MRC | Cognition and Brain Sciences Unit

Visualising data using MATLAB

Lydia Barnes

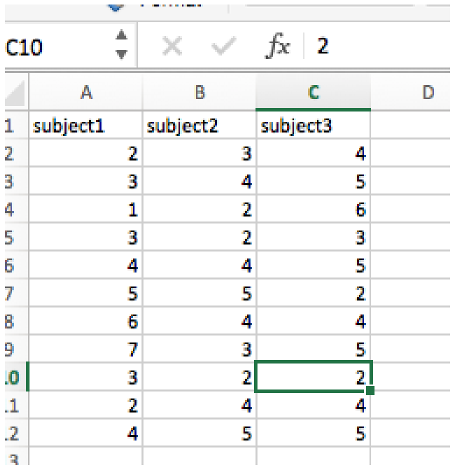
MRC Cognition and Brain Sciences Unit

8th November 2019

(slides from Sneha Shashidhara and Kate Storrs!)

Good practice for plotting

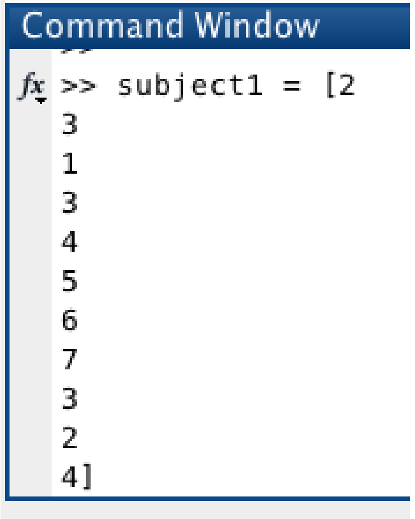
How not to use MATLAB to visualise data



C10 | fx | 2

	A	B	C	D
1	subject1	subject2	subject3	
2	2	3	4	
3	3	4	5	
4	1	2	6	
5	3	2	3	
6	4	4	5	
7	5	5	2	
8	6	4	4	
9	7	3	5	
.0	3	2	2	
.1	2	4	4	
.2	4	5	5	

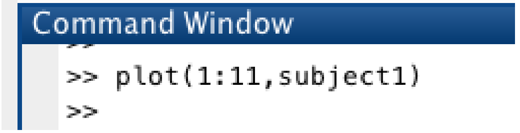
Type data into Excel…



Command Window

```
>> subject1 = [2
3
1
3
4
5
6
7
3
2
4]
```

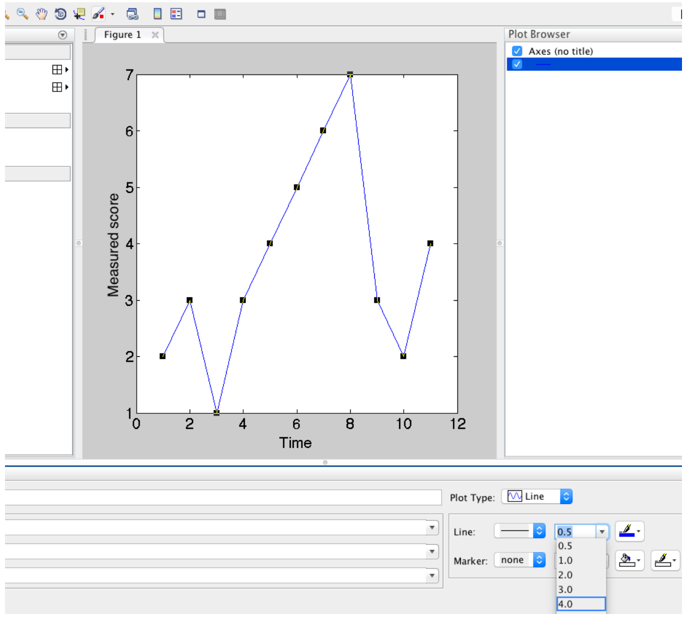
…copy-paste into Matlab…



Command Window

```
>> plot(1:11,subject1)
>>
```

…type in command line instructions to plot…



…fiddle with plot using interactive plotting interface until satisfied.

What's wrong with this?

Good practice: help your future self

- Assume you will forget everything!
- Comment
  - Intro: %this script was made to…
  - Brief note on each line or chunk: %load the data, %reformat the plot…
- Structure the code
  - clear your workspace
  - set parameters
  - load data
  - then make plots…
- Move from concrete to flexible
  - function [out] = makeMePlots(dataDirectory,plotDirectory,includePValues)

Good practice: make your process transparent

- What is Open Science?
  - Reporting what you’ve done
  - Sharing every step
- Why share scripts?
  - Limited room for explanation in a Methods section
  - Scripts hold all the detail you need!
- Reproduceable plotting
  - Script everything – including loading the data
  - Comment everything
    - Name and contact details
    - Background on the data at the top of the script, or in a ‘README’ in the script’s folder

Good practice: some resources

‘Good enough practices in scientific computing’, Wilson et al. 2017

‘Everything is f-ed’ open science syllabus, Sanjay Srivastava (@hardsci)

Starting a script

Starting a script: exercise

- open a new script
- save it
  - meaningful location, meaningful name…
  - avoid clashing with built-in function names (mean, max, plot…)
- intro
  - % for comments (things MATLAB won’t try to run)
  - your info
  - what will this script do?

