

Содержание

1	Часть А	2
1.1	Алгоритм ИОМ(m)	2
1.2	Реализация ИОМ(m) на Python	2
1.3	Решение системы проекционным методом	4
2	Часть Б	4
2.1	Реализация ИОМ(m) на Python. Матрично-векторный подход. .	4
2.2	Вычислительные эксперименты	7
2.2.1	Тесты. Параболоид.	7
2.2.2	Тесты. Квадратично-линейная функция.	8
2.2.3	Тесты. Индивидуальная функция.	8
3	Приложение	9

1 Часть А

Реализуйте указанный ниже проекционный метод решения СЛАУ и найдите решение системы $Ax = b$ из ЛР1. При этом матрица A должна храниться в памяти ЭВМ (и задаваться, например, в форме массива в программе).

1.1 Алгоритм ИОМ(m)

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.

1.2 Реализация ИОМ(m) на Python

```
1 def IOM_m(matr_A, vec_b, m):
2     k = 1 # количество векторов, к которым будет ортогонален
           ↪ очередной вектор
3     x0 = np.zeros(N) # Начальное приближение
4     r0 = vec_b - np.dot(matr_A, x0) # вектор начальной невязки
5     while abs(LinAl.norm(r0)) > 10 ** (-12):
6         V = np.zeros((N, m + 1)) # матрица базисных векторов из
           ↪ пространства K
7         H = np.zeros((m + 1, m)) # матрица коэффициентов
           ↪ ортогонализации
8         r0 = vec_b - np.dot(matr_A, x0)
9         beta = LinAl.norm(r0) # норма начальной невязки
```

```

10 V[:, 0] = r0 / beta # первый базисный вектор
    ↪ пространства K
11 for j in range(1, m + 1):
12     omega_j = np.dot(A, V[:, j - 1]) # базисный вектор
    ↪ пространства L
13     for i in range(max(1, j - k + 1), j + 1):
14         H[i - 1][j - 1] = np.dot(np.transpose(omega_j),
    ↪ V[:, i - 1]) # вычисление коэффициента
    ↪ орт-ции
15         omega_j = omega_j - H[i - 1][j - 1] * V[:, i - 1]
    ↪ # орт-ция очередного базисного вектора про-ва
    ↪ L
16     H[j][j - 1] = LinAl.norm(omega_j) # норма орт-го
    ↪ вектора
17     if abs(H[j][j - 1]) < 10 ** (-8):
18         m = j
19         break
20     V[:, j] = omega_j / H[j][j - 1] # вычисление
    ↪ следующего вектора про-ва K
21 e_1 = np.zeros(m + 1) # орт
22 e_1[0] = 1
23 g = beta * e_1 # вектор правой части вспомогательной
    ↪ СЛАУ
24 H = np.c_[H, g] # добавление к матрице системы правой
    ↪ части
25 H = givens(H, m + 1) # зануляем поддиагональ вращениями
    ↪ Гивенса
26 g = H[:, m] # перезаписываем измененную правую часть
27 H = np.delete(np.delete(H, m, 1), m, 0) # удаляем вектор
    ↪ правой части из системы
28 y = Gauss_back_step(H, g, m) # обратный ход метода
    ↪ Гауса
29 # Уточнение решения
30 sumyivi = np.zeros(N) # уточняющий вектор
31 for i in range(m):
32     sumyivi += y[i] * V[:, i] # вычисление уточняющего
    ↪ вектора
33 solution = x0 + sumyivi # уточнение
34 r0 = vec_b - np.dot(A, solution) # вычисление вектора
    ↪ начальной невязки
35 x0 = solution # изменение начального приближения
36 return solution

```

1.3 Решение системы проекционным методом

$$A_{4 \times 4} = \begin{pmatrix} 4 & 1 & 0.5 & 0 \\ 1 & 4 & 1 & 0.5 \\ 0.5 & 1 & 4 & 1 \\ 0 & 0.5 & 1 & 4 \end{pmatrix}, x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}, b = \begin{pmatrix} 4 \\ 1.5 \\ 1.5 \\ 4 \end{pmatrix} \quad (1)$$

$x = [1.000000000e+00 \ 5.85306487e-15 \ 5.83677963e-15 \ 1.000000000e+00]$
 $\|r\|_2 = 1.4023659425114338e-13$

2 Часть Б

В реализованной процедуре проекционного метода измените реализацию матрично-векторного умножения на случай матрицы системы (*) (см. приложение).

