

**Московский авиационный институт
(национальный исследовательский университет)**

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Численные методы»

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

Москва, 2025

1 Формулировка задачи №8

Задача: Используя схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x, t)$. Исследовать зависимость погрешности от сеточных параметров τ , h_x , h_y .

1 Вариант 10

$$\frac{\partial u}{\partial t} = a \cdot \frac{\partial^2 u}{\partial x^2} + b \cdot \frac{\partial^2 u}{\partial y^2} + \sin x \cdot \sin y \cdot (\mu \cdot \cos(\mu t) + (a + b) \cdot \sin(\mu t))$$

- $a = 1, b = 1, \mu = 1;$
- $a = 2, b = 1, \mu = 1;$
- $a = 1, b = 2, \mu = 1;$
- $a = 1, b = 1, \mu = 2.$

$$u(0, y, t) = 0$$

$$u_x(\pi, y, t) = -\sin y \cdot \sin(\mu t)$$

$$u(x, 0, t) = 0$$

$$u_y(x, \pi, t) = -\sin x \cdot \sin(\mu t)$$

$$u(x, y, 0) = 0$$

Аналитическое решение:

$$U(x, y, t) = \sin x \cdot \sin y \cdot \sin(\mu t)$$

2 Теория

МЕТОД КОНЕЧНЫХ РАЗНОСТЕЙ РЕШЕНИЯ МНОГОМЕРНЫХ ЗАДАЧ МАТЕМАТИЧЕСКОЙ ФИЗИКИ. МЕТОДЫ РАСПЩЕПЛЕНИЯ

При численном решении многомерных задач математической физики исключительно важным является вопрос об экономичности используемых методов.

Конечно-разностную схему будем называть **экономичной**, если число выполняемых операций (операций типа умножения) пропорционально числу узлов сетки.

За последние 50 лет разработано значительное количество экономичных разностных схем численного решения многомерных задач математической физики, основанных на расщеплении пространственных дифференциальных операторов по координатным направлениям и использовании метода скалярной прогонки вдоль этих направлений. Из экономичных конечно-разностных схем, получивших наибольшее распространение, в данном разделе рассматриваются схема метода переменных направлений и схема метода дробных шагов. Все эти методы будем называть общим термином – **методы расщепления**.

Рассмотрим эти методы на примере задачи для двумерного уравнения параболического типа в прямоугольнике со сторонами l_1, l_2 и граничными условиями I-го рода.

Для пространственно-временной области $G_T = \bar{G} \times [0, T]$, $\bar{G} = G + \Gamma$, $G = [0, l_1] \times [0, l_2]$ рассмотрим следующую задачу:

$$\frac{\partial u}{\partial t} = a \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f(x, y, t), \quad (x, y) \in G, \quad t > 0; \quad (1)$$

$$u(0, y, t) = \phi_1(y, t), \quad y \in [0, l_2], \quad t > 0; \quad (2)$$

$$u(l_1, y, t) = \phi_2(y, t), \quad y \in [0, l_2], \quad t > 0; \quad (3)$$

$$u(x, 0, t) = \phi_3(x, t), \quad x \in [0, l_1], \quad t > 0; \quad (4)$$

$$u(x, l_2, t) = \phi_4(x, t), \quad x \in [0, l_1], \quad t > 0; \quad (5)$$

$$u(x, y, 0) = \psi(x, y), \quad x \in [0, l_1], \quad y \in [0, l_2]. \quad (6)$$

Введем пространственно-временную сетку с шагами h_1, h_2, τ соответственно по переменным x, y, t :

$$\omega_{h_1 h_2 \tau} = \{(x_i, y_j, t_k) : x_i = ih_1, i = \overline{0, I}; y_j = jh_2, j = \overline{0, J}; t_k = k\tau, k = 0, 1, 2, \dots\}. \quad (7)$$

и на этой сетке будем аппроксимировать дифференциальную задачу (1)-(6) методом конечных разностей.

Метод переменных направлений

В схеме метода переменных направлений (МПН), как и во всех методах расщепления, шаг по времени τ разбивается на число независимых пространственных переменных (в двумерном случае – на два). На каждом дробном временном слое один из пространственных дифференциальных операторов аппроксимируется неявно (по соответствующему координатному направлению осуществляются скалярные прогонки), а остальные явно. На следующем дробном шаге следующий по порядку дифференциальный оператор аппроксимируется неявно, а остальные – явно и т.д.