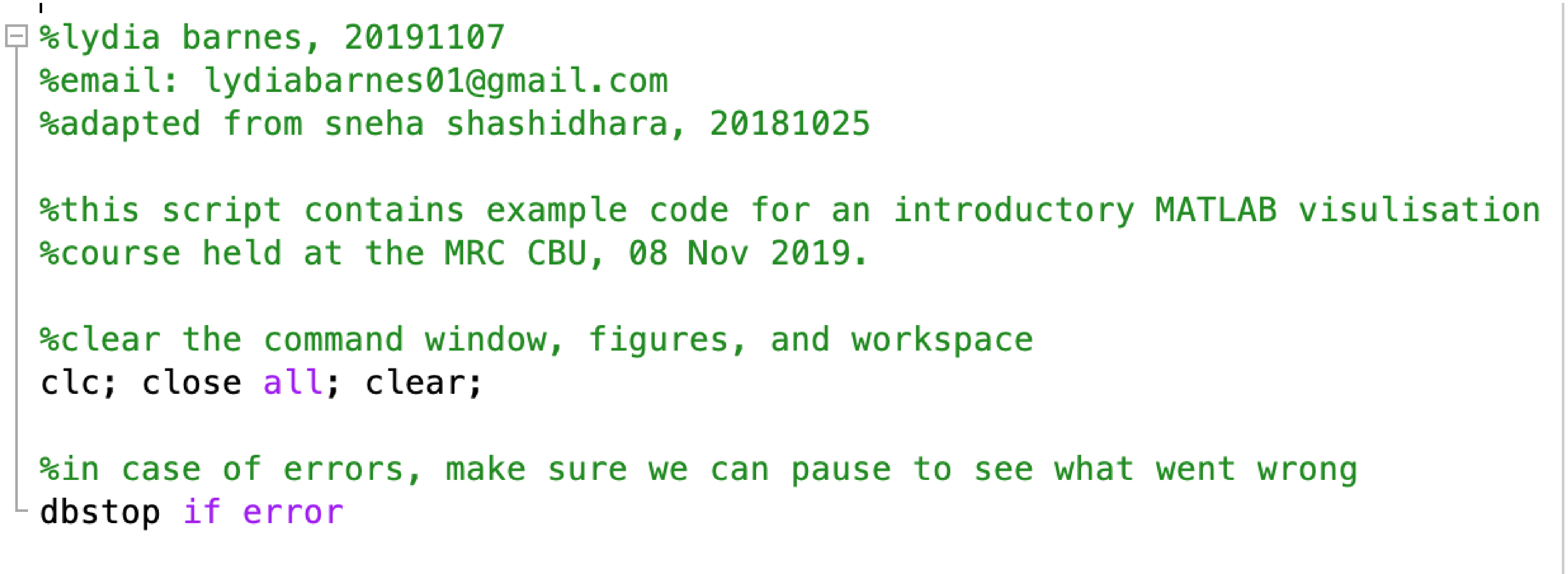
Starting a script: tips

- some safety checks
  - clean up your working environment to prevent clashes
  - prepare for errors

```
clc; close all; clear; %clear the command window, figures, and workspace
```

```
dbstop if error; %if there’s an error, pause at that line
```

Starting a script: example



```
%lydia barnes, 20191107
%email: lydiabarnes01@gmail.com
%adapted from sneha shashidhara, 20181025

%this script contains example code for an introductory MATLAB visulisation
%course held at the MRC CBU, 08 Nov 2019.

%clear the command window, figures, and workspace
clc; close all; clear;

%in case of errors, make sure we can pause to see what went wrong
dbstop if error
```

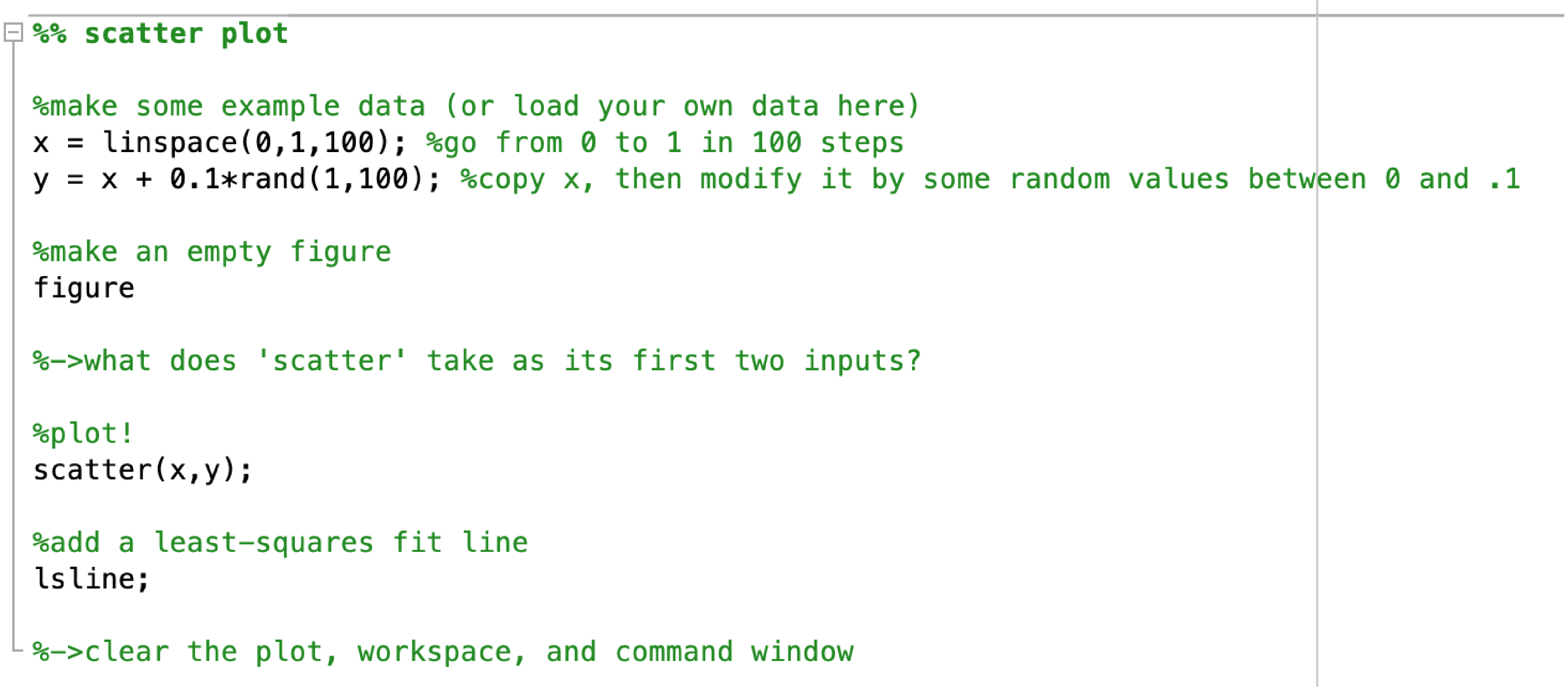
Scatter plots

Scatter plot: exercise

- make (or load) the data
  - make y relative to x
- “figure”
  - add to your script and run it!
- explore the scatter function
  - “help scatter” or “doc scatter” in the command window
- plot
  - “scatter(x,y)”
- “lsline”

```
x = linspace(0,1,100); %go from 0 to 1 in 100 steps

y = x + 0.1*rand(1,100); %copy x, then modify it by some random values between 0 and .1
```

