Министерство образования и науки РФ

Федеральное государственное образовательное учреждение

высшего образования

«Ярославский государственный технический университет»

Кафедра «Информационные системы и технологии»

Отчет защищен

с оценкой *\_\_\_\_\_\_\_\_*

Доцент, канд. ф-м. наук

\_\_\_\_\_\_\_\_\_\_\_\_ А.Б. Раухваргер

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021

**ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ**

по курсу “Методы исследования и моделирования информационных процессов и технологий”

ЯГТУ 09.04.02 – 001 ЛР

Отчет выполнил

студент гр. ЭМИС-14м \_\_\_\_\_\_\_\_\_ В.А. Матвеев

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2021

2021

Лабораторная работа №1

Задание: разработать в C# с использованием Windows Forms программу, моделирующую сложное событие, заданное формулой алгебры событий, через независимые события A, B, C, D, E.

Входные данные – вероятности исходных событий A, B, C, D, E и число испытаний.

Выходные данные – частоты и вероятность сложного события.

Заданное уравнение:

Краткое описание алгоритма:

Для получения вероятности сложного события необходимо преобразовать логическое выражение в арифметическое выражение согласно правилам теории вероятности, где:

Вероятность сложного события равна:

Чтобы получить частоту, необходимо для каждого события генерировать новое событие в диапазоне от 0 до 1 с помощью датчика случайных чисел.

Инструкция по работе с программой:

Для получения значений частоты и вероятности сложного события необходимо задать вероятности для каждого события A, B, C, D, E и количество испытаний. Получить результаты можно с кнопкой «Моделировать». Интерфейс программы представлен на рисунке 1.

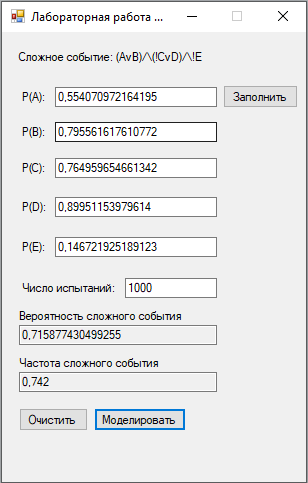


Рисунок 1 - Интерфейс программы

Код программы:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab1

{

public partial class Form1 : Form

{

double pa, pb, pc, pd, pe;

int step;

private void button1\_Click(object sender, EventArgs e) //Заполнить поля

{

Random ra = new Random();

Random pra = new Random(ra.Next(10000));

Random prb = new Random(ra.Next(10000));

Random prc = new Random(ra.Next(10000));

Random prd = new Random(ra.Next(10000));

Random pre = new Random(ra.Next(10000));

PA.Text = pra.NextDouble().ToString();

PB.Text = prb.NextDouble().ToString();

PC.Text = prc.NextDouble().ToString();

PD.Text = prd.NextDouble().ToString();

PE.Text = pre.NextDouble().ToString();

}

public Form1()

{

InitializeComponent();

}

public void SetValues(string pa, string pb, string pc, string pd, string pe, string step)

{

try

{

this.pa = Convert.ToDouble(pa);

this.pb = Convert.ToDouble(pb);

this.pc = Convert.ToDouble(pc);

this.pd = Convert.ToDouble(pd);

this.pe = Convert.ToDouble(pe);

this.step = Convert.ToInt32(step);

}

catch(Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка");

return;

}

}

private void Clear\_Click(object sender, EventArgs e) //Очистить все поля

{

PA.Text = "";

PB.Text = "";

PC.Text = "";

PD.Text = "";

PE.Text = "";

Pmain.Text = "";

chastota.Text = "";

steps.Text = "";

}

double GetPmain()

{

return (1 - (1 - pa) \* (1 - pb)) \* (1 - (1 - (1 - pc)) \* (1 - pd)) \* (1 - pe);

}

private void Model\_Click(object sender, EventArgs e) //Моделирование

{

SetValues(PA.Text, PB.Text, PC.Text, PD.Text, PE.Text, steps.Text);

Pmain.Text = GetPmain().ToString();

double count = 0;

Random ra = new Random();

for (int i = 0; i<step; ++i)

{

Random pra = new Random(ra.Next(10000));

Random prb = new Random(ra.Next(10000));

Random prc = new Random(ra.Next(10000));

Random prd = new Random(ra.Next(10000));

Random pre = new Random(ra.Next(10000));

bool a = pra.NextDouble() <= pa;

bool b = prb.NextDouble() <= pb;

bool c = prc.NextDouble() <= pc;

bool d = prd.NextDouble() <= pd;

bool ee = pre.NextDouble() <= pe;

if((a || b) && (!c || d) && !ee)

{

count++;

}

}

chastota.Text = (count / step).ToString();

}

}

}

Исследовательская часть

Рассмотрим, как меняется частота при разном количестве испытаний. Результаты работы программы представлены на рисунках 2-4.

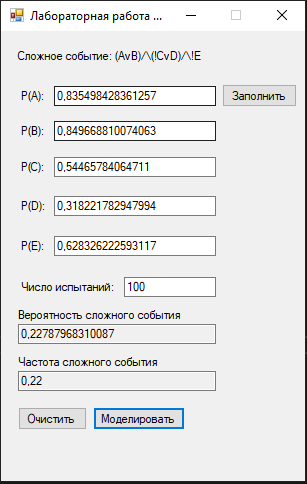


Рисунок 2 - Результат работы при 100 испытаниях

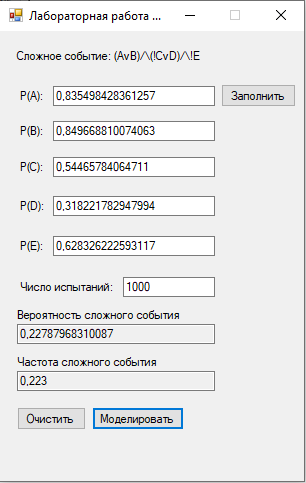


Рисунок 3 - Результат работы при 1000 испытаниях

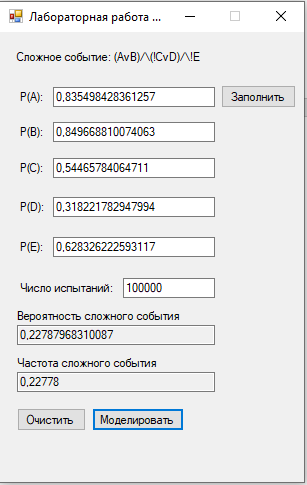


Рисунок 4 - Результат работы при 100000 испытаниях

Вывод: разработана программа, моделирующая сложное событие, заданное формулой алгебры событий, через независимые события A, B, C, D, E.

В ходе исследования выявлена закономерность, заключающаяся в том, что при увеличении количества испытаний частота сложного события стремится к значению вероятности.

Лабораторная работа №2

Задание: разработать в C# с использованием Windows Forms программу, моделирующее экспоненциальное распределение с помощью генератора случайных чисел.

Входные данные: параметр распределения **λ,** количество испытаний.

Выходные данные: экспериментальная гистограмма и расчетный график.

