Содержание

1 Задание

Рассматривается начально-краевая задача для нелинейного уравнения теплопроводности:

$$\begin{cases} y_t = \left(\left(1 + y^{3/2}\right) y_x\right)_x + f(t,x), x \in (0, L), t \in (0, T), \\ y|_{t=0} = \varphi(x), \\ y|_{x=0} = g_0(t), y|_{x=L} = g_L(t) \end{cases} \tag{1}$$

Искомым решением является функция $y = y(t,x)$.

Необходимо построить *чисто неявную* и *симметричную* разностную схему, а также реализовать *метод простой итерации* для решения нелинейной системы уравнений.

Выполнить серию вычислительных экспериментов на стандартных тестах. Сравнить время работы PC на одном и том же отрезке $[0, T]$. Составьте (для каждого тестового расчета) таблицы зависимости времени расчета и количества итераций на шаге от шага сетки τ.

2 Теоретическая часть

2.1 Тестовые примеры

2.1.1 Плоскость

$y(t,x) = 2x + 3t + 5$

$$\begin{cases} y_t = \left(\left(1 + y^{3/2}\right) y_x\right)_x + 3 - 6\sqrt{2x + 3t + 5}, \\ x \in (0, L), t \in (0, T), \\ y|_{t=0} = 2x - 5, \\ y|_{x=0} = 3t - 5, \\ y|_{x=L} = 2L + 3t - 5 \end{cases} \tag{2}$$

2.1.2 Линейно-квадратичная функция

$y(t,x) = 2x^2 + t + 3$

$$\begin{cases} y_t = \left(\left(1 + y^{3/2}\right) y_x\right)_x - 3 - 4y(t,x)^{3/2} - 24x^2 y^{1/2}, \\ x \in (0, L), t \in (0, T), \\ y|_{t=0} = 2x^2 + 3, \\ y|_{x=0} = t + 3, \\ y|_{x=L} = 2L^2 + t + 3 \end{cases} \tag{3}$$

2.1.3 Параболоид

$y(t,x) = 3x^2 + 2t^2 + 1$

$$\begin{cases} y_t = \left(\left(1 + y^{3/2}\right) y_x\right)_x + 4t - 6 - 6y(t,x)^{3/2} - 54x^2 y(t,x)^{1/2}, \\ x \in (0, L), t \in (0, T), \\ y|_{t=0} = 3x^2 + 1, \\ y|_{x=0} = 2t^2 + 1, \\ y|_{x=L} = 3L^2 + 2t^2 + 1 \end{cases} \tag{4}$$

2.1.4 Индивидуальная функция

$y(t,x) = e^{\sin^2 x + \cos^2 t}$

$$\begin{cases} y_t = \left(\left(1 + y^{3/2}\right) y_x\right)_x + y(t,x) \left(-\sin 2t - 2\cos 2x(1 + y(t,x)^{3/2}) - \right. \\ \left. - \sin^2 2x \left(1 + \frac{5}{2} y(t,x)^{3/2}\right), \right. \\ x \in (0, L), t \in (0, T), \\ y|_{t=0} = e^{\sin^2 x + 1}, \\ y|_{x=0} = e^{\cos^2 t}, \\ y|_{x=L} = e^{\sin^2 L + \cos^2 t} \end{cases} \tag{5}$$

2.2 Разностные схемы

2.2.1 Симметричная схема

$$\begin{cases} \dfrac{U^{j+1} - U^j}{\tau} = \dfrac{1}{2}\left[\Lambda\left(t^{j+1}\right) U^{j+1} + \Lambda\left(t^j\right) U^j + \varphi^j\right], \\ U_0^{j+1} = g_0(t_{j+1}), \\ U_N^{j+1} = g_L(t_{j+1}). \end{cases} \tag{6}$$

2.2.2 Чисто неявная схема

$$\begin{cases} \dfrac{U^{j+1} - U^j}{\tau} = \Lambda\left(t^{j+1}\right) U^{j+1} + \varphi^{j+1}, \\ U_0^{j+1} = g_0(t_{j+1}), \\ U_N^{j+1} = g_L(t_{j+1}). \end{cases} \tag{7}$$

2.3 Метод решения нелинейной системы уравнений

2.3.1 Метод простой итерации

Для чисто неявной схемы:

$$U^{(s+1)} = U^j + \frac{\tau}{h}\left(\Lambda\left(t^{(s)}\right)U^{(s)}\right) + \tau f^{(s)} \tag{8}$$

