Московский авиационный институт (Национальный исследовательский университет) Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Отчет по лабораторным работам по курсу «Численные методы»

Вариант 2

Выполнил: Баранов А.Д.
Группа: М8О-408Б-20
Проверил: проф. Пивоваров Д.Е.
Дата:
Оценка:

Москва, 2023

ЛАБОРАТОРНАЯ РАБОТА №1. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ПАРАБОЛИЧЕСКОГО ТИПА

Задание

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 2

$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2}, a > 0,$

$u(0,t) = 0,$

$u(1,t) = 1,$

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

Аналитическое решение: $U(x,t) = x + \exp(-\pi^2 at)\sin(\pi x)$

Ход решения

Так как на границах $x = 0$ и $x = l = 1$ заданы краевые условия вида $u(0,t) = 0, u(1,t) = 1$, т.е. граничные условия первого рода, и, кроме того, заданы начальные условия $u(x,0) = x + \sin \pi x$ то решаемую задачу называют первой начально-краевой задачей для уравнения теплопроводности.

Подобные задачи решаются методом конечных разностей. Для этого на пространственно-временную область $0 \le x \le l, 0 \le t \le T$ наносится конечно-разностная сетка $\omega_{h\tau} = \{x_j = jh, j = \overline{0,N}; t^k = k\tau, k = \overline{0,K}\}$ с пространственным шагом $h = \frac{l}{N}$ и шагом по времени $\tau = \frac{T}{K}$.

Сеточная функция u_j^k задачи – однозначное отображение целых аргументов j, k в значение функции $u_j^k = u(x_j, t^k)$.

На введенной сетке введем сеточные функции u_j^k, u_j^{k+1}, первая из которых известна, вторая – подлежит определению. Для её нахождения аппроксимируем $\frac{\partial u}{\partial t}$ и $\frac{\partial^2 u}{\partial x^2}$:

$$\left.\frac{\partial u}{\partial t}\right|_j^k = \frac{u_j^{k+1} - u_j^k}{\tau} + O(\tau),$$

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_j^k = \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + O(h^2).$$

Заметим, так как в моем варианте даны условия первого рода, мы можем сразу же заполнить первый и последний столбцы сетки – u_0^k и u_N^k соответственно, а также её нижнюю строку u_j^0.

Подставляя полученные аппроксимации в исходное уравнение получаем <u>явную конечно-разностную схему:</u>

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a^2 \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + O(\tau + h^2), j = \overline{1, N-1}, k = \overline{1, K-1},$$

$$u_0^k = 0, u_N^k = 1, k = \overline{0, K}, u_j^0 = x_j + \sin \pi x_j\,, j = \overline{0, N}.$$

Из всех переменных нам неизвестна только u_j^{k+1}. Выразим её явно:

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

Схема хороша тем, что позволяет получить ответ сразу, не решая СЛАУ, однако также она обладает существенным недостатком – она является условно устойчивой с условием $\sigma \leq \frac{1}{2}$.

Если в исходное уравнение подставить следующую аппроксимацию:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_j^{k+1} = \frac{u_{j+1}^{k+1} - 2u_j^{k+1} + u_{j-1}^{k+1}}{h^2} + O(h^2),$$

то получим <u>неявную конечно-разностную схему:</u>

$$\frac{u_j^{k+1}-u_j^k}{\tau}=a^2\frac{u_{j+1}^{k+1}-2u_j^{k+1}+u_{j-1}^{k+1}}{h^2}+O(\tau+h^2), j=\overline{1,N-1}, k=\overline{1,K-1},$$

$$u_0^{k+1}=0, u_N^{k+1}=1, k=\overline{0,K}, u_j^0=x_j+\sin\pi x_j, j=\overline{0,N}.$$

Домножим обе стороны равенства на τ:

$$u_j^{k+1}-u_j^k=\sigma\left(u_{j+1}^{k+1}-2u_j^{k+1}+u_{j-1}^{k+1}\right),$$

сгруппируем и окончательно получаем:

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

где $a_j=\sigma, b_j=-(2\sigma+1), c_j=\sigma$ – коэффициенты трехдиагональной матрицы, $d_j=-u_j^k$ – компоненты вектора правой стороны. Причем стоит учитывать, что $d_1=-u_1^k, d_{N-1}=-(u_{N-1}^k+\sigma)$. Полученную СЛАУ лучше всего решать методом прогонки, так как матрица получилась трехдиагональной.

