Институт информационных и вычислительных технологий

Лабораторная работа №4
“Численное решение задачи Коши”
по курсу
“**Вычислительные методы**”

Выполнил:
Студент Балашов С.А.
Проверила:
Старший преподаватель кафедры МКМ
Шевченко О.В.

Москва, 2021

Содержание

Задача 1

Найти приближенное решение задачи Коши для обыкновенного дифференциального уравнения (ОДУ) 1 порядка

$$y' = r(t)y(t) \quad t \in [t_0, T],$$
$$y(t_0) = y_0$$

и вычислить и погрешность приближенного решения.

r(t) = 0.4tg(0.4t), y0 = 1, t0 = 0, T = pi/2

1. Найти аналитическое решение задачи.
   *y(t) = 1/cos(0.4*t)*
2. Найти приближенное решение задачи Коши с шагом h= (T - t0) / 10 явным методом Эйлера, методом Эйлера-Коши, усовершенствованным методом Эйлера.
3. Используя метод Рунге-Кутты 4 порядка точности, найти приближенное решение задачи Коши с тем же шагом h.
4. Найти величины погрешностей приближенных решений по формуле $\varepsilon = \max_{0 \le i \le N} | y(t_i) - y_i |$, где y(ti) и yi - значения точного и приближенного решений в узлах сетки ti , i=0..N.
5. Построить таблицы и графики найденных решений (на одном чертеже). Сравнить полученные результаты.

Табл. 1 Результаты численного решения задачи Коши

Координата t:	Метод Эйлера	Метод Эйлера-Коши	Усовершенствованный метод Эйлера	Метод Рунге-Кутты 4	Аналитическое решение
0	1	1	1	1	1
pi/20	1.0	1.0019765225644712	1.0019745705306631	1.0019771732421643	1.0019771730711422
pi/10	1.0039530451 2894	1.0079492702 50284	1.0079374194700566	1.0079479715039292	1.0079479708092973

3pi/20	1.011921931405191	1.0180380680746308	1.0180078061813587	1.0180321498031268	1.0180321481910424
pi/5	1.024050642491016 3	1.0324492397800047	1.0323909850241977	1.0324358744188675	1.0324358714173398
pi/4	1.040571116859595	1.0514861242652809	1.0513886151272767	1.0514622292251905	1.0514622242382672
3pi/10	1.061814695263788 1	1.0755651914393731	1.0754147005916814	1.0755273147751745	1.0755273070222477
7pi/20	1.088229343143446 7	1.1052394096244428	1.1050186834147178	1.1051835903248115	1.1051835787563995
2pi/5	1.120404395246899 2	1.1412314292341348	1.1409182036322914	1.1411530204 1545	1.1411530035922413
9pi/20	1.159105519865351 3	1.1844805354313235	1.1840453947537275	1.1843739738227248	1.1843739497369181
pi/2	1.205324048583731 5	1.2362095143267275	1.2356127140257167	1.2360680116 95188	1.2360679774997896

Табл. 2 Погрешности приближенных решений

	Погрешность приближенного решения
Метод Эйлера	0.030743928916058
Метод Эйлера-Коши	0.00014153682693796
Усовершенствованный метод Эйлера	0.000455263474072876
Метод Рунге-Кутты 4	3.4195398512793E-08

