Московский авиационный институт

(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6

по курсу “Численные методы”

Студент: Пясковский Н.Л.

Группа: М8О-409Б-19

Руководитель: Пивоваров Д.Е.

Оценка: _____

Москва 2022

1) 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.

2) Вариант

8.

$\frac{\partial^2 u}{\partial t^2} + 2\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + 2\frac{\partial u}{\partial x} - 3u,$

$u(0,t) = 0,$

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

$u(x,0) = 0,$

$u_t(x,0) = 2\exp(-x)\sin x.$

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

3) Теория

Классическим примером уравнения гиперболического типа является волновое уравнение, которое в области $0 < x < l$, $t > 0$ имеет вид:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < l, \quad t = 0.$$

Данное уравнение описывает, в частности, процесс малых поперечных колебаний струны. В этом случае *u(x,t)* - поперечные перемещения (колебания) струны, *a* – скорость распространения малых возмущений в материале, из которого изготовлена струна.

Если концы струны движутся по заданным законам, то есть на концах заданы перемещения (или значения искомой функции), *то первая начально-краевая задача для волнового уравнения имеет вид:*

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2}, & 0 < x < l, \quad t > 0; \\ u(0,t) = \varphi_0(t), & x = 0, \quad t > 0; \\ u(l,t) = \varphi_l(t), & x = l, \quad t > 0; \\ u(x,0) = \psi_1(x), & 0 \le x \le l, \quad t = 0; \\ \frac{\partial u(x,0)}{\partial t} = \psi_2(x), & 0 \le x \le l, \quad t = 0, \end{cases}$$

Если на концах струны заданы значения силы, которая по закону Гука пропорциональна значениям производной перемещения по пространственной переменной (то есть на концах заданы значения первых производных по переменной x), то ставится *вторая начально-краевая задача для волнового уравнения:*

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} = a^2 \dfrac{\partial^2 u}{\partial x^2}, \quad 0 < x < l, \quad t > 0; \\ \dfrac{\partial u(0,t)}{\partial x} = \varphi_0(t), \quad x = 0, \quad t > 0; \\ \dfrac{\partial u(l,t)}{\partial x} = \varphi_l(t), \quad x = l, \quad t > 0; \\ u(x,0) = \psi_1(x), \quad 0 \le x \le l, \quad t = 0; \\ \dfrac{\partial u(x,0)}{\partial t} = \psi_2(x), \quad 0 \le x \le l, \quad t = 0. \end{cases}$$

В условиях, когда концы струны *свободны*, функции $\varphi_0(t) = \varphi_l(t) = 0$.

Наконец в условиях, когда концы закреплены *упруго*, т.е. на концевые заделки действуют силы, пропорциональные перемещениям, ставится *третья начально-краевая задача для волнового уравнения:*

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} = a^2 \dfrac{\partial^2 u}{\partial x^2}, \quad 0 < x < l, \quad t > 0; \\ \alpha \dfrac{\partial u(0,t)}{\partial x} + \beta u(0,t) = \varphi_0(t), \quad x = 0, \quad t > 0; \\ \gamma \dfrac{\partial u(l,t)}{\partial x} + \delta u(l,t) = \varphi_l(t), \quad x = l, \quad t > 0; \\ u(x,0) = \psi_1(x), \quad 0 \le x \le l, \quad t = 0; \\ \dfrac{\partial u(x,0)}{\partial t} = \psi_2(x), \quad 0 \le x \le l, \quad t = 0. \end{cases}$$

