Московский авиационный институт

(Национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторные работы

по курсу «Численные методы»

Вариант 10

Выполнил: Черных С. Д.

Группа: М8О-405Б-20

Проверил: доц. Иванов И. Э.

Дата:

Оценка:

Москва, 2023

Лабораторная работа №8

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,y,t). Исследовать зависимость от сеточных параметров.

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial y^2} + \sin(x)\sin(y)\,[\cos(\mu t) + (a+b)\sin(\mu t)]$$

$u(0, y, t) = 0$

$u_x(\pi, y, t) = -\sin(y)\sin(\mu t)$

$u(x, 0, t) = 0$

$u_y(x, \pi, t) = -\sin(x)\sin(\mu t)$

$u(x, y, 0) = 0$

в) $a = 1, b = 2, \mu = 1$

Аналитическое решение: $U(x, y, t) = \sin(x)\sin(y)\sin(\mu t)$

Теоретические сведения

Метод переменных направлений

Разобьём шаг по времени на два шага. На первом дробном временном слое неявно аппроксимируем дифференциальный оператор $\frac{\partial^2}{x^2}$, а на следующем слое аппроксимируем уже явно дифференциальный оператор $\frac{\partial^2}{y^2}$. Получаем следующую схему:

$$\frac{u_{i,j}^{k+\frac{1}{2}} - u_{i,j}^{k}}{\frac{\tau}{2}} = \frac{a}{h_1^2}\left(u_{i+1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}}\right) + \frac{b}{h_2^2}\left(u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}\right) + f_{i,j}^{k+\frac{1}{2}}$$

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k+\frac{1}{2}}}{\frac{\tau}{2}} = \frac{a}{h_1^2}\left(u_{i+1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}}\right) + \frac{b}{h_2^2}\left(u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1}\right) + f_{i,j}^{k+\frac{1}{2}}$$

На первом дробном шаге с помощью $J-1$ скалярных прогонок в направлении x получаем распределение сеточной функции $u_{ij}^{k+\frac{1}{2}}, i = 1, \ldots, I-1, j = 1, \ldots, J-1$ на первом временном полуслое. На втором шаге с помощью $I-1$ скалярных прогонок в направлении у получаем распределение сеточной функции $u_{ij}^{k+1}, i = 1, \ldots, I-1, j = 1, \ldots, J-1$ на втором временном полуслое.

На двух границах области заданы условия первого рода, на них сеточная функция определяется однозначно для любого момента времени. На двух "правых" границах заданы граничные условия второго рода, аппроксимируем дифференциальные операторы следующим образом:

$$u^k{}_{0j} = \phi_2(y_J, t^k)2h_x + u_{0j}^{k-1} \qquad u^k{}_{i0} = \phi_4(x_I, t^k)2h_y + u_{i0}^{k-1}$$

Сеточная функция на нулевом слое определяется начальным условием.

Схема имеет второй порядок точности по времени и является абсолютно устойчивой (двумерный случай).

Метод дробных шагов

Этот метод использует только неявные конечно разностные-операторы. Разобьём шаг по времени на два шага . На первом дробном шаге проводится аппроксимация только одного из пространственных дифференциальных операторов, например $\frac{\partial^2}{x^2}$. Тогда на следующем дробном шаге аппроксимируется $\frac{\partial^2}{y^2}$. Получаем следующую схему:

$$\frac{u_{i,j}^{k+\frac{1}{2}} - u_{i,j}^{k}}{\tau} = \frac{a}{h_1^2}\left(u_{i+1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}}\right) + \frac{f_{i,j}^{k}}{2}$$

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k+\frac{1}{2}}}{\tau} = \frac{b}{h_2^2}\left(u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1}\right) + \frac{f_{i,j}^{k+1}}{2}$$

Такая схема имеет первый порядок точности по времени и является абсолютно устойчивой.

