ГУАП

КАФЕДРА № 41

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | Е.К.Григорьев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| МОДЕЛИРОВАНИЕ ГЕНЕРАТОРОВ РАВНОМЕРНО РАСПРЕДЕЛЕННЫХ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ |
| по курсу: МОДЕЛИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4117 |  |  |  | Р.А.Сорокин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

**Цель работы:**

Получить навыки моделирования наиболее известных генераторов равномерно распределенных псевдослучайных чисел в программной среде MATLAB/GNU Octave, а также первичной оценки качества полученных псевдослучайных чисел.

**Ход работы:**

В ходе работы были рассмотрены различные варианты генераторов псевдослучайных последовательностей. Были разработаны классы, описывающие работу генераторов, созданы функции, вычисляющие метрики сгенерированных последовательностей чисел (Рисунок 1-3). Данные сгенерированные последовательности были записаны в .csv файл. Данная часть работы была выполнена в среде разработки Qt Creator версии 4.12.9, с использованием языка программирования C++, код программы представлен в приложении А.

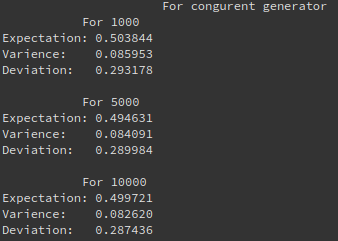


Рисунок 1 – Метрики для мультипликативного конгруэнтного генератора.

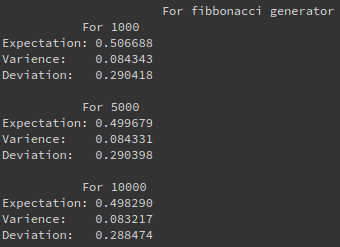


Рисунок 2 – Метрики для генератора Фибоначчи с запаздыванием.

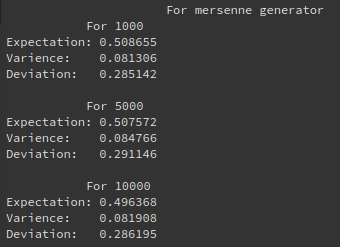


Рисунок 3 – Метрики для генератора вихря Мерсенна.

Далее где при помощи языка python были выведены графики: гистограмма (Рисунок 4), эмпирическая функция распределения (Рисунок 5), распределение на плоскости (Рисунок 6), график плотности распределения (Рисунок 7). Код программы представлен в приложении Б.

