МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторные работы**

**по курсу «Численные методы»**

Выполнил: А.С. Федоров

Преподаватель: Д.Л. Ревизников

Группа: 8О-407Б Дата:

Оценка:

Подпись:

Москва, 2022

**5 Численные методы решения дифференциальных уравнений с частными производными.**

**5.1 Параболические одномерные уравнения**

**Задача**

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 . Исследовать зависимость погрешности от сеточных параметров .

**Вариант 10**

,  , .



.

Аналитическое решение: .

Исходный код

import math

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.widgets import Slider, Button, TextBox

def Explicit\_schema(u, a, b, c, tau):

    time\_steps, n = u.shape

    for k in range(0, time\_steps-1):

        u[k+1, 0] = 0

        u[k+1, n-1] = 0

        for i in range(1, n-1):

            u[k+1, i] = ((a\*tau)/(h\*\*2))\*(u[k, i-1]-2\*u[k, i]+u[k, i+1])+ \

                        ((b\*tau)/(2\*h))\*(u[k, i+1]-u[k, i-1])+ \

                        c\*u[k, i]\*tau+ \

                        0\*tau+ \

                        u[k, i]

def Implicit\_schema(u, a, b, c, tau):

    time\_steps, n = u.shape

    alpha = (a\*tau)/(h\*\*2)-(b\*tau)/(2\*h)

    beta = -1-2\*(a\*tau)/(h\*\*2)+c\*tau

    gamma = (a\*tau)/(h\*\*2)+(b\*tau)/(2\*h)

    A = np.zeros((n, n))

    A[0, 0] = 1

    A[n-1, n-1] = 1

    for i in range(1, n-1):

        A[i, i-1] = alpha

        A[i, i] = beta

        A[i, i+1] = gamma

    for k in range(1, time\_steps):

        b = -u[k-1]

        u[k] = np.linalg.solve(A, b)

def Combined\_scheme(u, a, b, c, tau):

    tetha = 1/2

    time\_steps, n = u.shape

    alpha = ((a\*tau)/(h\*\*2)-(b\*tau)/(2\*h))\*(tetha)

    beta = (-1)+(((-2)\*a\*tau)/(h\*\*2)+c\*tau)\*(tetha)

    gamma = ((a\*tau)/(h\*\*2)+(b\*tau)/(2\*h))\*(tetha)

    A = np.zeros((n, n))

    A[0, 0] = 1

    A[n-1, n-1] = 1

    for i in range(1, n-1):

        A[i, i-1] = alpha

        A[i, i] = beta

        A[i, i+1] = gamma

    for k in range(1, time\_steps):

        explicit\_part = np.empty(n)

        explicit\_part[0] = 0

        explicit\_part[-1] = 0

        for i in range(1, n-1):

            explicit\_part[i] = ((a\*tau)/(h\*\*2))\*(u[k-1, i-1]-2\*u[k-1, i]+u[k-1, i+1])+ \

                               ((b\*tau)/(2\*h))\*(u[k-1, i+1]-u[k-1, i-1])+ \

                               c\*u[k-1, i]\*tau+ \

                               0\*tau

        d = -u[k-1]-(1-tetha)\*explicit\_part

        u[k] = np.linalg.solve(A, d)

class Solver:

    def \_\_init\_\_(self, a, b, c, u\_start, left\_border\_condition, right\_border\_condition, left\_border, right\_border, n, sigma, end\_time) -> None:

        self.a = a

        self.b = b

        self.c = c

        self.u\_start = u\_start

        self.left\_border = left\_border

        self.right\_border = right\_border

        self.l = right\_border-left\_border

        self.n = n

        self.sigma = sigma

        self.end\_time = end\_time

        self.h = self.l/(n-1)

        self.time\_steps = int((end\_time\*a\*n\*\*2)/(sigma\*self.l\*\*2))-1

        self.tau = (sigma\*self.l\*\*2)/(a\*n\*\*2)

        self.left\_border\_condition = left\_border\_condition

        self.right\_border\_condition = right\_border\_condition

        self.left\_a = 1

        self.left\_b = 1

        self.right\_a = 1

        self.right\_b = 1

    def solve(self, scheme, boundary\_conditions\_interpolation):

        u = np.zeros((self.time\_steps, self.n))

        u[0] = self.u\_start(np.linspace(self.left\_border, self.right\_border+self.h, self.n))

        if scheme == 'explicit':

            for k in range(0, self.time\_steps-1):

                for i in range(1, self.n-1):

                    u[k+1, i] = ((self.a\*self.tau)/(self.h\*\*2))\*(u[k, i-1]-2\*u[k, i]+u[k, i+1])+ \

                                ((self.b\*self.tau)/(2\*self.h))\*(u[k, i+1]-u[k, i-1])+ \

                                self.c\*u[k, i]\*self.tau+ \

                                0\*self.tau+ \

                                u[k, i]

                if boundary\_conditions\_interpolation == '2\_points\_1st\_order':

                    u[k+1, 0] = (self.left\_border\_condition((k+1)\*self.tau, self.a, self.b, self.c)-(self.left\_a/self.h)\*u[k+1, 1])/(-(self.left\_a/self.h)+self.left\_b)

                    u[k+1, -1] = (self.right\_border\_condition((k+1)\*self.tau, self.a, self.b, self.c)+(self.right\_a/self.h)\*u[k+1, -2])/ \

                                       ((self.right\_a/self.h)+self.right\_b)

                if boundary\_conditions\_interpolation == '3\_points\_2nd\_order':

                    u[k+1, 0] = (self.left\_border\_condition((k+1)\*self.tau, self.a, self.b, self.c)-((4\*self.left\_a)/(2\*self.h)\*u[k+1, 1])+(self.left\_a/(2\*self.h))\*u[k+1, 2])/ \

                                (((-3)\*self.left\_a)/(2\*self.h)+self.left\_b)

                    u[k+1, -1] = (self.right\_border\_condition((k+1)\*self.tau, self.a, self.b, self.c)+((4\*self.right\_a)/(2\*self.h)\*u[k+1, -2])-(self.right\_a/(2\*self.h))\*u[k+1, -3])/ \

                                       ((3\*self.right\_a)/(2\*self.h)+self.right\_b)

                if boundary\_conditions\_interpolation == '2\_points\_2nd\_order':

                    u[k+1, 0] = (self.left\_border\_condition((k+1)\*self.tau, self.a, self.b, self.c)-u[k+1, 1]\*(self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))-u[k, 0]\*(self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(self.h\*\*2/(2\*self.a\*self.tau)))/((self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(-1-(self.h\*\*2)/(2\*self.a\*self.tau)+(self.c\*self.h\*\*2)/(2\*self.a))+self.left\_b)

                    u[k+1, -1] = (self.right\_border\_condition((k+1)\*self.tau, self.a, self.b, self.c)-u[k+1, -2]\*(self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))-u[k, -1]\*(self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(self.h\*\*2/(2\*self.a\*self.tau)))/((self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(-1-(self.h\*\*2)/(2\*self.a\*self.tau)+(self.c\*self.h\*\*2)/(2\*self.a))+self.right\_b)

        if scheme == 'implicit':

            alpha = ((self.a\*self.tau)/(self.h\*\*2))-((self.b\*self.tau)/(2\*self.h))

            beta = -1-(2\*(self.a\*self.tau)/(self.h\*\*2))+(self.c\*self.tau)

            gamma = ((self.a\*self.tau)/(self.h\*\*2))+((self.b\*self.tau)/(2\*self.h))

            A = np.zeros((self.n, self.n))

            if boundary\_conditions\_interpolation == '2\_points\_1st\_order':

                A[0, 0] = (-(self.left\_a/self.h)+self.left\_b)

                A[0, 1] = (self.left\_a/self.h)

                A[-1, -1] = ((self.right\_a/self.h)+self.right\_b)

                A[-1, -2] = -(self.right\_a/self.h)

            if boundary\_conditions\_interpolation == '3\_points\_2nd\_order':

                A[0, 0] = (((-3)\*self.left\_a)/(2\*self.h)+self.left\_b)

                A[0, 1] = (4\*self.left\_a)/(2\*self.h)

                A[0, 2] = (-self.left\_a)/(2\*self.h)

                A[-1, -1] = ((3\*self.right\_a)/(2\*self.h)+self.right\_b)

                A[-1, -2] = ((-4)\*self.right\_a)/(2\*self.h)

                A[-1, -3] = (self.right\_a)/(2\*self.h)

            if boundary\_conditions\_interpolation == '2\_points\_2nd\_order':

