

ECE 5630: Programming #2

Due on Tuesday, November 24, 2014

Scott Budge 3:00pm

Tyler Travis A01519795

Contents

Problem 1	3
(a)	3
(b)	3
Problem 2	5
(a)	6
(b)	6
(c)	6
Problem 3	9
(a)	11
(b)	12

Problem 1

Design a linear-phase FIR digital filter with the following characteristics:

- 256 coefficients
- Lowpass filter
- Passband frequency of 300 Hz with unit gain.
- Stopband frequency of 400 Hz with zero gain.
- A signal sample rate of 11.025 kHz.

(a)

Plot the impulse response $h(n)$ of the filter.

Figure 1 shows the Impulse response of the filter $h[n]$.

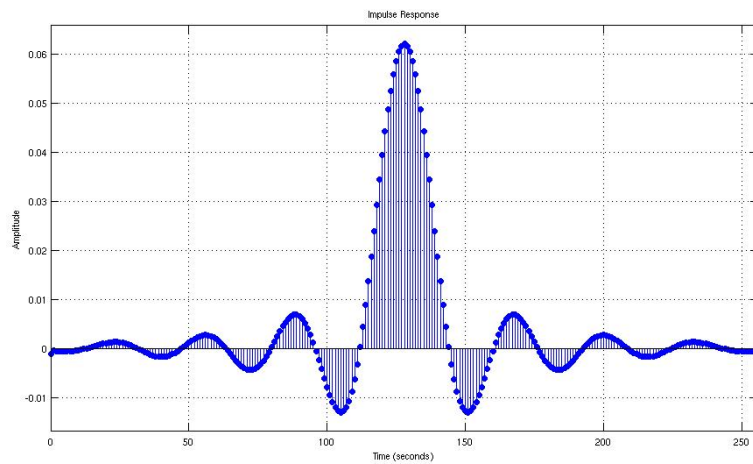


Figure 1: Impulse Response

(b)

Plot the desired magnitude response, the actual magnitude response, and phase response.

Figure 2 shows the Magnitude and Phase response of the filter $h[n]$.

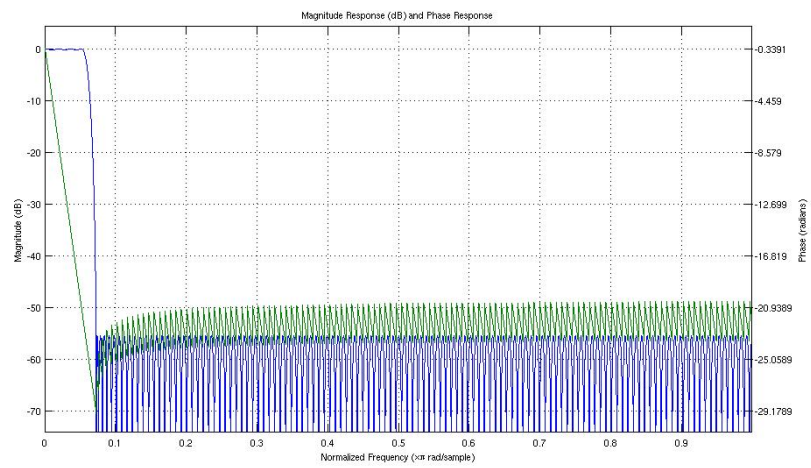


Figure 2: Impulse Response

Problem 2

In C or C++, write a program that performs the FIR filtering in the time domain. Your program should be written so that you can filter an infinite length signal.

Listing 1 shows the first program.

