# Sophia Milask: Stats Library #2

Purpose: To test formulas programmed in the second Stats Library for Probability and Applied Statistics by using problems from the textbook.

## Poisson Distribution

The method doPoissonDistribution takes in two arguments. The first being a double for the lambda value, and the second being an integer for the y-value. With these two values, the method directly applied the Poisson Distribution formula to find the probability of y events occurring over a specified interval of time with a constant average rate of occurrence.

*Code Segment for doPoissonDistribution*

```java
//method to find Poisson distribution with arguments for lambda and y
public double doPoissonDistribution(double lambda, int y) {
    //validating that lambda is positive and y is nonnegative
    if(lambda > 0 && y >= 0) {
        //poisson formula. Calling doFactorial to help, changing to a double to return
        return (Math.pow(Math.E, -lambda) * Math.pow(lambda, y)) / doFactorial(y).doubleValue();
    }
    else {
        //printed statement if the lambda and y-value don't meet the requirements
        System.out.println("Lambda must be positive and y must be nonnegative");
        //returning not a number if invalid
        return Double.NaN;
    }
}
```

# Problem 1: 3.122 –

Customers arrive at a checkout counter in a department store according to a Poisson distribution at an average of seven per hour. During a given hour, what are the probabilities that

a. No more than three customers arrive?
b. At least two customers arrive?
c. Exactly five customers arrive?

For part a, the question is asking for the summation of y = 0, 1, 2, and 3. So, in my tester class I will call my doPoissonDistribution method 4 times, passing through (7, 0), (7, 1), (7, 2), and (7, 3). I will then add up the return values and print them out.

## *Input (3.122 part a)*

```java
    public static void main(String[] args) {
        //format to 3 decimal places
        DecimalFormat format = new DecimalFormat("#.###");
        StatsFunctions s = new StatsFunctions();
        System.out.println(format.format(s.doPoissonDistribution(7, 0) + s.doPoissonDistribution(7, 1) + s.doPoissonDistribution(7, 2) +
                s.doPoissonDistribution(7, 3)));
```

## *Output (3.122 part a)*

```
Problems  Javadoc  Declaration  Console ×  Progress
<terminated> StatsTester (1) [Java Application] C:\Users\sophi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.0.v20240919-1706\jre\bin\javaw.exe  (May 3, 2025, 10:10:1
0.082
```

For part b, the question is asking for the complement of the summation of y = 0 and 1. So, in my tester class I will call my doPoissonDistribution method twice, passing through (2, 0) and (2, 1), add them, and subtract their sum from 1.

## *Input (3.122 part b)*

```java
    public static void main(String[] args) {
        //format to 3 decimal places
        DecimalFormat format = new DecimalFormat("#.###");
        StatsFunctions s = new StatsFunctions();
        System.out.println(format.format(1 - (s.doPoissonDistribution(7, 0) + s.doPoissonDistribution(7,  1))));

    }
```

*Output (3.122 part b)*

```
Problems  Javadoc  Declaration  Console ×  Progress
<terminated> StatsTester (1) [Java Application] C:\Users\sophi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.0.v20240919-1706\jre\b
0.993
```

For part c, I simply have to call doPoissonDistribution once and pass through (7, 5).

*Input (3.122 part c)*

```java
    public static void main(String[] args) {
        //format to 3 decimal places
        DecimalFormat format = new DecimalFormat("#.###");
        StatsFunctions s = new StatsFunctions();
        System.out.println(format.format(s.doPoissonDistribution(7, 5)));

    }
```

*Output (3.122 part c)*

```
Problems  Javadoc  Declaration  Console ×  Progress
<terminated> StatsTester (1) [Java Application] C:\Users\sophi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot
0.128
```

## Problem 2: 3.128 –

Cars arrive at a toll booth according to a Poisson process with mean 80 cars per hour. If the attendant makes a one-minute phone call, what is the probability that at least 1 car arrives during the call?

