6.094 Introduction to MATLAB®
January (IAP) 2009

6.094

Introduction to Programming in MATLAB®

Lecture 2: Visualization and Programming

Sourav Dey
Danilo Šćepanović
Ankit Patel
Patrick Ho

IAP 2009

Outline

(1) Plotting Continued

(2) Scripts

(3) Functions

(4) Flow Control

Cartesian Plots

- We have already seen the plot function

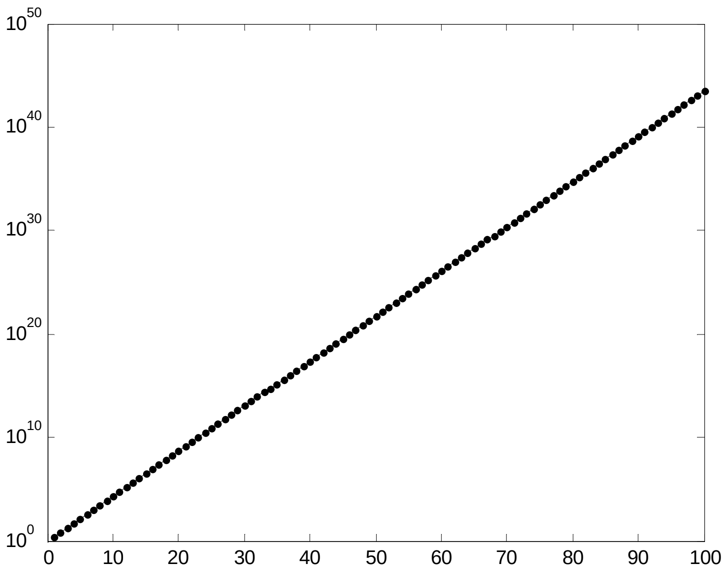
```
» x=-pi:pi/100:pi;
» y=cos(4*x).*sin(10*x).*exp(-abs(x));
» plot(x,y,'k-');
```

- The same syntax applies for semilog and loglog plots

```
» semilogx(x,y,'k');
» semilogy(y,'r.-');
» loglog(x,y);
```

- For example:

```
» x=0:100;
» semilogy(x,exp(x),'k.-');
```



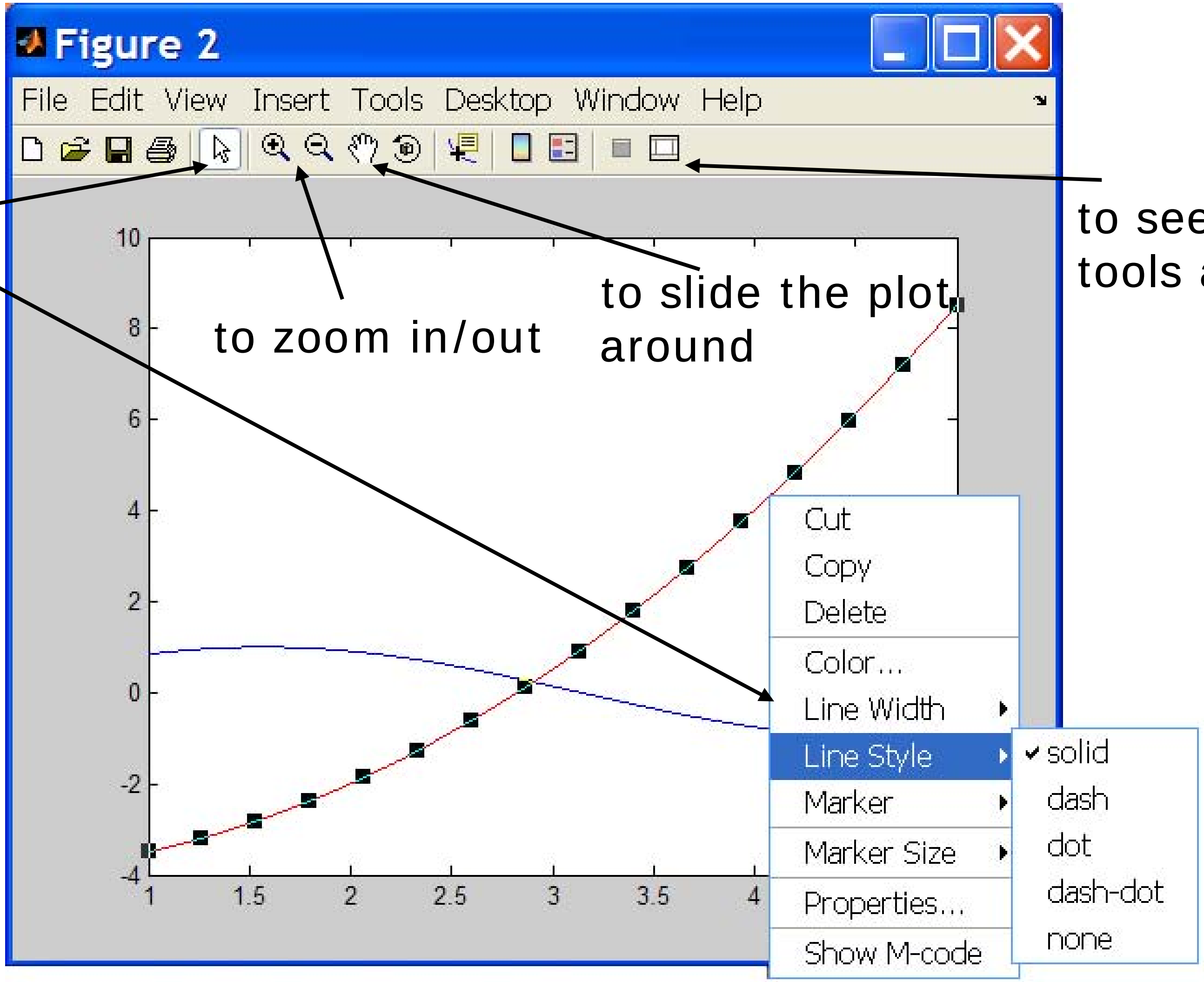
Playing with the Plot

to select lines and delete or change properties

to zoom in/out

to slide the plot around

to see all plot tools at once



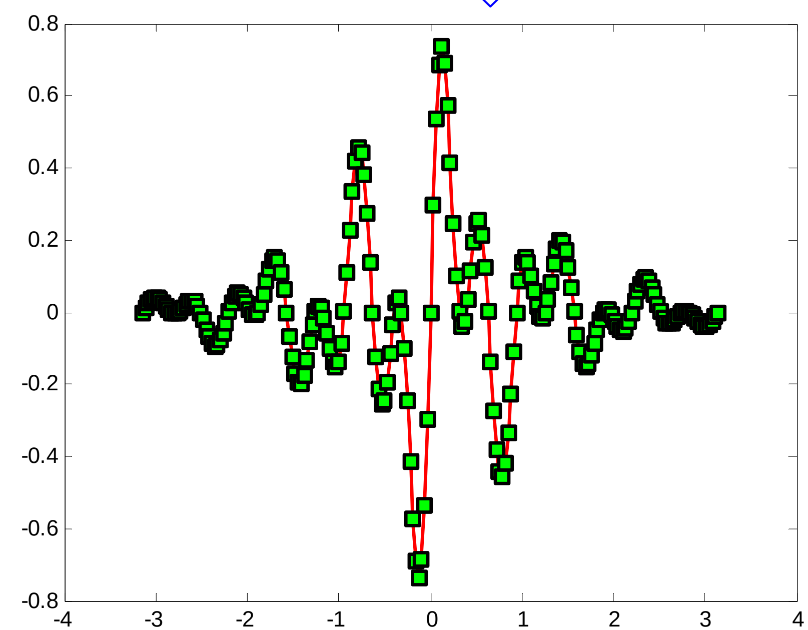
Courtesy of The MathWorks, Inc. Used with permission.

