EP3 - Modelagem de um Sistema de Resfriamento de Chips

MAP3121- Métodos Numéricos e Aplicações

Arthur Pedroso Porto Belli - 11804608
Letícia Harumi Furusawa - 11965585

14 de julho de 2022

1 Introdução

Este relatório descreve os resultados obtidos no exercício programa, no qual estudamos um modelo simplificado de representação da evolução do calor em um chip praticamente plano. Para tal, serão utilizados componentes dos Exercícios-Programas passados.

O grupo optou pela implementação em Python 3.9.7 e utilizou as bibliotecas Numpy 1.21.1, MatPlotLib 3.4.2, além de uma coletânea de funções próprias (*utils.py*).

2 Exercícios propostos

2.1 Determinação das expressões de integração

Dado um chip de comprimento L e n pontos de apoio, teremos um passo $h = L/(n+1)$ para n *splines* que comporão nossa base de vetores que gera o espaço das soluções do método dos elementos finitos. Esses pontos de apoio serão denotados $\bar{x_i}$ quando não estiverem normalizados em $[0, 1]$ e x_i quando estiverem normalizados.

Do enunciado, queremos resolver o seguinte sistema linear:

$$\begin{pmatrix} < \phi_1, \phi_1 >_L & < \phi_1, \phi_2 >_L & \cdots & < \phi_1, \phi_n >_L \\ < \phi_2, \phi_1 >_L & < \phi_2, \phi_2 >_L & \cdots & < \phi_2, \phi_n >_L \\ \vdots & \vdots & \ddots & \vdots \\ < \phi_n, \phi_1 >_L & < \phi_n, \phi_2 >_L & \cdots & < \phi_n, \phi_n >_L \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} < f, \bar{\phi_1} > \\ < f, \bar{\phi_2} > \\ \vdots \\ < f, \bar{\phi_n} > \end{pmatrix}$$

De início, sabemos que $< \phi_i(x), \phi_j(x) >_L = 0$ para $|i - j| > 1$ uma vez que são ortogonais, de forma que nos resta uma matriz tridiagonal a ser resolvida. Ao estudarmos as expressões da diagonal principal, obtemos a seguinte relação (3.1, 3.2):

$$< \phi_i(x), \phi_i(x) >_L = \int_0^1 k(x)\phi_i^{'}(x)\phi_i^{'}(x)dx = \int_0^1 k(x)\phi_i^{'}(x)^2 dx$$

. Como a função $\phi_i(x)$ tem o seguinte comportamento:

$$\phi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h}, \ x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{h}, \ x \in [x_i, x_{i+1}] \\ 0, \ \text{para valores fora de } [x_{i-1}, x_{i+1}] \end{cases}$$

Sabemos que sua derivada se comporta de forma análoga:[1]

$$\phi_i^{'}(x) = \begin{cases} \frac{1}{h}, \ x \in [x_{i-1}, x_i] \\ -\frac{1}{h}, \ x \in [x_i, x_{i+1}] \\ 0, \ \text{para valores fora de } [x_{i-1}, x_{i+1}] \end{cases}$$

[1] Analogamente à notação utilizada para os pontos de apoio, os splines também receberão uma barra quando não estiverem normalizados

Portanto, os elementos da diagonal principal serão dados por

$$\boxed{< \phi_i, \phi_i >_L = \int_{x_{i-1}}^{x_i} k(x)\left(\frac{1}{h}\right)^2 dx + \int_{x_i}^{x_{i+1}} k(x)\left(-\frac{1}{h}\right)^2 dx = \int_{x_{i-1}}^{x_{i+1}} \frac{k(x)}{h^2} dx}$$

Além disso, temos que, pela linearidade do produto interno, as diagonais superior e inferior serão dadas por:

$$\boxed{< \phi_i, \phi_{i+1} >_L = < \phi_{i+1}, \phi_i >_L = \int_{x_i}^{x_{i+1}} -\frac{k(x)}{h^2 L} dx}$$

Por fim, $< f, \phi_i >$ é a mais trivial das expressões, uma vez que não exige nenhuma manipulação matemática, apenas sua codificação direta no programa:

$$\boxed{< f, \bar{\phi_i} > = \int_{x_{\bar{i-1}}}^{x_{\bar{i}}} f(x)\bar{\phi_i}(x) dx + \int_{x_{\bar{i}}}^{x_{\bar{i+1}}} f(x)\bar{\phi_i}(x) dx}$$

.

