

Содержание

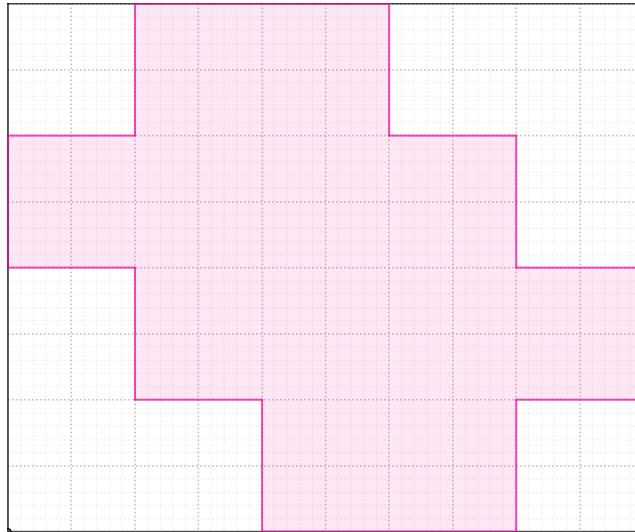
1	Задание	2
2	Теоретический материал	3
2.1	Метод Якоби	3
2.2	Неполная ортогонализация с рестартами ИОМ(m)	4
3	Тестовые задачи	5
3.1	Плоскость	5
3.2	Параболоид	6
3.3	Тригонометрическая функция	6
4	Вычислительный эксперимент	6
4.1	Прямоугольная область	6
4.1.1	Метод Якоби	6
4.1.2	ИОМ(m)	7
4.2	Индивидуальная область	9
4.2.1	Метод Якоби	9
4.2.2	ИОМ(m)	10
5	Приложение	11

1 Задание

Решить численно задачу Дирихле для уравнения Пуассона.

$$\begin{cases} -\Delta u = f, (x, y) \in \Omega, \\ u|_{\partial\Omega} = g. \end{cases}$$

Задачу необходимо решить для области $\Omega \subset \mathbb{R}^2$ в форме прямоугольника (с произвольными задаваемыми пользователем размерами), а также в форме, указанной в индивидуальном варианте:



Для решения СЛАУ следует использовать метод простой итерации (Якоби) и ИОМ(m).

Для задачи в прямоугольнике подготовить тестовые примеры, в каждом из которых необходимо выдавать норму ошибки (разности точного и вычисленного решений). В качестве тестовых, в том числе, следует использовать задачи, решением которых являются плоскости, параболоиды, комбинации тригонометрических функций.

Предусмотрите подсчет количества итераций, потребовавшихся для достижения заданной точности.

Сравните время работы двух методов при разном значении точности.

Примечание 1. Начало координат можно выбрать в любом удобном для заданной фигуры месте. Шаги сеток по обоим переменным можно взять равными.

Примечание 2. Структура данных для хранения решения (и, возможно, промежуточных переменных) должна быть оптимальной по памяти и не содержать фиктивные узлы, не входящие в заданную область.

2 Теоретический материал

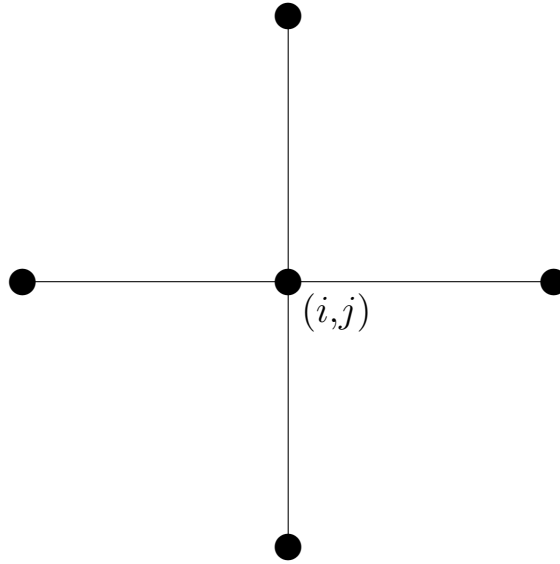
2.1 Метод Якоби

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

Погрешность аппроксимации

$$\Psi = \mathcal{O}(h_1^2 + h_2^2)$$

Шаблон РС



Количество итераций

Зафиксируем произвольную точность ε , $h_1 = h_2 = h$, $\frac{1}{h} = N$

$$K(\varepsilon) \approx \frac{2 \ln \left(\frac{1}{\varepsilon} \right)}{\pi^2} \left(\frac{1}{h} \right)^2 = \mathcal{O}(N^2)$$

Расчетные формулы для программирования:

$$U_{i,j}^{(m+1)} = C_0 F_{i,j} + C_1 \left(U_{i,j-1}^{(m)} + U_{i,j+1}^{(m)} \right) + C_2 \left(U_{i-1,j}^{(m)} + U_{i+1,j}^{(m)} \right)$$

где

$$C_0 = \frac{0.5h_1^2h_2^2}{h_1^2 + h_2^2}, \quad C_1 = \frac{0.5h_1^2}{h_1^2 + h_2^2}, \quad C_2 = \frac{0.5h_2^2}{h_1^2 + h_2^2}$$

2.2 Неполная ортогонализация с рестартами ИОМ(m)

Выберем в качестве подпространств \mathcal{K} и \mathcal{L} подпространство Крылова

$$\mathcal{K} = \mathcal{L} = \mathcal{K}_m(v_r, A),$$

для построения которого используется орт начальной невязки

$$v_1 = r_0 / \|r_0\|_2.$$

Для построения базиса в \mathcal{K} (и также в \mathcal{L}) будем использовать ортогонализацию Арнольди с начальным вектором v_1 .

В соответствии с общим проекционным подходом решение должно уточняться по формуле

$$x = x_0 + Vy,$$

где y является решением системы

$$(W^T AV)y = W^T r_0,$$

в которой V и W - матрицы, составленные из базисных векторов подпространства \mathcal{K} и \mathcal{L} соответственно.

В нашем случае эти матрицы равны

$$V = W = V_m.$$

Следовательно,

$$W^T AV = V_m^T AV_m = H_m$$

Обозначим для краткости $\beta = \|r_0\|_2$.

$$W^T r_0 = V_m^T r_0 = V_m^T (\beta v_1) = \beta e_1,$$

где e_1 - единичный декартов орт, первая координата которого равна единице, остальные - нулю.

Таким образом, вспомогательная СЛАУ принимает вид

$$H_m y = \beta e_1.$$

Матрица H_m невырождена, а также легко обратима, поскольку имеет хессенбергову форму. Для решения системы с такой матрицей требуется занулить одну поддиагональ. Это можно сделать гауссовыми исключениями, либо ортогональными вращениями Гивенса.

Так как приходится хранить все базисные вектора пространства \mathcal{K} , то на это уходит много объема памяти.

Один из способов уменьшения затрат в алгоритме состоит в том, чтобы периодически обновлять алгоритм, а именно, останавливать раньше при некотором заданном значении m (как правило, небольшом). Поскольку в такой ситуации решение системы, скорее всего, найдено не будет, то процесс нужно повторять, используя в качестве нового начального приближения вектор x_m , полученный после неполного выполнения алгоритма.

Другим вариантом экономии ресурсов памяти в алгоритме является введение неполной ортогонализации. Идея заключается в том, что очередной вектор v_{j+1} (получаемый на j -ом шаге основного цикла) делается ортогональным только к k предыдущим векторам, а не ко всем.