Scatter plot: example



```
%% scatter plot

%make some example data (or load your own data here)
x = linspace(0,1,100); %go from 0 to 1 in 100 steps
y = x + 0.1*rand(1,100); %copy x, then modify it by some random values between 0 and .1

%make an empty figure
figure

%->what does 'scatter' take as its first two inputs?

%plot!
scatter(x,y);

%add a least-squares fit line
lsline;

%->clear the plot, workspace, and command window
```

Line plots

Line plot: exercise

- make (or load) the data
  - we’ll use the sine and cosine of the same data vector
- ‘figure’
- explore the ‘plot’ function
  - “help plot”
- plot
  - plot(x,y)

```
x = linspace(0,360,100); %0:360 in 100 steps
y = sind(x);
z = cosd(x);
```

Line plot: tips

- store the figure information
  - look inside the handle. what are its properties?
- make variables for different colours
  - make black
  - define your own three colours…

```
h = figure; %open a figure, and store a ‘handle’ to it in a variable
```

```
red=[1 0 0];
green=[0 1 0];
blue=[0 0 1];
black=[0 0 0];
```

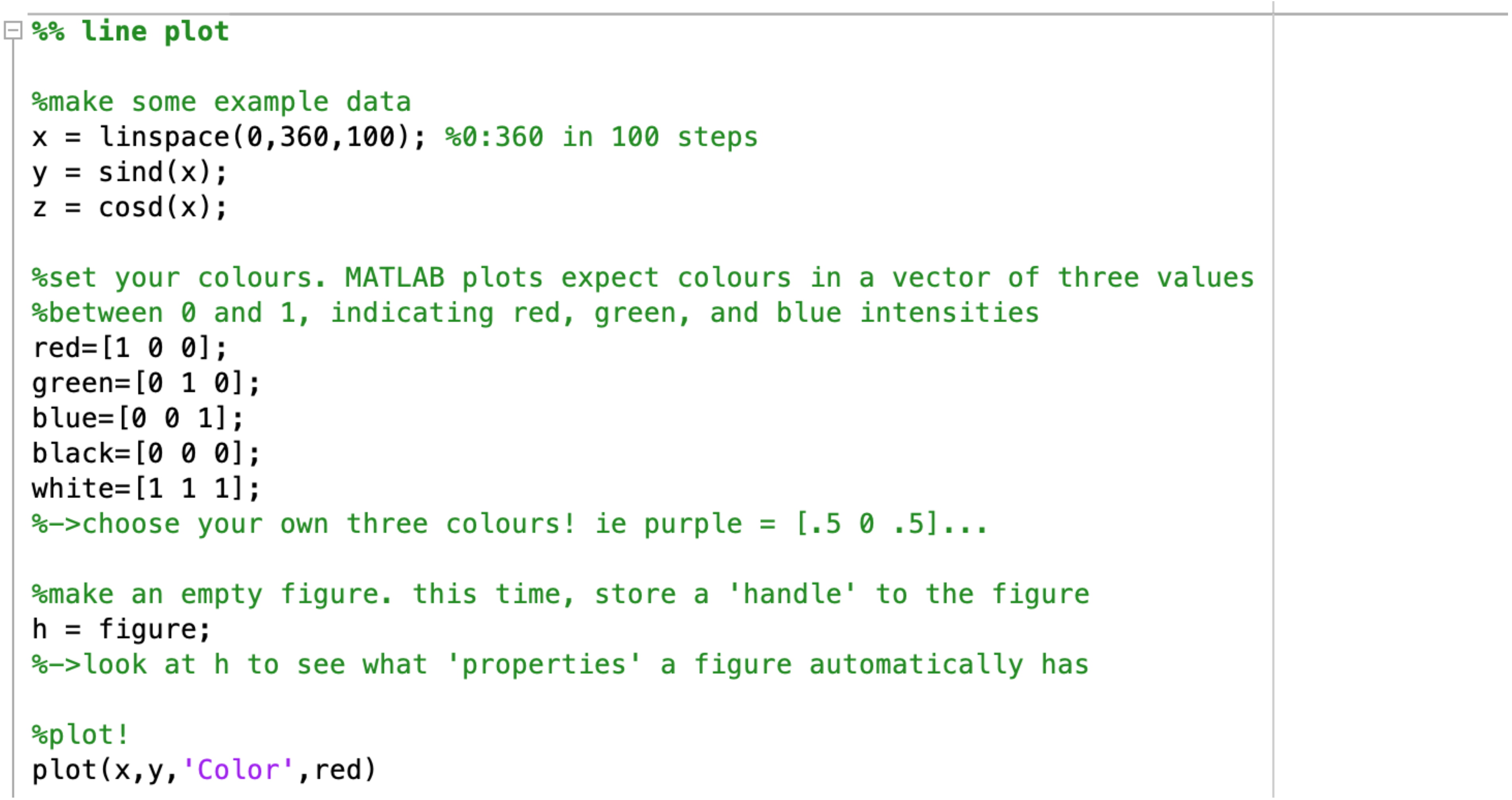
Line plot: tips

- store the figure information
  - look inside the handle. what are its properties?
- make variables for different colours
  - make black
  - define your own three colours…
- plot(x,y) again, specifying the colour
  - **Name-Value pairs**
  - go back to “help plot” if you need to!

```
h = figure; %open a figure, and store a ‘handle’ to it in a variable
```

```
red=[1 0 0];
green=[0 1 0];
blue=[0 0 1];
black=[0 0 0];
```

