Министерство ПРОСВЕЩЕНИЯ Российской Федерации

Частное проффесиональное образовательное учреждение

«Московский Городской Открытый Колледж»

СПЕЦИАЛЬНОСТЬ 09.02.03 Программирование в компьютерных системах

**пояснительная записка**

к курсовому проекту на тему:

РАЗРАБОТКА ПРОГРАММЫ РЕШЕНИЯ

ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Группа

Листов 67

|  |  |  |
| --- | --- | --- |
| Студент |  |  |
|  | (подпись) |  |
| Руководитель курсового проекта |  | Т.М. Босенко |
|  | (подпись) |  |

Москва 2019

Министерство ПРОСВЕЩЕНИЯ Российской Федерации

Частное проффесиональное образовательное учреждение

«Московский Городской Открытый Колледж»

УТВЕРЖДАЮ

Председатель ПЦК специальности 09.02.03

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

задание

на выполнение курсового проекта

По дисциплине МДК.01.02 Прикладное программирование

Студент

(фамилия, инициалы, индекс группы)

Руководитель Т.М. Босенко

(фамилия, инициалы)

1. ***Тема курсового проекта***

Разработка программы решения дифференциальных уравнений

1. ***Техническое задание***

Написать программу решения дифференциальных уравнений методами Эйлера и усовершенствованным методом Эйлера

1. ***Оформление курсового проекта***

3.1. Пояснительная записка на \_\_\_\_\_\_\_ листах формата А4.

3.2. Перечень графического материала КП (плакаты, схемы, чертежи и т.п.) – схемы алгоритма программы

График выполнения работы: 20% к 3 нед., 40% к 6 нед., 60% к 9 нед., 80% к 12 нед.,

100% к 15 нед.

Дата выдачи задания « 16 » января 2019 г.

Руководитель курсового проекта Т.М. Босенко

Содержание

[ВВЕДЕНИЕ 5](#_Toc514012928)

[1 Постановка задачи 7](#_Toc514012929)

[1.1 Метод Эйлера 7](#_Toc514012930)

[1.2 Усовершенствованный метод Эйлера 7](#_Toc514012931)

[1.3 Системные требования, предъявляемые к программе 8](#_Toc514012932)

[2 Структура программы 10](#_Toc514012933)

[3 Схемы алгоритма программы 13](#_Toc514012934)

[3.1 Схема алгоритма основной программы 13](#_Toc514012935)

[3.2 Схема алгоритма процедуры calculateFunc 14](#_Toc514012936)

[3.3 Схема алгоритма процедуры safeinput 15](#_Toc514012937)

[3.4 Схема алгоритма процедуры outputlist() 16](#_Toc514012938)

[3.5 Схема алгоритма конструктора list 17](#_Toc514012939)

[4 Отладка программы 18](#_Toc514012940)

[4.1 Синтаксические ошибки, обнаруженные в программе 18](#_Toc514012941)

[4.2 Ошибки компоновки, обнаруженные в программе 18](#_Toc514012942)

[4.3 Ошибки выполнения, обнаруженные в программе 19](#_Toc514012943)

[5 Оптимизация программы 25](#_Toc514012944)

[6 Тестирование программы 28](#_Toc514012945)

[6.1 Тестирование в нормальных условиях 28](#_Toc514012946)

[6.2 Тестирование в экстремальных условиях 34](#_Toc514012947)

[6.3 Тестирование в исключительных ситуациях 39](#_Toc514012948)

[7 Руководство пользователя 43](#_Toc514012949)

[Заключение 47](#_Toc514012950)

[Список использованных источников 48](#_Toc514012951)

[Приложение А 49](#_Toc514012952)

[А.1 Листинг основной программы 50](#_Toc514012953)

[А.2 Листинг модуля list 58](#_Toc514012954)

[Приложение Б Результаты выполнения программы 63](#_Toc514012955)

# ВВЕДЕНИЕ

Дифференциальное уравнение – уравнение, в которое входят производные функции, и может входить сама функция, независимая переменная и параметры. Порядок входящих в уравнение производных может быть различен (формально он ничем не ограничен). Производные, функции, независимые переменные и параметры могут входить в уравнение в различных комбинациях или могут отсутствовать вовсе, кроме хотя бы одной производной. Не любое уравнение, содержащее производные неизвестной функции, является дифференциальным уравнением. Например, не является дифференциальным уравнением.

Дифференциальное уравнение порядка выше первого можно преобразовать в систему уравнений первого порядка, в которой число уравнений равно порядку исходного дифференциального уравнения.

В отличие от алгебраических уравнений, в результате решения которых ищется число (несколько чисел), при решении дифференциальных уравнений ищется функция (семейство функций).

Современные быстродействующие ЭВМ эффективно дают численное решение обыкновенных дифференциальных уравнений, не требуя получения его решения в аналитическом виде. Это позволило некоторым исследователям утверждать, что решение задачи получено, если её удалось свести к решению [обыкновенного дифференциального уравнения](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8B%D0%BA%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5).

Основой теории дифференциальных уравнений стало дифференциальное исчисление, созданное [Лейбницем](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%B9%D0%B1%D0%BD%D0%B8%D1%86,_%D0%93%D0%BE%D1%82%D1%84%D1%80%D0%B8%D0%B4_%D0%92%D0%B8%D0%BB%D1%8C%D0%B3%D0%B5%D0%BB%D1%8C%D0%BC) и [Ньютоном](https://ru.wikipedia.org/wiki/%D0%9D%D1%8C%D1%8E%D1%82%D0%BE%D0%BD,_%D0%98%D1%81%D0%B0%D0%B0%D0%BA) (1642—1727). Сам термин «дифференциальное уравнение» был предложен в 1676 году Лейбницем.

Из огромного числа работ [XVIII века](https://ru.wikipedia.org/wiki/XVIII_%D0%B2%D0%B5%D0%BA) по дифференциальным уравнениям выделяются работы [Эйлера](https://ru.wikipedia.org/wiki/%D0%AD%D0%B9%D0%BB%D0%B5%D1%80,_%D0%9B%D0%B5%D0%BE%D0%BD%D0%B0%D1%80%D0%B4) (1707—1783) и [Лагранжа](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D0%B3%D1%80%D0%B0%D0%BD%D0%B6,_%D0%96%D0%BE%D0%B7%D0%B5%D1%84_%D0%9B%D1%83%D0%B8) (1736—1813).

Если не удается никакими способами разделить переменные дифференциального уравнения  для которого существует частное решение, а при интегрировании получается не берущийся интеграл, то можно воспользоваться методами приближенных вычислений, которые позволяют зачастую с высочайшей точностью “сымитировать” функцию    на некотором промежутке. Идея методов Эйлера состоит в том, чтобы заменить фрагмент графика  ломаной линией.

Программа, данная в качестве задания курсового проекта, должна решать дифференциальные уравнения методами Эйлера и усовершенствованным методом Эйлера.

# Постановка задачи

Целью разработки курсового проекта является написание программы по теме: «Решение дифференциальных уравнений». Решение дифференциальных уравнений осуществляется следующими методами:

– методом Эйлера;

– усовершенствованным методом Эйлера.

## 1.1 Метод Эйлера

В качестве начальных условий задается обыкновенное дифференциальное уравнение и соответствующая этому уравнению задача Коши , где функция f определена на некоторой области . Решение ищется на интервале . На этом интервале вводятся узлы:

Приближенное решение в узлах , которое обозначим через , определяется по формуле:

Эти формулы непосредственно обобщаются на случай систем обыкновенных дифференциальных уравнений.

## 1.2 Усовершенствованный метод Эйлера

Усовершенствованный метод Эйлера позволяет повысить точность и устойчивость вычисления.

