**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 4**

по курсу «Численные методы»

Тема: Численные методы решения обыкновенных дифференциальных уравнений.

Студент: Сорокин Н.Э.

Группа: М8О-303Б-20

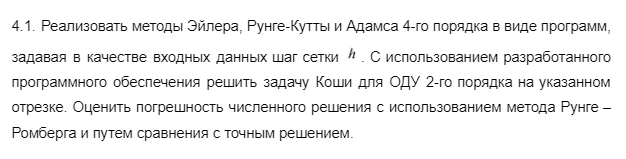
Преподаватель: Иванов И.Э.

Оценка:

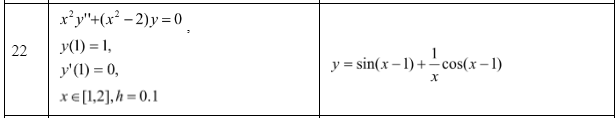
Москва, 2023

**№1**

**Постановка задачи:**

****

**Вариант 22:**

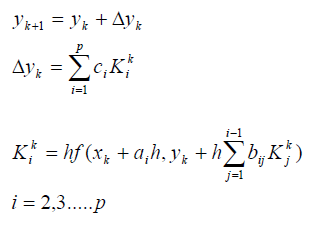
****

**Теория:**

Итерационная формула метода Эйлера:

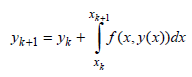


Совокупность формул для семейства методов Рунге-Кутты:

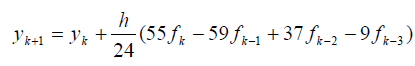


Коэффициенты подбираются так, чтобы значение у совпадало со значением разложения точного решения в ряд Тейлора в точке x.

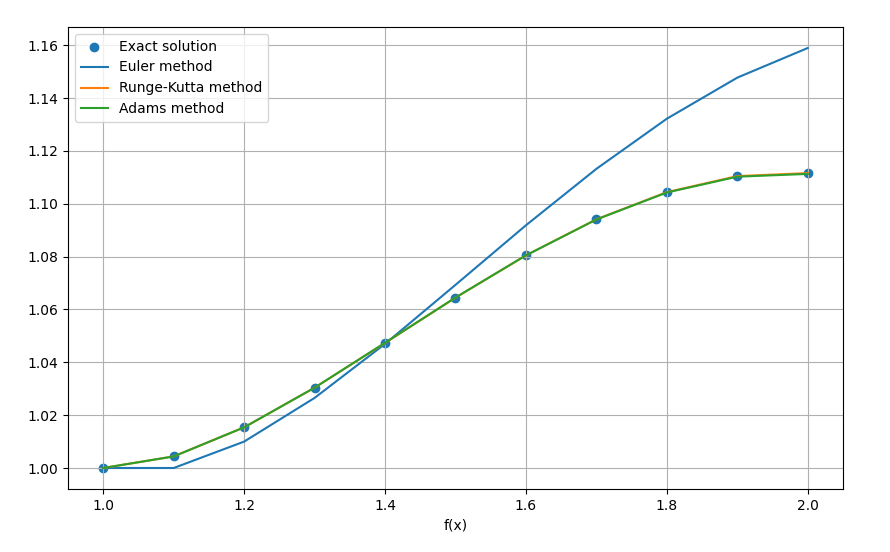
Решение дифф. уравнения удовлетворяет интегральному соотношению:



Если аппроксимировать подынтегральную функцию многочленом какой-либо степени, а затем вычислить интеграл, то можно получить формулу Адамса. При использовании многочлена 3 степени получим формулу Адамса 4 порядка:



**Полученный ответ:**

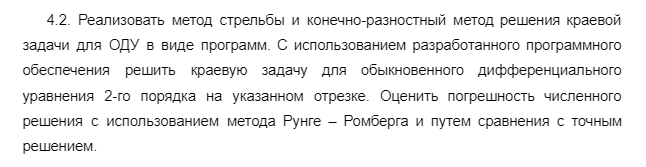
****

**Вывод:** В ходе выполнения лабораторной работе были реализованы методы Эйлера, Рунге-Кутты и Адамса 4-го порядка. Была оценена погрешность с помощью метода Рунге – Ромберга и путем сравнения с точным решением. Можно сделать вывод, что методы Рунге-Кутты и Адамса являются более точными чем метод Эйлера.

