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.

```
                    For congurent generator
          For 1000
Expectation: 0.503844
Varience:    0.085953
Deviation:   0.293178

          For 5000
Expectation: 0.494631
Varience:    0.084091
Deviation:   0.289984

          For 10000
Expectation: 0.499721
Varience:    0.082620
Deviation:   0.287436
```

Рисунок 1 – Метрики для мультипликативного конгруэнтного генератора.

```
                    For fibbonacci generator
          For 1000
Expectation: 0.506688
Varience:    0.084343
Deviation:   0.290418

          For 5000
Expectation: 0.499679
Varience:    0.084331
Deviation:   0.290398

          For 10000
Expectation: 0.498290
Varience:    0.083217
Deviation:   0.288474
```

Рисунок 2 – Метрики для генератора Фибоначчи с запаздыванием.

```
                    For mersenne generator
           For 1000
Expectation: 0.508655
Varience:    0.081306
Deviation:   0.285142

           For 5000
Expectation: 0.507572
Varience:    0.084766
Deviation:   0.291146

           For 10000
Expectation: 0.496368
Varience:    0.081908
Deviation:   0.286195
```

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.

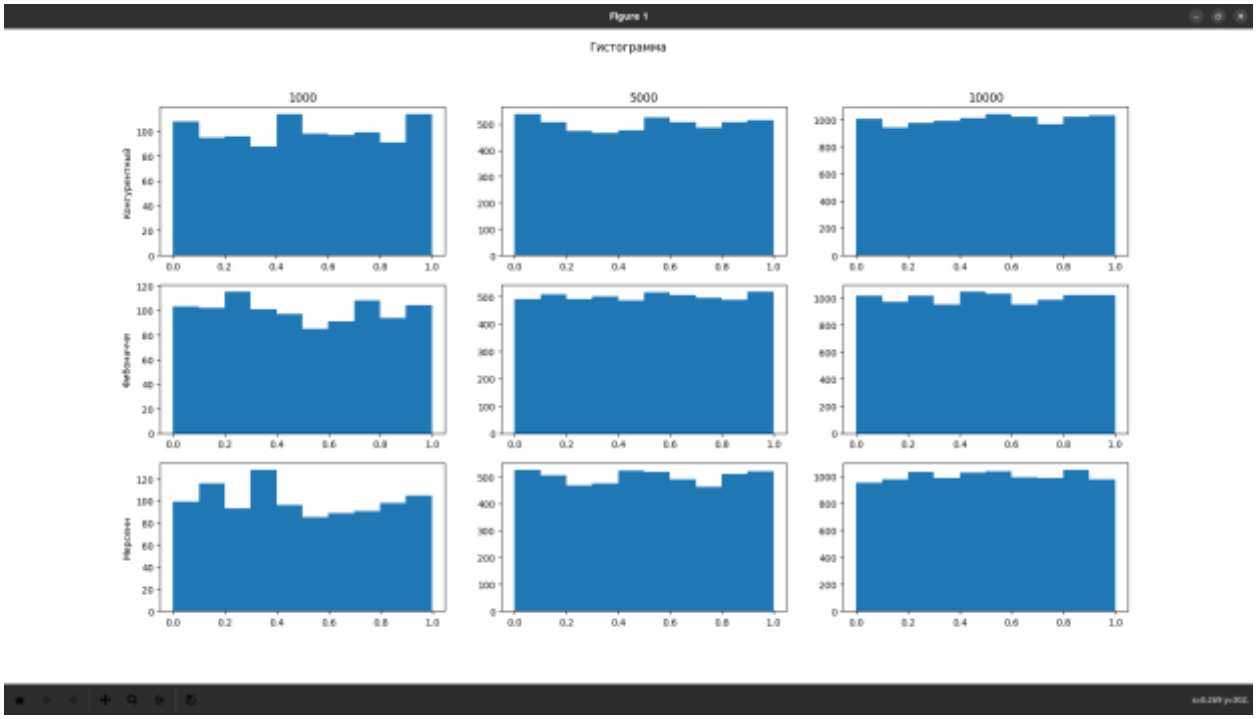

Рисунок 4 – Гистограмма.

