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 № 4 |
| Реализация на ЭВМ алгоритмов описательной статистики. |
| 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 № 4 по дисциплине

«Прикладная теория вероятностей и статистика»

I. Тема: Реализация на ЭВМ алгоритмов описательной статистики.

II. Исходные данные: выборка значений ТМП объемом 100 элементов и уровень значимости по вариантам.

Вариант 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1,03 | 1,24 | 1,19 | 1,32 | 1,30 | 1,01 | 1,24 | 1,18 | 1,33 | 1,15 |
| 1,53 | 1,39 | 1,15 | 1,37 | 1,13 | 1,50 | 1,14 | 1,14 | 1,35 | 1,26 |
| 1,59 | 1,13 | 1,58 | 1,36 | 1,30 | 1,59 | 1,14 | 1,27 | 1,38 | 1,37 |
| 1,33 | 1,18 | 1,31 | 1,31 | 1,34 | 1,35 | 1,15 | 1,33 | 1,33 | 1,17 |
| 1,30 | 1,36 | 1,45 | 1,31 | 1,17 | 1,30 | 1,40 | 1,44 | 1,30 | 1,25 |
| 1,32 | 1,21 | 1,40 | 1,28 | 1,25 | 1,33 | 1,21 | 1,39 | 1,29 | 1,34 |
| 1,25 | 1,14 | 1,37 | 1,36 | 1,34 | 1,25 | 1,16 | 1,40 | 1,34 | 1,26 |
| 1,24 | 1,23 | 1,28 | 1,35 | 1,24 | 1,25 | 1,21 | 1,28 | 1,35 | 1,23 |
| 1,36 | 1,19 | 1,35 | 1,21 | 1,22 | 1,40 | 1,35 | 1,31 | 1,20 | 1,23 |
| 1,15 | 1,33 | 1,30 | 1,31 | 1,24 | 1,34 | 1,20 | 1,34 | 1,30 | 1,35 |

Уровень значимости α = 0,025.

III. Выполнить:

1. Построить две гистограммы распределения с количеством интервалов *q*min+1 и с количеством интервалов *q*max – 1, где

*q*min = 0,55 *N*0,4 и *q*max = 1,25 *N*0,4,

*N* = 100 – объем исходных данных.

Сделать вывод о том, как влияет количество интервалов на форму гистограммы.

1. Получить основные статистические характеристики исходного ряда: математическое ожидание, медиану, моду, СКО, дисперсию, коэффициент асимметрии, коэффициент эксцесса, стандартную ошибку среднего. Представить α-процентные доверительные интервалы для среднего.
2. Проверить гипотезу о нормальности распределения с помощью критерия согласия χ2 при заданном уровне значимости. Сделать вывод.
3. Номер варианта соответствует порядковому номеру в списке учебной группы.

Номер варианта (C) = 2.

1. **Код программы расчета статистических характеристик наблюдаемого ряда и проверки гипотезы о нормальном распределении**

1. ./DataSetMetrics.cs

﻿**using** Lab4;

