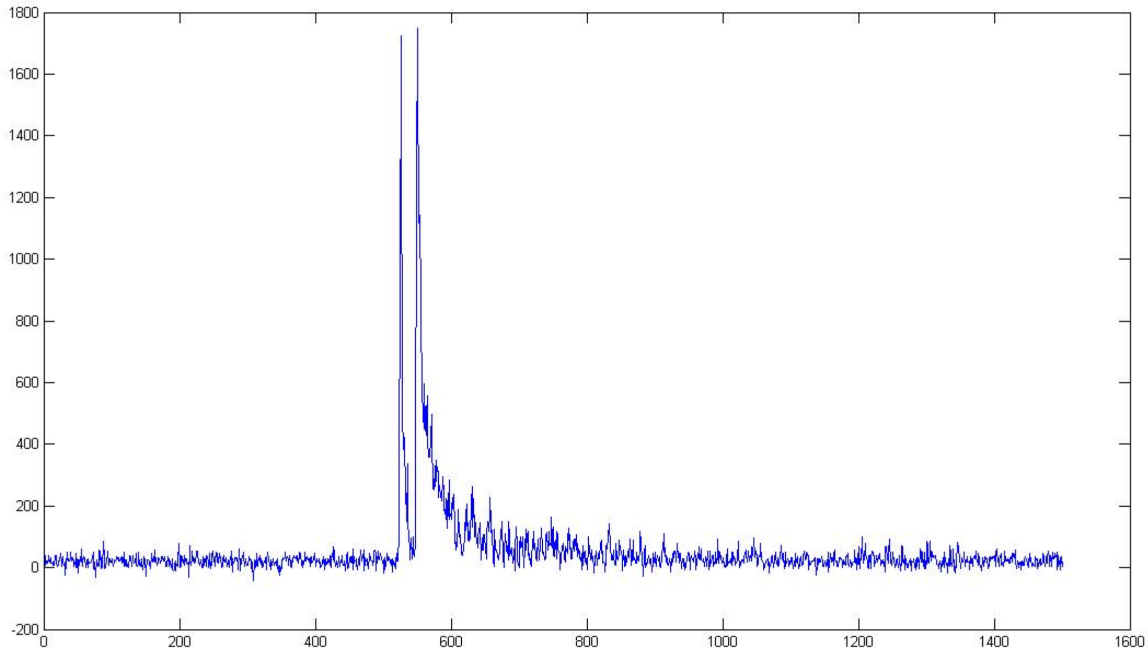
CS 3370
Program 4 – Data Reduction with C++ Algorithms

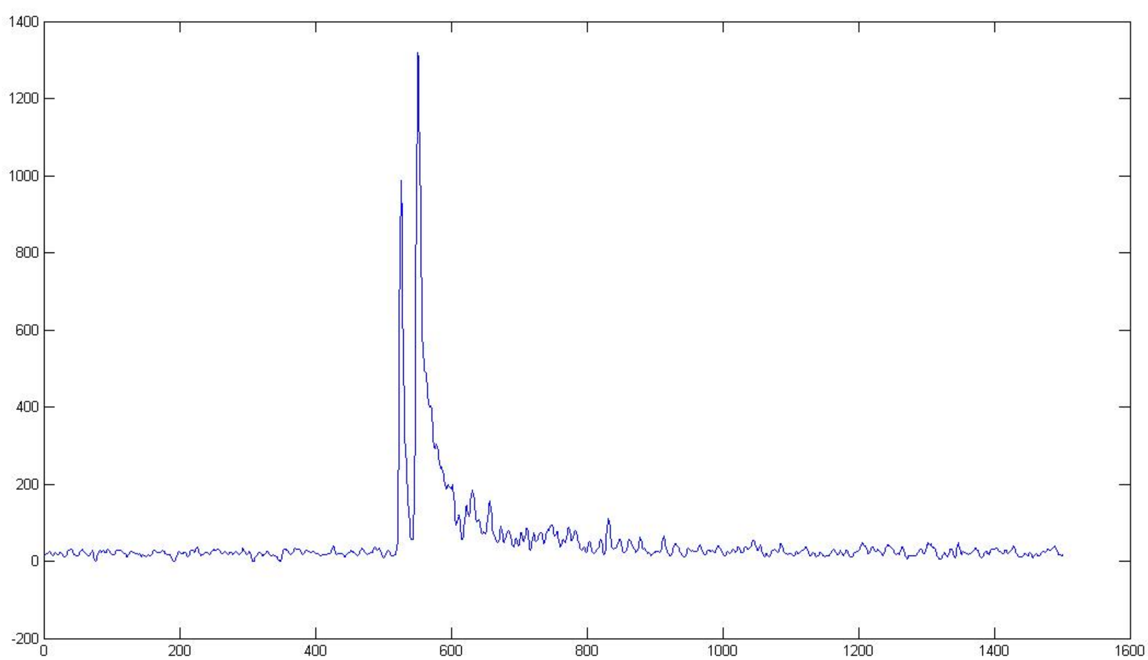
In this program you will analyze digitized data for pulses. The data comes from an actual digitizer that reports voltages (except all the data has been reversed by the hardware – you must negate all values before proceeding). Here is a graph of one of the sample data files (which consist of integers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



The data is quite jagged, so for finding pulses we will use a smooth version instead. Starting with the 4th point of the file (point [3], or course), and ending with the 4th from the last, replace each of those points with the following average (the current point in question is *pointed to* by "iter"):

```
(iter[-3] + 2*iter[-2] + 3*iter[-1] + 3*iter[0] + 3*iter[1] + 2*iter[2] + iter[3]) / 15
```

Here's what the smoothed data looks like:



The smooth 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. 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 also 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, unsmooth 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 (where a pulse is *adjacent* to another) from a wide pulse with some variation to it. 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 you find adjacent pulses that 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 (non-inclusive) 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 an .ini file like the following 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 so ignore them. Blank lines are allowed.

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

```
2_Record2308.dat: 1019 1030
3540 22916
2_Record3388.dat: 1020 1035
43272 41264
as_ch01-0537xx_Record1042.dat: 1050
8228
```

The first .dat file has pulses of interest at (zero-based) positions 1019 and 1030, with respective areas 3540 and 22916. The .dat files are in this ZIP file. (The order of appearance of the .dat files is not important).

Implementation Notes

Use STL algorithms such as **copy**, **transform**, and **accumulate**, along with **istream_iterator**s to make short work of this program and avoid some loops. How are you going to discover all the .dat files in the current directory? (*Hint*: use **system** from <**cstdlib**> in conjunction with **dir** (Windows) or **ls** (UNIX).) Verify that the .ini file is present and valid. FYI, my program is 179 lines, including whitespace and comments. I also used lambda expressions, **for_each**, **adjacent_find**, **bind**, **back_inserter**, **max_element**, **count_if**, **insert**, **erase**, and **remove** (the latter is from <**cstdio**>).