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 № 2 |
| Моделирование на ЭВМ случайных величин RAND и GAUSS |
| 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 № 2 по дисциплине

«Прикладная теория вероятностей и статистика»

I. ТЕМА: Моделирование на ЭВМ случайных величин RAND и GAUSS.

1. Смоделировать измерения телеметрируемого параметра, распределенного по равномерному закону , (где = 1, 2, … – номер варианта), при объеме выборки  и . Построить гистограммы распределений, вычислить основные статистические характеристики (математическое ожидание, мода, медиана, дисперсия, среднее квадратическое отклонение).
2. Смоделировать измерения телеметрируемого параметра, распределенного по нормальному закону , (где – номер варианта), при объеме выборки и . Построить гистограммы распределений, вычислить основные статистические характеристики.
3. Номер варианта *N* соответствует порядковому номеру в списке учебной группы.

Номер варианта N = 2.

1. **Код программы моделирования равномерно распределённой случайной величины и записи ее в файл**

1. ./Lab2.cs

﻿

**using** Core;

**using** Lab2.Processing;

**using** Markdig;

**using** ScottPlot;

**using** ScottPlot.Colormaps;

**using** System.Runtime.CompilerServices;

**using** System.Text;

**using** System.Web;

**namespace** Lab2

{

**internal** **class** Lab2

{

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

**private** **const** **string** UNIFORM\_100\_FILENAME = $"2\_uniform\_100.txt";

**private** **const** **string** UNIFORM\_1000\_FILENAME = $"2\_uniform\_1000.txt";

**private** **const** **string** GAUSS\_100\_FILENAME = $"2\_gauss\_100.txt";

**private** **const** **string** GAUSS\_1000\_FILENAME = $"2\_gauss\_1000.txt";

**private** **static** **readonly** **string** REPORT\_FILENAME = "report.html";

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

**private** **static** **readonly** **double** \_a = (**double**)-VARIANT / 10;

**private** **static** **readonly** **double** \_b = (**double**)VARIANT / 2;

**private** **static** **readonly** **double** \_m = (**double**)VARIANT;

**private** **static** **readonly** **double** \_sigma = (**double**)VARIANT / 3;

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

{

**await** PrepareDataSet();

**var** uniform100Values = **await** Utils.ReadValues(UNIFORM\_100\_FILENAME);

**var** uniform1000Values = **await** Utils.ReadValues(UNIFORM\_1000\_FILENAME);

**var** gauss100Values = **await** Utils.ReadValues(GAUSS\_100\_FILENAME);

**var** gauss1000Values = **await** Utils.ReadValues(GAUSS\_1000\_FILENAME);

**await** PrepareFileEnvironment();

**await** File.AppendAllLinesAsync(REPORT\_FILENAME, [Markdown.ToHtml($"# Лабораторная работа №2")]);

**await** File.AppendAllLinesAsync(REPORT\_FILENAME, [Markdown.ToHtml($"## Равномерное распределение, l = 100")]);

**await** ProcessValues(uniform100Values);

**await** File.AppendAllLinesAsync(REPORT\_FILENAME, [Markdown.ToHtml($"## Равномерное распределение, l = 1000")]);

**await** ProcessValues(uniform1000Values);

**await** File.AppendAllLinesAsync(REPORT\_FILENAME, [Markdown.ToHtml($"## Нормальное распределение, l = 100")]);

**await** ProcessValues(gauss100Values);

**await** File.AppendAllLinesAsync(REPORT\_FILENAME, [Markdown.ToHtml($"## Нормальное распределение, l = 1000")]);

**await** ProcessValues(gauss1000Values);

Utils.OpenPath(REPORT\_FILENAME);

}

**private** **static** **async** Task PrepareFileEnvironment()

{

**if** (File.Exists(REPORT\_FILENAME))

{

File.Delete(REPORT\_FILENAME);

}

**await** File.WriteAllLinesAsync(REPORT\_FILENAME,

[@"

<style>

table, th, td {

border: 1px solid black;

}

</style>"]);

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

{

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

}

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

{

Directory.CreateDirectory(TMP\_DIRECTORY\_NAME);

}

}

**private** **static** **async** Task ProcessValues(

**double**[] values,

**string** name = "График")

{

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

markdown.AppendLine($"{name}");

