ECE 5630: Programming #2

Due on Tuesday, November 24, 2014

Scott Budge 3:00pm

Tyler Travis A01519795

Contents

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 impuse resonse h(n) of the filter.

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

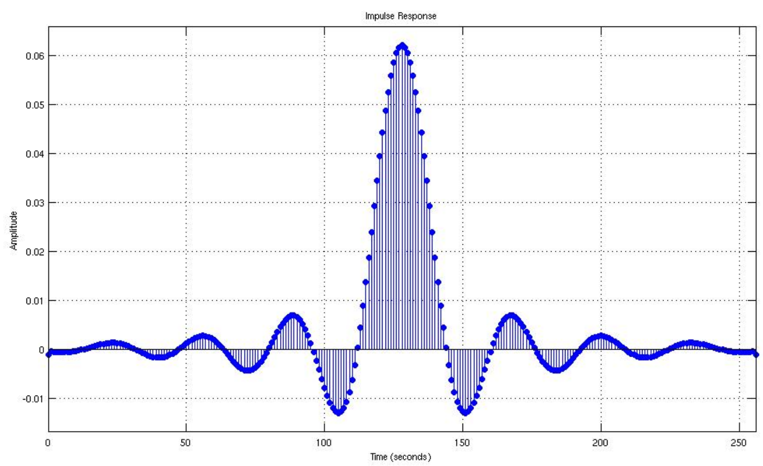


Figure 1: Impulse Response

(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].

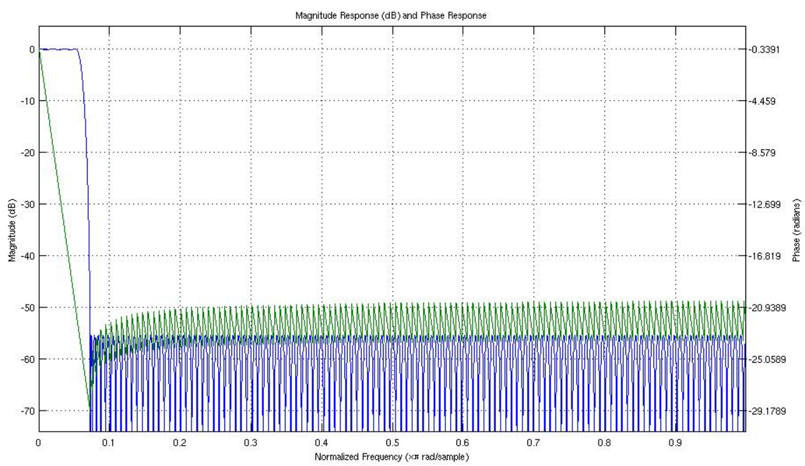


Figure 2: Impulse Response

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>
#include <cstdlib>
#include <cmath>
#include "../includes/fft842.c"

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

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

  // filter of length Nf = 256
  double h[Nf];

  // input vairable
  double in;

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

  // 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");