**namespace** Core

{

**public** **class** DataSetMetrics

{

*/// <summary>*

*/// Кол-во наблюдений*

*/// </summary>*

**public** **double** N { **get**; **set**; }

*/// <summary>*

*/// Размах*

*/// </summary>*

**public** **double** R { **get**; **set**; }

*/// <summary>*

*/// Математическое ожидание*

*/// </summary>*

**public** **double** ExpectValue { **get**; **internal** **set**; }

*/// <summary>*

*/// Мода*

*/// </summary>*

**public** **double** Mode { **get**; **internal** **set**; }

*/// <summary>*

*/// Медиана*

*/// </summary>*

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

*/// <summary>*

*/// Статистика сегментированная (распреление для гистограммы)*

*/// </summary>*

**public** Dictionary<**int**, SegmentStatistics> QSegmentStatistics { **get**; **set**; } = new();

*/// <summary>*

*/// Дисперсия*

*/// </summary>*

**public** **double** D { **get**; **internal** **set**; }

*/// <summary>*

*/// СКО*

*/// </summary>*

**public** **double** Sigma => Math.Sqrt(D);

*/// <summary>*

*/// Коэффициент ассиметрии*

*/// </summary>*

**public** **double** Skewness { **get**; **internal** **set**; }

*/// <summary>*

*/// Коэффициент эксцесса*

*/// </summary>*

**public** **double** ExcessKurtosis { **get**; **internal** **set**; }

*/// <summary>*

*/// Стандартная ошибка среднего*

*/// </summary>*

**public** **double** StandardError { **get**; **internal** **set**; }

*/// <summary>*

*/// Получить квантиль распределения Стьюдента*

*/// </summary>*

*/// <param name="a">Уровень значимости α</param>*

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

**public** **double** GetTValue(**double** a)

{

**var** v = N - 1;

**return** Student.GetT(a, v);

}

}

}

2. ./Metrics.cs

﻿**namespace** Core

{

**public** **class** Metrics

{

**public** **static** DataSetMetrics CountMetrics(

**double**[] values,

**int**[] qValues)

{

**var** ret = new DataSetMetrics();

ret.N = values.Length;

ret.R = values.Max() - values.Min();

ret.ExpectValue = values.Average();

**var** sortedValues = values.ToList();

sortedValues.Sort();

**if** (sortedValues.Count % 2 == 0)

{

ret.Median = (sortedValues[sortedValues.Count / 2 - 1]

+ sortedValues[sortedValues.Count / 2]) / 2;

}

**else**

{

ret.Median = sortedValues[sortedValues.Count / 2 - 1];

}

**foreach** (**var** q **in** qValues)

{

**var** statistics = BuildSegmentStatistics(values, q);

ret.QSegmentStatistics.**Add**(q, statistics);

}

**if** (ret.QSegmentStatistics.Count != 0)

{

**var** elderStatistics = ret.QSegmentStatistics

.MaxBy(p => p.Key);

**var** mainSegmentValues = elderStatistics.**Value**.Parts

.MaxBy(p => p.**Value**.Values.Length)

.**Value**.Values;

**if** (mainSegmentValues.Length != 0)

{

ret.Mode = mainSegmentValues.Order().ToArray()[mainSegmentValues.Length / 2];

}

}

ret.D = values

.**Select**(v => Math.Pow(v - ret.ExpectValue, 2))

.Sum() / (values.Length - 1);

ret.Skewness = GetCentralMoment(values, 3) /

Math.Pow(ret.Sigma, 3);

ret.ExcessKurtosis = (GetCentralMoment(values, 4) /

Math.Pow(ret.Sigma, 4)) - 3;

ret.StandardError = ret.Sigma / Math.Sqrt(values.Length);

**return** ret;

}

**private** **static** SegmentStatistics BuildSegmentStatistics(

**double**[] values,

**double** q)

{

**var** max = values.Max();

**var** min = values.Min();

**var** r = max - min;

**var** delta = r / q;

**var** ret = new SegmentStatistics();

**for** (**int** i = 1; i <= q; i++)

{

**var** **from** = min + (i - 1) \* delta;

**var** to = min + i \* delta;

**var** part = new SegmentStatisticsPart();

part.**From** = **from**;

part.To = to;

part.Values = values

.**Where**(v => v >= **from**

&& (v < to

|| v == to

&& i == q))

.ToArray();

ret.Parts.**Add**(part.**From**, part);

}

ret.**From** = min;

ret.To = max;

**return** ret;

}

*/// <summary>*

*/// Получить n-ый центральный момент*

*/// </summary>*

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

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

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

**public** **static** **double** GetCentralMoment(

**double**[] values,

**double** number)

{

**decimal** sum = 0;

**decimal** count = values.Length;

**var** expect = values.Average();

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

{

sum += (**decimal**)Math.Pow((**value** - expect), number);

}

**return** (**double**)(sum / count);

}

}

}

3. ./PlotBuilder.cs

﻿**using** ScottPlot;

**namespace** Core

{

**public** **class** PlotBuilder

{

**public** **static** Plot BuildGistogram(

SegmentStatistics segmentStatistics)

{

**var** plot = new Plot();

**var** bars = new List<Bar>();

**var** ticks = new List<Tick>();

**int** cnt = 0;

**foreach** (**var** (**from**, part) **in** segmentStatistics.Parts)

{

cnt++;

**var** x = part.**From** + (part.To - part.**From**) / 2;

**var** bar = new Bar()

{

Position = cnt,

**Value** = part.Values.Length,

ValueBase = 0,

Orientation = Orientation.Vertical,

*//Size = 0*

};

ticks.**Add**(new(cnt, cnt.ToString() + "**\r\n**" + bar.**Value**.ToString()));

bars.**Add**(bar);

