ELEC 342 Lab 2: The Discrete Time Fourier Transform and Introduction to Simulink

In Part I of this lab, the discrete time Fourier transform of a pulse input will be implemented using loops and arrays.
In Part II, the basics of Simulink, a simulation environment within MATLAB will be introduced.

PART I

Introduction:

In this portion, we will explore the MATLAB implementation of the Discrete Time Fourier Transform which is defined by the equation

$$X(e^{j\omega}) = \sum_{n=-\infty}^{n=+\infty} x[n]e^{-j\omega n}$$

Note that this equation is equation 11.28 appearing in page 477 of the course textbook, with *k* replaced by *n* and Ω replaced by *w*.

The following pseudocode algorithm may be used to obtain the DTFT of a given signal x[n]:

```
1. Define the signal x[n] over some interval   i <= n <=  j

2. Define the range of frequencies (w) over which the transform
   will be computed.

3. For every value of w
    {
        set sum = 0 ;
        for every value of  n
        {
           sum = sum + ( x[n] * e -jwn )
         }
       store sum associated with a value w into an array
    }
```

The values stored in the array will be complex numbers (since in general e^{-jw} is a complex number). The MATLAB plot command will ignore any imaginary parts of complex numbers and plot only the real parts. For example, if we have an array of three complex numbers:

```
>> mycomplex = [ 1 + j*1   1 + j*0.5   1  + j*0.2 ]

mycomplex =

   1.0000 + 1.0000i   1.0000 + 0.5000i   1.0000 + 0.2000i
```

and if we attempt to directly plot these values for the following three values of w (frequency):

```
>> w = [ 0, 0.5, 2*pi]

w =

         0    0.5000    6.2832
```

by using the `plot` command:

```
>> plot(w, mycomplex)
Warning: Imaginary parts of complex X and/or Y arguments ignored
```

MATLAB will ignore the imaginary parts of every complex number in the array `my_complex` and plot only the real parts (which in this example are all equal to 1) generating the plot shown in Figure 1:

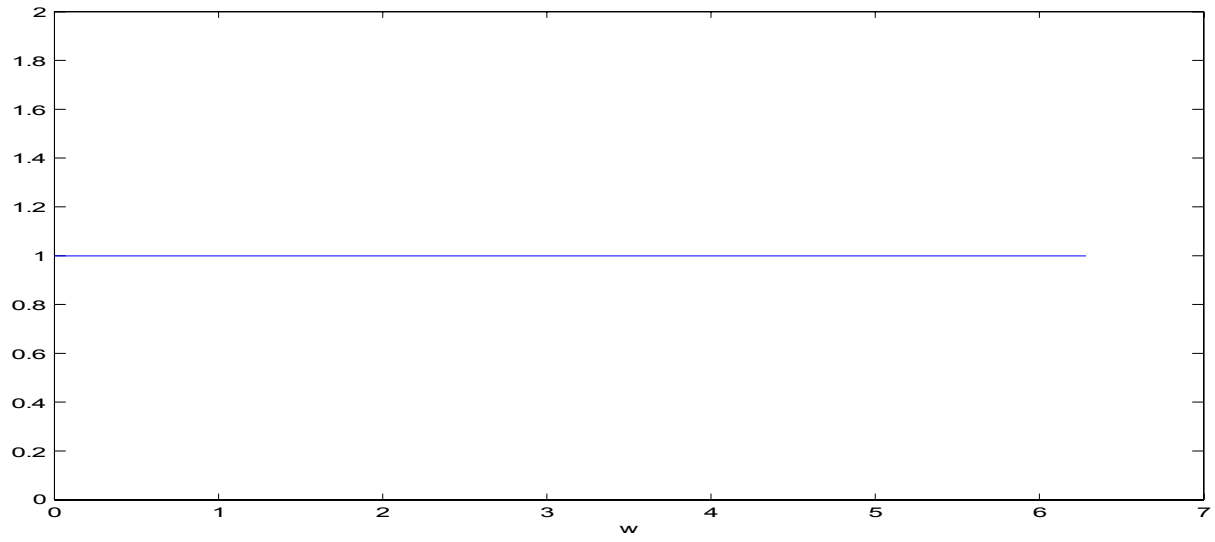

Figure 1: Plot obtained in which imaginary components are ignored.

When plotting the DTFT, we must obtain the **magnitude** of the complex number. This can be readily done using the abs function as in:

```
>> complex1 = 1 + j*1
complex1 =

   1.0000 + 1.0000i

>> abs(complex1)

ans =

    1.4142
```

Returning to our example array of three complex numbers, the magnitudes may be saved into a new array and plotted by performing:

```
>> mag_of_mycomplex = abs(mycomplex)

mag_of_mycomplex =

    1.4142    1.1180    1.0198

>> plot(w, mag_of_mycomplex)
```

Note how in this case, no warnings concerning the omission of imaginary components is generated.

