|  |
| --- |
| **EE-384: Digital Signal Processing Spring 2020**  Laboratory 2: Discrete-time Signals in MATLAB  *Instructor: Mr Ammar Naseer* *EE UET New Campus* |

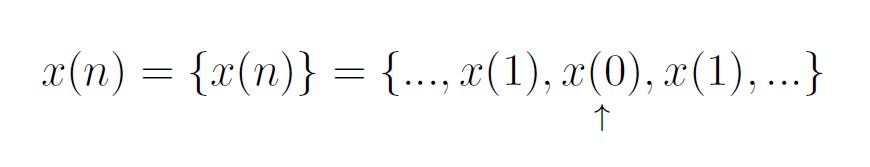
# Aims

We begin with the concepts of signals in discrete time. A number of important types of signals and their operations are introduced. The emphasis in this chapter is on the representations and implementation of signals using MATLAB.

**Pre-Lab:**

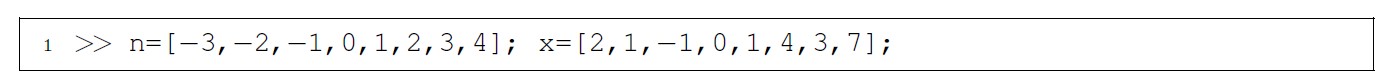
## Discrete time Signals

Signals are broadly classified into analog and discrete signals. An analog signal will be denoted by x(t), in which the variable t can represent any physical quantity, but we will assume that it represents time in seconds. A discrete signal will be denoted by x(n), in which the variable n is integer-valued and represents discrete instances in time. Therefore it is also called a discrete-time signal, which is a number sequence and will be denoted by one of the following notations.



where the up-arrow indicates the sample at n = 0.

In MATLAB we can represent a finite-duration sequence by a row vector of appropriate values. However, such a vector does not have any information about sample position n. Therefore a correct representation of x(n) would require two vectors, one each for x and n. For example, a sequence x(n) = [2 1 -1 0 1 4 3 7] can be represented in MATLAB by



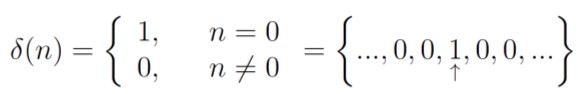
Generally, we will use the x-vector representation alone when the sample position information is not required or when such information is trivial (e.g. when the sequence begins at n = 0). An arbitrary infinite-duration sequence cannot be represented in MATLAB due to the finite memory limitations.

Page 1

## Type of Sequences

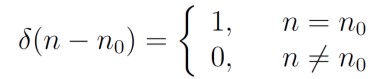
**2.A** We use several elementary sequences in digital signal processing for analysis purposes. Their definitions and MATLAB representations follow.

### Unit Sample Sequence

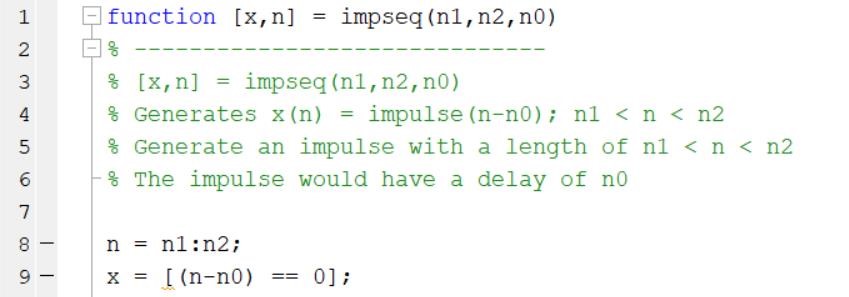


In MATLAB the function zeros (1,N) generates a row vector of N zeros, which can be used to implement

(n) over a finite interval. However, the logical relation n==0 is an elegant way of implementing (n). For example, to implement



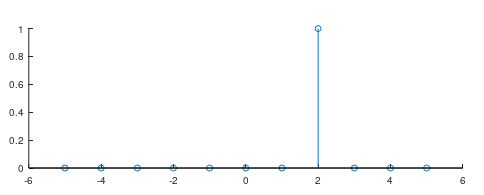
over the n1 < n0 < n2 interval, we will use the following MATLAB function



