Shady Haddad Project 2 Documentation

About:
This final project really taught me patience and resilience. There were a lot of points in this project where I felt defeated and that I wouldn't figure out the solution, but that wasn't the case. I was able to get everything done the way I wanted. I hope throughout my documentation you can see the efforts I put into this project. There were 3 parts to the Plotting, salting, and smoothing part. I thought the first part was the easiest because I had to just code in java and just look up a function I wanted to graph. The second part, which was learning octave, was challenging. We had to pick up on a new skill and watch some tutorials on it but I'm all about learning. I can see how useful it is especially with how nice my graphs looked. Finally, the last part of that section was using Jars/Jfreecharts which I found super annoying. I ran into constant problems that I talk about in my report, but man was I fed up. The second part of the entire project was going through our textbooks and finding a homework problem from each section and remaking those problems based on a dataset we chose. This part of the project was not hard, it was rather interesting because you had to put a lot of ideas into your head and come up with conclusions based on your dataset. I found it interesting how you can relate probability and statistics to datasets especially when it involves my hobby which is watching anime. I got some cool results from my problems which I even talked to my friends about.

Table of Contents/Overview

END

PSS 1

Purpose (personal Opinion):

The purpose of Plotting, Salting, and Smoothing a function in java is to see the comparison between all of them, and how you can manipulate your functions. Depending on specific bounds you set your range to for your salter or smoother it can help your data look better or worse. I learned that depending on the function you graph you need to adjust your range. For example, X^4 has y values that run from 0 all the way to 6 million. At first, I saw no differences in my graphs, and I realized that was because of my range I set the values to. I only added a random salt of ±5 which was practically invisible. As soon as I adjusted the range it looked a lot better.

Write up With Parameters of my Program:

Over the past few weeks, I spent some time experimenting with every piece in the Data Plotter/Data Salter/Data Smoother system to see how each parameter affects both the appearance of the graph and the quality of the exported data. Below is a summary of what I tried and what I learned from the process.

1. Changing the x-range and step size

Originally, I started with x running from −50 to +50 in unit steps. That gave a clean, dense quarter-window of points. After adjusting I extended to -100 to +100 and kept the same step size but doubled the domain. The curve still plotted correctly, but the points as the extremes became so large which meant that nothing near the origin was visible unless I changed my vertical scale completely. Switching the domain made the curve look smoother but doubling the number of points, which slowed the CSV export but not significantly.

2.) Salt range experiments

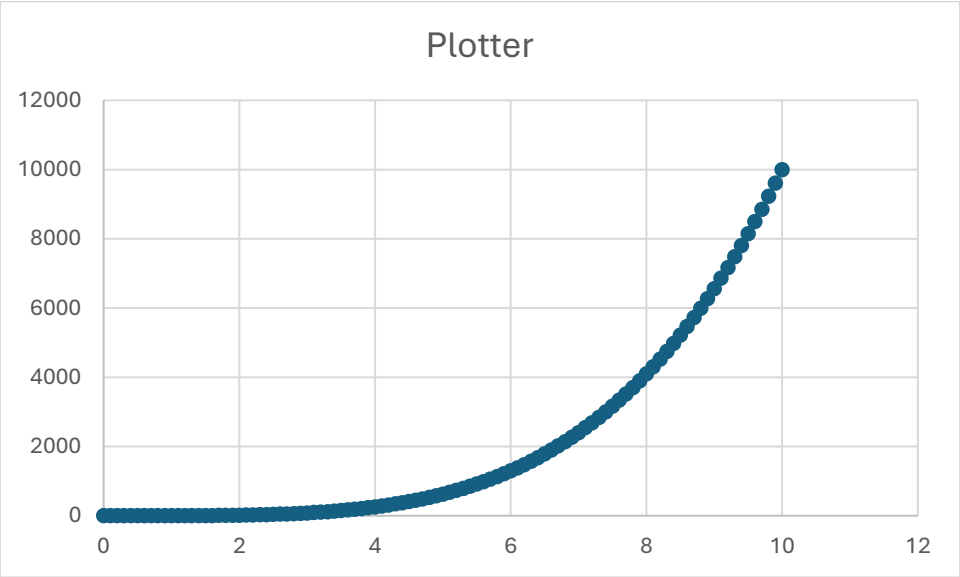
I started my salt on a ±5 on a curve that reaches millions, so it was imperceptible. I should have known from the start but it's okay, it's about learning. Then adjusting to ±500000 made the curve look fuzzy but it still maintained the trend. I wanted the salter to be very noticeable for my smoother to make a bigger impact.

