

EEE 304

Lab Exercise 1

1. Introduction to MATLAB

MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation. Using MATLAB, one can solve technical computing problems easier than with traditional programming languages, such as C, C++, and Fortran due to its interpreted nature. The resulting code is vastly more efficient in terms of code development and debugging time, code reusability and maintainability. Although net speed of execution is inferior to that of an optimized low-level code, recent versions allow the auto-coding of MATLAB programs into executables whose speed is often comparable to the more traditional codes.

MATLAB is useful in a wide range of applications, including signal and image processing, communications, control design, test and measurement, financial modeling and analysis, and computational biology. Add-on toolboxes (collections of special-purpose MATLAB functions, available separately) extend the MATLAB environment to solve particular classes of problems in these application areas. MATLAB provides a number of features for integrating code with other languages and applications and documenting and sharing work. (See Mathworks website <http://www.mathworks.com>.)

The objective of this sequence of laboratory exercises is to explore a few fundamental problems arising in System Theory and, at the same time, develop expertise in problem solving with MATLAB.

2.1 MATLAB basics: Setting the path

The MATLAB Integrated Development Environment (IDE) requires that any function or script to be used must be within its search path. Additionally, MATLAB, by default, starts at what is called the "Working Directory" which is always at the top of the search path. In the example below you should set the path where your program is so that MATLAB can find your program and execute it ("E:\Dropbox (ASU)\EEE304\labs\Summer_2016_Matlab_Materials\Lab1").

TIP: You could also add paths by typing the following command
`>> addpath('C:\Users\cwang135\Documents\MATLAB\EEE304\Lab1')`

So, if you would like more than one directory to be in your path you could programmatically add them in this manner.

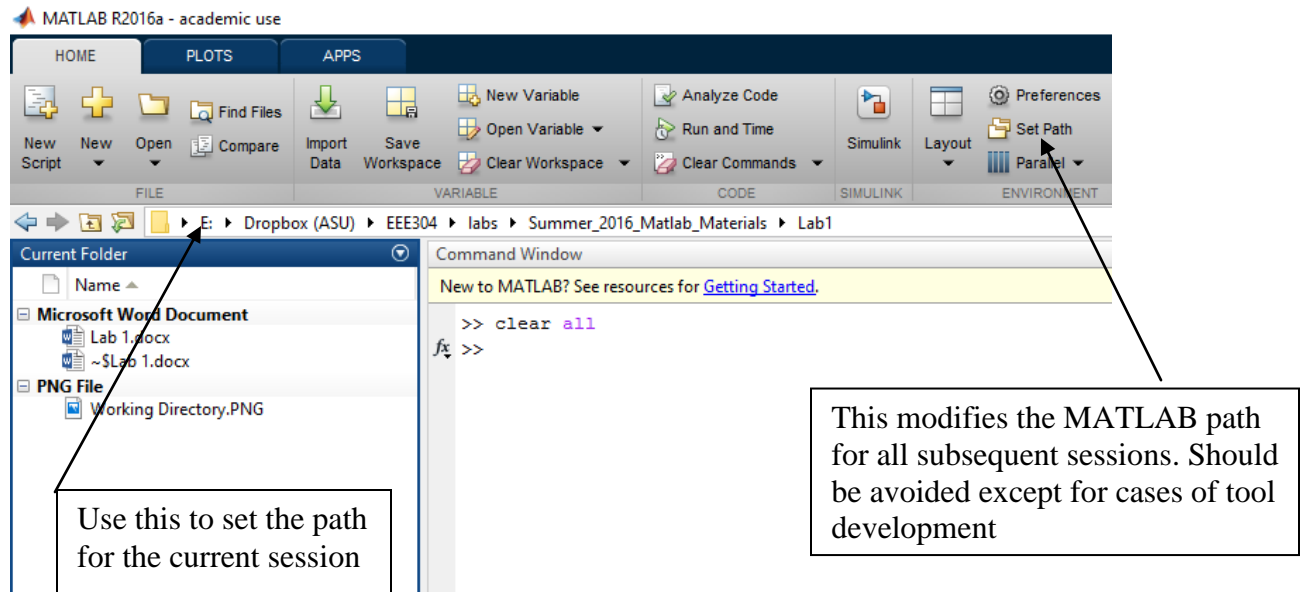


Fig 1. Setting the path of your program

2.2 MATLAB Commands

There are 2 ways to use MATLAB: One is to type the MATLAB Command in the "Command Window". For example, the MATLAB Command "**bench**" is to compare your computer execution speed with other computers by computing some task. By typing "**bench**" and press the "Enter" key, you will see some figures and a table about your computer execution speed.

```
>>bench
```

If you want to know more information about "**bench**", type "**help bench**" as below and press the "Enter" key.

```
>>help bench
```

You will find more information about what "**bench**" does and how to use it.

TIP: The MATLAB documentation is another way of accessing help about commands. It is graphically pleasing and provides links to tutorials and examples at times.

```
>>doc bench
```

If there are more than a few commands to be entered and you would wish to repeat these commands by varying parameters, the second, more elegant, way to use MATLAB is to write a script with extension name "**m**". MATLAB scripts are very useful in that they can be saved for later use, you could divide them into "**sections**", you could also run each line in the scripts one at a time. Let us look at an example of how to do so.

