Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

ОТЧЕТ

по лабораторной работе №1
по дисциплине “Статистическое моделирование случайных процессов и систем”

Выполнил Стахеев Д. И.
студент гр. 3530904/00104

Преподаватель Чуркин В. В.

Санкт-Петербург
2023

Содержание

Цель работы

1. Получение на ЭВМ с помощью программного датчика базовой последовательности псевдослучайных чисел, имеющих равномерное распределение.
2. Освоение методов статистической оценки полученного распределения: вычисление эмпирических значений для математического ожидания и дисперсии.
3. Освоение методов оценки статистики связи: вычисление значений автокорреляционной функции и построение коррелограммы.
4. Освоение методов графического представления законов распределения: построение функции плотности распределения и интегральной функции распределения.

Генерация псевдослучайной последовательности

Для генерации последовательности был использован предопределённый генератор псевдослучайных чисел *mt19937,* сид для которого был сгенерирован при помощи генератора истинных случайных чисел *random_device*. Также была использована функция равномерного распределения *uniform_real_distribution* (для постобработки выходных данных ГПСЧ). Все упомянутые элементы являются частью заголовочного файла *random*.

Вычисление эмпирических значений математического ожидания и дисперсии

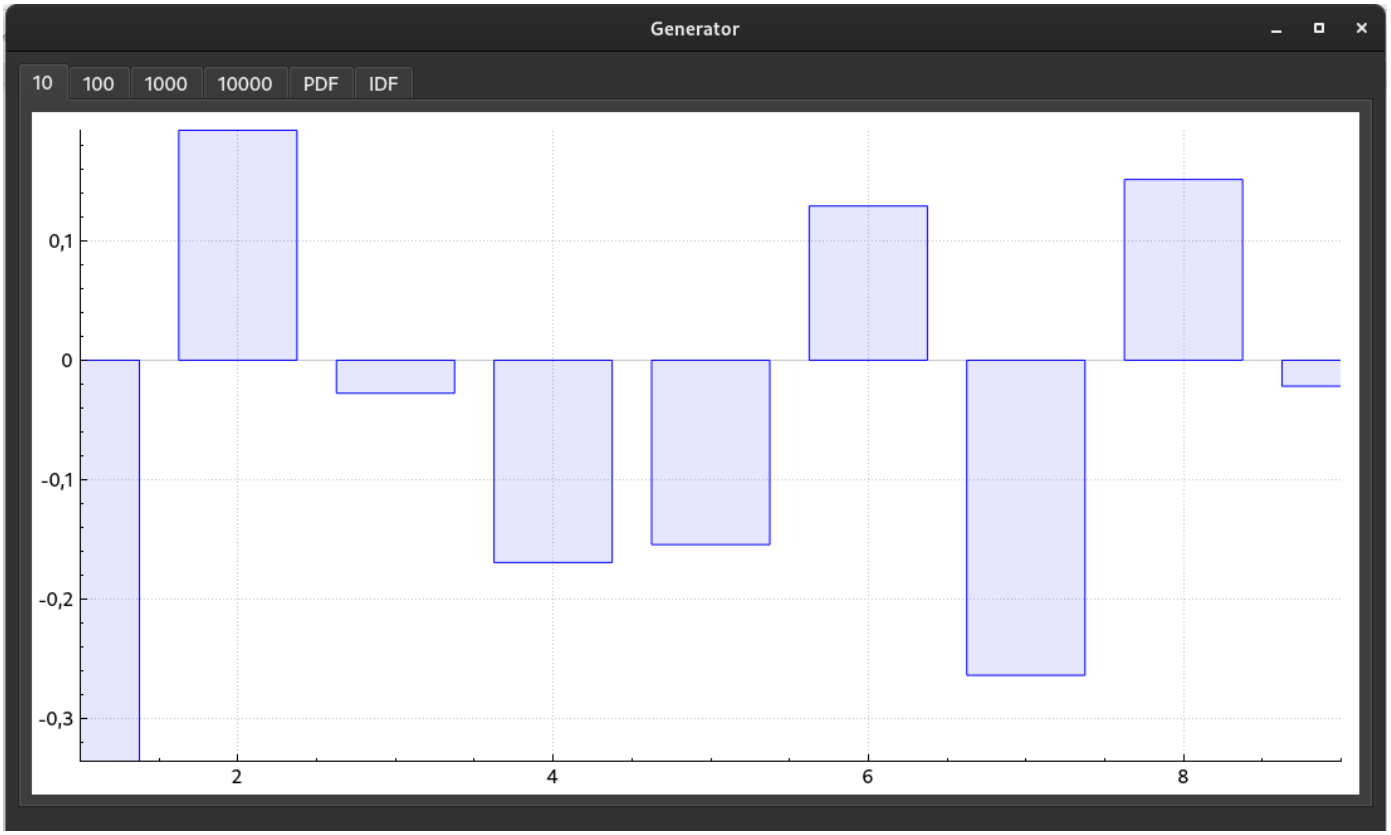
N	Оценка распр.	Эксперимент	Теоретическое значение	Отклонение
10	M	0.538654	0.5	-0.038654
	D	0.0706622	0.08333	0.0126678
100	M	0.507071	0.5	-0.007071
	D	0.0947384	0.08333	-0.0114084
1000	M	0.493757	0.5	0.006243
	D	0.0818335	0.08333	0.0014965
10000	M	0.50016	0.5	-0.00016
	D	0.0835473	0.08333	-0.0002173

