2023

DIGITAL SIGNAL PROCESSING LABORATORY

MANUAL

IMPLEARN

List of Experiments

1. Introduction To MATLAB

AIM

To familiarize with MATLAB and its working

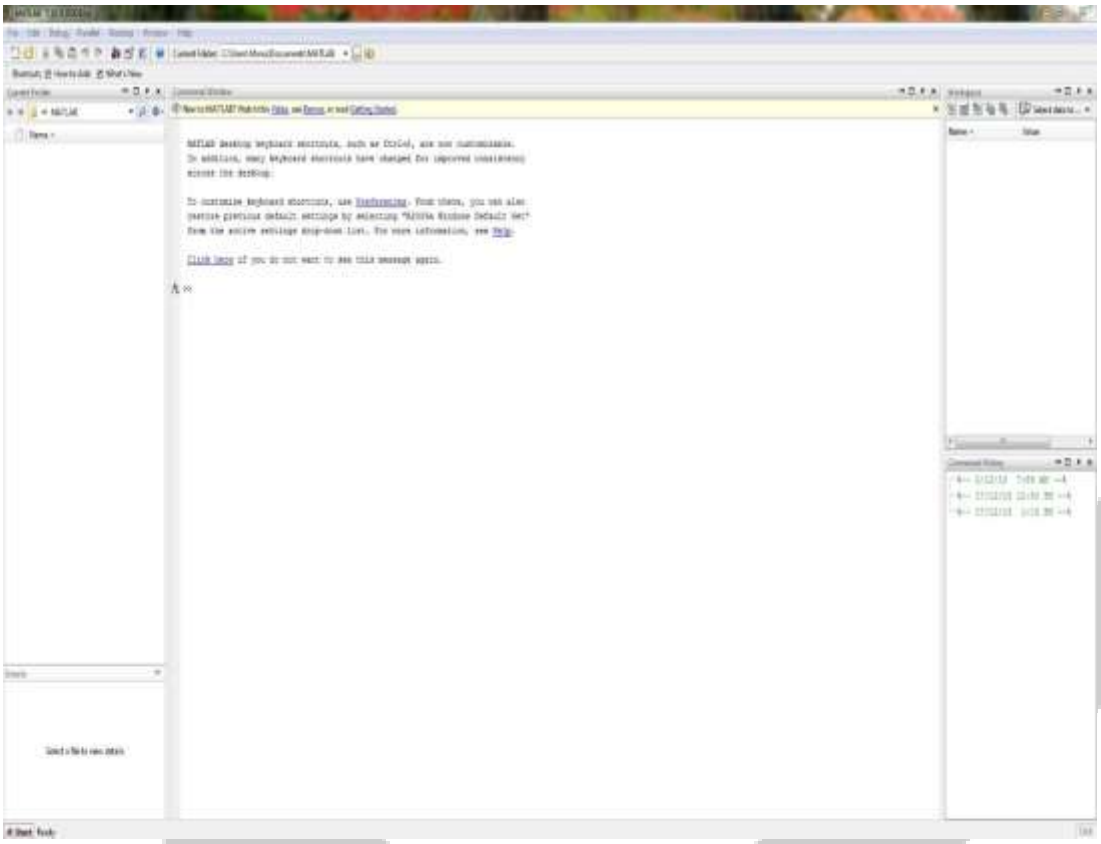
Overview of the MATLAB Environment

MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation. Using the MATLAB product, technical computing problems can be solved faster than the traditional programming languages, such as C, C++, and Fortran.

MATLAB can be used in a wide range of applications, including signal and image processing, communications, control design, test and measurement, financial modelling 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 documenting and sharing the work. MATLAB code can be integrated with other languages and applications, and distribute MATLAB algorithms and applications. Features include:

- High-level language for technical computing
- Development environment for managing code, files, and data
- Interactive tools for iterative exploration, design, and problem solving
- Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, and numerical integration
- 2-D and 3-D graphics functions for visualizing data
- Tools for building custom graphical user interfaces



```
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

MATLAB desktop keyboard shortcuts, such as Ctrl+S, are now customizable.
In addition, many keyboard shortcuts have changed for improved consistency
across the desktop.

To customize keyboard shortcuts, use Preferences. From there, you can also
restore previous default settings by selecting "R2009a Windows Default Set"
from the active settings drop-down list. For more information, see Help.

Click here if you do not want to see this message again.

>>
```

Fig. 1. MATLAB window

Functions for integrating MATLAB based algorithms with external applications and languages, such as C, C++, Fortran, Java™, COM, and Microsoft® Excel

Starting MATLAB

We can enter MATLAB by double-clicking on the MATLAB shortcut *icon* (MATLAB 7.0.4) on the Windows desktop. When MATLAB starts, a special window called the MATLAB desktop appears. The desktop is a window that contains *other* windows. The major tools within or accessible from the desktop are:

- The Command Window
- The Command History
- The Workspace
- The Current Directory
- The Help Browser
- The Start button

When MATLAB is started for the first time, the screen looks like the one that shown in the Fig. 1.1 .This illustration also shows the default configuration of the MATLAB desktop. The arrangement of tools and documents can be customized to suit the needs.

MATLAB desktop contains the prompt (>>) in the Command Window. Usually, there are 2 types of prompt:

Array Creation

To create an array with four elements in a single row, separate the elements with either a comma (,) or a space.

a = [1 2 3 4]

returns

a =

1 2 3 4

This type of array is a *row vector*.

To create a matrix that has multiple rows, separate the rows with semicolons.

a = [1 2 3; 4 5 6; 7 8 10]

a =

1 2 3

4 5 6

7 8 10

Another way to create a matrix is to use a function, such as ones, zeros, or rand. For example, create a 5-by-1 column vector of zeros.

z = zeros(5,1)

z =

0

0

0

0

0

Matrix and Array Operations

MATLAB allows us to process all of the values in a matrix using a single arithmetic operator or function.

a + 10

```
ans =
11 12 13
14 15 16
17 18 20
sin(a)
ans =
0.8415 0.9093 0.1411
-0.7568 -0.9589 -0.2794
0.6570 0.9894 -0.5440
```

To transpose a matrix, use a single quote ('):

```
a'
ans =
1 4 7
2 5 8

3 6 10
```

We can perform standard matrix multiplication, which computes the inner products between rows and columns, using the * operator. For example, confirm that a matrix times its inverse returns the identity matrix:

```
p = a*inv(a)
p =
1.0000 0 -0.0000
0 1.0000 0
0 0 1.0000
```

Notice that p is not a matrix of integer values. MATLAB stores numbers as floating-point values, and arithmetic operations are sensitive to small differences between the actual value and its floating-point representation. We can display more decimal digits using the format command:

```
format long
p = a*inv(a)
p =
1.000000000000000 0 -0.000000000000000
0 1.000000000000000 0
0 0 0.999999999999998
```

Reset the display to the shorter format using format short format affects only the display of numbers, not the way MATLAB computes or saves them. To perform element-wise multiplication rather than matrix multiplication,
use the .* operator:

```
p = a.*a
p =

1 4 9
16 25 36
49 64 100
```

The matrix operators for multiplication, division, and power each have a corresponding array operator that operates element-wise. For example, raise each element of a to the third power:

```
a.^3
ans =
1 8 27
64 125 216
343 2 1000
```

1.1.MATLAB Script files

MATLAB offers the possibility to store commands in special files - called M-files – with the ending .m. There are two types of M-files: Scripts and Functions. To edit an M-file, select ‘New’ or ‘Open’ from the file-menu. In each case the MATLAB Editor window will be opened and we can continue by editing either a new file or, after opening, an existing file. When the MATLAB Editor is open, we can load further files or save and close opened files by selecting the appropriate item from the file menu of the Editor. Select ‘Save As’ and the browser if we save a new file or if we want to change the name of an existing file, or ‘Save’ if we have edited an existing file whose name we want to leave unchanged. The directory in which the files should be saved is usually the working directory

In a Script M-file we just write a sequence of commands separated by semicolons (result not displayed) or commas (result displayed). A Script M-file is treated as if the commands were written in the Command window. All variables defined in the workspace are accessed and the computations are carried out in the workspace.

Example: To multiply a complex number c by e^{iq} and to extract magnitude and polar angle of the product.

If both c and q are defined in the workspace under the variable names c and theta, type the following commands into a file,

```
product=c1*e^(i*theta);
magn=abs(product);phi=angle(product);
```

Save the file as product.m and call it at appropriate stages of the calculation. We can access the variables magn and phi in the workspace which are assigned the computed values. For calling the Script M-file there are three options:

- If the file is opened in the MATLAB Editor select 'Debug->Run' from the Editor's 'Tools'-menue
- Type `product' and press the Enter key in the workspace

A Script M-file may contain an arbitrary number of commands. The advantage of using Script M-files is that we can repeat calculations easily whenever new data have been computed, and that it is easy to make changes.

Example:

```
disp('Plotting the sine is fun');
fplot('sin(x)',[-2*pi,2*pi],'k');
hold on
fplot('sin(2*x)',[-2*pi,2*pi],'r');
```

If this file is saved as sinescript.m and executed, we are informed that Plotting the sine is fun in the command window, and two sine curves in the interval -2p £ x £ 2p are plotted. Here the command hold on has the effect that previous plots in the figure window are kept when a new plotting command is executed. Type hold off if we want to clear the window for the next plot. The command fplot is reserved for Matlab functions which can be buildt in functions such as sine.m as well as user defined function M-files (see below). Note the apostrophes: the first argument passed to fplot is the filename and therefore must be a character string.

2. Simulation of Basic signals in MATLAB

AIM

To generate the following waveforms in discrete & continuous form. cosine wave, sawtooth wave, triangular wave, square wave, sinc wave, exponential wave, unit step wave, unit impulse wave, signum wave & ramp wave.

