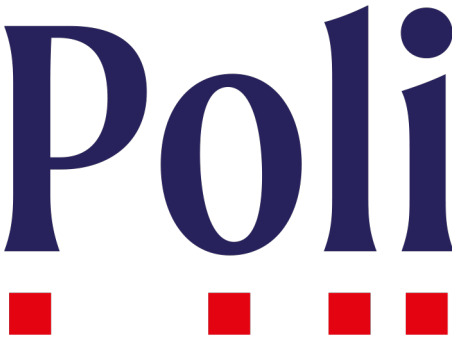


ESTUDO DA TRANSFERÊNCIA DE CALOR EM DISSIPADOR DE CALOR DE ALETAS RETAS POR MEIO DO MÉTODO DOS ELEMENTOS FINITOS IMPLEMENTADO EM PYTHON

Mateus Duarte de Oliveira

Projeto de Graduação apresentado ao Curso de Engenharia Mecânica da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Gustavo Rabello dos Anjos

Rio de Janeiro
Março de 2025

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Departamento de Engenharia Mecânica

DEM/POLI/UFRJ

ESTUDO DA TRANSFERÊNCIA DE CALOR EM DISSIPADOR DE CALOR DE ALETAS RETAS POR MEIO DO MÉTODO DOS ELEMENTOS FINITOS IMPLEMENTADO EM PYTHON

Mateus Duarte de Oliveira

PROJETO FINAL SUBMETIDO AO CORPO DOCENTE DO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO MECÂNICO.

Aprovada por:

_____

Prof. Gustavo Rabello dos Anjos, Ph.D.

_____

Prof. Albino José Kalab Leiroz, Ph.D.

_____

Prof. Fábio da Costa Figueiredo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2025

de Oliveira, Mateus Duarte

Estudo da transferência de calor em dissipador de calor de aletas retas por meio do Método dos Elementos Finitos implementado em Python/ Mateus Duarte de Oliveira. – Rio de Janeiro: UFRJ/Escola Politécnica, 2025.

XIII, 107 p.: il.; 29, 7cm.

Orientador: Gustavo Rabello dos Anjos

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia Mecânica, 2025.

Referências Bibliográficas: p. 66 – 68.

1. Dissipadores de calor. 2. Transferência de Calor. 3. Mecânica dos Fluidos. 4. Método dos Elementos Finitos. 5. Python. I. Rabello dos Anjos, Gustavo. II. Universidade Federal do Rio de Janeiro, UFRJ, Curso de Engenharia Mecânica. III. Estudo da transferência de calor em dissipador de calor de aletas retas por meio do Método dos Elementos Finitos implementado em Python.

"As coisas estão no mundo, só que eu preciso aprender."

Paulinho da Viola.

Agradecimentos

Agradeço primeiramente a Deus por ter me dado vida e saúde, e por todas as oportunidades que Ele pôs em meu caminho.

Agradeço à minha mãe, Rosângela, que nunca mediu esforços para me oferecer tudo o que estivesse em seu alcance para me ajudar a trilhar meus caminhos, mesmo em momentos de tamanha dificuldade. Agradeço também ao meu pai José Carlos, ao meu irmão Lucas, à minha avó Maria José e à minha avó Juraci (*in memoriam*).

Também agradeço a todos os companheiros de jornada que fiz na UFRJ, pela amizade e por todos os momentos que passamos, em cada café que tomamos nos quiosques do segundo andar do CT durante os intervalos. Agradeço em especial aos companheiros Mateus Mesquita e Lucas Schmidt que me influenciaram bastante na escolha do tema do projeto final e sua linha de pesquisa, e a Rodrigo Chueri, companheiro da Engenharia Mecânica e amigo de longa data.

Expresso também gratidão à KFC Projetos e Consultoria, em especial a João Mota, e à Equipe Icarus UFRJ de Formula SAE, por ter me introduzido ao mundo do CFD e da simulação computacional, com destaque especial à minha amiga e companheira de equipe Lara Castro.

Por fim, gostaria de agradecer a todo o corpo docente da Engenharia Mecânica da UFRJ, em especial ao meu orientador Professor Gustavo Rabello dos Anjos, por seu auxílio e paciência, além de suas aulas das disciplinas de Transferência de Calor II e Mecânica dos Fluidos e Transferência de Calor Computacional.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Mecânico

ESTUDO DA TRANSFERÊNCIA DE CALOR EM DISSIPADOR DE CALOR DE ALETAS RETAS POR MEIO DO MÉTODO DOS ELEMENTOS FINITOS IMPLEMENTADO EM PYTHON

Mateus Duarte de Oliveira

Março/2025

Orientador: Gustavo Rabello dos Anjos

Programa: Engenharia Mecânica

O projeto consiste em encontrar a quantidade ótima de aletas de um dissipador de calor de cobre de aletas retas usado para arrefecimento de um modelo de CPU por meio de convecção forçada com o ar externo. Este projeto se utiliza de um programa desenvolvido na linguagem de programação Python adequado à resolução da equação do calor tridimensional por meio do Método dos Elementos Finitos, dadas as condições de contorno de fluxo prescrito advindo da CPU e de convecção forçada com o ar em contato com o dissipador. Após as simulações de todos os dissipadores estudados, chegou-se ao modelo final do dissipador de calor com número ótimo de 53 aletas.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Mechanical Engineer

STUDY OF HEAT TRANSFER IN A PLATE FIN HEAT SINK USING THE FINITE ELEMENT METHOD IMPLEMENTED IN PYTHON

Mateus Duarte de Oliveira

March/2025

Advisor: Gustavo Rabello dos Anjos

Department: Mechanical Engineering

The project consists of determining the optimal number of fins for a straight-fin copper heat sink used to cool a CPU model through forced convection with external air. This project employs a program developed in the Python programming language, which is suitable for solving the three-dimensional heat equation using the Finite Element Method, given the boundary conditions of prescribed heat flux from the CPU and forced convection with the air in contact with the heat sink. After simulating all the studied heat sinks, the final model was determined to have an optimal number of 53 fins.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização do problema

Com as evoluções contínuas na indústria de dispositivos eletrônicos como computadores e *smartphones* impulsionada pelas novas tendências de mercado e necessidades dos consumidores, existe sempre a procura por parte da indústria de oferecer dispositivos mais potentes, mais rápidos e cada vez mais portáteis.

Com dispositivos cada vez menores e mais potentes, houve cada vez mais um aumento na geração de calor por parte de CPUs, GPUs, *chipsets* e módulos de memória RAM com o passar dos anos.

A geração de calor é inerente a todos os dispositivos eletrônicos, devido à eficiência não ser de 100%. Entretanto, o calor excedente deve ser corretamente dissipado para o ambiente a fim de evitar que o dispositivo sofra com superaquecimento, podendo causar a falha do componente, danificá-lo ou até queimá-lo, caso sua temperatura de junção, ou T_{JUNC} (temperatura da porção mais quente do chip) ou temperatura do encapsulamento, ou T_{CASE} (temperatura da superfície externa do dispositivo, a tampa) ultrapasse um limite máximo permitido indicado pelo fabricante, por exemplo.

1.1.1 O dissipador de calor

Comumente, o gerenciamento térmico de dispositivos eletrônicos é realizado por meio de um dissipador de calor. Os dissipadores de calor têm como princípio fundamental o aumento da área superficial disponível para troca térmica entre componente

eletrônico e o fluido de arrefecimento - normalmente o ar - reduzindo a resistência térmica associada e aumentando o fluxo de calor que sai do componente.

O dissipador de calor deve conferir a menor resistência térmica possível à condução do calor proveniente da CPU. Com isso, seu material deve apresentar altas condutividade térmica e capacidade térmica. Os dissipadores de calor são, portanto, fabricados comumente de alumínio ou cobre [12], este último permite espessuras menores ou aletas mais altas em comparação ao alumínio, por conta de sua maior condutividade térmica. Por sua vez, apesar de ter menor condutividade térmica em relação ao cobre, o alumínio tem a vantagem de ser mais barato.

Também são encontrados dissipadores de grafite, que alia uma condutividade térmica similar à do cobre porém é mais leve, e de diamante, com condutividade mais alta que a do cobre, porém bem mais caro.

Os dissipadores são fixados à tampa (ou *case*) da CPU a partir da TIM *(thermal interface material)*, também conhecida como pasta térmica ou composto térmico, como visto na figura 1.1. Além de fixar o dissipador à CPU, a TIM serve para preencher sulcos de ar entre as superfícies da base do dissipador de calor e da tampa da CPU, reduzindo drasticamente a resistência térmica de contato entre as superfícies causada por rugosidades, defeitos e desalinhamentos da interface. De acordo com SARVAR et. al. (2006) [13], A TIM pode ser uma pasta térmica (*thermal grease*), materiais de mudança de fase (*Phase Change Materials* ou PCMs), matrizes poliméricas preenchidas (elastômeros) ou materiais à base de carbono.

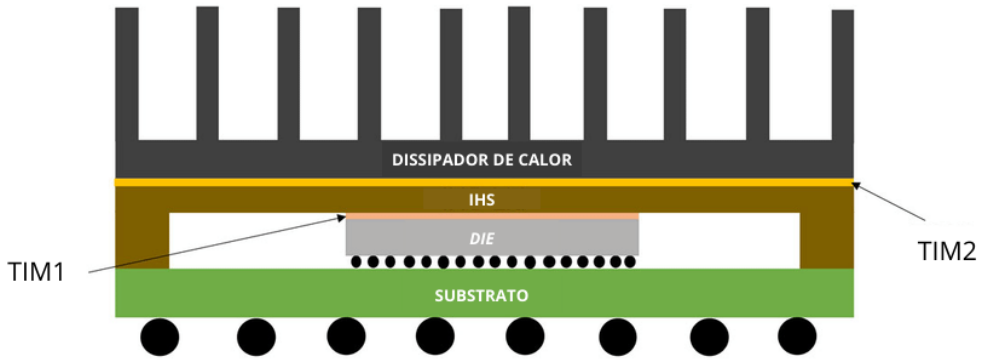


Figura 1.1: Montagem do dissipador de calor sobre uma CPU. Adaptada de [1].

Os dissipadores de calor podem ser classificados de acordo com o tipo de convecção. Dissipadores de calor passivos dependem da convecção natural para dissipar

o calor do componente, sem o uso de ventiladores. Neste caso, o movimento do ar ocorre devido à força de empuxo gerada pelo gradiente de densidade causado pelo aquecimento do ar mais próximo do dissipador. Em dissipadores de calor ativos, um ventilador ou soprador podem ser utilizados para gerar um escoamento forçado de ar entre as aletas do dissipador, gerando coeficientes de transferência de calor mais altos com relação a dissipadores passivos e retirando mais calor do sistema. Por sua vez, dissipadores de calor híbridos utilizam sistemas de controle para que o ventilador ou soprador permaneça inativo enquanto o sistema opere em temperaturas mais baixas. Quando a temperatura do sistema aumenta a partir de certo nível, o modo de convecção passa a ser forçado.

Dissipadores de calor passivos são normalmente encontrados em microcontroladores, microprocessadores e *chipsets*. Dissipadores ativos, por sua vez, são comuns em GPUs e CPUs [14]. Os dissipadores de calor ativos, passivos ou híbridos podem ser autônomos ou integrados a câmaras de vapor ou tubos de calor (*heat pipes*), como na figura 1.2.

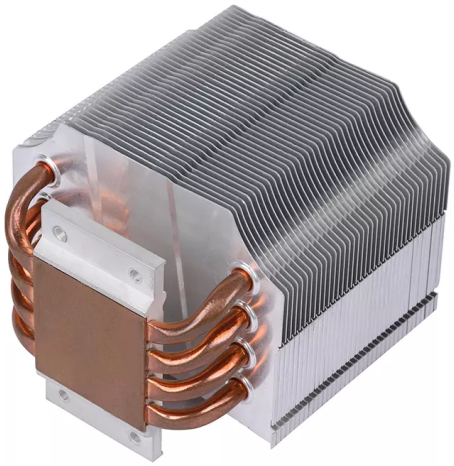


Figura 1.2: Dissipador de aletas retas combinado com *heat pipe*. De [2].

Dissipadores de calor são constituídos de uma base sobre a qual existem aletas para aumentar a área de troca térmica com o ar externo. As aletas podem ser uma parte integral da base ou serem montadas separadamente. Existem diversas geometrias diferentes para dissipadores de calor com base no arranjo das aletas, sendo as mais comuns dissipadores de calor de placas retas e dissipadores de calor de pinos.

Contudo, outras geometrias também são encontradas, como dissipadores de calor de aletas alargadas. A figura 1.3 exemplifica os três tipos de aletas supracitados.

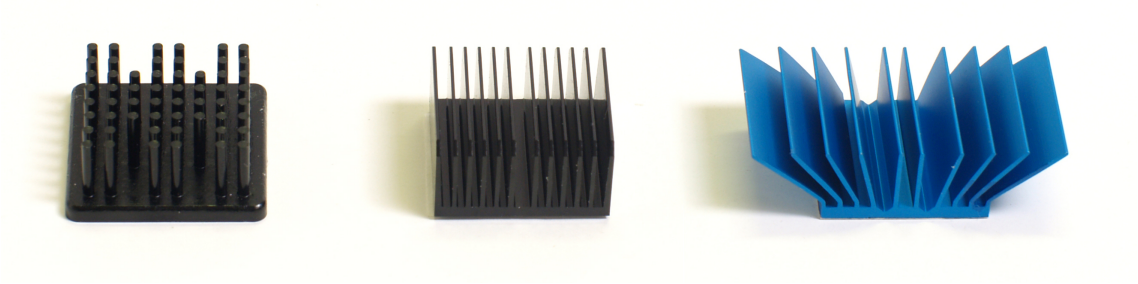


Figura 1.3: Dissipadores de calor de pinos, de aletas retas e de aletas alargadas, da esquerda para a direita. De [3].

Dissipadores de aletas retas são mais compactos e oferecem mais performance para convecção forçada por apresentarem mais área. Por sua vez, dissipadores de pinos podem ser mais vantajosos caso o fluxo seja multidirecional [15]. Além disso, são mais fáceis de limpar.

Os métodos de fabricação de dissipadores de calor mais utilizados são citados em LEE (1995) [16]. Os principais métodos de fabricação são: estampagem das chapas de alumínio ou cobre (*stamping*), extrusão de bloco sólido (*extrusion*), colagem com epóxi condutor de chapas extrudadas em base extrudada ranhurada (*bonding*), fundição (*casting*) e soldagem ou brasagem de chapa dobrada em arranjo de serpentina sobre a base do dissipador (*folding*).

Com relação à montagem dos dissipadores na PCB, em dissipadores pequenos costuma-se usar pastas térmicas. Para dissipadores grandes, pode ser usado epóxi, clipes e pinos [17].

1.2 Motivação

COPELAND (2000) [18] estudou experimentalmente dissipadores com aletas de diferentes espessuras e calculou as dimensões ótimas do passo entre aletas (*fin pitch*) para as seguintes condições: velocidade de aproximação fixa, perda de carga fixa, potência do ventilador fixa e resistência térmica fixa. Observou para a condição de potência fixa do ventilador, nas espessuras de aletas estudadas, a presença de um passo entre aletas ótimo, no qual a resistência térmica é mínima.

Fixando-se a potência do ventilador, a vazão de ar que passa pelo dissipador de calor corresponde à interseção da curva do sistema (do dissipador) com a curva do ventilador - o ponto de trabalho. Dito isto, a vazão volumétrica de ar entre as aletas do dissipador e, consequentemente, a velocidade do ar entre as aletas e o coeficiente convectivo serão funções do número de aletas, dado que a altura, largura e comprimento do dissipador, bem como a espessura das aletas se mantenham fixas.

Nessa condição, ao se adicionar aletas a um dissipador de calor de aletas paralelas, aumenta-se a área de troca térmica convectiva junto ao ambiente e também o coeficiente de transferência de calor até certo ponto. Contudo, pode-se verificar também um aumento na perda de carga do escoamento de ar sobre as aletas e uma redução em sua vazão mássica e velocidade em cada canal formado por aletas adjacentes, reduzindo assim o coeficiente de transferência de calor convectiva a partir de um dado número de aletas.

Existe, portanto um número ótimo de aletas na geometria, correspondente ao ponto no qual a resistência térmica do sistema é a mínima e, consequentemente, a temperatura no componente é a mínima.

1.3 Objetivo

O presente trabalho tem como objetivo obter o número ótimo de aletas de um dissipador de calor de cobre de aletas paralelas utilizado para dissipar o calor gerado pelo processador Intel Xeon 6710E - resfriado por convecção forçada com uso do ventilador San Ace 9CRH0648P6G001 da Sanyo Denki - ao mesmo tempo que se controla a máxima temperatura da tampa $T_{\text{CASE, MÁX}}$ retirada da folha de dados do fabricante - Intel [19].

Para encontrar o número ótimo de aletas, a temperatura média no fundo do dissipador é monitorada como parâmetro de otimização, para cada número de aletas.

Para este estudo, utiliza-se um programa desenvolvido pelo autor na linguagem de programação *Python* a fim de resolver a equação de calor tendo como domínio o dissipador de calor, de modo a se obter o perfil de temperatura no mesmo por meio do MEF - Método dos Elementos Finitos.

Para uso deste programa, tanto a geometria representativa do domínio, como a

sua discretização (malha) são feitas no *Gmsh*. Depois de feitas as simulações, seus resultados são visualizados no *Paraview*.

1.4 Organização do texto

- No capítulo 2 será mostrada toda a fundamentação teórica necessária para desenvolvimento do projeto. Será discorrido sobre fundamentos da Transferência de Calor e do Método dos Elementos Finitos, além de fundamentação acerca da perda de carga em escoamentos em canais e de uma teoria básica de ventiladores. Ao final, serão apresentadas as correlações usadas para obter-se o coeficiente de transferência de calor médio por convecção e a perda de pressão para o dissipador do presente trabalho.

- No capítulo 3 será apresentada toda a metodologia do presente trabalho além de toda a modelagem física, matemática e numérica do problema proposto, bem de sua implementação computacional em *Python*, parâmetros das simulações e controle de resultados.

- No capítulo 4 o código do programa será validado a partir da verificação do mesmo usando uma solução analítica de um problema retirado de um livro de condução de calor [10] contendo as mesmas condições de contorno do problema proposto.

- No capítulo 5, os resultados das simulações serão apresentados e será realizada uma discussão sobre os perfis de temperatura do dissipador de calor e sua dependência do seu número de aletas. Também serão mostradas figuras contendo o perfil de temperatura da geometria que itimiza a transferência de calor desde o dissipador, obtidas pelo *Paraview*.

- No capítulo 6, o trabalho será concluído. Além disso, serão apresentadas as limitações do mesmo, e sugestões para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Transferência de Calor

Segundo VAN WYLEN, SONNTAG e BORGNAKKE (1998) [20], calor é definido como sendo a forma de transferência de energia através da fronteira de um sistema a uma dada temperatura a outro sistema - ou o meio - em uma temperatura inferior, quando os sistemas são postos em contato térmico. Ou seja, o fenômeno de transferência de calor ocorre em razão unicamente da diferença de temperatura entre os sistemas. Além disso, trata-se de um fenômeno dito *transitório* - uma vez que pode somente ser identificado quando atravessa a fronteira do sistema.

De acordo com ÖZİŞİK (1985) [21], a ciência da *Termodinâmica* trata da relação entre o calor e outras formas de energia. Por sua vez, a ciência da *Transferência de Calor* se preocupa com a análise da taxa de transferência de calor de um sistema, buscando calcular o fluxo de calor a partir da distribuição de temperatura do meio, grandezas de grande interesse para a Engenharia.

A transferência de calor se dá por três mecanismos diferentes, que na prática se dão simultaneamente, e que serão discorridos nas próximas subseções: *condução*, *convecção* e *radiação*.

2.1.1 Condução

A condução de calor se dá pela movimento cinético ou pela colisão direta de moléculas em fluidos em repouso ou pelo movimento de elétrons, no caso de metais [21].

O fluxo de calor por condução $\mathbf{q}_{\text{cond}}$ é descrito matematicamente pela Lei de Fourier, sendo proporcional ao gradiente de temperatura $\boldsymbol{\nabla} T$ e diretamente proporcional à condutividade térmica k, da seguinte forma:

$$\mathbf{q}_{\text{cond}} = -k\nabla T \tag{2.1}$$

2.1.2 Convecção

Segundo BEJAN (1992) [11], convecção é o processo de transferência de calor causado pelo escoamento de um fluido, estando esse fluido em contato com um sólido.

De acordo com VAN WYLEN, SONNTAG e BORGNAKKE (1998) [20], o escoamento desloca matéria numa dada temperatura sobre uma superfície sólida a uma temperatura diferente da do fluido. Desta forma, a condução no sólido é influenciada pela natureza do escoamento, isto é, a taxa de transferência de calor é dependente de como é realizado o contato térmico entre a superfície e o meio.

A convecção pode ser do tipo *forçada* através do escoamento forçado de fluido pela ação de máquinas de fluxo, por exemplo, ou do tipo *natural*, no qual os gradientes de temperatura no meio causam gradientes de densidade e, por consequência, de pressão, ocasionando o movimento do fluido por forças de empuxo.

O fluxo de calor $\mathbf{q}_{\text{conv}}$ trocado por convecção se relaciona com a diferença entre as temperaturas do meio e da superfície do sólido de acordo com a Lei de Resfriamento de Newton da seguinte forma:

$$\mathbf{q}_{\text{conv}} = h_{\text{conv}}(T_w - T_\infty) \tag{2.2}$$

Onde T_w é a temperatura da superfície do sólido, T_∞ é a temperatura do meio fluido, e h_{conv} é o coeficiente de transferência de calor por convecção.

De modo contrário à condutividade térmica, o coeficiente de transferência de calor por convecção não é uma propriedade da substância, e sim um parâmetro experimental que depende da velocidade do escoamento, das propriedades do fluido, da geometria do sólido, entre outros fatores. Em geral, os coeficientes de transferência de calor por convecção natural são mais baixos do que no caso de convecção forçada.

2.1.3 Radiação

Outro mecanismo de transferência de calor é a radiação, na qual energia é transmitida em forma de ondas eletromagnéticas. Tal transmissão, diferentemente das demais, pode ocorrer no vácuo, mas é necessário um meio material para que haja emissão (geração) e absorção de energia [20].

O fluxo de energia por radiação - E(T) - é dado como fração da emissão de um corpo negro perfeito (calculada a partir da Lei de Stefan-Boltzmann) tal como se segue:

$$\mathbf{E(T)} = \epsilon\sigma T_s^4 \tag{2.3}$$

Onde T_s é a temperatura da superfície, σ é a constante de Stefan-Boltzmann, que vale $5{,}67 \times 10^{-8}\,\mathrm{W/m^2K^4}$, e $0 \leq \epsilon \leq 1$ é a emissividade do corpo.

2.1.4 A equação do calor

BIRD (1960) [22] propôs um procedimento para se chegar à equação do calor através da análise de um volume de controle estacionário (ver figura 2.1) através do qual escoa um fluido incompressível.

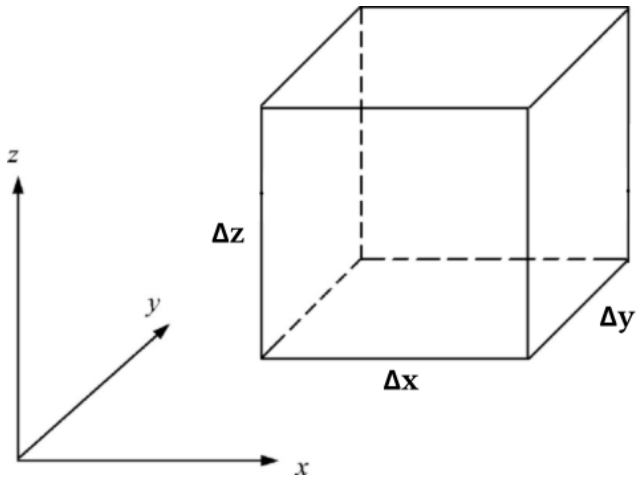


Figura 2.1: Volume de controle cúbico.

O procedimento consiste em aplicar-se a Primeira Lei da Termodinâmica para um sistema aberto em regime transiente a fim de se obter a *equação da energia* para o volume de controle.

Paralelamente, a *equação da energia mecânica* pode ser deduzida ao se efetuar o produto escalar da velocidade local $\mathbf{v}$ com a equação de movimento fruto de um balanço de momento linear no volume de controle.

Subtraindo-se a equação da energia mecânica à equação da energia, obtemos a *equação da energia interna* (ou térmica), que pode, por sua vez, ser escrita em termos da temperatura e calor específico.

Equação da energia

A lei de conservação de energia para o fluido de massa específica ρ contido no volume de controle cúbico de dimensões Δx, Δy e Δz da figura 2.1 sob ação da aceleração da gravidade $\mathbf{g}$ e sujeito a pressão p pode ser escrita de forma compacta em notação tensorial, para um observador fixo no espaço, da seguinte forma:

$$\underbrace{\frac{\partial}{\partial t}\rho(u+\tfrac{1}{2}v^2)}_{\substack{\text{tx. de ganho de energia}\\ \text{por un. de volume}}} = - \underbrace{(\boldsymbol{\nabla}\cdot\rho\mathbf{v}(u+\tfrac{1}{2}v^2))}_{\substack{\text{tx. de entrada de energia por}\\ \text{convecção por un. de volume}}} - \underbrace{(\boldsymbol{\nabla}\cdot\mathbf{q})}_{\substack{\text{tx. de entrada de energia por condução}\\ \text{por un. de volume}}} + \underbrace{\rho(\mathbf{v}\cdot\mathbf{g})}_{\substack{\text{tx. de trab. realizado no fluido}\\ \text{por forças grav. por un. de volume}}} - \underbrace{(\boldsymbol{\nabla}\cdot(p\mathbf{v}))}_{\substack{\text{tx. de trab. realizado no fluido}\\ \text{por forças de pressão por un. de volume}}} - \underbrace{(\boldsymbol{\nabla}\cdot[\boldsymbol{\tau}\cdot\mathbf{v}])}_{\substack{\text{tx. de trab. realizado no fluido}\\ \text{por forças viscosas por un. de volume}}} \tag{2.4}$$

Onde $(u+\frac{1}{2}v)$ representa a taxa de acúmulo de energia cinética mais energia interna por unidade de massa para um elemento de fluido escoando através do volume de controle, $\mathbf{q}$ é o fluxo de calor que entra ou sai do mesmo por condução, $\boldsymbol{\tau}$ é o tensor das tensões viscosas e $\boldsymbol{\nabla}\cdot\boldsymbol{\phi}$ é o operador Divergente.