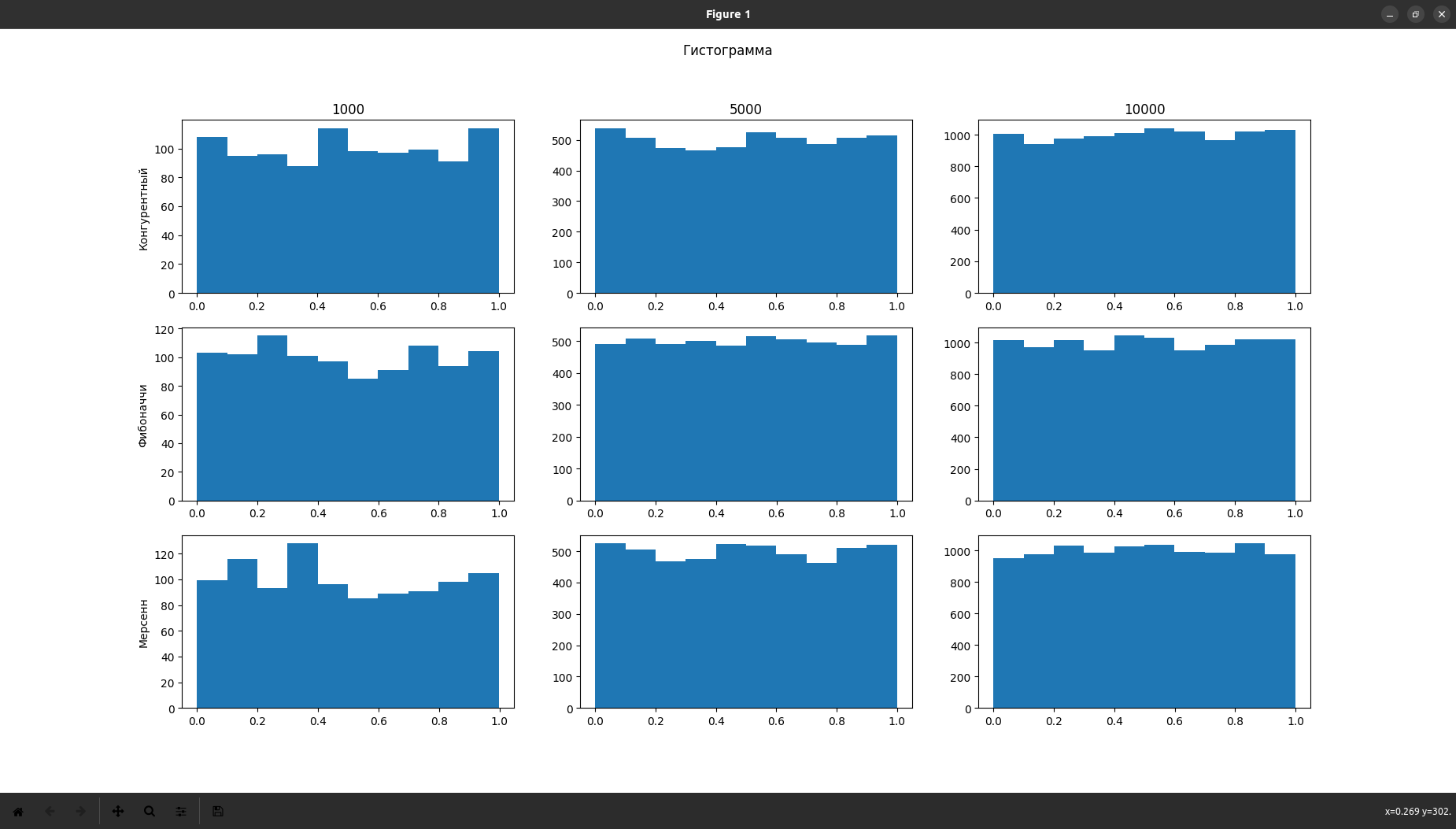


Рисунок 4 – Гистограмма.