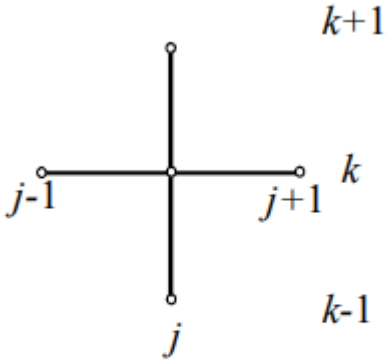
Рассмотрим первую начально-краевую задачу для волнового уравнения (5.33)-(5.37). На пространственно-временной сетке (5.12) будем аппроксимировать дифференциальное уравнение (5.33) одной из следующих конечно-разностных схем:

$$\frac{u_j^{k+1} - 2u_j^k + u_j^{k-1}}{\tau^2} = a^2 \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + O\left(\tau^2 + h^2\right), \quad j = \overline{1, N-1}; \quad k = 1,2,\ldots \qquad (5.38)$$

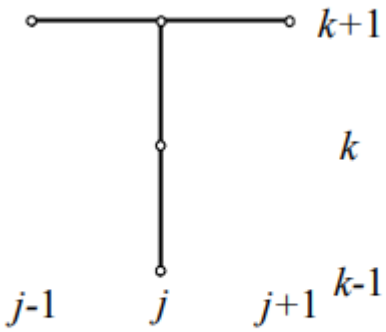
с шаблоном на рисунке 5.4а и

$$\frac{u_j^{k+1} - 2u_j^k + u_j^{k-1}}{\tau^2} = a^2 \frac{u_{j+1}^{k+1} - 2u_j^{k+1} + u_{j-1}^{k+1}}{h^2} + O\left(\tau + h^2\right), \quad j = \overline{1, N-1}; \quad k = 1,2,\ldots \qquad (5.39)$$

с шаблоном на рисунке 5.4 б



а)



б)

В обоих вариантах необходимо знать значения u_j^{k-1} и u_j^k на нижних временных слоях.
Для k=1:
$u_j^0 = \psi_1(x_j),\ j = \overline{0,N},$
Для определения же u_j^1 воспользуемся простейшей аппроксимацией второго начального условия:

$$\frac{u_j^1 - u_j^0}{\tau} = \psi_2(x_j)$$

откуда получим выражение:
$u_j^1 = \psi_1(x_j) + \psi_2(x_j)\tau$
Для повышения порядка аппроксимации можно воспользоваться другим способом. Для этого разложим u_j^1 в ряд Тейлора:

$$u_j^1 = u(x_j, 0+\tau) = u_j^0 + \left.\frac{\partial u}{\partial t}\right|_j^0 \tau + \left.\frac{\partial^2 u}{\partial t^2}\right|_j^0 \frac{\tau^2}{2} + O(\tau^3)$$

и воспользуемся исходным уравнением:

$$\left.\frac{\partial^2 u}{\partial t^2}\right|_j^0 = a^2 \left.\frac{\partial^2 u}{\partial x^2}\right|_j^0 = a^2 \psi_1''(x_j)$$

откуда получим функцию со вторым порядком точности:

$$u_j^1 = \psi_1(x_j) + \psi_2(x_j)\tau + a^2 \psi_1''(x_j)\frac{\tau^2}{2}$$

4) Программа

```
import numpy as np
from matplotlib import pyplot as plt
import warnings

T = 1  # верхняя граница времени
N = 50  # количество отрезков x
```

```
K = 1000  # количество отрезков t
l = np.pi  # правая граница координат
a = 1  #
b = 2  #
c = -3  #
e = 2  #
h = l / N  # пространственный шаг
tau = T / K  # шаг по времени
sigma = a * tau / (h ** 2)  # число Куранта, используется для анализа устойчивости решения


# аналитическое решение
def analytic_solution(x, t):
    return np.exp(-t - x) * np.sin(x) * np.sin(2 * t)


# условия
def phi0(t):
    return 0


def phi1(t):
    return 0


def psi1(x):
    return 0


def psi2(x):
    return 2 * np.exp(-x) * np.sin(x)


# явная (крест) конечно-разностная схема
def explicit_scheme(bound_condition, initial_condition):
    if sigma > 1:
        warnings.warn("Sigma > 1")

    u = np.zeros((K, N))

    for i in range(0, N):
        u[0][i] = psi1(i * h)

        if initial_condition == 1:
            u[1][i] = psi1(i * h) + psi2(i * h) * tau + 0
        elif initial_condition == 2:
            u[1][i] = psi1(i * h) + tau * psi2(i * h) + 0.5 * (tau ** 2) * (-e * psi2(i * h) + c * psi1(i * h))
        else:
            warnings.warn("Approximation type not found")
```

