

MATLAB Introduction Course: Lecture 2

Øivind K. Kjerstad

3. October 2014

Recap

Last time we looked at:

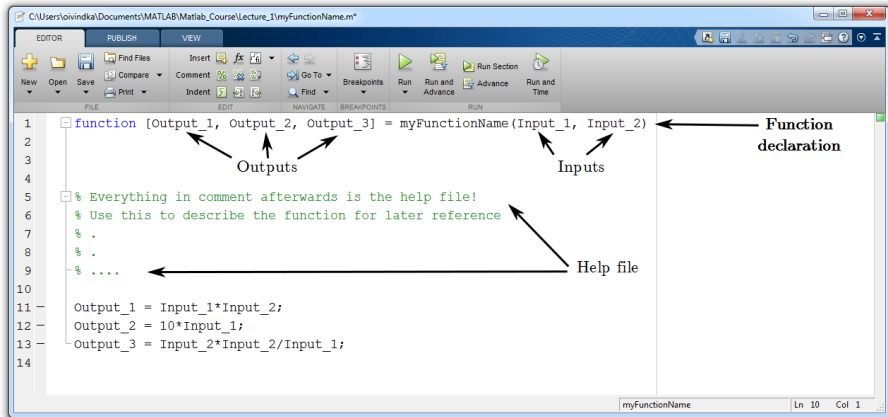
- Scripts
- Variables
- Variable manipulation
- Simple plotting
- Help

TOC

- 1 **Functions**
- 2 Basic programming
- 3 Line Plots
- 4 Surface Plots
- 5 Vectorization

User-defined functions

- Functions look exactly like scripts, but for one difference; Functions must have a function declaration



The screenshot shows the MATLAB Editor window with a function file named 'myFunctionName.m'. The code is as follows:

```
1 function [Output_1, Output_2, Output_3] = myFunctionName(Input_1, Input_2)
2
3
4
5 % Everything in comment afterwards is the help file!
6 % Use this to describe the function for later reference
7 % .
8 % .
9 % ....
10
11 Output_1 = Input_1*Input_2;
12 Output_2 = 10*Input_1;
13 Output_3 = Input_2*Input_2/Input_1;
14
```

Annotations in the image:

- Function declaration:** Points to the first line of code: `function [Output_1, Output_2, Output_3] = myFunctionName(Input_1, Input_2)`.
- Outputs:** Points to the output variables in the declaration: `Output_1, Output_2, Output_3`.
- Inputs:** Points to the input variables in the declaration: `Input_1, Input_2`.
- Help file:** Points to the comment block starting at line 5: `% Everything in comment afterwards is the help file!`.

This is a valid function!

User-defined functions

The function declaration

```
function [x, y, z] = functionName(a,b,c)
```

- Must have the reserved word: function
 - If more than one output, then must have brackets
 - Function name **should** match MATLAB file name
 - Inputs must be specified
-
- No need for return: MATLAB 'returns' the variables whose names match those in the function declaration
 - Variable scope: Any variables created within the function but not returned disappear after the function stops running

Why use functions?

- They allow us to reuse code instead of rewriting it
- Functions allow us to test small parts of our program in isolation from the rest
- They allow us to conceive of our program/script as a bunch of sub-steps (Each sub-step can be its own function)
- Functions allow us to keep our variable workspace clean

Functions overloading

- We are familiar with functions such as
 - ▶ `zeros`
 - ▶ `size`
 - ▶ `length`
 - ▶ `sum`
- Look at the help file for how to invoke the function
 - ▶ `help size`
- Several ways are possible
 - ▶ `d = size(X)`
 - ▶ `[m,n] = size(X)`
 - ▶ `m = size(X,dim)`
 - ▶ `[d1,d2,d3,...,dn] = size(X)`

Functions overloading

- MATLAB functions are generally overloaded
 - ▶ Can take a variable number of inputs
 - ▶ Can return a variable number of outputs
- What would the following commands return:
 - ▶ `>> a=zeros(2,4,8); %n-dimensional matrices are OK`
 - ▶ `>> D=size(a)`
 - ▶ `>> [m,n]=size(a)`
 - ▶ `>> [x,y,z]=size(a)`
 - ▶ `>> m2=size(a,2)`
- You can overload your own functions by having variable input and output arguments (see `varargin`, `nargin`, `varargout`, `nargout`)