Очередное значение  будет точнее, если значение производной, то есть угловой коэффициент прямой, замещающей интегральную кривую на отрезке , будет вычисляться не по левому краю (то есть в точке ), а по центру отрезка . Но так как значение производной между точками не вычисляется, то нужно перейти к сдвоенным участкам с центром, которым является точка . При этом уравнение прямой получает вид:

## 1.3 Системные требования, предъявляемые к программе

Программа должна быть разработана для компьютеров со следующей конфигурацией:

– процессор Intel Core i7, AMD Phenom II;

– количество ОЗУ от 2гб;

– место на жёстком диске от 20 мб;

– любая встроенная или дискретная видеокарта

Программа должна быть написана на C++.

C++ – чрезвычайно мощный язык, содержащий средства создания эффективных программ практически любого назначения, от низкоуровневых утилит и драйверов до сложных программных комплексов самого различного назначения.

Преимущества языка C++:

– Высокая совместимость с языком С, позволяющая использовать весь существующий С – код (код С может быть с минимальными переделками скомпилирован компилятором С++; библиотеки, написанные на С, обычно могут быть вызваны из С++ непосредственно без каких-либо дополнительных затрат, в том числе и на уровне функций обратного вызова, позволяя библиотекам, написанным на С, вызывать код, написанный на С++).

– Поддерживаются различные стили и технологии программирования, включая традиционное директивное программирование, ООП, обобщенное программирование (шаблоны, макросы).

–Имеется возможность работы на низком уровне с памятью, адресами, портами.

–Возможность создания обобщённых контейнеров и алгоритмов для разных типов данных, их специализация и вычисления на этапе компиляции, используя шаблоны.

–Кроссплатформенность. Доступны компиляторы для большого количества платформ, на языке C++ разрабатывают программы для самых различных платформ и систем.

– Эффективность. Язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы.

Ввиду выше перечисленных фактов о языке C++, играющих важную роль при разработке любого ПО, а также благодаря множеству надежных и проверенных временем библиотек, которые имеются в наличие у данного языка, для решения поставленной задачи был выбран С++.

* + 1. Требования к программе
       1. Требования к функциональным характеристикам

К программе предъявляются определённые требования, которые включают в себя наличие следующих функций:

1) просмотра и сортировки различных данных, интегрированных в программу, в том числе организованных в виде таблиц;

2) модуляции конструктора снаряжения похожая на игровой аналог;

3) просмотра интегрированных в программу обучающих видео.

* + - 1. Требования к надёжности

Программа должна обладать высокой степенью отказоустойчивости. В программной системе необходимо предусмотреть защиту данных от случайного удаления и изменения. С помощью программных средств, требуется установить режим просмотра данных «Только для чтения».

* + - 1. Входные данные

В таблице 1.1 описаны переменные, задействованные в программе.

Таблица 1.1 – Описание переменных

|  |  |
| --- | --- |
| Переменная | Назначение |
| F | Временная переменная для обработки данных |
| Name | Массив названий для заполнения таблиц |
| Uron | Массив значений для заполнения таблиц |
| Prob | Массив значений для заполнения таблиц |
| Sell | Массив значений для заполнения таблиц |
| Image | Переменная для обработки изображений |
| продолжение Таблицы 1.1 | |
| Money | Общий бюджет пользователя (Конструктор) |
| Vichet | Сумма трат пользователя ( Конструктор) |

* + - 1. Требования к составу и параметрам технических средств

Минимальная аппаратная конфигурация системы, обеспечивающей нормальное функционирование обучающей системы должна быть не ниже следующей:

1) процессор Intel Core2Duo 900 МГц и выше;

2) оперативная память 1 Гб и выше;

3) свободного места на жестком диске не менее 350 Мб.

* + - 1. Требования к информационной и программной совместимости

Для эксплуатации программного продукта необходимо наличие следующих компонентов:

1) операционная система семейства Microsoft®Windows® (не ниже 7-ой версии);

2)система должна работать под управлением семейства операционных систем Windows 7 и выше;

Для разработки программного продукта был выбран язык С#. Этот язык необходим для разработки программного приложения.

# Структура программы

На рисунке 2.1 представлена структура программы



Рисунок 2.1 - структура программы

Подпрограммы, используемые в основной программе, приведены в таблице 2.1.

Таблица 2.1

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| safeinput | Процедура безопасного ввода данных |
| calculateFunc | Процедура вычисления дифференциального уравнения методом Эйлера или усовершенствованным методом Эйлера |

Переменные, используемые в основной программе, приведены в таблице 2.2

Таблица 2.2

|  |  |
| --- | --- |
| Переменная | Назначение |
| nf | Номер дифференциального уравнения |
| nm | Номер метода решения |
| fncslist | Объект, содержащий выбранное дифференциальное уравнение |
| num | Количество итераций для вычисления интервала решения дифференциального уравнения |
| h | Шаг итерации |
| e | Объект типа errlist, генерирующийся при перехвате исключения в результате выбора несуществующего ДУ для объекта fncslist |

Подпрограммы, используемые в модуле list, приведены в таблице 2.3.

Таблица 2.3

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| outputlist | Вывод списка всех дифференциальных уравнений |
| f0 | Функция, возвращающая значение ДУ: |
| f1 | Функция, возвращающая значение ДУ: |
| f2 | Функция, возвращающая значение ДУ: |
| f3 | Функция, возвращающая значение ДУ: |
| f4 | Функция, возвращающая значение ДУ: |
| f5 | Функция, возвращающая значение ДУ: |
| f6 | Функция, возвращающая значение ДУ: |
| f7 | Функция, возвращающая значение ДУ: |
| f8 | Функция, возвращающая значение ДУ: |
| f9 | Функция, возвращающая значение ДУ: |

Константы, используемые в модуле list, приведены в таблице 2.5.

Таблица 2.5

|  |  |
| --- | --- |
| Константа | Назначение |
| listlen | Количество дифференциальных уравнений |
| flist | Массив объектов структуры типа funcs, содержащий виды дифференциального уравнения, их начальные условия и указатели на соответствующие функции |
| currf | Текущее используемое ДУ |

# Схемы алгоритма программы

## 3.1 Схема алгоритма основной программы



## 3.2 Схема алгоритма процедуры calculateFunc



## 3.3 Схема алгоритма процедуры safeinput



## 3.4 Схема алгоритма процедуры outputlist()



## 3.5 Схема алгоритма конструктора list



# Отладка программы

В процессе разработки программы были обнаружены синтаксические ошибки, ошибки компоновки и ошибки выполнения.

## 4.1 Синтаксические ошибки, обнаруженные в программе

При компилировании программы компилятор указывал на ошибки, допущенные при кодировании, а также на их место возникновения, что значительно сократило время на устранения выявленных проблем.

Ошибка: « Error C2238: непредвиденные лексемы перед ";" » – указывала на следующий фрагмент программы:

class errlist

{

public:

wchar\_t \*errtype;

int errvalue;

errlist(wchar\_t\*, int);

}

list(int);

– и была причиной того, что после определения класса errlist отсутствовала точка с запятой. Исправленный фрагмент программы выглядит так:

class errlist

{

public:

wchar\_t \*errtype;

int errvalue;

errlist(wchar\_t\*, int);

};

list(int);

## 4.2 Ошибки компоновки, обнаруженные в программе

При попытке запустить программу на выполнение, компоновщик выдавал следующее сообщение об ошибке:

1>list.obj : error LNK2001: неразрешенный внешний символ ""private: static double \_\_cdecl list::f9(double,double)" (?f9@list@@CANNN@Z)"

1>C:\Users\Шостак\Documents\Visual Studio 2010\Projects\CourseProjectV2\Debug\CourseProjectV2.exe : fatal error LNK1120: 1 неразрешенных внешних элементов

Из сообщения видно, что неразрешенной оказалась ссылка, указывающая на функцию модуля list – double list::f9(double, double). Проблема заключалась в том, что в заголовочном файле данного модуля был объявлен прототип функции, однако в исходном файле её определение отсутствовало. Устранение данной ошибки свелось к написанию заглушки для функции в исходном файле с последующим написанием вместо неё (заглушки) программы.

