Project 2

Jaiden Sincere Nunez

Probability and Applied Statistics

Professor Byron Hoy

Table of Contents

Part 1: Data Manipulation and Plotting

Introduction

This project included plotting numerical values into a visual representation in Octave or Matlab. My task is to take already listed values and plot them into visual graphs in either Octave or Matlab. The task included taking a data set of x and y values based on an existing function, for the project I have chosen $X^2 - 2x + 1$.

Breakdown

The next task following obtaining our clean x and y values is to "salt" these values. "Salting" values are when random data is fed to an already existing data set. Once the values were salted in a new program the data is sent to a new .csv file titled "salted_data.csv". Once we have our salted values, we now need to "smooth" our .csv values. A smoothing data program once created, would be fed the "salted_data.csv" to attempt to rewrite the data to look more unified and in order based on our original function. Once smoothed, a .csv file named "smoothed_data.csv" the data did not look 1:1 compared to the original. Thus once the "smoothed" data was looped back into the smoothing method six times, the data looked nearly identical compared to the original.

Once all the .csv files were now created, I used the open-source free software known as Octave due to its simplicity and ease of compatibility with my .csv files. Once my program was finished Octave was able to display all of the data from every .csv file I created in a neat and organized graph, showing the difference of all the data sets, and showing the smoothing process slowly looking more and more like the original data set.

Program Plotter:

My ProgramPlotter is a comprehensive Java program written with the sole purpose of generating a .csv data file with a list of all the x and y values of the quadratic function $y = x^2 - 2x + 1$ from the range of x = -10.0 → 10.0 to x = -50.0 → 50.0. This portion of the project helps demonstrate a comprehensive and straightforward understanding of some of the basic constructs of creating a Java program, such as using loops, data formatting, and file handling to solve the given problem.

The Java project file consists of two classes, the ProgramPlotter class, and the Tester class. The purpose of the ProgramPlotter class is to contain all of the core functionality in one class, leaving the Tester class to be its stand-alone class where the variables can be adjusted, making the code very easy for one to edit if they were looking for different ranges of data. The given and most important data types are the following:

- Double start; This double is where the range should start from for our data set (ex. -10.0).
- Double end; This double is where the range of our data set will end (ex. 10.0)
- Double step; This double is how much each step of our data we will increment by, for this project all data will be increment by one-tenth of a whole number (0.1) for every calculation.
- String fileName; This string is where the output of the .csv file will be sent to

The next portion of this code takes the following data types and using the parameter input from the Tester class, it passes the data into the generateAndExport class and creates a .csv file. Using a for loop whose endpoint can be altered from the tester class, we can make a very modular program for plotting all the points we wish to have to a .csv file.

```
public class ProgramPlotter {
    private double start; // range start
    private double end; // range end
    private double step; //increment sizes
    private String fileName; // output of the file
```

```
// constructor
public ProgramPlotter(double start, double end, double step, String fileName) {
    this.start = start;
    this.end = end;
    this.step = step;
    this.fileName = fileName;
}
```

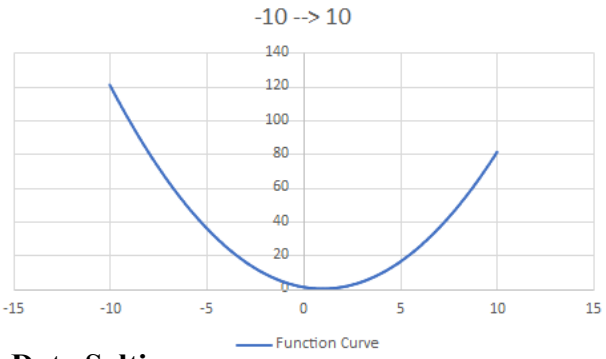
```
// calculate the function and export it to the .csv file
public void generateAndExport() {
    List<String> data = new ArrayList<>();
    DecimalFormat df = new DecimalFormat("#.###"); // format the output

    // calculate the function
    for (double x = start; x <= end; x += step) {
        double y = (x * x) - (2 * x) + 1; // y = x^2 - 2x + 1
        // format the output
        data.add(df.format(x) + "," + df.format(y));
    }

    // export to .csv
    exportToCSV(data, fileName);
    System.out.println("Data successfully exported to " + fileName + "!");
}
```

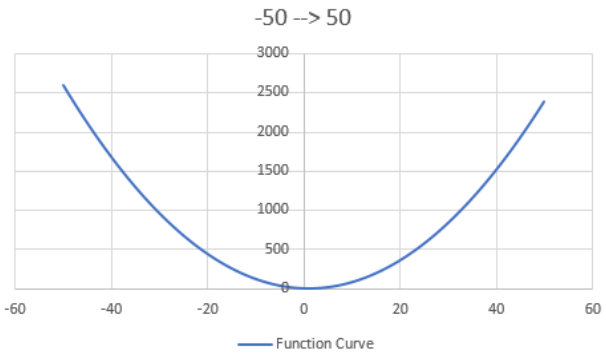
```
public class Tester {

    public static void main(String[] args) {

        ProgramPlotter plotter = new ProgramPlotter(-50.0, 50.0, 0.1, "data4.csv");
        plotter.generateAndExport();
    }
}
```

Key Functionality:

- Generate Y values after being inserted into the function $y = x^2 - 2x + 1$
- It uses a custom range of x values from start to end incrementing by 0.1.
- Exports all values to a .csv file where all y values are formatted to three decimal places
- Automatically generates an output .csv file.

Graphs:





Data Salting:

The next action in our process is to "salt" our data. The process known as 'salting" is when random data points are added to the y values in the data set. This offsets the nice curve that the function has provided to us in the first program. Salting is a great way to replicate real-world data imperfections. In the real world, data is not going to be perfect and neatly ordered, as our PlotterProgram suggested. This creates a much more unorganized variety to help us in the future with "smoothing" our data later on.

ProgramSalter reads the data from the original .csv file created from the ProgramPlotter class and adds random noise ± the range the user wishes to calculate by. The range can be altered from the double data value saltRange in the Tester class. The program preserves the x values since they are meant to never change and be stagnant, meanwhile, the y values are manipulated, and "salt" is added. After this entire process, the program writes all the modified data to a brand new file labeled salted_"original_file_name".csv.

```
// parse x and y values
String[] parts = line.split(",");
double x = Double.parseDouble(parts[0]);
double y = Double.parseDouble(parts[1]);
// adds random noise to the y values only
double salt = (Math.random() * 2 * saltRange) - saltRange;
double saltedY = y + salt;
saltedData.add(df.format(x) + ", " + df.format(saltedY));
```

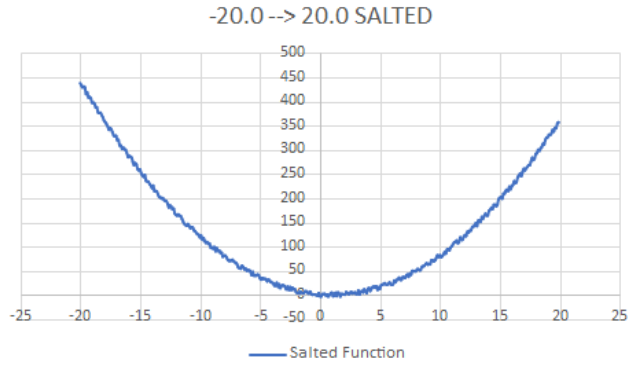
```java
public static void main(String[] args) {
    String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/Prog
    double saltRange = 5.0; // salt range

    ProgramSalter salter = new ProgramSalter(inputFileName, saltRange);
    salter.addSalt();
}
```

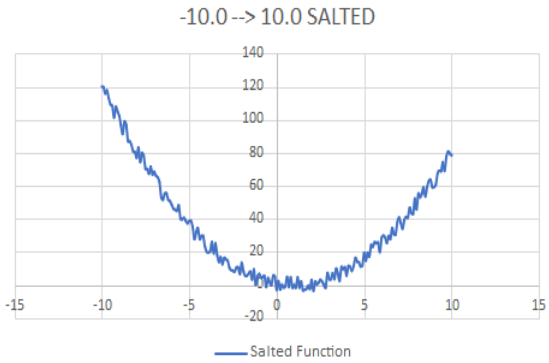
```java
public class ProgramSalter {
    private String inputFileName;
    private double saltRange; // max range of the salting

    // constructor
    public ProgramSalter(String inputFileName, double saltRange) {
        this.inputFileName = inputFileName;
        this.saltRange = saltRange;
    }
```

Key Functionalities:

- Adds "salt" to any existing data.csv file
- Salt is changeable by user input in the tester class
- Creates a semi-realistic portrayal of real-life data sets not being perfectly uniform
- Attempts to handle exceptions related to missing file issues
- Automatically generated output file names based on the input name

Graphs:





Smoothing our data:

The last Java program that needs to be created is a program to smooth. The process of smoothing data is to now reverse what was just applied to the original data set in the salting program. The new program called ProgramSmoother aims to deal with this task by using averages to attempt to move around the data slots to create a more uniform and "smooth" curve to our graphs. The program takes in the salted_data.csv file and converts it to smoothed_salted_data.csv.

ProgramSmoother reads the data from the salted .csv file created by the ProgramSalter program and applies an average smoothing algorithm to the y values and only the y values since the x values are never changing. The program also has a modular int called “smooth passes” in the Tester class that indicates how many times this algorithm will be run. After the program is run the specified amount of times, the program exports the freshly smoothed data to a brand new .csv file.

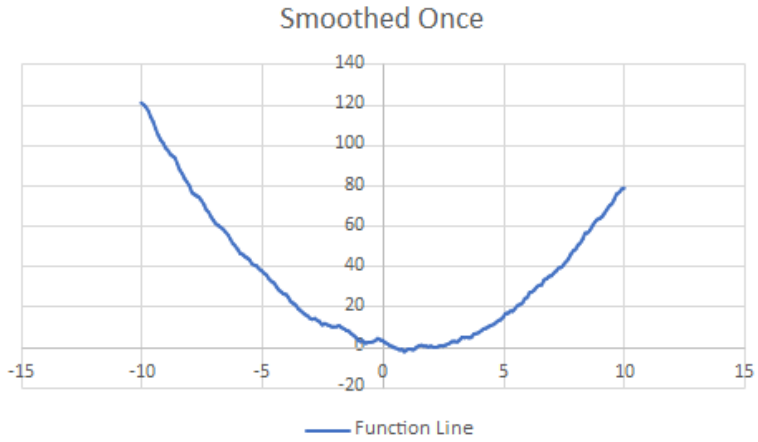
The specific data set I decided to use was the -10.0 → 10.0 data set. Once my program returned the smoothed results and I plugged the data into Excel to get a graph, The graph was noticeably more uniform compared to the salted graph. However, it was not a perfect graph. Therefore I took the smoothed .csv files and reinserted the file back into my program. I repeated this process ten times, and finally twenty times until the graph looked nearly identical to the original graph.

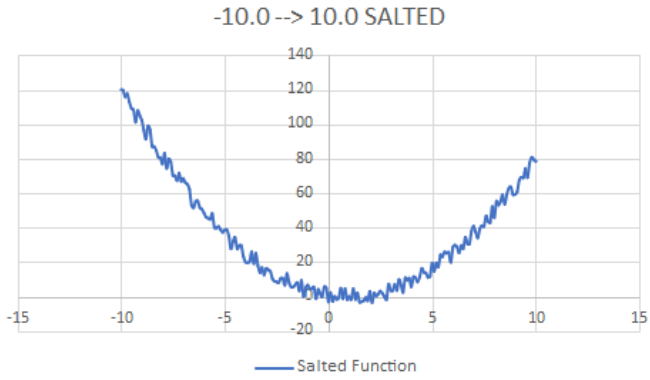
```
public class Tester {
    public static void main(String[] args) {
        String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/salted_data.csv";
        int windowValue = 3; // window size
        int smoothPasses = 1; // number of times the process gets smoothed

        ProgramSmoother smoother = new ProgramSmoother(inputFileName, windowValue, smoothPasses);
        smoother.smoothData();
    }

}
```

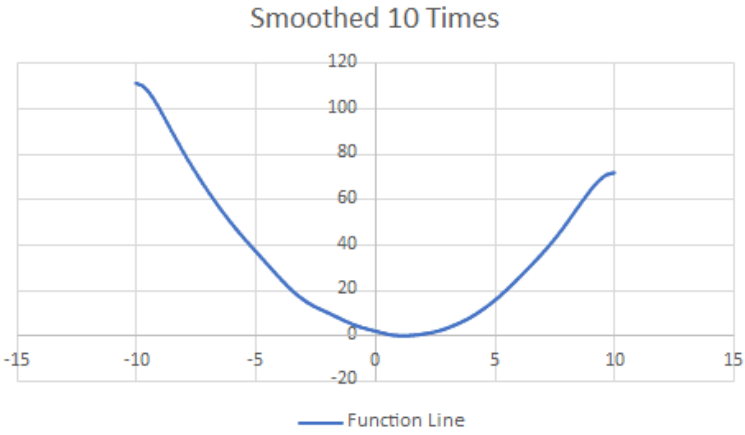
```
    // smoothes the data the amount of times the double value in the tester class is valued at
    for (int pass = 0; pass < smoothPasses; pass++) {
        yValues = smoothOnce(yValues);
    }

    // writes smoothed data to its .csv file
    try (FileWriter writer = new FileWriter(outputFileName)) {
        writer.write("x,y\n");
        DecimalFormat df = new DecimalFormat("#.###");

        for (int i = 0; i < xValues.size(); i++) {
            writer.write(df.format(xValues.get(i)) + ", " + df.format(yValues.get(i)) + "\n");
        }
        System.out.println("Smoothed data successfully exported to " + outputFileName + "!");
    } catch (IOException e) {
        System.err.println("Error writing output file: " + e.getMessage());
    }
}
```

