Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа 3

"Численное решение смешанной задачи для уравнения теплопроводности"

Подготовил:
студент 3 курса 3 группы
Тев Никита Михайлович

Преподаватель:
Горбачёва Юлия
Николаевна

Минск, 2019 г.

1. <u>Постановка задачи</u>

   На сетке узлов $\overline{\omega}_{h\tau}$ найти численное решение смешанной задачи для одномерного уравнения теплопроводности с использованием:

   - явной разностной схемы с $h = \tau = 0.1$ и $h = 0.1, \tau = \frac{h^2}{2} = 0.005$;
   - чисто неявной разностной схемы с $h = \tau = 0.1$;
   - разностной схемы Кранка-Николсона с $h = \tau = 0.1$.

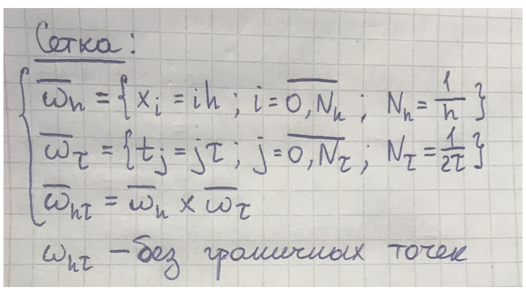
   Выписать соответствующие разностные схемы, указать их порядок аппроксимации, указать, являются ли схемы абсолютно устойчивыми по начальным данным. Вычислить погрешность численного решения. Построить графики, демонстрирующие устойчивое и неустойчивое поведение явной разностной схемы.

2. <u>Условие задачи:</u>

$$\begin{cases} \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + cosx(cost + sint),\ 0 < x < 1,\ 0 < t \leq 0.5 \\ u(x,0) = u_0(x) = 0,\ 0 \leq x \leq 1 \\ u(0,t) = u_1(t) = sint,\ 0 \leq t \leq 0.5 \\ u(1,t) = u_2(t) = sin(t)cos(1),\ 0 \leq t \leq 0.5 \end{cases}$$

   Точное решение: $u(x,t) = sin(t)cos(x)$

3. <u>Сетка</u>

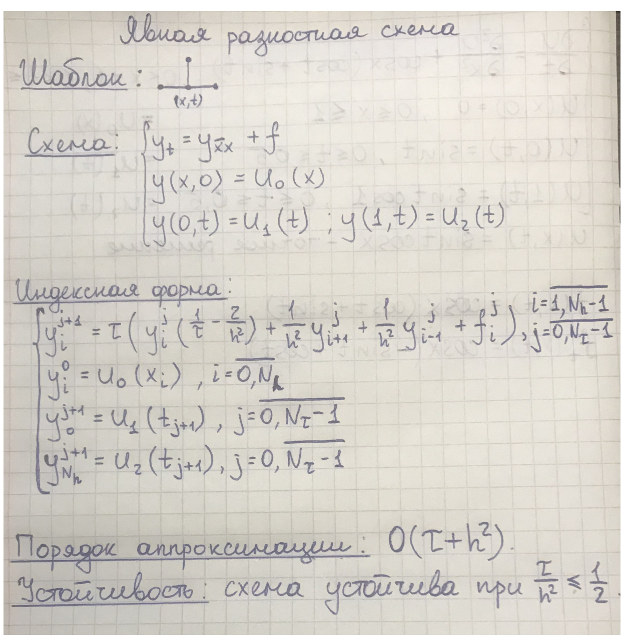


Сетка:

$$\begin{cases} \overline{\omega}_h = \{x_i = ih;\ i = \overline{0,N_h};\ N_h = \frac{1}{h}\} \\ \overline{\omega}_\tau = \{t_j = j\tau;\ j = \overline{0,N_\tau};\ N_\tau = \frac{1}{2\tau}\} \\ \overline{\omega}_{h\tau} = \overline{\omega}_h \times \overline{\omega}_\tau \end{cases}$$

$\omega_{h\tau}$ — без граничных точек

4. <u>Разностные схемы</u>



Явная разностная схема

Шаблон: (x,t)

