**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное государственное автономное образовательное**

**учреждение высшего образования**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**(РУТ(МИИТ))**

Кафедра «Вычислительные системы, сети и информационная безопасность»

**ОТЧЕТ ПО ДИСЦИПЛИНЕ**

**«Проектная деятельность»**

**«РАЗРАБОТКА WINDOWS ПРИЛОЖЕНИЯ C GUI ДЛЯ РЕШЕНИЯ ИРРАЦИОНАЛЬНЫХ УРАВНЕНИЙ»**

*Направление:* 10.03.01 Информационная безопасность

*Профиль:* ***Безопасность компьютерных систем***

Выполнили: студенты группы УИБ-112

Бакеев Руслан Эльдусович

Орлов Андрей Витальевич

Проверил:

Старший преподаватель Цыганова Наталия Алексеевна

Москва 2021 г.

Оглавление

[1 Введение 6](#_Toc93591432)

[2 Общее задание 8](#_Toc93591433)

[3 Решение уравнения в Excel 9](#_Toc93591434)

[4 Решение уравнения с помощью метода прямого перебора 11](#_Toc93591435)

[4.1 Общее теоретическое описание метода 11](#_Toc93591436)

[4.2 Таблица имён 11](#_Toc93591437)

[4.3 Блок-схема 12](#_Toc93591438)

[4.3.1 Отладка блок-схемы 13](#_Toc93591439)

[4.4 Код вычислительной процедуры 17](#_Toc93591440)

[4.5 Результат / скриншот 17](#_Toc93591441)

[4.6 Промежуточный вывод 18](#_Toc93591442)

[5 Решение уравнения с помощью Метода итераций 19](#_Toc93591443)

[5.1 Общее теоретическое описание метода 19](#_Toc93591444)

[5.2 Таблица имён 20](#_Toc93591445)

[5.3 Блок-схема 20](#_Toc93591446)

[5.3.1 Отладка блок-схемы 21](#_Toc93591447)

[5.4 Код вычислительной процедуры: 22](#_Toc93591448)

[5.5 Результат / скриншот 22](#_Toc93591449)

[5.6 Промежуточный вывод 23](#_Toc93591450)

[6 Решение уравнения двумя методами 24](#_Toc93591451)

[6.1 Общее теоретическое описание метода 24](#_Toc93591452)

[6.2 Код вычислительной процедуры 24](#_Toc93591453)

[6.3 Результат / скриншот 26](#_Toc93591454)

[6.4 Промежуточный вывод 26](#_Toc93591455)

[7 Проектирование графического интерфейса 27](#_Toc93591456)

[7.1 Comeback() 27](#_Toc93591457)

[7.1.1 Код функции 27](#_Toc93591458)

[7.1.2 Скриншот работы программы 28](#_Toc93591459)

[7.2 Clicked() 28](#_Toc93591460)

[7.2.1 Код функции 29](#_Toc93591461)

[7.2.2 Скриншот работы программы 29](#_Toc93591462)

[7.3 Iterations() 29](#_Toc93591463)

[7.3.1 Код функции 30](#_Toc93591464)

[7.3.2 Скриншоты работы программы 32](#_Toc93591465)

[7.4 Brute() 32](#_Toc93591466)

[7.4.1 Код функции 33](#_Toc93591467)

[7.4.2 Скриншоты работы программы 34](#_Toc93591468)

[7.5 Special() 35](#_Toc93591469)

[7.5.1 Код функции 36](#_Toc93591470)

[7.5.2 Скриншоты работы программы 36](#_Toc93591471)

[8 Оценка времени решения 37](#_Toc93591472)

[8.1 Замер времени выполнения программы оптимизированного прямого перебора. 37](#_Toc93591473)

[8.2 Замер времени выполнения программы методом итераций. 38](#_Toc93591474)

[9 Вывод: 39](#_Toc93591475)

[10 Приложение A. Код программы 41](#_Toc93591476)

# 1 Введение

Задача состоит в решении уравнения несколькими способами: модуль «Поиск решения» в Excel’e, метод итераций, метод прямого перебора значений и одновременно 2 метода – итераций и перебора (доп. задание). Ниже идут записи начала работы.

Задача состоит в решении уравнения несколькими способами: модуль «Поиск решения» в Excel’e, метод итераций, метод прямого перебора значений и одновременно 2 метода – итераций и перебора (доп. задание). Ниже идут записи начала работы.  
Методы решения задачи последующие: понять, имеет ли уравнение корни с помощью производной, затем исследовать с помощью графика Excel, далее идёт нахождение корней исходного уравнения с помощью программного кода, заключающего пару итерационных методов, далее сделать простой пользовательский интерфейс для удобного вычисления корней уравнения.

Начали искать решение уравнения вручную, по итогу сошлись на интересных результатах – изначальное уравнение не имеет корней.  
Изначально уравнение имело вид exp(-x) + x + 2 = 0. Обозначили уравнение за функцию, ищем её минимальное значение с помощью производной: exp(-x)’ = -1\*exp(-x), x’ = 1, 2’ = 0.  
Уравнение производной по итогу приняло вид -1\*exp(-x) + 1 = 0; -exp(-x) = -1, exp(-x) = 1; exp(-x) = exp(0); -x = 0; x = 0.  
Подставляя найденную точку экстремума, определяем знаки производной, выясняем, что это точка минимума. Подставим её в изначальную функцию, получается exp(-0) + 0 + 2 -> 1+2=3.  
Получается, что минимальное значение функции оказалось < 0, из этого следует, что в рациональном мире чисел решения данного уравнения не существует, остаётся искать его в комплексном.  
Так как нахождение корней в комплексном виде выходит за рамки установленного задания, оно было заменено уравнением e^(-x) – x + 2.  
Исследование уравнения в Excel'e показало наличие корней у этого уравнения – специально рисовали график на всём данном нам отрезке с шагом 0.1.

Далее работали над методом прямого перебора – это простейший алгоритм для поиска корней. При небольшой точности он сравнительно быстр, но с всё меньшей точностью растёт время выполнения программы. Реализовали следующим образом: пользователь вводит шаг алгоритма (мы назвали это точностью вычисления – чем меньше шаг, тем больше точность), далее алгоритм идёт от левого края данного отрезка к правому с указанным шагом, вычисляя значение функции каждый раз – если значение функции по модулю оказалось меньше, чем до этого найденное наименьшее значение функции, следовательно такой х ближе к ответу, а значит, перезаписываем НЗФ и новый ответ в соответствующие переменные.

Метод итераций (последовательных приближений) реализован, исходя из выражения x = f(x). Сначала кидаем в алгоритм любое число, после этого вычисляем значение преобразованной функции (выразили чистый х в левой части, и оставили некоторое выражение с х в правой части, в нашем случае это х = е^(-x) = 2) от этого числа, подставляем найденное значение вместо x в правой части, вычисляем, полученное значение подставляем вместо х в правой части – и так это будет продолжаться столько раз, сколько пожелает пользователь – именно потому точность у нас здесь представлена числом натуральным – не может такого быть, чтобы пользователь решил пройти 1,5 круга таких вычислений, 0 в расчёт не берём, так как при нём метод итераций попросту не работает.

Графику реализовали через модуль tkinter в Python из-за удобства его использования – также по нему были доступны хорошие курсы на учебной платформе Stepik, так что учиться работать с ним было намного проще. Кроме того, в питоне очень удобно реализованы f – строки, что позволяют вставлять переменную абсолютно в любое место их текста – таким образом, были очень удобно оформлены вызовы основных вычислительных процедур.