Для задачи (*) (см. приложение) необходимо иметь возможность задавать произвольные размеры прямоугольной области Ω и разные (не обязательно равные) шаги h_x и h_y . Функции $f(x,y)$ и $g(x,y)$ следует задавать в коде программы (и изменять с последующей перекомпиляцией).

Расчет следует провести для СЛАУ как можно большего порядка и предусмотреть фиксацию времени, затраченного непосредственно на решение системы.

Решение $\{u_{ij}\}$ визуализируйте в виде двумерного поля температур.

2.1 Реализация ИОМ(m) на Python. Матрично-векторный подход.

```
1  # произведение матрицы системы A на произвольный массив p
2  def multA(p):
3      Ap = np.copy(p)
4      for i in range(1, Ap.shape[0] - 1):
5          for j in range(1, Ap.shape[1] - 1):
6              Ap[i][j] = -1 / Pe * ((p[i - 1][j] - 2 * p[i][j] +
              ↪ p[i + 1][j]) / h_x ** 2 + (
7                  p[i][j - 1] - 2 * p[i][j] + p[i][j + 1]) /
              ↪ h_y ** 2)
8              if v1_func(i, j) > 0:
9                  Ap[i][j] += v1_func(i, j) * (p[i][j] - p[i -
              ↪ 1][j]) / h_x
10             else:
```

```

11         Ap[i][j] += v1_func(i, j) * (p[i + 1][j] -
    ↪      p[i][j]) / h_x
12     if v2_func(i, j) > 0:
13         Ap[i][j] += v2_func(i, j) * (p[i][j] - p[i][j -
    ↪      1]) / h_y
14     else:
15         Ap[i][j] += v2_func(i, j) * (p[i][j + 1] -
    ↪      p[i][j]) / h_y
16     return Ap
17
18
19 # скалярное произведение вектор-матриц
20 def Scr(a, b):
21     sum = 0
22     for i in range(a.shape[0]):
23         for j in range(a.shape[1]):
24             sum += a[i][j] * b[i][j]
25     return sum
26
27
28 # задание правой части
29 B2 = np.zeros((Nx + 1, Ny + 1))
30 for i in range(Ny + 1):
31     B2[0][i] = g_func(0, i)
32     B2[Nx][i] = g_func(Nx, i)
33 for i in range(Nx + 1):
34     B2[i][0] = g_func(i, 0)
35     B2[i][Ny] = g_func(i, Ny)
36 for i in range(1, Nx):
37     for j in range(1, Ny):
38         B2[i][j] = f_func(i, j)
39
40
41 def IOM_m(vec_b, m):
42     solution = np.zeros((Nx + 1, Ny + 1))
43     k = 1 # количество векторов, к которым будет ортогонален
    ↪ очередной вектор
44     x0 = np.zeros((Nx + 1, Ny + 1)) # Начальное приближение
    ↪ # задание краевых значений
45     for ik in range(Ny + 1):
46         x0[0][ik] = g_func(0, ik)
47         x0[Nx][ik] = g_func(Nx, ik)
48     for jk in range(Nx + 1):

```

```

50     x0[jk][0] = g_func(jk, 0)
51     x0[jk][Ny] = g_func(jk, Ny)
52     r0 = vec_b - multA(x0)  # вектор начальной невязки
53     while abs(LA.norm(r0)) > 10 ** (-8):
54         V = np.zeros((Nx + 1, (m + 1) * (Ny + 1)))  # матрица
55         ↪ базисных векторов из пространства K
56         H = np.zeros((m + 1, m))  # матрица коэффициентов
57         ↪ ортогонализации
58         r0 = vec_b - multA(x0)  # вектор начальной невязки
59         beta = LA.norm(r0)  # норма начальной невязки
60         V[:, :Ny + 1] = r0 / beta  # первый базисный вектор
61         ↪ пространства K
62         for j in range(1, m + 1):
63             omega_j = multA(V[:, (j - 1) * (Ny + 1): j * (Ny + 1)])
64             ↪ 1)])  # базисный вектор пространства L
65             for i in range(max(1, j - k + 1), j + 1):
66                 H[i - 1][j - 1] = Scr(omega_j,
67                                     V[:, (i - 1) * (Ny + 1): i * (Ny + 1)])
68                 ↪ * (Ny + 1)])  #
69                 ↪ вычисление коэффициента
70                 ↪ орт-ции
71                 omega_j = omega_j - H[i - 1][j - 1] * V[:, (i - 1) * (Ny + 1): i * (Ny + 1)]
72                 ↪ 1) * (Ny + 1): i * (Ny + 1)]  # орт-ция очередного
73                 ↪ базисного вектора про-ва L
74                 H[j][j - 1] = LA.norm(omega_j)  # норма орт-го
75                 ↪ вектора
76                 if abs(H[j][j - 1]) < 10 ** (-8):
77                     m = j
78                     break
79                 V[:, j * (Ny + 1): (j + 1) * (Ny + 1)] = omega_j /
80                 ↪ H[j][j - 1]  # вычисление следующего вектора
81                 ↪ про-ва K
82             e_1 = np.zeros(m + 1)  # орт
83             e_1[0] = 1
84             g = beta * e_1  # вектор правой части вспомогательной
85             ↪ СЛАУ
86             H = np.c_[H, g]  # добавление к матрице системы правой
87             ↪ части
88             H = givens(H, m + 1)  # зануляем поддиагональ вращениями
89             ↪ Гивенса
90             g = H[:, m, m]  # перезаписываем измененную правую часть