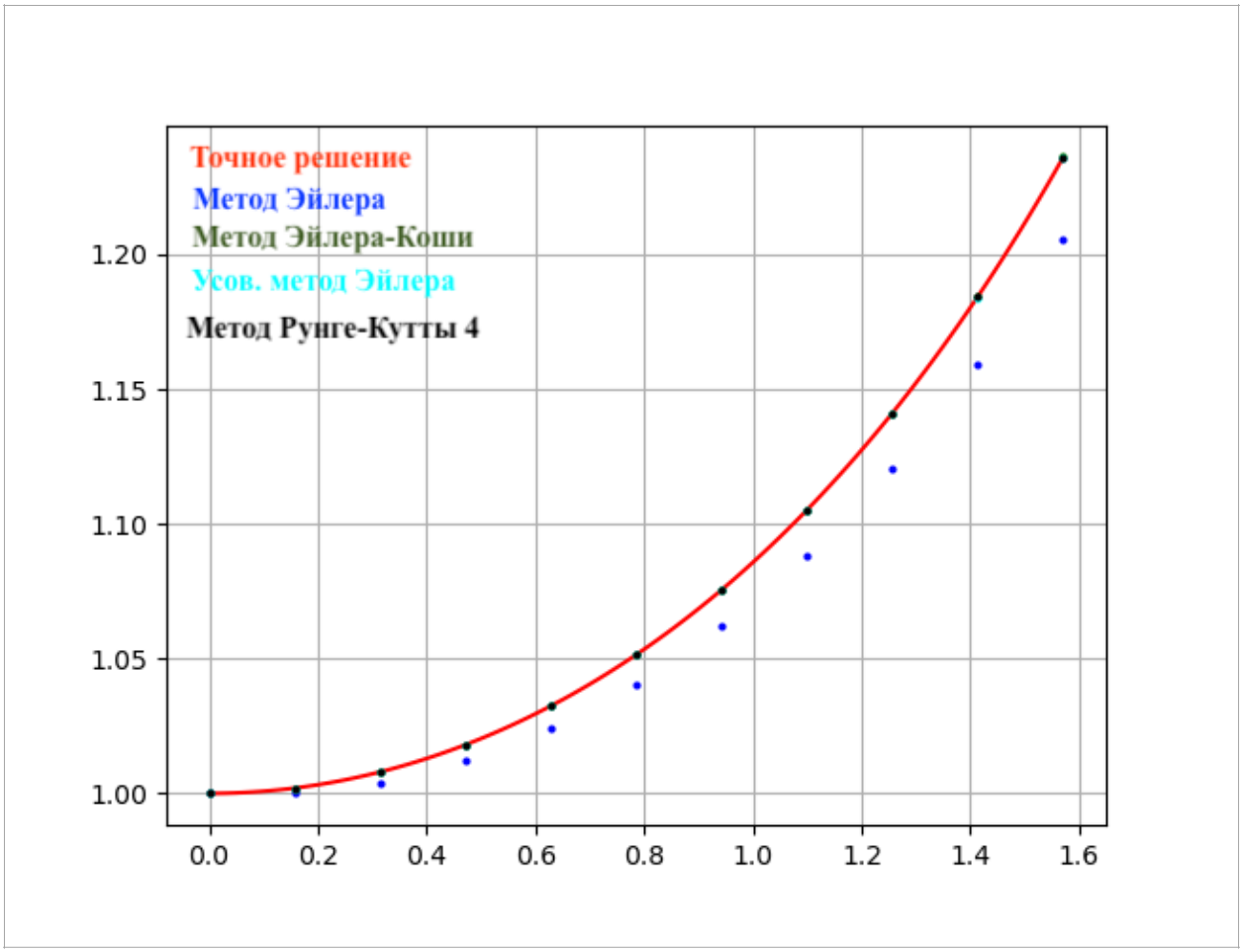

Рис. 1 График с точным и приближенными решениями

Вывод: По значениям погрешностей из Табл. 2 видно, что наиболее приближенным к точному решению является метод Рунге-Кутты 4 порядка, собственно из-за своего порядка точности. Далее следуют метод Эйлера-Коши и усовершенствованный метод Эйлера (порядок точности p у обоих = 2) и метод Эйлера (порядок точности p = 1).

Задача 2

Найти приближенное решение задачи Коши для обыкновенного дифференциального уравнения (ОДУ) 1-го порядка

$$y' = f(t, y), \quad t \in [t_0, T],$$

$$y(t_0) = y_0$$

и оценить погрешность решения задачи при разных значениях шага. Подобрать шаг, при котором достигается точность $\varepsilon = 10^{-4}$.

-20y+20-19e(-t), t0 = 0, T = 1.5, y0 = 1

Метод Рунге-Кутты 3 порядка (вариант 2)

1. Составить программу, реализующую метод, указанный в индивидуальном варианте. Проверить правильность работы программы на задаче Коши для уравнения $y' = y + 3t^2 - t^3$, точным решением которого является функция y(t) = t^3 (t0 = 0, T = 1, y0 = 0).

