BIEN 3200: Lab 4

Laboratory Exercise #4

Peter Dobbs
Lab Section 401
Nicholas Heugel

18 October 2016

Abstra

In the field of biomedical engineering, it is becoming more and more important to understand signal processing and event detection. In this paper, a previously implemented algorithm (FS1) for detecting QRS complexes is recreated and analyzed. An algorithm for detecting arterial systolic and diastolic pressures (BP) is generated and implemented. A third and final program (CC) is developed to determine the correlation between two input sets of data. The FS1 program is proven to have an inferior detection algorithm to that of others that exist. The BP program is proven to hold a perfect detection algorithm for the input data set. Further testing against other sets of data will be necessary for determining the extent of the algorithms ability. The CC program is used for showing the comparison between a control set of rats and rats treated with halothane. Through this exercise, the author practices programming, creation of detection algorithms, and analytical technical writing.

Introduction:

Various methods of event detection exist for the various types of events and the range of pathologies. This exercise was performed to introduce students to techniques for event and feature detection in physiologic signals, and expose them to real-world applications. Through the writing of this lab report, technical writing skills are to be reinforced.

Equations:

$$first\ derivative = d1\ n = ABS\ X\ n+1 - X\ n-1\ ,2 < n < N$$

$$second\ derivative = d2\ n = ABS\ X\ n+2 - 2X\ n + X(n-2)\ ,2 < n < N$$

$$FS1\ equation = y\ n = 1.3d1\ n + 1.1d2\ n\ ,2 < n < N$$

$$correlation\ coefficient = r_{xy} = \frac{\sum_{n=1}^{N}(x\ n - \bar{x})(y\ n - \bar{y})}{\sum_{n=1}^{N}(x\ n - \bar{x})^2 \sum_{n=1}^{N}(y\ n - \bar{y})^{2\ 1/2}}$$

$$sensitivity = Se = 100 \times \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$positive\ predictivity = PP = 100 \times \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Data Files:

Sinus*i*.txt	*i* = 1, 2, 3, 4, 5	(1)
BPirreg.txt		(2)
ERP_C_*i*.txt	*i* = 0, 5, 10, 15, 20	(3)
ERP_H_*i*.txt	*i* = 0, 5, 10, 15, 20	(4)

Methods:

The first algorithm was implemented to detect QRS complexes. Identified as FS1, this algorithm uses the scaled and summed first and second derivative of a filtered signal. A set of ECG data (Data File 1) is taken as input. Using dynamic memory allocation, the data set is stored in a pair of arrays, one for time and the other for amplitude. The signal amplitude is filtered through a three point moving average filter. Then the scaled first and second derivatives are summed. This resulting equation is used for the detection by comparing it to the expected threshold for detecting a QRS complex. All of these manipulations are made possible by the use of looping structures, such as the *for-loop*. The sensitivity and positive predictability are calculated based on the results of the event detection algorithm. Both are applied to the results of the input *sinus2.txt* data file.

The second algorithm was implemented to detect arterial systolic and diastolic pressures. Using the first derivative (D_{BP}) of the signal, the locations of the relevant pressures were determined. A file containing irregular blood pressure data (Data File 2) was used as input. Local maxima of the signal were determined by looping through the D_{BP} and checking if the point to the left is greater than zero and if the point to the right is less than zero. If the amplitude of the signal at the detected local maximum is greater than a set threshold (in this case 75 mmHg), then

a systolic pressure has been detected. Otherwise, if the amplitude is greater than a set threshold (in this case 60 mmHg), a diastolic pressure has been detected. If there exists a detected amplitude that does not cross either threshold, then it is not stored. The sensitivity and positive predictability are calculated based on the results of the event detection algorithm. Both metrics are applied to the results of the *BPirreg.txt* data file.

The third algorithm was implemented to determine the correlation coefficient of a pair of signals. Electroencephalogram data related to an animal study performed to test the effects of halothane on brain function was used in this program. A pair of EEG data sets (some combination of Data Files 3 and Data Files 4) are selected by the user as inputs for the program. The program is then used to calculate and plot the correlation between the two input signals. The correlation between the two signals is calculated by using the correlation coefficient equation.

Results:

FS1 Algorithm Program

```
Enter input file name
sinus2.txt
number of data points in file = 2500
Press any key to continue . . .
Press any key to continue . . .
Average Heart Rate (beats/minute) = 66.026405
Enter ouput file name
out2.txt
Press any key to continue . . .
```

Figure 1: Screenshot of the output for the FS1 algorithm program.

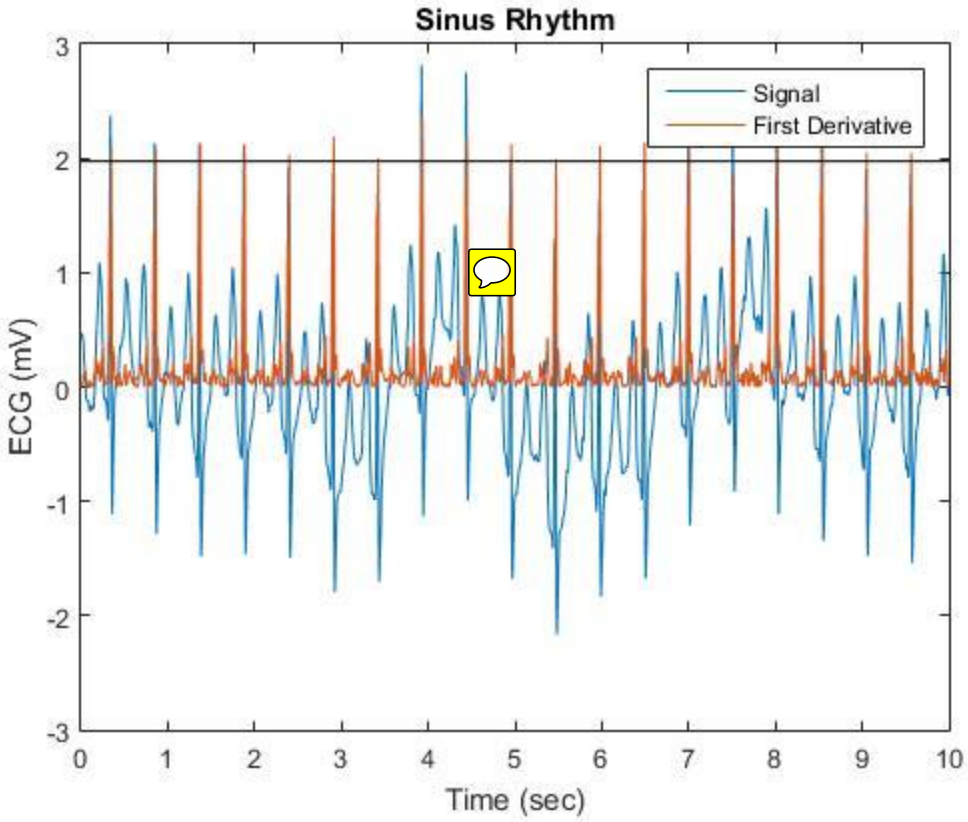
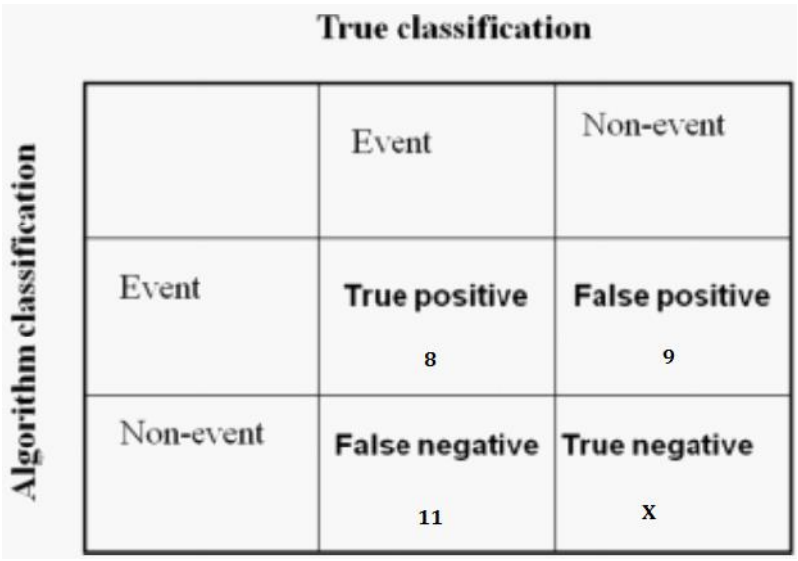