Expandindo-se o primeiro termo do lado direito da equação 2.4, temos:

$$\rho\left[\frac{\partial}{\partial t}(u+\tfrac{1}{2}v^2)+(\mathbf{v}\cdot\boldsymbol{\nabla}(u+\tfrac{1}{2}v^2))\right]+(u+\tfrac{1}{2}v^2)\left[\frac{\partial\rho}{\partial t}+(\boldsymbol{\nabla}\cdot\rho\mathbf{v})\right] = -(\boldsymbol{\nabla}\cdot\mathbf{q})+\rho(\mathbf{v}\cdot\mathbf{g})-(\boldsymbol{\nabla}\cdot p\mathbf{v})-(\boldsymbol{\nabla}\cdot[\boldsymbol{\tau}\cdot\mathbf{v}]) \tag{2.5}$$

Neste caso, o segundo termo do lado esquerdo da equação 2.5 é nulo pela consideração do fluido ser incompressível. Reescrevendo a equação 2.5 usando a definição de derivada material, temos a forma final da equação da energia:

$$\rho\frac{D}{Dt}(u+\tfrac{1}{2}v^2) = -(\boldsymbol{\nabla}\cdot\mathbf{q}) + \rho(\mathbf{v}\cdot\mathbf{g}) - (\boldsymbol{\nabla}\cdot p\mathbf{v}) - (\boldsymbol{\nabla}\cdot[\boldsymbol{\tau}\cdot\mathbf{v}]) \tag{2.6}$$

Equação da energia mecânica

De modo análogo à dedução da equação da energia, o balanço de momento linear no mesmo volume de controle estacionário de volume $\Delta x\Delta y\Delta z$ da figura 2.1 através do qual um fluido escoa nos leva à equação de momento linear para um observador se movendo juntamente ao escoamento de fluido.

$$\underbrace{\rho\frac{D\mathbf{v}}{Dt}}_{\substack{\text{massa específica vezes}\\ \text{aceleração}}} = - \underbrace{\boldsymbol{\nabla}p}_{\substack{\text{força de pressão no V.C.}\\ \text{por un. de volume}}} - \underbrace{[\boldsymbol{\nabla}\cdot\boldsymbol{\tau}]}_{\substack{\text{força viscosa no V.C. por}\\ \text{un. de volume}}} + \underbrace{\rho\mathbf{g}}_{\substack{\text{força gravitacional no}\\ \text{V.C. por un. de volume}}} \tag{2.7}$$

Ao se fazer o produto escalar da velocidade local $\mathbf{v}$ do fluido com a equação 2.7, obtemos uma equação que descreve o balanço de energia cinética para um elemento fluido.

$$\rho\frac{D}{Dt}(\tfrac{1}{2}v) = -(\mathbf{v}\cdot\boldsymbol{\nabla}p) - (\mathbf{v}\cdot[\boldsymbol{\nabla}\cdot\boldsymbol{\tau}]) + \rho(\mathbf{v}\cdot\mathbf{g}) \tag{2.8}$$

Abrindo o termo de pressão e o termo viscoso usando identidades vetoriais, temos a seguir a versão final da equação da energia mecânica para um observador solidário ao escoamento de fluido na equação 2.9.

$$\begin{aligned}\underbrace{\frac{D}{Dt}(\tfrac{1}{2}\rho v)}_{\substack{\text{tx. de ganho de energia cinética}\\ \text{por un. de volume}}} = &- \underbrace{(\boldsymbol{\nabla}\cdot p\mathbf{v})}_{\substack{\text{tx. de trab. realizado pela pressão}\\ \text{das vizinhanças no V.C.}}} - \underbrace{p(-\boldsymbol{\nabla}\cdot\mathbf{v})}_{\substack{\text{tx. de conversão } \textit{reversível}\\ \text{para energia interna}}}\\ &- \underbrace{(\boldsymbol{\nabla}\cdot[\boldsymbol{\tau}\cdot\mathbf{v}])}_{\substack{\text{tx. de trab. realizado por forças}\\ \text{viscosas no V.C.}}} - \underbrace{(-\boldsymbol{\tau}:\boldsymbol{\nabla}\mathbf{v})}_{\substack{\text{tx. de conversão } \textit{irreversível}\\ \text{para energia interna}}}\\ &+ \underbrace{\rho(\mathbf{v}\cdot\mathbf{g})}_{\substack{\text{tx. de trab. realizado no fluido}\\ \text{por forças grav. por un. de volume}}}\end{aligned} \tag{2.9}$$

Equação do calor

A equação da energia interna é obtida subtraindo-se a equação da energia mecânica 2.9 da equação da energia tal como escrita em 2.6, tal como se segue.

$$\underbrace{\rho\frac{Du}{Dt}}_{\substack{\text{tx. de ganho de energia}\\ \text{interna por un. de volume}}} = -\underbrace{(\boldsymbol{\nabla}\cdot\mathbf{q})}_{\substack{\text{tx. de entrada de energia por}\\ \text{condução por un. de volume}}} - \underbrace{p(\boldsymbol{\nabla}\cdot\mathbf{v})}_{\substack{\text{tx. de aum. de energia } \textit{reversível}\\ \text{por compressão por un. de volume}}} - \underbrace{(\boldsymbol{\tau}:\boldsymbol{\nabla}\mathbf{v})}_{\substack{\text{tx. de aum. de energia}\\ \textit{irreversível} \text{ por dissipação viscosa}}} \tag{2.10}$$

Onde u é a energia interna do fluido. Usando o conceito de calor específico juntamente à Lei de Fourier - equação 2.1 - e expressando $\boldsymbol{\tau}$ em termos de gradientes de velocidade, temos a equação do calor para um fluido newtoniano com condutividade térmica constante.

$$\rho c_v\frac{DT}{Dt} = k\boldsymbol{\nabla}^2T - T\left(\frac{\partial p}{\partial T}\right)_\rho(\boldsymbol{\nabla}\cdot\mathbf{v}) + \mu\Phi_v \tag{2.11}$$

Onde Φ_v é a chamada função dissipação, c_v é o calor específico a volume constante do fluido, μ é sua viscosidade dinâmica e $\boldsymbol{\nabla}^2\phi$ é o operador Laplaciano. O último termo da direita da equação 2.11 representa a geração de calor por dissipação viscosa.

Reescrevendo a equação do calor para sólidos (em sólidos, o termo convectivo da derivada material se anula) e generalizando o termo fonte de gerações internas de calor Q_{ger}, obtemos a versão final da equação do calor.

$$\frac{\partial T}{\partial t} = \alpha\boldsymbol{\nabla}^2T + \frac{Q_{ger}}{\rho c_p} \tag{2.12}$$

Onde $\alpha = \frac{k}{\rho c_p}$ é a difusividade térmica do material e c_p é seu calor específico a pressão constante.

Em regime permanente, não é necessário usar a definição de difusividade térmica, que é associada à variação temporal da temperatura, dando origem à equação de Poisson:

$$k\boldsymbol{\nabla}^2 T + Q_{ger} = 0 \tag{2.13}$$

De acordo com ÖZİŞİK (1985) [21], a equação do calor pode ter três diferentes condições de contorno a fim de especificar a condição térmica nas superfícies das fronteiras do domínio. Elas são ditas de Dirichlet (de primeira espécie, na qual a temperatura é prescrita), de Neumann (de segunda espécie, na qual o fluxo de calor é prescrito), e de Robin (de terceira espécie, na qual existe convecção), apresentadas a seguir nas equações 2.14:

$$T = T_i, \qquad \text{em } \Gamma_d \tag{2.14a}$$

$$k\nabla T \cdot \mathbf{n} = -q, \qquad \text{em } \Gamma_n \tag{2.14b}$$

$$k\nabla T \cdot \mathbf{n} = hT_\infty - hT_w, \qquad \text{em } \Gamma_r \tag{2.14c}$$

Onde Γ_d, Γ_n e Γ_r são, respectivamente, os contornos dos tipos de Dirichlet, de Neumann e de Robin.

2.2 Perda de carga em escoamentos

A *perda de carga total*, designada por h_{l_T}, representa, de acordo com FOX et. al. (2010) [4], uma perda de energia total por unidade de massa de um fluido que escoa através de um tubo, duto ou canal.

A perda de carga total pode ser calculada para o escoamento permanente de um fluido incompressível através de um tubo circular ao se escrever a equação de energia para o sistema entre duas seções, 1 e 2:

$$h_{l_T} = \left(\frac{p_1}{\rho} + \alpha_1 \frac{\bar{V}_1^2}{2} + gz_1\right) - \left(\frac{p_2}{\rho} + \alpha_2 \frac{\bar{V}_2^2}{2} + gz_2\right) \tag{2.15}$$

Onde, para as seções 1 e 2, p é a pressão, z é a elevação, $\bar{V}$ é a velocidade média do fluido e α é o coeficiente de energia cinética dado de acordo com a equação 2.16 a seguir na qual V é a velocidade local em um ponto da seção transversal e $\dot{m}$ é a vazão mássica do fluido:

$$\alpha = \frac{\int_A \rho V^3 dA}{\dot{m}\bar{V}^2} \tag{2.16}$$

Segundo FOX et. al. (2010), três fatores tendem a reduzir a pressão em um escoamento tubular, sendo eles a redução na área da seção do tubo (aumenta a velocidade do escoamento), uma ascenção na sua inclinação positiva (aumento da cota vertical) e o atrito.

A equação 2.15 indica que, no caso de um tubo circular a pressão do escoamento irá cair mesmo que não haja redução no diâmetro ou aumento da cota z na tubulação. Neste caso, energia mecânica é convertida de forma irreversível em energia térmica por transferência de calor.

A perda de carga total é um somatório das perdas de carga "maiores" h_l - causadas por efeitos de atrito - com as perdas de carga "menores" h_{l_m} frutos de acidentes como entradas, saídas, variações de diâmetro e acessórios.

Perdas maiores

O desenvolvimento de um escoamento em duto de seção circular pode ser visto na figura 2.2. Considerando um escoamento laminar completamente desenvolvido num tubo horizontal circular de área constante, a perda de carga maior é expressa como a perda de pressão, e a equação 2.15 se reduz à equação 2.17:

$$h_{l_T} = h_l = \frac{\Delta p}{\rho} \tag{2.17}$$

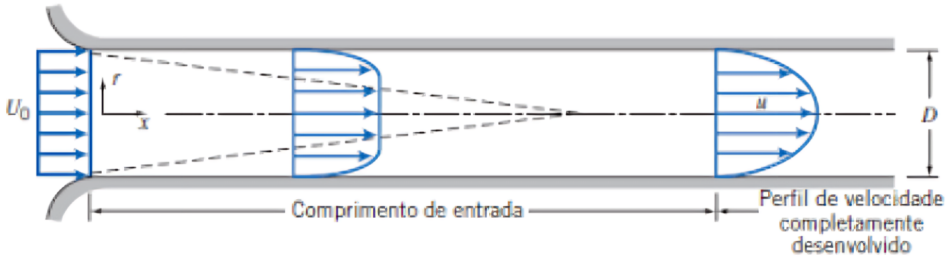


Figura 2.2: Desenvolvimento de um escoamento em duto de seção circular. De [4].

A perda de pressão pode ser calculada ao se desenvolver expressões para o perfil de velocidades para um escoamento laminar incompressível. Pode-se, portanto,

escrever a perda de pressão em função da vazão volumétrica para escoamentos em tubos, dutos ou canais e substituí-la em 2.17.

A perda de pressão pode ser alternativamente calculada através do fator de atrito f definido em ÖZİŞİK (1985) [21] usada para determinar-se o gradiente de pressão no duto ao longo do escoamento sem a necessidade de calcular-se o gradiente de velocidade na parede:

$$\frac{dP}{dx} = -f\frac{\rho\bar{V}^2}{2D} \tag{2.18}$$

Integrando a equação 2.18 para uma distância L temos a expressão final para a perda de pressão do fluido:

$$\Delta P = f\frac{L}{D}\frac{\rho\bar{V}^2}{2} \tag{2.19}$$

Perdas menores

Perdas de carga adicionais podem ser experimentadas quando o fluido passa por acidentes como curvas, válvulas, expansões ou reduções de área [4].

As perdas menores são tradicionalmente calculadas de duas formas: pelos *coeficientes de perda* K ou pelo *comprimento equivalente* L_e, ambos encontrados em tabelas de dados experimentais.

$$h_{l_m} = K\frac{\bar{V}^2}{2} \tag{2.20}$$

$$h_{l_m} = f\frac{L_e}{D}\frac{\bar{V}^2}{2} \tag{2.21}$$

2.3 Ventiladores e sopradores

De acordo com FOX et. al. (2010) [4], ventiladores são turbomáquinas que realizam trabalho sobre gases ou vapores aumentando sua pressão. Se diferenciam dos sopradores por serem capazes de gerar um menor aumento de pressão ao fluido para a mesma vazão.

Como qualquer turbomáquina, ventiladores e sopradores seguem a conservação do momento angular, na qual seu torque é produzido devido a uma variação no momento angular do fluido.

Ventiladores e sopradores podem ser caracterizados por gráficos de altura de carga, torque, potência de entrada e eficiência, sempre em função da vazão volumétrica, normalmente dada em CFM - *cubic feet per minute* ou pés cúbicos por minuto - formando as chamadas *curvas do ventilador*.

Assim como uma bomba, ventiladores e sopradores produzem uma altura de carga menor com o aumento da vazão, ao passo que a carga requerida para o sistema manter o escoamento cresce com a vazão volumétrica.

Logo, um ventilador sempre irá operar no chamado *ponto de trabalho*, ou seja, a interseção da *curva do ventilador* - altura de carga adicionada ao fluido em função da vazão - com a *curva do sistema* (ou curva de impedância do sistema).

A curva de impedância do sistema representa seu requisito de pressão. Os requisitos de pressão do sistema para uma dada vazão são compostos pela queda de pressão total do sistema, ou seja, das perdas maiores causadas por atrito e das perdas menores causadas por entradas, acidentes e saídas.

A figura 2.3 exemplifica a obtenção do ponto no qual o sistema ventilador-dissipador de calor irá operar.

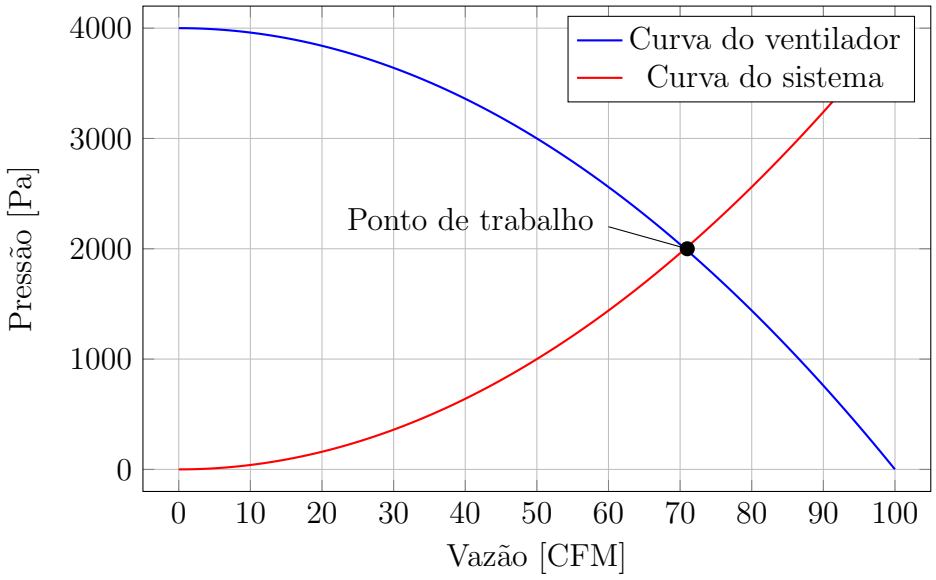


Figura 2.3: Exemplo de obtenção de ponto de trabalho de sistema-ventilador.

2.4 O Método dos Elementos Finitos

Em muitas situações da engenharia nos deparamos com sistemas *contínuos* nos quais o problema é definido de forma "infinitesimal" resultando muitas vezes em equações diferenciais parciais (ou EDPs) complexas com uma quantidade infinita de variáveis. Por sua vez, há situações nas quais é possível modelar problemas como sistemas *discretos*, nos quais a geometria de interesse é dividida em um número discreto de componentes - ou elementos - nos quais a resposta real do sistema é descrita diretamente pela solução de um número finito de variáveis [5] [6].

Resolver um sistema contínuo se torna quase impraticável uma vez que suas EDPs têm soluções exatas quase impossíveis por métodos analíticos para geometrias complexas e não triviais. Métodos numéricos, nos quais as equações sejam discretizadas, se fazem portanto necessários. Tal sistema *discretizado* é uma versão refinada dos sistemas discretos citados anteriormente, nos quais a precisão da solução pode ser controlada alterando o número de variáveis (de elementos).

A importância do Método dos Elementos Finitos - MEF - se destaca nesse contexto. As técnicas de elementos finitos possibilitam a solução de sistemas contínuos de maneira sistemática [6].

O MEF consiste na aproximação numérica de equações diferenciais parciais discretizando-se o domínio de interesse em elementos individuais e as equações diferenciais parciais complexas em equações simultâneas lineares ou não lineares, reduzindo então o problema contínuo, que tem um número infinito de incógnitas, para um com um número finito de incógnitas em pontos específicos chamados *nós*, a fim de obter-se uma solução global aproximada para o sistema [6].

2.4.1 História do Método

Segundo LIU et. al. (2022) [23], a história do MEF remonta ao início da década de 1940. Em 1941, Hrennikof publicou um artigo sobre seu modelo de membrana e placa como uma estrutura em treliça, tendo discretizado o domínio da solução em uma malha em treliça, sendo a forma mais antiga de discretização por malha.

No mesmo ano, Courant proferiu uma palestra sobre seu tratamento numérico utilizando um método variacional para resolver uma EDP de segunda ordem no

problema de torção de Saint-Venant para um cilindro, utilizando o método de Rayleigh–Ritz (variacional) com uma função de teste definida em malha triangular, sendo uma forma primitiva do MEF (posteriormente publicado como artigo em 1943 [24]).

De acordo com FISH et. al. [25] (2009), Turner, Clough, Martin e Topp estabeleceram os procedimentos de montagem da matrizes elementares e formulações elementares em 1956, chamado na época de "Método de Rigidez Matricial" [23]. Segundo ZIENKIEWICZ (2000) [5], foi Clough, em 1960, quem primeiro cunhou o termo "elementos finitos", no uso direto de uma "*metodologia padrão aplicada a sistemas discretos*".

A figura 2.4, retirada de [5], mostra a evolução dos métodos numéricos que resultaram no Método dos Elementos Finitos tal como conhecido nos dias de hoje.

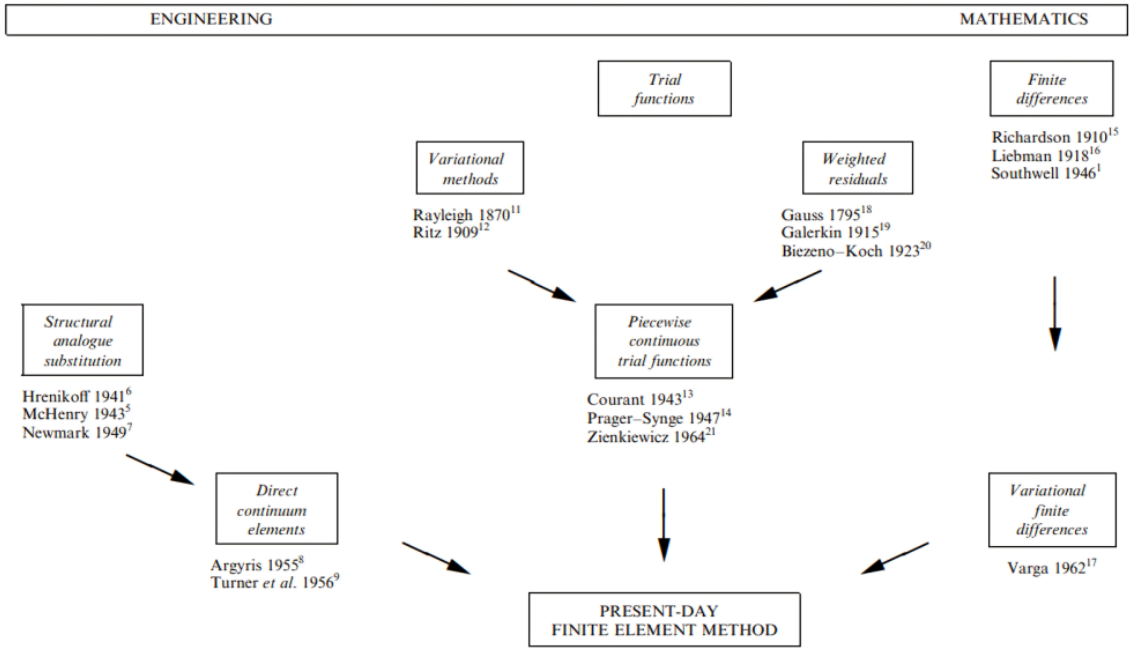


Figura 2.4: Evolução do Método dos Elementos Finitos. De [5].

2.4.2 Etapas do Método dos Elementos Finitos

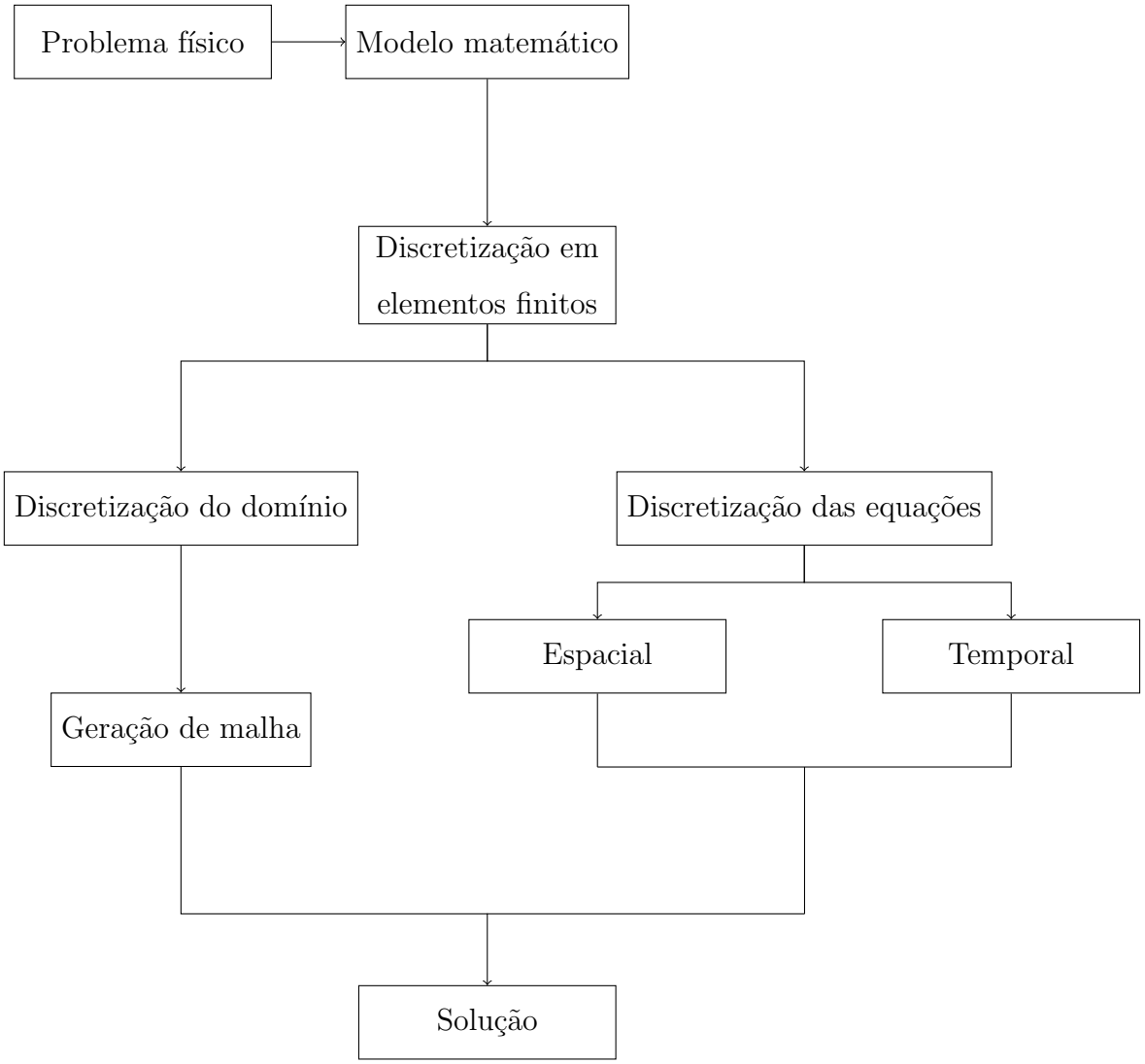


Figura 2.5: Obtenção de modelo numérico para MEF. Adaptada de [6].

A solução de um problema por meio do MEF segue o seguinte procedimento, também apresentado na figura 2.5, segundo LEWIS et al. (2016) [6]:

1. *Discretização do continuum*

   Dividir a região de solução em elementos de diversos tipos como triângulos, quadriláteros, tetraedros. Cada elemento é formado pela conexão de um certo número de nós (ver figura 2.6).

2. *Seleção das funções de forma*

   Escolher as funções de interpolação que aproximam a solução desconhecida dentro do elemento em termos dos valores da solução nos seus nós.

3. *Formulação elementar*

   Gerar das matrizes do laplaciano $[\mathbf{K}]^e$ e de massa $[\mathbf{M}]^e$ de cada elemento.

