

Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Численные методы»

Студент: А. А. Каримов
Преподаватель: Д. В. Беляков
Группа: М8О-406Б-22
Дата:
Оценка:
Подпись:

Москва, 2025

1 Формулировка задачи №7

Задача: Решить краевую задачу для дифференциального уравнения эллиптического типа. Аппроксимацию уравнения произвести с использованием центрально-разностной схемы. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x, y)$. Исследовать зависимость погрешности от сеточных параметров h_x, h_y .

0.1 Вариант 8

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2 \cdot \frac{\partial u}{\partial x} - 2 \cdot \frac{\partial u}{\partial y} - 4 \cdot u$$

$$u(0, y) = e^{-y} \cos y$$

$$u\left(\frac{\pi}{2}, y\right) = 0$$

$$u(x, 0) = e^{-x} \cdot \cos x$$

$$u\left(x, \frac{\pi}{2}\right) = 0$$

Аналитическое решение:

$$U(x, y) = e^{-x-y} \cdot \cos x \cdot \cos y$$

2 Теория

МЕТОД КОНЕЧНЫХ РАЗНОСТЕЙ ДЛЯ РЕШЕНИЯ УРАВНЕНИЙ ЭЛЛИПТИЧЕСКОГО ТИПА

Постановка задач для уравнений эллиптического типа

Классическим примером уравнения эллиптического типа является уравнение Пуассона

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (1)$$

или уравнение Лапласа при $f(x, y) \equiv 0$.

Здесь функция $u(x, y)$ имеет различный физический смысл, а именно: стационарное, независящее от времени, распределение температуры, скорость потенциального (безвихревого) течения идеальной (без трения и теплопроводности) жидкости, распределение напряженностей электрического и магнитного полей, потенциала в силовом поле тяготения и т.п.

Если на границе Γ расчетной области $\bar{\Omega} = \Omega + \Gamma$ задана искомая функция, то соответствующая первая краевая задача для уравнения Лапласа или Пуассона называется **задачей Дирихле**:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \in \Omega; \quad (2)$$

$$u(x, y) = \phi(x, y), \quad (x, y) \in \Gamma. \quad (3)$$

Если на границе Γ задается нормальная производная искомой функции, то соответствующая вторая краевая задача называется **задачей Неймана** для уравнения Лапласа или Пуассона:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \in \Omega; \quad (4)$$

$$\frac{\partial u(x, y)}{\partial n} = \phi(x, y), \quad (x, y) \in \Gamma. \quad (5)$$

При этом n – направление внешней к границе Γ нормали.

Более приемлемой является координатная форма краевого условия (5)

$$\frac{\partial u}{\partial x} \cos(\hat{n}, i) + \frac{\partial u}{\partial y} \cos(\hat{n}, j) = \phi(x, y),$$

где $\cos(\hat{n}, i)$, $\cos(\hat{n}, j)$ – направляющие косинусы внешнего вектора единичной нормали к границе Γ , i и j – орты базисных векторов.

Наконец **третья краевая задача** для уравнения Пуассона (Лапласа) имеет вид:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \in \Omega; \quad (6)$$

$$\frac{\partial u}{\partial n} + \alpha u = \phi(x, y), \quad (x, y) \in \Gamma. \quad (7)$$

Конечно-разностная аппроксимация задач для уравнений эллиптического типа

Рассмотрим краевую задачу для уравнений Лапласа или Пуассона (2), (3) в прямоугольнике $x \in [0, l_1]$, $y \in [0, l_2]$, на который наложим сетку:

$$\omega_{h_1 h_2} = \{(x_i, y_j) : x_i = ih_1, i = \overline{0, N_1}; y_j = jh_2, j = \overline{0, N_2}\}. \quad (8)$$

На этой сетке аппроксимируем дифференциальную задачу во внутренних узлах с помощью отношения конечных разностей по следующей схеме (вводится сеточная функция $u_{i,j}$, $i = \overline{0, N_1}$, $j = \overline{0, N_2}$):

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_1^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_2^2} = f(x_i, y_j) + O(h_1^2 + h_2^2), \quad (9)$$

$$i = \overline{1, N_1 - 1}, j = \overline{1, N_2 - 1},$$

которая на шаблоне имеет второй порядок по переменным x и y , поскольку шаблон центрально симметричен.

СЛАУ (9) имеет пятидиагональный вид (каждое уравнение содержит пять неизвестных и при соответствующей нумерации переменных матрица имеет ленточную структуру). Решать ее можно различными методами линейной алгебры, например, итерационными методами, методом матричной прогонки и т.п.