Listing 1: Program 1 - part1.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
5 #include <cstdlib>
#include <cmath>
#include "../includes/fft842.c"

// Filter Length
10 #define Nf 256
// Length of Signal
#define N 25600
// Sampling frequency
const double Fs = 11025;

15
int main(int argc, char** argv)
{
    // Input stream for filter
    std::ifstream filterIn("../data/LowPassFilter.dat");
20
    // filter of length Nf = 256
    double h[Nf];

    // input vairable
25 double in;

    // Read in the filter data
    for(int n = 0; n < Nf; ++n)
    {
30         filterIn >> in;
        h[n] = in;
    }

    // Output streams for the input x signal
35 // and the output y signal
    std::ofstream x_dat("../data/x.dat");
    std::ofstream y_dat("../data/y.dat");

    // input x signal of length N = 25600
40 double x[N];

    // output y signal of Length N = 25600
    double y[N];
```

```
45  // f0 = f/Fs
    // Normalized frequency
    double f = atof(argv[1]);
    double f0 = f/Fs;

50  // Generate input signal x[n]
    for(int n = 0; n < N; ++n)
    {
        x[n] = cos(2*M_PI*f0*n);
        x_dat << x[n] << std::endl;
55  }

    double temp;
    for(int n = 0; n < N; ++n)
    {
60        temp = 0;
        for(int k = 0; k < Nf; ++k)
        {
            temp += x[n-k]*h[k];
        }
65        y[n] = temp;
        y_dat << y[n] << std::endl;
    }

