МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ |

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | К.А. Кочин |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3 |
| Реализация на ЭВМ алгоритмов статистического оценивания параметров выборки |
| по дисциплине: Прикладная теория вероятностей и статистика |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | Z1431 |  |  |  | М.Д. Быстров |
|  | номер группы |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | 2021/3572 | |  |  |  |

Санкт-Петербург 2024

ЗАДАНИЕ

на лабораторное занятие № 3 по дисциплине

«Прикладная теория вероятностей и статистика»

I. Тема: Реализация на ЭВМ алгоритмов статистического оценивания параметров выборки.

II. Исходные данные.

Произведено *N* наблюдений константы *с* – параметра с постоянным значением:

 ,

где  – наблюдаемое значение параметра,  – истинное значение параметра,  – значение ошибки (помехи).

*Задача*: по полученной реализации вектора наблюдения  найти оценку  параметра *х*: , т.е. найти правило, или *алгоритм*,  преобразования вектора наблюдений  в оценку .

III. Выполнить:

1. Построить вариационные ряды для выборок различного объема.
2. Синтезировать алгоритм вычисления оценки  – статистического (выборочного) математического ожидания .
3. Синтезировать алгоритм вычисления оценки  – полусуммы минимального и максимального элементов выборки.
4. Синтезировать алгоритм вычисления оценки  – статистической (выборочной) медианы .
5. Синтезировать алгоритм вычисления оценки  – среднего арифметического с отбросом с обеих сторон вариационного ряда по *k* крайних членов (*k* = 1 при *N* < 10, *k* = 2 при *N* ≤ 15, *k* = 3 при *N* > 15).
6. Вычислить оценку  среднего квадратического отклонения .
7. Оценить точность полученных результатов по формуле при условии, что «истинное» (эталонное) значение параметра *х*эт = с.
8. Ошибка оценивания: Δ*х* = |*х*эт – |.
9. Относительная ошибка оценивания: δ*х* = Δ*х* / *х*эт.
10. Сделать выводы по работе.

Номер варианта соответствует порядковому номеру в списке учебной группы

Номер варианта (C) = 2.

1. **Код программы моделирования случайных величин и оценивания параметров выборки**

1. ./Utils.cs

﻿**using** System.Diagnostics;

**namespace** Core

{

**public** **class** Utils

{

**public** **static** **void** OpenPath(**string** fileName)

{

**var** processStart = new ProcessStartInfo()

{

FileName = "cmd",

Arguments = $"/c {fileName}"

};

**var** process = new Process();

process.StartInfo = processStart;

process.Start();

}

**public** **static** **double**[] GetUniformRandomValues(**int** count)

{

**var** randomValues = new **double**[count];

**var** random = new Random();

**for** (**int** i = 0; i < count; i++)

{

randomValues[i] = random.NextDouble();

}

**return** randomValues;

}

**public** **static** **async** Task SaveValues(**string** fileName, **double**[] values)

{

**await** File.AppendAllLinesAsync(

fileName,

values.**Select**(v => v.ToString()));

}

**public** **static** **async** Task<**double**[]> ReadValues(**string** fileName)

{

**return** (**await** File.ReadAllLinesAsync(fileName))

.**Select**(**double**.Parse)

.ToArray();

}

*/// <summary>*

*/// подтоговка значений равномерно распределенной случайной величины*

*/// </summary>*

*/// <param name="count"> кол-во значений </param>*

*/// <param name="a"> значение a для линейного преобразования </param>*

*/// <param name="b"> значение b для линейного преобразования </param>*

*/// <returns></returns>*

**public** **static** **double**[] PrepareUniformData(**int** count, **double** a = 0, **double** b = 1)

{

**var** values = Utils.GetUniformRandomValues(count).**Select**(v => UniformLinearTransform(v, a, b));

**return** values.ToArray();

}

*/// <summary>*

*/// Линейное преобразование значения для элемента ряда равномерного распределения*

*/// </summary>*

*/// <param name="value"></param>*

*/// <param name="a"></param>*

*/// <param name="b"></param>*

*/// <returns></returns>*

**public** **static** **double** UniformLinearTransform(

**double** **value**,

**double** a,

**double** b)

{

**return** a + **value** \* (b - a);

}

**public** **static** **double**[] PrepareGaussData(**int** count, **double** m, **double** sigma)

{

**var** values = Utils.GetUniformRandomValues(count \* 2);

**var** ret = new **double**[count];

**for** (**int** i = 0; i < count; i++)

{

ret[i] = CreateGaussValue(values[i], values[i + 1]);