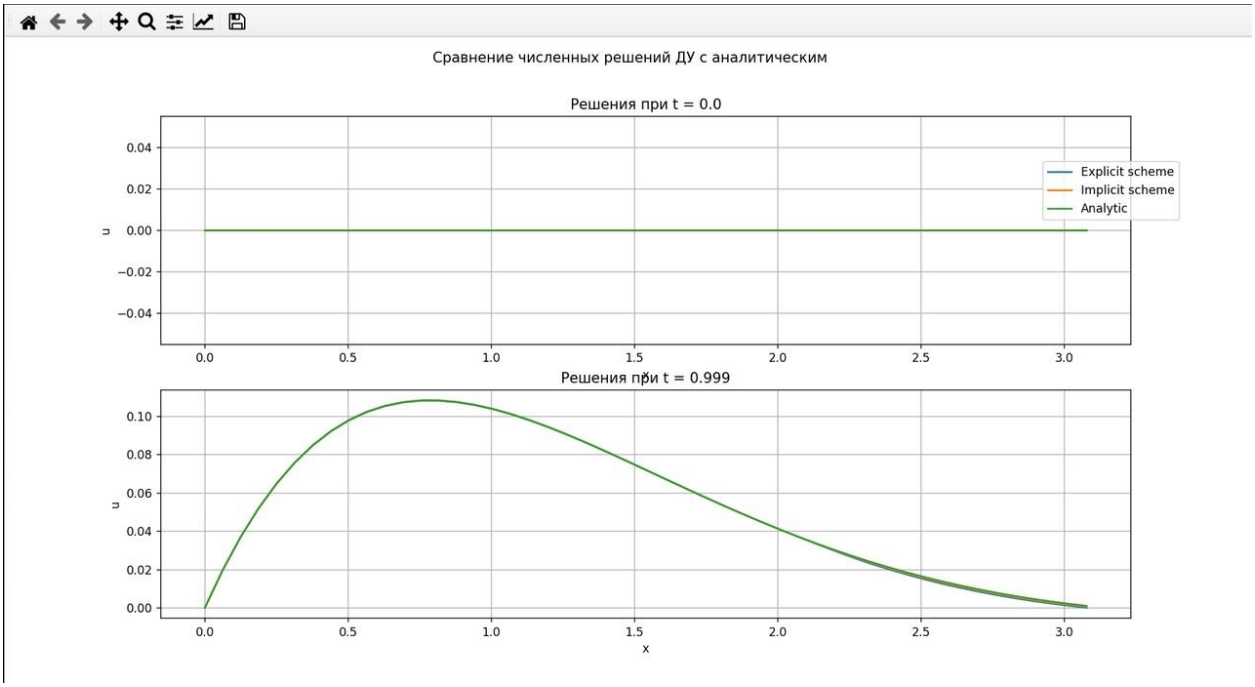
```
    for k in range(2, K):
        for i in range(1, N - 1):
            u[k][i] = (4 * u[k - 1][i] + (e * tau - 2) * u[k - 2][i] + 2 * a * (tau ** 2) * (
                    u[k - 1][i - 1] - 2 * u[k - 1, i] + u[k - 1][i + 1]) / (h ** 2) + b * (tau ** 2) * (
                            u[k - 1][i + 1] - u[k - 1][i - 1]) / h + 2 * (tau ** 2) * (c * u[k - 1][i])) / (
                           2 + e * tau)

        if bound_condition == 1:
            u[k][0] = 0
            u[k][-1] = 0
        elif bound_condition == 2:
            u[k][0] = (phi0(k * tau) + u[k][2] / (2 * h) - 2 * u[k][1] / h) * 2 * h / -3
            u[k][-1] = phi1(k * tau)
        elif bound_condition == 3:
            u[k][0] = 2 * h * phi1(k * tau)
            u[k][-1] = 2*h*phi1(k * tau)
        else:
            warnings.warn("Bound condition not found")

    return u


# неявная конечно-разностная схема
def implicit_scheme(bound_condition, initial_condition):
    u = np.zeros((K, N))

    ai = np.zeros(N)
    bi = np.zeros(N)
    ci = np.zeros(N)
    di = np.zeros(N)

    for i in range(0, N):
        u[0][i] = psi1(i * h)

        if initial_condition == 1:
            u[1][i] = psi1(i * h) + psi2(i * h) * tau
        elif initial_condition == 2:
            u[1][i] = psi1(i * h) + tau * psi2(i * h) + 0.5 * (tau ** 2) * (-e * psi2(i * h) + c * psi1(i * h))
        else:
            warnings.warn("Approximation type not found")

    for k in range(2, K):
        for i in range(1, N - 1):
            ai[i] = 2 * a - h * b
            bi[i] = 2 * (h ** 2) * (-e / (2 * tau) - 1 / (tau ** 2) + c) - 4 * a
            ci[i] = h * b + 2 * a
            di[i] = -4 * (h ** 2) * u[k - 1][i] / (tau ** 2) + (2 * (h ** 2) / (tau ** 2) - e * (h ** 2) / tau) *
\
```

