**Discrete time signals**

**Unit Impulse sequence**



Command window output

>> impulseseq(1,5,4)

ans =

1×5 logical array

0 0 0 1 0

Code

function[y,n]=impulseseq(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which impulse exists

% a,b,c must be integers for discrete signals

n=a:b;

y=(n-c)==0;

stem(n,y)

% save this as impulseseq.m

% dont run the program

% run in command window by calling the function as impulseseq(a,b,c) give

% values of a,b,c in function call

% example : impulseseq(1,5,4)

**To generate impulse sequence of amplitude ‘A’**



**Program to generate impulse sequence of amplitude ‘A’**

Command Window

>> impulseseqofamplitudeA(1,5,4)

ans =

1×5 logical array

0 0 0 1 0

Code

function[y,n]=impulseseqofamplitudeA(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which impulse exists

% a,b,c must be integers for discrete signals

n=a:b;

y=(n-c)==0;

A=2;

stem(n,A\*y)

% save this as impulseseqofamplitudeA.m

% dont run the program

% run in command window by calling the function as impulseseq(a,b,c) give

% values of a,b,c in function call

% example : impulseseq(1,5,4)

**Plot Delta(n) for -5<=n and n<=5**

Code

function[y,n]=deltaofn()

% a and b are the lower and upper time limit

% c is the instant at which impulse exists

% a,b,c must be integers for discrete signals

n=-5:5; % n=a:b;

y=(n-0)==0; % y=(n-c)==0

stem(n,y)

% c=0 for delta(n)

Command window

>> deltaofn()

ans =

1×11 logical array

0 0 0 0 0 1 0 0 0 0 0



**Plot delta(n+3) for -4<=n and n<=6**

****

Code

function[y,n]=deltaofnplus3()

% a and b are the lower and upper time limit

% c is the instant at which impulse exists

% a,b,c must be integers for discrete signals

n=-4:6; % n=a:b;

y=(n+3)==0; % y=(n-c)==0

stem(n,y)

% c=-3 for delta(n+3)

Command window

>> deltaofnplus3()

ans =

1×11 logical array

0 1 0 0 0 0 0 0 0 0 0

**Plot delta(n-2) for -5<=n and n<=7**



Command window

>> deltaofnminus2()

ans =

1×13 logical array

0 0 0 0 0 0 0 1 0 0 0 0 0

Code

function[y,n]=deltaofnminus2()

% a and b are the lower and upper time limit

% c is the instant at which impulse exists

% a,b,c must be integers for discrete signals

n=-5:7; % n=a:b;

y=(n-2)==0; % y=(n-c)==0

stem(n,y)

% c=2 for delta(n-2)

**Unit Step Sequence**



Code

function[y,n]=stepseq(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which step exists

% a,b,c must be integers for discrete signals

n=a:b;

y=(n-c)>=0;

stem(n,y)

Command Window

>> stepseq(1,10,5)

ans =

1×10 logical array

0 0 0 0 1 1 1 1 1 1

**Program to generate step sequence of amplitude ‘A’**

****

Code

function[y,n]=stepseqofAmplitudeA(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which step exists

% a,b,c must be integers for discrete signals

n=a:b;

A=3;

y=(n-c)>=0;

stem(n,A\*y)

Command Window

>> stepseqofAmplitudeA(1,8,3)

ans =

1×8 logical array

0 0 1 1 1 1 1 1

**Plot u(n) for -5<=n and n<=5**

****

Code

function[y,n]=uofn()

% a and b are the lower and upper time limit

% c is the instant at which step exists

% a,b,c must be integers for discrete signals

n=-5:5; %n=a:b;

y=(n-0)>=0; %y=(n-c)>=0;

stem(n,y)

Command Window

>> uofn()

ans =

1×11 logical array

0 0 0 0 0 1 1 1 1 1 1

**Plot u(n+2) for -4<=n and n<=6**

****

Code

function[y,n]=uofnplus2()

% a and b are the lower and upper time limit

% c is the instant at which step exists

% a,b,c must be integers for discrete signals

n=-4:6; %n=a:b;

y=(n+2)>=0; %y=(n-c)>=0;

stem(n,y)

