BIEN 3200: Lab 6

Laboratory Exercise #6

Peter Dobbs
Lab Section 401
Nicholas Heugel

23 November 2016

Abstract:

The value of digital filters and the use of the impulse response and method of convolution to biomedical signal processing is certainly great. In this lab, the mentioned skills are explored and implemented for the better understanding of their value. Two programs were created that perform a number of important tasks. The first determines the impulse response function, frequency amplitude spectrum, and cut-off frequency of a given filter. The second manipulates an image using a 2-dimensional filter to reveal the edges, or sharpen the image. Thresholding techniques that are implemented prove to aid in edge detection and exposure.

Introduction:

This laboratory was conducted to expose students to impulse response functions, digital filtering, and convolution, as well as their value in signal and image processing. These abilities are important for the biomedical engineer because of the physiological signals and images that need manipulation for proper understanding of the data.

Data Files:

EEG_0.txt (1)

This file contains one second of electroencephalogram (EEG) data from the response of rats to flash stimulation. It was sampled at 500 Hz.

im2.txt (2)

This text file contains a CT image of a human torso.

HIGHPASS.txt (3)

This file holds the filter coefficients for a high pass digital filter.

LOWPASS.txt (4)

This file holds the filter coefficients for a low pass digital filter.

Methods:

The first program was developed to perform three processes: determining the impulse response function, frequency amplitude spectrum, and cut-off frequency of a given filter. With the functionality of this program, the students are able to provide data for graphing the impulse response and frequency amplitude spectra of a low-pass and high-pass filter, as well as the plots of manipulations on an EEG dataset. Specifically, the manipulations include: low-pass filtered, high-pass filtered, and the frequency amplitude spectra of the filtered data.

Determining the impulse response of a given filter is quite simple. Sending an impulse, where the amplitude is 1 at time 0 and 0 elsewhere, through the filter results in an output of the impulse response from the filter. That response can be output to a text file for later use. Filtering the input EEG signal is just the same as determining the impulse response, except that the input to the filter is the EEG signal and the output is the filtered signal.

Manipulating the given filter file to create a two-dimensional filter is straightforward. Simply create an array of pointer arrays. Then allocate memory in each of those pointer arrays for the two dimensions of the filter.

Results:

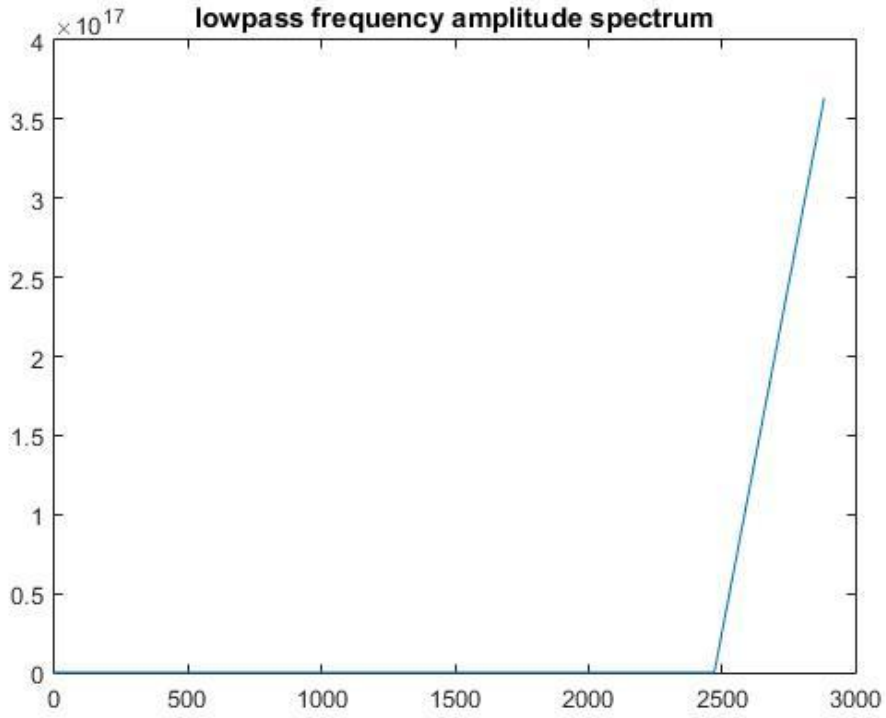

Figure 1: Plot of the frequency amplitude spectrum for a low-pass filter

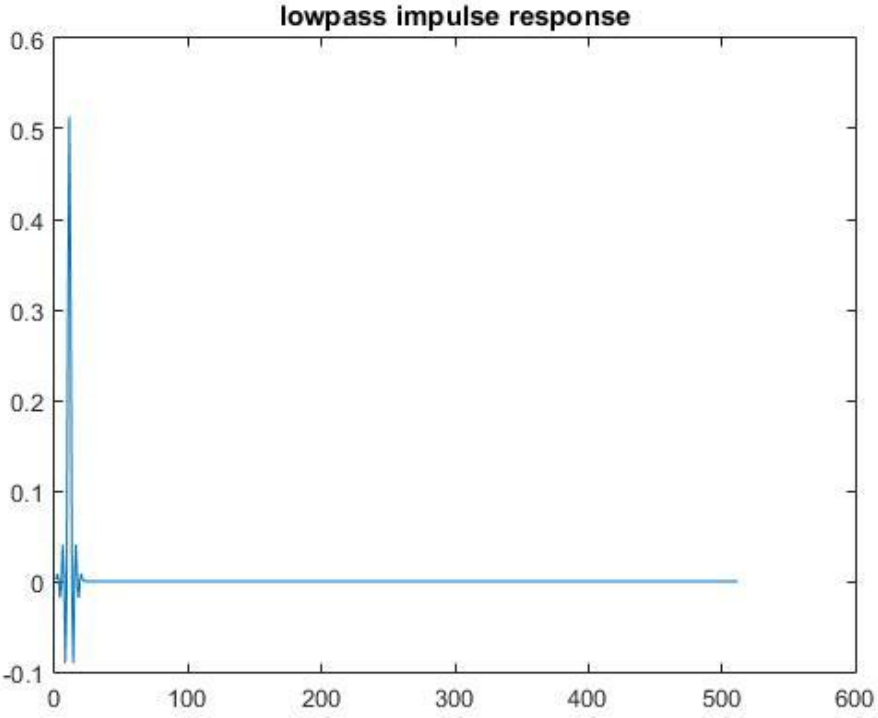

Figure 2: Plot of the impulse response from a low-pass digital filter

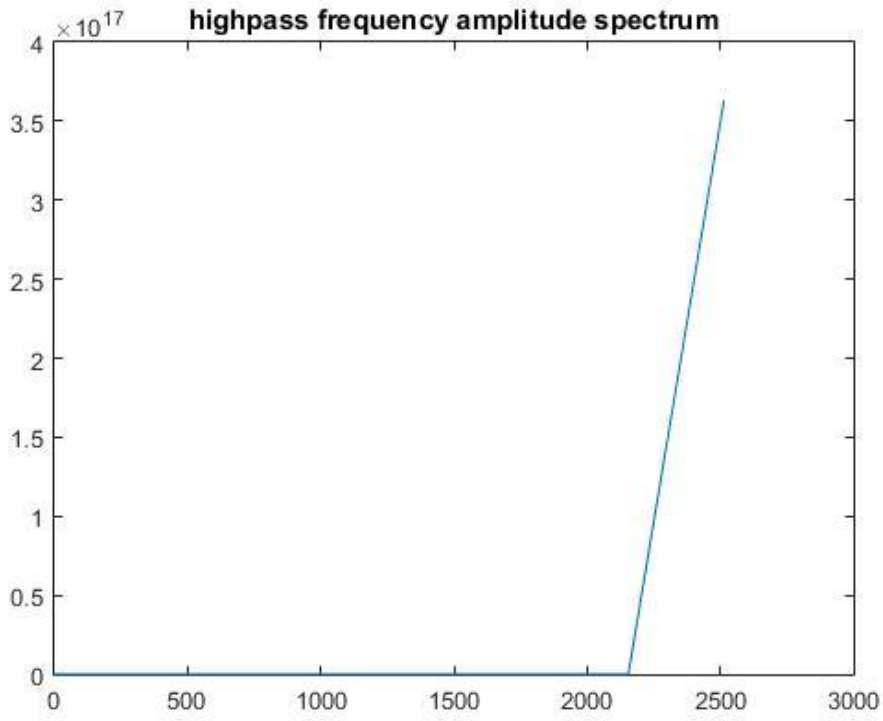

Figure 3: Plot of the frequency amplitude spectrum for a high-pass digital filter