Figure 2: Plot of the signal for sinus2.txt, its first derivative and the threshold line



$$Se = \frac{8}{8+11} \times 100 = 42.10\%$$

$$PP = \frac{8}{8+9} \times 100 = 47.06\%$$

BP Detection Algorithm Program

```
Blood Pressure Event Detection
number of data points in file = 10001
Data written to file: 'out.txt'
Press any key to continue . . .
Plotting In MatLab:
Press any key to continue . . .
```

Figure 3: Screenshot of the output for the BP algorithm program.

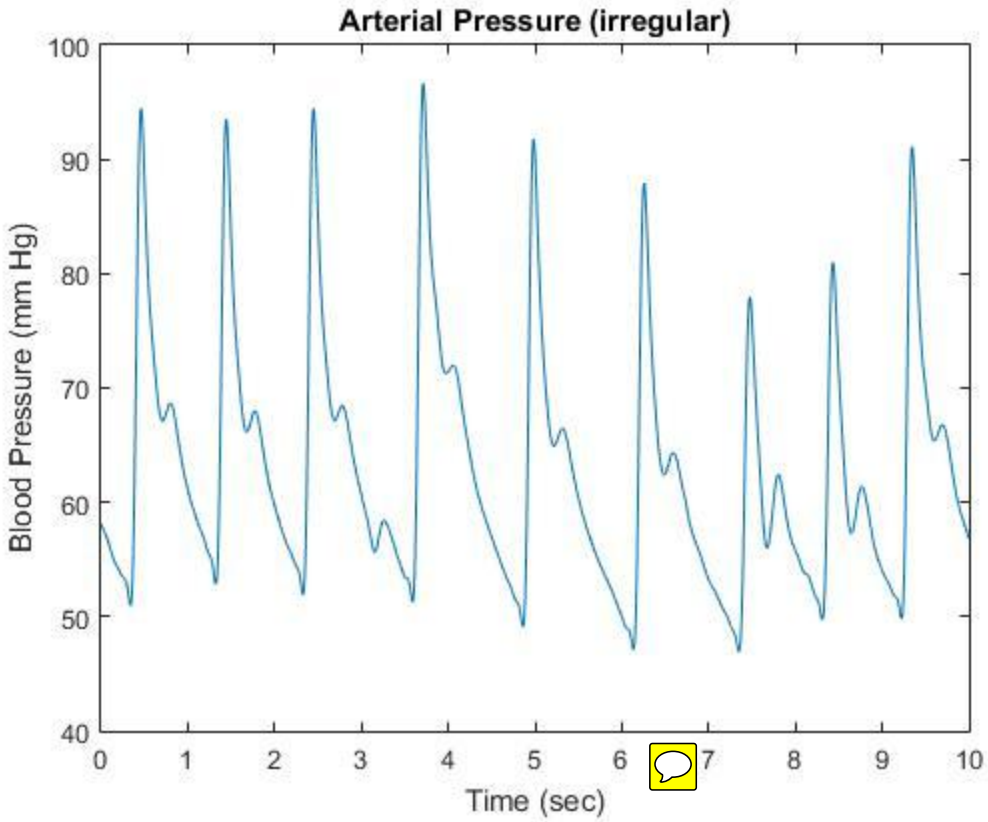


Figure 4: Plot of the input BPirreg.txt data set

True classification

Algorithm classification	Event	Non-event
Event	**True positive** 18	**False positive** 0
Non-event	**False negative** 0	**True negative** 1

$$Se = \frac{10}{10 + 0} \times 100 = 100\%$$

$$PP = \frac{10}{10 + 0} \times 100 = 100\%$$

Correlation Coefficient Program

```
Enter 1st input file name
ERP_C_0.txt
Enter 2nd input file name
ERP_C_5.txt
number of data points in file = 500
number of data points in file = 500
Correlation Coefficient: 0.750511
Press any key to continue . . .
```

Figure 5: Screenshot of the output for the Correlation Coefficient program.

