Московский авиационный институт

(национальный исследовательский университет)

Институт № 8 «Информационные технологии и прикладная математика»

Лабораторная работа № 5

по курсу «Численные методы»

«Численное решение уравнений с частными производными параболического типа»

**Выполнил:**

Гришин П. Ф.

**Группа:**

М8О-401Б-21

**Преподаватель:**

Ревизников Д. Л.

Москва, 2025

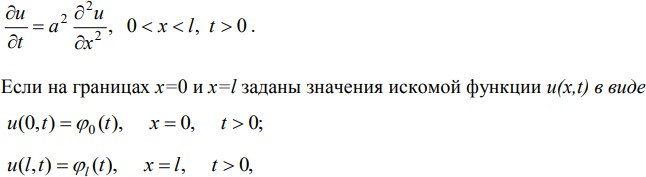
# Задача

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.

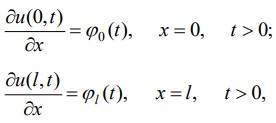
# Описание метода

Классический пример уравнения параболического типа – уравнение теплопроводности, которое в одномерном по пространству случае имеет вид:

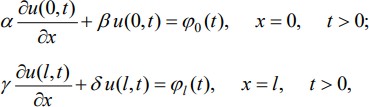


- граничные условия первого рода. И заданы начальные условия:



Также на границах могут быть заданы значения производных искомой функции по пространственной переменной, то есть граничные условия второго рода:

Или граничные условия третьего рода, то есть линейные комбинации искомой функции и ее производной по пространственной переменной:



Для решения такой задачи применяют метод конечных разностей.

Для этого вводится понятие разностной сетки с пространственным шагом h и временным τ

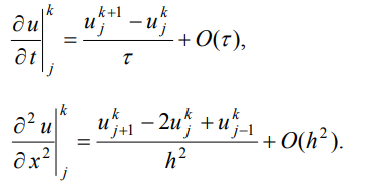
Также вводится понятие сеточной функции – однозначное отображение целых аргументов j, k в значения функции 𝑢𝑘 = 𝑢(𝑥𝑗, 𝑡𝑘).

Вводится два временных слоя: нижний 𝑡𝑘 = 𝑘τ, на котором распределение искомой функции известно, и верхний 𝑡𝑘+1 = (𝑘 + 1)τ, на котором распределение искомой функции подлежит определению.

𝑗

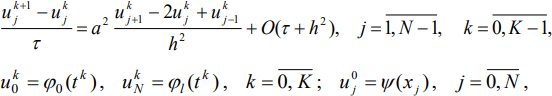
Далее аппроксимируем дифференциальные операторы отношением конечных разностей.

При аппроксимации второй производной по пространству на нижнем временном слое, получаем явную конечно-разностную схему.



Явная конечно-разностная схема:

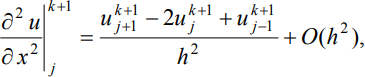
𝑗



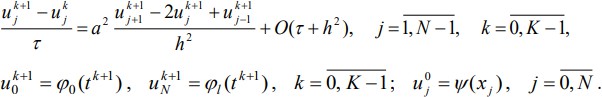
где для каждого уравнения неизвестна только одна величина 𝑢𝑘+1, которую можно явно

выразить:

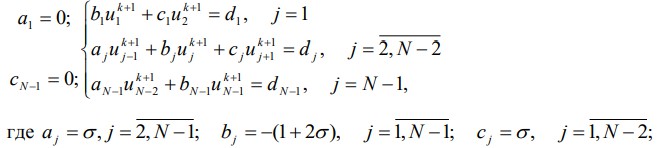
Данная схема будет устойчива при условии σ <= ½.

При аппроксимации второй производной по пространству на верхнем временном слое, получаем неявную конечно-разностную схему.

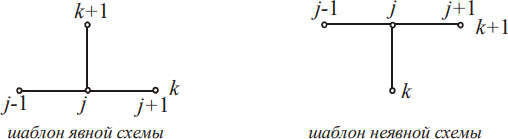
Неявная конечно-разностная схема:



где для нахождения сеточной функции на верхнем временном слое необходимо решить СЛАУ с трехдиагональной матрицей:



Приведем шаблоны конечно-разностных схем (их геометрическую интерпретацию на конечно-разностной сетке):



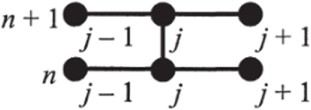
Явно-неявная схема имеет вид:



где ϴ - вес неявной части, причем 0≤ϴ≤1. При ϴ=0 получаем явную схему, при ϴ=1 – неявную схему, при ϴ=1/2 – схему Кранка-Николсона.