Замер времени проводили через модуль time() в питоне самым простым способом – замеряли время с начала работы программы, затем замеряли в конце, и вычисляли модуль разности таких значений по модулю.

# Общее задание

Задано иррациональное уравнение e^(-x) – x + 2 = 0 на диапазоне [-1.5;5].

1. Найти решение уравнения на указанном диапазоне с заданной точностью, используя модуль «Поиск решения» пакета Microsoft Excel.
2. Найти решение уравнения на указанном диапазоне с заданной точностью, используя метод прямого перебора значений в пределах диапазона.
3. Найти решение уравнения на указанном диапазоне с заданной точностью, используя численный метод – Метод итераций (последовательных приближений).
4. Программа должна быть написана с применением элементов графического интерфейса пользователя (окна, кнопки, поля, редактирования, метки и т. п.)
5. Определить время вычисления по процедурам (2) и (3), сравнить результаты и пояснить их(опционально).

# 3 Решение уравнения в Excel

Задание 1: найти решение уравнения на указанном диапазоне с заданной точностью, используя модуль «Поиск решения» пакета Microsoft Excel.

Описание: найдено решение уравнения на указанном диапазоне с заданной точностью, использовав модуль «Поиск решения». С помощью точности, которая была равна 0.1, постепенно перебирался диапазон значений, подставляя эти значения вместо «x». Также был начерчен график для того, чтобы посмотреть, как зависит функция от самих значений (если при отрицательном значении – функция положительна, то при положительном значении – функция стремится к минус бесконечности)

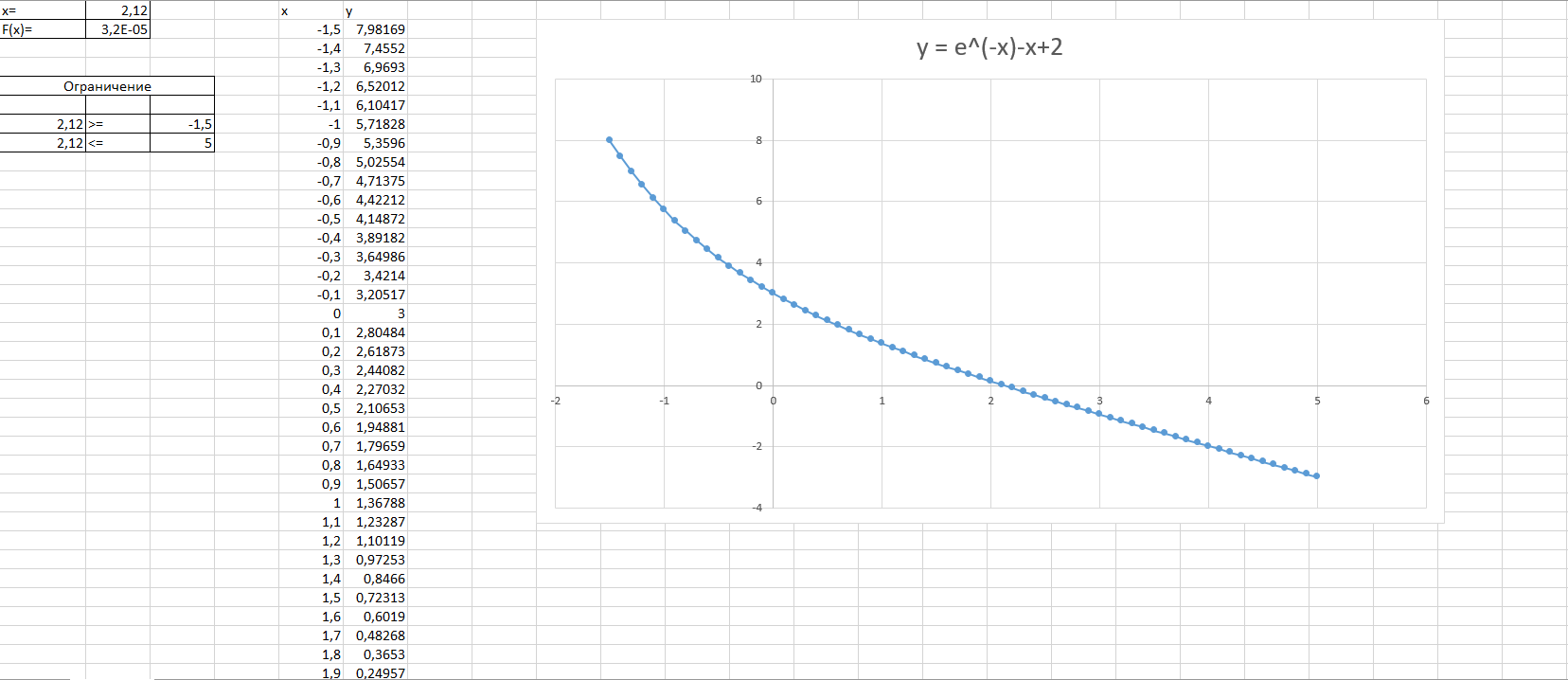




Рисунок 1 - Общий вид решения Excel

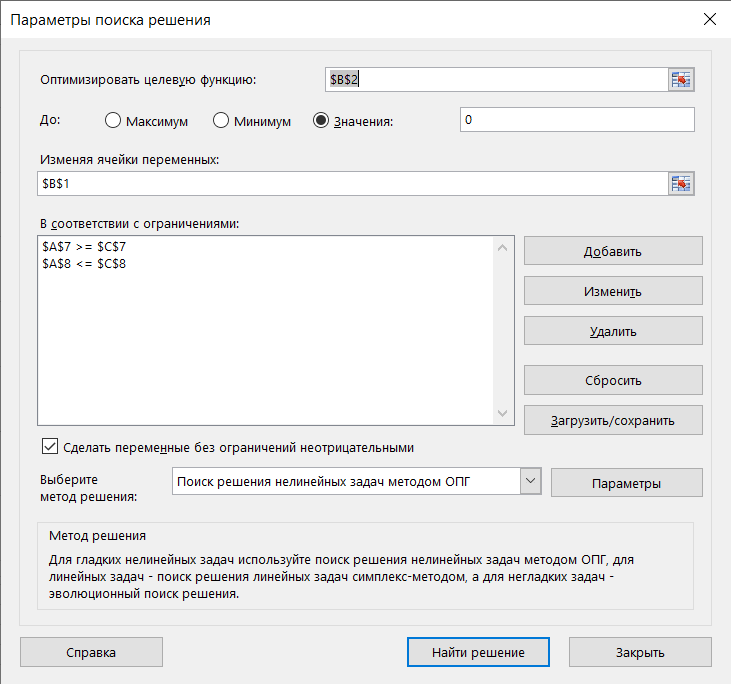


Рисунок 2 – вид модуля «Поиск решения»

Вывод: Нашли решение уравнения, изначально построив 2 таблицы: таблицу ограничений и таблицу с нахождение функции F(x). Далее изучив модуль «Поиск решения», построили график зависимости значения y от значения переменной x, которое изменялось на всём промежутке: чем меньше значения x – тем больше значение y, и наоборот.

