Exercício de programação 3 - Cálculo Numérico

Modelagem de um Sistema de Resfriamento de Chips

Gabriel Souza Lima - 11820106

Lucas Pereira da Fonseca - 11808126

1 Introdução

Neste exercício de programação, nos foi proposto modelar o comportamento da difusão térmica que ocorre em um processador ou chip de computador de tamanho L × L e altura h ao usarmos um resfriador ("cooler" ou placa fria) colado na parte superior do bloco do chip.

É possível obter a distribuição de calor no interior do chip utilizando a equação do calor, obtida à partir da Lei de Fourier, e da propriedade de conservação da energia. Para o caso unidimensional, temos:

$$\rho C \frac{\partial T(t,x)}{\partial t} = \frac{\partial}{\partial x}\left(k(x)\frac{\partial T(x)}{\partial x}\right) + Q(t,x)$$

onde

- $T(x,y)$ é a temperatura do chip na posição x e no instante t.
- ρ é a densidade do material do chip
- C é o calor específico do material
- κ representa a condutividade térmica do material
- Q é a soma do calor gerado pelo chip com o calor retirado pelo resfriador

2 Estado Estacionário

Para simplificar, nós assumimos que o resfriador extraia a mesma quantidade de calor gerado pelo chip. Ou seja:

$$\frac{\partial T(t,x)}{\partial t} = 0$$

substituindo este resultado na equação anterior, obtemos

$$-\frac{\partial}{\partial x}\left(k(x)\frac{\partial T(x)}{\partial x}\right) = Q(x)$$

3 Método de Elementos Finitos

3.1 Escolha do Espaço U_n e sua base: elementos finitos

O espaço sobre o qual iremos realizar as operações acima é U_n, definido por Splines Lineares uniformemente espalhados em $[0,1]$. O espaçamento entre dois nós consecutivos é constante e igual a $h = 1/(n+1)$, e os nós são definidos como $x_i = ih$ com $0 \leq i \leq n+1$. Cada spline é definido como uma função contínua em $[0,1]$ reta que liga dois nós consecutivos e que se anula nos extremos, ou seja:

$$S^0_{2,n}[0,1] = \{s(x) \in C[0,1] : s(0) = s(1) = 0 \; e \; s|_{x_i,x_{i+1}} \in P_1\}$$

Da definição acima, sabemos que cada spline ligará dois nós consecutivos com uma reta. Assim, tanto o nó x_0 quanto o x_{n+1} estarão contidos em um spline, enquanto que os nós $x_i, \; i \neq 0, n+1$ estarão contidos em dois splines diferentes. Logo, o número de splines será dado por $2 \times (n+1) \; - \; 2 \; = \; n$. Percebemos, então, que $S^0_{2,n}[0,1]$ é um espaço vetorial de dimensão n. Podemos definir uma base para o espaço de splines dado por funções ϕ_i tais que $\phi_i = (x - x_{i-1})/h$ se $x \in [x_{i-1}, x_i]$, $\phi_i = (x_{i+1} - x)/h$ se $x \in [x_i, x_{i+1}]$ e $\phi_i = 0$ se $x \notin [x_{i-1}, x_{i+1}]$. Facilmente podemos perceber que a intersecção entre ϕ_i e ϕ_j será vazia se, e somente se, $|i-j| > 1$. Ou seja, $\langle \phi_i, \phi_j \rangle_L = 0$, se $|i-j| > 1$. Decorre que a matriz do sistema linear seja tridiagonal.

3.2 Montagem do sistema e solução da matriz

Para montar o sistema, primeiro nós devemos calcular $\langle \phi_i, \phi_j \rangle_L$, para $|i-j| \leq 1, \langle \phi_i, \phi_i \rangle_L$, e $\langle f, \phi_i \rangle$, onde $\langle u, v \rangle_L = \int_0^1 [k(x)u'(x)v'(x) + q(x)u(x)v(x)]dx$ e $\langle u, v \rangle = \int_0^1 u(x)v(x)$. Para isso, nós usaremos o código desenvolvido no Exercício Programa 2, para aproximar integrais através da fórmula de Gauss. Após montar os vetores das diagonais, nós usaremos as funções desenvolvidas no Exercício Programa 1 para resolver sistemas tridiagonais.

3.3 Solução do método de elementos finitos

Após concluir o passo 3.2, nós teremos obtido as constantes α_i, tais que $\overline{u}_n(x) = \sum_{i=1}^n \alpha_i \phi_i(x)$ que melhor aproxima a solução $u(x)$ da equação do calor para o regime permanente.

3.4 Condições de fronteira não homogêneas

Para o caso em que $u(0) = a$ e $u(1) = b$, utiliza-se a estratégia de transformar o problema no caso homogêneo ao montar a equação:

$$L(v(x)) = f(x) + (b-a)k'(x) - q(x)(a + (b-a)x), \; v(0) = v(1) = 0 \qquad (2)$$

A solução da equação do calor com condições de contorno $u(0) = a$ e $u(1) = b$ é $u(x) = v(x) + a + (b-a)x$.

Demonstração: Seja $u(x)$ a solução da equação (1) com $u(0) = a$ e $u(1) = b$. Vamos substituir $u(x)$ por $u(x) = v(x) + a + (b-a)x$, tal que $v(x)$ seja solução de (2). Ficamos com:

$$L(u(x)) = L(v(x) + a + (b-a)(x)) = \{-k(x)\;[v(x) + a + (b-a)x]'\;\}' + q(x)[v(x) + a + (b-a)x]$$

Agora, vamos separar os termos:

$$L(u(x)) = [-k(x)v'(x)]' + q(x)v(x) + [-k(x)(b-a)]' + q(x)[a + (b-a)x]$$

Ora, mas já sabemos que $[-k(x)v'(x)]' + q(x)v(x) = f(x)$. Segue, então, que

$$L(u(x)) = f(x) - k'(x)(b-a) + q(x)(a + (b-a)x) = \tilde{f}(x)$$

Como pode-se ver, chegamos à equação cuja solução nos dará $u(x)$.

Além disso, quando $k(x)' = 0$ e $q(x) = 0$, ficamos com $L(u(x)) = \tilde{f(x)} = f(x)$ e $u(x) = v(x) + a + (b-a)x$.

3.5 Intervalo [0,L]

Para os casos em que o intervalo da equação diferencial for $[0,L]$, o espaçamento dos splines lineares será $h = L/(n+1)$ e $x_i = ih$ com $0 \leq i \leq n+1$. Além disso, a definição do Espaço de Splines será

$$S^0_{2,n}[0,L] = \{s(x) \in C[0,L] : s(0) = s(L) = 0 \; e \; s|_{x_i,x_{i+1}} \in P_1\}$$

Além disso, $\langle u, v \rangle_L = \int_0^L [k(x)u'(x)v'(x) + q(x)u(x)v(x)]dx$ e $\langle u, v \rangle = \int_0^L u(x)v(x)$.

Para o caso em que temos condições de fronteira não homogêneas, basta resolvermos o caso:

$$L(v(x)) = f(x) + (b-a)k'(x) - q(x)(a + (b-a)x) = \tilde{f}(x), \; v(0) = v(1) = 0 \qquad (4)$$

para o qual solução da equação do calor com condições de contorno $u(0) = a$ e $u(L) = b$ é $u(x) = v(x) + a + (b-a)x/L$. Como podemos conferir, ao usar a expressão de $u(x)$ nos extremos do intervalo, temos: $u(0) = v(0) + a + (b-a)0/L = a$ e $u(L) = v(L) + a + (b-a)L/L = b$.

4 Tarefas

Nesta seção vamos começar a implementar os códigos para resolver as tarefas.

4.1 Código

Abaixo, vamos aplicar os conceitos desenvolvidos acima e avaliar os resultados.

Importando as bibliotecas

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import math
import seaborn as sns
```

Configurações dos gráficos

In [2]:

```
sns.set_style('darkgrid') # darkgrid, white grid, dark, white and ticks
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=15)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=15)    # fontsize of the tick labels
plt.rc('ytick', labelsize=15)    # fontsize of the tick labels
plt.rc('legend', fontsize=15)    # legend fontsize
plt.rc('font', size=15)          # controls default text sizes
```

Funções

$geraVetor()$

A função $geraVetor()$ irá criar os nós que serão utilizados para montar as funções "chapéus". Ela recebe como input o número de nós e o tamanho L do intervalo e retorna tanto o espaçamento quanto os nós (x_vetor).

In [3]:

```
def geraVetor(nos_chip,L):
    h = L/(nos_chip+1)
    x_vetor = np.array([i*h for i in range(nos_chip+2)])
    return h, x_vetor
```

$geraTheta()$

Criados os nós e o espaçamento, agora podemos gerar as funções ϕ_i. Perceba que o valor a ser retornado depende do valor de entrada x, conforme definimos na seção 3.1.

Entradas :

- x: valor de x para o qual se deseja calcular a função
- i: índice i da função ϕ_i
- h: espaçamento entre dois nós consecutivos
- x_vetor: array com os nós

Saída :

- theta_i: valor da função $\phi_i(x)$ no ponto x

In [4]:

```
def geraPhi(x, i, h, x_vetor):
    if x_vetor[i-1]<=x and x<=x_vetor[i]: #verificando se x está no intervalo [xi-1,xi]
        phi_i = (x-x_vetor[i-1])/h
    elif x_vetor[i]<=x and x<=x_vetor[i+1]: #verificando se x está no intervalo [xi,xi+1]
        phi_i = (x_vetor[i+1]-x)/h
    else: #se não estiver nos intervalos anteriores, a função é nula
        phi_i = 0
    return  phi_i
```

montaVetores()

A função abaixo irá montar os vetores da matriz tridiagonal do sistema linear e o vetor do lado direito da equação.

phi_i_phi_i representa $\langle \phi_i, \phi_i \rangle = \int_{x_{i-1}}^{x_{i+1}} [k(x)\phi_i(x)'\phi_i(x)' + q(x)\phi_i(x)\phi_i(x)]dx$. Como $|\phi_i(x)'| = |1/(x_i - x_{i-1})| = 1/h$, a integral acima será dada por $\int_{x_{i-1}}^{x_{i+1}} [k(x)/h^2 + q(x)\phi_i(x)^2]dx$

phi_i_phi_j representa $\langle \phi_i, \phi_j \rangle = \int_{x_{i-1}}^{x_{i+1}} [k(x)\phi_i(x)'\phi_j(x)' + q(x)\phi_i(x)\phi_j(x)]dx$. Vamos assumir $j = i+1$, pois o caso $i = j+1$ será igual. Como $\phi_i(x) = 0, x \notin [x_{i-1}, x_{i+1}]$ e $\phi_j(x) = 0, x \notin [x_i, x_{i+2}]$, a integral se resume a $\int_{x_i}^{x_{i+1}} [-k(x)/h^2 + q(x)\phi_i(x)\phi_j(x)]dx$. Perceba que, neste intervalo, $\phi_i(x)' = -1/(x_{i+1} - x_i) = -1/h$ e $\phi_j(x)' = 1/(x_{i+1} - x_i) = 1/h$.

Finalmente, $\langle f, \phi_i \rangle = \int_{x_{i-1}}^{x_{i+1}} f(x)\phi_i(x)$. Aqui, nós separamos a integral em dois intervalo e resolvemos cada integral separadamente, pois a função $\phi_i(x)$ terá expressões diferentes em cada intervalo. $\int_{x_{i-1}}^{x_{i+1}} f(x)\phi_i(x) = \int_{x_{i-1}}^{x_i} f(x)\phi_i(x) + \int_{x_i}^{x_{i+1}} f(x)\phi_i(x)$. Ou seja, $\int_{x_{i-1}}^{x_{i+1}} f(x)\phi_i(x) = \int_{x_{i-1}}^{x_i} f(x)(x - x_{i-1})/h + \int_{x_i}^{x_{i+1}} f(x)(x_{i+1} - x)/h$. A integral $\int_{x_{i-1}}^{x_i} f(x)(x - x_{i-1})/h$ foi chamada de $f\_phi\_i\_0$ no código e a integral $\int_{x_i}^{x_{i+1}} f(x)(x_{i+1} - x)/h$ foi chamada de $f\_phi\_i\_1$.

No código abaixo iremos calcular essas integrais para cada phi. O único cuidado necessário foi com os índices, uma vez que a contagem de x_i começa em $i = 0$ e a contagem de ϕ_i começa em $i = 1$. Por isso, foi necessário somar ou subtrair 1 de alguns nós.

Por fim, os vetores a e c serão simétricos a descontar o primeiro e o último elementos. Isso acontece por que $\langle \phi_i, \phi_{i+1} \rangle = \langle \phi_{i+1}, \phi_i \rangle = \int_{x_i}^{x_{i+1}} [k(x)\phi_i(x)'\phi_{i+1}(x)' + q(x)\phi_i(x)\phi_{i+1}(x)]dx$.

Entradas :

- nos_chip: quantidade de nós em que calcularemos a temperatura do chip
- nos_integral: quantidade de nós sobre em que aproximaremos a integral pelo método de Gauss
- k: condutividade térmica do material
- f: calor fornecido ou retirado do chip
- q: forçante do sistema, ou, lado direito da equação
- x_vetor: array com os nós
- h: espaçamento entre dois nós consecutivos

Saídas :

- a,b,c: vetores da matriz tridiagonal
- d: vetor do lado direito da equação

In [5]:

```
def montaVetores(nos_chip, nos_integral, k, q, f, h, x_vetor):

    a = np.zeros(nos_chip)
    b = np.zeros(nos_chip)
    c = np.zeros(nos_chip)
    d = np.zeros(nos_chip)

    #montando as funções que serão usadas nas integrais
    for i in range(1,nos_chip+1):
        #montando a função de <phi_i, phi_i>l
        phi_i_phi_i = lambda x: k(x)/(h**2) + q(x)*geraPhi(x, i, h, x_vetor)**2

        #montando a função de <phi_i, phi_j>l |j-1|=1
        phi_i_phi_j = lambda x: -k(x)/(h**2) + q(x)*geraPhi(x, i, h, x_vetor)*geraPhi(x, i+1, h, x_vetor)

        #montando a função de <f, phi_i>
        f_phi_i_0 = lambda x: f(x)*(x-x_vetor[i-1])/h
        f_phi_i_1 = lambda x: f(x)*(x_vetor[i+1]-x)/h
        #criando vetores a,b,c

        #não calculamos a para i=n, pois usamos o intervalo inferior x_vetor[i+1] e o superior x_vetor[i+2]
        if i!=nos_chip:
            c[i-1] = calcula_integral(nos_integral,x_vetor[i],x_vetor[i+1],phi_i_phi_j)
        #a condição anterior não é usada para b e d, pois usamos o intervalo inferior x_vetor[i-1] e
        #o superior x_vetor[i+1]
        b[i-1] = calcula_integral(nos_integral,x_vetor[i-1],x_vetor[i+1],phi_i_phi_i)
        d[i-1] = (calcula_integral(nos_integral,x_vetor[i-1],x_vetor[i],f_phi_i_0) +
                  calcula_integral(nos_integral,x_vetor[i],x_vetor[i+1],f_phi_i_1))

