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 №2. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ГИПЕРБОЛИЧЕСКОГО ТИПА

Задание

Используя явную схему крест и неявную схему, решить начально-краевую задачу для дифференциального уравнения гиперболического типа. Аппроксимацию второго начального условия произвести с первым и со вторым порядком. 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

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

$u_x(0,t) - u(0,t) = 0,$

$u_x(\pi,t) - u(\pi,t) = 0,$

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

$u_t(x,0) = -a(\sin x + \cos x).$

Аналитическое решение: $U(x,t) = \sin(x - at) + \cos(x + at)$

Ход решения

Действуем способом, аналогичным тому, что применялся в предыдущей лабораторной работе: Задаем пространственно-временную сетку и аппроксимируем производные в уравнении. Получаем явную конечно-разностную схему:

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

и неявную конечно-разностную схему:

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

Явная схема условно устойчива с условием $\sigma = \frac{a^2\tau^2}{h^2} < 1$.

Для обоих методов для вычисления u_j^{k+1} необходимо знать u_j^{k-1} и u_j^k. $u_j^0 = \sin x_j + \cos x_j$. Для нахождения u_j^1 воспользуемся аппроксимацией второго порядка начального условия:

$$\frac{u_j^1 - u_j^0}{\tau} = -a\big(\sin x_j + \cos x_j\big),$$

откуда $u_j^1 = \sin x_j + \cos x_j + -a\tau\big(\sin x_j + \cos x_j\big)$. Разложим полученный результат в ряд Тейлора. В итоге получаем:

$$u_j^1 = \left(1 - a\tau - \frac{a^2\tau^2}{2}\right)\big(\sin x_j + \cos x_j\big).$$

В результате для явной схемы получаем формулу:

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

Для неявной схемы имеем СЛАУ, которая опять же решается прогонкой, так как полученная матрица является трёхдиагональной. Для этой СЛАУ: $a_j = \sigma, b_j = -(2\sigma + 1), c_j = \sigma, d_j = -2u_j^k + u_j^{k-1}$. Причем с учетом аппроксимации первым порядком граничных условий:

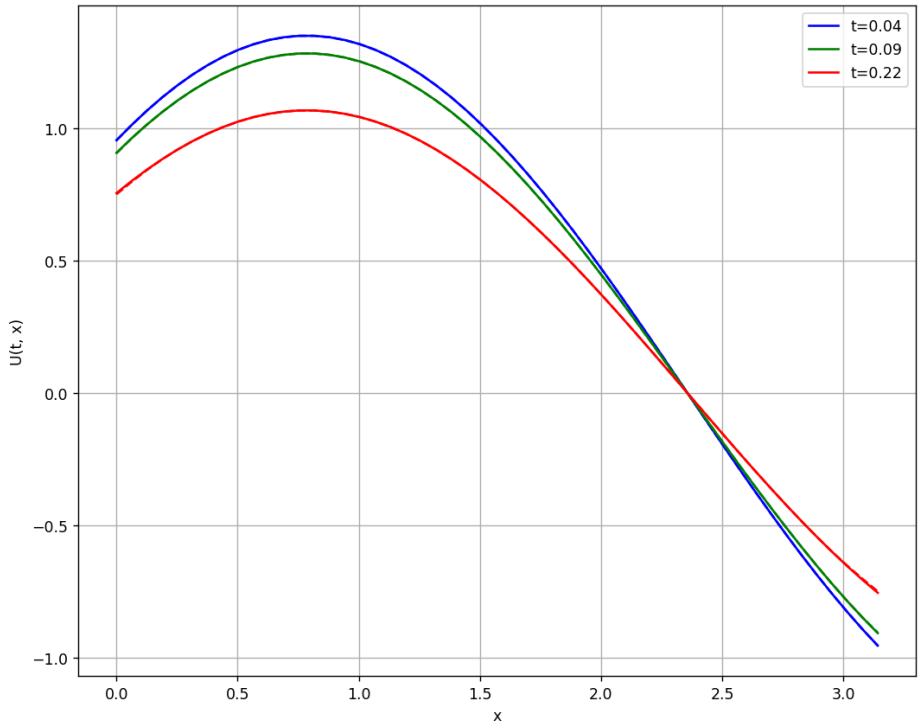
$$\begin{cases} \frac{u_1^{k+1} - u_0^{k+1}}{h} - u_0^{k+1} = 0 \\ \frac{u_N^{k+1} - u_{N-1}^{k+1}}{h} - u_N^{k+1} = 0 \end{cases} => \begin{cases} u_0^{k+1} = \frac{u_1^{k+1}}{h+1} \\ u_N^{k+1} = \frac{u_{N-1}^{k+1}}{1-h} \end{cases}$$

для $j = 1$ и $j = N - 1$ имеем $b_1 = -(2\sigma + 1) + \frac{\sigma}{h+1}$ и $b_{N-1} = -(2\sigma + 1) + \frac{\sigma}{1-h}$.

