Relatório da Tarefa 3 de MAP3121

Gabriel Youssef Campos 10884301 Victor Nascimento Pereira - 10773530

0 Introdução

O objetivo desta tarefa é modelar o comportamento da difusão térmica de um chip, de dimensões $L \times L \times h$, sob um resfriador. Será considerada apenas uma dimensão para análise; tomamos h tal qual a variação da temperatura vertical é desprezível, com troca de calor perfeita entre as superfícies (chip e resfriador), e a base do chip não perde calor para o ambiente, portanto, analisaremos as variações de temperatura na direção x.

1 Parte teórica

O modelo utilizado para descrever a difusão térmica no conjunto chip+bloco é o seguinte:

$$\rho C \frac{\partial T(t,x)}{\partial t} = \frac{\partial}{\partial x}(k(x)\frac{\partial T(t,x)}{\partial x}) + Q(t,x), \tag{1}$$

obtido através da lei de Fourier e da conservação de energia, onde:

- T (t, x) é a temperatura do chip na posição x e instante de tempo t,
- ρ é a densidade do material do chip,
- C é o calor específico do material,
- k é o parâmetro de condutividade térmica do material,
- Q é uma fonte de calor, dada pela soma do calor gerado pelo chip (Q_+) com o calor retirado pelo resfriador (Q_-), onde Q = $Q_+ - Q_-$.

Q_+ será $Q_+(P,V)$, onde, P é a potência do chip e V seu volume, sendo $Q_+(P,V) = \frac{P}{V}$.

Para resolver (1) precisamos do estado inicial da distribuição de temperatura no chip, T(0,x), bem como T(t,0) e T(t,L) (fronteiras do chip). adotaremos que $T(t,0) = T(t,L) = T_{amb}$℃ ($T_{amb}$ é a temperatura ambiente).

1.1 Estado estacionário

Considerando que a temperatura está em estado de equilíbrio, ou seja,

$$\frac{\partial T(t,x)}{\partial t} = 0, \tag{2}$$

obteremos,

$$-\frac{\partial}{\partial x}(k(x)\frac{\partial T(x)}{\partial x}) = Q(x), \tag{3}$$

Se soubermos Q(x), assim como T(0) e T(L) (temperatura nos extremos), obteremos soluções de equilíbrio resolvendo numericamente as equações com o método dos elementos finitos.

1.2 Método dos Elementos Finitos

Sendo $\phi_1, \phi_2, ..., \phi_n$ bases de U_n, a resolução do problema parte da resolução do sistema linear:

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

Supondo ϕ_i e f, conhecidos, precisaremos resolver o sistema acima para encontrar o α_i. Com isso, a nossa solução aproximada, $\bar{u}_n$, será dada por $\sum_{i=1}^{n} \alpha_i \phi_i$.

1.3 Escolhendo espaço U_n e sua base: Elementos finitos

Escolhemos $U_n = S^0_{2,n}[0,1]$, ou seja, tomando $h = \frac{1}{(n+1)}$, $x_i = ih, i = 0, 1, ..., n+1$, e $s(0) = s(1) = 0$.

Temos então Splines Lineares igualmente espaçados se anulando nos extremos: $s(x_i) = f(x_i) = y_i$.

$S^0_{2,n}[0,1]$ simboliza diversos seguimentos de reta, sendo estas, $S_i = tg * x + \frac{(i+1)}{tg} + s_{i+1}, tg = \frac{s_i - s_{i+1}}{(x_{i+1} - x_i)}$, i = 1, 2, ..., n, portanto, podemos tomar os seguimentos de reta S_i como um espaço vetorial de dimensão n.

Como o produto escalar esta definido pela integral no intervalo, temos que:

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

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

$$< \phi_{i-1}, \phi_{i+1} >_L = \int_{x_{i-1}}^{x_{i+1}} \phi_{i-1}(x)\phi_{i+1}(x)dx = 0$$

portando, $U =$

$$\begin{bmatrix} \frac{2}{h} & -\frac{1}{h} & \cdots & 0 \\ -\frac{1}{h} & \vdots & \ddots & \vdots \\ 0 & \cdots & -\frac{1}{h} & \frac{2}{h} \end{bmatrix}$$

1.4 Montagem da Matriz e Solução do sistema

Utilizamos o seguinte algorítimo para aproximar os produtos internos via formula de Gauss de dois pontos:

```
1   import numpy as np
2   import math
3
4   from le_arquivo_ep3 import determinaNos
5
6   precisao_de_nos=10
7
8   def integra(f):
9       n=determinaNos(precisao_de_nos) #matriz com os valores dos nos (ni[0])
10                                                                  e pesos (ni[1])
11      soma1 = 0
12      for i in n:
13          soma1 += f(i[0])*i[1]
14      return soma1
```

