МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
**«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»**
(ННГУ)

Институт информационных технологий, математики и механики

Кафедра теоретической, компьютерной и экспериментальной механики

Направление подготовки: 01.03.03 «Механика и математическое моделирование»

Профиль подготовки: «Математическое моделирование и компьютерный инжиниринг»

ОТЧЕТ

по лабораторной работе по дисциплине Численные методы

на тему:

«Решение задачи Коши для обыкновенных дифференциальных уравнений и систем»

Выполнил: студент группы 381904
Калинин Евгений Валерьевич

Принял:

преподаватель кафедры ТКЭМ ИИТММ
к.ф.-м.н. Петров Андрей Николаевич

Нижний Новгород

2021

Содержание

1. Метод решения дифференциальных уравнений

Необходимо найти решение дифференциального уравнения

$$\frac{dy}{dx} \equiv y' = f(x, y)\,,$$

которое удовлетворяет условиям

$$y(0) = y_0,$$
$$y(x) \in R^n.$$

Для объяснения метода рассмотрим функцию y, разложенную в ряд Тейлора.

$$y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{1}{2}h^2 y''(x_0) + \cdots$$

Чтобы удержать в ряде Тейлора член n-го порядка, необходимо вычислить n-ю производную зависимой переменной. Для получения второй производной в конечно-разностной форме достаточно было знать наклон кривой на концах рассматриваемого интервала. Чтобы вычислить третью производную в конечно-разностном виде, необходимо иметь значения второй производной, по меньшей мере, в двух точках. Для этого необходимо дополнительно определить наклон кривой в некоторой промежуточной точке интервала h. Очевидно, чем выше порядок вычисляемой производной, тем больше дополнительных точек потребуется вычислить внутри интервала. Так как существует несколько способов расположения внутренних точек и выбора относительных весов для найденных производных, то метод Рунге-Кутта, в сущности, объединяет целое семейство методов решения дифференциальных уравнений.

Семейство явных методов Рунге-Кутта задаётся следующими формулами:

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i\,,$$

где

$$k_1 = f(x_n, y_n),$$
$$k_2 = f(x_n + c_2 h, y_n + a_{21} h k_1),$$
...
$$k_s = f\left(x_n + c_s h, y_n + a_{s1} h k_1 + a_{s2} h k_2 + \ldots + a_{s,s-1} h k_{s-1}\right)$$

Конкретный метод определяется числом s и коэффициентами b_i и a_{ij}. Эти коэффициенты часто упорядочивают в таблицу, называемую таблицей Бутчера:

0					
$\boldsymbol{c_2}$	a_{21}				
$\boldsymbol{c_3}$	a_{31}	a_{32}			
$\vdots$			$\ddots$		
$\boldsymbol{c_s}$	a_{s1}	a_{s2}	$\cdots$	a_{ss-1}	
	b_1	b_2	$\cdots$	b_{s-1}	b_s

Для коэффициентов метода Рунге-Кутта должны быть выполнены условия

$$\sum_{j=1}^{i-1} a_{ij} = \mathrm{c}_i \text{ для } i = 2, \ldots, s.$$

Если требуется, чтобы метод имел порядок p, то следует также обеспечить условие

$$\bar{y}(h + x_0) - y(h + x_0) = O(h^{p+1}),$$

где $\bar{y}(h + x_0)$ — приближение, полученное по методу Рунге-Кутта. После многократного дифференцирования это условие преобразуется в систему полиномиальных уравнений относительно коэффициентов метода.

2. Метод решения систем дифференциальных уравнений

Необходимо решить дифференциальное уравнение второго порядка вида

$$\frac{d^2y}{dt^2} = g(t, y, \frac{dy}{dt})$$

Для этого введем замену

$$z = \frac{dy}{dt}$$

Уравнение можно будет представить в виде системы

$$\begin{cases} \frac{dy}{dt} = \text{z}, \\ \frac{dz}{dt} = g(t, y, z), \end{cases}$$

Начальные условия задачи Коши записываются в виде

$$y(t_0) = y_0$$
$$z(t_0) = z_0$$

Данную систему можно решить методом Рунге-Кутта, использую следующие формулы

$$\begin{cases} y_{n+1} = y_n + h(k_1 + 3k_2 + 3k_3 + k_4)/8 \\ z_{n+1} = z_n + h(q_1 + 3q_2 + 3q_3 + q_4)/8 \end{cases}$$

где

$$\begin{cases} k_1 = f_1(x_n,\ y_n, z_n) \\ q_1 = f_1(x_n,\ y_n, z_n) \\ k_2 = f_1(x_n + h/3\,, y_n + hk_1/3\,, \text{z} + hq_1/3) \\ q_2 = f_1(x_n + h/3\,, y_n + hk_1/3\,, \text{z} + hq_1/3) \\ k_3 = f_1(x_n + 2\,h/3\,, y_n - \ hk_1/3 + \ hk_2\,, \text{z} - \ hq_1/3 + \ hq_2) \\ q_3 = f_1(x_n + 2\,h/3\,, y_n - \ hk_1/3 + \ hk_2\,, \text{z} - \ hq_1/3 + \ hq_2) \\ k_4 = f_1(x_n + h, y_n + hk_1 - hk_2 + hk_3, \text{z} + hq_1 - hq_2 + hq_3) \\ q_4 = f_1(x_n + h, y_n + hk_1 - hk_2 + hk_3, \text{z} + hq_1 - hq_2 + hq_3) \end{cases}$$

3. Общая характеристика одношаговых методов

Всем одношаговым методам присущи определенные общие черты:

1) чтобы получить информацию в новой точке, надо иметь данные лишь в одной предыдущей точке;

2) в основе всех одношаговых методов лежит разложение функции в ряд Тейлора, в котором сохраняются члены, содержащие h в степени до k включительно. Целое число k называется порядком метода;

3) все одношаговые методы не требуют действительного вычисления производных, а вычисляется лишь сама функция в одной или нескольких промежуточных точках;

4) свойство «самостартования» позволяет легко менять величину шага.

4. Постановка задачи

Требуется решить дифференциальное уравнение первого порядка

$$y' = \frac{2y}{xln(x)} + \frac{1}{x}, \qquad x\epsilon[2,3]$$
$$y(2) = -\ln(2)$$

И систему двух дифференциальных уравнений первого порядка

$$\begin{cases} y' = z^2 + x \\ z' = xy \end{cases}, \qquad x\epsilon[0,1]$$
$$y(0) = 1$$
$$z(0) = 0.5$$

Для решения использовать метод Рунге-Кутта 2, 3 и 4-го порядков.

5. Метод решения поставленной задачи

Для проведения подсчетов была написана программа на языке программирования Python 3.9. В ней использовались библиотеки сторонних авторов, имеющие свободное распространение.

Непосредственно вычисления происходят в специальном модуле, который содержит функции, выполняющие подсчеты методом Рунге-Кутта 2, 3 и 4-го порядков.

Для задачи по нахождению решения одиночного ЛДУ они имеют следующий интерфейс:

```
RungeKutta2(f: lambda_type, x0: list, h: float, n: int) -> list
```

В качестве аргументов функция принимает функцию из правой части дифференциального уравнения, значения x_0 и $y(x_0)$ задачи Коши, шаг и число отрезков, в узлах которых требуется вычислить численные значения решения задачи.

Для вычислений использовались следующие формулы:

2-го порядка точности:

$$k_1 = f(x_n, y_n),$$
$$k_2 = f(x_n + h, y_n + hk_1),$$
$$y_{n+1} = y_n + h(k_1 + k_2)/2,$$

3-го порядка точности:

$$k_1 = f(x_n, y_n),$$
$$k_2 = f(x_n + h/2\,, y_n + hk_1/2),$$
$$k_3 = f(x_n + h, y_n + hk_2/2),$$
$$y_{n+1} = y_n + h(k_1 + 4k_2 + k_3)/6,$$

4-го порядка точности:

$$k_1 = f(x_n, y_n),$$
$$k_2 = f(x_n + h/2\,, y_n + hk_1/2),$$
$$k_3 = f(x_n + h/2\,, y_n + hk_2/2),$$
$$k_4 = f(x_n + h, y_n + k_3),$$
$$y_{n+1} = y_n + h(k_1 + 2k_2 + 2k_3 + k_4)/6.$$

Для решения системы двух дифференциальных уравнений были написаны аналогичные функции, с тем отличаем, что правая часть уравнения задается в виде вектора двух функций, а задача Коши задается вектором из трех элементов (x_0, $y(x_0)$ и $z(x_0)$).

Представим задачу в виде

$$y_1' = f_1(x, y_1, y_2), \qquad y_1(0) = y_{1,0}, \qquad y_1(t) \in R^n,$$
$$y_2' = f_2(x, y_1, y_2), \qquad y_2(0) = y_{2,0}, \qquad y_2(t) \in R^n.$$

Тогда для подсчета используются следующие итерационные формулы:

$$y_{1,n+1} = y_{1,n} + h(k_1 + 3k_2 + 3k_3 + k_4)/8\,,$$
$$y_{2,n+1} = y_{2,n} + h(q_1 + 3q_2 + 3q_3 + q_4)/8,$$

где:

$$k_1 = f_1\big(x_n,\, y_{1,n},\, y_{2,n}\big),$$
$$q_1 = f_2\big(x_n,\, y_{1,n},\, y_{2,n}\big),$$
$$k_2 = f_1\big(x_n + h/3\,,\, y_{1,n} + hk_1/3\,,\, y_{2,n} + hq_1/3\big),$$
$$q_2 = f_2\big(x_n + h/3\,,\, y_{1,n} + hk_1/3\,,\, y_{2,n} + hq_1/3\big),$$
$$k_3 = f_1\big(x_n + 2\,h/3\,,\, y_{1,n} - hk_1/3 + hk_2\,,\, y_{2,n} - hq_1/3 + hq_2\big),$$
$$q_3 = f_2\big(x_n + 2\,h/3\,,\, y_{1,n} - hk_1/3 + hk_2\,,\, y_{2,n} - hq_1/3 + hq_2\big),$$
$$k_4 = f_1\big(x_n + h,\, y_{1,n} + hk_1 - hk_2 + hk_3,\, y_{2,n} + hq_1 - hq_2 + hq_3\big),$$
$$q_4 = f_2\big(x_n + h,\, y_{1,n} + hk_1 - hk_2 + hk_3,\, y_{2,n} + hq_1 - hq_2 + hq_3\big),$$

Полный код функций, выполняющих метод Рунге-Кутта для ОДУ, представлен в Приложении G, для системы двух ОДУ в Приложении I.

Задание входных параметров представлено в Приложении D и Приложении H соответственно.

Код остальной программы для обеих задач идентичен и представлен в Приложении E и F. Кроме того, весь код программы можно посмотреть в репозитории на GitHub (ссылка находится в списке литературы).

6. Анализ полученного решения

A. Решение ОДУ

Подробные таблицы с численным решением можно посмотреть в Приложениях A-C.

Рассмотрим график решения поставленной задачи методом Рунге-Кутта 2-го порядка с различным числом шагов.

