МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**Лабораторная работа №3**

**«**Изучение циклических алгоритмов, операторов цикла, программирование циклического вычислительного процесса**»**

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-322», «АВТФ»  *Писарев Михаил Вячеславович*  «3» мая 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Глеб Андреевич*  «3» мая 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

**Цели и задачи работы**:изучение циклических алгоритмов, операторов цикла, программирование циклического вычислительного процесса.

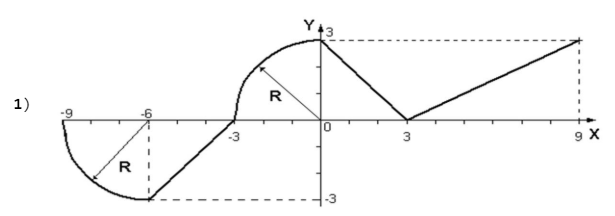
**Задание к работе:** Реализовать циклический вычислительный процесс. Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

**Задание 1.** Вычислить и вывести на экран или в файл в виде таблицы значения функции, заданной графически, на интервале от Xнач до Xкон с шагом dx. Интервал и шаг задать таким образом, чтобы проверить все ветви программы. Таблица должна иметь заголовком и шапку.

**Задание 2.** Тесты на простоту (реализовать метод перебора делителей, тесты Миллера, Поклингтона, ГОСТ Р 34.10-94

**Задание 3\*.** Решить задачу об остывании чашки кофе из кофейни НГТУ. Написать программу, моделирующую процесс остывания кофе. Программа должна позволять задавать все необходимые параметры. Построить графики или таблицы зависимостей величин. Считаем, что температуру кофе измеряли через некоторые, необязательно равномерные промежутки времени. Написать программу, которая бы по данным измерений строила линейную модель, вычисляя статистические критерии их адекватности.

**Индивидуальное задание 1**:



**Задание 2:**

1. Построить таблицу простых чисел, меньших 500, с помощью решета Эратосфена. С использованием этой таблицы:

а) реализовать процедуру получения простых чисел заданной длины на основе теста Миллера;

б) реализовать процедуру получения простых чисел заданной длины на основе теста Поклингтона;

в) реализовать процедуру генерации простых чисел заданной длины ГОСТ Р 34.10-94.

**Примечания к выводу в задании 2:**

Для теста Миллера и теста Поклингтона:

Выводится таблица простых чисел в формате:

Первая строка - № построенного числа.

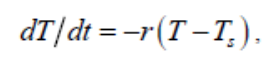
Вторая строка - построенное и проверенное тестом миллера число с вероятностью ошибки < 0.1.

Третья строка - резулльтат проверки вероятностынм тестом(если ‘+’, то вероятностный тест определил число как простое, если ‘-’, то вероятностный тест определил число как составное).

Четвёртая строка - параметр k, который отображает количество чисел отвергнутых тестом как составные с шансом ошибки < 0.1, но принятых за простое число вероятностным тестом.

**Задание 3:**

Природа переноса тепла от кофе к окружающему пространству сложна и в общем случае включает в себя механизмы конвекции, излучения, испарения и теплопроводности. В том случае, когда разность температур между объектом и окружающей средой не очень велика, скорость изменения Т объекта можно считать пропорциональной этой разности температур. Это утверждение формулируется на языке дифференциальных уравнений так



где Т – температура кофе, Ts – температура окружающей среды, r – коэффициент остывания. Коэффициент остывания зависит от механизма передачи тепла, площади тела, находящегося в контакте со средой и тепловых свойств самого тела. Знак (–) появляется во избежание нефизического эффекта увеличения температуры тела, когда T > Ts. Данное соотношение называется законом теплопроводности Ньютона.

**Выполнение задания 1**

**Листинг программы на C++:**

#include <iostream>

#include <cmath>

using namespace std;

// Функция для вычисления значения y на основе заданного графика

double calculateY(double x) {

// Проверяем специальные случаи для x = -9 и x = 9

if (x == -9) {

return 0; // Значение y в точке x = -9 равно 0

}

else if (x == 9) {

return 3; // Значение y в точке x = 9 равно 3

}

if (x <= -9 || x >= 9) {

return NAN; // Неопределенное значение для x вне заданного интервала

}

else if (x < -6) {

// Дуга от (-9,0) до (-6,-3)

return sqrt(9 - pow(x + 9, 2)) \* -1; // умножение на -1 для инверсии y

}

else if (x < -3) {

// Прямая линия от (-6,-3) до (-3,0)

return x + 3;

}

else if (x <= 0) {

// Дуга от (-3,0) до (0,3)

return sqrt(9 - pow(x, 2));

}

else if (x <= 3) {

// Прямая линия от (0,3) до (3,0)

return -1 \* (3.0 / 3.0) \* x + 3; // уравнение прямой y = kx + b, k = -3/3, b = 3

}

else {

// Прямая линия от (3,0) до (9,3)

return (3.0 / 6.0) \* (x - 3); // уравнение прямой y = kx + b, k = 3/6, b = 0

}

}

int main() {

// Задаем интервал и шаг

double startX = -9.0;

double endX = 9.0;

double step = 1;

// Выводим заголовок таблицы

cout << "X\t\tY" << endl;

// Выводим значения функции для заданного интервала с заданным шагом

for (double x = startX; x <= endX; x += step) {

double y = calculateY(x);

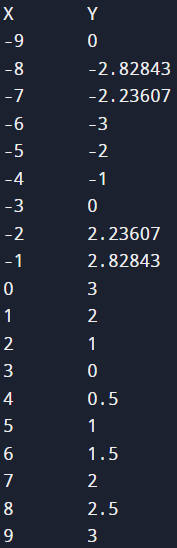
cout << x << "\t\t" << y << endl;

}

return 0;

}

Рисунок 1: Результат работы программы на С++



**Листинг программы на Java:**

public class Main {

// Метод для вычисления значения y на основе заданного графика

public static double calculateY(double x) {

// Проверяем специальные случаи для x = -9 и x = 9

if (x == -9) {

return 0; // Значение y в точке x = -9 равно 0

} else if (x == 9) {

return 3; // Значение y в точке x = 9 равно 3

}

if (x <= -9 || x >= 9) {

return Double.NaN; // Неопределенное значение для x вне заданного интервала

} else if (x < -6) {

// Дуга от (-9,0) до (-6,-3)

return -Math.sqrt(9 - Math.pow(x + 9, 2)); // умножение на -1 для инверсии y

} else if (x < -3) {

// Прямая линия от (-6,-3) до (-3,0)

return x + 3;

} else if (x <= 0) {

// Дуга от (-3,0) до (0,3)

return Math.sqrt(9 - Math.pow(x, 2));

} else if (x <= 3) {

// Прямая линия от (0,3) до (3,0)

return -1 \* (3.0 / 3.0) \* x + 3; // уравнение прямой y = kx + b, k = -3/3, b = 3

} else {

// Прямая линия от (3,0) до (9,3)

return (3.0 / 6.0) \* (x - 3); // уравнение прямой y = kx + b, k = 3/6, b = 0

}

}

public static void main(String[] args) {

// Задаем интервал и шаг

double startX = -9.0;

double endX = 9.0;

double step = 1;

// Выводим заголовок таблицы

System.out.println("X\t\tY");

// Выводим значения функции для заданного интервала с заданным шагом

for (double x = startX; x <= endX; x += step) {

double y = calculateY(x);

System.out.println(x + "\t\t" + y);