Some helpful array related functions:

The MATLAB language includes several very helpful functions related to arrays. The following presents some examples of the more frequently used [2].

size:

The `size` function returns the size of each array dimension. For example:

```
>> my_array = [ 1 2 3 ]

my_array =

     1     2     3

>> size(my_array)

ans =

     1     3
```

The answer returned by `size(my_array)` signifies that the array my_array consists of 1 row and 3 columns.

length:

The `length` function returns the size of the largest array dimension; for a row vector the function will return the number of elements in the vector:

```
>> my_row_vec = [ 1 ,2 , 3, 4, 5 ]

my_row_vec =

     1     2     3     4     5

 length(my_row_vec)

ans =

     5
```

Question 1: (25/80)

Obtain the DTFT of a pulse $x[n]$ shown in Figure 2 over the interval $-10 \leq n \leq 10$.

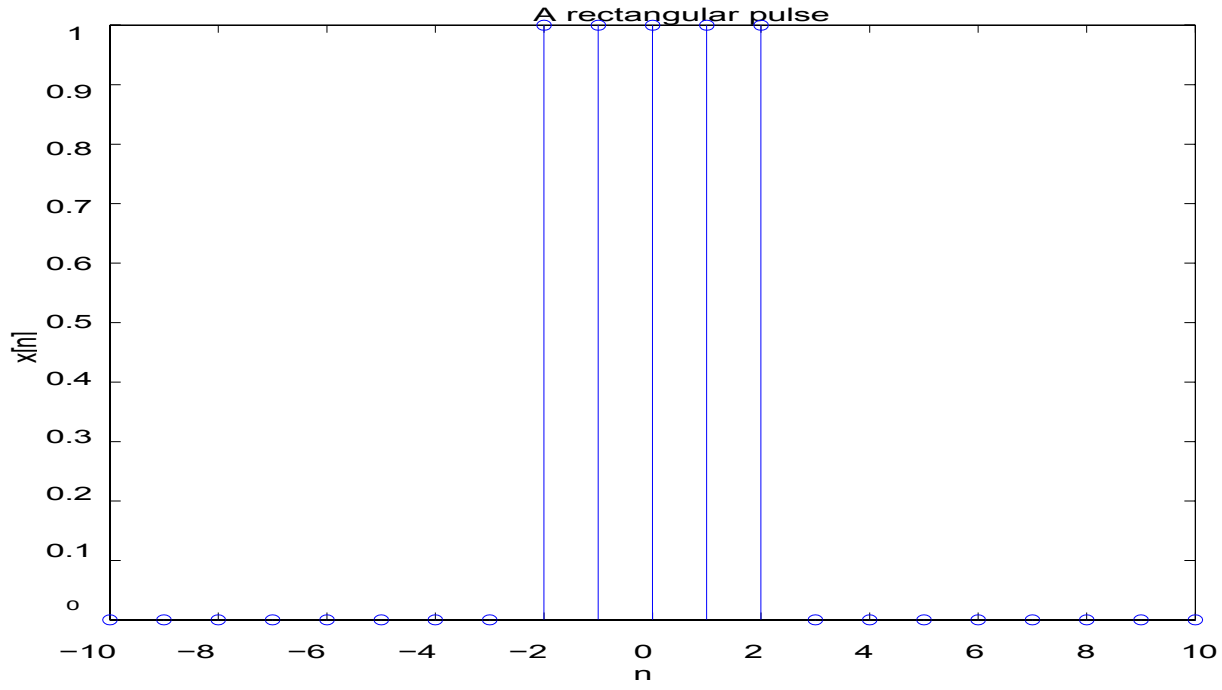

Figure 2: A rectangular pulse.

Compute the transform over the frequency interval $-\pi \leq w \leq \pi$ using a user-defined input value for the step size. Plot the signal $x[n]$ (using `stem`) together with its transform (using `plot`). Use the `length` function to control the number of iterations of the outer for loop given in the pseudocode algorithm.

Question 2: (25/80)

MATLAB has a built-in function called `fft` which computes the discrete Fourier transform of an input vector $x(n)$:

```
>> help fft
 fft Discrete Fourier transform.
    fft(X) is the discrete Fourier transform (DFT) of vector X.

 For length N input vector x, the DFT is a length N vector X,
    with elements
                      N
        X(k) =       sum  x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k
<= N.
                     n=1
```

Re-write your script for Question 1 such that the transform is computed over the interval of $0 \leq w \leq 2\pi$ and compare the results of your transform with that produced by the MATLAB `fft` command. Note that if the input signal $x[n]$ is a vector of length 21 (21 values of n ranging from -10 to +10), then x_fft = fft(x) will be a 1 x 21 array. In order to compare the results of `plot(w, x_fft)` with the results of your transform, it will be necessary to choose a step size for w such that the size of the array holding your transform values also has length 21.

