DIGITAL COMMUNICATION LAB MANUAL

Prepared by:	Ms. Khuong T. T. Pham
Date:	January, 2018

EXPERIMENT # 1

MATLAB Basics for Communication System Design

Objective

- To understand the use of MATLAB for solving communication engineering problems.
- Learn the basics of MATLAB as used in Analogue Communication.
- To develop understanding of MATLAB environment, commands and syntax.

MATLAB

MATLAB is a powerful tool that is utilized by the engineers and others professionals in development and testing of various projects. It is versatile software, with the help of which you can solve and develop any sort of engineering problem. The name MATLAB stands for MATRIX LABORAORY. All the work done in MATLAB is basically in the form of matrices. Scalars are referred as 1-to-1 matrix and vectors are matrices having more than 1 row and column. MATLAB is programmable and have the same logical, relational, conditional and loop structures as in other programming languages, such as C, Java etc. It's very easy to use MATLAB, all we need is to practice it and become a friend of it.

Summary:

- **Scalars**
- **Vectors**
- **Matrices**
- **Plotting**
- **m-files**
- **functions**

Getting Started:

a) Go to the start button, then programs, MATLAB and then start MATLAB. It is preferred that you have MATLAB2015a. You can then start MATLAB by double clicking on its icon on Desktop, if there is any.

b) The Prompt:

   >>

   The operator shows above is the prompt in MATLAB. MATLAB is interactive language like C, Java etc. We can write the commands over here.

c) In MATLAB we can see our previous commands and instructions by pressing the up key. Press the key once to see the previous entry, twice to see the entry before that and so on. We can also edit the text by using forward and back-word keys.

Help in MATLAB

In order to use the built-in help of the MATLAB we use the **help** keyword. Write it on the prompt and see the output.

```
>> help sin
```

Also try

```
>> lookfor sin
```

Scalars

A scalar is a single number. A scalar is stored in the MATLAB as a 1 x 1 matrix. Try these on the prompt.

```
>> A = 2;

>> B = 3;

>> C = A^B

>> C = A*B
```

Try these instructions as well

```
>> C = A+B

>> C = A-B

>> C = A/B

>> C = A\B
```

Note the difference between last two instructions.

Try to implement these two relations and show the result in the provided space

a) $25 (3^{1/3}) + 2 (2+9^2) =$ ______________________

b) $5x^3 + 3x^2 + 5x + 14$ for $x = 3$ is ______________________

Vectors

Vectors are also called arrays in MATLAB. Vectors are declared in the following format.

```
>> X = [1 2 3 4]

>> Y = [2 5 8 9]
```

Try these two instructions in MATLAB and see the result

>> length (X) = __________

>> size (X)   = ___________

What is the difference between these two?

____

Try these instructions and see the results.

>> X.*Y = __________________

>> X.^Y = __________________

>> X+Y = __________________

>> X-Y = __________________

>> X./Y = __________________

>> X'   = __________________

Also try some new instructions for this like and notice the outputs in each case.

>> ones (1,4)

>> ones (2,4)

>> ones (4,1)

>> zeros (1,4)

>> zeros (2,4)

There is an important operator, the colon operator (**:**), it is very important operator and frequently used during these labs. Try this one.

>> X = [0:0.1:1]

Notice the result. And now type this

>> length (X)

>> size (X)

What did the first and second number represent in the output of last instruction?

____
____

Now try this one.

>>A= [ones(1,3), [2:2:10], zeros(1,3)] What is the length and size of this?

>> Length = ____________________

Size = ____________________

Try '*help ones*' and '*help zeros*' as well, and note down its important features.

MATRICES

Try this and see the output.

>> A = [1 2 3;4 5 6;7 8 9]

>> B = [1,2,3;4,5,6;7,8,9]

Is there any difference between the two? Try to implement 2-to-3 matrix and 3-to-2 matrix.

Also take help on **mod, rem, det, inv** and **eye** and try to implement them. Try to use **length** and **size** commands with these matrices as well and see the results.

Try to solve these.

1. $6x + 12y + 4z = 70$

   $7x - 2y + 3z = 5$

   $2x + 8y - 9z = 64$

2. A = [2 3 4 5; 1 8 9 0; 2 3 1 3; 5 8 9 3]

   Solve $6A - 2I + A^2 =$

PLOTTING

Plotting is very important as we have to deal with various type of waves and we have to view them as well.
Try these and have a look on the results.

```
>> x = [0:0.1:10];
>> y = sin (x);
>> z = cos (x);
>> subplot (3,1,1);
>> plot (x,y);
>> grid on;
>> subplot (3,1,2);
>> plot (x,z);
>> grid on; hold on;
>> subplot (3,1,3);
```

```
>> stem (x,z);
>> grid on;
>> hold on;
>> subplot (3,1,3);
>> stem (x,y, ,'r');
```

Take **help** on the functions and commands that you don't know. See the difference between the **stem** and **plot**.

See help on plot, figure, grid, hold, subplot, stem and other features of it.

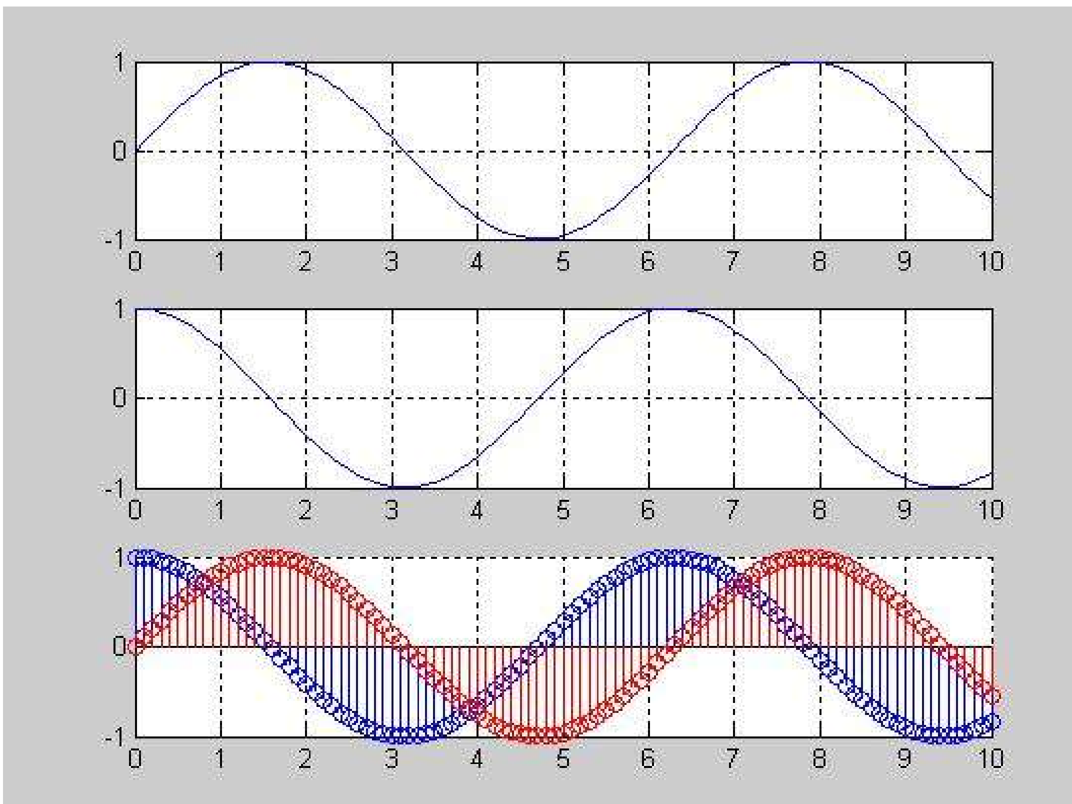


Figure 1.1

M-FILES

MATLAB can execute a sequence of statements stored in disk files. Such files are called M-files because they must have the file type ‘**.m**’. Lot of our work will be done with creation of m-files.

There are two types of m-files: Script and function files.

Script Files

We can use script files in order to write long programs such as one on the previous page. A script file may contain any command that can be entered on the prompt. Script files can have any name but they should be saved with “.m” extension. In order to excurse an m-file from the prompt, just type its name on the prompt. You can make an m-file by typing **edit** on the prompt or by clicking on the file then new and m-file. See an example of m-file. Write it and see the results.

```
% This is comment
% A comment begins with a percent symbol
% The text written in the comments is ignored by the MATLAB
% comments in your m-files.

clear;
clc;
x = [0:0.1:10];
y = sin (x);
subplot (2,2,1);
plot (x,y, ,'r');
grid on;
z = cos (x);
subplot (2,2,2);
plot (x,z);
grid on;
w = 90;
yy = 2*pi*sin
(x+w)
subplot (2,2,3);
plot (x,yy);
grid on;
zz = sin (x+2*w);
subplot (2,2,4);
stem (x,zz, ,'g');
hold on;
stem (x,y, ,'r');
grid on;
```

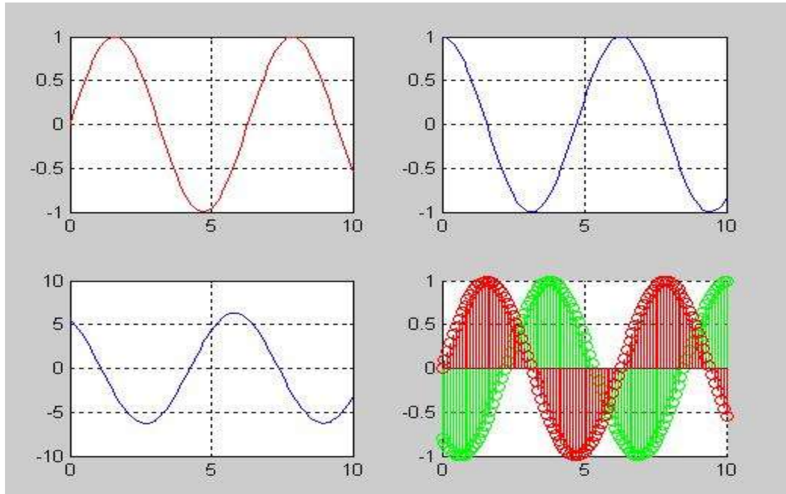


Figure 1.2

Function Files

MATLAB have many built-in functions including trigonometry, logarithm, calculus and hyperbolic functions etc. In addition we can define our own functions and we can use built-in functions in our functions files as well. The function files should be started with the function definition and should be saved with the name of function. The general format of the function file is

Function [output_variables] = function name (input_variables)

See the following example and implement it.

% this is a function file

% this function computes the factorial of a number function [y] = my_func (x)

y = factorial (x);

POST LAB

Try to develop a function that will compute the maximum and minimum of two numbers.

Experiment # 2

Communication Signals: Generation and Interpretation

Objective

- To the use of MATLAB for generation of different signals important in communication theory.
- Learn the basics of signals and its operations as used in Analogue Communication.
- To develop understanding of communication signals and their properties.

Generation of Signals

Signals are represented mathematically as a function of one or more independent variables. We will generally refer to the independent variable as time. Therefore we can say a signal is a function of time. Write these instructions in m-file as execute to see the result.

Sinusoidal Sequence:

```
% Example 2.1
% Generation of sinusoidal signals
% 2sin( 2πτ-π/2)
t=[-5:0.01:5];
x=2*sin((2*pi*t)-(pi/2));
plot(t,x)
grid on;
axis([-6 6 -3 3])
ylabel ('x(t)')
xlabel ('Time(sec)')
title ('Figure 2.1')
```

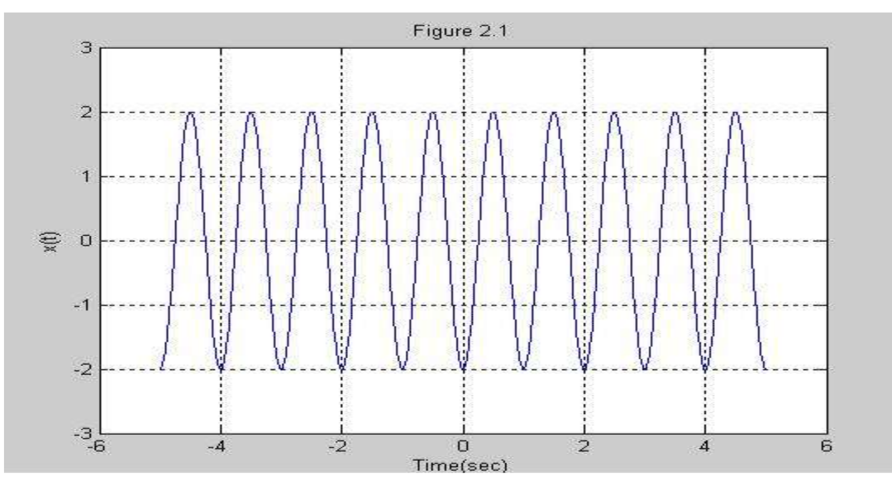


Figure 2.1

See the output, change the phase shift value and observe the differences.

