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 $U(x,t)$. Исследовать зависимость погрешности от сеточных параметров τ, h.

Вариант 10

$\dfrac{\partial u}{\partial t} = a\dfrac{\partial^2 u}{\partial x^2} + b\dfrac{\partial u}{\partial x} + cu$, $a > 0$, $b > 0$, $c < 0$.

$u_x(0,t) + u(0,t) = \exp((c-a)t)(\cos(bt) + \sin(bt))$,

$u_x(\pi,t) + u(\pi,t) = -\exp((c-a)t)(\cos(bt) + \sin(bt))$,

$u(x,0) = \sin x$.

Аналитическое решение: $U(x,t) = \exp((c-a)t)\sin(x+bt)$.

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)
```

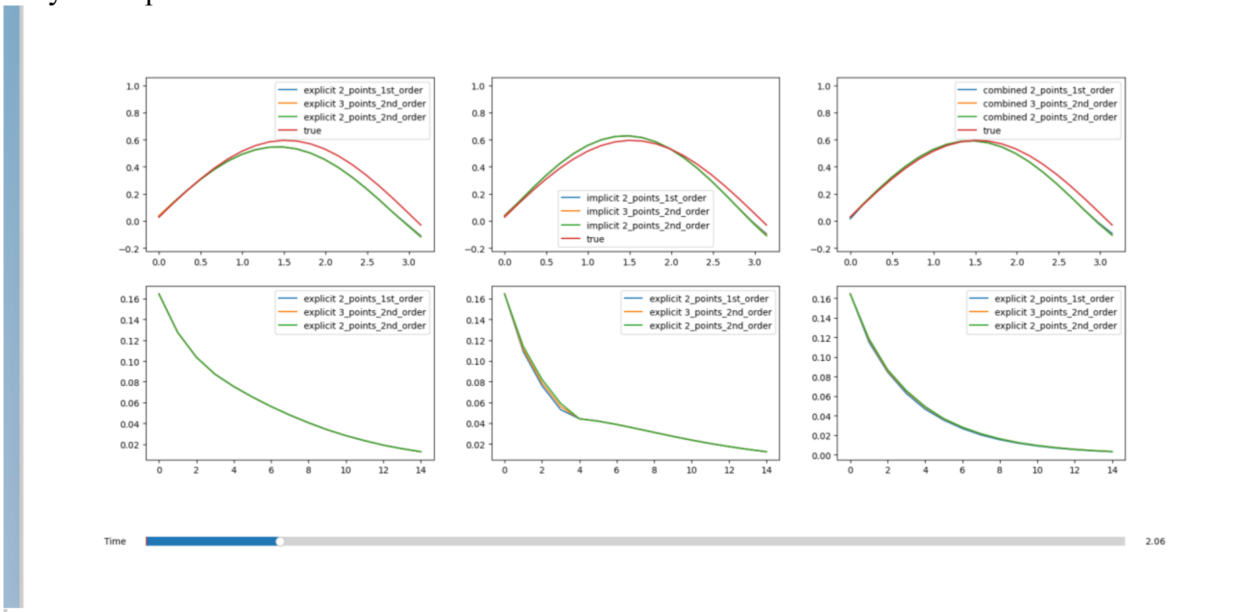
```
                    u[k+1, -1] = (self.right_border_condition((k+1)*self.tau, 
self.a, self.b, self.c)+(self.right_a/self.h)*u[k+1, -2])/ \
                                       ((self.right_a/self.h)+self.right_b)
                if boundary_conditions_interpolation == '3_points_2nd_order':
                    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/(s
elf.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)
```

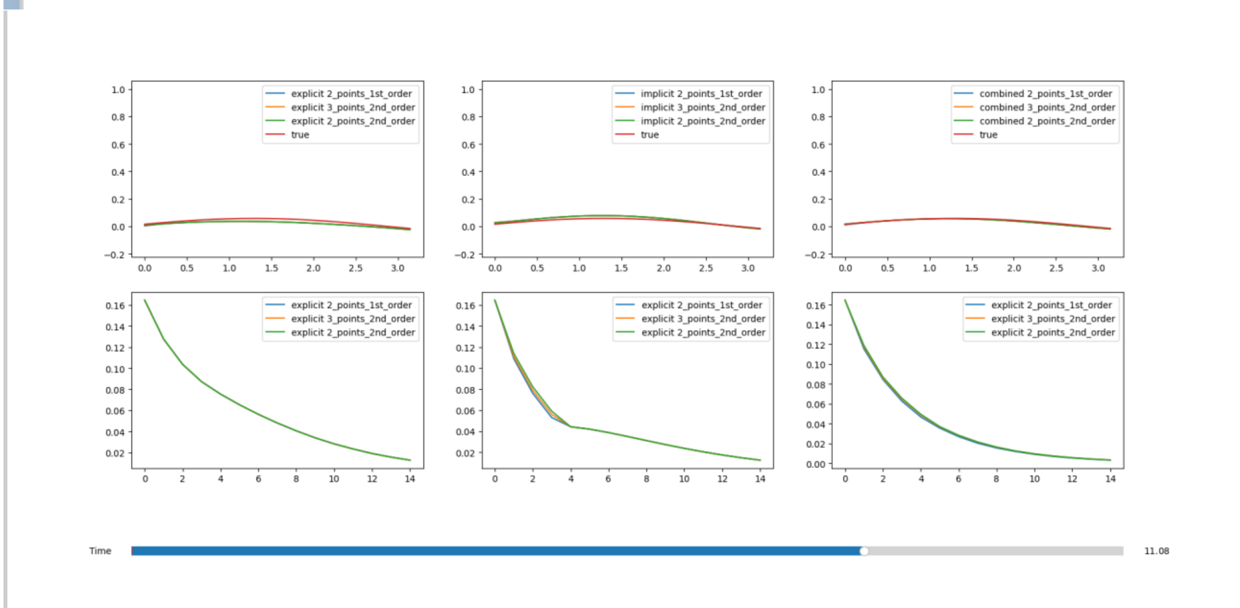
```
                A[-1, -3] = (self.right_a)/(2*self.h)
            if boundary_conditions_interpolation == '2_points_2nd_order':
                A[0, 0] = (self.left_a/(self.h-(self.b*self.h**2)/(2*self.a)))*(-
1-(self.h**2)/(2*self.a*self.tau)+(self.c*self.h**2)/(2*self.a))+self.left_b
                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_p
art[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/(s
elf.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 $U(x,t)$. Исследовать зависимость погрешности от сеточных параметров τ, h.

