**Texas A&M University-Kingsville**

**Electrical Engineering and**

**Computer Science Department**

**Fall 2015**

**EEEN 5331 Digital Signal Processing**

**Computer Assignment**

**By**

**Teja Renukaradhya**

**K00396303**

**ASSIGNMENT 2.2**

1. Write a program to generate x(n) = 3 + 5.657cos[2π(.1)n] for n = 0, 1, Λ, 99 and print the result in a file (s.dat).

**C-Program:**

#include <stdio.h>

#include <math.h>

#define pi 3.141592654

int main()

{

int value;

double signal;

FILE \*r;

r= fopen("sig.dat", "w");

for (value=0; value<100; value++){

signal = 3. + 5.657\*cos(2.\*pi\*.1\*value);

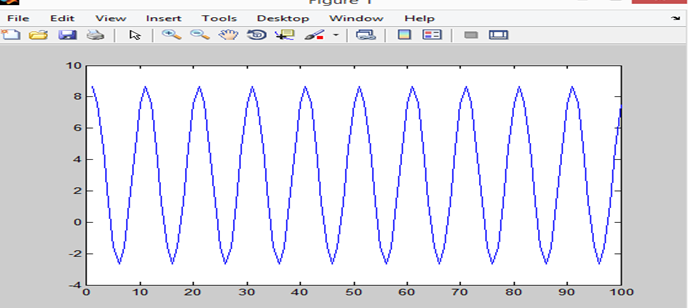
fprintf (result, "%f\n",signal);

}

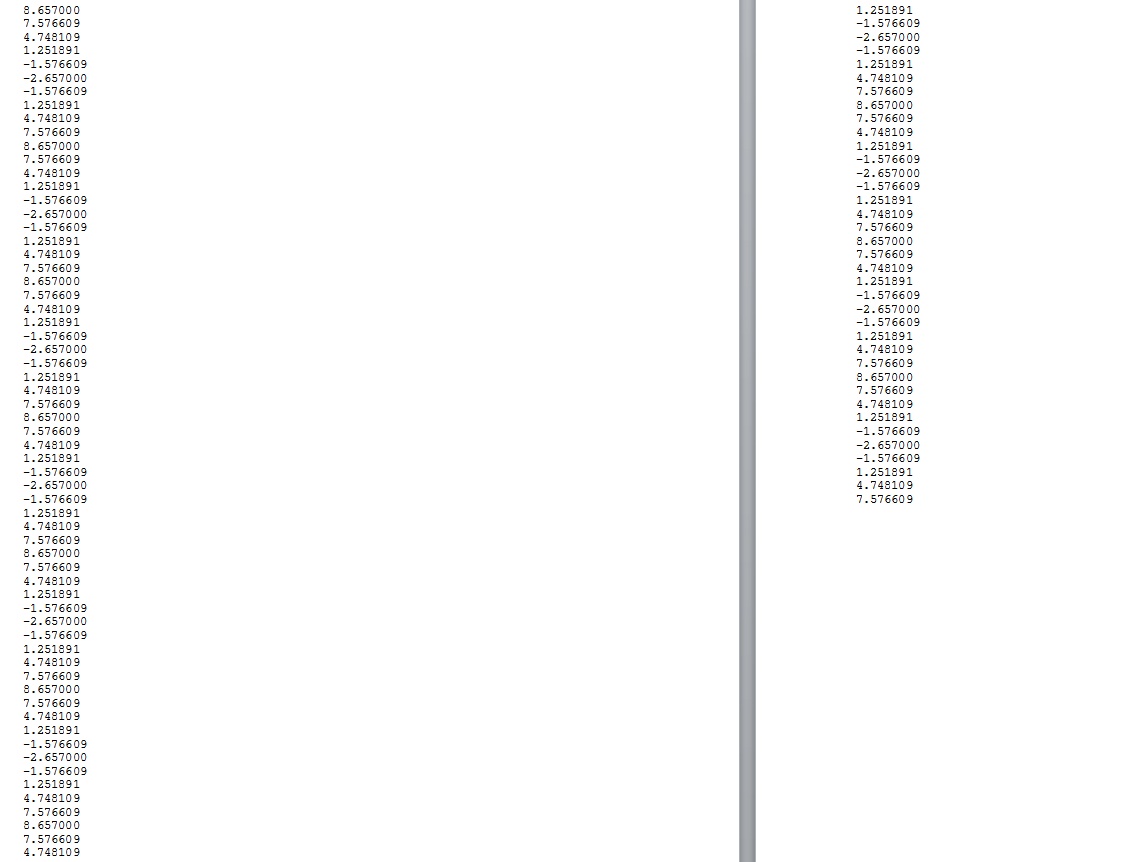
fclose(r);

return 0;

