Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский государственный технический университет имени Н. Э. Баумана**

**(национальный исследовательский университет)»**

**Московский техникум космического приборостроения**

**Специальность: 09.02.03 Программирование в компьютерных системах**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту по теме:**

РАЗРАБОТКА ПРОГРАММЫ РЕШЕНИЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

|  |  |  |
| --- | --- | --- |
| Руководитель разработки от техникума | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) | Е. С. Демина  (Ф.И.О.) |
| Разработчик | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) | П. О. Королёв  (Ф.И.О.) |

Москва 2019

**«Московский государственный технический университет имени Н. Э. Баумана**

**(национальный исследовательский университет)»**

**Московский техникум космического приборостроения**

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

|  |
| --- |
| УТВЕРЖДАЮ  Председатель ПЦК спец. 09.02.03  Н. А. Жилкина  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г. |

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по профессиональному модулю «МДК 03.01 Технология разработки программного обеспечения»

Студент П. О. Королёв ТМП-61

(фамилия, инициалы, индекс группы)

Руководитель Е. С. Демина \_\_

(фамилия, инициалы)

График выполнения работы: 25% к 4 нед., 50% к 8 нед., 75% к 12 нед., 100% к 15 нед.

1. ***Тема курсового проекта***

Разработка программы решения дифференциальных уравнений.

1. ***Техническое задание***

Разработать программу решения дифференциальных уравнений:

|  |
| --- |
| − методом Эйлера; |
| −усовершенствованным методом Эйлера. |

1. ***Оформление курсового проекта***
   1. Расчетно-пояснительная записка на \_\_64\_\_ листах формата А4.
   2. Перечень графического материала КП (плакаты, схемы, чертежи и т.п.) – схемы алгоритма

Дата выдачи задания «16» января 2019 г.

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. С. Демина

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5

1 Постановка задачи 7

1.1 Метод Эйлера 7

1.2 Усовершенствованный метод Эйлера 8

1.3 Системные требования, предъявляемые к программе 8

2 Структура программы 10

3 Схемы алгоритма программы 13

3.1 Схема алгоритма основной программы 13

3.2 Схемы алгоритма модуля Methods\_DE 14

3.2.1 Схема алгоритма функции Euler() 14

3.2.2 Схема алгоритма функции ImprovedEuler() 15

3.3 Схемы алгоритма модуля Input 16

3.3.1 Схема алгоритма функции catan() 16

3.3.2 Схема алгоритма функции acatan() 17

3.3.3 Схема алгоритма функции Enter\_float() 18

3.3.4 Схема алгоритма функции Enter\_int() 19

3.3.5 Схема алгоритма функции Formula() 20

3.3.6 Схема алгоритма функции CheckErrorFormula() 21

4 Отладка программы 22

4.1 Синтаксические ошибки, обнаруженные в программе 22

4.2 Ошибки компоновки, обнаруженные в программе 23

4.3 Ошибки выполнения, обнаруженные в программе 23

5 Оптимизация программы 25

6 Тестирование программы 28

6.1 Тестирование в нормальных условиях 28

6.2 Тестирование в экстремальных условиях 32

6.3 Тестирование в исключительных условиях 36

7 Руководство пользователя 39

Заключение 42

Список использованных источников 43

Приложение А 44

А.1 Листинг основной программы 45

A.2 Листинг модуля Methods\_DE 48

A.3 Листинг модуля Input 53

Приложение Б Результаты выполнения программы 61

ВВЕДЕНИЕ

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

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 котором 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.

Основной теорией дифференциальных уравнений стало дифференциальное исчисление, созданное Лейбницем и Ньютоном (1642-1727). Сам термин «дифференциальное уравнение» был предложен в 1676 году Лейбницем.

Из огромного числа работ XVIII века по дифференциальным уравнениям выделяются работы Эйлера (1707-1783) и Лагранжа (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.

1 Постановка задачи

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

* методом Эйлера;
* усовершенствованным методом Эйлера.

1.1 Метод Эйлера

Метод Эйлера — простейший численный метод решения систем дифференциальных уравнений. Впервые описан Леонардом Эйлером в 1768 году в работе «Интегральное исчисление». Метод Эйлера является явным, одношаговым методом первого порядка точности. Он основан на аппроксимации интегральной кривой кусочно-линейной функцией, так называемой ломаной 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 , вычисляется по формуле:

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, замещающий 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 и 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 – 950;
* количество ОЗУ от 4 гб;
* место на жестком диске от 15 мб;
* любая встроенная или дискретная видеокарта.

Программа должна быть написана на языке программирования Python.

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объем полезных функций.

Преимущества языка Python:

* кроссплатформенность. Интерпретатор языка Python реализован практически на всех платформах и операционных системах;
* расширяемость языка, то есть имеется возможность совершенствования языка всеми заинтересованными программистами. Интерпретатор написан на языке программирования С и исходный код доступен для любых манипуляций;
* наличие большого числа подключаемых к программе модулей, обеспечивающих различные дополнительные возможности;
* Python обладает простым и понятным синтаксисом, что позволяет создавать легко-читаемый код.

Ввиду выше перечисленных фактов о языке Python, играющих важную роль при разработке любого ПО, а также благодаря множеству надежных библиотек, которые имеются в наличии у данного языка, для решения поставленной задачи был выбран язык программирования Python.

2 Структура программы

На рисунке 2.1 представлена структура программы.



Рисунок 2.1 – Структура программы

Модули, используемые в программе, приведены в таблице 2.1.

Таблица 2.1

|  |  |
| --- | --- |
| Модуль | Назначение |
| Methods\_DE | Модуль для решения ДУ |
| Input | Модуль для ввода и проверки переменных |

Подпрограммы, используемые в модуле Methods\_DE, приведены в таблице 2.2.