Вариант 10

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

$$u_x(0,t) = \exp(-t),$$

$$u_x(\pi,t) = -\exp(-t),$$

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

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

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

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)
```

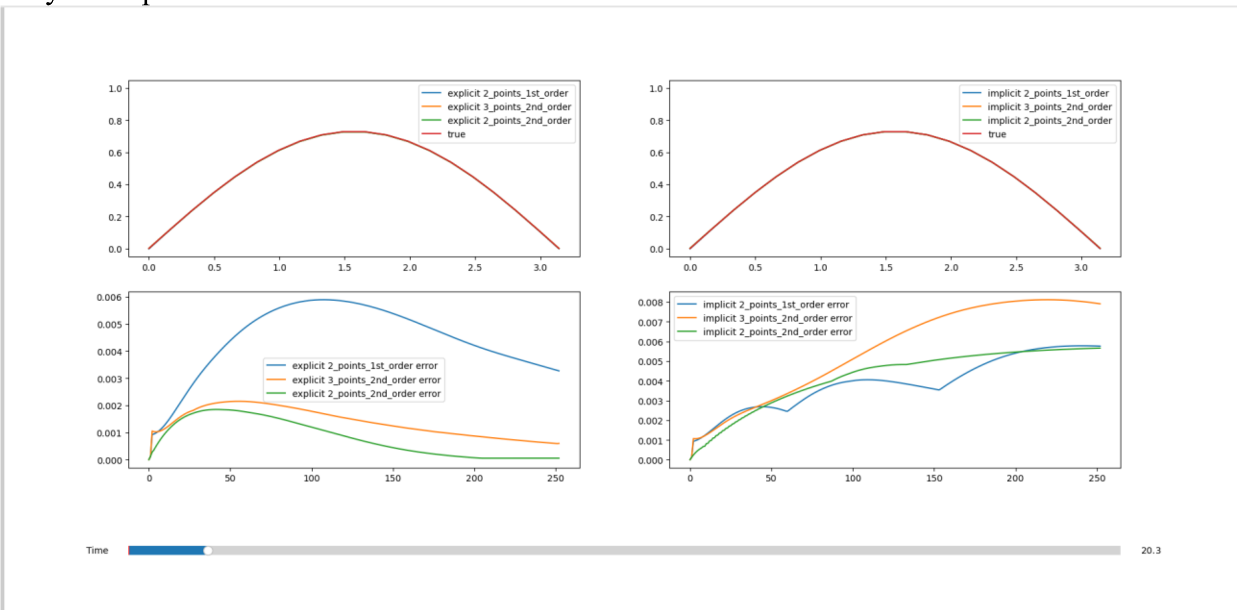
```
        self.time_steps = int((end_time*a**2*n)/(sigma*self.l))-1
        self.tau = (sigma*self.l)/(a**2*n)