Для симметричной схемы:

$$U^{(s+1)} = U^j + \frac{\tau}{2h}\left(\Lambda\left(t^{(s)}\right)U^{(s)} + \Lambda\left(t^j\right)U^j\right) + \tau f^{(s)} \tag{9}$$

2.4 Аппроксимация коэффициента теплопроводности

В качестве аппроксимации коэффициента $k(t, x, y)$ выберем:

$$a_i = \frac{1}{2}(k(t,x_i,y_i) - k(t,x_{i-1},y_{i-1})) \tag{10}$$

Погрешность аппроксимации:

$$\Psi_\Lambda = \mathcal{O}(h^2) \tag{11}$$

3 Практическая часть

3.1 Вычислительные эксперименты над симметричной схемой

Начальные параметры: $\tau = 10^{-5}$, $h = 0.1$, $L = 1$, $T = 1$, $\varepsilon = 10^{-6}$

3.1.1 Плоскость

τ	*Time, sec*	*Iterations(per step)*
τ	21.6	2
$\tau/2$	44.5	2
$\tau/4$	86.3	2
$\tau/8$	175.8	2

3.1.2 Линейно-квадратичная функция

τ	*Time, sec*	*Iterations(per step)*
τ	23.7	2
$\tau/2$	49.9	2
$\tau/4$	94.2	2
$\tau/8$	190	2

3.1.3 Параболоид

τ	*Time, sec*	*Iterations(per step)*
τ	24.1	2
$\tau/2$	49.4	2
$\tau/4$	102.5	2
$\tau/8$	201	2

3.1.4 Индивидуальная функция

τ	*Time, sec*	*Iterations(per step)*
τ	41.1	2
$\tau/2$	80.7	2
$\tau/4$	163.5	2
$\tau/8$	324.2	2

3.2 Вычислительные эксперименты над чисто неявной схемой

3.2.1 Плоскость

τ	*Time, sec*	*Iterations(per step)*
τ	16.1	2 - 3
$\tau/2$	27.9	2
$\tau/4$	55.5	2
$\tau/8$	110.1	2

3.2.2 Линейно-квадратичная функция

τ	*Time, sec*	*Iterations(per step)*
τ	15.7	2
$\tau/2$	31.3	2
$\tau/4$	61.8	2
$\tau/8$	124.5	2

3.2.3 Параболоид

τ	*Time, sec*	*Iterations(per step)*
τ	17.5	2
$\tau/2$	32.7	2
$\tau/4$	66.2	2
$\tau/8$	130.5	2

3.2.4 Индивидуальная функция

τ	*Time, sec*	*Iterations(per step)*
τ	32.7	2
$\tau/2$	65.2	2
$\tau/4$	128.7	2
$\tau/8$	264.2	2

Вывод: В ходе лабораторной работы №4 были построены разностные схемы: чисто неявная, симметричная. С помощью этих схем решалось уравнение теплопроводности с нелинейным коэффициентом теплопроводности, поэтому понадобилось решать нелинейную систему уравнений. Метод простой итерации для этого не очень подходит, так как возникает условие на малость шага τ и величины τ/h, в силу данного факта наблюдается тенденция к минимальному количеству итераций на шаге(2-3). По результатам вычислений можно сделать вывод, что чисто неявная работает немного быстрее симметричной, но погрешность у симметричной (второй порядок точности по τ) меньше, чем у чисто неявной(первый порядок точности по τ).

