1 Introduction

In this programming project, we are going to implement three algorithms we learned from lecture, including 1-D filtering, the Haar transform and autocorrelation. All these algorithms are basic but useful tools for signal processing. By implementing these algorithms on real medical data, you can get better insight into the material.

Again, this assignment can be done individually or in pairs, though we strongly encourage you to work in pairs. You can use any languages to do it. But this time, you are supposed to implement your own algorithms so that you CANNOT use any built-in functions or third-party code of these algorithms directly. If you have any questions about this requirement, please ask the TA (chenwang@cs.cornell.edu).

2 Dataset

2.1 Overview on the Dataset

The data consist of electrocorticographic (ECoG) recordings, courtesy of Keith Purpura, (Associate Prof. of Neuroscience, Brain and Mind Research Institute) and Theodore Schwartz (Associate Prof. of Neurosurgery).

The data are recorded while the subject carries out two tasks:

- memsac: a task in which the subject needs to make an eye movement to a remembered location
- maze: a task in which the subject needs to find a path through a maze, and to trace the track with his eyes

There are two sets of recordings for each task; each set of recordings consists of 10 channels of data. In other words, we have 40 continuous signals in this dataset.

But we don't to need to care about the physical meaning of the data in this assignment. What we need to keep in mind is that they are "continuous" data. We put quotes on "continuous" here because we cannot get real continuous data in a computer system. The sampling rate is 500Hz so that the data points are so dense that we can view the data as "continuous".

2.2 Data File Format

2.2.1 Matlab File

The data are contained in the matlab structure ecog_data in 'CS5540_ecog.mat'.

In general, ecog_data is a cell array with 40 cells. There is a 1-D array stored in each cell. You can refer to the i-th signal by $ecog\_data\{i\}$.

2.2.2 Textual File

The data are stored in the textual file 'CS5540_ecog.txt'.

Since we have 40 different signals in the dataset, there are 40 lines in the txt file. Each line starts with a number n, the length, followed by n numbers representing the signal itself.

3 Assignment

3.1 Zero-crossings

Let's start with a warm-up task, write a function to find the zero-crossings on a signal for some given value of "zero".

More precisely, given the signal f and a threshold v, you are supposed to find all the time stamps t such that $(f(t) \leq v) \wedge (f(t+1) \geq v)$ or $(f(t) \geq v) \wedge (f(t+1) \leq v)$.

3.2 Convolution and Filters

3.2.1 Convolution

First, we need to implement our own convolution function since all the filtering algorithm consists of is the convolution of the original signal and some stencil. The idea of discrete convolution is just local averaging which was illustrated in lecture. Given two discrete functions f and g, their convolution is defined as (indexed from 0):

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m] \tag{1}$$

In signal processing, we can use g to represent our signal and use f to represent our stencil. In practice, both the signal and stencil cannot be infinitely long. One way to solve this problem is to pad 0 to both f and g to extend them to be infinitely long. Suppose there are n and m elements in f and g respectively, it's easy to show there are only $n+m-1$ non-trivial values (all others are 0).

For example, suppose $f = [1/3, 1/3, 1/3]$, $g = [3, 6, 9, 6]$, we have $f * g = [1, 3, 6, 7, 5, 2]$. The whole computation process is shown below:

$$\begin{aligned}
(f*g)[0] &= f[0]*g[0] = \frac{1}{3}*3 = 1 \\
(f*g)[1] &= f[0]*g[1] + f[1]*g[0] = \frac{1}{3}*6 + \frac{1}{3}*3 = 3 \\
(f*g)[2] &= f[0]*g[2] + f[1]*g[1] + f[2]*g[0] = \frac{1}{3}*9 + \frac{1}{3}*6 + \frac{1}{3}*3 = 6 \\
(f*g)[3] &= f[0]*g[3] + f[1]*g[2] + f[2]*g[1] = \frac{1}{3}*6 + \frac{1}{3}*9 + \frac{1}{3}*6 = 7 \\
(f*g)[4] &= f[1]*g[3] + f[2]*g[2] = \frac{1}{3}*6 + \frac{1}{3}*6 = 5 \\
(f*g)[5] &= f[2]*g[3] = \frac{1}{3}*6 = 2
\end{aligned} \tag{2}$$

We can also see from this example again that the whole computation procedure is very similar with the local weighted averaging.[1]

In this part, you are supposed the implement this convolution function which takes two vectors f and g with finite length n and m, and output the result with length $(n+m-1)$.

[1]Be careful! If you look at the formula and example carefully, you may find that the order of the weights in the stencil is reversed when we do convolution, e.g., $[1, 2, 3] * [-1, 1] = [-1, -1, -1, 3]$. In fact, there is an other operation called correlation and we will see it in the third part of this assignment. Another remark is convolution and correlation are the same when the stencil is symmetric, like the mean filter.

3.2.2 Mean Filter

One critical task in signal processing is smooth the original signal to reduce the noise. The mean filter is one of the most classical noise reduction techniques. The stencil for the mean filter is extremely simple, i.e., the stencil of length k is $[\underbrace{1/k, \ldots, 1/k}_{k \text{ elements}}]$. Then we can convolve it with the signal.

