Probability and Applied Statistics  
Project Two Report

Tyler Santosuosso  
Stockton University

Contents

[Octave Tutorial Summary 3](#_Toc121265085)

[Octave: Plot, Salt, and Smooth Program 7](#_Toc121265086)

[Java: Plot, Salt, and Smooth Program 9](#_Toc121265087)

[Chapter Four Write-Up 12](#_Toc121265088)

[The Normal Probability Distribution 12](#_Toc121265089)

[The Gamma Probability Distribution 13](#_Toc121265090)

[The Beta Probability Distribution 15](#_Toc121265091)

[Poker Project 16](#_Toc121265092)

[Resources 17](#_Toc121265093)

# Octave Tutorial Summary

Octave is a high-level language very similar to python, however it is catered towards numerical computations. Octave contains libraries for vectors, matrices, plotting, text and file output, mathematical functions, polynomials, differential equations, linear algebra, sets, and more. Octave also allows you to write scripts to create your own functions.

The layout is rather straightforward and quick to get started. Using the command window, a user can access all the functions of Octave by simply typing them in, like one would with Windows command prompt. Using the *help* command followed by a function name, one can find a quick explanation for the function of the command specified. The *lookfor* command followed by a function name, will find all relevant functions with that keyword, and display them in the command window. Moreover, there is a built-in documentation function specified by *doc* followed by the command in question. This produces a window that lists all the documentation on the specified function. The command window can also be cleared using *clc*.

Text

Description automatically generatedText, letter

Description automatically generatedGraphical user interface, application

Description automatically generated

The output of every command typed is stored by default in a variable named *ans*. This can be changed by simply equating a new variable to the command being typed. There are a few rules to variables in Octave: variables can store numeric or string values, variable names are case sensitive, variable names cannot include special characters or start with a number, and strings must be enclosed with quotes. Variables are stored in a window on the Octave program, where they can be viewed, edited, and deleted.

Text, schematic

Description automatically generatedGraphical user interface, application, Word

Description automatically generated

Simple arithmetic can be done with *+ (addition), - (subtraction), \* (multiplication), / (division),* and *^ (exponentiation).* There are many mathematical functions within Octave, as described above, such as: *sin, cos, tan, asin, acos, atan, log, log10, exp, abs,* etc. The following constants are already defined: *pi*, *e, i, j, inf,* and *NaN*.

Graphical user interface, text

Description automatically generated

In Octave, a user can create either a row vector or column vector. A vector must be enclosed with square brackets and each entry is separated by a comma or semi-colon, for a row vector or column vector respectively. A matrix can be defined using similar logic. Each row is separated with commas, and then every new row needs to be prefixed with a semi-colon.

Chart

Description automatically generatedChart

Description automatically generated Text

Description automatically generated with low confidence

Matrices and vectors can be manipulated by the standard operators *+, -,* *\*,* and *^*. There is also a *‘ (transpose)* operator that can be used. Matrices, however, must be prefaced with the dot operator (*.*).

Text

Description automatically generated with medium confidence

Specific elements can also be accessed by using indexing in vectors and matrices. The row and column must be specified in a matrix by using *(row, column)* as the index parameters.

Text

Description automatically generated

It is also possible to specify ranges of elements as well by using the *start:step:stop* notation. This can also be used when creating a range of values, for example in a vector.

A picture containing logo

Description automatically generated

Plotting is relatively straightforward as well at the base level. Using the *plot()* command which takes in a set of X coordinates and a set of Y coordinates and returns a graphical output. The command *linspace(b,l,n)* is a useful function that creates a vector by taking in a start (*b*) value and stop (*l*) value with *n* linearly spaced numbers between those values. The following set of commands produces the given graph.



Chart, line chart

Description automatically generated

There are a multitude of formatting commands for graphs that can be accessed using the FMT argument in the command *plot(x,y, FMT, …)*. This section specifically applies to line plots, however there are many other plots available such as histograms, bar plots, tree plots, etc.

Octave also provides the ability to write scripts and custom functions. An octave file type is suffixed with ‘*.m*’. Scripts can be as simple as a few lines of commands that you can run in the command window.

Graphical user interface

Description automatically generated with low confidence

Functions are able to be created by the following syntax:   
*function {output variable} = fn\_functionName(p1,p2,…)*. They are then called the same as the script.