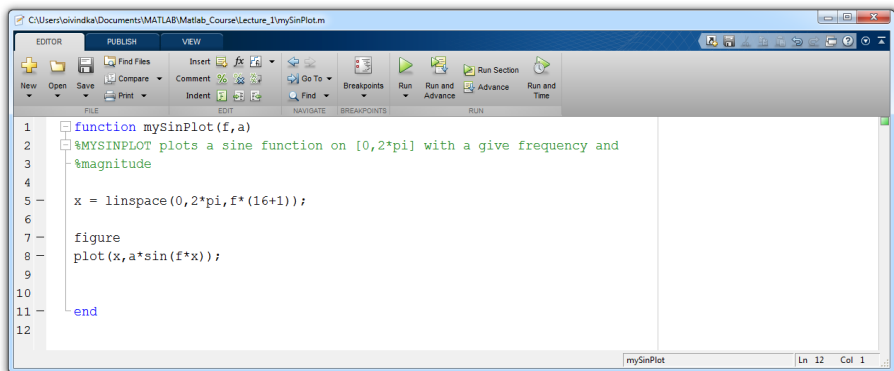
As you learn MATLAB, don't worry too much about writing functions that handle overloading

Exercise 1: A sine wave plot function

Create a function that:

- Takes two inputs; frequency and amplitude
- Has no output
- Creates a plot of a sine wave with the given frequency and amplitude on the range $[0, 2\pi]$
- For good sampling, use 16 or more points per period
- Use function naming as you like

Exercise 1: One possible solution



```
1 function mySinPlot(f,a)
2 %MYSINPLOT plots a sine function on [0,2*pi] with a give frequency and
3 %magnitude
4
5 x = linspace(0,2*pi,f*(16+1));
6
7 figure
8 plot(x,a*sin(f*x));
9
10
11 end
12
```

Solutions are not unique! As long as it works, do it your way

TOC

- 1 Functions
- 2 Basic programming**
- 3 Line Plots
- 4 Surface Plots
- 5 Vectorization

Relational Operators

- MATLAB relational operators

- ▶ (`==`) equal
- ▶ (`~=`) not equal
- ▶ (`>`) greater than
- ▶ (`<`) less than
- ▶ (`>=`) greater or equal
- ▶ (`<=`) less or equal

- Logical operators

- ▶ (`&&`) And
- ▶ (`||`) Or
- ▶ (`~`) Not
- ▶ (`xor`) Xor
- ▶ (`all`) all
- ▶ (`any`) any

- Boolean values: zero is false, nonzero is true

if/else/elseif

- Basic flow-control, common to most languages

IF

```
if cond
  commands
end
```

ELSE

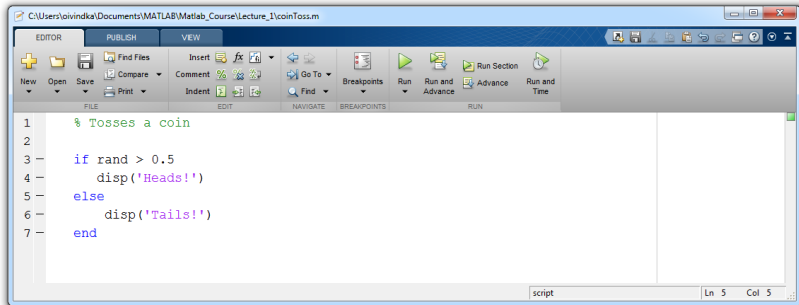
```
if cond
  commands
else
  commands
end
```

ELSEIF

```
if cond1
  commands
elseif cond2
  commands
else
  commands
end
```

- No need for parentheses around `cond`, command blocks are between reserved words

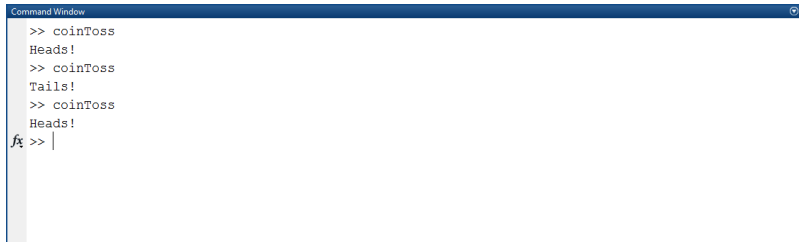
Example: Coin toss



The image shows the MATLAB Editor window for a file named 'coinToss.m'. The script contains the following code:

```
1 % Tosses a coin
2
3 if rand > 0.5
4     disp('Heads!')
5 else
6     disp('Tails!')
7 end
```

The status bar at the bottom right of the editor indicates 'script' and 'Ln 5 Col 5'.



