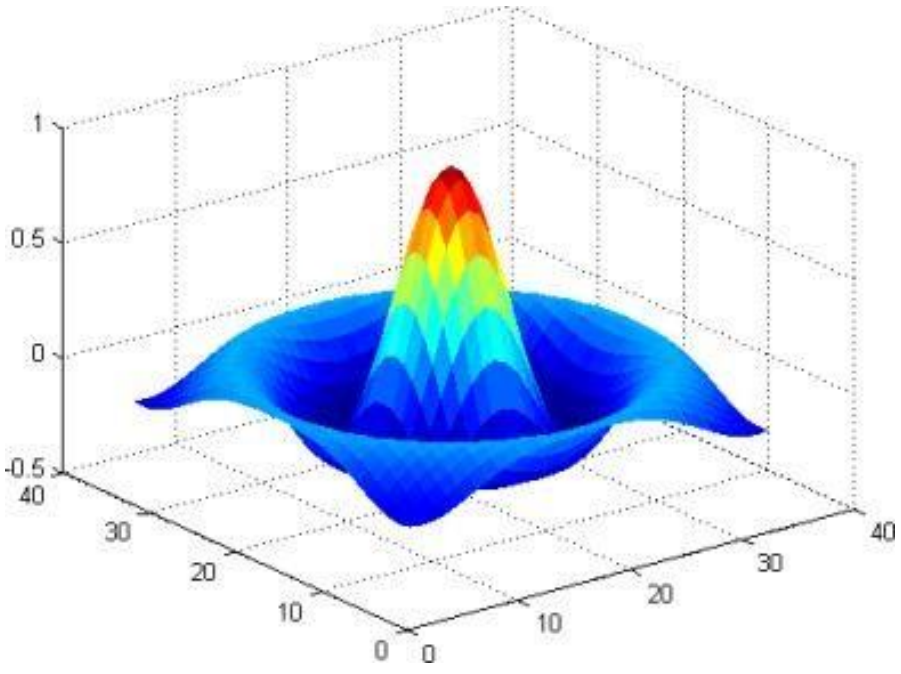
EC333 Digital Signal Processing Lab

LABORATORY MANUAL



VISION

To nurture the talents of electronics and communication engineers, making them highly competent for growth of the society.

MISSION

- To deliver excellence in teaching - learning process.
- Promote safe, orderly, caring and supportive environment to learners.
- Development of skilled engineers to perform innovative Research for betterment of the society.
- To encourage industry - institute interaction, career advancement, innovation and entrepreneurship development.

PROGRAM EDUCATIONAL OUTCOME (PEO)

PEO1: To acquire a strong foundation in mathematics and scientific fundamentals, to develop an ability to analyze various functional elements of different disciplines of electronics and communication engineering.

PEO2: Develop technical competence to move in pace with rapid changes in technology.

PEO3: Equip learners to strengthen knowledge and soft skills for carrier advancement.

PEO4: Adhere to ethics to contribute for betterment of the society.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1. To understand principles and applications of various electronic components/devices and circuits.

PSO2. Enable learners to solve complex problems using modern hardware and software tools.

COURSE CODE	COURSE NAME	L-T-P-C	YEAR OF INTRODUCTION
EC333	Digital Signal Processing Lab	0-0-3-1	2015

Prerequisite:
EC 213 Electronics Design Automation Lab, EC 202 Signals & Systems

Course objectives:

- Enable the students to explore the concepts of design, simulation and implementation of various systems using MATLAB/SciLab/OCTAVE and DSP kit.

List of Experiments:

Part A: Experiments on Digital Signal Processor/ DSP kits: (All experiments are mandatory)

1. Generation of sine wave and standard test signals.
2. Convolution : Linear and Circular
3. Real Time FIR Filter implementation (Low-pass, High-pass and Band-pass) by inputting a signal from the signal generator.
4. Real Time IIR Filter implementation (Low-pass, High-pass and Band-pass) by inputting a signal from the signal generator.
5. Sampling of analog signal and study of aliasing.

Part B: Experiments based on MATLAB/SciLab/OCTAVE (7 experiments are mandatory)

1. Generation of Waveforms (Continuous and Discrete)
2. Verification of Sampling Theorem.
3. Time and Frequency Response of LTI systems (First and second order).
4. Linear Convolution, Circular Convolution and Linear Convolution using Circular Convolution.
5. To find the DFT and IDFT for the given input sequence.
6. Linear convolution using DFT (Overlap-add and Overlap-Save methods).
7. To find the DCT and IDCT for the given input sequence.
8. To find FFT and IFFT for the given input sequence.
9. FIR and IIR filter design using Filter Design Toolbox.
10. FIR Filter (Low-pass, High-pass and Band-pass) design (Window method).
11. IIR Filter (Low-pass, High-pass and Band-pass) design (Butterworth and Chebychev).
12. Generation of AM, FM & PWM waveforms and their spectrum.
13. Generation of DTMF signal.
14. Study of sampling rate conversion (Decimation, Interpolation, Rational factor).
15. Filtering of noisy signals
16. Implementation of simple algorithms in audio processing (delay, reverb, flange etc.).
17. Implementation of simple algorithms in image processing (detection, de-noising, filtering etc.)

Expected outcome:

The student should able to: Design, Simulate and realize various systems related to DSP.

COURSE OUTCOME

C308.1	Students will be able to **Simulate** various waveform generations
C308.2	Students will be able to **Simulate and realize** DFT and IDFT
C308.3	Students will be able to **simulate and realize** IIR Filters
C308.4	Students will be able to **simulate and realize** FIR Filters

CO-PO MAPPING

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
C308.1	1	2	2	1	-	2	-	-	-	-	-	-	2	-
C308.2	2	2	2	2	-	1	-	-	-	-	-	-	1	-
C308.3	3	2	2	3	-	1	-	-	-	-	-	-	2	2
C308.4	3	2	2	2	-	-	-	-	-	-	-	-	3	-
C308.1	2.25	2	2	2	-	1	-	-	-	-	-	-	2	0.5

LIST OF EXPERIMENTS

Cycle 1: Experiments based on MATLAB	
1.	Waveform generation
2.	DFT and IDFT
3.	Convolution and Deconvolution
4.	Convolution and Deconvolution using DFT
5.	FIR filter design using Hanning , Hamming, Kaiser window
6.	Butterworth Filter
7.	Chebychev filter
Cycle 2: Experiments based on digital Signal Processor/DSP kit	
1.	Generation of Sine wave
2.	Convolution
3.	Real time filter Implementation

INDEX

PART A : Experiments on DSP Kit

PART B : Experiments based on MATLAB

TMS320C50 Architecture Overview

1. INTRODUCTION

It is needless to say that in order to utilize the full feature of the DSP chipTMS320C50, the DSP engineer must have a complete knowledge of the DSP device. This chapter is an introduction to the hardware aspects of the TMS320C50. The important units of TMS320C50 are discussed.

2. THE DSP CHIP TMS320C50

The TMS320C50 is a 16-bit fixed point digital signal processor that combines the flexibility of a high speed controller with the numerical capability of an array processor, thereby offering an inexpensive alternative to multichip bit-slice processors. The highly paralleled architecture and efficient instruction set provide speed and flexibility capable of executing 10 MIPS (Million Instructions per Second). The TMS320C50 optimizes speed by implementing functions in hardware that other processors implement through microcode or software. This hardware intensive approach provides the design engineer with processing power previously unavailable on a single chip. The TMS320C50 is the third generation digit l signal processor in the TMS320 family. Its powerful instruction set, inherent flexibility, high speed number crunching capabilities, and innovative architecture have made this high-performance, cost-effective processor the ideal solution to many telecommunications, computer, commercial, industrial, and military applications.

3. KEY FEATURES OF TMS320C50

The key features of the Digital Signal Processor TMS320C50 are:

- 35-/50-ns single-cycle fixed-point instruction execution time (28.6/20 MIPS)
- Upward source-code compatible with all `C1X and `C2x devices
- RAM-based memory operation (`C50)
- 9K x 16-bit single-cycle on-chip program/data RAM (`C50)
- 2K x 16-bit single-cycle on-chip boot ROM (`C50)
- 1056 x 16-bit dual-access on-chip data RAM
- 224K x 16-bit maximum addressable external memory space (64K program, 64Kdata, 64K I/O, and 32K global)

- 32-bit arithmetic logic unit (ALU), 32-bit accumulator (ACC), and 32-bitaccumulator buffer (ACCB)
- 16-bit parallel logic unit (PLU)
- 16 x 16-bit parallel multiplier with a 32-bit product capability.
- Single-cycle multiply/accumulate instructions
- Eight auxiliary registers with a dedicated auxiliary register arithmetic unit for indirect addressing.
- Eleven context-switch registers (shadow registers) for storing strategic CPU controlled registers during an interrupt service routine
- Eight-level hardware stack
- 0- to 16-bit left and right data barrel-shifters and a 64-bit incremental data shifter
- Two indirectly addressed circular buffers for circular addressing
- Single-instruction repeat and block repeat operations for program code
- Block memory move instructions for better program/data management
- Full-duplex synchronous serial port for direct communication between the `C5x and another serial device
- Time-division multiple-access (TDM) serial port
- Interval timer with period, control, and counter registers for software reset
- 64K parallel I/O ports, 16 of which are memory mapped
- Sixteen software programmable wait memory spaces.

4. ARCHITECTURE

A detailed architectural block diagram of TMS320C50 is illustrated in figure 1-1. The TMS320C50 utilizes a modified Harvard architecture for speed and flexibility. In a strict Harvard architecture, program and data memory are in two separate spaces, permitting a full TMS320 family's modification of the Harvard program and data spaces, thereby increasing the flexibility modification permits coefficients stored in program memory to be read into RAM, eliminating the need for a separate coefficient ROM. It also makes available immediate instructions and subroutines based on computed values

.

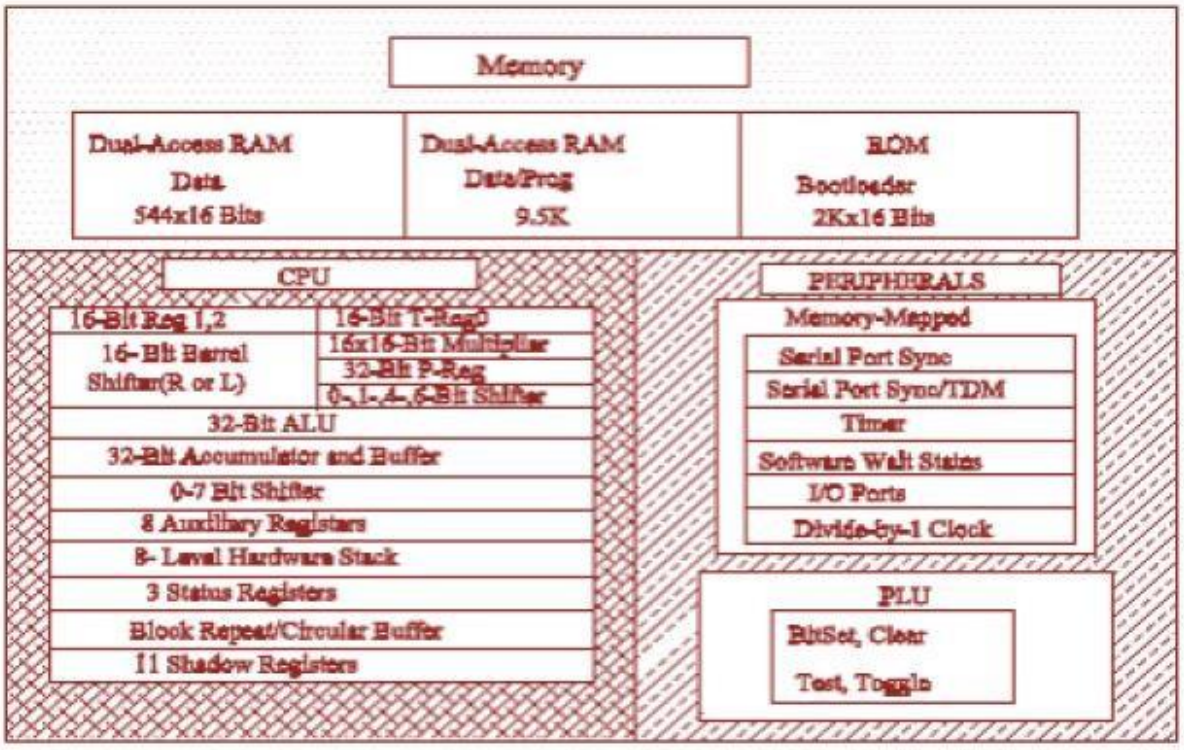


Figure 1.1 TMS320C50 Block Diagram

32-Bit Accumulator:

The TMS320C50 contains a 32 double-precision, two's complement arithmetic. The ALU is a general purpose arithmetic unit that operates on 16 from immediate instructions. In addition to the usual can perform Boolean operations, pro high-speed controller. The accumulator stores the output from the ALU and an input to the ALU. Its word length is 32 high order word (bits 31 through 16) and a provided for storing and loading the high and lower order accumulator words to memory.

16 x 16-Bit Parallel Multiplier

The multiplier performs a 16 x 16 32-bit result in a single instruction cycle. The multiplier consists of three units: the T-Register, P-Register, and multiplier array. The 16-bit T-Register temporarily stores the multiplicand and the P-Register stores the 32-bit product. Multiplier values either come from the data memory or are derived immediately from the MPY (multiply immediate) instruction word. The fast on-chip multiplier allows the device to perform fundamental operations such as convolution, correlation, and filtering. Two multiply/accumulate instructions in the instruction set fully utilize the computational bandwidth of the multiplier, allowing both operands to be processed simultaneously.

Shifters

A 16-bit scaling shifter is available at the accumulator input. This shifter produces a left shift of 0 to 16-bits on the input data to accumulator. TMS320C50 also contains a shifter at the accumulator output. This shifter provides a left shift of 0 to 7, on the data from either the ACCH or ACCL register, right, before transferring the product to accumulator.

Data and Program Memory

Since the TMS320C50 uses Harvard architecture, data and program memory reside in two separate spaces. Additionally TMS320C50 has one more memory space called I/O memory space. The total memory capacity of TMS320C50 is 64KW each of Program, Data and I/O memory. The 64KW of data memory is divided into 512 pages with each page containing 128 words. Only one page can be active at a time. One data page selection is done by setting data page pointer. TMS320C50 has 1056 words of dual access on chip data RAM and 9K words of single access Data/Program RAM. The 1056 words of on chip data memory is divided as three blocks B0, B1 &B2, of which B0 can be configured as program or data RAM. Out of the 64KW of total program memory, TMS320C50 has 2K words of on-chip program ROM. The TMS320C50 offers two modes of operation defined by the state of the MC/MP pin: the microcomputer mode (MC/MP = 1) or the microprocessor mode (MC/MP = 0). In the microcomputer mode, on-chip ROM is mapped into the memory space with upto 2K words of memory available. In the microprocessor mode all 64K words of program memory are external.