Com todos os valores constituintes do sistema linear obtidos, utilizaremos funções do EP1 para resolver esse sistema pelo método de decomposição LU e obter o vetor resposta $\bar{\alpha}$. Com as soluções do sistema, é possível compor a função que será retornada pelo método dos elementos finitos (3.3):

$$v(x) = \sum_{i=1}^{n} \alpha_i \bar{\phi_i}(x)$$

Com a resposta em mãos, podemos adaptá-la para condições de contorno não homogêneas:[2]

$$\boxed{u(x) = v(x) + a + \frac{(b-a)x}{L}}$$

[2]Importante notar que todas as integrais desse Exercício-Programa estão sendo realizadas com as fórmulas de integração de Gauss para dois pontos e dois pesos, uma vez que nosso splines são lineares e dados por pares de pontos.

3 Funções utilizadas

3.1 Funções do EP1

3.1.1 A_to_LU_tridig

Recebe como parâmetros os vetores a, b, c que armazenam as diagonais da matriz tridiagonal e retorna os vetores $U_{ii}, U_{i+1} e\ L_{i+1}$.

```
def A_to_LU_tridig(a, b, c): #recebe as diagonais
    dim = b.shape[0] #n mero de elementos na diagonal principal    a dimmens o da
                                                    matriz
    l_ip1 = np.zeros(dim, dtype=np.double) #l_i+1
    u_ii = np.zeros(dim, dtype=np.double) #u_ii
    u_ip1 = c.copy() #u_i+1

    u_ii[0] = b[0]
    for i in range(1, dim):
        l_ip1[i] = a[i]/u_ii[i-1] #multiplicadores
        u_ii[i] = b[i]-l_ip1[i]*c[i-1]

    return u_ii, u_ip1, l_ip1
```

3.1.2 solve_lin_sys_tridiag

Resolve sistemas de equações lineares ridiagonais usando a decomposição LU. Recebe como parâmetros os vetores U_{ii}, U_{i+1} e L_{i+1} (valores em forma vetorial armazenamento otimizado) e retorna os valores que resolvem o sistema no vetor x.

```
def solve_lin_sys_tridiag(u, u_ip1, l, d):
    #Ly = d
    dim = u.shape[0]
    y = np.zeros((dim, 1), dtype=np.double) #vetor coluna
    y[0][0] = d[0]

    for i in range(1, dim):
        y[i][0] = d[i] - l[i]*y[i-1][0]

    #Ux = y
    x = np.zeros((dim, 1), dtype=np.double) #vetor coluna
    x[dim-1] = y[dim-1][0]/u[dim-1]

    for i in range(dim-1, -1, -1):
        x[i-1] = (y[i-1][0]-u_ip1[i-1]*x[i])/u[i-1]

    return x
```

3.2 Funções do EP2

3.2.1 x_w(n)

Dada a natureza linear dos *splines* utilizados no EP3, foi necessário incluir os valores de nós e pesos para a fórmula de integração de Gauss. Como só utilizamos esses valores, os outros valores utilizados no último exercício-programa foram removidos.

```
def x_w(n):
    if n == 2:
        x = [-0.57735026918962576450914878, 0.57735026918962576450914878]
        w = [1.0, 1.0]
    return x, w
```

3.2.2 gauss(f, lim_inf, lim_sup, n, nós, pesos)

Função adaptada do EP2 para resolução de integrais simples, ao invés de duplas. Por motivos de concisão, optou-se por essa solução, mas seria possível resolver o presente exercício-programa utilizando

a função gauss_2 desenvolvida no último exercício, a principal diferença seria a necessidade de fixar os extermos de integração externos em 0 e 1, a fim de garantir a igualdade entre a integral simples e a dupla.

```
def gauss(f, lim_inf, lim_sup, n, nos, pesos):
    '''
    integra  o simples no intervalo [a,b] da fun  o f(x) com n n s (x) e pesos (w
                                                    )
    '''
    linear_scaling, linear_transposition = find_lin_scale_transp(lim_inf, lim_sup)
    sum = 0
    v = [linear_scaling*pesos[i] for i in range(n)] # pesos adaptados
    y = [(linear_scaling*nos[i]+linear_transposition) for i in range(n)] # n s
                                                    adaptados
    for i in range (n): # soma iterada da integral
        sum += v[i]*f(y[i])

    return sum
```