При увеличении числа испытаний (N) эмпирические значения приближаются к теоретическим. Так, при N = 10000 наблюдается минимальное по модулю отклонение.

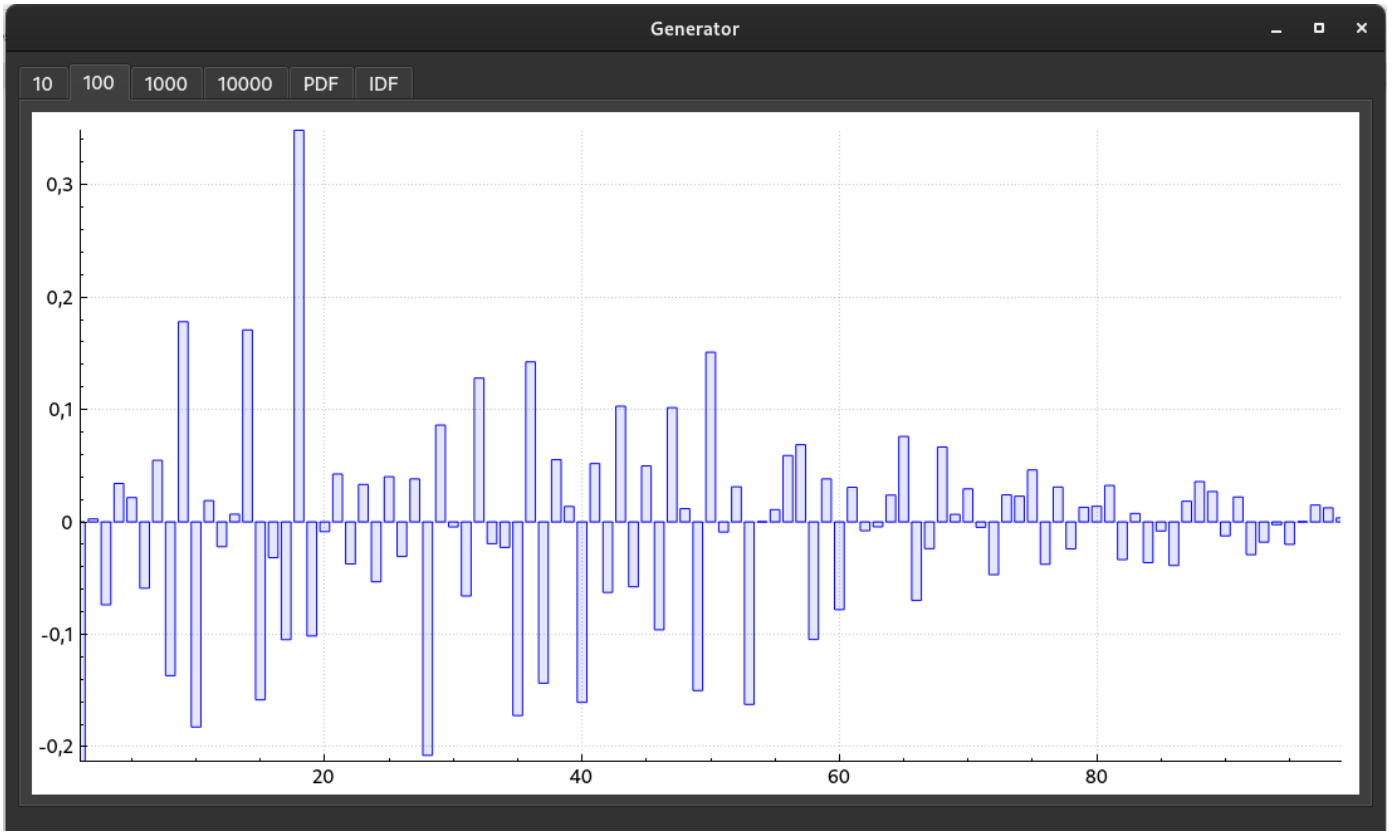
Вычисление значений автокорреляционной функции и построение коррелограммы

