Metoda Elementów Skończonych

Projekt - Nieustalona wymiana ciepła w układzie dwuwymiarowym na przykładzie okna drzwi piekarnika.

Mikołaj Stępniewski
286247, IS (WIMIP)

WSTĘP

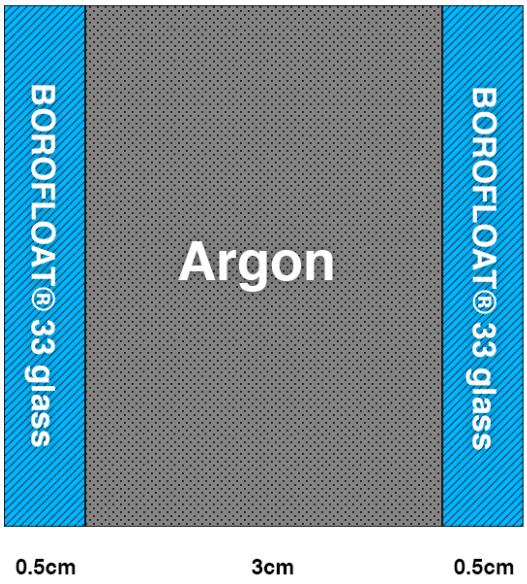
Program MES został użyty do przeprowadzenia symulacji ogrzewania okna drzwi piekarnika. Głównym celem było zbadanie jak zmienia się temperatura na zewnętrznej powierzchni okna - a tym samym jakie straty ciepła zachodzą podczas korzystania z piekarnika. W dalszej części przedstawione będą wykresy rozkładu ciepła w czasie oraz animacje, które w najlepszy sposób zobrazują rozkład gradientu temperatur w badanej strukturze.

ZAŁOŻENIA

Przyjąłem, że badane okno będzie składać się z trzech warstw (od lewej):

1. Borokrzemianowa płaska szyba (typu float) BOROFLOAT® 33 produkowana przez firmę Schott **o grubości 5 mm**.
2. Przestrzeń **o szerokości 3 cm**, wypełniona argonem.
3. Tak jak w punkcie 1.

Strukturę okna przedstawia poniższy rysunek:



Całość badanego obszaru ma wymiary 4 cm x 4 cm i jest reprezentowane przez siatkę MES o wielkości 41 x 41 węzłów (40 x 40 elementów), gdzie każdy element ma szerokość 1 mm. Ważne dla stabilności obliczeń było, aby każda z warstw składała się z conajmniej pięciu węzłów.

Parametry fizyczne oraz termiczne dla szkła BOROFLOAT® 33 zostały wzięte z oficjalnej strony producenta. Dla argonu posiłkowałem się arkuszem (*cieplo_sc.xls*) znalezionym w internecie. Pliki pdf pobrane ze strony schott.com oraz arkusz z parametrami argonu znajdują się w katalogu docs. Całość została przedstawiona w poniższej tabeli:

	Ciepło właściwe [J/kgK]	Współczynnik przewodzenia ciepła [W/mK]	Gęstość [kg/m3]
BOROFLOAT® 33	830	1,2	2230
Argon	520	0,017	1,7

Aby policzyć współczynnik konwekcji $\alpha \left[\frac{W}{m^2 K}\right]$ posłużyłem się wzorem znalezionym w "*Zbiór zadań z techniki cieplnej", Wydawnictwo Politechniki Warszawskiej rok. 1973:*

Powierzchnie wewnętrzne:
przy $t_{f_1} - \vartheta_1 < 5°C$

$$\alpha = 3{,}49 + 0{,}093(t_{f_1} - \vartheta_1)$$

przy $t_{f_1} - \vartheta_1 > 5°C$

$$\alpha = \varphi\sqrt[4]{t_{f_1} - \vartheta_1}$$

gdzie: t_{f_1} - temperatura otoczenia
ϑ_1 - temperatura powierzchni

przy czym: $\varphi = 2{,}32$ - dla powietrza w zamkniętym pomieszczeniu (po lewej stronie okna oraz przy wyłączonym termoobiegu)
$\varphi = 3{,}2$ - dla powietrza w pomieszczeniach produkcyjnych z wirującymi elementami maszyn.