Код программы

Программа выполнена на языке python, в среде разработки Jupyter Notebook. Код программы будет в виде незапущенного кода в Jupyter Notebook, вывод программы, сокращён до графиков, построенных в среде Jupyter Notebook.

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.

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial y^2} + \sin x \sin y[\cos\mu t + (a+b)\sin\mu t]$$

$$ u(0, y, t) = 0 \ u_x(\pi, y, t) = -\sin{y} \sin{\mu t} \ u(x, 0, t) = 0 \ u_y(x, \pi, t) = -\sin{x} \sin{\mu t} \ u(x, y, 0) = 0

В) $a = 1, b = 2, \mu = 1$

аналитическое решение

$$U(x,y,t) = \sin x \sin y \sin\mu t$$

```
import math
import typing
from typing import List
import matplotlib.pyplot as plt
import copy
import plotly.graph_objects as go
from functools import lru_cache
```

Входные условия

```python
x_left = 0
x_right = math.pi
y_left = 0
y_right = math.pi

a = 1
b = 2
mu = 1

def phi_1(y:float, t:float, mu:float=1) -> float:
    return 0

def phi_2(y:float, t:float, mu:float=1) -> float:
    return -math.sin(y) * math.sin(mu * t)


def phi_3(x:float, t:float, mu:float=1) -> float:
    return 0

def phi_4(x:float, t:float, mu:float=1) -> float:
    return -math.sin(x) * math.sin(mu * t)

def psi(x:float, y:float, mu:float=1) -> float:
    return 0

def f(x:float, y:float, t:float, mu:float=1, a:float=1, b:float=2) -> float:
    return math.sin(x) * math.sin(y) * (mu * math.cos(mu * t) + (a + b) * math.sin(mu
* t))


def U(x:float, y:float, t:float, mu:float=1) -> float:
    return math.sin(x) * math.sin(y) * math.sin(mu * t)

def real_U(X:List[float], Y:List[float], T:List[float]) -> List[List[List[float]]]:
    U_true = [[[0] * len(X) for _ in range(len(Y))] for __ in range(len(T))]
    for i in range(len(T)):
        for k in range(len(Y)):
            for j in range(len(X)):
                U_true[i][k][j] = U(X[j], Y[k], T[i])
    return U_true
```

Вспомогательные функции

```python
def local_error (U_my:list, U_true:list) -> float:
    return sum([abs(a - b) for a, b in zip(U_my, U_true)]) / len(U_my)

def get_error_array_with_h(N:list, x_left:float, x_right:float, y_left:float, y_right:
float, a:float, b:float,
                           mu:float, find_u:typing.Callable,t_end:float=1)  -> (list,
list): # H, error


    H_x = [x_right/(n - 1) for n in N]
    H_y = [y_right/(n - 1) for n in N]
    ERROR_X = []
    ERROR_Y = []
    for n in N:
        XX, YY, TT, UU = find_u(x_left, x_right, y_left, y_right, a, b, mu, n_x=n, n_y
=n, t_end=t_end)
        U_true = real_U(XX, YY, TT)
        t = len(TT) // 2
```

```
        y = len(YY) // 2
        x = len(XX) // 2
        ERROR_X.append(local_error(UU[t][y], U_true[t][y]))
        ERROR_Y.append(local_error(UU[t][:][x], U_true[t][:][x]))

    return H_x, H_y, ERROR_X, ERROR_Y

def h_error_plot(H:list, ERROR:list, s:str=' x') -> None:
    plt.plot(H, ERROR)
    plt.xlabel("h" + s)
    plt.ylabel("error")
    plt.show()

def frange(start:float, stop:float, step:float) -> float:
    while start < stop:
        yield start
        start += step

def get_y(y0:float, y_end:float, h:float) -> List[float]:
    return [i for i in frange(y0, y_end+h, h)]

def get_x(x_0:float, x_l:float, h:float) -> List[float]:
    return [i for i in frange(x_0, x_l+h, h)]

def get_t(t_0:float, t_end:float, h:float) -> List[float]:
    return [i for i in frange(t_0, t_end+h, h)]

def solve_PQ(A0:list, A1:list, A2:list, B:list) -> list:
    P = [-A2[0] / A1[0]]
    Q = [B[0] / A1[0]]
    for i in range(1, len(B)):

        P.append(-A2[i] / (A1[i] + A0[i] * P[i - 1]))
        Q.append((B[i] - A0[i] * Q[i - 1]) / (A1[i] + A0[i] * P[i - 1]))

    res = [Q[-1]]

    for i in range(len(B) - 2, -1, -1):
        res.append(P[i] * res[-1] + Q[i])

    return res[::-1]

def border_conds(U_values:List[List[float]], X:List[float], Y:List[float], T:List[floa
t], mu:float, hx:float, hy:float) -> None:

    for i in range(len(Y)):
        for i in range(len(X)):
            U_values[0][i][j] = psi(X[j], Y[i], mu=mu)
    for k in range(len(T)):
        for i in range(len(X)):
            U_values[k][i][0] = phi_3(X[i], T[k], mu=mu)
            U_values[k][i][-1] = phi_4(X[i], T[k], mu=mu) * 2 * hy + U_values[k][i][-2
]
        for j in range(len(Y)):
            U_values[k][0][j] = phi_1(Y[j], T[k], mu=mu)
            U_values[k][-1][j] = phi_2(Y[j], T[k], mu=mu) * 2 * hx + U_values[k][-2][j
]

def initial_T(U_values:List[List[float]], X:List[float], Y:List[float], mu:float) -> N
one:
    for j in range(len(Y)):
```

```
        for i in range(len(X)):
            U_values[0][j][j] = psi(X[i], Y[j], mu=mu)
```