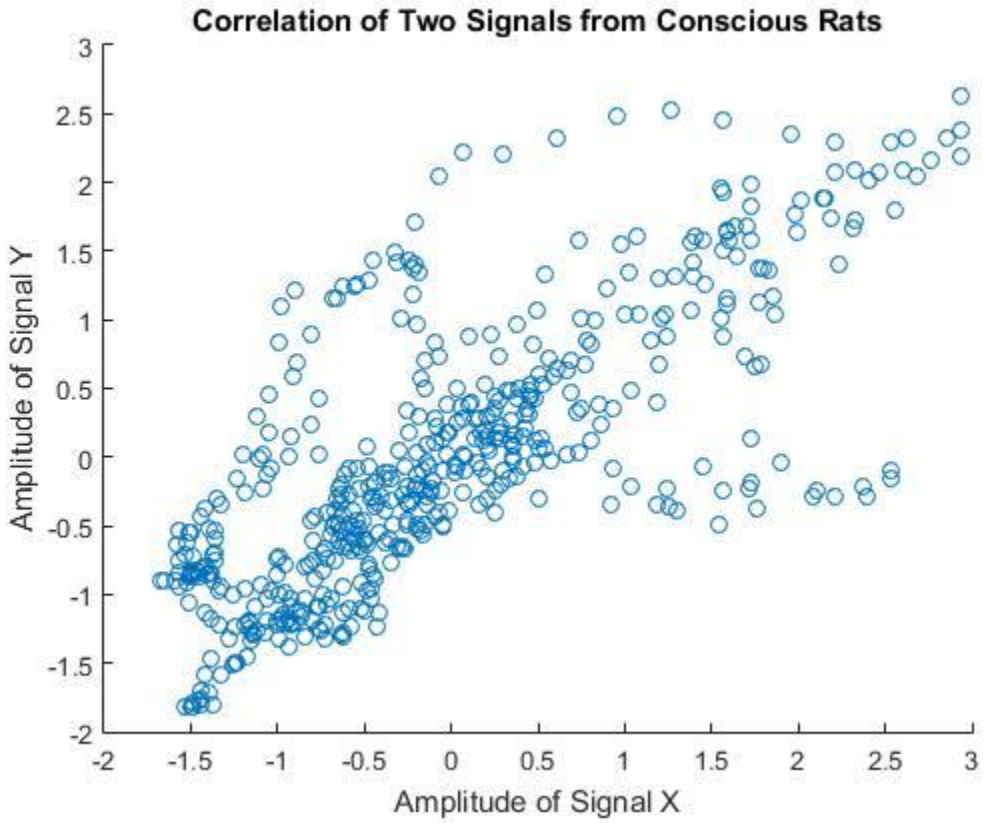


Figure 6: Plot of correlation between the data from ERP_C_0.txt and ERP_C_5.txt

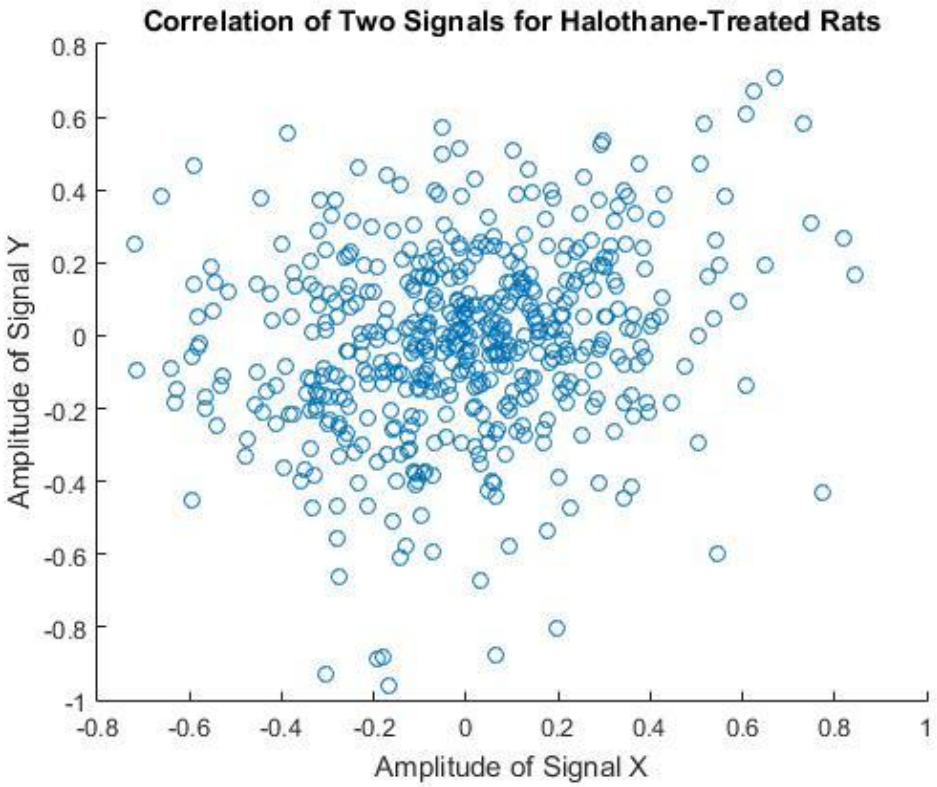


Figure 7: Plot of the correlation between data sets from ERP_H_0.txt and ERP_H_5.txt

