Лабораторная работа № 8

Бабичева Анна М8О-404Б-17

Вариант № 2

Импортирую необходимые библиотеки, включая собственную matrix и graphics_for_labs.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import ipympl
import math
import pylab
import random
import matrix

import plotly
import plotly.graph_objs as go
from plotly.subplots import make_subplots

from graphics_for_labs import Graphic
from numpy import arange
from numpy import meshgrid
from matplotlib import mlab
from mpl_toolkits.mplot3d import Axes3D

```

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 погрешности от сеточных параметров h_x, h_y, τ.

Начально-краевая задача:

$$\begin{cases} \frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + a\frac{\partial^2 u}{\partial y^2} \\ u(0, y, t) = cos(\mu_2 y)e^{-(\mu_1^2+\mu_2^2)at} \\ u(\frac{\pi}{2}\mu_1, y, t) = 0 \\ u(x, 0, t) = cos(\mu_1 x)e^{-(\mu_1^2+\mu_2^2)at} \\ u(x, \frac{\pi}{2}\mu_2, t) = 0 \\ u(x, y, 0) = cos(\mu_1 x)cos(\mu_2 y) \end{cases}$$

$1)\mu_1 = 1 \quad \mu_2 = 1$

$2)\mu_1 = 2 \quad \mu_2 = 1$

$3)\mu_1 = 1 \quad \mu_2 = 2$

Аналитическое решение:

$$U(x, y, t) = cos(\mu_1 x)cos(\mu_2 y)e^{-(\mu_1^2+\mu_2^2)at}$$

Задам параметр a:

In [11]:
```
a = 1

```

Для удобства пропишу некоторые вспомогательные функции:

In [3]:
```
def exp_(x):
    return math.exp(x)

def sin_(x):
    return math.sin(x)

def cos_(x):
    return math.cos(x)

pi = math.pi

```

In [4]:
```
def U(x, y, t, m1, m2):
    return cos_(m1 * x) * cos_(m2 * y) * exp_( - (m1 * m1 + m2

```

Начальные и краевые условия:

$$u_{0j}^k = cos(\mu_2 y_j)e^{-(\mu_1^2+\mu_2^2)at^k}$$

$$u_{N_x j}^k = 0$$

$$u_{i0}^k = cos(\mu_1 x_i)e^{-(\mu_1^2+\mu_2^2)at^k}$$

$$u_{iN_y}^k = 0$$

$$u_{ij}^0 = cos(\mu_1 x_i)cos(\mu_2 y_j)$$

In [5]:
```
def u0jk(m1, m2, y, t, j, k):
    return cos_(m2 * y[j]) * exp_( - (m1 * m1 + m2 * m2) * a *

def uNxjk(m1, m2, y, t, j, k):
    return 0

def ui0k(m1, m2, x, t, i, k):
    return cos_(m1 * x[i]) * exp_( - (m1 * m1 + m2 * m2) * a *
```

```

def uiNyk(m1, m2, x, t, i, k):
    return 0

def uij0(m1, m2, x, y, i, j):
    return cos_(m1 * x[i]) * cos_(m2 * y[j])

```

Метод переменных направлений

Конечно-разностные аппроксимации сводят задачу к решению СЛАУ методом прогонки на каждом вспомогательном слое:

$$\frac{u_{ij}^{k+1/2} - u_{ij}^{k}}{\tau/2} = \frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2} - 2u_{ij}^{k+1/2} + u_{i-1j}^{k+1/2}\right) + \frac{a}{h_2^2}\left(u_{ij+1}^{k} - 2u_{ij}^{k} + u_{ij-1}^{k}\right) + f_{ij}^{k+1/2}, \quad (5.78)$$

$$\frac{u_{ij}^{k+1} - u_{ij}^{k+1/2}}{\tau/2} = \frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2} - 2u_{ij}^{k+1/2} + u_{i-1j}^{k+1/2}\right) + \frac{a}{h_2^2}\left(u_{ij+1}^{k+1} - 2u_{ij}^{k+1} + u_{ij-1}^{k+1}\right) + f_{ij}^{k+1/2}. \quad (5.79)$$

In [6]:

```
def MAD(T, Nx, Ny, K, m1, m2, lx=0, rx=pi/2, ly=0, ry=pi/2, ui
    rx = rx * m1
    ry = ry * m2

    tau = T / K
    hx = (rx - lx) / Nx
    hy = (ry - ly) / Ny

    x = [lx + i * hx for i in range(Nx + 1)]
    y = [ly + j * hy for j in range(Ny + 1)]
    t = [k * tau / 2 for k in range(2 * K + 1)]

    u = []

    row_x = []
    for i in range(Nx + 1):
        row_y = []
        for j in range(Ny + 1):
            row_y.append(uij0(m1, m2, x, y, i, j))
        row_x.append(row_y)
    u.append(row_x)
    u = np.array(u)

    ax = a / hx ** 2
    ay = a / hy ** 2

    for k in range(0, 2 * K + 1 - 2, 2):

        # PART 1 ##############################################
        u = np.append(u, [[[0] * (Ny + 1)] * (Nx + 1)], axis=0
        for j in range(Ny + 1):
            u[k + 1][0][j] = x_lims[0](m1, m2, y, t, j, k + 1)
            u[k + 1][Nx][j] = x_lims[1](m1, m2, y, t, j, k + 1
        for i in range(Nx + 1):
```

```
            u[k + 1][i][0] = y_lims[0](m1, m2, x, t, i, k + 1)
            u[k + 1][i][Ny] = y_lims[1](m1, m2, x, t, i, k + 1

        for j in range(1, Ny):
            Ax = []
            bx = []
            for i in range(1, Nx):
                rows = []

                if i == 1:
                    bx.append( - (ay * u[k][i][j - 1] + 2 * (-
                    rows = [ - 2 * (ax + 1 / tau) if (p == 1)
                    rows[1] = ax
                    Ax.append(rows)
                    continue
                elif i == Nx - 1:
                    bx.append( - (ay * u[k][i][j - 1] + 2 * (-
                    rows = [ - 2 * (ax + 1 / tau) if (p == Nx
                    rows[Nx - 3] = ax
                    Ax.append(rows)
                    continue
                else:
                    bx.append( - (ay * u[k][i][j - 1] + 2 * (

                for l in range(1, Nx):
                    if (l == i - 1) | (l == i + 1):
                        rows.append(ax)
                    elif l == i:
                        rows.append(- 2 * (ax + 1 / tau))
                    else:
                        rows.append(0)
                Ax.append(rows)
            res = matrix.Progonka(matrix.Matrix(Ax), bx)

            for i in range (1, Nx):
                u[k + 1][i][j] = res[i - 1]

        # PART 2  ##############################################

        u = np.append(u, [[[0] * (Ny + 1)] * (Nx + 1)], axis=0
        for j in range(Ny + 1):
            u[k + 2][0][j] = x_lims[0](m1, m2, y, t, j, k + 2)
            u[k + 2][Nx][j] = x_lims[1](m1, m2, y, t, j, k + 2
        for i in range(Nx + 1):
            u[k + 2][i][0] = y_lims[0](m1, m2, x, t, i, k + 2)
            u[k + 2][i][Ny] = y_lims[1](m1, m2, x, t, i, k + 2
        for i in range(1, Nx):
            Ay = []
            by = []
            for j in range(1, Ny):
                rows = []

                if j == 1:
                    by.append( - (ax * u[k + 1][i - 1][j] + 2 *
                    rows = [ - 2 * (ay + 1 / tau) if (p == 1)
                    rows[1] = ay
                    Ay.append(rows)
                    continue
                elif j == Ny - 1:
```