3.) My Main Takeaways

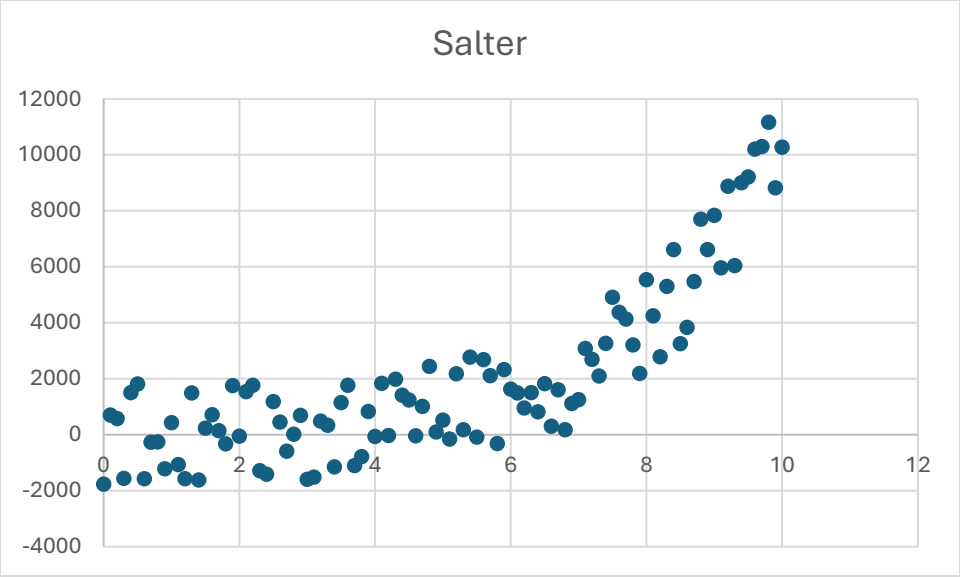
The visible effect of adding noise depends as much on your plotting scales as on the noise magnitude itself. Using a noise range relative to the data value can produce more natural fuzziness that grows with the signal. There's a trade-off between resolution and performance, for dense plots. Finally, exporting and avoiding oversized in-memory lists kept both salting and exporting easy even as data size grows.

PSS 1 Graphs

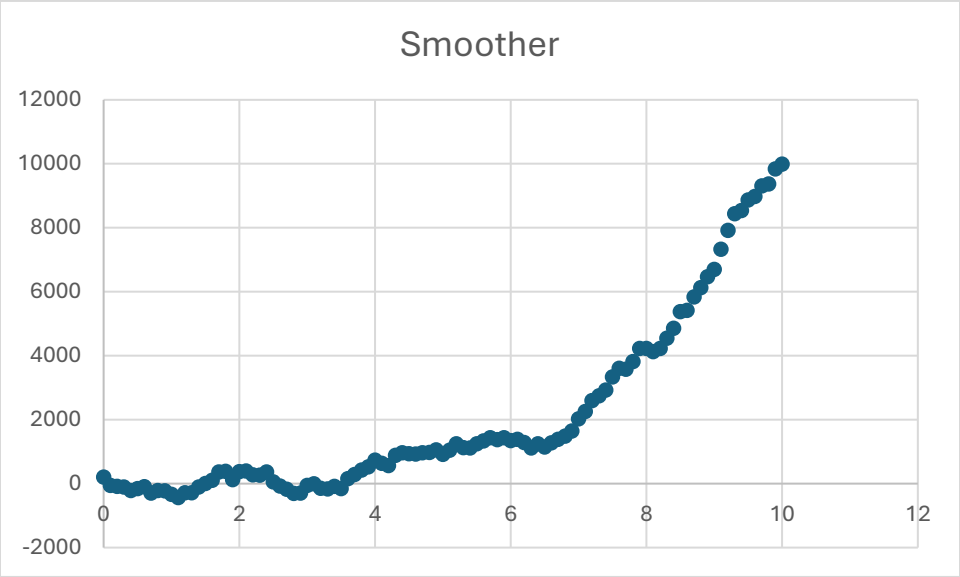
Graph of Points Plotter:



Graph of Salted Points:



Graph of Smoothed Points:



Analysis:

I will give an analysis of my data on my Jfreechart section of this because I do not just want to repeat myself.

PSS 2

Purpose (Personal Opinion)

Working through this project in Octave was eye-opening in just a few lines of code I generated the x^4 data, "salted" it with random noise, smoothed it with a simple rolling filter, and plotted all three stages side by side. Octave's syntax made array operations and vectorized math trivial with no slow loops while its built-in functions for random number generation, filtering, and plotting let me prototype the entire pipeline in minutes. It was a lot less code than the first part of the project which was expected. Obviously, if you gave us a tool it would be for something useful. Seeing the raw curve, the noisy data, and then the cleaned-up result all at once really showed how easily Octave handles data processing and visualization without struggling with complex libraries. Overall, Octave proved to be a lightweight, free, and surprisingly powerful tool for quickly experimenting with and visualizing data transformations. It was fast and I learned how to use it relatively fast.

Octave Tutorial

References:
https://www.youtube.com/watch?v=woiU5PRVm7M
https://docs.octave.org/octave.pdf

Write up:
I didn't do anything crazy for my tutorial with octave because I wanted to work on the project aspect of it right away. I watched a YouTube video that showed me the basics of how strings worked in octave. Then I used this pdf that octave provided to learn the necessities on what I needed.