Interrupts and Subroutines:

The TMS320C50 has three external maskable user interrupts available for external devices that interrupt the processor. The TMS320C50 contains an eight-level hardware stack for saving the contents of the program counter during interrupts and subroutine calls. Instructions are available for saving the device's complete context. PUSH and POP instructions permit a level of nesting restricted only by the amount of available RAM.

Serial Port:

A full-duplex on-chip serial port provides direct communication with serial devices such as codecs, serial A/D converters and other serial systems. The interface signals are compatible with codecs and many others serial devices with a minimum of external hardware.

Input and Output:

The 16-bit parallel data bus can be utilized to perform I/O functions in twocycles. The I/O ports are addressed by the four LSBs on the address lines, allowing 16 input and 16 output ports. In addition, polling input for bit test and jump operations (BIO) and three interrupt pins (INT0 - INT2) have been incorporated for multitasking.

SOFTWARE OVERVIEW

This section illustrates the use of program and execution mainly in the **standalone mode**. The Micro-50 EB has 3 software development tools namely

1. Standalone Mode
2. Monitor program
3. Serial Mode

In "Standalone Mode" the **Micro-50 EB** works with a 104 keys keyboard and16x2 LCD display and line assembler. With this configuration, the student can enter his program through the keyboard and edit and display it on the LCD display. The user can enter the Mnemonics using the Line Assembler, and debug the program to run it on **Micro-50 EB.**"Monitor Program" is used to enter data directly into Data or Program memory, display the data etc. It has several commands to enter the user program, for editing and debugging. In "Serial Mode", it works with an IBM PC computer and program entry and debugging is done at the PC level.

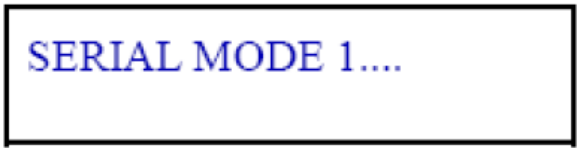
PROGRAM & EXECUTION:

1. Serial Monitor Mode:

Connect the serial monitor cable from Serial port of Micro-50 EB kit to the serial port COM1 or COM2 of PC XT/AT (prefer COM1 for default selection). Now execute communication software (XTALK.EXE) in PC. Power on the Micro-50 EB Kit with all its set up ready and enter the followingcommand at the prompt.

```
Micro-50EB Trainer
#SM
```

Press the enter key to enter into serial monitor mode and the screen displays the following Message



Now the following menu will appear on the monitor.

Micro-50 EB Serial Monitor, Ver.1.0
(C) Copyright 1996 by Vi Microsystems (P) Ltd. Chennai.
#

Now enter "HE" to view the help menu of the serial monitor. To assemble the program given in the example enter "AS" at the prompt and press the Enter Key, the screen displays the following message.

#Micro-50 EB Line Assembler, Version 2.0
Enter Address:

Now enter the program memory starting address "C000H" and press Enter Key. Now the screen display next consequent message as

C000H

Enter the mnemonics of the program sequentially viewing opcodes of the respective mnemonics after pressing enter key. On completion of assembling enter dot (.) and press the enter key to come out to prompt.

To execute the program use the command "GO C000" and press the Enter Key, where C000 is program memory starting address. To abort the execution, press the reset switch once to reset the Micro-50 Kit. Now enter "SM" in Micro-50 EB trainer to re-enter into serial mode. To verify the execution, dump the data memory from 8000H to 9000H using "DD" command. This operation is same as Line assembler in standalone mode.

Note:

1. At any occasion to abort the serial monitor from the execution of theprogram press the reset switch of Micro-50 EB kit.
2. To quit of serial monitor mode enter "QU" at the prompt and press enterkey.

Exp No 1

Generation of Sine Wave & Standard Test Signals

AIM

To generate different wave forms by using TMS320C50 DSP processor.

APPARATUS REQUIRED

TMS320C50 DSP processor kit, PC, CRO with probe.

PROCEDURE

1. Start the program.
2. Load the amplitude of the input signal of 5 volts.
3. Load the frequency of the input signal.
4. Observe the waveform by using CRO.
5. Stop the program.

PROGRAM

a) Program for Generation of Sine Waveform

```
TXD         .SET        0H
STS         .SET        1H
DATA        .SET        2H
TEMP        .SET        3H
B3          .SET        0F000H
B2          .SET        0F00H
B1          .SET        00F0H
B0          .SET        000FH
      .MMREGS
      .TEXT
START:
      LDP    #100H
      LACC   #TABLE
      SACL   TEMP
REP1:
      LACC   #TABLE
      SACL   TEMP
      LAR    AR0, #372
REP:
      LACC   TEMP
```

```
        TBLR    DATA
  OUT DATA, 04H
  LACC    TEMP
        ADD     #1H
        SACL    TEMP
        MAR     *, AR0
        BANZ    REP,*-
        B    REP1
HLT:    B       HLT
```

b) Program for Generation of Square Waveform

```
                .MMREGS
                .TEXT
START:          LDP     #100H
                LACC #0FFFH ; change this value for amplitude.
LOOP:           SACL    0
                RPT     #0FFH    ; change this value for frequency.
                OUT     0,04H    ; address for DAC.
                CMPL
                B       LOOP
                .END
```

c) Program for Generation of Sawtooth Waveform

```
                .MMREGS
                 .TEXT
START:          LDP     #120H
                LACC    #0H     ; change lower amplitude
                SACL    0
LOOP:           LACC    0
                OUT     0,04H
                ADD     #05h    ; change frequency
                SACL    0
                SUB     #0FFFh ; change upper amplitude
                BCND    LOOP, LEQ
                B       START
                .END
```

d) Program for Generation of Triangular Waveform

```
AMPLITUDE .SET 4
FREQ      .SET 350
TEMP    .SET    0
;
                .MMREGS
                .TEXT
START:          LDP     #100H
```

```
        SPLK    #0, TEMP
CONT1:  LAR     AR2, #FREQ
CONT:   OUT     TEMP, 4
        LACC    TEMP
        ADD     #AMPLITUDE
        SACL    TEMP
        MAR     *, AR2
        BANZ    CONT,*-
        LAR     AR2, #FREQ
CONTx:  OUT     TEMP, 4
        LACC    TEMP
        SUB     #AMPLITUDE
        SACL    TEMP
        MAR     *, AR2
        BANZ    CONTx
        B       CONT1
```

OBSERVATION

S.No	Name of Waveforms	Time Period (msec)	Amplitude(Volts)
1.	Sine wave		
2.	Square Wave		
3	Saw tooth Wave		
4	Triangular Wave		

RESULT

Thus the Sine, Square, Saw tooth and Triangular Wave was generated using TMS320C50 DSP Processor.

Exp No 2

Convolution: Linear & Circular

AIM

To perform the linear and circular convolution of the two sequences using TMS320C50 DSP Processor.

ALGORITHM

1. Get the two sequence x(n) and h(n) in matrix form.
2. The convolution of the two sequences is given by

$$y(n) = x(n) * h(n) = \sum_{k=-\alpha}^{\alpha} x(k)h(n-k)$$ For Linear Convolution

$$y(n) = x(n) \odot h(n) = \sum_{n=0}^{N-1} x(n)h(m-n)$$ For Linear Convolution

3. Stop the program.

PROGRAM

a) Program to Perform Linear Convolution of Two Sequences

```
      .MMREGS
      .TEXT
START:
      LDP #02H
      LAR AR1, #8100H ; x(n) datas
      LAR AR0, #8200H ; h(n) datas
      LAR AR3, #8300H ; y(n) starting
      LAR AR4, #0007      ; N1+N2-1
; to fold the h(n) values
      LAR AR0, #08203H
      LACC #0C100H
      MAR *, AR0
      RPT #3
      TBLW *-
; padding of zeros for x(n) values
      LAR AR6, #8104H
      MAR     *, ar6
      LACC    #0H
      RPT     #3H
      SACL    *+
; convolution operation starts
```

```
LOP:    MAR *, AR1
        LACC *+
        SACL 050H     ; starting of the scope of multiplication
        LAR AR2, #0153H ; end of the array, to be multiplied with h(n) {150+N1-1}
        MAR *, AR2
        ZAP
        RPT #03H      ; N1-1 times so that N1 times


         MACD 0C100H,*-
        APAC          ; toaccumulate the final product sample
        MAR *, AR3
        SACL *+
        MAR *, AR4
        BANZ LOP,*-

H:      B HLT
```

INPUT AND OUTPUT SEQUENCE

```
; INPUT ( x(n) )

; 8100 - 1
; 8101 - 3
; 8102 - 1
; 8103 - 3

; INPUT ( h(n) )

; 8200 - 0
; 8201 - 1
; 8202 - 2
; 8203 - 1

; OUTPUT ( y(n) )

; 8300 - 0
; 8301 - 1
; 8302 - 5
; 8303 - 8
; 8304 - 8
; 8305 - 7
; 8306 - 3
```

b) Program to Perform Circular Convolution of Two Sequences

```
        .MMREGS
        .TEXT
START:
```

```
      LDP    #100H
      LACC   0H            ; length of the input is given in 8000
      SUB    #1H
      SACL   1H
      LAR    AR0, 1H
      LAR    AR1, #8060H;
      LAR    AR2, #8100H
COPYX2:
      MAR    *, AR1
      LACC   *+
      MAR    *, AR2
      SACL   *+
      MAR    *, AR0
      BANZ   COPYX2,*-
      LAR    AR0, 1H
      LAR    AR2, #8010H
LOOP3:
      LAR    AR1, #8060H; give the inputs x1[n] &h2[n] in AR1 & AR3
      LAR    AR3, #8050H
      LAR    AR4, 1H
      ZAP
LOOP:
      MAR    *, AR3; multiply x1[n] & X2[n] and add the; multiplication
      LT     *+
      MAR    *, AR1; output
      MPY    *+
      SPL    5H; store the partial result at location 8005
      ADD    5H ; add the partial result with accumulator
      MAR    *, AR4
      BANZ   LOOP,*-
      MAR    *, AR2
                                    ; outputs of correlation are stored in AR2
      SACL   *+
      CALL   ROTATE
LOOP2:
      MAR    *, AR0
      BANZ   LOOP3,*-

H:    B  HLT

ROTATE:
      LDP    #100H         ; rotate the values of X1[n]
      LACC   1H
      SUB    #1H
      SACL   2H
      LACC   0050H
      SACB         ; 8050 data moved to the accumulator to the accumulator buffer
      LAR    AR3, #8051H
      LAR    AR5, #8070H
```

```
      LAR     AR6, 2H
LOOP1:
      MAR     *, AR3
      LACC    *+
      MAR     *, AR5
      SACL    *+
      MAR     *, AR6
      BANZ    LOOP1,*-; DATA FROM 8051-8053 TO 8070-8072
      LACB        ; move the data accumulator buffer to accumulator as last data
      MAR     *, AR5
      SACL    *+
      LACC    #8070H
      SAMM    BMAR
      LAR     AR3, #8050H
      MAR     *, AR3
      RPT     #3H ; rotate 4 times
      BLDD    BMAR,*+ ; BMARautomatically incremented, to copy shifted data to 8050
      RET
```

INPUT AND OUTPUT SEQUENCE

```
; INPUT:
;             8000-0004
;

; X1(n) =   8050 - 0002
;             8051 - 0001
;             8052 - 0002
;             8053 - 0001
;

; H2(n ) =  8060 - 0001
;             8061 - 0002
;             8062 - 0003
;             8063 - 0004


; OUTPUT:
;             8010-000E
;             8011-0010
;             8012-000E
;             8013-0010
;
```

RESULT

Thus the linear convolution& circular convolution of the two sequences was performed using TMS320C50 DSP Processor.

Exp No 3

Implementation of FIR Filter

AIM

To design FIR filter for the following specification:

Approximation type: Window design - Blackmann Window
Filter type: Low-pass filter
Filter Order: 52
Cutoff frequency in KHz = 3.000000

APPARATUS REQUIRED

TMS320C50 DSP Kit, PC, RS232 Cable

PROCEDURE

1. Click on C50 debugger icon and start a new project.
2. Type the program and save it as an assembly file.
3. The input signal is given to the DSP kit.
4. Program is executed by giving G0C000 in communication window.
5. The sampled output can be viewed in CRO.

PROGRAM

```
.MMREGS
.TEXT                    ; Move the Filter coefficients from program memory to data
memory
START:
MAR*,AR0
LAR AR0,#0200H
RPT #33H
BLKP        CTABLE,*+
SETC        CNF          ; Input data and perform convolution
ISR:  LDP   #0AH
LACC        #0
SACL        0
IN    0,06H
LAR  AR7,#0              ;change value to modify sampling freq.
MAR*,AR7
BACK:       BANZ BACK,*-
IN    0,04H
NOP
```

```
NOP
NOP
NOP
MAR  *,AR1
LAR    AR1,#0300H
LACC 0
AND   #0FFFH
XOR   #800H
SUB    #800H
SACL  *
LAR    AR1,#333H
MPY #0
ZAC
ZAP
RPT    #33H
MACD          0FF00H,*-
APAC
LAR    AR1,#0300H
MAR  *,AR1
RPT    #01H
SFR
SACH *                          ; give as sach *,1 incase ofoverflow
LACC *
ADD   #800H
SACL  *
OUT   *,04H
IN      0,16H
NOP
B       ISR
NOP
NOP
HLT:  B       HLT
CTABLE:
      .word    0FE8BH
      .word    0FEE2H
      .word    0FF74H
      .word    02BH
      .word    0ECH
      .word    0196H
      .word    0208H
      .word    022BH
      .word    01EDH
      .word    0150H
      .word    061H
      .word    0FF40H
      .word    0FE18H
      .word    0FD19H
      .word    0FC77H
      .word    0FC5EH
```

```
.word   0FCEEH
.word   0FE35H
.word   02BH
.word   02B4H
.word   059EH
.word   08A9H
.word   0B8FH
.word   0E07H
.word   0FD4H
.word   010C7H
.word   010C7H
.word   0FD4H
.word   0E07H
.word   0B8FH
.word   08A9H
.word   059EH
.word   02B4H
.word   02BH
.word   0FE35H
.word   0FCEEH
.word   0FC5EH
.word   0FC77H
.word   0FD19H
.word   0FE18H
.word   0FF40H
.word   061H
.word   0150H
.word   01EDH
.word   022BH
.word   0208H
.word   0196H
.word   0ECH
.word   02BH
.word   0FF74H
.word   0FEE2H
.word   0FE8BH
```

RESULT

Thus a FIR low pass filter is designed using TMS320C50 processor.