The image shows the MATLAB Command Window with the following output:

```
>> coinToss
Heads!
>> coinToss
Tails!
>> coinToss
Heads!
fx >> |
```

for loops

- Use for a known number of iterations
- MATLAB syntax:

FOR LOOPS

```
for n=1:100  
    commands  
end
```

- The loop variable `n` (can have any naming, but be aware of built-in variables)
 - ▶ Is defined as a vector
 - ▶ Is a scalar within the command block
 - ▶ Does not have to have consecutive values
- The command block
 - ▶ Anything between the `for` line and the `end`

while loop

- A more general loop, does not need to know the number of iterations
- MATLAB syntax:

WHILE LOOPS

```
while cond
    commands
end
```

- The command block will execute while the conditional expression is true

Infinite loops will occur if cond is true always

Be aware, it may lock up your program! (Use `Ctrl + c` to end script/function)

Exercise 2: Fibonacci numbers

Create a function that:

- Takes one input, the length of the Fibonacci sequence
- Outputs a row vector of n Fibonacci numbers

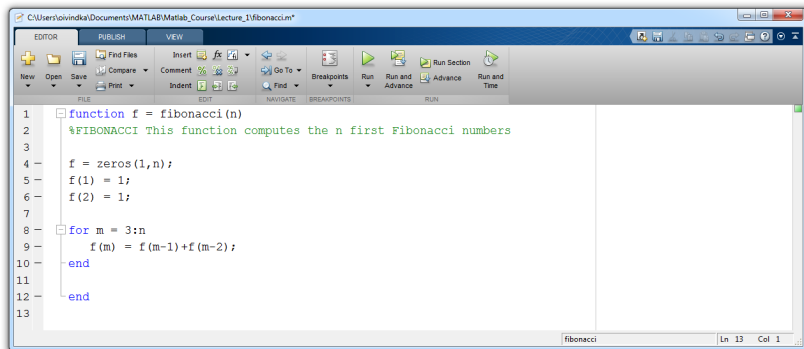
In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

with seed values $F_1 = 1, F_2 = 1$.

[1 1 2 3 5 8 13 ...]

Exercise 2: One possible solution



The image shows the MATLAB Editor window with the following code:

```
1 function f = fibonacci(n)
2 %FIBONACCI This function computes the n first Fibonacci numbers
3
4 f = zeros(1,n);
5 f(1) = 1;
6 f(2) = 1;
7
8 for m = 3:n
9     f(m) = f(m-1)+f(m-2);
10 end
11
12 end
13
```

The status bar at the bottom right of the editor window indicates "Ln 13 Col 1".



The image shows the MATLAB Command Window with the following output:

```
>> fibonacci(6)

ans =

     1     1     2     3     5     8

fx >>
```

TOC

- 1 Functions
- 2 Basic programming
- 3 Line Plots**
- 4 Surface Plots
- 5 Vectorization

Plot Options

- You may change the line color, marker style, and line style by adding a string argument
 - `>> plot(x,y,'k.-');`
 - This is a black line (k) with point markers (.) and a solid line (-)

Specifier	Color
y	yellow
m	magenta
c	cyan
r	red
g	green
b	blue
w	white
k	black

Specifier	Marker
o	Circle
+	Plus sign
*	Asterisk
.	Point
x	Cross
s	Square
d	Diamond
^	Upw. triangle
v	Downw. triangle
>	Right triangle
<	Left triangle
p	Pentagram
h	Hexagram

Specifier	Line style
-	Solid line
--	Dashed line
:	Dotted line
-.	Dash-dot line

Line and Marker Options

- Everything on a line in a plot can be customized
 - ▶ Color
 - ▶ LineStyle
 - ▶ LineWidth
 - ▶ Marker
 - ▶ MarkerEdgeColor
 - ▶ MarkerFaceColor
 - ▶ MarkerSize
- See `doc line_props` for a full list of properties that can be specified

Example

```
>> plot(x,y,'-mo','LineWidth',2,'MarkerEdgeColor','k',...  
    'MarkerFaceColor',[.49 1 .63],'MarkerSize',10);
```

Working with axis and labels

- Set range of plot axis using either
 - ▶ `Axis`
 - ▶ `xlim`, `ylim`, `zlim`
- Set axis labels using
 - ▶ `xlabel('text')`
 - ▶ `ylabel('text')`
 - ▶ `zlabel('text')`
- Set legend using
 - ▶ `legend('text1','text2','..')`
- Set plot title using `title('text')`

