ELEC 342 Lab 1: Additional MATLAB features, Properties of Signals and Systems, Convolution and System Response

Revised 2023-August

Part I of this lab will introduce more features of the MATLAB programming language: looping and conditional selection, as well as array processing features. Various properties of signals and systems such as **linearity, evenness**, and **oddness** will be verified using simple MATLAB scripts. In Part II, the MATLAB convolution function will be used to determine a system's response to an input signal.

PART I

The `for` loop:

The general form of a `for` loop in MATLAB is:

```
for variable = index_expression
  statement1;
  statement2;
  statement3;
  ...
  statementn;
end
```

Example 1: The following for loop computes the sum of all the integers between 1 and 10:

```
% finds the sum of the integers from 1 to 10
clear
sum = 0;  % initialize running sum
for i =  1 : 10
  sum = sum + i;
end
disp('The sum is = ' )  % output a descriptive string
disp(sum)               % followed by the value of sum
```

The `disp` command is used to display the value of a variable (or a string enclosed in single quotes).

Example 2: The following loop finds the sum of the squares of the integers from 1 to 5, sum = $1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 1 + 4 + 9 + 16 + 25 = 55$.

```
clear;
sum = 0; % initialize sum to 0
for index =   1 : 5  % set up the loop index
  square = index ^ 2; % compute square of current loop index and
  sum = sum + square; % add it to the running sum
end
disp('The sum of the squares of 1 to 5 is')
disp(sum)
```

Example 3: MATLAB allows an array to be specified as the index_expression. The code in Example 2 may be rewritten more concisely using arrays as:

```
clear;
x = [ 1 : 5 ];   % intialize the x array
y = x .^2  ;     % compute the squares of the elements in the
                 % x array
sum = 0;
for index = y    % loop through every element in the y array
  sum = sum + index;  % and add each element to sum
end
disp('The sum of the squares of 1 to 5 is')
disp(sum)
```

The conditional (`if`) statement:

A conditional statement is used to change the flow of control within a script. MATLAB has two basic conditional statements: the `if` and the `switch`. Some examples of the variations of the if statement are presented.

Example 1: The form of a basic `if` statement is:

```
if ( condition )
  statement1
  statement2
  ...
  statementn
end
```

The condition to be tested is formed using the relational operators:

```
<   less than
<=  less than or equal
>   greater than
>   greater than or equal
==  equal
~=  not equal
```

Two conditions may be combined using the `&` (logical AND) and the `|` (logical OR) operators to form more complex conditions such as:

```
 ( x < 25 ) & ( y > 0 )
```

If the specified condition is true, the statements within the if are executed, otherwise the statements are not executed.

Example 2: A simple `if` statement together with the `input` statement

```
clear
number = input('Enter a number ')
if ( number < 0 )
  disp('Number is negative')
end
```

This example also introduces the `input` statement which displays the specified string and reads input from the keyboard and assigns it to the variable specified on the left-hand side of the = operator.

The output produced for two different values of input are:

```
Enter a number 2

number =

     2
```

Running the script, a second time and entering -4 results in:

```
Enter a number -4

number =

    -4

Number is negative
```

Example 3: `If - else` statement

An `if` statement may have an `else` branch as in:

```
clear
number = input('Enter a number ')
if ( number < 0 )
```

```
  disp('Number is negative')
else
  disp('Number is positive')
end
```

Only one of the two `disp` statements will be executed depending upon the value provided as input.

Example 4: When comparing non-integer numbers, one should never compare for direct equality as round-off errors in the representation of some number may lead to unexpected results as in:

```
clear
x = 0.1 + 0.1 + 0.1
if ( x == 0.3)
 disp( ' x is equal to 0.3')
else
 disp( ' x is not equal to 0.3')
end
```

`The output produced is:`

```
x =

    0.3000

 x is not equal to 0.3
```

Rather than comparing for direct equality, we should check that the absolute value of the difference between x and 0.3 is less than some threshold value (0.001 for example):

```
clear
x = 0.1 + 0.1 + 0.1
if ( abs( x - 0.3 ) <= 0.0001)
 disp( ' x is equal to 0.3')
else
 disp( ' x is not equal to 0.3')
end
```

The program now gives the “correct” output:

```
x =

    0.3000

 x is equal to 0.3
```

The same idea can be used to “compare” whether two vectors are equal:

```
first  = [ 0.3 0.29 0.29 0.3 ]
second = [ 0.3 0.3  0.3  0.3 ]
if ( first == second )
disp('equal')
else
disp('not equal')
end
```

The output will be: `not equal`

If for our purposes we consider our accuracy to be 0.01, then we can compute the difference between the two vectors as:

```
diff = first -second
```

and then loop through every element in this `diff` vector and check if:

```
for index = diff
if ( abs(diff) > 0.01 )
disp('not equal')
end
end
```

For the values the given values of `first` and `second`, this for loop will **not** display the `'not equal'` output since the difference does not exceed 0.01.

