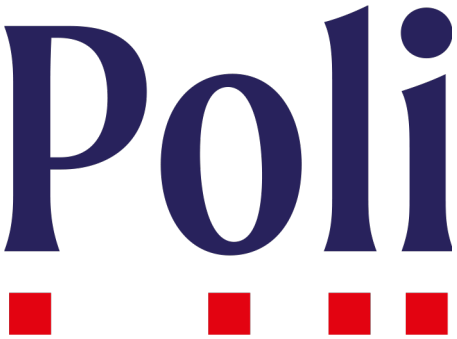


SIMULAÇÃO NUMÉRICA TÉRMICA E DINÂMICA EM PLACAS COM PERFIS INTENSIFICADORAS DE TROCA DE CALOR.

Guilherme de Oliveira Rosa

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

Junho de 2026

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Departamento de Engenharia Mecânica

DEM/POLI/UFRJ

SIMULAÇÃO NUMÉRICA TÉRMICA E DINÂMICA EM PLACAS COM PERFIS INTENSIFICADORAS DE TROCA DE CALOR.

Guilherme de Oliveira Rosa

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, D.Sc.

_____

Prof. Carolina Palma Naveira Cotta, D.Sc.

_____

Prof. Daniel Onofre de Almeida Cruz, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

JUNHO DE 2026

Rosa, Guilherme de Oliveira

Simulação Numérica Térmica E Dinâmica Em Placas com Perfis Intensificadoras De Troca De Calor./ Guilherme de Oliveira Rosa. – Rio de Janeiro: UFRJ/Escola Politécnica, 2026.

XIX, 188 p.: il.; 29, 7cm.

Orientador: Gustavo Rabello dos Anjos

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia Mecânica, 2026.

Referências Bibliográficas: p. 127 – 130.

1. Elementos Finitos. 2. Mecânica dos Fluidos. 3. Transferência de Calor. I. dos Anjos, Gustavo Rabello. II. Universidade Federal do Rio de Janeiro, UFRJ, Curso de Engenharia Mecânica. III. Simulação Numérica Térmica E Dinâmica Em Placas com Perfis Intensificadoras De Troca De Calor..

Aos meus pais e amigos.

Agradecimentos

Agradeço, sobretudo, aos meus pais, por terem contribuído para a formação da pessoa que sou hoje e por terem feito de tudo para incentivar e apoiar minha jornada. Agradeço à minha família pelo carinho e pelo suporte constantes, mesmo à distância. Agradeço também aos meus amigos mais próximos, que acompanharam meu caminho desde a infância ou adolescência e que sempre me incentivaram, acolheram e ofereceram apoio nos momentos de insegurança.

Agradeço a todo o corpo discente da Engenharia Mecânica, que contribuiu para que eu vivenciasse uma excelente formação em engenharia. Agradeço, ainda, ao meu professor orientador e aos demais professores que me ofereceram suporte, orientação e conhecimento ao longo de toda a graduação, possibilitando meu desenvolvimento acadêmico, profissional e pessoal. Agradeço também ao LABMEMS, por ter dado a oportunidade de aprimoramento e desenvolvimento durante a graduação.

Por fim, agradeço à instituição UFRJ pela formação, oportunidades e experiências durante estes anos de graduação.

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

SIMULAÇÃO NUMÉRICA TÉRMICA E DINÂMICA EM PLACAS COM PERFIS INTENSIFICADORAS DE TROCA DE CALOR.

Guilherme de Oliveira Rosa

Junho/2026

Orientador: Gustavo Rabello dos Anjos

Programa: Engenharia Mecânica

Neste projeto, é apresentado a metodologia e o desenvolvimento de um algoritmo de simulação computacional para analisar o efeito de diferentes perfis geométricos no desenvolvimento térmico e dinâmico de um escomaento bidimensional entre placas. Para isso, foi utilizado a formulação corrente-vorticidade para a solução cinemática de mecânica dos flúidos e a equação geral do calor para a equação térmica. Por meio da linguagem de programação Python, se determina a solução numérica por meio do método de elementos finitos.

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

NUMERICAL THERMAL AND KINEMATIC SIMULATION IN PLATES WITH HEAT EXCHANGE ENHANCING PROFILES.

Guilherme de Oliveira Rosa

June/2026

Advisor: Gustavo Rabello dos Anjos

Department: Mechanical Engineering

This project presents the methodology and development of a computational simulation algorithm to analyze the effect of different geometric profiles on the thermal and kinematic development of a two-dimensional flow between plates. For this purpose, the current-vorticity formulation was used for the kinematic solution of fluid mechanics and the general heat equation for the thermal equation. Using the Python programming language, the numerical solution is determined through the finite element method.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

CFD	Computational Fluid Dynamics (Dinâmica de Fluido Computacional), p. 37
COPPE	Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia, p. 16
EDO	Equação Diferencial Ordinária, p. 16
EDP	Equação Diferencial Parcial, p. 16
LABMFA	Laboratório de Mecânica dos Fluidos e Aerodinâmica, p. 16
MDF	Método de Diferenças Finitas, p. 16
MEF	Método de Elementos Finitos, p. 16
MVF	Método de Volumes Finitos, p. 16

Capítulo 1

Introdução

Projetos de trocadores de calor esbarram em um dilema fundamental da engenharia, sendo este o perfil de troca entre fluidos. Nestes projetos, os fenômenos de transferência de quantidade de movimento e energia estão acoplados, o que significa que qualquer alteração na velocidade do escoamento impacta diretamente o perfil de temperaturas e, por consequência, a eficácia da troca térmica. Isso gera uma dificuldade de parametrizar o funcionamento da troca de calor que ocorre nas interfaces dos fluido s de trabalho. Para fins de dimensionamento de um trocador, utiliza-se o parâmetro de coeficiente de transferência de calor médio por convecção conforme os métodos apresentados por Incropera [1] e Shah [2]. Todavia, determinar este parâmetro exige, portanto, que se compreenda a fundo o comportamento fluidodinâmico do sistema.

A principal dificuldade reside na natureza matemática do problema. Segundo White [3], as equações que regem o movimento dos fluidos, denominadas equações de Navier-Stokes, são notoriamente não lineares e de solução analítica indeterminada para a maioria das configurações de interesse prático. Essa complexidade é agravada pela possibilidade de transição entre os regimes laminar, transicional e turbulento. Na prática, pequenas variações no perfil da parede, obstáculos ao longo do escoamento ou alterações na temperatura ao longo do escoamento podem desencadear comportamentos completamente distintos no escoamento, inviabilizando abordagens puramente teóricas.

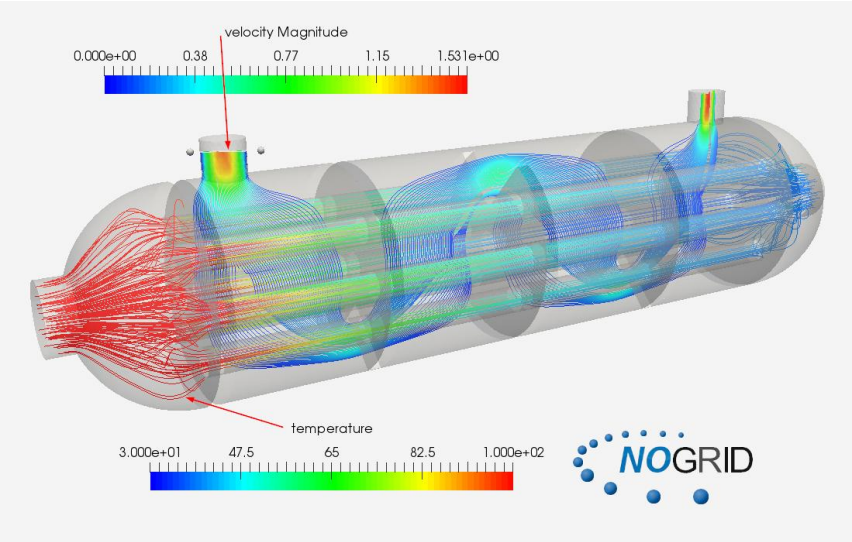


Figura 1.1: Exemplo de escoamento fluidodinâmico complexo em uma configuração típica de troca de calor.

Fonte: Adaptado de NOGRID (2025).

