolay	Savitzky-Golay filter design

Filter Utilities

digitalFilter	Digital filter
double	Cast coefficients of digital filter to double precision
dspfwiz	Create Simulink filter block using Realize Model panel
filt2block	Generate Simulink filter block
fvtool	Open Filter Visualization Tool
info	Information about digital filter
isdouble	Determine if digital filter coefficients are double precision
issingle	Determine if digital filter coefficients are single precision
single	Cast coefficients of digital filter to single precision

Apps

Filter Designer	Design filters starting with algorithm selection

Filters

```
F = designfilt;

Filtered = filtfilt(F, Unfiltered);
```

Thank you for your attention!

Some graphic material used in the course was taken from publicly available online resources that do not contain references to the authors and any restrictions on material reproduction.

POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

OPEN
POLYTECH