}

plot.Axes.Bottom.TickGenerator = new ScottPlot.TickGenerators.NumericManual(ticks.ToArray());

plot.Axes.Bottom.MajorTickStyle.Length = 0;

plot.HideGrid();

*// tell the plot to autoscale with no padding beneath the bars*

plot.Axes.Margins(bottom: 0);

**var** barsPlot = plot.**Add**.Bars(bars);

**return** plot;

}

}

}

4. ./PlotDescriptor.cs

﻿**using** ScottPlot;

**using** System.Reflection;

**namespace** Core

{

**public** **class** PlotDescriptor

{

**public** List<PlotSegment> Segments { **get**; **set**; } = new();

**public** **double** MinX { **get**; **set**; } = 0;

**public** **double** MaxX { **get**; **set**; } = 1;

**public** List<Coordinates> GetCoords(**double** fromX, **double** toX, **double** step = 0.01)

{

**var** ret = new SortedList<**double**, Coordinates>();

**foreach** (**var** segment **in** Segments)

{

**var** segmentFromX = Math.Max(fromX, segment.MinX);

**var** segmentToX = Math.Min(toX, segment.MaxX);

**double** x = segmentFromX;

**for** (**int** i = 0; x <= segmentToX ; i++)

{

x = segmentFromX + (i \* step);

**var** coords = new Coordinates(x, segment.Func(x));

ret.**Add**(x, coords);

}

}

**return** ret.ToList().ConvertAll(v => v.**Value**);

}

}

}

5. ./PlotSegment.cs

﻿**namespace** Core

{

*/// <summary>*

*/// Segment of plot*

*/// </summary>*

**public** **class** PlotSegment

{

**public** Func<**double**, **double**> Func { **get**; **set**; } = (x) => x;

**public** **double** MinX { **get**; **set**; } = **double**.MinValue;

**public** **double** MaxX { **get**; **set**; } = **double**.MaxValue;

**public** **double** Step { **get**; **set**; } = 0.01;

**public** List<**double**> GetXCoords()

{

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

**for** (**double** i = MinX; i <= MaxX; i += Step)

{

ret.**Add**(i);

}

**return** ret;

}

**public** List<**double**> GetYCoords()

{

**var** ret = GetXCoords().ConvertAll(x => Func(x));

**return** ret;

}

**public** **bool** HasMapping(**double** x) => x <= MaxX && x >= MinX;

}

}

6. ./Program.cs

﻿*// See https://aka.ms/new-console-template for more information*

**using** Core;

**using** ScottPlot;

Console.WriteLine("Hello, World!");

Plot myPlot = new();

**var** plotDescriptor = new PlotDescriptor();

**var** plotSegment = new PlotSegment()

{

MinX = 0,

MaxX = 1

};

plotDescriptor.Segments.**Add**(plotSegment);

plotDescriptor.Segments.**Add**(new PlotSegment()

{

Func = (x) => 0,

MaxX = -0.001

});

plotDescriptor.Segments.**Add**(new PlotSegment()

{

Func = (x) => 0,

MinX = 1.001

});

**var** values = plotDescriptor.GetCoords(-0.1, 1.1, 0.001);

**var** scatter = myPlot.**Add**.Scatter(

values);

scatter.LineStyle.Width = 5;

scatter.MarkerStyle.Size = 5;

**var** fileName = "demo.png";

myPlot.SavePng(fileName, 800, 600);

Utils.OpenPath(fileName);

7. ./SegmentStatistics.cs

﻿**namespace** Core

{

*/// <summary>*

*/// эмпирическая статистика для построения гистограммы и вычисления моды*

*/// </summary>*

**public** **class** SegmentStatistics

{

**public** **int** PartsNumber { **get** => Parts.Count; }

**public** **double** **From** { **get**; **set**; }

**public** **double** To { **get**; **set**; }

**public** SortedList<**double**, SegmentStatisticsPart> Parts { **get**; **set**; } = new();

**public** **int** ValuesCount => Parts.Sum(p => p.**Value**.Values.Length);

}

}

8. ./SegmentStatisticsPart.cs

﻿**namespace** Core

{

**public** **class** SegmentStatisticsPart

{

**public** **double** **From** { **get**; **set**; }

**public** **double** To { **get**; **set**; }

**public** **double**[] Values { **get**; **set**; } = Array.Empty<**double**>();

}

}

9. ./Student.cs

﻿**namespace** Lab4

{

*/// <summary>*

*/// Code from https://www.meracalculator.com/math/t-distribution-critical-value-table.php*