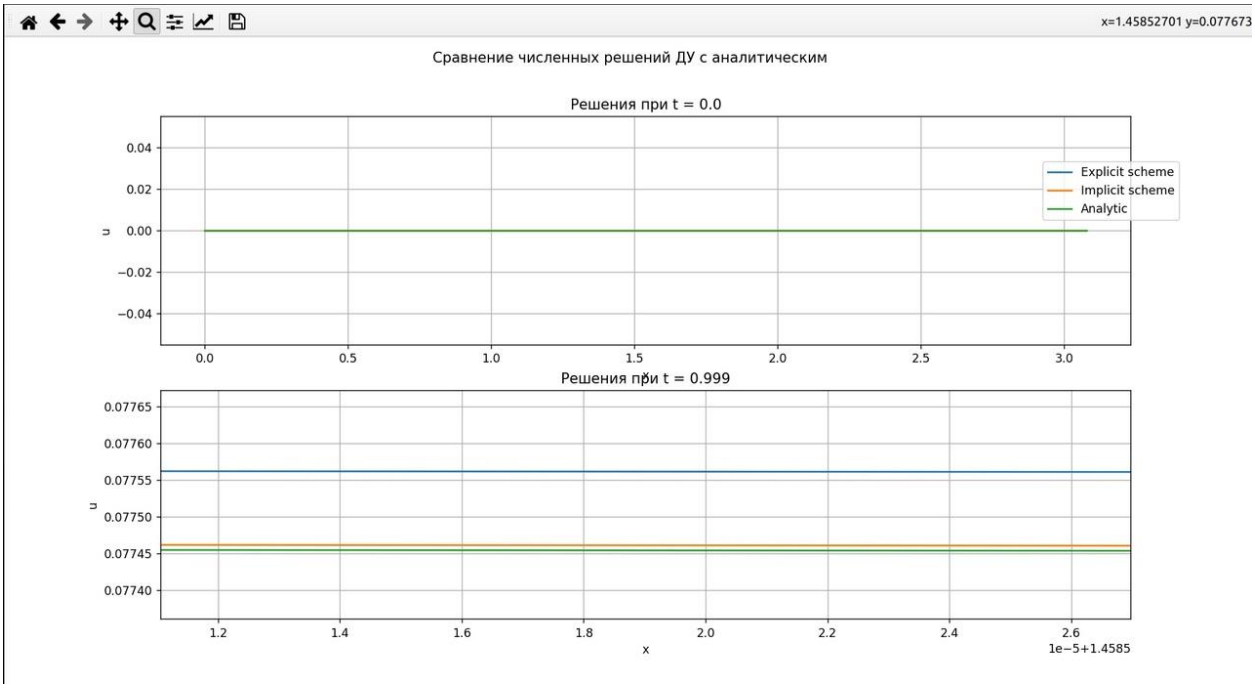
```
                u[k - 2][i]

        if bound_condition == 1:
            bi[0] = h
            ci[0] = 0
            di[0] = h * phi0(k * tau)
            ai[-1] = 2 * sigma
            bi[-1] = -(1 + 2 * sigma - c * tau)
            # di[-1] = -(u[k - 1][-1] + sigma * phi1(k * tau))
            di[-1] = -phi1(k * tau)
        elif bound_condition == 2:
            bi[0] = -(1 + 2 * sigma - c * tau)
            ci[0] = 2 * sigma
            di[0] = -(u[k - 1][0] - 2 * a * tau * phi0(k * tau) / h)
            ai[-1] = 2 * sigma
            bi[-1] = -(1 + 2 * sigma - c * tau)
            # di[-1] = -(u[k - 1][-1] + sigma * phi1(k * tau))
            di[-1] = -phi1(k * tau)
        elif bound_condition == 3:
            bi[0] = -(1 + 2 * sigma - c * tau)
            ci[0] = 2 * sigma
            di[0] = -((1 - sigma) * u[k - 1][1] + sigma / 2 * u[k - 1][0]) - sigma * phi0(k * tau)
            ai[-1] = 2 * sigma
            bi[-1] = -(1 + 2 * sigma - c * tau)
            di[-1] = -phi1(k * tau)
        else:
            warnings.warn("Bound condition not found")

        u[k] = thomas_algorithm(ai, bi, ci, di)

    return u


# вывод решения в виде графиков
def show_result(t_axis, x_axis, u1, u2):
    fig, ax = plt.subplots(2)
    fig.suptitle('Сравнение численных решений ДУ с аналитическим')
    fig.set_figheight(15)
    fig.set_figwidth(16)
    time = 0
    for i in range(2):
        ax[i].plot(x_axis, u1[time, :], label='Explicit scheme')
        ax[i].plot(x_axis, u2[time, :], label='Implicit scheme')
        ax[i].plot(x_axis, [analytic_solution(x, t_axis[time]) for x in x_axis], label='Analytic')
        ax[i].grid(True)
        ax[i].set_xlabel('x')
        ax[i].set_ylabel('u')
        ax[i].set_title(f'Решения при t = {time / K}')
        time += K - 1
```

```
    plt.legend(bbox_to_anchor=(1.05, 2), loc='upper right', borderaxespad=0)
    plt.show()

    fig = plt.figure(num=1, figsize=(19, 12), clear=True)
    ax = fig.add_subplot(1, 1, 1, projection='3d')
