1.0 MATLAB HELP

1.1 General
Enter help at the MATLAB prompt. A list of “toolboxes” which contain different commands will be listed. Then type help toolbox_name (for example, help matlab\general or help general).

1.2 Specific Commands
To get help on a specific topic, type help topic_name. The MATLAB commands are very intuitive. For example, sqrt(x) is the command for square root and exp(x) is the command for an exponential. To get complete syntax and use for the square root operator, for example, type help sqrt.

1.3 Interactive Help Window
Type helpwin to bring up an interactive help window. Toolboxes and individual commands can be selected and viewed by clicking on them.

2.0 WORKSPACE MANAGEMENT

2.1 The Command Prompt >>

2.1.1 Basic Concept
At the command prompt, any valid MATLAB command or mathematical expression can be executed. If a calculation is executed without assigning the result to a variable name, the result will be stored in a variable named ans. If a variable assignment is made, this variable will remain in the MATLAB workspace memory. It can then be used or changed as needed.

\[
\begin{align*}
8 + 3 & \Rightarrow \text{ans} = 11, \text{ans remains in the workspace} \\
x = 8 + 3 & \Rightarrow x \text{ is a scalar with a value of 11, x is now in the workspace} \\
y = [x \ 5] & \Rightarrow y \text{ is a 1x2 vector with a value of } [11 \ 5] \\
x = x + 2 & \Rightarrow x \text{ now has the value 13}
\end{align*}
\]

2.1.2 Miscellaneous
- MATLAB is case sensitive. ‘theta’ is not the same as ‘THETA’.
- Type a variable name to see its value
- who shows all variables currently in the workspace
- whos variable_name shows the variable name, size, and data type
- whos shows all variables currently in the workspace along with their size and data type
- Use a semicolon (;) after a statement to suppress statement result to the screen
- Multiple commands can be given on the same line. Separate the commands with a comma (to see result of the statement), or a semicolon (to execute statement without echoing result to the screen)
- clear var1 var2 removes var1 and var2 from the workspace
- clear removes all variables from the workspace
• **clc** clears the command window (does not affect variables in workspace)
• **type filename** displays the contents of the m-file `filename.m`

### 2.2 Variable Scope

All variables in the workspace (those which show up when using the `whos` command) are visible and available for use at the command prompt, m-files, and most blocks in SIMULINK.

If an m-file is defined as a function (see Section 10), the variables within the function are local to that module. That is, those variables are not part of the MATLAB workspace memory. Also, variables in the workspace are not visible to the function. All input/output for the function must occur through the call arguments.

### 2.3 Saving and Loading Workspace Variables

It is possible to save the current workspace variables (their definitions and their values) for future use. Workspace variables are saved in *.mat files. These files have MATLAB-specific format and are not readable by a text editor.

To save workspace variables:

```
save filename var1 var2 ...  (saves var1, var2, ...)
save filename                (saves all variables currently in the workspace)
```

Note, do not include the .mat extension on the filename.

To load the workspace variables saved in a *.mat file:

```
load filename
```

Note, do not include the .mat extension on the filename.

### 3.0 m-FILES

#### 3.1 General

3.1.1 Description

m-files are simply a collection of command prompt statements. Because they are saved in a file and can have a logical flow, they look like a program or subroutine. However, when they are run, it is exactly equivalent to entering each line of the m-file at the command prompt. Any command that can be issued at the command prompt can be executed in an m-file.

3.1.2 Editing/Saving an m-file

Start a new m-file by entering the MATLAB editor. Edit an existing m-file by opening the file with the MATLAB editor. When saving the file, avoid using spaces and symbols in the file name. The file will be saved with a *.m extension.
### 3.1.3 Running an m-file

To execute an m-file, simply enter the filename (without the *.m extension) at the command prompt. m-files can also be executed from within other m-files in the same manner. Note that m-files use variables defined in the workspace, and any variable assignments made within the m-file will become part of the workspace memory. Be sure the m-file being executed is in the current directory (or in a defined path, see `help path`).

### 3.2 Functions

An m-file named `func_name.m` can be created as a user-defined function by making a statement of the following form at the top of the m-file:

```
function [xout, yout, ...] = func_name(xin, yin, ...)
```

where `xout, yout,...` are output variables which are assigned somewhere in the function; and `xin, yin,...` are input arguments for the function. Note, variables in the MATLAB workspace are not visible to the function, and internal function variables are not visible outside of the function.

The following example function calculates the hypotenuse of a right triangle. The file name is `hypot.m`:

```
function h = hypot(a,b);
    h = sqrt(a^2 + b^2);
```

To use the function, simply enter the function name with its arguments:

```
check = hypot(3,4);
```

The result is: `check = 5`

All function output variables are passed according to the variable list defined in the function name declaration at the top of the function. Variables are passed by position in the definition list, not by name. For example, consider a function which calculates the parameters of the standard atmosphere given an altitude input. The function definition might look like:

```
function [T, P, rho] = atmos(h)
```

a call to the function (retrieving all the output variables) might look like:

```
[temp, pressure, density] = atmos(10000)
```

the result would be:

```
temp = 483
pressure = 1440
density = 1.7533e-3
```
If the following call to the function is made:

```matlab
pressure = atmos(10000)
```

the result would be:

```matlab
pressure = 483
```

(which is not the answer the caller wants)

Note, the numerical value passed back is the value of the first variable in the argument list. Since only one output variable is requested in the function call, MATLAB returns the first variable in the argument list. In order to get the value of pressure, at least two output variables must be specified in the function call, and pressure will be the one in the second position (according to the function definition statement).

3.3 When to use an m-file versus a function

Functions provide useful stand-alone, modular code which can be used in future applications to perform a well-defined procedure. The strength of a function is that its variables are not visible to the workspace. This means the variables within the function will not affect any variables already defined in the parent routine or workspace. An m-file, on the other hand, will interact directly with the workspace. This means it will overwrite any variable in the workspace with the same name as a variable in the m-file. So, if a modular procedure which can be used without fear of affecting workspace variables is needed, the function is preferred. A difficulty with the function, however, is that all inputs and outputs must be specified. This can sometimes become cumbersome, particularly when there are a large number of calling arguments.

The m-file requires no calling arguments. This makes it a preferred approach for simple and quick applications, particularly ones which are batch-like. A difficulty with the m-file is that special care must be taken to keep track of variables since they will all end up in the workspace. This is typically not too much of a difficulty until an m-file procedure is used in a new application. Then, every variable in the m-file must be checked to make sure it will not conflict with variables in the parent routine.

3.4 Program Comments, Documentation, and Format

It is good programming practice to put comment statements throughout an m-file or function. Use the `%` symbol to begin a comment. Comments can occur on an individual line or at the end of an executable statement.

Use indentation when formatting blocks of program structures such as if blocks, for loops, and while loops. When a block includes nested structures, use additional indentation.