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

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

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

  // 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;
  }

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

  return 0;
}
```

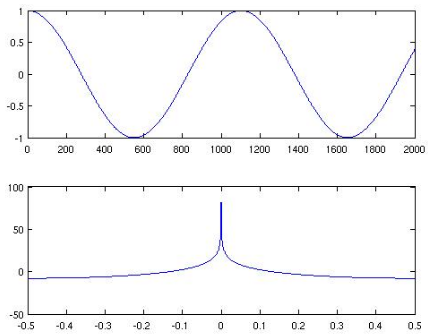
(a)

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

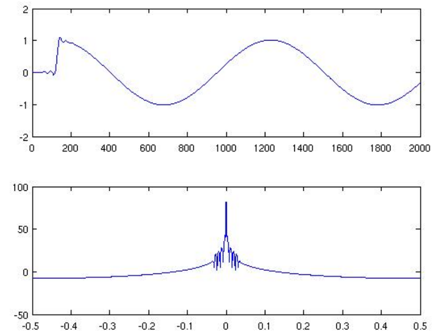
The number of multiples is

(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.

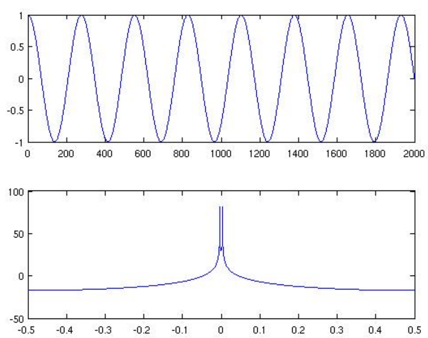


(a) x[n]

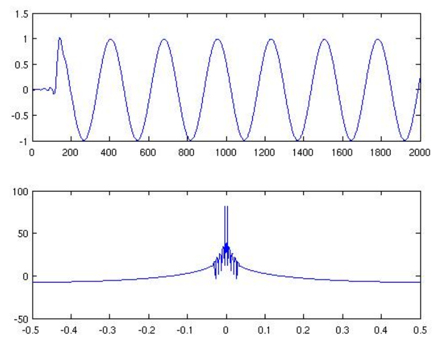


(b) y[n]

Figure 3: Input(a) and Output(b) with f = 10Hz

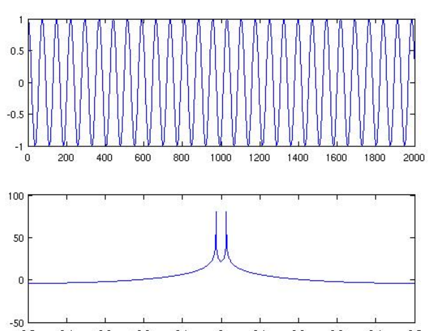


(a) x[n]

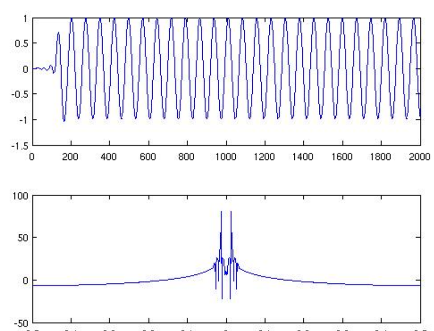


(b) y[n]

Figure 4: Input(a) and Output(b) with f = 40Hz



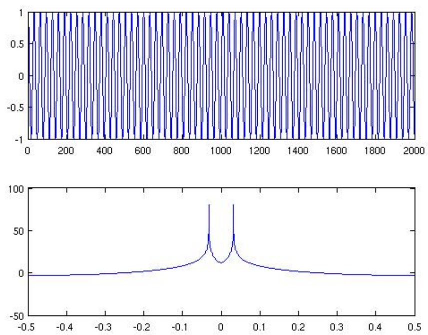
(a) x[n]



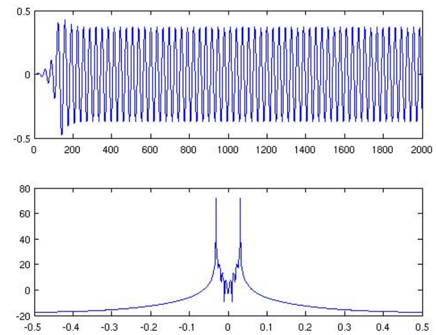
(b) y[n]

Figure 5: Input(a) and Output(b) with f = 150Hz

Problem 2 continued on next page. . .

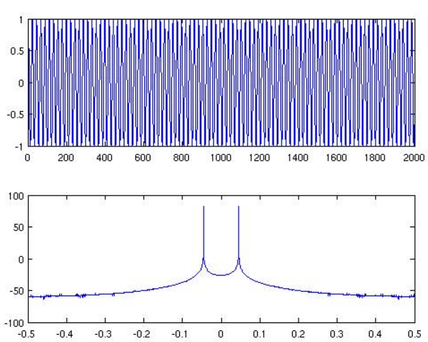


(a) x[n]

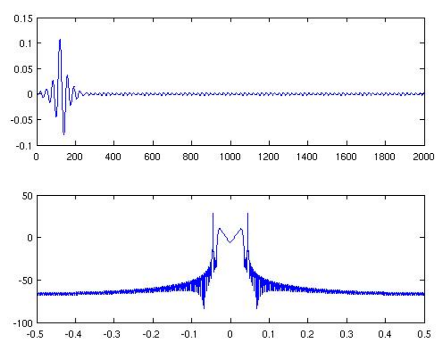


(b) y[n]

Figure 6: Input(a) and Output(b) with f = 350Hz



(a) x[n]



(b) y[n]

Figure 7: Input(a) and Output(b) with f = 500Hz

(c)

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

	10Hz	40Hz	150Hz	350Hz	500Hz
Matlab					
Matlab					
Matlab					

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 prgram.
- 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>
#include <cstdlib>
#include <cmath>
#include <cstring>
#include "../includes/fft842.c"

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

complx mult(complx, complx);

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

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

  // input vairable
  double in;

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

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

  // 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");

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

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

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

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

  // Cacluating 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;
  int stepsize = nfft - overlap;

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

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

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

Problem 3 continued on next page...