Построение коррелограмм выполнялось при помощи библиотеки QCustomPlot для фреймворка Qt.

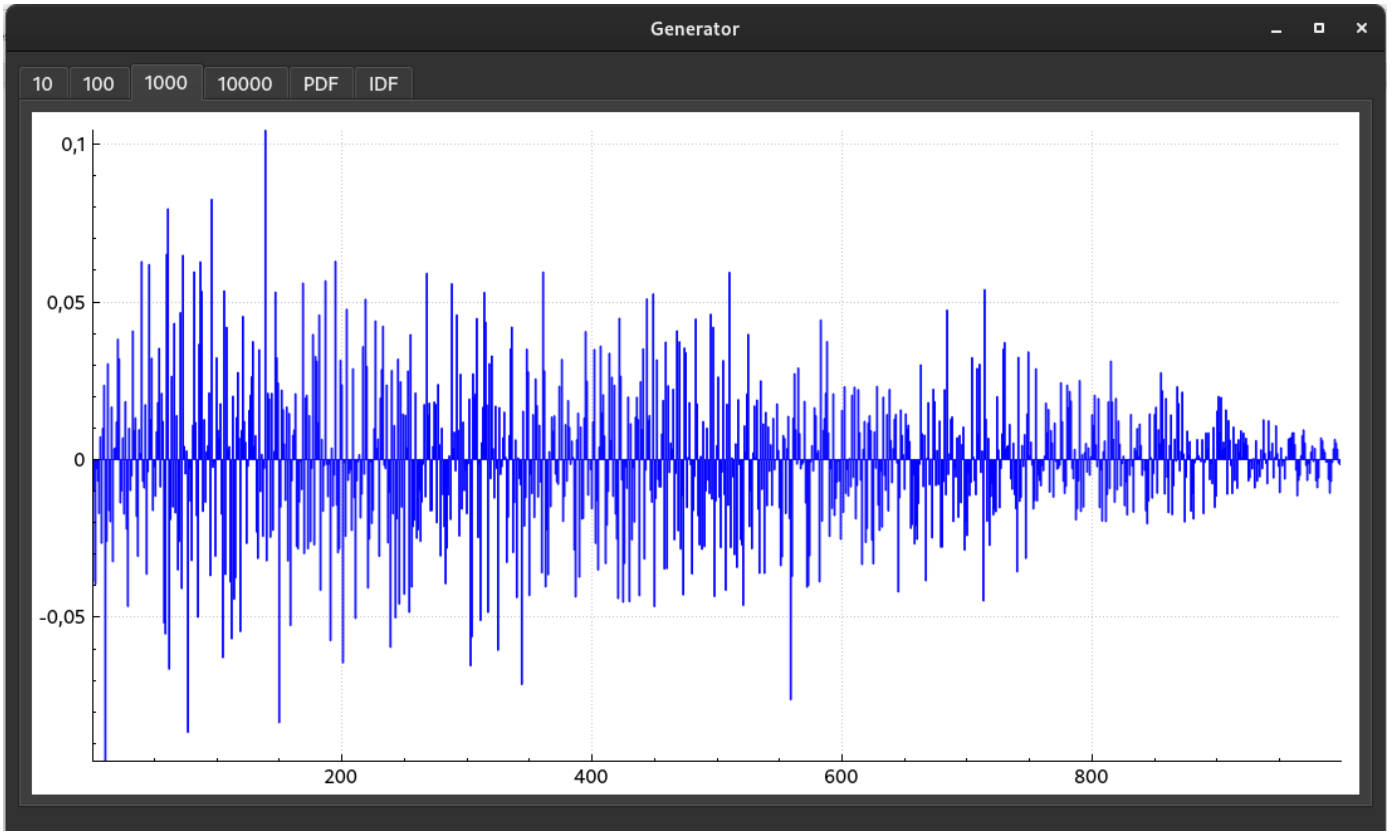
N = 10



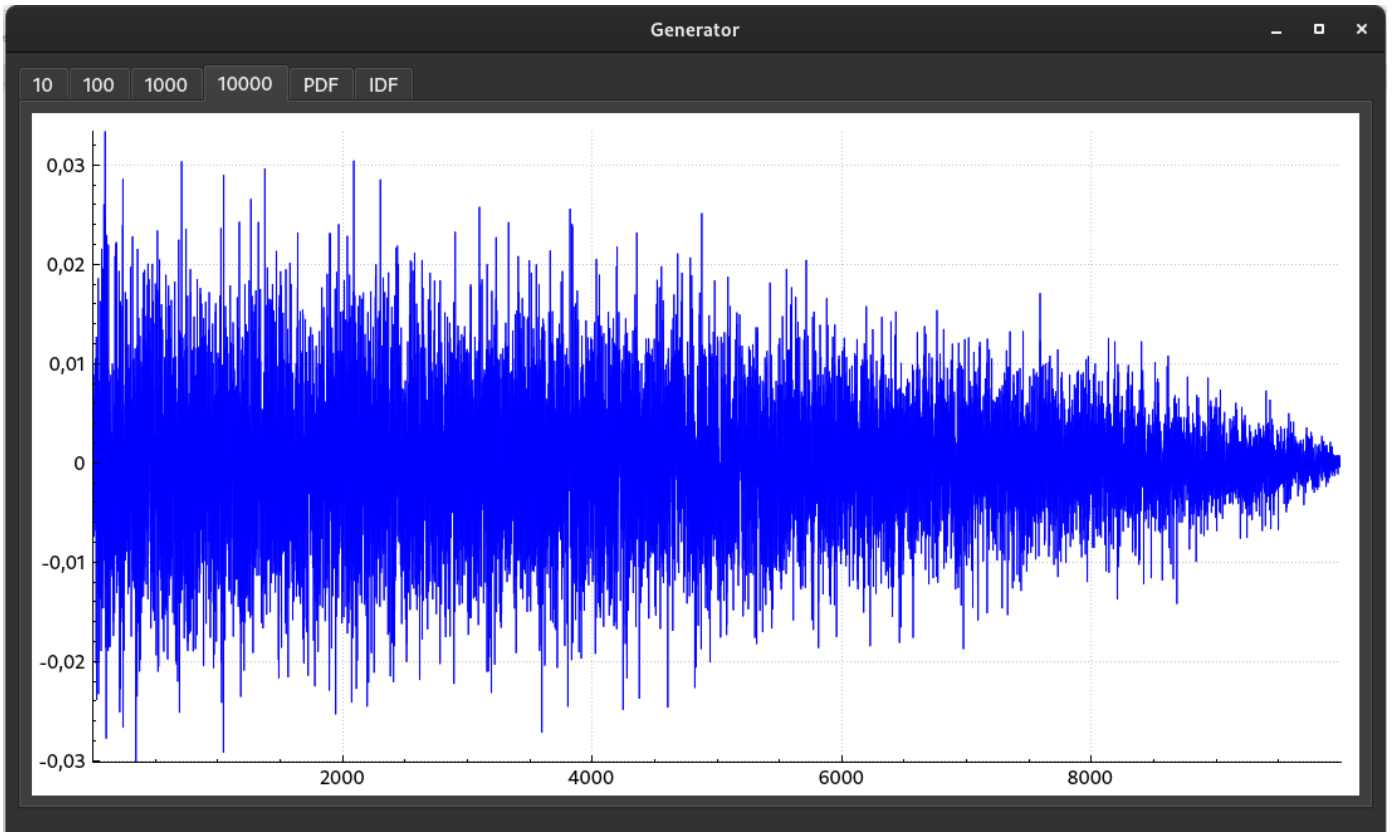
N = 100


Generator
10
100
1000
10000
PDF
IDF
0,3
0,2
0,1
0
