**Методы численного анализа**

**Отчёт по лабораторной работе**

**“Численное методы решения задачи Коши”**

**Вариант 2**

**Студент**

Малиев Эмиль Енгибарович

**Преподаватель**

Будник Анатолий Михайлович

ФПМИ БГУ  
2022

**Постановка задачи**

Найти приближенное решение задачи Коши:

(1)

*u*(1) = 1*,*

*t* ∈ [0*,*1]

на сетке узлов при 10-ти разбиениях отрезка интегрирования.

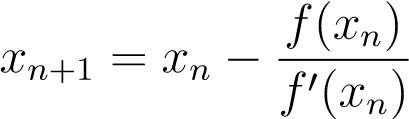
**Алгоритм**

Пункт 1:

**Неявный метод Эйлера, используя метод Ньютона для решения уравнения:**

Представим уравнение (1) в виде

|  |  |
| --- | --- |
| *u*′ = *f*(*t,u*)  Формула для нахождения решения уравнения (2) неявным методом Эйлера: |  |
| *yj*+1 = *yj* + *τf*(*tj*+1*,yj*+1)*, (2)*  *tn* = *nτ*  В нашем случае *τ* = 0*.*1. Для решения (2) используем метод Ньютона: |  |



Если

*то заканчиваем итерационный процесс*

Пункт 2:

**Метод Рунге-Кутта по таблице Бутчера**

Таблица Бутчера в условии имеет следующий вид:

0

0

0

1

1

0

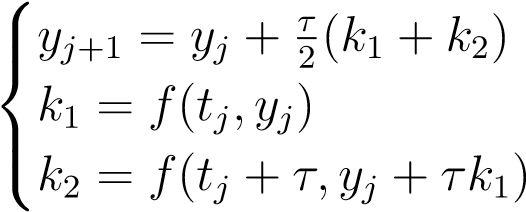
1

2

1

2

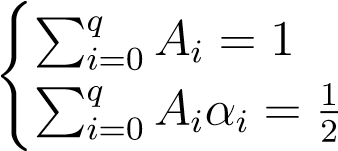
Данная таблица соответствует аналогу формулы трапеции для решения задачи Коши, значит формула для нахождения решения имеют следующий вид:



Пункт 3:

**Метод последовательного повышения порядка точности 2-ого порядка при q = 0**

Чтобы найти параметры *Ai*, требуется решить следующую систему:

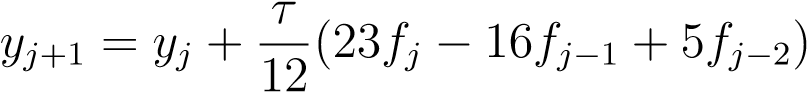


При q = 0 система имеет единственное решение, , что приводит к следующему вычислительному правилу:

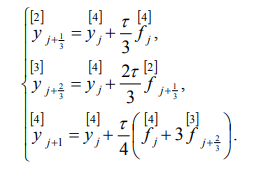
Пункт 4:

**Экстраполяционный метод Адамса 3-ого порядка с началом, построенным по соответствующему методу последовательного повышения порядка точности**

Так как мы имеем равностоящую сетку, соответствующий метод примет следующий вид:

 (8)

Начало таблицы ищем по формуле:



**Листинг**

import math

import sympy as sym

import numpy as np

from sympy import symbols

time, u, x = symbols('time u x')

def newton\_method(f, x0=0., eps=10\*\*-15):

x1 = x0

x2 = x0+1.0

df = sym.diff(f, x)

while np.abs(x1 - x2) >= eps:

x2 = x1

x1 = x1 - (f.subs(x, x1) / df.subs(x, x1))

return x1

def euler\_method(f, u0=1., partitions=10, a=1., b=2.):

len = b-a

tau = len / partitions

print(tau)

t = np.zeros(partitions + 1)

for i in range(partitions + 1):

t[i] = i \* tau + a

y = np.zeros(partitions + 1)

y[0] = u0

for i in range(1, partitions + 1, 1):

equ = -x+y[i - 1]+tau\*f.subs(time, t[i]).subs(u, x)

y[i] = newton\_method(equ)

return np.array(y)

def mpppt2(f, u0=1., partitions=10, a=1., b=2.):

len = b - a

tau = len / partitions

t = np.zeros(partitions + 1)

for i in range(partitions + 1):

t[i] = i \* tau + a

y = np.zeros(partitions + 1)

y[0] = u0

for j in range(partitions):

tmp = y[j] + tau\* 0.5 \* f.subs(time, t[j]).subs(u, y[j])

y[j + 1] = y[j] + f.subs(u, tmp).subs(time, t[j] + 0.5 \* tau) \* tau

return np.array(y)

def mpppt3(f, u0=1., partitions=10, a=1., b=2.):

len = b - a

tau = len / partitions

t = np.zeros(partitions + 1)

for i in range(partitions + 1):