Exp No 4

Implementation of IIR Filter

AIM

To design an IIR low pass filter using TMS320C50.

APPARATUS REQUIRED

TMS320C50 DSP Kit, PC, RS232 Cable

PROCEDURE

1. Click on C50 debugger icon and start a new project.
2. Type the program and save it as an assembly file.
3. The input signal is given to the DSP kit.
4. Program is executed by giving G0C000 in communication window.
5. The sampled output can be viewed in CRO.

PROGRAM

```
.MMREGS
.TEXT
START:
        LDP #100H
        LACC #0
        SACL 02H
        SACL 03H
        SACL 00H
  LOOP:
        LACC #0
        SACL 00H
        IN 0,06H
        LAR AR7,#30H
        MAR *,AR7
  BACK:BANZ BACK,*-
        IN 0,04H
        NOP
        NOP
        NOP
        NOP
        LACC 00H
        AND #0FFFH
        XOR #800H
        SUB #800H
        SACL 00H
```

```
        LT 00H
        MPY #315EH
        PAC
        SACH 2H,1
        LT 03H
        MPY #4E9FH
        PAC
        SACH 04H,1
        LACC 02H
        ADD 04H
        SACL 03H
        ADD #800H
        SACL 00H
        OUT 00H,04H
        IN 0,16H
        B LOOP
        NOP
        NOP
HLT: B HLT
```

RESULT

Thus an IIR low pass filter is designed using TMS320C50 processor.

Exp No 5

Sampling of Analog Signal

AIM

To write a program to sample and display an analog signal using TMS320C50 Kit.

APPARATUS REQUIRED

TMS320C50 DSP Kit, PC, RS232 Cable

PROCEDURE:

6. Click on C50 debugger icon and start a new project.
7. Type the program and save it as an assembly file.
8. The input signal is given to the DSP kit.
9. Program is executed by giving G0C000 in communication window.
10. The sampled output can be viewed in CRO.

PROGRAM

```
DATA .SET 2H
DELAY .SET3H
.MMREGS
.TEXT
START:          LDP#100H
                LAR AR0, #9000H
                LAR AR1, #719H
REP:            IN 0, 06
        RPT #OFH
                 NOP
                IN 0, 04
                SPLK #00FFH, DELAY
                RPT DELAY
                NOP
                LACC 0
                AND #0FFFH
                MAR*, AR0
                SACL*+, 0, AR1
                BANZ REP,*-
CONT1:           LAR AR0, #9000H
                LAR AR1, #719H
                LOOP: MAR*, AR0
                RPT #1FFH
                NOP
                OUT*+, 04, AR1
```

```
                BANZ LOOP,*-
B CONT1
```

RESULT:

Thus the input signal was sampled and displayed.

Familiarization of MATLAB

MATLAB is an interactive program for doing matrix calculations and has now grown to a high level mathematical language that can solve integrals and differential equations numerically and plot a wide variety of two and three dimensional graphs.

1. Definition of Variables

Variables are assigned numerical values by typing the expression directly. The answer will not be displayed when a semicolon is put at the end of an expression.
MATLAB utilizes the following arithmetic operators:

+ Addition
- Subtraction
* Multiplication
/ Division
^ Power operator
‘Transpose

A variable can be assigned using a formula that utilizes these operators and either numbers or previously defined variables. To determine the value of a previously defined quantity, type the quantity by itself. If your expression does not fit on one line, use an ellipsis (three or more periods at the end of the line) and continue on the next line. There are several predefined variables which can be used at any time, in the same manner as user defined variables:

i - sqrt(-1)
j - sqrt(-1)
pi - 3.1416...

There are also a number of predefined functions that can be used when defining a variable. Some common functions that are used in this text are:

Abs- magnitude of a number (absolute value for real numbers)
angle- angle of a complex number, in radians
cos - cosine function, assumes argument is in radians
sin - sine function, assumes argument is in radians
exp - exponential function(can be used on complex numbers

2. Definition of Matrices

MATLAB is based on matrix and vector algebra; even scalars are treated as 1x1 matrices. Therefore, vector and matrix operations are as simple as common calculator operations. Vectors can be defined in two ways. The first method is used for arbitrary elements.

v = [1 3 5 7]; Creates a 1x4 vector with elements 1, 3, 5 and 7.

Note that commas could have been used in place of spaces to separate the elements. Additional elements can be added to the vector,v(5) = 8; yields the vector v = [1 3 5 7 8]. Previously defined vectors can be used to define a new vector.

The second method is used for creating vectors with equally spaced elements.

t = 0:0.1:10; Creates a 1x11 vector with the elements 0, .1, .2, .3 ... 10. Note that the middle number defines the increment. If only two numbers are given, then the increment is set to a default of 1.

k = 0:10; Creates a 1x11 vector with the elements 0, 1, 2... 10.

Matrices are defined by entering the elements row by row.

M = [1 2 4; 3 6 8]; creates the matrix

M = [1 2 4]

[3 6 8]

There are a number of special matrices that can be defined as follows.

Null matrix: M = [ ];

nxm matrix of zeros: M = zeros(n,m);

nxm matrix of ones: M = ones(n,m);

nxn identity matrix: M = eye(n);

A particular element of a matrix can be assigned as, M (1, 2) = 5; Places the number 5 in the first row, second column. Operations and functions that were defined for scalars in the previous section can also be used on vectors and matrices.

Functions are applied element by element. For example,

t = 0:10;

x = cos(2*t); Creates a vector x with elements equal to cos(2t) for t = 0, 1, 2, ..., 10. Operations that need to be performed element-by-element can be accomplished by preceding the operation by a ".". For example, to obtain a vector x that contains the elements of x (t) = tcos(t) at specific points in time, you cannot simply multiply the vector t with the vector cos(t). Instead you multiply their elements together:

```
t = 0:10;
x = t.*cos(t);
```

3. General information

MATLAB is case sensitive so "a" and "A" are two different names. Comment statements are preceded by a "**%**". On-line help for MATLAB can be reached by typing help for the full menu or typing help followed by a particular function name or M-file name. For example, help cos gives help on the cosine function.

The number of digits displayed is not related to the accuracy. To change the format of the display, type **format short e** for scientific notation with 5 decimal places, **format long e** for scientific notation with 15 significant decimal places and format bank for placing two significant digits to the right of the decimal.

The commands **who** and **whos** give the names of the variables that have been defined in the workspace. The command **length(x)** returns the length of a vector x and **size(x)** returns the dimension of the matrix x.

4. M-files

M-files are macros of MATLAB commands that are stored as ordinary text files with the extension "m", that is *filename*.m. An M-file can be either a function with input and output variables or a list of commands.

The following describes the use of M-files on a PC version of MATLAB.

MATLAB requires that the M-file must be stored either in the working directory or in a directory that is specified in the MATLAB path list. For example, consider using MATLAB on a PC with a user-defined M-file stored in a directory called "\MATLAB\MFILES". Then to access that M-file, either changes the working directory by typing cd\matlab\mfiles from within the MATLAB command window or by adding the directory to the path. Permanent

addition to the path is accomplished by editing the \MATLAB\matlabrc.m file, while temporary modification to the path is accomplished by typing addpath c:\matlab\mfiles from within MATLAB.

MATLAB M-files are most efficient when written in a way that utilizes matrix or vector operations. Loops and if statements are available, but should be used sparingly since they are computationally inefficient. An if statement can be used to define conditional statements. The allowable comparisons between expressions are >=, <=, <, >, ==, and ~=. Suppose that you want to run an M-file with different values of a variable T. The following command line within the M-file defines the value.

T = input ('Input the value of T: ')

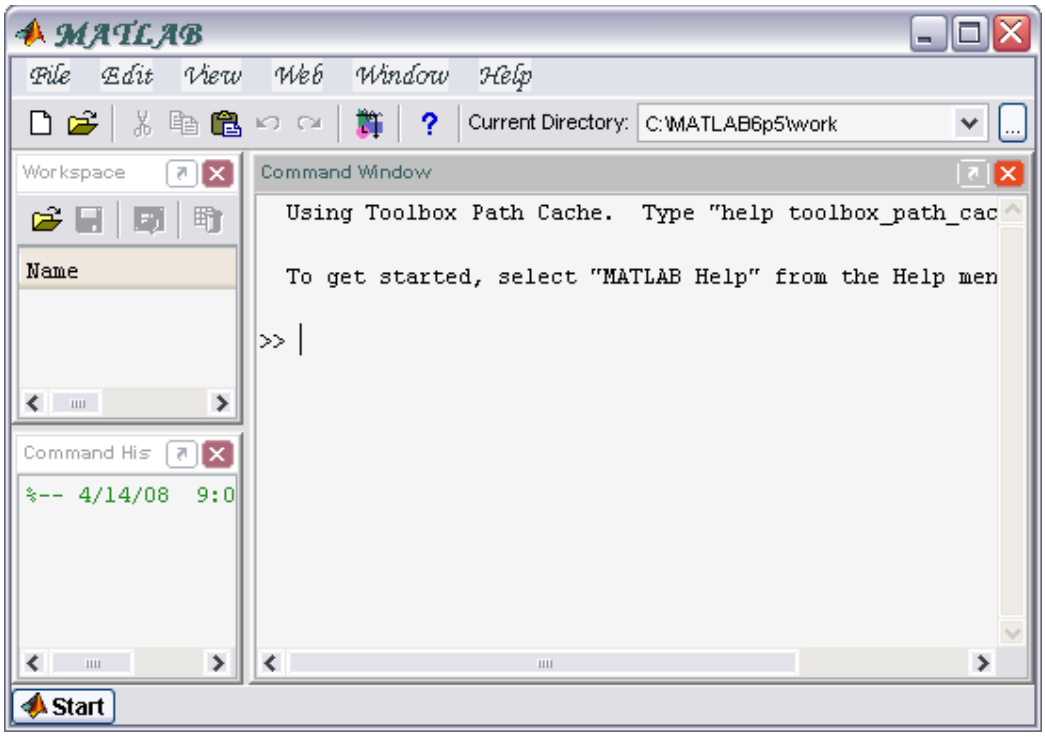
Whatever comment is between the quotation marks are displayed to the screen when the M-file is running, and the user must enter an appropriate value.

5. How to get started??

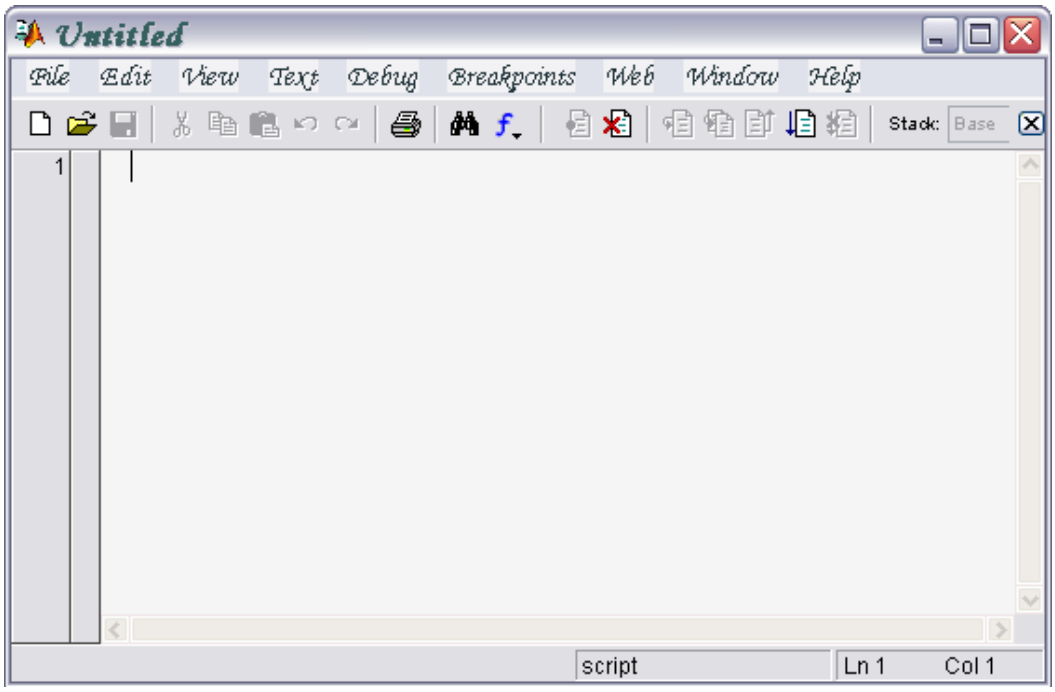
Double click on icon for MATLAB. Within about 30 seconds MATLAB will open, a screen like the picture given below.

This is the MATLAB screen. It has broken into 3 parts.

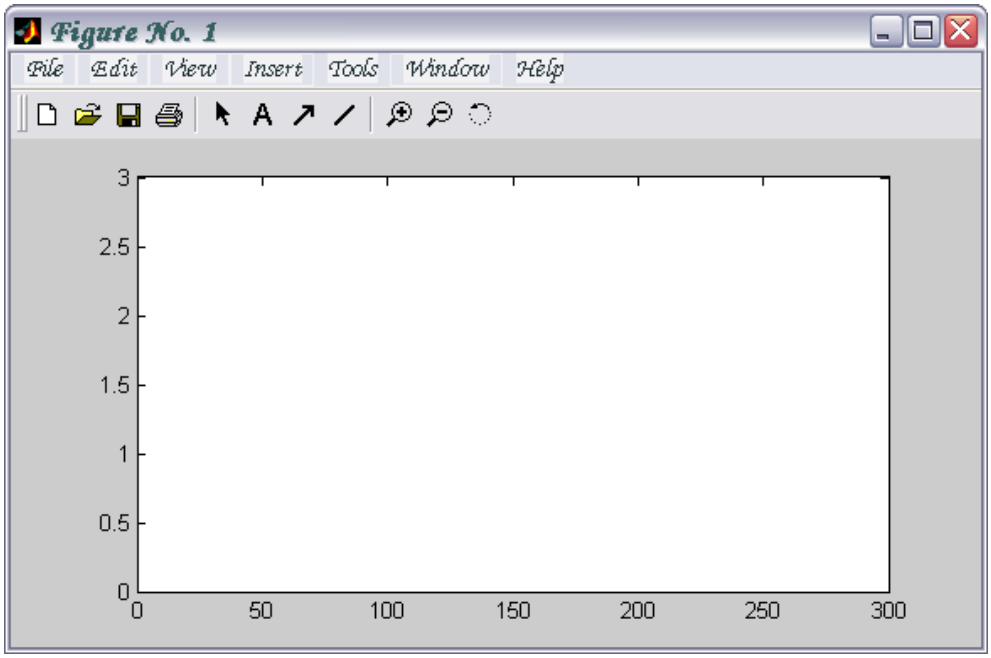
- ***Command Window*** – This is where you can type commands and usually the answers (or error messages) appear here too. You will see the cursor flickering after the >> prompt. This means that MATLAB is waiting for further instructions.
- ***Workspace***– if you define new quantities (called variables) their names should be listed here.
- ***Command History***– This is past commands are remembered. If you want to re-run a previous command or to edit it you can drag it from this window to the command window to re-run it.



