```
#! python3.7
# -*- coding: utf-8 -*-
from numpy import linspace, empty, tanh, meshgrid
from scipy import sparse
from scipy.sparse.linalg import spsolve
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

# Функция f подготавливает массив, содержащий элементы вектор-функции,
# определяющей правую часть решаемой системы ОДУ
def f(p,t,x,h_x,N_x,y,h_y,N_y,u_left,u_right,u_top,u_bottom,eps) :
    f = empty((N_x - 1)*(N_y - 1))

    f[0] =  eps/h_x**2*(p[1] - 2*p[0] + u_left(y[1],t)) + \
        eps/h_y**2*(p[N_x-1] - 2*p[0] + u_bottom(x[1],t)) + \
        1/(2*h_x)*p[0]*(p[1] - u_left(y[1],t)) + \
        1/(2*h_y)*p[0]*(p[N_x-1] - u_bottom(x[1],t)) + p[0]**3
    for i in range(1,N_x-2) :
        f[i] =  eps/h_x**2*(p[i+1] - 2*p[i] + p[i-1]) + \
            eps/h_y**2*(p[(N_x-1) + i] - 2*p[i] + u_bottom(x[i+1],t)) + \
            1/(2*h_x)*p[i]*(p[i+1] - p[i-1]) + \
            1/(2*h_y)*p[i]*(p[(N_x-1) + i] - u_bottom(x[i+1],t)) + p[i]**3
    f[N_x-2] =  eps/h_x**2*(u_right(y[1],t) - 2*p[N_x-2] + p[N_x-3]) + \
        eps/h_y**2*(p[(N_x-1) + (N_x-2)] - 2*p[N_x-2] + u_bottom(x[N_x-1],t)) + \
        1/(2*h_x)*p[N_x-2]*(u_right(y[1],t) - p[N_x-3]) + \
        1/(2*h_y)*p[N_x-2]*(p[(N_x-1) + (N_x-2)] - u_bottom(x[N_x-1],t)) + p[N_x-2]**3

    for j in range(1,N_y-2) :
        f[(N_x-1)*j + 0] =  eps/h_x**2*(p[(N_x-1)*(j) + (1)] - 2*p[(N_x-1)*(j) +
        (0)] + u_left(y[j+1],t)) + \
            eps/h_y**2*(p[(N_x-1)*(j+1) + (0)] - 2*p[(N_x-1)*(j) + (0)] +
            p[(N_x-1)*(j-1) + (0)]) + \
            1/(2*h_x)*p[(N_x-1)*(j) + (0)]*(p[(N_x-1)*(j) + (1)] - u_left(y[j+1],t))
            + \
            1/(2*h_y)*p[(N_x-1)*(j) + (0)]*(p[(N_x-1)*(j+1) + (0)] - p[(N_x-1)*(j-1)
            + (0)]) + p[(N_x-1)*(j) + (0)]**3
        for i in range(1,N_x-2) :
            f[(N_x-1)*j + i] =  eps/h_x**2*(p[(N_x-1)*(j) + (i+1)] - 2*p[(N_x-1)*(j)
            + (i)] + p[(N_x-1)*(j) + (i-1)]) + \
                eps/h_y**2*(p[(N_x-1)*(j+1) + (i)] - 2*p[(N_x-1)*(j) + (i)] +
                p[(N_x-1)*(j-1) + (i)]) + \
                1/(2*h_x)*p[(N_x-1)*(j) + (i)]*(p[(N_x-1)*(j) + (i+1)] -
                p[(N_x-1)*(j) + (i-1)]) + \
```

```
                1/(2*h_y)*p[(N_x-1)*(j) + (i)]*(p[(N_x-1)*(j+1) + (i)] -
                p[(N_x-1)*(j-1) + (i)]) + p[(N_x-1)*(j) + (i)]**3
        f[(N_x-1)*j + N_x-2] =  eps/h_x**2*(u_right(y[j+1],t) - 2*p[(N_x-1)*(j) +
        (N_x-2)] + p[(N_x-1)*(j) + (N_x-3)]) + \
            eps/h_y**2*(p[(N_x-1)*(j+1) + (N_x-2)] - 2*p[(N_x-1)*(j) + (N_x-2)] +
            p[(N_x-1)*(j-1) + (N_x-2)]) + \
            1/(2*h_x)*p[(N_x-1)*(j) + (N_x-2)]*(u_right(y[j+1],t) - p[(N_x-1)*(j) +
            (N_x-3)]) + \