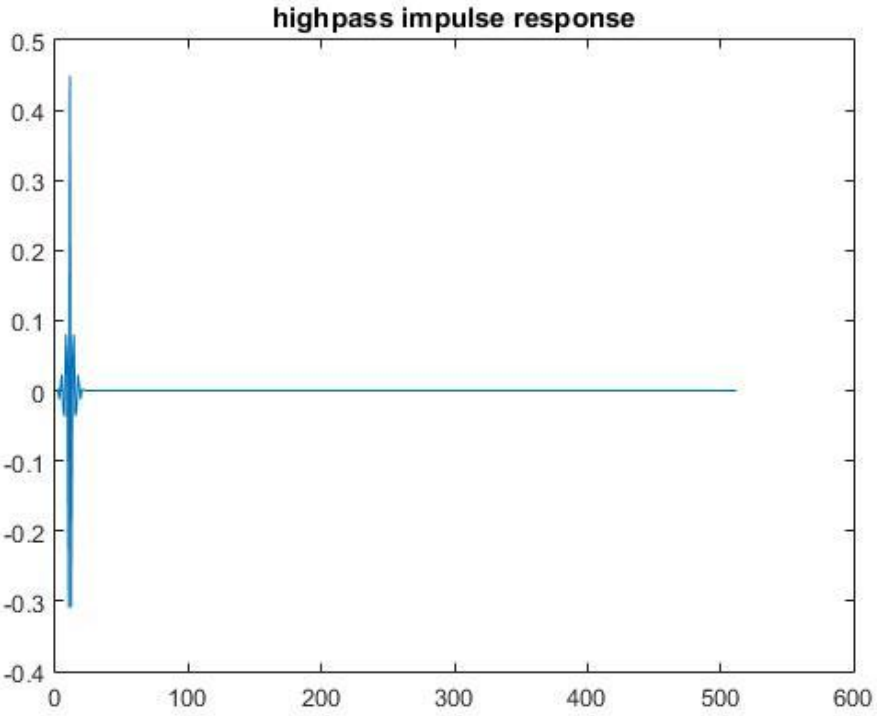

Figure 4: Plot of the impulse response for a high-pass digital filter

EEG Signal Plots

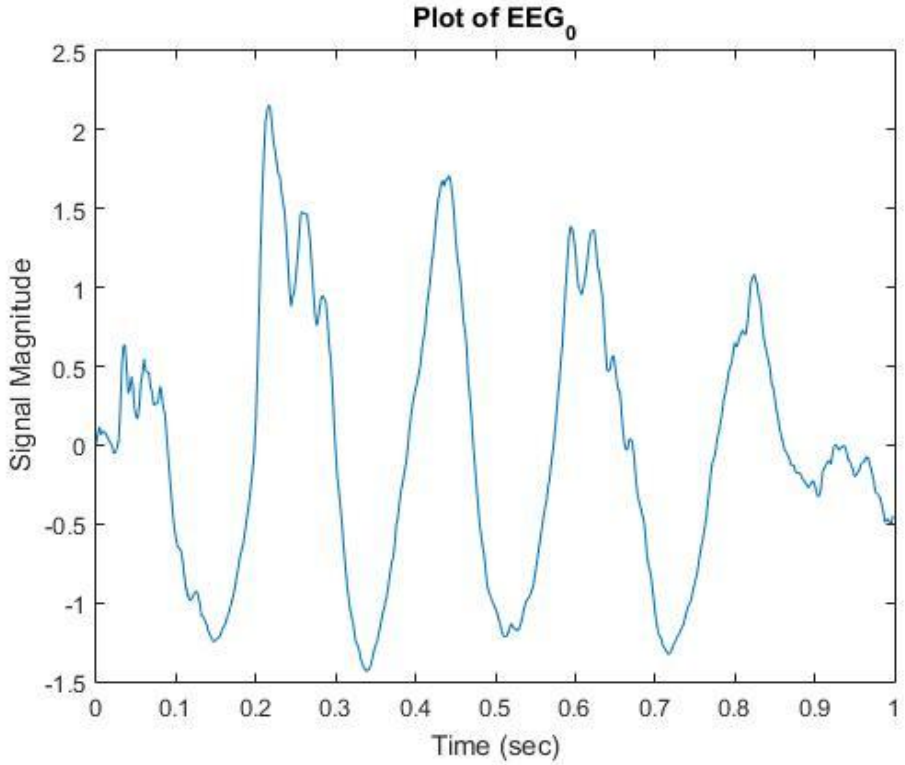

Figure 5: Plot of the data contained in the EEG_0.txt file

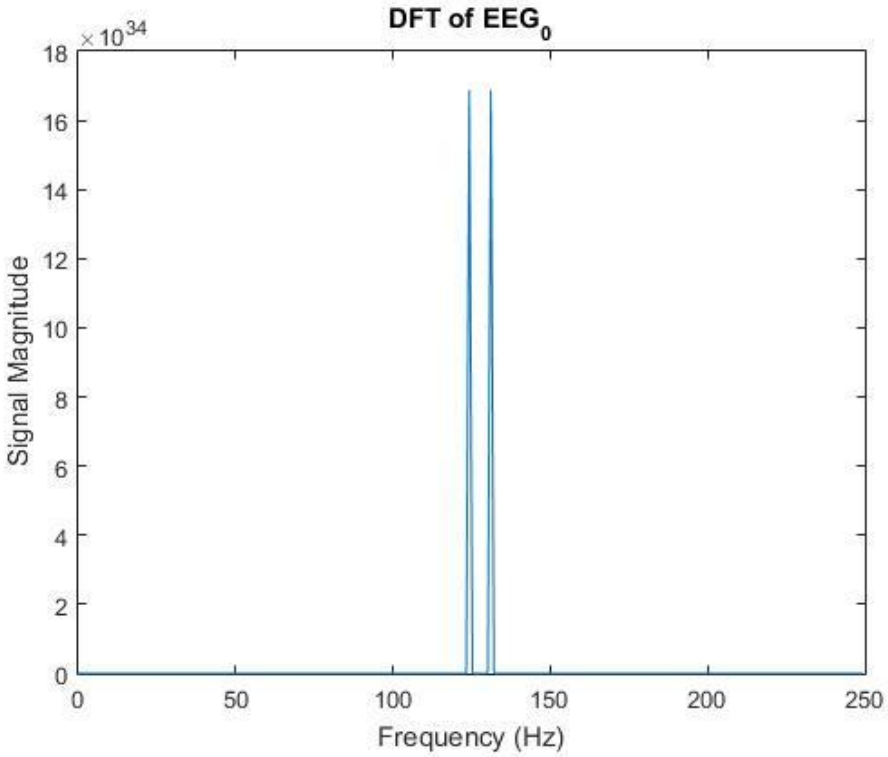

Figure 6: Plot of the unfiltered EEG signal's frequency amplitude spectrum

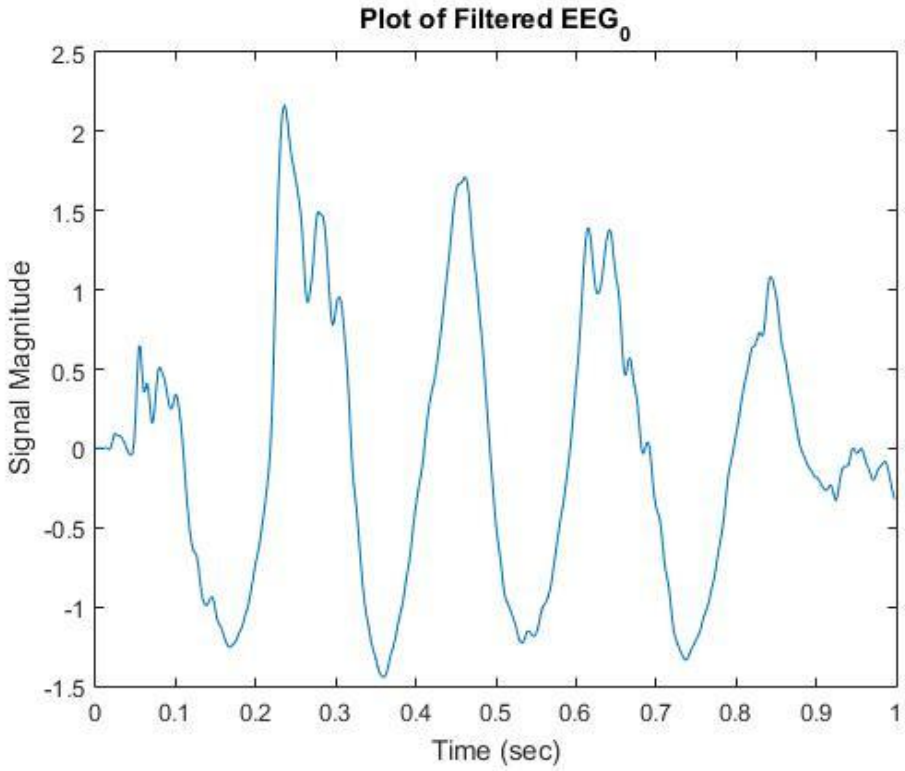

Figure 7: Plot of the EEG signal after filtration

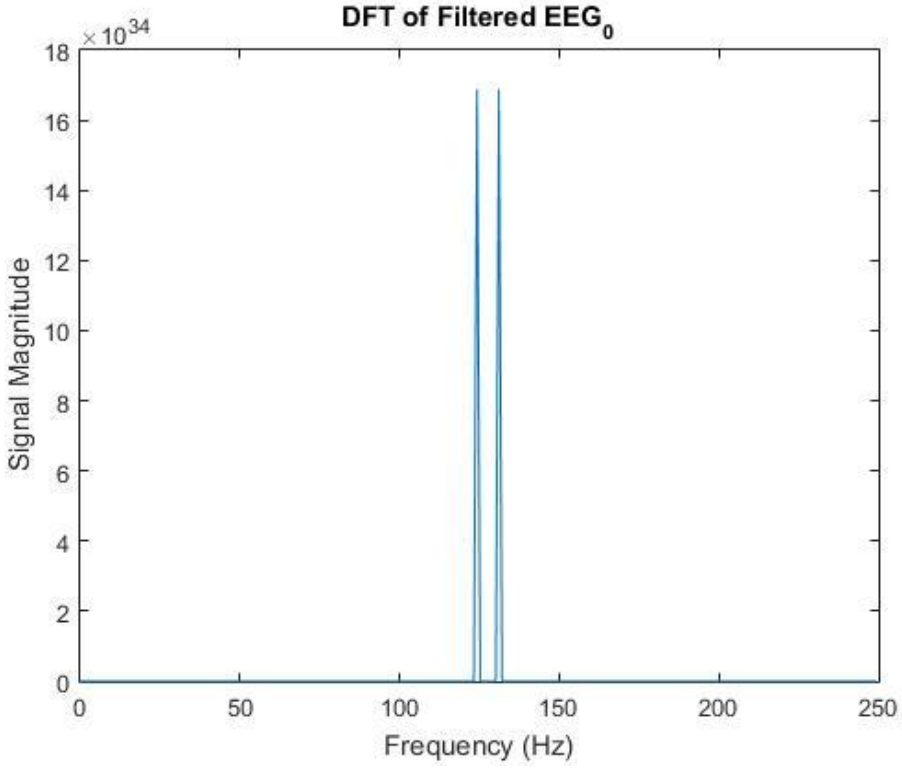

Figure 8: Plot of the frequency amplitude spectrum for the filtered EEG signal

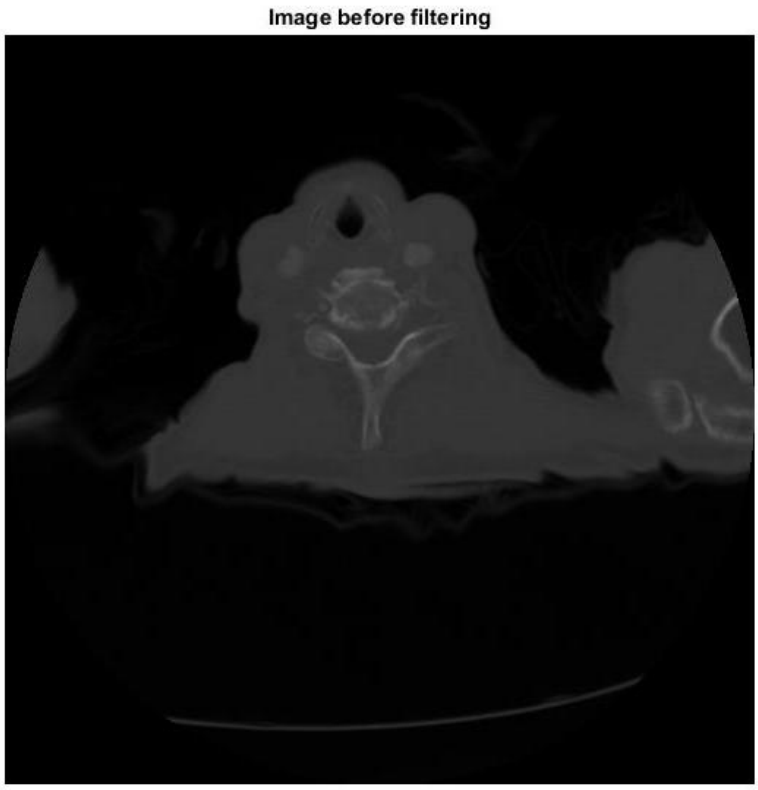

Figure 9: Matlab plot of the CT scan contained in im2.txt

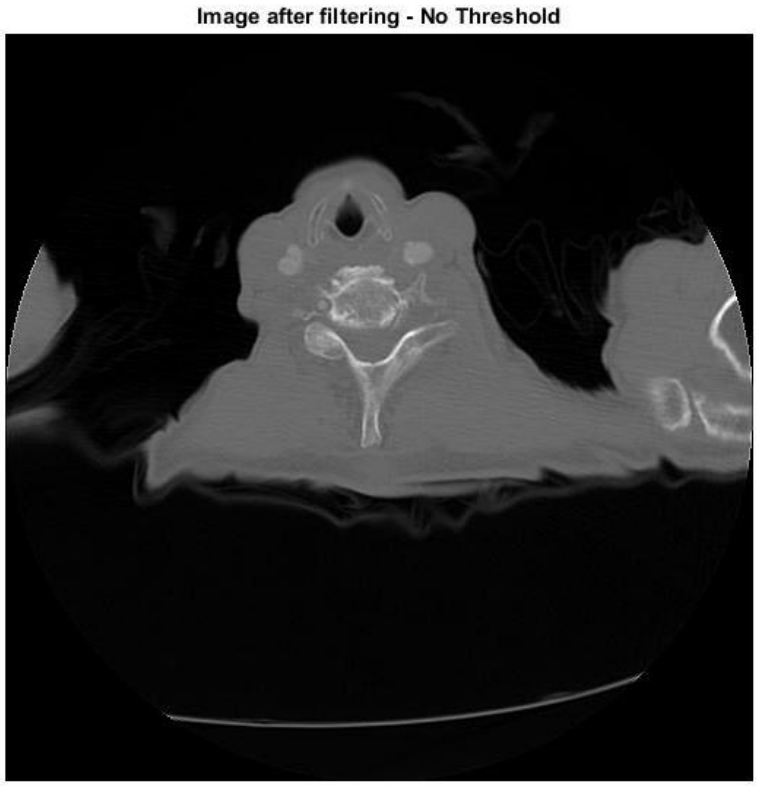

Figure 10: CT image after filtering without threshold

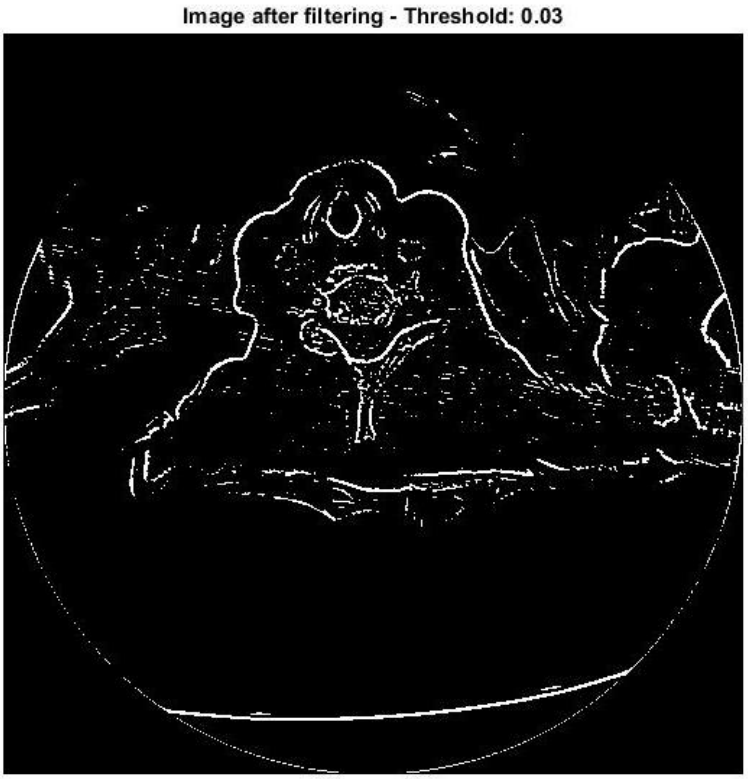