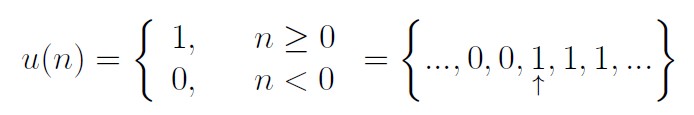
For example, to generate  ; -5 ≤ n ≤ 5, we will need the following MATLAB script:

>> [x,n] = impseq(-5,5,2);

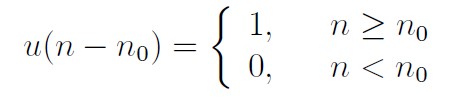
>> stem(n,x)



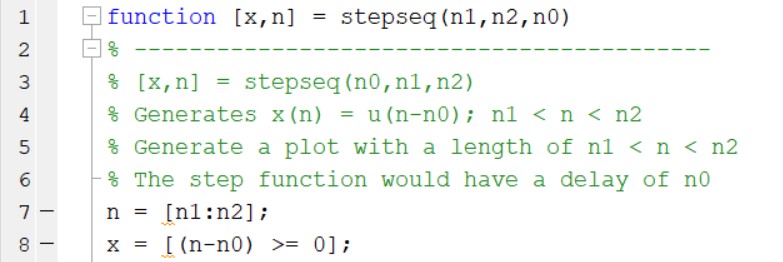
### Unit Step Sequence



In MATLAB the function ones(1,N) generates a row vector of N ones. It can be used to generate u(n) over a finite interval. Once again an elegant approach is to use the logical relation n>=0. To implement



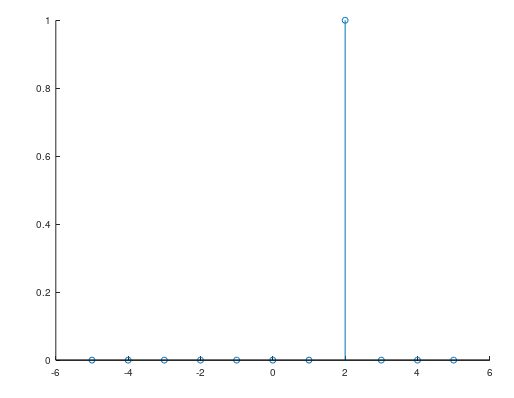
over the n1 < n0 < n2 interval, we will use the following MATLAB function



For example, to generate ; -5 ≤ n ≤ 5, we will need the following MATLAB script:

>> [x,n] = stepseq(-5,5,-1);

>> stem(n,x)



### Real-valued exponential sequence

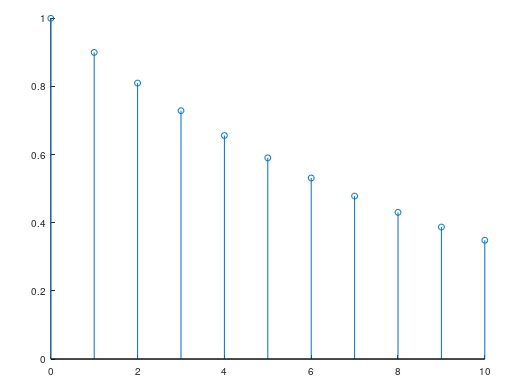


In MATLAB an array operator "." is required to implement a real exponential sequence.

For example, to generate x(n) = (0.9)n; 0 ≤ n ≤ 10, we will need the following MATLAB script:

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

>> stem(n,x)



### Complex-valued exponential sequence

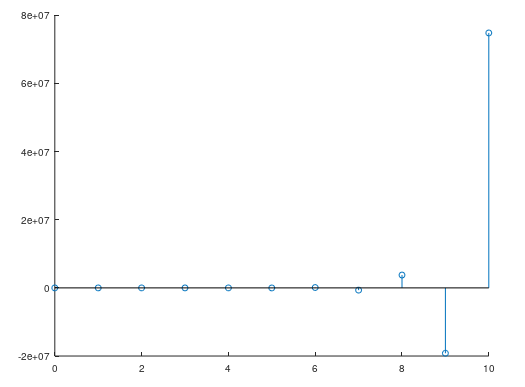


Where  produces an attenuation (if < 0) or amplification (if > 0) and  is the frequency in radians. A MATLAB function **exp** is used to generate exponential sequences.

