Karolina Rojowska IS 3, gr 3

Nieustalony transfer ciepła w oparciu o standardowy algorytm MES

Metoda Elementów Skończonych – zaawansowana metoda rozwiązywania układów równań różniczkowych, opierająca się na podziale dziedziny (tzw. dyskretyzacja) na skończone elementy, dla których rozwiązanie jest przybliżane przez konkretne funkcje, i przeprowadzaniu faktycznych obliczeń tylko dla węzłów tego podziału.

Program ma na celu obliczenie wartości temperatur we wszystkich węzłach siatki w iteracjach czasu. Całość sprowadza się do rozwiązania układu równań:

$$\left([H]+\frac{[C]}{\Delta\tau}\right)\{t_1\}-\left(\frac{[C]}{\Delta\tau}\right)\{t_0\}+\{P\}=0$$

gdzie:

$$[H]=\int_V k(t)\left(\left\{\frac{\partial\{N\}}{\partial x}\right\}\left\{\frac{\partial\{N\}}{\partial x}\right\}^T+\left\{\frac{\partial\{N\}}{\partial y}\right\}\left\{\frac{\partial\{N\}}{\partial y}\right\}^T\right)dV+\int_S \alpha\{N\}\{N\}^T\,dS$$

$$\{P\}=-\int_S \alpha\{N\}t_\_\,dS$$

- Stworzenie siatki MES

- utworzenie elementów i węzłów - podstawowe klasy Element oraz Node

```
class Element {

public:
    int id_element;
    int id_node[4];
    int id_walls[4];
    double localvectorP[4];
    double**localmatrixH;
```

id_element - id elementu

id_node[ ] - tablica z id węzłów należących do danego elementu

id_walls [ ] - tablica z id ścian danego elementu

localvectorP [ ] - tablica przechowująca wektor P danego elementu

localmatrix - tablica przechowująca lokalną macierz H danego elementu

```
class Node {

public:
    int id;
    double x, y;
    bool edge;
```

id -id węzła

x, y - współrzędne węzła

edge - zmienna logiczna - czy jest warunek brzegowy na krawędzi

- przesłanie danych w pliku tekstowym .txt

```
0.100
0.100
4
4
7800
300
100
1200
25
700
500
50
9
16
```

```
switch (numOf_line){
case 1: H = stod(line, &st); break;
case 2: L = stod(line, &st); break;
case 3: nH = atoi(line.c_str()); break;
case 4: nL = atoi(line.c_str()); break;
case 5: ro = stod(line.c_str()); break;
case 6: alfa = stod(line.c_str()); break;
case 7: t0 = stod(line.c_str()); break;
case 8: t_env = stod(line.c_str()); break;
case 9: conductivity = stod(line.c_str()); break;
case 10: specific_heat = stod(line.c_str()); break;
case 11: sym_time = stod(line.c_str()); break;
case 12: step_time = stod(line.c_str()); break;
case 13: num_elements = atoi(line.c_str()); break;
case 14: num_nodes = atoi(line.c_str()); break;
}
```

- stworzenie klasy LocalLayout przechowującą wszystkie dane na temat funkcji kształtu

```
class LocalLayout {

public:
    double ksi[4];
    double eta[4];

    double ksi_S[8];
    double eta_S[8];

    double NdV[4][4];
    double NdS[8][4];
    double dNdksi[4][4];
    double dNdeta[4][4];
```

ksi [ ], eta [ ] - tablice wartości dla ksi i eta po objętości (do obliczenia pierwszej części macierzy H)

ksi_S [ ], eta_S [ ] - tablice wartości dla ksi i eta po powierzchni (do obliczenia drugiej części macierzy H)

- Obliczenie jakobianu dla 2d potrzebnego do wyznaczenia pierwszej części macierzy H - po objętości, obliczenie macierzy H po dV oraz do dS co daje całą macierz H

$$\int_V k(t)\left(\left\{\frac{\partial\{N\}}{\partial x}\right\}\left\{\frac{\partial\{N\}}{\partial x}\right\}^T+\left\{\frac{\partial\{N\}}{\partial y}\right\}\left\{\frac{\partial\{N\}}{\partial y}\right\}^T\right)dV$$

- wyznaczenie jakobianu przekształcenia

$$\frac{\partial x}{\partial \xi}=\frac{\partial N_1}{\partial \xi}x_1+\frac{\partial N_2}{\partial \xi}x_2+\frac{\partial N_3}{\partial \xi}x_3+\frac{\partial N_4}{\partial \xi}x_4$$

$$\frac{\partial x}{\partial \eta}=\frac{\partial N_1}{\partial \eta}x_1+\frac{\partial N_2}{\partial \eta}x_2+\frac{\partial N_3}{\partial \eta}x_3+\frac{\partial N_4}{\partial \eta}x_4$$

$$\frac{\partial y}{\partial \xi}=\frac{\partial N_1}{\partial \xi}y_1+\frac{\partial N_2}{\partial \xi}y_2+\frac{\partial N_3}{\partial \xi}y_3+\frac{\partial N_4}{\partial \xi}y_4$$

$$\frac{\partial y}{\partial \eta}=\frac{\partial N_1}{\partial \eta}y_1+\frac{\partial N_2}{\partial \eta}y_2+\frac{\partial N_3}{\partial \eta}y_3+\frac{\partial N_4}{\partial \eta}y_4$$

