```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace
from matplotlib.pyplot import style, figure, axes

def k(x) :
    if x < x_special :
        k = 0.3
    elif x >= x_special :
        k = 1.
    return k

def r(x) :
    return 1.

def f(x) :
    return 10.

# Определение функции, вычисляющей значение x_n
def x(n) :
    x = a + (b-a)/N*n
    return x

# Функция подготавливает массивы, которые содержат
# элементы диагоналей трёхдиагональной матрицы решаемой СЛАУ
def DiagonalsPreparation() :

    # 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)   a(N)   ]

    # Выделение памяти под массивы,
    # содержащие элементы соответствующих диагоналей
    a = zeros(N+1); b = zeros(N+1); c = zeros(N+1)

    # Заполнение массивов ненулевыми значениями трёхдиагональной матрицы
    a[0] = 1.
    for n in range(1,N) :
        b[n] = 2*k(x(n-1/2))/(x(n) - x(n-1)) - 1/2*r(x(n-1/2))*(x(n) - x(n-1))
        a[n] = -2*k(x(n+1/2))/(x(n+1) -x(n)) - 2*k(x(n-1/2))/(x(n) - x(n-1))-\
            1/2*r(x(n+1/2))*(x(n+1) - x(n)) - 1/2*r(x(n-1/2))*(x(n) - x(n-1))
        c[n] = 2*k(x(n+1/2))/(x(n+1) - x(n)) - 1/2*r(x(n+1/2))*(x(n+1) - x(n))
    a[N] = 1.

    return a, b, c

# Функция подготавливает вектор правой части решаемой СЛАУ
def B() :
    B = zeros(N+1)
    B[0] = u_left
    for n in range(1,N) :
        B[n] = f(x(n-1/2))*(x(n) - x(n-1)) + f(x(n+1/2))*(x(n+1) - x(n))
    B[N] = u_right
    return B

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

```python

    # Входные параметры:
    # B - вектор правой части длины n
    # a, b, c - вектора длины n, содержащие элементы
    # диагоналей (b(0) и c(n-1) не используются)

    # [ a(0) c(0)                       ] [ X(0) ]   [ B(0) ]
    # [ b(1) a(1) c(1)                  ] [ X(1) ]   [ B(1) ]
    # [      b(2) a(2) c(2)             ] [      ]   [      ]
    # [           ...  ...  ...         ] [ ...  ] = [ ...  ]
    # [                ...  ...  c(n-2) ] [X(n-2)]   [B(n-2)]
    # [                   b(n-1) a(n-1) ] [X(n-1)]   [B(n-1)]

    n = len(B)
    v = zeros(n)
    X = zeros(n)

    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

# Определение входных данных задачи
a = 0.; b = 1.
u_left = 0.; u_right = 0.

# Определение точки разрыва коэффициента
x_special = 0.7

# Определение числа интервалов сетки,
# на которой будет искаться приближённое решение
N = 50

# Вычисление решения
u = TridiagonalMatrixAlgorithm(*DiagonalsPreparation(),B())

# Отрисовка решения
style.use('dark_background')

fig1 = figure()
ax1 = axes(xlim=(a,b), ylim=(-3.,3.))
ax1.set_xlabel('x'); ax1.set_ylabel('u')
ax1.plot([x(n) for n in range(N+1)],u,'-ow',markersize=5)

# Листинг программы, реализущей приближённое решение
# краевой задачи для ОДУ второго порядка с помощью бикомпактной схемы
# в случае слоистой среды
```