МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
учреждение высшего образования
"Национальный исследовательский университет "МЭИ"

Отчёт по курсовой работе

Численные методы

Тема:

"Расчёт температурного поля пластины в зависимости от положения источника"

Студент: Кутдусов Р.К.
Группа: А-13а-19
Преподаватель: Амосова О.А.

Москва
2021

Содержание

1 Постановка задачи

Прямоугольная металлическая пластина с вырезом используется как теплоотводящий элемент. В угловом вырезе пластины (границы Γ_2 и Γ_3) расположен источник тепла. Распределение температуры $T(x, y)$ по площади пластины описывается уравнением Лапласа:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1}$$

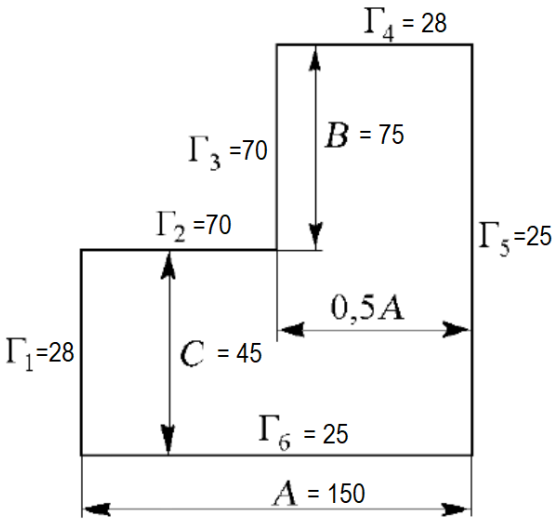

Рис. 1: Конфигурация пластины

Требуется найти распределение $T(x, y)$.

2 Построение разностной аппроксимации и тестовых примеров

2.1 Построение разностной аппроксимации

Имеем задачу вида:

$$\Delta u(x, y) = -f(x, y) \tag{2}$$

$$u|_\Gamma = \varphi(x, y) \tag{3}$$

Здесь $\Delta u = u_{xx} + u_{yy}$ – оператор Лапласа, уравнение (2) – *уравнение Пуассона* относительно неизвестной функции $u(x, y)$, функция $f(x, y)$ известна, (3) – граничные условия первого рода. В нашем случае $f(x, y) \equiv 0$ и уравнение (2) называется *уравнением Лапласа*. Задача (2) , (3) называется *задачей Дирихле* для уравнения Пуассона (Лапласа).

Для численного решения поставленной задачи Дирихле воспользуемся *методом конечных разностей*. Для аппроксимации уравнения (2) возьмём пятиточечный шаблон.

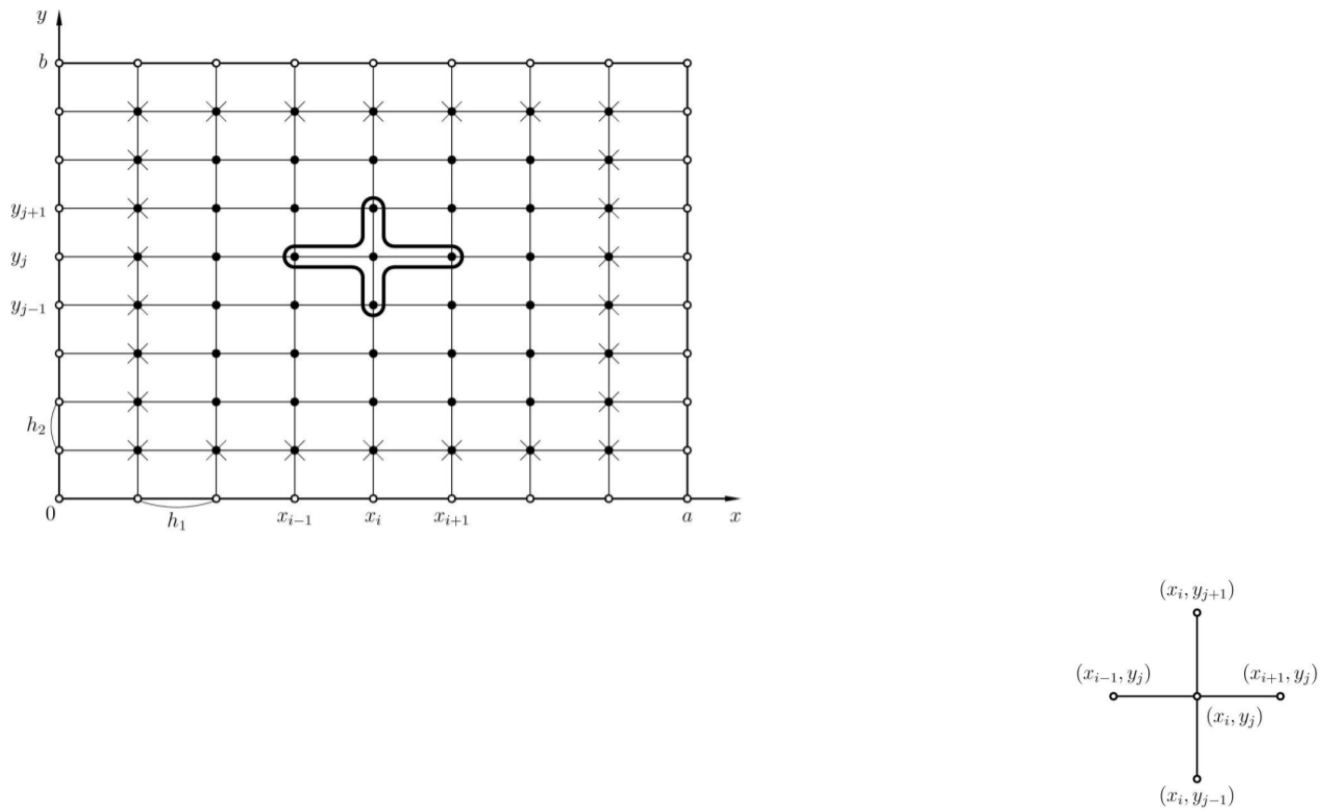

Рис. 2: Двумерная пространственная сетка и пятиточечный шаблон

По каждой из пространственных переменных введём операторы второй разностной производной

$$\Lambda_1[U](x_i, y_j) = \frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h_1^2}$$

для $i = 1, \ldots, N_1 - 1$ и $j = 0, \ldots, N_2$ и

$$\Lambda_2[U](x_i, y_j) = \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h_2^2}$$

для $i = 0, \ldots, N_1$ и $j = 1, \ldots, N_2 - 1$.