It is also good practice to include a header at the top of every m-file or function, as illustrated below.
3.4.1 Header Guidelines

% ************************************************************************
% routine_name.m
% Brief description of the routine. Include assumptions made and
% limitations of the procedure.
% Author: , date
% Input Variables:
% variable_name = description (units)
% Output Variables:
% variable_name = description (units)
% Constants:
% variable_name = description (units)
% Local Variables:
% variable_name = description (units)
% Required Subroutines/Fuctions:
% function_name, what the function is used for
% Output Files:
% Filename, what is in the file
% References:
% Include full bibliographic reference here for all references.
% Equations, constants, or parameters that are not commonly known
% should be referenced. Provide enough information for someone
% completely unfamiliar with the procedure to be able to find the
% background in a reference.
% Documentation:
% Document any help you received here.
% ************************************************************************
3.4.2 Example Program Format

% This program calculates the velocity and position of an object
% dropped from an arbitrary height and writes the time history
% to an output file.
% Assumptions:
% - Constant object mass
% - Constant gravitational field
% - Constant drag coefficient
% - Sea Level atmospheric conditions (density)
% - Object is dropped with zero initial velocity
% Author: Scott Wells, Spring 2003

% Input Variables:
% dropHeight = height above ground that the object is dropped (ft)
% m = object mass (slug)

% Output Variables:
% h = current height above ground (ft)
% t = current time (s)
% v = current velocity (ft/s)

% Constants:
% g = acceleration due to gravity (ft/s^2)
% rho = atmospheric density (slug/ft^3)
% CD = drag coefficient (unitless)

% Local Variables:
% dt = integration time step (s)
% a = current acceleration (ft/s^2)

% Required Subroutines/Functions:
% None

% Output Files:
% drop.txt - table of velocity and position versus time

% References:

% Documentation:
% None

% Initialize physical constants
% Initialize iteration parameters
% Prompt user for object mass and initial height
%  m = input('Input object mass (slug): ')
%  dropHeight = input('Input drop height (ft): ')
%  h = dropHeight;

% Open output file and write headers
myFile=fopen('drop.txt','w');                         % Open file for write
fprintf(myFile,' Time      Height     Velocity\r');   % Write Header
fprintf(myFile,' (sec)      (ft)        (ft/s)\r');   % Write Header
fprintf(myFile, '%6.1f     %6.2f      %6.2f\r', t, h, v);  % Write initial values
% Start iteration loop
while h > 0                           % Loop until height is zero
  t = t + dt;                       % Increment time

  % Use F = ma to find the current acceleration
  a = (m*g - CD*0.5*rho*v^2)/m;     % Acceleration (ft/s^2)

  % Update the current velocity and height
  v = v + a*dt;                     % Velocity (ft/s)
  h = h - v*dt;                     % Height (ft)

% Break if time exceeds reasonable value
if t > 10000
  break
end %if

% Write results to output file
fprintf(myFile, '%6.1f     %6.2f      %6.2f', t, h, v);   % Write to file
end %while

% Close output file
fclose(myFile);

4.0 FILE HANDLING

4.1 Opening/Closing a file for input or output
To open a data file

    fid = fopen('filename.txt', permission)

where *fid* is the name of the file ID handle (and can be given any valid variable name), *filename.txt* is the text file name, and *permission* indicates the mode in which the file will be opened. Common permission values include: ‘r’ (read), and ‘w’ (write, create if necessary). See help fopen for additional syntax options. Also see the sample m-files in Section 10.

To close a data file

    fclose(fid)

where *fid* is the file ID handle assigned in an fopen statement. See the sample m-files in Section 10.

4.2 Outputting formatted data to a text file
To write data to a text file

    fprintf(fid, format, A,...)

where *fid* is the file ID assigned in an fopen statement, *format* is a string specifying the data format, and *A*... are the variables to be written. See help fprintf for documentation on syntax. Syntax is essentially the same as C language [with a key difference being that “fprintf is ‘vectorized’ for the case when A is nonscalar. The format string is recycled through the elements of A (columnwise) until all the elements are used up. It is then recycled in a similar manner through any additional matrix arguments”, ref. help fprintf]; see Section 4.4 for format syntax.
For example, consider the following statements:

```matlab
x = 0:0.1:1; y = exp(x); z=10*y; %Assign variables
fid = fopen('exp.txt','w'); %Open file for write
fprintf(fid,' x   y   z
'); %Write header
fprintf(fid,'%6.2f  %12.8f  %10.5e
', [x; y; z]); %Write vector data
fclose(fid); %Close file
```

This results in a text file `exp.txt` containing:

```
x        y         z
0.00    1.00000000  1.00000e+001
0.10    1.10517092  1.10517e+001
0.20    1.22140276  1.22140e+001
0.30    1.34985881  1.34985e+001
0.40    1.49182470  1.49182e+001
0.50    1.64872127  1.64872e+001
0.60    1.82211880  1.82212e+001
0.70    2.01375271  2.01375e+001
0.80    2.22554093  2.22554e+001
0.90    2.45960311  2.45960e+001
1.00    2.71828183  2.71828e+001
```

An alternate example to achieve the same output (in this case, 3 scalars are written multiple times):

```matlab
fid = fopen('exp.txt','w'); %Open file for write
fprintf(fid,' x   y   z
'); %Write header
for I=0:0.1:1; %Loop
    x = I; y = exp(x); z=10*y; %Assign variables
    fprintf(fid,'%6.2f  %12.8f  %10.5e
',x,y,z); %Write scalar data
end
fclose(fid); %Close file
```

4.3 **Inputting formatted data from a text file**

To read data from a text file

```
A = fscanf(fid, format, size)
```

Where `A` is a matrix containing the all the data that is read, `fid` is the file ID assigned in an `fopen` statement, `format` is a string specifying the data format, and `size` specifies number of elements to be read from the file. If `size` is omitted, the entire file is read. Valid entries for `size` are:

```
"N" read at most N elements into a column vector.
"inf" read at most to the end of the file.
[M,N] read at most M * N elements filling at least an M-by-N matrix, in column order.
N can be inf, but not M."           (Ref: help fscanf)
```
See `help fscanf` for documentation on syntax. Syntax is essentially the same as C language (with a key difference being that fscanf is vectorized).

For example, consider the text file `exp.txt` given above (Section 4.2). In order to read this data file into MATLAB and assign the vectors `x`, `y`, and `z`, the following statements can be used:

```matlab
fid = fopen('exp.txt','r'); %Open file for read
H=fscanf(fid,'%c',[1,27]); %Read Header
M=fscanf(fid,'%f',[3,inf]); %Read data
fclose(fid); %Close file
x=M(1,:); %Extract x vector from input matrix
y=M(2,:); %Extract y vector from input matrix
z=M(3,:); %Extract z vector from input matrix
```

This results in the following variables in the MATLAB workspace (note, the matrix and vectors are shown as transposes here for aesthetic reasons).

```
H = '  x           y          z'
```

```
M'' =
0  1.0000  10.0000
0.1000  1.1052  11.0517
0.2000  1.2214  12.2140
0.3000  1.3499  13.4986
0.4000  1.4918  14.9182
0.5000  1.6487  16.4872
0.6000  1.8221  18.2212
0.7000  2.0138  20.1375
0.8000  2.2255  22.2554
0.9000  2.4596  24.5960
1.0000  2.7183  27.1828
```

```
x'' = 0.5000
y'' = 1.6487
z'' = 16.4872
```

### 4.4 Some basic file I/O format syntax


The `format` argument is a string containing C language conversion specifications. A conversion specification controls the notation, alignment, significant digits, field width, and other aspects of output format. The format string can contain escape characters to represent non-printing characters such as newline characters and tabs.
Conversion specifications begin with the % character and contain these optional and required elements:

- Flags (optional)
- Width and precision fields (optional)
- A subtype specifier (optional)
- Conversion character (required)

You specify these elements in the following order:

Start of conversion specification — % — Field width — Precision

### Flags

You can control the alignment of the output using any of these optional flags.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>A minus sign (−)</td>
<td>Left-justifies the converted argument in its field.</td>
<td>%−5.2d</td>
</tr>
<tr>
<td>A plus sign (+)</td>
<td>Always prints a sign character (+ or -).</td>
<td>%+5.2d</td>
</tr>
<tr>
<td>Zero (0)</td>
<td>Pad with zeros rather than spaces.</td>
<td>%05.2d</td>
</tr>
</tbody>
</table>

### Field Width and Precision Specifications

You can control the width and precision of the output by including these options in the format string.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field width</td>
<td>A digit string specifying the minimum number of digits to be printed.</td>
<td>%6f</td>
</tr>
<tr>
<td>Precision</td>
<td>A digit string including a period (.) specifying the number of digits to be printed to the right of the decimal point.</td>
<td>%6.2f</td>
</tr>
</tbody>
</table>
## Conversion Characters

Conversion characters specify the notation of the output.

