PURDUE™ AERONAUTICS & ASTRONAUTICS

Introduction to MATLAB

Isaac Tetzloff

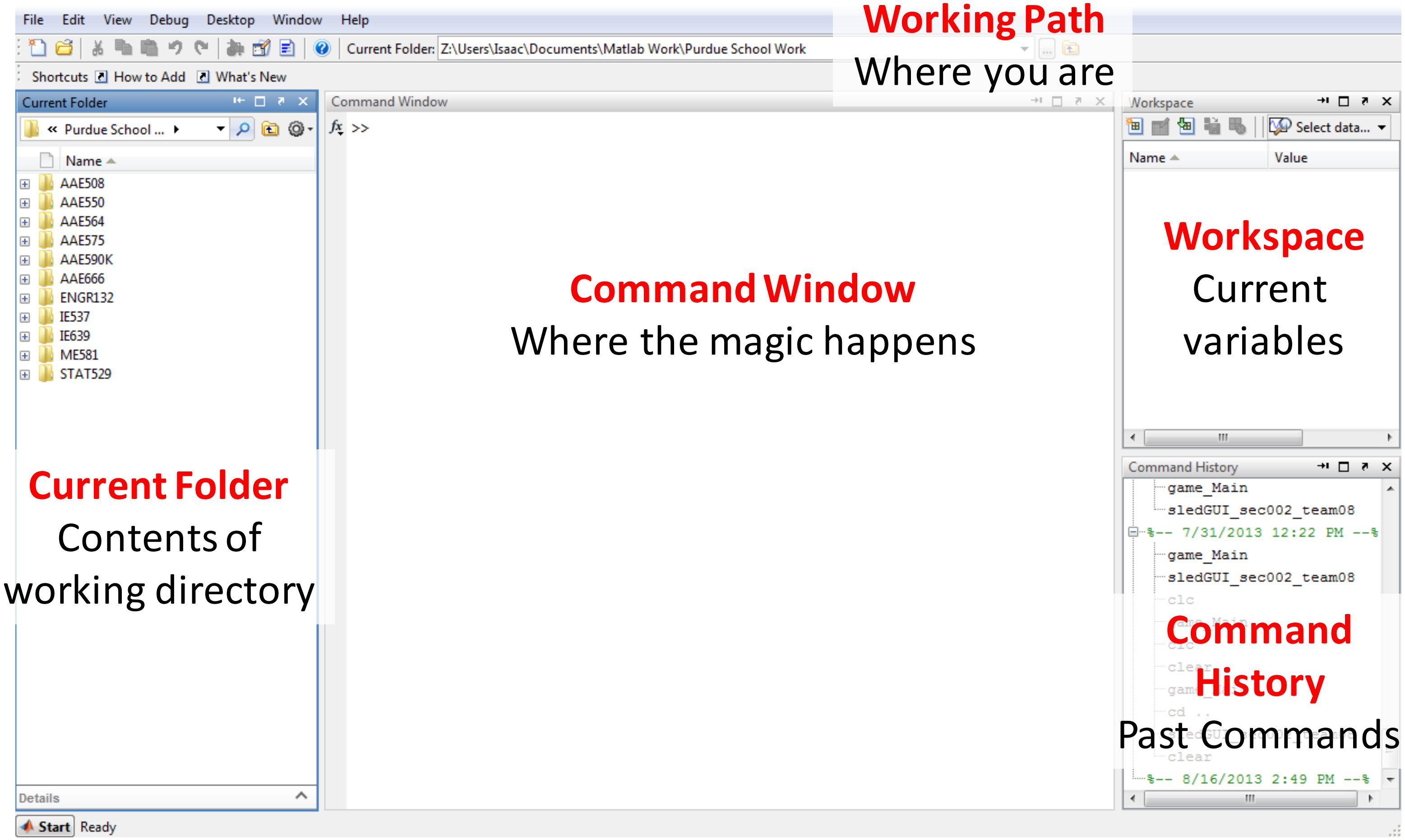
isaact@purdue.edu

Welcome To MATLAB

- MATLAB is a program for doing numerical computations, originally designed for solving linear algebra type problems
  - MATLAB = **MAT**rix **LAB**oratory
- MATLAB is an interpreter
  - Code does not need to be compiled
  - Can make a little slower than compiled code
  - Can be linked to C / C++, JAVA, SQL, etc.
- Widely used in engineering industry and academia, especially at Purdue and aerospace industry
- Can do much more than just math!
  - Wide variety of toolboxes and functions available

MATLAB Environment

(R2012a or Earlier)



Current Folder: Z:\Users\Isaac\Documents\Matlab Work\Purdue School Work

- AAE508
- AAE550
- AAE564
- AAE575
- AAE590K
- AAE666
- ENGR132
- IE537
- IE639
- ME581
- STAT529

```
>>
```

```
game_Main
sledGUI_sec002_team08
%-- 7/31/2013 12:22 PM --%
game_Main
sledGUI_sec002_team08
clc
clear
cd ..
clear
%-- 8/16/2013 2:49 PM --%
```