Some Properties of Signals and Systems

Linearity:

The next example checks to see if a given system satisfies the additive property of linear systems:

given an input signal $x_1[n]$, the output response produced is $y_1[n]$ and
given an input signal $x_2[n]$, the output response produced is $y_2[n]$,
then for an input signal $x_3[n] = x_1[n] + x_2[n]$, the output response $y_3[n] = y_1[n] + y_2[n]$ [1]

Example 1: This example determines whether the system defined by $y[n] = 2x[n]$ gives outputs which are consistent with a linear system.

```
% Ted Obuchowicz
% Apr 23, 2012 16:28
% determines if the given system
% y[n] = 2 * x[n] produces outputs consistent with a linear
% system

clear
% define n
n = [ 0 : 4 ]

% define the input x1[n] = n
x1 = n

% define input x2[n] = 2* n
x2 = 2 * n

% define the response y1[n] = 2* x1[n]
y1 = 2 * x1

% define the response y2[n] = 2 * x2[n]
y2 = 2 * x2

% define x3[n] = A*x1[n] + B*x2[n]
% for simplicity we make A = B = 1 in this example

x3 = x1 + x2 ;

% define the response y3[n] = 2 * x3[n]
y3 = 2 * x3
% Check if y3[n] = y1[n] + y2[n]

if ( y3 == ( y1 + y2 ) )

disp( 'Outputs are consistent with a linear system')
else
disp( 'System is not linear')
end

% stem plots of the responses may also be obtained.
```

Even-Odd signals:

Recall that a signal x[n] is said to be [2]:

even if x[n] = x[-n] and

odd if x[n] = - x[-n]

The following example determines if the signal $sin\left(\frac{2\pi}{10}n\right)$ for n = 0, 1, 2, …, 9 is even or odd:

```
% determines if a signal x[n] = function(n)
% is even or odd

% x[n] is EVEN is  x[n] = x[-n]
% x[n] is ODD  if  x[n] = -x[-n]
% define n
n = 0 : 9

% define x1[n]
x1 = sin( 2*pi/10 * n )
subplot(1,2,1)
stem(n, x1)

% define x2 = x1[-n]
x2 = sin( 2*pi/10 * (-n)
) subplot(1,2,2)
stem(-n, x2)

if ( x2 == x1 )
 disp('EVEN')
elseif ( x2 == (-x1) )
 disp('ODD')
else
 disp('NEITHER EVEN NOR ODD')
end
```

Note that we should be checking that the absolute value of the difference between the elements of x1 and x2 is less than some threshold in case there are round-off/truncation errors.

Example 2: This example determines the even and odd components of a signal which consists of a unit step function. It illustrates how one overcomes the limitation that MATLAB does not allow for negative array indices (which are used in the representation of a discrete signal over some span of intervals such as -n to +n).

Suppose we wish to create a unit step function (x1[n]) and its mirror image (x2[n] = x1[-n]) as shown in Figure 1:

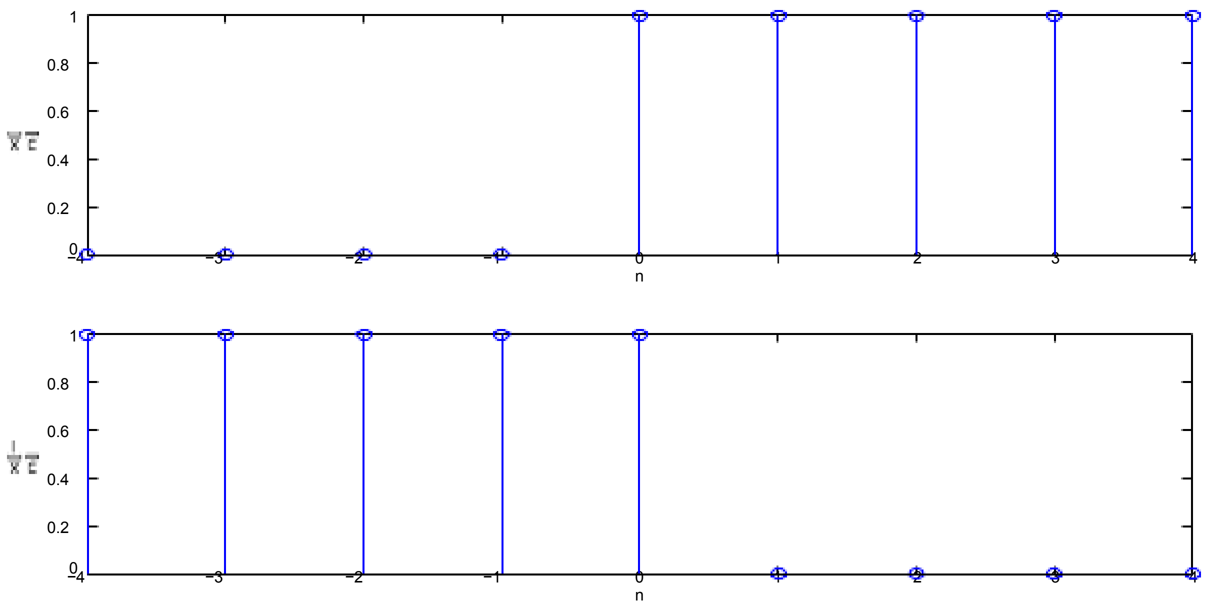

Figure 1: A unit step function and its mirror image.

We can define x1[n] with the following code:

```
% define n
n = -4  : 4
% define x1[n] as unit step
x1 = ones(1,9);    % set all 9 array elements to 1
x1(1:4) = 0;         % set the first 4 elements to 0
```

Note that `x1` is a 1 x 9 array with elements equal to x1(1), x1(2), x1(3), ... , x1(9) since in MATLAB array indices start from 1.

We may try to define x2[n] simply as:

```
x2 = zeros(1,9); % define a 1x9 array of zeros
% try to define x2[n] = x1[-n] for n = -4, -3, -2 .. 0 , 1 , 2 ,3
4
x2 = x1(-n);
```

but MATLAB will report an error concerning the existence of negative array indices:

```
Error in problems_with_negative_indices (line 17)
x2 = x1(-n);
```

We can circumvent this by noting the following:

x2(1) (the MATLAB variable) corresponds to what we call x2[-4] and
x2(2) (the MATLAB variable) corresponds to what we call x2[-3]
...

We can now assign the elements of the MATLAB variable x2 in the following manner:

```
% a brute-force approach
% not particularly elegant, but it accomplishes the task
x2(1) = x1(9)
x2(2) = x1(8)
x2(3) = x1(7)
x2(4) = x1(6)
x2(5) = x1(5)
x2(6) = x1(4)
x2(7) = x1(3)
x2(8) = x1(2)
x2(9) = x1(1)
```

It should be noted that the above 9 assignment statements may be expressed more succinctly using a `for` loop as in:

```
for index = 1 : 9
  x2(index) = x1(10 -index);
 end
```