```
  complx 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 resposne
  complx xt[nfft];

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

    // Calculating the corrisponding fft
    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]);
    }
    // Compute the inverse fft
    fft842(1, nfft, yt);
    // The overlap-save portion
    for(int j = M-1; j < nfft; ++j)
    {
      y[j-M+position] = yt[j];
    }
    position += stepsize;
  }
  // Output the data
  for(int n = 0; n < N; ++n)
  {
    y_dat << y[n].re << std::endl;
  }
  return 0;
}

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

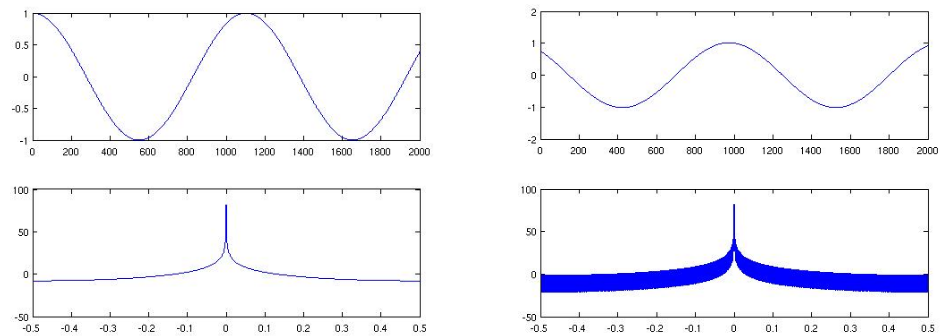
(a)

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

The number of multiples is

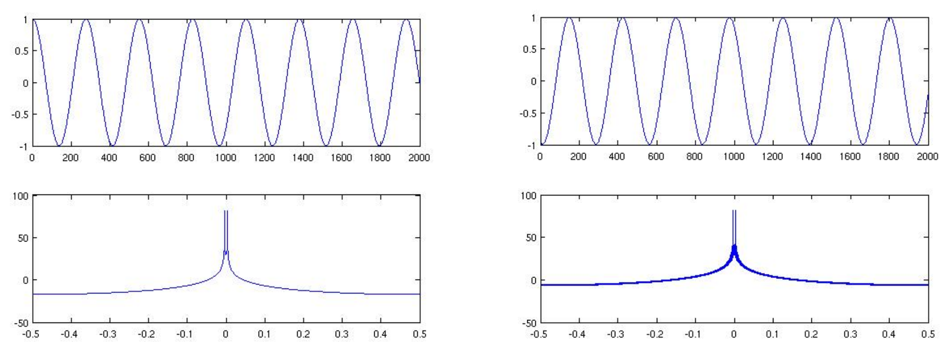
(b)