Line and Marker Options

- Everything on a line can be customized

```
» plot(x,y,'--rs','LineWidth',2,...
               'MarkerEdgeColor','k',...
               'MarkerFaceColor','g',...
               'MarkerSize',10)
```

- See **doc line** for a full list of properties that can be specified



Labels

- Last time we saw how to add titles and labels using the GUI. Can also do it command-line:
  » `title('Stress-Strain');`
  » `xlabel('Force (N)');`

- For multiple lines, add a legend entry for each line
  » `legend('Steel','Aluminum','Tungsten');`

- Can specify font and size for the text
  » `ylabel('Distance (m)','FontSize',14,...`
  `'FontName','Helvetica');`
    - use ... to break long commands across multiple lines

- To put parameter values into labels, need to use **num2str** and concatenate:
  » `str = [‘Strength of ' num2str(d) 'cm diameter rod'];`
  » `title(str)`

Axis

- A grid makes it easier to read values
  » **grid on**

- **xlim** sets only the x axis limits
  » **xlim([-pi pi]);**
- **ylim** sets only the y axis limits
  » **ylim([-1 1]);**

- To specify both at once, use **axis**:
  » **axis([-pi pi -1 1]);**
    - sets the x axis limits between -pi and pi and the y axis limits between -1 and 1

- Can specify tickmarks
  » **set(gca,'XTick', linspace(-pi,pi,3))**
    - see **doc axes** for a list of properties you can set this way
    - more on advanced figure customization in lecture 4

Axis Modes

- Built-in axis modes

  » **`axis square`**
  - makes the current axis look like a box

  » **`axis tight`**
  - fits axes to data

  » **`axis equal`**
  - makes x and y scales the same

  » **`axis xy`**
  - puts the origin in the bottom left corner (default)

  » **`axis ij`**
  - puts the origin in the top left corner (for viewing matrices)

Multiple Plots in one Figure

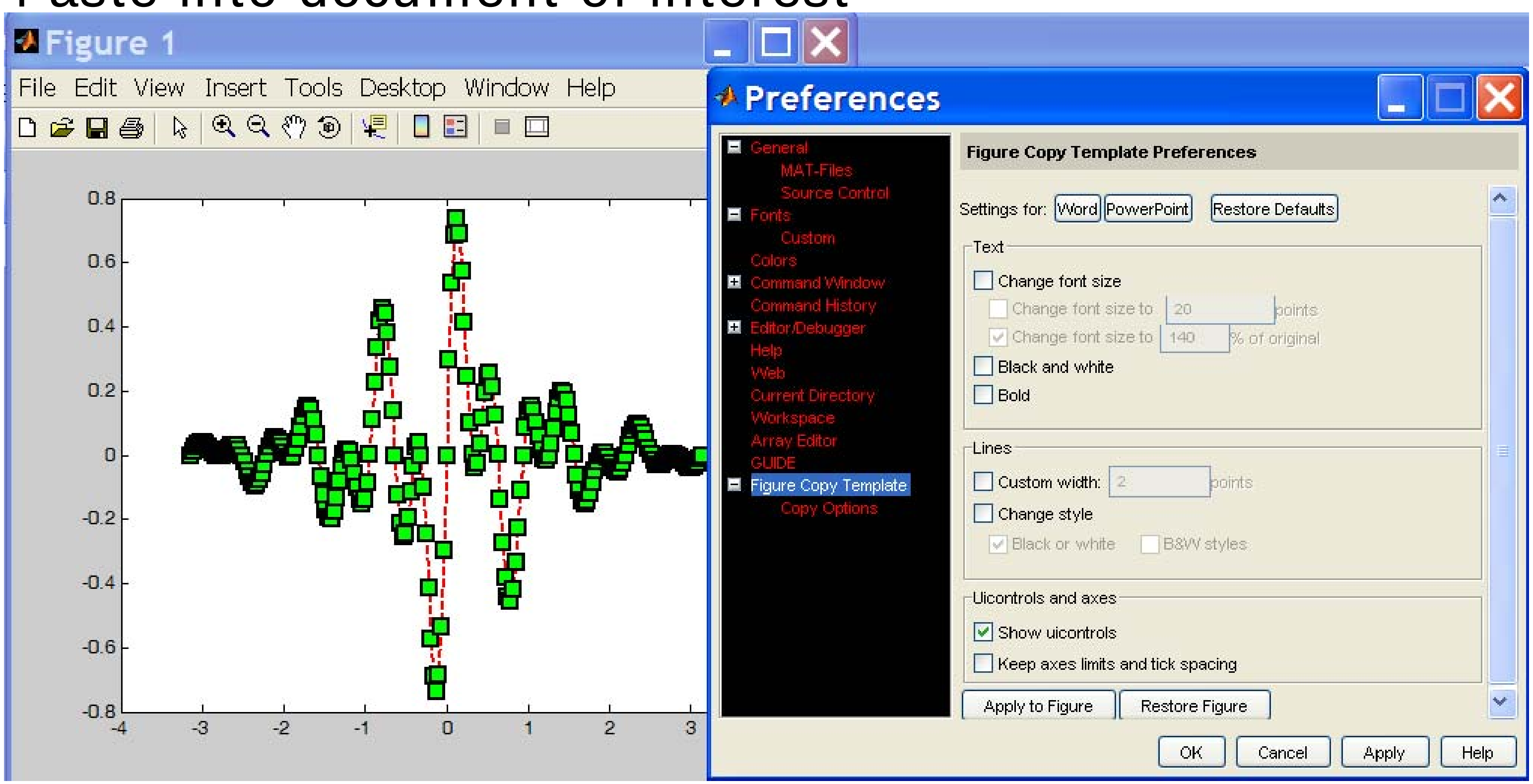
- Use the figure command to open a new figure
  - » **`figure`**
- or activate an open figure
  - » **`figure(1)`**

- To have multiple axes in one figure
  - » **`subplot(2,3,1) or subplot(231)`**
    - ➢ 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)

Copy / Paste Figures

- Figures can be pasted into other apps (word, ppt, etc)
- *Edit→ copy options→ figure copy template*
  - ➢ Change font sizes, line properties; presets for word and ppt
- *Edit→ copy figure* to copy figure
- Paste into document of interest



Courtesy of The MathWorks, Inc. Used with permission.