Как и в неявной схеме для решения явно-неявной необходимо решать СЛАУ с трехдиагональной матрицей.

Шаблон схемы Кранка-Николсона:

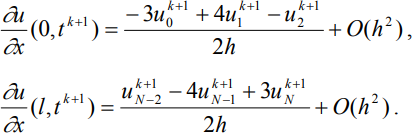


Также рассмотрим три варианта аппроксимации граничных условий, содержащих производные.

Двухточечная аппроксимация с первым порядком:



Трехточечная аппроксимация со вторым порядком:

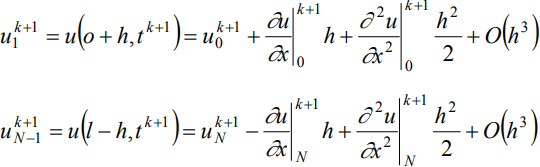


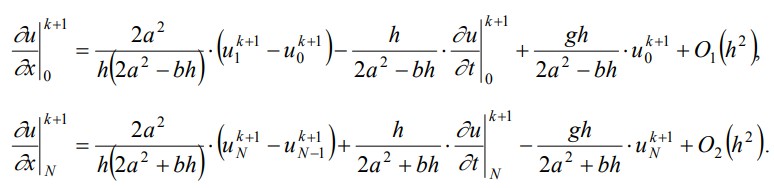
Двухточечная аппроксимация со вторым порядком:

Для получения формул разложим 𝑢𝑘+1в ряд Тейлора в окрестности x = 0 и 𝑢𝑘+1 в

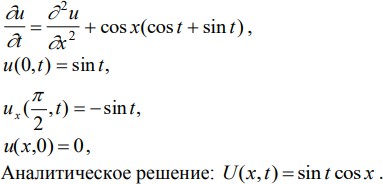
1 𝑁

окрестности x = l:



Далее при помощи информации из исходного уравнения, выражая оттуда вторую производную и подставляя полученные выражения, получим:

# Вариант



**Решение и код**

**Вспомогательные функции:**  
import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

def f(x, t):

    return np.cos(x) \* (np.cos(t) + np.sin(t))

# Функции для начальных и краевых условий

def f\_t(t):

    return np.sin(t)

def f\_x(x):

    return 0

def q\_l(t):

    return -np.sin(t)

# Метод прогонки для решения трехдиагональной системы

def solve\_tridiagonal(a, b, c, d):

    n = len(d)

    c\_new = np.zeros(n - 1)

    d\_new = np.zeros(n)

    c\_new[0] = c[0] / b[0]

    d\_new[0] = d[0] / b[0]

    for i in range(1, n):

        denom = b[i] - a[i-1] \* c\_new[i-1]

        if i < n - 1:

            c\_new[i] = c[i] / denom

        d\_new[i] = (d[i] - a[i-1] \* d\_new[i-1]) / denom

    x = np.zeros(n)

    x[-1] = d\_new[-1]

    for i in range(n-2, -1, -1):

        x[i] = d\_new[i] - c\_new[i] \* x[i+1]

    return x

def u\_exact(x, t):

    return np.sin(t) \* np.cos(x)

#/////////////////////////////////////////////////////  
alpha = 1

L = np.pi / 2

T = 3.0

N = 9

M = 1000

h = L / N

tau = T / M

if alpha \* tau / h\*\*2 > 0.5:

    raise ValueError("Схема неустойчива: уменьшите tau или увеличьте h.")

u = np.zeros((M + 1, N + 1))

x = np.linspace(0, L, N + 1)

t = np.linspace(0, T, M + 1)

u[0, :] = f\_x(x)

for n in range(M):

    u[n + 1, 0] = f\_t((n + 1) \* tau)

    # Основное разностное уравнение

    for i in range(1, N):

        u[n + 1, i] = u[n, i] + (alpha \* tau / h\*\*2) \* (u[n, i + 1] - 2 \* u[n, i] + u[n, i - 1]) + tau \* f(x[i], t[n])

    method = 1

    # Граничное условие для x = pi/2

    # 1. Двухточечная аппроксимация с первым порядком

    if method == 1:

        u[n + 1, -1] = u[n + 1, -2] + q\_l((n + 1) \* tau) \* h

    # 2. Трехточечная аппроксимация со вторым порядком

    if method == 2:

        u[n + 1, -1] = (4 \* u[n + 1, -2] - u[n + 1, -3] + 2 \* q\_l((n + 1) \* tau) \* h) / 3