Рассмотрим неявно-явную схему:

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

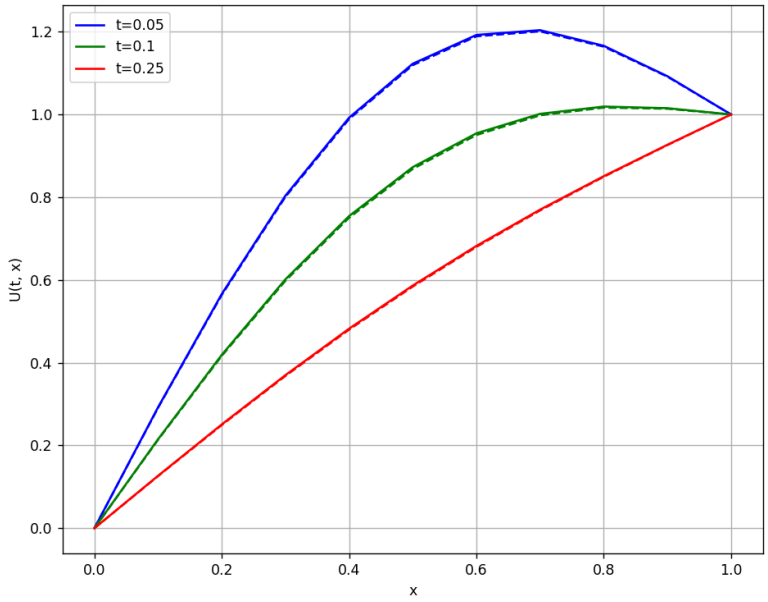
где θ – вес неявной части конечно-разностной схемы, $(1-\theta)$ – вес для явной части, причем $0\le\theta\le1$. При $\theta=\frac{1}{2}$ имеем схему Кранка-Николсона. Она абсолютно устойчива и имеет второй порядок аппроксимации по времени и пространственной переменной x.

Производя аналогичные преобразования получаем СЛАУ, где коэффициенты трёхдиагональной матрицы равны: $a_j=\sigma\theta, b_j=-(2\sigma\theta+1), c_j=\sigma\theta$, а компоненты вектора правой стороны - $d_j=-(1-\theta)\sigma u_{j+1}^k+(2\sigma(1-\theta)-1)u_j^k-(1-\theta)\sigma u_{j-1}^k$. Аналогично, стоит учесть, что $d_1=-(1-\theta)\sigma\left(u_2^k+u_0^k\right)+(2\sigma(1-\theta)-1)u_1^k, d_{N-1}=-(1-\theta)\sigma\left(u_N^k+u_{N-2}^k\right)+(2\sigma(1-\theta)-1)*u_{N-1}^k-\sigma\theta$.

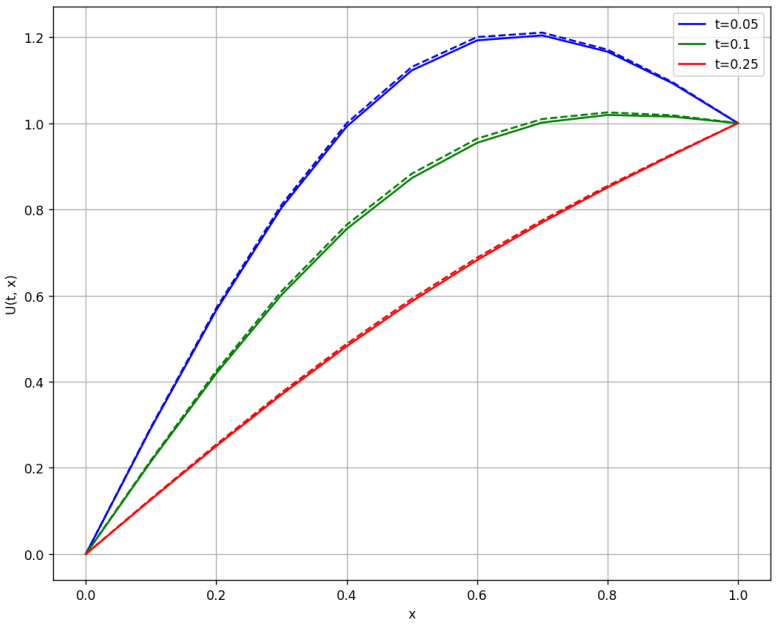
Результат работы программы

Программа считает задачу с одним заранее заданным $\sigma=\frac{2}{5}$. На основании этого значения она рассчитывае шаги по времени и координате таким об-

разом, чтобы выполнялось условие устойчивости для явного метода. На графики она выводит значение функции при трёх различных фиксированных t. Сплошной линией на графике изображено точное решение, пунктирной – рассчитанное численно. Для построения графика погрешностей методов перебирается 5 различных разбиений сетки по координате x.