```

```

77     H = np.delete(np.delete(H, m, 1), m, 0)  # удаляем вектор
      ↪ правой части из системы
78     y = Gauss_back_step(H, g, m)  # обратный ход метода
      ↪ Гауса
79     # Уточнение решения
80     sumyivi = np.zeros((Nx + 1, Ny + 1))  # уточняющий
      ↪ вектор
81     for f in range(1, m + 1):
82         sumyivi += y[f - 1] * V[:, (f - 1) * (Ny + 1): f *
      ↪ (Ny + 1)]  # вычисление уточняющего вектора
83     solution = x0 + sumyivi  # уточнение
84     r0 = vec_b - multA(solution)  # вычисление вектора
      ↪ начальной невязки
85     x0 = solution  # изменение начального приближения
86     return solution, r0

```

2.2 Вычислительные эксперименты

2.2.1 Тесты. Параболоид.

$$h_x = 0.1, h_y = 0.1$$

$$Nx = 20, Ny = 20$$

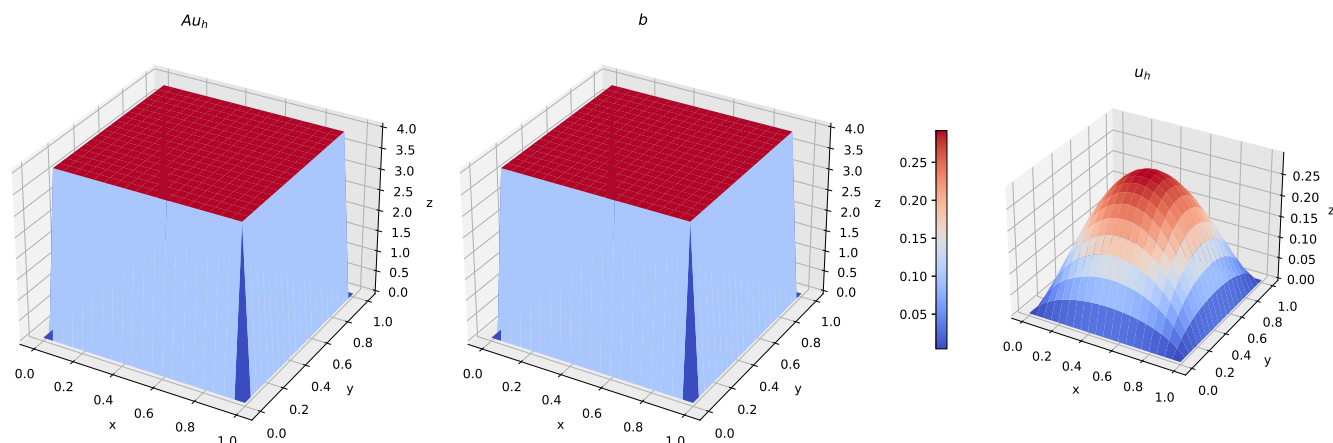
$$Pe = 1$$

$$u = 1 - (x - 1)^2 - (y - 2)^2$$

$$g = 0$$

$$f = 4$$

$$v_1 = 0, v_2 = 0$$



$$\|r\|_2 \approx 9.23 \cdot 10^{-9}$$

$$time \approx 2.22$$

2.2.2 Тесты. Квадратично-линейная функция.

