MATLAB BASICS
CS412 Spring 2011
Instructor: Eftychios Sifakis
To launch MATLAB from an instructional lab machine:
Enter *matlab* at the shell prompt
MATLAB desktop keyboard shortcuts, such as Ctrl+S, are now customizable. In addition, many keyboard shortcuts have changed for improved consistency across the desktop.

To customize keyboard shortcuts, use Preferences. From there, you can restore previous default settings by selecting "R2009a UNIX Default Settings" from the active settings drop-down list. For more information, see Help.

Click here if you do not want to see this message again.
Links to introductory tutorials are offered when MATLAB first launches.
Hello! It looks like this image shows a MATLAB interface. The command window is open, displaying a list of command topics and a help prompt. The prompt reads: "New to MATLAB? Watch this Video, see Demos, or read Getting Started." Below the prompt, there is a list of HELP topics grouped under MATLAB subdirectories.

Some topics include:
- General purpose commands.
- Operators and special characters.
- Programming language constructs.
- Elementary matrices and matrix manipulation.
- Elementary math functions.
- Specialized math functions.
- Matrix functions - numerical linear algebra.
- Data analysis and Fourier transforms.
- Interpolation and polynomials.
- Function functions and ODE solvers.
- Sparse matrices.
- Annotation and Plot Editing.
- Two dimensional graphics.
- Three dimensional graphics.
- Specialized graphs.
- Handle Graphics.
- Graphical User Interface Tools.
- Character strings.
- Image and scientific data input/output.
- File input and output.
- Audio and Video support.
- Time and dates.
- Data types and structures.
- Version control.

The Command Window at the bottom of the image shows the command `help` entered, indicating that the user is seeking help within the MATLAB environment.
The `help` command provides documentation for practically all MATLAB features.
Additional help is available by clicking on the subtopics
The help menu provides an additional, searchable reference.
New variables can be defined and initialized using the assignment operator “=”
MATLAB responds by evaluating the variable just defined.
Ending a statement with a semicolon (;) suppresses MATLAB’s output -- this is useful in programs.
Every variable in MATLAB is treated as a matrix. Even in this simple case, $x$ is treated as the $1 \times 1$ matrix $[x]$.
Actual vectors and matrices can be entered using the bracket notation. Here $v$ defines a row vector (or, equivalently a $1 \times 3$ matrix).
Elements of row vectors can also be entered with commas (,) between them. A matrix is entered by using commas between elements of the same row, and separates each row from the next using semicolons (;)
Column vectors can be entered either by separating elements with semicolons, or by defining a row vector and using the apostrophe (‘) to denote its transpose.
Note the command history window, which lists the last executed commands.
The variable display window lists the defined variables and its contents.
We can access components of matrices and vectors using the parenthesis operator (\()\). In the case of a column vector, an element can be accessed either with a single index, or in matrix notation (with a column index of 1).
We can use the same notation to access the element of a matrix.
If we try to read an element that is out of bounds, MATLAB reports an error.
However, if we try to write to an element out of bounds, MATLAB resizes the matrix to fit the new entry.

```
>> A=[1,2,3;4,5,6;7,8,9]
A =
     1     2     3
     4     5     6
     7     8     9
>> A(2,3)
ans =
     6
>> A(3,4)
??? Index exceeds matrix dimensions.
>> A(3,4)=10
A =
     1     2     3     0
     4     5     6     0
     7     8     9     10
```
The colon (:) is a special index, denoting all possible rows and/or columns. For example, A(:,2) means elements on the second column for all rows (i.e., the second row of A).
A(3,:) are the elements on the 3rd row, for all columns (i.e. the 3rd row of A)
A(:, :) means all rows and columns of A (i.e. the entire matrix)
The colon operator can also define a range, i.e. a vector whose elements form an arithmetic progression.
Using two colons we can define a custom step; in this case we use a step of 2.
if statements evaluate an expression (or execute a command) conditionally, based on the truth value of the expression in the parenthesis.
for statements evaluate an expression (or execute a command) for all values of a given range
if/for statements can be combined and nested.
MATLAB includes a number of built-in functions, such as square root, trigonometric functions, and exponentials/logarithms.
We can also define our own, custom functions. To do that, select File>New>Function.
This spawns an editor, which starts with an empty function template.
This is an example of a function, corresponding to the expression \( f(x) = x^2 + \sqrt{x} \).
The value(s) inside the bracket after the function keyword is the return variable of the function. When the return value has been computed, simply assign it to the respective variable.
Note that a function can return a vector or a matrix, too (we will see this later).
After the “=” sign we list the function name and the argument list (which can again be either numbers or matrices).
Additional local variables can be defined as needed.
When finished, save the function definition file with the same name as the new function, and the extension .m.
>> myfunc(4)