Таблица 2.2

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| Euler | Функция вычисления дифференциального уравнения методом Эйлера |
| ImprovedEuler | Функция вычисления дифференциального уравнения усовершенствованным методом Эйлера |

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

Таблица 2.3

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| catan | Функция котангенса |
| acatan | Функция арккотангенса |
| Enter\_float | Функция безопасного ввода вещественных данных |
| Enter\_int | Функция безопасного ввода целочисленных данных |
| Formula | Функция безопасного ввода ДУ |
| CheckErrorFormula | Функция определения допустимости значений Х и У в введенном уравнении |

Переменные, используемые в основной программе, приведены в таблице 2.4.

Таблица 2.4

|  |  |
| --- | --- |
| Переменная | Назначение |
| Flag | Флаг для выхода из цикла |
| exit | Переменная, отвечающая за выход из программы по нажатию клавиши |
| Сhoice | Переменная для выбора номера метода решения ДУ |

3 Схемы алгоритма программы

3.1 Схема алгоритма основной программы



3.2 Схемы алгоритма модуля Methods\_DE

3.2.1 Схема алгоритма функции Euler()



3.2.2 Схема алгоритма функции ImprovedEuler()



3.3 Схемы алгоритма модуля Input

3.3.1 Схема алгоритма функции catan()



3.3.2 Схема алгоритма функции acatan()



3.3.3 Схема алгоритма функции Enter\_float()



3.3.4 Схема алгоритма функции Enter\_int()



3.3.5 Схема алгоритма функции Formula()



3.3.6 Схема алгоритма функции CheckErrorFormula()



4 Отладка программы

Отладка программы представляет собой процесс локализации и исправления ошибок. Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить ее причину, то есть оператор или фрагмент, содержащий ошибку. Отладка программы занимает значительную часть рабочего времени программиста, нередко большую, чем составление программы. Практически любая программа перед началом отладки содержит как минимум одну ошибку.

В соответствии с этапом обработки, на котором проявляются ошибки, различают следующие виды ошибок:

1. синтаксические ошибки (ошибки, обнаруживаемые компилятором при выполнении синтаксического и частично семантического анализа);
2. ошибки компоновки (ошибки, обнаруживаемые компоновщиком при объединении модулей программы);
3. ошибки выполнения (ошибки, обнаруживаемые ОС, аппаратными средствами или пользователем при выполнении программы).

Во время отладки данной программы были допущены некоторые ошибки. К ним относятся некоторые синтаксические ошибки, ошибки компоновки, а также ошибки выполнения.

4.1 Синтаксические ошибки, допущенные в программе

Была допущена такая синтаксическая ошибка, как пропуск знака (:) после начала цикла с параметром, что является языковой особенностью Python. Так же среди синтаксических ошибок можно отметить опечатку в написании функции print(). Если вывод необходимой информации содержит кавычки (‘), то в самой функции print() необходимо использовать двойные кавычки (“). Так же была допущена такая синтаксическая ошибка, как неправильное имя оператора. Например, в строке str(ch).index('.') есть ошибка потому, что переменной ch не существует. Правильная запись этой строки – str(h).index('.').

4.2 Ошибки компоновки, допущенные в программе

Кроме того, были допущены некоторые ошибки компоновки. Например, при обращении к функции acatan() было получено следующее сообщение об ошибке:

**NameError: name 'acatan' is not defined**

Данная ошибка возникла из-за того, что модуль Input, в котором находится данная функция не был подключен к основной программе. Чтобы исправить эту ошибку, следовало подключить модуль Input, содержащий функцию acatan():

**from Input import \***

4.3 Ошибки выполнения, допущенные в программе

Так же в программе были обнаружены различные ошибки выполнения. Одной из ошибок выполнения стал неправильно организованный цикл с параметром.

Неправильная запись:

**for i in range (sizecicle+1):**

**chx = x**

**chy = y**

**fxy = eval (Form)**

Данный цикл заканчивается значением sizecicle+1, но поскольку переменная sizecicle может быть не целым числом, следовательно, это значение необходимо округлять для правильности работы цикла.

Правильная запись:

**for i in range (round(sizecicle)+1):**

**chx = x**

**chy = y**

**fxy = eval (Form)**

Также была допущена ошибка в неправильности указания условия в операторе if, из-за чего программа выдала неверные результаты.

Неправильная запись:

**if exit == 99:**

**Flag = True**

Так как переменная exit отвечает за нажатие определённой клавиши, а программа завершится при нажатии на клавишу Escape, код которой равен 27, то в условие необходимо сравнить значение переменной exit с кодом клавиши Escape.

Правильная запись:

**if exit == 27:**

**Flag = True**

В ходе отладки программы при локализации ошибок использовался метод печати в узлах, который подразумевает печать промежуточных значений переменных, интересующих программиста, в выбранных им местах (узлах) программы.

Для точной диагностики и устранения ошибок применялся метод индукции, так как необходимо было изучить причины появления ошибок, организовать данные, изучить взаимосвязь этих данных для выдвижения гипотезы с целью последующего доказательства.

Все ошибки были исправлены.

Для того, чтобы убедиться в правильности работы программы, необходимо провести тестирование.

5 Оптимизация программы

Оптимизация программы – это процесс модификации кода программы с целью улучшения ее эффективности.

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 ее 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 ее 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 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 оптимизации 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, что позволило существенно сэкономить память. Так же при вычислении дифференциального уравнения методом Эйлера полученные значения переменных сразу выводились на экран после вычисления, а в следующей итерации значения этих же переменных вычислялись заново и снова выводились на экран. Данный прием позволил сэкономить память, так как не использовались дополнительные переменные для хранения результатов вычислений. Соответствующий фрагмент кода приведен ниже:

**for i in range (k+1):**

**fxy = eval (Form)**

**dy = fxy\*h**

**TbE.add\_row ([i, "%.2f" %x, "%.2f" %y, "%.5f" %fxy, "%.5f" %dy])**

**y += dy**

**x += h**

Для оптимизации программы по времени были произведены следующие действия:

* были оптимизированы условные выражения. Например, в программе использовалось следующее условное выражение:

**if Choice == 1:**

**Euler()**

**if Choice == 2:**

**ImprovedEuler()**

В этом случае, даже если переменная Choice = 1, будет выполняться второй оператор if. При использовании конструкции с else сравнение можно прекратить, как только будет найдено истинное условие. Фрагмент оптимизированного кода:

**if Choice == 1:**

**Euler()**

**else:**

**ImprovedEuler()**

* более долгие для выполнения операции, такие как деление или возведение в степень были заменены на операцию умножения. Например:

**x = x+h/2**

В этом выражении используется более долгая для выполнения операция деления, поэтому она была заменена на умножение на 0.5, что помогло повысить эффективность программы:

**x = x+h\*0.5**

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

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

Все вышеприведенные приемы помогли существенно повысить эффективность работы программы.