}

Matlab wave form

**s.dat file output**



**(2)**

#include <stdio.h>

#include <stdlib.h>

main() /\* This program is to compute mean, variance, and autocorrelation\*/

{

int value, tt;

double signal[100], m, msquare, v, acorrelation, ma;

FILE \*ip, \*result\_1, \*result\_2;

ip = fopen("sig.dat", "r");

result\_1 = fopen("result\_2.dat", "w");

result\_2 = fopen("result\_3.dat", "w");

m = 0.;

msquare = 0.;

for (value=0; value<100; value++)

fscanf (ip, "%lf", &signal[value]);

for (value=0; value<100; value++)

{

m = m + signal[value];

msquare = msquare + signal[value]\*signal[value];

}

m = m / 100.;

msquare = msquare/100.;

v = msquare - m\*m;

fprintf (result\_1, "mean = %f\tmean square value = %f\tvariance = %f\n", m, msquare,

v);

for (tt=0; tt<100; tt++)

{

acorrelation = 0.;

for(value=0; value<(100-tt); value++) acorrelation = acorrelation + signal[value]\*signal[value+tt];

ma = acorrelation/(100-tt);

acorrelation = acorrelation/100;

fprintf(result\_2, "%f %f\n", acorrelation, ma);

}

fclose(ip);

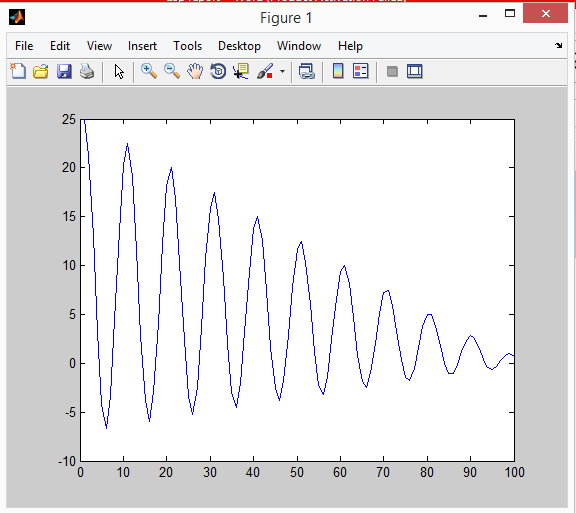
fclose(result\_1);

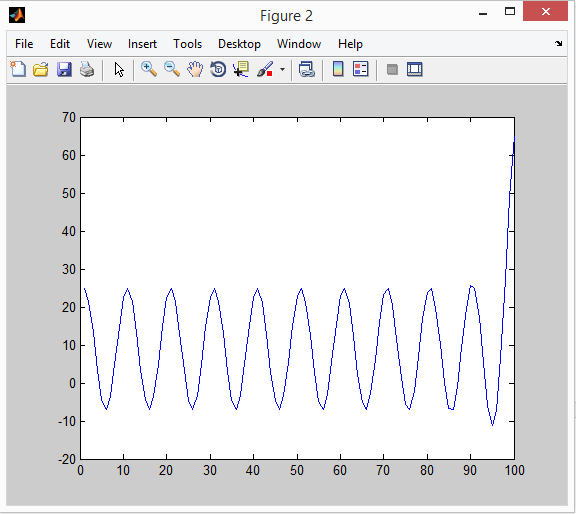
fclose(result\_2);

}

Output:

mean = 3.000000 mean square value = 25.000824 variance = 16.000824





**ASSIGNMENT 3.1**

First creating the given signal =>s(n) = 2cos[2π(.05)n] + sin[2π(.2)n]

#include <stdio.h>

#include <math.h>

#define PI 3.14159265

void main()

{

int value;

double si;

FILE \*out;

out = fopen("signal\_3.dat", "w");

for (value=0; value<256; value++)