Tendo os produtos internos calculados, utilizamos o seguinte algorítimo LU para resolver o sistema:

```
1   import numpy as np
2
3   def decomposicao_lu_tridiagonal(vetor_c, vetor_b, vetor_a, d):
4   # vetor_c     a subdiagonal de cima
5   # vetor_b     a diagonal principal
6   # vetor_a     a subdiagonal de baixo
7   # d     o lado direito da equa   o matricial Ax=d
8
9   n=len(vetor_b) # n     a quantidade de elementos da diagonal principal
10
11  # Constru   o da matriz A (tridiagonal) a partir de vetor_c, vetor_b e vetor_a.
12  A = np.zeros((n, n))
13
14  for i in range(0, n): # Gera   o da diagonal principal
15      A[i, i]=vetor_b[i]
16
17  for i in range(0, n-1): # Gera   o das diagonais secund rias
18      A[i,i+1]=vetor_c[i]
19      A[i+1,i]=vetor_a[i+1]
20
21  vetor_u = np.zeros(n)
22  vetor_l = np.zeros(n)
```

```
23
24      vetor_u[0]=vetor_b[0]
25
26      for i in range(1, n):
27          vetor_l[i]=vetor_a[i]/vetor_u[i-1]
28          vetor_u[i]=vetor_b[i]-vetor_l[i]*vetor_c[i-1]
29
30      #n1=len(vetor_l)
31
32      y = np.zeros(n)
33      x = np.zeros(n)
34
35      y[0]=d[0]
36      for i in range(1, n):
37          y[i]=d[i]-vetor_l[i]*y[i-1]
38
39      x[n-1]=y[n-1]/vetor_u[n-1]
40
41      for i in range(n-2, -1, -1):
42          x[i]=(y[i]-(vetor_c[i]*x[i+1]))/vetor_u[i]
43
44
45      return vetor_l, vetor_u, x
```

Sendo x o vetor solução do problema.

1.5 Solução do método de elementos finitos

A solução encontrada foi um $\bar{u}_n(x) = \sum_{i=1}^{n} \alpha_i \phi_i(x)$, que aproxima $u(x)$, e, sendo $u(x) \in C^2[0,1]$, teremos $\|\bar{u}_n(x) - u(x)\| = O(h^2)$, pois em cada passo a para a convergência estamos ignorando termos da ordem de h^2.

2 Variações do problema

2.1 Condições de fronteira não homogêneas

As condições de fronteiras tratadas até o momento eram, $T(t,0) = T(t,L) = T_{amb}$℃ ($T_{amb}$ é a temperatura ambiente), e $\frac{\partial T(t,x)}{\partial t} = 0$, tínhamos, $u(0) = u(1) = 0$. Agora, vamos tomar $u(0) = a$ e $u(1) = b$; assim, teremos uma nova equação para a resolução do problema, onde:

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

provar que $u(x) = v(x) + a + (b-a)x$

2.2 Intervalo [0, L]

O intervalo sendo tratado era $I = [0,1]$, generalizando para o intervalos $I_2 = [0,L]$, teremos um novo h, $h_2 = \frac{L}{(n+1)}$ e $x_i = ih, i = 0, 1..., n+1$.

poderemos agora definir $u(0) = a$ e $u(L) = b, v(0) = v(L) = 0$

definir produto escalar

mostrar nova matriz

3 Testes

3.1 Método de Ritz-Raleigth

3.2 Aplicação do nosso código à resolução do Exemplo 1 da seção 11.5 do livro texto Burden/Faires

Antes de implementarmos o método de elementos finitos, validamos o nosso método de Ritz - Raleigh usando o problema com valor limite mostrado em 4.

$$-\ddot{y} + \pi^2 y = 2\pi^2 sen(\pi x), 0 \leq x \leq 1, y(0) = y(1) = 0 \tag{4}$$

Utilizamos um valor de $n = 10$, com i variando de 0 a 9. A correspondência entre a notação que estamos utilizando e a equação 4 é dada por: $f(x) = 2\pi^2 sen(\pi x)$, $k(x) = 1$ e $q(x) = \pi^2$.

Resolvemos o sistema tridiagonal, encontramos os coeficientes α_i e, por fim, encontramos a solução aproximada $\bar{u}$.

Os coeficientes obtidos podem ser vistos em 1:

```
O vetor de coef. alpha eh:  [0.31028879 0.59020434 0.81234659 0.95497069 1.0041156  0.95497069
 0.81234659 0.59020434 0.31028879]
```

Figura 1: Vetor de coeficientes obtido para o exemplo do livro

Sabendo que, para a equação do exemplo, a solução exata é dada por $y = sen(\pi x)$, plotamos uma comparação entre a solução exata e a solução aproximada. O gráfico gerado pode ser visto na figura 2.