Figure 11: CT image after filtering with a threshold amplitude of 0.03

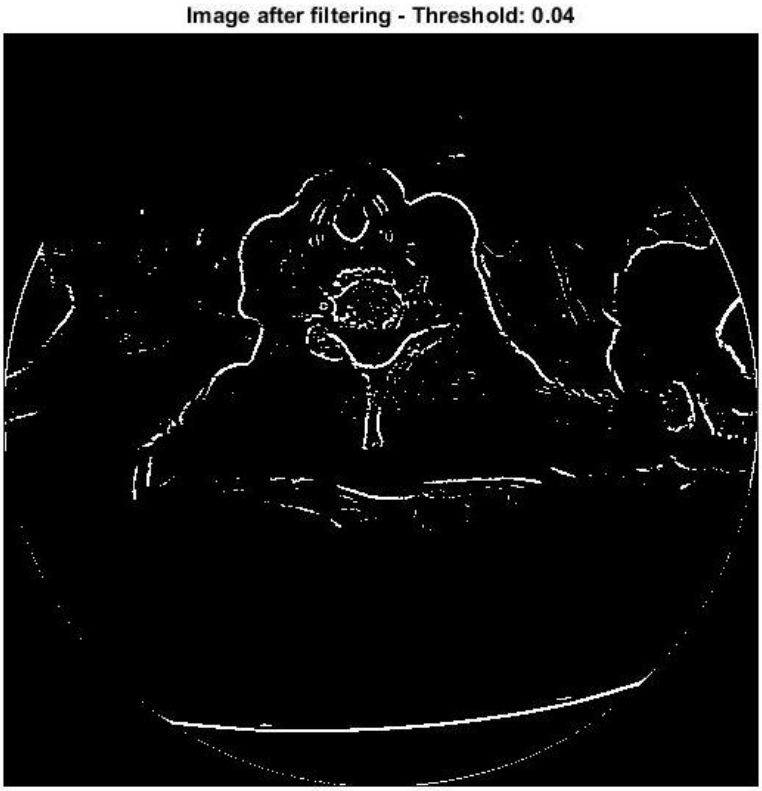

Figure 12: CT image after filtering with a threshold amplitude of 0.04

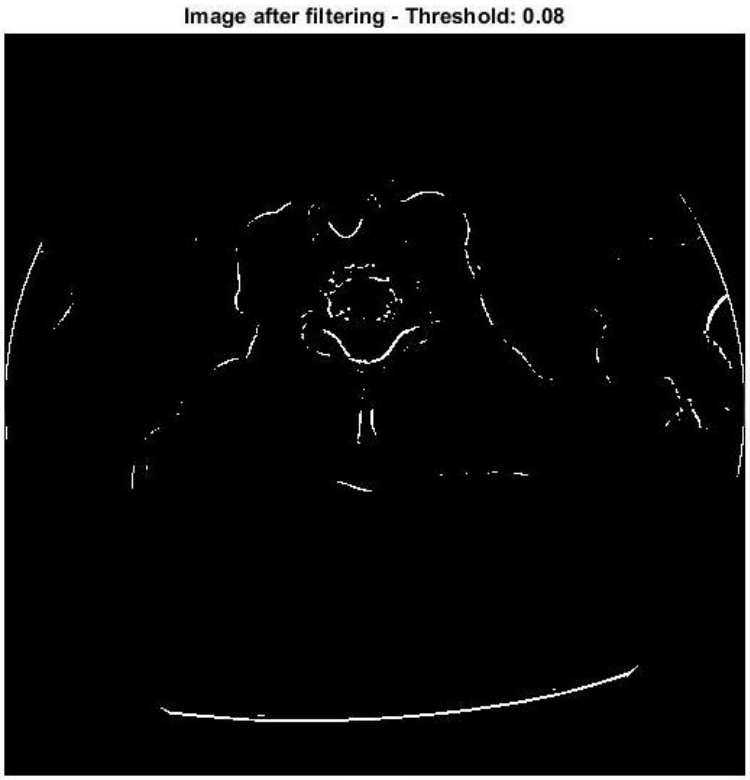

Figure 13: CT image after filtering with a threshold amplitude of 0.08

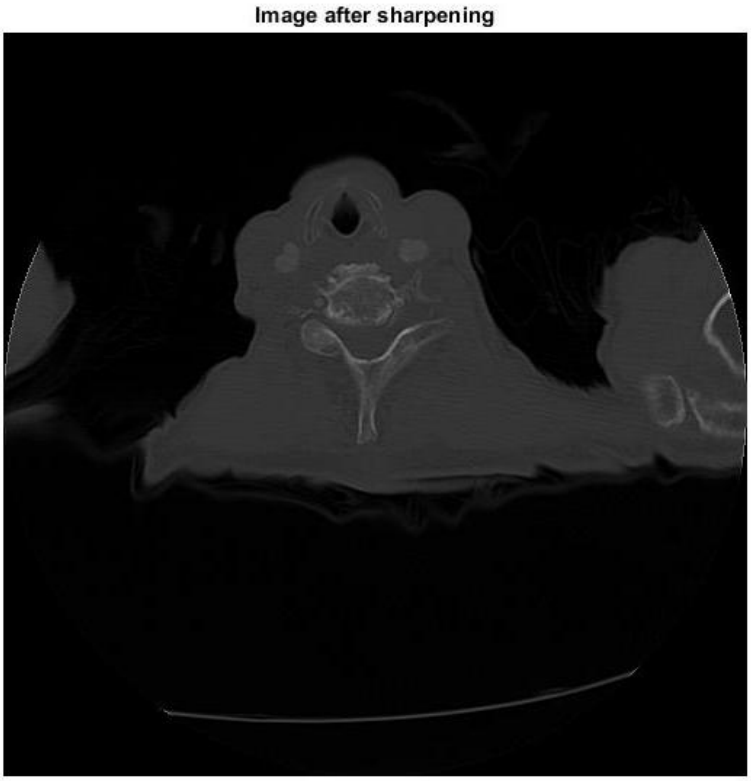

Figure 14: CT image after filtering through a sharpening kernel

Discussion:

The two filters do not appear to have incredibly different frequency amplitude spectra. The difference, though, is noticeable. The shape of the graphs is surprising. The graph of the frequency amplitude spectrum for the high-pass filter looks like that for a standard high-pass filter. However, the graph for the low-pass filter looks like that for a high-pass filter as well. Perhaps the input signal for determining the frequency amplitude spectrum is not correct for the low-pass filter. Perhaps the filter itself is not a true low-pass filter.

The variation of the filter kernel has a great influence on CT image filtration. The property of the kernel depends on the values in the various cells of the kernel. Using a high-pass filter for edge detection requires a 3x3 kernel whose elements sum to zero and are arranged such that the negative values surround the single, larger, positive value in the center. Using a 3x3 sharpening filter requires a kernel whose elements sum to one and are arranged in a plus formation with the larger positive number centered.

Thresholding and variation of the threshold value also affect the image of a CT. Without thresholding the filtration just brightens the image. Using a threshold allows for edge detection and exposure. With a greater threshold the edges become more defined until a point where the edges cannot be detected because the threshold is too high. The key is to find the sweet spot.

Conclusion:

Clearly, digital filtering is especially helpful in the process of signal processing. With the application of C programming and digital filters, biomedical signals can be cleaned and manipulated in order to gain a better understanding of what is going on, physiologically. In an EEG signal, a digital filter is able to get rid of a lot of the noise that occurs. In an image, a filtering kernel can expose the edges, reveal surfaces, and even sharpen the image. These abilities are extremely helpful in the field of biomedical engineering.