**var** stats = Metrics.CountMetrics(values, [5, 7]);

**var** segmentStatistics = stats.QSegmentStatistics[5];

markdown.AppendLine(Markdown.ToHtml($"### Статистические показатели"));

markdown.AppendLine(BuildStatisticsHtml(stats));

**string** fileName = BuildBars(segmentStatistics);

markdown.AppendLine(Markdown.ToHtml($"### q = 5"));

markdown.AppendLine(BuildTableHtml(segmentStatistics));

markdown.AppendLine(HttpUtility.UrlDecode(Markdown.ToHtml($"![График]({fileName} **\"**График q = 5**\"**)")));

segmentStatistics = stats.QSegmentStatistics[7];

fileName = BuildBars(segmentStatistics);

markdown.AppendLine(Markdown.ToHtml($"### q = 7"));

markdown.AppendLine(BuildTableHtml(segmentStatistics));

markdown.AppendLine(HttpUtility.UrlDecode(Markdown.ToHtml($"![График]({fileName} **\"**График q = 7**\"**)")));

**var** html = markdown.ToString();

**await** File.AppendAllLinesAsync(REPORT\_FILENAME, [html]);

}

**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}");

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

}

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

.UseAdvancedExtensions()

.Build();

**private** **static** **string** BuildTableHtml(SegmentStatistics segmentStatistics)

{

**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);

}

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

{

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

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

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

fileName.Replace(Path.DirectorySeparatorChar, Path.AltDirectorySeparatorChar);

plot.SavePng(fileName, 800, 600);

**return** fileName;

}

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

{

**if** (!File.Exists(UNIFORM\_100\_FILENAME))

{

**double**[] values = PrepareUniformData(100, \_a, \_b);

**await** Utils.SaveValues(UNIFORM\_100\_FILENAME, values);

}

**if** (!File.Exists(UNIFORM\_1000\_FILENAME))

{

**double**[] values = PrepareUniformData(1000, \_a, \_b);

**await** Utils.SaveValues(UNIFORM\_1000\_FILENAME, values);

}

**if** (!File.Exists(GAUSS\_100\_FILENAME))

{

**double**[] values = PrepareGaussData(100, \_m, \_sigma);

**await** Utils.SaveValues(GAUSS\_100\_FILENAME, values);

}

**if** (!File.Exists(GAUSS\_1000\_FILENAME))

{

**double**[] values = PrepareGaussData(1000, \_m, \_sigma);

**await** Utils.SaveValues(GAUSS\_1000\_FILENAME, values);

}

}

*/// <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>*

**private** **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>*

**private** **static** **double** UniformLinearTransform(

**double** **value**,

**double** a,

**double** b)

{

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

}

**private** **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>*

**private** **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>*

**private** **static** **double** CreateGaussValue(

**double** uniformValue1,

**double** uniformValue2)

{

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

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

}

}

}

2. ./Metrics.cs

﻿**using** Lab2.Processing;

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

**using** System.Net.Sockets;

**namespace** Lab2

{

**internal** **class** Metrics

{

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

**double**[] values,

**int**[] qValues)

{

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

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);

**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;

}

}

}

3. ./Processing/DataSetMetrics.cs

﻿**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

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

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** Lab2.Processing

{

**internal** **class** DataSetMetrics

{

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

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

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

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

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

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

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

}

}

4. ./Processing/PlotBuilder.cs

﻿**using** ScottPlot;

**namespace** Lab2.Processing

{

**internal** **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;

}

}

}

5. ./Processing/SegmentStatistics.cs

﻿**namespace** Lab2.Processing

{

*/// <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);

}

}

6. ./Processing/SegmentStatisticsPart.cs

﻿**namespace** Lab2.Processing

{

**public** **class** SegmentStatisticsPart

{

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

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

**public** **double**[] Values { **get**; **set**; } = new **double**[0];

}

}

7. ./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**);

}

}

}

8. ./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;

}

}

9. ./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();