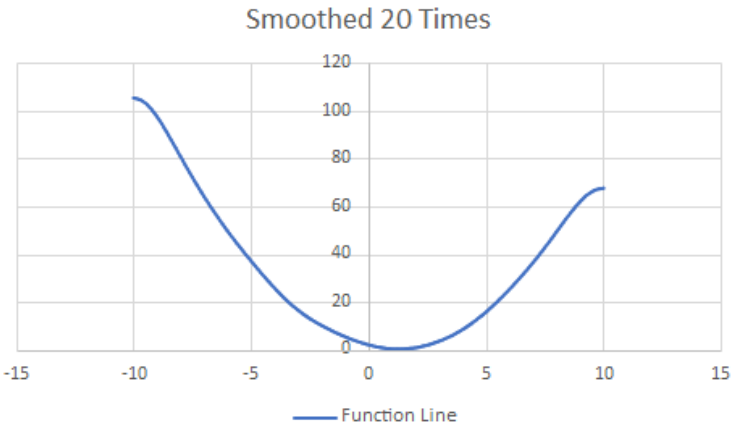
```java
// smoothes the data one time
private List<Double> smoothOnce(List<Double> yValues) {
    List<Double> smoothed = new ArrayList<>();
    int n = yValues.size();

    for (int i = 0; i < n; i++) {
        double sum = 0;
        int count = 0;
        for (int j = i - windowValue; j <= i + windowValue; j++) {
            if (j >= 0 && j < n) {
                sum += yValues.get(j);
                count++;
            }
        }
        // get average and adds it to smoothed list
        smoothed.add(sum / count);
    }

    return smoothed;
}
```

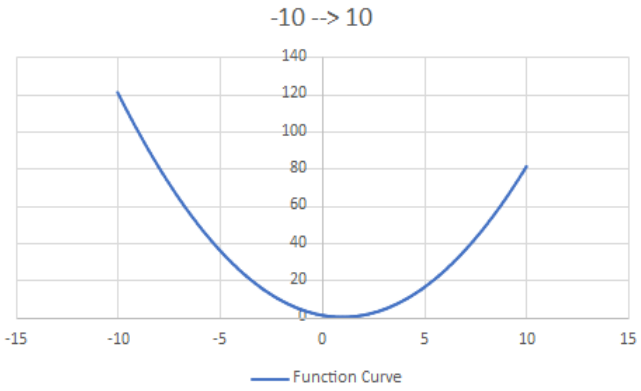
Graphs:


Smoothed Once
140
120
100
80
60
40
20
0
-20
-15
-10
-5
0
5
10
15
Function Line


-10.0 --> 10.0 SALTED
140
120
100
80
60
40
20
0
-20
-15
-10
-5
0
5
10
15
Salted Function


Smoothed 10 Times
120
100
80
60
40
20
0
-20
-15
-10
-5
0
5
10
15
Function Line


Smoothed 20 Times
120
100
80
60
40
20
0
-15
-10
-5
0
5
10
15
Function Line

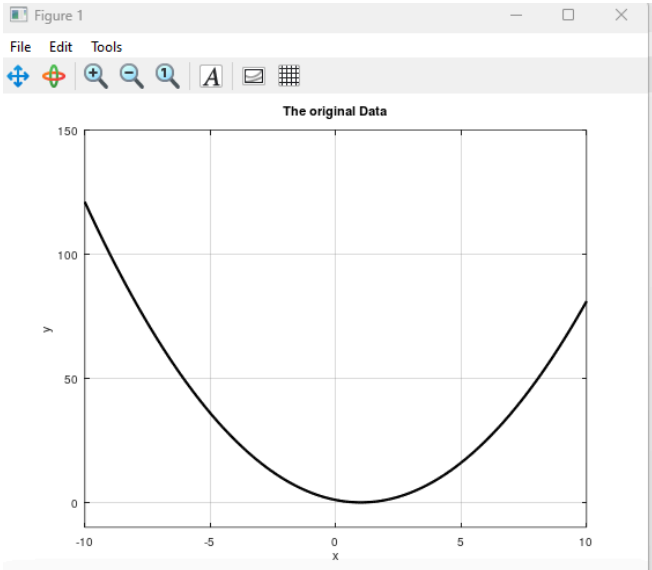

-10 --> 10
140
120
100
80
60
40
20
0
-15
-10
-5
0
5
10
15
Function Curve

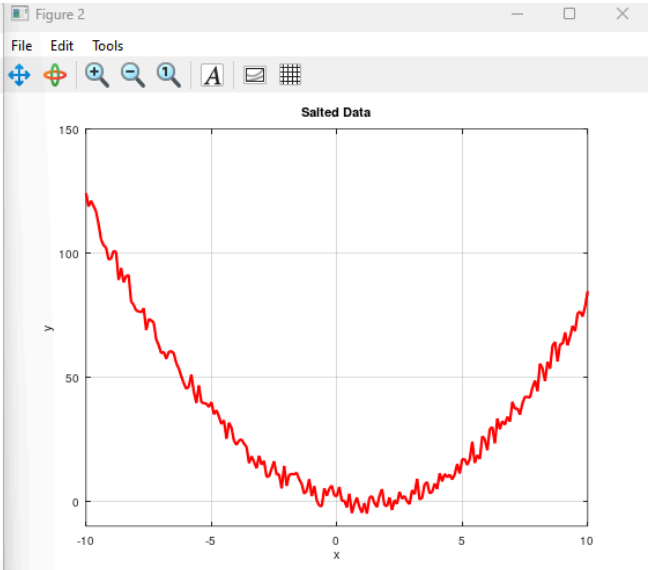
Octave Plotting:

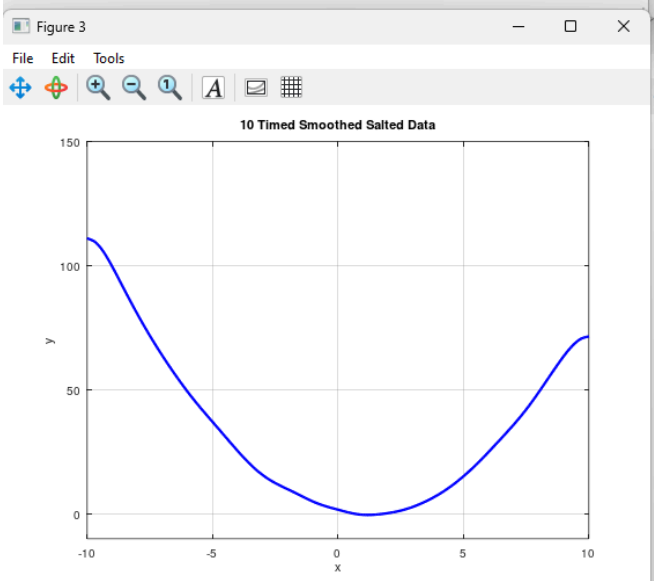
Octave is an open-source public programming language that is used primarily for numerical computations and visualizing values. Octave itself provides a solution for performing numerical experiments with large data values. Octave was originally founded in 1994 under the leadership of John W. Eaton. Octave was developed to be a free alternative to the also equally as popular MATLAB. MATLAB however costs a staggering 980 USD per year of its use. Alternatively, it can be purchased for a one-time purchase of 2450 USD. Therefore Octave proves to be a suitable and reasonable free alternative to MATLAB. The name Octave comes from the Chemical Engineer professor Octave Levenspiel who taught at the University of Oregon State.

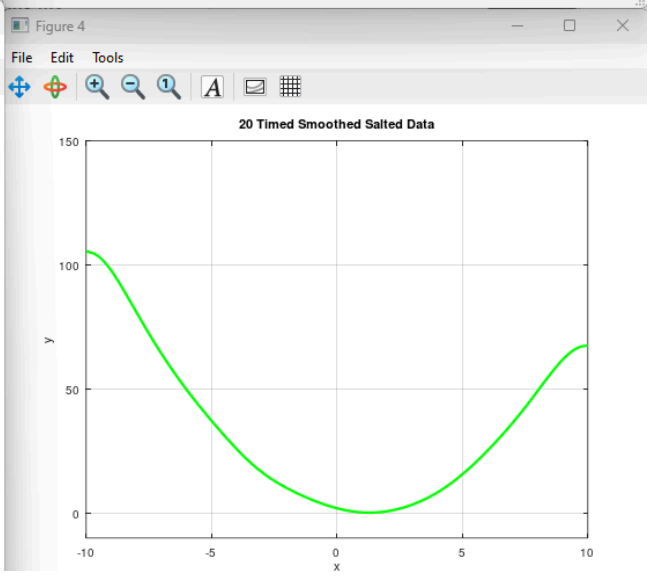
For many of the reasons above, for this project, I decided to use Octave for plotting my .csv values onto 3D graphs. My Octave script focuses on visualizing my original -10.0 → 10.0 values, my salted values, the ten-timed smoothed values, and finally, the twenty-timed smoothed values. While Java takes care of creating the values we want to analyze, we use Octave to plot and compare the values into graphs without having to import them to multiple different Excel sheets.

Compared to Java I would say Octave complements Java incredibly well. As Java produces the files Octave is incredibly efficient when it comes to visualizing the Java-made .csv files. This Octave script reads the data from the Java projects, maps out the 2D grid, and places all of the x and y values on the grid producing the graphs you have just seen. This process is copy/pasted for all 4 data values with the input adjusted to display the correct values for each of the data sets.










```
% plot the orignial csv data to a graph
data = csvread('data.csv', 1, 0); % skips top row
x = data(:, 1); % x values
y = data(:, 2); % y values
figure(1); % labels which figure youre looking at
plot(x, y, 'k-', 'LineWidth', 1.5); % the line used to display the data
title('The original Data'); % title
xlabel('x');
ylabel('y');
ylim([y_min y_max]) %limits the y values to make sure the data is even
grid on;
```

JFreeChart Integration:

JFreeChat is a free open-source Java-based charting library that is incredibly helpful to any developer to create a visual graph to directly complement their programs. In this Project, JFreeChart was an external library that once downloaded, could be implemented directly into the original three Java programs. Essentially this library allows the user to instead of importing their data into either Excel or Octave in order to get their graphs, JFreeChart allows these graphs to be displayed immediately once the new .csv files are created. This helps immensely, especially since all of these graphics are created immediately rather than plugging the data into an external source.

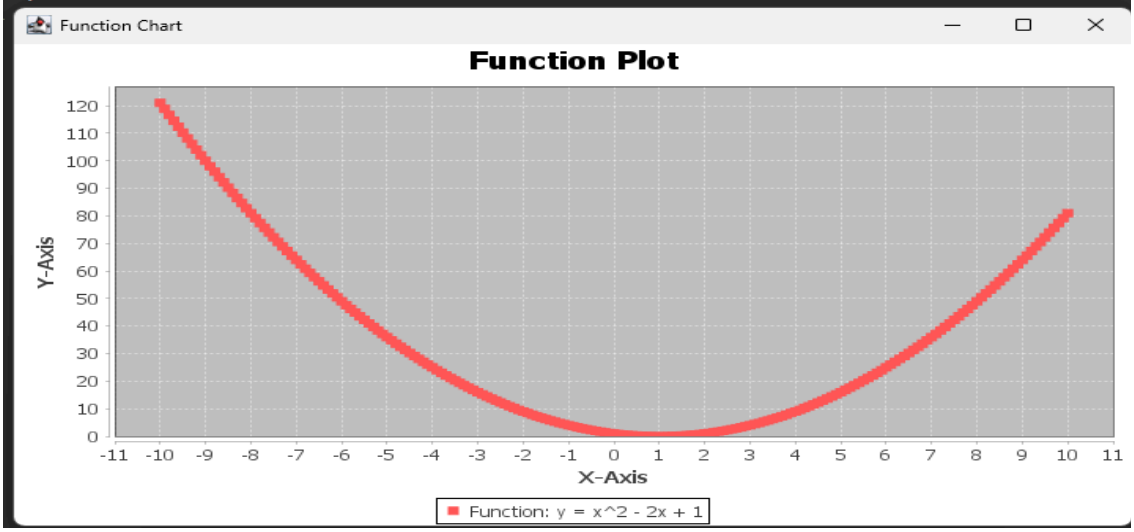
Learning how to import and use an external library was something I was not used to and took a while for me to learn. However, once I figured out how to import from an external library, it was a breeze to import JFreeChart into my already existing programs. Once the pre-existing code was implemented into my Programs, it was as simple as “plug and play”. Only a couple of lines of code were needed in order to get visual graphs of the data I was creating, and once compared to the Octave and Excel sheets, it was clear that the charts were accurate.

