Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

**­­­­­Лабораторная работа №6**

**по дисциплине «Вычислительная математика»**

**«Численное решение обыкновенных дифференциальных уравнений»**

Вариант №3

Группа: P3212

Выполнил: Балин А. А.

Проверила: Наумова Н. А.

# Цель работы

Решить задачу Коши для обыкновенных дифференциальных уравнений численными методами.

# Программная часть

## Код

from numpy import exp, inf, divide

class Odu:

    @staticmethod

    def get\_constant\_1(x0,y0):

        return -exp(x0)\*x0-divide(exp(x0),y0)

    @staticmethod

    def y1(x,y,c):

        return divide(-exp(x),c+exp(x)\*x)

    @staticmethod

    def get\_constant\_2(x0,y0):

        return divide(y0-x0-1,exp(x0))

    @staticmethod

    def y2(x,y,c):

        return c\*exp(x)+x+1

    @staticmethod

    def get\_constant\_3(x0,y0):

        return divide(y0-divide(exp(x0),2),exp(-x0))

    @staticmethod

    def y3(x,y,c):

        return c\*exp(-x)+divide(exp(x),2)

    @staticmethod

    def f1(x,y):

        return y+(1+x)\*y\*\*2

    @staticmethod

    def f2(x,y):

        return y-x

    @staticmethod

    def f3(x,y):

        return -y+exp(x)

    def \_\_init\_\_(self,x,y,b,f,e):

        assert x!=b, "The interval is empty"

        if(f==1):

            self.f = self.f1

            self.constant = self.get\_constant\_1(x,y)

            self.y = self.y1

        elif(f==2):

            self.f = self.f2

            self.constant = self.get\_constant\_2(x,y)

            self.y = self.y2

        elif(f==3):

            self.f = self.f3

            self.constant = self.get\_constant\_3(x,y)

            self.y = self.y3

        assert self.constant is not inf, "The constant is infinite"

        self.y0 = y

        self.a = x

        self.b = b

        self.e = e

    def Eiler(self):

        h = self.b-self.a

        res = self.Eiler\_mod(h)

        while not res[0]:

            h = h/2

            x = res[1].copy()

            check = res[2].copy()

            res = self.Eiler\_mod(h,check[-1])

        return [x,check]

    #Контролируем точность конечного интервала

    def Eiler\_mod(self,h,check=None):

        x = [self.a]

        while x[-1]<self.b:

            x.append(x[-1]+h)

        y = [self.y0]

        for x\_ in x[:len(x)-1]:

            y.append(y[-1]+h\*self.f(x\_,y[-1]))

        if check!=None:

            if abs(y[-1]-check)<self.e:ы

                return [True,[],[]]

            else:

                return [False,x,y]

        else:

            return [False,x,y]

    def Runge\_Kutta(self):

        h = self.b-self.a

        res = self.Runge\_Kutta\_mod(h)

        while not res[0]:

            h = h/2

            x = res[1].copy()

            check = res[2].copy()

            res = self.Runge\_Kutta\_mod(h,check[-1])

        return [x,check]

    def Runge\_Kutta\_mod(self,h,check=None):

        x = [self.a]

        while x[-1]<self.b:

            x.append(x[-1]+h)

        y = [self.y0]

        for x\_ in x[:len(x)-1]:

            k1 = h\*self.f(x\_,y[-1])

            k2 = h\*self.f(x\_+h/2,y[-1]+k1/2)

            k3 = h\*self.f(x\_+h/2,y[-1]+k2/2)

            k4 = h\*self.f(x\_+h,y[-1]+k3)

            y.append(y[-1]+(k1+2\*k2+2\*k3+k4)/6)

        if check!=None:

            if abs(y[-1]-check)<self.e:

                return [True,[],[]]

            else:

                return [False,x,y]

        else:

            return [False,x,y]

    def Milne(self,h=None):

        if h is None:

            h = (self.b-self.a)/4

        x = [self.a]

        while x[-1]<self.b:

            x.append(x[-1]+h)

        y = [self.y0]

        for x\_ in x[:3]:

            y.append(y[-1]+h\*self.f(x\_,y[-1]))

        f = [self.f(x[i],y[i]) for i in range(3)]

        y\_progn = y[-4]+4\*h/3\*(2\*f[-1]-f[-2]+2\*f[-3])

        f\_prog = 0

        y\_corr = 0

        i = 4

        while len(y)!=len(x):

            f\_prog = self.f(x[i],y\_progn)

            y\_corr = y[-2]+h/3\*(f[-2]+4\*f[-1]+f\_prog)

            if(abs(y\_corr-y\_progn)<self.e):

                y.append(y\_corr)

                f.append(f\_prog)

                y\_progn = y[-4]+4\*h/3\*(2\*f[-1]-f[-2]+2\*f[-3])

            else:

                y\_progn = y\_corr

                i-=1

            i+=1

        T = True

        for i in range(len(y)):

            if abs(y[i]-self.y(x[i],y[i],self.constant))>self.e:

                T = False

                break

        if T:

            return [x,y]

        else:

            return self.Milne(h/2)

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 1. Пример выполнения программы.