Working Path
Where you are

Command Window
Where the magic happens

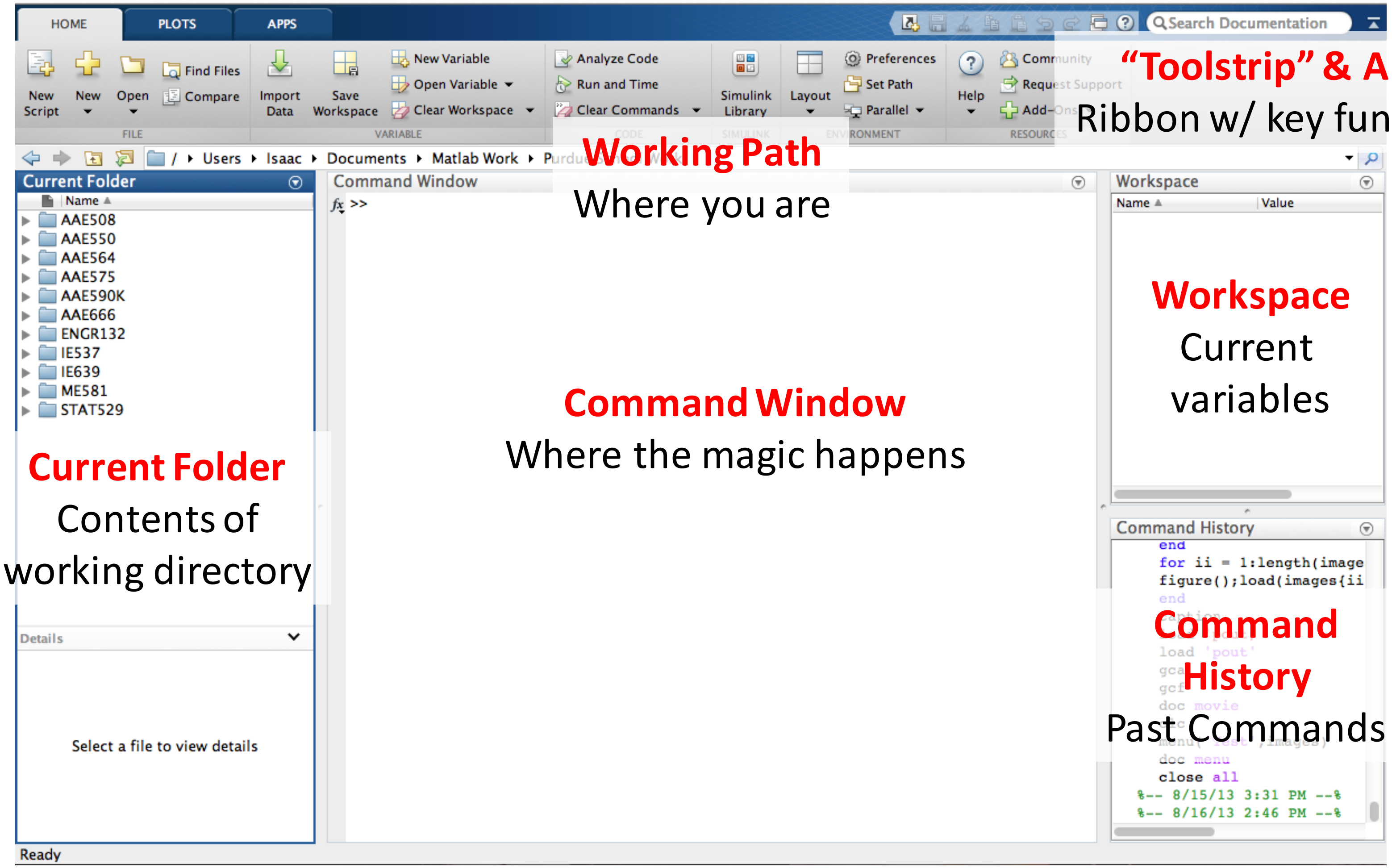
Workspace
Current variables

Current Folder
Contents of working directory

Command History
Past Commands

MATLAB Environment

(R2012b or Later)



```
end
for ii = 1:length(image
figure();load(images{ii
end
caption
load 'pout'
gca
gcf
doc movie
menu('Test',images)
doc menu
close all
%-- 8/15/13 3:31 PM --%
%-- 8/16/13 2:46 PM --%
```

"Toolstrip" & Apps

Ribbon w/ key functions

Working Path

Where you are

Workspace

Current variables

Command Window

Where the magic happens

Current Folder

Contents of working directory

Command History

Past Commands

Variables

- Do not have to be previously declared and can take any type (and switch that type)
  - **Types:** logical, char, numeric, cell, structure, function handles
- Variable names can contain up to 63 characters
  - **Must** start with a letter and can be followed by letters, digits, and underscores
- Variable (and function) names are case sensitive
  - X and x are *two* different variables

Pre-Defined Variables

PURDUE™ AERONAUTICS & ASTRONAUTICS

- MATLAB has several pre-defined / reserved variables
  - **Beware:** These variables can be overwritten with custom values!

`ans`	Default variable name for results
`pi`	Value of π
`eps`	Smallest incremental number (2.2204e-16)
`Inf` / **`inf`**	Infinity
`NaN` / **`nan`**	Not a number (e.g., 0/0)
`realmin`	Smallest usable positive real number (2.2251e-308)
`realmax`	Largest usable positive real number (1.7977e+308)
`i` / **`j`**	Square root of (-1)

Assignment and Operators

Assignment (assign b to a)	`=`	`a = b`
Addition	`+`	`a + b`
Subtraction	`-`	`a - b`
Multiplication: Matrix	`*`	`a * b`
Multiplication: Element-by-Element	`.*`	`a .* b`
Division: Matrix	`/`	`a / b`
Division: Element-by-Element	`./`	`a ./ b`
Power: Matrix	`^`	`a ^ b`
Power: Element-by-Element	`.^`	`a .^ b`