Here the complete script file which also decomposes the original signal into its even and odd components :

```
% T. Obuchowicz
% determines if a signal x[n] = function(n)
% is even or odd
% Tue Apr 24 14:14:13 EDT 2012

% x[n] is EVEN is  x[-n] = x[n]
% x[n] is ODD  if  x[-n] = -x[n]

% define n over the interval -4 to
+4 n = [-4    : 4 ]

% define x1[n]
x1 = ones(1,9)
x1(1:4) = 0
```

```
% define x2[n] = x1[-n]
x2 = zeros(1,9)

% manually assign the values since
% MATLAB does not allow for negative array indices
% what we want to do is:
% x2[n] = x1[-n] for  n in  -4, -3, -2, -1, 0, 1 ,2 3, 4
% x2[-4] = x2[4]
% x2[-3] = x1[3]
% x2[-2] = x1[2]
% etc
% x2[1] = x1[-1]
% x2[2] = x1[-2]
% x2[3] = x1[-3]
% x2[4] = x1[-4]
% but MATLAB does not allow for negative array indices so we
% do the following instead:

 for index = 1 : 9
  x2(index) = x1(10 -index);
 end

% plot the two signals

subplot(4,1,1)
stem(n,x1)
ylabel(‘ x[n] ‘ )
xlabel(‘ n ‘ )
subplot(4,1,2)
stem(n,x2)
ylabel(‘ x[-n] ‘)
xlabel(‘ n ‘ )

% check for even/odd
% since working with integers values, we can compare for direct
equality

if ( x2 == x1 )
 disp(‘EVEN’)
elseif ( x2 == (-x1) )
 disp(‘ODD’)
else
 disp(‘NEITHER EVEN NOR ODD’)
end
```

```
% decompose the signal into its
% even and odd components
% using the definitions
% even component = (1/2) * ( x[n] + x[-n])
% odd component  = (1/2) * ( x[n] - x[-n])

even_comp = (1/2) * ( x1 + x2 )
odd_comp  = (1/2) * ( x1 - x2 )

% plot the even and odd components as a
% visual verification that the even component is indeed even
% and the odd component is indeed odd

subplot(4,1,3)
stem(n, even_comp)
xlabel(‘ n ‘ )
ylabel(‘ Even component of x[n] ‘)
subplot(4,1,4)
stem(n, odd_comp)
xlabel(‘ n ‘ )
ylabel(‘ Odd component of x[n] ‘)
```

The plots produced by this script are shown in Figure 2.