Алгоритм

1. Вычислить $r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, $v_1 := r_0/\beta$
2. **For** $j = 1, 2, \dots, m$
3. Вычислить $w_j := Av_j$
4. **For** $i = \max\{1, j - k + 1\} \dots j$
5. $h_{i,j} := (w_j, v_i)$
6. $w_j := w_j - h_{i,j}v_i$
7. **EndFor**
8. Вычислить $h_{j+1,j} := \|w_j\|_2$
9. **If** $h_{j+1,j} = 0$ **then** положить $m := j$ и выйти из цикла **EndIf**
10. Вычислить $v_{j+1} := w_j/h_{j+1,j}$
11. **EndFor**
12. Преобразовать матрицу H (порядка m), исключив из нее поддиагональ.

Выполнить соответствующие преобразования с вектором $g = \beta e_1$

13. Решить треугольную СЛАУ $Hy = g$ порядка m
14. Вычислить $x_m = x_0 + \sum_{i=1}^m y_i v_i$
15. **If** качество приближения удовлетворительное **then** выход
16. Положить $x_0 = x_m$, вернуться в шаг 1.

3 Тестовые задачи

3.1 Плоскость

$$\begin{cases} -\Delta u = 0, \\ u|_{\partial\Omega} = 2x + 3y - 5 \end{cases}$$

3.2 Параболоид

$$\begin{cases} -\Delta u = -2, \\ u|_{\partial\Omega} = 3x^2 - 2y^2 - 1 \end{cases}$$

3.3 Тригонометрическая функция

$$\begin{cases} -\Delta u = -e^{\sin^2 x + \cos^2 y} (\sin^2 2x + 2 \cos 2x + \sin 2y - 2 \cos 2y) \\ u|_{\partial\Omega} = e^{\sin^2 x + \cos^2 y} \end{cases}$$

4 Вычислительный эксперимент

4.1 Прямоугольная область

4.1.1 Метод Якоби

$$h_1 = h_2 = h_0 = 0.1, \varepsilon_0 = 0.1$$

Тестовая задача №1

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0023	0.0086	0.036	0.1464
$\varepsilon_0/10$	0.0127	0.0725	0.3163	1.334
$\varepsilon_0/100$	0.0182	0.217	2.123	12.9
$\varepsilon_0/1000$	0.0274	0.3697	4.541	50.1

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	10	10	11	11
$\varepsilon_0/10$	50	94	95	101
$\varepsilon_0/100$	96	276	656	976
$\varepsilon_0/1000$	142	462	1400	3803

Тестовая задача №2

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0018	0.0068	0.0263	0.105
$\varepsilon_0/10$	0.0069	0.0404	0.1966	0.8841
$\varepsilon_0/100$	0.0161	0.1614	1.304	6.852
$\varepsilon_0/1000$	0.026	0.3188	3.703	35.55

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	6	7	7	7
$\varepsilon_0/10$	33	50	58	65
$\varepsilon_0/100$	78	202	398	513
$\varepsilon_0/1000$	124	387	1100	2625

Тестовая задача №3

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0086	0.0423	0.1827	0.7716
$\varepsilon_0/10$	0.0364	0.2554	1.389	6.422
$\varepsilon_0/100$	0.0993	0.9446	8.053	47.99
$\varepsilon_0/1000$	0.123	1.663	20.41	213.6

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	8	9	10	10
$\varepsilon_0/10$	38	61	78	89
$\varepsilon_0/100$	84	223	463	661
$\varepsilon_0/1000$	130	408	1188	2961

4.1.2 ИОМ(m)

$m = 90$, $k = 2$ - количество векторов, к которым происходит ортогонализация

Тестовая задача №1

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0657	0.2517	0.9379	14.4999
$\varepsilon_0/10$	0.069	0.2414	1.8968	14.4807
$\varepsilon_0/100$	0.0645	0.2288	1.8473	17.9485
$\varepsilon_0/1000$	0.0642	0.2434	1.8159	17.7229

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	1	1	1	4
$\varepsilon_0/10$	1	1	1	4
$\varepsilon_0/100$	1	1	2	5
$\varepsilon_0/1000$	1	1	2	5

Тестовая задача №2

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0644	0.2404	0.9174	10.8944
$\varepsilon_0/10$	0.0643	0.2386	1.8933	14.3311
$\varepsilon_0/100$	0.0668	0.2391	1.8157	15.1238
$\varepsilon_0/1000$	0.0652	0.2349	1.8497	18.3193

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	1	1	1	3
$\varepsilon_0/10$	1	1	2	4
$\varepsilon_0/100$	1	1	2	4
$\varepsilon_0/1000$	1	1	2	5

Тестовая задача №3

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0707	0.2348	1.9425	15.0457
$\varepsilon_0/10$	0.0648	0.2418	1.9311	14.8658
$\varepsilon_0/100$	0.0654	0.2288	1.8317	18.2153
$\varepsilon_0/1000$	0.0652	0.2411	1.858	22.4428

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	1	1	2	4
$\varepsilon_0/10$	1	1	2	4
$\varepsilon_0/100$	1	1	2	5
$\varepsilon_0/1000$	1	1	2	6

4.2 Индивидуальная область

4.2.1 Метод Якоби

$$h_1 = h_2 = 0.5 = h_0, L_1 = 5, L_2 = 4, \varepsilon_0 = 0.1$$

Тестовая задача №1

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0071	0.0688	0.5821	6.0051
$\varepsilon_0/10$	0.0091	0.1013	0.8987	11.6856
$\varepsilon_0/100$	0.0112	0.116	1.3587	18.0
$\varepsilon_0/1000$	0.018	0.1969	1.7485	24.1388

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	16	53	163	490
$\varepsilon_0/10$	24	85	298	1018
$\varepsilon_0/100$	31	118	432	1561
$\varepsilon_0/1000$	39	150	566	2101

Тестовая задача №2

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0051	0.0704	0.8356	9.1455
$\varepsilon_0/10$	0.0103	0.1015	1.2036	15.2191
$\varepsilon_0/100$	0.008	0.1599	1.6532	21.3458
$\varepsilon_0/1000$	0.0101	0.1979	1.9913	27.5801

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	21	71	241	789
$\varepsilon_0/10$	28	104	375	1332
$\varepsilon_0/100$	35	136	509	1873
$\varepsilon_0/1000$	43	169	642	2413

Тестовая задача №3

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0146	0.106	1.25	11.0604
$\varepsilon_0/10$	0.0102	0.2424	2.9116	36.5327
$\varepsilon_0/100$	0.017	0.4943	4.7117	66.0623
$\varepsilon_0/1000$	0.0402	0.456	6.7337	96.749

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	9	27	77	193
$\varepsilon_0/10$	15	56	194	632
$\varepsilon_0/100$	23	88	324	1151
$\varepsilon_0/1000$	30	121	457	1687

4.2.2 ИОМ(m)

Тестовая задача №1

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.0783	0.1935	0.525	8.4078
$\varepsilon_0/10$	0.0868	0.1513	1.4902	10.294
$\varepsilon_0/100$	0.0764	0.189	1.5829	11.8592
$\varepsilon_0/1000$	0.1337	0.1829	1.4936	13.6049

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	1	1	1	5
$\varepsilon_0/10$	1	1	3	6
$\varepsilon_0/100$	1	1	3	7
$\varepsilon_0/1000$	2	1	3	8

Тестовая задача №2

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.1501	0.1936	3.3179	30.2108
$\varepsilon_0/10$	0.1375	0.358	3.8941	43.4151
$\varepsilon_0/100$	0.1552	0.4538	5.4603	43.4714
$\varepsilon_0/1000$	0.1048	0.5733	7.3711	92.1802

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	2	1	7	18
$\varepsilon_0/10$	2	2	9	26
$\varepsilon_0/100$	2	3	12	26
$\varepsilon_0/1000$	2	4	17	56

Тестовая задача №3

Время расчета(в секундах) от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	0.2309	0.2379	2.7948	49.0137
$\varepsilon_0/10$	0.1956	0.437	3.2799	54.093
$\varepsilon_0/100$	0.1995	0.4077	3.6902	72.4402
$\varepsilon_0/1000$	0.2247	0.7736	4.0087	87.2611