4 Приложение

```python
import numpy as np
import matplotlib.pyplot as plt
import time

# Границы x и t
L = 1
T = 1
# Сетка
tau_grid = 0.00000125
x_grid = 0.1
# сетка по x
Nx = int(L / x_grid + 1)
h = np.linspace(0, L, Nx)
# сетка по t
Nt = int(T / tau_grid + 1)
tau = np.linspace(0, T, Nt)


# Коэффициент теплопроводности
def Thermal_conductivity(t, x, y):
    return 1 + y ** (3 / 2)
```

```
22
23
24  # Тестовые примеры
25  # Плоскость
26  def tf1(t, x):
27      return 2 * x + 3 * t + 5
28
29
30  def fx_tf1(t, x):
31      return 3 - 6 * np.sqrt(2 * x + 3 * t + 5)
32
33
34  def yt0_tf1(t, x):
35      return 2 * x + 5
36
37
38  def yx0_tf1(t, x):
39      return 3 * t + 5
40
41
42  def yxL_tf1(t, x):
43      return 2 * L + 3 * t + 5
44
45
46  # Линейно-квадратичная функция
47  def tf2(t, x):
48      return 2 * x ** 2 + t + 3
49
50
51  def fx_tf2(t, x):
52      return -3 - 4 * ((2 * x ** 2 + t + 3) ** (3 / 2)) - (24 * x
        ↪  ** 2) * np.sqrt(2 * x ** 2 + t + 3)
53
54
55  def yt0_tf2(t, x):
56      return 2 * x ** 2 + 3
57
58
59  def yx0_tf2(t, x):
60      return t + 3
61
62
63  def yxL_tf2(t, x):
```

```
64      return 2 * L ** 2 + t + 3
65
66
67  # Параболоид
68  def tf3(t, x):
69      return 3 * x ** 2 + 2 * t ** 2 + 1
70
71
72  def fx_tf3(t, x):
73      return 4 * t - 6 - 6 * (tf3(t, x) ** 1.5) - 54 * (x ** 2) *
        ↪  (tf3(t, x) ** 0.5)
74
75
76  def yt0_tf3(t, x):
77      return 3 * x ** 2 + 1
78
79
80  def yx0_tf3(t, x):
81      return 2 * t ** 2 + 1
82
83
84  def yxL_tf3(t, x):
85      return 3 * L ** 2 + 2 * t ** 2 + 1
86
87
88  # Индивидуальная функция
89  def tf4(t, x):
90      return np.exp(np.sin(x) ** 2 + np.cos(t) ** 2)
91
92
93  def fx_tf4(t, x):
94      return tf4(t, x) * (-1 * np.sin(2 * t) - 2 * np.cos(2 * x) *
        ↪  (1 + (tf4(t, x) ** 1.5)) - 1 * (np.sin(2 * x) ** 2)
95                          * (1 + 2.5 * (tf4(t, x) ** 1.5)))
96
97
98  def yt0_tf4(t, x):
99      return np.exp((np.sin(x) ** 2) + 1)
100
101
102 def yx0_tf4(t, x):
103     return np.exp(np.cos(t) ** 2)
104
```

```
105
106 def yxL_tf4(t, x):
107     return np.exp(np.sin(L) ** 2 + np.cos(t) ** 2)
108
109
110 # аппроксимация коэффициента теплопроводности
111 def a(t_1, x_1, y_1, t_2, x_2, y_2):
112     return 0.5 * (Thermal_conductivity(t_1, x_1, y_1) +
        ↪ Thermal_conductivity(t_2, x_2, y_2))
113
114
115 # Чисто неявная схема
116 def pis(a, fp, yleft, yright, tm, phif, epsilon):
117     # Искомый слой
118     res = np.zeros(Nx)
119     # Начальное приближение
120     for i in range(Nx):
121         res[i] = fp[i]  # fp - предыдущий слой
122     res[0] = yleft  # Краевое условие слева
123     res[-1] = yright  # Краевое условие справа
124     res_help = res.copy()  # вспомогательный слой
125     res_const = res.copy()  # предыдущий слой
126     res_prom = res.copy()  # слой после итерации
127     k = 0  # счетчик
128     while True:
129         k += 1
130         for i in range(1, Nx - 1):
131             res_prom[i] = (tau_grid / x_grid ** 2) * (
132                         a(tm, h[i + 1], res_help[i + 1], tm,
                             ↪ h[i], res_help[i]) * (res_help[i +
                             ↪ 1] - res_help[i]) - a(
133                     tm, h[i], res_help[i], tm, h[i - 1],
                         ↪ res_help[i - 1]) * (
134                                     res_help[i] - res_help[i -
                                         ↪ 1])) + tau_grid *
                                         ↪ phif[i] + res_const[i]
135         res_help = res_prom.copy()
136         # проверка условия выхода из цикла итераций
137         if np.linalg.norm(res_help - res) < epsilon:
138             print(k)  # вывод количества итераций
139             return res_help
140         else:
141             res = res_help.copy()
```

```
142
143
144 # симметричная схема
145 def simsh(a, fp, yleft, yright, tm, phif, epsilon):
146     # Вектор решения в определенный момент времени
147     res = np.zeros(Nx)
148     # Начальное приближение
149     for i in range(Nx):
150         res[i] = fp[i]
151     res_const = res.copy()  # предыдущий слой
152     res[0] = yleft  # Краевое условие слева
153     res[-1] = yright  # Краевое условие справа
154     res_help = res.copy()  # вспомогательный слой
155     res_prom = res.copy()  # слой после итерации
156     k = 0  # счетчик
157     while True:
158         k += 1