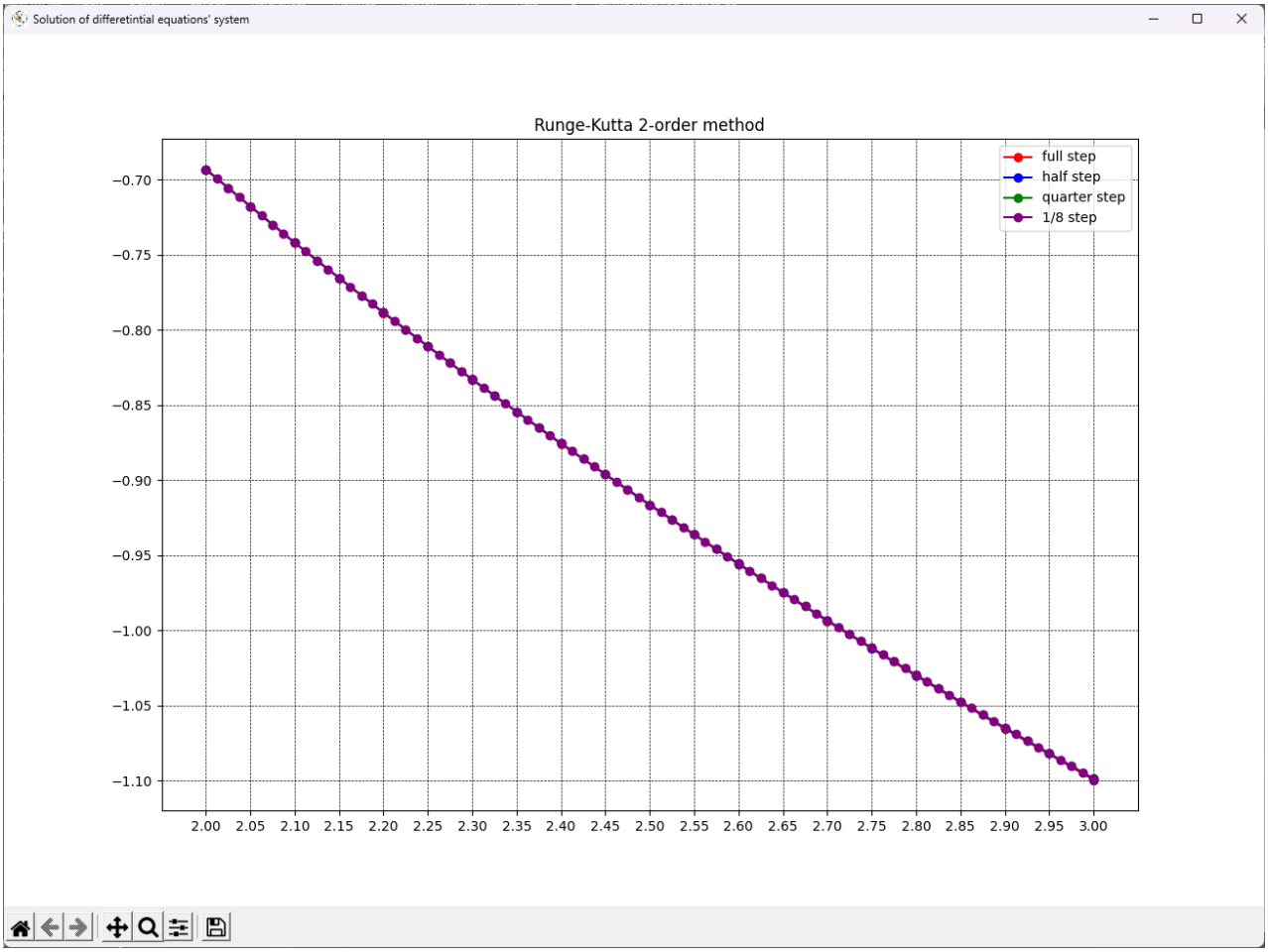

Рис. 1

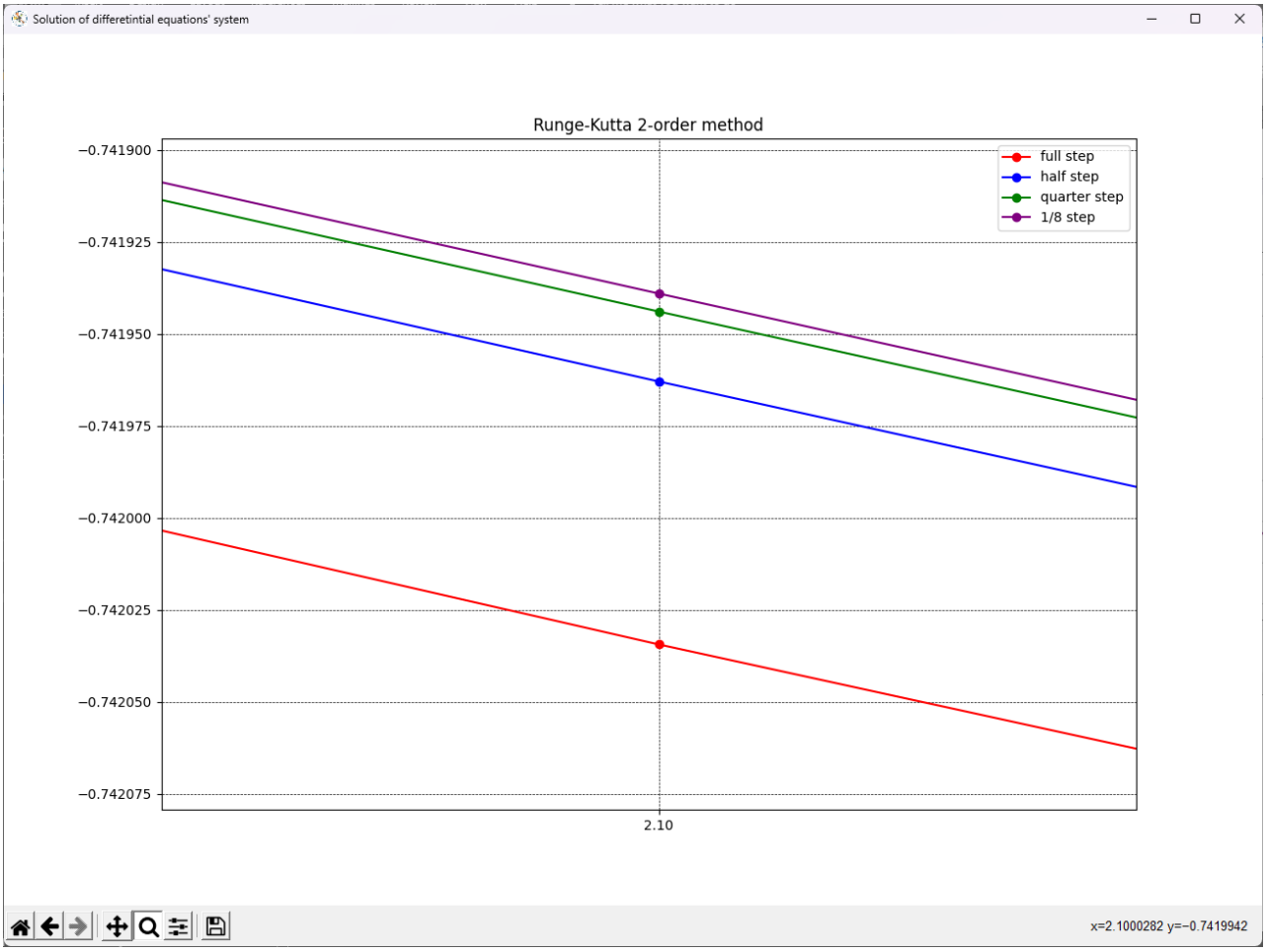

Рис. 2

Как можно заметить по Рис. 1, решение обладает большой точность, поэтому рассмотрим разницу на примере отдельной точки.

По Рис. 2 можем сделать вывод, что при увеличении шага вдвое конечная точность возрастает в 4 раза.

Аналогично можем заметить, что для метода 3-го порядка при уменьшении шага вдвое точность увеличивается в 8 раз (Рис. 4).

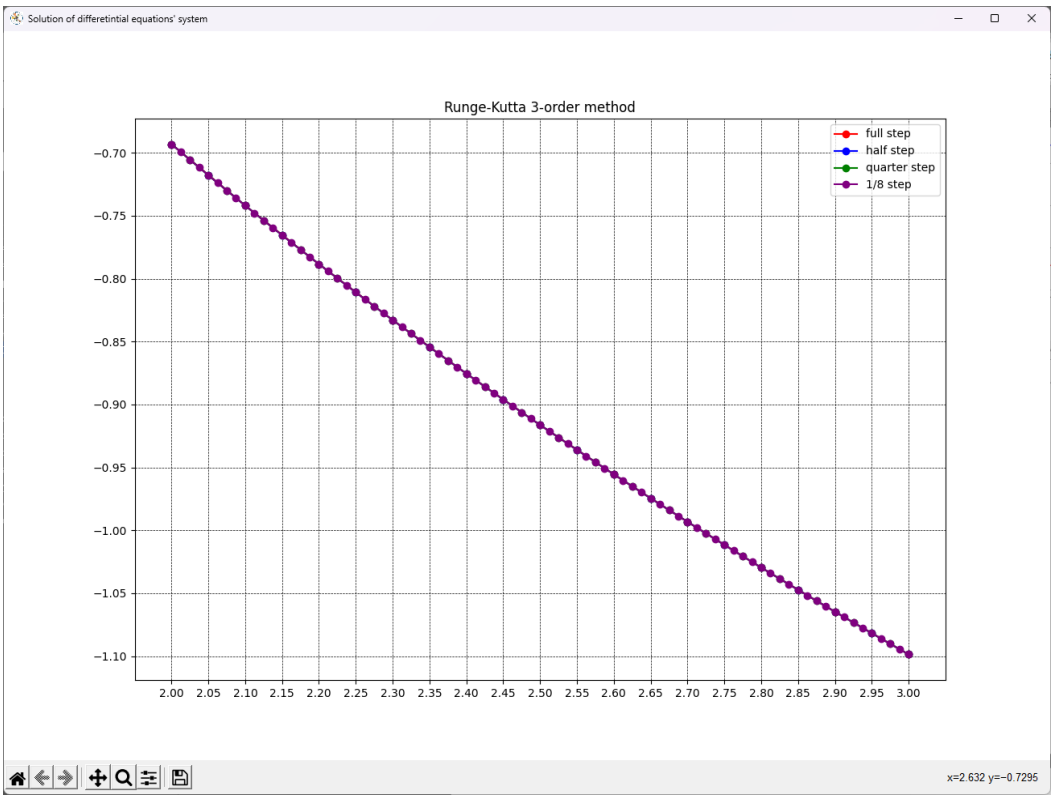
Solution of differetintial equations' system
Runge-Kutta 3-order method
full step
half step
quarter step
1/8 step
−0.70
−0.75
−0.80
−0.85
−0.90
−0.95
−1.00
−1.05
−1.10
2.00 2.05 2.10 2.15 2.20 2.25 2.30 2.35 2.40 2.45 2.50 2.55 2.60 2.65 2.70 2.75 2.80 2.85 2.90 2.95 3.00
x=2.632 y=−0.7295

Рис. 3

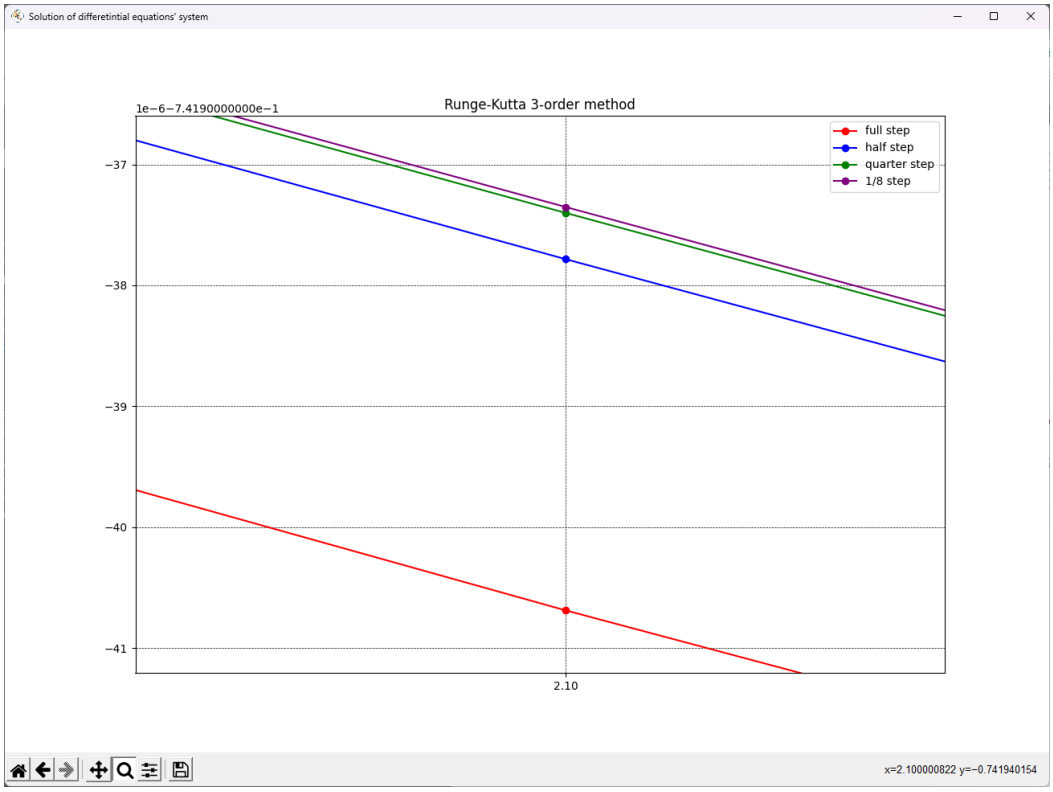
Solution of differetintial equations' system
Runge-Kutta 3-order method
1e−6−7.4190000000e−1
full step
half step
quarter step
1/8 step
−37
−38
−39
−40
−41
2.10
x=2.100000822 y=−0.741940154

Рис. 4

При использовании метода 4-го порядка сгущение сетки в 2 раза дает уменьшение погрешности в 16 раз (Рис. 6).

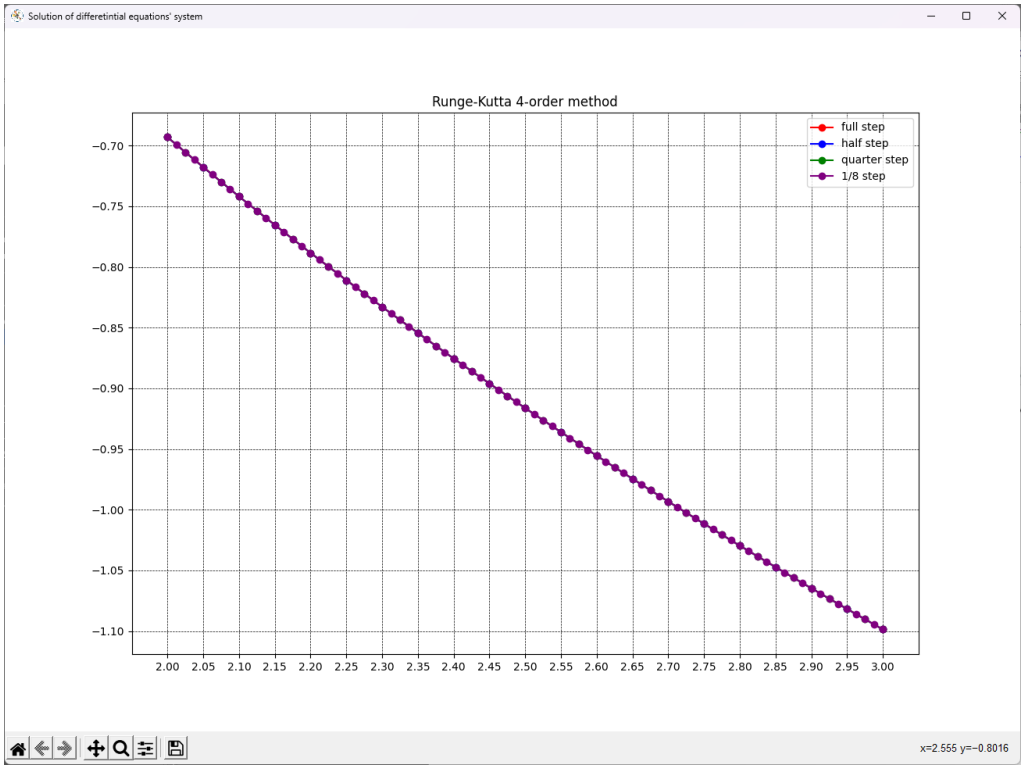

Рис. 5

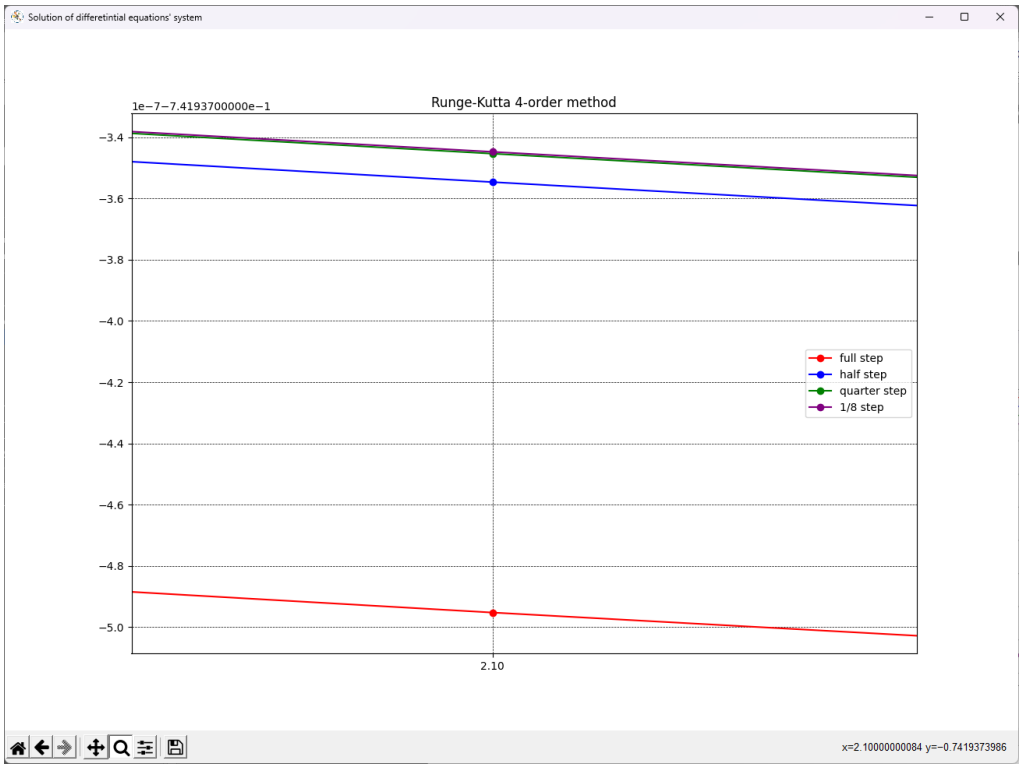

Рис. 6

Сравним точность решения, которую дают методы Рунге-Кутта 2, 3 и 4-го порядка а так же решение, полученное с помощью библиотеки *numpy*, а в частноести ее функции *odeint*.

