**StatsLibrary - Full Documentation**

Author: Arnab Das Utsa

# Abstract

The StatsLibraryEnding Java class is a custom-built statistical utility designed to simplify the computation of important statistical measures and distributions. It includes functionalities for the Poisson distribution, Uniform distribution, and Chebyshev’s Theorem. These are fundamental tools in probability and statistics, especially for modeling real-world phenomena such as rare events (Poisson), uniformly distributed data, and probability bounds (Tchebyshev). This documentation explains every method and constructor within the class, with clear mathematical context and usage examples. An additional testing class demonstrates real usage and validation of each method implemented.

# Class Overview: StatsLibraryEnding

This class contains several methods that are helpful for basic and intermediate-level statistical computations. It aims to serve students and early-stage data analysts by implementing commonly taught formulas directly in Java.

## Constructors

Default Constructor:

public StatsLibraryEnding() {}

This constructor initializes a new object with no parameters. It is useful when you plan to call methods independently and don’t need to pass any data during the creation of the object.

Constructor with String:

public StatsLibraryEnding(String input) {}

This version prepares the class to be used in scenarios where input data might be passed as a string. Although not currently used, it offers flexibility for future enhancements.

Constructor with Double:

public StatsLibraryEnding(double input) {}

This constructor is similar to the string constructor but is tailored to cases where a numerical input is immediately required.

## Method: findFactorial

public BigInteger findFactorial(int n) {  
 BigInteger factorial = new BigInteger(Integer.toString(n));  
 BigInteger zero = new BigInteger("0");  
 BigInteger one = new BigInteger("1");  
 if (factorial.equals(zero)) {  
 factorial = one;  
 } else {  
 for (int i = n; i > 1; i--) {  
 BigInteger index = new BigInteger(Integer.toString(i));  
 BigInteger oneLessThanIndex = index.subtract(one);  
 factorial = factorial.multiply(oneLessThanIndex);  
 }  
 }  
 return factorial;  
}

This method calculates the factorial of a given integer n using the BigInteger class, which can handle extremely large numbers. It starts from n and multiplies down to 1 (n!). The factorial of 0 is explicitly checked and set to 1, as 0! = 1. This method is critical for computing probabilities in discrete distributions such as Poisson, where factorials are often used.

## Method: findPoisson

public double findPoisson(double l, int y) {  
 double e = Math.E;  
 double factorialOfy = findFactorial(y).doubleValue();  
 double numerator = (Math.exp(-l) \* Math.pow(l, y));  
 double poisson = numerator / factorialOfy;  
 return poisson;  
}

This method implements the formula for the Poisson probability distribution:  
P(Y = y) = (e^-λ \* λ^y) / y!  
  
It calculates the probability that exactly y events will occur in a fixed time interval given an average rate λ (lambda). The exponential and power functions are applied to λ, and then the result is divided by the factorial of y. The use of Math.E ensures mathematical accuracy for e.

## Method: findExpectedPoisson

public double findExpectedPoisson(double l) {  
 return l;  
}

This method returns the expected value (mean) of a Poisson distribution, which is equal to λ (lambda). In Poisson distributions, the average rate of occurrence is both the mean and the variance.

## Method: findVariancePoisson

public double findVariancePoisson(double l) {  
 return l;  
}

This method returns the variance of a Poisson distribution. As with the mean, the variance of a Poisson distribution is also equal to the rate parameter λ. This makes Poisson a useful model when the spread and average of event counts are equal.

## Method: findChebyshev

public double findChebyshev(double k) {  
 double chebyshev = 1 - (1 / Math.pow(k, 2));  
 return chebyshev;  
}

Chebyshev’s Theorem gives a lower bound on the probability that a random variable lies within k standard deviations of the mean. This method computes that bound using the formula: 1 - 1/k². The result shows that regardless of the distribution shape, at least this much probability mass lies within the range.

## Method: findUniform

public double findUniform(double theta1, double theta2) {  
 double uniform = 1 / (theta2 - theta1);  
 return uniform;  
}

This method calculates the height (constant probability density) of a continuous uniform distribution over the interval [θ1, θ2]. The uniform distribution assumes all values in the range have equal probability, and the result is 1 divided by the width of the interval.

## Method: findExpectedUniform

public double findExpectedUniform(double theta1, double theta2) {  
 double expected = (theta1 + theta2) / 2;  
 return expected;  
}

This method returns the expected value (mean) of a uniform distribution. It is calculated as the midpoint between the lower bound (θ1) and the upper bound (θ2), reflecting the symmetric nature of the distribution.

## Method: findVarianceUniform

public double findVarianceUniform(double theta1, double theta2) {  
 double variance = Math.pow((theta2 - theta1), 2) / 12;  
 return variance;  
}

This method computes the variance of a uniform distribution, which measures the spread of values. The formula used is ((θ2 - θ1)²) / 12, based on the properties of uniform distributions.

# Class: StatsLibraryEndingTester

This tester class demonstrates the usage of all the methods in the StatsLibraryEnding class. It initializes a sample instance and prints results for different statistical operations.

public static void main(String[] args) {  
 StatsLibraryEnding test = new StatsLibraryEnding();  
 double testerResults1 = test.findPoisson(2.3, 4);  
 double testerResults2 = test.findExpectedPoisson(2.3);  
 double testerResults3 = test.findVariancePoisson(2.3);  
 double testerResults4 = test.findChebyshev(2.7);  
 double testerResults5 = test.findUniform(0.2, 0.7);  
 double testerResults6 = test.findExpectedUniform(0.2, 0.7);  
 double testerResults7 = test.findVarianceUniform(0.2, 0.7);  
  
 System.out.println("Poisson Distribution: " + testerResults1);  
 System.out.println("Expected Poisson: " + testerResults2);  
 System.out.println("Variance Poisson: " + testerResults3);  
 System.out.println("Chebyshev's Theorem: " + testerResults4);  
 System.out.println("Uniform Distribution: " + testerResults5);  
 System.out.println("Expected Uniform: " + testerResults6);  
 System.out.println("Variance Uniform: " + testerResults7);  
}

This method serves as a driver program to ensure all implemented features work correctly. Each line executes a specific statistical function and prints the outcome to the console with a description label.