

Лабораторная работа № 8

Бабичева Анна М8О-404Б-17

Вариант № 2

Импортирую необходимые библиотеки, включая собственную matrix и graphics_for_labs.

```
In [2]: 1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import ipyml
5 import math
6 import pylab
7 import random
8 import matrix
9
10 import plotly
11 import plotly.graph_objs as go
12 from plotly.subplots import make_subplots
13
14 from graphics_for_labs import Graphic
15 from numpy import arange
16 from numpy import meshgrid
17 from matplotlib import mlab
18 from mpl_toolkits.mplot3d import Axes3D
19
```

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]: 1 a = 1
          2
```

Для удобства пропишу некоторые вспомогательные функции:

```
In [3]: 1 def exp_(x):
          2     return math.exp(x)
          3
          4 def sin_(x):
          5     return math.sin(x)
          6
          7 def cos_(x):
          8     return math.cos(x)
          9
          10 pi = math.pi
          11
```

```
In [4]: 1 def U(x, y, t, m1, m2):
          2     return cos_(m1 * x) * cos_(m2 * y) * exp_( - (m1 * m1 + m2
          3
```

Начальные и краевые условия:

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

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

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

$$u_{i N_y}^k = 0$$

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

```
In [5]: 1 def u0jk(m1, m2, y, t, j, k):
          2     return cos_(m2 * y[j]) * exp_( - (m1 * m1 + m2 * m2) * a *
          3
          4 def uNxjk(m1, m2, y, t, j, k):
          5     return 0
          6
          7 def ui0k(m1, m2, x, t, i, k):
          8     return cos_(m1 * x[i]) * exp_( - (m1 * m1 + m2 * m2) * a *
```

```

9
10 def uiNyk(m1, m2, x, t, i, k):
11     return 0
12
13 def uij0(m1, m2, x, y, i, j):
14     return cos_(m1 * x[i]) * cos_(m2 * y[j])
15

```

Метод переменных направлений

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

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

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

```

In [6]: 1 def MAD(T, Nx, Ny, K, m1, m2, lx=0, rx=pi/2, ly=0, ry=pi/2, ui:
2         rx = rx * m1
3         ry = ry * m2
4
5         tau = T / K
6         hx = (rx - lx) / Nx
7         hy = (ry - ly) / Ny
8
9         x = [lx + i * hx for i in range(Nx + 1)]
10        y = [ly + j * hy for j in range(Ny + 1)]
11        t = [k * tau / 2 for k in range(2 * K + 1)]
12
13        u = []
14
15        row_x = []
16        for i in range(Nx + 1):
17            row_y = []
18            for j in range(Ny + 1):
19                row_y.append(uij0(m1, m2, x, y, i, j))
20            row_x.append(row_y)
21        u.append(row_x)
22        u = np.array(u)
23
24        ax = a / hx ** 2
25        ay = a / hy ** 2
26
27        for k in range(0, 2 * K + 1 - 2, 2):
28
29            # PART 1 #####
30            u = np.append(u, [[[0] * (Ny + 1)] * (Nx + 1)], axis=0)
31            for j in range(Ny + 1):
32                u[k + 1][0][j] = x_lims[0](m1, m2, y, t, j, k + 1)
33                u[k + 1][Nx][j] = x_lims[1](m1, m2, y, t, j, k + 1)
34            for i in range(Nx + 1):

```

```

35     u[k + 1][i][0] = y_lims[0](m1, m2, x, t, i, k + 1)
36     u[k + 1][i][Ny] = y_lims[1](m1, m2, x, t, i, k + 1)
37
38     for j in range(1, Ny):
39         Ax = []
40         bx = []
41         for i in range(1, Nx):
42             rows = []
43
44             if i == 1:
45                 bx.append(- (ay * u[k][i][j - 1] + 2 * (-
46                     rows = [- 2 * (ax + 1 / tau) if (p == 1)
47                     rows[1] = ax
48                     Ax.append(rows)
49                     continue
50             elif i == Nx - 1:
51                 bx.append(- (ay * u[k][i][j - 1] + 2 * (-
52                     rows = [- 2 * (ax + 1 / tau) if (p == Nx
53                     rows[Nx - 3] = ax
54                     Ax.append(rows)
55                     continue
56             else:
57                 bx.append(- (ay * u[k][i][j - 1] + 2 * (-
58
59             for l in range(1, Nx):
60                 if (l == i - 1) | (l == i + 1):
61                     rows.append(ax)
62                 elif l == i:
63                     rows.append(- 2 * (ax + 1 / tau))
64                 else:
65                     rows.append(0)
66             Ax.append(rows)
67             res = matrix.Progonka(matrix.Matrix(Ax), bx)
68
69             for i in range(1, Nx):
70                 u[k + 1][i][j] = res[i - 1]
71
72     # PART 2 #####
73
74     u = np.append(u, [[[0] * (Ny + 1)] * (Nx + 1)], axis=0)
75     for j in range(Ny + 1):
76         u[k + 2][0][j] = x_lims[0](m1, m2, y, t, j, k + 2)
77         u[k + 2][Nx][j] = x_lims[1](m1, m2, y, t, j, k + 2)
78     for i in range(Nx + 1):
79         u[k + 2][i][0] = y_lims[0](m1, m2, x, t, i, k + 2)
80         u[k + 2][i][Ny] = y_lims[1](m1, m2, x, t, i, k + 2)
81     for i in range(1, Nx):
82         Ay = []
83         by = []
84         for j in range(1, Ny):
85             rows = []
86
87             if j == 1:
88                 by.append(- (ax * u[k + 1][i - 1][j] + 2 *
89                     rows = [- 2 * (ay + 1 / tau) if (p == 1)
90                     rows[1] = ay
91                     Ay.append(rows)
92                     continue
93             elif j == Ny - 1:

```