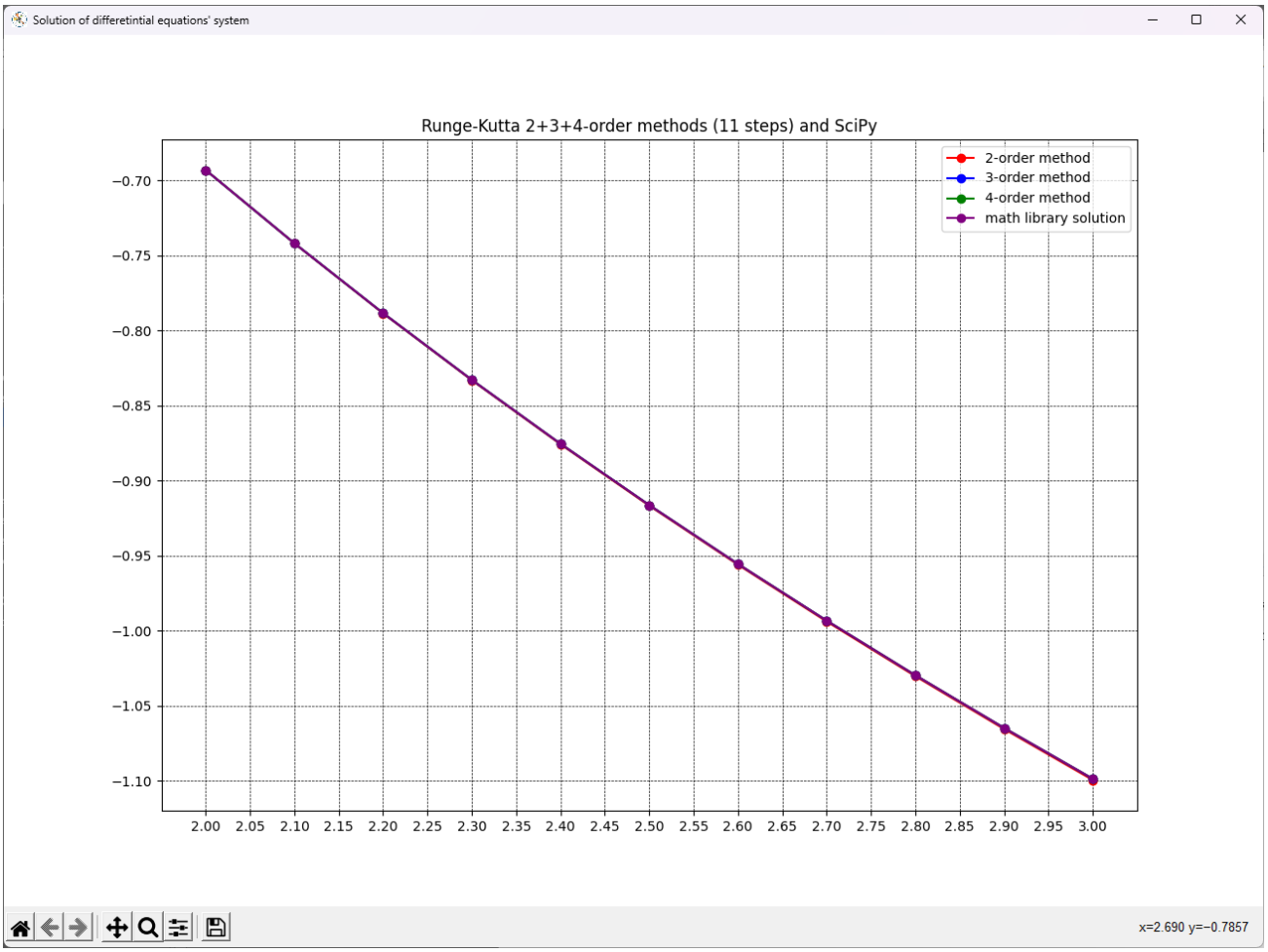

Рис. 7

Как видим по Рис. 8, погрешность решения имеет огромную зависимоть от порядка используемого метода.

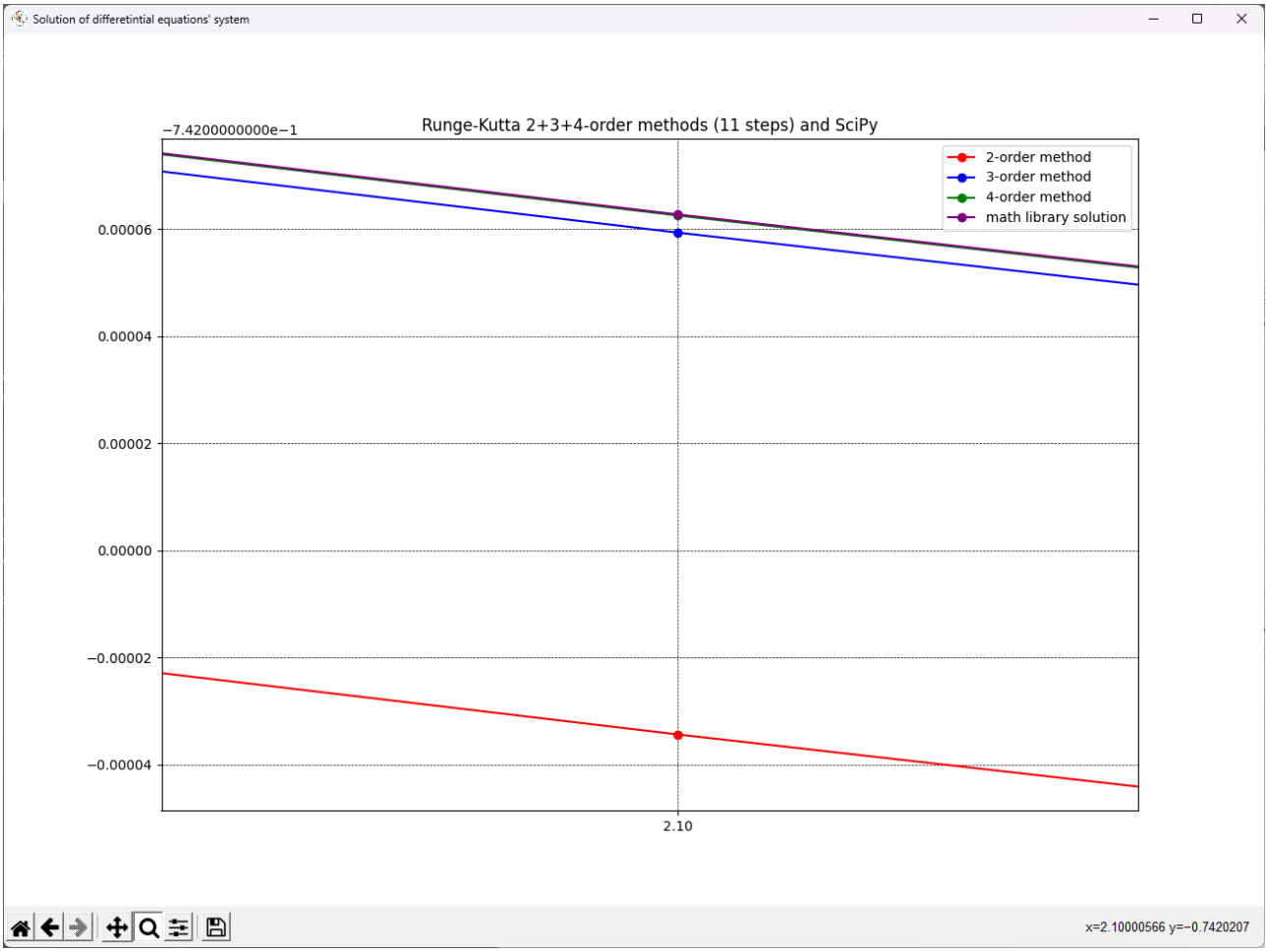
Solution of differetintial equations' system
Runge-Kutta 2+3+4-order methods (11 steps) and SciPy
−7.4200000000e−1
2-order method
3-order method
4-order method
math library solution
0.00006
0.00004
0.00002
0.00000
−0.00002
−0.00004
2.10
x=2.10000566 y=−0.7420207

Рис. 8

B. Решение системы 2-х ОДУ

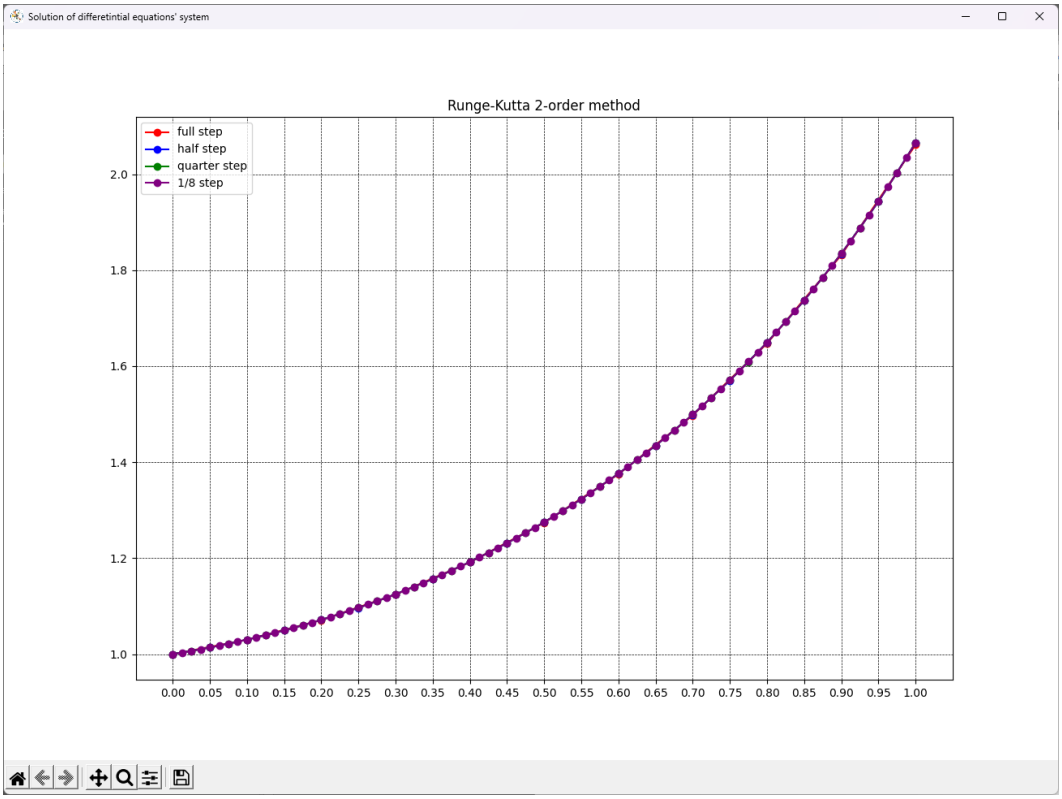

Рис. 9

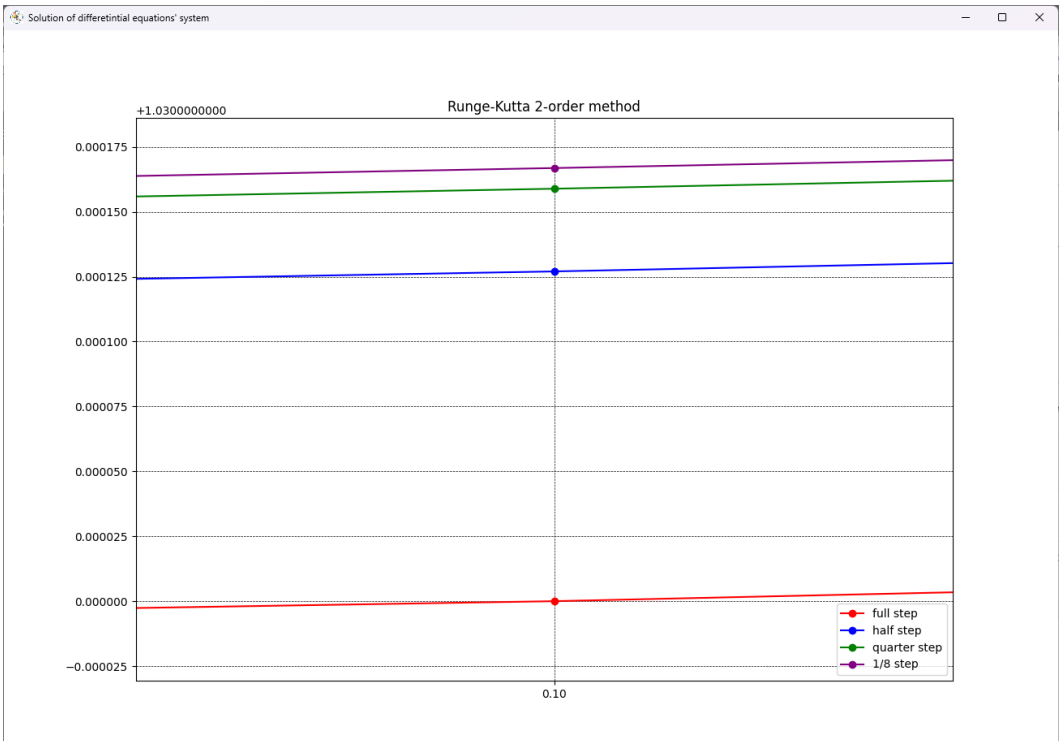

Рис. 10

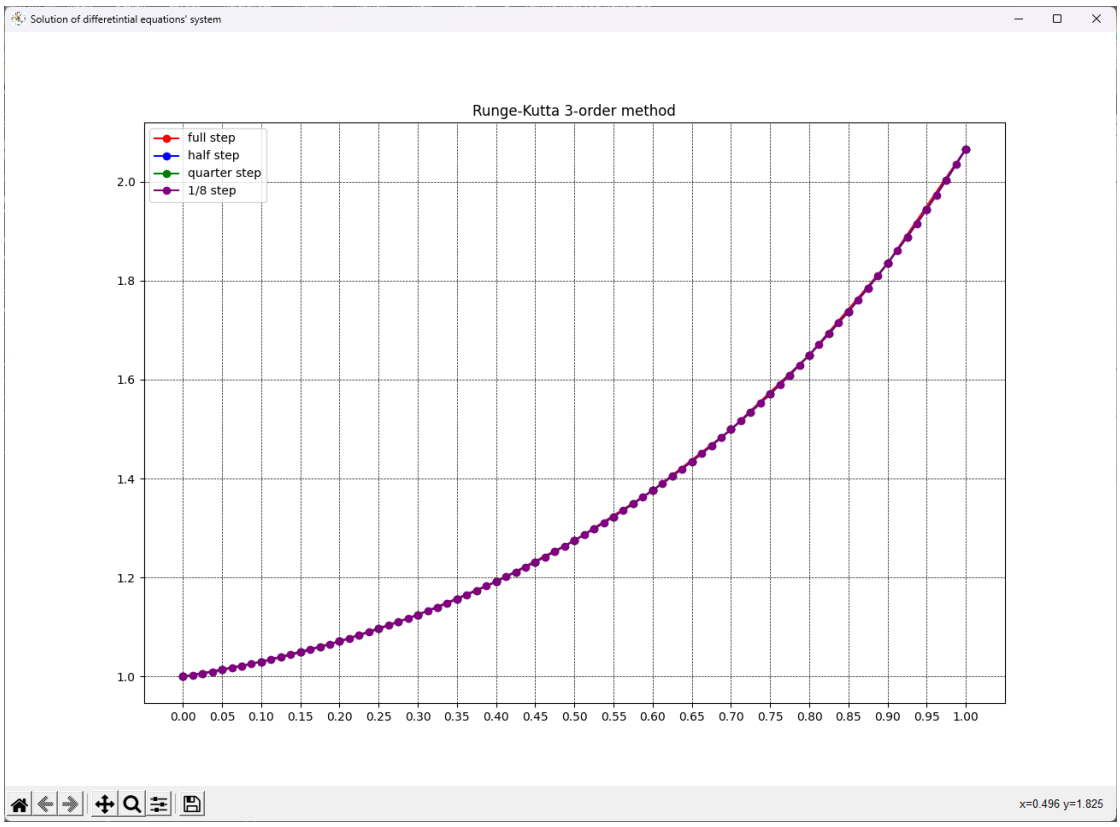
Solution of differetintial equations' system
Runge-Kutta 3-order method
full step
half step
quarter step
1/8 step
2.0
1.8
1.6
1.4
1.2
1.0
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00
x=0.496 y=1.825