    fig.suptitle('Аналитическое решение')
    xgrid, tgrid = np.meshgrid(x_axis, t_axis)
    ax.plot_surface(xgrid, tgrid, analytic_solution(xgrid, tgrid))
    ax.set(xlabel='x', ylabel='t', zlabel='u')
    fig.tight_layout()
    plt.show()


# вывод изменения ошибки со временем
def show_inaccuracy(t_axis, x_axis, u):
    inaccuracy = np.zeros(K)
    for i in range(K):
        inaccuracy[i] = np.max(np.abs(u[i] - np.array([analytic_solution(x, t_axis[i]) for x in
x_axis])))

    plt.figure(figsize=(14, 8))
    plt.plot(t_axis[1:], inaccuracy[1:], 'violet', label='Ошибка')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper right', borderaxespad=0.)
    plt.title('График изменения ошибки во времени')
    plt.xlabel('t')
    plt.ylabel('error')
    plt.grid(True)
    plt.show()


# метод прогонки
def thomas_algorithm(a, b, c, d):
    size = len(a)
    P = np.zeros(size)
    Q = np.zeros(size)
    P[0] = -c[0] / b[0]
    Q[0] = d[0] / b[0]

    for i in range(1, size):
        s = (b[i] + a[i] * P[i - 1])
        P[i] = -c[i] / s
        Q[i] = (d[i] - a[i] * Q[i - 1]) / s

    result = np.zeros(size)
    result[-1] = Q[-1]

    for i in range(size - 2, -1, -1):
        result[i] = P[i] * result[i + 1] + Q[i]
```

```
    return result


# Первый аргумент схем - тип аппроксимации граничных условий, содержащих производные
# 1 - двухточечная аппроксимация с первым порядком
# 2 - трехточечная аппроксимация со вторым порядком
# 3 - двухточечная аппроксимация со вторым порядком
# В выданном мне 8-м варианте производных в краевых условиях нет
# Второй аргумент схем - тип аппроксимации начальных условий - 1-ый или 2-ой порядок
def main():
    u1 = explicit_scheme(1, 1)
    u2 = implicit_scheme(1, 1)

    t_axis = np.zeros(K)
    for i in range(K):
        t_axis[i] = tau * i

    x_axis = np.zeros(N)
    for i in range(N):
        x_axis[i] = i * h

    show_result(t_axis, x_axis, u1, u2)

    show_inaccuracy(t_axis, x_axis, u1)


if __name__ == '__main__':
    main()
```