Line plot: example

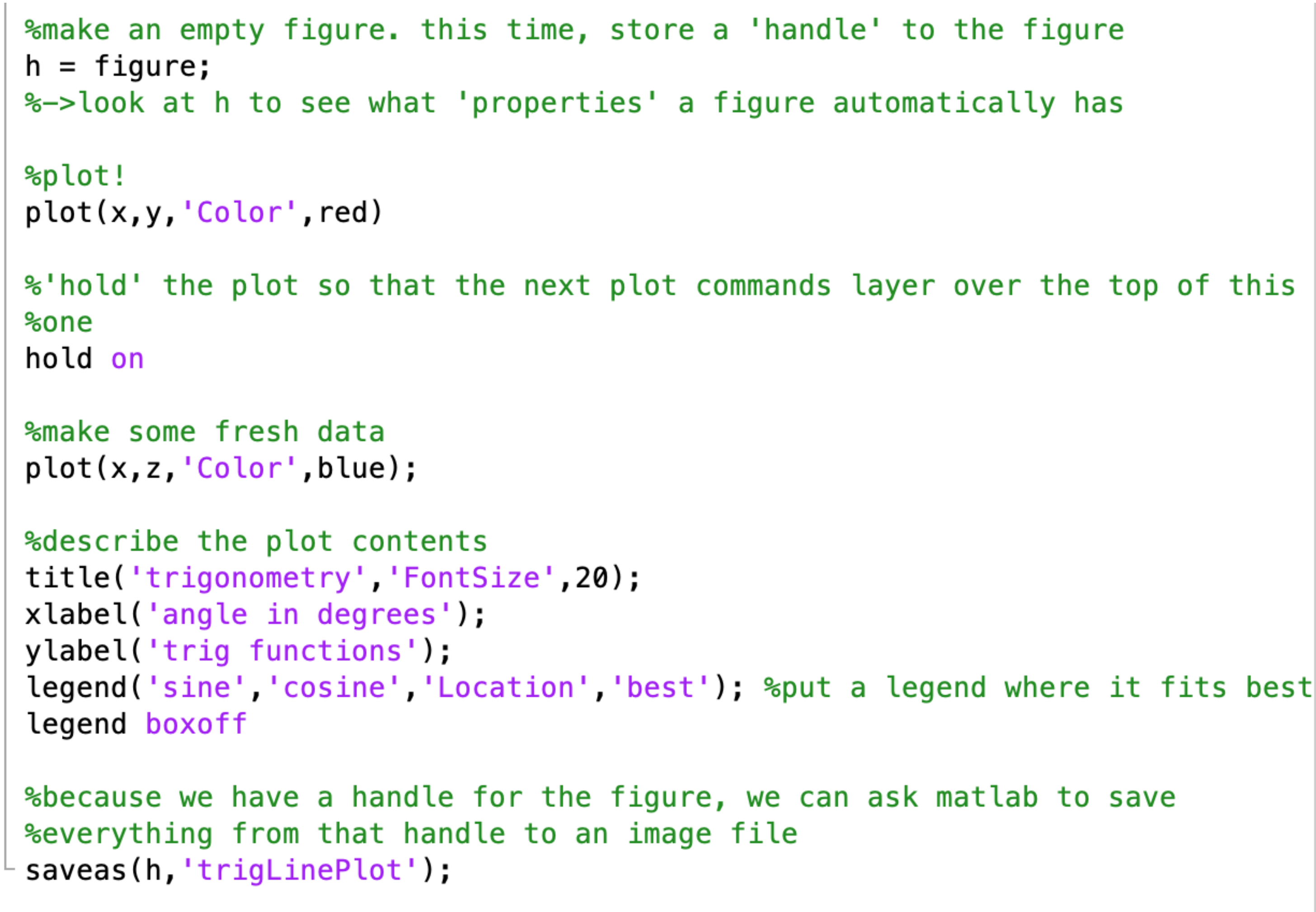


```
%% line plot

%make some example data
x = linspace(0,360,100); %0:360 in 100 steps
y = sind(x);
z = cosd(x);

%set your colours. MATLAB plots expect colours in a vector of three values
%between 0 and 1, indicating red, green, and blue intensities
red=[1 0 0];
green=[0 1 0];
blue=[0 0 1];
black=[0 0 0];
white=[1 1 1];
%->choose your own three colours! ie purple = [.5 0 .5]...

%make an empty figure. this time, store a 'handle' to the figure
h = figure;
%->look at h to see what 'properties' a figure automatically has

%plot!
plot(x,y,'Color',red)
```

Line plot: exercise

- add another plot
  - “hold on” under first plotting command
  - use x and z data variables to make another line
  - give this line a different colour
- add labels
  - try out “xlabel”, “ylabel”, and “title”
  - explore “box”, “axis”, and “legend” commands to make plot look ‘publishable’…
- save with “saveas” and the figure handle
  - see if you can work out how to save it as a jpeg!

Line plot: example



```
%make an empty figure. this time, store a 'handle' to the figure
h = figure;
%->look at h to see what 'properties' a figure automatically has

%plot!
plot(x,y,'Color',red)

%'hold' the plot so that the next plot commands layer over the top of this
%one
hold on

%make some fresh data
plot(x,z,'Color',blue);

%describe the plot contents
title('trigonometry','FontSize',20);
xlabel('angle in degrees');
ylabel('trig functions');
legend('sine','cosine','Location','best'); %put a legend where it fits best
legend boxoff

%because we have a handle for the figure, we can ask matlab to save
%everything from that handle to an image file
saveas(h,'trigLinePlot');
```

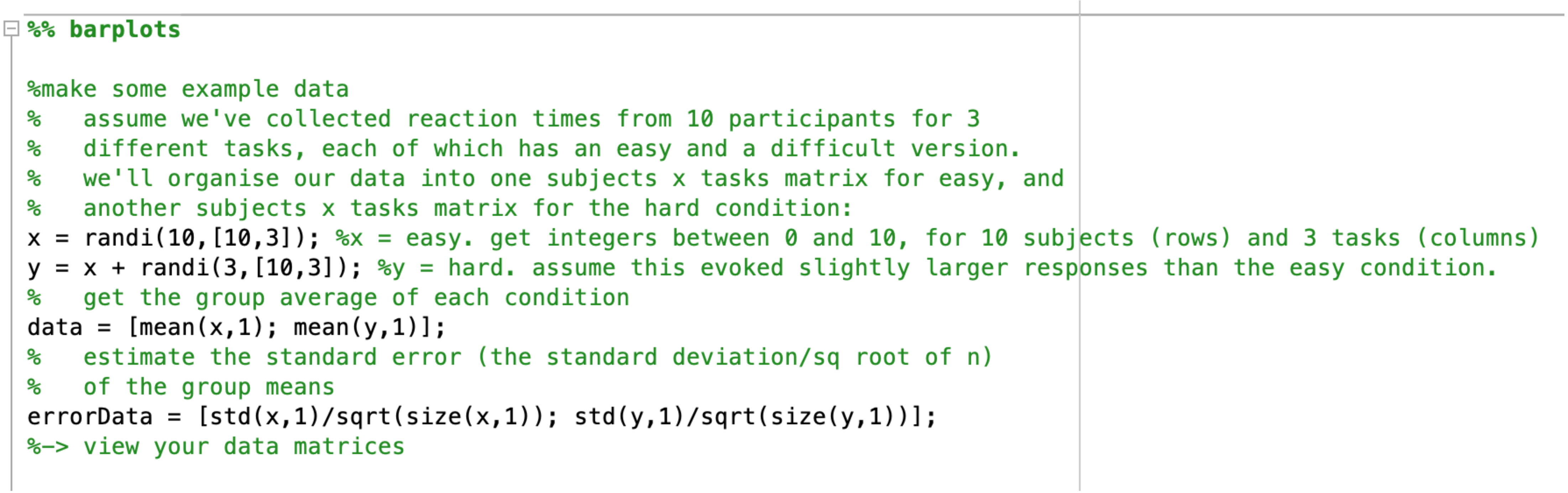
Bar plots

Bar plots: basics (exercise)