```
                by.append( - (ax * u[k + 1][i - 1][j] + 2 *
                rows = [ - 2 * (ay + 1 / tau) if (p == Ny
                rows[Ny - 3] = ay
                Ay.append(rows)
                continue
            else:
                by.append( - (ax * u[k + 1][i - 1][j] + 2 *

            for l in range(1, Ny):
                if (l == j - 1) | (l == j + 1):
                    rows.append(ay)
                elif l == j:
                    rows.append(- 2 * (ay + 1 / tau))
                else:
                    rows.append(0)
            Ay.append(rows)
        res = matrix.Progonka(matrix.Matrix(Ay), by)

        for j in range (1, Ny):
            u[k + 2][i][j] = res[j - 1]

    return x, y, t, u

```

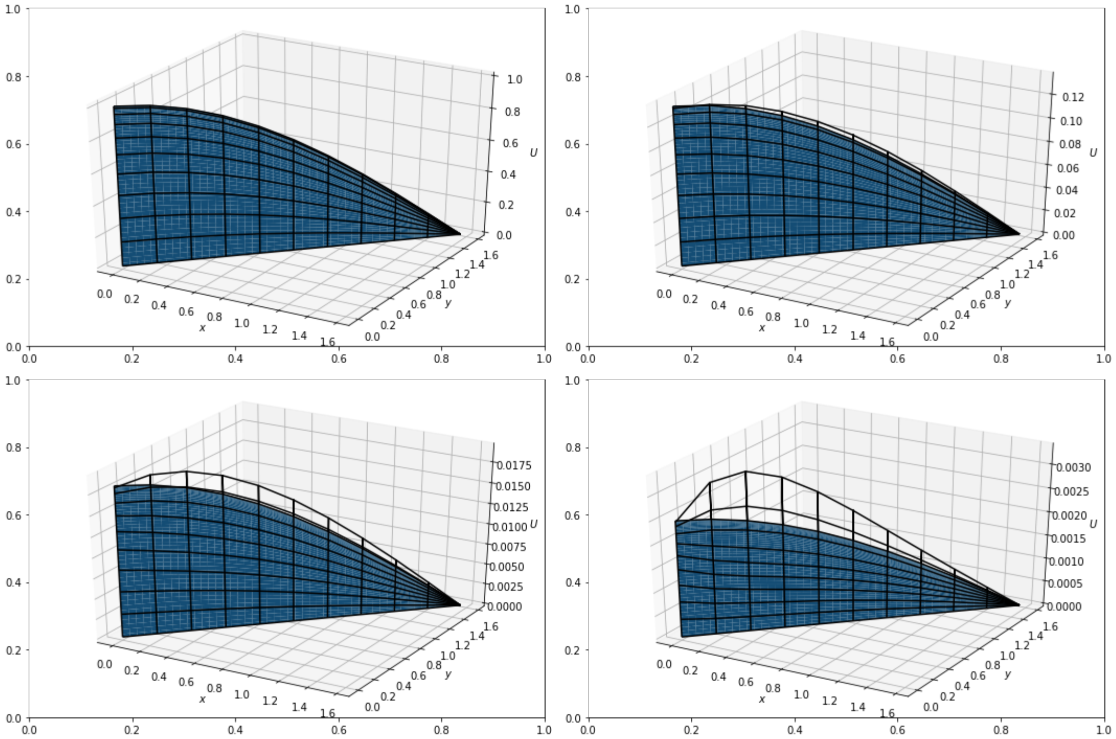
Мы получаем в 2 раза больше временных слоев, поэтому убираем вспомогательные:

In [6]:
```
def make_u_t(u, t):
    new_u = np.array([u[k] for k in range(0, len(u), 2)])
    new_t = np.array([t[k] for k in range(0, len(t), 2)])
    return new_t, new_u

```

In [23]:
```
m1 = 1
m2 = 1

x, y, t, u = MAD(3, 10, 10, 3, m1, m2)

```

In [24]:
```
t, u = make_u_t(u, t)

```

In [10]:
```
1 Graphic.draw_surfaces_levels(x, y, t, m1, m2, u, U, 0, pi / 2,
2
```



In [11]:
```
1 Graphic.draw_level_3d(x, y, 2, m1, m2, u, U, 0, pi / 2, 0, pi /
2
```

Sulutions at t = 2

MAD

MSE:

In [7]:
```
def MSE(x, y, t, u, U, m1, m2):
    s = 0
    for k, t_ in enumerate(t):
        for i, x_ in enumerate(x):
            for j, y_ in enumerate(y):
                s += (U(x_, y_, t_, m1, m2) - u[k][i][j]) *
    return math.sqrt(s)

```

In [25]:
```
print("MSE = {}".format(MSE(x, y, t, u, U, m1, m2)))

```