}

}

}

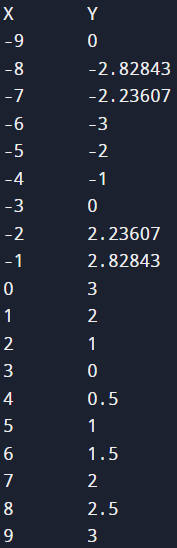


Рисунок 2: Результат работы программы на Java

**Выполнение задания 2 тест Миллера**

**Листинг программы на C++:**

#include <iostream>

#include <vector>

#include <tuple>

#include <cmath>

#include <random>

#include <limits>

#include <algorithm>

using namespace std;

// Функция решета Эратосфена для нахождения всех простых чисел до n

vector<int> primes(int n);

// Функция для построения числа и проведения теста

pair<int, vector<int>> builder\_test(vector<int> prime, int bit);

// Тест Миллера для проверки числа на простоту

int test\_millera(int n, int t, vector<int> q);

// Функция для возведения числа a в степень b по модулю n

int power\_mod(int a, int b, int n);

// Функция для генерации случайного числа в диапазоне [a, b]

int rn(int a, int b);

// Функция для вывода результатов

void print\_results(const vector<int>& res, const vector<string>& res\_ver\_test, const vector<int>& otvegnutie);

int main() {

int size\_primes = 500; // Размер для генерации простых чисел

vector<int> prime = primes(size\_primes); // Генерация простых чисел

int bit = 0; // Размер числа в битах

cin >> bit; // Ввод размера числа

vector<int> q; // Вектор для хранения множителей

int n; // Сгенерированное число

int k = 0; // Счетчик отвергнутых чисел

int probability; // Вероятность простоты

vector<int> res; // Вектор для хранения результатов

vector<string> res\_ver\_test; // Вектор для хранения результатов теста

vector<int> otvegnutie; // Вектор для хранения количества отвергнутых чисел

// Генерация 10 различных простых чисел

while (res.size() != 10) {

// Построение числа и проведение теста

tie(n, q) = builder\_test(prime, bit);

probability = test\_millera(n, 10, q);

// Проверка на простоту

if (probability == 1) {

// Проверка на уникальность числа

if (find(res.begin(), res.end(), n) == res.end()) {

res.push\_back(n); // Добавление числа в результаты

// Дополнительная проверка числа

probability = test\_millera(n, 1, q);

if (probability == 1) {

res\_ver\_test.push\_back("+"); // Число прошло проверку

} else {

res\_ver\_test.push\_back("-"); // Число не прошло проверку

}

otvegnutie.push\_back(k); // Добавление количества отвергнутых чисел

k = 0; // Сброс счетчика отвергнутых чисел

}

} else {

k++; // Увеличение счетчика отвергнутых чисел

}

}

// Вывод результатов

print\_results(res, res\_ver\_test, otvegnutie);

}

// Функция решета Эратосфена для нахождения всех простых чисел до n

vector<int> primes(int n) {

vector<bool> is\_prime(n + 1, true); // Вектор для отметки простоты чисел

vector<int> primes; // Вектор для хранения простых чисел

// Реализация решета Эратосфена

for (int p = 2; p \* p <= n; ++p) { // Для всех чисел от 2 до sqrt(n)

if (is\_prime[p]) { // Если число p простое

for (int i = p \* p; i <= n; i += p) // Удаление кратных чисел p

is\_prime[i] = false;

}

}

// Сбор всех простых чисел в вектор

for (int p = 2; p <= n; ++p) {

if (is\_prime[p])

primes.push\_back(p);

}

return primes; // Возвращение вектора простых чисел

}

// Функция для построения числа и проведения теста

pair<int, vector<int>> builder\_test(vector<int> prime, int bit) {

int max\_index = 0; // Максимальный индекс в векторе простых чисел

int max\_pow = 1; // Максимальная степень двойки

// Нахождение максимального индекса, такого что prime[max\_index] < 2^(bit - 1)

for (; (prime[max\_index] < pow(2, bit - 1)) && max\_index < prime.size(); max\_index++);

// Нахождение максимальной степени двойки, такой что 2^max\_pow < 2^(bit - 1)

for (; pow(2, max\_pow) < pow(2, bit - 1); max\_pow++);

int m = 1; // Начальное значение m

vector<int> q; // Вектор для хранения множителей

// Построение числа m

while (true) {

int num = rn(0, max\_index); // Случайный индекс простого числа

int power = rn(1, max\_pow); // Случайная степень

// Проверка на переполнение и добавление множителя

if (pow(prime[num], power)) {

if (m \* pow(prime[num], power) < INT32\_MAX) {

m \*= pow(prime[num], power);

q.push\_back(prime[num]);

}

}

// Проверка на соответствие условиям

if (m > pow(2, bit - 2)) {

if (m >= pow(2, bit - 1)) {

m = 1;

q.clear();

} else {

break;

}

}

}

int n = 2 \* m + 1; // Инициализация числа n

return make\_pair(n, q); // Возвращение числа и множителей

}

// Тест Миллера для проверки числа на простоту

int test\_millera(int n, int t, vector<int> q) {

vector<int> a; // Вектор для хранения случайных чисел

int aj; // Случайное число

// Генерация t случайных чисел

while (a.size() != t) {

aj = rn(2, n - 1);

if (find(a.begin(), a.end(), aj) == a.end()) {

a.push\_back(aj);

}

}

// Первоначальная проверка простоты

for (int aj : a) {

if (power\_mod(aj, n - 1, n) != 1) {

return 0;

break;

}

}

// Дополнительная проверка простоты

bool flag = true;

int i = 0;

for (int aj : a) {

if (q[i] != 0 && power\_mod(aj, (n - 1) / q[i], n) != 1) {

flag = false;

if (i < q.size()) {

i++;

}

}

}

// Проверка результата

if (flag) {

return 0;

}

return 1;

}

// Функция для возведения числа a в степень b по модулю n

int power\_mod(int a, int b, int n) {

long long result = 1; // Начальное значение результата

while (b > 0) { // Пока степень больше 0

if (b % 2 == 1) // Если степень нечетная

result = (result \* a) % n; // Умножение результата на a по модулю n

a = (a \* a) % n; // Возведение a в квадрат по модулю n

b /= 2; // Деление степени на 2

}

return result; // Возвращение результата

}

// Функция для генерации случайного числа в диапазоне [a, b]

int rn(int a, int b) {

mt19937 mt\_rand(random\_device{}()); // Инициализация генератора случайных чисел

return uniform\_int\_distribution<int>(a, b)(mt\_rand); // Возвращение случайного числа

}

// Функция для вывода результатов

void print\_results(const vector<int>& res, const vector<string>& res\_ver\_test, const vector<int>& otvegnutie) {

cout << "Prime Numbers\tTest Results\tOccurrences" << endl; // Заголовок таблицы

cout << "----------------------------------------------" << endl; // Разделитель

// Вывод каждого результата

for (size\_t i = 0; i < res.size(); ++i) {

cout << res[i] << "\t\t" << res\_ver\_test[i] << "\t\t" << otvegnutie[i] << endl;

}

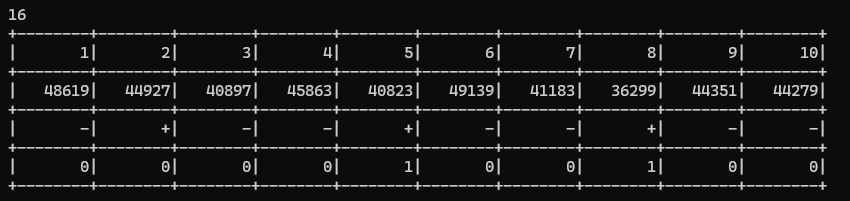
}

Рисунок 3: Результат работы программы на С++ при требуемоей размерности простого числа 16.