Заданная функция:

Краткое описание алгоритма:

Для построения экспериментальной гистограммы требуется использовать плотность точек, попадающих в заданный интервал. С помощью датчика случайных чисел находится значение F(y), выраженную из формулы экспоненциального распределения.

Инструкция по работе с программой:

После запуска программы пользователем задается параметр распределения и число испытаний с помощью соответствующих полей. Далее требуется нажать кнопку “Рассчитать”. Интерфейс программы представлен на рисунке 5.

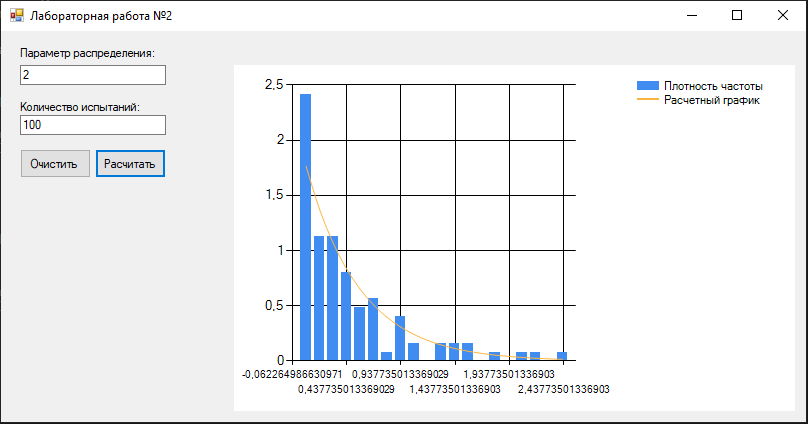


Рисунок 5- Интерфейс программы

Код программы:

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Windows.Forms;

namespace Laba2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

double koef;

int steps;

bool setValues()

{

try

{

koef = Convert.ToDouble(textBox1.Text);

steps = Convert.ToInt32(textBox2.Text);

return true;

}

catch(Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка");

return false;

}

}

private void button1\_Click(object sender, EventArgs e)

{

if (!setValues())

{

return;

}

Random mo = new Random();

List<double> list1 = new List<double>();

chart1.Series[0].Points.Clear();

chart1.Series[1].Points.Clear();

for (int i = 0; i < steps; i++)

{

double y = (-1 / koef) \* Math.Log(1 - mo.NextDouble());

if (y != 1) list1.Add(y);

}

double range = list1.Max() / 20;

double a = 0;

double b = range;

for (int i = 0; i < 20; i++)

{

int count = list1.Where(x => x > a && x <= b).ToArray().Count();

double chastota = count / (steps \* range);

chart1.Series[0].Points.AddXY((a + b) / 2, chastota);

chart1.Series[1].Points.AddXY((a + b) / 2, koef \* Math.Exp(-koef \* ((a + b) / 2)));

a += range;

b += range;

}

}

private void button2\_Click(object sender, EventArgs e)

{

textBox1.Text = "";

textBox2.Text = "";

chart1.Series[0].Points.Clear();

chart1.Series[1].Points.Clear();

}

}

}

Исследовательская часть

Рассмотрим, как меняется график при разных значениях количества испытаний. Результаты работы программы представлены на рисунках 6-11.