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%c</td>
<td>Single character</td>
</tr>
<tr>
<td>%d</td>
<td>Decimal notation (signed)</td>
</tr>
<tr>
<td>%e</td>
<td>Exponential notation (using a lowercase e as in (3.1415e+00))</td>
</tr>
<tr>
<td>%E</td>
<td>Exponential notation (using an uppercase E as in (3.1415E+00))</td>
</tr>
<tr>
<td>%f</td>
<td>Fixed-point notation</td>
</tr>
<tr>
<td>%g</td>
<td>The more compact of %e or %f, as defined in [2]. Insignificant zeros do not print.</td>
</tr>
<tr>
<td>%G</td>
<td>Same as %g, but using an uppercase E</td>
</tr>
<tr>
<td>%i</td>
<td>Decimal notation (signed)</td>
</tr>
<tr>
<td>%o</td>
<td>Octal notation (unsigned)</td>
</tr>
<tr>
<td>%s</td>
<td>String of characters</td>
</tr>
<tr>
<td>%u</td>
<td>Decimal notation (unsigned)</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal notation (using lowercase letters a-f)</td>
</tr>
<tr>
<td>%X</td>
<td>Hexadecimal notation (using uppercase letters A-F)</td>
</tr>
</tbody>
</table>

## Escape Characters

This table lists the escape character sequences you use to specify non-printing characters in a format specification.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>\f</td>
<td>Form feed</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal tab</td>
</tr>
<tr>
<td>&quot;</td>
<td>Backslash</td>
</tr>
<tr>
<td>&quot; or &quot;</td>
<td>Single quotation mark</td>
</tr>
<tr>
<td>(two single quotes)</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>Percent character</td>
</tr>
</tbody>
</table>
4.5 A related (non-MATLAB) issue:
Creating an Excel spreadsheet from a text file

MATLAB is often used to create text files filled with formatted columns of data. In order to manipulate this data and have more flexibility in formatting, it is often desired to import the data into a spreadsheet. If the data are in fixed-width columns, perform the following:
1. Open Excel
2. File – Open – Files of type – select All Files
3. Open the .txt file
4. When prompted, choose Fixed Width
5. Adjust the width of the break lines if necessary
6. Next
7. Column data format – General
8. Finish

If the data columns are separated by tabs or commas (or some other delimiter), in step 4, select “Delimited” instead of “Fixed Width.”

5.0 COMMON COMMANDS

5.1 Predefined Constants

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi</td>
<td>3.14159265358979…(to machine accuracy)</td>
</tr>
<tr>
<td>inf</td>
<td>Infinity (machine limit)</td>
</tr>
<tr>
<td>i</td>
<td>sqrt(-1)</td>
</tr>
<tr>
<td>j</td>
<td>sqrt(-1)</td>
</tr>
</tbody>
</table>

Note: these constants can be overloaded. That is, if a variable is defined in the workspace with the same name as a MATLAB constant, the value of that variable will be that which is defined in the workspace. For example, if the statement pi = 5 is executed, the next reference to pi will have the value 5.
5.2 Basic Math functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>Notes</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>+</td>
<td></td>
<td>x=a+b</td>
</tr>
<tr>
<td>Subtract</td>
<td>-</td>
<td></td>
<td>x=a-b</td>
</tr>
<tr>
<td>Multiply</td>
<td>*</td>
<td>* and / take precedence over + and -</td>
<td>x=a*b</td>
</tr>
<tr>
<td>Divide</td>
<td>/</td>
<td></td>
<td>x=a/b</td>
</tr>
<tr>
<td>Absolute Value</td>
<td>abs(number)</td>
<td></td>
<td>x=abs(y)</td>
</tr>
<tr>
<td>Cosine</td>
<td>cos(angle)</td>
<td>angle must be in radians</td>
<td>x=cos(0.5)</td>
</tr>
<tr>
<td>Sine</td>
<td>sin(angle)</td>
<td>angle must be in radians</td>
<td>x=sin(0.5)</td>
</tr>
<tr>
<td>Tangent</td>
<td>tan(angle)</td>
<td>angle must be in radians</td>
<td>x=tan(0.5)</td>
</tr>
<tr>
<td>Inverse Cosine</td>
<td>acos(angle)</td>
<td>angle must be in radians</td>
<td>x=acos(0.5)</td>
</tr>
<tr>
<td>Inverse Sine</td>
<td>asin(angle)</td>
<td>angle must be in radians</td>
<td>x=asin(0.5)</td>
</tr>
<tr>
<td>Powers</td>
<td>x^y</td>
<td>x raised to the y power</td>
<td>x=theta^2</td>
</tr>
<tr>
<td>Square root</td>
<td>sqrt(number)</td>
<td>will return imaginary if number&lt;0</td>
<td>x=sqrt(theta)</td>
</tr>
<tr>
<td>Exponential</td>
<td>exp(number)</td>
<td>returns e^(number)</td>
<td>x=exp(theta)</td>
</tr>
<tr>
<td>Natural log</td>
<td>log(number)</td>
<td></td>
<td>x=log(0.5)</td>
</tr>
<tr>
<td>Log base 10</td>
<td>log10(number)</td>
<td></td>
<td>x=log10(0.5)</td>
</tr>
<tr>
<td>Log base 2</td>
<td>log2(number)</td>
<td></td>
<td>x=log2(0.5)</td>
</tr>
</tbody>
</table>

5.3 Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>is equal to</td>
</tr>
<tr>
<td>~=</td>
<td>is NOT equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>is less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>is greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>is less than or equal to</td>
</tr>
<tr>
<td>=&gt;</td>
<td>is greater than or equal to</td>
</tr>
<tr>
<td>and (A, B)</td>
<td>True if A and B are true</td>
</tr>
<tr>
<td>or (A, B)</td>
<td>True if A or B is true</td>
</tr>
</tbody>
</table>
6.0 ARRAYS
One of the most powerful features of MATLAB is its ability to easily handle vectors and matrices.

6.1 Array Dimensions
A vector is a 1-dimensional array. That is, it has only one index: \( v(n) \), where \( n \) is any positive integer. A matrix is a 2-dimensional array. That is, it has two indices: \( A(n,m) \), where \( n \) and \( m \) are any positive integers. It is also possible to define higher dimensioned arrays, such as \( B(n,m,p,q,...) \). Note, all MATLAB arrays are 1-based arrays (the first index is 1, not 0).

6.2 Entering an Array
Entering an array is simple and can be accomplished many different ways. Below are examples of entering a 3×3 matrix:

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

A matrix can be formed using row vectors
\[
\begin{align*}
&\text{r1} = [1 \ 2 \ 3]; \\
&\text{r2} = [4 \ 5 \ 6]; \\
&\text{r3} = [7 \ 8 \ 9]; \\
&A = [\text{r1}; \text{r2}; \text{r3}];
\end{align*}
\]

A matrix can be formed using column vectors
\[
\begin{align*}
&\text{c1} = [1; \ 4; \ 7]; \\
&\text{c2} = [2; \ 5; \ 8]; \\
&\text{c3} = [3; \ 6; \ 9]; \\
&A = [\text{c1} \ \text{c2} \ \text{c3}];
\end{align*}
\]

Sometimes it is beneficial to fill a matrix from within a loop
\[
\begin{align*}
&\text{count} = 0 \\
&\text{for } I = 1:3 \\
&\text{\quad for } J = 1:3 \\
&\text{\quad \quad \text{count} = \text{count} + 1} \\
&\text{\quad \quad A(I,J) = \text{count}} \\
&\text{\quad end} \\
&\text{end}
\end{align*}
\]

Individual elements of an array can be accessed directly using the matrix indices. For example, if the A matrix above is already defined,
A(2,2) = 0; results in \[ A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 6 \\ 7 & 8 & 9 \end{bmatrix} \]

A(:,3) = 0; results in \[ A = \begin{bmatrix} 1 & 2 & 0 \\ 4 & 5 & 0 \\ 7 & 8 & 0 \end{bmatrix} \]

A(1,:) = 0; results in \[ A = \begin{bmatrix} 0 & 0 & 0 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \]

To create a vector with values that are equally spaced across a given range:

\[ t = 0:0.1:1 \]

results in the vector \[ t = [0 ~ 0.1 ~ 0.2 ~ 0.3 ~ 0.4 ~ 0.5 ~ 0.6 ~ 0.7 ~ 0.8 ~ 0.9 ~ 1.0] \]

### 6.3 Some Special Matrices

Identity matrix: \texttt{eye(n)} or \texttt{eye(n,n)} results in an \( n \times n \) identity matrix.