## 4.3 Ошибки выполнения, обнаруженные в программе

Ошибки выполнения появились при написании защиты от ошибок. Проблема заключалась в зацикливании или лишних итерациях (с игнорированием входного потока cin) цикла, который должен был обеспечить безопасность и корректность ввода данных в переменные. Для наглядности, на рисунке 4.1 проиллюстрирована суть проблемы:

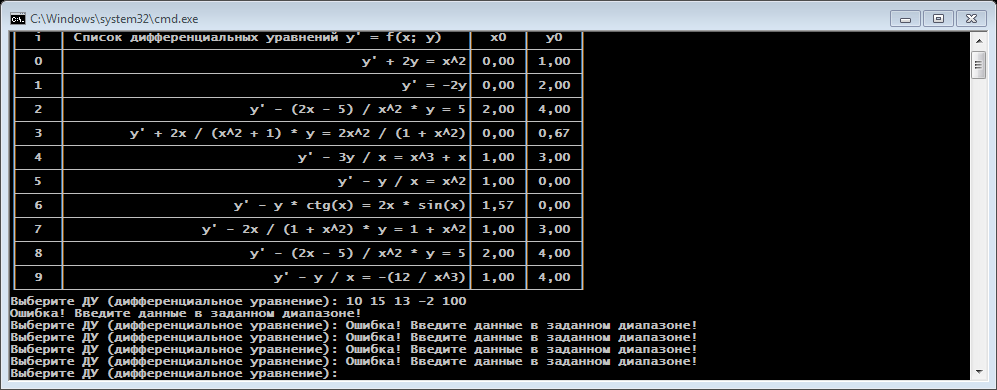


Рисунок 4.1 - Лишние итерации цикла при некорректных данных

Если же в качестве значений целочисленной переменной вводились не просто избыточные данные, выходящие за допустимый диапазон значений, а символы, то программа окончательно прекращала работать в результате зацикливания. Рисунок 4.2 демонстрирует данную проблему:

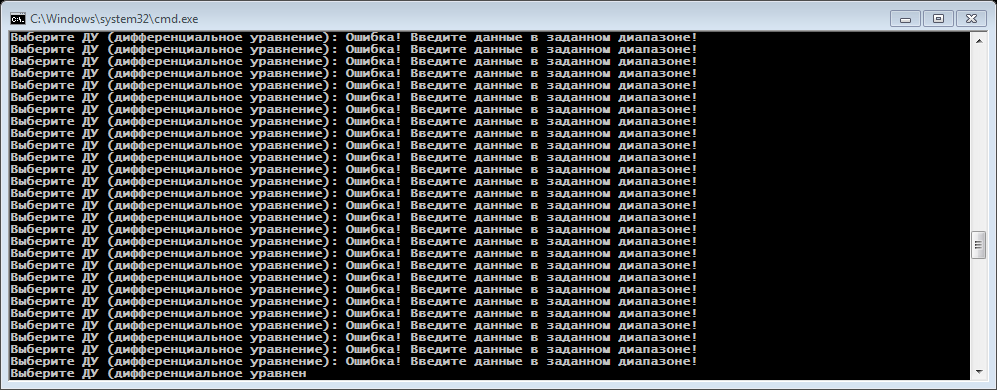


Рисунок 4.2 - Зацикливание программы в результате ввода символов в целочисленную переменную

Фрагмент программы, на котором происходил сбой, был известен, так как его тестирование производилось сразу после написания. Его код представлен ниже:

nf = -1;

bool flag = false;

while(flag != true)

{

wcout << L"Выберите ДУ (дифференциальное уравнение): ";

cin >> nf;

if(nf < 0 || nf > 9)

wcout << L"Ошибка! Введите данные в заданном диапазоне!" << endl;

else

flag = true;

}

Можно было предположить, что проблема крылась во входном потоке cin, так как именно после использования данного объекта возникали проблемы. Для точной диагностики данной ошибки и её устранения применялся метод индукции, так как необходимо было изучить симптомы этого явления, организовать данные о нем воедино и изучить взаимосвязь этих данных для выдвижения гипотезы и её последующего доказательства.

В ходе тестирования программы и изучения симптомов ошибки были замечены следующие закономерности:

1. при вводе избыточной информации количество итераций цикла равнялось количеству введенных в избытке некорректных данных;
2. при вводе символов происходило зацикливание программы.

В обоих случаях входной поток cin игнорировался: либо определенное количество раз в первом случае, либо на всем последующем протяжении выполнения программы во втором. Для объяснения данных проблем необходимо было изучить работу объекта cin более детально. В процессе изучения выявились следующие факты:

1. объект cin взаимодействует с потоковым буфером, и производит чтение из него;
2. все данные, вводимые с клавиатуры, отправляются в потоковый буфер, включая пробельные и управляющие символы. Если буфер пустой, то средствами языка процессору посылается команда прерывания для ввода данных с клавиатуры, после чего дальнейшее выполнение программы приостанавливается до тех пор, пока в буфере не будет встречен символ EOF. Если же в буфере уже есть данные, то прерывание не срабатывает, и программа продолжает свое выполнение;
3. в классе объекта cin перегружена операция побитового сдвига вправо, которая служит для вывода информации из буфера в операнд, стоящий справа от операции сдвига. Запись значения в данный операнд будет производиться до первого пробельного символа. Все уже считанные данные из потокового буфера стираются;
4. объект cin наследует от класса ios флаги статуса ошибок. Если вводимые данные имеют тип, отличный от типа принимающей эти данные переменной, то срабатывает флаг failbit, что в дальнейшем приводит к остановке чтения данных из буфера. Методы объекта cin позволяют проверять состояние флагов потока.

Исходя из выше перечисленных фактов, можно выдвинуть гипотезы, объясняющие поведение программы в тех или иных исключительных условиях:

1. при вводе избыточной информации, в буфер поступает несколько значений. В переменную nf записывается только первое из этих значений, остальные же пропускаются. Если первое из значений не проходит проверку, то за этим следует новая итерация цикла, которая выдает сообщение пользователю с просьбой ввести данные. Однако уже для этого ввода не срабатывает прерывание, т.к. буфер был заполнен при предыдущей итерации цикла и там остались данные, которые последовательно считываются в переменную. Зацикливание продолжается до тех пор, пока буфер не очистится. Затем, после его очистки, снова срабатывает прерывание, и пользователь может ввести информацию;
2. при вводе символов в буфер и последующей попытке считывания из буфера (посредством объекта cin) этих символов в целочисленную переменную nf, срабатывает флаг статуса ошибки failbit, который блокирует считывание данных из буфера в переменную. При этом сама переменная сохраняет свое исходное значение (в данном случае nf = -1). При последующих итерациях считывания уже не происходит, т.к. флаг failbit теперь равен единице. Как результат, переменная nf всегда равна -1 и происходит вечное зацикливание.

Для однозначного доказательства данных гипотез и исправления ошибки необходимо принять следующие меры:

1. очищать буфер целиком после записи в него данных и считывания в переменную первого значения из всех введенных. Очистка буфера может осуществляться посредством функции fflush() (с указанием потока стандартного ввода stdin в качестве параметра);
2. проверять биты статуса ошибок. В случае их обнаружения очищать буфер и сбрасывать флаги ошибок.