$$h_x = 0.1, h_y = 0.05$$

$$Nx = 50, Ny = 40$$

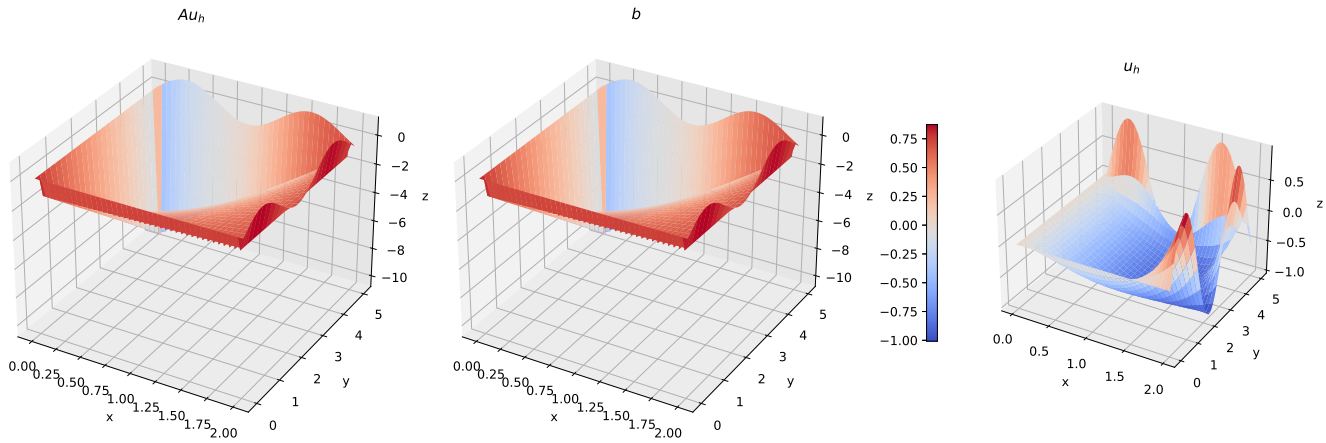
$$Pe = 1$$

$$u = \frac{1}{2}x^2 - \frac{1}{2}y$$

$$g = \sin(xy)$$

$$f = -1 + xy - 2x$$

$$v_1 = y - 1, v_2 = 2x$$



$$\|r\|_2 \approx 8.8 \cdot 10^{-9}$$

$$time \approx 24.73$$

2.2.3 Тесты. Индивидуальная функция.