Discrete Time Sequences:

See the example below:

```
 % Example 2.2
% Generation of discrete time signals
n = [-5:5];
x = [0 0 1 1 -1 0 2 -2 3 0 -1];
stem (n,x);
axis ([-6 6 -3 3]);
xlabel ('n'); ylabel
('x[n]'); title
('Figure 2.2');
```

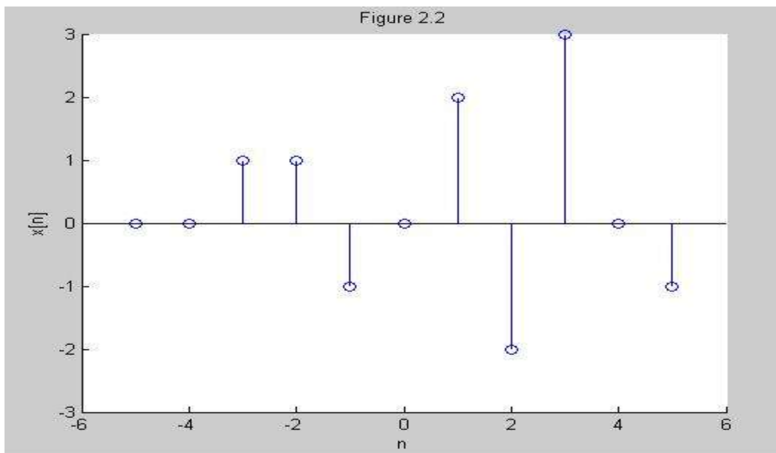


Figure 2.2

Unit Impulse Sequence:

A unit impulse sequence is defined as

Delta (n) = 1 n = 0

= 0 n ≠ 0

We are making a function named imseq and we further use this function in next experiments of this lab. The MATLAB code is given below:

```
function [x,n] = impseq(n0,n1,n2)

% Generates x(n) = delta (n-n0); n1<=n,n0 <= n2
% x[x,n] = imseq(n0,n1,n2)
% n0 = impulse position, n1 = starting index, n2 = ending index
If ((n0 < n1) | (n0 > n2) | (n1 > n2))
    Error('arguments must satisfy n1 <= n0 <=
n2') end
n = [n1:n2];
% x = [zeros(1,(n0-n1)),1,zeros(1,(n2-n0))];

x = [(n-n0) == 0];
```

Unit Step Sequence:

It is defined as

u(n) = 1 n ≥ 0

0 n ≤ 0

The MATLAB code for stem sequence function is given below:

```
function [x,n] = stepseq(n0,n1,n2)
% Generates x(n) = u(n-n0); n1 <= n, n0<=n2
% [x,n] = stepseq(n0,n1,n2)
if ((n0 < n1) | (n0 > n2) | (n1 > n2))
    error('arguments must satisfy n1 <= n0 <= n2')
end
n = [n1:n2];
% x = [zeros(1,(n0-n1)),ones(1,(n2-n0+1))];
x = [(n-n0) >= 0];
```

Real Valued Exponential Sequence:

It is define as:

x (n) = an, for all n; a € Real numbers

We require an array operator " **.^** " to implement a real exponential sequence. See the MATLAB code below

```
>> n = [0:10];
>>x = (0.9).^n;
```

Observe the result

Complex Valued Exponential Sequence:

It is define as:

x(n) = e (a + jb) n , for all n

Where **a** is called the attenuation and **b** is the frequency in radians. It can be implemented by following MATLAB script.

```
>> n = [0:10];
>> x = exp ((2+3j)*n);
```

Random Sequence:

Many practical sequences cannot be described by the mathematical expressions like above, these are called random sequences. In MATLAB two types of random sequences are available. See the code below:

```
>>rand (1,N)
```

```
>> randn (1,N)
```

The above instruction generates a length **N** random sequence whose elements are uniformly distributed between [0,1]. And the last instruction, **randn** generates a length **N** Gaussian random sequence with mean 0 and variance 1. Plot these sequences.

% example 2.3

```
%Generation of random sequence
n = [0:10];
x = rand (1, length (n));
y = randn (1, length (n));
plot (n,x) ;
grid on;
hold on;
plot(n,y,'r');
ylabel ('x & y')
xlabel ('n')
title ('Figure 2.3')
```

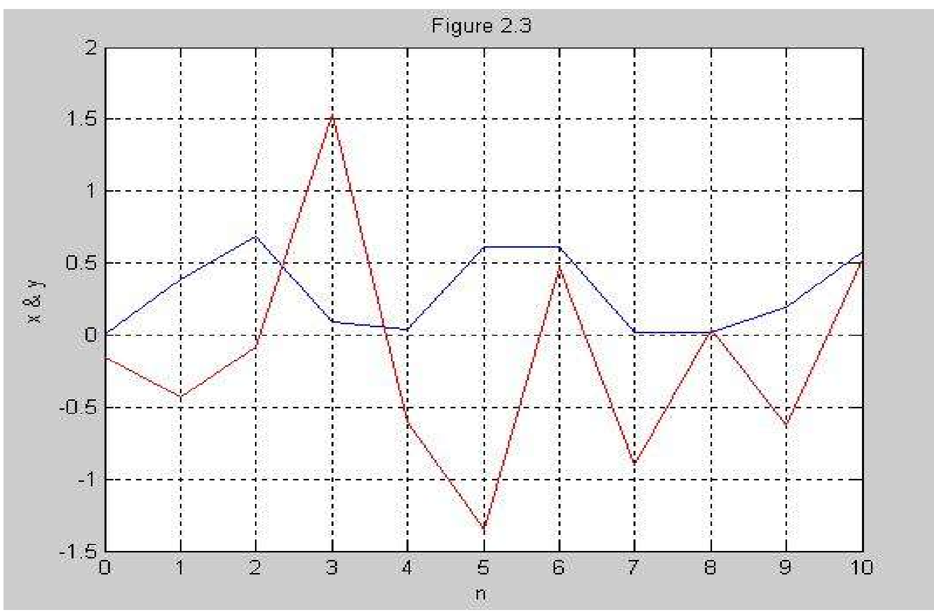


Figure 2.3

Periodic Sequences:

A sequence is periodic if it repeats itself after equal interval of time. The smallest interval is called the fundamental period. Implement code given below and see the periodicity.

% **Example 2.4**

```
% Generation of periodic sequences
```

```
n = [0:4];
x = [1 1 2 -1 0];
subplot (2,1,1);
stem (n,x);
grid on;
axis ([0 14 -1 2]);
xlabel ('n');
ylabel ('x(n)');
title ('Figure 2.4(a)');
xtilde = [x,x,x];
length_xtilde = length (xtilde);
n_new = [0:length_xtilde-1];
subplot (2,1,2);
stem (n_new,xtilde,'r');
grid on;
xlabel ('n');
ylabel ('perodic x(n)');
title ('Figure 2.4(b)');
```

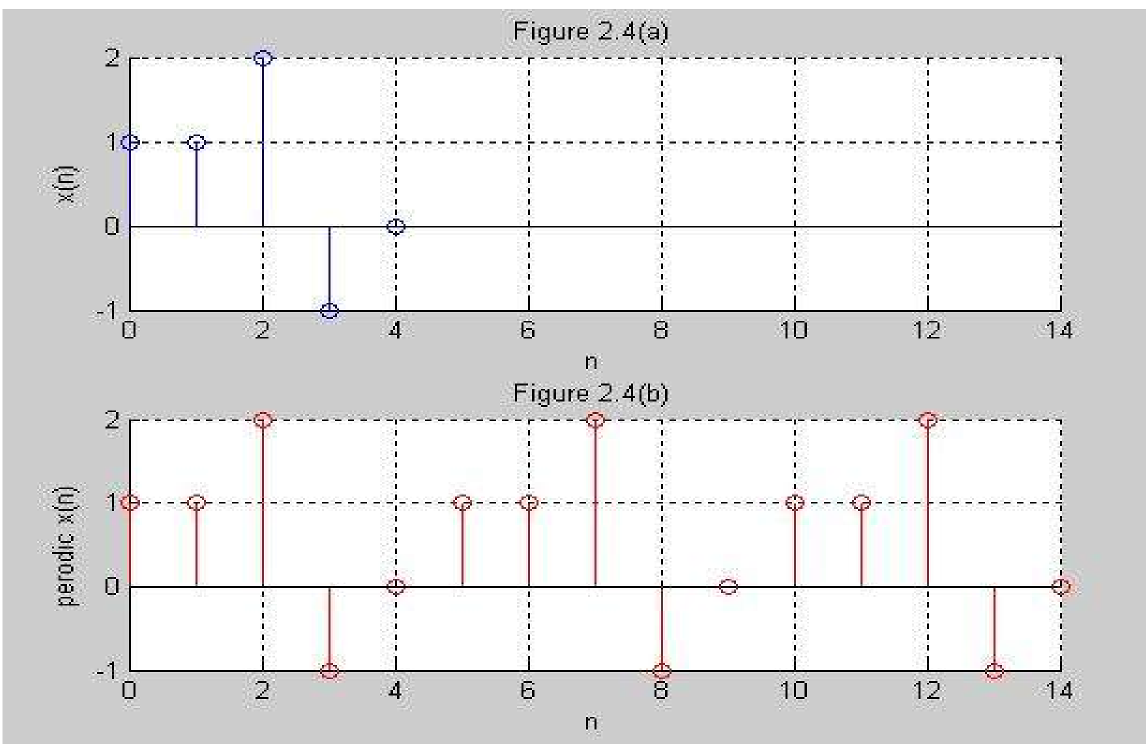


Figure 2.4

SIGNALS OPERATIONS:

Signal Addition

This is basically sample by sample addition. The definition is given below:

$$\{x1(n)\} + \{x2(n)\} = \{x1(n) + x2(n)\}$$

The length of x1 and x2 should be equal. See the MATLAB code below:

```
function [y,n] = sigadd(x1,n1,x2,n2)
% implement y(n) = x1(n) + x2 (n)
% [y,n] = sigadd (x1,n1,x2,n2)
% y = sum sequence over n, which include n1 and n2
% x1 = first sequence over n1
% x2 = second sequence over n2 (n2 can be different from n1)
n = min(min(n1),min(n2)): max(max(n1),max(n2));          %duration of y(n)
y1    = zeros(1,length(n));                               % initialization
y2    = y1;
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;               % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;               % x2 with duration of y
y = y1 + y2;
```

See example of signal addition below

% **Example 2.5**

```
% signal addition using sigadd function
clear;
clc;
n1 = [0:10];
x1 = sin (n1);
n2 = [-5:7];
x2 = 4*sin(n2);
[y,n] = sigadd(x1,n1,x2,n2);
subplot (3,1,1);
stem (n1,x1);
grid on;
axis ([-5 10 -5 5]);
xlabel ('n1');    ylabel ('x1(n)');
title ('1st signal');
subplot (3,1,2);
stem (n2,x2);
grid on; hold on;
axis ([-5 10 -5 5]);
xlabel ('n2');    ylabel ('x2(n)');
title ('2nd signal');
subplot (3,1,3);   stem (n,y,'r');
grid on;
```

```
axis ([-5 10 -5 5]);
xlabel ('n');        ylabel ('y(n)');
title ('Added Signals');
```

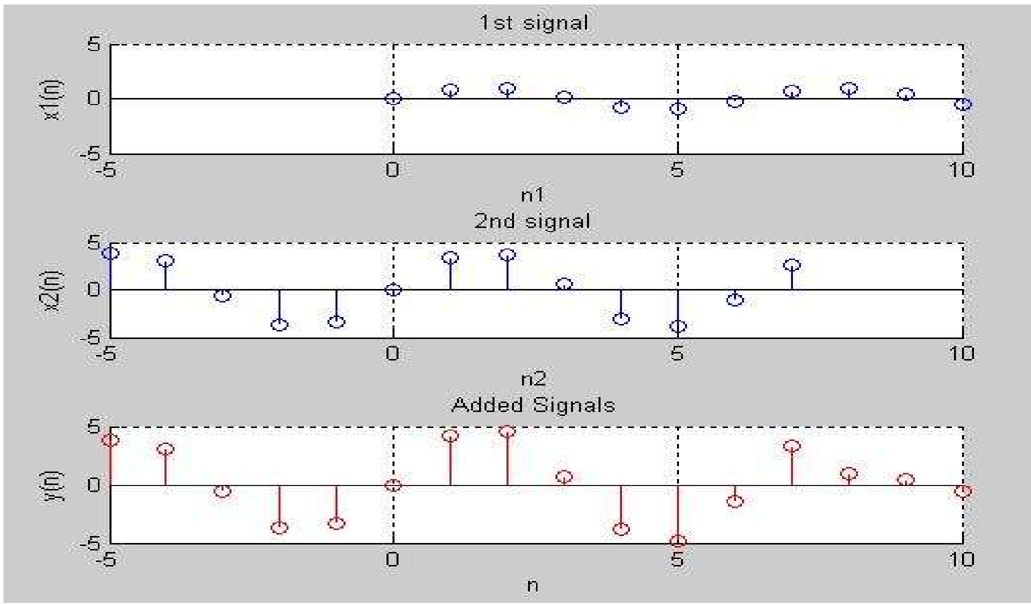


Figure 2.5

Signal Multiplication:

The multiplication of two signals is basically sample by sample multiplication or you can say dot multiplication. By definition it is

$$\{x1(n)\} . \{x2(n)\} = \{x1(n)x2(n)\}$$

It is implemented by the array operator ‘ **.*** ‘ that we studied in last lab. A signal multiplication function is developed that is similar to the sigadd function. See the code below:

```
function [y,n] = sigmult (x1,n1,x2,n2)

% implement y(n) = x1(n) * x2 (n)

% [y,n] = sigmult (x1,n1,x2,n2)

% y = product sequence over n, which include n1 and n2

% x1 = first sequence over n1

% x2 = second sequence over n2 (n2 can be different from n1)

n = min(min(n1),min(n2)): 0.1 : max(max(n1),max(n2));   %duration of y(n)
```

```
y1 = zeros(1,length(n));                              % initialization
y2 = y1;
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;          % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;          % x2 with duration of y
y = y1 .* y2;
```

See the example below:

% **Example 2.6**

```
% signal multiplication using sigmult function
clear;
clc;
n1 = [0:0.1:10];
x1 = sin (n1);
n2 = [-5:0.1:7];
x2 = 4*sin (n2);
[y,n] = sigmult(x1,n1,x2,n2);
subplot (3,1,1);
stem (n1,x1);
grid on;
axis ([-5 10 -5 5]);
xlabel ('n1');
ylabel ('x1(n)');
title ('1st signal');
subplot (3,1,2);
stem (n2,x2);
grid on;
hold on;
axis ([-5 10 -5 5]);
xlabel ('n2');
ylabel ('x2(n)');
title ('2nd signal');
subplot (3,1,3);
stem (n,y,'r');
grid on;
axis ([-5 10 -5 5]);
xlabel ('n');
ylabel ('y(n)');
title ('Multiplied Signals');
```

