Санкт-Петербургский политехнический университет

Высшая школа теоретической механики, ФизМех

Направление подготовки

«01.03.03 Механика и математическое моделирование»

Индивидуальное задание № 3

тема "Решение плоской задачи теплопроводности"
дисциплина "Вычислительная механика"
Вариант 2

Выполнил студент гр. 5030103/90301 М. А. Бенюх

Преподаватель: Е.Ю. Витохин

Санкт-Петербург

2022

Содержание:

1. Формулировка задачи.

Требуется определить стационарное распределение температур в плотине. На границе контакта плотины и окружающей среды зададим граничные условия – температуры сред: синим цветом - грани, имеющие температуру воды, красным – температуру воздуха.

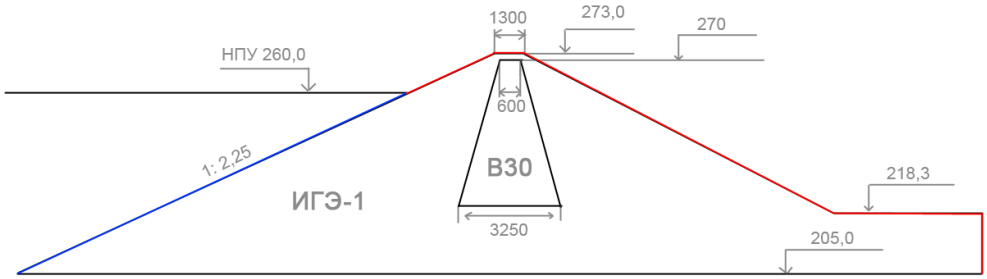

Рис.1. Постановка задачи

Ниже приведены параметры, используемые в задаче.

Параметр	Значение
Коэффициент теплопроводности грунта	$1.5\ \frac{Вт}{м \cdot К}$
Коэффициент теплопроводности бетона	$1.75\ \frac{Вт}{м \cdot К}$
Температура воздуха	$25℃$
Температура воды	$5℃$

Таблица 1. Параметры задачи

2. Алгоритм метода.

Введем треугольный конечный элемент и функции форм

$$T = A + Bx + Cy$$
$$T = T_i N_i + T_j N_j + T_k N_k = [N] \cdot \{T\} \quad (1)$$

Запишем Закон Фурье

$$\underline{h} = -\varkappa \nabla T, \qquad \varkappa - \text{коэффициент теплопроводности} \quad (2)$$

Распишем закон Фурье из (2) покомпонентно:

$$h_x = -\lambda_x \frac{\partial T}{\partial x}, \qquad h_y = -\lambda_y \frac{\partial T}{\partial y}, \quad (3)$$

Подставим (3) в (2), а затем в соотношение (1):

$$\{h\} = \begin{Bmatrix} -\lambda_x \left(\frac{\partial N_i}{\partial x} T_i + \frac{\partial N_j}{\partial x} T_j + \frac{\partial N_k}{\partial x} T_k \right) \\ -\lambda_y \left(\frac{\partial N_i}{\partial y} T_i + \frac{\partial N_j}{\partial y} T_j + \frac{\partial N_k}{\partial y} T_k \right) \end{Bmatrix} \quad (4)$$

Вынесем из (4) T_i, T_j, T_k:

$$\{h\} = -\lambda [B] \{T^e\} \quad (5)$$

Где $[B]$ – матрица температурных градиентов,

$$[B] = \begin{bmatrix} \frac{\partial N_i}{\partial x} & \frac{\partial N_j}{\partial x} & \frac{\partial N_k}{\partial x} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_j}{\partial y} & \frac{\partial N_k}{\partial y} \end{bmatrix}$$

Перейдем к решению задачи теплопроводности. Запишем уравнение баланса внутренней энергии:

$$\rho \dot{u} = -\nabla \cdot \underline{h}$$

Выражение для внутренней энергии:

$$u = C_V T, \qquad C_V - \text{удельная теплоемкость при постоянном объеме}$$

Подставим (2) в выражение баланса энергии (1):

$$\rho C_V \dot{T} = -\nabla \cdot \underline{h} \quad (6)$$

Подставим выражения (1) и (5) в (6):

$$\rho \mathrm{C}_V [N] \cdot \{\dot{T^e}\} - \lambda [B] \{T^e\} = 0 \quad (7)$$