Saving Figures

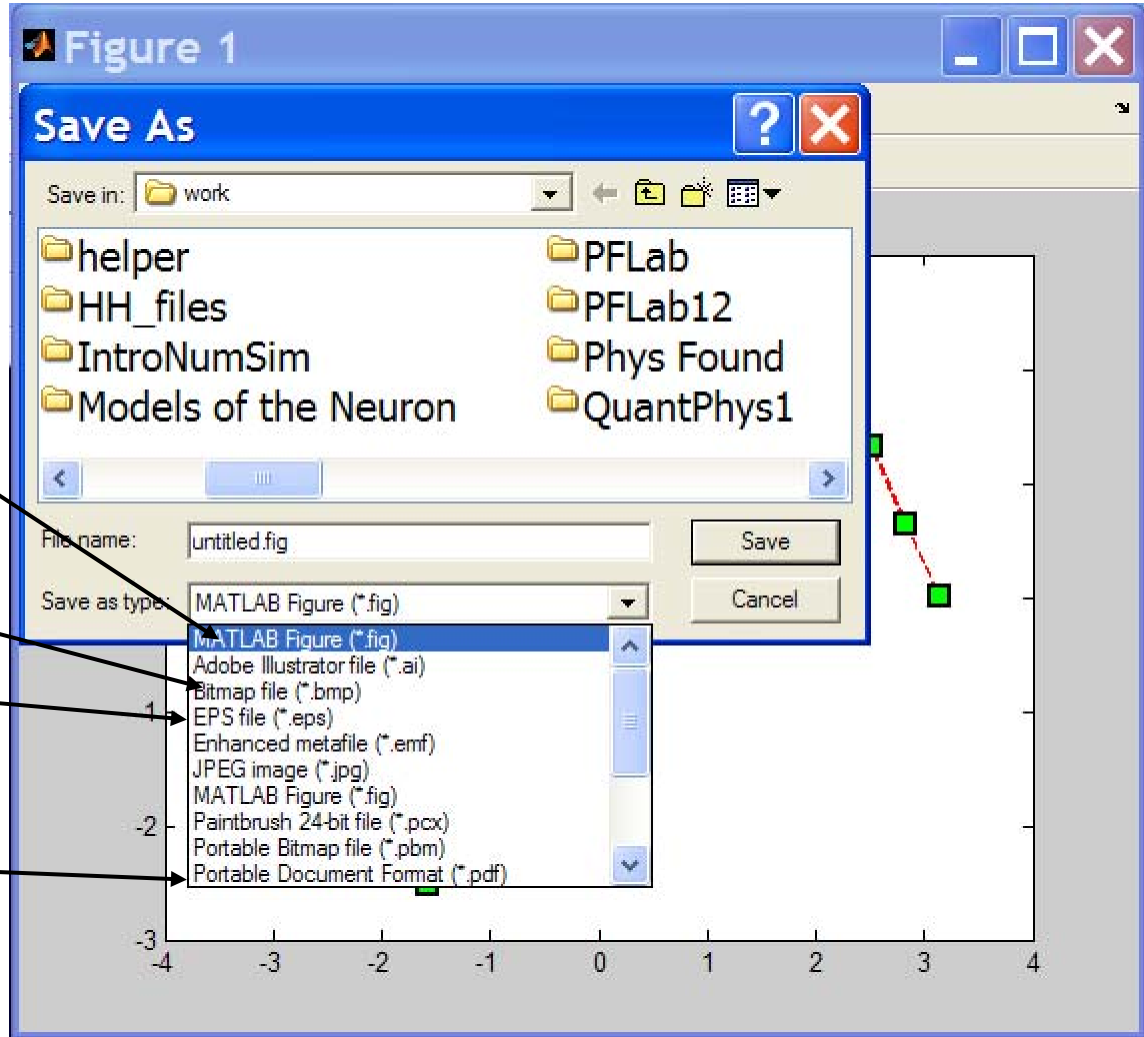
- Figures can be saved in many formats. The common ones are:

.fig preserves all information

.bmp uncompressed image

.eps high-quality scaleable format

.pdf compressed image

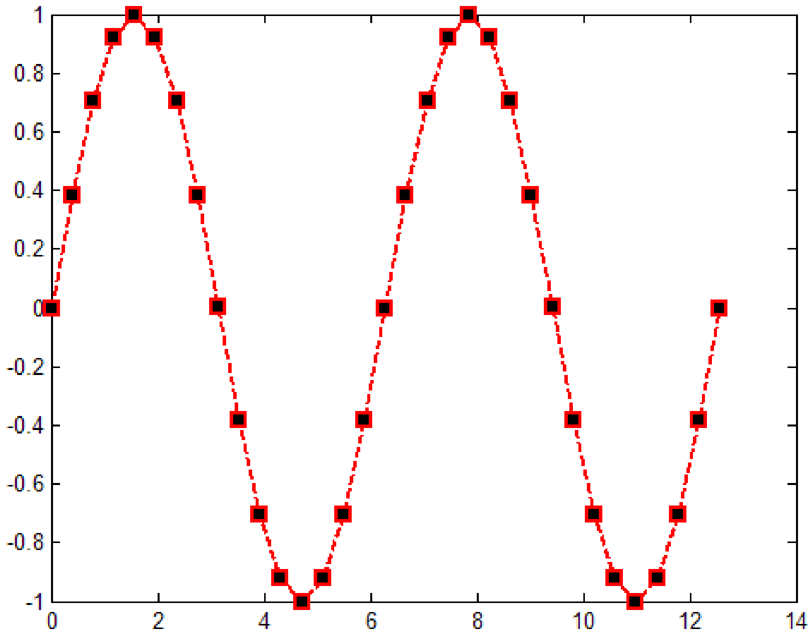


Figures: Exercise

- Open a figure and plot a sine wave over two periods with data points at 0, pi/8, 2pi/8... . Use black squares as markers and a dashed red line of thickness 2 as the line
  - » **figure**
  - » **plot(0:pi/4:4*pi,sin(0:pi/4:4*pi),'rs--',... 'LineWidth',2,'MarkerFaceColor','k');**

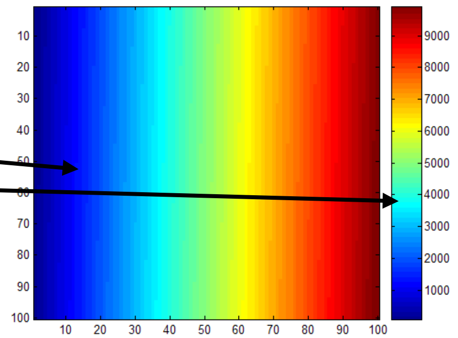
- Save the figure as a pdf

- View with pdf viewer.