Схема:
$$\begin{cases} y_t = y_{\bar{x}x} + f \\ y(x,0) = u_0(x) \\ y(0,t) = u_1(t);\ y(1,t) = u_2(t) \end{cases}$$

Индексная форма:
$$\begin{cases} y_i^{j+1} = \tau\left(y_i^j\left(\frac{1}{\tau} - \frac{2}{h^2}\right) + \frac{1}{h^2} y_{i+1}^j + \frac{1}{h^2} y_{i-1}^j + f_i^j\right),\ i = \overline{1,N_h-1},\ j = \overline{0,N_\tau-1} \\ y_i^0 = u_0(x_i),\ i = \overline{0,N_h} \\ y_0^{j+1} = u_1(t_{j+1}),\ j = \overline{0,N_\tau-1} \\ y_{N_h}^{j+1} = u_2(t_{j+1}),\ j = \overline{0,N_\tau-1} \end{cases}$$

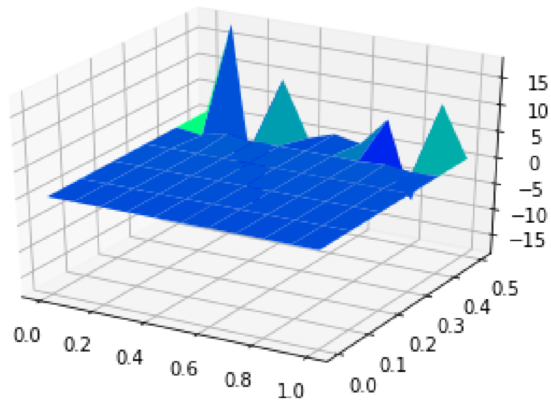
Порядок аппроксимации: $O(\tau + h^2)$.

Устойчивость: схема устойчива при $\frac{\tau}{h^2} \leq \frac{1}{2}$.

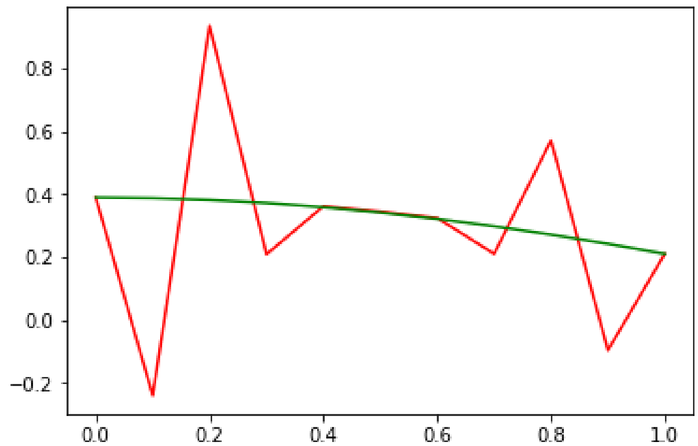
   В случае $h = \tau = 0.1$ явная разностная схема является неустойчивой.

   Полученная погрешность: $\Delta = 0.6273$

   3D-график численного решения:



Рассмотрим также сечение в момент времени $t = 0.4$ (зеленым - точное решение, красным - численное):

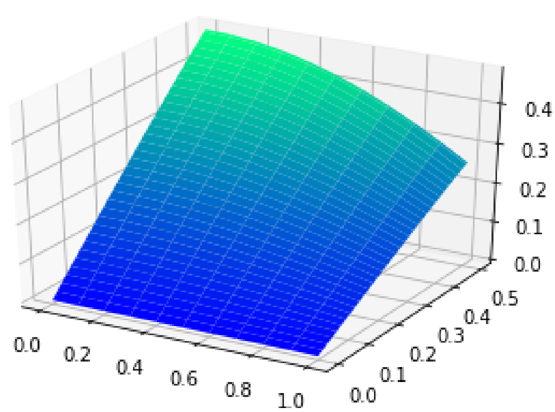


По графикам хорошо видно неустойчивое поведение схемы при заданных параметрах.

