Лабораторная работа №4

«Численное решение смешанной задачи для уравнения теплопроводности»

Вариант 10

Выполнил студент 3 курса 2 группы ФПМИ

Сараев Владислав Максимович

Минск, 2020

Постановка задачи

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

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

Выписать соответствующие разностные схемы, указать их порядок аппроксимации, указать являются ли схемы абсолютно устойчивыми по начальным данным. Вычислить погрешность численного решения (т.е. найти $\max\limits_{i,j} | y_i^j - u(x_i,\ t_j)|$). Построить графики, демонстрирующие устойчивое и неустойчивое поведение явной разностной схемы.

Задача:

$$\begin{cases} \dfrac{\partial u}{\partial u} = \dfrac{\partial^2 u}{\partial x^2} - e^{-t}\sin(x+t)\,, \ \ 0 < x < 1, \ \ 0 < t \leq 0.5 \\ u(x,0) = cosx, \ \ 0 \leq x \leq 1 \\ u(0,t) = e^{-t}cost, \ \ 0 \leq t \leq 0.5 \\ u(1,t) = e^{-t}\cos(1+t)\,, \ \ 0 \leq t \leq 0.5 \end{cases}$$

$$u(x,t) = e^{-t}\cos(x+t)$$

Краткие теоретические сведения

Имеем одномерное уравнение теплопроводности с постоянными коэффициентами в прямоугольнике $\overline{D} = \{0 \le x \le 1; 0 \le t \le T\}$:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \dfrac{\partial^2 u}{\partial x^2} + f(x,t), \;\; 0 < x < 1, \;\; 0 < t \le T \\ u(x,0) = u_0(x), \;\; 0 \le x \le 1 \\ u(0,t) = \mu_0(t), \;\; 0 \le t \le T \\ u(1,t) = \mu_1(t), \;\; 0 \le t \le T \end{cases}$$

при этом функции, задающие дополнительные условия, должны быть согласованы, т.е. $u_0(0) = \mu_0(0)$ и $u_0(1) = \mu_1(0)$.

Введем равномерную пространственно-временную сетку

$\overline{\omega}_{h\tau} = \{ (x_i, t_j): x_i = ih, \; t_j = j\tau; h = \frac{1}{N_1}, \tau = \frac{T}{N_2}; i = \overline{0, N_1}, j = \overline{0, N_2}\}$.

Для аппроксимации уравнения будем использовать следующую разностную схему:

$$\begin{cases} y_t = \sigma \hat{y}_{\bar{x}x} + (1-\sigma) y_{\bar{x}x} + \varphi, (x,t) \in \overline{\omega}_{h\tau} \\ y(x,0) = u_0(x), \;\; x \in \overline{\omega}_h \\ y(0,t) = \mu_0(t), \;\; t \in \overline{\omega}_\tau \\ y(1,t) = \mu_1(t), \;\; t \in \overline{\omega}_\tau \end{cases}$$

где φ – сеточная функция, аппроксимация $f(x,t)$.

Данная схема в индексной форме имеет вид:

$$\begin{cases} \dfrac{y_i^{j+1} - y_i^j}{\tau} = \sigma \dfrac{y_{i+1}^{j+1} - 2y_i^{j+1} + y_{i-1}^{j+1}}{h^2} + (1-\sigma) \dfrac{y_{i+1}^j - 2y_i^j + y_{i-1}^j}{h^2} + \varphi_i^j, \\ i = \overline{1, N_1 - 1}, j = \overline{0, N_2 - 1} \\ y_i^0 = u_0(x_i), \;\; i = \overline{0, N_1} \\ y_0^{j+1} = \mu_0(t_{j+1}), \;\; j = \overline{0, N_2 - 1} \\ y_{N_1}^{j+1} = \mu_1(t_{j+1}), \;\; j = \overline{0, N_2 - 1} \end{cases}$$

Дальнейшее решение зависит от параметра σ:

1. $\sigma = 0$. Тогда получаем разностная схема становится **явной**:

$$y_t = y_{\bar{x}x} + \varphi$$

   или в индексной форме:

$$\frac{y_i^{j+1} - y_i^j}{\tau} = \frac{y_{i+1}^j - 2y_i^j + y_{i-1}^j}{h^2} + \varphi_i^j$$

   Тогда порядок расчетов будет следующим:

   1) заполняем нулевой слой по формулам:

$$y_i^0 = u_0(x_i), i = \overline{0, N_1}$$

   2) для всех $j = \overline{0, N_2 - 1}$ заполняем очередной $(j+1)$-ый слой по формулам:
      $y_0^{j+1} = \mu_0(t_j + 1)$, $y_i^{j+1} = (1 - 2\gamma) y_i^j + \gamma(y_{i-1}^j + y_{i+1}^j) + \tau \varphi_i^j, i = \overline{1, N_1 - 1}$
      $y_{N_1}^{j+1} = \mu_1(t_{j+1})$, где $\gamma = \frac{\tau}{h^2}$

2. $\sigma \neq 0$. Тогда получаем **неявную** разностную схему:

$$\begin{cases} \sigma \dfrac{y_{i+1}^{j+1} - 2y_i^{j+1} + y_{i-1}^{j+1}}{h^2} - \dfrac{1}{\tau} y_i^{j+1} = -\dfrac{1}{\tau} y_i^j - (1-\sigma)\dfrac{y_{i+1}^j - 2y_i^j + y_{i-1}^j}{h^2} - \varphi_i^j, \\ i = \overline{1, N_1 - 1} \\ y_0^{j+1} = \mu_0(t_{j+1}) \\ y_{N1}^{j+1} = \mu_1(t_{j+1}) \end{cases}$$

или

$$\begin{cases} \dfrac{\sigma}{h^2} y_{i-1}^{j+1} - \left(\dfrac{1}{\tau} + \dfrac{2\sigma}{h^2}\right) y_i^{j+1} + \dfrac{\sigma}{h^2} y_{i+1}^{j+1} = -F_i^j, \;\; i = \overline{1, N_1 - 1} \\ y_0^{j+1} = \mu_0(t_{j+1}) \\ y_{N1}^{j+1} = \mu_1(t_{j+1}) \end{cases}$$

где $F_i^j = \frac{1}{\tau} y_i^j + (1-\sigma)\frac{y_{i-1}^j - 2y_i^j + y_{i+1}^j}{h^2} + \varphi_i^j$. Решение данной системы можно найти с помощью метода прогонки. Данная схема при $\sigma = 1$ называется **чисто неявной схемой** или **схемой с опережением**, а при $\sigma = \frac{1}{2}$ **симметричной схемой** или **схемой Кранка-Николсон**.

Порядок аппроксимации и устойчивость:

1. явная разностная схема:
   - $\psi = O(\tau + h^2)$, при $\varphi = f(x,\; t+\tau)$
   - не является абсолютно устойчивой (устойчива при $\tau \leq \frac{h^2}{2}$)
2. чисто неявная разностная схема:
   - $\psi = O(\tau + h^2)$, при $\varphi = f(x,\; t+\tau)$
   - является абсолютно устойчивой
3. разностная схема Кранка-Николсон
   - $\psi = O(\tau^2 + h^2)$, при $\varphi = f(x, t + \frac{\tau}{2})$
   - является абсолютно устойчивой