                A[0, 0] = (self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(-1-(self.h\*\*2)/(2\*self.a\*self.tau)+(self.c\*self.h\*\*2)/(2\*self.a))+self.left\_b

                A[0, 1] = self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a))

                A[-1, -1] = ((self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(-1-(self.h\*\*2)/(2\*self.a\*self.tau)+(self.c\*self.h\*\*2)/(2\*self.a))+self.right\_b)

                A[-1, -2] = (self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))

            for i in range(1, self.n-1):

                A[i, i-1] = alpha

                A[i, i] = beta

                A[i, i+1] = gamma

            for k in range(1, self.time\_steps):

                d = -u[k-1].copy()

                d[0] = self.left\_border\_condition(k\*self.tau, self.a, self.b, self.c)

                d[-1] = self.right\_border\_condition(k\*self.tau, self.a, self.b, self.c)

                if boundary\_conditions\_interpolation == '2\_points\_2nd\_order':

                    d[0] -= u[k-1, 0]\*(self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(self.h\*\*2/(2\*self.a\*self.tau))

                    d[-1] -= u[k-1, -1]\*(self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(self.h\*\*2/(2\*self.a\*self.tau))

                u[k] = np.linalg.solve(A, d)

        if scheme == 'combined':

            tetha = 1/2

            alpha = ((self.a\*self.tau)/(self.h\*\*2)-(self.b\*self.tau)/(2\*self.h))\*(tetha)

            beta = (-1)+(((-2)\*self.a\*self.tau)/(self.h\*\*2)+self.c\*self.tau)\*(tetha)

            gamma = ((self.a\*self.tau)/(self.h\*\*2)+(self.b\*self.tau)/(2\*self.h))\*(tetha)

            A = np.zeros((self.n, self.n))

            if boundary\_conditions\_interpolation == '2\_points\_1st\_order':

                A[0, 0] = (-(self.left\_a/self.h)+self.left\_b)

                A[0, 1] = (self.left\_a/self.h)

                A[-1, -1] = ((self.right\_a/self.h)+self.right\_b)

                A[-1, -2] = -(self.right\_a/self.h)

            if boundary\_conditions\_interpolation == '3\_points\_2nd\_order':

                A[0, 0] = (((-3)\*self.left\_a)/(2\*self.h)+self.left\_b)

                A[0, 1] = (4\*self.left\_a)/(2\*self.h)

                A[0, 2] = (-self.left\_a)/(2\*self.h)

                A[-1, -1] = ((3\*self.right\_a)/(2\*self.h)+self.right\_b)

                A[-1, -2] = ((-4)\*self.right\_a)/(2\*self.h)

                A[-1, -3] = (self.right\_a)/(2\*self.h)

            if boundary\_conditions\_interpolation == '2\_points\_2nd\_order':

                A[0, 0] = (self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(-1-(self.h\*\*2)/(2\*self.a\*self.tau)+(self.c\*self.h\*\*2)/(2\*self.a))+self.left\_b

                A[0, 1] = self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a))

                A[-1, -1] = ((self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(-1-(self.h\*\*2)/(2\*self.a\*self.tau)+(self.c\*self.h\*\*2)/(2\*self.a))+self.right\_b)

                A[-1, -2] = (self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))

            for i in range(1, self.n-1):

                A[i, i-1] = alpha

                A[i, i] = beta

                A[i, i+1] = gamma

            for k in range(1, self.time\_steps):

                explicit\_part = np.empty(self.n)

                for i in range(1, self.n-1):

                    explicit\_part[i] = ((self.a\*self.tau)/(self.h\*\*2))\*(u[k-1, i-1]-2\*u[k-1, i]+u[k-1, i+1])+ \

                                       ((self.b\*self.tau)/(2\*self.h))\*(u[k-1, i+1]-u[k-1, i-1])+ \

                                       self.c\*u[k-1, i]\*self.tau+ \

                                       0\*self.tau

                if boundary\_conditions\_interpolation == '2\_points\_1st\_order':

                    explicit\_part[0] = (self.left\_border\_condition(k\*self.tau, self.a, self.b, self.c)-(self.left\_a/self.h)\*explicit\_part[1])/(-(self.left\_a/self.h)+self.left\_b)

                    explicit\_part[self.n-1] = (self.right\_border\_condition(k\*self.tau, self.a, self.b, self.c)+(self.right\_a/self.h)\*explicit\_part[-2])/ \

                                       ((self.right\_a/self.h)+self.right\_b)

                if boundary\_conditions\_interpolation == '3\_points\_2nd\_order':

                    explicit\_part[0] = (self.left\_border\_condition(k\*self.tau, self.a, self.b, self.c)-((4\*self.left\_a)/(2\*self.h)\*explicit\_part[1])+(self.left\_a/(2\*self.h))\*explicit\_part[2])/ \

                                (((-3)\*self.left\_a)/(2\*self.h)+self.left\_b)

                    explicit\_part[self.n-1] = (self.right\_border\_condition(k\*self.tau, self.a, self.b, self.c)+((4\*self.right\_a)/(2\*self.h)\*explicit\_part[-2])-(self.right\_a/(2\*self.h))\*explicit\_part[-3])/ \

                                       ((3\*self.right\_a)/(2\*self.h)+self.right\_b)

                if boundary\_conditions\_interpolation == '2\_points\_2nd\_order':

                    explicit\_part[0] = (self.left\_border\_condition((k)\*self.tau, self.a, self.b, self.c)-explicit\_part[1]\*(self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))-u[k-1, 0]\*(self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(self.h\*\*2/(2\*self.a\*self.tau)))/((self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(-1-(self.h\*\*2)/(2\*self.a\*self.tau)+(self.c\*self.h\*\*2)/(2\*self.a))+self.left\_b)

                    explicit\_part[-1] = (self.right\_border\_condition((k)\*self.tau, self.a, self.b, self.c)-explicit\_part[-2]\*(self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))-u[k-1, -1]\*(self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(self.h\*\*2/(2\*self.a\*self.tau)))/((self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(-1-(self.h\*\*2)/(2\*self.a\*self.tau)+(self.c\*self.h\*\*2)/(2\*self.a))+self.right\_b)

                d = -u[k-1].copy()

                d[0] = self.left\_border\_condition(k\*self.tau, self.a, self.b, self.c)

                d[-1] = self.right\_border\_condition(k\*self.tau, self.a, self.b, self.c)

                if boundary\_conditions\_interpolation == '2\_points\_2nd\_order':

                    d[0] -= u[k-1, 0]\*(self.left\_a/(self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(self.h\*\*2/(2\*self.a\*self.tau))