Метод переменных направлений

$$\frac{u_{i,j}^{k+\frac{1}{2}} - u_{i,j}^{k}}{\frac{\tau}{2}} = \frac{a}{h_1^2}\left(u_{i+1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}}\right) + \frac{b}{h_2^2}\left(u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}\right) + f_{i,j}^{k+\frac{1}{2}}$$

идём по j = $1..y_l - h_y$ по x неявно, по y явно

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k+\frac{1}{2}}}{\frac{\tau}{2}} = \frac{a}{h_1^2}\left(u_{i+1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}}\right) + \frac{b}{h_2^2}\left(u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1}\right) + f_{i,j}^{k+\frac{1}{2}}$$

идём по i = $1..x_l - h_x$ по x явно, по y неявно

```
def changing_directions(x_left:float, x_right:float, y_left:float, y_right:float, a:fl
oat, b:float, mu:float,
                        n_x:float=10, n_y:float=10, t_end:float=1) -> (List[float], Li
st[float], List[List[float]], List[List[float]]):

    t_right = t_end
    hx = x_right / (n_x - 1)
    hy = y_right / (n_y - 1)
    tau = (hx ** 2) / 2 / a**2


    X = get_x(x_left, x_right, hx)
    Y = get_y(y_left, y_right, hy)
    T = get_t(0, t_right, tau)


    U_values = [[[0] * len(X) for _ in range(len(Y))] for __ in range(len(T))]
    sigma_x = (a * tau) / (hx ** 2)
    sigma_y = (b * tau) / (hy ** 2)

    initial_T(U_values, X, Y, mu)
    A0 = [0 for _ in range(len(X))]
    A1 = [0 for _ in range(len(X))]
    A2 = [0 for _ in range(len(X))]
    B = [0 for _ in range(len(X))]
    for k in range(1, len(T)):
        U_part = [[0 for _ in range(len(Y))] for _ in range(len(X))]

        for i in range(1, len(X) - 1):
            A1[0] = -1
            A2[0] = 0
            B[0] = phi_1(Y[0], T[k-1] + tau/2, mu=mu)
            for j in range(1, len(Y) - 1):
                A0[j] = - sigma_x / 2
                A1[j] = 1 + sigma_x
                A2[j] = - sigma_x /2
                B[j] = f(X[i], Y[j], T[k-1] + tau/2, mu=mu) * tau / 2  + sigma_y * (U_
values[k-1][j+1][i] - 2 * U_values[k-1][j][i] + U_values[k-1][j-1][i]) / 2 + U_values[
k-1][j][i]
            A0[-1] = 0
            A1[-1] =  -1
            A2[-1] =  0
            B[-1] = phi_2(Y[-1], T[k-1] + tau/2, mu=mu) * 2 * hx + U_values[k-1][j][-2
]
```

```
            tmp_res = solve_PQ(A0, A1, A2, B)
            U_part[i] = tmp_res

        for j in range(1, len(Y) - 1):
            A1[0] = -1
            A2[0] = 0
            B[0] = phi_3(X[0], T[k], mu=mu)
            for i in range(1, len(X) - 1):
                A0[i] = - sigma_y / 2
                A1[i] = 1 + sigma_y
                A2[i] = - sigma_y / 2
                B[i] = f(X[i], Y[j], T[k-1] + tau/2, mu=mu) * tau / 2 + sigma_x * (U_part[i][j+1] - 2 * U_part[i][j] + U_part[i][j-1]) / 2 + U_part[i][j]
            A0[-1] = 0
            A1[-1] =  -1
            B[-1] = phi_4(X[-1], T[k], mu=mu) * 2 * hy  + U_values[k-1][-2][i]


            tmp_res = solve_PQ(A0, A1, A2, B)
            U_values[k][j] = tmp_res

    return X, Y, T, U_values
```

Демонстрация работы

```
X, Y, T, UU = changing_directions(x_left, x_right, y_left, y_right, a, b, mu, n_x=20, n_y=20, t_end=1)
U_true = real_U(X, Y, T)

fig = go.Figure(data=[go.Surface(z=UU[len(T) // 2], x=X, y=Y)])
fig.show()

fig = go.Figure(data=[go.Surface(z=U_true[len(T) // 2], x=X, y=Y)])
fig.show()

fig = go.Figure(data=[go.Surface(z=UU[len(T) // 2], x=X, y=Y, colorscale='Reds', name='Метод переменных направлений'), go.Surface(z=U_true[len(T) // 2], x=X, y=Y, colorscale='Blues', name='Аналитическое решение')])
fig.update_layout(title='Метод переменных направлений - красный, Аналитическое решение - синий', showlegend=True)
fig.show()
```

Графики ошибок

```
N = [10, 15, 20, 30, 40]
H_X, H_Y, ERROR_X, ERROR_Y = get_error_array_with_h(N, x_left, x_right, y_left, y_right, a, b, mu, find_u=changing_directions, t_end=1)

h_error_plot(H_X, ERROR_X)
h_error_plot(H_Y, ERROR_Y, s=' y')

T[1] - T[0]
```

Метод дробных шагов

Здесь слой $k+\frac{1}{2}$ становится "полностью вирутальныйм"

$$\frac{u_{i,j}^{k+\frac{1}{2}} - u_{i,j}^{k}}{\tau} = \frac{a}{h_1^2}\left(u_{i+1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}}\right) + \frac{f_{i,j}^{k}}{2}$$

идём по j = $1..y_l - h_y$

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k+\frac{1}{2}}}{\tau} = \frac{b}{h_2^2}\left(u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1}\right) + \frac{f_{i,j}^{k+1}}{2}$$