Zero matrix: \texttt{zeros(n,m)} results in an \( n \times m \) matrix with 0 in all the elements.

Ones matrix: \texttt{ones(n,m)} results in an \( n \times m \) matrix with 1 in all the elements.

### 6.4 Some Elementary Matrix Operations

Note that all matrix and vector operations must be consistent with typical matrix and vector algebra. For example, dimensions must be compatible or an error will result.

Matrix addition, subtraction: \( A+B, ~ A-B \)

Matrix multiplication: \( A*B \)

Matrix inverse: \( \text{inv}(A) \)

Matrix transpose: \( A' \)

Matrix determinant: \( \text{det}(A) \)

Matrix eigenvalues: \( \text{eig}(A) \)
In order to force an element-by-element operation, precede the operator with a “./”.

For example, if \(\mathbf{A}\) and \(\mathbf{B}\) are defined as

\[
\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}; \quad \mathbf{B} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix};
\]

\[
\mathbf{C} = \mathbf{A}/\mathbf{B} \quad \text{results in} \quad \mathbf{C} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \quad \text{this is essentially the same as} \quad \mathbf{A} \times \text{inv}(\mathbf{B})
\]

\[
\mathbf{C} = \mathbf{A}./\mathbf{B} \quad \text{results in} \quad \mathbf{C} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}
\]

### 7.0 PROGRAM STRUCTURES

#### 7.1 If statement

The general form of the *if* statement is

```matlab
if expression
    statements
elseif expression
    statements
else
    statements
end
```

The statements are executed if the real part of the expression has all non-zero elements. The *else* and *elseif* parts are optional. Zero or more *elseif* parts can be used as well as nested if’s. The *expression* is usually of the logical form \(x \text{ logic } y\), where *logic* is \(==, <, >, <=, >=, ~=\).

Example

```matlab
if I == J
    A(I,J) = 2;
elseif abs(I-J) == 1
    A(I,J) = -1;
else
    A(I,J) = 0;
end
```
7.2 For Loop
Repeats statements a set number of times. Use the *for* loop when the number of loop iterations is known. The general form of a *for* statement is:

```
for I=n:m
    statements
end
```

where *I* is the incremental index whose range varies from *n* to *m*. Default step size for the increment is 1. To specify a different step size, use *I=n:step:m*, where *step* can have positive or negative decimal values. Note, never assign a value to the index *I* inside the loop.

For example:
```
N = 10;
for I = 1:N,
    for J = 1:N,
        A(I,J) = 1/(I+J-1);
    end
end
```

7.3 While Loop
Repeats statements an indefinite number of times. Use the *while* loop when the number of loop iterations is unknown. Note, this structure has the potential of creating an infinite loop. Always include a break statement to prevent an infinite loop. The general form of a *while* statement is:

```
while expression
    statements
end
```

The statements are executed while the real part of the expression has all non-zero elements. The expression is usually of the logical form *x* *logic* *y*, where *logic* is =, <, >, <=, >=, ~=. 

For example:
```
Vlol = 150;          % Lift-off velocity (ft/s)
V    = 0;           % Initialize current velocity (ft/s)
t    = 0;           % Initialize time (s)
while V < Vlol     % LOOP until lift-off velocity is reached
    V = V + accel*dt;   % calculate new velocity
    t = t + dt;        % increment time
    if t > 500          % break if time exceeds reasonable value
        break
    end
end
```
7.4 Break statement

The break statement can be used to terminate a *for* or *while* loop prematurely.

For example:

```plaintext
E = 0*A; F = E + eye(size(E)); N = 1;
while norm(E+F-E,1) > 0,
    E = E + F;
    F = A*F/N;
    N = N + 1;
    if (N >= 100)
        break
    end
end
```

8.0 PLOTTING

8.1 Generating a MATLAB Plot

The basic command to generate a plot window is

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

where x and y are vectors of the same length. In order to put multiple series of data on the same plot, include the x-y vector pairs of the additional data in the plot command.

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

This plots three lines of data. Note (x1,y1) must be the same length, (x2,y2) must be the same length, etc.; however, x1, x2, and x3 do NOT have to be the same length.

Multiple data series on the same plot can also be accomplished using the *hold* command.

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

Multiple plots can be placed on a single page using the *subplot* command.

```plaintext
subplot(2,1,1);  % 2,1 specifies 2×1 plot layout on the page; the last digit, 1, selects the first plot
plot(x1,y1,x2,y2);  % plots two lines in the first plot
subplot(2,1,2);  % selects the second plot on the page
plot(x3,y3);  % plots one line in the second plot
```

To save a MATLAB plot for use later, click file save on the plot window. The figure will be saved as a *.fig file which can be opened at the command prompt window. Note, the format of the file is only readable by MATLAB. The *.fig file can not be directly imported into Word or image editing programs.

To print the plot, click the printer icon on the figure window tool bar.
8.2 Formatting a MATLAB Plot

8.2.1 Formatting a plot directly from the plot window
- Click on the arrow icon on the toolbar to activate editing options
- Right-click anywhere on the figure to bring up the editing menu
  --- Select properties from the pop-up menu
  --- Click on the grid box for both x and y axis to add a grid
  --- Add a title on the title box
  --- Add labels for the x and y axis
  --- Select OK to implement
- Right-click on the plot line to edit the type of line
  --- Select properties from the pop-up menu
  --- Select color, line width, line style, and marker style
  --- Select OK to implement
- Click on the A icon on the toolbar to add text to the plot
  --- Left-click on the plot at the location to add text
  --- Click anywhere outside the text box to exit adding text

8.2.2 Formatting a plot from the command prompt
All the formatting can also be accomplished at the command prompt. Often, if a particular plot is to be generated many times, it is more efficient to do all the formatting in this manner and put it into an m-file. See the example m-file in Section 10. Some common formatting statements include:

```
plot(x,y,'r--')          % specifies color: 'r' is red ; and line type: '--' is dashed; see help plot
xlabel('text')          % creates an x-axis label with the string 'text'
ylabel('text')          % creates a y-axis label with the string 'text'
title('text')            % creates a plot title with the string 'text'
axis([x1  x2  y1  y2])   % forces the axis limits:  x1 to x2, and y1 to y2
text(x,y,'text')        % creates a text box with the string ‘text’ at plot location x,y
```

8.2.3 Special characters and string formatting
Greek symbols and special characters are available for use in plot string formatting.

For example:

```
ylabel('Angle of Attack, \alpha, (deg)')
```

results in the y-axis label: Angle of Attack, $\alpha$, (deg)

```
title('C_D vs C_L^2')
```

results in the title: $C_D$ vs $C_L^2$
title('C_{L_\alpha} in \text{italics} and \text{bold} and \text{both}')

results in the title: $C_{L_\alpha}$ \text{in italics and bold and both}

A full table listing available characters and the character sequence used to define them is given in Table 1, page 40. Also, see the example m-file given in Section 10.