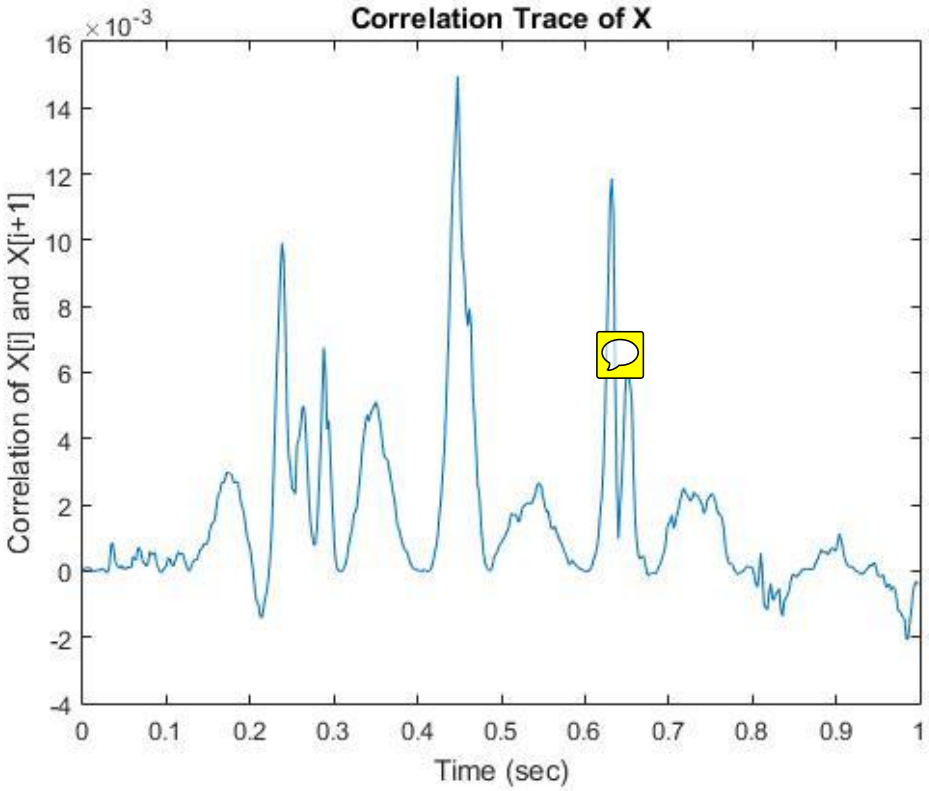


Figure 8: Plot of the correlation trace for conscious rat data

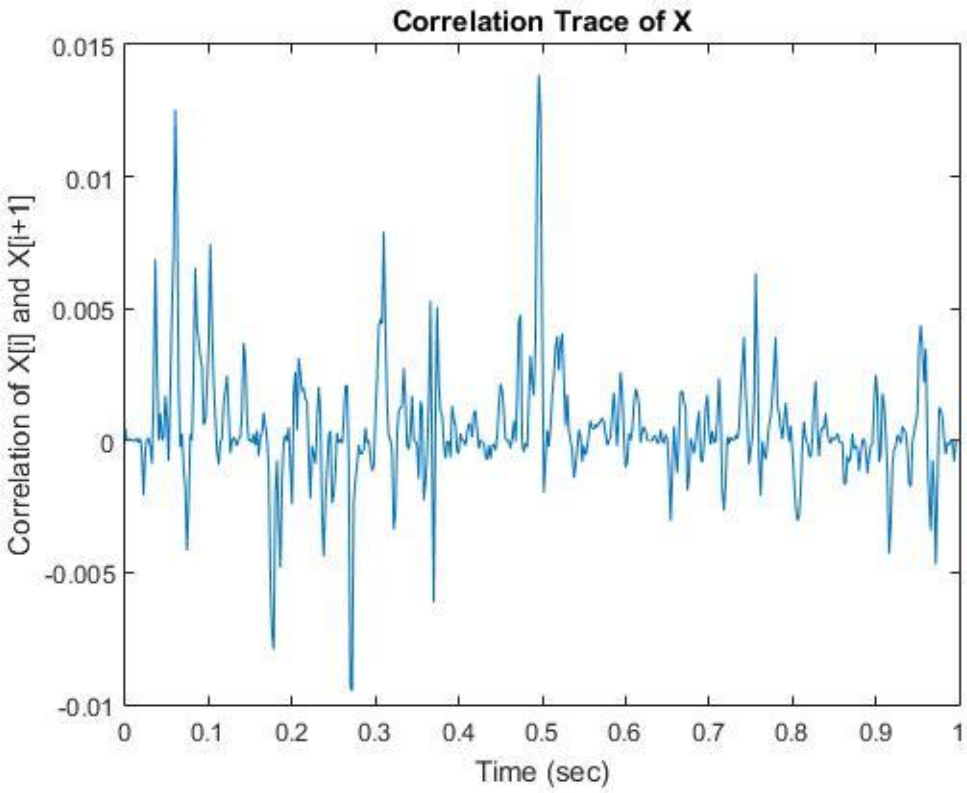


Figure 9: Plot of the correlation trace for halothane-treated rats

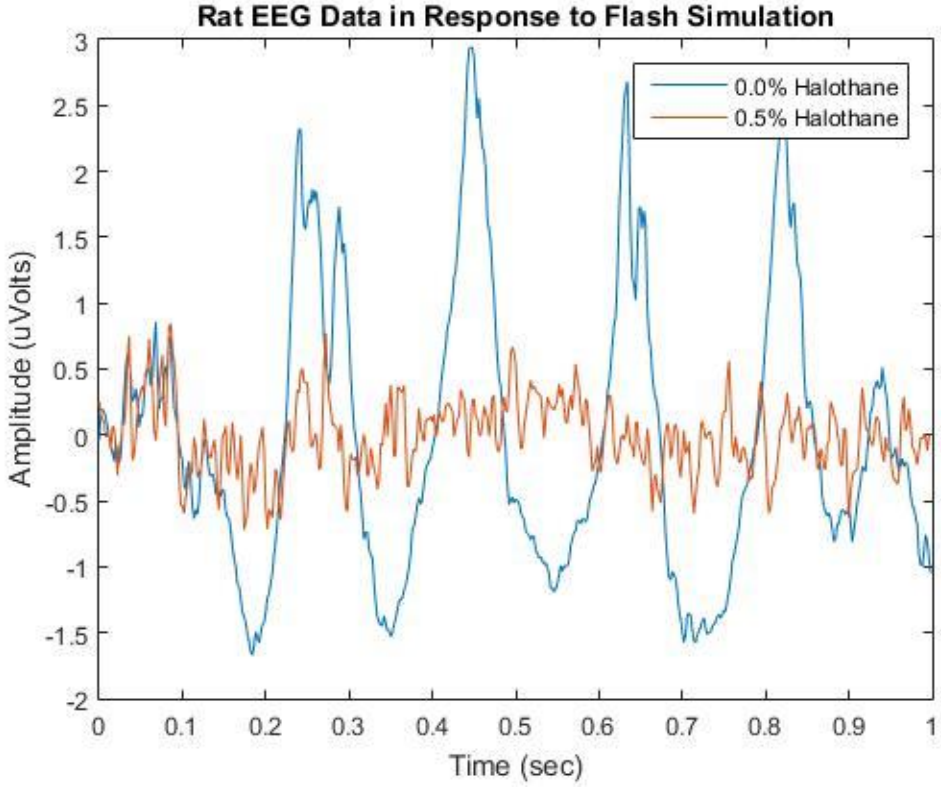


Figure 10: Plot comparing the data set ERP_H_0.txt and ERP_C_0.txt

Discussion:

The expectations of improved technical writing and understanding of techniques for event and feature detection in physiologic signals were met. Each of the algorithms were implemented properly, meeting the predicted results. The FS1 algorithm worked about as well as was expected, since it performed poorly in the Friesen study. Its detection method is overly complicated for what it could be. As a general rule, the more complicated the algorithm the more likely it is to fail when stressed. Unfortunately the algorithm does not really do a good job of detecting QRS complexes, even in simple cases of normal ECG data.

The BP algorithm detected all of the correct events and classified them correctly. It is the perfect algorithm for the given data set. That description, however, is not really a positive one. The algorithm is likely not very flexible, due to the set thresholds of 75 for systolic and 60 for diastolic. These thresholds would fail in a data set that contained normal blood pressure data. One solution to this problem would be to implement the thresholds as percentages of the global maximum of the input data set. That implementation would be especially simple since we have already had to do that in previous lab exercises.

The CC program performs as expected. The results of the correlation analysis shows that the conscious rat data is a good control. The correlation between the ERP_C_i.txt data sets is all relatively close to one, which is good. When looking at the halothane-treated rats, one can observe the real effects of the halothane on the brain function of the rats. The correlation between consecutive signals is very slow, indicating a dulling effect on the brain activity.

During programming, multiple issues arose. Problems with memory allocation and the heap came to wreak havoc on the code structure. The proper indexing of arrays proved to be more difficult than initially expected. Being used to modularization as a coding standard, this

kind of program showed a challenge for anyone used to software development with typical higher level programming standards.

Conclusion:

The FS1 algorithm was found to be a poor detection algorithm for the ECG data sets. The created BP algorithm was determined to meet all expectations. The correlation coefficient program performs as intended.

Questions:

The P wave, QRS complex, and T wave of the ECG signal result from a series of cardiac cycle events. The spread of depolarization through the atria causes the P wave. Atrial contraction follows. The electrical depolarization of the ventricles results in the QRS complex, about 0.16 seconds after the onset of the P wave. The Q represents the signal dividing into left and right branches. From Q to S, the signal spreads across the cells of the ventricle walls. The R wave marks left ventricle contraction, while the S wave marks right ventricle contraction. Following the onset of ventricular systole, one may observe the ventricular T wave. This wave represents the repolarization of the ventricles, when the ventricular muscle fibers begin to relax.

Typical examples of noise artifacts include power line interference; electrode contact noise; motion artifacts; muscle contraction (electromyography, EMG); baseline drift and ECG amplitude modulation with respiration; instrumentation noise generated by electronic devices used in signal processing; and electrosurgical noise. Power line interference comes from the pickup of 60 Hz in the U.S. (varies by country) and harmonics. Electrode contact noise arises from poor electrode contact with the patient, including complete loss of contact or intermittent loss from movement or vibration. Movement of the electrodes, resulting in changes in impedance leads to motion artifacts. In electromyography, the recording of the electrical activity of muscle

tissue, muscle contractions cause artefactual potentials to be generated, thus producing noise. Respiration causes about 15% variation in the ECG signal amplitude and significant drift in the baseline. The very electronic devices used to process these signals produce noise, which must be corrected in order to take useful ECG readings. During electrosurgical procedures, the noise created by the current from the blade destroys the ECG, resulting in insufficient readings for heart activity.

