

BIEN 3200: Lab 5
Laboratory Exercise #5
Peter Dobbs
Lab Section 401
Nicholas Heugel
8 November 2016

Abstract:

In the field of biomedical engineering, it is extremely important to have a knowledge of system analysis. In this lab, a sine wave generator is implemented and used to create a messy signal. A program that finds the frequency amplitude spectrum for a given data set is generated and implemented. A third and final program (CC) is developed to collect data from a function generator. Through this exercise, the author practices programming, creation of detection algorithms, and analytical technical writing.

Introduction:

Techniques, such as discrete Fourier Transform, fast Fourier Transform, and analog-to-digital (A/D) signal conversion, are necessary for properly understanding a signal. This exercise was performed to introduce students to techniques for system analysis and signal processing.

Through the writing of this lab report, technical writing skills are reinforced.

Data Files:

Afib.txt

Contains 4 seconds of electrocardiogram (ECG) data, sampled at 250 Hz, collected from a patient in atrial fibrillation

Sinus.txt

Contains 4 seconds of electrocardiogram (ECG) data, sampled at 250 Hz, collected from a patient exhibiting normal sinus rhythm

EEG_0.txt

Contains 1 second of electroencephalogram (EEG) data, sampled at 500 Hz, collected from a rat responding to flash stimulation following exposure to halothane, an anesthetic, at 0.0% concentration

EEG_05.txt

Contains 1 second of electroencephalogram (EEG) data, sampled at 500 Hz, collected from a rat responding to flash stimulation following exposure to halothane, an anesthetic, at 0.5% concentration

Methods:

The sine wave generation program (program #1) created a signal based on the sum of five sine waves with varying frequency and amplitude. This program produced multiple data files

which are visualized in figures 3, 5, and 7. The frequency amplitude spectrum program (program #2) determines the spectrum of frequency amplitudes for a given signal by performing a Fourier Transform on the data and applying the results of the transform to a high pass filter. The data is read into dynamically allocated memory and then sent to the `fft` function, as defined in the header file *fftfilter.h*.

The third program, from part 2 of the lab, consists of an A/D converter, which simply required filling in the blanks. The A/D data acquisition card is a National Instruments USB DAQ, USB 6008, connected to channel 0 via a USB cable. 12 bits are used for the conversion. The code must be adjusted to reflect the ability of user input to affect the course of the code flow. Parameters for the converter are set based on user input. But, there are configurations for the channels. The amplitude input ranges from -10V to +10V. Channel 0 is set to unipolar configuration. There should be zero DC offset. The adjustment in signal gain depends on the sampling rate. The sampling rate and sampling duration depend on the user input for those values.

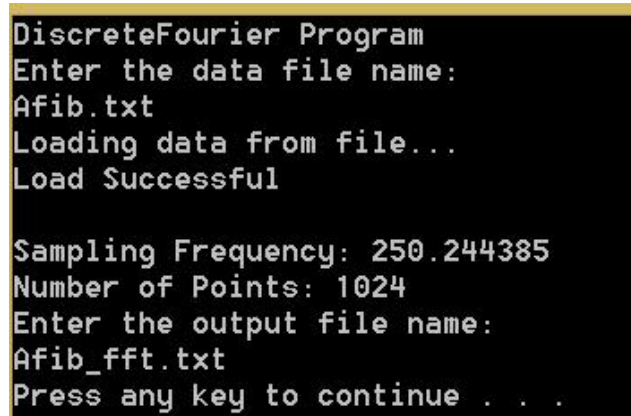
Results:

Screenshots of program output

A screenshot of a terminal window with a black background and white text. The text reads: "SineGeneration" followed by "Press any key to continue . . .".

```
SineGeneration
Press any key to continue . . .
```

Figure 1: Screenshot of the output for program #1

A screenshot of a terminal window with a black background and white text. The text reads: "DiscreteFourier Program", "Enter the data file name:", "Afib.txt", "Loading data from file...", "Load Successful", "Sampling Frequency: 250.244385", "Number of Points: 1024", "Enter the output file name:", "Afib_fft.txt", and "Press any key to continue . . .".

```
DiscreteFourier Program
Enter the data file name:
Afib.txt
Loading data from file...
Load Successful

Sampling Frequency: 250.244385
Number of Points: 1024
Enter the output file name:
Afib_fft.txt
Press any key to continue . . .
```

Figure 2: Screenshot of the output for program #2

Part 1 Plots

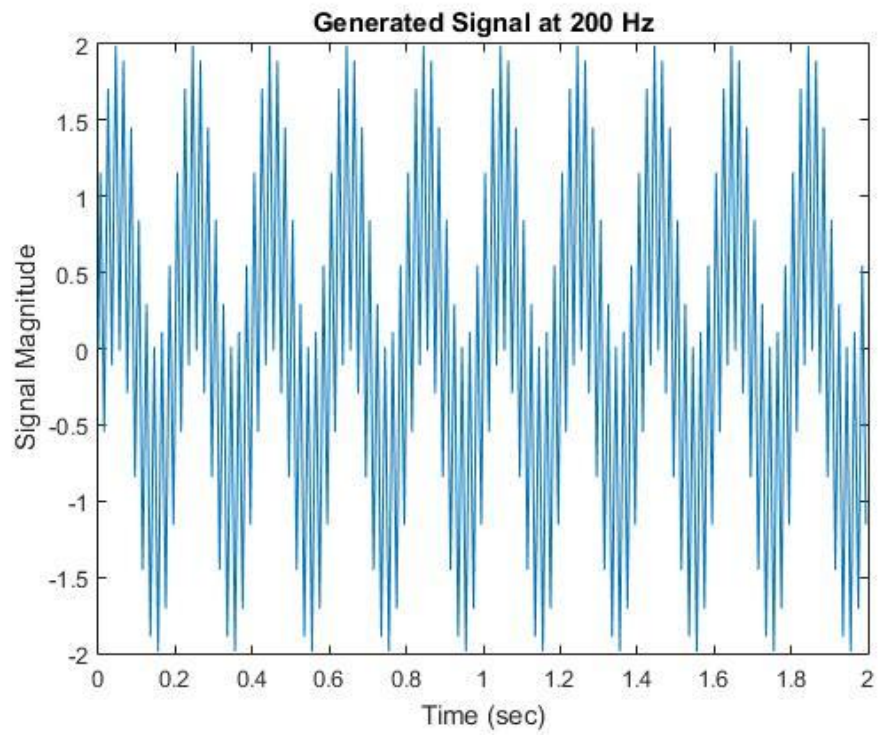


Figure 3: Plot of the sum of five independently calculated sine waves generated at 200 Hz