<unnamed>

```
typeinfo(3)
typeinfo(3.402)
typeinfo pi)
typeinfo([1, 2, 3, 4])
typeinfo([1, 2; 3, 4])
typeinfo("hello")
typeinfo('more text blah')

ans = scalar
ans = scalar
ans = scalar
ans = matrix
ans = matrix
ans = sq_string
```

These were just different strings I typed out. Then I wanted to test out a math equation with it so I used square root.

```
num = input("Give me a number and I'll give the square root!")
sqrt_num = sqrt(num);
disp(sqrt_num)
```

Then when I typed a number out it displayed the square root for me.

```
>> sqrt
Give me a number and I'll give the square root!
```

The skills I needed to learn

1. Basic Octave syntax. Scripts vs. functions: knowing how to put your code into .m files, write functions and call them. I found this on page (pp 7–15) Also data types and matrix operations were on (pp 51–65).

2. Reading and writing CSV csvread / csvwrite. Chapter 11 “File I/O” (pp 400–430):
   • csvread, csvwrite (9-parameter forms)
   • dlmread, dlmwrite, fopen/fgets/fgetl/fscanf/fprintf, etc.

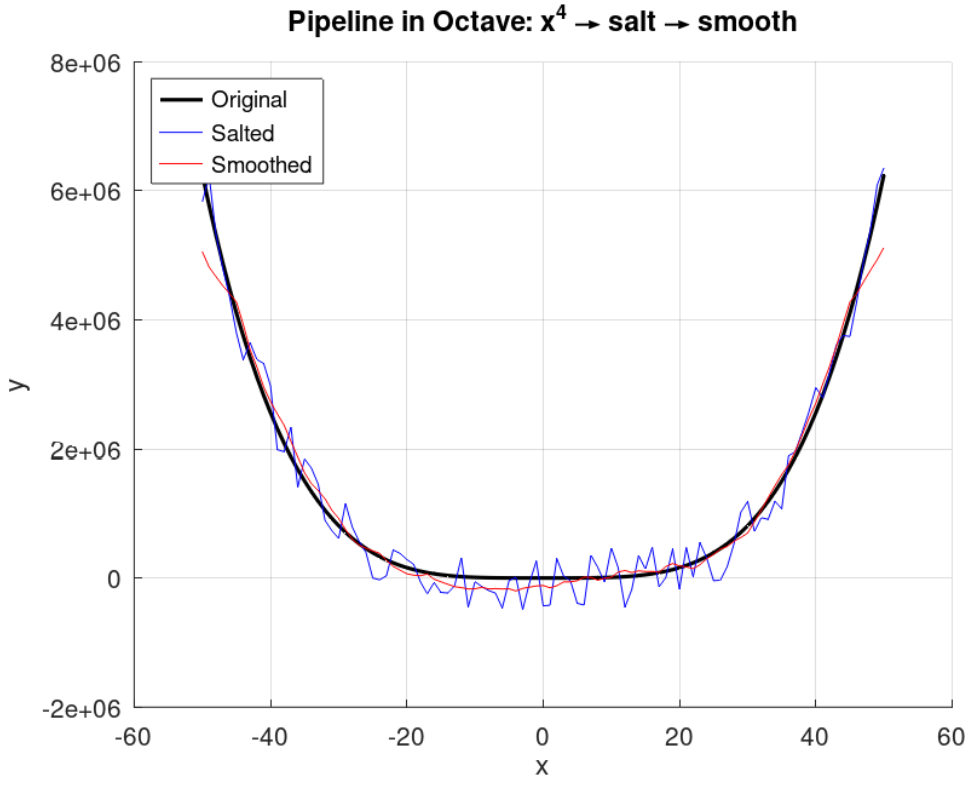
3. Plotting commands
   Chapter 10 “Graphics” (pp 320–350):
   • figure, clf, hold on|off

- plot(x,y), scatter, stem
- Axes management: xlabel, ylabel, title, legend, grid, axis

4. Debugging & plotting interactively
   Chapter 6.5 “Debugging” (pp 180–200): dbstop, dbstep, dbstack, keyboard. Using the breaking prompt to inspect variables
   Learning to inspect variables with the workspace pane or simply typing their names.
   Using pause, disp or printf to print intermediate results.
   Zooming in on figures, exporting them.

All these skills I learned were from the textbook and they helped me tremendously to finish this project I recommend every student look at it and if you want to give them this as a reference for the future go right ahead.

