Московский авиационный институт (национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Численные методы»

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

Дата:
Оценка:
Подпись:

Москва, 2025

1 Формулировка задачи №5

Задача: Используя явную и неявную конечно-разностные схемы, а также схему Кранка - Николсона, решить начально-краевую задачу для дифференциального уравнения параболического типа. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком. 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.

Вариант 10:

$$\frac{\partial u}{\partial t} = a \cdot \frac{\partial^2 u}{\partial x^2} + b \cdot \frac{\partial u}{\partial y} + c \cdot u$$

$$a > 0, \quad b > 0, \quad c < 0$$

$$u'_x(0,t) + u(0,t) = e^{(c-a)t} \cdot (\cos(bt) + \sin(bt))$$

$$u'_x(\pi,t) + u(\pi,t) = -e^{(c-a)t} \cdot (\cos(bt) + \sin(bt))$$

$$u(x,0) = \sin x$$

Аналитическое решение:

$$U(x,t) = e^{(c-a)t} \cdot \sin(x + bt)$$

2 Теория

Уравнения параболического типа

Классическим примером уравнения параболического типа является уравнение теплопроводности (диффузии). В одномерном случае уравнение имеет вид:

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2} + f(x,t)$$

где $u(x,t)$ – искомая функция (температура, концентрация), a^2 – коэффициент температуропроводности (диффузии), $f(x,t)$ – функция источников.

Постановка краевых задач

Для уравнения параболического типа рассматриваются три типа граничных условий:
Первого рода (условия Дирихле):

$$u(0,t) = \phi_0(t), \quad u(l,t) = \phi_l(t)$$

Второго рода (условия Неймана):

$$\frac{\partial u(0,t)}{\partial x} = \phi_0(t), \quad \frac{\partial u(l,t)}{\partial x} = \phi_l(t)$$

Третьего рода (смешанные условия):

$$\beta_0 \frac{\partial u(0,t)}{\partial x} + \alpha_0 u(0,t) = \phi_0(t)$$

$$\beta_l \frac{\partial u(l,t)}{\partial x} + \alpha_l u(l,t) = \phi_l(t)$$

Начальное условие:

$$u(x,0) = \psi(x), \quad 0 \le x \le l$$

Метод конечных разностей

Вводится равномерная сетка в области $[0,l] \times [0,T]$:

$$x_j = jh, \quad j = 0,1,\ldots,N; \quad h = l/N$$

$$t_k = k\tau, \quad k = 0,1,\ldots,K; \quad \tau = T/K$$

Аппроксимации производных:

$$\frac{\partial u}{\partial t} \approx \frac{u_j^{k+1} - u_j^k}{\tau} + O(\tau)$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + O(h^2)$$

Конечно-разностные схемы

Явная схема

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a^2 \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + f_j^k$$

или в развернутой форме:

$$u_j^{k+1} = u_j^k + \sigma(u_{j+1}^k - 2u_j^k + u_{j-1}^k) + \tau f_j^k$$

где $\sigma = \dfrac{a^2\tau}{h^2}$ – число Куранта.

Обладает условной устойчивостью: $\sigma \leq \dfrac{1}{2}$

Неявная схема

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a^2 \frac{u_{j+1}^{k+1} - 2u_j^{k+1} + u_{j-1}^{k+1}}{h^2} + f_j^{k+1}$$

или:

$$-\sigma u_{j-1}^{k+1} + (1 + 2\sigma)u_j^{k+1} - \sigma u_{j+1}^{k+1} = u_j^k + \tau f_j^{k+1}$$

Обладает абсолютной устойчивостью.

Схема Кранка-Николсона

$$\frac{u_j^{k+1} - u_j^k}{\tau} = \frac{a^2}{2}\left(\frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + \frac{u_{j+1}^{k+1} - 2u_j^{k+1} + u_{j-1}^{k+1}}{h^2}\right)$$

Порядок аппроксимации: $O(\tau^2 + h^2)$, схема абсолютно устойчива.

Аппроксимация граничных условий

Для граничных условий третьего рода на левой границе:

$$\beta_0 \frac{u_1^{k+1} - u_0^{k+1}}{h} + \alpha_0 u_0^{k+1} = \phi_0^{k+1} + O(h)$$

Для повышения точности до второго порядка используются разложения в ряд Тейлора:

$$u_1^{k+1} = u_0^{k+1} + h \frac{\partial u}{\partial x}\bigg|_0^{k+1} + \frac{h^2}{2} \frac{\partial^2 u}{\partial x^2}\bigg|_0^{k+1} + O(h^3)$$

Анализ устойчивости

Устойчивость схемы определяется выполнением условия:

$$\|u^{k+1}\| \leq \|u^k\|$$

Для явной схемы необходимое условие устойчивости:

$$\frac{a^2 \tau}{h^2} \leq \frac{1}{2}$$

Неявная схема и схема Кранка-Николсона абсолютно устойчивы.