-0,1
-0,2
20
40
60
80

N = 1000

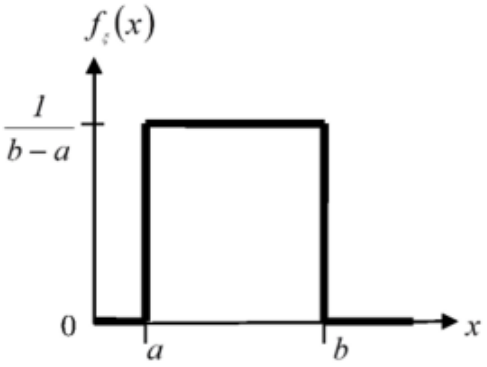

Generator
10
100
1000
10000
PDF
IDF
0,1
0,05
0
-0,05
200
400
600
800

N = 10000:

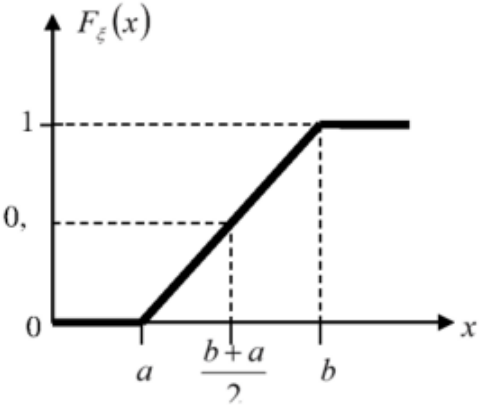

Generator
10
100
1000
10000
PDF
IDF
0,03