Visualizing matrices

- Any matrix can be visualized as an image
  » `mat=reshape(1:10000,100,100);`
  » `imagesc(mat);`
  » `colorbar`

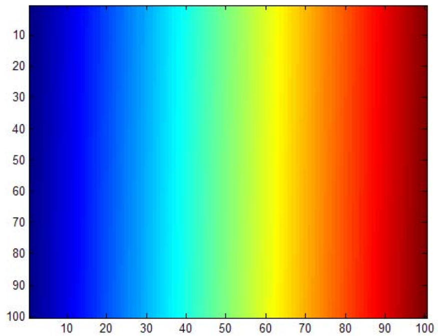


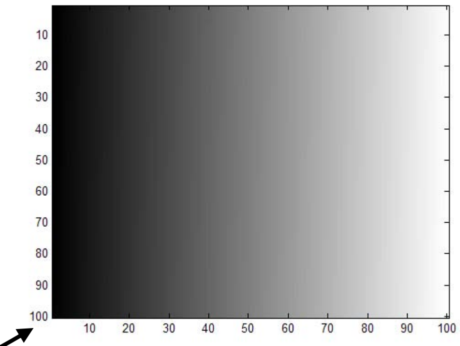
- **imagesc** automatically scales the values to span the entire colormap

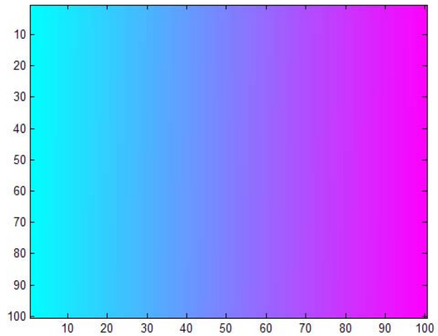
- Can set limits for the color axis (analogous to xlim, ylim)
  » `caxis([3000 7000])`

Colormaps

- You can change the colormap:
  - » **imagesc(mat)**
    - ➢ default map is jet
  - » **colormap(gray)**
  - » **colormap(cool)**
  - » **colormap(hot(256))**

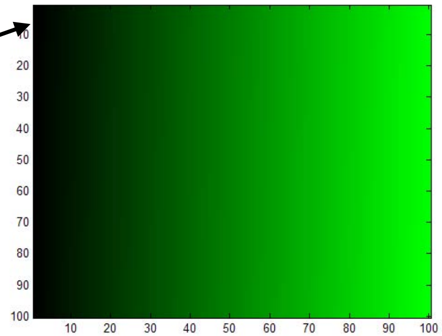


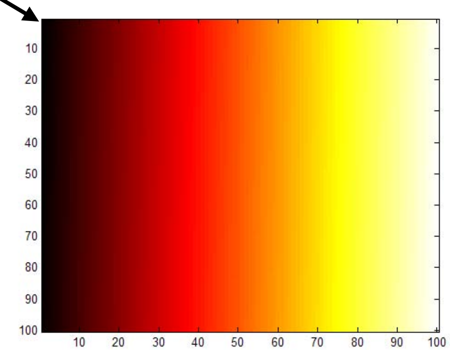




- See help hot for a list

- Can define custom colormap
  - » **map=zeros(256,3);**
  - » **map(:,2)=(0:255)/255;**
  - » **colormap(map);**





Images: Exercise

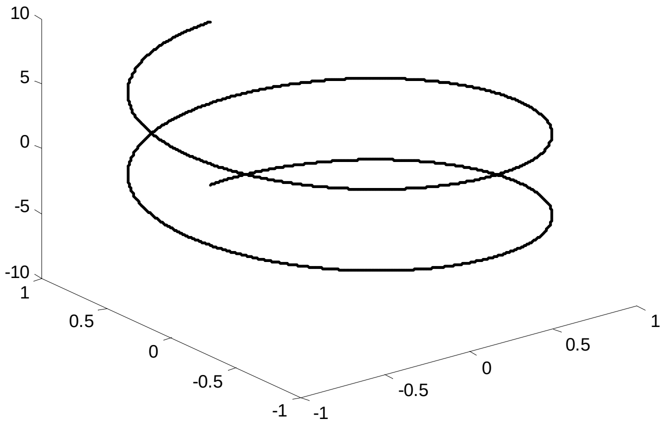
- Construct a Discrete Fourier Transform Matrix of size 128 using **`dftmtx`**
- Display the phase of this matrix as an image using a hot colormap with 256 colors

```
» dMat=dftmtx(128);
» phase=angle(dMat);
» imagesc(phase);
» colormap(hot(256));
```

3D Line Plots

- We can plot in 3 dimensions just as easily as in 2
  - » **time=0:0.001:4*pi;**
  - » **x=sin(time);**
  - » **y=cos(time);**
  - » **z=time;**
  - » **plot3(x,y,z,'k','LineWidth',2);**
  - » **zlabel('Time');**

- Use tools on figure to rotate it
- Can set limits on all 3 axes
  - » **xlim, ylim, zlim**



Surface Plots

- It is more common to visualize *surfaces* in 3D
- Example:

$$f(x,y) = sin(x)cos(y)$$
$$x \in [-\pi,\pi]; y \in [-\pi,\pi]$$

- **surf** puts vertices at specified points in space x,y,z, and connects all the vertices to make a surface
- The vertices can be denoted by matrices X,Y,Z
- How can we make these matrices
  - loop (DUMB)
  - built-in function: **meshgrid**

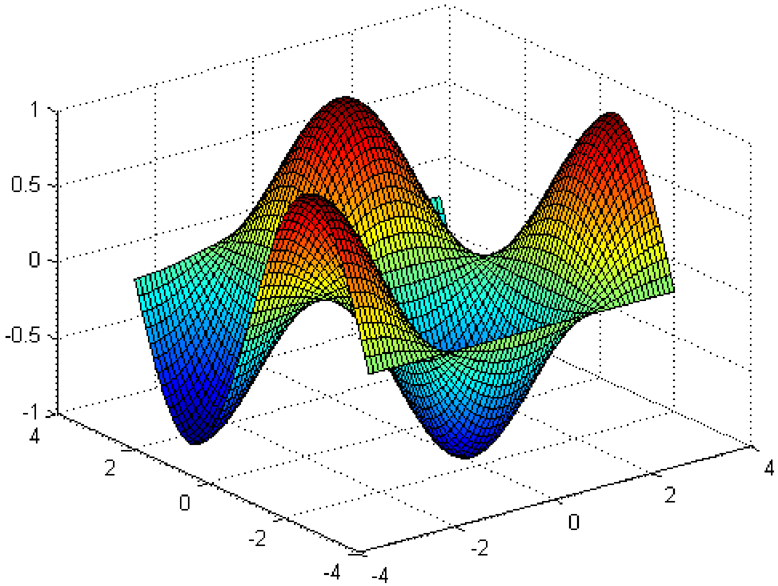
surf

- Make the x and y vectors
  - » `x=-pi:0.1:pi;`
  - » `y=-pi:0.1:pi;`

- Use meshgrid to make matrices (this is the same as loop)
  - » `[X,Y]=meshgrid(x,y);`