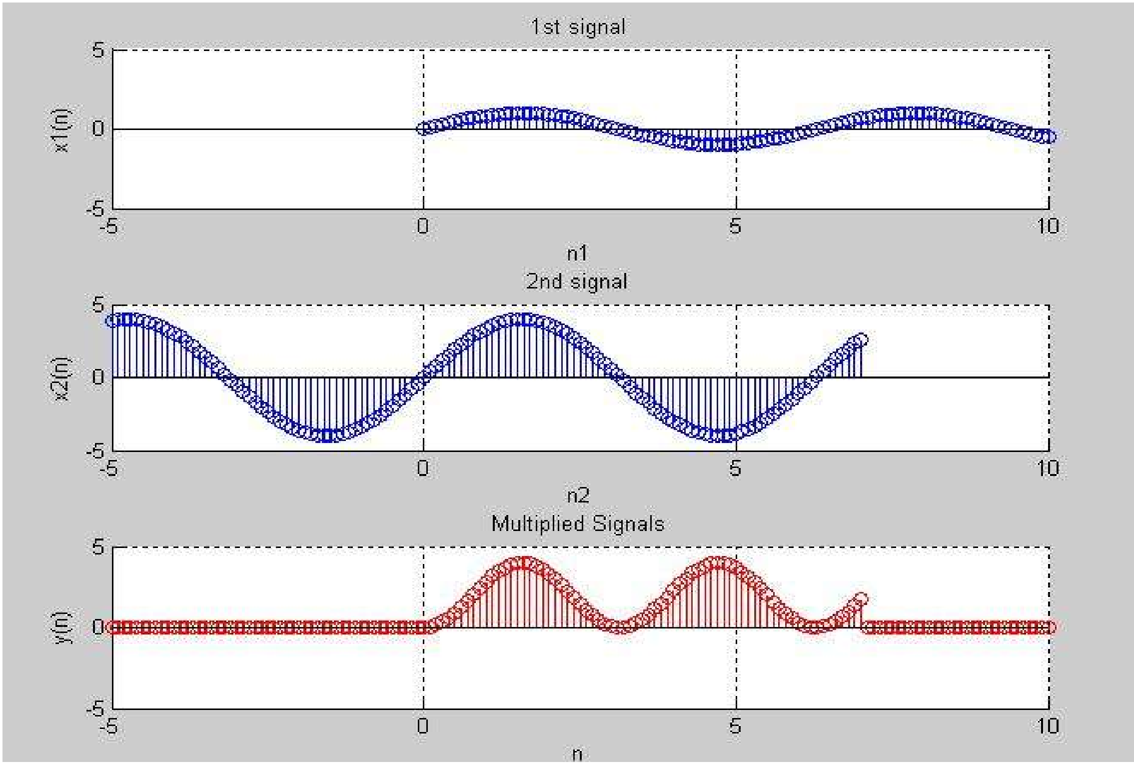


Figure 2.6

POST LAB: (please send to email: phamthaokhuong@gmail.com)

Write MATLAB code to plot these signals:

a. x [n] = 2sin (3n) + 2cos (3n)

b. x [n] = u[n] + 4cos (3n)

c. x [n] = n[u(n) – u(n-10)] + 10e-0.3(n-10)[u(n-10)-u(n-20)]

You are not allowed to multiply impulse sequences with a number. Implement this by using **impseq**, **stepseq** and **sigadd** functions.

Experiment # 3

Communication Signals: Operations

Objective

- To learn the use of MATLAB for different operations on signals.
- To develop a thorough understanding of communication signals and their basic operations as used in Analogue Communication.

SUMMARY

- Signal operations (Scaling, Shifting, Folding, Sample Summation, Sample product, Energy, Even and Odd sequences, Convolution)

SIGNAL OPERATIONS:

1. Scaling:

In this operation the samples of the signal is multiplied by a scalar α. The mathematical operator * is used for the implementation of the scaling property.

α{x(n)} = { α x(n)}

```
>> [x,n] = stepseq (-1,-5,5);
>> a = 2;
>> y = a*x;
>> subplot (2,1,1);
>>stem (n,x);grid on;
>> subplot (2,1,2);
>> stem (n,y, 'r');
>> grid on;
```

2. Shifting

In this operation, each sample of the signal is shifted by **k** to get a shifted signal. By definition: **y(n) = {x (n-k)}**

In this operation there is no change in the array or vector x, that contains the samples of the signal. Only n is changed be adding **k** to each element. The function is given below:

```
function [y,n] = sigshift (x,m,n0)
% implement y(n) = x(n-n0)
```

```
% x = samples of original signal
% m = index values of the signal
% n0 = shift amount , may be positive or negative
% [y,n] = sigshift(x,m,n0)
n = m+n0;
y = x;
```

See the example of above function:

% **Example 3.1**

```
% This example demonstrate the use of stepseq, sigshift, sidadd & sigmult
   function clc; clear;
   %-------------------------------------------
   [x,n] = stepseq (0,-
   10,10); subplot (3,2,1);
   stem (n,x);
   axis ([-12 12 0
   3]); grid on;
   xlabel ('n');
   ylabel
   ('x(n)');
   title ('Original Signals');
   %---------------------------------------------
   [y1,n1] = sigshift(x,n,2.5);
   subplot (3,2,2);
   stem (n1,y1); axis ([-12 12 0 3]); grid on;
   xlabel ('n');
   ylabel
   ('y1(n)');
   title ('Shifted signal,x(n-2.5)');
```

```
%-----------------------------------------------
[y2,n2] = sigshift (x,n,-2.5);
subplot (3,2,4);
stem (n2,y2);
axis ([-12 12 0 3]);
grid on;
xlabel ('n');
ylabel ('y2(n)');
title ('Shifted signal,x(n+2.5)');
%-----------------------------------------------
[y_add,n_add] = sigadd(y1,n1,y2,n2);
subplot (3,2,6);
stem (n_add,y_add,'r');
axis ([-12 12 0 3]);
grid on;
xlabel ('n');
ylabel ('y1(n) + y2(n)');
title ('Added Signal');
%-----------------------------------------------
[y_mul,n_mul] = sigmult(y1,n1,y2,n2);
subplot (3,2,5);
stem (n_mul,y_mul,'k');
axis ([-12 12 0 3]);
grid on;
xlabel ('n');
ylabel ('y1(n) * y2(n)');
title ('Multiplied Signal');
%-----------------------------------------------
```

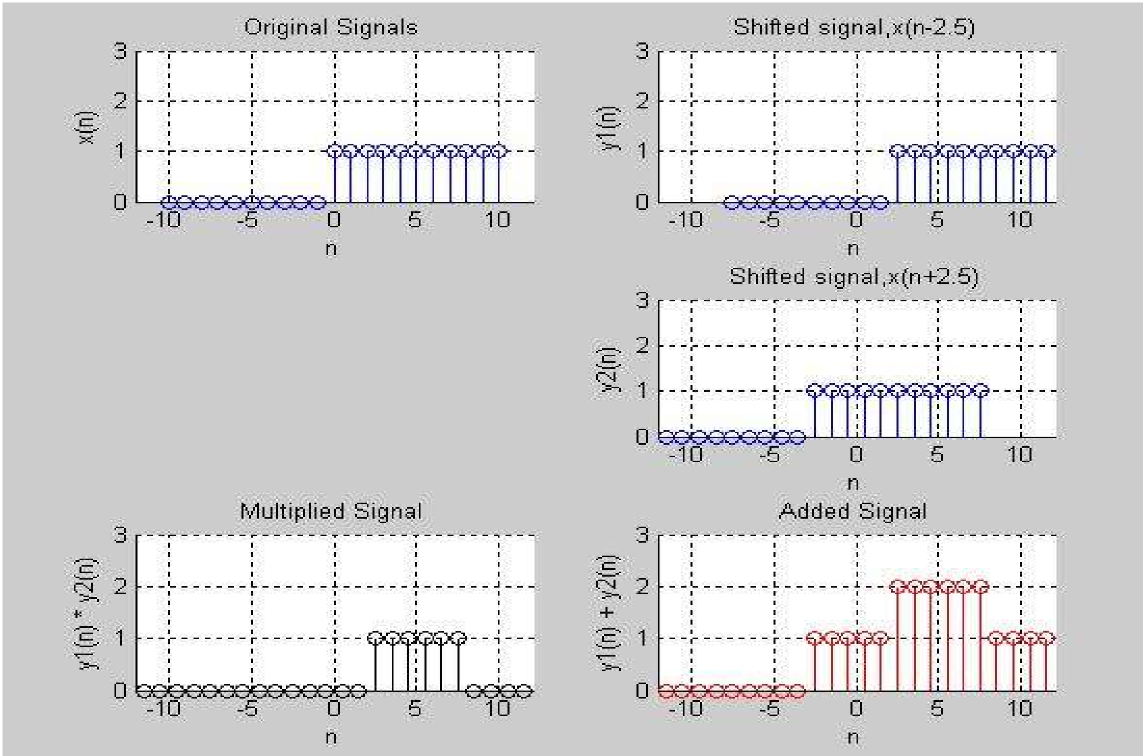


Figure 3.1

3. Folding:

In this operation each sample of **x(n)** is flipped around **n=0** to obtain a folded signal **y(n)**. **y (n) = {x(-n)}**

In MATLAB, this function is implemented by using a built-in function **fliplr(x)** and **–fliplr(x).** Its implementation is given below:

```
function [y,n] = sigfold(x,n)

% implements y(n) = x(-n)

% [y,n] = sigfold(x,n)

% x = samples of the original signal

% n = indexes of the original signal

y = fliplr(x);

n = -fliplr (n);
```

Do its example by yourself from any example signals.

4. Sample Summation:

This operation is different from **sigadd** function. In this operation we add all the sample values of any signal **x(n)** between any two of its index values. By definition

$$\sum x(n) = x(n1) + \ldots\ldots\ldots + x(n2)$$

In MATLAB it is implemented by the **sum(x(n1:n2))** command. See the code below for the demonstration of above function.

```
>> [x,n] = stepseq (5,0,10)

>> sum(x(2:7))
```

5. Sample Product:

This operation also differs from the **sigmult** function. It implies the sample values over the range **n1:n2**. It is implemented by the **prod(x(n1:n2))**. See the code below.

```
>> x = [0 1 2 3 4 5]

>> prod(x(2:5))
```

6. Energy:

The energy of any signal x is computed by the mathematical relation:

$$Ex = \sum x(n)\, x^*(n) = \sum |x(n)|\, 2$$

Where the subscript * is used for complex conjugate of the signal x. The energy of the finite duration signal is computed in MATLAB as.

```
>> Ex = sum (x.*conj(x));
```

Or

```
>> Ex = sum (abs(x).^2);
```

7. Even and Odd Sequence:

A real valued sequence $x_e(n)$ is called even if the following condition satisfies.

$$x_e(-n) = x_e(n)$$

Similarly a signal is said to be an odd signal if

$$xo(-n) = -xo(n)$$

See the example below:

```
% example 3.2

% Generation of even and odd signals

n1 = [0:0.01:1];
```

```
x1 = 2*n1;
n2 = [1:0.01:2];
x2 = -2*n2+4;
n = [n1,n2];
x = [x1,x2];
%Even Signal
[xe,ne] = sigfold(x,n);
%Plotting of original signal
subplot (3,1,1);
plot (n,x);
axis ([-4 4 0 2.5]);
grid on;
%Plotting of original signal + even signal
subplot (3,1,2);
plot (n,x/2,ne,xe/2);
axis ([-4 4 0 2.5]);
grid on;
% Plotting of original signal + odd
signal xo = -xe;
no = ne;
subplot (3,1,3);
plot (n,x/2,no,xo/2);
axis ([-4 4 -2.5 2.5]);
grid on;
```

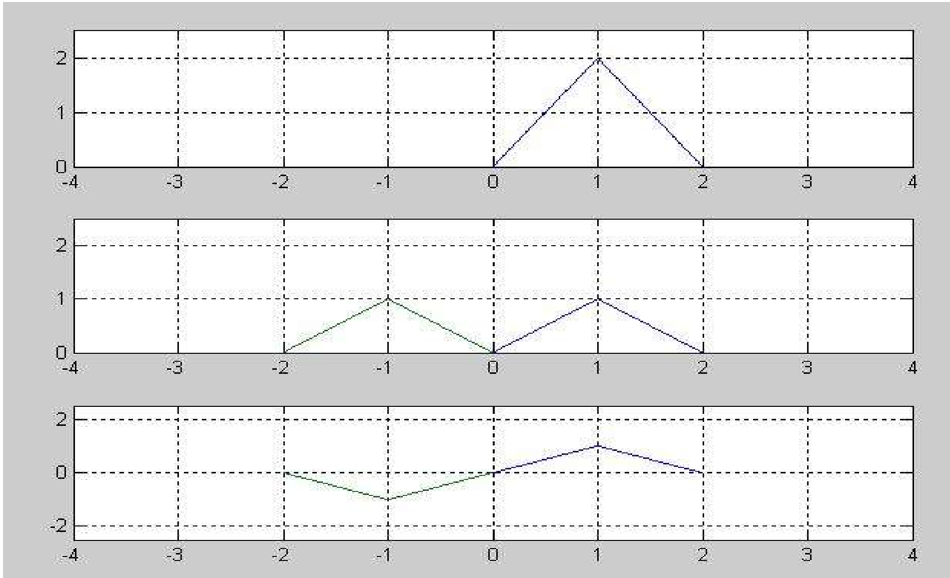


Figure 3.2

The above example shows to develop the even and odd signals from a given signal. Now we are going to develop a function to compute the even and odd signals for ourselves. See the code of function file below:

```
function [xe,xo,m] = evenodd (x,n)

% Decomposes a real function into its even and odd parts

% [xe,xo,m] = evenodd(x,n)

% xe = even signal

% xo = odd signal

% m = indexes

% x = original signal

% n = indexes for original signal

if any(imag(x)~=0)

                    error('x is not a real sequence')

end

m = -fliplr(n);

m1 = min([m,n]);

m2 = max([m,n]);

m = m1:m2;

nm = n(1)-m(1);
```

```
n1 = 1:length(n);

x1 = zeros(1,length(m));

x1(n1+nm) = x;

x = x1;

xe = 0.5*(x+fliplr(x));

xo = 0.5*(x-fliplr(x));
```

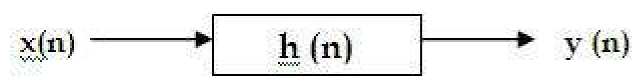
Now change the example 3.2 code to implement the same example with this function.