t[i] = i \* tau + a

y = np.zeros(partitions + 1)

y[0] = u0

for j in range(partitions):

tmp1 = y[j] + tau\* (1/3) \* f.subs(time, t[j]).subs(u, y[j])

tmp2 = y[j] + tau\* (2/3) \* f.subs(time, t[j] + (1/3) \* tau ).subs(u, tmp1)

y[j + 1] = y[j] + tau \* 0.25 \* (f.subs(time, t[j]).subs(u, y[j]) + 3\* f.subs(time, t[j] + (2/3) \* tau).subs(u, tmp2))

return np.array(y)

def adams\_bashforth\_third\_order(f, y, partitions=10, a=1., b=2.):

len = b - a

tau = len / partitions

t = np.zeros(partitions + 1)

for i in range(partitions + 1):

t[i] = i \* tau + a

for j in range(2, partitions, 1):

y[j+1] = y[j]+(tau/12) \* \

(23\*f.subs(time, t[j]).subs(u, y[j])

- 16\*f.subs(time, t[j-1]).subs(u, y[j-1])

+ 5\*f.subs(time, t[j-2]).subs(u, y[j-2]))

return np.array(y)

def runge\_kutta\_second\_order(f, alpha=1., u0=1., partitions=10, left=1., right=2.):

len = right-left

tau = len/partitions

t = np.zeros(partitions + 1)

for i in range(partitions + 1):

t[i] = i \* tau + left

y = np.zeros(partitions + 1)

y[0] = u0

for j in range(partitions):

k1 = f.subs(time, t[j]).subs(u, y[j])

k2 = f.subs(time, t[j]+alpha\*tau).subs(u, y[j]+tau\*k1\*alpha)

y[j+1] = y[j]+(tau/(2\*alpha))\*((2\*alpha-1)\*k1+k2)

return np.array(y)

f = ((u \*\* 2) \* sym.log(time) - u) / time

euler = euler\_method(f)

runge\_kutta = runge\_kutta\_second\_order(f)

mpppt = mpppt2(f)

adams = adams\_bashforth\_third\_order(f, mpppt3(f))

print('Explicit Euler Method \n', '\nValues\n', euler,

'Errors\n', euler - adams)

print('Runge-Kutta Method\n', '\nValues\n', runge\_kutta,

'Errors\n', runge\_kutta - adams)

print('Explicit Mpppt2\n', '\nValues\n', mpppt,

'Errors\n', mpppt - adams)

**Результаты**

Значения в точках *xi*, полученные разными методами:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *xi* | М. Эйлера | М. Рунге-Кутта | МПППТ | М. Адамса |
| 0.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 0.1 | 0.92343957 | 0.91260006 | 0.91371744 | 0.91293993 |
| 0.2 | 0.86284725 | 0.84517763 | 0.84688263 | 0.84573235 |
| 0.3 | 0.81362099 | 0.79140086 | 0.79343605 | 0.79084312 |
| 0.4 | 0.77277526 | 0.74737773 | 0.74960818 | 0.74634581 |
| 0.5 | 0.73828978 | 0.71058106 | 0.71293193 | 0.70927159 |
| 0.6 | 0.70874889 | 0.67929648 | 0.68172439 | 0.67784829 |
| 0.7 | 0.68313102 | 0.65231855 | 0.65479758 | 0.65079816 |
| 0.8 | 0.66068033 | 0.62877412 | 0.63128836 | 0.62721733 |
| 0.9 | 0.64082541 | 0.60801451 | 0.61055409 | 0.60644046 |
| 1.0 | 0.62312608 | 0.58954735 | 0.59210605 | 0.58796635 |

Погрешности в точках *xi* при нахождении значений разными методами(за «точные» результаты были взяты значения, полученные методом Адамса):

|  |  |  |  |
| --- | --- | --- | --- |
| *xi* | М. Эйлера | М. Рунге-Кутта | МПППТ |
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.1 | 0.01049964 | -0.00033987 | 0.00077751 |
| 0.2 | 0.01711489 | -0.00055472 | 0.00115027 |
| 0.3 | 0.02277788 | 0.00055774 | 0.00259293 |
| 0.4 | 0.02642945 | 0.00103192 | 0.00326237 |
| 0.5 | 0.02901819 | 0.00130947 | 0.00366033 |
| 0.6 | 0.0309006 | 0.00144819 | 0.0038761 |
| 0.7 | 0.03233286 | 0.0015204 | 0.00399942 |
| 0.8 | 0.033463 | 0.00155678 | 0.00407103 |
| 0.9 | 0.03438495 | 0.00157405 | 0.00411362 |
| 1.0 | 0.03515973 | 0.001581 | 0.00413971 |

Порядок локальной погрешности для разных методов:

Метод Эйлера - *O*(*τ*2)

Метод Рунге-Кутта - *O*(*τ*3)

Метод последовательного повышения порядка точности - *O*(*τ*3)

Метод Адамса - *O*(*τ*4)