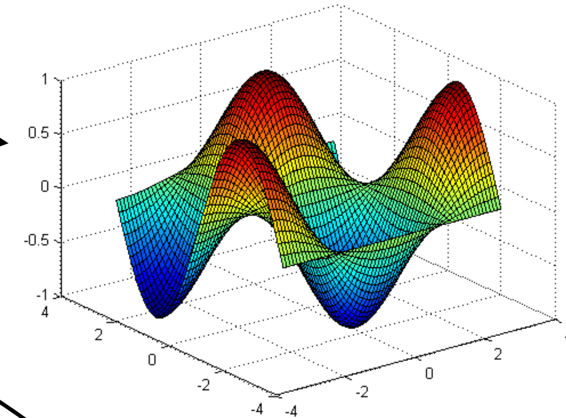
- To get function values, evaluate the matrices
  - » `Z =sin(X).*cos(Y);`

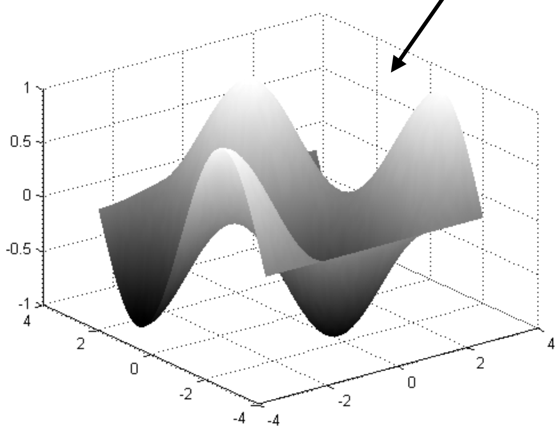
- Plot the surface
  - » `surf(X,Y,Z)`
  - » `surf(x,y,Z);`

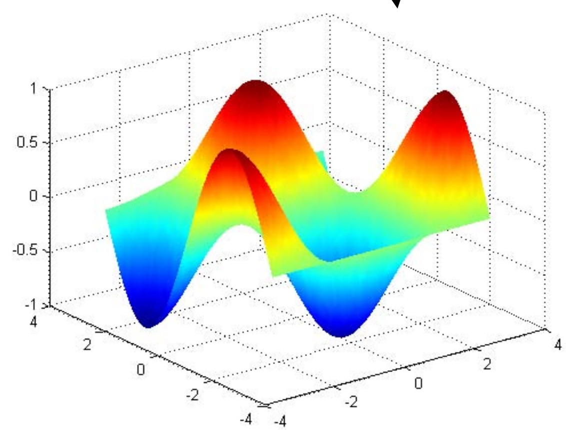


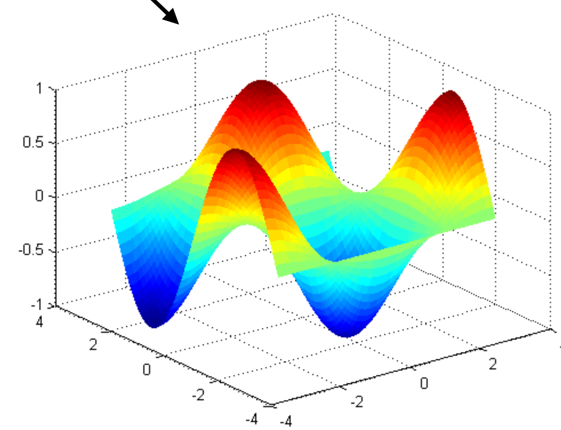
surf Options

- See **help surf** for more options
- There are three types of surface shading
  - » **`shading faceted`**
  - » **`shading flat`**
  - » **`shading interp`**
- You can change colormaps
  - » **`colormap(gray)`**



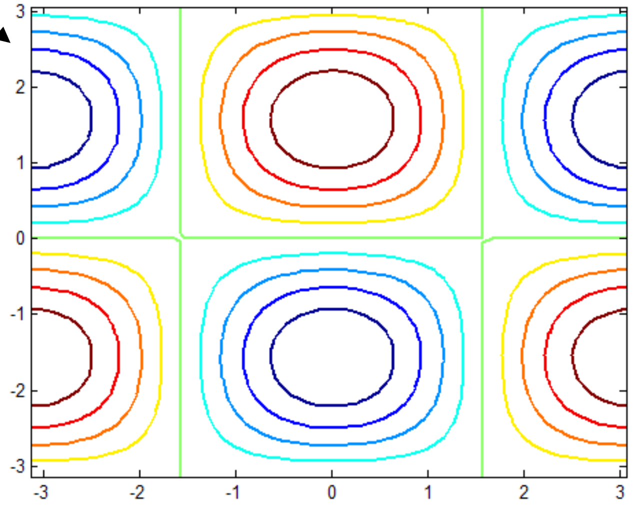


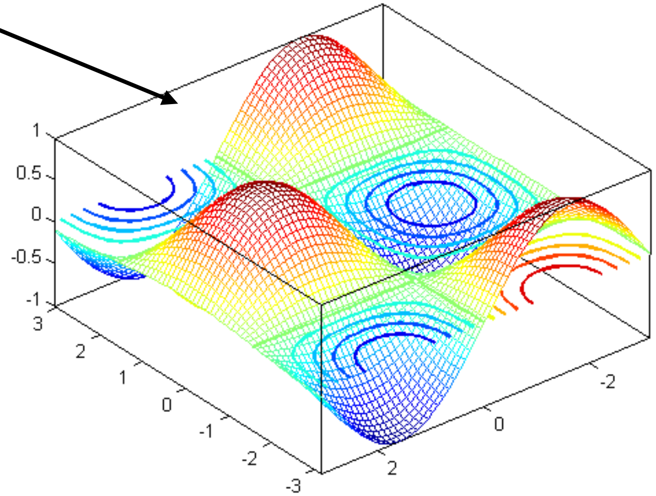




contour

- You can make surfaces two-dimensional by using contour
  - » **`contour(X,Y,Z,'LineWidth',2)`**
    - ➢ takes same arguments as surf
    - ➢ color indicates height
    - ➢ can modify linestyle properties
    - ➢ can set colormap
  - » **`hold on`**
  - » **`mesh(X,Y,Z)`**