В каждой внутренней точке пространственной сетки построим разностное уравнение

$$\Lambda_1[U] + \Lambda_2[U] = -F. \tag{4}$$

$F_{i,j} = f(x_i, y_i)$ – аппроксимация правой части уравнения (2) в случае непрерывности функции f. В развёрнутом виде уравнение (4) имеет вид

$$\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h_1^2} + \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h_2^2} = -F_{i,j}.$$

Поскольку граничные условия первого рода, то они аппроксимируются без погрешностей. В случае прямоугольной области граничные условия можно записать в виде

$$U_{i,0} = \varphi(x_i), \quad U_{i,N_2} = \varphi_2(x_i), \quad i = 0, 1, \ldots, N_1,$$

$$U_{0,j} = \chi_1(y_j), \quad U_{N_1,j} = \chi_2(y_j), \quad j = 0, 1, \ldots, N_2.$$

Построенная дискретная задача называется *пятиточечной разностной схемой* для задачи Дирихле.

2.2 Построение первого тестового примера

Необходимо строить тестовые примеры, для того чтобы можно было оценить корректность реализации методов решения задачи. Тестовые примеры будем строить по известному точному решению. То есть зафиксируем некоторую функцию $u(x, y)$ и границы прямоугольной области a, b. Получим явным образом функцию источника $f(x, y)$ и граничные условия. Если точное решение и приближенное с приемлемой точностью близки, то можно считать, что метод и программа работают корректно.

В качестве первого тестового примера возьмём задачу, в которой решение будет представлять собой собственную функцию задачи Штурма-Лиувилля

$$u(x, y) = \frac{8}{\pi^2} \cdot \sin\frac{\pi x}{4} \sin\frac{\pi y}{4}.$$

Найдём правую часть уравнения (2):

$$\begin{aligned}
u'_x &= \frac{8}{\pi^2} \cdot \frac{\pi}{4} \cos\frac{\pi x}{4} \sin\frac{\pi y}{4} \\
u''_{xx} &= -\frac{8}{\pi^2} \cdot \frac{\pi^2}{16} \sin\frac{\pi x}{4} \sin\frac{\pi y}{4} \\
u'_y &= \frac{8}{\pi^2} \cdot \frac{\pi}{4} \sin\frac{\pi x}{4} \cos\frac{\pi y}{4} \\
u''_{yy} &= -\frac{8}{\pi^2} \cdot \frac{\pi^2}{16} \sin\frac{\pi x}{4} \sin\frac{\pi y}{4} \\
f(x, y) &= -\Delta u(x, y) = \sin\frac{\pi x}{4} \sin\frac{\pi y}{4}.
\end{aligned}$$

Пусть наша область будет представлять собой квадрат 8×8 с вырезом 4×4. Тогда граничные условия

$$\begin{aligned}
u(0, y) &= 0, \quad 0 \le y < 4 \\
u(x, 4) &= 0, \quad 0 \le x < 4 \\
u(4, y) &= 0, \quad 4 \le y < 8 \\
u(x, 8) &= 0, \quad 4 \le x < 8 \\
u(8, y) &= 0, \quad 0 < y \le 8 \\
u(x, 0) &= 0, \quad 0 < x \le 8
\end{aligned}$$

И задача принимает вид:

$$\begin{cases}
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -\sin\frac{\pi x}{4} \sin\frac{\pi y}{4}, & 0 < x < 8, \quad 0 < y < 8 \\
u(0, y) = 0, & 0 \le y < 4 \\
u(x, 4) = 0, & 0 \le x < 4 \\
u(4, y) = 0, & 4 \le y < 8 \\
u(x, 8) = 0, & 4 \le x < 8 \\
u(8, y) = 0, & 0 < y \le 8 \\
u(x, 0) = 0, & 0 < x \le 8
\end{cases} \tag{5}$$

2.3 Построение второго тестового примера

В качестве второго тестового примера возьмём задачу, в которой решение будет иметь вид

$$u(x, y) = e^{-x} \cdot \sin(\pi y)$$

Найдём правую часть уравнения (2):

$$
\begin{aligned}
u'_x &= -e^{-x} \cdot \sin(\pi y) \\
u''_{xx} &= e^{-x} \cdot \sin(\pi y) \\
u'_y &= \pi \cdot e^{-x} \cdot \cos(\pi y) \\
u''_{yy} &= -\pi^2 \cdot e^{-x} \cdot \sin(\pi y) \\
f(x,y) &= -\Delta u(x,y) = e^{-x} \cdot \sin(\pi y)(\pi^2 - 1)
\end{aligned}
$$

Наша область будет представлять собой квадрат 2×2 с вырезом 1×1. Тогда граничные условия

$$
\begin{aligned}
&u(0,y) = \sin(\pi y), && 0 \le y < 1 \\
&u(x,1) = 0, && 0 \le x < 1 \\
&u(1,y) = e^{-1} \cdot \sin(\pi y), && 1 \le y < 2 \\
&u(x,2) = 0, && 1 \le x < 2 \\
&u(2,y) = e^{-2} \cdot \sin(\pi y), && 0 < y \le 2 \\
&u(x,0) = 0, && 0 < x \le 2
\end{aligned}
$$

Задача принимает вид:

$$
\begin{cases}
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = e^{-x} \cdot \sin(\pi y)(1 - \pi^2), & 0 < x < 2, \quad 0 < y < 2 \\
u(0,y) = \sin(\pi y), & 0 \le y < 1 \\
u(x,1) = 0, & 0 \le x < 1 \\
u(1,y) = e^{-1} \cdot \sin(\pi y), & 1 \le y < 2 \\
u(x,2) = 0, & 1 \le x < 2 \\
u(2,y) = e^{-2} \cdot \sin(\pi y), & 0 < y \le 2 \\
u(x,0) = 0, & 0 < x \le 2
\end{cases} \tag{6}
$$

3 Итерационные методы решения задачи Дирихле

Как следует из развёрнутой записи уравнения (4), при больших N_1 и N_2 матрица получающейся системы линейных алгебраических уравнений является сильно разреженной. Поэтому следует использовать итерационные методы.