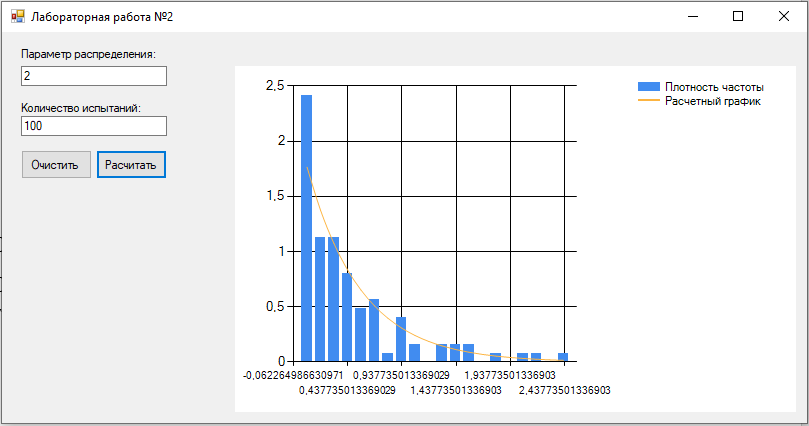


Рисунок 6 - Результат работы при 100 испытаниях

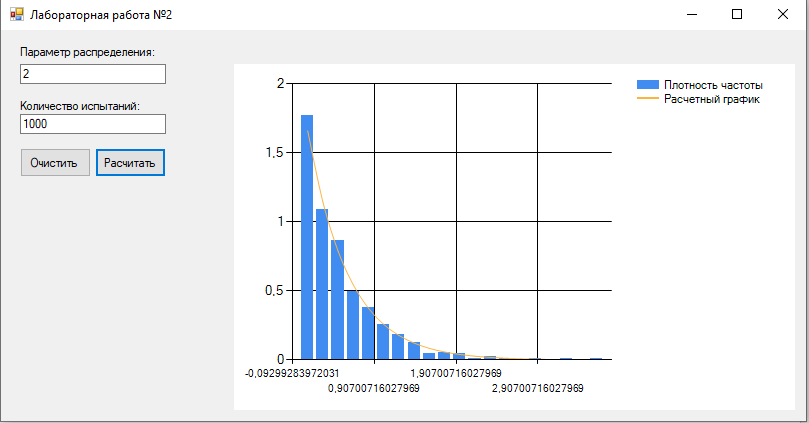


Рисунок 7 - Результат работы при 1000 испытаниях

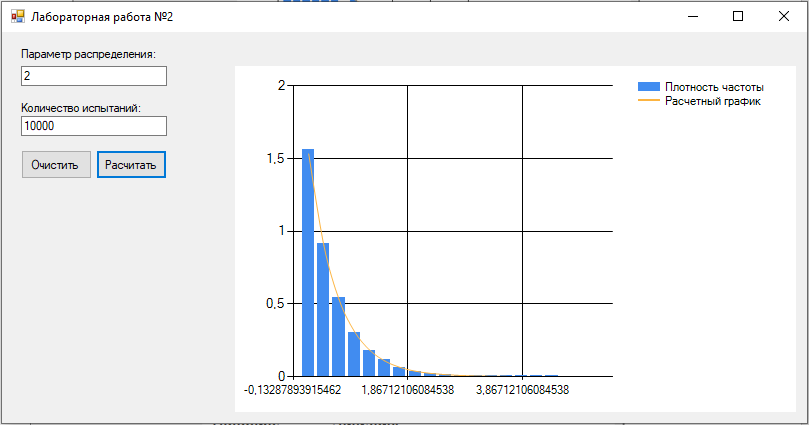


Рисунок 8 - Результат работы при 10000 испытаниях

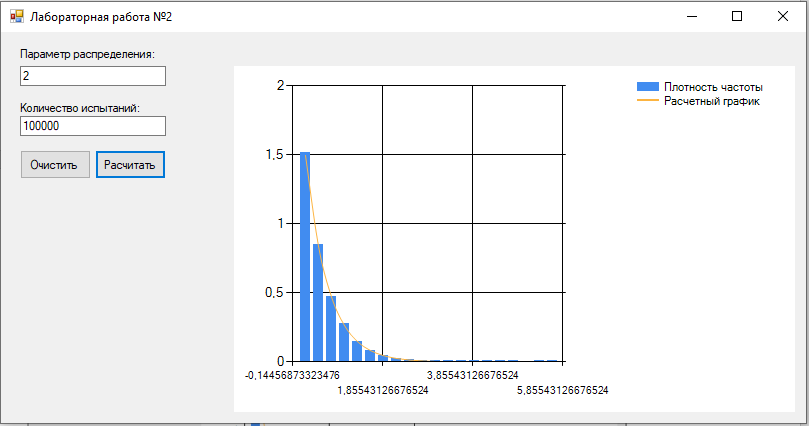


Рисунок 9 - Результат работы при 100000 испытаниях

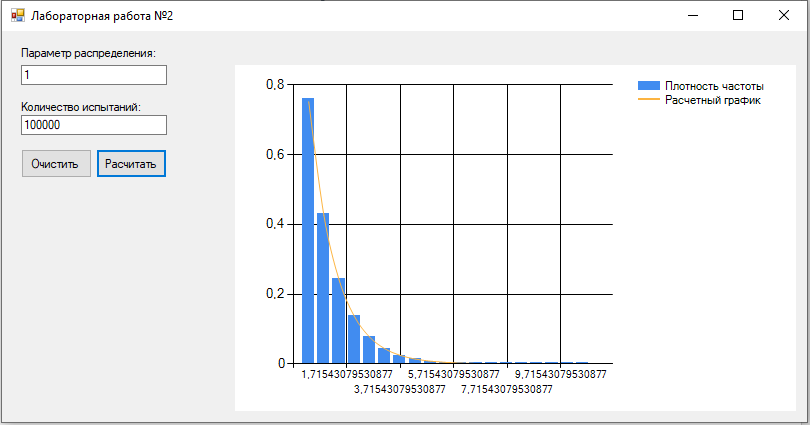


Рисунок 10 - Результат работы при параметре распределения 1

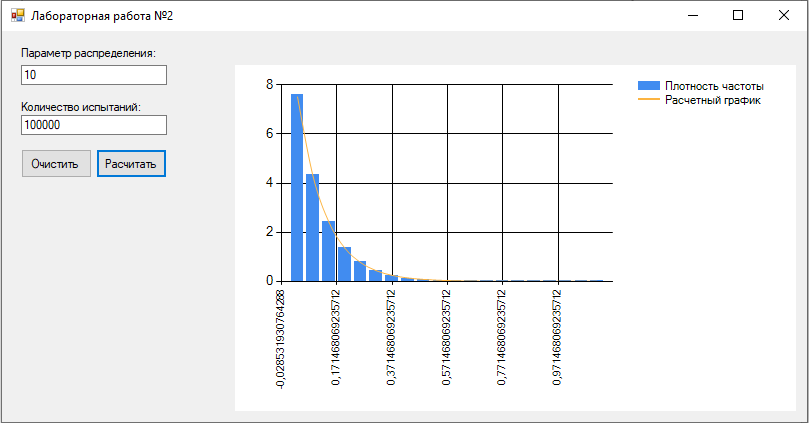


Рисунок 11 - Результат работы при параметре распределения 10

Вывод: разработана программа, моделирующая экспоненциальное распределение с задаваемым параметром распределения.

В ходе исследования было установлено, что при увеличении числа испытаний экспериментальный график стремится к расчетному графику, при большем количестве испытаний, расчетный график проходит ближе к середине столбцов экспериментального графика. Также с увеличением числа испытаний точность графиков, но падает скорость выполнения. При увеличении распределяемого параметра, максимум экспоненциального распределения стремится к заданному параметру распределения.

Лабораторная работа №3

Задание: 3 - разработать программу в C#, с использованием Windows Forms, моделирующую нормальное распределение

Входные данные: математическое ожидание, среднеквадратичное отклонение, количество испытаний.

Выходные данные: экспериментальная гистограмма плотности частоты и расчетный график дифференциальной функции распределения.

Заданная функция:

Краткое описание алгоритма:

Для построения экспериментальной гистограммы из функции нормального распределения выражается X и задается с помощью датчика случайных чисел значения. Для нахождения Y, с помощью площадей прямоугольника с маленьким шагом, вычисляется по формуле распределения значение Y, пока он не будет больше значения X. Для вывода экспериментальной гистограммы используется плотность точек, попадающих в заданный интервал.

Инструкция по работе с программой:

После запуска программы пользователем задается математическое ожидание, среднеквадратичное отклонение и количество испытаний с помощью соответствующих полей. Далее пользователем нажимается кнопка “Рассчитать”. Интерфейс программы представлен на рисунке 12.

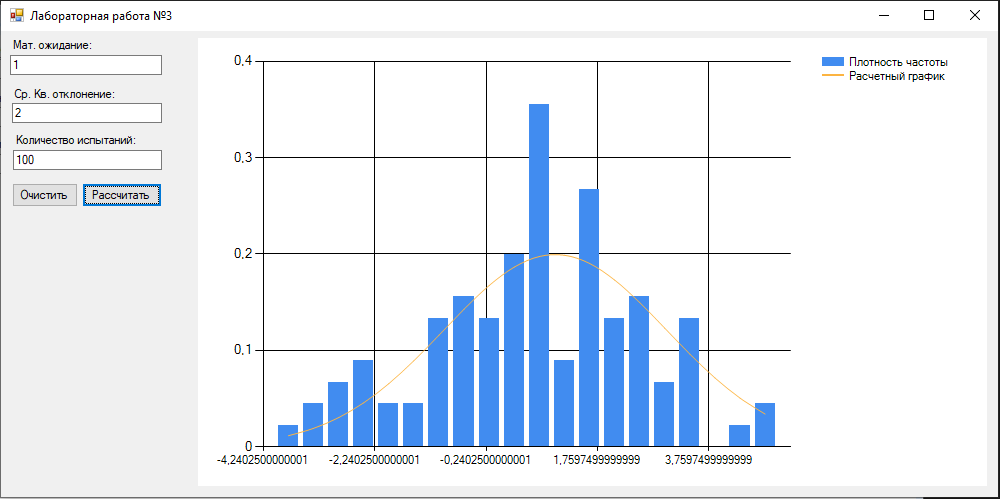


Рисунок 12 - Интерфейс программы

Код программы:

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Windows.Forms;

namespace NormRaspApp

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

double matOzid, otkonenie;

int steps;

bool setValues() //Проверить и установить значения

{

try

{

matOzid = Convert.ToDouble(textBox1.Text);

otkonenie = Convert.ToDouble(textBox2.Text);

steps = Convert.ToInt32(textBox3.Text);

return true;

}

catch(Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка");

return false;

}

}

private void button2\_Click(object sender, EventArgs e) //Очистить

{

textBox1.Text = "";

textBox2.Text = "";

textBox3.Text = "";

chart1.Series[0].Points.Clear();

chart1.Series[1].Points.Clear();

