```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_1,mass_2,l_1,l_2) :
    f = zeros(10)
    f[0] = u[4]
    f[1] = u[5]
    f[2] = u[6]
    f[3] = u[7]
    f[4] = u[8]/mass_1*2*u[0] - u[9]/mass_1*2*(u[2] - u[0])
    f[5] = -g + u[8]/mass_1*2*u[1] - u[9]/mass_1*2*(u[3] - u[1])
    f[6] = u[9]/mass_2*2*(u[2] - u[0])
    f[7] = -g + u[9]/mass_2*2*(u[3] - u[1])
    f[8] = u[0]**2 + u[1]**2 - l_1**2
    f[9] = (u[2] - u[0])**2 + (u[3] - u[1])**2 - l_2**2
    return f

# Функция подготавливает массив, содержащий элементы матрицы D
def D() :
    D = zeros((10,10))
    # Задаются ненулевые диагональные элементы матрицы D
    for i in range(8) :
        D[i,i] = 1.
    return D

# Функция подготавливает массив, содержащий элементы матрицы Якоби f_u
def f_u(u,mass_1,mass_2,) :
    f_u = zeros((10,10))
    # Задаются ненулевые компоненты матрицы Якоби
    f_u[0,4] = 1.
    f_u[1,5] = 1.
    f_u[2,6] = 1.
    f_u[3,7] = 1.
    f_u[4,0] = u[8]/mass_1*2 + u[9]/mass_1*2
    f_u[4,2] = -u[9]/mass_1*2
    f_u[4,8] = 1/mass_1*2*u[0]
    f_u[4,9] = -1/mass_1*2*(u[2] - u[0])
    f_u[5,1] = u[8]/mass_1*2 + u[9]/mass_1*2
    f_u[5,3] = -u[9]/mass_1*2
    f_u[5,8] = 1/mass_1*2*u[1]
    f_u[5,9] = -1/mass_1*2*(u[3] - u[1])
    f_u[6,0] = -u[9]/mass_2*2
    f_u[6,2] = u[9]/mass_2*2
    f_u[6,9] = 1/mass_2*2*(u[2] - u[0])
    f_u[7,1] = -u[9]/mass_2*2
    f_u[7,3] = u[9]/mass_2*2
    f_u[7,9] = 1/mass_2*2*(u[3] - u[1])
    f_u[8,0] = 2*u[0]
    f_u[8,1] = 2*u[1]
    f_u[9,0] = -2*(u[2] - u[0])
    f_u[9,1] = -2*(u[3] - u[1])
    f_u[9,2] = 2*(u[2] - u[0])
    f_u[9,3] = 2*(u[3] - u[1])
    return f_u

# Определение входных данных задачи
t_0 = 0.; T = 14.16
x_1_0 = 3.; y_1_0 = -4.; x_2_0 = 3.; y_2_0 = -6.
v_1_x_0 = 0.; v_1_y_0 = 0.; v_2_x_0 = 0.; v_2_y_0 = 0.
g = 9.81; l_1 = 5.; l_2 = 2.; mass_1 = 1.0; mass_2 = 0.1
```

```
# Определение множителя Лагранжа
lambda_1_0 = lambda_2_0 = 100.

# 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 = 5000
# 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 + 1,10))

# Задание начальных условий
# (записываются в строку с индексом 0 массива u)
u[0,:] = [x_1_0, y_1_0, x_2_0, y_2_0, v_1_x_0, v_1_y_0, v_2_x_0, v_2_y_0,
lambda_1_0, lambda_2_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_1,mass_2),\
        f(u[m],g,mass_1,mass_2,l_1,l_2))
    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 1
    ax.plot((0,u[m,0]),(0,u[m,1]), color="white")
    # Отрисовка груза 1
    ax.plot(u[m,0],u[m,1], color="white", marker='o', markersize=12)
    # Отрисовка подвеса 2
    ax.plot((u[m,0],u[m,2]),(u[m,1],u[m,3]), color="white")
    # Отрисовка груза 2
    ax.plot(u[m,2],u[m,3], color="white", marker='o', markersize=6)
    # Отрисовка следа груза 1
    ax.plot(u[:m,0],u[:m,1],'-g',linewidth=1)
    # Отрисовка следа груза 2
    ax.plot(u[:m,2],u[:m,3],'-y',linewidth=1)
    if m%10==0 : # Сохраняем только каждый десятый кадр
        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 двойного маятника)
```