Question 3: (10/80)

MATLAB has a function called `ifft` which computes the inverse discrete Fourier transform. Compute the `fft` of the input signal $x[n]$ given in Question 1, and then use it as input to the `ifft` function to obtain the original signal $x[n]$. Plot the original signal $x[n]$ together with the signal obtained from the `ifft` command.

PART II

Simulink

Simulink is a modelling and computer simulation environment integrated within MATLAB. It can be used to simulate both continuous and discrete systems as well as hybrid systems containing both continuous components and discrete components. Simulink models continuous systems with differential equations describing the physical system and with difference equations for discrete systems. Simulink allows an engineer to rapidly design and model a system to determine its response to various inputs. [3]

Getting Started

1. To start Simulink, type `simulink` from the MATLAB prompt, or select with the left mouse button the Simulink Library icon located in the middle of the top portion of the Matlab window:

```
>> simulink
>>
```

The Simulink Library Browser will appear as shown in Figure 3:

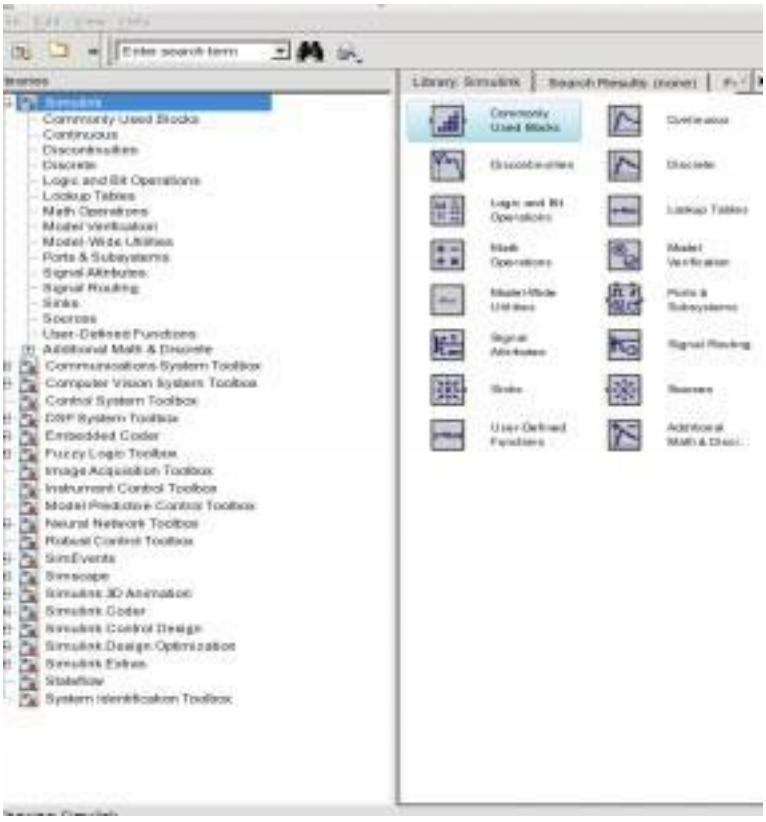

Figure 3: Simulink Library Browser

2. A simple Simulink model consisting of a Sine input Source block and a Scope Sink block will be created. [4] To start editing the model, select **File -> New -> Model** from the Simulink Library Browser. An empty Model window will appear as shown in Figure 4:

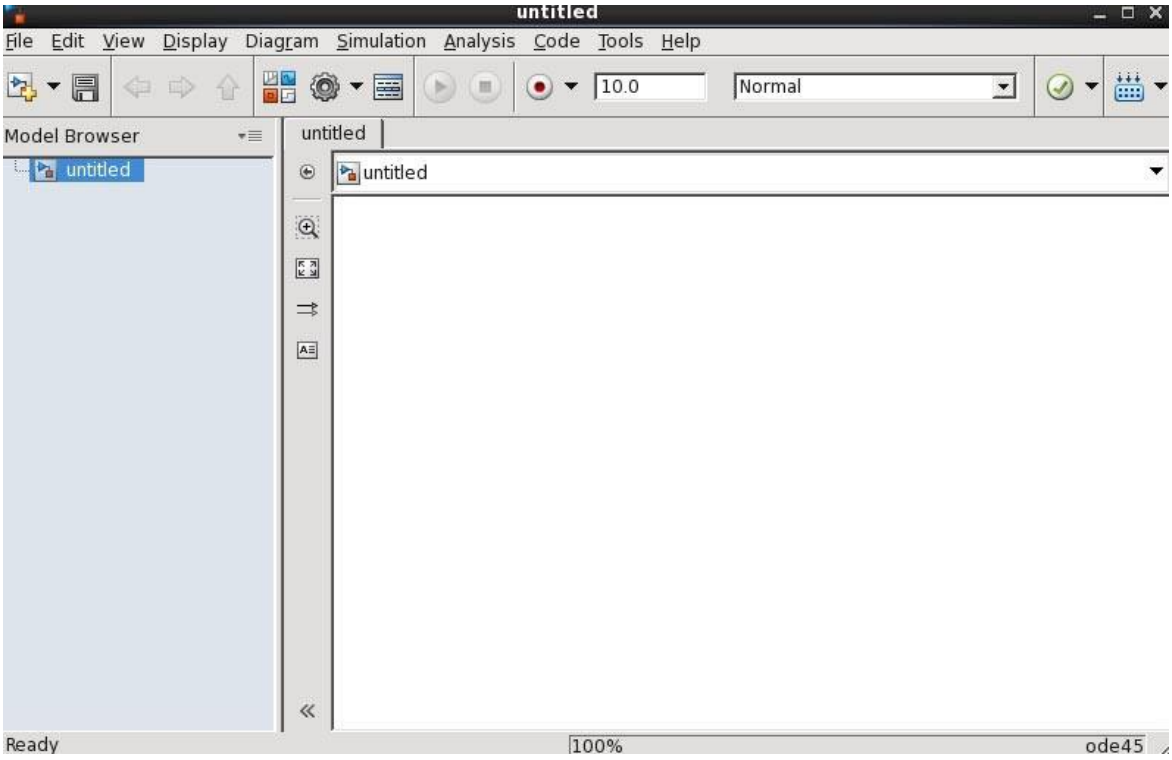

Figure 4: New Simulink Model window.

3. Select **Sources** from the Simulink Library Browser, select with the mouse the Sine Wave icon and drag it to the Model window.

4. Select **Sinks** from the Library Browser window, select the Scope icon and drag it to the Model window. The Model window will now contain a Sine Wave source and a Scope sink block as shown in Figure 5.

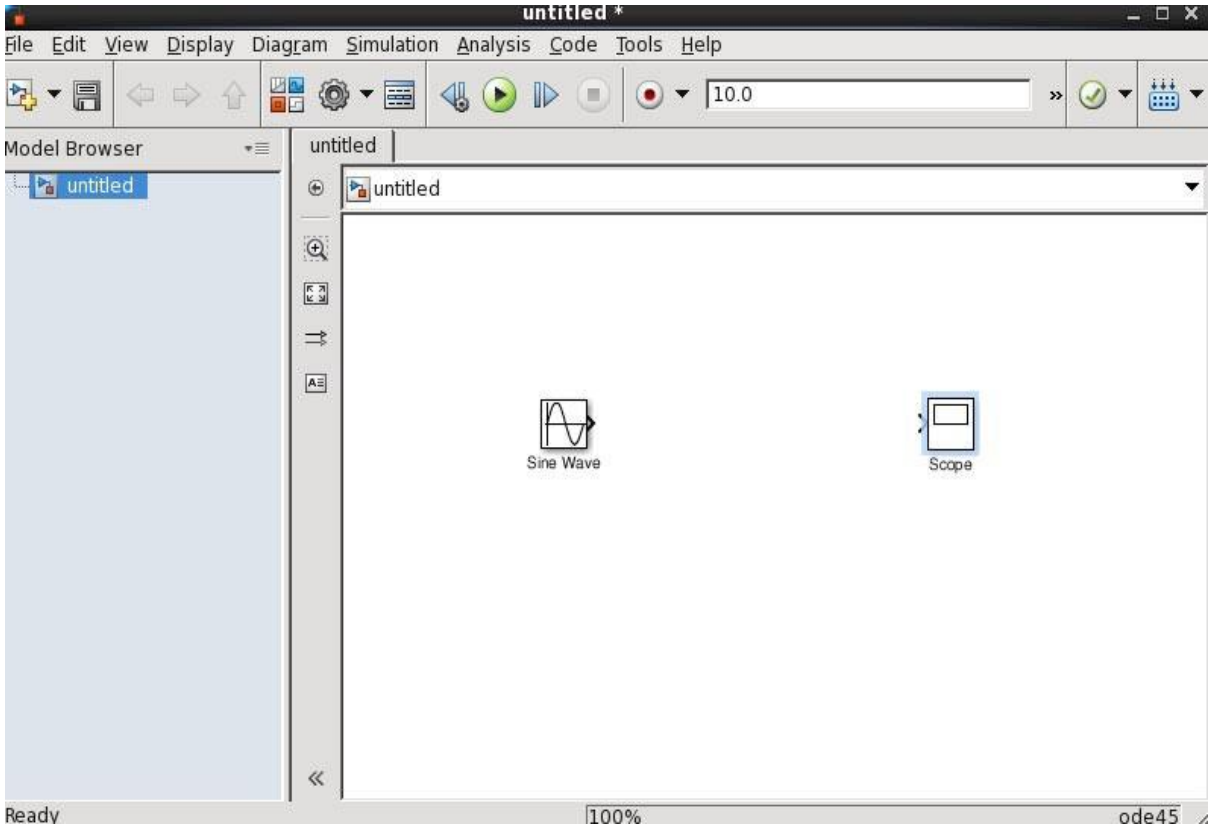

Figure 5: Model window with Sine and Scope blocks.