Исправленный фрагмент программы:

// Очистка буфера и сброс флагов ошибок,

// если таковые имелись до текущего запроса

// на ввод данных

cin.clear();

fflush(stdin);

// Цикл, выход из которого возможен только

// при успешном вводе данных

while(true)

{

// Запрос на ввод данных

wcout << L"Выберите ДУ (дифференциальное уравнение): ";

cin >> nf;

// Если биты статуса ошибок равны 0,

// то выходим из цикла

if(cin.good() && (nf >= 0) && (nf < 10))

break;

// Иначе выводим сообщение об ошибке,

// сбрасываем флаги ошибок и очищаем буфер

else

{

wcout << L"Ошибка ввода! Введите корректные данные!" << endl;

cin.clear();

fflush(stdin);

}

}

В результате учета всех нюансов работы с входным потоком cin и изменения алгоритма проверки вводимых данных, программа стала корректно работать при тестировании в исключительных ситуациях, что проиллюстрировано на рисунке 4.3.

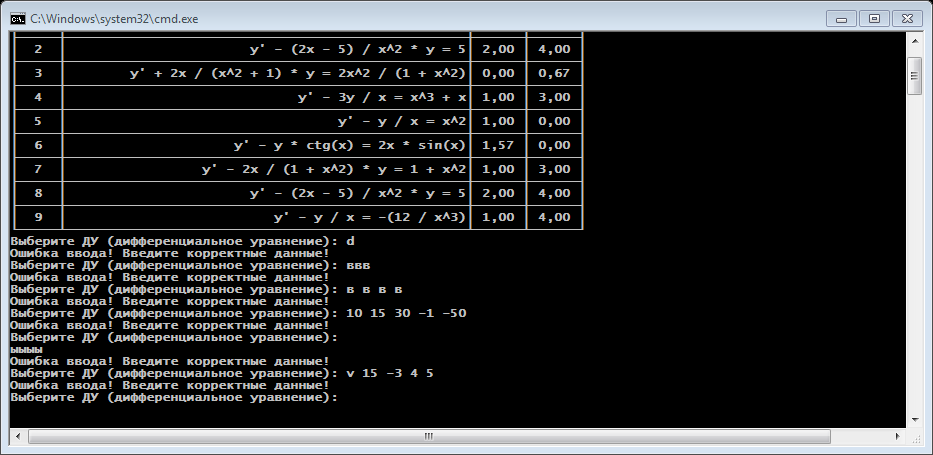


Рисунок 4.3 - Результат работы отлаженной программы

Все ошибки были исправлены. Для того, чтобы убедиться в правильности работы программы, необходимо провести тестирование.

# Оптимизация программы

Основной задачей программирования является создание правильных, а не эффективных программ. Зачастую для повышения производительности программ большую значимость имеют такие факторы как:

* четко выработанные требования к программе до начала её разработки;
* хорошая архитектура программы;
* хорошо спроектированные модули;
* правильно выбранный компилятор со встроенными средствами оптимизации.

Ввиду данных факторов основной целью разработки являлось создание правильной, хорошо спроектированной программы.

В большинстве случаев к оптимизации следует приступать только после того, как программа создана и выдает требуемые результаты. В противном случае можно получить не просто неудобочитаемый код и неработающую программу, но еще и неоптимизированное решение. Однако неизвестно, насколько эффективной будет программа, оптимизированная после её разработки, поэтому определять требования к эффективности следует на стадии проектирования.

Несмотря на вышеперечисленные факты, оптимизация оказывается очень полезной, когда программа работает правильно. Производя небольшие изменения в некоторых фрагментах кода, называемых критическими областями, оптимизация помогает повысить эффективность программы в несколько раз. Так, Дональд Кнут в своей классической работе “Эмпирическое исследование программ, написанных на Фортран” обнаружил, что менее 4% кода обычно соответствуют более чем 50% времени выполнения программы.

Так как современные ЭВМ отличаются высоким быстродействием, то оптимизация отдельных, редко встречающихся операторов является бесполезной тратой времени. Однако повышение эффективности на мощных машинах можно получить за счет правильной организации циклов и операторов, находящихся внутри тела циклов.

Программа, разрабатываемая в рамках курсового проекта, была оптимизирована как по памяти, так и по времени. Для оптимизации по памяти программа была разделена на модули и организованна таким образом, чтобы исключалось дублирование исходных данных и структурных типов. Например, все данные о дифференциальных уравнениях и соответствующие им функции располагались в классе list. При этом структурный константный массив flist в данном модуле был объявлен статическим, а это значит, что при создании экземпляров класса не будет происходить его дублирования, и, как следствие, будет экономиться память. Соответствующие фрагменты программы приведены ниже:

Определение переменной в заголовочном файле:

const static funcs flist[];

Инициализация переменной в исходном файле:

const list::funcs list::flist[] =

{

{L"y' + 2y = x^2", 0.0, 1.0, list::f0},

{L"y' = -2y", 0.0, 2.0, list::f1},

{L"y' - (2x - 5) / x^2 \* y = 5", 2.0, 4.0, list::f2},

{L"y' + 2x / (x^2 + 1) \* y = 2x^2 / (1 + x^2)", 0.0, 2.0 / 3.0, list::f3},

{L"y' - 3y / x = x^3 + x", 1.0, 3.0, list::f4},

{L"y' - y / x = x^2", 1.0, 0.0, list::f5},

{L"y' - y \* ctg(x) = 2x \* sin(x)", 3.14159265 / 2.0, 0.0, list::f6},

{L"y' - 2x / (1 + x^2) \* y = 1 + x^2", 1.0, 3.0, list::f7},

{L"y' - (2x - 5) / x^2 \* y = 5", 2.0, 4.0, list::f8},

{L"y' - y / x = -(12 / x^3)", 1.0, 4.0, list::f9}

};

Для оптимизации программы по времени были произведены следующие действия:

* исключены необоснованные, многократные обращения к элементам массивов по индексам. Например, элемент массива flist, с которым будет работать программа, выбирается сразу при создании объекта в конструкторе класса (посредством указателя currf на данный элемент):

list::list(int nf)

{

if((nf < listlen) && (nf >= 0))

currf = flist + nf;

else

throw errlist(L"Ошибка инициализации! Выход за пределы массива!", nf);

}

Теперь программа будет обращаться к данному указателю, а не непосредственно к массиву flist, что исключает индексирование:

const list::funcs\* list::operator -> ()

{

return currf;

}

* минимизированы преобразования типов в выражениях. Например, следующий фрагмент демонстрирует исключение преобразования типов в выражении, где все переменные и литерал являются вещественными:

yhalfh = y + h \* 0.5 \* fxy;

* более долгие для выполнения операции, такие как возведение в степень и деление, были заменены на операцию умножения. Приведенный выше фрагмент может служить примером, где литерал 1/2, участвующий в арифметическом выражении, заменен на 0.5.

# Тестирование программы

Тестирование – это набор процедур и действий, предназначенных для демонстрации правильности работы программы в заданных режимах и внешних условиях. Цель тестирования – выявить наличие ошибок или убедительно продемонстрировать их отсутствие.

Процесс тестирования проходит в три этапа:

* проверка поведения программы в нормальных условиях;
* проверка поведения программы в экстремальных условиях;
* проверка поведения программы в исключительных ситуациях.

Каждый из этапов предполагает задание определенного, характерного для данного этапа набора входных данных.

## 6.1 Тестирование в нормальных условиях

Для тестирования программы в нормальных условиях достаточно сформировать стартовые значения, при которых подразумевается безотказное выполнение программы. На рисунках 6.1, 6.2, 6.3 и 6.4 продемонстрировано тестирование в нормальных условиях.