5) Результат


Сравнение численных решений ДУ с аналитическим
Решения при t = 0.0
Explicit scheme
Implicit scheme
Analytic
u
Решения при t = 0.999
u
x


x=1.45852701 y=0.077673
Сравнение численных решений ДУ с аналитическим
Решения при t = 0.0
Explicit scheme
Implicit scheme
Analytic
u
Решения при t = 0.999
u
x
1e−5+1.4585

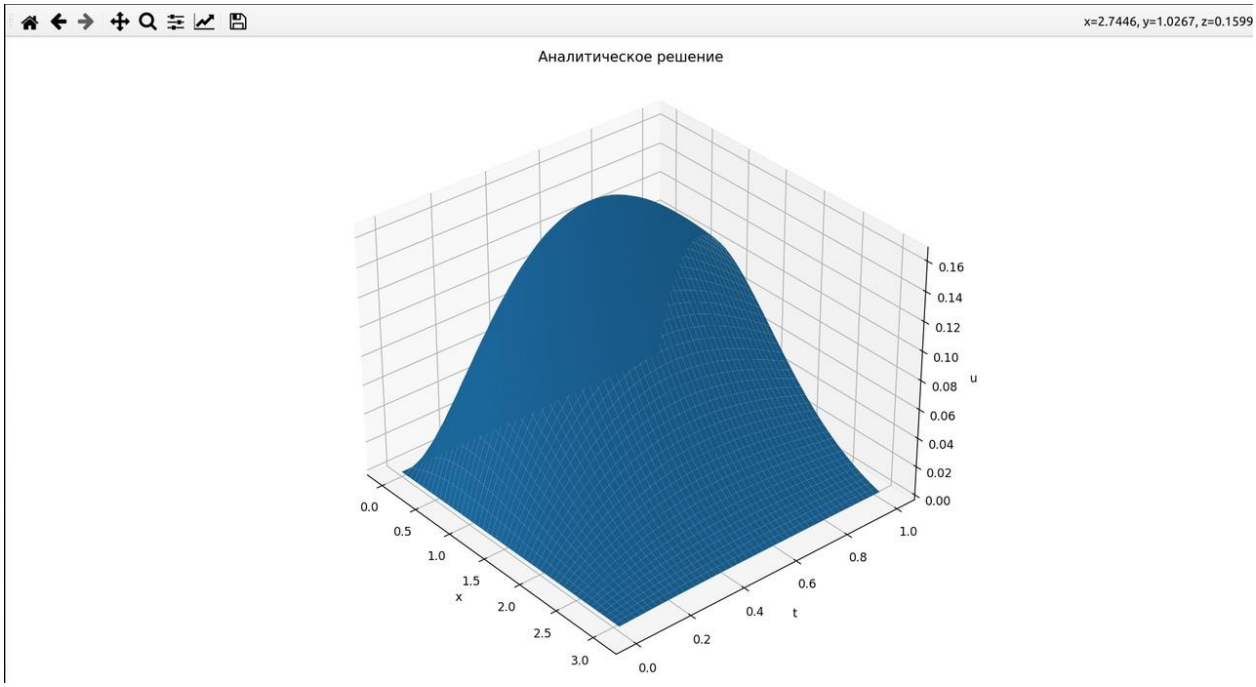


График изменения ошибки во времени для явной схемы:

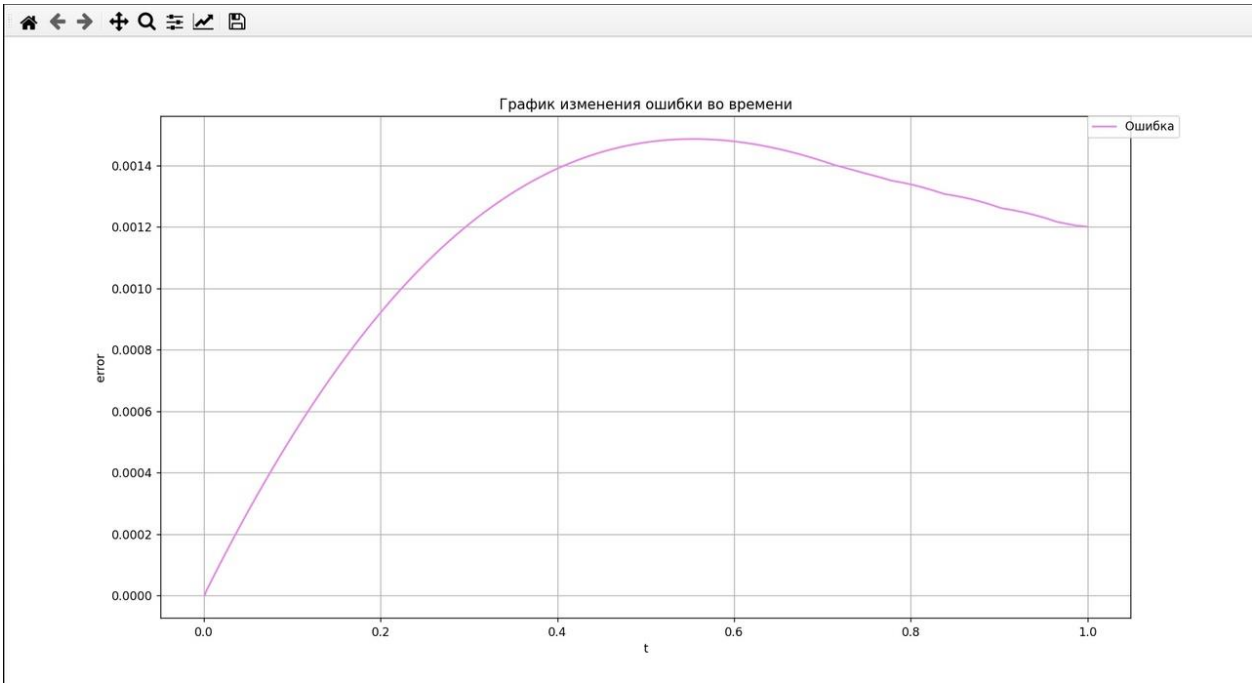


График изменения ошибки во времени
Ошибка
error
t
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0.0012
0.0014
0.0
0.2
0.4
0.6
0.8
1.0