Листинг

```
import numpy as np
from math import e, cos, sin, fabs
import matplotlib.pyplot as plt

L1 = 0
R1 = 1
L2 = 0
R2 = 0.5


def u(x, t):
    return e**(-t) * cos(x + t)


def u0(x):
    return cos(x)


def mu0(t):
    return e**(-t) * cos(t)


def mu1(t):
    return e**(-t) * cos(1 + t)


def f(x, t):
    return -e**(-t) * sin(x + t)


# значения функций в узлах
def init_values(h, tau):
    x_values = np.linspace(L1, R1, int((R1 - L1) / h) + 1)
    t_values = np.linspace(L2, R2, int((R2 - L2) / tau) + 1)
    u0_values = np.array([u0(x) for x in x_values])
    mu0_values = np.array([mu0(t) for t in t_values])
    mu1_values = np.array([mu1(t) for t in t_values])
    y_values = np.zeros((len(t_values), len(x_values)))
    u_values = np.array([np.array([u(x, t) for x in x_values]) for t in t_values])
    return x_values, t_values, u0_values, mu0_values, mu1_values, y_values, u_values


# расчет погрешности
def fault(y_values, u_values, N1, N2):
    max_delta = 0
    for i in range(N2):
        for j in range(N1):
            delta = fabs(y_values[i][j] - u_values[i][j])
            if delta > max_delta:
                max_delta = delta
    return max_delta


# вывод графика
def show_graphic(x_values, t_values, y_values, u_values):
    ax = plt.axes(projection="3d")
    xgrid, tgrid = np.meshgrid(x_values, t_values)
    ax.plot_wireframe(xgrid, tgrid, u_values, color='green', label="Function")
    ax.plot_wireframe(xgrid, tgrid, y_values, color='red', label="Approximation")
    ax.set_xlabel('x')
    ax.set_ylabel('t')
    ax.set_zlabel('y')
    plt.legend()
    plt.show()


# прогонка
def tridiagonal_matrix_algorithm(matrix, f):
```

