**Ковкель Никита, 2 ФИТ 4/2**

**Лабораторная работа №1. Вспомогательные функции**

**Цель работы:** приобретение навыков составления и отладки программ с использованием пользовательских функций для замера продолжительности процесса вычисления.

**1. Написание кода программы**

Исходный код файла Main.cpp приведен в листинге 1.1

|  |
| --- |
| #include "Auxil.h" // вспомогательные функции  #include <iostream>  #include <ctime>  #include <locale>  #include <chrono>  #define CYCLE 1000000 // количество циклов  int Fib(int i)  {  int value = 0;  if (i < 1) return 0;  if (i == 1) return 1;  return Fib(i - 1) + Fib(i - 2);  }  int task2()  {  double av1 = 0, av2 = 0;  auxil::start(); // старт генерации  auto start = std::chrono::steady\_clock::now();  Fib(50);  auto end = std::chrono::steady\_clock::now(); // фиксация времени  std::cout << "\nпродолжительность (н.с): " << std::chrono::duration\_cast<std::chrono::nanoseconds> (end - start).count();  std::cout << std::endl;  system("pause");  return 0;  }  int main()  {  setlocale(LC\_ALL, "rus");  task2();  system("pause");  double av1 = 0, av2 = 0;  //clock\_t t1 = 0, t2 = 0;  auxil::start(); // старт генерации  auto start = std::chrono::steady\_clock::now();  for (long long i = 0; i < 1000000; i++)  {  av1 += (int)auxil::iget(-100, 100);  av2 += auxil::dget(-100, 100);  }  auto end = std::chrono::steady\_clock::now(); // фиксация времени  std::cout << "количество циклов: " << CYCLE;  std::cout << "\nсреднее значение (int): " << av1 / CYCLE;  std::cout << "\nсреднее значение (double): " << av2 / CYCLE;  std::cout << "\nпродолжительность (н.с): " << std::chrono::duration\_cast<std::chrono::nanoseconds> (end - start).count();  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 1.1 Исходный код файла lab\_1.cpp

Исходный код файла Auxil.h приведен в листинге 1.2

|  |
| --- |
| #pragma once  #include <cstdlib>  namespace auxil  {  void start(); // старт генератора сл. чисел  double dget(double rmin, double rmax); // получить случайное число  int iget(int rmin, int rmax); // получить случайное число  }; |

Листинг 1.2 Исходный код файла Auxil.h

Исходный код файла Auxil.cpp приведен в листинге 1.3

|  |
| --- |
| #include "Auxil.h"  #include <ctime>  namespace auxil  {  void start() // старт генератора сл. чисел  {  srand((unsigned)time(NULL));  };  double dget(double rmin, double rmax) // получить случайное число  {  return ((double)rand() / (double)RAND\_MAX) \* (rmax - rmin) + rmin;  };  int iget(int rmin, int rmax) // получить случайное число  {  return (int)dget((double)rmin, (double)rmax);  };  } |

Листинг 1.3 Исходный код файла Auxil.cpp

График зависимости продолжительности процесса вычисления от количества циклов приведен в листинге 1.4.



Листинг 1.4 График зависимости продолжительности процесса вычисления от количества циклов

Найдем коэффициент корреляции для данной зависимости:



Рисунок 1.5 Коэффициент корреляции для графика 1.4

Так как коэффициент приблизительно равен единице, можем сделать вывод о том, что зависимость продолжительности процесса вычисления от количества цикла имеет линейный вид.