    #o vetor c é simétrico ao a, com exceção do primeiro termo, nulo em a, e do último termo, nulo em c
    #podemos otimizar o tempo calculando apenas o c e atribuindo-o ao a, com exceção dos extremos
    a[1:] = c[:-1]
    return a,b,c,d
```

encontraAlpha()

A função abaixo irá resolver o sistema tridiagonal utilizando os vetores gerados na função $montaVetores()$.

Entradas :

- nos_chip: quantidade de nós em que calcularemos a temperatura do chip
- nos_integral: quantidade de nós sobre em que aproximaremos a integral pelo método de Gauss
- k: condutividade térmica do material
- q: calor fornecido ou retirado do chip
- f: forçante do sistema, ou, lado direito da equação
- x_vetor: array com os nós
- h: espaçamento entre dois nós consecutivos

Saída :

- alpha: solução do sistema tridiagonal

In [6]:

```
def encontraAlpha(nos_chip, nos_integral, k, q, f, h, x_vetor):
    a, b, c, d = montaVetores(nos_chip, nos_integral, k, q, f, h, x_vetor) #gerando os vetores
    alpha = resolveSlTridiagonal(a,b,c,d) #resolvendo o sistema tridiagonal
    return alpha
```

calculaSolucao()

A função abaixo irá utilizar os valores de gerados na função função $alpha()$ para calcular $\overline{u}_n(x) = \sum_{i=1}^n \alpha_i \phi_i(x)$ que melhor aproxima a solução $u(x)$ da equação do calor. A função irá lidar com os casos em que $L \neq 1$, uma vez que os nós serão expressos pela fórmula $x = iL/(n+1), i = 0, 1, \ldots n$ e para o caso com condições de fronteira não homogêneas ao utilizar a expressão $u(x) = v(x) + a + (b-a)x/l$, conforme explicado na seção 3.5.

Entradas :

- nos_chip: quantidade de nós em que calcularemos a temperatura do chip
- nos_integral: quantidade de nós sobre em que aproximaremos a integral pelo método de Gauss
- k: condutividade térmica do material
- f: calor fornecido ou retirado do chip
- q: forçante do sistema, ou, lado direito da equação
- a: condição de fronteira u(0)
- b: condição de fronteira u(L)
- L: comprimento do chip

Saída :

- u_: array com a temperatura encontrada para cada nó x_i

In [7]:

```
def calculaSolucao(nos_chip, nos_integral, k, q, f, a, b, L):
    #gerando o espaçamento e os nós
    h, x_vetor = geraVetor(nos_chip,L)
    #pontos em que serão calculados a temperatura
    x = np.arange(0,L,L/1000)
    #solução do sistema tridiagonal
    alpha = encontraAlpha(nos_chip, nos_integral, k, q, f, h, x_vetor)
    #lista com valores encontrados
    solucao = []
    for j in range(len(x)): #percorrer x_vetor
        xj = x[j]
        v = 0
        for i in range(1,nos_chip+1): #percorrer phi
            v += alpha[i-1]*geraPhi(xj, i, h, x_vetor)
        #caso as condições de fronteira não sejam homogêneas
        if a != 0 or b != 0:
            u = v + a + (b - a)*xj/L
        else:
            u = v
        #adiciona o valor encontrado na lista de solução
        solucao.append(u)

    return solucao
```

calculaErroMaximo()

Essa função será usada para avaliar o algoritmo nos exemplos da seção 4.2. Ela apenas utiliza os resultados encontrados pela função $calculaSolucao()$ e avalia a maior diferença entre as temperaturas encontradas e as temperaturas reais No primeiro exemplo, a temperatura real é regida pela equação $u(x) = x^2(1-x)^2$, $k(x) = 0$ e $f(x) = 12x(1-x) - 2$. O segundo exemplo refere-se ao complemento para a seção 4.2, e temos $u(x) = (x-1)(e^{-x} - 1)$, $f(x) = e^x + 1$ e $k(x) = e^x$. Como as condições foram pré-determinadas pelo enunciado, as únicas entradas serão referentes à quantidade de nós e ao exemplo.

Entradas :

- nos_chip: quantidade de nós em que calcularemos a temperatura do chip
- nos_integral: quantidade de nós sobre em que aproximaremos a integral pelo método de Gauss
- exemplo: 1 para o exemplo da seção 4.2 e 2 para o complemento da seção 4.2

Saída :

- erro_max: maior erro absoluto entre as temperaturas encontradas e as temperaturas reais

In [8]:

```
def calculaErroMaximo(nos_chip, nos_integral, exemplo):
    #condições iniciais
    L = 1
    a = b = 0
    #gerando pontos para calcular a solução
    x = np.arange(0,L,L/1000)
    if exemplo == 1:
        k = lambda x: 1
        q = lambda x: 0
        f = lambda x: 12*x*(1-x)-2
        #calculando os valores reais da temperatura
        u = x ** 2 * (1-x)**2
    elif exemplo == 2:
        k = lambda x: (math.e)**x
        q = lambda x: 0
        f = lambda x: 1 + (math.e)**(x)
        #calculando os valores reais da temperatura
        u = (x-1)*(-1+(math.e)**(-x))
    #temperaturas encontradas pelo método
    u_ = calculaSolucao(nos_chip, nos_integral, k, q, f, a, b, L)
    #calculando o erro máximo
    erro_max = np.max(np.abs(u_ - u))
    return erro_max
```

plotaErros()

Essa função será usada para visualizar os erros para os exemplos da seção 4.2, o inicial e o complementar. Seu único parâmetro é o número do exemplo, pois as condições já foram determinadas no enunciado. Além disso, o algoritmo será calculado para diversos valores de n. Como podemos perceber, é criado um vetor com os valores para testar a solução chamado nos_chip. A função $calculaErroMaximo()$ será testada em cada um desses pontos. O erro cometido por em uma interpolação por splines lineares pode ser majorado por $E_n(\overline{x}) \leq \frac{máx|f''(x)|}{8}h^2$. Para a função $u(x) = x^2(1-x)^2$, $u''(x) = 12x^2 - 12x - 2$ e, para o intervalo [0,1], $máx|u''(x)| = 2$. Por isso, iremos plotar a curva $h^2/4$ junto ao erro cometido no exemplo 1. Similarmente, para a função do exemplo 2 $u(x) = (x-1)(e^{-x} - 1)$, $u''(x) = (e^{-x}x - 3e^{-x})$ e, para o intervalo [0,1], $máx|u''(x)| = 3$ e, então, iremos plotar o erro junto com a curva $3/8h^2$.

Entradas :

- exemplo: 1 para o exemplo da seção 4.2 e 2 para o complemento da seção 4.2

Saída :

- gráfico

In [9]:

```
def plotaErros(exemplo):
    nos_chip = np.arange(10,100,10)
    nos_integral = 6
    lista_erros = []
    lista_h = []
    for nos in nos_chip:
        h = geraVetor(nos,1)[0]
        erro_max = calculaErroMaximo(nos, nos_integral, exemplo)
        if exemplo==1:
            lista_h.append(1/4 * h**2)
        elif exemplo==2:
            lista_h.append(3/8 * h**2)
        lista_erros.append(erro_max)

    plt.figure(figsize=(10,5.33), tight_layout=True)
    plt.plot(nos_chip, lista_erros ,linewidth=2)
    plt.plot(nos_chip, lista_h ,linewidth=2)
    plt.xlabel('$n (nós)$')
    plt.title('Erro vs nós')
    if exemplo==1:
        plt.legend(labels = ['$1/4 h^2$','erro'])
    elif exemplo==2:
        plt.legend(labels = ['$3/8 h^2$','erro'])
    plt.show()
```

plotaSolucaoExemplo()

Essa função será usada para visualizar a temperatura do chip em diferentes pontos para os exemplos da seção 4.2.

Entradas :

- nos_chip: quantidade de nós em que calcularemos a temperatura do chip
- nos_integral: quantidade de nós sobre em que aproximaremos a integral pelo método de Gauss
- exemplo: 1 para o exemplo da seção 4.2 e 2 para o complemento da seção 4.2

Saída :

- gráfico

In [10]:

```
def plotaExemplo(nos_chip,nos_integral,exemplo):
    n = nos_chip
    #condições iniciais
    L = 1
    a = b = 0
    #gerando pontos para calcular a solução
    x = np.arange(0,L,L/1000)
    if exemplo == 1:
        k = lambda x: 1
        q = lambda x: 0
        f = lambda x: 12*x*(1-x)-2
        #calculando os valores reais da temperatura
        u = x ** 2 * (1-x)**2
    elif exemplo == 2:
        k = lambda x: (math.e)**x
        q = lambda x: 0
        f = lambda x: 1 + (math.e)**(x)
        #calculando os valores reais da temperatura
        u = (x-1)*(-1+(math.e)**(-x))
    #temperaturas encontradas pelo método
    u_ = calculaSolucao(nos_chip, nos_integral, k, q, f, a, b, L)
    #gerando o gráfico
    plt.figure(figsize=(10,5.33),tight_layout=True)
    plt.plot(x, u_ ,linewidth=2)
    plt.plot(x,u,linewidth=2)
    plt.xlabel('$x$')
    plt.ylabel('$Temperatura$')
    plt.title(f'Temperatura do chip com {nos_chip} nós no exemplo {exemplo}')
    plt.legend(labels = ['$ \overline{u}_{n}(x) $','$u(x)$'])
    plt.show()
```

plotaSolucao()

Essa função é similar à anterior, mas ela será usada quando não sabemos a solução exata.

Entradas :

- nos_chip: quantidade de nós em que calcularemos a temperatura do chip
- nos_integral: quantidade de nós sobre em que aproximaremos a integral pelo método de Gauss
- k: condutividade térmica do material
- q: forçante do sistema, ou, lado direito da equação
- f: calor fornecido ou retirado do chip
- a: condição de fronteira u(0)
- b: condição de fronteira u(L)
- L: comprimento do chip

Saída :

- gráfico

In [11]:

```
def plotaSolucao(nos_chip,nos_integral,k,q,f,a,b,L):
    #gerando pontos para calcular a solução
    x = np.arange(0,L,L/1000)
    #solução aproximada pelo método
    u_ = calculaSolucao(nos_chip, nos_integral, k, q, f, a, b, L)
    #gerando o gráfico
    plt.figure(figsize=(9,5),tight_layout=True)
    plt.plot(x, u_ ,linewidth=2)
    plt.xlabel('$x$')
    plt.ylabel('$Temperatura$')
    plt.title(f'Temperatura do chip com {nos_chip} nós')
    plt.legend(labels = ['$ \overline{u}_{n}(x)$'])
    plt.show()
```

Funções dos Exercícios Programas antigos

In [12]:

```
def cria_nos_pesos(n):
    """Cria os nós e os pesos que serão utilizados para calcular as integrais
    Args:
        n:  quantidade de pontos
    Return:
        x: nós
        w: pesos"""
    if n == 6:
        x = np.array([-0.2386191860831969086305017,-0.6612093864662645136613996,-0.9324695142031520278123016,
                      0.2386191860831969086305017,0.6612093864662645136613996,0.9324695142031520278123016])
        w = np.array([0.4679139345726910473898703,0.3607615730481386075698335,0.1713244923791703450402961,
                      0.4679139345726910473898703,0.3607615730481386075698335,0.1713244923791703450402961])
    elif n == 8:
        x = np.array([-0.1834346424956498049394761,-0.5255324099163289858177390,-0.7966664774136267395915539,-0
                      0.1834346424956498049394761,0.5255324099163289858177390,0.7966664774136267395915539,0.9
        w = np.array([0.3626837833783619829651504,0.3137066458778872873379622,0.2223810344533744705443560,0.101
                      0.3626837833783619829651504,0.3137066458778872873379622,0.2223810344533744705443560,0.1
    elif n == 10:
        x = np.array([-0.1488743389816312108848260,-0.4333953941292471907992659,-0.6794095682990244062343274,-0
                       0.1488743389816312108848260,0.4333953941292471907992659,0.6794095682990244062343274,0.
        w = np.array([0.2955242247147528701738930,0.2692667193099963550912269,0.2190863625159820439955349,0.149
                       0.2955242247147528701738930,0.2692667193099963550912269,0.2190863625159820439955349,0.
    return x, w


def calcula_integral(n,a,b,f):
    """Calcula a integral de uma função com limites inferior e superior constantes
    Args:
        a: limite inferior
        b: limite superior
        f: função a ser integrada
        n: número de pontos"""
    nos, w = cria_nos_pesos(n) #gerando os nós e os pesos
    nos_t = (nos*(b-a)+a+b)/2 #transportando os nós para o intervalo [a,b]
    w_t = w*(b-a)/2 #transportando os pesos para o intervalo [a,b]
    f_vec = np.vectorize(f)
    F = f_vec(nos_t) #calculando f(x) para os valores transportados
    I = F @ w_t  #calculando a integral
    return I

def decompoeTridiagonalLU(a,b,c):
    """Faz a decomposição em LU de uma matriz A nxn tridiagonal, com coeficientes a, b, c.
    Args:
        a, b, c: diagonais não nulas da matriz tridiagonal A
    Return:
        l: multiplicadores da matriz L
        u: diagonal principal de U
    """
    n = len(a)
    u = np.zeros(n)
    l = np.zeros(n)
    u[0] = b[0]
    for i in range(1,n,1):
        l[i] = a[i]/u[i-1]
        u[i] = b[i]-l[i]*c[i-1]
    return l, u

def resolveSlTridiagonal(a,b,c,d):
    """Resolve um sistema linear tridiagonal
    Args:
        a, b, c: diagonais não nulas da matriz tridiagonal A
        d: vetor tal que Ax = d
    Return:
        x: solução do sistema linear
    """
    #gerando os vetores que representam L e U
    l, u = decompoeTridiagonalLU(a,b,c)

    n = len(a)
    y = np.zeros(n)
    x = np.zeros(n)

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

    #Ux = y
    x[n-1] = y[n-1]/u[n-1]
    for i in range(n-2,-1,-1):
        x[i] = (y[i] - c[i]*x[i+1])/u[i]

    return x
```

Exercícios

4.2 Validação no exemplo original

Devíamos implementar o método de elementos finitos para resolver a equação com $q(x) = 0, k(x) = 1, f(x) = 12x(1-x) - 2, u(x) = x^2(1-x)^2, u(0) = 0, u(L) = 0, L = 1$. Essas condições já foram inseridas na função $calculaErroMaximo()$

Condições

In [13]:

```
nos_chip = np.array([7,15,31,63])
nos_integral = 10
exemplo = 1
```

Solução

In [14]:

```
lista_erros = []
for nos in nos_chip:
    erro_max = calculaErroMaximo(nos, nos_integral, exemplo=1)
    print(f"O maior erro cometido com {nos} nós foi: {erro_max} \n")
```

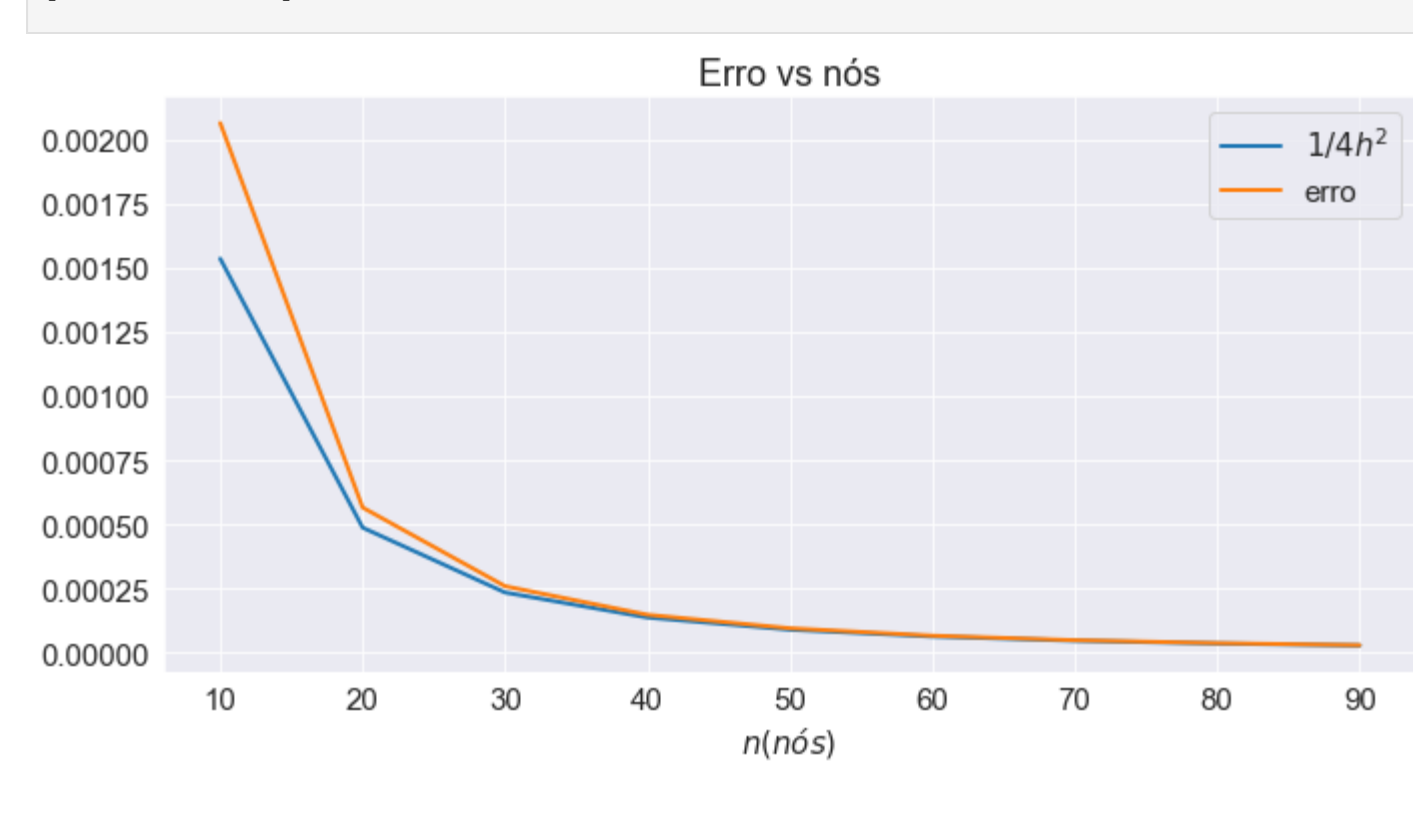
```
O maior erro cometido com 7 nós foi: 0.002565908124000001

O maior erro cometido com 15 nós foi: 0.0008011392187500005

O maior erro cometido com 31 nós foi: 0.00022161026367187092

O maior erro cometido com 63 nós foi: 5.814417157811694e-05
```

Visualização

Para facilitar, vamos visualizar os erros em um gráfico.

In [15]:

```
plotaErros(exemplo=1)
```



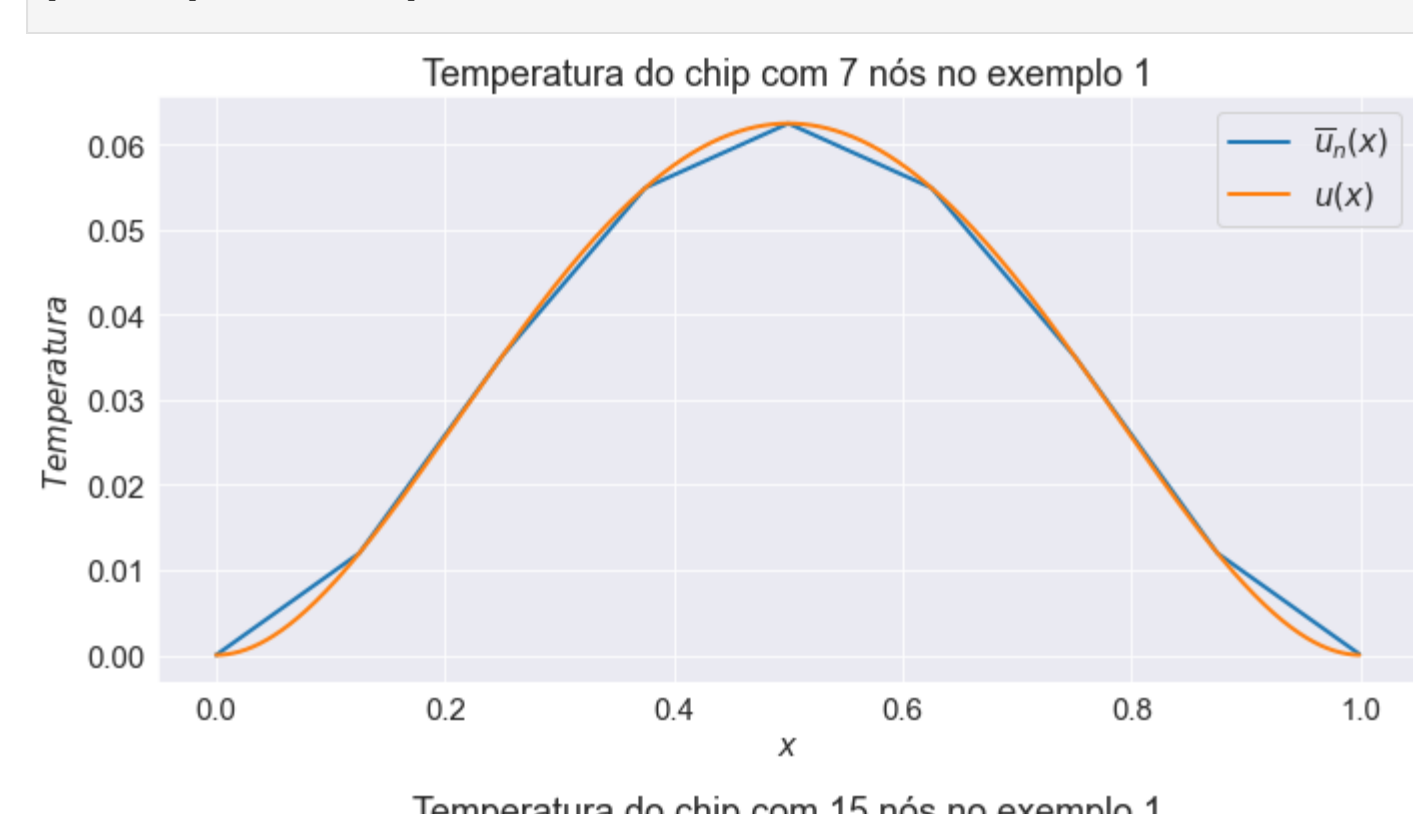
Resultados

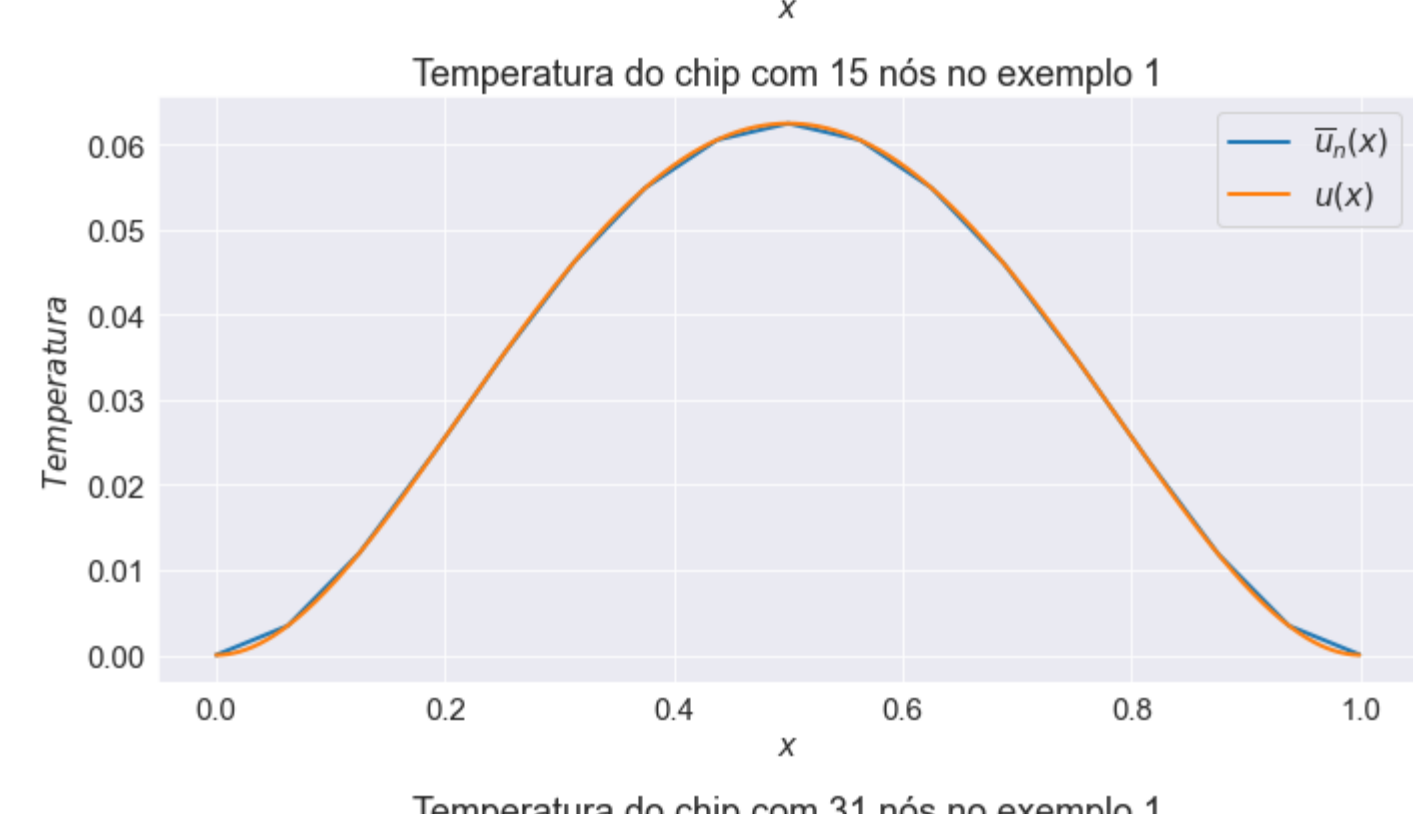
Como podemos ver no gráfico acima, o erro cometido fica muito próximo à $1/4h^2$. Apesar de o erro não ter sido menor, podemos ver que a maior diferença entre o erro e $1/4h^2$ foi 0.0005. Há duas razões para isso acontecer: a primeira é que o número de nós usados no método de elementos finitos era pequeno quando houve essa diferença, n=10 nós. Somado à isso, a integral não foi calculada de forma exata, mas foi aproximada pelo método de Gauss, o que propagou o erro na geração da matriz tridiagonal e, consequentemente, gerou erros de aproximação nos valores de α.

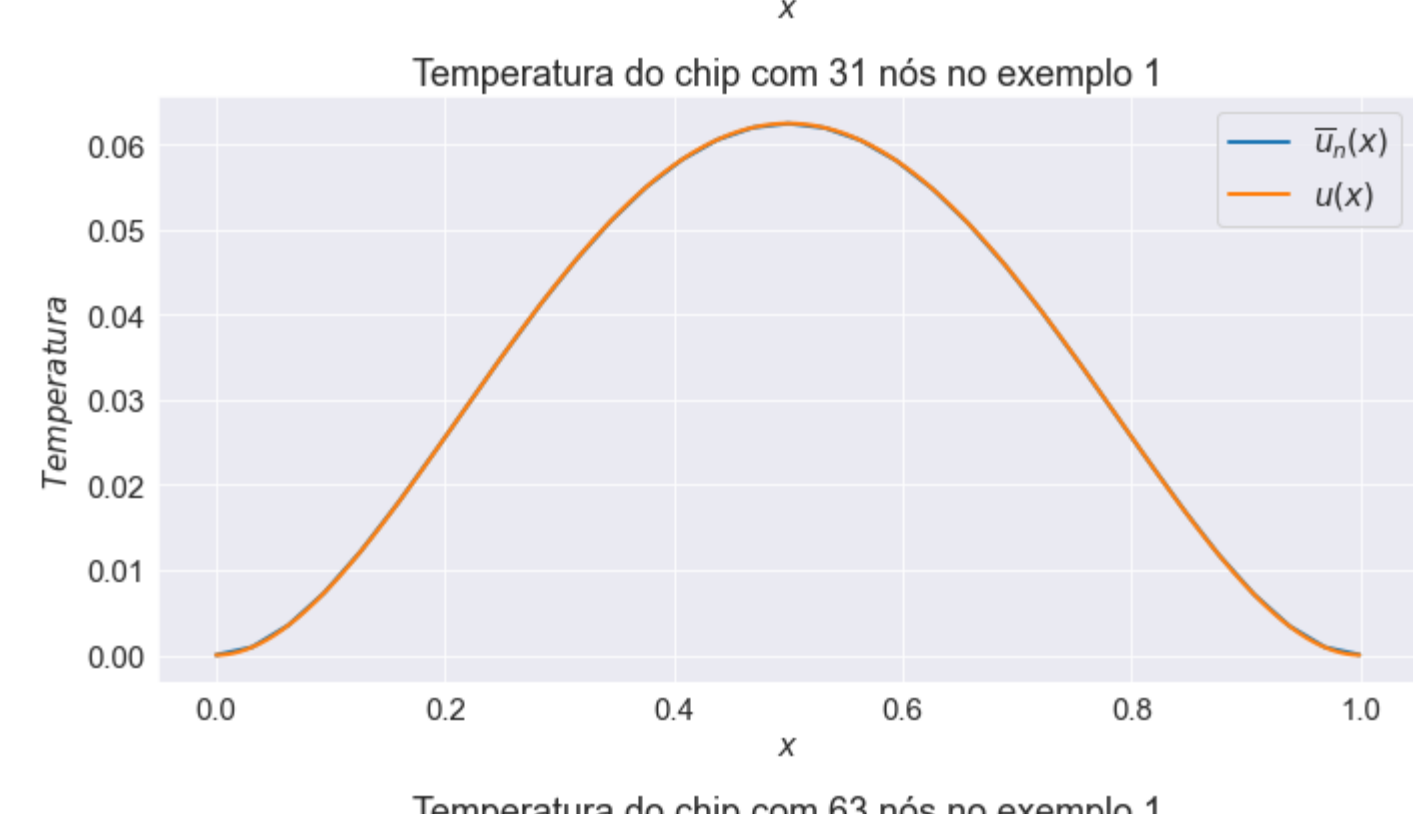
Além disso, podemos visualizar qual a temperatura encontrada para o chip nos diferentes pontos considerados.

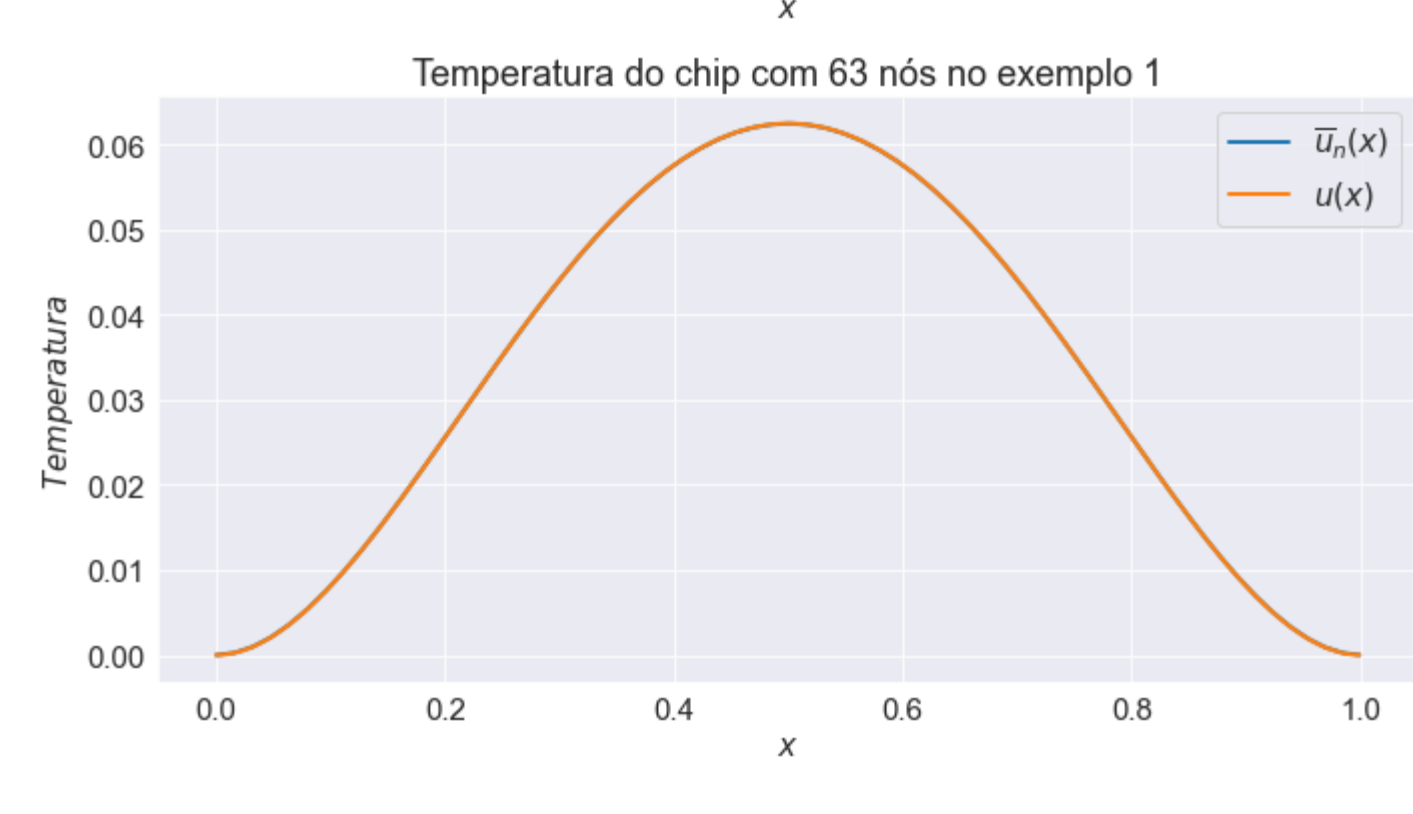
In [16]:

```
plotaExemplo(7,10,exemplo=1)
plotaExemplo(15,10,exemplo=1)
plotaExemplo(31,10,exemplo=1)
plotaExemplo(63,10,exemplo=1)
```









Resultados

Dos gráficos acima, podemos visualizar que a solução aproximada converge para a solução exata conforme aumentamos o número de nós.

4.2 Validação no exemplo complementar

Devíamos implementar o método de elementos finitos para resolver a equação com $q(x) = 0, k(x) = e^x, f(x) = e^x + 1, u(x) = (x-1)(e^{-x} - 1), u(0) = 0, u(L) = 0, L = 1$. Essas condições já foram inseridas na função $calculaErroMaximo()$

Condições

In [17]:

```
nos_chip = np.array([7,15,31,63])
nos_integral = 10
exemplo = 2
```

Solução

In [18]:

```
lista_erros = []
for nos in nos_chip:
    erro_max = calculaErroMaximo(nos, nos_integral, exemplo=2)
    print(f"O maior erro cometido para o exemplo {exemplo} com {nos} nós foi: {erro_max} \n")
```

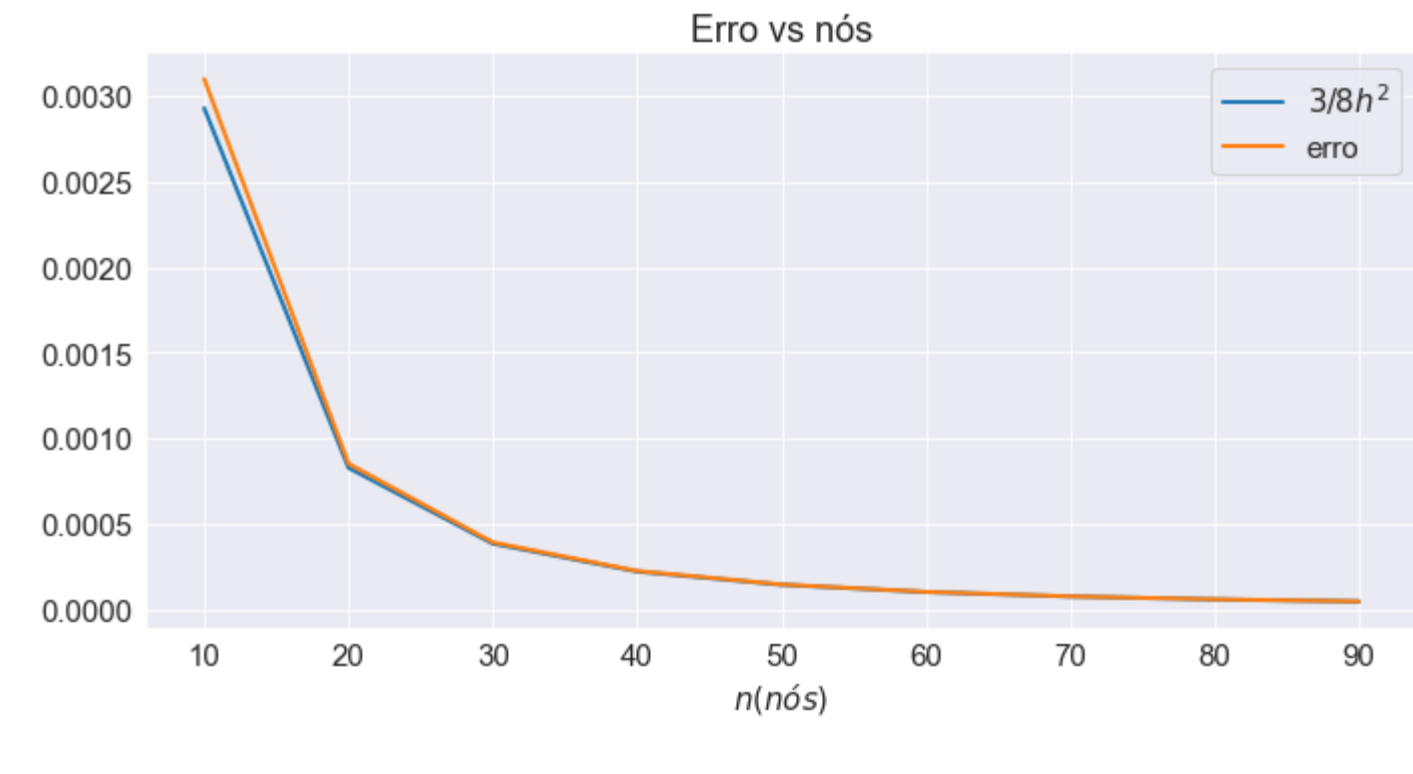
```
O maior erro cometido para o exemplo 2 com 7 nós foi: 0.0054205653001270365

O maior erro cometido para o exemplo 2 com 15 nós foi: 0.0014090298614416637

O maior erro cometido para o exemplo 2 com 31 nós foi: 0.000358917298911127

O maior erro cometido para o exemplo 2 com 63 nós foi: 9.06109820144451e-05
```

Visualização

Para facilitar, vamos visualizar os erros em um gráfico.

In [19]:

```
plotaErros(exemplo=2)
```

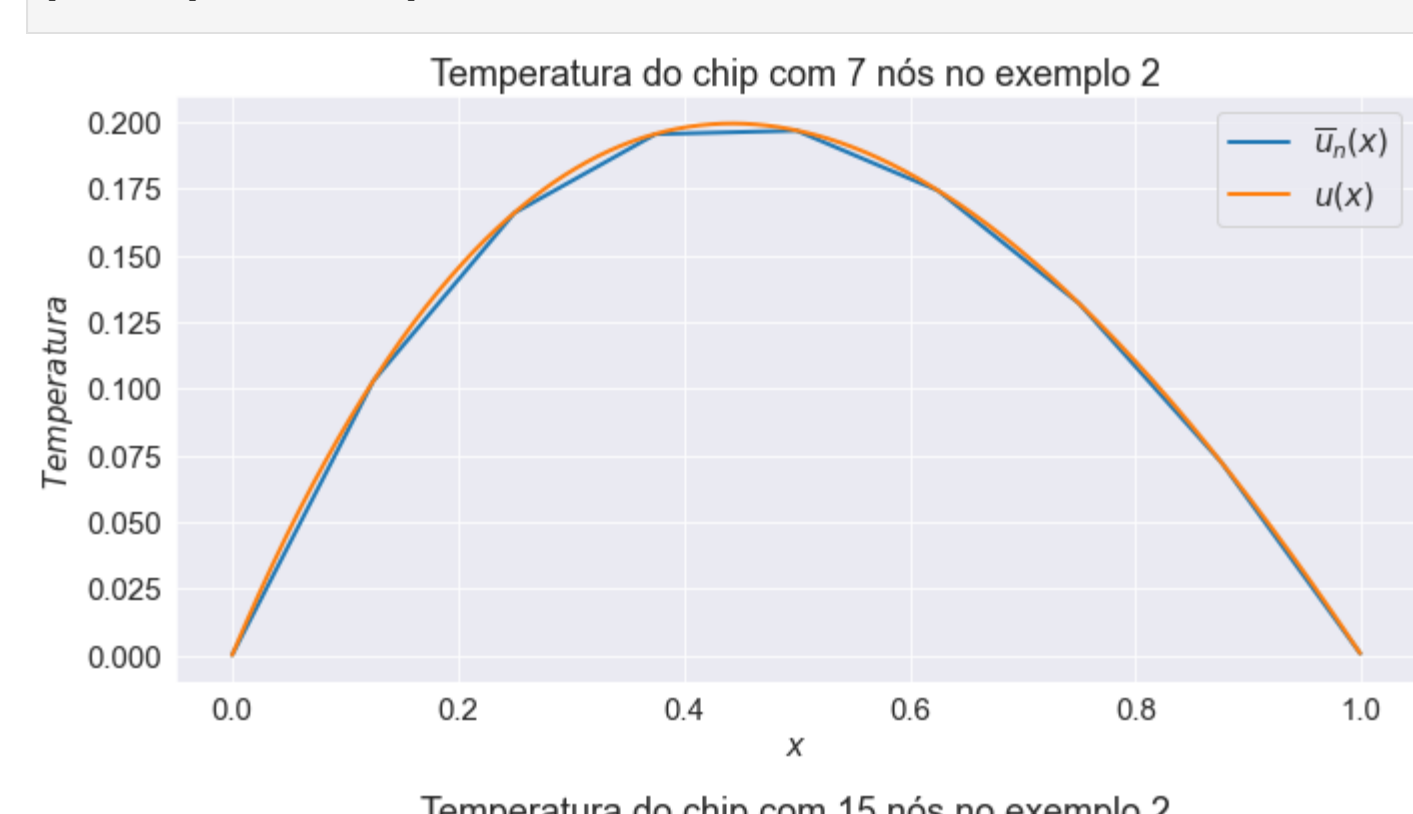


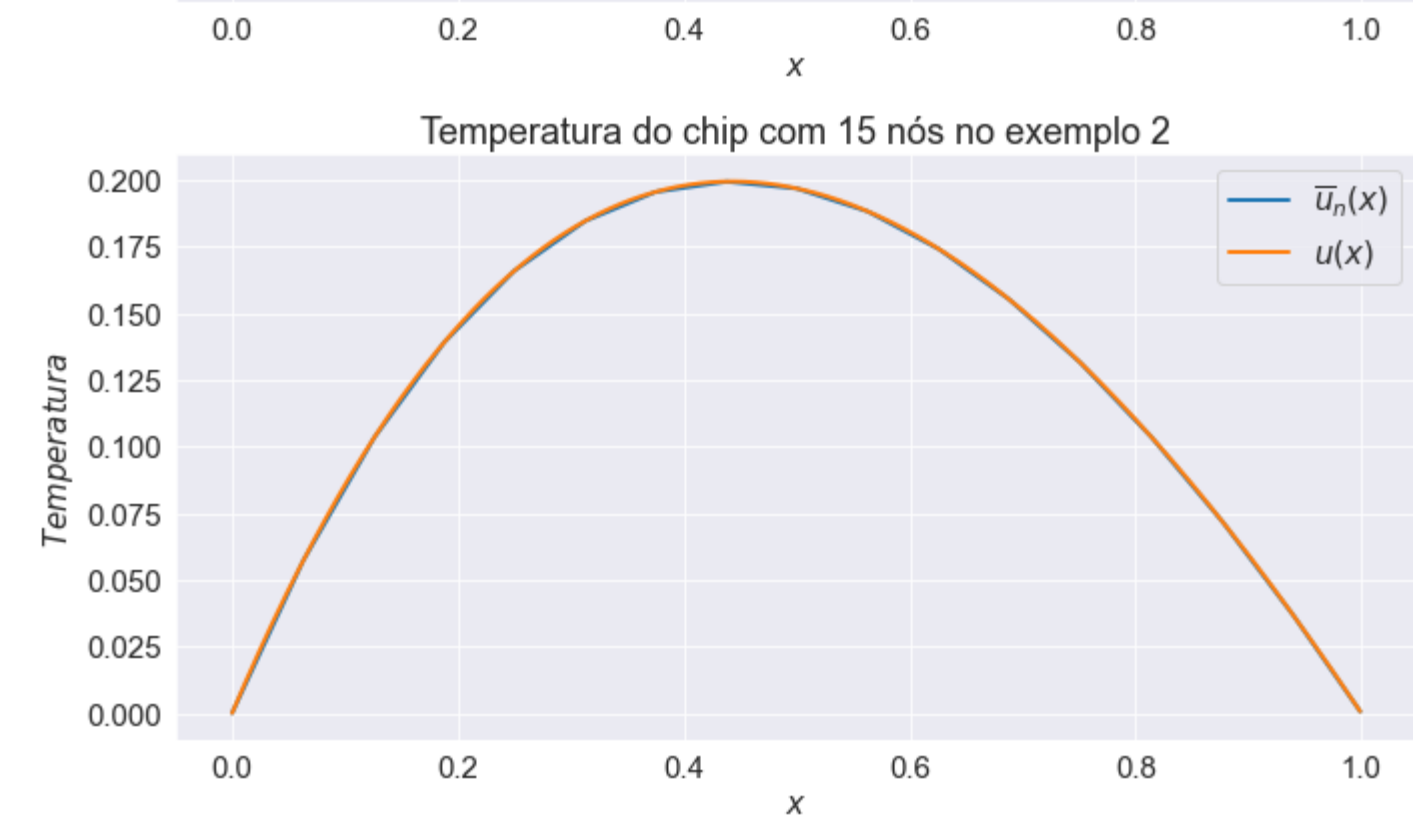
Resultados

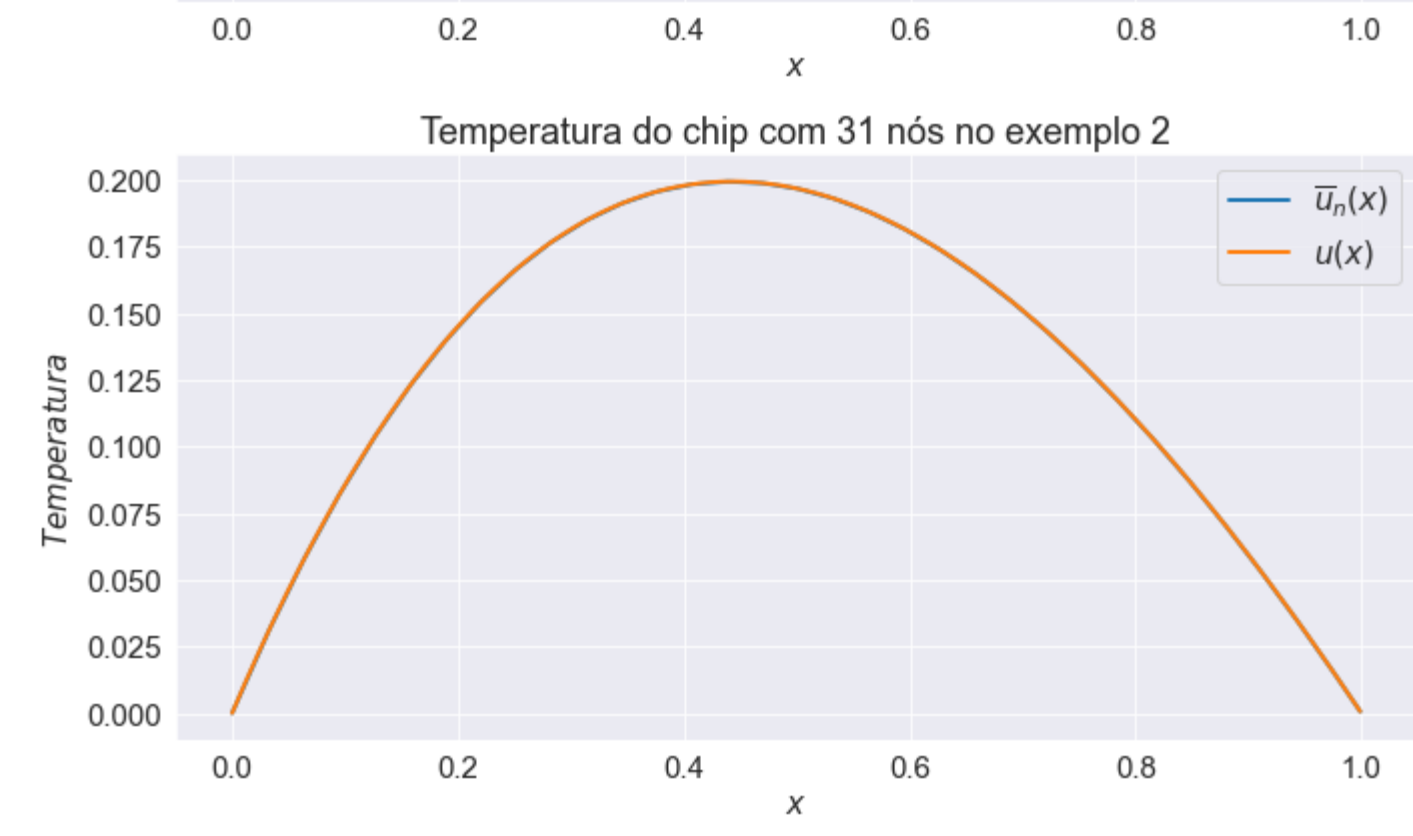
Como podemos ver no gráfico acima, o erro cometido praticamente sobrepõe a curva $3/8h^2$. Assim como no exemplo anterior, era esperado que o resultado não fosse perfeito pelos erros de aproximação cometidos na integral pelo método de Gauss, que é propagado na resolução do sistema tridiagonal, além do número de nós escolhidos para o método de elementos finitos.

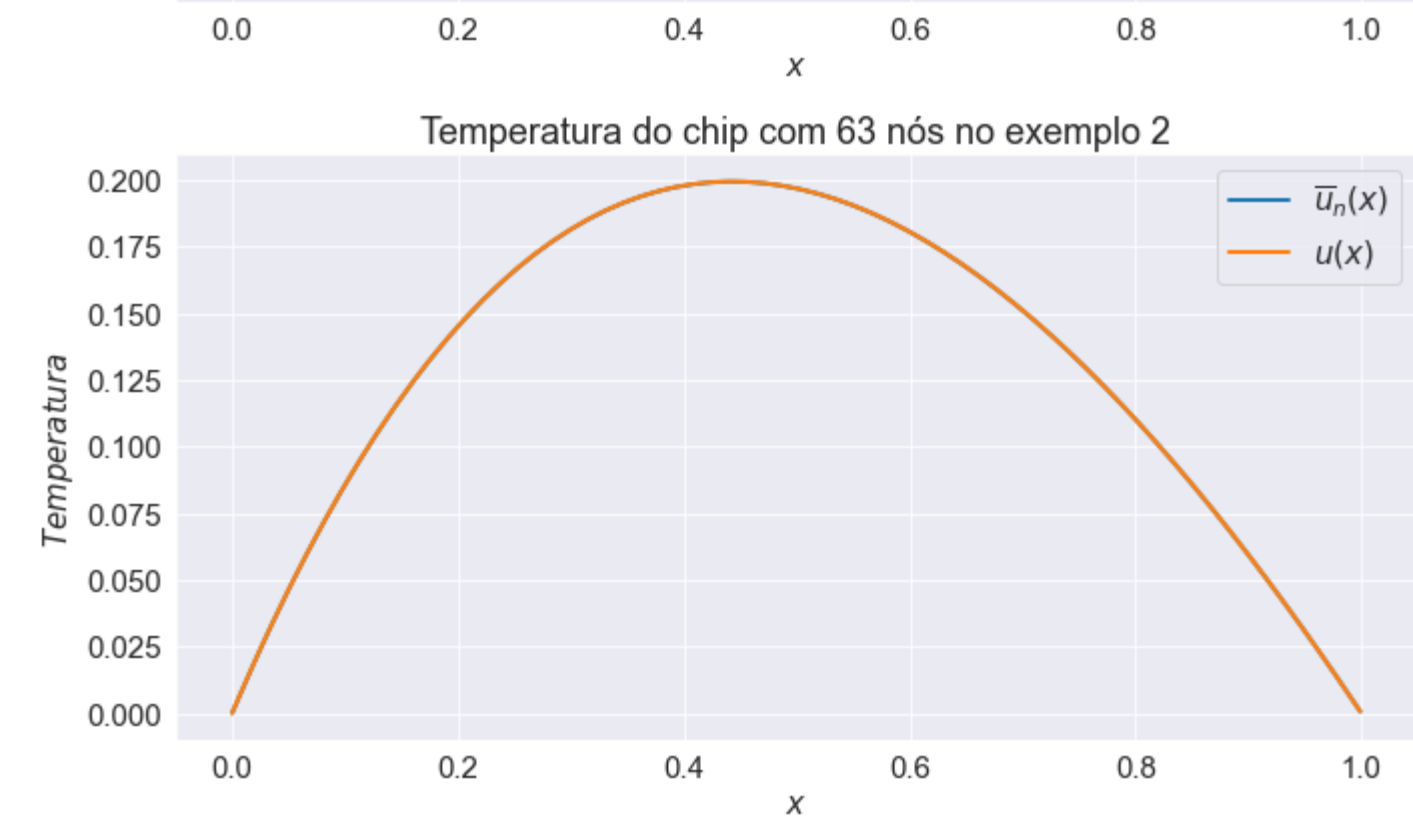
Além disso, podemos visualizar qual a temperatura encontrada para o chip nos diferentes pontos considerados.

In [20]:
```
plotaExemplo(7,10,exemplo=2)
plotaExemplo(15,10,exemplo=2)
plotaExemplo(31,10,exemplo=2)
plotaExemplo(63,10,exemplo=2)
```









Resultados

Dos gráficos acima, podemos visualizar que a solução aproximada converge para a solução exata conforme aumentamos o número de nós.

4.3 Equilíbrio com forçantes de calor

4.3.1 Aquecimento com dados do enunciado

Vamos considerar que o chip seja formado apenas de silício $k(x) = k = 3,6W/(mK)$, considerando que há produção de calor pelo chip. Vamos assumir que o chip esquenta mais em sua parte central que nas bordas, o que pode ser modelado por uma Gaussiana da seguinte forma $Q_+(x) = Q^0_+ e^{-(x-L/2)^2/\sigma^2}$

com Q_0 uma constante indicando o máximo de calor gerado no centro do chip e σ controlando a variação de geração de calor em torno do ponto central do chip. Se σ for muito pequeno, podemos ter uma calor gerado praticamente somente no centro do chip.

Perceba que usaremos o valor $L = 3$.

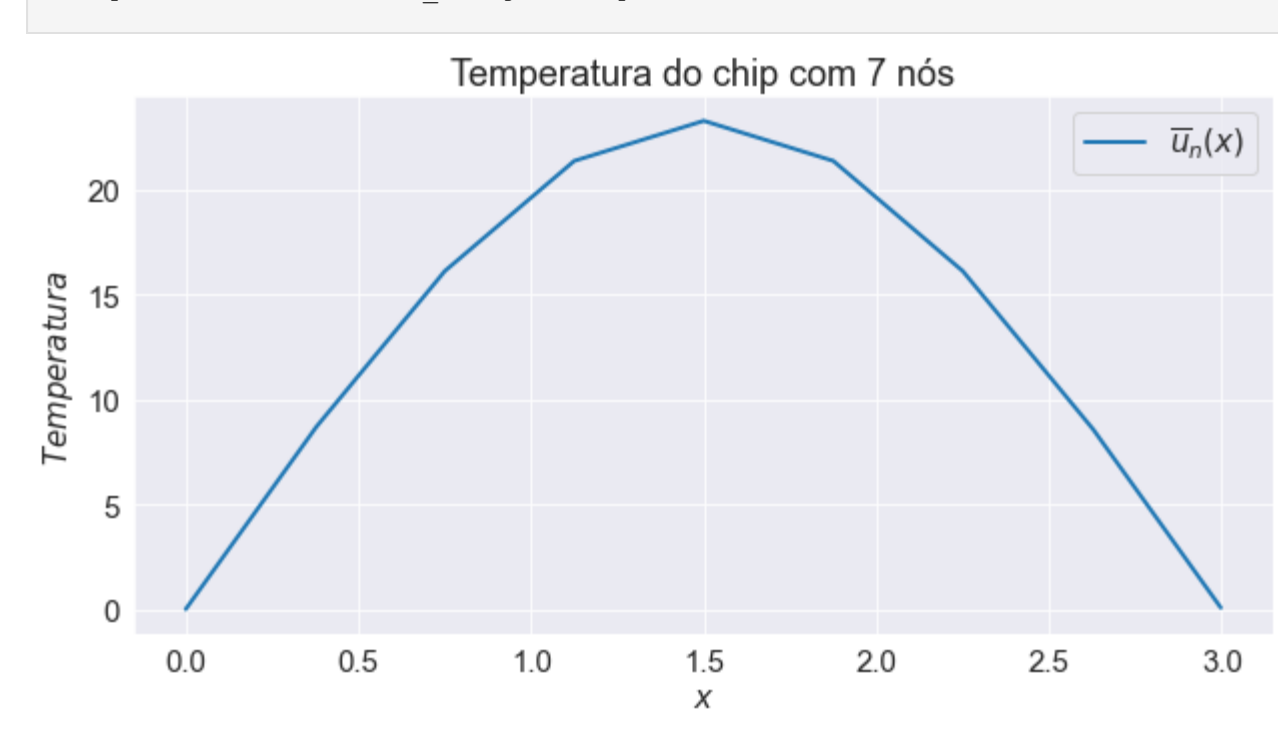
Condições

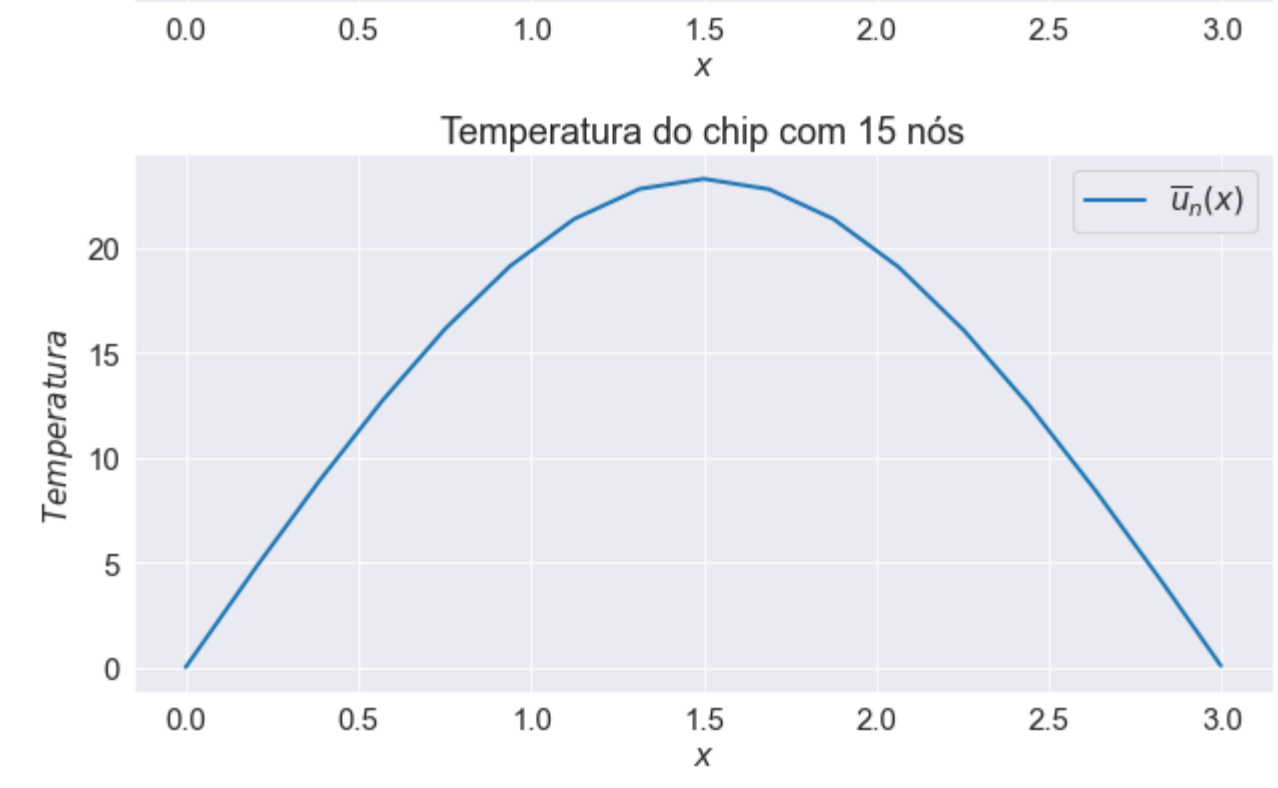
Vamos plotar os resultados para os seguintes valores de nós: $7, 15, 31, 63$.

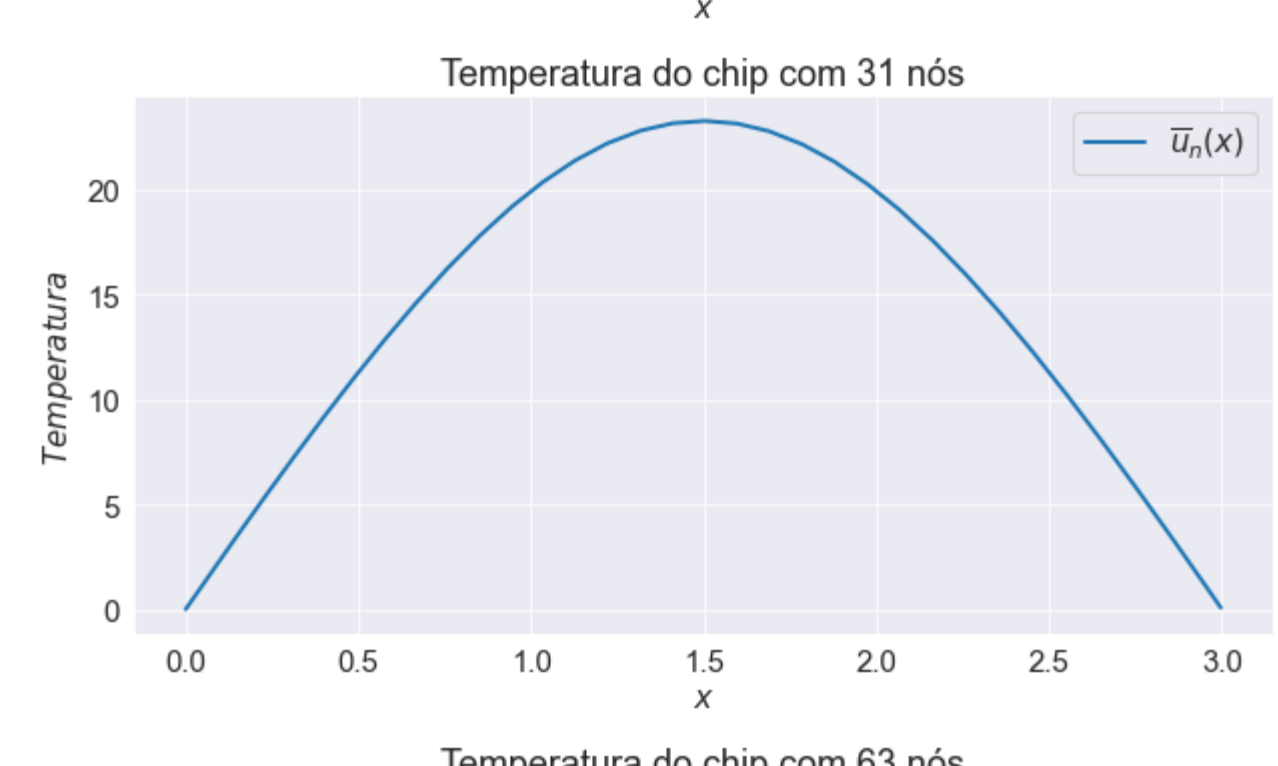
In [21]:
```
#lista com valores de nós que usaremos
nos_chip = [7,15,31,63]
nos_integral = 10
L = 3
a = 0
b = 0
sigma = 1
k = lambda x: 3.6
q = lambda x: 0
f = lambda x: 100*math.e**((-(x-L/2)**2)/sigma**2)
```

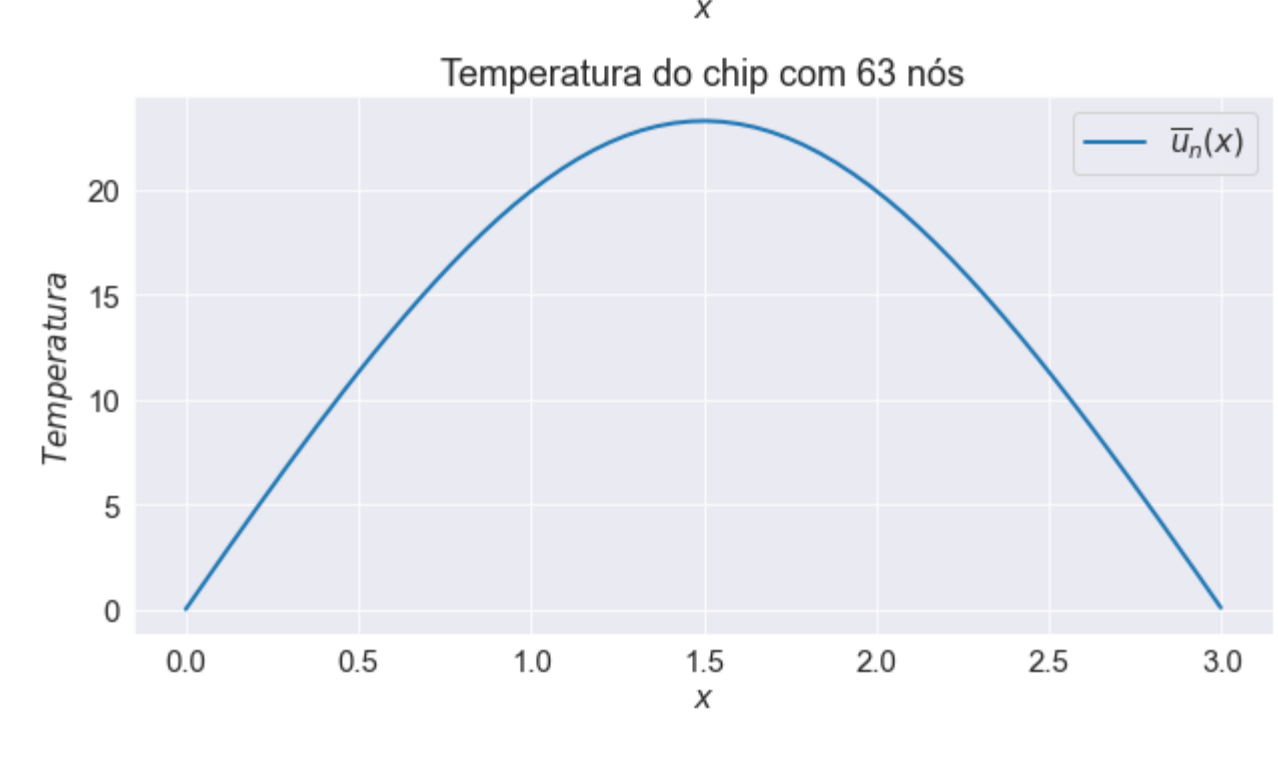
Visualização

In [22]:
```
#plotando os resultados
for nos in nos_chip:
    plotaSolucao(nos,nos_integral,k,q,f,a,b,L)
```









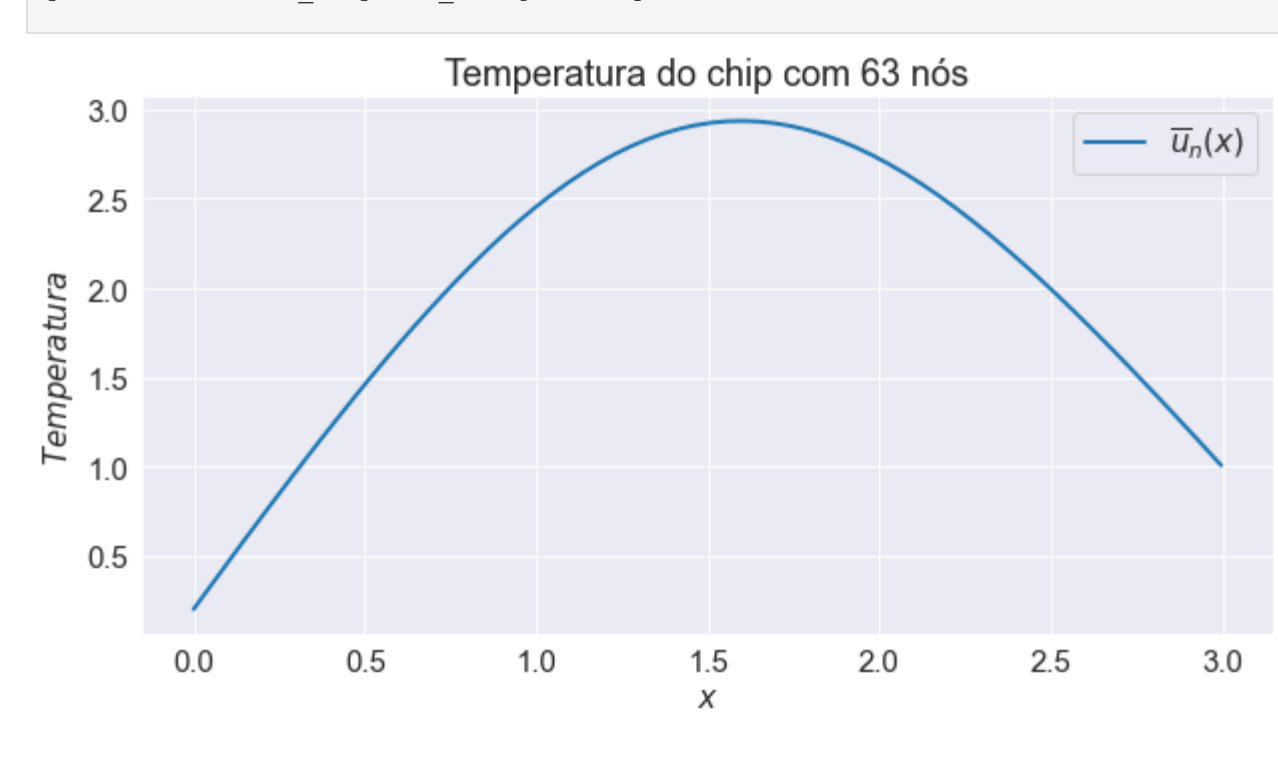
Resultados