Chart, line chart

Description automatically generated

All of the information here is relevant to the plot, salt, and smooth project I did on Octave. I learned all these functions from a few sources, which I’ll highlight below. Octave is very extensive and there is still a lot more I could learn about it.

# Octave: Plot, Salt, and Smooth Program

Using what I learned about Octave, I wrote four functions in Octave to create this program. I have a function for plotting, salting, smoothing, and running the program. The plot function *fn\_plotXY()* take in two values for the range of data. The salt function *fn\_saltXY()* takes in the X vector and Y vector from the output of the plot function. The smooth function *fn\_smoothXY()* takes in the X vector and Y vector from the output of the salt function, as well as a window value for the average calculation. Finally, the ­run function *fn­\_plotSaltSmooth()* calls all the functions, stores the values in column vectors, outputs to CSV files, and plots each set on the same graph.

Chart, line chart

Description automatically generated

Background pattern

Description automatically generated

These .CSV files can be opened in Microsoft Excel or Google Sheets to be viewed and manipulated. Generally, you would take that data and create a graph of some kind in Excel or Sheets, however we have Octave doing that for us this time around.

The following graphs show the output for one run that was recorded. You can see the three plots that represent each data set with a legend to differentiate each one. Then you can see the .CSV files that were created in the same directory as the script locations, and each of these contains two columns of X and Y values respectively.

**Text

Description automatically generated**

Figure 1 fn\_plotXY()

**Text

Description automatically generated**

Figure 2 fn\_saltXY()

**Graphical user interface, text, application

Description automatically generated**

Figure 3 fn\_smoothXY()

**Text

Description automatically generated**

Figure 4 fn\_plotSaltSmooth()

# Java: Plot, Salt, and Smooth Program

For this program, I took my original plot, salt, and smooth program that just outputted to a .CSV file and used the Apache Commons library and the JFreeCharts library. Using these libraries allowed me to change the way the smoother works and change how the data is outputted. The plot class and salt class were able to remain the same, with a few minor adjustments. The project is made up of the following classes: *ChartCreator, DataPlotter, DataSalter, DataSmoother,* and *Main*. The only difference in this version of this program in terms of project structure is the addition of the *ChartCreator* class and the removal of the *CSVFunctions* and *Coordinate* classes.

Apache Commons contains a statistics library with a *DescriptiveStatistics* class. This class provides the capability to hold input data and produce rolling statistics, which is what was needed for the *DataSmoother* class. The user can input a window value, which is then assigned to the *DescriptiveStatistics* instance via the *DescriptiveStatistics.setWindowSize()* function. Then, the data is looped through, in this case the Y values, and stored in the *DescriptiveStatistics* instance via the *DescriptiveStatistics.addValue()* function. Then, the rolling mean (moving average) across the window value is calculated via the *DescriptiveStatistics.getMean()* function, which is appropriately assigned as the Y value at a certain index for a certain coordinate.

Text

Description automatically generated

Figure Declaration of DescriptiveStatistics object

****

Figure setWindowSize function

Text

Description automatically generated

Figure For loop that creates the dataset with the Y moving average

The way the data is stored in this program is different from the previous version. Previously, I used a *Coordinate* class that stored an X and Y value in an object to make it easier for accessing each value for each coordinate. Now, using the *JFreeCharts* library, X and Y coordinates are stored in an object called *XYDataset*. These objects are individually stored in an object called *XYSeries*, which acts like an array, but for the *XYDataset* objects. The *XYSeries* object that stores all the *XYDataset* objects for each corresponding class are then added to an object called *XYSeriesCollection* so that it can be called in the functions that are used for creating the graphs. These objects are used in each class, as we need to store the X and Y values for each process throughout the salt, plot, and smooth process.



Figure 4 Declaration of XYSeries objects to store X and Y data points

Graphical user interface, text

Description automatically generated

Figure 5 For loop to add our data points to the XYSeries object



Figure 6 Calling of addSeries(XYSeries) to add to our XYSeriesCollection object

Continuing with the *JFreeCharts* library, this program outputs data in a graphical way, as opposed to a .CSV file that is then manually loaded into Microsoft Excel. In the *ChartCreator* class that I made for this program, we use *JFrame* and *JPanel* to create a window that stores the chart created with the *JFreeChart* class.Using the *createXYLineChart* function included in *ChartFactory* (*ChartFactory.createXYLineChart)*, we can create an object that takes in *chartTitle, xAxisLabel, yAxisLabel,* and *inputData* respectively. Then, using this data the *ChartCreator* constructor automatically creates a window with the graphical data when it is called.