Теперь рассмотрим ту же схему с $h = 0.1, \tau = \frac{h^2}{2} = 0.005$.

Полученная погрешность: $\Delta = 0.00014$

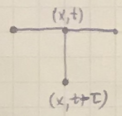
3D-график численного решения:



При таких параметрах схема устойчива и решение представляет собой гладкую поверхность.

Чисто неявная схема

Шаблон:



$\hat{u} = u(x,t+\tau)$

Схема:

$$\begin{cases} y_t = \hat{y}_{\bar{x}x} + f \\ y(x,0) = u_0(x) \\ y(0,t) = u_1(t), \ y(1,t) = u_2(t) \end{cases}$$

Индексная форма:

$$\begin{cases} \left(\frac{1}{\tau}+\frac{2}{h^2}\right)y_i^{j+1} - \frac{1}{h^2}y_{i+1}^{j+1} - \frac{1}{h^2}y_{i-1}^{j+1} = f_i^j + \frac{1}{\tau}y_i^j \\ y_i^0 = u_0(x_i), \ i=\overline{0,N_h} \\ y_0^{j+1} = u_1(t_{j+1}), \ j=\overline{0,N_\tau-1} \\ y_{N_h}^{j+1} = u_2(t_{j+1}), \ j=\overline{0,N_\tau-1} \end{cases}$$

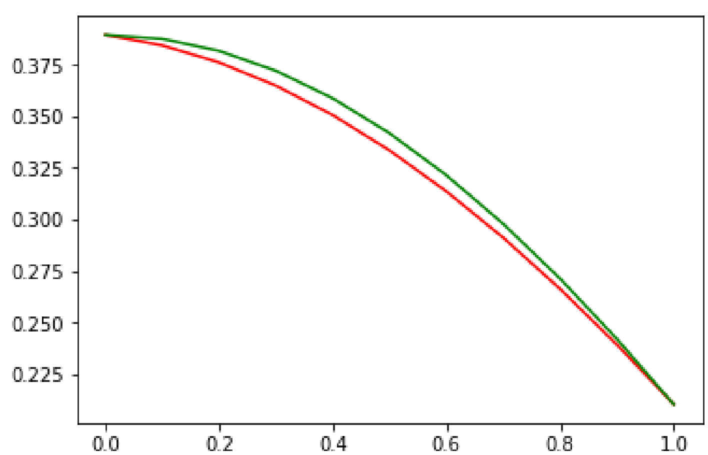
/ $i=\overline{1,N_h-1}$, $j=\overline{0,N_\tau-1}$

Порядок аппроксимации: $O(\tau+h^2)$

Устойчивость: Абсолютно устойчива.

Полученная погрешность: $\Delta = 0.00831$

Сечение в момент времени $t = 0.4$ (зеленым - точное решение, красным - численное) демонстрирует устойчивое поведение схемы:



3D-график численного решения:

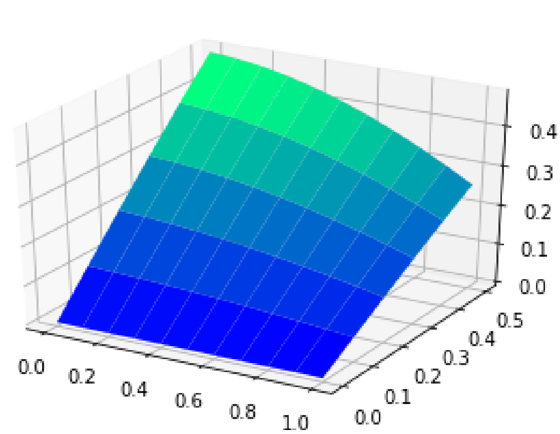


Схема Кранка-Никлсона:

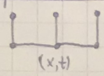
Шаблон: 

Схема: $\begin{cases} y_t = \frac{1}{2}(y_{\bar{x}x} + \hat{y}_{\bar{x}x}) + \varphi \ ; \ \varphi = f + \frac{\tau}{2}\frac{\partial f}{\partial t} \\ y(x,0) = u_0(x) \\ y(0,t) = u_1(t), \ y(1,t) = u_2(t) \end{cases}$