            1/(2*h_y)*p[(N_x-1)*(j) + (N_x-2)]*(p[(N_x-1)*(j+1) + (N_x-2)] -
            p[(N_x-1)*(j-1) + (N_x-2)]) + p[(N_x-1)*(j) + (N_x-2)]**3

    f[(N_x-1)*(N_y-2) + 0] =  eps/h_x**2*(p[(N_x-1)*(N_y-2) + (1)] -
    2*p[(N_x-1)*(N_y-2) + (0)] + u_left(y[N_y-1],t)) + \
        eps/h_y**2*(u_top(x[1],t) - 2*p[(N_x-1)*(N_y-2) + (0)] + p[(N_x-1)*(N_y-3) +
        (0)]) + \
        1/(2*h_x)*p[(N_x-1)*(N_y-2) + (0)]*(p[(N_x-1)*(N_y-2) + (1)] -
        u_left(y[j+1],t)) + \
        1/(2*h_y)*p[(N_x-1)*(N_y-2) + (0)]*(u_top(x[1],t) - p[(N_x-1)*(N_y-3) +
        (0)]) + p[(N_x-1)*(N_y-2) + (0)]**3
    for i in range(1,N_x-2) :
        f[(N_x-1)*(N_y-2) + i] =  eps/h_x**2*(p[(N_x-1)*(N_y-2) + (i+1)] -
        2*p[(N_x-1)*(N_y-2) + (i)] + p[(N_x-1)*(N_y-2) + (i-1)]) + \
            eps/h_y**2*(u_top(x[i+1],t) - 2*p[(N_x-1)*(N_y-2) + (i)] +
            p[(N_x-1)*(N_y-3) + (i)]) + \
            1/(2*h_x)*p[(N_x-1)*(N_y-2) + (i)]*(p[(N_x-1)*(N_y-2) + (i+1)] -
            p[(N_x-1)*(N_y-2) + (i-1)]) + \
            1/(2*h_y)*p[(N_x-1)*(N_y-2) + (i)]*(u_top(x[i+1],t) - p[(N_x-1)*(N_y-3)
            + (i)]) + p[(N_x-1)*(N_y-2) + (i)]**3
    f[(N_x-1)*(N_y-2) + N_x-2] =  eps/h_x**2*(u_right(y[N_y-1],t) -
    2*p[(N_x-1)*(N_y-2) + (N_x-2)] + p[(N_x-1)*(N_y-2) + (N_x-3)]) + \
        eps/h_y**2*(u_top(x[N_x-1],t) - 2*p[(N_x-1)*(N_y-2) + (N_x-2)] +
        p[(N_x-1)*(N_y-3) + (N_x-2)]) + \
        1/(2*h_x)*p[(N_x-1)*(N_y-2) + (N_x-2)]*(u_right(y[N_y-1],t) -
        p[(N_x-1)*(N_y-2) + (N_x-3)]) + \
        1/(2*h_y)*p[(N_x-1)*(N_y-2) + (N_x-2)]*(u_top(x[N_x-1],t) -
        p[(N_x-1)*(N_y-3) + (N_x-2)]) + p[(N_x-1)*(N_y-2) + (N_x-2)]**3

    return f

# Функция подготавливает матрицу СЛАУ в разреженном формате хранения данных
def A(p,t,x,h_x,N_x,y,h_y,N_y,u_left,u_right,u_top,u_bottom,eps,alpha,tau) :

    I = empty(12 + 8*(N_x-3) + 8*(N_y-3) + 5*(N_x - 3)*(N_y - 3))
    J = empty(12 + 8*(N_x-3) + 8*(N_y-3) + 5*(N_x - 3)*(N_y - 3))
    V = empty(12 + 8*(N_x-3) + 8*(N_y-3) + 5*(N_x - 3)*(N_y - 3))

    k = 0

    I[k] = 0; J[k] = 0; V[k] = eps/h_x**2*(-2) + eps/h_y**2*(- 2) + \
        1/(2*h_x)*(p[1] - u_left(y[1],t)) + 1/(2*h_y)*(p[N_x-1] - u_bottom(x[1],t))
        + 3*p[0]**2
    k = k + 1

    I[k] = 0; J[k] = 1; V[k] = eps/h_x**2*(1) + 1/(2*h_x)*p[0]
    k = k + 1

    I[k] = 0; J[k] = N_x-1; V[k] = eps/h_y**2*(1) + 1/(2*h_y)*p[0]*(1)
    k = k + 1

    for i in range(1,N_x-2) :

        I[k] = i; J[k] = i-1; V[k] = eps/h_x**2*(1) + 1/(2*h_x)*p[i]*(-1)
        k = k + 1

        I[k] = i; J[k] = i; V[k] = eps/h_x**2*(-2) + eps/h_y**2*(-2) + \
            1/(2*h_x)*(p[i+1] - p[i-1]) + 1/(2*h_y)*(p[(N_x-1) + i] -
            u_bottom(x[i+1],t)) + 3*p[i]**2
        k = k + 1

```

```
