BLG 252E - Object Oriented Programming
Assignment #3

Due: May, 16th 23:59

"If the number of experiments be very large, we may have precise information as to the value of mean."

-Student (William Sealy Gosset) (1908)

Introduction

For this assignment, you are expected to implement a "**Confidence Interval Estimator**". To get an intuition about the process, please check the "An Illustration of the Assignment" section. For any issues regarding the assignment, you can ask your questions on Ninova Message Board. Before writing your question please check whether your question has been asked. You can also contact T.A. Fatih Bektaş. (**bektas18@itu.edu.tr**)

Implementation Notes

The implementation details below are included in the grading:

1. Please follow a consistent coding style (indentation, variable names, etc.) with comments otherwise you will get **minus** points.
2. You are only allowed to use certain STL containers included in skeleton code.
3. You may (and should) implement any getter methods when they are needed.
4. Make sure that there is no memory leak in your code.
5. You need to decide correctly which attributes or methods will be private, public or protected. This will be a grading criteria.
6. Since your code will be tested automatically, make sure that your outputs match with sample scenario for given inputs. You will be provided with a calico file to check your assignment output.
7. Leave the "**main.cpp**" file as it is since there is no need to make any changes on it.

Submission Notes

1. Your program should compile and run on Linux environment. You can (and should) code and test your program on Docker's Virtual Environment. Do not use any pre-compiled header files. Be sure that you have submitted all of your files.
2. Submissions are made through **only** the Ninova system and have a strict deadline. Assignments submitted after the deadline will not be accepted.
3. This is not a group assignment and getting involved in any kind of cheating is subject to disciplinary actions. Your assignment should not include any copy-paste material (from the Internet or from someone else's paper/thesis/project).

1 Implementation Details

You are expected to implement 4 classes; **StatisticalDistribution**, **NormalDistribution**, **UniformDistribution**, **ExponentialDistribution**.

1.1 StatisticalDistribution

This class will be an **Abstract Base Class**.

Attributes:

1. "**mean**" (type: double): Theoretical value of the mean of chosen distribution.

Methods:

1. **Getter(s):** You should implement a getter method to get the "mean" value.
2. "**sample**": This will be a **pure virtual method**. You can figure out the return type and parameters looking at given "main.cpp" code.
3. "**calculate_confidence_interval**": Calculates the confidence interval for a given sample. You can figure out the return type and parameters looking at given "main.cpp" code. You need to do calculations by implementing the formula below. It will return a thee-element vector: [lower_bound, sample_mean, upper_bound].

$$(\tilde{X} - t \times \frac{S}{\sqrt{n}},\ \tilde{X},\ \tilde{X} + t \times \frac{S}{\sqrt{n}})$$

The term at the left hand side represents the "lower_bound", in the middle "sample_mean" and at the right hand side upper_bound. "$\tilde{X}$" is sample mean, "t" is the "t_value" provided in input text file. "S" is sample standard deviation and "n" is the sample size. You need do calculate "$\tilde{X}$" and "S" from your sample. The other variables are given in input file.

1.2 NormalDistribution

This class will be derived from "**StatisticalDistribution**" class.

Attributes:

1. "**stddev**" (type: double): Standard deviation value.

Methods:

1. **Constructor(s):** The constructor should initialize "mean" as 0.0 and "stddev" as 1.0.
2. "**sample**": Create a sample from a Normal Distribution, given the sample size and seed value for random number generator. Return the sample as a vector. You can generate a random number from a Normal Distribution using the code snippet below. You need to repeat this process to create a sample of given size. For more detail check normal_distribution reference.

```
std::default_random_engine generator;
generator.seed(seed_val);
std::normal_distribution<double> distribution(mean, stddev);
double number = distribution(generator);
```

1.3 UniformDistribution

This class will be derived from "**StatisticalDistribution**" class.

Attributes:

1. "**a**" (type: double): Beginning value of range.
2. "**b**" (type: double): Ending value of range.

Methods:

1. **Constructor(s):** The constructor should initialize "mean" as 0.0, "a" as -1.0 and "b" as 1.0.
2. "**sample**": Create a sample from a Uniform Distribution, given the sample size and seed value for random number generator. Return the sample as a vector. You can generate a random number from a Uniform Distribution using the code snippet below. You need to repeat this process to create a sample of given size. For more detail check uniform_real_distribution reference.

```
std::default_random_engine generator;
generator.seed(seed_val);
std::uniform_real_distribution<double> distribution(a, b);
double number = distribution(generator);
```

1.4 ExponentialDistribution

This class will be derived from "**StatisticalDistribution**" class.

Attributes:

1. "**lambda**" (type: double): Average rate of occurrence.

Methods:

1. **Constructor(s):** The constructor should initialize "mean" as 1.0 and "lambda" as 1.0.
2. "**sample**": Create a sample from a Exponential Distribution, given the sample size and seed value for random number generator. Return the sample as a vector. You can generate a random number from a Exponential Distribution using the code snippet below. You need to repeat this process to create a sample of given size. For more detail check exponential_distribution reference.

```
std::default_random_engine generator;
generator.seed(seed_val);
std::exponential_distribution<double> distribution(lambda);
double number = distribution(generator);
```

1.5 Necessary Methods You Need to Implement in "utils.cpp":

1. "**choose_function**": Choose a distribution function by given number.
   - 0 for "NormalDistribution"
   - 1 for "UniformDistribution"
   - 2 for "ExponentialDistribution"
   - Otherwise **throw an exception**: "Unidentified distribution function!"
2. "**many_trials**": Create a random sample and calculate the confidence interval for many times (repeat the number of times given as "trials"). Then, return the precision described below:

$$precision = \frac{\#\ of\ successes}{\#\ of\ trials}$$

   Success describes how many times your calculated confidence interval covered the theoretical mean.

Warning: Do not forget to **increment seed_value by one** at each trial. Otherwise, you will get the same sample at each trial.

Notice: You can figure out what numbers in the test cases represent by examining "main.cpp" file.

2 An Illustration of the Assignment

With this application, you are actually coding an illustration of a confidence level estimator. To see and play with it please visit "seeing-theory".

3 How to compile, run and test your code

If you want to compile and run the provided code on terminal, you can use these commands:

g++ -Wall -Werror src/main.cpp src/statistics.cpp src/utils.cpp -I include -o hw3 -lm
./hw3 test_case_1.txt

You can (and should) run the calico file on terminal to check your assignment with the command:

calico hw3.yaml --debug

4 Submission

Submit your **statistics.cpp**, **statistics.h**, **utils.cpp** and **utils.h** files to Ninova. Before submitting please make sure you are passing all cases in calico file. Calico file you received is not the grading file so taking 100 does not corresponds to full grade for your assignment. Passing all cases will show you that your output is compatible with the grading file. Otherwise you might not receive a full grade from the assignment.