Przyjąłem zatem, że dla włączonego termoobiegu:

$$\varphi = 3{,}75$$

Dla $\varphi = 2{,}32$ uzyskałem współczynnik konwekcji:

$$\alpha = \mathbf{3,49}$$

Dla $\varphi = 3{,}75$ uzyskałem współczynnik konwekcji:

$$\alpha = \mathbf{14,59}$$

W pliku GlobalData.swift, linijka 158:

```
1        static func calculateAlpha(t_surf:Double, t_ambient:Double, phi:Double? = nil) -> Double {
2            if t_ambient - t_surf < 5 {
3                return 3.49 + 0.093 * (t_ambient - t_surf)
4            } else {
5                return (phi ?? 2.32) * pow(t_ambient - t_surf, 0.25)
6            }
7        }
```

Powyższa metoda wywoływana jest przez każdy element znajdujący się na powierzchni. Dla reszty parametr *alpha* równy jest *nil.* Współczynnik ten będzie wykorzystywany podczas liczenia całki po powierzchni.

Ustaliłem, że po lewej stronie (strona pokoju) temperatura otoczenia wynosić będzie standardowo **21°C**, zaś po prawej stronie (wewnątrz piekarnika) **250°C**. Temperatura początkowa całości układu wynosić będzie **21°C** - wyrównanie z temperaturą pokojową.

Kolejnym elementem do policzenia jest krok czasowy, który powinien być dobrany pod względem badanego materiału w celu zachowania stabilności rozwiązania. Do obliczenia kroku czasowego posłużyłem się poniższym wzorem:

$$\Delta\tau = \frac{2c\rho(\frac{B}{nB})^2}{k}$$

gdzie: c - ciepło właściwe materiału
k - współczynnik przewodzenia ciepła
ρ - gęstość materiału

W pliku GlobalData.swift, linijka 148:

```
1        func setStableTimeStep(forMaterial material:ElementMaterial) {
2            let params = GlobalData.getParameters(for: material)
3            let k = params["k"] as! Double, c = params["c"] as! Double, ro = params["ro"] as! Double
4
5            let Asr = k/(c*ro)
6            self.d_tau = round(pow(B/Double(nB), 2.0)/(0.5*Asr))
7        }
```

Dla szyby $\Delta\tau$ wyniosło **3 sekundy** a dla argonu 2 sekundy. Uznałem, że mogę użyć kroku czasowego dla szyby, gdyż to ona odgrywa najważniejszą role w układzie - zwłaszcza, że różnica między krokiem czasowym dla szyby i argonu nie jest duża.

	Wielkość siatki MES	Temperatura początkowa układu	Temperatura otoczenia (strona lewa)	Temperatura otoczenia (strona prawa)	Współczynnik konwekcji (strona lewa)	Współczynnik konwekcji (strona prawa)	Krok czasowy
termoobieg	41 x 41 węzłów	21°C	21°C	250°C	3,49 W/m2K	14,59 W/ m2K	3 sekundy
brak termoobiegu	⬆	⬆	⬆	⬆	⬆	**3,49 W/ m2K**	⬆

KRÓTKI WSTĘP TEORETYCZNY

Rozwiązanie zagadnienia nieustalonej wymiany ciepła w układzie dwuwymiarowym sprowadza się do uzyskania układu równań MES na podstawie poniższego wzoru:

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

gdzie poszczególne wyrażenia obliczono za pomocą równań:

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

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

$$[C]=\int_V c\rho\{N\}\{N\}^T dV$$

gdzie: $\{t1\}$ - wektor szukanych temperatur węzłowych
$\{t0\}$ - wektor temperatur węzłowych na początku danego kroku czasowego
V - objętość
k - współczynnik przewodzenia ciepła
$\{N\}$ - wektor kolejnych węzłowych funkcji kształtu (określone lokalnie)
x, y - globalne współrzędne
T - oznacza wektor transponowany
α - współczynnik konwekcji
S - powierzchnia
t_∞ - temperatura otoczenia
c - ciepło właściwe
ρ - gęstość

WAŻNE!

W programie przez $[H]$ rozumiemy $\left([H]+\frac{[C]}{\Delta\tau}\right)$ a przez $\{P\}$ rozumiemy $\left(\frac{[C]}{\Delta\tau}\right)\{t_0\}+\{P\}$.