# Решение уравнения с помощью метода прямого перебора

Задание 2: найти решение уравнения на указанном диапазоне с заданной точностью, используя метод прямого перебора значений в пределах диапазона

# 4.1 Общее теоретическое описание метода

Данный алгоритм является самым простым методом нахождения корней линейного уравнения на заданном интервале. Замысел в последовательном вычислении функции от каждого значения на отрезке, перебираем их с помощью исходного заданного в программу шага, начиная от левого края отрезка, заканчивая правым. Чем меньше по модулю будет значение функции от очередного числа, тем ближе оно к ответу – тогда нам следует перезаписать как наименьшее найденное значение функции, так и найденное значение на отрезке – оно пойдёт в ответ. Таким образом, постепенно обновляя указанные выше переменные, приходим к ответу.

# 4.2 Таблица имён

|  |  |  |
| --- | --- | --- |
| Исходные данные | | |
| n | Вещественный | Переменная точности |
| Рабочие переменные | | |
| temp | Целочисленный | Переменная, содержащая уравнение |
| a | Целочисленный | Переменная-помощник правой границы ответа |
| b | Целочисленный | Переменная-помощник левой границы ответа |
| Результат | | |
| answer | Вещественный | Ответ уравнения |

Таблица 1 – Таблица имён

# 4.3 Блок-схема

Рисунок 3 – блок-схема

# 4.3.1 Отладка блок-схемы

1. Начало

2. Ввод с клавиатуры n = 0.01

3. Объявление переменных: a = -1.5

4. Объявление переменных: b = equation(n). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(-0.01) – 0.01 + 2 = 2.980049833749168. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 4.

4. b = 2.980049833749168

5. Объявление переменных: answer = -1

6. -1.5 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(1.5) + 1.5 + 2 = 7.981689070338064. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = 7.981689070338064

8. abs(temp) < b, 7.981689070338064 < 2.980049833749168, нет, идём к блоку 11

11. a = -1.49. Возврат к блоку 6.

6. -1.49 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(1.49) + 1.49 + 2 = 7.927095519003664. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = 7.927095519003664

8. abs(temp) < b, 7.927095519003664 < 2.980049833749168, нет, идём к блоку 11

11. a = -1.48. Возврат к блоку 6.

6. -1.48 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(1.48) + 1.48 + 2 = 7.872945680918756. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = 7.872945680918756

8. abs(temp) < b, 7.872945680918756 < 2.980049833749168, нет, идём к блоку 11

11. a = -1.47. Возврат к блоку 6.

Пройдя программа так множество раз, a = 0.02

6. 0.02 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(-0.02) – 0.02 + 2 = 2.960198673306755. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = 2.960198673306755

8. abs(temp) < b, 2.960198673306755 < 2.980049833749168, да, идём к блоку 9

9. Переприсваивание переменной b = abs(temp) = 2.960198673306755

10. Переприсваивание переменной answer = a = 0.02

11. a = 0.03. Возврат к блоку 6

6. 0.03 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(-0.03) – 0.03 + 2 = 2.9404455335485081. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = 2.9404455335485081

8. abs(temp) < b, 2.9404455335485081 < 2.960198673306755, да, идём к блоку 9

9. Переприсваивание переменной b = abs(temp) = 2.9404455335485081

10. Переприсваивание переменной answer = a = 0.03

11. a = 0.04. Возврат к блоку 6

6. 0.04 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(-0.04) – 0.04 + 2 = 2.920789439152323. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = 2.920789439152323

8. abs(temp) < b, 2.920789439152323 < 2.9404455335485081, да, идём к блоку 9

9. Переприсваивание переменной b = abs(temp) = 2.920789439152323

10. Переприсваивание переменной answer = a = 0.04

11. a = 0.05. Возврат к блоку 6

6. 0.05 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(-0.05) – 0.05 + 2 = 2.901229424500714. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = 2.901229424500714

8. abs(temp) < b, 2.901229424500714 < 2.920789439152323, да, идём к блоку 9

9. Переприсваивание переменной b = abs(temp) = 2.901229424500714

10. Переприсваивание переменной answer = a = 0.05

11. a = 0.06. Возврат к блоку 6

Пройдя программа так множество раз, a = 2.12

6. 2.12 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(-2.12) – 2.12 + 2 = 0.00003162851. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = 0.00003162851

8. abs(temp) < b, 0.00003162851 < 0.011237966433381(при a = 2.11) да, идём к блоку 9

9. Переприсваивание переменной b = abs(temp) = 0.00003162851

10. Переприсваивание переменной answer = a = 2.12

11. a = 2.13. Возврат к блоку 6

6. 2.13 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(-2.13) – 2.13 + 2 = - 0.0111627061475903. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = - 0.0111627061475903

8. abs(temp) < b, 0.0111627061475903 < 0.00003162851, нет, идём к блоку 11

11. a = 2.14. Возврат к блоку 6

6. 2.14 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(-2.14) – 2.14 + 2 = - 0.0223451569782208. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = - 0.0223451569782208

8. abs(temp) < b, 0.0223451569782208 < 0.00003162851, нет, идём к блоку 11

11. a = 2.15. Возврат к блоку 6

6. 2.15 < 5, да, идём к блоку 7

7. Объявление переменной temp = equation(a). Переходим к выполнению подпрограммы equation(x)

13. Возврат e^(-x) – x + 2 = e^(-2.15) – 2.15 + 2 = - 0.033515842226503. Завершаем выполнение подпрограммы equation(x). Идём обратно к блоку 7.

7. temp = - 0.033515842226503

8. abs(temp) < b, 0.033515842226503 < 0.00003162851, нет, идём к блоку 11

11. a = 2.16. Возврат к блоку 6

Пройдя программа так множество раз, a = 5

6. 5 < 5, нет, идём к блоку 12

12. Вывод answer = 2.12

14. Конец

Text

Description automatically generated

Рисунок 4 – Скриншот правдивости отладки

# 4.4 Код вычислительной процедуры

def method\_of\_bruteforce(n1):

def equation(x):

return exp(-x) - x + 2

n, accuracy = n1, -1.5

b, an = equation(n), -1

while accuracy < 5:

if abs(equation(accuracy)) < b:

b = abs(equation(accuracy))

an = accuracy

accuracy += n

return an

# 4.5 Результат / скриншот

Text

Description automatically generated

Рисунок 5 – результат работы

Text

Description automatically generated

Рисунок 6 – результат работы

4.6 Промежуточный вывод: из результатов можно понять, что программа довольно эффективна при маленькой точности (скажем, шага в 0.1), однако, чем меньше шаг, тем сильнее увеличивается время работы программы, и при точности 1е+09 программа уже работает довольно долго.

# 5 Решение уравнения с помощью Метода итераций

Задание 3: найти решение уравнения на указанном диапазоне с заданной точностью, используя численный метод – Метод итераций (последовательных приближений).

5.1 Общее теоретическое описание метода

Этот метод решения уравнений состоит в замене исходного уравнения эквивалентным ему выражением, где в левой части находится чистая переменная «x», а в правой - некое выражение, содержащее «x». Для начала найдём примерный корень уравнения, чтобы метод был гораздо точнее - для этого будем постепенно сжимать заданный отрезок. Начальный сегмент блок-схемы и кода, с громадным количеством условий, лишь шаг поиска решения, ведь он заключается в нахождении примерного ответа с целью уже заранее приблизить алгоритм к нужному числу. Число «а» постоянно растёт, в то время как «b» уходит в минус бесконечность. Это позволяет исследовать функцию на всей числовой оси Oх в целых числах. Как только находится отрезок ответа, берётся его середина и отправляется в указанное пользователем количество итераций, являющих собой вычисления ранее упомянутого выражения от найденного приближённого значения ответа, затем – подставление полученного значения обратно в выражение, вычисление его от полученного значения – и этот цикл будет длится столько раз, сколько пожелает пользователь – именно поэтому точность в программе является натуральным числом – ведь не может быть 1.5 итерации, к примеру. После всех вышеописанных процедур алгоритм выдаёт ответ.