Some of the key differences compared to Octave are quite interesting in my opinion. For one, creating a script in Octave is not needed anymore as the data I needed can now be produced in Java with only a few clicks and much less work than Octave needed. I have noticed some key differences between the two that should be noted, however. The advantages Octave provides are that Octave is compatible with more complex data sets and can support up to a 3-dimensional space that can be altered in the window tab once the script is run. JFreeChart however, eliminates the need for external dependencies where needed and gives much faster feedback compared to any other 3rd party services.

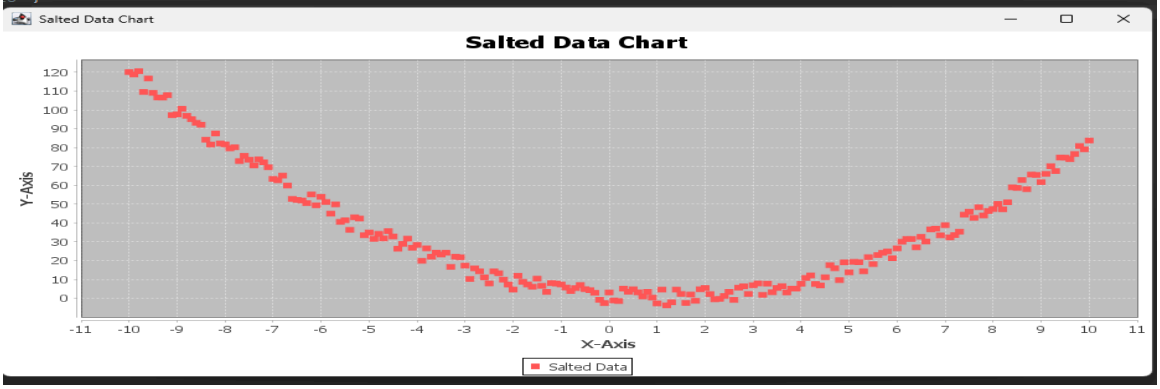
Screenshots:

```
// given code from JFreeChart to display the data in a window
public void displayChart() {
    XYSeries series = new XYSeries("Function: y = x^2 - 2x + 1");

    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
        String line;
        boolean isHeader = true;
        while ((line = reader.readLine()) != null) {
            if (isHeader) {
                isHeader = false; // Skip the header
                continue;
            }
            String[] parts = line.split(",");
            double x = Double.parseDouble(parts[0].trim());
            double y = Double.parseDouble(parts[1].trim());
            series.add(x, y);
        }
    } catch (Exception e) {
        System.err.println("Error reading data for chart: " + e.getMessage());
        return;
    }

    XYSeriesCollection dataset = new XYSeriesCollection(series);
    JFreeChart chart = ChartFactory.createScatterPlot("Function Plot","X-Axis","Y-Axis",dataset);

    JFrame frame = new JFrame("Function Chart");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.add(new ChartPanel(chart));
    frame.pack();
    frame.setVisible(true);
}
```

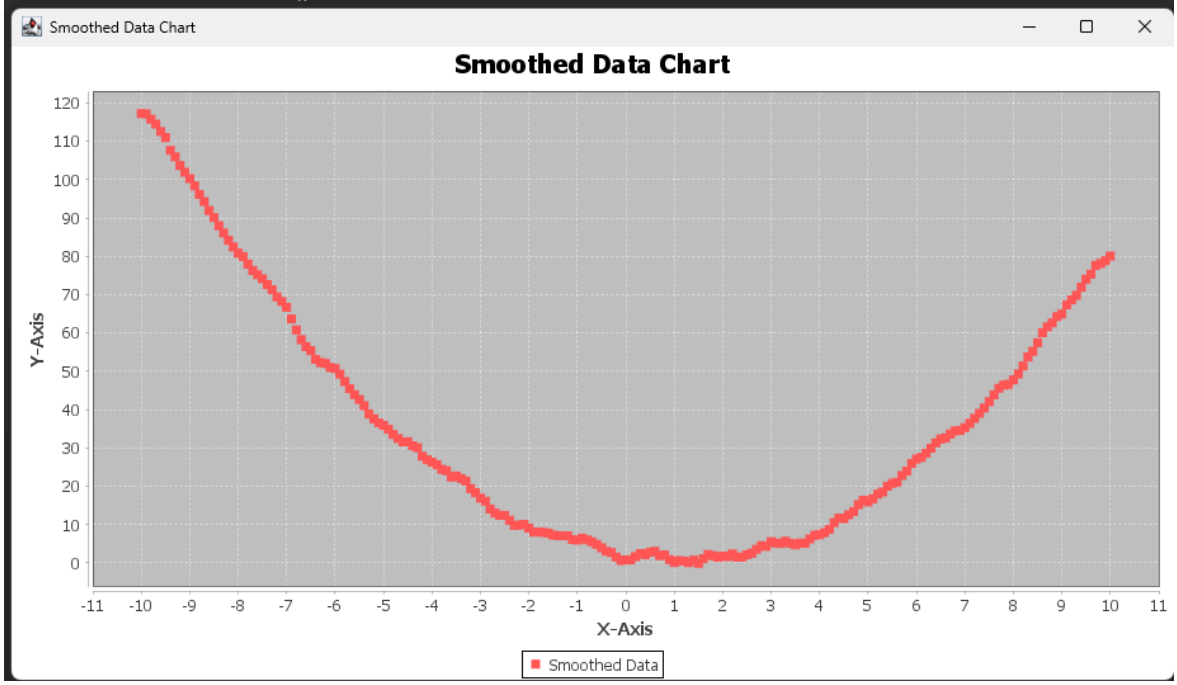
```java
public class Tester {

    public static void main(String[] args) {

        ProgramPlotter plotter = new ProgramPlotter(-10.0, 10.0, 0.1, "data4.csv");
        plotter.generateAndExport();
        plotter.displayChart();
    }
}
```




```java
public class Tester {
    public static void main(String[] args) {
        String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/data.csv";
        double saltRange = 5.0;

        // create and salt data
        ProgramSalter salter = new ProgramSalter(inputFileName, saltRange);
        salter.addSalt();

        // display the chart for the salted data
        String saltedFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/salted_data.csv";
        salter.displayChart(saltedFileName);
    }
}
```



```java
public class Tester {
    public static void main(String[] args) {
        String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/salted_data.csv";
        int windowValue = 3; // window size
        int smoothPasses = 1; // number of times the process gets smoothed

        ProgramSmoother smoother = new ProgramSmoother(inputFileName, windowValue, smoothPasses);
        smoother.smoothData();
```



Conclusion

The ProgramPlotter program served as the entire backbone of this project. It generated a clean graph that we would always come back to in order to compare any new values we obtained back to the original dataset. The ProgramSalter program added variety to our original .csv file. It simulated the real-world imperfections that are common in real-life data sets and observational data. The salted data once stored and graphed in Excel, demonstrated how randomness could distort trends in data. ProgramSmoother implemented a moving average algorithm in order to attempt to reduce the noise produced by the ProgramSalter to attempt to average out all of the data to attempt to create a more uniform data set comparable to the original dataset from the ProgramPlotter.

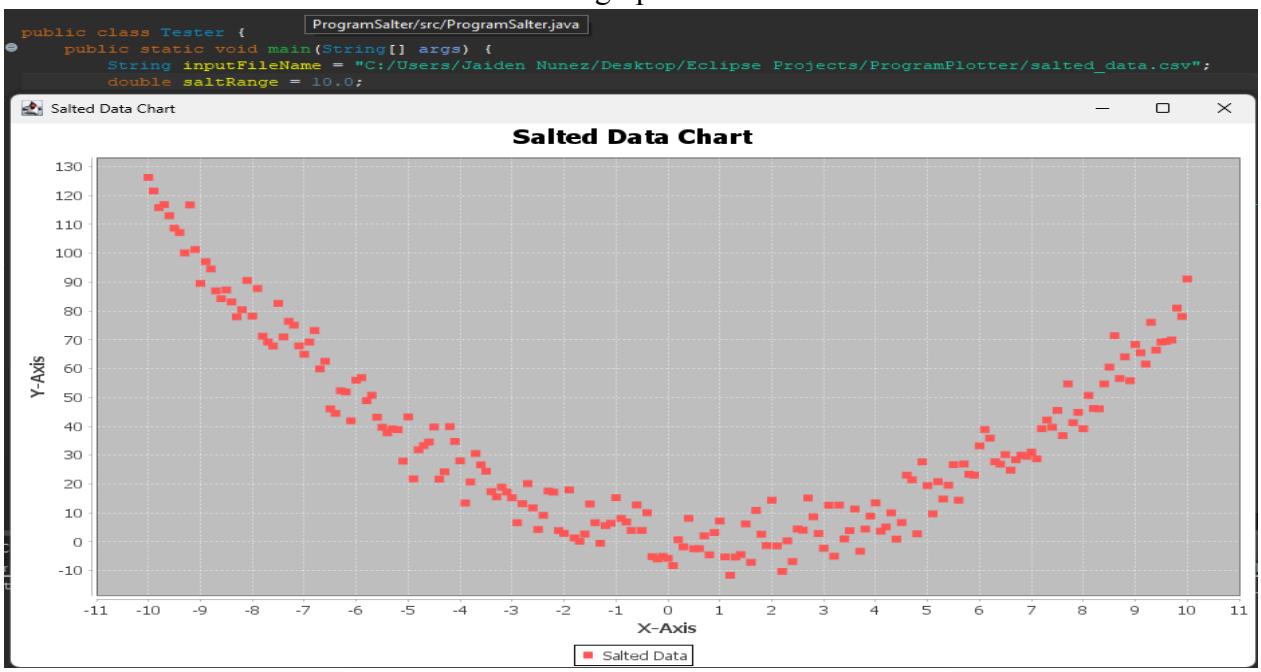
Octave provided a graphical representation of the dataset at each of the stages of the project, making comparison much easier and better. Through a clear and constant visualization, Octave complemented the Java programs by giving live visualizations of each step of the datasets. This project helped me realize that visualization is a crucial step in data analysis, which helps offer insights that numbers alone are difficult to convey comparatively.

JFreeChart however, brought an entirely new light to this project and visualized the data I was working with. The ability to import this library directly into Java programs really helps streamline the flow of the work process, preventing the need for any 3rd part sources like either Excel or Octave. The ability to get visual feedback on the data you're manipulating in real is a game-changer and really helps make the workflow feel much more linear.

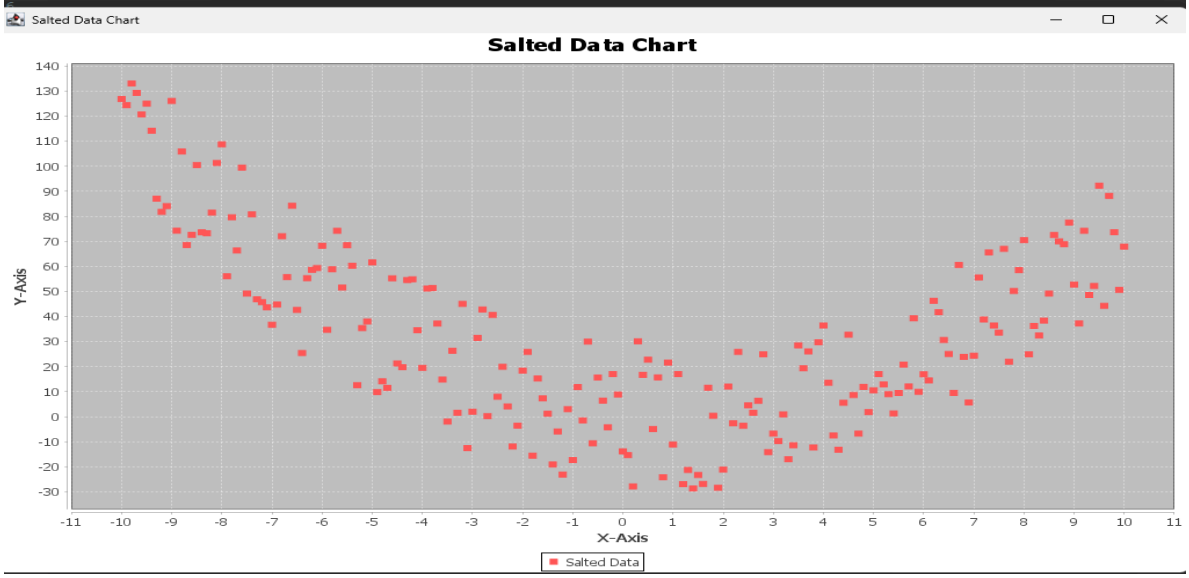
This project extends beyond any technical aspects and really helps us understand the real-world implementation of each step of the process. The importance of visualizing data analysis is not something I thought of that much when starting this project. However, seeing the changes in real-time as I progressed from my normal graph to the salted graph, and ending with the smoothed graph, offered a new insight into the realistic implications of these new strategies I learned in this project. By combining the strengths of Java with tools like JFreeChart and Octave, this project provides a really stable foundation for further exploration in data analysis and visualization.