- make (or load) some data
- get the group means of x and y for each task
- store the group means in a variable
  - 2 conditions (rows), 3 tasks (columns)
- estimate the standard error of each mean (standard deviation/square root of n)
  - “std” and “sqrt”

```
x = randi(10,[10,3]); %x = easy. get random integers between 0 and 10, for 10 subjects (rows) and 3 tasks (columns)

y = x + randi(3,[10,3]); %y = hard. assume this evoked slightly larger responses than the easy condition.
```

Bar plots: basics (example)



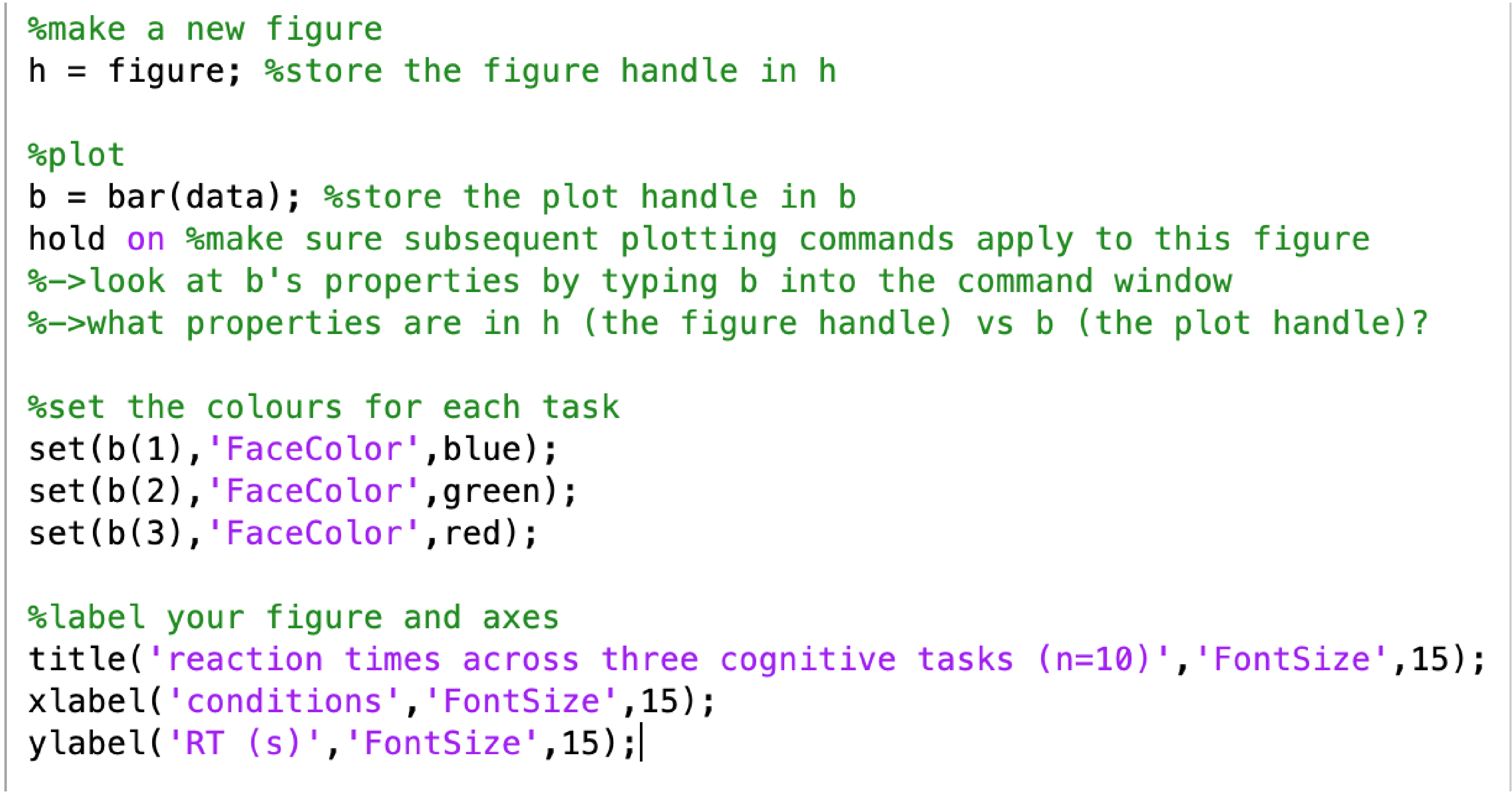
```
%% barplots

%make some example data
%   assume we've collected reaction times from 10 participants for 3
%   different tasks, each of which has an easy and a difficult version.
%   we'll organise our data into one subjects x tasks matrix for easy, and
%   another subjects x tasks matrix for the hard condition:
x = randi(10,[10,3]); %x = easy. get integers between 0 and 10, for 10 subjects (rows) and 3 tasks (columns)
y = x + randi(3,[10,3]); %y = hard. assume this evoked slightly larger responses than the easy condition.
%   get the group average of each condition
data = [mean(x,1); mean(y,1)];
%   estimate the standard error (the standard deviation/sq root of n)
%   of the group means
errorData = [std(x,1)/sqrt(size(x,1)); std(y,1)/sqrt(size(y,1))];
%-> view your data matrices
```

Bar plots: plot properties (exercise)

- h = figure;
- explore the “bar” function
- plot
- create a handle for the **plot**
    - explore the plot’s properties
- hold on
- modify the bar colours with the plot handle and “set”
- add a title and axis labels

b = bar(data);

Bar plots: plot properties (example)