Dos gráficos acima, podemos ver que a curva da temperatura fica mais suavizada com o aumento dos nós, conforme esperávamos já que a solução encontrada deve se aproximar da solução exata. Além disso, é possível ver que há uma região de aquecimento no meio do chip, já que é onde houve as maiores temperaturas. Ademais, a temperatura decai conforme se aproxima das bordas, uma vez que o aquecimento diminui, até chegar em zero nas extremidades, uma vez que essa era nossa condição de contorno.

Condições não homogêneas

Agora, vamos simular o problema para os seguintes parâmetros: $a = 0.2, b = 1, L = 3$.

In [23]:
```
#lista com valores de nós que usaremos
nos_chip = 63
nos_integral = 10
L = 3
a = 0.2
b = 1
sigma = 1
k = lambda x: 3.6
q = lambda x: 0
f = lambda x: 10*math.e**((-(x-L/2)**2)/sigma**2)
plotaSolucao(nos_chip,nos_integral,k,q,f,a,b,L)
```



Resultados

Podemos observar que o gráfico corrobora com a solução imaginada para o problema. Como a forçante é igual ao intem anterior, o formato da curva deveria ser parecido. É importante notar que a função se comportou de forma adequada mesmo que o tamanho do chip não seja unitário e as condições de fronteira não sejam homogêneas, pois $u(0) = 0.2$ e $u(3) = 1$.

4.3.2 Resfriamento

Vamos considerar que o chip seja formado apenas de silício $k(x) = k = 3,6W/(mK)$, considerando que há retirada de calor. Vamos assumir que mais calor é retirado dos extremos, o que pode ser modelado da seguinte forma:

$$Q_-(x) = Q^0_- \left(e^{-(x)^2/\theta^2} + e^{-(x-L)^2/\theta^2}\right)$$

Condições

Antes de visualizar a solução, vamos entender o problema para criar uma intuição sobre a solução.

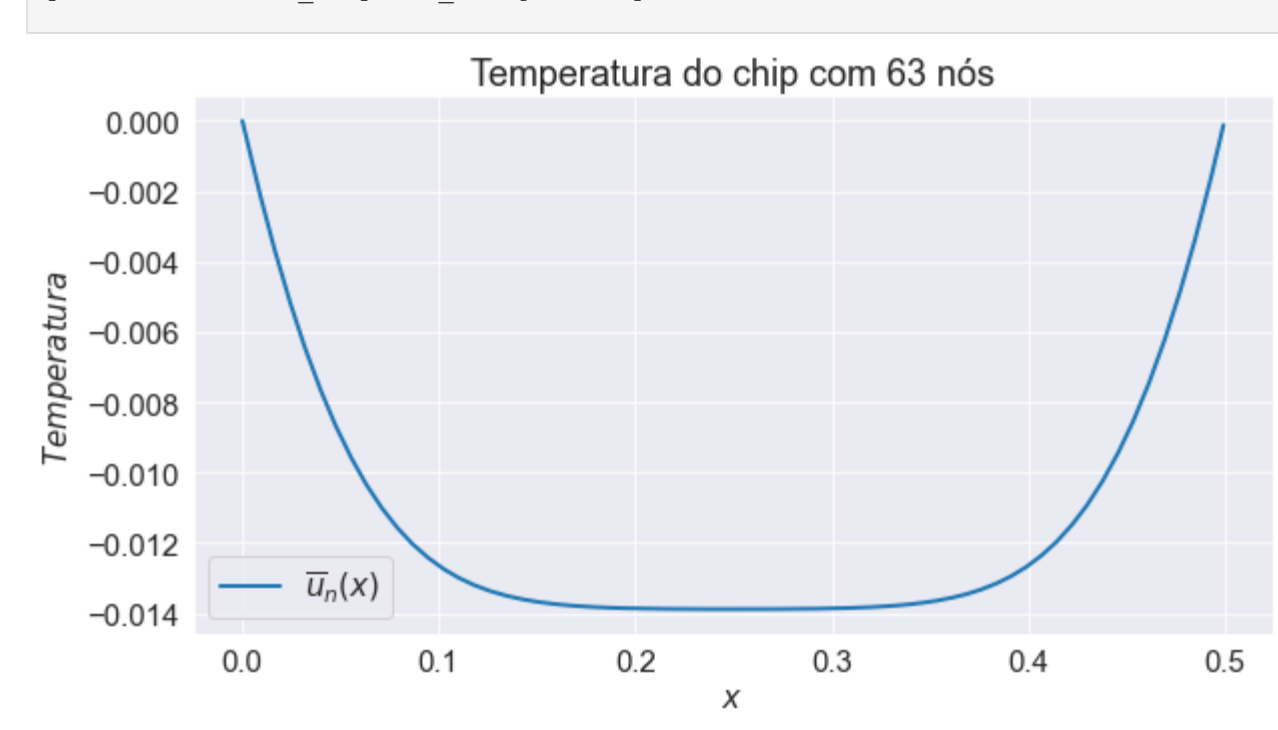
Neste exercício, Q^0_- representa quanto calor é retirado da barra. Por isso, será negativo.

A função $f(x)$ representa, neste caso, o calor retirado do chip. Como podemos observar da função, o módulo do calor retirado será maior nas extremidades do que no centro do chip. Logo, é de se esperar que as menores temperaturas ocorram nos extremos. Além disso, θ pequeno simboliza retirada de calor concentrada no centro do chip, enquanto $theta$ grande indica uma retirada de calor mais uniforme ao longo da barra.

Aqui, para não poluir o relatório com muitos gráficos, vamos utilizar 63 como o número de nós para o método de elementos finitos, uma vez que é o resultado que nos dará mais precisão.

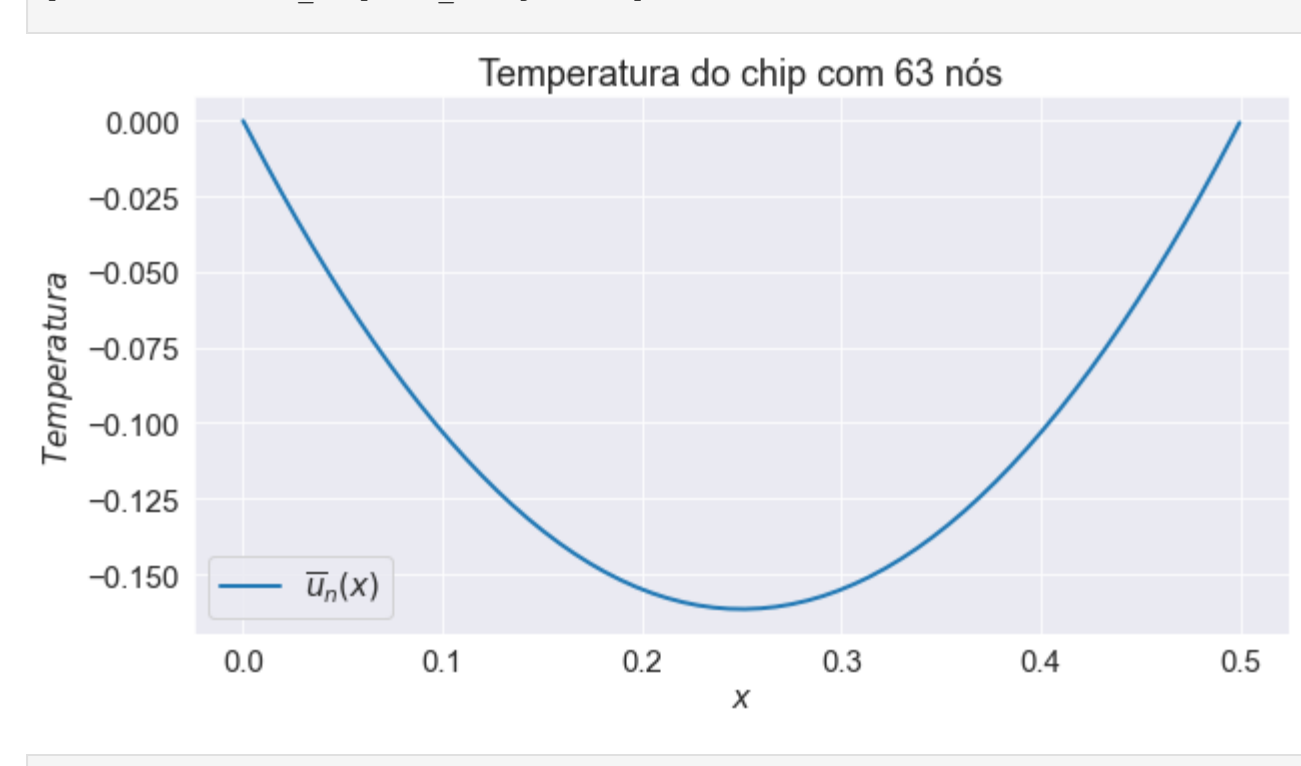
Além disso, usaremos $L = 0.5$.

In [24]:
```
#theta = 0.1
nos_chip = 63
nos_integral = 10
L = 0.5
a = 0
b = 0
theta = 0.1
k = lambda x: 3.6
q = lambda x: 0
f = lambda x: -10*(math.e**(-(x/theta)**2)+math.e**(-(x-L)**2 / theta**2))