# 5.2 Таблица имён

|  |  |  |  |
| --- | --- | --- | --- |
| Исходные данные | | | |
| n | Целочисленный | | Указанная точность |
| Рабочие переменные | | | |
| i | | Целочисленный | Переменная-счётчик |
| Результат | | | |
| b | | Целочисленный | Ответ уравнения |

Таблица 2 – Таблица имён

# 5.3 Блок-схема

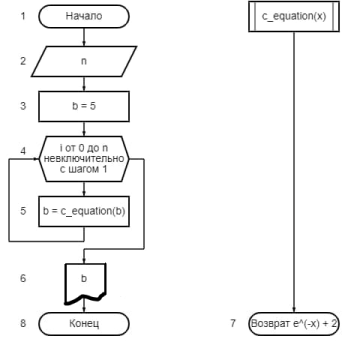


Рисунок 7 – блок-схема

# 5.3.1 Отладка блок-схемы

1. Начало

2. Ввод с клавиатуры n = 5

3. Объявление переменных: b = 5

4. Цикл: i = 0; 0 < 5; i = 1

5. Переходим к выполнению подпрограммы c\_equation(x)

7. Возврат e^(-x) + 2 = e^(-5) + 2 = 2.0067379469990856. Завершаем выполнение подпрограммы c\_equation(x). Идём обратно к блоку 5.

5. Присваивание переменной b = 2.0067379469990856

4. Цикл: i = 1; 1 < 5; i = 2

5. Переходим к выполнению подпрограммы c\_equation(x)

7. Возврат e^(-x) + 2 = e^(-2.0067379469990856) + 2 = 2.1344264664889456. Завершаем выполнение подпрограммы c\_equation(x). Идём обратно к блоку 5.

5. Присваивание переменной b = 2.1344264664889456

4. Цикл: i = 2; 2 < 5; i = 3

5. Переходим к выполнению подпрограммы c\_equation(x)

7. Возврат e^(-x) + 2 = e^(-2.1344264664889456) + 2 = 2.1183124270631644. Завершаем выполнение подпрограммы c\_equation(x). Идём обратно к блоку 5.

5. Присваивание переменной b = 2.1183124270631644

4. Цикл: i = 3; 3 < 5; i = 4

5. Переходим к выполнению подпрограммы c\_equation(x)

7. Возврат e^(-x) + 2 = e^(-2.1183124270631644) + 2 = 2.120234361654666. Завершаем выполнение подпрограммы c\_equation(x). Идём обратно к блоку 5.

5. Присваивание переменной b = 2.120234361654666

4. Цикл: i = 4; 4 < 5; i = 5

5. Переходим к выполнению подпрограммы c\_equation(x).

7. Возврат e^(-x) + 2 = e^(-2.120234361654666) + 2 = 2.120003500996521. Завершаем выполнение подпрограммы c\_equation(x). Идём обратно к блоку 5.

5. Присваивание переменной b = 2.120003500996521

4. Цикл: i = 5; 5 = 5 – выход из цикла

6. Вывод b = 2.120003500996521

8. Конец

Graphical user interface, text

Description automatically generated

Рисунок 8 – Скриншот правдивости отладки

# 5.4 Код вычислительной процедуры:

from math import exp

def c\_equation(x):

return exp(-x) + 2

b = 5

for I in range(input()):

b = c\_equation(b)

print(b)

# 5.5 Результат / скриншот

Text

Description automatically generated

Рисунок 9 – результат работы

Text

Description automatically generated

Рисунок 10 – результат работы

5.6 Промежуточный вывод: программа работает исправно с методом итераций, при заданном значении пользователем (вплоть до 1E+09, там программа работает уж слишком долго), выдавая конкретный ответ в окне вывода.

# Решение уравнения двумя методами

# 6.1 Общее теоретическое описание метода

Это метод решения уравнения двумя методами: итераций и прямого перебора значений (вычисления идут последовательно, отображаются уже в завершённом виде на экране программы, как будет показано далее).

# 6.2 Код вычислительной процедуры

# Так как этот метод по своей сути является просто сложением двух # предыдущих методов, то следует немного точнее сказать – это не КВП (код # вычислительной процедуры), а скорее КГИ (код графического интерфейса).  
# Оставим прежнее название для поддержания структуры отчёта касательно # пунктов 2 и 3.

def special():

def destroyer():

avocado.destroy()

bi.destroy()

words.destroy()

request.destroy()

something.destroy()

helped.destroy()

if please:

tr.destroy()

tr1.destroy()

comeback()

def callback(event):

global please, something

if something.get().isdigit():

if 0 < int(something.get()) < 10000001:

please = True

else:

helped.configure(text='Что же, как вводить число, вы поняли.\n'

'Но всё-таки следуйте требованиям по вводу, хорошо?', fg='red')

else:

helped.configure(text='Что же, как вводить число, вы поняли.\n'

' Но всё-таки следуйте требованиям по вводу, хорошо?', fg='red')

def strange():

global please, something

if something.get().isdigit():

if 0 < int(something.get()) < 10000001:

please = True

else:

helped.configure(text='Что же, как вводить число, вы поняли.\n'

'Но всё-таки следуйте требованиям по вводу, хорошо?', fg='red')

else:

helped.configure(text='Что же, как вводить число, вы поняли.\n'

' Но всё-таки следуйте требованиям по вводу, хорошо?', fg='red')

global please, something

bi = Button(window, text="Рассчитать!", bg='black', fg='white', command=strange)

helped = Label(window, text='(Для ввода нажмите Enter, или кнопку под полем ввода)', bg='black', fg='white')

something = Entry(window, width=15, fg='white', bg='black')

window.geometry("500x110")

lbl.destroy()

lbl1.destroy()

lbl2.destroy()

bt.destroy()

bt1.destroy()

btn2.destroy()

window.update()

avocado = Button(window, text="Вернуться", bg='black', fg='white', command=destroyer)

avocado.place(x=425, y=80)

words = Label(window, text='Вы выбрали способ нахождения решения уравнениями обоими методами.',

bg='black', fg='white')

words.place(x=5, y=2)

request = Label(window, text='Пожалуйста, укажите точность целым числом, до 10000000!',

bg='black', fg='white')

request.place(x=5, y=18)

helped.place(x=110, y=40)

something.place(x=10, y=47)

bi.place(x=10, y=80)

please = False

while not please:

window.update()

window.bind('<Return>', callback)

var = something.get()

mot = Decimal(method\_of\_iterations(int(var)))

mob = Decimal(method\_of\_bruteforce(1/int(var)))

tr = Label(window, text=f'Методом итерации получено значение х = {round(mot, 22)}', bg='black', fg='white')

tr1 = Label(window, text=f'Методом перебора получено значение х = {round(mob, 22)}', bg='black', fg='white')

if count != 0:

tr.configure(text=f'Методом итерации получено значение х = {mot}')

tr1.configure(text=f'Методом итерации получено значение х = {mob}')

tr.place(x=110, y=35)

tr1.place(x=110, y=55)

window.mainloop()

# 6.3 Результат / скриншот

Text

Description automatically generated

Рисунок 11 – результат работы

6.4 Промежуточный вывод: программа работает исправно как с методом итераций, так и с методом прямого перебора значений (параллельно) при заданных значениях пользователем (вплоть до 1E-23 и 1E+09 соответственно), выдавая конкретные ответы в окнах вывода.