{

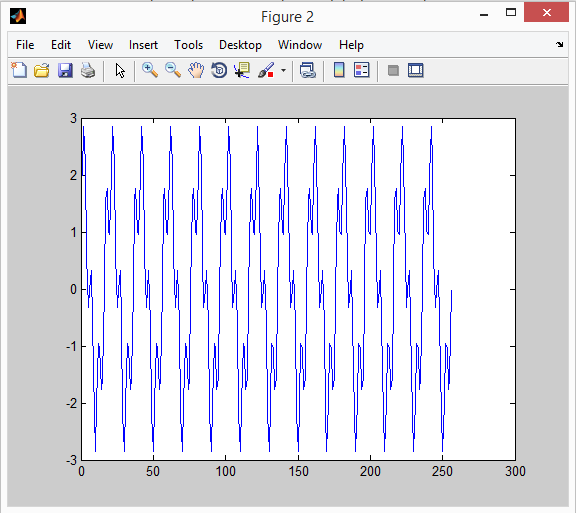
si = 2\*cos(2.\*PI\*.05\*value) + sin(2.\*PI\*.2\*value);

fprintf (out, "%f\n",si);

}

fclose(out)

}



**2)**

#include <stdio.h>

#include <math.h>

#define PI 3.14159265359

#define VAL 256

void d\_f\_t(double seq[], double m[], double n[]);

void main()

{

int value;

double seq[VAL], M[N], N[VAL], mag, ph;

FILE \*ip, \*G, \*p;

ip = fopen("signal.dat", "r");

G = fopen("magnit.dat", "w");

p = fopen("phase.dat", "w");

for (value=0; value<VAL; value++) fscanf (ip, "%lf", &sig[value]);

d\_f\_t (seq, M, N); // s: time-domain sequence, X&Y: real and imaginary parts of DFT

for (value=0; value<VAL; value++)

{

mag= sqrt( M[value]\*N[value] + N[value]\*N[value] ); // Compute the magnitude response phase = atan2(N[value],M[value]); // Phase response

fprintf (G, "%f\n", mag);

fprintf (result2, "%f\n", ph);

}

fclose(ip);

fclose(G);

fclose(result2);

}

void d\_f\_t(double seq[], double m[], double n[]) // s[]:input, x[]&y[]:outputs

{

int value, l;

for (l=0; l<VAL; l++)

{

m[l] = 0.;

n[l] = 0.;

for (value=0; value<VAL; value++)

{

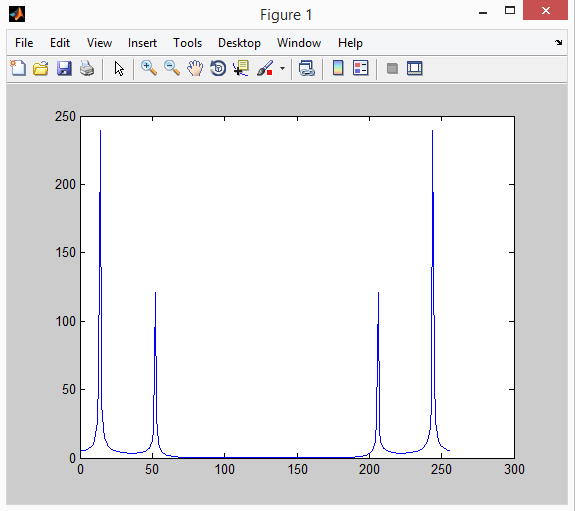
m[l] = m[l] + seq[value]\*cos(2.\*PI\*l\*value/VAL); // Computation of DFT

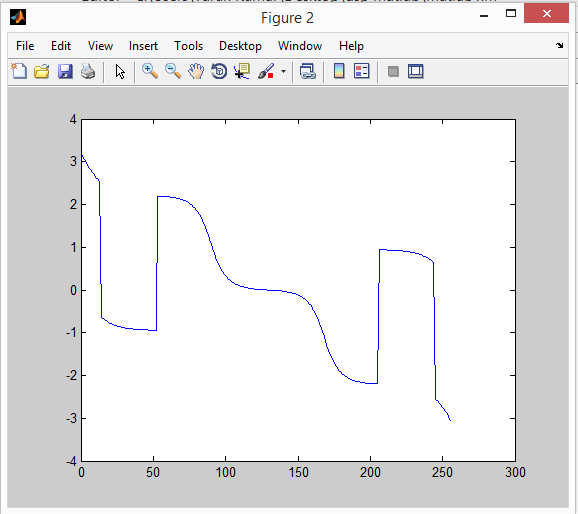
n[l] = n[l] - seq[value]\*sin(2.\*PI\*l\*value/VAL);