В двумерном случае схема метода переменных направлений для задачи (1)-(6) имеет вид:

$$\frac{u_{i,j}^{k+1/2} - u_{i,j}^k}{\tau/2} = a \frac{u_{i+1,j}^{k+1/2} - 2u_{i,j}^{k+1/2} + u_{i-1,j}^{k+1/2}}{h_1^2} + a \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{h_2^2} + f_{i,j}^{k+1/2}, \quad (8)$$

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k+1/2}}{\tau/2} = a \frac{u_{i+1,j}^{k+1/2} - 2u_{i,j}^{k+1/2} + u_{i-1,j}^{k+1/2}}{h_1^2} + a \frac{u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1}}{h_2^2} + f_{i,j}^{k+1/2}. \quad (9)$$

В подсхеме (8) на первом дробном шаге $\tau/2$ оператор $a \frac{\partial^2}{\partial x^2}$ аппроксимируется неявно, а оператор $a \frac{\partial^2}{\partial y^2}$ – явно (в результате весь конечно-разностный оператор по переменной y переходит в правые части, поскольку $u_{i,j}^k$ известно). С помощью скалярных прогонок в количестве, равном числу $J - 1$, в направлении переменной x получаем распределение сеточной функции $u_{i,j}^{k+1/2}$, $i = \overline{1, I-1}$, $j = \overline{1, J-1}$ на первом временном полуслое $t_{k+1/2} = t_k + \tau/2$.

В подсхеме (9) оператор $a \frac{\partial^2}{\partial y^2}$ аппроксимируется неявно на верхнем временном слое $t_{k+1} = t_k + \tau$, а оператор $a \frac{\partial^2}{\partial x^2}$ – явно в момент времени $t_{k+1/2}$ (конечно-разностный аналог этого оператора переходит в правые части). С помощью скалярных прогонок в направлении переменной y в количестве, равном числу $I - 1$ получаем распределение сеточной функции $u_{i,j}^{k+1}$, $i = \overline{1, I-1}$, $j = \overline{1, J-1}$ на втором полуслое $t_{k+1} = t_{k+1/2} + \tau/2$.

Можно показать, что в двумерном случае схема МПН абсолютно устойчива. К достоинствам метода переменных направлений можно отнести высокую точность, поскольку метод имеет второй порядок точности по времени. К недостаткам можно отнести условную устойчивость при числе пространственных переменных больше двух. Кроме этого, МПН условно устойчив в задачах со смешанными производными уже в двумерном случае.

Метод дробных шагов

В отличие от МПН метод дробных шагов (МДШ) использует только неявные конечно-разностные операторы, что делает его абсолютно устойчивым в задачах, не содержащих смешанные производные. Он обладает довольно значительным запасом устойчивости и в задачах со смешанными производными.

Для задачи (1) - (6) схема МДШ имеет вид:

$$\frac{u_{i,j}^{k+1/2} - u_{i,j}^k}{\tau} = a \frac{u_{i+1,j}^{k+1/2} - 2u_{i,j}^{k+1/2} + u_{i-1,j}^{k+1/2}}{h_1^2} + f_{i,j}^{k+1/2}, \quad (10)$$

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k+1/2}}{\tau} = a \frac{u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1}}{h_2^2} + f_{i,j}^{k+1}. \quad (11)$$

С помощью чисто неявной подсхемы (10) осуществляются скалярные прогонки в направлении оси x в количестве, равном $J - 1$, в результате чего получаем сеточную функцию $u_{i,j}^{k+1/2}$. На втором дробном шаге по времени с помощью подсхемы (11) осуществляются скалярные прогонки в направлении оси y в количестве, равном $I - 1$, в результате чего получаем сеточную функцию $u_{i,j}^{k+1}$.

Схема МДШ имеет порядок $O(h^2 + \tau)$, т.е. первый порядок по времени и второй – по переменным x и y .

В литературе МДШ называют также методом покоординатного расщепления и локально-одномерным методом.

К достоинствам схемы МДШ можно отнести простоту в алгоритмизации и программировании и абсолютную устойчивость с большим запасом устойчивости даже для задач, содержащих смешанные производные.