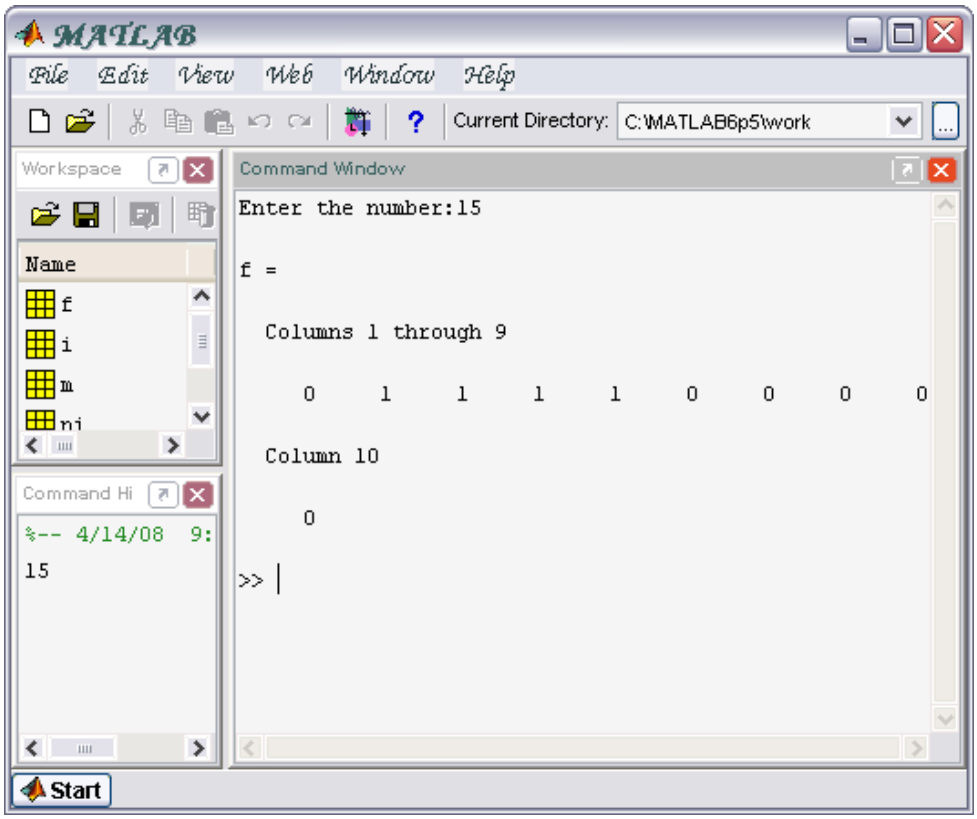
To begin to use MATLAB, click **New: M-file** from the **File** menu. This opens a blank window as shown below.



The M-file is executed using the Run command under the Tools menu. The output signal appears in Figure Window.



The output data appears in Command Window.



Exp No 6

Generation of Waveforms

AIM

To generate the basic signals (Continuous and Discrete).

THEORY

A signal is defined as any physical quantity that varies with time, space or any other independent variable. A system is defined as a physical device that performs an operation on a signal. Signal processing is any operation that changes the characteristics of a signal using a system. These characteristic include the amplitude, shape, phase and frequency content of the signal.

PROGRAM

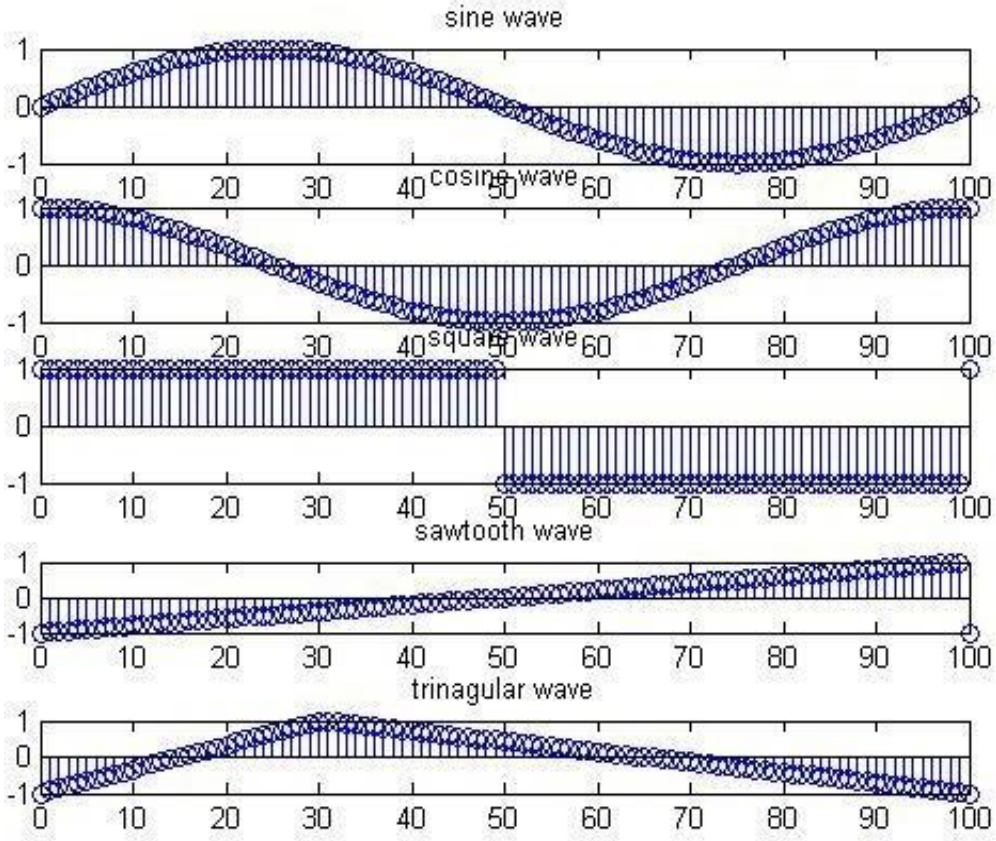
```
% periodic continuous signals
t=0:0.01:6*pi;
f=1;
%sine wave
x=sin(2*pi*f*t);
subplot(3,2,1);
plot(t,x);
title('sine wave');
grid on

%cosine wave
y=cos(2*pi*f*t);
subplot(323);
plot(t,y);
title('cosine wave');
grid on

%tangent wave
t=-3*pi:0.01:3*pi;
```

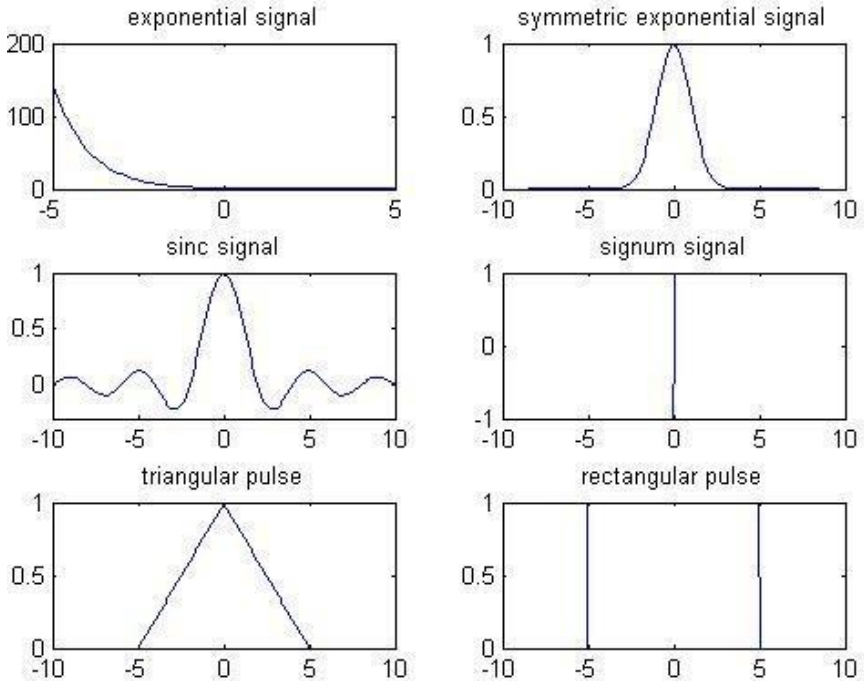
```
z=tan(2*pi*f*t);
subplot(325);
plot(t,z);
title('tan wave');
grid on

%square wave
t=0:0.001:1;
f=10;
a=square(2*pi*f*t);
subplot(3,2,2);
plot(t,a);
title('square wave');

%sawtooth wave
b=sawtooth(2*pi*f*t);
subplot(3,2,4);
plot(t,b);
title('sawtooth wave');

%triangular wave
c=sawtooth(2*pi*f*t,0.3);
subplot(3,2,6);
plot(t,c);
title('trinagular wave');

% discrete waveforms
n=0:1:100;
k=0.01;
%sine wave
x=sin(2*pi*k*n);
subplot(5,1,1);
stem(n,x);
```

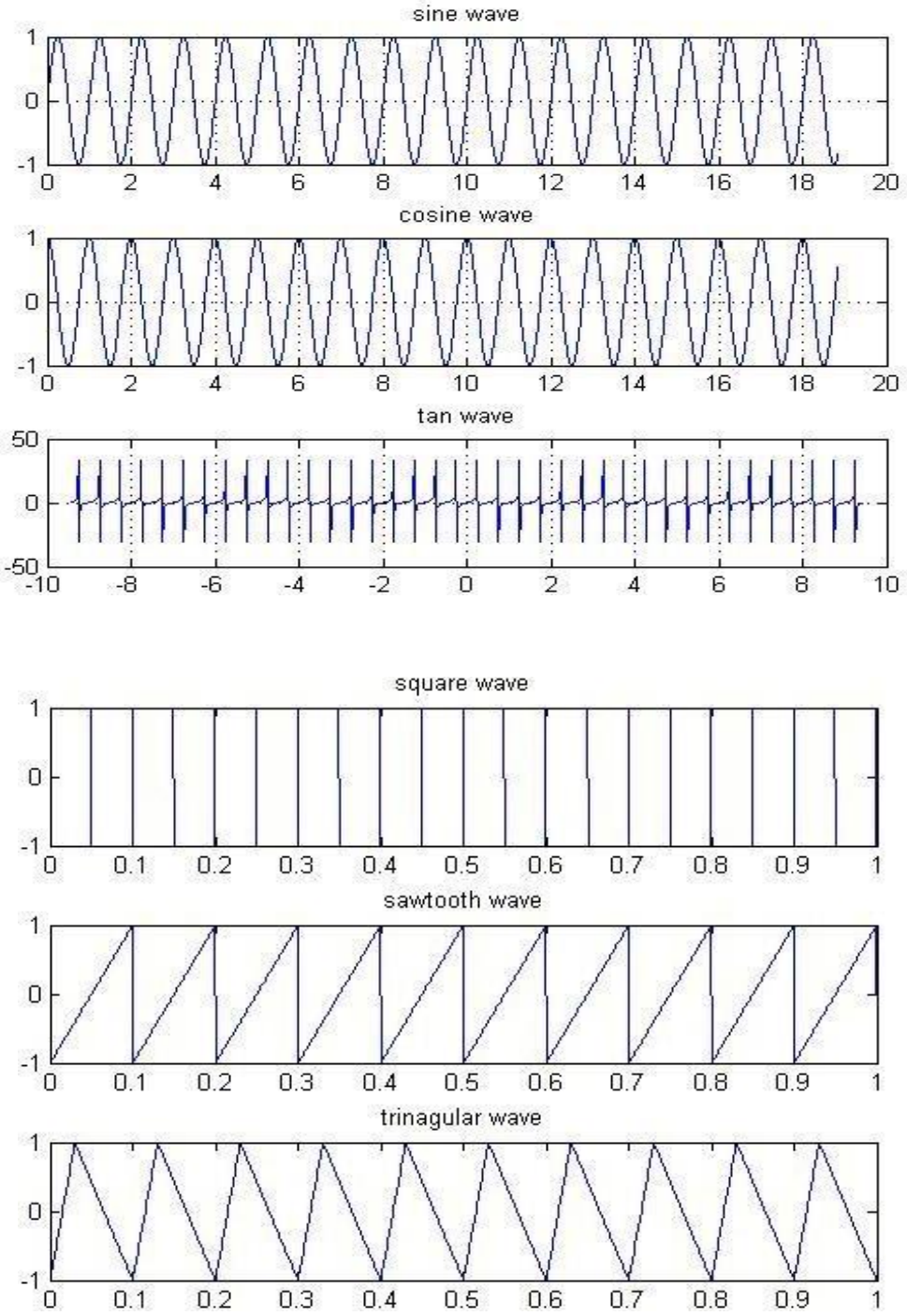
```
title('sine wave');

%cosine wave
y=cos(2*pi*k*n);
subplot(5,1,2);
stem(n,y);
title('cosine wave');

%square wave
z=square(2*pi*k*n);
subplot(5,1,3);
stem(n,z);
title('square wave');

%sawtooth wave
a=sawtooth(2*pi*k*n);
subplot(5,1,4);
stem(n,a);
title('sawtooth wave');

%triangular wave
b=sawtooth(2*pi*k*n,0.3);
subplot(5,1,5);
stem(n,b);
title('trinagular wave');

%aperiodic signals
%exponential signal
clc;
clear  all;
close  all;
t=-5:.1:5;
x1=exp(-t);
```

```
subplot(3,2,1);
plot(t,x1);
title('exponential signal');
t=-10:.1:10;
x2=exp(-(t.^2)/2);
subplot(3,2,2);
plot(t,x2);
title('symmetric exponential signal');

%sinc signal
x3=sinc(t/2);
subplot(3,2,3);
plot(t,x3);
title('sinc signal');
axis([-10 10 -0.3 1]);

%signumfsignal
x4=sign(t);
subplot(3,2,4);
plot(t,x4);
title('signum signal');

%triangular pulse
x5=tripuls(t/10);
subplot(3,2,5);
plot(t,x5);
title('triangular pulse');

%rectangular pulse
x6=rectpuls(t/10);
subplot(3,2,6);
plot(t,x6);
title('rectangular pulse');
```

OUTPUT WAVEFORMS







RESULT:

Using MATLAB functions, the required basic signals were obtained.