Devido à esta barreira matemática, a engenharia química e mecânica construíram, ao longo do século XX, uma série de correlações baseado em conhecimento empírico. Em vez de resolver as equações diferenciais a cada projeto, os engenheiros passaram a recorrer a correlações consolidadas como é apresentado por Shah [2] e Kakaç [4]. Destacam-se as equações de Dittus-Boelter, Sieder-Tate e Gnielinski para estimar coeficientes de transmissão de calor médios e fatores de atrito em situações clássicas.

Nas últimas décadas, no entanto, a busca por equipamentos mais compactos e energeticamente eficientes impôs novos desafios que as correlações tradicionais, por vezes, não conseguem abarcar com exatidão. A necessidade de otimização levou ao desenvolvimento de metodologias mais sofisticadas, capazes de manipular o escoamento em escala local. Nesse contexto, uma das técnicas implementadas é a intensificação de transferência de calor, conceito formalizado por Bergles [5] e apresentado por Webb [6]. Essas técnicas visam romper a camada limite térmica ou aumentar a mistura do fluido junto à parede de troca, elevando o coeficiente convectivo. O ganho térmico, porém, não é gratuito: ele vem acompanhado de um aumento na perda de carga, criando um trade-off clássico que precisa ser cuidadosamente avaliado no projeto do equipamento.

1.1 Motivação

Dentre as formas de técnicas de intensificação de transferência de calor, subdivide-se em técnicas ativas ou passivas.

Técnicas ativas envolvem o uso de trabalho ou energia externa para estimular a troca térmica, seja por meio de interações mecânicas como vibrações e deslocamentos, forças oriundas de campos elétromagnéticos ou pulsações ao longo do trocador. As técnicas passivas geralmente incluem alterações geométricas no perfil dos tubos de modo que a troca de calor seja intensificada, como o uso de aletas, rugosidades superficiais controladas ou espaçadores.

As técnicas passivas de intensificação de calor são extremamente presentes nas aplicações gerais de trocadores de calor. Entretanto, é difícil avaliar quantitativamente a influência das diferentes geometrias na intensificação da troca de calor. Isso decorre do comportamento diferencial não linear que rege os fenômenos de transporte de um trocador de calor. Desta forma, a forma de avaliar o efeito de uma superfície seria ou por meio de testes empíricos ou através de simulações numéricas.

Isso leva à motivação do projeto, visto que como podemos simular os efeitos térmicos e dinâmicos em um escoamento entre placas dado alterações geométricas das paredes em diferentes configurações por meio de métodos numéricos.

1.2 Objetivo

O objetivo deste projeto é analisar, portanto, através de simulações numéricas, os efeitos térmicos e dinâmicos da intensificação da transferência de calor promovida por técnicas passivas aplicadas a diferentes perfis geométricos de superfícies em um escoamento interno em regime laminar. Permitindo assim visualizar a influência das variações geométricas — como rugosidades controladas e modificações periódicas de perfil em formatos específicos — sobre os fenômenos de transporte térmico e fluidodinâmico, comparando o desempenho dessas configurações sem a necessidade exclusiva de ensaios experimentais.

Capítulo 2

Revisão Bibliográfica

2.1 Método Numéricos

Para descrever o comportamento matemático de fenômenos físicos normalmente se utiliza de equações ou correlações. Entretanto, dependendo da relação do fenômeno com seus parâmetros, utiliza-se de equações diferenciais para descrever o comportamento físico.

Estas equações diferenciais podem ser de diferentes tipos, sendo os mais comuns equações diferenciais ordinárias (EDOs) e equações diferenciais parciais (EDPs). Estes tipos de equações se diferenciam principalmente devido ao número de icógnitas presentes para descrever o comportamento, em que, enquanto equações diferenciais ordinárias utilizam derivadas em torno de um parâmetro, equações diferenciais parciais envolvem derivadas em diferentes parâmetros.

EDOs podem descritas para uma ordem ordem n como:

$$F\big(x, y, y', y'', \ldots, y^{(n)}\big) = 0 \tag{2.1}$$

Enquanto que, para uma EDP de 2 variáveis de ordem n, a equação pode ser denotada como:

$$F\left(x, y, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial x \partial y}, \frac{\partial^2 u}{\partial y^2}, \ldots, \frac{\partial^n u}{\partial y^n}\right) = 0 \tag{2.2}$$

Dependendo do sistema de equações diferenciais parciais, há a possibilidade de não haver resultado analítico. Desta forma, foram criados métodos para resolver de modo aproximado estas equações. Dentre os métodos mais comuns, destaca-se

o método de diferenças finitas (MDF), o método de elementos finitos (MEF) e o método de volumes finitos (MVF).

Para este projeto, será utilizado o método de elementos finitos em conjunto com o método de diferenaçs finitas para descrever o comportamento fluidodinâmico e térmico do escoamento.

2.1.1 Método de Diferenças Finitas

O Método de Diferenças Finitas ou MDF é a forma mais simples de aplicar uma solução numérica a um problema físico ou matemático que não possui solução analítica. Neste método, utiliza-se da expansão de Taylor:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \mathcal{O}\big((x-a)^{n+1}\big) \tag{2.3}$$

É possível aproximar equações em torno de um valor a qualquer por meio desta expansão, de modo a obter um erro de ordem controlável tal que o resultado obtido seja próximo o suficiente de resultados analíticos ou empíricos. Dessa forma, é possível reescrever derivadas de primeira ordem ou segunda ordem de uma função f como:

$$\frac{\partial f}{\partial x} = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2) \tag{2.4}$$

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2) \tag{2.5}$$

Sendo o valor $\mathcal{O}$ o erro numérico associado ao truncamento da expansão de taylor da variável.

Este método embora seja interessante de se aplicar em problemas simples, como é apresentado pela literatura em Versteeg [7] e Zienkiewicz [8], este método possui diversas adversidades em problemas que envolve uma difusão numérica considerável como são os problemas não lineares que envolvem métodos de transporte. Dessa forma, para esses tipos de problemas, é comum a utilização de métodos numéricos mais robustos como volumes finitos e elementos finitos.

Todavia, é comum a utilização do método de diferenças finitas para calcular as variações temporais de variáveis como velocidade, pressão e temperatura em problemas de mecânica dos fluido s. Como é apresentado por Cunha [9], Sousa [10], Carrasco [11] e Gomes [12], em problemas de mecânica dos fluido s computacional bidimensionais, é possível utilizar o método de elementos finitos em conjunto com diferenças finitas, de modo que, para obter os resultados espacialmente é utilizado o método de elementos finitos, enquanto as atualizações das variáveis ao longo do tempo é obtido por meio do método de diferenças finitas.

2.1.2 Método de Elementos Finitos

O Método de Elementos Finitos surgiu como uma forma de solução numérica de problemas complexos em domínios e geometrias não regulares, sendo esta uma configuração que impossibilita o método de diferenças finitas. Sua origem surgiu inicialmente na área de mecânica sólida e civil entre as décadas de 40 e 50. No contexto da mecânica dos fluidos computacional, este método foi incorporado como uma das principais técnicas de discretização espacial. Segundo Webb e Kim [6], a utilização do método de elementos finitos para a solução das equações de Navier-Stokes marcou uma transição de abordagens empíricas para um conjunto acoplado de abordagem empírica com simulações computacionais.

A metodologia de elementos finitos consiste em dividir o domínio em um conjunto de nós que são posteriormente interligados por meio de elementos. Conforme descrito por Zienkiewicz [8], estes elementos podem ter qualquer geometria desde que seja obedecido alguns critérios, como a compatibilidade entre os elementos, a continuidade das variáveis nas interfaces e uma representação geométrica do domínio físico aadequada. A partir dessa discretização, é necessário uma tratativa da própria equação diferencial para então aplicar o método.