8.3 Inserting a MATLAB Plot into a Microsoft product (Word, Excel, Powerpoint)

To copy a MATLAB figure for use in Word, Excel, or Powerpoint,

1. On the MATLAB plot window select Edit $\rightarrow$ Copy Figure (this saves the figure to the clipboard)
2. In the Microsoft program, select Edit $\rightarrow$ paste
9.0 CONTROLS TOOLBOX BASICS

One of MATLAB’s strengths is its toolbox for performing controls-related mathematical manipulations. Note that toolboxes are add-ons in MATLAB. If you do not have the Controls Toolbox, you will not be able to use any of its functions.

9.1 Transfer Functions

In MATLAB, transfer functions are handled in the traditional sense: a transfer function maps an input to an output in the Laplace domain.

\[
\begin{array}{c}
\text{u(s)} \\
G(s) \\
\text{y(s)}
\end{array}
\]

The continuous time transfer function \( G(s) \) is stored in the MATLAB workspace as an LTI (Linear Time Invariant) class. There are several types of LTI objects, two of which represent transfer functions: \( \text{tf object} \), and \( \text{zpk object} \). The \( \text{tf object} \) stores the numerator and denominator as vectors of coefficients of the polynomials in \( s \). The \( \text{zpk object} \) stores the numerator and denominator as vectors of zeros and poles multiplied by a gain.

For example, consider the following transfer function:

\[
\frac{25(s + 1)}{s(s + 10)(s^2 + 4s + 16)}
\]

If this is entered as a \( \text{tf object} \), MATLAB will display it as:

\[
G = \frac{25s + 25}{s^4 + 14s^3 + 56s^2 + 160s}
\]

the numerator is a vector of coefficients: [25 25]  
-or-  25*[1 1]
the denominator is a vector of coefficients: [1 14 56 160 0]

If this is entered as a \( \text{zpk object} \), MATLAB will display it as:

\[
G = \frac{25(s + 1)}{s(s + 10)(s^2 + 4s + 16)}
\]

the vector of zeros gives the location of the roots of the numerator: [-1]  
the vector of poles gives the location of the roots of the denominator: [-10 (-2+3.46i) (-2-3.46i) 0]  
the gain is simply: 25
Although MATLAB stores the representation of the transfer function differently in the *tf object* and *zp object*, both representations give the same dynamic results. The primary reason for choosing one type of object over another is the ease of input. This is illustrated in the following section.

### 9.1.1 Entering Transfer Functions

#### 9.1.1.1 Entering a tf object

The syntax for defining a *tf object* is:

\[
G = \text{tf(num,den)}
\]

where *num* is a vector of coefficients of the numerator polynomial, and *den* is a vector of coefficients of the denominator polynomial. The vector of coefficients has the form: \([a_n \ a_{n-1} \ a_{n-2} \ldots \ a_1 \ a_0]\). These are the coefficients of the polynomial \(a_n s^n + a_{n-1} s^{n-1} + a_{n-2} s^{n-2} + \cdots + a_1 s + a_0\).

**Example**: Enter the transfer function \(\frac{25(s+1)}{(s^2+4s+16)}\)

\[
\begin{align*}
\text{num} & = 25*[1 \ 1] \\
\text{den} & = [1 \ 4 \ 16] \\
G & = \text{tf(num,den)}
\end{align*}
\]

**Example**: Enter the transfer function \(\frac{10}{s(s^2+4s+16)}\)

\[
\begin{align*}
\text{num} & = 10 \\
\text{den} & = [1 \ 4 \ 16 \ 0] \\
G & = \text{tf(num,den)}
\end{align*}
\]

Note, the variables *num* and *den* can be any variable name. Also, they can be avoided altogether, as in the following example.

**Example**: Enter the transfer function \(\frac{10}{s(s^2+4s+16)}\)

\[
G = \text{tf(10,[1 \ 4 \ 16 \ 0])}
\]

When the polynomial has several terms and is given in factored form, there are a couple of options. The first option is to break the transfer function into pieces and multiply the pieces together.

**Example**: Enter the transfer function

\[
\frac{10(s+1)}{s(s^2+8s+25)(s^2+4s+16)} \Rightarrow \left(\frac{10}{s(s^2+8s^2+25s)}\right)\left(\frac{(s+1)}{(s^2+4s+16)}\right)
\]

\[
G = \text{tf(10,[1 \ 8 \ 25 \ 0])}*\text{tf([1 \ 1],[1 \ 4 \ 16])}
\]
The second option is to multiply the polynomials together before forming the transfer function. This is done using the convolve command: \texttt{conv(a,b)}. See discussion in Section 9.2.4.

\textbf{Example:} Enter the transfer function \[
\frac{10(s+1)}{s(s^2+8s+25)(s^2+4s+16)}
\]

\[G = \text{tf}(10*[1 1], \text{conv}([1 8 25 0],[1 4 16]))\]

\textbf{When to use the tf object:} any time the polynomial has an order higher than \(n=1\) and is given in unfactored coefficient form.

\subsection*{9.1.1.2 Entering a zpk object}

The syntax for defining a \textit{zpk} object is:

\[G = \text{zpk}(z,p,k)\]

where \(z\) is a vector of zero locations, \(p\) is a vector of pole locations, and \(k\) is the transfer function gain. The vector of zeros has the form: \([z_n \ z_{n-1} \ z_{n-2} \ ... \ z_1 \ z_0]\). These are the roots of the numerator polynomial \((s-z_n) (s-z_{n-1}) (s-z_{n-2}) \cdots (s-z_1) (s-z_0)\). The vector of poles has the form: \([p_n \ p_{n-1} \ p_{n-2} \ ... \ p_1 \ p_0]\). These are the roots of the denominator polynomial \((s-p_n) (s-p_{n-1}) (s-p_{n-2}) \cdots (s-p_1) (s-p_0)\).

\textbf{Example:} Enter the transfer function \[
\frac{25(s+1)}{(s+5)(s+10)(s-2)}
\]

\[
z = -1
\]

\[
p = [-5 -10 2]
\]

\[
k = 25
\]

\[G = \text{zpk}(z,p,k)\]

A root at \(s=0\) and the absence of a root are not the same thing. To illustrate, the following example has no zeros, and it has a pole at \(s=0\). To enter “no” roots, use a null vector: \(z = []\). To enter a root at \(s=0\), include a 0 in the vector of roots. If the roots are not real, enter the set of complex conjugates.

\textbf{Example:} Enter the transfer function \[
\frac{10}{s(s^2+4s+16)}
\]

\[
z = []
\]

\[
p = [0 -2+3.46*i -2-3.46*i]
\]

\[
k = 10
\]

\[G = \text{zpk}(z,p,k)\]

Note, the variables \(z\), \(p\), and \(k\) can be any variable name. Also, they can be avoided altogether, as in the following example.
Example: Enter the transfer function \[
\frac{10}{s(s^2 + 4s + 16)}
\]

\[
G = \text{zpk}([], [0 -2 + 3.46i -2 - 3.46i], 10)
\]

**When to use the zpk object:** any time the polynomial is given in factored form and has no terms with orders greater than n=1.

### 9.1.1.3 Combining tf and zpk objects

MATLAB allows you to combine *tf objects* and *zpk objects* and to change from one form to another. To illustrate, consider a transfer function with first and second order poles. Break the transfer function into pieces and multiply the pieces together, using the simplest input method for each of the pieces.