# 7 Проектирование графического интерфейса

Задание 4

Для реализации графики использовался модуль Tkinter. Так как проект изначально строился на С++, но затем возникли трудности написания окон через WinAPI и последующих операций с ними. Было принято решение писать в среде с максимальной простотой, т.к. по заданию нужно лишь взаимодействие с пользователем. Поэтому был выбран модуль Tkinter т.к. данная библиотека содержит достаточное количество необходимых функций для связки основных вычислительных процедур с графическим интерфейсом.

Графический интерфейс в программе представлен несколькими функциями:  
1) comeback (Главное меню программы, начальное окно)

2) clicked (Обработка решения пользователя после выбора метода решения)

2) iterations (Окно с возможностью расчёта методом итераций (посл. прибл.))  
3) brute (Окно с возможностью расчёта методом полного перебора)  
4) special (Окно с возможностью для обоих подсчётов одновременно)

# 7.1 Comeback()

Как уже упоминалось ранее, эта функция в конечном итоге является самым первым окном, которое видит пользователь. В ней задаётся название окна “Solver assistant”, задний фон выполнен чёрным цветом, созданы виджеты – линии текста и кнопки (3 штуки, две типа Checkbutton, последняя типа Button). Пользователь может выбрать любой из методов для расчёта, и с помощью нажатия кнопки <Выбор сделан> получить желаемый способ. Сразу же напрашиваются два вопроса – что, если пользователь одновременно выберет оба способа? Или, наоборот, просто забудет о выборе, и сразу нажмёт на кнопку с подтверждением выбора?

Эти своеобразные исключения необходимо обработать. За это отвечает функция clicked.

# 7.1.1 Код функции

def comeback():

global lbl, lbl1, lbl2, lb3, sel, sel1, bt, bt1, btn2

window.title("Solver Assistant")

window['bg'] = ['black']

lbl = Label(window, text="Задано иррациональное уравнение exp(-x) - x + 2 = 0", bg='black', fg='white')

lbl1 = Label(window, text="на интервале от -1.5 до 5.0 включительно.", bg='black', fg='white')

lbl2 = Label(window, text="Какой метод используем для вычисления переменной х?", bg='black', fg='white')

lb3 = Label(window, text='', bg='black', fg='red')

lbl.place(x=17, y=0)

lbl1.place(x=45, y=20)

lbl2.place(x=0, y=40)

sel = IntVar()

sel1 = IntVar()

bt = Checkbutton(window, text='Метод итераций (последовательных приближений)', onvalue=1, offvalue=0, variable=sel, bg='black', fg='white', selectcolor='red')

bt1 = Checkbutton(window, text='Метод прямого перебора чисел в рамках отрезка', onvalue=1, offvalue=0, variable=sel1, bg='black', fg='white', selectcolor='red')

bt.place(x=0, y=60)

bt1.place(x=0, y=80)

btn2 = Button(window, text="Выбор сделан", command=clicked, width='15', bg='black', fg='white')

btn2.place(x=5, y=110)

lb3.place(x=130, y=115)

window.resizable(0, 0

window.geometry("335x140")

window.mainloop()

# 7.1.2 Скриншот работы программы

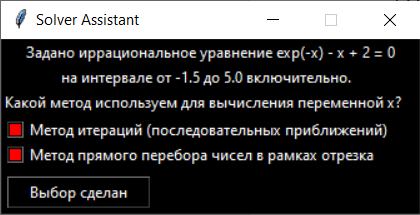


Рисунок 12 – меню выбора

# 7.2 Clicked()

Данная работа сразу запрашивает данные о выборе пользователя через переменные f, g, которые в свою очередь берут значения из переменных sel, sel1, которые в свою очередь напрямую привязаны к состояниям 2-х кнопок типа Checkbutton (Активное значение – 1, неактивное - 0). Если f, g оказались нулями, тогда функция лишь немного модернизирует начальное окно, красным текстом сообщив пользователю о необходимости выбора хотя бы одного варианта решения уравнения. Остальные выборы достаточно прямолинейны, так что если пользователь выберет метод итераций, то программа уйдёт в функцию iterations(), если выбран метод прямого перебора, то здесь будет иметь функция brute(), а если f, g обе равны единицам, тогда запускается особая функция special(), которая будет подсчитывать сразу два значения на основе введённой пользователем точности.

# 7.2.1 Код функции

f, g = sel.get(), sel1.get()

if f == 1 and g == 1:

special()

elif g == 1 and f == 0:

brute()

elif g == 0 and f == 1:

iterations()

else:

lb3.configure(text='Выберите хотя бы один вариант.')

# 7.2.2 Скриншот работы программы

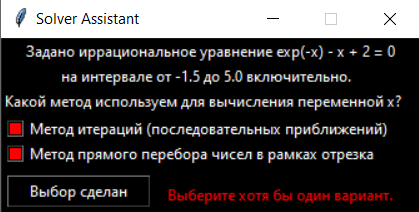


Рисунок 13 – пользователь не выбрал метод

# 7.3 Iterations()

Данная функция отвечает за подсчёт решения уравнения методом итераций (последовательных приближений). Она имеет в себе три подфункции – destroyer, callback, strange. Сперва меняется начальное окно по размеру, все прошлые виджеты исчезают с экрана, заменяясь новыми. Пользователь задаёт точность в окне типа Entry, которое получит на вход основная вычислительная процедура, а после этого выдаст ответ. Однако пользователь может ввести и неправильную точность, несмотря на предупреждение о границах введённой точности. Тогда функции callback и strange отвечают за получение значения путём нажатия клавиши Enter или простым нажатием кнопки <Рассчитать>, расположенной под полем ввода соответственно. Поэтому, если вдруг введено неверное значение, программа аналогично в первой функции красным текстом предупредит пользователя о необходимости соблюдения правил ввода.

# 7.3.1 Код функции

def iterations():

def destroyer():

new.destroy()

new1.destroy()

answer.destroy()

killjoy.destroy()

ass.destroy()

bi.destroy()

angriest.destroy()

if k:

wer.destroy()

comeback()

def callback(event):

global k, answer#ввод пользователя

if answer.get().isdigit():

if 0 < int(answer.get()) < 10000001:

k = True

else:

angriest.place(x=85, y=67)

else:

angriest.place(x=85, y=67)

def strange():

global k, answer

if answer.get().isdigit():#answer - поле для ввода, isdigit - вернёт false при проверке, если введено не число

if 0 < int(answer.get()) < 10000001:

k = True

else:

angriest.place(x=85, y=67)

else:

angriest.place(x=85, y=67)

bi = Button(window, text="Рассчитать!", bg='black', fg='white', command=strange)

bi.place(x=5, y=65)

global k, answer

window.geometry("310x110")

lbl.destroy()

lbl1.destroy()

lbl2.destroy()

bt.destroy()

bt1.destroy()

btn2.destroy()

window.update()

killjoy = Button(window, text='Вернуться', bg='black', fg='white', command=destroyer)

killjoy.place(x=240, y=65)

new = Label(window, text='Вы выбрали способ нахождения методом итераций.', bg='black', fg='white')

new1 = Label(window, text='Укажите точность натуральным числом до 10000000:', bg='black', fg='white')

ass = Label(window, text='(Для ввода нажмите Enter или кнопку под полем ввода)', bg='black', fg='white')

angriest = Label(window, text='Введите требуемое число!', bg='black', fg='red')

answer = Entry(window, bg='black', fg='white')#поле ввода

ass.place(x=0, y=90)

new.place(x=0, y=0)

new1.place(x=0, y=20)

answer.place(x=5, y=45)

window.update()

k = False#пользователь ничего не ввёл

while not k:

window.update()

window.bind("<Return>", callback)#после нажатия enter вызываем функцию callback

wer = Label(window, text=f'x = {method\_of\_iterations(int(answer.get()))}', bg='black', fg='white')#с помощью метода get получаем значение вводимое пользователем

new.configure(text=f'Для указанного значения точности {answer.get()} (итераций)')

new1.configure(text='Ответ к заданному уравнению: ')

new1.place(x=0, y=20)

wer.place(x=175, y=20)

window.geometry('310x95')

angriest.destroy()

ass.destroy()