Для решения задачи будем использовать метод Зейделя. Так как метод является неявным, то целесообразно нумеровать точки сетки так, чтобы матрица системы сеточных уравнений имела наиболее простой вид. Одним из таких способов является нумерация точек по рядам, начиная с выбранной угловой приграничной точки. Если за начало принять точку (x_1, y_1) и занумеровать, например так, как схематично изображено в виде матрицы на рис. 3 (по горизонтальным рядам), то двумерная задача может рассматриваться как одномерная.

```
[[ 0.   0.   0.   0.   0.  70.  28.  28.  28.  28.  26.5]
 [ 0.   0.   0.   0.   0.  70.  35.  36.  37.  38.  25. ]
 [ 0.   0.   0.   0.   0.  70.  31.  32.  33.  34.  25. ]
 [ 0.   0.   0.   0.   0.  70.  27.  28.  29.  30.  25. ]
 [ 0.   0.   0.   0.   0.  70.  23.  24.  25.  26.  25. ]
 [70.  70.  70.  70.  70.  70.  19.  20.  21.  22.  25. ]
 [28.  10.  11.  12.  13.  14.  15.  16.  17.  18.  25. ]
 [28.   1.   2.   3.   4.   5.   6.   7.   8.   9.  25. ]
 [26.5 25.  25.  25.  25.  25.  25.  25.  25.  25.  25. ]]
```

Рис. 3: Нумерация узлов сетки

Для построения метода Зейделя примем за основу нумерацию, приведённую на рис. 3. Тогда при вычислении очередного приближения $U_{i,j}$ значения с индексами $i+1$ и $j-1$ будут известными (уже вычислены или являются граничными условиями), а значения с индексами $i-1$ и $j+1$ берём с предыдущей итерации. Поэтому имеем следующую запись метода Зейделя:

$$\frac{U_{i+1,j}^{(k+1)} - 2U_{i,j}^{(k+1)} + U_{i-1,j}^{(k)}}{h_1^2} + \frac{U_{i,j-1}^{(k+1)} - 2U_{i,j}^{(k+1)} + U_{i,j+1}^{(k)}}{h_2^2} = -F_{i,j}$$

Примем $h_1 = h_2 = h$. Тогда расчётная формула для программирования будет иметь вид:

$$U_{i,j}^{(k+1)} = \frac{1}{4} \cdot (F_{i,j}h^2 + U_{i+1,j}^{(k+1)} + U_{i,j-1}^{(k+1)} + U_{i-1,j}^{(k)} + U_{i,j+1}^{(k)})$$

В качестве критерия окончания будет использована формула

$$\|U^{(k+1)} - U^{(k)}\|_1 \leq \varepsilon$$

Первая норма матрицы A вычисляется следующим образом

$$\|A\|_1 = \max_{0 \leq j < n} \sum_{i=0}^{n-1} |a_{i,j}|$$

Циклы будут организованы таким образом, что вырез пластины будет пропускаться. Код реализации представлен в приложении.

3.1 Решение первого тестового примера

Представлен график численного решения первого тестового примера (5), $h = 0.5$. С помощью метода Зейделя решение с $\varepsilon = 10^{-2}$ найдено за 34 итерации.

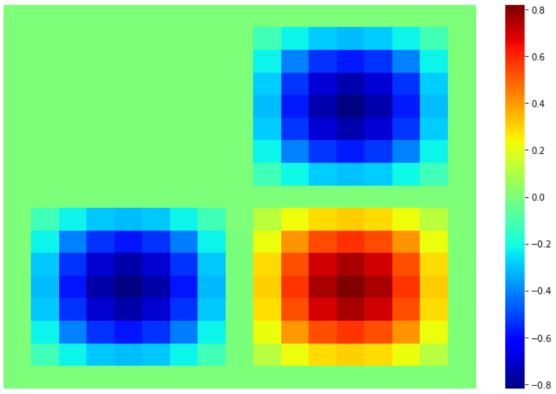

Рис. 4: Температурная карта численного решения первого тестового примера

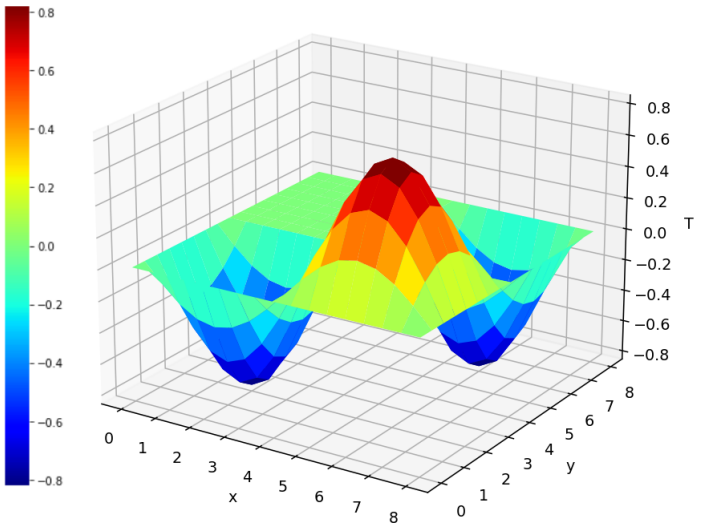

Рис. 5: Трёхмерный график численного решения первого тестового примера

Представлен график точного решения по тем же узлам, что и приближённое.

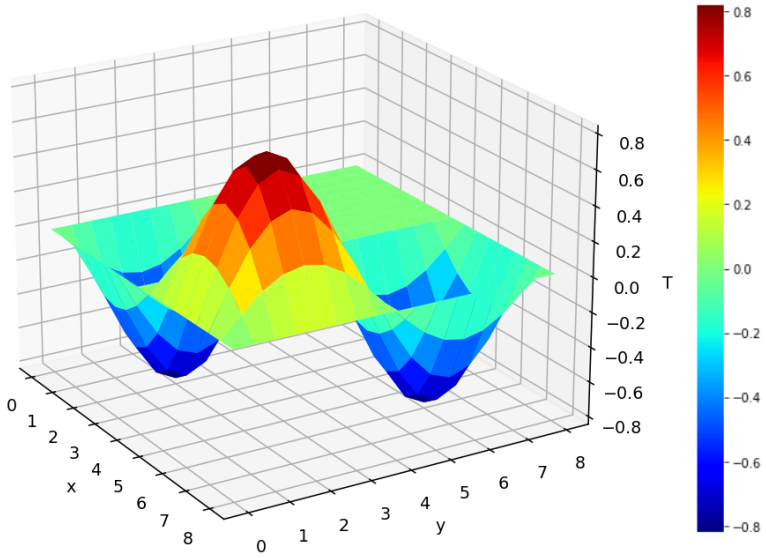

Рис. 6: Трёхмерный график точного решения первого тестового примера

Визуально решения совпадают. Построим график погрешности приближённого решения:

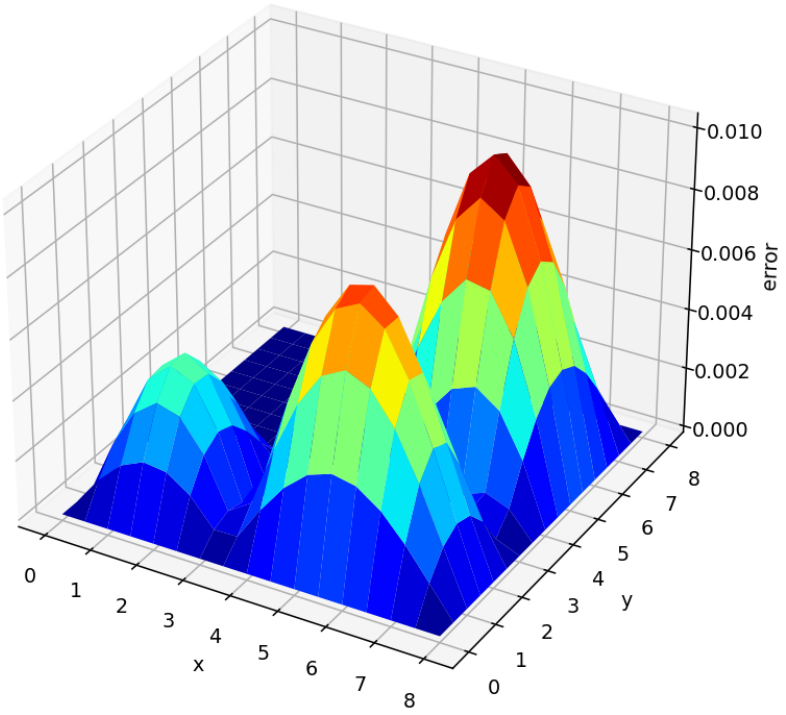

Рис. 7: Трёхмерный график погрешности численного решения первого тестового примера

Оба решения с приемлемой точностью близки, поэтому можно говорить о корректности разностной схемы и реализации метода Зейделя.

3.2 Решение второго тестового примера

Представлен график приближённого решения второго тестового примера (6), $h = 0.1$. С помощью метода Зейделя решение с $\varepsilon = 10^{-2}$ найдено за 49 итераций.

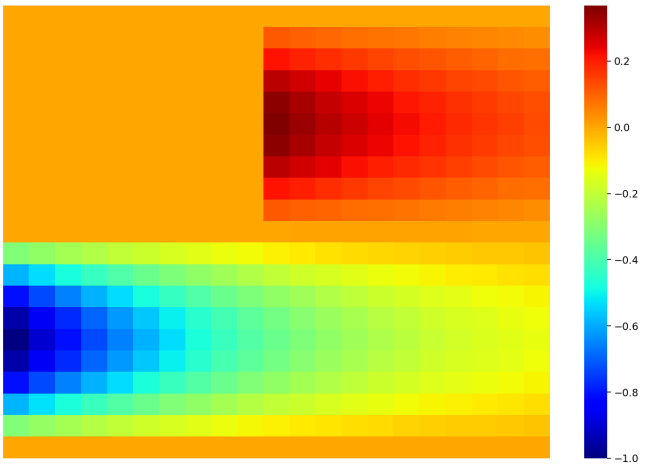

Рис. 8: Температурная карта приближённого решения второго тестового примера

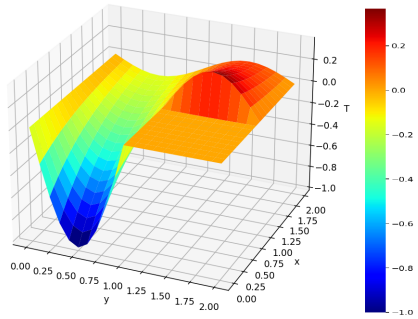

Рис. 9: Трёхмерный график приближённого решения второго тестового примера

Представлен график точного решения.

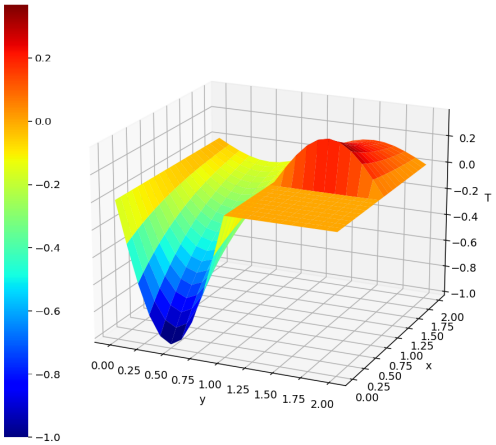

Рис. 10: Трёхмерный график точного решения второго тестового примера

Построим график погрешности приближённого решения:

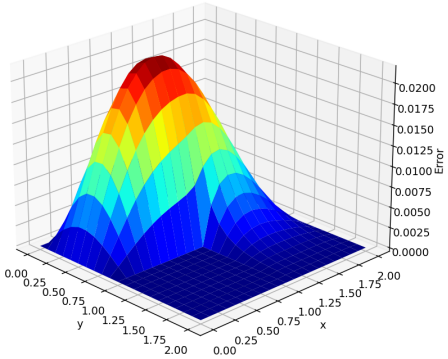

Рис. 11: Трёхмерный график погрешности приближённого решения второго тестового примера

Можно сделать вывод, аналогичный выводу из пункта 3.1.

3.3 Решение исходной задачи

Решать поставленную задачу методом Зейделя будем аналогичным образом, что и тестовые примеры. Находить погрешность будем по правилу Рунге.

Представлен график приближённого решения задачи из условия (1), $h = 1$. С помощью метода Зейделя решение с $\varepsilon = 10^{-1}$ найдено за 3697 итераций.