**Листинг программы на Java:**

import java.util.\*;

import java.util.stream.IntStream;

public class Main {

public static void main(String[] args) {

int sizePrimes = 500; // Размер для генерации простых чисел

List<Integer> prime = primes(sizePrimes); // Генерация простых чисел

Scanner scanner = new Scanner(System.in);

int bit = scanner.nextInt(); // Ввод размера числа в битах

List<Integer> q; // Вектор для хранения множителей

int n; // Сгенерированное число

int k = 0; // Счетчик отвергнутых чисел

int probability; // Вероятность простоты

List<Integer> res = new ArrayList<>(); // Вектор для хранения результатов

List<String> resVerTest = new ArrayList<>(); // Вектор для хранения результатов теста

List<Integer> otvegnutie = new ArrayList<>(); // Вектор для хранения количества отвергнутых чисел

// Генерация 10 различных простых чисел

while (res.size() != 10) {

// Построение числа и проведение теста

Pair<Integer, List<Integer>> result = builderTest(prime, bit);

n = result.getKey();

q = result.getValue();

probability = testMillera(n, 10, q);

// Проверка на простоту

if (probability == 1) {

// Проверка на уникальность числа

if (!res.contains(n)) {

res.add(n); // Добавление числа в результаты

// Дополнительная проверка числа

probability = testMillera(n, 1, q);

if (probability == 1) {

resVerTest.add("+"); // Число прошло проверку

} else {

resVerTest.add("-"); // Число не прошло проверку

}

otvegnutie.add(k); // Добавление количества отвергнутых чисел

k = 0; // Сброс счетчика отвергнутых чисел

}

} else {

k++; // Увеличение счетчика отвергнутых чисел

}

}

// Вывод результатов

printResults(res, resVerTest, otvegnutie);

}

// Функция решета Эратосфена для нахождения всех простых чисел до n

public static List<Integer> primes(int n) {

boolean[] isPrime = new boolean[n + 1]; // Вектор для отметки простоты чисел

Arrays.fill(isPrime, true);

List<Integer> primes = new ArrayList<>(); // Вектор для хранения простых чисел

// Реализация решета Эратосфена

for (int p = 2; p \* p <= n; ++p) { // Для всех чисел от 2 до sqrt(n)

if (isPrime[p]) { // Если число p простое

for (int i = p \* p; i <= n; i += p) { // Удаление кратных чисел p

isPrime[i] = false;

}

}

}

// Сбор всех простых чисел в вектор

for (int p = 2; p <= n; ++p) {

if (isPrime[p]) {

primes.add(p);

}

}

return primes; // Возвращение вектора простых чисел

}

// Функция для построения числа и проведения теста

public static Pair<Integer, List<Integer>> builderTest(List<Integer> prime, int bit) {

int maxIndex = 0; // Максимальный индекс в векторе простых чисел

int maxPow = 1; // Максимальная степень двойки

// Нахождение максимального индекса, такого что prime[maxIndex] < 2^(bit - 1)

while (maxIndex < prime.size() && prime.get(maxIndex) < Math.pow(2, bit - 1)) {

maxIndex++;

}

// Нахождение максимальной степени двойки, такой что 2^maxPow < 2^(bit - 1)

while (Math.pow(2, maxPow) < Math.pow(2, bit - 1)) {

maxPow++;

}

int m = 1; // Начальное значение m

List<Integer> q = new ArrayList<>(); // Вектор для хранения множителей

// Построение числа m

Random random = new Random();

while (true) {

int num = rn(0, maxIndex); // Случайный индекс простого числа

int power = rn(1, maxPow); // Случайная степень

// Проверка на переполнение и добавление множителя

if (Math.pow(prime.get(num), power) != 0) {

if (m \* Math.pow(prime.get(num), power) < Integer.MAX\_VALUE) {

m \*= Math.pow(prime.get(num), power);

q.add(prime.get(num));

}

}

// Проверка на соответствие условиям

if (m > Math.pow(2, bit - 2)) {

if (m >= Math.pow(2, bit - 1)) {

m = 1;

q.clear();

} else {

break;

}

}

}

int n = 2 \* m + 1; // Инициализация числа n

return new Pair<>(n, q); // Возвращение числа и множителей

}

// Тест Миллера для проверки числа на простоту

public static int testMillera(int n, int t, List<Integer> q) {

List<Integer> a = new ArrayList<>(); // Вектор для хранения случайных чисел

// Генерация t случайных чисел

Random random = new Random();

while (a.size() != t) {

int aj = rn(2, n - 1);

if (!a.contains(aj)) {

a.add(aj);

}

}

// Первоначальная проверка простоты

for (int aj : a) {

if (powerMod(aj, n - 1, n) != 1) {

return 0;

}

}

// Дополнительная проверка простоты

boolean flag = true;

int i = 0;

for (int aj : a) {

if (q.get(i) != 0 && powerMod(aj, (n - 1) / q.get(i), n) != 1) {

flag = false;

if (i < q.size() - 1) {

i++;

}

}

}

// Проверка результата

if (flag) {

return 0;

}

return 1;

}

// Функция для возведения числа a в степень b по модулю n

public static int powerMod(int a, int b, int n) {

long result = 1; // Начальное значение результата

long base = a;

while (b > 0) { // Пока степень больше 0

if (b % 2 == 1) { // Если степень нечетная

result = (result \* base) % n; // Умножение результата на a по модулю n

}

base = (base \* base) % n; // Возведение a в квадрат по модулю n

b /= 2; // Деление степени на 2

}

return (int) result; // Возвращение результата

}

// Функция для генерации случайного числа в диапазоне [a, b]

public static int rn(int a, int b) {

Random random = new Random();

return random.nextInt(b - a + 1) + a;

}

// Функция для вывода результатов

public static void printResults(List<Integer> res, List<String> resVerTest, List<Integer> otvegnutie) {

System.out.println("Prime Numbers\tTest Results\tOccurrences"); // Заголовок таблицы

System.out.println("----------------------------------------------"); // Разделитель

// Вывод каждого результата

for (int i = 0; i < res.size(); i++) {

System.out.println(res.get(i) + "\t\t" + resVerTest.get(i) + "\t\t" + otvegnutie.get(i));

}

}

}

// Вспомогательный класс для работы с парами

class Pair<K, V> {

private final K key;

private final V value;

public Pair(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey() {

return key;

}

public V getValue() {

return value;

}

}

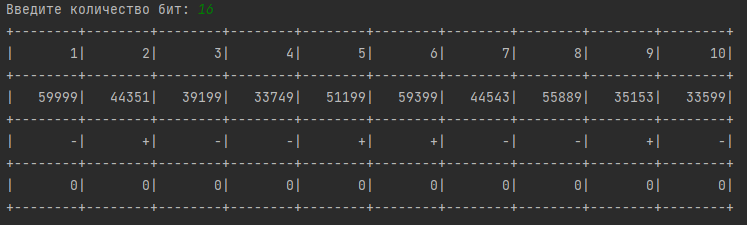


Рисунок 4: Результат работы программы на Java при требуемоей размерности простого числа 16.