PROGRAM

```
clc; clear
all; close
all;
f=input ('enter the frequency');
t=-1:.01:1;
a=sin (2*pi*f*t);
subplot (3,4,1);
plot (t,a);
xlabel('time index');
ylabel('amplitude');
title('sine wave');
b=cos(2*pi*f*t);
subplot(3,4,2);
plot(t,b);
xlabel('time index');
ylabel('amplitude');
title('cosine wave');
c=sawtooth(2*pi*f*t);
subplot(3,4,3);
plot(t,c);
xlabel('time index');
ylabel('amplitude');
title('sawtooth');
d=sawtooth(2*pi*f*t,.5);
subplot(3,4,4);
plot(t,d);
```

```
        xlabel('time index');
        ylabel('amplitude');
title('triangular wave')
e=square(2*pi*f*t);
subplot(3,4,5);
plot(t,e);   xlabel('time   index');
ylabel('amplitude');  title('square
wave'); t=-10:.01:10;
g=sinc(t/2); subplot(3,4,6);
plot(t,g);   xlabel('time   index');
ylabel('amplitude');      title('sinc
wave');
t=-1:.05:1;
h=exp(-2*t);
subplot(3,4,7);
plot(t,h);
xlabel('time');
title('exponential');
i=sign(2*pi*f*t);
subplot(3,4,8);
plot(t,i);
xlabel('time');
ylabel('amplitude');
        title('signum function');
        t=-10:10;
        j=[zeros(1,10),ones(1,11)];
        subplot(3,4,9);
        plot(t,j);
        xlabel('time index');
```

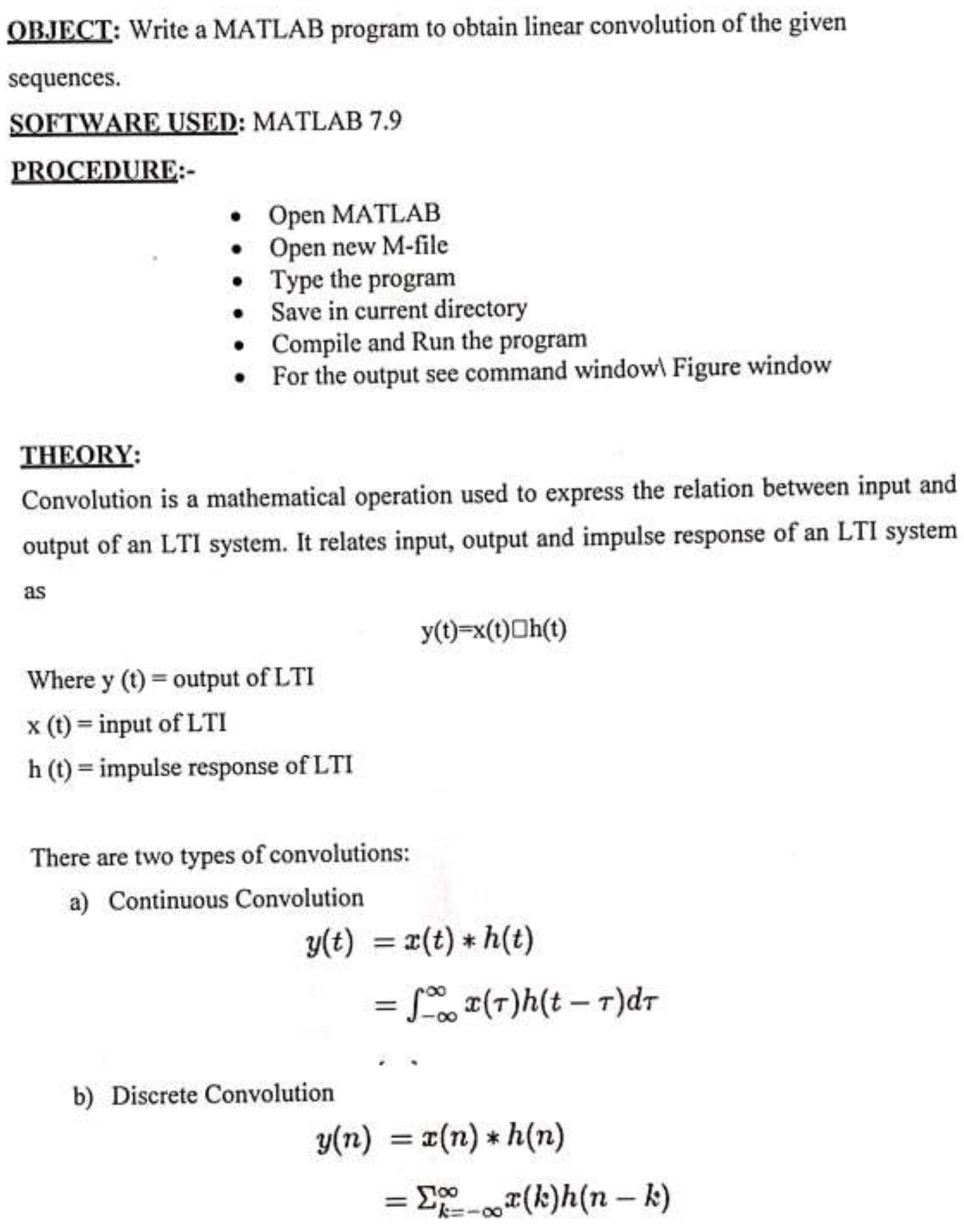
b)Discrete Waveform

```
clc;
ylabel('amplitude');
title('unit step'); t=-10:10;
k=[zeros(1,10),1,zeros(1,10)]; subplot(3,4,10);
plot(t,k); xlabel('time index'); ylabel('amplitude'); title('unit impulse'); t=-10:10;
x=input('enter the amplitude'); ramp=(x*t);
subplot(3,4,11);
plot(t,ramp);
xlabel('time'); ylabel('amplitude');
title('ramp function');
clear all;
close all;
f=input('enter the frequency'); t=-
1:.1:1;
a=sin(2*pi*f*t);
subplot(3,4,1);
stem(t,a); xlabel('time
index');
ylabel('amplitude');
title('sine wave');
b=cos(2*pi*f*t);
subplot(3,4,2);
stem(t,b);
xlabel('time                    index');
ylabel('amplitude');
title('cosine                    wave');
c=sawtooth(2*pi*f*t);
subplot(3,4,3);
```

```
stem(t,c);   xlabel('time  index');
ylabel('amplitude');
title('sawtooth');

d=sawtooth(2*pi*f*t,.5)
subplot(3,4,4);

stem(t,d);   xlabel('time  index');
ylabel('amplitude');

title('triangular                    wave');
e=square(2*pi*f*t);
subplot(3,4,5);

stem(t,e);

xlabel('time index');


        ylabel('amplitude');
        title('square  wave');
        t=-10:.1:10;
        g=sinc(t/2);
        subplot(3,4,6);
        stem(t,g);
        xlabel('time  index');
        ylabel('amplitude');
        title('sinc wave');
        t=-1:.5:1;
        h=exp(-2*t);
        subplot(3,4,7);

stem(t,h);
xlabel('time');
ylabel('amplitude');
title('exponential');
```

```
i=sign(2*pi*f*t);
subplot(3,4,8);

stem(t,i);

xlabel('time');    ylabel('amplitude');
title('signum function'); t=-10:10;

j=[zeros(1,10),ones(1,11)];
subplot(3,4,9);

stem(t,j);   xlabel('time   index');
ylabel('amplitude');      title('unit
step');

t=-10:10;

k=[zeros(1,10),1,zeros(1,10)];
subplot(3,4,10);
        stem(t,k);
        xlabel('time index');
        ylabel('amplitude');
        title('unit impulse');
        t=-10:10;
        x=input('enter the amplitude');
        ramp=(x*t);
        subplot(3,4,11);
        stem(t,ramp);
        xlabel('time');
        ylabel('amplitude');
        title('ramp function');
```

3. Linear convolution & circular convolution



OBJECT: Write a MATLAB program to obtain linear convolution of the given sequences.

SOFTWARE USED: MATLAB 7.9

PROCEDURE:-

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

THEORY:

Convolution is a mathematical operation used to express the relation between input and output of an LTI system. It relates input, output and impulse response of an LTI system as

y(t)=x(t)□h(t)

Where y (t) = output of LTI

x (t) = input of LTI

h (t) = impulse response of LTI

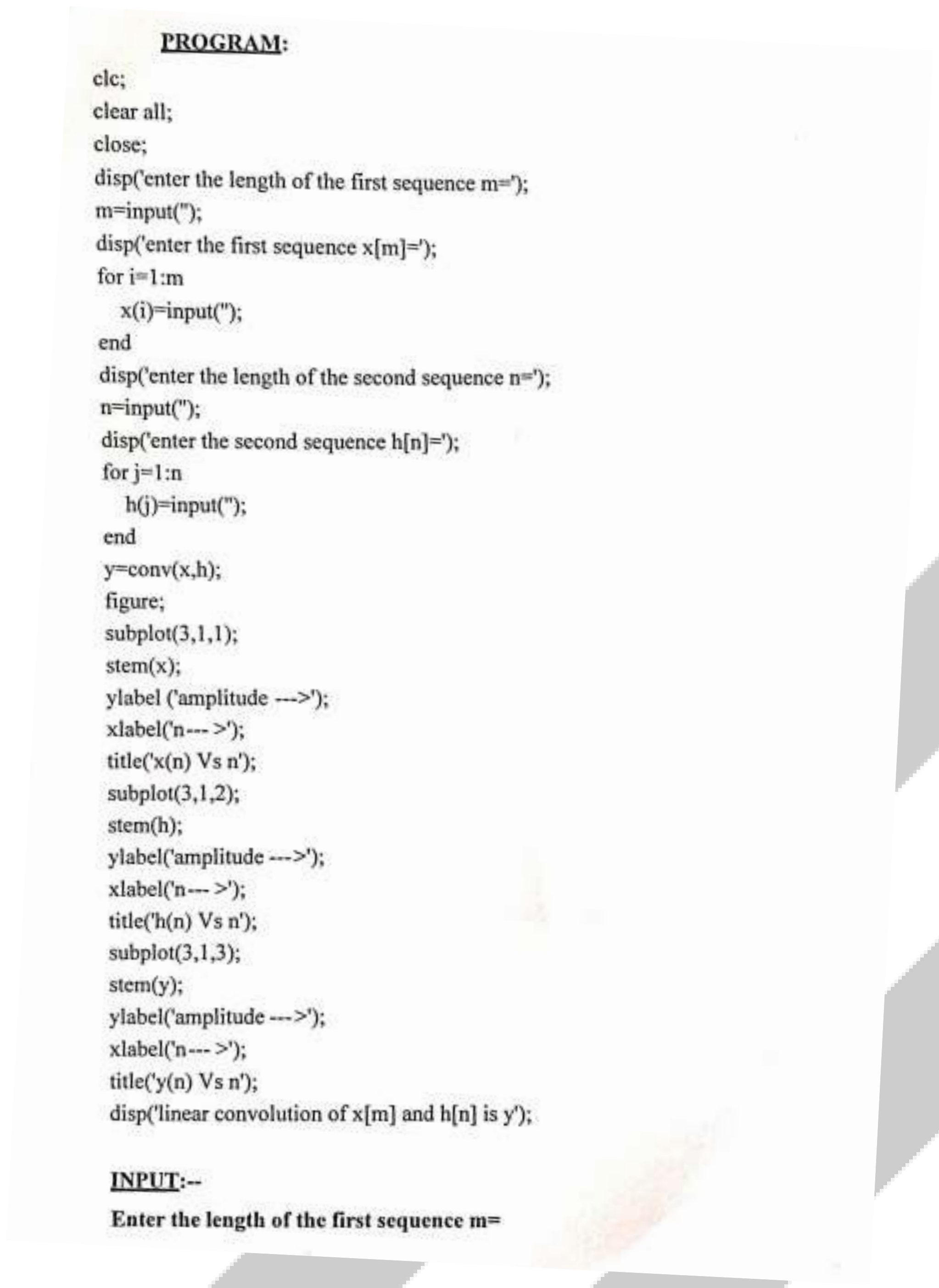
There are two types of convolutions:

a) Continuous Convolution

$$\begin{aligned} y(t) &= x(t) * h(t) \\ &= \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \end{aligned}$$

b) Discrete Convolution

$$\begin{aligned} y(n) &= x(n) * h(n) \\ &= \Sigma_{k=-\infty}^{\infty} x(k)h(n-k) \end{aligned}$$