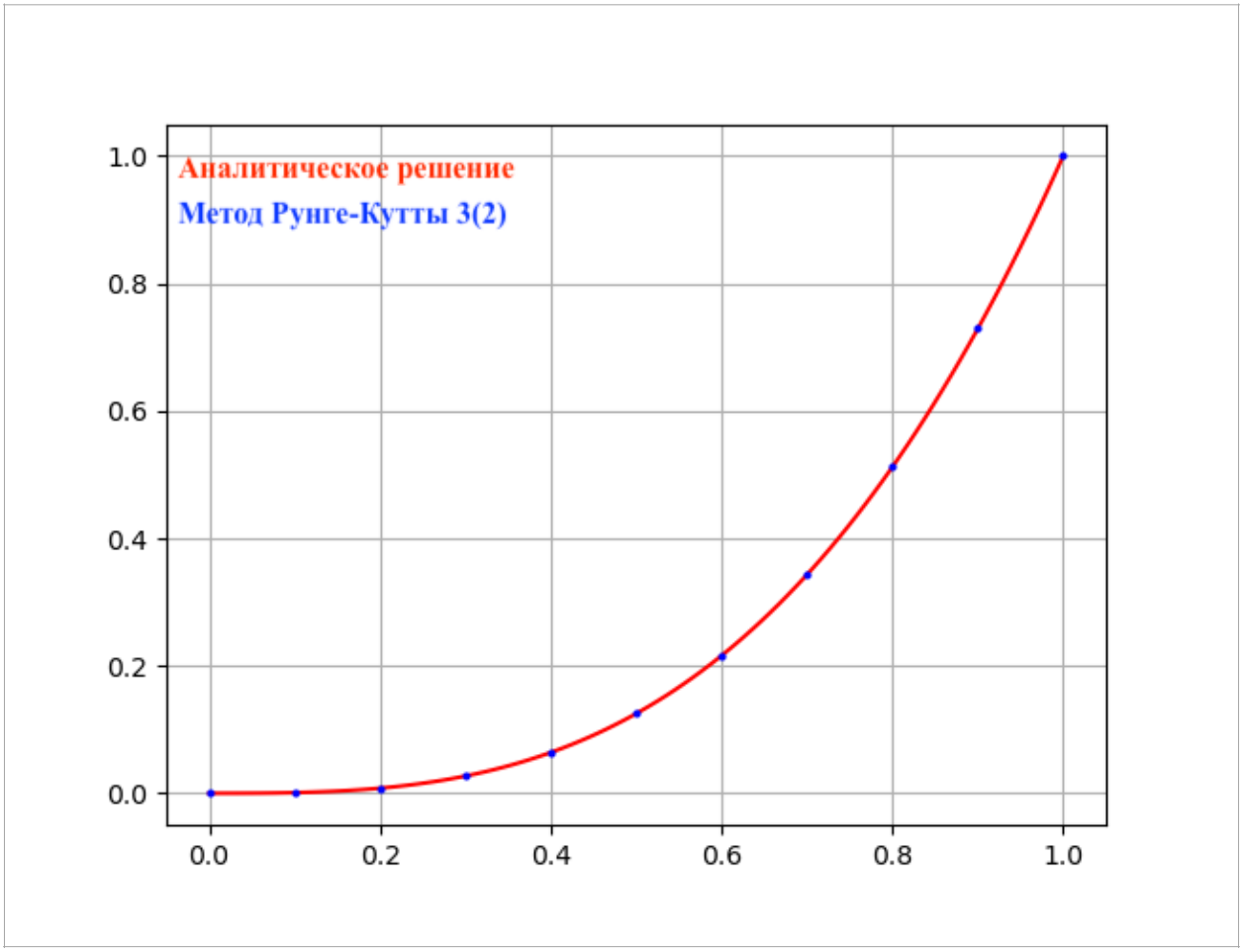

Рис. 2 График проверочной функции

2. Используя эту программу, найти решение исходной задачи с шагом h0 = (T - t0) /10 . Построить график полученного решения.

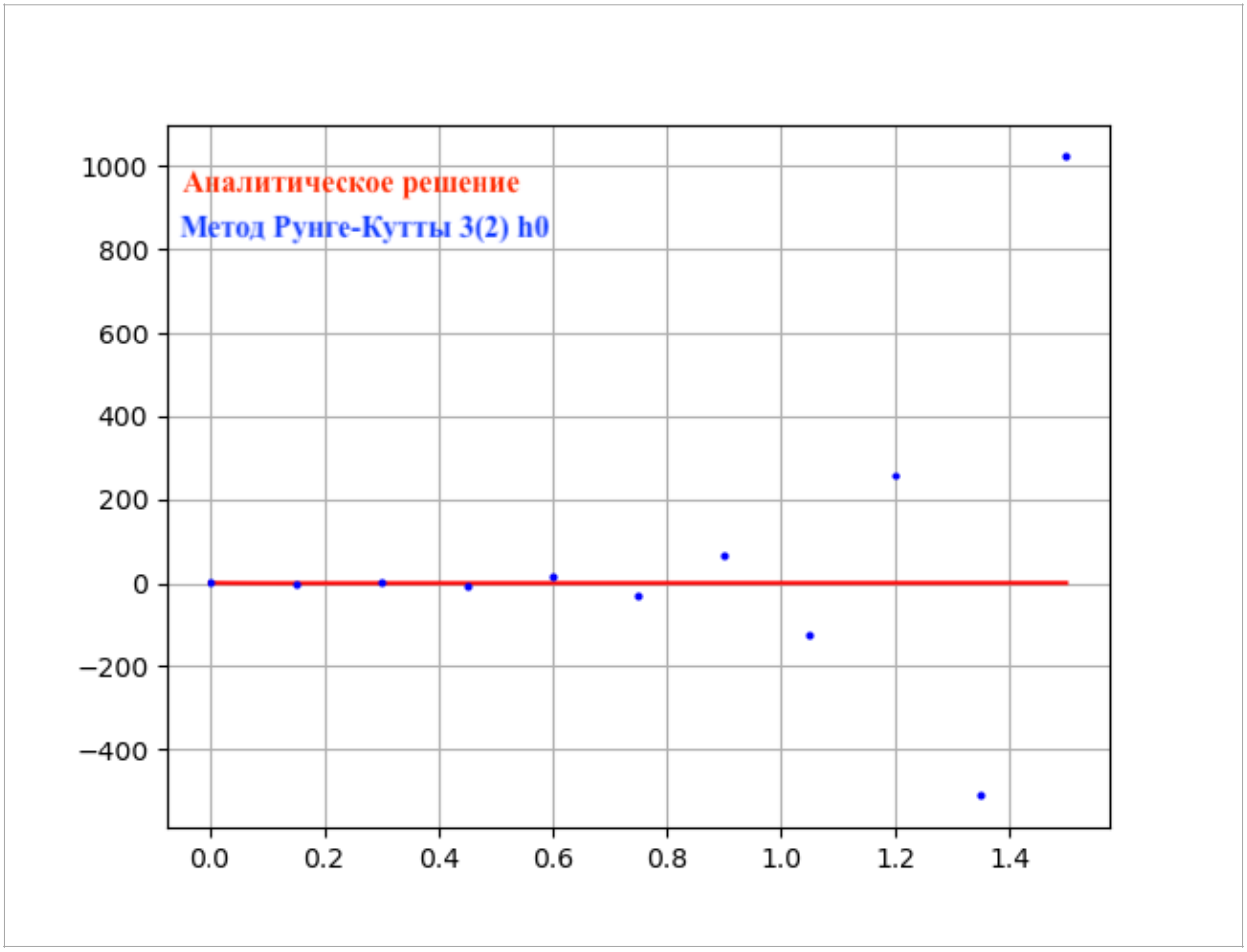

Рис. 3

3. Найти решение той же задачи с шагом $h = 2h0$ и определить погрешность r_i по правилу Рунге (в тех точках, в которых это возможно). Вычислить величину $R = \max|r|$