*/// </summary>*

**public** **class** Student

{

*//easyRoundOf(resConvert(pStuT(bhname, df2)), 4);*

**public** **static** **double** GetT(**double** a, **double** v)

{

**return** resConvert(pStuT(a, v));

}

**public** **static** **double** stuT(**double** a, **double** b)

{

a = Math.Abs(a);

**var** c = a / Math.Sqrt(b);

**var** d = Math.Atan(c);

**if** (1 == b)

{

**return** d / (Math.PI / 2);

}

**var** e = Math.Sin(d);

**var** f = Math.Cos(d);

**var** alpha = b % 2 == 1

? 1 - (d + e \* f \* stuComp(f \* f, 2, b - 3, -1)) / (Math.PI / 2)

: 1 - e \* stuComp(f \* f, 1, b - 3, -1);

**return** 1.0 - alpha;

}

**public** **static** **double** stuComp(**double** a, **double** b, **double** c, **double** d)

{

**double** e = 1;

**var** f = e;

**var** g = b;

**for** (; g <= c;)

{

e = e \* a \* g / (g - d);

f += e;

g += 2;

}

**return** f;

}

**public** **static** **double** pStuT(**double** a, **double** b)

{

**var** c = .5;

**var** d = .5;

**double** e = 0;

**for** (; d > 1e-6;)

{

e = (**double**)1 / c - 1;

d /= 2;

**var** qt = 1 - stuT(e, b);

**if** (qt > a)

{

c -= d;

}

**else**

{

c += d;

}

}

**return** e;

}

**public** **static** **double** resConvert(**double** a)

{

*//var b;*

**return** a >= 0 ? a + 5e-4 : a - 5e-4;

}

**public** **static** **double** easyRoundOf(**double** a, **int** b)

{

*/\*if (isNaN(a))*

*return 0;\*/*

**var** level = Math.Pow(10, b);

**var** c = Math.Round(a \* level) / level;

**return** c;

}

*/\*function parseConv(a)*

*{*

*return parseFloat(a)*

*}\*/*

}

}

10. ./Utils.cs

﻿**using** Markdig;

**using** ScottPlot;

**using** System.Diagnostics;

**using** System.Text;

**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));

}

**public** **static** **string** BuildSegmentStatTableHtml(SegmentStatistics segmentStatistics)

{

MarkdownPipeline pipeline = new MarkdownPipelineBuilder()

.UseAdvancedExtensions()

.Build();

**var** markdown = new StringBuilder();

markdown.AppendLine("| Номер интервала | x j | x j**\\**+1 | n j |");

markdown.AppendLine("|------------------|------|--------|-------|");

**int** cnt = 0;

**foreach** (**var** segment **in** segmentStatistics.Parts)

{

cnt++;

markdown.AppendLine($"| {cnt} | {Math.Round(segment.Value.From, 3)} | " +

$"{Math.Round(segment.Value.To, 3)} | {segment.Value.Values.Length} |");

}

markdown.AppendLine($"| Сумма | | | {segmentStatistics.ValuesCount} |");

**return** Markdown.ToHtml(markdown.ToString(), pipeline);

}

}

}

11. ./Lab4.cs

﻿**using** Core;

**using** Markdig;

**using** System.Text;

**using** System.Web;

**using** MathNet.Numerics.Distributions;

**using** Microsoft.Win32.SafeHandles;

**namespace** Lab4

{

**internal** **class** Lab4

{

**private** **const** **string** SOURCE\_FILENAME = "sourceData.txt";

**private** **const** **string** REPORT\_FILENAME = "report4.html";

**public** **static** **readonly** **double**[] \_sourceData = Utils.ReadValues(SOURCE\_FILENAME)

.GetAwaiter()

.GetResult();

**private** **const** **string** TMP\_DIRECTORY\_NAME = "tmp";

**private** **const** **double** A = 0.025;

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

{

PrepareEnvironment();

**var** metrics = Metrics.CountMetrics(

\_sourceData,

[(**int**)Math.Round(0.55 \* Math.Pow(\_sourceData.Length, 0.4) + 1),

(**int**)Math.Round(1.25 \* Math.Pow(\_sourceData.Length, 0.4) - 1)]);