Fun Extra Segment

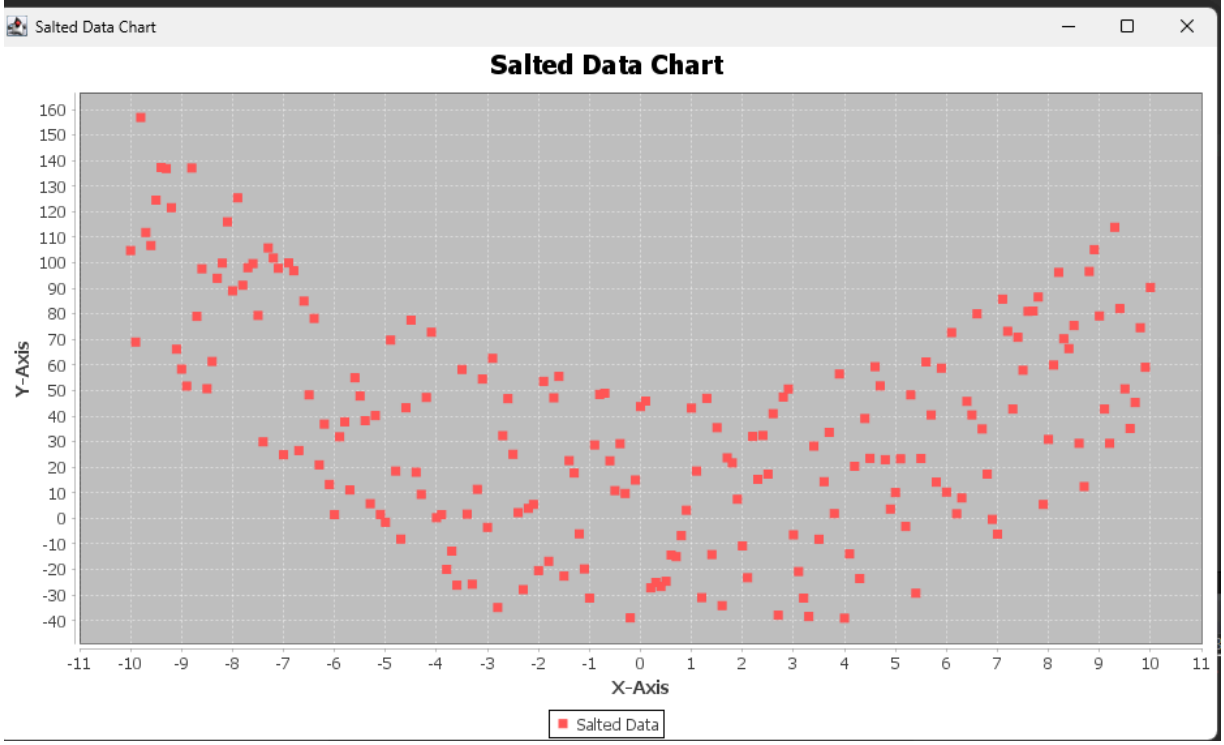
While creating part 1 of Project 2, I had a pretty fun idea that I wasn't sure where to implement, so I decided to make a new section just for my idea. So I wanted to see if I could increase the ProgramSalter range and see the outcome. Based on that outcome I wanted to see if it was possible to smooth the data after it's been severely skewered, and if it is possible to fix it. I also wondered if I could run the salted data back into the ProgramSalter to see how many runs I could do before the graph became visual clutter. Luckily, I thought of this after learning to implement JFreeChart, allowing the implementation and visual feedback to be snappy and instantly responsive. So I decided to throw my base graph from [-10.0, 0.0] into the slater and below I have attached screenshots of what the graph looked like.

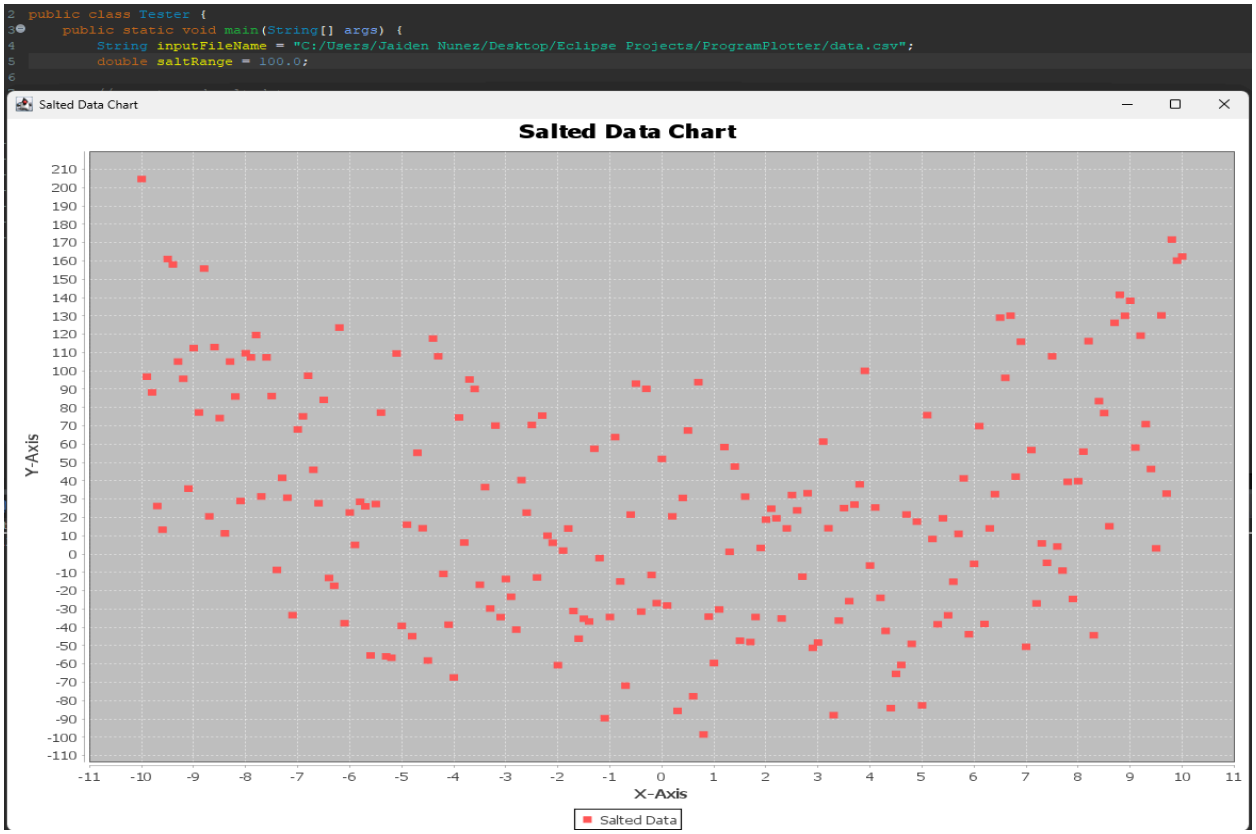


```
2 public class Tester {
3     public static void main(String[] args) {
4         String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/data.csv";
5         double saltRange = 30.0;
```



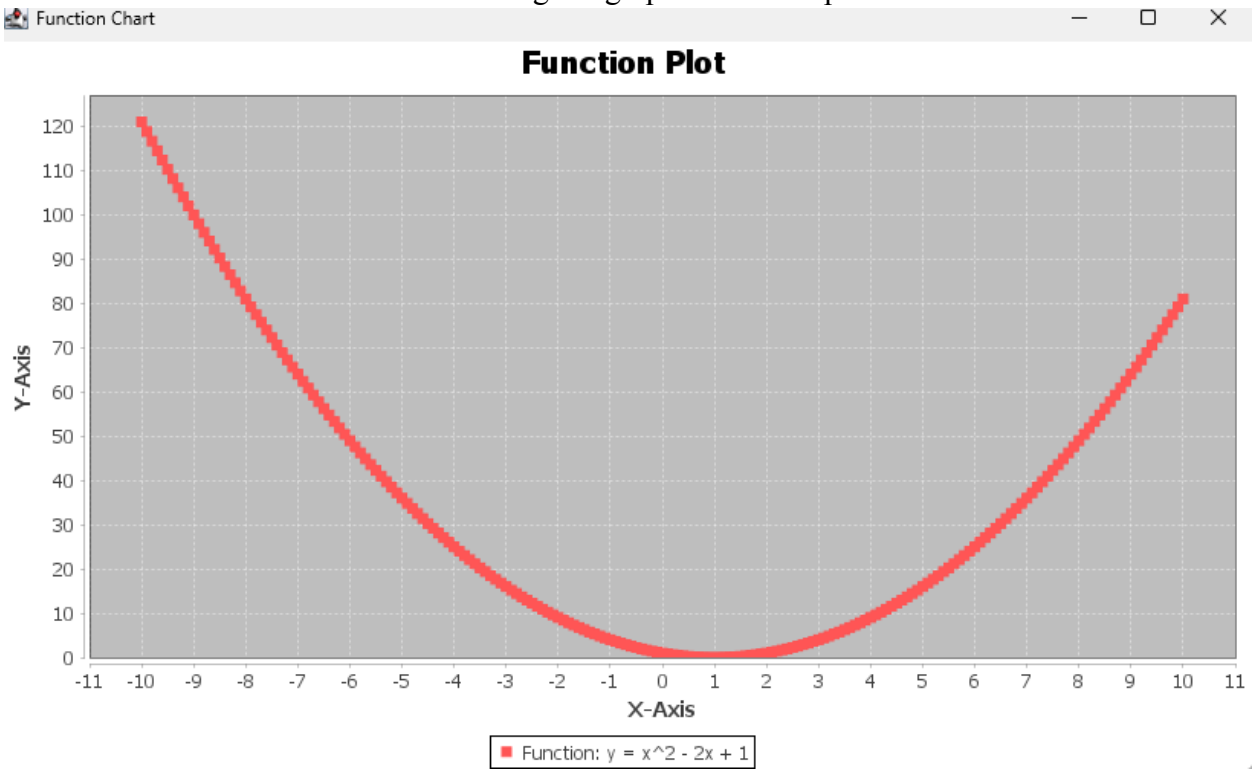

```
2 public class Tester {
3     public static void main(String[] args) {
4         String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/data.csv";
5         double saltRange = 50.0;
```



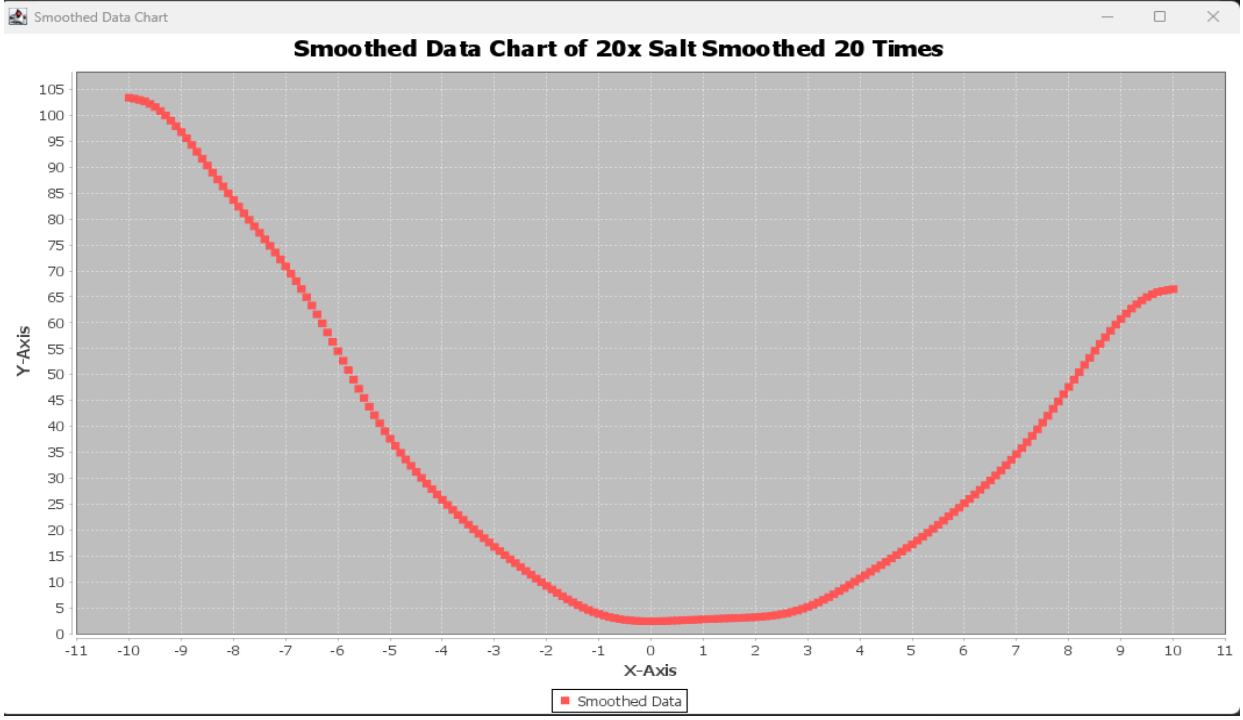


The following screenshots I provided show the data being salted in bigger and bigger intervals. The first screenshot is a range of 10, the second is a range of 20, the third is a range of 30, the fourth screenshot is a range of 50, and the final screenshot provided is a massive range of 100. As clearly demonstrated by the JFreeChart window, the scattered still somewhat represents a parabola, however, once we get to the salt range 50 and 100, it's just random dots scattered all over the place. Now for the fun part, time to see if it's somehow possible for my program to reverse the damage done by the salter using the smoother program. I will document how many passes it takes me if is even possible for the damage to be reversed. To be fair to my smoother program I will be smoothing the data as many times as I salted the data. (ex. 20x salted data will be smoothed 20 times)