```

94         by.append( - (ax * u[k + 1][i - 1][j] + 2 * u[k][i - 1][j]) / tau)
95     rows = [ - 2 * (ay + 1 / tau) if (p == Ny - 1) else 0 for p in range(1, Ny)]
96     rows[Ny - 3] = ay
97     Ay.append(rows)
98     continue
99 else:
100     by.append( - (ax * u[k + 1][i - 1][j] + 2 * u[k][i - 1][j]) / tau)
101
102     for l in range(1, Ny):
103         if (l == j - 1) | (l == j + 1):
104             rows.append(ay)
105         elif l == j:
106             rows.append(- 2 * (ay + 1 / tau))
107         else:
108             rows.append(0)
109     Ay.append(rows)
110     res = matrix.Progonka(matrix.Matrix(Ay), by)
111
112     for j in range(1, Ny):
113         u[k + 2][i][j] = res[j - 1]
114
115     return x, y, t, u
116

```

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

```

In [6]: 1 def make_u_t(u, t):
2         new_u = np.array([u[k] for k in range(0, len(u), 2)])
3         new_t = np.array([t[k] for k in range(0, len(t), 2)])
4         return new_t, new_u
5

```

```

In [23]: 1 m1 = 1
2         m2 = 1
3
4         x, y, t, u = MAD(3, 10, 10, 3, m1, m2)
5