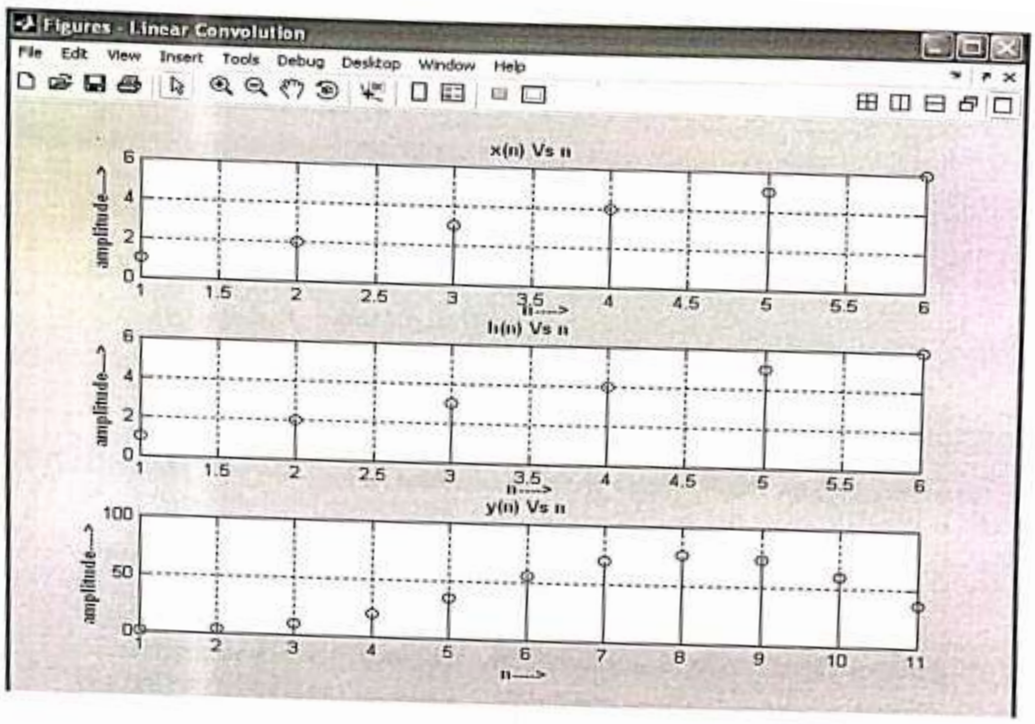


PROGRAM:

```
clc;
clear all;
close;
disp('enter the length of the first sequence m=');
m=input('');
disp('enter the first sequence x[m]=');
for i=1:m
   x(i)=input('');
end
disp('enter the length of the second sequence n=');
n=input('');
disp('enter the second sequence h[n]=');
for j=1:n
   h(j)=input('');
end
y=conv(x,h);
figure;
subplot(3,1,1);
stem(x);
ylabel ('amplitude --->');
xlabel('n--- >');
title('x(n) Vs n');
subplot(3,1,2);
stem(h);
ylabel('amplitude --->');
xlabel('n--- >');
title('h(n) Vs n');
subplot(3,1,3);
stem(y);
ylabel('amplitude --->');
xlabel('n--- >');
title('y(n) Vs n');
disp('linear convolution of x[m] and h[n] is y');
```

INPUT:--

Enter the length of the first sequence m=



PRECAUTIONS:

1) Program must be written carefully to avoid errors.

2) Programs can never be saved as standard function name.

3) Functions in MATLAB are case sensitive so commands must be written in proper format.

```
6
Enter the length of first sequence x[m]=
1
2
3
4
5
6
Enter the length of the second sequence n=
6
Enter the length of second sequence h[n]=
1
2
3
4
5
6
```

OUTPUT:-

```
Linear convolution of x[m] and h[n] is y=
1 4 10 20 35 56 70 76 73 60 36
```

RESULTS: - Thus the program for linear convolution is written using MATLAB and verified.

4. Sampling theorem verification

AIM

To verify and plot the sampling theorem in MATLAB

THEORY

The sampling theorem can be defined as the conversion of an analog signal into a discrete form by taking the sampling frequency as twice the input analog signal frequency. Input signal frequency denoted by Fm and sampling signal frequency denoted by Fs.

The output sample signal is represented by the samples. These samples are maintained with a gap, these gaps are termed as sample period or sampling interval (Ts). And the reciprocal of the sampling period is known as "sampling frequency" or "sampling rate". The number of samples is represented in the sampled signal is indicated by the sampling rate.

Sampling frequency **Fs=1/Ts**

Sampling theorem states that "continues form of a time-variant signal can be represented in the discrete form of a signal with help of samples and the sampled (discrete) signal can be recovered to original form when the sampling signal frequency Fs having the greater frequency value than or equal to the input signal frequency Fm.

If the sampling frequency (Fs) equals twice the input signal frequency (Fm), then such a condition is called the Nyquist Criteria for sampling. When sampling frequency equals twice the input signal frequency is known as "Nyquist rate".

Fs=2Fm

If the sampling frequency (Fs) is less than twice the input signal frequency, such criteria called an Aliasing effect.

Fs<2Fm

So, there are three conditions that are possible from the sampling frequency criteria. They are sampling, Nyquist and aliasing states.

Nyquist sampling theorem states that the sampling signal frequency should be double the input signal's highest frequency component to get distortion less output signal. As per the scientist's name, Harry Nyquist this is named as Nyquist sampling theorem.

Fs=2Fm

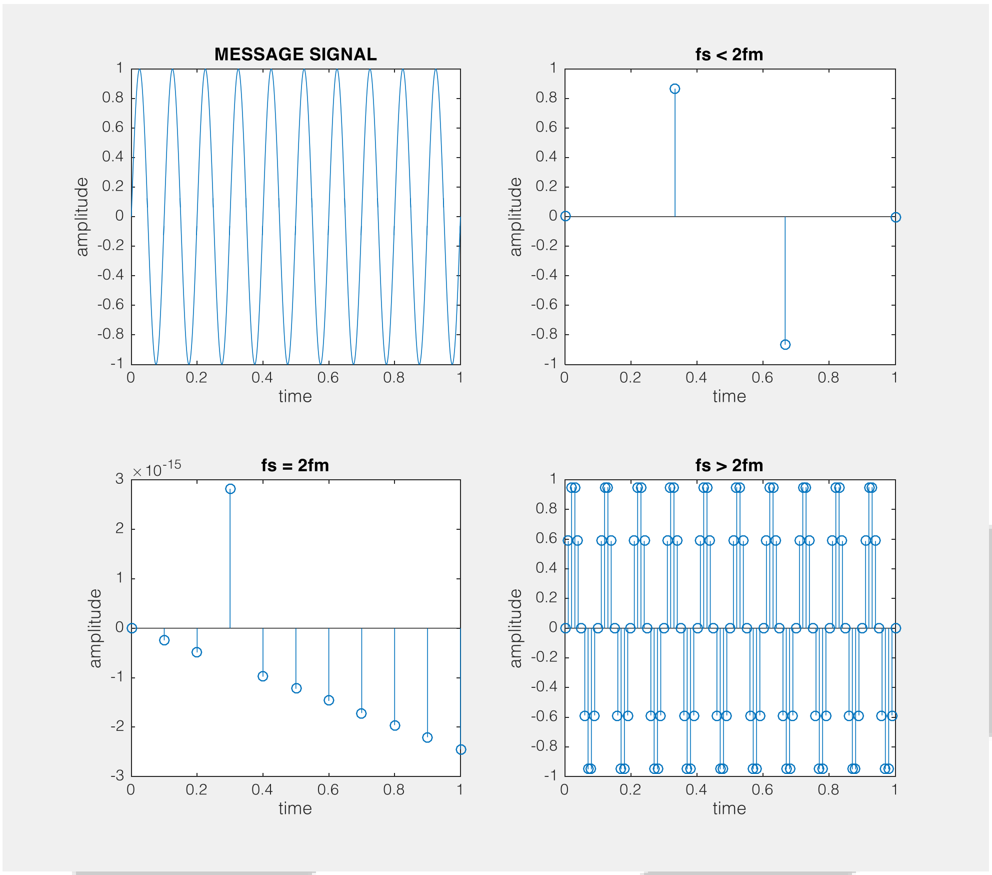
PROGRAM

```
clc;
clear all;
t=0:00.001:1;
fm=input('Enter the modulating signal frequency :');
x=sin(2*pi*fm*t);
```

```
fs1=input('Enter the sampling frequency < modulating signal
:');
fs2=input('Enter the sampling frequency = modulating signal
:');
fs3=input('Enter the sampling frequency > modulating signal
:');
% sampling at fs < 2fm
n=0:(1/fs1):1;
x1=sin(2*pi*fm*n);

% sampling at fs = 2fm
n=0:(1/fs2):1;
x2=sin(2*pi*fm*n);

% sampling at fs > 2fm
n=0:(1/fs3):1;
x3=sin(2*pi*fm*n);

% Plotting signals
subplot(2,2,1);
    plot(t,x);
xlabel('time');
ylabel('amplitude');
title('MESSAGE SIGNAL');
subplot(2,2,2);
    stem(n,x1);
xlabel('time');
ylabel('amplitude');
title(' fs < 2fm ');
subplot(2,2,3);
    stem(n,x2);
xlabel('time');
ylabel('amplitude');
title(' fs = 2fm ');
subplot(2,2,4);
    stem(n,x3);
xlabel('time');
ylabel('amplitude');
title(' fs > 2fm ');
```

OUTPUT

```
Enter the modulating signal frequency :10

Enter the sampling frequency < modulating signal :3

Enter the sampling frequency = modulating signal :10

Enter the sampling frequency > modulating signal :100
```



Result

5. Verification of the Properties of DFT

AIM

Generate and appreciate a DFT matrix.