Полученное уравнение решим с помощью метода Галеркина:

$$\int_V \left(\rho \mathrm{C}_V [N] \cdot \{\dot{T^e}\} - \lambda [B] \{T^e\} \right) \cdot [N] dV = 0$$

$$\rho \mathrm{C}_V \int_V [N] \cdot \{\dot{T^e}\} \cdot [N] dV - \lambda \int_V [B]^T \{B\} dV \{T^e\} = 0$$

Добавим граничные условия на температуру:

$$\rho \mathrm{C}_V \int_V [N]^T \cdot [N] dV \cdot \{\dot{T^e}\} + \lambda \int_V [B]^T \{B\} dV \{T^e\} =$$

$$= -\int_{S_1} \{h^b\} \{n\} [N]^T dS + \int_{S_2} h_s [N]^T dS - \varkappa \int_{S_3} (Ts - T_{\mathcal{F}}) [N]^T dS$$

$$[C] \cdot \left\{\dot{T}^e\right\} + ([K_c] + [K_\varkappa]) \{T^e\} = \{R_T\} + \{R_h\} + \{R_\varkappa\} \quad (8)$$

Где

$$[C] = \rho C_V \int_V [N] \cdot \{\dot{T^e}\} \cdot [N] dV - \text{матрица теплоемкости}$$

$$[K_c] = \lambda \int_V [B]^T \{B\} dV - \text{матрица теплопроводности}$$

$$[K_\varkappa] = \varkappa \int_{S_3} [N]^T [N] dS - \text{матрица конвективности}$$

Матрицы внешних нагрузок:

$$\{R_T\} = -\int_{S_1} \{h^b\}\{n\}[N]^T dS - \text{тепловой поток через границу } S_1$$

$$\{R_h\} = \int_{S_2} h_s [N]^T dS - \text{ тепловой поток через границу } S_2$$

$$\{R_\varkappa\} = \varkappa \int_{S_3} T_\mathcal{F} [N]^T dS - \text{ конфективный тепловой поток через границу } S_3$$

Для стационарного случая без конвективного теплообмена уравнение (8) примет вид:

$$[K_c]\{T^e\} = \{R_T\} + \{R_h\}$$

Для вычисления $[B]$ введем матрицу $[J]$:

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_i}{\partial \xi} & \frac{\partial N_j}{\partial \xi} & \frac{\partial N_k}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} & \frac{\partial N_j}{\partial \eta} & \frac{\partial N_k}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_i & y_i \\ x_j & y_j \\ x_k & y_k \end{bmatrix}$$

$$\begin{Bmatrix} \frac{\partial N_m}{\partial x} \\ \frac{\partial N_m}{\partial x} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial N_m}{\partial \xi} \\ \frac{\partial N_m}{\partial \eta} \end{Bmatrix}, \qquad m = i, j, k$$

3. Результаты работы в Abaqus

Для построения было использовано NN конечных элемента и NN узлов.