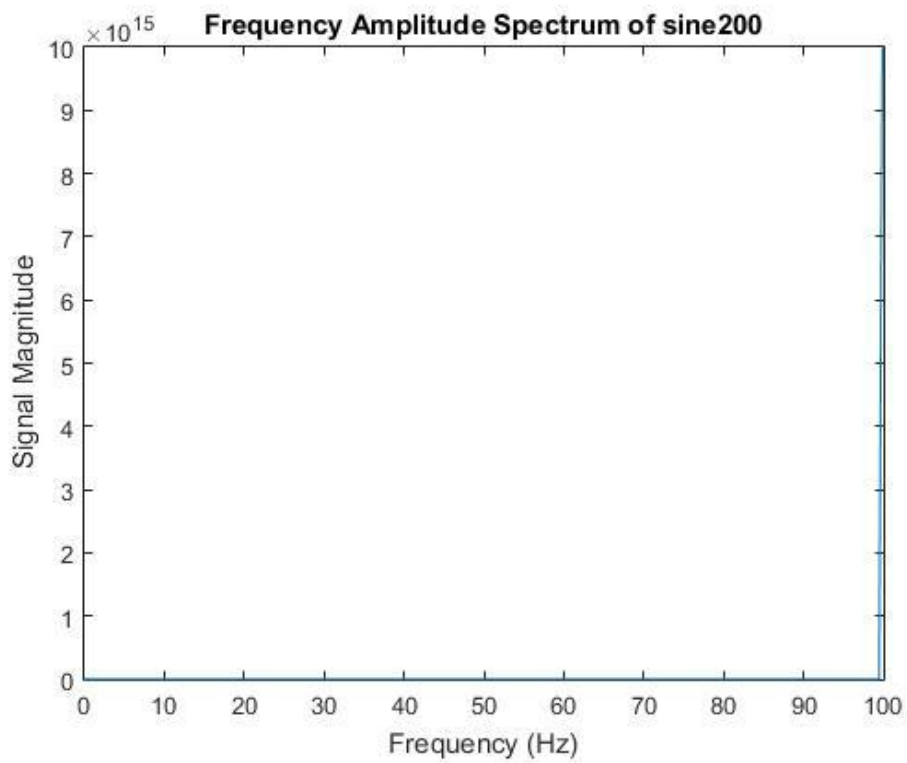


Figure 4: Plot of the frequency domain for Figure 3

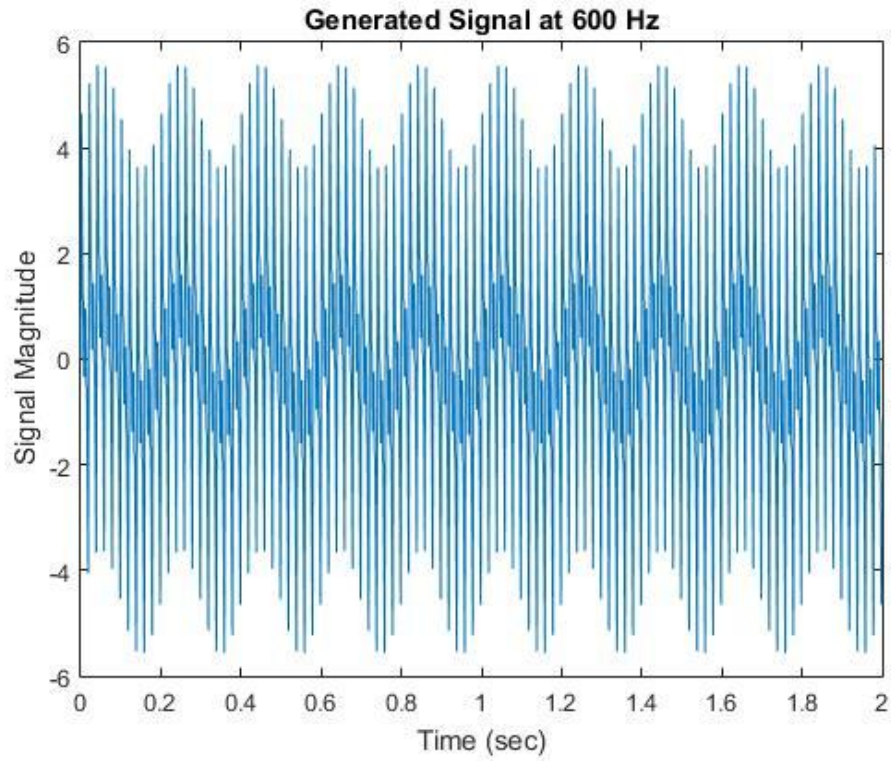


Figure 5: Plot of the sum of five independently calculated sine waves generated at 600 Hz

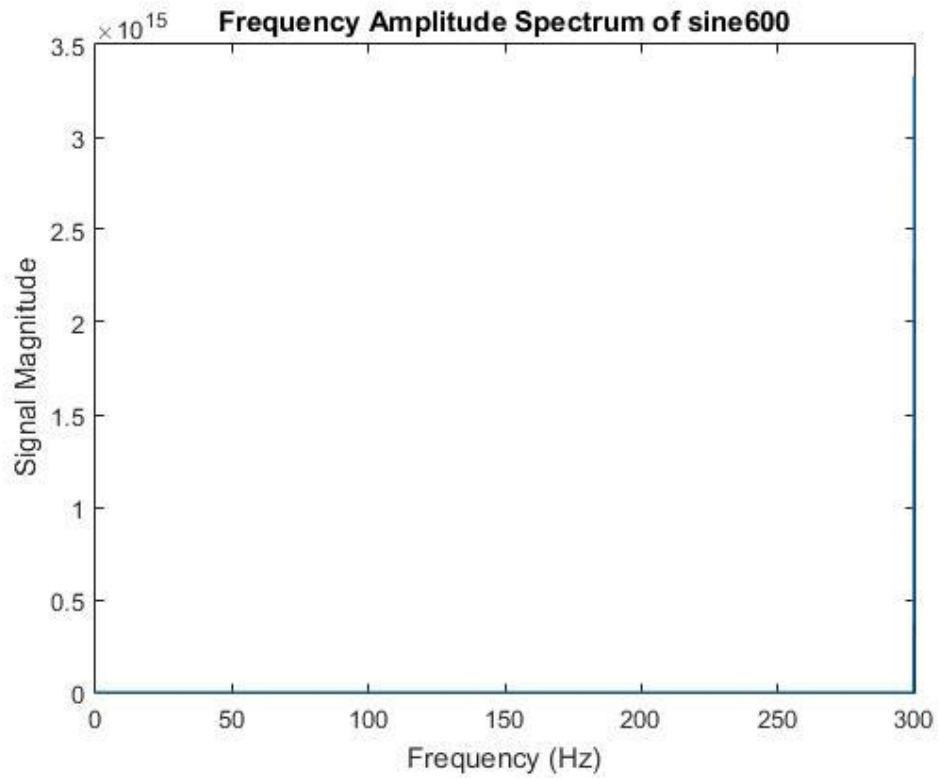


Figure 6: Plot of the frequency domain for Figure 5

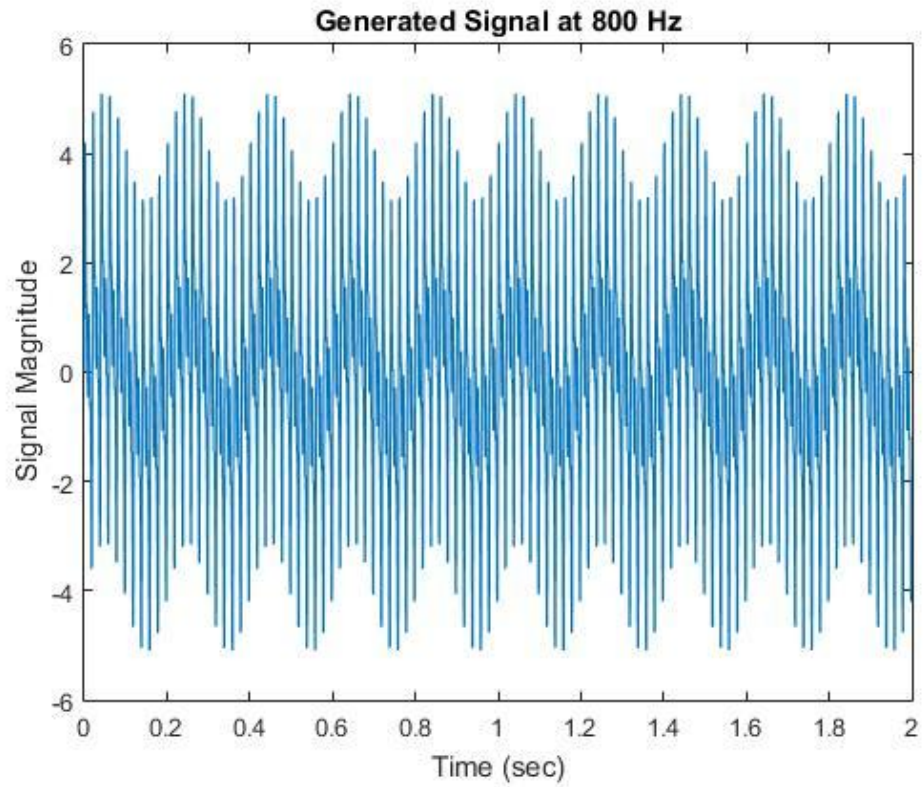


Figure 7: Plot of the sum of five independently calculated sine waves generated at 800 Hz