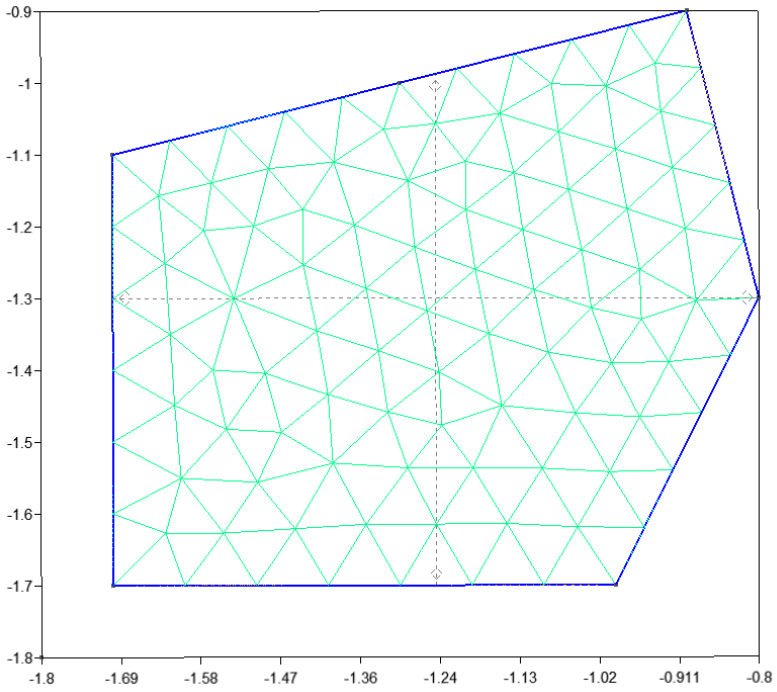


Figura 2.1: Geometria qualquer discretizada espacialmente.

Fonte: Autor (2026).

Segundo Zienkiewicz [8], para chegar à formulação discreta final de um problema de mecânica dos fluido s utilizando o Método de Elementos Finitos, é necessário determinar a forma fraca do problema. Para isto, multiplica-se o problema original por uma função auxiliar arbitrária. Em seguida, realiza-se a integração da equação resultante sobre todo o domínio do problema. Esse processo transforma a equação diferencial em uma equação integral.

Por exemplo, para um problema descrito por uma função f qualquer, temos:

$$\begin{cases} \nabla f + \nabla^2 f + f = S \quad \text{em } \Omega \\ f(\mathbf{x}) = g(\mathbf{x}) \quad \text{em } \Gamma_D \\ \nu \dfrac{\partial f}{\partial n} = h(\mathbf{x}) \quad \text{em } \Gamma_N \end{cases} \tag{2.6}$$

Para então chegar a uma forma fraca do tipo:

$$\int_\Omega w \left(\nabla f + \nabla^2 f + f - S \right) \, d\Omega = 0 \tag{2.7}$$

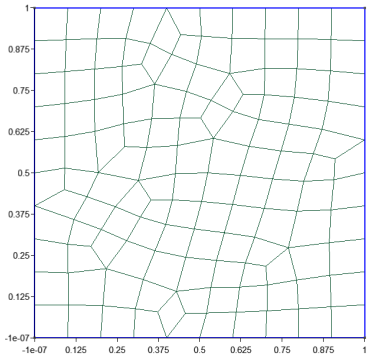
Deste novo problema integrativo, por meio de uma integração por partes, derivadas de ordem mais alta da forma forte são transferidas parcialmente para a função teste. Isso implica que a solução procurada não precisa mais possuir derivadas de

alta ordem contínuas, sendo suficiente que pertença a um espaço funcional mais amplo, como os Espaços de Sobolev, que admitem funções menos regulares.

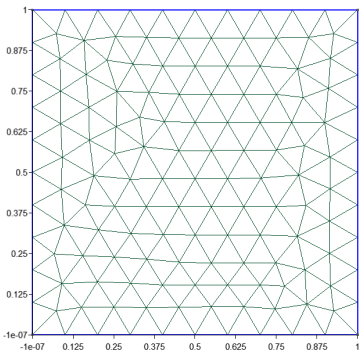
Uma vez estabelecida a forma fraca, torna-se possível aplicar o Método dos Elementos Finitos por meio de uma transição do sistema contínuo para um sistema discreto, utilizando da discretização formulada previamente pretende-se obter algo da forma:

$$K_{ij} f_i + M_{ij} f_i + G_{ij} f_i = g_j + c.c. \tag{2.8}$$

Normalmente, o método de elementos finitos está mais associado à aplicações sólidas, que, por meio de simuladores comerciais, é possível verificar propriedades como deflexões, tensões e comportamentes dinâmicos. Todavia, este modelo em elementos finitos também pode ser aplicado para para códigos de dinâmica de fluido computacional. Conforme apresentado por Miranda [13], Cunha [9] e Sousa [10], por este método ser robusto matematicamente, ele é uma metodologia válida para avaliar comportamento hidrodinâmicos. Excetuando o método de elementos finito, também destaca-se o método de volumes finitos como é apresentado por Versteeg [7], que também é de ampla aplicação para simulações hidrodinâmicas. O que torna este método interessante é a possibilidade de trabalhar com diferentes geometrias de elementos na composição do problema geométrica, como por exemplo é apresentado por Khosvagg [14], que utiliza de elementos de forma tetraédrica e hexaédrica ao longo da placa solar.



(a) Elementos Retangulares



(b) Elementos Triangulares

Figura 2.2: Diferentes discretizações em elementos finitos em um domínio retangular.

Fonte: Autor (2026).

Como se trata de uma aproximação, o Método de Elementos Finitos sempre retorna um resultado que possui erros que devem ser controlados para que a simulação possa ser considerada aplicável. Para isto, antes da simulação ser feita, é comum uma análise do perfil geométrico dos elementos. Por exemplo, em Khosvagg [14], Zheng [15] e Vila [16], para preparar o domínio geométrico do problema, foi necessário um refino dos elementos em situações de transição geométrica. Ou seja, para casos rugosos, a alteração de um perfil suave para um perfil corrugado demanda um número maior de elementos para que assim a simulação consiga reduzir erro de comportamentos não lineares.

Por exemplo, em Khosvagg [14], a região de transição da parte suave para a parte rugosa utiliza de elementos tetraédricos muito menores do que o restante da placa, que utiliza elementos hexaédricos regulares igualmente espaçados, devido ao caráter menos complexo da geometria do problema.

2.2 Transimssão de calor por Convecção

A convecção térmica constitui um dos três mecanismos fundamentais de transferência de calor, ao lado da condução e da radiação. Diferentemente da condução, que ocorre em meios estacionários, a convecção requer o movimento macroscópico de um fluido (líquido ou gás) em contato com uma superfície em diferentes temperaturas. Conforme apresentado por Cengel [17], a transferência de calor por convecção é expressa pela Lei do Resfriamento de Newton:

$$\dot{Q} = hA(T_s - T_\infty) \tag{2.9}$$

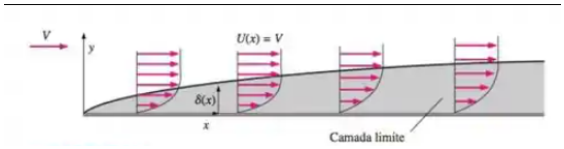
onde $\dot{Q}$ é a taxa de transferência de calor, A é a área superficial, T_s é a temperatura da superfície, T_∞ é a temperatura do fluido longe da superfície, e h é o coeficiente de transferência de calor por convecção, uma grandeza que encapsula a complexidade do fenômeno.

A convecção é classicamente classificada em duas categorias principais: **convecção forçada** e **convecção natural** (ou livre). Na convecção forçada, o movimento do fluido é induzido por meios externos, como bombas, ventiladores ou sopradores. Já na convecção natural, o escoamento surge devido a forças de empuxo

decorrentes de diferenças de densidade causadas pelos próprios gradientes térmicos no fluido. Cengel destacam ainda a existência da **convecção mista**, onde ambos os mecanismos atuam simultaneamente.

Esta forma de descrição do comportamento convectivo pela Lei do Resfriamento de Newton é uma simplificação de um comportamento local de transmissão de energia por convecção ligada ao conceito de camada limite, originalmente desenvolvido por Ludwig Prandtl em 1904. Neste modelo, quando um fluido em escoamento entra em contato com uma superfície sólida, duas camadas limite se desenvolvem simultaneamente: a camada limite hidrodinâmica (ou de velocidade) e a camada limite térmica.