Изображение выглядит как снимок экрана, текст

Автоматически созданное описание

Рисунок 2. Пример выполнения программы.

Изображение выглядит как снимок экрана, текст

Автоматически созданное описание

Рисунок 3. Пример выполнения программы.

## Рабочие формулы

Для метода Эйлера:

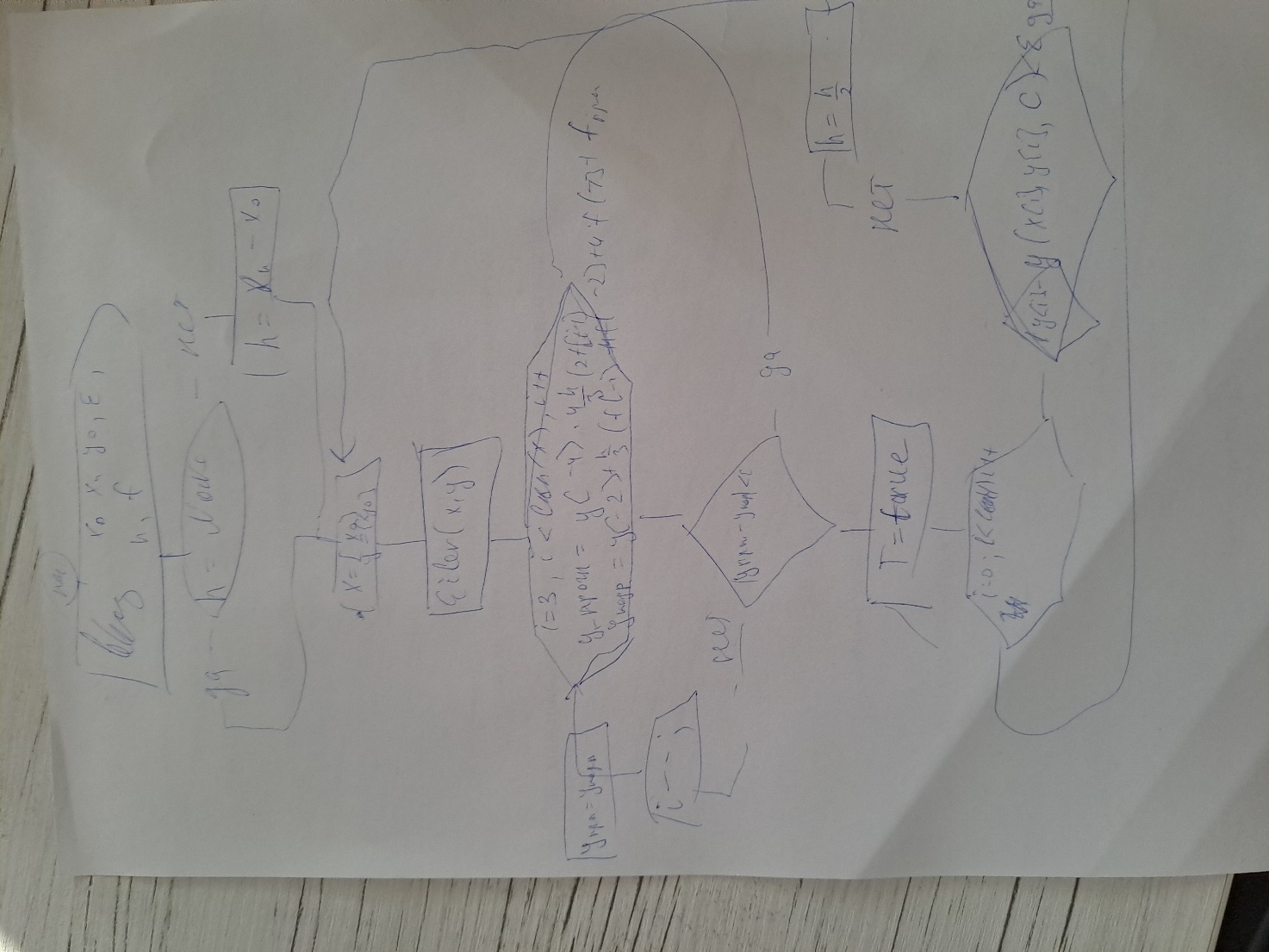
Для метода Рунге-Кутта:

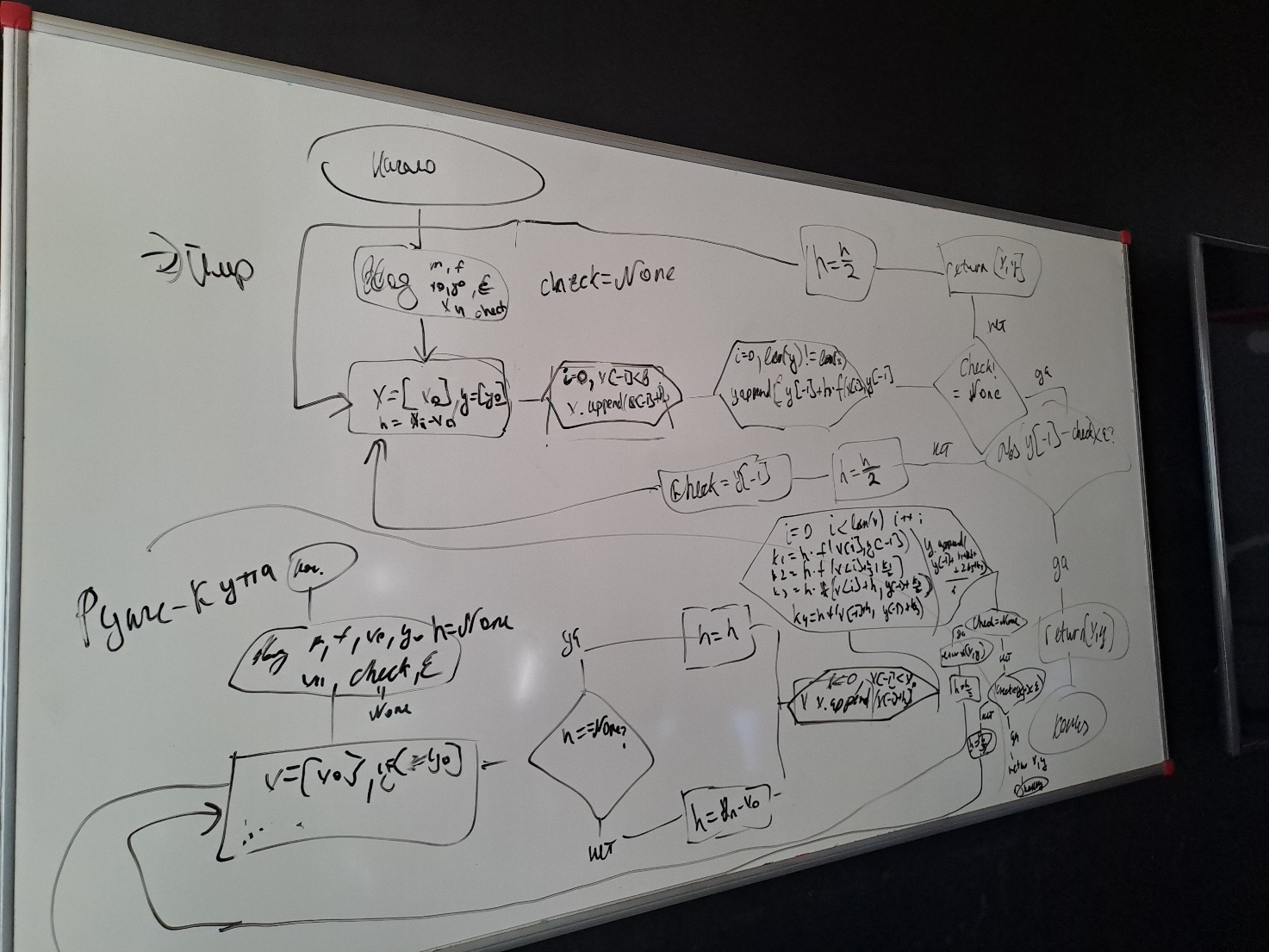
Для метода Милна:

Правило Рунге для одношаговых методов:

Для многошагового метода используем сравнение по модулю с точным значением функции:

## Блок схемы





## Репозиторий с исходниками

https://github.com/ta4ilka69/docs\_for\_labs/tree/main/Вычмат

# Вывод

В ходе реализации данной лабораторной работы я ознакомился с различными методами решения ОДУ.