}

ret = ret

.**Select**(v => GaussLinearTransform(v, m, sigma))

.ToArray();

**return** ret;

}

*/// <summary>*

*/// Линейное преобразование элемента нормально распределеного ряда*

*/// </summary>*

*/// <param name="value"></param>*

*/// <param name="m"></param>*

*/// <param name="sigma"></param>*

*/// <returns></returns>*

**public** **static** **double** GaussLinearTransform(

**double** **value**,

**double** m,

**double** sigma)

{

**return** m + sigma \* **value**;

}

*/// <summary>*

*/// распределение произведения двух независимых случайных величин,*

*/// одна из которых имеет распределение Релея,*

*/// а другая распределена по закону арксинуса, является нормальным*

*/// </summary>*

*/// <param name="uniformValue1"></param>*

*/// <param name="uniformValue2"></param>*

*/// <returns></returns>*

**public** **static** **double** CreateGaussValue(

**double** uniformValue1,

**double** uniformValue2)

{

**return** Math.Sin(2 \* Math.PI \* uniformValue1)

\* Math.Sqrt(-2 \* Math.Log(uniformValue2));

}

}

}

2. ./Lab3.cs

﻿**using** Core;

**using** Lab3.Processing;

**using** System.Net.Http.Headers;

**using** **static** Core.Utils;

**namespace** Lab3

{

**internal** **class** Lab3

{

**private** **const** **int** VARIANT = 2;

**private** **const** **int** C = VARIANT;

**private** **static** **readonly** **int**[] sampleCounts = [15, 30, 100, 1000];

**private** **const** **string** REPORT\_FILENAME = "reportTemplate.xlsx";

**static** **async** Task Main(**string**[] args)

{

*// подготовка набора данных*

**await** PrepareDataSet(sampleCounts, \_generators);

**var** valuesParameters = **await** ProcessData(

sampleCounts,

\_generators.Count);

CreateReport(valuesParameters);

}

**private** **static** **void** CreateReport(ValuesParameters[] parameters)

{

**var** newFileName = $"{DateTimeOffset.UtcNow.ToUnixTimeSeconds()}.xlsx";

**using** (**var** excelBuilder = new ExcelReportBuilder(

REPORT\_FILENAME,

newFileName))

{

**foreach** (**var** param **in** parameters)

{

excelBuilder.AppendReportLine(param);

}

excelBuilder.SaveReport();

}

Utils.OpenPath(newFileName);

}

*/// <summary>*

*/// Считывание и обработка данных*

*/// </summary>*

*/// <param name="counts"></param>*

*/// <param name="generatorsNum"></param>*

*/// <returns></returns>*

**private** **async** **static** Task<ValuesParameters[]> ProcessData(**int**[] counts, **int** generatorsNum)

{

**var** ret = new List<ValuesParameters>();

**foreach** (**var** count **in** counts)

{

**for** (**int** i = 0; i < generatorsNum; i++)

{

**var** fileName = BuildDataFileName(count, i + 1);

**var** values = **await** ReadValues(fileName);

**var** parameters = new ValuesParameters(

values,

values.Length / 4,

C);

ret.**Add**(parameters);

}

}

**return** ret.ToArray();

}

*/// <summary>*

*/// Генераторы последовательностей согласно заданию*

*/// </summary>*

**private** **static** List<ValuesGenerator> \_generators = new()

{

new (

count => PrepareGaussData(

count,

0,

(**double**) C / 100)

),

new (

count => PrepareGaussData(

count,

0,

(**double**) C / 20)

),

new (

count => PrepareUniformData(

count,

(**double**) - C / 100,

(**double**) C / 100)

),

new (

count => PrepareUniformData(

count,

(**double**) - C / 20,

(**double**) C / 20)

)

};

*/// <summary>*

*/// Подговка набора данных при необходимости*

*/// </summary>*

*/// <param name="counts">Список кол-ва</param>*

*/// <param name="generators"></param>*

*/// <returns></returns>*

**private** **static** **async** Task PrepareDataSet(

**int**[] counts,

List<ValuesGenerator> generators)