ans =
   18

>>
Here, we will describe 2 new user-defined functions, one of which depends on the other. \( \textit{myfunc} (x) \) returns \( x^2-5 \).
Function *bisection* takes 2 arguments \((x_1, x_2)\) and performs one step of the bisection method on the function \(myfunc\). The new shorter interval is given in the 2 return values.
```
>> x_min = 2;
>> x_max = 3;
>> [x_min x_max] = bisection(x_min, x_max)

x_min =
    2

x_max =
    2.5000
>> [x_min x_max] = bisection(x_min, x_max)

x_min =
    2

x_max =
    2.2500
```
\begin{verbatim}
>> [xmin xmax] = bisection(xmin, xmax)

xmin =
    2.1875

xmax =
    2.2500

>> [xmin xmax] = bisection(xmin, xmax)

xmin =
    2.2188

xmax =
    2.2500
\end{verbatim}
\begin{verbatim}
>> [xmin xmax] = bisection(xmin,xmax)

xmin =
2.2344

xmax =
2.2500

>> [xmin xmax] = bisection(xmin,xmax)

xmin =
2.2344

xmax =
2.2422
\end{verbatim}
Here we describe another function, which performs one step of bisection search on a general polynomial function $p(x)$.
Note the function `size(..)` which returns the length of a column vector. Also note the `if/elseif/else` structure.
function [ y1 y2 ] = bisection_poly( P, x1, x2 )
% Perform one step of bisection method on the polynomial
% p(x) = a0 + a1*x + a2*x^2 + ... + aN*x^N
% P = [a0 ; a1 ; ... ; aN] (column vector) contains the polynomial coefficients
% Returns a narrower interval containing the solution
x_midpoint=(x1+x2)/2;
f1=0;
f2=0;
f_midpoint=0;

for i=1:size(P)
    f1=f1+P(i)*x1^(i-1);
    f2=f2+P(i)*x2^(i-1);
    f_midpoint=f_midpoint+P(i)*x_midpoint^(i-1);
end

if(f_midpoint==0)
    y1=x_midpoint;
    y2=x_midpoint;
elseif(f1*f_midpoint<0)
    y1=x1;
    y2=x_midpoint;
else
    y1=x_midpoint;
    y2=x2;
end
Here we use `bisection_poly` to solve the equation $x^2 - x - 1 = 0$ (near the root $x \sim 1.618..$)
Here we use `bisection_poly` to solve the equation $x^2-x-1=0$ (near the root $x \sim 1.618..$)
Here we use *bisection_poly* to solve the equation \( x^2 - x - 1 = 0 \) (near the root \( x \sim 1.618.. \))
To generate a plot, we define a vector (say, \( x \)) of variable values, and another vector (say, \( y \)) of function values. The command \( \text{plot}(x,y) \) connects them with a line
To generate a plot, we define a vector (say, $x$) of variable values, and another vector (say, $y$) of function values. The command `plot(x,y)` connects them with a line.
If we decrease the step size in defining $x$ and $y$, the resulting plot will be smoother, instead of a jagged curve of line segments.
If we decrease the step size in defining \( x \) and \( y \), the resulting plot will be smoother, instead of a jagged curve of line segments.
We can also draw several curves on the same plot by calling `plot(x1,y1,x2,y2 ... xN,yN)`. Note the use of operator `.^` to denote raising to a power on an element-by-element basis.
We can also draw several curves on the same plot by calling `plot(x1,y1,x2,y2 ... xN,yN)`. Note the use of operator `.^` to denote raising to a power on an element-by-element basis.
We can output the current plot to a Postscript or PDF file, using the commands

```
print -dps <filename>
```
or

```
print -dpdf <filename>
```