6 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 3 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 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 и 6.2 продемонстрировано тестирование в нормальных условиях.

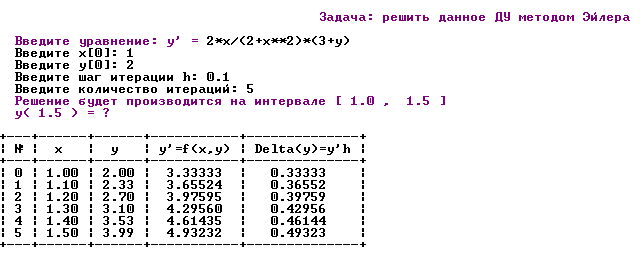


Рисунок 6.1 – Результаты вычислений с использованием метода Эйлера

Ручной просчет к результатам, отображенным на рисунке 6.1.

Дано:

.

Итерация 0:

Итерация 1:

Итерация 2:

Итерация 3:

Итерация 4:

Итерация 5:

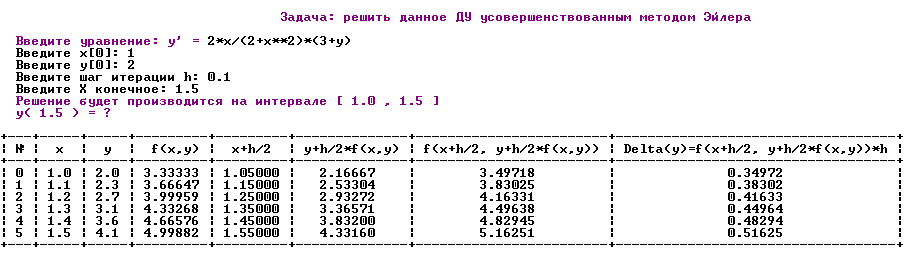


Рисунок 6.2 – Результаты вычислений с использованием усовершенствованного метода Эйлера

Ручной просчет к результатам, отображенным на рисунке 6.2.

Дано:

;

.

Итерация 0:

Итерация 1:

Итерация 2:

Итерация 3:

Итерация 4:

Итерация 5:

Тестирование программы в нормальных условиях и проверка полученных результатов с ручным просчетом показали, что программа работает правильно.

6.2 Тестирование в экстремальных условиях

При тестировании в экстремальных условиях совокупность исходных данных – это данные, лежащие на концах интервала допустимых значений. На рисунках 6.3, 6.4 и 6.5 иллюстрируется тестирование программы в экстремальных условиях с использованием начальных значений и нулевых примеров.

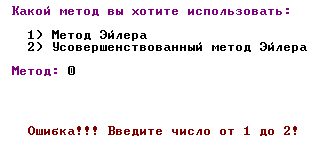


Рисунок 6.3 – Нулевые примеры и выбор первого способа решения

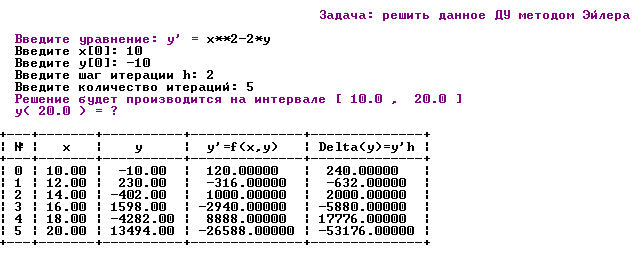


Рисунок 6.4 - Результаты вычислений с использованием метода Эйлера при экстремальных условиях

Ручной просчет к результатам, отображенным на рисунке 6.4.

Дано:

.

Итерация 0:

Итерация 1:

Итерация 2:

Итерация 3:

Итерация 4:

Итерация 5:

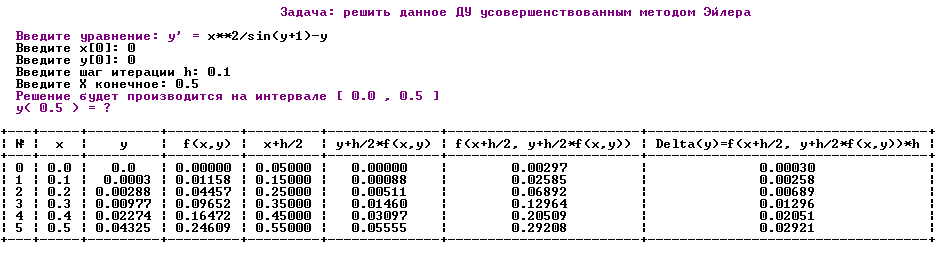


Рисунок 6.5 - Результаты вычислений с использованием усовершенствованного метода Эйлера при экстремальных условиях

Ручной просчет к результатам, отображенным на рисунке 6.5.

Дано:

;

.

Итерация 0:

;

Итерация 1:

Итерация 2:

Итерация 3:

Итерация 4:

Итерация 5:

Тестирование показало, что в экстремальных условиях результирующие значения, отображенные на рисунках 6.4 и 6.5, совпадают с эталонными значениями, вычисленными при помощи ручного просчета. Исходя из этого, можно сделать вывод, что программа в экстремальных условиях работает нормально.