C Code

```
/* FS1EventDetection.cpp : Defines the entry point for the console application.
              Peter Dobbs
              BIEN 3200 - Section 401
              18 October 2016
              Lab 4

              Program Description:
                     This program implements an event detection algorithm
                     that uses the first and second derivatives of a signal
                     to determine the location of a threshold crossing in
                     order to find the event of a QRS complex.
*/

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

int numPoints;

float getEcgMax(float *data) {
       float max = 0;
       for (int i = 0; i < numPoints; ++i) { //find maxECG from input data
              if (data[i] > max) max = data[i];
       }
       return max;
}

int main(void) {
       float time, amplitude;
       float samplingFrequency;
       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);
       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]);
		aData[i] = aData[i] / samplingFrequency; // modification
	}
	fclose(inputFile);

	float *filteredSignal = (float *)calloc(sizeof(float), numPoints);
	for (int i = 2; i < numPoints; ++i) {//3-point moving average filter (Hanning
filter)
		filteredSignal[i - 2] = (aData[i] + 2 * aData[i - 1] + aData[i - 2]) / 4;
	}

	float *y0 = (float *)calloc(sizeof(float), numPoints);
	float *y1 = (float *)calloc(sizeof(float), numPoints);
	float *y2 = (float *)calloc(sizeof(float), numPoints);
	for (int i = 1; i < numPoints - 3; ++i)
		y0[i] = fabsf(filteredSignal[i + 1] - filteredSignal[i - 1]);
	for (int i = 2; i < numPoints - 4; ++i)
		y1[i] = fabsf(filteredSignal[i + 2] - 2 * filteredSignal[i] -
filteredSignal[i - 2]);
	for (int i = 0; i < numPoints - 2; ++i)
		y2[i] = 1.3*y0[i] + 1.1*y1[i];

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

	mxArray* matX1 = mxCreateNumericMatrix(numPoints, 1, mxSINGLE_CLASS, mxREAL);
	mxArray* maty1 = mxCreateNumericMatrix(numPoints, 1, mxSINGLE_CLASS, mxREAL);
	mxArray* maty2 = mxCreateNumericMatrix(numPoints, 1, mxSINGLE_CLASS, mxREAL);

	memcpy((void*)mxGetPr(matX1), (void*)tData, sizeof(float) * numPoints);
	memcpy((void*)mxGetPr(maty1), (void*)aData, sizeof(float) * numPoints);
	memcpy((void*)mxGetPr(maty2), (void*)y0, sizeof(float) * numPoints);

	engPutVariable(ep, "t", matX1);
	engPutVariable(ep, "ampl", maty1);
	engPutVariable(ep, "d", maty2);

	engEvalString(ep, "figure;"
		"plot(t,ampl,t,d);"
		"title ('Sinus Rhythm');"
		"xlabel('Time (sec)');"
		"ylabel('ECG (mV)');");

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

	int blank = (0.1) * (samplingFrequency); //blanking period for the dataset
	float threshold = 0.7 * getEcgMax(filteredSignal);
	int numCrossings = 0;

	for (int i = 0; i < numPoints; ++i) {
		if (filteredSignal[i] > threshold) {
```

```
                ++numCrossings;
                i += blank;
        }
    }

    float *locations = (float *)calloc(sizeof(float), numPoints); //allocate memory
    float *amplitudes = (float *)calloc(sizeof(float), numPoints);
    int atLeastSix;
    for (int i = 0, j = 0; i < numPoints - 2; ++i) {
        if (y2[i] >= 1.0) {
                atLeastSix = 0;
                for (int n = i + 1; n <= 9 + i; ++n)
                        if (y2[n] >= 1.0)++atLeastSix;
                if (atLeastSix >= 6) {
                        locations[j] = tData[i + 2];
                        amplitudes[j] = aData[i + 2];
                        ++j;
                        i += blank;
                }
        }
    }

    //Calculate average heart rate
    float avgHR = numCrossings / (time / 60);
    printf("Average Heart Rate (beats/minute) = %f\n", avgHR);

    // output results to file
    char out_file[100];
    FILE *outFile;
    printf("Enter ouput file name\n");
    scanf("%s", out_file);
    outFile = fopen(out_file, "w");
    for (int i = 0; i < numCrossings; ++i) {
        fprintf(outFile, "%f\t%f\n", locations[i], amplitudes[i]);
    }
    fprintf(outFile, "Heart Rate: %f", avgHR);
    fclose(outFile);

    free(aData);
    free(filteredSignal);
    free(y0);
    free(y1);
    free(locations);
    free(amplitudes);
    free(tData);
    free(y2);

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

```
/* BPEventDetection.cpp : Defines the entry point for the console application.
		Peter Dobbs
		BIEN 3200 - Section 401
		18 October 2016
		Lab 4

		Program Description:
			This program implements a basic event detection algorithm to
determine
			the event of systolic pressure and diastolic pressure
*/

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

int main(void) {
	printf("Blood Pressure Event Detection\n");
	int n = 0;
	float *tData, *aData, t, y;
	FILE *inputFile = fopen("BPirreg.txt", "r");

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

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

	// move cursor to beginning of indicated file
	rewind(inputFile);

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

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

	float *d1 = (float *)calloc(sizeof(float), n);
	for (int i = 1; i < n - 1; ++i) // first derivative
		d1[i] = aData[i + 1] - aData[i - 1];

	//Event Detection Algorithm
	float *locationSys, *locationDia, *systolic, *diastolic;

