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.

г. Санкт-Петербург, 2024

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)
```

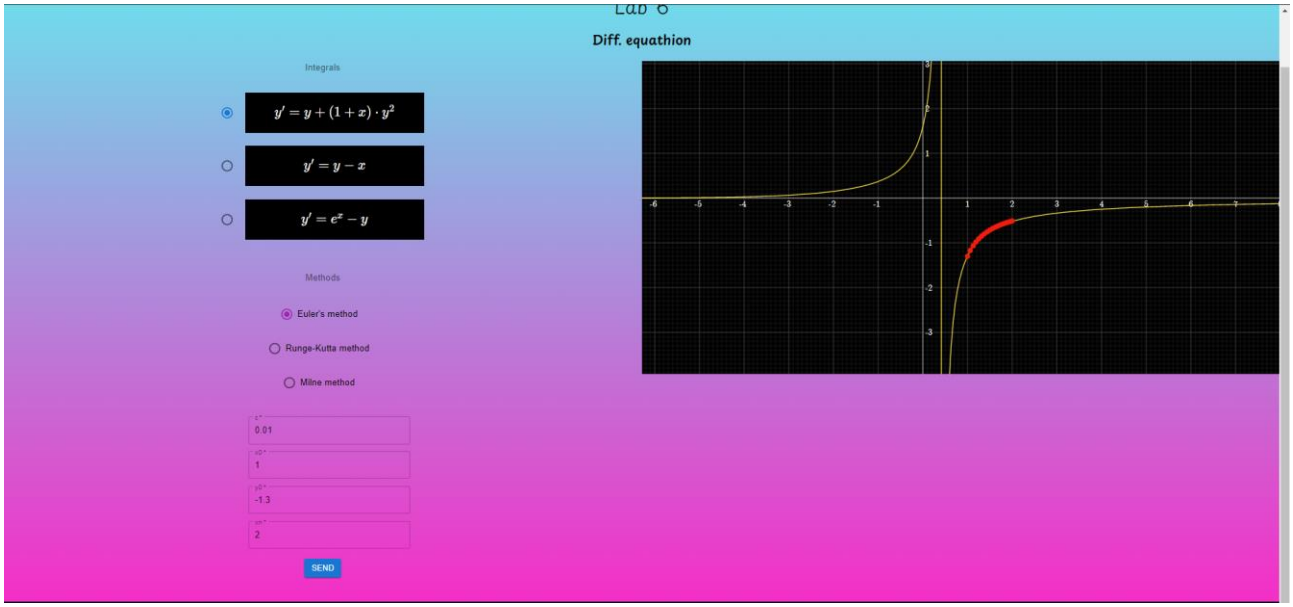


Рисунок 1. Пример выполнения программы.

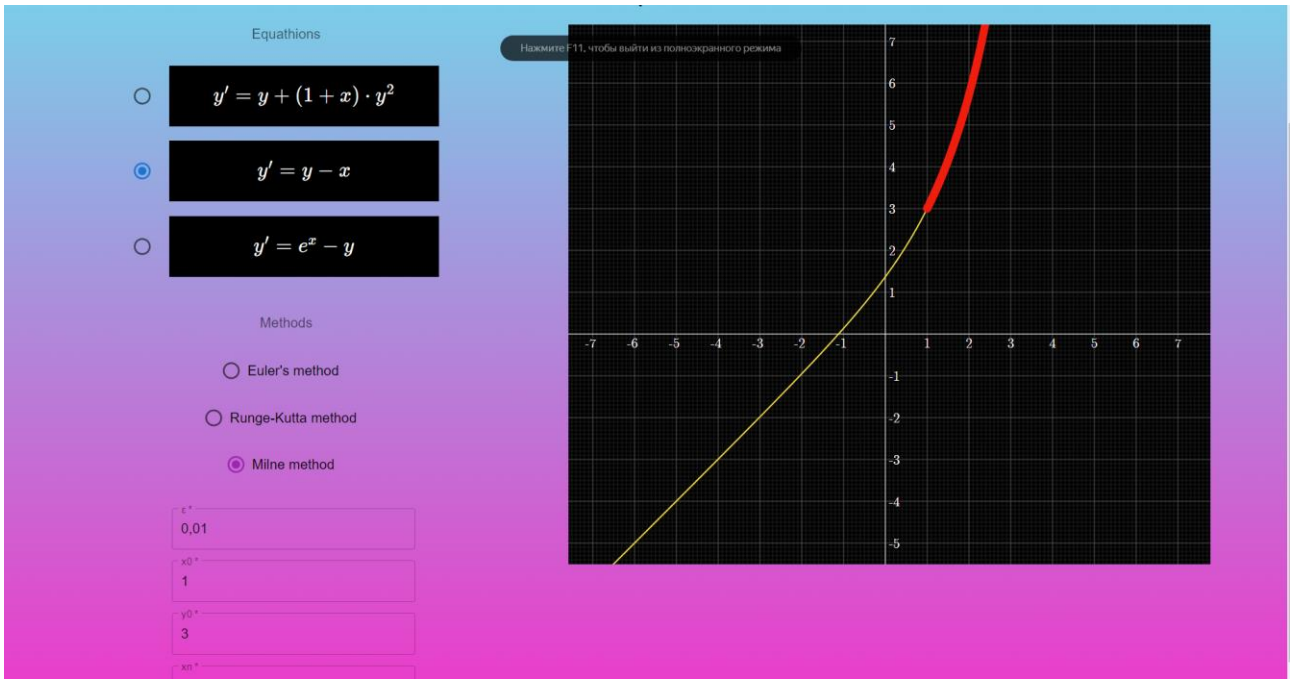


Рисунок 2. Пример выполнения программы.

