Introduction

In this assignment, we'll be looking at more 1-D signals. This time, we'll look at model fitting using least squares on data from currency exchange rates. We'll also take a look at using matched-filters to examine pulses in an EEG of seizure patients.

Completing the assignment

This assignment can be done individually or in pairs (though we strongly encourage you to work in pairs). **Note:** you should work with a different partner from the previous assignment, unless you get permission from the instructor.

As before, Python is the preferred language, and all the supporting code to handle I/O and display that we're providing is in Python.

For submission, package up your code as a zip or tar file. Include your written answers as a pdf or doc file named `writeup.[pdf|doc]`. There's a bunch of graphs we want you to plot, include these in your writeup, and please label them so we know which figures go with which subproblem. Please submit your code via CMS, so that we have it all in one place.

If you have any questions about this assignment, please contact the TA (`afix@cs.cornell.edu`).

1 Iterative Reweighted Least Squares

For this section, we'll be using least-squares methods to find trends in currency trading data. The data has the trading prices of various currencies, taken hourly over several days. See Figure 1 for an example. If we're ever going to make millions playing the market, one of our first tasks is trying to predict trends in the data. Fortunately, we already know about least-squares, and can use it to fit a line to the data!

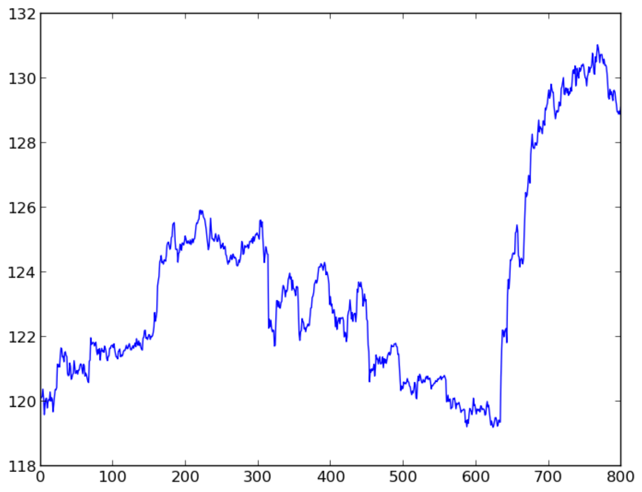

Figure 1: Trading prices of Euros to Japanese Yen.

Run the provided script `financial.py`, which runs weighted least-squares (with weights set to 1), and finds the best fit. As you can see, it doesn't really represent the data well. The problem is that our data seems to have at least 3 regions where the data is basically linear (with some noise), but these regions each has a distinct trend.

In order to fix this, we're going to use outlier suppression to isolate just the middle, mostly linear part of the data. Our tool for doing this is IRLS: Iterative Reweighted Least Squares.

Recall that IRLS deals with outliers by instead minimizing a cost function that looks like

$$\rho(r_i) = \min\{r_i^2, s^2\} \tag{1}$$

That is, we truncate the cost to s^2 once the residuals grow too large. Sample points i with $\rho(r_i) = s^2$ are our outliers.

Note that we can control how many inliers and outliers we get by changing the parameter s. We may have to play around with the value of s to get the results we want.

Your tasks for this section:

- The provided code gives you an implementation of weighted-least squares. Implement IRLS on top of this, using the weight-update rule we gave in class.
- Experiment with the value s until your least-squares result most-closely fits the middle section, without throwing away too many data-points. This is going to be somewhat subjective.

  Graph the resulting fitted line. You should plot the outliers in a different color from the inliers (controlled by the second argument to `pylab.plot`). Compute the number or inliers vs. outliers for the s that you choose.
- Run IRLS with the same value of s for each of the other currency data streams, and plot (but don't include in your writeup) each result. Experiment with changing the value of s to get a better fit for the different data sets. Describe in your writeup (a couple sentences) whether the value of s is robust or not (i.e., works for a variety of data sets) and what qualitative features of the data tends to make IRLS work well.

  The other data sets are `GBP-JPY.data`, `GBP-USD.data` and `USD-JPY.data`, each from the same time period as Figure 1. There is also a longer data stream `EUR-GBP.data`, which is much longer. Make sure your algorithm can handle the larger data set.

2 Segmented Least Squares

IRLS can work for predicting models where the signal is basically linear with a few outliers, but when the model looks like this financial data, we'd really like to be able to find a model with multiple linear pieces. With this richer class of models, we can better capture data which looks like the EUR-JPY graph: basically linear, with a few points of rapid change.

Segmented Least Squares is exactly built for this kind of task. Recall from class that SLS finds the best division of the domain into segments, and then fits a separate linear function to each segment. We talked briefly about how we can accomplish this via dynamic programming. Here are a few more specifics. We have a scoring function which penalizes both the sum of squared differences (like least squares) but which also penalizes for using a large number of segments. There's a tuneable parameter c which controls how much we penalize for having lots of segments.

