6.057
Introduction to MATLAB

Orhan Celiker, IAP 2019

Course Layout

Problem sets

- One per day, should take about 4 hours to complete
- Submit Word or PDF, include code and figures
- Some questions optional, but highly recommended!

Requirements for passing

- Attend 3/4 lectures (Friday is optional)
- Complete all problem sets (graded on a 3-level scale: -, √, +)...
- ... and achieve √ average

Prerequisites: You'll be fine!

MATLAB Basics

- MATLAB can be thought of as a super-powerful graphing calculator
  - Remember the TI-83 from calculus?
  - With many more buttons (built-in functions)

- In addition, it is a programming language
  - MATLAB is an interpreted language, like Python
  - Commands are executed line-by-line

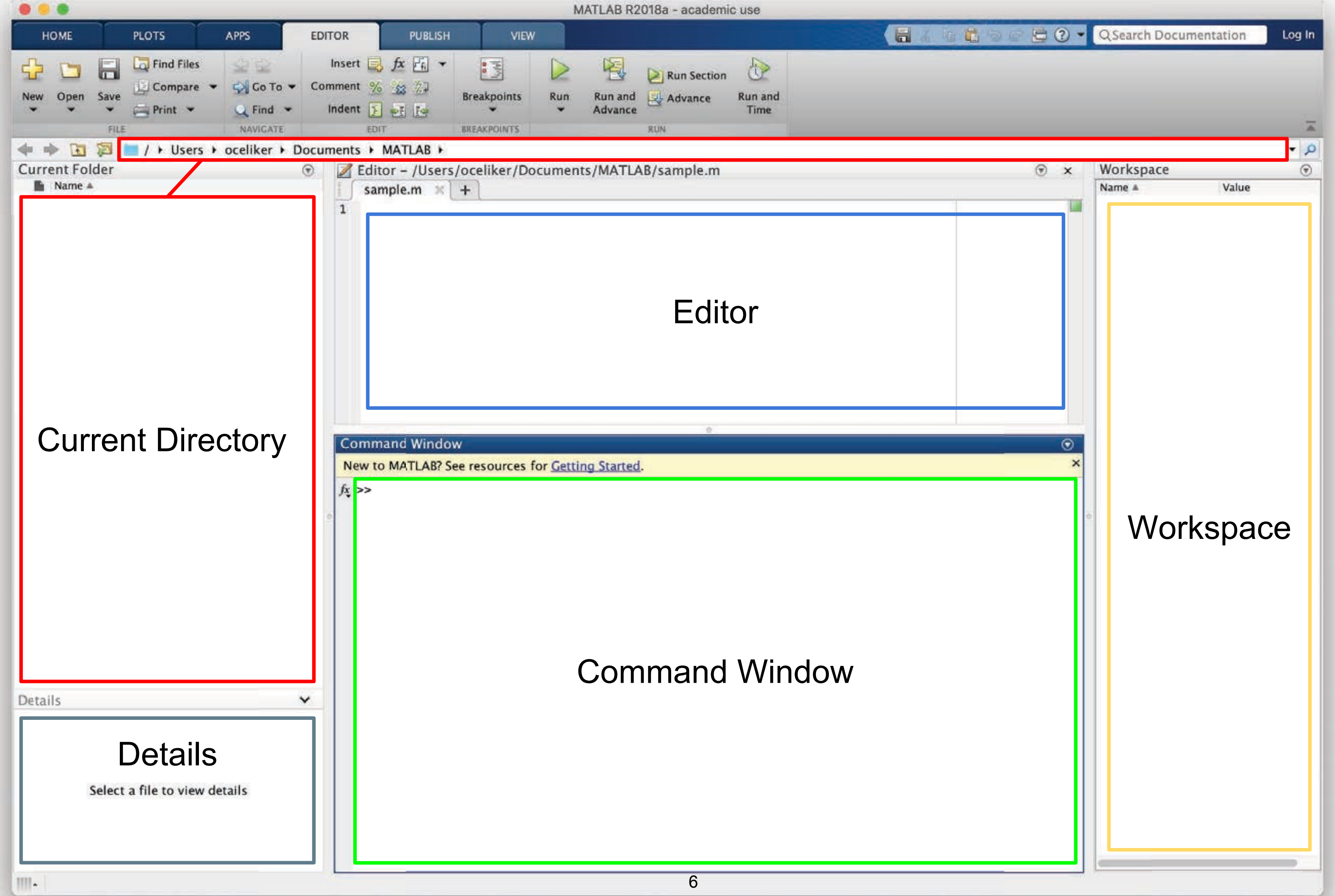
Outline

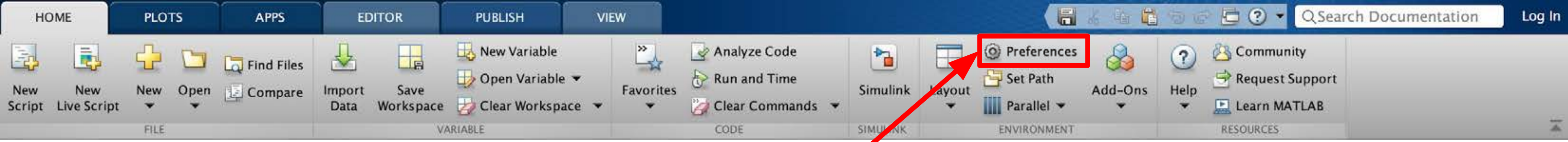
I. Getting Started
II. Scripts
III. Making Variables
IV. Manipulating Variables
V. Basic Plotting

Getting Started

- To get MATLAB Student Version for yourself

- You can also use MATLAB online
  - https://matlab.mathworks.com (requires Mathworks account with license)

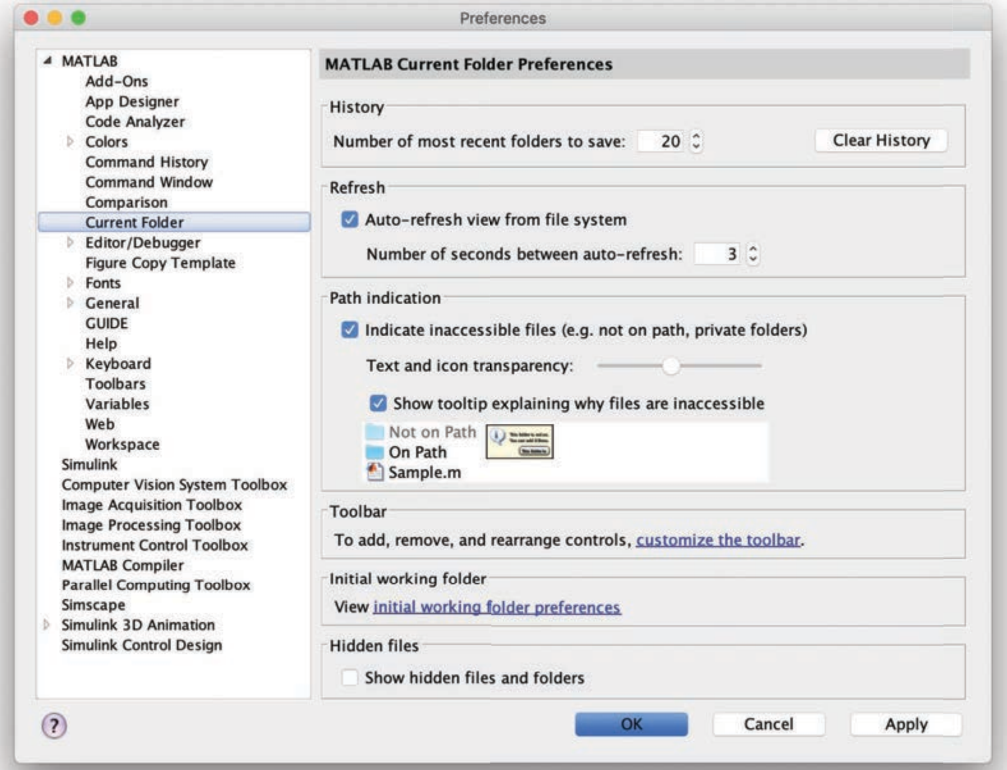

MATLAB R2018a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Log In
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
Run
Run and Advance
Run Section
Advance
Run and Time
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
/ Users oceliker Documents MATLAB
Current Folder
Name
Editor – /Users/oceliker/Documents/MATLAB/sample.m
sample.m
Editor
Workspace
Name
Value
Current Directory
Command Window
New to MATLAB? See resources for Getting Started.
Workspace
Details
Details
Select a file to view details

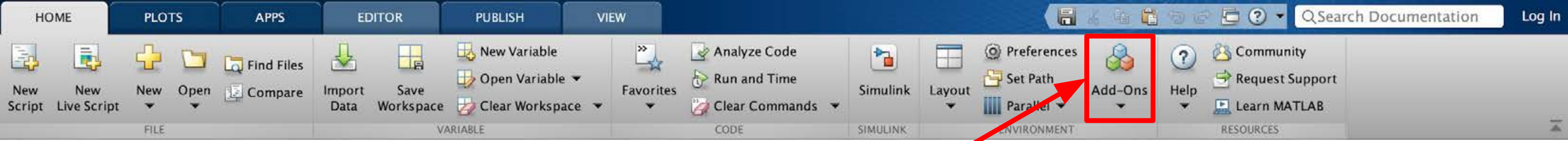


Customization

- In the top ribbon, navigate to:
  Home -> Environment -> Preferences

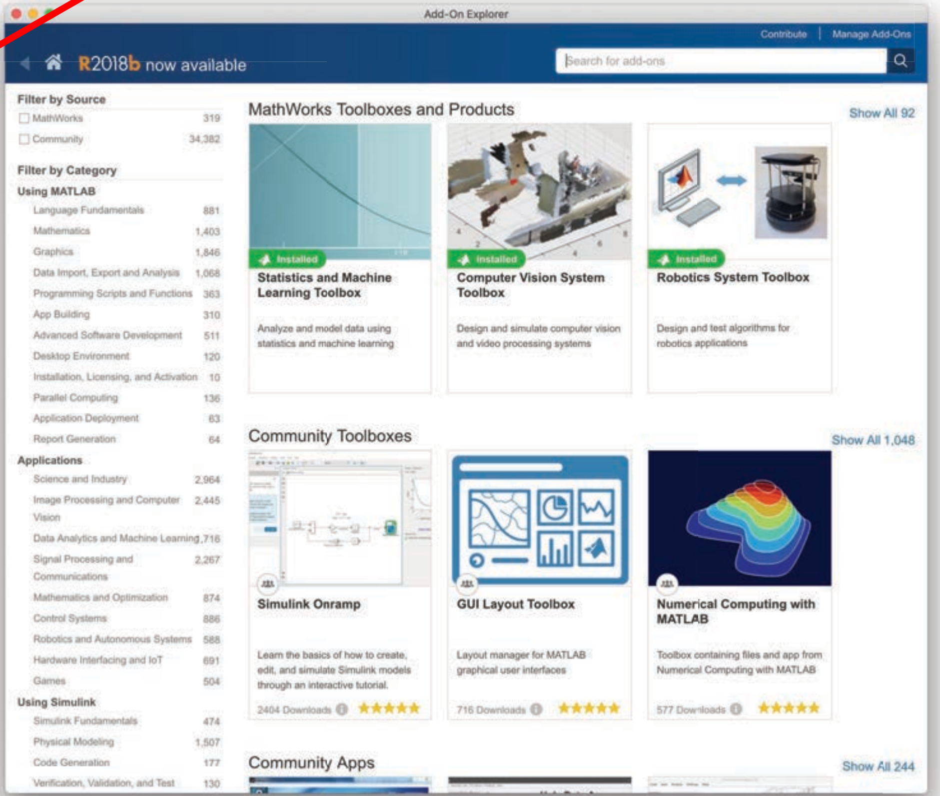
- Allows you to customize your MATLAB experience (colors, fonts, etc.)





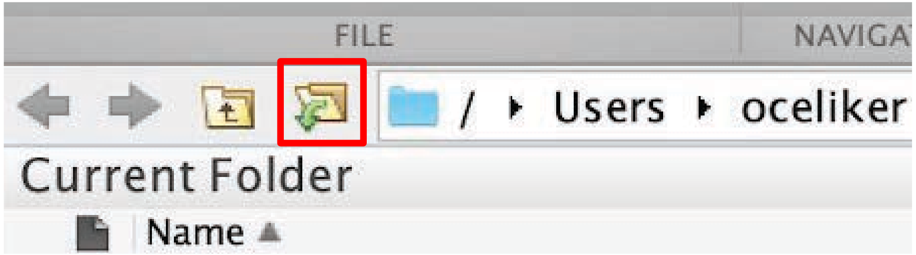
Installing Toolboxes

- In the top ribbon, navigate to: Home -> Environment -> Add-Ons
- Allows you to install toolboxes included with your license
- Recommended toolboxes:
  - Curve Fitting Toolbox
  - Computer Vision System Toolbox
  - Image Processing Toolbox
  - Optimization Toolbox
  - Signal Processing Toolbox
  - and anything related to your field!



Making Folders

- Use folders to keep your programs organized
- To make a new folder, click "Browse" next to the file path



- Click the Make New Folder button, and change the name of the folder. In the MATLAB folder (which should be open by default), make the following folder structure:

```
MATLAB
↳ IAP MATLAB
    ↳ Day1
```

Help/Docs

- `help`
  - The most important command for learning MATLAB on your own!
- To get info on how to use a function:
  - `help sin`
    - Help lists related functions at the bottom and links to the documentation
- To get a nicer version of help with examples and easy-to-read description:
  - `doc sin`
- To search for a function by specifying keywords:
  - `docsearch sin trigonometric`

Outline

I. Getting Started
II. <u>Scripts</u>
III. Making Variables
IV. Manipulating Variables
V. Basic Plotting

Scripts: Overview

- Scripts are
  - Collection of commands executed in sequence
  - Written in the MATLAB editor
  - Saved as m-files (.m extension)
- To create an m-file from the command line:
  - `edit MyFileName.m`
  - or click the "New Script" button on the top left

Scripts: Some notes

- COMMENT!
  - Anything following a % sign is interpreted as a comment
  - The first contiguous comment becomes the script's help file
  - Comment thoroughly to avoid wasting time later!
  - Mark beginning of a code block by using %%
- Note that scripts are somewhat static, with no explicit input and output
- All variables created or modified in a script retain their values after script execution

Exercise: Scripts

- Make a script with the name **helloWorld.m**
- When run, the script should show the following text:

```
Hello world!
I am going to learn MATLAB!
```

Hint: Use `disp(...)` to display strings. Strings are written between single quotes, e.g. `'This is a string'`

Outline

I. Getting Started
II. Scripts
III. Making Variables
IV. Manipulating Variables
V. Basic Plotting

Variable Types

- MATLAB is a "weakly typed" language
    - No need to initialize variables!
- MATLAB supports various types; the most popular ones are
    - 3.84
        - 64-bit double (default)
    - 'A'
        - 16-bit char
- Most variables you'll deal with are vectors, matrices, doubles or chars
- Other types are also supported: complex, symbolic, 16-bit and 8-bit integers (uint16 & uint8), etc.

Naming Variables

- To create a variable, simply assign a value to a name:

```
myNumberVariable = 3.14
myStringVariable = 'hello world!'
```

- Variable name rules
  - First character must be a LETTER
  - After that, any combination of numbers, letters and _
  - Names are CASE-SENSITIVE (e.g. `var1` is different than `Var1`)

Naming Variables (cont.)

Built-in variables (don't use these names for anything else!):

`i, j:` can be used to indicate complex numbers*

`pi:` has the value 3.1415...

`ans:` stores the result of the last unassigned value

`Inf, -Inf:` infinities

`NaN:` "Not a Number"

ops, use `ii, jj, kk`, etc. for loop counters.

Scalars

- A variable can be given a value explicitly
    - `a = 10`
    - Shows up in workspace!
- Or as a function of explicit values and existing variables
    - `c = 1.3 * 45 - 2 * a`
- To suppress output, end the line with a semicolon
    - `cooldude = 13/3;`

Arrays

- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays:
  - Matrix of numbers (either double or complex)
  - Cell array of objects (more advanced data structure)

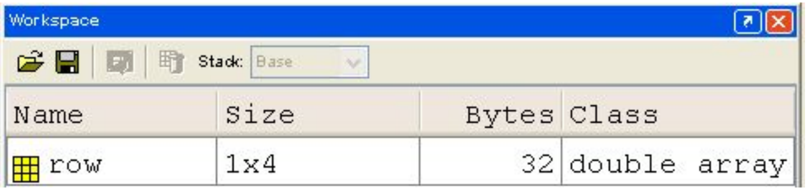
MATLAB makes vectors easy!
That's its power!

Row vectors

- Row vector: comma- or space-separated values between square brackets
  - `row = [ 1 2 3.2 4 6 5.4 ];`
  - `row = [ 1, 2, 4, 7, 4.3, 1.1 ];`
- Command window:

```
>> row=[1 2 5.4 -6.6]

row =

    1.0000    2.0000    5.4000   -6.6000
```

- Workspace:



Column vectors

- Column vector: semicolon-separated values between square brackets
    - `col = [ 1; 2; 3.2; 4; 6; 5.4 ];`
- Command window:

```
>> column=[4;2;7;4]

column =

     4
     2
     7
     4
```

- Workspace:

Workspace

Stack: Base

Name	Size	Bytes	Class
column	4x1	32	double array

Size and length

- You can tell the difference between a row and a column by:
  - Looking in the workspace
  - Displaying the variable in the command window
  - Using the size function

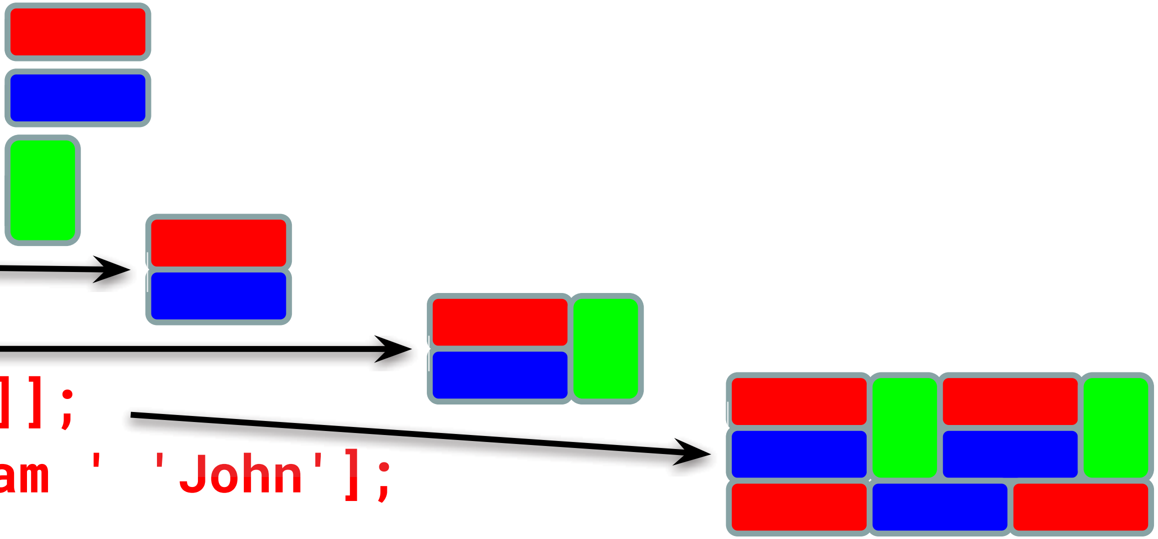
```
>> size(row)

ans =

     1     4

>> length(row)

ans =

     4
```

```
>> size(column)

ans =

     4     1

>> length(column)

ans =

     4
```

Matrices

- Make matrices like vectors
  - Element by element
    - `a= [1 2;3 4];` $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
- By concatenating vectors or matrices (dimension matters)

```
a = [1 2];
b = [3 4];
c = [5;6];
d = [a;b];
e = [d c];
f = [[e e];[a b a]];
str = ['Hello, I am ' 'John'];
```



- Strings are character vectors

save/clear/load

- Use save to save variables to a file
  - `save myFile a b`
  - Saves variables a and b to the file myFile.mat in the current directory
  - Default working directory is MATLAB unless you navigate to another folder
  - Make sure you are in the correct folder. Right now we should be in \MATLAB\IAP MATLAB\Day 1
- Use clear to save variables to a file
  - `clear a b`
  - Look at workspace: variables a and b are gone
- Use load to load variables into the workspace
  - `load myFile`
  - Look at workspace: a and b are back

Exercise: Variables

Get and save the current date and time

- Create a variable `start` using the function `clock`
- What is the size of `start`? Is it a row or column?
- What does `start` contain? See `help clock`
- Convert the vector start to a string. Use the function `datestr` and name the new variable `startString`
- Save `start` and `startString` into a mat file named `startTime`

Exercise: Variables II

- In helloWorld.m, read in variables you saved using `load`
- Display the following text:

  I started learning MATLAB on [date, time]

- Hint: Use the `disp` command again
- Remember that strings are just vectors of characters, so you can join two strings by making a row vector with the two strings as sub-vectors.

Outline

Basic Scalar Operations

- Arithmetic operations (+, -, *, /)
  - `7/45`
  - `(1+1i)*(1+2i)`
  - `1/0`
  - `0/0`
- Exponentiation
  - `4^2`
  - `(3+4*1j)^2`
- Complicated expressions: use parentheses
  - `((2+3)*3)^0.1`

Built-in Functions

- MATLAB has an <u>enormous</u> library of built-in functions
- Call using parentheses, passing parameters to function
  - `sqrt(2)`
  - `log(2), log10(0.23)`
  - `cos(1.2), atan(-.8)`
  - `exp(2+4*1i)`
  - `round(1.4), floor(3.3), ceil(4.23)`
  - `angle(1i); abs(1+1i);`

Exercise: Scalars

`helloWorld` script:

- Your learning time constant is 1.5 days. Calculate the number of seconds in 1.5 days and name this variable `tau`
- This class lasts 5 days. Calculate the number of seconds in 5 days and name this variable `endOfClass`
- This equation describes your knowledge as a function of time t:

$$k = 1 - e^{-t/\tau}$$

- How well will you know MATLAB at `endOfClass`? Name this variable `knowledgeAtEnd` (use exp)
- Using the value of `knowledgeAtEnd`, display the phrase:

```
At the end of 6.057, I will know X% of MATLAB
```

Hint: to convert a number to a string, use `num2str`

Transpose

- The transpose operator turns a column vector into a row vector, and vice versa
  - `a = [1 2 3 4+i]`
  - `transpose(a)`
  - `a'`
  - `a.'`
- The ' gives the Hermitian-transpose
  - Transposes and conjugates all complex numbers
- For vectors of real numbers .' and ' give same result
  - For transposing a vector, always use .' to be safe

Addition and Subtraction

- Addition and subtraction are element-wise
- Sizes must match (unless one is a scalar):

$$\begin{array}{r} [12 \quad 3 \quad 32 \quad -11] \\ +[2 \quad 11 \quad -30 \quad 32] \\ \hline =[14 \quad 14 \quad 2 \quad 21] \end{array}$$

$$\begin{bmatrix} 12 \\ 1 \\ -10 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 13 \\ 33 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ -23 \\ -33 \end{bmatrix}$$

Addition and Subtraction

- `c = row + column`

Use the transpose to make sizes compatible

- `c = row.’ + column`
- `c = row + column.’`

Can sum up or multiply elements of vector

- `s=sum(row);`
- `p=prod(row);`

Element-wise functions

- All the functions that work on scalars also work on vectors
  - t = [1 2 3];
    f = exp(t);
    is the same as
    f = [exp(1) exp(2) exp(3)];
- If in doubt, check a function's help file to see if it handles vectors element-wise
- Operators (* / ^) have two modes of operation
  - element-wise
  - standard

Element-wise functions

- To do element-wise operations, use the dot: . (.*, ./, .^)
- BOTH dimensions must match (unless one is scalar)!

```
a=[1 2 3];b=[4;2;1];

a.*b  , a./b  , a.^b → all errors

a.*b.', a./b.’, a.^(b.’) → all valid
```

Operators

- Multiplication can be done in a standard way or element-wise
- Standard multiplication (*) is matrix product
    - Remember from linear algebra: inner dimensions must MATCH!!
- Standard exponentiation (^) can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse
    - Our recommendation: for now, just multiply by inverse (more on this later)

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = 11$$

$$1 \times 3 * 3 \times 1 = 1 \times 1$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} ^\wedge 2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Must be square to do powers

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 6 & 12 & 18 \\ 9 & 18 & 27 \end{bmatrix}$$

$$3 \times 3 * 3 \times 3 = 3 \times 3$$

Exercise: Vector Operations

Calculate how many seconds elapsed since start of class

- In helloWorld.m, make variables called secPerMin, secPerHour, secPerDay, secPerMonth (assume 30.5 days per month), and secPerYear (12 months in year), which have the number of seconds in each time period
- Assemble a row vector called secondConversion that has elements in this order: secPerYear, secPerMonth, secPerDay, secPerHour, secPerMin, 1
- Make a currentTime vector by using clock
- Compute elapsedTime by subtracting currentTime from start
- Compute t (the elapsed time in seconds) by taking the dot product of secondConversion and elapsedTime (transpose one of them to get the dimensions right)

Exercise: Vector Operations

Display the current state of your knowledge

- Calculate currentKnowledge using the same relationship as before, and the t we just calculated:

$$k = 1 - e^{-t/\tau}$$

- Display the following text:
  At this time, I know X% of MATLAB

Automatic Initialization

- Initialize a vector of **ones**, **zeros**, or **random** numbers
  - » `o=ones(1,10)`
    - ➢ Row vector with 10 elements, all 1
  - » `z=zeros(23,1)`
    - ➢ Column vector with 23 elements, all 0
  - » `r=rand(1,45)`
    - ➢ Row vector with 45 elements (uniform (0,1))
  - » `n=nan(1,69)`
    - ➢ Row vector of NaNs (representing uninitialized variables)

Automatic Initialization

- To initialize a linear vector of values use **linspace**
  - » `a=linspace(0,10,5)`
    - ➢ Starts at 0, ends at 10 (inclusive), 5 values

- Can also use colon operator (**:**)
  - » `b=0:2:10`
    - ➢ Starts at 0, increments by 2, and ends at or before 10
    - ➢ Increment can be decimal or negative
  - » `c=1:5`
    - ➢ If increment is not specified, default is 1

- To initialize logarithmically spaced values use **logspace**
    - ➢ Similar to **linspace**, but see **help**

Exercise: Vector Functions

Calculate your learning trajectory

- In helloWorld.m, make a linear time vector `tVec` that has 10,000 samples between 0 and `endOfClass`
- Calculate the value of your knowledge (call it `knowledgeVec`) at each of these time points using the same equation as before:

$$k = 1 - e^{-t/\tau}$$

Vector Indexing

- <u>MATLAB indexing starts with **1**, not **0**</u>
    - ➢ We will not respond to any emails where this is the problem.
- a(n) returns the nth element

$$a = [13 \quad 5 \quad 9 \quad 10]$$

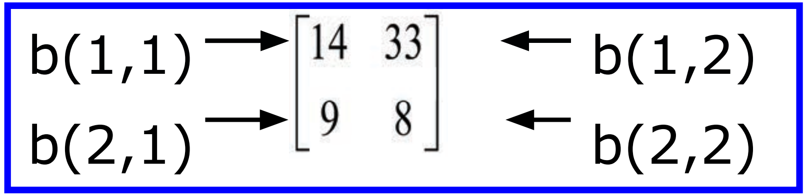
a(1) a(2) a(3) a(4)

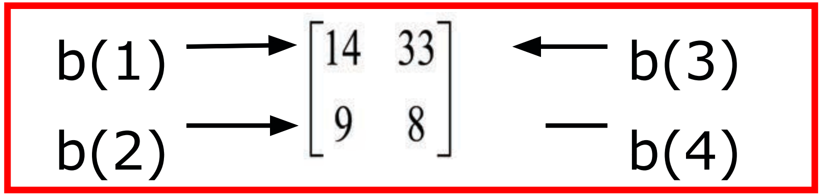
- The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.


```
» x=[12 13 5 8];
```

Matrix Indexing

- Matrices can be indexed in two ways
  - ➢ using **subscripts** (row and column)
  - ➢ using linear **indices** (as if matrix is a vector)
- Matrix indexing: subscripts or linear indices





- Picking submatrices


```
» A = rand(5) % shorthand for 5x5 matrix
```

Advanced Indexing 1

- To select rows or columns of a matrix, use the **:**

$$c = \begin{bmatrix} 12 & 5 \\ -2 & 13 \end{bmatrix}$$

```
» d=c(1,:);          d=[12 5];
» e=c(:,2);          e=[5;13];
» c(2,:)=[3 6];  %replaces second row of c
```

Advanced Indexing 2

- MATLAB contains functions to help you find desired values
  - » `vec = [5 3 1 9 7]`
- To get the minimum value and its index (similar for `max`):
  - » `[minVal,minInd] = min(vec);`
- To find the indices of specific values or ranges
  - » `ind = find(vec == 9); vec(ind) = 8;`
  - » `ind = find(vec > 2 & vec < 6);`
    - ➢ **find** expressions can be very complex, more on this later
    - ➢ When possible, **logical indexing** is faster than **find**!
    - ➢ E.g., `vec(vec == 9) = 8;`

Exercise: Indexing

When will you know 50% of MATLAB?

