```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, cos, pi, eye, 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')

# Определение функции, задающей начальное условие
def u_init_0(x) :
    if x >= 4 and x <= 6:
        u_init_0 = cos(pi/2*(x - 5))
    else:
        u_init_0 = 0.
    return u_init_0

# Определение функции, задающей второе начальное условие
def u_init_1(x) :
    u_init_1 = 0.
    return u_init_1

# Определение функции, задающей левое граничное условие
def u_left(t) :
    u_left = 0.
    return u_left

# Определение функции, задающей правое граничное условие
def u_right(t) :
    u_right = 0.
    return u_right

def g(x,t):
    g = 0.
    return g

# Функция f подготавливает массив, содержащий элементы вектор-функции,
# определяющей правую часть решаемой системы ОДУ
def f(y,x,N,h,t,c,u_left,u_right,g) :
    f = zeros(2*N-2)
    for n in range(0,N-1) :
        f[n] = y[N-1+n]
    f[N-1] = c**2*(y[1] - 2*y[0] + u_left(t))/h**2 + g(x[1],t)
    for n in range(N,2*N-3) :
        f[n] = c**2*(y[n-N+2] - 2*y[n-N+1] + y[n-N])/h**2 + g(x[n-N+2],t)
    f[2*N-3] = c**2*(u_right(t) - 2*y[N-2] + y[N-3])/h**2 + g(x[N-1],t)
    return f

# Функция подготавливает массив, содержащий элементы матрицы Якоби f_u
def f_y(y,x,N,h,t,c,u_left,u_right,g) :
    f_y = zeros((2*N-2,2*N-2))
    # Определениене ненулевых элементов матрицы Якоби
    for n in range(0,N-1) :
        f_y[n,N-1+n] = 1.
    for n in range(N-1,2*N-3) :
        f_y[n,n-N+2] = c**2/h**2
    for n in range(N-1,2*N-2) :
        f_y[n,n-N+1] = -2*c**2/h**2
    for n in range(N,2*N-2):
        f_y[n,n-N] = c**2/h**2
    return f_y

# Определение входных данных задачи
a = 0.; b = 10.
t_0 = 0.; T = 20.0
c = 1.

# Определение параметра схемы (нужный раскомментировать)
```

```python
alpha = (1 + 1j)/2  # CROS1 (схема Розенброка с комплексным коэффициентом)
# alpha = 1.         # DIRK1 (обратная схема Эйлера)

# Определение числа интервалов пространственно-временной сетки,
# на которой будет искаться приближённое решение
N = 500; M = 1000

# Определение сетки по пространству
h = (b - a)/N; x = linspace(a,b,N+1)
# Определение сетки по времени
tau = (T - t_0)/M; t = linspace(t_0,T,M+1)

# Выделение памяти под массив сеточных значений решения УЧП
# В строке с номером m этого массива будут храниться сеточные значения решения,
# соответствующие моменту времени t_m
u = zeros((M + 1,N + 1))
# Выделение памяти под вспомогательный массив y
y = zeros((M + 1,2*N - 2))

# Задание начального условия (на начальном временном слое)
for n in range(N + 1) :
    u[0,n] = u_init_0(x[n])

# Задание начального условия решаемой системы ОДУ
for n in range(N-1) :
    y[0,n] = u_init_0(x[n+1])
    y[0,n + N-1] = u_init_1(x[n+1])

# Реализация схемы из семейства ROS1
# (конкретная схема определяется коэффициентом alpha)
for m in range(M) :
    w_1 = linalg.solve(eye(2*N-2) - alpha*tau*f_y(y[m],x,N,h,t[m],c,u_left,u_right,g),f(y[m],x,N,h,t[m] + tau/2,c,u_left,u_right,g))
    y[m + 1] = y[m] + tau*w_1.real
    u[m + 1,0] = u_left(t[m+1])
    u[m + 1,1:N] = y[m + 1,0:N-1]
    u[m + 1,N] = u_right(t[m+1])

# Анимация отрисовки решения
style.use('dark_background')
fig = figure()
camera = Camera(fig)
ax = axes(xlim=(a,b), ylim=(-2.,2.))
ax.set_xlabel('x'); ax.set_ylabel('u')
# Отрисовываем только каждый 2-й кадр
for m in range(0,M + 1,2) :
    # Отрисовка решения в момент времени t_m
    ax.plot(x,u[m], color='y', ls='-', lw=2)
    camera.snap()
animation = camera.animate(interval=15, repeat=False, blit=True)

# Листинг программы, реализущей решение линейного уравнения
# гиперболического типа, описывающего колебание струны
# с помощью метода прямых
```