3.3 Novas funções

3.3.1 phi(x, apoios, h, i)

Definição genérica dos vetores da base do espaço dos resultados do método dos elementos finitos.

```
def phi(x, apoios, h, i):
    if apoios[i-1] <= x <= apoios[i]: #primeira metade do spline    crescente
        return (x-apoios[i-1])/h
    elif apoios[i] <= x <= apoios[i+1]: #segunda metade do spline    decrescente
        return (apoios[i+1]-x)/h
    return 0 # fora do intervalo
```

3.3.2 fem (n, L, k, f)

Responsável pela montagem do sistema linear a ser resolvido e composição do vetor resposta normalizado $u(x)$

```
def fem(n, L, k, f):
    '''
    retorna a solucao do problema normalizado em x in [0,1]

    n -> n mero de apoios
    L -> comprimento do chip
    k -> condutividade t rmica do material
    f -> fun  o de excita  o do sistema
    '''
    h = L/(n+1) #passo dos apoios
    apoios = [i*h for i in range(n+2)] #apoios simples
    apoios_barr = [apoios[i]/L for i in range(n+2)] #apoios normalizados
    #defini  o da matriz normal do MMQ

    #diagonal inferior
    a = [0 if i == 0 else (-1)*(1/(L*(h**2)))*gauss(k, apoios_barr[i], apoios_barr[i+1
                                                    ], 2, *x_w(2)) for i in range(n)]
    a = np.array(a, dtype=np.double)

    #diagonal principal
    b = [(1/(L*(h**2)))*(gauss(k, apoios_barr[i-1], apoios_barr[i], 2, *x_w(2)) +
                                                    gauss(k, apoios_barr[i], apoios_barr[i+
                                                    1], 2, *x_w(2))) for i in range(1, n+1)
                                                    ]
    b = np.array(b, dtype=np.double)

    #diagonal superior
    c = [0 if i == n-1 else (-1)*(1/(L*(h**2)))*gauss(k, apoios_barr[i], apoios_barr[i
                                                    +1], 2, *x_w(2)) for i in range(n)]
    c = np.array(c, dtype=np.double)
```

```
    #array de termos independentes
    d = [(gauss((lambda nos: f(nos)*phi(nos, apoios, h, i)), apoios[i-1], apoios[i], 2
                                                , *x_w(2)) + gauss((lambda nos: f(nos)*
                                                phi(nos, apoios, h, i)), apoios[i],
                                                apoios[i+1], 2, *x_w(2))) for i in
                                                range(1, n+1)]

    #resolu  o do sistema linear
    alphas = solve_lin_sys_tridiag(*A_to_LU_tridig(a, b, c), d)

    eixox = np.linspace(0, L, 100)
    v = [0]*len(eixox)
    for k in range(len(eixox)):
        v[k] = calc_v(alphas, eixox[k], apoios, h, n)
    return v #vetor solu  o normalizado
```

3.3.3 calc_v(alphas, x, apoios, h, n)

Resolve (3.3)

```
def calc_v(alphas, x, apoios, h, n):
    temp = float(sum([alphas[i-1]*phi(x, apoios, h, i) for i in range(1, n+1)]))
    return temp
```

3.3.4 solve_generic(n, L, k, f, a, b)

Chama a função fem para obter a resposta normalizada e adapta para condições de contorno não-homogêneas.

```
def solve_generic(n, L, k, f, a, b): # calcula a temperatura pela equa  o f dada
    '''
    n -> n mero de apoios a serem utilizados
    L -> comprimento do chip
    k -> condutividade t rmica do materia
    f -> fun  o de excita  o t rmica do chip
    a -> condi  o de contorno no extremo 0
    b -> condi  o de contorno no extremo L
    '''

    h = L/(n+1)
    eixox = np.linspace(0, L, 100)
    v = fem(n, L, k, f)
    u = [0]*len(eixox)
    for i in range(len(eixox)):
        u[i] = (v[i] + a + (b-a)*eixox[i]/L)
    return u #vetor solucao individualizado
```