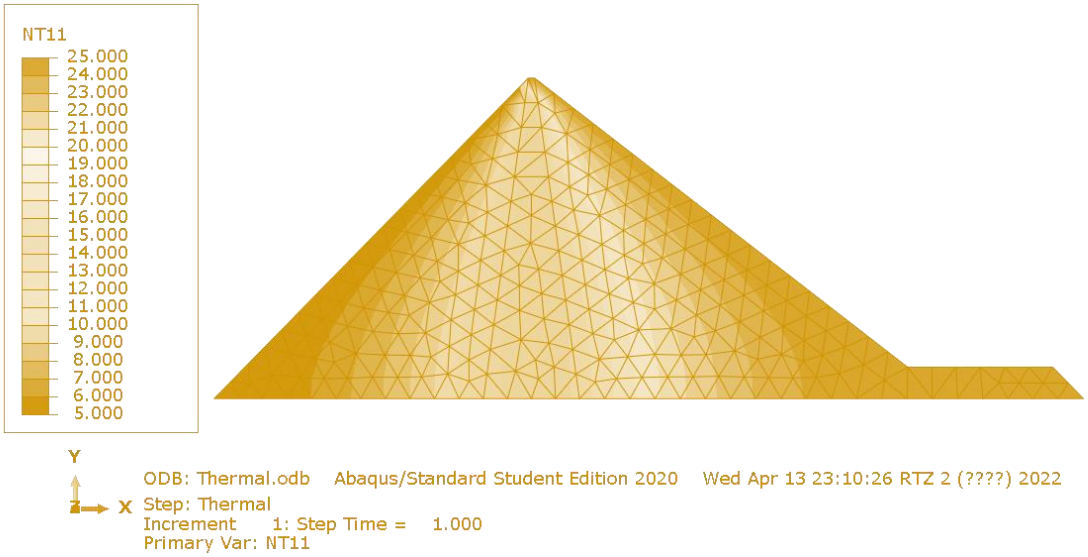

Рис.2. Поле температур, полученное с помощью Abaqus

4. Результаты в Python

Для построения были использованы те же узлы.
Поле температур и геометрия задачи отражены с помощью программы Paraview

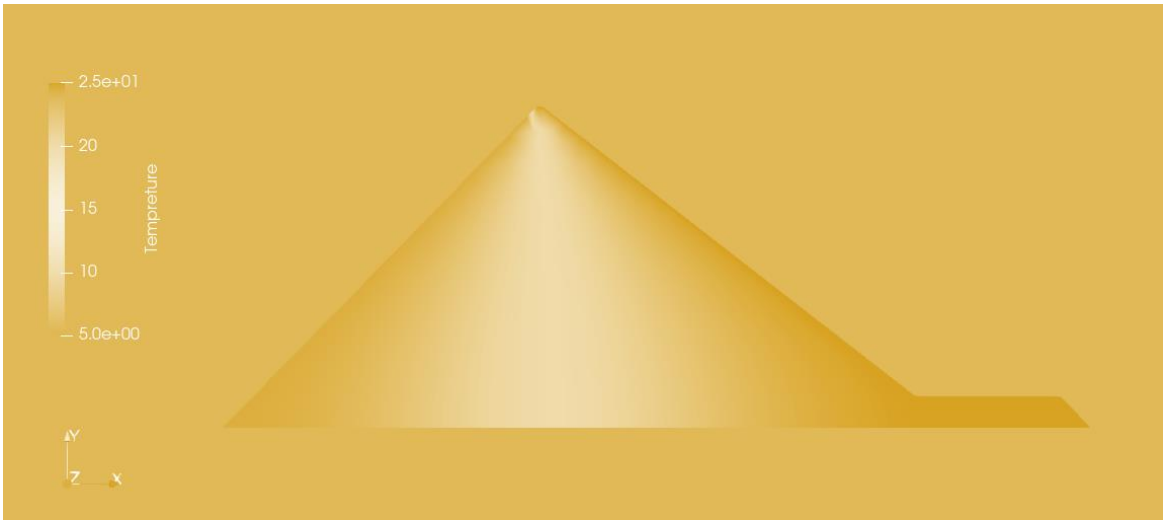

Рис.3. Поле температур, полученное с помощью Python

5. Сравнение результатов

Приведем таблицу с результатами полей температур, полученных в Python и Abaqus.

Таблица 1. Результаты полей температур, полученных в Python и Abaqus.

Номер узла	Температура в Python	Температура в Abaqus	Номер узла	Температура в Python	Температура в Abaqus
1			29		
2			30		
3			31		
4			32		
5			33		
6			34		
7			35		
8			36		
9			37		
10			38		
11			39		
12			40		
13			41		
14			42		
15			43		
16			44		
17			45		
18			46		
19			47		
20			48		
21			49		
22			50		
23			51		
24			52		
25			53		
26			54		
27			55		
28			56		

Построим график относительной погрешности в каждом узле по формуле

$$\delta T = \frac{|T_{abaqus} - T_{python}|}{|T_{abaqus}|}$$

Где T_{abaqus} – температура в i-том узле, полученная с помощью Abaqus, а T_{python} – температура в i-том узле, полученная с помощью Python.