Индексная форма:

$$\begin{cases} \left(\frac{1}{\tau} + \frac{1}{h^2}\right) y_i^{j+1} - \frac{1}{2h^2} y_{i+1}^{j+1} - \frac{1}{2h^2} y_{i-1}^{j+1} = \varphi_i^j + \frac{1}{\tau} y_i^j + \frac{1}{2h^2}\left(y_{i+1}^j - 2y_i^j + y_{i-1}^j\right), \ i = \overline{1, N_h - 1}, \ j = \overline{0, N_\tau - 1} \\ y_i^0 = u_0(x_i), \ i = \overline{0, N_h} \\ y_0^{j+1} = u_1(t_{j+1}), \ j = \overline{0, N_\tau - 1} \\ y_{N_h}^{j+1} = u_2(t_{j+1}), \ j = \overline{0, N_\tau - 1} \end{cases}$$

Порядок аппроксимации: $O(\tau^2 + h^2)$

Устойчивость: Абсолютно устойчива.

Полученная погрешность: $\Delta = 0.00027$

5. Листинг программы

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

def f(x, t):
    return np.cos(x) * (np.cos(t) + np.sin(t))

def u0(x):
    return 0

def u1(t):
    return np.sin(t)

def u2(t):
    return np.sin(t) * np.cos(1)

def u(x, t):
    return np.sin(t) * np.cos(x)

def phi(x, t, tau):
    return f(x, t) + (tau/2)*(np.cos(x) * (np.cos(t) - np.sin(t)))

def solve_tridiagonal_1(x, ypast, h, tau, t):
    N = x.shape[0] - 1
    a = np.zeros(N+1)
    c = np.zeros(N+1)
    b = np.zeros(N+1)
    F = np.zeros(N+1)

    b[0] = 0
    c[0] = 1
    F[0] = u1(t+tau)

    c[N] = 1
    a[N] = 0
    F[N] = u2(t+tau)

    for j in range(1,N):
        a[j] = 1/(h*h)
        c[j] = 2/(h*h) + 1/tau
        b[j] = 1/(h*h)
        F[j] = f(x[j], t) + (1/tau)*ypast[j]

    alpha = np.zeros(N+1)
    beta = np.zeros(N+2)
```

```
    alpha[1] = b[0]/c[0]
    beta[1] = F[0]/c[0]

    for i in range(1, N):
        alpha[i+1] = b[i]/(c[i] - a[i]*alpha[i])
        beta[i+1] = (F[i] + a[i]*beta[i])/(c[i] - a[i]*alpha[i])

    beta[N+1] = (F[N] + a[N]*beta[N])/(c[N] - a[N]*alpha[N])

    y = np.zeros(N+1)

    y[N] = beta[N+1]
    for i in reversed(range(0, N)):
        y[i] = alpha[i+1]*y[i+1] + beta[i+1]

    return y

def solve_tridiagonal_2(x, ypast, h, tau, t):
    N = x.shape[0] - 1
    a = np.zeros(N+1)
    c = np.zeros(N+1)
    b = np.zeros(N+1)
    F = np.zeros(N+1)

    b[0] = 0
    c[0] = 1
    F[0] = u1(t+tau)

    c[N] = 1
    a[N] = 0
    F[N] = u2(t+tau)

    for j in range(1,N):
        a[j] = 1/(2*h*h)
        c[j] = 1/(h*h) + 1/tau
        b[j] = 1/(2*h*h)
        F[j] = phi(x[j], t, tau) + (1/tau)*ypast[j] + 1/(2*h*h)*(ypast[j+1] - 2*ypast[j] + ypast[j-1])

    alpha = np.zeros(N+1)
    beta = np.zeros(N+2)

    alpha[1] = b[0]/c[0]
    beta[1] = F[0]/c[0]

    for i in range(1, N):
        alpha[i+1] = b[i]/(c[i] - a[i]*alpha[i])
        beta[i+1] = (F[i] + a[i]*beta[i])/(c[i] - a[i]*alpha[i])