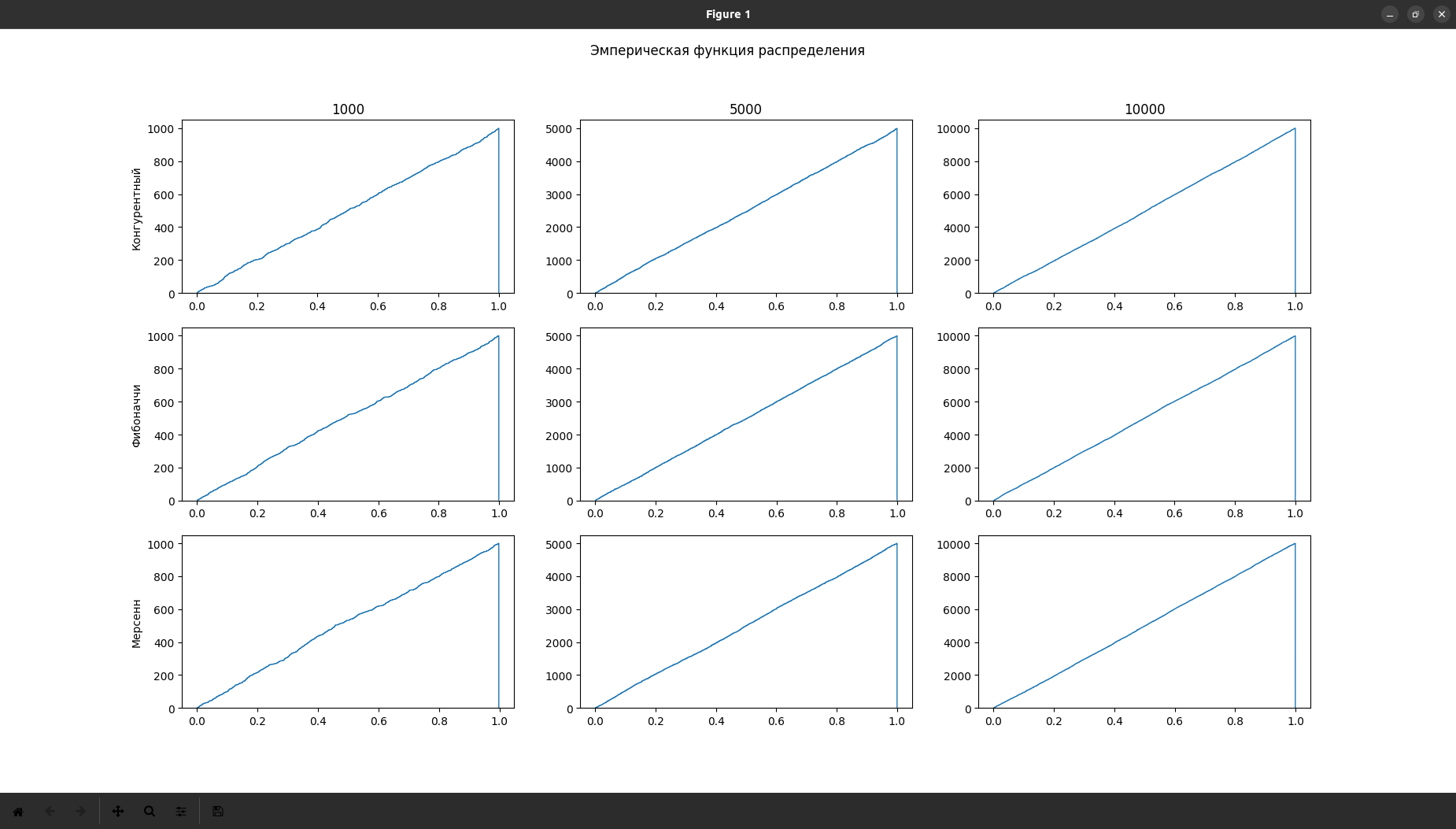


Рисунок 5 – Эмпирическая функция распределения.

