```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, pi, sqrt, linalg
from matplotlib.pyplot import style, figure, axes
from celluloid import Camera

# Набор команд, за счёт которых анимация строится в отдельном окне
from IPython import get_ipython
get_ipython().run_line_magic('matplotlib', 'qt')

# Функция f подготавливает массив, содержащий элементы вектор-функции,
# определяющей правую часть решаемой системы ОДУ
def f(u,g,mass,l) :
    f = zeros(5)
    f[0] = u[2]
    f[1] = u[3]
    f[2] = 2*u[4]*u[0]/mass
    f[3] = -g + 2*u[4]*u[1]/mass
    f[4] = u[0]**2 + u[1]**2 - l**2
    return f

# Функция подготавливает массив, содержащий элементы матрицы D
def D() :
    D = zeros((5,5))
    # Задаются ненулевые диагональные элементы матрицы D
    for i in range(4) :
        D[i,i] = 1.
    return D

# Функция подготавливает массив, содержащий элементы матрицы Якоби f_u
def f_u(u,mass) :
    f_u = zeros((5,5))
    # Задаются ненулевые компоненты матрицы Якоби
    f_u[0,2] = 1.
    f_u[1,3] = 1.
    f_u[2,0] = 2*u[4]/mass
    f_u[2,4] = 2*u[0]/mass
    f_u[3,1] = 2*u[4]/mass
    f_u[3,4] = 2*u[1]/mass
    f_u[4,0] = 2*u[0]
    f_u[4,1] = 2*u[1]
    return f_u

# Определение входных данных задачи
t_0 = 0.
x_0 = 3.; y_0 = -4.
v_x_0 = 0.; v_y_0 = 0.
g = 9.81; l = 5.0; mass = 1.0
T = 2*(2*pi*sqrt(l/g)) # T равно двум периодам колебаний маятника
# Определение множителя Лагранжа
lambda_0 = (y_0*g - v_x_0**2 - v_y_0**2)*mass/(2*l**2)

# Определение параметра схемы (нужный раскомментировать)
alpha = (1 + 1j)/2  # CROS1 (схема Розенброка с комплексным коэффициентом)
# alpha = 1.          # DIRK1 (обратная схема Эйлера)

# Определение числа интервалов сетки,
# на которой будет искаться приближённое решение
M = 500
# Определение сетки
tau = (T - t_0)/M
t = linspace(t_0,T,M + 1)

# Выделение памяти под массив сеточных значений решения системы ОДУ
# В строке с номером m этого массива хранятся сеточные значения решения,
# соответствующие моменту времени t_m
u = zeros((M + 1,5))

# Задание начальных условий
```

```
# (записываются в строку с индексом 0 массива u)
u[0,:] = [x_0, y_0, v_x_0, v_y_0, lambda_0]

# Реализация схемы из семейства ROS1
# конкретная схема определяется коэффициентом alpha
for m in range(M) :
    w_1 = linalg.solve(D() - alpha*tau*f_u(u[m],mass),f(u[m],g,mass,l))
    u[m + 1] = u[m] + tau*w_1.real

# Анимация отрисовки решения
style.use('dark_background')
fig = figure()
camera = Camera(fig)
ax = axes(xlim=(-5.5,5.5), ylim=(-8,1))
ax.set_aspect('equal'); ax.set_xlabel('x'); ax.set_ylabel('y')
for m in range(M + 1) :
    ax.plot(0,0, color="yellow", marker='o', markersize=5)
    ax.plot((-2,2),(0,0),'-', color="white")
    # Отрисовка подвеса
    ax.plot((0,u[m,0]),(0,u[m,1]), color="white")
    # Отрисовка груза
    ax.plot(u[m,0],u[m,1], color="white", marker='o', markersize=10)
    camera.snap()
animation = camera.animate(interval=15, repeat=False, blit=True)

# Листинг программы, реализущей решение
# системы дифференциально-алгебраических уравнений
# с помощью схемы CROS1 или DIRK1
# (на примереме моделирования движениея маятника)
```