**Example:** Enter the transfer function

\[
\frac{10(s + 1)}{s(s + 5)(s + 20)(s^2 + 4s + 16)} \quad \Rightarrow \quad \left( \frac{10(s + 1)}{s(s + 5)(s + 20)} \right) \left( \frac{1}{s^2 + 4s + 16} \right)
\]

\[
G = \text{zpk}(-1, [0 -5 -20], 10) * \text{tf}(1, [1 4 16])
\]

It is also possible to easily convert between data types. For example, the following defines a transfer function as a *tf object*, converts it to a *zpk object*, and then converts the *zpk object* back to a *tf object*.

\[
G1 = \text{tf}(1, [1 9 26 24])
\]

\[
\frac{1}{s^3 + 9s^2 + 26s + 24} \quad \Rightarrow \quad \text{tf object}
\]

\[
G2 = \text{zpk}(G1)
\]

\[
\frac{1}{(s + 2)(s + 3)(s + 4)} \quad \Rightarrow \quad \text{zpk object}
\]

\[
G3 = \text{tf}(G2)
\]

\[
\frac{1}{s^3 + 9s^2 + 26s + 24} \quad \Rightarrow \quad \text{tf object}
\]
9.1.2 Block Diagram Manipulation

9.1.2.1 Combining transfer functions in serial

Combine \(sys1\) and \(sys2\) to create \(sys3\):

\[sys3 = sys1 \times sys2\]

9.1.2.2 Closing a feedback loop

Combine \(sys1\) and \(sys2\) to create \(sys3\):

\[sys3 = \frac{sys1}{1 + sys1 \times sys2}\]

-or-

\[sys3 = \text{feedback}(sys1, sys2)\]

computes an LTI model for the closed-loop feedback system.
Note: when MATLAB combines LTI objects, it does not perform any pole/zero cancellations. For example, given

\[ g_1 = \frac{(s + 0.001)}{(s + 20)} \quad g_2 = \frac{25(s + 1)}{(s + 0.001)(s^2 + 8s + 25)} \]

If \( g_1 \) and \( g_2 \) are combined in serial, the resulting transfer function will be:

\[ g_3 = g_1 \ast g_2 \]

\[ g_3 = \frac{25(s + 0.001)(s + 1)}{(s + 20)(s + 0.001)(s^2 + 8s + 25)} \]

In order to force MATLAB to find the minimum realization (perform the pole/zero cancellations if they are “close”), use the MINREAL function. See Section 9.2.5 for a discussion of MINREAL.

9.2 Common Commands

9.2.1 TFDATA

Extract the coefficients of the numerator and denominator polynomials from a transfer function.

For a SISO model SYS, use the syntax

\[ [\text{num}, \text{den}] = \text{TFDATA}(\text{SYS},'v') \]

- SYS is the tf object
- num is the vector of numerator polynomial coefficients
- den is the vector of denominator polynomial coefficients
- ‘v’ forces the function to return the numerator and denominator as row vectors rather than cell arrays.

9.2.2 ROOTS

Find polynomial roots.

\[ \text{ROOTS(C)} \]

Computes the roots of the polynomial whose coefficients are the elements of the vector C.

The vector of coefficients has the form: \( C = [a_n \ a_{n-1} \ a_{n-2} \ \ldots \ a_1 \ a_0]. \)
These are the coefficients of the polynomial \( a_n s^n + a_{n-1} s^{n-1} + a_{n-2} s^{n-2} + \cdots + a_1 s + a_0. \)
Example:

\[ \text{num} = [1 \ 2 \ 16]; \]
\[ \text{den} = [1 \ 3 \ 27 \ 9]; \]
\[ \text{roots(num)} \]
\[ \text{ans} = \]
\[ -1.0000 + 3.8730i \]
\[ -1.0000 - 3.8730i \]
\[ \text{roots(den)} \]
\[ \text{ans} = \]
\[ -1.3275 + 4.9317i \]
\[ -1.3275 - 4.9317i \]
\[ -0.3450 \]

**9.2.3 DAMP**

Find eigenvalues, damping, and natural frequency of an LTI object. (Note: an Eigenvalue is a root)

\[ \text{damp(sys)} \]

where \( \text{sys} \) is an LTI object or vector of polynomial coefficients.

Example:

\[ \text{num} = [1 \ 2 \ 16]; \]
\[ \text{den} = [1 \ 3 \ 27 \ 9]; \]
\[ \text{g} = \text{tf(num,den)}; \]

\[ >> \text{damp(den)} \]

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Damping</th>
<th>Freq. (rad/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3.45e-001</td>
<td>1.00e+000</td>
<td>3.45e-001</td>
</tr>
<tr>
<td>-1.33e+000 + 4.93e+000i</td>
<td>2.60e-001</td>
<td>5.11e+000</td>
</tr>
<tr>
<td>-1.33e+000 - 4.93e+000i</td>
<td>2.60e-001</td>
<td>5.11e+000</td>
</tr>
</tbody>
</table>

\[ >> \text{damp(g)} \]

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>Damping</th>
<th>Freq. (rad/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3.45e-001</td>
<td>1.00e+000</td>
<td>3.45e-001</td>
</tr>
<tr>
<td>-1.33e+000 + 4.93e+000i</td>
<td>2.60e-001</td>
<td>5.11e+000</td>
</tr>
<tr>
<td>-1.33e+000 - 4.93e+000i</td>
<td>2.60e-001</td>
<td>5.11e+000</td>
</tr>
</tbody>
</table>
9.2.4 CONV
Vector convolution and polynomial multiplication.

**C = conv(A, B)**

convolves vectors A and B. If A and B are vectors of polynomial coefficients, this is equivalent to multiplying the two polynomials.

Example: Given \( x = (s^2 + 3s + 6) \); \( y = (s^2 + 4s + 8) \); \( z = (s^2 + 5s + 10) \)
Multiply the polynomials \( x \), \( y \), and \( z \) together.

\[
\begin{align*}
x &= [1 \ 3 \ 6]; \\
y &= [1 \ 4 \ 8]; \\
z &= [1 \ 5 \ 10];
\end{align*}
\]

**Do it in two steps:**

\[
\begin{align*}
a &= \text{conv}(x, y) \\
a &= 1 \ 7 \ 26 \ 48 \ 48 \\
b &= \text{conv}(a, z) \\
b &= 1 \ 12 \ 71 \ 248 \ 548 \ 720 \ 480
\end{align*}
\]

**Do it in a single step:**

\[
\begin{align*}
c &= \text{conv}(x, \text{conv}(y, z)) \\
c &= 1 \ 12 \ 71 \ 248 \ 548 \ 720 \ 480
\end{align*}
\]

**Do it in a single step without defining \( x \), \( y \), and \( z \):**

\[
\begin{align*}
d &= \text{conv}([1 \ 3 \ 6], \text{conv}([1 \ 4 \ 8],[1 \ 5 \ 10])) \\
d &= 1 \ 12 \ 71 \ 248 \ 548 \ 720 \ 480
\end{align*}
\]

So the answer is: \( b = c = d = s^6 + 12s^5 + 71s^4 + 248s^3 + 548s^2 + 720s + 480 \)
9.2.5 MINREAL

Find the minimum realization of an LTI system.

\[
\text{MSYS} = \text{minreal}(\text{SYS})
\]

produces, for a given LTI model SYS, an equivalent model MSYS where all canceling pole/zero pairs or non minimal state dynamics are eliminated.

For example, given

\[
g_1 = \frac{s + 0.001}{s + 20} \quad g_2 = \frac{25(s + 1)}{(s + 0.001)(s^2 + 8s + 25)}
\]

If g1 and g2 are combined in serial, the resulting transfer function will be:

\[
g_3 = g_1 \times g_2
\]

\[
g_3 = \frac{25(s + 0.001)(s + 1)}{(s + 20)(s + 0.001)(s^2 + 8s + 25)}
\]

Note: MATLAB does not perform the pole/zero cancellations automatically.

\[
g_4 = \text{minreal}(g_3) \quad -\text{or-} \quad g_4 = \text{minreal}(g_1 \times g_2)
\]

results in

\[
g_4 = \frac{25 (s + 1)}{(s + 20)(s^2 + 8s + 25)}
\]
9.3 Root Locus

9.3.1 RLOCUS

RLOCUS(SYS)

computes and plots the root locus of the single-input, single-output LTI model SYS. The root locus plot is used to analyze the negative feedback loop

and shows the trajectories of the closed-loop poles when the feedback gain K varies from 0 to Inf. RLOCUS automatically generates a set of positive gain values that produce a smooth plot.