3 Исходный код

Здесь располагается реализация задачи.

```
# %%
import math
import numpy as np
import matplotlib.cm
import matplotlib.pyplot as plt
from tld import tridiagonal_method

# %%
l = np.pi
n = 100
T = 1
theta = 1

# %% [markdown]
#
```

```
16
17 # %%
18 a = 1
19 b = 1
20 c = -1
21
22 # %% [markdown]
23 #
24
25 # %%
26 alpha_0 = 1
27 beta_0 = 1
28
29 alpha_l = 1
30 beta_l = 1
31
32 # %%
33 def u0(x):
34     return np.sin(x)
35
36 def gamma_0(t):
37     return np.exp((c - a) * t) * (np.cos(b * t) + np.sin(b * t))
38
39 def gamma_l(t):
40     return -np.exp((c - a) * t) * (np.cos(b * t) + np.sin(b * t))
41
42 # %%
43 def analytical(x, t):
44     return np.exp((c - a) * t) * np.sin(x + b * t)
45
46 # %%
47 h = l / (n - 1)
48 d_max = h**2 / (2 * a**2)
49 K = math.ceil(T / d_max) + 1
50
51 delta = T / (K - 1)
52
53 # %%
54 t = [j * delta for j in range(K - 1)]
55 t.append(T)
56 t = np.array(t)
57
58 x = [i * h for i in range(n - 1)]
59 x.append(l)
60 x = np.array(x)
61
62 # %%
63 bound_0t = []
64 bound_x0 = []
```

```
65  bound_lt = []
66
67  for i in range(n):
68      bound_x0.append(u0(x[i]))
69
70  for k in range(K):
71      bound_0t.append(gamma_0(t[k]))
72      bound_lt.append(gamma_l(t[k]))
73
74  # %%
75  def explicit_2d1o(u, k, n):
76      return (bound_0t[k + 1] - alpha_0 * u[1] / h) / (beta_0 - alpha_0 / h), \
77             (bound_lt[k + 1] + alpha_l * u[n - 2] / h) / (alpha_l / h + beta_l)
78
79  # %%
80  def explicit_3d2o(u, k, n):
81      return (2 * h * bound_0t[k + 1] - (-alpha_0) * u[2] - 4 * alpha_0 * u[1] ) \
82             / (2 * h * beta_0 - 3 * alpha_0), \
83          (2 * h * bound_lt[k + 1] - alpha_l * u[n - 3] - (-4 * alpha_l) * u[n - 2]) \
84             / (2 * h * beta_l + 3 * alpha_l)
85
86  # %%
87  def explicit_2d2o(u, u_old, k, n):
88      return ((b - 2 * a / h) * bound_0t[k + 1] + (alpha_0 / delta) * u_old[0] - (-2 *
            alpha_0 * a / (h**2)) * u[1]) \
89             / (alpha_0 * (1 / delta + 2 * a / (h**2) - c) + beta_0 * (b - 2 * a / h)),
                \
90          ((b + 2 * a / h) * bound_lt[k + 1] + (alpha_l / delta) * u_old[n - 1] - (-2 *
            alpha_l * a / (h**2)) * u[n - 2]) \
91             / (alpha_l * (1 / delta + 2 * a / (h**2) - c) + beta_l * (b + 2 * a / h))
92
93  # %%
94  u = np.array(bound_x0)
95
96  def solve_explicit(u, get_boundary):
97      res = [u]
98      for k in range(K-1):
99
100         u_new = np.zeros(n)
101
102         for i in range(1, n-1):
103             u_new[i] = u[i] + delta * (
104                 a * (u[i-1] - 2*u[i] + u[i+1]) / h**2 +
105                 b * (u[i+1] - u[i-1]) / (2*h) +
106                 c * u[i]
107             )
108
109         if(get_boundary == explicit_2d2o):
110             u_new[0], u_new[n-1] = get_boundary(u_new, u, k, n)
```

```
        else:
            u_new[0], u_new[n-1] = get_boundary(u_new, k, n)

        res.append(u_new)
        u = u_new

    return res

# solve_explicit(u, explicit_2d2o)[-1].tolist()

# %%
def implicit_2d1o(u, k, n, matrix):
    return (
        beta_0 - alpha_0 / h,
        alpha_0 / h,
        bound_0t[k+1],
        -alpha_l / h,
        alpha_l / h + beta_l,
        bound_lt[k + 1]
    )

# %%
def implicit_3d2o(u, k, n, matrix):
    return (
        2 * h * beta_0 - 3 * alpha_0 - matrix[0][1] * (-alpha_0 / matrix[2][1]),
        4 * alpha_0 - matrix[1][1] * (-alpha_0 / matrix[2][1]),
        2 * h * bound_0t[k+1] - matrix[3][1] * (-alpha_0 / matrix[2][1]),
        -4 * alpha_l - matrix[1][n - 2] * (alpha_l / matrix[0][n - 2]),
        2 * h * beta_l + 3 * alpha_l - matrix[2][n - 2] * (alpha_l / matrix[0][n - 2]),
        2 * h * bound_lt[k + 1] - matrix[3][n - 2] * (alpha_l / matrix[0][n - 2])
    )

# %%
def implicit_2d2o(u, k, n, matrix):
    return (
        alpha_0 * (1 / delta + 2 * a / (h**2) - c) + beta_0 * (b - 2 * a / h),
        -2 * alpha_0 * a / (h**2),
        (b - 2 * a / h) * bound_0t[k + 1] + alpha_0 / delta * u[0],
        -2 * alpha_l * a / (h**2),
        alpha_l * (1 / delta + 2 * a / (h**2) - c) + beta_l * (b + 2 * a / h),
        (b + 2 * a / h) * bound_lt[k + 1] + (alpha_l / delta) * u[n - 1]
    )

# %%
def solve_weighted_implicit(u, theta, get_boundary):

    res = [u]

    for k in range(K-1):
```

```
        A = np.zeros(n)
        B = np.zeros(n)
        C = np.zeros(n)
        F = np.zeros(n)

        for i in range(1, n-1):
            A[i] = theta * delta * (b/(2*h) - a/(h**2)) #  u_{i-1}^{k+1}
            B[i] = theta * delta * (2*a/(h**2) - c) + 1 #  u_i^{k+1}
            C[i] = theta * delta * (-a/(h**2) - b/(2*h)) #  u_{i+1}^{k+1}

            if theta < 1.0:
                u_xx = (u[i-1] - 2*u[i] + u[i+1]) / (h**2)
                u_x = (u[i+1] - u[i-1]) / (2*h)
                rhs_old = a * u_xx + b * u_x + c * u[i]
                F[i] = u[i] + (1 - theta) * delta * rhs_old
            else:
                F[i] = u[i]

        B[0], C[0], F[0], A[n-1], B[n-1], F[n-1] = get_boundary(u, k, n, [A, B, C, F])

        u_new = tridiagonal_method([A, B, C, F])

        res.append(u_new)
        u = u_new

    return res

# solve_weighted_implicit(u, 0.5, implicit_2d2o)[-1]

# %% [markdown]
#

# %%
explicit_result = solve_explicit(u, explicit_2d1o) # explicit_3d2o explicit_2d2o

time = 125

x_plt, t_plt = np.meshgrid(x, t)
true_data = analytical(x_plt, t_plt)

plt.title(" ")
plt.plot(x, explicit_result[time], label="u(x, t)")
plt.plot(x, true_data[time], label="u*(x, t)")
plt.legend()
plt.show()

# %% [markdown]
#