идём по i = $1..x_l - h_x$ по x явно, по y неявно

```
def partial_steps(x_left:float, x_right:float, y_left:float, y_right:float, a:float, b
:float, mu:float,
                        n_x:float=10, n_y:float=10, t_end:float=1) -> (List[float], Li
st[float], List[List[float]], List[List[float]]):
    t_right = t_end
    hx = x_right / (n_x - 1)
    hy = y_right / (n_y - 1)
    tau = (hx**2) / 2 / a**2


    X = get_x(x_left, x_right, hx)
    Y = get_y(y_left, y_right, hy)
    T = get_t(0, t_right, tau)


    U_values = [[[0] * len(X) for _ in range(len(Y))] for __ in range(len(T))]
    sigma_x = (a * tau) / (hx ** 2)
    sigma_y = (b * tau) / (hy ** 2)

    initial_T(U_values, X, Y, mu)
    A0 = [0 for _ in range(len(X))]
    A1 = [0 for _ in range(len(X))]
    A2 = [0 for _ in range(len(X))]
    B = [0 for _ in range(len(X))]
    for k in range(1, len(T)):
        U_part = [[0 for _ in range(len(Y))] for _ in range(len(X))]
        t_part = T[k-1] + tau/2

        for i in range(1, len(X) - 1):
            A1[0] = -1
            A2[0] = 0
            B[0] = phi_1(Y[0], t_part, mu=mu)
            for j in range(1, len(Y) - 1):
                A0[j] = - sigma_x
                A1[j] = 1 + 2 * sigma_x
                A2[j] = - sigma_x
                B[j] = f(X[i], Y[j], T[k-1], mu=mu) * tau / 2  + U_values[k-1][j][i]
            A0[-1] = 0
            A1[-1] =  -1
            A2[-1] =  0
            B[-1] = phi_2(Y[-1], t_part, mu=mu) * 2 * hx + U_values[k][j][-2]  ### тут
надо подумать что брать U_values

            tmp_res = solve_PQ(A0, A1, A2, B)
            U_part[i] = tmp_res

        t_part += tau/2
        for j in range(1, len(Y) - 1):
            A1[0] = -1
            A2[0] = 0
            B[0] = phi_3(X[0], t_part, mu=mu)
            for i in range(1, len(X) - 1):
                A0[i] = - sigma_y
                A1[i] = 1 + 2 * sigma_y
```

```
                A2[i] = - sigma_y
                B[i] = f(X[i], Y[j], T[k], mu=mu) * tau / 2 + U_part[i][j]
            A0[-1] = 0
            A1[-1] =  -1
            B[-1] = phi_4(X[-1], t_part, mu=mu) * 2 * hy  + U_values[k][-2][i]# аналог
ично предидущему варианту с шраницциами
            tmp_res = solve_PQ(A0, A1, A2, B) # езультат, заносим в U_values


            U_values[k][j] = tmp_res

    return X, Y, T, U_values
```

Демонстрация работы

```
XX, YY, TT, UUU = partial_steps(x_left, x_right, y_left, y_right, a, b, mu, n_x=20, n_
y=20, t_end=1)
U_true2 = real_U(XX, YY, TT)

fig = go.Figure(data=[go.Surface(z=UUU[len(T) // 2], x=XX, y=YY)])
fig.show()

fig = go.Figure(data=[go.Surface(z=U_true2[len(T) // 2], x=XX, y=YY)])
fig.show()

fig = go.Figure(data=[go.Surface(z=UUU[len(TT) // 2], x=XX, y=YY, colorscale='Reds', n
ame='Метод дробных шагов'), go.Surface(z=U_true2[len(TT) // 2], x=XX, y=YY, colorscale
='Blues', name='Аналитическое решение')])
fig.update_layout(title='Метод дробных шагов - красный, Аналитическое решение - синий'
, showlegend=True)
fig.show()
```