    # 3. Двухточечная аппроксимация со вторым порядком

    if method == 3:

        u[n + 1, -1] = u[n + 1, -2] + h \* q\_l((n + 1) \* tau) + (tau / 2) \* (q\_l((n + 1) \* tau) - q\_l(n \* tau))

U\_exact = np.array([[np.sin(ti) \* np.cos(xi) for xi in x] for ti in t])

X, T = np.meshgrid(x, t)

fig = plt.figure(figsize=(14, 6))

# Левый график: Численное решение

ax1 = fig.add\_subplot(121, projection='3d')

surf1 = ax1.plot\_surface(X, T, u, cmap='viridis', alpha=0.8)

fig.colorbar(surf1, ax=ax1, shrink=0.5, aspect=10)

ax1.set\_title("Численное решение")

ax1.set\_xlabel('x (позиция вдоль стержня)')

ax1.set\_ylabel('t (время)')

ax1.set\_zlabel('u(x, t) (температура)')

# Правый график: Аналитическое решение

ax2 = fig.add\_subplot(122, projection='3d')

surf2 = ax2.plot\_surface(X, T, U\_exact, cmap='plasma', alpha=0.8)

fig.colorbar(surf2, ax=ax2, shrink=0.5, aspect=10)

ax2.set\_title("Аналитическое решение")

ax2.set\_xlabel('x (позиция вдоль стержня)')

ax2.set\_ylabel('t (время)')

ax2.set\_zlabel('u(x, t) (температура)')

error = np.abs(u - U\_exact)

print(f"Максимальная погрешность: {np.max(error)}\n\n")

plt.tight\_layout()

plt.show()

#/////////////////////////////////////////////////////

Максимальная погрешность: 0.004694693025208781

Изображение выглядит как снимок экрана, Красочность, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

#/////////////////////////////////////////////////////

L = np.pi / 2

T = 3.0

Nx = 50  # Количество узлов по x

Nt = 500  # Количество шагов по времени

dx = L / (Nx - 1)

dt = T / Nt

alpha = dt / dx\*\*2  # Коэффициент схемы

x = np.linspace(0, L, Nx)

t = np.linspace(0, T, Nt + 1)

U = np.zeros((Nt + 1, Nx))

U[0, :] = 0

# Неявная схема

for n in range(Nt):

    a = -alpha \* np.ones(Nx - 2)

    b = (1 + 2 \* alpha) \* np.ones(Nx - 1)

    c = -alpha \* np.ones(Nx - 2)

    d = U[n, 1:-1] + dt \* f(x[1:-1], t[n+1])

    # Граничные условия

    d[0] += alpha \* np.sin(t[n+1])

    g\_t = -np.sin(t[n+1])  # Производная на правом краю

    method = 2

    if method == 1:  # Двухточечная аппроксимация 1-го порядка

        d[-1] -= alpha \* dx \* g\_t

        U[n+1, -1] = U[n+1, -2] - dx \* g\_t

    elif method == 2:  # Трёхточечная аппроксимация 2-го порядка

        b[-1] = 1 + 2 \* alpha

        d[-1] = U[n, -2] + dt \* f(x[-2], t[n+1]) + 2 \* dx \* alpha \* g\_t

        U[n+1, -1] = (4 \* U[n+1, -2] - U[n+1, -3]) / 3

    elif method == 3:  # Двухточечная аппроксимация 2-го порядка

        b[-1] = 1 + 2 \* alpha

        d[-1] = U[n, -2] + dt \* f(x[-2], t[n+1]) + dx \* alpha \* g\_t

        U[n+1, -1] = (3 \* U[n+1, -2] - U[n+1, -3]) / 2

    U[n+1, 1:-1] = solve\_tridiagonal(a, b, c, d)

    # Граничные условия

    U[n+1, 0] = np.sin(t[n+1])

U\_exact = np.array([[u\_exact(xi, ti) for xi in x] for ti in t])

X, T = np.meshgrid(x, t)

fig = plt.figure(figsize=(14, 6))

# Левый график: Численное решение

ax1 = fig.add\_subplot(121, projection='3d')

surf1 = ax1.plot\_surface(X, T, U, cmap='viridis', alpha=0.8)

fig.colorbar(surf1, ax=ax1, shrink=0.5, aspect=10)

ax1.set\_title("Численное решение")

ax1.set\_xlabel('x (позиция)')

ax1.set\_ylabel('t (время)')

ax1.set\_zlabel('u(x, t)')