**await** CreateReport(metrics);

}

**private** **static** **async** Task CreateReport(DataSetMetrics metrics)

{

**await** File.WriteAllLinesAsync(

REPORT\_FILENAME,

[@"

<style>

table, th, td {

border: 1px solid black;

}

</style>"]);

**var** html = new StringBuilder();

html.AppendLine(Markdown.ToHtml("# Лабораторная работа №4"));

html.AppendLine(Markdown.ToHtml("# 1. Гистограмма"));

**foreach** (**var** (order, stat) **in** metrics.QSegmentStatistics)

{

**var** fileName = BuildBars(stat);

html.AppendLine(Markdown.ToHtml($"## q = {order}"));

html.AppendLine(Utils.BuildSegmentStatTableHtml(stat));

html.AppendLine(

HttpUtility.UrlDecode(

Markdown.ToHtml(

$"![График]({fileName} **\"**График q = {order}**\"**)")));

}

**var** staticticsHtml = BuildStatisticsHtml(metrics);

html.AppendLine(Markdown.ToHtml("# 2. Основные статистические характеристики"));

html.AppendLine(staticticsHtml);

html.AppendLine(Markdown.ToHtml("# 3. Проверка гипотезы о нормальности распределения"));

html.AppendLine(BuildNormHtmlTable(metrics));

html.Append(Markdown.ToHtml("По таблице квантилей 𝜒2 распределения, " +

"при заданном уровне значимости α=0,025 " +

"и числе степеней свободы 𝜈=𝑘−𝑟−1=7−2−1=4, критическое значение = 11.143. " +

"Оснований для отклонения гипотезы нет."));

**await** File.AppendAllLinesAsync(

REPORT\_FILENAME,

[html.ToString()]);

Utils.OpenPath(REPORT\_FILENAME);

}

**private** **static** **string** BuildStatisticsHtml(DataSetMetrics metrics)

{

**var** markdown = new StringBuilder();

markdown.AppendLine($"+ Мат. ожидание {metrics.ExpectValue}");

markdown.AppendLine($"+ Медиана {metrics.Median}");

markdown.AppendLine($"+ Мода {metrics.Mode}");

markdown.AppendLine($"+ Дисперсия {metrics.D}");

markdown.AppendLine($"+ Ср. кв. откл. {metrics.Sigma}");

markdown.AppendLine($"+ Коэффициент ассиметрии {metrics.Skewness}");

markdown.AppendLine($"+ Коэффициент эксцесса {metrics.ExcessKurtosis}");

markdown.AppendLine($"+ Ст. ошибка среднего {metrics.StandardError}");

markdown.AppendLine($"+ Доверительный интервал для среднего, a = " +

$"{A}: {metrics.ExpectValue} ± {metrics.GetTValue(A) \* metrics.StandardError}");

**return** Markdown.ToHtml(markdown.ToString());

}

**private** **static** MarkdownPipeline pipeline = new MarkdownPipelineBuilder()

.UseAdvancedExtensions()

.Build();

**private** **static** **string** BuildNormHtmlTable(DataSetMetrics metrics)

{

**var** markdown = new StringBuilder();

**var** segments = metrics.QSegmentStatistics.MaxBy(kv => kv.Key).**Value**;

*//SegmentStatisticsPart? prevPart = null;*

markdown.AppendLine("|№|x j|x j+1|nj|Ф(x)| Ф(x+1)|pj|N \* pj|(n\_j-N∙p\_j )^2|(n\_j-N∙p\_j )^2/(N∙p\_j )|");

markdown.AppendLine("|---|---|---|---|---|---|---|---|---|---|");

Func<**double**, **double**> norm = **value** => Norm(**value**, metrics.ExpectValue, metrics.Sigma);

**int** cnt = 0;

**double** countedSum = 0;

**foreach** (**var** (\_, part) **in** segments.Parts)

{

cnt++;

**var** fFrom = cnt == 1 ? 0 : norm(part.**From**);

**var** fTo = cnt == segments.Parts.Count ? 1 : norm(part.To);

**var** diff = fTo - fFrom;

**var** counted1 = Math.Pow(part.Values.Length - segments.ValuesCount \* diff, 2);

**var** counted2 = counted1 / (segments.ValuesCount \* diff);

markdown.Append($"|{cnt.ToString("0.0000")}|{part.From}|{part.To}|{part.Values.Length}");

markdown.Append($"|{fFrom}|{fTo}|{diff}|{diff \* segments.ValuesCount}");

markdown.AppendLine($"|{counted1}|{counted2}|");

countedSum += counted2;