    return 0;
70 }
```

(a)

What are the number of multiplies and adds per output sample?

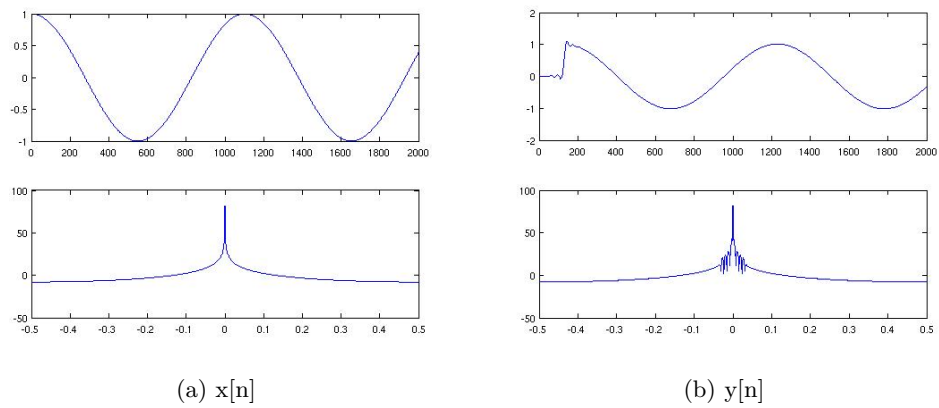
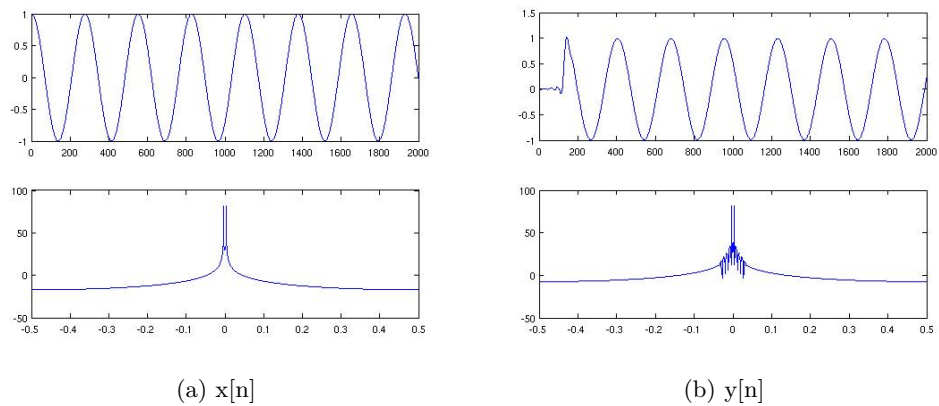
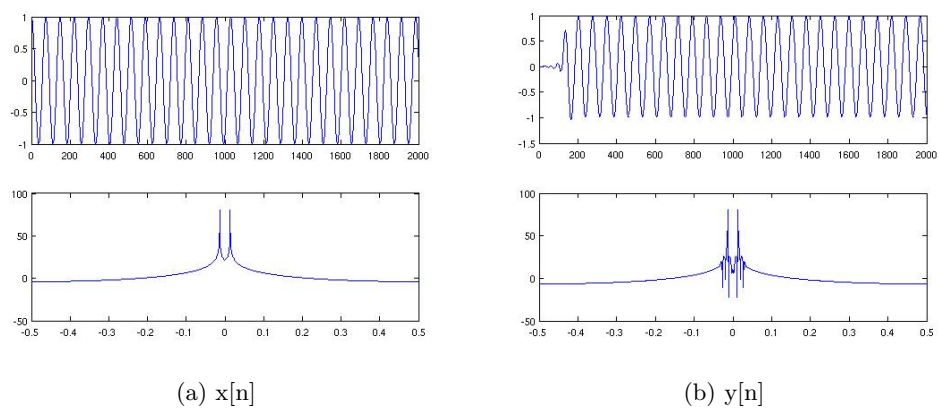
The number of multiples is

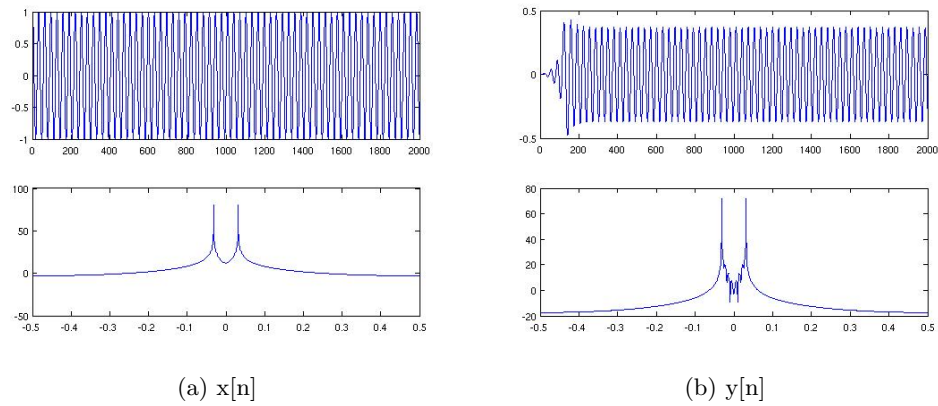
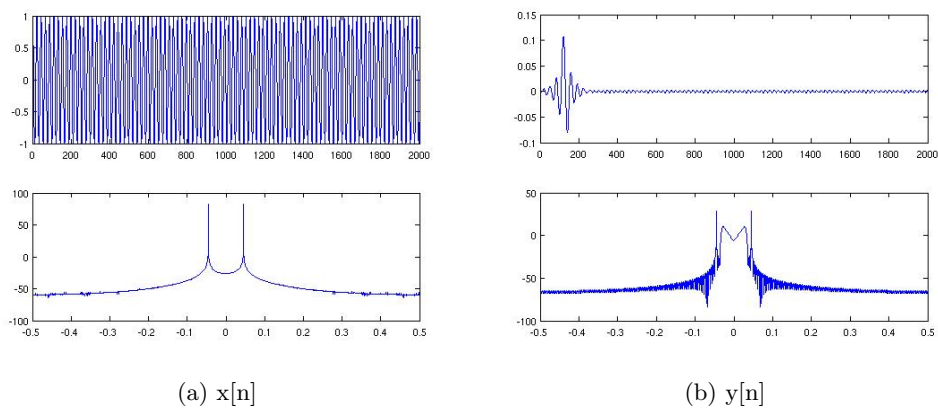
(b)

Verify that your filter coefficients are correct and your filter routine works as expected by running sinusoids through your filter and finding the magnitude response. (Remember to wait long enough that the filter transients have died down.) Do this for frequencies of $f = 10Hz$, $f = 40Hz$, $f = 150Hz$, $f = 350Hz$, and $f = 500Hz$. Use a sample rate of 11.025kHz.

(c)

Compare the computed magnitude response with the theoretical magnitude response (obtained from Matlab). Record the magnitude response for each of the input frequencies and plot them on a plot with the theoretical magnitude response.

Figure 3: Input(a) and Output(b) with $f = 10\text{Hz}$ Figure 4: Input(a) and Output(b) with $f = 40\text{Hz}$ Figure 5: Input(a) and Output(b) with $f = 150\text{Hz}$

Figure 6: Input(a) and Output(b) with $f = 350\text{Hz}$ Figure 7: Input(a) and Output(b) with $f = 500\text{Hz}$

Problem 3

In C or C++, write a program that preforms the FIR filtering in the frequency domain using fast convolution and at least a 512-point FFT. Consider the following:

- You may do **either** overlap-add or overlap-save in your program.
- You only need to perform the FFT on the lowpass filter once.
- The FFT must be a power of two.
- The number of operations/output sample will vary as a function of the length of the FFT used.
- Your program should be written so that you can filter an infinite length signal.

Listing 2 shows the first program.

Listing 2: Program 1 - part2.cpp