}

}

}





**3)**

**// Lowpass Fiter Design Using Windows**

#include <stdio.h>

#include <math.h>

#define PI 3.1415926536

#define VAL 1000

void magnitude\_dftransform(double m[], double magnit[]);

void d\_f\_t(double seq[], double XX[], double YY[]);

void main()

{

int value;

double HREC[16], HAM[16], BLACK[N], m[N], magnit[N], a;

double n1, n2, n3;

FILE \*f, \*o1, \*o2, \*o3;

f = fopen("f.dat", "w");

o1 = fopen("rect.dat", "w");

o2 = fopen("hamm.dat", "w");

o3 = fopen("blac.dat", "w");

for (value=0; value<=7; value++) // LPF design with rectangular window

{

a = (.5 + value) \* 0.2 \* PI; // argument for sinc function

HREC[8+value] = .2 \* sin(a)/a; // 0.5\*sinc(a)

HREC[7-value] = HREC[8+value]; // Use even symmetric property

}

for (value=0; value<=15; value++) // Compute LPF coefficients

{

HAM[value] = HREC[value]\*(0.54 - 0.46\*cos(2\*PI\*value/15) ); // Hamming window

BLACK[value] = HREC[value]\*(.42-.5\*cos(2\*PI\*value/15)+.08\*cos(4\*PI\*value/15)); //Blackman window

}

n1 = n2 = n3 = 0.;

for (value=0; value<=15; value++) // Find the frequency response at dc

{

n1 = n1 + HREC[value];

n2 = n2 + HAM[value];

n3 = n3 + BLACK[value];

}

for (value=0; value<=15; value++) // Normalize f coefficients

{

HREC[value] = HREC[value]/n1;

HAM[value] = HAM[value]/n2;

BLACK[value] = BLACK[value]/n3;

}

fprintf (f, "\tRectangular\tHamming\t\tBlackman\value");

for (value=0; value<=15; value++) fprintf (f,"%d\t%f\t%f \t%f\value",value, HREC[value],HAM[value],BLACK[value]);

for (value=16; value<N; value++) m[value] = 0.0; // Pad extra zeros

for (value=0; value<=15; value++) m[value] = HREC[value]; // Copy rectangular f coefficients

magnitude\_dftransform (m, magnit); // Magnitude response of LPF rectangular window

for (value=0; value<N; value++) fprintf (o1, "%f\value", magnit[value]);

for (value=0; value<=15; value++) m[value] = HAM[value]; // Copy Hamming f coefficients

magnitude\_dftransform (m, magnit); // Magnitude response of LPF using Hamming window

for (value=0; value<N; value++) fprintf (o2, "%f\value", magnit[value]);

for (value=0; value<=15; value++) m[value] = BLACK[value]; // Copy Blackman f coefficients

magnitude\_dftransform (m, magnit); // Magnitude response of LPF using Blackman window

for (value=0; value<N; value++) fprintf (o3, "%f\value", magnit[value]);

fclose(f);

fclose(o1);

fclose(o2);

fclose(o3);

}

void magnitude\_dftransform(double m[], double magnit[])

{

int k;

double XX[N], YY[N];

d\_f\_t(m, XX, YY);

for (k=0; k<N; k++)

{

magnit[k] = XX[k]\*XX[k] + YY[k]\*YY[k]; // magnitude squared

magnit[k] = 10.\*log10(magnit[k]);

}

}

void d\_f\_t(double seq[], double XX[], double YY[]) // seq[]:input, XX[]&YY[]:outputs

{

int value, k;

for (k=0; k<N; k++)

{

XX[k] = 0.;

YY[k] = 0.;

for (value=0; value<N; value++)