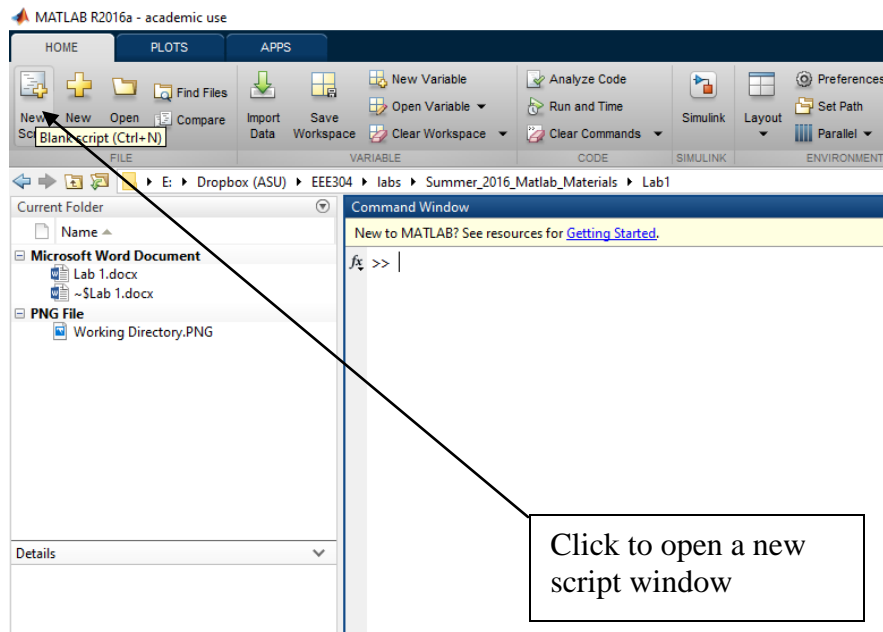


Fig 2. Opening a new script

As shown in fig 2, open a new script and type in the following commands and save the script with name 'eee304.m'.

```
x=1;
y=2;
z=x+y;
```

When you type the filename in the command window and press 'Enter', the file will be executed.

```
>>eee304
```

These three commands will add the variables x and y and assign the result to the variable z. All variables involved in executing commands are stored in what is called the MATLAB "workspace", see figure 3.

In MATLAB, applying a ";" semicolon after the end of a command will suppress any outputs in the command window. To see what MATLAB creates at the end of each command output, simply do not add the semicolon at the end. Try running the script "eee304.m" again but this time take out all three semicolons.

TIP: in order to clear the command window of any previous output type

```
>> clc
```