8. Convolution:

The convolution is very important operation as far the system as their impulse responses are concern. It is mathematically defines as:

$$y(n) = x(n) * h(n)$$

Where **h(n)** is the impulse response of the system. The above definition is best depicted by the following diagram.



In MATLAB convolution is implemented by the following instructions.

```
>> x = [1 5 3 9 1 2 3 8 5 -3 0 4];

>> h = [1 0 2 3];

>> y = conv(x,h);
```

A function is developed which will evaluate convolution in a more precise form and also calculate the indexes to help us plot the sequences.

```
function [y,ny] = conv_m(x,nx,h,nh)

% Modified convolution routine for signal processing

% [y,ny] = conv_m(x,nx,h,nh)

% [y,ny] = convolution result

% x = original signal

% nx = index values

% h = impulse response signal

% nh = index values for impulse response

nyb = nx(1) + nh(1);
```

```
nye = nx(length(x)) + nh(length(h));

ny = [nyb:nye];

y = conv(x,h);
```

POST LAB (please send to my email before next class)

a. x(n) = u(n) – u(n-5). Decompose into even and odd components and plot them.

b. n = [-2:2]

   x1 = [3,2,1,-2,-3];

   x2 = [1,1,1,1,1]

   Implement y = x1*x2

Experiment # 4

Introduction to Amplitude Modulation (Simulink Implementation)

Objective

- To identify the spectrum analyzer as used in frequency domain analysis
- To identify various types of linear modulated waveforms in time and frequency domain representation
- To implement theoretically functional circuits using the Communication Module Design System (CMDS)

Spectrum Analyzer and Function Generator

This section deals with looking at the spectrum of simple waves. We first look at the spectrum of a simple sine wave

.

To start Simulink: Start MATLAB then type simulink on the command line. A Simulink Library Window opens up as shown in figure 13.1.

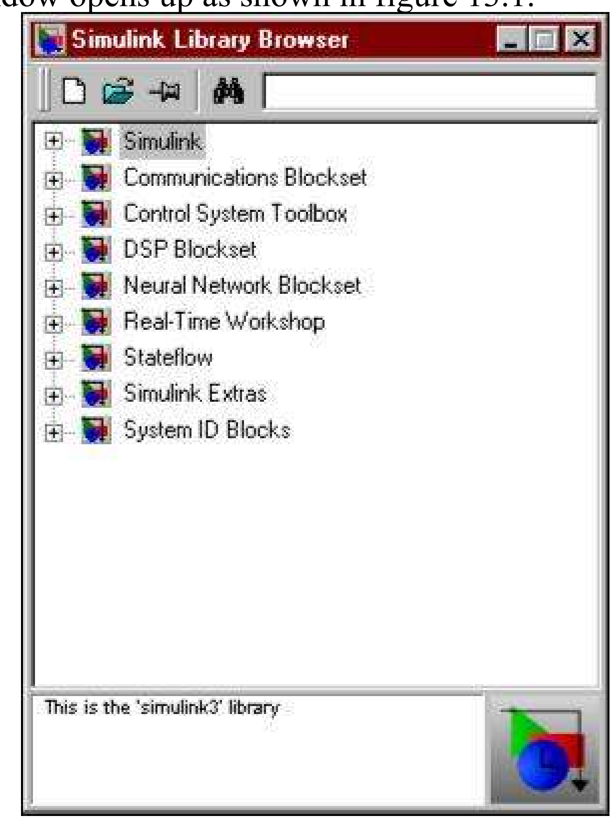


Figure 4.1

Spectrum of a simple sine wave: - Figure 13.2 shows the design for viewing the spectrum of a simple sine wave.

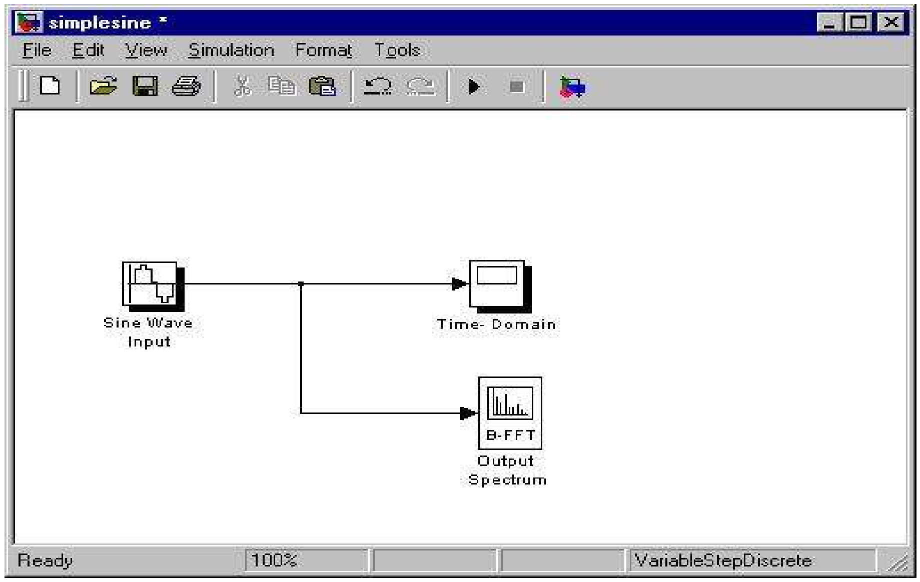


Figure 4 .2

Figure 4.3 shows the time-domain sine wave and the corresponding frequency domain is shown in figure 4.4. The frequency domain spectrum is obtained through a buffered-FFT scope, which comprises of a Fast Fourier Transform of 128 samples which also has a buffering of 64 of them in one frame. The property block of the B-FFT is also displayed in figure 4.5.

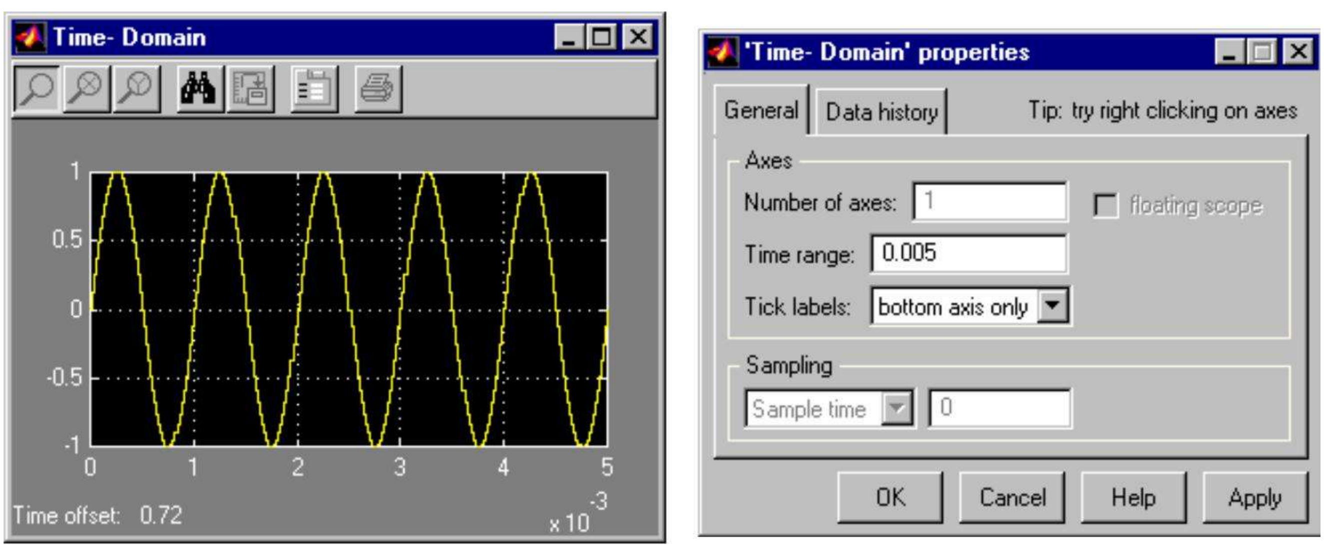


Figure 4.3

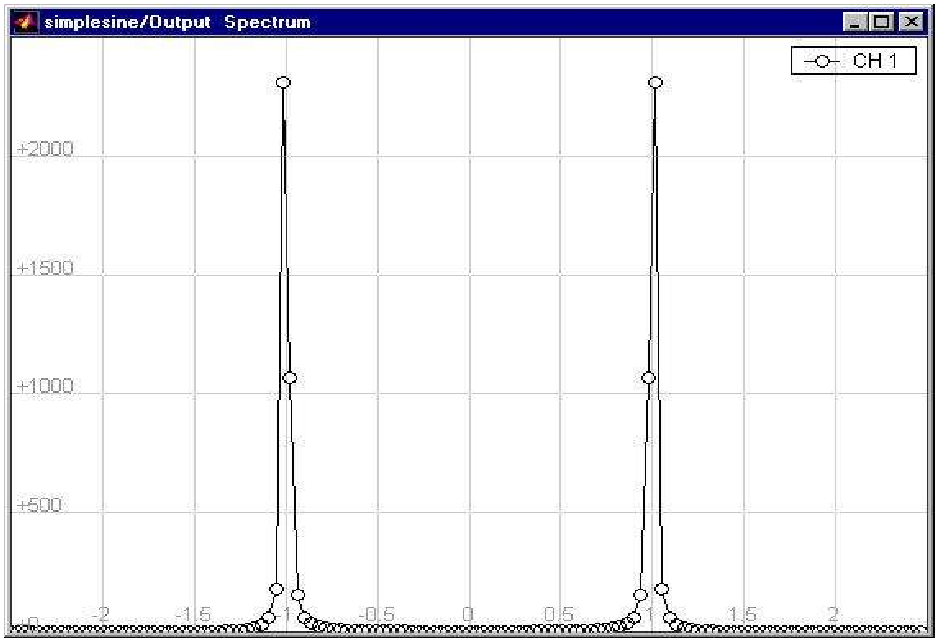


Figure 4.4

This is the property box of the Spectrum Analyzer

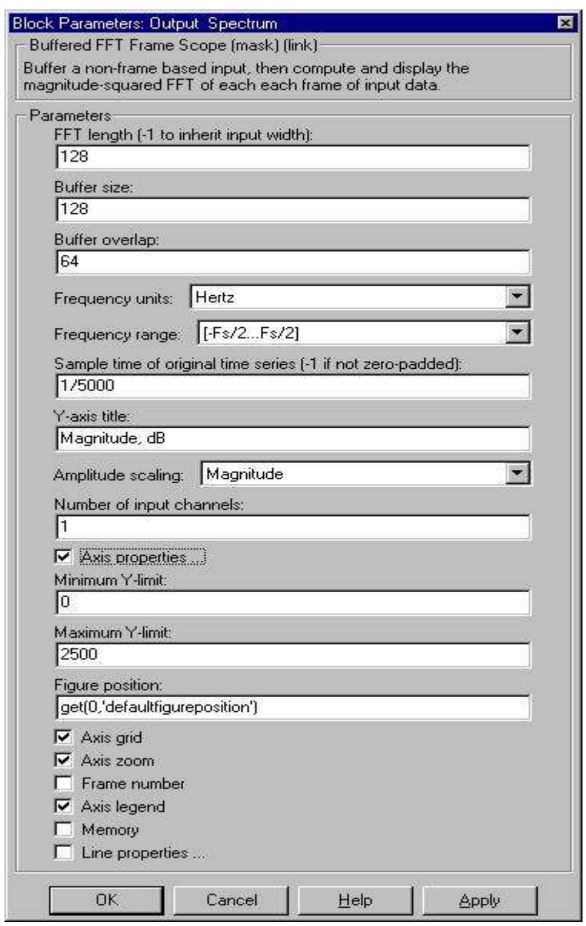


Figure 4.5

From the property box of the B-FFT scope the axis properties can be changed and the Line properties can be changed. The line properties are not shown in the above. The Frequency range can be changed by using the frequency range pop down menu and so can be the y-axis the amplitude scaling be changed to either real magnitude or the dB (log of magnitude) scale. The upper limit can be specified as shown by the Min and Max Y-limits edit box. The sampling time in this case has been set to 1/5000.

Note: The sampling frequency of the B-FFT scope should match with the sampling time of the input time signal.

Also as indicated above the FFT is taken for 128 points and buffered with half of them for an overlap.

Calculating the Power:
The power can be calculated by squaring the value of the voltage of the spectrum analyzer.

Note: The signal analyzer if chosen with half the scale, the spectrum is the single-sided analyzer, so the power in the spectrum is the total power.

Similar operations can be done for other waveforms – like the square wave, triangular. These signals can be generated from the signal generator block.

II. Waveform Multiplication (Modulation)

The equation $y = k_m \cos^2(2\pi\ (1{,}000)t)$ was implemented as in fig. 1B peak to peak voltage of the input and output signal of the multiplier was measured. Then km can be computed as

The spectrum of the output when k_m=1 was shown below:

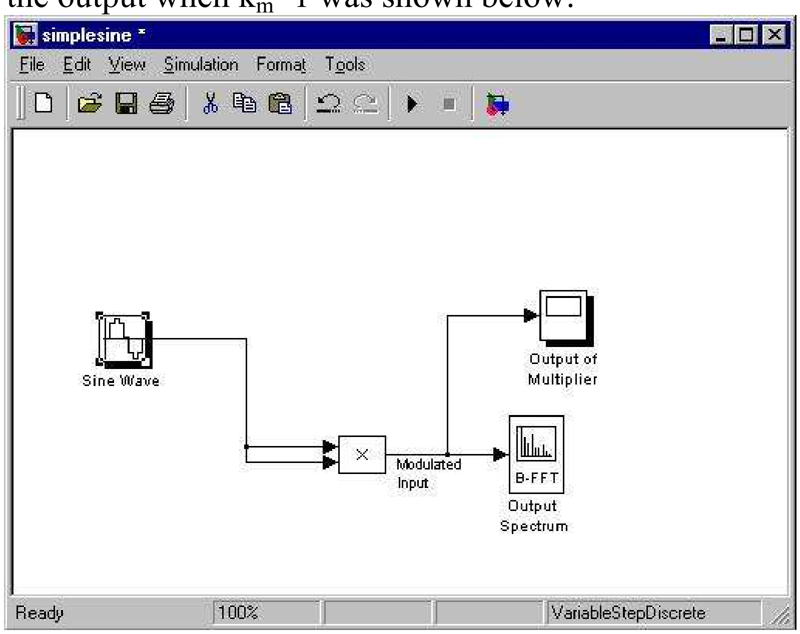


Figure 4.6

The following figure demonstrates the waveform multiplication. A sine wave of 1 kHz is generated using a sine wave generator and multiplied with a replica signal. The input signal and the output are shown in figures.