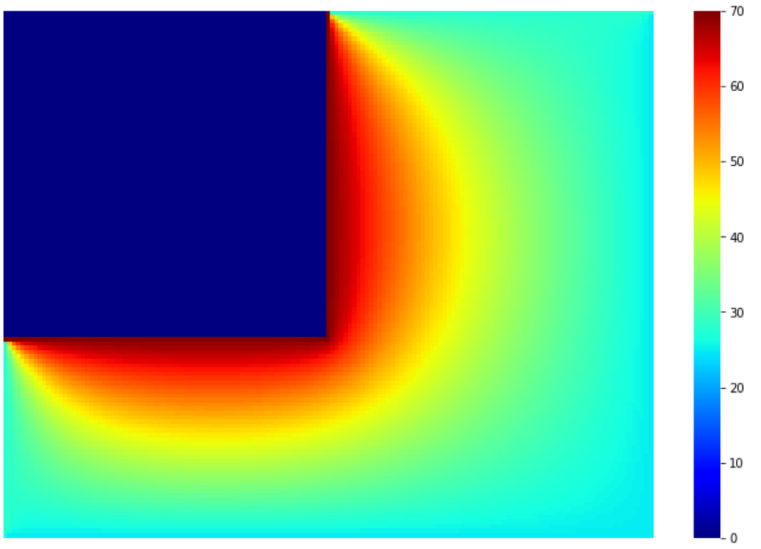

Рис. 12: Температурная карта численного решения задачи

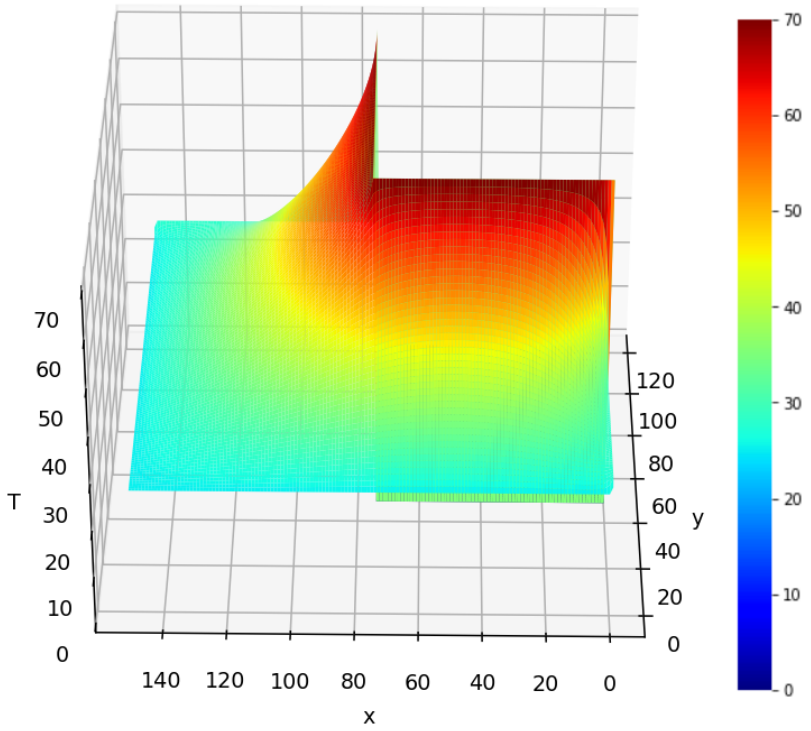

Рис. 13: Трёхмерный график приближённого решения задачи

Хорошо виден максимум решения на границе, на которой расположен источника тепла. Видно убывание температуры от этих границ к другим границам области.

Расположим источник тепла на границах Γ_5, Γ_6 и посмотрим на получившееся решение при таких условиях.

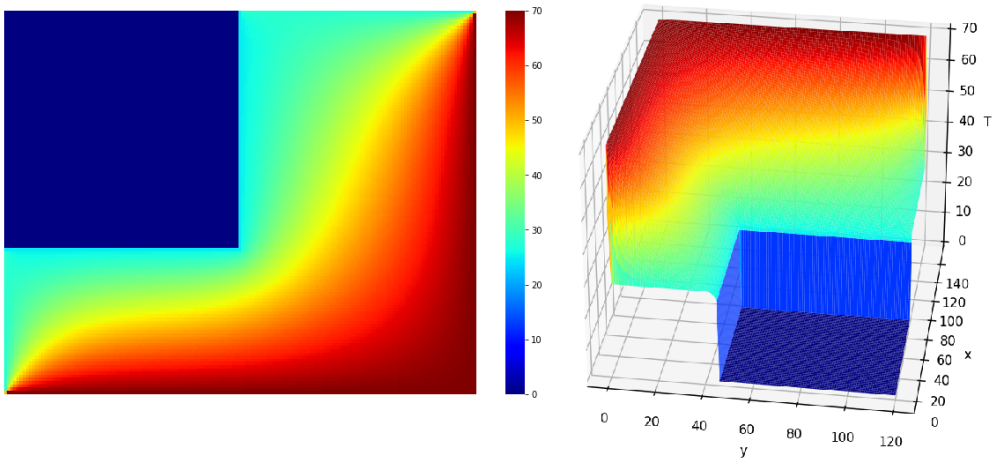

Рис. 14: Численное решение задачи с изменёнными граничными условиями

Снова виден максимум на границах, на которых расположен источник, и убывание температуры по мере удаления от этих границ.

Представлен трёхмерный график погрешности приближённого решения исходной задачи. Оценка погрешности произведена на основе правила Рунге $\varepsilon_{i,j}^h = \frac{|u_{i,j}^h - u_{i,j}^{h/2}|}{2^p - 1}, \quad p = 2.$

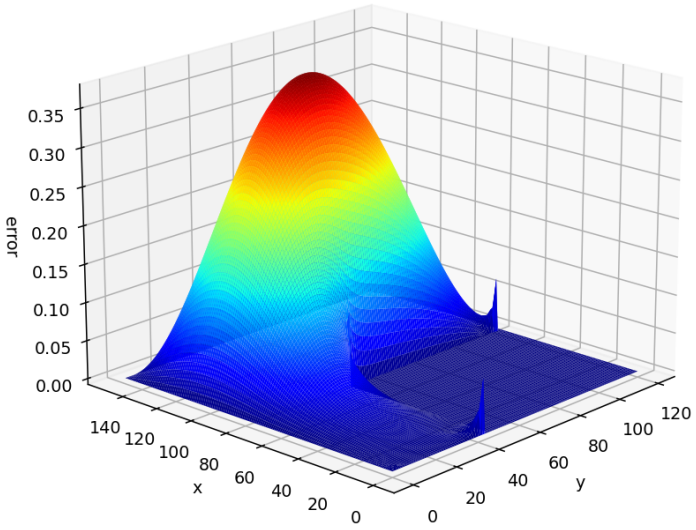

Рис. 15: Трёхмерный график погрешности приближённого решения задачи

Время нахождения решения данной задачи с точностью 0.35 — около 5 минут. Это очень длительный промежуток времени и хочется найти решение гораздо быстрее. Для этого будем пользоваться более быстрыми проекционными методами численного решения сеточного уравнения Лапласа.

