РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ГРУППОВОМУ ПРОЕКТУ

ТЕМА: “Метод Рунге-Кутта 4ого порядка.

Студент: Хиссен Али Уэддей

Группа: НПМнп-02-20

МОСКВА

2021 г.

Содержание

1 Математическая постановка задачи

1.1Метод Рунге-Кутта 4ого порядка

Метод рунге-кутта- большой класс численных методов решения задачи Коши для обыкновенных дифференциальных уравнений и их систем. Первые методы данного класса были предложены около 1900 года немецкими математиками К. Рунге и М. В. Куттой.

Метод Рунге — Кутты четвёртого порядка при вычислениях с постоянным шагом интегрирования столь широко распространён, что его часто называют просто методом Рунге — Кутты.

Рассмотрим задачу Коши для системы обыкновенных дифференциальных уравнений первого порядка.

Учитывая следующие входные данные,

- *Обычное дифференциальное уравнение, которое определяет значение* $\frac{dy}{dx}$ *в виде x и y.*
- *Начальное значение y,*

Таким образом, мы приведены ниже.

$$\begin{cases} \frac{dy}{dx} = f(x, y) \\ y(a) = y_0 \end{cases} \qquad x \epsilon [a, b]$$

Задача состоит в том, чтобы найти значение неизвестной функции y в заданной точке x.

Метод Рунге-Кутты находит приблизительное значение y для данного x. Только обыкновенные дифференциальные уравнения первого порядка могут быть решены с помощью метода 4-го порядка Рунге Кутты.

Ниже приведена формула, используемая для вычисления следующего значения y_{n+1} из предыдущего значения y_n. Значения $n = 0,1,2 \ldots \frac{(x - x0)}{h}$. Здесь h — высота шага, а $x_{n+1} = x_n + h$, меньший размер шага означает большую точность.

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k1}{2})$$

$$k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k2}{2})$$

$$k_4 = hf(x_n + h, y_n + k_3)$$

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

Формула в основном вычисляет следующее значение y_{n+1}, используя текущее значение y_n плюс средневзвешенное значение четырех приращений.

1.2.Результаты выполнения программы.

Используя метод, Рунге-Кутты порядка m =4, найти численные решения задачи Коши (шаг сетки h=0, 05):

$$\begin{cases} \dfrac{dy}{dx} = f(x, y) \\ y(a) = y_0 \end{cases} \qquad x\epsilon[a, b]$$

Для начала, найдем точное решение этого линейного уравнения первого порядка

$$y' - 3.x^2y - x^2e^{x^3} = 0, y(0) = 0\ x\epsilon[0,1], h = 0.01$$

$$y' = +3.x^2y + x^2e^{x^3}\ \ , y = uv$$

$$(uv)' - 3.x^2uv = x^2e^{x^3}$$

$$u'y + y'u - 3.x^2uv = x^2e^{x^3}$$

$$u'v + u(v' - 3.x^2v) = x^2e^{x^3}$$

$$\begin{cases} v' - 3.x^2v = 0 \\ u'v = x^2e^{x^3} \end{cases}$$

1) $v' - 3.x^2v = 0$

$$\frac{dv}{dx} = 3.x^2v$$

$$\int \frac{dv}{v} = \int 3.x^2dx$$

$$v = e^{x^3}$$

поставим в второе уравнение системы

2) $u'v = x^2e^{x^3} \quad \rightarrow \quad u'e^{x^3} = x^2e^{x^3}$

$$u' = x^2$$

$$\int dv = \int x^2dx$$

$$u = \frac{x^3}{3} + c$$

Найдем С с помощью ГУ

$$y = uv \ \rightarrow y = e^{x^3}\left(\frac{x^3}{3} + c\right), y(0) = 0$$

C=0

Тогда точное решение имеет вид $y = \frac{x^3}{3}e^{x^3}$

Найдем приблизительное численное решение дифференциального уравнения методом Рунге-Кутты (Рунге-Кутта) 4-го порядка.