# 7.3.2 Скриншоты работы программы

Работа окна в изначальном режиме:

Graphical user interface, text

Description automatically generated

Рисунок 14 – меню метода итераций

Работа программы в случае неправильного ввода:

Text

Description automatically generated

Рисунок 15 – пользователь ввёл неподходящее число

Работа программы в случае успешного ввода:

Text

Description automatically generated

Рисунок 16 – результат работы

# 7.4 Brute()

Данная функция отвечает за подсчёт решения уравнения методом прямого перебора значений. Также имеет в себе 3 подфункции: destroyer, callback и strange. После того, как пользователь выбрал этот метод, окно остаётся таким же, но виджеты текста, ввода и кнопок заменяются. Пользователь также должен ввести значение точности и нажать кнопку <Рассчитать> или Enter на клавиатуре. Здесь же точность измеряется в другом размере. Если пользователь введёт недопустимое значение точности - strange и callback получат это значение и выведут предупреждающую табличку.

# 7.4.1 Код функции

def brute():

def destroyer():

new.destroy()

new1.destroy()

new2.destroy()

answer1.destroy()

bi.destroy()

limelighting.destroy()

if p:

wer.destroy()

comeback()

def callback(event):

global answer1, p

if answer1.get()[2:].isdigit() and answer1.get()[:2] == "0.":

if 1 > float(answer1.get()) > 0:

p = True

else:

new2.configure(text='Если разобрались со вводом,\n введите число согласно требованиям!', fg='red')

else:

new2.configure(text='Если разобрались со вводом,\n введите число согласно требованиям!', fg='red')

def strange():

global answer1, p

if answer1.get()[2:].isdigit() and answer1.get()[:2] == "0.":#срезы

if 1 > float(answer1.get()) > 0:

p = True

else:

new2.configure(text='Если разобрались со вводом,\n введите число согласно требованиям!', fg='red')

else:

new2.configure(text='Если разобрались со вводом,\n введите число согласно требованиям!', fg='red')

global answer1, p

bi = Button(window, text="Рассчитать!", bg='black', fg='white', command=strange)

bi.place(x=5, y=70)

window.geometry("450x100")

lbl.destroy()

lbl1.destroy()

lbl2.destroy()

bt.destroy()

bt1.destroy()

btn2.destroy()

window.update()

limelighting = Button(window, text="Вернуться", bg='black', fg='white', command=destroyer)

limelighting.place(x=380, y=70)

new = Label(window, text='Вы выбрали способ нахождения методом прямого перебора значений.', bg='black', fg='white')

new1 = Label(window, text='Укажите точность в размере от от 0.0000000000000000000001 до 1:', bg='black', fg='white')

new2 = Label(window, text='(После ввода нажмите клавишу Enter)\n(Или кнопку под полем ввода)',

bg='black', fg='white')

new2.place(x=150, y=45)

answer1 = Entry(window, bg='black', fg='white')

new.place(x=0, y=0)

new1.place(x=0, y=20)

answer1.place(x=5, y=50)

window.update()

p = False

while not p:

window.update()

window.bind("<Return>", callback)

wer = Label(window, text=f'x = {method\_of\_bruteforce(float(answer1.get()))}', bg='black', fg='white')

new.configure(text=f'Для указанного значения точности {answer1.get()} (ухода к нулю)')

new1.configure(text='Ответ к заданному уравнению составляет')

new2.destroy()

new1.place(x=0)

wer.place(x=240, y=20)

# 7.4.2 Скриншоты работы программы

Работа окна в изначальном режиме:

Text

Description automatically generated

Рисунок 17 – меню метода прямого перебора

Работа программы в случае неправильного ввода:

Text

Description automatically generated

Рисунок 18 – пользователь ввёл неподходящее число

Работа программы в случае успешного ввода:

Text

Description automatically generated

Рисунок 19 – результат работы

# 7.5 Special()

Данная функция отвечает за подсчёт решения уравнения двумя методами: методом итераций и методом прямого перебора значений (параллельно).

Функция special имеет в себе 3 подфункции: destroyer, callback и strange. Переходя к этому методу окно программы остаётся тем же (кроме как размера самого окна), но виджеты изменяются. Пользователь задаёт значение точности в окне Entry, которое получит программа и выдаст ответ. Также пользователь может задать значение недопустимое для программы, тогда после нажатия пользователем кнопки Enter с клавиатуры или кнопки <Рассчитать> в программе - callback и strange получают и обрабатывают данное значение и на экран выводится предупреждающая табличка.

# 7.5.1 Код функции

Код графической процедуры вставлен в пункте 6.2.

# 7.5.2 Скриншоты работы программы

Работа окна в изначальном режиме:

Text

Description automatically generated

Рисунок 20 – меню обоих методов

Работа программы в случае неправильного ввода:

Text

Description automatically generated

Рисунок 21 – пользователь ввёл неподходящее число

Работа программы в случае успешного ввода:

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 22 - результат работы

# 8 Оценка времени решения

Задание 5

Определить время вычисления по процедурам (2) и (3), сравнить и пояснить их. Определяем через модуль time в языке Python. Для увеличения точности расчёта времени каждое измерение было рассчитано из арифметической суммы измерений (примерно десять тысяч слагаемых) при одинаковой, заданной пользователем точности вычисления корней исходного уравнения. Результат не округлён для увеличения количества чисел, после запятой.

# 8.1 Замер времени выполнения программы прямого перебора.

1.0: 0.0000012304927520826136 seconds

1e-01: 0.0010011196136474609375 seconds

1e-02: 0.0013002030297353223108 seconds

1e-03: 0.0030019283294677734375 seconds

1e-04: 0.0310287475585937500040 seconds

1e-05: 0.3072805404663085937501 seconds

1e-06: 3.0687961578369140625007 seconds

1e-07: 30.9168746471405029296594 seconds

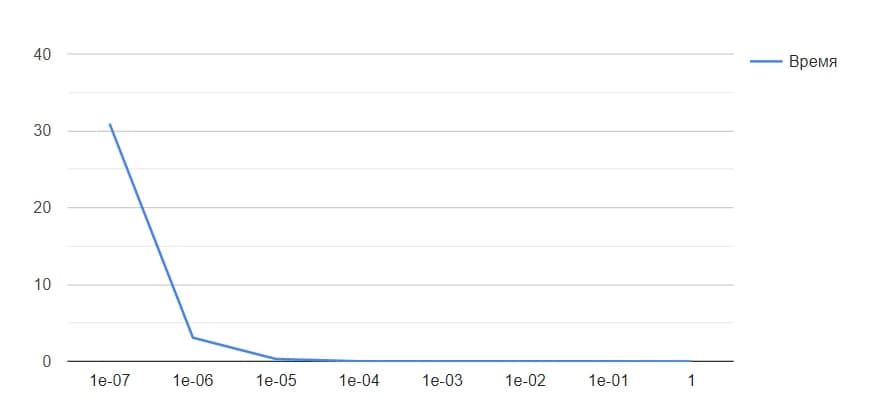


Рисунок 23 – график зависимости при методе перебора

Вывод: построив график, можно увидеть зависимость точности от времени исполнения: чем выше точность – тем выше время измерения.