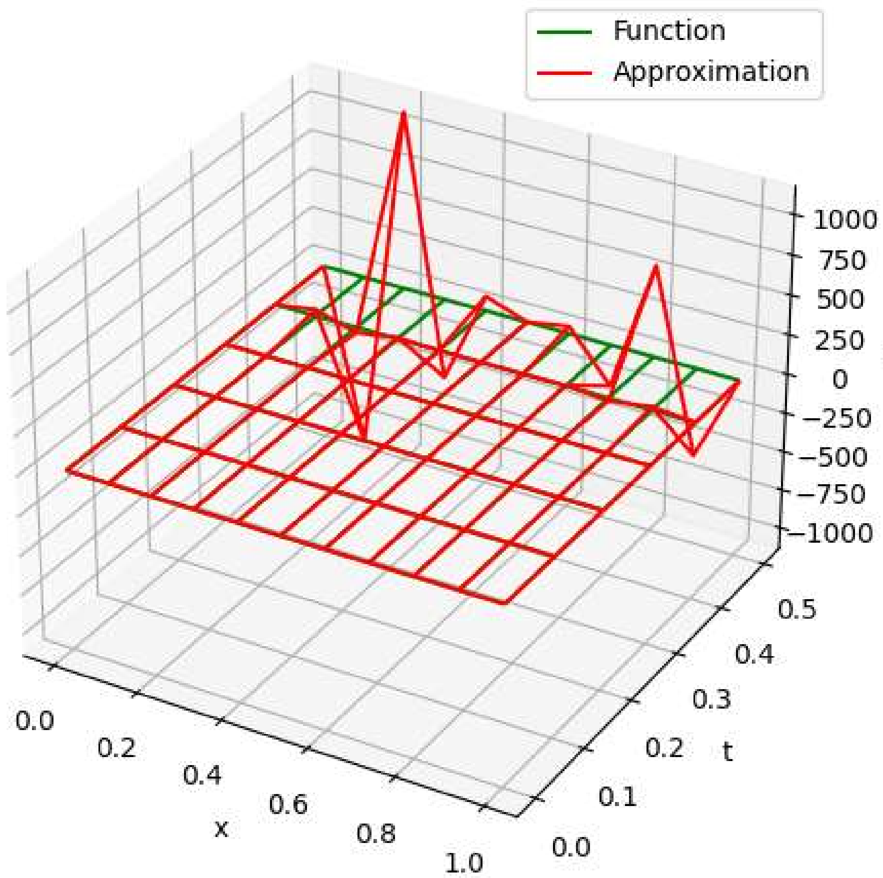
```
    n = len(f)
    matrix[2][0] /= matrix[1][0]
    f[0] /= matrix[1][0]
    matrix[1][0] = 1
    for i in range(1, n - 1):
        coeff = matrix[1][i] - matrix[2][i - 1] * matrix[0][i - 1]
        matrix[2][i] /= coeff
        f[i] = (f[i] - f[i - 1] * matrix[0][i - 1]) / coeff
        matrix[1][i] = 1
        matrix[0][i - 1] = 0
    f[n - 1] = (f[n - 1] - f[n - 2] * matrix[0][n - 2]) / (matrix[1][n - 1] -
matrix[2][n - 2] * matrix[0][n - 2])
    matrix[1][n - 1] = 1

    for i in range(n - 2, -1, -1):
        f[i] -= f[i + 1] * matrix[2][i]
        matrix[2][i] = 0


# явный метод
def explicit_method(h, tau):
    gamma = tau / (h ** 2)
    x_values, t_values, u0_values, mu0_values, mu1_values, y_values, u_values =
init_values(h, tau)
    N1 = len(x_values)
    N2 = len(t_values)

    def fi(x, t):
        return f(x, t + tau)

    fi_values = np.array([np.array([fi(x, t) for x in x_values]) for t in t_values])

    # заполняю нулевой слой
    for i in range(N1):
        y_values[0][i] = u0_values[i]

    # для всех слоев j от 1 до N1-1
    for j in range(1, N2):
        # берем y_0 из левого граничного условия
        y_values[j][0] = mu0_values[j]
        # рекуррентно вычисляем y_1 - y_(N1-2) через предыдущий слой
        for i in range(1, N1 - 1):
            y_values[j][i] = (1 - 2 * gamma) * y_values[j - 1][i] + gamma * (
                    y_values[j - 1][i - 1] + y_values[j - 1][i + 1]) + tau *
fi_values[j - 1][i]
        # берем y_(N1-1) из правого граничного условия
        y_values[j][N1 - 1] = mu1_values[j]

    max_delta = fault(y_values, u_values, N1, N2)
    show_graphic(x_values, t_values, y_values, u_values)
    return max_delta


# неявный метод
def implicit_method(h, tau, sigma):
    x_values, t_values, u0_values, mu0_values, mu1_values, y_values, u_values =
init_values(h, tau)
    N1 = len(x_values)
    N2 = len(t_values)

    def fi(x, t):
        if sigma == 1 / 2:
            return f(x, t + tau / 2)
        return f(x, t + tau)

    fi_values = np.array([np.array([fi(x, t) for x in x_values]) for t in t_values])

    # заполняю нулевой слой
    for i in range(N1):
        y_values[0][i] = u0_values[i]
```

```python
    # коэффициент на главной диагонали трехдиагональной матрицы
    coef1 = sigma / (h ** 2)
    # коэффициенты на поддиагонали и наддиагонали
    coef2 = -(1 / tau + 2 * sigma / (h ** 2))

    # для всех слоев j от 1 до N1-1
    for j in range(1, N2):
        # создаю трехдиагональную матрицу, где на главной диагонали стоит coef1,
        # а на поддиагонали и наддиагонали стоят coef2
        # матрица хранится ввиде трех векторов
        #    |coef1 coef2 0    ...   0    0| - строка для вычисления y_1       = b_1
        #    |coef2 coef1 coef2 0 ...     0| - строка для вычисления y_2       = b_2
        #    |0 coef2 coef1 coef2 0 ... 0| - строка для вычисления y_3       = b_3
        #    |            .....           |
        #    |0 ... 0    coef2 coef1 coef2| - строка для вычисления y_(N1-3)  = b_(N1-3)
        #    |0 ... 0     0   coef2 coef1 | - строка для вычисления y_(N1-2)  = b_(N1-2)
        matrix = np.array(
            [np.full(N1 - 3, coef1), np.full(N1 - 2, coef2), np.full(N1 - 3, coef1)]
        )
        # правая часть уравнения
        b = np.zeros(N1 - 2)

        # вычисление y_0 и y_(N1-1)
        y_values[j][0] = mu0_values[j]  # (1)
        y_values[j][N1 - 1] = mu1_values[j]  # (2)

        # вычисляю b_1 как -F_ji - coef1 * y_0
        b[0] = -coef1 * y_values[j][0] - (y_values[j - 1][1] / tau + (1 - sigma) *
                                           (y_values[j - 1][0] - 2 * y_values[j - 1][1]
+ y_values[j - 1][2]) / (
                                                       h ** 2) +
                                           fi_values[j - 1][1])

        # вычисляю b_2 - b_(N1-3) включительно как -F_ji
        for i in range(1, N1 - 3):
            b[i] = - (y_values[j - 1][i + 1] / tau + (1 - sigma) *
                      (y_values[j - 1][i] - 2 * y_values[j - 1][i + 1] + y_values[j -
1][i + 2]) / (h ** 2) +
                      fi_values[j - 1][i + 1])

        # вычисляю b_(N-2) как -F_ji - coef1 * y_(N1-1)
        b[N1 - 3] = -coef1 * y_values[j][N1 - 1] - \
                    (y_values[j - 1][N1 - 2] / tau + (1 - sigma) *
                     (y_values[j - 1][N1 - 3] - 2 * y_values[j - 1][N1 - 2]
                      + y_values[j - 1][N1 - 1]) / (h ** 2) + fi_values[j - 1][N1 -
2])

        # метод прогонки
        tridiagonal_matrix_algorithm(matrix, b)
        # заполняю y с 1 по (N1-2) индексы включительно
        #   y_0 и y_(N1-2) заполнены в (1) и (2)
        for i in range(1, N1 - 1):
            y_values[j][i] = b[i - 1]

    max_delta = fault(y_values, u_values, N1, N2)
    show_graphic(x_values, t_values, y_values, u_values)
    return max_delta


if __name__ == "__main__":
    # явная разностная схема с tau = h = 0.1
    print(explicit_method(0.1, 0.1))
    # явная разностная схема с h = 0.1 и tau = h^2/2
    print(explicit_method(0.1, 0.005))
    # чисто неявная разностная схема с tau = h = 0.1
    print(implicit_method(0.1, 0.1, 1))
    # разностная схема Кранка-Николсон с tau = h = 0.1
    print(implicit_method(0.1, 0.1, 1 / 2))
```