Matrices

- MATLAB treats all variables as matrices
  - For our purposes, a matrix can be thought of as an array, in fact, that is how it is stored
- Vectors are special forms of matrices and contain only **one row** or **one column**
- Scalars are matrices with only one row **and** one column
- Matrices are described as rows-by-columns
  - A 3 × 5 matrix as 3 rows and 5 columns

Matrices

- Columns are separated by spaces or commas (,)
- Rows are separated by semicolons (;)
- White space between numbers has no effect
  - [1,2,3] is the same as [1,   2   , 3]

```
row_vector = [1, 2, 3, 4,] or [1 2 3 4]
col_vector = [5; 6; 7; 8]
matrix = [1, 2, 3; 4, 5, 6; 7, 8, 9]
```

Extracting a Sub-Matrix

A portion of a matrix can be extracted and stored in a smaller matrix by specifying the names of both the rows and columns to extract

```
sub_matrix = matrix(r1:r2 , c1:c2)
sub_matrix = matrix(rows , columns)
```

Where **r1** and **r2** specify the beginning and ending rows, and **c1** and **r2** specify the beginning and ending columns to extract

Colon Operator

The colon operator helps to specify ranges

`a:b`	Goes from **a** to **b** in increments of 1. If **a** > **b**, results in null vector
`a:n:b`	Goes from **a** to **b** in increments of **n**. If **n** < 0 then **a** > **b**
`A(:, b)`	The **b**th column of **A**
`A(a, :)`	The **a**th row of **A**
`A(:, :)`	All of the rows and columns of **A** (i.e., the **A** matrix)
`A(a:b)`	Elements **a** to **b** (in increments of 1) of A. **NOTE:** Elements are counted down the columns and then across the rows!
`A(:, a:b)`	All rows and columns **a** to **b** (in increments of 1)
`A(:)`	All elements of **A** in a single column vector

Matrices

- Accessing single elements of a matrix:

  `A(a,b)` → Element in row a and column b

- Accessing multiple elements of a matrix:

  `A(1,4) + A(2,4) + A(3,4) + A(4,4)`

  `sum(A(1:4,4))` or **`sum(A(:,end))`**

  - In locations, the keyword **`end`** refers to the *last* row or column

- Deleting rows and columns:

  `A(:,2) = []` → Deletes the second column of A

- Concatenating matrices A and B:

  `C = [A ; B]` for vertical concatenation

  `C = [A , B]` for horizontal concatenation

Matrix Functions in MATLAB

`A = ones(m,n)`	Creates an m×n matrix of 1’s
`A = zeros(n,m)`	Creates an m×n matrix of 0’s
`A = eye(n)`	Creates an n×n identity matrix
`A = NaN(m,n)`	Creates an m×n matrix of NaN’s
`A = inf(m,n)`	Creates an m×n matrix of inf’s
`A = diag(x)`	Creates a diagonal matrix A of x *or*
`x = diag(A)`	Extracts diagonal elements from A
`[m,n] = size(A)`	Returns the dimensions of A
`n = length(A)`	Returns the largest dimension of A
`n = numel(A)`	Returns number of elements of A

Matrix Functions in MATLAB

`x = sum(A)`	Vector with sum of columns
`x = prod(A)`	Vector with product of columns
`B = A'`	Transposed matrix
`d = det(A)`	Determinant
`[x,y] = eig(A)`	Eigenvalues and eigenvectors
`B = inv(A)`	Inverse of square matrix
`B = pinv(A)`	Moore-Penrose pseudoinverse
`B = chol(A)`	Cholesky decomposition
`[Q,R] = qr(A)`	QR decomposition
`[U,D,V] = svd(A)`	Singular value decomposition

Logic in Matrices

`B = any(A)` Determine if any elements in each column of A are nonzero

`B = all(A)` Determine if all elements in each column of A are nonzero

`B = find(A)` Find indices of all non-zero elements of A

Can also use logic!

`B = find(A>4 & A<5)` Elements > 4 **and** < 5

`B = all(A~=9)` Elements **not equal** to 9

`B = any(A==3 | A==5)` Elements **equal** to 3 **or** 5

PLOTTING IN MATLAB

Plotting in MATLAB

- MATLAB has extensive plotting capabilities
- Basic function is `plot` to plot one vector vs. another vector (vectors must have same length)

```
plot(x, y)
```

