```
from numpy import *
from matplotlib.pyplot import *
from matplotlib.animation import *

# 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,g,mass,l):
    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 номером 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],g,mass,l),\
        f(u[m],g,mass,l))
    u[m + 1] = u[m] + tau*w_1.real

# Отрисовка решения
fig = figure()
ax = axes(xlim=(-5*1.62,5*1.62), ylim=(-7.5,3.5))
plot(0,0,'or', color="red", marker='o', markersize=7)
plot((-2,2),(0,0),'-')
rod, = ax.plot([], [], color="blue", lw=3)
bob, = ax.plot([], [], color="blue", marker='o', markersize=7)
def init():
    rod.set_data([], [])
    bob.set_data([], [])
    return rod,bob,
def animate(i):
    rod.set_data((0,u[i,0]),(0,u[i,1]))
    bob.set_data(u[i,0],u[i,1])
    return rod,bob,
anim = FuncAnimation(fig, animate, init_func=init,frames = M + 1, interval = 15,
blit=True)
```

Комментарий к файлу:

Листинг программы, реализующей решение задачи Коши для дифференциально-алгебраической системы уравнений, возникающей при моделировании движения материальной точки под действием силы тяжести и жёстком условии связи.