$$h_x = 0.1, h_y = 0.05$$

$$Nx = 50, Ny = 50$$

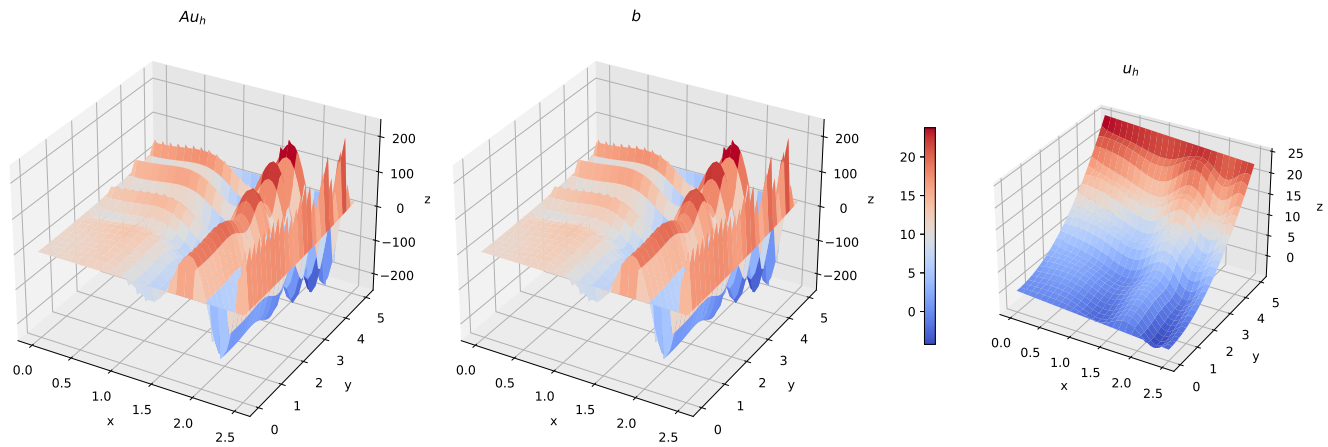
$$Pe = 1$$

$$u = \sin(x^2) + 2 \cos(2y^2)$$

$$g = x^2 - y$$

$$f = 4x^2 \sin(x^2) - 2 \cos(x^2) + 8 \sin(2y^2) + 32y^2 \cos(2y^2)$$

$$v_1 = 4y \sin(2y^2), v_2 = x \cos(x^2)$$



$$\|r\|_2 \approx 8.92 \cdot 10^{-9}$$

$$time \approx 173.3$$

3 Приложение

Программа к части А.

```

1  import numpy as np
2  import numpy.linalg as LinAl
3  import pandas as pd
4  import matplotlib as plt
5
6  num_list_group = 3  # номер в списке группы
7  N = 4  # размерность системы
8  K = N  # полуширина ленты
9  A = np.zeros((N, N))  # матрица A системы
10 elmin = max(N, 10)  # элемент главной диагонали
11 # формируем матрицу A
12 for i in range(N):
13     for j in range(N):
14         if i == j:
15             A[i][j] = elmin
16         if 0 <= i + j <= N - 1 and 1 <= j <= K:
17             A[i][i + j] = 1 / j
18         if 0 <= i - j <= N - 1 and 1 <= j <= K:
19             A[i][i - j] = 1 / j
20 x = np.zeros(N)  # вектор решения
21 # формируем вектор решения
22 for i in range(N):
23     if i == 0 or i == N - 1 or (i == num_list_group and 1 <=
        ↪ num_list_group < N - 1) or (

```

```

24         i == (N - 1) - 2 * num_list_group and 1 <= (N - 1) -
           ↪ 2 * num_list_group < N - 1):
25         x[i] = 1
26     B = np.zeros(N)    # вектор правой части
27     # формируем вектор правой части
28     B = np.dot(A, x)
29
30
31     def givens(A, N):
32         for l in range(N - 1):
33             for i in range(N - 1, 0 + 1, -1):
34                 j = i - 1
35                 if A[i][l] != 0:
36                     alem = A[j][l]
37                     belem = A[i][l]
38                     if np.abs(belem) > np.abs(alem):
39                         tau = alem / belem
40                         S = 1 / np.sqrt(1 + tau ** 2)
41                         C = S * tau
42                     else:
43                         tau = belem / alem
44                         C = 1 / np.sqrt(1 + tau ** 2)
45                         S = C * tau
46                     A[i], A[j] = A[i] * C - A[j] * S, A[j] * C + A[i]
                       ↪ * S
47         return A
48
49
50     def Gauss_back_step(A, B, N):
51         sol = np.zeros(N)
52         for i in range(N - 1, -1, -1):
53             s = 0
54             if i == N - 1:
55                 sol[i] = B[i] / A[i][i]
56             else:
57                 for j in range(i + 1, N, 1):
58                     s += A[i][j] * sol[j]
59                 sol[i] = (B[i] - s) / A[i][i]
60         return sol
61
62
63     def IOM_m(matr_A, vec_b, m):

```

```

64 k = 1 # количество векторов, к которым будет ортогонален
    ↪ очередной вектор
65 x0 = np.zeros(N) # Начальное приближение
66 r0 = vec_b - np.dot(matr_A, x0) # вектор начальной невязки
67 while abs(LinAl.norm(r0)) > 10 ** (-12):
68     V = np.zeros((N, m + 1)) # матрица базисных векторов из
    ↪ пространства K
69     H = np.zeros((m + 1, m)) # матрица коэффициентов
    ↪ ортогонализации
70     r0 = vec_b - np.dot(matr_A, x0)
71     beta = LinAl.norm(r0) # норма начальной невязки
72     V[:, 0] = r0 / beta # первый базисный вектор
    ↪ пространства K
73     for j in range(1, m + 1):
74         omega_j = np.dot(A, V[:, j - 1]) # базисный вектор
    ↪ пространства L
75         for i in range(max(1, j - k + 1), j + 1):
76             H[i - 1][j - 1] = np.dot(np.transpose(omega_j),
    ↪ V[:, i - 1]) # вычисление коэффициента
    ↪ орт-ции
77             omega_j = omega_j - H[i - 1][j - 1] * V[:, i - 1]
    ↪ # орт-ция очередного базисного вектора про-ва
    ↪ L
78             H[j][j - 1] = LinAl.norm(omega_j) # норма орт-го
    ↪ вектора
79             if abs(H[j][j - 1]) < 10 ** (-8):
80                 m = j
81                 break
82             V[:, j] = omega_j / H[j][j - 1] # вычисление
    ↪ следующего вектора про-ва K
83     e_1 = np.zeros(m + 1) # орт
84     e_1[0] = 1
85     g = beta * e_1 # вектор правой части вспомогательной
    ↪ СЛАУ
86     H = np.c_[H, g] # добавление к матрице системы правой
    ↪ части
87     H = givens(H, m + 1) # зануляем поддиагональ вращениями
    ↪ Гивенса
88     g = H[:m, m] # перезаписываем измененную правую часть
89     H = np.delete(np.delete(H, m, 1), m, 0) # удаляем вектор
    ↪ правой части из системы
90     y = Gauss_back_step(H, g, m) # обратный ход метода
    ↪ Гауса