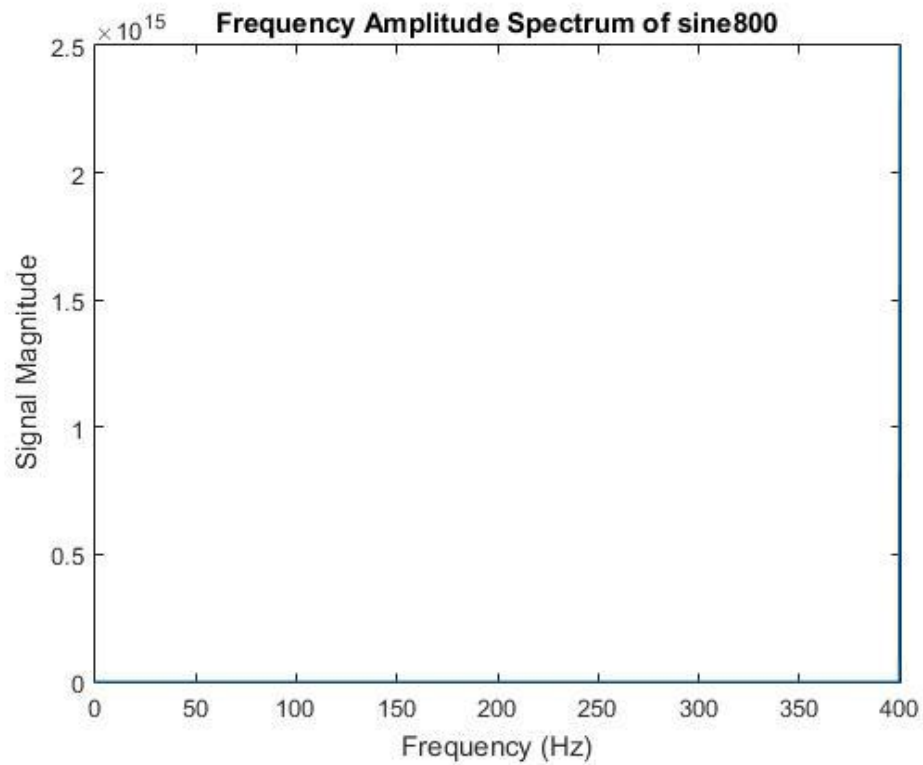


Figure 8: Plot of the frequency domain for Figure 7

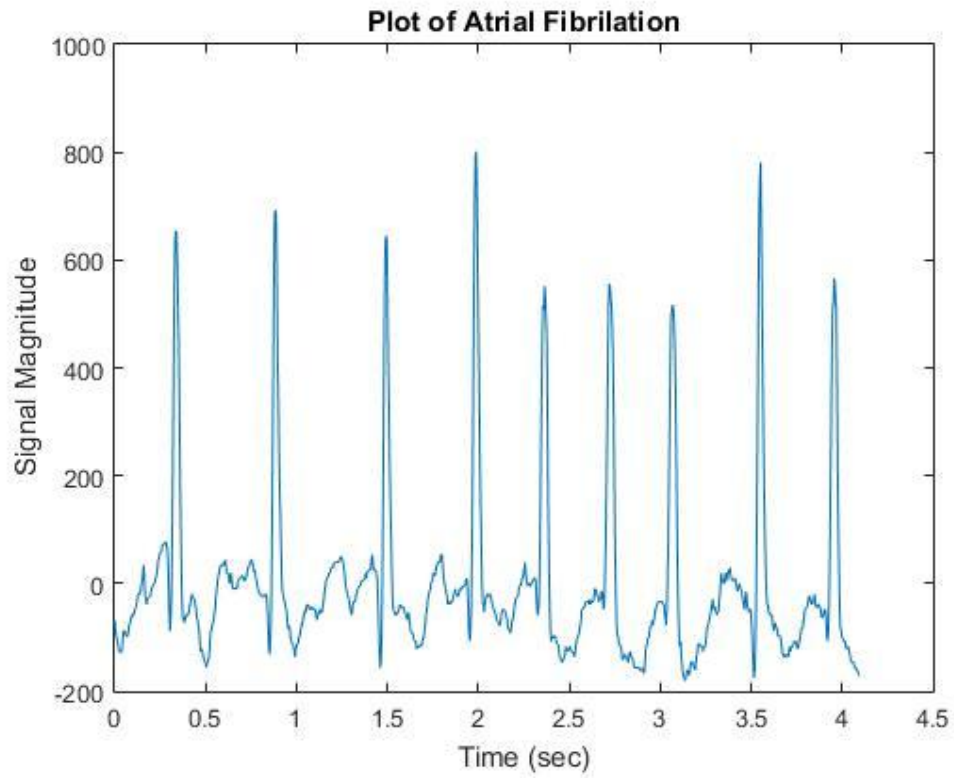


Figure 9: Plot of ECG data for a patient experiencing atrial fibrillation

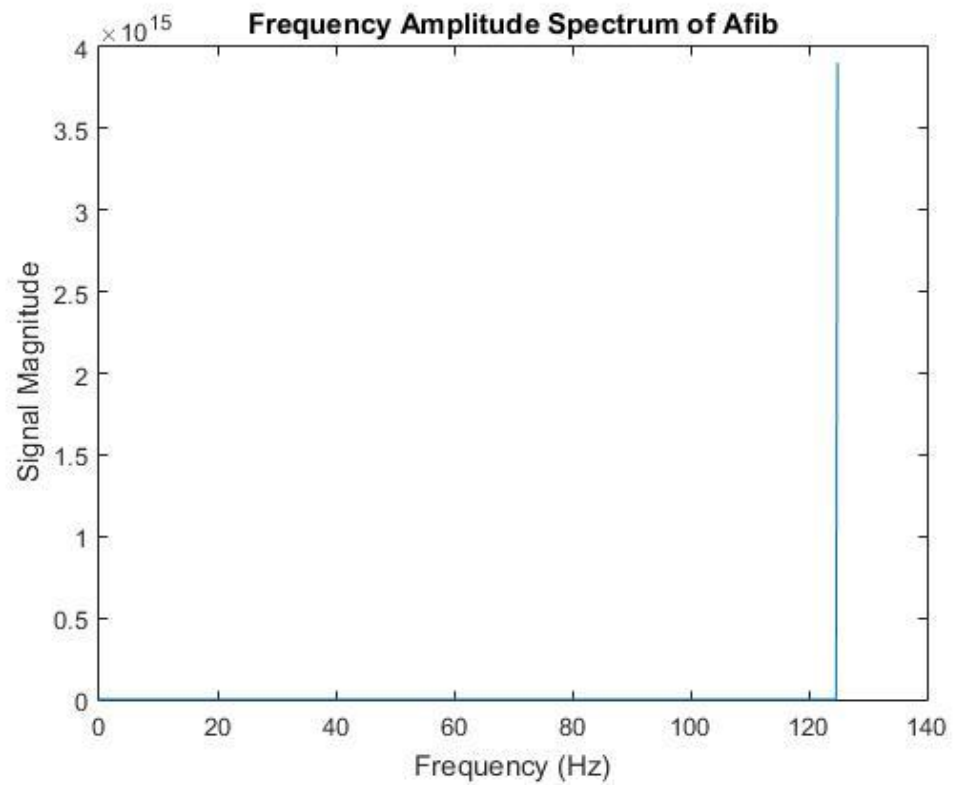


Figure 10: Plot of the frequency domain for Figure 9

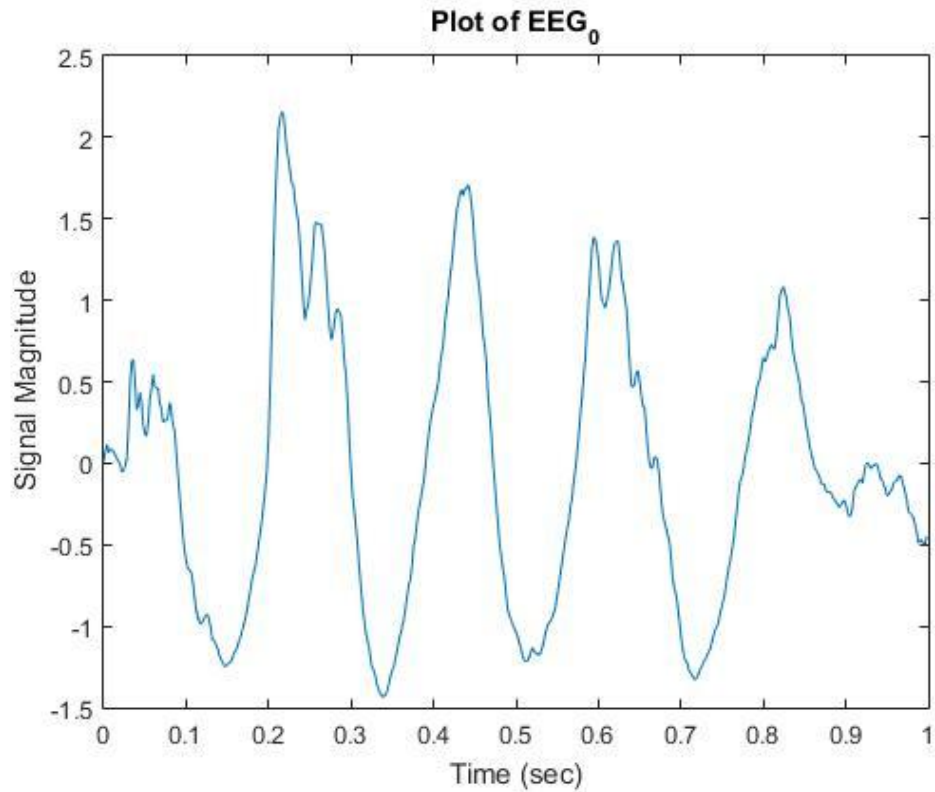


Figure 11: Plot of EEG data for slightly irregular brain activity

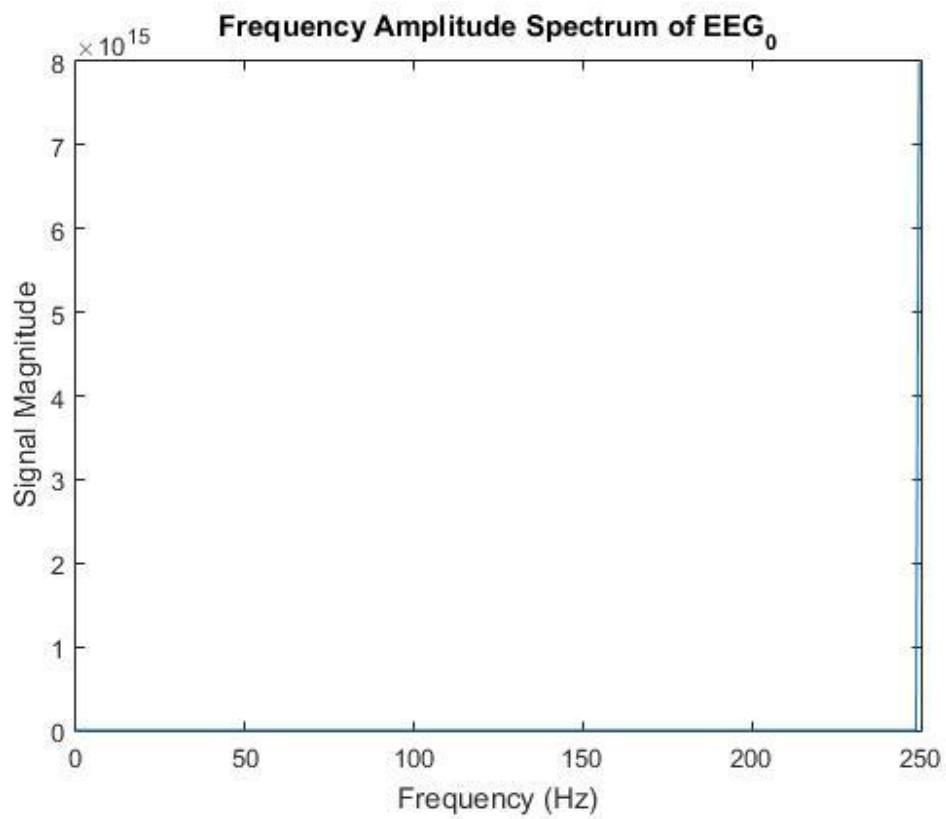


Figure 12: Plot of the frequency domain for Figure 11

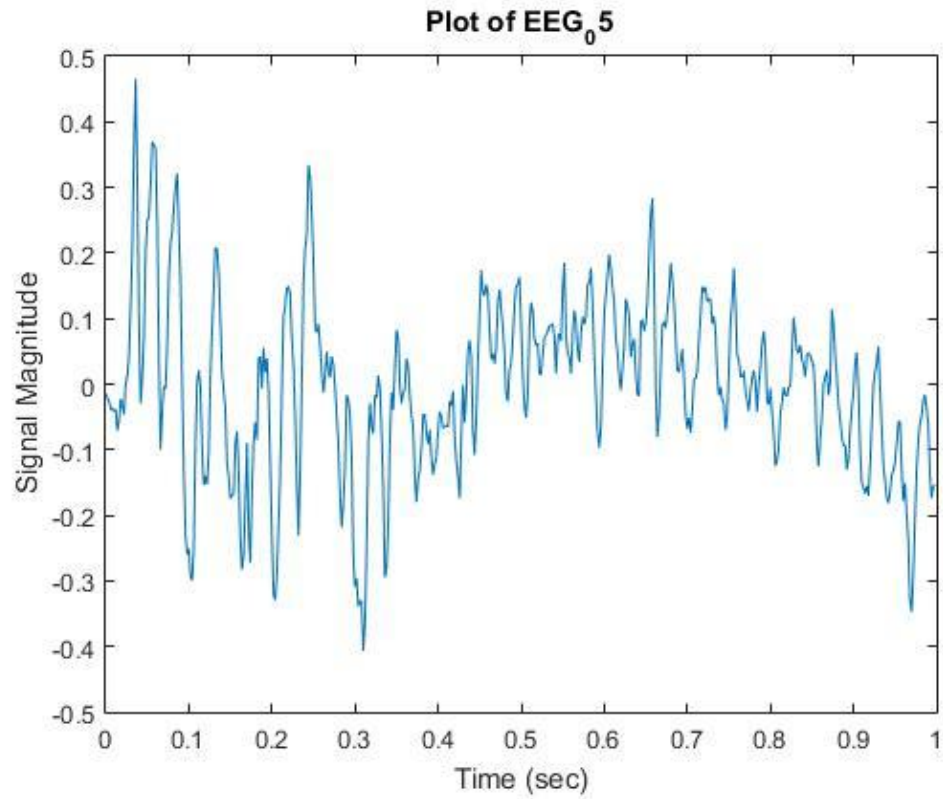


Figure 13: Plot of irregular, highly active brain activity from an EEG recording

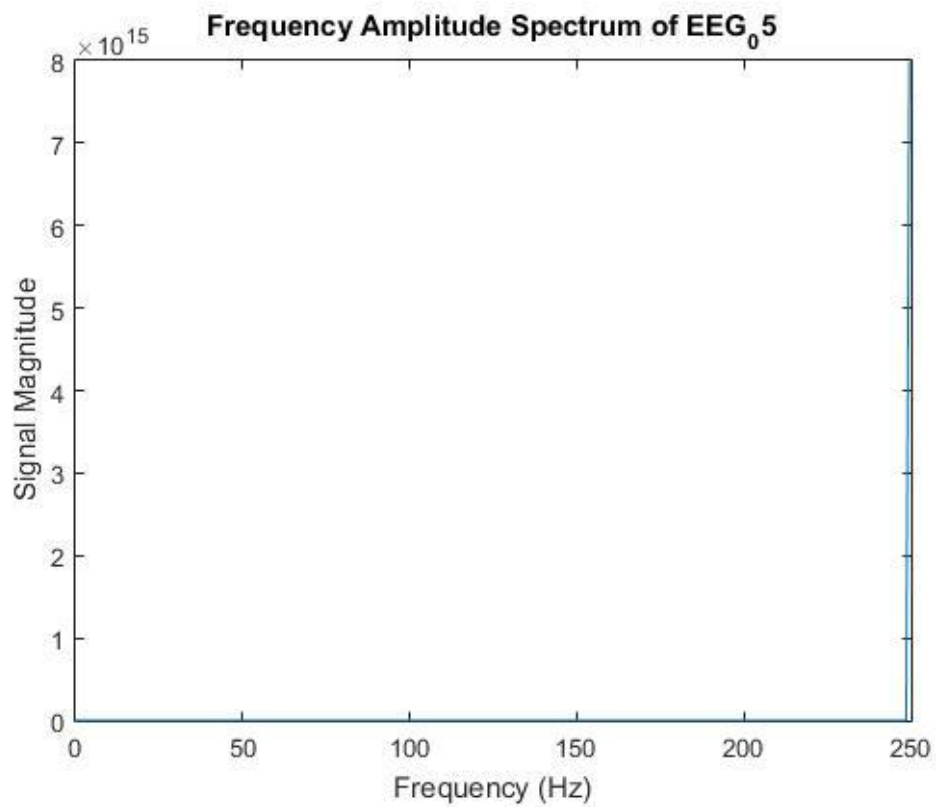


Figure 14: Plot of the frequency domain for Figure 13

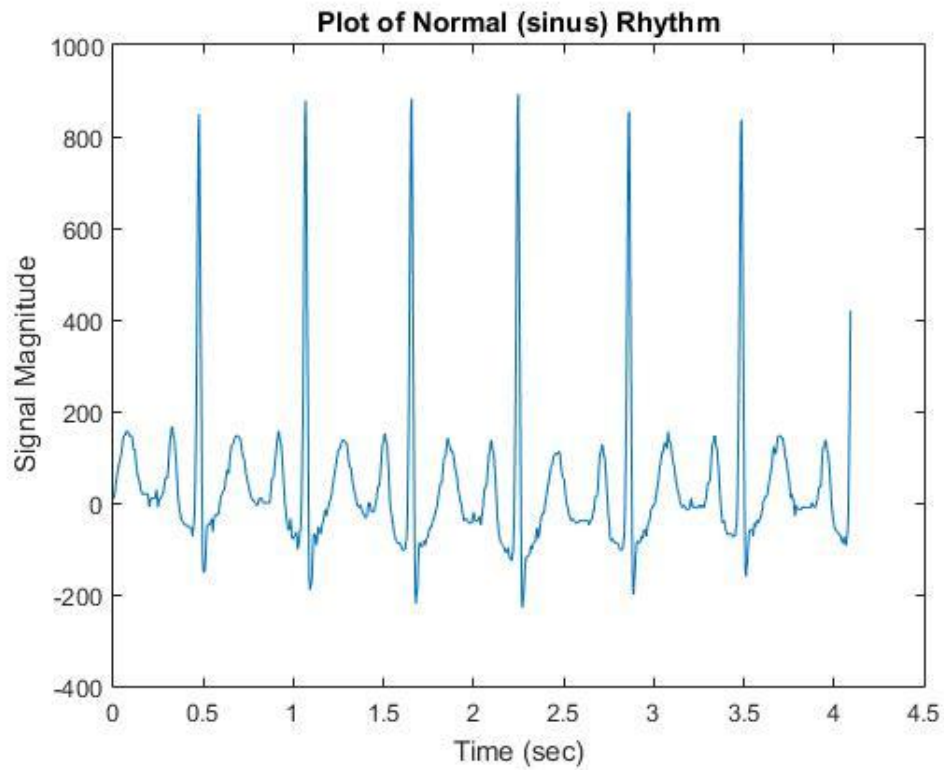


Figure 15: Plot of a patient ECG recording during normal sinus rhythm

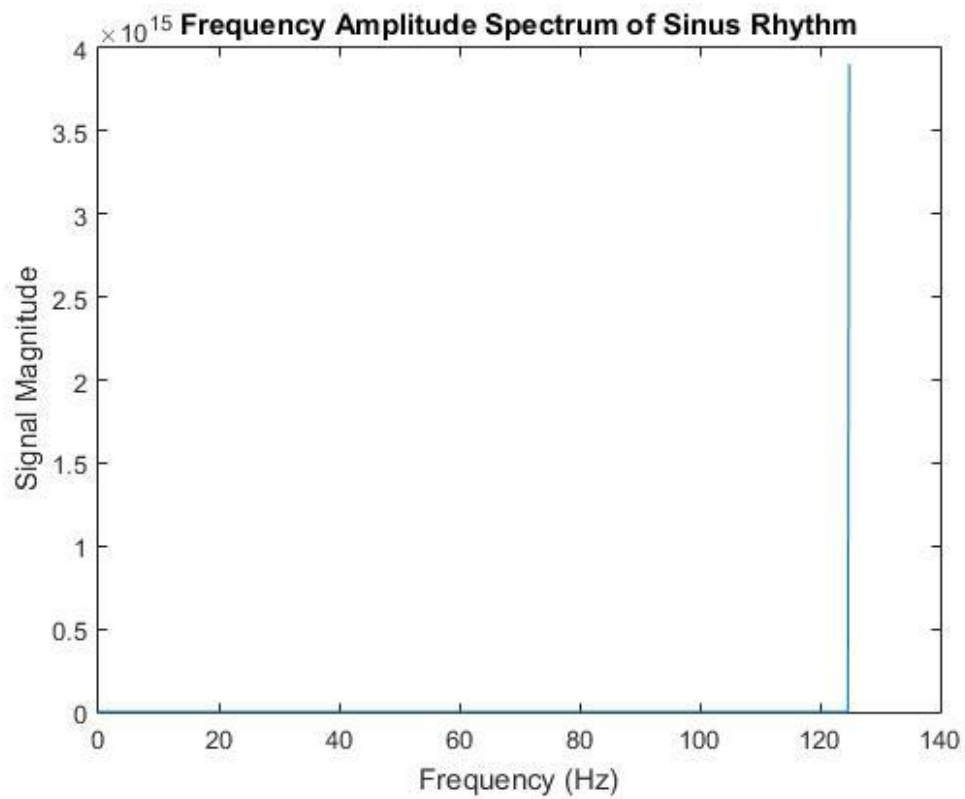


Figure 16: Plot of the frequency domain for Figure 15

Part 2 Plots

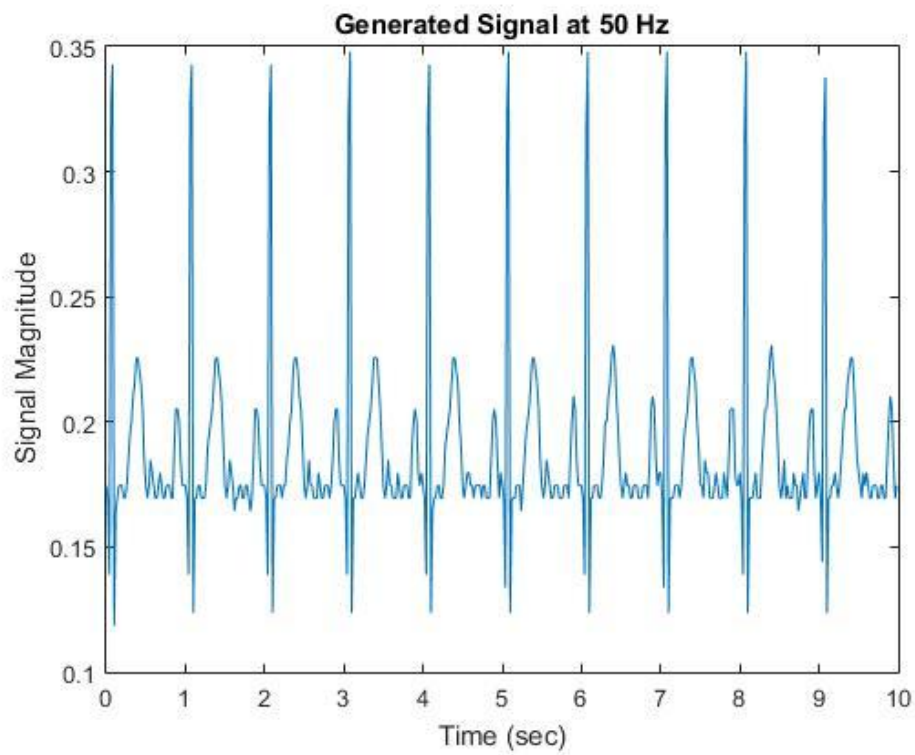


Figure 17: Plot of data acquired from the A/D converter at 50 Hz

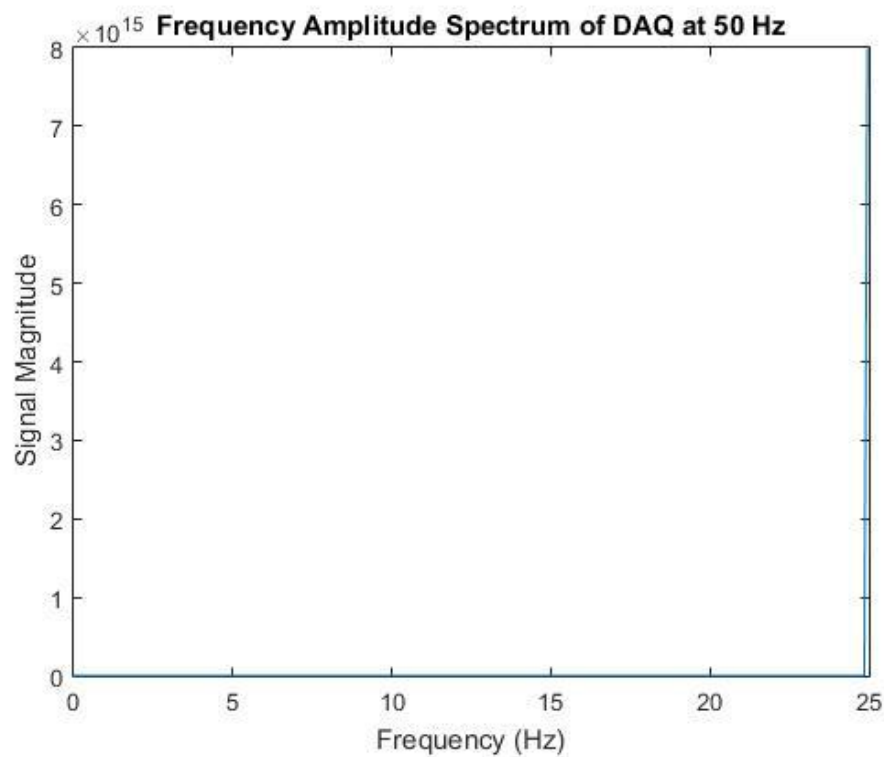


Figure 18: Plot of the frequency domain for Figure 17

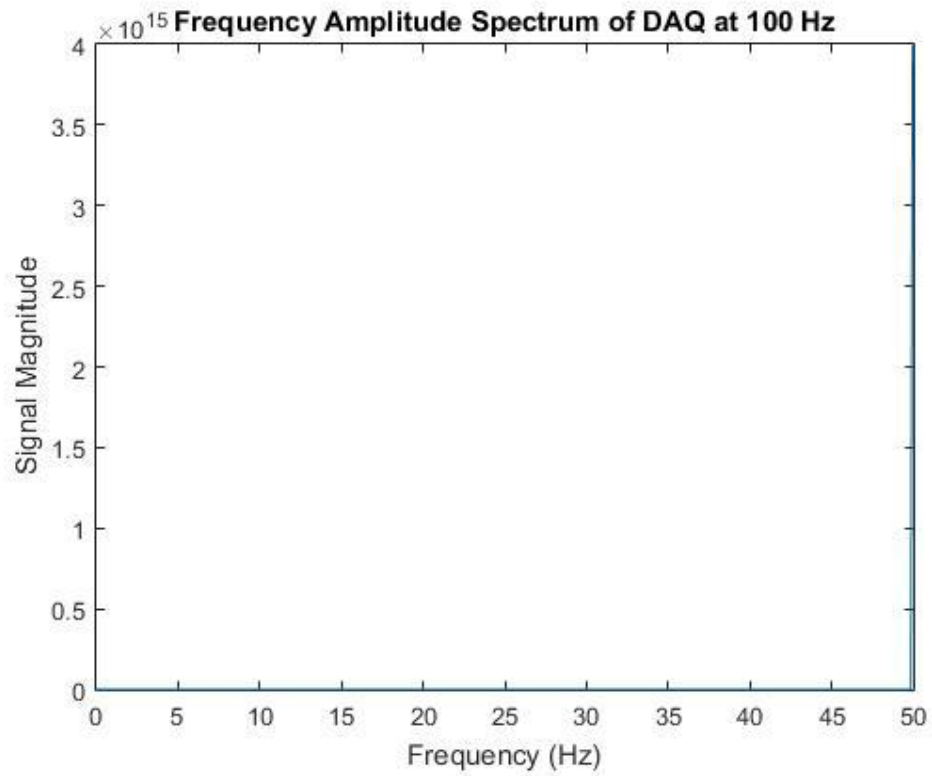


Figure 19: Plot of data acquired from the A/D converter at 100 Hz

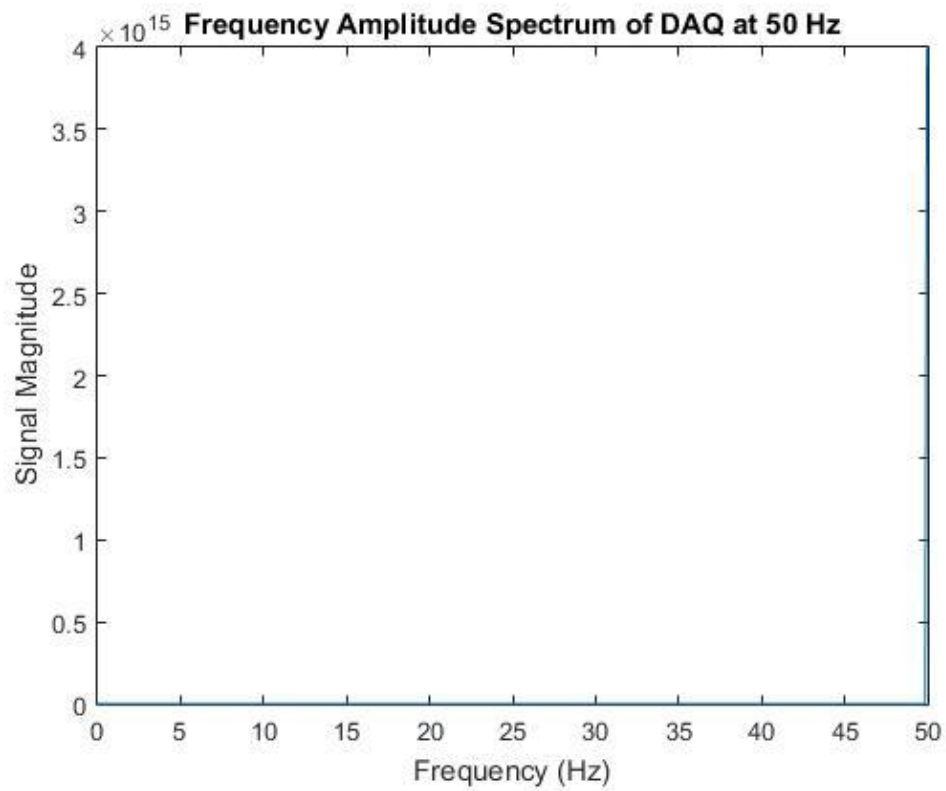


Figure 20: Plot of the frequency domain for Figure 19