Рассмотрим разностно-итерационный метод Либмана численного решения задачи Дирихле (2), (3). Для простоты изложения этого метода примем $h = h_1 = h_2$, тогда из схемы (9) получим (k -номер итерации):

$$u_{i,j}^{(k+1)} = \frac{1}{4} \left[u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)} - h^2 f_{i,j} \right], \quad (10)$$

$$i = \overline{1, N_1 - 1}, j = \overline{1, N_2 - 1},$$

где $f_{i,j} = f(x_i, y_j)$.

На каждой координатной линии (например, $y_j = \text{const}$, $j = \overline{1, N_2 - 1}$) с помощью линейной интерполяции граничных значений $\phi(x, y)$ определим $u_{i,j}^{(0)}$, подставив которые в (10), получим распределение $u_{i,j}^{(1)}$ на первой итерации:

$$u_{i,j}^{(1)} = \frac{1}{4} \left[u_{i+1,j}^{(0)} + u_{i-1,j}^{(0)} + u_{i,j+1}^{(0)} + u_{i,j-1}^{(0)} - h^2 f_{i,j} \right].$$

Это распределение снова подставляется в (10), получаем распределение $u_{i,j}^{(2)}$ и т.д. Процесс Либмана прекращается, когда

$$\|u^{(k+1)} - u^{(k)}\| \leq \varepsilon, \quad \|u^{(k)}\| = \max_{i,j} |u_{i,j}^{(k)}|,$$

где ε - наперед заданная точность.

3 Исходный код

Здесь располагается реализация задачи.

```

1  # %%
2  import matplotlib.cm
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import random
6
7  # %%
8  EPS = 1e-9
9
10 # %% [markdown]
11 #
12
13 # %%
14 omega = 1.5
15
16 # %% [markdown]
17 #
18
19 # %%
20 Lx = np.pi / 2
21 Ly = np.pi / 2
22 n = 35
23 m = 35
24
25 # %% [markdown]
26 #
27
28 # %%
29 a = 1
30 b = 0
31 c = 1
32 d = 2
33 e = 2
34 f = 4
35 g = 0
36
37 # %% [markdown]
```

```

38 | #
39 |
40 | # %%
41 | alpha_0y = 0
42 | beta_0y = 1
43 |
44 | alpha_Ly = 0
45 | beta_Ly = 1
46 |
47 | alpha_x0 = 0
48 | beta_x0 = 1
49 |
50 | alpha_xL = 0
51 | beta_xL = 1
52 |
53 | # %% [markdown]
54 | #
55 |
56 | # %%
57 | def gamma_0y(y):
58 |     return np.exp(-y) * np.cos(y)
59 |
60 |
61 | def gamma_Ly(y):
62 |     return 0
63 |
64 |
65 | def gamma_x0(x):
66 |     return np.exp(-x) * np.cos(x)
67 |
68 |
69 | def gamma_xL(x):
70 |     return 0
71 |
72 | # %% [markdown]
73 | #
74 |
75 | # %%
76 | def analytical(x, y):
77 |     return np.exp(-x - y) * np.cos(x) * np.cos(y)
78 |
79 | # %% [markdown]
80 | #  $h_x$   $h_y$ 
81 |
82 | # %%
83 | hx = Lx / (n - 1)
84 | hy = Ly / (m - 1)
85 |
86 | hy2 = hy**2

```

```

87 | hx2 = hx**2
88 |
89 | # %%
90 | x = [i * hx for i in range(n - 1)]
91 | x.append(Lx)
92 | x = np.array(x)
93 |
94 | y = [j * hy for j in range(m - 1)]
95 | y.append(Ly)
96 | y = np.array(y)
97 |
98 | # %%
99 | bound_x0 = []
100 | bound_xL = []
101 |
102 | for i in range(n):
103 |     bound_x0.append(gamma_x0(x[i]))
104 |     bound_xL.append(gamma_xL(x[i]))
105 |
106 | bound_0y = []
107 | bound_Ly = []
108 |
109 | for j in range(m):
110 |     bound_0y.append(gamma_0y(y[j]))
111 |     bound_Ly.append(gamma_Ly(y[j]))
112 |
113 | # %%
114 | x_plt, y_plt = np.meshgrid(x, y, indexing='ij')
115 | true_data = analytical(x_plt, y_plt)
116 |
117 | # %%
118 | def get_delta(u, u_next):
119 |     delta = -1
120 |     for i in range(n):
121 |         for j in range(m):
122 |             delta = max(delta, abs(u[i][j] - u_next[i][j]))
123 |     return delta
124 |
125 | # %%
126 | def get_boundary(u_next):
127 |     for i in range(n):
128 |         u_next[i][0] = 0
129 |         u_next[i][m - 1] = 0
130 |
131 |     for j in range(m):
132 |         u_next[0][j] = 0
133 |         u_next[n - 1][j] = 0
134 |
135 |