- Can also simply plot one vector vs. its index

```
plot(x)
```

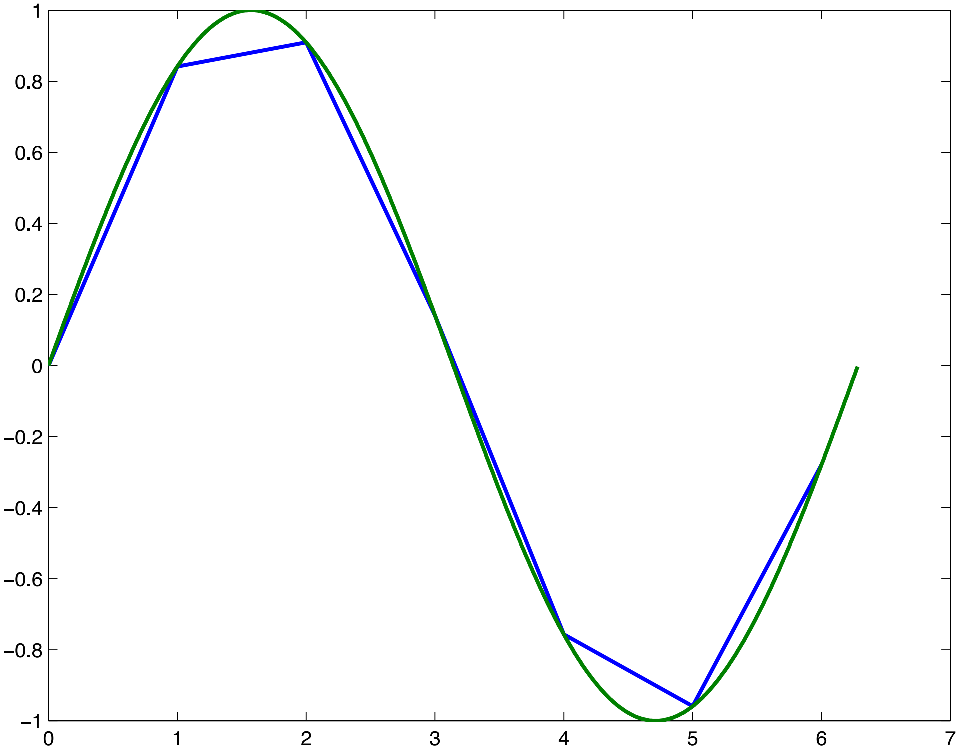
- Repeat three arguments to plot multiple vectors
  - Different pairs of x and y data can have different sizes!

```
plot(x1, y1, x2, y2, x3, y3)
```

Plotting in MATLAB

```
>> x1 = 0:1:2*pi;
>> y1 = sin(x1);
>> x2 = 0:0.01:2*pi;
>> y2 = sin(x2);
>> plot(x1,y1,x2,y2)
```

MATLAB will automatically change the colors of the lines if plotted with one plot command!

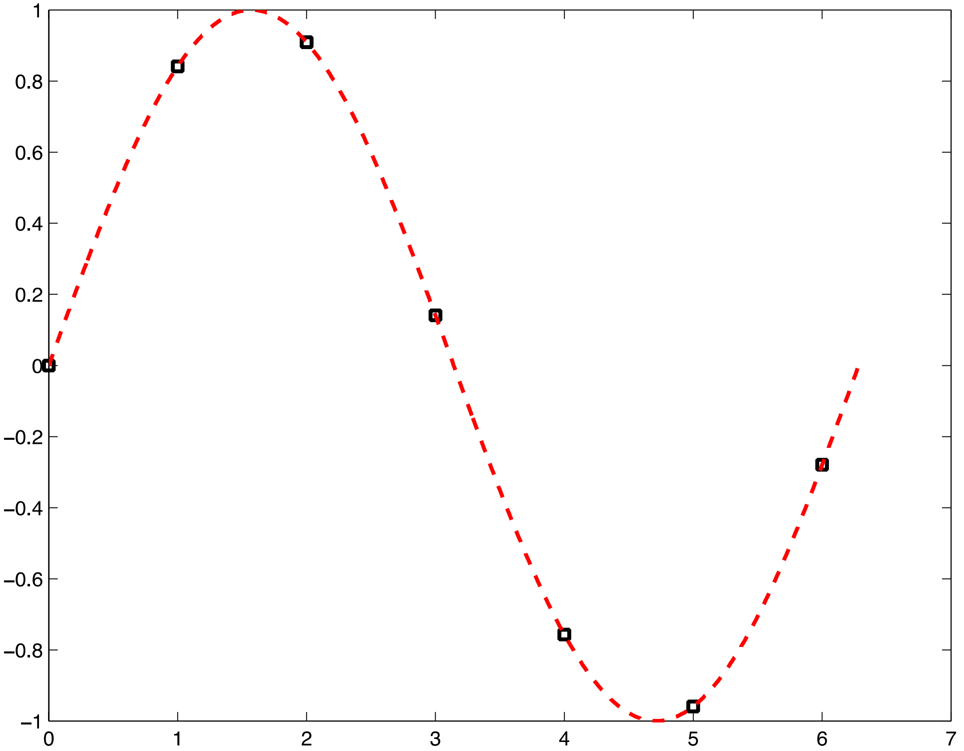


Plotting in MATLAB

- The line style, marker symbol, and color of the plot is specified by the **`LineSpec`**
- **`LineSpec`** is specified for each line after the y data and is optional
- To see all options in MATLAB: **`doc LineSpec`**
- Common formatting:
  - **Lines:** **`'-'`** solid, **`'--'`** dashed, **`':'`** dotted, **`'-.'`** dash-dot
  - **Markers:** **`'+'`** plus, **`'o'`** circle, **`'.'`** point, **`'s'`** square, **`'d'`** diamond, **`'x'`** cross, and more!
  - **Colors:** **`'r'`** red, **`'g'`** green, **`'b'`** blue, **`'k'`** black, **`'y'`** yellow, **`'c'`** cyan, **`'m'`** magenta