    beta[N+1] = (F[N] + a[N]*beta[N])/(c[N] - a[N]*alpha[N])

    y = np.zeros(N+1)

    y[N] = beta[N+1]
    for i in reversed(range(0, N)):
        y[i] = alpha[i+1]*y[i+1] + beta[i+1]

    return y

from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

def plot_answer(X, Y, Z):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    X, Y = np.meshgrid(X, Y)

    surf = ax.plot_surface(X, Y, Z, cmap=cm.winter,
                    linewidth=3, antialiased=True)

    plt.show()

def make_table(h, tau):
    tn, hn = int(0.5/tau) + 1, int(1/h) + 1
    return np.zeros((tn, hn))

def make_grid(h, tau):
    xgrid = np.linspace(0, 1, int(1/h)+1, True)
    tgrid = np.linspace(0, 0.5, int(0.5/tau)+1, True)
    return xgrid, tgrid

xgrid, tgrid = make_grid(0.1, 0.1)
xgrid2, tgrid2 = make_grid(0.1, 0.005)
```

```
y1 = make_table(0.1, 0.1)
y2 = make_table(0.1, 0.005)
y3 = make_table(0.1, 0.1)
y4 = make_table(0.1, 0.1)

def get_error(y, xgrid, tgrid):
    error = 0.0
    for i in range(0, len(tgrid) - 1):
        for j in range(0, len(xgrid) - 1):
            if (error < np.abs(y[i][j] - u(xgrid[j], tgrid[i]))):
                error = np.abs(y[i][j] - u(xgrid[j], tgrid[i]))
    return error

tau = 0.1
h = 0.1
for j in range(0, len(xgrid)):
    y1[0][j] = u0(xgrid[j])
for i in range(0, len(tgrid)):
    y1[i][0] = u1(tgrid[i])
    y1[i][len(xgrid) - 1] = u2(tgrid[i])
for i in range(0, len(tgrid) - 1):
    for j in range(1, len(xgrid) - 1):
        y1[i+1][j] = tau * (y1[i][j]*(1/tau - 2/(h*h)) + y1[i][j+1]/(h*h) + y1[i][j-1]/(h*h) + f(xgrid[j], tgrid[i]))
print(get_error(y1, xgrid, tgrid))

plot_answer(xgrid, tgrid, y1[:])
plt.plot(xgrid, y1[4], 'r', xgrid, u(xgrid, tgrid[4]), 'g')

tau = 0.005
h = 0.1
for j in range(0, len(xgrid2)):
    y2[0][j] = u0(xgrid2[j])
for i in range(0, len(tgrid2)):
    y2[i][0] = u1(tgrid2[i])
    y2[i][len(xgrid2) - 1] = u2(tgrid2[i])
for i in range(0, len(tgrid2) - 1):
    for j in range(1, len(xgrid2) - 1):
        y2[i+1][j] = tau * (y2[i][j]*(1/tau - 2/(h*h)) + y2[i][j+1]/(h*h) + y2[i][j-1]/(h*h) + f(xgrid2[j], tgrid2[i]))
print(get_error(y2, xgrid2, tgrid2))

plot_answer(xgrid2, tgrid2, y2[:])

tau = 0.1
h = 0.1
for j in range(0, len(xgrid)):
    y3[0][j] = u0(xgrid[j])
for i in range(0, len(tgrid) - 1):
    y3[i+1] = solve_tridiagonal_1(xgrid, y3[i], h, tau, tgrid[i])
print(get_error(y3, xgrid, tgrid))

plot_answer(xgrid, tgrid, y3[:])
plt.plot(xgrid, y3[4], 'r', xgrid, u(xgrid, tgrid[4]), 'g')

tau = 0.1
h = 0.1
for j in range(0, len(xgrid)):
    y4[0][j] = u0(xgrid[j])
for i in range(0, len(tgrid) - 1):
    y4[i+1] = solve_tridiagonal_2(xgrid, y4[i], h, tau, tgrid[i])
print(get_error(y4, xgrid, tgrid))
```