```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, tanh, 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(x) :
    u_init = 0.5*tanh((x - x_0)/eps)
    return u_init

# Определение функции, задающей левое граничное условие
def u_left(t) :
    u_left = -0.5
    return u_left

# Определение функции, задающей правое граничное условие
def u_right(t) :
    u_right = 0.5
    return u_right

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

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

# Определение входных данных задачи
a = 0.; b = 1.
t_0 = 0.; T = 6.0

x_0 = 0.6
eps = 10**(-2.0)

# 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 = 200

# 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,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[0,1:N]

# Реализация схемы из семейства ROS1
# (конкретная схема определяется коэффициентом alpha)
for m in range(M) :
    w_1 = linalg.solve(eye(N-1) - alpha*tau*f_y(y[m],t[m],h,N,u_left,u_right,eps),f(y[m],t[m] + tau/2,h,N,u_left,u_right,eps))
    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]
    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=(-1.,1.))
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 Бюргерса методом прямых
```