The input signal as generated by the sine wave is shown in figure.
The output of the multiplier is shown in figure and the spectral output is shown in figure.

It can be seen that the output of the multiplier in time domain is basically a sine wave but doesn't have the negative sides since they get cancelled out in the multiplication.

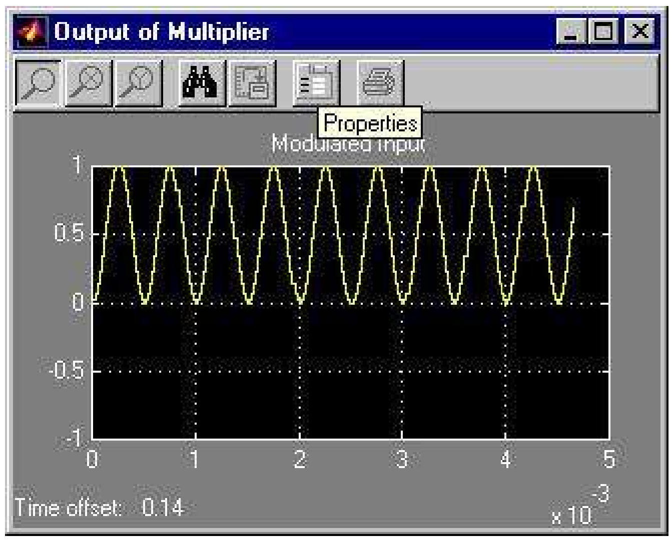


Figure 4.7

The spectral output of the spectrum is shown below. It can be seen that there are two side components in spectrum. The components at fc + fm and –(fc + fm) can be seen along with a central impulse.

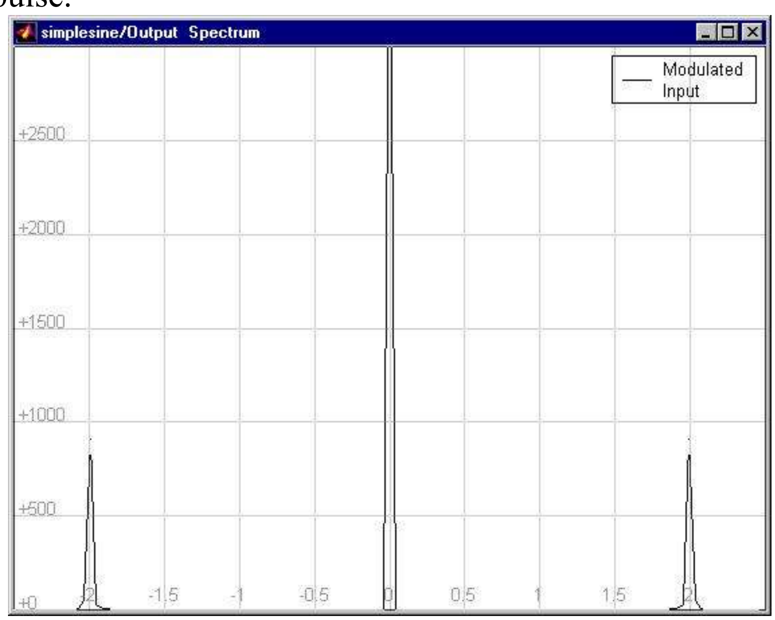


Figure 4.8

If a DC component was present in the input waveform, then

$$y = k_m*(\cos(2\pi(1{,}000)t) + V_{dc})^2$$

The effect of adding a dc component to the input has the overall effect of raising the amplitude of the 2 KHz component and decreases the 2 KHz component. However, for a value of V_{dc} = 0.1V, the 1KHz component reduces and for any other increase in the V_{dc} value, the 1KHz component increases.

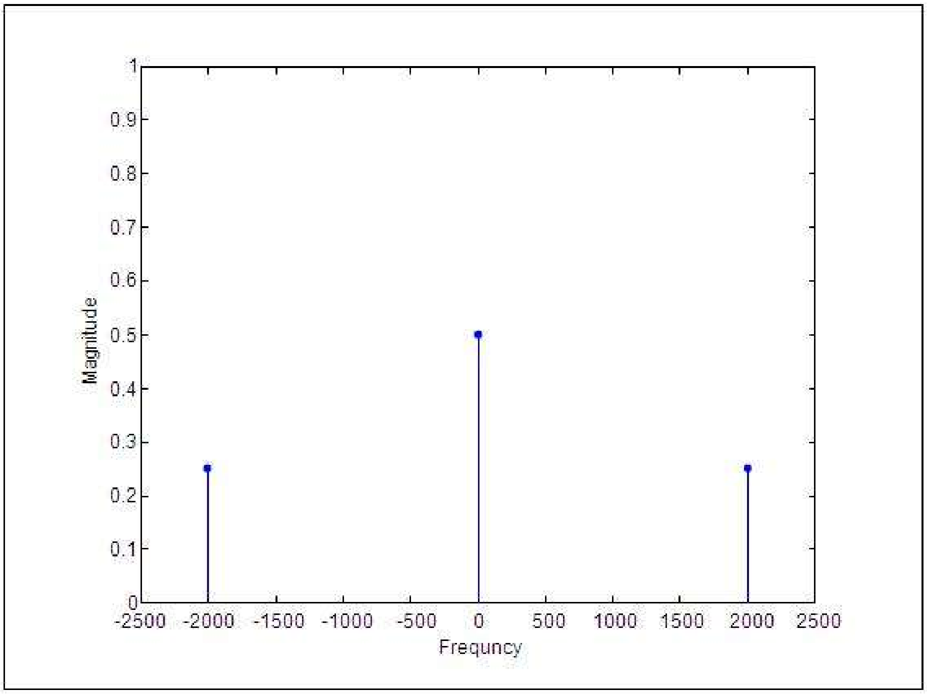


Figure 4.9

I. Double Side-Band Suppressed Carrier Modulation

Figure shows the implementation of a DSB-SC signal. The Signals are at 1 kHz and 10 kHz.

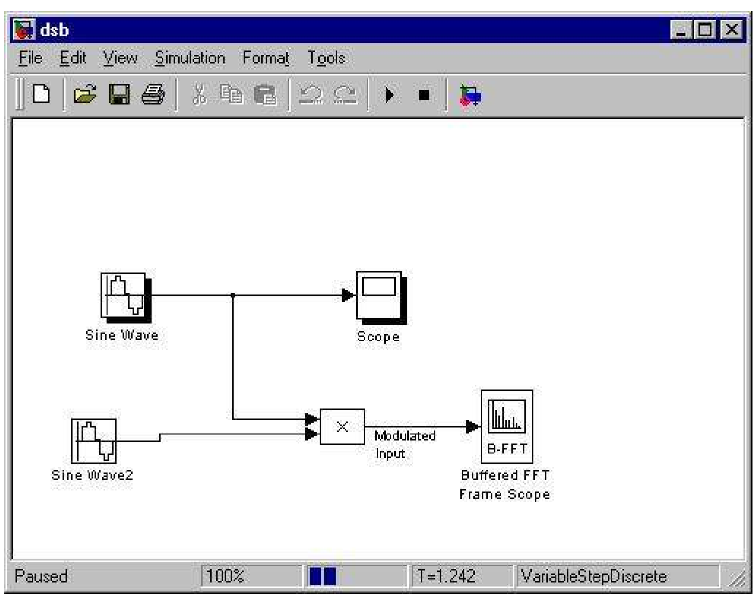


Figure 4.10

The output is shown below. It can be seen that the output consists of just two side bands at +(fc + fm) and the other at –(fc + fm) , i.e. at 9kHz and 11kHz.

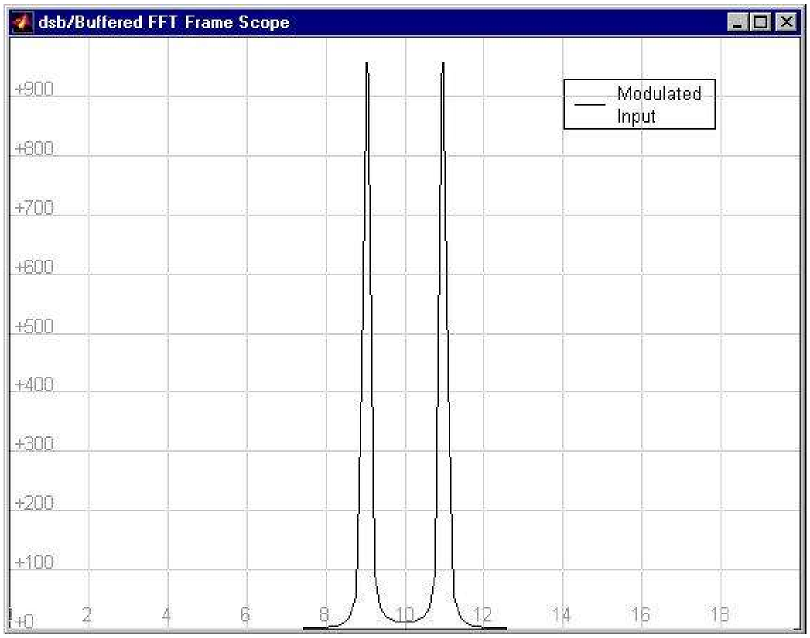


Figure 4.11

By multiplying the carrier signal and the message signal, we achieve modulation.

$Y*m(t) = [km \cos(2\pi 1000t) * \cos(2\pi 10000t)]$

We observe the output to have no 10 KHz component i.e., the carrier is not present. The output contains a band at 9 KHz (fc-fm) and a band at 11 KHz (fc + fm). Thus we observe a double side band suppressed carrier. All the transmitted power is in the 2 sidebands.

Effect of Variations in Modulating and Carrier frequencies on DSB – SC signal.

By varying the carrier and message signal frequencies, we observe that the 2 sidebands move according to equation fc ± fm.

Now, using a square wave as modulating signal, we see that DSBSC is still achieved.

The output from spectrum analyzer was slightly different from the theoretical output. In the result from the spectrum analyzer, there is a small peak at frequency = 10kHz (the carrier frequency) and other 2 peak at 0 and 1000 Hz. This may caused by the incorrectly calibrated multiplier.

Next, the changes to the waveform parameters have been made and then the outputs have been observed. And here are the changes that have been made

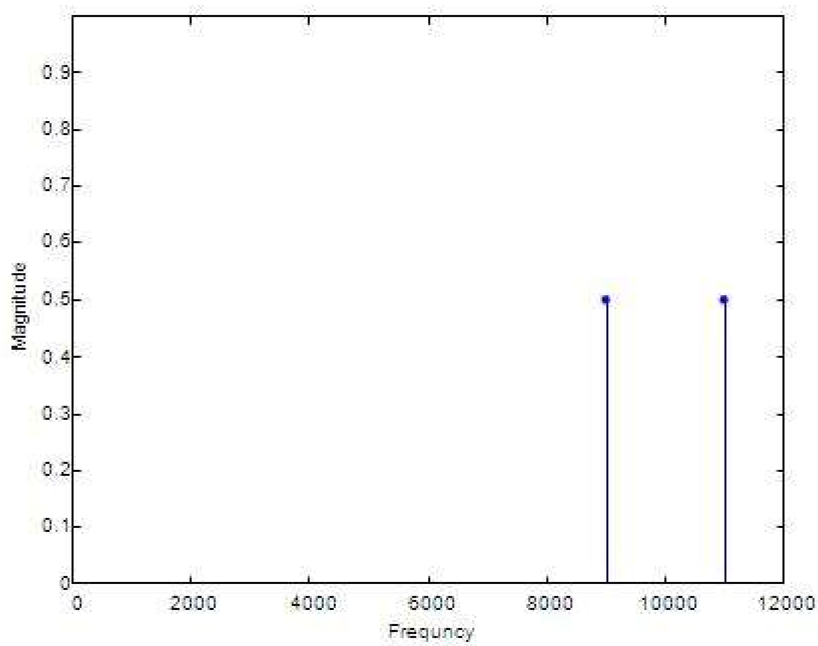


Figure 4.12

Amplitude Modulation

This experiment is the amplitude modulation for modulation index a = 1 and 0.5.

From the equation of the AM

$$y = k_m (1 + a \cdot \cos(2\pi\,(1000)t) \cdot \cos(2\pi\,(10000)t$$

The representation of the signal in both time-domain and frequency domain when km=1 for a=1 and a=0.5 were found to be as shown in figures.