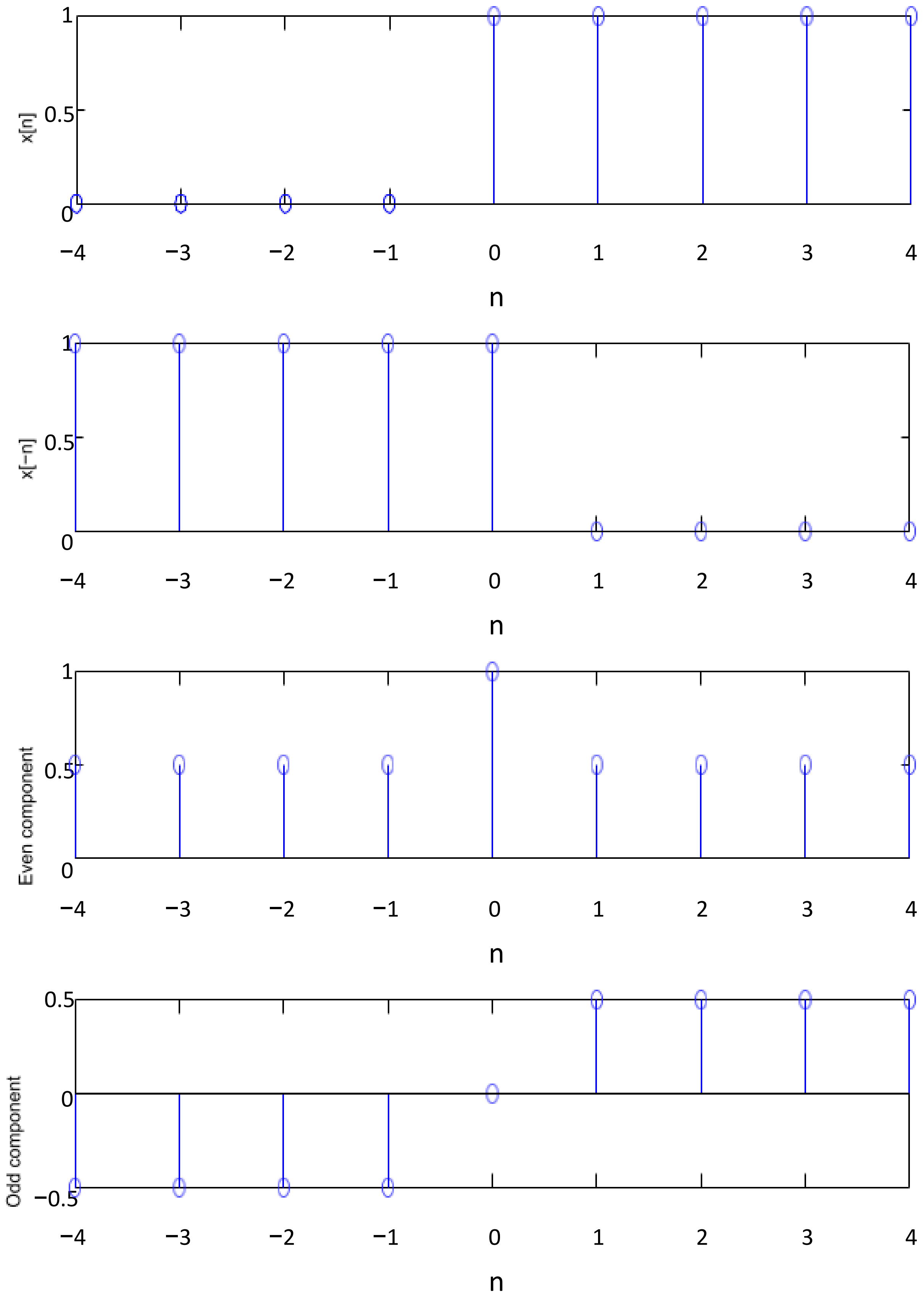

Figure 2: Results of script in Example 2.

Saving Plot outputs:

MATLAB allows the results of a plot window to be saved to a file. To save the results of the current plot window to a postscript file, use the `print` command in MATLAB as in:

```
>> print -dpsc even_odd_components.ps
```

Self-study: Use `help print` to learn what the various options (`-dpsc`) to the print command mean.

Questions

Create script files for each of the following questions. Include your name and ID as comments in each script. For each question, submit as part of the written lab report the script file in the appendix and any required plots and display values in the results and discussion section.

Question 1: (10/80)

(a) Plot the input signal $x[n] = n$ and the output signal $y[n] = x^2[n]$ over the interval n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Compute the total energy in the signal $x[n]$ and $y[n]$ (Hint: the total energy in a signal is equal to the sum of the squares of all the values contained in the signal). Use the `disp` command to display the two energies.

(b) Repeat part (a) using the input signal $x[n] = sin\left(\left(\frac{2\pi}{10}\right)n\right)$, n = 0, 1, 2, …, 9. Use the MATLAB function `sin` to compute the values of the input signal over the specified interval. Use the `help sin` facility to learn how to use the sin function.

Question 2: (20/80)

(a) Determine whether the discrete-time system which has an output $y[n] = 2x[n]$ over the interval $0 \le n \le 10$ is linear or not by determining the response $y_1[n]$ to the input signal $x_1[n] = sin\left(\left(\frac{2\pi}{10}\right)n\right)$ and the response $y_2[n]$ to the input signal $x_2[n] = cos\left(\left(\frac{2\pi}{10}\right)n\right)$.
Determine the response $y_3[n]$ to the input signal $x_3[n] = x_1[n] + x_2[n]$ and compare it with $y_4[n] = y_1[n] + y_2[n]$. Plot (using `stem`) in one graph all the input signals and their corresponding output signals. Use the `disp` command to output whether the system has 'outputs consistent with a linear system' or 'not linear'.

(b) Design an experiment to test whether the following systems are linear and time-invariant: (**Note**: *Student must answer this question in their lab report.*)

(i) $y[n] = x^2[n]$

(ii) $y[n] = 2x[n] + 5\delta[n]$

Use the input data x[n] = [0,1]. Next, using a **larger** set of values, for your choice of the input data, repeat the experiment and **analyze** and **interpret** the results obtained with this new data set. Do the new results validate or invalidate the original results obtained with x[n] = [0,1] as a choice of input data? Explain how a choice of data used may impact the results obtained.

Question 3: (10/80)

(a) Plot the following signal $x[n]$, its mirror image $x[-n]$, and its even and odd

components: $x[n] = e^{-2|n|} sin\left(\left(\frac{2\pi}{36}\right)n\right),\ 0 \leq n \leq 10$

Use the MATLAB functions `exp` and `abs`.

(b) Repeat part (a) for the signal:

$$x[n] = (-1)^{n-1},\quad -5 \leq n \leq 5$$

(c) Compare and contrast the two methods used to generate the two MATLAB arrays x1 and x2 in the following MATLAB code:

```
% T. Obuchowicz
%Fri Apr 27 16:03:27 EDT 2012

clear
n = [1 : 20 ]

x1 = sin((2*pi/40) * n) .* cos((2*pi/40) * n)

for index = 1 : 20

% Note: In MATLAB, no need to pre-allocate the array,
% unlike C++ and other high-level programming languages. x2(index)

  = sin((2*pi/40) * index) * cos((2*pi/40) * index)

end

subplot(2,1,1)
stem(n, x1)
title('Elegant method making full use of MATLABs array capabilities') xlabel('n')
ylabel('x[n]')

subplot(2,1,2)
stem(n, x2)
title('Gets the job done, but it is a lot of work and we are not in the MATLAB
mindset')
xlabel('n')
ylabel('x[n]')
```