}

}

}

1. Результаты работы программы

# Лабораторная работа №2

Вариант №2

a = -0,2, b = 1

M = 2, sigma = 0,6666666666666666

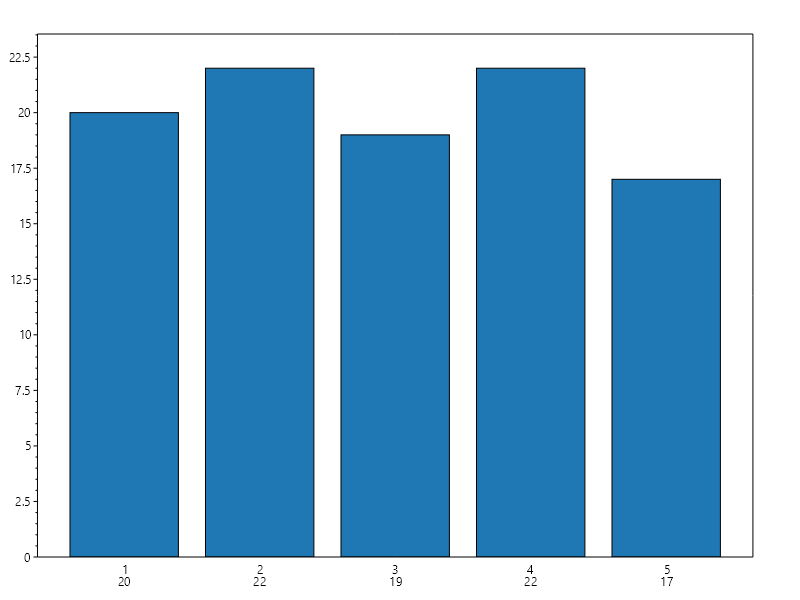
## Равномерное распределение, l = 100

### Статистические показатели

* Мат. ожидание 0,3978377152245035
* Медиана 0,4148912029788061
* Мода 0,7386979402256191
* Дисперсия 0,10377004187640673
* Ср. кв. откл. 0,3221335776916258

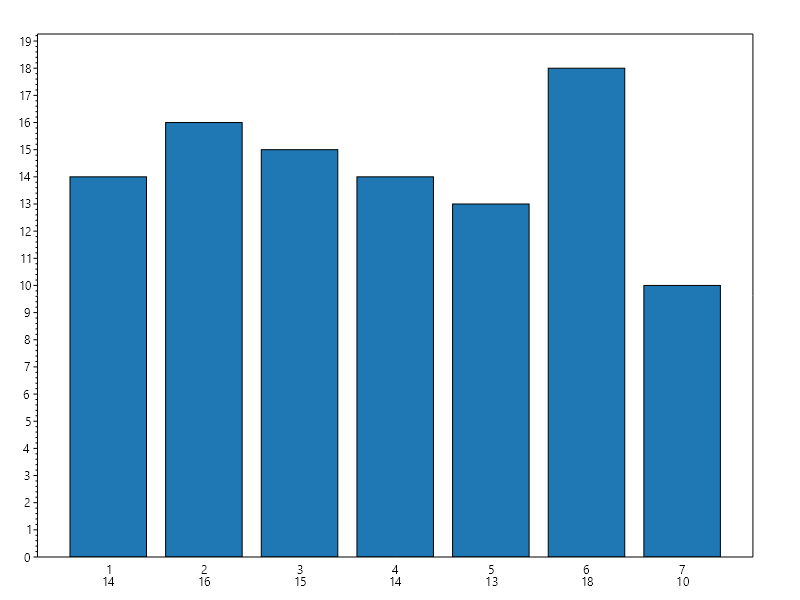
### q = 5

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | -0,151 | 0,075 | 20 |
| 2 | 0,075 | 0,301 | 22 |
| 3 | 0,301 | 0,527 | 19 |
| 4 | 0,527 | 0,753 | 22 |
| 5 | 0,753 | 0,979 | 17 |
| Сумма |  |  | 100 |



### q = 7

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | -0,151 | 0,011 | 14 |
| 2 | 0,011 | 0,172 | 16 |
| 3 | 0,172 | 0,333 | 15 |
| 4 | 0,333 | 0,495 | 14 |
| 5 | 0,495 | 0,656 | 13 |
| 6 | 0,656 | 0,818 | 18 |
| 7 | 0,818 | 0,979 | 10 |
| Сумма |  |  | 100 |