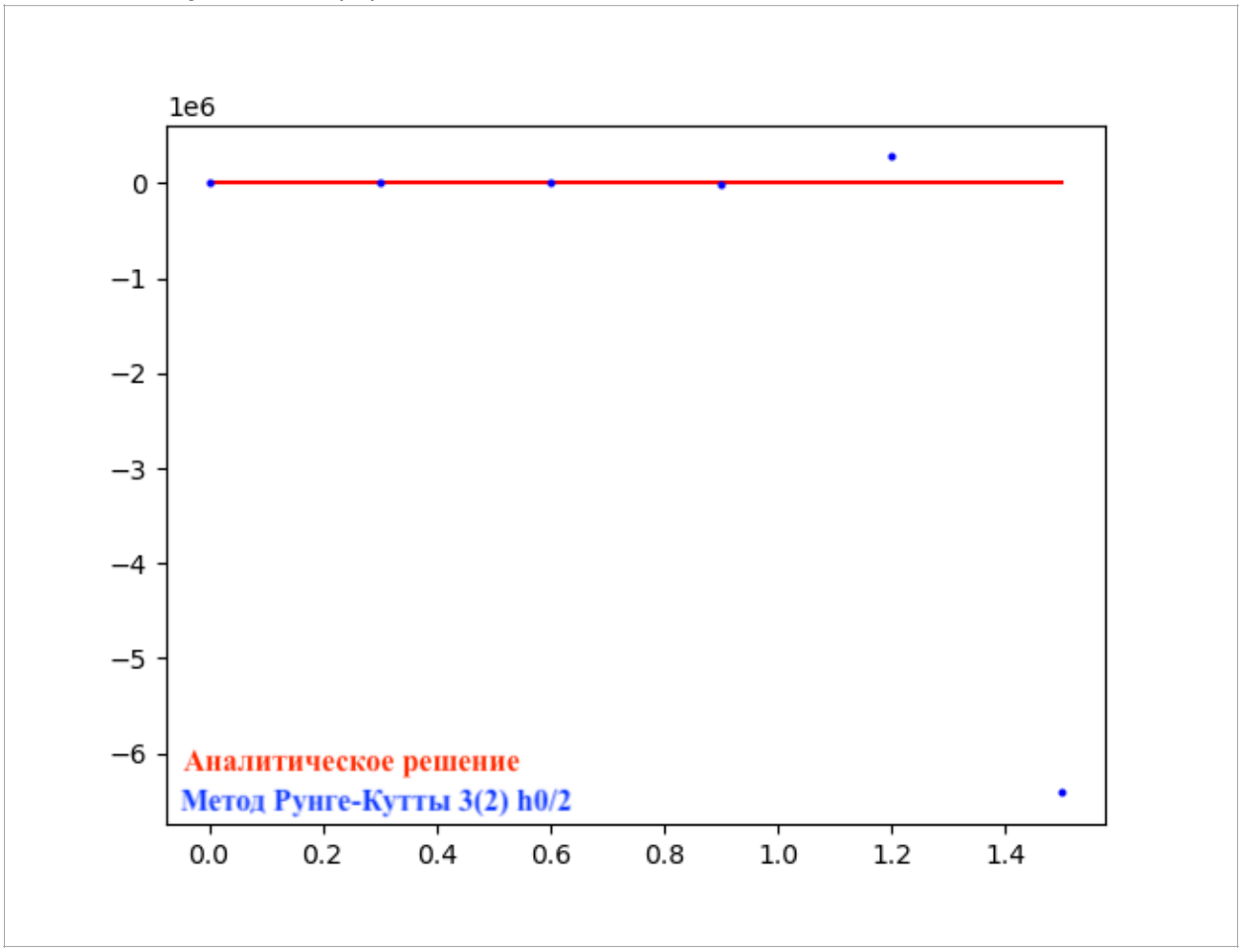

Рис. 4

4. Подобрать шаг $h2$, при котором погрешность по правилу Рунге (R =max|r|) не превосходит точности eps = 10^{-4} . Построить график полученного решения.