Гафик ошибок

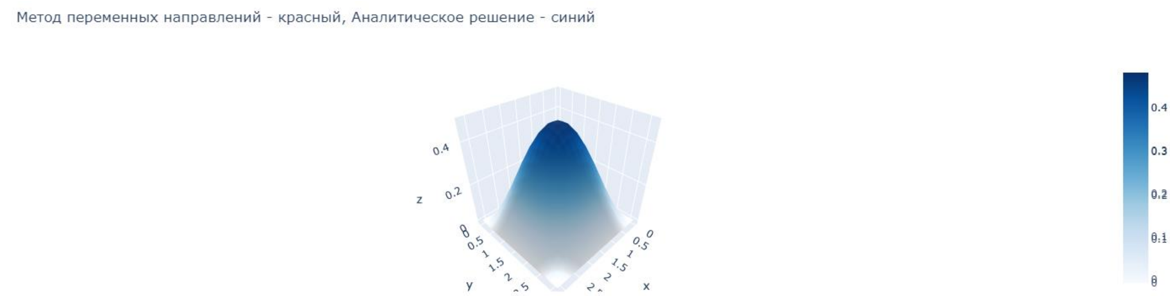
```
N = [10, 15, 20, 30, 40]
H_X, H_Y, ERROR_X, ERROR_Y = get_error_array_with_h(N, x_left, x_right, y_left, y_righ
t, a, b, mu, find_u=partial_steps)

h_error_plot(H_X, ERROR_X)
h_error_plot(H_Y, ERROR_Y, s=' y')
```

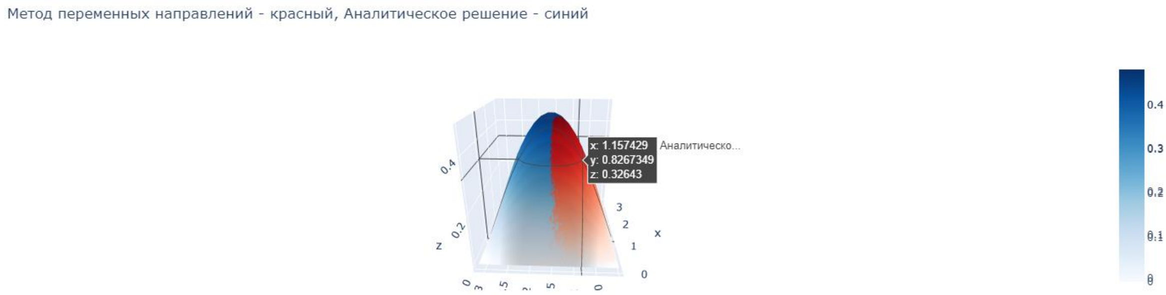
Вывод программы