Формулы для метода Рунге-Кутты:

$$y_{n+1} = y_n + hk_n$$

$$k_n = \frac{1}{6}(k_n^1 + 2k_n^2 + 3k_n^3 + k_n^4)$$

$$k_n^1 = \ f(x_n, y_n)$$

$$k_n^2 = f(x_n + \frac{h}{2}, y_n + h\frac{k_n^1}{2})$$

$$k_n^3 = f(x_n + \frac{h}{2}, y_n + h\frac{k_n^2}{2})$$

$$k_n^4 = f(x_n + h, y_n + hk_n^3)$$

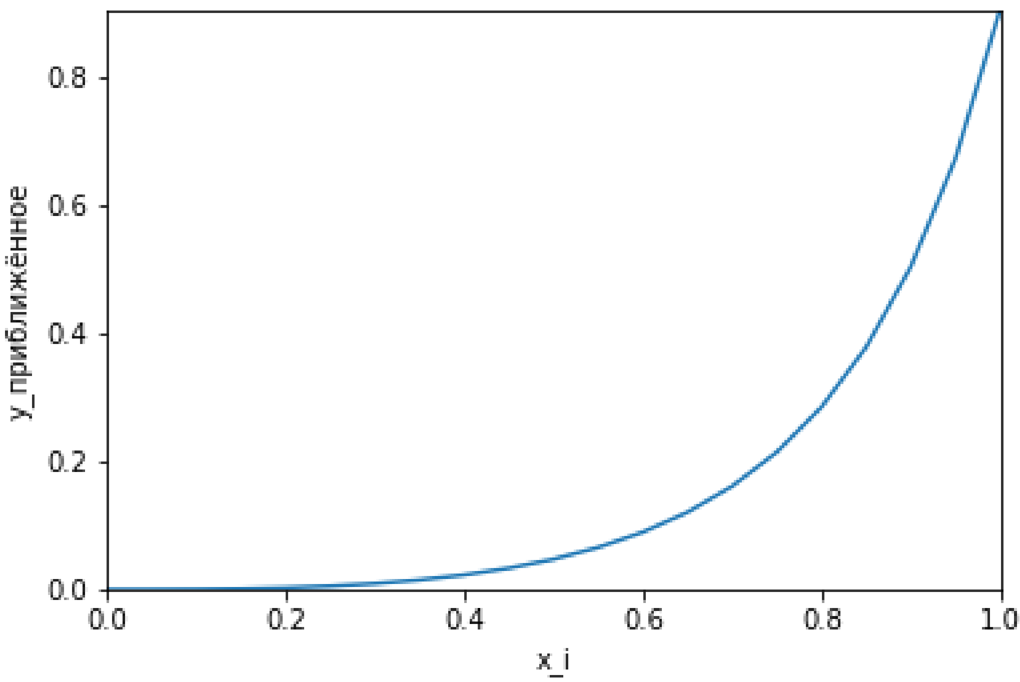
Для выполнения работы было использовано программа python *3*

