Лютов Александр Владимирович №в5130904/30022

**Вариант 1**

Значение y(t) – было взято из 1 работы и равняется 15.6204

Кол-во сгенерированных чисел – 100 000

**Описание стандартного генератора:**

* std::random\_device - Генератор истинно случайных чисел
* std::mt19937 - Псевдослучайный генератор на основе алгоритма Mersenne Twister
* std::uniform\_real\_distribution<> - Равномерное распределение вещественных чисел в заданном диапазоне

**Подсчет параметров (𝑚𝑟 , 𝐷𝑟 , σr ) и сравнение с теоретическими:**

Вычисленное мат. ожидание: 0.499945 (теоретическое: 0.500000) Вычисленная дисперсия: 0.083331 (теоретическая: 0.083333)

Вычисленное СКО: 0.288672 (теоретическое: 0.288675)

**Частотная диаграмма:**

**Описание собственного ГСЧ:**

Для реализации собственной ГСЧ был выбран метод Мюллера, так как он:

* Точно преобразует равномерное распределение в нормальное.
* Эффективен (генерирует числа парами).
* Позволяет задавать произвольные параметры (m\_x) и (sigma\_x)

Каждый вызов generate() возвращает одно число, сохраняя второе число пары для следующего вызова

**Подсчет параметров (𝑚𝑟 , 𝐷𝑟 , σr ) и сравнение с теоретическими:**

Параметры распределения: mx = 0.000000, σx = 0.781020

Вычисленное мат. ожидание: 0.001288 (теоретическое: 0.000000)

Вычисленная дисперсия: 0.612953 (теоретическая: 0.609992)

Вычисленное СКО: 0.782913 (теоретическое: 0.781020)

**Частотная диаграмма:**

**Код программы:**

main.cpp

#include <iostream>

#include <vector>

#include <cmath>

#include <iomanip>

#include <fstream>

#include "normaldistributiongenerator.h"

// Функция для записи вектора чисел в файл (по одному значению на строку)

void write\_to\_file(const std::vector<double>& values, const std::string& filename) {

std::ofstream outfile(filename);

if (!outfile) {

std::cerr << "Ошибка открытия файла " << filename << " для записи" << std::endl;

return;

}

outfile << std::fixed << std::setprecision(15);

for (const auto& value : values) {

outfile << value << '\n';

}

outfile.close();

}

void test\_builtin\_rng() {

const int n = 100000;

std::vector<double> numbers(n);

// Инициализация генератора случайных чисел

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_real\_distribution<> dis(0.0, 1.0);

// Генерация чисел и вычисление суммы

double sum = 0.0;

for (int i = 0; i < n; ++i) {

numbers[i] = dis(gen);

sum += numbers[i];

}

// Запись в файл

write\_to\_file(numbers, "builtin\_rng\_values.txt");

// Вычисление мат. ожидания

double m\_r = sum / n;

// Вычисление дисперсии и СКО

double variance = 0.0;

for (int i = 0; i < n; ++i) {

variance += (numbers[i] - m\_r) \* (numbers[i] - m\_r);

}

variance /= n;

double sigma\_r = std::sqrt(variance);

// Теоретические значения

const double theoretical\_m = 0.5;

const double theoretical\_variance = 1.0 / 12.0;

const double theoretical\_sigma = std::sqrt(theoretical\_variance);

// Вывод результатов

std::cout << "=== Проверка встроенного ГСЧ ===" << std::endl;

std::cout << std::fixed << std::setprecision(6);

std::cout << "Количество чисел: " << n << std::endl;

std::cout << "Вычисленное мат. ожидание: " << m\_r << " (теоретическое: " << theoretical\_m << ")" << std::endl;

std::cout << "Вычисленная дисперсия: " << variance << " (теоретическая: " << theoretical\_variance << ")" << std::endl;

std::cout << "Вычисленное СКО: " << sigma\_r << " (теоретическое: " << theoretical\_sigma << ")" << std::endl;

}

void test\_custom\_rng() {

const int n = 100000;

const double mx = 0.0;

const double max\_y\_table = 15.6204;

const double sigma\_x = 0.05 \* max\_y\_table;

NormalDistributionGenerator generator;

std::vector<double> numbers(n);

// Генерация чисел и вычисление суммы

double sum = 0.0;

for (int i = 0; i < n; ++i) {

numbers[i] = generator.generate(mx, sigma\_x);

sum += numbers[i];

}

// Запись в файл

write\_to\_file(numbers, "custom\_rng\_values.txt");

// Вычисление мат. ожидания

double m\_x = sum / n;

// Вычисление дисперсии и СКО

double variance\_x = 0.0;

for (int i = 0; i < n; ++i) {

variance\_x += (numbers[i] - m\_x) \* (numbers[i] - m\_x);

}

variance\_x /= n;

double sigma\_x\_calculated = std::sqrt(variance\_x);

// Теоретические значения

const double theoretical\_mx = mx;

const double theoretical\_sigma\_x = sigma\_x;

const double theoretical\_variance\_x = sigma\_x \* sigma\_x;

// Вывод результатов

std::cout << "\n=== Проверка собственного ГСЧ ===" << std::endl;

std::cout << std::fixed << std::setprecision(6);

std::cout << "Количество чисел: " << n << std::endl;

std::cout << "Параметры распределения: mx = " << mx << ", σx = " << sigma\_x << std::endl;

std::cout << "Вычисленное мат. ожидание: " << m\_x << " (теоретическое: " << theoretical\_mx << ")" << std::endl;

std::cout << "Вычисленная дисперсия: " << variance\_x << " (теоретическая: " << theoretical\_variance\_x << ")" << std::endl;

std::cout << "Вычисленное СКО: " << sigma\_x\_calculated << " (теоретическое: " << theoretical\_sigma\_x << ")" << std::endl;

}

int main()

{

// Проверка встроенного ГСЧ

test\_builtin\_rng();

// Тестирование собственного ГСЧ

test\_custom\_rng();

return 0;

}

Normaldistributiongenerator.cpp

#include "normaldistributiongenerator.h"

NormalDistributionGenerator::NormalDistributionGenerator() : gen(rd()), dis(0.0, 1.0), hasSpare(false) {}

double NormalDistributionGenerator::generate(double mean, double sigma)

{

if (hasSpare) {

hasSpare = false;

return spare \* sigma + mean;

}

double u, v, s;

do {

u = dis(gen) \* 2.0 - 1.0;

v = dis(gen) \* 2.0 - 1.0;

s = u \* u + v \* v;

} while (s >= 1.0 || s == 0.0);

s = std::sqrt(-2.0 \* std::log(s) / s);

spare = v \* s;

hasSpare = true;

return u \* s \* sigma + mean;

}

Normaldistributiongenerator.h

#ifndef NORMALDISTRIBUTIONGENERATOR\_H

#define NORMALDISTRIBUTIONGENERATOR\_H

#include <cmath>

#include <random>

class NormalDistributionGenerator

{

public:

explicit NormalDistributionGenerator();

double generate(double mean = 0.0, double sigma = 1.0);

private:

std::random\_device rd;

std::mt19937 gen;

std::uniform\_real\_distribution<> dis;

bool hasSpare;

double spare;

};

#endif // NORMALDISTRIBUTIONGENERATOR\_H