```
MSE = 0.016339597363757716
```

Метод дробных шагов

Схема:

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

$$\frac{u_{ij}^{k+1} - u_{ij}^{k+1/2}}{\tau} = \frac{a}{h_2^2}\left(u_{ij+1}^{k+1} - 2u_{ij}^{k+1} + u_{ij-1}^{k+1}\right) + \frac{f_{ij}^{k+1}}{2}.$$

In [8]:
```
def FSM(T, Nx, Ny, K, m1, m2, lx=0, rx=pi/2, ly=0, ry=pi/2, ui
    rx = rx * m1
    ry = ry * m2

    tau = T / K
    hx = (rx - lx) / Nx
    hy = (ry - ly) / Ny

    x = [lx + i * hx for i in range(Nx + 1)]
    y = [ly + j * hy for j in range(Ny + 1)]
    t = [k * tau / 2 for k in range(2 * K + 1)]

    u = []
    row_x = []
    for i in range(Nx + 1):
        row_y = []
        for j in range(Ny + 1):
            row_y.append(uij0(m1, m2, x, y, i, j))
        row_x.append(row_y)
    u.append(row_x)
    u = np.array(u)

    ax = a / hx ** 2
```

```
    ay = a / hy ** 2

    for k in range(0, 2 * K + 1 - 2, 2):

        # PART 1 ##############################################
        u = np.append(u, [[[0] * (Ny + 1)] * (Nx + 1)], axis=0
        for j in range(Ny + 1):
            u[k + 1][0][j] = x_lims[0](m1, m2, y, t, j, k + 1)
            u[k + 1][Nx][j] = x_lims[1](m1, m2, y, t, j, k + 1
        for i in range(Nx + 1):
            u[k + 1][i][0] = y_lims[0](m1, m2, x, t, i, k + 1)
            u[k + 1][i][Ny] = y_lims[1](m1, m2, x, t, i, k + 1

        for j in range(1, Ny):
            Ax = []
            bx = []
            for i in range(1, Nx):
                rows = []

                if i == 1:
                    bx.append( - u[k][i][j] / tau - ax * u[k +
                    rows = [ - (2 * ax + 1 / tau) if (p == 1)
                    rows[1] = ax
                    Ax.append(rows)
                    continue
                elif i == Nx - 1:
                    bx.append( - u[k][i][j] / tau - ax * u[k +
                    rows = [ - (2 * ax + 1 / tau) if (p == Nx
                    rows[Nx - 3] = ax
                    Ax.append(rows)
                    continue
                else:
                    bx.append( - u[k][i][j] / tau)

                for l in range(1, Nx):
                    if (l == i - 1) | (l == i + 1):
                        rows.append(ax)
                    elif l == i:
                        rows.append(- (2 * ax + 1 / tau))
                    else:
                        rows.append(0)
                Ax.append(rows)
            res = matrix.Progonka(matrix.Matrix(Ax), bx)

            for i in range (1, Nx):
                u[k + 1][i][j] = res[i - 1]

        # PART 2  #############################################

        u = np.append(u, [[[0] * (Ny + 1)] * (Nx + 1)], axis=0
        for j in range(Ny + 1):
            u[k + 2][0][j] = x_lims[0](m1, m2, y, t, j, k + 2)
            u[k + 2][Nx][j] = x_lims[1](m1, m2, y, t, j, k + 2
        for i in range(Nx + 1):
            u[k + 2][i][0] = y_lims[0](m1, m2, x, t, i, k + 2)
            u[k + 2][i][Ny] = y_lims[1](m1, m2, x, t, i, k + 2
        for i in range(1, Nx):
            Ay = []
            by = []
```