```

#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
5 #include <cstdlib>
#include <cmath>
#include <cstring>
#include "../includes/fft842.c"

10 // Filter Length
#define Nf 256
// Length of Signal
#define N 25600
// Sampling frequency
15 const double Fs = 11025;

complex mult(complex, complex);

int main(int argc, char** argv)
20 {
    // Input stream for filter
    std::ifstream filterIn("../data/LowPassFilter.dat");

    // filter of length Nf = 256
    // Nf*4 for zero padding
25 complex h[4*Nf];

    // input vairable
    double in;

30
    // Read in the filter data
    for(int n = 0; n < Nf; ++n)
    {
        filterIn >> in;
35 h[n].re = in;

```

```

    h[n].im = 0;
    h[n+Nf].re = 0;
    h[n+Nf].im = 0;
    h[n+2*Nf].re = 0;
40    h[n+2*Nf].im = 0;
    h[n+3*Nf].re = 0;
    h[n+3*Nf].im = 0;
}

45    // Output streams for the input x signal
    // and the output y signal
    std::ofstream x_dat("../data/x.dat");
    std::ofstream y_dat("../data/y.dat");
    std::ofstream H_dat("../data/H.dat");

50    // input x signal of length N = 25600
    complex x[N];

    // output y signal of Length N = 25600
55    complex y[N+Nf-1];

    // f0 = f/Fs
    // Normalized frequency
    double f = atof(argv[1]);
60    double f0 = f/Fs;

    // Generate input signal x[n]
    for(int n = 0; n < N; ++n)
    {
65        x[n].re = cos(2*M_PI*f0*n);
        x[n].im = 0;
        x_dat << x[n].re << std::endl;
    }

70    // Calculating the fft using the overlap and save method
    // using the fft842 with a 1024-point fft
    int M = 256;
    int overlap = M-1;
    int nfft = 1024;
75    int stepsize = nfft - overlap;

    complex H[nfft];
    memcpy(H, h, sizeof(h));
    // generate fft of the filter
80    fft842(0, nfft, H);

    // Send the data to the corresponding file
    for(int i = 0; i < nfft; ++i)
    {
85        H_dat << H[i].re << "\t" << H[i].im << std::endl;
    }

    // yt is a temp variable for y - the output

```

```

90     complex yt[nfft];
    // xt is a temp variable for storing the correct values of x
    // for computing the fft and multiplying it by the filter's response
    complex xt[nfft];

    // The process for the computing the convolution
95     int position = 0;
    while(position + nfft <= N)
    {
        // Pull out the required data of the x input
        for(int j = 0; j < nfft; ++j)
100         {
            xt[j] = x[j + position];
        }