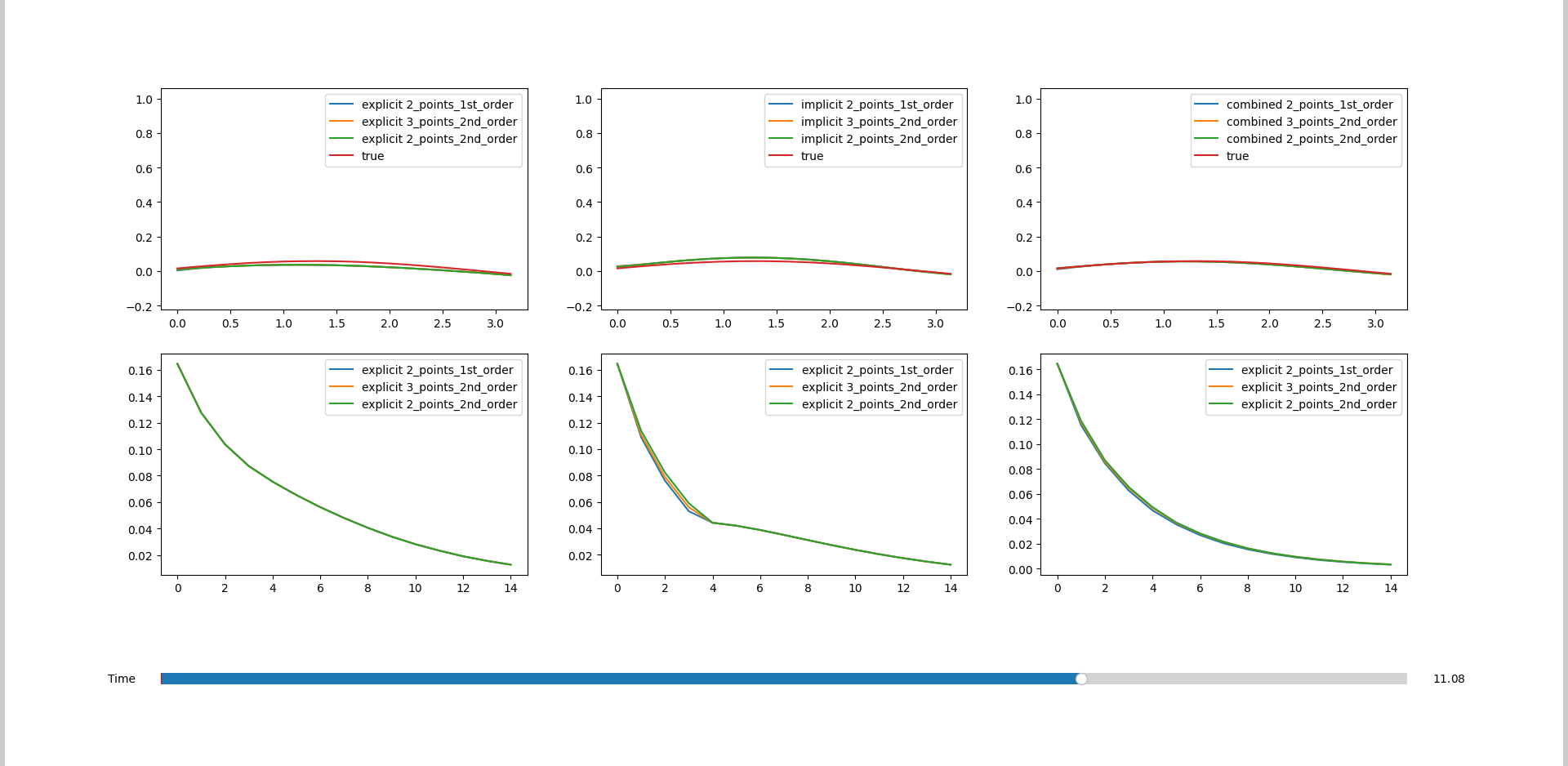
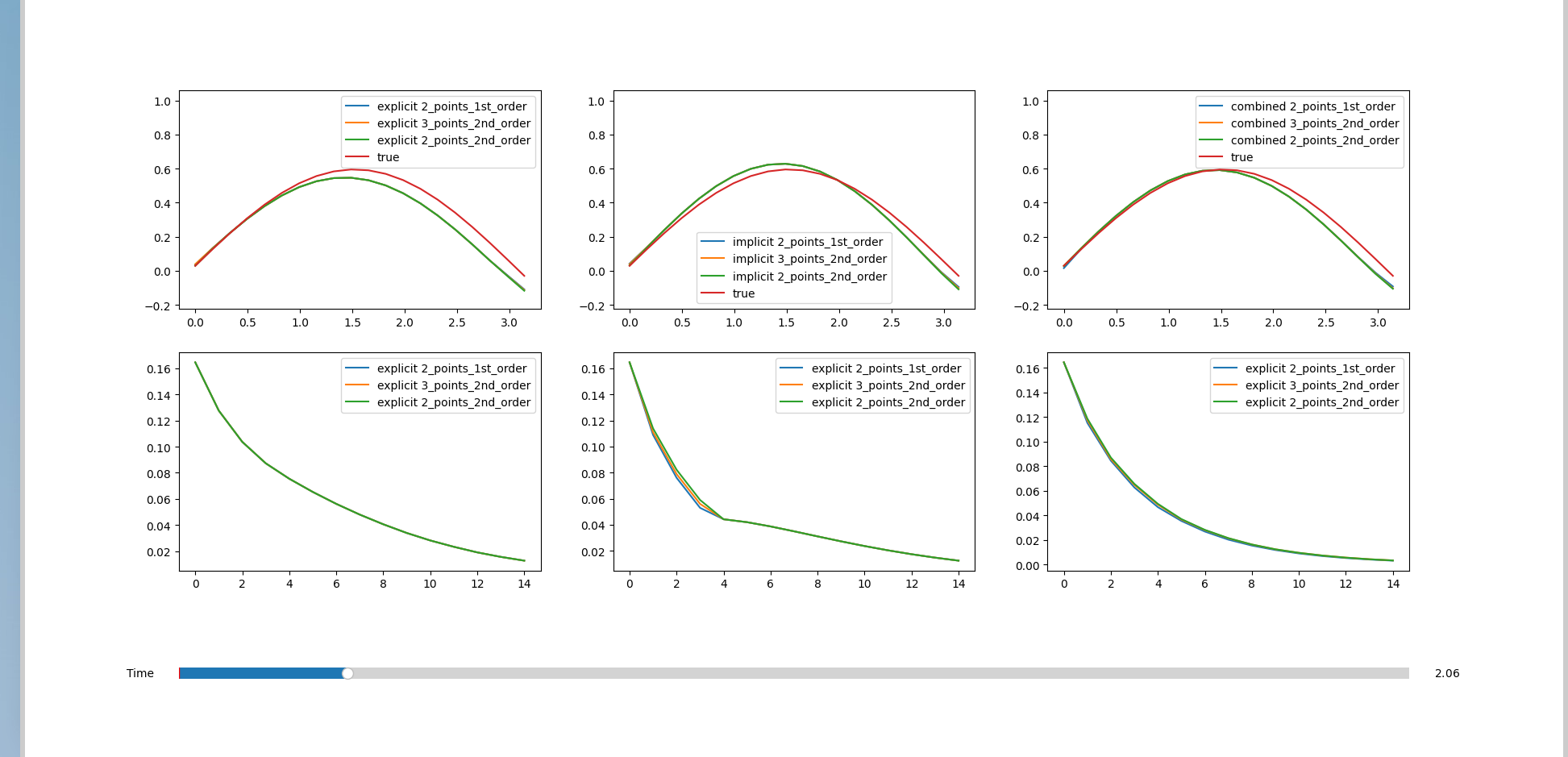
                    d[-1] -= u[k-1, -1]\*(self.right\_a/(-self.h-(self.b\*self.h\*\*2)/(2\*self.a)))\*(self.h\*\*2/(2\*self.a\*self.tau))

                d = d-(1-tetha)\*explicit\_part

                u[k] = np.linalg.solve(A, d)

        return u

Результат работы



**5.2 Гиперболические одномерные уравнения**

**Задача**

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 . Исследовать зависимость погрешности от сеточных параметров .

**Вариант 10**

,



,

.

Аналитическое решение: .

Исходный код

import math

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.widgets import Slider, Button

class Solver:

    def \_\_init\_\_(self, a, b, c, d, f, u\_start, dt\_u\_start, left\_border\_condition, right\_border\_condition, left\_border, right\_border, n, sigma, end\_time) -> None:

        self.a = a

        self.b = b

        self.c = c

        self.d = d

        self.f = f

        self.u\_start = u\_start

        self.dt\_u\_start = dt\_u\_start

        self.left\_border = left\_border

        self.right\_border = right\_border

        self.l = right\_border-left\_border

        self.n = n

        self.sigma = sigma

        self.end\_time = end\_time

        self.h = self.l/(n-1)

        self.time\_steps = int((end\_time\*a\*\*2\*n)/(sigma\*self.l))-1

        self.tau = (sigma\*self.l)/(a\*\*2\*n)

        self.left\_border\_condition = left\_border\_condition

        self.right\_border\_condition = right\_border\_condition

        self.left\_a = 1

        self.left\_b = 1

        self.right\_a = 1

        self.right\_b = 1

    def solve(self, scheme, boundary\_conditions\_interpolation):

        a = self.a

        b = self.b

        c = self.c

        d = self.d

        f = self.f

        u\_start = self.u\_start

        dt\_u\_start = self.dt\_u\_start

        left\_border = self.left\_border

        right\_border = self.right\_border

        l = self.l

        n = self.n

        sigma = self.sigma

        end\_time = self.end\_time

        h = self.h

        time\_steps = self.time\_steps

        tau = self.tau

        left\_border\_condition = self.left\_border\_condition

        right\_border\_condition = self.right\_border\_condition

        left\_a = self.left\_a

        left\_b = self.left\_b

        right\_a = self.right\_a

        right\_b = self.right\_b

        u = np.zeros((time\_steps, n))

        linspace = np.linspace(left\_border, right\_border, n)

        u[0] = u\_start(linspace)

        u[1] = u[0]+tau\*dt\_u\_start(linspace)

        if scheme == 'explicit':

            for k in range(1, time\_steps-1):

                for i in range(1, n-1):

                    u[k+1, i] = ((a\*\*2)\*(u[k, i-1]-2\*u[k, i]+u[k, i+1])/(h\*\*2)+ \

                                b\*(u[k, i+1]-u[k, i-1])/(2\*h)+ \

                                c\*u[k, i]+ \

                                f(left\_border+i\*h, k\*tau)-d\*(u[k-1, i]/(2\*tau))-(u[k-1, i]-2\*u[k, i])/(tau\*\*2)) \