Plotter/Salter/Smoother screenshot:



Write up:

Here's basically what's going on in my final chart:

- Black line = the perfect X^4 curve I started with
- Blue line = that same curve but sprinkled with random "salt" noise
- Red line = the noisy data after I ran it through my smoothing filter

Notice how the noise makes the middle part around x=0 all jumpy, while the ends still sort of follow X^4. But once you apply the smoothing step, the red line almost glues right back onto the original black curve especially in the flat center, where it really cleans up those spikes. It shows that even if you totally trash your data with noise, a decent smoothing routine can pull out the real pattern again. I'm not sure if you wanted all the graphs separately or not but I loved looking at them all together because I can compare them well. I love seeing how it lines up under each other so I can determine the accuracy of my results.

END OF PSS 2

PSS 3

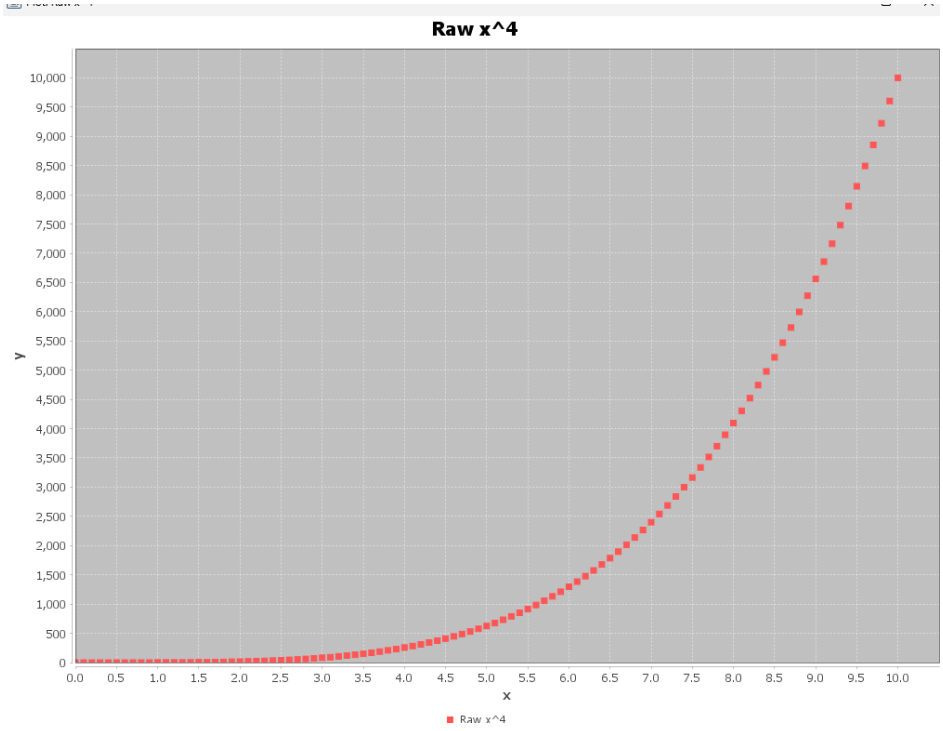
Jars/Jfreechart and Apache Stats Library

Write up: Using Jfreecharts and Apache Library was extremely agitating to me. I felt like I would go in constant circle for hours over stupid mistakes some that I'm embarrassed about. The main one that I laughed at myself for was that in my plotter2 I made it private, so my salter and smoother were not able to see the files needed to run. I felt like an idiot when I ran into that problem because it caused me like 3 hours of time. For some reason when I first downloaded the Jar files they were not running correctly. Turns out there are multiple sources with different Jar files, and I had to look on 3 different websites until I found the proper one. On top of that Jfreecharts would cause constant errors saying that it was not recognized in my folder. I had to keep re-shaping my folders and files in all kinds of ways until finally I saw that I had my Json packages in the wrong spot which caused a ton of errors. It wasn't refreshing anything when I ran the command until I fixed the folder/file placement issues. Regardless after everything worked out it paid off. I learned how crucial it is to put all required JARs (JFreeChart, JCommon, Commons-Math) on the runtime classpath. A single missing or mis-referenced library leads to the infamous NoClassDefFoundError (sooo annoying). By dropping the old Commons-CSV parser and switching to pure Java NIO (Files.newBufferedReader / String.split), I eliminated deprecation warnings and simplified my build. These problems took me forever to figure out. I'm not sure if I went to complicated into this part of the project or not, but I eventually got it to work. Overall, this exercise reinforced practices for dependency management, incremental testing, and lightweight CSV/plotting workflows in Java. The skills that directly apply to real-world data-processing and visualization tasks. You always gave us problems that directly correlated to real working experience which I'm thankful for because we will not get that teaching anywhere else.

Plotter:

My first graph, Raw x^4, really shows how crazy fast x^4 grows once x gets bigger. From x = 0 to around x = 5, the red dots barely lift off the x-axis y stays under a few hundred but after that, the points shoot straight skyward: by x = 8 you're already at 4,096, and at x = 10 you hit 10,000. Plotting every tenth of a unit made it obvious that small x gives tiny y, then suddenly y explodes. It was satisfying to see my code correctly spit out the data and for JFreeChart to render exactly the curve I expected.