plotaSolucao(nos_chip,nos_integral,k,q,f,a,b,L)
```

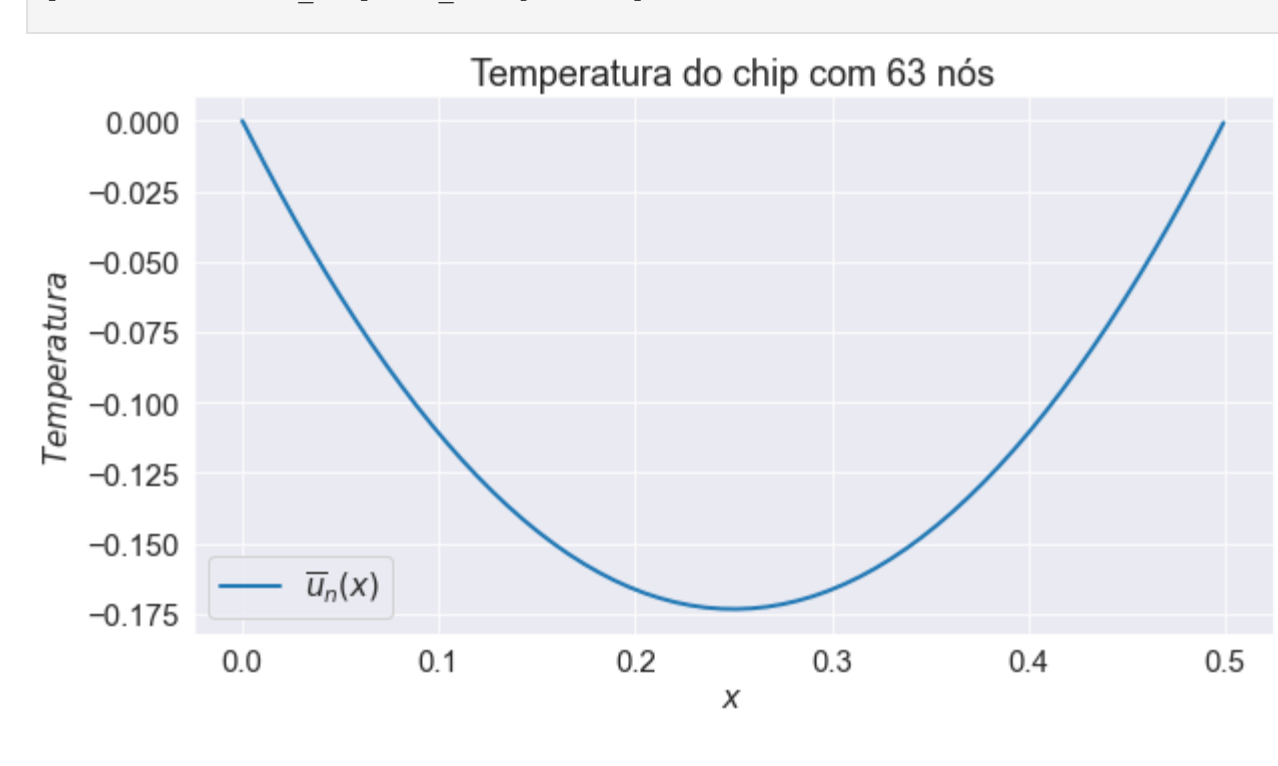


Solução

In [25]:
```
#theta = 1
nos_chip = 63
nos_integral = 10
L = 0.5
a = 0
b = 0
theta = 1
k = lambda x: 3.6
q = lambda x: 0
f = lambda x: -10*(math.e**(-(x/theta)**2)+math.e**(-(x-L)**2 / theta**2))
plotaSolucao(nos_chip,nos_integral,k,q,f,a,b,L)
```



In [26]:
```
#theta = 1
nos_chip = 63
nos_integral = 10
L = 0.5
a = 0
b = 0
theta = 10
k = lambda x: 3.6
q = lambda x: 0
f = lambda x: -10*(math.e**(-(x/theta)**2)+math.e**(-(x-L)**2 / theta**2))
plotaSolucao(nos_chip,nos_integral,k,q,f,a,b,L)
```



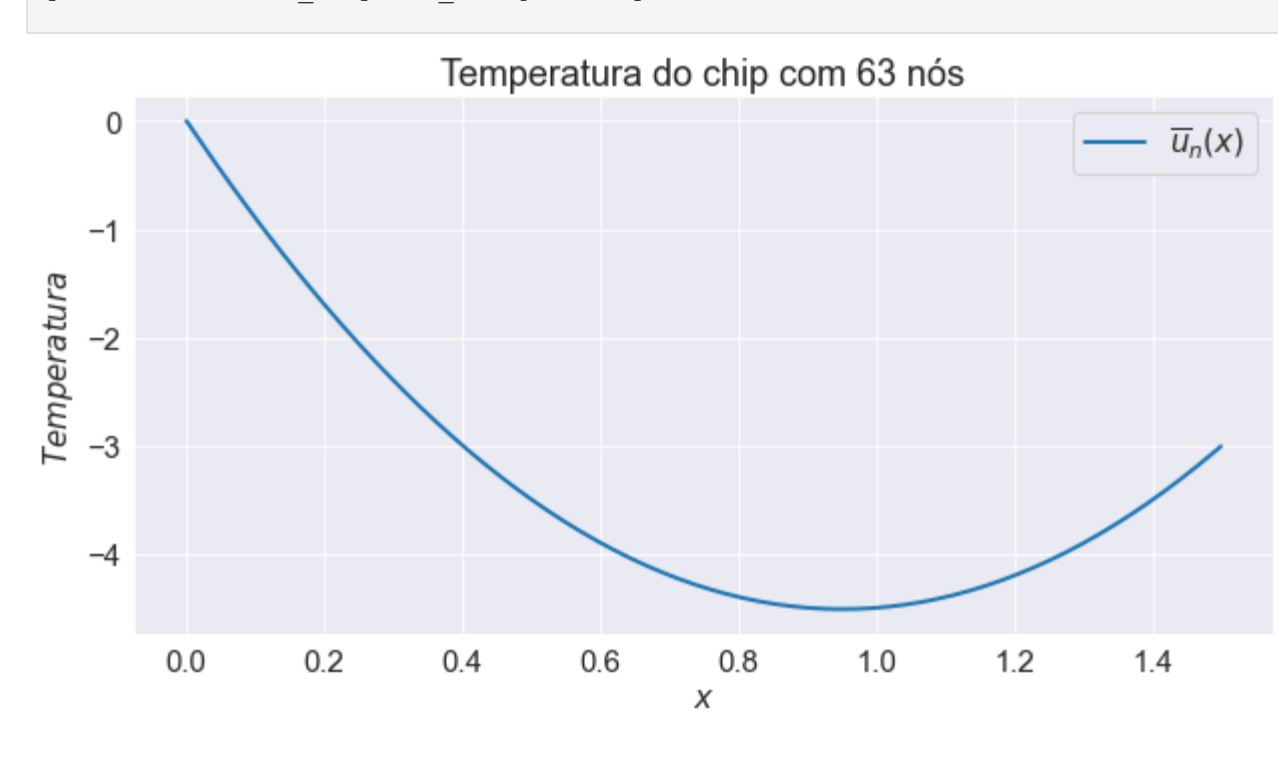
Resultados

Assim como previsto, aumentar o θ fez com que a variação de temperatura fosse suavizada.

Condições não homogêneas

Podemos, também, alterar os outros parâmetros. Vamos modificar a taxa de calor retirado e as condições de fronteira.

In [27]:
```
nos_chip = 63
nos_integral = 10
L = 1.5
a = 0
b = -3
k = lambda x: 1
q = lambda x: 0
f = lambda x: -10
plotaSolucao(nos_chip,nos_integral,k,q,f,a,b,L)
```



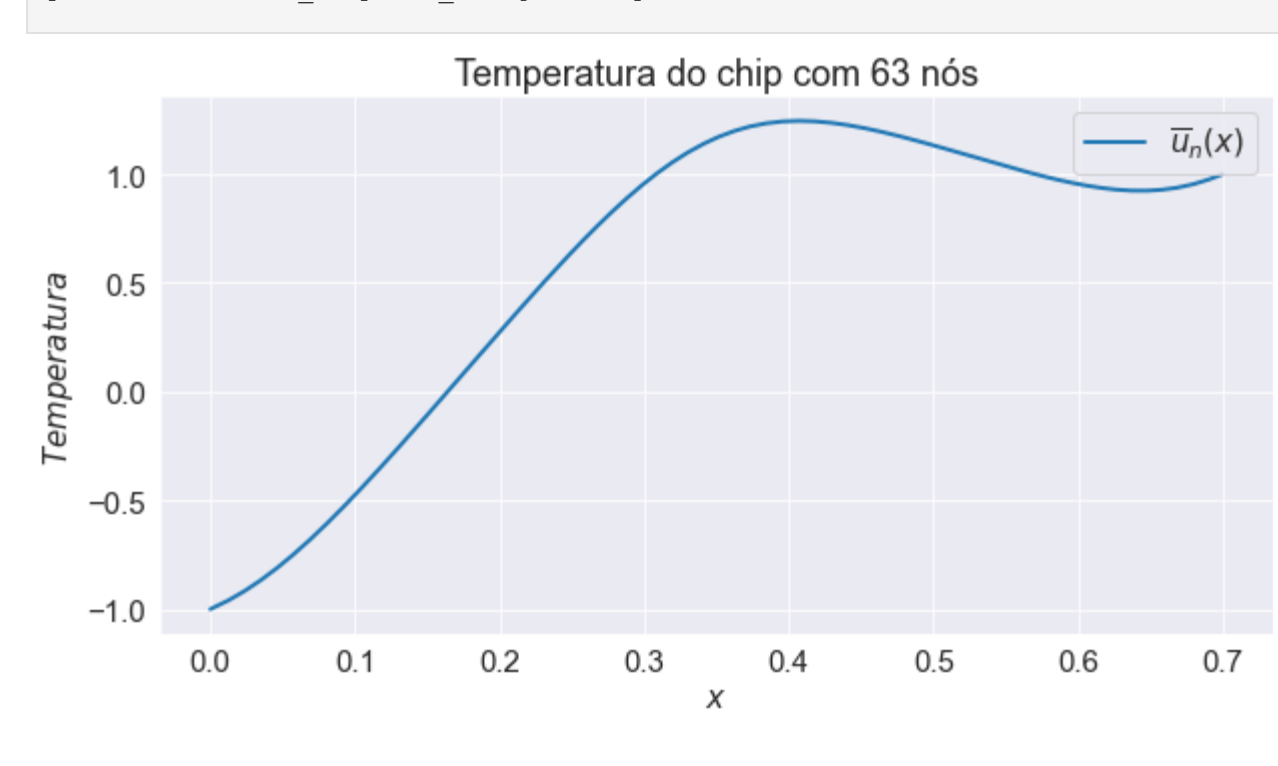
Resultados

Desta vez, a taxa de calor retirada é constante e igual à -10. A curva tem um formato tal que $u(0) = 0$ e $u(1.5) = -3$. Podemos perceber que embora a taxa de retirada de calor seja constante e as condições de fronteira não homogêneas, a menor temperatura foi obtida em $x \approx 0.8$, o que pode parecer contra-intuitivo em um primeiro momento, por isso o método do elementos finitos pode ser tão proveitoso.

Fornecimento e retirada de calor

Por fim, vamos testar o algoritmo combinando as situações anteriores.

In [28]:
```
nos_chip = 63
nos_integral = 10
L = 0.7
a = -1
b = 1
theta = 0.1
sigma = 0.1
k = lambda x: 3.6
q = lambda x: 0
f = lambda x: -200*(math.e**(-(x/theta)**2)+math.e**(-(x-L)**2 / theta**2)) + 200*math.e**((-(x-L/2)**2)/sigma*
plotaSolucao(nos_chip,nos_integral,k,q,f,a,b,L)
```



Resultados

Finalmente, ao combinarmos o aquecimento no meio do chip com o resfriamento nas bordas, chegamos no gráfico acima. Como podemos ver, a temperatura atinge o máximo no centro, e o mínimo próximo às bordas, o que vai de acordo com as conclusões anteriores. Além disso, o gráfico mostra que a função respeita os valores nas fronteiras, mesmo que o tamanho do chip não seja unitário e a fronteira não seja homogênea.

4.4 Equilíbrio com variação de material