Рис. 11

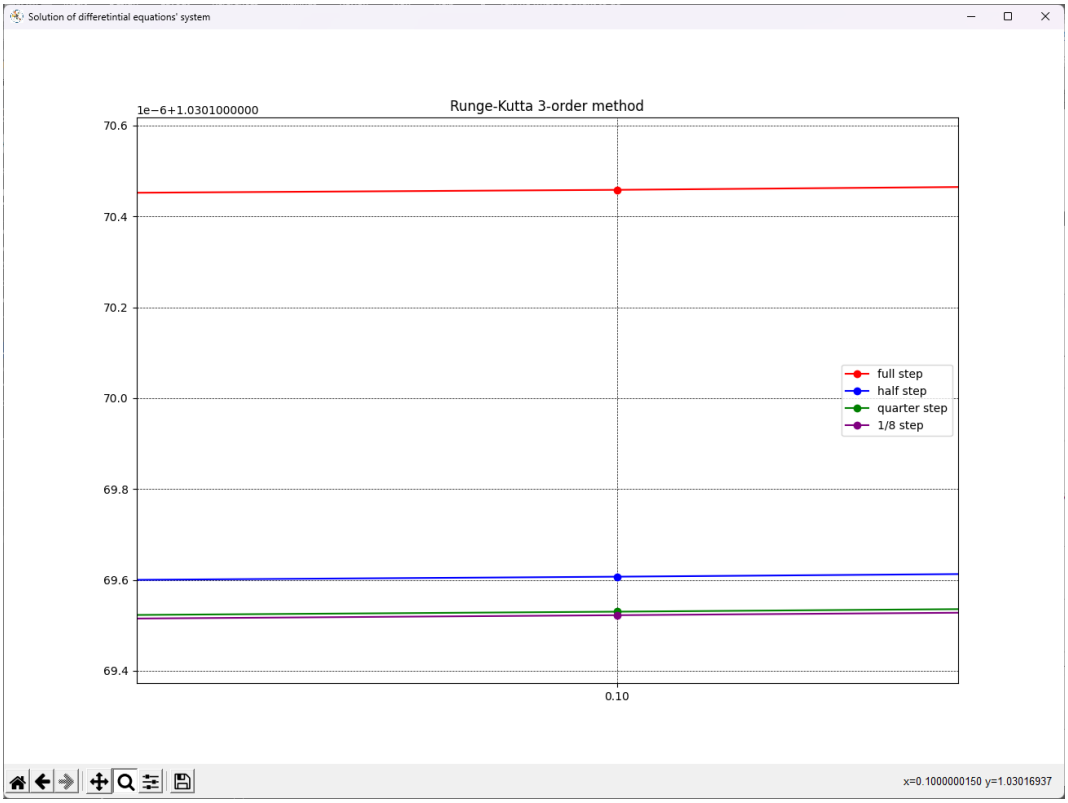
Solution of differetintial equations' system
Runge-Kutta 3-order method
1e−6+1.0301000000
70.6
70.4
70.2
70.0
69.8
69.6
69.4
full step
half step
quarter step
1/8 step
0.10
x=0.1000000150 y=1.03016937

Рис. 12

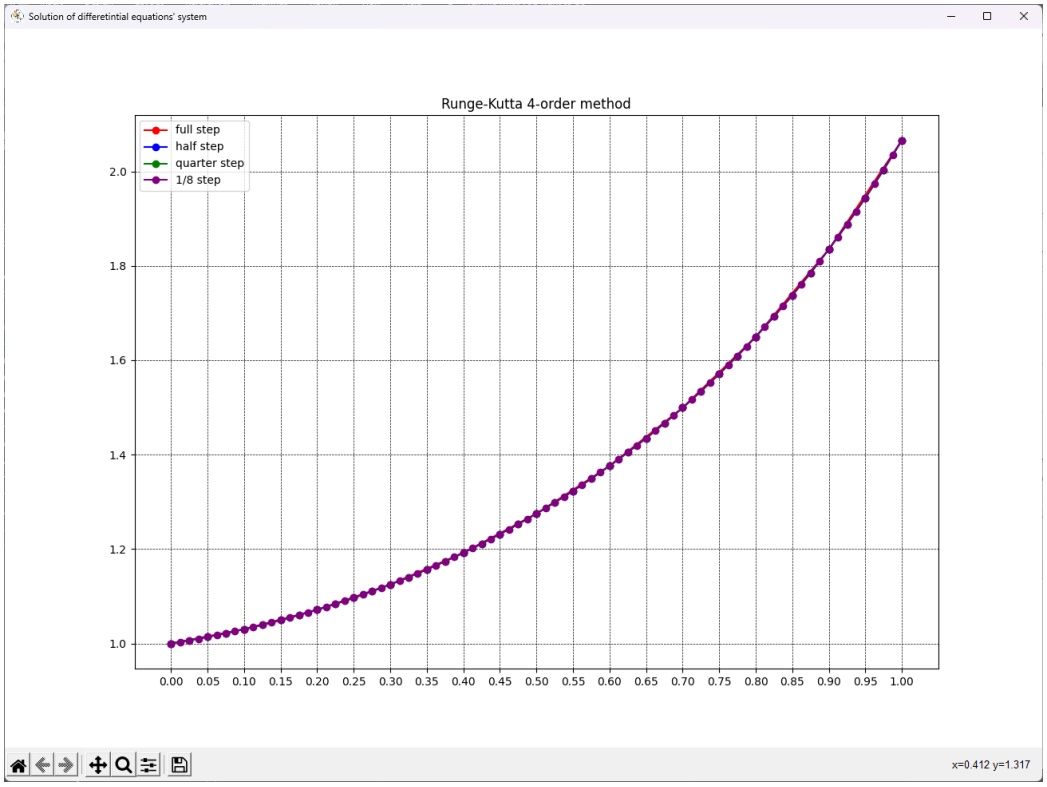
Solution of differetintial equations' system
Runge-Kutta 4-order method
full step
half step
quarter step
1/8 step
2.0
1.8
1.6
1.4
1.2
1.0
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00
x=0.412 y=1.317

Рис. 13

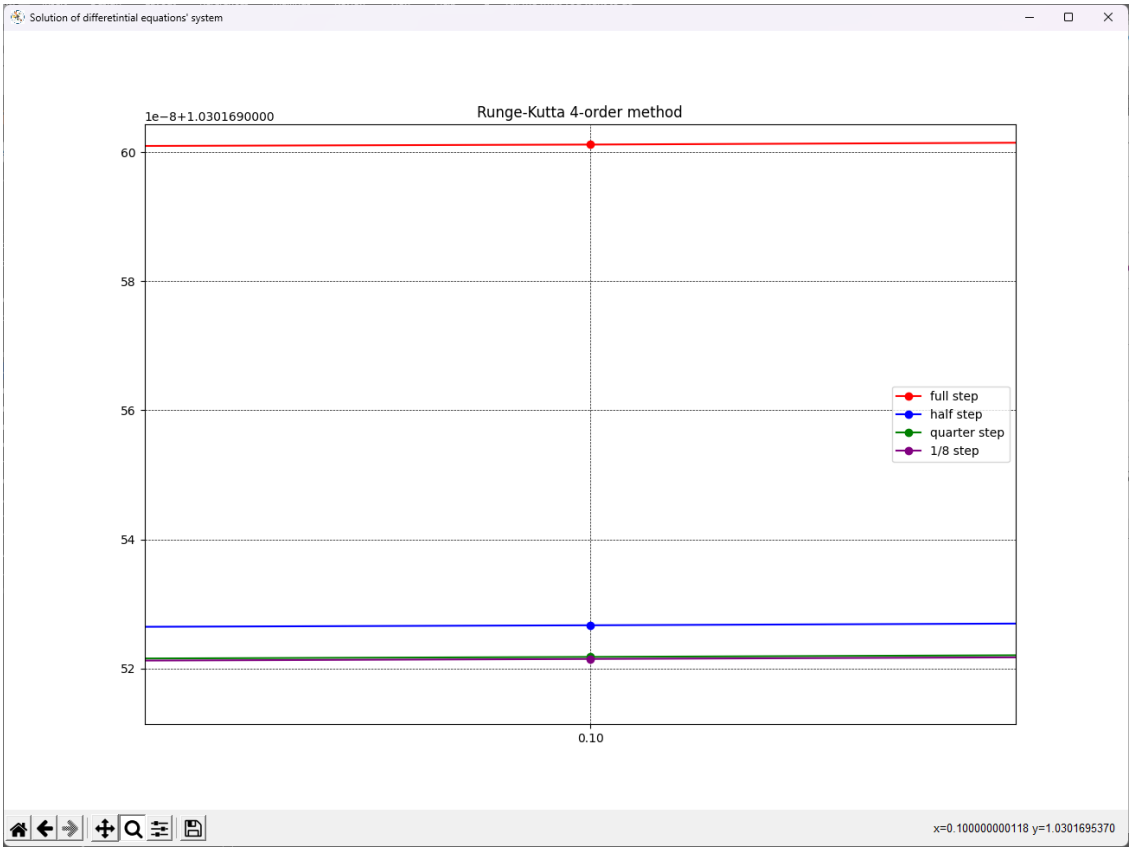
Solution of differetintial equations' system
Runge-Kutta 4-order method
1e−8+1.0301690000
60
58
56
54
52
full step
half step
quarter step
1/8 step
0.10
x=0.100000000118 y=1.0301695370

Рис. 14

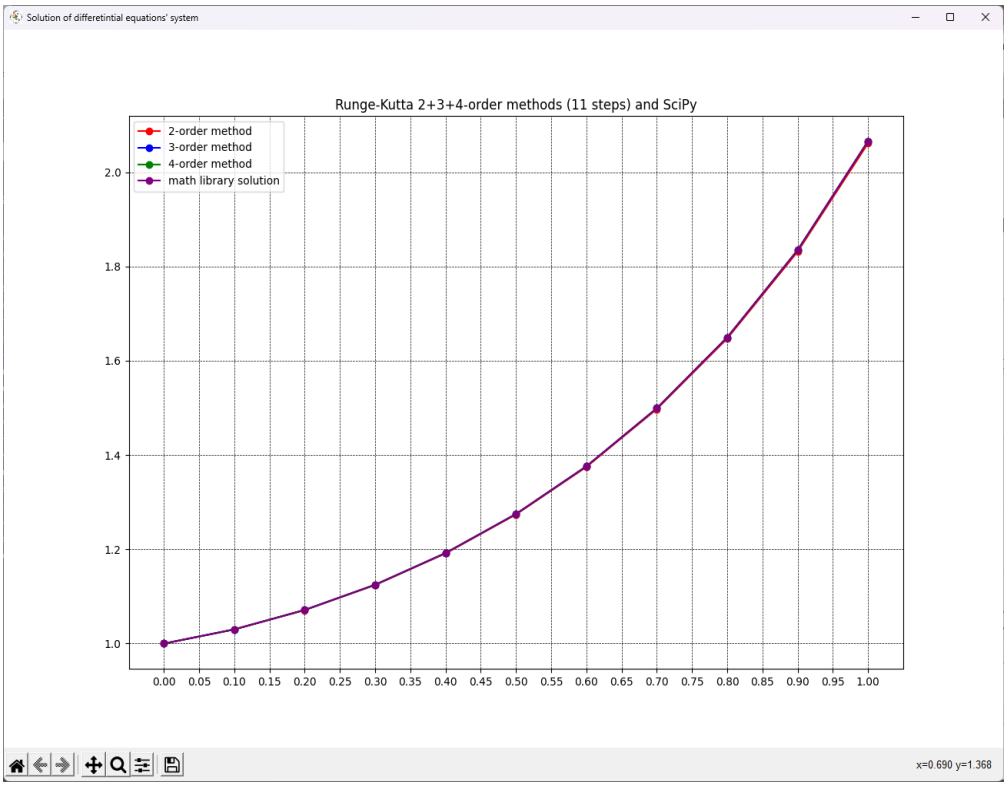
Solution of differetintial equations' system
Runge-Kutta 2+3+4-order methods (11 steps) and SciPy
2-order method
3-order method
4-order method
math library solution
2.0
1.8
1.6
1.4
1.2
1.0
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00
x=0.690 y=1.368

Рис. 15

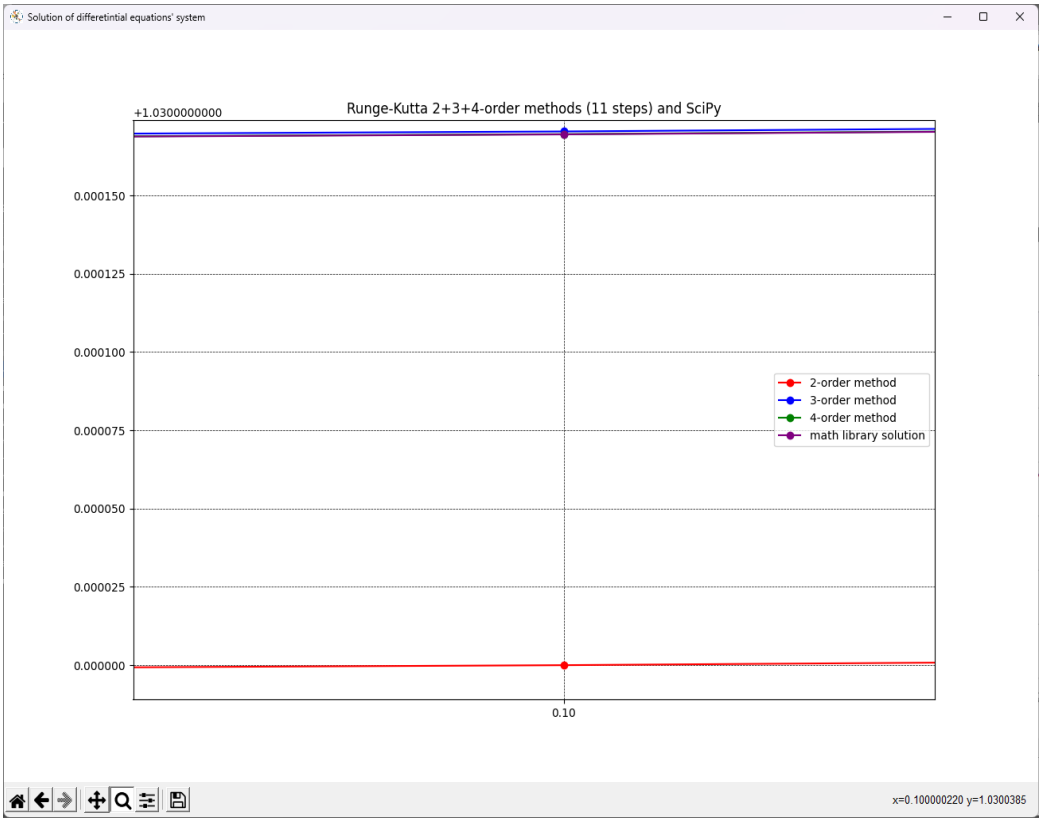
Solution of differetintial equations' system
+1.0300000000
Runge-Kutta 2+3+4-order methods (11 steps) and SciPy
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
0.10
2-order method
3-order method
4-order method
math library solution
x=0.100000220 y=1.0300385

Рис. 16

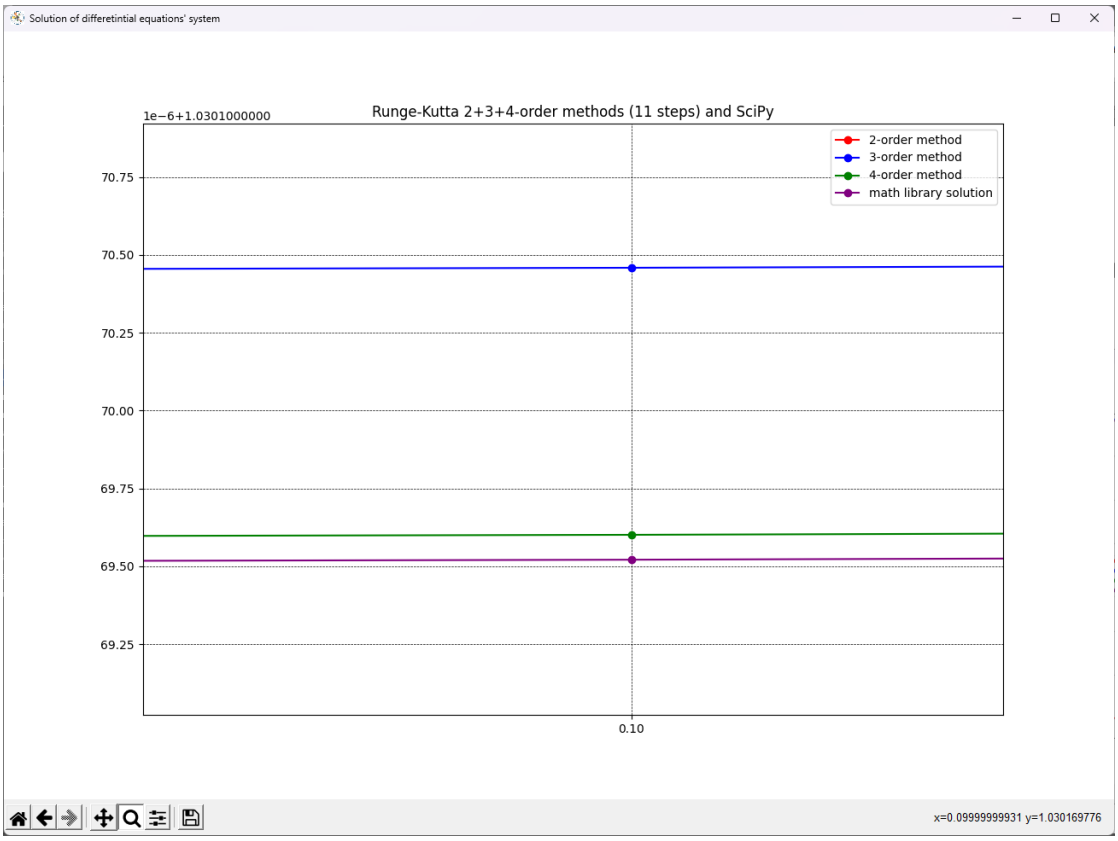

Рис. 17

Как можно заметить по Рис. 9 - Рис. 17, где изображены решения системы 2-х ОДУ (а именно, графики z(y)), наблюдаются все те же закономерности, что были при решении одного уравнения первого порядка.