- First, find the index where **knowledgeVec** is closest to 0.5. Mathematically, what you want is the index where the value of ~ $|knowledgeVec - 0.5|$ is at a minimum (use **abs** and **min**)
- Next, use that index to look up the corresponding time in **tVec** and name this time **halfTime**
- Finally, display the string:
  Convert **halfTime** to days by using secPerDay. I will know half of MATLAB after X days

Outline

(1) Getting Started
(2) Scripts
(3) Making Variables
(4) Manipulating Variables
(5) **Basic Plotting**

Did everyone sign in?

Plotting

- Example

```
» x=linspace(0,4*pi,10);
» y=sin(x);
```

- Plot values against their index

```
» plot(y);
```

- Usually we want to plot y versus x

```
» plot(x,y);
```

MATLAB makes visualizing data fun and easy!

What does plot do?

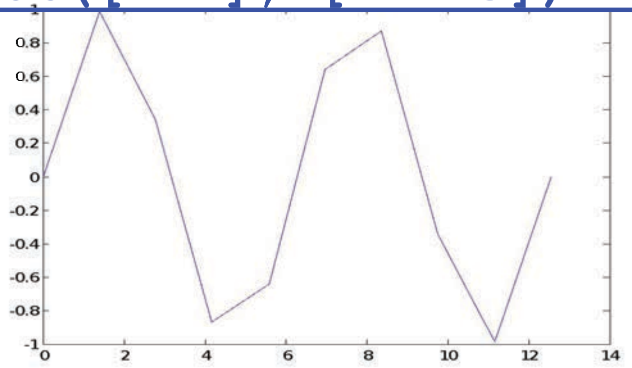
- **plot** generates dots at each (x,y) pair and then connects the dots with a line
- To make plot of a function look smoother, evaluate at more points

```
» x=linspace(0,4*pi,1000);
» plot(x,sin(x));
```

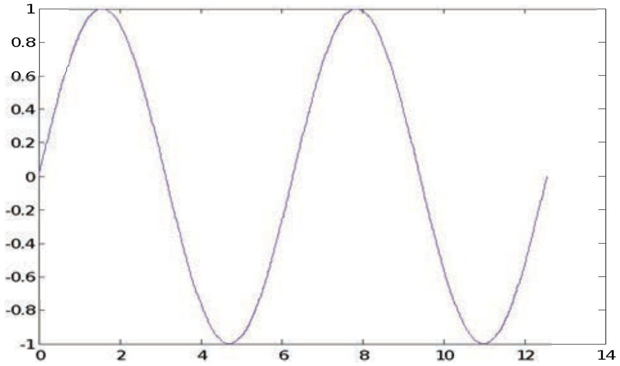
- x and y vectors must be same size or else you'll get an error

```
» plot([1 2], [1 2 3])
```

10 x values:



1000 x values:



Exercise: Plotting

Plot the learning trajectory

- In helloWorld.m, open a new figure (use `figure`)
- Plot knowledge trajectory using `tVec` and `knowledgeVec`
- When plotting, convert `tVec` to days by using `secPerDay`
- Zoom in on the plot to verify that `halfTime` was calculated correctly

End of Lecture 1

(1) Getting Started

(2) Scripts

(3) Making Variables

(4) Manipulating Variables

(5)

Hope that wasn't too much and you enjoyed it!!

6.057
Introduction to programming in MATLAB

Lecture 2: Visualization and Programming

Orhan Celiker

IAP 2019

Homework 1 Recap

Some things that came up:

- Plotting a straight line
  - » `x = 1:10`
  - » `plot(x, 0)`
    - ➢ Not an error, but probably not what you meant

- Use of semicolon – never required if one command per line. You can also put multiple commands on one line; in this case, a semicolon is necessary to separate commands:
  - » `x=1:10; y=(x-5).^2; z = x.*y;`

Plotting

- Example
  - » `x=linspace(0,4*pi,10);`
  - » `y=sin(x);`

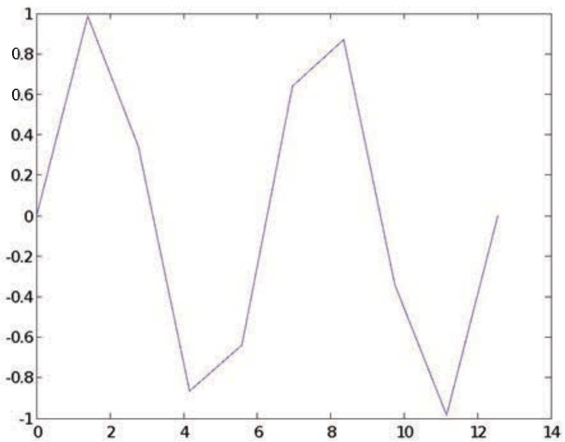
- Plot values against their index
  - » `plot(y);`
- Usually we want to plot y versus x
  - » `plot(x,y);`

MATLAB makes visualizing data fun and easy!

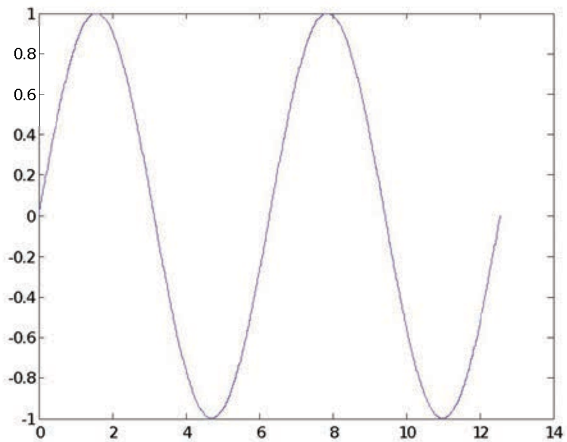
What does plot do?

- **plot** generates dots at each (x,y) pair
  and then connects the dots with a line
- To make plot of a function look smoother, evaluate at more points
  - » `x=linspace(0,4*pi,1000);`
  - » `plot(x,sin(x));`
- x and y vectors must be same size or else you'll get an error
  - » `plot([1 2], [1 2 3])`
    - ➢ error!!

10 x values:



1000 x values:



Exercise: Plotting

Plot the learning trajectory

- In helloWorld.m, open a new figure (use **`figure`**)
- Plot knowledge trajectory using **`tVec`** and **`knowledgeVec`**
- When plotting, convert **`tVec`** to days by using **`secPerDay`**
- Zoom in on the plot to verify that **`halfTime`** was calculated correctly

Outline for Lec 2

(1) **Functions**
(2) Flow Control
(3) Line Plots
(4) Image/Surface Plots
(5) Efficient Codes
(6) Debugging

User-defined Functions

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

C:\MATLAB6p5\work\stats.m

File Edit View Text Debug Breakpoints Web Window Help

```
% stats: computes the average, standard deviation, and range
% of a given vector of data
%
% [avg,sd,range]=stats(x)
% avg - the average (arithmetic mean) of x
% sd - the standard deviation of x
% range - a 2x1 vector containing the min and max values in x
% x - a vector of values
function [avg,sd,range]=stats(x)
avg=mean(x);
sd=std(x);
range=[min(x); max(x)];
```

Help file

Function declaration

Outputs

Inputs

coinToss.m stats.m

stats Ln 12 Col 24

User-defined Functions

- Some comments about the function declaration

Inputs

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

Must have the reserved word: function

Function name should match m-file name

If more than one output, must be in brackets

- No need for return: MATLAB 'returns' the variables whose names match those in the function declaration (though, you can use **`return`** to break and go back to invoking function)
- Variable scope: Any variable created within the function but not returned disappears after the function stops running (They're called "local variables")

Functions: overloading

- 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: 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)
```

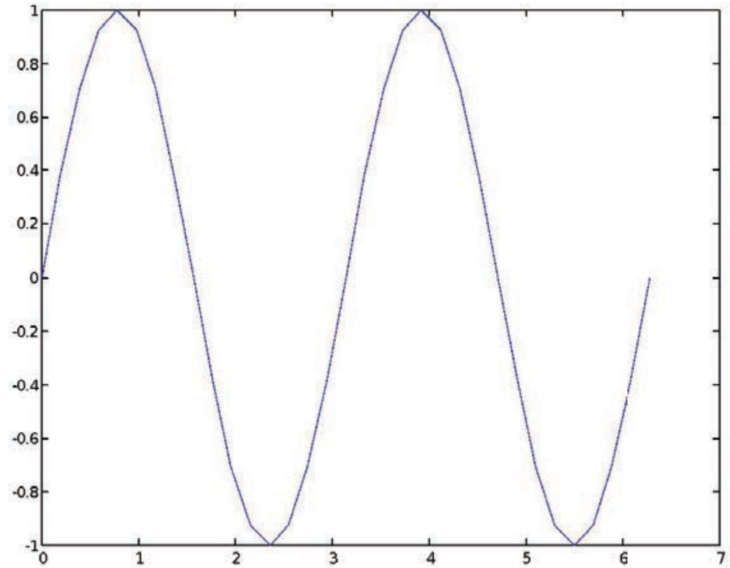
- You can overload your own functions by having variable number of input and output arguments (see `varargin`, `nargin`, `varargout`, `nargout`)

Functions: Exercise

- Write a function with the following declaration:
  `function plotSin(f1)`

- In the function, plot a sine wave with frequency f1, on the interval [0,2π]: $\sin\left(f_1 x\right)$

- To get good sampling, use 16 points per period.



Outline

(1) Functions
(2) **Flow Control**
(3) Line Plots
(4) Image/Surface Plots
(5) Efficient Codes
(6) Debugging

Relational Operators

- MATLAB uses *mostly* standard relational operators
  - equal ==
  - **not** equal ~=
  - greater than >
  - less than <
  - greater or equal >=
  - less or equal <=
- Logical operators elementwise short-circuit (scalars)
  - 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
- Lots of `elseif`'s? consider using `switch`

for

- **for** loops: use for a known 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 (but it's usually cleaner if they're consecutive)
- The command block
  - ➢ Anything between the **for** line and the **end**

while

- The while is like a more general for loop:
  - No 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! CTRL+C?!
- You can use `break` to exit a loop

Exercise: Conditionals

- Modify your `plotSin(f1)` function to take two inputs: `plotSin(f1,f2)`

- If the number of input arguments is 1, execute the plot command you wrote before. Otherwise, display the line `'Two inputs were given'`
- Hint: the number of input arguments is stored in the built-in variable `nargin`

Outline

(1) Functions
(2) Flow Control
(3) **Line Plots**
(4) Image/Surface Plots
(5) Efficient Codes
(6) Debugging

Plot Options

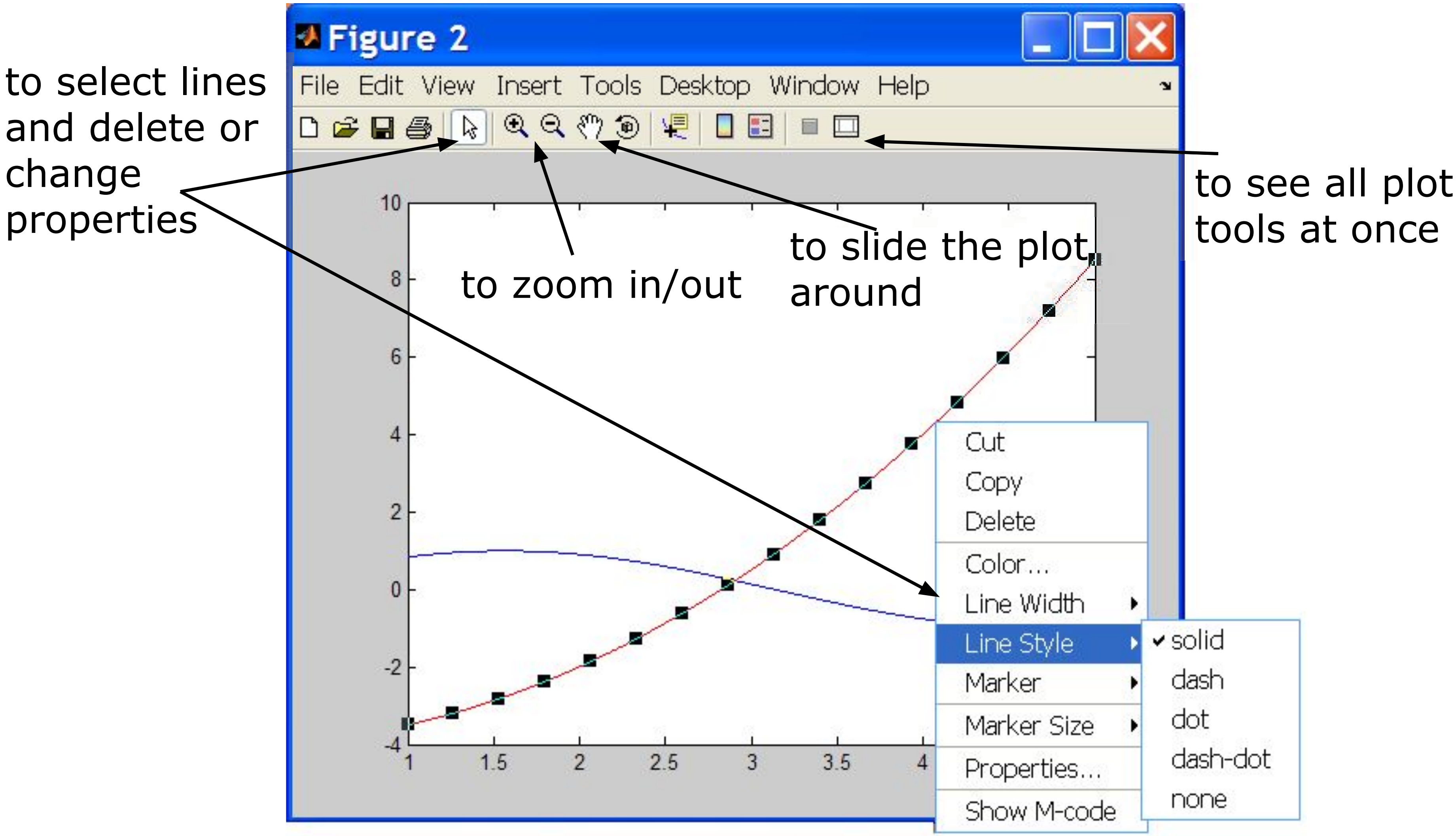
- Can change the line color, marker style, and line style by adding a string argument
  - » `plot(x,y,'k.-');`

    color marker line-style

- Can plot without connecting the dots by omitting line style argument
  - » `plot(x,y,'.')`

- Look at **help plot** for a full list of colors, markers, and line styles

Playing with the Plot

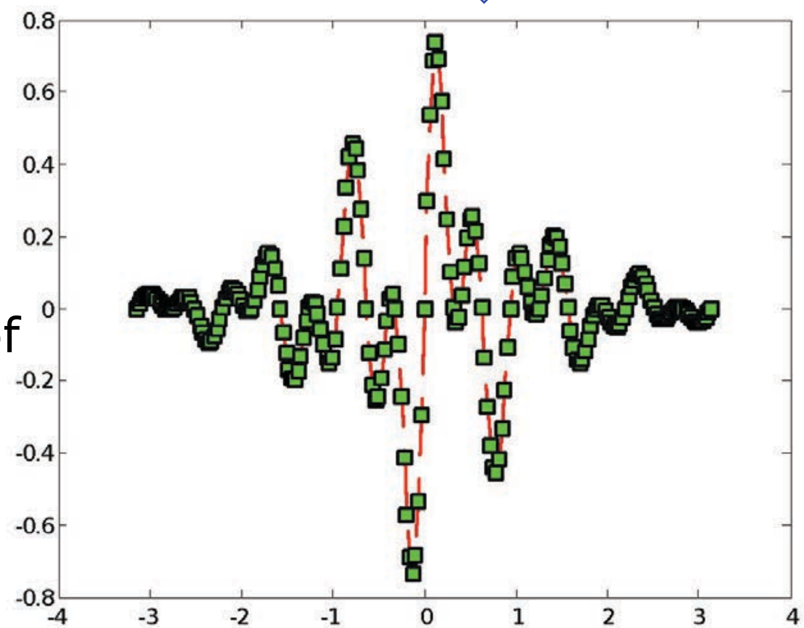


Line and Marker Options

- Everything on a line can be customized

```
» plot(x,y,'s--','LineWidth',2,...
           'Color', [1 0 0], ...
           'MarkerEdgeColor','k',...
           'MarkerFaceColor','g',...
           'MarkerSize',10)
```

You can set colors by using a vector of [R G B] values or a predefined color character like 'g', 'k', etc.



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

Cartesian Plots

- We have already seen the plot function

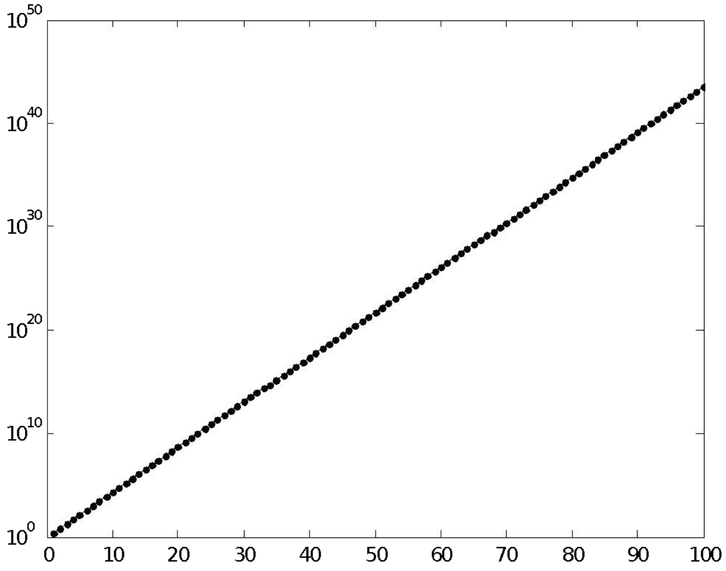
```
» x=-pi:pi/100:pi;
» y=cos(4*x).*sin(10*x).*exp(-abs(x));
» plot(x,y,'k-');
```

- 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.-');
```



3D Line Plots

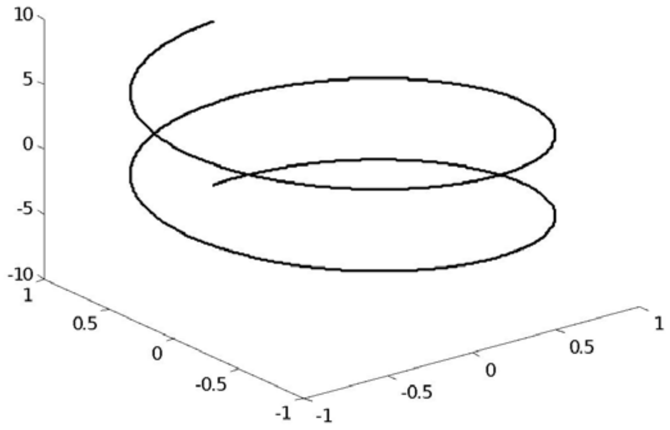
- We can plot in 3 dimensions just as easily as in 2D

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

3D Line Plots

- We can plot in 3 dimensions just as easily as in 2D
  - » `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`



Axis Modes

- Built-in axis modes (see `doc axis` for more modes)

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

  » `axis tight`
  - ➢ fits axes to data

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

  » `axis xy`
  - ➢ puts the origin in the lower left corner (default for plots)

  » `axis ij`
  - ➢ puts the origin in the upper left corner (default for matrices/images)

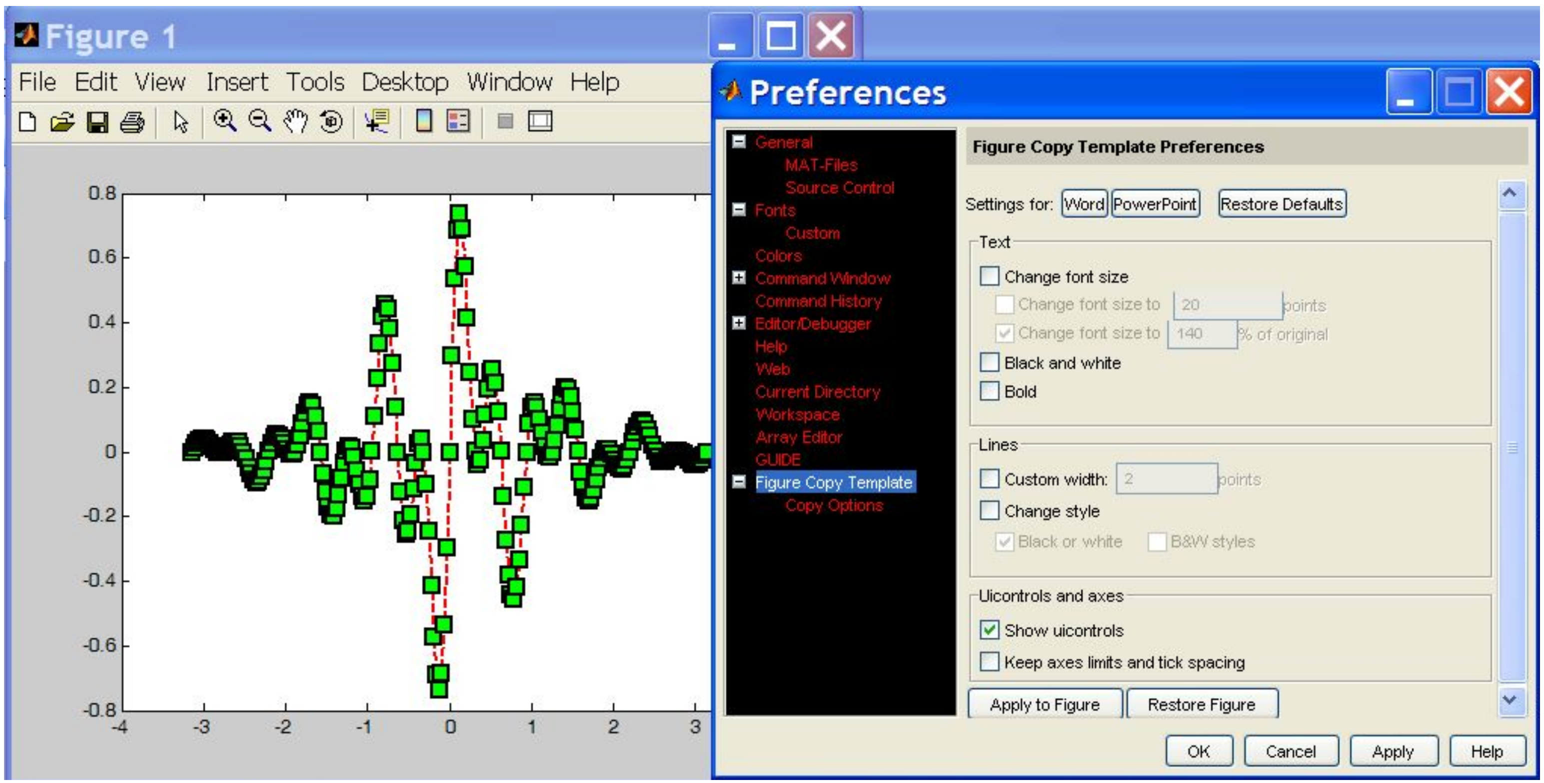
Multiple Plots in one Figure

- To have multiple axes in one figure
  - » `subplot(2,3,1)`
    - ➢ makes a figure with 2 rows and 3 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)`
    - ➢ activates a range of axes and fuses them into one

