```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, exp, 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 = -x + 1.
    return u_init

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

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

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

# Определение входных данных задачи
a = 0.; b = 1.
t_0 = 0.; T = 0.3

# 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 = 300

# 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))

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

# Заполнение строки с индексом 0 вспомогательного массива
y[0] = u[0,1:N+1]

```

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

# Анимация отрисовки решения
style.use('dark_background')
fig = figure()
camera = Camera(fig)
ax = axes(xlim=(a,b), ylim=(0.,3.))
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=15, repeat=False, blit=True)

# Листинг программы, реализущей решение нелинейного уравнения переноса
# методом прямых
```