```

```
# %%
implicit_result = solve_weighted_implicit(u, 1, implicit_3d2o) # implicit_2d1o
    implicit_2d2o
time = 255

x_plt, t_plt = np.meshgrid(x, t)
true_data = analytical(x_plt, t_plt)

plt.title(" ")
plt.plot(x, implicit_result[time], label="u(x, t)")
plt.plot(x, true_data[time], label="u*(x, t)")
plt.legend()
plt.show()

# %% [markdown]
#  -

# %%
nikol_result = solve_weighted_implicit(u, 0.5, implicit_3d2o) # implicit_2d1o
    implicit_2d2o

time = 108

x_plt, t_plt = np.meshgrid(x, t)
true_data = analytical(x_plt, t_plt)

plt.title(" -")
plt.plot(x, nikol_result[time], label="u(x, t)")
plt.plot(x, true_data[time], label="u*(x, t)")
plt.legend()
plt.show()

# %%
fig = plt.figure(figsize = (20, 20))
ax = fig.add_subplot(221, projection='3d')

ax.plot_surface(t_plt, x_plt, np.array(nikol_result), cmap='viridis')

ax.set_xlabel('t')
ax.set_ylabel('x')

ax2 = fig.add_subplot(222, projection='3d')

ax2.plot_surface(t_plt, x_plt, true_data, cmap='viridis')

ax2.set_xlabel('t')
ax2.set_ylabel('x')

plt.show()
```

```
256
257
258 # %% [markdown]
259 #
260
261 # %%
262 data = explicit_result
263
264 err = np.abs(data - true_data)
265
266 plt.title("  t")
267 plt.plot(t, np.max(err, axis = 1))
268 plt.show()
```