1. Write a function that returns the N point DFT matrix $\mathbf{V}_N$ for a given N.
2. Plot its real and imaginary parts of $\mathbf{V}_N$ as images using *imshow* commands for $N = 16$, $N = 64$ and $N = 1024$
3. Compute the DFTs of 16-point, 64-point and 1024-point random sequences using the above matrices.
4. Use some iterations to plot the times of computation against γ. Plot and understand this curve. Plot the times of computation for the *fft* function over this curve and appreciate the computational saving with FFT.

• Circular Convolution.

• Parseval's Theorem

1. For the complex random sequences $x_1[n]$ and $x_2[n]$,

PROGRAM

% Part 1: (For N = 16)

```
clear all
close all
clc
%Part 1 for N=16;
N=16;
x=randi(10,N,1)+i.*randi(10,N,1);
%To generate the twiddle factor
tic %Timer is ON to compute the overall execution time
W_N=exp(-2*1j*pi/N);
%To generate the twiddle factor matrix
for i=1:N
    for j=1:N
        V_N(i,j)=W_N.^((i-1)*(j-1));
    end
end
time(1)=toc;
%2. To pot the real and imaginary parts of Vn
% V_real=real(V_N);
% V_imaginary=imag(V_N);
subplot(2,1,1)
imshow(real(V_N));
title('Real parts of twiddle factor matrix as image')
subplot(2,1,2)
imshow(imag(V_N));
```

```
title('Imaginary parts of twiddle factor matrix as image')
tic
X_direct1=fft(x,N)
time_fft(1)=toc;
```

% Part 2: (For N=64)

```
N=64;
x=randi(10,N,1)+i.*randi(10,N,1);
%To generate the twiddle factor
tic %Timer is ON to compute the overall execution time
W_N=exp(-2*1j*pi/N);
%To generate the twiddle factor matrix
for i=1:N
    for j=1:N
    V_N(i,j)=W_N.^((i-1)*(j-1));
    end
end
time(2)=toc;
%2. To pot the real and imaginary parts of Vn
figure
subplot(2,1,1)
imshow(real(V_N));
title('Real parts of twiddle factor matrix as image')
subplot(2,1,2)
imshow(imag(V_N));
title('Imaginary parts of twiddle factor matrix as image')
tic
X_direct2=fft(x,N);
time_fft(2)=toc;
```

% Part 3: (For N=1024)

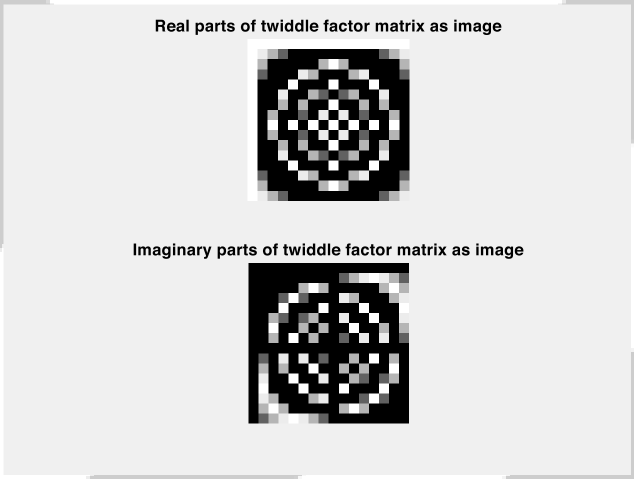
```
N=1024;
x=randi(10,N,1)+i.*randi(10,N,1);
%To generate the twiddle factor
tic %Timer is ON to compute the overall execution time
W_N=exp(-2*1j*pi/N);
%To generate the twiddle factor matrix
for i=1:N
    for j=1:N
    V_N(i,j)=W_N.^((i-1)*(j-1));
    end
end
X=V_N*x;
time(3)=toc;
%2. To pot the real and imaginary parts of Vn
figure
subplot(2,1,1)
imshow(real(V_N));
```

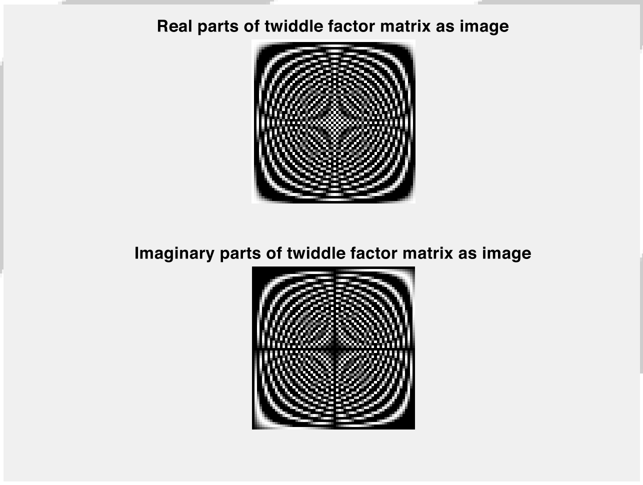
```
title('Real parts of twiddle factor matrix as image')
subplot(2,1,2)
imshow(imag(V_N));
title('Imaginary parts of twiddle factor matrix as image')
tic
X_direct3=fft(x,N);
time_fft(3)=toc;
```

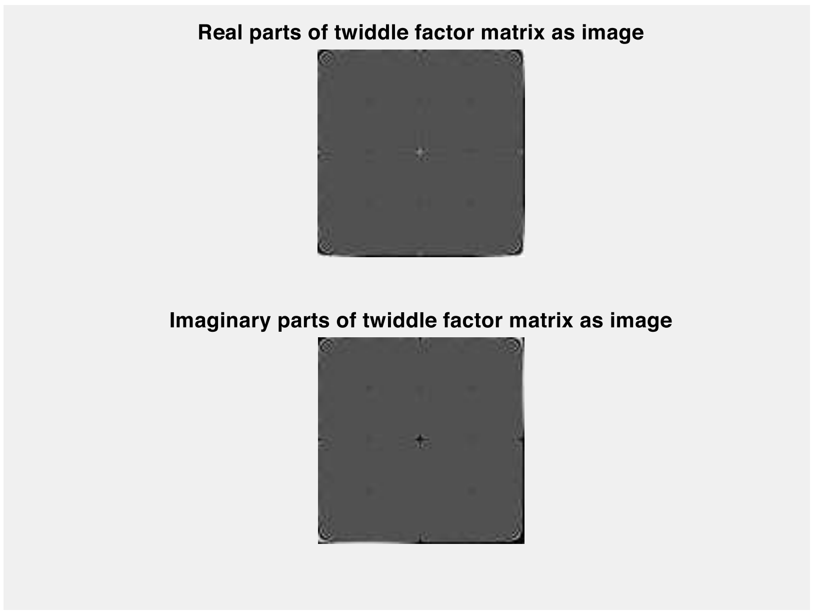
%Part 4: (Plotting time performance curve)

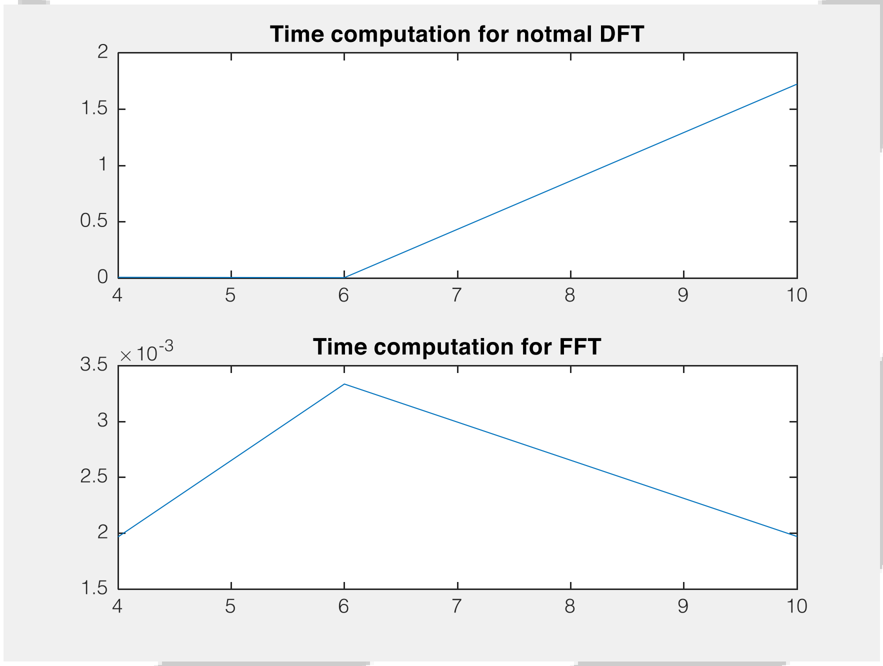
```
figure
gamma=[4,6,10];
subplot(2,1,1)
plot(gamma,time)
title('Time computation for notmal DFT')
subplot(2,1,2)
plot(gamma,time_fft)
title('Time computation for FFT')
```

Output









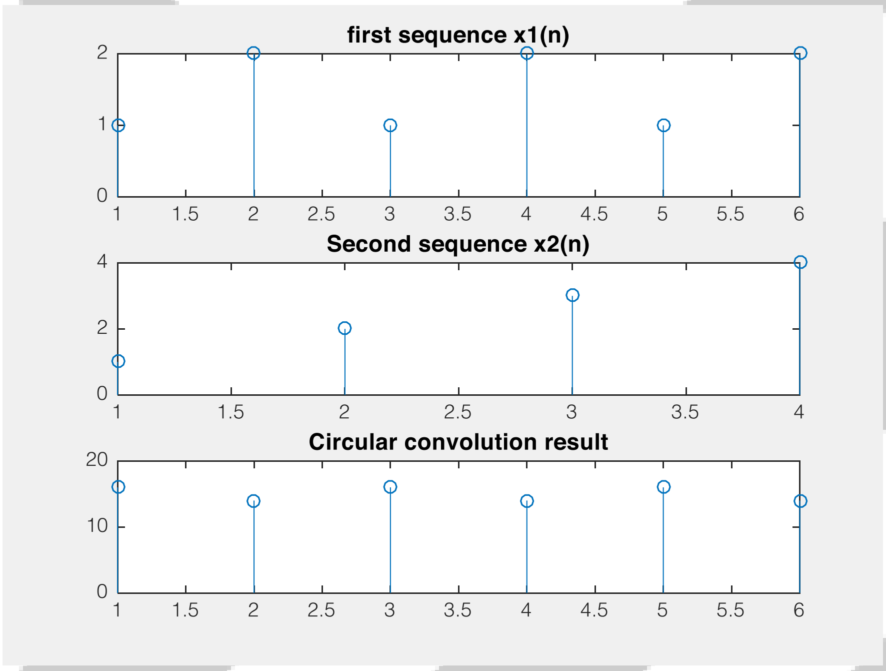
Circular convolution

```
clc
clear all
x1=input('Enter the first sequence x1(n) : ');
x2=input('Enter the second sequence x2(n) : ');
N1=length(x1);
N2=length(x2);
N=max(N1,N2);
disp('circular convuoultion of the result is ')
c=cconv(x1,x2,N)
% Plotting the sequences
subplot(3,1,1);
```

```
stem(x1);
title('first sequence x1(n)');
subplot(3,1,2);
stem(x2);
title('Second sequence x2(n)');
subplot(3,1,3);
stem(c);
title('Circular convolution result');
```