4. *"Assembly" das matrizes globais*

   Promover a montagem das matrizes elementares em matrizes globais que representem o comportamento de todo o domínio.

5. *Imposição das condições de contorno*

   Promover as modificações necessárias na matriz global do laplaciano $[\mathbf{K}]$ e no vetor global de carregamento $\{\mathbf{f}\}$ de acordo com o tipo de condição de contorno.

6. *Resolução do sistema linear*

   Resolver o sistema linear de equações algébricas 2.22 para obter-se os valores nodais da variável de interesse (por exemplo, temperatura):

$$[\mathbf{K}]\{\mathbf{T}\} = \{\mathbf{f}\} \tag{2.22}$$

7. *Cálculo das variáveis secundárias*

   Calcular os fluxos de calor a partir das temperaturas nodais, por exemplo.

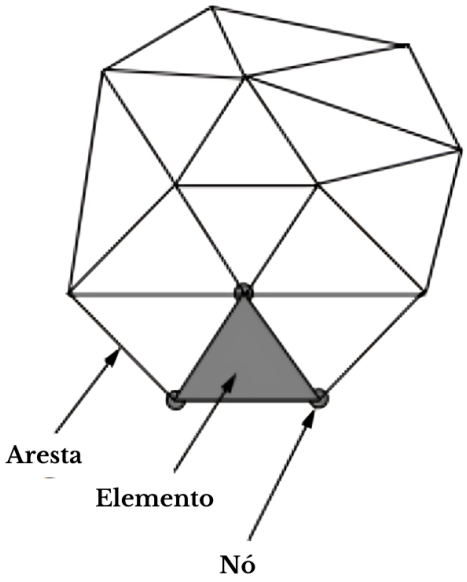


Figura 2.6: Malha de elementos finitos. Adaptada de [6].

2.4.3 Funções de forma

O método dos elementos finitos envolve a discretização tanto do domínio quanto das equações governantes. Neste processo, divide-se o domínio de solução em elementos finitos, e a solução é aproximada sobre esses elementos por uma função conhecida adequada. As funções utilizadas para representar a natureza da solução dentro de cada elemento são chamadas de funções de forma. Elas são usadas para determinar o valor da variável de campo dentro de um elemento, interpolando os valores nodais [6].

Majoritariamente são escolhidas funções de forma polinomiais, uma vez que elas podem ser integradas ou diferenciadas facilmente, e a precisão dos resultados pode ser melhorada aumentando a ordem do polinômio.

As funções de forma $N_i(e)$ devem seguir regras básicas para serem utilizadas, tais quais:

- Serem contínuas dentro do domínio do elemento;
- Valerem 1 no nó i e 0 nos demais nós.

Se as funções de forma elementares de cada nó i do mesmo elemento e forem iguais, temos o chamado Método de Galerkin - método de resíduos ponderados utilizado no decorrer do presente trabalho.

2.4.4 Discretização da equação do calor pelo método de Galerkin

A equação do calor para sólidos em regime permanente é obtida ao fim da subseção 2.1.4 e tem a forma da equação de Poisson:

$$k\boldsymbol{\nabla}^2 T + Q_{ger} = 0 \tag{2.13}$$

Nesta subseção procederemos com a discretização da equação de Poisson 2.13 pelo método de Galerkin. O método de Galerkin, de acordo com LEWIS et. al. [6], é um método de resíduos ponderados e requer, portanto, que média ponderada do resíduo seja nula sobre o domínio, onde o resíduo é definido, no caso da equação do

calor, como o lado direito da equação 2.13 quando utilizado um campo $\bar{T}$ aproximando a temperatura, da seguinte forma:

$$k\boldsymbol{\nabla}^2\bar{T} + Q_{ger} = R \tag{2.23}$$

O campo de temperaturas é aproximado da seguinte forma, com n sendo o número de nós da malha, $\mathbf{N}$ sendo o vetor linha das funções de forma e $\mathbf{T}$, o vetor coluna das temperaturas nodais:

$$T \approx \bar{T} = \sum_{i=1}^{n} N_i\, T_i = \mathbf{NT} \tag{2.24}$$

As equações 2.13 e 2.23 são as formas "fortes" da equação. Para continuar com o método de Galerkin, é necessário obter a forma "fraca" da equação, multiplicando o resíduo R por uma função peso w, integrando-a no domínio Ω e igualando a 0.

$$\int_{\Omega} \mathbf{w}\left(k\boldsymbol{\nabla}^2\bar{T} + Q_{ger}\right)\, d\Omega = 0 \tag{2.25}$$

Onde $\mathbf{w}$ é o vetor coluna das funções peso.

Para a função peso, a definimos como sendo igual à função de forma. O método de Galerkin diferencia-se de outros métodos de resíduos ponderados no que determina que a função de forma escolhida para a função peso seja a mesma escolhida para aproximar o campo de temperaturas. Desta forma temos:

$$\mathbf{w} = \mathbf{N^T} \tag{2.26}$$

Substituindo 2.26 em 2.25, temos:

$$\int_{\Omega} \mathbf{N^T}\left(k\boldsymbol{\nabla}^2\bar{T} + Q_{ger}\right)\, d\Omega = 0 \tag{2.27}$$

Em seguida, é usada a identidade de Green para separar os termos do contorno e do miolo, obtendo a forma fraca final da seguinte forma:

$$\int_{\Gamma} \mathbf{N^T} k\, \nabla\bar{T}\cdot\mathbf{n}\, d\Gamma - \int_{\Omega} \nabla\mathbf{N^T} k\nabla\bar{T}\, d\Omega + \int_{\Omega} \mathbf{N^T}\, Q_{ger}\, d\Omega = 0 \tag{2.28}$$

Onde $\mathbf{n}$ é o vetor normal à superfície.

Quando a condição de contorno é do tipo de Dirichlet, a função peso w é igual a zero e a integral no contorno é nula. Portanto, a integral em Γ sofre contribuição apenas das condições de contorno de Neumann (equação 2.14b) e de Robin (equação 2.14c). Incorporando as condições de contorno supracitadas e procedendo com a formulação para o elemento finito, obtemos:

$$
\begin{aligned}
&- \int_{\Omega^{(e)}} \nabla \mathbf{N^{(e)^T}} k \nabla \bar{T} \, d\Omega + \int_{\Omega^{(e)}} \mathbf{N^{(e)^T}} Q_{ger} \, d\Omega \\
&+ \int_{\Gamma_n^{(e)}} \mathbf{N^{(e)^T}} q \, d\Gamma - \int_{\Gamma_r^{(e)}} \mathbf{N^{(e)^T}} h \bar{T} \, d\Gamma + \int_{\Gamma_r^{(e)}} \mathbf{N^{(e)^T}} h T_\infty \, d\Gamma = 0
\end{aligned} \tag{2.29}
$$

Onde $\Gamma_n^{(e)}$ e $\Gamma_r^{(e)}$ representam, respectivamente, os elementos de contorno sujeitos a fluxo prescrito (condição de contorno de Neumann) e convecção (condição de contorno de Robin). Por fim, substituindo a aproximação para o campo de temperaturas da equação 2.24, obtemos a forma final da discretização da equação para o elemento finito:

$$
\begin{aligned}
&- \int_{\Omega^{(e)}} \nabla \mathbf{N^{(e)^T}} k \nabla \mathbf{N}^{(e)} \mathbf{T}^{(e)} \, d\Omega + \int_{\Omega^{(e)}} \mathbf{N^{(e)^T}} Q_{ger} \, d\Omega \\
&+ \int_{\Gamma_n^{(e)}} \mathbf{N^{(e)^T}} q \, d\Gamma - \int_{\Gamma_r^{(e)}} \mathbf{N^{(e)^T}} h \mathbf{N}^{(e)} \mathbf{T}^{(e)} \, d\Gamma + \int_{\Gamma_r^{(e)}} \mathbf{N^{(e)^T}} h T_\infty \, d\Gamma = 0
\end{aligned} \tag{2.30}
$$

Isolando os termos que multiplicam a temperatura, podemos montar o seguinte sistema linear do elemento.

$$
[\mathbf{K}]^e \{\mathbf{T}\}^e = \{\mathbf{f}\}^e \tag{2.31}
$$

Temos então o vetor $\{\mathbf{f}\}^\mathbf{e}$ de carregamento e a matriz $[\mathbf{K}]^\mathbf{e}$ do Laplaciano escrita usando a definição da matriz do gradiente elementar $[\mathbf{B}]^\mathbf{e}$ da equação 2.32, para o elemento.

$$
[\mathbf{B}]^\mathbf{e} = \nabla \mathbf{N^e} \tag{2.32}
$$

Logo:

$$[\mathbf{K}]^{\mathbf{e}} = \int_{\Omega^{(e)}} k[\mathbf{B}]^{\mathbf{e}T}[\mathbf{B}]^{\mathbf{e}} d\Omega + \int_{\Gamma_r^{(e)}} \mathbf{N^{(e)^T}} h \mathbf{N}^{(e)}\, d\Gamma \tag{2.33}$$

$$\{\mathbf{f}\}^{\mathbf{e}} = \int_{\Omega^{(e)}} \mathbf{N^{(e)^T}} Q_{ger}\, d\Omega + \int_{\Gamma_n^{(e)}} \mathbf{N^{(e)^T}} q d\Gamma + \int_{\Gamma_r^{(e)}} \mathbf{N^{(e)^T}} h T_\infty d\Gamma \tag{2.34}$$

A partir da geração das matrizes e vetores elementares, é possível montar as matrizes globais do sistema e proceder com a solução do sistema linear da equação 2.22.

2.5 CPU

De acordo com MANURUNG et. al. (2021) [26], a CPU, ou Unidade Central de Processamento, também chamada de processador, é a principal unidade de processamento de informações em um computador. A CPU regula todas as atividades e a execução de todos os programas, incluindo aplicativos e *software*. Qualquer processo no computador, mesmo os mais simples, sempre será controlado pela CPU. Ela lê e executa comandos e processa dados de *software*.

De acordo com a nota de aplicação da AMD [7], CPUs podem ser encapsuladas de diversas formas, com ou sem uma tampa (chamada de dissipador de calor integrado, ou IHS - *Integrated Heat Spreader*), ou *lid*. De modo geral, dispositivos sem tampa oferecem melhor desempenho térmico, pois não adicionam resistência térmica extra na forma de um TIM1 dentro do encapsulamento e na própria tampa.

Por outro lado, dispositivos com tampa podem ser mais fáceis de projetar, pois representam a abordagem mais tradicional no encapsulamento de circuitos integrados (*IC packaging*), e soluções térmicas disponíveis no mercado são mais fáceis de encontrar.

Um exemplo de dispositivo com tampa é mostrado na figura 2.7. As tampas (IHS) desses encapsulamentos são geralmente feitas de cobre revestido com níquel e são unidas ao *die* (bloco de material semicondutor) por meio do TIM1. O dissipador de calor aletado pode então ser unido por cima do IHS por meio de um TIM2, como na figura 1.1.

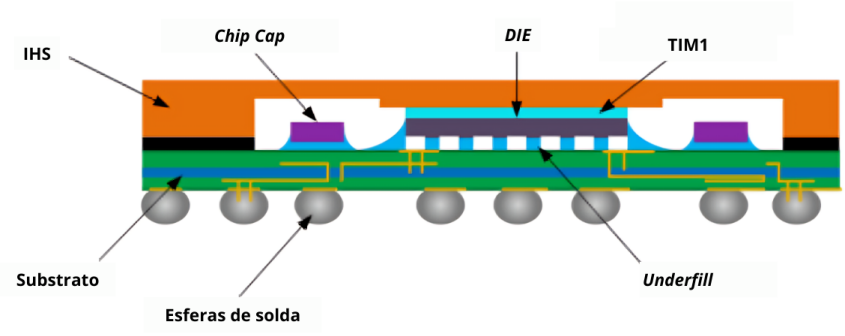


Figura 2.7: Seção transversal de uma tampa forjada. Adaptada de [7].

2.6 Correlações para o dissipador de calor

Na presente seção serão apresentadas as correlações utilizadas no presente trabalho para se obter o coeficiente de transferência de calor médio por convecção e a perda de pressão ao longo das aletas do dissipador.

2.6.1 Determinação do coeficiente convectivo médio nas aletas do dissipador

TEERSTRA et. al. (1999) [8] propuseram um modelo analítico para previsão do coeficiente de transferência de calor médio por convecção forçada para um dissipador de calor como o da figura 2.8 de aletas retas montadas sobre uma base condutiva isotérmica resfriado a ar usado para aplicações eletrônicas, objeto do presente estudo, no qual o número de Nusselt médio pode ser calculado em função da geometria do dissipador e da velocidade do fluido.

Os autores assumiram uma velocidade uniforme U_{ar} através dos canais formados pelas aletas sem *bypass* de ar por fora das arestas dos canais, ou seja, com fluxo totalmente canalizado. Além disso, a face superior da base, onde são montadas as aletas é considerada adiabática, bem como as duas faces que se encontram nas extremidades dos canais formados pelas aletas.

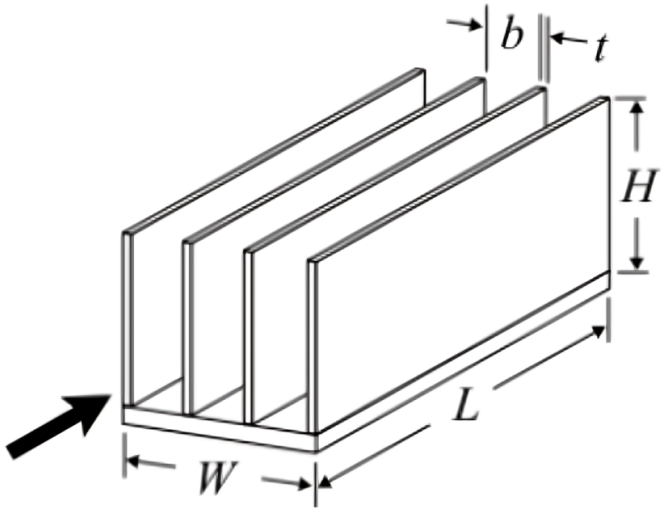


Figura 2.8: Dimensões do dissipador de calor. De [8].

Número de Nusselt ideal

Primeiramente foi estudado o caso no qual cada uma das paredes do canal estão em uma temperatura uniforme T_w igual à temperatura da base T_s, também uniforme, tendo como resultado o número de Nusselt médio ideal Nu_i. Depois, chega-se ao número de Nusselt médio real Nu_b, corrigido pela eficiência das aletas, com base nos parâmetros geométricos do dissipador.

Para cálculo de Nu_i, TEERSTRA et. al. consideraram como soluções assintóticas (Nu_{fd} e Nu_{dev}) os casos limitantes escoamento completamente desenvolvido e escoamento termicamente e hidrodinamicamente em desenvolvimento simultâneo para o problema da convecção forçada entre placas paralelas isotérmicas. O número de Nusselt pode ser obtido através da combinação das duas soluções assintóticas usando a forma inversa da técnica de solução composta de CHURCHILL e USAGI (1972) [27]:

$$Nu = \left[\left(\frac{1}{Nu_{fd}} \right)^n + \left(\frac{1}{Nu_{dev}} \right)^n \right]^{-1/n} \tag{2.35}$$

Onde n é um parâmetro de combinação empírico usado para controlar o comportamento do modelo na região de transição entre as assíntotas. A solução composta é apresentada de forma gráfica na figura 2.9.

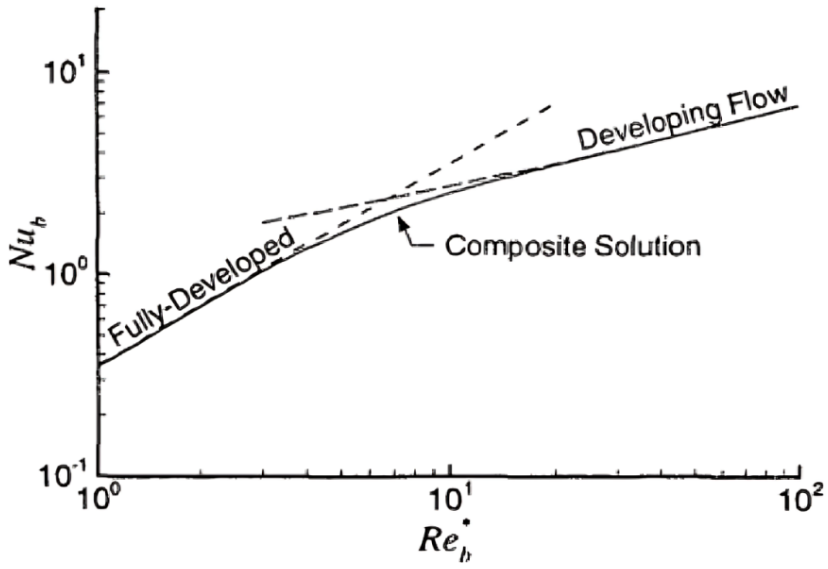


Figura 2.9: Solução composta proposta. De [8].

TEERSTRA et. al. chegaram ao valor de $n = 3$ para o coeficiente de combinação a partir de simulações numéricas, com RMS entre o modelo e os dados numéricos de 2,1%.

Desta forma, com as soluções assintóticas calculadas pelos autores, pode ser desenvolvida uma expressão final para Nu_i:

$$Nu_i = \left[\left(\frac{1}{\frac{Re_b^* Pr}{2}} \right)^3 + \left(\frac{1}{0,664\sqrt{Re_b^*} Pr^{1/3} \sqrt{1 + \frac{3,65}{\sqrt{Re_b^*}}}} \right)^3 \right]^{-1/3} \tag{2.36}$$

Onde Re_b^* é o número de Reynolds de canal, dado pela seguinte equação com base na notação da figura 2.8:

$$Re_b^* = \frac{\rho U_{arr} b}{\mu} \cdot \left(\frac{b}{L} \right) \tag{2.37}$$

Número de Nusselt real

O número de Nusselt médio real proposto para o modelo é obtido a partir de Nu_i considerando-se a eficiência η da aleta com ponta adiabática, da seguinte forma:

$$\eta = \frac{\tanh(mH)}{mH} \tag{2.38}$$

Onde m é o parâmetro da aleta dado pela seguinte equação:

$$m = \sqrt{\frac{hP}{kA_c}} \tag{2.39}$$

Onde $P = 2(t + L)$ é o perímetro das aletas, k é a condutividade térmica do material das aletas, $A_c = tL$ é a área da seção transversal das aletas e $h = Nu_i \cdot \frac{k_f}{b}$ é o coeficiente médio de transferência de calor por convecção.

Substituindo os parâmetros geométricos das aletas do dissipador tal como na figura 2.8, temos:

$$\eta = \frac{\tanh\sqrt{2Nu_i\frac{k_{ar}}{k}\frac{H_f}{b}\frac{H_f}{t}\left(\frac{t}{L}+1\right)}}{\sqrt{2Nu_i\frac{k_{ar}}{k}\frac{H_f}{b}\frac{H_f}{t}\left(\frac{t}{L}+1\right)}} \tag{2.40}$$

Onde k é a condutividade térmica das aletas e k_{ar} é a condutividade térmica do ar.

O número de Nusselt médio real é, então:

$$Nu_b = \eta Nu_i \tag{2.41}$$

Coeficiente de transferência de calor

Uma vez obtido o número de Nusselt médio real para o dissipador, o coeficiente de transferência de calor médio nas aletas é dado pela seguinte equação.

$$h = Nu_b \cdot \frac{k_f}{b} \tag{2.42}$$

De acordo com os autores, o modelo resultante é aplicável para a faixa $0.1 < Re_b^* < 100$.

2.6.2 Determinação da perda de pressão no dissipador

COPELAND (2000) [18] mencionado na seção 1.2, demonstra o desenvolvimento de uma expressão para perda de pressão em dissipadores de calor de aletas retas através da análise do perfil laminar de velocidades de um fluido que escoa com velocidade média U_{ar} através de um duto retangular de comprimento L, largura b e altura H_f, usando o modelo de COPELAND (1995) [28] modificado para a condição de contorno isotérmica.

Escoamento completamente desenvolvido

O fator de atrito para o escoamento laminar completamente desenvolvido é dado pela equação 2.43.

$$fRe = 19{,}64 \cdot \frac{\left[\left(\frac{b}{H_f}\right)^2 + 1\right]}{\left[\left(\frac{b}{H_f} + 1\right)^2\right]} + 4{,}7 \tag{2.43}$$

Onde f é o fator de atrito completamente desenvolvido, b é a largura do canal e Re é o número de Reynolds dado por:

$$Re = \frac{\rho U_{ar} D_h}{\mu} \tag{2.44}$$

E sendo D_h o diâmetro hidráulico dado por:

$$D_h = \frac{2bH_f}{b + H_f} \tag{2.45}$$

Escoamento em desenvolvimento

O fator de atrito para escoamento laminar hidrodinamicamente em desenvolvimento foi obtido de SHAH e LONDON (1978) [29] e ajustados a uma equação na forma de Churchill-Usagi.

$$f_{app}Re = \{[3{,}2(x^+)^{-0{,}57}]^2 + (fRe)^2\}^{1/2} \tag{2.46}$$

Onde f_{app} é o fator de atrito aparente e x^+ é a distância adimensional ao longo do canal na região de entrada termodinâmica dada por:

$$x^+ = \frac{L}{ReD_h} \tag{2.47}$$

Perda de pressão

Segundo KAYS e LONDON (1984) [30], a perda de pressão num canal de comprimento L sofre a contribuição da entrada, do atrito aparente e da saída do canal segundo a seguinte equação:

$$\Delta P = (K_c + 4 f_{app} Rex^{+} + K_e) H_{ar} \tag{2.48}$$

Onde H_{ar} é a altura de carga hidrodinâmica do arranjo de aletas, definida como:

$$H_{ar} = \frac{\rho U_{ar}^2}{2} \tag{2.49}$$

Na equação 2.48, K_c e K_e são, respectivamente, os coeficientes para escoamento laminar em canais na entrada (contração) e na saída (expansão) do escoamento.

$$K_c = 0{,}8 - 0{,}4 \left(\frac{b}{p}\right)^2 \tag{2.50}$$

$$K_e = 1{,}0 \left(1 - \left(\frac{b}{p}\right)^2\right) - 0{,}4 \left(\frac{b}{p}\right) \tag{2.51}$$

Onde p é o passo, ou *pitch*, das aletas no dissipador de calor.

A vazão volumétrica $\dot{V}$ do escoamento de ar pelos canais do dissipador pode ser relacionada com a velocidade do ar entre as aletas U_{ar} e parâmetros geométricos a partir da seguinte equação:

$$\dot{V} = \frac{W b H_f U_{ar}}{p} \tag{2.52}$$

Capítulo 3

Metodologia

3.1 Descrição do problema

Conforme mencionado na seção 1.3, o objetivo deste trabalho é obter o número ótimo de aletas em um dissipador de calor de cobre montado sobre o processador Intel Xeon 6710E [19], o qual é resfriado de forma forçada pelo ventilador San Ace 9CRH0648P6G001 [9].

Para chegar-se a este resultado, deve-se obter um gráfico de desempenho do dissipador de calor *versus* seu número de aletas. Para este trabalho, o desempenho do dissipador é quantificado a partir da temperatura média de sua base, uma vez que quanto menor esta for, menor terá sido a resistência à transferência de calor do sistema.

Foram realizadas simulações para diversos dissipadores de calor, cada qual com um número de aletas retas. Para tal, cada geometria CAD e malha foram geradas pelo *Gmsh*, utilizando um script elaborado pelo autor, que juntamente da geometria e malha, também é responsável por gerar um arquivo JSON para posteriormente servir de arquivo de input para o *solver* de elementos finitos.

Uma vez em posse das malhas e do arquivo JSON, cada simulação foi realizada usando um programa desenvolvido pelo autor em *Python* que usa o Método dos Elementos Finitos. Tal programa se utiliza da biblioteca *Meshio* para efetuar a leitura da malha e das bibliotecas *Numpy* e *Scipy* para gerar, manipular e realizar cálculos com vetores e matrizes inerentes ao método.

Deve ser ressaltado que a análise deve estar dentro dos limites de aplicação das

correlações usadas e mostradas nas subseções 2.6.1 e 2.6.2. Os números mínimo e máximo de aletas que podem ser contempladas no estudo serão descritos na subseção 3.6.

Para cada uma das simulações, o coeficiente de transferência de calor usado será diferente. Isso porque, como demonstrado ao longo do capítulo 2, o ventilador irá operar no ponto de trabalho do sistema, conferindo para cada geometria uma vazão diferente, o que se traduz em diferentes velocidades U_{arr} - que juntamente a diferentes parâmetros geométricos - dão resultado a diferentes h.

A curva do ventilador San Ace 9CRH0648P6G001 foi retirada de sua folha de dados [9]. Foi utilizada a ferramenta online *Webplot Digitizer* para obter os dados do gráfico encontrado na folha de dados em extensão XLSX. Os pontos correspondentes do gráfico são apresentados de forma tabular no apêndice B.

Os dados foram posteriormente transformados em *arrays* de pressão e vazão que, plotados em *Python* usando a biblioteca *Matplotlib*, deram origem ao gráfico da figura 3.1:

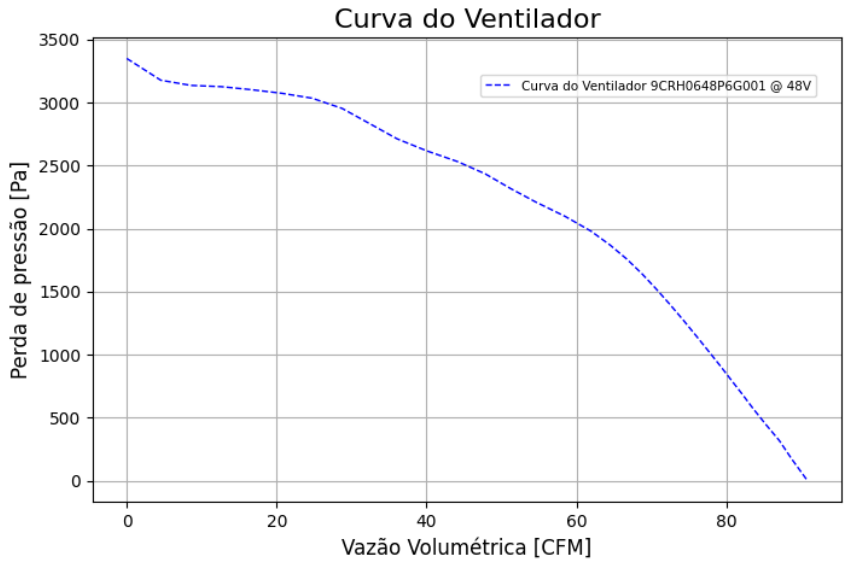


Figura 3.1: Curva do ventilador selecionado. De [9].

