# Практическое занятие №2. Численные методы решения ОДУ. Вариант 9

## Формулировка задания

Реализовать на любом языке программирования программу для численного интегрирования дифференциальных уравнений в соответствие с вариантом следующими методами

* Метод Эйлера -
* y' = x + cos(y/sqrt(2))
* Усовершенствованный метод Эйлера
* Метод Рунге-Кутта - y' = 1 - sin(x+y)+0,5y/(x+2)

Начальные условия задаются с клавиатуры. Выходными данными является набор пар значений X и Y для функции в заданном отрезке.

## Ход решения

Для каждого метода была разработанна отдельная программа.

Ее словесный алгоритм приведен ниже:

1. Запросить начало отрезка
2. Запросить конец отрезка
3. Запросить шаг отрезка
4. Запросить начальное значение Y0

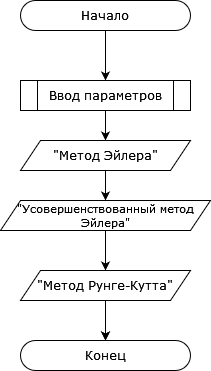
Для печати результатов использовалась функция со следующим алгоритмом:

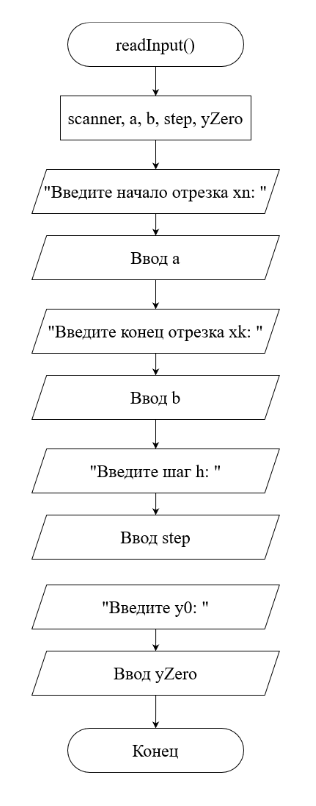
1. Открыть цикл по всем элементам списка значений
2. Напечатать пару значений X и Y, отформатированную до 4 знака после запятой
3. Закрыть цикл

Алгоритмы методов решения не был принципиально изменен, за исключением того, что начальные значения передаются в функции напрямую, а не запрашиваются у пользователя индивидуально для каждого прохода. Также результаты работы выводятся после окончания расчетов. В связи с этим словесное описание опущено, блок схемы приведены ниже.

## Блок-схемы функций программы

Основная функция.





## Программа

import math as mth

class Solver:

    def \_\_init\_\_(self):

        self.a:float = float(input("Введите начало отрезка xn: "))

        self.b:float = float(input("Введите конец отрезка xk: "))

        self.step:float = float(input("Введите шаг h: "))

        self.yZero:float = float(input("Введите yо: "))

    def execute(self):

        print("Метод Эйлера")

        self.solveEulerBasic(self.a, self.b,self.step,self.yZero)

        print("Усовершенствованный метод Эйлера")

        self.solveEulerAdvance(self.a, self.b,self.step,self.yZero)

        print("Метод Рунге-Кутта")

        self.solveRungeKutt(self.a, self.b,self.step,self.yZero)

    def printResults(self, xValues, yValues):

        for i in range(len(xValues)):

            print("X[" + str(i) + "]=" + str(xValues[i]) + " Y[" + str(i) + "]=" + str(yValues[i]))

        print("\n")

    def solveEulerBasic(self, a:float,b:float,step:float,yZero:float):

        iterationCount:int = int(mth.ceil(b-a)/step)

        xValues = []

        yValues = []

        y:float = yZero

        x:float = a

        xValues.append(x)

        yValues.append(y)

        yDer:float

        yDelta:float

        for i in range(iterationCount):

            yDer = x + mth.cos(y/mth.sqrt(2))

            yDelta = step\*yDer

            y += yDelta

            x += step

            xValues.append(x)

            yValues.append(y)

        print("Результаты решения ОДУ методом Эйлера\n")

        self.printResults(xValues, yValues)

    def solveEulerAdvance(self, a:float, b:float, step:float, yZero:float):

        iterationCount:int = (int)(mth.ceil(b-a)/step)

        xValues = []

        yValues = []

        y:float = yZero

        x:float = a

        f:float

        hy:float

        x2:float

        y2:float

        f2:float

        hy2:float

        xValues.append(x)

        yValues.append(y)

        for i in range(iterationCount):

            f = self.euAdvFunc(x, y)

            hy = f\*step/2

            x2=x+step/2

            y2=y+hy

            f2 = self.euAdvFunc(x2, y2)

            hy2 = step\*f2

            y += hy2

            x += step

            xValues.append(x)

            yValues.append(y)

        self.printResults(xValues, yValues)

    # 368.0))x8.1sin(x(213.

    def euAdvFunc(self, x:float, y:float):

        return 0.213\*(mth.pow(x, 2) + mth.sin(1.8\*x))+0.\*y

    def solveRungeKutt(self, a:float,b:float,step:float,yZero:float):

        iterationCount:int = int((mth.ceil(b-a)/step))

        xValues = []

        yValues = []

        y:float = yZero

        x:float = a

        f1:float

        k1:float

        f2:float

        k2:float

        f3:float

        k3:float

        f4:float

        k4:float

        dy :float

        xValues.append(x)

        yValues.append(y)

        for i in range(iterationCount):

            f1 = self.rungeKuttFunc(x, y)

            k1 = step \* f1

            f2 = self.rungeKuttFunc(x+step/2, y+k1/2)

            k2 = step \* f2

            f3 = self.rungeKuttFunc(x+step/2, y+k2/2)

            k3 = step \* f3

            f4 = self.rungeKuttFunc(x+step, y+k3)

            k4 = step \* f4

            dy = (k1+k2\*2+k3\*2+k4)/6

            y += dy

            x += step

            xValues.append(x)

            yValues.append(y)

        self.printResults(xValues, yValues)

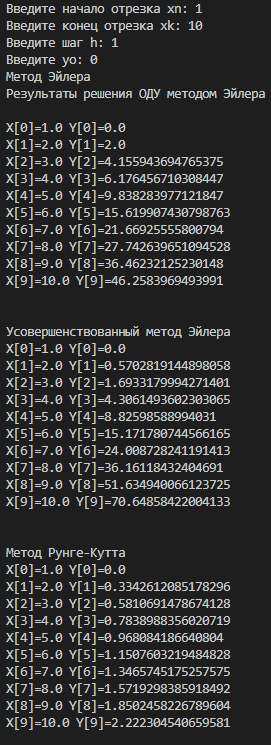
    def rungeKuttFunc(self, x, y):

        return (mth.cos(y)/(1.5+x))+0.1\*y\*y

s = Solver()

s.execute()

## Скриншоты



## Реализация на Excel

## 

## Вывод

В процессе работы были исследованы три численных подхода для решения дифференциальных уравнений первого порядка, которые были реализованы с использованием языка программирования Java и Excel.