- To close existing figures
  - » `close([1 3])`
    - ➢ closes figures 1 and 3
  - » `close all`
    - ➢ closes all figures (useful in scripts)

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



Saving Figures

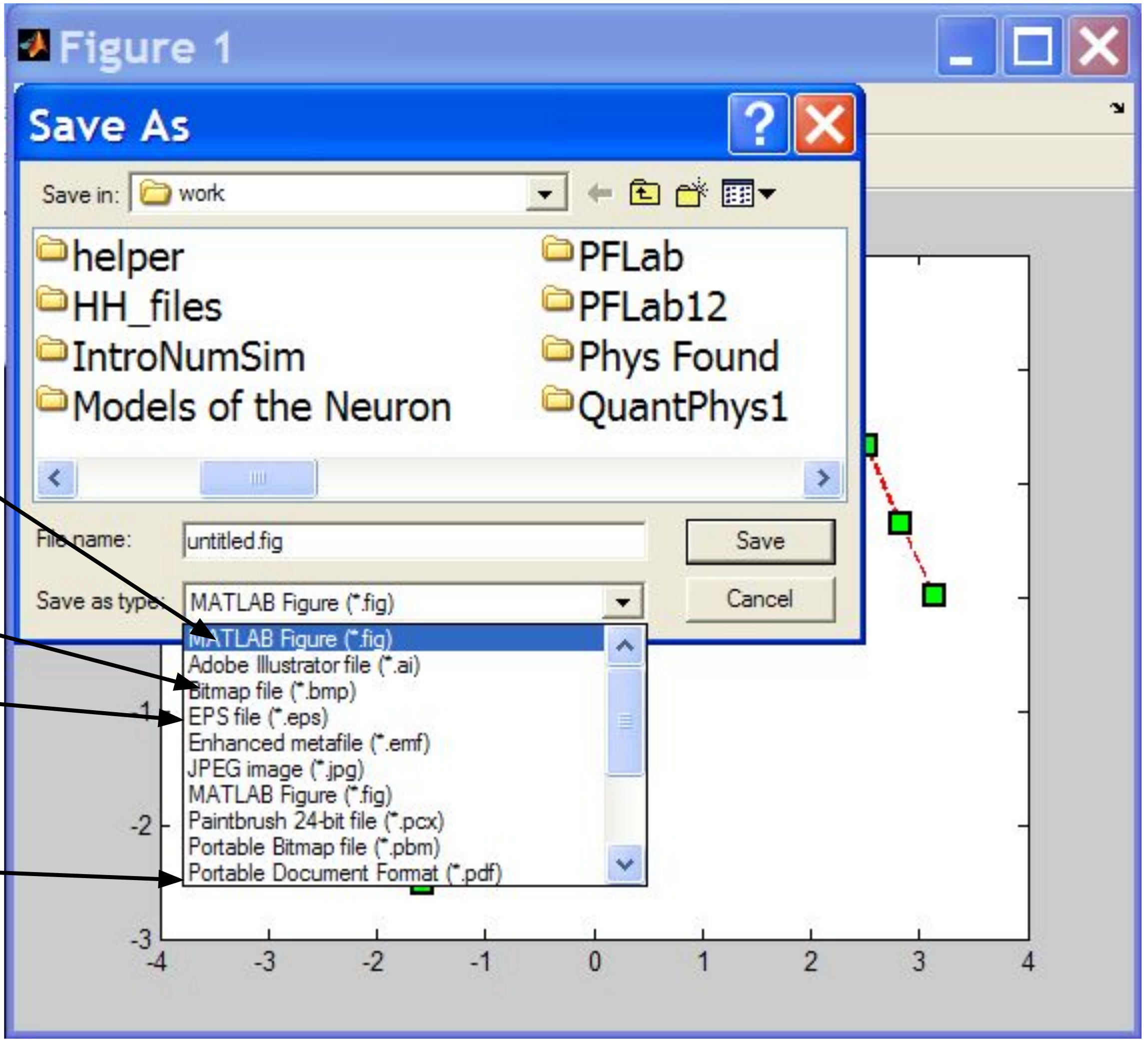
- Figures can be saved in many formats. The common ones are:

.fig preserves all information

.bmp uncompressed image

.eps high-quality scaleable format

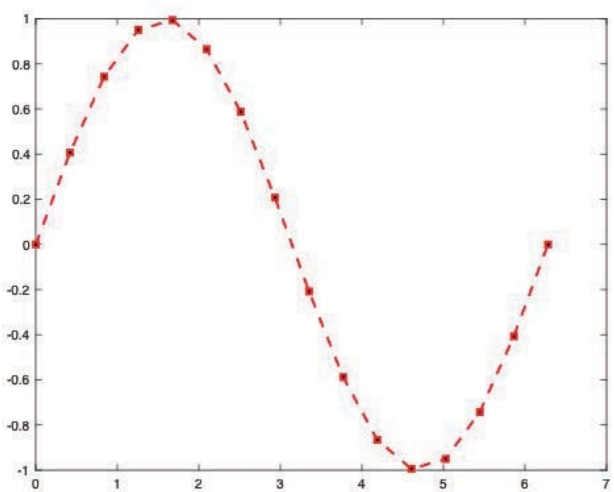
.pdf compressed image



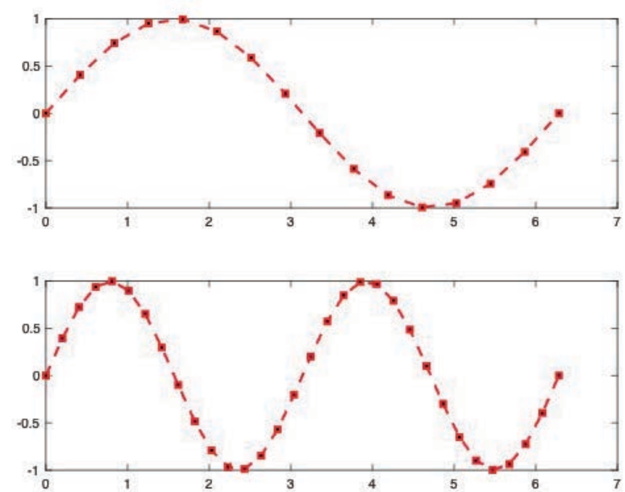
Advanced Plotting: Exercise

- Modify the plot command in your plotSin function to use **squares** as markers and a **dashed red** line of **thickness** 2 as the line. Set the marker face color to be **black** (properties are `LineWidth`, `MarkerFaceColor`)
- If there are 2 inputs, open a new figure with 2 axes, one on top of the other (not side by side), and plot both frequencies (`subplot`)

`plotSin(6)`



`plotSin(1,2)`

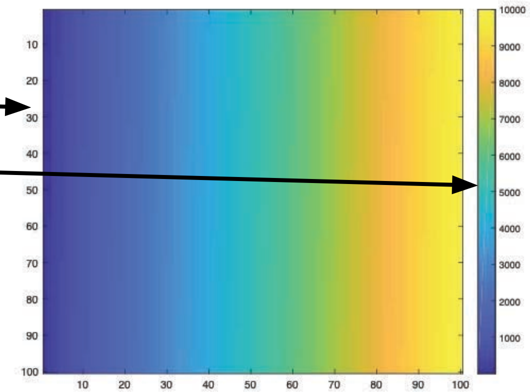


Outline

(1) Functions
(2) Flow Control
(3) Line Plots
(4) **Image/Surface Plots**
(5) Efficient Codes
(6) Debugging

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

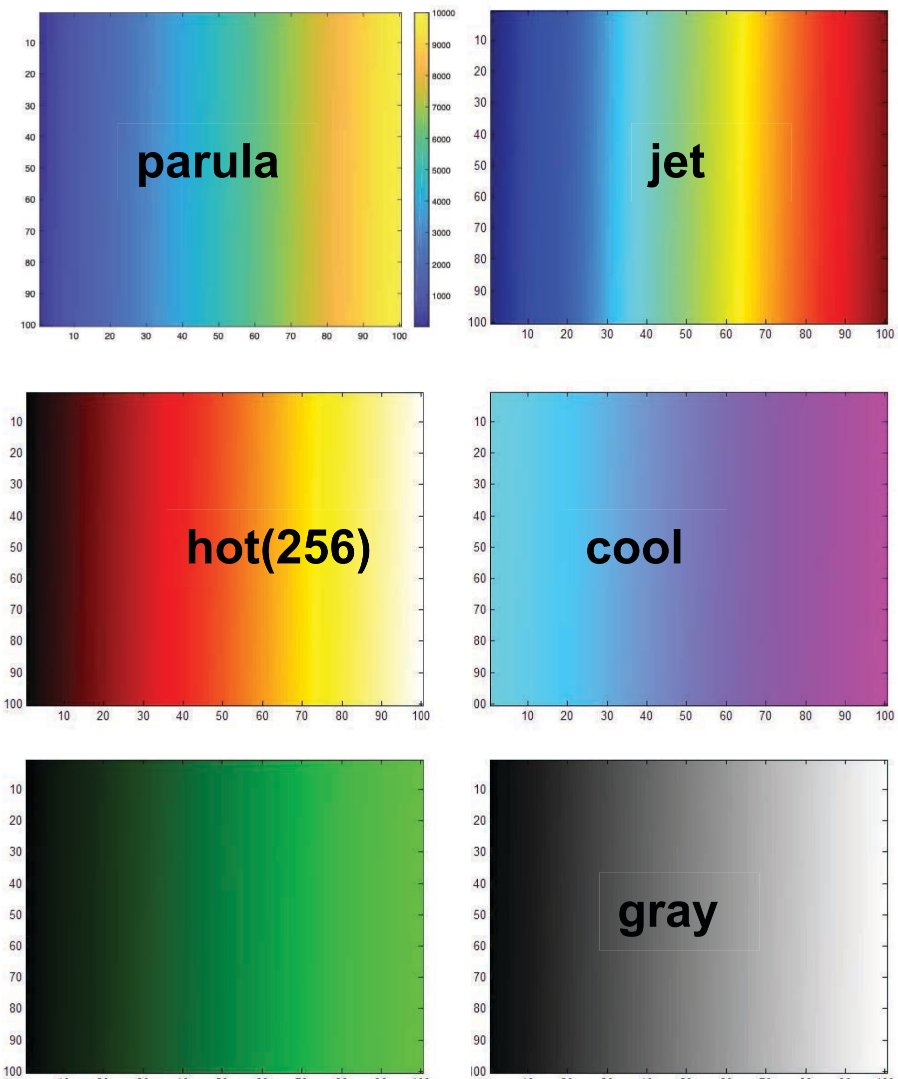
- Can set limits for the color axis (analogous to `xlim`, `ylim`)
  - » `caxis([3000 7000])`

Colormaps

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

- See `help hot` for a list

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

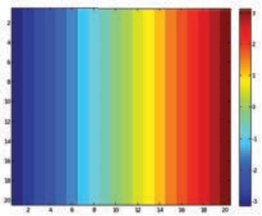


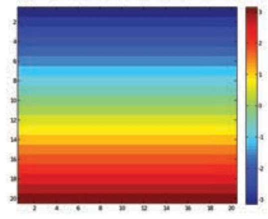
Surface Plots

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

$$\boxed{\begin{array}{l} f(x,y) = sin(x)cos(y) \\ x \in [-\pi,\pi];\ y \in [-\pi,\pi] \end{array}}$$

- **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
  - ➢ built-in function: **meshgrid**

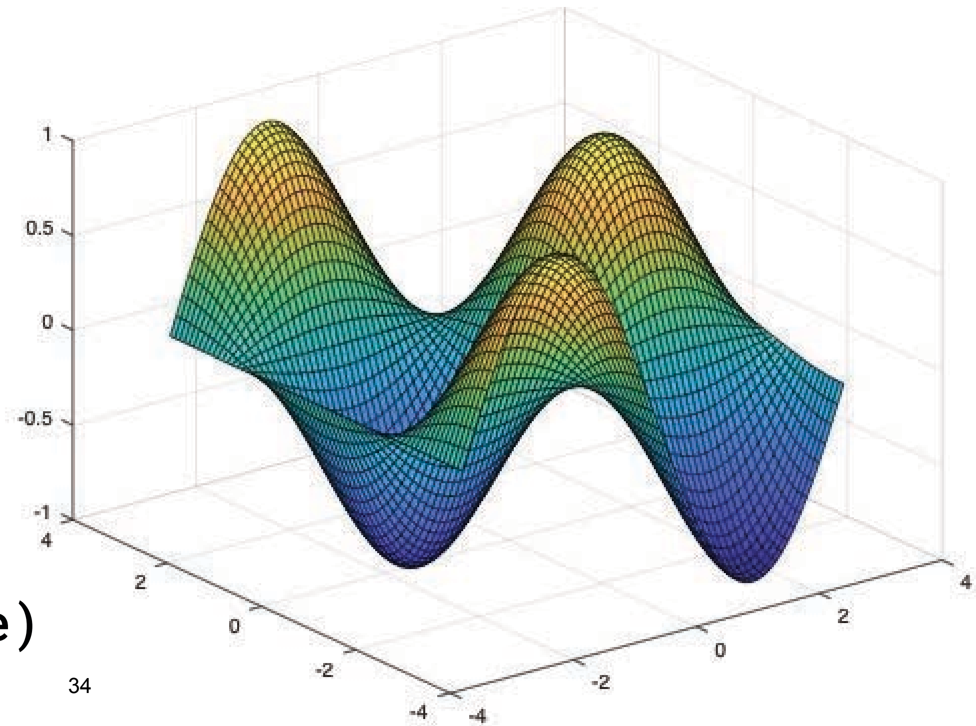




surf

- Make the x and y vectors
  - » `x=-pi:0.1:pi;`
  - » `y=-pi:0.1:pi;`
- Use meshgrid to make matrices
  - » `[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);`

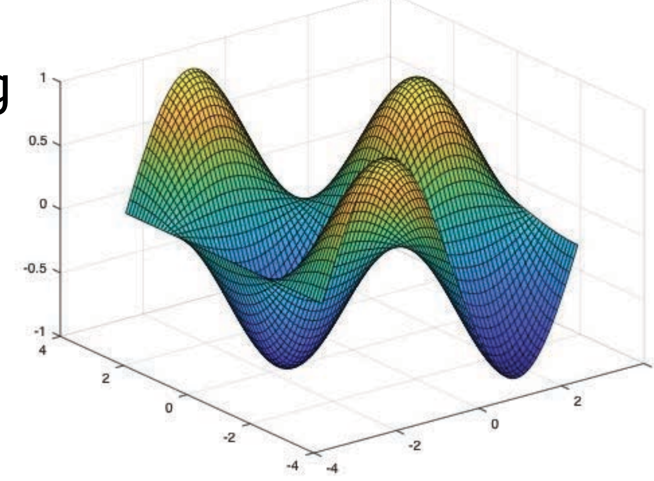
*Try typing `surf(membrane)`

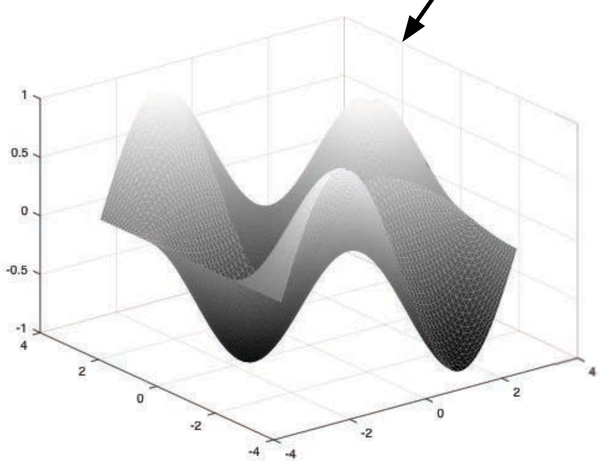


surf Options

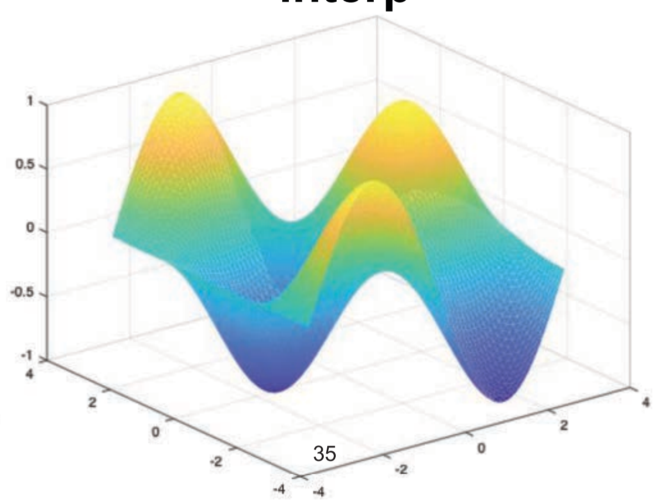
- See **help surf** for more options
- There are three types of surface shading
  - » `shading faceted`
  - » `shading flat`
  - » `shading interp`
- You can also change the colormap
  - » `colormap(gray)`

faceted

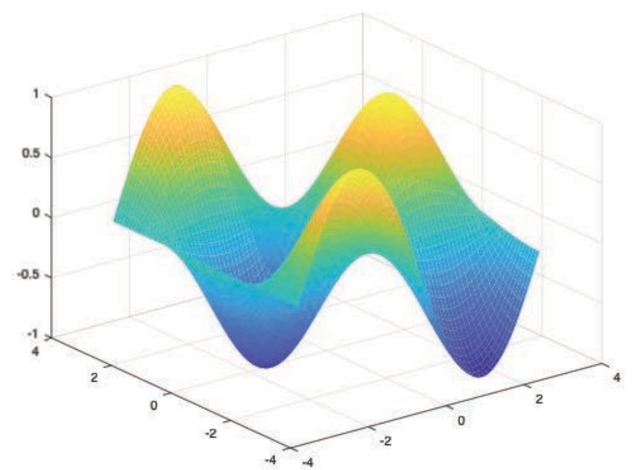




interp

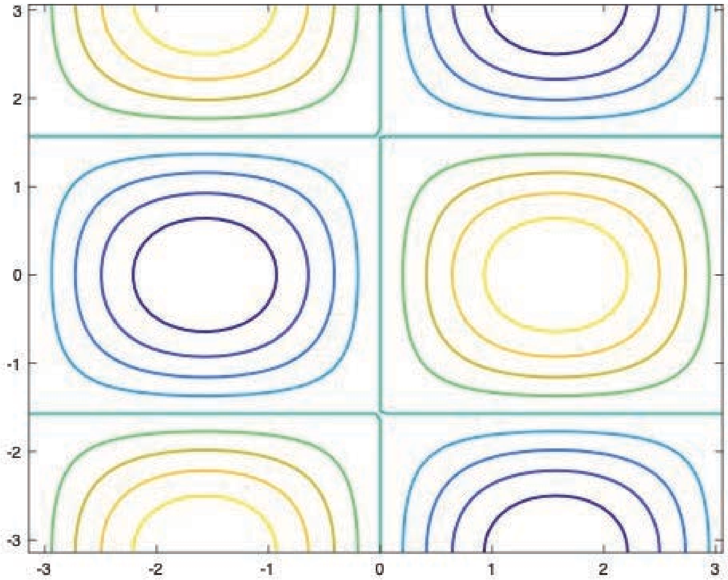


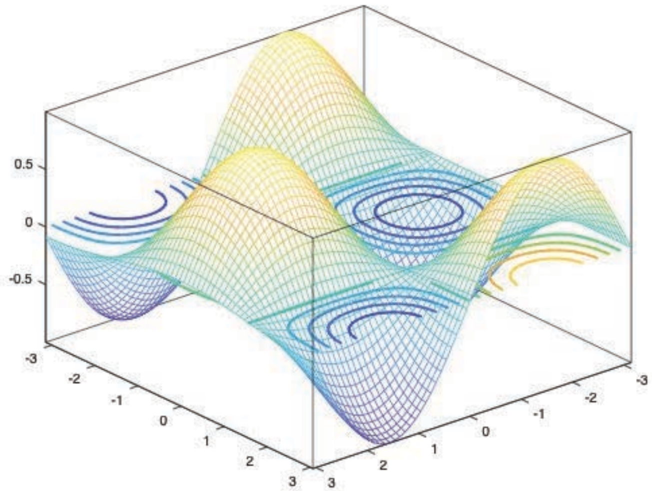
flat



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

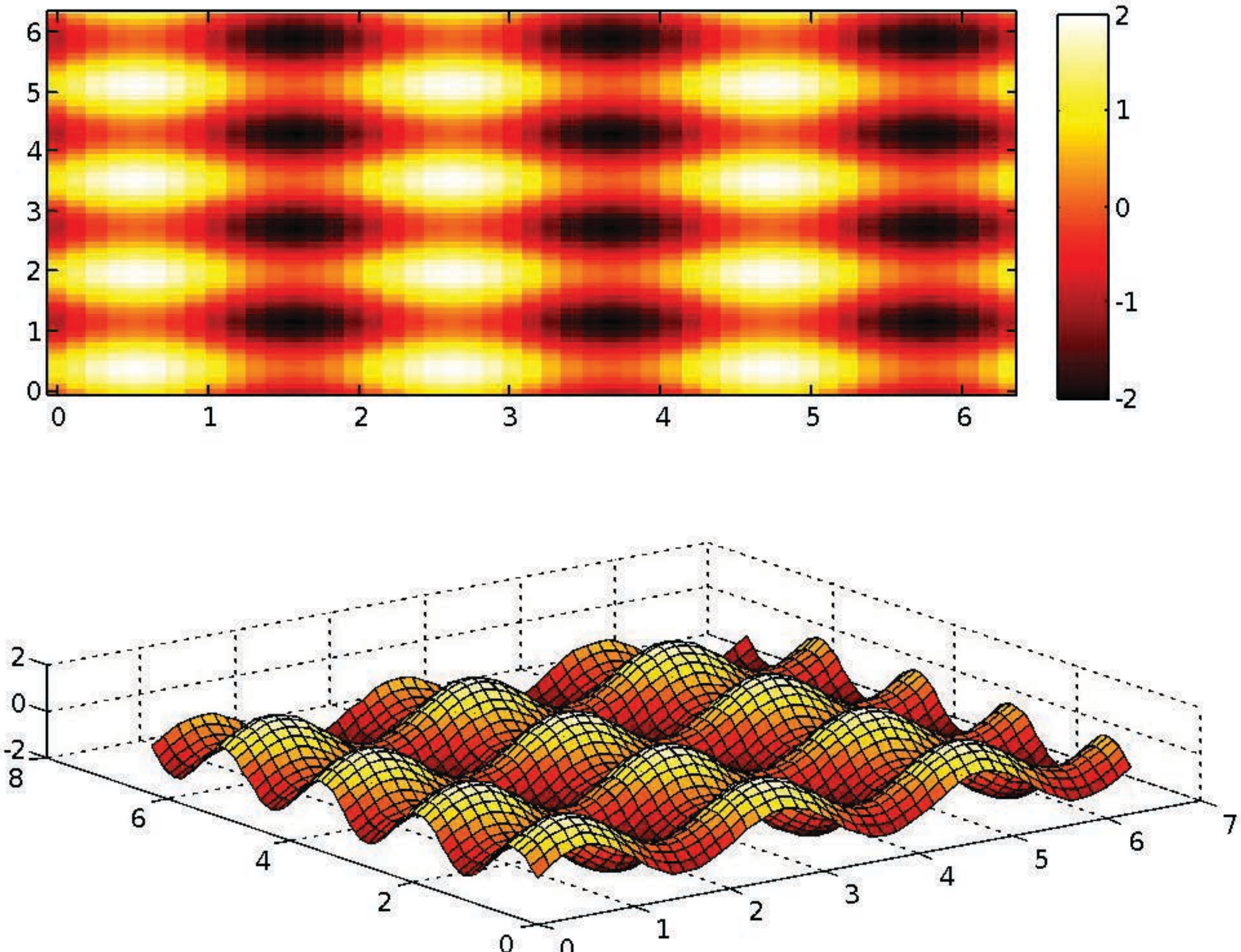
- Modify `plotSin` to do the following:
- If two inputs are given, evaluate the following function:

$$Z = \sin(f_1 x) + \sin(f_2 y)$$

- y should be just like x, but using f2. (use `meshgrid` to get the X and Y matrices)
- In the top axis of your subplot, display an image of the Z matrix. Display the colorbar and use a `hot` colormap. Set the axis to xy (`imagesc`, `colormap`, `colorbar`, `axis`)
- In the bottom axis of the subplot, plot the 3-D surface of Z (`surf`)

Exercise: 3-D Plots

`plotSin(3,4)` generates this figure



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) Functions
(2) Flow Control
(3) Line Plots
(4) Image/Surface Plots
(5) **Efficient codes**
(6) Debugging

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` contains 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

& combines the two vectors using logical **and** operator

find returns the indices of the 1's

Example: Avoiding Loops

- 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));
```

Is there a better way?!

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!
- Built-in functions will make it faster to write and execute

Vectorization

- Avoid loops
  - This is referred to as vectorization
- Vectorized code is more efficient for MATLAB
- Use indexing and matrix operations to avoid loops
- For instance, to add every two consecutive terms:

Vectorization

- Avoid loops
  - ➢ This is referred to as vectorization
- Vectorized code is more efficient for MATLAB
- Use indexing and matrix operations to avoid loops
- For instance, to add every two consecutive terms:

```
» 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

Vectorization

- Avoid loops
  - ➢ This is referred to as vectorization
- Vectorized code is more efficient for MATLAB
- Use indexing and matrix operations to avoid loops
- For instance, to add every two consecutive terms:

```
» 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. Can also do this using `conv`

Preallocation

- Avoid variables growing inside a loop
- Re-allocation of memory is time consuming
- Preallocate the required memory by initializing the array to a default value
- For example:

```
» for n=1:100
»       res = % Very complex calculation %
»       a(n) = res;
» end
```

  ➢ Variable `a` needs to be resized at every loop iteration

Preallocation

- Avoid variables growing inside a loop
- Re-allocation of memory is time consuming
- Preallocate the required memory by initializing the array to a default value
- For example:

```
» a = zeros(1, 100);
» for n=1:100
»     res = % Very complex calculation %
»     a(n) = res;
» end
```

  ➢ Variable `a` is only assigned new values. No new memory is allocated

Outline

(1) Functions
(2) Flow Control
(3) Line Plots
(4) Image/Surface Plots
(5) Efficient codes
(6) **Debugging**

Display

- When debugging functions, use **disp** to print messages
  - » `disp('starting loop')`
  - » `disp('loop is over')`
    - ➢ `disp` prints the given string to the command window

- It's also helpful to show variable values
  - » `disp(['loop iteration ' num2str(n)]);`
    - ➢ Sometimes it's easier to just remove some semicolons

Debugging

- To use the debugger, set breakpoints
  - ➢ Click on – next to line numbers in m-files
  - ➢ Each red dot that appears is a breakpoint
  - ➢ Run the program
  - ➢ The program pauses when it reaches a breakpoint
  - ➢ Use the command window to probe variables
  - ➢ Use the debugging buttons to control debugger

Performance Measures

- It can be useful to know how long your code takes to run
  - ➢ To predict how long a loop will take
  - ➢ To pinpoint inefficient code
- You can time operations using **tic**/**toc**:

```
» tic
» Mystery1;
» a=toc;
» Mystery2;
» b=toc;
```

  - ➢ tic resets the timer
  - ➢ Each toc returns the current value in seconds
  - ➢ Can have multiple tocs per tic

Performance Measures

- Example: Sparse matrices

```
» A=zeros(10000); A(1,3)=10; A(21,5)=pi;
» B=sparse(A);
» inv(A);   % what happens?
» inv(B);   % what about now?
```

- If system is sparse, can lead to large memory/time savings

```
» A=zeros(1000); A(1,3)=10; A(21,5)=pi;
» B=sparse(A);
» C=rand(1000,1);
» tic; A\C; toc;   % slow!
» tic; B\C; toc;   % much faster!
```

Performance Measures

- For more complicated programs, use the profiler
  - » **`profile on`**
    - ➢ Turns on the profiler. Follow this with function calls
  - » **`profile viewer`**
    - ➢ Displays gui with stats on how long each subfunction took

Profile Summary

Generated 04-Jan-2006 09:53:26

Number of files called: 19

Filename	File Type	Calls	Total Time	Time Plot
newplot	M-function	1	0.802 s	
gcf	M-function	1	0.460 s	
newplot/ObserveAxesNextPlot	M-subfunction	1	0.291 s	
...matlab/graphics/private/clo	M-function	1	0.251 s	
allchild	M-function	1	0.100 s	
setdiff	M-function	1	0.050 s	

End of Lecture 2

(1) **Functions**
(2) **Flow Control**
(3) **Line Plots**
(4) **Image/Surface Plots**
(5) **Efficient codes**
(6) **Debugging**

Vectorization makes coding fun!

6.057
Introduction to MATLAB

Lecture 3 : Solving Equations, Curve Fitting, and Numerical Techniques

Orhan Celiker

IAP 2019

Outline

(1) Linear Algebra
(2) Polynomials
(3) Optimization
(4) Differentiation/Integration
(5) Differential Equations

Systems of Linear Equations

MATLAB makes linear algebra fun!

- Given a system of linear equations
  - x+2y-3z=5
  - -3x-y+z=-8
  - x-y+z=0
- Construct matrices so the system is described by Ax=b

```
» A=[1 2 -3;-3 -1 1;1 -1 1];
» b=[5;-8;0];
```

- And solve with a single line of code!

