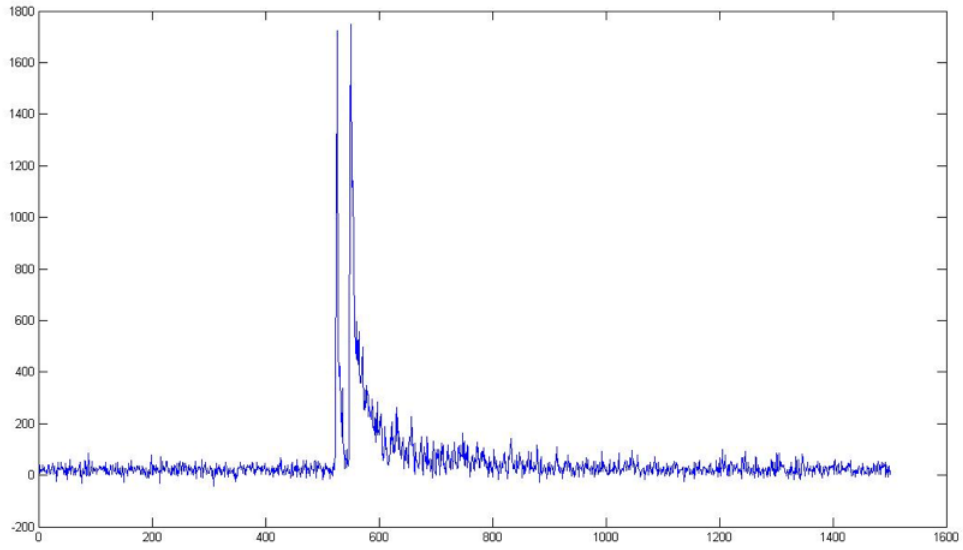
Programming Project D

Data Analysis and Visualization

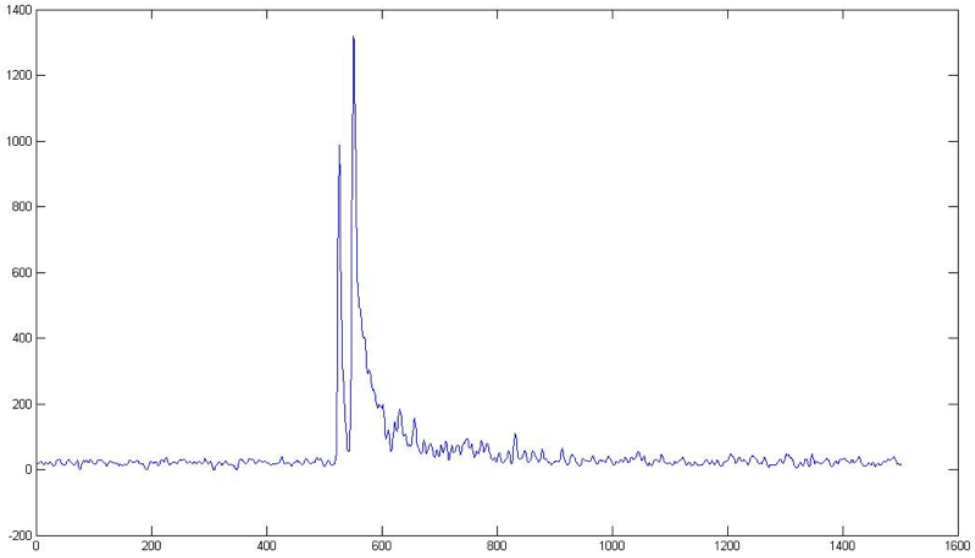
In this program you will analyze digitized data for **pulses**. The data comes from an actual digitizer that reports voltages (except all the data has been reversed by the hardware – you must negate all values before proceeding). Here is a graph of one of the sample data files (which consist of integers separated by whitespace) after negating the values:



The data is quite jagged, so we "smooth" it before analyzing it. Starting with the 4th point of the file, and ending with the 4th from the last, replace each of those points with the following average in a new array (the current point in question is pointed to by p_i):

$(p_{i-3} + 2p_{i-2} + 3p_{i-1} + 3p_i + 3 * p_{i+1} + 2p_{i+2} + p_{i+3})$ // 15 (*Note*: Use integer division)

The first 3 and the last 3 smoothed data values are unchanged from the original. Here's what the smoothed data looks like:



The smoothed data will be better for detecting pulses (you can see two noticeable ones in the pictures above).