For example, to generate x(n) = exp[(2+j3)n], 0 ≤ n ≤ 10, we will need the following MATLAB script:

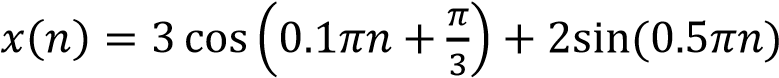
>> n = [0:10];x = exp((2+3j)\*n);

>> stem(n,x)



### Sinusoidal sequence

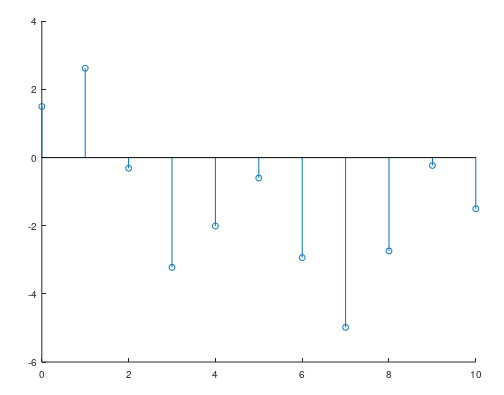


where A is an amplitude and  is the phase in radians. A MATLAB function cos (or sin) is used to generate sinusoidal sequences. For example, to generate .

0<n<10, we will need the following MATLAB script:

>> n = [0:10]; x = 3\*cos(0.1\*pi\*n+pi/3) + 2\*sin(0.5\*pi\*n);

>> stem(n,x)



### Periodic Sequence

A sequence x(n) is periodic if x(n) = x(n+N); n. The smallest integer N that satisfies this relation is called the fundamental period. We will use (n) to denote a periodic sequence. To generate P periods of  (n) from one period {x(n), 0<n<N1}, we can copy x(n) P times:

>> xtilde = [x,x,...,x];

But an elegant approach is to use MATLABs powerful indexing capabilities. First we generate a matrix containing P rows of x(n) values. Then we can concatenate P rows into a long row vector using the construct (:). However, this construct works only on columns. Hence we will have to use the matrix transposition operator ' to provide the same effect on rows.

>> xtilde = x' \* ones(1,P); % P columns of x; x is a row vector

>> xtilde = xtilde(:); % long column vector

>> xtilde = xtilde'; % long row vector

Note that the last two lines can be combined into one for compact coding.

