Лютов Александр Владимирович №в5130904/30022

Вариант 31

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

Кол-во сгенерированных чисел – 100 000

Описание стандартного генератора:

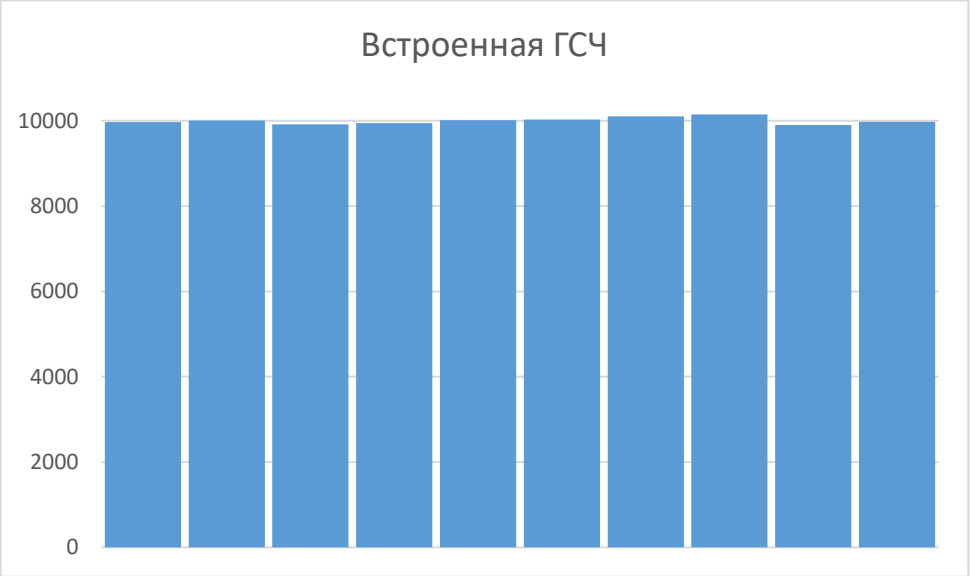
- std::random_device - Генератор истинно случайных чисел
- std::mt19937 - Псевдослучайный генератор на основе алгоритма Mersenne Twister
- std::uniform_real_distribution<> - Равномерное распределение вещественных чисел в заданном диапазоне

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:

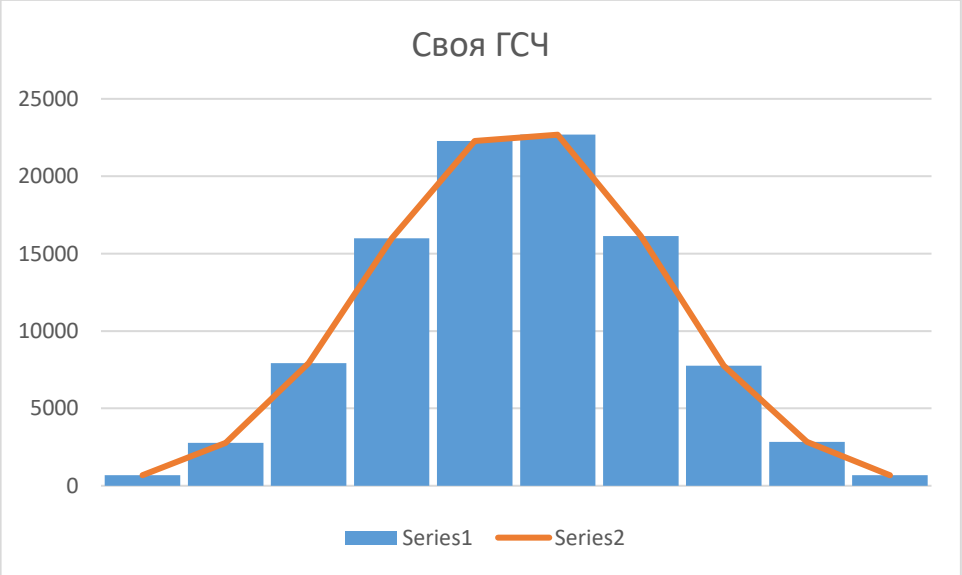
Параметры распределения: mx = 0.000000, σx = 0.781020

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

```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

```cpp
#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
```