```

```

136     for i in range(n):
137         u_next[i][0] += (bound_x0[i] - alpha_x0 / hy * u_next[i][1]) / \
138             (beta_x0 - alpha_x0 / hy)
139
140     for i in range(n):
141         u_next[i][m - 1] += (bound_xL[i] - alpha_xL / hy * u_next[i][m - 2]) / \
142             (beta_xL - alpha_xL / hy)
143
144
145     for j in range(m):
146         u_next[0][j] += (bound_0y[j] - alpha_0y / hx * u_next[1][j]) / \
147             (beta_0y - alpha_0y / hx)
148
149     for j in range(m):
150         u_next[n - 1][j] += (bound_Ly[j] + alpha_Ly / hx * u_next[n - 2][j]) / \
151             (beta_Ly + alpha_Ly / hx)
152
153
154     u_next[n - 1][m - 1] /= 2
155     u_next[n - 1][0] /= 2
156     u_next[0][m - 1] /= 2
157     u_next[0][0] /= 2
158
159     return u_next
160
161
162 # %% [markdown]
163 #
164
165 # %%
166 def liebmann_method():
167     delta = 1
168     iter_count = 0
169
170     u = np.ones((n, m)) * 0.1
171
172     while delta > EPS:
173         u_next = np.zeros((n, m))
174
175         for i in range(1, n - 1):
176             for j in range(1, m - 1):
177                 u_next[i][j] = (c / hy2 - e / (2 * hy)) * u[i][j - 1] + \
178                     (c / hy2 + e / (2 * hy)) * u[i][j + 1] + \
179                     (a / hx2 - d / (2 * hx)) * u[i - 1][j] + \
180                     (a / hx2 + d / (2 * hx)) * u[i + 1][j] + g
181                 u_next[i][j] /= (2 * (a / hx2 + c / hy2) - f)
182
183         u_next = get_boundary(u_next)
184         delta = get_delta(u, u_next)

```



```

185         u = u_next
186         iter_count += 1
187
188         # print(f"Iteration counter: {iter_count}")
189         return u
190
191     # %%
192     liebmann_data = liebmann_method().transpose()
193
194     # %%
195     x_plt, y_plt = np.meshgrid(x, y)
196     true_data = analytical(x_plt, y_plt)
197
198     y_indices = random.sample(range(len(y) - 1), k=min(3, len(y)))
199     y_indices.sort()
200
201     fig = plt.figure(figsize=(20, 12))
202
203     for i, y_idx in enumerate(y_indices):
204         ax = fig.add_subplot(2, 3, i+1)
205         ax.plot(x, liebmann_data[y_idx], label="")
206         ax.plot(x, true_data[y_idx], label="")
207         ax.set_title(f"y = {y[y_idx]:.2f}")
208         ax.set_xlabel('x')
209         ax.set_ylabel('u(x,y)')
210         ax.legend()
211         ax.grid(True)
212
213     ax1 = fig.add_subplot(2, 2, 3, projection='3d')
214     ax1.plot_surface(x_plt, y_plt, liebmann_data, cmap='plasma')
215     ax1.set_title(' ')
216     ax1.set_xlabel('x')
217     ax1.set_ylabel('y')
218     ax1.set_zlabel('u(x,y)')
219
220     ax2 = fig.add_subplot(2, 2, 4, projection='3d')
221     ax2.plot_surface(x_plt, y_plt, true_data, cmap='plasma')
222     ax2.set_title(' ')
223     ax2.set_xlabel('x')
224     ax2.set_ylabel('y')
225     ax2.set_zlabel('u(x,y)')
226
227     plt.show()
228
229     # %% [markdown]
230     #
231
232     # %%
233     def seidel_method():