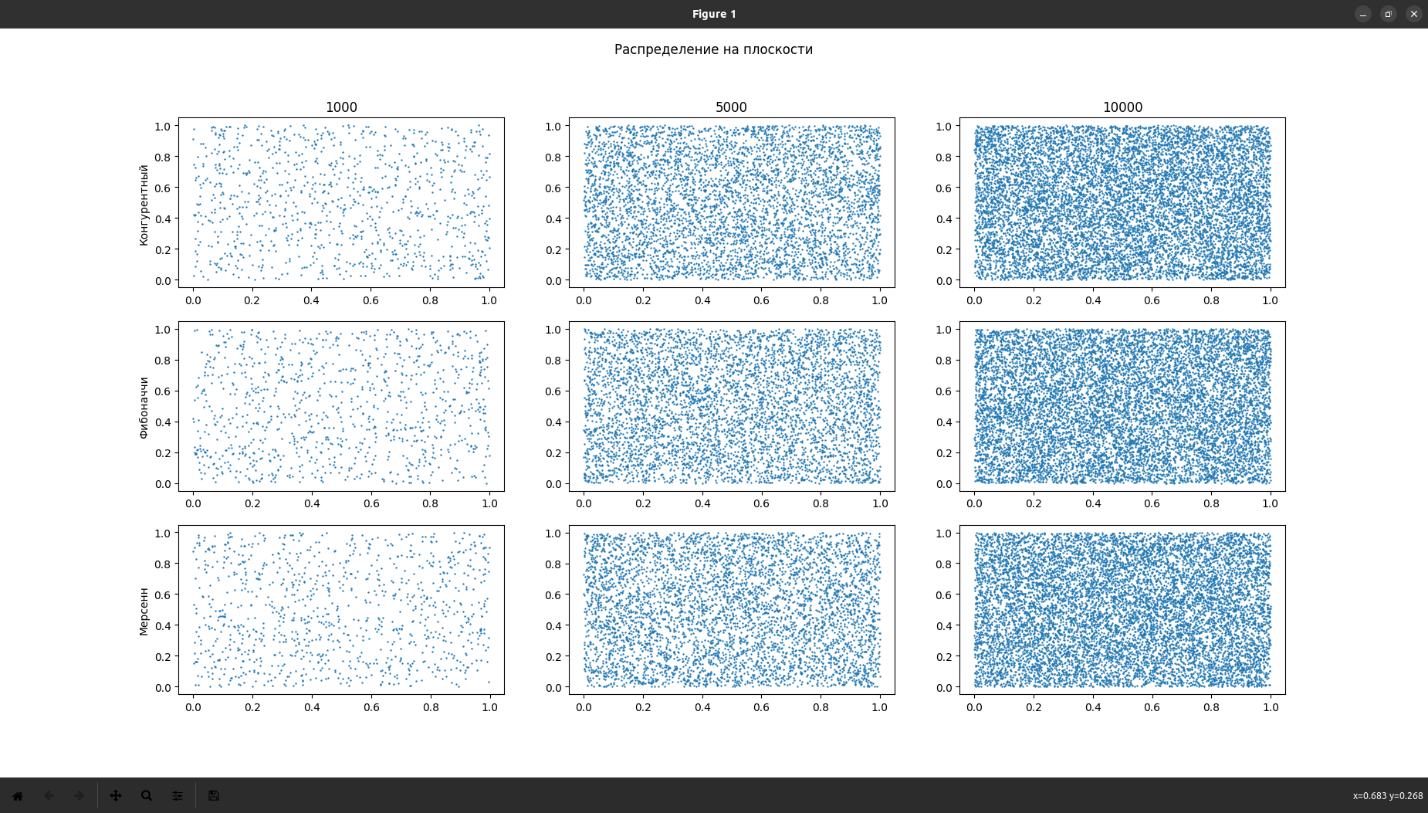


Рисунок 6 – Распределение на плоскости.