3.3.5 erro_max(n, L, u_teor, u_obtido)

Calcula o erro máximo do método dos elementos finitos em relação à solução analítica para um dado par de valores n e L.

```
def erro_max(n, L, u_teor, u_fem): # devolve o erro maximo entre o valor encontrado
                                                pelo fem e o teorico
    erro = u_teor - u_fem
    erro_m = np.max(erro)

    return erro_m
```

3.3.6 que(x, Qp0, Qm0, sigma, theta, L)

Função que calcula a forçante total sobre o sistema, como definido na secção 4.3 do enunciado.

```
def que(x, Qp0=600, Qm0=300, sigma=0.1, theta=1, L=1):
    q = Qp0*np.exp(-(x-L/2)**2/sigma**2) - Qm0*(np.exp(-(x)**2/theta**2) + np.exp(-((x
                                                    -L)**2/theta**2)))
    return q
```

3.3.7 mat_varia(q,L,n)

Calcula a variação da temperatura através do método dos elementos finitos quando há varaiação do material que constitui o chip.

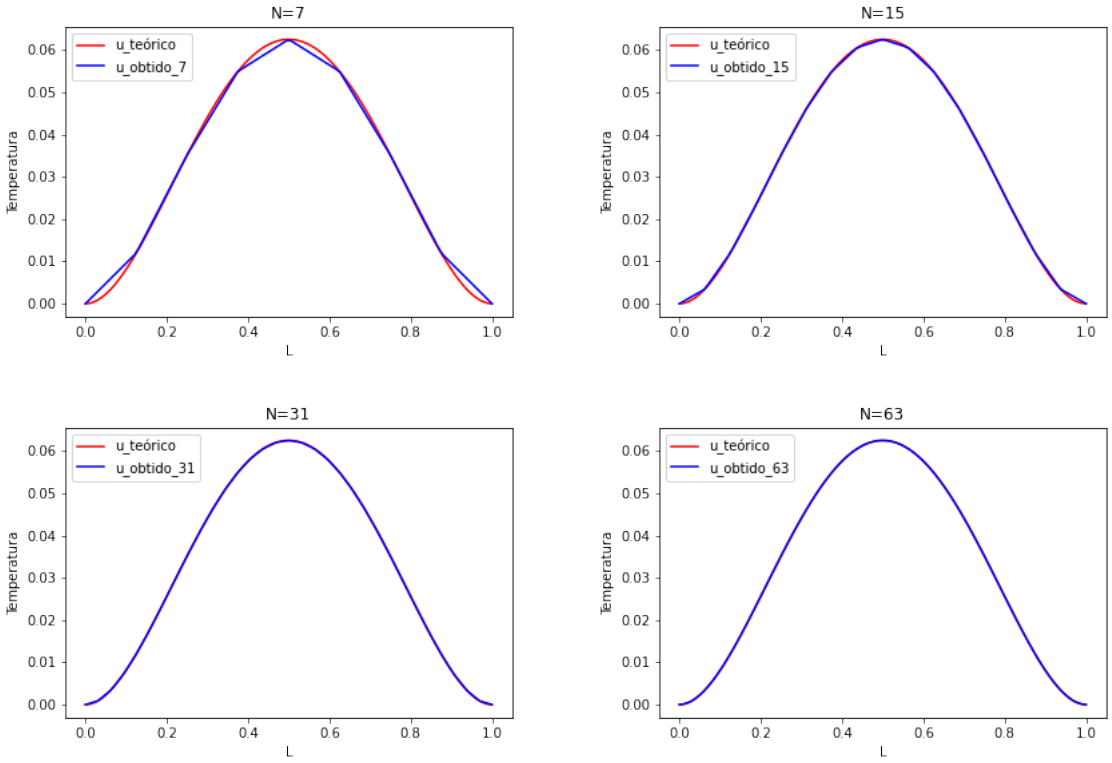
```
def mat_varia(q,L,n,):

  material21 = solve_generic(n, L, k_mix, q, 0, 0)
  return material21
```