**Выполнение задания 2 тест Поклингтона**

**Листинг программы на C++:**

#include <iostream>

#include <vector>

#include <tuple>

#include <cmath>

#include <random>

#include <limits>

#include <algorithm>

using namespace std;

// Функция для нахождения всех простых чисел до n (решето Эратосфена)

vector<int> primes(int n);

// Функция для построения числа и проведения теста

pair<int, vector<int>> builder\_test(vector<int> prime, int bit);

// Тест Поклингтона для проверки числа на простоту

int test\_poklin(int n, int t, vector<int> q);

// Функция для возведения числа a в степень b по модулю n

int power\_mod(int a, int b, int n);

// Функция для генерации случайного числа в диапазоне [a, b]

int rn(int a, int b);

// Функция для вывода результатов

void print\_results(const vector<int>& res, const vector<string>& res\_ver\_test, const vector<int>& otvegnutie);

int main() {

int size\_primes = 500; // Размер для генерации простых чисел

vector<int> prime = primes(size\_primes); // Генерация простых чисел

int bit = 0; // Размер числа в битах

cin >> bit; // Ввод размера числа

vector<int> q; // Вектор для хранения множителей

int n; // Сгенерированное число

int k = 0; // Счетчик отвергнутых чисел

int probability; // Вероятность простоты

vector<int> res; // Вектор для хранения результатов

vector<string> res\_ver\_test; // Вектор для хранения результатов теста

vector<int> otvegnutie; // Вектор для хранения количества отвергнутых чисел

// Генерация 10 различных простых чисел

while (res.size() != 10) {

// Построение числа и проведение теста

tie(n, q) = builder\_test(prime, bit);

probability = test\_poklin(n, 10, q);

// Проверка на простоту

if (probability == 1) {

// Проверка на уникальность числа

if (find(res.begin(), res.end(), n) == res.end()) {

res.push\_back(n); // Добавление числа в результаты

// Дополнительная проверка числа

probability = test\_poklin(n, 1, q);

if (probability == 1) {

res\_ver\_test.push\_back("+"); // Число прошло проверку

} else {

res\_ver\_test.push\_back("-"); // Число не прошло проверку

}

otvegnutie.push\_back(k); // Добавление количества отвергнутых чисел

k = 0; // Сброс счетчика отвергнутых чисел

}

} else {

k++; // Увеличение счетчика отвергнутых чисел

}

}

// Вывод результатов

print\_results(res, res\_ver\_test, otvegnutie);

}

// Функция решета Эратосфена для нахождения всех простых чисел до n

vector<int> primes(int n) {

vector<bool> is\_prime(n + 1, true); // Вектор для отметки простоты чисел

vector<int> primes; // Вектор для хранения простых чисел

// Реализация решета Эратосфена

for (int p = 2; p \* p <= n; ++p) { // Для всех чисел от 2 до sqrt(n)

if (is\_prime[p]) { // Если число p простое

for (int i = p \* p; i <= n; i += p) // Удаление кратных чисел p

is\_prime[i] = false;

}

}

// Сбор всех простых чисел в вектор

for (int p = 2; p <= n; ++p) {

if (is\_prime[p])

primes.push\_back(p);

}

return primes; // Возвращение вектора простых чисел

}

// Функция для построения числа и проведения теста

pair<int, vector<int>> builder\_test(vector<int> prime, int bit) {

int max\_index = 0; // Максимальный индекс в векторе простых чисел

int max\_pow = 1; // Максимальная степень двойки

// Нахождение максимального индекса, такого что prime[max\_index] < 2^((bit / 2) + 1)

for (; (prime[max\_index] < pow(2, (bit / 2) + 1)) && (max\_index < prime.size()); max\_index++);

// Нахождение максимальной степени двойки, такой что 2^max\_pow < 2^((bit / 2) + 1)

for (; pow(2, max\_pow) < pow(2, (bit / 2) + 1); max\_pow++);

int f = 1; // Начальное значение f

vector<int> q; // Вектор для хранения множителей

// Построение числа f

while (true) {

int num = rn(0, max\_index); // Случайный индекс простого числа

int power = rn(1, max\_pow); // Случайная степень

// Проверка на переполнение и добавление множителя

if (pow(prime[num], power)) {

if (f \* pow(prime[num], power) < INT32\_MAX) {

f \*= pow(prime[num], power);

q.push\_back(prime[num]);

}

}

// Проверка на соответствие условиям

if (f > pow(2, (bit / 2))) {

if (f >= pow(2, (bit / 2) + 1)) {

f = 1;

q.clear();

} else {

break;

}

}

}

int R; // Случайное четное число R

// Генерация случайного четного числа R

do {

R = rn(pow(2, (bit / 2) - 1) + 1, pow(2, bit / 2));

} while (R % 2 != 0);

int n = R \* f + 1; // Инициализация числа n

return make\_pair(n, q); // Возвращение числа и множителей

}

// Тест Поклингтона для проверки числа на простоту

int test\_poklin(int n, int t, vector<int> q) {

vector<int> a; // Вектор для хранения случайных чисел

int aj; // Случайное число

// Генерация t случайных чисел

while (a.size() != t) {

aj = rn(2, n - 1);

if (find(a.begin(), a.end(), aj) == a.end()) {

a.push\_back(aj);

}

}

// Первоначальная проверка простоты

for (int aj : a) {

if (power\_mod(aj, n - 1, n) != 1) {

return 0;

break;

}

}

// Дополнительная проверка простоты

bool flag = true;

int i = 0;

for (int aj : a) {

if (q[i] != 0 && power\_mod(aj, (n - 1) / q[i], n) == 1) {

flag = false;

return 0;

break;

}

}

// Проверка результата

if (flag) {

return 1;

}

return 1;

}

// Функция для возведения числа a в степень b по модулю n

int power\_mod(int a, int b, int n) {

long long result = 1; // Начальное значение результата

while (b > 0) { // Пока степень больше 0

if (b % 2 == 1) // Если степень нечетная

result = (result \* a) % n; // Умножение результата на a по модулю n

a = (a \* a) % n; // Возведение a в квадрат по модулю n

b /= 2; // Деление степени на 2

}

return result; // Возвращение результата

}

// Функция для генерации случайного числа в диапазоне [a, b]

int rn(int a, int b) {

mt19937 mt\_rand(random\_device{}()); // Инициализация генератора случайных чисел

return uniform\_int\_distribution<int>(a, b)(mt\_rand); // Возвращение случайного числа

}

// Функция для вывода результатов

void print\_results(const vector<int>& res, const vector<string>& res\_ver\_test, const vector<int>& otvegnutie) {

cout << "Prime Numbers\tTest Results\tOccurrences" << endl; // Заголовок таблицы

cout << "----------------------------------------------" << endl; // Разделитель

// Вывод каждого результата

for (size\_t i = 0; i < res.size(); ++i) {

cout << res[i] << "\t\t" << res\_ver\_test[i] << "\t\t" << otvegnutie[i] << endl;

}

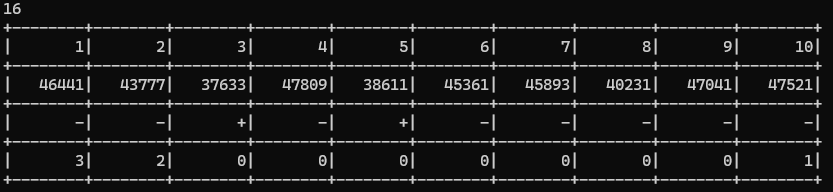
}

Рисунок 5: Результат работы программы на С++ при требуемоей размерности простого числа 16.