113             I[k] = i; J[k] = i+1; V[k] = eps/h_x**2*(1) + 1/(2*h_x)*p[i]*(1)
114             k = k + 1
115 
116             I[k] = i; J[k] = (N_x-1) + i; V[k] = eps/h_y**2*(1) + 1/(2*h_y)*p[i]
117             k = k + 1
118 
119         I[k] = N_x-2; J[k] = N_x-3; V[k] = eps/h_x**2*(1) + 1/(2*h_x)*p[N_x-2]*(-1)
120         k = k + 1
121 
122         I[k] = N_x-2; J[k] = N_x-2; V[k] = eps/h_x**2*(-2) + eps/h_y**2*(-2) + \
123             1/(2*h_x)*(u_right(y[1],t) - p[N_x-3]) +  1/(2*h_y)*(p[(N_x-1) + (N_x-2)] -
            u_bottom(x[N_x-1],t)) + 3*p[N_x-2]**2
124         k = k + 1
125 
126         I[k] = N_x-2; J[k] = (N_x-1) + (N_x-2); V[k] =  eps/h_y**2*(1) +
        1/(2*h_y)*p[N_x-2]*(1)
127         k = k + 1
128 
129         for j in range(1,N_y-2) :
130 
131             I[k] = (N_x-1)*j + 0; J[k] = (N_x-1)*(j-1) + (0); V[k] = eps/h_y**2*(1) +
            1/(2*h_y)*p[(N_x-1)*(j) + (0)]*(-1)
132             k = k + 1
133 
134             I[k] = (N_x-1)*j + 0; J[k] = (N_x-1)*(j) + (0); V[k] = eps/h_x**2*(-2) +
            eps/h_y**2*(-2) + \
135                 1/(2*h_x)*(p[(N_x-1)*(j) + (1)] - u_left(y[j+1],t)) +
                1/(2*h_y)*(p[(N_x-1)*(j+1) + (0)] - p[(N_x-1)*(j-1) + (0)]) +
                3*p[(N_x-1)*(j) + (0)]**2
136             k = k + 1
137 
138             I[k] = (N_x-1)*j + 0; J[k] = (N_x-1)*(j) + (1); V[k] = eps/h_x**2*(1) +
            1/(2*h_x)*p[(N_x-1)*(j) + (0)]*(1)
139             k = k + 1
140 
141             I[k] = (N_x-1)*j + 0; J[k] = (N_x-1)*(j+1) + (0); V[k] = eps/h_y**2*(1) +
            1/(2*h_y)*p[(N_x-1)*(j) + (0)]*(1)
142             k = k + 1
143 
144             for i in range(1,N_x-2) :
145 
146                 I[k] = (N_x-1)*j + i; J[k] = (N_x-1)*(j-1) + (i) ; V[k] = eps/h_y**2*(1)
                + 1/(2*h_y)*p[(N_x-1)*(j) + (i)]*(-1)
147                 k = k + 1
148 
149                 I[k] = (N_x-1)*j + i; J[k] = (N_x-1)*(j) + (i-1); V[k] = eps/h_x**2*(1)
                + 1/(2*h_x)*p[(N_x-1)*(j) + (i)]*(-1)
150                 k = k + 1
151 
152                 I[k] = (N_x-1)*j + i; J[k] = (N_x-1)*(j) + (i); V[k] = eps/h_x**2*(-2) +
                eps/h_y**2*(-2) + \
153                     1/(2*h_x)*(p[(N_x-1)*(j) + (i+1)] - p[(N_x-1)*(j) + (i-1)]) + \
154                     1/(2*h_y)*(p[(N_x-1)*(j+1) + (i)] - p[(N_x-1)*(j-1) + (i)]) +
                    3*p[(N_x-1)*(j) + (i)]**2
155                 k = k + 1
156 
157                 I[k] = (N_x-1)*j + i; J[k] = (N_x-1)*(j) + (i+1); V[k] = eps/h_x**2*(1)
                + 1/(2*h_x)*p[(N_x-1)*(j) + (i)]*(1)
158                 k = k + 1
159 
160                 I[k] = (N_x-1)*j + i; J[k] = (N_x-1)*(j+1) + (i); V[k] = eps/h_y**2*(1)