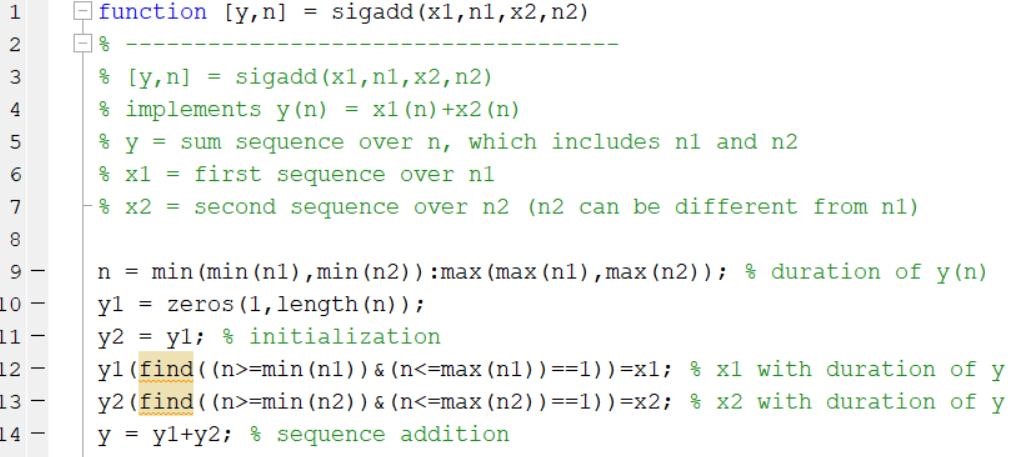
**Operation on sequence**

**2.B** Here we briefly describe basic sequence operations and their MATLAB equivalents.

**1. Signal addition**: This is a sample-by-sample addition given by



It is implemented in MATLAB by the arithmetic operator +. However, the lengths of x1(n) and x2(n) must be the same. If sequences are of unequal lengths, or if the sample positions are different for equal-length sequences, then we cannot directly use the operator +. We have to first augment x1(n) and x2(n) so that they have the same position vector n (and hence the same length). This requires careful attention to MATLABs indexing operations. In particular, logical operation of intersection **&**, relational operations like ;= and ==, and the **find** function are required to make x1(n) and x2(n) of equal length. The following function, called the sigadd function, demonstrates these operations.



For example, to generate  + ; -5 ≤ n ≤ 5, we will need the following

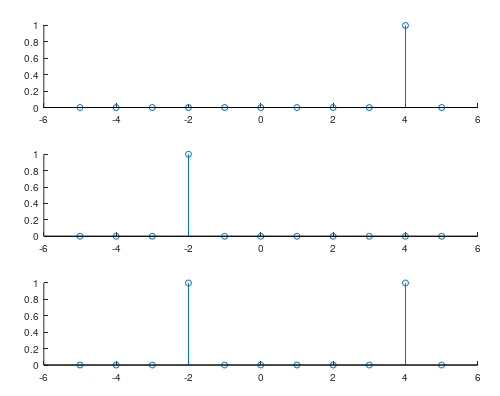
MATLAB script:

>> [x1,n1] = impseq(-5,5,4);

>> [x2,n2] = impseq(-5,5,-2);

>> [x,n] = sigadd(x1,n1,x2,n2);

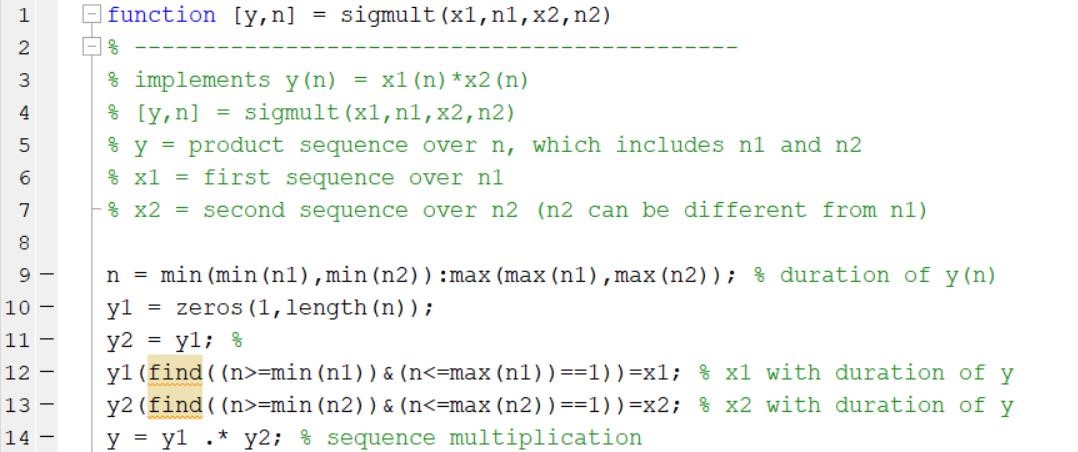
>> stem(n,x)



1. **Signal Multiplications:** This is a sample-by-sample (or "dot") multiplication) given by



It is implemented in MATLAB by the array operator ” \* ”. Once again, the similar restrictions apply for the “ .\* operator as for the + operator”. Therefore we have developed the sigmult function, which is similar to the sigadd function.