**Листинг программы на Java:**

import java.util.\*;

import java.util.concurrent.ThreadLocalRandom;

public class Main {

public static void main(String[] args) {

int sizePrimes = 500; // Размер для генерации простых чисел

List<Integer> prime = primes(sizePrimes); // Генерация простых чисел

Scanner scanner = new Scanner(System.in);

int bit = scanner.nextInt(); // Ввод размера числа

List<Integer> q; // Вектор для хранения множителей

int n; // Сгенерированное число

int k = 0; // Счетчик отвергнутых чисел

int probability; // Вероятность простоты

List<Integer> res = new ArrayList<>(); // Вектор для хранения результатов

List<String> resVerTest = new ArrayList<>(); // Вектор для хранения результатов теста

List<Integer> otvegnutie = new ArrayList<>(); // Вектор для хранения количества отвергнутых чисел

// Генерация 10 различных простых чисел

while (res.size() != 10) {

// Построение числа и проведение теста

Tuple<Integer, List<Integer>> result = builderTest(prime, bit);

n = result.first;

q = result.second;

probability = testPoklin(n, 10, q);

// Проверка на простоту

if (probability == 1) {

// Проверка на уникальность числа

if (!res.contains(n)) {

res.add(n); // Добавление числа в результаты

// Дополнительная проверка числа

probability = testPoklin(n, 1, q);

if (probability == 1) {

resVerTest.add("+"); // Число прошло проверку

} else {

resVerTest.add("-"); // Число не прошло проверку

}

otvegnutie.add(k); // Добавление количества отвергнутых чисел

k = 0; // Сброс счетчика отвергнутых чисел

}

} else {

k++; // Увеличение счетчика отвергнутых чисел

}

}

// Вывод результатов

printResults(res, resVerTest, otvegnutie);

}

// Функция для нахождения всех простых чисел до n (решето Эратосфена)

public static List<Integer> primes(int n) {

boolean[] isPrime = new boolean[n + 1]; // Вектор для отметки простоты чисел

Arrays.fill(isPrime, true);

List<Integer> primes = new ArrayList<>(); // Вектор для хранения простых чисел

// Реализация решета Эратосфена

for (int p = 2; p \* p <= n; p++) { // Для всех чисел от 2 до sqrt(n)

if (isPrime[p]) { // Если число p простое

for (int i = p \* p; i <= n; i += p) // Удаление кратных чисел p

isPrime[i] = false;

}

}

// Сбор всех простых чисел в вектор

for (int p = 2; p <= n; p++) {

if (isPrime[p])

primes.add(p);

}

return primes; // Возвращение вектора простых чисел

}

// Функция для построения числа и проведения теста

public static Tuple<Integer, List<Integer>> builderTest(List<Integer> prime, int bit) {

int maxIndex = 0; // Максимальный индекс в векторе простых чисел

int maxPow = 1; // Максимальная степень двойки

// Нахождение максимального индекса, такого что prime[maxIndex] < 2^((bit / 2) + 1)

while (prime.get(maxIndex) < Math.pow(2, (bit / 2) + 1) && maxIndex < prime.size()) {

maxIndex++;

}

// Нахождение максимальной степени двойки, такой что 2^maxPow < 2^((bit / 2) + 1)

while (Math.pow(2, maxPow) < Math.pow(2, (bit / 2) + 1)) {

maxPow++;

}

int f = 1; // Начальное значение f

List<Integer> q = new ArrayList<>(); // Вектор для хранения множителей

// Построение числа f

while (true) {

int num = rn(0, maxIndex); // Случайный индекс простого числа

int power = rn(1, maxPow); // Случайная степень

// Проверка на переполнение и добавление множителя

if (Math.pow(prime.get(num), power) != 0) {

if (f \* Math.pow(prime.get(num), power) < Integer.MAX\_VALUE) {

f \*= Math.pow(prime.get(num), power);

q.add(prime.get(num));

}

}

// Проверка на соответствие условиям

if (f > Math.pow(2, (bit / 2))) {

if (f >= Math.pow(2, (bit / 2) + 1)) {

f = 1;

q.clear();

} else {

break;

}

}

}

int R; // Случайное четное число R

// Генерация случайного четного числа R

do {

R = rn((int) Math.pow(2, (bit / 2) - 1) + 1, (int) Math.pow(2, bit / 2));

} while (R % 2 != 0);

int n = R \* f + 1; // Инициализация числа n

return new Tuple<>(n, q); // Возвращение числа и множителей

}

// Тест Поклингтона для проверки числа на простоту

public static int testPoklin(int n, int t, List<Integer> q) {

List<Integer> a = new ArrayList<>(); // Вектор для хранения случайных чисел

// Генерация t случайных чисел

while (a.size() != t) {

int aj = rn(2, n - 1);

if (!a.contains(aj)) {

a.add(aj);

}

}

// Первоначальная проверка простоты

for (int aj : a) {

if (powerMod(aj, n - 1, n) != 1) {

return 0;

}

}

// Дополнительная проверка простоты

for (int aj : a) {

for (int qi : q) {

if (qi != 0 && powerMod(aj, (n - 1) / qi, n) == 1) {

return 0;

}

}

}

return 1;

}

// Функция для возведения числа a в степень b по модулю n

public static int powerMod(int a, int b, int n) {

long result = 1; // Начальное значение результата

long base = a;

while (b > 0) { // Пока степень больше 0

if (b % 2 == 1) { // Если степень нечетная

result = (result \* base) % n; // Умножение результата на a по модулю n

}

base = (base \* base) % n; // Возведение a в квадрат по модулю n

b /= 2; // Деление степени на 2

}

return (int) result; // Возвращение результата

}

// Функция для генерации случайного числа в диапазоне [a, b]

public static int rn(int a, int b) {

return ThreadLocalRandom.current().nextInt(a, b + 1); // Возвращение случайного числа

}

// Функция для вывода результатов

public static void printResults(List<Integer> res, List<String> resVerTest, List<Integer> otvegnutie) {

System.out.println("Prime Numbers\tTest Results\tOccurrences"); // Заголовок таблицы

System.out.println("----------------------------------------------"); // Разделитель

// Вывод каждого результата

for (int i = 0; i < res.size(); i++) {

System.out.println(res.get(i) + "\t\t" + resVerTest.get(i) + "\t\t" + otvegnutie.get(i));

}

}

// Класс для хранения пары значений

static class Tuple<X, Y> {

public final X first;

public final Y second;

public Tuple(X first, Y second) {

this.first = first;

this.second = second;

}

}

}

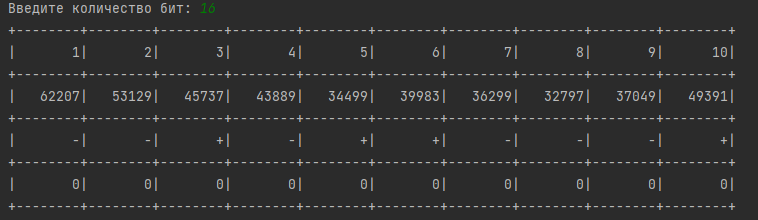


Рисунок 6: Результат работы программы на Java при требуемоей размерности простого числа 16.