3.2 Modelo físico

O modelo a ser simulado neste trabalho corresponde à imagem da figura 3.2 do dissipador com 35 aletas, no qual as setas indicam a direção e sentido do escoamento

de ar considerado, concordante com as correlações da seção 2.6 e com a figura 2.8. O modelo consiste nas aletas unidas à uma base, omitindo portanto a CPU, a TIM e a PCB (placa de circuito impresso).

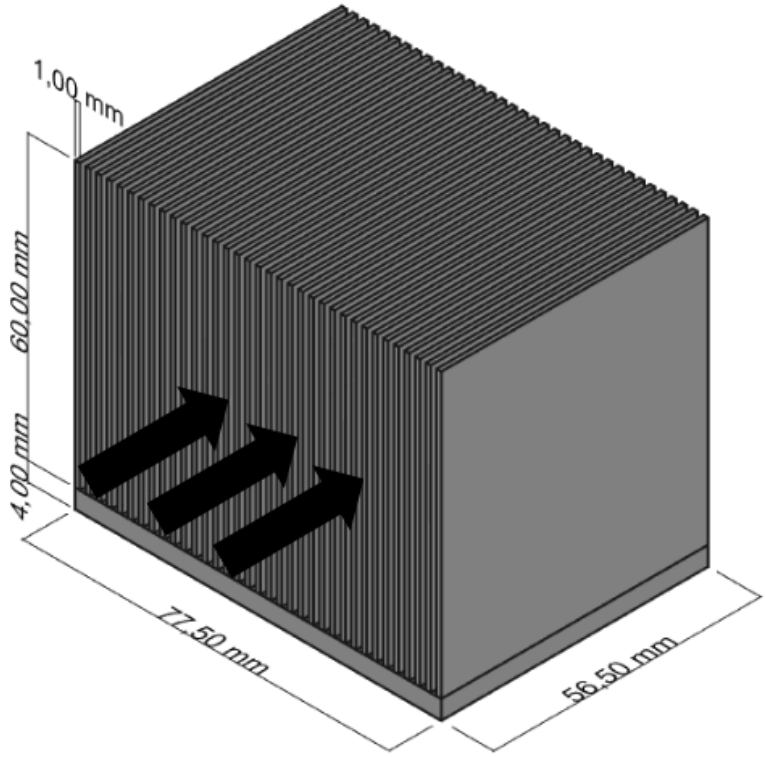


Figura 3.2: Modelo de dissipador de calor com suas dimensões principais. Exemplo com 35 aletas.

Algumas hipóteses e considerações foram levantadas a fim de simplificar o modelo e, consequentemente, seu tratamento matemático e numérico. São elas:

1. O *die* não foi modelado, de tal forma que a temperatura a ser controlada é T_{CASE} e não T_{JUNC};

2. Não foi considerada uma resistência extra resultante da adição de um material de interface (TIM) entre o case e o dissipador. Desta forma, T_{CASE} é igual à temperatura do fundo do dissipador;

3. O calor gerado pela fonte (CPU) é resultado exclusivamente da sua TDP (*Thermal Design Power*, a taxa máxima de calor que a CPU pode dissipar em condições normais de operação), em [W];

4. O fluxo de calor gerado pela fonte é direcionado integralmente para o dissipador. A condução do *die* para a PCB é desprezada. Desta forma, o dissipador recebe um fluxo de calor igual à TDP dividida pela área do case, em [W/m²];

5. As simulações foram realizadas em regime permanente;

6. A base do dissipador tem as mesmas dimensões do case. Logo, não há resistência térmica de espalhamento;

7. As condições de convecção (coeficiente h e temperatura do ar próximo às aletas) calculadas pela correlação apresentada na seção 2.6.1 para as aletas, foram estendidas para as faces da base do dissipador, como mostrado na figura 3.3(a);

8. O escoamento de ar é paralelo à superfície do dissipador, movendo-se através dos canais formados pelas aletas, como na figura 3.2;

9. As pontas das aletas são isoladas, bem como as faces externas do dissipador;

10. O escoamento de ar é completamente canalizado, ocorrendo sem *bypass* (perda de fluxo pelas laterais do dissipador de calor) e sem *tip clearance* (perda de fluxo por cima do dissipador de calor);

11. A temperatura do ar nas proximidades das aletas foi fixada em 40 °C ao longo de todo o domínio;

12. A condutividade térmica do cobre foi considerada invariável com a temperatura e fixada em 393 W/m·°C, correspondente à condutividade do cobre na T_{CASE} máxima permitida pelo fabricante de 85°C [11];

13. O material do dissipador de calor é isotrópico.

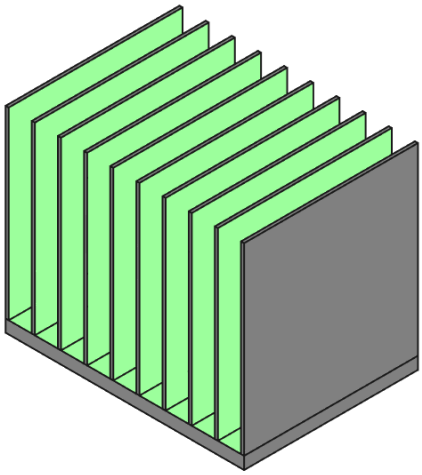
3.2.1 Condições de contorno

As condições de contorno adotas são mostradas nas figuras 3.3. A figura 3.3(a) mostra em verde as faces contempladas com a condição de contorno de convecção cujo coeficiente de transferência de calor h é calculado pelo procedimento em 2.6.1, com a temperatura das proximidades da 40 °C.

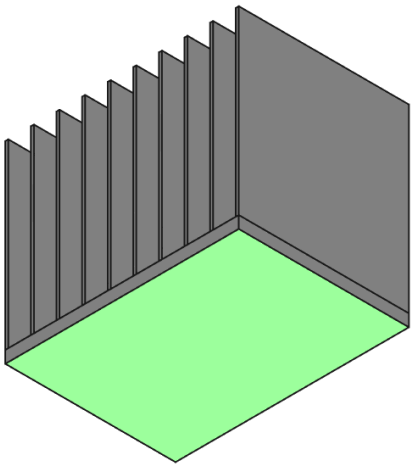
Na figura 3.3(b) a condição do fundo do dissipador prescreve todo o fluxo de calor gerado pela TDP da CPU. Desta forma, o fluxo de calor q será igual à TDP dividida pela área da base do dissipador:

$$q = \frac{TDP}{L \cdot W} = 46817 \text{ W/m}^2 \tag{3.1}$$

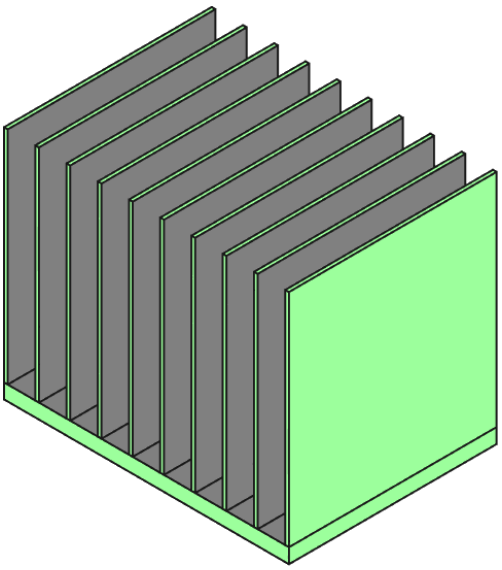
Em 3.3(c), o fluxo de calor é considerado nulo e as faces em verde são isoladas.



((a)) Condição de contorno de Robin.



((b)) Condição de contorno de Neumann não homogêneo.



((c)) Condição de contorno de Neumann homogêneo.

Figura 3.3: Condições de contorno para o dissipador. Exemplo com 10 aletas.

3.3 Modelo matemático

A equação final do calor para sólidos é dada em 2.12. Entretanto, é possível simplificar a equação tendo em vista as considerações feitas na seção anterior de regime permanente, condutividade térmica constante e ausência de geração de calor. Obtemos então a equação de Laplace:

$$\nabla^2 T = 0 \qquad em \quad \Omega \tag{3.2}$$

$$k\nabla T \cdot \mathbf{n} = -q \qquad em \quad \Gamma_n \tag{2.14b}$$

$$k\nabla T \cdot \mathbf{n} = hT_\infty - hT_w \qquad em \quad \Gamma_r \tag{2.14c}$$

Onde Ω representa o domínio do problema, Γ_r representa o contorno de Robin (convecção) e Γ_n é o contorno de Neumann, seja ele homogêneo ou não. Ademais, T_∞ é a temperatura do ar nas proximidades das aletas e $\mathbf{n}$ é um vetor unitário normal às superfícies.

3.4 Modelo numérico

É possível simplificar o vetor carregamento 2.34 uma vez que o modelo simulado não apresenta geração de calor. Desta forma, temos:

$$[\mathbf{K}]^{\mathbf{e}} = \int_{\Omega^{(e)}} k[\mathbf{B}]^{\mathbf{e}T}[\mathbf{B}]^{\mathbf{e}} d\Omega + \int_{\Gamma_r^{(e)}} \mathbf{N^{(e)}}^{\mathbf{T}} h \mathbf{N}^{(e)}\, d\Gamma \tag{2.33}$$

$$\{\mathbf{f}\}^{\mathbf{e}} = \int_{\Gamma_n^{(e)}} \mathbf{N^{(e)}}^{\mathbf{T}} q d\Gamma + \int_{\Gamma_r^{(e)}} \mathbf{N^{(e)}}^{\mathbf{T}} h T_\infty d\Gamma \tag{3.3}$$

Onde h é o coeficiente de transferência de calor obtido pela correlação desenvolvida em 2.6.1 e $T_\infty = 40$ °C como discutido na seção 3.2, válidos para os contornos mostrados na figura 3.3(a). O fundo do dissipador de calor, em verde na figura 3.3(b) recebe o fluxo de calor $q = 46817$ W/m^2.

O termo do "miolo" deve ser avaliado em um elemento de malha tridimensional. Para este trabalho, foi escolhido o tetraedro linear da figura 3.4, ou seja, cada elemento é constituído de 4 nós e as funções de forma são lineares.

Cabe ressaltar que a escolha por um elemento tridimensional se dá pelo fato de que as aletas das extremidades do dissipador apresentam condições de contorno de convecção em um de seus lados e são isoladas no outro, gerando um fluxo de calor transversal ao longo das espessuras das aletas. Caso a transferência de calor fosse unidimensional como nas demais aletas, poderiam ter sido escolhidos elementos de superfície, ou de "casca", o que reduziria drasticamente o número de elementos e o tempo de computação.

A temperatura em qualquer ponto do tetraedro linear pode ser interpolada usando as funções de forma de primeiro grau, tal como se segue:

$$T = N_1T_1 + N_2T_2 + N_3T_3 + N_4T_4 \tag{3.4}$$

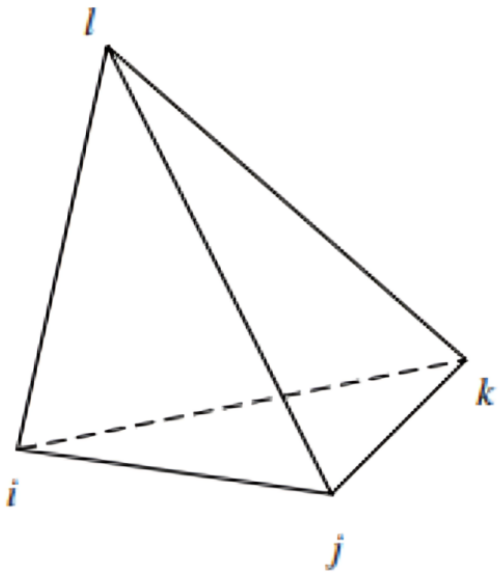


Figura 3.4: Tetraedro linear. De [6].

Onde:

$$N_i = \frac{1}{6V}(a_i + b_ix + c_iy + d_iz) \;\text{ com }\; i = 1, 2, 3, 4 \tag{3.5}$$

Por sua vez, o volume de tetraedro é calculado:

$$V = \frac{1}{6}\begin{vmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{vmatrix} \tag{3.6}$$

O termo do "miolo" dará origem à matriz do Laplaciano elementar, que posteriormente irá sofrer alteração pelo segundo termo da equação 2.33 proveniente da convecção.

A matriz no gradiente $[\mathbf{B}]^{\mathbf{e}}$ para o tetraedro linear é então definida:

$$[\mathbf{B}]^{\mathbf{e}} = \frac{1}{6V}\begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \tag{3.7}$$

Onde os termos b, c e d são dados:

$$\begin{aligned} b_1 &= (y_2 - y_4)(z_3 - z_4) - (y_3 - y_4)(z_2 - z_4) \\ b_2 &= (y_3 - y_4)(z_1 - z_4) - (y_1 - y_4)(z_3 - z_4) \\ b_3 &= (y_1 - y_4)(z_2 - z_4) - (y_2 - y_4)(z_1 - z_4) \\ b_4 &= b_1 + b_2 + b_3 \end{aligned} \tag{3.8}$$

$$\begin{aligned} c_1 &= (x_3 - x_4)(z_2 - z_4) - (x_2 - x_4)(z_3 - z_4) \\ c_2 &= (x_1 - x_4)(z_3 - z_4) - (x_3 - x_4)(z_1 - z_4) \\ c_3 &= (x_2 - x_4)(z_1 - z_4) - (x_1 - x_4)(z_2 - z_4) \\ c_4 &= -(c_1 + c_2 + c_3) \end{aligned} \tag{3.9}$$

$$\begin{aligned} d_1 &= (x_2 - x_4)(y_3 - y_4) - (x_3 - x_4)(y_3 - y_4) \\ d_2 &= (x_3 - x_4)(y_1 - y_4) - (x_1 - x_4)(y_3 - y_4) \\ d_3 &= (x_1 - x_4)(y_2 - y_4) - (x_2 - x_4)(y_1 - y_4) \\ d_4 &= -(d_1 + d_2 + d_3) \end{aligned} \tag{3.10}$$

Substituindo os termos das equações 3.8, 3.9 e 3.10 em 3.7 e substituindo na integral no "miolo" da equação 2.33, temos:

$$[\mathbf{K}]^{\mathbf{e}} = k([\mathbf{K_y}]^{\mathbf{e}} + [\mathbf{K_z}]^{\mathbf{e}} + [\mathbf{K_x}]^{\mathbf{e}}) \tag{3.11}$$

Onde:

$$[\mathbf{K_x}]^{\mathbf{e}} = \frac{1}{36V} \begin{bmatrix} b_1b_1 & b_1b_2 & b_1b_3 & b_1b_4 \\ b_2b_1 & b_2b_2 & b_2b_3 & b_2b_4 \\ b_3b_1 & b_3b_2 & b_3b_3 & b_3b_4 \\ b_4b_1 & b_4b_2 & b_4b_3 & b_4b_4 \end{bmatrix} \tag{3.12}$$

$$[\mathbf{K_y}]^{\mathbf{e}} = \frac{1}{36V} \begin{bmatrix} c_1c_1 & c_1c_2 & c_1c_3 & c_1c_4 \\ c_2c_1 & c_2c_2 & c_2c_3 & c_2c_4 \\ c_3c_1 & c_3c_2 & c_3c_3 & c_3c_4 \\ c_4c_1 & c_4c_2 & c_4c_3 & c_4c_4 \end{bmatrix} \tag{3.13}$$

$$[\mathbf{K_z}]^{\mathbf{e}} = \frac{1}{36V} \begin{bmatrix} d_1d_1 & d_1d_2 & d_1d_3 & d_1d_4 \\ d_2d_1 & d_2d_2 & d_2d_3 & d_2d_4 \\ d_3d_1 & d_3d_2 & d_3d_3 & d_3d_4 \\ d_4d_1 & d_4d_2 & d_4d_3 & d_4d_4 \end{bmatrix} \tag{3.14}$$

Para avaliação das integrais no contorno serão usados triângulos lineares (figura 3.5), uma vez que o elemento volumétrico é um tetraedro linear.

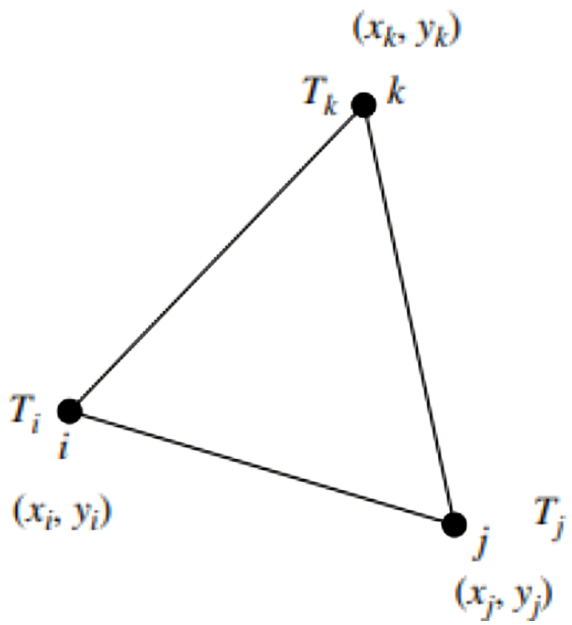


Figura 3.5: Triângulo linear. De [6].

A temperatura no triângulo é dada pela interpolação com as funções de forma lineares:

$$T = N_1 T_1 + N_2 T_2 + N_3 T_3 \tag{3.15}$$

As funções de forma são tais que:

$$\begin{aligned} N_1 &= \frac{1}{2A}(a_1 + b_1 x + c_1 y) \\ N_2 &= \frac{1}{2A}(a_2 + b_2 x + c_2 y) \\ N_3 &= \frac{1}{2A}(a_3 + b_3 x + c_3 y) \end{aligned} \tag{3.16}$$

Onde:

$$\begin{aligned} a_1 &= x_2 y_3 - x_3 y_2 \qquad b_1 = y_2 - y_3 \qquad c_1 = x_3 - x_2 \\ a_2 &= x_3 y_1 - x_1 y_3 \qquad b_2 = y_3 - y_1 \qquad c_2 = x_1 - x_3 \\ a_3 &= x_1 y_2 - x_2 y_1 \qquad b_3 = y_1 - y_2 \qquad c_3 = x_2 - x_1 \end{aligned} \tag{3.17}$$

A área do elemento é calculada como:

$$A = \frac{1}{2} \left| (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1) \right| \tag{3.18}$$

A contribuição da condição de contorno de convecção é dada pela matriz de massa $[\mathbf{M}]^e$, que aparece na integração na equação 2.33. Para um triângulo linear, a matriz de massa é:

$$[\mathbf{M}]^{\mathbf{e}} = \frac{A}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{3.19}$$

Logo, ao impormos a condição de convecção, a matriz LHS (matriz do lado esquerdo) dos elementos volumétricos que compartilham uma face no contorno convectivo final é conseguida substituindo-se as equações 3.19 e 3.11 na equação 2.33:

$$[\mathbf{K}]^{\mathbf{e}} = k([\mathbf{K_y}]^{\mathbf{e}} + [\mathbf{K_z}]^{\mathbf{e}} + [\mathbf{K_x}]^{\mathbf{e}}) + \frac{hA}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{3.20}$$

Por fim, os vetores da equação 3.3 são obtidos através da matriz de massa, de tal forma que os termos do lado direito da equação 3.3 se tornam, respectivamente:

$$\frac{A}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}\begin{bmatrix} hT_\infty \\ hT_\infty \\ hT_\infty \end{bmatrix} \text{ e } \frac{A}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}\begin{bmatrix} q \\ q \\ q \end{bmatrix}$$

Uma vez montadas, as matrizes e vetores elementares devem ser montados em uma matriz do Laplaciano e um vetor carregamento globais para resolução do sistema linear da equação 2.22:

$$\mathbf{[K]\{T\} = \{f\}} \tag{2.22}$$

3.5 Implementação computacional

A estrutura computacional para resolver o problema proposto neste estudo é dividida em duas partes. Primeiramente, é usado um *script* em *Python* com o propósito de gerar a malha no *Gmsh* e o arquivo JSON de input para o MEF, calculando o ponto de trabalho e o coeficiente de transferência de calor por convecção correspondente.

Em seguida, é usado um pacote MEF também em *Python* a fim de resolver o modelo numérico da seção anterior, sendo responsável por ler e mapear a malha gerada no passo anterior, além de gerar os vetores e matrizes de MEF, aplicar as condições de contorno e resolver o sistema linear para obter o campo de temperaturas. Tanto o *script* quanto os módulos do pacote MEF serão apresentados no Apêndice A.

O diagrama a seguir da figura 3.6 demonstra o fluxo de trabalho da estrutura para resolver o problema proposto. Nas próximas seções, cada aspecto dos módulos *Python* e seus arquivos gerados serão abordados.

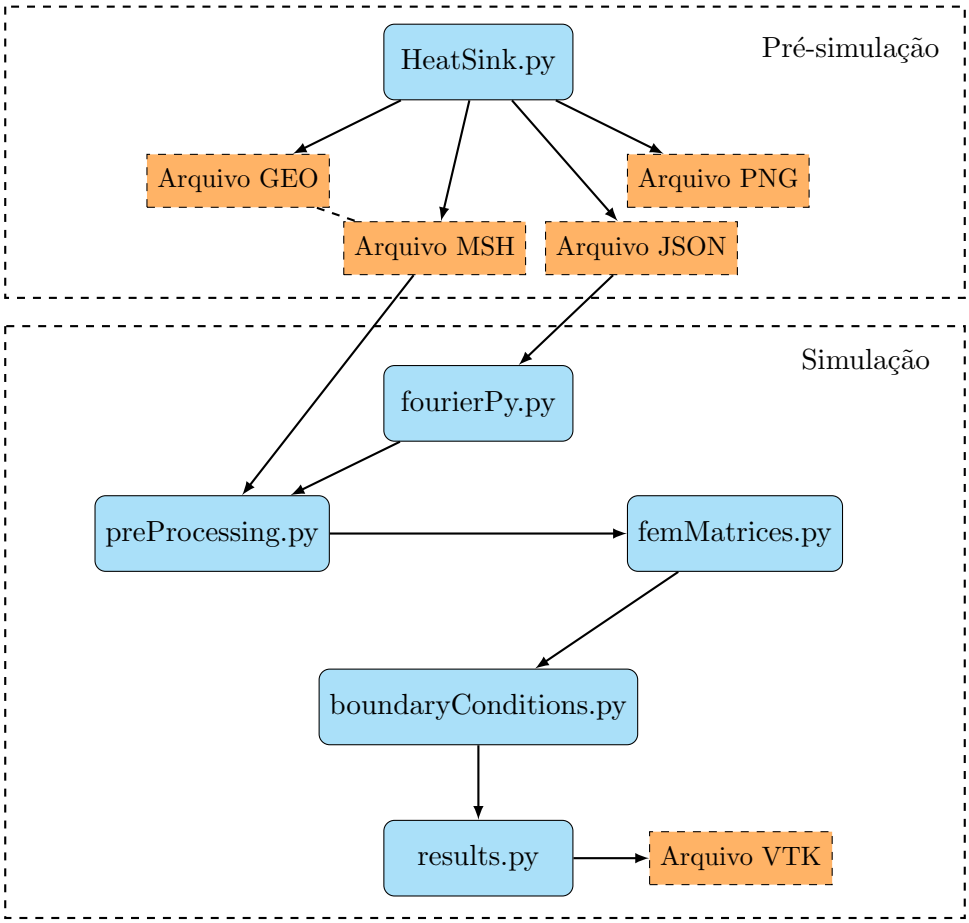


Figura 3.6: Diagrama esquemático do fluxo de trabalho da análise.

3.5.1 Script HeatSink.py

Toda a pré-simulação é realizada no *script HeatSink.py*. Nele, saídas personalizadas serão obtidas para cada conjunto de *inputs* a depender do número de aletas de cada dissipador.

Além do número n de aletas, o programa recebe como *inputs*: a espessura t das

aletas, sua altura H_f, as dimensões L e W do dissipador, e a altura da base do dissipador H_b. Por fim, recebe também os parâmetros de refino de malha ls, θ e *nDivThickness*, que serão discutidos no capítulo 4.

De posse dessas entradas, o programa irá criar um arquivo GEO, do *Gmsh*, adicionando pontos, linhas e superfícies (*elementary entities*), e daí extrudando as superfícies para formar a geometria 3D. Além disso, o arquivo GEO também é dotado dos *physical groups* que servirão como seleções nomeadas para as condições de contorno no pacote MEF.

A geração da geometria também inclui a criação de linhas e superfícies ditas "transfinitas" no arquivo de geometria, para a geração de malha estruturada nas aletas. Um arquivo MSH de malha do *Gmsh* é gerado em seguida com ajuda da biblioteca *subprocess*, da seguinte forma:

Excerto de algoritmo 3.1: Geração de malha e gravação do aquivo MSH em *Python*.

```
#MESH GENERATION
path = r"C:\Users\mduar\Downloads\gmsh-4.12.2-Windows64\
gmsh-4.12.2-Windows64\gmsh.exe"
geometry = str(n) + "fins" + str(tmm) + "mm" + str(hbmm) + "hbA.geo"
mesh = str(n) + "fins" + str(tmm) + "mm" + str(hbmm) + "hbA.msh"

command = [
    "gmsh",
    geometry,
    "-3", # Malha 3D
    "-format", "msh2", # Arquivo MSH versao 2
    "-o", mesh, ".msh"
]
subprocess.run(command, check=True)
```

O programa também irá gerar a curva (de impedância) do sistema, ou seja, irá prover um *array* de requisito de pressão para cada vazão indo de 1 CFM para 100 CFM, com incremento de 1 CFM, usando a função *pressureDrop(CFM)*. A vazão e pressão do ponto de trabalho serão obtidas pela interpolação da curva do sistema com a curva do ventilador correspondente à figura 3.1 usando *np.interp* da

biblioteca *Numpy*. A curva do ventilador, obtida da folha de dados da Sanyo Denki, foi discretizada em um arquivo XLSX, tendo sido lido pelo *HeatSink.py* com auxílio da biblioteca *Pandas*.