        // Calculating the corresponding fft
105         fft842(0, nfft, xt);
        // Multiply the points of the x and h magnitude
        for(int k = 0; k < nfft; ++k)
        {
            yt[k] = mult(xt[k], H[k]);
110         }
        // Compute the inverse fft
        fft842(1, nfft, yt);
        // The overlap-save portion
        for(int j = M-1; j < nfft; ++j)
115         {
            y[j-M+position] = yt[j];
        }
        position += stepsize;
    }
120     // Output the data
    for(int n = 0; n < N; ++n)
    {
        y_dat << y[n].re << std::endl;
    }
125     return 0;
}

// multiply complex data correctly
complex mult(complex a, complex b)
130 {
    complex ret;
    ret.re = a.re * b.re - a.im * b.im;
    ret.im = a.re * b.im + a.im * b.re;
    return ret;
135 }

```

(a)

What are the number of multiplies and adds per output sample for both overlap-add and overlap-save?

The number of multiples is

(b)

Repeat 2(b)-(c) with the frequency domain filter program

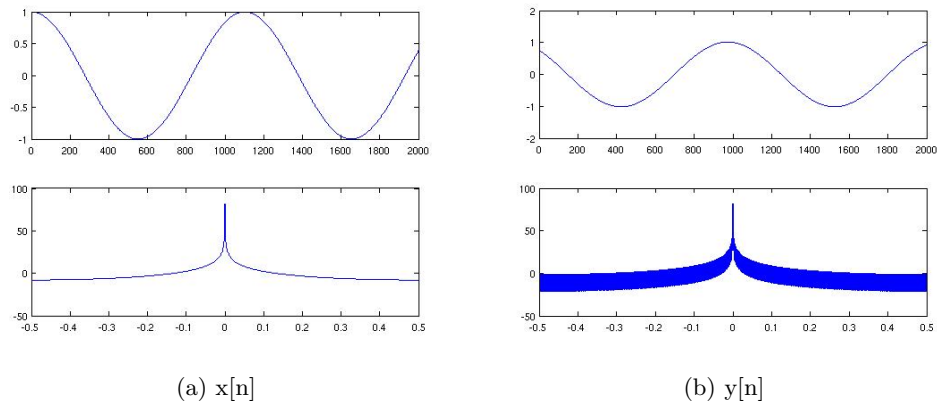


Figure 8: Input(a) and Output(b) with $f = 10\text{Hz}$

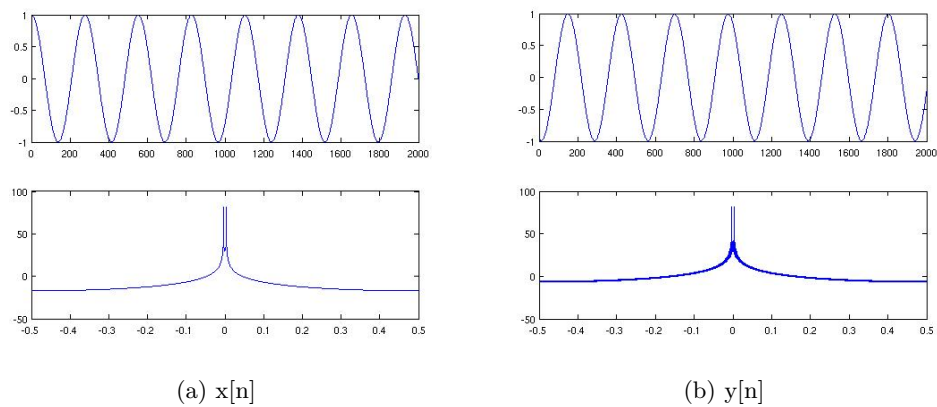
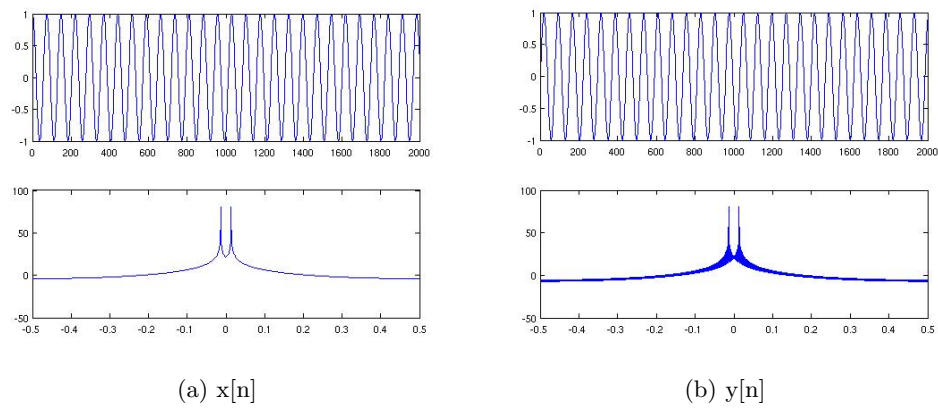
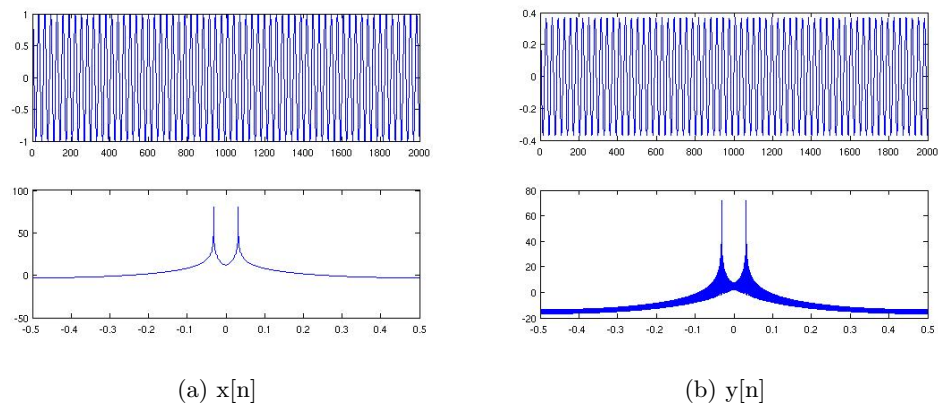
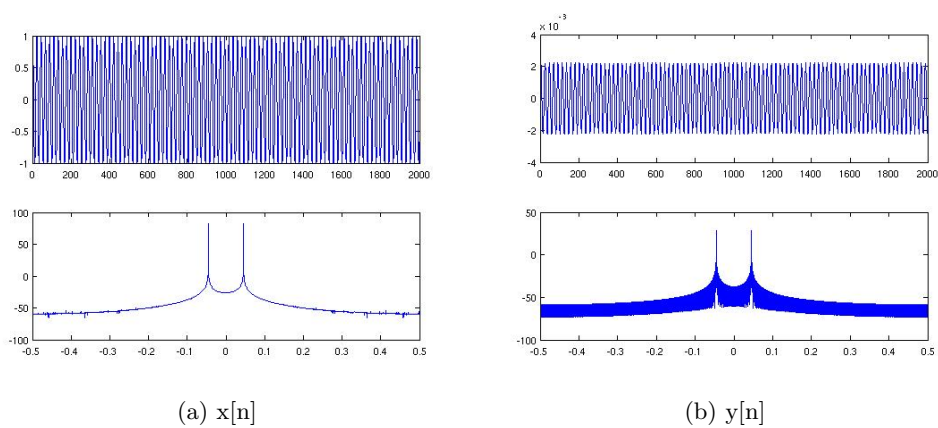


Figure 9: Input(a) and Output(b) with $f = 40\text{Hz}$

Figure 10: Input(a) and Output(b) with $f = 150\text{Hz}$ Figure 11: Input(a) and Output(b) with $f = 350\text{Hz}$ Figure 12: Input(a) and Output(b) with $f = 500\text{Hz}$