}

private void button1\_Click(object sender, EventArgs e) //Рассчитать

{

if (!setValues()) {return; }

chart1.Series[0].Points.Clear();

chart1.Series[1].Points.Clear();

double shag = (5 \* otkonenie) / 1000;

List<double> list = new List<double>();

Random r = new Random();

for (int i = 0; i < steps; i++)

{

double x = r.NextDouble();

double y0 = 0;

double S = 0;

double h = matOzid - 5 \* otkonenie;

while (S <= x \* otkonenie \* Math.Sqrt(2 \* Math.PI))

{

double y = Math.Exp(-1 \* Math.Pow((h - matOzid), 2) / (2 \* Math.Pow(otkonenie, 2)));

y0 = h;

S += y \* shag;

h += shag;

}

double Y = (y0 + h) / 2;

list.Add(Y);

}

double max = list.Max();

double min = list.Min();

double range = (max - min) / 20;

double a = min;

double b = min + range;

for (double i = min; i <= max; i += range)

{

int count = list.Where(x => x > a && x <= b).ToArray().Count();

double chastota = count / (steps \* range);

chart1.Series[0].Points.AddXY((a + b) / 2, chastota);

chart1.Series[1].Points.AddXY((a + b) / 2, (1 / (otkonenie \* Math.Sqrt(2 \* Math.PI))) \* Math.Exp(-1 \* (Math.Pow((((a + b) / 2) - matOzid), 2)) / (2 \* Math.Pow(otkonenie, 2))));

a += range;

b += range;

}

}

}

}

Исследовательская часть:

Рассмотрим изменения графиков при разных задаваемых параметрах. Результаты работы программы при разных параметрах представлены на рисунках 13 – 20.

