**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

по дисциплине «Методы численного анализа»

на тему: «Численные методы решения ОДУ»

Выполнил: студент гр. ИП-22

Коваленко А.И.

Принял: преподаватель

Процкая М.А.

Гомель 2022

**Цель работы**: получить навыки применения таких методов решения ОДУ и систем ОДУ, как метод Эйлера, метод Рунге-Кутта, научиться выполнять графическую интерпретацию полученных результатов

**Задание 1**

1. Получит решение ОДУ первого порядка следующими методами:

- метод Эйлера (h=0.2);

- метод Эйлера (h=0.1);

- метод Рунге-Кутта (h=0.2).

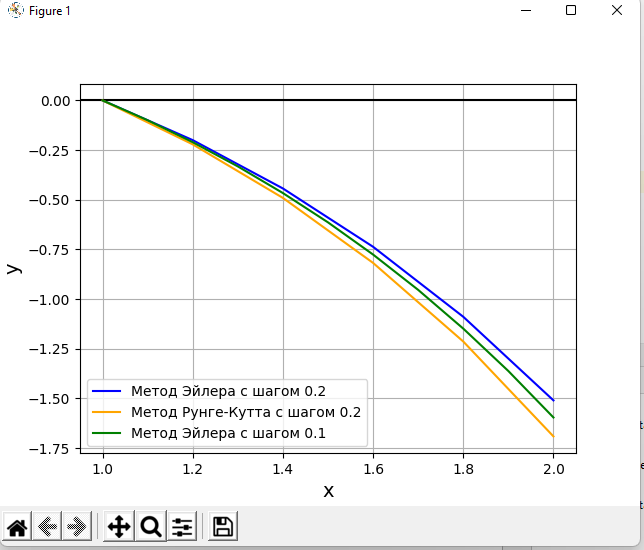
1. Сравнить полученные результаты с точным значением функции, сделать вывод о погрешностях методов.
2. Сделать графическую интерпретацию результатов для всех методов.



**Листинг программы:**

**import** numpy **as** np  
**from** math **import**\*  
**import** matplotlib.pyplot **as** plt  
  
  
**def** main():  
 task1()  
  
  
**def** f1(x, y):  
 **return** -np.sqrt(y\*\*2+x\*\*2)  
  
  
**def** euler\_method(f, xn, xk, yn, h):  
 steps = int((xk - xn) / h) + 1  
 x = np.linspace(xn, xk, steps)  
 y = np.zeros(steps)  
 y[1] = yn  
 **for** i **in** range(1, steps):  
 y[i] = y[i - 1] + h \* f(x[i - 1], y[i - 1])  
 **return** (x, y)  
  
  
**def** runge\_kutta\_method(f, xn, xk, yn, h):  
 steps = int((xk - xn) / h) + 1  
 x = np.linspace(xn, xk, steps)  
 y = np.zeros(steps)  
 y[1] = yn  
 **for** i **in** range(1, steps):  
 k1 = f(x[i - 1], y[i - 1])  
 k2 = f(x[i - 1] + h / 2, y[i - 1] + (h \* k1) / 2)  
 k3 = f(x[i - 1] + h / 2, y[i - 1] + (h \* k2) / 2)  
 k4 = f(x[i - 1] + h, y[i - 1] + h \* k3)  
 y[i] = y[i - 1] + (h \* (k1 + 2 \* k2 + 2 \* k3 + k4)) / 6  
 **return** (x, y)  
  
  
**def** task1():  
 xn = 1  
 xk = 2  
 yn = -0.75  
 h = 0.2  
 (x, y) = euler\_method(f1, xn, xk, yn, h)  
 print(**"Метод Эйлера (0.2): "**, euler\_method(f1, xn, xk, yn, h))  
 (fig, ax) = plt.subplots()  
 plt.xlabel(**'x'**, fontsize=14)  
 plt.ylabel(**'y'**, fontsize=14)  
 plt.grid()  
 ax.plot(x, y, color=**"blue"**, label=**"Метод Эйлера с шагом 0.2"**)  
 plt.axhline(0, color=**"black"**, linestyle=**'-'**)  
 h = 0.2  
 (x, y) = runge\_kutta\_method(f1, xn, xk, yn, h)  
 ax.plot(x, y, color=**"orange"**, label=**"Метод Рунге-Кутта с шагом 0.2"**)  
 h = 0.1  
 (x, y) = euler\_method(f1, xn, xk, yn, h)  
 ax.plot(x, y, color=**"green"**, label=**"Метод Эйлера с шагом 0.1"**)  
 ax.legend()  
 plt.show()  
  
  
main()