Явная схема



Неявная схема

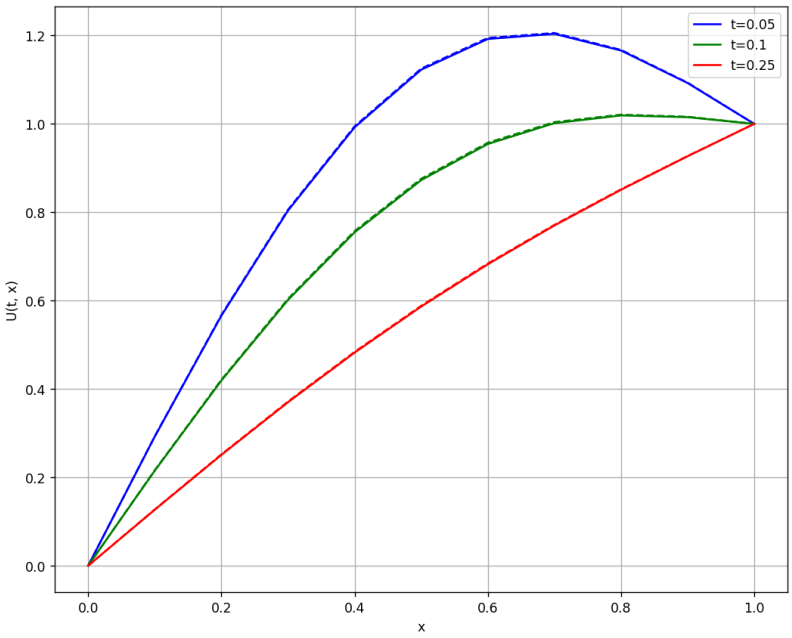


Схема Кранка-Николсона

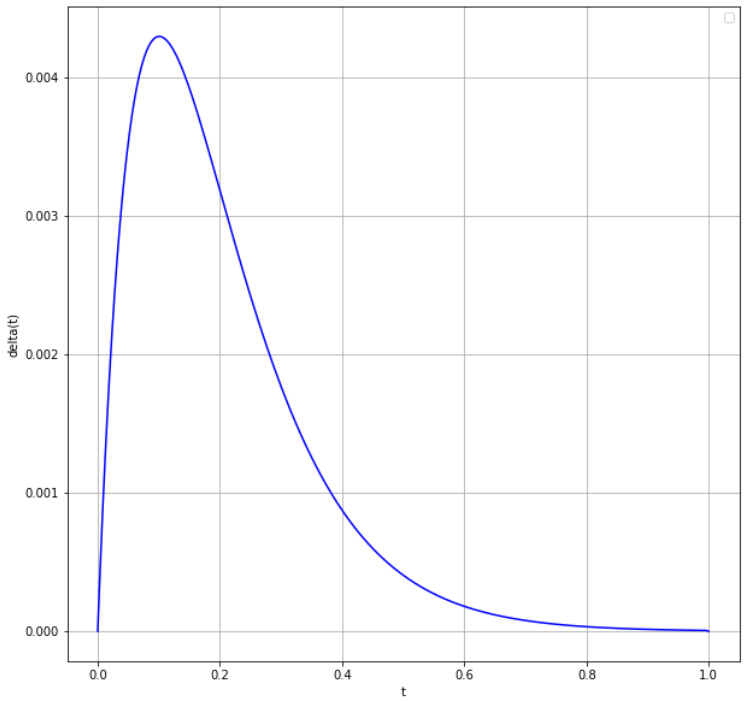


График погрешностей

Вывод

При решении уравнения каждым из трёх методов при выбранной σ решение оказалось достаточно точным. Наилучшим образом показал себя метод Кранка-Николсона. При увеличении длины шага по координате x погрешность вычислений закономерно увеличивается.