5. To connect the Sine wave source block to the Scope sink block, move the mouse cursor to the output port of the Sine block (the cursor will turn to a cross hair once it is over the port) , left click the mouse and drag it to the input port of the Scope block and left click. A wire will be drawn connecting the two blocks as shown in Figure 6.

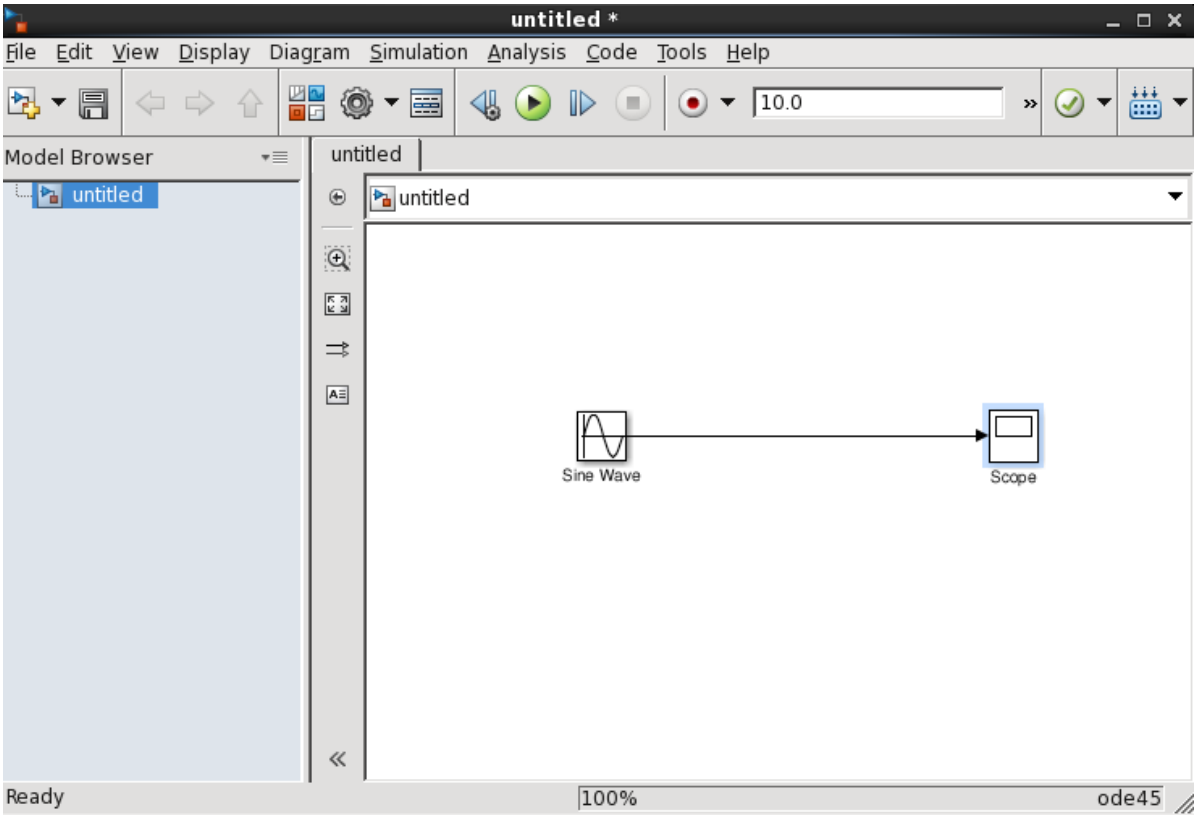

Figure 6: Sine source connected to Scope sink with a wire.

6. The next step is to assign a frequency of w = 2*pi*f = 2*pi*(1/150) to the Sine Wave source. Double click the Sine Wave block to open the Block Parameters window and enter the value of 2*pi*(1/150) for the Frequency as indicated in Figure 7. Select OK to enter the value and close the window.