{

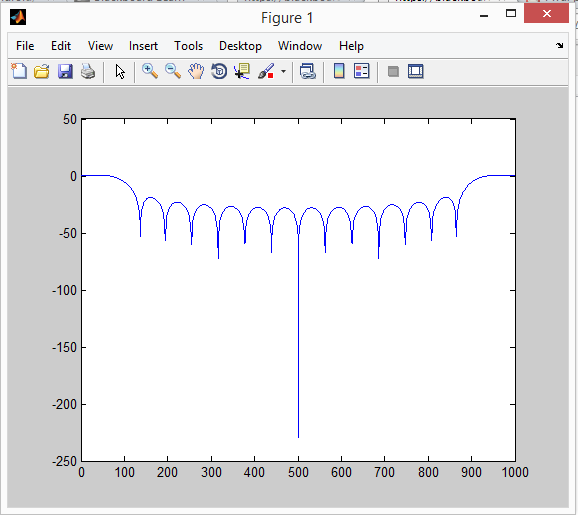
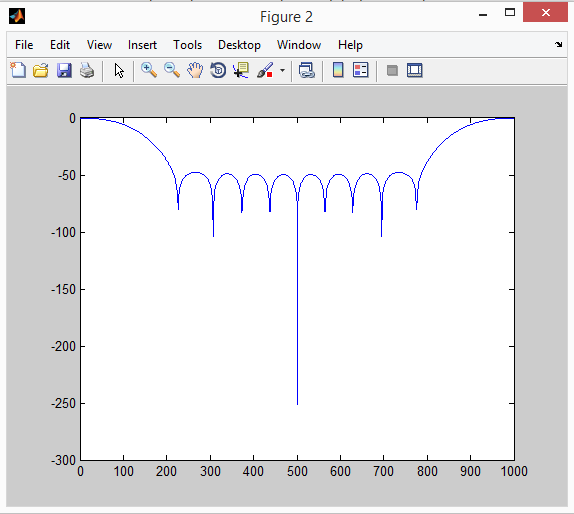
XX[k] = XX[k] + seq[value]\*cos(2.\*PI\*k\*value/N); // Computation of the DFT

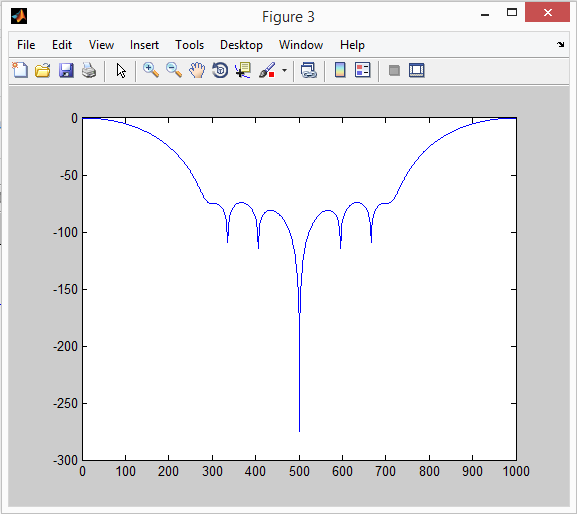
YY[k] = YY[k] - seq[value]\*sin(2.\*PI\*k\*value/N);

}

}

}





**(4)**

#include <stdio.h>

#include <math.h>

#define VAL 16

#define VAL1 15

void main()

{

int value;

double bf[VAL], H[VAL], lowpass;

FILE \*ip, \*result;

ip = fopen("signal.dat", "r");

result = fopen("output.dat", "w");

H[0] =-0.0035; H[1] =-0.0049; // LPF coefficients

H[2] =-0.0042; H[3] = 0.0089;

H[4] = 0.0442; H[5] = 0.1002;

H[6] = 0.1601; H[7] = 0.1991;

for (value=0; value<=7; value++) H[VAL1-value] = H[value]; // Copy H[8] - H[15]

for (value=0; value<=VAL1; value++) bf[value] = 0.; // Initialize the buffer with zero

while ( !feof(ip) ) // As long as there is data ip the file,

{ // the filtering process is continued.

fscanf (ip, "%lf\value", &bf[0]); // Read present input

lowpass = 0; // Initialize current output to be zero

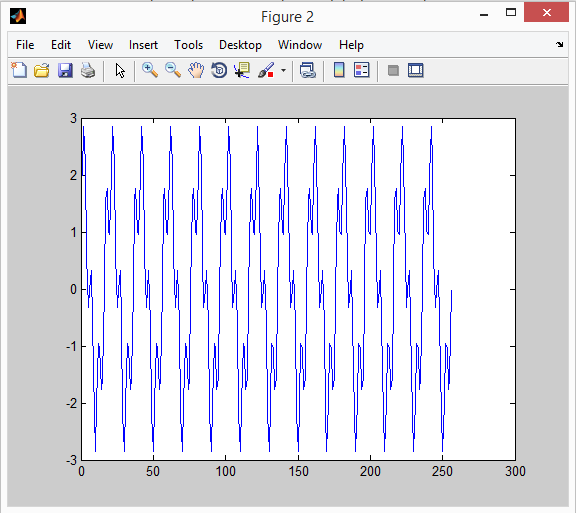
for (value=0; value<VAL; value++) lowpass = lowpass + H[value]\*bf[value]; // Lowpass filtering