6.3 Тестирование в исключительных ситуациях

Тестирование в исключительных ситуациях предполагает формирование набора данных, значения которых лежат за пределами допустимой области изменения. Программа, тестируемая в исключительных ситуациях, должна отвергать любые значения, которые она не в состоянии обрабатывать правильно.

Рисунки 6.6, 6.7, 6.8 показывают поведение программы при вводе данных, выходящих за пределы области изменения, а также при вводе некорректных данных. Видно, что программа отбрасывает бессмысленные данные и предлагает ввести данные повторно.

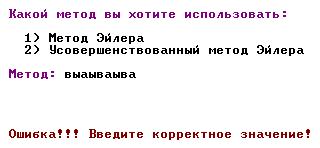


Рисунок 6.6 – Попытка ввода некорректных данных при выборе метода решения ДУ

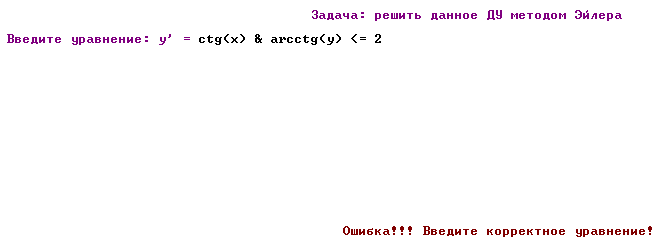


Рисунок 6.7 – Попытка ввода некорректного уравнения

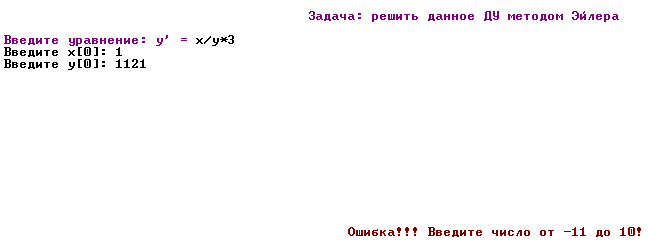


Рисунок 6.8 – Попытка ввода некорректных начальных значений переменных Х и У

Таким образом, просмотр результатов тестирования в исключительных ситуациях показал, что программа устойчива к бессмысленным и неправильным данным. Она отвергает такие данные и корректно продолжает свою работу.

Вывод: Тестирование показало, что программа прошла тестирование в нормальных, экстремальных условиях, а также в исключительных ситуациях, следовательно, она работает правильно.

Листинг программы приведен в приложении А, а результаты выполнения – в приложении Б.

7 Руководство пользователя

Системные требования, предъявляемые к программе:

* процессор Intel Core i7 – 950;
* количество ОЗУ от 4 гб;
* место на жестком диске от 15 мб;
* любая встроенная или дискретная видеокарта.

Для работы с программой следуйте пошаговому руководству, представленному ниже:

1. запустите файл Kursovik.py, находящийся в папке \Desktop\Решение ДУ (рисунок 7.1);

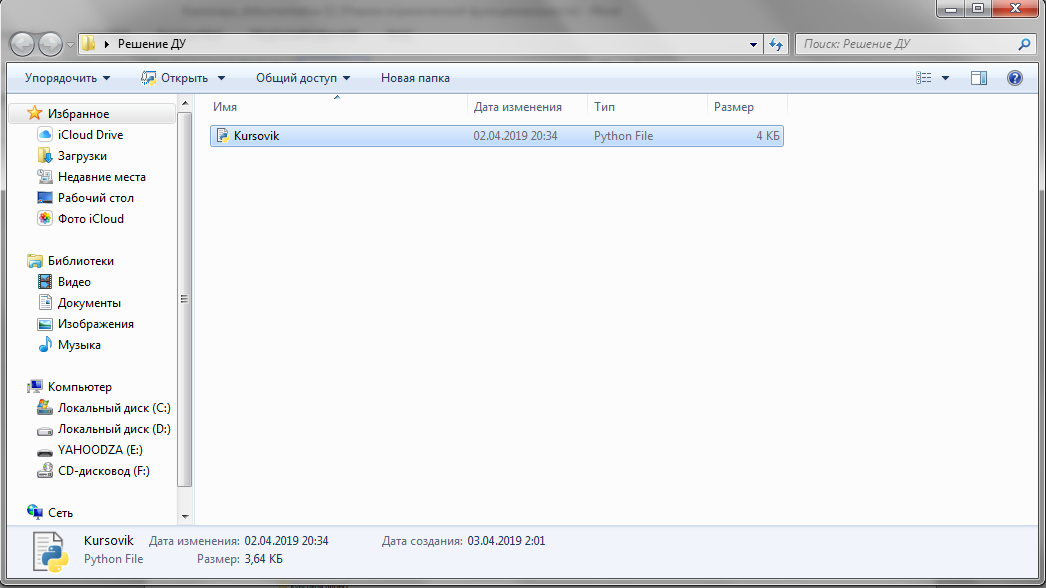


Рисунок 7.1 – Запуск программы

1. после запуска программы откроется консоль, где пользователю будет предложено выбрать метод решения дифференциального уравнения (рисунок 7.2);

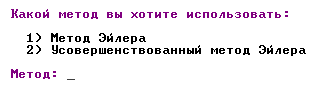


Рисунок 7.2 – Выбор метода решения ДУ

1. после выбора метода решения необходимо ввести дифференциальное уравнение (рисунок 7.3). Поддерживаемые функции:

* cos(x) – косинус от х;
* sin(x) – синус от х;
* tan(x) – тангенс от х;
* catan(x) – котангенс от х;
* asin(x) – арксинус от х;
* acos(x) – арккосинус от х;
* atan(x) – арктангенс от х;
* acatan – арккотангенс от х;
* log(x, a) – логарифм х по основанию а;
* x\*\*k – х в степени k.