                                / \

                                (1/(tau\*\*2)-d/(2\*tau))

                if boundary\_conditions\_interpolation == '2\_points\_1st\_order':

                    u[k+1, 0] = (left\_border\_condition((k+1)\*tau)-(left\_a/h)\*u[k+1, 1])/(-(left\_a/h)+left\_b)

                    u[k+1, -1] = (right\_border\_condition((k+1)\*tau)+(right\_a/h)\*u[k+1, -2])/ \

                                       ((right\_a/h)+right\_b)

                if boundary\_conditions\_interpolation == '3\_points\_2nd\_order':

                    u[k+1, 0] = (left\_border\_condition((k+1)\*tau)-((4\*left\_a)/(2\*h)\*u[k+1, 1])+(left\_a/(2\*h))\*u[k+1, 2])/ \

                                (((-3)\*left\_a)/(2\*h)+left\_b)

                    u[k+1, -1] = (right\_border\_condition((k+1)\*tau)+((4\*right\_a)/(2\*h)\*u[k+1, -2])-(right\_a/(2\*h))\*u[k+1, -3])/ \

                                       ((3\*right\_a)/(2\*h)+right\_b)

                if boundary\_conditions\_interpolation == '2\_points\_2nd\_order':

                    denominator = h-(b\*h\*\*2)/(2\*a\*\*2)

                    u[k+1, 0] = (left\_border\_condition((k+1)\*tau)-((left\_a\*h\*\*2)/(2\*a\*\*2))\*f(left\_border, (k+1)\*tau)-u[k+1, 1]\*left\_a/denominator-u[k, 0]\*((-left\_a\*d\*h\*\*2)/(2\*a\*\*2\*tau))/denominator-u[k-1, 0]\*((-left\_a\*h\*\*2)/(2\*a\*\*2\*tau\*\*2))/denominator)/(-left\_a/denominator-((left\_a\*h\*\*2)/(2\*a\*\*2\*tau\*\*2))/denominator+((left\_a\*c\*h\*\*2)/(2\*a))/denominator+((left\_a\*d\*h\*\*2)/(2\*a\*\*2\*tau))/denominator+left\_b)

                    denominator = -h-(b\*h\*\*2)/(2\*a\*\*2)

                    u[k+1, -1] = (right\_border\_condition((k+1)\*tau)-((right\_a\*h\*\*2)/(2\*a\*\*2))\*f(right\_border, (k+1)\*tau)-u[k+1, -2]\*right\_a/denominator-u[k, 0]\*((-right\_a\*d\*h\*\*2)/(2\*a\*\*2\*tau))/denominator-u[k-1, 0]\*((-right\_a\*h\*\*2)/(2\*a\*\*2\*tau\*\*2))/denominator)/(-right\_a/denominator-((right\_a\*h\*\*2)/(2\*a\*\*2\*tau\*\*2))/denominator+((right\_a\*c\*h\*\*2)/(2\*a))/denominator+((right\_a\*d\*h\*\*2)/(2\*a\*\*2\*tau))/denominator+right\_b)

        if scheme == 'implicit':

            alpha = (a\*\*2)/(h\*\*2)-(b)/(2\*h)

            beta = -1/(tau\*\*2)-(2\*a\*\*2)/(h\*\*2)+c+d/(2\*tau)

            gamma = (a\*\*2)/(h\*\*2)+(b)/(2\*h)

            denominator = h-(b\*h\*\*2)/(2\*a\*\*2)

            A = np.zeros((n, n))

            if boundary\_conditions\_interpolation == '2\_points\_1st\_order':

                A[0, 0] = (-(left\_a/h)+left\_b)

                A[0, 1] = (left\_a/h)

                A[-1, -1] = ((right\_a/h)+right\_b)

                A[-1, -2] = -(right\_a/h)

            if boundary\_conditions\_interpolation == '3\_points\_2nd\_order':

                A[0, 0] = (((-3)\*left\_a)/(2\*h)+left\_b)

                A[0, 1] = (4\*left\_a)/(2\*h)

                A[0, 2] = (-left\_a)/(2\*h)

                A[-1, -1] = ((3\*right\_a)/(2\*h)+right\_b)

                A[-1, -2] = ((-4)\*right\_a)/(2\*h)

                A[-1, -3] = (right\_a)/(2\*h)

            if boundary\_conditions\_interpolation == '2\_points\_2nd\_order':

                A[0, 0] = -left\_a/denominator-((left\_a\*h\*\*2)/(2\*a\*\*2\*tau\*\*2))/denominator+((left\_a\*c\*h\*\*2)/(2\*a))/denominator+((left\_a\*d\*h\*\*2)/(2\*a\*\*2\*tau))/denominator+left\_b

                A[0, 1] = left\_a/denominator

                A[-1, -1] = -right\_a/(-denominator)-((right\_a\*h\*\*2)/(2\*a\*\*2\*tau\*\*2))/(-denominator)+((right\_a\*c\*h\*\*2)/(2\*a))/(-denominator)+((right\_a\*d\*h\*\*2)/(2\*a\*\*2\*tau))/(-denominator)+right\_b

                A[-1, -2] = right\_a/(-denominator)

            for i in range(1, n-1):

                A[i, i-1] = alpha

                A[i, i] = beta

                A[i, i+1] = gamma

            for k in range(2, time\_steps):

                d\_vector = -np.array([f(left\_border+i\*h, k\*tau) for i in range(n)])

                d\_vector += (-2\*u[k-1]+u[k-2])/(tau\*\*2)+(d\*u[k-2])/(2\*tau)

                d\_vector[0] = left\_border\_condition(k\*tau)

                d\_vector[-1] = right\_border\_condition(k\*tau)

                if boundary\_conditions\_interpolation == '2\_points\_2nd\_order':