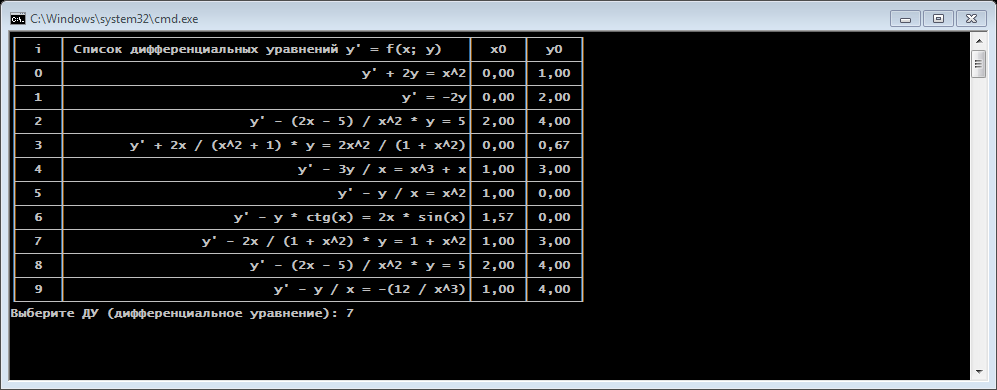


Рисунок 6.1 – Выбор дифференциального уравнения

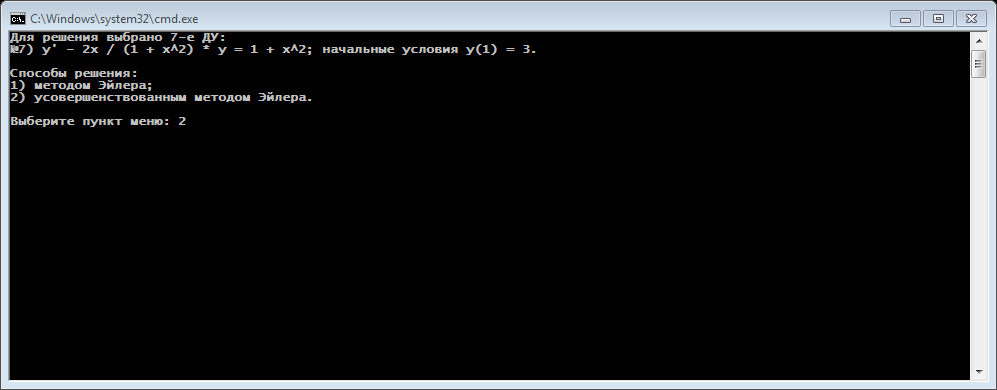


Рисунок 6.2 – Выбор решения ДУ усовершенствованным методом Эйлера

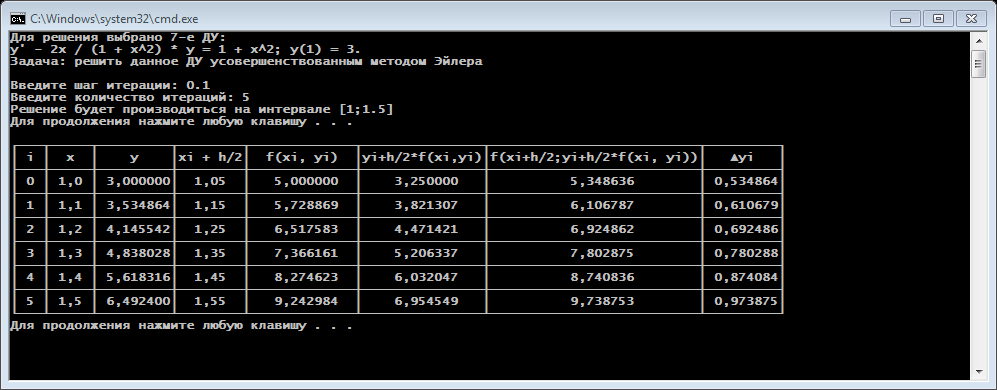


Рисунок 6.3 – Результаты вычислений с использованием усовершенствованного метода Эйлера

**Ручной расчет** к результатам, отображенным на рисунке 6.3:

ЗАКЛЮЧЕНИЕ

Целью данного курсового проекта являлось написание программы по теме: «Решение дифференциальных уравнений», которая была достигнута в ходе выполненной работы.

В ходе написания курсового проекта были получены практические навыки в области проектирования программы. Программа разрабатывалась с использованием стратегии анализа сообщений, реализующей метод нисходящей разработки. Анализ основывался на изучении потоков данных, обрабатываемых программой. Опыт, полученный при написании работы, подтвердил необходимость в первую очередь правильной выработки требований к программе и грамотного подхода к её проектированию перед началом написания кода.

Так же были получены практические навыки при оптимизации и тестировании программы. Появилось понимание важности и сложности отладки, её затратности по времени относительно всего процесса разработки, необходимости знания аспектов из разных областей IT-технологий, как, например, знание аппаратной части, операционных систем, самих реализуемых процессов, природы и специфики различных ошибок и т.д.

Результатом проведенной работы является правильно спроектированное приложение, а также работающий, эффективный, отлаженный и корректно отвечающий тестам соответствующих условий тестирования проект.

При разработке программы были изучены и применены методы решения дифференциальных уравнений, такие как метод Эйлера и усовершенствованный метод Эйлера.

Программа имеет надлежащую структуру, схемы алгоритма и необходимые комментарии.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Жмакин А.П. Архитектура ЭВМ. «БХВ-Петербург», 2010.
2. Иртегов Д. В. Введение в операционные системы. «БХВ-Петербург», 2008.
3. Керниган Брайан, Ритчи Деннис. Язык программирования С. «Вильямс», 2017.
4. Киркинский А.С. Математический анализ. «Академический Проект», 2006.
5. Лафоре Роберт. Объектно-ориентированное программирование в C++. «Питер», 2015.
6. Макконнелл Стив. Совершенный код. «БХВ-Петербург», 2017.
7. Письменный Д. Т. Конспект лекций по высшей математике. «Айрис-Пресс», 2017.
8. Страуструп Бьярне. Программирование, принципы и практика использования С++. «Вильямс», 2011.
9. Таненбаум Эндрю, Бос Х. Современные операционные системы. «Питер», 2017.
10. Таненбаум Эндрю, Остин Т. Архитектура компьютера. «Питер», 2017.
11. <https://studfiles.net/preview/2543941/page:2/>
12. <https://ru.wikipedia.org/wiki/Метод_Эйлера>
13. <https://habr.com/post/276593/>
14. <https://habr.com/post/188002/>
15. <http://www.mathprofi.ru/metody_eilera_i_runge_kutty.html>

ПРИЛОЖЕНИЕ А

(обязательное)

Листинги программы

## А.1 Листинг основной программы

//#### Программа CourseProject

//#### Курсовой проект

//#### по предмету МДК 03.01 Технология разработки программного

//#### обеспечения

//#### по теме "Разработка программы решения дифференциальных

//#### уравнений"

//#### Язык: C++

//#### Разработал: Шостак А.А.

//#### Задание:

//#### Разработка программы решения дифференциальных уравнений:

//#### 1) методом Эйлера;

//#### 2) усовершенствованным методом Эйлера.

//#### Основные переменные, используемые в программе:

//#### nf - номер дифференциального уравнения;

//#### nm - номер метода решения;

//#### fncslist - объект, содержащий выбранное дифференциальное

//#### уравнение;

//#### num - количество итераций для вычисления интервала решения

//#### дифференциального уравнения;

//#### h - шаг итерации;

//#### e - объект типа errlist, генерирующийся при перехвате

//#### исключения в результате выбора несуществующего ДУ для объекта

//#### fncslist.

//#### Вызываемые подпрограммы:

//#### safeinput - шаблон функции безопасного ввода данных;

//#### calculateFunc - процедура вычисления ДУ методом Эйлера или

//#### усовершенствованным методом Эйлера;

//#### checkmethod - функция проверки на корректность выбора метода

//#### решения;

//#### checkNumOrH - перегруженная функция проверки на корректность

//#### введенного шага итерации и количества итераций.