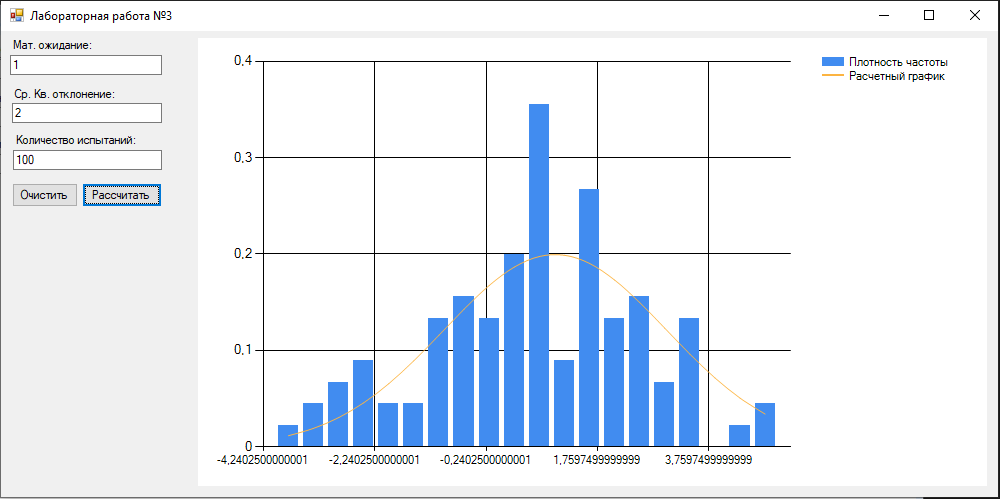


Рисунок 13 - Результат работы при 100 испытаниях

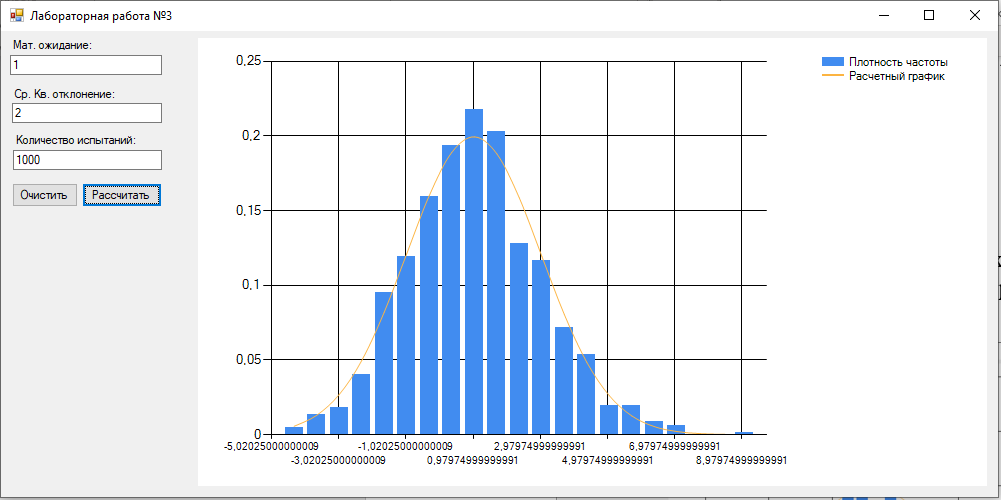


Рисунок 14 - Результат работы при 1000 испытаниях

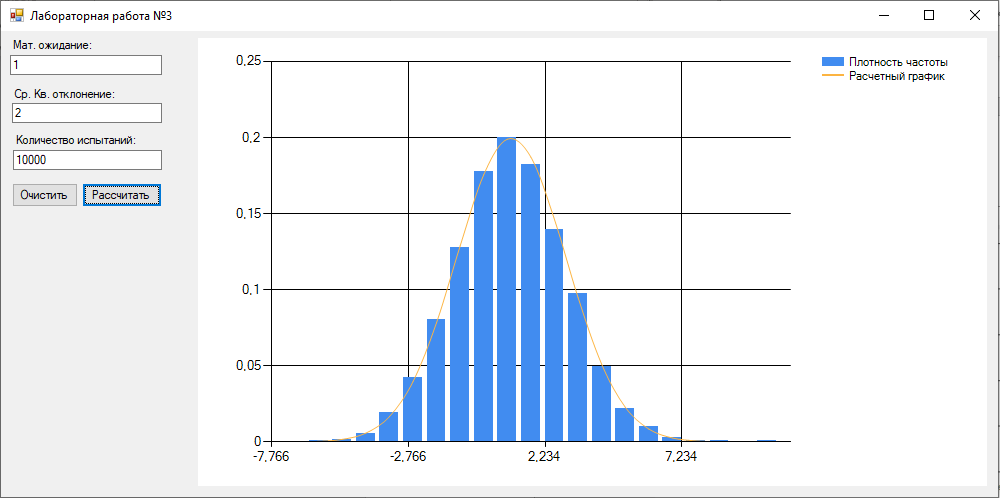


Рисунок 15- Результат работы при 10000 испытаниях

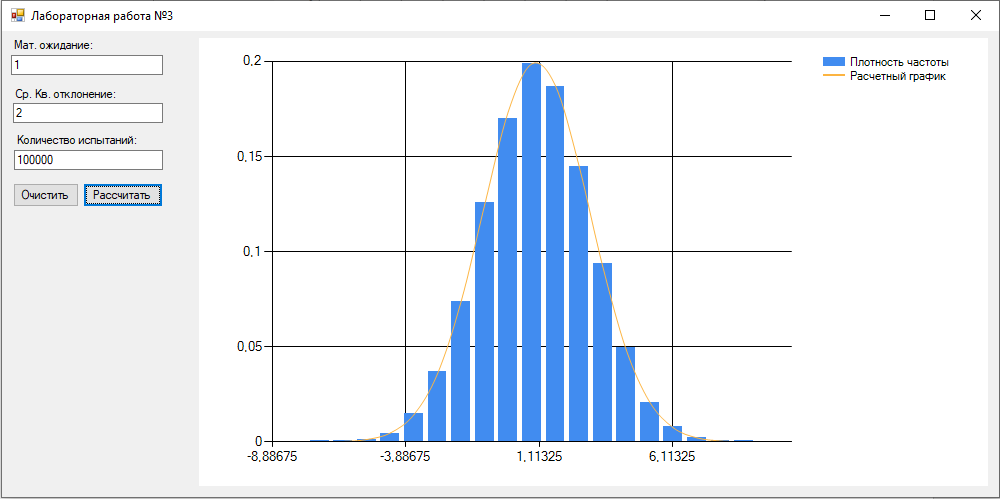


Рисунок 16 - Результат работы при 100000 испытаниях

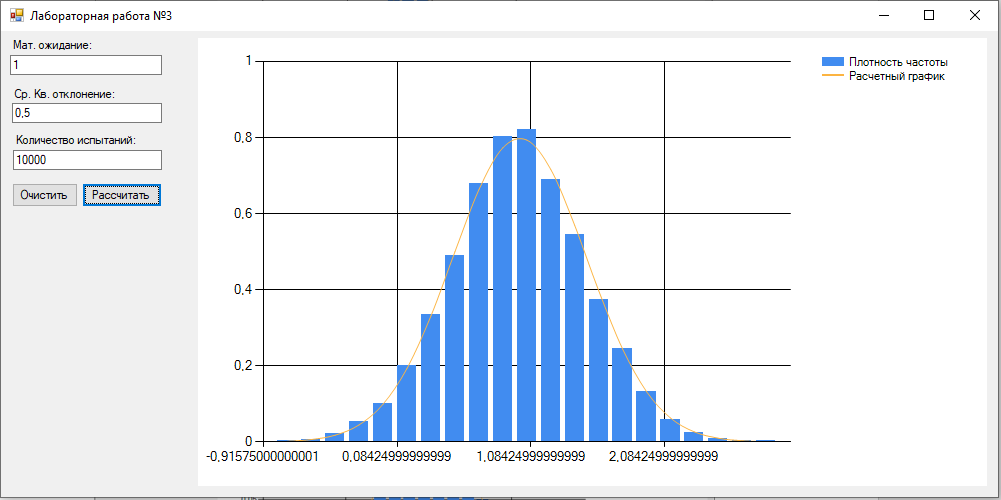


Рисунок 17 - Результат работы при среднеквадратичном отклонении 0,5

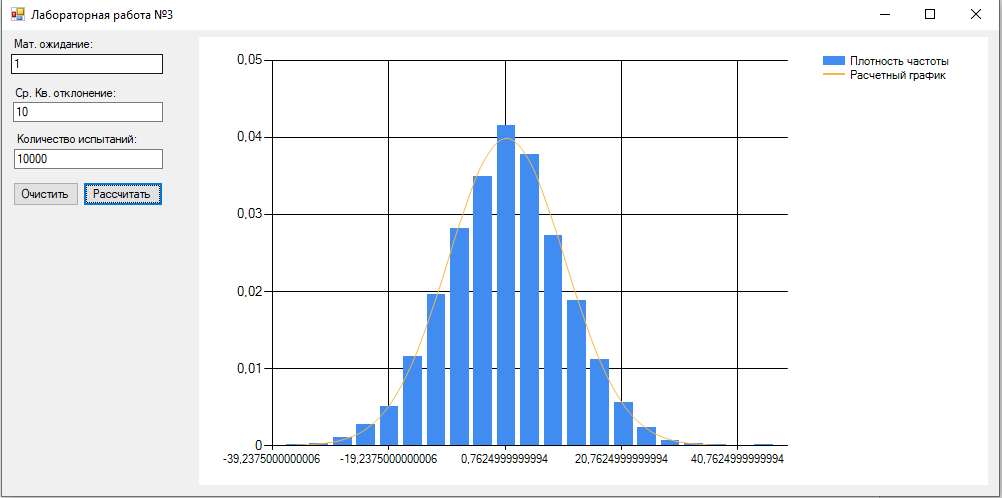


Рисунок 18 - Результат работы при среднеквадратичном отклонении 10

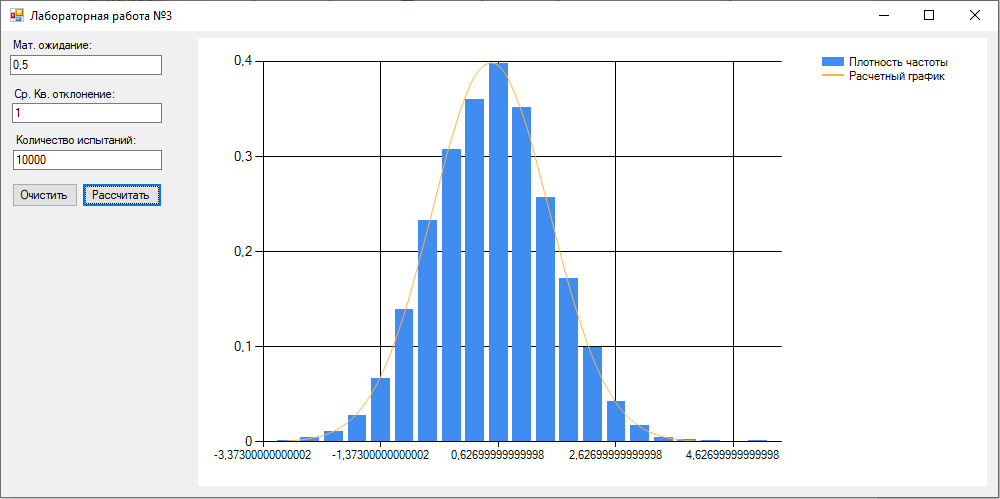


Рисунок 19 - Результат при математическом ожидании 0,5

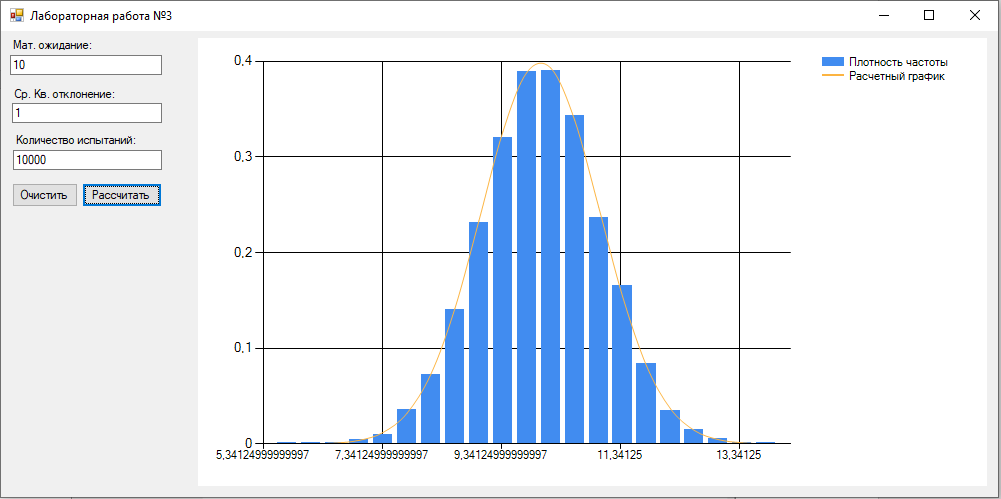


Рисунок 20 - Результат работы при математическом ожидании 10

Вывод: разработана программа, моделирующая нормальное распределение с задаваемым математическим ожиданием, среднеквадратичным отклонением, количеством испытаний.