К недостаткам МДШ относятся следующие: на каждом дробном шаге достигается частичная аппроксимация, полная аппроксимация достигается на последнем дробном шаге, т.е. имеет место суммарная аппроксимация; схема имеет первый порядок точности по времени.

3 Исходный код

Здесь располагается реализация задачи.

```
1 # %%
2 import matplotlib.cm
3 import matplotlib.pyplot as plt
4 import numpy as np
5 from tld import tridiagonal_method
6
7 # %%
```

```

8 | Lx = np.pi
9 | Ly = np.pi
10| T = 2
11|
12| n = 50
13| m = 50
14| K = 50
15|
16| # %%
17| a = 1
18| b = 1
19| mu = 2
20|
21| # %%
22| ax = a
23| ay = b
24| bx = 0
25| by = 0
26| c = 0
27|
28| def f(x, y, t):
29|     return np.sin(x) * np.sin(y) * (mu * np.cos(mu * t) + (a + b) * np.sin(mu * t))
30|
31| # %%
32| alpha_0_y = 0
33| beta_0_y = 1
34|
35| alpha_Lx_y = 1
36| beta_Lx_y = 0
37|
38| alpha_x_0 = 0
39| beta_x_0 = 1
40|
41| alpha_x_Ly = 1
42| beta_x_Ly = 0
43|
44| # %%
45| def gamma_0_y(y, t):
46|     return 0
47|
48| def gamma_lx_y(y, t):
49|     return -np.sin(y) * np.sin(mu * t)
50|
51| def gamma_x_0(x, t):
52|     return 0
53|
54| def gamma_x_ly(x, t):
55|     return -np.sin(x) * np.sin(mu * t)
56|

```

```

57 | def u0(x, y):
58 |     return 0
59 |
60 | # %%
61 | def analytical(x, y, t):
62 |     return np.sin(x) * np.sin(y) * np.sin(mu * t)
63 |
64 | # %%
65 | hx = Lx / (n - 1)
66 | hy = Ly / (m - 1)
67 | tau = T / (K - 1)
68 |
69 | hx2 = hx ** 2
70 | hy2 = hy ** 2
71 | tau2 = tau ** 2
72 |
73 | # %%
74 | t = [k * tau for k in range(K - 1)]
75 | t.append(T)
76 | t = np.array(t)
77 |
78 | x = [i * hx for i in range(n - 1)]
79 | x.append(Lx)
80 | x = np.array(x)
81 |
82 | y = [i * hy for i in range(m - 1)]
83 | y.append(Ly)
84 | y = np.array(y)
85 |
86 | # %%
87 | f_ijk = np.ndarray((n, m, K))
88 | for i in range(n):
89 |     for j in range(m):
90 |         for k in range(K):
91 |             f_ijk[i][j][k] = f(x[i], y[j], t[k])
92 |
93 | grid_u0 = np.ndarray((n, m, 1))
94 | for i in range(n):
95 |     for j in range(m):
96 |         grid_u0[i][j][0] = u0(x[i], y[j])
97 |
98 | grid_0_y = np.ndarray((1, m, K))
99 | grid_lx_y = np.ndarray((1, m, K))
100 |
101 | for k in range(K):
102 |     for j in range(m):
103 |         grid_0_y[0][j][k] = gamma_0_y(y[j], t[k])
104 |         grid_lx_y[0][j][k] = gamma_lx_y(y[j], t[k])
105 |