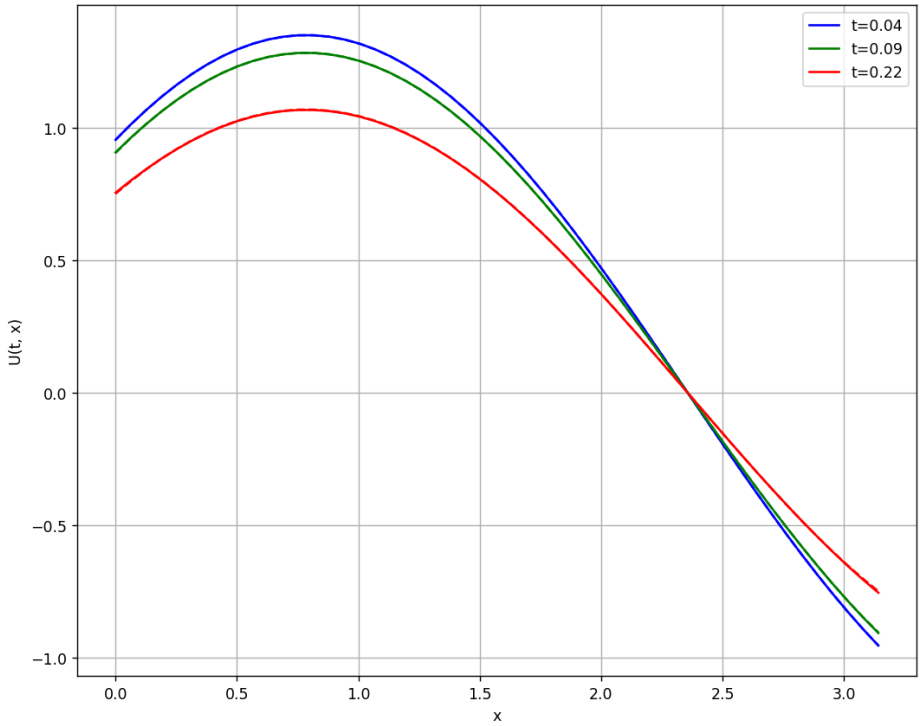
Результат работы программы

Программа считает задачу с одним заранее заданным $\sigma = \frac{1}{2}$. На основании этого значения она рассчитывае шаги по времени и координате таким образом, чтобы выполнялось условие устойчивости для явного метода. На графики она выводит значение функции при трёх различных фиксированных t.

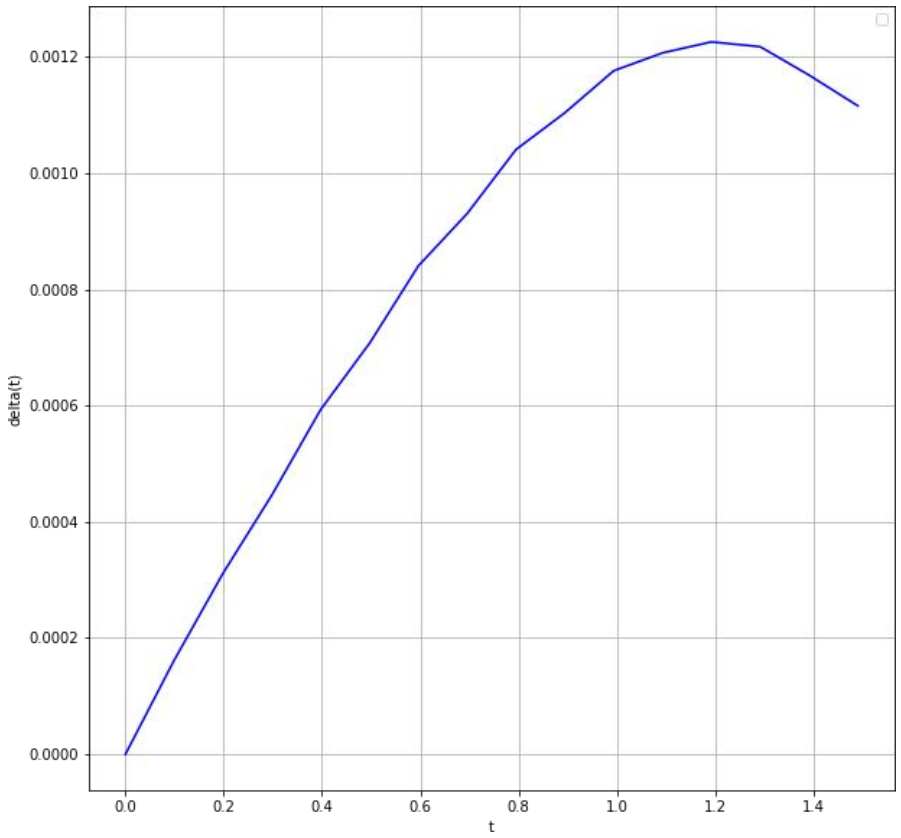
Сплошной линией на графике изображено точное решение, пунктирной – рассчитанное численно. Для построения графика погрешностей методов перебирается 3 различных разбиений сетки по координате x.