Exp No: 7

Verification of Sampling Theorem

AIM

To verify the sampling theorem.

THEORY

A bandlimited signal x(t) can be reconstructed exactly if it is sampled at a rate atleast twice the maximum frequency component in it. If the maximum frequency component of x(t) is fm. To recover the signal x(t) exactly from its samples it has to be sampled at a rate fs ≥ 2fm. The minimum required sampling rate fs = 2fm is called Nyquist rate.

PROGRAM

```
close all;
clear all;
t=-10:0.01:10;
T=8;
fm=1/T;
x=cos(2*pi*fm*t);
fs1=1.2*fm;
fs2=2*fm;
fs3=8*fm;
n1=-4:1:4;
xn1=cos(2*pi*n1*fm/fs1);
subplot(221)
plot(t,x);
xlabel('time in seconds');
ylabel('x(t)');
title('continous time signal');
subplot(222)
stem(n1,xn1);
hold on;
```

```
plot(n1,xn1);
xlabel('n');
ylabel('x(n)');
title('discrete time signal with fs<2fm');
%
n2=-5:1:5;
xn2=cos(2*pi*n2*fm/fs2);
subplot(223)
stem(n2,xn2);
hold on;
plot(n2,xn2);
xlabel('n');
ylabel('x(n)');
title('discrete time signal with fs=2fm');
%
n3=-20:1:20;
xn3=cos(2*pi*n3*fm/fs3);
subplot(224)
stem(n3,xn3);
hold on;
plot(n3,xn3);
xlabel('n');
ylabel('x(n)');
title('discrete time signal with fs>2fm');
```

RESULT

Written and executed a MATLAB program to verify the sampling theorem.

Exp No: 8

AM & FM Generation

AIM

To write a MATLAB program for the generation of amplitude modulated signal and frequency modulated signal.

THEORY

For the efficient transmission and reception of audio frequency signals transmitter and receiver antennas should have a height of quarter wavelength of the frequency used. Such a very long antennas are impractical to realize at low and medium frequency range. Also the audio frequency signals are susceptible to noise; to avoid this problem modulation is used.

AM is the method of transmitting signals such as sound or digital information, in which the amplitude of the carrier wave is changed according to the message signal. Here we are simply adding the carrier amplitude with message signal to obtain AM signal, then the instantaneous amplitude of the carrier get altered with respect to the modulating signal.

Let the modulating signal be

$Em(t)=Em \sin(wmt)$

and the carrier signal be

$Ec(t)=Ec \sin(wct)$

then the modulated signal e[t] is expressed as

$e(t)=Ec \sin(wct)(1+m\sin(wmt))$

In FM, frequency of the carrier signal having high frequency is varied in accordance with the instantaneous amplitude of the modulating signal having low frequency.FM signals can be easily plotted using simple MATLAB functions.

Let the modulating signal be

$Em(t)=Emcos(wmt)$

and the carrier signal be

$Ec(t)=Eccos(wct)$

then the modulated signal e[t] is expressed as

e(t)=Eccos((wct)+m sin(wmt)) , where m is known as the modulation index

ALGORITHM

For AM

1. Enter message frequency fm, message amplitude Em, carrier frequency fc, carrier amplitude Ec and modulation index m.
2. Generate message signal em by using the expression e(t)=Em sin(wmt)
3. Generate carrier signal ec by using the expression ec(t)=Ec sin(wct)
4. Generate amplitude modulated signal e[t] , e[t]=Ec sin(wct)(1+msin(wmt))
5. Plot message, carrier and modulated signal

PROGRAM

For AM

```
clc;
clear all;
close all;
Em=input('enter the amplitude of message signal \n');
Fm=input('enter the frequency of message signal \n');
Ec=input(‘enter the amplitude of carrier signal\n’);
Fc=input(‘enter the frequency of carrier signal\n’);
m=input('enter modulation index \n');
Fs=input('enter the sampling frequency\n');
t=linspace(0,5/fm,1000);
em=Em*sin(2*pi*Fm*t);
figure(1);
subplot(3,1,1);
plot(t,em);
xlabel('time(s)');
ylabel('amplitude(v)');
title('modulating signal');
ec=Ec*sin(2*pi*Fc*t);
subplot(3,1,2);
```

```
plot(t,ec);
xlabel('time(s)');
ylabel('amplitude(V)');
title('carrier signal');
e=Ec*(1+m*sin(2*pi*Fm*t)).*sin(2*pi*Fc*t);
subplot(3,1,3);
plot(t,e);
xlabel('time(s)');
ylabel('amplitude(v)');
title('modulated signal');

%spectrum
a=fftshift(fft(em,Fs));
b=fftshift(fft(ec,Fs));
c=fftshift(fft(e,Fs));
f=-Fs/2: Fs/2-1;
figure(2);
subplot(3,1,1);
plot(f,abs(a));
title('carrier spectrum');
axis([-100 100 0 500]);
subplot(3,1,2);
plot(f,abs(b));
title('message signal spectrum');
axis([-100 100 0 500]);
subplot(3,1,3);
plot(f,abs(c));
title('AM spectrum');
axis([-100 100 0 2500]);
```

OBSERVATIONS

Enter the amplitude of message signal

5

Enter the frequency of message signal

500

Enter the amplitude of the carrier signal

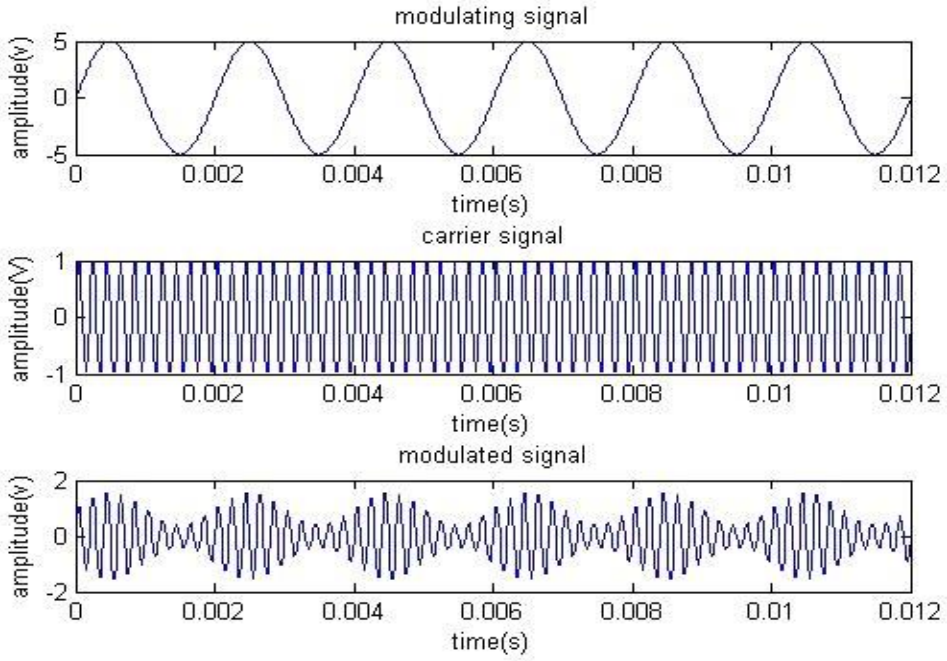
1

Enter the frequency of the carrier signal

5000

Enter modulation index

0.6



ALGORITHM

For FM

1. Enter message frequency fm, message amplitude Em, carrier frequency fc, carrier amplitude Ec and modulation index m.
2. Generate message signal em by using the expression em(t)=Emcos(wmt)
3. Generate carrier signal ec by using the expression ec(t)=Eccos(wct)
4. Generate frequency modulated signal e[t] , e[t]=Eccos((wct)+m*sin(wmt))

5. plot message, carrier and modulated signal

PROGRAM

For FM

```
clc;
clear all;
close all;
Em=input('enter the amplitude of message signal \n');
Fm=input('enter the frequency of message signal \n');
Ec=input('enter the amplitude of carrier signal\n');
Fc=input('enter the frequency of carrier signal\n');
Fs=input('enter the sampling frequency\n');
m=input('enter modulation index \n');
t=linspace(0,5/fm,1000);
em=Em*sin(2*pi*Fm*t);
figure(1);
subplot(3,1,1);
plot(t,em);
xlabel('time(s)');
ylabel('amplitude(v)');
title('modulating signal');
ec=Ec*sin(2*pi*Fc*t);
subplot(3,1,2);
plot(t,ec);
xlabel('time(s)');
ylabel('amplitude(V)');
title('carrier signal');
e=Ec*cos(2*pi*Fc*t+m*sin(2*pi*Fm*t))
subplot(3,1,3);
plot(t,e);
xlabel('time(s)');
ylabel('amplitude(v)');
title('modulated signal');
```

```
%FM spectrum
a=fftshift(fft(em,Fs));
b=fftshift(fft(ec,Fs));
c=fftshift(fft(e,Fs));
f=-Fs/2: Fs/2-1;
figure(2);
subplot(3,1,1);
plot(f,abs(a));
title('carrier spectrum');
axis([-100 100 0 500]);
subplot(3,1,2);
plot(f,abs(b));
title('message signal spectrum');
axis([-100 100 0 500]);
subplot(3,1,3);
plot(f,abs(c));
title('FM spectrum');
axis([-100 100 0 200]);
```

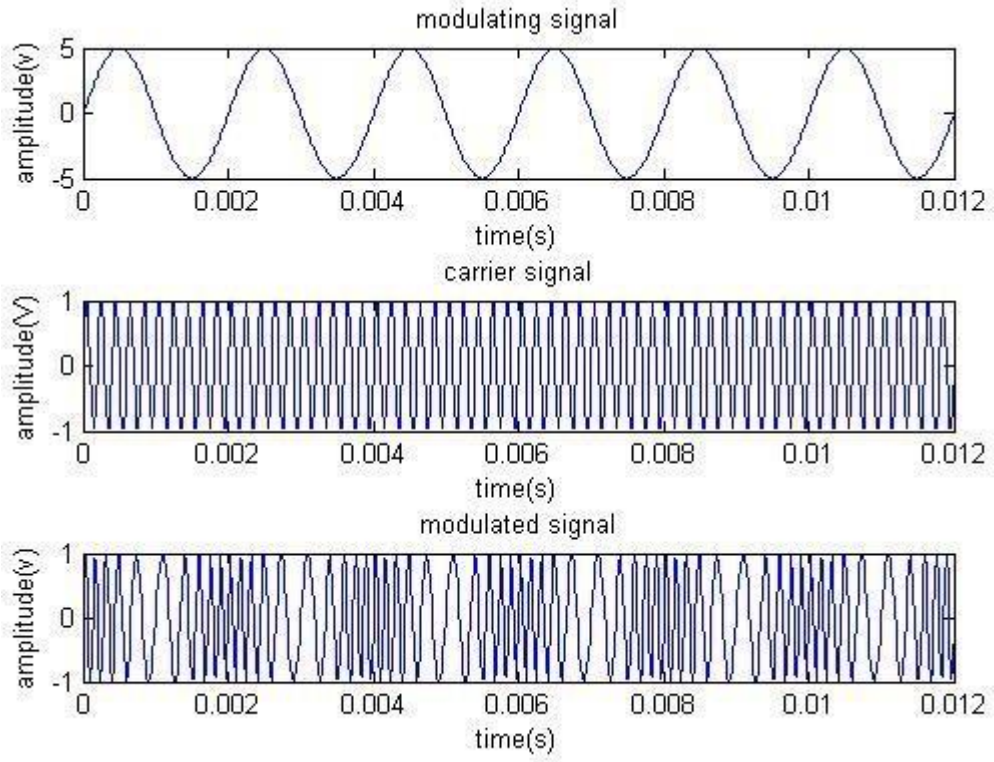
OBSERVATION

```
Enter the amplitude of message signal
5
Enter the frequency of message signal
500
Enter the amplitude of carrier signal
1
Enter the frequency of carrier signal
```

5000

Enter modulation index

10



RESULT

Written and executed a MATLAB program to generate AM and FM waves and verified the output.

Exp No 9

Linear& Circular Convolution

(a) Linear Convolution

AIM

Write a MATLAB program to find the linear convolution of two sequences.

THEORY

Convolution is an integral concatenation of two signals. The most popular application is the determination of the output signal of a linear time-invariant system by convolving the input signal with the impulse response of the system. Convolving two signals is equivalent to multiplying the Fourier transform of the two signals. The linear convolution of two continuous time signals x(t) and h(t) is defined by

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

For discrete time signals x(n) and h(n),convolution is defined by

$$y[n] = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

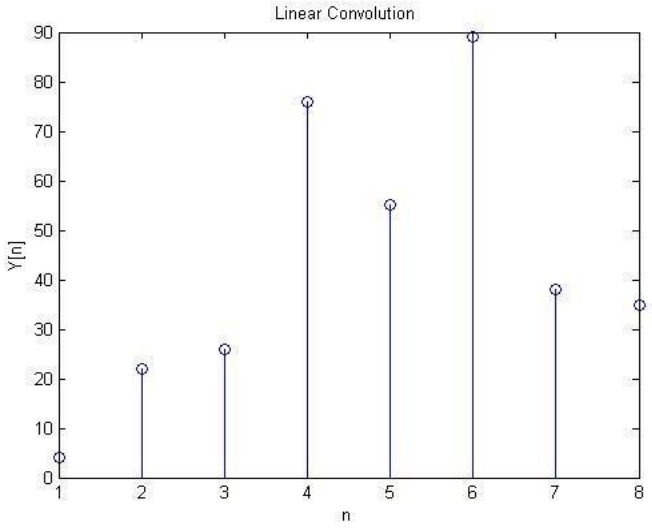
ALGORITHM

1. Get the input sequence x(n) and impulse response h(n).
2. Choose an initial value of n,the starting time for evaluating the output sequence y(n). If x(n) starts at n = n1 and h(n) starts at n = n2, then n = n1 + n2 is a good choice.
3. Express both sequences in terms of the index k.
4. Fold h(k) about k=0 to obtain h(-k) and shift by n to the right if n is positive and left if n is negative to obtain h(n-k).
5. Multiply x(n) and h(n-k) element by element and sum up the products to get y(n).
6. Increment the index n, shift the sequence h(n-k) to right by one sample and do Step 4.
7. Repeat step 5 until the sum of products is zero for all the remaining values of n.