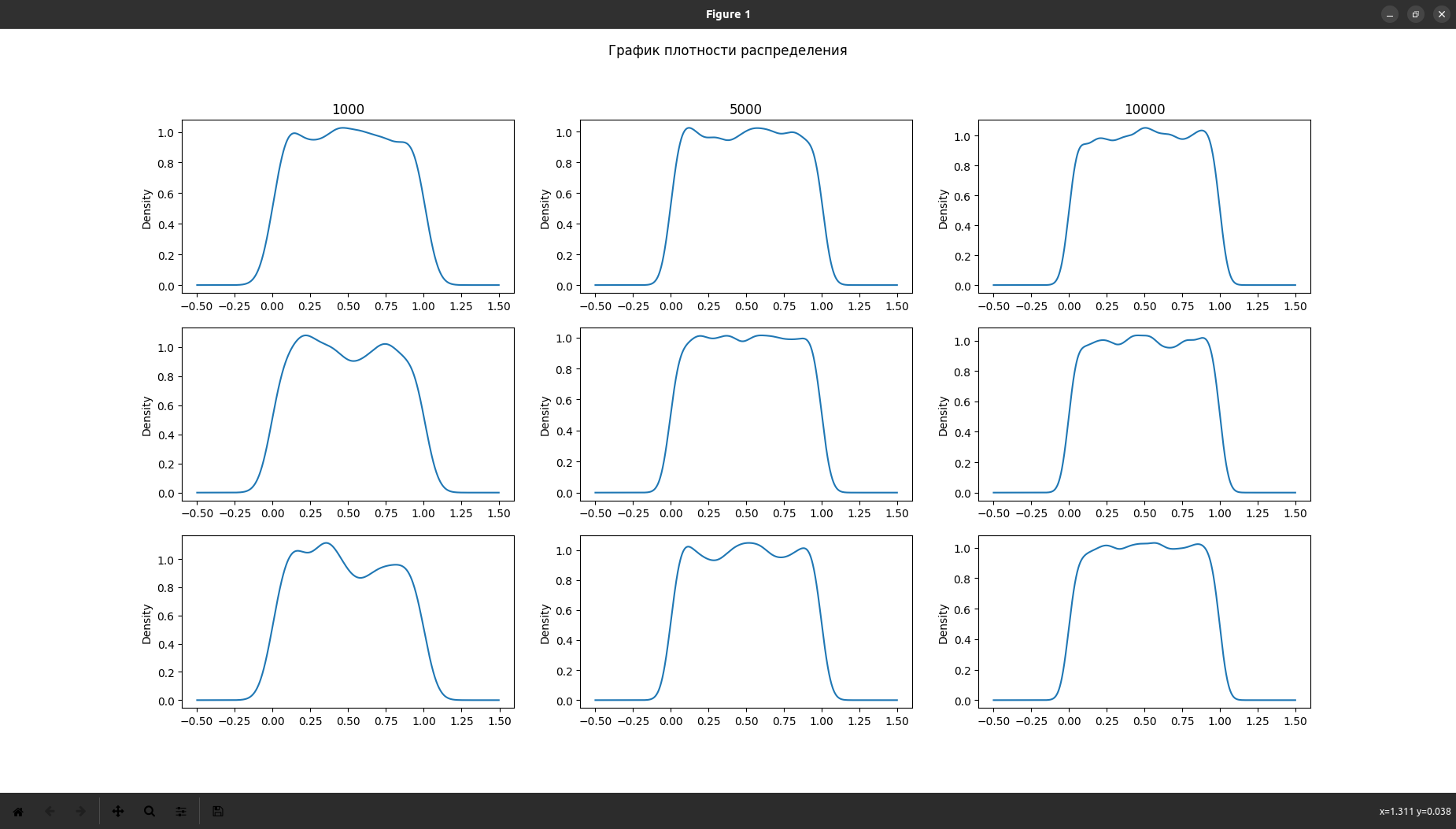


Рисунок 7 – График плотности распределения.

**Вывод:**

Получил навыки моделирования наиболее известных генераторов равномерно распределенных псевдослучайных чисел, а также первичной оценки качества полученных псевдослучайных чисел.

**ПРИЛОЖЕНИЕ А**

Листинг А.1 – Заголовочный файл интерфейса для генераторов generator\_interface.hpp

#ifndef \_\_GENERATOR\_INTERFACE\_HPP\_

#define \_\_GENERATOR\_INTERFACE\_HPP\_

namespace randGen

{

enum class GeneratorType

{

Congurent, Fibonacci, Mersenne,

};

class Generator

{

public:

virtual double rand() = 0;

virtual GeneratorType getType() = 0;

};

}//namespace randGen

#endif//\_\_GENERATOR\_INTERFACE\_HPP\_

Листинг А.2 – Заголовочный файл мультипликативного конгруэнтного генератора congurent\_generator.hpp

#ifndef \_\_CONGURENT\_GENERATOR\_HPP\_

#define \_\_CONGURENT\_GENERATOR\_HPP\_

#include "generator\_interface.hpp"

#include <cmath>

#include <cstdint>

namespace randGen

{

class Congurent : public Generator

{

private:

const uint16\_t A\_;

const uint16\_t C\_;

static const uint32\_t M\_ = -1;

double R\_;

public:

explicit Congurent(double seed = std::pow(2.,-52),

uint16\_t A = std::pow(7,5), uint16\_t C = 0);

double rand() override;