4 Проекционные методы решения задачи Дирихле

Мы решаем СЛАУ $Ax = b$, причём, исходя из схемы, $A \in \mathbb{R}^{n \times n}$, $A = A^T > 0$, A – сильно разрежена. Поэтому для того чтобы ускорить нахождение решения исходной задачи можем применять метод сопряжённых градиентов (CG). Код реализации метода представлен с 100 строки в приложении.

Метод удобен тем, что нет необходимости явно формировать матрицу системы A. Для того, чтобы реализовать метод, достаточно уметь вычислять произведение матрицы системы на произвольный массив, пользуясь формулами из левой части системы

$$-\left(\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h_1^2} + \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h_2^2}\right) = F_{i,j}.$$

В случае однородных граничных условий всё довольно просто: в каждом внутреннем узле значение координаты вектора b вычисляется по правой части приведенной системы, значение координаты вектора Ax – по левой части.

В случае неоднородных граничных условий, ненулевые граничные условия переносятся в правые части уравнений, таким образом изменяется вычисление координат вектора b. Уравнения системы, соответствующие всем приграничным узлам слева и справа, имеют вид

$$\begin{aligned}
-\left(\frac{-2U_{1,j} + U_{2,j}}{h_1^2} + \frac{U_{1,j-1} - 2U_{1,j} + U_{1,j+1}}{h_2^2}\right) &= F_{1,j} + \frac{g(0, y_j)}{h_1^2}, \\
-\left(\frac{U_{N_1-2,j} - 2U_{N_1-1,j}}{h_1^2} + \frac{U_{N_1-1,j-1} - 2U_{N_1-1,j} + U_{N_1-1,j+1}}{h_2^2}\right) &= F_{N_1-1,j} + \frac{g(X, y_j)}{h_1^2}, \\
j &= 2, \ldots, N_2 - 2.
\end{aligned}$$

Уравнения системы, соответствующие всем приграничным узлам снизу и сверху, имеют вид

$$\begin{aligned}
-\left(\frac{U_{i-1,1} - 2U_{i,1} + U_{i+1,1}}{h_1^2} + \frac{-2U_{i,1} + U_{i,2}}{h_2^2}\right) &= F_{i,1} + \frac{g(x_i, 0)}{h_2^2}, \\
-\left(\frac{U_{i-1,N_2-1} - 2U_{i,N_2-1} + U_{i+1,N_2-1}}{h_1^2} + \frac{U_{i,N_2-2} - 2U_{i,N_2-1}}{h_2^2}\right) &= F_{i,N_2-1} + \frac{g(x_i, Y)}{h_2^2}, \\
i &= 2, \ldots, N_1 - 2.
\end{aligned}$$

В узле сетки (x_1, y_1) уравнение принимает вид

$$-\left(\frac{-2U_{1,1} + U_{2,1}}{h_1^2} + \frac{-2U_{1,1} + U_{1,2}}{h_2^2}\right) = F_{1,1} + \frac{g(0, y_1)}{h_1^2} + \frac{g(x_1, 0)}{h_2^2}.$$

В узлах сетки $(x_1, y_{N_2-1}), (x_{N_1-1}, y_1), (x_{N_1-1}, y_{N_2-1})$ уравнения меняются аналогичным образом.

После изменения правой части системы, её можно решать как систему для однородной задачи.

Скалярное произведение матриц есть сумма произведений соответствующих координат:

$$(U, W) = \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} U_{i,j} V_{i,j}$$

Приведём алгоритм метода сопряжённых градиентов:

Метод сопряженных градиентов (CG)

$r_0 := b - Ax_0; \quad p_0 := r_0$
for $j = 0, 1, 2, \ldots$ **do**
  $\alpha := \frac{(r_j, r_j)}{(Ap_j, p_j)}$
  $x_{j+1} := x_j + \alpha p_j$
  $r_{j+1} := r_j - \alpha A p_j$
  $\beta := \frac{(r_{j+1}, r_{j+1})}{(r_j, r_j)}$
  $p_{j+1} := r_{j+1} + \beta p_j$
  if достигнута заданная точность **then**
    выйти из цикла
  end if
end for

В качестве критерия окончания будет использована формула

$$\|U^{(k+1)} - U^{(k)}\|_1 \leq \varepsilon$$

В качестве начального приближения U_0 во всех случаях, как и при решении итерационными методами, возьмём нулевой вектор.

Решения методом сопряженных градиентов получены и они совпадают с теми решениями, которые получены методом Зейделя. Приведём сводную таблицу результатов:

Задача	h	ε	Метод Зейделя	Метод CG
(5)	0.5	10^{-2}	34	2
(6)	0.1	10^{-2}	49	26
(1)	1	10^{-1}	3697	213

Видно, что применение метода CG позволило ускорить нахождение решения для исходной задачи в 17 раз (по количеству итераций). Как известно, метод CG обладает сверхлинейной скоростью сходимости.

5 Заключение

Мы рассмотрели некоторые методы для решения задачи Дирихле в области сложной формы. В целом, для таких задач очень хорошо подходит итерационный метод Зейделя. Однако, когда требуется получить четкий портрет решения, мы уменьшаем шаг и сталкиваемся с тем, что метод Зейделя начинает сходится очень медленно. В таком случае для ускорения нахождения решения применяют другие методы — проекционные. Один из самых знаменитых проекционных методов — метод сопряженных градиентов. Он отлично решает нашу задачу, намного быстрее, чем это делает итерационный метод. Одна итерация проекционного метода по вычислительной трудоёмкости приблизительно сравнима с итерацией по итерационному методу, что даёт явное преимущество при выборе метода для решения сложной задачи, в которой возникает матрица системы большой размерности.

6 Список литературы

1. Амосов, А. А. Вычислительные методы: учебное пособие / А. А. Амосов, Ю. А. Дубинский, Н. В. Копченова. — С-Пб.: Издательство «Лань», 2012. — 672с.

2. Казёнкин, К.О. Численное решение задач математической физики. Стационарные уравнения (уравнение Пуассона) : учебно-методическое пособие / К.О. Казёнкин, О.А. Амосова. — М.: Издательство МЭИ, 2017. — 36с.

3. Казёнкин, К.О. Численное решение стационарных уравнений математической физики: методические указания / К.О. Казёнкин, А.Е. Вестфальский, О.А. Амосова. — М.: Издательство МЭИ, 2018. — 48с.

4. Баландин, М. Ю. Методы решения СЛАУ боллшой размерности / М. Ю. Баландин, Э. П. Шурина. — Новосибирск: Издательство НГТУ, 2000. — 65с.

