Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе № 3**

Дисциплина: Вычислительная математика

Выполнил студент гр. 3530901/10001 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.Л. Симоновский

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Цыган

(подпись)

“28” февраля 2023 г.

Санкт-Петербург

2023

Оглавление

[Задание: 2](#_Toc128509334)

[Инструменты: 2](#_Toc128509335)

[Ход выполнения работы: 2](#_Toc128509336)

[***Порядок действий:*** 2](#_Toc128509337)

[***Первая задача:*** 2](#_Toc128509338)

[Полная формулировка: 2](#_Toc128509339)

[Решение: 3](#_Toc128509340)

[***Вторая задача:*** 3](#_Toc128509341)

[Полная формулировка: 3](#_Toc128509342)

[Решение: 3](#_Toc128509343)

[Результат: 6](#_Toc128509344)

[Вывод: 9](#_Toc128509345)

[Листинг кода: 10](#_Toc128509346)

[Ссылки: 12](#_Toc128509347)

# Задание:

Вариант 11:

Привести дифференциальное уравнение:   
к системе двух дифференциальных уравнений первого порядка.

Начальные условия: ;

Точное решение:

Решить на интервале:

1. Используя программу RKF45 с шагом печати = 0.1 и выбранной вами погрешностью EPS в диапазоне 0.001 – 0.00001, а также составить собственную программу и решить с шагом интегрирования = 0.1
2. Используя метод Рунге-Кутты 3-й степени точности.

Сравнить результаты, полученные заданными приближенными способами, с точным решением.

Исследовать влияние величины шага интегрирования на величины локальной и глобальной погрешностей решения заданного уравнения для чего решить уравнение, используя 2 – 3 значения шага интегрирования, существенно меньшие исходной величины 0.1 (например, = 0.05; = 0.025; = 0.0125)

# Инструменты:

Для работы был выбран язык программирования Python версии 3.11 по причине удобства его использования для поставленной задачи. Были выбраны следующие библиотеки:

* NumPy – для большей скорости расчетов и простоты обработки
* SciPy – для функции расчета решения дифура
* PrettyTable – для красивого вывода таблицы в консоль
* MatPlotLib – для вывода графиков

# Ход выполнения работы:

## ***Порядок действий:***

Поставленное задание легко можно разбить на две глобальные задачи:

1. Сведение поставленной задачи к системе двух дифференциальных уравнений первого порядка.
2. Получить решение используя RKF45 и методы Рунге-Кутты 3-й степени

## ***Первая задача:***

### Полная формулировка:

Привести дифференциальное уравнение:   
к системе двух дифференциальных уравнений первого порядка.

### Решение:

В начале сделаем коэффициент при старшей степени равным 1, для этого поделим уравнение на t:

Возьмем и , получим:

Решение этого уравнение эквивалентно решению системы , где , A – матрица Фробениуса вида: . Таким образом получим систему:

## ***Вторая задача:***

### Полная формулировка:

Решить систему дифференциальных уравнений перового порядка.

Начальные условия: ;

Точное решение:

Решить на интервале:

1. Используя программу RKF45 с шагом печати = 0.1 и выбранной вами погрешностью EPS в диапазоне 0.001 – 0.00001, а также составить собственную программу и решить с шагом интегрирования = 0.1
2. Используя метод Рунге-Кутты 3-й степени точности.

Сравнить результаты, полученные заданными приближенными способами, с точным решением.

Исследовать влияние величины шага интегрирования на величины локальной и глобальной погрешностей решения заданного уравнения для чего решить уравнение, используя 2 – 3 значения шага интегрирования, существенно меньшие исходной величины 0.1 (например, = 0.05; = 0.025; = 0.0125)

### Решение:

Для дальнейшего использования сразу же зададим функцию, для получения значений системы уравнений:

def f(t, X):dX = np.zeros(X.shape)  
 dX[0] = X[1]  
 dX[1] = (t + 1) / t \* X[1] + 2 \* (t - 1) / t \* X[0]  
 return dX

Так же нам понадобится функция для получения точно значения решения (чтоб сравнивать погрешности):

def g(T):return np.e \*\* (2 \* T)

Далее нам понадобится функция, которая будет моделировать RKF45, к счастью в библиотеке scipy уже имеется подходящий вариант, правда требующий дополнительной настройки. Выполним её в отдельной функции:

def rkf45(f, T, X0):runge = ode(f).set\_integrator('dopri5', atol=0.0001).set\_initial\_value(X0, T[0])  
 X = [X0, \*[runge.integrate(T[i]) for i in range(1, len(T))]]  
 return np.array([i[0] for i in X])

Метод Рунге-Кутты по заданию необходимо написать самостоятельно. Для начала вспомним, как выглядит метод Рунге-Кутты третьей степени:

Теперь необходимо реализовать его в виде метода:

def Runge\_Kutta(f, T, X0):X = np.zeros((len(T), len(X0)))  
 X[0] = X0  
 h = T[1] - T[0]  
 for i in range(0, len(T) - 1):  
 k\_1 = h \* f(T[i], X[i])  
 k\_2 = h \* f(T[i] + h / 2, X[i] + k\_1 / 2)  
 k\_3 = h \* f(T[i] + 3 \* h / 4, X[i] + 3 \* k\_2 / 4)  
 X[i + 1] = (X[i] + (2 \* k\_1 + 3 \* k\_2 + 4 \* k\_3) / 9)  
 return X[:, 0]

Все методы, необходимые для подсчета реализованы. Создадим отдельную функцию для подсчета решения с разным шагом (т.к. этого требует задание), в ней сразу же зададим начальные значения, узлы, по которым будет искаться решение и значение функции в этих точках (для дальнейшего подсчета погрешности):

def evaluate(h):# Начальные значения  
 X0 = np.array([np.e \*\* 2, 2 \* np.e \*\* 2])  
 # Значения в узлах  
 T = np.arange(1, 2 + h, h)  
 Y = g(T)

Воспользовавшись методами, которые были приведены выше, выполним расчеты:

# Расчет RKF45  
Y\_RKF45 = rkf45(f, T, X0)  
# Расчет Рунге-Кутты  
Y\_Runge\_Kutta = Runge\_Kutta(f, T, X0)  
# Погрешности

Y\_RKF45\_error = Y - Y\_RKF45  
Y\_Runge\_Kutta\_error = Y - Y\_Runge\_Kutta

Далее необходимо вывести полученные значения на экран, используем для этого библиотеку MatPlotLib. Отдельно выведем графики погрешности и полученные значения:

def print\_one\_graph(t, y, title, id, count\_graphs):  
 *"""  
 Функция для отрисовки одного графика  
 """* plt.subplot(1, count\_graphs, id)  
 plt.xlabel('t')  
 plt.ylabel('y')  
 plt.grid()  
 plt.title(title)  
 plt.plot(t, y, '-o')  
  
  
def print\_graph(t\_find, y\_real, Y\_RKF45, Y\_Runge\_Kutta, h):  
 *"""  
 Функция для отрисовки всех графиков  
 """* mpl.use('TkAgg')  
 plt.figure(figsize=(15, 4))  
 print\_one\_graph(t\_find, y\_real, 'Исходный график', 1, 3)  
 print\_one\_graph(t\_find, Y\_RKF45, 'График RKF45', 2, 3)  
 print\_one\_graph(t\_find, Y\_Runge\_Kutta, 'График Рунге-Кутты', 3, 3)  
 plt.savefig(f"Graphs\_{h}.jpg")  
 plt.show()  
  
  
def print\_error\_graph(t\_find, Y\_RKF45\_error, Y\_Runge\_Kutta\_error, h):  
 *"""  
 Функция для отрисовки погрешности  
 """* mpl.use('TkAgg')  
 plt.figure(figsize=(15, 4))  
 # Собственно сам график  
 print\_one\_graph(t\_find, Y\_RKF45\_error, 'Погрешность RKF45', 1, 2)  
 print\_one\_graph(t\_find, Y\_Runge\_Kutta\_error, 'Погрешность Рунге-Кутты', 2, 2)  
 plt.savefig(f"Error\_{h}.jpg")  
 plt.show()

Вызовем их из функции evaluate, для отрисовки всех графиков:

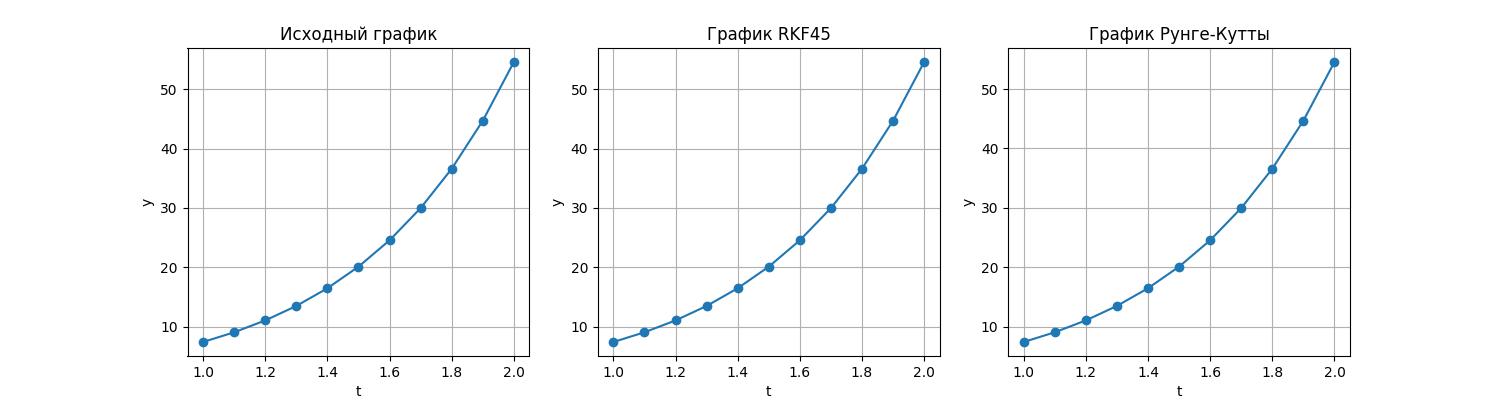
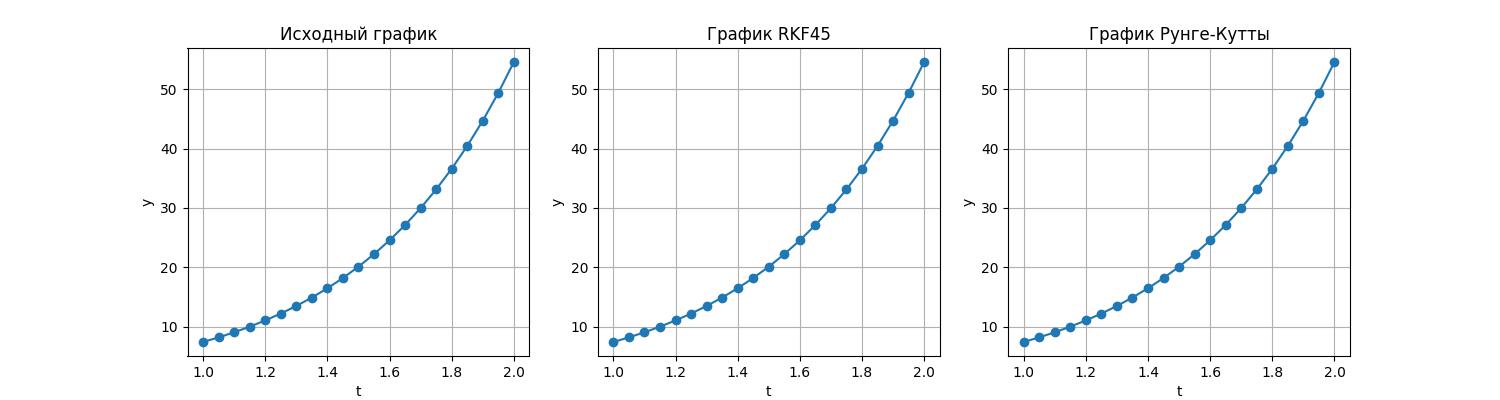
# Рисуем графики  
print\_graph(T, Y, Y\_RKF45, Y\_Runge\_Kutta, h)  
print\_error\_graph(T, Y\_RKF45\_error, Y\_Runge\_Kutta\_error, h)

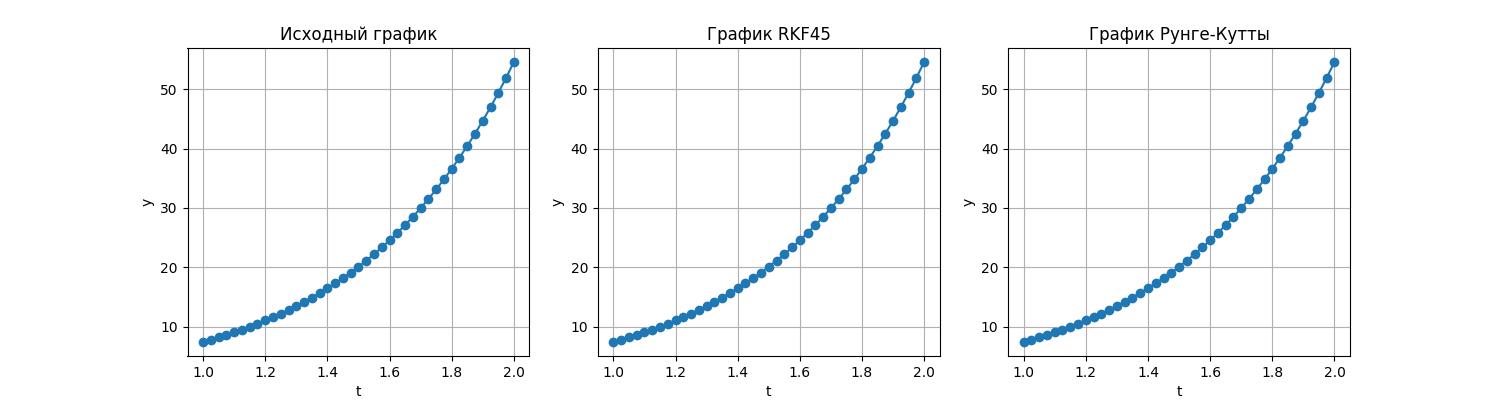
Далее для дополнительного анализа выведем значения в консоль, используя библиотеку prettytable:

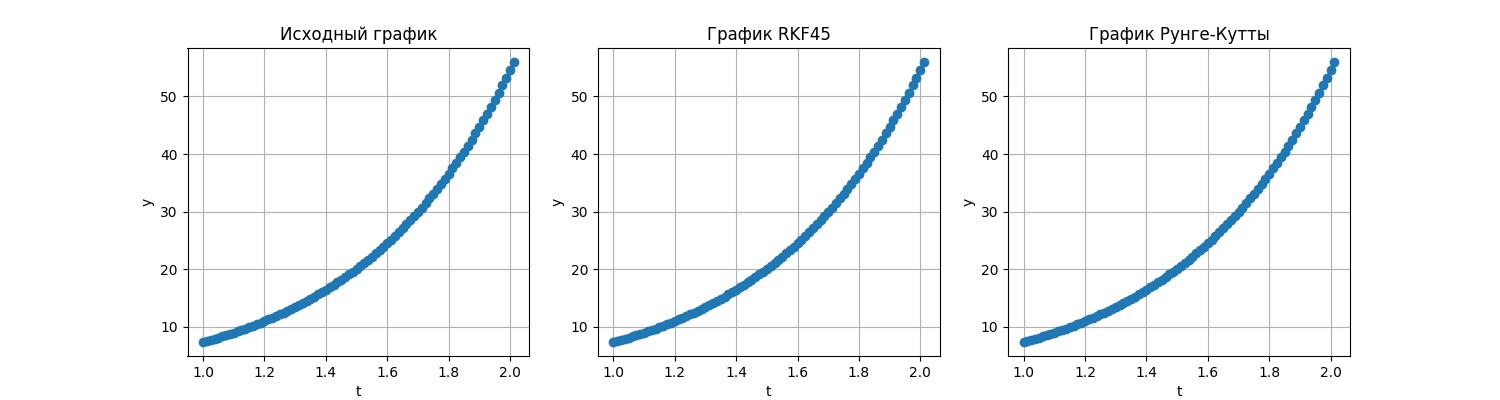
def print\_table(t\_find, y\_real, Y\_RKF45, Y\_RKF45\_error, Y\_Runge\_Kutta, Y\_Runge\_Kutta\_error, h):print(f'h = {h}')  
 koef = {0.1: 1, 0.05: 2, 0.025: 4, 0.0125: 8}.get(h)  
 pt = PrettyTable()  
 pt.add\_column('t', [f'{i:.1f}' for i in t\_find[::koef]])  
 pt.add\_column('real y', [f'{i:.15f}' for i in y\_real[::koef]])  
 pt.add\_column('RKF45 y', [f'{i:.15f}' for i in Y\_RKF45[::koef]])  
 pt.add\_column('Delta RKF45 y', [f'{i:.15f}' for i in Y\_RKF45\_error[::koef]])  
 pt.add\_column('Runge Kutta y', [f'{i:.15f}' for i in Y\_Runge\_Kutta[::koef]])  
 pt.add\_column('Delta Runge Kutta y', [f'{i:.15f}' for i in Y\_Runge\_Kutta\_error[::koef]])  
 print(pt)  
 print('First step of RKF45:', Y\_RKF45\_error[1])  
 print('First step of Runge Kutta:', Y\_Runge\_Kutta\_error[1])  
 print('Global of RKF45:', Y\_RKF45\_error[::koef].sum())  
 print('Global of Runge Kutta:', Y\_Runge\_Kutta\_error[::koef].sum())  
 print('h^4 is about:', h \*\* 4)  
 print('h^4 / Runge Kutta first step:', h \*\* 4 / Y\_Runge\_Kutta\_error[1])  
 print('=' \* 110)