        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':
```

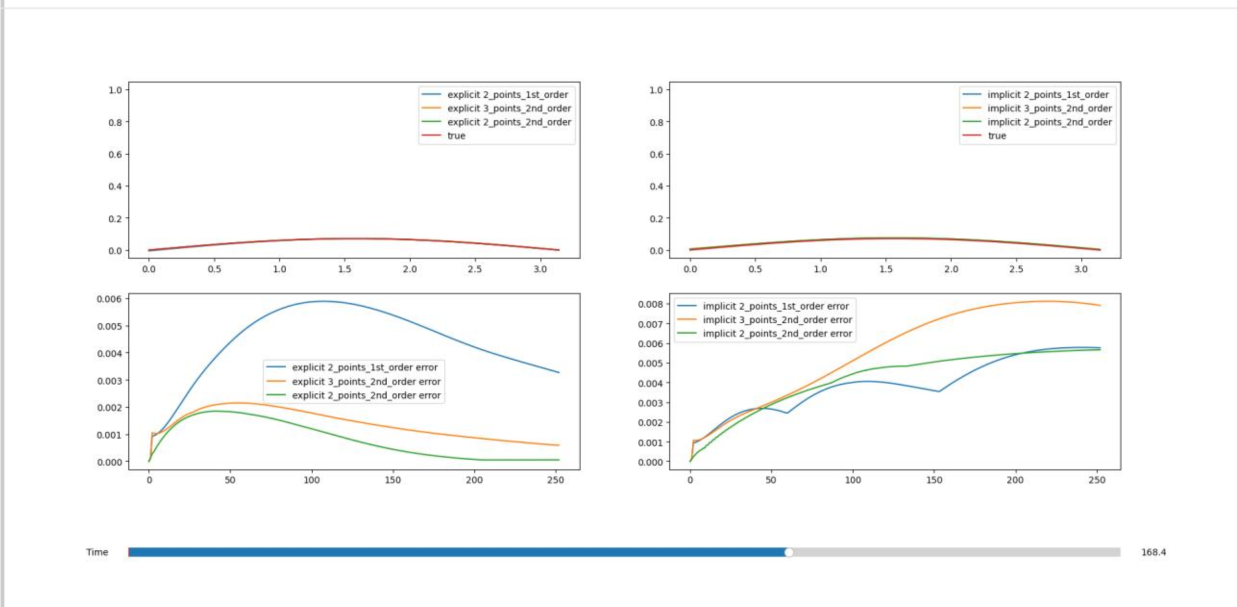
```
            for k in range(1, time_steps-1):
                for i in range(1, n-1):
                    u[k+1, i] = ((a**2)*(u[k, i-1]-2*u[k, i]+u[k, i+1])/(h**2)+ \
                                b*(u[k, i+1]-u[k, i-1])/(2*h)+ \
                                c*u[k, i]+ \
                                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 $U(x, y)$. Исследовать зависимость погрешности от сеточных параметров h_x, h_y.

Вариант 3

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0,$$

$$u(0, y) = \cos y,$$

$$u(1, y) = e \cos y,$$

$$u_y(x,0) = 0,$$

$$u_y(x, \frac{\pi}{2}) = -\exp(x).$$

Аналитическое решение: $U(x, y) = \exp(x) \cos y$.

