```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, tanh
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

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

x_0 = 0.6
eps = 10**(-1.5)

# Определение числа интервалов пространственно-временной сетки,
# на которой будет искаться приближённое решение
N = 200; M = 20000

# 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))

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

# Задание граниченых условий
for m in range(1,M + 1) :
    u[m,0] = u_left(t[m])
    u[m,N] = u_right(t[m])

# Реализация поиска приближённого решения
# с помощью явной разностной схемы
for m in range(M) :
    # Вычисление решения на новом временном слое t_{m+1}
    for n in range(1,N) :
        u[m + 1,n] = u[m,n] + eps*tau/h**2*(u[m,n+1] - 2*u[m,n] + u[m,n-1]) +
        tau/(2*h)*u[m,n]*(u[m,n+1] - u[m,n-1]) + tau*u[m,n]**3

# 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')
```

```
69  # Отрисовываем только каждый 30-й кадр
70  for m in range(0,M + 1,30) :
71      # Отрисовка решения в момент времени t_m
72      ax.plot(x,u[m], color='y', ls='-', lw=2)
73      camera.snap()
74  animation = camera.animate(interval=15, repeat=False, blit=True)
75
76  # Листинг программы, реализущей решение нелинейного уравнения
77  # типа Бюргерса с помощью явной разностной схемы
```