**Результат выполнения**



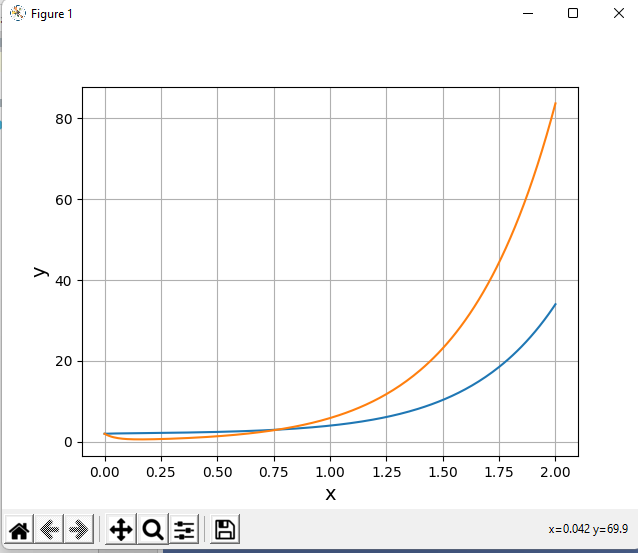
**Задание 2**

1. Получит решение ОДУ второго порядка методом Рунге-Кутта.
2. Сделать графическую интерпретацию результатов расчетов.

**Листинг программы:**

**import** math  
  
**import** numpy **as** np  
**from** math **import**\*  
**import** matplotlib.pyplot **as** plt  
  
**def** f(x, y):  
 f = np.zeros((2), **'float'**)  
 f[0] = y[1]  
 f[1] = 16\*x\*math.exp(2\*x) + 4\*y[0] - 20\*y[1]  
 **return** f  
  
  
**def** rk(f, t0, y0, h):  
 **def** danrk(f, t,y, h):  
 k0 = h \* f(t, y)  
 k1 = h \* f(t + h/2., y + k0/2.)  
 k2 = h \* f(t + h/2., y + k1 / 2.)  
 k3 = h \* f(t + h, y + k2)  
 **return** (k0 + 2. \* k1 + 2. \* k2 + k3)/6  
 t = []  
 y = []  
 t.append(t0)  
 y.append(y0)  
 **while** t0 < tn:  
 y0 = y0 + danrk(f, t0, y0, h)  
 t0 = t0 + h  
 t.append(t0)  
 y.append(y0)  
 **return** np.array(t), np.array(y)  
  
t0 = 0.  
tn = 2  
y0 = np.array([2, 2])  
h = 0.01  
t1, y1 = rk(f, t0, y0, h)  
r = y1[:, 0]  
k = y1[:, 1]  
plt.plot(t1, r)  
plt.plot(t1, k)  
plt.xlabel(**'x'**, fontsize=14)  
plt.ylabel(**'y'**, fontsize=14)  
plt.grid()  
plt.show()

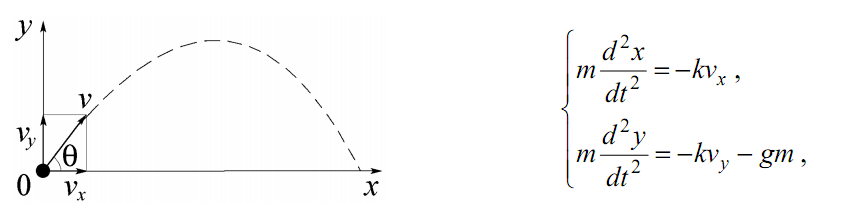
**Результат выполнения**



**Задача 3 Модель «Движение снаряда»**

**Вариант 19-2**

Решить задачу Коши, описывающую движение тела, брошенного с начальной скоростью v0 под углом α к горизонту в предположении, что сопротивление воздуха пропорционально квадрату скорости.