A curva do ventilador San Ace 9CRH0648P6G001, sobreposta à curva do sistema, é gerada em seguida, como no exemplo da figura 3.7.

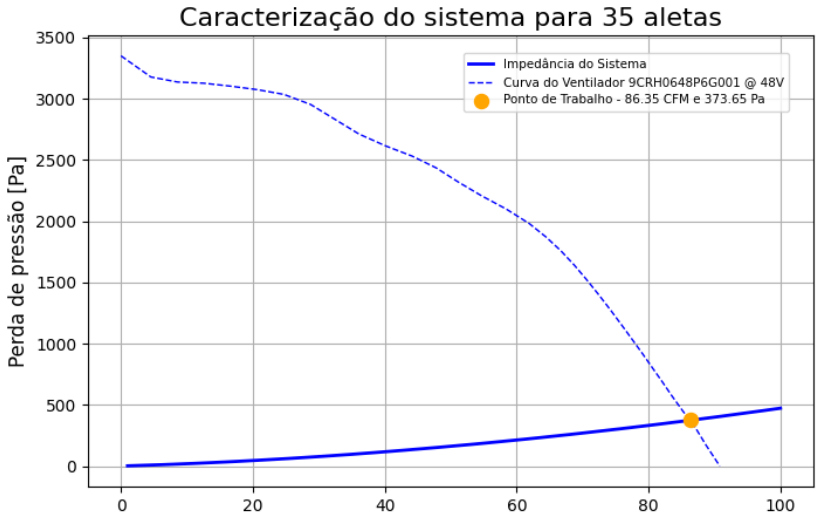


Figura 3.7: Obtenção do ponto de trabalho. Exemplo com 35 aletas.

Uma vez calculado o ponto de trabalho, a correlação desenvolvida em 2.6.1 é usada para calcular o coeficiente médio de transferência de calor, em [W/m^{2}·°C] com a função *convectionCoeficcient(CFM)*.

Por fim, é gerado um arquivo JSON com os dados que serão transportados para a simulação MEF. A seguir um exemplo de arquivo JSON.

Excerto de algoritmo 3.2: Exemplo de arquivo JSON para 35 aletas.

```
{
    "copper": {
        "alpha": "393",
        "g": "0"
    },
    "finsConvection": {
        "tipo": "3",
        "valor": "56.60029689119155"
    },
    "baseConvection": {
        "tipo": "3",
        "valor": "56.60029689119155"
    },
    "finsTop": {
        "tipo": "2",
        "valor": "0"
    },
    "sinkEnds": {
        "tipo": "2",
        "valor": "0"
    },
    "sinkSides": {
        "tipo": "2",
        "valor": "0"
    },
    "sinkBottom": {
        "tipo": "2",
        "valor": "46817.01398801028"
    }
}
```

3.5.2 Pacote fourierPy

O pacote *fourierPy* é constituído de 5 módulos mostrados na figura 3.6, os quais serão brevemente explicados nesta subseção, sendo eles:

1. *fourierPy.py* - Trata-se do módulo principal do pacote. Recebe os *inputs* do arquivo JSON ou do usuário, caso não exista um arquivo JSON. Serve para chamar as funções dos demais módulos.

2. *preProcessing.py* - Serve para ler e mapear a malha com a biblioteca *Meshio*. Gera as matrizes de condutividade do miolo (IEN) e do contorno (IENbound). Mapeia a geração de calor em cada elemento, e o elemento de contorno de cada condição.

3. *mefMatrices.py* - Neste módulo, são calculadas as matrizes de condutividade e de massa para todos os elementos da malha, para depois serem base para montagem das matrizes globais. Por fins de economia de memória RAM, a matriz [**K**] global é inicializada pela classe *scipy.sparse.lil_matrix*. O vetor global de carregamento é também criado a partir do produto da matriz de massa global com o vetor de geração de calor.

4. *boundaryConditions.py* - Aqui, a matriz de condutividade e o vetor carregamento globais são modificados com base na imposição de cada um dos três tipos de condição de contorno.

5. *Results* - Por fim, o sistema linear é resolvido. A matriz [**K**] é convertida para uma matriz esparsa do tipo *scipy.sparse.csc_matrix* e o sistema é resolvido pela função *sp.solve* da biblioteca *Scipy*. Uma vez resolvido o sistema o *array* relativo ao campo de temperaturas é obtido. O módulo gera um arquivo VTK, que permite que o resultado seja visualizado no *software Paraview*.

3.6 Parâmetros da análise

O domínio das simulações consiste num dissipador de calor de cobre como discutido na seção 3.2. Suas dimensões básicas são mostradas na figura 3.2 e reunidas na tabela 3.1:

Tabela 3.1: Dimensões do dissipador de calor.

W [mm]	L [mm]	H_f [mm]	H_b [mm]	t [mm]	b [mm]
77,5	56,5	60	4	1	$\frac{L-nt}{n-1}$

As dimensões da base são as mesmas da tampa da CPU escolhida, para eliminar uma resistência de espalhamento. A altura H_f das aletas é igual ao diâmetro do ventilador selecionado. A espessura t das aletas foi fixada em 1 mm. O único parâmetro variável é o espaçamento b entre as aletas, função do número n de aletas e do comprimento W do dissipador, considerando um passo constante de posicionamento das aletas.

A tabela 3.2 reúne as propriedades físicas e constantes adimensionais do ar e do cobre (material do dissipador), retiradas de [11], para uso nas correlações da seção 2.6 e na simulação em MEF.

Tabela 3.2: Propriedades e constantes adimensionais. Retiradas de [11].

k [W/m·°C]	Pr_{ar}	ρ_{ar} [kg/m^3]	μ_{ar} [Pa·s]	k_{ar} [W/m·°C]
393	0,71	1,13	0,000019	0,027

Antes de iniciar a análise, se faz necessário delimitar seu escopo. As correlações usadas para calcular a perda de carga e o coeficiente de transferência de calor médio apresentadas na seção 2.6 têm limites claros de aplicação.

A correlação apresentada por COPELAND (2000) para a perda de carga requer que o escoamento seja laminar. Logo, o número de Reynolds definido na equação 2.44 não pode exceder 2300 [18].

O *script HeatSink.py* foi utilizado para calcular o número de Reynolds para o ponto de trabalho no sistema dissipador-ventilador, partindo do número mínimo de 4 aletas até o número máximo de 72 aletas.

Substituindo a equação 2.45 do diâmetro hidráulico em 2.44, escrevendo a velocidade do ar entre as aletas U_{ar} em função da vazão volumétrica e dos parâmetros geométricos das aletas e substituindo a equação para o espaçamento b entre aletas da tabela 3.1, temos para cada ponto de trabalho:

$$Re = \frac{2\rho_{ar}\dot{V}p}{\mu(np - b)} \tag{3.21}$$

Na qual o passo p é definido na equação 3.22:

$$p = b + t \tag{3.22}$$

A equação 3.21 nos dá a variação do número de Reynolds com a vazão volumétrica e o número de aletas n. Como o sistema só pode operar no seu ponto de trabalho, a equação deve ser vista como a dependência do número de Reynolds unicamente com o número de aletas.

Como resultado, foi verificado que para $n < 35$ os escoamentos entram no regime de transição tubulenta. Portanto, $n = 35$ deve ser o limite inferior da análise.

Por sua vez, TEERSTRA et. al. (1999) afirmam em [8] que a análise é aplicável dentro da faixa de $0.1 < Re_b^* < 100$, em que Re_b^* é definido na equação 2.37.

De maneira análoga, ao escrevermos a velocidade do ar ao passar pelas aletas em função da vazão volumétrica e parâmetros geométricos do dissipador e substituirmos a relação para b presente na tabela 3.1, a equação 2.37 pode ser reescrita:

$$Re_b^* = \frac{\rho_{ar}\dot{V}p(L - nt)}{\mu L^2 H_f} \tag{3.23}$$

Foi verificado que para $n > 68$, obtemos valores de Re_b^* menores que 0.1. O valor de $n = 68$ deve portanto ser o limite superior do estudo.

Conforme descrito na seção 3.5, o coeficiente de transferência de calor foi então calculado para a vazão relativa ao ponto de trabalho de cada uma das 34 geometrias de dissipadores e carregado para arquivos JSON.

3.7 Controle dos resultados

Ao término cada simulação, os resultados considerados de temperatura média e máxima no fundo dos dissipadores de calor foram incluídos num arquivo XLSX juntamente com outros dados do arranjo. Os dados da planilha de resultados são apresentados de forma tabular no apêndice C. A seguir um exemplo de uma linha da planilha resultante na tabela 3.3.

Tabela 3.3: Exemplo de linha do arquivo XLSX.

Número de aletas	35
Espaçamento [mm]	1,25
Área de troca convectiva [m^2]	0,23
Pressão de trabalho [Pa]	373,65
Vazão de trabalho [CFM]	86,35
Coef. h de trabalho [$W/m^2°C$]	56,60
Eficiência η das aletas	0,68
Número de tetraedros	713328
Temp. Máx. no Fundo [°C]	61,24
Temp. Méd. no Fundo [°C]	61,14

Capítulo 4

Verificação

Para o uso do pacote MEF descrito na seção 3.5.2 na resolução do problema proposto, o mesmo deve ser verificado e validado com base em dois aspectos aprofundados:

- O *software* deve ser capaz de resolver, com erro considerado baixo, um problema de condução de calor em sólidos no qual sejam impostas as mesmas condições de contorno das que serão encontradas no presente problema;
- Uma vez o *software* validado, o mesmo deve se traduzir em uma análise com resultados realistas, minimizando ao máximo o erro sem um refino exagerado de malha.

4.1 Validação por comparação com solução analítica

Primeiramente o código foi validado resolvendo-se um problema com solução analítica. Para tal, foi escolhido um problema de ÖZİŞİK (1993) [10], no qual há um fluxo de calor q_0'' não nulo em um dos lados de uma parede plana infinita. No outro lado da parede a mesma troca calor por convecção a um fluido à temperatura T_∞ com coeficente h, resultando no problema 1D da figura 4.1.

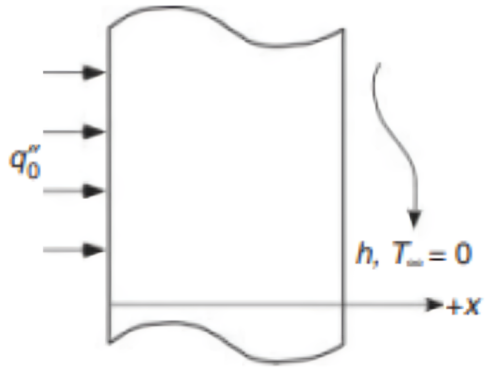


Figura 4.1: Problema térmico para validação do código MEF. De [10].

Tal problema tem a seguinte solução dependente do tempo:

$$T(x,t) = \sum_{n=1}^{\infty} C_n \cos(\lambda_n x) e^{-\alpha \lambda_n^2 nt} + \frac{q_0''}{k}(L-x) + \frac{q_0''}{h} \tag{4.1}$$

Em regime permanente, a solução se reduz a:

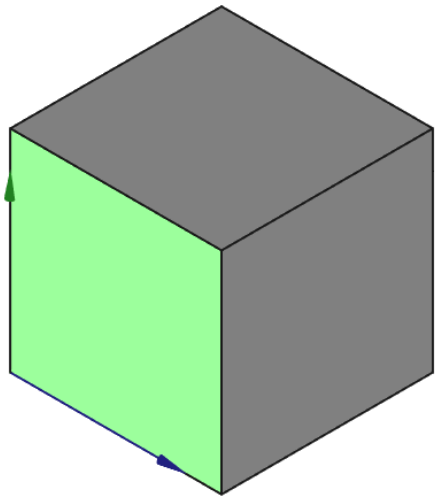
$$T(x) = \frac{q_0''}{k}(L-x) + \frac{q_0''}{h} \tag{4.2}$$

Para testar o código, foi escolhido como geometria um cubo de dimensões unitárias. Para captar os efeitos da transferência unidimensional de calor, foi controlada a temperatura no eixo de simetria do cubo ao longo do eixo x quando o mesmo é exposto ao fluxo de calor não nulo em sua face esquerda, cuja normal é $-\hat{x}$ e à convecção em sua face direita, de normal $\hat{x}$, seguindo as entradas da tabela 4.1:

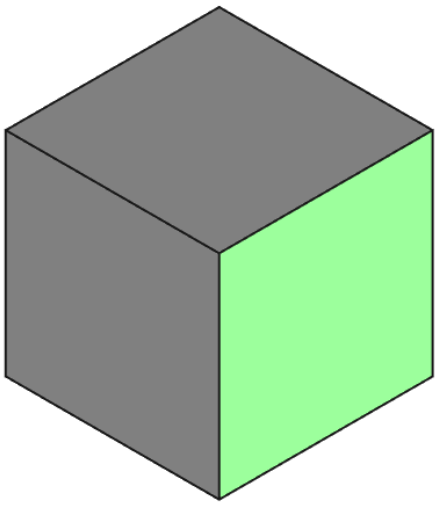
Tabela 4.1: *Inputs* do problema para validação do código MEF.

k [W/(m·°C)]	q_0'' [W/m^2]	h [W/(m^2·°C)]	T [°C]
200	500	50	0

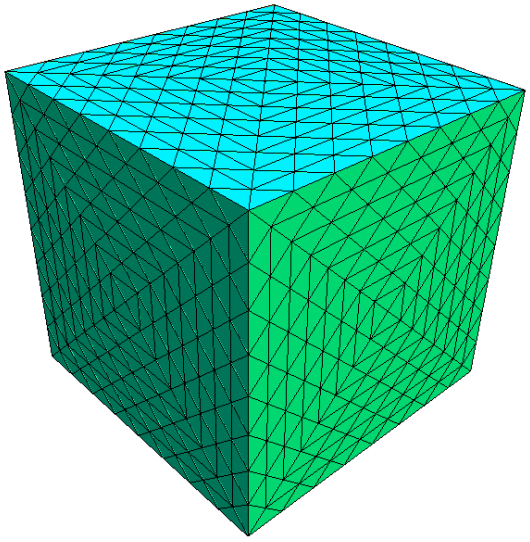
As condições de contorno do problema podem ser visualizadas nas figuras 4.2(a), 4.2(b) e 4.2(c).



((a)) Face esquerda: fluxo de calor q_0''.



((b)) Face direita: convecção.



((c)) Malha do cubo utilizado para simulação de validação do código MEF com 12288 tetraedros.

Figura 4.2: Condições de contorno e malha para caso de validação do pacote MEF.

Como saída do pacote MEF, obtemos a solução numérica da distribuição de temperatura nos 9 pontos no eixo de simetria em x do cubo. A solução analítica (equação 4.2) é então representada sobreposta à solução numérica na figura 4.3:

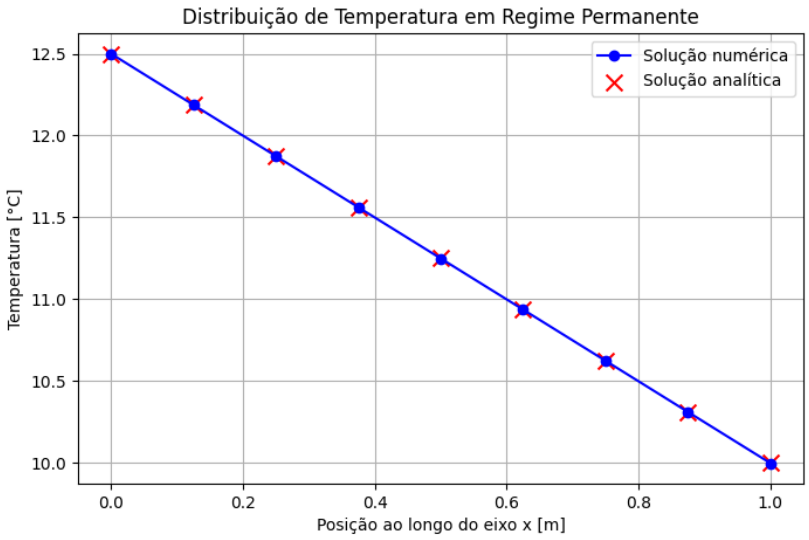


Figura 4.3: Resultados da simulação de validação *versus* solução analítica.

Agora, apresentando as temperaturas analíticas (T_{ref}) e numéricas (T_{calc}) para os 9 pontos de forma tabular, é possível perceber erros relativos muito baixos, da ordem de 10^{-11}. Podemos, portanto, afirmar que o *software* obteve uma aproximação satisfatória para a equação do calor dado o problema proposto.

Tabela 4.2: Resultados da simulação de validação do pacote MEF.

Ponto	x [m]	T_{ref} [°C]	T_{calc} [°C]	Erro relativo (%)
1	0	12,5	12,50000	3,07665e-11
2	0,125	12,1875	12,18750	3,14388e-11
3	0,25	11,875	11,87500	3,19969e-11
4	0,375	11,5625	11,56250	3,25390e-11
5	0,5	11,25	11,25000	3,29060e-11
6	0,625	10,9375	10,93750	3,33590e-11
7	0,75	10,625	10,62500	3,35711e-11
8	0,875	10,3125	10,31250	3,37960e-11
9	1	10	10,00000	3,39995e-11

4.2 Estudo de convergência de malha

Uma vez validado o código, é possível proceder com as simulações do modelo proposto. Faz-se necessário, à priori, realizar um estudo de convergência de malha para garantir que tenhamos o menor erro possível sem compromissos computacionais causados por refinos de malha além do necessário.

Partindo de uma malha mais grosseira, há uma tendência de, a cada refino, o erro diminuir até certo ponto em que refinos subsequentes não acarretariam melhoras no resultado, senão apenas um custo e tempo maiores de computação.

Para a realização deste estudo, foi tomado como geometria um dissipador de calor de apenas 4 aletas. A ideia do estudo consiste em refinar a malha desta geometria sucessivamente até que o erro chegue a um platô e, então, transportar os parâmetros de geração de malha para todas as outras geometrias.

O erro relativo é definido, neste caso, com T_1 e T_2 sendo a temperatura (média ou máxima) em iterações subsequentes, na equação 4.3. Foram então criados os seguintes parâmetros de refino:

- Parâmetro ls - parâmetro inerente à criação de um ponto enquanto entidade elementar no *Gmsh* que dita o tamanho de um elemento de malha no ponto em questão, impactando o tamanho de todos os elementos de malha nas suas proximidades. Neste trabalho, a premissa de haver no mínimo 5 elementos na espessura da base do dissipador foi tomada. Logo, o valor máximo de ls adotado foi de 0,0009, uma vez que valores maiores, como passo de 0,0001 acarretariam malhas estruturadas demasiado grossas. O parâmetro ls foi aplicado aos pontos destacados na figura 4.4. As figuras 4.5(a) e 4.5(b) exemplificam o efeito da variação de ls;

- Parâmetro θ - fator multiplicativo à altura e à largura das aletas resultando respectivamente no número de divisões da malha estruturada das aletas em sua altura e comprimento, de modo que: $nDivHeight = \theta H_f$ e $nDivExtrusion = \theta L$. As figuras 4.5(c) e 4.5(d) exemplificam o efeito da variação de θ;

- Parâmetro *nDivThickness* - Dá o número de pontos na espessura da aleta, de forma que o número de elementos em sua espessura é dado por

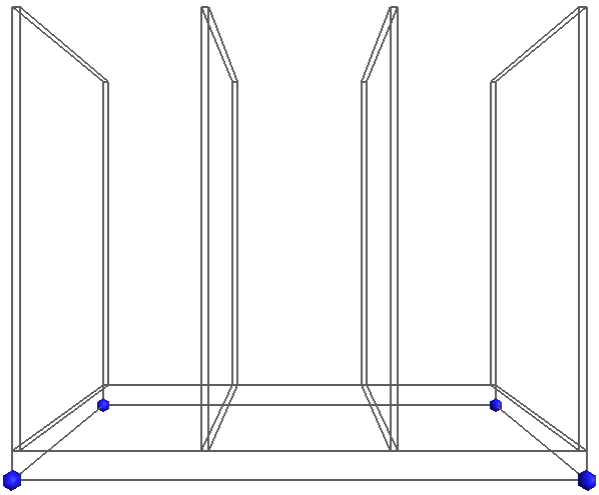


Figura 4.4: Pontos nos quais é aplicado o parâmetro ls.

$nDivThickness - 1$. Foi fixado em 6 nesta análise, com 5 elementos na espessura da aleta.

Foram performadas simulações controlando as temperaturas média e máxima do fundo do dissipador de calor de 4 aletas de cobre. Os dados de entrada para todas as simulações foram obtidos para o ponto de trabalho do sistema-ventilador (figura 4.6), tendo sido obtido um coeficiente de transferência de calor por convecção igual a 39,87 W/(m·°C).

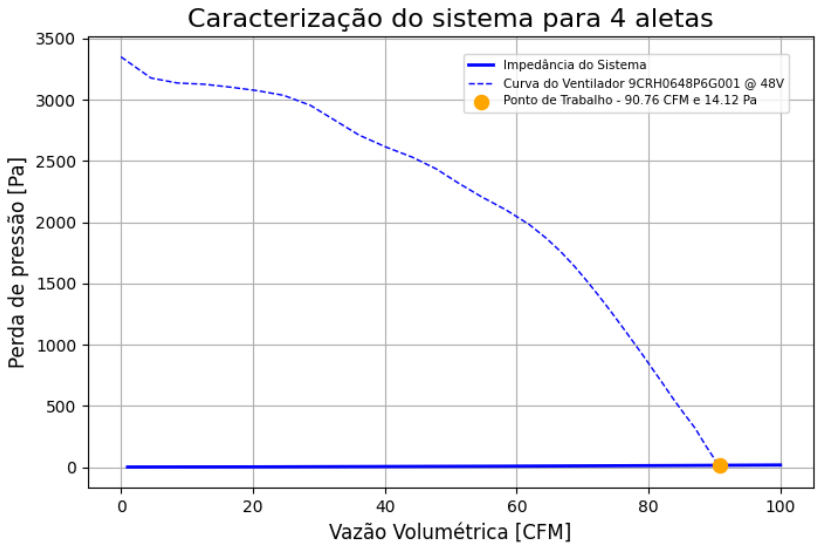
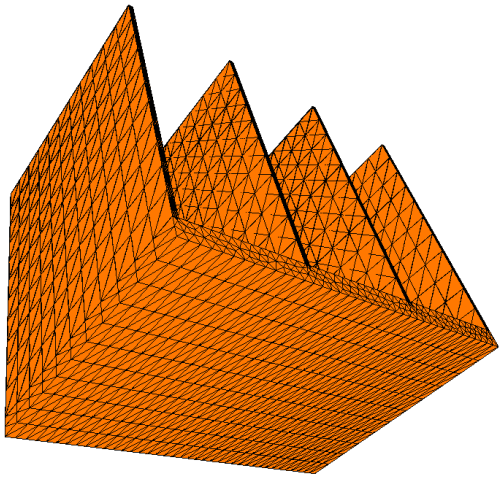
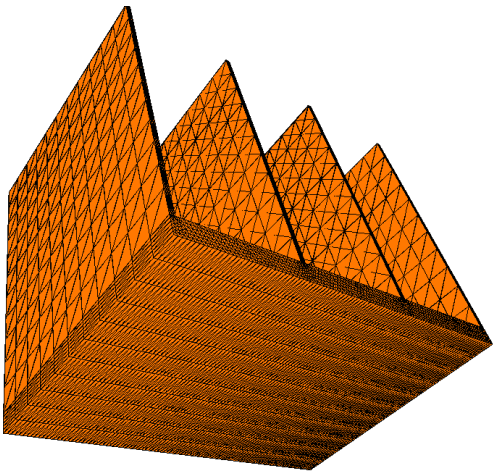


Figura 4.6: Obtenção do ponto de trabalho para 4 aletas.

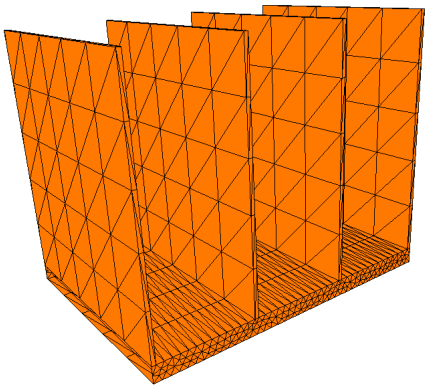
Primeiramente, variou-se o parâmetro θ mantendo-se fixo $ls = 0{,}0009$. A tabela



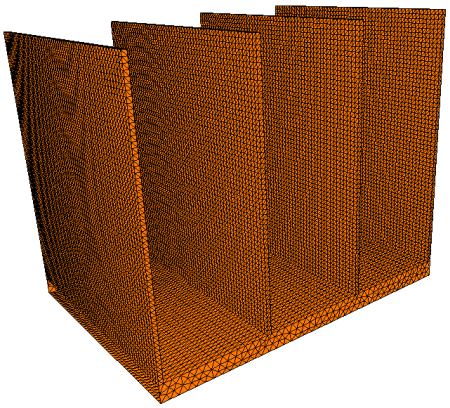
((a)) Malha com $ls = 0.002$.



((b)) Malha com $ls = 0.00075$.



((c)) Malha com $\theta = 0.1$.



((d)) Malha com $\theta = 1$.

Figura 4.5: Refinos pontuais nos 4 pontos do fundo do dissipador (a) e (b) e refinos com diferentes números de divisões na altura e comprimento das aletas mantendo-se os demais parâmetros de refino invariáveis (c) e (d).

4.3 reúne os resultados desta primeira análise com dados das temperaturas média $T_{méd}$ e máxima $T_{máx}$ no fundo do dissipador.