```
» x=A\b;
```

  - x is a 3x1 vector containing the values of x, y, and z

- **The \ will work with square or rectangular systems.**
- Gives least squares solution for rectangular systems. Solution depends on whether the system is over or underdetermined.

Worked Example: Linear Algebra

- Solve the following systems of equations:

  - System 1:

$$x + 4y = 34$$
$$-3x + y = 2$$

```
» A=[1 4;-3 1];
» b=[34;2];
» rank(A)
» x=inv(A)*b;
» x=A\b;
```

  - System 2:

$$2x - 2y = 4$$
$$-x + y = 3$$
$$3x + 4y = 2$$

```
» A=[2 -2;-1 1;3 4];
» b=[4;3;2];
» rank(A)
```

  - rectangular matrix

```
» x=A\b;
```

  - gives least squares solution

```
» error=abs(A*x1-b)
```

More Linear Algebra

- Given a matrix
  - » `mat=[1 2 -3;-3 -1 1;1 -1 1];`
- Calculate the rank of a matrix
  - » `r=rank(mat);`
    - ➢ the number of linearly independent rows or columns
- Calculate the determinant
  - » `d=det(mat);`
    - ➢ mat must be square; matrix invertible if det nonzero
- Get the matrix inverse
  - » `E=inv(mat);`
    - ➢ if an equation is of the form A*x=b with A a square matrix, x=A\b is (mostly) the same as x=inv(A)*b
- Get the condition number
  - » `c=cond(mat);` (or its reciprocal: `c = rcond(mat);`)
    - ➢ if condition number is large, when solving A*x=b, small errors in b can lead to large errors in x (optimal c==1)

Matrix Decompositions

- MATLAB has many built-in matrix decomposition methods

- The most common ones are
  - » `[V,D]=eig(X)`
    - ➢ Eigenvalue decomposition
  - » `[U,S,V]=svd(X)`
    - ➢ Singular value decomposition
  - » `[Q,R]=qr(X)`
    - ➢ QR decomposition
  - » `[L,U]=lu(X)`
    - ➢ LU decomposition
  - » `R=chol(X)`
    - ➢ Cholesky decomposition (R must be positive definite)

Exercise: Fitting Polynomials

- Find the best second-order polynomial that fits the points: (-1,0), (0,-1), (2,3).

$$a(-1)^2 + b(-1) + c = 0$$
$$a(0)^2 + b(0) + c = -1$$
$$a(2)^2 + b(2) + c = 3$$

Outline

(1) Linear Algebra

(2) Polynomials

(3) Optimization

(4) Differentiation/Integration

(5) Differential Equations

Polynomials

- Many functions can be well described by a high-order polynomial

- MATLAB represents a polynomials by a vector of coefficients
  - ➢ if vector P describes a polynomial

$$ax^3+bx^2+cx+d$$

P(1) P(2) P(3) P(4)

- P=[1 0 -2] represents the polynomial x^2-2

- P=[2 0 0 0] represents the polynomial $2x^3$

Polynomial Operations

- P is a vector of length N+1 describing an N-th order polynomial
- To get the roots of a polynomial
  - » `r=roots(P)`
    - ➢ r is a vector of length N

- Can also get the polynomial from the roots
  - » `P=poly(r)`
    - ➢ r is a vector length N

- To evaluate a polynomial at a point
  - » `y0=polyval(P,x0)`
    - ➢ x0 is a single value; y0 is a single value

- To evaluate a polynomial at many points
  - » `y=polyval(P,x)`
    - ➢ x is a vector; y is a vector of the same size

Polynomial Fitting

- MATLAB makes it very easy to fit polynomials to data

- Given data vectors X=[-1 0 2] and Y=[0 -1 3]
  - **» p2=polyfit(X,Y,2);**
    - ➢ finds the best (least-squares sense) second-order polynomial that fits the points (-1,0),(0,-1), and (2,3)
    - ➢ see **help polyfit** for more information
  - **» plot(X,Y,'o', 'MarkerSize', 10);**
  - **» hold on;**
  - **» x = -3:.01:3;**
  - **» plot(x,polyval(p2,x), 'r--');**

Exercise: Polynomial Fitting

- Evaluate $y = x^2$ for x=-4:0.1:4.

- Add random noise to these samples. Use **randn**. Plot the noisy signal with **.** markers

- Fit a 2nd degree polynomial to the noisy data

- Plot the fitted polynomial on the same plot, using the same x values and a red line

Outline

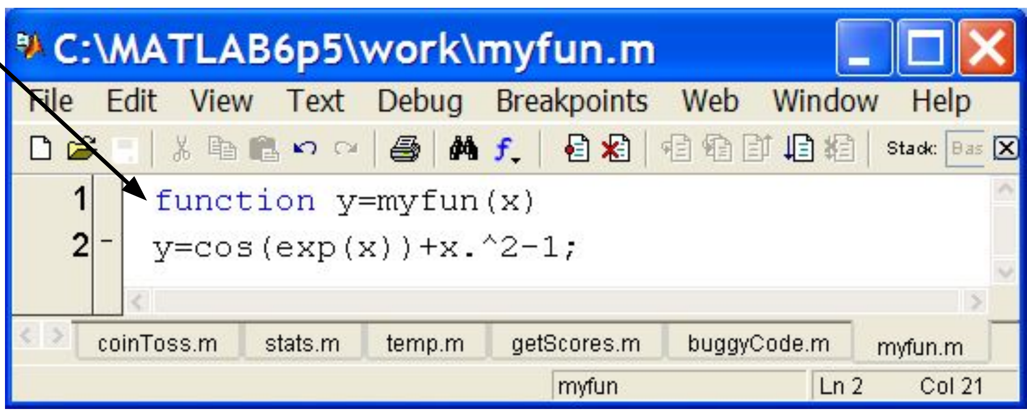
(1) Linear Algebra
(2) Polynomials
(3) Optimization
(4) Differentiation/Integration
(5) Differential Equations

Nonlinear Root Finding

- Many real-world problems require us to solve f(x)=0
- Can use **fzero** to calculate roots for *any* arbitrary function

- **fzero** needs a function passed to it.
- We will see this more and more as we delve into solving equations.

- Make a separate function file
  - » `x=fzero('myfun',1)`
  - » `x=fzero(@myfun,1)`
    - ➢ 1 specifies a point close to where you think the root is



Minimizing a Function

- **fminbnd**: minimizing a function over a bounded interval
  - » `x=fminbnd('myfun',-1,2);`
    - ➢ myfun takes a scalar input and returns a scalar output
    - ➢ myfun(x) will be the minimum of myfun for $-1 \leq x \leq 2$
- **fminsearch**: unconstrained interval
  - » `x=fminsearch('myfun',.5)`
    - ➢ finds the local minimum of myfun starting at x=0.5
- Maximize g(x) by minimizing f(x)=-g(x)
- Solutions may be local!

Anonymous Functions

- You do not have to make a separate function file
  - » `x=fzero(@myfun,1)`
    - ➢ What if myfun is really simple?

- Instead, you can make an anonymous function
  - » `x=fzero(@(x)(cos(exp(x))+x.^2-1), 1 );`

    input function to evaluate

  - » `x=fminbnd(@(x) (cos(exp(x))+x.^2-1),-1,2);`

- Can also store the function handle
  - » `func=@(x) (cos(exp(x))+x.^2-1);`
  - » `func(1:10);`

Optimization Toolbox

- If you are familiar with optimization methods, use the optimization toolbox
- Useful for larger, more structured optimization problems
- Sample functions (see **`help`** for more info)
  - » **`linprog`**
    - ➢ linear programming using interior point methods
  - » **`quadprog`**
    - ➢ quadratic programming solver
  - » **`fmincon`**
    - ➢ constrained nonlinear optimization

Exercise: Min-Finding

- Find the minimum of the function $f(x) = \cos(4x)\sin(10x)e^{-|x|}$ over the range −π to π. Use **`fminbnd`**.
- Plot the function on this range to check that this is the minimum.

Digression: Numerical Issues

- Many techniques in this lecture use floating point numbers
- **This is an approximation!**

- Examples:
  - » **`sin(pi) = ?`**
  - » **`sin(2 * pi) = ?`**
  - » **`sin(10e16 * pi) = ?`**
    - ➢ Both sin and pi are approximations!

  - » **`A = (10e13)*ones(10) + rand(10)`**
    - ➢ A is nearly singular, poorly conditioned (see **`cond(A)`**)
  - » **`inv(A)*A = ?`**

A Word of Caution

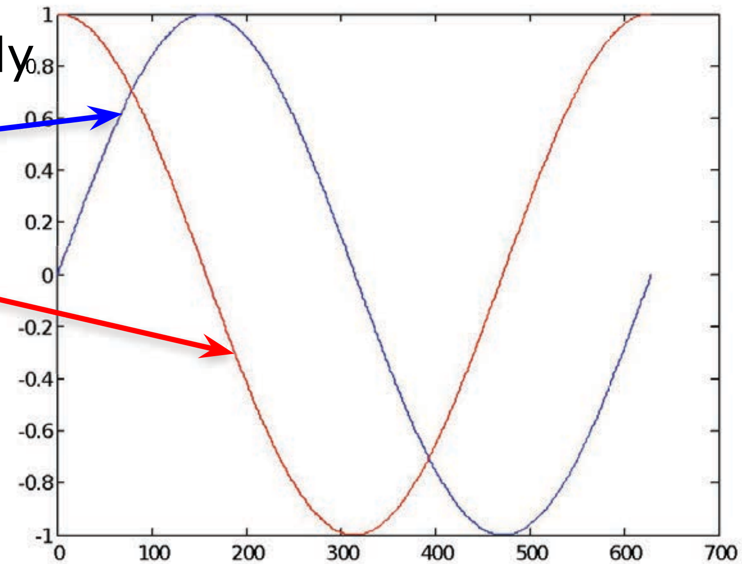
- MATLAB knows no fear!
- Give it a function, it optimizes / differentiates / integrates
  - ➢ That's great! It's so powerful!
- Numerical techniques are powerful **but** not magic
- **Beware of overtrusting the solution!**
  - ➢ You will get an answer, but it may not be what you want
- Analytical forms may give more intuition
  - ➢ Symbolic Math Toolbox

Outline

(1) Linear Algebra
(2) Polynomials
(3) Optimization
(4) Differentiation/Integration
(5) Differential Equations

Numerical Differentiation

- MATLAB can 'differentiate' numerically
  » `x=0:0.01:2*pi;`
  » `y=sin(x);`
  » `dydx=diff(y)./diff(x);`
    ➢ diff computes the first difference



- Can also operate on matrices
  » `mat=[1 3 5;4 8 6];`
  » `dm=diff(mat,1,2)`
    ➢ first difference of mat along the 2nd dimension, dm=[2 2;4 -2]
    ➢ The opposite of **`diff`** is the cumulative sum **`cumsum`**

- 2D gradient
  » `[dx,dy]=gradient(mat);`

- Higher derivatives / complicated problems: Fit spline (see **help**)

Numerical Integration

- MATLAB contains common integration methods

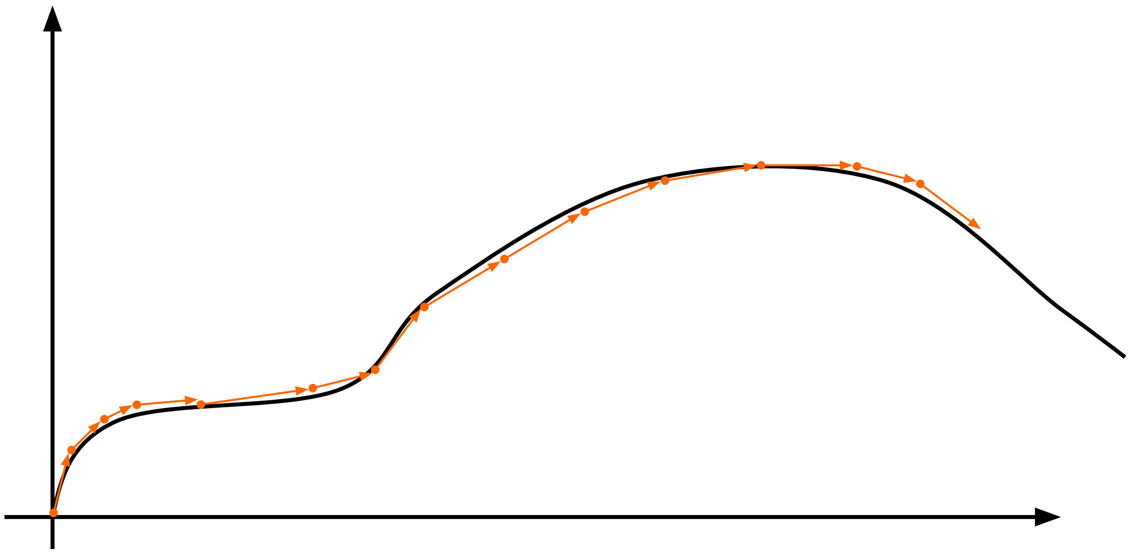
- Adaptive Simpson's quadrature (input is a function)
  - » `q=quad('myFun',0,10)`
    - ➢ q is the integral of the function `myFun` from 0 to 10
  - » `q2=quad(@(x) sin(x).*x,0,pi)`
    - ➢ q2 is the integral of `sin(x).*x` from 0 to pi
- Trapezoidal rule (input is a vector)
  - » `x=0:0.01:pi;`
  - » `z=trapz(x,sin(x))`
    - ➢ z is the integral of sin(x) from 0 to pi
  - » `z2=trapz(x,sqrt(exp(x))./x)`
    - ➢ z2 is the integral of $\sqrt{e^x}/x$ from 0 to pi

Outline

(1) Linear Algebra
(2) Polynomials
(3) Optimization
(4) Differentiation/Integration
(5) Differential Equations

ODE Solvers: Method

- Given a differential equation, the solution can be found by integration:



  - ➢ Evaluate the derivative at a point and approximate by straight line
  - ➢ Errors accumulate!
  - ➢ Variable timestep can decrease the number of iterations

ODE Solvers: MATLAB

- MATLAB contains implementations of common ODE solvers

- Using the correct ODE solver can save you lots of time and give more accurate results
  - » `ode23`
    - ➢ Low-order solver. Use when integrating over small intervals or when accuracy is less important than speed
  - » `ode45`
    - ➢ High order (Runge-Kutta) solver. High accuracy and reasonable speed. Most commonly used.
  - » `ode15s`
    - ➢ Stiff ODE solver (Gear's algorithm), use when the diff eq's have time constants that vary by orders of magnitude

ODE Solvers: Standard Syntax

- To use standard options and variable time step

```
» [t,y]=ode45('myODE',[0,10],[1;0])
```

ODE integrator:
23, 45, 15s

ODE function

Time range

Initial conditions

- Inputs:
  - ➢ ODE function name (or anonymous function). This function should take inputs (t,y), and returns dy/dt
  - ➢ Time interval: 2-element vector with initial and final time
  - ➢ Initial conditions: column vector with an initial condition for each ODE. This is the first input to the ODE function
  - ➢ Make sure all inputs are in the same (variable) order
- Outputs:
  - ➢ t contains the time points
  - ➢ y contains the corresponding values of the variables

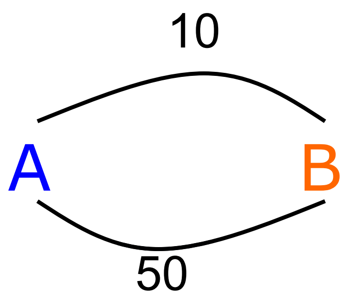
ODE Function

- The ODE function must return the value of the derivative at a given time and function value

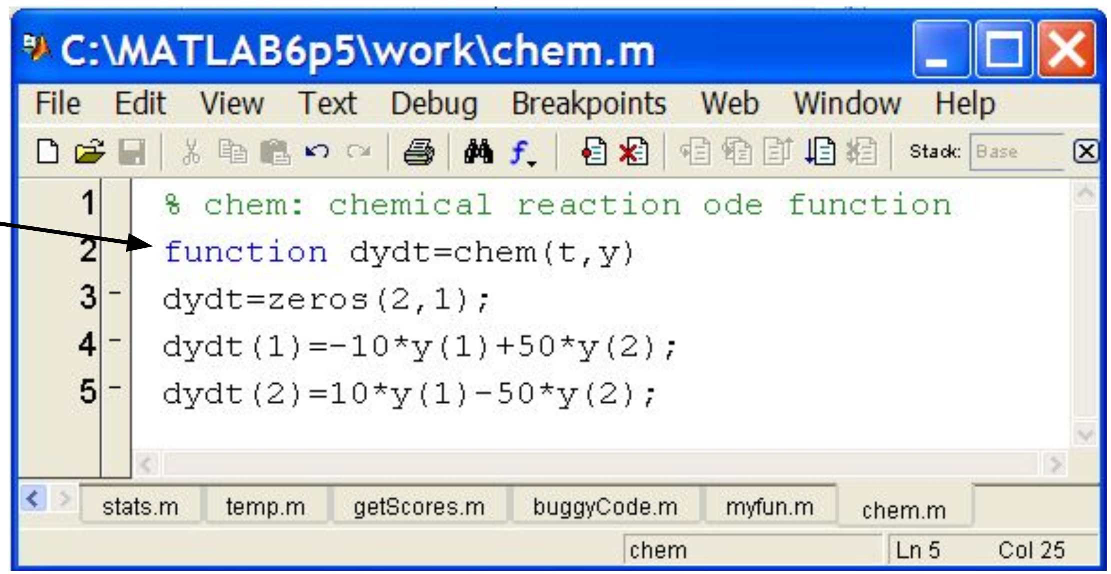
- Example: chemical reaction
  - ➢ Two equations

$$\frac{dA}{dt} = -10A + 50B$$

$$\frac{dB}{dt} = 10A - 50B$$



  - ➢ ODE file:
    - y has [A;B]
    - dydt has [dA/dt;dB/dt]



ODE Function: viewing results

- To solve and plot the ODEs on the previous slide:

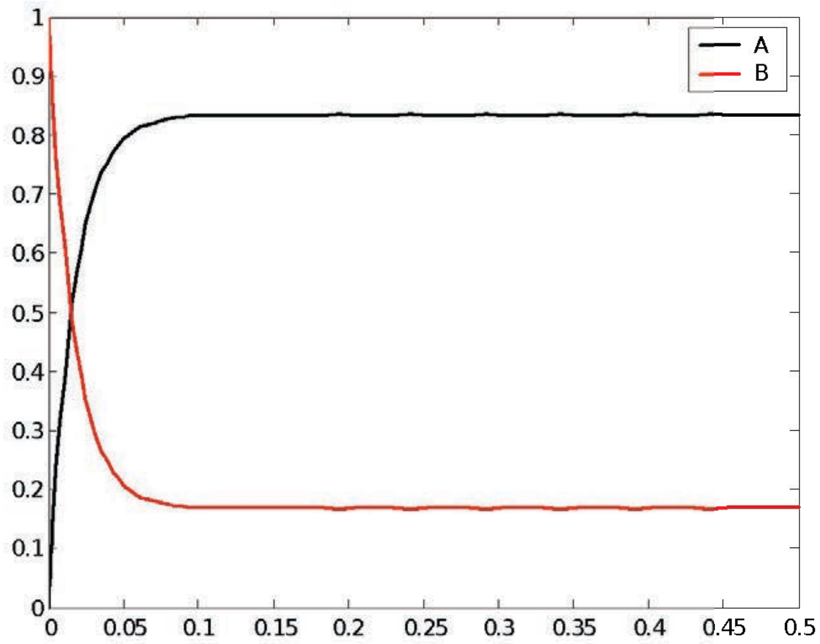
```
» [t,y]=ode45('chem',[0 0.5],[0 1]);
```

  - ➢ assumes that only chemical B exists initially

```
» plot(t,y(:,1),'k','LineWidth',1.5);
» hold on;
» plot(t,y(:,2),'r','LineWidth',1.5);
» legend('A','B');
» xlabel('Time (s)');
» ylabel('Amount of chemical (g)');
» title('Chem reaction');
```

ODE Function: viewing results

- The code on the previous slide produces this figure



Higher Order Equations

- Must make into a system of first-order equations to use ODE solvers
- Nonlinear is OK!
- Pendulum example:

$$\ddot{\theta} + \frac{g}{L} sin(\theta) = 0$$

$$\ddot{\theta} = -\frac{g}{L} sin(\theta)$$

$$let \quad \dot{\theta} = \gamma$$

$$\dot{\gamma} = -\frac{g}{L} sin(\theta)$$

$$\boxed{\vec{x} = \begin{bmatrix} \theta \\ \gamma \end{bmatrix} \qquad \frac{d\vec{x}}{dt} = \begin{bmatrix} \dot{\theta} \\ \dot{\gamma} \end{bmatrix}}$$

C:\MATLAB6p5\work\pendulum.m

File Edit View Text Debug Breakpoints Web Window Help

