```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import linspace, empty, tanh, meshgrid
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,y) :
    u_init = 0.5*tanh(1/eps*((x - 0.5)**2 + (y - 0.5)**2 - 0.35**2)) - 0.17
    return u_init

# Определение функций, задающих граничные условия 1-го рода (условия Дирихле)
def u_left(y,t) :
    u_left = 0.33
    return u_left

def u_right(y,t) :
    u_right = 0.33
    return u_right

def u_top(x,t) :
    u_top = 0.33
    return u_top

def u_bottom(x,t) :
    u_bottom = 0.33
    return u_bottom

# Определение входных данных задачи
a = -2.; b = 2.
c = -2.; d = 2.
t_0 = 0.; T = 5.

eps = 10**(-1.0)

# Определение числа интервалов пространственно-временной сетки,
# на которой будет искаться приближённое решение
N_x = 50; N_y = 50; M = 500

# Определение сеток по пространству
h_x = (b - a)/N_x; x = linspace(a,b,N_x+1)
h_y = (d - c)/N_y; y = linspace(c,d,N_y+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
u = empty((M + 1,N_x + 1,N_y + 1))

# Задание начального условия (на начальном временном слое)
for i in range(N_x + 1) :
    for j in range(N_y + 1) :
        u[0,i,j] = u_init(x[i],y[j])

# Задание граничных условий
for m in range(1,M + 1) :
    for j in range(1,N_y) :
        u[m,0,j] = u_left(y[j],t[m])
        u[m,N_x,j] = u_right(y[j],t[m])
    for i in range(0,N_x + 1) :
        u[m,i,N_y] = u_top(x[i],t[m])
        u[m,i,0] = u_bottom(x[i],t[m])

# Реализация поиска приближённого решения
# с помощью явной разностной схемы
```

```
for m in range(M) :
    print('m=',m)
    # Вычисление решения на новом временном слое t_{m+1}
    for i in range(1,N_x) :
        for j in range(1,N_y) :
            u[m+1,i,j] =  u[m,i,j] + \
                tau*(eps*((u[m,i+1,j] - 2*u[m,i,j] + u[m,i-1,j])/h_x**2 +
                          (u[m,i,j+1] - 2*u[m,i,j] + u[m,i,j-1])/h_y**2) +
                     u[m,i,j]*((u[m,i+1,j] - u[m,i-1,j])/(2*h_x) +
                               (u[m,i,j+1] - u[m,i,j-1])/(2*h_y)) +
                     u[m,i,j]**3)

# Анимация отрисовки решения
style.use('dark_background')
fig = figure()
camera = Camera(fig)
ax = fig.add_subplot(111,xlim=(a,b), ylim=(c,d))
ax.set_xlabel('x'); ax.set_ylabel('y'); ax.set_aspect('equal')
X, Y = meshgrid(x,y)
for m in range(M + 1) :
    # Отрисовка решения в момент времени t_m
    ax.pcolor(X,Y,u[m,:,:],shading='auto')
    camera.snap()
animation = camera.animate(interval=20, repeat=False, blit=True)

# # Анимация отрисовки решения
# style.use('dark_background')
# fig = figure()
# camera = Camera(fig)
# ax = fig.add_subplot(111, projection='3d',xlim=(a,b), ylim=(c,d), zlim=(0.,1.2))
# ax.set_xlabel('x'); ax.set_ylabel('y'); ax.set_zlabel('u')
# X, Y = meshgrid(x,y)
# for m in range(M + 1) :
#     # Отрисовка решения в момент времени t_m
#     ax.plot_wireframe(X,Y,u[m,:,:])
#     camera.snap()
# animation = camera.animate(interval=20, repeat=False, blit=True)

# Листинг программы, реализущей решение двумерного нелинейного уравнения
# типа Бюргерса с помощью чисто явной схемы
```