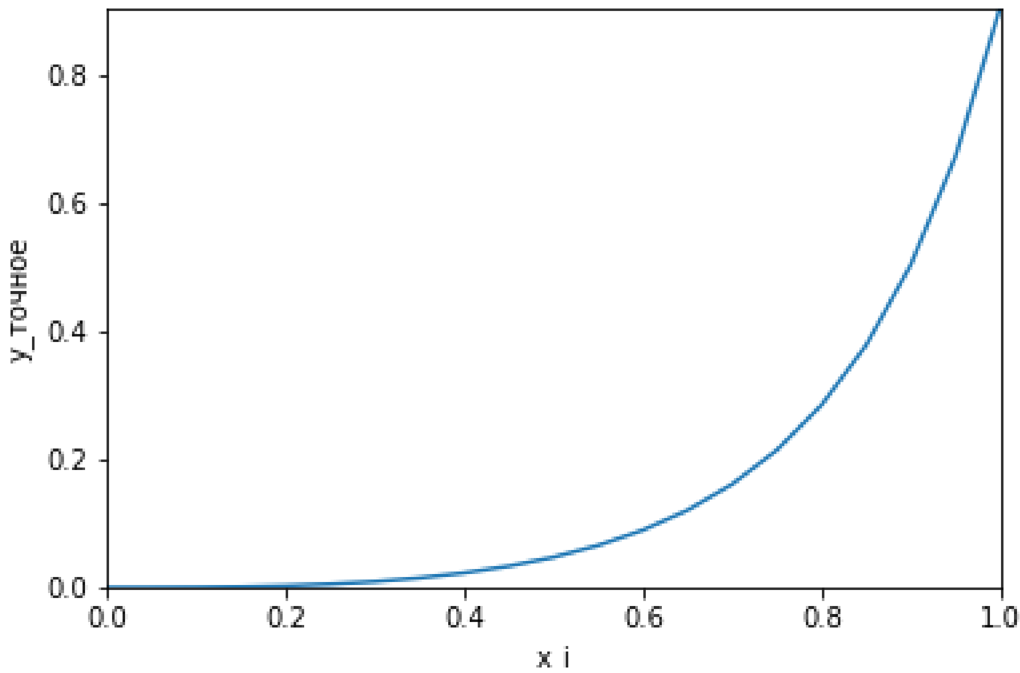
xi	k1	k2	k3	k4	y_приближеньное	н_точное
0.05	0.0	3.125048828506472e-05	3.125195315170308e-05	0.0001250273454590351	4.167203805542845e-05	4.16718753255344e-05
0.1	0.000125031252990874	0.00028145658571746365	0.00028152257765470763	0.0005009850420069194	0.00033366780834578446	0.00033366683338890285
0.15000000000000002	0.000501000751795873	0.0007841465141747921	0.00078447832561508	0.0011325770326428967	0.001128805719015536 9	0.0011288032894407791
0.2	0.0011326130087424562	0.001547265495054191	0.0015482178999836882	0.0020321263127225427	0.0026880900709389966	0.0026880855613447296
0.25	0.0020321927114341815	0.0025883760206827 25	0.002590487779185028	0.003223698256678306	0.005290359832246995	0.005290352648888988 5
0.3	0.0032238087127607084	0.0039390091554497774	0.003943065682960655	0.004747806356891006	0.009246320623325758	0.009246310224871403
0.35	0.0047479804408506006	0.005649802390556395	0.005656946511314221	0.006667167924560244	0.01491776165151777	0.014917747578909546
0.39999999999999997	0.006667434261307873	0.007796981501624528	0.007808894695174743	0.00907417894241266	0.022743322584404285	0.022743304506912105
0.44999999999999996	0.009074578932117742	0.010490978228827844	0.010510166013050465	0.012101059680865528	0.033272977100527595	0.033272954894153516
0.49999999999999994	0.012101651644146364	0.013888312639358 13	0.013918546293386 48	0.015934037790607102	0.047214544983901374	0.04721451887778441
0.5499999999999999	0.015934901100231626	0.01820837194298468	0.01825536884806222	0.020833562822420007	0.0654972025680256	0.06549717343508041
0.6	0.020834801048818824	0.023757483828773383	0.023829957228332567	0.027163509451015436	0.0893594013370333	0.08935937128804834
0.65	0.027165250494211892	0.030963865405938342	0.031075152952180326	0.0354338229484839 4	0.1204722530301888 3	0.12047222663842176
0.7000000000000001	0.03543621210619324	0.040428873758231336	0.04059948236812208	0.04636340721897391	0.16111497495983448	0.1611149617802576
0.7500000000000001	0.046366585326745896	0.05300288880973966	0.0532645043361083	0.06097377229928877	0.21442749894612292	0.2144275186684679
0.8000000000000002	0.06097782395727272	0.06989877767031849	0.07030063875711086	0.08072990474397984	0.2847785925388081	0.2847786854638368
0.8500000000000002	0.08073474840819497	0.09286331029071111	0.09348243547305768	0.10775444265238399	0.37830870630349417	0.3783089545357221
0.9000000000000002	0.10775960978723922	0.12443895240985575	0.1253967115370138	0.145156974120192	0.5037400249370225	0.5037405951154218
0.9500000000000003	0.14516117888241853	0.16836827923581218	0.16985752237880258	0.19754633758099016	0.6735998782191288	0.6736011075496987
1.0000000000000002	0.1975466531270249	0.23022622848285118	0.2325561800820486	0.2718375001681291	0.9060913732899544	0.9060939428196829