#include "stdafx.h"

#include "list.h"

using std::wcout;

using std::endl;

using std::cin;

void calculateFunc(list f, double h, int num, char nm);

template <class T>

void safeinput(T &n, wchar\_t \*str, bool (\*condition) (T) = NULL);

// Проверка на корректность выбора метода решения

// Формальный параметр:

// nm - номер выбранного метода.

inline bool checkmethod(char nm)

{

return ((nm == '1') || (nm == '2')) && (cin.rdbuf()->in\_avail() == 1);

}

// Проверка на корректность введенного количества

// итераций

// Формальный параметр:

// n - количество итераций.

inline bool checkNumOrH(int n)

{

return (n > 0) ? true : false;

}

// Проверка на корректность введенного шага итерации

// Формальный параметр:

// n - шаг итерации.

inline bool checkNumOrH(double n)

{

return (n > 0.0 ) ? true : false;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus\_rus.866");

int nf;

char nm;

// Вывод списка ДУ на консоль

list::outputlist();

try

{

safeinput(nf, L"Выберите ДУ (дифференциальное уравнение): ");

// Создание объекта со списком ДУ и

// его настройка на выбранное ДУ

list fncslist(nf);

system("cls");

// Вывод текущей информации о проделанном

// шаге на консоль

wcout <<

L"Для решения выбрано " << nf << L"-е ДУ:\n" <<

L"№" << nf << L") " << fncslist->view <<

L"; начальные условия y(" << fncslist->x0 << ") = " << fncslist->y0 << "." <<

endl;

std::cout << endl;

wcout <<

L"Способы решения:\n" <<

L"1) методом Эйлера;\n" <<

L"2) усовершенствованным методом Эйлера.\n" <<

endl;

// Выбор способа решения ДУ

safeinput(nm, L"Выберите пункт меню: ", checkmethod);

wchar\_t \*mtype;

if(nm == '1')

mtype = L"методом Эйлера";

else

mtype = L"усовершенствованным методом Эйлера";

system("cls");

// Вывод текущей информации о проделанном

// шаге на консоль

wcout <<

L"Для решения выбрано " << nf << L"-е ДУ:\n" <<

fncslist->view << L"; y(" << fncslist->x0 << ") = " << fncslist->y0 << ".\n" <<

L"Задача: решить данное ДУ " << mtype <<

endl;

std::cout << endl;

int num = NULL;

double h = NULL;

// Безопасный ввод шага итерации

while(true)

{

safeinput(h, L"Введите шаг итерации: ", checkNumOrH);

if(h > fncslist.maxStep)

wcout << L"Ошибка! Максимальный шаг итерации равен: " << fncslist.maxStep << endl;

else

break;

}

// Безопасный ввод количества итераций

while(true)

{

safeinput(num, L"Введите количество итераций: ", checkNumOrH);

if(num > fncslist.maxIt)

wcout << L"Ошибка! Максимальное количество итераций равно: " << fncslist.maxIt << endl;

else

break;

}

// Вычисление и вывод интервала, на котором

// будет производиться решение ДУ

wcout << L"Решение будет производиться на интервале [" << fncslist->x0 << ';' << (fncslist->x0 + num \* h) << ']' << endl;

system("pause");

std::cout << endl;

// Решение ДУ и вывод таблицы с результатами

// на консоль

calculateFunc(fncslist, h, num, nm);

system("pause");

}

// Перехват исключения в случае, если номер

// ДУ выходит за пределы списка с ДУ

catch(list::errlist e)

{

wcout << e.errtype << L" Значение индекса = " << e.errvalue << L". Элементов в массиве = " << list::listlen << endl;

system("pause");

}

return 0;

}

// Вычисления ДУ методом Эйлера или усовершенствованным

// методом Эйлера

// Формальные параметры:

// f - выбранное дифференциальное уравнение;

// h - шаг итерации;

// num - количество итераций;

// nm - метод решения.

// Локальные переменные:

// x - аргумент функции;

// y - функция;

// fxy - значение производной, зависящее от x и y;

// hf - произведение шага итерации на fxy;

// xhalfh - сумма аргумента функции и половины шага итерации;

// yhalfh - сумма функции и произведения fxy на половину шага

// итерации;

// fxy2 - значение производной, зависящее от xhalfh и yhalfh;

// i - управляющая переменная-счетчик.

void calculateFunc(list f, double h, int num, char nm)

{

double fxy, hf;

// Инициализация переменных начальными

// значениями

double x = f->x0, y = f->y0;

// Если выбран первый пункт, то решение

// производится методом Эйлера

if(nm == '1')

{

wcout << L"┌─────┬───────┬─────────────────┬─────────────────┬────────────────┐" << endl;

wcout << L"│ i │ x │ y │ f(xi, yi) │ hf(xi, yi) │" << endl;

wcout << L"├─────┼───────┼─────────────────┼─────────────────┼────────────────┤" << endl;

// Цикл с построением ломанных прямых,

// параллельных касательным, проходящим по

// левым краям интервала разбиения функции

for(int i = 0; i <= num; i++)

{

// Вычисление значения ДУ в заданных точках

fxy = f->ptrf(x, y);

hf = h \* fxy;

wprintf(L"│%3i │%5.1f │%12.5f │%12.5f │%12.5f │\n", i, x, y, fxy, hf);

// Переход к следующим точкам на заданном

// интервале

x += h;

y += hf;

if(i < num)

wcout << L"├─────┼───────┼─────────────────┼─────────────────┼────────────────┤" << endl;

}

wcout << L"└─────┴───────┴─────────────────┴─────────────────┴────────────────┘" << endl;

}

// Если выбран второй пункт, то решение

// производится усовершенствованным методом

// Эйлера

else

{

// Определение вспомогательных переменных,

// посредством которых, будет производиться

// уточнение в решении

double xhalfh, yhalfh, fxy2;

wcout << L"┌───┬─────┬─────────┬────────┬─────────────┬───────────────┬──────────────────────────┬─────────┐" << endl;

wcout << L"│ i │ x │ y │xi + h/2│ f(xi, yi) │yi+h/2\*f(xi,yi)│f(xi+h/2;yi+h/2\*f(xi, yi))│ ▲yi │" << endl;

wcout << L"├───┼─────┼─────────┼────────┼─────────────┼───────────────┼──────────────────────────┼─────────┤" << endl;

// Цикл с построением ломанных прямых,

// параллельных касательным, проходящим по

// середине интервала разбиения функции

for(int i = 0; i <= num; i++)

{

// Вычисление значения ДУ в заданных точках

fxy = f->ptrf(x, y);

// Уточнение решения

xhalfh = x + h \* 0.5;

yhalfh = y + h \* 0.5 \* fxy;

fxy2 = f->ptrf(xhalfh, yhalfh);

hf = h \* fxy2;

wprintf(L"│%2i │%4.1f │ %0.6f│%6.2f │ %10.6f │%12.6f │%18.6f │%9.6f│\n", i, x, y, xhalfh, fxy, yhalfh, fxy2, hf);

// Переход к следующим точкам на заданном

// интервале

x += h;

y += hf;

if(i < num)

wcout << L"├───┼─────┼─────────┼────────┼─────────────┼───────────────┼──────────────────────────┼─────────┤" << endl;

}

wcout << L"└───┴─────┴─────────┴────────┴─────────────┴───────────────┴──────────────────────────┴─────────┘" << endl;

}

}

// Безопасный ввод данных

// Формальные параметры:

// n - переменная, в которую необходимо записать данные;

// str - сообщение, выводимое на консоль при вводе;

// condition - дополнительное условие для безопасного ввода данных.

template <class T>

void safeinput(T &n, wchar\_t \*str, bool (\*condition) (T))

{

// Сброс флагов ошибок входного потока

// и очистка его буфера

cin.clear();

fflush(stdin);

while(true)

{

// Вывод сообщения на консоль с запросом на

// ввод значения переменной

wcout << str;

cin >> n;

// Если после ввода во входном потоке

// отсутствуют ошибки и не заданы

// дополнительные условия, то происходит

// выход из цикла

if(cin.good() && (condition == NULL))

break;

// Если заданы дополнительные условия и они

// выполняются, то так же происходи выход из

// цикла

else if(cin.good() && condition(n))

break;

// Иначе выводится сообщение об ошибке,

// очищается буфер и флаги битов ошибок

// входного потока, и производится новая

// итерация с запросом на ввод данных

else

{

wcout << L"Ошибка ввода! Введите корректные данные!" << endl;

cin.clear();

fflush(stdin);

}

}

}

## А.2 Листинг модуля list

//#### Модуль list

//#### Константы, используемые в модуле:

//#### flist - массив объектов структуры типа funcs, содержащий виды

//#### дифференциального уравнения, их начальные условия и указатели

//#### на соответствующие функции;

//#### listlen - количество дифференциальных уравнений;

//#### currf - текущее используемое дифференциальное уравнение;

//#### maxIt - максимальное количество итераций для решения ДУ;

//#### maxStep - максимальный шаг итерации для решения ДУ.