### Результат:

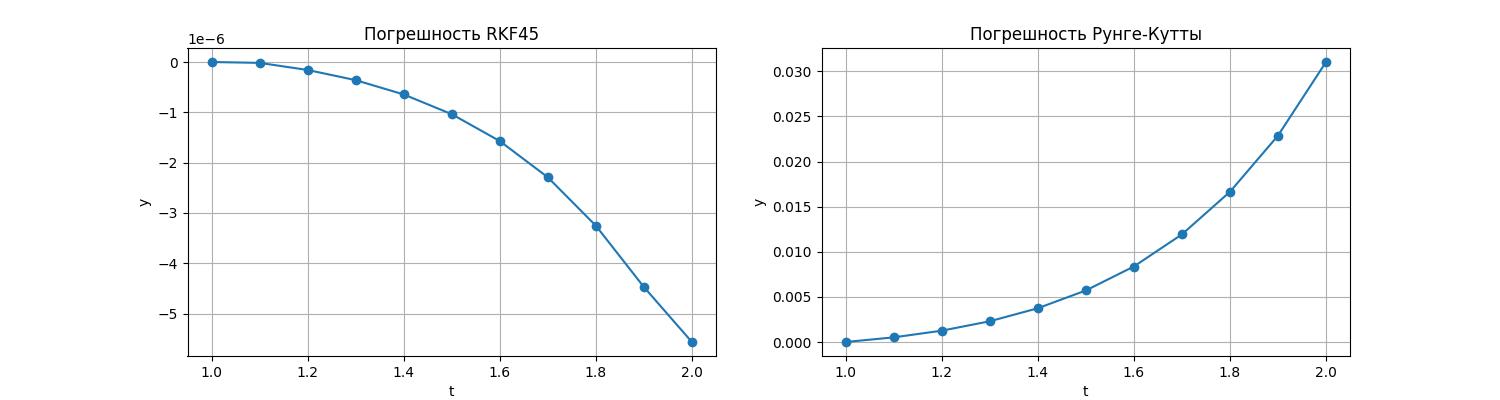
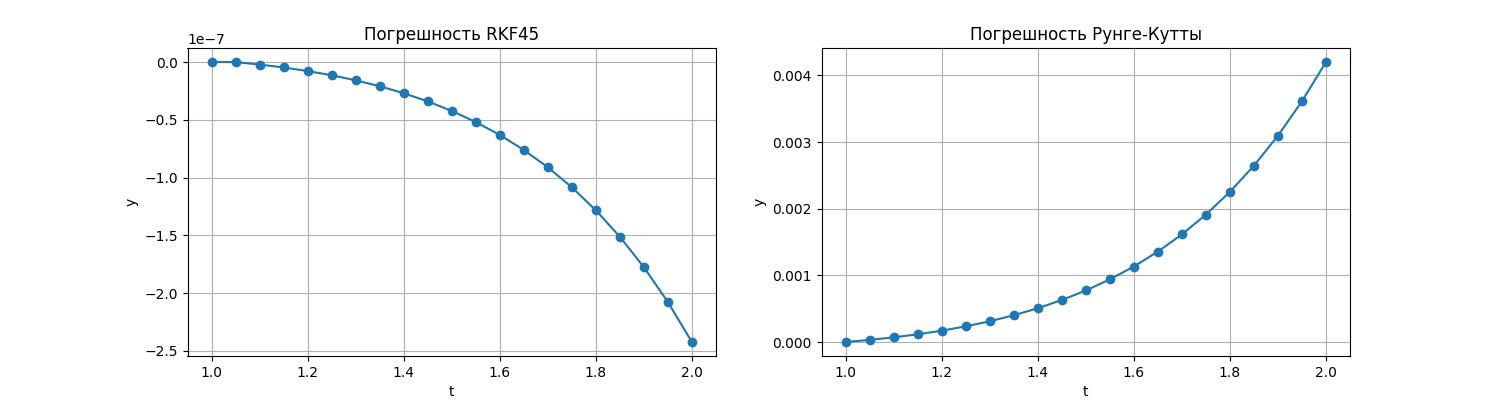
Как и ожидалось, полученные графики практически не отличаются:

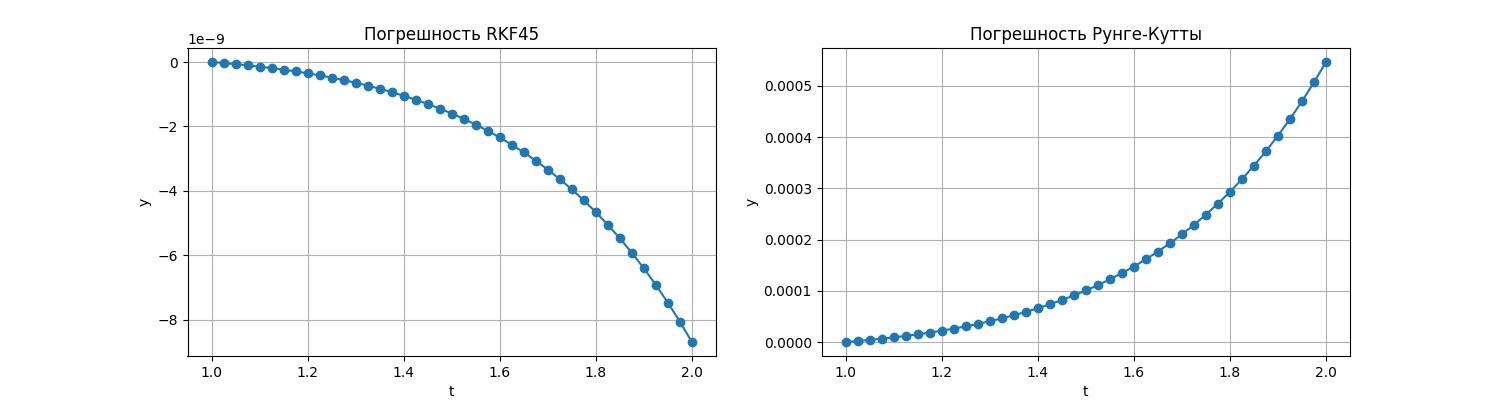
  
Рис. 1. График решения при h = 0.1  
  
Рис. 2. График решения при h = 0.05

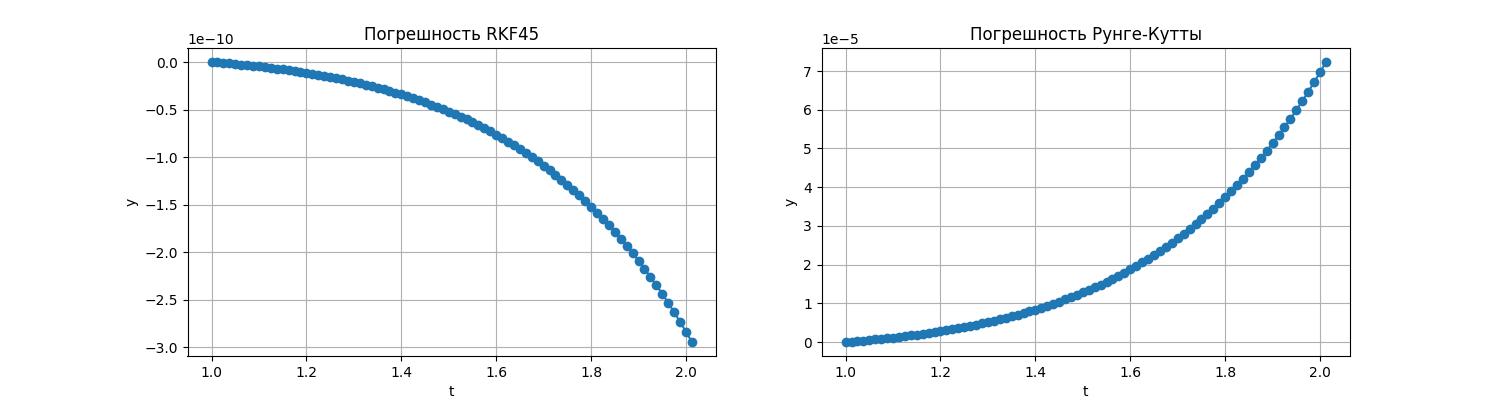
  
Рис. 3. График решения при h = 0.025

  
Рис. 4. График решения при h = 0.0125

Больший интерес представляют графики погрешности:

  
Рис. 5. График погрешности при h = 0.1  
  
Рис. 6. График погрешности при h = 0.05

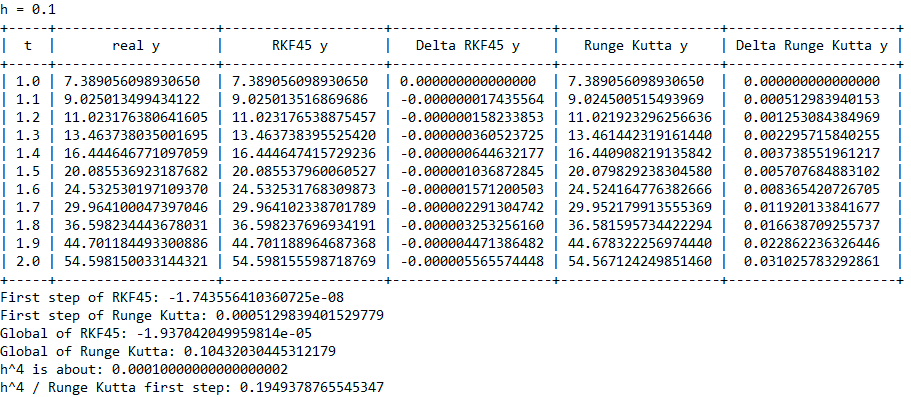
  
Рис. 7. График погрешности при h = 0.025

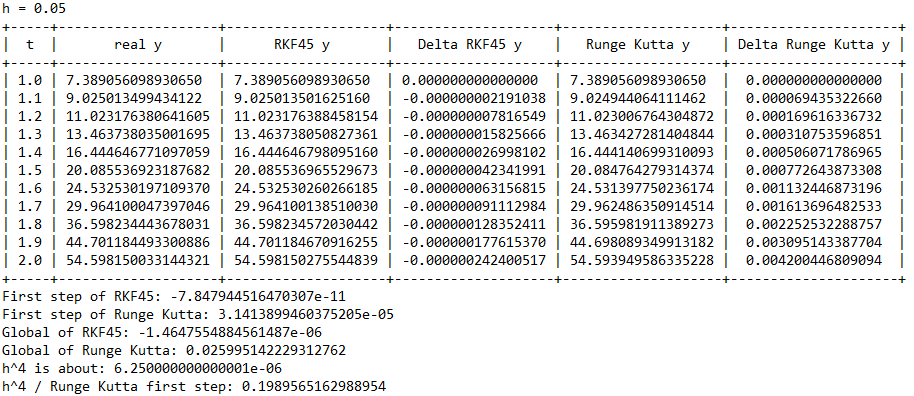
  
Рис. 8. График погрешности при h = 0.0125

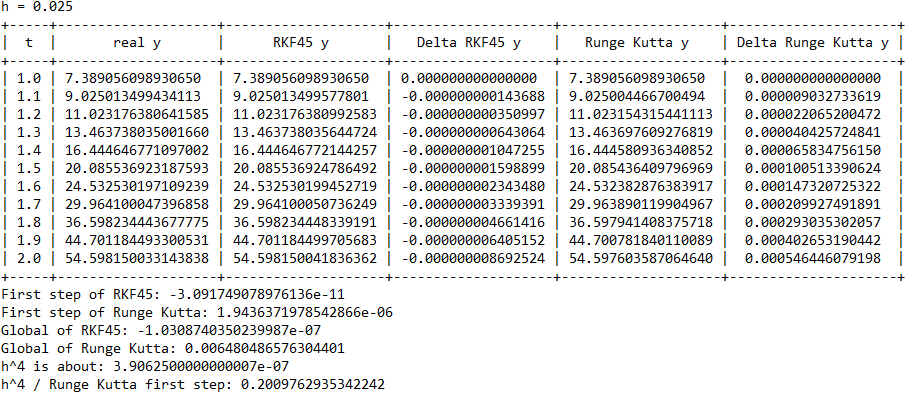
Как можно заметить, форма самого графика остается одинаковой, однако порядок сильно меняется, при изменении шага.

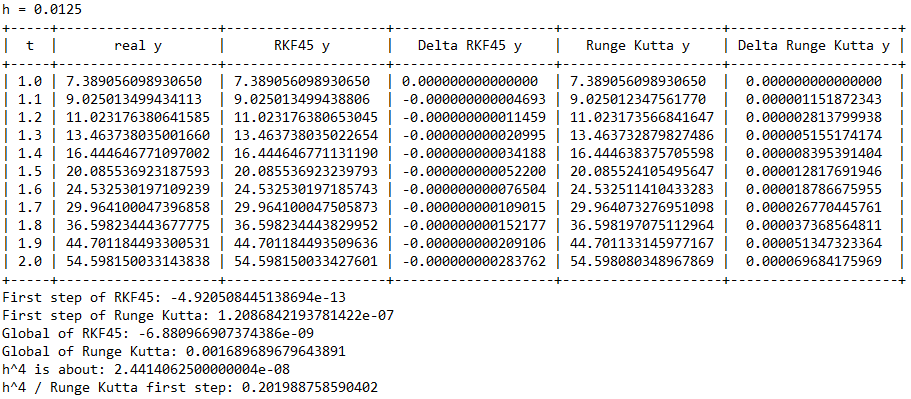
Так же даже по графику заметно, что погрешность метода Рунге-Кутты третьей степени значительно проигрывает программе RKF45, что достаточно ожидаемо, ведь в основе этой программы лежат методы Рунге-Кутты четвертой и пятой степени точности.

Для анализа зависимости шага интегрирования и величины локальной и глобальной погрешностей решения заданного уравнения обратимся к численным результатам исследования:

  
Рис. 9. Результат исследования при h = 0.1

   
Рис. 10. Результат исследования при h = 0.05

   
Рис. 11. Результат исследования при h = 0.025

   
Рис. 12. Результат исследования при h = 0.0125

Можно заметить, что локальная погрешность первого шага пропорциональна четвертой степени шага, как и было предсказано.

Так же из приведенных результатов более заметна разница между методами Рунге-Кутты 3 степени точности и программой RKF45, как после первого шага, так и при подсчете глобальной погрешности, что сильнее проявляется при уменьшении шага.

# Вывод:

В ходе выполненной работы мы привели линейное дифференциальное уравнение к системе двух дифференциальных уравнений первого порядка, после чего решили эту систему при заданных НУ, используя методы Рунге-Кутты третьей степени точности, а также программу RKF45. Была найдена зависимость шага интегрирования h и величины глобальной и локальной погрешности а также наглядно продемонстрирована разница между используемыми методами.

# Листинг кода:

import numpy as np  
from scipy.integrate import ode  
import matplotlib as mpl  
import matplotlib.pyplot as plt  
from prettytable import PrettyTable  
  
  
def rkf45(f, T, X0):  
 *"""  
 Решает `x' = f(t, x)` для каждого `t` в `T`  
 С начальным значением `X0`, используя аналог rkf45  
 """* runge = ode(f).set\_integrator('dopri5', atol=0.0001).set\_initial\_value(X0, T[0])  
 X = [X0, \*[runge.integrate(T[i]) for i in range(1, len(T))]]  
 return np.array([i[0] for i in X])  
  
  
def Runge\_Kutta(f, T, X0):  
 *"""  
 Решает `x' = f(t, x)` для каждого `t` в `T`  
 С начальным значением `X0`, используя формулы Рунге-Кутты 3 степени  
 """* X = np.zeros((len(T), len(X0)))  
 X[0] = X0  
 h = T[1] - T[0]  
 for i in range(0, len(T) - 1):  
 k\_1 = h \* f(T[i], X[i])  
 k\_2 = h \* f(T[i] + h / 2, X[i] + k\_1 / 2)  
 k\_3 = h \* f(T[i] + 3 \* h / 4, X[i] + 3 \* k\_2 / 4)  
 X[i + 1] = (X[i] + (2 \* k\_1 + 3 \* k\_2 + 4 \* k\_3) / 9)  
 return X[:, 0]  
  
  
def f(t, X):  
 *"""  
 Правая часть `x' = f(t, x)`.  
 """* dX = np.zeros(X.shape)  
 dX[0] = X[1]  
 dX[1] = (t + 1) / t \* X[1] + 2 \* (t - 1) / t \* X[0]  
 return dX  
  