                + 1/(2*h_y)*p[(N_x-1)*(j) + (i)]*(1)
161                 k = k + 1
162 
163             I[k] = (N_x-1)*j + N_x-2; J[k] = (N_x-1)*(j-1) + (N_x-2); V[k] =
            eps/h_y**2*(1) + 1/(2*h_y)*p[(N_x-1)*(j) + (N_x-2)]*(-1)
164             k = k + 1
165 
166             I[k] = (N_x-1)*j + N_x-2; J[k] = (N_x-1)*(j) + (N_x-3); V[k] =
```

```
            eps/h_x**2*(1) +  1/(2*h_x)*p[(N_x-1)*(j) + (N_x-2)]*(-1)
            k = k + 1

            I[k] = (N_x-1)*j + N_x-2; J[k] = (N_x-1)*(j) + (N_x-2) ; V[k] =
            eps/h_x**2*(-2) + eps/h_y**2*(-2) + \
                1/(2*h_x)*(u_right(y[j+1],t) - p[(N_x-1)*(j) + (N_x-3)]) + \
                1/(2*h_y)*(p[(N_x-1)*(j+1) + (N_x-2)] - p[(N_x-1)*(j-1) + (N_x-2)]) +
                3*p[(N_x-1)*(j) + (N_x-2)]**2
            k = k + 1

            I[k] = (N_x-1)*j + N_x-2; J[k] = (N_x-1)*(j+1) + (N_x-2); V[k] =
            eps/h_y**2*(1) + 1/(2*h_y)*p[(N_x-1)*(j) + (N_x-2)]*(1)
            k = k + 1

        I[k] = (N_x-1)*(N_y-2) + 0; J[k] = (N_x-1)*(N_y-3) + (0); V[k] = eps/h_y**2*(1)
        + 1/(2*h_y)*p[(N_x-1)*(N_y-2) + (0)]*(-1)
        k = k + 1

        I[k] = (N_x-1)*(N_y-2) + 0; J[k] = (N_x-1)*(N_y-2) + (0); V[k] = eps/h_x**2*(-2)
        + eps/h_y**2*(-2) + \
            1/(2*h_x)*(p[(N_x-1)*(N_y-2) + (1)] - u_left(y[j+1],t)) + \
            1/(2*h_y)*(u_top(x[1],t) - p[(N_x-1)*(N_y-3) + (0)]) + 3*p[(N_x-1)*(N_y-2) +
            (0)]**2
        k = k + 1

        I[k] = (N_x-1)*(N_y-2) + 0; J[k] = (N_x-1)*(N_y-2) + (1); V[k] = eps/h_x**2*(1)
        + 1/(2*h_x)*p[(N_x-1)*(N_y-2) + (0)]*(1)
        k = k + 1

        for i in range(1,N_x-2) :

            I[k] = (N_x-1)*(N_y-2) + i; J[k] = (N_x-1)*(N_y-3) + (i); V[k] =
            eps/h_y**2*(1) + 1/(2*h_y)*p[(N_x-1)*(N_y-2) + (i)]*(-1)
            k = k + 1

            I[k] = (N_x-1)*(N_y-2) + i; J[k] = (N_x-1)*(N_y-2) + (i-1); V[k] =
            eps/h_x**2*(1) + 1/(2*h_x)*p[(N_x-1)*(N_y-2) + (i)]*(-1)
            k = k + 1

            I[k] = (N_x-1)*(N_y-2) + i; J[k] = (N_x-1)*(N_y-2) + (i); V[k] =
            eps/h_x**2*(-2) + eps/h_y**2*(-2) + \
                1/(2*h_x)*(p[(N_x-1)*(N_y-2) + (i+1)] - p[(N_x-1)*(N_y-2) + (i-1)]) + \
                1/(2*h_y)*(u_top(x[i+1],t) - p[(N_x-1)*(N_y-3) + (i)]) +
                3*p[(N_x-1)*(N_y-2) + (i)]**2
            k = k + 1

            I[k] = (N_x-1)*(N_y-2) + i; J[k] = (N_x-1)*(N_y-2) + (i+1); V[k] =
            eps/h_x**2*(1) + 1/(2*h_x)*p[(N_x-1)*(N_y-2) + (i)]*(1)
            k = k + 1

        I[k] = (N_x-1)*(N_y-2) + N_x-2; J[k] = (N_x-1)*(N_y-3) + (N_x-2); V[k] =
        eps/h_y**2*(1) + 1/(2*h_y)*p[(N_x-1)*(N_y-2) + (N_x-2)]*(-1)
        k = k + 1

        I[k] = (N_x-1)*(N_y-2) + N_x-2; J[k] = (N_x-1)*(N_y-2) + (N_x-3); V[k] =