```

```

91         # Уточнение решения
92         sumyivi = np.zeros(N) # уточняющий вектор
93         for i in range(m):
94             sumyivi += y[i] * V[:, i] # вычисление уточняющего
           ↪ вектора
95         solution = x0 + sumyivi # уточнение
96         r0 = vec_b - np.dot(A, solution) # вычисление вектора
           ↪ начальной невязки
97         x0 = solution # изменение начального приближения
98     return solution, r0
99
100
101 sol, nv = IOM_m(A, B, 2)
102 print('Решение - ', sol)
103 print('Невязка - ', LinAl.norm(nv))

```

Программа к части Б.

```

1  import numpy as np
2  import numpy.linalg as LA
3  import matplotlib.pyplot as plt
4  from matplotlib import cm
5  import time
6
7  h_x = 0.1
8  h_y = 0.1
9  Nx = 40
10 Ny = 40
11 Pe = 1
12
13 # -----
14 # Тест №1 - параболоид
15 # test = 1
16 # Тест №2 - квадратично-линейная функция
17 # test = 2
18 # Тест №3 - Индивидуальная функция
19 test = 3
20 # -----
21
22
23 def u_func(i, j):
24     x = i * h_x

```

```

25     y = j * h_y
26     match test:
27         case 1:
28             return 1 - ((x - 1) ** 2) - ((y - 2) ** 2)
29         case 2:
30             return 0.5 * x ** 2 - 0.5 * y
31         case 3:
32             return np.sin(x ** 2) + 2 * np.cos(2 * y ** 2)
33
34
35 def g_func(i, j):
36     x = i * h_x
37     y = j * h_y
38     match test:
39         case 1:
40             return 0
41         case 2:
42             return np.sin(x * y)
43         case 3:
44             return x ** 2 - y
45
46
47 def f_func(i, j):
48     x = i * h_x
49     y = j * h_y
50     match test:
51         case 1:
52             return 4
53         case 2:
54             return -1 + x * y - 2 * x
55         case 3:
56             return 4 * x ** 2 * np.sin(x ** 2) - 2 * np.cos(x **
57                 ↪ 2) + 8 * np.sin(2 * y ** 2) + 32 * y ** 2 *
58                 ↪ np.cos(
59                 ↪ 2 * y ** 2)
60
61 def v1_func(i, j):
62     x = i * h_x
63     y = j * h_y
64     match test:
65         case 1:
66             return 0

```

```

66         case 2:
67             return y - 1
68         case 3:
69             return 4 * y * np.sin(2 * y ** 2)
70
71
72 def v2_func(i, j):
73     x = i * h_x
74     y = j * h_y
75     match test:
76         case 1:
77             return 0
78         case 2:
79             return 2 * x
80         case 3:
81             return x * np.cos(x ** 2)
82
83
84 def solve_of_task(U):
85     trsol = np.zeros((Nx + 1, Ny + 1))
86     for i in range(1, Nx):
87         for j in range(1, Ny):
88             trsol[i][j] = U(i, j)
89     return trsol
90
91
92 def givens(A, N):
93     for l in range(N - 1):
94         for i in range(N - 1, 0 + 1, -1):
95             j = i - 1
96             if A[i][l] != 0:
97                 alem = A[j][l]
98                 belem = A[i][l]
99                 if np.abs(belem) > np.abs(alem):
100                     tau = alem / belem
101                     S = 1 / np.sqrt(1 + tau ** 2)
102                     C = S * tau
103                 else:
104                     tau = belem / alem
105                     C = 1 / np.sqrt(1 + tau ** 2)
106                     S = C * tau
107                 A[i], A[j] = A[i] * C - A[j] * S, A[j] * C + A[i]