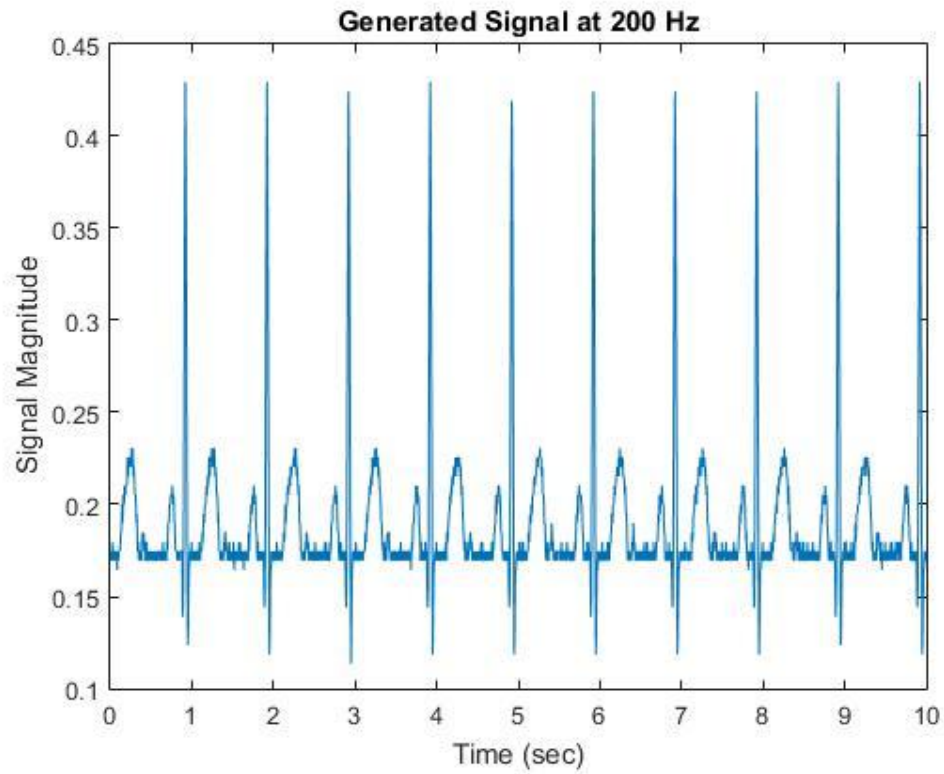


Figure 21: Plot of data acquired from the A/D converter at 200 Hz

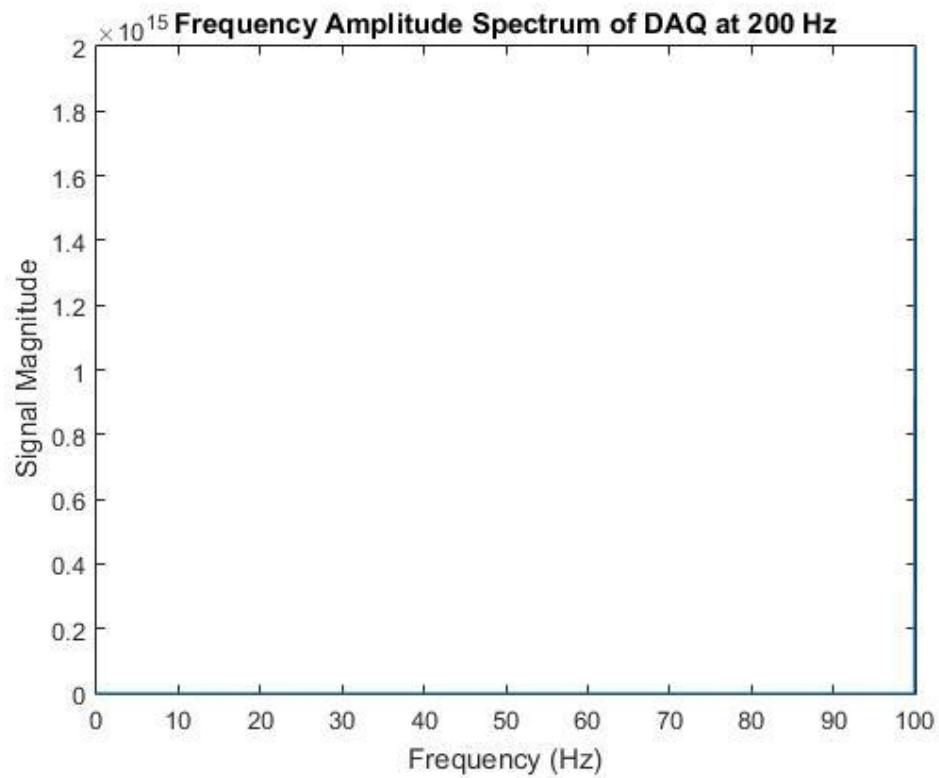


Figure 22: Plot of the frequency domain for Figure 21

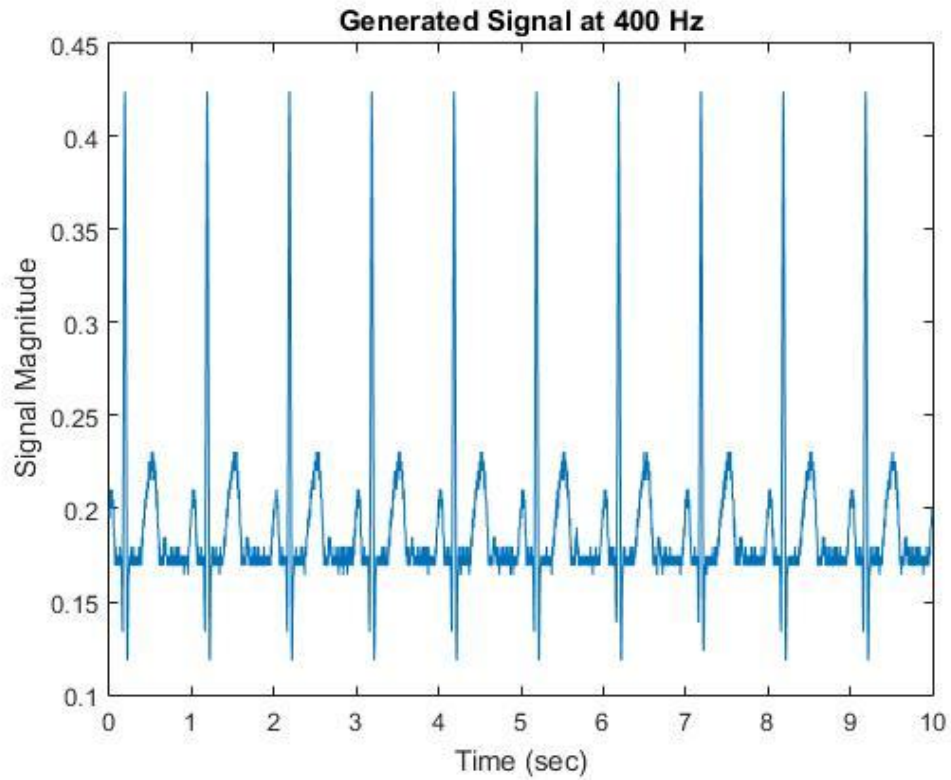


Figure 23: Plot of data acquired from the A/D converter at 400 Hz

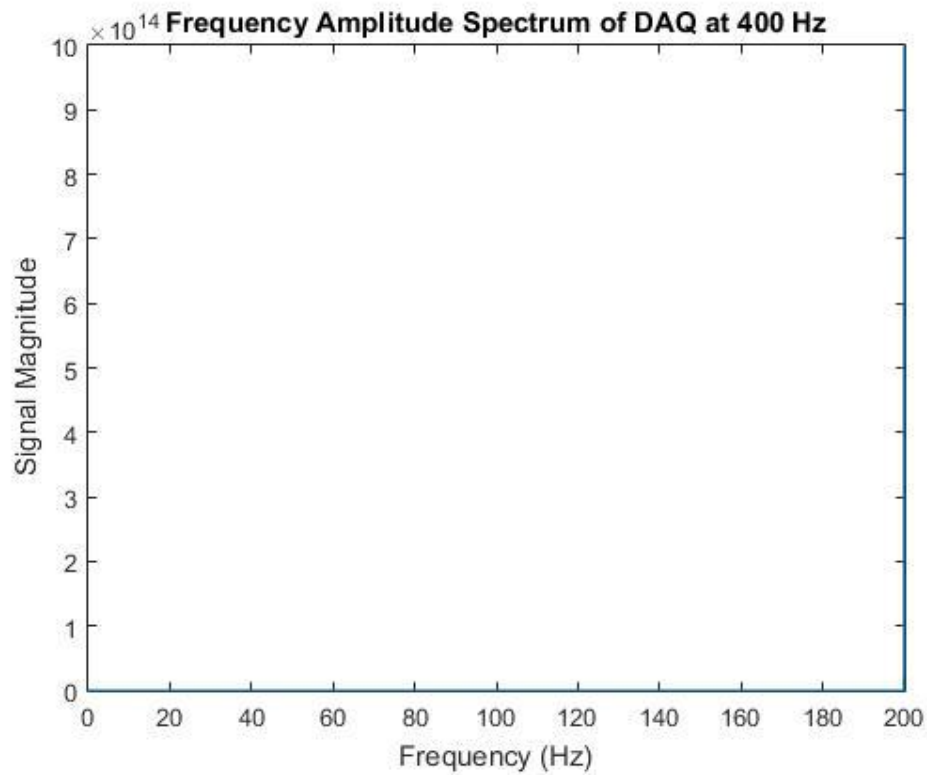


Figure 24: Plot of the frequency domain for Figure 23

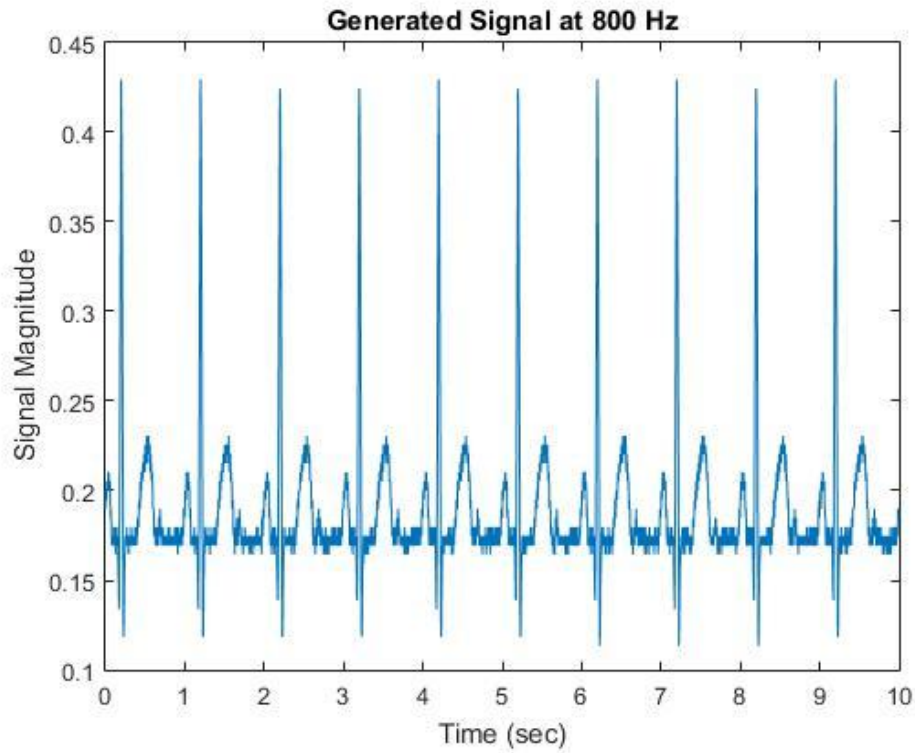


Figure 25: Plot of data acquired from the A/D converter at 800 Hz

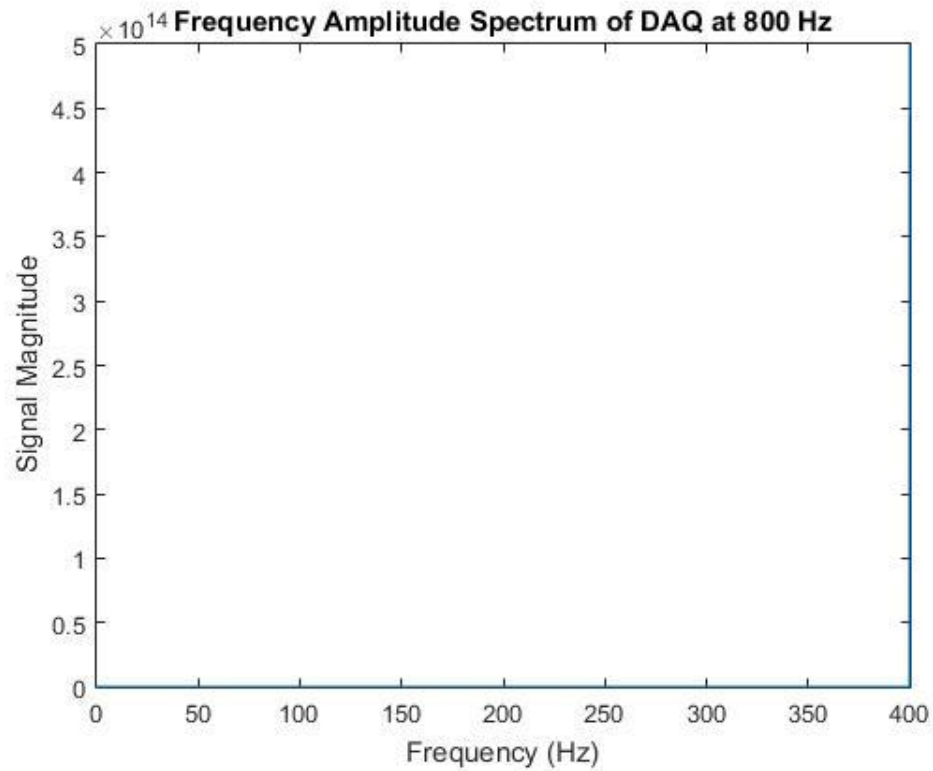


Figure 26: Plot of the frequency domain for Figure 25

Discussion:

The objectives of familiarizing and exposing students to Fourier Transforms, LTI systems, A/D conversion, and the Nyquist criterion were all met. Three programs were implemented that demonstrate the extent to which students understand the concepts of this lab. Though, these results are not all correct, the work put into developing the current solution was rewarding in meeting the objectives.

According to the results for the sine generation project, the frequency content changes with the input sample frequency. While this observation may not be true or valid, due to problems with the code, the published results show that there is a relationship between sampling frequency and the resulting content spectrum.

Conclusion:

While there is yet more work to do in understanding the concepts placed forth in this lab, a greater familiarization with the ideas was achieved.

C Code:

```

/*
Peter Dobbs
BIEN 3200
8 November 2016
Lab 5: Fourier Analysis, Data Aquisition, A/D conversion

Program Description:
    Generates a data set containing the sum of five sine waves with varying
    frequencies and amplitudes
*/

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

/*
Function to generate sine wave with given
length (seconds), fundamental frequency (Hz),
amplitude (mV), and sampling frequency (Hz)
*/
void sine(float *wave, float length, float funFrequency, float amplitude, float
samplingFrequency) {
    for (int i = 0; i < length; ++i) {
        wave[i] = amplitude * sin(i * funFrequency * (2 * 3.14159) /
samplingFrequency);
    }
}

void main() {
    printf("SineGeneration\n");
    // declare/initialize variables
    float samplingFrequency = 800;
    float length = samplingFrequency * 2;

    // generate sine waves
    float *sin_5_1 = (float*)calloc(sizeof(float), length);
    sine(sin_5_1, length, 5, 1, samplingFrequency);
    float *sin_50_2 = (float*)calloc(sizeof(float), length);
    sine(sin_50_2, length, 50, 2, samplingFrequency);
    float *sin_100_2 = (float*)calloc(sizeof(float), length);
    sine(sin_100_2, length, 100, 2, samplingFrequency);
    float *sin_150_1 = (float*)calloc(sizeof(float), length);