**Выполнение задания 2 ГОСТ 31.10-94**

**Листинг программы на C++:**

#include <iostream>

#include <vector>

#include <cmath>

#include <random>

#include <algorithm>

using namespace std;

// Функция решета Эратосфена для нахождения всех простых чисел до n

vector<int> primes(int n);

// Функция для генерации нового простого числа из старого по ГОСТ

int build\_new\_from\_old(vector<int> prime, int bit);

// Функция для возведения числа a в степень b по модулю n

int power\_mod(int a, int b, int n);

// Функция для генерации случайного целого числа в диапазоне [a, b]

int rn\_int(int a, int b);

// Функция для генерации случайного дробного числа в диапазоне [a, b]

double rn\_double(int a, int b);

// Функция для печати результата

void print\_res(vector<int> res);

int main() {

// Генерация всех простых чисел до 500

vector<int> prime = primes(500);

// Ввод требуемого размера числа в битах

int bit;

cin >> bit;

// Вектор для хранения сгенерированных простых чисел

vector<int> res;

// Генерация 10 различных простых чисел заданной длины

while (res.size() != 10) {

// Генерация нового простого числа

int p = build\_new\_from\_old(prime, bit);

// Проверка на уникальность сгенерированного числа

if (find(res.begin(), res.end(), p) == res.end()) {

res.push\_back(p);

}

}

// Печать результата

print\_res(res);

}

vector<int> primes(int n) {

// Инициализация вектора, где is\_prime[i] показывает, является ли число i простым

vector<bool> is\_prime(n + 1, true);

// Вектор для хранения простых чисел

vector<int> primes;

// Реализация решета Эратосфена

for (int p = 2; p \* p <= n; ++p) { // Для всех чисел от 2 до sqrt(n)

if (is\_prime[p]) { // Если число p простое

for (int i = p \* p; i <= n; i += p) // Удаление кратных чисел p

is\_prime[i] = false;

}

}

// Сбор всех простых чисел в вектор

for (int p = 2; p <= n; ++p) {

if (is\_prime[p])

primes.push\_back(p);

}

return primes; // Возвращение вектора простых чисел

}

// Алгоритм перехода от меньшего простого числа к большему по ГОСТ

int build\_new\_from\_old(vector<int> prime, int bit) {

int q; // Переменная для хранения выбранного простого числа

int max\_index = 0; // Максимальный индекс в векторе простых чисел

int p; // Переменная для хранения нового простого числа

// Нахождение максимального индекса, такого что prime[max\_index] < 2^(bit / 2)

for (; prime[max\_index] < pow(2, bit / 2); max\_index++);

while (true) {

// Выбор случайного простого числа q

q = prime[rn\_int(0, max\_index)];

// Проверка, что q находится в заданном диапазоне

if (q > pow(2, (bit / 2) - 1) && q <= pow(2, bit / 2) - 1) break;

}

while (true) {

// Генерация случайного числа от 0 до 1

double zakaruchka = rn\_double(0, 1);

// Вычисление нового числа n на основе q и zakaruchka

double n = (pow(2, bit - 1) / q) + (pow(2, bit - 1) \* zakaruchka / q);

// Если n нечетное, сделать его четным

if ((int)n % 2 == 1) n++;

// Перебор всех u, начиная с 0 с шагом 2

for (int u = 0; true; u += 2) {

// Вычисление кандидата в простые числа

p = (n + u) \* q + 1;

// Если p превышает 2^bit, выйти из цикла

if (p > pow(2, bit)) break;

// Проверка условия теоремы Диемитко

if (power\_mod(2, p - 1, p) == 1 && power\_mod(2, n + u, p) != 1) return p;

}

}

}

// Функция для возведения числа a в степень b по модулю n

int power\_mod(int a, int b, int n) {

int result = 1; // Результат возведения в степень по модулю

while (b > 0) { // Пока степень b больше 0

if (b % 2 == 1) // Если b нечетное

result = (result \* a) % n; // Умножить результат на a по модулю n

a = (a \* a) % n; // Возвести a в квадрат по модулю n

b /= 2; // Разделить b на 2

}

return result; // Возвращение результата

}

// Функция для генерации случайного целого числа в диапазоне [a, b]

int rn\_int(int a, int b) {

// Инициализация генератора случайных чисел

static mt19937 mt\_rand(random\_device{}());

return uniform\_int\_distribution<int>(a, b)(mt\_rand); // Возвращение случайного числа

}

// Функция для генерации случайного дробного числа в диапазоне [a, b]

double rn\_double(int a, int b) {

// Инициализация генератора случайных чисел

static mt19937 mt\_rand(random\_device{}());

return uniform\_real\_distribution<double>(a, b)(mt\_rand); // Возвращение случайного числа

}

// Функция для печати результата

void print\_res(vector<int> res) {

// Печать каждого элемента вектора

for (int i = 0; i < res.size(); i++) {

cout << i + 1 << "\t\t" << "|" << "\t\t" << res[i] << endl;

}

}

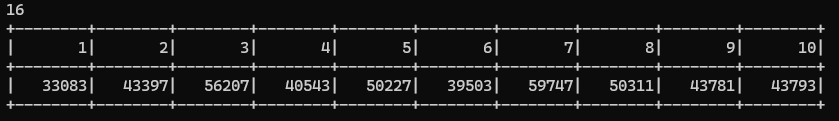


Рисунок 7: Результат работы программы на С++ при требуемоей размерности простого числа 16.

**Листинг программы на Python:**

import java.util.\*;

import java.util.stream.IntStream;

public class Main {

public static void main(String[] args) {

// Генерация всех простых чисел до 500

List<Integer> prime = primes(500);

// Ввод требуемого размера числа в битах

Scanner scanner = new Scanner(System.in);

int bit = scanner.nextInt();

// Вектор для хранения сгенерированных простых чисел

List<Integer> res = new ArrayList<>();

// Генерация 10 различных простых чисел заданной длины

while (res.size() != 10) {

// Генерация нового простого числа

int p = buildNewFromOld(prime, bit);

// Проверка на уникальность сгенерированного числа

if (!res.contains(p)) {

res.add(p);

}

}

// Печать результата

printRes(res);

}

// Функция решета Эратосфена для нахождения всех простых чисел до n

public static List<Integer> primes(int n) {

boolean[] isPrime = new boolean[n + 1];

Arrays.fill(isPrime, true);

List<Integer> primes = new ArrayList<>();

for (int p = 2; p \* p <= n; ++p) {

if (isPrime[p]) {

for (int i = p \* p; i <= n; i += p) {

isPrime[i] = false;

}

}

}

for (int p = 2; p <= n; ++p) {

if (isPrime[p]) {

primes.add(p);

}

}

return primes;

}

// Алгоритм перехода от меньшего простого числа к большему по ГОСТ

public static int buildNewFromOld(List<Integer> prime, int bit) {

int q;

int maxIndex = 0;

int p;

// Нахождение максимального индекса, такого что prime[maxIndex] < 2^(bit / 2)

while (prime.get(maxIndex) < Math.pow(2, bit / 2)) {

maxIndex++;

}

Random random = new Random();

while (true) {

// Выбор случайного простого числа q

q = prime.get(rnInt(0, maxIndex));

if (q > Math.pow(2, (bit / 2) - 1) && q <= Math.pow(2, bit / 2) - 1) break;

}

while (true) {

// Генерация случайного числа от 0 до 1

double zakaruchka = rnDouble(0, 1);

double n = (Math.pow(2, bit - 1) / q) + (Math.pow(2, bit - 1) \* zakaruchka / q);

// Если n нечетное, сделать его четным

if ((int) n % 2 == 1) n++;

// Перебор всех u, начиная с 0 с шагом 2

for (int u = 0; true; u += 2) {

// Вычисление кандидата в простые числа

p = (int) ((n + u) \* q + 1);

if (p > Math.pow(2, bit)) break;

// Проверка условия теоремы Диемитко

if (powerMod(2, p - 1, p) == 1 && powerMod(2, (int) n + u, p) != 1) return p;