Korzystając z zasady rozwiązywania układu równań tworzona jest macierz odwrotna jakobianu przekształcenia

$$[A]^{-1}=\frac{1}{\det A}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Stąd:

$$\begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} = \frac{1}{\det[J]} \begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial y}{\partial \xi} \\ -\frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \xi} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix}$$

następnie obie macierze po x i y są transponowane i dodawane do siebie, potem mnożone razy stałą k i razy wyznacznik, aby zniknęła całka po dV.

I tym sposobem tworzona jest macierz H po dV dla jednego elementu i dla danego punktu na jednej ze ścian. Następnie sumowane są macierze dla wszystkich 4 punktów na 4 ścian i otrzymywana jest macierz H po dV dla jednego elementu.

- utworzenie macierzy po dS

$$\int_S \alpha \{N\} \{N\}^T \, dS$$

alfa (współczynnik konwekcji) odczytywana jest z pliku tekstowego; wyznaczenie punktów całkowania, dwa dla każdego boku; w zależności czy są spełnione warunki brzegowe czy nie mogą być wartości funkcji kształtu nierezowe lub zerowe; mnożone jest wszystko przez jakobian (bok/2), aby pozdyć się całki po dS

- H po dV + H po dS = H całościowe

sumowane są obie otrzymane macierze, po dV i dS i tak otrzymana jest macierz H dla danego elementu

- utworzenie macierzy C
liczone są funkcje kształtu dla wszystkich 4 puktów całkowania, następnie transponowane i mnożone razy ciepło właściwe i gęstość (odczytywane z pliku txt); następnie mnożone razy detJ, który policzony był wcześniej

-uworzenie wektora P

wyznaczenie po dwa punkty całkowania na ścianę; sprawdzanie warunków brzegowych i obliczenie funkcji kształtu; pomnożenie razy wyznacznik (bok/2), aby pozbyć się całki po powierzchni; pomnożenie przez współczynnik konwekcji i temperaturę otoczenia

- agregacja

Kiedy są policzone sktruktury lokalne dla każdego elementu, łączone są w całość do macierzy globalnej.

Dla macierzy H wygląda to tak:

```
for (int i = 0; i < num_elements; i++) {
    localH = matrixH_full(i+1);
    for (int j = 0; j < 4; j++) {
        for (int k = 0; k < 4; k++) {
            global_matrixH[arr_elem[i].id_node[j] - 1][arr_elem[i].id_node[k] - 1] += localH[j][k];
        }
    }
}
```

- sprowadzenie do układu równań

Po agregacji następuje sumowanie niektórych wartości

$$\left([H]+\frac{[C]}{\Delta\tau}\right)$$

powyższe wartości są dodawane i tworzą nową macierz; tau dane w pliku txt

$$\left(\frac{[C]}{\Delta\tau}\right)\{t_0\}+\{P\}$$

to samo tutaj, w tym miejscu otrzymywany jest nowy wektor; t0 jest dane w pliku txt

ostatecznie do rozwiązania pozostaje

[H']{T1} + {P'}=0

do którego używana jest motoda Gaussa, która zwraca nowy wektor temperatur

```
double* Grid::gauss( double ** HP, double * T){
    const double eps = 1e-12;
    int i, j, k;
    double m, s; // eliminacja współczynników
    for (i = 0; i < num_nodes - 1; i++){
        for (j = i + 1; j < num_nodes; j++){
            if (fabs(HP[i][i]) < eps) return NULL;
            m = -HP[j][i] / HP[i][i];
            for (k = i + 1; k <= num_nodes; k++)
                HP[j][k] += m * HP[i][k];
        }
    }// wyliczanie niewiadomych
    for (i = num_nodes - 1; i >= 0; i--){
        s = HP[i][num_nodes];
        for (j = num_nodes - 1; j >= i + 1; j--)
            s -= HP[i][j] * T[j];
        if (fabs(HP[i][i]) < eps) return NULL;
        T[i] = s / HP[i][i];
    }
    return T;
}
```