При увеличении количества испытаний график с плотностью частоты стремится к расчетному графику. При большом числе испытаний, расчетный график проходит через середины столбцов плотности частоты. При увеличении математического ожидания максимальное значение нормального распределения графика смещается. При увеличении среднеквадратичного отклонения максимум расчетного графика уменьшается.

Лабораторная работа №4

Задание:

Разработать в C#, с использованием Windows Forms, программу, моделирующую нормальное распределение с задаваемым математическим ожиданием и среднеквадратичным отклонением с помощью метода Бокса-Мюллера.

Входные данные: математическое ожидание, среднеквадратичное отклонение, количество испытаний.

Выходные данные: экспериментальная гистограмма плотности частоты и расчетный график дифференциальной функции распределения.

Заданная функция: 1d√2П𝑒−(𝑦−𝑚)22d2

Краткое описание алгоритма:

Метод Бокса-Мюллера позволяет получить пару независимых нормально распределенных случайных величин с математическим ожиданием 0 и дисперсией 1 следующим образом:

Для вычисления величины нормального распределения случайной величины с математическим ожиданием и стандартным отклонением по формуле: Z = q \* u + m

Для вывода экспериментальной гистограммы используется плотность точек, попадающих в заданный интервал.

Инструкция по работе с программой:

После запуска программы пользователем задается математическое ожидание, среднеквадратичное отклонение и количество испытаний с помощью соответствующих полей. Далее пользователем нажимается кнопка “Рассчитать”. Интерфейс программы представлен на рисунке 21.

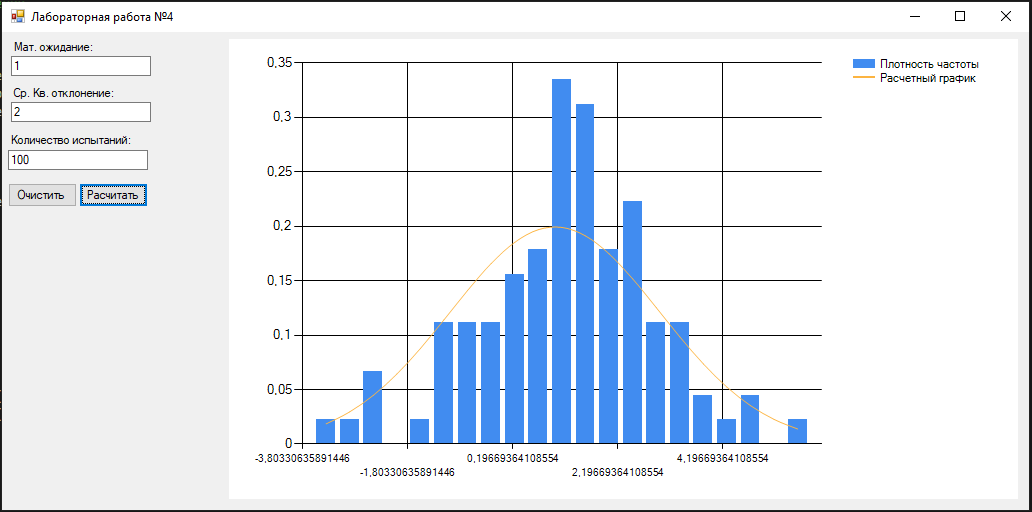


Рисунок 21 - Интерфейс программы

Код программы:

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Windows.Forms;

namespace Laba4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

double matOzid, otkonenie;

int steps;

bool setValues() //Установка значений

{

try

{

matOzid = Convert.ToDouble(textBox1.Text);

otkonenie = Convert.ToDouble(textBox2.Text);

steps = Convert.ToInt32(textBox3.Text);

return true;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка");

return false;

}

}

private void button2\_Click(object sender, EventArgs e) //Очистить

{

textBox1.Text = "";

textBox2.Text = "";

textBox3.Text = "";

chart1.Series[0].Points.Clear();

chart1.Series[1].Points.Clear();

}

private void button1\_Click(object sender, EventArgs e) //Расчитать

{

if (!setValues())

{

return;

}

chart1.Series[0].Points.Clear();

chart1.Series[1].Points.Clear();

Random r = new Random();

Random x = new Random(r.Next(10000));

Random y = new Random(r.Next(10000));

List<double> list = new List<double>();

for (int i = 0; i < steps; i++)

{

double u = Math.Cos(2 \* Math.PI \* x.NextDouble()) \* Math.Sqrt(-2 \* Math.Log(y.NextDouble()));

double z = otkonenie \* u + matOzid;

list.Add(z);

}

double max = list.Max();

double min = list.Min();

double range = (max - min) / 20;

double a = min;

double b = min + range;

for (double i = min; i <= max; i += range)

{

int count = list.Where(p => p > a && p <= b).ToArray().Count();

double chastota = count / (steps \* range);

chart1.Series[0].Points.AddXY((a + b) / 2, chastota);

chart1.Series[1].Points.AddXY((a + b) / 2, (1 / (otkonenie \* Math.Sqrt(2 \* Math.PI))) \*

Math.Exp(-1 \* (Math.Pow((((a + b) / 2) - matOzid), 2)) / (2 \* Math.Pow(otkonenie, 2))));

a += range;

b += range;

}

}

}

}

Исследовательская часть:

Рассмотрим изменения графиков при задаваемых значениях. Результаты работы программы при разных параметрах представлены на рисунках 21 -27.