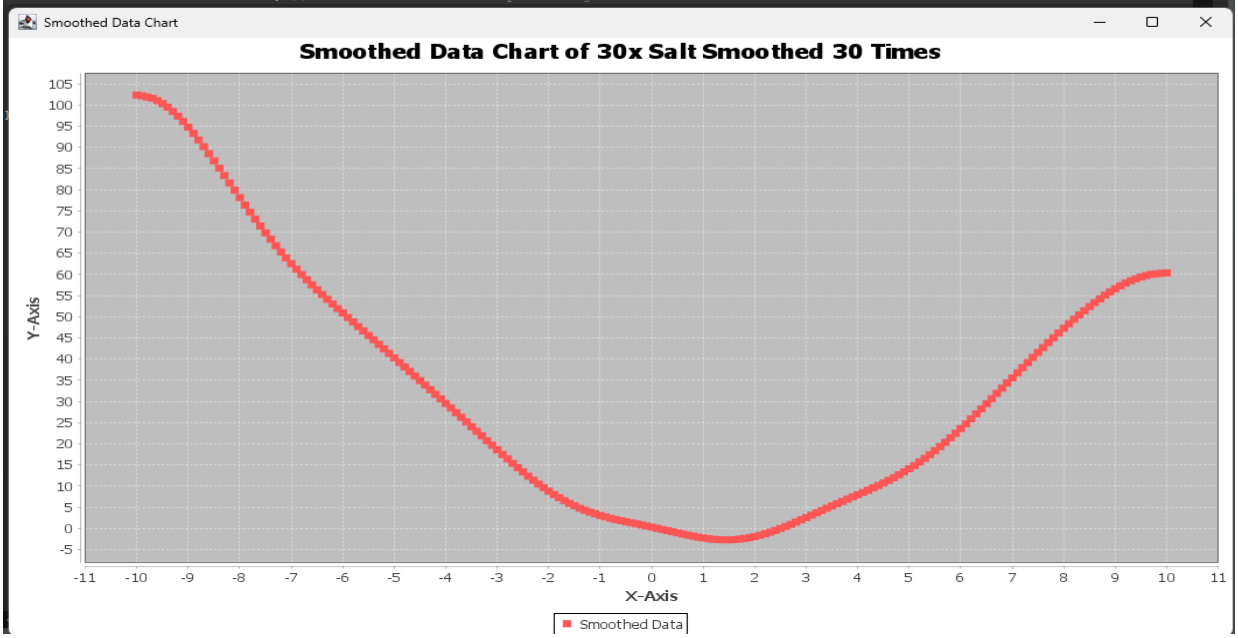
The first screenshot below will be the original graph so the comparison will be easier to make



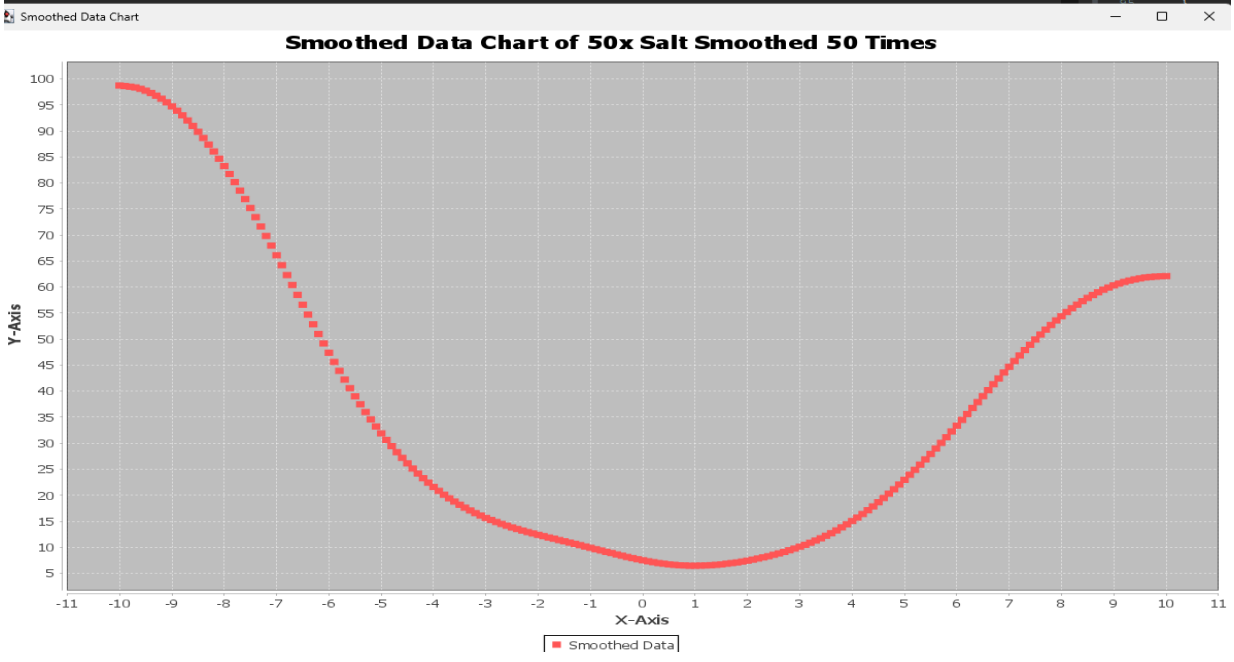

```
public class Tester {
    public static void main(String[] args) {
        String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/MultiSaltedR20_data.csv";
        int windowValue = 3; // window size
        int smoothPasses = 20; // number of times the process gets smoothed
```



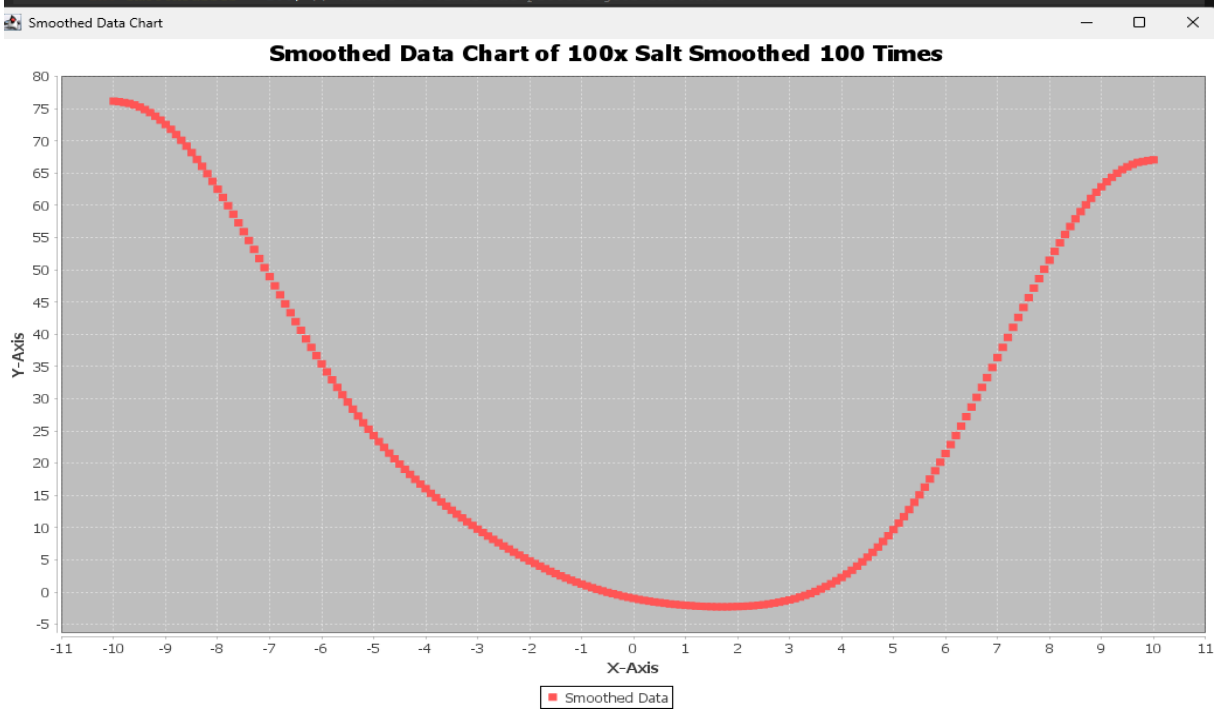
```
public class Tester {
    public static void main(String[] args) {
        String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/MultiSaltedR30_data.csv";
        int windowValue = 3; // window size
        int smoothPasses = 30; // number of times the process gets smoothed
```




```
public class Tester {
    public static void main(String[] args) {
        String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/MultiSaltedR50_data.csv";
        int windowValue = 3; // window size
        int smoothPasses = 50; // number of times the process gets smoothed

        ProgramSmoother smoother = new ProgramSmoother(inputFileName, windowValue, smoothPasses);
        smoother.smoothData();
```

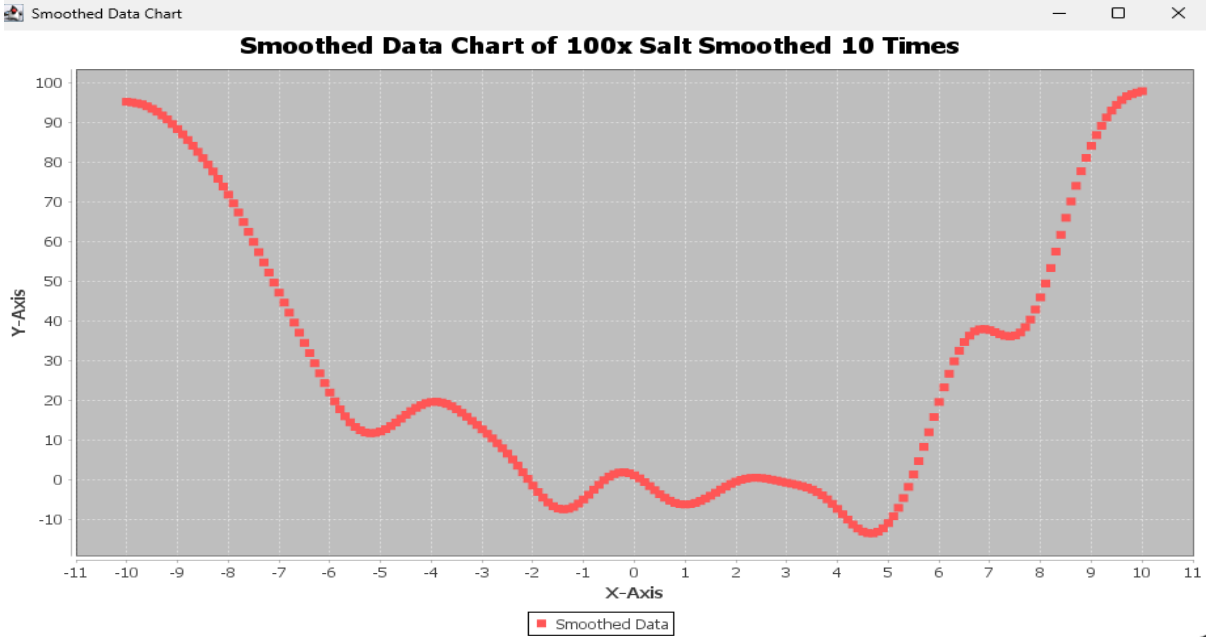


```
public class Tester {
    public static void main(String[] args) {
        String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/MultiSaltedR100_data.csv";
        int windowValue = 3; // window size
        int smoothPasses = 100; // number of times the process gets smoothed
```



Compared to 100x salted data that has only been smoothed 10 times

```
public class Tester {
    public static void main(String[] args) {
        String inputFileName = "C:/Users/Jaiden Nunez/Desktop/Eclipse Projects/ProgramPlotter/MultiSaltedR100_data
        int windowValue = 3; // window size
        int smoothPasses = 10; // number of times the process gets smoothed
```



To my utter shock, my salter was able to somewhat able to reverse the damage of the salting process. Comparing the salted data and the smoothed data, there is a surprisingly night and day difference from the two. Even the 100 times salted data getting smoothed 10 times compared to 100 times is night and day. I actually had a lot of fun in this mini section trying to test the limit of what my program could do. I can definitely say I am impressed with the outcome of my smoothers ability to smooth incredibly scewered data. Thank you for reading this mini portion of the project I wanted to include.

Part 2: Data Set Questions

For this section of my report, I will be analyzing a data set I found online for the most popular birth names in the year 2003. The list is a .csv file that is ranked from the most common names to the least common names that were documented in the United States of America. I will be analyzing the provided data set and applying a statistical conceptual grasp of not only the data but also Probability and Applied Statistics as a whole.

Mean

Using the baby name frequency data from the dataset provided, can we calculate the mean frequency for the top 20 most popular names?

Answer: The Mean Frequency is 21,641.15

Variance

From the baby name frequency dataset, calculate the variance of the frequencies for names of the top 10 most frequent names in the data set?