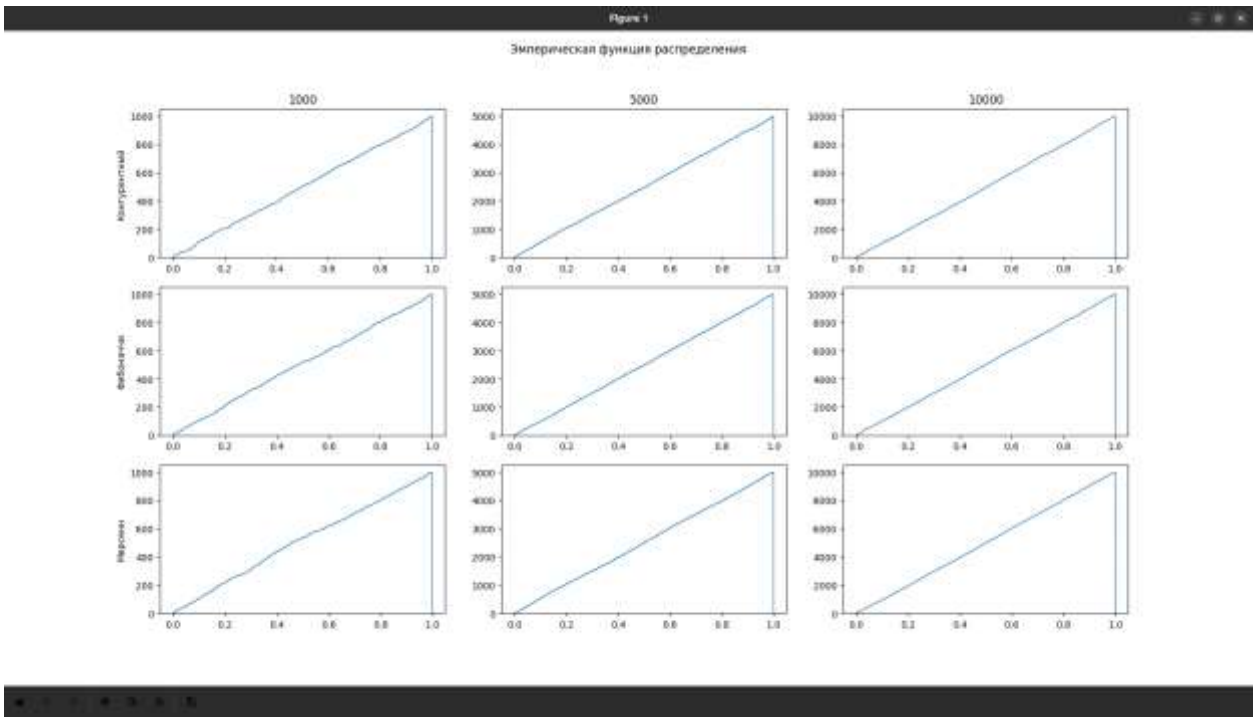

Рисунок 5 – Эмпирическая функция распределения.