# Правый график: Аналитическое решение

ax2 = fig.add\_subplot(122, projection='3d')

surf2 = ax2.plot\_surface(X, T, U\_exact, cmap='plasma', alpha=0.8)

fig.colorbar(surf2, ax=ax2, shrink=0.5, aspect=10)

ax2.set\_title("Аналитическое решение")

ax2.set\_xlabel('x (позиция)')

ax2.set\_ylabel('t (время)')

ax2.set\_zlabel('u(x, t)')

error = np.abs(U - U\_exact)

print(f"Максимальная погрешность: {np.max(error)}\n\n")

plt.tight\_layout()

plt.show()

#/////////////////////////////////////////////////////

Максимальная погрешность: 0.0626763187881481

Изображение выглядит как снимок экрана, Красочность, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

#/////////////////////////////////////////////////////

L = np.pi / 2

T = 1.0

Nx = 100  # Количество узлов по x

Nt = 500  # Количество шагов по времени

dx = L / (Nx - 1)

dt = T / Nt

alpha = dt / (2 \* dx\*\*2)  # Коэффициент схемы Кранка-Николсона

x = np.linspace(0, L, Nx)

t = np.linspace(0, T, Nt + 1)

U = np.zeros((Nt + 1, Nx))

U[0, :] = 0

for n in range(Nt):

    a = -alpha \* np.ones(Nx - 2)

    b = (1 + 2 \* alpha) \* np.ones(Nx - 1)

    c = -alpha \* np.ones(Nx - 2)

    d = alpha \* U[n, :-2] + (1 - 2 \* alpha) \* U[n, 1:-1] + alpha \* U[n, 2:] + \

        (dt / 2) \* (f(x[1:-1], t[n+1]) + f(x[1:-1], t[n]))

    # Граничные условия

    d[0] += alpha \* np.sin(t[n+1])  # Левое граничное условие

    g\_t = -np.sin(t[n+1])  # Производная на правом краю

    method = 1

    # Выбор метода аппроксимации

    if method == 1:  # Двухточечная аппроксимация 1-го порядка

        d[-1] -= alpha \* dx \* g\_t

        U[n+1, -1] = U[n+1, -2] - dx \* g\_t

    elif method == 2:  # Трёхточечная аппроксимация 2-го порядка

        d[-1] = U[n, -2] + dt \* f(x[-2], t[n+1]) + 2 \* dx \* alpha \* g\_t

        U[n+1, -1] = (4 \* U[n+1, -2] - U[n+1, -3]) / 3 - (2 \* dx / 3) \* g\_t

    elif method == 3:  # Двухточечная аппроксимация 2-го порядка

        d[-1] = U[n, -2] + dt \* f(x[-2], t[n+1]) + dx \* alpha \* g\_t

        U[n+1, -1] = (3 \* U[n+1, -2] - U[n+1, -3]) / 2 - dx \* g\_t

    U[n+1, 1:-1] = solve\_tridiagonal(a, b, c, d)

    # Граничные условия

    U[n+1, 0] = np.sin(t[n+1])

U\_exact = np.array([[u\_exact(xi, ti) for xi in x] for ti in t])

X, T = np.meshgrid(x, t)

fig = plt.figure(figsize=(14, 6))

# Левый график: Численное решение

ax1 = fig.add\_subplot(121, projection='3d')

surf1 = ax1.plot\_surface(X, T, U, cmap='viridis', alpha=0.8)

fig.colorbar(surf1, ax=ax1, shrink=0.5, aspect=10)

ax1.set\_title("Численное решение")

ax1.set\_xlabel('x (позиция)')

ax1.set\_ylabel('t (время)')

ax1.set\_zlabel('u(x, t)')

# Правый график: Аналитическое решение

ax2 = fig.add\_subplot(122, projection='3d')

surf2 = ax2.plot\_surface(X, T, U\_exact, cmap='plasma', alpha=0.8)

fig.colorbar(surf2, ax=ax2, shrink=0.5, aspect=10)

ax2.set\_title("Аналитическое решение")

ax2.set\_xlabel('x (позиция)')

ax2.set\_ylabel('t (время)')

ax2.set\_zlabel('u(x, t)')

error = np.abs(U - U\_exact)

print(f"Максимальная погрешность: {np.max(error)}\n\n")

plt.tight\_layout()

plt.show()

#/////////////////////////////////////////////////////

Максимальная погрешность: 0.013351308404451747

Изображение выглядит как снимок экрана, Красочность, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

# Выводы

В ходе выполнения данной работы для решения начально-краевой задачи для

дифференциального уравнения параболического типа были реализованы явная и неявная конечно-разностные схемы, схема Кранка-Николсона. Также реализованы три варианта аппроксимации начальных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная со вторым и двухточечная со вторым.

Путем сравнения результатов с приведенным точным решением была вычислена погрешность методов путем нахождения максимальной ошибки.