Answer: The variance is 8,135664.49

Standard Deviation

Determine the standard deviation of the frequencies for the top 15 baby names in the provided dataset.

Answer: The Standard Deviation is 2,972.34

Probability Axioms

Using the baby name dataset, assume the total frequency of the top 10 names in the dataset is the sample space *S*. Consider the top 10 baby names as individual events. Assign probabilities to each name (event) by dividing their frequency by the total frequency of all names in the dataset. What is the probability that a randomly chosen name belongs to one of the top 3 baby names?

Answer: The total probability of a randomly chosen name being one of the top 3 names (Jacob, Michael, or Emily) is 0.0217 or 2.17%.

Permutations:

Suppose you want to create a unique order for the top 5 baby names, how many distinct arrangements can you make?

Answer: 120 Different unique arrangements

Combinations

If you want to select 3 baby names from the top 10 without considering the order, how many different groups can you form?

Answer: 120 Different combinations

Definition 2.9 Conditional Probability

Considering the top 10 baby names in the data set, let A represent the event that a name is male, and let B represent the event that the name starts with the letter “J”. Find the probability a name in the top 10 most baby names begin with J and are male.

Answer: The chances a randomly chosen name is from the top 10, male, and begins with a J is 31.7%

Independence of Events

For the dataset of the top 10 baby names in 2003:

- Let A represent the event that a name starts with the letter "A."
- Let B represent the event that the name is female.

Determine whether the events A and B are independent.

Answer:

- $P(A) = 9.22\%$
- $P(B) = 20.15\%$
- $P(A \cap B) = 0\%$
- $P(A \cap B) \neq P(A) \bullet P(B)$

This shows that events A and B are not independent.

Bayes Rule:

- Let B_1 represent the event that a randomly chosen name is male, and B_2 represent the event the name is female.
- Let A represent the event that a name starts with the letter “J”

Answer:

- $P(B_1 \mid A)$ = The probability a name starts with the letter “J” is male in the top 10 is 100%
- $P(A \mid B_1) = 30.0\%$ The probability a name starting with “J” is male in the top 10
- $P(A \mid B_2) = 0.00\%$ The probability that a name starting with “J” is female in the top 10

Probability Distribution for Y:

Consider Y, a random variable representing the number of names starting with each letter of the alphabet among the top 10 baby names in 2003. Construct the probability distribution for Y by calculating the proportion of names starting with each letter in the dataset.

Answer:

- $P(Y = "J") = 31.7\%$
- $P(Y = "E") = 29.0\%$
- $P(Y = "M") = 21.3\%$

- $P(Y = "A") = 9.2\%$
- $P(Y = "D") = 8.8\%$
- Total = 100.00%

Binomial Distribution

In 2003, the name "Emily" occurred 25,692 times out of a total of 296,334 occurrences for the top 20 names. Assume the probability of selecting "Emily" at random is $p = \frac{25692}{296334}$ If 10 names are selected randomly with replacement, let Y represent the number of times "Emily" is chosen.

1. What is the probability that "Emily" is chosen exactly 4 times?
2. What is the probability that "Emily" is chosen at least 2 times?

Answer:

- The probability Emily is chosen at least 4 times is 0.69%
- The probability Emily is chosen 21.29%

Geometric Probability Distribution:

Suppose the probability of encountering the name "Emily" in the dataset is $P = \frac{25692}{296334}$. If trials are conducted to find the first occurrence of the name "Emily," and the trials are independent:

1. What is the probability that the first occurrence of "Emily" happens on the 5th trial?
2. Calculate the expected number of trials needed to find the first occurrence of "Emily".
3. Find the variance in the number of trials needed to find the first occurrence of "Emily".

Answer:

1. $P(y) = q^{y-1}p = P(5) = (0.9133)^4 \bullet 0.0867 = 0.0605 = 6.05\%$
2. $E(Y) = \frac{1}{p} = \frac{1}{0.0867} = 11.53$ Expected number of trials
3. $V(Y) = \frac{0.9133}{(0.0867)^2} = \frac{0.9133}{0.00753} = 121.46$ For Variance

Negative Binomial Probability:

Suppose you are searching for the name "Emily" in the dataset. Assume the probability of encountering "Emily" is $p = \frac{25692}{296334}$, and the trials are independent. Let Y be the trial on which the 3rd occurrence of "Emily" happens.

1. What is the probability that the 10th trial is the one where the 3rd occurrence of Emily happens?
2. Find the mean and variance of Y, the trial on which the 3rd occurrence of Emily.

Answer:

1. $P(10) = \binom{9}{2}(0.0867)^2(0.9133)^7 = 12.43\%$
2. **Expected Value:** $E(Y) = \frac{r}{p} = \frac{3}{0.0867} = 34.6$
   Variance: $V(Y) = \frac{r(1-p)}{p^2} = \frac{3 \bullet 0.9133}{(0.086)2} = 362.56$

Hypergeometric Probability Distribution:

From the top 20 names listed in the dataset, there are 8 female names and 12 male names. If a random sample of 5 names is selected without replacement, let Y denote the number of female names in the sample.

1. What is the probability that exactly 2 of the selected names are female?
2. Calculate the expected value and variance of Y.

Answer:

1. $P(y) = \frac{\binom{y}{n}\binom{N-r}{n-y}}{\binom{N}{n}}$

- N = 20 (total names)
- r = 8 (total female names)
- n = 5 (sample size)
- y = 2 (number of female names in the sample)

$$P(y) = \frac{\binom{8}{2}\binom{12}{3}}{\binom{20}{5}} = 0.397 = 39.7\%$$

2. **Expected:** $E(Y) = n\frac{r}{N} = 5 \bullet \frac{8}{20} = 2$

   Variance: $V(Y) = n\frac{r}{N}\frac{N-r}{N}\frac{N-n}{N-1} = 0.947$

Poisson Probability Distribution:

From the dataset, assume that the top 20 names occur in a Poisson manner with a mean occurrence rate (λ) of 5 names per sample interval.

1. What is the probability that exactly 3 names occur in an interval?
2. Calculate the expected value and variance of the number of names occurring in a sample interval.

Answer:

$\lambda = 3$

$y = 3$

1. $p(y) = \frac{\lambda^y e^{-\lambda}}{y!} = \frac{5^3 e^{-5}}{3!} = 0.1404$
2. Expected value and Varience = 5

Definition Probability Density Function:

The frequency of names in the dataset is modeled as a continuous random variable Y with a density function

$$f(y) = cy, \quad 0 \leq y \leq 2$$

Where c is a constant.

- Find the value of c such that f(y) is a valid probability density function
- Compute $F(y)$, the distribution function
- Calculate $P(1 \geq Y \geq 2$ using $F(y)$

Answer:

1. **Find c**

$$\int_0^2 f(y)dy = 1$$

$$\int_0^2 c \bullet ydy$$

$$2c = 1 ==> c = \frac{1}{2}$$

2. **Compute** $F(y)$

Substitute c = $\frac{1}{2}$

$$F(y) = \int_0^y f(t)\, dt$$

$$F(y) = \frac{1}{2}\int_0^y t\,dt$$

$$F(y) = \frac{1}{2} \bullet \frac{y^2}{2} = \frac{y^2}{4}$$

3. **Compute** $P(1 \leq Y \leq 2)$

$$P(1 \leq Y \leq 2) = F(2) - F(1)$$

Substitute $F(y) = \frac{y^2}{4}$

$$P(1 \leq Y \leq 2) = 0.75$$

Probability Density Function Expected Value and Variance:

The number of occurrences of names in the top 100 names from the dataset is modeled as a continuous random variable Y with a probability density function

$$f(y) = \frac{1}{2}(2 - y), \qquad 0 \leq y \leq 2$$

1. Find the expected number of occurrences $E(Y)$ for a randomly selected name.
2. Calculate the variance $V(Y)$ of the number of occurrences.

Answer:

1. **Expected Value**

$$E(Y) = \int_0^2 y \bullet \frac{1}{2}(2 - y)\,dy$$

$$E(Y) = \frac{2}{3}$$

2. **Variance**

$$V(Y) = E(Y^2) - [E(Y)]^2$$

$$E(Y^2) = \int_0^2 y^2 \bullet \frac{1}{2}(2 - y)\,dy$$

$$E(Y^2) = \frac{2}{3}$$

$$V(Y) = \frac{2}{3} - \frac{4}{9} = \frac{2}{9}$$

Uniform Distribution:

a) Imagine in the US in 2003 theoretically, a born would be born in the first 15 seconds of a one-minute period. What is the probability a baby would be born not in the first 15 seconds of a one minute period?
b) Find Mean and Variance

Answer:

a) A = 0, B = 60, C = 15 D = 60 $\rightarrow \frac{d-c}{b-a} = \frac{60-15}{60} = \frac{45}{60} = 75\%$

b) **Mean:** $\frac{a+b}{2} = \frac{60+0}{2} = \frac{75}{2} = 30$ seconds

Variance: $\frac{(60-0)^2}{12} = 300 \text{ seconds}^2$

Exponential Distribution:

The frequency of a given name in the provided dataset is modeled as an exponential distribution with a mean of 2.4 occurrences per 1000 births. Find the probability that

1. The frequency of a name exceeds 3.0 occurrences per 1000 births.
2. Find the Mean and Variance

Answer:

1. $f(y) = \frac{1}{\beta} e^{-\frac{y}{\beta}} \quad y \geq 0$ Where $\beta = 2.4$ y = frequency

   $P(Y > 3.0) = \int_{\infty}^{3} \frac{1}{2.4} e^{-\frac{y}{2.4}} dy$

   $f(\infty) - f(3) = 0.286$

2. **Mean** = $\beta = 2.4$ **Variance** = $\beta^2 = 5.76$

Joint Probability Function

A hospital tracks the choice of two expectant parents selecting names for their child from three distinct categories: popular, traditional, and unique. Each parent independently selects a category for their baby's name. Let $Y1$ denote the number of parents who choose the "popular" category, and $Y2$ denote the number who select the "traditional" category.

- Find the joint probability function of $Y1$ and $Y2$

Answer:

Step 1: Set up sample space

S={(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)}.

Step 2: Set up a table

	y_1		
y_2	0	1	2
0	1/9	2/9	1/9
1	2/9	2/9	0
2	1/9	0	0

$P(Y1 = 1, Y2 = 1) = \frac{2}{9}$ 22%

$P(Y1 = 1, Y2 = 1) = \frac{1}{9}$ 11%

Joint Probability Density Function:

Suppose two names are randomly selected from the top 5 most popular names for 2003 in the United States. Let $Y1$ denote the rank of the first name selected (where rank 1 is the most popular), and $Y2$ denote the rank of the second name selected. Assume the selections are independent and that the probability density function of $Y1$ and $Y2$ is given by:

$f(y1, y2) = \{\frac{1}{25}, 1 \leq y1, y2 \leq 5, \quad elsewhere \quad 0$.

a. **Find** $P(Y_1 + Y_2 \leq 6)$
b. **Find** $P(Y_1 \leq Y_2)$

Answer:

A) $P(Y_1 + Y_2 \leq 6)$

$$\int_1^5 \int_1^{6-y1} f(y_1 y_2) dy_2 dy_1$$

$$\int_{1}^{5}\int_{1}^{6-y1}\frac{1}{25}dy_2dy_1$$

$$\frac{1}{25}\int_{1}^{5}(5-y_1)dy_1$$

$$\frac{1}{25}\bullet 8 = 0.32$$

B) $P(Y_1 \leq Y_2)$

$$P(Y_1 \leq Y_2) = \frac{1}{25} \bullet 8 = 0.32$$

Part 3: Porting Pokemon Project to smartphone

For the final part of Project 2, we were tasked to port our Pokemon game project that was created in Project 1 to our smartphones, where they can be locally stored, edited, and run all from our mobile device. This process, however, was a bit tricky for me and my device. The smartphone I have is an iPhone 13 Pro. Famously, the creator of the iPhone Apple Inc. is well known for not allowing any third-party applications to be sideloaded from any outside device onto their devices. This restriction although seemed very restrictive at first, turned out to actually not be too big of a deal when it came to porting the project to my phone.