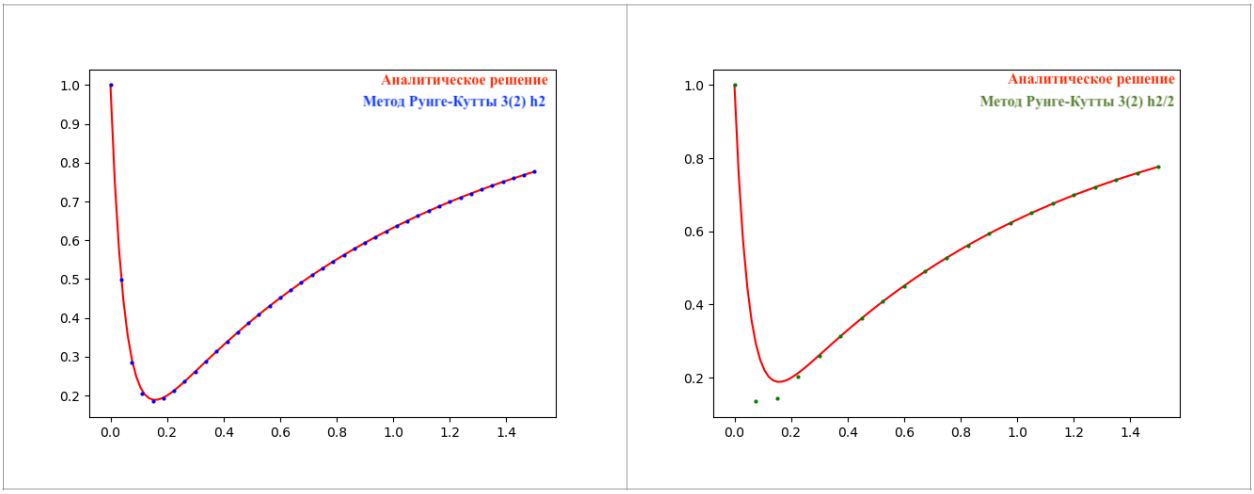

Рис. 5 и 6

5. Используя метод Рунге-Кутты 4 порядка точности (см. Приложение А), найти решение исходной задачи с шагом h0 = (T - t0) /10 и погрешность этого решения по правилу Рунге.

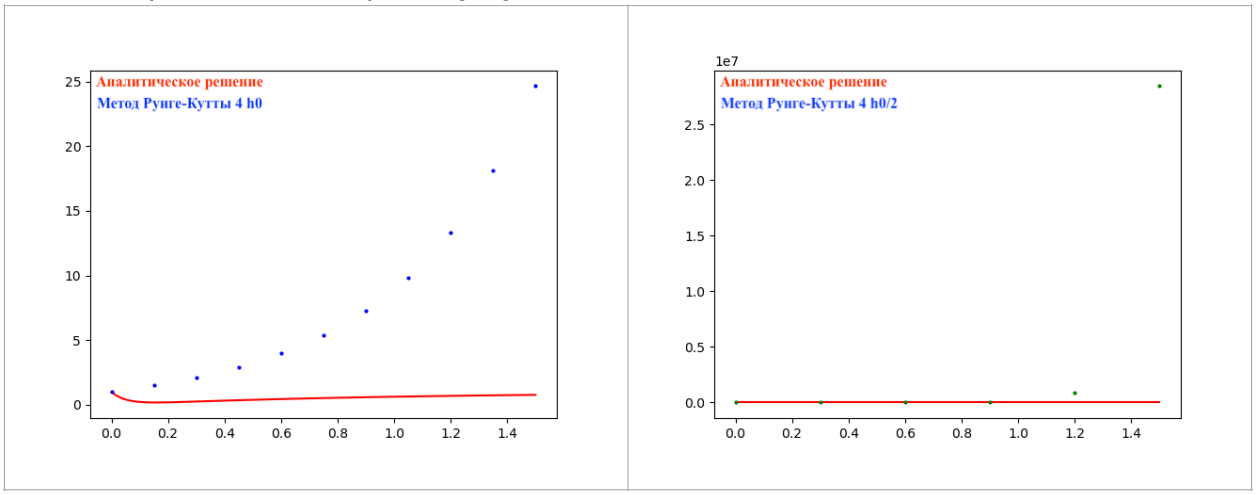

Рис. 7 и 8

6. Найдите шаг $h3$, при котором погрешность встроенной функции (R=max|r| , найденная по правилу Рунге) не превосходит той же точности eps = 10^{-4} .

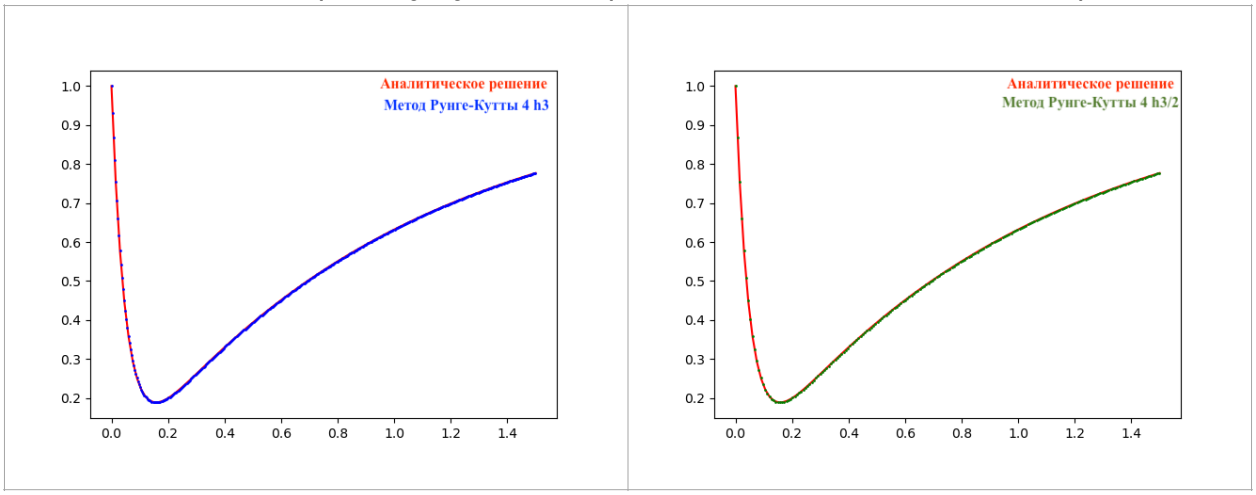

Рис. 9 и 10

7. Заполнить таблицу

Табл. 3 Порядки точности

Метод	Порядок точности метода	Погрешность при шаге h0	Шаг при котором достигается точность eps = 10^{-4}
Метод Рунге-Кутты 3 порядка (2 вариант)	3	916051.261341593	0.0075
Метод Рунге-Кутты 4 порядка	4	1896645.66936417	0.01875

Вывод: По графикам (рис. 3, 4, 5, 6, 7, 8, 9, 10) и табл. 3 видно, что методы Рунге-Кутты 3 и 4 порядка не отличаются стабильностью, однако, для метода 4-го порядка заданная точность достигается быстрее, чем для 3-го порядка. Это объясняется как раз порядком точности.