A definição de camada limite térmica é a região do escoamento adjacente à superfície de contato onde os gradientes de temperatura são significativos. Nessa região, a temperatura do fluido varia desde a temperatura da superfície T_s até a temperatura da corrente livre T_∞, que é alcançada no limite externo da camada limite.



(a) Camada Limite Hidrodinâmia

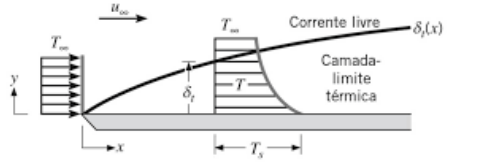


FIGURA 6.2 Desenvolvimento da camada-limite térmica sobre uma placa plana isotérmica.

(b) Camada Limite Térmica

Figura 2.3: Representação Imagética das Diferentes Camadas Limites

Fonte: Incropera (2007).

Formalmente, a espessura da camada limite térmica δ_t é definida como a distância da superfície na qual a diferença de temperatura atinge 99% da diferença total:

$$\frac{T(y) - T_s}{T_\infty - T_s} = 0,99 \tag{2.10}$$

Conforme apresentado por Cengel [17], o desenvolvimento das camadas limite hidrodinâmica e térmica ocorre de forma acoplada, mas não necessariamente simultânea. A razão entre suas espessuras é governada pelo **número de Prandtl**, um parâmetro adimensional definido como:

$$Pr = \frac{\nu}{\alpha} = \frac{\text{difusividade molecular de quantidade de movimento}}{\text{difusividade molecular de calor}} \tag{2.11}$$

Como é apresentado por Shah [2], para aplicações de projeto em trocadores de calor, o parâmetro principal para descrever efetivamente um processo de convecção é o coeficiente de transferência de calor por convecção global, que depende de parâmetros físicos e geométricos do escoamento, já que ele engloba também os efeitos hidrodinâmicos do processo de troca de calor.

Entretanto,um fator considerável nestes projetos é a possível ateração de comportamentos hidrodinâmicos e térmicos gerados pela alteração das propriedades do fluido ao longo de um escoamento, visto que o compoprtamento convectivo depende de propriedades como a difusividade térmica e cinemática, que são parâmetros que alteram com a temeperatura de um fluido . Dessa forma, para casos em que não há mudança de fase e com fluido s que operam com uma baixa variação de suas propriedades em determinada faixa de temperatura, é possível tomar as propriedades a partir da temperatura de filme, temperatura esta definida por:

$$Tfilme = (Tinlet + Twall)/2 \tag{2.12}$$

Tomando estes dados na temperatura de filme, é possível chegar a resultados de projeto satisfatoriamente próximos da realidade de campo.

2.3 Mecânica dos Fluidos

As equações que regem o comportamento de mecânica dos fluidos podem ser compreendidas a partir da aplicação das equações de transporte de Reynolds, que, dado um volume de controle, representa a forma integral das leis de conservação de massa, quantidade de movimento e energia. Tomando o volume de controle de tamanho infinitesimal, pode-se aplicar o teorema da divergência, chegando a formulação diferencial das equações que é apresentada por White [3]. Quando observa-se a equação da quantidade de movimento, no entando, adicionando as condições de operação para um fluido Newtoniano e incompressível, chegamos às equações de Navier-Stokes:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g} \tag{2.13}$$

Onde:

- $\mathbf{u}$: Vetor velocidade do fluido
- ρ: Massa específica do fluido
- ∇p: Gradiente de pressão
- ν: Viscosidade cinemática
- $\mathbf{g}$: Vetor aceleração gravitacional

Uma forma particularmente elucidativa de interpretar as equações de Navier-Stokes é por meio do conceito de derivada material. Segundo Fox [18], a variação temporal da velocidade de uma partícula fluida não é apenas função do tempo, mas também da sua posição no campo de escoamento. Assim, a equação da quantidade de movimento pode ser escrita como a derivada material da velocidade igual ao balanço entre forças de pressão, viscosas e de corpo. O que torna essa análise interessante é que, conforme é apresentado por Taylor [19] e Çengel [17], da mesma forma que os efeitos convectivos da cinemática do escoamento podem ser verificados por meio desse operador, efeitos convectivos de outras propriedades como a energia também podem ser verificados se aplicados à derivada material.

Um problema comum na hora de se resolver as equações de Navier-Stokes advém das variáveis que compõem o sistema. Basicamente, ao mesmo tempo que necessitamos resolver o campo de velocidades, também é necessário determinar o comportamento da pressão e seu gradiente ao longo do domínio para conseguir resolver efetivamente o problema hidrodinâmico. Todavia, dependendo da complexidade do sistema sendo analisado, é possível efetuar simplificações de modo a excluir o termo da pressão para a resolução do campo de veloicidades.

Esta simplificação é feita em situações bidimensionais, denominada de formulação função corrente e vorticidade. Nessa técnica, o campo de velocidades é descrito por meio de uma função corrente, enquanto a vorticidade representa a rotação local do fluido definida a partir da aplicação do operador rotacional nas equações de

Navier-Stokes. Por meio disso, a formulação é escrita em função de uma propriedade rotacional, sendo dessa definida por:

$$\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{2.14}$$

Essa reformulação elimina explicitamente o termo de pressão das equações governantes, reduzindo o número de variáveis primárias e facilitando a solução numérica do problema. Trabalhos recentes, como o de Zheng [15], Patel [20] e Gomes [12], demonstram a eficiência dessa abordagem quando combinada a métodos numéricos avançados, como o método dos elementos virtuais, na solução das equações de Navier-Stokes em duas dimensões.

A partir da aplicação do rotacional , a equação de Navier-Stokes ao invés de solucionar um campo vetorial referente à velocidade e à pressão, a equação a ser resolveida opera em função do parâmetro ω, que é a vorticidade do escoamento. Assim, a equação torna-se:

$$\frac{\partial \omega}{\partial t} + \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} = \nu \left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right) \tag{2.15}$$

Resolvido o campo de vorticidade, resolve-se consequentemente a função corrente ψ por meio da definição:

$$\nabla^2 \psi = -\omega \tag{2.16}$$

E, dado a função corrente, é possível resolver as equações para os campos de velocidade por meio das relações abaixo quando considera-se um fluido incompressível.

$$u = \frac{\partial \psi}{\partial y}, \qquad v = -\frac{\partial \psi}{\partial x} \tag{2.17}$$

2.4 Superfícies Intensificadoras de Transmissão de Calor

A intensificação da transferência de calor constitui uma das principais estratégias para o aumento da eficiência térmica de equipamentos, especialmente em aplicações envolvendo trocadores de calor. Em sistemas convencionais, o comportamento físico

da camada limite térmica e hidrodinâmica define o desempenho térmico do equipamento assim como o seu coeficiente de transferência de calor por convecção. Entretanto, conforme apresentado por Bergles [5], por meio de modificações do escoamento, é possível aumentar a superfície de troca térmica do equipamento, e assim, aumentar o coeficiente de troca de calor por convecção.

De maneira geral, as técnicas de intensificação podem ser classificadas em métodos ativos e passivos. Os métodos ativos requerem a introdução de energia externa no sistema, seja por meio de vibrações mecânicas ou campos eletromagnéticos. Os métodos passivos baseiam-se exclusivamente em modificações geométricas ou na inserção de dispositivos que alteram o escoamento, permitindo a geração de turbulências localizadas que levam a uma maior troca térmica. Os métodos passivos se tornam mais interessantes para aplicações industriais visto que, segundo Bergles [5], métodos passivos apresentam maior simplicidade construtiva, confiabilidade e menor custo energético associado.

Dentre os métodos passivos, a alteração do perfil geométrico das paredes de escoamento é uma forma aplicável em diversas situações operacionais. Essas modificações podem assumir diferentes configurações, como a introdução de rugosidades artificiais controladas, aletas, canais corrugados ou geometrias ondulatórias, perturbando a camada limite e promovendo a mistura do fluido próximo à superfície. Este efeito está diretamente relacionada à modificação do campo de velocidades local. A presença de descontinuidades geométricas induz a formação de recirculações, separações de escoamento e regiões de turbulência, fenômenos que contribuem para o aumento do transporte de quantidade de movimento e energia entre as regiões centrais do escoamento e a parede.