5. Самарский, А. А. Численные методы: учебное пособие / А. А. Самарский, А. В. Гулин. — М.: Наука, 1989. — 432с.

```
1.  import numpy as np
2.  from matplotlib import pyplot as plt
3.  import seaborn as sns
4.
5.  # Метод Зейделя
6.  # 1-ый тестовый пример
7.
8.  A = 8
9.  B = 4
10. C = 4
11.
12. def F(x, y): # правая часть
13.     return np.sin(np.pi * x * 0.25) * np.sin(np.pi * y * 0.25)
14.
15. def seidel(eps = 10 ** (-2), h = 0.5):
16.     vert_1 = int(C / h - 1.0)
17.     vert_2 = int(B / h)
18.     start_1 = 1
19.     start_2 = int(0.5 * (A / h) + 1)
20.     T = np.zeros((int((B + C) / h + 1), int(A // h + 1)))
21.     T_ = np.copy(T)  # T_ - матрица-состояние на предыдущей итерации
22.     it = 1
23.     while True:
24.         for i in reversed(range(T_.shape[0] - 1 - vert_1, T_.shape[0] - 1)):
25.             for j in range(start_1, T_.shape[1] - 1):
26.                 T[i][j] = 0.25 * (h ** 2 * F(j * h, (B + C - i) * h) + T[i][j - 1] + T[i + 1][j] +
    T_[i][j + 1] + T_[i - 1][j])
27.
28.         for i in reversed(range(1, vert_2 + 1)):
29.             for j in range(start_2, T_.shape[1] - 1):
30.                 T[i][j] = 0.25 * (h ** 2 * F(j * h, (B + C - i) * h) + T[i][j - 1] + T[i + 1][j] +
    T_[i][j + 1] + T_[i - 1][j])
31.
32.         if  np.linalg.norm(np.subtract(T, T_), ord = 1) < eps:
33.             break
34.         else:
35.             it += 1
36.             T_ = np.copy(T)
37.
38.     print('Количество итераций ', it)
39.
40.     return T
41.
42. # 2-ой тестовый пример
43. # здесь матрица T инициализируется не нулевой матрицей, а матрицей-конфигурацией пластины
44.
45. A = 2
46. B = 1
47. C = 1
48.
49. def F(x, y): # правая часть
50.     return np.exp(-x) * np.sin(np.pi * y) * (np.pi ** 2 - 1)
51.
52. def g(y): # граничные условия
53.     return np.sin(np.pi * y)
54.
55. def initial(h = 0.1):
56.     size_1 = int((B + C) / h + 1)
57.     size_2 = int(A / h + 1)
58.     mtx = np.zeros((size_1, size_2))
59.     for i in reversed(range(int(B / h + 1), size_1 - 1)):
60.         mtx[i][0] = g(i * h)
61.     for j in range(1, int((A / 2) / h + 1)):
62.         mtx[int(B / h)][j] = 0
63.     for i in reversed(range(1, int(B / h))):
64.         mtx[i][int((A / 2) / h)] = g(i * h) / np.exp(1)
```

```
    for j in range(int((A / 2) / h + 1), size_2 - 1):
        mtx[0][j] = 0
    for i in range(1, size_1 - 1):
        mtx[i][size_2 - 1] = g(i * h) / np.exp(2)
    for j in range(1, size_2 - 1):
        mtx[size_1 -1][j] = 0
    return mtx

# исходная задача
# изменение в функции initial, отсутствует F(x, y)

A = 150
B = 75
C = 45
Gamma_1 = Gamma_4 = 28 # граничные условия
Gamma_2 = Gamma_3 = 70
Gamma_5 = Gamma_6 = 25

def initial(h = 1):
    size_1 = int((B + C) / h + 1)
    size_2 = int(A / h + 1)
    mtx = np.zeros((size_1, size_2))
    mtx[size_1 - 2 : int(B / h) : -1, 0] = Gamma_1
    mtx[int(B / h), : int((A / 2) / h + 1)] = Gamma_2
    mtx[int(B / h - 1) : : -1, int((A / 2) / h)] = Gamma_3
    mtx[0, int((A / 2) / h + 1) : size_2 - 1] = Gamma_4
    mtx[1 : size_1, size_2 - 1] = Gamma_5
    mtx[size_1 - 1, 1 : size_2 - 1] = Gamma_6
    mtx[size_1 - 1, 0] = mtx[0, size_2 - 1] = (Gamma_1 + Gamma_6) / 2
    return mtx

# Метод CG
# 1-ый тестовый пример

A = 8
B = 4
C = 4
h = 0.5

vert_1 = int(C / h - 1.) # сдвиги
vert_2 = int(B / h)
start_1 = 1
start_2 = int(0.5 * (A / h) + 1.)

def F(x, y):
    return np.sin(np.pi * x * 0.25) * np.sin(np.pi * y * 0.25)

def A_(P): # матрица системы, умноженная на матрицу-вектор P (A*P)
    P_ = np.zeros((int((B + C) / h + 1), int(A / h + 1)))
    for i in reversed(range(P.shape[0] - 1 - vert_1, P.shape[0] - 1)):
        for j in range(start_1, P.shape[1] - 1):
            P_[i][j] = (-1) * (h ** (-2)) * (P[i - 1][j] + P[i + 1][j] + P[i][j + 1] + P[i][j - 1]
    - 4 * P[i][j])

    for i in reversed(range(1, vert_2 + 1)):
        for j in range(start_2, P.shape[1] - 1):
            P_[i][j] = (-1) * (h ** (-2)) * (P[i - 1][j] + P[i + 1][j] + P[i][j + 1] + P[i][j - 1]
    - 4 * P[i][j])

    return P_

def b(): # вектор b
    mtx = np.zeros((int((B + C) / h + 1), int(A / h + 1)))
    for i in reversed(range(mtx.shape[0] - 1 - vert_1, mtx.shape[0] - 1)):
        for j in range(start_1, mtx.shape[1] - 1):
            mtx[i][j] = F(i * h, j * h)

    for i in reversed(range(1, vert_2 + 1)):
        for j in range(start_2, mtx.shape[1] - 1):
            mtx[i][j] = F(i * h, j * h)
```

```
133.
134.     return mtx
135.
136. def CG(eps = 10 ** (-2)):
137.     T = np.zeros((int((B + C) / h + 1), int(A / h + 1)))
138.     T_ = np.copy(T)
139.     r0 = b()
140.     p = np.copy(r0)
141.     it = 1
142.     while True:
143.         tmp = A_(p)
144.         alp = np.sum(r0 * r0) / np.sum(tmp * p)
145.         T = T_ +  alp * p
146.         r = r0 - alp * tmp
147.         bett = np.sum(r * r) / np.sum(r0 * r0)
148.         p = r + bett * p
149.