PROGRAM

```
clc;
clear all;
close all;
x=input('enter the input sequence');
h=input('enter the impulse response');
m=length(x);
n=length(h);
X=[x,zeros(1,n)];
H=[h,zeros(1,m)];
fori=1:n+m-1
Y(i)=0;
for j=1:m
if(i-j+1>0)
Y(i)=Y(i)+X(j)*H(i-j+1);
else
end
end
end
disp('Output Y');
disp(Y);
stem(Y);
ylabel('Y[n]');
xlabel('n');
title('Linear Convolution ');
```

OUTPUT:



Enter the input sequence: [1 5 2 7]

Enter the impulse response: [4 2 8 4 5]

Output Y =: [4 22 26 76 55 89 38 35]

(b) Circular Convolution

AIM

Write a MATLAB program to find the Circular Convolution of two sequences.

THEORY

Circular convolution of two finite duration sequences x1[n] and x2[n] both of the length N is defined by

$$x_3[n] = x_1[n]x_2[n] = \sum_{m=0}^{M-1} x_1(m)x_2((n-m)_N)$$

Multiplication of the DFTs of two sequences is equivalent to the circular convolution of the two sequences in time domain. If x1[n] is a sequence of L number of samples and x2[n] with M samples ,after convolution y[n] will contain N= max(L,M) samples. It cannot be used to find the response of a linear filter without zero padding.

ALGORITHM

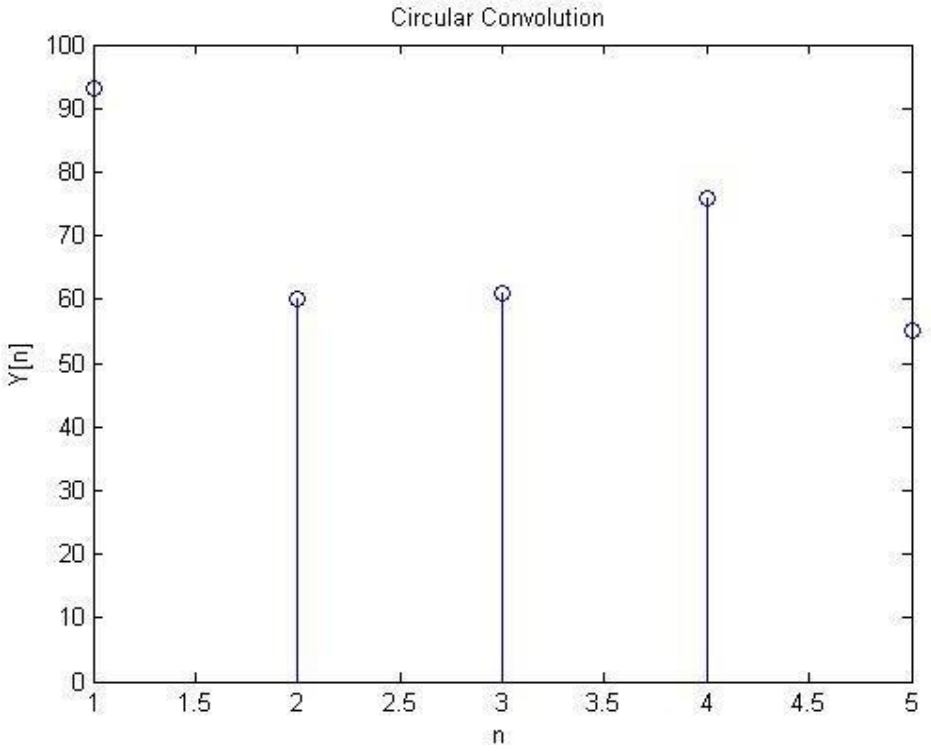
1. Get the input sequence x(n) and impulse response h(n).

2. Fold h(k) about k=0 to obtain h(-k) and shift by n to the right if n is positive and left if n is negative to obtain h(n-k).
3. Multiply corresponding samples on the two circles and sum the products to produce output.
4. Rotate h(n-k) one sample at a time in counter clockwise direction and go to step 3 to obtain the next value of output.
5. Repeat the step 4 until h(k) first sample lines up with the first sample of the x(k) once again.

PROGRAM

```
clc;
clear all;
close all;
x=input('enter the input sequence');
h=input('enter the impulse response');
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;
fori=0:(N-1)
j=mod(n-i,N);
y(n+1)=y(n+1)+x(i+1)*h(j+1);
end
end
disp('Output Y');
disp(y);
stem(y);
ylabel('Y[n]');
xlabel('n');
title('Circular Convolution ');
```

OUTPUT:



Enter the input sequence: [1 5 2 7]

Enter the impulse response: [4 2 8 4 5]

Output Y =: [93 60 61 76 55]

RESULT

Obtained the linear and circular convolution of two sequences and verified the output.

Exp No: 10

DFT & IDFT

AIM

Write a MATLAB program to find the DFT and IDFT for the given input sequence.

THEORY

Given a sequence of N samples x(n), indexed by n = 0..N-1, the Discrete Fourier Transform (DFT) is defined as X(k),where k=0..N-1:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi(nk)/N} \qquad k = 0, 1, ..., N-1$$

The sequence x(n) can be calculated from X(k) using the Inverse Discrete Fourier Transform (IDFT):

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi nk/N} \qquad 0 \le n < N$$

ALGORITHM

DFT

1. Get the input sequence x(n).
2. Specify the length of dft (N), N should be a power of 2.
3. If N> length of x then pad N-l zeros to the input.
4. Implement the DFT equation

PROGRAM

DFT

```
x=input('enter the input sequence x[]\n');
N=input('length\n');
N1=length(x);
x=[x zeros(1,N-N1)];
for k=0:N-1
```

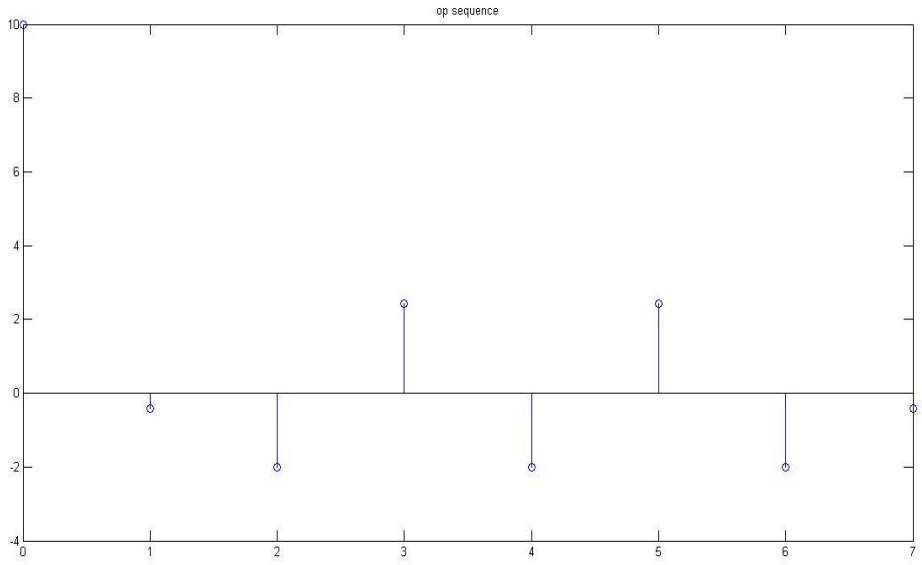
```
y(k+1)=0;
for n=0:N-1
y(k+1)=y(k+1)+x(n+1)*exp(-j*2*pi*k*n/N);
end
end
stem([0:length(y)-1],y)
title('op sequence');
```

OUTPUT

Enter the sequence x[1 2 3 4]

The length: 8

Ans:10.0000 -0.4142 - 7.2426i -2.0000 + 2.0000i 2.4142 - 1.2426i -2.0000 - 0.0000i 2.4142 + 1.2426i -2.0000 - 2.0000i -0.4142 + 7.2426i



IDFT

ALGORITHM

1. Enter the input sequence.
2. Calculate the length of input sequence.
3. By zero padding create matrix for output sequence.
4. Find IDFT by using the equation.

PROGRAM

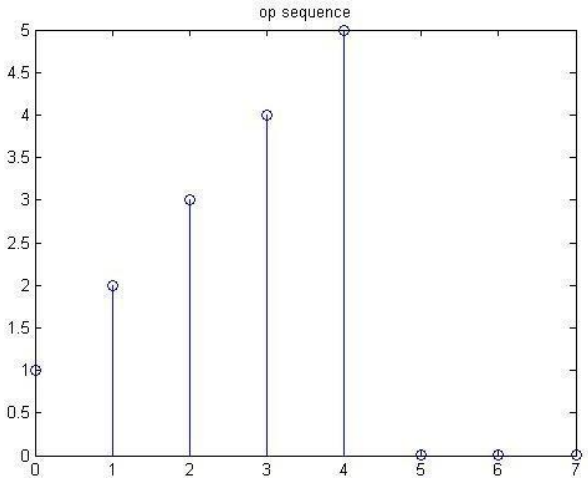
```
x=input('enter the input sequence x[]\n');
N=input('length\n');
N1=length(x);
x=[x zeros(1,N-N1)];
for n=0:N-1
z(n+1)=0;
for k=0:N-1
z(n+1)=z(n+1)+(x(k+1)*exp(j*2*pi*k*n/N));
end
end
z=z/N
stem(z)
title('op sequence');
```

OUTPUT

Enter the sequence: x[15 -5.4142-7.2426i 3+2i -2.5858-1.2426i 3 -2.5858+1.2426i 3-2i -5.4142+7.2426i]

The length: 8

Ans: 1.0000 - 0.0000i 2.0000 - 0.0000i 3.0000 - 0.0000i 4.0000 + 0.0000i 5.0000 - 0.0000i 0.0000 + 0.0000i 0.0000 - 0.0000i 0.0000 + 0.0000i



RESULT

Obtained DFT and IDFT of the given sequence and verified.

Exp No: 11

IIR FilterDesign-Butterworth &Chebychev

THEORY

The filters designed by considering all the infinite samples of impulse response are called IIR filters. The impulse response is obtained by taking inverse Fourier Transform of ideal frequency response. The popular methods for such filter design use the technique of transforming the analog filter to an equivalent digital filter. We know that the analog filter with transfer function Ha(s) is stable if all its poles lie in the left half of the s-plane. Consequently, if the conversion technique is to be effective, it should posses the following desirable properties.

1. The imaginary axis in the s-plane should map into the unit circle in the z-plane .Thus there will be a direct relationship between the two frequency variables in the two domains.
2. The left half of the s-plane should map into the interior of the unit circle in the z-plane. Thus a suitable analog filter will be converted to a stable digital filter.

The IIR filter is a discrete time system that is designed to pass the spectral content of the input signal in a specified band of frequencies. Based on the frequency response the filters are classified into four types. They are Low pass; High pass, Band pass, and Band stop filters. A number of solutions to the approximation problem of analog filter design are well developed. The popular among them are Butterworth and Chebyshev approximation. For designing a digital IIR filter, first an equivalent analog filter is designed using any one of the approximation technique and the given specifications. The result of the analog filter design will be an analog filter transfer function Ha(s) .The analog transfer function is converted to digital transfer function H(z) using either Bilinear or Impulse invariant transformation . The digital transfer function H(z) can be realized in a software that runs on a digital hardware (or it can be implemented in firmware).

Important features of IIR filters

1. The physically realizable IIR filters does not have linear phase.

2. The IIR filter specifications include the desired characteristics for the magnitude response only.

Butterworth Filters

(a) BUTTERWORTH IIR LPF

AIM

To write a program to study the characteristics of Butterworth IIR low pass filter.

ALGORITHM

1. Enter the passband ripple rp, stopband ripple rs, passband frequency in Hz, sampling frequency in Hz, stopband frequency in Hz.
2. Compute the frequency in radians.
3. Find cut off frequency and order of filter using command word 'buttord'.
4. Find the coefficients of filter using command word 'butter'.
5. Find the frequency response of filter using the coefficients with command word freqz. Let it be h and omega.
6. Find the magnitude and phase of filter output h.
7. Plot the magnitude and phase response of filter.

PROGRAM

```
clc;
rp=input('Enter the passband ripple:');
rs= input('Enter the stopband ripple:');
wp= input('Enter the passband frequency:');
ws= input('Enter the stopband frequency:');
fs=input('Enter the sampling frequency:');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs);
[b a]=butter(n,wn);
W=[0 :0.01:pi];
[h,omega]=freqz(b,a,w);
```

```
m=20*log10(abs(h));
an=angle(h);
% plot amplitude response
Subplot(2,1,1);
Plot(omega/pi,m);
Xlabel('Normalised frequency');
Ylabel('Gain in dB');
Title('Amplitude  response');
% plot phase response
Subplot(2,1,2);
Xlabel('Normalised frequency');
Ylabel('phase in radians');
Title('phase  response');
```

OBSERVATIONS

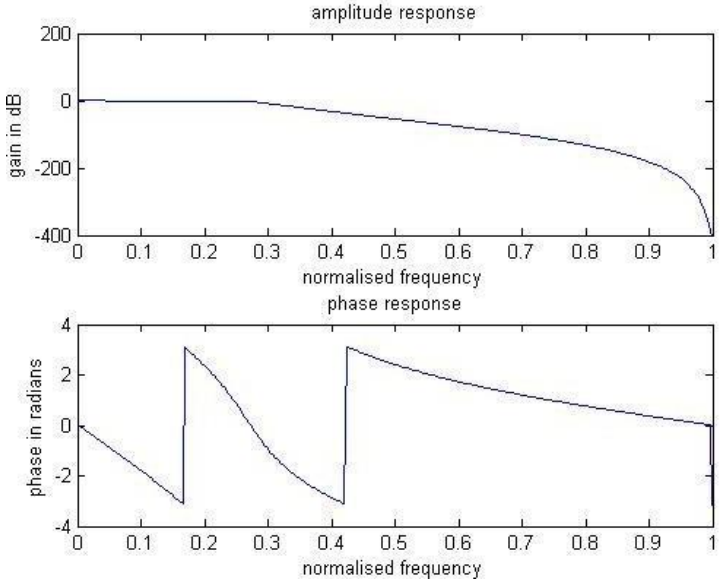
Enter the passband ripple: 0.5

Enter the stopband ripple: 50

Enter the passband frequency: 1200

Enter the stopband frequency: 2400

Enter the sampling frequency: 10000



(b) BUTTERWORTH IIR HPF

AIM

Write a MATLAB program to design IIR Butterworth high-pass filter.