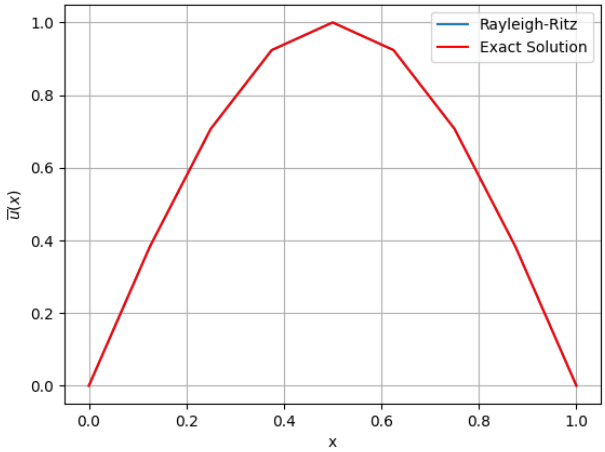

Figura 2: Comparação entre a solução aproximada e a solução exata do exemplo do livro.

3.3 Teste da validação do item 4.2 do enunciado do Exercício Programa

Para este teste teremos: $I = [0, 1], k(x) = 1, q(x) = 0, f(x) = 12x(1 - x) - 2$, e $u(0) = u(1) = 0$, como $u(x) = x^2(1 - x)^2$, podemos descrever o erro do método, onde erro$\equiv e = \|\bar{u}_n - u\| = max_{i=1,\ldots,n}|\bar{u}_n(x_i) - u(x_i)|$.

Foi utilizado o seguinte código para fazer as avaliações:

```
    # Valida  o: item 4.2 do enunciado.

import numpy as np
import matplotlib.pyplot as plt

from decomp_lu_tridiagonal import decomposicao_lu_tridiagonal
from funcoes_final import plotador, integrate, erroMax

L=0.02 # L=20 mm = 0.02 m
a=2 # u(0)=a
b=3 # u(L)=b

n=63
h=L/(n+1)
x_disc=np.arange(0,(L+0.0001),h)

def phi(i, x):
        if(x <= x_disc[i - 1]):
                return 0
        elif(x<=x_disc[i]):
                hi=(x_disc[i] - x_disc[i-1])
                return ((x-x_disc[i-1])/hi)
        elif(x<=x_disc[i+1]):
                hi=(x_disc[i+1]-x_disc[i])
                return ((x_disc[i + 1] - x) / hi)
```

```
26          else:
27                  return 0
28
29  def phi_diff(i, x):
30          if(x<=x_disc[i - 1]):
31                  return 0
32          elif(x<=x_disc[i]):
33                  return (1 / (x_disc[i] - x_disc[i - 1]))
34          elif(x<=x_disc[i + 1]):
35                  return (1 / (x_disc[i] - x_disc[i + 1]))
36          else:
37                  return 0
38
39  def f(x):
40      return 12*x*(1-x)-2
41
42  '''
43  def f_til(x):
44      # Implementando u(x)=v(x)+a+(b-a)*x, quando as condicoes de fronteira sao nao
45                                                                          homogeneas.
46      # Vamos resolver para f_{til}(x)=f(x)+(b-a)*k'(x)-q(x)*(a+(b-a)*x)
47      # k'(x)=0, para k(x)=1
48      return (f-q*(a+(b-a)))
49  '''
50
51  # k(x) corresponde ao p(x) do livro do Burden
52  def p(x):
53      return 1
54
55  def q(x):
56      return 0
57
58  def uExato(x):
59      return (x**2)*((1-x)**2)
60
61  # Geracao da matriz A do sistema Ax=d
62  def matA(x, i, j):
63          value = p(x)*phi_diff(i, x)*phi_diff(j, x)
64          value = value+q(x)*phi(i, x)*phi(j, x)
65          return value
66
67  def m(i, j):
68          i = i + 1
69          j = j + 1
70          return integrate(matA, 0, 1, i, j)
71
72  # Gera  o do vetor de coeficientes
73  def vetorD(x, i):
74          return (f(x)*phi(i, x))
75  def b(i):
76          i = i + 1
77          return integrate(vetorD, 0, 1, i)
78
79  A=np.zeros([n,n])
80
81  vecC=np.zeros(n)
82  vecB=np.zeros(n)
83  vecA=np.zeros(n)
84  vecD=np.zeros(n)
85
86  # Geracao dos vetores para uso na resolu  o do sist.
87  # a_{i,i+1} <=> vetor_c     a subdiagonal de cima
88  # Deve ser declarado com um zero no final para que possa chamar decomp_lu_tridiagonal.
89  # a_{i,i}   <=> vetor_b     a diagonal principal
90  # a_{i,i-1} <=> vetor_a     a diagonal de baixo
91  # Deve ser declarado com um zero no in cio para chamar decomp_lu_tridiagonal.
92  # b_{i}     <=> vetor_d     o lado direito da equacao matricial Ax=d
93
94  for i in range(0, n):
95          vecD[i]=b(i)
96          A[i, i] = m(i, i)
97          if (i - 1) >= 0:
98                  A[i, i - 1] = m(i, i - 1)
99          if (i + 1) < n:
100                 A[i, i + 1] = m(i, i + 1)
101
```

```
for i in range(0, n): # Gera  o da diagonal principal
        vecB[i]=A[i, i]

for i in range(0, n-1): # Gera  o das diagonais secund rias
        vecC[i]=A[i,i+1]
        vecA[i+1]=A[i+1,i]

l,u,alpha=decomposicao_lu_tridiagonal(vecC,vecB,vecA,vecD)

def uDotPhi(sol, x):
    v=0
    u=0
    i = 0
    for c in sol:
        v+=c*phi(i + 1, x)
        i = i + 1
    #u=v+a+(b-a)*x
    return v

x_plot=np.arange(0.0, (L+0.001), h)
uBarra=[]
for i in x_plot:
        uBarra.append(uDotPhi(alpha, i))

plotador(uBarra,x_plot,uExato)
erroMaximo = erroMax(uBarra, uExato)
print(erroMaximo)
```