Число итераций от точности и шага сетки

	h_0	$h_0/2$	$h_0/4$	$h_0/8$
$\varepsilon_0/1$	3	1	6	30
$\varepsilon_0/10$	3	3	7	33
$\varepsilon_0/100$	3	3	8	44
$\varepsilon_0/1000$	4	5	9	53

5 Приложение

На прямоугольной области

```
1 import numpy as np
2 import time
3 import numpy.linalg as lin
4
5 L_x = 2 # размер области по x
6 L_y = 1 # размер области по y
7 h_x = 0.1 # шаг по x
8 h_y = 0.1 # шаг по y
9 eps = 10 ** (-8) # точность решения
10
11 #####
12 # -----Плоскость-----#
```

```

13  # test = 1
14
15
16  # -----Параболоид-----#
17  # test = 2
18
19
20  # -----Индивидуальная функция-----#
21  # test = 3
22  #####
23
24  # функция g
25  def func_u(x, y):
26      match test:
27          case 1:
28              return 2 * x + 3 * y - 5
29          case 2:
30              return 3 * x ** 2 - 2 * y ** 2 - 1
31          case 3:
32              return np.exp((np.sin(x)) ** 2 + (np.cos(y)) ** 2)
33
34
35  # функция f
36  def func_f(x, y):
37      match test:
38          case 1:
39              return 0
40          case 2:
41              return -2
42          case 3:
43              return -1 * np.exp(
44                  (np.sin(x)) ** 2 + (np.cos(y)) ** 2) * \
45                  ((np.sin(2 * x)) ** 2 + 2 * np.cos(
46                      2 * x) + np.sin(2 * y) -
47                      2 * np.cos(2 * y))
48
49
50  # Метод простой итерации(Якоби)
51  def Cross_Jac(f, h_1, h_2, y0, ep):
52      # счетчик времени
53      tic = time.perf_counter()
54      # коэффициенты в расчетной формуле
55      C_0 = (0.5 * (h_1 ** 2) * (h_2 ** 2)) / (

```

```

56         h_1 ** 2 + h_2 ** 2)
57     C_1 = (0.5 * (h_1 ** 2)) / (h_1 ** 2 + h_2 ** 2)
58     C_2 = (0.5 * (h_2 ** 2)) / (h_1 ** 2 + h_2 ** 2)
59     # число точек разбиений по осям
60     Nx = int(L_x / h_1) + 1
61     Ny = int(L_y / h_2) + 1
62
63     # матрица решений
64     U = np.zeros((Ny, Nx))
65     # промежуточная матрица решений
66     U_m = np.zeros((Ny, Nx))
67
68     # сетка по x
69     x = np.linspace(0, L_x, Nx)
70     # сетка по y
71     y = np.linspace(0, L_y, Ny)
72
73     # точное решение
74     X, Y = np.meshgrid(x, y)
75     Z = func_u(X, Y)
76
77     # Заполняем начальное приближение с учетом нач.условий
78     for i in range(0, Ny):
79         for j in range(0, Nx):
80             U[i][j] = y0(x[j], 0)
81     for i in range(0, Nx): # y = 1
82         U[Ny - 1][i] = y0(x[i], L_y)
83     for i in range(0, Ny): # x = 1
84         U[i][Nx - 1] = y0(L_x, y[i])
85     for i in range(0, Ny): # x = 0
86         U[i][0] = y0(0, y[i])
87
88     for i in range(0, Ny):
89         for j in range(0, Nx):
90             U_m[i][j] = y0(x[j], 0)
91     for i in range(0, Nx): # y = 1
92         U_m[Ny - 1][i] = y0(x[i], L_y)
93     for i in range(0, Ny): # x = 1
94         U_m[i][Nx - 1] = y0(L_x, y[i])
95     for i in range(0, Ny): # x = 0
96         U_m[i][0] = y0(0, y[i])
97
98     norm = 1 # значение нормы

```

```

99     k = 0  # число итераций
100     while norm > ep:
101         for i in range(Nx - 2, 0, -1):
102             for j in range(1, Ny - 1):
103                 U_m[j][i] = C_0 * f(x[i], y[j]) + C_1 * (
104                     U[j - 1][i] + U[j + 1][
105                         i]) + C_2 * (U[j][i - 1] + U[j][
106                             i + 1])
107             norm = np.max(np.abs(U_m - U))
108             k += 1
109             for i in range(0, Ny):
110                 for j in range(0, Nx):
111                     U[i][j] = U_m[i][j]
112             print(norm)
113
114     # фиксируем счетчик времени
115     toc = time.perf_counter()
116     tme = round(toc - tic, 4)
117     print("Время = ", tme)
118     print("Погрешность с точным решением: ", lin.norm(Z - U_m))
119     print("Количество итераций = ", k)
120     return U_m
121
122
123     print("*****Jacobi*****")
124     sol = Cross_Jac(func_f, h_x, h_y, func_u, eps)
125
126
127     # Точное решение
128     def solve_of_task(U):
129         trsol = np.zeros((N_x + 1, N_y + 1))
130         for i in range(N_x + 1):
131             for j in range(N_y + 1):
132                 trsol[i][j] = U(i * h_x, j * h_y)
133         return trsol
134
135
136     # Вращения Гивенса
137     def givens(A, N):
138         for l in range(N - 1):
139             for i in range(N - 1, 0 + 1, -1):
140                 j = i - 1
141                 if A[i][1] != 0:

```

```

142         alem = A[j][1]
143         belem = A[i][1]
144         if np.abs(belem) > np.abs(alem):
145             tau = alem / belem
146             S = 1 / np.sqrt(1 + tau ** 2)
147             C = S * tau
148         else:
149             tau = belem / alem
150             C = 1 / np.sqrt(1 + tau ** 2)
151             S = C * tau
152         A[i], A[j] = A[i] * C - A[j] * S, A[j] * C + A[i]
            ↪ * S
153     return A
154
155
156     # обратный ход метода Гаусса
157     def Gauss_back_step(A, B, N):
158         sol = np.zeros(N)
159         for i in range(N - 1, -1, -1):
160             s = 0
161             if i == N - 1:
162                 sol[i] = B[i] / A[i][i]
163             else:
164                 for j in range(i + 1, N, 1):
165                     s += A[i][j] * sol[j]
166                 sol[i] = (B[i] - s) / A[i][i]
167         return sol
168
169
170     # произведение матрицы системы A на произвольный массив p
171     def multA(p):
172         Ap = np.copy(p)
173         for i in range(1, Ap.shape[0] - 1):
174             for j in range(1, Ap.shape[1] - 1):
175                 Ap[i][j] = -1 * ((p[i - 1][j] - 2 * p[i][j] + p[i +
            ↪ 1][j])) / h_x ** 2 + (
176                     p[i][j - 1] - 2 * p[i][j] + p[i][j + 1]) /
            ↪ h_y ** 2)
177         return Ap
178
179
180     # скалярное произведение вектор-матриц
181     def Scr(a, b):