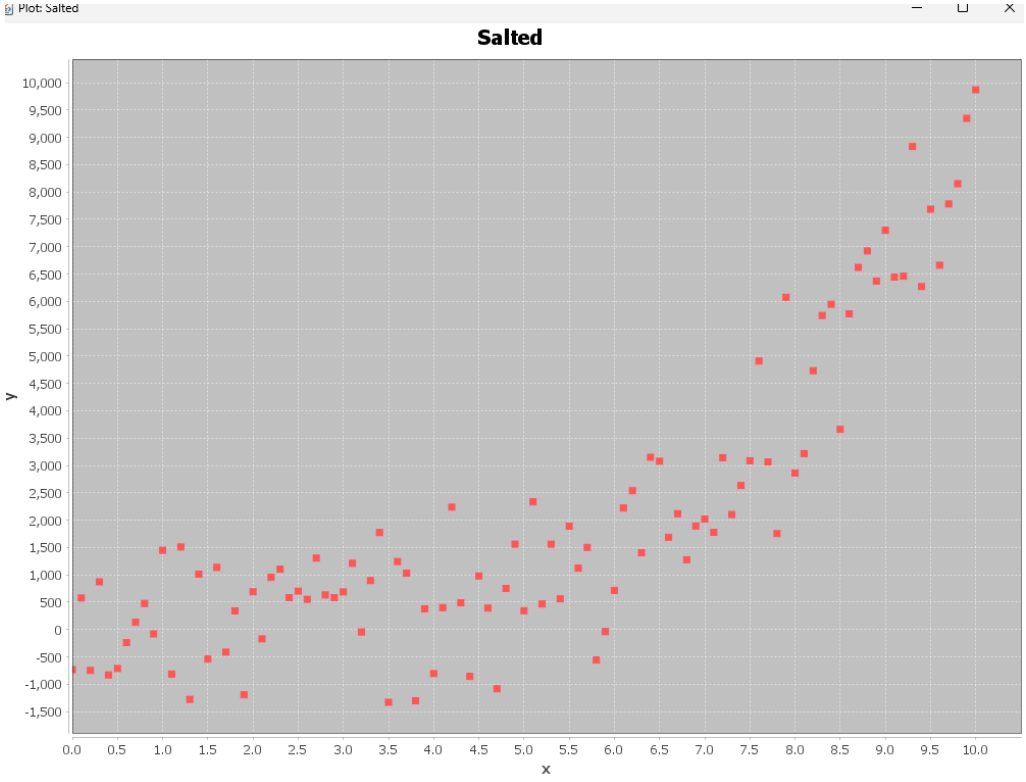
Screenshot below:



Salter:

In the Salted plot, you can instantly see how adding random noise scatters the clean x^4 curve all over the place. In the low-x region you still spot a loose band rising from near zero up to a few hundred, but individual points jump above and below the trend by hundreds or even more some values dip below zero or spike up past 1,500. As x grows, the same randomness rides on the quartic growth, around x = 6 the points straggle between a few hundred and 3,200, and by x = 10 you've got salted values reaching as high as nearly 10,000 or dipping as low as 6,000. It's a great visual reminder of how real-world data experimental error, or natural variability can blur a good equation, and why smoothing or filtering is so critical before drawing conclusions. I wanted my salter to be kind of crazy because I wanted to see how significant my smoother could be.