TIP: to clear workspace and start anew, type

```
>> clear all
```

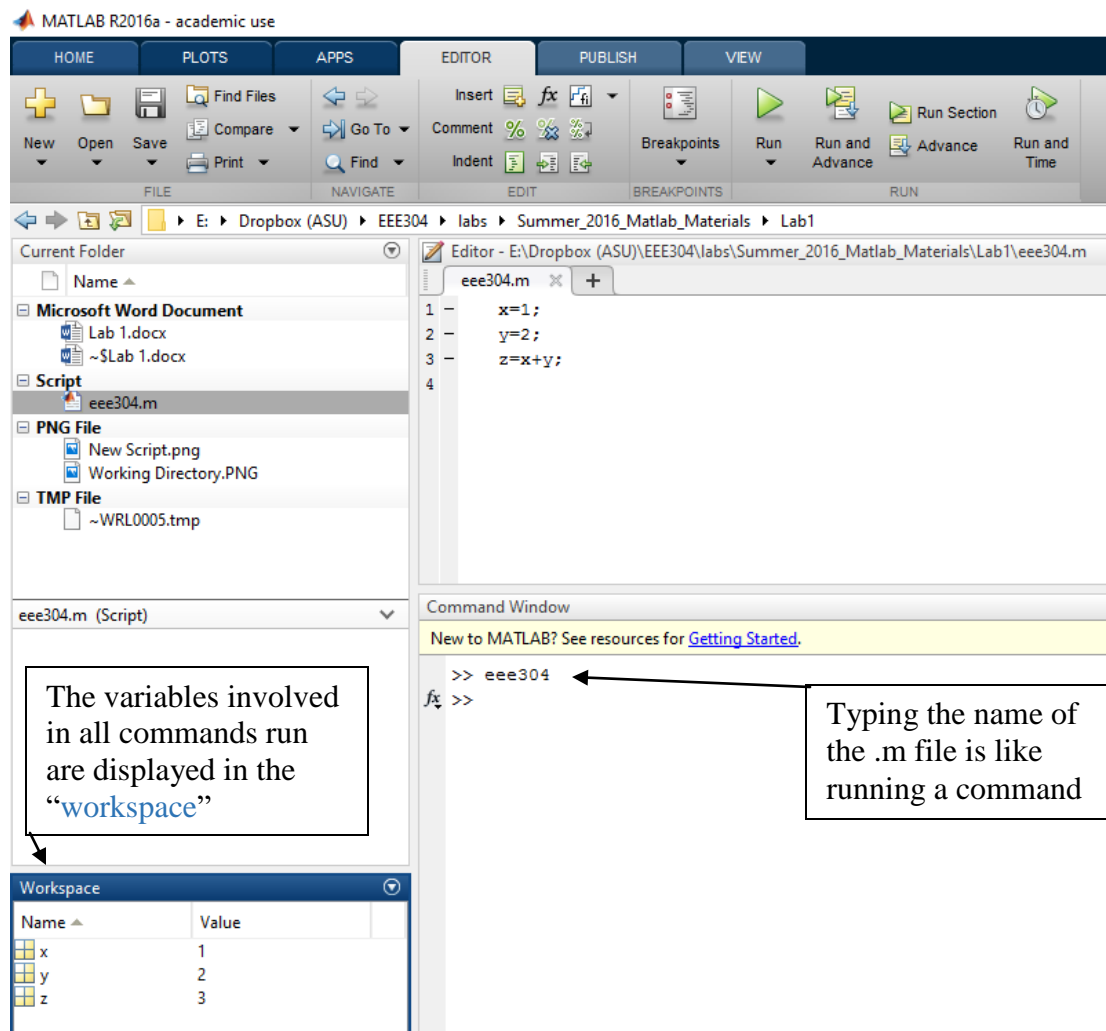


Fig 3. Executing a script

In the following we will work with commands that are useful in the analysis of systems. For detail help, consult the MATLAB Manual, or use the built-in “help” or “doc” commands.

3.1 Create a LTI system by Transfer Function Models

Related Function: [tf](#).

Arguments: numerator polynomial, denominator polynomial, sampling time (optional).

You can create the transfer function $h(s) = \frac{4}{s^2 + 4}$ by

```
>> h = tf([4], [1 0 4])
```

Note that in the “tf” command the two inputs are the coefficients of the numerator and denominator polynomials up to the highest order of the polynomial in the expression. In this example the output “h” is called a “system” object in MATLAB. In many MATLAB examples and documentation, such objects are referred to as “SYS”.

3.2 Step and Bode frequency response of LTI models

Related Function: [bode](#), [step](#)

The command,

```
>>bode(SYS)
```

draws the Bode plot of the LTI model SYS (created with `tf` or other system creation commands). The frequency range and number of points are chosen automatically (a manual selection is possible, see "`help bode`").

Likewise the

```
>>step(SYS)
```

Command generates the step response of the LTI model SYS. For multi-input models, independent step commands are applied to each input channel. The time range and number of points are chosen automatically. The final time of simulation can be modified with the optional second input argument of time, see "`doc step`".

3.3 Continuous to discrete time system conversion

Related Function : `c2d`

In order to convert a continuous time LTI system to a discrete one i.e. discretization, use the MATLAB `c2d` command as follows

```
>>c2d(SYS,TS)
```

where, SYS is the continuous time system object and TS is the sample time. Compare the step responses of a system and its discretization, for different sample times: e.g.,

```
>>step(SYS,c2d(SYS,TS));
```

where TS is varied.

3.4 The frequency analysis of a signal

Related Function : `fft`

FFT stands for Fast Fourier Transform. This command implements the Discrete Fourier transform evaluated at discrete frequencies. It is particularly useful when one wants to know the frequency contents of a sampled signal. The use of the FFT command can be seen with the example below.

Create a script file named `EEE304_fftTest.m` and copy the following piece of code in it

```
fs = 1000; % specify sampling rate
t = 0:1/fs:0.3; % create time vector
w = 0.1; % specify width of pulse
y = rectpuls((t-0.1),w); % generate the rectangular pulse with
delay
plot(t,y) % plot to view the pulse signal
%% compute FFT and plot it. Double '%%' breaks the code into sections
no_pts = 2^nextpow2(length(y)); % create the no. of points of FFT
Y = fft(y,no_pts)/(no_pts); % generate FFT and divide by no._pts to
normalize fft ;
f = (0:no_pts/2-1)*fs/no_pts;

figure
plot(f,2*abs(Y(1:floor(length(Y)/2)))); %plot with linear scale
ylabel('|Y(f)|'); xlabel('Hz')
```

```
figure
semilogx(f,20*log10((2*abs(Y(1:floor(length(Y)/2)))))); %plot with log
dB scales
ylabel('|Y(f)| (dB)'); xlabel('Hz')
```

Run the script by calling its name from the MATLAB command window. This script may come in handy later if you need discrete Fourier Transforms of a generated signal.

TIP: In MATLAB valid variable names must always start with letters, no spaces, dashes or special characters are allowed other than the underscore '_' character. For more see "http://www.mathworks.com/help/matlab/matlab_prog/variable-names.html"

3.5 Filtering

The response of a linear system to arbitrary inputs can be computed with the MATLAB command `lsim`.

```
>>lsim(SYS,U,T)
```

plots the time response of the LTI model `SYS` to the input signal described by `U` having time `T`. The time vector `T` consists of regularly spaced time samples and `U` is a matrix with as many columns as inputs and whose `i`-th row specifies the input value at time `T(i)`. For example, try the following (either in command line or via script)

```
h = tf([3],[1 2 3])
[u,t] = gensig('square',15,30,0.1);
lsim(h,u,t)
```

This simulates the response of a single-input model `h` to the input `u(t)` which is a periodic pulse wave during 30 seconds. For discrete-time models, `U` should be sampled at the same rate as `SYS` (`T` is then redundant and can be omitted or set to the empty matrix).