Output

```
Enter the first sequence x1(n) : [1 2 1 2 1 2]

Enter the second sequence x2(n) : [1 2 3 4]

circular convuoultion of the result is

c =

   16    14    16    14    16    14
```



Parseval's theorem :

It is an important theorem used to relate the product or square of functions using their respective Fourier series components. Theorems like Parseval's theorem are helpful in signal processing, studying behaviours of random processes, and relating functions from one domain to another. Parseval's theorem states that the integral of the square of its function is equal to the square of the function's Fourier components.

This theorem is helpful when dealing with signal processing and when observing the behaviour of random processes. When signals are challenging to process with time as their domain, transforming the domain is the best course of action so that the values are easier to work with. This is where Fourier transforms and the Parseval's theorem enters. Taking a look at the equation of Parseval's theorem for continuous functions, a signal's power (or energy) will be much easier to capitalize on and will provide insight on how these quantities behave through a different domain, say frequency.

$$\sum_{i=0}^{n-1} |x_i|^2 = \frac{1}{n}\sum_{k=0}^{n-1} |x_k|^2$$

Program

```
%Program to verify the Parseval's theorem
clear all
close all
clc
N=5000;
x=randi(10,N,1)+1i.*randi(10,N,1);
y=randi(10,N,1)+1i.*randi(10,N,1);
%To find x(n).y*(n)
yc=conj(y);
disp('Left hand side of Parseval's theorem')
lhs=sum(x.*yc)
X=fft(x);
Y=fft(y);
Yc=conj(Y);
%To find (1/N).X(n).Y*(n)
disp('right hand side of Parseval's theorem')
rhs=(1/N).*sum(X.*Yc)
```

Output

```
Left hand side of Parseval's theorem

lhs =

   3.0408e+05 + 2.0650e+03i

right hand side of Parseval's theorem

rhs =

   3.0408e+05 + 2.0650e+03i
```

6. Familiarization of DSP Hardware

AIM

To familiarize DSP hardware, Code Composer Studio and testing of C program for controlling LEDs through switches.

DEVICES AND SOFTWARE NEEDED

VSK TMS 320C 6748 KIT, CCS 3.3

KIT FAMILIARIZATION

The VSK-6747 is a standalone development platform that enables users to evaluate and develop applications for the DSP processor.

Hardware Overview

The VSK comes with a full complement of on board devices that suit a wide variety of Application environments.

Key features include:

- A Texas Instruments TMS320C6747 device with a DSP floating point processor and processor operating up to
- 300 MHz
- 64 Megabytes SDRAM
- SPI Boot EEPROM
- TLV320AIC3106 Stereo Codec
- RS-232 Interface
- On chip real time clock
- Configurable boot load options
- 8 user LEDs/8 position user DIP switch
- Single voltage power supply (+5V)
- Expansion connectors for SPI/GPIO termination.
- Embedded JTAG Emulation
- 14 Pin TI JTAG/20 Pin JTAG Interfaces

VSK – 6748 Module

Fig: 1- VSK-6748 Module

Functional Overview of the VSK-6748

The TMS320C6748 on the VSK interfaces to on-board peripherals through the 16- bit wide multiplexed EMIF interface pins. The SDRAM memory is connected to its own dedicated 16 bit wide bus.

An on-board AIC3106 codec allows the DSP to transmit and receive analog audio Signals. The I2C bus is used for the codec control interface, while the McASP controls the audio stream. Signal interfacing is done through 3.5mm audio jacks that correspond to microphone input, headphone output, line input, and line output. The VSK includes 8 user LEDs, a 8 position user DIP switch, and on chip real time clock.

An included +5V external power supply is used to power the board. On-board switching voltage regulators provide the CPU core voltage, +3.3V, +1.8V for peripheral interfacing. The board is held in reset by the on-board power controller until these supplies are within operating specifications. Code Composer Studio communicates with the VSK through an embedded emulator or via the TI 14 pin JTAG connectors.

Basic Operation

The VSK is designed to work with TI’s Code Composer Studio IDE. Code Composer communicates with the board through an on board JTAG emulator. To start, follow the instructions in the Quick Start Guide to install Code Composer. This process will install all of the necessary development tools, documentation and drivers.

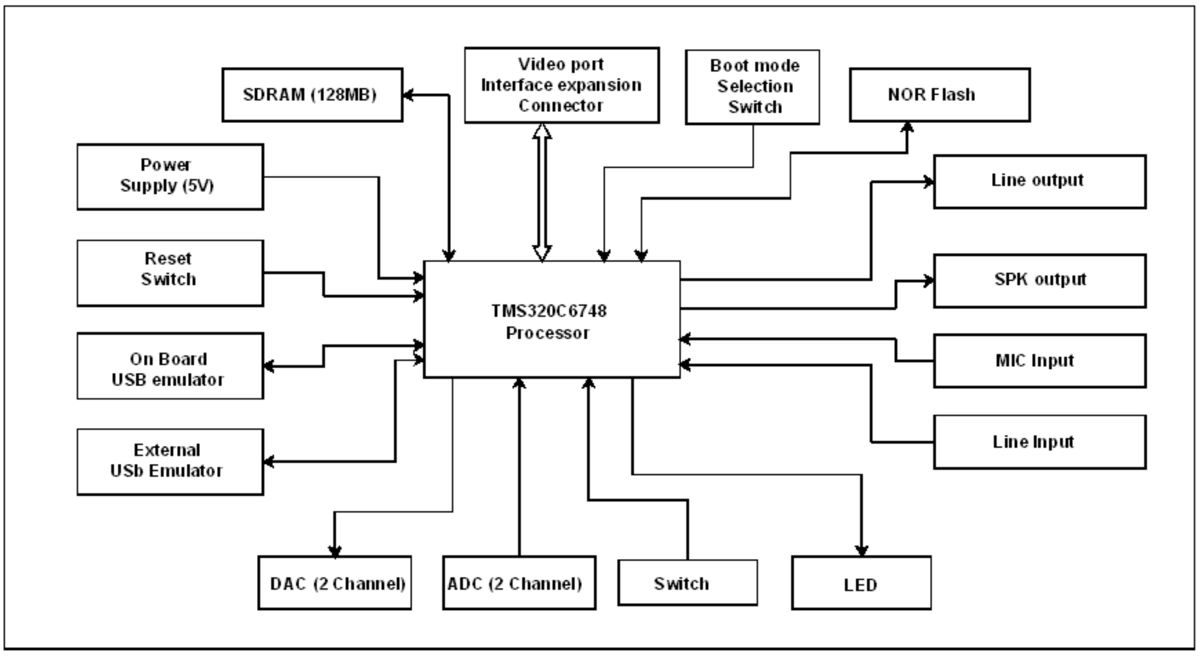


Fig 2 : Block of VSK - 6748

Software

Code Composer Studio ™ software is an integrated development environment (IDE) that supports TI's microcontroller (MCU) and embedded processor portfolios. Code Composer Studio software comprises a suite of tools used to develop and debug embedded applications. The software includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler and many other features. The intuitive IDE provides a single-user interface that takes you through each step of the application development flow. Familiar tools and interfaces let you get started faster than ever before. Code Composer Studio software combines the advantages of the Eclipse software framework with advanced embedded-debug capabilities from TI resulting in a compelling feature-rich development environment for embedded developers.

Result

7. Convolution: Linear & Circular

AIM

To find linear & circular convolution of two sequences in DSP kit (Using VSK-6748 KIT) and using MATLAB

TOOLS USED

MATLAB, Code composer Studio

THEORY

In mathematical terms, given two finite, discrete-time signals x[n]$x[n]$ and h[n]$h[n]$, both of length NN, and their DFTs

Linear Convolution Definition

$$x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m], \quad n \in \mathbb{Z}.$$

Circular Convolution

Here, $\circledast$ symbol denotes the circular convolution. It is defined as

$$x[n] \circledast h[n] = \sum_{m=0}^{N-1} x[m]h[(n-m)\%N],$$

where % denotes the modulo operation,

Linear convolution - Program

C Program:
```
#include<fastmath67x.h>
void Vi_DSKC6747_init( );
void main()
{
int *Xn,*Hn,*Yout; // Input ,Output Sequence
int *xnlen,*hnlen; // Sequence Length
int i,k,n,l,m;
Xn=(int *)0x80000100;
Hn=(int *)0x80000200;
xnlen = (int *)0x80000300;
hnlen = (int *)0x80000304;
Yout=(int *)0x80000400;
l = *xnlen;
m = *hnlen;
Vi_DSKC6747Rev1_init( ); // Board Initialization
for(i=0;i<50;i++) // Memory Clear
{
Yout[i]=0;
Xn[l+i]=0;
Hn[m+i]=0;
xnlen[2+i]=0;
}
for(n=0;n<(l+m-1);n++)
{
for(k=0;k<=n;k++)
{
Yout[n]=Yout[n]+(Xn[k]*Hn[n-k]);
}
}
}
```

Input & Output:

Sequence Memory Address Values

Sequence	Memory Address	Values
Xn - First input	0x80000100	1,2,1,1
Hn - Second input	0x80000200	1,1,1
xnlen – 1st Seq length	0x80000300	4
hnlen – 2nd Seq length	0x80000304	3
Yout – Output Seq	0x80000400	1,3,4,4,2,1

