**ИНСТИТУТ ТРАНСПОРТА И СВЯЗИ**



ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК И ТЕЛЕКОММУНИКАЦИЙ

КАФЕДРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ КОМПЬЮТЕРНЫХ СИСТЕМ

**Лабораторная работа №4**

по дисциплине

„Численные методы”

Тема: «Численные методы решения дифференциальных уравнений первого порядка.»

Выполнил: Денис Белов, Андрей Савкин, Евгений Хрущ

Рига.

2020.

**1)** **Формулировка задания**

В данной работе необходимо было реализовать два метода решения дифференциальных уравнений: метод Эйлера и индивидуальный метод: Рунге-Кутта 4-го порядка. Для реализации алгоритма требовалось предусмотреть ряд входных данных, таких как первый коэффициент дифференциального уравнения при х, второго коэффициента уравнения при х2, точность проводимого вычисления, начальное условие и правая граница интервала.

Для эталонного решения взяли

**2) Метод Эйлера**

class DifferentialEquationsSolver:

    def \_\_init\_\_(self, function, step\_h, exact\_ys=None):

        self.function = function

        self.step\_h = step\_h

        self.method\_name = ""

        self.exact\_ys = exact\_ys

    @staticmethod

    def compare\_errors(approx\_ys, exact\_ys):

        return [abs(y2 - y1) for y1, y2 in zip(approx\_ys[1:], exact\_ys[1:])]

    def \_get\_next\_y(self, current\_x, current\_y):

        return self.function(current\_x, current\_y)

    def \_get\_exact\_y\_and\_error(self, current\_x, current\_y, i):

        exact\_y = self.function(current\_x, current\_y)

        error = abs(current\_y - exact\_y)

        return exact\_y, error

    def \_table\_line(self, current\_x, current\_y, i):

        if self.exact\_ys is None:

            print(f'{"%0.8f" % current\_x:<16} | {"%0.8f" % current\_y:<16}')

        else:

            error = abs(current\_y - self.exact\_ys[i])

            print(f'{"%0.8f" % current\_x:<16} | {"%0.8f" % current\_y:<16} | {"%0.8f" % self.exact\_ys[i]:<16} | '

                  f'{"%0.8f" % error:<16}')

    def get\_xy(self, initial\_x, initial\_y, last\_x):

        xs = np.arange(initial\_x, last\_x + self.step\_h, self.step\_h)

        ys = []

        current\_y = initial\_y

        self.\_description\_line(initial\_x, initial\_y, last\_x)

        self.\_table\_header()

        for i, current\_x in enumerate(xs):

            self.\_table\_line(current\_x, current\_y, i)

            ys.append(current\_y)

            current\_y = self.\_get\_next\_y(current\_x, current\_y)

        return xs, ys

class Euler(DifferentialEquationsSolver):

    def \_\_init\_\_(self, function, step\_h, exact\_ys=None):

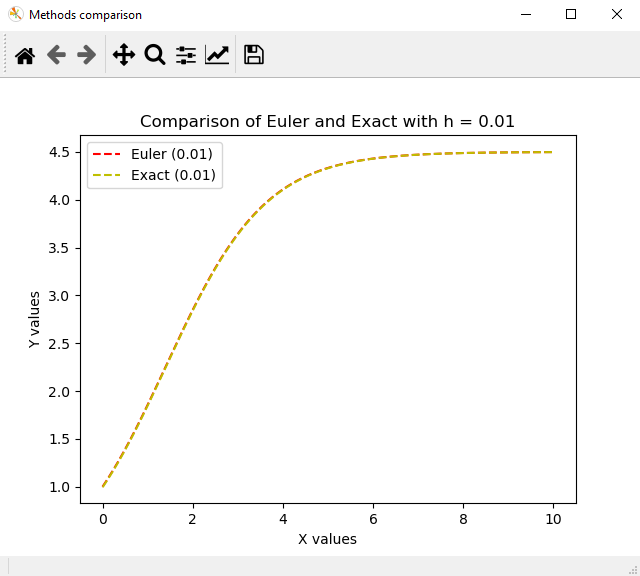
        super().\_\_init\_\_(function, step\_h, exact\_ys)

        self.method\_name = "Euler"