TIP: See also the signal processing Function "[filter](#)" which performs the same operation albeit with slightly different commands

A different approach is to use the SIMULINK GUI environment to perform the simulation (more details will follow).

3.6 Filter design

Design a Butterworth filter (lowpass, highpass or bandstop)

Related Function : [butter](#)

The MATLAB "[butter](#)" command allows us to create butterworth filters of lowpass, highpass, bandpass or bandstop type.

```
>>[B,A] = butter(N,Wn,'type'); % the analog Butterworth filter
```

The command returns two vectors `B` and `A` which are the coefficients of the digital butterworth filter with normalized cutoff frequency at `Wn` w.r.t Nyquist frequency `fs/2`.

3.7 Sample code

Let us look at how the butterworth filter can be used to filter some noise. Create a new .m file and save it with the following list of commands.

```
fs=1000;% sampling frequency 1000Hz

% generate noise
% the length of noise is 1024 points. It is normally distributed
```

```

noise=randn(1,1024);

% compute FFT and plot it.
no_pts = 2^nextpow2(length(noise));      % create the no. of points of
FFT
AF_noise = fft(noise,no_pts)/(no_pts);    % amplitude-frequency of
noise
f = fs/2*linspace(0,1,length(noise)/2); %the frequency range

figure % create new figure window
plot(f,2*abs(AF_noise(1:floor(length(AF_noise)/2)))); %plot with linear
scales
ylabel('|AF_noise(f)|'); xlabel('Hz')
title('Freq. Resp. of noise data')

%% Now let us filter the noise and look at its frequency response
% bandpass filter
Wn=[50 200]; % pass band is 50~200Hz

%fir fliter
n = 2; %order n
Wn = Wn/(fs/2); %cutoff frequency [300 6000]/sampling frequency/2
[b,a] = butter(n,Wn,'bandpass');

noise_filt = filter(b,a,noise);

% compute FFT and plot it.
no_pts = 2^nextpow2(length(noise_filt)); % create the no. of points
of FFT
AF_noise_filt = fft(noise_filt,no_pts)/(no_pts); % amplitude-
frequency of noise
f = fs/2*linspace(0,1,length(noise_filt)/2); %the frequency range

figure
plot(f,2*abs(AF_noise_filt(1:floor(length(AF_noise_filt)/2)))); %plot
with linear scales
ylabel('|AF_noise(f)|'); xlabel('Hz')

%% Let us view both filtered and unfiltered frequency responses at the
same time
figure
subplot(2,1,1),
plot(f,2*abs(AF_noise(1:floor(length(AF_noise_filt)/2)))); %plot with
linear scales
ylabel('|AF_noise(f)|'); xlabel('Hz')
subplot(2,1,2),
plot(f,2*abs(AF_noise_filt(1:floor(length(noise_filt)/2)))); %plot with
linear scales
ylabel('|AF_noise(f)|'); xlabel('Hz')

```

You could choose to run the entire script at once or you could select each section by clicking and bringing the cursor anywhere in that section then hitting the run section button as shown in figure 4.