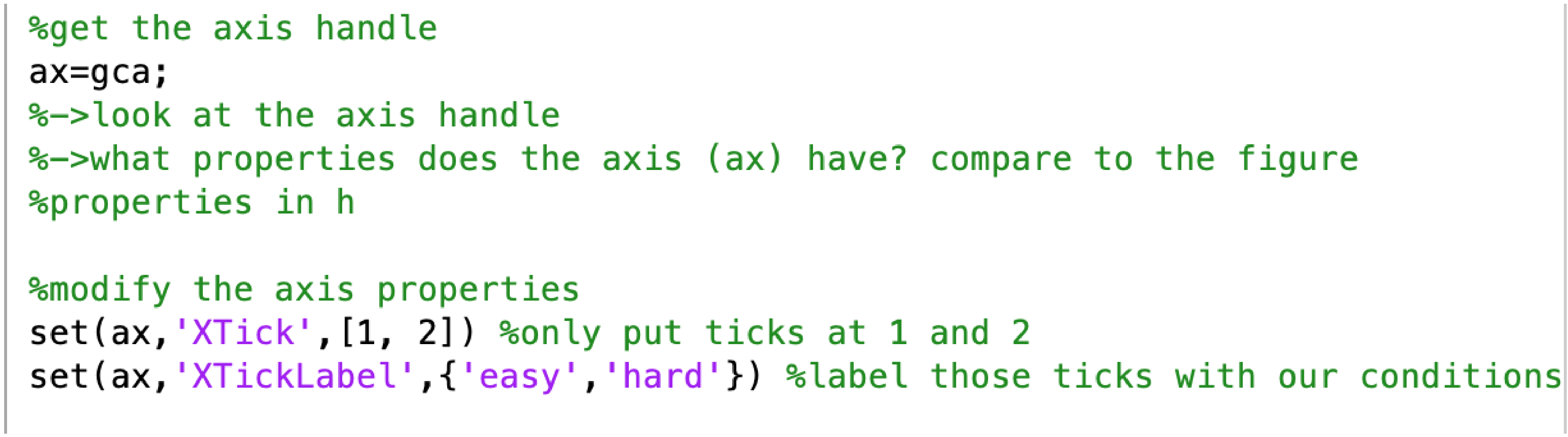
```
%make a new figure
h = figure; %store the figure handle in h

%plot
b = bar(data); %store the plot handle in b
hold on %make sure subsequent plotting commands apply to this figure
%->look at b's properties by typing b into the command window
%->what properties are in h (the figure handle) vs b (the plot handle)?

%set the colours for each task
set(b(1),'FaceColor',blue);
set(b(2),'FaceColor',green);
set(b(3),'FaceColor',red);

%label your figure and axes
title('reaction times across three cognitive tasks (n=10)','FontSize',15);
xlabel('conditions','FontSize',15);
ylabel('RT (s)','FontSize',15);
```

Bar plot: axis properties (exercise)

- get the axis handle
  - compare the properties in the figure, plot, and axis handles
- change the x-axis ticks and labels
  - hint: use ‘set’ and the axis handle

```
ax = gca; %get current axis
```

Bar plot: axis properties (example)



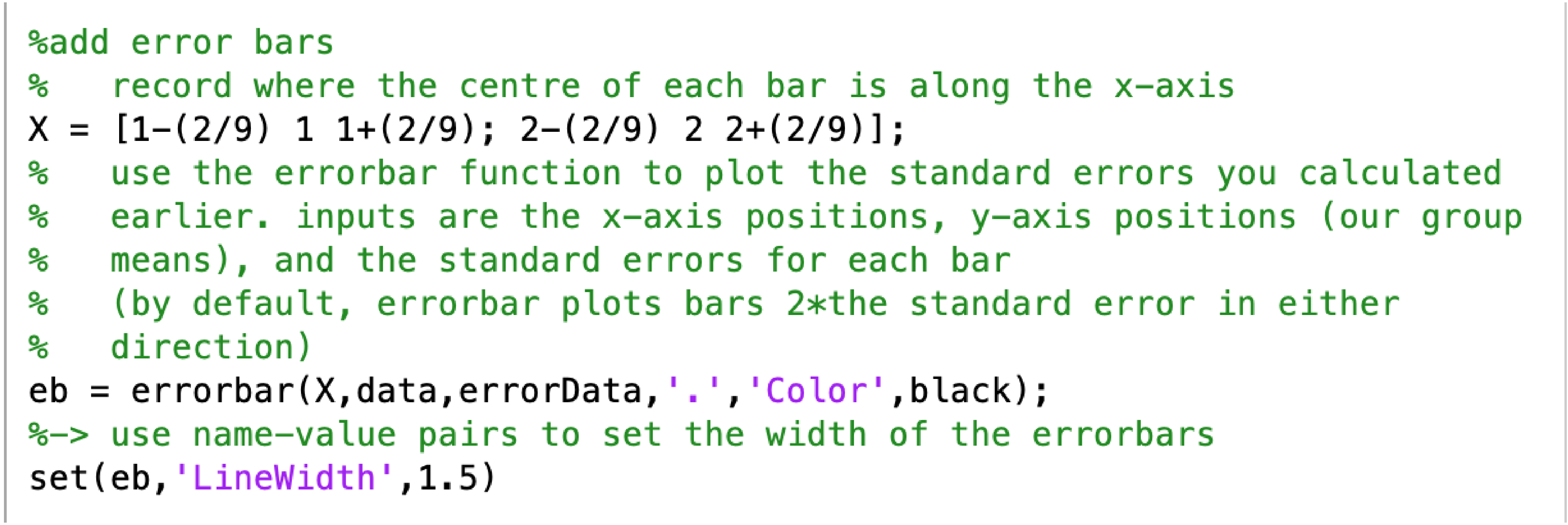
```
%get the axis handle
ax=gca;
%->look at the axis handle
%->what properties does the axis (ax) have? compare to the figure
%properties in h

%modify the axis properties
set(ax,'XTick',[1, 2]) %only put ticks at 1 and 2
set(ax,'XTickLabel',{'easy','hard'}) %label those ticks with our conditions
```

Bar plots: error bars (exercise)

- find the location of each bar on the x-axis and store it in a variable
- use “errorbar” to plot the standard error of the means
  - hint: use the errors you calculated earlier

```
X = [1-(2/9) 1 1+(2/9); 2-(2/9) 2 2+(2/9)];
```

Bar plots: error bars (example)