You detect a pulse by looking for a rise over three consecutive points, y_i, y_{i+1}, y_{i+2}. If the rise ($y_{i+2} - y_i$) exceeds vt, (for "voltage threshold" – supplied by an input parameter), then a pulse begins at position *i*. After finding a pulse, move forward through the data starting at y_{i+2} until the samples start to decrease before looking for the next pulse.

Write a program that process all files with a ".dat" extension in the current directory. There can be an arbitrary number of such files, and an arbitrary number of sample data values within each file. Print out where pulses are found, and the "area" underneath the pulses. The area is merely the sum of the values starting at the pulse start and going for width samples (another input parameter), or until the start of the next pulse, whichever comes first. Use the original, unsmoothed data to compute the area, however. (The smooth data is just for detecting pulses.)

There is one "gotcha". Sometimes it is hard to distinguish "piggyback" pulses (a weak pulse followed quickly by another pulse) from a wide pulse with some variation to it. We discard piggyback pulses. To distinguish these, we use the following parameters:

Parameter	Description
`drop_ratio`	A number less than 1
`below_drop_ratio`	The number of values less than `drop_ratio`
`pulse_delta`	The gap between pulses to look for piggybacks

When *adjacent* pulses begin within `pulse_delta` positions of each other, find how many points between the peak of the first pulse and the start of the second pulse fall below `drop_ratio` times the peak of the first pulse. If the number exceeds `below_drop_ratio`, omit the first pulse from further consideration—it is not a pulse of interest.

Read all parameters from the file provided (gage2scope.ini) with the following format at program startup:

```
# Pulse parameters
vt=100
width=100
pulse_delta=15
drop_ratio=0.75
below_drop_ratio=4
```

All parameters are necessary, and none others are allowed. *Enforce this*. Lines beginning with '#' are comments and may be ignored.

If a .dat file has no pulses, ignore it. For those that do, report the pulse start positions and their areas as follows:

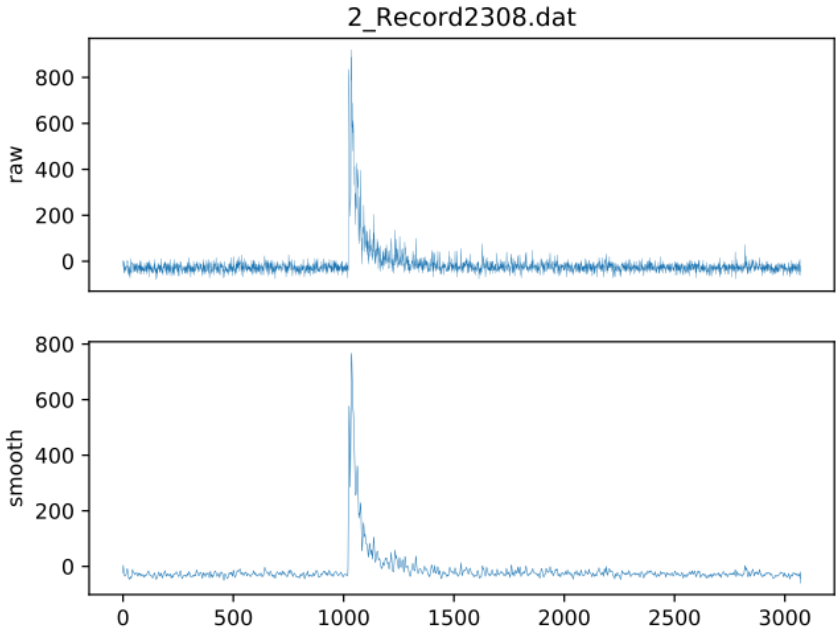
```
as_ch01-0537xx_Record1042.dat:
Found piggyback at 1020
Found piggyback at 1035
1050 (8228)

2_Record2308.dat:
1019 (3540)
1030 (22916)

2_Record3388.dat:
1020 (43272)
1035 (41264)
```

The first .dat file has a pulse of interest at (zero-based) position 1020 with an area of 8228. The .dat files and the .ini file are in the assignment folder in Canvas. (The order of appearance of the .dat files is not important).

Also, plot both the raw and smoothed data using **matplotlib**, saving it in a .pdf file. Here is the plot for 2_Record2308.dat, saved in file 2_Record2308.pdf:



Use **sys.argv** to get the name of the .ini file from the command line:

```
$ python3 analyze.py gage2scope.ini
```

Implementation Issues

Do not use list objects to hold the raw or smooth data. Use **array.array** instead. You will find that most of the common list methods also work for **array**.

Submit your *analyze.py* file (call it that), your output data screen shot, and your .pdf files.