Circular convolution – Program

```
#include<fastmath67x.h>

void main()

{

int Xn[4]= {1,2,3,4};      // x(n) input

int Hn[4] = {1,1,2,2};   // h(n) input

int xn = 4;              // length of x(n)

int hn= 4;                                          // length of h(n)

int *Yn;                 // output array

int i,n,m,l;

Yn=(int *)0x80010000;   // o/p starting address

for(i=0;i<xn+hn;i++)    // memory clear

            Yn[i]=0;

for(n=0;n<xn;n++)

            {

            for(m=0;m<hn;m++)
```

```
                {
                l=n-m;
                        if(l<0)
                        {
                        l=l+xn;
                        }
                Yn[n]=Yn[n]+(Xn[m]*Hn[l]);    // circular convolution
                }
        }
}
```

OUTPUT

Input & Output:

Sequence Memory Address Values
Xn – [1 2 3 4]
Hn – [1 1 2 2]
xnlen – 4
hnlen – 4
Yout – Output Seq 0x80010000 15 17 15 13

MATLAB :

```
x= input('ENTER THE FIRST SEQUENCE');
subplot(3,1,1); stem(x);
title('FIRST SEQUENCE');
ylabel('x(n)');
xlabel('n');
h= input('ENTER THE SECOND SEQUENCE');
subplot(3,1,2); stem(h);
title('SECOND SEQUENCE');
ylabel('h(n)');
xlabel('n'); N1=length(x); N2=length(h); N= max(N1,N2);
x=[x zeros(1,N-N1)];
h=[h zeros(1,N-N2)];
for n=0:N-1
    y(n+1)=0;
    for i=0:N-1
        j=mod(n-i,N);
        y(n+1)=y(n+1)+x(i+1)*h(j+1);
    end
end
disp('OUTPUT SEQUENCE IS');
disp(y); subplot(3,1,3); stem(y);
```

```
title('OUTPUT SEQUENCE')
ylabel('y(n)');
xlabel('n');
```

OUTPUT

```
ENTER THE FIRST SEQUENCE[1 2 3 4]

ENTER THE SECOND SEQUENCE[1 1 2 2]

OUTPUT SEQUENCE IS

   15    17    15    13
```

Result

8. Overlap-add block convolution

AIM

Implement the overlap add block convolution method in MATLAB for N=10, 100, 1000

TOOLS USED

MATLAB

THEORY

Steps to follow in overlap add method

1. Break the input signal x(n) into non-overlapping blocks $x_m(n)$ of length L.

2. Zero pad h(n) to be of length N = L + M − 1.

3. Take N-DFT of h(n) to give H(k), k = 0, 1, . . . , N − 1.

4. For each block m:

   4.1 Zero pad $x_m(n)$ to be of length N = L + M − 1.

   4.2 Take N-DFT of $x_m(n)$ to give $X_m(k)$, k = 0, 1, . . . , N − 1.

   4.3 Multiply: $Y_m(k) = X_m(k) \cdot H(k)$, k = 0, 1, . . . , N − 1.

   4.4 Take N-IDFT of $Y_m(k)$ to give $y_m(n)$, n = 0, 1, . . . , N − 1.

5. Form y(n) by overlapping the last M − 1 samples of $y_m(n)$ with the first M − 1 samples of $y_{m+1}(n)$ and adding the result.

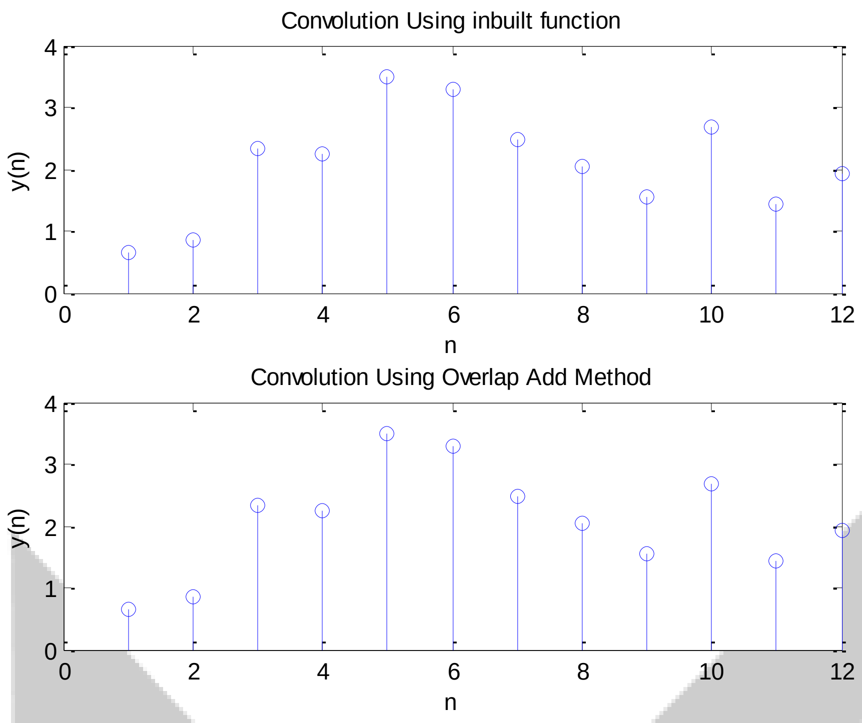
PROGRAM

```
% Overlap Add method for Linear Convolution
close All
clear All
clc
N=input('Enter the length of x(n) : ');
x=rand(1,N);          % Random N Numbers
h=input('Enter the values of h(n)=');
L=length(h);
N1=length(x);
M=length(h);
lc=conv(x,h);
x=[x zeros(1,mod(-N1,L))];
```

```
N2=length(x);
h=[h zeros(1,L-1)];
H=fft(h,L+M-1);
S=N2/L;
index=1:L;
X=[zeros(M-1)];
for stage=1:S
    xm=[x(index) zeros(1,M-1)];     % Selecting sequence to
process
    X1=fft(xm,L+M-1);
    Y=X1.*H;
    Y=ifft(Y);
    Z=X((length(X)-M+2):length(X))+Y(1:M-1);    %Samples Added
in every stage
    X=[X(1:(stage-1)*L) Z Y(M:M+L-1)];
    index=stage*L+1:(stage+1)*L;
end
i=1:N1+M-1;
X=X(i);
figure()
subplot(2,1,1)
stem(lc);
title('Convolution Using inbuilt function')
xlabel('n');
ylabel('y(n)');
subplot(2,1,2)
stem(X);
title('Convolution Using Overlap Add Method')
xlabel('n');
ylabel('y(n)');
```

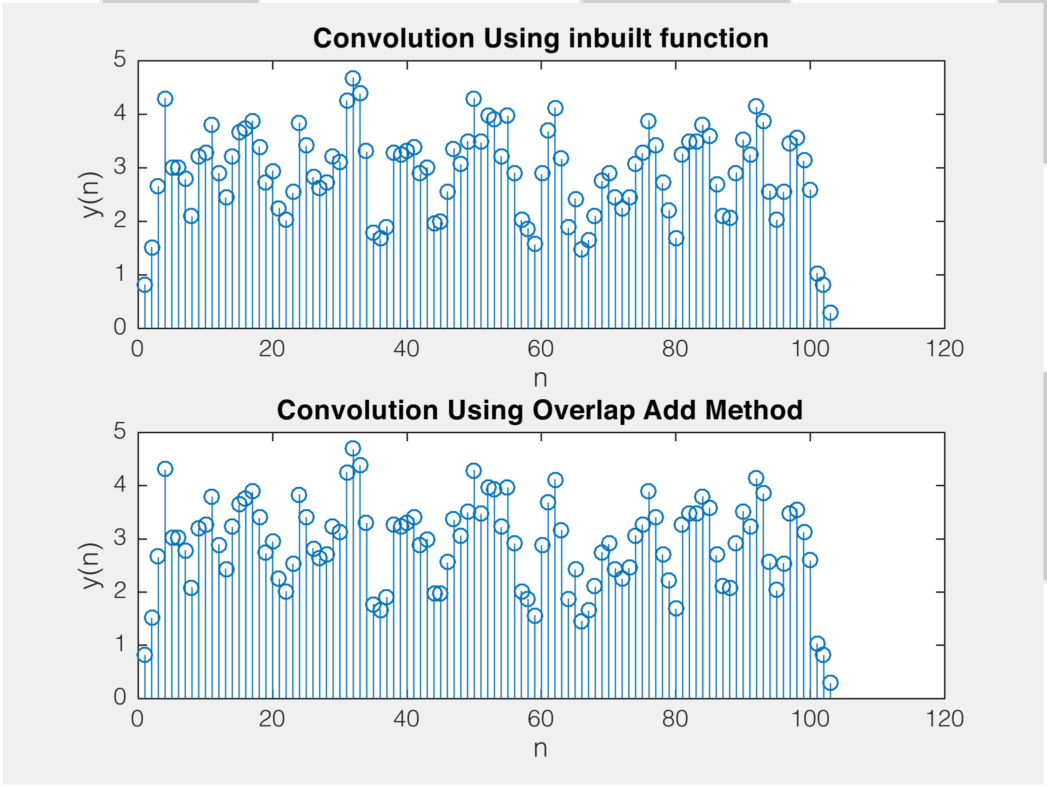
OUTPUT

```
Enter the length of x(n) : 10
Enter the values of h(n)=[1 1 2]
```



