Содержание

1 Задание **2**

2 Теоретическая часть **2**

2.1 Тестовые примеры 2

2.1.1 Тестовая задача №1 2

2.1.2 Тестовая задача №2 2

2.1.3 Тестовая задача №3 3

2.1.4 Тестовая задача №4 3

2.2 Локально – одномерная схема 2 3

3 Практическая часть **3**

3.1 Тестовый пример №1 4

3.2 Тестовый пример №2 4

3.3 Тестовый пример №3 4

3.4 Тестовый пример №4 4

4 Приложение **5**

1 Задание

Рассматривается начально-краевая задача для двумерного уравнения теплопроводности:

$$\begin{cases} u_t = a(u_{xx} + u_{yy}) + f,\ (x,y) \in \Omega = (0, L_1) \times (0, L_2), t \in (0, T), \\ u|_{t=0} = \varphi, \\ u|_{\partial\Omega} = g. \end{cases} \quad (1)$$

Здесь $a > 0$ – константа, решение $u = u(t,x,y)$.

Реализовать разностную схему РС2.

Подготовить несколько стандартных тестовых примеров.

Организовать численный эксперимент для исследования устойчивости РС, а также порядка точности РС по каждой переменной.

При измельчении шага по времени расчет следует вести до одного и того же значения $T = 5(L_1 + L_2)$. Предусмотреть возможность исследования прямоугольных областей с разным соотношением сторон.

2 Теоретическая часть

2.1 Тестовые примеры

2.1.1 Тестовая задача №1

$u(t,x,y) = 2x + 3y + 5t$

$$\begin{cases} u_t = a(u_{xx} + u_{yy}) + 5,\ (x,y) \in \Omega = (0, L_1) \times (0, L_2), t \in (0, T), \\ u|_{t=0} = 2x + 3y, \\ u|_{\partial\Omega} = \begin{cases} 2x + 5t,\ y = 0 \\ 3y + 5t,\ x = 0 \\ 2L_1 + 3y + 5t,\ x = L_1 \\ 2x + 3L_2 + 5t,\ y = L_2 \end{cases} \end{cases} \quad (2)$$

2.1.2 Тестовая задача №2

$u(t,x,y) = 2x^2 + y - 3t + 2$

$$\begin{cases} u_t = a(u_{xx} + u_{yy}) - 4a - 3,\ (x,y) \in \Omega = (0, L_1) \times (0, L_2), t \in (0, T), \\ u|_{t=0} = 2x^2 + y + 2, \\ u|_{\partial\Omega} = \begin{cases} 2x^2 - 3t + 2,\ y = 0 \\ y - 3t + 2,\ x = 0 \\ 2L_1^2 + y - 3t + 2,\ x = L_1 \\ 2x^2 + L_2 - 3t + 2,\ y = L_2 \end{cases} \end{cases} \quad (3)$$

2.1.3 Тестовая задача №3

$u(t,x,y) = 3x^2 - 2y^2 + t + 4$

$$\begin{cases} u_t = a(u_{xx} + u_{yy}) + 1 - 2a,\ (x,y) \in \Omega = (0, L_1) \times (0, L_2), t \in (0, T), \\ u|_{t=0} = 3x^2 - 2y^2 + 4, \\ u|_{\partial\Omega} = \begin{cases} 3x^2 + t + 4,\ y = 0 \\ -2y^2 + t + 4,\ x = 0 \\ 3L_1^2 - 2y^2 + t + 4,\ x = L_1 \\ 3x^2 - 2L_2^2 + t + 4,\ y = L_2 \end{cases} \end{cases} \tag{4}$$

2.1.4 Тестовая задача №4

$u(t,x,y) = 4x^3 - y^3 + t - 2$

$$\begin{cases} u_t = a(u_{xx} + u_{yy}) + 1 - a(24x - 6y), \\ (x,y) \in \Omega = (0, L_1) \times (0, L_2), t \in (0, T), \\ u|_{t=0} = 4x^3 - y^3 - 2, \\ u|_{\partial\Omega} = \begin{cases} 4x^3 + t - 2,\ y = 0 \\ -y^3 + t - 2,\ x = 0 \\ 4L_1^3 - y^3 + t - 2,\ x = L_1 \\ 4x^3 - L_2^3 + t - 2,\ y = L_2 \end{cases} \end{cases} \tag{5}$$