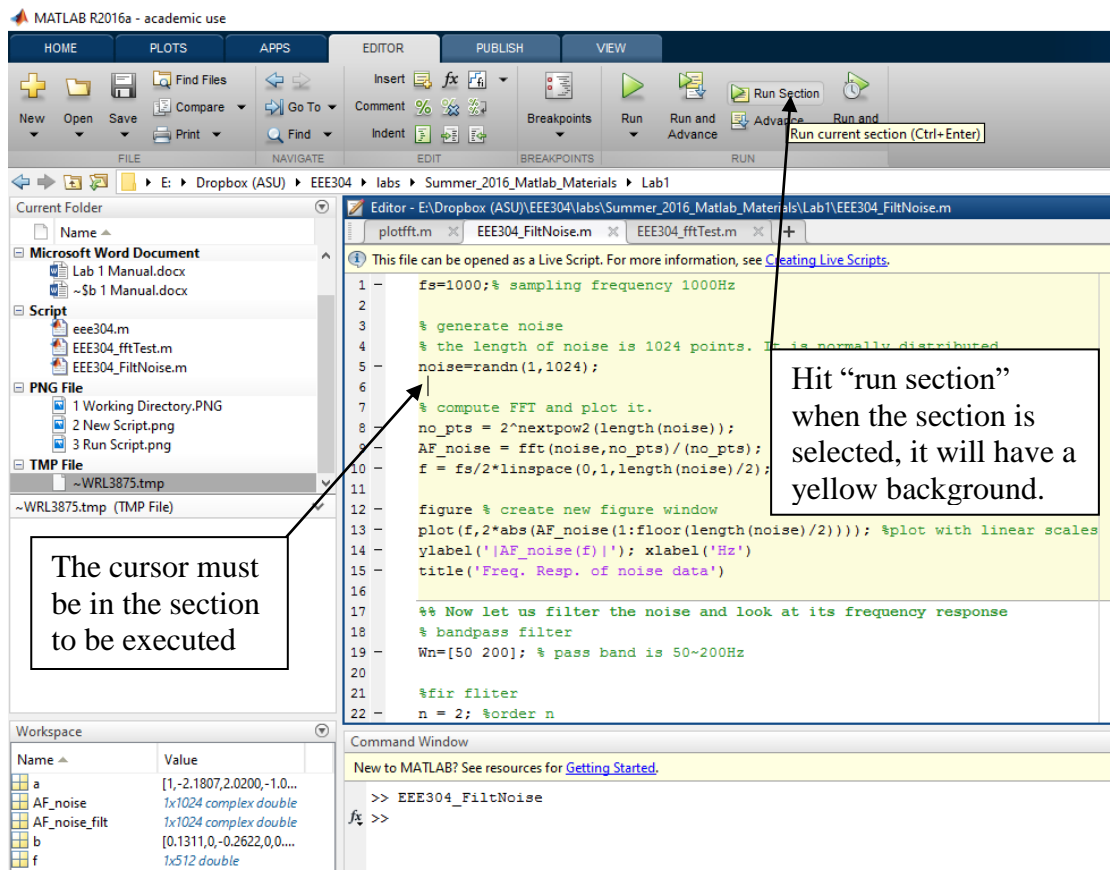


Fig 4. Running a section of code at a time

4.1 Introduction to SIMULINK

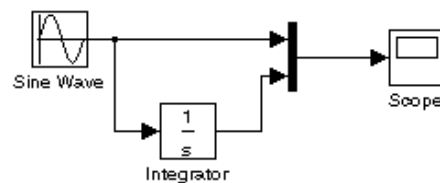
SIMULINK is a platform for multi-domain simulation and Model-Based Design of dynamic systems. It provides an interactive graphical environment and a customizable set of block libraries that allows for an accurate design, simulation, implementation, and testing of control, signal processing, communications, and other time-varying system applications.

Add-on products extend the SIMULINK environment with tools for specific modeling and design tasks and for code generation, algorithm implementation, test, and verification.

SIMULINK is integrated with MATLAB, providing immediate access to an extensive range of tools for algorithm development, data visualization, data analysis and access, and numerical computation.

4.2 Using SIMULINK to build a Model

This example shows how to build a model using some of the model-building commands and actions. A "model" is the SIMULINK equivalent of a code in MATLAB or other languages. The model integrates a sine wave and displays the result along with the sine wave. The block diagram of the model looks like this.



To create the model, first create a new model as shown in figure 5.

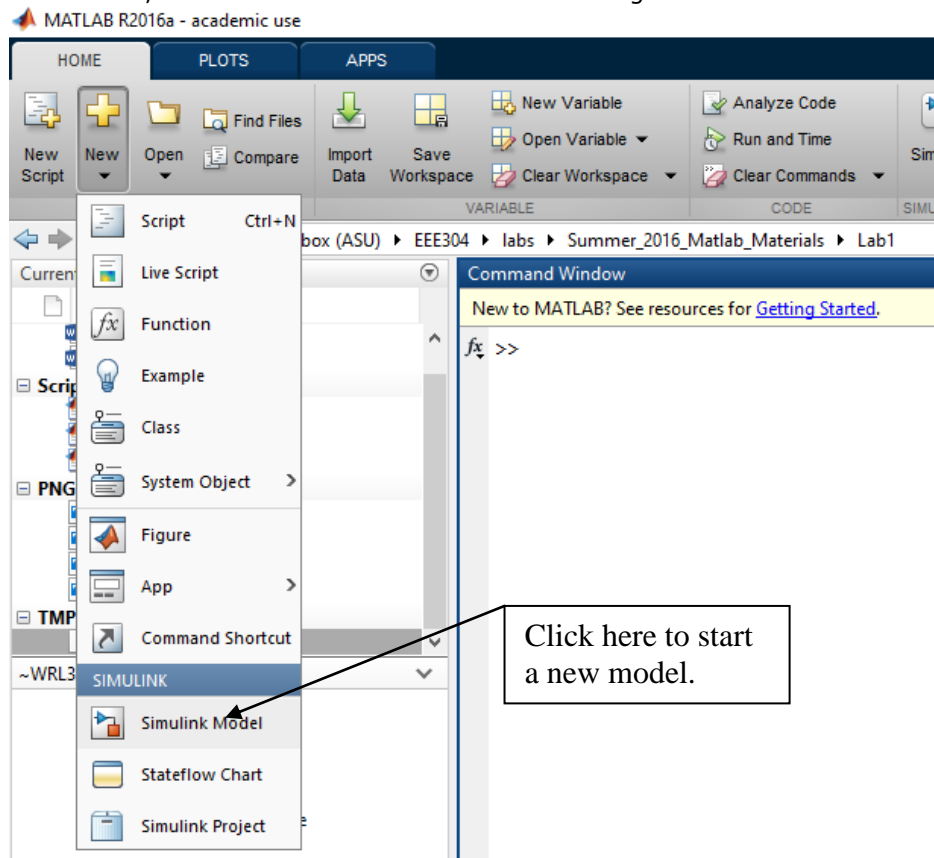


Fig 5. Creating a new Simulink model

Alternatively you may enter

```
>> simulink
```

in the MATLAB Command Window. Then a window appears and asks you to select an operation; choose "Blank Model". The SIMULINK model window will open and it will be empty, see figure 6.