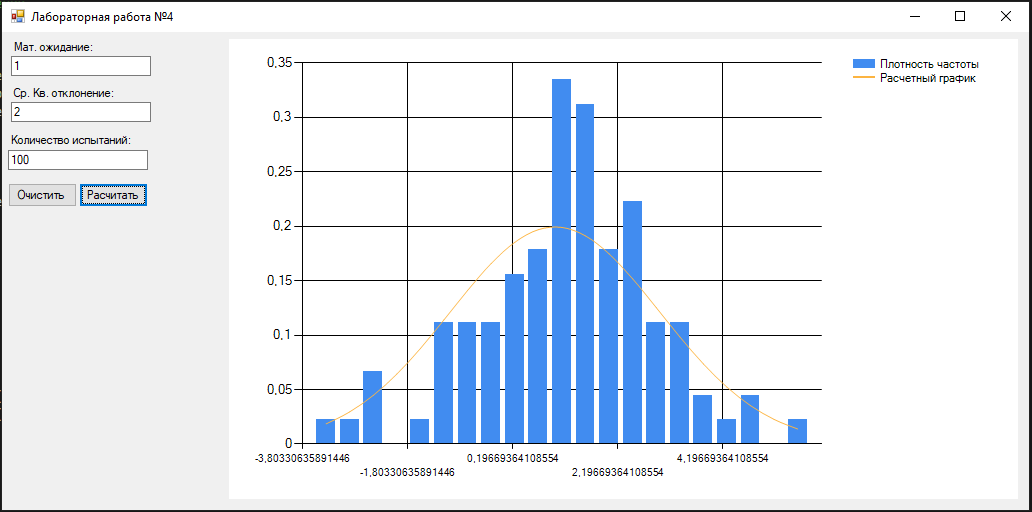


Рисунок 22 - Результат работы с 100 испытаниями

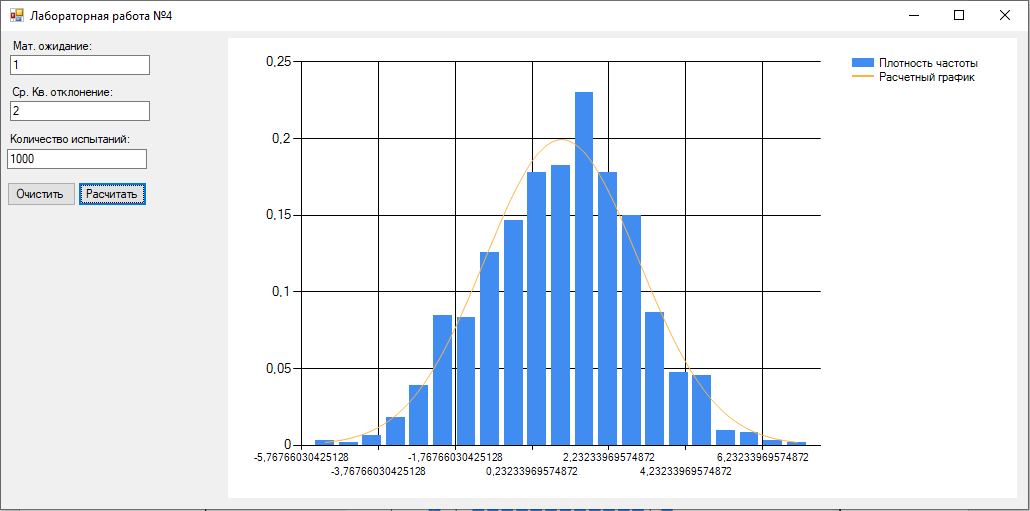


Рисунок 23 - Результат работы с 1000 испытаниями

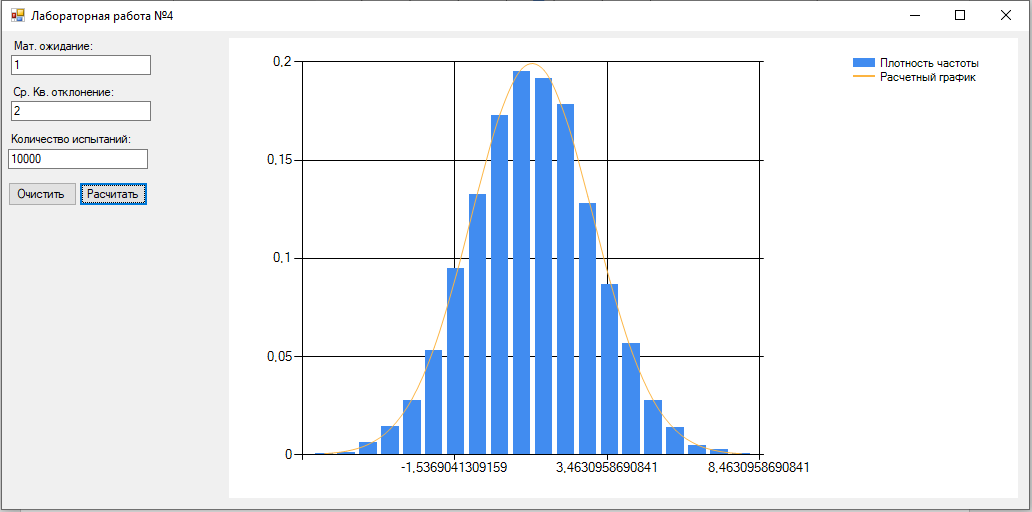


Рисунок 24 - Результат работы с 10000 испытаниями

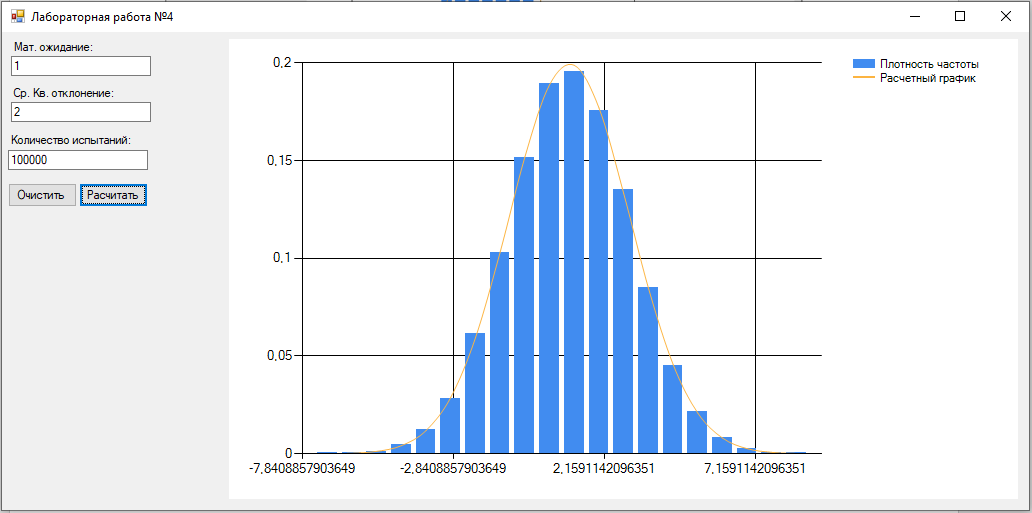


Рисунок 25 - Результат работы с 100000 испытаниями

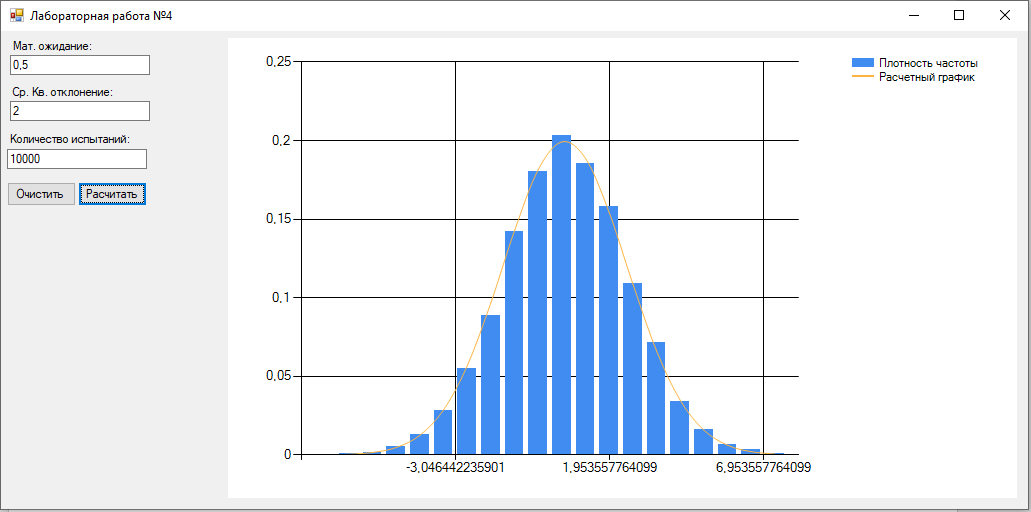


Рисунок 26 - Результат работы с математическим ожиданием 0,5

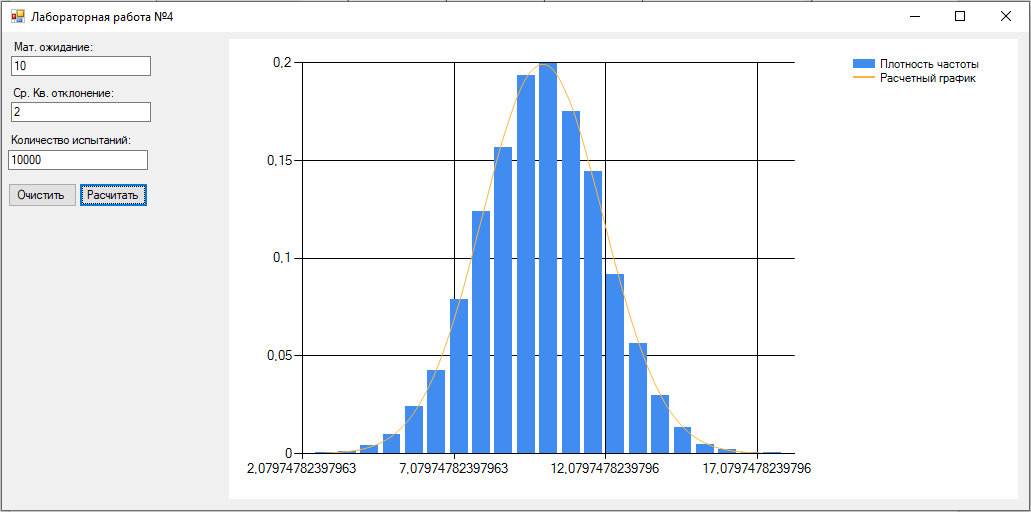


Рисунок 27 - Результат работы с математическим ожиданием 10

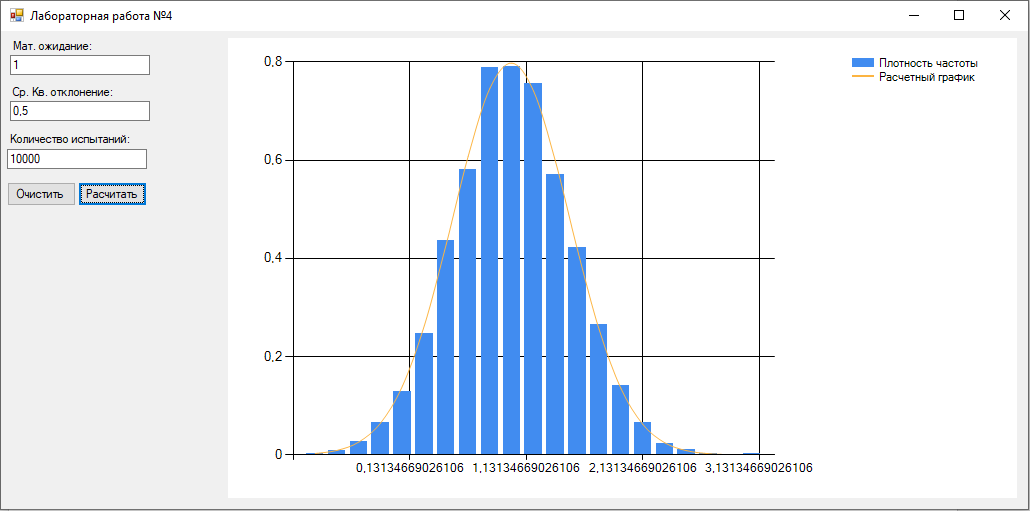


Рисунок 28 - Результат работы с среднеквадратичным отклонением 0,5