Рисунок 7.3 – Ввод уравнения

1. следующим шагом необходимо ввести следующие значения: , , шаг, количество итераций (если метод Эйлера), х конечное (если усовершенствованный метод Эйлера) (рисунок 7.4);



Рисунок 7.4 – Ввод начальных условий

1. при соблюдении всех шагов инструкции программа должна успешно выполнить вычисления. После нажатия любой клавиши программа выведет на экран таблицу со всеми проведенными операциями по решению введённого ДУ выбранным методом (рисунок 7.5);

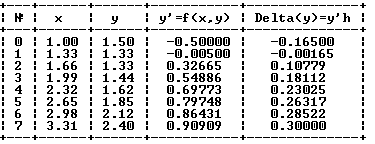


Рисунок 7.5 – Таблица результатов

1. после выполнения программы пользователю будет предложено работать с программой дальше, либо закончить работу. Чтобы выйти из программы, требуется нажать клавишу Esc, чтобы продолжить работу – клавишу Enter.

ЗАКЛЮЧЕНИЕ

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

В ходе написание курсового проекта были получены практические навыки в области проектирования программы. Программа разрабатывалась с использованием стратегии анализа сообщений, реализующей метод нисходящий 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 ее проектированию перед началом написания кода.

Так же были получены практические навыки при оптимизации и тестировании программы. Появилось понимание важности и сложности отладки, ее 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 усовершенствованным метод Эйлера.

Программа имеет надлежащую структуру, схемы алгоритма и необходимые комментарии.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бизли Дэвид. Python. Подробный справочник. «Символ», 2010
2. Вержбицкий В.М. [Численные методы. Математический анализ и обыкновенные дифференциальные уравнения](https://www.twirpx.com/file/12609/). «Высшая школа», 2001
3. Иртегов Д.В. Введение в операционные системы. «БХВ-Петербург», 2008
4. Киркинский А.С. Математический анализ. «Академический проект», 2006
5. Лутц Марк. Программирование на Python. «Символ-плюс», 2011
6. Любанович Билл. Простой Python. Современный стиль программирования. «Питер», 2016
7. Макконелл Стив. Совершенный код. «БХВ-Петербург», 2017
8. Самарский А.А. Введение в численным методы. «Курс лекций вычислительной математики МГУ»
9. Спиридонов Э.С. Операционные системы. «Книжный дом ЛИБРОКОМ», 2010
10. Танебаум Эндрю, Бос Х. Современные операционные системы. «Питер», 2017
11. <https://ru.wikipedia.org/wiki/Метод_Эйлера>
12. <https://helpiks.org/2-60379.html>
13. http://cyclowiki.org/wiki/Усовершенствованный\_метод\_Эйлера
14. <https://studme.org/180940/matematika_himiya_fizik/metod_eylera_metod_eylera_pereschetom>
15. <https://habr.com/ru/post/29778/>

ПРИЛОЖЕНИЕ А

(обязательное)

Листинги программы

А.1 Листинг основной программы

'''

Курсовой проект по предмету МДК 03.01 "Технология разработки программного

обеспечения", по теме: "Разработка программы решения дифференциальных

уравнений"

Название: Course\_project.

Разработал: Королёв Петр Олегович, группа ТМП-61.

Дата и номер версии: 01.04.2019 v3.0.

Язык: Python.

Краткое описание программы: Данная программа решает дифференциальные

уравнения.

Задание: Разработка программы решения дифференциальных уравнений:

1) методом Эйлера;

2) усовершенствованным методом Эйлера.

Входными данными являются x[0], y[0], h - шаг, количество итераций (при методе

Эйлера), X конечное (при усовершенствованном методе Эйлера).

Функция, вызываемая из модуля Input:

Enter\_int() - функция ввода целочисленных значений.

Функции, вызываемые из модуля Methods:

Euler() - функция, вычисляющая ДУ методом Эйлера;

ImprovedEuler() - функция, вычисляющая ДУ усовершенствованным методом Эйлера.

Краткое описание используемых переменных в основной программе:

Flag - флаг для выхода из цикла;

exit - переменная, отвечающая за выход из программы по нажатию клавиши;

Сhoice - переменная для выбора номера метода решения ДУ.

'''

import os

from msvcrt import getch

from Methods\_DE import \*

os.system("mode con cols=125 lines=40")

os.system('color f0')

Flag = False

while Flag == False: #Цикл работы программы

os.system ("cls") #пока не нажата клавиша Esc

ShowCursor(True)

exit = None

gotoxy(43, 10)

SetColor(5, 15)

print ('Какой метод вы хотите использовать:')

SetColor(0, 15)

gotoxy(45, 12)

print ('1) Метод Эйлера')

gotoxy(45, 13)

print ('2) Усовершенствованный метод Эйлера')

gotoxy(43, 15)

SetColor(5, 15)

print ('Метод:')

SetColor(0, 15)

Сhoice = Enter\_int (1, 2, '', 50, 15) #Условие выбора метода решения ДУ

if Сhoice == 1:

os.system ("cls")

Euler()

gotoxy(1, 30)

print ('\n')

ShowCursor(False)

os.system ("pause")

else:

os.system ("cls")

ImprovedEuler()

gotoxy(1, 30)

print ('\n')

ShowCursor(False)

os.system ("pause")

os.system ("cls")

ShowCursor(False)

gotoxy(44, 10)

SetColor(5, 15)

print ('Нажмите клавишу на вашей клавиатуре')

SetColor(0, 15)

gotoxy(29, 17)

SetColor(2, 15)

print ('Enter')

SetColor(0, 15)

gotoxy(13, 18)

print ('- для продолжения работы программы')

gotoxy(91, 17)

SetColor(4, 15)

print ('Escape')

SetColor(0, 15)

gotoxy(80, 18)

print ('- для выхода из программы')

while exit != 13 and exit != 27: #Цикл определения нажатой клавиши

exit = ord(getch())

if exit == 27: #Условие выхода из главного цикла

Flag = True #while

А.2 Листинг модуля Methods\_DE

'''

Вспомогательный модуль основной программы.

