```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, sin, pi, 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(x) :
    u_init = -100*sin(pi*x)**100 + 100*x**2
    return u_init

# Функция f подготавливает массив, содержащий элементы вектор-функции,
# определяющей правую часть решаемой системы ОДУ
def f(y,h,N):
    f = zeros(N-1)
    f[0] = -1/2*y[0] - h/8*y[0]**2
    f[1] = -(y[1] - 7/4*y[0]) - h/2*y[0]*(y[1] - 1/4*y[0])
    for n in range(2,N-1) :
        f[n] = -(y[n] - 2*y[n-1] + y[n-2]) - h/2*y[n-1]*(y[n] - y[n-2])
    return f

# Функция подготавливает матрицу дифференциального оператора решаемой системы ОДУ
def D(N,h) :
    D = zeros((N-1,N-1))
    # Определениене ненулевых элементов матрицы D
    D[0,0] = 1/4*(2 - h**2)
    D[1,0] = -(7/4 + h**2)
    D[1,1] = 1.
    for n in range(2,N-1) :
        D[n,n-2] = 1.
        D[n,n-1] = -(2 + h**2)
        D[n,n] = 1.
    return D

# Функция подготавливает массив, содержащий элементы матрицы Якоби f_u
def f_y(y,h,N):
    f_y = zeros((N-1,N-1))
    # Определениене ненулевых элементов матрицы Якоби
    f_y[0,0] = -1/2 - h/4*y[0]
    f_y[1,0] = 7/4 - h/2*(y[1] - 1/2*y[0])
    f_y[1,1] = -1 - h/2*y[0]
    for n in range(2,N-1) :
        f_y[n,n-2] = -1 + h/2*y[n-1]
        f_y[n,n-1] = 2 - h/2*(y[n] - y[n-2])
        f_y[n,n] = -1 - h/2*y[n-1]
    return f_y

# Определение входных данных задачи
a = 0.; b = 1.
t_0 = 0.;  T = 0.5

# Определение параметра схемы (нужный раскомментировать)
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 = 200; M = 250

# Определение сетки по пространству
```

```python
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,N - 1))

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

# Задание начального условия решаемой системы ОДУ
y[0] = u[0,2:N+1]

# Реализация схемы из семейства ROS1
# (конкретная схема определяется коэффициентом alpha)
for m in range(M) :
    w_1 = linalg.solve(D(N,h) - alpha*tau*f_y(y[m],h,N),f(y[m],h,N))
    y[m + 1] = y[m] + tau*w_1.real
    u[m + 1,0] = 0
    u[m + 1,1] = 1/4*y[m + 1,0]
    u[m + 1,2:N+1] = y[m + 1]

# Анимация отрисовки решения
style.use('dark_background')
fig = figure()
camera = Camera(fig)
ax = axes(xlim=(a,b), ylim=(-130.,130.))
ax.set_xlabel('x'); ax.set_ylabel('u')
for m in range(M + 1) :
    # Отрисовка решения в момент времени t_m
    ax.plot(x,u[m], color='y', ls='-', lw=2)
    camera.snap()
animation = camera.animate(interval=50, repeat=False, blit=True)

# Листинг программы, реализущей решение нелинейного уравнения
# типа Бенджамена-Бона-Махони-Бюргерса методом прямых
```