Plotting in MATLAB

```
>> plot(x1,y1,'ks',x2,y2,'r--')
```



Plotting in MATLAB

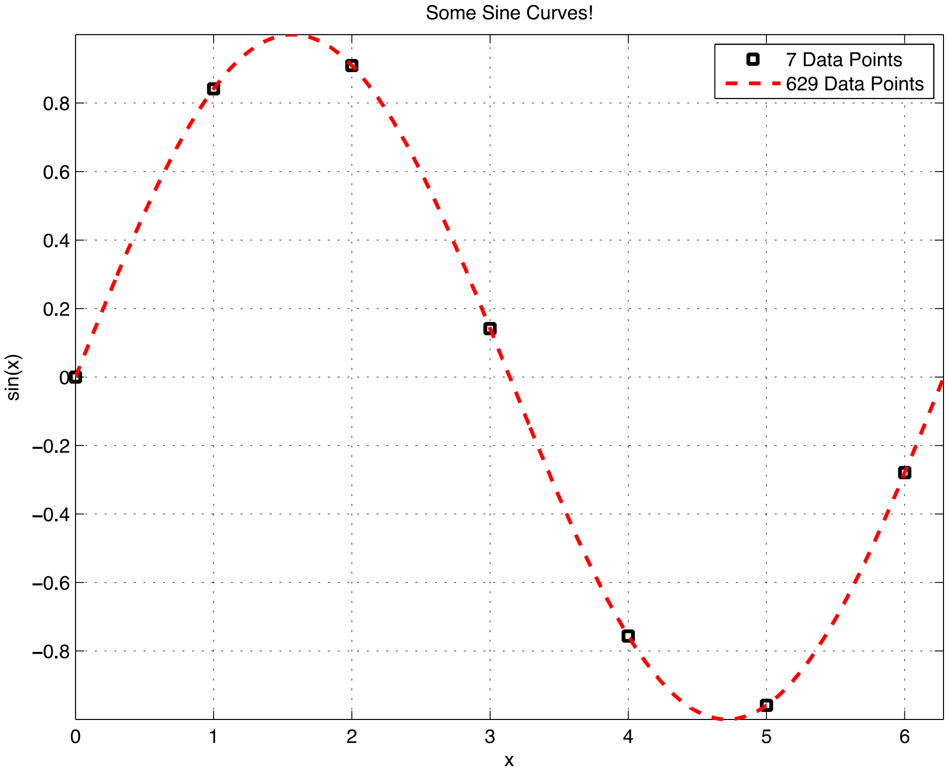
- Other commands allow you to modify the plot
  - **Annotation**: `title`, `xlabel`, `ylabel`, `zlabel`
  - **Grid**: `grid on`, `grid off`, `grid minor`
  - **Axes**: `axis([xmin xmax ymin ymax])`, `axis keyword` (`doc axis` for full keyword list)
  - **Legend:** `legend('Line 1','Line 2','Location','Position')`
- Another way to plot multiple lines is with the `hold` command

```
hold on
plot(x1,y1)
plot(x2,y2)
hold off
```

- Unless a new figure is created using `figure()`, any plotting function will overwrite the current plot

Plotting in MATLAB

```
>> plot(x1,y1,'sk',x2,y2,'r--')
>> legend('7 Data Points','629 Data Points','Location','NorthEast')
>> title('Some Sine Curves!')
>> xlabel('x')
>> ylabel('sin(x)')
>> grid on
>> axis tight
```



Plotting in MATLAB

- Subplot function in MATLAB
  - **`subplot(m,n,p)`**
- Functionality
  - Breaks the figure into an **m** (rows) by **n** (cols) grid, and places the plot in location **p** (counts across rows first)
  - Plot can span across multiple locations by setting **p** as a *vector* → **`subplot(2, 3, [2 5])`**
  - Set the subplot location with subplot command, then use normal plotting commands (**`plot`**, **`hist`**, **`surf`**, etc.)
- Title Over ALL Subplots
- Use command **`suptitle('Title Text')`**
  - **`suptitle`** must be **LAST** command of entire subplot

Plotting in MATLAB

- Other plotting functions in MATLAB
  - **Log scales:** semilogx, semilogy, loglog
  - **Two y-axes scales:** plotyy
  - **3D line plots:** plot3
  - **Surface and mesh plots:** surf, surfc, mesh, meshc, waterfall, ribbon, trisurf, trimesh
  - **Histograms:** hist, histc, area, pareto
  - **Bar plots:** bar, bar3, barh, bar3h
  - **Pie charts:** pie, pie3, rose
  - **Discrete data:** stem, stem3, stairs, scatter, scatter3, spy, plotmatrix
  - **Polar plots:** polar, rose, compass
  - **Contour plots:** contour, contourf, contourc, contour3, contourslice
  - **Vector fields:** feather, quiver, quiver3, compass, streamslice, streamline

PROGRAMMING IN MATLAB

Programming in MATLAB

- Elements of MATLAB as a programming language:
  - Expressions
  - Flow Control Blocks
    - Conditional
    - Iterations (Loops)
  - Scripts
  - Functions
  - Objects and classes (not covered here)
- Be mindful of existing variables and function names!
  - Creating a variable or function that is already used by MATLAB will cause troubles and errors!
  - Example: Saving a variable as **`sin = 10`** will prevent you from using the sine function! Use something more descriptive such as **`sin_x = 10`**

Relational Operators

- MATLAB has six relational Operators

– Less Than	**<**
– Less Than or Equal	**<=**
– Greater Than	**>**
– Greater Than or Equal	**>=**
– Equal To	**==**
– Not Equal To	**~=**

- Relational operators can be used to compare scalars to scalars, scalars to matrices/vectors, or matrices/vectors to matrices/vectors of the same size
- Relational operators to precedence after addition / subtraction

Logical Operators

- MATLAB supports four logical operators
  - Not `~`
  - And `&` or `&&`
  - Or `|` or `||`
  - Exclusive Or (xor) `xor()`
- Not has the highest precedence and is evaluated after parentheses and exponents
- And, or, xor have lowest precedence and are evaluated last