```

```

182     sum = 0
183     for i in range(a.shape[0]):
184         for j in range(a.shape[1]):
185             sum += a[i][j] * b[i][j]
186     return sum
187
188
189     # число точек разбиений по осям
190     N_x = int(L_x / h_x)
191     N_y = int(L_x / h_x)
192     # задание правой части
193     B2 = np.zeros((N_x + 1, N_y + 1))
194     for i in range(N_y + 1):
195         B2[0][i] = func_u(0 * h_x, i * h_y)
196         B2[N_x][i] = func_u(N_x * h_x, i * h_y)
197     for i in range(N_x + 1):
198         B2[i][0] = func_u(i * h_x, 0 * h_y)
199         B2[i][N_y] = func_u(i * h_x, N_y * h_y)
200     for i in range(1, N_x):
201         for j in range(1, N_y):
202             B2[i][j] = func_f(i * h_x, j * h_y)
203
204
205     # Проекционный метод IOM(m)
206
207     def IOM_m(vec_b, m):
208         solution = np.zeros((N_x + 1, N_y + 1))
209         k = 1 # количество векторов, к которым будет ортогонален
210             ↪ очередной вектор
211         x0 = np.zeros((N_x + 1, N_y + 1)) # Начальное приближение
212         # задание краевых значений
213         for ik in range(N_y + 1):
214             x0[0][ik] = func_u(0 * h_x, ik * h_y)
215             x0[N_x][ik] = func_u(N_x * h_x, ik * h_y)
216         for jk in range(N_x + 1):
217             x0[jk][0] = func_u(jk * h_x, 0 * h_y)
218             x0[jk][N_y] = func_u(jk * h_x, N_y * h_y)
219
220         r0 = vec_b - multA(x0) # вектор начальной невязки
221         count_iter = 0
222         while abs(lin.norm(r0)) > eps:
223             V = np.zeros((N_x + 1, (m + 1) * (N_y + 1))) # матрица
224                 ↪ базисных векторов из пространства K

```



```

223 H = np.zeros((m + 1, m)) # матрица коэффициентов
    ↪ ортогонализации
224 r0 = vec_b - multA(x0) # вектор начальной невязки
225 beta = lin.norm(r0) # норма начальной невязки
226 V[:, :N_y + 1] = r0 / beta # первый базисный вектор
    ↪ пространства K
227 for j in range(1, m + 1):
228     omega_j = multA(V[:, (j - 1) * (N_y + 1): j * (N_y +
    ↪ 1)]) # базисный вектор пространства L
229     for i in range(max(1, j - k + 1), j + 1):
230         H[i - 1][j - 1] = Scr(omega_j,
231                                V[:, (i - 1) * (N_y + 1): i
    ↪ * (N_y + 1)]) #
    ↪ вычисление коэффициента
    ↪ орт-ции
232         omega_j = omega_j - H[i - 1][j - 1] * V[:, (i -
    ↪ 1) * (N_y + 1): i * (
233                 N_y + 1)] # орт-ция очередного базисного
    ↪ вектора про-ва L
234 H[j][j - 1] = lin.norm(omega_j) # норма орт-го
    ↪ вектора
235 if abs(H[j][j - 1]) < 10 ** (-8):
236     m = j
237     break
238 V[:, j * (N_y + 1): (j + 1) * (N_y + 1)] = omega_j /
    ↪ H[j][j - 1] # вычисление следующего вектора
    ↪ про-ва K
239 e_1 = np.zeros(m + 1) # орт
240 e_1[0] = 1
241 g = beta * e_1 # вектор правой части вспомогательной
    ↪ СЛАУ
242 H = np.c_[H, g] # добавление к матрице системы правой
    ↪ части
243 H = givens(H, m + 1) # зануляем поддиагональ вращениями
    ↪ Гивенса
244 g = H[:, m] # перезаписываем измененную правую часть
245 H = np.delete(np.delete(H, m, 1), m, 0) # удаляем вектор
    ↪ правой части из системы
246 y = Gauss_back_step(H, g, m) # обратный ход метода
    ↪ Гауса
247 # Уточнение решения
248 sumyivi = np.zeros((N_x + 1, N_y + 1)) # уточняющий
    ↪ вектор

```

```

249         for f in range(1, m + 1):
250             sumyivi += y[f - 1] * V[:, (f - 1) * (N_y + 1): f *
                ↪ (N_y + 1)] # вычисление уточняющего вектора
251             solution = x0 + sumyivi # уточнение
252             r0 = vec_b - multA(solution) # вычисление вектора
                ↪ начальной невязки
253             x0 = solution # изменение начального приближения
254             count_iter += 1
255         return solution, count_iter
256
257
258     print("*****projection*****")
259     ts = time.time()
260     lol, ver = IOM_m(B2, 100)
261     tf = time.time()
262     sol_anal = solve_of_task(func_u)
263     print('Время = ', tf - ts)
264     print('Погрешность с точным решением: = ', lin.norm(sol_anal -
        ↪ lol))
265     print('Количество итераций = ', ver)

```

На сложной области

```

1
2 import numpy as np
3 import time
4 import numpy.linalg as lin
5
6 dim = 1 # размер одного квадрата подобласти
7 L_x = 5 * dim # размер области по x
8 L_y = 4 * dim # размер области по y
9 h_x = 0.25 # шаг по x
10 h_y = 0.25 # шаг по y
11 eps = 0.00001 # точность
12
13 #####
14 # -----Плоскость-----#
15 # test = 1
16
17
18 # -----Параболоид-----#

```

```

19  # test = 2
20
21
22  # -----Индивидуальная функция-----#
23  test = 3
24
25
26  #####
27  # функция g
28  def func_u(x, y):
29      match test:
30          case 1:
31              return 2 * x + 3 * y - 5
32          case 2:
33              return 3 * x ** 2 - 2 * y ** 2 - 1
34          case 3:
35              return np.exp((np.sin(x)) ** 2 + (np.cos(y)) ** 2)
36
37
38  # функция f
39  def func_f(x, y):
40      match test:
41          case 1:
42              return 0 + x - x
43          case 2:
44              return -2 + x - x
45          case 3:
46              return -1 * np.exp(
47                  (np.sin(x)) ** 2 + (np.cos(y)) ** 2) * \
48                  ((np.sin(2 * x)) ** 2 + 2 * np.cos(
49                      2 * x) + np.sin(2 * y) -
50                      2 * np.cos(2 * y))
51
52
53  # Метод простой итерации(Якоби)
54  def Cross_Jac(f, h_1, h_2, y0, ep):
55      # коэффициенты в расчетной формуле
56      C_0 = (0.5 * (h_1 ** 2) * (h_2 ** 2)) / (
57          h_1 ** 2 + h_2 ** 2)
58      C_1 = (0.5 * (h_2 ** 2)) / (h_1 ** 2 + h_2 ** 2)
59      C_2 = (0.5 * (h_1 ** 2)) / (h_1 ** 2 + h_2 ** 2)
60      # число точек разбиений по осям
61      Nx = int(L_x / h_1) + 1

```

```

62 Ny = int(L_y / h_2) + 1
63 # сетка по x
64 x = np.linspace(0, L_x, Nx)
65 # сетка по y
66 y = np.linspace(0, L_y, Ny)
67
68 # 4 subareas
69 # 1:
70 Ny_1 = int((Ny - 1) / 4) + 1
71 Nx_1 = int(2 * (Nx - 1) / 5) + 1
72 U_sub_1 = np.zeros((Ny_1, Nx_1))
73 # 2:
74 Ny_2 = int((Ny - 1) / 4) + 1
75 Nx_2 = int(4 * (Nx - 1) / 5) + 1
76 U_sub_2 = np.zeros((Ny_2, Nx_2))
77 # 3:
78 Ny_3 = int((Ny - 1) / 4) + 1
79 Nx_3 = int(4 * (Nx - 1) / 5) + 1
80 U_sub_3 = np.zeros((Ny_3, Nx_3))
81 # 4:
82 Ny_4 = int((Ny - 1) / 4) + 1
83 Nx_4 = int(2 * (Nx - 1) / 5) + 1
84 U_sub_4 = np.zeros((Ny_4, Nx_4))
85
86 # input initial values
87 # First area
88 for i in range(Ny_1):
89     for j in range(Nx_1):
90         U_sub_1[i][j] = y0(x[int((Nx - 1) / 5) + j], y[Ny -
91             ↪ 1])
92 for i in range(Ny_1):
93     U_sub_1[i][0] = y0(x[int((Nx - 1) / 5)], y[Ny - 1 - i])
94 for i in range(Ny_1):
95     U_sub_1[i][Nx_1 - 1] = y0(x[int(3 * (Nx - 1) / 5)], y[Ny
96         ↪ - 1 - i])
97
98 # Second area
99 for i in range(Ny_2):
100     for j in range(Nx_2):
101         U_sub_2[i][j] = y0(x[j], y[Ny - 1])
102 for j in range(int((Nx - 1) / 5) + 1):
103     U_sub_2[0][j] = y0(x[j], y[int(3 * (Ny - 1) / 4)])
104 for i in range(Ny_2):