2.2 Локально – одномерная схема 2

1 этап

$$\begin{cases} \dfrac{\widetilde{U}^{(k+1)} - U^{(k)}}{\tau} = a\Lambda_1 \widetilde{U}^{(k+1)} + f^{(k+1)} \text{ в области}, \\ \widetilde{U}^{(k+1)} = \widetilde{g}^{(k+1)} = g^{(k+1)} - \tau a \Lambda_2 g^{(k+1)} \text{ на границе при } x = 0,\ x = L_1. \end{cases} \tag{6}$$

2 этап

$$\begin{cases} \dfrac{U^{(k+1)} - \widetilde{U}^{(k+1)}}{\tau} = a\Lambda_2 U^{(k+1)} \text{ в области}, \\ U^{(k+1)} = g^{(k+1)} \text{ на границе при } x = 0,\ x = L_1. \end{cases} \tag{7}$$

3 Практическая часть

Расчеты будут проводиться на прямоугольной области с квадратной сеткой(хотя можно и с прямоугольной, программа это предусматривает) с начальными параметрами: $a = 1$, $L_1 = 2$, $L_2 = 3$, $T = 5(L_1 + L_2) = 25$, $h_1 = 0.1$, $h_2 = 0.1$, $\tau = 1$

3.1 Тестовый пример №1

	h	$h/2$	$h/4$	$h/8$
τ	0.3	0.15	0.075	0.0375
$\tau/2$	0.3	0.15	0.075	0.0375
$\tau/4$	0.3	0.15	0.0750	0.0375
$\tau/8$	0.303207	0.15	0.0750	0.0375

3.2 Тестовый пример №2

	h	$h/2$	$h/4$	$h/8$
τ	0.470542	0.238111	0.119795	0.060076
$\tau/2$	0.607441	0.307756	0.154834	0.077656
$\tau/4$	0.707330	0.358297	0.180287	0.090426
$\tau/8$	0.758845	0.384346	0.193408	0.097008

3.3 Тестовый пример №3

	h	$h/2$	$h/4$	$h/8$
τ	1.179999	0.595	0.29875	0.149687
$\tau/2$	1.179999	0.595	0.29875	0.149687
$\tau/4$	1.179999	0.595	0.29875	0.149687
$\tau/8$	1.179999	0.595	0.29875	0.149687

3.4 Тестовый пример №4

	h	$h/2$	$h/4$	$h/8$
τ	2.610999	1.327624	0.669391	0.336096
$\tau/2$	3.205835	1.648068	0.835061	0.420286
$\tau/4$	3.988972	2.048587	1.0377	0.522204
$\tau/8$	4.394924	2.255602	1.14239	0.574823

$\boxed{\text{Вывод}}$ **:** В ходе выполнения лабораторной работы №3 была реализована разностная схема 2 – (локально - одномерная схема - 2), позволяющая решить начально-краевую задачу для двумерного уравнения теплопроводности. Можно заметить, что при более сильном разбиении сетки по времени наблюдается накапливание погрешности, а в некоторых случаях(при определенных функциях) не увеличивается вовсе. При увеличении разбиений по координатам x и y наблюдается противоположная ситуация – погрешность уменьшается, причем пропорционально изменению шага.

4 Приложение

```python
import numpy as np

# начальные данные
L_1 = 3
L_2 = 3
T = 5 * (L_1 + L_2)
N_x = 301
N_y = 201
N_t = 31
a = 1


# метод прогонки
def TDMA(a, b, c, f):
    a, b, c, f = tuple(
        map(lambda k_list: list(map(float, k_list)),
            (a, b, c, f)))

    alpha = [-b[0] / c[0]]
    beta = [f[0] / c[0]]
    n = len(f)
    x = [0] * n

    for i in range(1, n):
        alpha.append(-b[i] / (a[i] * alpha[i - 1] + c[i]))
        beta.append((f[i] - a[i] * beta[i - 1]) / (
                a[i] * alpha[i - 1] + c[i]))

    x[n - 1] = beta[n - 1]

    for i in range(n - 1, 0, -1):
        x[i - 1] = alpha[i - 1] * x[i] + beta[i - 1]

    return x


# Тестовый пример №1
def f(x, y, t):
    return 5