    sine(sin_150_1, length, 150, 1, samplingFrequency);
    float *sin_200_1 = (float*)calloc(sizeof(float), length);
    sine(sin_200_1, length, 200, 1, samplingFrequency);

    // sum of sine waves
    float *sum = (float*)calloc(sizeof(float), length);
    for (int i = 0; i < length; ++i) {
        sum[i] = sin_5_1[i] + sin_50_2[i] + sin_100_2[i] + sin_150_1[i] +
sin_200_1[i];
    }
}

```

```

    // write resulting waveform to file
    FILE *fp = fopen("out800.txt", "w");
    for (int i = 0; i < length; ++i) {
        float time = i / samplingFrequency;
        fprintf(fp, "%f\t%f\n", time, sum[i]);
    }
    fclose(fp);
    free(sum);
    free(sin_5_1);
    free(sin_50_2);
    free(sin_100_2);
    free(sin_150_1);
    free(sin_200_1);

    system("PAUSE");
}

/* DiscreteFourier.cpp : Defines the entry point for the console application.

Peter Dobbs
BIEN 3200
Lab 5
8 November 2016

Program Description:

*/

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

int numPoints;
void fft(float *data, int n, int invrs); // fast fourier transform function
int main() {
    printf("DiscreteFourier Program\n");

    char inFile[100];
    FILE *in;
    char outFile[100];
    FILE *out;
    float t, ampl;
    int exp; // buffer for size of power of 2
    int Ni; // number of points, including 0 buffer zone

    printf("Enter the data file name:\n");