# 8.2 Замер времени выполнения программы методом итераций.

1.0: 0.0000246128909523612904 seconds

1e+01: 0.0000213759878774230001 seconds

1e+02: 0.0000276500879000623412 seconds

1e+03: 0.0000421287809219850013 seconds

1e+04: 0.0020015239715576171875 seconds

1e+05: 0.0230798721313476562455 seconds

1e+06: 0.213983058929443359374 seconds

1e+07: 2.16754245758056640621 seconds

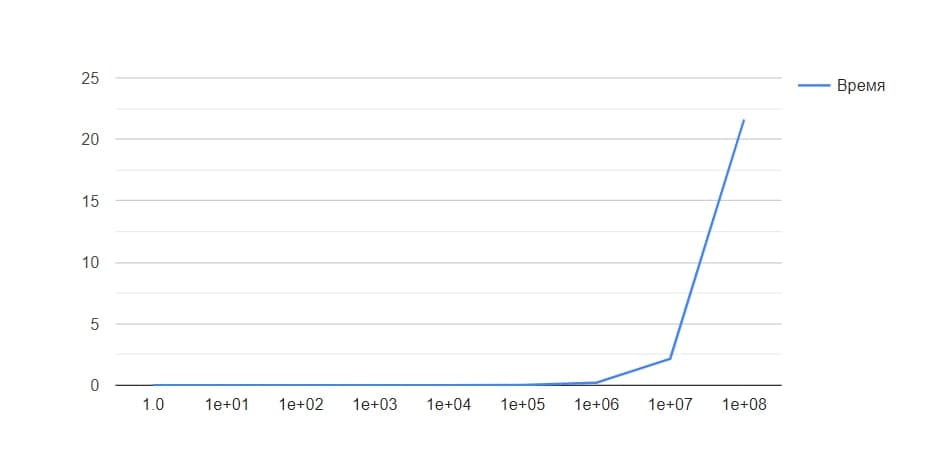
1e+08: 21.6156733036041259765625 seconds

Рисунок 24 – график зависимости при методе итераций

Вывод: построив график, можно увидеть зависимость точности от времени исполнения: чем выше точность – тем выше время измерения

8.3 Пояснения.

Из приведённых графиков видно, что по скорости вычисления при повышающейся точности метод итераций доминирует. Почему же так происходит? Попробуем ответить на этот вопрос через вычисление сложности алгоритма этих двух методов.

Начнём с метода итераций. Здесь всё довольно просто – сколько пользователь задал значение точности, столько раз и прогоним цикл вычислений. Следовательно, его сложность составляет O(n).

Далее попробуем оценить сложность метода прямого перебора. Если будем прогонять цикл вычисления с шагом 1, то весь отрезок будет пройден за 7 шагов, если с шагом 0.1 – тогда 65, далее при 0.01 получаем 650 шагов, и по итогу приходим к простому выводу – если задали точность n, тогда получим примерно 6,5\*(1/n) шагов. Однако осталась маленькая неточность, которую стоит учесть – если подставим, к примеру 1 или -0,2 получим 7 и 33 шагов соответственно. Следовательно, необходимо округлять полученное значение в большую сторону – ведь если подставим 1 или 0,2 в нашу формулу, получим соответственно 6,5 и 32,5 шагов соответственно. Поэтому итоговая сложность алгоритма – O(⌈(6,5\*(1/n))).

Как можем видеть, метод итераций по эффективнее, поэтому и наблюдаем некоторое превосходство одного алгоритма над другим.

# 9 Вывод:

В ходе выполнения работы были изучены: метод итераций, метод прямого перебора значений.

Было проделано: найдено решение уравнения на указанном диапазоне с заданной точностью, использовав модуль «поиск решения» (Excel).

Найдено решение уравнения на указанном диапазоне с заданной точностью, используя метод прямого перебора значений в пределах диапазона.

Найдено решение уравнения на указанном диапазоне с заданной точностью, используя численный метод – Метод итераций (последовательных приближений).

Программа написана с применением элементов графического интерфейса пользователя (окна, кнопки, поля ввода, строк текста и т.д.)

Было измерено время работы каждой из основных вычислительных процедур с помощью многократных измерений, построены графики зависимости времени работы алгоритма от заданной точности.

Был оформлен комплект документации на проектную работу.