GeneratorType getType() override;

};

}//namespace randGen

#endif//\_\_CONGURENT\_GENERATOR\_HPP\_

Листинг А.3 – Файл исходников мультипликативного конгруэнтного генератора congurent\_generator.cpp

#include "congurent\_generator.hpp"

randGen::Congurent::Congurent(double seed,

uint16\_t A, uint16\_t C)

: A\_(A)

, C\_(C)

, R\_(seed)

{

}

double randGen::Congurent::rand()

{

R\_ = std::fmod(A\_\*R\_+C\_, M\_);

static double whole;

R\_ = std::modf(R\_,&whole);

return R\_;

}

randGen::GeneratorType randGen::Congurent::getType()

{

return GeneratorType::Congurent;

}

Листинг А.4 – Заголовочный файл генератора Фибоначчи fibonacci\_generator.hpp

#ifndef \_\_FIBONACCI\_GENERATOR\_HPP\_

#define \_\_FIBONACCI\_GENERATOR\_HPP\_

#include "generator\_interface.hpp"

#include "congurent\_generator.hpp"

#include <vector>

#include <algorithm>

#include <cstdint>

namespace randGen

{

class Fibonacci : public Generator

{

private:

Congurent initGen;

std::vector<double> R\_arr;

double& R\_;

const size\_t a\_;

const size\_t b\_;

public:

explicit Fibonacci(double seed = std::pow(2.,-52),

size\_t a = 63, size\_t b = 31);

double rand() override;

GeneratorType getType() override;

};

}//namespace randGen

#endif//\_\_FIBONACCI\_GENERATOR\_HPP\_

Листинг А.5 – Файл исходников генератора Фибоначчи fibonacci\_generator.cpp

#include "fibonacci\_generator.hpp"

randGen::Fibonacci::Fibonacci(double seed, size\_t a, size\_t b)

: initGen(seed)

, R\_arr(std::max(a,b) + 1)

, R\_(R\_arr[0])

, a\_(a)

, b\_(b)

{

for(auto& R : R\_arr)

R = initGen.rand();

}

double randGen::Fibonacci::rand()

{

for(size\_t i = R\_arr.size() - 1; i > 0; i--)

R\_arr[i] = R\_arr[i-1];

R\_ = R\_arr[a\_] - R\_arr[b\_];

if(R\_arr[a\_] < R\_arr[b\_])

R\_++;

return R\_;

}

randGen::GeneratorType randGen::Fibonacci::getType()

{

return GeneratorType::Fibonacci;

}

Листинг А.6 – Заголовочный файл генератора вихря Мерсенна mersenne\_twister.hpp

#ifndef \_\_MERSENNE\_TWISTER\_HPP\_

#define \_\_MERSENNE\_TWISTER\_HPP\_

#include "generator\_interface.hpp"

#include <random>

#include <cmath>

namespace randGen

{

class Mersenne : public Generator

{

private:

std::mt19937 generator;

std::uniform\_real\_distribution<double> distr;

public:

explicit Mersenne(double seed = std::pow(2.,-52));

double rand() override;

GeneratorType getType() override;

};

}//namespace randGen

#endif//\_\_MERSENNE\_TWISTER\_HPP\_

Листинг А.7 – Файл исходников генератора вихря Мерсенна mersenne\_twister.cpp

#include "mersenne\_twister.hpp"

randGen::Mersenne::Mersenne(double seed)

: generator(seed)

, distr(0.,1.)