Command Window

>> uofnplus2()

ans =

1×11 logical array

0 0 1 1 1 1 1 1 1 1 1

**Plot u(n-1) for -5<=n and n<=7**

****

Code

function[y,n]=uofnminus1()

% a and b are the lower and upper time limit

% c is the instant at which step exists

% a,b,c must be integers for discrete signals

n=-5:7; %n=a:b;

y=(n-1)>=0; %y=(n-c)>=0;

stem(n,y)

Command Window

>> uofnminus1()

ans =

1×13 logical array

0 0 0 0 0 0 1 1 1 1 1 1 1

**Unit Ramp sequence**



Code

function[y,n]=rampseq(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which ramp exists

% a,b,c must be integers for discrete signals

n=a:b;

y=((n-c)>=0).\*(n-c);

stem(n,y)

Command Window

>> rampseq(1,10,0)

ans =

1 2 3 4 5 6 7 8 9 10

**Program to generate negative ramp sequence of amplitude ‘A’**

****

Code

function[y,n]=NegativeRampSeqofAmplitudeA(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which ramp exists

% a,b,c must be integers for discrete signals

n=a:b;

A=2;

y=((n-c)>=0).\*(n-c);

stem(n,-(A\*y))

Command Window

>> NegativeRampSeqofAmplitudeA(-5,5,0)

ans =

0 0 0 0 0 0 1 2 3 4 5

**Plot r(n) for -5<=n and n<=5**

****

Code

function[y,n]=rofn()

% a and b are the lower and upper time limit

% c is the instant at which ramp exists

% a,b,c must be integers for discrete signals

n=-5:5; %n=a:b;

y=((n-0)>=0).\*(n-0); %y=((n-c)>=0).\*(n-c);

stem(n,y)

Command Window

>> rofn()

ans =

0 0 0 0 0 0 1 2 3 4 5

**Plot r(n+2) for -4<=n and n<=6**

****

Code

function[y,n]=rofnplus2()

% a and b are the lower and upper time limit

% c is the instant at which ramp exists

% a,b,c must be integers for discrete signals

n=-4:6; %n=a:b;

y=((n+2)>=0).\*(n+2); %y=((n-c)>=0).\*(n-c);

stem(n,y)

Command Window

>> rofnplus2()

ans =

0 0 0 1 2 3 4 5 6 7 8

**Plot r(n-1) for -5<=n and n<=7**

****

Code

function[y,n]=rofnminus1()

% a and b are the lower and upper time limit

% c is the instant at which ramp exists

% a,b,c must be integers for discrete signals

n=-5:7; %n=a:b;

y=((n-1)>=0).\*(n-1); %y=((n-c)>=0).\*(n-c);

stem(n,y)

Command Window

>> rofnminus1()

ans =

0 0 0 0 0 0 0 1 2 3 4 5 6

RealExponentialSequence

CODE

function[y,n]=RealExponentialSequence(a,b,c)

% RealExponentialSequence ( a to the power of n)

% b and c are the lower and upper time limit

% a is a constant

n=b:c;

y=a.^n;

stem(n,y)

Command window

>> RealExponentialSequence(0.5,-10,10)

ans =

1.0e+03 \*

Columns 1 through 10

1.0240 0.5120 0.2560 0.1280 0.0640 0.0320 0.0160 0.0080 0.0040 0.0020

Columns 11 through 20

0.0010 0.0005 0.0003 0.0001 0.0001 0.0000 0.0000 0.0000 0.0000 0.0000

Column 21

0.0000



Code

function[y,n]=expseq1()

n=-5:5;

y=(0.9).^n;

stem(n,y)

Command Window

>> expseq1

ans =

Columns 1 through 10

1.6935 1.5242 1.3717 1.2346 1.1111 1.0000 0.9000 0.8100 0.7290 0.6561

Column 11

0.5905



function[y,n]=expseq2()

n=-4:6;

y=(1.1).^n;

stem(n,y)