PART II

Introduction:

Consider the transfer function defined by the following equation:

$$H(z) = \frac{1}{1-\frac{1}{4}z^{-1}}$$

From Table 13.1 of the textbook (p.572 Mandel and Asif) it can be determined that the discrete-time representation of $H(n)$ is

$$H(n) = \left(\frac{1}{4}\right)^n U[n]$$

The following difference equation can also be used to represent the system:

$y[n] = x[n] + \frac{1}{4}y[n - 1]$

Figure 3 gives the block diagram of the system.

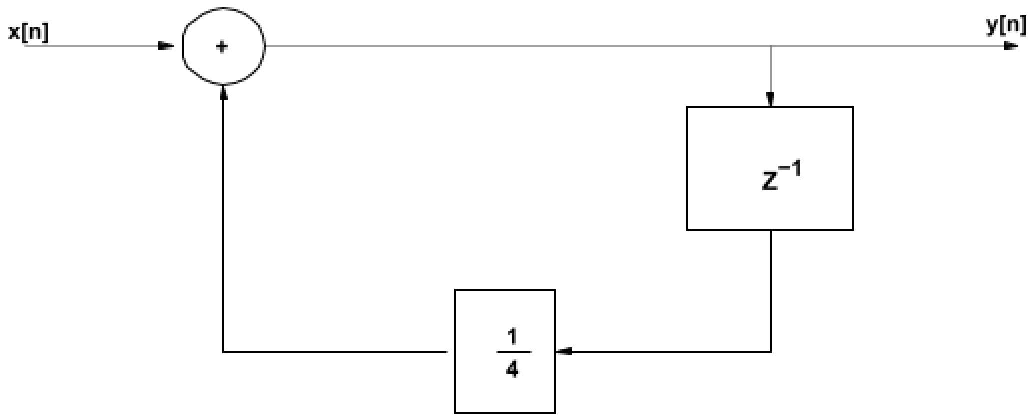

Figure 3: Block diagram of the system in question 6.

Questions

Question 1:   (10/80)

For this system, compute the response $y[n]$ over the interval $0 \leq n \leq 9$ using the input signal $x[n]$ given in Figure 4. (Using the difference equation of the system given above)

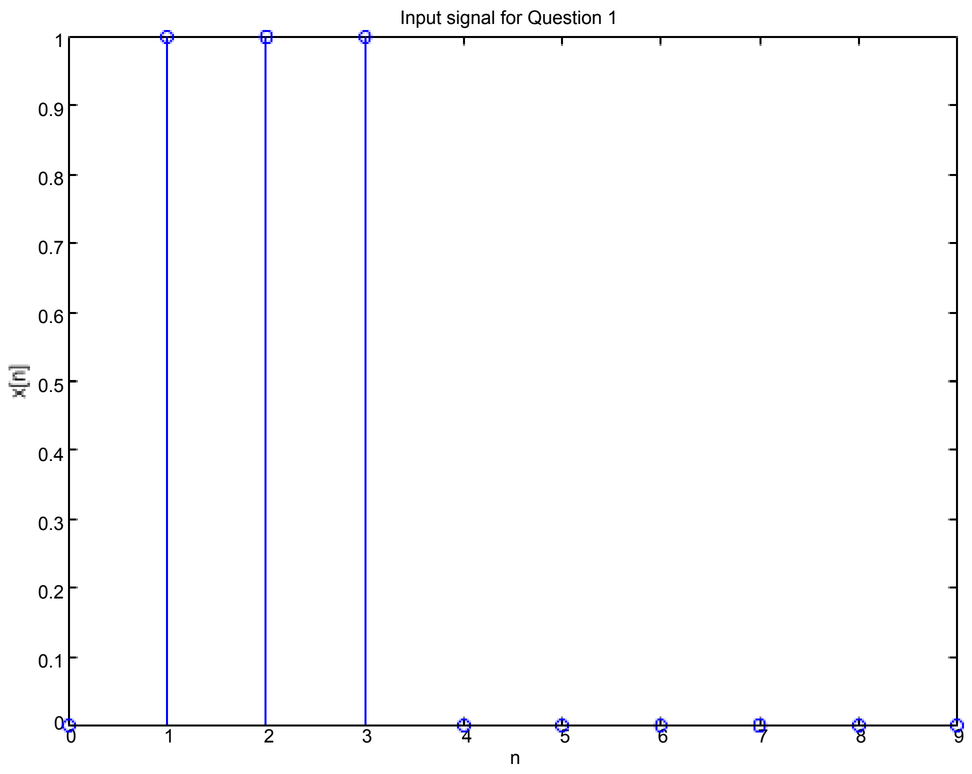

Figure 4: Input signal to be used to determine system response.