0,02
0,01
0
-0,01
-0,02
-0,03
2000
4000
6000
8000

Графическое представление законов распределения

Теоретические кривые



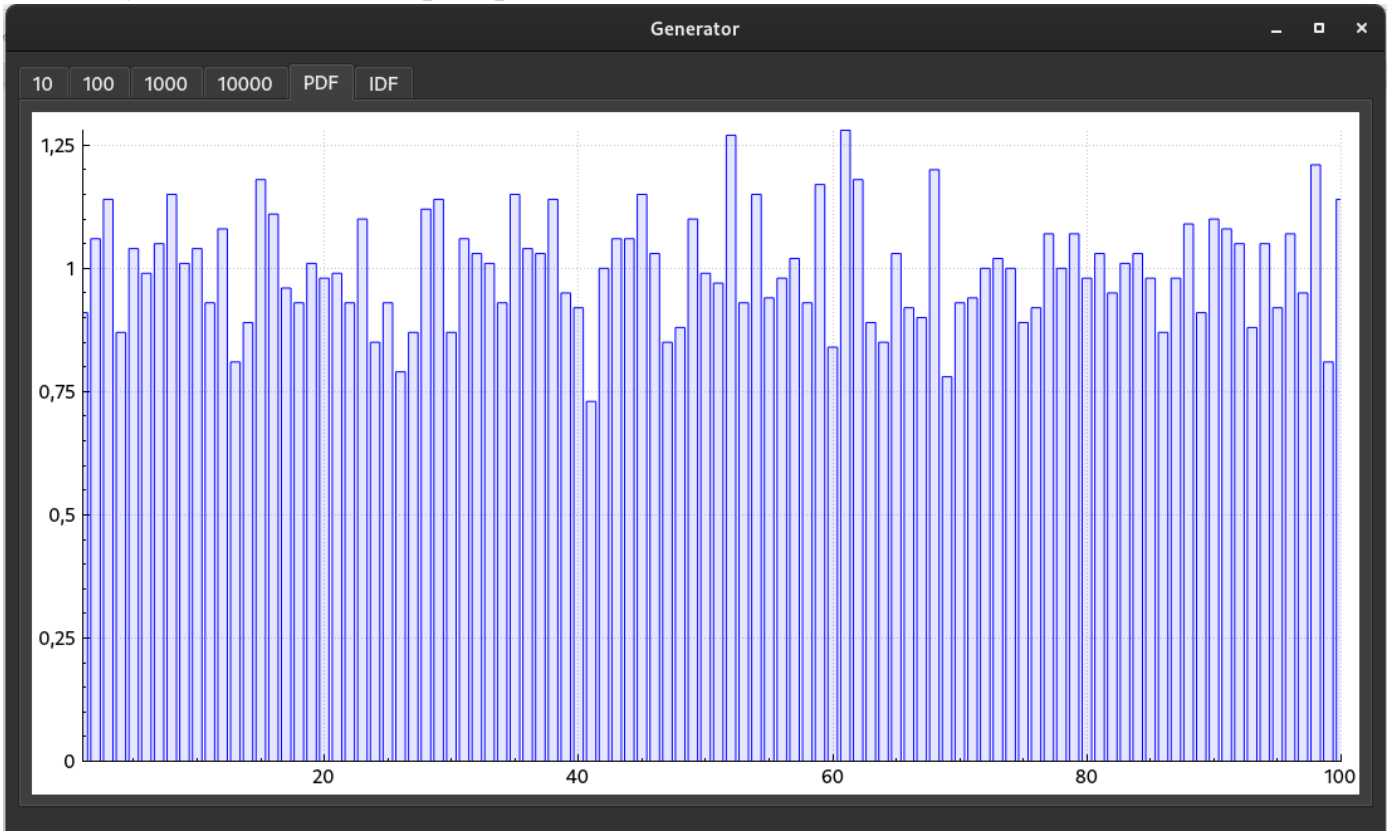
а – дифференциальная функция равномерно распределенной случайной величины