Название: Methods\_DE.

Разработал: Королёв Петр Олегович, группа ТМП-61.

Дата и номер версии: 02.04.2019 v4.0.

Язык: Python.

Используемые в модуле функции:

Euler() - функция, вычисляющая ДУ методом Эйлера;

ImprovedEuler() - функция, вычисляющая ДУ усовершенствованным методом Эйлера.

'''

from prettytable import PrettyTable as pt

from time import sleep

from Input import \*

from math import cos, sin, tan, log, pi, e, acos, asin, atan

'''

Euler() - функция, вычисляющая ДУ методом Эйлера.

Локальные переменные:

TbE - результирующая таблица;

TbE.field\_names - название столбцов результирующей таблицы;

Form - вводимое уравнение;

x - значение х[0];

y - значение y[0];

h - значение шага итерации;

k - количество итераций;

i - счетчик для цикла;

fxy - значение y';

dy - значение дельта y.

'''

def Euler ():

TbE = pt()

TbE.field\_names = ["№", "x", "y", "y'=f(x,y)", "Delta(y)=y'h"]

gotoxy(40, 2)

SetColor(5, 15)

print ('Задача: решить данное ДУ методом Эйлера')

SetColor(0, 15)

Form = Formula()

x, y = CheckErrorFormula (Form)

gotoxy(2, 7)

print ('Введите шаг итерации h:')

h = Enter\_float (0, 2, '', 26, 7)

gotoxy(2, 8)

print ('Введите количество итераций:')

k = Enter\_int (1, 10, '', 31, 8)

gotoxy(2, 9)

SetColor(5, 15)

print ('Решение будет производится на интервале [',x,', ',h\*k+x,']')

gotoxy(2, 10)

print ('y(',h\*k+x,') = ?\n')

SetColor(0, 15)

for i in range (k+1): #Цикл вычисления ДУ методом Эйлера

fxy = eval (Form)

dy = fxy\*h

TbE.add\_row ([i, "%.2f" %x, "%.2f" %y, "%.5f" %fxy, "%.5f" %dy])

y += dy

x += h

print (TbE)

'''

ImprovedEuler() - функция, вычисляющая ДУ усовершенствованным методом Эйлера.

Локальные переменные:

TbImE - результирующая таблица;

TbImE.field\_names - название столбцов результирующей таблицы;

Form - вводимое уравнение;

x - значение х[0];

y - значение y[0];

h - значение шага итерации;

t - количество знаков шага итерации после запятой;

Flag - флаг для выхода из цикла;

k - конечное значение х;

sizecicle - количество итераций на отрезке [x, k];

i - счетчик для цикла;

chx - переменная, хранящая значение переменной x;

chy - переменная, хранящая значение переменной y;

fxy - значение y';

xh - переменная, хранящая значение переменной x=x+h/2;

yh - переменная, хранящая значение переменной y=y+h/2\*f(x,y);

fxhyh - значение y' при x=x+h/2 и y=y+h/2\*f(x,y);

dy - значение дельта y при x=x+h/2 и y=y+h/2\*f(x,y).

'''

def ImprovedEuler ():

TbImE = pt()

TbImE.field\_names = ["№", "x", "y", "f(x,y)", "x+h/2", "y+h/2\*f(x,y)", "f(x+h/2, y+h/2\*f(x,y))", "Delta(y)=f(x+h/2, y+h/2\*f(x,y))\*h"]

gotoxy(40, 2)

SetColor(5, 15)

print ('Задача: решить данное ДУ усовершенствованным методом Эйлера')

SetColor(0, 15)

Form = Formula()

x, y = CheckErrorFormula (Form)

gotoxy(2, 7)

print ('Введите шаг итерации h:')

h = Enter\_float (0, 2, '', 26, 7)

t = len(str(h)) - str(h).index('.') - 1

Flag = False

gotoxy(2, 8)

print ('Введите X конечное:')

while Flag == False: #Цикл проверки правильности вода x

ShowCursor(True) #конечного и расчёта количества итераций

k = Enter\_float (x+h-0.0001, 20, '', 22, 8)

sizecicle = (k - x) / h

if round(k - h \* round(sizecicle), t) == x: #Условие проверки правильности вода x

Flag = True #конечного

else:

ShowCursor(False)

gotoxy(22, 8)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(33, 20)

SetColor(4, 15)

print ('Ошибка!!! Введите X конечное, соответствующее шагу итерации!')

sleep(4)

SetColor(0, 15)

gotoxy(33, 20)

print ('\t\t\t\t\t\t\t\t\t')

gotoxy(2, 9)

SetColor(5, 15)

print ('Решение будет производится на интервале [',x,',',k,']')

gotoxy(2, 10)

print ('y(',k,') = ?\n')

SetColor(0, 15)

for i in range (round(sizecicle)+1): #Цикл вычисления ДУ

chx = x #усовершенствованным методом Эйлера

chy = y

fxy = eval (Form)

x = x+h\*0.5

y = y+h\*0.5\*fxy

xh = x

yh = y

fxhyh = eval (Form)

x = chx

y = chy

dy = fxhyh\*h

TbImE.add\_row ([i, round(x, t), round(y, t),"%.5f" %fxy, "%.5f" %xh, "%.5f" %yh,

"%.5f" %fxhyh, "%.5f" %dy])

y += dy

x += h

print (TbImE)

А.3 Листинг модуля Input

'''

Вспомогательный модуль основной программы.

Название: Input.

Разработал: Королёв Петр Олегович, группа ТМП-61.

Дата и номер версии: 01.04.2019 v3.0.

Язык: Python.

Используемые в модуле функции:

catan() - функция котангенса;

acatan() - функция арккотангенса;

Enter\_float() - функция ввода вещественных значений;

Enter\_int() - функция ввода целочисленных значений;

Formula() - функция ввода ДУ;

CheckErrorFormula() - функция проверки начальных значений x[0] и y[0]

на возможность участия в введенном уравнении.