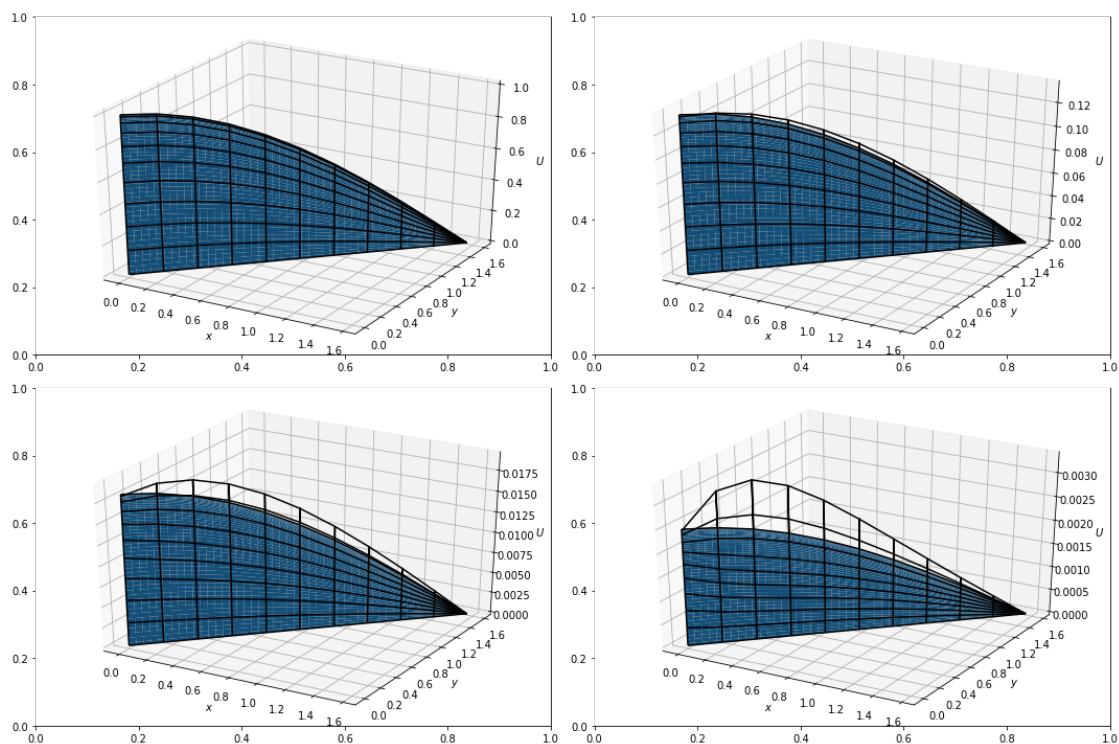
```

```

In [24]: 1 t, u = make_u_t(u, t)
2

```

```
1 Graphic.draw_surfaces_levels(x, y, t, m1, m2, u, U, 0, pi / 2,
```



```
1 Graphic.draw_level_3d(x, y, 2, m1, m2, u, U, 0, pi / 2, 0, pi /
```

Solutions at $t = 2$

MAD

MSE:

```
In [7]: 1 def MSE(x, y, t, u, U, m1, m2):
2         s = 0
3         for k, t_ in enumerate(t):
4             for i, x_ in enumerate(x):
5                 for j, y_ in enumerate(y):
6                     s += (U(x_, y_, t_, m1, m2) - u[k][i][j]) *
7                 return math.sqrt(s)
8
```

```
In [25]: 1 print("MSE = {}".format(MSE(x, y, t, u, U, m1, m2)))
2
```

MSE = 0.016339597363757716

Метод дробных шагов

Схема:

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

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

```
In [8]: 1 def FSM(T, Nx, Ny, K, m1, m2, lx=0, rx=pi/2, ly=0, ry=pi/2, ui:
2         rx = rx * m1
3         ry = ry * m2
4
5         tau = T / K
6         hx = (rx - lx) / Nx
7         hy = (ry - ly) / Ny
8
9         x = [lx + i * hx for i in range(Nx + 1)]
10        y = [ly + j * hy for j in range(Ny + 1)]
11        t = [k * tau / 2 for k in range(2 * K + 1)]
12
13        u = []
14        row_x = []
15        for i in range(Nx + 1):
16            row_y = []
17            for j in range(Ny + 1):
18                row_y.append(uij0(m1, m2, x, y, i, j))
19            row_x.append(row_y)
20        u.append(row_x)
21        u = np.array(u)
22
23        ax = a / hx ** 2
```

```

24     ay = a / hy ** 2
25
26     for k in range(0, 2 * K + 1 - 2, 2):
27
28         # PART 1 #####
29         u = np.append(u, [[[0] * (Ny + 1)] * (Nx + 1)], axis=0)
30         for j in range(Ny + 1):
31             u[k + 1][0][j] = x_lims[0](m1, m2, y, t, j, k + 1)
32             u[k + 1][Nx][j] = x_lims[1](m1, m2, y, t, j, k + 1)
33         for i in range(Nx + 1):
34             u[k + 1][i][0] = y_lims[0](m1, m2, x, t, i, k + 1)
35             u[k + 1][i][Ny] = y_lims[1](m1, m2, x, t, i, k + 1)
36
37         for j in range(1, Ny):
38             Ax = []
39             bx = []
40             for i in range(1, Nx):
41                 rows = []
42
43                 if i == 1:
44                     bx.append(-u[k][i][j] / tau - ax * u[k +
45                     rows = [- (2 * ax + 1 / tau) if (p == 1)
46                     rows[1] = ax
47                     Ax.append(rows)
48                     continue
49                 elif i == Nx - 1:
50                     bx.append(-u[k][i][j] / tau - ax * u[k +
51                     rows = [- (2 * ax + 1 / tau) if (p == Nx
52                     rows[Nx - 3] = ax
53                     Ax.append(rows)
54                     continue
55                 else:
56                     bx.append(-u[k][i][j] / tau)
57
58                 for l in range(1, Nx):
59                     if (l == i - 1) | (l == i + 1):
60                         rows.append(ax)
61                     elif l == i:
62                         rows.append(- (2 * ax + 1 / tau))
63                     else:
64                         rows.append(0)
65                 Ax.append(rows)
66                 res = matrix.Progonka(matrix.Matrix(Ax), bx)
67
68                 for i in range(1, Nx):
69                     u[k + 1][i][j] = res[i - 1]
70
71         # PART 2 #####
72
73         u = np.append(u, [[[0] * (Ny + 1)] * (Nx + 1)], axis=0)
74         for j in range(Ny + 1):
75             u[k + 2][0][j] = x_lims[0](m1, m2, y, t, j, k + 2)
76             u[k + 2][Nx][j] = x_lims[1](m1, m2, y, t, j, k + 2)
77         for i in range(Nx + 1):
78             u[k + 2][i][0] = y_lims[0](m1, m2, x, t, i, k + 2)
79             u[k + 2][i][Ny] = y_lims[1](m1, m2, x, t, i, k + 2)
80         for i in range(1, Nx):
81             Ay = []
82             by = []

```