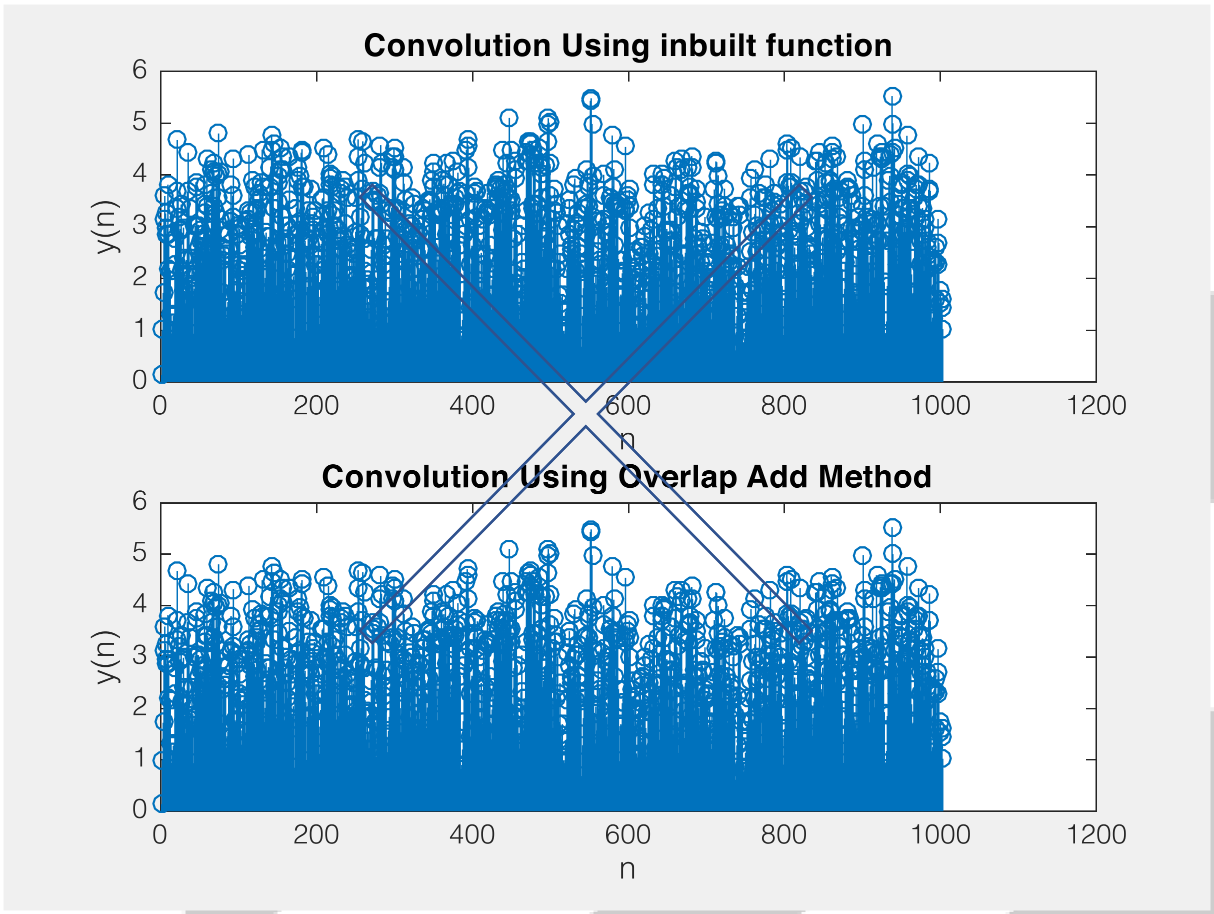
Enter the length of x(n) : 100

Enter the values of h(n)=[1 1 2 2]



Enter the length of x(n) : 1000

Enter the values of h(n)=[1 1 2 2]



RESULT

9. Overlap-save block convolution

AIM

Implement the overlap save block convolution method in MATLAB for N=10, 100, 1000

TOOLS USED

MATLAB,

THEORY

<u>Steps to follow in overlap save method</u>

1. Insert M − 1 zeros at the beginning of the input sequence x(n).

2. Break the padded input signal into overlapping blocks $x_m(n)$ of length N = L + M − 1 where the overlap length is M − 1.

3. Zero pad h(n) to be of length N = L + M − 1.

4. Take N-DFT of h(n) to give H(k), k = 0, 1, . . . , N − 1.

5. For each block m:

5.1 Take N-DFT of $x_m(n)$ to give $X_m(k)$, k = 0, 1, . . . , N − 1.

5.2 Multiply: $Y_m(k)$ = Xm(k) · H(k), k = 0, 1, . . . , N − 1.

5.3 Take N-IDFT of $Y_m(k)$ to give $y_m(n)$, n = 0, 1, . . . , N − 1.

5.4 Discard the first M − 1 points of each output block ym(n).

6. Form y(n) by appending the remaining (i.e., last) L samples of each block ym(n).

PROGRAM

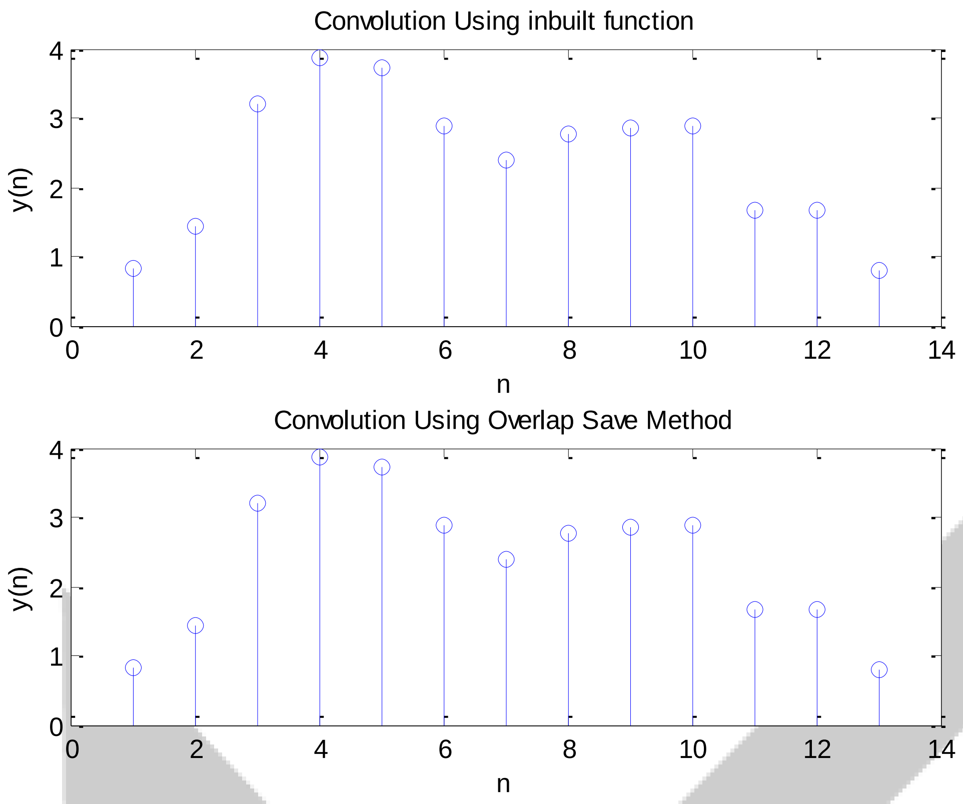
```
% Overlap Save method for Linear Convolution
close All
close All
clear All
clc
N=input('Enter the length of x(n) : ');
x=rand(1,N);          % Random N Numbers
h=input('Enter the values of h(n)=');
L=length(h);
N1=length(x);
M=length(h);
lc=conv(x,h);
x=[x zeros(1,mod(-N1,L)) zeros(1,L)];
N2=length(x);
```

```
h=[h zeros(1,L-1)];
H=fft(h,L+M-1);
S=N2/L;
index=1:L;
xm=x(index);          % For first stage Special Case
x1=[zeros(1,M-1) xm];    %zeros appeded at Start point
X=[];
for stage=1:S
    X1=fft(x1,L+M-1);
    Y=X1.*H;
    Y=ifft(Y);
    index2=M:M+L-1;
    Y=Y(index2);          %Discarding Samples
    X=[X Y];
    index3=(((stage)*L)-M+2):((stage+1)*L);      % Selecting
Sequence to process
    if(index3(L+M-1)<=N2)
    x1=x(index3);
    end
end;
i=1:N1+M-1;
X=X(i);
figure()
subplot(2,1,1)
stem(lc);
title('Convolution Using inbuilt function')
xlabel('n');
ylabel('y(n)');
subplot(2,1,2)
stem(X);
title('Convolution Using Overlap Save Method')
xlabel('n');
ylabel('y(n)');
```