	locationSys = (float *)calloc(sizeof(float), n);
	locationDia = (float *)calloc(sizeof(float), n);
	systolic = (float *)calloc(sizeof(float), n);
	diastolic = (float *)calloc(sizeof(float), n);
	int countSys = 0, countDia = 0;
```

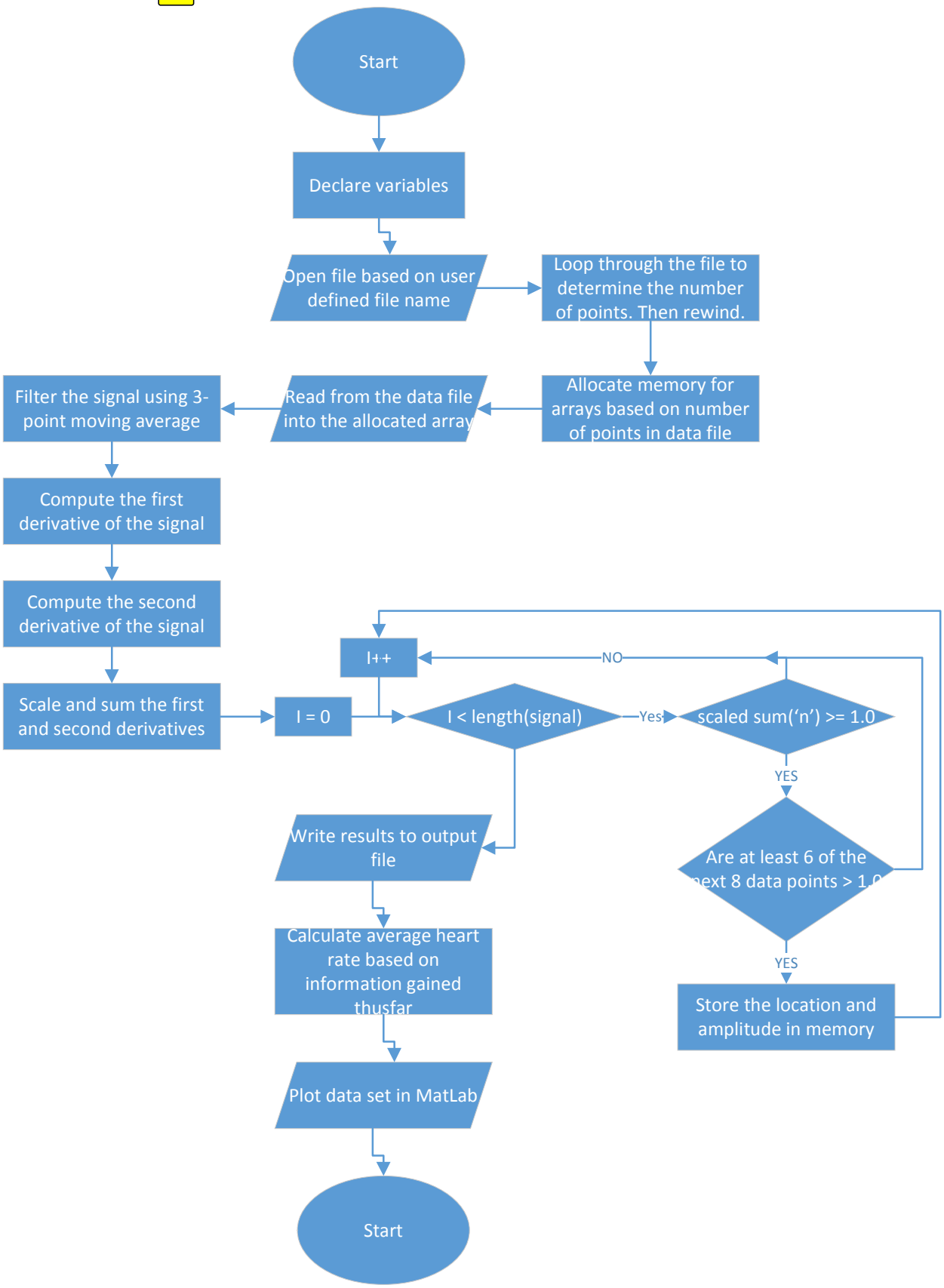
```
for (int i = 1; i < n - 2; ++i) {
	if (d1[i - 1] > 0 && d1[i + 1] < 0) { // the point is a local maximum
		if (aData[i - 1] >= 75) { // it is the systolic pressure
			systolic[countSys] = aData[i - 1];
			locationSys[countSys] = tData[i - 1];
			++countSys;
		} else if (aData[i - 1] >= 60) { // it is the diastolic pressure
			diastolic[countDia] = aData[i - 1];
			locationDia[countDia] = tData[i - 1];
			++countDia;
		}
		i += ((n / t)*0.01); // blanking period
	}
}

// ouput algorithm results to file
FILE *outFile;
outFile = fopen("out.txt", "w");
for (int i = 0; i < countSys; ++i) {
	fprintf(outFile, "%f\t%f\n", locationSys[i], systolic[i]);
	fprintf(outFile, "%f\t%f\n", locationDia[i], diastolic[i]);
}
fclose(outFile);
printf("Data written to file: 'out.txt'\n");


system("PAUSE");
printf("Plotting In MatLab:\n");
////////====MATLAB====//////////////////
Engine *ep;
ep = engOpen(NULL);

mxArray* matx1 = mxCreateNumericMatrix(n, 1, mxSINGLE_CLASS, mxREAL);
mxArray* maty1 = mxCreateNumericMatrix(n, 1, mxSINGLE_CLASS, mxREAL);
mxArray* matD1 = mxCreateNumericMatrix(n, 1, mxSINGLE_CLASS, mxREAL);

memcpy((void*)mxGetPr(matx1), (void*)tData, sizeof(float) * n);
memcpy((void*)mxGetPr(maty1), (void*)aData, sizeof(float) * n);
memcpy((void*)mxGetPr(matD1), (void*)d1, sizeof(float) * n);

engPutVariable(ep, "time", matx1);
engPutVariable(ep, "ampl", maty1);
engPutVariable(ep, "d1", matD1);

engEvalString(ep, "figure; plot(time,ampl);"
	"title ('Arterial Pressure (irregular)');"
	"xlabel('Time (sec)');"
	"ylabel('Blood Pressure (mm Hg)');"
	);

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


free(locationSys);
free(locationDia);
```

```
	free(systolic);
	free(diastolic);
	free(tData);
	free(aData);
	free(d1);

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

/* CorrelationCoefficient.cpp : Defines the entry point for the console application.
		Peter Dobbs
		BIEN 3200 - Section 401
		18 October 2016
		Lab 4

		Program Description:
			This program determines the correlation between two input datasets
*/

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

float getArithMean(float *data, int length) {
	float sum = 0;
	for (int i = 0; i < length; ++i)
		sum += data[i];
	return (sum / length);
}

float getVariance(float *data, int length) {
	float mean = getArithMean(data, length);
	float var = 0;
	for (int i = 0; i < length; ++i) {
		var += pow(data[i] - mean, 2);
	}
	return var;
}

float getCorrCoeff(float *X, float *Y, int n) {
	float varX = getVariance(X, n);
	float varY = getVariance(Y, n);
	float meanX = getArithMean(X, n);
	float meanY = getArithMean(Y, n);
	float numer = 0;
	for (int i = 0; i < n; ++i) {
		numer += ((X[i] - meanX)*(Y[i] - meanY));
	}
	return (numer / sqrt(varX*varY));
}