{

**foreach** (**var** count **in** counts)

{

**for** (**int** i = 0; i < generators.Count; i++)

{

ValuesGenerator generator = generators[i];

**string** fileName = BuildDataFileName(count, i + 1);

**if** (!File.Exists(fileName))

{

*// генерация помехи*

**double**[] values = generator.Generator!(count);

*// добавление к помехе коэффициента*

**for** (**int** j = 0; j < values.Length; j++)

{

values[j] += C;

}

**await** SaveValues(fileName, values);

}

}

}

}

**private** **static** **string** BuildDataFileName(**int** count, **int** order)

{

**return** $"{order}\_{count}\_values.txt";

}

}

}

3. ./Processing/EstimateErrorEngine.cs

﻿**namespace** Lab3.Processing

{

**public** **class** EstimateErrorEngine

{

*/// <summary>*

*/// Среднее значение - статистическое мат ожидание*

*/// </summary>*

*/// <param name="values"></param>*

*/// <returns></returns>*

**public** **static** **double** Average(IEnumerable<**double**> values)

{

**decimal** sum = 0;

**int** cnt = 0;

**foreach** (**var** **value** **in** values)

{

sum += (**decimal**)**value**;

cnt++;

}

sum /= cnt;

**return** (**double**)sum;

}

*/// <summary>*

*/// Полусумма*

*/// </summary>*

*/// <param name="values"></param>*

*/// <returns></returns>*

**public** **static** **double** HalfSum(IEnumerable<**double**> values)

{

**return** (values.Max() + values.Min()) / 2;

}

*/// <summary>*

*/// Медиана*

*/// </summary>*

*/// <param name="values"></param>*

*/// <returns></returns>*

**public** **static** **double** Median(IEnumerable<**double**> values)

{

**var** cnt = values.Count();

values = values.Order();

**if** (cnt % 2 == 0)

{

**return** (values.ElementAt(cnt / 2) + values.ElementAt(cnt / 2 - 1)) / 2;

}

**else**

{

**return** values.ElementAt(cnt / 2);

}

}

*/// <summary>*

*/// Среднее значение с отбросом K членов*

*/// </summary>*

*/// <param name="values"></param>*

*/// <param name="k"></param>*

*/// <returns></returns>*

*/// <exception cref="ApplicationException"></exception>*

**public** **static** **double** SegmentAverage(

IEnumerable<**double**> values,

**int** k)

{

**var** cnt = values.Count();

values = values.Order();

**if** (k >= cnt / 2)

{

**throw** new ApplicationException("k greater or equal than N/2");

}

**return** Average(values.Skip(k).Take(cnt - 2 \* k));

}

*/// <summary>*

*/// Дисперсия*

*/// </summary>*

*/// <param name="values"></param>*

*/// <returns></returns>*

**public** **static** **double** Dispersion (IEnumerable<**double**> values)

{

**var** cnt = values.Count();

**decimal** sum = 0;

**double** average = Average(values);

**foreach** (**var** **value** **in** values)

{

**var** difference = **value** - average;

**var** quad = Math.Pow(difference, 2);

sum += (**decimal**)quad;

}

**var** ret = sum / (cnt - 1);

**return** (**double**)ret;

}

*/// <summary>*

*/// Относительная ошибка оценивания*

*/// </summary>*

**public** **static** **double** RelativeEstimationError(**double** c, **double** x) =>

Math.Abs(c - x) / c;

}

}

4. ./Processing/ExcelReportBuilder.cs

﻿**using** OfficeOpenXml;

**using** System.Runtime.CompilerServices;

**namespace** Lab3.Processing

{

**internal** **class** ExcelReportBuilder : IDisposable

{

**public** ExcelPackage \_excelPackage;

**private** **bool** \_disposedValue;

**private** **int** \_cnt = 0;

**private** **int** \_sampleBlockSize;

**private** **const** **int** HEADER\_SIZE = 2;

**private** **const** **int** BLOCK\_HEADER\_SIZE = 2;

**private** **const** **int** START\_COLUMN = 1;

**public** ExcelReportBuilder(

**string** templateFileName,

**string** newFileName,

**int** sampleBlockSize = 4)

{

**if** (!File.Exists(templateFileName))

{

**throw** new FileNotFoundException(templateFileName);

}

\_excelPackage = new ExcelPackage(

new FileInfo(newFileName),

new FileInfo(templateFileName));

ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

\_sampleBlockSize = sampleBlockSize;

}

**private** **int** GetRowIdx(**int** count)

{

**return** HEADER\_SIZE

+ BLOCK\_HEADER\_SIZE \* ((count - 1) / \_sampleBlockSize + 1)

+ count;

}

**public** **void** AppendReportLine(ValuesParameters parameters)

{

**var** row = GetRowIdx(\_cnt + 1);

**var** column = START\_COLUMN;

**var** worksheet = \_excelPackage.Workbook.Worksheets[0];

worksheet.Cells[row, column + 1].**Value** = parameters.Average;

worksheet.Cells[row, column + 2].**Value** = parameters.HalfSum;

worksheet.Cells[row, column + 3].**Value** = parameters.Median;

worksheet.Cells[row, column + 4].**Value** = parameters.SegmentMedian;

worksheet.Cells[row, column + 5].**Value** = parameters.AverageError;

worksheet.Cells[row, column + 6].**Value** = parameters.HalfSumError;

worksheet.Cells[row, column + 7].**Value** = parameters.MedianError;

worksheet.Cells[row, column + 8].**Value** = parameters.SegmentMedianError;

worksheet.Cells[row, column + 1, row, column + 8].Style.Numberformat.Format = "0.000000";

worksheet.Cells[row, column + 1, row, column + 8].AutoFitColumns();

\_cnt++;

}

**public** **void** SaveReport()

{

\_excelPackage.Save();

}

**protected** **virtual** **void** Dispose(**bool** disposing)

{

**if** (!\_disposedValue)

{

**if** (disposing)

{

\_excelPackage.Dispose();

}

\_disposedValue = **true**;

}

}

**public** **void** Dispose()

{

*// Не изменяйте этот код. Разместите код очистки в методе "Dispose(bool disposing)".*

Dispose(disposing: **true**);

GC.SuppressFinalize(**this**);

}

}

}

5. ./Processing/ValuesGenerator.cs

﻿**namespace** Lab3.Processing

{

**internal** **class** ValuesGenerator

{

**public** ValuesGenerator(Func<**int**, **double**[]> generator)

{

Generator = generator;

}

**public** Func<**int**, **double**[]> Generator { **get**; }

}

}

6. ./Processing/ValuesParameters.cs

﻿**namespace** Lab3.Processing

{

**internal** **class** ValuesParameters

{

**private** **double**[] \_values;

**public** ValuesParameters(

**double**[] values,

**int** k,

**int** c)

{

**this**.\_values = values;

Average = EstimateErrorEngine.Average(\_values);

HalfSum = EstimateErrorEngine.HalfSum(\_values);

Median = EstimateErrorEngine.Median(\_values);

SegmentMedian = EstimateErrorEngine.SegmentAverage(\_values, k);

AverageError = EstimateErrorEngine.RelativeEstimationError(c, Average);

HalfSumError = EstimateErrorEngine.RelativeEstimationError(c, HalfSum);

MedianError = EstimateErrorEngine.RelativeEstimationError(c, Median);

SegmentMedianError = EstimateErrorEngine.RelativeEstimationError(c, SegmentMedian);

Dispersion = EstimateErrorEngine.Dispersion(\_values);

}

**public** **double** Average { **get**; }

**public** **double** HalfSum { **get**; }

**public** **double** Median { **get**; }

**public** **double** SegmentMedian { **get**; }

**public** **double** AverageError { **get**; }

**public** **double** HalfSumError { **get**; }

**public** **double** MedianError { **get**; }

**public** **double** SegmentMedianError { **get**; }

**public** **double** Dispersion { **get**; }

}

}

1. Таблица с результатами



**Выводы**

В ходе выполнения третьей лабораторной работы произведена реализация на ЭВМ алгоритмов статистического оценивания параметров выборки.

Сгенерировано 16 вариационных рядов помех, рассчитаны оценки для каждого из рядов.

Написана программа для выполнения операций в соответствии с заданием.

Приобретены навыки по реализации на ЭВМ алгоритмов статистического оценивания параметров выборки.