        eps/h_x**2*(1) + 1/(2*h_x)*p[(N_x-1)*(N_y-2) + (N_x-2)]*(-1)
        k = k + 1

        I[k] = (N_x-1)*(N_y-2) + N_x-2; J[k] = (N_x-1)*(N_y-2) + (N_x-2); V[k] =
        eps/h_x**2*(-2) + eps/h_y**2*(-2) + \
            1/(2*h_x)*(u_right(y[N_y-1],t) - p[(N_x-1)*(N_y-2) + (N_x-3)]) + \
            1/(2*h_y)*(u_top(x[N_x-1],t) - p[(N_x-1)*(N_y-3) + (N_x-2)]) +
            3*p[(N_x-1)*(N_y-2) + (N_x-2)]**2
        k = k + 1

        f_p = sparse.coo_matrix((V,(I,J)),shape=((N_x-1)*(N_y-1),(N_x-1)*(N_y-1))).tocsr()

        A = sparse.eye((N_x-1)*(N_y-1),format="csr") - alpha*tau*f_p

```

```
    return A

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

eps = 10**(-1.5)

# 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_x = 100; N_y = 100; 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))

# Выделение памяти под вспомогательный массив y
p = empty((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 начального условия решаемой системы ОДУ
for j in range(N_y - 1) :
    for i in range(N_x - 1) :
        p[(N_x-1)*j + i] = u[0,i+1,j+1]

# Реализация схемы из семейства ROS1
# (конкретная схема определяется коэффициентом alpha)
for m in range(M) :
    print('m=',m)
    w_1 =
    spsolve(A(p,t,x,h_x,N_x,y,h_y,N_y,u_left,u_right,u_top,u_bottom,eps,alpha,tau),
                  f(p,t,x,h_x,N_x,y,h_y,N_y,u_left,u_right,u_top,u_bottom,eps))
    p = p + tau*w_1.real
    # Заполняем массив u вычисленными сеточными значениями,
    # содержащимися в векторе p
    for j in range(N_y - 1) :
        for i in range(N_x - 1) :
            u[m+1,i+1,j+1] = p[(N_x-1)*j + i]
    # Учитываем граничные условия
    u[m + 1,0,:] = u_left(y[1:N_y],t[m+1])
    u[m + 1,N_x,:] = u_right(y[1:N_y],t[m+1])
    u[m + 1,:,N_y] = u_top(x[:],t[m+1])
    u[m + 1,:,0] = u_bottom(x[:],t[m+1])

# 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')
```

```
287         camera.snap()
288     animation = camera.animate(interval=20, repeat=False, blit=True)
289 
290     # # Анимация отрисовки решения
291     # style.use('dark_background')
292     # fig = figure()
293     # camera = Camera(fig)
294     # ax = fig.add_subplot(111, projection='3d',xlim=(a,b), ylim=(c,d), zlim=(0.,1.2))
295     # ax.set_xlabel('x'); ax.set_ylabel('y'); ax.set_zlabel('u')
296     # X, Y = meshgrid(x,y)
297     # for m in range(M + 1) :
298     #     # Отрисовка решения в момент времени t_m
299     #     ax.plot_wireframe(X,Y,u[m,:,:])
300     #     camera.snap()
301     # animation = camera.animate(interval=20, repeat=False, blit=True)
302 
303     # Листинг программы, реализущей решение двумерного нелинейного уравнения
304     # типа Бюргерса методом прямых
```