In this part, you are supposed to:

1. Randomly pick 3-4 signals from the dataset, use stencil length $k = 3$ to do the mean filtering. Draw the plots of original signal and the smoothed signal and compare them. Can you see any significant differences between them?
2. Try some other values for k in a large range, e.g., $k = 11, 101, 1001, 10001, \ldots$. Answer the following questions: Can you see the signal become more smoothed with proper k? What can you see when k is extremely large? How does the quality of signals (in terms of keeping information and reducing noise) vary with different k's? Why is choosing the parameter k properly important?
3. As we saw in class, noise may produce extraneous zero-crossings. Therefore, the number of zero-crossings might be a good quantitative measure of the smoothness of signal. Choose the original signal and the best smoothed signal we obtained above and proper threshold v. Use the function in Part I to compute its zero-crossings and compare them.

3.3 Haar Transform

Wavelet analysis is an important tool in signal processing. It is becoming more and more popular in recent years. There are a bunch of useful wavelet bases and Haar basis is the most straightforward one among them. We are going to implement Haar transform in this part.

To simplify this problem, you may assume that the input signal will always have size 2^n. Then the Haar transform procedure is:

- Compute the pairwise average, to get 2^{n-1} coefficients
- Compute the pairwise difference, to get 2^{n-1} coefficients
- Recursively do the same procedure on the first 2^{n-1} coefficients until there is only 1 element

For example, the Haar transform of $[9, 7, 3, 5]$ is $[6, 2, 1, -1]$. Here's the detailed procedure:

- We firstly compute the pairwise average, $(9 + 7)/2 = 8$, $(3 + 5)/2 = 4$
- Then we compute the pairwise difference, $(9 - 7)/2 = 1$, $(3 - 5)/2 = -1$
- In the first iteration, we get $[9, 7, 3, 5] \rightarrow [8, 4, 1, -1]$
- We recursively do it on the first half $[8, 4]$, and we get $[6, 2]$, so that the Haar coefficients of $[9, 7, 3, 5]$ are $[6, 2, 1, -1]$

Meanwhile, we can also easily inverse the procedure above to recovery the original signal from the Haar coefficients.

In this part, you are supposed to:

1. Implement the Haar transform and its inverse.
2. Randomly pick 3-4 signals from the dataset, cut it into the longest possible length which is a power of 2 (i.e., 2^n for some n). Compute the Haar coefficients of the original signal and see can we recovery it exactly from the coefficients.

3. Using some good filters we obtained from part II to smooth these signals, and then compute the Haar coefficients again. Compared the coefficients of the original signal and the smoothed signal. The coefficients at the beginning may somehow represent low-frequency information of the signal, e.g., the first coefficient is called DC, which is the global average. While the coefficients at the end (we obtained in the first, second iterations) may represent high-frequency information. Is our mean filter effective to smooth the high frequency part? Use some figures to support your conclusion.

3.4 Autocorrelation

In statistics, the correlation between two series $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ is defined as:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{Y})^2}} \tag{3}$$

where $\bar{X}$ and $\bar{Y}$ are the mean value of series X and Y respectively.

Similarly, in signal processing, we can define the autocorrelation as the correlation of a signal with itself with some lag. It's a good tool to find the repeated patterns hidden behind the signal with noise.

More precisely, we can only sample signals with finite length in practice. Suppose our signal is $X = \{x_1, x_2, \ldots, x_n\}$, we can define the autocorrelation coefficient r_t at lag t as the correlation of X_1 and X_2, where $X_1 = \{x_1, x_2, \ldots, x_{n-t}\}$ is the first $n - t$ points in X and $X_2 = \{x_{t+1}, x_{t+2}, \ldots, x_n\}$ is the last $n - t$ points in X. The plot of these autocorrelation coefficients at $t = 1, 2, \ldots$ is also called a correlogram.

In this part, you are supposed to:

1. Implement the function to compute autocorrelation coefficient of a given signal.
2. Randomly pick 3-4 signals from the dataset, compute their autocorrelation and draw its correlogram. Can the correlogram tell you about the periodicity of the signal?
3. Using some good filters we obtained from part II to smooth these signals, and then compute the correlogram again. Compare the correlograms and answer the following question. Are these correlograms similar or dissimilar? Is autocorrelation robust to noise according to your experience?

4 Assignment Submission

Please submit an archived file through CMS including your report, source code and any other necessary files to run your program and, if applicable, an executable. It is highly recommended a README file including build instructions about your code.

5 Academic Integrity

Academic integrity is important in this course. You must follow the school's code (`http://cuinfo.cornell.edu/Academic/AIC.html`).

Since this is a programming project, we would like to emphasize the following rules:

- Having discussions with other people, using open sources and public tools, getting ideas from research papers is allowed, but proper citations and acknowledgements are required. Otherwise, any direct or indirect copy from other's work, Internet, etc. is strictly forbidden.
- All the results you reported in this project must be generated by your submitted programs.

Violations of academic integrity are taken very seriously. Please feel free to contact Professor Zabih if you have any questions or concerns about this topic, or if you feel there is any possibility that you may be violating the code of academic integrity.