O Código utiliza de base algumas funções, sendo estas:

```
    import numpy as np
import matplotlib.pyplot as plt  # pip install matplotlib


def plotador(eixoy, x, solExata):
    # plotting
    plt.grid()
    plt.plot(x, eixoy, x, solExata(x), 'r')
    plt.legend(['Rayleigh-Ritz', 'Exact Solution'])
    plt.xlabel(r'x')
    plt.ylabel(r'$\phi(x)$')
    plt.savefig('Results')
    plt.show()
    # plt.grid()
    #plt.loglog(np.linspace(0, N, N+1),error, 'g')
    #plt.title(r'\textbf{Error growth of the integrator}')
    # plt.xlabel(r'x')
    # plt.ylabel(r'Error')
    # plt.savefig('Error')
    # plt.show()


def quadGauss2NosSemTipo(func, a, b):
    no1 = np.sqrt(3)/3
    # peso=1
    soma = 0
    # Fator de escala para transportar do intervalo [-1,1] para [a,b]
    ftr = (b-a)/2
    if(a != -1 or b != 1):
        soma += ftr*func(no1*ftr+(a+b)/2, a, b)+ftr * \
            func(-no1*ftr+(a+b)/2, a, b)
    else:
        soma += func(no1, a, b)+func(-no1, a, b)
    return soma


def fDotPhi(N, phi, x):
    fEscalarPhi = np.zeros([N, N])
    for i in range(N):
        for j in range(N):
            if(x[i-1] <= x[j] <= x[i]):
                # phi[i,j]=(x[j]-x[i-1])/h
                fEscalarPhi[i] = quadGauss2NosSemTipo(
                    f*phi[i, :], x[i-1], x[i])
            elif(x[i] <= x[j] <= x[i+1]):
                # phi[i,j]=(x[i+1]-x[j])/h
```

```
                    fEscalarPhi[i] = quadGauss2NosSemTipo(
                        f*phi[i, :], x[i], x[i+1])
    return fEscalarPhi


def integrate(func, start, end, *args):
    steps = 10000
    h = ((end - start) / steps)
    half = h / 2
    value = 0
    for i in range(0, steps):
        dh = func(start + h * i + half, *args)
        if i == 0 or i == (steps - 1):
            value += dh
        else:
            value += (2 * dh)
    return (value * (h / 2))


def erroMax(aprox, exato):  # calcula o erro maximo em modulo
    erroMax = 0
    h = 1/(len(aprox)+1)
    for i in range(len(aprox)):
        if abs(exato(i*h)-aprox[i]) > erroMax:
            erroMax = abs(exato(i*h)-aprox[i])
    return erroMax
```

Sabendo que, para a equação de validação, a solução exata é dada por $u(x) = x^2(1-x)^2$, plotamos uma comparação entre a solução exata e a solução aproximada os valores de $n = 7$, 15, 31 e 63, que podem ser vistos nas figuras 3, 4, 5 e 6, respectivamente.

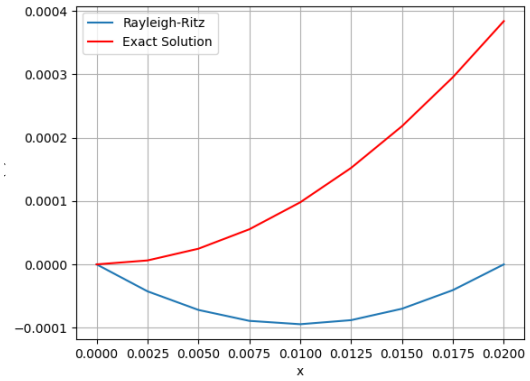

Figura 3: Comparação entre a solução aproximada e a solução exata da validação do item 4.2, para $n = 7$.