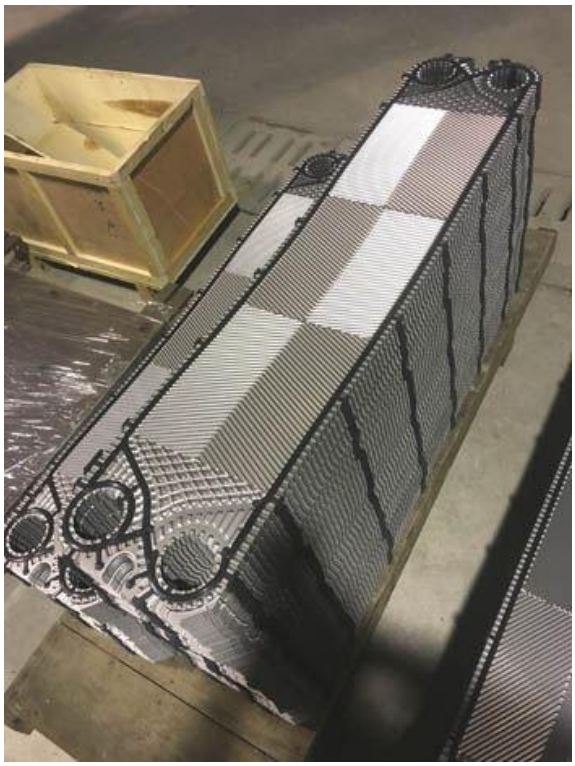


Figura 2.4: Geometria de canal corrugado utilizada para intensificação passiva da transferência de calor.
Fonte: Adaptado de WTSML (2021).

Bergles [5] explicita que esse tipo de intervenção é particularmente eficaz em regimes turbulentos, em que a intensificação da mistura pode resultar em ganhos significativos de desempenho térmico. Entretanto, estas pertubações devem ser analisadas de modo diferente quando o número de Reynolds é baixo, visto que, para o escoamento laminar, o aumento de desempenho térmico advém da geração de movimentos secundários induzidos, instabilidades cinemáticas ou uma antecipação do escoamento para um caso de transição ou turbulento. Para isto, alterações geométricas periódicas ou rugosidades mais intensas são necessárias do que um caso de regime turbulento.

Analisando projetos prévios, Juhui Chen [21] aplica em um trocador de calor de modo conjunto e periódico variações geométricas nas paredes do tubo com espaçadores que agem como obstáculos para o escoamento, obtendo um perfil de velocidades e temperaturas bem diferentes e soluções analíticas comuns para tubos lisos simples, enquanto Anil Singh Yadav [22] faz um estudo de diversas condições geométricas destas variações periódicas para um escoamento em uma placa plana

de um absorvedor. Ademais, Cai-Nin Feng [15] faz uma análise da eficiência do escoamento em uma situação de escoamento cruzado, com variações geométrica periódicas triangulares. Entretanto, projetos como os de Simon Kügele [16] utilizam condições geométricas não controladas, isto é, variações geométricas não periódicas, ou aleatórias, que pode ser uma estratégia em certos casos para aumentar a troca térmica assim como torna-se mais próximo de condições reais experimentais até certa escala.

Capítulo 3

Fundamentação Teórica

Neste capítulo são apresentados as formulações necessárias para a modelagem matemática dos fenômenos de escoamento e transferência de calor. Posteriormente, será descrito a modelagem numérica para determinar a forma fraca das equações apresentadas neste capítulo por meio do Método de Elementos Finitos (MEF).

3.1 Comportamento do Escoamento de Fluidos

Como é apresentado na literatura de Fox [18] e White [3], o comportamento de fluidos pode ser descrito por meio de leis de conservação de algumas propriedades em um volume de controle. Dentre elas, destacam-se:

- Conservação de massa
- Conservação da quantidade de movimento
- Conservação de energia

Por meio da aplicação destas leis, é possível definir as equações que modelam o escoamento, seja por meio de uma abordagem diferencial ou integrativa.

3.2 Equação da Continuidade

A conservação de massa para um fluido é descrita pela equação da continuidade:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{3.1}$$

Considerando um fluido incompressível, em que a densidade é constante para todo o fluido, a equação é reduzida para:

$$\nabla \cdot \vec{v} = 0 \tag{3.2}$$

3.3 Equações de Navier-Stokes

Aplicando a segunda lei de Newton a um volume de controle infinitesimal, é obtida as equações de Navier-Stokes apresentada na equação 2.11. Desta forma, consegue-se avaliar o comportamento do fluido de modo diferencial.

3.3.1 Forma Geral

Como foi apresentado na seção 2.3, uma forma de expressar os comportamentos é por meio do operador derivada material. Assim, aplicando a segunda lei de Newton a um volume de controe infinitesimal, temos que:

$$\rho \frac{D\vec{v}}{Dt} = \sum \vec{F} \tag{3.3}$$

Em que $\frac{D}{Dt}$ representa a derivada material, que é definida por:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \vec{V} \cdot \nabla \tag{3.4}$$

3.3.2 Forças Atuantes

As forças presentes no elemento infinitesimal de fluido podem ser categorizadas em 2 tipos, sendo estes:

- Forças de Superfície (tensões)
- Forças de Corpo (como a gravidade)

As forças de superfícies podem ser expressas pelo termo do tensor de tensões $\boldsymbol{\sigma}$. Em que a força aplicada é descrita por:

$$\vec{F}_{\text{superfície}} = \nabla \cdot \boldsymbol{\sigma} \tag{3.5}$$

Para as forças de corpo, a representação comum é:

$$\vec{F}_{\text{corpo}} = \rho\vec{f} \tag{3.6}$$

3.3.3 Equação de Movimento

Aplicando as forças descritas na seção 3.3.2 na equação 3.3, temos que:

$$\rho\frac{D\vec{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho\vec{f} \tag{3.7}$$

3.3.4 Fluido Newtoniano

Conforme é apresentado na literatura de Fox [18] e White [3], em fluidos Newtonianos, o tensor de tensões é escrito como:

$$\boldsymbol{\sigma} = -p\mathbf{I} + \boldsymbol{\tau} \tag{3.8}$$

Em que p é a pressão e $\boldsymbol{\tau}$ é o tensor de tensões viscosas, equacionado por:

$$\boldsymbol{\tau} = \mu\left[\nabla\vec{v} + (\nabla\vec{v})^T\right] \tag{3.9}$$

3.3.5 Forma Clássica

Aplicando as condições constitutivas evidenciadas na seção 3.3.4, chega-se à formulação clássica de Navier-Stokes:

$$\rho\left(\frac{\partial\vec{v}}{\partial t} + \vec{v}\cdot\nabla\vec{v}\right) = -\nabla p + \mu\nabla^2\vec{v} + \rho\vec{f} \tag{3.10}$$

Em que, considerando fluidos incompressíveis, divide-se ρ em ambos os lados da equação para chegar à forma:

$$\frac{\partial\vec{v}}{\partial t} + \vec{v}\cdot\nabla\vec{v} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{v} + \vec{f} \tag{3.11}$$

Sendo ν a viscosidade cinemática do fluido.

3.4 Equação da Vorticidade

A partir das equações de escoamento em sua versão vetorial, aplica-se o operador $\nabla\times$. Desta forma, a equação de Navier-Stokes passa a resolver o parâmetro da vorticidade ω_z:

$$\frac{\partial \omega}{\partial t} + \vec{v} \cdot \nabla \omega = \nu \nabla^2 \omega + \vec{f} \tag{3.12}$$

Dado que, conforme foi apresentado na equação 2.12, a definição da vorticidade ω_z é:

$$\omega = \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \tag{3.13}$$

3.5 Função Corrente

Em escoamentos bidimensionais e incompressíveis, é possível aplicar um conceito apresentado na literatura de Fox [18] e White [3] de função corrente. Define-se a função corrente ψ de modo que:

$$v_x = \frac{\partial \psi}{\partial y}, \quad v_y = -\frac{\partial \psi}{\partial x} \tag{3.14}$$

$$\nabla^2 \psi = -\omega \tag{3.15}$$

Por meio da utilização desta função, é possível garantir o atendimento da equação da continuidade do escoamento apresentada na seção 3.2.