First, we must realize that we have to use dimensional analysis to find the lambda value that we need to plug in. We do this by $\frac{80\ cars}{1\ hour} \times \frac{1\ hour}{60\ minutes} = \frac{80\ cars}{60\ miutes} = \frac{4}{3}$ cars per minute. So, 4/3 is our lambda value and the question is asking for the complement of when y = 0. So, I will call my doPoissonDistribution method and pass through (4/3, 0) and subtract that value from 1.

### *Input (3.128)*

```
    public static void main(String[] args) {
        //format to 3 decimal places
        DecimalFormat format = new DecimalFormat("#.###");
        StatsFunctions s = new StatsFunctions();
        System.out.println(format.format(1 - s.doPoissonDistribution((4.0/3), 0)));
```

### *Output (3.128)*

```
Problems  @ Javadoc  Declaration  Console ×  Progress
<terminated> StatsTester (1) [Java Application] C:\Users\sophi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.wi
0.736
```

## Tchebysheff's Theorem

The method findPercentOfData takes in four arguments. The first two are doubles: one for the upper bound and one for the lower bound around the mean. The third argument is a double for the mean, and fourth is a double for the standard deviation. With this information, the method finds the percent of data which lies between the upper and lower bounds given.

### *Code Segment for findPercentOfData*

```
    //method which takes arguments for the upper and lower bound around the mean, the mean, and standard deviation to find the
    //percent of data which lies between the bounds by applying Tchebysheff's Theorem
    public String findPercentOfData(double lower, double upper, double mean, double stdDev) {
        //ensuring the lower and upper bounds are symmetric about the mean so we can apply the theorem
        if((mean - lower) == (upper - mean)) {
            //formula and logic statement
            return ((1 - 1 / Math.pow((mean - lower) / stdDev, 2)) * 100) + "% of the data falls between " + lower + " and " + upper;
        } else {
            //print statement if the bounds are not symmetric about the mean
            return "The interval must be symmetric around the mean to directly apply Tchebycheff's Theorem";
        }
    }
```

## Problem 1: 3.167 –

Let Y be a random variable with mean 11 and variance 9. Using Tchebysheff's Theorem, find a lower bound for $P(6 < Y < 16)$.

We first need to realize we are given variance, but we need standard deviation for the formula. So, the standard deviation would be 3. Next, we are given the upper and lower bounds and mean, so we have everything we need to call the method. So, I simply just have to print out findPercentOfData(6, 16, 11, 3).

### *Input (3.167 part a)*

```
        StatsFunctions s = new StatsFunctions();
        System.out.println(s.findPercentOfData(6, 16, 11, 3));

    }
```

### *Output (3.167 part a)*

```
Problems  @ Javadoc  Declaration  Console ×  Progress
<terminated> StatsTester (1) [Java Application] C:\Users\sophi\.p2\pool\plugins\org.eclipse.justj.ope
64% of the data falls between 6.0 and 16.0
```

## Problem 2: 3.170 –

The U.S. mint produces dimes with an average diameter of .5 inch and standard deviation .01. Using Tchebysheff's Theorem, find a lower bound for the number of coins in a lot of 400 coins that are expected to have a diameter between .48 and .52 inches.

This is a 2-part problem. Since we are given the lower and upper bounds, mean, and standard deviation, we can simply call findPercentOfData(.48, .52, .5, .01) and find what percent of the coins fall between .48 and .54 inches. The console shows us the answer is 75%. So, 75% of the 400 coins are expected to fall within that range. 400(.75) = 300 coins.

*Input (3.170)*

```
        DecimalFormat format = new DecimalFormat("#.###");
        StatsFunctions s = new StatsFunctions();
        System.out.println(s.findPercentOfData(.48, .52, .5, .01));

    }
```

*Output (3.170)*

```
Problems  Javadoc  Declaration  Console  Progress
<terminated> StatsTester (1) [Java Application] C:\Users\sophi\.p2\pool\plugins\org.eclipse.justj.openjdk.h
75% of the data falls between 0.48 and 0.52
```

## Uniform Distribution

The method doUniformDistribution takes in four arguments. The first two are given from the problem: the known upper and lower bounds of the uniform distribution. The second two arguments are given by the upper and lower bounds we want to find the probability of success occurring between.