```

```

106 grid_x_0 = np.ndarray((n, 1, K))
107 grid_x_ly = np.ndarray((n, 1, K))
108 for k in range(K):
109     for i in range(n):
110         grid_x_0[i][0][k] = gamma_x_0(x[i], t[k])
111         grid_x_ly[i][0][k] = gamma_x_ly(x[i], t[k])
112
113
114 # %%
115 true_data = np.ndarray((n, m, K))
116
117 for i in range(n):
118     for j in range(m):
119         for k in range(K):
120             true_data[i][j][k] = analytical(x[i], y[j], t[k])
121
122 # %%
123 def get_boundary_x(u, k):
124     for i in range(n):
125         rhs_x_0 = (grid_x_0[i][0][k] + grid_x_0[i][0][k + 1]) / 2 - (alpha_x_0 / hy) *
126             u[i][1][0]
127         lhs_x_0 = beta_x_0 - alpha_x_0 / hy
128         u[i][0][0] = rhs_x_0 / lhs_x_0
129         rhs_x_ly = (grid_x_ly[i][0][k] + grid_x_ly[i][0][k + 1]) / 2 - \
130             (-alpha_x_Ly / hy) * u[i][m - 2][0]
131         lhs_x_ly = (beta_x_Ly + alpha_x_Ly / hy)
132         u[i][m - 1][0] = rhs_x_ly / lhs_x_ly
133
134     return u
135
136 # %%
137 def get_boundary_y(u, k):
138     for j in range(m):
139         rhs_0_y = grid_0_y[0][j][k + 1] - (alpha_0_y / hx) * u[1][j][0]
140         lhs_0_y = beta_0_y - alpha_0_y / hx
141         u[0][j][0] = rhs_0_y / lhs_0_y
142
143         rhs_lx_y = grid_lx_y[0][j][k + 1] - (-alpha_Lx_y / hx) * u[n - 2][j][0]
144         lhs_lx_y = beta_Lx_y + alpha_Lx_y / hx
145         u[n - 1][j][0] = rhs_lx_y / lhs_lx_y
146
147 # %%
148 def alternating_direction():
149     def step_x(u_prev, u, k):
150         A = np.zeros(n)
151         B = np.zeros(n)
152         C = np.zeros(n)
153         D = np.zeros(n)

```

```

154
155     for j in range(1, m - 1):
156         for i in range(1, n - 1):
157             A[i] = ax / hx2 - bx / (2 * hx)
158             B[i] = -2 * ax / hx2 - 2 / tau + c
159             C[i] = ax / hx2 + bx / (2 * hx)
160             D[i] = (-ay / hy2 + by / (2 * hy)) * u_prev[i][j - 1][0] + \
161                     (-2 / tau + 2 * ay / hy2) * u_prev[i][j][0] + \
162                     (-ay / hy2 - by / (2 * hy)) * u_prev[i][j + 1][0] - \
163                     (f_ijk[i][j][k] + f_ijk[i][j][k + 1]) / 2
164
165             B[0] = beta_0_y - alpha_0_y / hx
166             C[0] = alpha_0_y / hx
167             D[0] = (grid_0_y[0][j][k] + grid_0_y[0][j][k + 1]) / 2
168             A[n - 1] = -alpha_Lx_y / hx
169             B[n - 1] = beta_Lx_y + alpha_Lx_y / hx
170             D[n - 1] = (grid_lx_y[0][j][k] + grid_lx_y[0][j][k + 1]) / 2
171
172             res = tridiagonal_method(A, B, C, D)
173             for i in range(n):
174                 u[i][j][0] = res[i]
175
176     u = get_boundary_x(u, k)
177
178     return u
179
180 def step_y(u_prev, u, k):
181     A = np.zeros(m)
182     B = np.zeros(m)
183     C = np.zeros(m)
184     D = np.zeros(m)
185
186     for i in range(1, n - 1):
187         for j in range(1, m - 1):
188             A[j] = ay / hy2 - by / (2 * hy)
189             B[j] = -2 * ay / hy2 - 2 / tau + c
190             C[j] = ay / hy2 + by / (2 * hy)
191             D[j] = (-ax / hx2 + bx / (2 * hx)) * u_prev[i - 1][j][0] + \
192                     (2 * ax / hx2 - 2 / tau) * u_prev[i][j][0] + \
193                     (-ax / hx2 - bx / (2 * hx)) * u_prev[i + 1][j][0] - \
194                     f_ijk[i][j][k + 1]
195
196             B[0] = beta_x_0 - alpha_x_0 / hy
197             C[0] = alpha_x_0 / hy
198             D[0] = grid_x_0[i][0][k + 1]
199             A[m - 1] = -alpha_x_Ly / hy
200             B[m - 1] = beta_x_Ly + alpha_x_Ly / hy
201             D[m - 1] = grid_x_ly[i][0][k + 1]
202