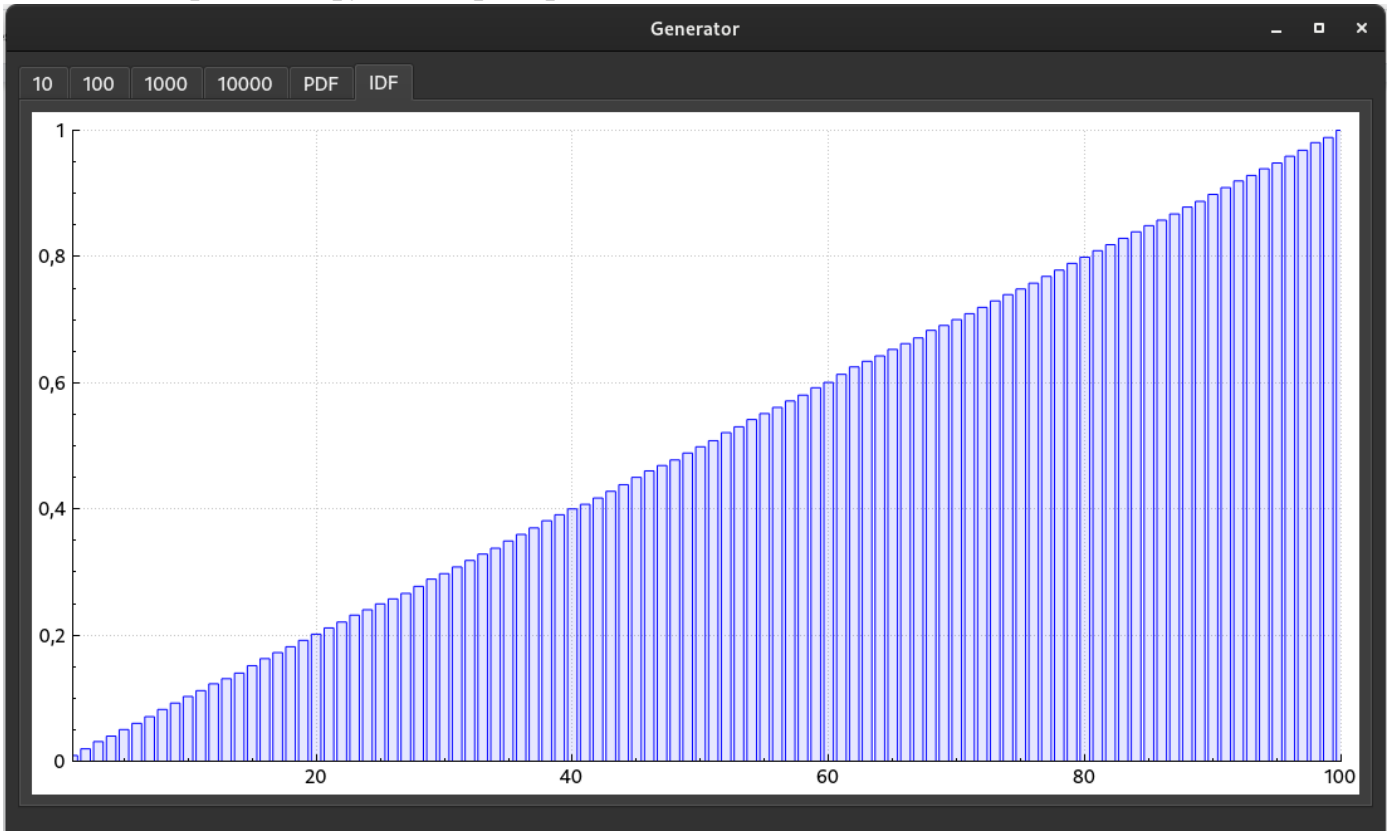
б – интегральная функция равномерно распределенной случайной величины