Рис.4. Значения относительной погрешности

Можно заметить, что результаты имеют малую погрешность порядка N знака за запятой.

Заключение

Для решения плоской задачи теплопроводности плотины был использован метод конечных элементов, были получены поле температур в КЭМ-пакете Abaqus и с помощью программирования на Python.

Было проведено сравнение результатов и построен график относительной погрешности результатов для каждого узла. Результаты, полученные разными способами, имеют незначительную относительную погрешность, что говорит о том, что решение произведено правильно.

Код программы

```
import numpy as np
import sympy as sym
from scipy.linalg import solve
import matplotlib.pyplot as plt
from scipy import integrate

class thermal_conductivity:
    def __init__(self, x, l, Mass_node, Mass_Element, Mass_Element_Priming,
Mass_node_T_air, Mass_node_T_water,
                 Lambda_Concrete, Lambda_Priming, T_air, T_water):
        print("__init__ termal")
        self.x = x
        self.l = l
        self.E = E
        # self.rho = rho
        self.g = 9.8
        # self.P = P
        self.T_air = T_air
        self.T_water = T_water
        self.Lambda_Concrete = Lambda_Concrete
        self.Lambda_Priming = Lambda_Priming
        self.Mass_node = Mass_node
        self.Mass_Element = Mass_Element
        self.Mass_Element_Priming = Mass_Element_Priming
        self.Mass_node_T_air = Mass_node_T_air
        self.Mass_node_T_water = Mass_node_T_water
        self.sym_eta = sym.Symbol('x')
        self.sym_ksi = sym.Symbol('y')
        self.sym_N_i = 1 - self.sym_ksi - self.sym_eta
        self.sym_N_j = self.sym_eta
        self.sym_N_k = self.sym_ksi
        self.B = np.matrix([[-1, 0, 1], [-1, 1, 0]])

        # self.k_e = self.CreateMatrix_k_e()
        # self.M_e = self.CreateMatrix_m_e()

    def CreateMatrix_C_e(self):
        print("CreateMatrix_M_e2")
        N = np.matrix([self.sym_N_i, self.sym_N_j, self.sym_N_k]).copy()
        CCC = sym.simplify(np.dot(N.T, N)).copy()
        C_e = np.zeros((4, 4)).copy()
        for i, row in enumerate(CCC):
            for j, element in enumerate(row):
                C_e[i, j] = sym.integrate(element, (self.sym_eta, -1, 1))
        C_e *= self.rho * self.l / 2
        # print(M_e, "M_e")
        return C_e

    def get_global(self, ind, Matrix_e):
        K = np.zeros((len(self.Mass_node), len(self.Mass_node)))
        K[ind[0], ind[0]] = Matrix_e[0, 0]
        K[ind[0], ind[1]] = Matrix_e[0, 1]
        K[ind[1], ind[0]] = Matrix_e[1, 0]
        K[ind[1], ind[2]] = Matrix_e[1, 2]
        K[ind[2], ind[1]] = Matrix_e[2, 1]
        K[ind[0], ind[2]] = Matrix_e[0, 2]
        K[ind[2], ind[0]] = Matrix_e[2, 0]
        K[ind[1], ind[1]] = Matrix_e[1, 1]
        K[ind[2], ind[2]] = Matrix_e[2, 2]
        return K
```

```
    def get_stiffness_matrix(self, coords, Lambda):
        Coord = np.matrix([[coords[0, 0], coords[0, 1]], [coords[1, 0],
coords[1, 1]], [coords[2, 0], coords[2, 1]]])
        J = self.B * Coord
        # print(J)
        # print(self.B)
        Res_B = np.zeros((3, 2))
        # new_B = np.matrix([[-1, -1], [0, 1], [1, 0]])

        for i in range(3):
            vec_B = np.array([self.B[0, i], self.B[1, i]])
            # print('vec_B', vec_B)
            B = solve(J, vec_B)
            Res_B[i, 0] = B[0]
            Res_B[i, 1] = B[1]