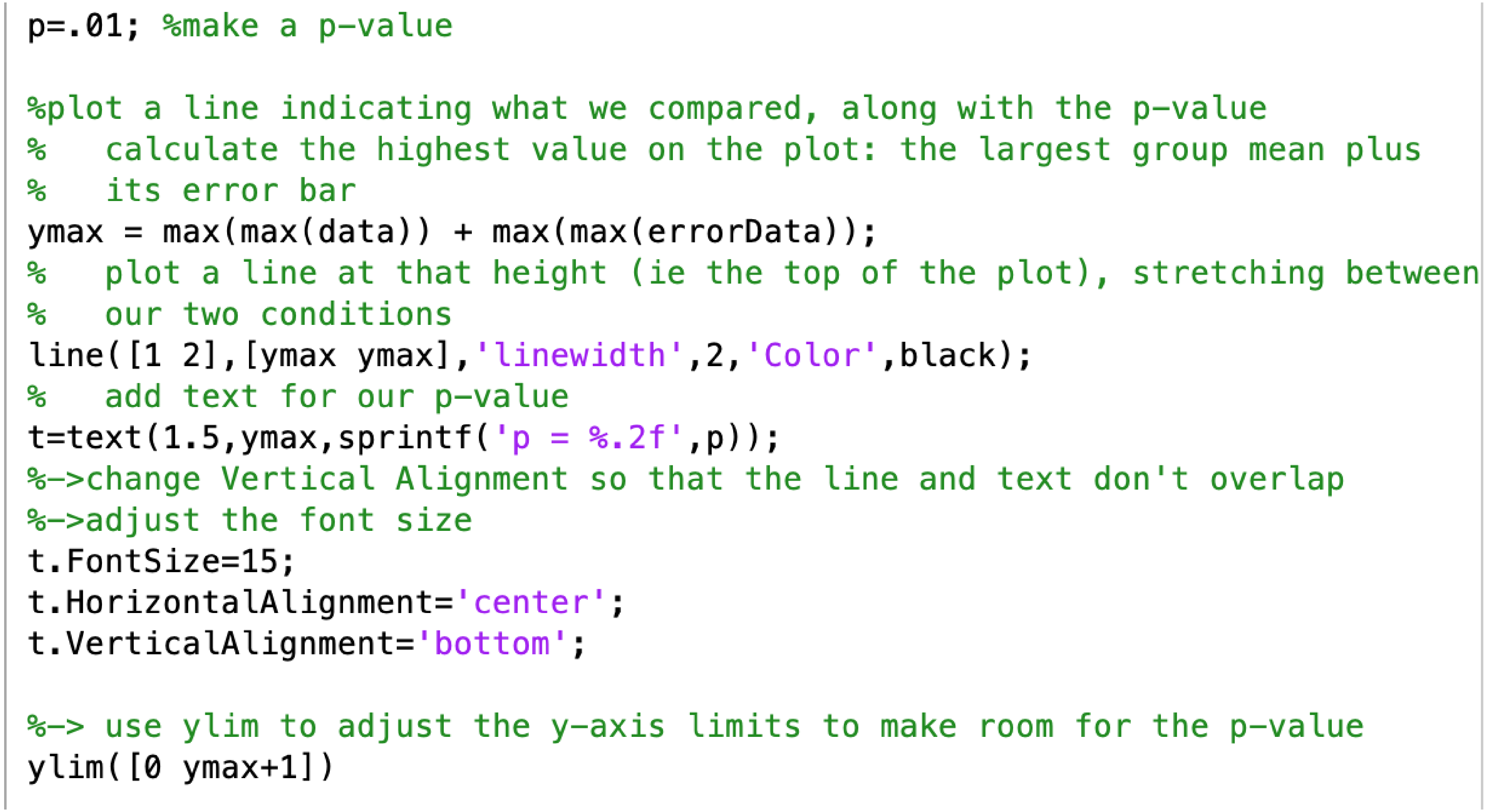


```
%add error bars
%   record where the centre of each bar is along the x-axis
X = [1-(2/9) 1 1+(2/9); 2-(2/9) 2 2+(2/9)];
%   use the errorbar function to plot the standard errors you calculated
%   earlier. inputs are the x-axis positions, y-axis positions (our group
%   means), and the standard errors for each bar
%   (by default, errorbar plots bars 2*the standard error in either
%   direction)
eb = errorbar(X,data,errorData,'.','Color',black);
%-> use name-value pairs to set the width of the errorbars
set(eb,'LineWidth',1.5)
```

Bar plots: reporting stats (exercise)

- find the highest value on the plot
    - without looking at the plot!
- “line” and “text”
    - plot a line from the easy to the hard condition along the top of the plot
    - put text in the centre showing a p-value
        - make a handle to the text
        - modify the text size and vertical alignment
- “ylim”
    - adjust the y-axis limits to give the p-value more space

Bar plots: reporting stats (example)