int main(void) {
	int numPoints1 = 0, numPoints2 = 0;
```

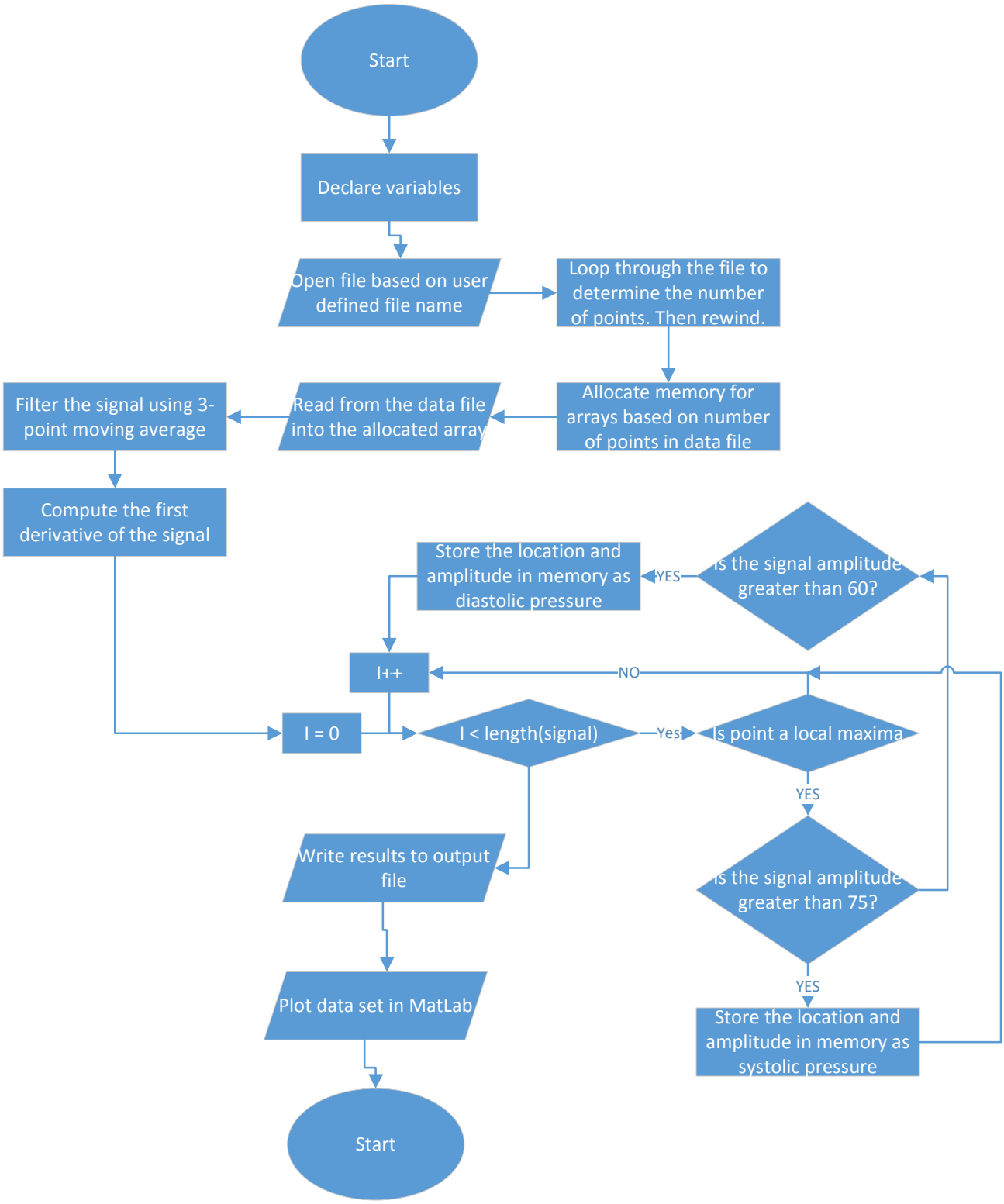
```
float *t1, *t2, *X, *Y, t, a;
char in_file[100];
FILE *inputFile1, *inputFile2;

printf("Enter 1st input file name\n");    // open first file to be read
scanf("%s", in_file);
inputFile1 = fopen(in_file, "r");
printf("Enter 2nd input file name\n");    // open 2nd file to be read
scanf("%s", in_file);
inputFile2 = fopen(in_file, "r");

if (!(inputFile1) || !(inputFile2)) {
	printf("One of the files does not exist.\n");
	exit(EXIT_FAILURE);
}

// determine the number of points in the opened file
for (numPoints1 = 0; fscanf(inputFile1, "%f %f", &t, &a) != EOF; ++numPoints1);
printf("number of data points in file = %i\n", numPoints1);
int samplingFrequency1 = numPoints1 / t;
for (numPoints2 = 0; fscanf(inputFile2, "%f %f", &t, &a) != EOF; ++numPoints2);
printf("number of data points in file = %i\n", numPoints2);
int samplingFrequency2 = numPoints2 / t;

rewind(inputFile1); // move cursor to beginning of indicated file
rewind(inputFile2);

t1 = (float *)calloc(sizeof(float), numPoints1); //allocate memory
X = (float *)calloc(sizeof(float), numPoints1);
t2 = (float *)calloc(sizeof(float), numPoints2);
Y = (float *)calloc(sizeof(float), numPoints2);

for (int i = 0; i < numPoints1; ++i)
	fscanf(inputFile1, "%f\t%f", &t1[i], &X[i]);
fclose(inputFile1);
for (int i = 0; i < numPoints2; ++i)
	fscanf(inputFile2, "%f\t%f", &t2[i], &Y[i]);
fclose(inputFile2);

// Correlation Coefficient
float r_xy = getCorrCoeff(X, Y, numPoints1);
printf("Correlation Coefficient: %f\n", r_xy);

//char out_file[100];
//FILE *outFile;

//printf("Enter ouput file name\n"); //write to file
//scanf("%s", out_file);
//outFile = fopen(out_file, "w");

// Correlation Traces
float *trace = (float *)calloc(sizeof(float), numPoints1);
float varX = getVariance(X, numPoints1);
float varY = getVariance(Y, numPoints1);
float meanX = getArithMean(X, numPoints1);
float meanY = getArithMean(Y, numPoints1);
/*for (int i = 0; i < numPoints1; ++i) {
	trace[i] = ((X[i] - meanX)*(Y[i] - meanY)) / sqrt(varX*varY);
```

```
	}*/
	for (int i = 0; i < numPoints1 - 1; ++i) {
		trace[i] = ((X[i] - meanX)*(X[i + 1] - meanX)) / varX;
	}

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

	mxArray* matT1 = mxCreateNumericMatrix(numPoints1, 1, mxSINGLE_CLASS, mxREAL);
	mxArray* matx1 = mxCreateNumericMatrix(numPoints1, 1, mxSINGLE_CLASS, mxREAL);
	mxArray* maty1 = mxCreateNumericMatrix(numPoints2, 1, mxSINGLE_CLASS, mxREAL);
	mxArray* matTrace = mxCreateNumericMatrix(numPoints1, 1, mxSINGLE_CLASS, mxREAL);

	memcpy((void*)mxGetPr(matT1), (void*)t1, sizeof(float) * numPoints1);
	memcpy((void*)mxGetPr(matx1), (void*)X, sizeof(float) * numPoints1);
	memcpy((void*)mxGetPr(maty1), (void*)Y, sizeof(float) * numPoints2);
	memcpy((void*)mxGetPr(matTrace), (void*)trace, sizeof(float) * numPoints1);

	engPutVariable(ep, "t", matT1);
	engPutVariable(ep, "X", matx1);
	engPutVariable(ep, "Y", maty1);
	engPutVariable(ep, "trace", matTrace);

	engEvalString(ep, "figure; scatter(X,Y);"
		"title('Correlation of Two Signals');"
		"xlabel('Amplitude of Signal X');"
		"ylabel('Amplitude of Signal Y');"
		"figure; plot(t,trace);"
		"title('Correlation Trace of X');"
		"xlabel('Time (sec)');"
		"ylabel('Correlation of X[i] and X[i+1]');");

	system("pause");

	engClose(ep); // close the engine
	////////////======C======/////////////////

	free(t1);
	free(X);
	free(t2);
	free(Y);
	free(trace);