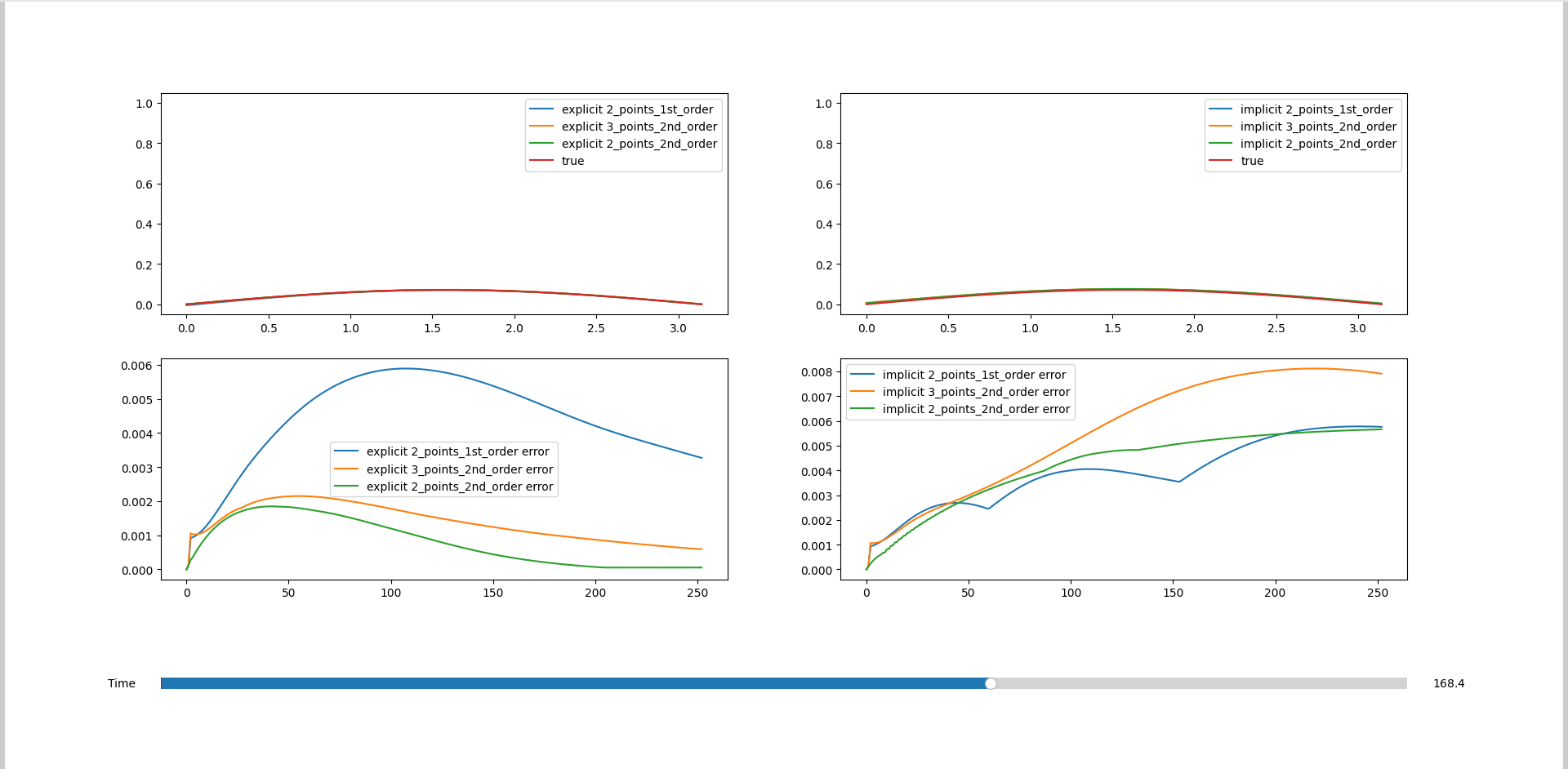
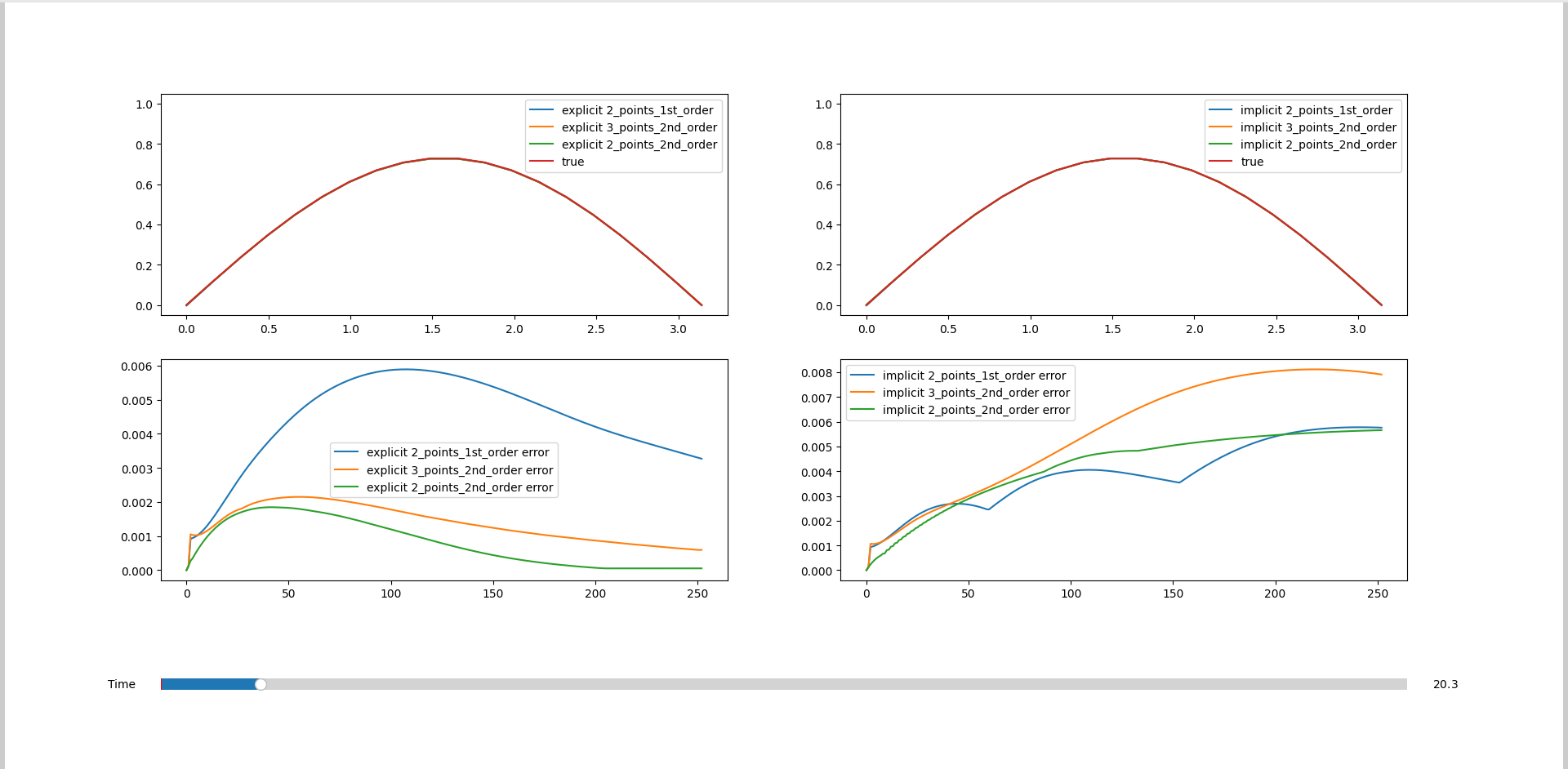
                    d\_vector[0] -= ((left\_a\*h\*\*2)/(2\*a\*\*2))\*f(left\_border, k\*tau)+u[k-1, 0]\*((-left\_a\*d\*h\*\*2)/(2\*a\*\*2\*tau))/denominator-u[k-2, 0]\*((-left\_a\*h\*\*2)/(2\*a\*\*2\*tau\*\*2))/denominator

                    d\_vector[-1] -= ((right\_a\*h\*\*2)/(2\*a\*\*2))\*f(right\_border, k\*tau)+u[k-1, -1]\*((-right\_a\*d\*h\*\*2)/(2\*a\*\*2\*tau))/(-denominator)-u[k-2, -1]\*((-right\_a\*h\*\*2)/(2\*a\*\*2\*tau\*\*2))/(-denominator)

                u[k] = np.linalg.solve(A, d\_vector)

        return u

Результат работы



**5.3 Эллиптические одномерные уравнения**

**Задача**

Решить краевую задачу для дифференциального уравнения эллиптического типа. Аппроксимацию уравнения произвести с использованием центрально-разностной схемы. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением . Исследовать зависимость погрешности от сеточных параметров .

**Вариант 3**

,



,

.

Аналитическое решение: .

Исходный код

import math

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.widgets import Slider, Button

class Solver:

    def \_\_init\_\_(self, ax, ay, bx, by, c,

    left\_border\_condition, left\_a, left\_b,

    right\_border\_condition, right\_a, right\_b,

    bottom\_border\_condition, bottom\_a, bottom\_b,

    top\_border\_condition, top\_a, top\_b,

    left\_border, right\_border, bottom\_border, top\_border,

    nx, ny) -> None:

        self.ax = ax

        self.ay = ay

        self.bx = bx

        self.by = by

        self.c = c

        self.left\_border = left\_border

        self.right\_border = right\_border

        self.top\_border = top\_border

        self.bottom\_border = bottom\_border

        self.lx = right\_border-left\_border

        self.ly = top\_border-bottom\_border

        self.nx = nx

        self.ny = ny

        self.hx = self.lx/(nx-1)

        self.hy = self.ly/(ny-1)

        self.left\_border\_condition = left\_border\_condition

        self.right\_border\_condition = right\_border\_condition

        self.top\_border\_condition = top\_border\_condition

        self.bottom\_border\_condition = bottom\_border\_condition

        self.left\_a = left\_a

        self.left\_b = left\_b

        self.right\_a = right\_a

        self.right\_b = right\_b

        self.top\_a = top\_a

        self.top\_b = top\_b

        self.bottom\_a = bottom\_a

        self.bottom\_b = bottom\_b

    def solve\_libman(self, e):

        left\_border = self.left\_border

        bottom\_border = self.bottom\_border

        hx = self.hx

        hy = self.hy

        nx = self.nx

        ny = self.ny

        left\_border\_condition = self.left\_border\_condition

        right\_border\_condition = self.right\_border\_condition

        bottom\_border\_condition = self.bottom\_border\_condition

        top\_border\_condition = self.top\_border\_condition

        left\_a = self.left\_a

        left\_b = self.left\_b

        right\_a = self.right\_a

        right\_b = self.right\_b

        top\_a = self.top\_a

        top\_b = self.top\_b

        bottom\_a = self.bottom\_a

        bottom\_b = self.bottom\_b

        hist = np.zeros((nx, ny, 0))

        u = np.zeros((nx, ny, 1))

        next\_u = np.empty((nx, ny, 1))

        cur\_e = np.Infinity

        while True:

            cur\_e = -np.Infinity

            for x in range(1,nx-1):

                for y in range(1, ny-1):

                    next\_u[x,y,0] = ((u[x-1,y,0]+u[x+1,y,0])/(hx\*hx)+(u[x,y-1,0]+u[x,y+1,0])/(hy\*hy))/(2\*(1.0/(hx\*hx)+1.0/(hy\*hy)))

            for x in range(1, nx-1):

                next\_u[x,0,0] = (bottom\_border\_condition(left\_border+x\*hx)-(bottom\_a/hy)\*next\_u[x,1,0])/(bottom\_b-(bottom\_a/hy))

                next\_u[x,-1,0] = (top\_border\_condition(left\_border+x\*hx)+(top\_a/hy)\*next\_u[x,-2,0])/(top\_b+(top\_a/hy))

            for y in range(1, ny-1):

                next\_u[0,y,0] = (left\_border\_condition(bottom\_border+y\*hy)-(left\_a/hx)\*next\_u[1,y,0])/(left\_b-(left\_a/hx))

                next\_u[-1,y,0] = (right\_border\_condition(bottom\_border+y\*hy)+(right\_a/hx)\*next\_u[-2,y,0])/(right\_b+(right\_a/hx))

            for x in range(1,nx-1):

                for y in range(1, ny-1):

                    cur\_e = max(cur\_e, np.abs(next\_u[x,y,0]-u[x,y,0]))

            u, next\_u = next\_u, u

            hist = np.append(hist, u, 2)

            if not cur\_e > e and cur\_e != 0.0:

                break

        return hist

    def solve\_seidel(self, e):

        left\_border = self.left\_border

        bottom\_border = self.bottom\_border

        hx = self.hx

        hy = self.hy

        nx = self.nx

        ny = self.ny

        left\_border\_condition = self.left\_border\_condition

        right\_border\_condition = self.right\_border\_condition

        bottom\_border\_condition = self.bottom\_border\_condition

        top\_border\_condition = self.top\_border\_condition

        left\_a = self.left\_a

        left\_b = self.left\_b

        right\_a = self.right\_a

        right\_b = self.right\_b

        top\_a = self.top\_a

        top\_b = self.top\_b

        bottom\_a = self.bottom\_a

        bottom\_b = self.bottom\_b

        hist = np.zeros((nx, ny, 0))

        u = np.zeros((nx, ny, 1))

        cur\_e = np.Infinity

        while True:

            cur\_e = -np.Infinity

            for x in range(1,nx-1):

                for y in range(1, ny-1):

                    cur\_e = max(cur\_e, abs(u[x,y]-((u[x-1,y,0]+u[x+1,y,0])/(hx\*hx)+(u[x,y-1,0]+u[x,y+1,0])/(hy\*hy))/(2\*(1.0/(hx\*hx)+1.0/(hy\*hy)))))

                    u[x,y,0] = ((u[x-1,y,0]+u[x+1,y,0])/(hx\*hx)+(u[x,y-1,0]+u[x,y+1,0])/(hy\*hy))/(2\*(1.0/(hx\*hx)+1.0/(hy\*hy)))

            for x in range(1, nx-1):

                u[x,0,0] = (bottom\_border\_condition(left\_border+x\*hx)-(bottom\_a/hy)\*u[x,1,0])/(bottom\_b-(bottom\_a/hy))

                u[x,-1,0] = (top\_border\_condition(left\_border+x\*hx)+(top\_a/hy)\*u[x,-2,0])/(top\_b+(top\_a/hy))

            for y in range(1, ny-1):

                u[0,y,0] = (left\_border\_condition(bottom\_border+y\*hy)-(left\_a/hx)\*u[1,y,0])/(left\_b-(left\_a/hx))

                u[-1,y,0] = (right\_border\_condition(bottom\_border+y\*hy)+(right\_a/hx)\*u[-2,y,0])/(right\_b+(right\_a/hx))

            hist = np.append(hist, u, 2)

            print(cur\_e)

            if not cur\_e > e and cur\_e != 0.0:

                break

        return hist

    def solve\_libman\_relaxed(self, e, w = 1):

        left\_border = self.left\_border

        bottom\_border = self.bottom\_border

        hx = self.hx

        hy = self.hy

        nx = self.nx

        ny = self.ny

        left\_border\_condition = self.left\_border\_condition

        right\_border\_condition = self.right\_border\_condition

        bottom\_border\_condition = self.bottom\_border\_condition

        top\_border\_condition = self.top\_border\_condition

        left\_a = self.left\_a

        left\_b = self.left\_b

        right\_a = self.right\_a

        right\_b = self.right\_b

        top\_a = self.top\_a

        top\_b = self.top\_b

        bottom\_a = self.bottom\_a

        bottom\_b = self.bottom\_b

        hist = np.zeros((nx, ny, 0))

        u = np.zeros((nx, ny, 1))

        next\_u = np.empty((nx, ny, 1))

        cur\_e = np.Infinity

        while True:

            cur\_e = -np.Infinity

            for x in range(1,nx-1):

                for y in range(1, ny-1):

                    next = ((u[x-1,y,0]+u[x+1,y,0])/(hx\*hx)+(u[x,y-1,0]+u[x,y+1,0])/(hy\*hy))/(2\*(1.0/(hx\*hx)+1.0/(hy\*hy)))

                    next\_u[x,y,0] = next+w\*(next-u[x,y])

            for x in range(1, nx-1):

                next\_u[x,0,0] = (bottom\_border\_condition(left\_border+x\*hx)-(bottom\_a/hy)\*next\_u[x,1,0])/(bottom\_b-(bottom\_a/hy))

                next\_u[x,-1,0] = (top\_border\_condition(left\_border+x\*hx)+(top\_a/hy)\*next\_u[x,-2,0])/(top\_b+(top\_a/hy))

            for y in range(1, ny-1):

                next\_u[0,y,0] = (left\_border\_condition(bottom\_border+y\*hy)-(left\_a/hx)\*next\_u[1,y,0])/(left\_b-(left\_a/hx))

                next\_u[-1,y,0] = (right\_border\_condition(bottom\_border+y\*hy)+(right\_a/hx)\*next\_u[-2,y,0])/(right\_b+(right\_a/hx))

            for x in range(1,nx-1):

                for y in range(1, ny-1):

                    cur\_e = max(cur\_e, np.abs(next\_u[x,y,0]-u[x,y,0]))