Plot the response using the `stem` function.

Question 2:   (10/80)

MATLAB contains a built-in function called `conv` which performs the convolution of two vectors:

```
>> help conv
 conv Convolution and polynomial multiplication.
    C = conv(A, B) convolves vectors A and B.  The resulting vec-
tor is
   length MAX([LENGTH(A)+LENGTH(B)-1,LENGTH(A),LENGTH(B)]). If A
and B are
vectors of polynomial coefficients, convolving them is
equivalent to multiplying the two polynomials.
```

Compute the system response (using the input signal $x[n]$ given in Question (1) and H[n] given in the introduction of part II) by convolving $H[n]$ and $x[n]$ using the MATLAB `conv` function. Plot the response using the `stem`. Compare this response with that obtained in Question 1. Explain any differences between the two outputs.

Question 3: (20/80)

(**Note**: *Student must answer all Question 3 in their lab report.*)

Sometimes a system that we are interested in is a black box, i.e., we can see what goes into the system and what comes out of the system but we cannot see what is inside the system.

In such a case we can try to find out some things about the system by choosing particular inputs to put in and then observing what comes out.

We could also use this same process of putting inputs in and observing the output to determine the system entirely. When we do this, it is called system identification.

Here you are given a system called Sys1. Note, that Sys1.p is a Matlab P-code file, available at this link. To use this P-code file, right-click on Sys1.p available at the link above then click on "save link as" to save the file. Make sure to save this file in the same directory that your .m files are saved. Now you can call this file and use it in your script in a similar way that a MATLAB built-in function is called and used.

y = Sys1(x)

Inputs to the function is the input vector `x`. The vector `y` is a vector which is the corresponding output signal. Note that `y` may not be the same length as `x`.

(a) Is the system Linear?

i. To show that a system is linear we must show it in general. To show it is not linear we must provide a counter-example. Here we will give inputs to the system and get the outputs. We can do this as many times as we like. Write a hypothesis that you will design an experiment to test. Comment on the possible outcomes of your experiment.
ii. Design an experiment to test linearity. Discuss why you chose particular inputs to put into the system and what you are expecting to learn by observing the outputs. This answer should be 0.5 to 1 page long.
iii. Describe what you observed when you put your inputs into the system.
iv. What did you conclude from your experiments? Was the system linear or not? Can you state this conclusion with certainty?

(b) Is the system Time Invariant?

i. To show that a system is time invariant we must show it in general. To show it is not time-invariant or time-varying we must provide a counter example. Here we will give inputs to the system and get the outputs. We can do this as many times as we like. Write a hypothesis that you will design an experiment to test. Comment on the possible outcomes of your experiment.
ii. Design an experiment to test time invariance. Discuss why you chose particular inputs to put into the system and what you are expecting to learn by observing the outputs. This answer should be 0.5 to 1 page long.
iii. Describe what you observed when you put your inputs into the system.
iv. What did you conclude from your experiments? Was the system time-invariant or not? Can you state this conclusion with certainty

References

1. *Continuous and Discrete Time Signals and Systems*, M. Mandal and A. Asif, Cambridge University Press, ISBN 0-521-85455-5, 2007, p.73.

2. *Continuous and Discrete Time Signals and Systems*, M. Mandal and A. Asif, Cambridge University Press, ISBN 0-521-85455-5, 2007, p.21.

3. *Signals and Systems, 2nd ed.*, A.Oppenheim, A.S. Willsky, S. Nawad, Prentice-Hall, ISBN 0- 13-814757-4, 1997, p.784.

4. *Signals and Systems, 2nd ed.*, A.Oppenheim, A.S. Willsky, S. Nawad, Prentice-Hall, ISBN 0- 13-814757-4, 1997, p. 785.