Для вывода программы был взят вывод графиков, чтобы не дублировать код.

```
Метод переменных направлений, график численного решения и график аналитического
решения
```

Две поверхности на одном графике





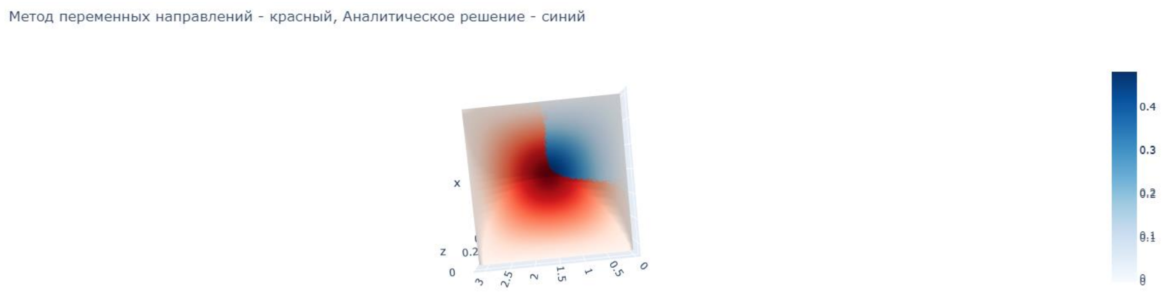
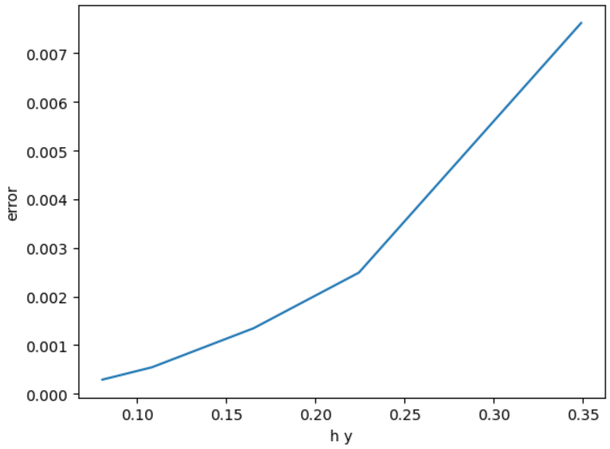
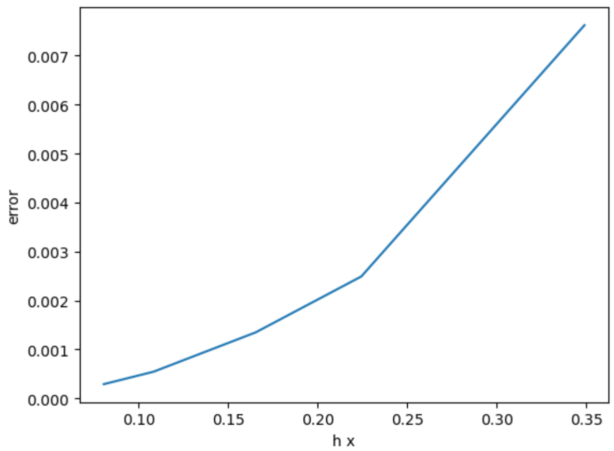


