The Learning Process For JFreeCharts

For this assignment, basically the only source of my research was TutorialsPoint, save for the occasional StackOverflow lookup. I had never used any system like JFreeCharts or Apache Commons before, but I've certainly worked with different libraries and was used to going into APIs for information on classes and their methods.

When I was looking at JFreeCharts, I realized that there were three main steps that I needed to figure out in order to complete the program.

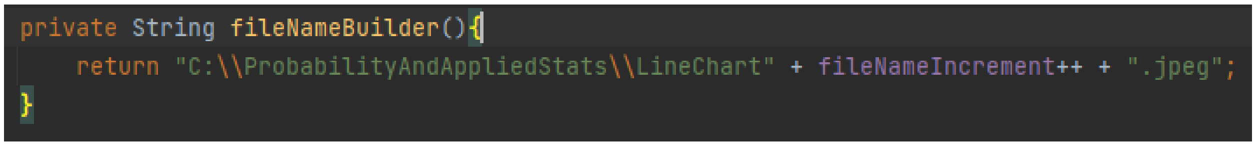
1. Create a dataset and populate it with information
2. Utilize the ChartFactory class to generate a line chart
3. Display the chart somehow

Luckily, steps one and two were extremely easy. TutorialsPoint had a variety of examples showing how to use the ChartFactory class, and each of them showed how to insert data into a dataset for the chart to use. I ended up copying and pasting an entire section of code from their website and sticking it into Intellij to see how it exactly worked (See it in TestingChart.java). After reading through it and doing a bit of experimenting, I had a decent understanding of what I needed to do. However, the site didn't have all that much information about how to really incorporate the graphs in a GUI other than using a Chart Panel, which I was unsure about using. This is when I remembered the MVC program I had written for the software design patterns. It already had a GUI ready and had most of the meat of the program already implemented. I simply needed to tweak it and add the extra features from JFreeCharts and Apache Commons.

The MVC program had a text area that would display the output of the function, but I of course needed it to display the graph instead. This is where one extremely convenient class comes in: ImageIcon. It allows me to generate an image of the chart and place it in my files. All I had to do was create a JLabel and add the chart image to it, and it would display easily. However, this is where I ran into my main issue.

Do you recall the Java web app we were creating with html on Eclipse, where I had the issue with Eclipse not "updating" its files properly that I needed? I had the same issue with Intellij. Every image generated by the program was called "LineChart.jpeg", and I would simply overwrite the image when the user wanted to graph something new. Unfortunately, Intellij wouldn't recognize the change in the image until the program was ended and rerun again. I can only assume that when the program runs, any files it uses are saved in some way in the program's memory, and so any changes to those files aren't reflected at all. This meant that I could only ever graph one single image, and any attempt at clearing the GUI and plotting something new was going to fail.

Luckily, I discovered a nice workaround.



```java
private String fileNameBuilder(){
    return "C:\\ProbabilityAndAppliedStats\\LineChart" + fileNameIncrement++ + ".jpeg";
}
```

By simply changing the name of the file with every attempt at graphing something new, I forced Intellij to update its information and use the new file's contents. As a result, the user can now graph a certain function, change it, clear it, then graph the new function.

The Learning Process for Apache Commons

Figuring out how to implement the smoother for Apache Commons was probably the most confusing thing to research for me. I made the initial mistake of using Apache Commons 4.0, which is actually a beta. Lesson learned, as I had a lot of trouble finding information about the library when it was generally new and not completely developed. Any information I did find was about version 3.6, which was often incompatible with the beta version. After a few attempts at making it work, I ended up switching to 3.6.

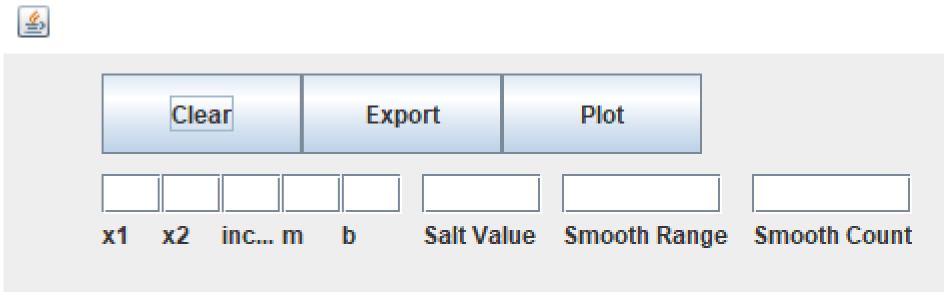
I found that there were no dedicated functions or classes for smoothing or salting data the way I needed. In fact, the closest I found to a smoother was the SmoothingPolynomialBicupicSplineInterpolator class, and it was too much to unpack for me to even attempt using. Instead, I asked Alex and he told me about the DescriptiveStatistics class he used. It comes with two extremely useful functions for the smoother: a window and a mean method. This made the smoother extremely simple, since all I needed to do was add each index to the window one at a time and take the average of it. I placed the salter and smoother in the Model class and added some new components to the GUI to work with them.