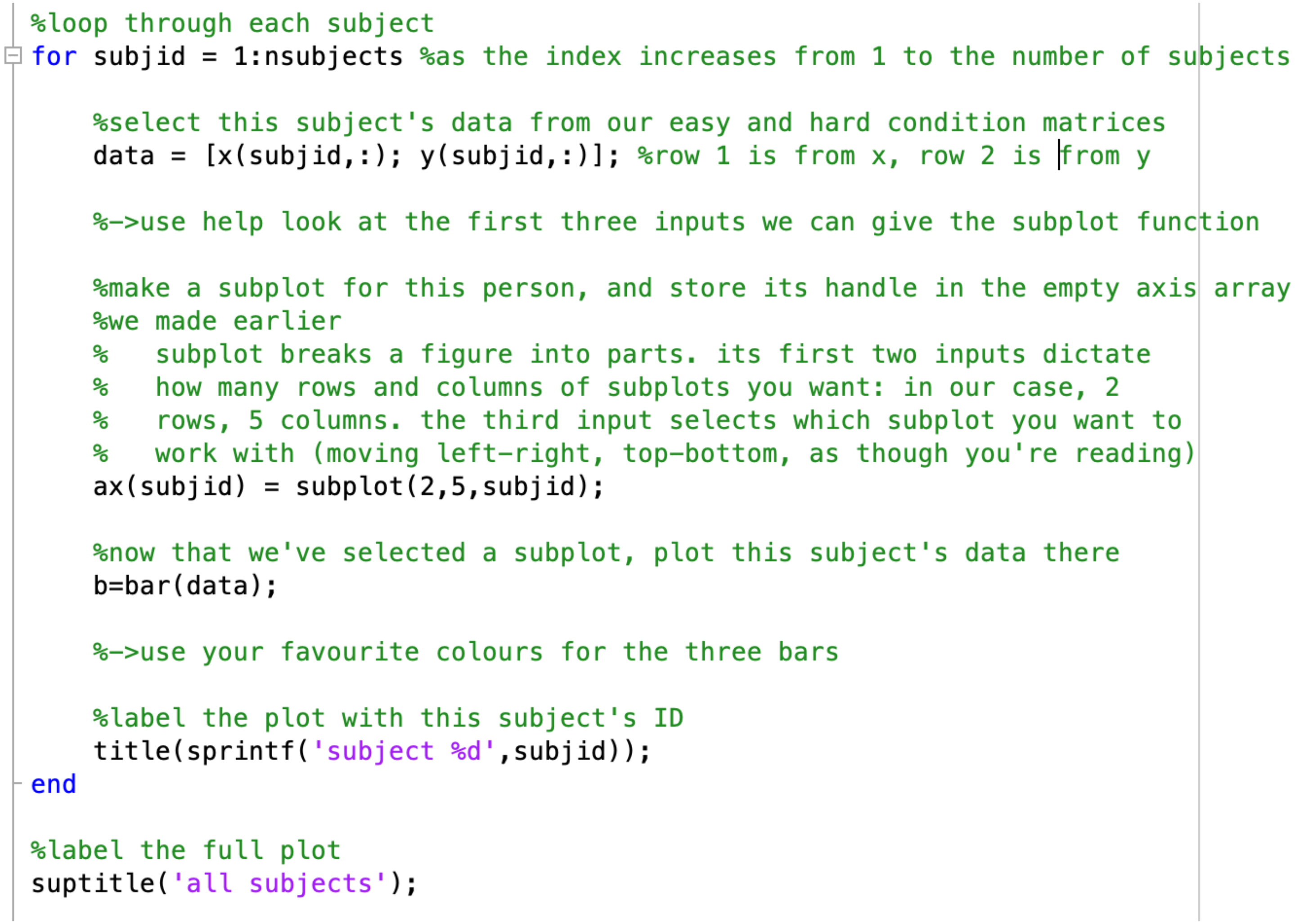
```
p=.01; %make a p-value

%plot a line indicating what we compared, along with the p-value
%   calculate the highest value on the plot: the largest group mean plus
%   its error bar
ymax = max(max(data)) + max(max(errorData));
%   plot a line at that height (ie the top of the plot), stretching between
%   our two conditions
line([1 2],[ymax ymax],'linewidth',2,'Color',black);
%   add text for our p-value
t=text(1.5,ymax,sprintf('p = %.2f',p));
%->change Vertical Alignment so that the line and text don't overlap
%->adjust the font size
t.FontSize=15;
t.HorizontalAlignment='center';
t.VerticalAlignment='bottom';

%-> use ylim to adjust the y-axis limits to make room for the p-value
ylim([0 ymax+1])
```

Subplots

Subplots (exercise)

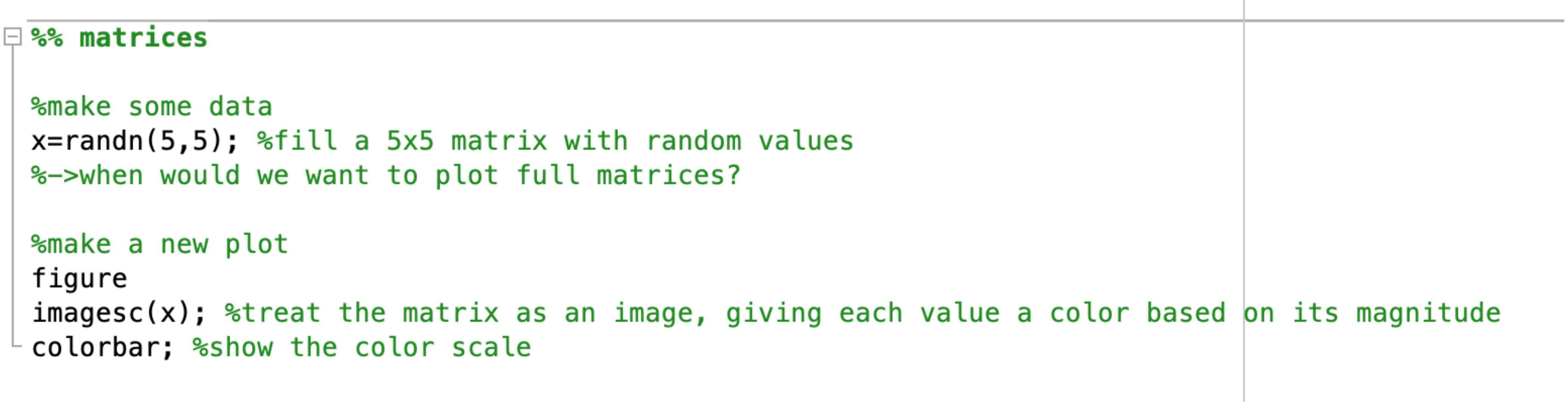
- explore the “subplot” function
- make a figure with subplots
  - 1 per person in our bar plot dataset
  - 2 rows, 5 columns
- loop through the subjects
- for each person, plot their 3 tasks and 2 conditions as you did for the group average bar plots
- give each subplot a title
- calculate the group range and use it to set the y-axis limits

Subplots (example)



```
%loop through each subject
for subjid = 1:nsubjects %as the index increases from 1 to the number of subjects

    %select this subject's data from our easy and hard condition matrices
    data = [x(subjid,:); y(subjid,:)]; %row 1 is from x, row 2 is from y

    %->use help look at the first three inputs we can give the subplot function

    %make a subplot for this person, and store its handle in the empty axis array
    %we made earlier
    %   subplot breaks a figure into parts. its first two inputs dictate
    %   how many rows and columns of subplots you want: in our case, 2
    %   rows, 5 columns. the third input selects which subplot you want to
    %   work with (moving left-right, top-bottom, as though you're reading)
    ax(subjid) = subplot(2,5,subjid);

    %now that we've selected a subplot, plot this subject's data there
    b=bar(data);

    %->use your favourite colours for the three bars

    %label the plot with this subject's ID
    title(sprintf('subject %d',subjid));
end

%label the full plot
suptitle('all subjects');
```

Just for fun



```
%% matrices

%make some data
x=randn(5,5); %fill a 5x5 matrix with random values
%->when would we want to plot full matrices?

%make a new plot
figure
imagesc(x); %treat the matrix as an image, giving each value a color based on its magnitude
colorbar; %show the color scale
```