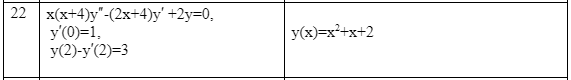
Можно также отметить, что метод Рунге-Кутты может быть вспомогательным в методах решения краевых задач.

**№2**

**Постановка задачи:**

****

**Вариант 22:**

****

**Теория:**

**Метод стрельбы:** Пусть необходимо решить краевую задачу





Вместо нее рассматривается задача Коши с начальными условиями:

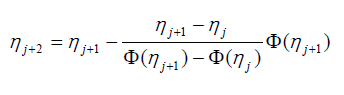


Для решения этих вспомогательных задач используется метод Рунге-Кутты для решения задач Коши.

Задача формулируется таким образом: требуется найти такое значение , чтобы решение в правом конце отрезка совпало со значением .

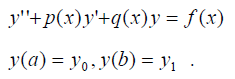
Это эквивалентно нахождению корня уравнения:

Для решения уравнения применяется итерационная формула:

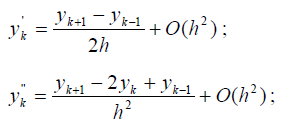


Итерации выполняются до удовлетворения заданной точности.

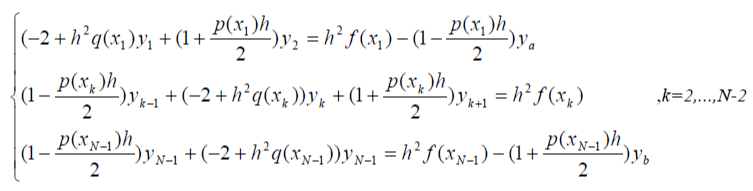
**Метод конечных разностей:** пусть необходимо решить краевую задачу вида:



Разобьем отрезок [a,b] на интервалы и аппроксимируем производные:

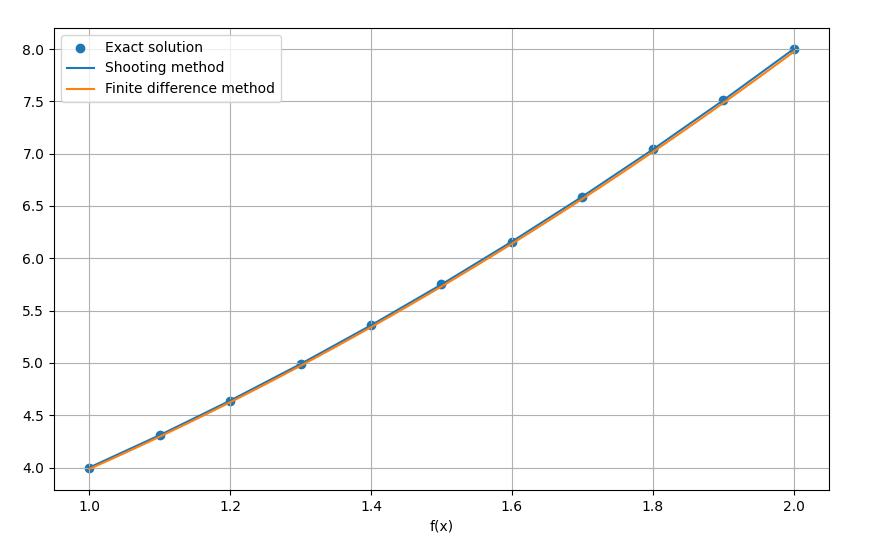


Подставляя аппроксимации и приведя подобные, получим систему линейных уравнений с трехдиагональной матрицей, которую удобно решать методом прогонки:



Решение системы будет являться решением исходной задачи в виде табличной функции.

**Полученный ответ:**

****

**Вывод:** В ходе выполнения лабораторной работе были реализованы метод стрельбы и конечно-разностный метод решения краевой задачи. Оба метода хорошо приближают аналитическое решение краевой задачи.