Код программы

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

FIGSIZE = 10
COLORS = ['blue', 'green', 'red']

plt.rcParams['figure.figsize'] = [FIGSIZE, FIGSIZE]

L = 1
U0 = 0
UL = 1
a = 1
T = 1
N = 10
sigma = 0.4
theta = 0.5
h = L / N
tau = sigma * h ** 2 / a
K = int(round(T / tau))


def U(x, t):
    return x + np.exp(-np.pi ** 2 * t) * np.sin(np.pi * x)


def psi(x):
    return x + np.sin(np.pi * x)

def Check(A):
    if np.shape(A)[0] != np.shape(A)[1]:
        return False
    n = np.shape(A)[0]
    for i in range(n):
        sum = 0
        for j in range(n):
            if i != j:
                sum += abs(A[i][j])
        if abs(A[i][i]) < sum:
            return False
    return True


def calc(a, b):
    if (Check(a)):
        p = np.zeros(len(b))
        q = np.zeros(len(b))
        p[0] = -a[0][1] / a[0][0]
        q[0] = b[0] / a[0][0]
        for i in range(1, len(p) - 1):
            p[i] = -a[i][i + 1] / (a[i][i] + a[i][i - 1] * p[i - 1])
            q[i] = (b[i] - a[i][i - 1] * q[i - 1]) / (a[i][i] + a[i][i - 1]
* p[i - 1])
        i = len(a) - 1
        p[-1] = 0
        q[-1] = (b[-1] - a[-1][-2] * q[-2]) / (a[-1][-1] + a[-1][-2] * p[-
2])
        x = np.zeros(len(b))
        x[-1] = q[-1]
        for i in reversed(range(len(b) - 1)):
            x[i] = p[i] * x[i + 1] + q[i]
        return x
```

```
def error(sigma, L, a, T, U):
    NArray = [5, 8, 11, 14, 17]
    size = np.size(NArray)
    hArray = np.zeros(size)
    tauArray = np.zeros(size)
    KArray = np.zeros(size)
    errors1 = np.zeros(size)
    errors2 = np.zeros(size)
    errors3 = np.zeros(size)
    for i in range(0, size):
        hArray[i] = L / NArray[i]
        tauArray[i] = np.sqrt(sigma * hArray[i] ** 2 / a)
        KArray[i] = int(round(T / tauArray[i]))
        xArray = np.arange(0, L + hArray[i], hArray[i])
        u1 = ExplicitMethod(NArray[i], int(KArray[i]), sigma)
        u2 = ImplicitMethod(NArray[i], int(KArray[i]), sigma)
        u3 = CrankNickolson(NArray[i], int(KArray[i]), sigma)
        t = tauArray[i] * KArray[i] / 2
        if (np.size(xArray) != NArray[i] + 1):
            xArray = xArray[:NArray[i] + 1]
        uCorrect = U(xArray, t)
        u1Calc = u1[int(KArray[i] / 2)]
        u2Calc = u2[int(KArray[i] / 2)]
        u3Calc = u3[int(KArray[i] / 2)]
        errors1[i] = np.amax(np.abs(uCorrect - u1Calc))
        errors2[i] = np.amax(np.abs(uCorrect - u2Calc))
        errors3[i] = np.amax(np.abs(uCorrect - u3Calc))
    return NArray, errors1, errors2, errors3


def showSolution(h, tau, K, L, u, U):
    xArray = np.arange(0, L + h, h)
    fig, ax = plt.subplots()
    t = [int(K * 0.05), int(K * 0.1), int(K * 0.25)]
    COLORS = ['blue', 'green', 'red']
    for i in range(len(t)):
        uCorrect = U(xArray, t[i] * tau)
        uCalc = u[t[i]]
        plt.plot(xArray, uCorrect, color=COLORS[i], label='t=%s' %
round(t[i] * tau, 2))
        plt.plot(xArray, uCalc, color=COLORS[i], linestyle='--')
    ax.set_xlabel('x')
    ax.set_ylabel('U(t, x)')
    plt.grid()
    ax.legend()
    plt.show()


def showErrors(sigma, L, a, T, U):
    NArray, errors1, errors2, errors3 = error(sigma, L, a, T, U)
    deltaX = np.zeros(np.size(NArray))
    for i in range(np.size(NArray)):
        deltaX[i] = L / NArray[np.size(NArray) - i - 1]
    fig, ax = plt.subplots()
    plt.plot(deltaX, errors1, color=COLORS[0], label='Явный метод')
    plt.plot(deltaX, errors2, color=COLORS[1], label='Неявный метод')
    plt.plot(deltaX, errors3, color=COLORS[2], label='Метод Кранка-
Николсона')
    ax.set_xlabel('delta X')
    ax.set_ylabel('Epsilon')
    plt.grid()
    ax.legend()
    plt.show()
```