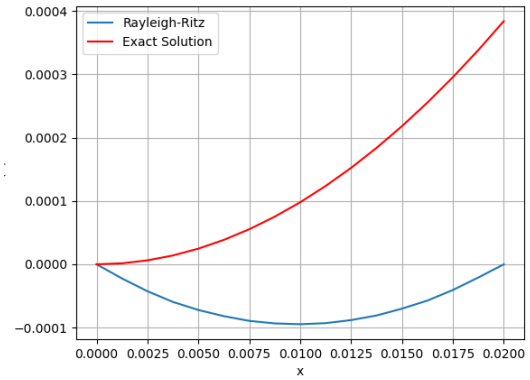

Figura 4: Comparação entre a solução aproximada e a solução exata da validação do item 4.2, para $n = 15$.

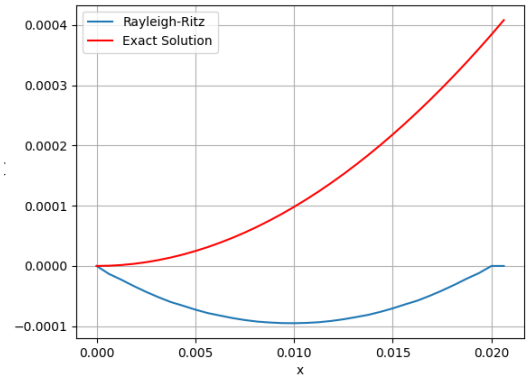

Figura 5: Comparação entre a solução aproximada e a solução exata da validação do item 4.2, para $n = 31$.

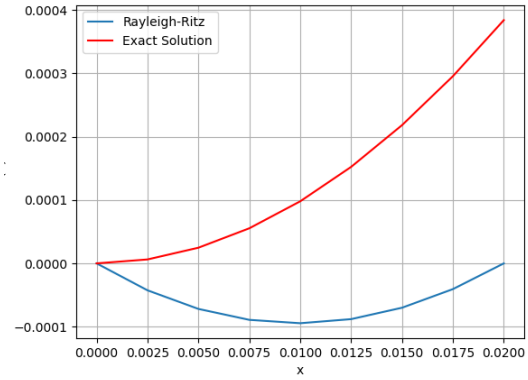

Figura 6: Comparação entre a solução aproximada e a solução exata da validação do item 4.2, para $n = 63$.

3.3.1 Erro Máximo em módulo para a validação acima

O Erro Máximo entre a aproximação calculada e a função da aproximação exata em modulo é aferido através do seguinte método:

```
1     def erroMax(aprox, exato): #calcula o erro maximo em modulo
2     erroMax = 0
3     h=1/(len(aprox)+1)
4     for i in range(len(aprox)):
5         if abs(exato(i*h)-aprox[i]) > erroMax:
6             erroMax = abs(exato(i*h)-aprox[i])
7     return erroMax
```

A tabela abaixo explicita os erros com relação aos ns:

n	erro Máximo em modulo
7	0.06258804902179672
15	0.06259298013969214
31	0.06249249743415325
63	0.06256873909034279

3.4 Teste do arquivo "complemento" enviado pelos professores como um adendo ao Exercício Programa

Para o teste do complemento teremos: $I = [0, 1], k(x) = e^x, q(x) = 0, f(x) = e^x + 1$, e $u(0) = u(1) = 0$, como $u(x) = (x - 1)(e^{-x} - 1)$, podemos descrever o erro do método, onde erro$\equiv e = \|\bar{u}_n - u\| = max_{i=1,\ldots,n}|\bar{u}_n(x_i) - u(x_i)|$.

Foi utilizado o seguinte código para fazer as avaliações:

```python
import numpy as np
import matplotlib.pyplot as plt

from decomp_lu_tridiagonal import decomposicao_lu_tridiagonal
from funcoes_final import plotador, integrate, erroMax

n = 7
h = 1/(n+1)
x_disc = np.arange(0, 1.0001, h)


def f(x):
    return np.exp(x)+1

# k(x) corresponde ao p(x) do livro do Burden
def p(x):
    return np.exp(x)


def q(x):
    return 0


def uExato(x):
    return (x-1)*((np.exp(-x))-1)


A = np.zeros([n, n])

vecC = np.zeros(n)
vecB = np.zeros(n)
vecA = np.zeros(n)
vecD = np.zeros(n)

# Geracao dos vetores para uso na resolu  o do sist.
# a_{i,i+1} <=> vetor_c     a subdiagonal de cima
# Deve ser declarado com um zero no final para que possa chamar decomp_lu_tridiagonal.
# a_{i,i}   <=> vetor_b     a diagonal principal
# a_{i,i-1} <=> vetor_a     a diagonal de baixo
# Deve ser declarado com um zero no in cio para chamar decomp_lu_tridiagonal.
# b_{i}     <=> vetor_d     o lado direito da equa  o matricial Ax=d

for i in range(0, n):
    vecD[i] = b(i)
    A[i, i] = m(i, i)
    if (i - 1) >= 0:
        A[i, i - 1] = m(i, i - 1)
    if (i + 1) < n:
        A[i, i + 1] = m(i, i + 1)

for i in range(0, n):  # Gera  o da diagonal principal
    vecB[i] = A[i, i]

for i in range(0, n-1):  # Gera  o das diagonais secund rias
    vecC[i] = A[i, i+1]
    vecA[i+1] = A[i+1, i]

l, u, alpha = decomposicao_lu_tridiagonal(vecC, vecB, vecA, vecD)


x_plot = np.arange(0.0, 1.0, 0.01)
uBarra = []
for i in x_plot:
    uBarra.append(function(alpha, i))

plotador(uBarra, x_plot, uExato)
erroMaximo = erroMax(uBarra, uExato)
print(erroMaximo)
```

O Código utiliza de base algumas funções, sendo estas as mesmas do código anterior.

