Juan Villaman

05/02/2024

**Using JFreeChart to Visualize our PSS**

**Use of Apache Stats Library**

# What is JFreeChart

JFreeChart is free Java library that allows us to visualize data within our programs ([Source](https://www.jfree.org/jfreechart/)). JFreeChart has extensive usability when it comes to handling data, and how to visualize it. Its clear documentation allowed me to navigate the library to successfully visualize my **Plotter**, **Salter**, and **Smoother** (**PSS**) programs. The purpose for using JFreeChart is having the program handle the graphing for us, instead of having to manually export the CSVs to Excel and insert our own graph.

## Extra Credit: Using Maven

Researching JFreeChart, trying to access the latest version of the library always brought me to a [GitHub repository.](https://github.com/jfree/jfreechart/releases/tag/v1.5.2) I could not access the .jar file needed to use the library in my own projects. However, making a Maven Java project would allow me to access the library through a **dependency**.

Why Maven? Maven makes the process of downloading dependencies which refer to the JAR file we are trying to use. With Maven, I was able to easily access the JFreeChart library by adding its dependency to the pom.xml file.

What is the POM.xml file? The Project Object Model file is an extremely important file in a Maven project that handles the configuration of our project, its dependencies, and project information needed to run our Java program ([Source](https://www.browserstack.com/guide/what-is-pom-in-maven#:~:text=for%20in%20Maven-,pom.,%2C%20goals%2C%20and%20build%20lifecycle.)). Adding a dependency is simple. Open the POM.xml file and navigate to the dependency section of the file. To add a dependency, simply copy and paste the dependency code. In JFreeChart’s case, it looks like this:

A screen shot of a computer

AI-generated content may be incorrect.

As the green comment line states, I got this dependency using mvnrepository, a website that has a vast library of dependencies for easy access. After adding the dependency to our POM.xml file, we are ready to use JFreeChart in our Java program. All that’s left to do is import it. These are the imports I used from the library to handle our PSS.

A black background with white text

AI-generated content may be incorrect.

Maven makes it simple to manage dependencies as they are all in one file making it easy to see exactly which ones you are using. VSCode also helps us set the Maven Java project correctly, as all necessary files (POM.xml) are generated in order to start the project. Mvnrepository shows us the infinite possibilities of the libraries we can use, and it’s just a simple copy-and-pasty into your POM.xml file to start using said libraries. Using Maven also prevents having to download individual JAR files and add them the referenced libraries within a regular Java project. The dependency within the POM.xml file retrieves the necessary JAR files to make the library work within our Java project.

### Processing Our CSV Files

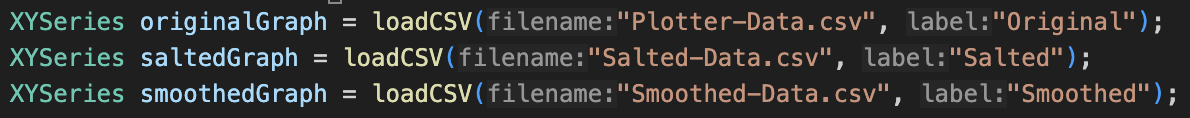
Using the class XYSeries from JFreeChart, I had a way to store our X and Y values and preserve them in that order (X, Y). After looking at documentation, there is a method add(double x, double y) which added an X value with a corresponding Y value into our XYSeries ([Source](https://www.jfree.org/jfreechart/api/javadoc/org/jfree/data/xy/XYSeries.html)).

To read the CSV file, I used Java’s BufferedReader to read the CSV files made by our Java PSS. Using logic learned while making the Java PSS, I successfully split the X and Y values, and used the add() method to add all those values to a XYSeries. Implemented below:

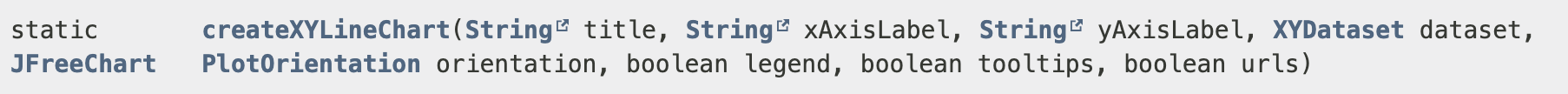
A screen shot of a computer program

AI-generated content may be incorrect.