Conditional Structures

- If / Then Structure

```
if expression
    commands
end
```

- Example

```
if (x > 4) && (y < 10)
    z = x + y;
end
```

- If / Else Structure

```
if expression
    commands
else
    commands
end
```

- Example

```
if (x > 4) && (y < 10)
    z = x + y;
else
    z = x * y;
end
```

Conditional Structures

- If / Elseif / Else Structure

```
if expression
    commands
elseif expression
    commands
else
    commands
end
```

- Example

```
if (x > 4) && (y < 10)
    z = x + y;
elseif (x < 3)
    z = 10 * x;
elseif (y > 12)
    z = 5 / y;
else
    z = x * y;
end
```

Conditional Structures

- Conditional Structures can be nested inside each other

```
if (x > 3)
  if (y > 5)
    z = x + y;
  elseif (y < 5)
    z = x - y;
  end
elseif (y < 10)
  z = x * y;
else
  z = x / y;
end
```

- MATLAB will auto-indent for you, but indentation is not required

Conditional Structures

- Switch / Case / Otherwise function used if known cases of a variable will exist
  - Used in place of If / Elseif / Else structure
- Syntax

```
switch switch_expression
   case case_expression
       statements
   case case_expression
       statements
   otherwise
       statements
end
```

Conditional Structures

if - elseif - else

```
if x == 1
    z = 5;
elseif x == 2
    z = 4;
elseif x == 3
    z = 3;
elseif (x == 4) || (x == 5)
    z = 2;
else
    z = 1;
end
```

switch - case - otherwise

```
switch x
    case 1
        z = 5;
    case 2
        z = 4;
    case 3
        z = 3;
    case {4 , 5}
        z = 2;
    otherwise
        z = 1;
end
```

MATLAB Iteration Structures

- Definite looping structures (**for**)

```
for var = expression
    commands
end
```

- Can also nest loops!
  - Can mix for / while loops

- Example

```
for ii = 1:1:25
    A(ii) = [ii, ii^2];
end
```

- Nested For Loop Example

```
for ii = 1:1:25
    for jj = [1 3 5 6]
        A(ii) = ii*jj;
    end
end
```

MATLAB Iteration Structures

- Indefinite looping structures (**while**)

```
while expression
    commands
end
```

- You need to make sure the variable in the while loop expression is changed during the loop!
  - May lead to an infinite loop!

- Example

```
x = 0; y = 0;
while x < 10
    y = y + x;
    x = x + 1;
end
```

- Infinite Loop

```
x = 0;
while x < 10
    y = x;
end
```

M-Files

- Text files containing MATLAB programs
  - Can be called from the command line or from other M-Files
- Contain “`.m`” file extension
- Two main types of M-Files
  - Scripts
  - Functions
- Comment character is `%`
  - `%` will comment out rest of line

M-Files – Scripts

- Scripts are simply M-Files with a set of commands to run
  - Do not require input values or have output values
  - Execute commands similarly to how they would be done if typed into the command window
- To create new M-File:
  - `>> edit filename`
  - Ctrl + N or ⌘ + N
  - Select New → Script from Menu
- To run M-File:
  - `>> filename`

M-Files – Scripts

```
>> edit demoPlot
```

```
% This Script Makes a Demo Plot!
%   Isaac Tetzloff - Aug 2014

figure()                        % New Figure
x1 = 0:1:2*pi; y1 = sin(x1);    % First Data Set
x2 = 0:0.01:2*pi; y2 = sin(x2); % Second Data Set
plot(x1,y1,'sk',x2,y2,'r--')    % Make Plot
title('Some Sine Curves!')      % Add Title, Labels, Legend, etc.
xlabel('x')
ylabel('sin(x)')
legend('7 Data Points','629 Data Points','Location','NorthEast')
grid on
axis tight
```

```
>> demoPlot
```

M-Files – Functions

- Functions typically require input or output values
- "What happens in the function, stays in the function"
  - Only variables visible *after* function executes are those variables defined as output
- Usually one file for each function defined
- Structure:

```
function [outputs] = funcName(inputs)
commands;
end
```

M-Files – Functions

```
function [outputs] = funcName(inputs)
```

- Function Definition Line Components
  1. Function keyword → Identifies M-File as a function
  2. Output Variables → Separated by commas, contained in **square brackets**
     - Output variables must match the name of variables inside the function!
  3. Function Name → Must match the name of the .m file!
  4. Input Variables → Separated by commas, contained in **parentheses**
     - Input variables must match the name of variables inside the function!
- When calling a function, you can use any name for the variable as input or output
  – The names **do not** have to match the names of the .m file

M-Files – Functions

```
function [area, perimeter] = demoFunc(base, height)

% Demo function to calculate the area and perimeter of a rectangle
% Function can handle scalar and vector inputs
%   Isaac Tetzloff - Aug 2014

area = base .* height;                 % Calculate the area
perimeter = 2 * (base + height);       % Calculate the perimeter

end
```

```
>> [a, p] = demoFunc(10, 15);           % Returns both values as a & p
>> area = demoFunc(10, 5);              % Returns area and saves as area
>> perim = demoFunc(5, 15);             % Returns area and saves as perim!
>> [perim, area] = demoFunc(5, 15);     % Saves area as perim, and vice versa!

>> x = [1 2 3]; y = [5 4 3];
>> [x, y] = demoFunc(x, y);             % Returns both and overwrites input!
```