3.6 Equação da Energia

Como foi citado no capítulo 2.3, a equação da conservação de energia para um fluido também pode ser descrita utilizando termos de derivada material. Todavia, para a propriedade de energia, o parâmetro analisado é a temperatura do escoamento. Assim, define-se a equação de conservação de energia para um fluido como:

$$\rho c_p \left(\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T \right) = k \nabla^2 T + \Phi \tag{3.16}$$

Sendo:

- T a temperatura
- c_p o calor específico
- k a condutividade térmica
- Φ representa a dissipação viscosa

Ademais, admitindo propriedades constantes para o calor específico, condutividade e a imcompressibilidade do fluido, consegue-se reescrever a equação como:

$$\left(\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T\right) = \alpha \nabla^2 T + \Phi \tag{3.17}$$

Em que α é a difusividade térmica do fluido.

Como é evidenciado pela literatura de Taylor [19] e Çengel [17] e projetos como o de Gomes [12], em casos de escoamento laminar, o termo da dissipação viscosa pode ser desprezado na conservação de energia, chegando à equação:

$$\left(\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T\right) = \alpha \nabla^2 T \tag{3.18}$$

3.7 Adimensionalização

Para análise geral dos fenômenos, é comum padronizar o escoamento por meio da utilização de variáveis adimensionais. Um dos principais números adimensionais para descrever os comportamentos dinâmicos e térmicos são o número de Reynolds e o número de Prandtl, sendo estes equacionados por:

$$Re = \frac{\rho U L}{\mu} \tag{3.19}$$

$$Pr = \frac{\nu}{\alpha} \tag{3.20}$$

onde:

- U é uma velocidade característica
- L é um comprimento característico
- ρ é a massa específica

- μ é a viscosidade dinâmica
- ν é a viscosidade cinemática
- α é a difusividade térmica

O número de Reynolds indica a razão entre forças inerciais e viscosas no escoamento. Enquanto o número de Prandtl indica a relação entre os efeitos de difusão dinâmico e térmico em um escoamento.

3.8 Hipóteses do Modelo

Para a modelagem do escoamento e transferência de calor, são adotadas as seguintes hipóteses:

- Escoamento laminar
- Escoamento bidimensional
- Fluido incompressível
- Propriedades físicas constantes
- Ausência de geração interna de calor
- Ausência de forças externas ao escoamento

Por meio destas hipóteses, as equações governantes podem ser simplificadas, permitindo a obtenção de um resultado numérico de interesse com modelos mais simples.

3.9 Condições de Contorno

Para conseguir solucionar qualquer equação diferencial parcial, é necessário adotar e inferir as condições de contorno que representam fisicamente o comportamento adequado do modelo. Nesta seção, será apresentado quais as condições de contorno foram aplicadas para a modelagem computacional do problema.

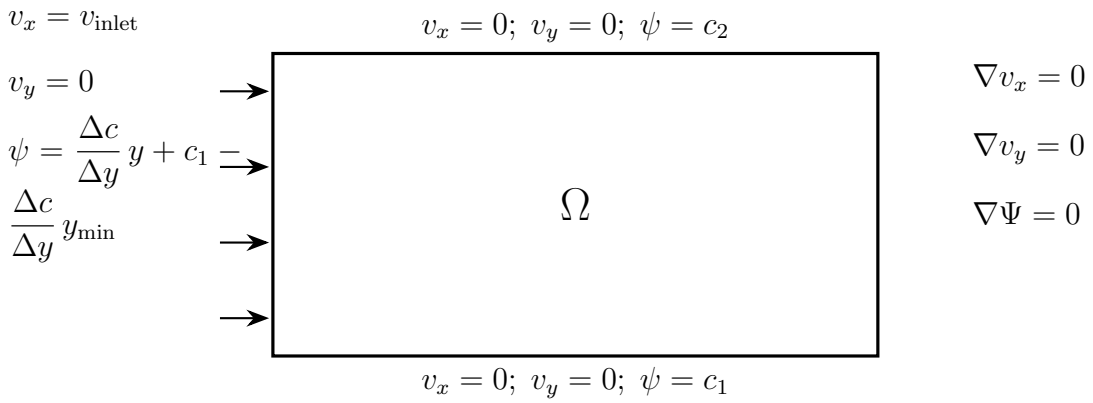


Figura 3.1: Domínio matemático e condições de contorno.

A modelagem matemática do problema pode ser representada pela seguinte imagem:

Nela, é visualizado não somente o domínio interno de cada problema representado pela letra Ω, como também é observado as condições de contorno para as paredes superiores e inferiores assim como para as superfícies de entrada e saída do fluido.

3.9.1 Entrada do Escoamento

Na entrada do escoamento, será adotado um perfil de velocidade uniforme na direção x:

$$v_x = v_{inlet}, \quad v_y = 0 \tag{3.21}$$

Devido ao perfil constante de velocidade da temperatura, também é definido a condição de contorno da função corrente ψ como:

$$\psi = \frac{\Delta c}{\Delta y} y + c_1 - \frac{\Delta c}{\Delta y} y_{\min} \tag{3.22}$$

em que os valores c_1 assim como Δc são as linhas de corrente do domínio. Para este problema, será utilizado:

$$c_1 = 0; \Delta c = \int_{y_{min}}^{y_{max}} v_x \cdot dy \tag{3.23}$$

Para a temperatura, será também adotado uma Temperatura prescrita para o fluido:

$$T = T_{\text{inlet}} \tag{3.24}$$

3.9.2 Saída

Na saída, será adotado gradientes nulos para as todas as funções do modelo:

$$\frac{\partial \phi}{\partial x} = 0 \tag{3.25}$$

Sendo ϕ a variável de interesse.

3.9.3 Paredes

Para as paredes, a condição de contorno aplicada é a condição de não deslizamento:

$$v_x = 0, \quad v_y = 0 \tag{3.26}$$

Para a função corrente, também será aplicada uma condição de contorno do tipo de valor preescrito, sendo este:

$$\psi = \begin{cases} c_1, & y = y_{\min}, \\ \\ c_2, & y = y_{\max}. \end{cases} \tag{3.27}$$

Sendo esses valores de y_{min} e y_{max} atrelados ao contorno da parede inferior e superior, podendo ter seu valor alterado ao longo do domínio Ω.

Para a temperatura, a condição adotada é a de Temperatura prescrita conforme abaixo:

- Temperatura constante:

$$T = T_{wall} \tag{3.28}$$

Capítulo 4

Fundamentação Numérica

Neste capítulo é apresentada a formulação matemática numérica utilizada para descrever de modo aproximado a formulação teórica explicitada no capítulo 3. Ademais, neste capítulo será discutido as condições de contorno e simplificações adotadas para o modelo.

4.1 Forma Forte

Para compor a forma forte do modelo matemática, considera-se a formulação matemática apresentada pelas equações 3.11, 3.12, 3.13, 3.14, 3.15 e 3.18. Desta forma, o comportamento diferencial matemático necessário para explicar o comportamento físico analisado é estruturado.

4.2 Obtenção da Forma Fraca

A forma fraca de uma equação é o produto da ponderação da forma forte integrada sobre o domínio de análise. Dessa forma, para obbter a forma fraca de interesse para o problema apresentado, toma-se resíduos R_i nas equações de forma forte da vorticidade ω_z, função corrente e velocidade a partir da função corrente:

$$\frac{\partial \vec{\omega}}{\partial t} + \vec{v} \cdot \nabla \vec{\omega} - \nu \nabla^2 \vec{\omega} = \vec{R_1} \tag{4.1}$$

$$\nabla^2 \psi + \vec{\omega} = \vec{R_2} \tag{4.2}$$

$$\vec{v} - (\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x}) = \vec{R_3} \tag{4.3}$$

$$\left(\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T\right) - \alpha \nabla^2 T = \vec{R_4} \tag{4.4}$$

A partir dessa nova equação, impõe-se o valor médio de cada resíduo como nulo, de forma que:

$$\int_\Omega \vec{R_1}.\vec{\delta} d\Omega = 0 \tag{4.5}$$

$$\int_\Omega \vec{R_2}.\vec{\xi} d\Omega = 0 \tag{4.6}$$

$$\int_\Omega \vec{R_3}.\vec{\phi} d\Omega = 0 \tag{4.7}$$

$$\int_\Omega \vec{R_4}.\vec{\lambda} d\Omega = 0 \tag{4.8}$$

Onde δ, ξ, ϕ e λ são as funções de peso de cada equação, respectivamente. Estas função pesos são arbitrárias e servem para obter as componentes de contribuição de cada nó, gerando assim as matrizes de elementos finitos.