  
def g(T):  
 *"""  
 Точное решение  
 """* return np.e \*\* (2 \* T)  
  
  
def print\_one\_graph(t, y, title, id, count\_graphs):  
 *"""  
 Функция для отрисовки одного графика  
 """* plt.subplot(1, count\_graphs, id)  
 plt.xlabel('t')  
 plt.ylabel('y')  
 plt.grid()  
 plt.title(title)  
 plt.plot(t, y, '-o')

def print\_graph(t\_find, y\_real, Y\_RKF45, Y\_Runge\_Kutta, h):  
 *"""  
 Функция для отрисовки всех графиков  
 """* mpl.use('TkAgg')  
 plt.figure(figsize=(15, 4))  
 print\_one\_graph(t\_find, y\_real, 'Исходный график', 1, 3)  
 print\_one\_graph(t\_find, Y\_RKF45, 'График RKF45', 2, 3)  
 print\_one\_graph(t\_find, Y\_Runge\_Kutta, 'График Рунге-Кутты', 3, 3)  
 plt.savefig(f"Graphs\_{h}.jpg")  
 plt.show()  
  
  
def print\_error\_graph(t\_find, Y\_RKF45\_error, Y\_Runge\_Kutta\_error, h):  
 *"""  
 Функция для отрисовки погрешности  
 """* mpl.use('TkAgg')  
 plt.figure(figsize=(15, 4))  
 # Собственно сам график  
 print\_one\_graph(t\_find, Y\_RKF45\_error, 'Погрешность RKF45', 1, 2)  
 print\_one\_graph(t\_find, Y\_Runge\_Kutta\_error, 'Погрешность Рунге-Кутты', 2, 2)  
 plt.savefig(f"Error\_{h}.jpg")  
 plt.show()  
  
  
def print\_table(t\_find, y\_real, Y\_RKF45, Y\_RKF45\_error, Y\_Runge\_Kutta, Y\_Runge\_Kutta\_error, h):  
 *"""  
 Функция для отрисовки таблицы  
 """* print(f'h = {h}')  
 koef = {0.1: 1, 0.05: 2, 0.025: 4, 0.0125: 8}.get(h)  
 pt = PrettyTable()  
 pt.add\_column('t', [f'{i:.1f}' for i in t\_find[::koef]])  
 pt.add\_column('real y', [f'{i:.15f}' for i in y\_real[::koef]])  
 pt.add\_column('RKF45 y', [f'{i:.15f}' for i in Y\_RKF45[::koef]])  
 pt.add\_column('Delta RKF45 y', [f'{i:.15f}' for i in Y\_RKF45\_error[::koef]])  
 pt.add\_column('Runge Kutta y', [f'{i:.15f}' for i in Y\_Runge\_Kutta[::koef]])  
 pt.add\_column('Delta Runge Kutta y', [f'{i:.15f}' for i in Y\_Runge\_Kutta\_error[::koef]])  
 print(pt)  
 print('First step of RKF45:', Y\_RKF45\_error[1])  
 print('First step of Runge Kutta:', Y\_Runge\_Kutta\_error[1])  
 print('Global of RKF45:', Y\_RKF45\_error.sum())  
 print('Global of Runge Kutta:', Y\_Runge\_Kutta\_error.sum())  
 print('h^4 is about:', h \*\* 4)  
 print('h^4 / Runge Kutta first step:', h \*\* 4 / Y\_Runge\_Kutta\_error[1])  
 print('=' \* 110)

def evaluate(h):  
 *"""  
 Получение решения при разных шагах  
 """* # Начальные значения  
 X0 = np.array([np.e \*\* 2, 2 \* np.e \*\* 2])  
 # Значения в узлах  
 T = np.arange(1, 2 + h, h)  
 Y = g(T)  
 # Расчет RKF45  
 Y\_RKF45 = rkf45(f, T, X0)  
 # Расчет Рунге-Кутты  
 Y\_Runge\_Kutta = Runge\_Kutta(f, T, X0)  
 # Погрешности  
 Y\_RKF45\_error = Y - Y\_RKF45  
 Y\_Runge\_Kutta\_error = Y - Y\_Runge\_Kutta  
 # Рисуем графики  
 print\_graph(T, Y, Y\_RKF45, Y\_Runge\_Kutta, h)  
 print\_error\_graph(T, Y\_RKF45\_error, Y\_Runge\_Kutta\_error, h)  
 # Выводим данные в консоль  
 print\_table(T, Y, Y\_RKF45, Y\_RKF45\_error, Y\_Runge\_Kutta, Y\_Runge\_Kutta\_error, h)  
  
  
def main():  
 h = 0.1  
 for i in range(4):  
 evaluate(h)  
 h /= 2  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Ссылки:

Листинг код: [github.com](https://github.com/DafterT/comp_math_3)

Документация по SciPy: [docs.scipy.org](https://docs.scipy.org/doc/scipy/index.html)