Sabendo que, para a equação do complemento, a solução exata é dada por $u(x) = (x-1)e^{-x} - 1$, plotamos uma comparação entre a solução exata e a solução aproximada os valores de $n = 7$, 15, 31 e 63, que podem ser vistos nas figuras 7, 8, 9 e 10, respectivamente.

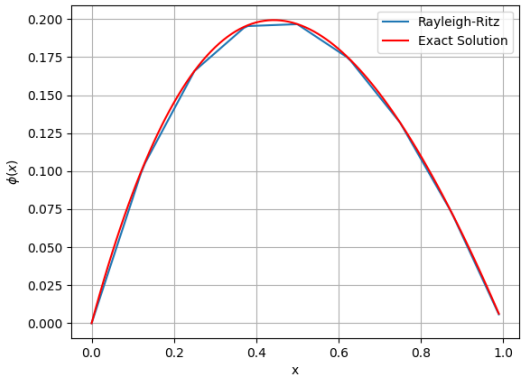

Figura 7: Comparação entre a solução aproximada e a solução exata do complemento do item 4.2, para $n = 7$.

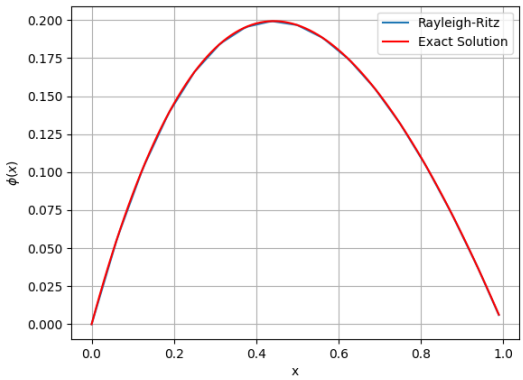

Figura 8: Comparação entre a solução aproximada e a solução exata do complemento do item 4.2, para $n = 15$.

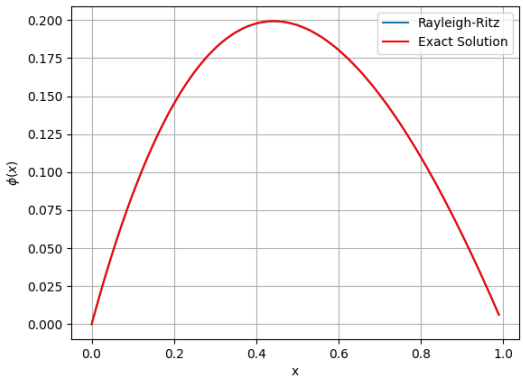

Figura 9: Comparação entre a solução aproximada e a solução exata do complemento do item 4.2, para $n = 31$.

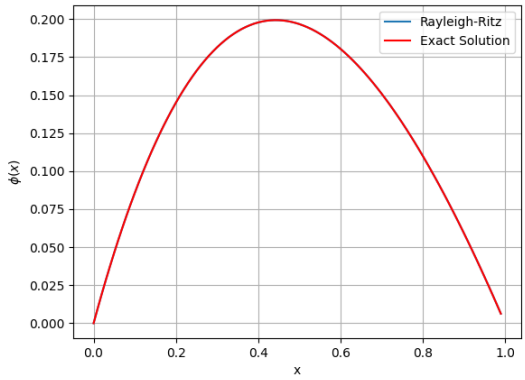

Figura 10: Comparação entre a solução aproximada e a solução exata do complemento do item 4.2, para $n = 63$.

3.4.1 Erro Máximo em módulo para o complemento acima

O Erro Máximo entre a aproximação calculada e a função da aproximação exata em modulo é aferido através do método mesmo método da anterior.

A tabela abaixo explicita os erros com relação aos ns:

n	erro Máximo em modulo
7	0.007057778469206716
15	0.0062831853413078155
31	0.006147526435723114
63	0.006078048835395258

3.5 Teste do equilíbrio com forçantes de calor do item 4.3 do enunciado do Exercício Programa

Para este segundo teste teremos: $I = [0, L], k(x) = k = 3,6W(xK)$;

$Q_+(x) = Q_+^0 e^{-(x-L/2)^2/\sigma^2}$

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

$Q = Q_+ - Q_-$

$Q = Q_+^0 e^{-(x-L/2)^2/\sigma^2} - Q_-^0 (e^{-x^2/\theta^2} + e^{-(x-L)^2/\theta^2}) = f(x)$

Percebemos que nesse teste não temos um valor de solução exata com o qual possamos comparar a resposta obtida. Assim, nossa análise se baseou em variar os parâmetros e θ das expressões acima para verificar os gráficos de distribuição de temperatura obtidos.