Эмпирические кривые (для N = 10000)

1. Функция плотности распределения



2. Интегральная функции распределения



Несмотря на локальные “скачки” общий вид графика функции плотности распределения совпадает с теоретическим. Эмпирический график интегральной функции распределения имеет вид теоретического графика.

Вывод

В результате выполнения лабораторной работы было установлено, что комбинация генераторов случайных чисел *mt19937* и *random_device* и функции распределения *uniform_real_distribution* является корректной для генерации случайных чисел с равномерным законом распределения.

Приложение 1. Листинг программы

main.cpp

```
#include "MainWindow.hpp"

#include <QApplication>

int main(int argc, char *argv[])
{
 QApplication a(argc, argv);
 MainWindow w;
 w.show();
 return a.exec();
}
```

MainWindow.hpp

```
#ifndef MAINWINDOW_HPP
#define MAINWINDOW_HPP

#include <QMainWindow>
#include <QVector>
#include <numeric>
#include <random>
#include <limits>
#include <cmath>
#include <algorithm>

#include "qcustomplot.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
 Q_OBJECT

public:
 MainWindow(QWidget *parent = nullptr);

 void calculate_data(const int num_of_numbers);
 void fill_vec_of_numbers(const int num_of_numbers);

 ~MainWindow();

private:
 Ui::MainWindow *ui;
 QVector<double> vec_of_numbers;
 QVector<double> vec_of_k;

 void draw_histogram(const int num_of_numbers);
```

```
 void draw_pdf(const QVector<double>& vec_of_f);
 void draw_idf(const QVector<double>& vec_of_big_f);
};
#endif // MAINWINDOW_HPP
```