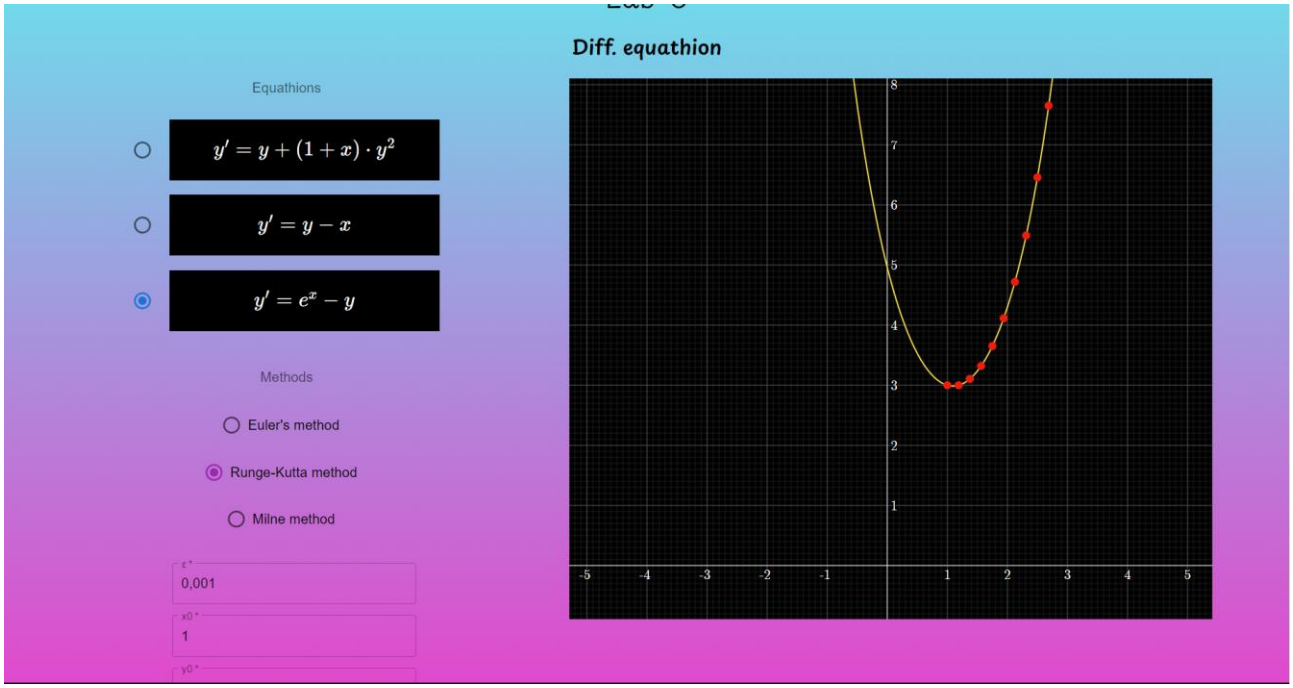


Рисунок 3. Пример выполнения программы.