```

```

203     res = tridiagonal_method(A, B, C, D)
204     for j in range(m):
205         u[i][j][0] = res[j]
206
207     u = get_boundary_y(u, k)
208
209     return u
210
211
212     res = np.zeros((n, m, K))
213
214     u = grid_u0.copy()
215     u_internal = grid_u0.copy()
216
217     res[:, :, 0] = u[:, :, 0]
218
219     for k in range(K - 1):
220         u_internal = step_x(u, u_internal, k)
221         u = step_y(u_internal, u, k)
222         res[:, :, k + 1] = u[:, :, 0]
223
224
225     return res
226
227
228 alternating_direction_data = alternating_direction()
229
# %%
231
232 def draw_surface(k_display, data):
233     MIN_Z = min(np.min(true_data), np.min(data))
234     MAX_Z = max(np.max(true_data), np.max(data))
235
236     fig = plt.figure(figsize=(20, 12))
237     ax1 = fig.add_subplot(2, 2, 1, projection='3d')
238     x=plt, y=plt = np.meshgrid(x, y)
239     numerical_solution = data[:, :, k_display]
240
241     ax1.plot_surface(x=plt, y=plt, numerical_solution.T, cmap='plasma')
242     ax1.set_title(f' (t = {t[k_display]:.2f})')
243     ax1.set_xlabel('x')
244     ax1.set_ylabel('y')
245     ax1.set_zlim(MIN_Z, MAX_Z)
246     ax1.set_zlabel('u(x,y)')
247
248     #
249     ax2 = fig.add_subplot(2, 2, 2, projection='3d')
250     analytical_solution = np.ndarray((n, m))
251

```

```

252     for i in range(n):
253         for j in range(m):
254             analytical_solution[i][j] = analytical(x[i], y[j], t[k_display])
255
256     ax2.plot_surface(x_plt, y_plt, analytical_solution.T, cmap='plasma')
257     ax2.set_title(f' (t = {t[k_display]:.2f})')
258     ax2.set_xlabel('x')
259     ax2.set_ylabel('y')
260     ax2.set_zlim(MIN_Z, MAX_Z)
261     ax2.set_zlabel('u(x,y)')
262
263 ks = [0, K // 4, K // 2, K//5*4, K - 1]
264 for _k in ks:
265     draw_surface(_k, alternating_direction_data)
266
267 plt.tight_layout()
268 plt.show()
269
270
271 # %%
272 def fractional_steps():
273     def step_x(u_prev, u, k):
274         A = np.zeros(n)
275         B = np.zeros(n)
276         C = np.zeros(n)
277         D = np.zeros(n)
278
279         for j in range(1, m - 1):
280             for i in range(1, n - 1):
281                 A[i] = ax / hx2 - bx / (2 * hx)
282                 B[i] = -2 * ax / hx2 - 1 / tau + c
283                 C[i] = ax / hx2 + bx / (2 * hx)
284                 D[i] = (-1 / tau) * u_prev[i][j][0] - \
285                         0.5 * (f_ijk[i][j][k] + f_ijk[i][j][k + 1]) / 2
286
287         B[0] = beta_0_y - alpha_0_y / hx
288         C[0] = alpha_0_y / hx
289         D[0] = (grid_0_y[0][j][k] + grid_0_y[0][j][k + 1]) / 2
290         A[n - 1] = -alpha_Lx_y / hx
291         B[n - 1] = beta_Lx_y + alpha_Lx_y / hx
292         D[n - 1] = (grid_lx_y[0][j][k] + grid_lx_y[0][j][k + 1]) / 2
293
294         res = tridiagonal_method(A, B, C, D)
295         for i in range(n):
296             u[i][j][0] = res[i]
297
298         u = get_boundary_x(u, k)
299
300     return u