```
% pendulum
function dxdt = pendulum(t,x)
L = 1;
theta = x(1);
gamma = x(2);

dtheta = gamma;
dgamma = -(9.8/L)*sin(theta);

dxdt = zeros(2,1);

dxdt(1)=dtheta;
dxdt(2)=dgamma;
```

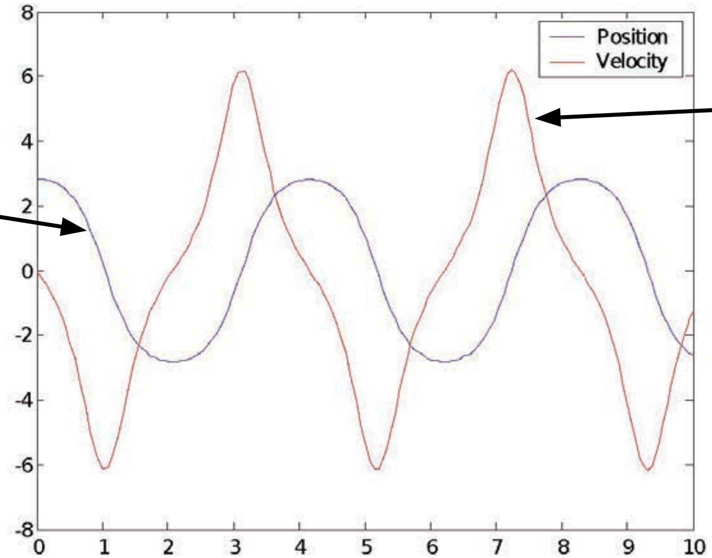
temp.m | getScores.m | buggyCode.m | myfun.m | chem.m | pendulum.m

pendulum Ln 13 Col 5

Plotting the Output

- We can solve for the position and velocity of the pendulum:
  - » `[t,x]=ode45('pendulum',[0 10],[0.9*pi 0]);`
    - ➢ assume pendulum is almost horizontal
  - » `plot(t,x(:,1));`
  - » `hold on;`
  - » `plot(t,x(:,2),'r');`
  - » `legend('Position','Velocity');`

Position in terms of angle (rad)



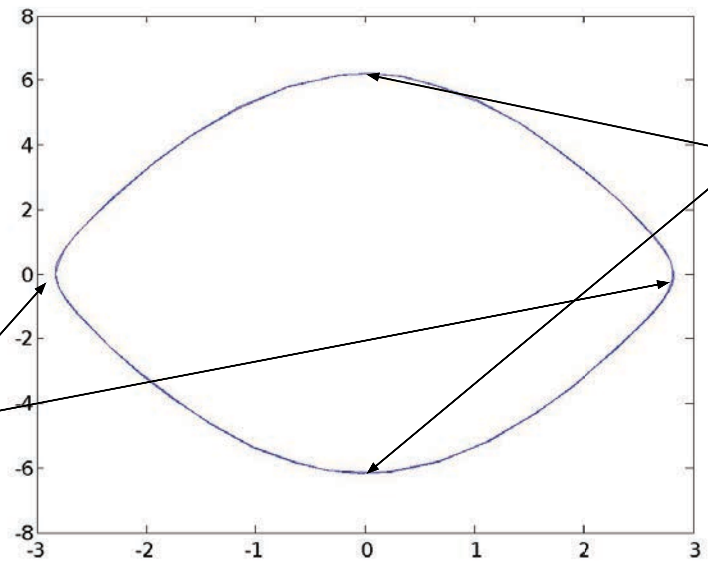
Velocity (m/s)

Plotting the Output

- Or we can plot in the phase plane:

```
» plot(x(:,1),x(:,2));
» xlabel('Position');
» yLabel('Velocity');
```

- The phase plane is just a plot of one variable versus the other:



ODE Solvers: Custom Options

- MATLAB's ODE solvers use a variable timestep
- Sometimes a fixed timestep is desirable
  - » `[t,y]=ode45('chem',[0:0.001:0.5],[0 1]);`
    - ➢ Specify timestep by giving a vector of (increasing) times
    - ➢ The function value will be returned at the specified points

- You can customize the error tolerances using odeset
  - » `options=odeset('RelTol',1e-6,'AbsTol',1e-10);`
  - » `[t,y]=ode45('chem',[0 0.5],[0 1],options);`
    - ➢ This guarantees that the error at each step is less than **`RelTol`** times the value at that step, and less than **`AbsTol`**
    - ➢ Decreasing error tolerance can considerably slow the solver
    - ➢ See doc odeset for a list of options you can customize

Exercise: ODE

- Use `ode45` to solve for $y(t)$ on the range t=[0 10], with initial condition $y(0)=10$ and $dy/dt=-t\,y/10$
- Plot the result.

Exercise: ODE

- Use `ode45` to solve for $y(t)$ on the range t=[0 10], with initial condition $y(0)=10$ and $dy/dt = -t\,y/10$
- Plot the result.

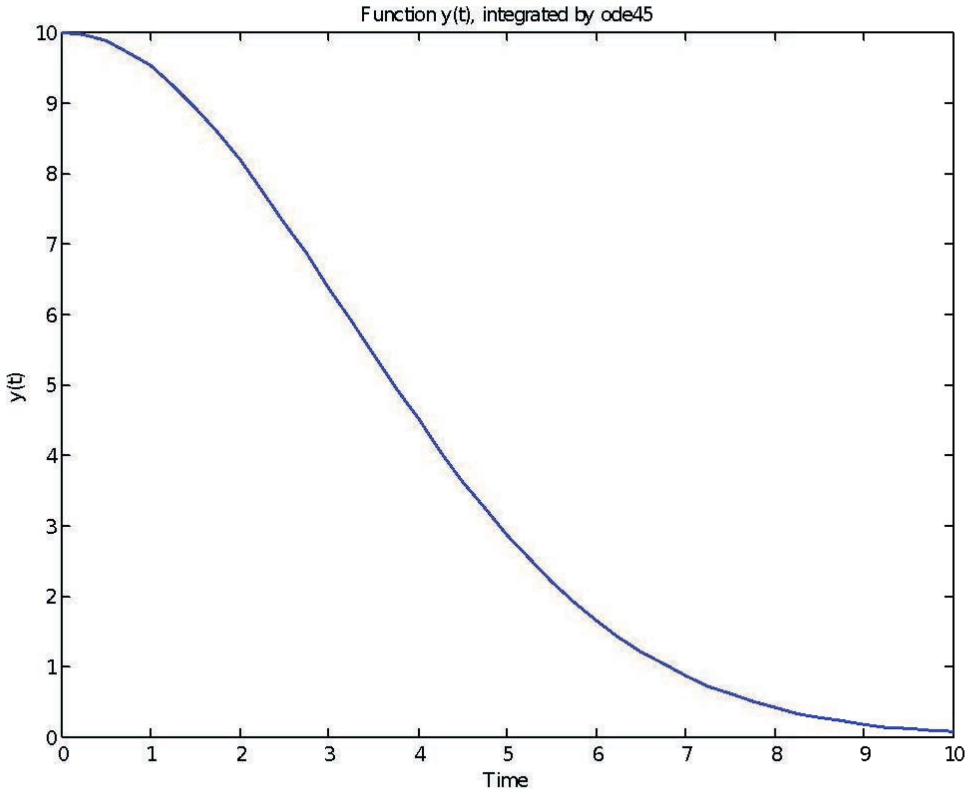
- Make the following function
  - » `function dydt=odefun(t,y)`
  - » `dydt=-t*y/10;`
- Integrate the ODE function and plot the result
  - » `[t,y]=ode45('odefun',[0 10],10);`

- Alternatively, use an anonymous function
  - » `[t,y]=ode45(@(t,y) -t*y/10,[0 10],10);`

- Plot the result
  - » `plot(t,y);xlabel('Time');ylabel('y(t)');`

Exercise: ODE

- The integrated function looks like this:



6.057
Introduction to programming in MATLAB

Lecture 4: Advanced Methods

Orhan Celiker

IAP 2019

Note about functions in files

- Whenever possible, write your functions in their own files

  - ➢ e.g. myfun should be in a file by itself, and the file should be called myfun.m*

  - ➢ If you include more than one function per file, only the first function is accessible in other scripts

  - ➢ More info here: https://www.mathworks.com/help/matlab/matlab_prog/create-functions-in-files.html

* If filename and function name differs, MATLAB recognizes your function by its filename**, not the function name

** yes, this is very confusing :(

Outline

(1) **Probability and Statistics**
(2) Data Structures
(3) Images
(4) File I/O

Statistics

- Whenever analyzing data, you have to compute statistics

```
» scores = 100*rand(1,100);   % random data
```

- Built-in functions
    - ➢ mean, median, mode

- To group data into a histogram

```
» hist(scores,5:10:95);
```

    - ➢ makes a histogram with bins centered at 5, 15, 25…95

```
» hist(scores,20);
```

    - ➢ makes a histogram with 20 bins

```
» N=histc(scores,0:10:100);
```

    - ➢ returns the number of occurrences between the specified bin *edges* 0 to <10, 10 to <20…90 to <100. you can plot these manually:

```
» bar(0:10:100,N,'r')
```

Random Numbers

- Many probabilistic processes rely on random numbers

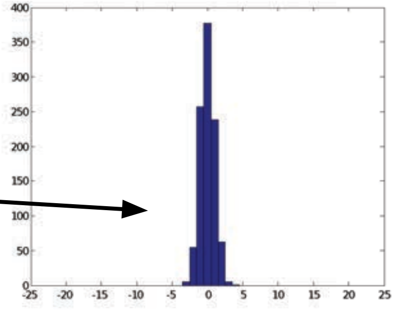
- MATLAB contains the common distributions built in
  - » **`rand`**
    - ➢ draws from the uniform distribution from 0 to 1
  - » **`randn`**
    - ➢ draws from the standard normal distribution (Gaussian)
  - » **`random`**
    - ➢ can give random numbers from many more distributions
    - ➢ see **help random**

- You can also seed the random number generators

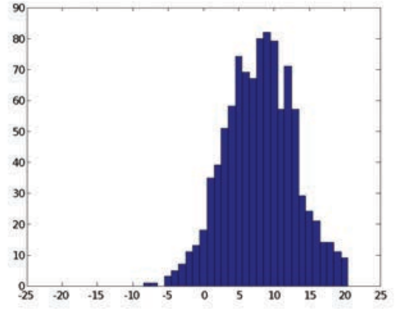
```
» rand('state',0); rand(1); rand(1);
  rand('state',0); rand(1);   % same random number
```

Changing Mean and Variance

- We can alter the given distributions
  - » `y=rand(1,100)*10+5;`
    - ➢ gives 100 uniformly distributed numbers between 5 and 15
  - » `y=floor(rand(1,100)*10+6);`
    - ➢ gives 100 uniformly distributed integers between 6 and 15. floor or ceil is better to use here than round
    - ➢ you can also use `randi([6,15],1,100)`



  - » `y=randn(1,1000)`
  - » `y2=y*5+8`
    - ➢ increases std to 5 and makes the mean 8



Exercise: Probability

- We will simulate Brownian motion in 1 dimension. Call the script 'brwn'
- Make a 10,001 element vector of zeros
- Write a loop to keep track of the particle's position at each time
- Assume middle of the vector is position 0. To get the new position, pick a random number, and if it's <0.5, go left; if it's >0.5, go right. Keep count of how many times each position is visited.
- Plot a 50 bin histogram of the positions.

Outline

(1) Probability and Statistics
(2) Data Structures
(3) Images
(4) File I/O

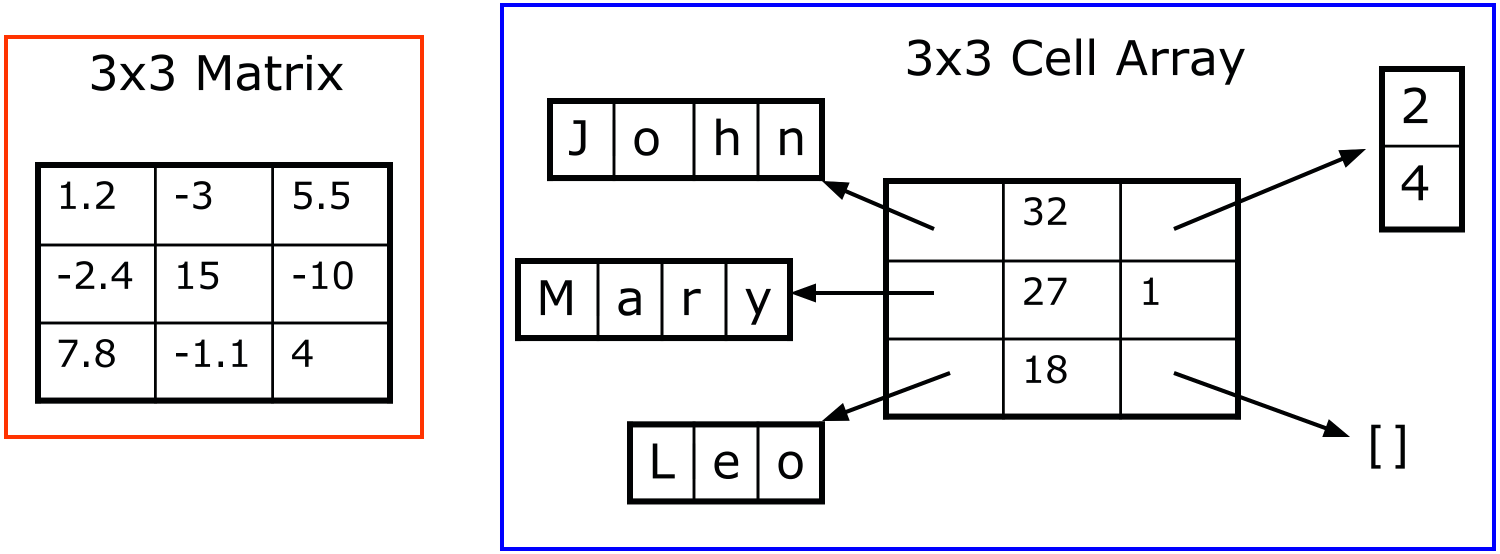
Advanced Data Structures

- We have used 2D matrices
  - ➢ Can have n-dimensions (e.g., RGB images)
  - ➢ Every element must be the same type (ex. integers, doubles, characters…)
  - ➢ Matrices are space-efficient and convenient for calculation
  - ➢ Large matrices with many zeros can be made sparse
    - – More on this later this lecture

- Sometimes, more complex data structures are more appropriate
  - ➢ **Cell array**: it's like an array, but elements don't have to be the same type
  - ➢ **Structs**: can bundle variable names and values into one structure
    - – Like object oriented programming in MATLAB

Cells: organization

- A cell is just like a matrix, but each field can contain anything (even other matrices):



- One cell can contain people's names, ages, and the ages of their children
- To do the same with matrices, you would need 3 variables and padding

Cells: initialization

- To initialize a cell, specify the size
  - » **a=cell(3,10);**
    - ➢ a will be a cell with 3 rows and 10 columns
- or do it manually, with curly braces {}
  - » **c={'hello world',[1 5 6 2],rand(3,2)};**
    - ➢ c is a cell with 1 row and 3 columns
- Each element of a cell can be anything
- To access a cell element, use curly braces {}
  - » **a{1,1}=[1 3 4 -10];**
  - » **a{2,1}='hello world 2';**
  - » **a{1,2}=c{3};**

Exercise: Cells

- Write a script called `sentGen`
- Make a 2x3 cell, and put three **names** into the first row, and **adjectives** into the second row
- Pick two random integers (values 1 to 3)
- Display a sentence of the form '[name] is [adjective].'
- Run the script a few times

Structs

- Structs allow you to name and bundle relevant variables
  - ➢ Like C-structs, which are containers with fields
- To initialize an empty struct:

```
» s=struct;
```

  - ➢ size(s) will be 1x1
  - ➢ initialization is optional but is recommended when using large structs
- To add fields

```
» s.name = 'Leo';
» s.age = 18;
» s.childAge = [];
```

  - ➢ Fields can be anything: matrix, cell, even struct
  - ➢ Useful for keeping variables together
- For more information, see **help struct**

Struct Arrays

- To initialize a struct array, give field, values pairs

```
» ppl=struct('name',{'John','Mary','Leo'},...
  'age',{32,27,18},'childAge',{[2;4],1,[]});
```

  - ➢ size(ppl)=1x3
  - ➢ every cell must have the same size

```
» person=ppl(2);
```

  - ➢ person is now a struct with fields name, age, children
  - ➢ the values of the fields are the second index into each cell

```
» ppl(3)=s;
```

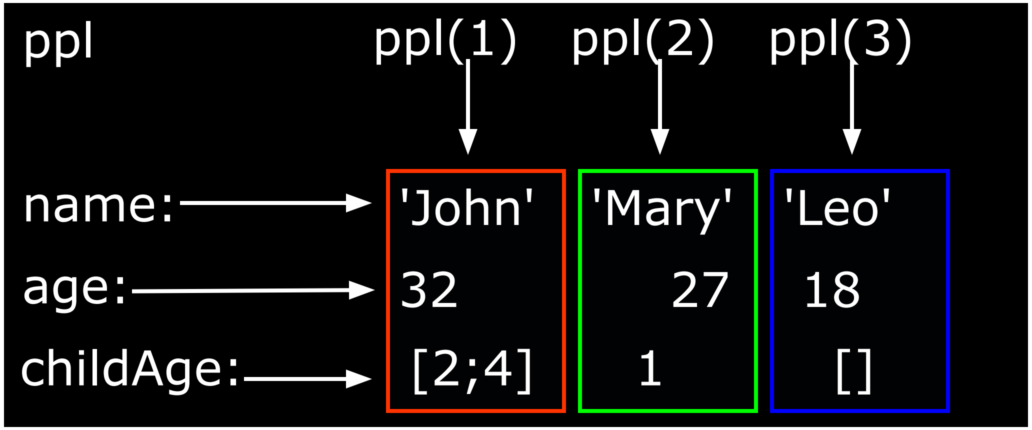
  - ➢ adds struct (fields must match)

```
» person.name
```

  - ➢ returns 'Mary'

```
» ppl(1).age
```

  - ➢ returns 32



Structs: Access

- To access 1x1 struct fields, give name of the field
  - » `stu=s.name;`
  - » `a=s.age;`
    - ➢ 1x1 structs are useful when passing many variables to a function. Put them all in a struct, and pass the struct

- To access nx1 struct arrays, use indices
  - » `person=ppl(2);`
    - ➢ person is a struct with name, age, and child age
  - » `personName=ppl(2).name;`
    - ➢ personName is 'Mary'
  - » `a=[ppl.age];`
    - ➢ a is a 1x3 vector of the ages; this may not always work, the vectors must be able to be concatenated

Exercise: Structs

- Modify the script `sentGen`
- Create a struct array with a field “name” and a field “adj” containing the values from the previous cell array
- Do not create it from scratch! Use the previously defined cell array!
- Modify the display command to use the struct array
- Run the script a few times

Outline

(1) Probability and Statistics
(2) Data Structures
(3) Images
(4) File I/O

Handles

- Manipulate graphics objects using 'handles'
  » `L=plot(1:10,rand(1,10));`
    ➢ gets the handle for the plotted line
  » `A=gca;`
    ➢ gets the handle for the current axis
  » `F=gcf;`
    ➢ gets the handle for the current figure
- To see the current property values, use get
  » `get(L);`
  » `yVals=get(L,'YData');`
- To change the properties, use set
  » `set(A,'FontName','Arial','XScale','log');`
  » `set(L,'LineWidth',1.5,'Marker','*');`
- Everything you see in a figure is completely customizable through handles

Reading/Writing Images

- Images can be imported as a matrix of pixel values
  - » **`im=imread('myPic.jpg');`**
  - » **`imshow(im);`**

- Matlab supports almost all image formats
  - ➢ jpeg, tiff, gif, bmp, png, ...
  - ➢ see **help imread** for details (e.g., pixel format and types)

- To write an image, give:
  - ➢ rgb matrix (0 to 1 doubles, or 0 to 255 uint8)
  - » **`imwrite(rand(300,300,3),'t1.jpg');`**
  - ➢ indices and colormap
  - » **`imwrite(ceil(rand(200)*256),jet(256),'t2.jpg');`**
  - ➢ see **help imwrite** for more options

MATLAB's built-in images

```
AT3_1m4_01.tif          AT3_1m4_02.tif
AT3_1m4_03.tif          AT3_1m4_04.tif
AT3_1m4_05.tif          AT3_1m4_06.tif
AT3_1m4_07.tif          AT3_1m4_08.tif
AT3_1m4_09.tif          AT3_1m4_10.tif
autumn.tif              bag.png
blobs.png               board.tif
cameraman.tif           canoe.tif
cell.tif                circbw.tif
circles.png             circuit.tif
coins.png               concordaerial.png
concordorthophoto.png   eight.tif
fabric.png              football.jpg
forest.tif              gantrycrane.png
glass.png               greens.jpg
hestain.png             kids.tif
liftingbody.png         logo.tif
m83.tif                 mandi.tif
moon.tif                mri.tif
office_1.jpg            office_2.jpg
office_3.jpg            office_4.jpg
office_5.jpg            office_6.jpg
onion.png               paper1.tif
pears.png               peppers.png
pillsetc.png            pout.tif
rice.png                saturn.png
shadow.tif              snowflakes.png
spine.tif               tape.png
testpat1.png            text.png
tire.tif                tissue.png
trees.tif               westconcordaerial.png
westconcordorthophoto.png
```

Load these like you'd load anything else in your current directory:

```
>> load(cameraman.tif');
```

Outline

(1) Probability and Statistics
(2) Data Structures
(3) Images
(4) File I/O

Importing Data

- Matlab is a great environment for processing data. If you have a text file with some data:

```
jane joe jimmy
10 11 12
5 4 2
5 6 4
```

- To import data from files on your hard drive, use importdata
  - » **`a=importdata('textFile.txt');`**
    - ➢ a is a struct with `data`, `textdata`, and `colheaders` fields