```
83         for j in range(1, Ny):
84             rows = []
85
86             if j == 1:
87                 by.append( - u[k + 1][i][j] / tau - ay * u
88                 rows = [ - (2 * ay + 1 / tau) if (p == 1)
89                 rows[1] = ay
90                 Ay.append(rows)
91                 continue
92             elif j == Ny - 1:
93                 by.append( - u[k + 1][i][j] / tau - ay * u
94                 rows = [ - (2 * ay + 1 / tau) if (p == Ny
95                 rows[Ny - 3] = ay
96                 Ay.append(rows)
97                 continue
98             else:
99                 by.append( - u[k + 1][i][j] / tau)
100
101             for l in range(1, Ny):
102                 if (l == j - 1) | (l == j + 1):
103                     rows.append(ay)
104                 elif l == j:
105                     rows.append(- (2 * ay + 1 / tau))
106                 else:
107                     rows.append(0)
108             Ay.append(rows)
109             res = matrix.Progonka(matrix.Matrix(Ay), by)
110
111             for j in range(1, Ny):
112                 u[k + 2][i][j] = res[j - 1]
113
114         return x, y, t, u
```

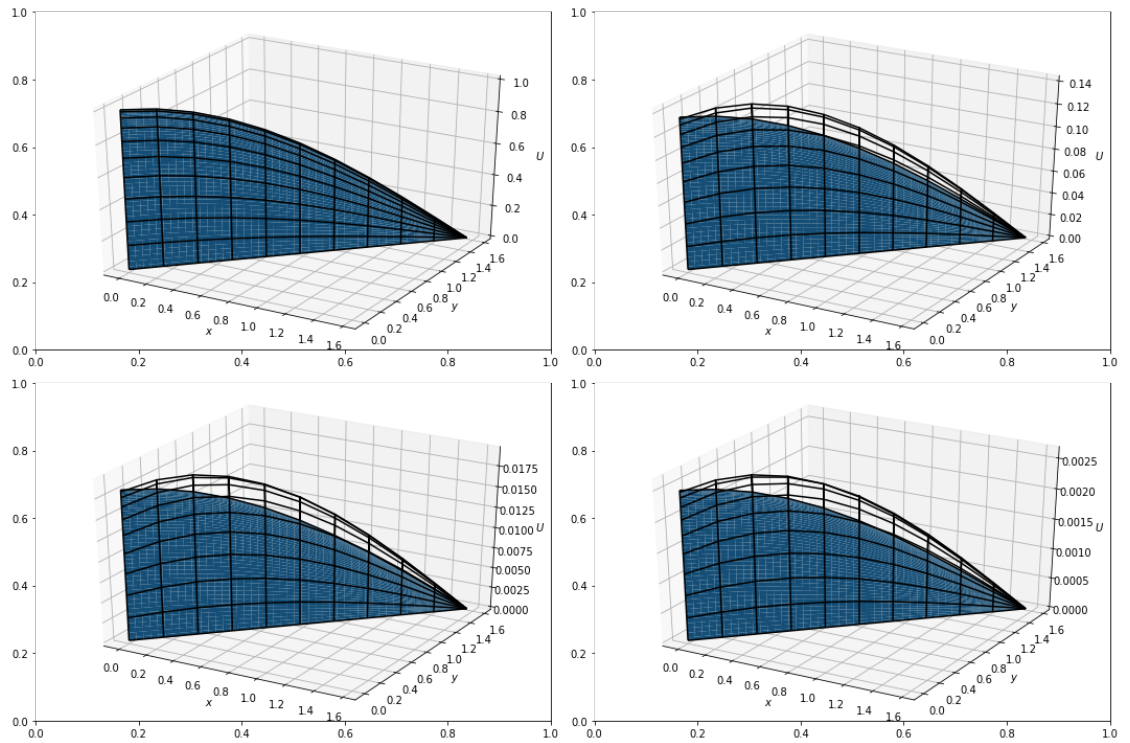
```
115 In [12]: 1 m1 = 1
          2 m2 = 1
          3
          4 x, y, t, u = FSM(3, 10, 10, 3, m1, m2)
          5
```

```
In [13]: 1 t, u = make_u_t(u, t)
          2
```

In [15]:

```
1 Graphic.draw_surfaces_levels(x, y, t, m1, m2, u, U, 0, pi / 2,
```

```
2
```



In [14]:

```
1 Graphic.draw_level_3d(x, y, 1, m1, m2, u, U, 0, pi / 2, 0, pi /
```

```
2
```

Solutions at $t = 1$

FSM

```
In [28]: 1 print("MSE = {}".format(MSE(x, y, t, u, U, m1, m2)))  
        2  
        MSE = 0.11793100897630901
```

```
In [ ]: 1
```