    scanf("%s", &inFile);
    in = fopen(inFile, "r");

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

    printf("Loading data from file...\n");

```

```

for (numPoints = 0; fscanf(in, "%f %f", &t, &ampl) != EOF; ++numPoints);
printf("Number of Points: %i\n", numPoints);

for (exp = 1; pow(2, exp) < numPoints; ++exp);
Ni = pow(2, exp);
printf("Number of Points (with buffer region): %i\n", Ni);

float samplingFrequency = numPoints / t;
printf("Sampling Frequency: %f\n", samplingFrequency);
rewind(in);
float *time = (float *)calloc(sizeof(float), Ni);
float *data = (float *)calloc(sizeof(float), Ni);
for (int i = 0; i < numPoints; ++i) {
    fscanf(in, "%f\t%f", &time[i], &data[i]);
}
fclose(in);
printf("Load Successful\n\n");
free(time);
free(data);

fft(data - 1, Ni, 1);

float *ampSpec = (float *)calloc(sizeof(float), Ni / 2);
float *kValues = (float *)calloc(sizeof(float), Ni / 2);

for (int k = 0, i = 2; k < Ni / 2; ++k) {
    ampSpec[k] = sqrt(pow(data[i], 2) + pow(data[i + 1], 2))*2.0 /
(float)numPoints;
    kValues[k] = ((float)k*samplingFrequency) / (float)Ni;
    i += 2;
}

printf("Enter the output file name:\n");
scanf("%s", &outFile);
out = fopen(outFile, "w");
for (int i = 0; i < Ni / 2; ++i) {
    fprintf(out, "%f\t%f\n", kValues[i], ampSpec[i]);
}
fclose(out);
free(ampSpec);
free(kValues);

system("PAUSE");
return 0;
}