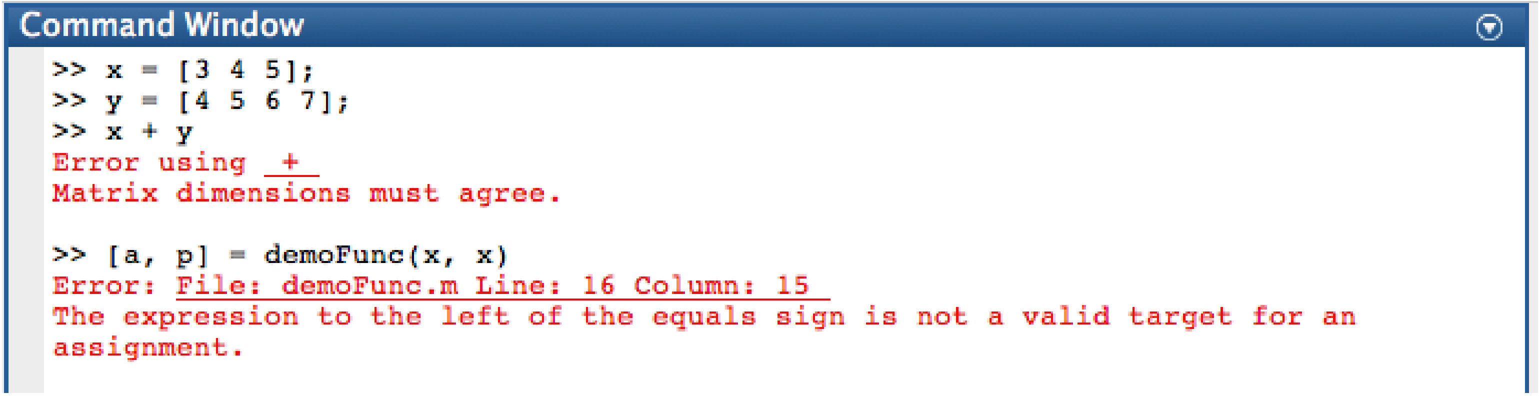
M-Files – Functions

- In modified function below, only variables output are **area** and **perimeter**
  - MATLAB and other functions will not have access to **depth**, **mult**, **add**, or **volume**!
  - **REMEMBER:** *What happens in the function stays in the function!*

```
function [area, perimeter] = demoFunc(base, height)

depth = 10;             % Assume 3D prism has depth of 10
mult = base .* height;  % Multiply base by height
add  = base + height;   % Add base and height

area = mult;            % Calculate the area
perimeter = 2 * add;    % Calculate the perimeter
volume = mult * depth;  % Calculate the volume

end
```

Debugging in MATLAB

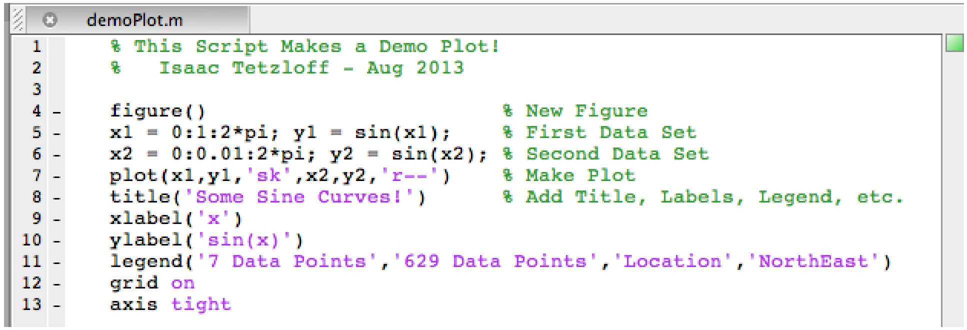
- MATLAB errors are very descriptive and provide specifics about error
  - If a function or script causes an error, MATLAB will give the line of code and file with the error



```
>> x = [3 4 5];
>> y = [4 5 6 7];
>> x + y
Error using  +
Matrix dimensions must agree.

>> [a, p] = demoFunc(x, x)
Error: File: demoFunc.m Line: 16 Column: 15
The expression to the left of the equals sign is not a valid target for an
assignment.
```

Debugging in MATLAB

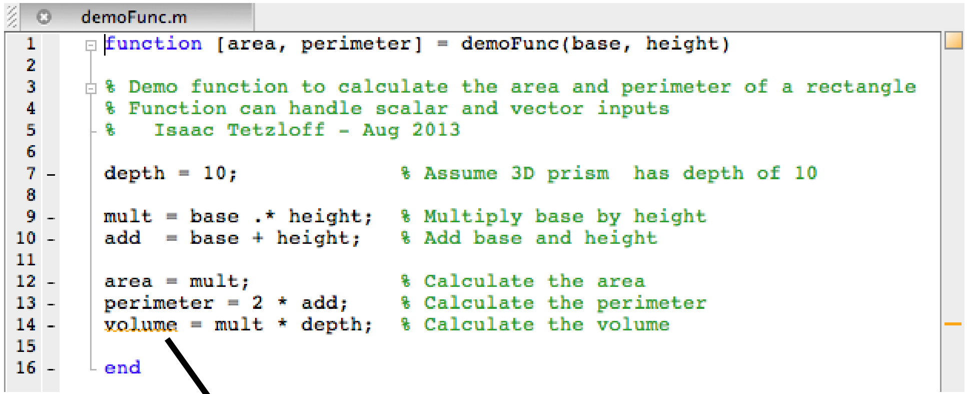
- The MATLAB Editor provides on-the-fly debugging help!



```
% This Script Makes a Demo Plot!
%   Isaac Tetzloff - Aug 2013

figure()                         % New Figure
x1 = 0:1:2*pi; y1 = sin(x1);     % First Data Set
x2 = 0:0.01:2*pi; y2 = sin(x2); % Second Data Set
plot(x1,y1,'sk',x2,y2,'r--')     % Make Plot
title('Some Sine Curves!')       % Add Title, Labels, Legend, etc.
xlabel('x')
ylabel('sin(x)')
legend('7 Data Points','629 Data Points','Location','NorthEast')
grid on
axis tight
```