The experimental set up for generating an AM signal looks like this: -

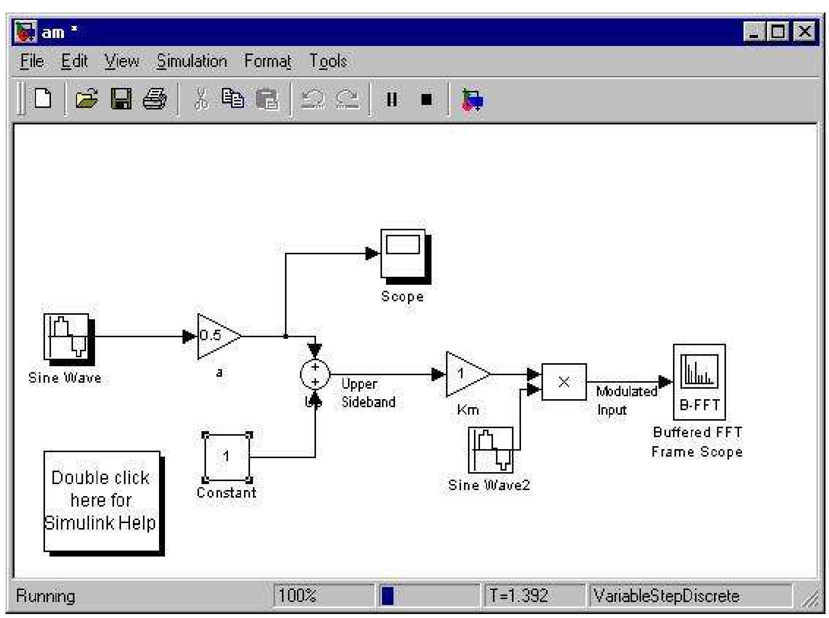


Figure 4.12

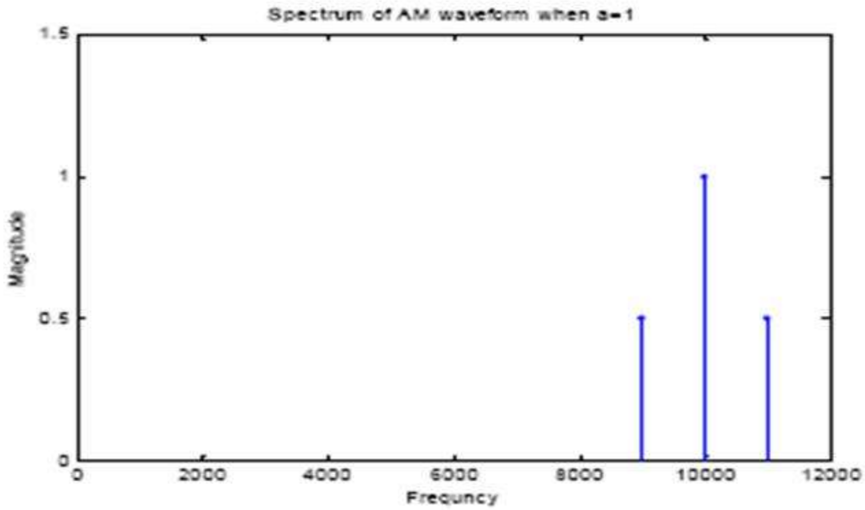


Figure 4.13

The input waveform 50% modulated is shown in figure:

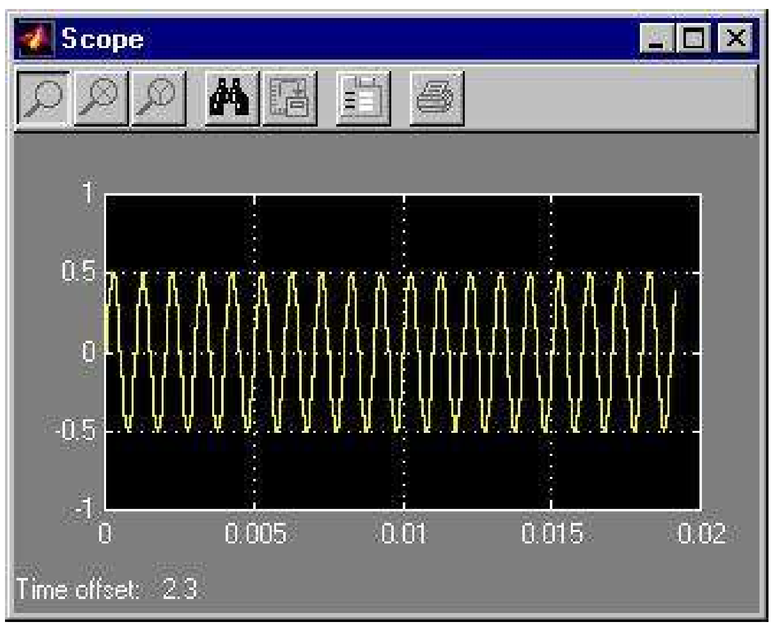


Figure 4.14

The output spectrum is shown below

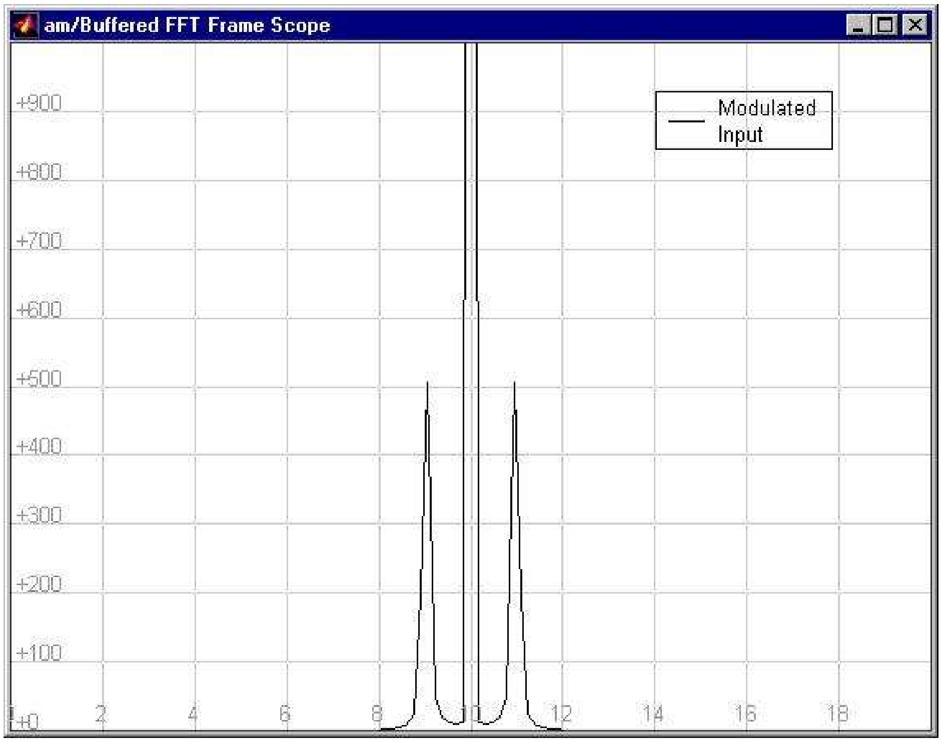


Figure 4.15

It must be noted here that the A.M signal can be converted into a DSB-SC signal by making the constant = 0.

The waveforms at various levels of modulation are shown in the following figures.

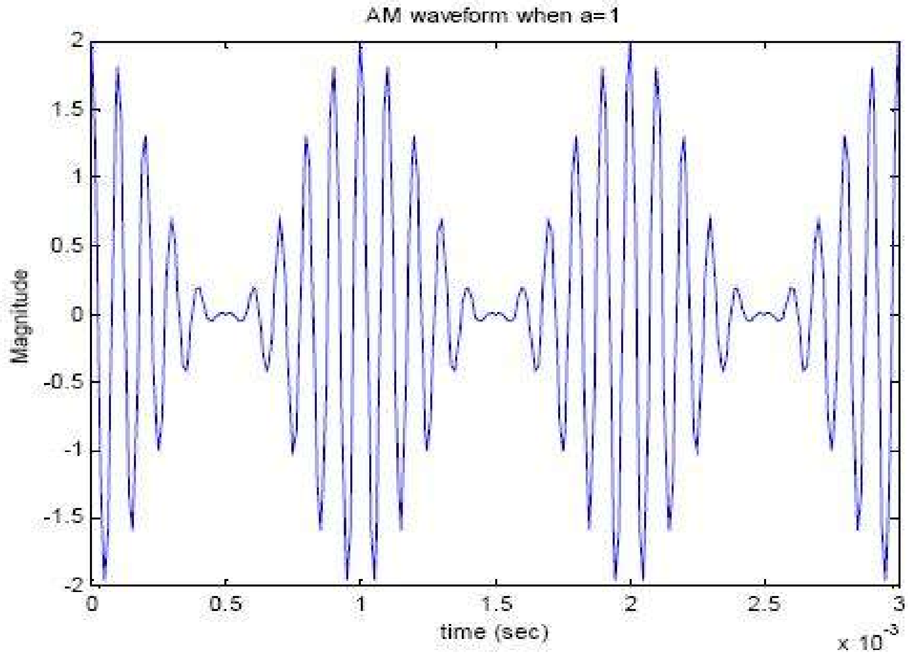


Figure 4.16

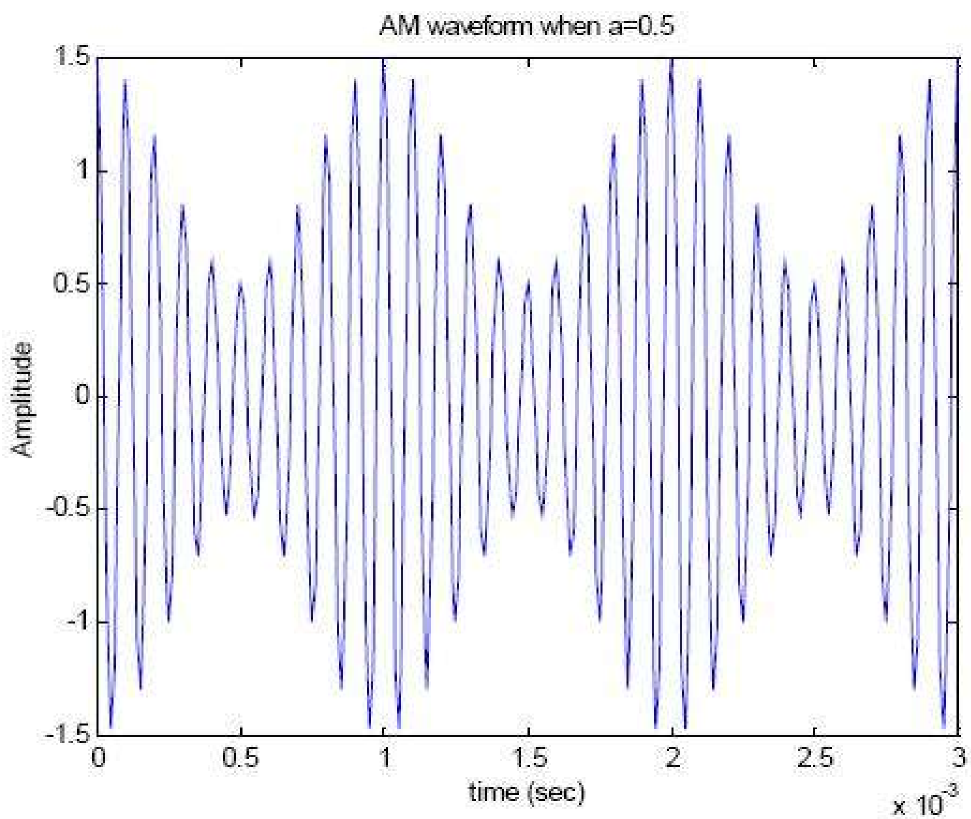


Figure 4.17

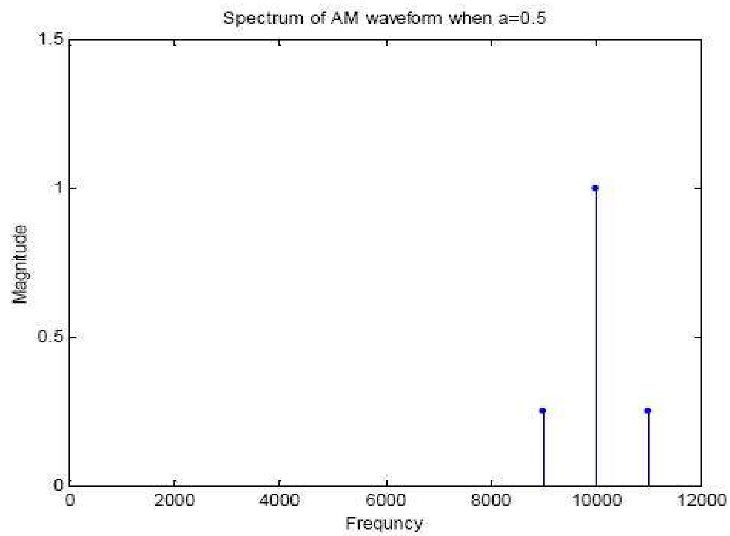


Figure 4.18

The results from the experiment were shown. The results from the experiment are pretty much the same as in the theoretical ones except there are 2 other peaks at 0 and 1000 kHz. This is the same as earlier experiment. The cause of this problem is probably the multiplier.

II. Two Tone Modulation

The last experiment in this section is the two tone modulation. In this experiment, the 2 kHz signal had been added to the modulating signal in the above experiment. Theoretically, the representation of the modulated signal in time-domain and frequency domain would have been as in the figure below. In the figure, 1 kHz and 2 kHz signals were modulated with 10 kHz carrier.