MainWindow.cpp

```
#include "MainWindow.hpp"
#include "ui_MainWindow.h"

MainWindow::MainWindow(QWidget *parent)
 : QMainWindow(parent)
 , ui(new Ui::MainWindow)
{
 ui->setupUi(this);

 for (int i = 10; i <= 10000; i*=10)
 {
  fill_vec_of_numbers(i);
  calculate_data(i);
  draw_histogram(i);
 }
}

void MainWindow::calculate_data(const int num_of_numbers)
{
 vec_of_k.clear();
 vec_of_k.resize(num_of_numbers - 1);

 // Calculate M
 double sum = std::accumulate(vec_of_numbers.begin(), vec_of_numbers.end(), 0.0);
 const double EXPECTED_VAL = sum / num_of_numbers;

 // Calculate dispersion and correlation coefficients
 double numerator_of_disp = 0;
 double denominator = 0;

 for (int k = 0; k < num_of_numbers; k++)
  denominator += std::pow((vec_of_numbers[k] - EXPECTED_VAL), 2);

 for (int j = 1; j <= num_of_numbers; j++)
 {
  numerator_of_disp += std::pow((vec_of_numbers[j - 1] - EXPECTED_VAL), 2) ;
  double numerator = 0;

  for (int k = 0; k < num_of_numbers - j; k++)
   numerator += (vec_of_numbers[k] - EXPECTED_VAL) * (vec_of_numbers[k + j] - EXPECTED_VAL);

  if (j < num_of_numbers)
   vec_of_k[j - 1] = numerator / denominator;
 }

 // Calculate Probability density function and integral distribution function
```

```
if (num_of_numbers == 10000)
{
  const int num_of_intervals = 100;
  const double step = 0.01;
  QVector<double> vec_of_f(num_of_intervals);
  QVector<double> vec_of_bf(num_of_intervals);

  for (int i = 1; i <= num_of_intervals; i++)
  {
    double previous_step = step * (i - 1);
    double current_step = step * i;

    int num_of_occur_f = std::count_if(vec_of_numbers.begin(), vec_of_numbers.end(), [&](double j) {
      return (j < current_step && j >= previous_step);
    });

    vec_of_f[i - 1] = num_of_occur_f / (vec_of_numbers.size() * step) ;

    double num_of_occur_bf = std::count_if(vec_of_numbers.begin(), vec_of_numbers.end(), [&](double j) {
      return (j < current_step);
    });

    vec_of_bf[i - 1] = num_of_occur_bf / vec_of_numbers.size();
  }

  draw_pdf(vec_of_f);
  draw_idf(vec_of_bf);
}

qDebug() << "M = " << EXPECTED_VAL;
qDebug() << "D = " << (numerator_of_disp / num_of_numbers);
qDebug() << "S = " << std::sqrt(numerator_of_disp / num_of_numbers);
qDebug() << "---------------------------------";
}

void MainWindow::fill_vec_of_numbers(const int num_of_numbers)
{
vec_of_numbers.clear();
vec_of_numbers.resize(num_of_numbers);

std::random_device dev;
std::mt19937 rng(dev());
std::uniform_real_distribution<> dist(0.0, std::nextafter(1.0, std::numeric_limits<double>::max()));

std::generate(vec_of_numbers.begin(), vec_of_numbers.end(), [&]() mutable {return dist(rng);});
}

void MainWindow::draw_histogram(const int num_of_numbers)
{
QVector<double> keyData(num_of_numbers - 1);
std::iota(keyData.begin(), keyData.end(), 1);

QCPBars *myBars;
QCustomPlot* plot_ptr;
```

```
 switch(num_of_numbers)
 {
   case 10:
     plot_ptr = ui->plot_10;
     break;
   case 100:
     plot_ptr = ui->plot_100;
     break;
   case 1000:
     plot_ptr = ui->plot_1000;
     break;
   case 10000:
     plot_ptr = ui->plot_10000;
     break;
 }

 myBars = new QCPBars(plot_ptr->xAxis, plot_ptr->yAxis);
 myBars->setData(keyData, vec_of_k);

 plot_ptr->rescaleAxes();
 plot_ptr->replot();
}

void MainWindow::draw_pdf(const QVector<double>& vec_of_f)
{
 QVector<double> num_of_intervals(100);
 std::iota(num_of_intervals.begin(), num_of_intervals.end(), 1);

 QCPBars *pdf_ptr = new QCPBars(ui->plot_pdf->xAxis, ui->plot_pdf->yAxis);
 pdf_ptr->setData(num_of_intervals, vec_of_f);

 ui->plot_pdf->rescaleAxes();
 ui->plot_pdf->replot();
}

void MainWindow::draw_idf(const QVector<double>& vec_of_big_f)
{
 QVector<double> num_of_intervals(100);
 std::iota(num_of_intervals.begin(), num_of_intervals.end(), 1);

 QCPBars *idf_ptr = new QCPBars(ui->plot_idf->xAxis, ui->plot_idf->yAxis);
 idf_ptr->setData(num_of_intervals, vec_of_big_f);

 ui->plot_idf->rescaleAxes();
 ui->plot_idf->replot();
}

MainWindow::~MainWindow()
{
 delete ui;
}
```