```

```

103     U_sub_2[i][0] = y0(x[0], y[int(3 * (Ny - 1) / 4) - i])
104 for j in range(int((Nx - 1) / 5) + 1):
105     U_sub_2[Ny_2 - 1][j] = y0(x[j], y[int(2 * (Ny - 1) / 4)])
106 for j in range(int((Nx - 1) / 5) + 1):
107     U_sub_2[0][j + int(3 * (Nx_2 - 1) / 4)] = y0(x[int(3 *
    ↪ (Nx - 1) / 5) + j], y[int(3 * (Ny - 1) / 4)])
108 for i in range(int(Ny_2)):
109     U_sub_2[i][Nx_2 - 1] = y0(x[int(4 * (Nx - 1) / 5)],
    ↪ y[int(3 * (Ny - 1) / 4) - i])
110
111 # Third area
112 for i in range(Ny_3):
113     for j in range(Nx_3):
114         U_sub_3[i][j] = y0(x[j + int((Nx - 1) / 5)], y[Ny -
    ↪ 1])
115 for i in range(Ny_3):
116     U_sub_3[i][0] = y0(x[int((Nx - 1) / 5)], y[int(2 * (Ny -
    ↪ 1) / 4) - i])
117 for j in range(Ny_3):
118     U_sub_3[Ny_3 - 1][j] = y0(x[int((Nx - 1) / 5) + j],
    ↪ y[int((Ny - 1) / 4)])
119 for j in range(Ny_3):
120     U_sub_3[Ny_3 - 1][j + int(3 * (Nx_3 - 1) / 4)] =
    ↪ y0(x[int(4 * (Nx - 1) / 5) + j], y[int((Ny - 1) /
    ↪ 4)])
121 for i in range(Ny_3):
122     U_sub_3[i][Nx_3 - 1] = y0(x[Nx - 1], y[int(2 * (Ny - 1) /
    ↪ 4) - i])
123 for j in range(Ny_3):
124     U_sub_3[0][j + int(3 * (Nx_3 - 1) / 4)] = y0(x[int(4 *
    ↪ (Nx - 1) / 5) + j], y[int(2 * (Ny - 1) / 4)])
125
126 # fourth area
127 for i in range(Ny_4):
128     for j in range(Nx_4):
129         U_sub_4[i][j] = y0(x[int(2 * (Nx - 1) / 5) + j], y[Ny
    ↪ - 1])
130 for i in range(Ny_4):
131     U_sub_4[i][0] = y0(x[int(2 * (Nx - 1) / 5)], y[int((Ny -
    ↪ 1) / 4) - i])
132 for j in range(Nx_4):
133     U_sub_4[Ny_4 - 1][j] = y0(x[int(2 * (Nx - 1) / 5) + j],
    ↪ y[0])

```

```

134     for i in range(Ny_4):
135         U_sub_4[i][Nx_4 - 1] = y0(x[int(4 * (Nx - 1) / 5)],
            ↪ y[int((Ny - 1) / 4) - i])
136
137     # обойдем с 1 по 4 области
138     prev_sol1 = U_sub_1.copy()
139     prev_sol2 = U_sub_2.copy()
140     prev_sol3 = U_sub_3.copy()
141     prev_sol4 = U_sub_4.copy()
142     it = 0 # умеем считать
143     nrm = 1
144     while nrm > eps:
145         # first area
146         for i in range(1, Ny_1 - 1):
147             for j in range(1, Nx_1 - 1):
148                 U_sub_1[i][j] = C_0 * f(x[j + int((Nx - 1) / 5)],
            ↪ y[Ny - 1 - i]) + C_1 * (
149                     U_sub_1[i + 1][j] + U_sub_1[i - 1][j]) \
150                     + C_2 * (U_sub_1[i][j - 1] +
            ↪ U_sub_1[i][j + 1])
151         # border between first and second area
152         for j in range(1, Nx_1 - 1):
153             U_sub_1[Ny_1 - 1][j] = C_0 * f(x[j + int((Nx - 1) /
            ↪ 5)], y[Ny - Ny_1]) + C_1 * (
154                 U_sub_2[1][int((Nx_2 - 1) / 4) + j] +
            ↪ U_sub_1[Ny_1 - 2][j]) \
155                 + C_2 * (U_sub_1[Ny_1 - 1][j -
            ↪ 1] + U_sub_1[Ny_1 - 1][j +
            ↪ 1])
156             U_sub_2[0][int((Nx_2 - 1) / 4) + j] = U_sub_1[Ny_1 -
            ↪ 1][j]
157
158         # second area
159         for i in range(1, Ny_2 - 1):
160             for j in range(1, Nx_2 - 1):
161                 U_sub_2[i][j] = C_0 * f(x[j], y[int(3 * (Ny - 1)
            ↪ / 4) - i]) + C_1 * (
162                     U_sub_2[i + 1][j] + U_sub_2[i - 1][j]) \
163                     + C_2 * (U_sub_2[i][j - 1] +
            ↪ U_sub_2[i][j + 1])
164         # border between second and third area
165         for j in range(int((Nx_2 - 1) / 4) + 1, Nx_2 - 1):

```

```

166     U_sub_2[Ny_2 - 1][j] = C_0 * f(x[j], y[int(3 * (Ny -
    ↪ 1) / 4) - Ny_2 + 1]) + C_1 * (
167         U_sub_3[1][j - int((Nx_2 - 1) / 4)] +
    ↪ U_sub_2[Ny_2 - 2][j]) \
168         + C_2 * (U_sub_2[Ny_2 - 1][j -
    ↪ 1] + U_sub_2[Ny_2 - 1][j +
    ↪ 1])
169     U_sub_3[0][j - int((Nx_2 - 1) / 4)] = U_sub_2[Ny_2 -
    ↪ 1][j]
170     # third area
171     for i in range(1, Ny_3 - 1):
172         for j in range(1, Nx_3 - 1):
173             U_sub_3[i][j] = C_0 * f(x[int((Nx - 1) / 5) + j],
    ↪ y[int(2 * (Ny - 1) / 4) - i]) + C_1 * (
174                 U_sub_3[i + 1][j] + U_sub_3[i - 1][j]) \
175                 + C_2 * (U_sub_3[i][j - 1] +
    ↪ U_sub_3[i][j + 1])
176     # border between third and fourth area
177     for j in range(int((Nx_3 - 1) / 4) + 1, Nx_3 - int((Nx_3
    ↪ - 1) / 4) - 1):
178         U_sub_3[Ny_3 - 1][j] = C_0 * f(x[int((Nx - 1) / 5) +
    ↪ j], y[int(2 * (Ny - 1) / 4) - Ny_3 + 1]) + C_1 *
    ↪ (
179             U_sub_4[1][j - int((Nx_3 - 1) / 4)] +
    ↪ U_sub_3[Ny_3 - 2][j]) \
180             + C_2 * (U_sub_3[Ny_3 - 1][j -
    ↪ 1] + U_sub_3[Ny_3 - 1][j +
    ↪ 1])
181         U_sub_4[0][j - int((Nx_3 - 1) / 4)] = U_sub_3[Ny_3 -
    ↪ 1][j]
182     # fourth area
183     for i in range(1, Ny_4 - 1):
184         for j in range(1, Nx_4 - 1):
185             U_sub_4[i][j] = C_0 * f(x[int(2 * (Nx - 1) / 5) +
    ↪ j], y[int((Ny - 1) / 4) - i]) + C_1 * (
186                 U_sub_4[i + 1][j] + U_sub_4[i - 1][j]) \
187                 + C_2 * (U_sub_4[i][j - 1] +
    ↪ U_sub_4[i][j + 1])
188     # calculate norm on all areas
189     mx1 = lin.norm(np.abs(prev_sol1 - U_sub_1))
190     mx2 = lin.norm(np.abs(prev_sol2 - U_sub_2))
191     mx3 = lin.norm(np.abs(prev_sol3 - U_sub_3))
192     mx4 = lin.norm(np.abs(prev_sol4 - U_sub_4))