150.         if  np.linalg.norm(np.subtract(T, T_), ord = 1) < eps:
151.             break
152.         else:
153.             it += 1
154.             T_ = np.copy(T)
155.             r0 = np.copy(r)
156.
157.     print('Количество итераций ', it)
158.
159.     return T
160.
161. # 2-ой тестовый пример
162. # изменения: матрица T инициализируется не нулевой матрицей, а матрицей-конфигурацией пластины,
163. # поскольку появляются ненулевые граничные условия, то изменяется функция b()
164.
165. A = 2
166. B = 1
167. C = 1
168. h = 0.1
169. vert_1 = int(C / h - 1.)
170. vert_2 = int(B / h)
171. start_1 = 1
172. start_2 = int(0.5 * (A / h) + 1.)
173.
174. def g(y):
175.     return np.sin(np.pi * y)
176.
177. def F(x, y):
178.     return np.exp(-x) * np.sin(np.pi * y) * (np.pi ** 2 - 1)
179.
180. def initial(h = 0.1):
181.     size_1 = int((B + C) / h + 1)
182.     size_2 = int(A / h + 1)
183.     mtx = np.zeros((size_1, size_2))
184.     for i in reversed(range(int(B / h + 1), size_1 - 1)):
185.         mtx[i][0] = g(i * h)
186.     for j in range(1, int((A / 2) / h + 1)):
187.         mtx[int(B / h)][j] = 0
188.     for i in reversed(range(1, int(B / h))):
189.         mtx[i][int((A / 2) / h)] = g(i * h) / np.exp(1)
190.     for j in range(int((A / 2) / h + 1), size_2 - 1):
191.         mtx[0][j] = 0
192.     for i in range(1, size_1 - 1):
193.         mtx[i][size_2 - 1] = g(i * h) / np.exp(2)
194.     for j in range(1, size_2 - 1):
195.         mtx[size_1 -1][j] = 0
196.     return mtx
197.
198. def b():
199.     mtx = np.zeros((int((B + C) / h + 1), int(A / h + 1)))
200.     for i in reversed(range(mtx.shape[0] - 1 - vert_1, mtx.shape[0] - 1)):
201.         for j in range(start_1, mtx.shape[1] - 1):
202.             if j == start_1:
```

```
                mtx[i][j] = F(j * h, i * h) + g(i * h) / h ** 2
            elif j == mtx.shape[1] - 2:
                mtx[i][j] = F(j * h, i * h) + np.exp(-2) * g(i * h) / h ** 2
            else:
                mtx[i][j] = F(j * h, i * h)

    for i in reversed(range(1, vert_2 + 1)):
        for j in range(start_2, mtx.shape[1] - 1):
            if j == start_2:
                mtx[i][j] = F(j * h, i * h) + np.exp(-1) * g(i * h) / h ** 2
            elif j == mtx.shape[1] - 2:
                mtx[i][j] = F(j * h, i * h) + np.exp(-2) * g(i * h) / h ** 2
            else:
                mtx[i][j] = F(j * h, i * h)

    return mtx

# исходная задача
# изменение в функции initial, отсутствует F(x, y), изменние в функции b(), поскольку все граничные условия ненулевые

A = 150
B = 75
C = 45
Gamma_1 = Gamma_4 = 28 # граничные условия
Gamma_2 = Gamma_3 = 70
Gamma_5 = Gamma_6 = 25
h = 1

vert_1 = int(C / h - 1.)
vert_2 = int(B / h)
start_1 = 1
start_2 = int(0.5 * (A / h) + 1.)

def initial():
    size_1 = int((B + C) / h + 1)
    size_2 = int(A / h + 1)
    mtx = np.zeros((size_1, size_2))
    mtx[size_1 - 2 : int(B / h) : -1, 0] = Gamma_1
    mtx[int(B / h), : int((A / 2) / h + 1)] = Gamma_2
    mtx[int(B / h - 1) : : -1, int((A / 2) / h)] = Gamma_3
    mtx[0, int((A / 2) / h + 1) : size_2 - 1] = Gamma_4
    mtx[1 : size_1, size_2 - 1] = Gamma_5
    mtx[size_1 - 1, 1 : size_2 - 1] = Gamma_6
    mtx[size_1 - 1, 0] = mtx[0, size_2 - 1] = (Gamma_1 + Gamma_6) / 2
    return mtx

def b():
    mtx = np.zeros((int((B + C) / h + 1), int(A / h + 1)))
    for i in reversed(range(mtx.shape[0] - 1 - vert_1, mtx.shape[0] - 1)):
        for j in range(start_1, mtx.shape[1] - 1):
            if j == start_1 and i == mtx.shape[0] - 2:
                mtx[i][j] = (Gamma_1 + Gamma_6) / h ** 2
            elif j == start_1 and i == mtx.shape[0] - vert_1 - 1:
                mtx[i][j] = (Gamma_1 + Gamma_2) / h ** 2
            elif j == mtx.shape[1] - 2 and i == mtx.shape[0] - 2:
                mtx[i][j] = (Gamma_6 + Gamma_5) / h ** 2
            elif i == mtx.shape[0] - 2:
                mtx[i][j] = Gamma_6 / h ** 2
            elif i == mtx.shape[0] - vert_1 - 1 and j < start_2:
                mtx[i][j] = Gamma_2 / h ** 2
            elif j == start_1:
                mtx[i][j] = Gamma_1 / h ** 2
            elif j == mtx.shape[1] - 2:
                mtx[i][j] = Gamma_5 / h ** 2
            else:
                mtx[i][j] = 0

    for i in reversed(range(1, vert_2 + 1)):
        for j in range(start_2, mtx.shape[1] - 1):
```

```
272.             if j == start_2 and i == 1:
273.                 mtx[i][j] = (Gamma_3 + Gamma_4) / h ** 2
274.             elif j == mtx.shape[1] - 2 and i == 1:
275.                 mtx[i][j] = (Gamma_4 + Gamma_5) / h ** 2
276.             elif i == 1:
277.                 mtx[i][j] = Gamma_4 / h ** 2
278.             elif j == start_2:
279.                 mtx[i][j] = Gamma_3 / h ** 2
280.             elif j == mtx.shape[1] - 2:
281.                 mtx[i][j] = Gamma_5 / h ** 2
282.             else:
283.                 mtx[i][j] = 0
284.
285.     return mtx
```