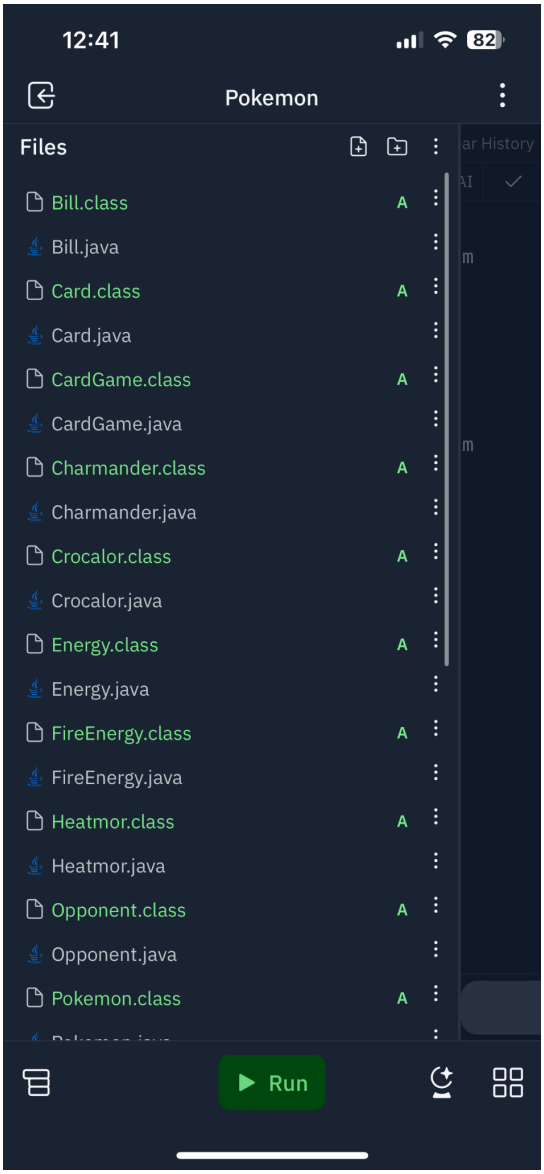
The method I used to get my Pokemon project ported to my phone was to download an IDE onto my phone from the Apple app store. At first, I started downloading a bunch of random apps on the app store that promised to be able to “write code in multiple languages”. However, the majority of these applications not only didn't work but also had a myriad of negative reviews backing up these claims that these apps now working. However, I ended up stumbling upon a certain app called Replit Replit is a cloud-based IDE that allows users to code many different languages Although the review it had on the Apple app store wasn't the most promising, it turned out to be my saving grace in this section of the project.

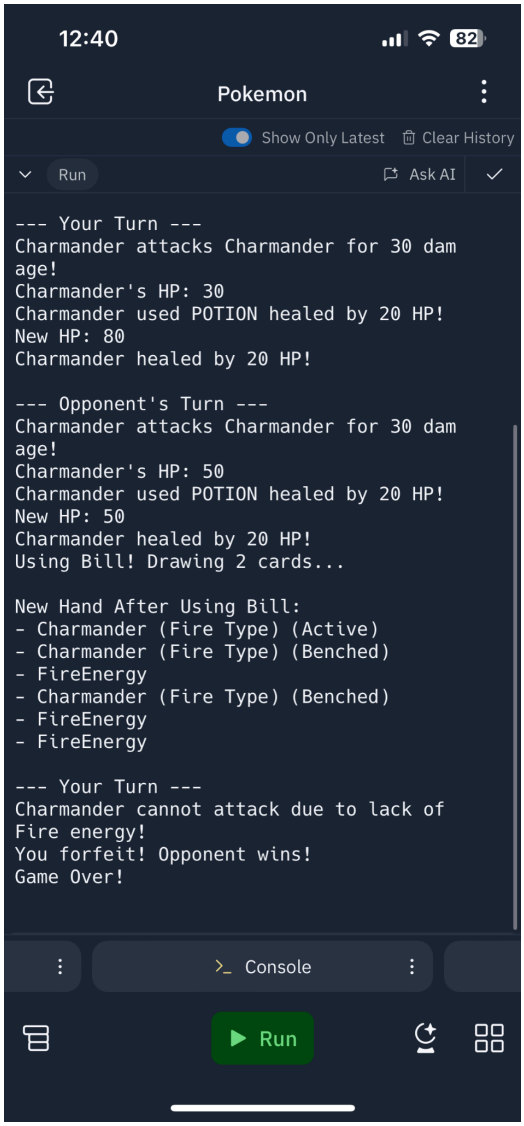
Replit was the answer to my problems all condensed into one simple-to-use application. The way I got the Pokemon project onto my phone was a bit tricky at first, but all I had to do was upload the project source code to my Google Drive. I uploaded the zip file of the Pokemon project to my local drive from the computer I originally coded the project on, allowing it to be accessible from my phone. Now that the project can be downloaded onto my iPhone using Google Drive app, I was once step closer to getting to my goal. Once the project was in my phone's files, The app Repelit allowed the user to upload preexisting source code, so I was able to upload the source code to the compiler and after all of that was done, I did run into a few bugs. The only real issue however was I needed to open each class for a few seconds in order for the compiler to load, and after that it was good to go. The code ran and the console output the results of the Pokemon game.

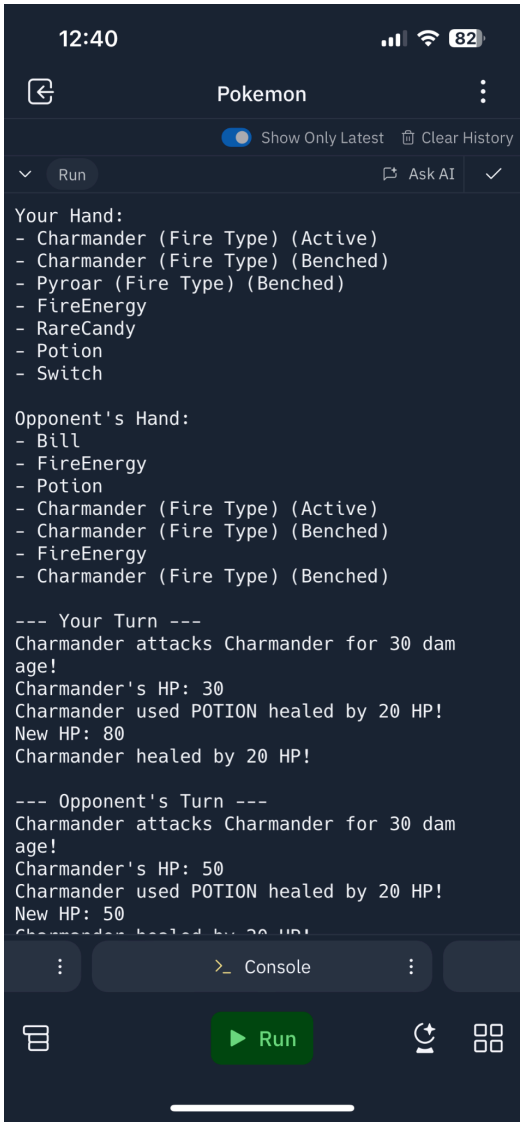
Reflecting on this experience its taught me a few things. I learned that trying to get something as simple as a Java code to run on an iPhone is not as easy as it first seemed. However, through trial and error, I was eventually able to succeed in my task. The restrictive environment of Apple OS helped me learn to think outside the box and find a different external solution to my problem. Gaining familiarity with Replits ability to store and execute code from their app was also a fun learning moment as well.

In conclusion, while porting the Pokemon project to my iPhone, several unexpected challenges arose. However, with some determination, I was able to overcome these challenges and ported the project to my phone. This process helped teach me the value of using every resource around me to work around a difficult problem. Not only was I able to complete the process and obtain my goal of porting the project, but I also gained a firmer understanding and much more appreciation for cloud-based IDE’s.


12:41
82
Pokemon
Files
Heatmor.java
Opponent.class
A
Opponent.java
Pokemon.class
A
Pokemon.java
Potion.class
A
Potion.java
Pyroar.class
A
Pyroar.java
Rapidash.class
A
Rapidash.java
RareCandy.class
A
RareCandy.java
Switch.class
A
Switch.java
Tester.class
A
Tester.java
Trainer.class
A
Trainer.java
Run


12:41
82
Pokemon
Files
Bill.class
A
Bill.java
Card.class
A
Card.java
CardGame.class
A
CardGame.java
Charmander.class
A
Charmander.java
Crocalor.class
A
Crocalor.java
Energy.class
A
Energy.java
FireEnergy.class
A
FireEnergy.java
Heatmor.class
A
Heatmor.java
Opponent.class
A
Opponent.java
Pokemon.class
A
Run


12:40
82
Pokemon
Show Only Latest
Clear History
Run
Ask AI
--- Your Turn ---
Charmander attacks Charmander for 30 dam
age!
Charmander's HP: 30
Charmander used POTION healed by 20 HP!
New HP: 80
Charmander healed by 20 HP!
--- Opponent's Turn ---
Charmander attacks Charmander for 30 dam
age!
Charmander's HP: 50
Charmander used POTION healed by 20 HP!
New HP: 50
Charmander healed by 20 HP!
Using Bill! Drawing 2 cards...
New Hand After Using Bill:
- Charmander (Fire Type) (Active)
- Charmander (Fire Type) (Benched)
- FireEnergy
- Charmander (Fire Type) (Benched)
- FireEnergy
- FireEnergy
--- Your Turn ---
Charmander cannot attack due to lack of
Fire energy!
You forfeit! Opponent wins!
Game Over!
Console
Run


12:40
82
Pokemon
Show Only Latest
Clear History
Run
Ask AI
Your Hand:
- Charmander (Fire Type) (Active)
- Charmander (Fire Type) (Benched)
- Pyroar (Fire Type) (Benched)
- FireEnergy
- RareCandy
- Potion
- Switch
Opponent's Hand:
- Bill
- FireEnergy
- Potion
- Charmander (Fire Type) (Active)
- Charmander (Fire Type) (Benched)
- FireEnergy
- Charmander (Fire Type) (Benched)
--- Your Turn ---
Charmander attacks Charmander for 30 dam
age!
Charmander's HP: 30
Charmander used POTION healed by 20 HP!
New HP: 80
Charmander healed by 20 HP!
--- Opponent's Turn ---
Charmander attacks Charmander for 30 dam
age!
Charmander's HP: 50
Charmander used POTION healed by 20 HP!
New HP: 50
Console
Run

Misc. Formula Sheet

Geometric Probability

Definition 3.8 $P(y) = q^{y-1}p$

- Where p = probability of success
- Where q = probability of failure
- Where y = number of trials

Expected Value (Mean):

$$E(Y) = \frac{1}{p}$$

Variance:

$$V(Y) = \frac{1-p}{p^2}$$

When to use this formula?

- Binary outcomes: every trial results in either success or failure.
- Independent Trials: the outcome of one trial doesn't affect the others.
- Constant Probability: The probability of success p remains the same for all trials.

Example: You flip a coin until you get heads. What is the probability it happens on the 3rd flip?

P = $\frac{1}{2}$; Q = $\frac{5}{6}$; Y = 3

Negative Binomial Probability Distribution

Definition 3.9: $P(y) = \binom{y-1}{r-1} p^r q^{y-r}$

- Where p = probability of success
- Where q = probability of failure
- Where r = number of wanted successes
- Where y = number of trials

Expected Value (Mean) :

$$E(Y) = \frac{r}{p}$$

Variance:

$$V(Y) = \frac{r(1-p)}{p^2},$$

When to use this formula:

- When the number of successes is stated (r)
- Binary outcomes: every trial results in either success or failure.
- Independent trials: the outcome of one trial doesn't affect the others.
- Constant probability: the probability of success p remains the same for all trials.

Example: When rolling a die, what is the probability that you'll need 8 rolls to get 3 sixes?

P = $\frac{1}{6}$; Q = $\frac{5}{6}$; R = 3; Y = 8

Hypergeometric Probability Distribution

Definition 3.10 $P(y) = \frac{\binom{r}{y}\binom{N-r}{n-y}}{\binom{N}{n}}$

- Where N = population size
- Where r = The number total needed (ex. Total number of blue marbles)
- Where n = the sample size
- Where y = number of successes in the sample

Expected Value:

$E(Y) = n\frac{r}{N}$

Variance:

$V(Y) = n\frac{r}{N} \bullet \frac{N-r}{N} \bullet \frac{N-n}{N-1}$

When to use this formula:

- When a sample is used without replacing any elements (order does not matter, and elements are not returned to the population)
- When the total population size, the number of successes in the population, and the sample size are all known.
- A finite population (N),
- Fixed number of successes

Example: A population of 20 balls contains 8 red balls. If you randomly select 5 balls without replacement, what is the probability that exactly 3 of them are red?

N = 20; r = 8; n = 5; y = 3

Poisson Probability Function

Definition 3.11 $P(y) = \frac{\lambda^y \bullet e^{-\lambda}}{y!}$

- Where e = 2.718
- Where y = number of occurrences
- Where λ is the average value

Expected Value:

$E(Y) = \lambda$

Variance

$V(Y) = \lambda$

When to use this formula:

- When modeling the number of occurrences of an invention in a fixed interval or space (ex, “the number of cars passing through a toll booth in 1 hour”
- Events occur independently: One occurrence does not affect another.
- Constant rate: The average number of occurrences (λ) is constant over time or space.
- The random variable Y represents the number of occurrences in the given interval

Example: A store receives an average of 4 customers per hour. What is the probability that exactly 6 customers will arrive in a given hour?

$\lambda = 4$; Y = 6

Uniform Distribution

Definition 4.6 $P(y) = \frac{d-c}{b-a}$

- Where a = the upper bound
- Where b = the lower bound
- Where c = the starting point of the interval
- Where d = the ending point of the interval

Expected Value:

$E(Y) = \frac{a+b}{2}$

Variance:

$V(Y) = \frac{(b-a)^2}{12}$

When to use this formula:

- All outcomes are within an equally likely range (ex. [1,10])
- The distribution is continuous and probabilities are described in intervals and not discrete points
- Conditions: The variable is uniformly **BETWEEN** a and b, and the probability density function is constant between the interval (c and d)

Example: Suppose a random variable Y is uniformly distributed between 2 and 8. What is the mean, variance, and probability of Y falling between 4 and 6?