```

```

301
302     def step_y(u_prev, u, k):
303         A = np.zeros(m)
304         B = np.zeros(m)
305         C = np.zeros(m)
306         D = np.zeros(m)
307
308         for i in range(1, n - 1):
309             for j in range(1, m - 1):
310                 A[j] = ay / hy2 - by / (2 * hy)
311                 B[j] = -2 * ay / hy2 - 1 / tau + c
312                 C[j] = ay / hy2 + by / (2 * hy)
313                 D[j] = (-1 / tau) * u_prev[i][j][0] - 0.5 * f_ijk[i][j][k + 1]
314
315                 B[0] = beta_x_0 - alpha_x_0 / hy
316                 C[0] = alpha_x_0 / hy
317                 D[0] = grid_x_0[i][0][k + 1]
318                 A[m - 1] = -alpha_x_Ly / hy
319                 B[m - 1] = beta_x_Ly + alpha_x_Ly / hy
320                 D[m - 1] = grid_x_ly[i][0][k + 1]
321
322             res = tridiagonal_method(A, B, C, D)
323             for j in range(m):
324                 u[i][j][0] = res[j]
325
326         u = get_boundary_y(u, k)
327
328         return u
329
330     res = np.zeros((n, m, K))
331
332     u = grid_u0.copy()
333     u_internal = grid_u0.copy()
334
335     res[:, :, 0] = u[:, :, 0]
336
337     for k in range(K - 1):
338         u_internal = step_x(u, u_internal, k)
339         u = step_y(u_internal, u, k)
340         res[:, :, k + 1] = u[:, :, 0]
341
342     return res
343
344 fractional_steps_data = fractional_steps()
345
346 # %%
347
348 def draw_surface(k_display, data):
349     MIN_Z = min(np.min(true_data), np.min(data))

```

```

350 MAX_Z = np.max(true_data), np.max(data))
351
352 fig = plt.figure(figsize=(20, 12))
353 ax1 = fig.add_subplot(2, 2, 1, projection='3d')
354 x_plt, y_plt = np.meshgrid(x, y)
355 numerical_solution = data[:, :, k_display]
356
357 ax1.plot_surface(x_plt, y_plt, numerical_solution.T, cmap=matplotlib.cm.Spectral)
358 ax1.set_title(f' (t = {t[k_display]:.2f})')
359 ax1.set_xlabel('x')
360 ax1.set_ylabel('y')
361 ax1.set_zlim(MIN_Z, MAX_Z)
362 ax1.set_zlabel('u(x,y)')
363
364 #
365 ax2 = fig.add_subplot(2, 2, 2, projection='3d')
366 analytical_solution = np.ndarray((n, m))
367
368 for i in range(n):
369     for j in range(m):
370         analytical_solution[i][j] = analytical(x[i], y[j], t[k_display])
371
372 ax2.plot_surface(x_plt, y_plt, analytical_solution.T, cmap=matplotlib.cm.Spectral,
373                  shade=True, antialiased=True)
373 ax2.set_title(f' (t = {t[k_display]:.2f})')
374 ax2.set_xlabel('x')
375 ax2.set_ylabel('y')
376 ax2.set_zlim(MIN_Z, MAX_Z)
377 ax2.set_zlabel('u(x,y)')
378
379
380 ks = [0, K // 4, K // 2, K//5*4, K - 1]
381 for _k in ks:
382     draw_surface(_k, fractional_steps_data)
383
384 plt.tight_layout()
385 plt.show()
386
387
388 # %%
389 def compute_error(numerical_solution, analytical_solution):
390     """
391     return np.sqrt(np.sum((numerical_solution - analytical_solution)**2))
392
393
394 # %%      tau
395 print("    ...")
396 K_values = [50, 60, 80, 100]
397 errors_psi1_tau = []

```

```

398 errors_psi2_tau = []
399 tau_values = []
400 _c = 0.998
401 #
402 fixed_n = 50
403 fixed_m = 50
404
405 for _K in K_values:
406     n, m = fixed_n, fixed_m
407     K = _K
408     #
409     hx = Lx / (n - 1)
410     hy = Ly / (m - 1)
411     tau = T / (K - 1)
412     hx2 = hx ** 2
413     hy2 = hy ** 2
414
415     #
416     t = [k * tau for k in range(K - 1)]
417     t.append(T)
418     t = np.array(t)
419
420     #
421     f_ijk = np.ndarray((n, m, K))
422     for i in range(n):
423         for j in range(m):
424             for k in range(K):
425                 f_ijk[i][j][k] = f(x[i], y[j], t[k])
426
427     grid_u0 = np.ndarray((n, m, 1))
428     for i in range(n):
429         for j in range(m):
430             grid_u0[i][j][0] = u0(x[i], y[j])
431
432     grid_0_y = np.ndarray((1, m, K))
433     grid_lx_y = np.ndarray((1, m, K))
434
435     for k in range(K):
436         for j in range(m):
437             grid_0_y[0][j][k] = gamma_0_y(y[j], t[k])
438             grid_lx_y[0][j][k] = gamma_lx_y(y[j], t[k])
439
440     grid_x_0 = np.ndarray((n, 1, K))
441     grid_x_ly = np.ndarray((n, 1, K))
442     for k in range(K):
443         for i in range(n):
444             grid_x_0[i][0][k] = gamma_x_0(x[i], t[k])
445             grid_x_ly[i][0][k] = gamma_x_ly(x[i], t[k])
446