```

```

193         nrm = max(mx1, mx2, mx3, mx4)
194
195         # change initial solve
196         prev_sol1 = U_sub_1.copy()
197         prev_sol2 = U_sub_2.copy()
198         prev_sol3 = U_sub_3.copy()
199         prev_sol4 = U_sub_4.copy()
200
201         print(nrm)
202         it += 1
203
204         # r1 = lin.norm(sub1_z - U_sub_1)
205         # r2 = lin.norm(sub2_z - U_sub_2)
206         # r3 = lin.norm(sub3_z - U_sub_3)
207         # r4 = lin.norm(sub4_z - U_sub_4)
208
209         # r1 = lin.norm()
210
211         # r = max(r1, r2, r3, r4)
212         return nrm, it
213
214
215     print("*****Jacobi*****")
216     # start time
217     ts = time.time()
218     sol, pop = Cross_Jac(func_f, h_x, h_y, func_u, eps)
219     # finish time
220     tf = time.time()
221     print('Невязка = ', sol)
222     print('Итераций = ', pop)
223     print('Время = ', tf - ts)
224
225
226     # Точное решение
227     def solve_of_task(U):
228         trsol = np.zeros((N_x + 1, N_y + 1))
229         for i in range(N_x + 1):
230             for j in range(N_y + 1):
231                 trsol[i][j] = U(i * h_x, j * h_y)
232         return trsol
233
234
235     # Вращения Гивенса

```



```

236 def givens(A, N):
237     for l in range(N - 1):
238         for i in range(N - 1, 0 + 1, -1):
239             j = i - 1
240             if A[i][1] != 0:
241                 alem = A[j][1]
242                 belem = A[i][1]
243                 if np.abs(belem) > np.abs(alem):
244                     tau = alem / belem
245                     S = 1 / np.sqrt(1 + tau ** 2)
246                     C = S * tau
247                 else:
248                     tau = belem / alem
249                     C = 1 / np.sqrt(1 + tau ** 2)
250                     S = C * tau
251                 A[i], A[j] = A[i] * C - A[j] * S, A[j] * C + A[i]
                    ↪ * S
252     return A
253
254
255 # обратный ход метода Гаусса
256 def Gauss_back_step(A, B, N):
257     sol = np.zeros(N)
258     for i in range(N - 1, -1, -1):
259         s = 0
260         if i == N - 1:
261             sol[i] = B[i] / A[i][i]
262         else:
263             for j in range(i + 1, N, 1):
264                 s += A[i][j] * sol[j]
265             sol[i] = (B[i] - s) / A[i][i]
266     return sol
267
268
269 def multA(area1, area2, area3, area4):
270     # initialization mult
271     mult_area1 = np.copy(area1)
272     mult_area2 = np.copy(area2)
273     mult_area3 = np.copy(area3)
274     mult_area4 = np.copy(area4)
275
276     # first area
277     for i in range(1, mult_area1.shape[0] - 1):

```

```

278         for j in range(1, mult_area1.shape[1] - 1):
279             mult_area1[i][j] = -1 * ((area1[i - 1][j] - 2 *
↪ area1[i][j] + area1[i + 1][j]) / h_x ** 2 + (
280                 area1[i][j - 1] - 2 * area1[i][j] +
↪ area1[i][j + 1]) / h_y ** 2)
281     for j in range(1, mult_area1.shape[1] - 1):
282         mult_area1[mult_area1.shape[0] - 1][j] = -1 * (
283             (area1[mult_area1.shape[0] - 2][j] - 2 *
↪ area1[mult_area1.shape[0] - 1][j] +
↪ area2[1][int((area2.shape[1] - 1) / 4) + j])
↪ / h_y ** 2 + (
284             area1[mult_area1.shape[0] - 1][j - 1] - 2 *
↪ area1[mult_area1.shape[0] - 1][j] +
↪ area1[mult_area1.shape[0] - 1][j + 1]) / h_x
↪ ** 2)

285
286     # second area
287     for i in range(1, mult_area2.shape[0] - 1):
288         for j in range(1, mult_area2.shape[1] - 1):
289             mult_area2[i][j] = -1 * ((area2[i - 1][j] - 2 *
↪ area2[i][j] + area2[i + 1][j]) / h_x ** 2 + (
290                 area2[i][j - 1] - 2 * area2[i][j] +
↪ area2[i][j + 1]) / h_y ** 2)
291     for j in range(int((mult_area2.shape[1] - 1) / 4) + 1,
↪ mult_area2.shape[1] - 1):
292         mult_area2[mult_area2.shape[0] - 1][j] = -1 *
↪ ((area2[mult_area2.shape[0] - 2][j] - 2 *
↪ area2[mult_area2.shape[0] - 1][j] + area3[1][
293             j - int((mult_area2.shape[1] - 1) / 4)]) / h_x ** 2 +
↪ (area2[mult_area2.shape[0] - 1][j - 1] - 2 *
↪ area2[mult_area2.shape[0] - 1][j] +
↪ area2[mult_area2.shape[0] - 1][j + 1]) / h_y **
↪ 2)

294     # third area
295     for i in range(1, mult_area3.shape[0] - 1):
296         for j in range(1, mult_area3.shape[1] - 1):
297             mult_area3[i][j] = -1 * ((area3[i - 1][j] - 2 *
↪ area3[i][j] + area3[i + 1][j]) / h_x ** 2 + (
298                 area3[i][j - 1] - 2 * area3[i][j] +
↪ area3[i][j + 1]) / h_y ** 2)
299     for j in range(int((mult_area3.shape[1] - 1) / 4) + 1,
↪ mult_area3.shape[1] - int((mult_area3.shape[1] - 1) / 4)
↪ - 1):

```

```

300     mult_area3[mult_area3.shape[0] - 1][j] = -1 *
        ↪ ((area3[mult_area3.shape[0] - 2][j] - 2 *
        ↪ area3[mult_area3.shape[0] - 1][j] + area4[1][
301     j - int((mult_area3.shape[1] - 1) / 4)) / h_x ** 2 +
        ↪ (area3[mult_area3.shape[0] - 1][j - 1] - 2 *
        ↪ area3[mult_area3.shape[0] - 1][j] +
        ↪ area3[mult_area3.shape[0] - 1][j + 1]) / h_y **
        ↪ 2)
302     # fourth area
303     for i in range(1, mult_area4.shape[0] - 1):
304         for j in range(1, mult_area4.shape[1] - 1):
305             mult_area4[i][j] = -1 * ((area4[i - 1][j] - 2 *
        ↪ area4[i][j] + area4[i + 1][j]) / h_x ** 2 + (
306             area4[i][j - 1] - 2 * area4[i][j] +
        ↪ area4[i][j + 1]) / h_y ** 2)
307
308     return mult_area1, mult_area2, mult_area3, mult_area4
309
310
311     # calculate scalar product
312     def Scalar(a1, a2, a3, a4, b1, b2, b3, b4):
313         sum = 0
314         for i in range(a1.shape[0]):
315             for j in range(a1.shape[1]):
316                 sum += a1[i, j] * b1[i, j]
317         for i in range(a2.shape[0]):
318             for j in range(a2.shape[1]):
319                 sum += a2[i, j] * b2[i, j]
320         for i in range(a3.shape[0]):
321             for j in range(a3.shape[1]):
322                 sum += a3[i, j] * b3[i, j]
323         for i in range(a4.shape[0]):
324             for j in range(a4.shape[1]):
325                 sum += a4[i, j] * b4[i, j]
326         return sum
327
328
329     # calculate norm on all areas
330     def NNorma(r1, r2, r3, r4):
331         e1 = np.linalg.norm(r1)
332         e2 = np.linalg.norm(r2)
333         e3 = np.linalg.norm(r3)
334         e4 = np.linalg.norm(r4)