Neste exercício, foi preciso simular o que aconteceria caso o chip fosse formado por diferentes materiais. Nesta seção, usaremos o valores de condutividade térmica do silício: $148W/mK$, do cobre: $401W/mK$, do selênio: $2.04W/mK$ e do aerogel $0.013W/mK$. Os três primeiros são bastante usados na indústria elétrica e o último é um dos materiais com menores condutivade térmica e será usado apenas por curiosidade.

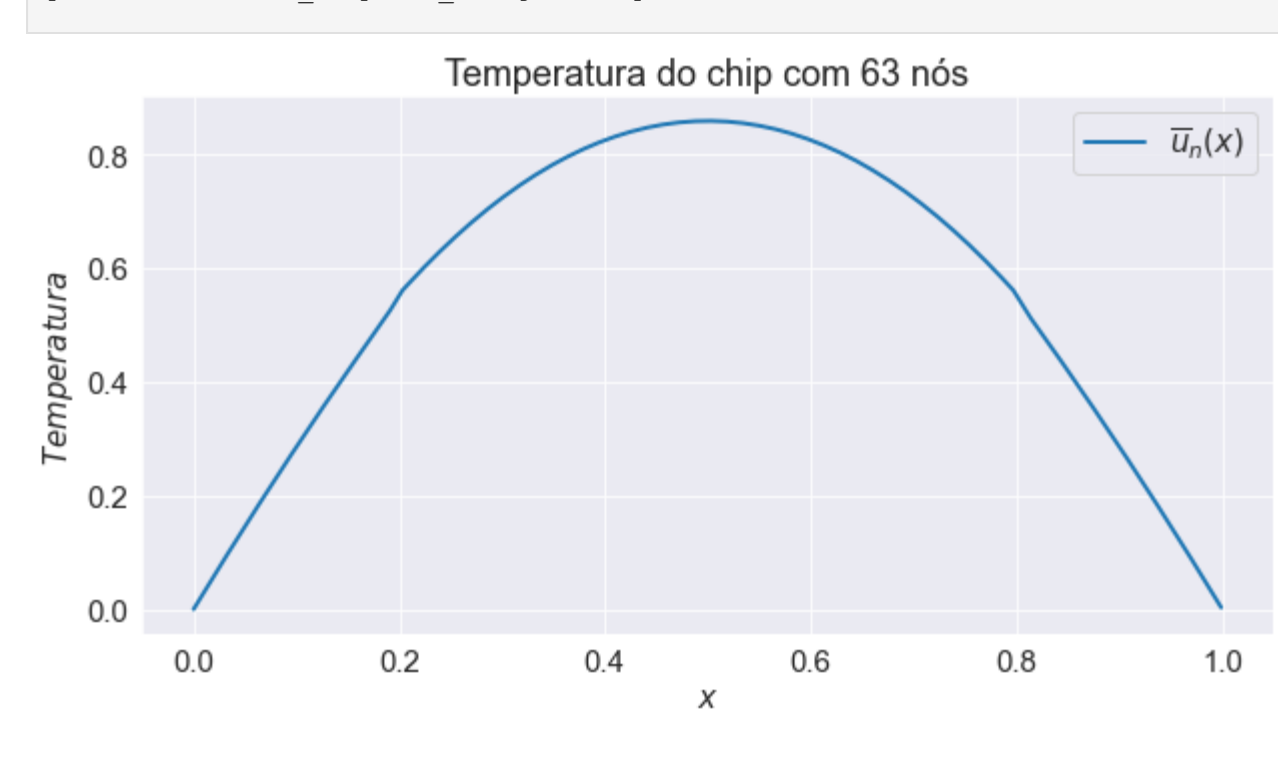
Fontes: Silício: https://pt.wikipedia.org/wiki/Sil%C3%ADcio, Selênio: https://pt.wikipedia.org/wiki/Sel%C3%AAnio, Cobre: https://pt.wikipedia.org/wiki/Cobre#:~:text=O%20cobre%20%C3%A9%20um%20elemento,encontra%2Dse%20no%20estado%20s%C3%B3lido Aerogel: https://www.thermal-engineering.org/pt-br/o-que-e-aerogel-definicao/#:~:text=Condutividade%20t%C3%A9rmica%20de%20aerogel&text=Os%20valores%20t%C3%ADpicos%20de%20condutividade,t%C

Silício e cobre

In [29]:
```
nos_chip = 63
nos_integral = 10
L = 1
a = 0
b = 0
d = 0.3
k = lambda x: 148 if ((L/2-d)<=x and x<=(L/2+d)) else 401
q = lambda x: 0
f = lambda x: 1000
```

Visualização

In [30]:
```
plotaSolucao(nos_chip,nos_integral,k,q,f,a,b,L)
```



Resultados

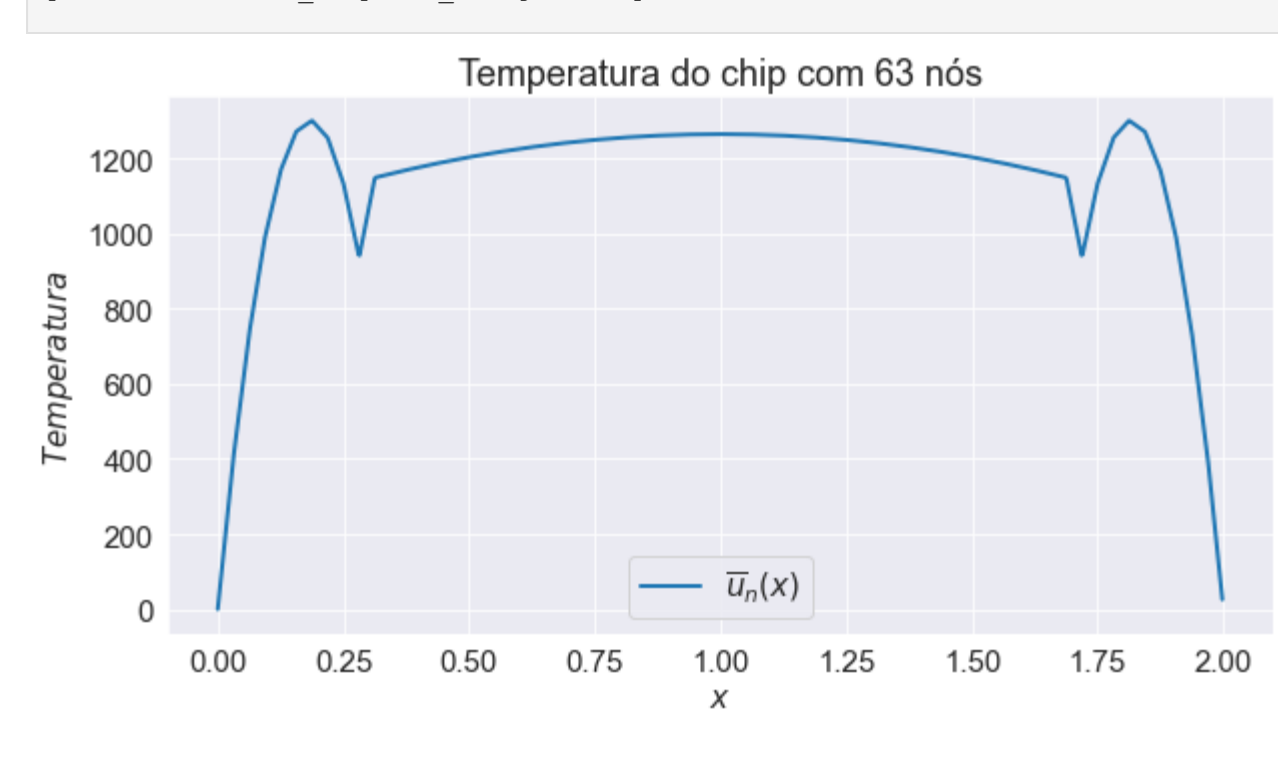
Podemos ver o efeito de ter um material com maior condutividade térmica junto a outra com menor. O Cobre tem maior k e por isso conduz mais calor, como podemos ver no centro do gráfico. Já o silício conduz menos calor e por isso é possível ver uma grande diminuição na temperatura do chip para o extremos.

Selênio e aerogel

In [31]:
```
nos_chip = 63
nos_integral = 10
L = 2
a = 0
b = 0
d = 0.7
k = lambda x: 2.04 if ((L/2-d)<=x and x<=(L/2+d)) else 0.013
q = lambda x: 0
f = lambda x: 1000
```

Visualização

In [32]:
```
plotaSolucao(nos_chip,nos_integral,k,q,f,a,b,L)
```



Resultados

O Selênio tem maior k e por isso conduz mais calor, como podemos ver nas extremidades do gráfico. Já o aerogel conduz menos calor e por isso é possível ver que pouco calor foi trocado no centro do chip, deixando a temperatura próxima de constante.

5 Conclusão

Neste exercício programa, nós pudemos aplicar o Método dos Elementos Finitos para calcular resolver uma equação diferencial e calcular a temperatura de um chip em diversas condições. Para isso, nós usamos resultados obtidos anteriormente para resolver Sistemas Tridiagonais e aproximar Integrais pelo Método de Gauss.

Os três exercícios de programação foram excelentes formas de aplicar os conceitos desenvolvidos ao longo do semestre, na matéria de Cálculo Numérico.