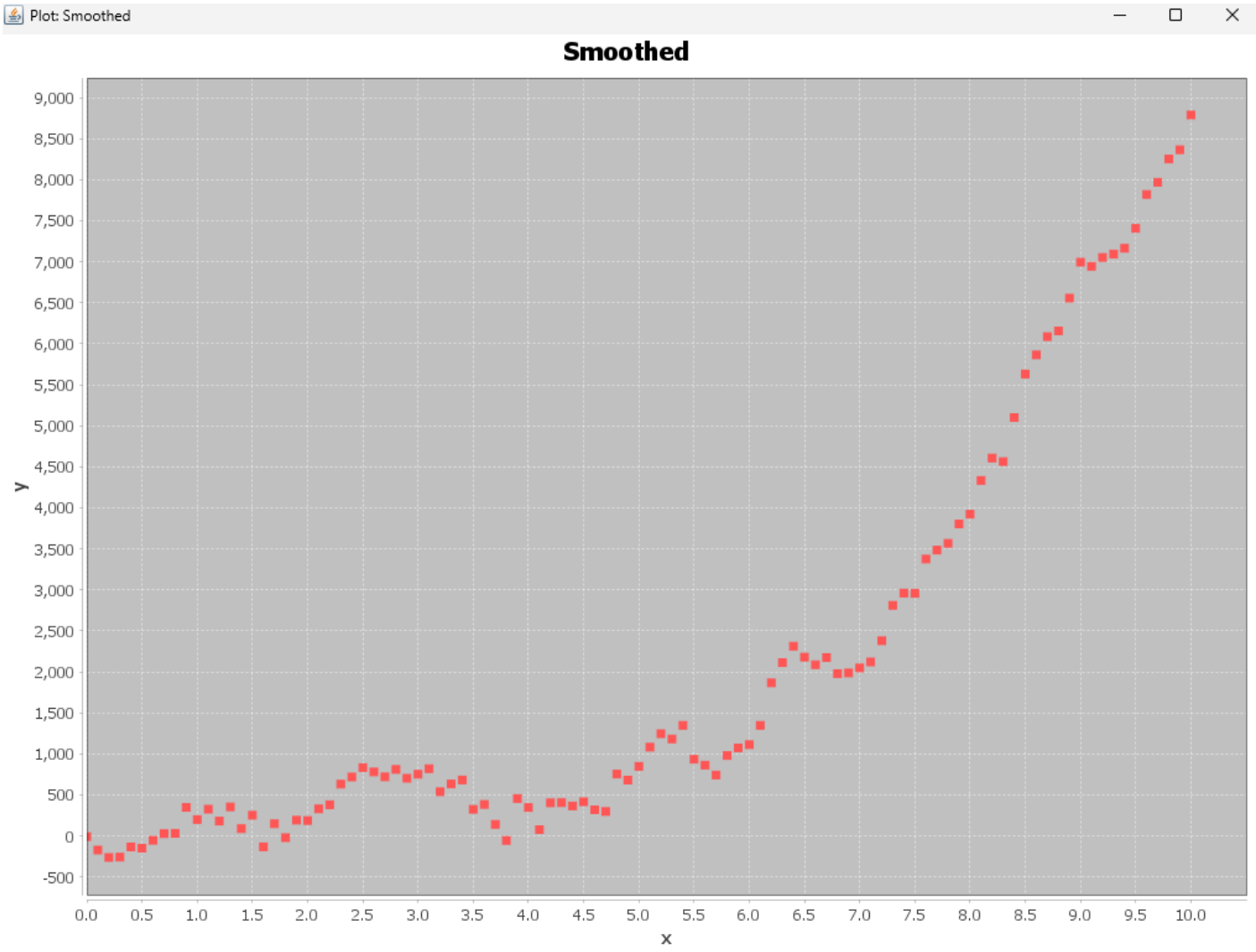
Screenshot below:



Smoother:

The Smoothed plot really brings home how a simple moving-average can recover the underlying growth trend from noisy data. I thought my smoother did a great job especially because of how my salted graph looked. I'm not sure if you think it isn't good enough smoothed graph or not, but I had it is. Instead of the crazy scattered points in the Salted graph, here you see a gentler, more coherent curve. Up through x=2 the smoothed y-values hover around a few hundred, then they climb steadily roughly 1,200 at x=5, about 2,000 at x=6.5, then accelerating past 4,000 by x=8 and nearly 8,500 by x=10. Of course, I know my smoothing graph isn't perfect, but I was really satisfied with my results.

Screenshot Below:



<u>Final Writeup:</u>

These three charts show the full journey from pure math to messy data and back again. the Raw x^4 plot underscores the explosive nature of a quartic relationship, the Salted plot demonstrates how even modest noise can completely obscure that underlying curve, and the Smoothed plot illustrates how a simple moving average can pull the truth back into focus. By building and visualizing each step, I not only confirmed that my data pipeline generation, and smoothing works correctly, but I also gained an intuitive sense of why mathematicians and engineers spend so much time cleaning data before trusting their models. Seeing the clean growth explode, then get buried in randomness, and then re-emerge through smoothing felt like a mini case study in real-world data analysis. I thought it was a perfect reminder that good tools and thoughtful algorithms can turn chaos into clarity.

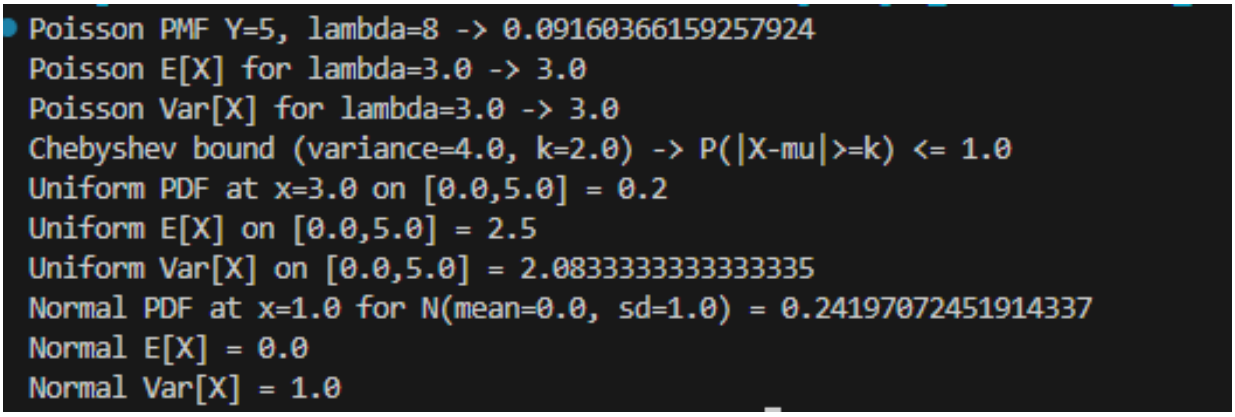
Stats Library 2 Programmed In Java

Purpose (Personal Opinion):