```

```

335     return max(e1, e2, e3, e4)
336
337
338 # Проекционный метод IOM(m)
339
340 def IOM_m(m):
341     # count points on axis
342     Nx = int(L_x / h_x) + 1
343     Ny = int(L_y / h_y) + 1
344     # grid
345     x = np.linspace(0, L_x, Nx)
346     y = np.linspace(0, L_y, Ny)
347
348     # initialization right part and initial approximation
349     # 1:
350     Ny_1 = int((Ny - 1) / 4) + 1
351     Nx_1 = int(2 * (Nx - 1) / 5) + 1
352     U_sub_1 = np.zeros((Ny_1, Nx_1))
353     x_init_1 = U_sub_1.copy()
354     # 2:
355     Ny_2 = int((Ny - 1) / 4) + 1
356     Nx_2 = int(4 * (Nx - 1) / 5) + 1
357     U_sub_2 = np.zeros((Ny_2, Nx_2))
358     x_init_2 = U_sub_2.copy()
359     # 3:
360     Ny_3 = int((Ny - 1) / 4) + 1
361     Nx_3 = int(4 * (Nx - 1) / 5) + 1
362     U_sub_3 = np.zeros((Ny_3, Nx_3))
363     x_init_3 = U_sub_3.copy()
364     # 4:
365     Ny_4 = int((Ny - 1) / 4) + 1
366     Nx_4 = int(2 * (Nx - 1) / 5) + 1
367     U_sub_4 = np.zeros((Ny_4, Nx_4))
368     x_init_4 = U_sub_4.copy()
369
370     # filling in the area
371     # First area
372     for i in range(1, Ny_1):
373         for j in range(1, Nx_1 - 1):
374             U_sub_1[i][j] = func_f(x[int((Nx - 1) / 5) + j], y[Ny
375                                     ↪ - 1 - i])

```

```

376         U_sub_1[0][j] = func_u(x[int((Nx - 1) / 5) + j], y[Ny -
    ↪ 1])
377         x_init_1[0][j] = U_sub_1[0][j].copy()
378     for i in range(Ny_1):
379         U_sub_1[i][0] = func_u(x[int((Nx - 1) / 5)], y[Ny - 1 -
    ↪ i])
380         x_init_1[i][0] = U_sub_1[i][0].copy()
381     for i in range(Ny_1):
382         U_sub_1[i][Nx_1 - 1] = func_u(x[int(3 * (Nx - 1) / 5)],
    ↪ y[Ny - 1 - i])
383         x_init_1[i][Nx_1 - 1] = U_sub_1[i][Nx_1 - 1].copy()
384
385     # Second area
386     for i in range(Ny_2):
387         for j in range(1, Nx_2):
388             U_sub_2[i][j] = func_f(x[j], y[int(3 * (Ny - 1) / 4)
    ↪ - i])
389     for j in range(int((Nx - 1) / 5) + 1):
390         U_sub_2[0][j] = func_u(x[j], y[int(3 * (Ny - 1) / 4)])
391         x_init_2[0][j] = U_sub_2[0][j].copy()
392     for i in range(Ny_2):
393         U_sub_2[i][0] = func_u(x[0], y[int(3 * (Ny - 1) / 4) -
    ↪ i])
394         x_init_2[i][0] = U_sub_2[i][0].copy()
395     for j in range(int((Nx - 1) / 5) + 1):
396         U_sub_2[Ny_2 - 1][j] = func_u(x[j], y[int(2 * (Ny - 1) /
    ↪ 4)])
397         x_init_2[Ny_2 - 1][j] = U_sub_2[Ny_2 - 1][j].copy()
398     for j in range(int((Nx - 1) / 5) + 1):
399         U_sub_2[0][j + int(3 * (Nx_2 - 1) / 4)] = func_u(x[int(3
    ↪ * (Nx - 1) / 5) + j], y[int(3 * (Ny - 1) / 4)])
400         x_init_2[0][j + int(3 * (Nx_2 - 1) / 4)] = U_sub_2[0][j +
    ↪ int(3 * (Nx_2 - 1) / 4)].copy()
401     for i in range(int(Ny_2)):
402         U_sub_2[i][Nx_2 - 1] = func_u(x[int(4 * (Nx - 1) / 5)],
    ↪ y[int(3 * (Ny - 1) / 4) - i])
403         x_init_2[i][Nx_2 - 1] = U_sub_2[i][Nx_2 - 1].copy()
404
405     # Third area
406     for i in range(Ny_3):
407         for j in range(1, Nx_3 - 1):
408             U_sub_3[i][j] = func_f(x[j + int((Nx - 1) / 5)],
    ↪ y[int(2 * (Ny - 1) / 4) - i])

```

```

409     for i in range(Ny_3):
410         U_sub_3[i][0] = func_u(x[int((Nx - 1) / 5)], y[int(2 *
            ↪ (Ny - 1) / 4) - i])
411         x_init_3[i][0] = U_sub_3[i][0].copy()
412     for j in range(Ny_3):
413         U_sub_3[Ny_3 - 1][j] = func_u(x[int((Nx - 1) / 5) + j],
            ↪ y[int((Ny - 1) / 4)])
414         x_init_3[Ny_3 - 1][j] = U_sub_3[Ny_3 - 1][j].copy()
415     for j in range(Ny_3):
416         U_sub_3[Ny_3 - 1][j + int(3 * (Nx_3 - 1) / 4)] =
            ↪ func_u(x[int(4 * (Nx - 1) / 5) + j], y[int((Ny - 1) /
            ↪ 4)])
417         x_init_3[Ny_3 - 1][j + int(3 * (Nx_3 - 1) / 4)] =
            ↪ U_sub_3[Ny_3 - 1][j + int(3 * (Nx_3 - 1) / 4)].copy()
418     for i in range(Ny_3):
419         U_sub_3[i][Nx_3 - 1] = func_u(x[Nx - 1], y[int(2 * (Ny -
            ↪ 1) / 4) - i])
420         x_init_3[i][Nx_3 - 1] = U_sub_3[i][Nx_3 - 1].copy()
421     for j in range(Ny_3):
422         U_sub_3[0][j + int(3 * (Nx_3 - 1) / 4)] = func_u(x[int(4
            ↪ * (Nx - 1) / 5) + j], y[int(2 * (Ny - 1) / 4)])
423         x_init_3[0][j + int(3 * (Nx_3 - 1) / 4)] = U_sub_3[0][j +
            ↪ int(3 * (Nx_3 - 1) / 4)].copy()
424
425     # fourth area
426     for i in range(Ny_4):
427         for j in range(1, Nx_4 - 1):
428             U_sub_4[i][j] = func_f(x[int(2 * (Nx - 1) / 5) + j],
            ↪ y[int((Ny - 1) / 4) - i])
429     for i in range(Ny_4):
430         U_sub_4[i][0] = func_u(x[int(2 * (Nx - 1) / 5)],
            ↪ y[int((Ny - 1) / 4) - i])
431         x_init_4[i][0] = U_sub_4[i][0].copy()
432     for j in range(Nx_4):
433         U_sub_4[Ny_4 - 1][j] = func_u(x[int(2 * (Nx - 1) / 5) +
            ↪ j], y[0])
434         x_init_4[Ny_4 - 1][j] = U_sub_4[Ny_4 - 1][j].copy()
435     for i in range(Ny_4):
436         U_sub_4[i][Nx_4 - 1] = func_u(x[int(4 * (Nx - 1) / 5)],
            ↪ y[int((Ny - 1) / 4) - i])
437         x_init_4[i][Nx_4 - 1] = U_sub_4[i][Nx_4 - 1].copy()
438