Приложение. Код программы

main.py

```
from math import cos, pi, tan
import matplotlib.pyplot as plt
import numpy as np
import methods


def func(t, y):
    return 0.4 * tan(0.4 * t) * y


def Yt(t):
    return cos(0.4 * t) ** -1


def f2(t, y):
    return y + 3 * t ** 2 - t ** 3


def Yt2(t):
    return t ** 3


def f2_2(t, y):
    return 20 * (1 - y) - 19 * np.exp(-t)


def Yt2_2(t):
    return np.exp(-20 * t) - np.exp(-t) + 1


def first():
    y0 = 1
    t0 = 0
    T = 0.5 * pi
    N = 10
    h = (T - t0) / N

    euler = methods.Euler(func, h, t0, y0, N, Yt)
    y1 = euler[0]
    eps1 = euler[1]
    epsilon1 = max(max(eps1), -min(eps1))
    cauchy = methods.EulerCauchy(func, h, t0, y0, N, Yt)
    y2 = cauchy[0]
    eps2 = cauchy[1]
    epsilon2 = max(max(eps2), -min(eps2))
    b_euler = methods.BetterEuler(func, h, t0, y0, N, Yt)
    y3 = b_euler[0]
    eps3 = b_euler[1]
    epsilon3 = max(max(eps3), -min(eps3))
    runge = methods.RK4(func, h, t0, y0, N, Yt)
    y4 = runge[0]
    eps4 = runge[1]
    epsilon4 = max(max(eps4), -min(eps4))
    for i in range(N + 1):
        print('Yt[', i, '] = ', Yt(t0 + i * h))
    print('\nEuler rule:\n', y1, '\nEps = ', epsilon1,
          '\nEuler-Cauchy rule:\n', y2, '\nEps = ', epsilon2,
          '\nBetter Euler rule:\n', y3, '\nEps = ', epsilon3,
          '\nRunge-Kutta rule:\n', y4, '\nEps = ', epsilon4)

    Ytv = np.vectorize(Yt)
```

```
    plt.grid(True)
    xx = np.linspace(t0, T, 100)
    plt.plot(xx, Ytv(xx), color='red', label='Original')
    x = np.linspace(t0, T, N + 1)
    plt.plot(x, y1, color='blue', label='Euler', ls='', marker='.', markersize='4')
    plt.plot(x, y2, color='green', label='Euler-Cauchy', ls='', marker='.', markersize='4')
    plt.plot(x, y3, color='cyan', label='Better Euler', ls='', marker='.', markersize='4')
    plt.plot(x, y4, color='black', label='Runge-Kutta4', ls='', marker='.', markersize='4')
    plt.show()


def second():
    # Пункт 1
    N = 10
    t0 = 0
    T = 1
    y0 = 0
    h = (T - t0) / N

    Runge_Kutta_3 = methods.RK3(f2, h, t0, y0, N, Yt2_2)
    y5 = Runge_Kutta_3[0]
    print('Task 1:\n', y5, '\n')
    # График
    Ytv2 = np.vectorize(Yt2)
    plt.grid(True)
    xx2 = np.linspace(t0, T, 100)
    plt.plot(xx2, Ytv2(xx2), color='red', label='Original')
    x2 = np.linspace(t0, T, N + 1)
    plt.plot(x2, y5, color='blue', label='RK4', ls='', marker='.', markersize='4')
    plt.show()

    # Пункт 2
    N_2 = N
    t0_2 = 0
    T_2 = 1.5
    y0_2 = 1
    h_2 = (T_2 - t0_2) / N_2
    # Точное решение 2 части
    yt = [0] * (N_2 + 1)
    for i in range(0, N_2 + 1):
        yt[i] = (Yt2_2(t0_2 + i * h_2))
    print('\nYt2_2:\n', yt, '\n')

    Runge_Kutta_3_2 = methods.RK3(f2_2, h_2, t0_2, y0_2, N_2, Yt2_2)
    y5_2 = Runge_Kutta_3_2[0]
    print('\nTask 2:\n', y5_2, '\n')
    # График
    Ytv2_2 = np.vectorize(Yt2_2)
    plt.grid(True)
    xx2_2 = np.linspace(t0_2, T_2, 100)
    plt.plot(xx2_2, Ytv2_2(xx2_2), color='red', label='Original')
    x2_2 = np.linspace(t0_2, T_2, N_2 + 1)
    plt.plot(x2_2, y5_2, color='blue', label='RK3', ls='', marker='.', markersize='4')
    plt.show()

    # Пункт 3
    t0_3 = 0
    T_3 = 1.5
    y0_3 = 1
    N_3 = round(N / 2)
    h_3 = (T_3 - t0_3) / N_3
    Runge_Kutta_3_3 = methods.RK3(f2_2, h_3, t0_3, y0_3, N_3, Yt2_2)
    y5_3 = Runge_Kutta_3_3[0]
    print('\nTask 3:\n', y5_3, '\n')
```

```
# График
xx2_3 = np.linspace(t0_3, T_3, 100)
plt.plot(xx2_3, Ytv2_2(xx2_3), color='red', label='Original')
x2_3 = np.linspace(t0_3, T_3, N_3 + 1)
plt.plot(x2_3, y5_3, color='blue', label='RK32', ls='', marker='.', markersize='4')
plt.show()

r_3 = [0] * (N_3 + 1)
tmp_3 = [0] * (N_3 + 1)
p_3 = 3
n_3 = 2 ** p_3 - 1
i_1 = j_1 = 0
while (i_1 < N_2) & (j_1 < N_3):
    tmp_3[j_1] = y5_2[i_1]
    j_1 += 1
    i_1 += 2
for j in range(0, N_3 + 1):
    r_3[j] = abs((tmp_3[j] - y5_3[j]) / n_3)