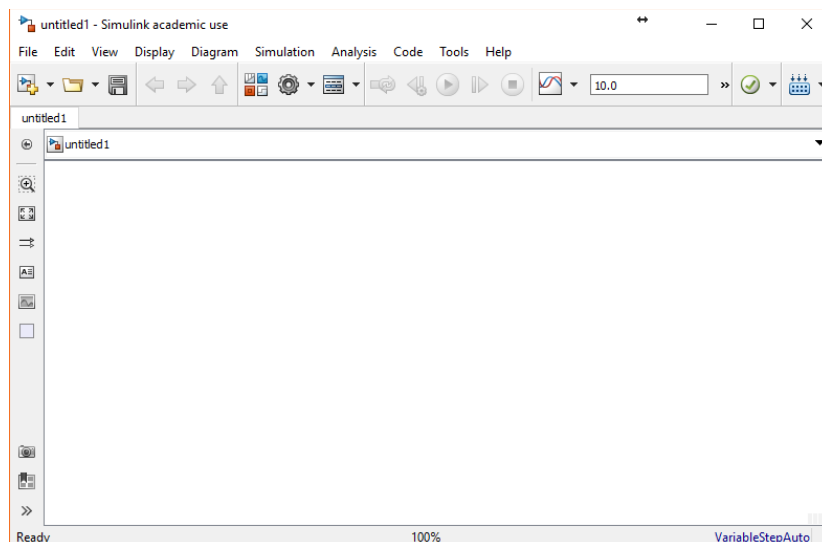
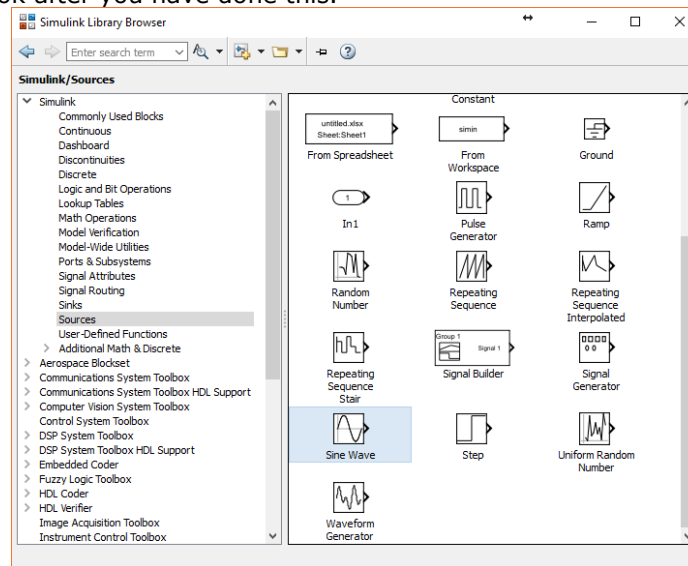


Fig 6. Blank Simulink model

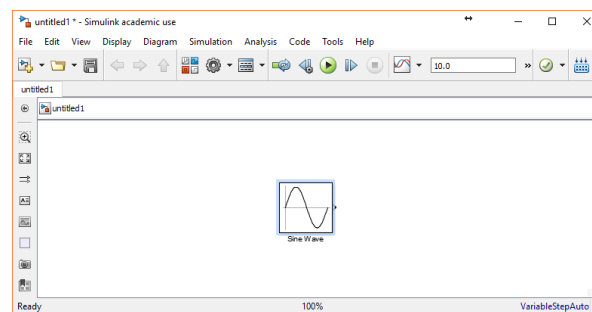
In order to complete the model, you then need to copy blocks into new model file (.slx) by opening the "library browser" window. Go to *View -> Library Browser*. Then the blocks needed can be found from the following SIMULINK block libraries:

- Sources library (the Sine Wave block)
- Sinks library (the Scope block)
- Continuous library (the Integrator block)
- Signal Routing library (the Mux block)

Copying is performed either by drag-and-drop, or with the standard Windows commands (ctrl-C, ctrl-V). You can copy a Sine Wave block from the Sources library, using the Library Browser (Windows only) or the Sources library window (UNIX and Windows). To copy the Sine Wave block from the Library Browser, first expand the Library Browser tree to display the blocks in the Sources library. Do this by clicking the Sources node to display the Sources library blocks. Finally, click the Sine Wave node to select the Sine Wave block. Here is how the Library Browser should look after you have done this.



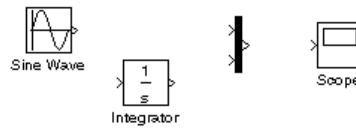
Now drag a copy of the Sine Wave block from the browser and drop it in the model window. Alternatively, you may right-click on the "sine wave" block and select "Add block to model ..."



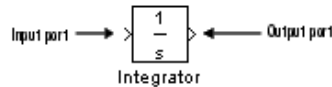
Copy the rest of the blocks in a similar manner from their respective libraries into the model window. You can move a block from one place in the model window to another by dragging the block. You can move a block a short distance by selecting the block, then pressing the arrow keys.

TIP: Another method of adding blocks introduced in R2016a is to simply click in the model and start typing the name of the block. A search bar will open and sort search the blocks based on the typed character.