```
a =
         data: [3x3 double]
     textdata: {'jane'  'joe'  'jimmy'}
   colheaders: {'jane'  'joe'  'jimmy'}
```

  - » **`x=a.data;`**
  - » **`names=a.colheaders;`**

Importing Data

- With `importdata`, you can also specify delimiters. For example, for comma separated values, use:
  - » `a=importdata('filename', ',');`
    - ➢ The second argument tells matlab that the tokens of interest are separated by commas

- `importdata` is very robust, but sometimes it can have trouble. To read files with more control, use `fscanf` (similar to C/Java), `textscan`. See **help** for information on how to use these functions

Writing Excel Files

- Matlab contains specific functions for reading and writing Microsoft Excel files

- To write a matrix to an Excel file, use **`xlswrite`**
  ```
  » xlswrite('randomNumbers',rand(10));
  » xlswrite('randomNumbers',rand(10),...
    'Sheet1','C11:L20');
  ```
  - ➢ Sheet name and range optional

- You can also write a cell array if you have mixed data:
  ```
  » C={'hello','goodbye';10,-2;-3,4};
  » xlswrite('randomNumbers',C,'mixedData');
  ```

- See **help xlswrite** for more usage options

Reading Excel Files

- Reading excel files is equally easy

- To read from an Excel file, use **xlsread**
  - » `[num,txt,raw]=xlsread('randomNumbers.xls');`
    - ➢ Reads the first sheet
    - ➢ `num` contains numbers, `txt` contains strings, `raw` is the entire cell array containing everything
  - » `[num,txt,raw]=xlsread('randomNumbers.xls',... 'mixedData');`
    - ➢ Reads the **mixedData** sheet
  - » `[num,txt,raw]=xlsread('randomNumbers.xls',-1);`
    - ➢ Opens the file in an Excel window and lets you click on the data you want!

- See **help xlsread** for even more fancy options

Reading ANY File

- You can read any file as binary data

- To read from a file, use fopen
  - » **`fid = fopen('fileName', 'r');`**
    - ➢ Returns a handle to a file
  - » **`data = fread(fid, 10);`**
    - ➢ Reads the next 10 bytes from the file and stores them in **`data`**
  - » **`fseek(fid, 5, 0);`**
    - ➢ Moves forward 5 bytes from the current position

- See **help fopen/fread/fwrite/ftell/fseek** for even more fancy options

Lecture 5

- Not mandatory – but highly recommended!

- More cool stuff Matlab has to offer

- Some things we can cover:
  - Animations
  - Build a GUI for your projects!
  - Use cool toolboxes
  - Interact with hardware (scopes, analyzers, Arduino, Raspberry PI, Lego Mindstorm…)
  - Use Simulink to graphically build complex systems and simulate
  - Do image processing
  - Plus… No Homework assignment!

Don't Forget….

- Comment your code!
- help and Google are your best friends – use them!
- Vectorize whenever possible
- Matlab is powerful but it is not a substitute for your own insights

End of Lecture 4

(1) Probability and Statistics
(2) Data Structures
(3) Images
(4) File I/O

THE END (ALMOST)

6.057
Introduction to programming in MATLAB

Lecture 5: Various functions and toolboxes

Orhan Celiker

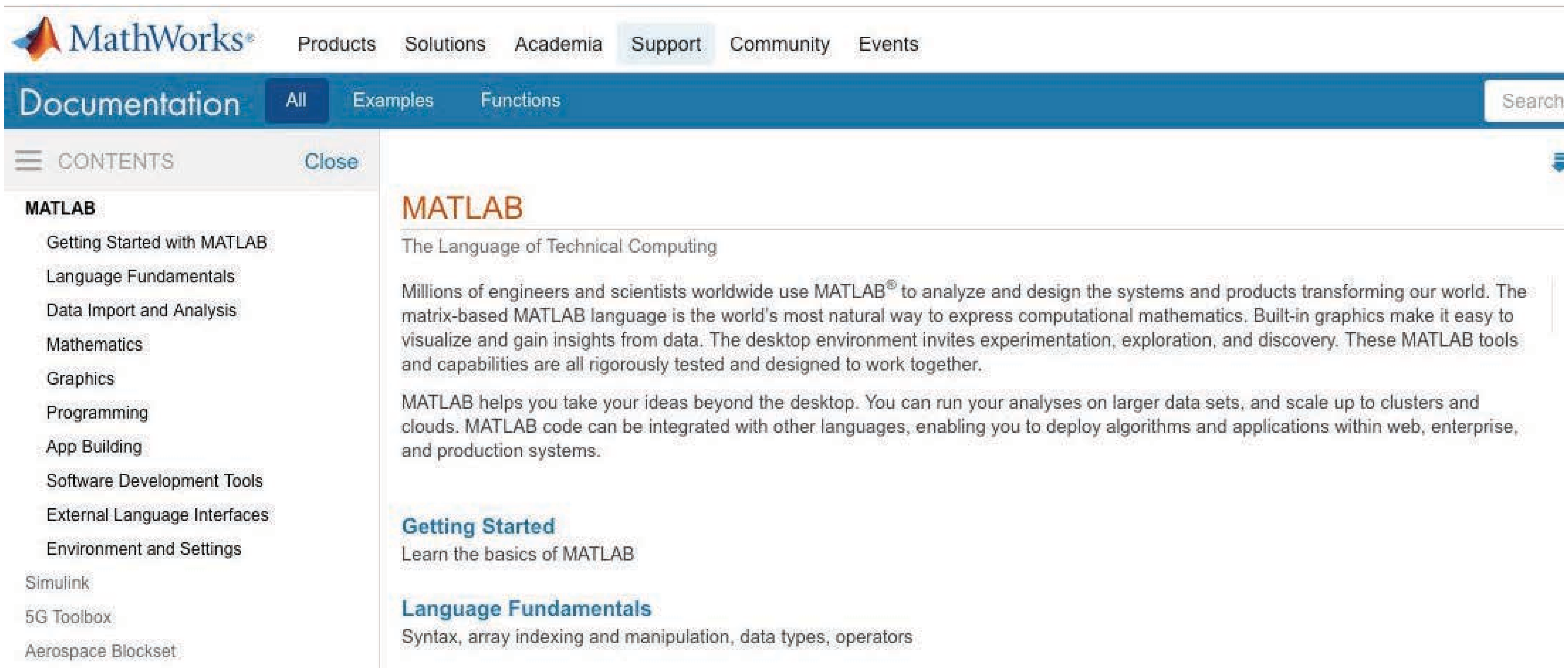
IAP 2019

Outline

- Documentation
- Misc. Useful Functions
- Graphical User Interfaces
- Simulink
- Symbolic Toolbox
- Image Processing
- Hardware Interface

Official Documentation

- http://www.mathworks.com/help/matlab/



Miscellaneous Matlab (1)

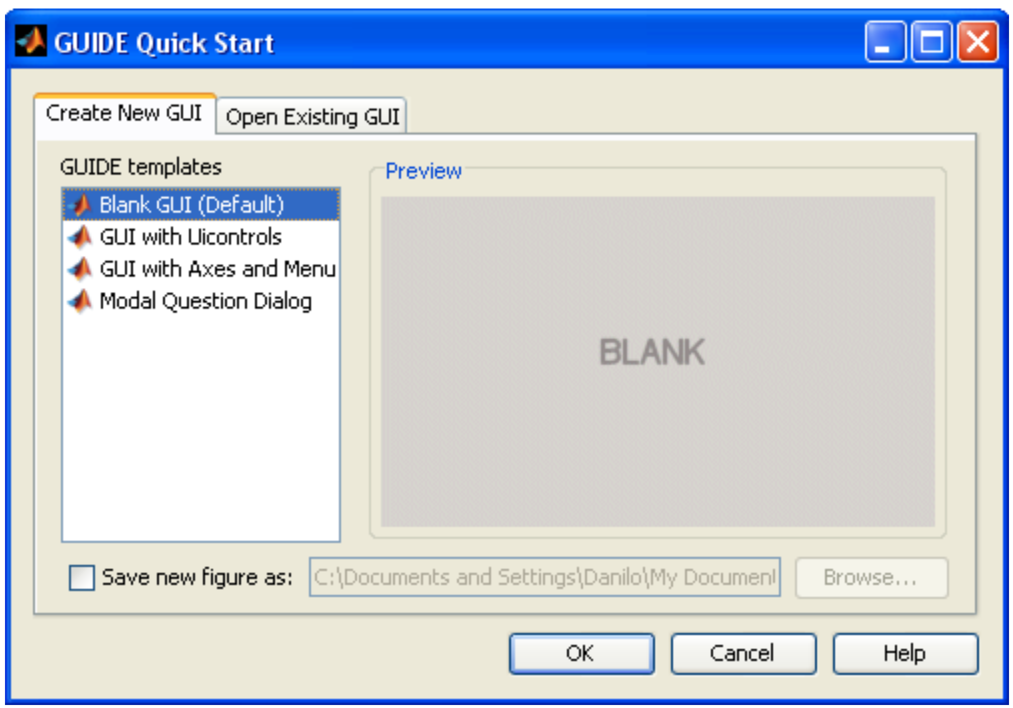
- The command **deal** can make variable initialization simpler
  - » `[x, y, z] = deal(zeros(20, 30));`
  - » `[a, b, c, d] = 5;`
  - » `[m, n] = deal(1, 100);`
- The command **eval** can execute a string!
  - » `a1 = 1; n = 1;`
  - » `eval([‘a’ num2str(n) ‘ = 5;’]);`
  - » `disp([‘a1 is now ‘ num2str(a1)]);`
- The command **repmat** can create replicas easily
  - » `A = repmat([1 2;3 4], 2, 2);`
- Execute Perl scripts using the command **perl**
  - » `perl(‘myPerlFile.pl’);`

Miscellaneous Matlab (2)

- Use `regexp` for powerful regular expression operations
  » `str = 'The staff email is example@example.edu';`
  » `pat = '([\w-.])+@([\w-.])+';`
  » `r = regexp(str, pat, 'tokens')`
  » `name = r{1}{1};      % name = '6.057-staff'`
  » `domain = r{1}{2};   % domain = 'mit.edu'`
- Set the root defaults by using the handle 0
  » `get(0, 'Default')`
  » `set(0, 'DefaultLineLineWidth', 2);`
- Edit the datatip text display function to show customized information
- You can also import Java classes (but don't)
  » `import java.util.Scanner`
- If you're not sure about something – just ask Matlab `why`

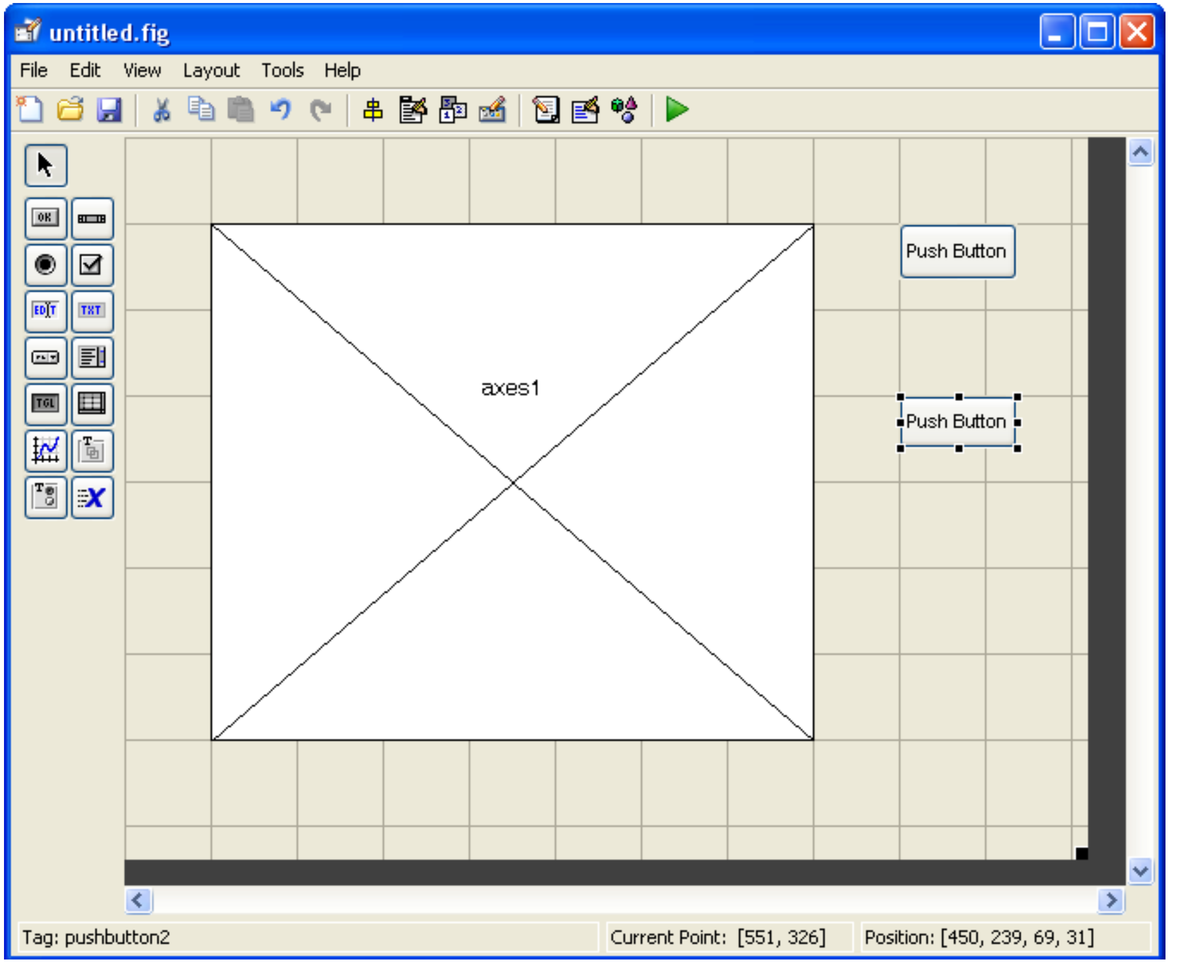
Making GUIs

- It's really easy to make a graphical user interface in Matlab
- To open the graphical user interface development environment, type **guide**
  - » **guide**
    - Select Blank GUI



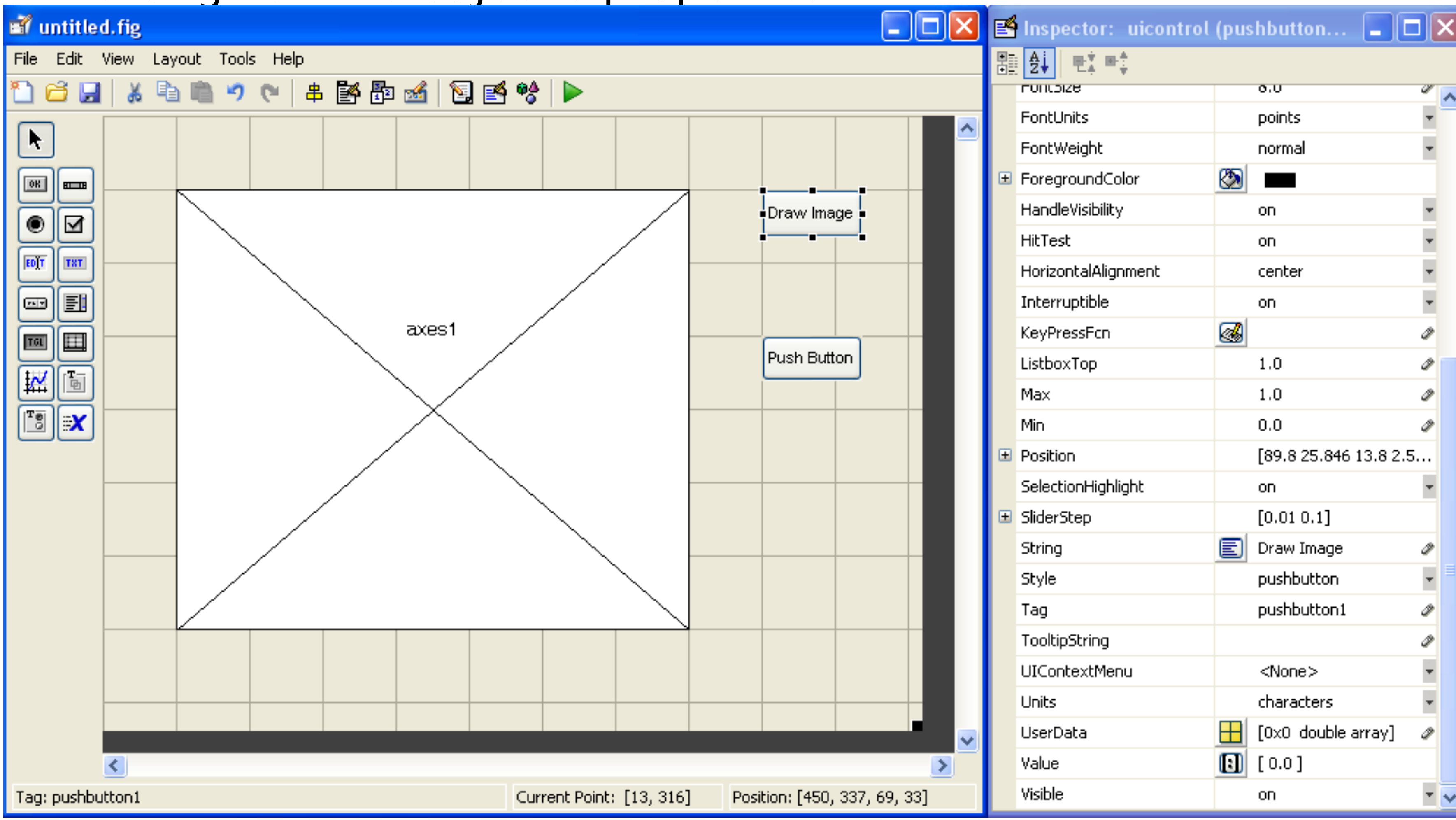
Draw the GUI

- Select objects from the left, and draw them where you want them



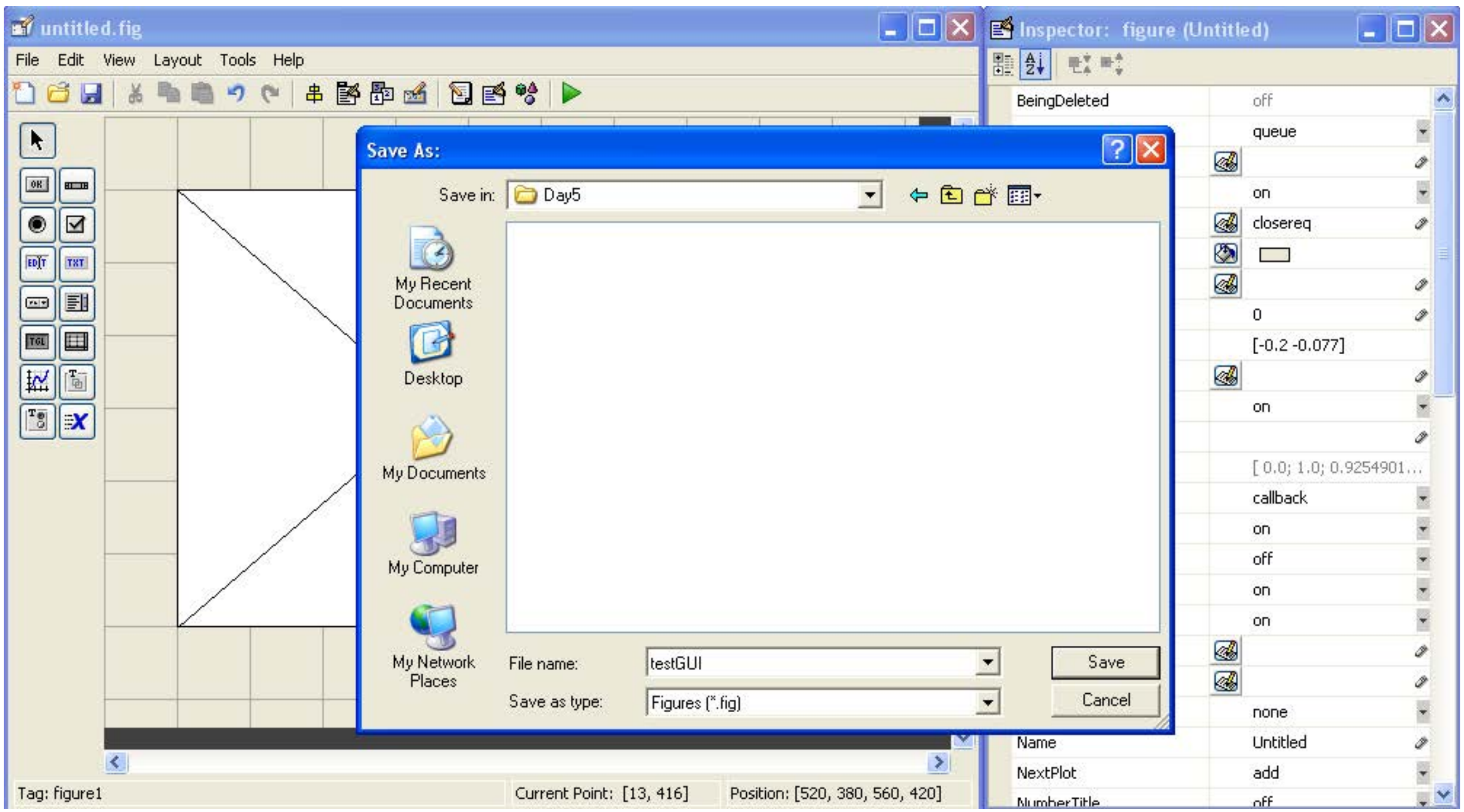
Change Object Settings

- Double-click on objects to open the Inspector. Here you can change all the object's properties.



Save the GUI

- When you have modified all the properties, you can save the GUI
- Matlab saves the GUI as a .fig file, and generates an m-file!



Add Functionality to M-File

- To add functionality to your buttons, add commands to the 'Callback' functions in the m-file. For example, when the user clicks the Draw Image button, the **`drawimage_Callback`** function will be called and executed
- All the data for the GUI is stored in the handles, so use **`set`** and **`get`** to get data and change it if necessary
- Any time you change the handles, save it using **`guidata`**
  - » **`guidata(handles.Figure1,handles);`**

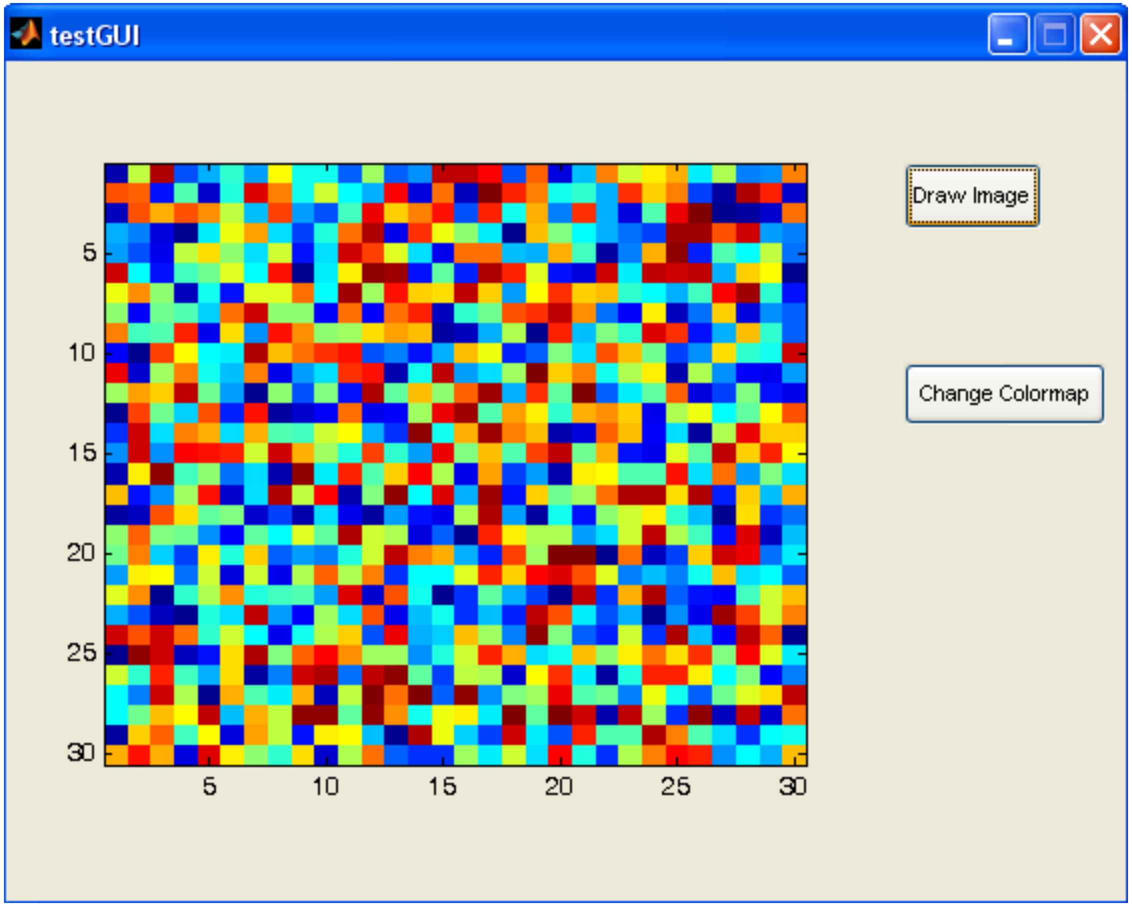
```
% --- Executes on button press in drawimage.
function drawimage_Callback(hObject, eventdata, handles)
% hObject    handle to drawimage (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in changeColormap.
function changeColormap_Callback(hObject, eventdata, handles)
% hObject    handle to changeColormap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

textFile.txt × numbers.txt × testGUI.m ×

testGUI

Running the GUI

- To run the GUI, just type its name in the command window and the GUI will pop up. The debugger is really helpful for writing GUIs because it lets you see inside the GUI

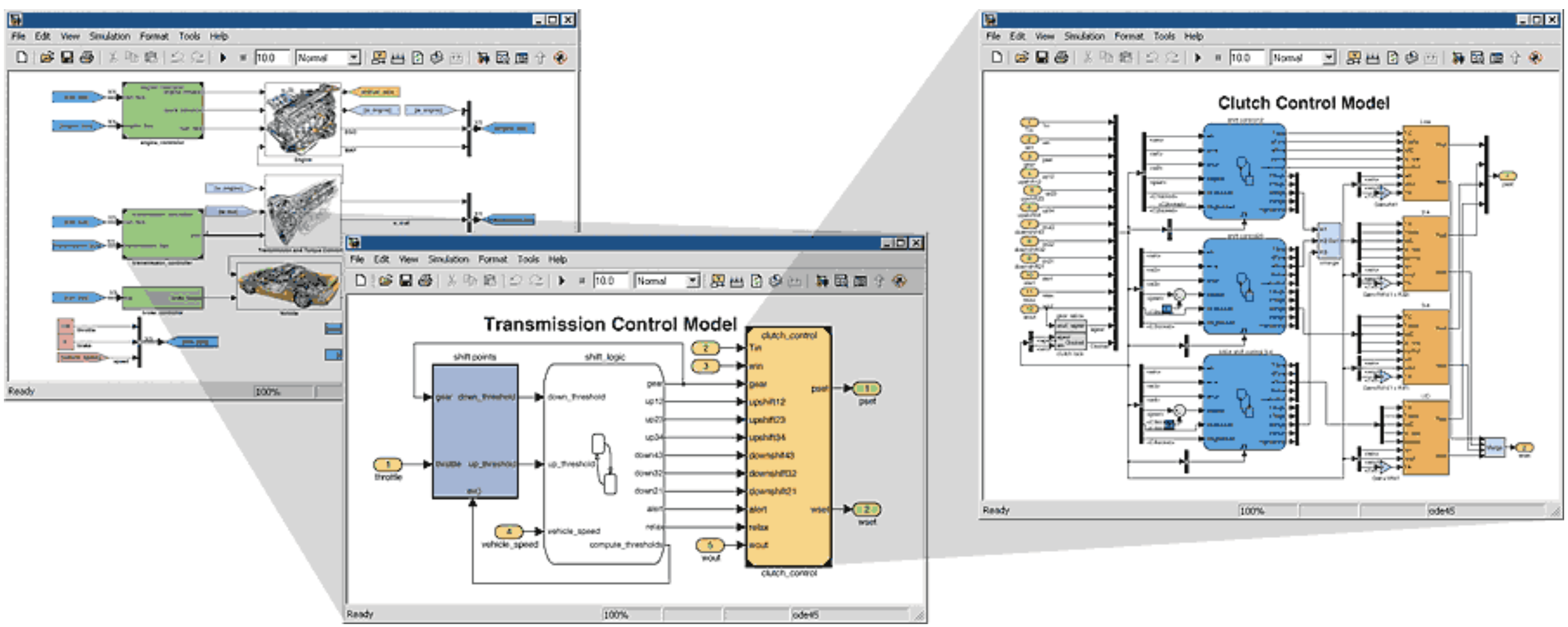


GUI Helper Functions

- Use keyboard to allow debugging from command window. GUI variables will appear in the workspace. Use return to exit debug mode
- Use built-in GUI modals for user input:
  » **`uigetfile`**
  » **`uiputfile`**
  » **`inputdlg`**
    - ➢And more… (see help for details)

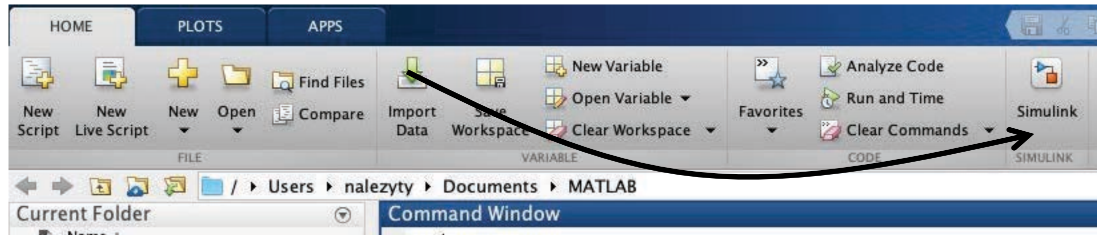
SIMULINK

- Interactive graphical environment
- Block diagram based MATLAB add-on environment
- Design, simulate, implement, and test control, signal processing, communications, and other time-varying systems