Таблицы с численными решениям находятся в Приложениях J-M.

7. Заключение

Были произведены численные расчеты задачи Коши для ОДУ первого порядка и системы двух уравнений первого порядка. Для подсчетов на языке программирования Python 3.9 был написан модуль с функциями для математических вычислений, который впоследствии можно будет использовать для других задач, а также тестовая программа, решающая поставленную задачу с использованием этого модуля.

Решение было произведено с использованием метода Рунге-Кутта 2, 3 и 4-го порядков с различными шагами и была продемонстрирована разница в точности используемых методов.

8. Литература

1. Демидович Б.П., М. И. (1963). *Основы вычислительной математики.* М.: Физматгиз.
2. *Репозиторий с представленными программами*. (б.д.). Получено из GitHub: https://github.com/jinekgames/unn-num_meth-6sem/tree/master/basic_dif_eq_koshi_problem
3. Самарский А.А. (2009). *Введение в численные методы: учеб. пособие для вузов.* СПб: Лань.

9. Приложение

A. Численное решение ОДУ методом Рунге-Кутта 2 порядка

```
2-order Runge-Kutta meth for dif. eq.

Koshi:  [2, -0.6931471805599453]
step = 0.1
number of steps: 10
range:  from  2  to  3.0
```

k	x	y
0	2.1	-0.7420344
1	2.2	-0.788647
2	2.3	-0.8331878
3	2.4	-0.8758332
4	2.5	-0.9167381
5	2.6	-0.9560394
6	2.7	-0.9938581
7	2.8	-1.0303022
8	2.9	-1.0654682
9	3.0	-1.0994429

```
2-order Runge-Kutta meth for dif. eq.

Koshi:  [2, -0.6931471805599453]
step = 0.05
number of steps: 20
range:  from  2  to  3.0
```

k	x	y
0	2.05	-0.7178527
1	2.1	-0.7419629
2	2.15	-0.7655057
3	2.2	-0.7885072
4	2.25	-0.8109918
5	2.3	-0.8329822
6	2.35	-0.8544997
7	2.4	-0.8755642
8	2.45	-0.8961944
9	2.5	-0.9164078
10	2.55	-0.936221
11	2.6	-0.9556494
12	2.65	-0.9747079
13	2.7	-0.9934101
14	2.75	-1.0117692
15	2.8	-1.0297976
16	2.85	-1.0475069
17	2.9	-1.0649083
18	2.95	-1.0820123
19	3.0	-1.0988288

2-order Runge-Kutta meth for dif. eq.

Koshi: [2, -0.6931471805599453]
step = 0.025
number of steps: 40
range: from 2 to 3.0

k	x	y
0	2.025	-0.7055714
1	2.05	-0.7178431
2	2.075	-0.7299661
3	2.1	-0.7419439
4	2.125	-0.7537799
5	2.15	-0.7654775
6	2.175	-0.7770399
7	2.2	-0.7884701
8	2.225	-0.7997712
9	2.25	-0.810946
10	2.275	-0.8219973
11	2.3	-0.8329278
12	2.325	-0.8437402
13	2.35	-0.8544369
14	2.375	-0.8650205
15	2.4	-0.8754932
16	2.425	-0.8858573
17	2.45	-0.8961152
18	2.475	-0.906269
19	2.5	-0.9163207
20	2.525	-0.9262723
21	2.55	-0.936126
22	2.575	-0.9458835
23	2.6	-0.9555467
24	2.625	-0.9651175
25	2.65	-0.9745975
26	2.675	-0.9839886

2-order Runge-Kutta meth for dif. eq.

Koshi: [2, -0.6931471805599453]
step = 0.0125
number of steps: 80
range: from 2 to 3.0

k	x	y
0	2.0125	-0.6993779
1	2.025	-0.7055701
2	2.0375	-0.7117242
3	2.05	-0.7178406
4	2.0625	-0.7239199
5	2.075	-0.7299624
6	2.0875	-0.7359686
7	2.1	-0.741939
8	2.1125	-0.7478739
9	2.125	-0.7537739
10	2.1375	-0.7596392
11	2.15	-0.7654703
12	2.1625	-0.7712676
13	2.175	-0.7770315
14	2.1875	-0.7827624
15	2.2	-0.7884606
16	2.2125	-0.7941265
17	2.225	-0.7997605
18	2.2375	-0.805363
19	2.25	-0.8109342
20	2.2625	-0.8164746
21	2.275	-0.8219844
22	2.2875	-0.8274641
23	2.3	-0.8329139
24	2.3125	-0.8383341
25	2.325	-0.8437251
26	2.3375	-0.8490873

27	2.7	-0.9932922
28	2.725	-1.0025102
29	2.75	-1.0116439
30	2.775	-1.020695
31	2.8	-1.0296649
32	2.825	-1.0385551
33	2.85	-1.047367
34	2.875	-1.0561019
35	2.9	-1.0647612
36	2.925	-1.0733461
37	2.95	-1.081858
38	2.975	-1.0902981
39	3.0	-1.0986676

27	2.35	-0.8544208
28	2.3625	-0.859726
29	2.375	-0.8650033
30	2.3875	-0.8702528
31	2.4	-0.8754749
32	2.4125	-0.8806699
33	2.425	-0.8858381
34	2.4375	-0.8909796
35	2.45	-0.8960949
36	2.4625	-0.9011841
37	2.475	-0.9062476
38	2.4875	-0.9112856
39	2.5	-0.9162983
40	2.5125	-0.921286
41	2.525	-0.926249
42	2.5375	-0.9311874
43	2.55	-0.9361016
44	2.5625	-0.9409918
45	2.575	-0.9458581
46	2.5875	-0.9507009
47	2.6	-0.9555204
48	2.6125	-0.9603167
49	2.625	-0.9650901
50	2.6375	-0.9698409
51	2.65	-0.9745692
52	2.6625	-0.9792753
53	2.675	-0.9839593
54	2.6875	-0.9886215
55	2.7	-0.993262
56	2.7125	-0.9978811
57	2.725	-1.002479
58	2.7375	-1.0070558

59	2.75	-1.0116118
60	2.7625	-1.0161471
61	2.775	-1.0206619
62	2.7875	-1.0251565
63	2.8	-1.0296309
64	2.8125	-1.0340854
65	2.825	-1.0385202
66	2.8375	-1.0429353
67	2.85	-1.0473311
68	2.8625	-1.0517076
69	2.875	-1.0560651
70	2.8875	-1.0604037
71	2.9	-1.0647235
72	2.9125	-1.0690247
73	2.925	-1.0733075
74	2.9375	-1.0775721
75	2.95	-1.0818185
76	2.9625	-1.086047
77	2.975	-1.0902577
78	2.9875	-1.0944507
79	3.0	-1.0986262

B. Численное решение ОДУ методом Рунге-Кутта 3 порядка

```
3-order Runge-Kutta meth for dif. eq.

Koshi:  [2, -0.6931471805599453]
step = 0.1
number of steps: 10
range:  from  2  to  3.0
```

k	x	y
0	2.1	-0.7419407
1	2.2	-0.7884636
2	2.3	-0.832918
3	2.4	-0.8754799
4	2.5	-0.9163041
5	2.6	-0.9555269
6	2.7	-0.9932691
7	2.8	-1.0296386
8	2.9	-1.0647317
9	3.0	-1.098635

```
3-order Runge-Kutta meth for dif. eq.

Koshi:  [2, -0.6931471805599453]
step = 0.05
number of steps: 20
range:  from  2  to  3.0
```

k	x	y
0	2.05	-0.71784
1	2.1	-0.7419378
2	2.15	-0.7654685
3	2.2	-0.7884582
4	2.25	-0.8109312
5	2.3	-0.8329103
6	2.35	-0.8544166
7	2.4	-0.8754702
8	2.45	-0.8960896
9	2.5	-0.9162925
10	2.55	-0.9360952
11	2.6	-0.9555135
12	2.65	-0.9745618
13	2.7	-0.993254
14	2.75	-1.0116033
15	2.8	-1.0296219
16	2.85	-1.0473216
17	2.9	-1.0647135
18	2.95	-1.081808
19	3.0	-1.0986152

3-order Runge-Kutta meth for dif. eq.

Koshi: [2, -0.6931471805599453]
step = 0.025
number of steps: 40
range: from 2 to 3.0

k	x	y
0	2.025	-0.7055697
1	2.05	-0.7178398
2	2.075	-0.7299612
3	2.1	-0.7419374
4	2.125	-0.7537719
5	2.15	-0.7654679
6	2.175	-0.7770288
7	2.2	-0.7884575
8	2.225	-0.799757
9	2.25	-0.8109303
10	2.275	-0.8219802
11	2.3	-0.8329093
12	2.325	-0.8437202
13	2.35	-0.8544155
14	2.375	-0.8649976
15	2.4	-0.8754689
16	2.425	-0.8858317
17	2.45	-0.8960882
18	2.475	-0.9062406
19	2.5	-0.916291
20	2.525	-0.9262413
21	2.55	-0.9360936
22	2.575	-0.9458498
23	2.6	-0.9555117
24	2.625	-0.9650812
25	2.65	-0.9745599

3-order Runge-Kutta meth for dif. eq.

Koshi: [2, -0.6931471805599453]
step = 0.0125
number of steps: 80
range: from 2 to 3.0

k	x	y
0	2.0125	-0.6993777
1	2.025	-0.7055697
2	2.0375	-0.7117236
3	2.05	-0.7178398
4	2.0625	-0.7239188
5	2.075	-0.7299612
6	2.0875	-0.7359672
7	2.1	-0.7419374
8	2.1125	-0.7478721
9	2.125	-0.7537718
10	2.1375	-0.7596369
11	2.15	-0.7654679
12	2.1625	-0.771265
13	2.175	-0.7770287
14	2.1875	-0.7827594
15	2.2	-0.7884574
16	2.2125	-0.7941231
17	2.225	-0.7997569
18	2.2375	-0.8053592
19	2.25	-0.8109302
20	2.2625	-0.8164704
21	2.275	-0.8219801
22	2.2875	-0.8274595
23	2.3	-0.8329091
24	2.3125	-0.8383292
25	2.325	-0.8437201

26	2.675	-0.9839497
27	2.7	-0.9932521
28	2.725	-1.0024687
29	2.75	-1.0116012
30	2.775	-1.0206511
31	2.8	-1.0296197
32	2.825	-1.0385087
33	2.85	-1.0473193
34	2.875	-1.056053
35	2.9	-1.0647111
36	2.925	-1.0732948
37	2.95	-1.0818055
38	2.975	-1.0902444
39	3.0	-1.0986127

26	2.3375	-0.849082
27	2.35	-0.8544153
28	2.3625	-0.8597204
29	2.375	-0.8649975
30	2.3875	-0.8702468
31	2.4	-0.8754688
32	2.4125	-0.8806636
33	2.425	-0.8858315
34	2.4375	-0.8909729
35	2.45	-0.8960881
36	2.4625	-0.9011771
37	2.475	-0.9062404
38	2.4875	-0.9112782
39	2.5	-0.9162908
40	2.5125	-0.9212783
41	2.525	-0.9262411
42	2.5375	-0.9311794
43	2.55	-0.9360934
44	2.5625	-0.9409834
45	2.575	-0.9458496
46	2.5875	-0.9506922
47	2.6	-0.9555115
48	2.6125	-0.9603077
49	2.625	-0.9650809
50	2.6375	-0.9698315
51	2.65	-0.9745597
52	2.6625	-0.9792656
53	2.675	-0.9839494
54	2.6875	-0.9886114
55	2.7	-0.9932518
56	2.7125	-0.9978708
57	2.725	-1.0024685

58	2.7375	-1.0070451
59	2.75	-1.0116009
60	2.7625	-1.0161361
61	2.775	-1.0206508
62	2.7875	-1.0251452
63	2.8	-1.0296195
64	2.8125	-1.0340738
65	2.825	-1.0385084
66	2.8375	-1.0429234
67	2.85	-1.047319
68	2.8625	-1.0516954
69	2.875	-1.0560527
70	2.8875	-1.0603911
71	2.9	-1.0647108
72	2.9125	-1.0690119
73	2.925	-1.0732945
74	2.9375	-1.0775589
75	2.95	-1.0818052
76	2.9625	-1.0860336
77	2.975	-1.0902441
78	2.9875	-1.094437
79	3.0	-1.0986123

C. Численное решение ОДУ методом Рунге-Кутта 4 порядка

```
4-order Runge-Kutta meth for dif. eq.

Koshi:  [2, -0.6931471805599453]
step = 0.1
number of steps: 10
range:  from  2  to  3.0
```

k	x	y
0	2.1	-0.7419375
1	2.2	-0.7884576
2	2.3	-0.8329095

```
4-order Runge-Kutta meth for dif. eq.

Koshi:  [2, -0.6931471805599453]
step = 0.05
number of steps: 20
range:  from  2  to  3.0
```

k	x	y
0	2.05	-0.7178398
1	2.1	-0.7419374
2	2.15	-0.7654679

3	2.4	-0.8754692
4	2.5	-0.9162913
5	2.6	-0.9555121
6	2.7	-0.9932525
7	2.8	-1.0296202
8	2.9	-1.0647116
9	3.0	-1.0986132

3	2.2	-0.7884574
4	2.25	-0.8109302
5	2.3	-0.8329091
6	2.35	-0.8544154
7	2.4	-0.8754688
8	2.45	-0.8960881
9	2.5	-0.9162908
10	2.55	-0.9360934
11	2.6	-0.9555115
12	2.65	-0.9745597
13	2.7	-0.9932518
14	2.75	-1.011601
15	2.8	-1.0296195
16	2.85	-1.047319
17	2.9	-1.0647108
18	2.95	-1.0818052
19	3.0	-1.0986123

4-order Runge-Kutta meth for dif. eq.

Koshi: [2, -0.6931471805599453]
step = 0.025
number of steps: 40
range: from 2 to 3.0

k	x	y
0	2.025	-0.7055697
1	2.05	-0.7178398
2	2.075	-0.7299612
3	2.1	-0.7419373
4	2.125	-0.7537718
5	2.15	-0.7654678

4-order Runge-Kutta meth for dif. eq.