Example

```
plot(time,force_1);
hold on;
plot(time,force_2);
xlim([0 100]);
ylim([-10 10]);
xlabel('Time [s]');
ylabel('Force [N]');
title('Force plot');
legend('Force1','Force2');
```

The above labels, legend, and title can be customized similarly to `lineSpec`

Working with the figure

- Set figure size and position on screen by either
 - ▶ `figure('Position',[xpos ypos width height]);`
 - ▶ `f = figure(n);`
`set(f,'Position',[xpos ypos width height]);`
 - ★ This is beneficial to plot in figure `n`
- Whatever is plotted after a figure deceleration will appear in that figure
 - ▶ Until a new figure is declared
- The output of `figure` is called the figure handle
- Use the figure handle to add more data to your figures

Example

```
f1 = figure(1);  
plot(data_1);  
hold on;  
plot(data_2);  
  
f2 = figure(2);  
plot(data_3);  
hold on;  
plot(data_4);
```

Multiple Plots in one Figure

- To have multiple axes in one figure
 - ▶ `>> subplot(2,3,1)`
 - ★ makes a figure with 2 rows and three columns of axes, and activates the first axis for plotting
 - ★ each axis can have labels, a legend, and a title
- `>> subplot(2,3,4:6)`
 - ▶ activating a range of axes fuses them into one
- To close existing figures
 - ▶ `>> close([1 3])`
 - ★ closes figures 1 and 3
 - ▶ `>> close all`
 - ★ closes all figures (useful in scripts/functions)

Example

```
f1 = figure(1);
subplot(2,1,1);
plot(data_1);
hold on;
plot(data_2);

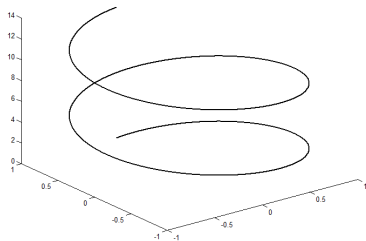
subplot(2,1,2);
plot(data_3);
hold on;
plot(data_4);
```


3D Line Plots

- We can plot in 3 dimensions just as easily as in 2, using `plot3`

Example

```
time=0:0.001:4*pi;  
x=sin(time);  
y=cos(time);  
z=time;  
plot3(x,y,z,'k','LineWidth',2);
```



- The same customization properties apply to `plot3` as for `plot`

Saving Figures

- Figures can be saved using the toolbar on the figure or using
 - ▶ `savefig`
 - ▶ `saveas`
- The most common formats are
 - ▶ **.fig** preserves all information
 - ▶ **.pdf** compressed image
 - ▶ **.bmp** uncompressed bitmap image
 - ▶ **.eps** high-quality scalable format
- Only use **.jpg** for photos!

If you would like to have scalable vector graphics in pdf with latex fonts, check out [plotpdf_{tex}](#) in Matlab central file exchange

TOC

- 1 Functions
- 2 Basic programming
- 3 Line Plots
- 4 Surface Plots**
- 5 Vectorization

3-D Surface Plots

- It is often more useful to visualize 3-D data as a surface
- MATLAB has many functions for this

Examples

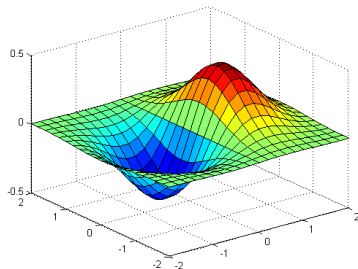
`surf`

`mesh`

`ribbon`

`contour3`

`++`



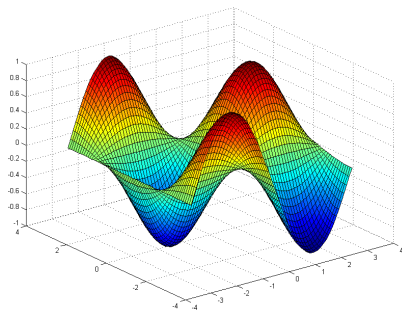
- Input arguments are either a matrix or 3 vectors (x,y,z)

surf

- `surf` puts vertices at specified points in space x,y,z , and connects all the vertices to make a surface

Example

```
x=-pi:0.1:pi;  
y=-pi:0.1:pi;  
[X,Y]=meshgrid(x,y);  
Z =sin(X).*cos(Y);  
surf(Z)
```



- There are three types of surface shading
 - ▶ `faceted`, `flat`, `interp`
- You can change colormaps
 - ▶ `colormap(gray)`