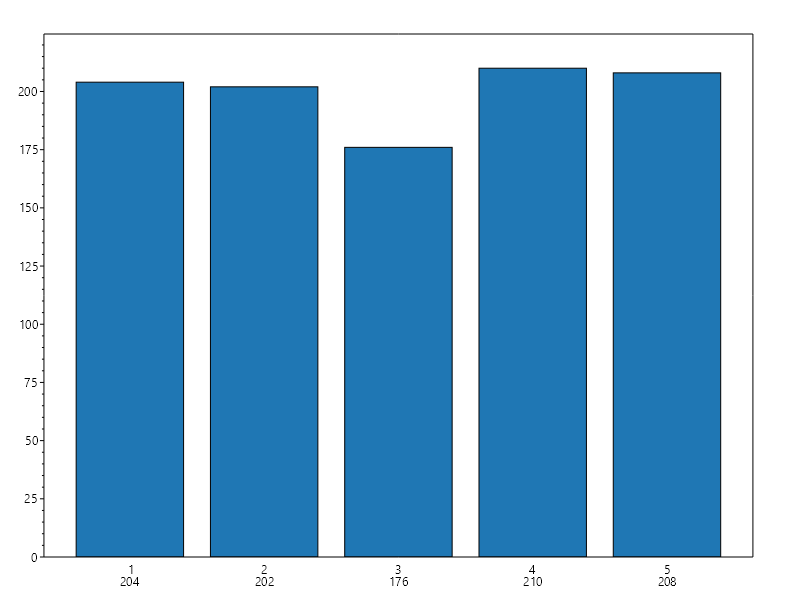
## Равномерное распределение, l = 1000

### Статистические показатели

* Мат. ожидание 0,4009632436171699
* Медиана 0,40908177768438686
* Мода -0,11053235372670894
* Дисперсия 0,1230736190679468
* Ср. кв. откл. 0,35081849875390947

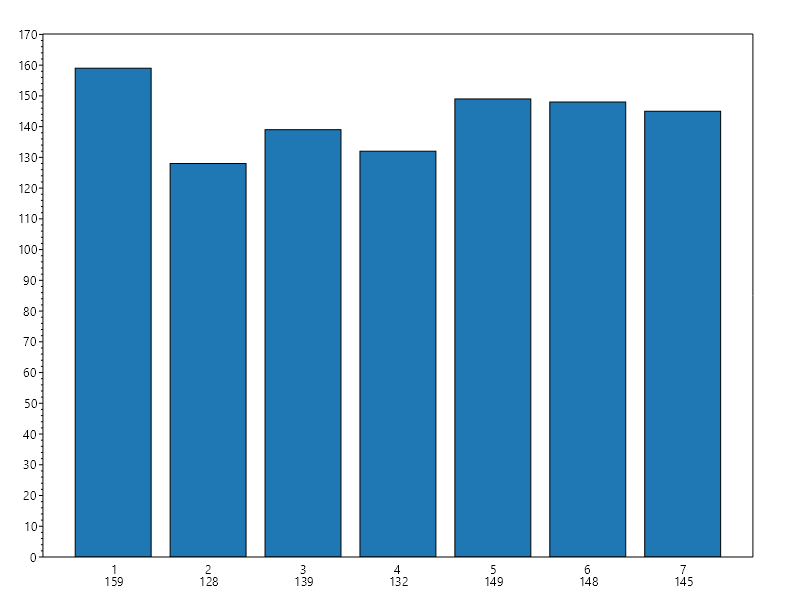
### q = 5

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | -0,2 | 0,04 | 204 |
| 2 | 0,04 | 0,279 | 202 |
| 3 | 0,279 | 0,519 | 176 |
| 4 | 0,519 | 0,758 | 210 |
| 5 | 0,758 | 0,998 | 208 |
| Сумма |  |  | 1000 |



### q = 7

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | -0,2 | -0,029 | 159 |
| 2 | -0,029 | 0,142 | 128 |
| 3 | 0,142 | 0,313 | 139 |
| 4 | 0,313 | 0,484 | 132 |
| 5 | 0,484 | 0,655 | 149 |
| 6 | 0,655 | 0,826 | 148 |
| 7 | 0,826 | 0,998 | 145 |
| Сумма |  |  | 1000 |