```

```

108     return A
109
110
111 def Gauss_back_step(A, B, N):
112     sol = np.zeros(N)
113     for i in range(N - 1, -1, -1):
114         s = 0
115         if i == N - 1:
116             sol[i] = B[i] / A[i][i]
117         else:
118             for j in range(i + 1, N, 1):
119                 s += A[i][j] * sol[j]
120             sol[i] = (B[i] - s) / A[i][i]
121     return sol
122
123
124 # произведение матрицы системы A на произвольный массив p
125 def multA(p):
126     Ap = np.copy(p)
127     for i in range(1, Ap.shape[0] - 1):
128         for j in range(1, Ap.shape[1] - 1):
129             Ap[i][j] = -1 / Pe * ((p[i - 1][j] - 2 * p[i][j] +
130                 ↪ p[i + 1][j]) / h_x ** 2 + (
131                 p[i][j - 1] - 2 * p[i][j] + p[i][j + 1]) /
132                 ↪ h_y ** 2)
133             if v1_func(i, j) > 0:
134                 Ap[i][j] += v1_func(i, j) * (p[i][j] - p[i -
135                 ↪ 1][j]) / h_x
136             else:
137                 Ap[i][j] += v1_func(i, j) * (p[i + 1][j] -
138                 ↪ p[i][j]) / h_x
139             if v2_func(i, j) > 0:
140                 Ap[i][j] += v2_func(i, j) * (p[i][j] - p[i][j -
141                 ↪ 1]) / h_y
142             else:
143                 Ap[i][j] += v2_func(i, j) * (p[i][j + 1] -
144                 ↪ p[i][j]) / h_y
145     return Ap
146
147
148 # скалярное произведение вектор-матриц
149 def Scr(a, b):
150     sum = 0

```

```

145     for i in range(a.shape[0]):
146         for j in range(a.shape[1]):
147             sum += a[i][j] * b[i][j]
148     return sum
149
150
151     # задание правой части
152     B2 = np.zeros((Nx + 1, Ny + 1))
153     for i in range(Ny + 1):
154         B2[0][i] = g_func(0, i)
155         B2[Nx][i] = g_func(Nx, i)
156     for i in range(Nx + 1):
157         B2[i][0] = g_func(i, 0)
158         B2[i][Ny] = g_func(i, Ny)
159     for i in range(1, Nx):
160         for j in range(1, Ny):
161             B2[i][j] = f_func(i, j)
162
163
164     def IOM_m(vec_b, m):
165         solution = np.zeros((Nx + 1, Ny + 1))
166         k = 1 # количество векторов, к которым будет ортогонален
167             ↪ очередной вектор
168         x0 = np.zeros((Nx + 1, Ny + 1)) # Начальное приближение
169             # задание краевых значений
170         for ik in range(Ny + 1):
171             x0[0][ik] = g_func(0, ik)
172             x0[Nx][ik] = g_func(Nx, ik)
173         for jk in range(Nx + 1):
174             x0[jk][0] = g_func(jk, 0)
175             x0[jk][Ny] = g_func(jk, Ny)
176         r0 = vec_b - multA(x0) # вектор начальной невязки
177         while abs(LA.norm(r0)) > 10 ** (-8):
178             V = np.zeros((Nx + 1, (m + 1) * (Ny + 1))) # матрица
179                 ↪ базисных векторов из пространства K
180             H = np.zeros((m + 1, m)) # матрица коэффициентов
181                 ↪ ортогонализации
182             r0 = vec_b - multA(x0) # вектор начальной невязки
183             beta = LA.norm(r0) # норма начальной невязки
184             V[:, :Ny + 1] = r0 / beta # первый базисный вектор
185                 ↪ пространства K
186             for j in range(1, m + 1):

```



```

183     omega_j = multA(V[:, (j - 1) * (Ny + 1): j * (Ny +
    ↪ 1)]) # базисный вектор пространства L
184     for i in range(max(1, j - k + 1), j + 1):
185         H[i - 1][j - 1] = Scr(omega_j,
186                               V[:, (i - 1) * (Ny + 1): i
    ↪ * (Ny + 1)]) #
    ↪ вычисление коэффициента
    ↪ орт-ции
187         omega_j = omega_j - H[i - 1][j - 1] * V[:, (i -
    ↪ 1) * (Ny + 1): i * (
188             Ny + 1)] # орт-ция очередного
    ↪ базисного вектора про-ва L
189     H[j][j - 1] = LA.norm(omega_j) # норма орт-го
    ↪ вектора
190     if abs(H[j][j - 1]) < 10 ** (-8):
191         m = j
192         break
193     V[:, j * (Ny + 1): (j + 1) * (Ny + 1)] = omega_j /
    ↪ H[j][j - 1] # вычисление следующего вектора
    ↪ про-ва K
194     e_1 = np.zeros(m + 1) # орт
195     e_1[0] = 1
196     g = beta * e_1 # вектор правой части вспомогательной
    ↪ СЛАУ
197     H = np.c_[H, g] # добавление к матрице системы правой
    ↪ части
198     H = givens(H, m + 1) # зануляем поддиагональ вращениями
    ↪ Гивенса
199     g = H[:, m, m] # перезаписываем измененную правую часть
200     H = np.delete(np.delete(H, m, 1), m, 0) # удаляем вектор
    ↪ правой части из системы
201     y = Gauss_back_step(H, g, m) # обратный ход метода
    ↪ Гауса
202     # Уточнение решения
203     sumyivi = np.zeros((Nx + 1, Ny + 1)) # уточняющий
    ↪ вектор
204     for f in range(1, m + 1):
205         sumyivi += y[f - 1] * V[:, (f - 1) * (Ny + 1): f *
    ↪ (Ny + 1)] # вычисление уточняющего вектора
206     solution = x0 + sumyivi # уточнение
207     r0 = vec_b - multA(solution) # вычисление вектора
    ↪ начальной невязки
208     x0 = solution # изменение начального приближения