Явная схема



Неявная схема



Погрешности

Вывод

В результате выполнения данной лабораторной работы, было выяснено, что явная, как и неявная схемы практически в равной степени дают достаточно точный результат. При увеличении шага по кооординате x закономерно росла и погрешность. Однако стоит отметить, что неявная схема сильнее реагирует на изменение шага – если он увеличивается, погрешность этой схемы увеличится заметно больше, чем погрешность явной схемы.

Код программы

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

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

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

a = 1
T = 1
N = 50
sigma = 0.5
l = np.pi
h = l / N
tau = np.sqrt(sigma * h ** 2 / a)
K = int(round(T / tau))
u0 = np.zeros((K + 1, N + 1))


def U(x, t, a):
    return np.sin(x - a * t) + np.cos(x + a * t)

def psi(x):
    return np.sin(x) + np.cos(x)

def dPsi(x, a, tau):
    return (1 - a * tau - a ** 2 * tau ** 2 / 2) * (np.sin(x) + np.cos(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 solve(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 = [10, 20, 40]
    size = np.size(NArray)
    hArray = np.zeros(size)
    tauArray = np.zeros(size)
    KArray = np.zeros(size)
    errors1 = np.zeros(size)

    errors2 = 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, tauArray[i],
a, hArray[i])
        u2 = implicitMethod(NArray[i], int(KArray[i]), sigma, tauArray[i],
a, hArray[i])
        t = tauArray[i] * KArray[i] / 2
        if (np.size(xArray) != NArray[i] + 1):
            xArray = xArray[:NArray[i] + 1]
        uCorrect = U(xArray, t, a)
        u1Calc = u1[int(KArray[i] / 2)]
        u2Calc = u2[int(KArray[i] / 2)]
        errors1[i] = np.amax(np.abs(uCorrect - u1Calc))
        errors2[i] = np.amax(np.abs(uCorrect - u2Calc))
    return NArray, errors1, errors2


def showSolution(h, tau, K, l, u, U, a):
    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, a)
        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 = error(sigma, l, a, T, U)
    colors = ['blue', 'green']
    deltaX = np.zeros(np.size(NArray))
    for i in range(np.size(NArray)):
        deltaX[i] = l / NArray[i]
    fig, ax = plt.subplots()
    plt.plot(deltaX, errors1, color=colors[0], label='Явный метод')
    plt.plot(deltaX, errors2, color=colors[1], label='Неявный метод')
    ax.set_xlabel('delta X')
    ax.set_ylabel('Epsilon')
    plt.grid()
    ax.legend()
    plt.show()
```

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


def implicitMethod(N, K, sigma, tau, a, h):
    aj = sigma
    bj = -(1 + 2 * sigma)
    cj = sigma
    u = np.zeros((K + 1, N + 1))
    for i in range(0, N + 1):
        u[0][i] = psi(i * h)
        u[1][i] = dPsi(i * h, a, tau)
    for k in range(2, K + 1):
        matrix = np.zeros((N - 1, N - 1))
        d = np.zeros(N - 1)
        matrix[0][0] = bj + sigma / (h + 1)
        matrix[0][1] = cj
        d[0] = -2 * u[k - 1][1] + u[k - 2][1]
        for j in range(1, N - 2):
            matrix[j][j - 1] = aj
            matrix[j][j] = bj
            matrix[j][j + 1] = cj
            d[j] = -2 * u[k - 1][j + 1] + u[k - 2][j + 1]
        matrix[N - 2][N - 3] = aj
        matrix[N - 2][N - 2] = bj + sigma / (1 - h)
        d[N - 2] = -2 * u[k - 1][N - 1] + u[k - 2][N - 1]
        # Решем СЛАУ методом прогонки
        ans = solve(matrix, d)
        u[k][1:N] = ans
        u[k][0] = u[k][1] / (h + 1)
        u[k][N] = u[k][N - 1] / (1 - h)
    return u


def main():
    for k in range(0, K + 1):
        for j in range(0, N + 1):
            u0[k][j] = U(j * h, k * tau, a)
    u1 = explicitMethod(N, K, sigma, tau, a, h)
    showSolution(h, tau, K, l, u1, U, a)
    u2 = implicitMethod(N, K, sigma, tau, a, h)
    showSolution(h, tau, K, l, u2, U, a)
    showErrors(sigma, l, a, T, U)


main()
```