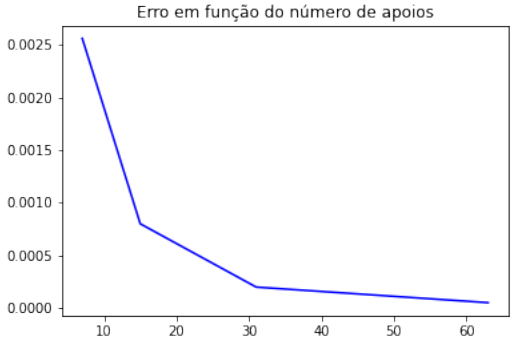
4 Resultados

4.1 Validação

Para as condições fornecidas no enunciado: $k(x) = 1$, $q(x) = 0$, $a = b = 0$, $L = 1$ e $f(x) = 12x(1-x) - 2$. Obteve-se as seguintes figuras comparativas entre a resposta esperada e a obtida:



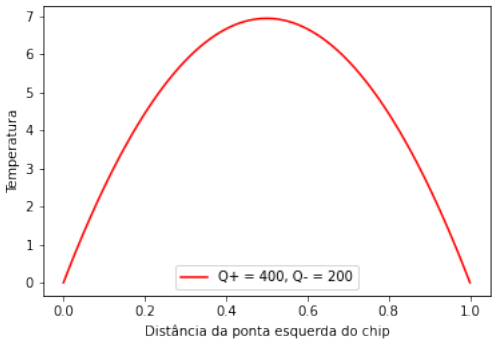
Uma vez obtidas as figuras, construiu-se também uma figura da evolução do erro em função do número de apoios (n) utilizados no método dos elementos finitos (fem).

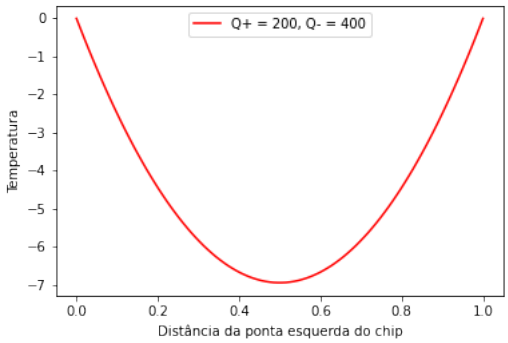


Vale notar que a curva de erro em função de n tem corportamento proporcional a $1/n^2$. Isso vai de encontro com a análise proposta na secção 4.2 do enunciado, uma vez que espera-se que o erro seja uma função de h^2. Como $h = \frac{L}{n+1}$, temos que o erro será proporcional à $1/n^2$, com excessão de constantes multiplicativas e e lineares. Assim, atesta-se o bom funcionamento do programa desenvolvido pela dupla.

4.2 Equilíbrio com forçantes de calor

Para estudarmos a influência das forçantes de calor no problema, vamos começar do caso mais básico: tanto a injeção como a retirada de calor do chip são constantes em toda sua extensão. Para tal, utilizaremos os seguintes parâmetros: $Q_+^0 = 400$, $Q_-^0 = 200$, $n = 50$, $L = 1$, $k = 3.6$ e condições de fronteira homogêneas.





Percebe-se na figura a direita a implicação da inversão dos papeis na resolução do problema. Ao retirarmos mais calor, espera-se que a temperatura decresça, enquanto no caso em que $Q_+^0 > Q_-^0$, temos que as temperaturas serão positivas, uma vez que temos mais calor sendo injetado no sistema do que retirado.