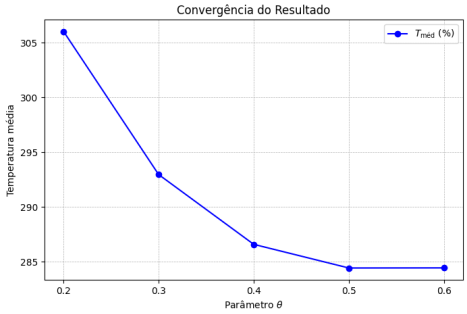
Onde o resultado é comparado com a malha anterior mais grossa, e o erro é dado por:

$$\text{Erro relativo} = \frac{|T_2 - T_1|}{T_1} \times 100 \tag{4.3}$$

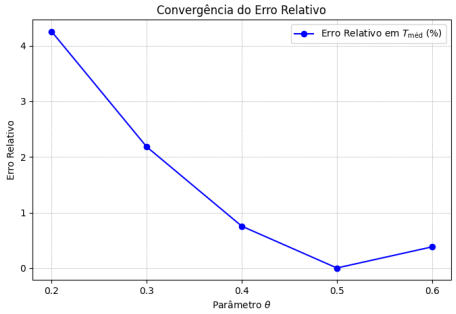
Os gráficos abaixo das figuras 4.7 e 4.7(b) mostram a evolução da temperatura média ao longo das iterações variando θ.

Tabela 4.3: Influência do parâmetro θ na acurácia do campo de temperaturas resultante para o dissipador de 4 aletas.

θ	$T_{méd}$ [°C]	$T_{máx}$ [°C]	Erro em $T_{méd}$ (%)	Erro em $T_{máx}$ (%)
0.1	306.05	307.36	-	-
0.2	292.97	293.71	4.2570	4.4364
0.3	286.59	287.42	2.1826	2.1451
0.4	284.43	285.13	0.7554	0.7970
0.5	284.44	285.13	0.0035	$\approx$ 0.0000
0.6	285.53	286.23	0.3828	0.3855



((a)) Convergência da temperatura média com variação de θ.



((b)) Convergência do erro na temperatura média com relação à malha anterior.

Figura 4.7: Influência do parâmetro θ na temperatura média obtida no fundo do dissipador de 4 aletas.

Como a temperatura começa a se estabilizar com erro relativo menor que 1 % a partir de $\theta = 0{,}4$, este valor é escolhido. Fixando $\theta = 0{,}4$ e refinando ls não obtemos melhoras significativas no resultado, com erros menores que 1 % com relação ao refino anterior, conforme tabela 4.4:

Tabela 4.4: Influência do parâmetro ls na acurácia do campo de temperaturas resultante para o dissipador de 4 aletas.

ls	$T_{méd}$ [°C]	$T_{máx}$ [°C]	Erro em $T_{méd}$ (%)	Erro em $T_{máx}$ (%)
0.0009	284.43	285.13	-	-
0.0008	284.45	285.15	0.0070	0.0070
0.0007	285.34	286.04	0.3129	0.3123
0.0006	285.35	286.05	0.0035	0.0035

Ao fim do estudo de convergência de malha, os parâmetros estudados foram analisados e selecionados, tais que $ls = 0{,}0009$, $\theta = 0{,}4$ e $nDivThickness = 6$.

A malha final da análise para 4 aletas pode ser vista na figura 4.8. Os mesmos parâmetros foram então transportados para todas as geometria analisadas e apresentadas no capítulo 5.

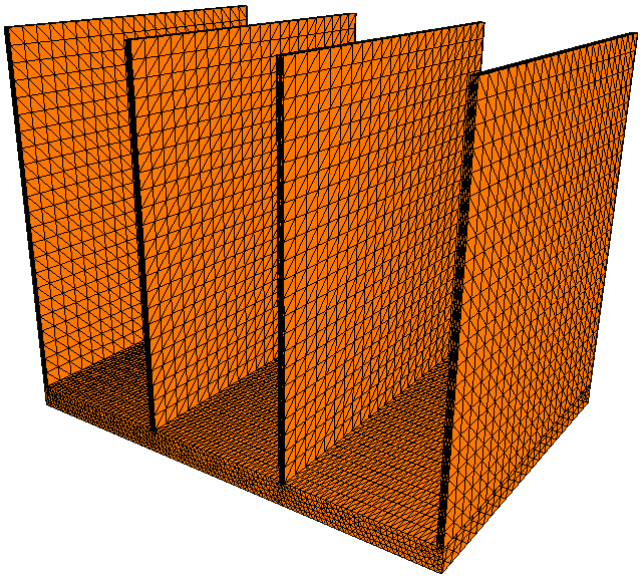


Figura 4.8: Malha final para 4 aletas.

Capítulo 5

Resultados e discussões

Nesse capítulo serão apresentados os resultados obtidos das simulações das 34 geometrias de dissipadores de calor, a fim de se obter a relação entre a resistência térmica de convecção do dissipador com seu número de aletas. O gráfico da figura 5.1 a seguir demonstra a variação do desempenho com o número de aletas do dissipador.

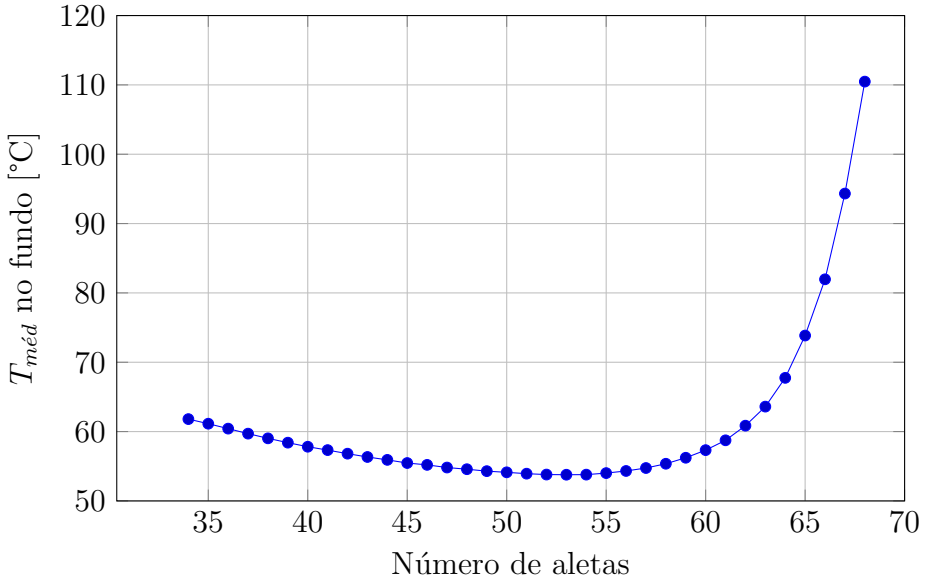


Figura 5.1: Temperatura média do dissipador de calor *versus* número de aletas.

Mensurando o desempenho do dissipador de calor como a capacidade de reduzir a temperatura em seu fundo - que seria a mesma temperatura da tampa do processador Intel Xeon 6710E - notamos uma tendência de melhora de desempenho num primeiro momento com o aumento do número n de aletas graças ao aumento de área de troca

convectiva.

Entretanto, a partir de um ponto ótimo, caracterizado por **n=53**, a temperatura do fundo começa a subir rapidamente, mesmo com o aumento de área de troca térmica. Tal comportamento é esperado e acontece pois a resistência térmica de convecção cresce muito em função da queda no coeficiente h. Na figura 5.2, a curva de variação do coeficiente de transferência de calor por convecção (curva em vermelho) é sobreposta à curva da figura 5.1.

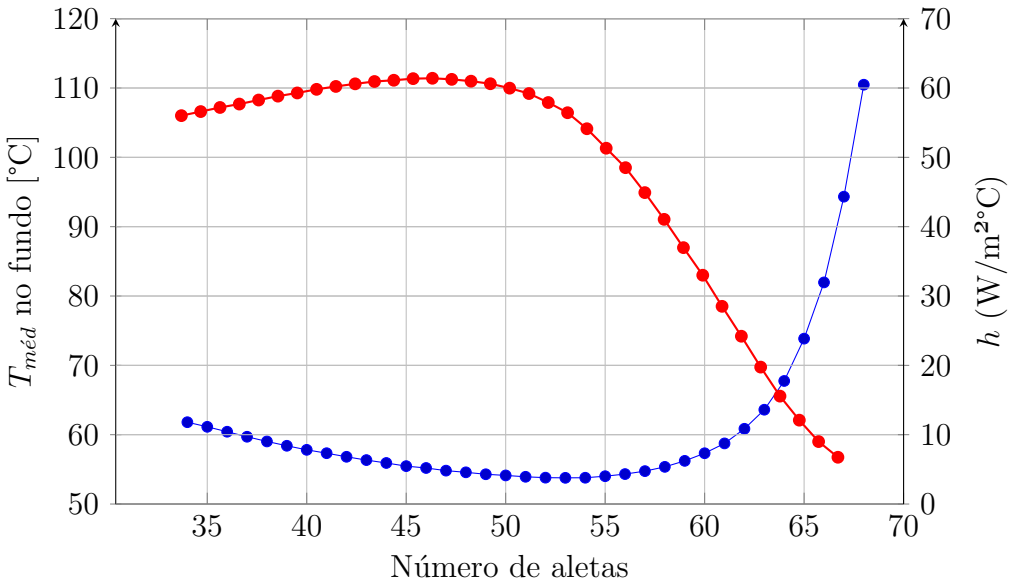


Figura 5.2: Coeficiente de transferência de calor e temperatura média no fundo do dissipador *versus* número de aletas.

Da seção 2.6.1 considerando a condutividade térmica do ar constante, vemos que o coeficiente h é função no número de Nusselt real Nu_b e do espaçamento b entre as aletas de acordo com equação 2.42. Com o aumento do número de aletas, b diminui e há tendência de aumento em h. Entretanto, o comportamento de Nu_b provoca o efeito contrário.

Os dados da simulação e a obtenção do ponto de trabalho para a geometria que otimiza a transferência de calor desde o processador são apresentados a seguir na tabela 5.1 e na figura 5.5. A malha gerada para esta geometria pode ser vista na figura 5.6.

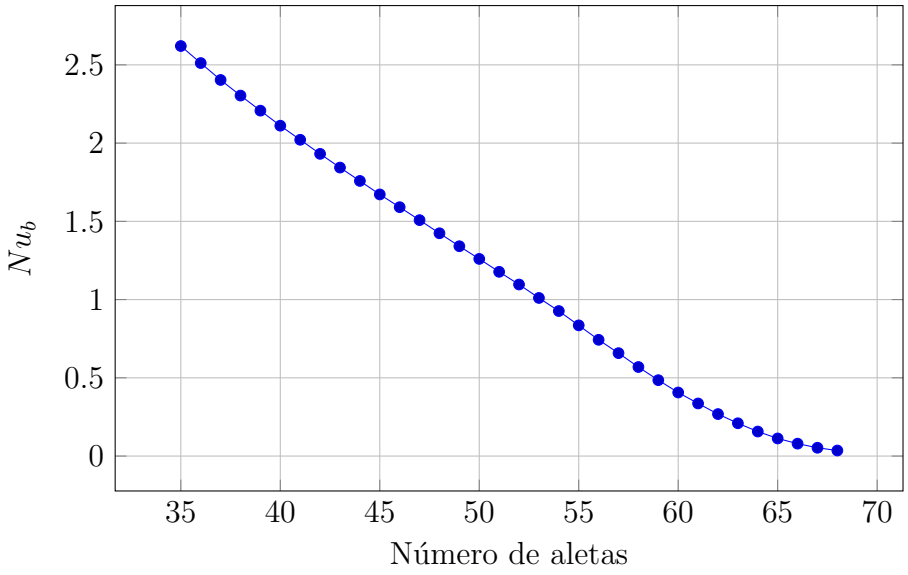


Figura 5.3: Número de Nusselt real Nu_b *versus* número de aletas.

Tabela 5.1: Dados da simulação do dissipador de calor de 53 aletas.

n	$\dot{V}$ [CFM]	p [Pa]	h [W/m^2·°C]	T_{ar} [°C]	q [W/m^2]	$T_{méd}$ [°C]	tempo total [s]	Tetraedros
53	65,86	1785,83	57,91	40	46817	53,8	1482	1072896

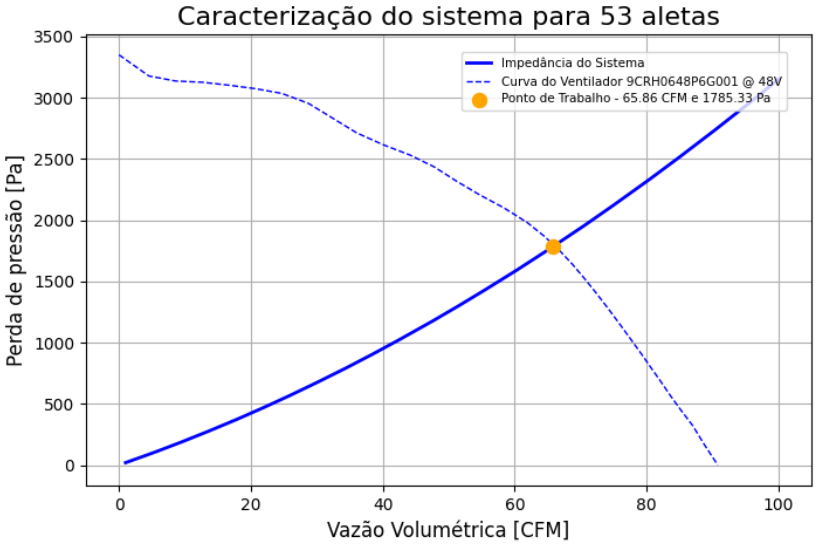


Figura 5.5: Ponto de trabalho para dissipador com 53 aletas.

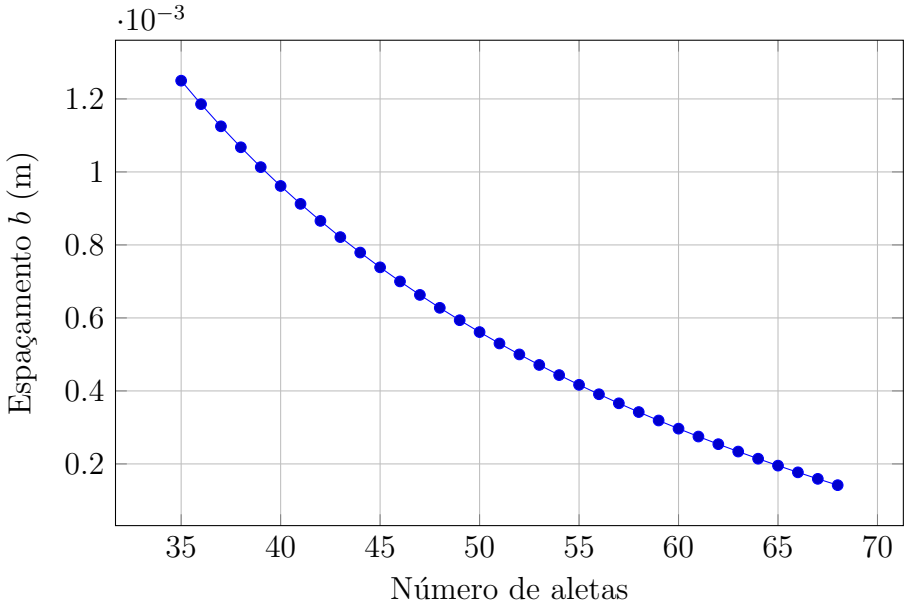


Figura 5.4: Espaçamento b em função do número de aletas.

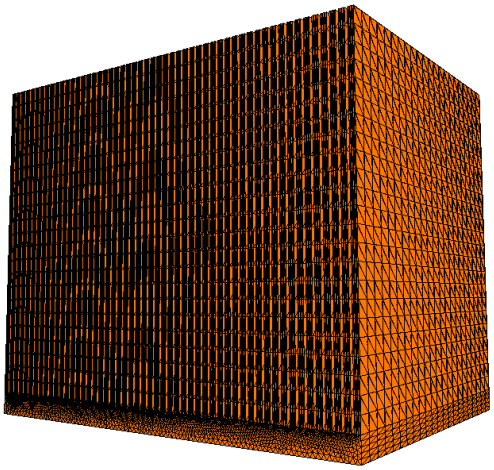
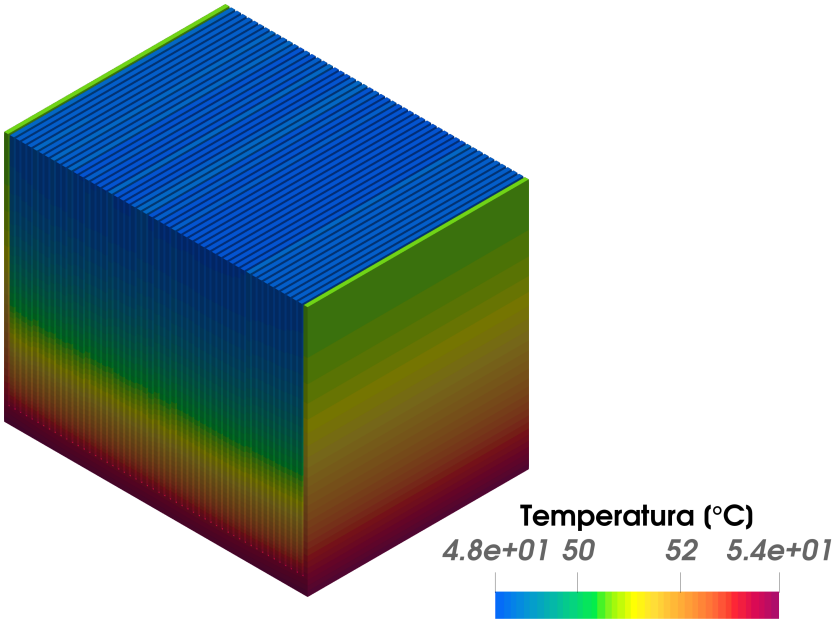


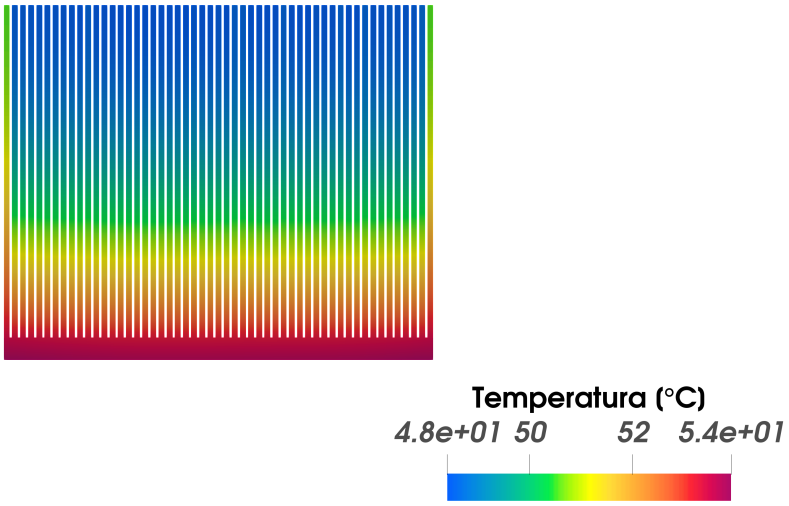
Figura 5.6: Malha gerada para a simulação do dissipador de calor de 53 aletas com 1072896 tetraedros.

Uma vez atingido o objetivo de se encontrar o número ótimo de aletas, pode-se usar o arquivo VTK correspondente gerado pelo pacote MEF para se realizar a visualização dos resultados. Para tal, foi utilizado o *software Paraview*.

A distribuição de temperaturas obtidas está dentro do esperado, tal como apresentada nas figuras 5.7(a) e 5.7(b), dadas as condições de contorno do problema, uma vez que as aletas mais externas do dissipador só trocam calor pela sua face mais interna, por convecção.



((a)) Vista isométrica.



((b)) Vista frontal.

Figura 5.7: Distribuição de temperaturas no dissipador de calor de 53 aletas.

Capítulo 6

Conclusão

A presente análise atingiu seu objetivo de encontrar o número ótimo de aletas em um dissipador de calor de aletas retas adequado para o resfriamento de um processador Intel Xeon 6710E por convecção forçada com ventilador Sanyo Denki San Ace 9CRH0648P6G001. Ao mesmo tempo, a temperatura média encontrada para o dissipador com número ótimo de aletas foi de 53,8 °C, bem abaixo da temperatura máxima permitida na tampa pelo fabricante do processador, igual a 85 °C. O problema em questão pode ser caracterizado como de otimização, uma vez que os ganhos trazidos pelo aumento de área de troca térmica são, depois de um certo ponto, freados pela redução progressiva do coeficiente de transferência de calor por convecção.

O estudo de otimização envolveu simulações numéricas pelo Método dos Elementos Finitos de todas as geometrias de dissipadores dentro dos limites de aplicação das correlações da seção 2.6, considerando, entre outras coisas, que a base do dissipador tenha as mesmas dimensões da tampa do processador, e que o escoamento de ar é completamente canalizado, isto é, sem perdas por *bypass* ou *tip clearance*.

Pode-se dizer que o ventilador San Ace 9CRH0648P6G001 da Sanyo Denki foi corretamente selecionado, uma vez que "cobre" todas as curvas de impedância dos sistemas dissipador para qualquer arranjo de aletas, ou seja, a curva do ventilador intersecta com a curva do sistema de todos os dissipadores, apresentando alta faixa de pressões. Desta forma foi possível obter o ponto de trabalho para todas as geometrias de dissipador, sobre o qual foi calculado cada coeficiente de transferência de calor.

O código MEF de autoria própria se mostrou adequado a resolver o modelo proposto, e obteve resultados dentro do esperado considerando-se a natureza física do problema. Além disso, o programa foi verificado e validado frente a um problema com formulação matemática similar, e obteve erros relativos baixos - inferiores a 1% - em estudo de convergência de malha; ou seja, tanto o solver de MEF como o processo de geração de malha foram validados.

Por fim, o presente trabalho resultou em um método de análise de dissipadores de calor aplicável a inúmeras geometrias a partir das entradas do usuário para os parâmetros geométricos no *script* HeatSink.py.

6.1 Sugestões para trabalhos futuros

Os seguintes pontos podem ser abordados em desenvolvimentos futuros a fim de preencher limitações do presente trabalho.

- Inclusão de mais componentes do encapsulamento da CPU na análise como: *die*, PCB, IHS, além das TIM1 e TIM2, com o calor não indo integralmente para o dissipador;
- Desenvolvimento de um estudo como o atual porém contemplando perdas de fluxo por *bypass* e *tip clearance*, bem como resistência térmica de espalhamento;
- Promover um estudo de otimização automatizado;
- Realização de simulações do escoamento do ar de modo a obter o coeficiente de transferência de calor, acoplando-se à análise da condução;
- Aprimoramento do código de elementos finitos atual, com suporte a elementos de ordem superior;
- Catálogo de mais processadores e ventiladores, permitindo que o usuário do programa simule diversas combinações através de uma interface gráfica.

Referências Bibliográficas

[1] CHOWDHURY, A. S. M. R., RABBY, M. M., KABIR, M., et al., "A Comparative Study of Thermal Aging Effect on the Properties of Silicone-Based and Silicone-Free Thermal Gap Filler Materials", *Materials*, v. 14, pp. 3565, 06 2021.

[2] WINSHARE THERMALLOY, "Aluminum Heat Pipe Heatsink Cooling Heat Sink", `https://www.winsharethermalloy.com/aluminum-heat-pipe-heatsink-cooling-heat-sink.html`, Acessado em: 15/02/2025.

[3] OBEROI, D. A. S., "Analysis of Heat Transfer Through Perforated Plate Heat Sink", 08 2012.

[4] FOX, R. W., MCDONALD, A. T., PRITCHARD, P. J., *Introdução à Mecânica dos Fluidos*. 7th ed. LTC: Rio de Janeiro, 2010.

[5] ZIENKIEWICZ, O. C., TAYLOR, R. L., ZHU, J. Z., *The Finite Element Method: Volume 1*. 5th ed. Butterworth-Heinemann, 2000.

[6] NITHIARASU, P., LEWIS, R. W., SEETHARAMU, K. N., *Fundamentals of the Finite Element Method for Heat and Mass Transfer. 2 ed.*. Wiley: Chichester, West Sussex, 2016.

[7] "Designing Heatsinks and Thermal Solutions for Xilinx Devices", AMD Xilinx Application Note XAPP1377, 2022, Disponível em: `https://www.xilinx.com`.

[8] TEERTSTRA, P., YOVANOVICH, M., CULHAM, J., et al., "Analytical forced convection modeling of plate fin heat sinks". In: *Fifteenth Annual IEEE*

Semiconductor Thermal Measurement and Management Symposium (Cat. No.99CH36306), pp. 34–41, 1999.

[9] SANYO DENKI, "San Ace 60 9CRH TYPE Counter Rotating Fan", `https://www.sanyodenki.com`, 2025, Acessado em: 22 fev. 2025.

[10] ÖZIŞIK, M. N., *Heat Conduction*. John Wiley & Sons, 1993.

[11] BEJAN, A., *Heat Transfer*. Wiley, 1992.

[12] AXSOM, T., "Heat Sink Design Guide & Considerations", `https://www.fictiv.com/articles/heat-sink-design-guide`, 2022, Acessado em: 15 fev. 2025.

[13] SARVAR, F., WHALLEY, D. C., CONWAY, P. P., "Thermal Interface Materials - A Review of the State of the Art". In: *2006 1st Electronic Systemintegration Technology Conference*, v. 2, pp. 1292–1302, 2006.

[14] MACHINING, A., "Heat Sink Design: Basics, Principle, and Practical Tips", `https://at-machining.com/heat-sink-design/`, 2023, Acessado em: 15 fev. 2025.

[15] CELSIA, "Heat Sink Design Fundamentals", `https://celsiainc.com/technology/heat-sink-design/`, Acessado em: 15 fev. 2025.

[16] LEE, S., "How to Select a Heat Sink", *Advanced Thermal Engineering, Aavid Thermal Technologies, Inc., Laconia*, 1995.

[17] WAYKEN, "Heat Sink Design Basics: Factors & Calculation", `https://waykenrm.com/blogs/heat-sink-design-basics/`, Acessado em: 15 fev. 2025.