Aplicando as integrais nas equações de governo, obtém-se:

$$\int_\Omega \left(\frac{\partial \vec{\omega}}{\partial t} + \vec{v} \cdot \nabla \vec{\omega} - \nu \nabla^2 \vec{\omega}\right) .\vec{\delta} d\Omega = 0 \tag{4.9}$$

$$\int_\Omega \left(\nabla^2 \psi + \omega\right) .\vec{\xi} d\Omega = 0 \tag{4.10}$$

$$\int_\Omega \left[\vec{v} - (\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x})\right] .\vec{\phi} d\Omega = 0 \tag{4.11}$$

$$\int_\Omega \left[\left(\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T\right) - \alpha \nabla^2 T\right] .\vec{\lambda} d\Omega = 0 \tag{4.12}$$

Organizando as integrais, obtém-se:

$$\int_\Omega \frac{\partial \vec{\omega}}{\partial t} \cdot \vec{\delta} \Omega + \int_\Omega \vec{v} \cdot \nabla \vec{\omega} \cdot \vec{\delta} \Omega - \int_\Omega \nu \nabla^2 \vec{\omega} \cdot \vec{\delta} \Omega = 0 \tag{4.13}$$

$$\int_\Omega \nabla^2 \psi \, \vec{\xi} \Omega + \int_\Omega \omega \, \vec{\xi} \Omega = 0 \tag{4.14}$$

$$\int_\Omega \vec{v} \cdot \vec{\phi}\, d\Omega - \int_\Omega \left(\frac{\partial \psi_f}{\partial y}, -\frac{\partial \psi_f}{\partial x} \right) \cdot \vec{\phi}\, d\Omega = 0 \tag{4.15}$$

$$\int_\Omega \frac{\partial T}{\partial t} \cdot \vec{\lambda}\, d\Omega + \int_\Omega \vec{v} \cdot \nabla T \cdot \vec{\lambda}\, d\Omega - \int_\Omega \alpha \nabla^2 T \cdot \vec{\lambda}\, d\Omega = 0 \tag{4.16}$$

Utilizado o Teorema de Green nos termos difusivos das equações, temos:

$$-\int_\Omega \nu \nabla^2 \vec{\omega} \cdot \vec{\delta} d\Omega = \int_\Omega \nu \vec{\nabla} \vec{\omega} \cdot \vec{\nabla} \vec{\delta} d\Omega - \int_\Gamma \nu \vec{\delta} \cdot \vec{\nabla} \vec{\omega} \cdot \vec{n}\, d\Gamma \tag{4.17}$$

$$\int_\Omega \nabla^2 \psi \cdot \vec{\xi} d\Omega = -\int_\Omega \vec{\nabla} \psi \cdot \vec{\nabla} \vec{\xi} d\Omega + \int_\Gamma \vec{\xi} \cdot \vec{\nabla} \psi \cdot \vec{n}\, d\Gamma \tag{4.18}$$

$$-\int_\Omega \nabla^2 T \cdot \vec{\lambda} d\Omega = \int_\Omega \vec{\nabla} T \cdot \vec{\nabla} \vec{\lambda} d\Omega - \int_\Gamma \vec{\lambda} \cdot \vec{\nabla} T \cdot \vec{n}\, d\Gamma \tag{4.19}$$

Sendo $\vec{n}$ um vetor normal unitário, orientado para o exterior do contorno Γ. Como as condições de contorno explicitadas na seção 3.9 são do tipo de condição de contorno de Dirichlet, pode-se definir que no contorno Γ, $\delta = \phi = \xi = \lambda = 0$. Desta forma, as equações a serem utilizadas são:

$$\int_\Omega \frac{\partial \vec{\omega}}{\partial t} \cdot \vec{\delta} \Omega + \int_\Omega \vec{v} \cdot \nabla \vec{\omega} \cdot \vec{\delta} \Omega + \int_\Omega \nu \vec{\nabla} \vec{\omega} \cdot \vec{\nabla} \vec{\delta} d\Omega = 0 \tag{4.20}$$

$$-\int_\Omega \vec{\nabla} \psi \cdot \vec{\nabla} \vec{\xi} d\Omega + \int_\Omega \omega\, \vec{\xi} \Omega = 0 \tag{4.21}$$

$$\int_\Omega \vec{v} \cdot \vec{\phi}\, d\Omega - \int_\Omega \left(\frac{\partial \psi_f}{\partial y}, -\frac{\partial \psi_f}{\partial x} \right) \cdot \vec{\phi}\, d\Omega = 0 \tag{4.22}$$

$$\int_\Omega \frac{\partial T}{\partial t} \cdot \vec{\lambda}\, d\Omega + \int_\Omega \vec{v} \cdot \nabla T \cdot \vec{\lambda}\, d\Omega + \int_\Omega \alpha \vec{\nabla} T \cdot \vec{\nabla} \vec{\lambda} d\Omega = 0 \tag{4.23}$$

Assumindo que:

$$m_1 \left(\frac{\partial \vec{\omega}}{\partial t}, \vec{\delta} \right) = \int_\Omega \frac{\partial \vec{\omega}}{\partial t} \cdot \vec{\delta}\, d\Omega \tag{4.24}$$

$$g_1(\vec{v}, \vec{\delta}) = \int_\Omega \vec{v} \cdot \nabla \vec{\omega} \cdot \vec{\delta}\, d\Omega \tag{4.25}$$

$$k_1(\vec{\omega}, \vec{\delta}) = \int_\Omega \nabla \vec{\omega} \cdot \nabla \vec{\delta}\, d\Omega \tag{4.26}$$

$$k_2(\psi, \tilde{\xi}) = \int_\Omega \nabla\psi \cdot \nabla\tilde{\xi}\, d\Omega \tag{4.27}$$

$$m_2(\vec{\omega}, \tilde{\xi}) = \int_\Omega \vec{\omega}\, \tilde{\xi}\, d\Omega \tag{4.28}$$

$$m_3(\vec{v}, \vec{\phi}) = \int_\Omega \vec{v} \cdot \vec{\phi}\, d\Omega \tag{4.29}$$

$$g_3(\psi, \vec{\phi}) = \int_\Omega \left(\frac{\partial\psi}{\partial y}, -\frac{\partial\psi}{\partial x}\right) \cdot \vec{\phi}\, d\Omega \tag{4.30}$$

$$m_4\left(\frac{\partial T}{\partial t}, \vec{\lambda}\right) = \int_\Omega \frac{\partial T}{\partial t} \cdot \vec{\lambda}\, d\Omega \tag{4.31}$$

$$g_4(\vec{v}, \vec{\lambda}) = \int_\Omega \vec{v} \cdot \nabla T \cdot \vec{\lambda}\, d\Omega \tag{4.32}$$

$$k_3(T, \vec{\lambda}) = \int_\Omega \nabla T \cdot \nabla\vec{\lambda}\, d\Omega \tag{4.33}$$

Por fim, obtém-se a forma fraca das equações:

$$m_1\left(\frac{\partial\vec{\omega}}{\partial t}, \vec{\delta}\right) + g_1(\vec{v}, \vec{\delta}) + \nu\, k_1(\vec{\omega}, \vec{\delta}) = 0 \tag{4.34}$$

$$-k_2(\psi, \tilde{\xi}) + m_2(\vec{\omega}, \tilde{\xi}) = 0 \tag{4.35}$$

$$m_3(\vec{v}, \vec{\phi}) - g_3(\psi, \vec{\phi}) = 0 \tag{4.36}$$

$$m_4\left(\frac{\partial T}{\partial t}, \vec{\lambda}\right) + g_4(\vec{v}, \vec{\lambda}) + \alpha k_3(T, \vec{\lambda}) = 0 \tag{4.37}$$