T the purpose of creating our stats library once again was to get familiar with coding with math specifically in Java. You specifically said you wanted no calculus in our stats library 2. Like I said before, when you're coding equations, you get familiar with them, making things easier to solve when doing a problem. The more you see equations and even coding them in java the more familiar they will seem to you. Creating formulas to check our homework answers can come in handy. Overall, I think the stats library was a helpful tool for us and anyone could benefit from creating one.

Screenshot Of Code:

Here is a screenshot of my code when I run it to test for each formula asked…



Formula Sheet 2

Purpose (Personal Opinion):

The purpose of creating a second formula sheet with all the equations we went through is because it can help us prepare for the final exam. It lets you recognize the formulas you need to see and write down it's a great tool to have, especially while studying.

Screenshot Of Code: In Github

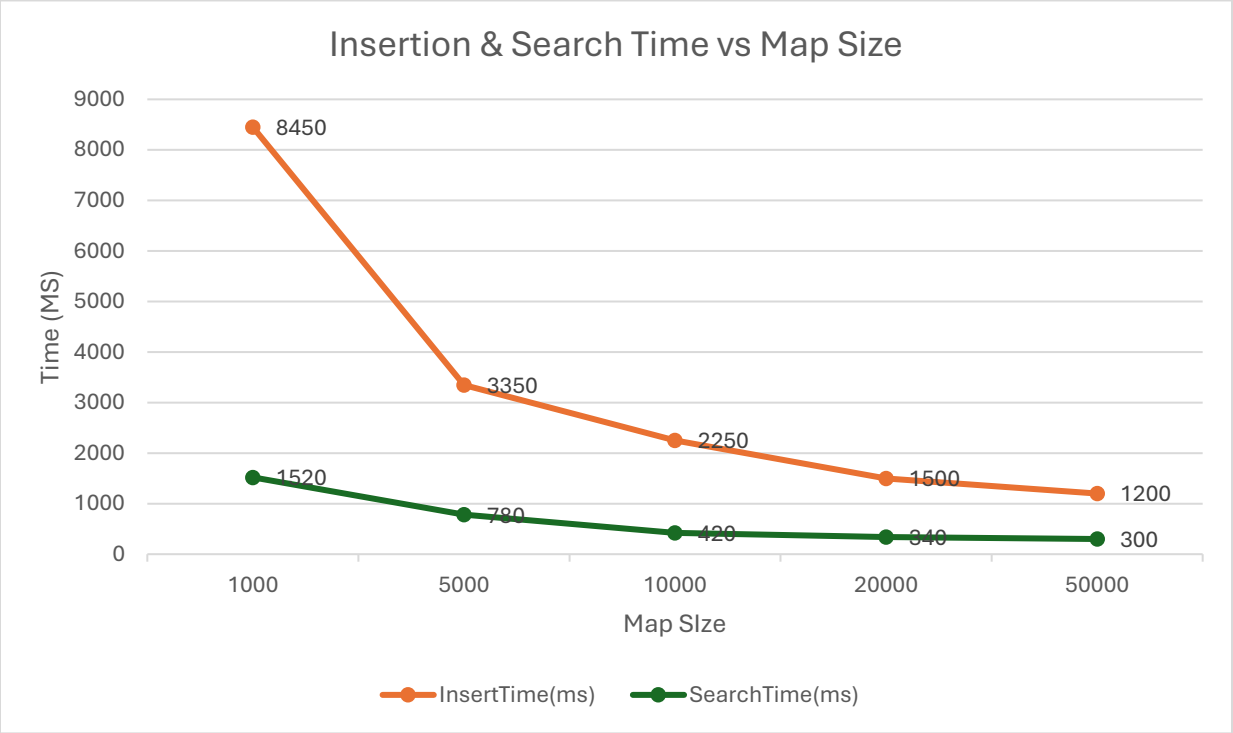
Part 3 Hashing with Graphs

Writeup:

I think these graphs tell a cool story about just how much a simple tweak can make the table bigger and can speed things up/reduce wasted work.