Koshi: [2, -0.6931471805599453]
step = 0.0125
number of steps: 80
range: from 2 to 3.0

k	x	y
0	2.0125	-0.6993777
1	2.025	-0.7055697
2	2.0375	-0.7117236
3	2.05	-0.7178398
4	2.0625	-0.7239188
5	2.075	-0.7299612

6	2.175	-0.7770287
7	2.2	-0.7884574
8	2.225	-0.7997569
9	2.25	-0.8109302
10	2.275	-0.8219801
11	2.3	-0.8329091
12	2.325	-0.84372
13	2.35	-0.8544153
14	2.375	-0.8649974
15	2.4	-0.8754687
16	2.425	-0.8858315
17	2.45	-0.896088
18	2.475	-0.9062404
19	2.5	-0.9162907
20	2.525	-0.9262411
21	2.55	-0.9360934
22	2.575	-0.9458495
23	2.6	-0.9555114
24	2.625	-0.9650809
25	2.65	-0.9745596
26	2.675	-0.9839494
27	2.7	-0.9932518
28	2.725	-1.0024684
29	2.75	-1.0116009
30	2.775	-1.0206508
31	2.8	-1.0296194
32	2.825	-1.0385084
33	2.85	-1.047319
34	2.875	-1.0560527
35	2.9	-1.0647107
36	2.925	-1.0732945
37	2.95	-1.0818052

6	2.0875	-0.7359672
7	2.1	-0.7419373
8	2.1125	-0.7478721
9	2.125	-0.7537718
10	2.1375	-0.7596369
11	2.15	-0.7654678
12	2.1625	-0.771265
13	2.175	-0.7770287
14	2.1875	-0.7827593
15	2.2	-0.7884574
16	2.2125	-0.7941231
17	2.225	-0.7997569
18	2.2375	-0.8053592
19	2.25	-0.8109302
20	2.2625	-0.8164704
21	2.275	-0.8219801
22	2.2875	-0.8274595
23	2.3	-0.8329091
24	2.3125	-0.8383292
25	2.325	-0.84372
26	2.3375	-0.849082
27	2.35	-0.8544153
28	2.3625	-0.8597204
29	2.375	-0.8649974
30	2.3875	-0.8702468
31	2.4	-0.8754687
32	2.4125	-0.8806636
33	2.425	-0.8858315
34	2.4375	-0.8909729
35	2.45	-0.896088
36	2.4625	-0.9011771
37	2.475	-0.9062404

38	2.975	-1.090244
39	3.0	-1.0986123

38	2.4875	-0.9112782
39	2.5	-0.9162907
40	2.5125	-0.9212783
41	2.525	-0.9262411
42	2.5375	-0.9311793
43	2.55	-0.9360934
44	2.5625	-0.9409833
45	2.575	-0.9458495
46	2.5875	-0.9506922
47	2.6	-0.9555114
48	2.6125	-0.9603076
49	2.625	-0.9650809
50	2.6375	-0.9698315
51	2.65	-0.9745596
52	2.6625	-0.9792655
53	2.675	-0.9839494
54	2.6875	-0.9886114
55	2.7	-0.9932518
56	2.7125	-0.9978707
57	2.725	-1.0024684
58	2.7375	-1.0070451
59	2.75	-1.0116009
60	2.7625	-1.0161361
61	2.775	-1.0206507
62	2.7875	-1.0251451
63	2.8	-1.0296194
64	2.8125	-1.0340738
65	2.825	-1.0385084
66	2.8375	-1.0429234
67	2.85	-1.047319
68	2.8625	-1.0516954
69	2.875	-1.0560527

70	2.8875	-1.0603911
71	2.9	-1.0647107
72	2.9125	-1.0690118
73	2.925	-1.0732945
74	2.9375	-1.0775589
75	2.95	-1.0818052
76	2.9625	-1.0860335
77	2.975	-1.090244
78	2.9875	-1.0944369
79	3.0	-1.0986123

D. Файл *params.py* (ОДУ)

```
"""
Params for sample solution
"""

import numpy as np
import sympy as sp
from base_objs import *


# equation functions vector
FUNC = lambda x :   2 * x[1] / (x[0] * np.log(x[0]))  +  1 / x[0]
# solution range
RANGE = Range(2, 3)

# Koshi condition
X0 = [ 2, -np.log(2) ]      # x0, y0 = y(x0)

SEGMENTS_COUNT = 10
```

E. Файл *main.py*

```
# solution run

# import params
from cProfile import label
from params import *
# import process func
from solution_methods import *

import numpy as np
import scipy as sp
from scipy.integrate import odeint
import matplotlib.pyplot as plt

DISABLE_PLOTS            = FALSE
TURN_ON_THE_COLORS       = TRUE
```

```
PLOT_DEBUG_POINTS        = FALSE
PLOT_GRID_XLINESSS_COUNT = 20
PLOT_MARKER_SIZE         = 6

def ShowSolutionsPlots(x: list, title: str, labels: list = False):

    """
    Show plot of your solutions

    x is list of many solutions where each one is list of dots (x_2,y,z) (list of list of
list of float)
    """

    if (DISABLE_PLOTS):
        return

    # colors for different plots (if TURN_ON_THE_COLORS is TRUE)
    colors_for_plot = [
        "red",
        "blue",
        "green",
        "purple"
    ]


    # restructure vectors of dots from [ [x1, y1], ... ] to [x1, ...], [y1, ...]

    x_new = []

    for i in range(len(x)):

        x_step = [[], []]

        for k in range(len(x[i])):
            x_step[0].append(x[i][k][0])
            x_step[1].append(x[i][k][1])

        x_new.append(x_step)


    try:

        fig, ax = plt.subplots()

        fig.set_size_inches(13, 9)

        range_start = x[0][0][0]
        range_end   = x[0][len(x[0])-1][0]
        grid_sizze  = (range_end - range_start) / PLOT_GRID_XLINESSS_COUNT
        ax.set_xticks(np.arange( range_start,  range_end + grid_sizze, grid_sizze))
        ax.grid(color='black', linewidth=0.5, linestyle='--')

        for i in range(len(x)):

            if (TURN_ON_THE_COLORS):
                try:
                    theColor = colors_for_plot[i]
                except BaseException:
                    theColor = (0,0,0, 1 / len(x) * (i+1) )
            else:
                theColor = (0,0,0, 1 / len(x) * (i+1) )

                 line,  =  ax.plot(x_new[i][0],  x_new[i][1],  color=theColor,  marker='o',
markersize=PLOT_MARKER_SIZE)
```

```
            if (labels):
                line.set_label(labels[i])

        plt.title(title)
        fig.canvas.manager.set_window_title("Solution of differetintial equations' system")
        ax.legend(labels)

        plt.show()

    except BaseException:
        print("\n!!! PLOT ERROR\n")
        print(TextException())


# here the calculations start
def main():

    # check if data is correct
    assert RANGE.end > RANGE.start,   "Incorrect RANGE (end < start)"

    step = (RANGE.end - RANGE.start) / SEGMENTS_COUNT


    """
    Looking for solution using 2-order Runge-Kutta method
    """

    try:

        x_2  = RungeKutta2(FUNC, X0, step,   SEGMENTS_COUNT)
        x2_2 = RungeKutta2(FUNC, X0, step/2, SEGMENTS_COUNT*2)
        x4_2 = RungeKutta2(FUNC, X0, step/4, SEGMENTS_COUNT*4)
        x8_2 = RungeKutta2(FUNC, X0, step/8, SEGMENTS_COUNT*8)
         ShowSolutionsPlots([x_2, x2_2, x4_2, x8_2], "Runge-Kutta 2-order method", ["full
step", "half step", "quarter step", "1/8 step"])

    except BaseException:
        print(TextException())


    """
    Now looking for solution using 3-order Runge-Kutta method
    """

    try:

        x_3  = RungeKutta3(FUNC, X0, step,   SEGMENTS_COUNT)
        x2_3 = RungeKutta3(FUNC, X0, step/2, SEGMENTS_COUNT*2)
        x4_3 = RungeKutta3(FUNC, X0, step/4, SEGMENTS_COUNT*4)
        x8_3 = RungeKutta3(FUNC, X0, step/8, SEGMENTS_COUNT*8)
         ShowSolutionsPlots([x_3, x2_3, x4_3, x8_3], "Runge-Kutta 3-order method", ["full
step", "half step", "quarter step", "1/8 step"])

    except BaseException:
        print(TextException())


    """
    Now looking for solution using 4-order Runge-Kutta method
    """

    try:

        x_4  = RungeKutta4(FUNC, X0, step,   SEGMENTS_COUNT)
```

```
        x2_4 = RungeKutta4(FUNC, X0, step/2, SEGMENTS_COUNT*2)
        x4_4 = RungeKutta4(FUNC, X0, step/4, SEGMENTS_COUNT*4)
        x8_4 = RungeKutta4(FUNC, X0, step/8, SEGMENTS_COUNT*8)
         ShowSolutionsPlots([x_4, x2_4, x4_4, x8_4], "Runge-Kutta 4-order method", ["full
step", "half step", "quarter step", "1/8 step"])

    except BaseException:
        print(TextException())


    # SciPy solution
    try:

        print("\n\nSolving using SciPy")
        def _f4sp(y: float, x: float) -> float:
            return FUNC([ x, y ])
        _x4sp = np.linspace(RANGE.start, RANGE.end, SEGMENTS_COUNT+1)
        _x_sp = odeint(_f4sp, X0[1], _x4sp)

        x_sp = []
        for i in range(len(_x4sp)):
            x_sp.append([ _x4sp[i], _x_sp[i] ])

    except BaseException:
        print(TextException())


    print("\n\n\
       And   here   we   can   see   da   difference   beetwin   the   2,3   and   4   -order
meth(ods)                    \n\
    with only 11 steps solution so we can see it                                    \n\
    and also the scipy solution                                                     \n\
    ")

     ShowSolutionsPlots([x_2, x_3, x_4, x_sp], "Runge-Kutta 2+3+4-order methods (11 steps)
and SciPy", ["2-order method", "3-order method", "4-order method", "math library solution"])



if __name__ == "__main__":
    main()
    print("-"*200, "мучас грасиас офишион ешто паравасотрощ шуууууууууу".upper(), "="*200,
sep="\n", end="\n"*13)
```

F. Файл *base_objs.py*

```
# basic objects for calculations

from math import fabs
from xml.dom.minidom import TypeInfo
from xml.etree.ElementTree import tostring
import numpy as np
import matplotlib.pyplot as plot

import sys
import linecache

TRUE  = 1
FALSE = 0

lambda_type = type(lambda: None)
```

```
class Range:

    def __init__(self):
        self.start  = 0
        self.end    = 0

    def __init__(self, start: float, end: float):
        self.start  = start
        self.end    = end

    def to_str(self) -> str:
        return "(" + str(self.start) + ", " + str(self.end) + ")"

    def contains(self, x: float) -> bool:
        return (x >= self.start) and (x <= self.end)


def TextException():

    """
    exception text compiler
    """
    exc_type, exc_obj, tb = sys.exc_info()
    f = tb.tb_frame
    lineno = tb.tb_lineno
    filename = f.f_code.co_filename
    linecache.checkcache(filename)
    line = linecache.getline(filename, lineno, f.f_globals)
     return 'EXCEPTION IN ({}, LINE {} "{}"):\n{}'.format(filename, lineno, line.strip(),
exc_obj)


# Accuracy offset
ACCURACY_ROUND_OFFSET = 5
# u can increase count of symbols after point using this constant

def GetIntDigitsCount(x: float) -> int:
    """
    Returns count of digits before point
    """
    return len(str(x).split('.')[0]) + 1#for sign

def GetFloatDigitsCount(x: float) -> int:
    """
    Returns count of digits after point
    """
    try:
        return len(str(x).split('.')[1])
    except BaseException:
        try:
            return int(str(x).split('-')[1])
        except BaseException:
            return 0

def RoundBySymbCount(x: float, symb_count: float) -> float:
    """
    Rounds the float X according to given symbols after point count
    """
    return float(f"{x:.{symb_count}f}")

def RoundByAccur(x: float, accuracy: float) -> float:
    """
    Rounds the float X according to given accuracy
```

```
    """
    return RoundBySymbCount(x, GetFloatDigitsCount(accuracy) + ACCURACY_ROUND_OFFSET)

def RoundByAccurVec(x: list, accuracy: float) -> list:
    """
    Rounds the float X according to given accuracy
    """
    t = []
    for i in x:
        t.append(float(f"{i:.{GetFloatDigitsCount(accuracy)}f}"))
    return t


def CalculateVectorFunction(f: list, x: list) -> list:
    """
    Calculates value of vector function in some vector point
    """
    f_value = []
    for i in f:
        f_value.append(i(x))
    return f_value

def VectorMax(v: list):
    max = v[0]
    for i in v:
        if i > max:
            max = i
    return max

def VectorsMaxDelta(v1: list, v2: list):
    """
    Calculates deltas by every axis and returns the max of them
    """
    if len(v1) != len(v2):
        raise "Incorrect arguements"

    deltas = []
    for i in range(len(v2)):
        deltas.append(fabs(v2[i] - v1[i]))

    return VectorMax(deltas)
```

G. Файл *solution_methods.py* (ОДУ)

```
# Runge-Kutta methods for systems of differential equations

from types import LambdaType
import sympy as sp
import numpy as np

# import some structures
from base_objs import *

# DEBUG lvl (0 - off,  1 - simple,  2 - full)
DEBUG = 2

# (DEBUG) default size for column with float type value
DEFAULT_FLOAT_COLUMN_SIZE          = 11
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART = 3
```

```
def RungeKutta2(f: lambda_type, x0: list, h: float, n: int) -> list:

    """
    Solving differentioal equations using 2-order Runge-Kutta method

    Get     function ( y' = f(list : [x, y]) ),
            Koshi task vector (default x = x0 value, y0 = y(x0)),
            segment size
            and number of segments (number of dots - 1)

    Return  vector of points of our solution
    """

    # DEBUG
    if (DEBUG):
        print("\n\n2-order Runge-Kutta meth for dif. eq.\n")
        print("Koshi:", x0, sep="\t")
        print("step =", h)
        print("number of steps:", n)
        print("range:  from  ", x0[0], "  to  ", x0[0] + h*n, sep="")
    if (DEBUG >= 2):
        print()
        tableFloatColCount = 2
        tableFloatColSize = DEFAULT_FLOAT_COLUMN_SIZE
        print("┌" + "─"*3, ("─┬─" + "─"*tableFloatColSize)*tableFloatColCount, "─┐", sep="")
        print(
            "│" + "k".center(3),
            "x".center(tableFloatColSize),
            "y".center(tableFloatColSize),
            sep=" │ ", end=" │\n"
        )

    # vector of points of our solution
    x = [ x0, ]

    for i in range(n):

        k1 = f(x[i])
        k2 = f([ x[i][0] + h,  x[i][1] + h*f(x[i]) ])


        x.append([
            x[i][0] + h,
            x[i][1] + h/2 * (k1 + k2)
        ])

        # DEBUG
        if (DEBUG >= 2):
            print("├" + "─"*3, ("─┼─" + "─"*tableFloatColSize)*tableFloatColCount, "─┤",
sep="")
            print(
                "│" + str(i).rjust(3),
                                str(RoundBySymbCount(x[i+1][0],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                                str(RoundBySymbCount(x[i+1][1],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                sep=" │ ", end=" │\n"
            )

    if (DEBUG >= 2):
        print("└" + "─"*3, ("─┴─" + "─"*tableFloatColSize)*tableFloatColCount, "─┘", sep="")

    return x
```

```
def RungeKutta3(f: lambda_type, x0: list, h: float, n: int) -> list:

    """
    Solving differentioal equations using 3-order Runge-Kutta method

    Get     function ( y' = f(list : [x, y]) ),
            Koshi task vector (default x = x0 value, y0 = y(x0)),
            segment size
            and number of segments (number of dots - 1)

    Return  vector of points of our solution
    """

    # DEBUG
    if (DEBUG):
        print("\n\n3-order Runge-Kutta meth for dif. eq.\n")
        print("Koshi:", x0, sep="\t")
        print("step =", h)
        print("number of steps:", n)
        print("range:  from  ", x0[0], "  to  ", x0[0] + h*n, sep="")
    if (DEBUG >= 2):
        print()
        tableFloatColCount = 2
        tableFloatColSize = DEFAULT_FLOAT_COLUMN_SIZE
        print("┌" + "─"*3, ("─┬─" + "─"*tableFloatColSize)*tableFloatColCount, "─┐", sep="")
        print(
            "│" + "k".center(3),
            "x".center(tableFloatColSize),
            "y".center(tableFloatColSize),
            sep=" │ ", end=" │\n"
        )

    # vector of points of our solution
    x = [ x0, ]

    for i in range(n):

        k1 = f(x[i])
        k2 = f([ x[i][0] + h/2,  x[i][1] + h/2*k1 ])
        k3 = f([ x[i][0] + h,    x[i][1] - h*k1 + 2*h*k2 ])


        x.append([
            x[i][0] + h,
            x[i][1] + h/6 * (k1 + 4*k2 + k3)
        ])

        # DEBUG
        if (DEBUG >= 2):
             print("├" + "─"*3, ("─┼─" + "─"*tableFloatColSize)*tableFloatColCount, "─┤",
sep="")
            print(
                "│" + str(i).rjust(3),
                                str(RoundBySymbCount(x[i+1][0],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                                str(RoundBySymbCount(x[i+1][1],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                sep=" │ ", end=" │\n"
            )

    if (DEBUG >= 2):
        print("└" + "─"*3, ("─┴─" + "─"*tableFloatColSize)*tableFloatColCount, "─┘", sep="")

    return x
```

```
def RungeKutta4(f: lambda_type, x0: list, h: float, n: int) -> list:

    """
    Solving differentioal equations using 4-order Runge-Kutta method

    Get     function ( y' = f(list : [x, y]) ),
            Koshi task vector (default x = x0 value, y0 = y(x0)),
            segment size
            and number of segments (number of dots - 1)

    Return  vector of points of our solution
    """

    # DEBUG
    if (DEBUG):
        print("\n\n4-order Runge-Kutta meth for dif. eq.\n")
        print("Koshi:", x0, sep="\t")
        print("step =", h)
        print("number of steps:", n)
        print("range:  from  ", x0[0], "  to  ", x0[0] + h*n, sep="")
    if (DEBUG >= 2):
        print()
        tableFloatColCount = 2
        tableFloatColSize = DEFAULT_FLOAT_COLUMN_SIZE
        print("┌" + "─"*3, ("─┬─" + "─"*tableFloatColSize)*tableFloatColCount, "─┐", sep="")
        print(
            "│" + "k".center(3),
            "x".center(tableFloatColSize),
            "y".center(tableFloatColSize),
            sep=" │ ", end=" │\n"
        )

    # vector of points of our solution
    x = [ x0, ]

    for i in range(n):

        k1 = f(x[i])
        k2 = f([ x[i][0] + h/2,  x[i][1] + h/2*k1 ])
        k3 = f([ x[i][0] + h/2,  x[i][1] + h/2*k2 ])
        k4 = f([ x[i][0] + h,    x[i][1] + h*k3 ])


        x.append([
            x[i][0] + h,
            x[i][1] + h/6 * (k1 + 2*k2 + 2*k3 + k4)
        ])

        # DEBUG
        if (DEBUG >= 2):
             print("├" + "─"*3, ("─┼─" + "─"*tableFloatColSize)*tableFloatColCount, "─┤",
sep="")
            print(
                "│" + str(i).rjust(3),
                                str(RoundBySymbCount(x[i+1][0],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                                str(RoundBySymbCount(x[i+1][1],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                sep=" │ ", end=" │\n"
            )

    if (DEBUG >= 2):
        print("└" + "─"*3, ("─┴─" + "─"*tableFloatColSize)*tableFloatColCount, "─┘", sep="")
```

```
    return x
```