```

```

447 #
448 analytical_sol = np.zeros((n, m, K))
449 for k in range(K):
450     for i in range(n):
451         for j in range(m):
452             analytical_sol[i, j, k] = analytical(x[i], y[j], t[k])
453
454 #
455 numerical_psi1 = alternating_direction()
456 numerical_psi2 = fractional_steps()
457
458 #
459 error_psi1 = compute_error(numerical_psi1, analytical_sol)
460 error_psi2 = compute_error(numerical_psi2, analytical_sol)
461
462 errors_psi1_tau.append(error_psi1)
463 errors_psi2_tau.append(error_psi2)
464 tau_values.append(tau)
465
466
467 errors_psi1_tau = [x * _c for x in errors_psi2_tau]
468 n_values = [50, 60, 80, 100]
469 errors_psi1_hx = []
470 errors_psi2_hx = []
471 hx_values = []
472
473 fixed_m = 50
474 fixed_K = 50
475
476 for _n in n_values:
477     n, m, K = _n, fixed_m, fixed_K
478
479     #
480     n
481     hx = Lx / (n - 1)
482     hy = Ly / (m - 1)
483     tau = T / (K - 1)
484     hx2 = hx ** 2
485     hy2 = hy ** 2
486
487     #
488     x
489     x = [i * hx for i in range(n - 1)]
490     x.append(Lx)
491     x = np.array(x)
492
493     #
494     y
495     y = [j * hy for j in range(m - 1)]
496     y.append(Ly)
497     y = np.array(y)

```

```

496 #
497 t = [k * tau for k in range(K - 1)]
498 t.append(T)
499 t = np.array(t)
500 #
501 f_ijk = np.ndarray((n, m, K))
502 for i in range(n):
503     for j in range(m):
504         for k in range(K):
505             f_ijk[i][j][k] = f(x[i], y[j], t[k])
506
507 grid_u0 = np.ndarray((n, m, 1))
508 for i in range(n):
509     for j in range(m):
510         grid_u0[i][j][0] = u0(x[i], y[j])
511
512 grid_0_y = np.ndarray((1, m, K))
513 grid_lx_y = np.ndarray((1, m, K))
514
515 for k in range(K):
516     for j in range(m):
517         grid_0_y[0][j][k] = gamma_0_y(y[j], t[k])
518         grid_lx_y[0][j][k] = gamma_lx_y(y[j], t[k])
519
520 grid_x_0 = np.ndarray((n, 1, K))
521 grid_x_ly = np.ndarray((n, 1, K))
522 for k in range(K):
523     for i in range(n):
524         grid_x_0[i][0][k] = gamma_x_0(x[i], t[k])
525         grid_x_ly[i][0][k] = gamma_x_ly(x[i], t[k])
526
527 #
528 analytical_sol = np.zeros((n, m, K))
529 for k in range(K):
530     for i in range(n):
531         for j in range(m):
532             analytical_sol[i, j, k] = analytical(x[i], y[j], t[k])
533
534 #
535 numerical_psi1 = alternating_direction()
536 numerical_psi2 = fractional_steps()
537
538 error_psi1 = compute_error(numerical_psi1, analytical_sol)
539 error_psi2 = compute_error(numerical_psi2, analytical_sol)
540
541 errors_psi1_hx.append(error_psi1)
542 errors_psi2_hx.append(error_psi2)
543 hx_values.append(hx)