C Code:

```
/* 1DDigitalFiltering.cpp : Defines the entry point for the console application.
Peter Dobbs
BIEN 3200
Lab #6
22 November 2016
*/

#include "stdafx.h"
#include "fftfilter.h"
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

void fft(float *data, int n, int invrs); // fast fourier transform function
int highpass(float *xdata, int n); // high pass filter function
int lowpass(float *xdata, int n); // low pass filter function

int getImpulseResponse() {
	printf("\nStart of: getImpulseResponse()\n");
	int N = 512;
	static float impulse[512];
	impulse[0] = 1.0;

	if (lowpass(impulse, N) != 0) {
		return 1;
	} // impulse[] contains lowpass filter's impulse response function
	//OR   (must comment out one or the other)
	/*if (highpass(impulse, N) != 0) {
		return 1;
	}*/ // impulse[] contains highpass filter's impulse response function

	// output the impulse response function to a data file
	char fileout[200];
	FILE *output;
	printf("Enter output file name:\n");
	scanf("%s", &fileout);
	output = fopen(fileout, "w");
	for (int i = 0; i < N; ++i) {
		fprintf(output, "%f\n", impulse[i]);
	}
	fclose(output);

	system("PAUSE");
	return 0;
}

int getFrequencyAmplitudeSpectrum() {
	printf("\nStart of: getFrequencyAmplitudeSpectrum()\n");
	// DECLARE VARIABLES
	int fs; // sampling frequency
	int N; // number of points in file
	float t, ampl; // scrap variables for determining N
	float *time, *data; // float pointers for holding data from file
	FILE *fpIn, *fpOut; // file pointers for input and output files
```

```
char inputFile[100], outputFile[100]; // stores input/output file names

printf("Enter input file name (.txt)\n");
scanf("%s", &inputFile);
fpIn = fopen(inputFile, "r");

if (!fpIn) {
	printf("file does not exist in expected location\n");
	return 1;
}

for (N = 0; (fscanf(fpIn, "%f %f", &t, &ampl)) != EOF; ++N);
fs = N / t;
printf("Sampling Frequency: %f\n", fs);
rewind(fpIn);

int exp = 1;
while (pow(2, exp) < N) {
	exp++;
}

int N2 = pow(2, exp);
printf("Updated Number of Points: %i \n", N2);

time = (float *)calloc(sizeof(float), N2);// allocate memory
data = (float *)calloc(sizeof(float), N2);

for (int i = 0; i < N; ++i) {
	fscanf(fpIn, "%f %f", &time[i], &data[i]);
}
fclose(fpIn);
free(time);
free(data);

// calculate discrete fourier transform (dft)
fft(data - 1, N2, 1); //returns first n/2 coefficients

float *amplSpec; // allocate memory
amplSpec = (float *)calloc(sizeof(float), (N2 / 2));
float *kValues;
kValues = (float *)calloc(sizeof(float), (N2 / 2));

for (int k = 0, i = 2; k < (N2 / 2); k++) {
	// i is regular index skip
	amplSpec[k] = sqrt(pow(data[i], 2) + pow(data[i + 1], 2)) * 2.0 / (float)N;
	kValues[k] = ((float)k*fs) / (float)N2;
	i += 2;
}

// output data
printf("Enter output filename (.txt)\n");
scanf("%s", &outputFile);
fpOut = fopen(outputFile, "w");
for (int k = 0; k < (N2 / 2); k++) {
	fprintf(fpOut, "%f %f\n", kValues[k], amplSpec[k]);
}

free(amplSpec);
```

```
	free(kValues);
	fclose(fpOut);

	system("PAUSE");
	return 0;
}

int main() {
	printf("Start of: main - 1DDigitalFiltering.cpp\n");

	/*Determine the Impulse Response of the Filter*/
	//getImpulseResponse();

	/*Determine the Filter's Frequency Amplitude Spectrum*/
	//getFrequencyAmplitudeSpectrum();

	/*Filter a Given Signal*/
	printf("\nStart of: filtering input signal\n");
	// declare variables
	char unfilterdata[200], filterdata[200]; // store names of files with unfiltered
and filtered data
	FILE *datap, *dataf; // file pointers for unfiltered and filtered data
	float *time, *data; // float pointers for DMA to store the values of the
unfiltered signal
	float t, d; // scrap variables used in determining number of points in the file
	int N; // number of points in the file
	int fs; // sampling frequency

	// input data to filter
	printf("Enter input file name (.txt)\n");
	scanf("%s", &unfilterdata);
	datap = fopen(unfilterdata, "r");
	for (N = 0; fscanf(datap, "%f %f", &t, &d) != EOF; ++N);
	fs = N / t;
	printf("Sampling Frequency: %f\n", fs);
	rewind(datap);

	// check that signal length is power of 2
	int exp = 1;
	while (pow(2, exp) < N) {
		exp++;
	}
	int N2 = pow(2, exp);

	time = (float *)calloc(sizeof(float), N2); // allocate memory
	data = (float *)calloc(sizeof(float), N2);

	for (int i = 0; i < N; ++i) {
		fscanf(datap, "%f %f", &time[i], &data[i]);
	}
	fclose(datap);

	// Filter the Input Signal
	/*Note: Before call, 'data' stores unfiltered signal.
			After call, 'data' stores the filtered signal.*/
	if (lowpass(data, N) != 0) {
		return 1;
	} // impulse[] contains lowpass filter's impulse response function
```

```
	//OR   (must comment out one or the other)

	/*if (highpass(data, N) != 0) {
		return 1;
	}*/ // impulse[] contains highpass filter's impulse response function

	// Write the Resulting Input Signal to an Output File
	printf("Enter filtered data output file name (.txt)\n");
	scanf("%s", &filterdata);
	dataf = fopen(filterdata, "w");
	for (int i = 0; i < N; ++i) {
		fprintf(dataf, "%f\t%f\n", time[i], data[i]);
	}
	fclose(dataf);
	free(time);
	free(data);

	// FIND THE AMPLITUDE FREQUENCY SPECTRUM FOR THE FILTERED DATA
	// calculate discrete fourier transform (dft)