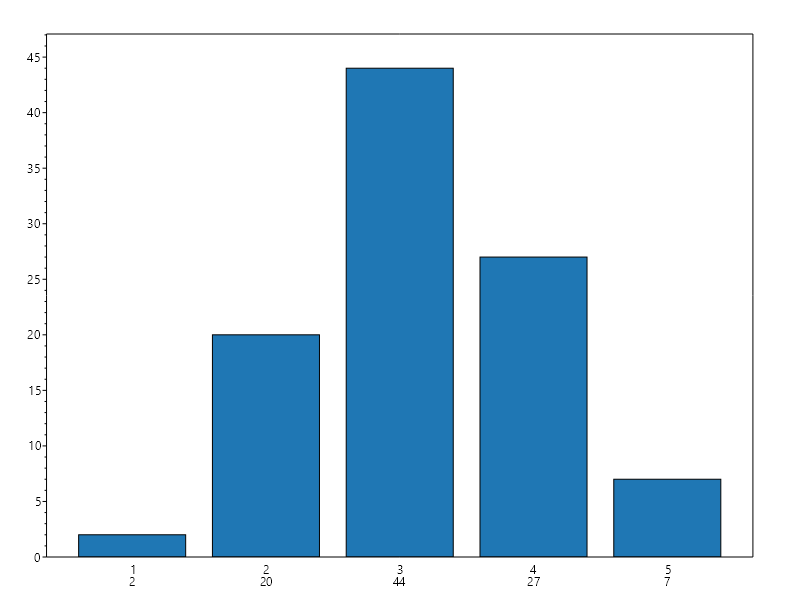
## Нормальное распределение, l = 100

### Статистические показатели

* Мат. ожидание 2,0373481776920865
* Медиана 2,083942592257844
* Мода 1,906614408971865
* Дисперсия 0,4962043897756014
* Ср. кв. откл. 0,7044177665104717

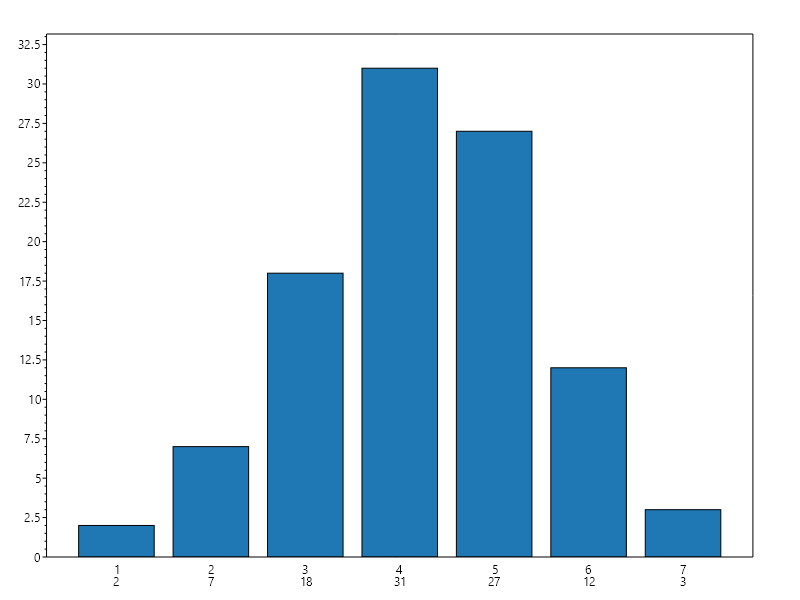
### q = 5

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | 0,03 | 0,784 | 2 |
| 2 | 0,784 | 1,537 | 20 |
| 3 | 1,537 | 2,29 | 44 |
| 4 | 2,29 | 3,044 | 27 |
| 5 | 3,044 | 3,797 | 7 |
| Сумма |  |  | 100 |



### q = 7

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | 0,03 | 0,568 | 2 |
| 2 | 0,568 | 1,106 | 7 |
| 3 | 1,106 | 1,645 | 18 |
| 4 | 1,645 | 2,183 | 31 |
| 5 | 2,183 | 2,721 | 27 |
| 6 | 2,721 | 3,259 | 12 |
| 7 | 3,259 | 3,797 | 3 |
| Сумма |  |  | 100 |



