```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, tanh, complex64
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 = 8/7*tanh((x - x_0)/eps)
    return u_init

# Определение функции, задающей левое граничное условие
def u_left(t) :
    u_left = -8/7
    return u_left

# Определение функции, задающей правое граничное условие
def u_right(t) :
    u_right = 8/7
    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 массивы, которые содержат
# элементы диагоналей трёхдиагональной матрицы
# [E - alpha*tau*f_y]
def DiagonalsPreparation(y,t,h,N,u_left,u_right,eps,tau,alpha) :
    # Входные данные:
    # y - решение системы ОДУ в текущий момент времени
    # t - текущий момент времени t_m
    # tau - текущий шаг по времени
    # x - сетка по пространственной координате x
    # h - шаг сетки
    # N - число интервалов сетки
    # u_left - функция, определяющая левое граничное условие

    # Выходные параметры:
    # a, b и c - диагонали трёхдиагональной матрицы
    #
    #  [ a(0)  c(0)                           ]
    #  [ b(1)  a(1)  c(1)                     ]
    #  [       b(2)  a(2)  c(2)               ]
    #  [             ...   ...   ...          ]
    #  [                   ...   ...   c(N-1) ]
    #  [                         b(N-2) a(N-2) ]

    # Выделение памяти под массивы,
    # содержащие соответствующие диагонали
    a = zeros(N-1,dtype=complex64)
    b = zeros(N-1,dtype=complex64)
    c = zeros(N-1,dtype=complex64)

    a[0] = 1. - alpha*tau*(-2*eps/h**2 + (y[1] - u_left(t))/(2*h) + 3*y[0]**2)
    c[0] = - alpha*tau*(eps/h**2 + y[0]/(2*h))
```

```
    for n in range(1,N-2) :
        b[n] = - alpha*tau*(eps/h**2 - y[n]/(2*h))
        a[n] = 1. - alpha*tau*(-2*eps/h**2 + (y[n+1] - y[n-1])/(2*h) + 3*y[n]**2)
        c[n] = - alpha*tau*(eps/h**2 + y[n]/(2*h))
    b[N-2] = - alpha*tau*(eps/h**2 - y[N-2]/(2*h))
    a[N-2] = 1. - alpha*tau*(-2*eps/h**2 + (u_right(t) - y[N-3])/(2*h) + 3*y[N-2]**2)

    return a, b, c

def TridiagonalMatrixAlgorithm(a,b,c,B) :
    # Функция реализует метод прогонки (алгоритм Томаса)
    # для решения СЛАУ A X = B с трёхдиагональной матрицей

    # Входные параметры:
    # B - вектор правой части длины n
    # a, b, c - вектора длины n, содержащие элементы
    # диагоналей (b(1) и c(n) не используются)

    # [ a(1) c(1)                      ] [ X(1) ]   [ B(1) ]
    # [ b(2) a(2) c(2)                 ] [ X(2) ]   [ B(2) ]
    # [      b(3) a(3) c(3)            ] [      ]   [      ]
    # [           ...  ...  ...        ] [ ...  ] = [ ...  ]
    # [                ...  ...  c(n-1)] [X(n-1)]   [B(n-1)]
    # [                     b(n)  a(n) ] [ X(n) ]   [ B(n) ]

    n = len(B)
    v = zeros(n,dtype=complex64)
    X = zeros(n,dtype=complex64)

    w = a[0]
    X[0] = B[0]/w
    for i in range(1,n) :
        v[i - 1] = c[i - 1]/w
        w = a[i] - b[i]*v[i - 1]
        X[i] = (B[i] - b[i]*X[i - 1])/w
    for j in range(n-2,-1,-1) :
        X[j] = X[j] - v[j]*X[j + 1]

    return X

# Функция находит приближённое решение уравнения в частных производных (УрЧП/PDE)
def PDESolving(a,b,N,t_0,T,M,u_init,u_left,u_right,eps,alpha) :
    # Входные параметры:
    # a, b - границы области по пространственно переменной x
    # N - число интервалов сетки по пространству
    # t_0, T - начальный и конечный моменты счёта
    # M - число интервалов сетки по времени
    # u_init - функция, определяющяя начальное условие
    # u_left - функции, определяющие левое и правое граничные условия

    # alpha - коэффициент, определяющий численную схему

    # Выходной параметр:
    # u - массив, содержащий сеточные значения решения УрЧП

    # Формирование сетки :
    # 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 УрЧП,
    u = zeros((M + 1,N + 1))
    # Выделение памяти под вспомогательный массив y,
    # в котором хранятся решения системы ОДУ в текущий момент времени t = t_m
    y = zeros(N - 1)

    # Задание начального условия (на начальном временном слое)
    u[0] = u_init(x)
```

```
    # Задание начального условия решаемой системы ОДУ
    y = u_init(x[1:N])

    # Реализация схемы из семейства ROS1
    # (конкретная схема определяется коэффициентом alpha)
    for m in range(M) :
        diagonal,codiagonal_down,codiagonal_up = DiagonalsPreparation(y,t[m],h,N,u_left,u_right,eps,tau,alpha)
        w_1 = TridiagonalMatrixAlgorithm(diagonal,codiagonal_down,codiagonal_up,f(y,t[m] + tau/2,h,N,u_left,u_right,eps))
        # Переопределение y в новый момент времени t_{m+1}
        y = y + tau*w_1.real

        u[m + 1,0] = u_left(t[m+1])
        u[m + 1,1:N] = y
        u[m + 1,N] = u_right(t[m+1])

    return u

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

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 = 300; M = 500

u  = PDESolving(a,b,N,t_0,T,M,u_init,u_left,u_right,eps,alpha)

# Анимация отрисовки решения
style.use('dark_background')
fig = figure()
camera = Camera(fig)
ax = axes(xlim=(a,b), ylim=(-10.,3.))
ax.set_xlabel('x'); ax.set_ylabel('u')
for m in range(M + 1) :
    # Отрисовка решения в момент времени t_m
    ax.plot(linspace(a,b,N+1),u[m], color='y', ls='-', lw=2)
    camera.snap()
animation = camera.animate(interval=20, repeat=False, blit=True)

# Листинг программы, реализущей решение нелинейного уравнения
# типа Бюргерса методом прямых
```