Temperatury na początku każdego kroku czasowego wyznaczane są za pomocą interpolacji w danym punkcie całkowania:

$$t_0=\sum_{i=1}^{4} N_{pc_i} t_{start_i}$$

gdzie: N_{pc} - zbiór funkcji kształtu dla danego punktu całkowania
t_{start} - zbiór temperatur węzłowych na początku kroku czasowego

Funkcje kształtu dla układu 2D wyglądają następująco:

$$N_1 = \frac{1}{4}\left(1-\xi\right)\left(1-\eta\right), \quad N_2 = \frac{1}{4}\left(1+\xi\right)\left(1-\eta\right), \quad N_3 = \frac{1}{4}\left(1+\xi\right)\left(1+\eta\right), \quad N_4 = \frac{1}{4}\left(1-\xi\right)\left(1+\eta\right)$$

Natomiast punkty całkowania Gaussa (2p):

pc	Ksi	Eta	Waga
1	$-1/\sqrt{3}$	$-1/\sqrt{3}$	1
2	$1/\sqrt{3}$	$-1/\sqrt{3}$	1
3	$1/\sqrt{3}$	$1/\sqrt{3}$	1
4	$-1/\sqrt{3}$	$1/\sqrt{3}$	1

Całkowanie za pomocą kwadratury Gaussa przeprowadzane jest na podstawie wzoru:

$$f(x) = \sum_{i=1}^{npc} \left(f(x_{pc_i}) * w_i\right) * detJ$$

gdzie: $f(x_{pc_i})$ - wartość funkcji całkowanej dla zinterpolowanego x

ALGORYTM I IMPLEMENTACJA

Wymieniony w powyższej sekcji układ równań musimy obliczyć, dla każdego kroku czasowego a następnie rozwiązać go za pomocą metody eliminacji Gaussa (w pliku *Solver.swift*). Natomiast dla uproszczenia, przedstawię algorytm obliczania macierzy $[H]$ wraz z wektorem $\{P\}$ dla danego kroku czasowego (metoda *compute* w pliku *GlobalData.swift*):

Dla każdego elementu w siatce:

1. Wyzeruj $[H]$ oraz $\{P\}$.
2. Dla każdego punktu całkowania:
   2.1.Oblicz macierz Jakobiego 2D.
   2.2.Oblicz wyznacznik macierzy Jakobiego 2D.
   2.3.Wyzeruj t_0.
   2.4.Oblicz $\frac{\delta N}{\delta x}, \frac{\delta N}{\delta y}$ oraz t_0.
   2.5.Oblicz i dołącz elementy macierzy $[H]$ oraz wektora $\{P\}$ (całka po V).
3. Dla każdej powierzchni zewnętrznej siatki:
   3.1.Oblicz wyznacznik macierzy Jakobiego 1D.
   3.2.Dla obu węzłów w powierzchni:
      3.2.1.Oblicz i dołącz elementy macierzy $[H]$ oraz wektora $\{P\}$ (całka po S).
4. Dokonaj agregacji lokalnej macierzy $[H]$ oraz lokalnego wektora $\{P\}$ do ich globalnych odpowiedników.

Punkt 2. z powyższego algorytmu został zaimplementowany w następujący sposób:

2.1., 2.2., 2.3. Oblicz macierz Jakobiego 2D, oblicz wyznacznik tej macierzy oraz wyzeruj t0
W pliku GlobalData, linijka 279:

```
1    // ipi - index of a current integration point
2    let jacobian = Jacobian(intPoint: ipi, xs: coordsX, ys: coordsY, dNdKsis:
     iPoints_dNdKsi, dNdEtas: iPoints_dNdEta)
3    detJ = abs(jacobian.det)
4    let dNdKsi = iPoints_dNdKsi[ipi], dNdEta = iPoints_dNdEta[ipi]
5    t0 = 0.0
```

2.4. Oblicz $\frac{\delta N}{\delta x}$, $\frac{\delta N}{\delta y}$ oraz t_0.
W pliku GlobalData, linijka 285:

```
1    for i in 0 ..< 4 {
2        dNdx[i] = jacobian.matrixInverted[0][0] * dNdKsi[i] + jacobian.matrixInverted[0]
     [1] * dNdEta[i]
3
4        dNdy[i] = jacobian.matrixInverted[1][0] * dNdKsi[i] + jacobian.matrixInverted[1]
     [1] * dNdEta[i]
5
6        //MARK: - Interpolating temperature of current IP, from every other element's IPs.
7        t0 += temperatures_start[i] * shapeFunctionsVals[ipi][i]
8    }
```

