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 = MATrix LABoratory

• 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 (R2012a) Environment

Introduction to Matlab
Isaac Tetzloff - isaact@purdue.edu
Matlab (R2013a) Environment

**Current Folder**
Contents of working directory

**Command Window**
Where the magic happens

**Workspace**
Current variables

**Command History**
Past Commands

**Working Path**
Where you are

“Toolstrip” & Apps
Ribbon w/ key functions
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

Matlab has several pre-defined / reserved variables

- **Beware:** These variables can be overwritten with custom values!

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ans</td>
<td>Default variable name for results</td>
</tr>
<tr>
<td>pi</td>
<td>Value of ( \pi )</td>
</tr>
<tr>
<td>eps</td>
<td>Smallest incremental number (2.2204e-16)</td>
</tr>
<tr>
<td>Inf/inf</td>
<td>Infinity</td>
</tr>
<tr>
<td>NaN/nan</td>
<td>Not a number (e.g., 0/0)</td>
</tr>
<tr>
<td>realmin</td>
<td>Smallest usable positive real number (2.2251e-308)</td>
</tr>
<tr>
<td>realmax</td>
<td>Largest usable positive real number (1.7977e+308)</td>
</tr>
<tr>
<td>i/j</td>
<td>Square root of (-1)</td>
</tr>
</tbody>
</table>
## Assignment and Operators

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>-operator</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment (assign b to a)</td>
<td>=</td>
<td>a = b</td>
</tr>
<tr>
<td>Addition</td>
<td>+</td>
<td>a + b</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>a - b</td>
</tr>
<tr>
<td>Multiplication: Matrix</td>
<td>*</td>
<td>a * b</td>
</tr>
<tr>
<td>Multiplication: Element-by-Element</td>
<td>.*</td>
<td>a .* b</td>
</tr>
<tr>
<td>Division: Matrix</td>
<td>/</td>
<td>a / b</td>
</tr>
<tr>
<td>Division: Element-by-Element</td>
<td>./</td>
<td>a ./ b</td>
</tr>
<tr>
<td>Power: Matrix</td>
<td>^</td>
<td>a ^ b</td>
</tr>
<tr>
<td>Power: Element-by-Element</td>
<td>.^</td>
<td>a .^ b</td>
</tr>
</tbody>
</table>
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 $3 \times 5$ 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]
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

\[
\text{sub\_matrix} = \text{matrix}(r1:r2, c1:c2)
\]
\[
\text{sub\_matrix} = \text{matrix}(\text{rows}, \text{columns})
\]

Where \(r1\) and \(r2\) specify the beginning and ending rows, and \(c1\) and \(r2\) specify the beginning and ending columns to extract
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:
  \[ A(a,b) \rightarrow \text{Element in row } a \text{ and column } b \]

• Accessing multiple elements of a matrix:
  \[ A(1,4) + A(2,4) + A(3,4) + A(4,4) \]
  \[ \text{sum}(A(1:4,4)) \text{ or } \text{sum}(A(:,\text{end})) \]
  – In locations, the keyword \text{end} refers to the last row or column

• Deleting rows and columns:
  \[ A(:,2) = [] \rightarrow \text{Deletes the second column of } A \]

• Concatenating matrices A and B:
  \[ C = [A ; B] \text{ for vertical concatenation} \]
  \[ C = [A , B] \text{ for horizontal concatenation} \]
Matrix Functions in Matlab

\[ A = \text{ones}(m,n) \] Creates an \( m \times n \) matrix of 1’s
\[ A = \text{zeros}(n,m) \] Creates an \( m \times n \) matrix of 0’s
\[ A = \text{eye}(n) \] Creates an \( n \times n \) identity matrix
\[ A = \text{NaN}(m,n) \] Creates an \( m \times n \) matrix of NaN’s
\[ A = \text{inf}(m,n) \] Creates an \( m \times n \) matrix of inf’s
\[ A = \text{diag}(x) \] Creates a diagonal matrix \( A \) of \( x \) or
\[ x = \text{diag}(A) \] Extracts diagonal elements from \( A \)
\[ [m,n] = \text{size}(A) \] Returns the dimensions of \( A \)
\[ n = \text{length}(A) \] Returns the largest dimension of \( A \)
\[ n = \text{numel}(A) \] Returns number of elements of \( A \)
# Matrix Functions in Matlab