R_3 = max(r_3)
print('\nR(RK 3) = ', R_3, '\n')

# Пункт 4
t0_4 = 0
T_4 = 1.5
y0_4 = 1
N_4 = 200
h_4 = (T_4 - t0_4) / N_4
N_42 = round(N_4 / 2)
h_42 = (T_4 - t0_4) / N_42
Runge_Kutta_3_4 = methods.RK3(f2_2, h_4, t0_4, y0_4, N_4, Yt2_2)
Runge_Kutta_3_42 = methods.RK3(f2_2, h_42, t0_4, y0_4, N_42, Yt2_2)
y5_5 = Runge_Kutta_3_4[0]
y5_52 = Runge_Kutta_3_42[0]
r_4 = [0] * (N_42 + 1)
tmp_4 = [0] * (N_42 + 1)
n_4 = 2 ** p_3 - 1
i_2 = j_2 = 0
while (i_2 < N_4) & (j_2 < N_42):
    tmp_4[j_2] = y5_5[i_2]
    j_2 += 1
    i_2 += 2
for j in range(0, N_42 + 1):
    r_4[j] = abs((tmp_4[j] - y5_52[j]) / n_4)
R_4 = max(r_4)

print('\nTask 4:\n', y5_5, '\n')
print('\nTask 4 (h = 2h2):\n', y5_52, '\n')
# График
xx2_4 = np.linspace(t0_4, T_4, 100)
plt.plot(xx2_4, Ytv2_2(xx2_4), color='red', label='Original')
x2_4 = np.linspace(t0_4, T_4, N_4 + 1)
plt.plot(x2_4, y5_5, color='blue', label='RK3', ls='', marker='.', markersize='4')
plt.show()
xx2_42 = np.linspace(t0_4, T_4, 100)
plt.plot(xx2_42, Ytv2_2(xx2_42), color='red', label='Original')
x2_42 = np.linspace(t0_4, T_4, N_42 + 1)
plt.plot(x2_42, y5_52, color='green', label='RK3', ls='', marker='.', markersize='4')
plt.show()
print('\nRh(RK 3) = ', R_4, '\n')
epsilon_4 = max(max(Runge_Kutta_3_4[1]), -min(Runge_Kutta_3_4[1]))
print('\nEpsilon = ', epsilon_4, '\n')

# Пункт 5
t0_5 = 0
```

```
T_5 = 1.5
y0_5 = 1
N_5 = N
h_5 = (T_5 - t0_5) / N_5
N_52 = round(N / 2)
h_52 = (T_5 - t0_5) / N_52
Runge_Kutta_4_5 = methods.RK4(f2_2, h_5, t0_5, y0_5, N_5, Yt2_2)
y5_5 = Runge_Kutta_4_5[0]
Runge_Kutta_4_52 = methods.RK4(f2_2, h_52, t0_5, y0_5, N_52, Yt2_2)
y5_52 = Runge_Kutta_4_52[0]
print('\nTask 5:\n', y5_5, '\n')
print('\nTask 5 (h = 2h0):\n', y5_52, '\n')
# График
xx2_5 = np.linspace(t0_5, T_5, 100)
plt.plot(xx2_5, Ytv2_2(xx2_5), color='red', label='Original')
x2_5 = np.linspace(t0_5, T_5, N_5 + 1)
plt.plot(x2_5, y5_5, color='blue', label='RK4', ls='', marker='.', markersize='4')
plt.show()
xx2_52 = np.linspace(t0_5, T_5, 100)
plt.plot(xx2_52, Ytv2_2(xx2_52), color='red', label='Original')
x2_52 = np.linspace(t0_5, T_5, N_52 + 1)
plt.plot(x2_52, y5_52, color='green', label='RK4', ls='', marker='.', markersize='4')
plt.show()

r_5 = [0] * (N_52 + 1)
tmp_5 = [0] * (N_52 + 1)
p_5 = 4
n_5 = 2 ** p_5 - 1
i_5 = j_5 = 0
while (i_5 < N_5) & (j_5 < N_52):
    tmp_5[j_5] = y5_5[i_5]
    j_5 += 1
    i_5 += 2
for j in range(0, N_52 + 1):
    r_5[j] = abs((tmp_5[j] - y5_52[j]) / n_5)
R_5 = max(r_5)
print('R(RK 4) = ', R_5, '\n')

# Пункт 6
t0_6 = 0
T_6 = 1.5
y0_6 = 1
N_6 = 80
h_6 = (T_6 - t0_6) / N_6
N_62 = round(N_6 / 2)
h_62 = (T_6 - t0_6) / N_62
Runge_Kutta_4_6 = methods.RK4(f2_2, h_6, t0_6, y0_6, N_6, Yt2_2)
Runge_Kutta_4_62 = methods.RK4(f2_2, h_62, t0_6, y0_6, N_62, Yt2_2)
y5_6 = Runge_Kutta_4_6[0]
y5_62 = Runge_Kutta_4_62[0]
r_6 = [0] * (N_62 + 1)
tmp_6 = [0] * (N_62 + 1)
p_6 = 4
n_6 = 2 ** p_6 - 1
i_6 = j_6 = 0
while (i_6 < N_6) & (j_6 < N_62):
    tmp_6[j_6] = y5_6[i_6]
    j_6 += 1
    i_6 += 2
for j in range(0, N_62 + 1):
    r_6[j] = abs((tmp_6[j] - y5_62[j]) / n_6)
R_6 = max(r_6)
```

```
    print('\nTask 6:\n', y5_6, '\n')
    print('\nTask 6 (h = 2h2):\n', y5_62, '\n')
    # График
    xx2_6 = np.linspace(t0_6, T_6, 100)
    plt.plot(xx2_6, Ytv2_2(xx2_6), color='red', label='Original')
    x2_6 = np.linspace(t0_6, T_6, N_6 + 1)
    plt.plot(x2_6, y5_6, color='blue', label='RK4', ls='', marker='.', markersize='2')
    plt.show()
    xx2_62 = np.linspace(t0_6, T_6, 100)
    plt.plot(xx2_62, Ytv2_2(xx2_62), color='red', label='Original')
    x2_62 = np.linspace(t0_6, T_6, N_62 + 1)
    plt.plot(x2_62, y5_62, color='green', label='RK4', ls='', marker='.', markersize='2')
    plt.show()
    print('\nR4h(RK 4) = ', R_6, '\n')
    epsilon = max(max(Runge_Kutta_4_6[1]), -min(Runge_Kutta_4_6[1]))
    print('\nEpsison = ', epsilon, '\n')

first()
second()
```