The method findUniformExpected takes in two arguments: the known upper and lower bounds of the uniform distribution and uses that information to find the expected value.

The method findUniformVariance takes in two arguments: the known upper and lower bounds of the uniform distribution and uses that information to find the variance of the distribution.

*Code Segment for doUniformDistribution*

```java
//method which takes arguments for the given upper and lower bounds, and the upper and lower bounds in question to find the
//uniform distribution
public double doUniformDistribution(double knownUpper, double knownLower, double chanceUpper, double chanceLower) {
    //formula
    return (chanceUpper - chanceLower) / (knownUpper - knownLower);
}
```

*Code Segment for findUniformExpected*

```java
//method which takes in the upper and lower bounds of a uniform distribution to find the expected value
public double findUniformExpected(double upper, double lower) {
    //formula
    return (upper + lower) / 2;
}
```

*Code Segment for findUniformVariance*

```java
//method which takes in the upper and lower bounds of a uniform distribution to find the variance
public double findUniformVariance(double upper, double lower) {
    //formula
    return (Math.pow((upper - lower), 2)) / 12;
}
```

## Problem 1: 4.45 –

Upon studying low bids for shipping contracts, a microcomputer manufacturing company finds that intrastate contracts have low bids that are uniformly distributed between 20 and 25, in units of thousands of dollars. Find the probability that the low bid on the next intrastate shipping contract

a. is below $22,000.
b. Is in excess of $24,000.

For part a, we first realize that our given range is from 20,000 to 25,000. Next, we see that below 22,000 means a range from 20,000 to 22,000. So, we are ready to plug into our formula and call doUniformDistribution(25000, 20000, 22000, 20000).

*Input (4.45 part a)*

```java
StatsFunctions s = new StatsFunctions();
System.out.println(s.doUniformDistribution(25000, 20000, 22000, 20000));
```

*Output (4.45 part a)*

```
Problems @ Javadoc Declaration Console × Progress
<terminated> StatsTester (1) [Java Application] C:\Users\sophi\.p2\pool\plugins\org.ecli
0.4
```

For part b, our given range of course stays the same, but the new range we are trying to find the probability of success within is above 24,000, which really means the range of 24,000 to 25,000. So, I can call doUniformDistribution(25000, 20000, 25000, 24000).

*Input (4.45 part b)*

```java
StatsFunctions s = new StatsFunctions();
System.out.println(s.doUniformDistribution(25000, 20000, 25000, 24000));
```

*Output (4.45 part b)*

```
Problems @ Javadoc Declaration Console × Progress
<terminated> StatsTester (1) [Java Application] C:\Users\sophi\.p2\pool\plugins\org.eclipse.justj.openjdk.ho
0.2
```

## Problem 2: 4.52 –

The cycle times for hauling concrete to a highway construction site is uniformly distributed over the interval 50 to 70 minutes. Find the mean and variance of the cycle times for the trucks.

This problem is very straightforward. The upper limit of the range is 70 and the lower limit is 50. So, all we have to do is call findUniformExpected(70, 50) and findUniformVariance(70, 50).

*Input (4.52)*

```java
public static void main(String[] args) {
    //format to 3 decimal places
    DecimalFormat format = new DecimalFormat("#.###");
    StatsFunctions s = new StatsFunctions();
    System.out.println(format.format(s.findUniformExpected(70, 50)));
    System.out.println(format.format(s.findUniformVariance(70, 50)));
```

*Output (4.52)*

```
Problems  @ Javadoc  Declaration  Console ×  Progress
<terminated> StatsTester (1) [Java Application] C:\Users\sophi\.p2\pool\plugins\org.e
60
33.333
```