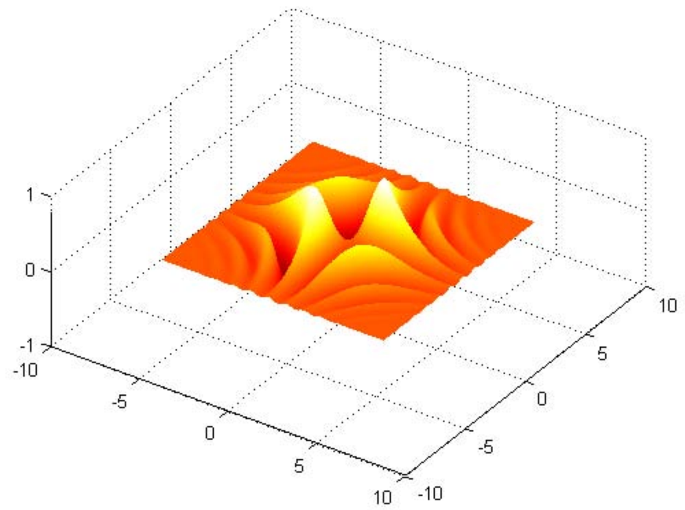




Exercise: 3-D Plots

- Plot exp(-.1(x^2+y^2))*sin(xy) for x,y in [–2*pi,2*pi] with interpolated shading and a hot colormap:

```
» x=-2*pi:0.1:2*pi;
» y=-2*pi:0.1:2*pi;
» [X,Y]=meshgrid(x,y);
» Z =exp(-.1*(X.^2+Y.^2)).*sin(X.*Y);
» surf(X,Y,Z);
» shading interp
» colormap hot
```



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

Outline

(1) Plotting Continued

(2) Scripts

(3) Functions

(4) Flow Control

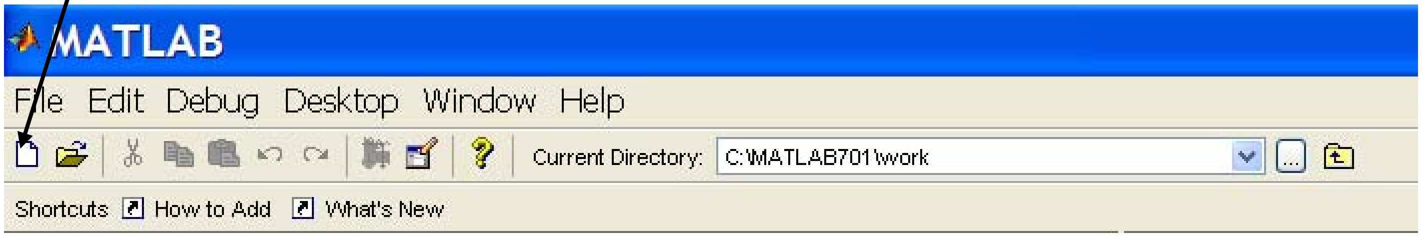
Scripts: Overview

- Scripts are
  - written in the MATLAB editor
  - saved as MATLAB files (.m extension)
  - evaluated line by line

- To create an MATLAB file from command-line

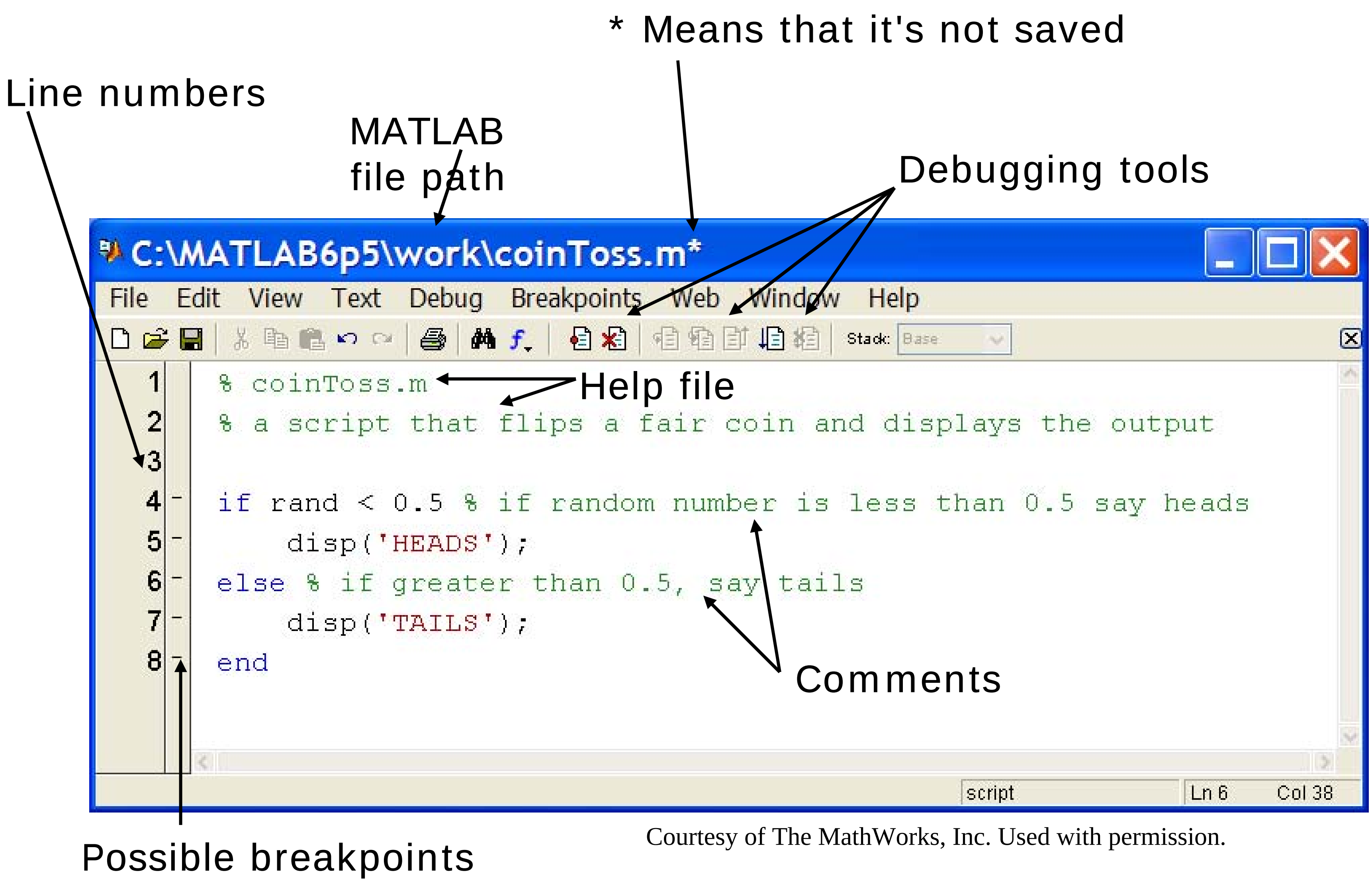
  » **edit myScript.m**

- or click



Courtesy of The MathWorks, Inc. Used with permission.