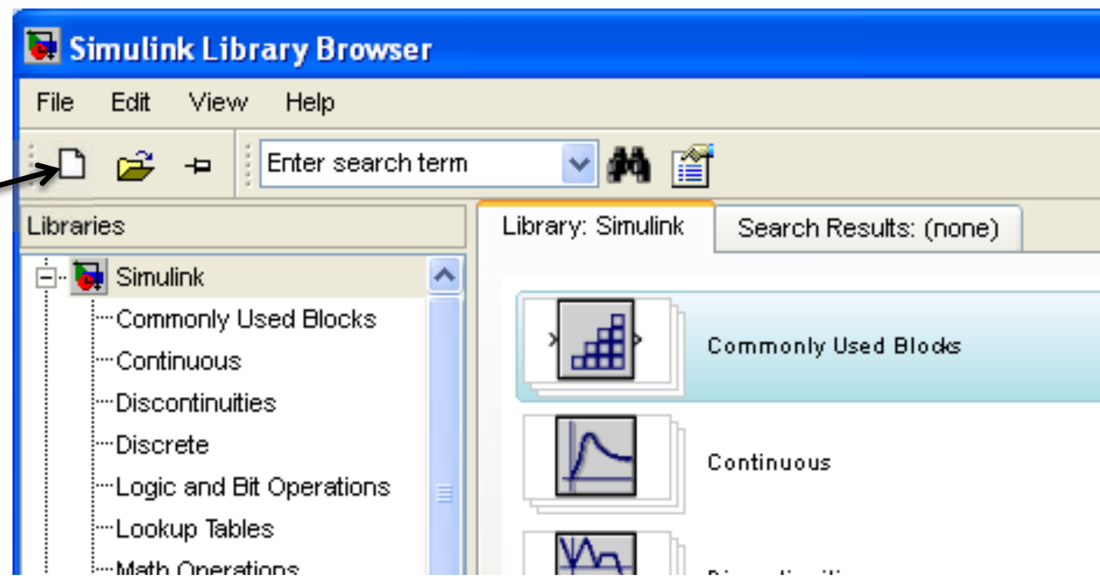


Getting Started

- In MATLAB, Start Simulink

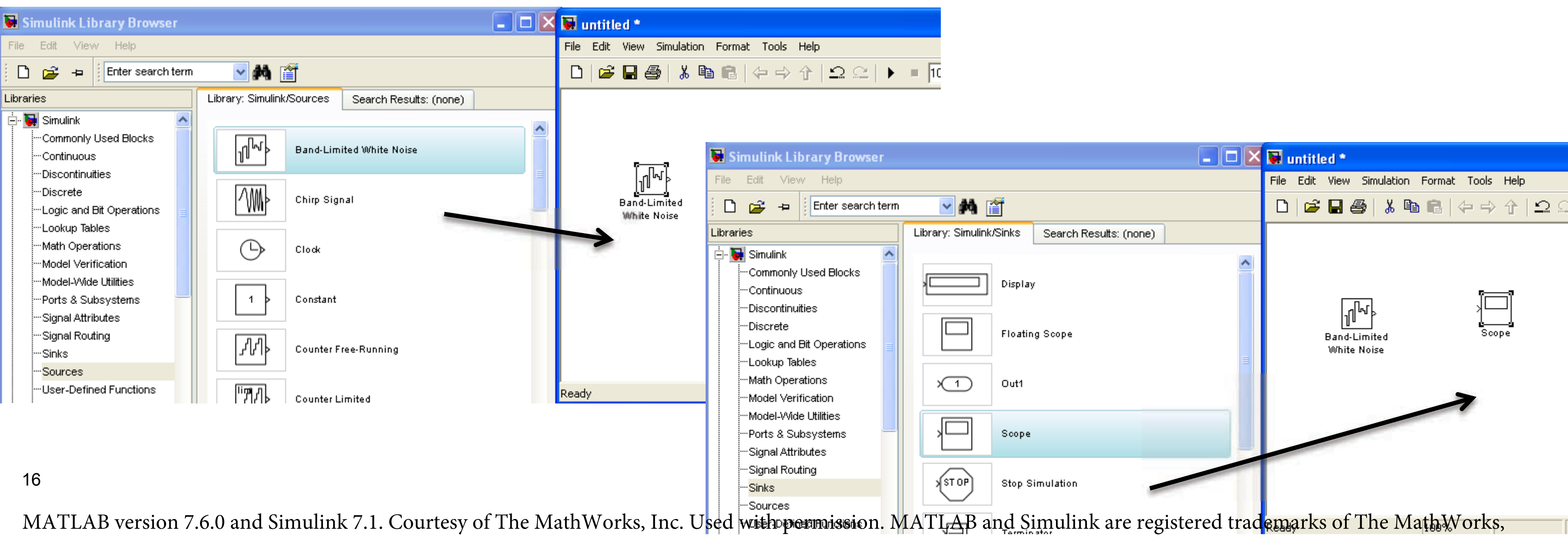


- Create a new Simulink file, similar to how you make a new script



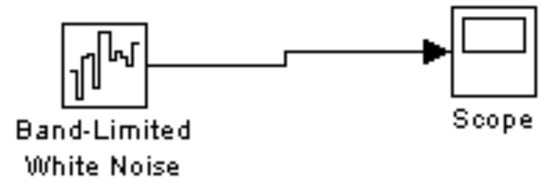
Simulink Library Browser

- The **Library Browser** contains various blocks that you can put into your model
- Examine some blocks:
  - ➢ Click on a library: "Sources"
    - – Drag a block into Simulink: "Band limited white noise"
  - ➢ Visualize the block by going into "Sinks"
    - – Drag a "Scope" into Simulink



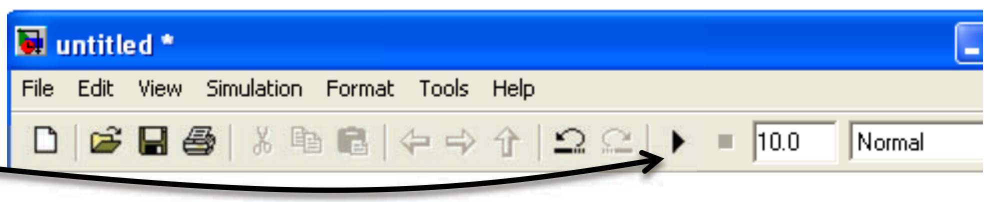
Connections

- Click on the carat/arrow on the right of the band limited white noise box



- Drag the line to the scope
  - You'll get a hint saying you can quickly connect blocks by hitting Ctrl
  - Connections between lines represent signals

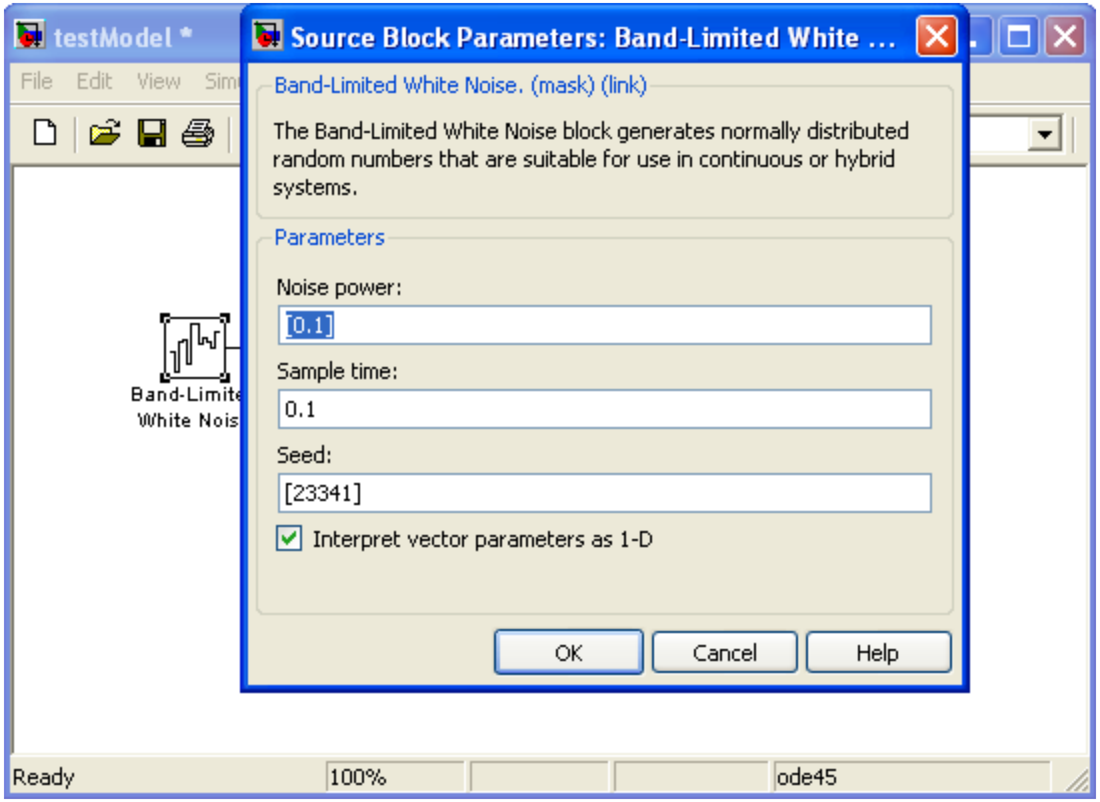
- Click the play button



- Double click on the scope.
  - This will open up a chart of the variable over the simulation time

Connections, Block Specification

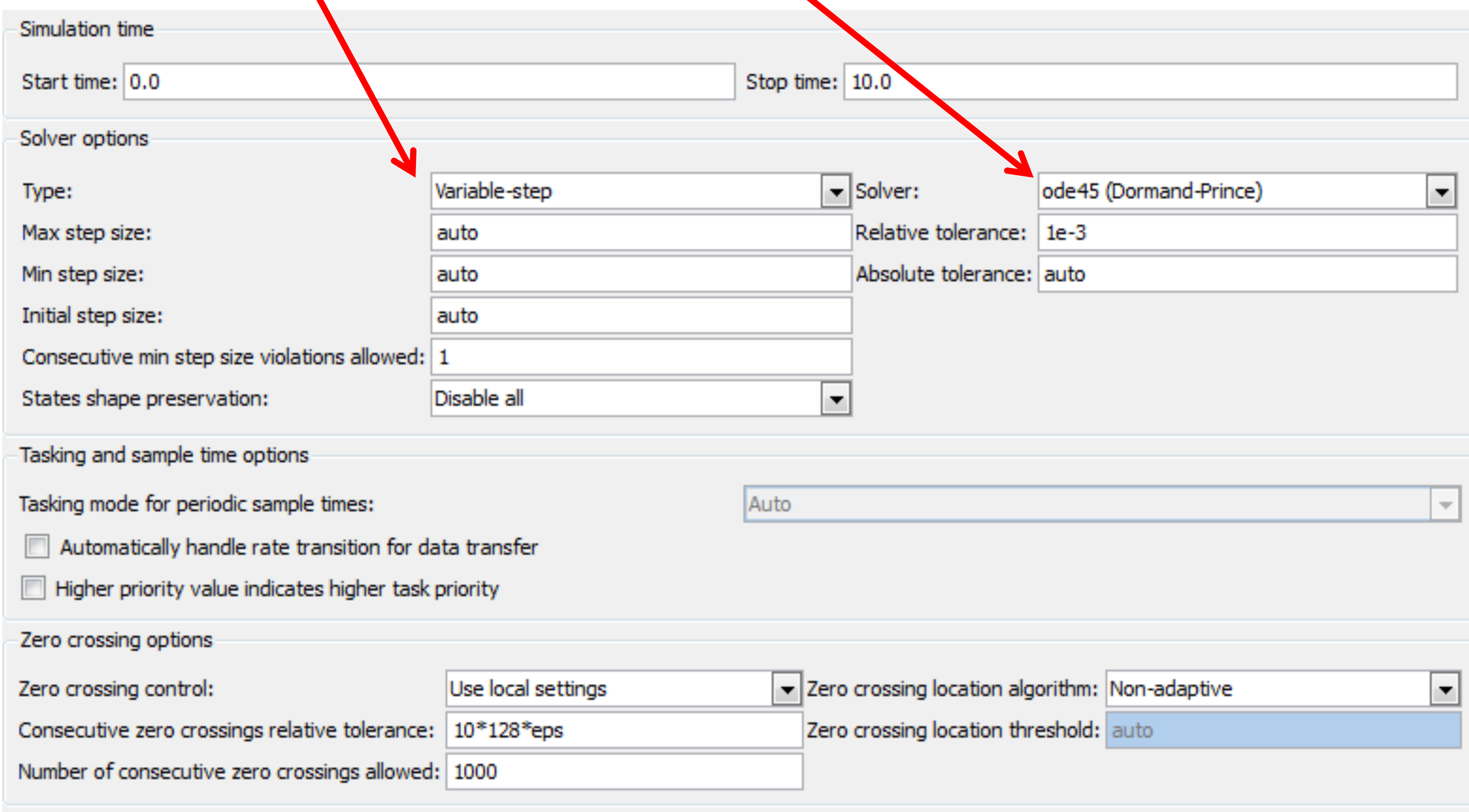
- To split connections, hold down 'Ctrl' when clicking on a connection, and drag it to the target block; or drag backwards from the target block
- To modify properties of a block, double-click it and fill in the property values.



Behind the curtain

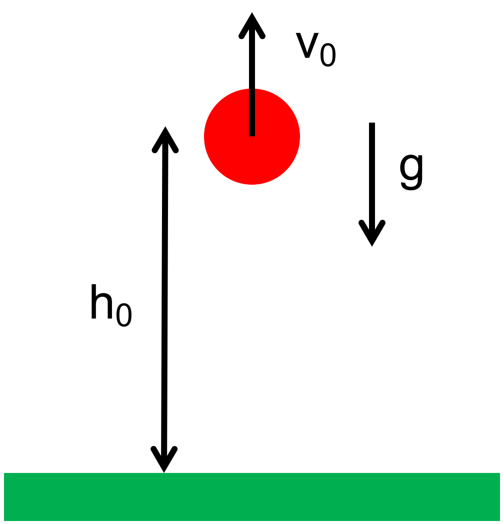
- Go to "Simulation"->"Configuration Parameters" at the top menu

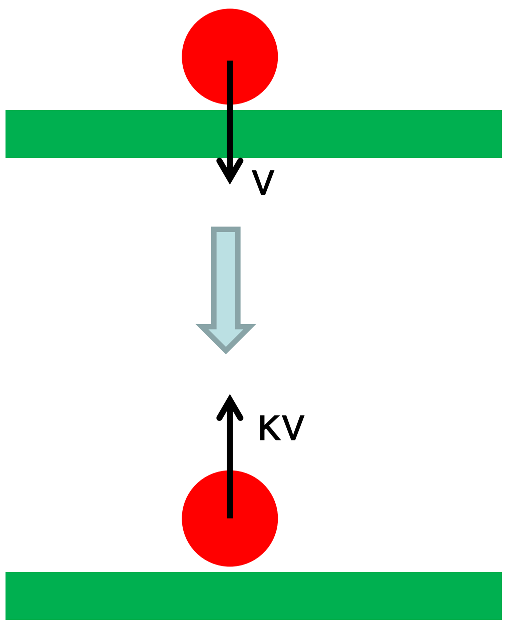
See ode45? Change the solver type here



Exercise: Bouncing Ball Model

- Let's consider the following 1 dimensional problem
- A rubber ball is thrown from height h0 with initial velocity v0 in the z-axis (up/down).
- When the ball hits the ground (z=0), its velocity instantaneously flips direction and is attenuated by the impact





Exercise: Bouncing Ball Model

- Let's consider the following 1 dimensional problem
- A rubber ball is thrown from height h0 with initial velocity v0 in the z-axis (up/down).
- When the ball hits the ground (z=0), its velocity instantaneously flips direction and is attenuated by the impact

$$m\frac{d^2z}{dt^2} = mg \quad v(t) = \frac{dz}{dt} \quad v\left(t^+\big|_{z=0}\right) = -\kappa v\left(t^-\big|_{z=0}\right)$$

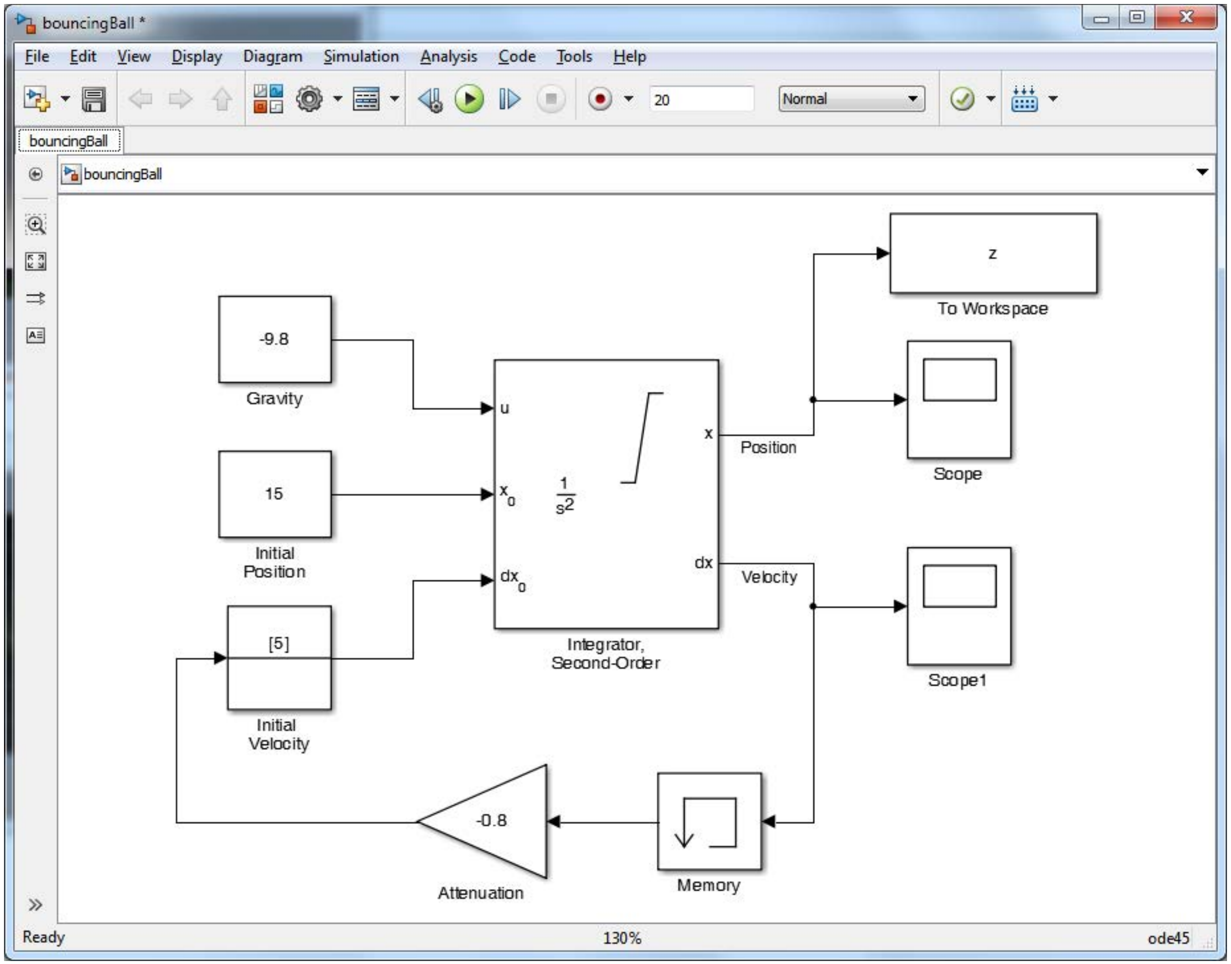
$$z(t=0) = h_0 \quad v(t=0) = v_0$$

- Integrating, we can obtain the balls height and velocity as a function of time

$$v(t) = \int_0^t g\,d\tau \quad z(t) = \int_0^t v(\tau)\,d\tau$$

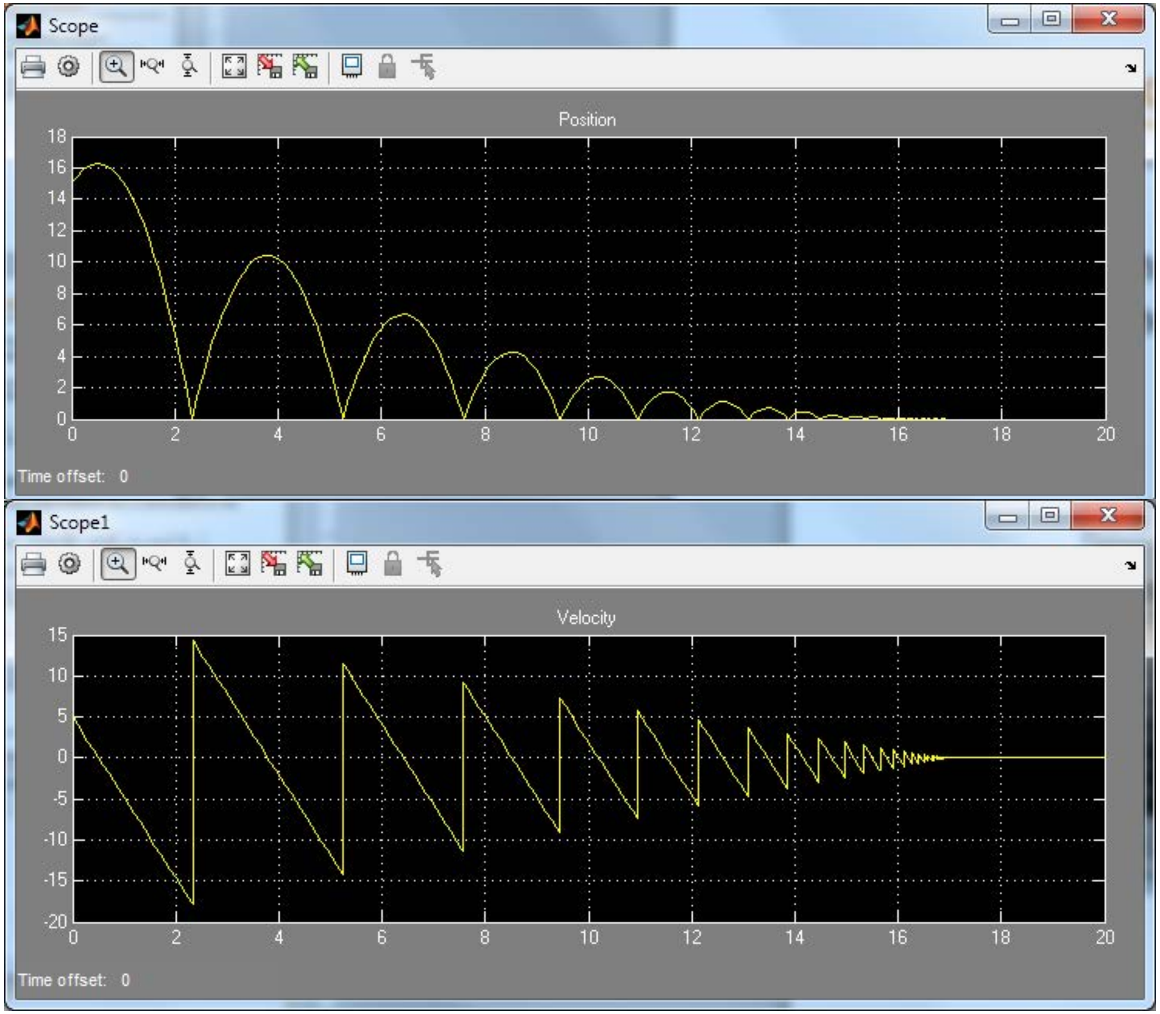
Exercise: Simulink Model

- Using the second order integrator with limits and reset, our model will look like this



Exercise: Simulink Results

- Running the model yields the balls height and velocity as a function of time



Toolboxes

- Math
  - ➢Takes the signal and performs a math operation
    - » `Add, subtract, round, multiply, gain, angle`
- Continuous
  - ➢Adds differential equations to the system
    - » `Integrals, Derivatives, Transfer Functions, State Space`
- Discontinuities
  - ➢Adds nonlinearities to your system
- Discrete
  - ➢Simulates discrete difference equations
  - ➢Useful for digital systems

Building systems

- Sources
  - » **`Step input, white noise, custom input, sine wave, ramp input,`**
    - ➢ Provides input to your system
- Sinks
  - » **`Scope: Outputs to plot`**
  - » **`simout: Outputs to a MATLAB vector (struct) on workspace`**
  - » **`Matlab mat file`**

Symbolic Toolbox

- Don't do nasty calculations by hand!
- Symbolics vs. Numerics

	Advantages	Disadvantages
Symbolic	•Analytical solutions •Lets you intuit things about solution form	•Sometimes can't be solved •Can be overly complicated
Numeric	•Always get a solution •Can make solutions accurate •Easy to code	•Hard to extract a deeper understanding •Num. methods sometimes fail •Can take a while to compute

Symbolic Variables

- Symbolic variables are a type, like **double** or **char**

- To make symbolic variables, use **`sym`**
  - » `a=sym('1/3');`
  - » `b=sym('4/5');`
  - » `mat=sym([1 2;3 4]);`
    - ➢ fractions remain as fractions
  - » `c=sym('c','positive');`
    - ➢ can add tags to narrow down scope
    - ➢ see **help sym** for a list of tags

- Or use **`syms`**
  - » `syms x y real`
    - ➢ shorthand for x=sym('x','real'); y=sym('y','real');

Symbolic Expressions

- Multiply, add, divide expressions
  - » **d=a*b**
    - ➢ does 1/3*4/5=4/15;

```
d =
4/15
```

  - » **expand((a-c)^2);**
    - ➢ multiplies out

```
ans =
1/9-2/3*c+c^2
```

  - » **factor(ans)**
    - ➢ factors the expression

```
ans =
1/9*(3*c-1)^2
```

  - » **pretty(ans)**
    - ➢ makes it look nicer

```
          2
(3 c - 1)
-----------
    9
```

Cleaning up Symbolic Statements

» `collect(3*x+4*y-1/3*x^2-x+3/2*y)`

- collects terms

```
ans =
2*x+11/2*y-1/3*x^2
```

» `simplify(cos(x)^2+sin(x)^2)`

```
ans =
1
```

- simplifies expressions

» `subs('c^2',c,5)`

```
ans =

    25
```

- replaces variables with numbers or expressions. To do multiple substitutions pass a cell of variable names followed by a cell of values

» `subs('c^2',c,x/7)`

```
ans =

x^2/49
```

More Symbolic Operations

- We can do symbolics with matrices too
  - » `mat=sym('[a b;c d]');`
  - » `mat=sym('A%d%d', [2 2]);`
    - ➢ symbolic matrix of specified size
  - » `mat2=mat*[1 3;4 -2];`
    - ➢ compute the product
  - » `d=det(mat)`
    - ➢ compute the determinant
  - » `i=inv(mat)`
    - ➢ find the inverse

```
mat2 =
[    a+4*b,  3*a-2*b]
[    c+4*d,  3*c-2*d]
```

```
d =
a*d-b*c
```

```
i =
[  d/(a*d-b*c),  -b/(a*d-b*c)]
[ -c/(a*d-b*c),   a/(a*d-b*c)]
```

- You can access symbolic matrix elements as before
  - » `i(1,2)`

```
ans =
-b/(a*d-b*c)
```

Exercise: Symbolics

- The equation of a circle of radius r centered at (a,b) is given by: $(x-a)^2+(y-b)^2=r^2$
- Use **`solve`** to solve this equation for x and then for y

- It's always annoying to integrate by parts. Use **`int`** to do the following integral symbolically and then compute the value by **`subs`**tituting 0 for a and 2 for b: $\int_a^b xe^x dx$