```

```

/*****
*
* ANSI C Example program:
*   ContAcqSamps-IntClk_main.c
*
* Example Category:
*   AI
*
* Description:

```

```

*   This example demonstrates how to continuously acquire data using
*   the device's internal timing (rate is governed by an internally
*   generated pulse train).
*
* Instructions for Running:
*   1. Select the physical channel to correspond to where your
*       signal is input on the DAQ device. Set to channel # 0 for this project.
*   2. Enter the minimum and maximum voltage ranges.
*   Note: For better accuracy try to match the input range to the
*         expected voltage level of the measured signal.
*   3. Set the sample rate of the acquisition.
*   Note: The rate should be at least twice as fast as the maximum
*         frequency component of the signal being acquired.
*
* Steps:
*   1. Create a task.
*   2. Create an analog input voltage channel.
*   3. Set the rate for the sample clock. Additionally, define the
*       sample mode to be continuous.
*   4. Call the Start function to start acquiring samples.
*   5. Read the waveform data in a loop until the user hits the stop
*       button or an error occurs.
*   6. Call the Clear Task function to clear the Task.
*   7. Display an error if any.
*
* I/O Connections Overview:
*   Make sure your signal input terminal matches the Physical
*   Channel I/O control.
*
* Recommended Use:
*   1. Call Configure and Start functions.
*   2. Call Read function in a timed loop.
*   3. Call Stop function at the end.
*
*****/
/* NOTE: this program collect up to 10 sec. If longer sampling time
needed than change the time in the ContAcqSamps-IntClk_Fn.c file from 10 to whatever
sampling time needed */
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <windows.h>
#include "NIDAQmx.h"
#include "ContAcqSamps-IntClk_Fn.h"

#define LOOPTIME    250

int main(int argc, char *argv[]) {
    printf("Start of Program: ContAcqSamps-IntClk_main\n");
    int32 error = 0;
    TaskHandle taskHandle = 0;
    clock_t startLoop = 0;
    clock_t startTime = 0;
    int32 read, totalRead = 0;
    float64 data[10000]; // output array

```

```

char      errBuff[2048] = { '\0' };
char      ch;
int buffer_size = 10000; //buffer size

/* Declare parameters that control the A/D converter
   * Sampling rate (samp_rate) in samples/sec or Hz
   * Range of signal (minimum and maximum values), Vmin and Vmax
   * run time (in seconds), run_time of program */
float samp_rate = 0; // sampling rate (Hz)
float Vmin = 0, Vmax = 0; // Range of signal, min and max values
float run_time = 0; // runtime of the program (seconds)

/* Declare the output file for storing sampled data */
char outputFile[100];
FILE *fp;

/* prompt user to enter values for the parameters that control the A/D converter
*/
printf("Enter the sampling rate (Hz)\n");
scanf("%f", &samp_rate);
printf("Enter the minimum signal value (Vmin)\n");
scanf("%f", &Vmin);
printf("Enter the maximum signal value (Vmax)\n");
scanf("%f", &Vmax);
printf("Enter the runtime of the program (seconds)\n");
scanf("%f", &run_time);

/*Calculate number of samples (num_samp = run_time* samp_rate) and check to make
sure that it does not
exceeds buffer size */
float num_samp = run_time * samp_rate;
if (num_samp > buffer_size) {
    return 1;
}

DAQmxErrChk(Configure_ContAcqSampsIntClk("Dev1/ai0", Vmin, Vmax, samp_rate,
&taskHandle, NULL));
// Note: ai0 sets ADC channel # to zero
DAQmxErrChk(Start_ContAcqSampsIntClk(taskHandle));
startTime = clock();
printf("Acquiring samples continuously. Press any key to interrupt\n");
while (!_kbhit()) {
    while (startLoop && clock() < startLoop + LOOPTIME)
        Sleep(10);
    startLoop = clock();
    DAQmxErrChk(Read_ContAcqSampsIntClk(taskHandle, num_samp, data,
buffer_size, &read));
    if (read > 0) {
        totalRead += read;
        printf("Acquired %d samples. Total %d\r", read, totalRead);
    }
}
printf("\nAcquired %d total samples.\n", totalRead);
ch = _getch();

/* prompt user to enter the output file name to save the sampled amplitude values
stored in the
array "data" along with the corresponding time points */

```

```

printf("Enter output file name\n");
scanf("%s", &outputFile);
fp = fopen(outputFile, "w");

for (int i = 0; i < num_samp; ++i) {
    fprintf(fp, "%f\t%f\n", i / samp_rate, data[i]);
}

fclose(fp);
return 0;

```

Error:

```

if (DAQmxFailed(error))
    DAQmxGetExtendedErrorInfo(errBuff, 2048);
if (taskHandle != 0)
    Stop_ContAcqSampsIntClk(taskHandle);
if (DAQmxFailed(error))
    printf("DAQmx Error: %s\n", errBuff);
printf("End of program, press any key to quit\n");
while (!_kbhit()) {}
ch = _getch();
}

```

/* Plotting.cpp : Defines the entry point for the console application.

Peter Dobbs
BIEN 3200 - Section 401
8 November 2016
Lab 5

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 ('Plot of EEG_05');"
        "xlabel('Time (sec)');"
        "ylabel('Signal Magnitude');");

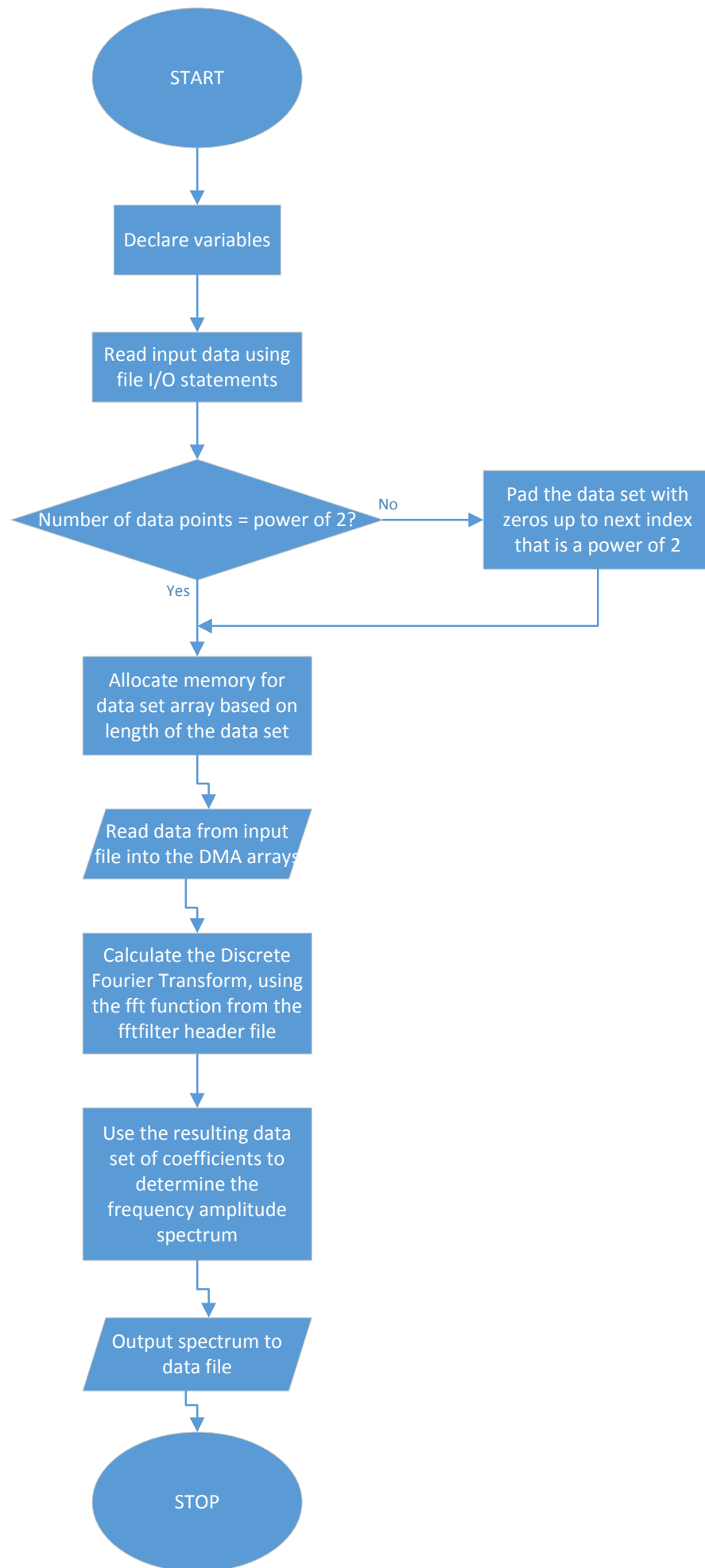
    system("pause"); //Pauses figure
    engClose(ep);
    //=====C=====

    free(aData);
    free(tData);

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

```

Discrete Fourier Transform



Plotting Project