```

```

439 k = 2 # количество векторов, к которым будет ортогонален
      ↪ очередной вектор
440
441 mu1, mu2, mu3, mu4 = multA(x_init_1, x_init_2, x_init_3,
      ↪ x_init_4)
442
443 # векторы начальных невязок
444 r01 = U_sub_1 - mu1
445 r02 = U_sub_2 - mu2
446 r03 = U_sub_3 - mu3
447 r04 = U_sub_4 - mu4
448
449 iter = 0 # count iteration
450 while NNorma(r01, r02, r03, r04) > eps:
451
452     print(NNorma(r01, r02, r03, r04))
453
454     # матрицы базисных векторов из пространства K
455     V1 = np.zeros((Ny_1, (m + 1) * Nx_1))
456     V2 = np.zeros((Ny_2, (m + 1) * Nx_2))
457     V3 = np.zeros((Ny_3, (m + 1) * Nx_3))
458     V4 = np.zeros((Ny_4, (m + 1) * Nx_4))
459
460     # матрица коэффициентов ортогонализации
461     H = np.zeros((m + 1, m))
462
463     mu1, mu2, mu3, mu4 = multA(x_init_1, x_init_2, x_init_3,
      ↪ x_init_4)
464
465     # векторы начальных невязок
466     r01 = U_sub_1 - mu1
467     r02 = U_sub_2 - mu2
468     r03 = U_sub_3 - mu3
469     r04 = U_sub_4 - mu4
470
471     # нормы начальных невязок
472     beta = NNorma(r01, r02, r03, r04)
473
474     # первые базисные вектора пространства K
475     V1[:, :Nx_1] = r01 / beta
476     V2[:, :Nx_2] = r02 / beta
477     V3[:, :Nx_3] = r03 / beta
478     V4[:, :Nx_4] = r04 / beta

```

```

479
480     for j in range(1, m + 1):
481         # базисные вектора пространства L
482         omega_j1, omega_j2, omega_j3, omega_j4 = multA(V1[:,
483             ↪ (j - 1) * Nx_1: j * Nx_1], V2[:, (j - 1) * Nx_2:
484             ↪ j * Nx_2],
485                 V3[:, (j - 1) * Nx_3: j * Nx_3],
486                 V4[:, (j - 1) * Nx_4: j * Nx_4])
487     for i in range(max(1, j - k + 1), j + 1):
488         # вычисление коэффициентов орт-ции
489         H[i - 1][j - 1] = Scalar(omega_j1, omega_j2,
490             ↪ omega_j3, omega_j4,
491             V1[:, (j - 1) * Nx_1: j
492             ↪ * Nx_1],
493             V2[:, (j - 1) * Nx_2: j
494             ↪ * Nx_2],
495             V3[:, (j - 1) * Nx_3: j
496             ↪ * Nx_3],
497             V4[:, (j - 1) * Nx_4: j
498             ↪ * Nx_4])
499
500         # ортогонализация очередных базисных векторов
501         ↪ про-ва L
502         omega_j1 = omega_j1 - H[i - 1][j - 1] * V1[:, (i
503             ↪ - 1) * Nx_1: i * Nx_1]
504         omega_j2 = omega_j2 - H[i - 1][j - 1] * V2[:, (i
505             ↪ - 1) * Nx_2: i * Nx_2]
506         omega_j3 = omega_j3 - H[i - 1][j - 1] * V3[:, (i
507             ↪ - 1) * Nx_3: i * Nx_3]
508         omega_j4 = omega_j4 - H[i - 1][j - 1] * V4[:, (i
509             ↪ - 1) * Nx_4: i * Nx_4]
510
511     # норма орт-го вектора
512     H[j][j - 1] = NNorma(omega_j1, omega_j2, omega_j3,
513         ↪ omega_j4)
514
515     if abs(H[j][j - 1]) < 10 ** (-8):
516         m = j
517         break
518
519     # вычисление следующих векторов про-ва K
520     V1[:, j * Nx_1: (j + 1) * Nx_1] = omega_j1 / H[j][j -
521     ↪ 1]

```



```

508         V2[:, j * Nx_2: (j + 1) * Nx_2] = omega_j2 / H[j][j -
        ↪ 1]
509         V3[:, j * Nx_3: (j + 1) * Nx_3] = omega_j3 / H[j][j -
        ↪ 1]
510         V4[:, j * Nx_4: (j + 1) * Nx_4] = omega_j4 / H[j][j -
        ↪ 1]

511
512     e_1 = np.zeros(m + 1) # орт
513     e_1[0] = 1
514     g = beta * e_1 # вектор правой части вспомогательной
        ↪ СЛАУ
515     H = np.c_[H, g] # добавление к матрице системы правой
        ↪ части
516     H = givens(H, m + 1) # зануляем поддиагональ вращениями
        ↪ Гивенса
517     g = H[:m, m] # перезаписываем измененную правую часть
518     H = np.delete(np.delete(H, m, 1), m, 0) # удаляем вектор
        ↪ правой части из системы
519     y = Gauss_back_step(H, g, m) # обратный ход метода
        ↪ Гауса

520
521     # Уточнение решения
522     sumyivi1 = np.zeros((Ny_1, Nx_1)) # уточняющий вектор
523     sumyivi2 = np.zeros((Ny_2, Nx_2)) # уточняющий вектор
524     sumyivi3 = np.zeros((Ny_3, Nx_3)) # уточняющий вектор
525     sumyivi4 = np.zeros((Ny_4, Nx_4)) # уточняющий вектор
526
527     for f in range(1, m + 1):
528         sumyivi1 += y[f - 1] * V1[:, (f - 1) * Nx_1: f *
        ↪ Nx_1] # вычисление уточняющего вектора
529         sumyivi2 += y[f - 1] * V2[:, (f - 1) * Nx_2: f *
        ↪ Nx_2] # вычисление уточняющего вектора
530         sumyivi3 += y[f - 1] * V3[:, (f - 1) * Nx_3: f *
        ↪ Nx_3] # вычисление уточняющего вектора
531         sumyivi4 += y[f - 1] * V4[:, (f - 1) * Nx_4: f *
        ↪ Nx_4] # вычисление уточняющего вектора

532
533     solution1 = x_init_1 + sumyivi1 # уточнение
534     solution2 = x_init_2 + sumyivi2 # уточнение
535     solution3 = x_init_3 + sumyivi3 # уточнение
536     solution4 = x_init_4 + sumyivi4 # уточнение
537

```

```

538     musol1, musol2, musol3, musol4 = multA(solution1,
        ↪     solution2, solution3, solution4)
539     r01 = U_sub_1 - musol1 # вычисление вектора невязки
540     r02 = U_sub_2 - musol2 # вычисление вектора невязки
541     r03 = U_sub_3 - musol3 # вычисление вектора невязки
542     r04 = U_sub_4 - musol4 # вычисление вектора невязки
543
544     # change initial solve
545     x_init_1 = solution1.copy()
546     x_init_2 = solution2.copy()
547     x_init_3 = solution3.copy()
548     x_init_4 = solution4.copy()
549     iter += 1
550
551     return NNorma(r01, r02, r03, r04), iter
552
553
554     print("*****IOM(m)*****")
555     # start time
556     ts = time.time()
557     nev, tr = IOM_m(30)
558     # finish time
559     tf = time.time()
560     print("Невязка = ", nev)
561     print("Итераций = ", tr)
562     print('Время = ', tf - ts)

```