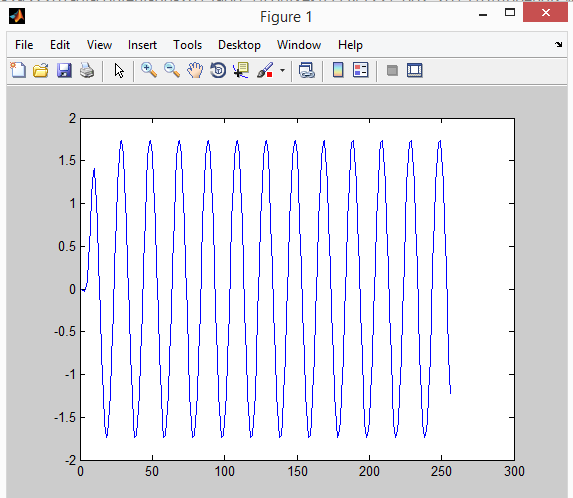
fprintf (result, "%f\value", lowpass);

for (value=VAL1; value>0; value--) bf[value] = bf[value-1]; // Update buffer

}

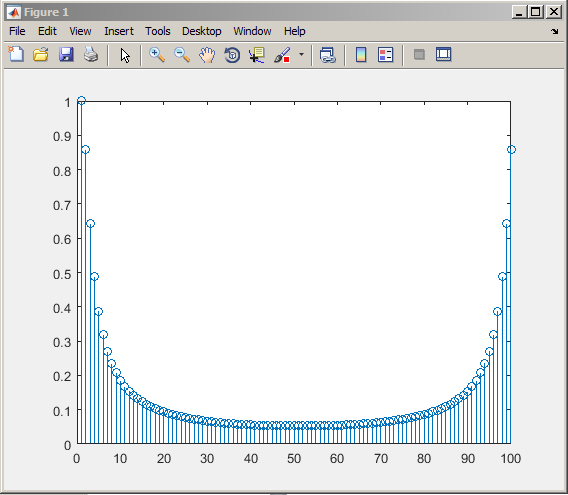
fclose(ip);



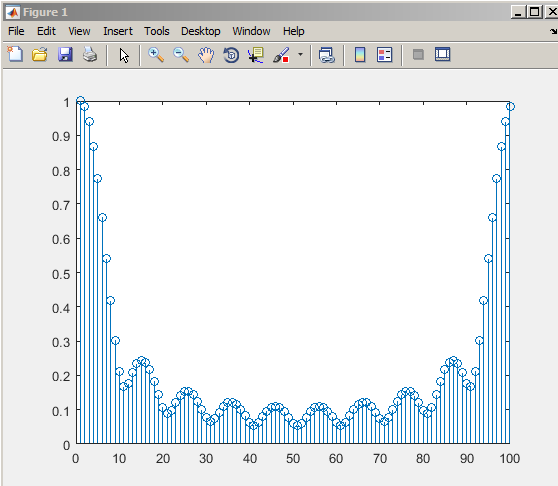


4) Matlab Examples

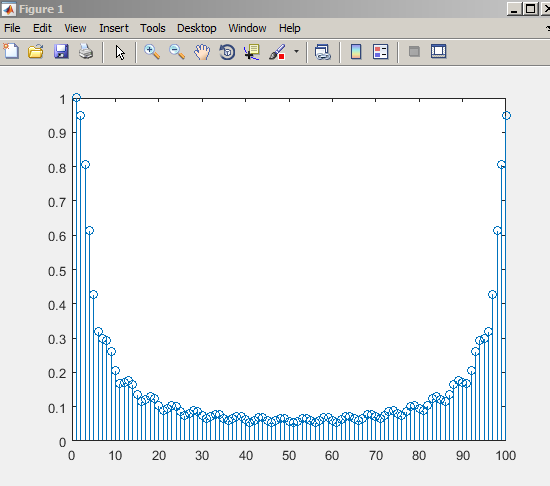
a)



b)



c)



d)