For example, to generate  \* ; -5 ≤ n ≤ 5, we will need the following

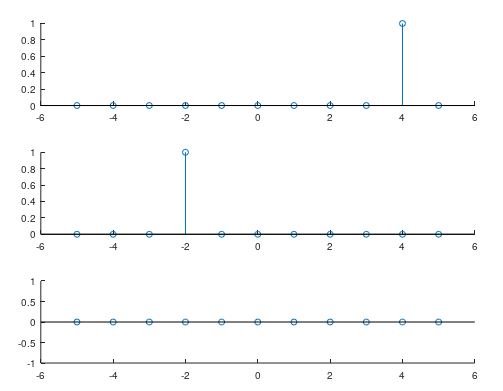
MATLAB script:

>> [x1,n1] = impseq(-5,5,4);

>> [x2,n2] = impseq(-5,5,-2);

>> [x,n] = sigmult(x1,n1,x2,n2);

>> stem(n,x)



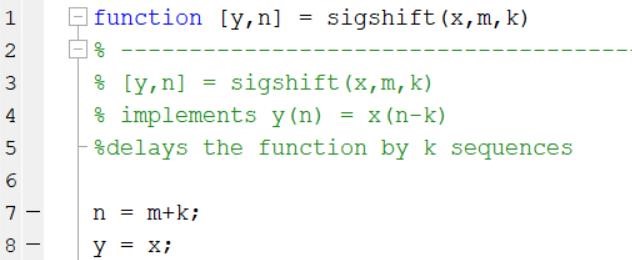
1. **Shifting** In this operation, each sample of x(n) is shifted by an amount k to obtain a shifted sequence y(n).



If we let m = nk, then n = m + k and the above operation is given by



Hence this operation has no effect on the vector x, but the vector n is changed by adding k to each element. This is shown in the function **sigshift**.



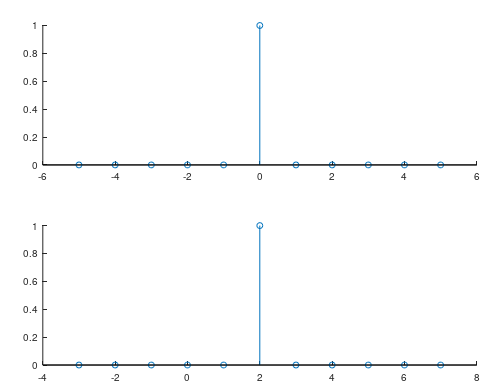
For example, to generate  ; -3 ≤ n ≤ 7, we can also write the following MATLAB script:

>> [x,n] = impseq(-5,5,0);

>> stem(n,x)

>> [y,n] = sigshift(x,n,2);

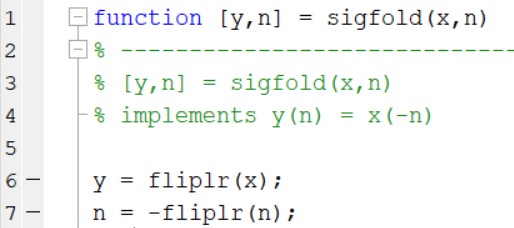
>> stem(n,y)



1. **Folding** In this operation each sample of x(n) is flipped around n = 0 to obtain a folded sequence y(n).



In MATLAB this operation is implemented by **fliplr(x)** function for sample values and by **-fliplr(n)** function for sample positions as shown in the **sigfold** function.



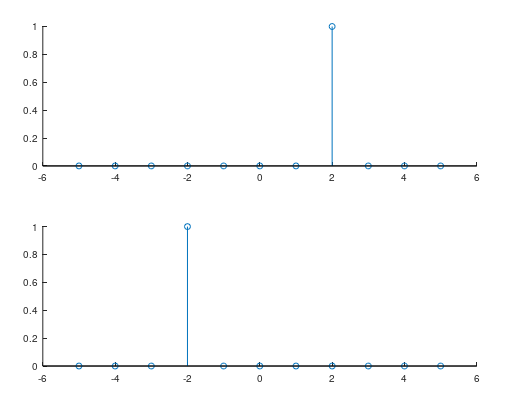
For example, to generate  ; -5 ≤ n ≤ 5, we can also write the following MATLAB script:

>> [x,n] = impseq(-5,5,2);

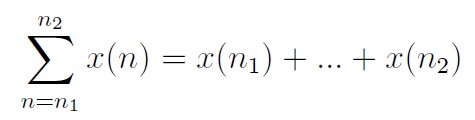
>> stem(n,x)

>> [y,n] = sigfold(x,n);

>> stem(n,y)



1. **Sample summation**. This operation differs from signal addition operation. It adds all sample values of x(n) between n1 and n2.