- wyniki dla Test Case

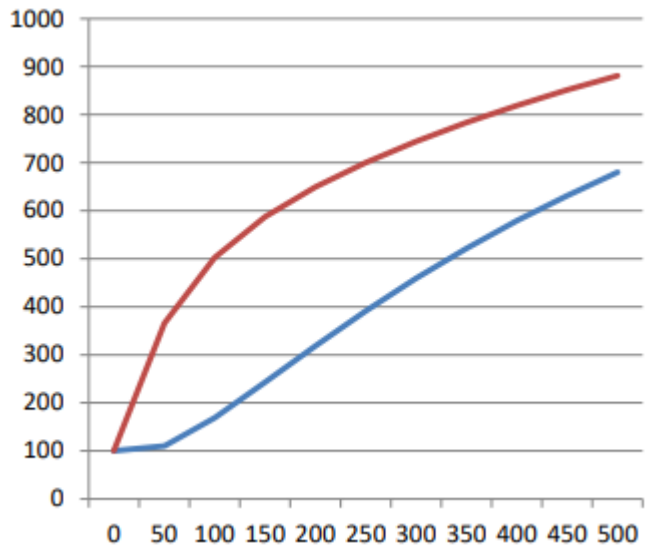
dane:

100 – initial temperature
500 – simulation time [s],
50 – simulation step time [s],
1200 – ambient temperature [C],
300 – alfa [W/m^{2}K],
0.100 – H [m],
0.100 – B [m],
4 – N_H,
4 – N_B,
700 – specific heat [J/(kg°C)],
25 – conductivity [W/(m°C)],
7800 – density [kg/m3].

wyniki:

```
Time[s] : 50
MIN: 110.038, MAX: 365.815

Time[s] : 100
MIN: 168.837, MAX: 502.592

Time[s] : 150
MIN: 242.801, MAX: 587.373

Time[s] : 200
MIN: 318.615, MAX: 649.387

Time[s] : 250
MIN: 391.256, MAX: 700.068

Time[s] : 300
MIN: 459.037, MAX: 744.063

Time[s] : 350
MIN: 521.586, MAX: 783.383

Time[s] : 400
MIN: 579.034, MAX: 818.992

Time[s] : 450
MIN: 631.689, MAX: 851.431

Time[s] : 500
MIN: 679.908, MAX: 881.058
```

zależność liniowa min od max:



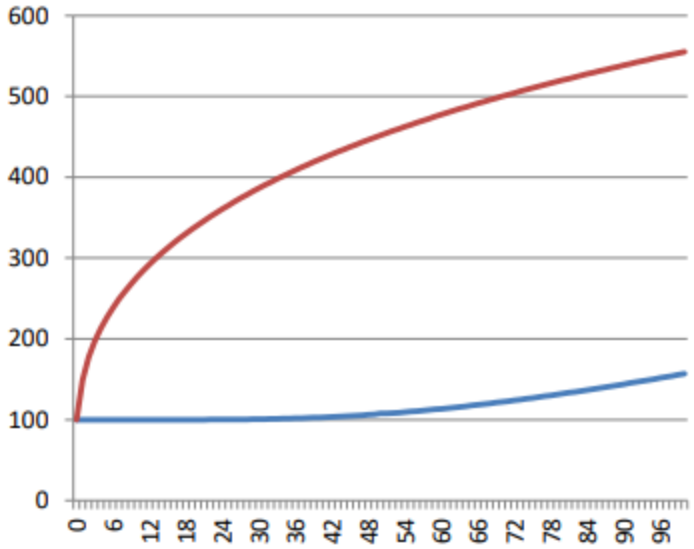
dane:

100 – initial temperature
100 – simulation time [s],
1 – simulation step time [s],
1200 – ambient temperature [C]
300 – alfa [W/m^2K],
0.100 – H [m],
0.100 – B [m],
31 – N_H,
31 – N_B,
700 – specific heat [J/(kg°C)],
25 – conductivity [W/(m°C)],
7800 – density [kg/m3].

wyniki:

```
Time[s] : 1
MIN: 100, MAX: 149.557

Time[s] : 2
MIN: 100, MAX: 177.445

Time[s] : 3
MIN: 100, MAX: 197.267

Time[s] : 4
MIN: 100, MAX: 213.153

Time[s] : 5
MIN: 100, MAX: 226.683

Time[s] : 6
MIN: 100, MAX: 238.607

Time[s] : 7
MIN: 100, MAX: 249.347

Time[s] : 8
MIN: 100, MAX: 259.165

Time[s] : 9
MIN: 100, MAX: 268.241

Time[s] : 10
MIN: 100, MAX: 276.701

Time[s] : 11
MIN: 100.001, MAX: 284.641

Time[s] : 12
MIN: 100.002, MAX: 292.134

Time[s] : 13
MIN: 100.003, MAX: 299.237

Time[s] : 14
MIN: 100.005, MAX: 305.997

Time[s] : 15
MIN: 100.009, MAX: 312.451

Time[s] : 16
MIN: 100.014, MAX: 318.631

Time[s] : 17
MIN: 100.021, MAX: 324.564

Time[s] : 18
MIN: 100.032, MAX: 330.271

Time[s] : 19
MIN: 100.046, MAX: 335.772

Time[s] : 20
MIN: 100.064, MAX: 341.085
```

zależność liniowa min od max:


600
500
400
300
200
100
0
0
6
12
18
24
30
36
42
48
54
60
66
72
78
84
90
96