	fft(data - 1, N2, 1); //returns first n/2 coefficients

	float *amplSpec; // allocate memory
	amplSpec = (float *)calloc(sizeof(float), (N2 / 2));
	float *kValues;
	kValues = (float *)calloc(sizeof(float), (N2 / 2));

	for (int k = 0, i = 2; k < (N2 / 2); k++) {
		amplSpec[k] = sqrt(pow(data[i], 2) + pow(data[i + 1], 2)) * 2.0 / (float)N;
		kValues[k] = ((float)k*fs) / (float)N2;
		i += 2; // i is index skip
	}

	// output data
	FILE *fpOut; // file pointer for output data set
	char outputFile[200]; // stores output filename
	printf("Enter DFT of filtered data output filename (.txt)\n");
	scanf("%s", &outputFile);
	fpOut = fopen(outputFile, "w");
	for (int k = 0; k < (N2 / 2); k++) {
		fprintf(fpOut, "%f %f\n", kValues[k], amplSpec[k]);
	}
	free(amplSpec);
	free(kValues);
	fclose(fpOut);

	printf("\nDONE\n");
	system("PAUSE");
	return 0;
}
```

```
/* 2-D filtering of a CT image */
//Image_filter.cpp
#pragma warning (disable: 4996)
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void) {

	int i, j, k, l, m, n;
	float *tmp, *out;
	float sum;
	char in_file[100], out_file[100];
	FILE *inputfile, *outfile;
	//Edge detection kernel
	float filt[3][3] = {
		{0,-1,0},
		{-1,5,-1},
		{0,-1,0}
	};


	// low pass filter
	/*float wt = 1.0 / 25.0;
	float filt[5][5] = {
		{wt,wt,wt,wt,wt},
		{wt,wt,wt,wt,wt},
		{wt,wt,wt,wt,wt},
		{wt,wt,wt,wt,wt},
		{wt,wt,wt,wt,wt}
	};*/

	// Open input file (unfiltered data)
	printf("Enter input file name\n");
	scanf("%s", in_file);
	inputfile = fopen(in_file, "r");
	// Check for file existance
	if (!(inputfile)) {
		printf("File does not exist.\n");
		exit(EXIT_FAILURE);
	} else {
		printf("Enter the number of rows:\n");
		scanf("%i", &m);
		printf("Enter the number of columns:\n");
		scanf("%i", &n);

		//dynamic memory allocation to store data
		tmp = (float *)calloc(sizeof(float), m*n);
		out = (float *)calloc(sizeof(float), m*n);

		// read in data from file
		for (i = 0; i < m*n; i++) {
			fscanf(inputfile, "%f", (tmp + i));
			*(out + i) = *(tmp + i);
		}
		fclose(inputfile);
```

```
		//convolution with filter kernel

		for (i = 1; i < n - 1; i++) {
			for (j = 1; j < m - 1; j++) {
				sum = 0.0;
				for (k = 0; k < 3; k++) {
					for (l = 0; l < 3; l++) {
						sum = sum + filt[k][l] * (*(tmp + (i - 1 + k)*m
+ (j - 1 + l)));
					}
				}
				//thresholding
				/*if (sum >= 0.03) {
					sum = 1.0;
				} else {
					sum = 0.0;
				}*/

				*(out + i*m + j) = sum;
			}

		}

		//Save filtered data to a .txt file

		printf("Enter output file name\n");
		scanf("%s", out_file);
		outfile = fopen(out_file, "w");

		//Proper formating
		for (i = 0; i < n; i++) {
			for (j = 0; j < m; j++) {
				fprintf(outfile, "%g ", *(out + i*m + j));
			}
			fprintf(outfile, "\n");
		}

		fclose(outfile);

		free(tmp);
		free(out);
	}
	system("PAUSE");
	return 0;
}
```

```
/* Plotting.cpp : Defines the entry point for the console application.
              Peter Dobbs
              BIEN 3200 - Section 401
              22 November 2016
              Lab 6

              Program Description:
                     This program reads in data file and plots the data in MatLab
*/

#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "engine.h"
#include "mex.h"

int main(void) {
       int numPoints;
       float time, amplitude;
       char in_file[100];
       FILE *inputFile;

       printf("Enter input file name\n"); // open file to be read
       scanf("%s", in_file);
       inputFile = fopen(in_file, "r");

       if (!(inputFile)) {
              printf("file does not exist.\n");
              exit(EXIT_FAILURE);
       }

       // determine the number of points in the opened file
       for (numPoints = 0; fscanf(inputFile, "%f %f", &time, &amplitude) != EOF;
++numPoints);
       printf("number of data points in file = %i\n", numPoints);
       float samplingFrequency = numPoints / time;

       rewind(inputFile);

       float *tData = (float *)calloc(sizeof(float), numPoints); //allocate memory
       float *aData = (float *)calloc(sizeof(float), numPoints);

       for (int i = 0; i < numPoints; ++i) {
              fscanf(inputFile, "%f\t%f", &tData[i], &aData[i]);
       }
       fclose(inputFile);

       system("PAUSE");
       ///////////////====MATLAB====/////////////////
       Engine *ep;
       ep = engOpen(NULL);

       mxArray* matX1 = mxCreateNumericMatrix(numPoints, 1, mxSINGLE_CLASS, mxREAL);
       mxArray* maty1 = mxCreateNumericMatrix(numPoints, 1, mxSINGLE_CLASS, mxREAL);

       memcpy((void*)mxGetPr(matX1), (void*)tData, sizeof(float) * numPoints);
```

```
        memcpy((void*)mxGetPr(maty1), (void*)aData, sizeof(float) * numPoints);

        engPutVariable(ep, "t", matX1);
        engPutVariable(ep, "a", maty1);

        engEvalString(ep, "figure; plot(t,a);"
               "title ('DFT of EEG_0');"
               "xlabel('Frequency (Hz)');"
               "ylabel('Signal Magnitude');");

        system("pause"); //Pauses figure
        engClose(ep);
        //////////////======C======//////////////////

        free(aData);
        free(tData);

        system("PAUSE");
        return 0; //end program
}
```

CT image filtering using 2-D arrays