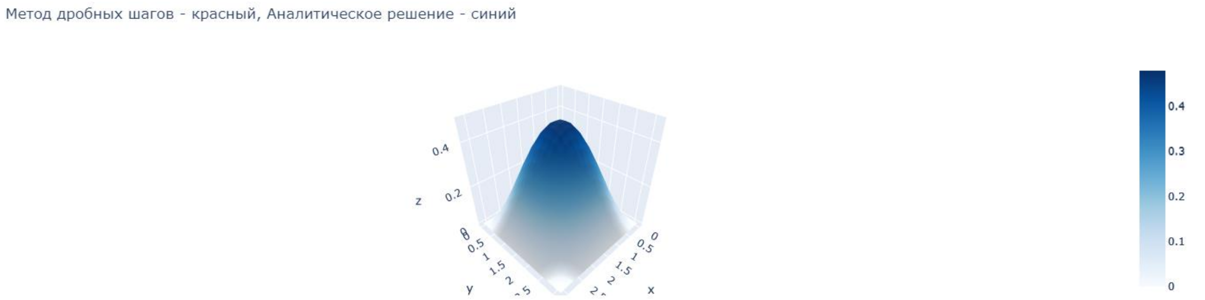
График ошибки в зависимости от шага(метод переменных направлений)

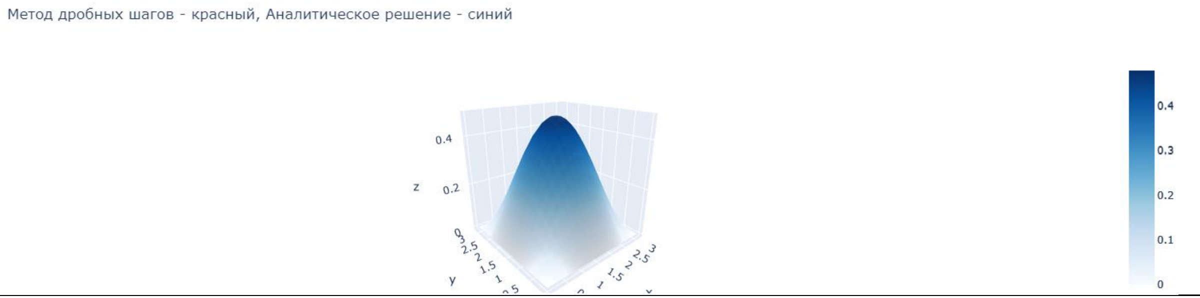


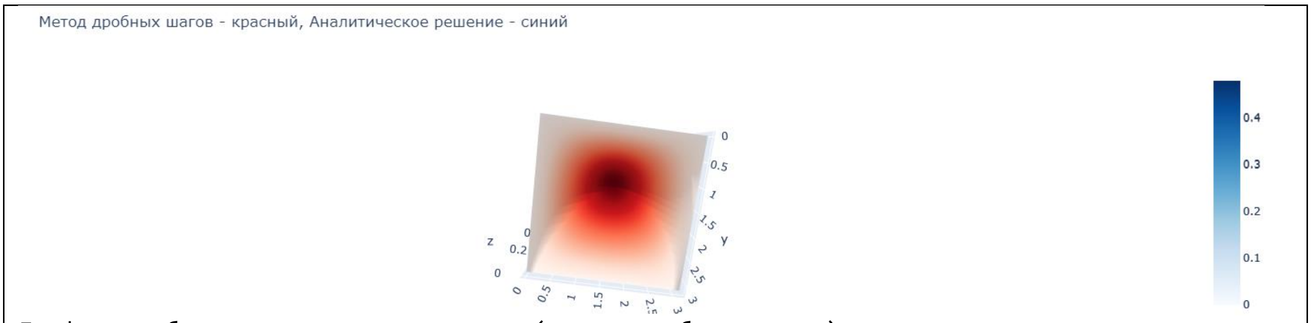


Метод дробных шагов, график численного решения и график аналитического решения

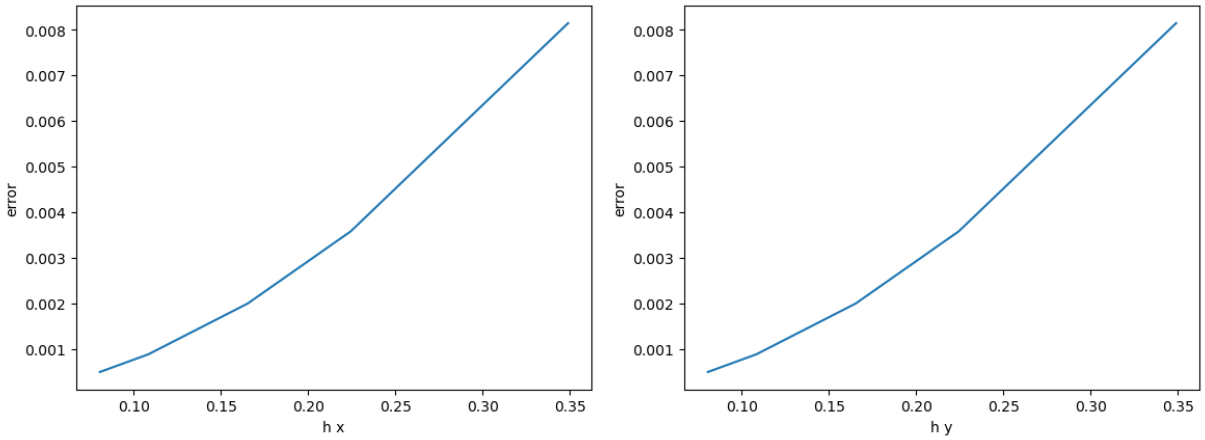
Две поверхности на одном графике








```
График ошибки в зависимости от шага(метод дробных шагов)
```



Заключение:

В данной лабораторной работе построены конечно-разностные схемы (метод переменных направлений и дробных шагов) для дифференциальных уравнений параболического типа с двумя пространственными и одной временной переменными.

Проведена аппроксимация краевых условий второго рода с первым порядком точности.

Схемы были программно реализованы на языке программирования Python в среде разработки Jupyter Notebook.

Исходя из графиков зависимости ошибки от длины шага можно сделать вывод, что для поставленной задачи метод переменных направлений оказался точнее метода дробных шагов. Также было показано, что с увеличением шага дробления практически линейно растет погрешность каждого метода.