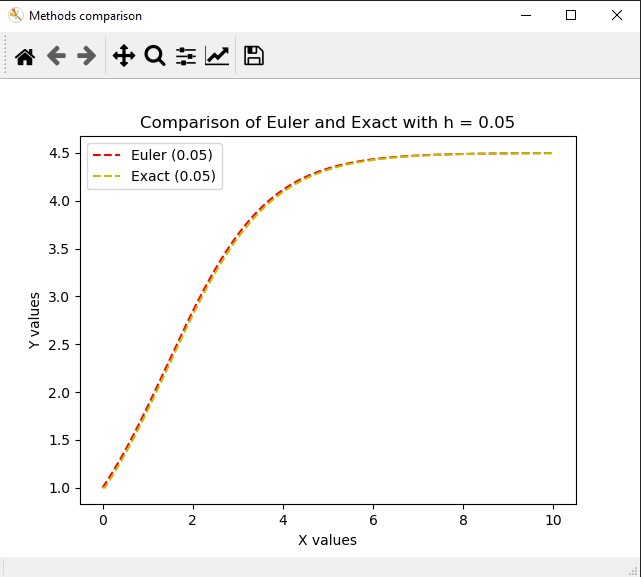
    def \_get\_next\_y(self, current\_x, current\_y):

        return current\_y + self.step\_h \* self.function(current\_x, current\_y)

**График 1:**



**График 2:**



**3)** **Индивидуальный метод: метод Рунге-Кутта 4-го порядка**

class DifferentialEquationsSolver:

    def \_\_init\_\_(self, function, step\_h, exact\_ys=None):

        self.function = function

        self.step\_h = step\_h

        self.method\_name = ""

        self.exact\_ys = exact\_ys

    @staticmethod

    def compare\_errors(approx\_ys, exact\_ys):

        return [abs(y2 - y1) for y1, y2 in zip(approx\_ys[1:], exact\_ys[1:])]

    def \_get\_next\_y(self, current\_x, current\_y):

        return self.function(current\_x, current\_y)

    def \_get\_exact\_y\_and\_error(self, current\_x, current\_y, i):

        exact\_y = self.function(current\_x, current\_y)

        error = abs(current\_y - exact\_y)

        return exact\_y, error

    def \_table\_line(self, current\_x, current\_y, i):

        if self.exact\_ys is None:

            print(f'{"%0.8f" % current\_x:<16} | {"%0.8f" % current\_y:<16}')

        else:

            error = abs(current\_y - self.exact\_ys[i])

            print(f'{"%0.8f" % current\_x:<16} | {"%0.8f" % current\_y:<16} | {"%0.8f" % self.exact\_ys[i]:<16} | '

                  f'{"%0.8f" % error:<16}')

    def get\_xy(self, initial\_x, initial\_y, last\_x):

        xs = np.arange(initial\_x, last\_x + self.step\_h, self.step\_h)

        ys = []

        current\_y = initial\_y

        self.\_description\_line(initial\_x, initial\_y, last\_x)

        self.\_table\_header()

        for i, current\_x in enumerate(xs):

            self.\_table\_line(current\_x, current\_y, i)

            ys.append(current\_y)

            current\_y = self.\_get\_next\_y(current\_x, current\_y)

        return xs, ys

class RungeKutta(DifferentialEquationsSolver):

    def \_\_init\_\_(self, function, step\_h, exact\_ys=None):

        super().\_\_init\_\_(function, step\_h, exact\_ys)

        self.method\_name = "Runge Kutta"

    def \_get\_next\_y(self, current\_x, current\_y):

        adjusted\_step\_h = self.step\_h / 2

        f1\_left\_point = self.function(

            current\_x, current\_y)

        f2\_central\_point\_euler = self.function(

            current\_x + adjusted\_step\_h, current\_y + adjusted\_step\_h \* f1\_left\_point)

        f3\_central\_point\_adjusted = self.function(

            current\_x + adjusted\_step\_h, current\_y + adjusted\_step\_h \* f2\_central\_point\_euler)