Результаты

1. Явная разностная схема с $\tau = h = 0.1$

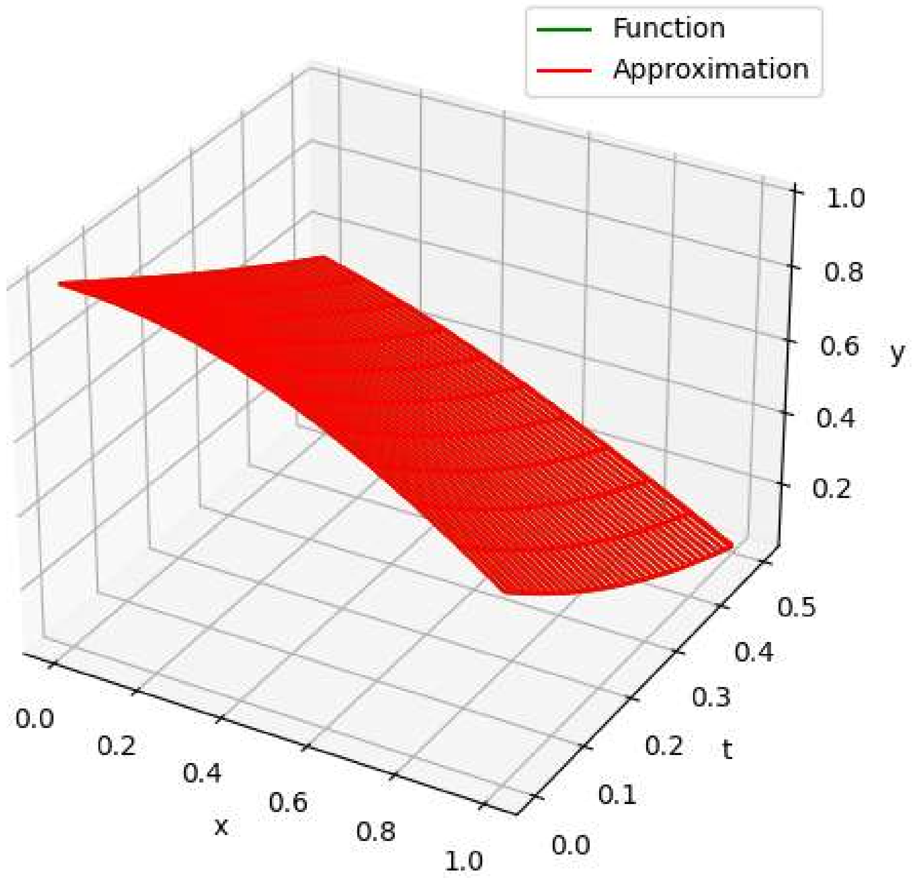
$\max\limits_{i,j} |\, y_i^j - u(x_i,\ t_j)| = 1127.868933310336$