            # print('B', B)
        print("Res_B", Res_B)
        print(np.dot(Res_B, Res_B.T))
        Ki = Lambda * np.dot(Res_B, Res_B.T) * np.linalg.det(J) / 2

        # print('Ki', Res_Ki)
        print('Ki', Ki)
        return Ki

    def Solve(self):  # , GU, time, dt
        print('solve')
        N = len(self.Mass_node)
        # print(N)
        K = np.zeros((N, N))

        for i in range(len(self.Mass_Element)):
            el = self.Mass_Element[i]
            # print('rrrrrr', i, el)
            Enodes = np.matrix([self.Mass_node[el[0]], self.Mass_node[el[1]],
self.Mass_node[el[2]]])
            print(Enodes)
            if i in Mass_Element_Priming:
                # print("Lambda_Priming")
                _lambda = self.Lambda_Priming
            else:
                # print('Lambda_Concrete')
                _lambda = self.Lambda_Concrete
            Ki = self.get_stiffness_matrix(Enodes, _lambda)
            Ki = self.get_global(self.Mass_Element[i], Ki)
            K = K + Ki
        F = np.zeros((N, 1))

        for i in range(len(self.Mass_node_T_air)):
            K[self.Mass_node_T_air[i], :] = 0
            # K[:,self.Mass_node_T_air[i]] = 0
            K[self.Mass_node_T_air[i], self.Mass_node_T_air[i]] = 1
            F[self.Mass_node_T_air[i]] = self.T_air

        for i in range(len(self.Mass_node_T_water)):
            K[self.Mass_node_T_water[i], :] = 0  # np.zeros((1, N))
            # K[:, self.Mass_node_T_water[i]] = 0
            K[self.Mass_node_T_water[i], self.Mass_node_T_water[i]] = 1
            F[self.Mass_node_T_water[i]] = self.T_water
        print(K)
        # for i in range(len(K)):
        #     for j in range(len(K[i])):
        #         print(K[i][j], end=' ')
        T = solve(K, F)
```

```
        np.savetxt('test1.txt', T, fmt='%.7f')
        print(T)
        print(len(T))


if __name__ == '__main__':
    node = open('nodes.txt', 'r')
    Mass_node = [[float(i) for i in (line.replace(" ", '').split(",")[1:])]
for line in node.read().splitlines()]
    elem_all = open('elem_nodes.txt', 'r')
    Mass_Element = [[int(i) - 1 for i in line.replace(" ",
'').split(",")[1:]] for line in elem_all.read().splitlines()]
    np.savetxt('MMMMM.txt', Mass_Element, fmt='%d')
    # elem_Concrete = open('elem_Concrete.txt', 'r')
    # Mass_Element_Concrete=[int(i) for i in
elem_Concrete.read().replace("\n", ',').replace(" ", '').split(",")]
    elem_Priming = open('elem_Priming.txt', 'r')
    Mass_Element_Priming = [int(i) - 1 for i in
elem_Priming.read().replace("\n", ',').replace(" ", '').split(",")]
    node_T_air = open('node_T_air.txt', 'r')
    Mass_node_T_air = [int(i) - 1 for i in node_T_air.read().replace("\n",
',').replace(" ", '').split(",")]
    node_T_water = open('node_T_water.txt', 'r')
    Mass_node_T_water = [int(i) - 1 for i in
node_T_water.read().replace("\n", ',').replace(" ", '').split(",")]

    print(Mass_node_T_water)

    print('start')

    Lambda_Concrete = 1.75
    Lambda_Priming = 1.5
    T_air = 25
    T_water = 5
    E = 2 * (10 ** 11)  # сталь
    # rho = 7800
    # M = 10000
    l = 0.1
    x = 1

    test = thermal_conductivity(x, l, Mass_node, Mass_Element,
Mass_Element_Priming, Mass_node_T_air, Mass_node_T_water,
                                Lambda_Concrete, Lambda_Priming, T_air,
T_water)
    test.Solve()
```