Рабочие формулы

Для метода Эйлера:

$$y_{i+1} = y_i + hf(x_i, y_i)$$

Для метода Рунге-Кутта:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = h \cdot f(x_i, y_i)$$

$$k_2 = h \cdot f\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right)$$

$$k_3 = h \cdot f\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right)$$

$$k_4 = g \cdot f(x_i + h, y_i + k_3)$$

Для метода Милна:

$$y_i^{\text{прогн}} = y_{i-4} + \frac{4h}{3}(2f_{i-3} - f_{i-2} + 2f_{i-1})$$

$$y_i^{\text{корр}} = y_{i-2} + \frac{h}{3}\left(f_{i-2} + 4f_{i-1} + f_i^{\text{прогн}}\right)$$

$$f_i^{\text{прогн}} = f\left(x_i, y_i^{\text{прогн}}\right)$$

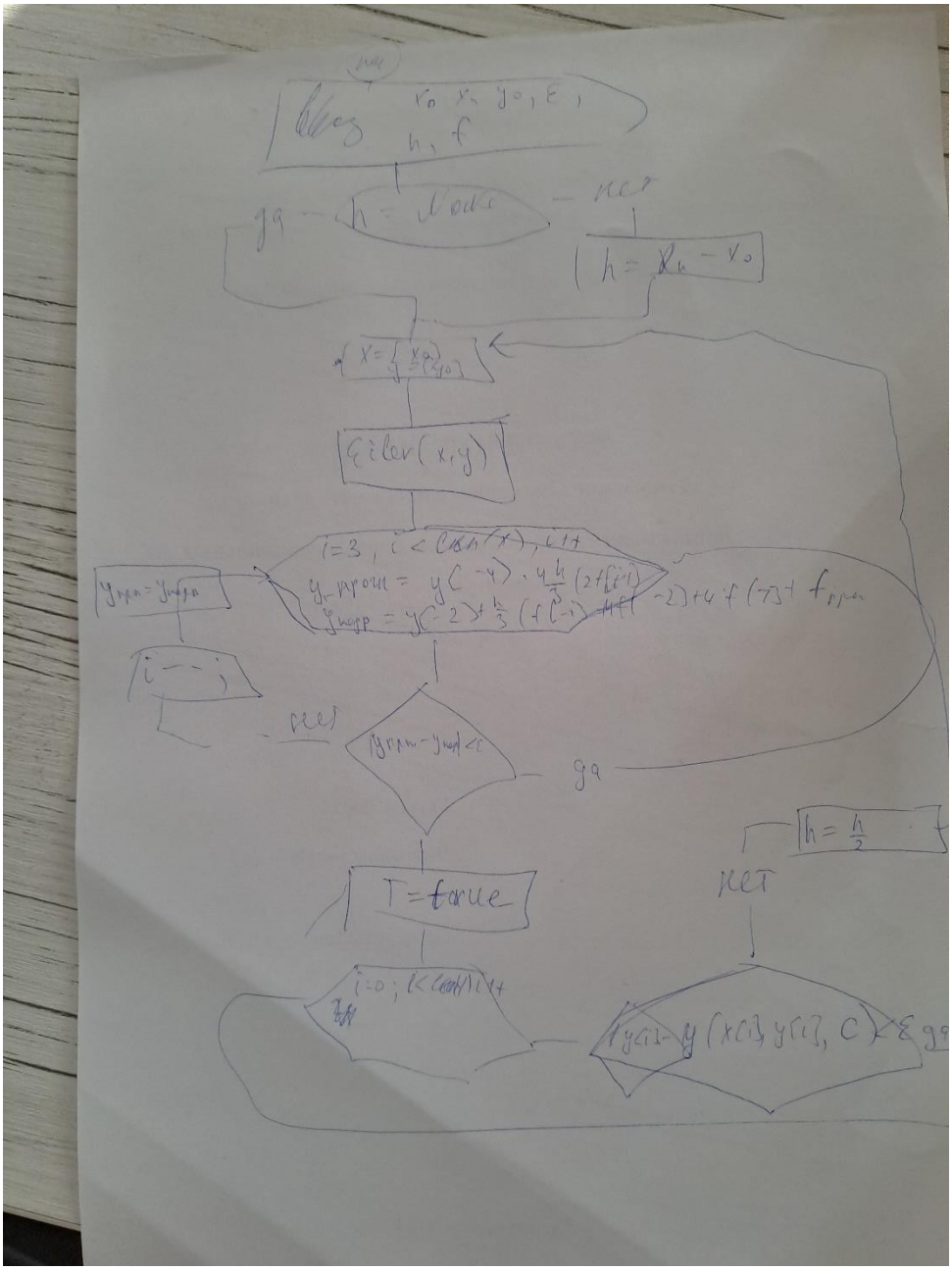
Правило Рунге для одношаговых методов:

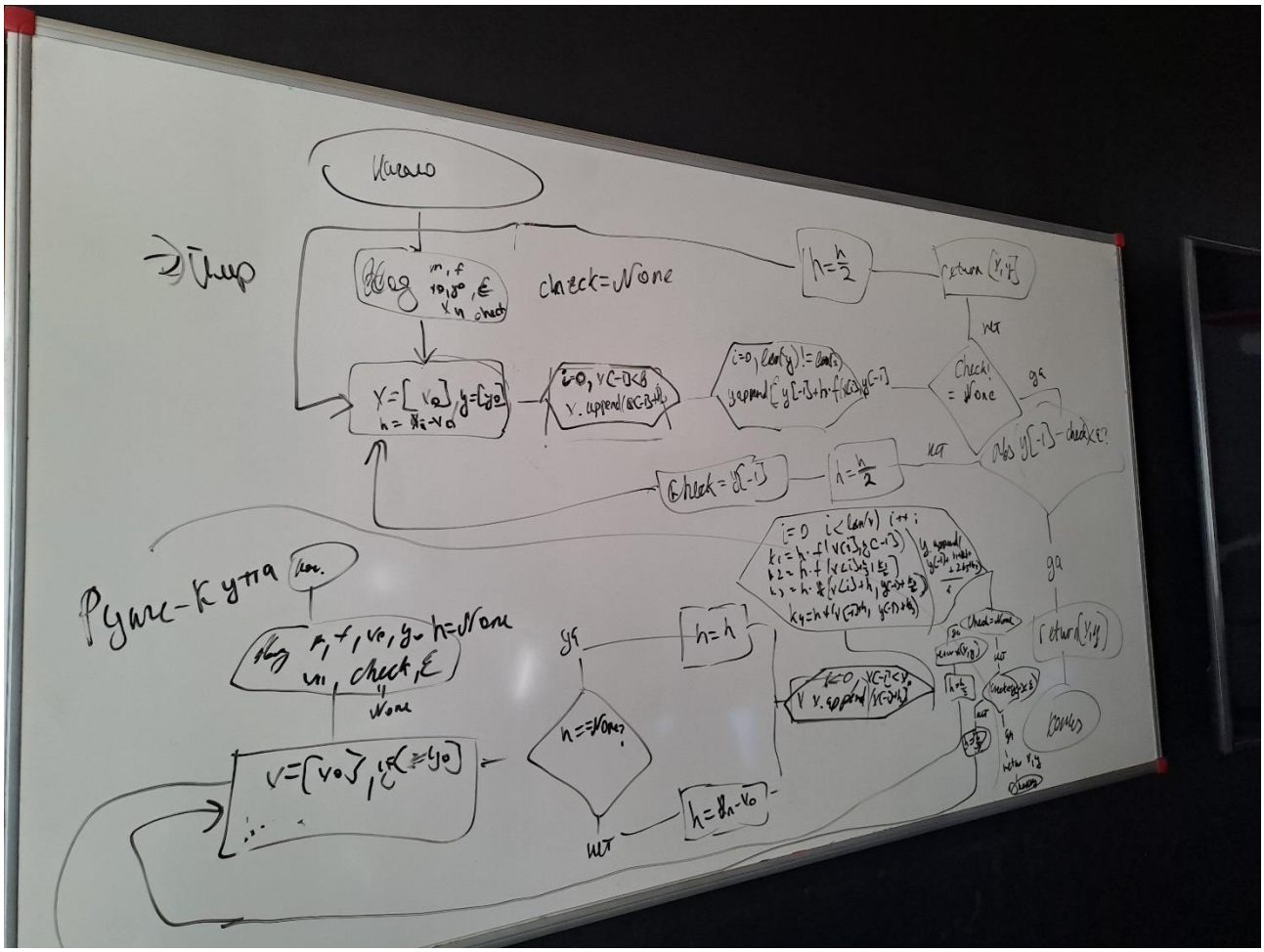
$$\frac{\left|y_i^h - y_i^{h/2}\right|}{2^p - 1} \leq \varepsilon$$

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

$$\varepsilon = max_{0 \leq i \leq n} |y_{i\text{точн}} - y_i|$$

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.