[18] COPELAND, D., "Optimization of parallel plate heatsinks for forced convection". In: *Sixteenth Annual IEEE Semiconductor Thermal Measurement and Management Symposium (Cat. No.00CH37068)*, pp. 266–272, 2000.

[19] INTEL CORPORATION, "Intel Xeon 6710E Processor Specifications", 2025, Acessado em: 22 fev. 2025.

[20] WYLEN, G. J. VAN; SONNTAG, R. E. B. C., *Fundamentos da Termodinâmica.*. Blucher, 1998.

[21] ÖZIŞIK, M. N., *Heat Transfer - A Basic Approach*. McGraw-Hill, 1985.

[22] BIRD, R. B., STEWART, W. E., LIGHTFOOT, E. N., *Transport Phenomena*. John Wiley & Sons, Inc.: New York, 1960.

[23] LIU, W. K., LI, S., PARK, H. S., "Eighty Years of the Finite Element Method: Birth, Evolution, and Future", *Archives of Computational Methods in Engineering*, v. 29, pp. 4431–4453, 2022.

[24] COURANT, R., "Variational methods for the solution of problems of equilibrium and vibrations", *Bulletin of the American Mathematical Society*, v. 49, n. 1, pp. 1 – 23, 1943.

[25] FISH, J., BELYTSCHKO, T., *Um primeiro curso em elementos finitos*. LTC: Rio de Janeiro, 2009.

[26] MANURUNG, I., PURBA, F., NAINGGOLAN, Y., et al., "Center Processing Unit Components", 06 2021.

[27] CHURCHILL, S. W., USAGI, R., "A General Expression for the Correlation of Rates of Transfer and Other Phenomena", *AIChE Journal*, v. 18, n. 6, pp. 1121–1128, 1972.

[28] COPELAND, D., "Manifold Microchannel Heat Sinks: Analysis and Optimization", *Thermal Science and Engineering*, v. 3, n. 1, pp. 7–12, 1995.

[29] SHAH, R. K., LONDON, A. L., *Laminar Flow Forced Convection in Ducts. Advances in Heat Transfer, Supplement 1*, Academic Press, 1978.

[30] KAYS, W. M., LONDON, A. L., *Compact Heat Exchangers*. McGraw-Hill, 1984.

Apêndice A

Código Fonte

A.1 Pré-simulação - script HeatSink.py

```
def pressureDrop(CFM):
    #Vazao e velocidade
    vfr = 0.000471947443*CFM
    vArr = round(vfr/(W*sigma*hf), 2)
    re = rho*dh*vArr/mu
    xplus = L/(re*dh)
    hydHead = 0.5*rho*vArr**2

    #Entrada e saida
    kc = 0.4*(1-sigma**2) + 0.4
    ke = (1-sigma)**2 - 0.4*sigma

    #Perdas viscosas
    G = ((s/hf)**2+1)/(((s/hf)+1)**2)
    fRe = 19.64*G + 4.7
    fappRe = ((3.2*(xplus**-0.57))**2 + fRe**2)**0.5

    #Perda de carga total
    deltaP = (kc + 4*fappRe*xplus + ke)*hydHead

    return deltaP, re
```

```
def convectionCoeficcient(CFM):
    #Vazao e velocidade
    vfr = 0.000471947443*CFM
    vArr = round(vfr/(W*sigma*hf), 2)

    #Constantes adimensionais
    res = rho*s*vArr/mu
    resStar = res*s/L
    rede = rho*2*s*vArr/mu

    #Completamente desenvolvido
    nufd = 0.5*resStar*pr

    #Em desenvolvimento
    gzde = rede*pr*de/L
    nude = ((0.664*((gzde)**0.5)/(pr**(1/6))))*(1 +
        (7.3*(pr/gzde)**0.5))**0.5
    nudev =
        0.664*(resStar**0.5)*(pr**(1/3))*(1+(3.65/(resStar**0.5)))**0.5

    #Modelo composto
    n = 3
    nui = ((1/nufd**n) + (1/nudev**n))**(-1.0/n)

    #Nusselt real
    eta = np.tanh((2*nui*kar*hf*hf*((t/L)+1)/(khs*s*t))**0.5)
        /(2*nui*kar*hf*hf*((t/L)+1)/(khs*s*t))**0.5
    nus = eta*nui

    #Coeficiente de transf. de calor
    h1 = nus*kar/s

    return h1, resStar
```

```
#PARAMETROS -----------------------------
#Parametros Geometricos
n = 53  #input
tmm = 1  #input
t = tmm/1000
Wmm = 77.5  #input
W = Wmm/1000
s = (W - n*t)/(n - 1)
Lmm = 56.5  #input
L = Lmm/1000
Hmm = 64.0  #input
H = Hmm/1000
hbmm = 4  #input
hb = hbmm/1000
hf = H - hb
sigma = s/(s+t)

baseArea = W*L
exchangeArea = (n-1)*s*L + 2*(n-1)*(H-hb)*L

dh = 2*s*hf/(s+hf)
de = 2*s

#Parametros de refino de malha
ls = 0.0009 #input
theta = 0.4  #input
nDivThickness = 6 #input
nDivHeight = theta*hf*1000
nDivExtrusion = theta*Lmm

#Propriedades fisicas
rho = 1.13
mu = 0.000019
```

```
kar = 0.027
khs = 393
pr = 0.71

#GEOMETRY BUILDER ----------------------
nPoints = 10
nLines = 11
nSurfaces = 3
nVolumes = 3
nLoops = 3
nTransfiniteSurf = 3

#Elementary Entities
#Base e aletas das extremidades
with open(str(n) + "fins" + str(tmm) + "mm" + str(hbmm) + "hbA.geo", "w")
    as f:
    f.write("SetFactory(" + "\"OpenCASCADE\"" + ");\n")
    f.write("Point(1) = {0, 0, 0, " + str(ls) + "};\n")
    f.write("Point(2) = {" + str(W) + ", 0, 0, " + str(ls) + "};\n")
    f.write("Point(3) = {0, " + str(hb) + ", 0, " + str(ls) + "};\n")
    f.write("Point(4) = {" + str(W) + ", " + str(hb) + ", 0, " + str(ls)
        + "};\n")
    f.write("Point(5) = {0, " + str(H) + ", 0, " + str(ls) + "};\n")
    f.write("Point(6) = {" + str(W) + ", " + str(H) + ", 0, " + str(ls) +
        "};\n")
    f.write("Point(7) = {" + str(t) + ", " + str(hb) + ", 0, " + str(ls)
        + "};\n")
    f.write("Point(8) = {" + str(W-t) + ", " + str(hb) + ", 0, " +
        str(ls) + "};\n")
    f.write("Point(9) = {" + str(t) + ", " + str(H) + ", 0, " + str(ls) +
        "};\n")
    f.write("Point(10) = {" + str(W-t) + ", " + str(H) + ", 0, " +
        str(ls) + "};\n")
```

```
f.write("Line(1) = {1, 2};\n")
f.write("Line(2) = {1, 3};\n")
f.write("Line(3) = {2, 4};\n")
f.write("Line(4) = {3, 5};\n")
f.write("Line(5) = {4, 6};\n")
f.write("Line(6) = {7, 9};\n")
f.write("Line(7) = {8, 10};\n")
f.write("Line(8) = {5, 9};\n")
f.write("Line(9) = {6, 10};\n")
f.write("Line(10) = {3, 7};\n")
f.write("Line(11) = {4, 8};\n")

f.write("Curve Loop(1) = {10,6,8,4};\n")
f.write("Curve Loop(2) = {11,5,9,7};\n")

f.write("Plane Surface(1) = {1};\n")
f.write("Plane Surface(2) = {2};\n")

f.write("Transfinite Line{8,10} = " + str(nDivThickness) + ";\n")
f.write("Transfinite Line{4,6} = " + str(nDivHeight) + ";\n")
f.write("Transfinite Surface {1} Left;\n")

f.write("Transfinite Line{9,11} = " + str(nDivThickness) + ";\n")
f.write("Transfinite Line{5,7} = " + str(nDivHeight) + ";\n")
f.write("Transfinite Surface {2} Left;\n")

lineListBase1 = [1, 3, 11]
lineListBase1.append(5*n+2)

lista_auxiliar = list(range(16,5*n+2,5))
lineListBase2 = []
lineListBase2.extend(i for pair in ([j, j + 2] for j in
    lista_auxiliar) for i in pair)
```

```
    lineListBase3 = [10, 2]

    lineListBase = lineListBase1 + lineListBase2 + lineListBase3
    lineStringBase = ", ".join(map(str, lineListBase))

#Loop das aletas
    tAcum = 0
    if n > 2:
        for i in range(1, n-1):
            f.write("Point(" + str(nPoints + 1) + ") = {" + str(W-(tAcum +
                t + s)) + ", " + str(hb) + ", 0, " + str(ls) + "};\n")
            f.write("Point(" + str(nPoints + 2) + ") = {" + str(W-(tAcum +
                2*t + s)) + ", " + str(hb) + ", 0, " + str(ls) + "};\n")
            f.write("Point(" + str(nPoints + 3) + ") = {" + str(W-(tAcum +
                t + s)) + ", " + str(H) + ", 0, " + str(ls) + "};\n")
            f.write("Point(" + str(nPoints + 4) + ") = {" + str(W-(tAcum +
                2*t + s)) + ", " + str(H) + ", 0, " + str(ls) + "};\n")

            if nPoints != 10:
                f.write("Line(" + str(nLines + 2) + ") = {" + str(nPoints
                    + 1) + ", " + str(nPoints - 2) + "};\n")
            f.write("Line(" + str(nLines + 1) + ") = {" + str(nPoints + 1)
                + ", " + str(nPoints + 3) + "};\n")
            f.write("Line(" + str(nLines + 4) + ") = {" + str(nPoints + 2)
                + ", " + str(nPoints + 4) + "};\n")
            f.write("Line(" + str(nLines + 3) + ") = {" + str(nPoints + 3)
                + ", " + str(nPoints + 4) + "};\n")
            f.write("Line(" + str(nLines + 5) + ") = {" + str(nPoints + 1)
                + ", " + str(nPoints + 2) + "};\n")

            lineListFins = [nLines + 1, nLines + 3, nLines + 4, nLines + 5]
            lineStringFins = ", ".join(map(str, lineListFins))
```

```
    f.write("Curve Loop(" + str(nLoops+1) + ") = {" +
        lineStringFins + "};\n")
    f.write("Plane Surface("+ str(nSurfaces+1) +") = {"+
        str(nSurfaces+1) +"};\n")

    lista_auxiliar1 = list(range(4,n+2,1))

    f.write("Transfinite Line{" + str(nLines + 3) + ", " +
        str(nLines + 5) + "} = " + str(nDivThickness) + ";\n")
    f.write("Transfinite Line{" + str(nLines + 1) + ", " +
        str(nLines + 4) + "} = " + str(nDivHeight) + ";\n")
    f.write("Transfinite Surface {"+ str(nTransfiniteSurf+1) +"}
        Left;\n")

    tAcum += t+s

    nPoints += 4
    if nPoints != 10:
        nLines += 5
    else:
        nLines += 4
    nSurfaces += 1
    nLoops += 1
    nVolumes += 1
    nTransfiniteSurf += 1

f.write("Line(" + str(nLines + 1) + ") = {8, 11};\n")
f.write("Line(" + str(nLines + 2) + ") = {7, " + str(4*(n-2)+8) +
    "};\n")

f.write("Curve Loop(" + str(nLoops+1) + ") = {" + lineStringBase +
    "};\n")
f.write("Plane Surface("+ str(nSurfaces+1) +") = {"+
    str(nSurfaces+1) +"};\n")
```

```
#Extrusao da malha
        lista_auxiliar2 = [1,2]
        lista_auxiliar3 = list(range(4,n+3,1))
        lista_auxiliar2.extend(lista_auxiliar3)
        extString = ", ".join(map(str, lista_auxiliar2))

        f.write("Extrude {0,0," + str(L) + "} { Surface{"+ extString +"};
            Layers {" + str(nDivExtrusion) + "};}\n")

#Physical Groups
        volumeList = list(range(1, n+2, 1))
        volumeString = ", ".join(map(str, volumeList))

        finsConvList = [n+4,n+9]
        lista_auxiliar4 = list(range(n+13, n+14+(5*(n-3)), 5))
        finsConvList.extend(i for pair in ([j, j + 2] for j in
            lista_auxiliar4) for i in pair)
        finsConvString = ", ".join(map(str, finsConvList))

        baseConvString = ", ".join(map(str, list(range(29+((n-4)*6),
            29+((n-4)*6) + (n-1), 1))))

        finsTopList = list(range(n+5,n+11,5))
        lista_auxiliar5 = list(range(n+14,n+15+(5*(n-3)),5))
        finsTopList.extend(lista_auxiliar5)
        finsTopString = ", ".join(map(str, finsTopList))

        sinkEndsList1 = [n+2, 33+(7*(n-4))]
        sinkEndsList2 = list(range(1,n+2,1))
        sinkEndsList2.remove(3)
        sinkEndsList3 = list(range(n+7, n+8+(5*(n-1)), 5))
        sinkEndsList = sinkEndsList1 + sinkEndsList2 + sinkEndsList3
        sinkEndsString = ", ".join(map(str, sinkEndsList))
```

```
finsTopEdges_1 = [30+((n-4)*5), 38+((n-4)*5)] +
    list(range(44+((n-4)*5), 53+((n-4)*13), 8))
finsTopEdges_2 = [8] + list(range(9, 5*n, 5))

lista_auxiliar6 = [41+(5*(n-4))] + [27+(5*(n-4)), 35+(5*(n-4))] +
    list(range(49+(5*(n-4)), 50+(13*(n-4)), 8))
finsTopEdges_3 = []
finsTopEdges_3.extend(i for pair in ([j, j + 2] for j in
    lista_auxiliar6) for i in pair)

bottomEdges = [1] + [56+(13*(n-4)), 57+(13*(n-4)), 58+(13*(n-4))]

lista_auxiliar7 = list(range(12, 18+(5*(n-4)), 5))
finsConvectionEdges_1 = [4,5,6,7]
finsConvectionEdges_1.extend(i for pair in ([j, j + 3] for j in
    lista_auxiliar7) for i in pair)

finsConvectionEdges_2 = [28+(5*(n-4)), 31+(5*(n-4))] +
    [36+(5*(n-4))] + [39+(5*(n-4))]
lista_auxiliar8 = list(range(42+(5*(n-4)), 51+(13*(n-4)), 8))
finsConvectionEdges_2.extend(i for pair in ([j, j + 4] for j in
    lista_auxiliar8) for i in pair)

baseConvectionEdges_1 = list(range(60+(13*(n-4)), 63+(14*(n-4)),
    1))
baseConvectionEdges_2 = list(range(18, 24+(5*(n-4)), 5)) +
    [22+(5*(n-4))]
baseConvectionEdges_6 = [33+(5*(n-4)), 25+(5*(n-4))]
lista_auxiliar9 = list(range(40+(5*(n-4)), 49+(13*(n-4)), 8))
baseConvectionEdges_6.extend(i for pair in ([j, j + 5] for j in
    lista_auxiliar9) for i in pair)
```

```
f.write("Physical Surface(" + "\"finsConvection\"" ", 1 ) = {" +
    finsConvString + "};\n")
f.write("Physical Surface(" + "\"baseConvection\"" ", 2 ) = {" +
    baseConvString + "};\n")
f.write("Physical Surface(" + "\"finsTop\"" ", 3 ) = {" +
    finsTopString + "};\n")
f.write("Physical Surface(" + "\"sinkEnds\"" ", 4 ) = { " +
    sinkEndsString + "};\n")
f.write("Physical Surface(" + "\"sinkSides\"" ", 5 ) = { " +
    str(10+(n-4)) + ", " + str(15+(n-4)) +"};\n")
f.write("Physical Surface(" + "\"sinkBottom\"" ", 6 ) = { " +
    str(27+(6*(n-4))) + "};\n")

f.write("Physical Volume(" + "\"copper\"" ", 1 ) = {" +
    volumeString + "};\n")

#Fins Top Edges
edgeNumber = 0
for i in range(len(finsTopEdges_1)):
    edgeNumber += 1
    f.write("Physical Curve(" + "\"edge" + str(edgeNumber) + " \""
        ", "+ str(edgeNumber) +" ) = { "+ str(finsTopEdges_1[i])
        +"};\n")

for i in range(len(finsTopEdges_2)):
    edgeNumber += 1
    f.write("Physical Curve(" + "\"edge" + str(edgeNumber) + " \""
        ", "+ str(edgeNumber) +" ) = { "+ str(finsTopEdges_2[i])
        +"};\n")

for i in range(len(finsTopEdges_3)):
    edgeNumber += 1
    f.write("Physical Curve(" + "\"edge" + str(edgeNumber) + " \""
        ", "+ str(edgeNumber) +" ) = { "+ str(finsTopEdges_3[i])
```

```
        +"};\n")

#Bottom edges
for i in range(len(bottomEdges)):
    edgeNumber += 1
    f.write("Physical Curve(" + "\"edge" + str(edgeNumber) + " \""
        ", "+ str(edgeNumber) +" ) = { "+ str(bottomEdges[i])
        +"};\n")

#Fins Convection
for i in range(len(finsConvectionEdges_1)):
    edgeNumber += 1
    f.write("Physical Curve(" + "\"edge" + str(edgeNumber) + " \""
        ", "+ str(edgeNumber) +" ) = { "+
        str(finsConvectionEdges_1[i]) +"};\n")

for i in range(len(finsConvectionEdges_2)):
    edgeNumber += 1
    f.write("Physical Curve(" + "\"edge" + str(edgeNumber) + " \""
        ", "+ str(edgeNumber) +" ) = { "+
        str(finsConvectionEdges_2[i]) +"};\n")

#Base Convection
for i in range(len(baseConvectionEdges_1)):
    edgeNumber += 1
    f.write("Physical Curve(" + "\"edge" + str(edgeNumber) + " \""
        ", "+ str(edgeNumber) +" ) = { "+
        str(baseConvectionEdges_1[i]) +"};\n")

for i in range(len(baseConvectionEdges_2)):
    edgeNumber += 1
    f.write("Physical Curve(" + "\"edge" + str(edgeNumber) + " \""
        ", "+ str(edgeNumber) +" ) = { "+
        str(baseConvectionEdges_2[i]) +"};\n")
```

```
        for i in range(len(baseConvectionEdges_6)):
            edgeNumber += 1
            f.write("Physical Curve(" + "\"edge" + str(edgeNumber) + " \""
                ", "+ str(edgeNumber) +" ) = { "+
                str(baseConvectionEdges_6[i]) +"};\n")

#PONTO DE TRABALHO ----------------------------------------
minCFM = 1
maxCFM = 100

#Fan curve
file = "Fan Curve.xlsx"
df = pd.read_excel(file)
fan_airflowPoints = df['airflow']
fanPressurePoints = df['pressure']
fan_airflowArray = fan_airflowPoints.to_numpy()
fan_pressureArray = fanPressurePoints.to_numpy()

#System curves
airflowArray = np.zeros(maxCFM*2 - 1)
pressureArray = np.zeros(maxCFM*2 - 1)
resistanceArray = np.zeros(maxCFM*2 - 1)

acum = 1
for i in range(0, maxCFM*2 - 1):
    airflowArray[i] = acum
    pressureArray[i] = pressureDrop(acum)[0]
    acum += 0.5

#Obtencao do ponto de trabalho
cfmRange = np.linspace(0, 100, 250)
pressureFanInterp = np.interp(cfmRange, fan_airflowArray,
    fan_pressureArray)
```

```
pressureSystemInterp = np.interp(cfmRange, airflowArray, pressureArray)
operatingCFM = cfmRange[np.argmin(np.abs(pressureFanInterp -
    pressureSystemInterp))]
operatingPressure = pressureDrop(operatingCFM)[0]
operatingRe = pressureDrop(operatingCFM)[1]
operatingResStar = convectionCoeficcient(operatingCFM)[-1]
operatingPoint = np.array([operatingCFM, operatingPressure], dtype =
    'float')
operatingPoint_convection = convectionCoeficcient(operatingCFM)
h1 = operatingPoint_convection[1]


#Plot
fig, ax1 = plt.subplots(figsize=(8, 5))
ax1.set_xlabel('Vazao Volumetrica [CFM]', fontsize=12)
ax1.set_ylabel('Perda de pressao [Pa]', fontsize=12, color='black')
ax1.tick_params(axis='y', labelcolor='black')
ax1.plot(airflowArray, pressureArray, color='blue', linewidth=2,
    label='Impedancia do Sistema')
ax1.plot(fan_airflowArray, fan_pressureArray, color='blue',
    linestyle='--', linewidth=1, label='Curva do Ventilador
    9CRH0648P6G001 @ 48V')
ax1.scatter(operatingPoint[0], operatingPoint[1], color='orange', s=75,
    zorder=5,
            label=f'Ponto de Trabalho - {round(operatingCFM, 2)} CFM e
                {round(operatingPressure, 2)} Pa')
ax1.set_title(f'Caracterizacao do sistema para {n} aletas', fontsize=16)
ax1.grid(True)
ax1.legend(loc='right', ncol=1, fontsize=7.5, bbox_to_anchor=(0.88, 0.8))
plt.show()

#ARQUIVO DE INPUT -----------------------
heatFlux = 205/baseArea
print('heat flux: ' + str(round(heatFlux, 2)) + ' W/m')
```

```
parameters = {
    'copper': {"alpha": '393', "g": '0'},
    'finsConvection': {"tipo": '3', "valor": str(h1)},
    'baseConvection': {"tipo": '3', "valor": str(h1)},
    'finsTop': {"tipo": '2', "valor": '0'},
    'sinkEnds': {"tipo": '2', "valor": '0'},
    'sinkSides': {"tipo": '2', "valor": '0'},
    'sinkBottom': {"tipo": '2', "valor": str(heatFlux)}
}

inputFile = str(n) + "fins" + str(tmm) + "mm" + str(hbmm) + "hbA.json"

with open(inputFile, 'w') as file:
    json.dump(parameters, file, indent=4)

print(f"parameters have been successfully exported to {inputFile}")

#GERACAO DE MALHA -----------------------
path =
    r"C:\Users\mduar\Downloads\gmsh-4.12.2-Windows64\gmsh-4.12.2-Windows64\gmsh.exe"

geometry = str(n) + "fins" + str(tmm) + "mm" + str(hbmm) + "hbA.geo"
mesh = str(n) + "fins" + str(tmm) + "mm" + str(hbmm) + "hbA.msh"

command = [
    "gmsh",
    geometry,
    "-3", # Para gerar uma malha 3D (-2 para 2D)
    "-format", "msh2", # Formato padrao Gmsh
    "-o", mesh, ".msh" # Nome do arquivo de saida
]
subprocess.run(command, check=True)
print(f"mesh has been successfully generated and saved as {mesh}")
```

A.2 Simulação - solver MEF

A.2.1 Módulo fourierPY

```
from preProcessing import meshReader, elementType, namedSelections,
    meshMapping, ccNodes, bordas
from femMatrices import tet, bVector
from boundaryConditions import dirichlet, neumann, robin
from results import steadySolution, steadyPointData

import json
import os

meshInput = input('Insira o nome da malha: ')
mesh = meshInput + '.msh'
borda = ''

parameters = {}
jsonFile = meshInput + '.json'
if os.path.exists(jsonFile):
    with open(jsonFile, 'r') as file:
        parameters = json.load(file)

_meshReader = meshReader(mesh)
_namedSelections = namedSelections(_meshReader)
_elementType = elementType(_meshReader)

#Parametros da simulacao
regime = 'permanente'
difusividade = []
geracao = []

for i in range(len(_namedSelections[2])):
    alpha = parameters['copper']['alpha']
    g = parameters['copper']['g']
```

```
    difusividade.append(float(alpha))
    geracao.append(float(g))

#Set das condicoes de contorno
_meshMapping = meshMapping(difusividade, geracao, _meshReader,
    _namedSelections)
_ccNodes = ccNodes(_meshReader, _elementType, _meshMapping)

element = _elementType[0]
boundNames = _namedSelections[1]
cc = _ccNodes[0]
ccType = []

for i in range(len(boundNames)):
    cond = parameters[boundNames[i]]['tipo']
    ccType.append(int(cond))

dictDirichlet = dict()
dictNeumann = dict()
dictRobin = dict()

for i in range(len(ccType)):
    if ccType[i] == 1:
        dictDirichlet[str(boundNames[i])] =
            parameters[boundNames[i]]['valor']

    if ccType[i] == 2:
        dictNeumann[str(boundNames[i])] =
            parameters[boundNames[i]]['valor']

    if ccType[i] == 3:
        dictRobin[str(boundNames[i])] =
            [parameters[boundNames[i]]['valor'],
```

```
                                    input('Selecione a temperatura
                                        infinita no contorno ' +
                                        str(boundNames[i]) + ' [C]: ')]

#Matrizes e vetor b
if element == 'tet1':
    tet_KM = tet(_meshReader, _elementType, _meshMapping)
    K = tet_KM[0]
    M = tet_KM[1]

_bVector = bVector(M, _meshReader, _meshMapping)
b = _bVector[0]

#Leitura das bordas
cells = _meshReader[6]
lista_dic_ccAux = []

if len(cells) > 2:
    _bordas = bordas(dictDirichlet, _meshReader, _namedSelections,
        _meshMapping, _ccNodes)
    lista_aux = _bordas[2]

#Imposicao das condicoes de contorno
_dirichlet = []
_neumann = []

if len(dictNeumann.keys()) != 0:
    _neumann = neumann(b, dictNeumann, element, lista_aux, _meshReader,
        _ccNodes, _bordas, borda)
    b = _neumann[0]

if len(dictRobin.keys()) != 0:
    K, b, bcc = robin(K, b, dictNeumann, dictRobin, element, lista_aux,
        _meshReader, _ccNodes, _bordas, _neumann, borda)
```

```
if len(dictDirichlet.keys()) != 0:
    _dirichlet = dirichlet(K, b, dictDirichlet, _meshReader, element,
        _ccNodes, _bordas, borda)
    K = _dirichlet[0]
    b = _dirichlet[1]

#Resultados
npoints = _meshReader[7]

if regime == 'permanente':
    T = steadySolution(K, b)
    steadyPointData(b, _meshReader, _meshMapping, T, meshInput)

#QUANTIDADES SECUNDARIAS
bottomTemp = []
ccName = _ccNodes[1]

for i in range(len(ccName)):
    if ccName[i] == 'sinkBottom':
        bottomTemp.append(T[i])

maxTemp = max(bottomTemp)
meanTemp = sum(bottomTemp) / len(bottomTemp)
```