```

```

234     delta = 1
235     iter_count = 0
236
237     u = np.ones((n, m)) * 0.1
238
239     while delta > EPS:
240         u_next = u.copy()
241
242         for i in range(1, n - 1):
243             for j in range(1, m - 1):
244                 u_next[i][j] = (c / hy2 - e / (2 * hy)) * u_next[i][j - 1] + \
245                     (c / hy2 + e / (2 * hy)) * u[i][j + 1] + \
246                     (a / hx2 - d / (2 * hx)) * u_next[i - 1][j] + \
247                     (a / hx2 + d / (2 * hx)) * u[i + 1][j] + g
248                 u_next[i][j] /= (2 * (a / hx2 + c / hy2) - f)
249
250         u_next = get_boundary(u_next)
251         delta = get_delta(u, u_next)
252         u = u_next
253         iter_count += 1
254
255         # print(f"Iteration counter: {iter_count}")
256     return u
257
258 # %%
259 seidel_data = seidel_method().transpose()
260
261 # %%
262 x_plt, y_plt = np.meshgrid(x, y)
263 true_data = analytical(x_plt, y_plt)
264
265 y_indices = random.sample(range(len(y) - 1), k=min(3, len(y)))
266 y_indices.sort()
267
268 fig = plt.figure(figsize=(20, 12))
269
270 for i, y_idx in enumerate(y_indices):
271     ax = fig.add_subplot(2, 3, i+1)
272     ax.plot(x, seidel_data[y_idx], label="")
273     ax.plot(x, true_data[y_idx], label="")
274     ax.set_title(f"y = {y[y_idx]:.2f}")
275     ax.set_xlabel('x')
276     ax.set_ylabel('u(x,y)')
277     ax.legend()
278     ax.grid(True)
279
280 ax1 = fig.add_subplot(2, 2, 3, projection='3d')
281 ax1.plot_surface(x_plt, y_plt, seidel_data, cmap='plasma')
282 ax1.set_title(' ')

```

```

283 ax1.set_xlabel('x')
284 ax1.set_ylabel('y')
285 ax1.set_zlabel('u(x,y)')
286
287 ax2 = fig.add_subplot(2, 2, 4, projection='3d')
288 ax2.plot_surface(x_plt, y_plt, true_data, cmap='plasma')
289 ax2.set_title(' ')
290 ax2.set_xlabel('x')
291 ax2.set_ylabel('y')
292 ax2.set_zlabel('u(x,y)')
293
294 plt.show()
295
296 # %% [markdown]
297 #
298
299 # %%
300 def relaxation_method():
301     omega = 1.0038003800380038
302     delta = 1
303     iter_count = 0
304
305     u = np.ones((n, m)) * 0.1
306
307     while delta > EPS and iter_count < 1e3:
308         u_next = np.zeros((n, m))
309
310         for i in range(1, n - 1):
311             for j in range(1, m - 1):
312                 u_next[i][j] = (c / hy2 - e / (2 * hy)) * u[i][j - 1] + \
313                     (c / hy2 + e / (2 * hy)) * u[i][j + 1] + \
314                     (a / hx2 - d / (2 * hx)) * u[i - 1][j] + \
315                     (a / hx2 + d / (2 * hx)) * u[i + 1][j] + g
316                 u_next[i][j] /= (2 * (a / hx2 + c / hy2) - f)
317
318         u_next = get_boundary(u_next)
319
320         u_next = u_next * omega + (1 - omega) * u
321         delta = get_delta(u, u_next)
322         u = u_next
323         iter_count += 1
324
325         # print(f"Iteration counter: {iter_count}")
326     return u
327
328 # %%
329 relaxation_data = relaxation_method().transpose()
330
331 # %%

