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 3](#_Toc43323906)

[2. Алгоритм метода 3](#_Toc43323908)

[3. Результаты в Abaqus 6](#_Toc43323909)  
[4. Результаты в Python 6](#_Toc43323909)

[5. Сравнение результатов 7](#_Toc43323909)  
[6. Заключение N](#_Toc43323909)  
[7. Код программы N](#_Toc43323909)

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 |
| Коэффициент теплопроводности бетона | 1.75 |
| Температура воздуха |  |
| Температура воды |  |
|  |  |

Таблица 1. Параметры задачи

1. **Алгоритм метода.**

Введем треугольный конечный элемент и функции форм

Запишем Закон Фурье

Распишем закон Фурье из (2) покомпонентно:

Подставим (3) в (2), а затем в соотношение (1):  
Вынесем из (4) :

Где – матрица температурных градиентов,

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

Выражение для внутренней энергии:

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

Подставим выражения (1) и (5) в (6):

Полученное уравнение решим с помощью метода Галеркина:

Добавим граничные условия на температуру:

Где

Матрицы внешних нагрузок:

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

Для вычисления введем матрицу :

1. **Результаты работы в Abaqus**

Для построения было использовано NN конечных элемента и NN узлов.

|  |
| --- |
|  |
| Рис.2. Поле температур, полученное с помощью Abaqus |

1. **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

Где – температура в i-том узле, полученная с помощью Abaqus, а – 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\_С\_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()