Scripts: the Editor



Scripts: Good Practice

- Take advantage of "smart indent" option

- Keep code clean
  - ➢ Use built-in functions
  - ➢ Vectorize, vectorize, vectorize
  - ➢ When making large matrices, allocate space first
    - – Use nan or zeros to make a matrix of the desired size

- Keep constants at the top of the MATLAB file

- **COMMENT!**
  - ➢ Anything following a **%** is seen as a comment
  - ➢ The first contiguous comment becomes the script's help file
  - ➢ Comment thoroughly to avoid wasting time later

Hello World

- Here are several flavors of Hello World to introduce MATLAB
- MATLAB will display strings automatically
  » **‘Hello 6.094’**
- To remove “ans =“, use **disp()**
  » **disp('Hello 6.094')**
- **sprintf()** allows you to mix strings with variables
  » **class=6.094;**
  » **disp(sprintf('Hello %g', class))**
    - The format is C-syntax

Exercise: Scripts

- A student has taken three exams. The performance on the exams is random (uniform between 0 and 100)
- The first exam is worth 20%, the second is worth 30%, and the final is worth 50% of the grade
- Calculate the student's overall score
- Save script as practiceScript.m and run a few times

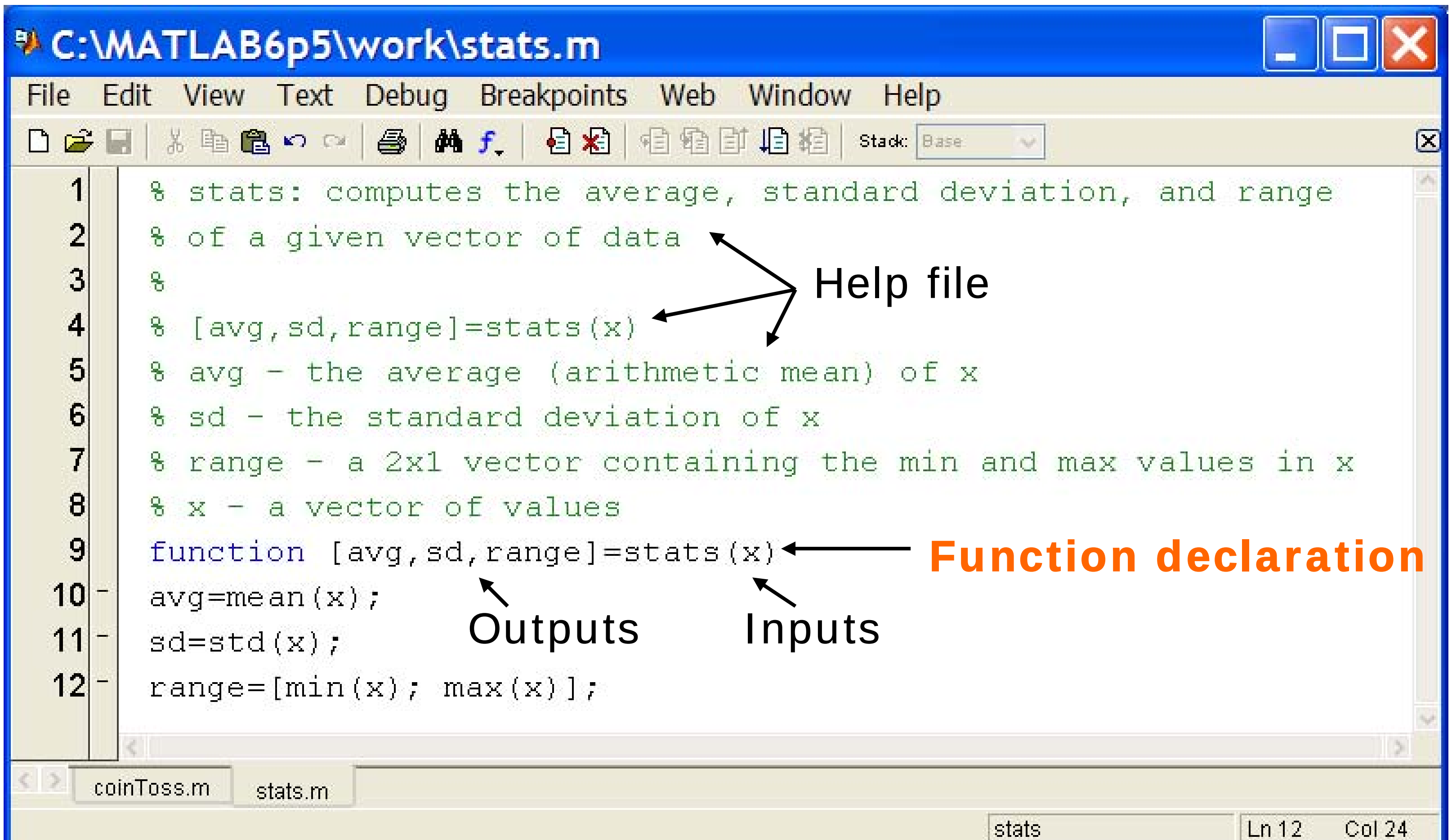
```
» scores=rand(1,3)*100;
» weights=[0.2 0.3 0.5];
» overall=scores*weights'
```

Outline

(1) Plotting Continued
(2) Scripts
(3) Functions
(4) Flow Control

User-defined Functions

- Functions look exactly like scripts, but for **ONE** difference
  - Functions must have a function declaration