Как видно из графика и погрешности, метод неустойчив при $\tau = h = 0.1$, т.к. условие его устойчивости $\tau \leq \frac{h^2}{2}$ не выполнено.

2. Явная разностная схема с $h = 0.1, \tau = \frac{h^2}{2}$

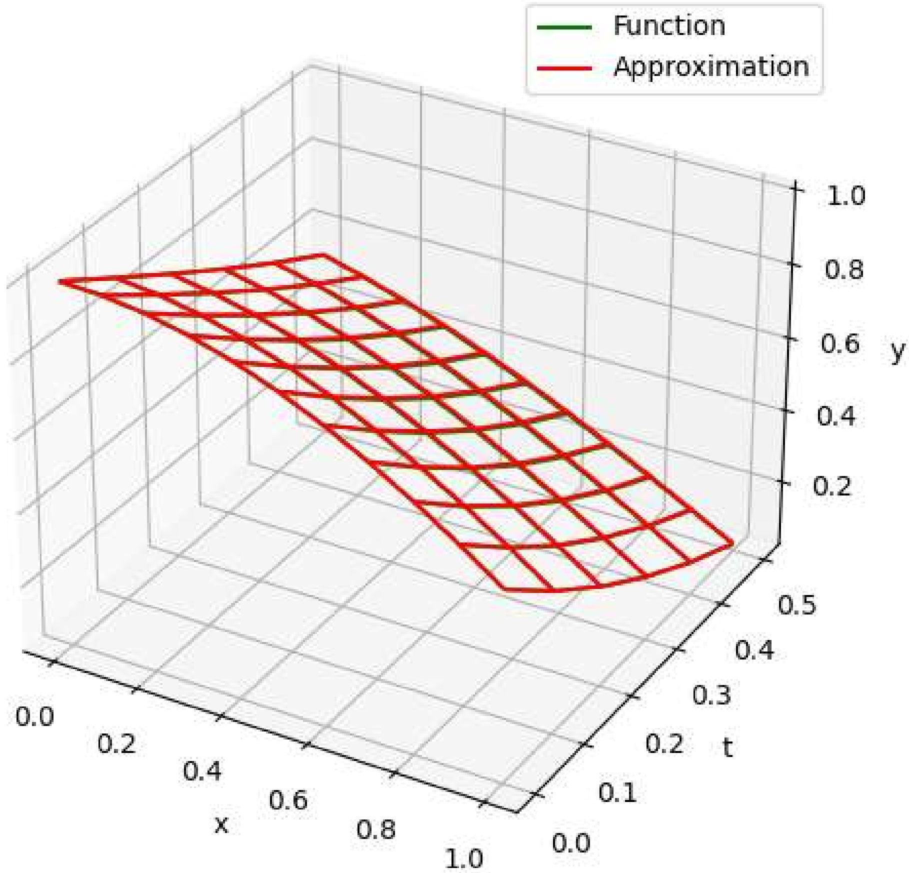
$\max_{i,j} | y_i^j - u(x_i,\ t_j)|$ = 0.00031326119148944453



Как видно из графика и погрешности, метод устойчив при $h = 0.1,\ \tau = \frac{h^2}{2} = 0.005$, т.к. условие его устойчивости $\tau \leq \frac{h^2}{2}$ выполнено.