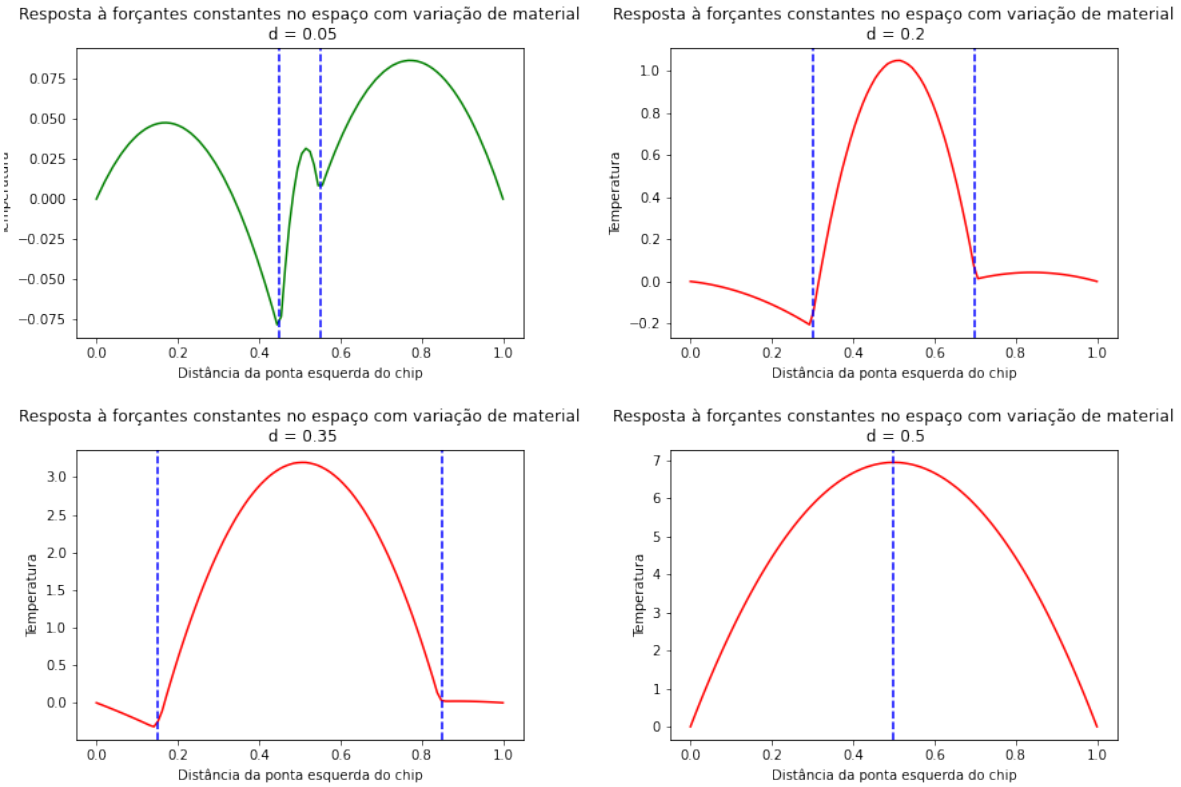
A figura a cima foi calculada pela fórmula de forçantes fornecida na secção 4.3 do enunciado, utilizando-se $\sigma = 0.1$ e $\theta = 1$ como parâmetros adicionais. O grupo também observou, ao fazer testes com o programa, que conforme o parâmetro $k(x)$ era variado, a curva de resposta resistia mais a fazer grandes excursões de temperatura.

4.3 Equilíbrio com variação de material

No estudo do caso em que o chip apresenta mais de um material, utilizamos uma outra implementação da função $k(x)$ que seleciona de acordo com a posição o coeficiente do material selecionado.

```
def k_mix(x, L, d): #k(x) que resolve exerc cio 4.4
    if (0 <= x <= L/2 - d):
        return 60 #aluminio
    elif (L/2 - d <= x <= L/2 + d):
        return 3.6 #silicio
    elif (L/2 + d <= x <= L):
        return 60 #aluminio
```

Ao conversar com colegas de curso, o grupo entendeu que a solução obtida não foi a mais adequada, pois como é possível observar pelas imagens a seguir, para diferentes valores de d, surgem diversas descontinuidades que não deveriam existir em um problema contínuo. Por mais que tenhamos interfaces entre os materiais, é de se esperar que haja algum tipo abrandamento da diferença das temperaturas na borda, coisa que não foi observada na soluçao do grupo.



O único resultado compatível com a teoria foi o caso de $d = 0.5$ que ocorre quando não há alumínio na composição da carcaça do chip.

5 Conclusões

Através deste exercício programa, conseguimos entender melhor o métodos dos elementos finitos e enxergar uma aplicação prática real de tal método. Também, aprendemos e vimos em prática o efeito do aumento dos números de apoios na melhora da aproximação da função calculada pelo método do elementos finitos em relação à função esperada, assim como a diminuição do erro. Para o equilíbrio com forçantes de calor, obtivemos resultados muito bons, já no equilíbrio com variação de material, obtivemos resultados diferentes do esperado. Foi muito interessante estudar a aplicação prática do método dos elementos finitos, especialmente neste caso de aquecimento e resfriamento de chips.