'''

from time import sleep

from ctypes import \*

from math import cos, sin, tan, log, pi, e, acos, asin, atan

class COORD(Structure):

#pass

\_fields\_ = [("X", c\_short), ("Y", c\_short)]

#COORD.\_fields\_ = [("X", c\_short), ("Y", c\_short)]

class CONSOLE\_CURSOR\_INFO(Structure):

#pass

\_fields\_ = [('dwSize', c\_int), ('bVisible', c\_bool)]

#CONSOLE\_CURSOR\_INFO.\_fields\_ = [('dwSize', c\_int), ('bVisible', c\_byte)]

def gotoxy(c, r):

h = windll.kernel32.GetStdHandle(-11) #STD\_OUTPUT\_HANDLE = -11

windll.kernel32.SetConsoleCursorPosition(h, COORD(c, r))

def SetColor(text, background):

h = windll.kernel32.GetStdHandle(-11) #STD\_OUTPUT\_HANDLE = -11

windll.kernel32.SetConsoleTextAttribute(h, ((background << 4) | text))

def ShowCursor(Visible):

h = windll.kernel32.GetStdHandle(-11) #STD\_OUTPUT\_HANDLE = -11

CursorInfo = CONSOLE\_CURSOR\_INFO()

CursorInfo.dwSize = 1

CursorInfo.bVisible = Visible

windll.kernel32.SetConsoleCursorInfo(h, byref(CursorInfo))

'''

catan() - функция котангенса.

Формальный параметр:

ax - переменная, передающая значение аргумента функции (x).

'''

def catan (ax):

return 1/tan(ax)

'''

acatan() - функция арккотангенса.

Формальный параметр:

ax - переменная, передающая значение аргумента функции (x).

Локальная переменная:

arcctg - переменная для вычисления арккотангенса.

'''

def acatan (ax):

if ax == 0: #Условие проверки при

arcctg = 0 #делении на 0

else:

arcctg = atan(1/ax)

return arcctg

'''

Enter\_float() - функция ввода вещественных значений.

Формальные параметры:

m - левая граница промежутка;

n - правая граница промежутка;

c - переменная для форматированного вывода;

x - переменная для вывода по оси Ox;

y - переменная для вывода по оси Oy.

Локальная переменная:

size - введённое значение.

'''

def Enter\_float (m, n, c, x, y):

while True: #Цикл проверки правильности

try: #ввода вещественных значений

ShowCursor(True)

gotoxy(x, y)

size = float (input("%s" %c))

except ValueError:

ShowCursor(False)

gotoxy(x-1, y)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(43, 20)

SetColor(4, 15)

print ('Ошибка!!! Введите корректное значение!')

sleep(4)

SetColor(0, 15)

gotoxy(43, 20)

print ('\t\t\t\t\t\t\t\t\t')

else:

if m < size <= n:

return size

else:

ShowCursor(False)

gotoxy(x-1, y)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(45, 20)

SetColor(4, 15)

print ('Ошибка!!! Введите число от %d до %d!' %(m, n))

sleep(4)

SetColor(0, 15)

gotoxy(45, 20)

print ('\t\t\t\t\t\t\t\t\t')

'''

Enter\_int() - функция ввода целочисленных значений.

Формальные параметры:

m - левая граница промежутка;

n - правая граница промежутка;

c - переменная для форматированного вывода;

x - переменная для вывода по оси Ox;

y - переменная для вывода по оси Oy.

Локальная переменная:

size - введённое значение.

'''

def Enter\_int (m, n, c, x, y):

while True: #Цикл проверки правильности

try: #ввода целочисленных значений

ShowCursor(True)

gotoxy(x, y)

size = int (input("%s" %c))

except ValueError:

ShowCursor(False)

gotoxy(x-1, y)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(43, 20)

SetColor(4, 15)

print ('Ошибка!!! Введите корректное значение!')

sleep(4)

SetColor(0, 15)

gotoxy(43, 20)

print ('\t\t\t\t\t\t\t\t\t')

else:

if m <= size <= n:

return size

else:

ShowCursor(False)

gotoxy(x-1, y)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(45, 20)

SetColor(4, 15)

print ('Ошибка!!! Введите число от %d до %d!' %(m, n))

sleep(4)

SetColor(0, 15)

gotoxy(45, 20)

print ('\t\t\t\t\t\t\t\t\t')

'''

Formula() - функция ввода ДУ.

Локальные переменные:

x - переменная для проверка на правильность введенного

уравнения;

y - переменная для проверка на правильность введенного

уравнения;

f - введённое уравнение.

'''

def Formula ():

x = 1

y = 1

while True: #Цикл проверки правильности

try: #ввода ДУ

ShowCursor(True)

gotoxy(2, 4)

SetColor(5, 15)

print ("Введите уравнение: y' = ")

SetColor(0, 15)

gotoxy(26, 4)

f = input()

f = f.lower()

eval (f)

except (NameError, SyntaxError, ValueError, ZeroDivisionError):

ShowCursor(False)

gotoxy(25, 4)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(44, 20)

SetColor(4, 15)

print ('Ошибка!!! Введите корректное уравнение!')

sleep(4)

SetColor(0, 15)

gotoxy(44, 20)

print ('\t\t\t\t\t\t\t\t\t')

else:

if ('%' in f or '&' in f or '=' in f or '<' in f or '>' in f or '^' in f or '>=' in f or '<=' in f or '~' in f):

ShowCursor(False)

gotoxy(25, 4)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(44, 20)

SetColor(4, 15)

print ('Ошибка!!! Введите корректное уравнение!')

sleep(4)

SetColor(0, 15)

gotoxy(44, 20)

print ('\t\t\t\t\t\t\t\t\t')

else:

if 'x' in f and 'y' in f:

return f

else:

ShowCursor(False)

gotoxy(25, 4)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(44, 20)

SetColor(4, 15)

print ('Ошибка!!! Введите корректное уравнение!')

sleep(4)

SetColor(0, 15)

gotoxy(44, 20)

print ('\t\t\t\t\t\t\t\t\t')

'''

CheckErrorFormula() - функция проверки начальных значений x[0] и y[0]

на возможность участия в введенном уравнении.