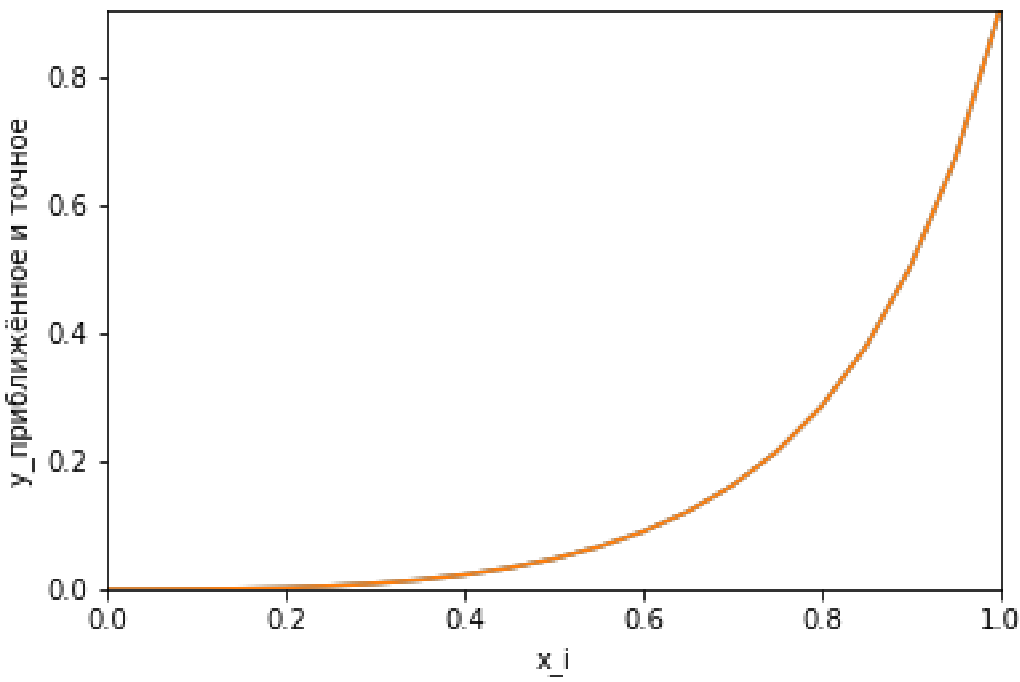
Не сложно заметить, что при использовании метом Рунге Кутта,

Приближенное и точное решение почти равны.

Сравним графики приближенного и точного решения.


0.8
0.6
0.4
0.2
0.0
y_приближённое
0.0
0.2
0.4
0.6
0.8
1.0
x_i


0.8
0.6
0.4
0.2
0.0
y_точное
0.0
0.2
0.4
0.6
0.8
1.0
x_i



y_приближённое и точное
0.0
0.2
0.4
0.6
0.8
0.0
0.2
0.4
0.6
0.8
1.0
x_i

Список пользователей

- https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%A0%D1%83%D0%BD%D0%B3%D0%B5_%E2%80%94_%D0%9A%D1%83%D1%82%D1%82%D1%8B
- https://matplotlib.org/2.1.1/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py