Example:

```matlab
» rlocus(SYS)
```
9.3.2 RLTOOL

RLTOOL opens the Root Locus Design graphic user interface (GUI). This GUI allows you to design a single-input/single-output (SISO) compensator using Root Locus techniques. A plant model can be imported into the GUI by selecting the Import Model item from the File menu.

RLTOOL(SYS) opens a Root Locus Design GUI using the plant model in SYS. SYS must be a SISO LTI model. When opened with one input argument, the compensator is initialized as a unity gain.

RLTOOL(SYS,COMP) initializes the compensator with the SISO LTI model COMP.

Gains can be changed by either pulling the roots with the hand on the rltool GUI or by entering a gain directly into the Gain box at the top of the GUI. Time response plots to a step or impulse can be seen by clicking in the step or impulse boxes in the lower left part of the GUI.

9.4 Time Response Plots

9.4.1 Step Input Response

STEP(SYS) plots the step response of the LTI transfer function SYS.

STEP(SYS,TFINAL) simulates the step response from t=0 to the final time t=TFINAL.

9.4.2 Impulse Response

IMPULSE(SYS) plots the impulse response of the LTI transfer function SYS.

9.4.3 LTI Viewer

LTIVIEW opens an interactive GUI which allows viewing of step, impulse, Bode, Nyquist or Nichols plots for multiple transfer functions (LTI systems) at the same time. Import the systems from the MATLAB workspace using the File menu.

Right click on the x or y axis to access most functions:
- Select the plot type and systems interactively.
- Display important parameters such as peak time, settling time, rise time, and steady state value on the step response.
10.0 EXAMPLE m-FILES

10.1 Sample 1: m-file which calls another m-file

File: sample1.m

% sample1.m
%
% This program is an example MATLAB m-file. It serves as a master program which controls file i/o and loops through a range of values to generate a table of data in a text file.
% It utilizes another m-file to generate sample data.
% Author: Scott Wells, Fall 2002
%
% Input Variables:
% Value1 Height above sea level (ft)
% Value2 Pressure (lb/ft^2)
%
% Output Variables:
% argVal Argument value used for calculation (no units)
%
% Local Variables:
% I Loop counter (no units)
%
% Output Files:
% sample.txt contains table of values for Value1 & Value2
%
% References:
% None
%
% Documentation:
% Dr Yechout helped me with the syntax for the fprintf statements.
%***********************************************************************

% Open output file and write headers
fid=fopen('sample.txt','w'); % Open file for write
fprintf(fid,' Value1   Value2  '); % Write Header
fprintf(fid,' (feet)    (psf)  '); % Write Header

% Master loop
for I=0:10                      % Loop through 10 iterations
    argVal = I;                 % Assign argument value
    sample1sub;                 % Call secondary m-file
    fprintf(fid, '%6.1f    %4.3e  ', Value1, Value2); % Write to file
end                            % end of For Loop

% Close the output file
fclose(fid);
File: sample1sub.m
% sample1sub.m
%
% This is an m-file which takes an input variable and
% creates two new values based on the value of the input.
% NOTE: All input variables used in this m-file must be
% previously defined in the MATLAB workspace. All output
% variables become part of the workspace. No formal passing of
% input/output arguments is necessary.
%
% Author: Scott Wells, Fall 2002
%
% Input Variables:
%   argVal      Argument value used for calculation (no units)
%
% Output Variables:
%   Value1      Height above sea level (ft)
%   Value2      Pressure (lb/ft^2)
%
% Local Variables:
%   None
%
% References:
%   Smith, John; “Fundamentals of Equations That Don’t Really
%   Mean Anything,” Associated Press; Phi, PA; 1999.
%
% Documentation:
%   None

%***********************************************************************
% Calculate the output variables
Value1 = argVal^2 - sqrt(argVal);  % Height above SL (ft), [Smith, eqn 3.1]
Value2 = 2*exp(argVal);           % Pressure (lb/ft^2), [Smith, eqn 9.2]

The resulting text file is:

File: sample.txt
Value1   Value2
   (feet)     (psf)
  0.0    2.000e+000
  0.0    5.437e+000
  2.6    1.478e+001
  7.3    4.017e+001
 14.0    1.092e+002
 22.8    2.968e+002
 33.6    8.069e+002
 46.4    2.193e+003
 61.2    5.962e+003
 78.0    1.621e+004
10.2 Sample 2: m-file which calls an m-file defined as a function

These m-files accomplish the same thing as Sample 1. The only difference is a function is used instead of a regular m-file. The data in the output text file are exactly the same.

File: sample2.m

```matlab
% sample2.m
% This program is an example MATLAB m-file.
% It serves as a master program which controls file i/o and loops through
% a range of values to generate a table of data in a text file.
% It utilizes a user-defined function to generate sample data.
% Author: Scott Wells, Fall 2002
% Input Variables:
% Value1 Height above sea level (ft)
% Value2 Pressure (lb/ft^2)
% Output Variables:
% argVal Argument value used for calculation (no units)
% Local Variables:
% I Loop counter (no units)
% Output Files:
% sample.txt contains table of values for Value1 & Value2
% References:
% None
% Documentation:
% Dr Yechout helped me with the syntax for the fprintf statements.
**********************************************************************
% Open output file and write headers
fid=fopen('sample.txt','w'); % Open file for write
fprintf(fid,' Value1   Value2
'); % Write Header
fprintf(fid,' (feet)    (psf)
'); % Write Header
% Master loop
for I=0:10 %Loop through 10 iterations
    argVal = I;
    [Value1,Value2]= sample2funct(argVal); % Call function
    fprintf(fid, '%6.1f    %4.3e
', Value1, Value2); % Write to file
end % end of For loop
% Close the output file
fclose(fid);
```
function [val1, val2] = sample2funct(x)

% sample2funct.m
%
% This is a function which takes an input variable and
% creates two new values based on the value of the input.
% NOTE: All input variables used in this m-file must be
% explicitly passed. No variables in the workspace are visible
% to the function. All output variables must be explicitly passed.
%
% Author: Scott Wells, Fall 2002
%
% Input Variables:
% x Argument value used for calculation (no units)
%
% Output Variables:
% val1 Height above sea level (ft)
% val2 Pressure (lb/ft^2)
%
% Local Variables:
% None
%
% References:
% Smith, John; “Fundamentals of Equations That Don’t Really
% Mean Anything,” Associated Press; Phi, PA; 1999.
%
% Documentation:
% None

%***********************************************************************
% Calculate the output variables
val1 = x^2-sqrt(x);  % Height above SL (ft), [Smith, eqn 3.1]
val2 = 2*exp(x);     % Pressure (lb/ft^2), [Smith, eqn 9.2]
10.3 Sample 3:  m-file which generates data, plots the data, formats the plot

File: sample3.m

% sample3.m
%
% This program is an example MATLAB m-file.
% It generates data which illustrates the effects of reducing the
% damping ratio in a transfer function.  It then plots the data
% and formats the plot.
%
% Author:  Scott Wells, Fall 2002
% Input Variables:
%  None
% Output Variables:
%  None
% Local Variables:
%  damp   Vector of damping ratios (no units)
%  I      Loop counter (no units)
%  G      Transfer function (LTI object)
% legendString Matrix of text strings used for legend (no units)
% maxval  Value of maximum output (no units)
% maxloc  Index of location for max output (no units)
% t      Time vector (sec)
% u      Transfer function input vector (no units)
% wn     Natural undamped frequency (rad/s)
% y      Time response output vector (no units)
%
% References:
%  None
% Documentation:
%  None