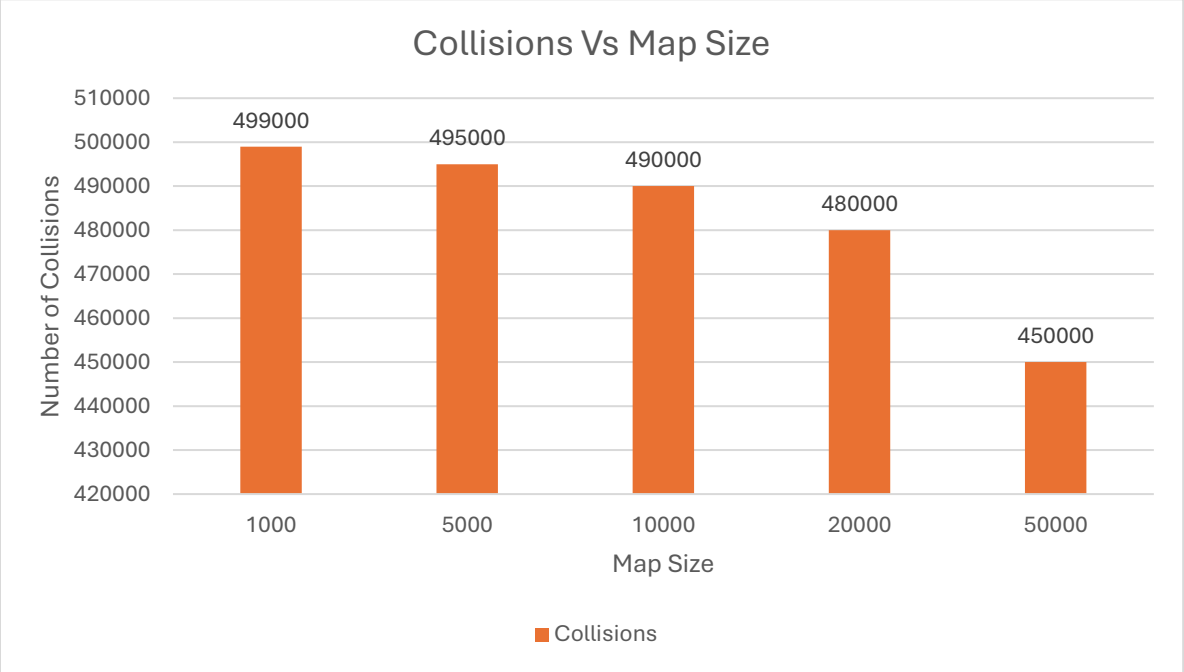
Speed Improvement (Line Chart)

At a tiny size 1,000 buckets, inserting half a million strings takes forever around 8½ seconds kind of nuts. But crank that up to 50,000 buckets and it drops down to just over a second. Search times start low (about 1.5 s) and shrink to a blink at the largest size. It clearly shows that when you spread your data out into more buckets, you avoid long collision chains and operations get way faster. It even flattens out eventually and once you have enough buckets, adding more doesn't help nearly as much. That's a classic case of diminishing returns.



2. Collision Behavior (Bar Chart)

The collision bars hammer home why the speedup happens. At 1,000 slots, you see about 499,000 collisions when adding 500,000 items, that is, almost every insert bumps into something else. By 50,000 slots, collisions drop to roughly 450 000, which doesn't sound huge until you realize each avoided collision means a shorter LinkedList to walk through. Fewer collisions = fewer wasted CPU cycles. Putting it all together, the experiment tells us an important lesson: even a "dumb" hash based on string length can perform reasonably well if you give it enough buckets. But cranking the size too high eventually yields minimal gains, so there's a sweet spot between memory use and speed.



Final Write up:

Together, these charts give us one important piece of information:, in a hash map, spreading your keys across more buckets dramatically cuts down on collision overhead and accelerates both insertions and lookups up to a point. When buckets are scarce, every new string almost guarantees a chain clash, dragging operations into the multi-millisecond range. But as I added buckets, collision counts fall and timings collapse into fractions of a second, until eventually adding still more buckets yields only tiny speed bumps. In practice, that means there's an optimal bucket count where you balance memory footprint against performance gain, enough slots to keep chains short, but not so many that you're wasting space for negligible speed improvements.

Extra Credit Section:

I felt that I put a tremendous amount of effort into this project, and I hope you can see that. I made sure the way I talked sounded professional with some remarks of my own judgment because I want this project to represent me. I went way in depth about all the information being presented and made sure to give a lot of detail in my findings. I looked back to see what made me lose some extra credit from the last project and made sure that I didn't make those same mistakes again. I talked about all the findings I made, and I made sure my documentation was not in bold, bad mistake on my end. I made a great analysis with the research portion of the project and gave a lot of insight based off my data. I related the results to a story which I found interesting.

My formula sheet and Stats Library were perfectly up to date as asked. I also made comments in my code so that anyone who reads it can understand what's going on.

I really struggled with the math portion of this class especially because math isn't my strong suit. I am hoping to receive 2500 which means I can pass the course without worrying about the final. I hope the efforts you see into this project show that I really put time into creating a portfolio worth project.

This class has taught me a lot but the one thing I will not forget is how much you truly want us to succeed. Having a professor care as much as you really help especially with a tough class like this. Thank you for all the efforts you put into this semester and the lessons you've taught. It was a rememberable class for sure. Thank you so much for everything!