            u, next\_u = next\_u, u

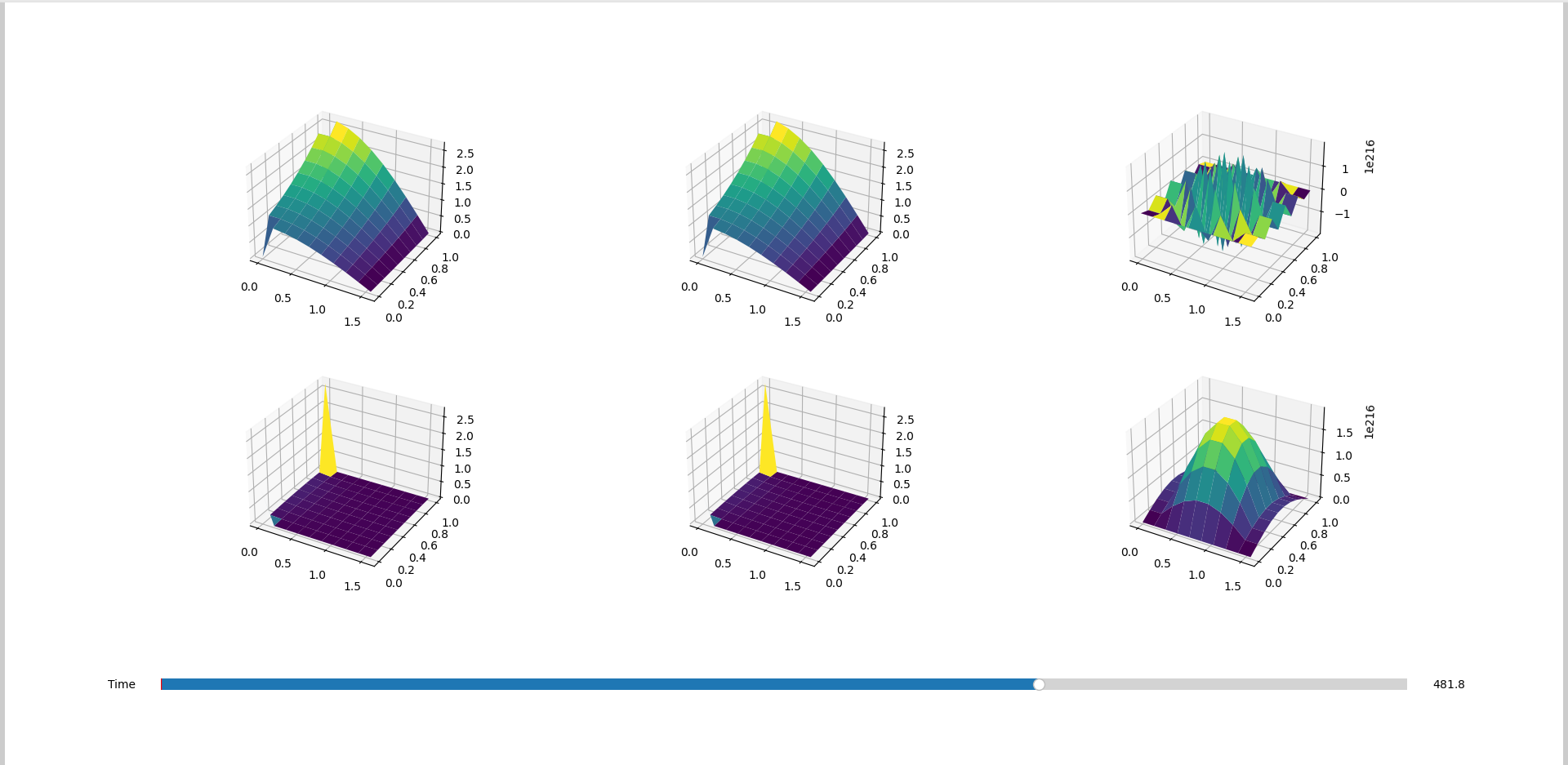
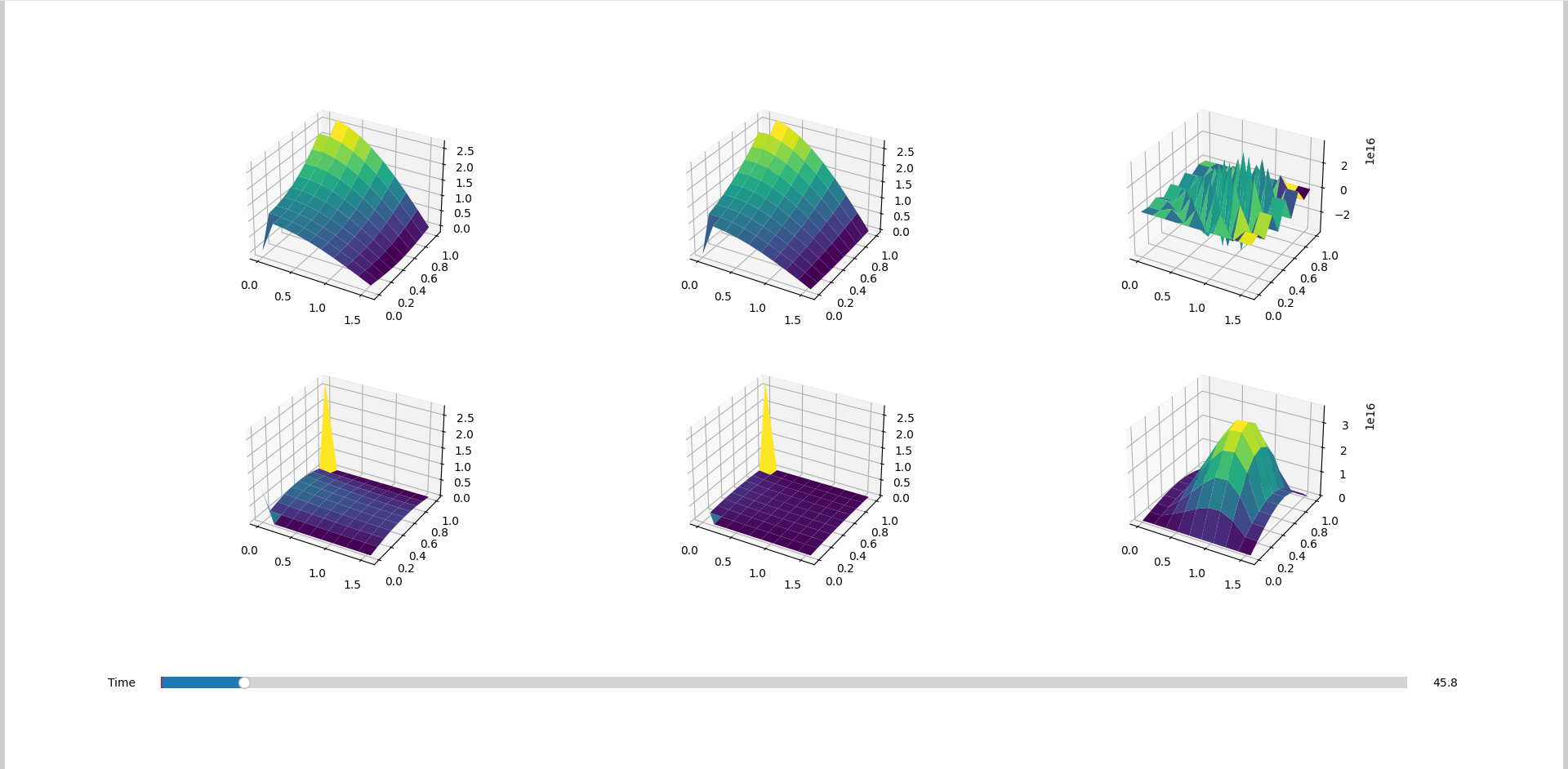
            hist = np.append(hist, u, 2)

            if not cur\_e > e and cur\_e != 0.0:

                break

        return hist

Результат работы



**5.4 Параболические двумерные уравнения**

**Задача**

Используя схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением . Исследовать зависимость погрешности от сеточных параметров .

**Вариант 4**

,



,

.

Аналитическое решение: .

Исходный код

import math

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.widgets import Slider, Button

class Solver:

    def \_\_init\_\_(self, ax, ay, bx, by, c,

    left\_border\_condition, left\_a, left\_b,

    right\_border\_condition, right\_a, right\_b,

    bottom\_border\_condition, bottom\_a, bottom\_b,

    top\_border\_condition, top\_a, top\_b,

    left\_border, right\_border, bottom\_border, top\_border,

    nx, ny,

    end\_time,

    time\_steps,

    u\_start) -> None:

        self.ax = ax

        self.ay = ay

        self.bx = bx

        self.by = by

        self.c = c

        self.left\_border = left\_border

        self.right\_border = right\_border

        self.top\_border = top\_border

        self.bottom\_border = bottom\_border

        self.lx = right\_border-left\_border

        self.ly = top\_border-bottom\_border

        self.nx = nx

        self.ny = ny

        self.hx = self.lx/(nx-1)

        self.hy = self.ly/(ny-1)

        self.end\_time = end\_time

        self.time\_steps = time\_steps

        self.tau = end\_time/time\_steps

        self.left\_border\_condition = left\_border\_condition

        self.right\_border\_condition = right\_border\_condition

        self.top\_border\_condition = top\_border\_condition

        self.bottom\_border\_condition = bottom\_border\_condition

        self.left\_a = left\_a

        self.left\_b = left\_b

        self.right\_a = right\_a

        self.right\_b = right\_b

        self.top\_a = top\_a

        self.top\_b = top\_b

        self.bottom\_a = bottom\_a

        self.bottom\_b = bottom\_b

        self.u\_start = u\_start

    def solve\_variable\_direction\_method(self):

        ax = self.ax

        ay = self.ay

        bx = self.bx

        by = self.by

        c = self.c

        left\_border = self.left\_border

        right\_border = self.right\_border

        top\_border = self.top\_border

        bottom\_border = self.bottom\_border

        lx = self.right\_border-left\_border

        ly = self.top\_border-bottom\_border

        nx = self.nx

        ny = self.ny

        hx = self.hx

        hy = self.hy

        end\_time = self.end\_time

        time\_steps = self.time\_steps

        tau = self.tau

        left\_border\_condition = self.left\_border\_condition

        right\_border\_condition = self.right\_border\_condition

        top\_border\_condition = self.top\_border\_condition

        bottom\_border\_condition = self.bottom\_border\_condition

        left\_a = self.left\_a

        left\_b = self.left\_b

        right\_a = self.right\_a

        right\_b = self.right\_b

        top\_a = self.top\_a

        top\_b = self.top\_b

        bottom\_a = self.bottom\_a

        bottom\_b = self.bottom\_b

        u\_start = self.u\_start

        hist = np.zeros((ny, nx, 0))

        start = np.empty((ny, nx, 1))

        for y in range(ny):

            for x in range(nx):

                start[y, x] = u\_start(left\_border+hx\*x, bottom\_border+hy\*y)

        hist = np.append(hist, start, 2)

        for k in range(0, time\_steps):

            half\_u = np.zeros((ny, nx, 1))

            next\_u = np.zeros((ny, nx, 1))

            for y in range(1, ny-1):