*Text

Description automatically generated*

Figure 7 Chart creator function within ChartCreator class

*Text

Description automatically generated*

Figure 8 Constructor for ChartCreator class that automatically displays graphical window



Figure 9 Example of a constructor call on ChartCreator within the DataPlotter class

Then, our *Main* class simply runs the program by calling the *DataPlotter, DataSalter,* and *DataSmoother* classes and then calling their corresponding functions to create each of our outputs, therefore completing the program.

A screenshot of a computer

Description automatically generated

Figure 10 Main function

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

# Chapter Four Write-Up

## The Normal Probability Distribution

The Normal Probability Distribution is the most widely used continuous probability distribution. It has the familiar bell shape and relates to the empirical rule. The normal density function is described like this: *A random variable Y is said to have a normal probability distribution if and only if, for* σ > 0 and - ∞ < µ < ∞, the density function of Y is

The normal density function contains the variables µ and σ, which are the expected value of Y and standard deviation of Y respectively. The variance of Y is .

and .

Diagram

Description automatically generated with low confidence

Figure 1 Normal Probability Density Function

## The Gamma Probability Distribution

The Gamma Probability Distribution adequately models random variables with density functions that are associated with populations that have a skewed frequency distribution. It is described as follows: *A random variable Y is said to have a gamma distribution with parameters α > 0 and β > 0 if and only if the density function of Y is*

*where*

The quantity is known as the *gamma function*. Direct integration will verify that . Integration by parts will verify that for any and that provided that n is an integer.

The density function also contains two parameters α and β. *If Y has a gamma distribution with parameters α and β, then*

and

Shape, rectangle

Description automatically generated

Figure 2 Skewed Density Function

Chart

Description automatically generated with low confidence

Figure 3 Gamma Density Function

There are two special cases of gamma-distributed random variables.

1. *Let v be a positive integer. A random variable Y is said to have a chi-square distribution with v degrees of freedom if and only if Y is a gamma-distributed random variable with parameters α = ν/2 and β = 2.* A random variable with a chi-square distribution is called a chi-square (χ2)random variable.

*If Y is a chi-square random variable with v degrees of freedom, then*

*and .*

1. *A random variable Y is said to have an exponential distribution with parameter β > 0 if and only if the density function of Y is*

*If Y is an exponential random variable with parameter β, then*

*and .*

## The Beta Probability Distribution

The beta density function is a two-parameter density function defined over the closed interval It is used as a model for proportions most often. *A random variable Y is said to have a beta probability distribution with parameters and if any only if the density function of Y is*

*where*

Diagram

Description automatically generated

Figure 4 Beta Density Function

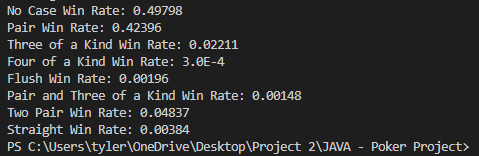
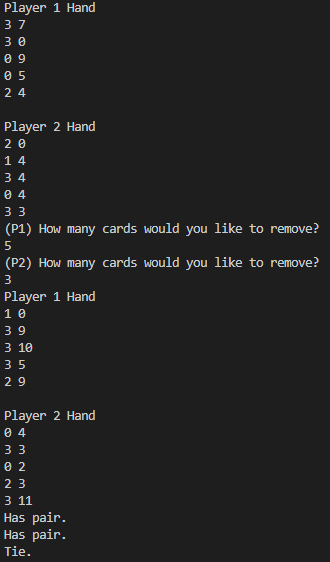
The cumulative distribution function for the beta random variable is commonly called the *incomplete beta function* and is denoted by

*If Y is a beta-distributed random variable with parameters α > 0 and β > 0, then*

*and*

# Poker Project

The poker project was rather straightforward. I created a crude design for calculating the probability, which came out correctly. The game design was also rather crude but did the job right. I was refactoring the whole project, but I ran out of time unfortunately. The refactored files are located in the Test folder. The following are the outputs for the crude version.



# Resources

* Relevant to Octave Tutorial Report
  + [Octave Tutorial - YouTube](https://www.youtube.com/watch?v=TqwSlEsbObg&t=1206s)
  + [Octave Programming Tutorial - Wikibooks, open books for an open world](https://en.wikibooks.org/wiki/Octave_Programming_Tutorial)