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



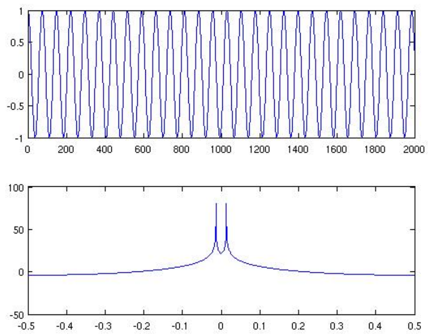
(a) x[n] (b) y[n]

Figure 8: Input(a) and Output(b) with f = 10Hz

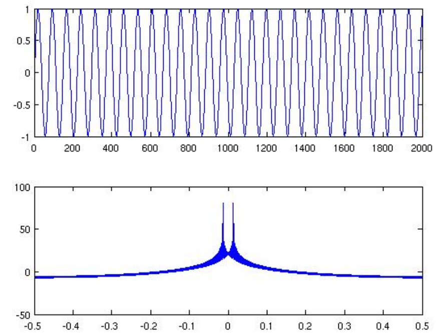


(a) x[n] (b) y[n]

Figure 9: Input(a) and Output(b) with f = 40Hz

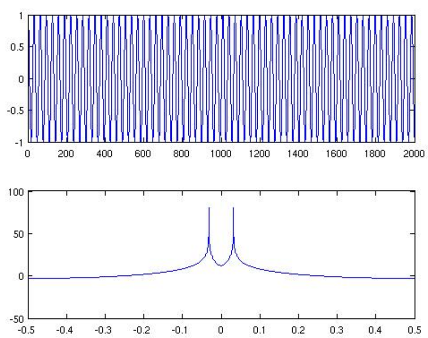


(a) x[n]

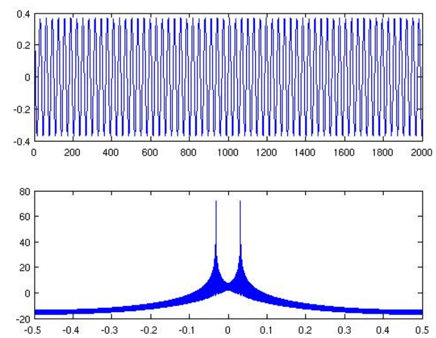


(b) y[n]

Figure 10: Input(a) and Output(b) with f = 150Hz

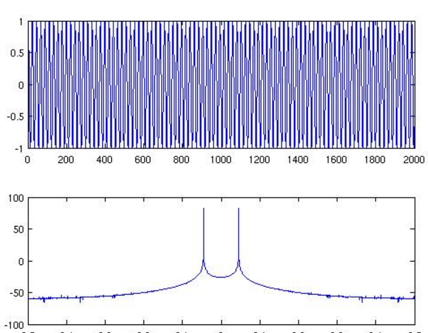


(a) x[n]

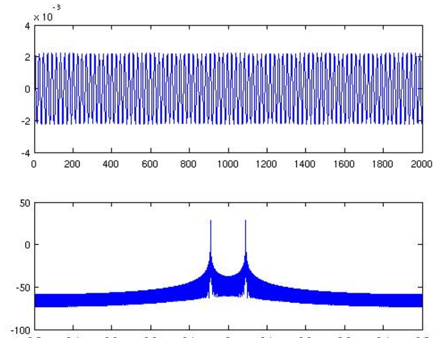


(b) y[n]

Figure 11: Input(a) and Output(b) with f = 350Hz



(a) x[n]



(b) y[n]

Figure 12: Input(a) and Output(b) with f = 500Hz

Problem 3 [(b)] continued on next page. . .

	10Hz	40Hz	150Hz	350Hz	500Hz
Matlab					
Matlab					
Matlab					

Problem 4

Use the Matlab function wavread() to generate the samples of the file galway11_mono_45sec.wav. Use your programs from 2 and 3 above to filter the sound file. The results should be the same for both programs. Does the filter remove the high frequency components? Does the processed file sound as you expected? Write out the final results in a .wav file for the instructor to listen to.

The filter removes the high frequency components from the signal. You can no longer hear the flute playing in the foreground. You can only hear the bass notes of deeper instruments. The processed file sounds like what I expected, the filter completely took out the higher sounds.