Формальный параметр:

f - введённое уравнение.

Локальные переменные:

x - значение x[0];

y - значение y[0].

'''

def CheckErrorFormula (f):

while True: #Цикл проверки корректности

try: #ввода значений x[0] и y[0]

ShowCursor(True)

gotoxy(2, 5)

print ('Введите x[0]:')

x = Enter\_float (-11, 10, '', 16, 5)

gotoxy(2, 6)

print ('Введите y[0]:')

y = Enter\_float (-11, 10, '', 16, 6)

eval(f)

except (ZeroDivisionError, ValueError):

ShowCursor(False)

gotoxy(16, 5)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(16, 6)

print ('\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t')

gotoxy(43, 20)

SetColor(4, 15)

print ('Ошибка!!! Введите корректные данные!')

sleep(4)

SetColor(0, 15)

gotoxy(43, 20)

print ('\t\t\t\t\t\t\t\t\t')

else:

return x, y

ПРИЛОЖЕНИЕ Б

(обязательное)

Результаты выполнения программы

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

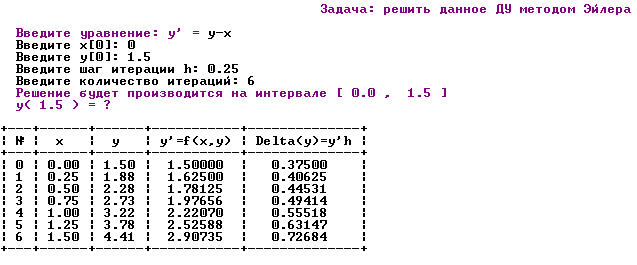


Рисунок Б.1 – Решение ДУ y’ = y - x методом Эйлера

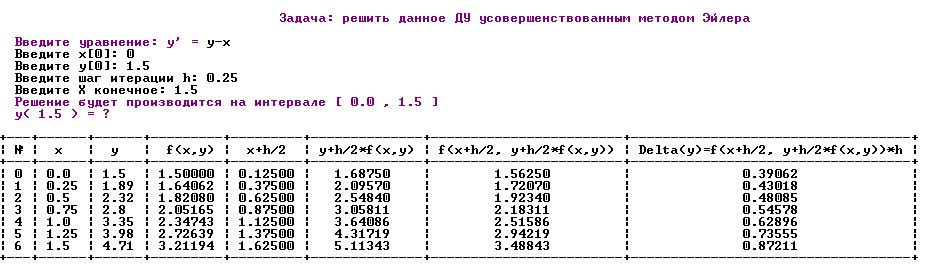


Рисунок Б.2 – Решение ДУ y’ = y – x усовершенствованным методом Эйлера

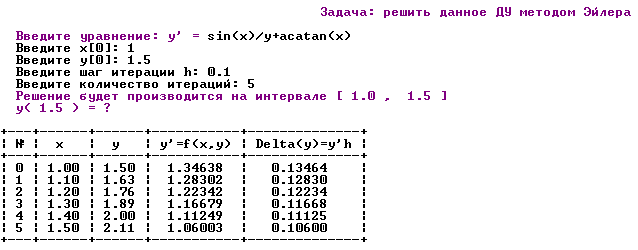


Рисунок Б.3 – Решение ДУ y’ = sin(x)/y + acatan(x) методом Эйлера

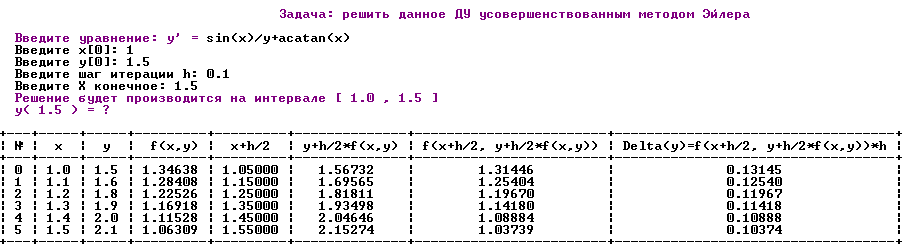


Рисунок Б.4 – Решение ДУ y’ = sin(x)/y + acatan(x) усовершенствованным методом Эйлера

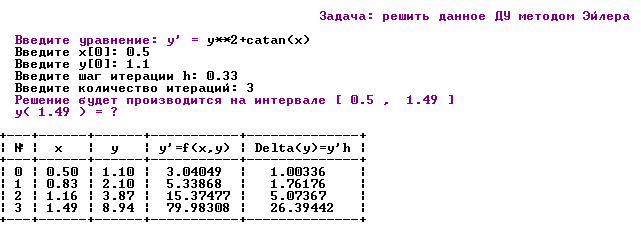


Рисунок Б.5 – Решение ДУ y’ = y^2 + catan(x) методом Эйлера

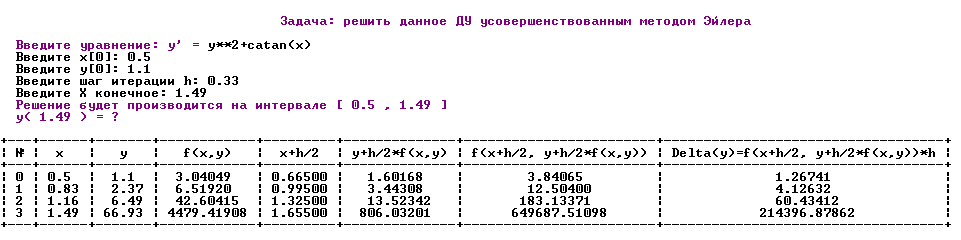


Рисунок Б.6 – Решение ДУ y’ = y^2 + catan(x) усовершенствованным методом Эйлера