Green square

No errors or warnings

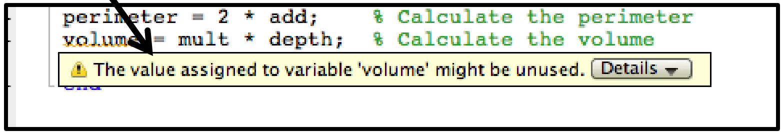


```
function [area, perimeter] = demoFunc(base, height)

% Demo function to calculate the area and perimeter of a rectangle
% Function can handle scalar and vector inputs
%   Isaac Tetzloff - Aug 2013

depth = 10;                 % Assume 3D prism  has depth of 10

mult  = base .* height;     % Multiply base by height
add   = base + height;      % Add base and height

area = mult;                % Calculate the area
perimeter = 2 * add;        % Calculate the perimeter
volume = mult * depth;      % Calculate the volume

end
```

Orange Square

Warning present, but code will **still run**

Indicated by **orange bar**



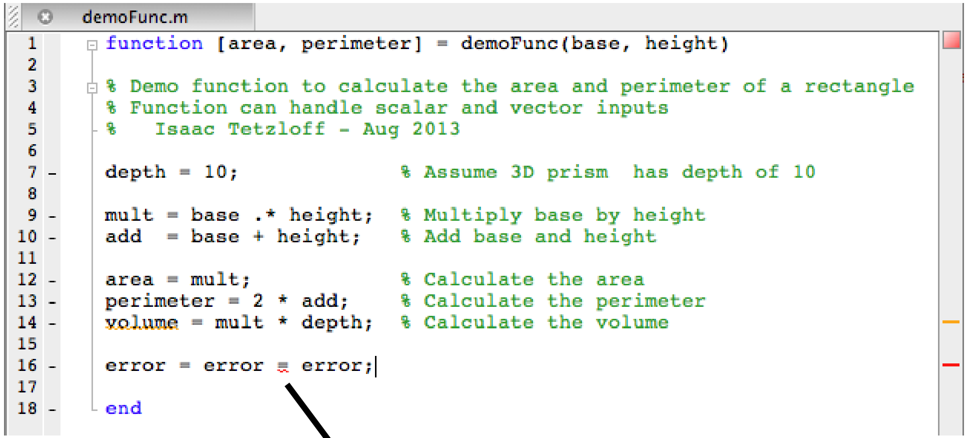
```
perimeter = 2 * add;      % Calculate the perimeter
volume = mult * depth;    % Calculate the volume
```

⚠ The value assigned to variable 'volume' might be unused.

Mouse over for warning message

Debugging in MATLAB

- The MATLAB Editor provides on-the-fly debugging help!

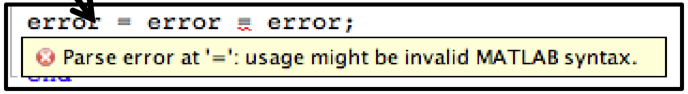


```
function [area, perimeter] = demoFunc(base, height)

% Demo function to calculate the area and perimeter of a rectangle
% Function can handle scalar and vector inputs
%   Isaac Tetzloff - Aug 2013

depth = 10;                 % Assume 3D prism  has depth of 10

mult = base .* height;      % Multiply base by height
add  = base + height;       % Add base and height

area = mult;                % Calculate the area
perimeter = 2 * add;        % Calculate the perimeter
volume = mult * depth;      % Calculate the volume

error = error = error;

end
```

<span>Red square</span>

Errors present and code **will not run!**

Indicated by **red bar**



```
error = error = error;
```

Parse error at '=': usage might be invalid MATLAB syntax.

Mouse over for error message

PURDUE™ AERONAUTICS & ASTRONAUTICS

Advanced Features to Explore

Symbolic Math

- Allows for symbolic manipulation of equations, including solving, simplifying, differentiating, etc.

Inline Functions

- Creates a workspace variable that is a simple equation

```
>> f = @(x) x^2 + 2*x + 1
>> y = f(3) → y = 16
```

Numerical Integration

- Solve differential equations / equations of motion using `ode45`, `ode23`, `ode113`, etc.

Optimization

- Solve constrained problems with `fmincon`, unconstrained with `fminunc`, bounded problems with `fminbnd`, etc.

Many Others!

- MATLAB is extremely powerful and has a lot of advanced features, too many to go through here!

Getting Help in MATLAB

- Within MATLAB:
  - Type **`help function`** to provide information about the function in the command window
  - Type **`doc function`** to open the documentation about the function
  - Type **`doc`** to pull up the documentation within MATLAB to explore
- Online
  - Documentation: http://www.mathworks.com/help/MATLAB/
  - Tutorials: http://www.mathworks.com/academia/student_center/tutorials/
  - MATLAB Primer / Getting Started with MATLAB (pdf): http://www.mathworks.com/help/pdf_doc/MATLAB/getstart.pdf