A possible solution is given by some number of breakpoints $b_0, \ldots, b_L$, and on each segment $[b_i, b_{i+1}]$ we use least squares to fit a line. Let $\texttt{ls}(b_i, b_{i+1})$ be the least-squares cost for the segment `x[b[i]:b[i+1]]`.

Then, our overall cost is

$$\sum_{i=0}^{L-1} \texttt{ls}(b_i, b_{i+1}) + cL \tag{2}$$

We can use dynamic programming to find the best solution. Let $Opt(j)$ be the best SLS fit for `x[0:j]`. If we've computed $Opt(j)$ for $j \leq n$, then we can compute $Opt(n+1)$ by

$$Opt(n+1) = \min_{j<n} Opt(j) + c + \texttt{ls}(j, n+1) \tag{3}$$

Your tasks for this section:

- Using the Least Squares algorithm we gave you (you can use `weighted_leastsq` with the weights set to 1), implement SLS.
- Experiment with choosing the best weight c so that SLS fits roughly 3 segments to the EUR-JPY dataset, and graph the resulting model.
- Experiment with using SLS on the other currency files, and again describe in a couple sentences how robust SLS is to the choice of regularization parameter. Also, subjectively compare how well SLS fits the data compared to IRLS from part 1.

  Note: since the EUR-GBP data set is much larger, you should find more segments on this instance.

3 Matched Filters

In this section, we'll be looking at some medical data, from an EEG of a seizure patient, anonymized as "Wonder Woman". We've given you 6 signals, each about 20 seconds worth of data, with 3 recordings taken during a seizure, and 3 recordings during otherwise normal activity.

As you can see from Figure 2, the seizure recording has very distinctive regular pulses, which vary slightly in strength and shape, but are all roughly alike. We would like to detect these pulses, which allows us to distinguish the normal recording from the seizure recording.

We're going to use matched-filters for this job, as they give us an obvious way to detect repeated instances of a pulse in a signal, even in the presence of noise and other distractors. Because these signals are naturally-occurring, and not generated, there isn't an already-defined pulse to use as your matched filter, so you'll have to come up with your own. We recommend finding a particularly clean pulse in the data, and using that as the starting point for your filter.

Your tasks for this section:

- Find a pulse to use as your matched filter, and graph it (include this in your writeup).
- Implement the matched filter algorithm using this filter, and run it on the data, graphing the result (just include the graph for `seizure1.data`). This should be mostly flat, with sharp peaks at each of the pulses in the seizure data.
- Use an algorithm similar to the peak-finding algorithm from Assignment 1 to find exactly at what times the pulses occur. For instance, you could find sample points `x[i]` in the matched filter result which are bigger than their immediate neighbors, and which are bigger than some height. You may have to tune the parameters to get a good result.

  Just to double check that this is working, there should be 9 pulses in the 3 seconds of seizure data graphed in Figure 2. Plot the peaks your algorithm finds for this section, and include this in your writeup.

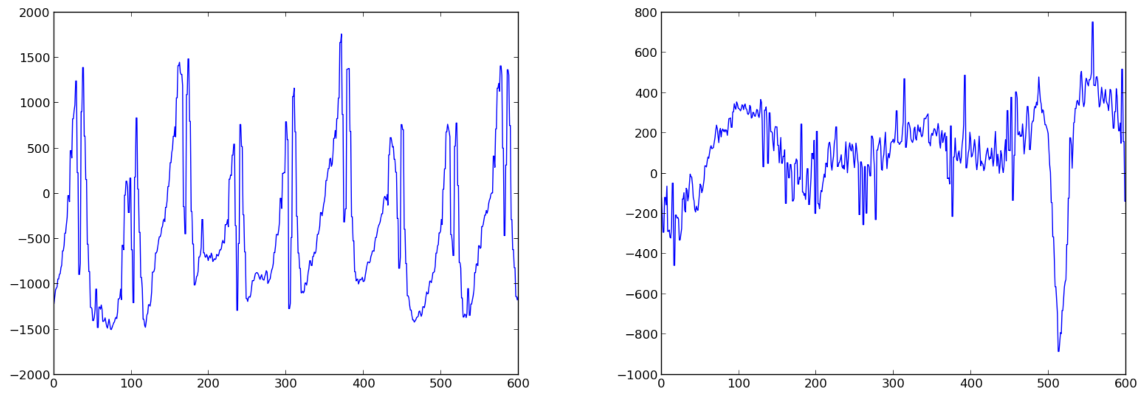

Figure 2: At left, 3 seconds of one of the seizure signals. Note the characteristic peaks (with an up-down-up pattern) that are absent in the normal EEG on the right.

- There aren't any pulses in the normal data, so hopefully your algorithm won't report any pulses in these signals (although it may be somewhat difficult to completely suppress false positives). Graph the matched filter output for `normal1.data`, and include this in your writeup.