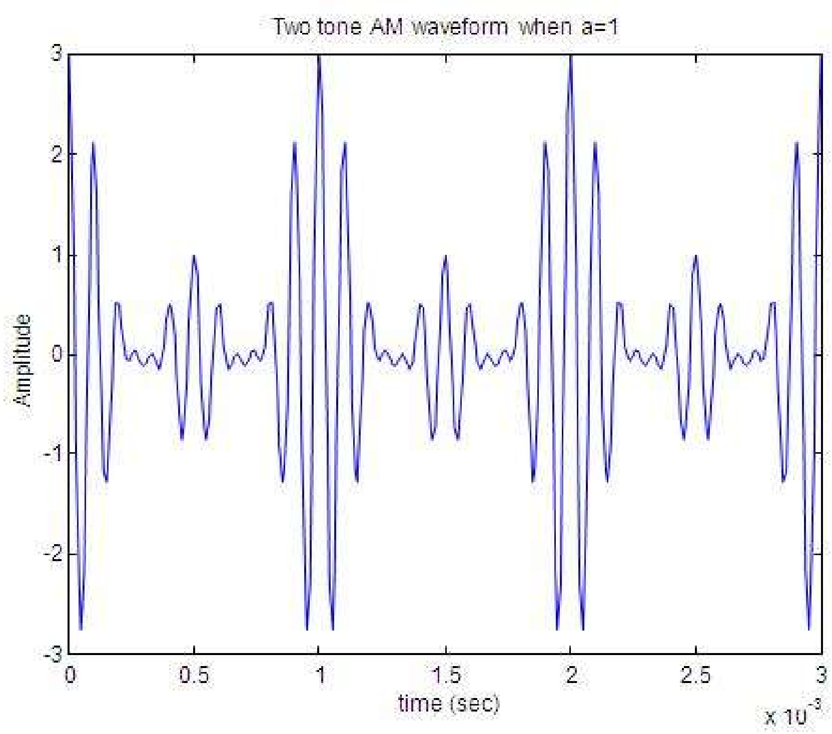


Figure 4.19

The experimental setup is shown below.

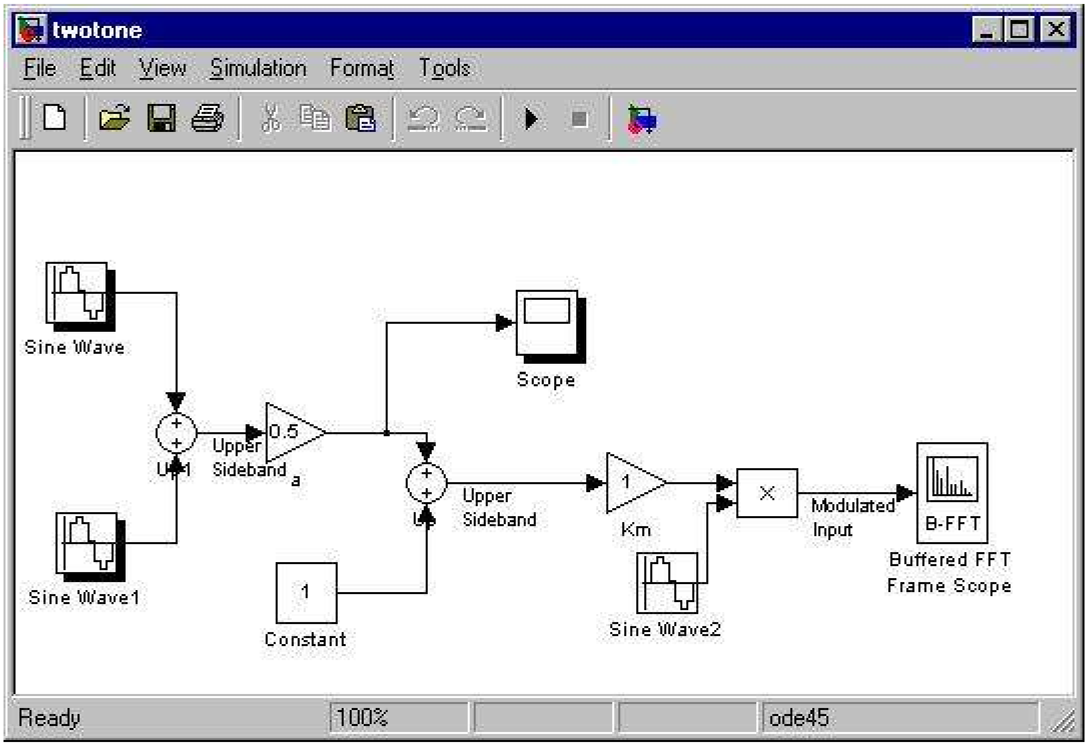


Figure 4.20

The two-tone waveform before being amplitude modulated.

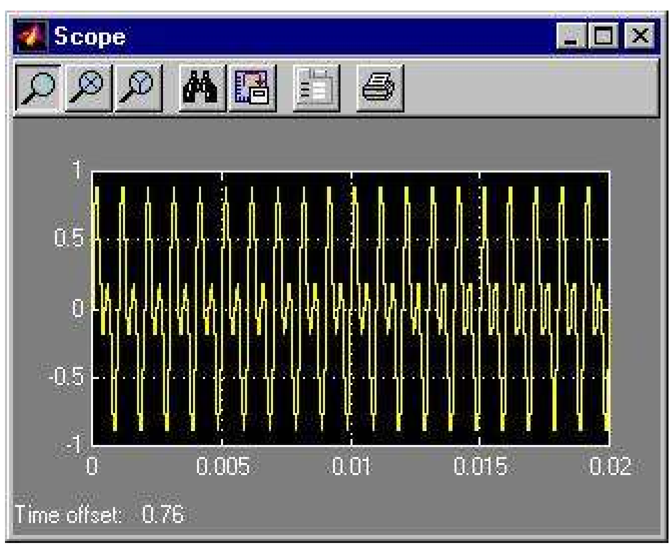


Figure 4.21

The two-tone signal is amplitude modulated using the same block model discussed in the previous section. The output spectrum is shown in figure. In this case the signals of 1 kHz and 2 kHz are modulated by a 10kHz carrier. The output spectrum is shown in figure

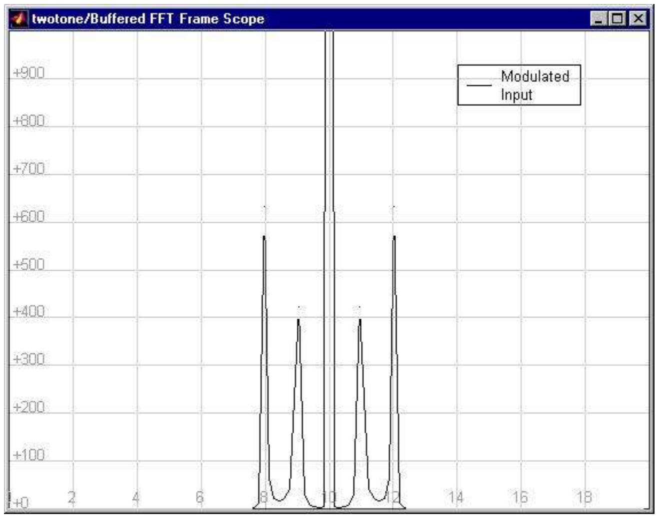


Figure 4.21

The result from the experiment was shown. The highest peak is at the carrier frequency as in the theoretical result. But there were differences on the sidebands. In the figure from MATLAB, both frequencies in the sidebands have the same magnitude, but from the experiment, the components at 9000Hz and 11000Hz have higher magnitude than the components at 8000Hz and 12000 Hz. There're also many small peaks of about 1000Hz apart in the experiment result. This might come from the incorrectly calibrated multiplier.

The final experiment in this section is to change the carrier frequency and the modulating frequency. When the carrier frequency increases, the spectrum of the modulated signal is expected to have the two sidebands centered at the new carrier frequency. And when one of the two modulating signals changes in frequency, the spectrum of the output signal should have two components move away from their original positions according to the change in frequency. The result from the experiment was shown. Both change in carrier frequency and modulating frequency is shown.

Experiment # 5

Introduction to Amplitude Modulation (MATLAB Implementation)

Objective

- To analyze the spectrum, in time and frequency domain, of Amplitude Modulation.

In this first part of the lab we will focus on a couple of simple examples and plot their spectrum, in time and in frequency domain. In second part of this lab we will write the code for Amplitude modulation with carrier and suppress carrier and then focus on two tune modulation and at the end of this lab we will write a code for single side band.

Sketch the time and frequency domain representations (magnitude only) of the following

Cos 2πft f = 1kHz

The time and frequency domain of the input signal is shown as below.

CODE:

```
%% Time specifications:
Fs = 10000;
dt = 1/Fs; StopTime = 0.5;
t = (0:dt:StopTime-dt)';
N = size(t,1);
Fc = 1000;
x = cos(2*pi*Fc*t);

subplot(2,1,1)
plot(t,x);
axis([0 1/100 -1 1]); xlabel('Time'); ylabel('Magnitude')
%% Fourier Transform:
X = fftshift(fft(x));
%% Frequency specifications:
dF = Fs/N;
f = -Fs/2:dF:Fs/2-dF;
%% Plot the spectrum: subplot(2,1,2) plot(f,abs(X)/N); xlabel('Frequency (in
hertz)'); ylabel('Magnitude')
%%
```

B. Square wave period = 1msec, amplitude = 1v

```
Fs = 1000000;
dt = 1/Fs;
StopTime = 0.5;
t    = (0:dt:StopTime-dt)'; N = size(t,1);
Fc = 1000;                                          :
x = SQUARE(2*3.14*Fc*t);

subplot(2,1,1)
plot(t,x);
axis([0 1/200 -2 2]); xlabel('Time'); ylabel('Magnitude');
   %%Fourier Transform:
X = fftshift(fft(x));
   %%Frequency specifications:
dF = Fs/N;
f = -Fs/2:dF:Fs/2-dF;
   %%Plot the spectrum:
subplot(2,1,2) plot(f,abs(X)/N); axis([-100000 100000 0 0.5]);
xlabel('Frequency (in hertz)'); ylabel('Magnitude');
```

C. $Cos^2(2\pi ft)$ f = 1kHz

```
Fs = 30000;
dt = 1/Fs;
StopTime = 0.5;
t = (0:dt:StopTime-dt)'; N = size(t,1);
Fc = 1000;
x = cos(2*pi*Fc*t); x=x.*x;

subplot(2,1,1)
plot(t,x);
xlabel('Time');
ylabel('Magnitude');
axis([0 1/100 -1 1]);
X = fftshift(fft(x));
dF = Fs/N;
f = -Fs/2:dF:Fs/2-dF; subplot(2,1,2) plot(f,abs(X)/N); axis([-5000
5000 0 0.75]) zoom on
xlabel('Frequency (in hertz)'); ylabel('Magnitude');
```

Experiment # 6

AMPLITUDE SHIFT KEYING

Aim: To generate and demodulate amplitude shift keyed (ASK) signal using MATLAB

Theory

Generation of ASK

Amplitude shift keying - ASK - is a modulation process, which imparts to a sinusoid two or more discrete amplitude levels. These are related to the number of levels adopted by the digital message. For a binary message sequence there are two levels, one of which is typically zero. The data rate is a sub-multiple of the carrier frequency. Thus the modulated waveform consists of bursts of a sinusoid. One of the disadvantages of ASK, compared with FSK and PSK, for example, is that it has not got a constant envelope. This makes its processing (eg, power amplification) more difficult, since linearity becomes an important factor. However, it does make for ease of demodulation with an envelope detector.

Demodulation

ASK signal has a well defined envelope. Thus it is amenable to demodulation by an envelope detector. Some sort of decision-making circuitry is necessary for detecting the message. The signal is recovered by using a correlator and decision making circuitry is used to recover the binary sequence.

Algorithm

Initialization commands

ASK modulation

1. Generate carrier signal.
2. Start FOR loop
3. Generate binary data, message signal(on-off form)
4. Generate ASK modulated signal.
5. Plot message signal and ASK modulated signal.
6. End FOR loop.
7. Plot the binary data and carrier.

ASK demodulation

1. Start FOR loop
2. Perform correlation of ASK signal with carrier to get decision variable
3. Make decision to get demodulated binary data. If x>0, choose '1' else choose '0'
4. Plot the demodulated binary data.

Program

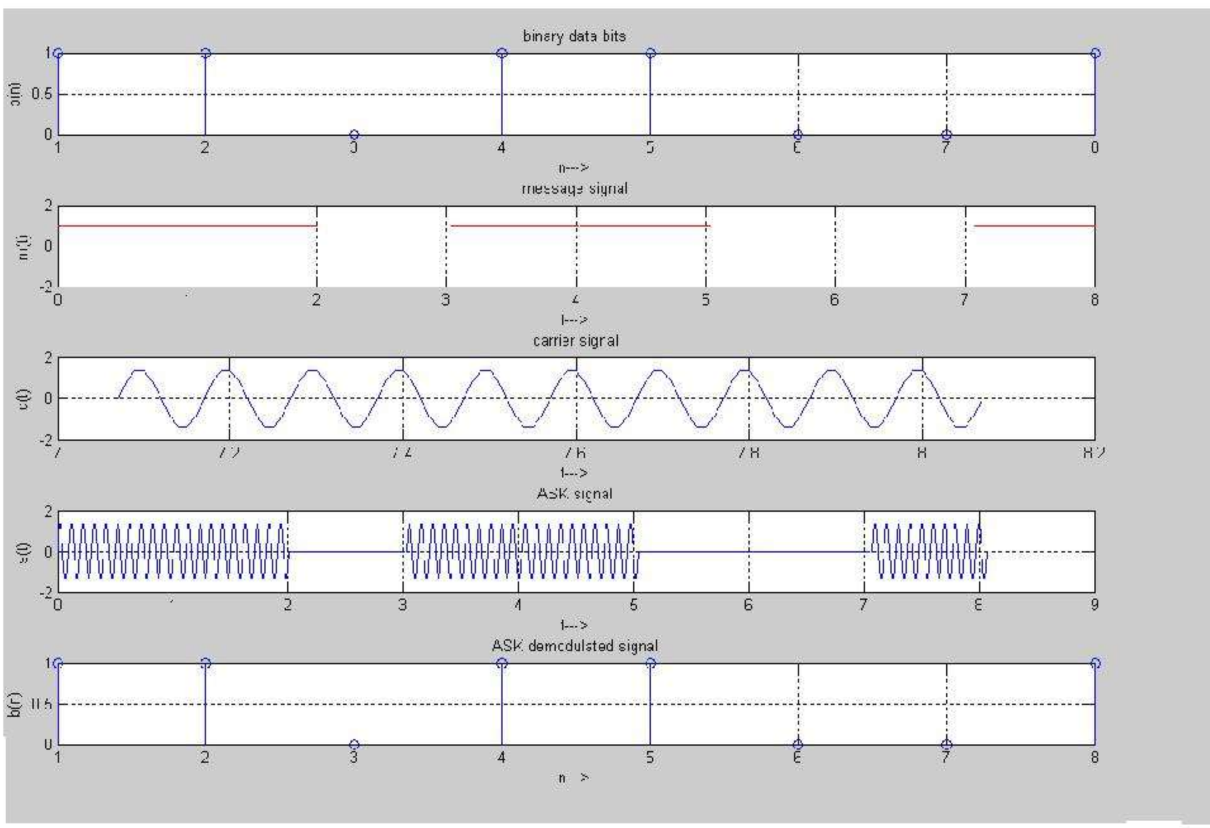
%ASK Modulation

```
clc;
clear all;
close all;
%GENERATE CARRIER SIGNAL
Tb=1; fc=10;
t=0:Tb/100:1;
c=sqrt(2/Tb)*sin(2*pi*fc*t);
%generate message signal
N=8;
m=rand(1,N);
t1=0;t2=Tb
for i=1:N
t=[t1:.01:t2]
 if m(i)>0.5
    m(i)=1;
     m_s=ones(1,length(t));
   else
     m(i)=0;
     m_s=zeros(1,length(t));
   end
message(i,:)=m_s;
%product of carrier and message
ask_sig(i,:)=c.*m_s;
t1=t1+(Tb+.01);
t2=t2+(Tb+.01);
%plot the message and ASK signal
subplot(5,1,2);axis([0 N -2 2]);
plot(t,message(i,:),'r');
title('message signal');xlabel('t--->');ylabel('m(t)');
grid on hold on;
subplot(5,1,4);plot(t,ask_sig(i,:));
title('ASK signal');xlabel('t--->');ylabel('s(t)');grid on hold on
 end
hold off
%Plot the carrier signal and input binary data
subplot(5,1,3);plot(t,c);
title('carrier signal');xlabel('t--->');ylabel('c(t)');grid on
subplot(5,1,1);stem(m);
title('binary data bits');xlabel('n--->');ylabel('b(n)');grid on
```