Foi utilizado o seguinte código para fazer as avaliações:

```
1
2       def k(x):
3       return 3.6
4
5       def Qmais(x):
6       return Q0_mais*(np.exp(-((x-L/2)**2)/(sigma**2)))
7
8       def Qmenos(x):
9       return Q0_menos*(np.exp(-((x)**2)/(theta**2))+np.exp(-((x-L)**2)/(theta**2)))
```

Tomando $I = [0, 1]$, $q(x) = 0$, $u(0) = u(1) = 0$, $Q_- = Q_+ = 1$, optamos, por método empírico, escolher dois valores de σ e dois valores de θ. Os gráficos obtidos podem ser vistos nas imagens abaixo, onde deixamos indicado na legenda os valores de σ e θ utilizados.

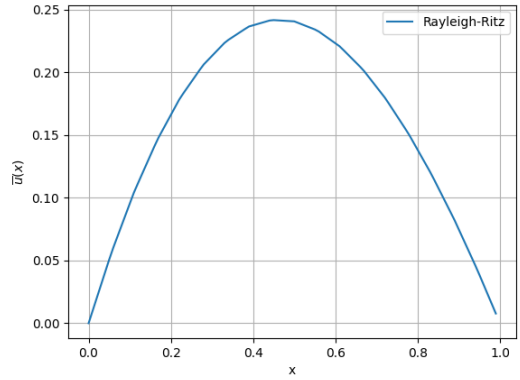

Figura 11: Distribuição de calor obtida para $\sigma = 1$ e $\theta = 1$, com $n = 17$.

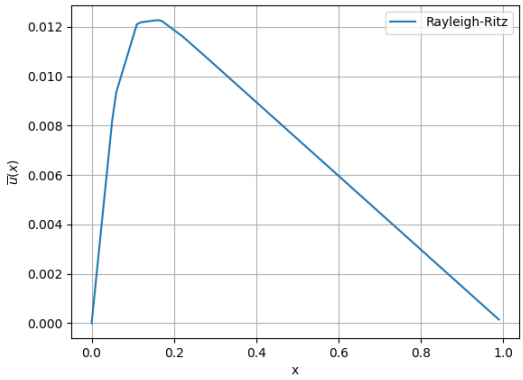

Figura 12: Distribuição de calor obtida para $\sigma = 0.1$ e $\theta = 0.1$, com $n = 17$.

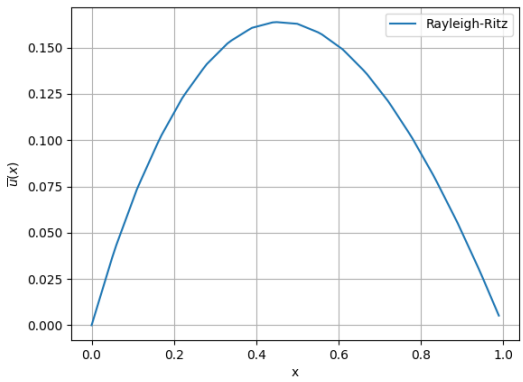

Figura 13: Distribuição de calor obtida para $\sigma = 0.1$ e $\theta = 1$, com $n = 17$.

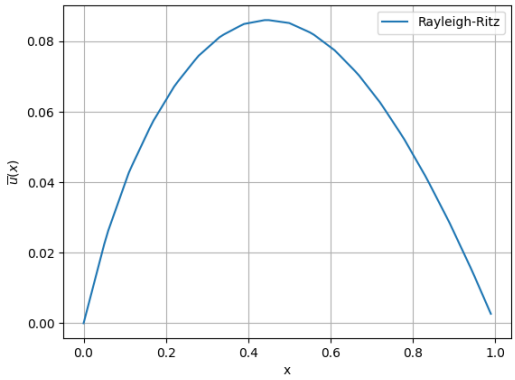

Figura 14: Distribuição de calor obtida para $\sigma = 1$ e $\theta = 0.1$, com $n = 17$.

Pelo que vemos nos gráficos acima, quando os valores das variâncias σ e θ diminuem em conjunto, por exemplo os dois em 0.1, o gráfico possui um pico indicando maior temperatura na extremidade esquerda da barra. Para outras combinações de variâncias (as duas maiores do que 1, ou uma diminuindo e a outra aumentando) a distribuição de temperatura parece não se alterar muito para um extremo específico do chip.