A.2.2 Módulo preProcessing

```
from collections import Counter

import numpy as np

import meshio
import random

def meshReader(string):
    msh = meshio.read(string)
    cellData = msh.cell_data
    cells = msh.cells
    coordinates = msh.points
    X = coordinates[:,0]
    Y = coordinates[:,1]
    Z = coordinates[:,2]
    IEN = cells[-1].data
    IENbound = cells[-2].data
    IENline = []

    if len(cells) > 2:
        IENline = cells[-3].data

    npoints = len(X)
    ne = IEN.shape[0]
    nebound = IENbound.shape[0]
    nnele = len(IEN[0])

    return msh, X, Y, IEN, IENbound, cellData, cells, npoints, ne,
        nebound, nnele, IENline, Z, coordinates

def elementType(_meshReader):
    msh = _meshReader[0]
    cells = _meshReader[6]
```

```
    Z = []
    if 'line,' in str(cells[-2]) and 'triangle,' in str(cells[-1]):
        element = 'tri1'
    elif 'line3,' in str(cells[-2]) and 'triangle6,' in str(cells[-1]):
        element = 'tri2'
    elif 'line,' in str(cells[-2]) and 'quad,' in str(cells[-1]):
        element = 'quad1'
    elif 'triangle,' in str(cells[-2]) and 'tetra,' in str(cells[-1]):
        element = 'tet1'
        Z = msh.points[:,2]

    return element, Z

def namedSelections(_meshReader):
   msh = _meshReader[0]
   cells = _meshReader[6]

   IENboundTypeElem = list(msh.cell_data['gmsh:physical'][-2])
   IENboundTypeElem_min = min(IENboundTypeElem)
   IENboundTypeElem -= IENboundTypeElem_min

   IENlineTypeEdges = []
   if len(cells) > 2:
       IENlineTypeEdges = list(msh.cell_data['gmsh:physical'][-3])
       IENlineTypeEdges_min = min(IENlineTypeEdges)
       IENlineTypeEdges -= IENlineTypeEdges_min
       edgeNumber = len(list(set(IENlineTypeEdges)))
   else:
       edgeNames = 'No edges detected with named selections'

   IENTypeRegion = list(msh.cell_data['gmsh:physical'][-1])
```

```
    regionNumber = len(list(set(list(msh.cell_data['gmsh:physical'][-1] -
        1))))
    namedSel = list(msh.field_data.keys())

    if len(cells) > 2:
        edgeNames = namedSel[:edgeNumber]
        boundNames = namedSel[edgeNumber:-regionNumber]
    else:
        edgeNames = 'No edges detected with named selections'
        boundNames = namedSel[:-regionNumber]

    regionNames = namedSel[-regionNumber:]

    return namedSel, boundNames, regionNames, IENboundTypeElem,
        IENTypeRegion, IENlineTypeEdges, edgeNames

def meshMapping(difusividade, geracao,_meshReader, _namedSelections):
    IEN = _meshReader[3]
    npoints = _meshReader[7]

    boundNames = _namedSelections[1]
    regionNames = _namedSelections[2]
    IENboundTypeElem = _namedSelections[3]
    IENTypeRegion = _namedSelections[4]
    IENlineTypeEdges = _namedSelections[5]
    edgeNames = _namedSelections[6]

    _IENTypeRegion = list(dict.fromkeys(IENTypeRegion))

    for i in range(len(IENTypeRegion)):
        IENTypeRegion[i] -= min(_IENTypeRegion)

    _IENTypeRegion = list(dict.fromkeys(IENTypeRegion))
```

```
IENlineEdges = []

if len(IENlineTypeEdges) >= 1:
    IENlineEdges = [edgeNames[elem] for elem in IENlineTypeEdges]

IENboundElem = [boundNames[elem] for elem in IENboundTypeElem]
IENRegion = [regionNames[elem] for elem in IENTypeRegion]
IENAlpha = [difusividade[elem] for elem in IENTypeRegion]
IENg = [geracao[elem] for elem in IENTypeRegion]

gNodes = [[] for i in range(npoints)]
for j in geracao:
    lista = []
    for i in range(0, len(IENg)):
        if IENg[i] == j:
            lista.append(IEN[i])
    arr = np.array(lista)
    a = np.unique(arr.reshape(arr.size))
    for k in a:
        gNodes[k] = j

alphaNodes = [[] for i in range(npoints)]
for j in difusividade:
    lista = []
    for i in range(0, len(IENAlpha)):
        if IENAlpha[i] == j:
            lista.append(IEN[i])
    arr = np.array(lista)
    a = np.unique(arr.reshape(arr.size))
    for k in a:
        alphaNodes[k] = j

return IENboundElem, IENRegion, IENAlpha, IENg, lista, gNodes,
    IENlineEdges, alphaNodes
```

```
def ccNodes(_meshReader, _elementType, _meshMapping):
    IENbound = _meshReader[4]
    X = _meshReader[1]
    element = _elementType[0]
    IENboundElem = _meshMapping[0]

    cc = np.unique(IENbound.reshape(IENbound.size))
    ccName = [[] for i in range( len(X) )]

    for elem in range(0, len(IENbound)):
        ccName[ IENbound[elem][0] ] = IENboundElem[elem]
        ccName[ IENbound[elem][1] ] = IENboundElem[elem]
        if element == 'tet1':
            ccName[ IENbound[elem][2] ] = IENboundElem[elem]

    ccNameTuple = []
    for i in range(len(ccName)):
            ccNameTuple.append(str(ccName[i]))
    ccNameTuple = tuple(ccNameTuple)

    return cc, ccName, ccNameTuple

def bordas(dictDirichlet, _meshReader, _namedSelections, _meshMapping,
    _ccNodes):
    IENbound = _meshReader[4]
    IENline = _meshReader[11]
    edgeNames = _namedSelections[6]
    IENlineEdges = _meshMapping[6]
    ccName = _ccNodes[1]

    listaDic = []

    if len(edgeNames) >= 1:
```

```
for i in edgeNames:
    edgeNodesDic = dict()
    edgeLines = np.empty((0,2), int)

    for j in range(len(IENlineEdges)):
        if IENlineEdges[j] == i:
            edgeLines = np.append(edgeLines, [IENline[j]], axis=0)

    edgeNodes =
        np.unique(edgeLines.reshape(edgeLines.size)).tolist()
    edgeNodesDic[i] = edgeNodes #chave: tag da aresta, valores: ns
        das arestas
    listaDic.append(edgeNodesDic)

lista_aux = []
for i in listaDic: #para cada aresta...
    for j in i.keys(): #loop de cada aresta
        #randomList = random.sample(i[j], k=3)
        randomList = random.sample(i[j], k=2)

        listaAux = []
        for k in randomList: #loop de cada n escolhido da aresta
            em questao
            for l in IENbound:
                if k in l:
                    for m in l:
                        if m!= k:
                            listaAux.append(ccName[m])

        cont = Counter(listaAux)

        if len(cont.most_common(2)) == 1:
            possible_ccNames = [cont.most_common(2)[0][0], '0']
        else:
```

```
        possible_ccNames = [cont.most_common(2)[0][0],
            cont.most_common(2)[1][0]]

    if possible_ccNames[0] in dictDirichlet.keys() and
        possible_ccNames[1] in dictDirichlet.keys():
        if dictDirichlet[possible_ccNames[0]] !=
            dictDirichlet[possible_ccNames[1]]:
            edgeBound = input('Selecione um contorno para a ' +
                str(j) + ' dentre as opcoes '
                    + str(possible_ccNames[0]) + ' e ' +
                        str(possible_ccNames[1]) + ': ')
            for l in i[j]:
                if edgeBound != '':
                    ccName[l] = edgeBound
    if possible_ccNames[0] in dictDirichlet.keys() and
        possible_ccNames[1] not in dictDirichlet.keys():
        edgeBound = str(possible_ccNames[0])
        for l in i[j]:
            ccName[l] = edgeBound
    if possible_ccNames[1] in dictDirichlet.keys() and
        possible_ccNames[0] not in dictDirichlet.keys():
        edgeBound = str(possible_ccNames[1])
        for l in i[j]:
            ccName[l] = edgeBound
    elif possible_ccNames[0] not in dictDirichlet.keys() and
        possible_ccNames[1] not in dictDirichlet.keys():
        nonDirichletList = []
        nonDirichletEdge = dict()

        nonDirichletList.append(str(possible_ccNames[0]))
        nonDirichletList.append(str(possible_ccNames[1]))
            #lista com contornos
        nonDirichletEdge[j] = nonDirichletList #dict: arestas e
            contornos nao-dirichlet
```

```
                lista_aux.append(nonDirichletEdge)

    ccNameTuple = []
    for i in range(len(ccName)):
        #if type(ccName[i]) != list:
            ccNameTuple.append(str(ccName[i]))
    ccNameTuple = tuple(ccNameTuple)

return ccName, ccNameTuple, lista_aux, listaDic
```

A.2.3 Módulo femMatrices

```
from scipy.sparse import lil_matrix

import numpy as np

rho = 1.0
cv = 1.0

def tet(_meshReader, _elementType, _meshMapping):
    X = _meshReader[1]
    Y = _meshReader[2]
    Z = _elementType[1]
    IEN = _meshReader[3]
    npoints = _meshReader[7]
    ne = _meshReader[8]
    IENAlpha = _meshMapping[2]

    K = lil_matrix((npoints, npoints), dtype = 'float')
    M = lil_matrix((npoints, npoints), dtype = 'float')

    for e in range(0,ne):
        v1,v2,v3,v4 = IEN[e]

        bi = (Y[v2]-Y[v4])*(Z[v3]-Z[v4]) - (Y[v3]-Y[v4])*(Z[v2]-Z[v4])
        bj = (Y[v3]-Y[v4])*(Z[v1]-Z[v4]) - (Y[v1]-Y[v4])*(Z[v3]-Z[v4])
        bk = (Y[v1]-Y[v4])*(Z[v2]-Z[v4]) - (Y[v2]-Y[v4])*(Z[v1]-Z[v4])
        bl = -1.0*(bi+bj+bk)

        ci = (X[v3]-X[v4])*(Z[v2]-Z[v4]) - (X[v2]-X[v4])*(Z[v3]-Z[v4])
        cj = (X[v1]-X[v4])*(Z[v3]-Z[v4]) - (X[v3]-X[v4])*(Z[v1]-Z[v4])
        ck = (X[v2]-X[v4])*(Z[v1]-Z[v4]) - (X[v1]-X[v4])*(Z[v2]-Z[v4])
        cl = -1.0*(ci+cj+ck)

        di = (X[v2]-X[v4])*(Y[v3]-Y[v4]) - (X[v3]-X[v4])*(Y[v2]-Y[v4])
```

```
dj = (X[v3]-X[v4])*(Y[v1]-Y[v4]) - (X[v1]-X[v4])*(Y[v3]-Y[v4])
dk = (X[v1]-X[v4])*(Y[v2]-Y[v4]) - (X[v2]-X[v4])*(Y[v1]-Y[v4])
dl = -1.0*(di+dj+dk)

volume = abs((1.0/6.0)*np.linalg.det([[1, X[v1], Y[v1], Z[v1]],
                                      [1, X[v2], Y[v2], Z[v2]],
                                      [1, X[v3], Y[v3], Z[v3]],
                                      [1, X[v4], Y[v4], Z[v4]]]))

mele = (volume/20)*np.array([[2,1,1,1],
                             [1,2,1,1],
                             [1,1,2,1],
                             [1,1,1,2]])

kxele = (1.0/(36*volume))*np.array([[bi*bi, bi*bj, bi*bk, bi*bl],
                                    [bj*bi, bj*bj, bj*bk, bj*bl],
                                    [bk*bi, bk*bj, bk*bk, bk*bl],
                                    [bl*bi, bl*bj, bl*bk, bl*bl]])

kyele = (1.0/(36*volume))*np.array([[ci*ci, ci*cj, ci*ck, ci*cl],
                                    [cj*ci, cj*cj, cj*ck, cj*cl],
                                    [ck*ci, ck*cj, ck*ck, ck*cl],
                                    [cl*ci, cl*cj, cl*ck, cl*cl]])

kzele = (1.0/(36*volume))*np.array([[di*di, di*dj, di*dk, di*dl],
                                    [dj*di, dj*dj, dj*dk, dj*dl],
                                    [dk*di, dk*dj, dk*dk, dk*dl],
                                    [dl*di, dl*dj, dl*dk, dl*dl]])

kele = IENAlpha[e]*(kxele+kyele+kzele)

for ilocal in range(0,4):
    iglobal = IEN[e,ilocal]
    for jlocal in range(0,4):
```

```
                jglobal = IEN[e,jlocal]

                K[iglobal,jglobal] += kele[ilocal,jlocal]
                M[iglobal,jglobal] += mele[ilocal,jlocal]

    return K, M

def bVector(M, _meshReader, _meshMapping):
    npoints = _meshReader[7]
    gNodes = _meshMapping[5]

    Q = np.zeros(npoints, dtype = 'float')

    for i in range(0, npoints):
        Q[i] = gNodes[i]/(rho*cv)

    b = M@Q

    return b, Q
```

A.2.4 Módulo boundaryConditions

```
import numpy as np

from numba import jit

def dirichlet(K, b, dictDirichlet, _meshReader, element, _ccNodes,
    _bordas, borda):
    cells = _meshReader[6]
    npoints = _meshReader[7]
    cc = _ccNodes[0]
    ccNameTuple = _ccNodes[2]

    if element == 'tet1' and len(cells) > 2 and borda != '1':
        ccNameTuple = _bordas[1]

    bccDir = np.zeros((npoints), dtype = 'float')
    dirArr = np.zeros((npoints), dtype = 'float')

    for i in range(0, len(ccNameTuple)):
        if str(ccNameTuple[i]) in dictDirichlet.keys():
            if dictDirichlet[ccNameTuple[i]] != '':
                dirArr[i] = dictDirichlet[ccNameTuple[i]]

    for i in cc:
        if str(ccNameTuple[i]) in dictDirichlet.keys():
            K[i,:] = 0.0
            K[i,i] = 1.0
            b[i] = dirArr[i]
            bccDir[i] = dirArr[i]

    return K, b, bccDir

def neumann(b, dictNeumann, element, lista_aux, _meshReader, _ccNodes,
    _bordas, borda):
```

```
ccNameTuple = _ccNodes[2]

if element == 'tet1' and borda != '1':
    listaDic = _bordas[3]

X = _meshReader[1]
Y = _meshReader[2]
Z = _meshReader[12]
IENbound = _meshReader[4]
npoints = _meshReader[7]

if element == 'tet1' and lista_aux != []:
    ccNameTuple = _bordas[1]

bcc = np.zeros((npoints), dtype = 'float')

for i in dictNeumann.keys():
    fluxNodes = []
    if dictNeumann[i] != '0':
        for j in range(len(ccNameTuple)):
            if ccNameTuple[j] == i:
                fluxNodes.append(j)

        listaVal = []
        if element == 'tet1' and lista_aux != []:
            for k in lista_aux:
                if i in list(k.values())[0]:
                    aresta = list(k.keys())[0]

                    for l in listaDic:
                        if aresta in l.keys():
                            listaVal = list(l.values())[0]

                            for m in listaVal:
```

```
                    if m not in fluxNodes:
                        fluxNodes.append(m) #adicao aos
                            fluxNodes dos nos das arestas

if element == 'tet1':
    qeleflux = -1.0*(float(dictNeumann[i]))*np.ones(3)

    @jit(nopython=True)
    def optimize_flux(IENbound, fluxNodes, X, Y, Z, qeleflux,
        b, bcc):
        for e in range(len(IENbound)):
            v1, v2, v3 = IENbound[e]

            if v1 in fluxNodes and v2 in fluxNodes and v3 in
                fluxNodes:
                AB = np.array([X[v2] - X[v1], Y[v2] - Y[v1],
                    Z[v2] - Z[v1]])
                AC = np.array([X[v3] - X[v1], Y[v3] - Y[v1],
                    Z[v3] - Z[v1]])

                cross_product = np.array([AB[1] * AC[2] - AB[2]
                    * AC[1],
                                          AB[2] * AC[0] - AB[0] *
                                              AC[2],
                                          AB[0] * AC[1] - AB[1] *
                                              AC[0]])

                norm_cross = np.sqrt(cross_product[0]**2 +
                    cross_product[1]**2 + cross_product[2]**2)

                area = 0.5 * norm_cross

                mele2d = (area / 12) * np.array([[2, 1, 1],
                                                 [1, 2, 1],
```

```
                                                    [1, 1, 2]])

                        beleflux = np.dot(mele2d, qeleflux)

                        for ilocal in range(3):
                            iglobal = IENbound[e, ilocal]
                            b[iglobal] -= beleflux[ilocal]
                            bcc[iglobal] += -beleflux[ilocal]

                    return b, bcc

                opt = optimize_flux(IENbound, fluxNodes, X, Y, Z,
                    qeleflux, b, bcc)
                b = opt[0]
                bcc = opt[1]

    return b, bcc

def robin(K, b, dictNeumann, dictRobin, element, lista_aux, _meshReader,
    _ccNodes, _bordas, _neumann, borda):
    npoints = _meshReader[7]
    ccNameTuple = _ccNodes[2]

    if element == 'tet1' and borda != '1':
        listaDic = _bordas[3]

    if element == 'tet1' and lista_aux != []:
        ccNameTuple = _bordas[1]

    if len(dictNeumann.keys()) != 0:
        bcc = _neumann[1]
    else:
        bcc = np.zeros((npoints), dtype = 'float')
```

```
IENbound = _meshReader[4]
X = _meshReader[1]
Y = _meshReader[2]
Z = _meshReader[12]

for i in dictRobin.keys():
    convectionNodes = []
    for j in range(len(ccNameTuple)):
        if ccNameTuple[j] == i:
            convectionNodes.append(j)

    listaVal = []
    if element == 'tet1' and lista_aux != []:
        for k in lista_aux:
            if i in list(k.values())[0]:
                aresta = list(k.keys())[0]

                for l in listaDic:
                    if aresta in l.keys():
                        listaVal = list(l.values())[0]

                        for m in listaVal:
                            if m not in convectionNodes:
                                convectionNodes.append(m)

    h = float(dictRobin[i][0])
    tinf = float(dictRobin[i][1])

    if element == 'tet1':
        matrix = np.array([[2,1,1],
                           [1,2,1],
                           [1,1,2]])

        @jit
```

```
def update_b_numba(IENbound, e, mele2d, qeleconv, b, bcc):
    beleconv = mele2d @ qeleconv

    for ilocal in range(3):
        iglobal = IENbound[e, ilocal]
        b[iglobal] += beleconv[ilocal]
        bcc[iglobal] += beleconv[ilocal]

    return b, bcc

def update_K_sparse(IENbound, e, mele2dconv, K):
    for ilocal in range(3):
        iglobal = IENbound[e, ilocal]
        for jlocal in range(3):
            jglobal = IENbound[e, jlocal]
            K[iglobal, jglobal] += mele2dconv[ilocal, jlocal]

    return K

# Funcao principal que itera sobre os elementos
def convection_loop(IENbound, convectionNodes, X, Y, Z, h,
    tinf, matrix, b, bcc, K):
    qeleconv = (h * tinf) * np.ones(3)

    for e in range(len(IENbound)):
        v1, v2, v3 = IENbound[e]

        if v1 in convectionNodes and v2 in convectionNodes and
            v3 in convectionNodes:
            AB = np.array([X[v2] - X[v1], Y[v2] - Y[v1], Z[v2]
                - Z[v1]])
            AC = np.array([X[v3] - X[v1], Y[v3] - Y[v1], Z[v3]
                - Z[v1]])
            area = 0.5 * np.linalg.norm(np.cross(AB, AC))
```

```
                    mele2d = (area / 12) * matrix
                    mele2dconv = h * mele2d

                    b, bcc = update_b_numba(IENbound, e, mele2d,
                        qeleconv, b, bcc)

                    K = update_K_sparse(IENbound, e, mele2dconv, K)

                return b, bcc, K

            b, bcc, K = convection_loop(IENbound, convectionNodes, X, Y,
                Z, h, tinf, matrix, b, bcc, K)
    return K, b, bcc
```

A.2.5 Módulo results

```
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import spsolve

import meshio

def steadySolution(K, b,):
    A = csc_matrix(K)
    T = spsolve(A, b)

    return T

def steadyPointData(b, _meshReader, _meshMapping, T, meshInput):
    msh = _meshReader[0]
    alphaNodes = _meshMapping[7]

    point_data = {'temperatura' : T, 'vetorb' : b, 'alphanodes':
        alphaNodes}

    meshio.write_points_cells(meshInput + '.vtk',
                              msh.points,
                              msh.cells,
                              #file_format="vtk-ascii",
                              point_data=point_data,
                              )
```

Apêndice B

Pontos da curva do ventilador

Tabela B.1: Pontos da curva do ventilador Sanyo Denki San Ace 9CRH0648P6G001.

Vazão volumétrica [CFM]	**Pressão [Pa]**
0	3350
4.536646392254859	3177.2646353484674
8.570965156301568	3136.144786593888
12.605283920348285	3126.219305860024
16.639602684394987	3102.114566934926
20.67392144844169	3073.756050552458
24.708240212488406	3034.9994114964175
28.742558976535108	2953.2323559269666
32.41012148930484	2832.551113988239
36.07768400207459	2711.7123247362765
40.11200276612129	2614.8207270961757
44.146321530167995	2531.163103767894
47.813884042937744	2434.5340849831864
51.114690304430496	2320.732409055762
54.59887469156175	2209.5302637967307
58.449815329969965	2097.750445055834
61.933999717101216	1979.4717996439554
64.50129347604003	1868.1252360144576
66.70183098370187	1758.0784377191558
68.71899036572523	1642.3126719533902
70.55277162	1525.8536980093863
72.38655287849497	1403.1558504612394
74.22033413487982	1275.2589415763064
76.05411539126469	1142.162971354588
77.70451852201107	1020.244983006959
79.17154352711897	911.0646949344555
80.63856853222687	797.9851108593621
82.10559353733477	686.2052921184654
83.57261854244267	570.5261773749799
85.22302167318902	447.56837675999395
87.05680292957392	317.5918433403467
88.70720606032027	171.49821977666306
90.8	0

Apêndice C

Tabela de Resultados

Tabela C.1: Dados de cada dissipador e resultados das suas simulações.

Núm. de aletas	**Espaçamento [mm]**	**Área de troca conv. [m²]**	**Pressão de trab. [Pa]**	**Vazão de trab. [CFM]**	***h* de trab. [W/m²°C]**	η	**Núm. de tetraedros**	**Temp. Máx. no fundo [°C]**	**Temp. Méd. no fundo [°C]**
35	1.25	0.23	373.65	86.35	56.60	0.68	713328	61.24	61.14
36	1.19	0.24	406.10	85.94	57.19	0.68	731478	60.53	60.43
37	1.13	0.25	438.32	85.14	57.68	0.67	756756	59.85	59.70
38	1.07	0.25	477.63	84.74	58.26	0.67	786786	59.15	59.03
39	1.01	0.26	520.88	84.34	58.83	0.67	812724	58.51	58.40
40	0.96	0.27	565.05	83.53	59.29	0.67	831270	57.91	57.82
41	0.91	0.27	617.99	83.13	59.81	0.66	857208	57.42	57.33
42	0.87	0.28	673.19	82.33	60.23	0.66	845988	56.92	56.81
43	0.82	0.29	734.34	81.53	60.60	0.66	861960	56.45	56.34
44	0.78	0.29	802.88	80.72	60.94	0.66	900900	56.03	55.92
45	0.74	0.30	873.81	79.52	61.12	0.66	904068	55.59	55.47
46	0.70	0.31	960.11	78.71	61.35	0.66	921756	55.29	55.20
47	0.66	0.31	1050.67	77.51	61.40	0.66	949212	54.93	54.82
48	0.63	0.32	1145.07	75.90	61.25	0.66	986568	54.71	54.57
49	0.59	0.33	1251.43	74.30	60.98	0.66	985116	54.40	54.29
50	0.56	0.33	1372.65	72.69	60.61	0.66	1020624	54.23	54.13
51	0.53	0.34	1500.49	70.68	59.98	0.66	1026036	54.04	53.93
52	0.50	0.35	1646.34	68.67	59.20	0.67	1050984	53.90	53.80
53	0.47	0.35	1785.33	65.86	57.91	0.67	1072896	53.91	53.78
54	0.44	0.36	1944.36	63.05	56.43	0.68	1091508	53.89	53.79
55	0.42	0.37	2073.04	59.04	54.12	0.69	1115400	54.12	54.01
56	0.39	0.37	2195.92	54.62	51.32	0.71	1138104	54.47	54.32
57	0.37	0.38	2354.00	50.60	48.51	0.72	1155396	54.87	54.75
58	0.34	0.39	2476.02	45.78	44.91	0.74	1174404	55.48	55.36
59	0.32	0.39	2601.26	40.96	41.06	0.76	1190904	56.38	56.23
60	0.30	0.40	2726.02	36.14	36.98	0.78	1203840	57.46	57.32
61	0.28	0.41	2878.23	31.73	33.00	0.80	1216248	58.86	58.74
62	0.25	0.41	2971.78	26.91	28.51	0.83	1233276	60.97	60.85
63	0.23	0.42	3070.98	22.49	24.20	0.85	1248192	63.74	63.60
64	0.21	0.43	3101.65	18.07	19.75	0.88	1263768	67.89	67.75
65	0.20	0.43	3092.85	14.06	15.56	0.91	1283304	73.98	73.85
66	0.18	0.44	3127.99	10.84	12.10	0.93	1303632	82.13	81.96
67	0.16	0.45	3114.57	8.03	9.01	0.94	1319076	94.47	94.33
68	0.14	0.45	3230.93	6.02	6.75	0.96	1345608	110.63	110.47