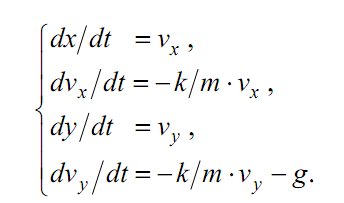
где vx = v cosθ и vy = v cosθ – горизонтальная и вертикальная составляющая

скорости снаряда, t – время, k – коэффициент трения, учитывающий сопротивление воздуха, m – масса снаряда, g ≈ 9,8 м/c2 – ускорение

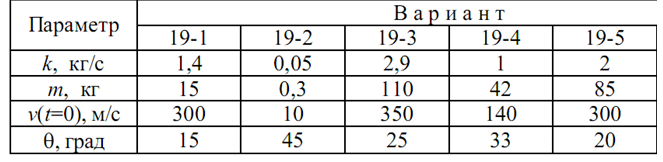
свободного падения.



С учетом vx = dx/dt и vy = dy/dt исходная система уравнений приводится к следующему виду:



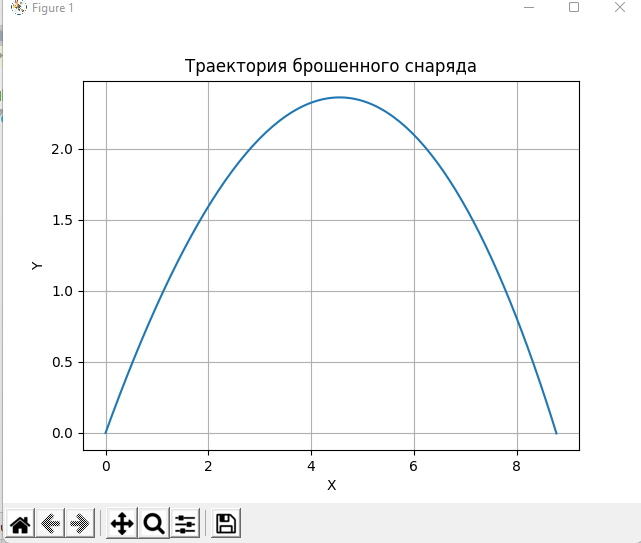
Рассчитайте зависимости x(t) и y(t). Постройте траекторию снаряда y(x) из точки x(t=0) = 0 и y(t=0) = 0 при заданных k, m, θ и начальной скорости v(t=0).



**Листинг программы:**

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
  
**def** f(t, y):  
 f = np.zeros(4)  
 f[0] = y[1]  
 f[1] = -k / m \* y[1]  
 f[2] = y[3]  
 f[3] = -k / m \* y[3] - g  
  
 **return** f  
  
  
**def** runge\_kutta\_method(t0, y0, h):  
 ti = t0  
 yi = y0  
  
 result\_t = np.array(ti)  
 result\_y = np.array(yi)  
  
 **while** yi[2] >= 0:  
 k1 = h \* f(ti, yi)  
 k2 = h \* f(ti + h / 2, yi + k1 / 2)  
 k3 = h \* f(ti + h / 2, yi + k2 / 2)  
 k4 = h \* f(ti + h, yi + k3)  
  
 yi += 1 / 6 \* (k1 + 2 \* k2 + 2 \* k3 + k4)  
 ti += h  
  
 result\_t = np.vstack((result\_t, ti))  
 result\_y = np.vstack((result\_y, yi))  
  
 **return** result\_t, result\_y  
  
  
t0 = 0  
theta = np.pi / 4  
v0 = 10  
k = 0.05  
g = 9.8  
m = 0.3  
y0 = np.array([0, v0 \* np.cos(theta), 0, v0 \* np.sin(theta)])  
h = 0.01  
  
result\_t, result\_y = runge\_kutta\_method(t0, y0, h)  
  
plt.plot(result\_y[:, 0], result\_y[:, 2])  
plt.title(**'Траектория брошенного снаряда'**)  
plt.xlabel(**'X'**)  
plt.ylabel(**'Y'**)  
plt.grid()  
plt.show()

**Результат выполнения**



**Вывод:** получили навыки применения таких методов решения ОДУ и систем ОДУ, как метод Эйлера, метод Рунге-Кутта, научились выполнять графическую интерпретацию полученных результатов.