}

}

}

// Функция для возведения числа a в степень b по модулю n

public static int powerMod(int a, int b, int n) {

int result = 1;

while (b > 0) {

if (b % 2 == 1) result = (result \* a) % n;

a = (a \* a) % n;

b /= 2;

}

return result;

}

// Функция для генерации случайного целого числа в диапазоне [a, b]

public static int rnInt(int a, int b) {

Random random = new Random();

return random.nextInt(b - a + 1) + a;

}

// Функция для генерации случайного дробного числа в диапазоне [a, b]

public static double rnDouble(int a, int b) {

Random random = new Random();

return a + (b - a) \* random.nextDouble();

}

// Функция для печати результата

public static void printRes(List<Integer> res) {

for (int i = 0; i < res.size(); i++) {

System.out.println((i + 1) + "\t\t|\t\t" + res.get(i));

}

}

}

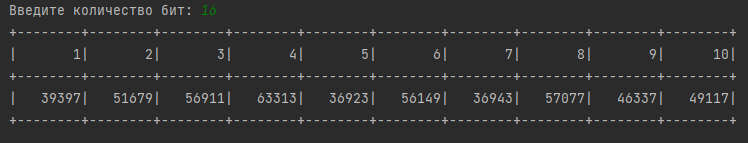


Рисунок 8: Результат работы программы на Java при требуемоей размерности простого числа 16.

**Выполнение задания 3**

**Листинг программы на C++:**

#include <iostream>

#include <vector>

#include <numeric>

#include <cmath>

#include <stdexcept>

using namespace std;

// Структура для хранения коэффициентов линейной аппроксимации

struct ApproximationResult {

double a; // Коэффициент a

double b; // Коэффициент b

};

// Структура для хранения результатов корреляции

struct CorrelationResult {

double correlation; // Коэффициент корреляции

double t\_value; // Значение t-критерия

};

// Функция для линейной аппроксимации данных

ApproximationResult aproks(const vector<double>& x, const vector<double>& y);

// Функция для вычисления коэффициентов корреляции

CorrelationResult korrel(const vector<double>& x, const vector<double>& y);

// Функция моделирования процесса остывания кофе

vector<double> coffee(double CoffeeTemperature, double RoomTemperature, double CoolingFactor, int Time);

// Функция для печати данных о температуре кофе в конкретный момент времени

void printCoffee(vector<double> temperatures, vector<double>times);

// Функция для печати результатов линейной аппроксимации

void printApproximationResult(ApproximationResult result);

// Функция для печати результатов корреляции

void printCorrelationResult(CorrelationResult result);

int main() {

// Параметры моделирования процесса остывания кофе

double CoffeeTemperature = 90; // Начальная температура кофе

double RoomTemperature = 26; // Температура комнаты

double CoolingFactor = 0.01; // Коэффициент остывания

int Time = 60; // Временной предел в минутах

// Моделирование процесса остывания кофе

vector<double> temperatures = coffee(CoffeeTemperature, RoomTemperature, CoolingFactor, Time);

// Создание временных меток для измерений

vector<double> times(Time + 1);

iota(times.begin(), times.end(), 0);

// Вычисление линейной аппроксимации и корреляции

ApproximationResult approx\_result = aproks(times, temperatures);

CorrelationResult corr\_result = korrel(times, temperatures);

// Печать результатов

printApproximationResult(approx\_result);

printCorrelationResult(corr\_result);

printCoffee(temperatures, times);

return 0;

}

// Функция для линейной аппроксимации данных

ApproximationResult aproks(const vector<double>& x, const vector<double>& y) {

// Суммирование значений x и y

double x\_sum = accumulate(x.begin(), x.end(), 0.0);

double y\_sum = accumulate(y.begin(), y.end(), 0.0);

// Суммирование квадратов значений x и произведений значений x и y

double x2\_sum = inner\_product(x.begin(), x.end(), x.begin(), 0.0);

double xy\_sum = inner\_product(x.begin(), x.end(), y.begin(), 0.0);

// Количество измерений

size\_t n = x.size();

// Вычисление коэффициентов линейной аппроксимации

double a = (n \* xy\_sum - x\_sum \* y\_sum) / (n \* x2\_sum - x\_sum \* x\_sum);

double b = (y\_sum - a \* x\_sum) / n;

return { a, b };

}

// Функция для вычисления среднего значения

double mean(const vector<double>& vec) {

return accumulate(vec.begin(), vec.end(), 0.0) / vec.size();

}

// Функция для вычисления коэффициентов корреляции

CorrelationResult korrel(const vector<double>& x, const vector<double>& y) {

// Вычисление средних значений

double x\_mean = mean(x);

double y\_mean = mean(y);

// Инициализация переменных для вычисления сумм

double sum\_xy = 0.0;

double sum\_x2 = 0.0;

double sum\_y2 = 0.0;

size\_t n = x.size();

// Вычисление сумм для формулы коэффициента корреляции

for (size\_t i = 0; i < n; ++i) {

sum\_xy += (x[i] - x\_mean) \* (y[i] - y\_mean);

sum\_x2 += (x[i] - x\_mean) \* (x[i] - x\_mean);

sum\_y2 += (y[i] - y\_mean) \* (y[i] - y\_mean);

}

// Вычисление коэффициента корреляции и t-критерия

double r = sum\_xy / (sqrt(sum\_x2) \* sqrt(sum\_y2));

double t = r \* sqrt(n - 2) / sqrt(1 - r \* r);

return { r, t };

}

// Функция моделирования процесса остывания кофе

vector<double> coffee(double CoffeeTemperature, double RoomTemperature, double CoolingFactor, int Time) {

// Вектор для хранения температур кофе в разные моменты времени

vector<double> temperatures;

// Моделирование процесса остывания

for (int t = 0; t <= Time; t++) {

double temperature = RoomTemperature + (CoffeeTemperature - RoomTemperature) \* exp(-CoolingFactor \* t);

temperatures.push\_back(temperature);

}

return temperatures;

}

// Функция для печати результатов линейной аппроксимации

void printApproximationResult(ApproximationResult result) {

cout << "Approximation Result:" << endl;

cout << "a: " << result.a << ", b = " << result.b << endl;

}

// Функция для печати результатов корреляции

void printCorrelationResult(CorrelationResult result) {

cout << "Correlation Result:" << endl;

cout << "Correlation: " << result.correlation << endl;

cout << "T-value: " << result.t\_value << endl;

}

// Функция для печати данных о температуре кофе в конкретный момент времени

void printCoffee(vector<double> temperatures, vector<double>times) {

for (int i = 0; i < temperatures.size(); i++) {

cout << "Time - " << times[i] << ": " << temperatures[i] << " C" << endl;

}

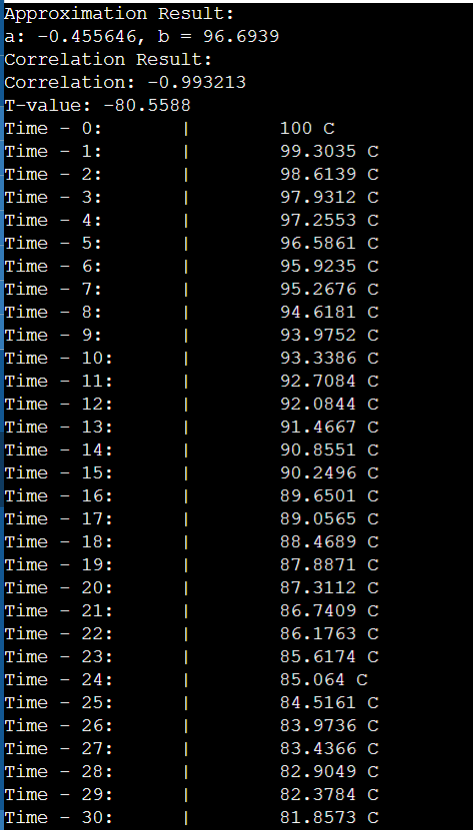
}

Рисунок 9: Результат работы программы на С++ при T = 100, Ts = 30, r = 0.01, time\_limit = 30.