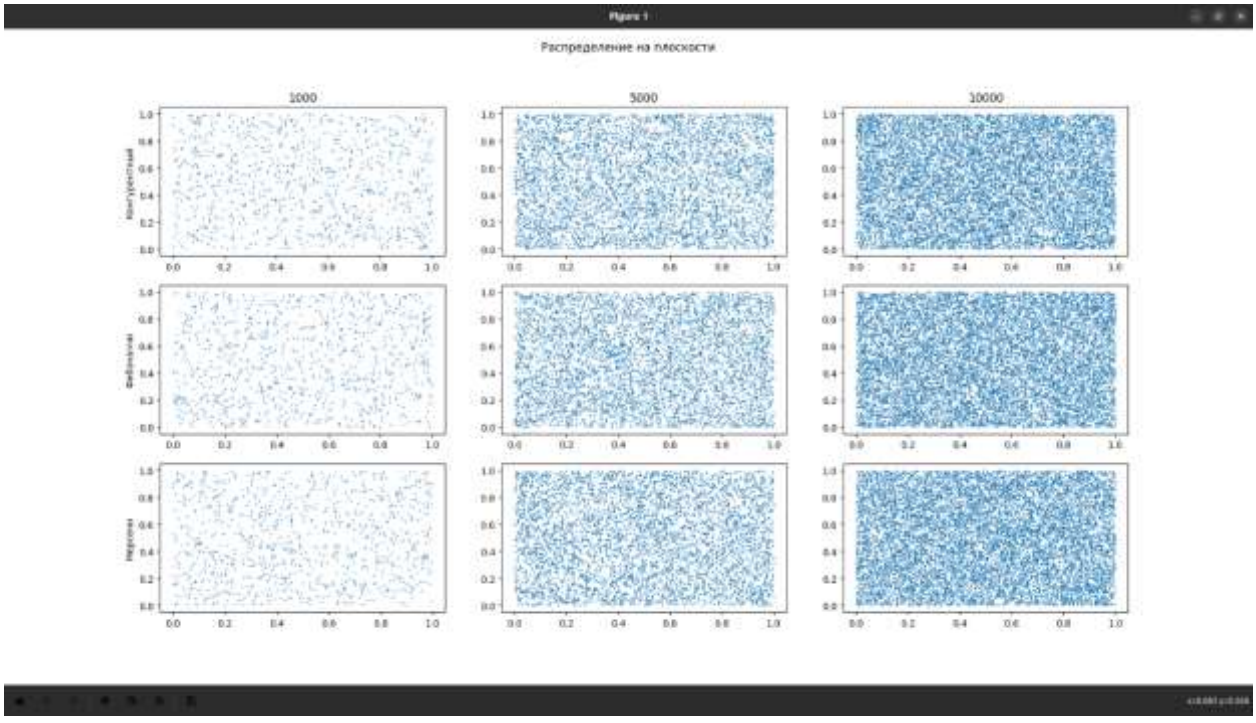

Рисунок 6 – Распределение на плоскости.

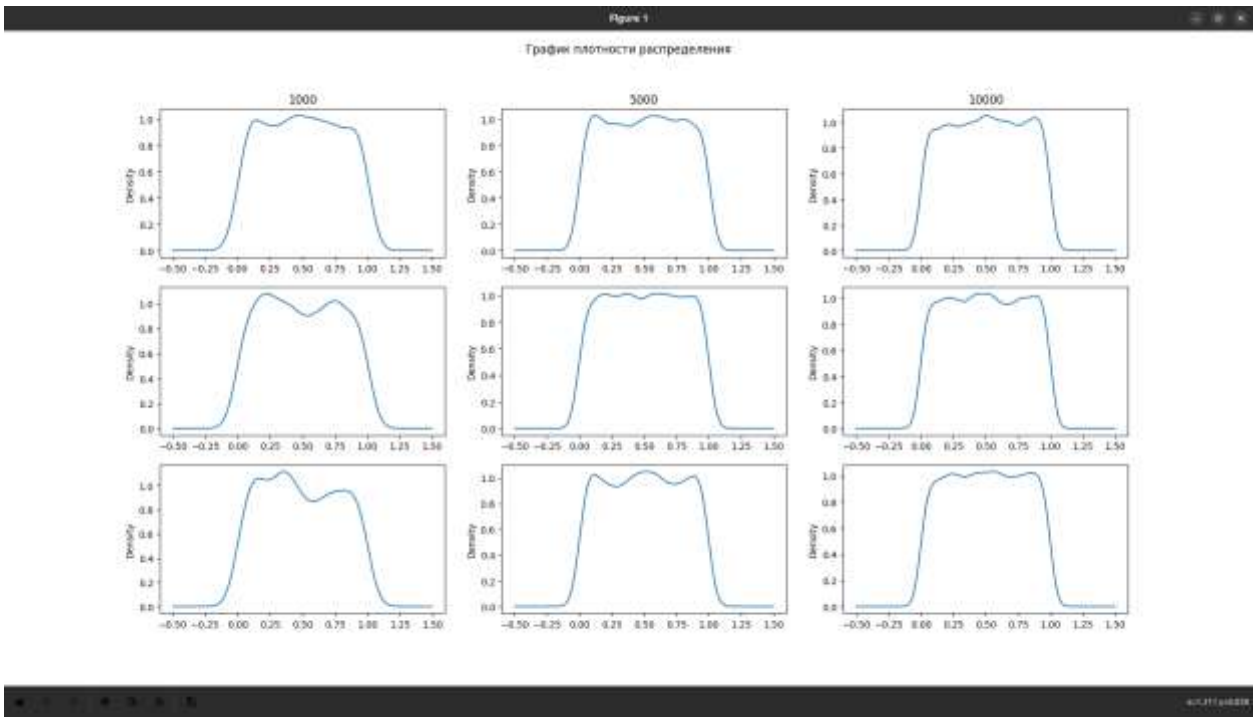

Рисунок 7 – График плотности распределения.

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])
```