It is implemented by the **sum(x(n1:n2))** function.

**CODE:**

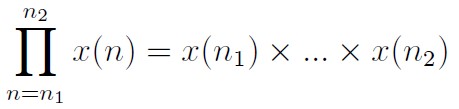
>> [x,n] = stepseq (5,0,10);

>> sum(x(2:7))

ans = 2

>>

1. **Sample Product:** This operation also differs from signal multiplication operation. It multiplies all sample values of x(n) between n1 and n2.



It is implemented by the **prod(x(n1:n2))** function.

**CODE:**

>> c=[0 1 2 3 4 5]

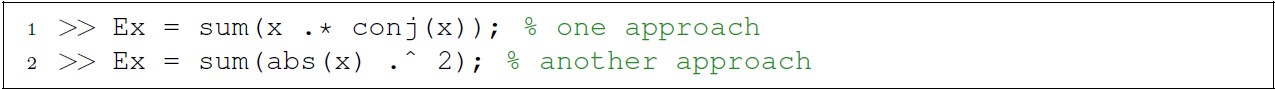
>> prod(c(2:5))

ans = 24

>>

1. **Signal energy.** The energy of a sequence x(n) is given by

 where superscript \* denotes the operation of complex conjugation. The energy of a finite-duration sequence x(n) can be computed in MATLAB using



**CODE:**

# >> x=[1 2 3 4 5];

# >> Ex=sum(x.\*conj(x));

# >> Ex

# Ex = 55

# >> Ex=sum(abs(x).^2)

# Ex = 55

# >>

# Main Lab

**3A:** Generate and plot each of the following sequences over the indicated interval.

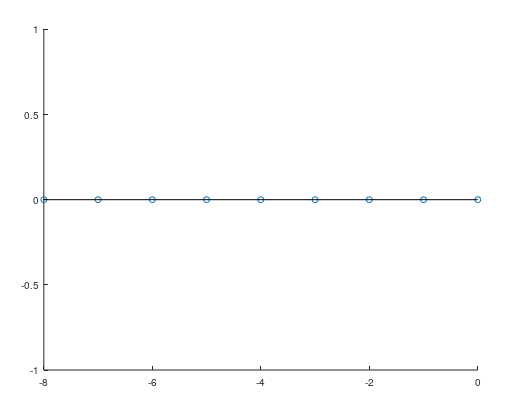
1. , 

**CODE:**

>> n=[-8:0];

>> x=3\*impseq(-8,0,3)-impseq(-8,0,6);

>> stem(n,x)



1. , 

**CODE:**

>> n=[0:20];

>> x1=n.\*(stepseq(0,0,20)-stepseq(10,0,20));

>> x2=10\*exp(-0.3\*(n-10)).\*(stepseq(10,0,20)-stepseq(20,0,20));

>> [x,n1]=sigadd(x1,n,x2,n);

>> subplot(3,1,1);

>> stem(n,x1);

>> axis([-10 20 -10 10])

>> grid on;

>> subplot(3,1,2);

>> stem(n,x2);

>> grid on;

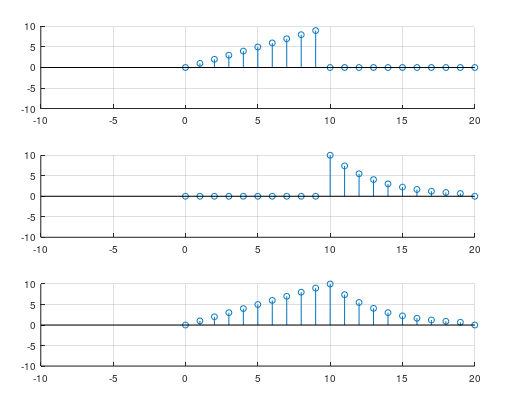
>> axis([-10 20 -10 10]);