```

```

332 x_plt, y_plt = np.meshgrid(x, y)
333 true_data = analytical(x_plt, y_plt)
334
335 y_indices = random.sample(range(len(y) - 1), k=min(3, len(y)))
336 y_indices.sort()
337
338 fig = plt.figure(figsize=(20, 12))
339
340 for i, y_idx in enumerate(y_indices):
341     ax = fig.add_subplot(2, 3, i+1)
342     ax.plot(x, relaxation_data[y_idx], label="")
343     ax.plot(x, true_data[y_idx], label="")
344     ax.set_title(f"y = {y[y_idx]:.2f}")
345     ax.set_xlabel('x')
346     ax.set_ylabel('u(x,y)')
347     ax.legend()
348     ax.grid(True)
349
350 ax1 = fig.add_subplot(2, 2, 3, projection='3d')
351 ax1.plot_surface(x_plt, y_plt, relaxation_data, cmap='plasma')
352 ax1.set_title(' ')
353 ax1.set_xlabel('x')
354 ax1.set_ylabel('y')
355 ax1.set_zlabel('u(x,y)')
356
357 ax2 = fig.add_subplot(2, 2, 4, projection='3d')
358 ax2.plot_surface(x_plt, y_plt, true_data, cmap='plasma')
359 ax2.set_title(' ')
360 ax2.set_xlabel('x')
361 ax2.set_ylabel('y')
362 ax2.set_zlabel('u(x,y)')
363
364 plt.show()
365
366 # %%
367 def seidel_relaxation_method():
368     delta = 1
369     iter_count = 0
370
371     u = np.ones((n, m)) * 0.1
372
373     while delta > EPS:
374         u_next = u.copy()
375
376         for i in range(1, n - 1):
377             for j in range(1, m - 1):
378                 u_next[i][j] = (c / hy2 - e / (2 * hy)) * u_next[i][j - 1] + \
379                     (c / hy2 + e / (2 * hy)) * u[i][j + 1] + \
380                     (a / hx2 - d / (2 * hx)) * u_next[i - 1][j] + \

```

```

381         (a / hx2 + d / (2 * hx)) * u[i + 1][j] + g
382     u_next[i][j] /= (2 * (a / hx2 + c / hy2) - f)
383     u_next[i][j] = u_next[i][j] * omega + (1 - omega) * u[i][j]
384
385     u_next = get_boundary(u_next)
386     delta = get_delta(u, u_next)
387     u = u_next
388     iter_count += 1
389
390     # print(f"Iteration counter: {iter_count}")
391     return u
392
393 # %%
394 seidel_relaxation_data = seidel_relaxation_method().transpose()
395
396 # %%
397 x_plt, y_plt = np.meshgrid(x, y)
398 true_data = analytical(x_plt, y_plt)
399
400 y_indices = random.sample(range(len(y) - 1), k=min(3, len(y)))
401 y_indices.sort()
402
403 fig = plt.figure(figsize=(20, 12))
404
405 for i, y_idx in enumerate(y_indices):
406     ax = fig.add_subplot(2, 3, i+1)
407     ax.plot(x, seidel_relaxation_data[y_idx], label="")
408     ax.plot(x, true_data[y_idx], label="")
409     ax.set_title(f"y = {y[y_idx]:.2f}")
410     ax.set_xlabel('x')
411     ax.set_ylabel('u(x,y)')
412     ax.legend()
413     ax.grid(True)
414
415 ax1 = fig.add_subplot(2, 2, 3, projection='3d')
416 ax1.plot_surface(x_plt, y_plt, seidel_relaxation_data, cmap='plasma')
417 ax1.set_title(' ')
418 ax1.set_xlabel('x')
419 ax1.set_ylabel('y')
420 ax1.set_zlabel('u(x,y)')
421
422 ax2 = fig.add_subplot(2, 2, 4, projection='3d')
423 ax2.plot_surface(x_plt, y_plt, true_data, cmap='plasma')
424 ax2.set_title(' ')
425 ax2.set_xlabel('x')
426 ax2.set_ylabel('y')
427 ax2.set_zlabel('u(x,y)')
428
429 plt.show()

```

```

430
431 # %%
432 def compute_error(numerical_solution, analytical_solution):
433     return np.sqrt(np.mean((numerical_solution - analytical_solution)**2))
434
435 fixed_n = 15
436 m_values = [10, 15, 20, 25, 30, 40]
437
438 errors_liebmman_hy = []
439 errors_seidel_hy = []
440 errors_relaxation_hy = []
441 hy_values = []
442
443
444 hx = Lx / (n - 1)
445 hy = Ly / (m - 1)
446
447 hy2 = hy**2
448 hx2 = hx**2
449
450 for m in m_values:
451     n = fixed_n
452     hy = (Ly - 0) / (m - 1)
453     hx = (Lx - 0) / (n - 1)
454     hx2 = hx * hx
455     hy2 = hy * hy
456
457     x = [i * hx for i in range(n - 1)]
458     x.append(Lx)
459     x = np.array(x)
460
461     y = [j * hy for j in range(m - 1)]
462     y.append(Ly)
463     y = np.array(y)
464
465     bound_x0 = []
466     bound_xL = []
467
468     for i in range(n):
469         bound_x0.append(gamma_x0(x[i]))
470         bound_xL.append(gamma_xL(x[i]))
471
472     bound_0y = []
473     bound_Ly = []
474
475     for j in range(m):
476         bound_0y.append(gamma_0y(y[j]))
477         bound_Ly.append(gamma_Ly(y[j]))
478