User-defined Functions

- Some comments about the function declaration

Inputs must be specified

```
function [x, y, z] = funName(in1, in2)
```

Must have the reserved word: function

If more than one output, must be in brackets

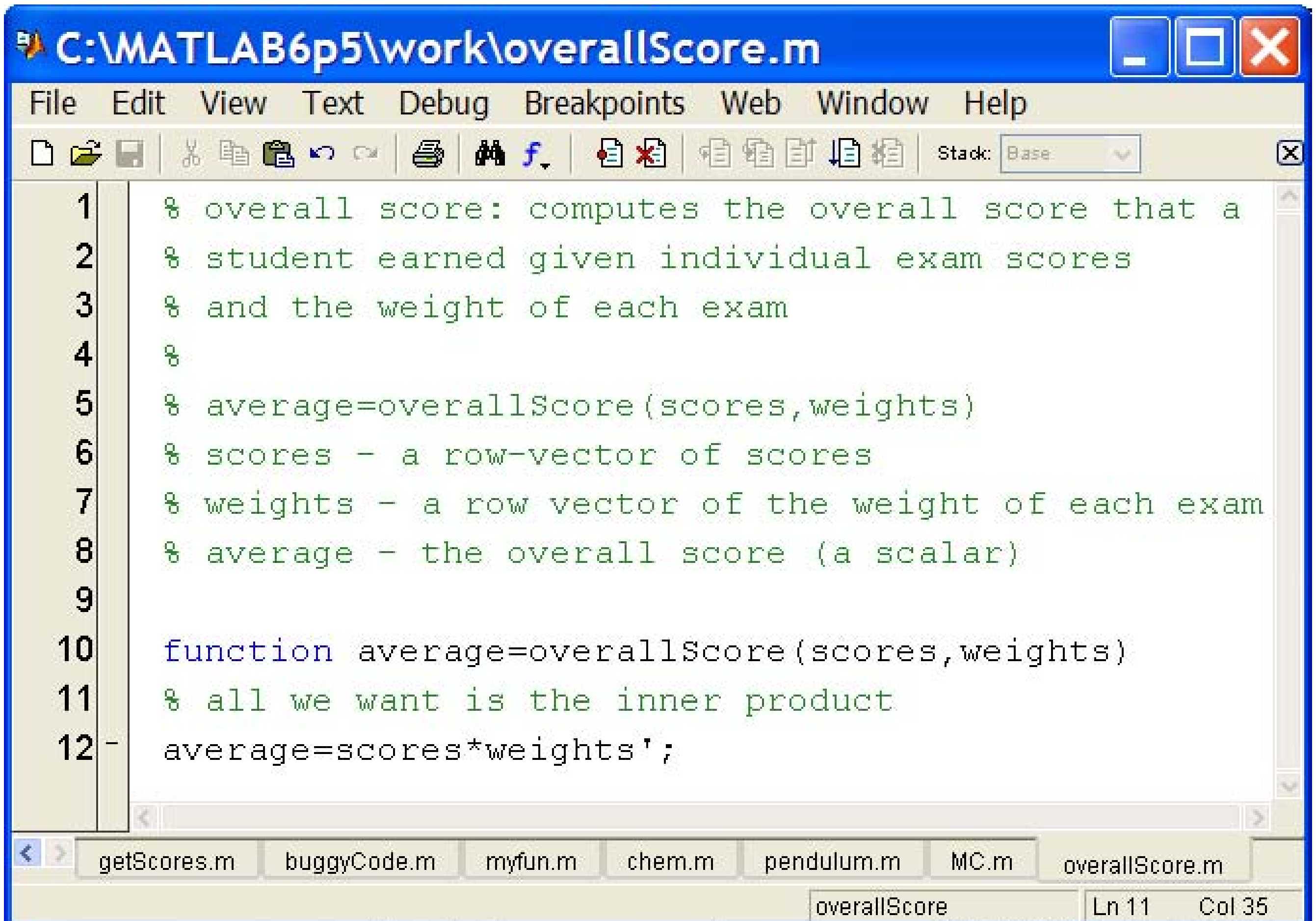
Function name should match MATLAB file name

- 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
- Can have variable input arguments (see **help varargin**)

Functions: Exercise

- Take the script we wrote to calculate the student's overall score and make it into a function
- The inputs should be
  - the scores row vector
  - the weight row vector, with the same length as scores
- The output should be
  - A scalar: the overall score
- Assume the user knows the input constraints (no need to check if the inputs are in the correct format\size)
- Name the function overallScore.m

Functions: Exercise



Courtesy of The MathWorks, Inc. Used with permission.

Functions

- We're familiar with
  - » **`zeros`**
  - » **`size`**
  - » **`length`**
  - » **`sum`**

- Look at the help file for size by typing
  - » **`help size`**

- The help file describes several ways to invoke the function
  - ➢ D = SIZE(X)
  - ➢ [M,N] = SIZE(X)
  - ➢ [M1,M2,M3,...,MN] = SIZE(X)
  - ➢ M = SIZE(X,DIM)

Functions

- 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);
» D=size(a)
» [m,n]=size(a)
» [x,y,z]=size(a)
» m2=size(a,2)
```

- Take advantage of overloaded methods to make your code cleaner!

Outline

(1) Plotting Continued

(2) Scripts

(3) Functions

(4) Flow Control

Relational Operators

- MATLAB uses *mostly* standard relational operators
  - equal ==
  - **not** equal ~=
  - greater than >
  - less than <
  - greater or equal >=
  - less or equal <=
- Logical operators normal bitwise
  - And & &&
  - Or | ||
  - **Not** ~
  - Xor xor
  - All true all
  - Any true any

- Boolean values: zero is false, nonzero is true
- See **help .** for a detailed list of operators

if / else / elseif

- Basic flow-control, common to all languages
- MATLAB syntax is somewhat unique

IF

```
if cond
    commands
end
```

Conditional statement: evaluates to true or false

ELSE

```
if cond
    commands1
else
    commands2
end
```

ELSEIF

```
if cond1
    commands1
elseif cond2
    commands2
else
    commands3
end
```

- No need for parentheses: command blocks are between reserved words

for

- for loops: use for a definite number of iterations
- MATLAB syntax:

Loop variable

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

Command block

- The loop variable
  - 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

- The while is like a more general for loop:
  - Don't need to know number of iterations

```
WHILE

while cond
   commands
end
```

- The command block will execute while the conditional expression is true
- Beware of infinite loops!

Exercise: Control-Flow

- Write a function to calculate the factorial of an integer N using a loop (you can use a for or while loop). If the input is less than 0, return NaN. Test it using some values.

```
» function a = factorial(N)
» if N<0,
»     a=nan,
» else
»     a = 1;
»     for k=1:N
»         a = a*k;
»     end
» end
```

- But note that factorial() is already implemented! You should see if there are built-in functions before implementing something yourself.

```
» which factorial
```

find

- **find** is a very important function
  - Returns indices of nonzero values
  - Can simplify code and help avoid loops

- Basic syntax: index=find(cond)

```
» x=rand(1,100);
» inds = find(x>0.4 & x<0.6);
```

- inds will contain the indices at which x has values between 0.4 and 0.6. This is what happens:
  - x>0.4 returns a vector with 1 where true and 0 where false
  - x<0.6 returns a similar vector
  - The & combines the two vectors using an and
  - The find returns the indices of the 1's

Exercise: Flow Control

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

Using a loop and if/else

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

Being more clever

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

length(x)	Loop time	Find time
100	0.01	0
10,000	0.1	0
100,000	0.22	0
1,000,000	1.5	0.04

- Avoid loops like the plague!
- Built-in functions will make it faster to write and execute

Efficient Code

- Avoid loops whenever possible
  - ➢ This is referred to as vectorization
- Vectorized code is more efficient for MATLAB
- Use indexing and matrix operations to avoid loops
- For example:

```
» a=rand(1,100);
» b=zeros(1,100);
» for n=1:100
»     if n==1
»         b(n)=a(n);
»     else
»         b(n)=a(n-1)+a(n);
»     end
» end
```

➢ Slow and complicated

```
» a=rand(1,100);
» b=[0 a(1:end-1)]+a;
```

➢ Efficient and clean

Exercise: Vectorization

- Alter your factorial program to work WITHOUT a loop. Use **`prod`**

```
» function a=factorial(N)
» a=prod(1:N);
```

- You can tic/toc to see how much faster this is than the loop!

- **BUT**...Don't ALWAYS avoid loops
  - Over-vectorizing code can obfuscate it, i.e. you won't be able to understand or debug it later
  - Sometime a loop is the right thing to do, it is clearer and simple

End of Lecture 2

(1) Plotting Continued

(2) Scripts

(3) Functions

(4) Flow Control

Vectorization makes coding fun!