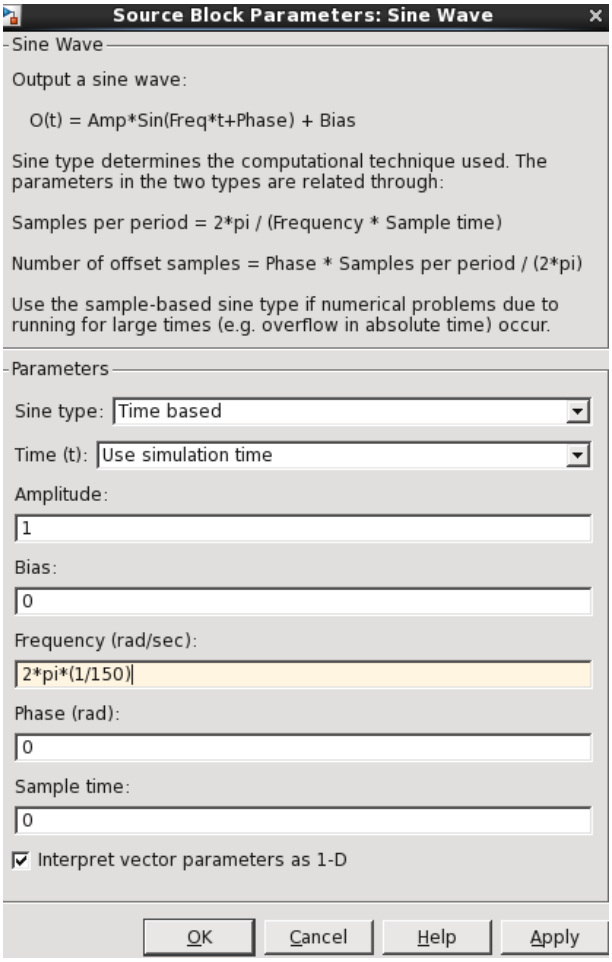

Figure 7: Source Block Parameters window with frequency specified.

7. The MATLAB data type of the Source block signal may be indicated by selecting **Display -> Signals & Ports -> Port Data Types** in the Simulink Model window. You will note that the wire representing the signal from the Sine Wave block is now labelled "double".

8. The next step is to specify the simulation stop time and to set some options which control the simulation. From the Simulink Model window select **Simulation -> Model Configuration Parameters**. In the Configuration Parameters window specify the following (refer to Figure 8):

Stop Time: 150 (this represents one period of the signal)
Type: Fixed-step
Solver: discrete (no continuous states)
Tasking mode for periodic sample times: SingleTasking

Save these parameters by selecting OK.

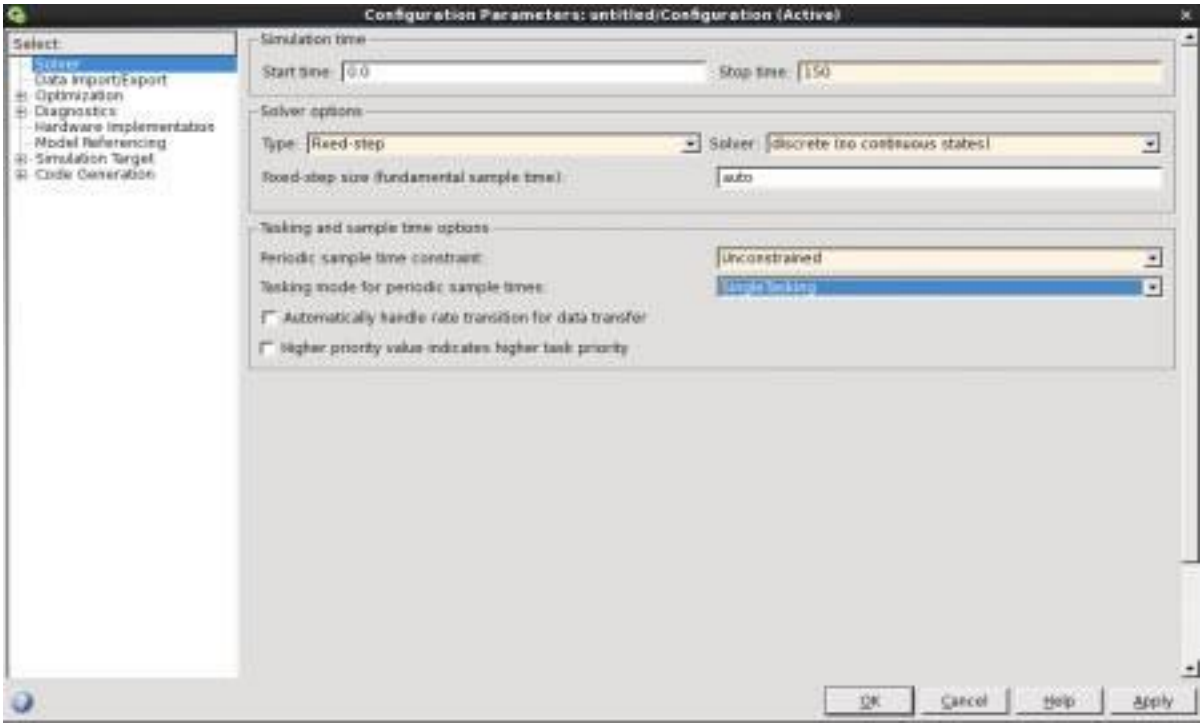

Figure 8: Setting the Configuration Parameters.

9. The Scope block will be configured to display one period of the signal and to autoscale the y-axis so that the waveform fills the display. Double click on the Scope block icon in the Model window to open the Scope block window as shown in Figure 9.