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x = sum(A)</code></td>
<td>Vector with sum of columns</td>
</tr>
<tr>
<td><code>x = prod(A)</code></td>
<td>Vector with product of columns</td>
</tr>
<tr>
<td><code>B = A'</code></td>
<td>Transposed matrix</td>
</tr>
<tr>
<td><code>d = det(A)</code></td>
<td>Determinant</td>
</tr>
<tr>
<td><code>[x,y] = eig(A)</code></td>
<td>Eigenvalues and eigenvectors</td>
</tr>
<tr>
<td><code>B = inv(A)</code></td>
<td>Inverse of square matrix</td>
</tr>
<tr>
<td><code>B = pinv(A)</code></td>
<td>Moore-Penrose pseudoinverse</td>
</tr>
<tr>
<td><code>B = chol(A)</code></td>
<td>Cholesky decomposition</td>
</tr>
<tr>
<td><code>[Q,R] = qr(A)</code></td>
<td>QR decomposition</td>
</tr>
<tr>
<td><code>[U,D,V] = svd(A)</code></td>
<td>Singular value decomposition</td>
</tr>
</tbody>
</table>
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)
  
  \[
  \text{plot}(x, y)
  \]

- Can also simply plot one vector vs. its index
  
  \[
  \text{plot}(x)
  \]

- Repeat three arguments to plot multiple vectors
  - Different pairs of x and y data can have different sizes!
    
    \[
    \text{plot}(x1, y1, x2, y2, x3, y3)
    \]
Plotting in Matlab

\[
\begin{align*}
\text{>> } & \ x1 = 0:1:2*\pi; \\
\text{>> } & \ y1 = \sin(x1); \\
\text{>> } & \ x2 = 0:0.01:2*\pi; \\
\text{>> } & \ y2 = \sin(x2); \\
\text{>> } & \text{plot(x1,y1,x2,y2)}
\end{align*}
\]

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
  - **Colors**: 'r' red, 'g' green, 'b' blue, 'k' black, 'y' yellow, 'c' cyan, 'm' magenta
Plotting in Matlab

```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
  ```matlab
  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

```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
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 $\text{sin} = 10$ will prevent you from using the sine function! Use something more descriptive such as $\text{sin}_x = 10$
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

• If / Else Structure
  
  if expression
  commands
  else
  commands
  end

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

• Example
  
  if (x > 4) && (y < 10)
    z = x + y;
  else
    z = x * y;
  end
Conditional Structures

• If / Elseif / Else Structure

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

• Example

```matlab
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

  ```matlab
  switch switch_expression
  case case_expression
    statements
  case case_expression
    statements
  otherwise
    statements
  end
  ```
### Conditional Structures

<table>
<thead>
<tr>
<th>if - elseif - else</th>
<th>switch - case - otherwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>if x == 1</td>
<td>switch x</td>
</tr>
<tr>
<td>z = 5;</td>
<td>case 1</td>
</tr>
<tr>
<td>elseif x == 2</td>
<td>z = 5;</td>
</tr>
<tr>
<td>z = 4;</td>
<td>case 2</td>
</tr>
<tr>
<td>elseif x == 3</td>
<td>z = 4;</td>
</tr>
<tr>
<td>z = 3;</td>
<td>case 3</td>
</tr>
<tr>
<td>elseif (x == 4)</td>
<td></td>
</tr>
<tr>
<td>z = 2;</td>
<td>case {4 , 5}</td>
</tr>
<tr>
<td>else</td>
<td>z = 2;</td>
</tr>
<tr>
<td>z = 1;</td>
<td>otherwise</td>
</tr>
<tr>
<td>end</td>
<td>z = 1;</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>
Matlab Iteration Structures

• Definite looping structures (**for**)
  ```matlab
  for var = expression
    commands
  end
  ```

• Can also nest loops!
  – Can mix for / while loops

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

• Nested For Loop Example
  ```matlab
  for ii = 1:1:25
    for jj = [1 3 5 6]
      A(ii) = ii*jj;
    end
  end
  ```
Matlab Iteration Structures

• Indefinite looping structures (while)
  
  ```matlab
  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
  ```matlab
  x = 0; y = 0;
  while x < 10
    y = y + x;
    x = x + 1;
  end
  ```

• Infinite Loop
  ```matlab
  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
% 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

% Add Title, Labels, Legend, etc.
title('Some Sine Curves!')
xlabel('x')
ylabel('sin(x)')
legend('7 Data Points','629 Data Points','Location','NorthEast')
grid on
axis tight
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
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 \textit{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 \textit{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 \textbf{do not} have to match the names of the .m file
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

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!
• 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!

```matlab
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
Debugging in Matlab

• The Matlab Editor provides on-the-fly debugging help!

Introduction to Matlab
Isaac Tetzloff - isaact@purdue.edu

Green square
No errors or warnings

Orange Square
Warning present, but code will still run
Indicated by orange bar

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
```

- Red square
- Errors present and code **will not run!**
- Indicated by **red bar**

Mouse over for error message
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
  \[
  \begin{align*}
  &\text{>> } f = @(x) x^2 + 2*x + 1 \\
  &\text{>> } y = f(3) \rightarrow y = 16
  \end{align*}
  \]

Numerical Integration
• Solve differential equations / equations of motion using \texttt{ode45}, \texttt{ode23}, \texttt{ode113}, etc.

Optimization
• Solve constrained problems with \texttt{fmincon}, unconstrained with \texttt{fminunc}, bounded problems with \texttt{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