3.6 Teste do equilíbrio com variação de material do item 4.4 do enunciado do Exercício Programa

Para este segundo teste teremos: $k(x) = k_s, sex \in (L/2 - d, L/2 + d),\ k_a, casocontrário$

Com $k_s = 3.6W$, $k_a = 60W$, $d = 0,01$ e $n = 7$ temos:

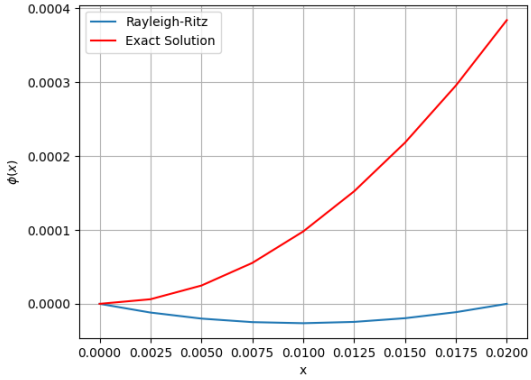

Figura 15: Distribuição de calor obtida para $d = 0.01$, com $n = 7$.

Com $k_s = 3.6W$, $k_a = 60W$, $d = 0,01$ e $n = 31$ temos:

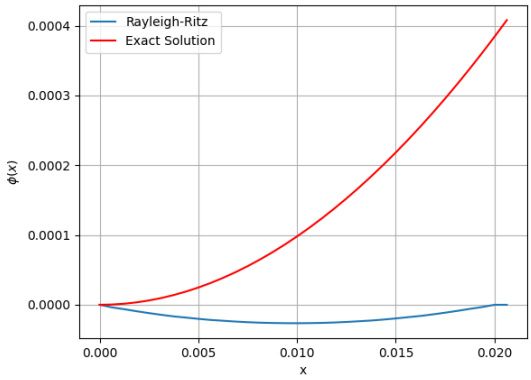

Figura 16: Distribuição de calor obtida para $d = 0.01$, com $n = 31$.

Com $k_s = 3.6W$, $k_a = 60W$, $d = 0,01$ e $n = 7$ temos:

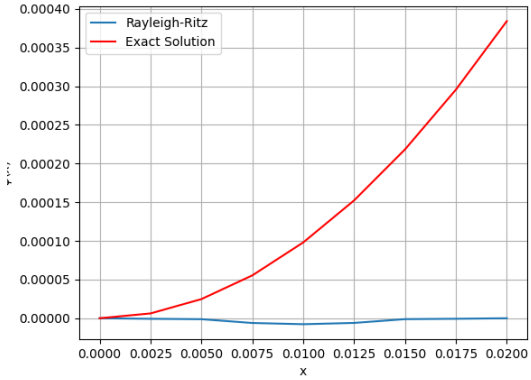

Figura 17: Distribuição de calor obtida para $d = 0.005$, com $n = 7$.

Com $k_s = 3.6W$, $k_a = 60W$, $d = 0,005$ e $n = 31$ temos:

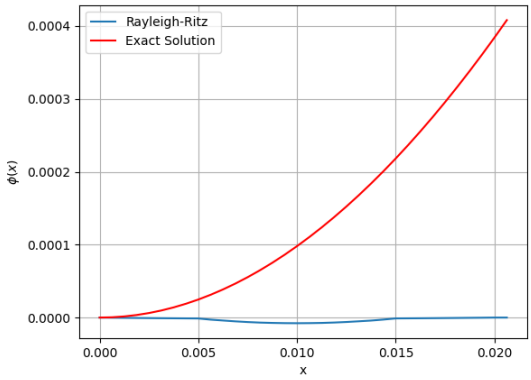

Figura 18: Distribuição de calor obtida para $d = 0.005$, com $n = 31$.

A partir dos gráficos podemos notar como a distribuição de calor foi alterada em se tratando dos trechos onde o material era diferente.

4 Conclusão

Após a elaboração deste EP pudemos chegar a conclusão de que, primeiramente, há muita dificuldade envolvendo o trabalho de pesquisa de métodos os quais não se teve contato anteriormente, após compreendido o método e colocado um modelo simples, outros modelos mais elaborados vão sendo aplicados com grau de dificuldade menor e o problema vai obtendo soluções mais precisas.

Pudemos observar, por exemplo em 3.4, o erro vinculado a solução exata sempre diminuindo com o aumento da precisão das análises, mas também, em 3.3, vimos uma estabilização do erro, onde as aproximações estavam bem próximas, mas dado o modelo para o problema e os fatores de propagação de erros, não havia grande ganho de precisão do resultado aumentando a precisão dos pontos de analise.

Após um vislumbre mais acurado do problema, com diversos fatores sendo testados, pudemos chegar a uma elaboração de modelos satisfatórios para o calculo numérico do problema proposto (propagação de calor num chip sob efeito de um cooler), que foram demonstrados e explicitados no corpo deste relatório.

Finalmente, gostaríamos de dizer que o método de Ritz - Railegh se torna quase secundário (dizemos isso com uma certa dose de exagero), frente ao método de elementos finitos, que acaba influenciando muito mais nos resultados obtidos.