>> expseq2

ans =

Columns 1 through 10

0.6830 0.7513 0.8264 0.9091 1.0000 1.1000 1.2100 1.3310 1.4641 1.6105

Column 11

1.7716



5. Complex Exponential Sequence (e^(jwn))

% complex exponential with w=0.3 radians

% script file

n=-4:4;

y=exp(j\*0.3\*n);

subplot(2,1,1);

stem(n,abs(y));

subplot(2,1,2);

stem(n,angle(y));

% phase is in radians and within + or - pi

>> ComplexExponentialSequence



% complex exponential with w=0.3 radians

% script file

n=-5:5;

y=exp((1+(j\*0.3))\*n);

subplot(2,1,1);

stem(n,abs(y));

subplot(2,1,2);

stem(n,angle(y));

% phase is in radians and within + or - pi

>> ComplexExponentialSequence2



% sinusoidal signal of f=0.1 cycle per sample (or N=10)

n=-10:10;

y=sin(2\*0.1\*pi\*n);

stem(n,y);

>> SinusoidalSequence



% sinusoidal signal of f=0.1 cycle per sample (or N=10)

% phase delay of 36degrees

n=-10:10;

y=sin((2\*0.1\*pi\*n)+(pi\*0.2));

stem(n,y);

>> SinusoidalSequenceWithPhaseDelay



% cosine signal of f=0.2 cycle per sample

n=-10:10;

y=sin((2\*0.2\*pi\*n)-(0.5\*pi));

stem(n,y);

>> CosineSignal



Continuous time signals

1.Step signal



function[y,t]=stepsig(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which impulse exists

% a,b,c are real numbers

t=a:0.01:b;

y=(t-c)>=0;

plot(t,y)

>> stepsig(-10,10,0)

ans =

1×2001 logical array

Columns 1 through 25

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 26 through 50

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 51 through 75

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 76 through 100

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 101 through 125

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 126 through 150

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 151 through 175

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 176 through 200

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 201 through 225

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 226 through 250

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 251 through 275

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 276 through 300

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 301 through 325

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 326 through 350

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 351 through 375

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 376 through 400

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 401 through 425

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 426 through 450

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 451 through 475

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 476 through 500

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 501 through 525

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 526 through 550

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 551 through 575

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 576 through 600

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 601 through 625

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 626 through 650

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 651 through 675

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 676 through 700

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 701 through 725

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 726 through 750

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 751 through 775

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 776 through 800

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 801 through 825

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 826 through 850

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 851 through 875

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 876 through 900

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 901 through 925

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 926 through 950

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 951 through 975

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 976 through 1000

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 1001 through 1025

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1026 through 1050

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1051 through 1075

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1076 through 1100

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1101 through 1125

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1126 through 1150

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1151 through 1175

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1176 through 1200

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1201 through 1225

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1226 through 1250

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1251 through 1275

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1276 through 1300

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1301 through 1325

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1326 through 1350

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1351 through 1375

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1376 through 1400

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1401 through 1425

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1426 through 1450

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1451 through 1475

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1476 through 1500

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1501 through 1525

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1526 through 1550

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1551 through 1575

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1576 through 1600

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1601 through 1625

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1626 through 1650

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1651 through 1675

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1676 through 1700

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1701 through 1725

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1726 through 1750

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1751 through 1775

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1776 through 1800

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1801 through 1825

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1826 through 1850

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1851 through 1875

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1876 through 1900

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1901 through 1925

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1926 through 1950

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1951 through 1975

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 1976 through 2000

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Column 2001

1



function[y,t]=stepsig1()

t=-10:0.01:10;

y=(t-0)>=0;

plot(t,y);

2. Ramp signal

function[y,t]=rampsig(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which ramp exists

% a,b,c are real numbers

t=a:0.1:b;

y=(((t-c)>=0).\*(t-c));

plot(t,y);



3. Real Exponential signal (e^(a\*t))

function[y,t]=expsig(a,b,c)

% b and c are the lower and upper time limit

% a is a constant

t=b:0.01:c;

y=exp(a\*t);

plot(t,y)



function[y,t]=expsig1()

t=0:0.01:10;

y=exp(2\*t);

plot(t,y)

Complex Exponential signal e^(j\*w\*t)

% complex exponential with f=5 Hz

% script file

t=-4:0.01:4;

y=exp(j\*2\*pi\*5\*t);

subplot(2,1,1);

plot(t,abs(y));

subplot(2,1,2);

plot(t,angle(y));