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

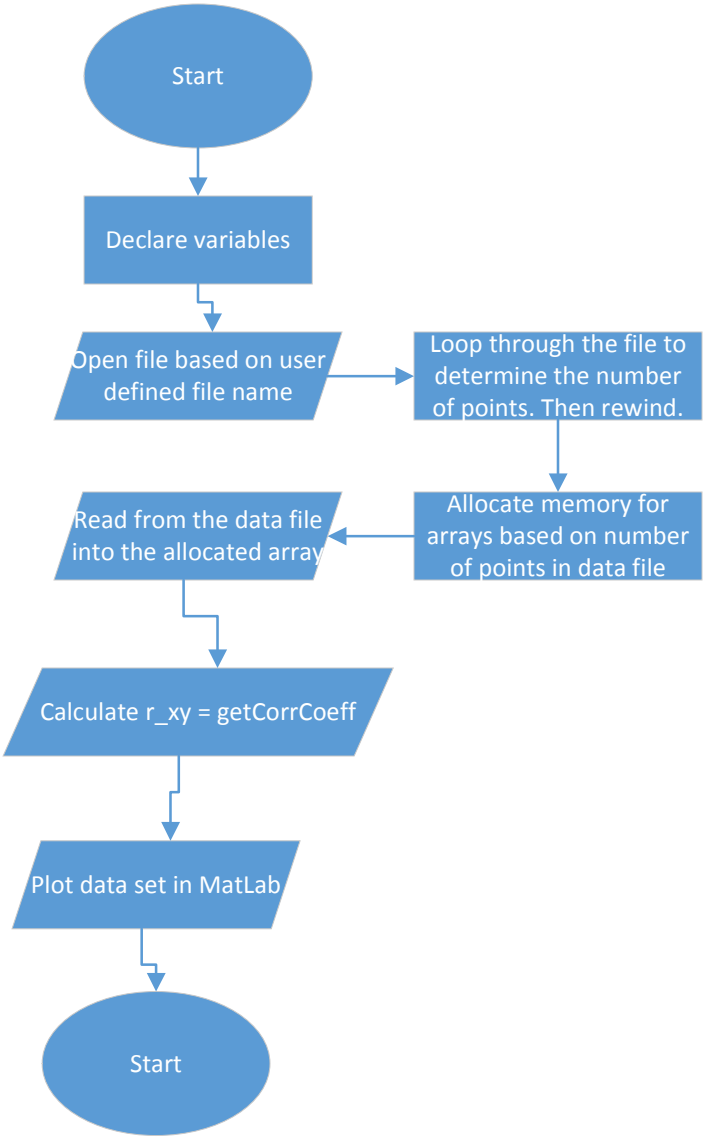
FS1 Algorithm



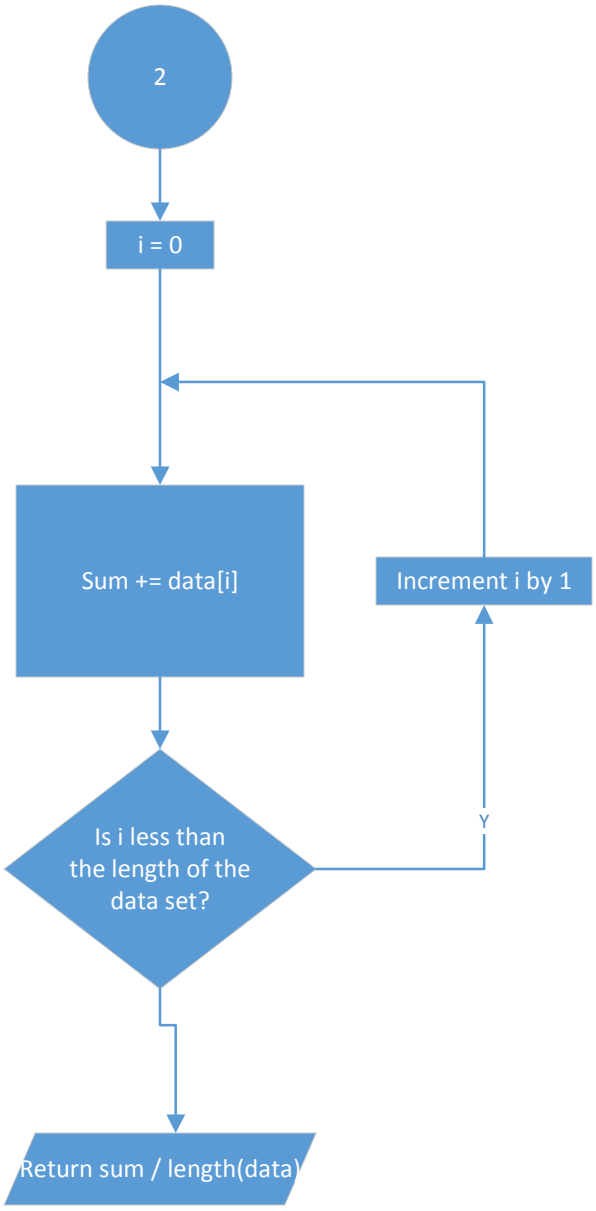
BP Algorithm



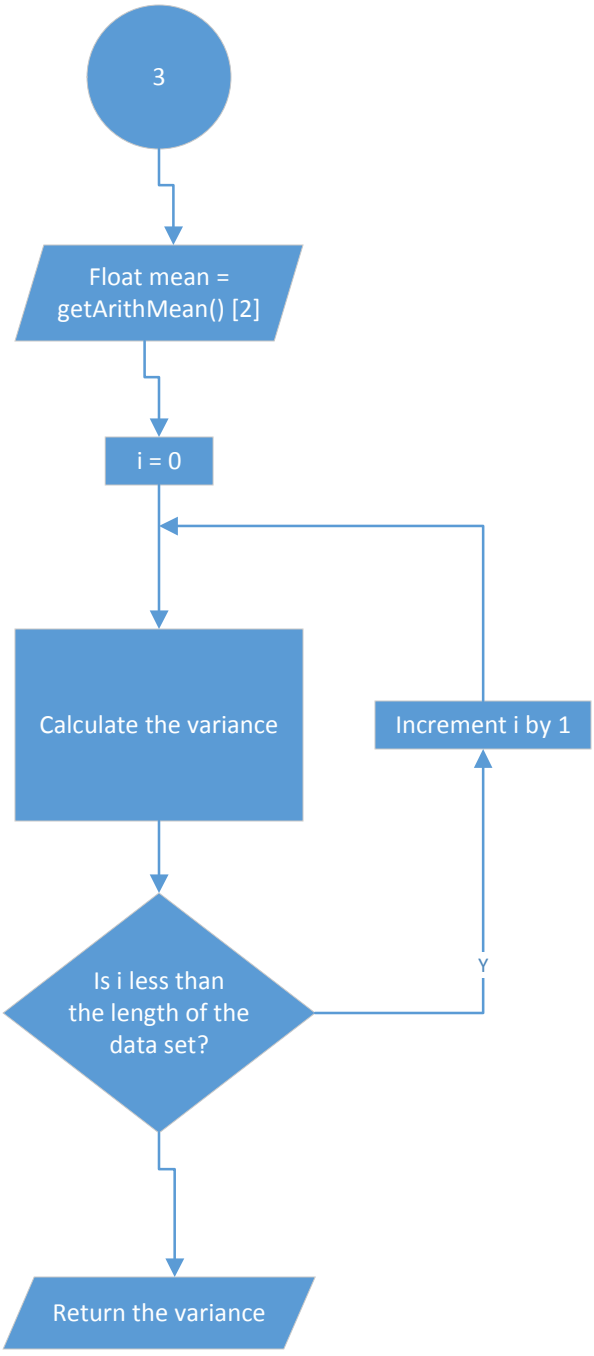
Correlation Coefficient



Correlation Coefficient



Correlation Coefficient



Correlation Coefficient