**Листинг программы на Java:**

import java.util.ArrayList;

import java.util.List;

public class Main {

// Структура для хранения коэффициентов линейной аппроксимации

static class ApproximationResult {

double a; // Коэффициент a

double b; // Коэффициент b

public ApproximationResult(double a, double b) {

this.a = a;

this.b = b;

}

}

// Структура для хранения результатов корреляции

static class CorrelationResult {

double correlation; // Коэффициент корреляции

double t\_value; // Значение t-критерия

public CorrelationResult(double correlation, double t\_value) {

this.correlation = correlation;

this.t\_value = t\_value;

}

}

// Функция для линейной аппроксимации данных

static ApproximationResult aproks(List<Double> x, List<Double> y) {

// Суммирование значений x и y

double x\_sum = x.stream().mapToDouble(Double::doubleValue).sum();

double y\_sum = y.stream().mapToDouble(Double::doubleValue).sum();

// Суммирование квадратов значений x и произведений значений x и y

double x2\_sum = x.stream().mapToDouble(a -> a \* a).sum();

double xy\_sum = 0.0;

for (int i = 0; i < x.size(); i++) {

xy\_sum += x.get(i) \* y.get(i);

}

// Количество измерений

int n = x.size();

// Вычисление коэффициентов линейной аппроксимации

double a = (n \* xy\_sum - x\_sum \* y\_sum) / (n \* x2\_sum - x\_sum \* x\_sum);

double b = (y\_sum - a \* x\_sum) / n;

return new ApproximationResult(a, b);

}

// Функция для вычисления среднего значения

static double mean(List<Double> vec) {

return vec.stream().mapToDouble(Double::doubleValue).average().orElse(Double.NaN);

}

// Функция для вычисления коэффициентов корреляции

static CorrelationResult korrel(List<Double> x, List<Double> y) {

// Вычисление средних значений

double x\_mean = mean(x);

double y\_mean = mean(y);

// Инициализация переменных для вычисления сумм

double sum\_xy = 0.0;

double sum\_x2 = 0.0;

double sum\_y2 = 0.0;

int n = x.size();

// Вычисление сумм для формулы коэффициента корреляции

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

sum\_xy += (x.get(i) - x\_mean) \* (y.get(i) - y\_mean);

sum\_x2 += (x.get(i) - x\_mean) \* (x.get(i) - x\_mean);

sum\_y2 += (y.get(i) - y\_mean) \* (y.get(i) - y\_mean);

}

// Вычисление коэффициента корреляции и t-критерия

double r = sum\_xy / (Math.sqrt(sum\_x2) \* Math.sqrt(sum\_y2));

double t = r \* Math.sqrt(n - 2) / Math.sqrt(1 - r \* r);

return new CorrelationResult(r, t);

}

// Функция моделирования процесса остывания кофе

static List<Double> coffee(double CoffeeTemperature, double RoomTemperature, double CoolingFactor, int Time) {

// Вектор для хранения температур кофе в разные моменты времени

List<Double> temperatures = new ArrayList<>();

// Моделирование процесса остывания

for (int t = 0; t <= Time; t++) {

double temperature = RoomTemperature + (CoffeeTemperature - RoomTemperature) \* Math.exp(-CoolingFactor \* t);

temperatures.add(temperature);

}

return temperatures;

}

// Функция для печати результатов линейной аппроксимации

static void printApproximationResult(ApproximationResult result) {

System.out.println("Approximation Result:");

System.out.println("a: " + result.a + ", b = " + result.b);

}

// Функция для печати результатов корреляции

static void printCorrelationResult(CorrelationResult result) {

System.out.println("Correlation Result:");

System.out.println("Correlation: " + result.correlation);

System.out.println("T-value: " + result.t\_value);

}

// Функция для печати данных о температуре кофе в конкретный момент времени

static void printCoffee(List<Double> temperatures, List<Double> times) {

for (int i = 0; i < temperatures.size(); i++) {

System.out.println("Time - " + times.get(i) + ": " + temperatures.get(i) + " C");

}

}

public static void main(String[] args) {

// Параметры моделирования процесса остывания кофе

double CoffeeTemperature = 90; // Начальная температура кофе

double RoomTemperature = 26; // Температура комнаты

double CoolingFactor = 0.01; // Коэффициент остывания

int Time = 60; // Временной предел в минутах

// Моделирование процесса остывания кофе

List<Double> temperatures = coffee(CoffeeTemperature, RoomTemperature, CoolingFactor, Time);

// Создание временных меток для измерений

List<Double> times = new ArrayList<>();

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

times.add((double) i);

}

// Вычисление линейной аппроксимации и корреляции

ApproximationResult approx\_result = aproks(times, temperatures);

CorrelationResult corr\_result = korrel(times, temperatures);

// Печать результатов

printApproximationResult(approx\_result);

printCorrelationResult(corr\_result);

printCoffee(temperatures, times);

}

}

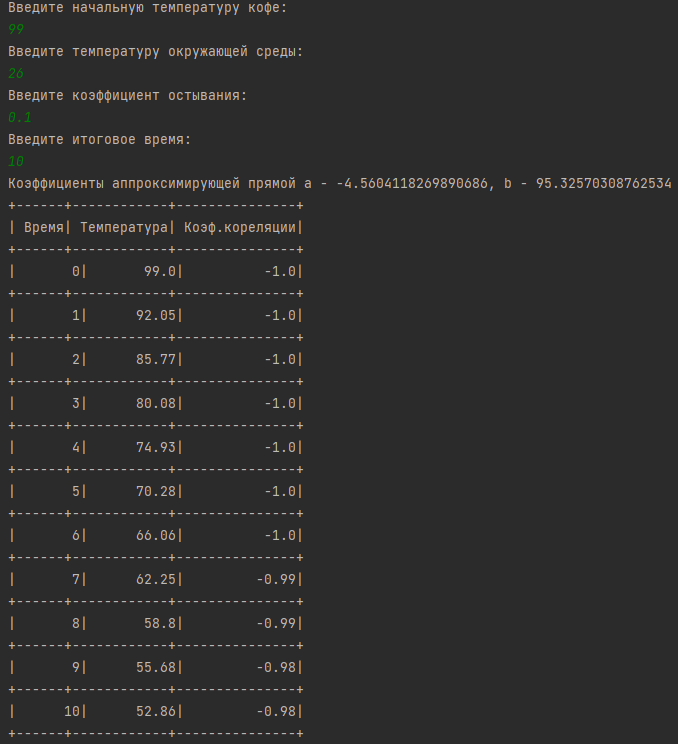


Рисунок 10: Результат работы программы на Java при T = 99, Ts = 26, r = 0.1, time\_limit=10.

**Гит репозиторий**

[https://github.com/RandredNSTU/LaboratoryWork3.git](https://github.com/DerzhiZhabu/laba3)

**Вывод**

В результате выполнения лабораторной работы были успешно изучены и применены циклические алгоритмы, разработаны программы для решения задач по заданным критериям, что позволило углубить понимание циклических вычислительных процессов и их применение в реальных задачах.