```
            for j in range(1, Ny):
                rows = []

                if j == 1:
                    by.append( - u[k + 1][i][j] / tau - ay * u
                    rows = [ - (2 * ay + 1 / tau) if (p == 1)
                    rows[1] = ay
                    Ay.append(rows)
                    continue
                elif j == Ny - 1:
                    by.append( - u[k + 1][i][j] / tau - ay * u
                    rows = [ - (2 * ay + 1 / tau) if (p == Ny
                    rows[Ny - 3] = ay
                    Ay.append(rows)
                    continue
                else:
                    by.append( - u[k + 1][i][j] / tau)

                for l in range(1, Ny):
                    if (l == j - 1) | (l == j + 1):
                        rows.append(ay)
                    elif l == j:
                        rows.append(- (2 * ay + 1 / tau))
                    else:
                        rows.append(0)
                Ay.append(rows)
            res = matrix.Progonka(matrix.Matrix(Ay), by)

            for j in range (1, Ny):
                u[k + 2][i][j] = res[j - 1]

    return x, y, t, u

```

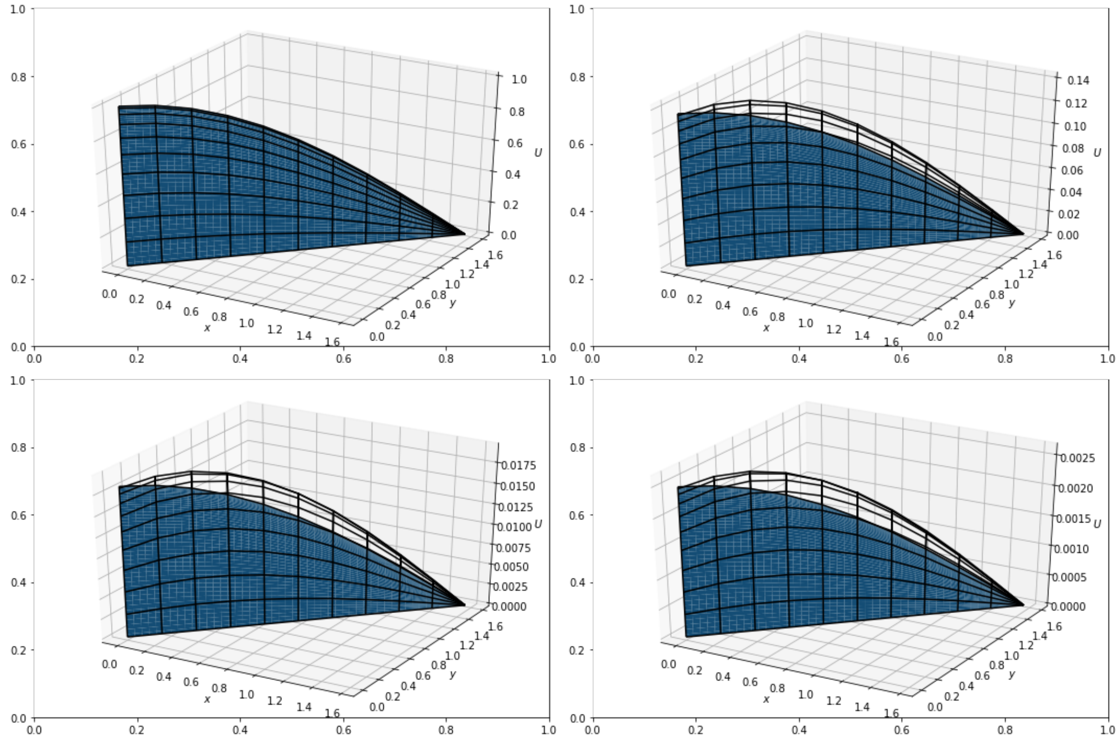
In [12]:

```
m1 = 1
m2 = 1

x, y, t, u = FSM(3, 10, 10, 3, m1, m2)

```

In [13]:

```
t, u = make_u_t(u, t)

```

In [15]:
```
Graphic.draw_surfaces_levels(x, y, t, m1, m2, u, U, 0, pi / 2,
```



In [14]:
```
Graphic.draw_level_3d(x, y, 1, m1, m2, u, U, 0, pi / 2, 0, pi /
```

Sulutions at t = 1

FSM

In [28]:
```
print("MSE = {}".format(MSE(x, y, t, u, U, m1, m2)))

```

```
MSE = 0.11793100897630901
```

In [ ]:
```

```