{}

double randGen::Mersenne::rand()

{

return distr(generator);

}

randGen::GeneratorType randGen::Mersenne::getType()

{

return GeneratorType::Mersenne;

}

Листинг А.8 – Заголовочный файл библиотеки генераторов псевдослучайных чисел myrandom.hpp

#ifndef \_\_MY\_RANDOM\_HPP\_

#define \_\_MY\_RANDOM\_HPP\_

#include "generator\_interface.hpp"

#include "congurent\_generator.hpp"

#include "fibonacci\_generator.hpp"

#include "mersenne\_twister.hpp"

#endif//\_\_MY\_RANDOM\_HPP\_

Листинг А.9 – Заголовочный файл метрик для последовательности чисел metrics.hpp

#ifndef \_\_METRICS\_HPP\_

#define \_\_METRICS\_HPP\_

#include <vector>

#include <cmath>

#include <numeric>

#include <algorithm>

namespace metrics

{

double expectation(std::vector<double> const& arr);

double varience(std::vector<double> const& arr);

double deviation(std::vector<double> const& arr);

}//namespace mestics

#endif//\_\_METRICS\_HPP\_

Листинг А.10 – Файл исходников метрик для последовательности чисел metrics.cpp

#include "metrics.hpp"

double metrics::expectation(const std::vector<double> &arr)

{

double val = 0;

std::for\_each(arr.begin(),arr.end(),

[&val](auto n){

val+=n;

});

return val / static\_cast<double>(arr.size());

}

double metrics::varience(const std::vector<double> &arr)

{

const double M\_ = expectation(arr);

double val = 0;

std::for\_each(arr.begin(),arr.end(),

[&val,M\_](auto x){

val += (x - M\_) \* (x - M\_);

});

return val / static\_cast<double>(arr.size());

}

double metrics::deviation(const std::vector<double> &arr)

{

double val = std::sqrt(varience(arr));

return val;

}

Листинг А.11 – Файл исходников точки входа в программу main.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <unordered\_map>

#include <algorithm>

#include <cstdlib>

#include "metrics.hpp"

#include "myrandom.hpp"

void toFile(std::ofstream& stream, std::vector<double> const& arr);

void printMetrics(std::ostream& stream, randGen::Generator\* gen);

std::string getMetrics(std::vector<double> const& arr);

std::vector<double> genArray(randGen::Generator\* gen, size\_t size);

int main(int argc, char\* argv[])

{

if(argc != 1) return -1;

randGen::Congurent congurent;

randGen::Fibonacci fibonacci;

randGen::Mersenne mersenne;

printMetrics(std::cout,&congurent);

printMetrics(std::cout,&fibonacci);

printMetrics(std::cout,&mersenne);

std::string filePath = argv[0];

filePath = filePath.substr(0,filePath.size() - 4)

+ std::string("out.csv");

std::ofstream file(filePath);

file.precision(48);

for(size\_t i : {1'000,5'000,10'000})

toFile( file, genArray(&congurent,i) );

for(size\_t i : {1'000,5'000,10'000})

toFile( file, genArray(&fibonacci,i) );

for(size\_t i : {1'000,5'000,10'000})

toFile( file, genArray(&mersenne,i) );

file.close();

std::string sysCall = std::string("python3 script.py ") + filePath;

std::system(sysCall.c\_str());

return 0;

}

void toFile(std::ofstream& stream, std::vector<double> const& arr)

{

std::for\_each(arr.begin(), arr.end(),

[&stream](auto num){

stream << num << ", ";

});

stream << std::endl;

}

void printMetrics(std::ostream& stream, randGen::Generator\* gen)

{

static const std::unordered\_map<randGen::GeneratorType, std::string>

genTypeToStr

{

{ randGen::GeneratorType::Congurent , "congurent generator" },

{ randGen::GeneratorType::Fibonacci , "fibbonacci generator" },

{ randGen::GeneratorType::Mersenne , "mersenne generator" }

};

stream << "\n\t\tFor " << genTypeToStr.at(gen->getType())

<< std::endl;

for(size\_t i : {1'000,5'000,10'000})

{

stream << "\tFor " << i << "\n"

<< getMetrics( genArray(gen,i) )

<< "\n" << std::endl;

}

}

std::string getMetrics(std::vector<double> const& arr)

{

std::string metrics{};

metrics += std::string("Expectation: ")