>> subplot(3,1,3);

>> stem(n1,x);

>> grid on;

>> axis([-10 20 -10 10]);





**CODE:**

>> x=[5 4 3 2 1];

>> n=0:length(x)-1;

>> subplot(2,1,1);

>> stem(n,x);

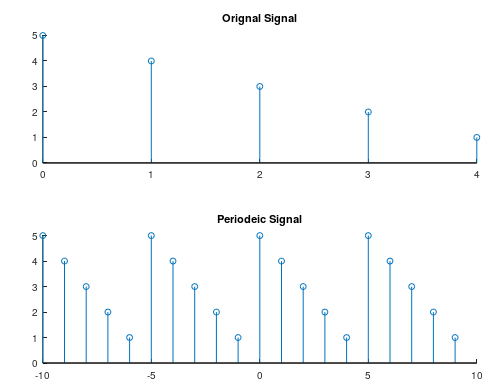
>> title("Orignal Signal");

>> x1=[x,x,x,x];

>> n=-10:9;

>> stem(n,x1);

>> title("Periodeic Signal");



**3B:** Determine and plot the following sequences

1. 

**CODE:**

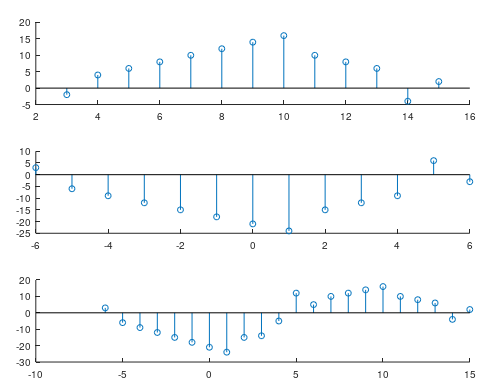
>> n=[-2,-1,0,1,2,3,4,5,6,7,8,9,10];

>> x=[-1,2,3,4,5,6,7,8,5,4,3,-2,1];

>>[y1,n1]=sigshift(2\*x,n,5);

>>[y2,n2]=sigshift(-3\*x,n,-4);

>>[x1,n3]=sigadd(y1,n1,y2,n2)



1. 

**CODE:**

>> [y1,n1]=sigshift(4\*x,-n,-5);

>> [y2,n2]=sigshift(-3\*x,n,-2);

>> [x1,n3]=sigadd(y1,n1,y2,n2);

>> subplot(3,1,1)

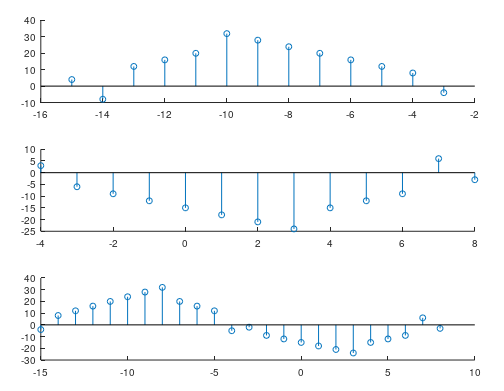
>> stem(n1,y1)

>> subplot(3,1,2)

>> stem(n2,y2)

>> subplot(3,1,3)

>> stem(n3,x1)



**3C:** Generate the complex-valued signal

and plot its magnitude, phase, the real part, and the imaginary part in four separate subplots.

**CODE:**

>> n=[-10:10];

>> x=exp((0.1+0.3j)\*n);

>> subplot(4,1,1);

>> stem(n,real(x));

>> xlabel('Time Samples');

>> ylabel('Real Value');

>> subplot(4,1,2);

>> stem(n,imag(x));

>> xlabel('Time Samples');

>> ylabel('Imaginary Value');

>> subplot(4,1,3);

>> stem(n,abs(x));

>> xlabel('Time Samples');

>> ylabel('Magnitude Value');

>> subplot(4,1,4);

>> stem(n,angle(x));

>> xlabel('Time Samples');

>> ylabel('Angle Value');

>>