}

markdown.Append($"|Сумма|-|-|{segments.ValuesCount}");

markdown.Append($"|-|-|1|{ segments.ValuesCount}");

markdown.AppendLine($"|χ02=|{countedSum}|");

**return** Markdown.ToHtml(markdown.ToString(), pipeline);

}

*/// <summary>*

*/// Нормальная функция распределения*

*/// </summary>*

*/// <param name="X">Значение</param>*

*/// <param name="M">Математическое ожидание</param>*

*/// <param name="S">Среднее квадратное отклонение</param>*

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

**private** **static** **double** Norm(**double** X, **double** M = 0, **double** S = 1)

{

**return** Normal.CDF(M, S, X);

}

**private** **static** **string** BuildBars(SegmentStatistics statistics)

{

**var** plot = PlotBuilder.BuildGistogram(statistics);

**var** fileName = Path.GetRandomFileName() + ".png";

fileName = Path.Combine(TMP\_DIRECTORY\_NAME, fileName);

plot.SavePng(fileName, 800, 600);

**return** fileName;

}

**private** **static** **void** PrepareEnvironment()

{

**if** (Directory.Exists(TMP\_DIRECTORY\_NAME))

{

Directory.Delete(TMP\_DIRECTORY\_NAME, **true**);

}

**if** (!Directory.Exists(TMP\_DIRECTORY\_NAME))

{

Directory.CreateDirectory(TMP\_DIRECTORY\_NAME);