2.5. Oblicz i dołącz elementy macierzy $[H]$ oraz wektora $\{P\}$ (całka po V)
W pliku GlobalData, linijka 297:

```
1    for i in 0..<4 {
2        for j in 0..<4 {
3            C_ij = c * ro * shapeFunctionsVals[ipi][i] * shapeFunctionsVals[ipi][j] * detJ
4            H_current[i][j] += k * (dNdx[i] * dNdx[j] + dNdy[i] * dNdy[j]) * detJ + (C_ij /
     d_tau)
5            P_current[i] += (C_ij / d_tau) * t0
6        }
7    }
```

3.1. Oblicz wyznacznik macierzy Jakobiego 1D ($detJ = \frac{x_2 - x_1}{2}$).
W pliku GlobalData, linijka 314:

```
1    detJ = 0.5 * sqrt(pow(surface.ND[0].x - surface.ND[1].x, 2) +
2                      pow(surface.ND[0].y - surface.ND[1].y, 2))
```

3.2.1. Oblicz i dołącz elementy macierzy $[H]$ oraz wektora $\{P\}$ (całka po S).
W pliku GlobalData, linijka 318:

```
let shapeFunc = localElement.localSF[siid].shapeFunctionsVals
for i in 0..<2 {
    for j in 0..<4 {
        for k in 0..<4 {
            H_current[j][k] += alpha * shapeFunc![i][j] * shapeFunc![i][k] * detJ
        }
        P_current[j] += alpha * t_ambient * shapeFunc![i][j] * detJ
    }
}
```

4. Dokonaj agregacji lokalnej macierzy $[H]$ oraz lokalnego wektora $\{P\}$ do ich globalnych odpowiedników.
W pliku GlobalData, linijka 330:

```
for i in 0..<4 {
    let first = element.ND[i].iid
    for j in 0..<4 {
        let next = element.ND[j].iid
        H_global[first][next] += H_current[i][j]
    }
    P_global[first] += P_current[i]
}
```

Oprócz tego, obliczanie macierzy Jakobiego zostało zaimplementowane na podstawie obliczeń z pliku *Jakobian.pdf* na stronie prowadzącego i wygląda w następujący sposób:
W pliku Jakobian.swift, linijka 35:

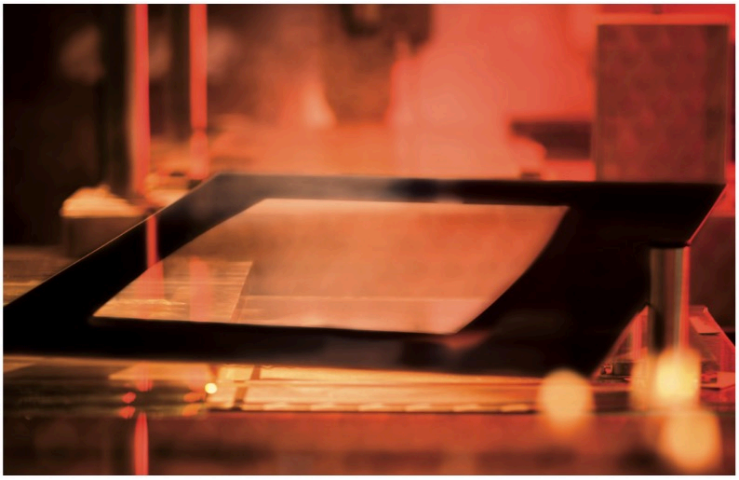
```
for i in 0..<4 {
    matrix[0][0] += self.points_dNdKsi[intPoint][i] * xs[i]  // dxdKsi
    matrix[0][1] += self.points_dNdKsi[intPoint][i] * ys[i]  // dydKsi
    matrix[1][0] += self.points_dNdEta[intPoint][i] * xs[i]  // dxdEta
    matrix[1][1] += self.points_dNdEta[intPoint][i] * ys[i]  // dydEta
}
```

Macierz Jakobiego w matematycznym zapisie:

$$\begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial y}{\partial \xi} \\ -\frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \xi} \end{bmatrix}$$

Wszelkie całkowanie w programie jest przeprowadzane za pomocą wspomnianej już kwadratury Gaussa.

WYNIKI



Thermally resistant oven door made with BOROFLOAT® 33.