Instructions Manual

Important Notes

Before using the program, it may be necessary to go into the code and adjust file paths. In the Model.java file, lines 146, 173, and 188 contain the file path specific to how my files are structured. Additionally, you may find TestingChart.java and TestingSmoother.java useless, and they are. They were used in the testing and learning process and aren't used in the final product.

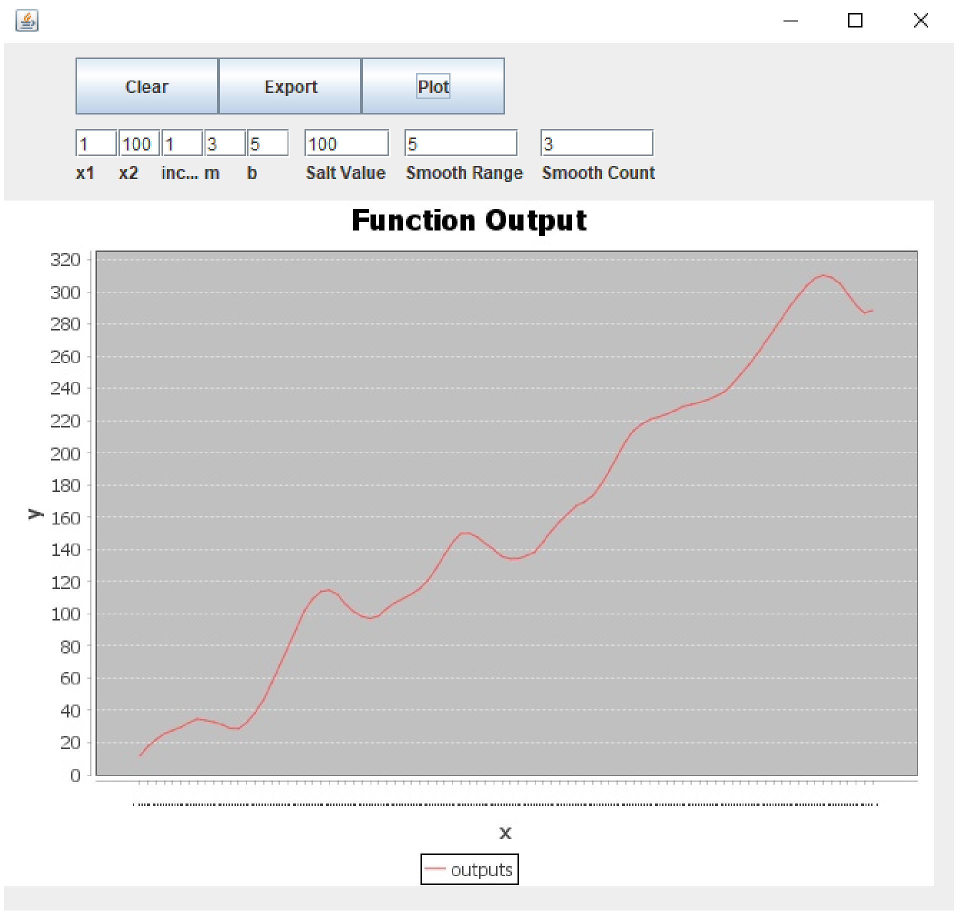
When running the program, you'll be met with this:



The five input boxes on the left, x1 - b, are for specifying what function of y = mx + b you'd like to see graphed. x1 and x2 combined specify the x range, inc is how much the x increments when you go to the next value, and m and b are for slope and b is the value of y when x is 0. Each of these fields are mandatory, and the x1 should be less than x2, naturally. An example of acceptable inputs would be [0, 19, 1, 2, 6].

Salt value, smooth range, and smooth count are for the salter and smoother's methods. While these fields are also mandatory, putting 0s in them will effectively mean that you aren't salting or smoothing them at all. So if you only care about the function itself unchanged, then leave them as 0s.

Plot will display the graph that is created from your inputted information onto the screen.



x1: 1, x2: 100, inc...: 1, m: 3, b: 5, Salt Value: 100, Smooth Range: 5, Smooth Count: 3

The x values can’t be shown properly when the range is very high.

The clear button will simply remove the graph from your screen and should be used before attempting to create another graph with new information. If you don’t use the clear button, the information in the program won’t be reset when you display the next graph.

The export button will take the generated graph and place it into your directory as LineChart.jpeg.