%***********************************************************************
% Initializations
damp = 1:-0.25:0.25;         % Range of damping ratios
wn   = 10;                   % Natural frequency (rad/s)
t = 0:.05:2;                 % Define time vector (sec)
u = ones(1,length(t));       % Define input vector

% Generate the Data
for I=1:length(damp)         % Loop through the number of different damping ratios
    G = tf([wn^2],[1 2*damp(I)*wn wn^2]);  % Define 2-nd order transfer function
    y(:,I) = lsim(G,u,t);    % Run simulation and load up output matrix
end                          % End For Loop

% Plot the data
% NOTE:  A simple way to plot all the data is the following.
% To accept the default line styles and colors:
% plot(t,u,t,y);
% To specify line styles, marker styles, and colors:
% plot(t,u,t,y(:,1),'bo-',t,y(:,2),'rs-',t,y(:,3),'gd-',t,y(:,4),'cv-')
% To specify additional parameters, each line must have its own plot statement:
% plot(t,u,'k--'); hold on
% plot(t,y(:,1),'bo-','LineWidth',1,'MarkerSize',3);
% plot(t,y(:,2),'rs-','LineWidth',1,'MarkerSize',3);
The resulting plot is given below. Note, the line colors are changed to black for better black & white printing results. This plot is inserted into the Word document as discussed in Section 8.3.
### Table 1: Special characters and the character sequence used to define them

<table>
<thead>
<tr>
<th>Character Sequence</th>
<th>Symbol</th>
<th>Character Sequence</th>
<th>Symbol</th>
<th>Character Sequence</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>\alpha</td>
<td>α</td>
<td>\upsilon</td>
<td>υ</td>
<td>\sim</td>
<td>~</td>
</tr>
<tr>
<td>\beta</td>
<td>β</td>
<td>\phi</td>
<td>ϕ</td>
<td>\leq</td>
<td>≤</td>
</tr>
<tr>
<td>\gamma</td>
<td>γ</td>
<td>\chi</td>
<td>χ</td>
<td>\infty</td>
<td>∞</td>
</tr>
<tr>
<td>\delta</td>
<td>δ</td>
<td>\psi</td>
<td>ψ</td>
<td>\clubsuit</td>
<td>♣</td>
</tr>
<tr>
<td>\epsilon</td>
<td>ε</td>
<td>\omega</td>
<td>ω</td>
<td>\diamondsuit</td>
<td>♦</td>
</tr>
<tr>
<td>\zeta</td>
<td>ζ</td>
<td>\Gamma</td>
<td>Γ</td>
<td>\heartsuit</td>
<td>♥</td>
</tr>
<tr>
<td>\eta</td>
<td>η</td>
<td>\Delta</td>
<td>∆</td>
<td>\spadesuit</td>
<td>♠</td>
</tr>
<tr>
<td>\theta</td>
<td>θ</td>
<td>\Theta</td>
<td>Θ</td>
<td>\leftarrow &amp;rightarrow</td>
<td>← →</td>
</tr>
<tr>
<td>\vartheta</td>
<td>\vartheta</td>
<td>\Lambda</td>
<td>Λ</td>
<td>\leftrightarrow</td>
<td>↔</td>
</tr>
<tr>
<td>\iota</td>
<td>ι</td>
<td>\Xi</td>
<td>Ξ</td>
<td>\uparrow =&gt;</td>
<td>➔</td>
</tr>
<tr>
<td>\kappa</td>
<td>κ</td>
<td>\Pi</td>
<td>Π</td>
<td>\rightarrow</td>
<td>→</td>
</tr>
<tr>
<td>\lambda</td>
<td>λ</td>
<td>\Sigma</td>
<td>Σ</td>
<td>\downarrow</td>
<td>↓</td>
</tr>
<tr>
<td>\mu</td>
<td>µ</td>
<td>\Upsilon</td>
<td>Υ</td>
<td>\circ</td>
<td>°</td>
</tr>
<tr>
<td>\nu</td>
<td>ν</td>
<td>\Phi</td>
<td>Φ</td>
<td>\pm</td>
<td>±</td>
</tr>
<tr>
<td>\xi</td>
<td>ξ</td>
<td>\Psi</td>
<td>Ψ</td>
<td>\geq</td>
<td>≥</td>
</tr>
<tr>
<td>\pi</td>
<td>π</td>
<td>\Omega</td>
<td>Ω</td>
<td>\propto</td>
<td>≠</td>
</tr>
<tr>
<td>\rho</td>
<td>ρ</td>
<td>\forall</td>
<td>∀</td>
<td>\partial</td>
<td>∂</td>
</tr>
<tr>
<td>\sigma</td>
<td>σ</td>
<td>\exists</td>
<td>∃</td>
<td>\bullet</td>
<td>•</td>
</tr>
<tr>
<td>\varsigma</td>
<td>ζ</td>
<td>\ni</td>
<td>ν</td>
<td>\div</td>
<td>÷</td>
</tr>
<tr>
<td>\tau</td>
<td>τ</td>
<td>\cong</td>
<td>≡</td>
<td>\leq</td>
<td>≤</td>
</tr>
<tr>
<td>\equiv</td>
<td>=</td>
<td>\approx</td>
<td>≈</td>
<td>\aleph</td>
<td>ℵ</td>
</tr>
<tr>
<td>\Im</td>
<td>ℑ</td>
<td>\Re</td>
<td>ℜ</td>
<td>\wp</td>
<td>℘</td>
</tr>
<tr>
<td>\times</td>
<td>⊗</td>
<td>\oplus</td>
<td>⊕</td>
<td>\oslash</td>
<td>\₀</td>
</tr>
<tr>
<td>\cap</td>
<td>∩</td>
<td>\cup</td>
<td>∪</td>
<td>\supseteq</td>
<td>⊃</td>
</tr>
<tr>
<td>\cupset</td>
<td>⊃</td>
<td>\subscripteq</td>
<td>≼</td>
<td>\subset</td>
<td>⊂</td>
</tr>
<tr>
<td>\lnot</td>
<td>~</td>
<td>\in</td>
<td>∈</td>
<td>\lo</td>
<td>₀</td>
</tr>
<tr>
<td>\lceil</td>
<td>[</td>
<td>\lceil</td>
<td>[</td>
<td>\nabla</td>
<td>∇</td>
</tr>
<tr>
<td>\floor</td>
<td>]</td>
<td>\floor</td>
<td>]</td>
<td>\ldots</td>
<td>...</td>
</tr>
<tr>
<td>\perp</td>
<td>⊥</td>
<td>\neg</td>
<td>¬</td>
<td>\prime</td>
<td>′</td>
</tr>
<tr>
<td>\wedge</td>
<td>∧</td>
<td>\times</td>
<td>×</td>
<td>\ldots</td>
<td>...</td>
</tr>
<tr>
<td>\lceil</td>
<td>[</td>
<td>\surd</td>
<td>√</td>
<td>\mid</td>
<td></td>
</tr>
<tr>
<td>\lfill</td>
<td>\vphantom{\times}X</td>
<td>\copyright</td>
<td>©</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\langle</td>
<td>(</td>
<td>\rangle</td>
<td>)</td>
<td>\lfill</td>
<td>\irmware{\times}X</td>
</tr>
</tbody>
</table>

Subscript x \( x \)
Superscript x \( ^{(x)} \)
Italics x (\( \textit{x} \))
Bold x (\( \textbf{x} \))
Font x \( \textsc{\texttt{fontname(times)x}} \)
Font Size \( \text{'Fontsize',10} \)
Alignment \( \text{'HorizontalAlignment','Left'} \) (also, \( \text{'Right','Center'} \))
\( \text{'VerticalAlignment','bottom'} \) (also, \( \text{'Top','Middle','Cap','Baseline'} \))