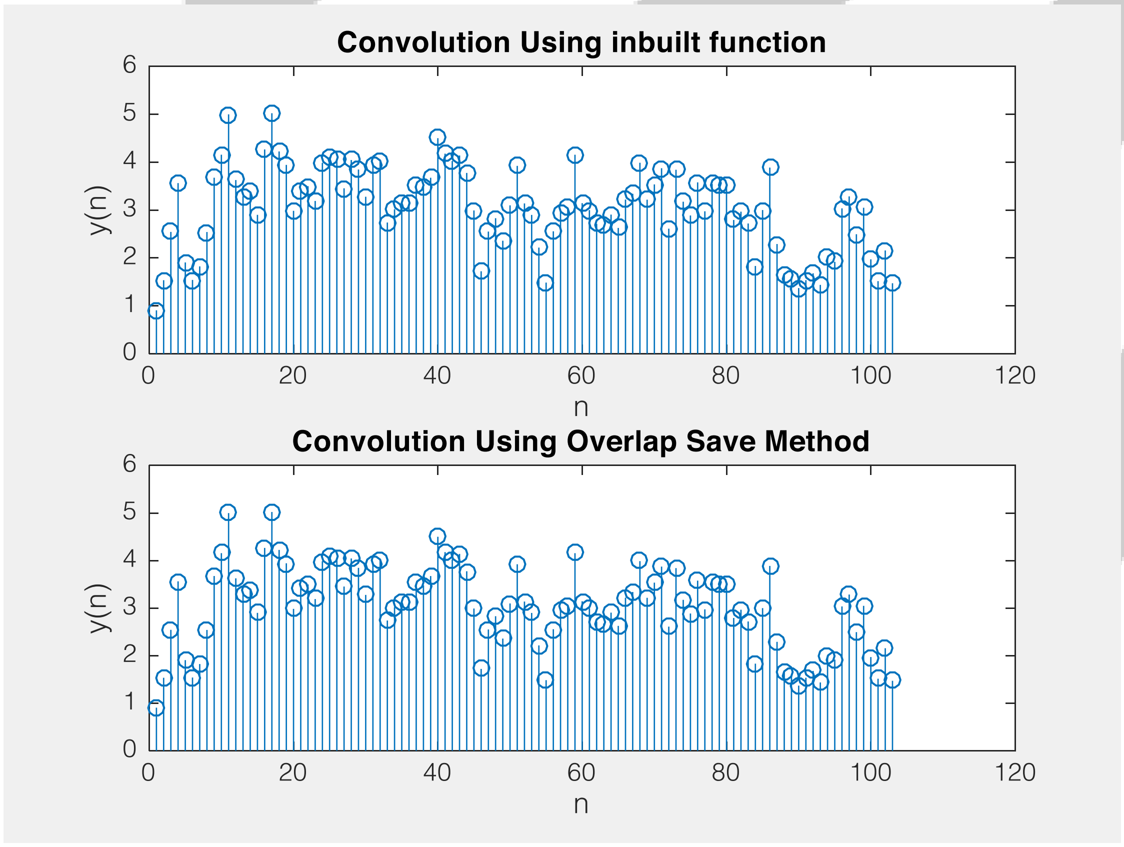
OUTPUT

Enter the length of x(n) : 10

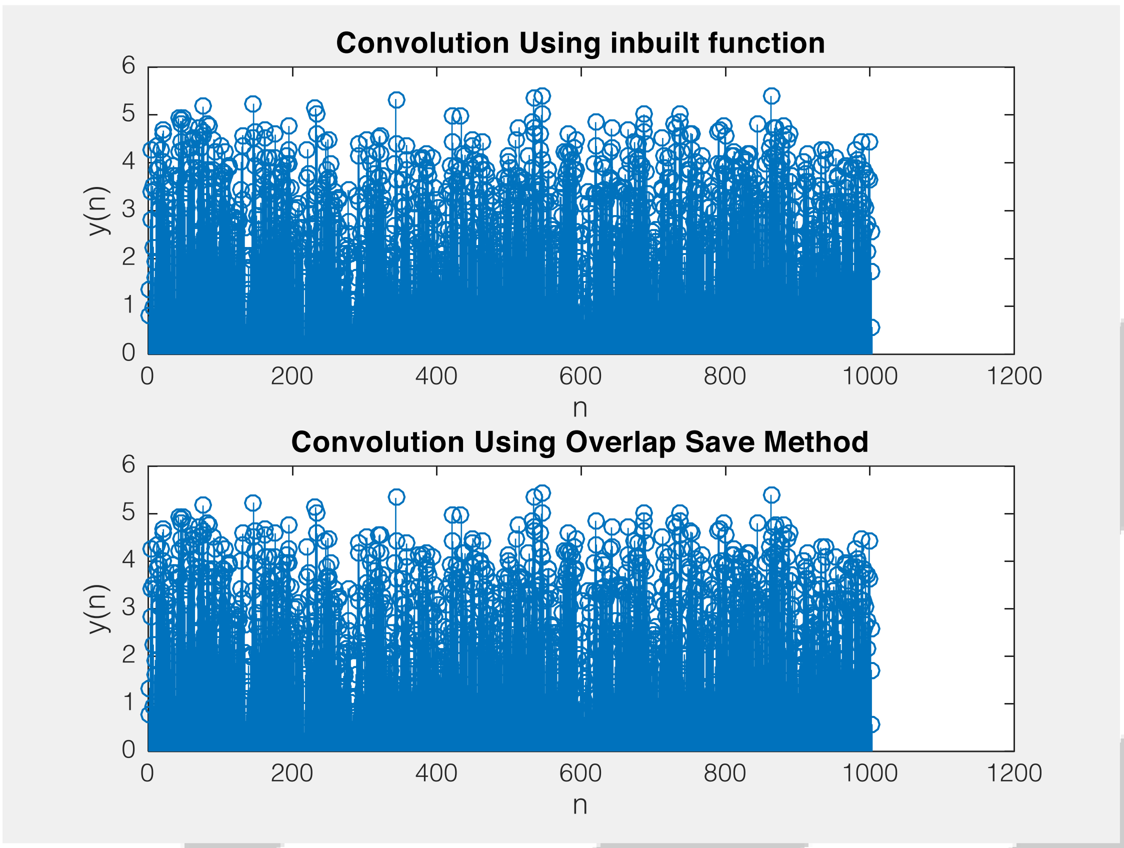
Enter the values of h(n)=[1 1 2 2]



```
Enter the length of x(n) : 100

Enter the values of h(n)=[1 1 2 2]
```



```
Enter the length of x(n) : 1000

Enter the values of h(n)=[1 1 2 2]
```



RESULT

10.FFT of signal

AIM

To find FFT of a given signal in DSP kit (Using VSK-6748 KIT) and using MATLAB

TOOLS USED

MATLAB, Code composer Studio, signal generator,

THEORY

The "Fast Fourier Transform" (FFT) is an important measurement method in the science of audio and acoustics measurement. It converts a signal into individual spectral components and thereby provides frequency information about the signal. FFTs are used for fault analysis, quality control, and condition monitoring of machines or systems.

C -Program

```
#include<fastmath67x.h>
#include<math.h>
#include<stdio.h>
#define pi 3.14159
void main()
{
float *XinSeq;
int N = 8;
float *Xreal,*Ximag;
float Wr,Wi;
int m,count,s,bsep,bwidth,i;
int r,p,j,topval,botval,a,t,l,n,remain;
float theta;// angle //
float t1,t2,t3,t4,t5,t6;
        // INPUT //
XinSeq=(float *)0x80010000;
        // OUTPUT //
```

```
Xreal=(float *)0x80011000;
Ximag=(float *)0x80012000;
        // MEMORY CLEAR //
for(i=0;i<N;i++)
        {
                XinSeq[i+N]=0;
                Xreal[i]=0;
                Ximag[i]=0;
        }
        // STAGES  //
count=N;
m=0;
while(count>1)
        {
                count=count/2;
                m=m+1; // m= no of stages //
        }
//  BIT REVERSAL //
for(n=0;n<N;n++)
{
        a=0;
        t=N/2;
        l=n;
        for(i=1;i<=m;i++)
        {
        remain=l%2;
        a=a+(remain*t);
        t=t/2;
        l=l/2;
```

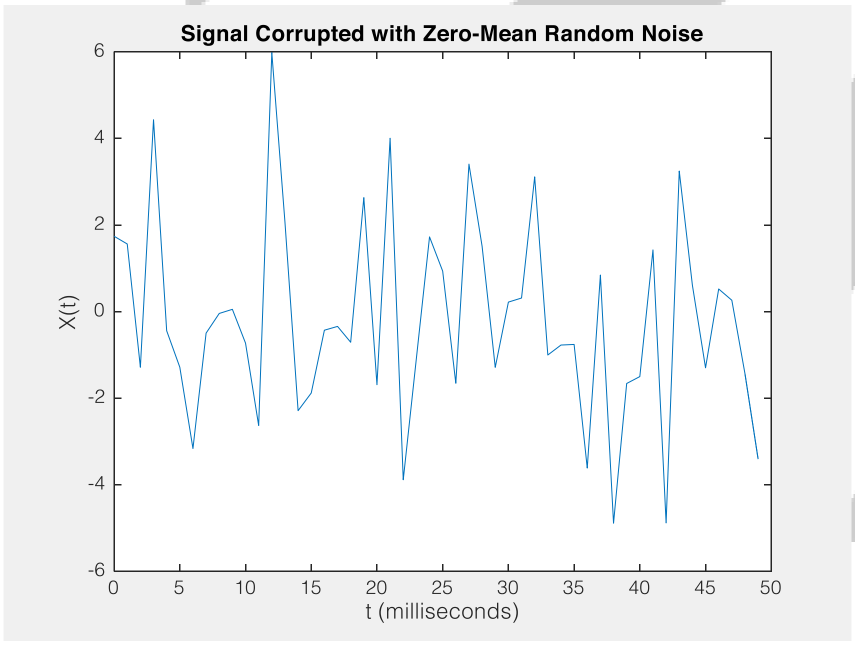
```
            }
Xreal[n]=XinSeq[a];
}
            //  calculation of DFT USING FFT //
for(s=1;s<=m;s++)
            {
            bsep=pow(2,s);//  distance between butterfullies //
            p=N/bsep;
            bwidth=bsep/2;           // distance between two inputs //
                        for(j=0;j<bwidth;j++)
                        {
                        r=p*j;
                        theta=((2*pi*r)/N);
                        Wr=cos(theta);
                        Wi=-(sin(theta));
                                    for(topval=j;topval<=N;topval+=bsep)
                                    {
                                    botval=topval+bwidth;
                                    t1=Xreal[topval];
                                    t2=Ximag[topval];
                                    t3=Xreal[botval]*Wr;
                                    t4=Ximag[botval]*Wi;
                                    t5=Xreal[botval]*Wi;
                                    t6=Ximag[botval]*Wr;    // multiply with twidel factor
                                    Xreal[topval]=t1+t3-t4; // real o/op
                                    Ximag[topval]=t2+t5+t6;   // imaginary o/p
                                    Xreal[botval]=t1-t3+t4;
                                    Ximag[botval]=t2-t5-t6;
                           }
```

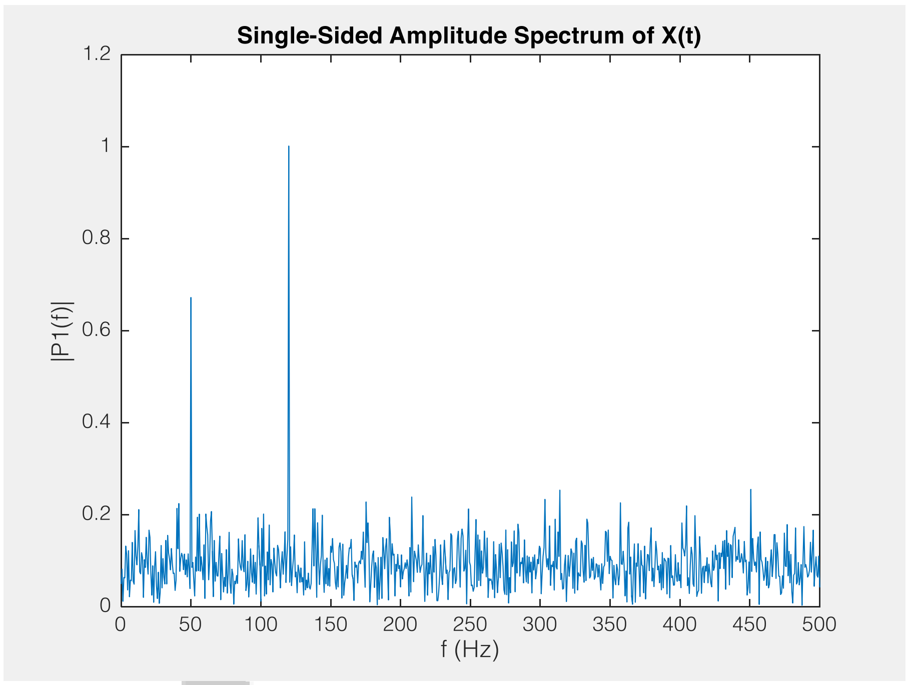
```
                }

        }

}
```

MATLAB Program

```
Fs = 1000;                % Sampling frequency
T = 1/Fs;                 % Sampling period
L = 1500;                 % Length of signal
t = (0:L-1)*T;            % Time vector
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
X = S + 2*randn(size(t));
plot(1000*t(1:50),X(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('t (milliseconds)')
ylabel('X(t)')
Y = fft(X);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(L/2))/L;
figure
plot(f,P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```

Output




Single-Sided Amplitude Spectrum of X(t)
|P1(f)|
f (Hz)
0
0.2
0.4
0.6
0.8
1
1.2
0
50
100
150
200
250
300
350
400
450
500

Result