ALGORITHM:

1. Enter the passband ripple rp, stopband ripple rs, passband frequency in Hz, sampling frequency in Hz, stopband frequency in Hz.
2. Compute the frequency in radians.
3. Find cut off frequency and order of filter using command word 'buttord'.
4. Find the coefficients of filter using command word 'butter'.
5. Find the frequency response of filter using the coefficients with command word freqz. Let it be h and omega.
6. Find the magnitude and phase of filter output h.
7. Plot the magnitude and phase response of filter.

PROGRAM

```
clc;
rp=input('Enter the passband ripple:');
rs= input('Enter the stopband ripple:');
wp= input('Enter the passband frequency:');
ws= input('Enter the stopband frequency:');
fs=input('Enter the sampling frequency:');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs);
[b a]=butter(n,wn,'high');
W=[0 :0.01:pi];
[h,omega]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
% plot amplitude response
Subplot(2,1,1);
Plot(omega/pi,m);
```

```
Xlabel('Normalised frequency');
Ylabel('Gain in dB');
Title('Amplitude  response');
% plot phase response
Subplot(2,1,2);
Xlabel('Normalised frequency');
Ylabel('phase in radians');
Title('phase  response');
```

OBSERVATIONS

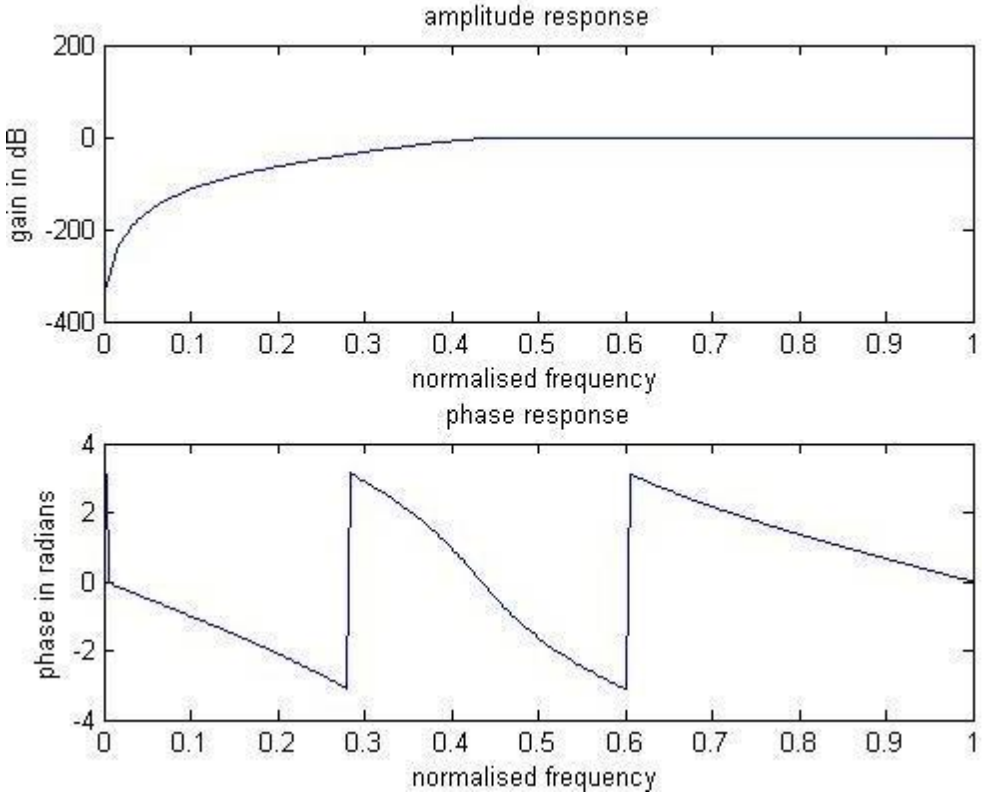
Enter the passband ripple: 0.5

Enter the stopband ripple: 50

Enter the passband frequency: 1200

Enter the stopband frequency: 2400

Enter the sampling frequency: 10000



(c) BUTTERWORTH IIR BPF

.

AIM:

Write a MATLAB program to design IIR Butterworth band-pass filter

ALGORITHM

1. Enter the passband ripple rp, stopband ripple rs, passband frequency in Hz, stopband frequency in Hz, sampling frequency in Hz.
2. Compute the frequency in radian.
3. Find cut-off frequency and order of the filter using command word **'buttord'.**
4. Find the coefficient of filter using the command word **'butter'.**
5. Find the frequency response of filter using the coefficient with command word **'freqz'.** Let it be h & w.
6. Find the magnitude and phase of filter output h
7. Plot the magnitude and phase response of filter

PROGRAM

```
clc;
rp=input('enter the passband ripple:');
rs=input('enter the stopband ripple:');
wp=input('enter the passband frequency:');
ws=input('enter the stopband frequency:');
fs=input('enter the sampling frequency:');
w1=2*wp/fs;
w2=2*ws/fs;
[n]=buttord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=butter(n,wn,'stop');
w=[0:0.01:pi];
[h,omega]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
%plot amplitude response
subplot(2,1,1);
```

```
plot(omega/pi,m);
xlabel('normalised frequency');
ylabel('gain in dB');
title('amplitude response');
%plot phase response
subplot(2,1,2);
plot(omega/pi,an);
xlabel('normalised frequency');
ylabel('phase in radians');
title('phase response');
```

OBSERVATIONS

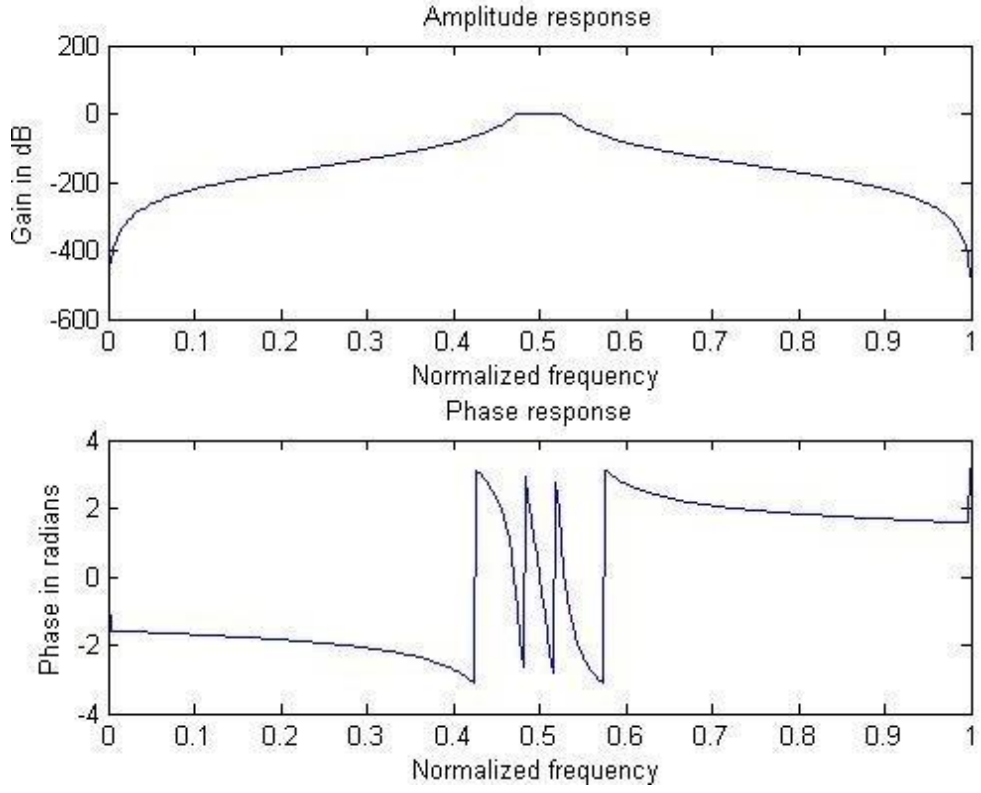
Enter the passband ripple: 0.2

Enter the stopband ripple: 40

Enter the passband frequency: [4800 5200]

Enter the stopband frequency: [4500 5500]

Enter the sampling frequency: 20000



Chebyshev Filters

(a) Chebyshev type1 LPF

AIM

Write a matlab program to study the characteristics of a chebyshev low pass filter

PROGRAM

```
clc;
rp=input('Enter the passband ripple:');
rs=input('Enter the stopband ripple:');
wp=input('Enter the passband frequency:');
ws=input('Enter the stopband frequency:');
fs=input('Enter the sampling frequency:')
w1=2*wp/fs;
w2=2*ws/fs;
[nwn]=cheb1ord(w1,w2,rp,rs);
[b a]=cheby1(n,rp,wn);
w= [0:.01:pi];
[h,omega]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1)
plot(omega/pi,m)
xlabel('Normalized Frequency')
ylabel('Gain in dB')
title('amplitude response')
subplot(2,1,2)
plot(omega/pi,an)
xlabel('Normalized Frequency')
ylabel('Phase in Radians')
title('Phase response')
```

OBSERVATION

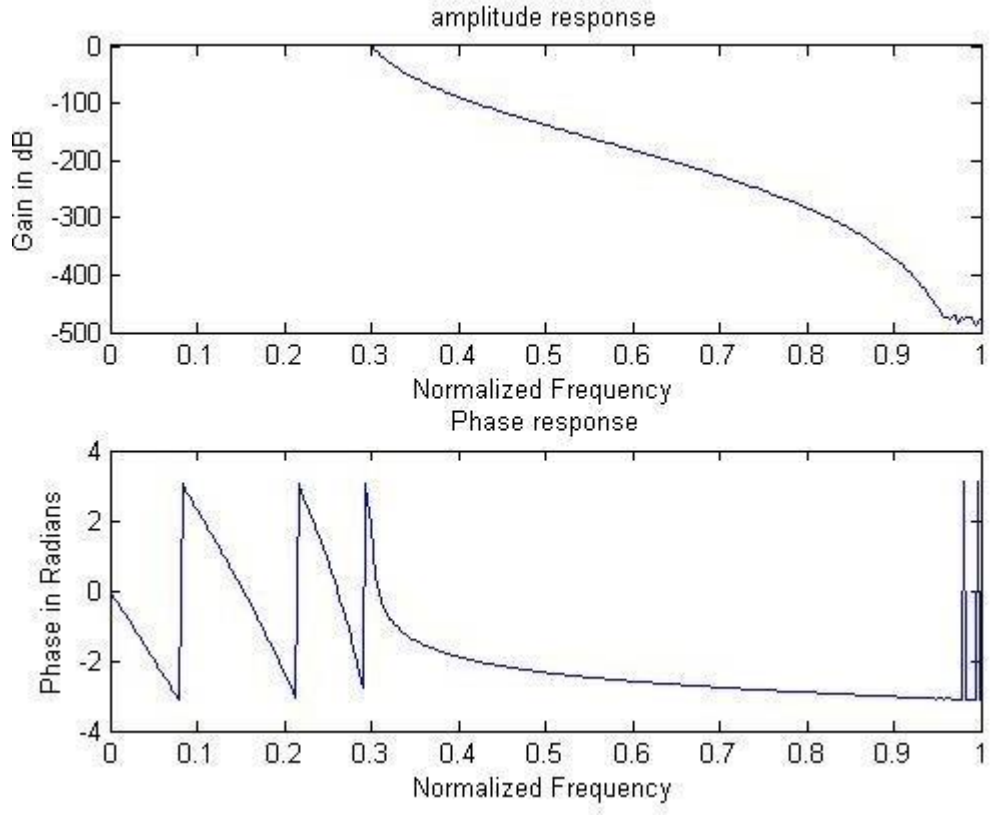
Enter the passband ripple: 0.2

Enter the stopband ripple: 45

Enter the passband frequency: 1500

Enter the stopband frequency: 1700

Enter the sampling frequency: 10000



(b) Chebyshev type1 HPF

AIM

Write a MATLAB program to study the characteristics of a chebyshev high-pass filter.

