Plotter Salter Smoother Report

By: Dillon Groh

Abstract

This section of the project required us to write and code programs to graph a function. We were required to run through a function using three different techniques to plot data on a graph. For all the parts of the project, we had to pick a function and then plot the points and line of this function. After we were able to create the graph, a second graph or line was needed to create where we had salted through all the data points of the graph and then graph those as well. Finally, we would make another method to smooth over the salted data by applying a running average to the graph to make our third and final graph/line. This project was broken into 3 separate parts where in each part we had to construct all that data using only Java applications, Octave, and Java libraries. The first part only required us to print out csv files for each of the three graphs. The second part required us to use software called Octave and use their methods and API to create the lines for the graph. Finally, the third part of the project required us to use Commons Apache Math and JFreeChart libraries to construct these graphs as well. This was very insightful because it gave us three various ways of producing the same type of data. Having a variety of different ways to attack a problem is an excellent way to tackle problems when working.

Results

Part 1 (Java and CSV)

The function I used for this project was and here are the results from creating .csv files using Java:

A picture containing chart

Description automatically generated

A picture containing text, crossword puzzle

Description automatically generated

Table

Description automatically generated with medium confidence

Conclusion of Part 1

What do all these graphs mean? Well, the first graph is just your basic representation of made into a .csv file and has had the points plot on a graph. The second graph is salting through the data. If you don’t understand what that means, it basically means that for every point of the function, a random number from the range [-2,2] will be added to the point. The start and endpoints of the graph remain unaffected, but the rest of the graph has all these different values added to them. This is why the second graph doesn’t seem very uniform because it is in the right ballpark of what the original point was but has either been added or subtracted by 2. The third part of the project was for the program to smooth over the salted data that was provided in the second graph. This includes a running average for each point it iterates over. A certain number of points (or window) is needed to calculate the running average by taking the points to the right and left of the point that you are trying to find the smoothed number of and then calculate the average from the number of numbers in the window. When smoothing, you must run the function twice in order to get a “smoother” result. For every single part of the project, this was the hardest part to implement into the code. Overall, this resource is valuable because if you are working with common people in the workplace, most do not have knowledge of Java, Octave, or Libraries, but know how Excel works. This is the best way to show a basic graph without needing any external tools besides coding knowledge and Excel.

Part 2

Application

Description automatically generated with medium confidenceFor this part of the lab, we are still going to be using the function, , but we will be using a program called Octave. We were given the choice of using Octave or Matlab, but Matlab is expensive and only on school computers, while Octave is free to use.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generatedConclusion of Part 2

The Octave program or API I thought was very useful to construct lines. There was a learning curve to understanding how it worked. I followed this website, <https://docs.octave.org/v4.0.0/Two_002dDimensional-Plots.html>, to help me with figuring out how to graph a function in Octave. The running average was a bit of a challenge to me as well, so I used this resource, <https://docs.octave.org/v6.3.0/Statistics-on-Sliding-Windows-of-Data.html>, which gave me an understanding of using the movmean() capability for Octave. The hardest part of this section for the project was understanding how Octave worked and if I was going to input the function, or just the points.

Part 3

The final part of this section for the project was to use the same function that I have been using in the previous parts, but to import the Commons Apache Math and JFreeCharts libraries to create graphs of my function. The .jar files needed to be added into the IDE I was using so I could use these libraries in the first place, which provided a challenge. The blue line is the normal plotter function. The red line is the function that has had each of the points salted. The green line is the function that has had the salted data point smoothed over.

Chart, line chart

Description automatically generated

Conclusion on Part 3

This section of the project was the hardest and took the longest for me. I struggled after doing the first two parts of the project easily, so I ventured off and decided to work on the other areas of the project before tackling this one last. I had trouble with knowing which .jar files to import out of the 5 that it gave for Apache and when I downloaded the JFreeCharts most up-to-date version, there was no .jar file included in the .zip file so I had to download an older version which ended up having a .jar file. The implementation for the code was also tricky, and it became a challenge to create the chart using JFreeCharts methods and functionalities. I also ran into the issue of only having JFreeCharts and not having JCommon in my library as well. I did not realize for a while the JCommon was required to make JFreeCharts work, and I’m embarrassed to say that.