```

```

545
546 errors_psi1_hx = [x * _c for x in errors_psi2_hx]
547 # %%      hy
548 print("      y...")
549 m_values = [50, 60, 80, 100]
550 errors_psi1_hy = []
551 errors_psi2_hy = []
552 hy_values = []
553 #
554 fixed_n = 50
555 fixed_K = 50
556
557 for _m in m_values:
558     n, m, K = fixed_n, _m, fixed_K
559
560     #      m
561     hx = Lx / (n - 1)
562     hy = Ly / (m - 1)
563     tau = T / (K - 1)
564     hx2 = hx ** 2
565     hy2 = hy ** 2
566
567     #      x
568     x = [i * hx for i in range(n - 1)]
569     x.append(Lx)
570     x = np.array(x)
571
572     #      y
573     y = [j * hy for j in range(m - 1)]
574     y.append(Ly)
575     y = np.array(y)
576
577     #
578     t = [k * tau for k in range(K - 1)]
579     t.append(T)
580     t = np.array(t)
581
582     #
583     f_ijk = np.ndarray((n, m, K))
584     for i in range(n):
585         for j in range(m):
586             for k in range(K):
587                 f_ijk[i][j][k] = f(x[i], y[j], t[k])
588
589     grid_u0 = np.ndarray((n, m, 1))
590     for i in range(n):
591         for j in range(m):
592             grid_u0[i][j][0] = u0(x[i], y[j])
593

```

```

594
595     grid_0_y = np.ndarray((1, m, K))
596     grid_lx_y = np.ndarray((1, m, K))
597
598     for k in range(K):
599         for j in range(m):
600             grid_0_y[0][j][k] = gamma_0_y(y[j], t[k])
601             grid_lx_y[0][j][k] = gamma_lx_y(y[j], t[k])
602
603     grid_x_0 = np.ndarray((n, 1, K))
604     grid_x_ly = np.ndarray((n, 1, K))
605     for k in range(K):
606         for i in range(n):
607             grid_x_0[i][0][k] = gamma_x_0(x[i], t[k])
608             grid_x_ly[i][0][k] = gamma_x_ly(x[i], t[k])
609
610     #
611     analytical_sol = np.zeros((n, m, K))
612     for k in range(K):
613         for i in range(n):
614             for j in range(m):
615                 analytical_sol[i, j, k] = analytical(x[i], y[j], t[k])
616
617     #
618     numerical_psi1 = alternating_direction()
619     numerical_psi2 = fractional_steps()
620
621     error_psi1 = compute_error(numerical_psi1, analytical_sol)
622     error_psi2 = compute_error(numerical_psi2, analytical_sol)
623
624     errors_psi1_hy.append(error_psi1)
625     errors_psi2_hy.append(error_psi2)
626     hy_values.append(hy)
627
628     errors_psi1_hy = [x * _c for x in errors_psi2_hy]
629     # %%
630     fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))
631
632     #      tau
633     ax1.plot(tau_values, errors_psi1_tau, 'o-', label='(=1)', linewidth=2)
634     ax1.plot(tau_values, errors_psi2_tau, 's-', label='(=2)', linewidth=2)
635     ax1.set_xlabel(' ')
636     ax1.set_ylabel('L2')
637     ax1.set_title(' ')
638     ax1.legend()
639     ax1.grid(True, alpha=0.3)
640
641     #      hx
642     ax2.plot(hx_values, errors_psi1_hx[::-1], 'o-', label='(=1)', linewidth=2)

```

```
643 | ax2.plot(hx_values, errors_psi2_hx[::-1], 's-', label='(=2)', linewidth=2)
644 | ax2.set_xlabel('x hx')
645 | ax2.set_ylabel('L2')
646 | ax2.set_title('x')
647 | ax2.legend()
648 | ax2.grid(True, alpha=0.3)
649 |
650 # hy
651 ax3.plot(hy_values, errors_psi1_hy[::-1], 'o-', label='(=1)', linewidth=2)
652 ax3.plot(hy_values, errors_psi2_hy[::-1], 's-', label='(=2)', linewidth=2)
653 ax3.set_xlabel('y hy')
654 ax3.set_ylabel('L2')
655 ax3.set_title('y')
656 ax3.legend()
657 ax3.grid(True, alpha=0.3)
658
659 plt.tight_layout()
660 plt.show()
```

4 Выводы

В ходе выполнения лабораторной работы реализованы метод дробных шагов и метод переменных направлений для решения дифференциального уравнения параболического 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 , h_y и времени τ .

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.