        f4\_right\_point = self.function(

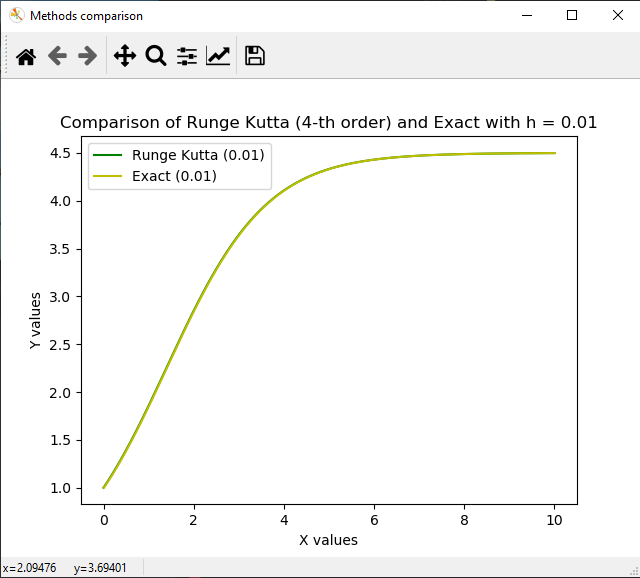
            current\_x + self.step\_h, current\_y + self.step\_h \* f3\_central\_point\_adjusted)

        return current\_y + (self.step\_h / 6) \* (

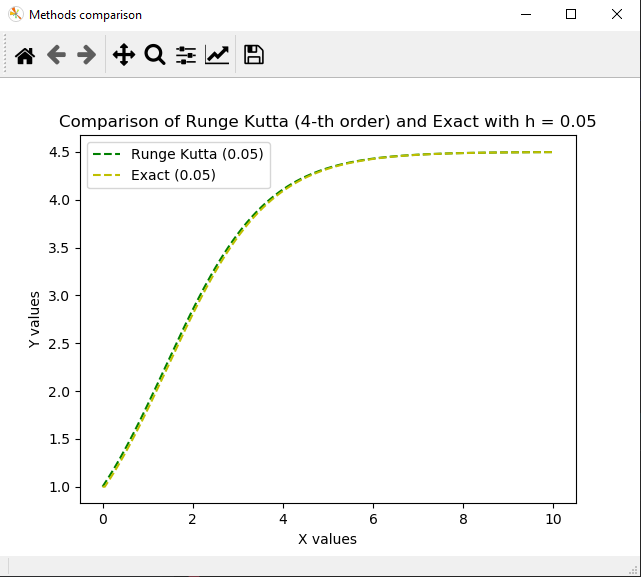
            f1\_left\_point + 2 \* f2\_central\_point\_euler + 2 \* f3\_central\_point\_adjusted + f4\_right\_point

        )

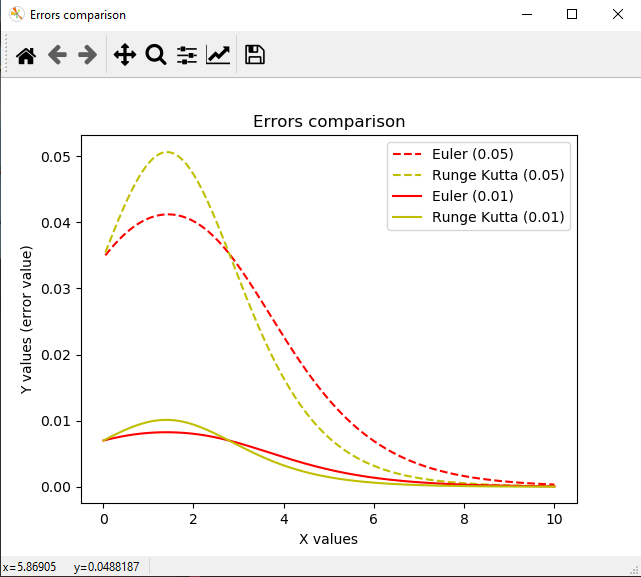
**График 1:**



**График 2:**



**4) Вывод**



На данном графике видно, что метод Рунге-Кутта 4-го порядка быстрее сходиться в отличии от метода Эйлера. В начале метод Эйлера кажется чуть точнее, но быстро уступает. С увеличением точности, графики стали более приземистые, но особо не поменялись, они аналогичны, как и при точности 0.05. Изменение точности внешне на графике не отобразилось, и, если сравнивать разницу в точке х=10, разница в значениях минимальна. Изменение точности слабо повлияло на скорость сходимости.