Colormaps

- You can change the colormap of your plots
 - ▶ MATLAB default is `jet`
 - ▶ `colormap(gray)`
 - ▶ `colormap(cool)`
 - ▶ `colormap(hot)`
- It is possible to define custom colormaps

Example

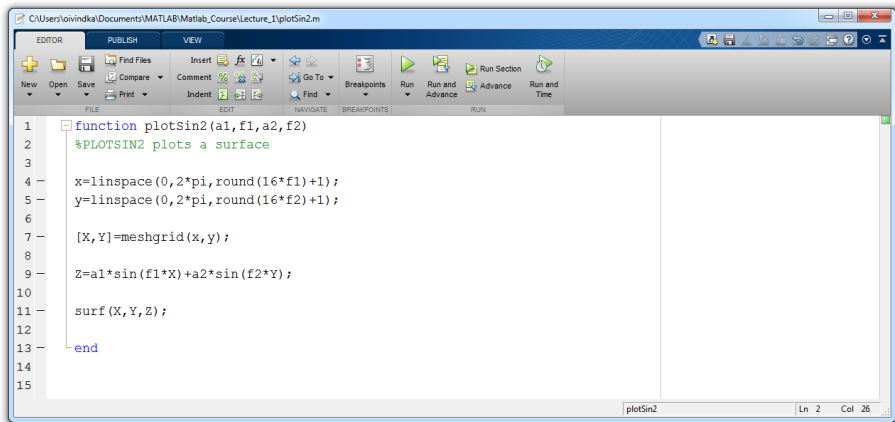
```
map = zeros(256,3);  
map(:,2)=(0:255)/255;  
colormap(map);
```

Exercise 3: Surface plot

Create a function that:

- Takes four inputs; two frequencies and two amplitudes
- Has no output
- Creates a surface plot of the function $z = a_1 \sin(f_1 x) + a_2 \sin(f_2 y)$ on the range $x \in [0, 2\pi]$, $y \in [0, 2\pi]$
- Use function naming as you like

Exercise 3: One possible solution



The image shows a screenshot of the MATLAB Editor window. The title bar indicates the file path: C:\Users\oivindka\Documents\MATLAB\Matlab_Course\Lecture_1\plotSin2.m. The window is divided into several panes: EDITOR, PUBLISH, and VIEW. The EDITOR pane contains the following MATLAB code:

```
1 function plotSin2(a1,f1,a2,f2)
2 %PLOTSIN2 plots a surface
3
4 x=linspace(0,2*pi,round(16*f1)+1);
5 y=linspace(0,2*pi,round(16*f2)+1);
6
7 [X,Y]=meshgrid(x,y);
8
9 Z=a1*sin(f1*X)+a2*sin(f2*Y);
10
11 surf(X,Y,Z);
12
13 end
14
15
```

The status bar at the bottom right of the editor shows "plotSin2" and "Ln 2 Col 26".

Specialized Plotting Functions

- MATLAB has a lot of specialized plotting functions
- `polar` - to make polar plots
 - ▶ `>> polar(0:0.01:2*pi,cos((0:0.01:2*pi)*2))`
- `bar` - to make bar graphs
 - ▶ `>> bar(1:10,rand(1,10))`
- `quiver` - to add velocity vectors to a plot
 - ▶ `>> [X,Y]=meshgrid(1:10,1:10)`
 - ▶ `>> quiver(X,Y,rand(10),rand(10))`
- `stairs` - plot piecewise constant functions
 - ▶ `>> stairs(1:10,rand(1,10))`
- `fill` - draws and fills a polygon with specified vertices
 - ▶ `>> fill([0 1 0.5],[0 0 1], 'r')`
- see help on these functions for syntax
- `doc specgraph` - for a complete list

TOC

- 1 Functions
- 2 Basic programming
- 3 Line Plots
- 4 Surface Plots
- 5 Vectorization**

Vectorization

Vectorization

In MATLAB vectorization is the art of avoiding loops. This makes the programs execute more efficient!

- `find` is a very important function
 - ▶ Returns indices of nonzero values
 - ▶ Can simplify code and help avoid loops
- Built-in functions will make it faster to write and execute

In small programs/scripts don't worry too much about vectorization

Example: Avoiding Loops

- Given `x = sin(linspace(0,10*pi,100))`, how many of the entries are positive?

Inefficient code

```
count = 0;
for n=1:length(x)
    if x(n)>0
        count=count+1;
    end
end
```

Efficient code

```
count=length(find(x>0));
```

Vectorization will come natural as you get better in writing code!

>> THE END