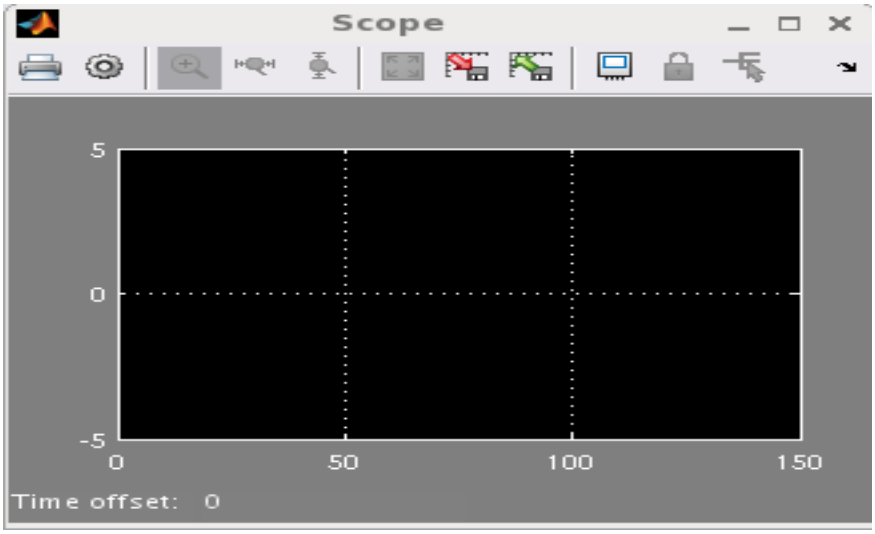

Figure 9: Scope block window.

In the Scope block window, select the Parameters icon (the second one from the top left, next to the printer icon)

10. In the Scope Parameters (refer to Figure 10), specify:

Time range: 150

Select OK.

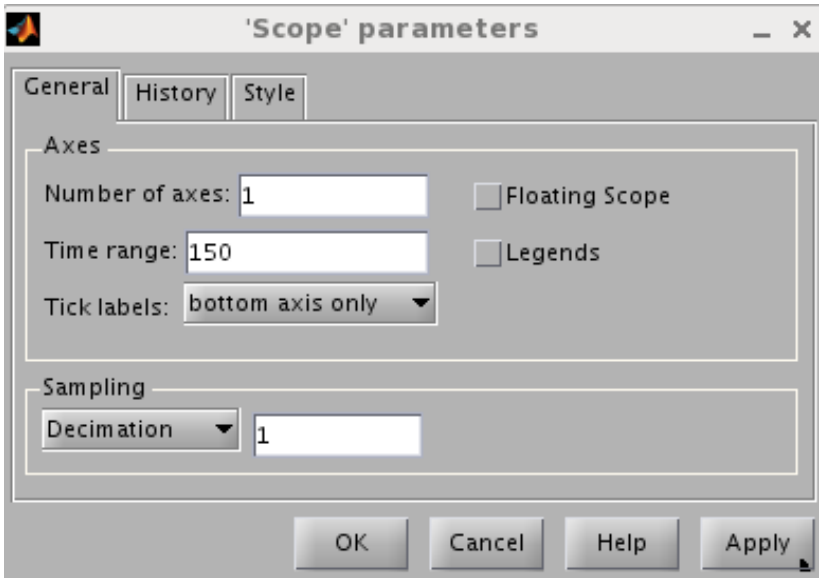

Figure 10: Setting the Scope Parameters.

11. To run the simulation, select **Simulation -> Run** from the Simulink Model window. The Scope block will be updated to show the simulation results (one period of the sine wave). To autoscale the y-axis of the Scope block, select the Autoscale icon (the 6th from the left). The y-axis will be scaled so that the waveform occupies the whole display as shown in Figure 11.

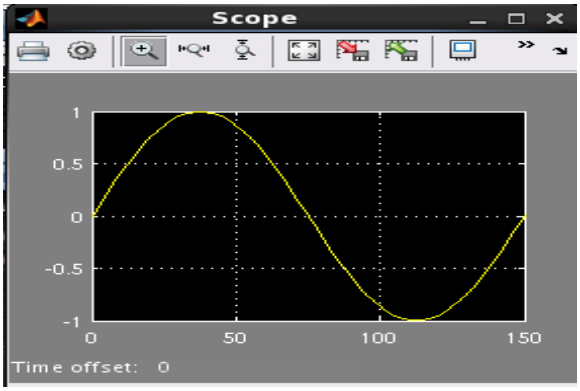

Figure 11: Scope block showing the simulation results.

12. To save the model, select File -Save from the Model window and specify a file name (such as sine_source) and Files of Type: Simulink Models(*.mdl) and select Save. If you wish to open the model at some future time, you may specify File -> Open and specify the filename.