```

```

479     x_plt, y_plt = np.meshgrid(x, y, indexing='ij')
480     true_data = analytical(x_plt, y_plt)
481
482     liebmann_sol = liebmann_method()
483     seidel_sol = seidel_method()
484     relaxation_sol = seidel_relaxation_method()
485
486     errors_liebmann_hy.append(compute_error(liebmann_sol, true_data))
487     errors_seidel_hy.append(compute_error(seidel_sol, true_data))
488     errors_relaxation_hy.append(compute_error(relaxation_sol, true_data))
489     hy_values.append(hy)
490
491     fixed_m = 20
492     n_values = [10, 15, 20, 25, 30, 40]
493
494     errors_liebmann_hx = []
495     errors_seidel_hx = []
496     errors_relaxation_hx = []
497     hx_values = []
498
499     for n in n_values:
500         m = fixed_m
501         hy = (Ly - 0) / (m - 1)
502         hx = (Lx - 0) / (n - 1)
503         hx2 = hx * hx
504         hy2 = hy * hy
505
506         x = [i * hx for i in range(n - 1)]
507         x.append(Lx)
508         x = np.array(x)
509
510         y = [j * hy for j in range(m - 1)]
511         y.append(Ly)
512         y = np.array(y)
513
514         bound_x0 = []
515         bound_xL = []
516
517         for i in range(n):
518             bound_x0.append(gamma_x0(x[i]))
519             bound_xL.append(gamma_xL(x[i]))
520
521         bound_0y = []
522         bound_Ly = []
523
524         for j in range(m):
525             bound_0y.append(gamma_0y(y[j]))
526             bound_Ly.append(gamma_Ly(y[j]))
527

```

```

528     x_plt, y_plt = np.meshgrid(x, y, indexing='ij')
529     true_data = analytical(x_plt, y_plt)
530
531     liebmann_sol = liebmann_method()
532     seidel_sol = seidel_method()
533     relaxation_sol = seidel_relaxation_method()
534
535     errors_liebmann_hx.append(compute_error(liebmann_sol, true_data))
536     errors_seidel_hx.append(compute_error(seidel_sol, true_data))
537     errors_relaxation_hx.append(compute_error(relaxation_sol, true_data))
538     hx_values.append(hx)
539
540 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
541
542 ax1.plot(hy_values, errors_liebmann_hy, 'o-', label=' ', linewidth=2)
543 ax1.plot(hy_values, errors_seidel_hy, 's-', label=' ', linewidth=2)
544 ax1.plot(hy_values, errors_relaxation_hy, '^-', label=' ', linewidth=2)
545 ax1.set_xlabel(' hy')
546 ax1.set_ylabel(' L2')
547 ax1.set_title(f'   hy')
548 ax1.legend()
549 ax1.grid(True, alpha=0.3)
550 ax1.set_yscale('log')
551 ax1.set_xscale('log')
552
553 ax2.plot(hx_values, errors_liebmann_hx, 'o-', label=' ', linewidth=2)
554 ax2.plot(hx_values, errors_seidel_hx, 's-', label=' ', linewidth=2)
555 ax2.plot(hx_values, errors_relaxation_hx, '^-', label=' ', linewidth=2)
556 ax2.set_xlabel(' hx')
557 ax2.set_ylabel(' L2')
558 ax2.set_title(f'   hx')
559 ax2.legend()
560 ax2.grid(True, alpha=0.3)
561 ax2.set_yscale('log')
562 ax2.set_xscale('log')
563
564 plt.tight_layout()
565 plt.show()

```


4 Выводы

В ходе выполнения лабораторной работы реализованы метод простых итераций (метод Либмана), метод Зейделя и метод простых итераций с верхней релаксацией для решения уравнения эллиптического типа. Также была реализована визуализация поверхности численного решения, а также графики срезов численного и аналитического решения вместе с графиками зависимости погрешности от шага h_x и h_y .

Список литературы

- [1] Каханер Д., Моулер К., Нэш С. *Численные методы и программное обеспечение*. М.: Мир, 1998. 575 с.
- [2] Golub G. H., Van Loan C. F. *Matrix Computations*. 4th ed. Baltimore: Johns Hopkins University Press, 2013. 756 p.