```

```python

def es(x, y, t):
    return 2 * x + 3 * y + 5 * t


def phi(x, y, t=0):
    return 2 * x + 3 * y + t


# тестовый пример 2

def f_2(x, y, t):
    return -4 * a - 3


def es_2(x, y, t):
    return 2 * x ** 2 + y - 3 * t + 2


def phi_2(x, y, t=0):
    return 2 * x ** 2 + y - 3 * t + 2


# тестовый пример 3
def f_3(x, y, t):
    return 1 - 2 * a


def es_3(x, y, t):
    return 3 * x ** 2 - 2 * y ** 2 + t + 4


def phi_3(x, y, t=0):
    return 3 * x ** 2 - 2 * y ** 2 + t + 4


# тестовый пример 4

def f_4(x, y, t):
    return 1 - a * (24 * x - 6 * y)


def es_4(x, y, t):
```

```
    return 4*x**3 - y**3 + t -2


def phi_4(x, y, t=0):
    return 4*x**3 - y**3 + t -2


def LODS_2(function_f, function_g, function_phi):
    # шаг по сетке <<x>>
    h_x = L_1 / (N_x - 1)
    # шаг по сетке <<y>>
    h_y = L_2 / (N_y - 1)
    # шаг по сетке <<t>>
    tau = T / (N_t - 1)

    # полученное решение
    solution = np.zeros((N_t, N_x, N_y))

    # Определим сетки
    x_grid = np.linspace(0, L_1, N_x)
    y_grid = np.linspace(0, L_2, N_y)
    tau_grid = np.linspace(0, T, N_t)

    # заполним <<нулевой слой>>
    for i in range(0, N_x):
        for j in range(0, N_y):
            solution[0][i][j] = function_phi(x_grid[i],
                                             y_grid[j])

    def UTilda(num_k):
        # Решим систему для первого этапа
        # главная диагональ
        md = np.full(N_x - 2, -1 * (h_x ** 2) - 2 * a * tau)
        # наддиагональ
        ud = np.full(N_x - 2, a * tau)
        # поддиагональ
        dd = np.full(N_x - 2, a * tau)
        dd[0] = 0
        ud[N_x - 3] = 0

        # Вспомогательный слой
        U_tilda = np.zeros((N_x, N_y))