**1.2. Исследование рекурсивного алгоритма на примере чисел Фибоначчи.**

Исходный код программы приведен в листинге 1.6

|  |
| --- |
| int Fib(int i)  {  int value = 0;  if (i < 1) return 0;  if (i == 1) return 1;  return Fib(i - 1) + Fib(i - 2);  }  int task2()  {  double av1 = 0, av2 = 0;  auxil::start(); // старт генерации  auto start = std::chrono::steady\_clock::now();  Fib(50);  auto end = std::chrono::steady\_clock::now(); // фиксация времени  std::cout << "\nпродолжительность (н.с): " << std::chrono::duration\_cast<std::chrono::nanoseconds> (end - start).count();  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 1.6 Исходный код файла main.cpp

График зависимости продолжительности процесса вычисления от порядкового номера числа Фибоначчи приведен в листинге 1.7



Листинг 1.7 Зависимость продолжительности процесса вычисления от порядкового номера числа Фибоначчи (для N от 1 до 40)

На основании полученных результатов можно сделать вывод о том, что график зависимости стремится к показательной. Проведем аппроксимацию функции для доказательства вида зависимости:

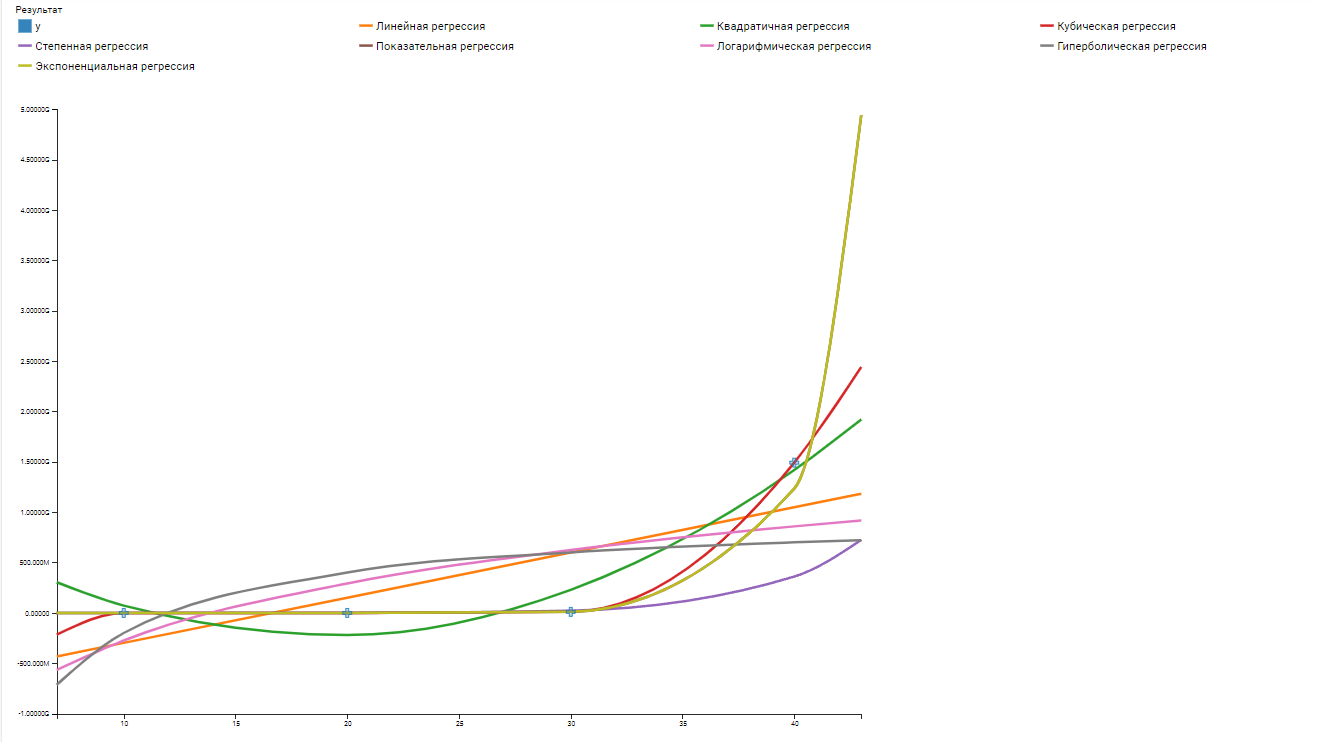


Рисунок 1.9 Аппроксимация функции

Вывод: скорость выполнения программы линейно зависит от количества итераций цикла. Скорость нахождения N-го числа Фибоначчи имеет вид кубической зависимости.