+ std::to\_string(metrics::expectation(arr)) + "\n";

metrics += std::string("Varience: ")

+ std::to\_string(metrics::varience(arr)) + "\n";

metrics += std::string("Deviation: ")

+ std::to\_string(metrics::deviation(arr));

return metrics;

}

std::vector<double> genArray(randGen::Generator\* gen, size\_t size)

{

std::vector<double> arr{}; arr.reserve(size);

while(size--)

arr.push\_back(gen->rand());

return arr;

}

Листинг А.12 – Файл системы сборки проекта CMakeLists.txt

cmake\_minimum\_required(VERSION 3.5)

project(randomNumGen LANGUAGES CXX)

set(CMAKE\_CXX\_STANDARD 17)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

set(HEADERS\_GEN

include/myrandom.hpp

include/generator\_interface.hpp

include/fibonacci\_generator.hpp

include/congurent\_generator.hpp

include/mersenne\_twister.hpp

)

set(SOURCES\_GEN

src/congurent\_generator.cpp

src/fibonacci\_generator.cpp

src/mersenne\_twister.cpp

)

add\_library(randomgenlib ${HEADERS\_GEN} ${SOURCES\_GEN})

set(HEADERS\_MET

include/metrics.hpp

)

set(SOURCES\_MET

src/metrics.cpp

)

add\_library(metricslib ${HEADERS\_MET} ${SOURCES\_MET})

include\_directories(include)

add\_executable(main src/main.cpp)

target\_link\_libraries(main randomgenlib

metricslib

)

**ПРИЛОЖЕНИЕ Б**

Листинг Б.1 – Файл исходников скрипта script.py

import pandas as pd

import matplotlib.pyplot as plt

import sys

from copy import deepcopy

def printFoo(arr: list[list[float]]) -> None:

fig, axs = plt.subplots(3, 3)

fig.suptitle('Эмперическая функция распределения')

for i in range(9):

axs[i // 3, i % 3].hist(arr[i], histtype='step', cumulative=True, bins=len(arr[i]))

axs[0,0].set\_title('1000')

axs[0,1].set\_title('5000')

axs[0,2].set\_title('10000')

axs[0,0].set(ylabel='Конгурентный')

axs[1,0].set(ylabel='Фибоначчи')

axs[2,0].set(ylabel='Мерсенн')

plt.show()

def planGraph(arr: list[list[float]]) -> None:

arr\_ = deepcopy(arr)

for i in range(len(arr\_)):

arr\_[i].insert(0,2\*\*(-52))

arr\_[i].pop()

fig, axs = plt.subplots(3, 3)

fig.suptitle('Распределение на плоскости')

for i in range(9):

axs[i // 3, i % 3].scatter(arr[i],arr\_[i], s = 0.5)

axs[0,0].set\_title('1000')

axs[0,1].set\_title('5000')

axs[0,2].set\_title('10000')

axs[0,0].set(ylabel='Конгурентный')

axs[1,0].set(ylabel='Фибоначчи')

axs[2,0].set(ylabel='Мерсенн')

plt.show()

def kde(arr: list[list[float]])->None:

for i in range(len(arr)):

plt.subplot(3,3,i+1)

if(i == 0):

plt.title('1000')

plt.ylabel('Конгурентный')

if(i == 1):

plt.title('5000')

if(i == 2):

plt.title('10000')

if(i == 3):

plt.ylabel('Фибоначчи')

if(i == 6):

plt.ylabel('Мерсенн')

s = pd.Series(arr[i])

s.plot.kde()

plt.show()

def main(filePath: str) -> None:

arr = []

with open(filePath) as file:

samp = [i.split(', ')[:-1] for i in file.read().splitlines()]

for line in samp:

arr.append([float(i) for i in line])

#printFoo(arr)

#planGraph(arr)

kde(arr)

if \_\_name\_\_ == "\_\_main\_\_":

if (len(sys.argv) < 2):

sys.exit("ERROR filepath is undefinded")

main(sys.argv[1])