```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, cos, pi, abs, sign, 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 + 0.2
    return u_init

# Функция f подготавливает массив, содержащий элементы вектор-функции,
# определяющей правую часть решаемой системы ОДУ
def f(y,h,N) :
    f = zeros(N+1)
    f[0] = -3/2*y[0] + 2*y[1] - 1/2*y[2]
    f[1] = -5/2*y[0] + 9*y[1] - 12*y[2] + 7*y[3] - 3/2*y[4]
    sum = 0.
    for k in range(1,N) :
        sum = sum + abs(y[k])**(q - 1)*sign(y[k])
    for n in range(2,N-1) :
        f[n] = - 1/h**4*(y[n+2] - 4*y[n+1] + 6*y[n] - 4*y[n-1] + y[n-2]) + \
            1/h**p*(abs(y[n+1] - y[n])**(p - 1)*sign(y[n+1] - y[n]) - abs(y[n] -
            y[n-1])**(p - 1)*sign(y[n] - y[n-1])) +  \
                abs(y[n])**(q - 1)*sign(y[n]) -  1/l*(abs(y[0])**(q -
                1)*sign(y[0])/2 + sum + abs(y[N])**(q - 1)*sign(y[N])/2)*h
    f[N-1]= 5/2*y[N] - 9*y[N-1] + 12*y[N-2] - 7*y[N-3] + 3/2*y[N-4]
    f[N] = 3/2*y[N] - 2*y[N-1] + 1/2*y[N-2]
    return f

# Функция подготавливает матрицу дифференциального оператора решаемой системы ОДУ
def D(N) :
    D = zeros((N+1,N+1))
    # Определениене ненулевых элементов матрицы D
    for n in range(2,N-1) :
        D[n,n] = 1.
    return D

# Функция подготавливает массив, содержащий элементы матрицы Якоби f_u
def f_y(y,h,N) :
    f_y = zeros((N+1,N+1))
    f_y[0,0] = -3/2; f_y[0,1] = 2.; f_y[0,2] = -1/2
    f_y[1,0] = -5/2; f_y[1,1] = 9.; f_y[1,2] = -12.; f_y[1,3] = 7.; f_y[1,4] = -3/2
    for n in range(2,N-1) :
        f_y[n,n-2] = -1/h**4
        f_y[n,n-1] = 4/h**4 + 1/h**p*(p - 1)*abs(y[n] - y[n-1])**(p - 2)
        f_y[n,n] = -6/h**4 - 1/h**p*(p-1)*(abs(y[n+1]-y[n])**(p - 2) + abs(y[n] -
        y[n-1])**(p - 2)) + \
            (q - 1)*abs(y[n])**(q - 2)
        f_y[n,n+1] = 4/h**4 + 1/h**p*(p - 1)*abs(y[n+1] - y[n])**(p - 2)
        f_y[n,n+2] = -1/h**4
    f_y[N-1,N-4] = 3/2; f_y[N-1,N-3] = -7.; f_y[N-1,N-2] = 12.; f_y[N-1,N-1] = -9.;
    f_y[N-1,N] = 5/2
    f_y[N,N-2] = 1/2; f_y[N,N-1]= -2.; f_y[N,N] = 3/2
    for n in range(2,N-1) :
        f_y[n,0] = f_y[n,0] - 1/(2*l)*(q - 1)*abs(y[0])**(q - 2)*h
        for k in range(1,N) :
            f_y[n,k] = f_y[n,k] - 1/l*(q - 1)*abs(y[k])**(q - 2)*h
        f_y[n,N] = f_y[n,N] - 1/(2*l)*(q - 1)*abs(y[N])**(q - 2)*h
    return f_y

# Определение входных данных задачи
a = 0.; b = pi; l = b - a
t_0 = 0.; T = 0.04
p = 3.5; q = 4.5
```

```
66
67  # Определение параметра схемы (нужный раскомментировать)
68  alpha = (1 + 1j)/2  # CROS1 (схема Розенброка с комплексным коэффициентом)
69  # alpha = 1.         # DIRK1 (обратная схема Эйлера)
70
71  # Определение числа интервалов пространственно-временной сетки,
72  # на которой будет искаться приближённое решение
73  N = 50; M = 100
74
75  # Определение сетки по пространству
76  h = (b - a)/N; x = linspace(a,b,N+1)
77  # Определение сетки по времени
78  tau = (T - t_0)/M; t = linspace(t_0,T,M+1)
79
80  # Выделение памяти под массив сеточных значений решения УЧП
81  # В строке с номером m этого массива будут храниться сеточные значения решения,
82  # соответствующие моменту времени t_m
83  u = zeros((M + 1,N + 1))
84  # Выделение памяти под вспомогательный массив y
85  y = zeros((M + 1,N + 1))
86
87  # Задание начального условия (на начальном временном слое)
88  for n in range(N+1) :
89      u[0,n] = u_init(x[n])
90
91  # Задание начального условия решаемой системы ОДУ
92  y[0] = u[0]
93
94  # Реализация схемы из семейства ROS1
95  # (конкретная схема определяется коэффициентом alpha)
96  for m in range(M) :
97      w_1 = linalg.solve(D(N) - alpha*tau*f_y(y[m],h,N),f(y[m],h,N))
98      y[m + 1] = y[m] + tau*w_1.real
99      u[m + 1] = y[m + 1]
100
101 # Анимация отрисовки решения
102 style.use('dark_background')
103 fig = figure()
104 camera = Camera(fig)
105 ax = axes(xlim=(a,b), ylim=(-2.,9.))
106 ax.set_xlabel('x'); ax.set_ylabel('u')
107 # Отрисовываем только каждый 1-й кадр
108 for m in range(0,M + 1,1) :
109     # Отрисовка решения в момент времени t_m
110     ax.plot(x,u[m], color='y', ls='-', lw=2)
111     camera.snap()
112 animation = camera.animate(interval=15, repeat=False, blit=True)
113
114 # Листинг программы, реализущей решение нелинейного интегро-дифференциального
    уравнения,
115 # которое описывает эпитаксиальный рост наноразмерных тонких плёнок
```