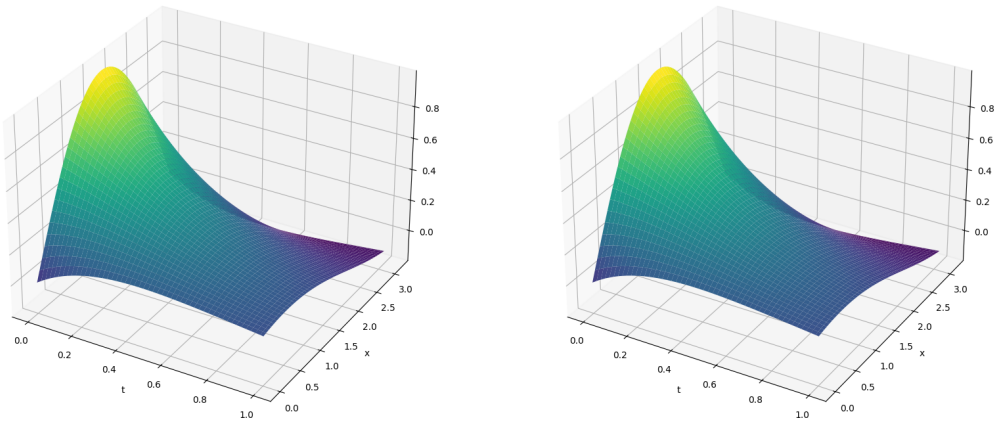

Рис. 1: Визуализация аналитического и численного решения

4 Результат

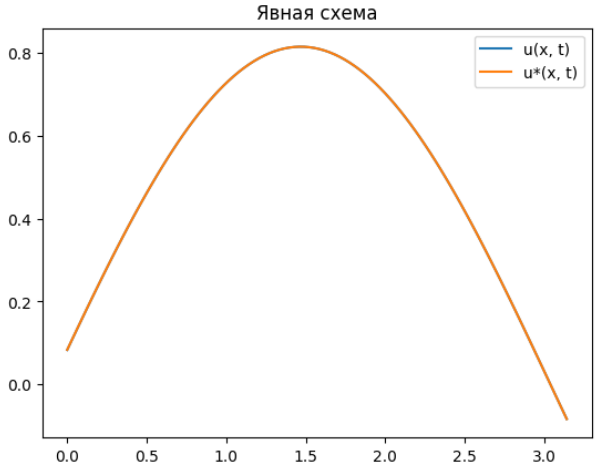

Рис. 2: Явная схема

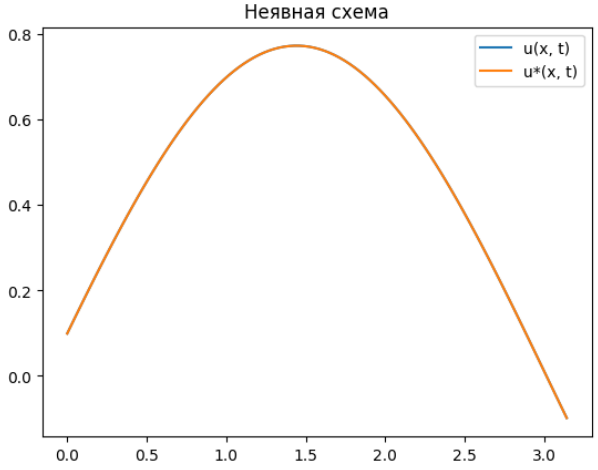

Рис. 3: Неявная схема

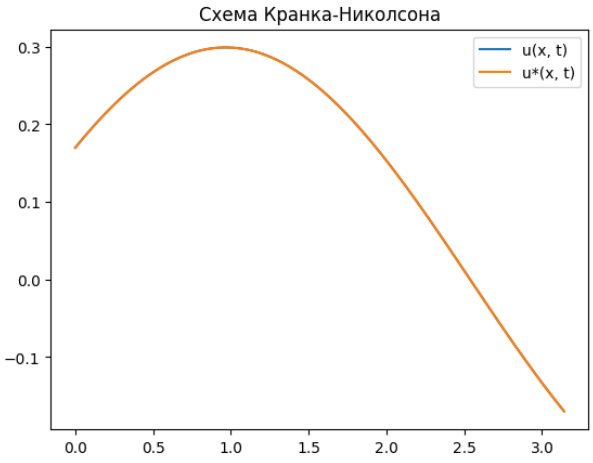

Рис. 4: Схема Кранка-Николсона

5 Выводы

В ходе выполнения лабораторной работы реализованы явная, неявная схема, а также схема Кранка-Николсона. Также были реализованы методы аппроксимации 1го и 2го порядка по двум и трем точкам.

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.