As shown above, the method ultimate ends up returning the series now filled with the X and Y values of our CSV files. This is important, as we need this method to help process and store our Plotter, Salter, and Smoother CSV files into their own respective XYSeries. Below is the creation of the XYSeries and having them populated with their correct X and Y values using the method:



To plot these XYSeries, we can use the class ChartFactory to plot a XY line chart. Using the JFreeChart ChartFactory documentation, we need to create a dataset for the method createXYLineChart. Here is a snippet of the [official documentation](https://www.jfree.org/jfreechart/javadoc/org/jfree/chart/ChartFactory.html):



To create the dataset containing all of our XYSeries, we can use a XYSeriesCollection. Implemented below, I added all our XYSeries to our dataset

A black background with text

AI-generated content may be incorrect.

Finally, the ChartFactory is ready to be set up! Below is the implementation:

A screen shot of a computer program

AI-generated content may be incorrect.

All of the parameters directly explain what they are: the title is the title of the graph, the X and Y axis are being labeled, we are using the our dataset that has all of our XYSeries in them, PlotOrientation is vertical (X is horizontal and Y is vertical), the legend helps us differentiate the lines, tooltips help us how to navigate the graph, and no URLs are included in this graph.

To see this graph in its own window, we must use the JFrame library. This library comes included in Java, therefore no Maven dependency is needed. Using online tutorials and YouTube videos ([Tutorial](https://youtu.be/5o3fMLPY7qY)), I was able to make a simple window which handled the ChartFactory we had created. Implemented below:

A screen shot of a computer code

AI-generated content may be incorrect.

The first line sets the name of the window. The second line tells the program to stop running once we close the window. The ChartPanel we are adding contains the chart with our ChartFactory in it. The size of the window is set, then we set the window to be visible, and we get our graphs plotted using JFreeChart.

**Our Data plotted using JFreeChart**

**The function being used:**

A graph of different colored lines

AI-generated content may be incorrect.

**Original Graph by itself (using Excel)**

**Salted Graph by itself (using Excel)**

The Noise range for our Y-value is from 1 to 15 in this example.

**Smoothed Graph by itself (using Excel)**

This data was smoothed **7 times**.

**CSV Snippets**

|  |  |  |
| --- | --- | --- |
| **Plotter-Data.csv** | **Salted-Data.csv** | **Smoothed-Data.csv** |

# What is Apache Descriptive Statistics?

Apache Descriptive Statistics is a library in Java in which many statistics can be simply solved using their built-in methods. Just like our JFreeChart Program, Maven (extra credit) was used to import the package. This is what the dependency looks like in our Maven POM.xml file

A computer code with white text

AI-generated content may be incorrect.

All that is left to do is explore Apache’s library and documentation and use those methods to find interesting things about data we have. First, we must import the parts of the library we will be using. The ones used for my program are below:

A screen shot of a computer program

AI-generated content may be incorrect.

As you can already tell, Apache can handle many kinds of probability distributions. My program taken an array of random double made up by me and used Apache Stats Library to easily calculate some of these statistics and probabilities. Our array of data is:

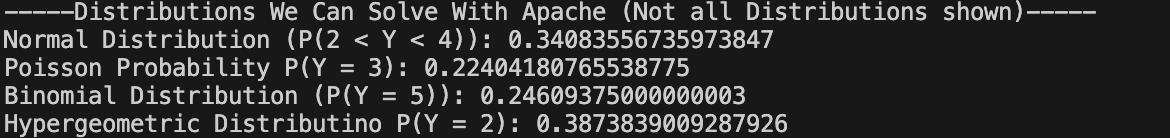


The results that come from this data.

A black screen with white text

AI-generated content may be incorrect.

Our distributions take different values, and those values are required to be put in when using their constructor. The values needed are made and just for demonstration purposes:



Apache can truly simplify these kinds of calculation without having to code their logic by hand which may be very difficult to handle, especially as probability distributions get more complicated.