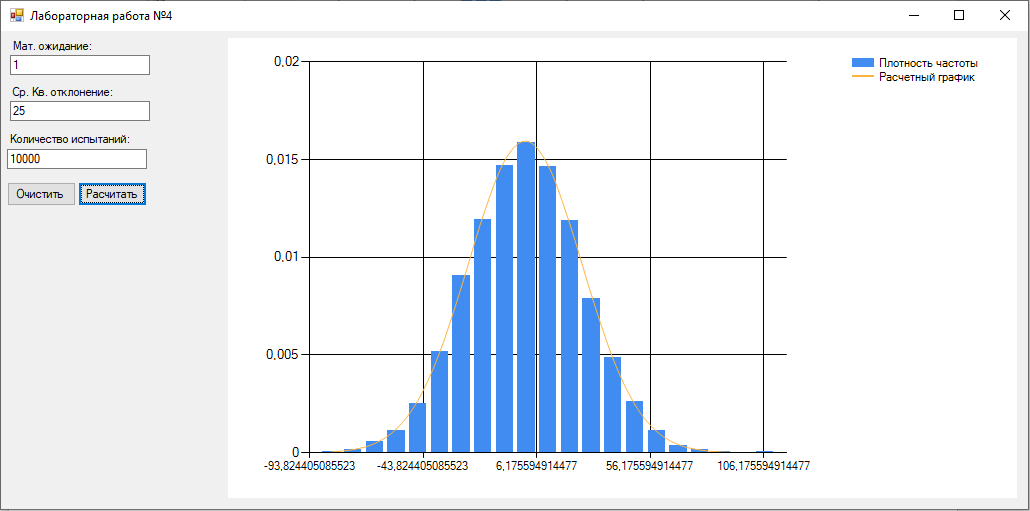


Рисунок 29 - Результат работы с среднеквадратичным отклонением 25

Вывод: разработана программа, моделирующая нормальное распределение с задаваемым математическим ожиданием, среднеквадратичным отклонением с помощью метода Бокса-Мюллера.

При увеличении числа испытаний график плотности частоты стремиться к расчетному графику. При большем количестве испытаний расчетный график проходит через середины столбцов графика плотности частоты. При увеличении среднеквадратичного отклонения максимум расчетного графика уменьшается. При увеличении математического ожидания максимальное значение нормального распределения графика смещается.