```

```python
        # Краевые точки
        U_tilda_0 = 0
        U_tilda_N = 0

        for k in range(1, N_y - 1):
            # вектор правой части
            b = np.zeros(N_x - 2)
            # заполним вектор правой части
            for l in range(0, N_x - 2):
                if l == 0:
                    b[l] = -1 * tau * (h_x ** 2) * function_f(x_grid[l + 1], y_grid[k], tau_grid[num_k + 1]) - 1 * (
                            h_x ** 2) * solution[num_k][l + 1][k] + ((a * tau / h_y) ** 2) * (
                                   function_g(x_grid[l], y_grid[k - 1], tau_grid[num_k + 1]) -
                                   (2 + (h_y ** 2 / (a * tau))) * function_g(x_grid[l], y_grid[k],

                                   function_g(x_grid[l], y_grid[k + 1], tau_grid[num_k + 1]))
                    U_tilda_0 = (-1 * a * tau / (h_y ** 2)) * (
                            function_g(x_grid[l], y_grid[k - 1], tau_grid[num_k + 1]) -
                            (2 + (h_y ** 2 / (a * tau))) * function_g(x_grid[l], y_grid[k], tau_grid[num_k + 1]) +
                            function_g(x_grid[l], y_grid[k + 1], tau_grid[num_k + 1]))
                elif l == N_x - 3:
                    b[l] = -1 * tau * (h_x ** 2) * function_f(x_grid[l], y_grid[k], tau_grid[num_k + 1]) - (h_x ** 2) * \
```

```
                                        solution[num_k][l][k] + ((a * tau / h_y) ** 2) * (
                                                function_g(x_grid[l + 1], y_grid[k - 1], tau_grid[num_k + 1]) -
                                                (2 + (h_y ** 2 / (a * tau))) * function_g(x_grid[l + 1], y_grid[k],

                                                function_g(x_grid[l + 1], y_grid[k + 1], tau_grid[num_k + 1]))
                                U_tilda_N = (-1 * a * tau / (h_y ** 2)) * (
                                        function_g(x_grid[l + 1], y_grid[k - 1], tau_grid[num_k + 1]) -
                                        (2 + (h_y ** 2 / (a * tau))) * function_g(x_grid[l + 1], y_grid[k], tau_grid[num_k + 1]) +
                                        function_g(x_grid[l + 1], y_grid[k + 1], tau_grid[num_k + 1]))
                            else:
                                b[l] = -1 * tau * (h_x ** 2) * function_f(x_grid[l + 1], y_grid[k], tau_grid[num_k + 1]) - (
                                        h_x ** 2) * \
                                       solution[num_k][l + 1][k]
                        # решим методом прогонки СЛАУ
                        res = TDMA(dd, ud, md, b)
                        # заносим решение во вспомогательный слой
                        U_tilda[0][k] = U_tilda_0
                        U_tilda[N_x - 1][k] = U_tilda_N
                        for i in range(1, N_x - 1):
                            U_tilda[i][k] = res[i - 1]
                    return U_tilda

                def gtes(num_k):
                    # Решим систему для второго этапа
                    # главная диагональ
```

```
        md = np.full(N_y - 2, -1 * (2 * a * tau / (h_y ** 2) + 1))
        # наддиагональ
        ud = np.full(N_y - 2, a * tau / (h_y ** 2))
        # поддиагональ
        dd = np.full(N_y - 2, a * tau / (h_y ** 2))
        dd[0] = 0
        ud[N_y - 3] = 0

        # приближенное решение
        U = np.zeros((N_x, N_y))
        U_0 = 0
        U_N = 0

        # получим вспомогательный слой
        U_support = UTilda(num_k - 1)

        for k in range(0, N_x):
            # вектор правой части
            b = np.zeros(N_y - 2)
            # заполним вектор правой части
            for l in range(0, N_y - 2):
                if l == 0:
                    b[l] = -1 * U_support[k][l + 1] - (a * tau / (h_y ** 2)) * function_g(x_grid[k], y_grid[l],

                    U_0 = function_g(x_grid[k], y_grid[l], tau_grid[num_k])
                elif l == N_y - 3:
                    b[l] = -1 * U_support[k][l] - (a * tau / (h_y ** 2)) * function_g(x_grid[k], y_grid[l + 1],

                    U_N = function_g(x_grid[k], y_grid[l + 1], tau_grid[num_k])
                else:
                    b[l] = -1 * U_support[k][l + 1]
            # решим методом прогонки СЛАУ
            res = TDMA(dd, ud, md, b)
            # заносим решение
```

```
                U[k][0] = U_0
                U[k][N_y - 1] = U_N
                for i in range(1, N_y - 1):
                    U[k][i] = res[i - 1]
            return U

        for d in range(1, N_t):
            solution[d] = gtes(d)
        return solution


    # получение тензора точного решения
    def the_exact_solution(UFunc):
        solution = np.zeros((N_t, N_x, N_y))
        x_grid = np.linspace(0, L_1, N_x)
        y_grid = np.linspace(0, L_2, N_y)
        tau_grid = np.linspace(0, T, N_t)
        for depth in range(0, N_t):
            for x_axis in range(0, N_x):
                for y_axis in range(0, N_y):
                    solution[depth][x_axis][y_axis] =
                     ↪ UFunc(x_grid[x_axis], y_grid[y_axis],
                     ↪ tau_grid[depth])
        return solution


    # вычисление погрешности
    def error(Ux, Ua):
        return np.max(np.abs(Ux - Ua))


    # вычисление приближенного решения
    result = LODS_2(f, es, phi)
    # вычисление точного решения
    exact_result = the_exact_solution(es)

    print("Шаг по t -- ", T / (N_t - 1))
    print("Шаг по x -- ", L_1 / (N_x - 1))
    print("Шаг по y -- ", L_2 / (N_y - 1))
    print("Ошибка -- ", error(result, exact_result))
```