# 10 Приложение A. Код программы from math import \* from decimal import \* from tkinter import \* k, p, please = [False for \_ in range(3)] answer, answer1, something, count, lbl, lbl1, lbl2, lb3, sel, sel1, bt, bt1, btn2 = [0 for \_ in range(13)] window = Tk() def method\_of\_iterations(n): def c\_equation(x): return exp(-x) + 2 b = 5 for \_ in range(n): b = c\_equation(b) return b def method\_of\_bruteforce(n1): def equation(x): return exp(-x) - x + 2 n, accuracy = n1, -1.5 b, an = equation(n), -1 while accuracy < 5: if abs(equation(accuracy)) < b: b = abs(equation(accuracy)) an = accuracy accuracy += n return an def iterations(): def destroyer(): new.destroy() new1.destroy() answer.destroy() killjoy.destroy() ass.destroy() bi.destroy() angriest.destroy() if k: wer.destroy() comeback() def callback(event): global k, answer#ввод пользователя if answer.get().isdigit(): if 0 < int(answer.get()) < 10000001: k = True else: angriest.place(x=85, y=67) else: angriest.place(x=85, y=67) def strange(): global k, answer if answer.get().isdigit():#answer - поле для ввода, isdigit - вернёт false при проверке, если введено не число if 0 < int(answer.get()) < 10000001: k = True else: angriest.place(x=85, y=67) else: angriest.place(x=85, y=67) bi = Button(window, text="Рассчитать!", bg='black', fg='white', command=strange) bi.place(x=5, y=65) global k, answer window.geometry("310x110") lbl.destroy() lbl1.destroy() lbl2.destroy() bt.destroy() bt1.destroy() btn2.destroy() window.update() killjoy = Button(window, text='Вернуться', bg='black', fg='white', command=destroyer) killjoy.place(x=240, y=65) new = Label(window, text='Вы выбрали способ нахождения методом итераций.', bg='black', fg='white') new1 = Label(window, text='Укажите точность натуральным числом до 10000000:', bg='black', fg='white') ass = Label(window, text='(Для ввода нажмите Enter или кнопку под полем ввода)', bg='black', fg='white') angriest = Label(window, text='Введите требуемое число!', bg='black', fg='red') answer = Entry(window, bg='black', fg='white')#поле ввода ass.place(x=0, y=90) new.place(x=0, y=0) new1.place(x=0, y=20) answer.place(x=5, y=45) window.update() k = False#пользователь ничего не ввёл while not k: window.update() window.bind("<Return>", callback)#после нажатия enter вызываем функцию callback wer = Label(window, text=f'x = {method\_of\_iterations(int(answer.get()))}', bg='black', fg='white')#с помощью метода get получаем значение вводимое пользователем new.configure(text=f'Для указанного значения точности {answer.get()} (итераций)') new1.configure(text='Ответ к заданному уравнению: ') new1.place(x=0, y=20) wer.place(x=175, y=20) window.geometry('310x95') angriest.destroy() ass.destroy() def brute(): def destroyer(): new.destroy() new1.destroy() new2.destroy() answer1.destroy() bi.destroy() limelighting.destroy() if p: wer.destroy() comeback() def callback(event): global answer1, p if answer1.get()[2:].isdigit() and answer1.get()[:2] == "0.": if 1 > float(answer1.get()) > 0: p = True else: new2.configure(text='Если разобрались со вводом,\n введите число согласно требованиям!', fg='red') else: new2.configure(text='Если разобрались со вводом,\n введите число согласно требованиям!', fg='red') def strange(): global answer1, p if answer1.get()[2:].isdigit() and answer1.get()[:2] == "0.":#срезы if 1 > float(answer1.get()) > 0: p = True else: new2.configure(text='Если разобрались со вводом,\n введите число согласно требованиям!', fg='red') else: new2.configure(text='Если разобрались со вводом,\n введите число согласно требованиям!', fg='red') global answer1, p bi = Button(window, text="Рассчитать!", bg='black', fg='white', command=strange) bi.place(x=5, y=70) window.geometry("450x100") lbl.destroy() lbl1.destroy() lbl2.destroy() bt.destroy() bt1.destroy() btn2.destroy() window.update() limelighting = Button(window, text="Вернуться", bg='black', fg='white', command=destroyer) limelighting.place(x=380, y=70) new = Label(window, text='Вы выбрали способ нахождения методом прямого перебора значений.', bg='black', fg='white') new1 = Label(window, text='Укажите точность в размере от от 0.00000001 до 1:', bg='black', fg='white') new2 = Label(window, text='(После ввода нажмите клавишу Enter)\n(Или кнопку под полем ввода)', bg='black', fg='white') new2.place(x=150, y=45) answer1 = Entry(window, bg='black', fg='white') new.place(x=0, y=0) new1.place(x=0, y=20) answer1.place(x=5, y=50) window.update() p = False while not p: window.update() window.bind("<Return>", callback) wer = Label(window, text=f'x = {method\_of\_bruteforce(float(answer1.get()))}', bg='black', fg='white') new.configure(text=f'Для указанного значения точности {answer1.get()} (шага по интервалу)') new1.configure(text='Ответ к заданному уравнению составляет') new2.destroy() new1.place(x=0) wer.place(x=240, y=20) def special(): def destroyer(): avocado.destroy() bi.destroy() words.destroy() request.destroy() something.destroy() helped.destroy() if please: tr.destroy() tr1.destroy() comeback() def callback(event): global please, something if something.get().isdigit(): if 0 < int(something.get()) < 10000001: please = True else: helped.configure(text='Что же, как вводить число, вы поняли.\n' 'Но всё-таки следуйте требованиям по вводу, хорошо?', fg='red') else: helped.configure(text='Что же, как вводить число, вы поняли.\n' ' Но всё-таки следуйте требованиям по вводу, хорошо?', fg='red') def strange(): global please, something if something.get().isdigit(): if 0 < int(something.get()) < 10000001: please = True else: helped.configure(text='Что же, как вводить число, вы поняли.\n' 'Но всё-таки следуйте требованиям по вводу, хорошо?', fg='red') else: helped.configure(text='Что же, как вводить число, вы поняли.\n' ' Но всё-таки следуйте требованиям по вводу, хорошо?', fg='red') global please, something bi = Button(window, text="Рассчитать!", bg='black', fg='white', command=strange) helped = Label(window, text='(Для ввода нажмите Enter, или кнопку под полем ввода)', bg='black', fg='white') something = Entry(window, width=15, fg='white', bg='black') window.geometry("500x110") lbl.destroy() lbl1.destroy() lbl2.destroy() bt.destroy() bt1.destroy() btn2.destroy() window.update() avocado = Button(window, text="Вернуться", bg='black', fg='white', command=destroyer) avocado.place(x=425, y=80) words = Label(window, text='Вы выбрали способ нахождения решения уравнениями обоими методами.', bg='black', fg='white') words.place(x=5, y=2) request = Label(window, text='Пожалуйста, укажите точность натуральным числом, до 10000000!', bg='black', fg='white') request.place(x=5, y=18) helped.place(x=110, y=40) something.place(x=10, y=47) bi.place(x=10, y=80) please = False while not please: window.update() window.bind('<Return>', callback) var = something.get() mot = Decimal(method\_of\_iterations(int(var))) mob = Decimal(method\_of\_bruteforce(1/int(var))) tr = Label(window, text=f'Методом итерации получено значение х = {round(mot, 22)}', bg='black', fg='white') tr1 = Label(window, text=f'Методом перебора получено значение х = {round(mob, 22)}', bg='black', fg='white') if count != 0: tr.configure(text=f'Методом итерации получено значение х = {mot}') tr1.configure(text=f'Методом итерации получено значение х = {mob}') tr.place(x=110, y=35) tr1.place(x=110, y=55) window.mainloop() def clicked(): f, g = sel.get(), sel1.get() if f == 1 and g == 1: special() elif g == 1 and f == 0: brute() elif g == 0 and f == 1: iterations() else: lb3.configure(text='Выберите хотя бы один вариант.')#если не выбрать не один вариант, меняется компаратор def comeback(): global lbl, lbl1, lbl2, lb3, sel, sel1, bt, bt1, btn2 window.title("Solver Assistant") window['bg'] = ['black'] lbl = Label(window, text="Задано иррациональное уравнение exp(-x) - x + 2 = 0", bg='black', fg='white')#label - строчка lbl1 = Label(window, text="на интервале от -1.5 до 5.0 включительно.", bg='black', fg='white') lbl2 = Label(window, text="Какой метод используем для вычисления переменной х?", bg='black', fg='white') lb3 = Label(window, text='', bg='black', fg='red')#нет текста lbl.place(x=17, y=0)#отсчёт ведётся всегда с левого верхнего угла(0;0) вправо положительный x вниз положительный y lbl1.place(x=45, y=20) lbl2.place(x=0, y=40) sel = IntVar()#считывание значений от кнопок и виджетов sel1 = IntVar() bt = Checkbutton(window, text='Метод итераций (последовательных приближений)', onvalue=1,#сам квадратик и то что можно поставить галочку offvalue=0, variable=sel, bg='black', fg='white', selectcolor='red') bt1 = Checkbutton(window, text='Метод прямого перебора чисел в рамках отрезка', onvalue=1, offvalue=0, variable=sel1, bg='black', fg='white', selectcolor='red') bt.place(x=0, y=60) bt1.place(x=0, y=80) btn2 = Button(window, text="Выбор сделан", command=clicked, width='15', bg='black', fg='white') btn2.place(x=5, y=110) lb3.place(x=130, y=115) window.resizable(0, 0)#окошко не растягиваемое из-за функции resizable window.geometry("335x140")#размер окошка window.mainloop()#зацикливание, чтобы окошко не убралось comeback()