Exercise: Symbolics

- The equation of a circle of radius r centered at (a,b) is given by: $(x-a)^2+(y-b)^2=r^2$
- Use **solve** to solve this equation for x and then for y

```
» syms a b r x y
» solve('(x-a)^2+(y-b)^2=r^2','x')
» solve('(x-a)^2+(y-b)^2=r^2','y')
```

- It's always annoying to integrate by parts. Use **int** to do the following integral symbolically and then compute the value by **subs**tituting 0 for a and 2 for b: $\int_a^b xe^x dx$

```
» Q=int('x*exp(x)',a,b)
» subs(Q,{a,b},{0,2})
```

Image Processing

- http://www.mathworks.com/help/images/index.html

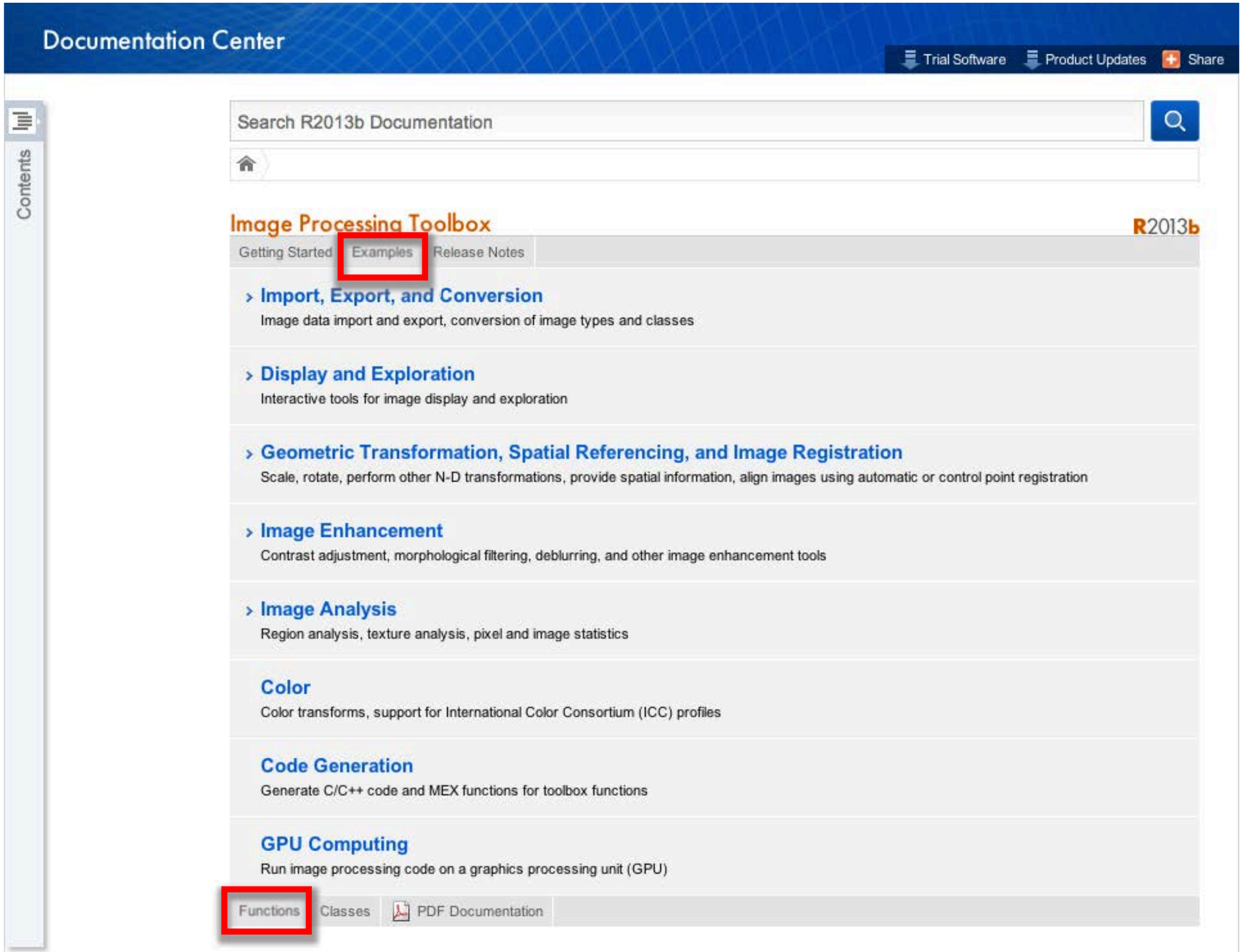


Image Processing

- Image enhancement
  - Adjust image contrast, intensities, etc.
- Filtering and deblurring
  - Convolution and deconvolution

- Finding edges
  - Image gradient, `edge`
- Finding circles
  - Hough transform

- Training an object detector
  - Computer vision toolbox: `trainCascadeObjectDetector`

Image Processing

- Image Restoration
  - Denoising
- Image Enhancement & Analysis
  - Contrast Improvement
    - imadjust, histeq, adapthisteq
  - Edge Detection
    - edge
  - Image Sharpening
  - Image Segmentation
- Image Compression
  - Wavelet toolbox (Chap. 3 of Gonzalez book on DIP)

Exercise: Contrast Improvement

- In this exercise, first we want to load the image "pout.tif". You can use **`imread.`**

- Then for a better comparison we want our image to have a width of 200 pixels. Use **`imresize`**

- Finally, we want to compare the results of three functions **`imadjust, histeq, adapthisteq`** for contrast enhancement. Display the original image and the three enhanced images in a single figure.

Exercise: Contrast Improvement

```
» % Loading the our image into the workspace
» Image                  = imread('pout.tif');
»
» % For comparison, it is better to have a predefined width
» width                  = 200;
»
» % Resizing the image using bicubic interpolation
» dim                    = size(Image);
» Image                  = imresize(Image , width * [dim(1) / dim(2) 1] , 'bicubic');
»
» % Adjusting the contrast using imadjust
» Image_imadjust         = imadjust(Image);
»
» % Adjusting the contrast using histogram equalization
» Image_histeq           = histeq(Image);
»
» % Adjusting the contrast using adaptive histogram equalization
» Image_adapthisteq      = adapthisteq(Image);
»
```

Exercise: Contrast Improvement

```
» % Displaying the original image and the results in a single figure to compare with each other
» figure
» subplot(2 , 2 , 1);
» imshow(Image);
» title('Original Image');
»
» subplot(2 , 2 , 2);
» imshow(Image_imadjust);
» title('Enhanced Image using Imadjust');
»
» subplot(2 , 2 , 3);
» imshow(Image_histeq);
» title('Enhanced Image using Histeq');
»
» subplot(2 , 2 , 4);
» imshow(Image_adapthisteq);
» title('Enhanced Image using Adapthisteq');
```

Exercise: Contrast Improvement

Original Image

Enhanced Image using Imadjust

Enhanced Image using Histeq

Enhanced Image using Adapthisteq

Exercise: Edge Detection

- We know that edge detection is mainly highpass filtering the image.

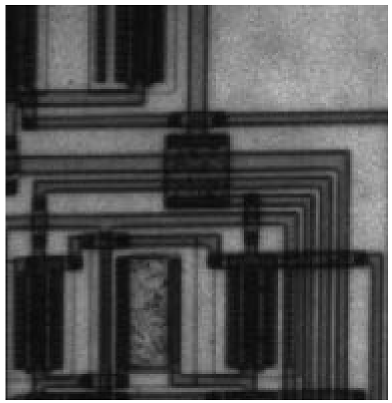
- First load the image "circuit.tif" and then plot the edges in that figure using the function **`edge`** and the filters **`"sobel"`**, **`"prewitt"`**. Also use **`"canny"`** as another method for edge detection using **`edge`**.

Exercise: Edge Detection

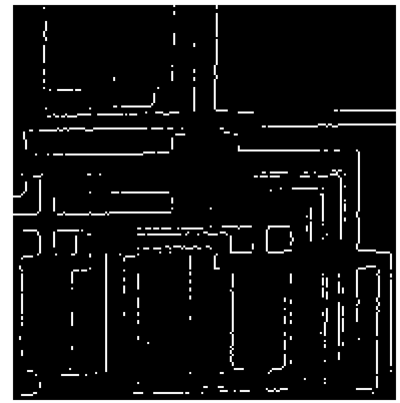
```
» I   = imread('circuit.tif');
» I1  = edge(I , 'sobel');
» I2  = edge(I , 'canny');
» I3  = edge(I , 'prewitt');
»
» figure
» subplot(2 , 2 , 1);
» imshow(I);
» title('Original Image');
»
» subplot(2 , 2 , 2);
» imshow(I1);
» title('Edges found using sobel filter');
»
» subplot(2 , 2 , 3);
» imshow(I2);
» title('Edges found using the "canny" method');
»
» subplot(2 , 2 , 4);
» imshow(I3);
» title('Edges found using prewitt filter');
```

Exercise: Edge Detection

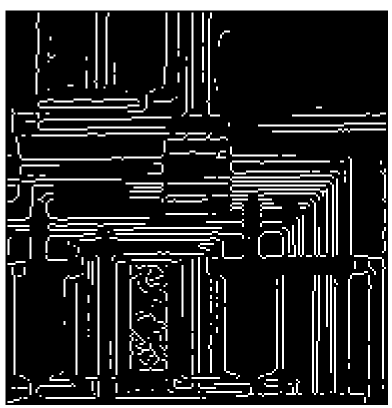
Original Image



Edges found using sobel filter



Edges found using the "canny" method



Edges found using prewitt filter

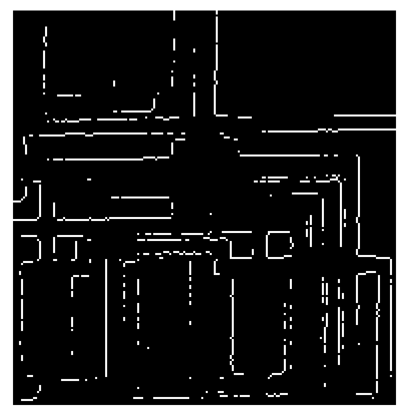


Image Enhancement

- Commonly-used: **`imread`**, **`imwrite`**, **`imshow`**, **`imresize`**

```
» im = imread('pout.tif');
  % image included in toolbox
» imtool(im);
```

  - ➢ Convenient for editing in figure window

- Adjust intensity values / colormap

```
» imadjust(im);
```

  - ➢ Increase contrast
    (1% of data saturated at low/high intensities)

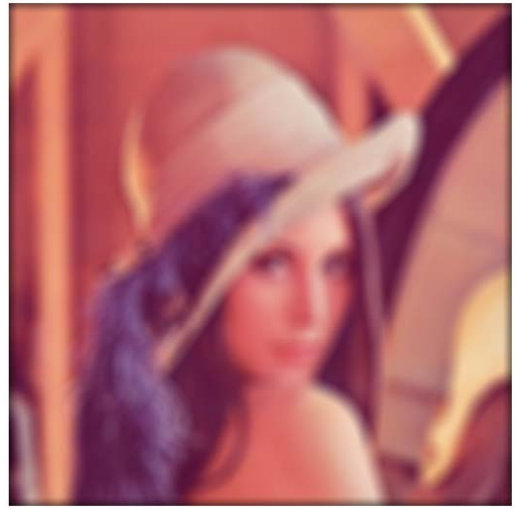
```
» imadjust(im,[.4 .6],[0 1]);
```

  - ➢ Clips off intensities below .4 and above .6
    Stretches resulting intensities to 0 and 1
  - ➢ What happens if used [1 0] instead of [0 1]?
  - ➢ Also works for RGB; see **doc**

Filtering and Deblurring

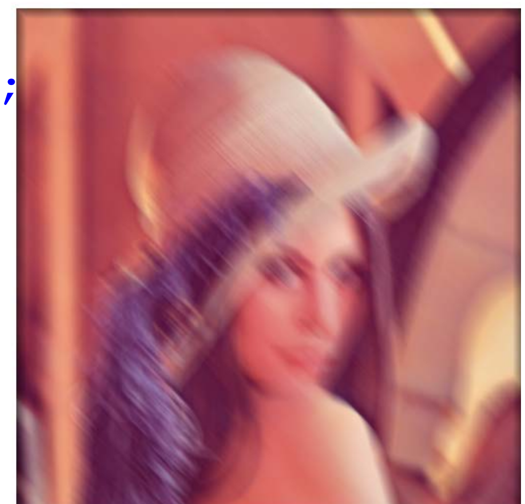
Pillbox filter:

```
f = fspecial('disk',10);
imblur = imfilter(im,f);
deconvblind(imblur,f);
```



Linear motion blur:

```
f=fspecial('motion',30,135);
imblur = imfilter(im,f);
deconvblind(imblur,f);
```



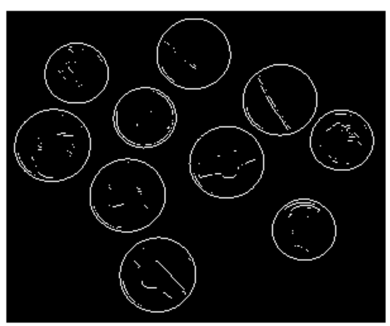
Deblurring	
deconvblind	Deblur image using blind deconvolution
deconvlucy	Deblur image using Lucy-Richardson m
deconvreg	Deblur image using regularized filter
deconvwnr	Deblur image using Wiener filter

Finding Edges

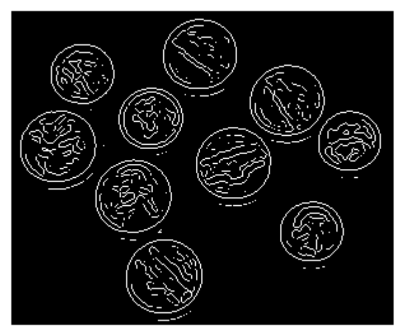
- Image gradients: **imgradient, imgradientxy**
- Application: **edge**
  - » `edge(im); % Sobel`
  - » `edge(im, 'canny');`
- Images must be in grayscale
  - » `rgb2gray`

Original (coins.png)

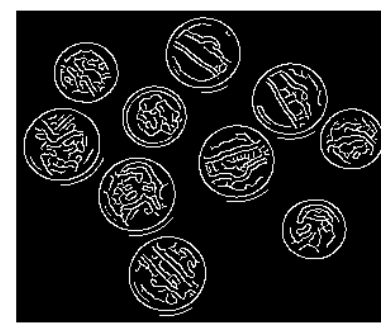
Sobel



Laplacian



Canny



Other Cool Stuff

- Finding circles
  - » `im = imread('coins.png');`
  - » `[centers,radii,metric] = imfindcircles(im, [15 30]);`
    - ➢ Finds circles with radii within range, ordered by strength
  - » `imshow(im)`
  - » `viscircles(centers(1:5,:), radii(1:5));`

- Extract other shapes with Hough transform

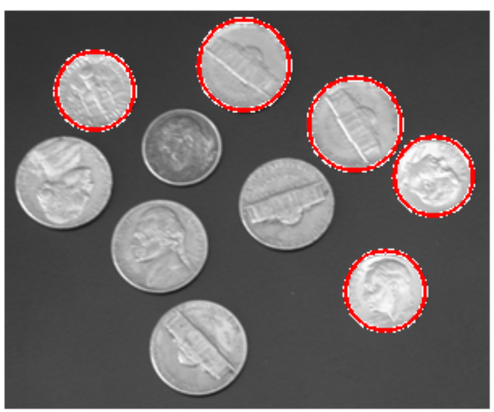


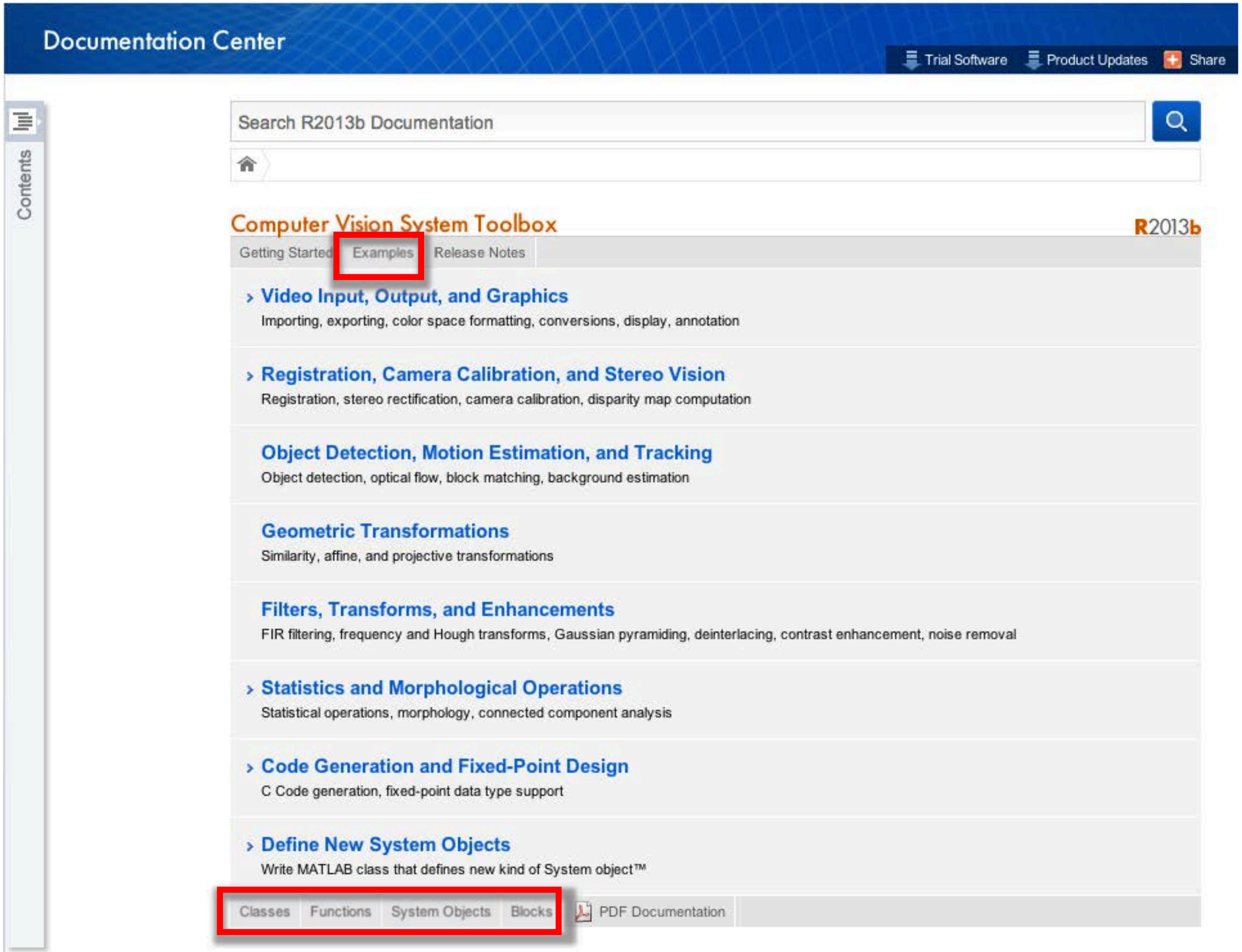
Image Analysis

Object Analysis

`bwboundaries`	Trace region boundaries in binary image
`bwtraceboundary`	Trace object in binary image
`corner`	Find corner points in image
`cornermetric`	Create corner metric matrix from image
`edge`	Find edges in intensity image
`hough`	Hough transform
`houghlines`	Extract line segments based on Hough transform
`houghpeaks`	Identify peaks in Hough transform
`imfindcircles`	Find circles using circular Hough transform
`imgradient`	Gradient magnitude and direction of an image
`imgradientxy`	Directional gradients of an image

… and also Computer Vision

- http://www.mathworks.com/help/vision/index.html



… and also Computer Vision

- http://www.mathworks.com/help/vision/functionlist.html

Feature Detection, Extraction, and Matching

detectFASTFeatures	Find corners using FAST algorithm
detectHarrisFeatures	Find corners using Harris–Stephens algorithm
detectMinEigenFeatures	Find corners using minimum eigenvalue algorithm
detectMSERFeatures	Detect MSER features
detectSURFFeatures	Detect SURF features
extractFeatures	Extract interest point descriptors
extractHOGFeatures	Extract Histograms of Oriented Gradients (HOG) features
matchFeatures	Find matching features
showMatchedFeatures	Display corresponding feature points
binaryFeatures	Object for storing binary feature vectors
cornerPoints	Object for storing corner points
SURFPoints	Object for storing SURF interest points
MSERRegions	Object for storing MSER regions
vision.BoundaryTracer	Trace object boundary
vision.CornerDetector	Detect corner features
vision.EdgeDetector	Find object edge

Object Detection, Motion Estimation, and Tracking

configureKalmanFilter	Create Kalman filter for object tracking
disparity	Disparity map between stereo images
trainCascadeObjectDetector	Train cascade object detector model
detectFASTFeatures	Find corners using FAST algorithm
detectHarrisFeatures	Find corners using Harris–Stephens algorithm
detectMinEigenFeatures	Find corners using minimum eigenvalue algorithm
detectMSERFeatures	Detect MSER features
detectSURFFeatures	Detect SURF features
extractFeatures	Extract interest point descriptors
extractHOGFeatures	Extract Histograms of Oriented Gradients (HOG) features
insertObjectAnnotation	Annotate truecolor or grayscale image or video stream
assignDetectionsToTracks	Assign detections to tracks for multiobject tracking
matchFeatures	Find matching features
cornerPoints	Object for storing corner points
SURFPoints	Object for storing SURF interest points
MSERRegions	Object for storing MSER regions
vision.KalmanFilter	Kalman filter for object tracking
vision.BlockMatcher	Estimate motion between images or video frames
vision.CascadeObjectDetector	Detect objects using the Viola-Jones algorithm
vision.ForegroundDetector	Detects foreground using Gaussian mixture models
vision.HistogramBasedTracker	Histogram-based object tracking
vision.OpticalFlow	Estimate object velocities
vision.PeopleDetector	Detect upright people using HOG features
vision.PointTracker	Track points in video using Kanade-Lucas-Tomasi (KLT) algorithm
vision.TemplateMatcher	Locate template in image

Also consider OpenCV+MATLAB
http://www.mathworks.com/discovery/matlab-opencv.html

Object Detection

- Train a cascade object detector (introduced in R2013a)
- http://www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html
- http://www.mathworks.com/help/vision/ref/traincascadeobjectdetector.html

- Inputs to **`trainCascadeObjectDetector`**:
  - ➢ Image files with bounding boxes for positive instances
  - ➢ Image files of negative instances (‘background’)
  - ➢ Optional: FP/TP rates, # cascade stages, feature type
- Output: An XML file with object detector parameters

```
» detector=vision.CascadeObjectDetector('my.xml');
```

- Use the detector on new images:

```
» bbox=step(detector, imread('testImage.jpg'));
```

- See links above for full example

Machine Learning (Stats Toolbox)

- http://www.mathworks.com/help/stats/index.html

Supervised Learning

Regression, support vector machines, parametric and nonparametric classification, decision trees

Linear Regression
Multiple, stepwise, multivariate regression models, and more

Nonlinear Regression
Nonlinear fixed and mixed-effects regression models

Generalized Linear Models
Logistic regression, multinomial regression, Poisson regression, and more

Classification Trees and Regression Trees
Decision trees for regression and classification

Support Vector Machines
Support vector machines for binary classification

Discriminant Analysis
Linear and quadratic discriminant analysis classification

Naive Bayes Classification
Train Naive Bayes classifiers

Nearest Neighbors
Find nearest neighbors for classification

Model Building and Assessment
Feature selection, cross validation, predictive performance evaluation

Unsupervised Learning

Clustering, Gaussian mixture models, hidden Markov models

Hierarchical Clustering
Produce nested sets of clusters

k-Means Clustering
Cluster by minimizing mean distance

Gaussian Mixture Models
Cluster based on Gaussian mixture models using the EM algorithm

Hidden Markov Models
Markov models for data generation

Cluster Evaluation
Evaluate number of clusters

Ensemble Learning

Ensembles for Boosting, Bagging, or Random Subspace

Boosting
Improve predictions using AdaBoost, RobustBoost, GentleBoost, and more

Bagging
Improve predictions using bootstrap aggregation

Random Subspace
Improve predictions using random subspace

50

Hardware Interface

- Matlab can interact directly with many forms of external hardware, from lab equipment to standalone micro-controllers
- Interaction can be done at various levels of abstraction
- Ideal when processor intensive DSP is required and target system cannot handle it on it's own
- Probably not suitable for real-time systems due to the communication overhead

Low Level

- Most basic link – through the serial port using `serial`
  » `s = serial('com3')`
    - Can also provide additional properties, see `help serial`
- From here on, treat `s` as a file handler
  » `fopen(s)`
  » `fwrite(s, data)`
  » `fprintf(s, 'string');`
  » `res = fscanf(s);`
- Don't forget to close!
  » `fclose(s);`

GPIB

- GPIB – General Purpose Interface Bus (IEEE-488)
- Created by HP in the 1960's, but highly adopted today in many lab instruments
- A standardized communication protocol for sending and receiving information
- Simply create using the command gpib
  - » `g = gpib('agilent', 7, 1);`
    - ➢ See `help gpib` for option details
    - ➢ From now on, treat as file handler
  - » `fopen(g);`
  - » `fprintf(g, '*IDN?')`
  - » `idn = fscanf(g);`
- Don't forget to close!
  - » `fclose(g);`

Higher Levels

- Customized function packages for different platforms created by Mathworks and the user community
- http://www.mathworks.com/hardware-support/home.html
- http://makerzone.mathworks.com/

Where to go from here

- 6.555 Biomedical Signal and Image Processing*
- EdX MATLAB courses
  https://www.edx.org/learn/matlab
- GNU Octave (free software implementation of MATLAB)
  https://www.gnu.org/software/octave/
- MathWorks itself?

*and probably many other courses I'm not aware of

Takeaway lessons

- MATLAB is a MATrix LABoratory; optimized for parallel processing of large data
- It simplifies your computation, but cannot provide insights on its own
- Use MATLAB to process data, but always interpret results yourself
- When possible, vectorize computations for faster results
- Use **`help`** all day and every day
- If in doubt, Google your problem: MATLAB has excellent online documentation, and Stack Overflow has tons of answers
- Master the use of traceback and debugging tools
- Have fun!