methods.py

```
def Euler(f, h, t0, y0, N, Yt):
    y = [0] * (N + 1)
    eps = [0] * (N + 1)
    y[0] = y0
    eps[0] = abs(Yt(t0) - y0)
    for i in range(0, N):
        y[i+1] = y[i] + h * f(t0 + i * h, y[i])
        eps[i + 1] = Yt(t0 + (i + 1) * h) - y[i + 1]
    return y, eps


def EulerCauchy(f, h, t0, y0, N, Yt):
    y = [0] * (N + 1)
    eps = [0] * (N + 1)
    y[0] = y0
    eps[0] = abs(Yt(t0) - y0)
    for i in range(0, N):
        y_tmp = y[i] + h * f(t0 + i * h, y[i])
        y[i + 1] = y[i] + 0.5 * h * \
                   (f(t0 + i * h, y[i]) + f(t0 + (i + 1) * h, y_tmp))
        eps[i + 1] = Yt(t0 + (i + 1) * h) - y[i + 1]
    return y, eps


def BetterEuler(f, h, t0, y0, N, Yt):
    y = [0] * (N + 1)
    eps = [0] * (N + 1)
    y[0] = y0
    eps[0] = abs(Yt(t0) - y0)
    for i in range(0, N):
        y[i + 1] = y[i] + h * \
                   f(t0 + i * h + 0.5 * h, y[i] + 0.5 * h * f(t0 + i * h, y[i]))
        eps[i + 1] = Yt(t0 + (i + 1) * h) - y[i + 1]
    return y, eps


def RK4(f, h, t0, y0, N, Yt):
    y = [0] * (N + 1)
    eps = [0] * (N + 1)
    y[0] = y0
    eps[0] = abs(Yt(t0) - y0)
    for i in range(0, N):
        K1 = f(t0 + i * h, y[i])
```

```
        K2 = f(t0 + i * h + h / 2, y[i] + h * K1 / 2)
        K3 = f(t0 + i * h + h / 2, y[i] + h * K2 / 2)
        K4 = f(t0 + i * h + h, y[i] + h * K3)
        y[i + 1] = y[i] + h / 6 * (K1 + 2 * K2 + 2 * K3 + K4)
        eps[i + 1] = Yt(t0 + (i + 1) * h) - y[i + 1]
    return y, eps


def RK3(f, h, t0, y0, N, Yt):
    y = [0] * (N + 1)
    eps = [0] * (N + 1)
    y[0] = y0
    eps[0] = abs(Yt(t0) - y0)
    # print('h = ', h, '\n')
    for i in range(0, N):
        K1 = f(round(t0 + i * h, 4), round(y[i], 8))
        K2 = f(round(t0 + i * h + h / 3, 4), round(y[i] + h * K1 / 3, 8))
        K3 = f(round(t0 + i * h + 2 * h / 3, 4), round(y[i] + 2 * h * K2 / 3, 8))
        y[i + 1] = round(y[i] + 0.25 * h * (K1 + 3 * K3), 8)
        eps[i + 1] = Yt(t0 + (i + 1) * h) - y[i + 1]
    return y, eps
```