% ASK Demodulation

```
t1=0;t2=Tb
for i=1:N
t=[t1:Tb/100:t2]
%correlator
x=sum(c.*ask_sig(i,:));
%decision device
if x>0
    demod(i)=1;
  else
    demod(i)=0;
  end
t1=t1+(Tb+.01);
t2=t2+(Tb+.01);
 end
%plot demodulated binary data bits
subplot(5,1,5);stem(demod);
title('ASK demodulated signal'); xlabel('n--->');ylabel('b(n)');grid
on
```

Model Graphs



Experiment # 7

PHASE SHIFT KEYING

Aim: To generate and demodulate phase shift keyed (PSK) signal using MATLAB

Generation of PSK signal

PSK is a digital modulation scheme that conveys data by changing, or modulating, the phase of a reference signal (the carrier wave). PSK uses a finite number of phases, each assigned a unique pattern of binary digits. Usually, each phase encodes an equal number of bits. Each pattern of bits forms the symbol that is represented by the particular phase. The demodulator, which is designed specifically for the symbol-set used by the modulator, determines the phase of the received signal and maps it back to the symbol it represents, thus recovering the original data.

In a coherent binary PSK system, the pair of signal $S_1(t)$ and S_2 (t) used to represent binary symbols 1 & 0 are defined by

S_1 (t) = $\sqrt{2E_b/T_b}$ Cos $2\pi f_c t$
S_2 (t) = $\sqrt{2E_b/T_b}$ $(2\pi f_c t+\pi)$ = - $\sqrt{2E_b/T_b}$ Cos $2\pi f_c t$ where $0 \le t < T_b$ and
Eb = Transmitted signed energy for bit
The carrier frequency fc $=n/T_b$ for some fixed integer n.

Algorithm

Initialization commands

PSK modulation

1. Generate carrier signal.
2. Start FOR loop
3. Generate binary data, message signal in polar form
4. Generate PSK modulated signal.
5. Plot message signal and PSK modulated signal.
6. End FOR loop.
7. Plot the binary data and carrier.

PSK demodulation

1. Start FOR loop
   Perform correlation of PSK signal with carrier to get decision variable
2. Make decision to get demodulated binary data. If x>0, choose ‘1’ else choose ‘0’
3. Plot the demodulated binary data.

Program

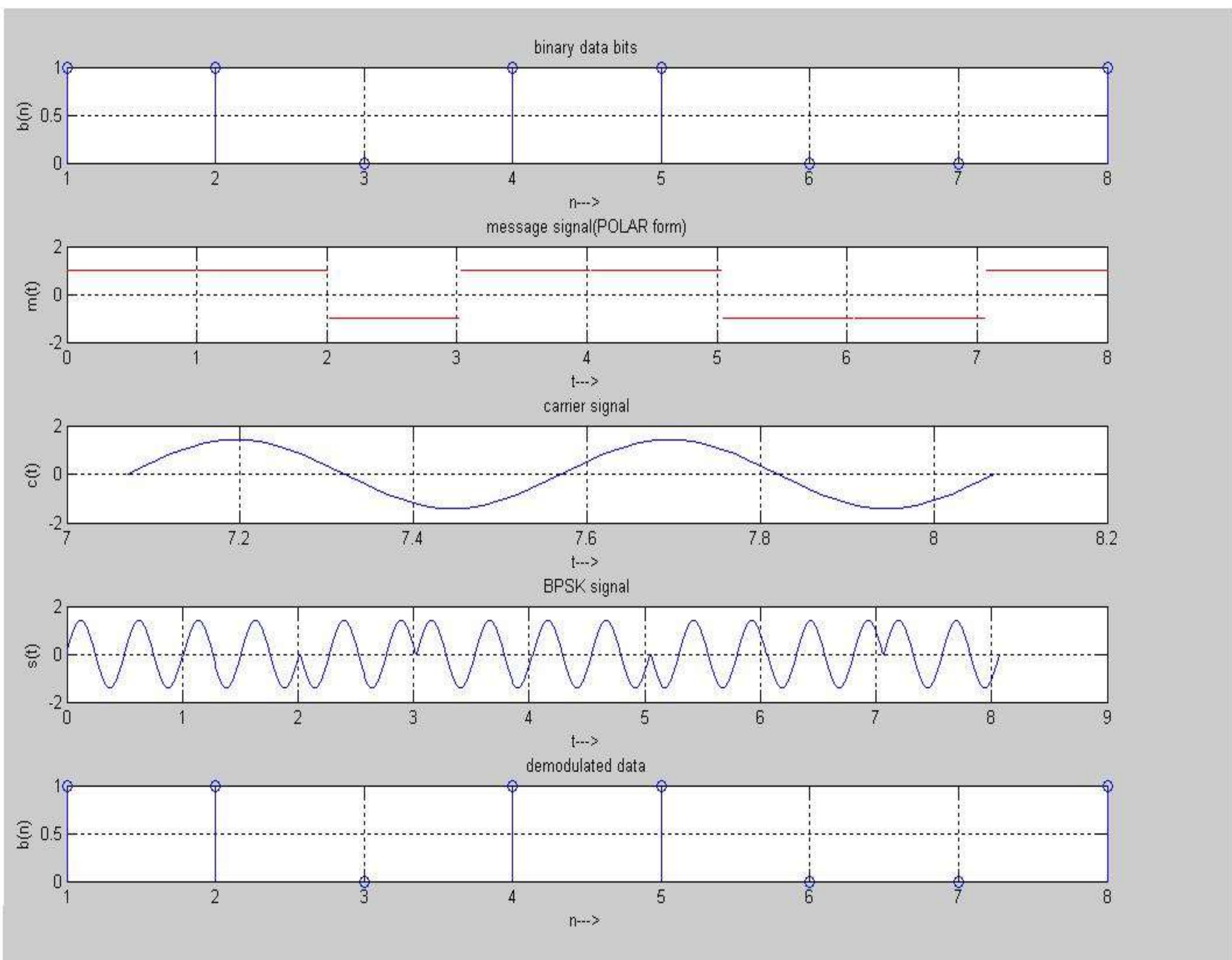
% **PSK modulation**

```
clc;
clear all;
close all;
%GENERATE CARRIER SIGNAL
Tb=1;
t=0:Tb/100:Tb;
fc=2;
c=sqrt(2/Tb)*sin(2*pi*fc*t);
%generate message signal
N=8;
m=rand(1,N);
t1=0;t2=Tb
for i=1:N
t=[t1:.01:t2]
 if m(i)>0.5
    m(i)=1;
     m_s=ones(1,length(t));
   else
     m(i)=0;
     m_s=-1*ones(1,length(t));
   end
message(i,:)=m_s;
%product of carrier and message signal
bpsk_sig(i,:)=c.*m_s;
%Plot the message and BPSK modulated signal
subplot(5,1,2);axis([0 N -2 2]);plot(t,message(i,:),'r');
title('message signal(POLAR form)');xlabel('t--->');ylabel('m(t)');
grid on; hold on;
subplot(5,1,4);plot(t,bpsk_sig(i,:));
title('BPSK signal');xlabel('t--->');ylabel('s(t)');
grid on;  hold on;
t1=t1+1.01; t2=t2+1.01;
end
hold off
%plot the input binary data and carrier signal
subplot(5,1,1);stem(m);
title('binary data bits');xlabel('n--->');ylabel('b(n)');
grid on;
subplot(5,1,3);plot(t,c);
title('carrier signal');xlabel('t--->');ylabel('c(t)');
grid on;
```

```
%   PSK Demodulation

t1=0;t2=Tb
 for i=1:N
  t=[t1:.01:t2]
%correlator
  x=sum(c.*bpsk_sig(i,:));
%decision device
  if x>0
    demod(i)=1;
  else
    demod(i)=0;
  end
   t1=t1+1.01;
  t2=t2+1.01;
 end
%plot the demodulated data bits
 subplot(5,1,5);stem(demod);
 title('demodulated data');xlabel('n--->');ylabel('b(n)'); grid on
```

Modal Graphs



Experiment # 8

FREQUENCY SHIFT KEYING

Aim: To generate and demodulate frequency shift keyed (FSK) signal using MATLAB

Theory

Generation of FSK

Frequency- shift keying (FSK) is a frequency modulation scheme in which digital information is transmitted through discrete frequency changes of a carrier wave. The simplest FSK is binary FSK (BFSK). BFSK uses a pair of discrete frequencies to transmit binary (0s and 1s) information. With this scheme, the "1" is called the mark frequency and the "0" is called the space frequency.

In binary FSK system, symbol 1 & 0 are distinguished from each other by transmitting one of the two sinusoidal waves that differ in frequency by a fixed amount.

Si (t) = $\sqrt{2E/T_b} \cos 2\pi f_1 t$ $\quad\quad 0 \leq t \leq T_b$

0 elsewhere Where i=1, 2 &
E_b=Transmitted energy/bit
Transmitted freq= $f i = (nc+i)/T_b$, and n = constant (integer), T_b = bit interval
Symbol 1 is represented by S_1 (t)
Symbol 0 is represented by S_0 (t)

Algorithm

Initialization commands

FSK modulation

1. Generate two carriers signal.
2. Start FOR loop
3. Generate binary data, message signal and inverted message signal
4. Multiply carrier 1 with message signal and carrier 2 with inverted message signal
5. Perform addition to get the FSK modulated signal
6. Plot message signal and FSK modulated signal.
7. End FOR loop.
8. Plot the binary data and carriers.

FSK demodulation

1. Start FOR loop
2. Perform correlation of FSK modulated signal with carrier 1 and carrier 2 to get two decision variables x1 and x2.

3. Make decisionon x = x1-x2 to get demodulated binary data. If x>0, choose ‘1’ else choose ‘0’.
4. Plot the demodulated binary data.

Program

% FSK Modulation

```
clc;
clear all;
close all;
%GENERATE CARRIER SIGNAL
Tb=1; fc1=2;fc2=5;
t=0:(Tb/100):Tb;
c1=sqrt(2/Tb)*sin(2*pi*fc1*t);
c2=sqrt(2/Tb)*sin(2*pi*fc2*t);
%generate message signal
N=8;
m=rand(1,N);
t1=0;t2=Tb
for i=1:N
  t=[t1:(Tb/100):t2]
  if m(i)>0.5
    m(i)=1;
    m_s=ones(1,length(t));
    invm_s=zeros(1,length(t));
  else
    m(i)=0;
    m_s=zeros(1,length(t));
    invm_s=ones(1,length(t));
  end
  message(i,:)=m_s;
%Multiplier
  fsk_sig1(i,:)=c1.*m_s;
  fsk_sig2(i,:)=c2.*invm_s;
  fsk=fsk_sig1+fsk_sig2;
%plotting the message signal and the modulated signal
subplot(3,2,2);axis([0 N -2 2]);plot(t,message(i,:),'r'); title('message
signal');xlabel('t---->');ylabel('m(t)');grid on;hold on;
subplot(3,2,5);plot(t,fsk(i,:));
title('FSK signal');xlabel('t---->');ylabel('s(t)');grid on;hold on;
t1=t1+(Tb+.01); t2=t2+(Tb+.01); end

hold off
%Plotting binary data bits and carrier signal
subplot(3,2,1);stem(m);
title('binary data');xlabel('n---->');
ylabel('b(n)');grid on;
subplot(3,2,3);plot(t,c1);
title('carrier signal-1');xlabel('t---->');ylabel('c1(t)');grid on;
subplot(3,2,4);plot(t,c2);
title('carrier signal-2');xlabel('t---->');ylabel('c2(t)');grid on;
```

% FSK Demodulation

```
 t1=0;t2=Tb
  for i=1:N
  t=[t1:(Tb/100):t2]
```

```
%correlator
  x1=sum(c1.*fsk_sig1(i,:));
  x2=sum(c2.*fsk_sig2(i,:));
  x=x1-x2;
%decision device
  if x>0
    demod(i)=1;
  else
    demod(i)=0;
  end
   t1=t1+(Tb+.01);
  t2=t2+(Tb+.01);
 end
%Plotting the demodulated data bits
 subplot(3,2,6);stem(demod);
 title(' demodulated data');xlabel('n---->');ylabel('b(n)'); grid on;
```

Modal Graphs