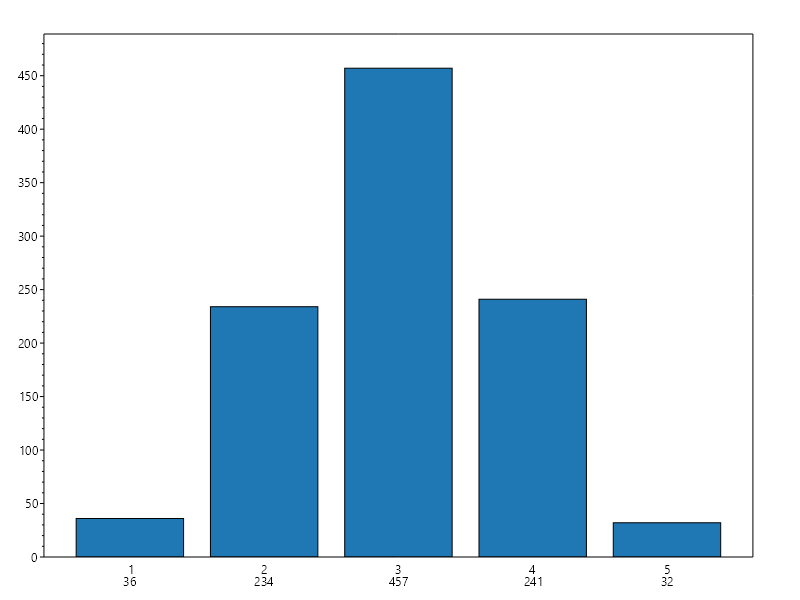
## Нормальное распределение, l = 1000

### Статистические показатели

* Мат. ожидание 2,014119539193223
* Медиана 2,045738085764583
* Мода 2,0356278014189004
* Дисперсия 0,44479150703841525
* Ср. кв. откл. 0,6669269128161011

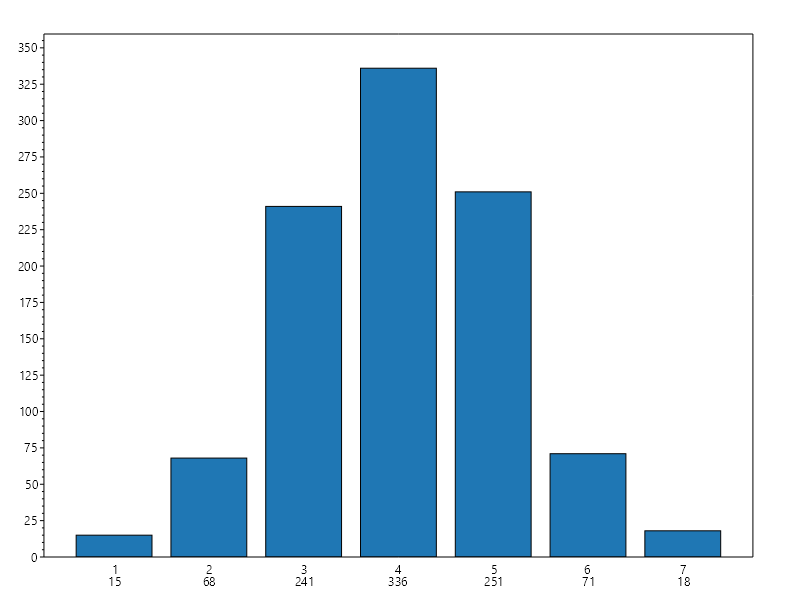
### q = 5

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | -0,061 | 0,762 | 36 |
| 2 | 0,762 | 1,586 | 234 |
| 3 | 1,586 | 2,41 | 457 |
| 4 | 2,41 | 3,233 | 241 |
| 5 | 3,233 | 4,057 | 32 |
| Сумма |  |  | 1000 |



### q = 7

| **Номер интервала** | **x j** | **x j+1** | **n j** |
| --- | --- | --- | --- |
| 1 | -0,061 | 0,527 | 15 |
| 2 | 0,527 | 1,115 | 68 |
| 3 | 1,115 | 1,704 | 241 |
| 4 | 1,704 | 2,292 | 336 |
| 5 | 2,292 | 2,88 | 251 |
| 6 | 2,88 | 3,468 | 71 |
| 7 | 3,468 | 4,057 | 18 |
| Сумма |  |  | 1000 |



**Выводы**

В ходе выполнения второй лабораторной работы произведено моделирование на ЭВМ случайных величин RAND и GAUSS.

Написана программа, вычисляющая статистические характеристики смоделированных наборов значений. Построены графики распределения случайных величин для l=100, l=1000, q = 5, q = 7.

Приобретены навыки по обработке наборов случайных значений на ЭВМ.