159         for i in range(1, Nx - 1):
160             res_prom[i] = (tau_grid / (2 * x_grid ** 2)) * (
161                         a(tm, h[i + 1], res_help[i + 1], tm,
                             ↪  h[i], res_help[i]) * (res_help[i +
                             ↪  1] - res_help[i]) - a(
162                     tm, h[i], res_help[i], tm, h[i - 1],
                         ↪  res_help[i - 1]) * (res_help[i] -
                         ↪  res_help[i - 1])) + (
163                                     tau_grid / (2 * x_grid **
                                         ↪  2)) * (
164                                     a(tm - tau_grid, h[i + 1],
                                         ↪  res_const[i + 1], tm -
                                         ↪  tau_grid, h[i],
165                                       res_const[i]) *
                                           ↪  (res_const[i + 1] -
                                           ↪  res_const[i]) - a(tm
                                           ↪  - tau_grid, h[i],
166
167
```

```
168
169                                                                 res_const[i] -
                                                                     ↪ res_const[i
                                                                     ↪ - 1])) +
                                                                     ↪ tau_grid *
                                                                     ↪ phif[i] +
                                                                     ↪ res_const[i]
170             res_help = res_prom.copy()
171             # проверка условия выхода из цикла итераций
172             if np.linalg.norm(res_help - res) < epsilon:
173                 print(k)  # вывод количества итераций
174                 return res_help
175             else:
176                 res = res_help.copy()
177
178
179     def solve_pis(ut0, ux0, uxL, tf, ep):
180         print('---Чисто неявная схема---')
181         solve_matr = np.zeros((Nt, Nx))  # Матрица решений
182         # заполнение матрицы начальным условием t = 0
183         for i in range(Nx):
184             solve_matr[Nt - 1][i] = ut0(0, h[i])
185         fi = np.zeros(Nx)  # начальное приближение
186         for tme in range(1, Nt):
187             for j in range(Nx):  # заполнение начального
                 ↪ приближения
188                 fi[j] = tf(tau[tme], h[j])
189             s = pis(a, solve_matr[Nt - tme], ux0(tau[tme], h[0]),
                 ↪ uxL(tau[tme], h[Nx - 1]), tau[tme], fi, ep)
190             solve_matr[Nt - tme - 1] = s
191         return solve_matr
192
193
194     def solve_simshame(ut0, ux0, uxL, tf, ep):
195         print('---Симметричная схема---')
196         solve_matr = np.zeros((Nt, Nx))  # Матрица решений
197         # заполнение матрицы начальным условием t = 0
198         for i in range(Nx):
```

```
199             solve_matr[Nt - 1][i] = ut0(0, h[i])
200         fi = np.zeros(Nx)  # начальное приближение
201         for tme in range(1, Nt):
202             for j in range(Nx):
203                 fi[j] = tf(tau[tme], h[j])  # заполнение начального
                    ↪  приближения
204             s = simsh(a, solve_matr[Nt - tme], ux0(tau[tme], h[0]),
                ↪  uxL(tau[tme], h[Nx - 1]), tau[tme], fi, ep)
205             solve_matr[Nt - tme - 1] = s
206         return solve_matr
207
208
209     # вычисление погрешности найденного решения
210     def error_pis(func, ut0, ux0, uxL, tf, ep):
211         t_matr = np.zeros((Nt, Nx))
212         for i in range(Nt - 1, -1, -1):
213             for j in range(Nx):
214                 t_matr[i][j] = func(tau[Nt - i - 1], h[j])
215         nt_matr = solve_pis(ut0, ux0, uxL, tf, ep)
216         print(np.max(np.abs(nt_matr - t_matr)))
217
218
219     # вычисление погрешности найденного решения
220     def error_simsh(func, ut0, ux0, uxL, tf, ep):
221         t_matr = np.zeros((Nt, Nx))
222         for i in range(Nt - 1, -1, -1):
223             for j in range(Nx):
224                 t_matr[i][j] = func(tau[Nt - i - 1], h[j])
225         nt_matr = solve_simshame(ut0, ux0, uxL, tf, ep)
226         print(np.max(np.abs(nt_matr - t_matr)))
227
228
229     start_time = time.time()
230     error_simsh(tf3, yt0_tf3, yx0_tf3, yxL_tf3, fx_tf3, 0.000001)
231     print('Simsh = ', time.time() - start_time)
```