Заметим, что в методе конечных разностей используется метод прогонки реализованный в лабораторной работе №1.

**Код программы:**

import numpy as np

import matplotlib.pyplot as plt

from LinearAlgebra import \*

#module41

def Euler\_method(func, a, b, y0, x0, h):

x = np.arange(a, b + h / 2, h)

y = np.zeros((y0.size, x.size))

y[:,0] = y0

x[0] = x0

for i in range(x.size - 1):

y[:, i + 1] = y[:, i] + h \* func(y[:, i], x[i])

return x, y

def Runge\_Kutta\_method(func, a, b, y0, x0, h):

x = np.arange(a, b + h / 2, h)

y = np.zeros((y0.size, x.size))

y[:,0] = y0

x[0] = x0

for i in range(x.size - 1):

K1 = h \* func(y[:, i], x[i])

K2 = h \* func(y[:, i] + K1 / 2, x[i] + h / 2)

K3 = h \* func(y[:, i] + K2 / 2, x[i] + h / 2)

K4 = h \* func(y[:, i] + K3, x[i] + h)

y\_del = (K1 + 2 \* K2 + 2 \* K3 + K4) / 6

y[:, i + 1] = y[:, i] + y\_del

return x, y

def Adams\_method(func, a, b, y0, x0, h):

x = np.arange(a, b + h / 2, h)

y = np.zeros((y0.size, x.size))

y[:,0] = y0

x[0] = x0

print(x)

x[:4], y[:,:4] = Runge\_Kutta\_method(func, a, a + 2 \* h, y0, x0, h)

for i in range(3, x.size - 1):

y\_del = h \* (55 \* func(y[:, i], x[i])

- 59 \* func(y[:, i - 1], x[i - 1])

+ 37 \* func(y[:, i - 2], x[i - 2])

- 9 \* func(y[:, i - 3], x[i - 3])) / 24

y[:, i + 1] = y[:, i] + y\_del

return x, y

#module42

def shooting\_method(func, a, b, h):

eta1 = 1

eta2 = 0.8

phi = lambda y: y[0] - y[1] - 3

y002 = 3

y01 = np.array([eta1, y002])

x, y1 = Runge\_Kutta\_method(func, a, b, y01, a, h)

y02 = np.array([eta2, y002])

x, y2 = Runge\_Kutta\_method(func, a, b, y02, a, h)

y\_prev = y1

y\_curr = y2

eps = 0.001

eta\_prev = eta1

eta\_curr = eta2

phi\_prev = phi(y\_prev[:,-1])

phi\_curr = phi(y\_curr[:,-1])

k = 0

while(abs(phi\_curr) > eps and k < 100):

k += 1

eta\_next = eta\_curr - (eta\_curr - eta\_prev) / (phi\_curr - phi\_prev) \* phi\_curr

y0 = np.array([eta\_next, y002])

x, y\_next = Runge\_Kutta\_method(func, a, b, y0, a, h)

phi\_prev = phi\_curr

phi\_curr = phi(y\_next[:,-1])

eta\_prev = eta\_curr

y\_prev = y\_curr

eta\_curr = eta\_next

y\_curr = y\_next

return x, y\_curr

def finite\_difference\_method(func, a, b, h):

p = lambda x: -(2 \* x + 4) / (x \* (x + 4))

q = lambda x: 2 / (x \* (x + 4))

x = np.arange(a, b + h / 2, h)

A = np.zeros((x.size, x.size))

b = np.zeros(x.size)

A[0][0] = 1 / h

A[0][1] = -1 / h

b[0] = -3

A[-1][-2] = 1 / h

A[-1][-1] = (1 - 1 / h)

b[-1] = 3

for i in range(1, x.size - 1):

A[i][i - 1] = (1 - p(x[i]) \* h / 2) / h\*\*2

A[i][i] = (-2 + q(x[i]) \* h\*\*2) / h\*\*2

A[i][i + 1] = (1 + p(x[i]) \* h / 2) / h\*\*2

y = sweep\_method(A, b)

return x, y

def Runge\_Rombegr\_method(I\_h, I\_kh, k):

p = 1

R = (I\_h - I\_kh) / (k\*\*p - 1)

return R