% phase is in radians and with in + or - pi



5. Sinusoidal Signal



% sinusoidal signal of f=5 Hz (or T=0.2 sec)

t=-0.5:0.01:0.5;

y=sin(2\*0.5\*pi\*t);

plot(t,y);

the above 5 for discrete time signal

1. Step signal

function[y,t]=stepsigDTS(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which step exists

% a,b,c are real numbers

t=a:0.1:b;

y=(t-c)>=0;

stem(t,y)

function[y,t]=stepsig1DTS()

t=-10:0.1:10;

y=(t-0)>=0;

stem(t,y);



1. Ramp signal

function[y,t]=rampsigDTS(a,b,c)

% a and b are the lower and upper time limit

% c is the instant at which ramp exists

% a,b,c are real numbers

t=a:0.1:b;

y=(((t-c)>=0).\*(t-c));

stem(t,y)

function[y,t]=rampsig1DTS()

t=-10:0.1:10;

y=(((t-0)>=0).\*(t-0));

stem(t,y);



1. Real Exponential Signal

function[y,t]=expsigDTS(a,b,c)

% b and c are the lower and upper time limit

% a is a constant

t=b:0.01:c;

y=exp(a\*t);

stem(t,y)

function[y,t]=expsig1DTS()

t=0:0.1:10;

y=exp(2\*t);

stem(t,y)



1. Complex Exponential Signal

% complex exponential with f=5 Hz

% script file

t=-4:0.1:4;

y=exp(j\*2\*pi\*5\*t);

subplot(2,1,1);

stem(t,abs(y));

subplot(2,1,2);

stem(t,angle(y));

% phase is in radians and with in + or - pi



1. Sinusoidal Signal

% sinusoidal signal of f=5 Hz (or T=0.2 sec)

t=-0.5:0.01:0.5;

y=sin(2\*0.5\*pi\*t);

stem(t,y);



To generate square and triangular waves by using the functions available in matlab











Code to generate the overdamped or underdamped sinusoidal signal

function[y,t]=damp1

b=input('Enter starting time instant: ');

c=input('Enter ending time instant: ');

t=b:0.01:c;

f=input('Enter the damped frequency of oscillations: ')

y=sin(2\*pi\*f\*t);

a=input('Enter a positive exponent for overdamped signal.\n Enter a negative exponent for underdamped signal. ')

m=(expsig(a,b,c)).\*y;

plot(t,m)





Select wave type generation through keyboard by giving suitable data

clc;

clear all;

close all;

t=0:0.001:0.3

choice=input(' 1.Square Wave\n 2.Rectangular Wave\n 3.Symmetric Triangular Wave\n 4.Negative Sawtooth Wave\n 5.Positive Sawtooth Wave\n Enter Choice !!')

if(choice==1)

f1=input('Enter the frequency of square wave,f1=')

sq\_wave1=square(2\*pi\*f1\*t); % square wave of 50% duty cycle

figure;

plot(t,sq\_wave1);

elseif(choice==2)

f2=input('Enter the frequency of Rectangular wave,f2=')

d=input('Enter duty cycle with the range (0,100) (except 50)')

sq\_wave2=square(2\*pi\*f2\*t,d); % square wave of d% duty cycle

figure;

plot(t,sq\_wave2);

elseif(choice==3)

f3=input('Enter the frequency of triangular wave,f3=')

tri\_wave1=sawtooth(2\*pi\*f3\*t,0.5); % symmetric triangular wave

figure;

plot(t,tri\_wave1);

elseif(choice==4)

f4=input('Enter the frequency of Negative sawtooth wave,f4=')

saw\_tooth1=sawtooth(2\*pi\*f4\*t,0); % negative sawtooth wave

figure;

plot(t,saw\_tooth1);

elseif(choice==5)

f5=input('Enter the frequency of Positive sawtooth wave,f5=')

saw\_tooth2=sawtooth(2\*pi\*f5\*t,1); % positive sawtooth wave

figure;

plot(t,saw\_tooth2);

else

print(' Wrong Choice ... ')

end