A = 2; B = 8; C = 4; D = 6

Exponential Distribution

Definition: $f(y) = \{ \frac{1}{\beta} e^{-\frac{y}{\beta}}, \; if \; y \geq 0$

- Where β is the Mean
- Where y is the random variable

Expected Value:

$E(Y) = \beta$

Variance:

$V(Y) = \beta^2$

When to use this formula:

- When modeling the time between independent events that occur at a constant rate (ex. Time between arrivals of customers at a store)
- Events occur independently
- The probability of an event occurring in the future does not depend on how much time has already passed. The chance of something is always constant

Example: Suppose the mean time between customer arrivals at a bank is 5 minutes. What is the probability that the next customer arrives within 3 minutes?

$\beta = 5 \; (mean \; time \; between \; arivals)$

$y = 3 \; (time \; until \; next \; arrival)$

Joint Probability Function

Definition $p(y_1, y_2) = P(Y_1 = y_1, Y_2 = y_2), \quad -\infty < y_1 < \infty, \ -\infty < y_2 < \infty$

- Where Y_1 *and* Y_2 are Discrete Random Variables
- Where y_1 *and* y_2 are specific values taken by the random variables Y_1 *and* Y_2
- $p(y_1, y_2)$ is the joint probability that Y_1 takes the value of y_1 and Y_2 takes the value of y_2

When to use this formula:

- When dealing with two or more discrete random variables and want to calculate the probability of their outcomes (ex. The probability that one die shows a 3 ($Y_1 = 3$) and another die shows a 5 ($Y_2 = 5$)

Example: A pair of fair dice is rolled. What is the joint probability that the first die shows a 2 ($Y_1 = 2$) and the second die shows a 5 ($Y_2 = 5$)?

- Notice how both outcomes are independent. If each die has 6 sides then the total number of possible outcomes is 36 (6x6=36)
- $p(y_1 = 2, y_2 = 5) = \frac{1}{36}$ is the probability
- .

Joint Distribution of Continuous Random Variables

Definition: $F(y_1, y_2) = \int_{-\infty}^{y_1} \int_{-\infty}^{y_2} f(t_1, t_2)\, dt_2 dt_1$

- Where $f(y_1, y_2) \geq 0$ for all y_1, y_2
- $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(y_1, y_2)\, dy_1 dy_2 = 1$

Best to replace y_1 *and* y_2 *as* x *and* y

When to use this formula:

- When dealing with two continuous random variables that are defined or an interval (that COULD be infinite).
- Joint probabilities: probability both variables fall within a specific range
- Marginal Probabilities: integrate out one of the variables either x or y
- Calculate over a range (ex. y_1 *and* y_2 represent the height and weight of individuals. y_1 could represent a height between 60 inches and 72 inches (5 and 6 feet) and $y_2$represents a weight between 100 and 200 pounds.

Expected Value (Mean):

$E(Y) = \int_{-\infty}^{\infty} y f(y)\, dy$

Variance:

$V(Y) = E(Y^2) - [E(Y)]^2$

Example: Let $F(y_1, y_2) = y_1 y_2$ *for* $y_1, \quad y_2 \geq 0, \quad$ *Find* $P(Y_1 \leq 1, Y_2 \leq 2)$

Setup $\rightarrow F(1,2) = \int_0^1 \int_0^2 y_1 y_2 \, dy_2 dy_1$

Marginal Probabilities

Definition $f_1(y_1) = \int_{-\infty}^{\infty} f(y_1, y_2) \, dy_2, \quad f_2(y_2) = \int_{-\infty}^{\infty} f(y_1, y_2) \, dy_1$

Used for Finding the marginal densities from a joint density in continuous cases. Best to replace $y_1 \, and \, y_2 \, as \, x \, and \, y$

- Where $f(y_1, y_2)$: Joint Probability Distribution Function of $Y_1 \, and \, Y_2$
- $f_1(y_1)$: Marginal Probability Distribution Function of Y_1
- $f_2(y_2)$: Marginal Probability Distribution Function of Y_2
- Calculate the Integral sum

Suppose the joint Probability Density function is

$f(y_1, y_2) \{ 2, \; 0 \leq y_1 \leq 1, \quad 0 \leq y_2 \leq 1 \qquad 0, \; otherwise$

Setup for $Y_1 \rightarrow f_1(y_1) = \int_0^1 f(y_1, y_2) \, dy_2 = \int_0^1 2 dy_2 = 2$

Setup for $Y_2 \rightarrow f_2(y_2) = \int_0^1 f(y_1, y_2) \, dy_1 = \int_0^1 2 dy_1 = 2$

Answer: For $0 \leq y_2 \leq 1, \; f_2(y_2) = 2$

For $0 \leq y_1 \leq 1, \; f_1(y_1) = 2$

Otherwise = 0

Geometric Probability

```java
public class GeometricProbability {

    private double p; // the probability of success
    private int y;    // number of trials

    // constructor
    public GeometricProbability(double p, int y) {
        this.p = p;
        this.y = y;
    }

    // the method to calculate p(y)
    public double calculateProbability() {
        double q = 1 - p; // probability of failure
        return Math.pow(q, y - 1) * p;
    }

    // calculate expected value
    public double calculateExpectedValue() {
        return 1 / p;
    }

    // calclate Variance
    public double calculateVariance() {
        return (1 - p) / (p * p);
    }

    // display calculations
    public void displayResults() {
        System.out.println("Geometric Probability Distribution Results:");
        System.out.println("P(y): " + calculateProbability());
        System.out.println("Expected Value: " + calculateExpectedValue());
        System.out.println("Variance: " + calculateVariance());
    }

}
```

```java
public class Tester {

    public static void main(String[] args) {

        new GeometricProbability(0.5, 2).displayResults(); // prints Geometric Probability

    }

}
```

```
Console
<terminated> Tester (15) [Java Application] C:\Users\Jaiden Nunez\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230425
Geometric Probability Distribution Results:
P(y): 0.25
Expected Value: 2.0
Variance: 2.0
```

Negative Binomial Probability

```java
public class NegativeBinomialProbability {
    private double p; // probability of success
    private int r;    // number of successes wanted
    private int y;    // number of trials

    // constructor
    public NegativeBinomialProbability(double p, int r, int y) {
        this.p = p;
        this.r = r;
        this.y = y;
    }

    // method to calculate factorial
    private long factorial(int num) {
        long result = 1;
        for (int i = 2; i <= num; i++) {
            result *= i;
        }
        return result;
    }

    // calculate binomial coefficient
    private long binomialCoefficient(int n, int k) {
        return factorial(n) / (factorial(k) * factorial(n - k));
    }

    // method used to calculate p(y)
    public double calculateProbability() {
        double q = 1 - p; // Probability of failure
        long coefficient = binomialCoefficient(y - 1, r - 1);
        return coefficient * Math.pow(p, r) * Math.pow(q, y - r);
    }

    // calculate expected value
    public double calculateExpectedValue() {
        return (double) r / p;
    }

    // calculate Variance
    public double calculateVariance() {
        return r * (1 - p) / (p * p);
    }

    // display results
    public void displayResults() {
        System.out.println("Negative Binomial Probability Distribution Results:");
        System.out.println("P(y): " + calculateProbability());
        System.out.println("Expected Value (E[Y]): " + calculateExpectedValue());
        System.out.println("Variance (Var[Y]): " + calculateVariance());
    }

}
```

```java
public class Tester {

    public static void main(String[] args) {

        //new GeometricProbability(0.5, 2).displayResults(); // prints
        new NegativeBinomialProbability(0.4, 3, 6).displayResults();
```

```
Console
<terminated> Tester (15) [Java Application] C:\Users\Jaiden Nunez\.p2\pool\plugins\org.eclipse.justj.openjdk.hot
Negative Binomial Probability Distribution Results:
P(y): 0.13824
Expected Value (E[Y]): 7.5
Variance (Var[Y]): 11.249999999999996
```

Hypergeometric Probability Distribution

```java
public class HypergeometricProbability {

    private int N; // the population size
    private int r; // number of needed successes
    private int n; // the sample size
    private int y; // the number of successes in the sample

    // constructor
    public HypergeometricProbability(int N, int r, int n, int y) {
        this.N = N;
        this.r = r;
        this.n = n;
        this.y = y;
    }

    // factorial calculator
    private long factorial(int num) {
        long result = 1;
        for (int i = 2; i <= num; i++) {
            result *= i;
        }
        return result;
    }

    // calculate binomial coefficient
    private long binomialCoefficient(int n, int k) {
        return factorial(n) / (factorial(k) * factorial(n - k));
    }

    // calculate p(y)
    public double calculateProbability() {
        long numerator = binomialCoefficient(r, y) * binomialCoefficient(N - r, n - y);
        long denominator = binomialCoefficient(N, n);
        return (double) numerator / denominator;
    }

    // expected vale calculator
    public double calculateExpectedValue() {
        return n * ((double) r / N);
    }

    // variance calculator
    public double calculateVariance() {
        double term1 = n * ((double) r / N);
        double term2 = (N - r) / (double) N;
        double term3 = (N - n) / (double) (N - 1);
        return term1 * term2 * term3;
    }

    // display results
    public void displayResults() {
        System.out.println("Hypergeometric Probability Distribution Results:");
        System.out.println("P(y): " + calculateProbability());
        System.out.println("Expected Value (E[Y]): " + calculateExpectedValue());
        System.out.println("Variance (Var[Y]): " + calculateVariance());
    }

}
```

```java
        new HypergeometricProbability(50, 20, 10, 5).displayResults();

    }

}
```

```
Console
<terminated> Tester (15) [Java Application] C:\Users\Jaiden Nunez\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot
Hypergeometric Probability Distribution Results:
P(y): 7752.0
Expected Value (E[Y]): 4.0
Variance (Var[Y]): 1.9591836734693877
```

Poisson Probability

```java
public class PoissonProbability {

    private double lambda; // mean number of events. aka Half life symbol
    private int y;         // number of occurrences

    // constructor
    public PoissonProbability(double lambda, int y) {
        this.lambda = lambda;
        this.y = y;
    }

    // factorial calculator
    private long factorial(int num) {
        long result = 1;
        for (int i = 2; i <= num; i++) {
            result *= i;
        }
        return result;
    }

    // calculation of p(y)
    public double calculateProbability() {
        return (Math.pow(lambda, y) * Math.exp(-lambda)) / factorial(y);
    }

    // expected value
    public double calculateExpectedValue() {
        return lambda;
    }

    // variance
    public double calculateVariance() {
        return lambda;
    }

    // Method to display all results
    public void displayResults() {
        System.out.println("Poisson Probability Distribution Results:");
        System.out.println("P(y): " + calculateProbability());
        System.out.println("Expected Value (E[Y]): " + calculateExpectedValue());
        System.out.println("Variance (Var[Y]): " + calculateVariance());
    }

}
```

```java
public class Tester {

    public static void main(String[] args) {

        //new GeometricProbability(0.5, 2).displayResults(); // prints Geometric Probab
        //new NegativeBinomialProbability(0.4, 3, 6).displayResults(); //prints Negativ
        //new HypergeometricProbability(50, 20, 10, 5).displayResults();
        new PoissonProbability(3.5, 4).displayResults();
    }

}
```

```
Console
<terminated> Tester (15) [Java Application] C:\Users\Jaiden Nunez\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17
Poisson Probability Distribution Results:
P(y): 0.18881228540881959
Expected Value (E[Y]): 3.5
Variance (Var[Y]): 3.5
```

Uniform Distribution

```java
public class UniformDistribution {

    private double a; // lower bound
    private double b; // upper bound
    private double c; // starting point of interval
    private double d; // ending point of interval

    // constructor
    public UniformDistribution(double a, double b, double c, double d) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.d = d;
    }

    // calculate p(y)
    public double calculateProbability() {
        if (c >= a && d <= b && a < b) {
            return (d - c) / (b - a);
        } else {
            throw new IllegalArgumentException("Invalid bounds or intervals!");
        }
    }

    // calculate the expected value
    public double calculateExpectedValue() {
        return (a + b) / 2;
    }

    // calculate Variance
    public double calculateVariance() {
        return Math.pow(b - a, 2) / 12;
    }

    // display results
    public void displayResults() {
        System.out.println("Uniform Distribution Results:");
        System.out.println("P(y): " + calculateProbability());
        System.out.println("Expected Value (E[Y]): " + calculateExpectedValue());
        System.out.println("Variance (Var[Y]): " + calculateVariance());
    }

}
```

```java
public class Tester {

    public static void main(String[] args) {

        //new GeometricProbability(0.5, 2).displayResults(); // prints Geometric Probability
        //new NegativeBinomialProbability(0.4, 3, 6).displayResults(); //prints Negative Binomi
        //new HypergeometricProbability(50, 20, 10, 5).displayResults();
        //new PoissonProbability(3.5, 4).displayResults();
        new UniformDistribution(0, 10, 3, 7).displayResults();
    }

}
```

```
Console
<terminated> Tester (15) [Java Application] C:\Users\Jaiden Nunez\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230
Uniform Distribution Results:
P(y): 0.4
Expected Value (E[Y]): 5.0
Variance (Var[Y]): 8.333333333333334
```