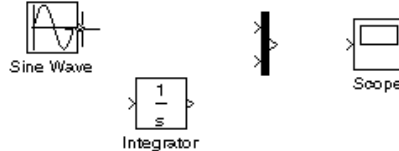
With all the blocks copied into the model window, the model should look something like this.



If you examine the blocks, you see an angle bracket on the right of the Sine Wave block and two on the left of the Mux block. The > symbol pointing out of a block is an output port; if the symbol points to a block, it is an input port. A signal travels out of an output port and into an input port of another block through a connecting line. When the blocks are connected, the port symbols disappear.

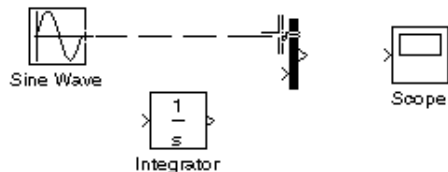


Now it's time to connect the blocks. Connect the Sine Wave block to the top input port of the Mux block. Position the pointer over the output port on the right side of the Sine Wave block. Notice that the cursor shape changes to crosshairs.

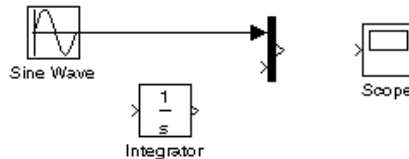


Hold down the mouse button and move the cursor to the top input port of the Mux block.

Notice that the line is dashed while the mouse button is down and that the cursor shape changes to double-lined crosshairs as it approaches the Mux block.



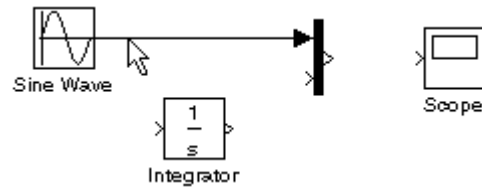
Now release the mouse button. The blocks are connected. You can also connect the line to the block by releasing the mouse button while the pointer is over the block. If you do, the line is connected to the input port closest to the cursor's position.



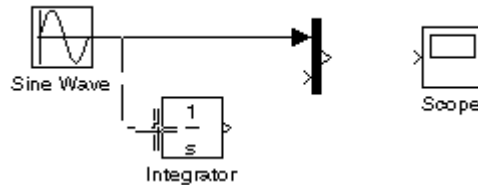
If you look again at the model at the beginning of this section (see Building a Model), you'll notice that most of the lines connect output ports of blocks to input ports of other blocks. However, one line connects a line to the input port of another block. This line, called a branch line, connects the Sine Wave output to the Integrator block, and carries the same signal that passes from the Sine Wave block to the Mux block.

Drawing a branch line is slightly different from drawing the line you just drew. To weld a connection to an existing line, follow these steps:

1. First, position the pointer on the line between the Sine Wave and the Mux block.

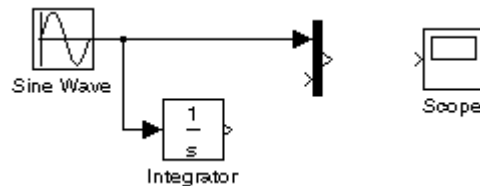


2. Press and hold down the Ctrl key (or click the right mouse button). Press the mouse button, then drag the pointer to the Integrator block's input port or over the Integrator block itself.

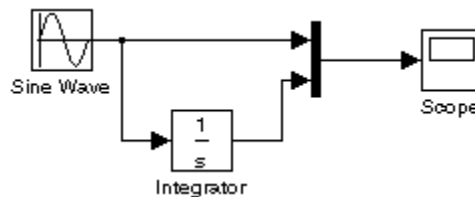


3. Release the mouse button. SIMULINK draws a line between the starting point and the Integrator block's input port.

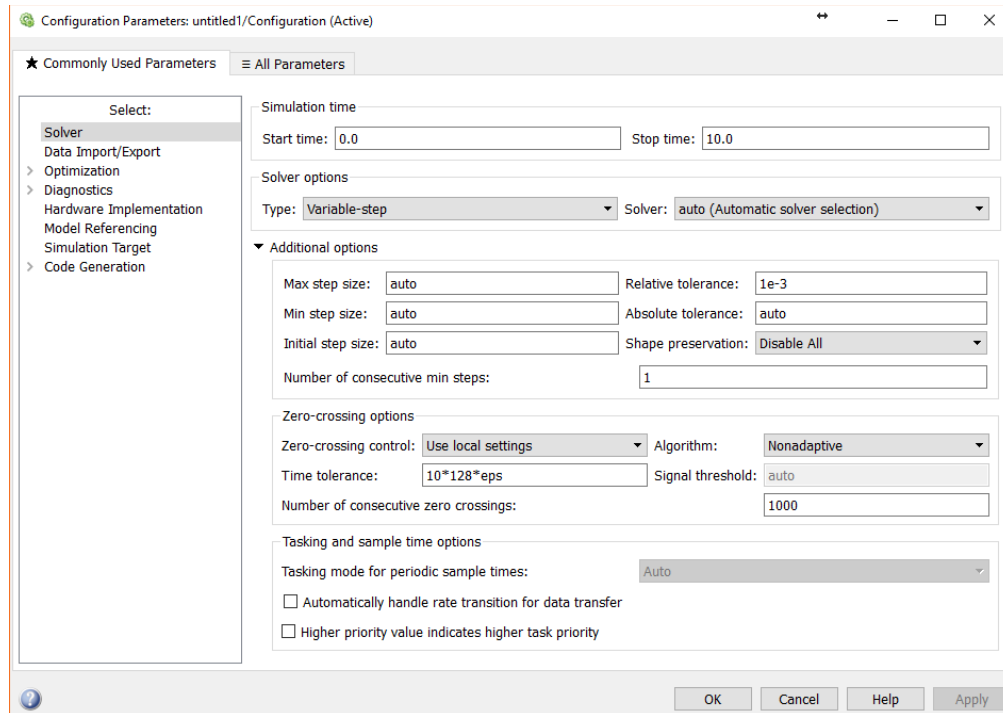
Note that an alternative procedure, not using the ctrl key, is to begin from the input port of the integrator and connect to a point on the sine wave signal line. However, if the final point is not exactly on the signal line the connection is not made and the input line remains dashed.