                A = np.zeros((nx, nx))

                d = np.empty(nx)

                A[0, 0] = (-(left\_a/hx)+left\_b)

                A[0, 1] = (left\_a/hx)

                A[-1, -1] = ((right\_a/hx)+right\_b)

                A[-1, -2] = -(right\_a/hx)

                d[0] = left\_border\_condition(bottom\_border+hy\*y, tau\*k+(tau/2))

                d[-1] = right\_border\_condition(bottom\_border+hy\*y, tau\*k+(tau/2))

                for x in range(1, nx-1):

                    A[x, x-1] = -(ax/(hx\*\*2))+bx/(2\*hx)

                    A[x, x] = (2/tau)+((2\*ax)/(hx\*\*2))-c

                    A[x, x+1] = -(ax/(hx\*\*2))-bx/(2\*hx)

                    d[x] = (1/(tau/2))\*hist[y, x, -1]+(ay/hy\*\*2)\*(hist[y+1, x, -1]-2\*hist[y, x, -1]+hist[y-1, x, -1])+(by/(2\*hy))\*(hist[y+1, x, -1]-hist[y-1, x, -1])

                solution = np.linalg.solve(A, d)

                for x in range(nx):

                    half\_u[y, x, 0] = solution[x]

            for x in range(nx):

                half\_u[0, x, -1] = (bottom\_border\_condition(left\_border+hx\*x, tau\*k+(tau/2))-(bottom\_a/hy)\*half\_u[1, x, -1])/((-bottom\_a/hy)+bottom\_b)

                half\_u[-1, x, -1] = (top\_border\_condition(left\_border+hx\*x, tau\*k+(tau/2))+(top\_a/hy)\*half\_u[-2, x, -1])/((top\_a/hy)+top\_b)

            for x in range(1, nx-1):

                A = np.zeros((ny, ny))

                d = np.empty(ny)

                A[0, 0] = (-(bottom\_a/hy)+bottom\_b)

                A[0, 1] = (bottom\_a/hy)

                A[-1, -1] = ((top\_a/hy)+top\_b)

                A[-1, -2] = -(top\_a/hy)

                d[0] = bottom\_border\_condition(left\_border+hx\*x, tau\*k+(tau/2))

                d[-1] = top\_border\_condition(left\_border+hx\*x, tau\*k+(tau/2))

                for y in range(1, ny-1):

                    A[y, y-1] = -(ay/(hy\*\*2))+by/(2\*hy)

                    A[y, y] = (2/tau)+((2\*ay)/(hy\*\*2))-c

                    A[y, y+1] = -(ay/(hy\*\*2))-by/(2\*hy)

                    d[y] = (1/(tau/2))\*half\_u[y, x, -1]+(ax/hx\*\*2)\*(half\_u[y, x+1, -1]-2\*half\_u[y, x, -1]+half\_u[y, x-1, -1])+(bx/(2\*hx))\*(half\_u[y, x+1, -1]-half\_u[y, x-1, -1])

                solution = np.linalg.solve(A, d)

                for y in range(ny):

                    next\_u[y, x, 0] = solution[y]

            for y in range(ny):

                next\_u[y, 0, -1] = (left\_border\_condition(bottom\_border+hy\*y, tau\*k+(tau/2))-(left\_a/hx)\*next\_u[y, 1, -1])/((-left\_a/hx)+left\_b)

                next\_u[y, -1, -1] = (right\_border\_condition(bottom\_border+hy\*y, tau\*k+(tau/2))+(right\_a/hx)\*next\_u[y, -2, -1])/((right\_a/hx)+right\_b)

            hist = np.append(hist, next\_u, 2)

        return hist

    def solve\_fractional\_step\_method(self):

        ax = self.ax

        ay = self.ay

        bx = self.bx

        by = self.by

        c = self.c

        left\_border = self.left\_border

        right\_border = self.right\_border

        top\_border = self.top\_border

        bottom\_border = self.bottom\_border

        lx = self.right\_border-left\_border

        ly = self.top\_border-bottom\_border

        nx = self.nx

        ny = self.ny

        hx = self.hx

        hy = self.hy

        end\_time = self.end\_time

        time\_steps = self.time\_steps

        tau = self.tau

        left\_border\_condition = self.left\_border\_condition

        right\_border\_condition = self.right\_border\_condition

        top\_border\_condition = self.top\_border\_condition

        bottom\_border\_condition = self.bottom\_border\_condition

        left\_a = self.left\_a

        left\_b = self.left\_b

        right\_a = self.right\_a

        right\_b = self.right\_b

        top\_a = self.top\_a

        top\_b = self.top\_b

        bottom\_a = self.bottom\_a

        bottom\_b = self.bottom\_b

        u\_start = self.u\_start

        hist = np.zeros((ny, nx, 0))

        start = np.empty((ny, nx, 1))

        for y in range(ny):

            for x in range(nx):

                start[y, x] = u\_start(left\_border+hx\*x, bottom\_border+hy\*y)

        hist = np.append(hist, start, 2)

        for k in range(1, time\_steps+1):

            half\_u = np.zeros((ny, nx, 1))

            next\_u = np.zeros((ny, nx, 1))

            for y in range(1, ny-1):

                A = np.zeros((nx, nx))

                d = np.empty(nx)

                A[0, 0] = (-(left\_a/hx)+left\_b)

                A[0, 1] = (left\_a/hx)

                A[-1, -1] = ((right\_a/hx)+right\_b)

                A[-1, -2] = -(right\_a/hx)

                d[0] = left\_border\_condition(bottom\_border+hy\*y, tau\*(k+0.5))

                d[-1] = right\_border\_condition(bottom\_border+hy\*y, tau\*(k+0.5))

                for x in range(1, nx-1):

                    A[x, x-1] = (ax/(hx\*\*2))-bx/(2\*hx)

                    A[x, x] = (-1/tau)-(2\*ax)/(hx\*\*2)+c

                    A[x, x+1] = (ax/(hx\*\*2))+bx/(2\*hx)

                    d[x] = (-1/tau)\*hist[y, x, -1]

                solution = np.linalg.solve(A, d)

                for x in range(nx):

                    half\_u[y, x, 0] = solution[x]

            for x in range(nx):

                half\_u[0, x] = (bottom\_border\_condition(left\_border+hx\*x, tau\*(k+0.5))-(bottom\_a/hy)\*half\_u[1, x])/((-bottom\_a/hy)+bottom\_b)

                half\_u[-1, x] = (top\_border\_condition(left\_border+hx\*x, tau\*(k+0.5))+(top\_a/hy)\*half\_u[-2, x])/((top\_a/hy)+top\_b)

            for x in range(1, nx-1):

                A = np.zeros((ny, ny))

                d = np.empty(ny)

                A[0, 0] = (-(bottom\_a/hy)+bottom\_b)

                A[0, 1] = (bottom\_a/hy)

                A[-1, -1] = ((top\_a/hy)+top\_b)

                A[-1, -2] = -(top\_a/hy)

                d[0] = bottom\_border\_condition(left\_border+hx\*x, tau\*(k))

                d[-1] = top\_border\_condition(left\_border+hx\*x, tau\*(k))

                for y in range(1, ny-1):

                    A[y, y-1] = (ay/(hy\*\*2))-by/(2\*hy)

                    A[y, y] = (-1/tau)-(2\*ay)/(hy\*\*2)+c

                    A[y, y+1] = (ay/(hy\*\*2))+by/(2\*hy)

                    d[y] = (-1/tau)\*half\_u[y, x, -1]

                solution = np.linalg.solve(A, d)

                for y in range(ny):

                    next\_u[y, x, 0] = solution[y]

            for y in range(ny):

                next\_u[y, 0] = (left\_border\_condition(bottom\_border+hy\*y, tau\*(k))-(left\_a/hx)\*next\_u[y, 1])/((-left\_a/hx)+left\_b)

                next\_u[y, -1] = (right\_border\_condition(bottom\_border+hy\*y, tau\*(k))+(right\_a/hx)\*next\_u[y, -2])/((right\_a/hx)+right\_b)

            hist = np.append(hist, next\_u, 2)

        return hist

Результат работы