}

}

}

}

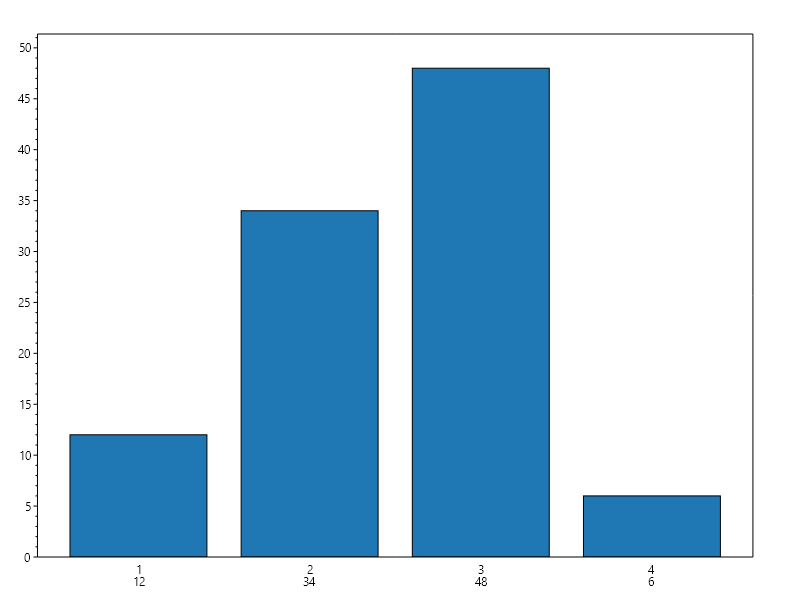
1. Результаты

# Лабораторная работа №4

# 1. Гистограмма

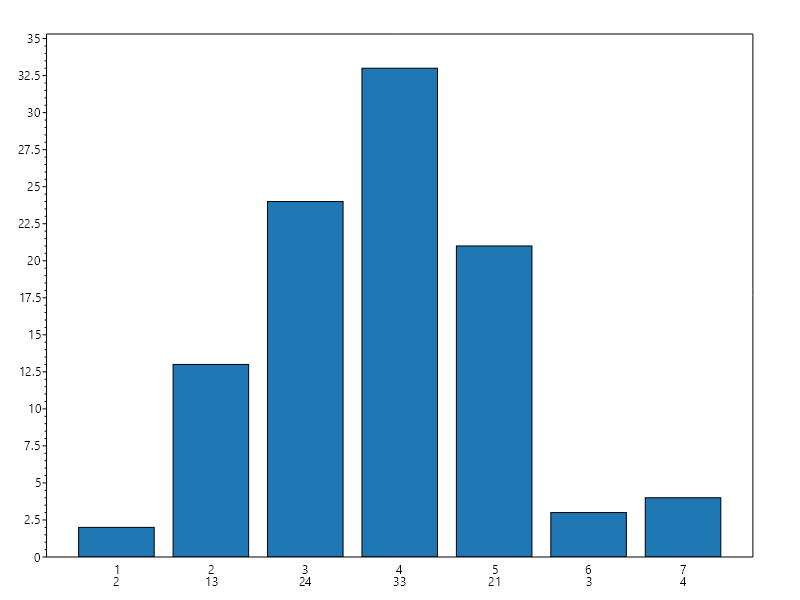
## q = 4

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | 1,01 | 1,155 | 12 |
| 2 | 1,155 | 1,3 | 34 |
| 3 | 1,3 | 1,445 | 48 |
| 4 | 1,445 | 1,59 | 6 |
| Сумма |  |  | 100 |



## q = 7

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | 1,01 | 1,093 | 2 |
| 2 | 1,093 | 1,176 | 13 |
| 3 | 1,176 | 1,259 | 24 |
| 4 | 1,259 | 1,341 | 33 |
| 5 | 1,341 | 1,424 | 21 |
| 6 | 1,424 | 1,507 | 3 |
| 7 | 1,507 | 1,59 | 4 |
| Сумма |  |  | 100 |



# 2. Основные статистические характеристики

* Мат. ожидание 1,2897
* Медиана 1,3
* Мода 1,31
* Дисперсия 0,011307989898989913
* Ср. кв. откл. 0,10633903281011123
* Коэффициент ассиметрии 0,2964084524320268
* Коэффициент эксцесса 0,7648234341195406
* Ст. ошибка среднего 0,010633903281011122
* Доверительный интервал для среднего, a = 0,025: 1,2897 ± 0,024208096500406874

# 3. Проверка гипотезы о нормальности распределения

| **№** | **x j** | **x j+1** | **nj** | **Ф(x)** | **Ф(x+1)** | **pj** | **N \* pj** | **(n\_j-N∙p\_j )^2** | **(n\_j-N∙p\_j )^2/(N∙p\_j )** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1,00000 | 1,01000 | 1,09286 | 2,00000 | 0,00000 | 0,03208 | 0,03208 | 3,20785 | 1,45890 | 0,45479 |
| 2,00000 | 1,09286 | 1,17571 | 13,00000 | 0,03208 | 0,14188 | 0,10980 | 10,98021 | 4,07956 | 0,37154 |
| 3,00000 | 1,17571 | 1,25857 | 24,00000 | 0,14188 | 0,38486 | 0,24298 | 24,29839 | 0,08904 | 0,00366 |
| 4,00000 | 1,25857 | 1,34143 | 33,00000 | 0,38486 | 0,68668 | 0,30181 | 30,18113 | 7,94603 | 0,26328 |
| 5,00000 | 1,34143 | 1,42429 | 21,00000 | 0,68668 | 0,89718 | 0,21050 | 21,05012 | 0,00251 | 0,00012 |
| 6,00000 | 1,42429 | 1,50714 | 3,00000 | 0,89718 | 0,97956 | 0,08239 | 8,23861 | 27,44307 | 3,33103 |
| 7,00000 | 1,50714 | 1,59000 | 4,00000 | 0,97956 | 1,00000 | 0,02044 | 2,04369 | 3,82714 | 1,87266 |
| Сумма | - | - | 100,00000 | - | - | 1 | 100,00000 | χ02= | 6,29708 |

По таблице квантилей 𝜒2 распределения, при заданном уровне значимости α=0,025 и числе степеней свободы 𝜈=𝑘−𝑟−1=7−2−1=4, критическое значение = 11.143. Оснований для отклонения гипотезы нет.

**Выводы**

В ходе выполнения четвертой лабораторной работы произведена реализация на ЭВМ алгоритмов описательной статистики.

Определены статистические показатели набора данных в соответствии с вариантом.

Выполнена проверка гипотезы о нормальности распределения с помощью критерия согласия χ2 при заданном уровне значимости α.

Написана программа для выполнения операций в соответствии с заданием.

Приобретены навыки по реализации на ЭВМ алгоритмов описательной статистики.