Finish making block connections. When you're done, your model should look something like this.

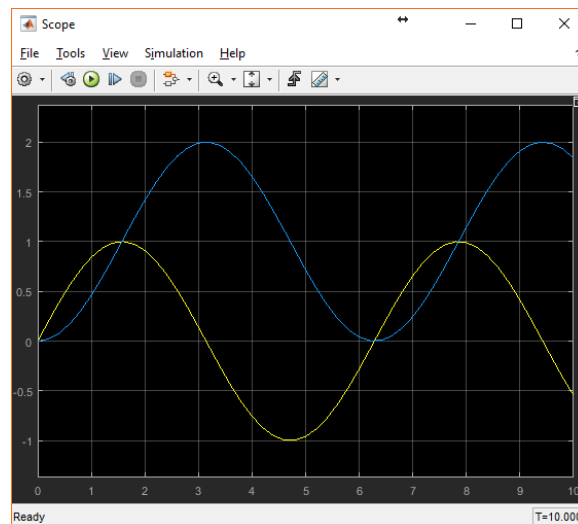


Now set up SIMULINK to run the simulation for 10 seconds. First, open the Configuration Parameters dialog box by choosing "[Model Configuration Parameters](#)" from the "Simulation" menu. On the dialog box that appears, notice that the Stop time is set to 10.0 (its default value).



Close the Configuration Parameters dialog box by clicking the OK button. SIMULINK applies the parameters and closes the dialog box.

Now double-click the Scope block to open its display window. Finally, choose Start from the Simulation menu and watch the simulation output on the Scope.



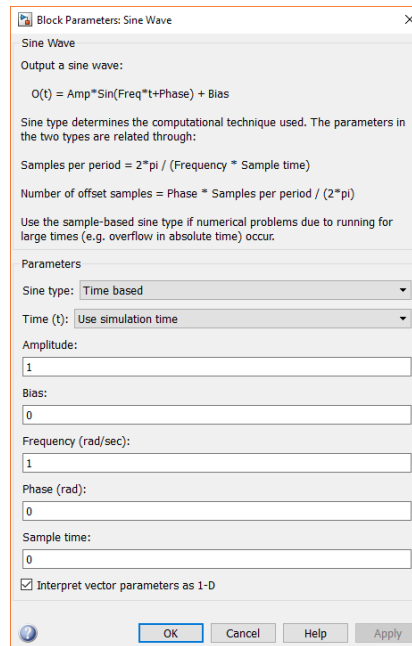
The simulation stops when it reaches the stop time specified in the Configuration Parameters dialog box or when you choose Stop from the Simulation menu or click the Stop button on the model window's toolbar (Windows only).

To save this model, choose Save from the File menu and enter a filename and location. That file contains the description of the model.

To terminate SIMULINK and MATLAB, choose Exit MATLAB (on a Microsoft Windows system) or Quit MATLAB (on a UNIX system). You can also enter quit in the MATLAB Command Window. If you want to leave SIMULINK but not terminate MATLAB, just close all SIMULINK windows.

Parameter setting

For the Sine Wave block, there are some adjustable parameters. By double mouse click on the component, there will be a dialog shown. You can adjust the parameters in this dialog.



The image shows a MATLAB/Simulink dialog box titled "Block Parameters: Sine Wave". It contains the following information and controls:

- Sine Wave**
- Output a sine wave:
- $$O(t) = \text{Amp} * \sin(\text{Freq} * t + \text{Phase}) + \text{Bias}$$
- Sine type determines the computational technique used. The parameters in the two types are related through:
 - Samples per period = $2 * \pi / (\text{Frequency} * \text{Sample time})$
 - Number of offset samples = $\text{Phase} * \text{Samples per period} / (2 * \pi)$
- Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.
- Parameters**
 - Sine type: **Time based** (dropdown menu)
 - Time (t): **Use simulation time** (dropdown menu)
 - Amplitude:
 - Bias:
 - Frequency (rad/sec):
 - Phase (rad):
 - Sample time:
 - ☒ Interpret vector parameters as 1-D
- Buttons: **OK**, **Cancel**, **Help**, **Apply**