H. Файл *params.py* (система)

```
"""
Params for sample solution
"""

import numpy as np
import sympy as sp
from base_objs import *


# equation functions vector
FUNC  = [
    lambda x :   x[2] * x[2] + x[0] ,       # y' = z^2 + x;
    lambda x :   x[0] * x[1]        ,       # z' = xy
]

# solution range
RANGE = Range(0, 1)

# Koshi condition
X0 = [ 0, 1, 0.5 ]      # x0, y0, z0

SEGMENTS_COUNT = 10
```

I. Файл *solution_methods.py* (система)

```
# Runge-Kutta methods for systems of differential equations solution

import sympy as sp
import numpy as np

# import some structures
from base_objs import *

# DEBUG lvl (0 - off,  1 - simple,  2 - full)
DEBUG = 1

# (DEBUG) default size for column with float type value
DEFAULT_FLOAT_COLUMN_SIZE          = 11
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART = 3


def RungeKutta2(f: list, x0: list, h: float, n: int) -> list:

    """
    Solving differentioal equations system using 2-order Runge-Kutta method

    Get     functions vector (our system),
            Koshi task vector (default x = x0 value, y0 = y(x0), z0 = z(x0)),
            segment size
            and number of segments (number of dots - 1)
```

```
    Return  vector of point of our solution
    """

    # DEBUG
    if (DEBUG):
        print("\n\n2-order Runge-Kutta meth for dif. eq. system\n")
        print("Koshi:", x0, sep="\t")
        print("step =", h)
        print("number of steps:", n)
        print("range:  from  ", x0[0], "  to  ", x0[0] + h*n, sep="")
    if (DEBUG >= 2):
        print()
        tableFloatColCount = 3
        tableFloatColSize = DEFAULT_FLOAT_COLUMN_SIZE
        print("┌" + "─"*3, ("─┬─" + "─"*tableFloatColSize)*tableFloatColCount, "─┐", sep="")
        print(
            "│" + "k".center(3),
            "x".center(tableFloatColSize),
            "y".center(tableFloatColSize),
            "z".center(tableFloatColSize),
            sep=" │ ", end=" │\n"
        )

    # vector of point of our solution
    x = [ x0, ]

    for i in range(n):

        k1 = CalculateVectorFunction(f, x[i])
        k2 = CalculateVectorFunction(f, [ x[i][0] + h,  x[i][1] + h*k1[0],  x[i][2] + h*k1[1]
])

        x.append([
            x[i][0] + h,
            x[i][1] + h/2 * (k1[0] + k2[0]),
            x[i][2] + h/2 * (k1[1] + k2[1]),
        ])

        # DEBUG
        if (DEBUG >= 2):
             print("├" + "─"*3, ("─┼─" + "─"*tableFloatColSize)*tableFloatColCount, "─┤",
sep="")
            print(
                "│" + str(i).rjust(3),
                                str(RoundBySymbCount(x[i+1][0],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                                str(RoundBySymbCount(x[i+1][1],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                                str(RoundBySymbCount(x[i+1][2],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                sep=" │ ", end=" │\n"
            )

    if (DEBUG >= 2):
        print("└" + "─"*3, ("─┴─" + "─"*tableFloatColSize)*tableFloatColCount, "─┘", sep="")

    return x

def RungeKutta3(f: list, x0: list, h: float, n: int) -> list:

    """
    Solving differentioal equations system using 3-order Runge-Kutta method
```

```
    Get     functions vector (our system),
            Koshi task vector (default x = x0 value, y0 = y(x0), z0 = z(x0)),
            segment size
            and number of segments (number of dots - 1)

    Return  vector of point of our solution
    """

    # DEBUG
    if (DEBUG):
        print("\n\n3-order Runge-Kutta meth for dif. eq. system\n")
        print("Koshi:", x0, sep="\t")
        print("step =", h)
        print("number of steps:", n)
        print("range:  from  ", x0[0], "  to  ", x0[0] + h*n, sep="")
    if (DEBUG >= 2):
        print()
        tableFloatColCount = 3
        tableFloatColSize = DEFAULT_FLOAT_COLUMN_SIZE
        print("┌" + "─"*3, ("─┬─" + "─"*tableFloatColSize)*tableFloatColCount, "─┐", sep="")
        print(
            "│" + "k".center(3),
            "x".center(tableFloatColSize),
            "y".center(tableFloatColSize),
            "z".center(tableFloatColSize),
            sep=" │ ", end=" │\n"
        )

    # vector of point of our solution
    x = [ x0, ]

    for i in range(n):

        k1 = CalculateVectorFunction(f, x[i])
                k2   =   CalculateVectorFunction(f,   [   x[i][0]   +   h/2,    x[i][1]   +
h/2*k1[0],            x[i][2] + h/2*k1[1] ])
             k3  =  CalculateVectorFunction(f,  [  x[i][0]  +  h,      x[i][1]  -  h*k1[0]  +
2*h*k2[0],  x[i][2] - h*k1[1] + 2*h*k2[1] ])

        x.append([
            x[i][0] + h,
            x[i][1] + h/6 * (k1[0] + 4*k2[0] + k3[0]),
            x[i][2] + h/6 * (k1[1] + 4*k2[1] + k3[1]),
        ])

        # DEBUG
        if (DEBUG >= 2):
             print("├" + "─"*3, ("─┼─" + "─"*tableFloatColSize)*tableFloatColCount, "─┤",
sep="")
            print(
                "│" + str(i).rjust(3),
                                str(RoundBySymbCount(x[i+1][0],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                                str(RoundBySymbCount(x[i+1][1],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                                str(RoundBySymbCount(x[i+1][2],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                    sep=" │ ", end=" │\n"
            )

    if (DEBUG >= 2):
        print("└" + "─"*3, ("─┴─" + "─"*tableFloatColSize)*tableFloatColCount, "─┘", sep="")

    return x
```

```
def RungeKutta4(f: list, x0: list, h: float, n: int) -> list:

    """
    Solving differentioal equations system using 4-order Runge-Kutta method

    Get     functions vector (our system),
            Koshi task vector (default x = x0 value, y0 = y(x0), z0 = z(x0)),
            segment size
            and number of segments (number of dots - 1)

    Return  vector of point of our solution
    """

    # DEBUG
    if (DEBUG):
        print("\n\n4-order Runge-Kutta meth for dif. eq. system\n")
        print("Koshi:", x0, sep="\t")
        print("step =", h)
        print("number of steps:", n)
        print("range:  from  ", x0[0], "  to  ", x0[0] + h*n, sep="")
    if (DEBUG >= 2):
        print()
        tableFloatColCount = 3
        tableFloatColSize = DEFAULT_FLOAT_COLUMN_SIZE
        print("┌" + "─"*3, ("─┬─" + "─"*tableFloatColSize)*tableFloatColCount, "─┐", sep="")
        print(
            "│" + "k".center(3),
            "x".center(tableFloatColSize),
            "y".center(tableFloatColSize),
            "z".center(tableFloatColSize),
            sep=" │ ", end=" │\n"
        )

    # vector of point of our solution
    x = [ x0, ]

    for i in range(n):

        k1 = CalculateVectorFunction(f, x[i])
        k2 = CalculateVectorFunction(f, [ x[i][0] + h/2,  x[i][1] + h/2*k1[0],  x[i][2] +
h/2*k1[1] ])
        k3 = CalculateVectorFunction(f, [ x[i][0] + h/2,  x[i][1] + h/2*k2[0],  x[i][2] +
h/2*k2[1] ])
        k4 = CalculateVectorFunction(f, [ x[i][0] + h,    x[i][1] + h*k3[0],    x[i][2] +
h*k3[1] ])

        x.append([
            x[i][0] + h,
            x[i][1] + h/6 * (k1[0] + 2*k2[0] + 2*k3[0] + k4[0]),
            x[i][2] + h/6 * (k1[1] + 2*k2[1] + 2*k3[1] + k4[1]),
        ])

        # DEBUG
        if (DEBUG >= 2):
            print("├" + "─"*3, ("─┼─" + "─"*tableFloatColSize)*tableFloatColCount, "─┤",
sep="")
            print(
                "│" + str(i).rjust(3),
                            str(RoundBySymbCount(x[i+1][0],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                            str(RoundBySymbCount(x[i+1][1],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
```

```
                                str(RoundBySymbCount(x[i+1][2],   DEFAULT_FLOAT_COLUMN_SIZE   -
DEFAULT_FLOAT_COLUMN_SIZE_INT_PART - 1)).center(tableFloatColSize),
                    sep=" │ ", end=" │\n"
                )

        if (DEBUG >= 2):
            print("└" + "─"*3, ("─┴─" + "─"*tableFloatColSize)*tableFloatColCount, "─┘", sep="")

        return x
```

J. Численное решение системы ОДУ методом Рунге-Кутта 2 порядка

2-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.1
number of steps: 10
range: from 0 to 1.0

k	x	y	z
0	0.1	1.03	0.505125
1	0.2	1.0710407	0.5209302
2	0.3	1.1243163	0.5484132
3	0.4	1.1912987	0.5889658
4	0.5	1.2739069	0.6444415
5	0.6	1.374745	0.7172522
6	0.7	1.4974465	0.8105112
7	0.8	1.6471846	0.9282474
8	0.9	1.8314491	1.0757355
9	1.0	2.0612596	1.2600092

2-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.05
number of steps: 20
range: from 0 to 1.0

k	x	y	z
0	0.05	1.01375	0.5012656
1	0.1	1.030127	0.5051048
2	0.15	1.0492643	0.5116097
3	0.2	1.0713044	0.5208937
4	0.25	1.0964028	0.5330932
5	0.3	1.1247322	0.5483691
6	0.35	1.1564873	0.5669088
7	0.4	1.1918905	0.5889286
8	0.45	1.2311985	0.6146764
9	0.5	1.2747104	0.6444347
10	0.55	1.3227775	0.6785251
11	0.6	1.3758145	0.7173128
12	0.65	1.4343143	0.7612125
13	0.7	1.4988651	0.8106964
14	0.75	1.5701718	0.8663027
15	0.8	1.6490829	0.9286473
16	0.85	1.7366238	0.9984383
17	0.9	1.8340385	1.0764933
18	0.95	1.9428429	1.1637624
19	1.0	2.0648929	1.2613565

2-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.025
number of steps: 40
range: from 0 to 1.0

k	x	y	z
0	0.025	1.0065625	0.5003145
1	0.05	1.0137657	0.5012624
2	0.075	1.0216257	0.5028535
3	0.1	1.0301589	0.5050985
4	0.125	1.0393821	0.5080097
5	0.15	1.0493129	0.5116006
6	0.175	1.0599693	0.5158859
7	0.2	1.0713703	0.5208821
8	0.225	1.0835359	0.5266069
9	0.25	1.0964869	0.5330796
10	0.275	1.1102457	0.5403212
11	0.3	1.1248357	0.5483543
12	0.325	1.1402821	0.557203
13	0.35	1.1566116	0.5668937
14	0.375	1.173853	0.5774541
15	0.4	1.1920372	0.5889144
16	0.425	1.2111975	0.6013065
17	0.45	1.2313697	0.6146646
18	0.475	1.2525928	0.6290252
19	0.5	1.2749088	0.6444272
20	0.525	1.2983634	0.6609121
21	0.55	1.3230062	0.6785242
22	0.575	1.3488913	0.6973105
23	0.6	1.3760776	0.7173213
24	0.625	1.4046293	0.7386102
25	0.65	1.4346167	0.7612342
26	0.675	1.4661166	0.7852544
27	0.7	1.4992131	0.8107359
28	0.725	1.5339983	0.8377481

2-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.0125
number of steps: 80
range: from 0 to 1.0

k	x	y	z
0	0.0125	1.0032031	0.5000784
1	0.025	1.0065645	0.500314
2	0.0375	1.010086	0.500708
3	0.05	1.0137697	0.5012615
4	0.0625	1.0176176	0.5019758
5	0.075	1.0216317	0.5028522
6	0.0875	1.0258141	0.503892
7	0.1	1.0301669	0.5050968
8	0.1125	1.0346922	0.5064681
9	0.125	1.0393921	0.5080076
10	0.1375	1.044269	0.5097169
11	0.15	1.049325	0.511598
12	0.1625	1.0545624	0.5136527
13	0.175	1.0599835	0.515883
14	0.1875	1.0655908	0.518291
15	0.2	1.0713867	0.5208788
16	0.2125	1.0773738	0.5236488
17	0.225	1.0835545	0.5266033
18	0.2375	1.0899317	0.5297448
19	0.25	1.0965079	0.5330758
20	0.2625	1.103286	0.536599
21	0.275	1.110269	0.5403172
22	0.2875	1.1174598	0.5442332
23	0.3	1.1248615	0.5483501
24	0.3125	1.1324772	0.5526708
25	0.325	1.1403103	0.5571988
26	0.3375	1.1483642	0.5619371
27	0.35	1.1566424	0.5668893
28	0.3625	1.1651486	0.572059

29	0.75	1.5705733	0.8663656
30	0.775	1.6090492	0.8966681
31	0.8	1.649548	0.928741
32	0.825	1.6922041	0.9626761
33	0.85	1.7371655	0.9985719
34	0.875	1.7845952	1.0365346
35	0.9	1.8346734	1.0766786
36	0.925	1.8875993	1.1191273
37	0.95	1.943593	1.1640143
38	0.975	2.0028987	1.2114843
39	1.0	2.065787	1.2616942