13. We can specify a different value for the sample time of the Sine block (the default is 0 which causes the block to operate in continuous mode) by opening the Sine block and entering a non-zero value for the **Sample time.** Recall that we specified a sine wave with a period of 150s, if we enter 5 for the sample time, the block will now operate in discrete mode, producing a new sample every 5 seconds. It is readily apparent as we increase the sample time (fewer sample per period), the greater the quantization error will be. Enter 5 as the **Sample time** and select OK and re-run the simulation to see the effects of sampling (refer to Figure 12).

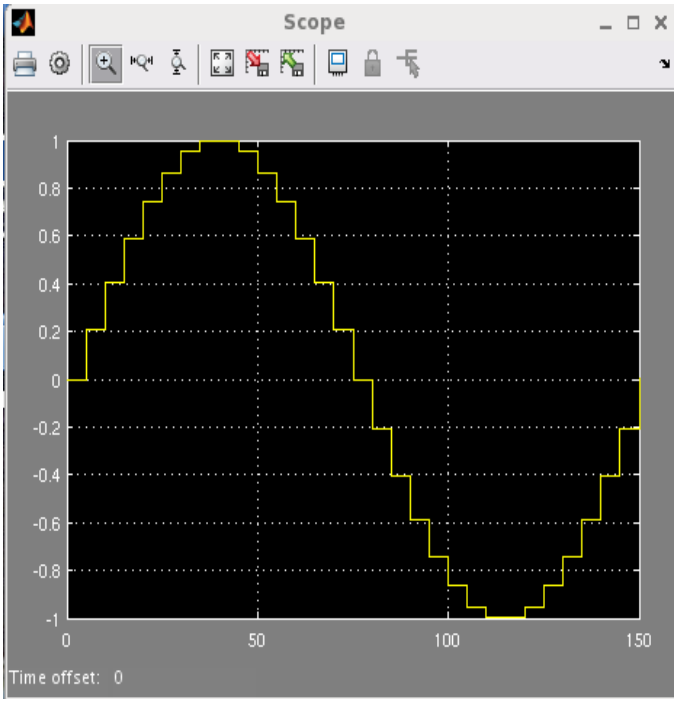

Figure 12: Simulation results using a Sample Time of 5.

Questions

Question 1: (20/80)

Use Simulink to simulate the behavior of the difference equation $y[n] = x[n] + \frac{1}{4}y[n-1]$. Use a sine wave as the input $x[n]$. Decide upon a frequency and sample time of your choice. Use the following Simulink blocks to build the Simulink model of the system as shown in Figure 13:

• an **Integer Delay** block from the **Discrete** library.

• a **Gain** block from the **Math Operations** library (double click the block to specify the gain parameter as 0.25)

• an **Add** block chosen from the **Math Operations** library

• A **Sine Wave** Source block and a **Scope Sink** Block

Connect the blocks as shown in Figure 13.

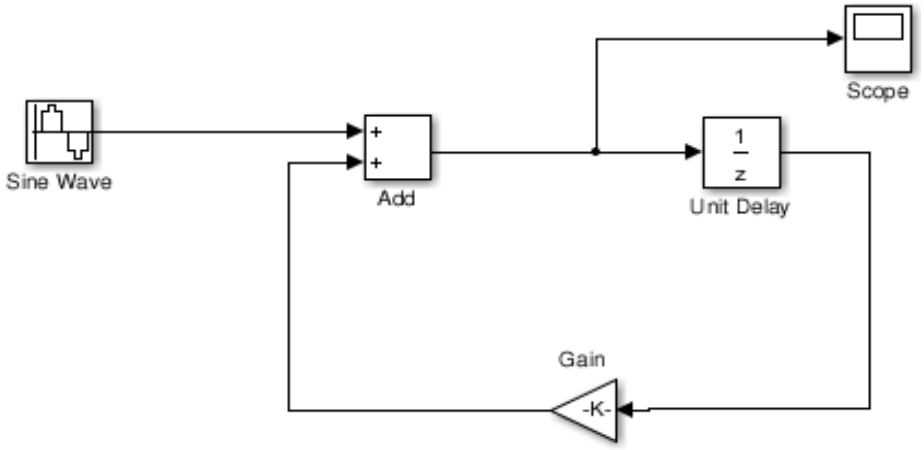

Figure 13: Simulink model of $y[n] = x[n] + \frac{1}{4}y[n-1]$.

References

1. *Continuous and Discrete Time Signals and Systems*, M. Mandal and A. Asif, Camebridge University Press, ISBN 0-521-85455-5, 2007, p.477.

2. *MATLAB Programming*, David C. Kunicky, Pearson Education Inc., 2004.

3. *Mastering Simulink*, James B. Dabney, Thomas L. Harman, Prentice-Hall, ISBN 0-13-142477-7, 2004, p.1.

4.. *Getting Started with System Generator for DSP Lab 1* - Using Simulink, Software Documentation within SysGen installtion, Xilinx Inc.