PROGRAM

```
clc;
rp=input('Enter the passband ripple:');
rs=input('Enter the stopband ripple:');
wp=input('Enter the passband frequency:');
```

```
ws=input('Enter the stopband frequency:');
fs=input('Enter the sampling frequency:')
w1=2*wp/fs;
w2=2*ws/fs;
[nwn]=cheb1ord(w1,w2,rp,rs);
[b a]=cheby1(n,rp,wn,'high');
w=[0:.01:pi];
[h,omega]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1)
plot(omega/pi,m)
xlabel('Normalized Frequency')
ylabel('Gain in dB')
title('amplitude response')
subplot(2,1,2)
plot(omega/pi,an)
xlabel('Normalized Frequency')
ylabel('Phase in Radians')
title('Phase response')
```

OBSERVATION

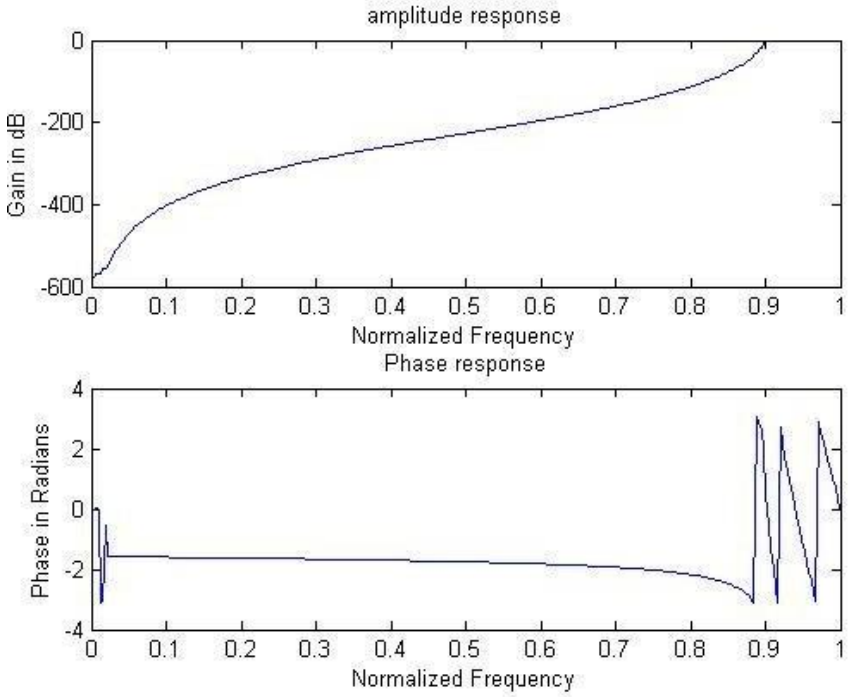
Enter the passband ripple: 0.5

Enter the stopband ripple: 40

Enter the passband frequency: 4500

Enter the stopband frequency: 4400

Enter the sampling frequency: 10000



(c) Chebyshev type1 BPF

AIM

Write a MATLAB program to study the characteristics of a chebyshev band-pass filter

PROGRAM

```
clc;
rp=input('Enter the passband ripple:');
rs=input('Enter the stopband ripple:');
wp=input('Enter the passband frequency:');
ws=input('Enter the stopband frequency:');
fs=input('Enter the sampling frequency:')
w1=2*wp/fs;
w2=2*ws/fs;
[nwn]=cheb1ord(w1,w2,rp,rs);
[b a]=cheby1(n,rp,wn,'bandpass');
w=[0:.01:pi];
[h,omega]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1)
```

```
plot(omega/pi,m)
xlabel('Normalized Frequency')
ylabel('Gain in dB')
title('amplitude response')
subplot(2,1,2)
plot(omega/pi,an)
xlabel('Normalized Frequency')
ylabel('Phase in Radians')
title('Phase response')
```

OBSERVATION

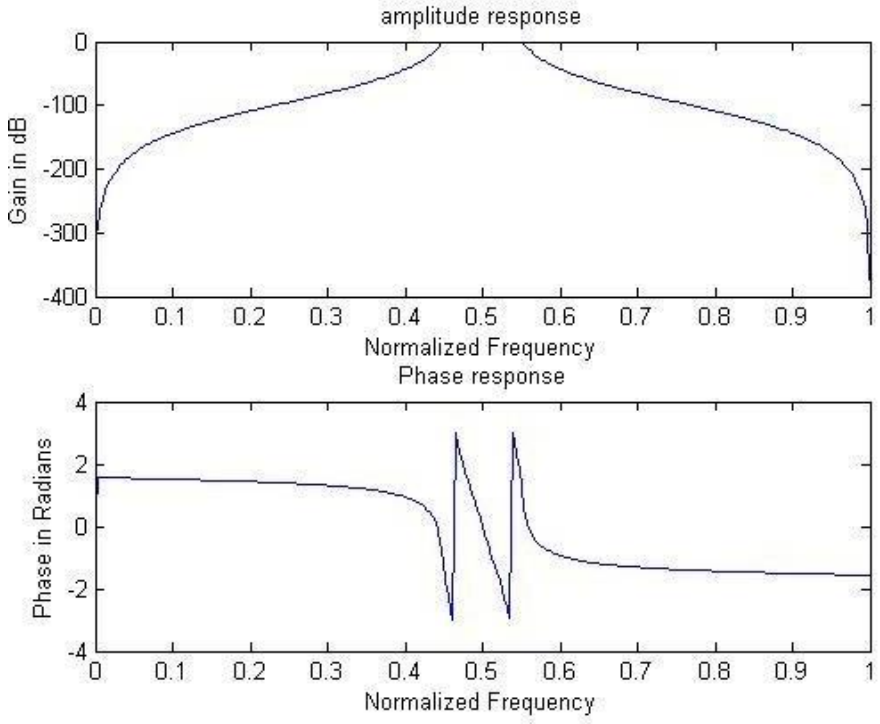
Enter the passband ripple: 0.5

Enter the stopband ripple: 40

Enter the passband frequency: [4500 5500]

Enter the stopband frequency: [4800 5200]

Enter the sampling frequency: 20000



RESULT

Obtained and verified the amplitude and phase response of IIR butterworth HP, LP & BPF.

Exp No 12

FIR Filter -Window Method

AIM

Write a program to design FIR filters using rectangular, Bartlett, hanning and hamming windows.

THEORY

The filters designed by using finite number of samples of impulse response are called FIR filters. These finite number of samples are obtained from the infinite duration desired impulse response hd (n) .Here hd(n) is the inverse Fourier Transform of Hd(ù), where Hd(ù) is the ideal(desired) frequency response. The various methods of designing FIR filters differs only in the method of determining the samples of h(n) from the samples of hd(n).

Various steps in designing FIR filters

1. Choose an ideal (desired) frequency response, Hd(ù).
2. Take inverse Fourier Transform of Hd(ù) to get hd(n) or sample Hd(ù) at finite number of points (N-point) to get H(k).
3. If hd(n) is determined then convert the infinite duration hd(n) to a finite duration h(n),(usually h(n) is an N-point sequence) or if H(k) is determined then take N-point inverse DFT to get h(n).
4. Take Z-transform of h(n) to get H(z), where H(z) is the transfer function of the digital filter.
5. Choose a suitable structure and realize the filter.
6. Verify the design by simulation.

Advantages of FIR filters

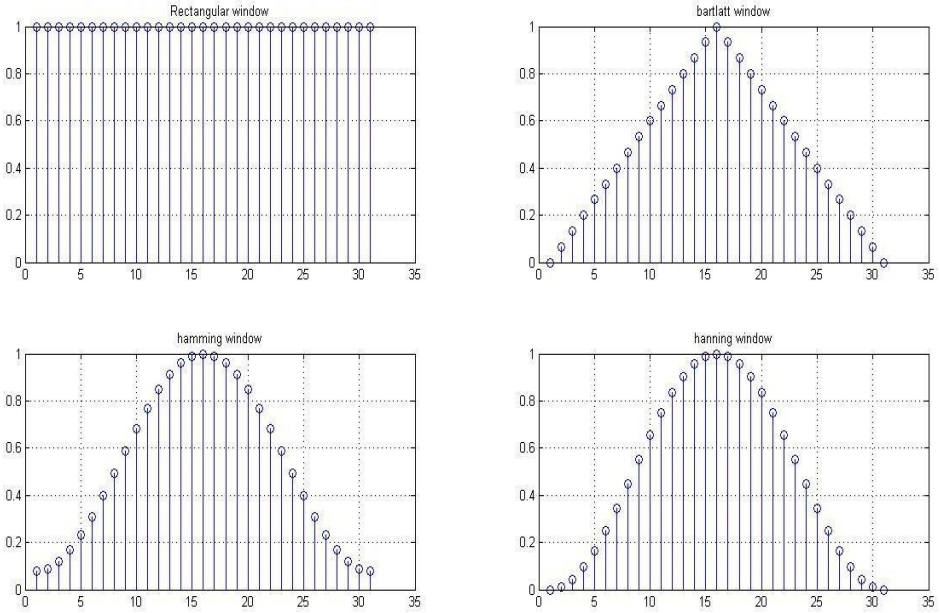
1. FIR filters with exactly linear phase can be easily designed.
2. Efficient realization of FIR filter exist as both recursive and non-recursive structures.
3. FIR filters realized non-recursively i.e., by direct convolution are always stable.
4. Round off noise, which is inherent in realizations with finite precision arithmetic can easily be made small for non-recursive realization of FIR filters.

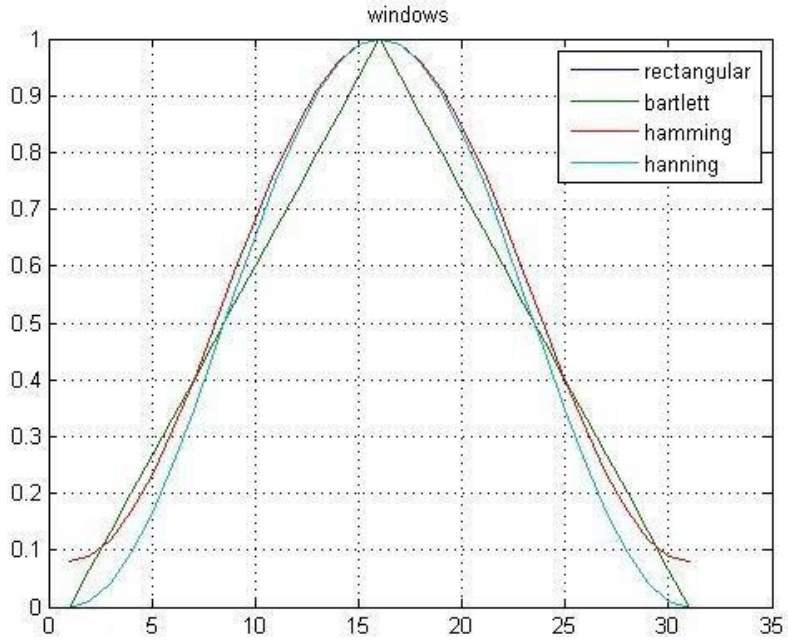
PROGRAM

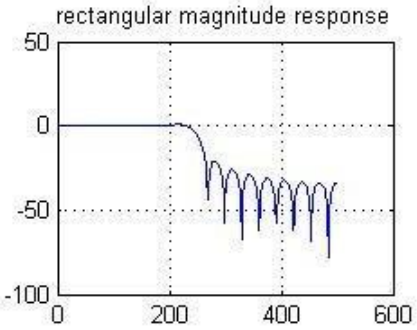
```
clc;
clear all;
n=30;wn=0.5;
w_rect=rectwin(n+1);
b1=fir1(n,wn,w_rect);
[h1 w1]=freqz(b1,1,512,1000);
w_bar=bartlett(n+1);
b2=fir1(n,wn,w_bar);
[h2 w2]=freqz(b2,1,512,1000);
w_ham=hamming(n+1);
b3=fir1(n,wn,w_ham);
[h3 w3]=freqz(b3,1,512,1000);
w_hann=hann(n+1);
b4=fir1(n,wn,w_hann);
[h4 w4]=freqz(b4,1,512,1000);
window_t=[w_rectw_barw_hamw_hann];
figure(1);
subplot(221);
stem(w_rect);
title('Rectangular window');
grid
subplot(222);
stem(w_bar);
title('bartlatt window');
grid
subplot(223);
stem(w_ham);
title('hamming window');
grid
subplot(224);
stem(w_hann);
title('hanning window');
```

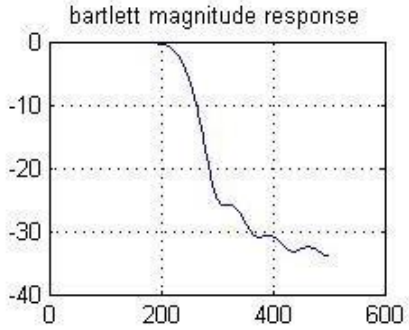
```
grid
figure(2);
plot(window_t);
legend('Rectengular','barlett','hamming','hanning');

title('Windows');
grid;
%figure(3);
%subplot(221);
%plot(w1,20*log10(abs(h1)));
```

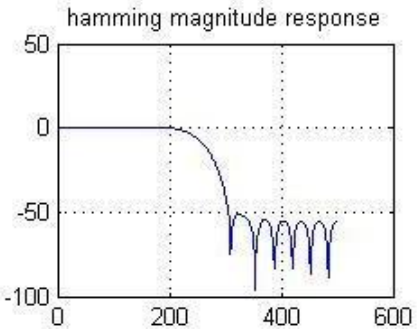
OUTPUT

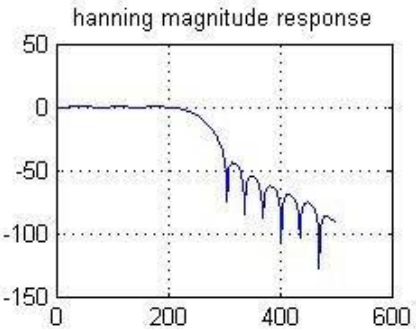


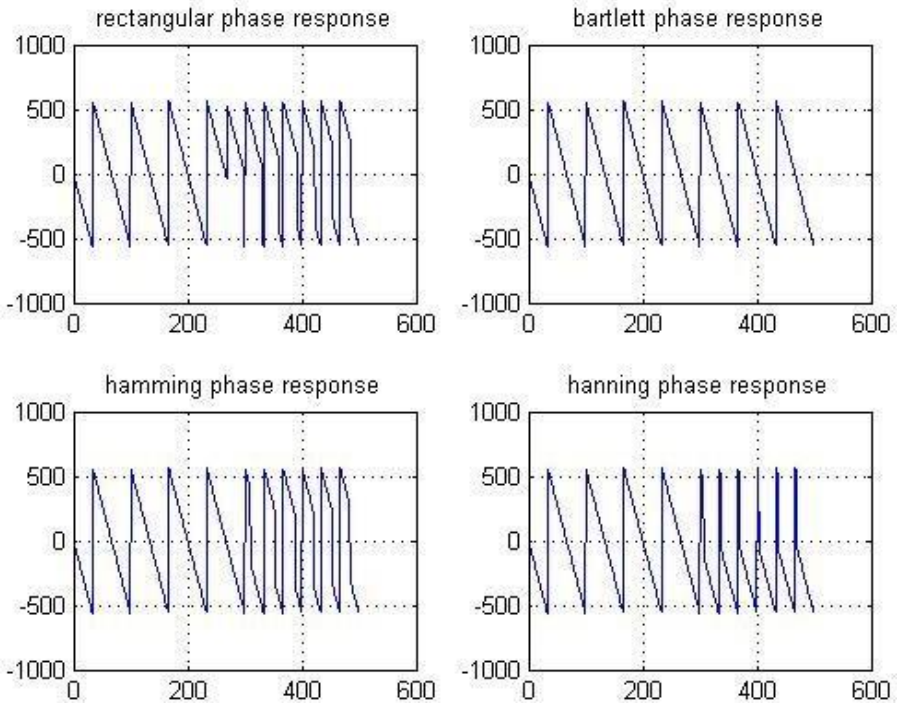

windows
rectangular
bartlett
hamming
hanning


rectangular magnitude response


bartlett magnitude response


hamming magnitude response


hanning magnitude response



RESULT

Wrote and executed a MATLAB program to generate FIR filter response using rectangular, hamming, hanning and Bartlett windows.