Предложение

```
import numpy as np
import random
from math import *
import matplotlib.pyplot as plt

# Программа Python для реализации метода Рунге Кутты
# Пример дифференциального уравнения

def dydx(x, y):

    return 3*x**2*y+x**2*exp(x**3)

def totchnoe(x):
    resultat=np.zeros(len(x))

    for i in range(len(x)):

        resultat[i]=x[i]**3*exp(x[i]**3)/3

    return resultat

def fonctionSin(x,y):
    N=13
    n=61
    return ((3*N+7)/(N+n-60+1))*np.sin(((2*N+5)/(N*sqrt(x)))*y)
# Находит значение y для заданного x, используя размер шага h
# и начальное значение y0 при x0.

def rungeKutta(x0, y0, x, h):

    # Подсчитать количество итераций, используя размер шага или

    # высота шага h
    f1=open(r'C:\Users\HISSEINI\Desktop\anaconda et vs\analyseDonnée\x.txt','w')
    f=open(r'C:\Users\HISSEINI\Desktop\anaconda et vs\analyseDonnée\y.txt','w')
    f2=open(r'C:\Users\HISSEINI\Desktop\anaconda et vs\analyseDonnée\kutta.txt','w')
    f3=open(r'C:\Users\HISSEINI\Desktop\anaconda et vs\analyseDonnée\k1.txt','w')
    n = (int)((x - x0)/h)
    x_i=np.zeros(n+1)
    y_i=np.zeros(n+1)
    x_i[0]=x0
    y_i[0]=y0
    # Итерировать по количеству итераций
    y = y0
    con=[]
    for i in range(1, n + 1):


        "Apply Runge Kutta Formulas to find next value of y"

        k1 = h * dydx(x0, y)

        k2 = h * dydx(x0 + 0.5 * h, y + 0.5 * k1)

        k3 = h * dydx(x0 + 0.5 * h, y + 0.5 * k2)

        k4 = h * dydx(x0 + h, y + k3)

        y = y + (1.0 / 6.0)*(k1 + 2 * k2 + 2 * k3 + k4)
```

```python
        # Обновить следующее значение x

        x0 = x0 + h

        x_i[i]=x0

        y_i[i]=y
        f1.write((str(x0)+'\n'))
        f.write((str(y)+'\n'))
        f3.write((str(k1)+'\n'))
        f2.write( str(k2)+ "   " + str(k3)+ "   " +str(k4)+'\n')
        con.append([(str(k1) + "   " + str(k2)+ "   " + str(k3)+ "   " +str(k4))])
    f1.close()
    f.close()
    f2.close()
    f3.close()
    return  x_i,y_i,con


# Метод драйвера

x0 = 0

y = 0

x = 1

h = 0.05

x_i,y_i,conteneur=rungeKutta(x0, y,x,h)

y_totchnoe=totchnoe(x_i)
plt.plot(x_i,y_i,label="график приближённоя решения");
plt.xlim(x_i[0],x_i[-1]);
plt.ylim(y_i[0],y_i[-1]);
plt.xlabel("x_i");
plt.ylabel("y_приближённое");
plt.savefig("приближённое")
plt.legend();

plt.plot(x_i,y_totchnoe,label="график приближённоя точное");
plt.xlim(x_i[0],x_i[-1]);
plt.ylim(y_totchnoe[0],y_totchnoe[-1]);
plt.xlabel("x_i");
plt.ylabel("y_точное");
plt.savefig("точное")
plt.legend();


plt.plot(x_i,y_totchnoe,label="график точного решения");
plt.plot(x_i,y_i,label="график приближённоя решения");
plt.xlim(x_i[0],x_i[-1]);
plt.ylim(y_i[0],y_i[-1]);
plt.xlabel("x_i");
plt.ylabel("y_приближённое и точное");
plt.savefig("оба")
plt.legend();

f4=open(r'C:\Users\HISSEINI\Desktop\anaconda et vs\analyseDonnée\totchnoe.txt','w')
for i in range(1,len(y_totchnoe)):
    f4.write((str(y_totchnoe[i])+'\n'))
f4.close()
```