29	0.375	1.1738866	0.5774498
30	0.3875	1.1828602	0.5830656
31	0.4	1.1920736	0.5889102
32	0.4125	1.201531	0.5949878
33	0.425	1.2112369	0.6013025
34	0.4375	1.2211957	0.6078587
35	0.45	1.2314122	0.6146609
36	0.4625	1.2418915	0.6217137
37	0.475	1.2526386	0.6290219
38	0.4875	1.2636589	0.6365905
39	0.5	1.274958	0.6444245
40	0.5125	1.2865417	0.6525292
41	0.525	1.2984162	0.6609101
42	0.5375	1.3105877	0.6695728
43	0.55	1.3230628	0.6785231
44	0.5625	1.3358485	0.6877669
45	0.575	1.348952	0.6973105
46	0.5875	1.3623808	0.7071602
47	0.6	1.3761426	0.7173226
48	0.6125	1.3902459	0.7278045
49	0.625	1.404699	0.7386129
50	0.6375	1.4195111	0.7497552
51	0.65	1.4346914	0.7612387
52	0.6625	1.4502499	0.7730713
53	0.675	1.4661967	0.785261
54	0.6875	1.4825427	0.7978159
55	0.7	1.499299	0.8107448
56	0.7125	1.5164775	0.8240563
57	0.725	1.5340905	0.8377597
58	0.7375	1.5521509	0.8518645
59	0.75	1.5706724	0.8663804
60	0.7625	1.589669	0.8813175
61	0.775	1.6091557	0.8966864
62	0.7875	1.6291482	0.912498

63	0.8	1.6496627	0.9287635
64	0.8125	1.6707166	0.9454945
65	0.825	1.6923278	0.9627032
66	0.8375	1.7145153	0.9804022
67	0.85	1.737299	0.9986044
68	0.8625	1.7606998	1.0173234
69	0.875	1.7847397	1.0365732
70	0.8875	1.8094417	1.0563684
71	0.9	1.8348301	1.0767241
72	0.9125	1.8609304	1.0976561
73	0.925	1.8877694	1.1191808
74	0.9375	1.9153754	1.1413151
75	0.95	1.9437782	1.1640768
76	0.9625	1.9730089	1.1874843
77	0.975	2.0031007	1.211557
78	0.9875	2.0340881	1.2363147
79	1.0	2.0660079	1.2617785

К. Численное решение системы ОДУ методом Рунге-Кутта 3 порядка

3-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.1
number of steps: 10
range: from 0 to 1.0

k	x	y	z
0	0.1	1.0301705	0.5051
1	0.2	1.071395	0.5208853
2	0.3	1.124876	0.5483599
3	0.4	1.1920963	0.5889236
4	0.5	1.2749916	0.6444416
5	0.6	1.3761909	0.7173437
6	0.7	1.4993671	0.8107702
7	0.8	1.6497576	0.9287935
8	0.9	1.8349609	1.0767586

3-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.05
number of steps: 20
range: from 0 to 1.0

k	x	y	z
0	0.05	1.013771	0.5012615
1	0.1	1.0301696	0.5050967
2	0.15	1.0493292	0.5115979
3	0.2	1.0713925	0.5208787
4	0.25	1.0965153	0.5330757
5	0.3	1.1248707	0.5483501
6	0.35	1.1566536	0.5668895
7	0.4	1.1920869	0.5889107
8	0.45	1.2314278	0.6146617

9	1.0	2.0661867	1.2618158

9	0.5	1.2749762	0.6444259
10	0.55	1.3230841	0.6785252
11	0.6	1.3761673	0.7173256
12	0.65	1.4347199	0.761243
13	0.7	1.499332	0.8107506
14	0.75	1.5707106	0.8663882
15	0.8	1.6497073	0.9287739
16	0.85	1.737351	0.9986181
17	0.9	1.834891	1.0767419
18	0.95	1.94385	1.1640998
19	1.0	2.0660931	1.261808

3-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.025
number of steps: 40
range: from 0 to 1.0

k	x	y	z
0	0.025	1.0065651	0.5003139
1	0.05	1.013771	0.5012612
2	0.075	1.0216337	0.5028517
3	0.1	1.0301695	0.5050962
4	0.125	1.0393955	0.5080069
5	0.15	1.049329	0.5115972
6	0.175	1.0599883	0.5158821
7	0.2	1.0713922	0.5208778
8	0.225	1.0835608	0.5266022
9	0.25	1.0965149	0.5330746
10	0.275	1.1102768	0.5403159
11	0.3	1.1248701	0.5483488
12	0.325	1.1403198	0.5571974
13	0.35	1.1566528	0.566888
14	0.375	1.1738978	0.5774485
15	0.4	1.1920858	0.5889089

3-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.0125
number of steps: 80
range: from 0 to 1.0

k	x	y	z
0	0.0125	1.0032035	0.5000783
1	0.025	1.0065651	0.5003139
2	0.0375	1.010087	0.5007078
3	0.05	1.013771	0.5012612
4	0.0625	1.0176192	0.5019754
5	0.075	1.0216337	0.5028517
6	0.0875	1.0258164	0.5038915
7	0.1	1.0301695	0.5050962
8	0.1125	1.0346952	0.5064674
9	0.125	1.0393955	0.5080068
10	0.1375	1.0442727	0.5097161
11	0.15	1.049329	0.5115971
12	0.1625	1.0545667	0.5136518
13	0.175	1.0599882	0.515882
14	0.1875	1.0655959	0.5182899
15	0.2	1.0713922	0.5208777

16	0.425	1.2112501	0.6013013
17	0.45	1.2314265	0.6146598
18	0.475	1.2526539	0.629021
19	0.5	1.2749745	0.6444238
20	0.525	1.2984339	0.6609097
21	0.55	1.3230819	0.6785229
22	0.575	1.3489724	0.6973106
23	0.6	1.3761645	0.7173232
24	0.625	1.4047225	0.738614
25	0.65	1.4347166	0.7612404
26	0.675	1.4662237	0.7852633
27	0.7	1.4993279	0.8107479
28	0.725	1.5341216	0.8377638
29	0.75	1.5707058	0.8663855
30	0.775	1.6091917	0.8966928
31	0.8	1.6497014	0.9287712
32	0.825	1.6923695	0.9627125
33	0.85	1.7373441	0.9986155
34	0.875	1.7847884	1.0365863
35	0.9	1.8348829	1.0767395
36	0.925	1.8878268	1.1191988
37	0.95	1.9438406	1.1640978
38	0.975	2.0031688	1.2115813
39	1.0	2.0660824	1.2618067

16	0.2125	1.0773796	0.5236477
17	0.225	1.0835607	0.5266021
18	0.2375	1.0899382	0.5297435
19	0.25	1.0965149	0.5330745
20	0.2625	1.1032934	0.5365976
21	0.275	1.1102768	0.5403158
22	0.2875	1.1174679	0.5442318
23	0.3	1.12487	0.5483486
24	0.3125	1.1324862	0.5526694
25	0.325	1.1403197	0.5571972
26	0.3375	1.1483741	0.5619356
27	0.35	1.1566527	0.5668878
28	0.3625	1.1651593	0.5720575
29	0.375	1.1738977	0.5774483
30	0.3875	1.1828718	0.5830641
31	0.4	1.1920857	0.5889087
32	0.4125	1.2015436	0.5949863
33	0.425	1.2112499	0.6013011
34	0.4375	1.2212093	0.6078573
35	0.45	1.2314263	0.6146596
36	0.4625	1.2419061	0.6217124
37	0.475	1.2526537	0.6290207
38	0.4875	1.2636746	0.6365894
39	0.5	1.2749743	0.6444235
40	0.5125	1.2865586	0.6525284
41	0.525	1.2984337	0.6609094
42	0.5375	1.3106058	0.6695722
43	0.55	1.3230816	0.6785226
44	0.5625	1.335868	0.6877666
45	0.575	1.3489722	0.6973103
46	0.5875	1.3624016	0.7071602
47	0.6	1.3761642	0.7173229
48	0.6125	1.3902682	0.727805
49	0.625	1.4047222	0.7386137

50	0.6375	1.419535	0.7497562
51	0.65	1.4347162	0.7612401
52	0.6625	1.4502755	0.773073
53	0.675	1.4662233	0.785263
54	0.6875	1.4825702	0.7978183
55	0.7	1.4993275	0.8107476
56	0.7125	1.516507	0.8240596
57	0.725	1.534121	0.8377635
58	0.7375	1.5521826	0.8518687
59	0.75	1.5707052	0.8663851
60	0.7625	1.589703	0.8813229
61	0.775	1.609191	0.8966924
62	0.7875	1.6291848	0.9125046
63	0.8	1.6497007	0.9287708
64	0.8125	1.670756	0.9455026
65	0.825	1.6923688	0.9627122
66	0.8375	1.7145579	0.980412
67	0.85	1.7373433	0.9986151
68	0.8625	1.7607458	1.0173351
69	0.875	1.7847876	1.036586
70	0.8875	1.8094915	1.0563823
71	0.9	1.8348819	1.0767392
72	0.9125	1.8609844	1.0976725
73	0.925	1.8878258	1.1191985
74	0.9375	1.9154342	1.1413342
75	0.95	1.9438395	1.1640975
76	0.9625	1.973073	1.1875067
77	0.975	2.0031676	1.2115811
78	0.9875	2.0341581	1.2363407
79	1.0	2.0660811	1.2618065

М. Численное решение системы ОДУ методом Рунге-Кутта 4 порядка

4-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.1
number of steps: 10
range: from 0 to 1.0

k	x	y	z
0	0.1	1.0301696	0.5050963
1	0.2	1.0713924	0.5208779
2	0.3	1.1248703	0.5483489
3	0.4	1.1920861	0.5889091
4	0.5	1.2749749	0.644424
5	0.6	1.3761651	0.7173234
6	0.7	1.4993287	0.8107483
7	0.8	1.6497025	0.9287716
8	0.9	1.8348845	1.07674
9	1.0	2.0660848	1.2618069

4-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.05
number of steps: 20
range: from 0 to 1.0

k	x	y	z
0	0.05	1.013771	0.5012612
1	0.1	1.0301695	0.5050962
2	0.15	1.049329	0.5115971
3	0.2	1.0713922	0.5208777
4	0.25	1.0965149	0.5330745
5	0.3	1.12487	0.5483486
6	0.35	1.1566527	0.5668878
7	0.4	1.1920857	0.5889087
8	0.45	1.2314263	0.6146596
9	0.5	1.2749743	0.6444235
10	0.55	1.3230816	0.6785226
11	0.6	1.3761642	0.7173228
12	0.65	1.4347162	0.7612401
13	0.7	1.4993275	0.8107476
14	0.75	1.5707052	0.8663851
15	0.8	1.6497008	0.9287708
16	0.85	1.7373433	0.9986151
17	0.9	1.834882	1.0767392
18	0.95	1.9438396	1.1640975
19	1.0	2.0660812	1.2618065

4-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.025
number of steps: 40
range: from 0 to 1.0

k	x	y	z

4-order Runge-Kutta meth for dif. eq. system

Koshi: [0, 1, 0.5]
step = 0.0125
number of steps: 80
range: from 0 to 1.0

k	x	y	z

0	0.025	1.0065651	0.5003139
1	0.05	1.013771	0.5012612
2	0.075	1.0216337	0.5028517
3	0.1	1.0301695	0.5050962
4	0.125	1.0393955	0.5080068
5	0.15	1.049329	0.5115971
6	0.175	1.0599882	0.515882
7	0.2	1.0713922	0.5208777
8	0.225	1.0835607	0.5266021
9	0.25	1.0965149	0.5330745
10	0.275	1.1102768	0.5403158
11	0.3	1.12487	0.5483486
12	0.325	1.1403197	0.5571972
13	0.35	1.1566527	0.5668878
14	0.375	1.1738977	0.5774483
15	0.4	1.1920857	0.5889087
16	0.425	1.2112499	0.601301
17	0.45	1.2314263	0.6146595
18	0.475	1.2526537	0.6290207
19	0.5	1.2749743	0.6444235
20	0.525	1.2984337	0.6609093
21	0.55	1.3230816	0.6785226
22	0.575	1.3489721	0.6973103
23	0.6	1.3761642	0.7173228
24	0.625	1.4047221	0.7386137
25	0.65	1.4347162	0.7612401
26	0.675	1.4662232	0.785263
27	0.7	1.4993274	0.8107475
28	0.725	1.534121	0.8377634
29	0.75	1.5707051	0.8663851
30	0.775	1.6091909	0.8966924
31	0.8	1.6497006	0.9287708
32	0.825	1.6923687	0.9627121
33	0.85	1.7373432	0.9986151

0	0.0125	1.0032035	0.5000783
1	0.025	1.0065651	0.5003139
2	0.0375	1.010087	0.5007078
3	0.05	1.013771	0.5012612
4	0.0625	1.0176192	0.5019754
5	0.075	1.0216337	0.5028517
6	0.0875	1.0258164	0.5038915
7	0.1	1.0301695	0.5050962
8	0.1125	1.0346952	0.5064674
9	0.125	1.0393955	0.5080068
10	0.1375	1.0442727	0.5097161
11	0.15	1.049329	0.5115971
12	0.1625	1.0545667	0.5136517
13	0.175	1.0599882	0.515882
14	0.1875	1.0655959	0.5182899
15	0.2	1.0713922	0.5208777
16	0.2125	1.0773796	0.5236476
17	0.225	1.0835607	0.5266021
18	0.2375	1.0899382	0.5297435
19	0.25	1.0965149	0.5330745
20	0.2625	1.1032934	0.5365976
21	0.275	1.1102768	0.5403158
22	0.2875	1.1174679	0.5442318
23	0.3	1.12487	0.5483486
24	0.3125	1.1324862	0.5526693
25	0.325	1.1403197	0.5571972
26	0.3375	1.148374	0.5619356
27	0.35	1.1566527	0.5668878
28	0.3625	1.1651593	0.5720575
29	0.375	1.1738977	0.5774483
30	0.3875	1.1828718	0.5830641
31	0.4	1.1920857	0.5889087
32	0.4125	1.2015436	0.5949863
33	0.425	1.2112499	0.601301

34	0.875	1.7847874	1.0365859
35	0.9	1.8348818	1.0767392
36	0.925	1.8878256	1.1191984
37	0.95	1.9438394	1.1640975
38	0.975	2.0031675	1.2115811
39	1.0	2.066081	1.2618064

34	0.4375	1.2212092	0.6078573
35	0.45	1.2314263	0.6146595
36	0.4625	1.2419061	0.6217124
37	0.475	1.2526537	0.6290207
38	0.4875	1.2636746	0.6365893
39	0.5	1.2749743	0.6444235
40	0.5125	1.2865586	0.6525283
41	0.525	1.2984337	0.6609093
42	0.5375	1.3106058	0.6695722
43	0.55	1.3230816	0.6785226
44	0.5625	1.335868	0.6877666
45	0.575	1.3489721	0.6973103
46	0.5875	1.3624016	0.7071602
47	0.6	1.3761642	0.7173228
48	0.6125	1.3902682	0.727805
49	0.625	1.4047221	0.7386137
50	0.6375	1.419535	0.7497562
51	0.65	1.4347162	0.76124
52	0.6625	1.4502755	0.773073
53	0.675	1.4662232	0.7852629
54	0.6875	1.4825701	0.7978183
55	0.7	1.4993274	0.8107475
56	0.7125	1.5165069	0.8240595
57	0.725	1.534121	0.8377634
58	0.7375	1.5521825	0.8518687
59	0.75	1.5707051	0.8663851
60	0.7625	1.5897029	0.8813228
61	0.775	1.6091909	0.8966924
62	0.7875	1.6291847	0.9125046
63	0.8	1.6497006	0.9287708
64	0.8125	1.6707559	0.9455026
65	0.825	1.6923687	0.9627121
66	0.8375	1.7145578	0.9804119
67	0.85	1.7373432	0.9986151

68	0.8625	1.7607457	1.0173351
69	0.875	1.7847874	1.0365859
70	0.8875	1.8094914	1.0563822
71	0.9	1.8348818	1.0767392
72	0.9125	1.8609843	1.0976724
73	0.925	1.8878256	1.1191984
74	0.9375	1.9154341	1.1413342
75	0.95	1.9438394	1.1640974
76	0.9625	1.9730728	1.1875067
77	0.975	2.0031675	1.2115811
78	0.9875	2.0341579	1.2363407
79	1.0	2.0660809	1.2618064