```

```

209         print(LA.norm(r0))
210     return solution, r0
211
212
213     ts = time.time()
214     lol, nev = IOM_m(B2, 3)
215     tf = time.time()
216     print('Время = ', tf - ts)
217     print('Невязка = ', LA.norm(nev))
218     t = np.linspace(0, Nx * h_x, Nx + 1)
219     y = np.linspace(0, Ny * h_y, Ny + 1)
220     x_grid, y_grid = np.meshgrid(y, t)
221     fig = plt.figure(figsize=(15, 5))
222     ax = plt.subplot(131, projection='3d')
223     ax.set_title(r'$Au_h$')
224     ax.set_xlabel(r'x')
225     ax.set_ylabel(r'y')
226     ax.set_zlabel(r'z')
227     ax1 = plt.subplot(132, projection='3d')
228     ax2 = plt.subplot(133, projection='3d')
229     ax1.set_title(r'$b$')
230     ax1.set_xlabel(r'x')
231     ax1.set_ylabel(r'y')
232     ax1.set_zlabel(r'z')
233     surf = ax.plot_surface(x_grid, y_grid, multA(lol),
234         ↪ cmap=cm.coolwarm,
235         linewidth=0, antialiased=False,
236         ↪ label=r'$\backslash r \backslash $')
237     surf1 = ax1.plot_surface(x_grid, y_grid, B2, cmap=cm.coolwarm,
238         linewidth=0, antialiased=False)
239     ax2.set_title(r'$u_h$')
240     ax2.set_xlabel(r'x')
241     ax2.set_ylabel(r'y')
242     ax2.set_zlabel(r'z')
243     surf2 = ax2.plot_surface(x_grid, y_grid, lol, cmap=cm.coolwarm,
244         linewidth=0, antialiased=False)
245     fig.colorbar(surf2, shrink=0.5, location='left')
246     plt.show()

```

Задача конвекции-диффузии

Рассмотрим классическую задачу конвекции – диффузии

$$\begin{cases} -\frac{1}{Pe} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + v_1 \frac{\partial u}{\partial x} + v_2 \frac{\partial u}{\partial y} = f, \\ u|_{\partial\Omega} = g \end{cases}$$

в прямоугольной области $\Omega = \{(x, y) \in \mathbb{R}^2 \mid 0 < x < X, 0 < y < Y\}$.

Здесь $v = (v_1(x, y), v_2(x, y))^T$ – заданный вектор скорости, Pe – безразмерный параметр (число Пекле). Искомой является температура $u = u(x, y)$.

Для численного решения задачи заменим область непрерывного изменения аргументов Ω конечномерной сеткой, а производные, входящие в уравнение, их разностными аппроксимациями. В результате получим систему сеточных уравнений

$$\begin{aligned} & -\frac{1}{Pe} \left(\frac{u_{i-1j} - 2u_{ij} + u_{i+1j}}{h_x^2} + \frac{u_{ij-1} - 2u_{ij} + u_{ij+1}}{h_y^2} \right) + \\ & + v_{1ij} D_x u_{ij} + v_{2ij} D_y u_{ij} = f_{ij}, \quad i = 1, \dots, N_x - 1, \quad j = 1, \dots, N_y - 1. \quad (*) \end{aligned}$$

Для аппроксимации производных первого порядка будем использовать простейший вариант разностей против потока:

$$\begin{aligned} D_x u_{ij} &= \begin{cases} \frac{u_{ij} - u_{i-1j}}{h_x} & \text{если } v_{1ij} > 0, \\ \frac{u_{i+1j} - u_{ij}}{h_x} & \text{если } v_{1ij} < 0, \end{cases} \\ D_y u_{ij} &= \begin{cases} \frac{u_{ij} - u_{ij-1}}{h_y} & \text{если } v_{2ij} > 0, \\ \frac{u_{ij+1} - u_{ij}}{h_y} & \text{если } v_{2ij} < 0. \end{cases} \end{aligned}$$

Система (*) является системой линейных алгебраических уравнений вида

$$Au = b \quad (**)$$

относительно неизвестных u_{ij} .