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
```

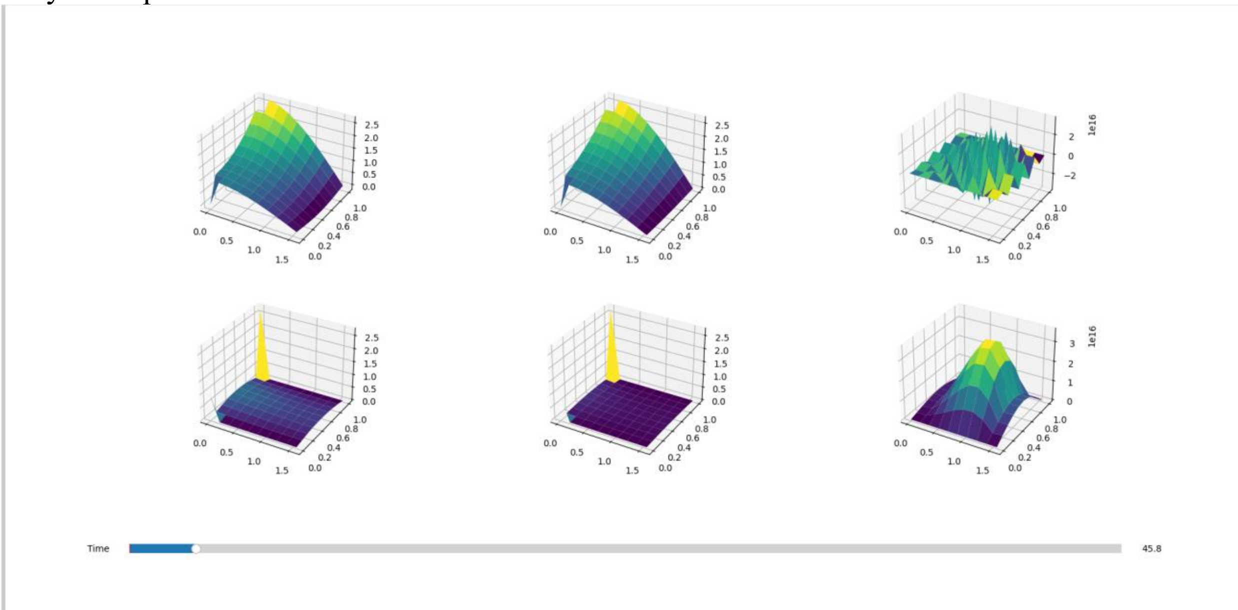
```
        right_a = self.right_a
        right_b = self.right_b

        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
```

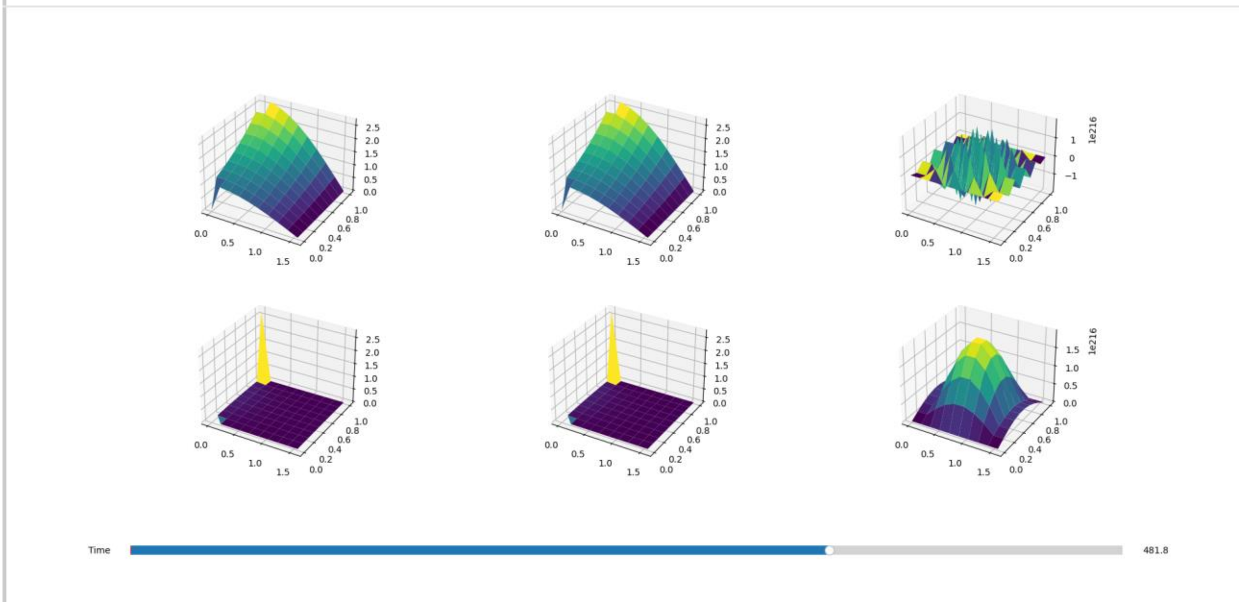
```
        hy = self.hy

        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 $U(x,t)$. Исследовать зависимость погрешности от сеточных параметров τ, h_x, h_y.

Вариант 4

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0,$$

$u_x(0, y) = \exp(y),$

$u_x(\pi, y) = -\exp(y),$

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

$u(x,1) = e \sin x.$

Аналитическое решение: $U(x, y) = \sin x \exp(y)$.

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
```

```python
        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