//#### Подпрограммы модуля:

//#### list - конструктор, задающий начальные настройки для работы с

//#### модулем (объектом);

//#### outputlist - процедура вывода списка имеющихся

//#### дифференциальных уравнений на консоль;

//#### f0..f9 - функции, вычисляющие значения производных

//#### соответствующих дифференциальных уравнений.

class list

{

private:

struct funcs

{

wchar\_t\* view;

double x0, y0;

double (\*ptrf) (double, double);

};

const static funcs flist[];

const funcs \*currf;

static inline double

f0(double, double),

f1(double, double),

f2(double, double),

f3(double, double),

f4(double, double),

f5(double, double),

f6(double, double),

f7(double, double),

f8(double, double),

f9(double, double);

public:

const static int maxIt;

const static double maxStep;

const static int listlen;

class errlist

{

public:

wchar\_t \*errtype;

int errvalue;

errlist(wchar\_t\*, int);

};

list(int);

static void outputlist();

const funcs\* operator -> ();

};

#include "list.h"

#include "math.h"

using std::wcout;

using std::endl;

const int list::maxIt = 10;

const double list::maxStep = 2;

// Определение массива с дифференциальными

// уравнениями

const list::funcs list::flist[] =

{

{L"y' + 2y = x^2", 0.0, 1.0, list::f0},

{L"y' = -2y", 0.0, 2.0, list::f1},

{L"y' - (2x - 5) / x^2 \* y = 5", 2.0, 4.0, list::f2},

{L"y' + 2x / (x^2 + 1) \* y = 2x^2 / (1 + x^2)", 0.0, 2.0 / 3.0, list::f3},

{L"y' - 3y / x = x^3 + x", 1.0, 3.0, list::f4},

{L"y' - y / x = x^2", 1.0, 0.0, list::f5},

{L"y' - y \* ctg(x) = 2x \* sin(x)", 3.14159265 / 2.0, 0.0, list::f6},

{L"y' - 2x / (1 + x^2) \* y = 1 + x^2", 1.0, 3.0, list::f7},

{L"y' - (2x - 5) / x^2 \* y = 5", 2.0, 4.0, list::f8},

{L"y' - y / x = -(12 / x^3)", 1.0, 4.0, list::f9}

};

// Вычисление длины массива

const int list::listlen = sizeof(flist) / sizeof(flist[0]);

// Конструктор, задающий начальные настройки для работы с модулем

// Формальный параметр:

// nf - номер выбранного ДУ.

list::list(int nf)

{

// Если nf не выходит за границы массива,

// то устанавливаем указатель на ДУ с номером

// nf

if((nf < listlen) && (nf >= 0))

currf = flist + nf;

// Иначе генерируем исключение

else

throw errlist(L"Ошибка инициализации! Выход за пределы массива!", nf);

}

// Вывод списка имеющихся дифференциальных уравнений на консоль

// Локальная переменная:

// i - управляющая переменная-счетчик.

void list::outputlist()

{

wcout << L"┌─────┬──────────────────────────────────────────────────┬──────┬──────┐" << endl;

wcout << L"│ i │ Список дифференциальных уравнений y' = f(x; y) │ x0 │ y0 │" << endl;

wcout << L"├─────┼──────────────────────────────────────────────────┼──────┼──────┤" << endl;

// Цикл вывода дифференциальных уравнений

// массива flist

for(int i = 0; i < listlen; i++)

{

wprintf(L"│ %i │%\*s│%5.2f │%5.2f │\n", i, 50, flist[i].view, flist[i].x0, flist[i].y0);

if(i < listlen - 1)

wcout << L"├─────┼──────────────────────────────────────────────────┼──────┼──────┤" << endl;

}

wcout << L"└─────┴──────────────────────────────────────────────────┴──────┴──────┘" << endl;

}

const list::funcs\* list::operator -> ()

{

return currf;

}

// Функция дифференциального уравнения f0

double list::f0(double x, double y)

{

return x \* x - 2.0 \* y;

}

// Функция дифференциального уравнения f1

double list::f1(double x, double y)

{

return (-2.0) \* y;

}

// Функция дифференциального уравнения f2

double list::f2(double x, double y)

{

return (2.0 \* x - 5.0) / (x \* x) \* y + 5.0;

}

// Функция дифференциального уравнения f3

double list::f3(double x, double y)

{

double sqrx = x \* x, incsqrx = sqrx + 1.0;

return (2.0 \* sqrx) / incsqrx - (2.0 \* x) / incsqrx \* y;

}

// Функция дифференциального уравнения f4

double list::f4(double x, double y)

{

return 3 \* y / x + x \* x \* x + x;

}

// Функция дифференциального уравнения f5

double list::f5(double x, double y)

{

return y / x + x \* x;

}

// Функция дифференциального уравнения f6

double list::f6(double x, double y)

{

return y \* (cos(x) / sin(x)) + 2.0 \* x \* sin(x);

}

// Функция дифференциального уравнения f7

double list::f7(double x, double y)

{

return 2.0 \* x / (1.0 + x \* x) \* y + 1.0 + x \* x;

}

// Функция дифференциального уравнения f8

double list::f8(double x, double y)

{

return (2.0 \* x - 5.0) / (x \* x) \* y + 5.0;

}

// Функция дифференциального уравнения f9

double list::f9(double x, double y)

{

return y / x - 12.0 / (x \* x \* x);

}

ПРИЛОЖЕНИЕ Б

(обязательное)

Результаты выполнения программы

На рисунках Б.1, Б.2, Б.3, Б.4, Б.5, Б.6 продемонстрированы результаты решения нескольких ДУ методом Эйлера и усовершенствованным методом Эйлера.