```
def ExplicitMethod(N, K, sigma):
    # Устойчивость
    u = np.zeros((K + 1, N + 1))
    for i in range(0, N):
        u[0][i] = psi(i * h)
    for i in range(0, K):
        u[i][0] = U0
        u[i][N] = UL
    if (sigma > 0.5):
        raise Exception("Измените параметры сетки")
    for k in range(1, K):
        for j in range(1, N):
            u[k][j] = sigma * u[k - 1][j + 1] + (1 - 2 * sigma) * u[k -
1][j] + sigma * u[k - 1][j - 1]
    return u


def ImplicitMethod(N, K, sigma):
    aj = sigma
    bj = -(1 + 2 * sigma)
    cj = sigma
    u = np.zeros((K + 1, N + 1))
    for i in range(0, N):
        u[0][i] = psi(i * h)
    for i in range(0, K):
        u[i][0] = U0
        u[i][N] = UL
    # Верхние слои по неявной конечно-разностной схеме
    for k in range(1, K + 1):
        matrix = np.zeros((N - 1, N - 1))
        d = np.zeros(N - 1)
        matrix[0][0] = bj
        matrix[0][1] = cj
        d[0] = -(u[k - 1][1] + sigma * U0)
        for j in range(1, N - 2):
            matrix[j][j - 1] = aj
            matrix[j][j] = bj
            matrix[j][j + 1] = cj
            d[j] = -u[k - 1][j + 1]
        matrix[N - 2][N - 3] = aj
        matrix[N - 2][N - 2] = bj
        d[N - 2] = -(u[k - 1][N - 1] + sigma * UL)
        # Решем СЛАУ методом прогонки
        ans = calc(matrix, d)
        u[k][1:N] = ans
    return u


def CrankNickolson(N, K, sigma):
    aj = sigma * theta
    bj = -(1 + 2 * sigma * theta)
    cj = sigma * theta
    u = np.zeros((K + 1, N + 1))
    for i in range(0, N):
        u[0][i] = psi(i * h)
    for i in range(0, K):
        u[i][0] = U0
        u[i][N] = UL
    for k in range(1, K + 1):
        matrix = np.zeros((N - 1, N - 1))
        d = np.zeros(N - 1)
        matrix[0][0] = bj
        matrix[0][1] = cj
        d[0] = (
            -sigma * (1 - theta) * (u[k - 1][2] + u[k - 1][0]) + (2 * sigma
```

```
* (1 - theta) - 1) * u[k - 1][1] - sigma * theta * U0
        )
        for j in range(1, N - 2):
            matrix[j][j - 1] = aj
            matrix[j][j] = bj
            matrix[j][j + 1] = cj
            d[j] = (
                -sigma * (1 - theta) * (u[k - 1][j + 2] + u[k - 1][j]) + (2
* sigma * (1 - theta) - 1) * u[k - 1][j + 1]
            )
            matrix[N - 2][N - 3] = aj
            matrix[N - 2][N - 2] = bj
            d[N - 2] = (
                -sigma * (1 - theta) * (u[k - 1][N] + u[k - 1][N - 2]) + (2
* sigma * (1 - theta) - 1) * u[k - 1][N - 1] - sigma * theta * UL
            )
            # Решем СЛАУ методом прогонки
            ans = calc(matrix, d)
            u[k][1:N] = ans
    return u


def main():
    u1 = ExplicitMethod(N, K, sigma)
    showSolution(h, tau, K, L, u1, U)
    u2 = ImplicitMethod(N, K, sigma)
    showSolution(h, tau, K, L, u2, U)
    u3 = CrankNickolson(N, K, sigma)
    showSolution(h, tau, K, L, u3, U)
    showErrors(sigma, L, a, T, U)


main()
```