3. Чисто неявная разностная схемы с $\tau = h = 0.1$

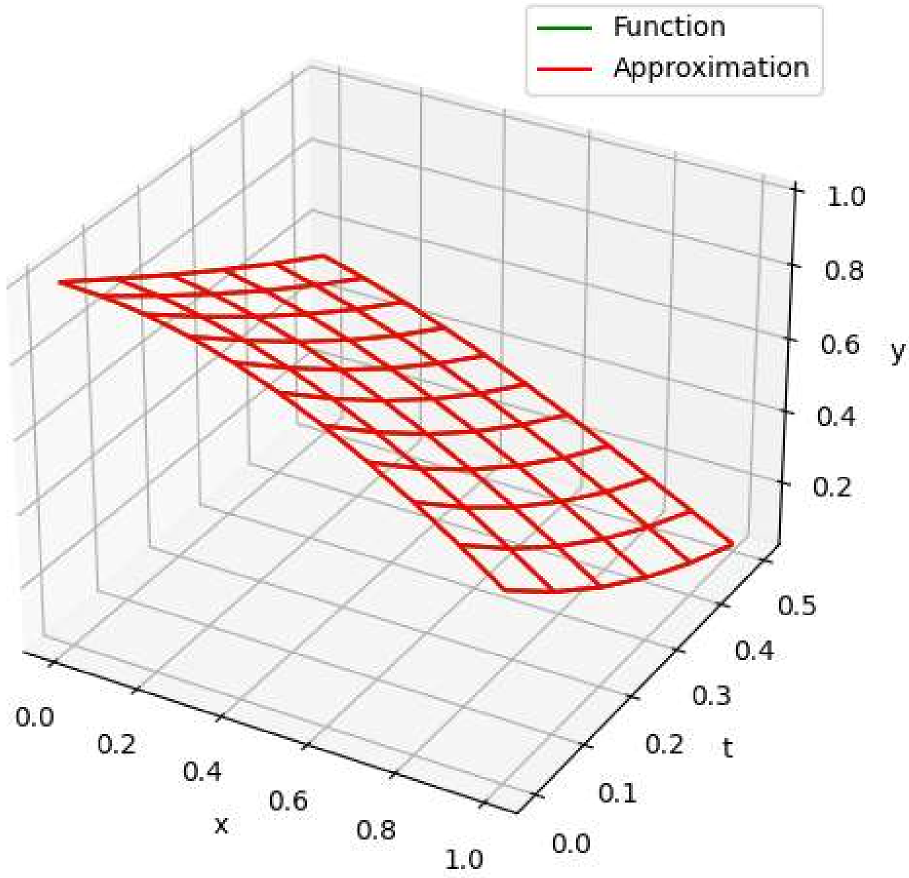
$\max\limits_{i,j} |\ y_i^j - u(x_i,\ t_j)| = 0.006208571445543987$



Как видно из графика и погрешности, метод устойчив при $\tau = h = 0.1$, т.к. он является абсолютно устойчивым.

4. Разностная схема Кранка-Николсон с $\tau = h = 0.1$

$\max\limits_{i,j} | y_i^j - u(x_i,\ t_j)|$ = 0.00010642457938037087



Как видно из графика и погрешности, метод устойчив при $\tau = h = 0.1$, т.к. он является абсолютно устойчивым.

Выводы

С помощью разностных схем можно достаточно точно найти численное решение смешанной задачи для одномерного уравнения теплопроводности. Для этого можно использовать как явный метод, так и неявные. Явный метод проще в реализации, но для его устойчивости необходим достаточно малый шаг по t. В свою очередь, неявный метод хоть и требует решение (t - 1) СЛАУ с трехдиагональной матрицей, абсолютно устойчивым и точнее, если использовать схему Кранка-Николсон.