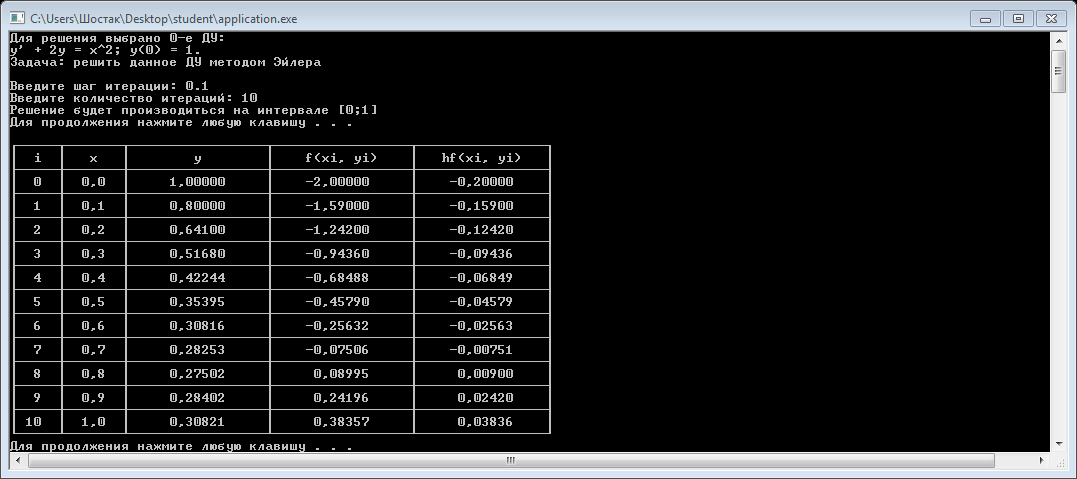


Рисунок Б.1 – Решение ДУ с номером 0 методом Эйлера

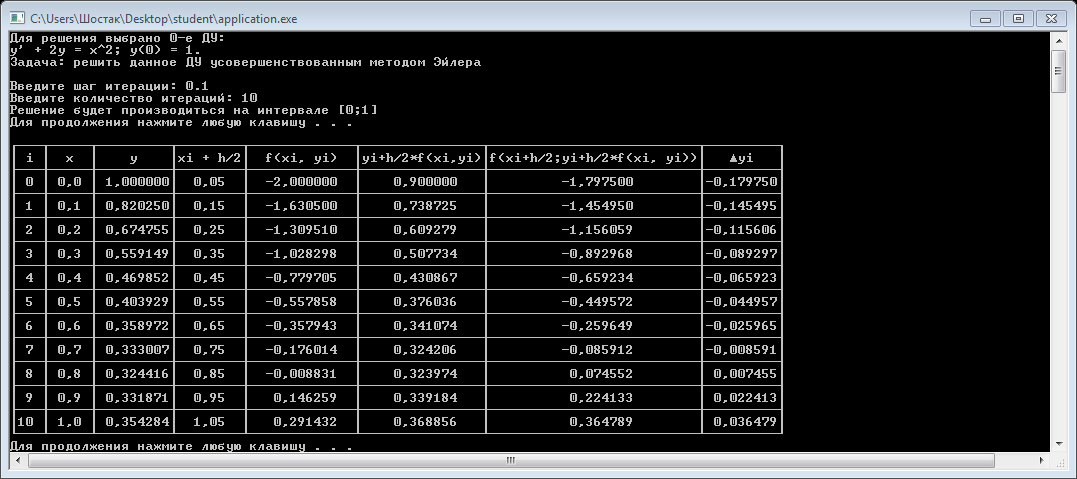


Рисунок Б.2 – Решение ДУ с номером 0 усовершенствованным методом Эйлера

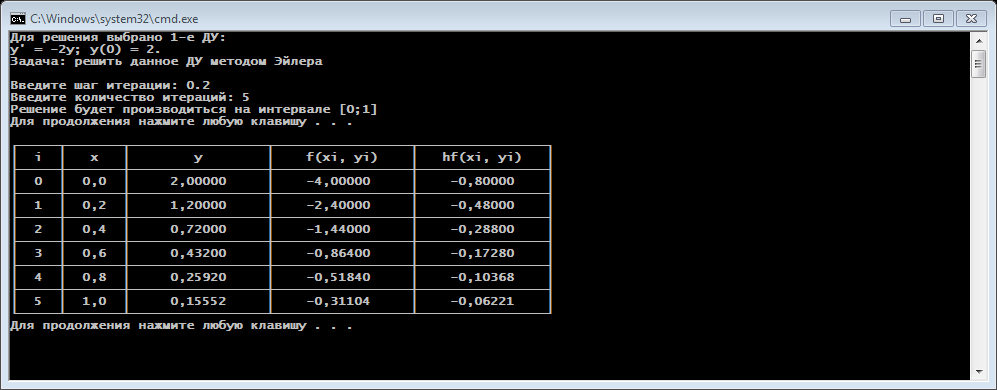


Рисунок Б.3 – Решение ДУ с номером 1 методом Эйлера

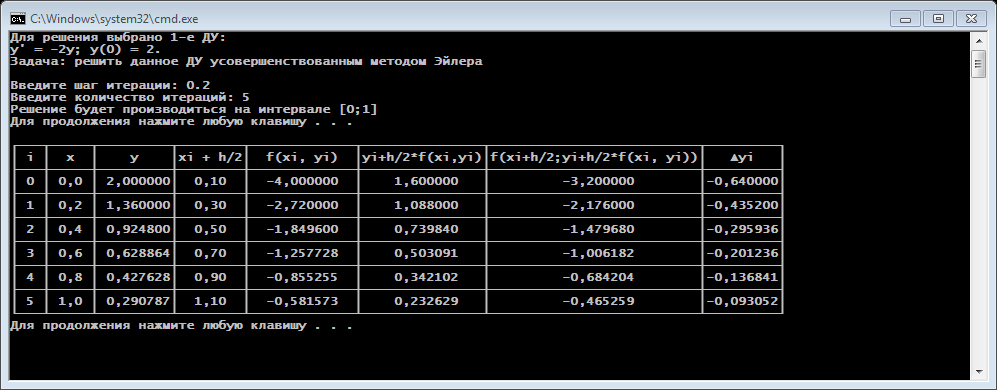


Рисунок Б.4 – Решение ДУ с номером 1 усовершенствованным методом Эйлера

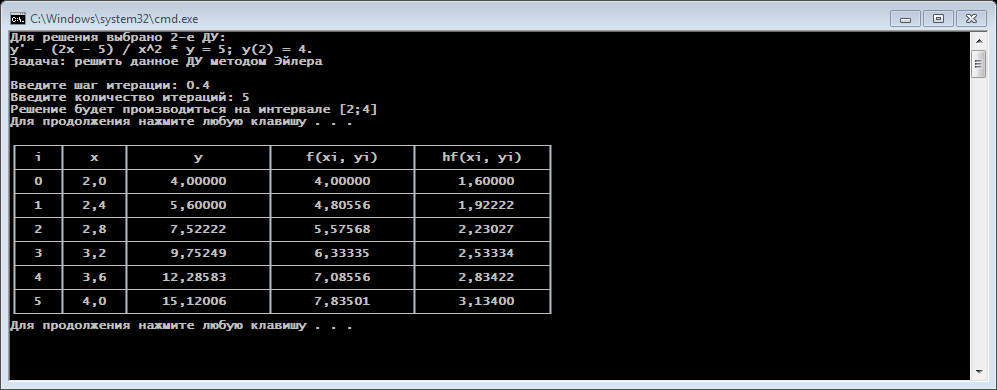


Рисунок Б.5 – Решение ДУ с номером 2 методом Эйлера

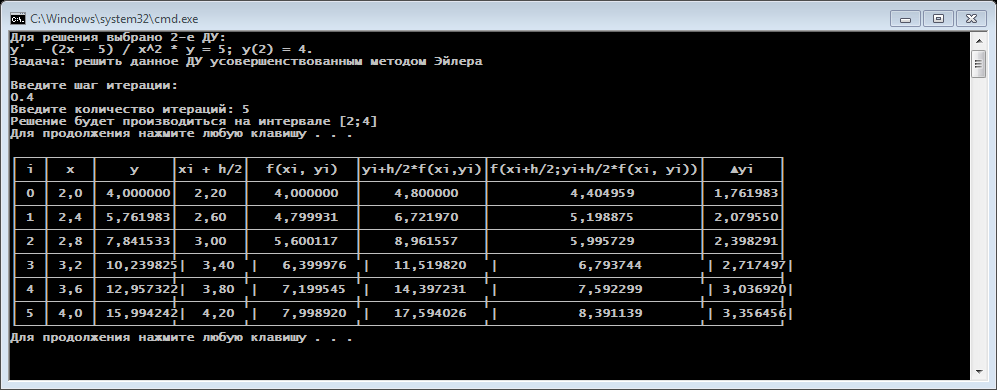


Рисунок Б.6 – Решение ДУ с номером 2 усовершенствованным методом Эйлера