Para os seguintes conjuntos de funções bases:

$$\mathbf{W} = \left\{\omega \in \Omega \to \mathbb{R}^2 : \int_\Omega \omega^2\, d\Omega < \infty; \omega_i \in \mathcal{W}_T\right\} \tag{4.38}$$

$$\mathbf{P} = \left\{\psi \in \Omega \to \mathbb{R}^2 : \int_\Omega \psi^2\, d\Omega < \infty; \psi_f \in \mathcal{H}_T\right\} \tag{4.39}$$

$$\mathbf{V} = \left\{ v \in \Omega \to \mathbb{R}^2 : \int_\Omega v^2 \, d\Omega < \infty; v \in \mathcal{U}_T \right\} \tag{4.40}$$

$$\mathbf{Z} = \left\{ T \in \Omega \to \mathbb{R}^2 : \int_\Omega T^2 \, d\Omega < \infty; T \in \mathcal{U}_T \right\} \tag{4.41}$$

4.3 Discretização Espacial

As funções pesos escolhidas para um projeto de elementos finitos podem adquirir diverss formas e comportmentos. Para este projeto, foi utilizada Formulação de Galerkin devido à simplicidade do caráter físico do problema. Nesta formulação, as funções peso de cada variável são iguais de modo que as equações apresentadas se tornam:

$$\int_\Omega \frac{\partial \omega}{\partial t} \delta \, d\Omega + \int_\Omega v_x \frac{\partial \omega}{\partial x} \delta \, d\Omega + \int_\Omega v_y \frac{\partial \omega}{\partial y} \delta \, d\Omega + \int_\Omega \nu \left(\frac{\partial \omega}{\partial x}\frac{\partial \delta}{\partial x} + \frac{\partial \omega}{\partial y}\frac{\partial \delta}{\partial y} \right) d\Omega = 0 \tag{4.42}$$

$$- \int_\Omega \left(\frac{\partial \psi}{\partial x}\frac{\partial \xi}{\partial x} + \frac{\partial \psi}{\partial y}\frac{\partial \xi}{\partial y} \right) d\Omega + \int_\Omega \omega \xi \, d\Omega = 0 \tag{4.43}$$

$$\int_\Omega v_x \phi \, d\Omega - \int_\Omega \frac{\partial \psi}{\partial y} \phi \, d\Omega = 0 \tag{4.44}$$

$$\int_\Omega v_y \phi \, d\Omega + \int_\Omega \frac{\partial \psi}{\partial x} \phi \, d\Omega = 0 \tag{4.45}$$

$$\int_\Omega \frac{\partial T}{\partial t} \lambda \, d\Omega + \int_\Omega v_x \frac{\partial T}{\partial x} \lambda \, d\Omega + \int_\Omega v_y \frac{\partial T}{\partial y} \lambda \, d\Omega + \int_\Omega \alpha \left(\frac{\partial T}{\partial x}\frac{\partial \lambda}{\partial x} + \frac{\partial T}{\partial y}\frac{\partial \lambda}{\partial y} \right) d\Omega = 0 \tag{4.46}$$

Assim, as discretizações são aplicadas sobre um domínio com n_e elementos e n_p nós. As variáveis discretizadas se tornam:

$$\omega(\vec{x}, t) = \sum_{i=1}^{n_p} \omega_i(t) N_i(\vec{x}) \tag{4.47}$$

$$\psi(\vec{x}, t) = \sum_{i=1}^{n_p} \psi_i(t) N_i(\vec{x}) \tag{4.48}$$

$$v_x(\vec{x},t) = \sum_{i=1}^{n_p} v_{f,x,i}(t) N_i(\vec{x}) \tag{4.49}$$

$$v_y(\vec{x},t) = \sum_{i=1}^{n_p} v_{f,y,i}(t) N_i(\vec{x}) \tag{4.50}$$

$$T(\vec{x},t) = \sum_{i=1}^{n_p} T_i(t) N_i(\vec{x}) \tag{4.51}$$

Em que os valores das funções em cada nó $\omega_i = [\omega_1, \ldots, \omega_{np}]$, $\psi = [\psi_i, \ldots, \psi_{np}]$, $v_x = [v_{x,1}, \ldots, v_{x,np}]$, $v_y = [v_{y,1}, \ldots, v_{y,np}]$ e $T_i = [T_1, \ldots, T_{np}]$ são as icógnitas. Como estas funções operam indepedentemente do tempo, elas são retiradas das integrais sobre o domínio Ω. Em relação às funções de aproximação $N_i(\vec{x}) = [N_1(\vec{x}), \ldots, N_{np}(\vec{x})]$ são denotadas de funções base, e são escolhidas arbitrariamente. Como foi apresentado previamente, para a formulação de Galerkin, as funções de base se tornam:

$$\delta(\vec{x},t) = \sum_{j=1}^{n_p} \delta_j(t) N_j(\vec{x}) \tag{4.52}$$

$$\phi(\vec{x},t) = \sum_{j=1}^{n_p} \phi_j(t) N_j(\vec{x}) \tag{4.53}$$

$$\xi(\vec{x},t) = \sum_{j=1}^{n_p} \xi_j(t) N_j(\vec{x}) \tag{4.54}$$

$$\lambda(\vec{x},t) = \sum_{j=1}^{n_p} \lambda_j(t) N_j(\vec{x}) \tag{4.55}$$

Dessa forma, as equações em suas formas variacionais discretizadas espacialmente se tornam:

$$\begin{aligned}
&\int_\Omega \sum_{i=1}^{n_p} \frac{\partial \omega}{\partial t} N_i \sum_{j=1}^{n_p} \delta_j N_j \, d\Omega \\
&+ \int_\Omega v_x \sum_{i=1}^{n_p} \frac{\partial(\omega_i N_i)}{\partial x} \sum_{j=1}^{n_p} \delta_j N_j \, d\Omega \\
&+ \int_\Omega v_y \sum_{i=1}^{n_p} \frac{\partial(\omega_i N_i)}{\partial y} \sum_{j=1}^{n_p} \delta_j N_j \, d\Omega \\
&+ \int_\Omega \nu \sum_{i=1}^{n_p} \sum_{j=1}^{n_p} \left(\frac{\partial(\omega_i N_i)}{\partial x} \frac{\partial(\delta_j N_j)}{\partial x} + \frac{\partial(\omega_i N_i)}{\partial y} \frac{\partial(\delta_j N_j)}{\partial y} \right) d\Omega = 0
\end{aligned} \tag{4.56}$$

$$-\int_\Omega \left(\sum_{i=1}^{n_p} \frac{\partial \psi_i N_i}{\partial x} \sum_{j=1}^{n_p} \frac{\partial \xi_j N_j}{\partial x} + \sum_{i=1}^{n_p} \frac{\partial \psi_i N_i}{\partial y} \sum_{j=1}^{n_p} \frac{\partial \xi_j N_j}{\partial y} \right) + \int_\Omega \sum_{i=1}^{n_p} \omega_i N_i \sum_{j=1}^{n_p} \xi_j N_j d\Omega = 0 \tag{4.57}$$

$$\int_\Omega \sum_{i=1}^{n_p} v_{x,i} N_i \sum_{j=1}^{n_p} \phi_j N_j d\Omega - \int_\Omega \sum_{i=1}^{n_p} \frac{\partial \psi_i N_i}{\partial y} \sum_{j=1}^{n_p} \phi_j N_j d\Omega = 0 \tag{4.58}$$

$$\int_\Omega \sum_{i=1}^{n_p} v_{y,i} N_i \sum_{j=1}^{n_p} \phi_j N_j d\Omega + \int_\Omega \sum_{i=1}^{n_p} \frac{\partial \psi_i N_i}{\partial x} \sum_{j=1}^{n_p} \phi_j N_j d\Omega = 0 \tag{4.59}$$

$$\begin{aligned}
&\int_\Omega \sum_{i=1}^{n_p} \frac{\partial T}{\partial t} N_i \sum_{j=1}^{n_p} \delta_j N_j \, d\Omega \\
&+ \int_\Omega v_x \sum_{i=1}^{n_p} \frac{\partial (T_i N_i)}{\partial x} \sum_{j=1}^{n_p} \delta_j N_j \, d\Omega \\
&+ \int_\Omega v_y \sum_{i=1}^{n_p} \frac{\partial (T_i N_i)}{\partial y} \sum_{j=1}^{n_p} \delta_j N_j \, d\Omega \\
&+ \int_\Omega \alpha \left(\sum_{i=1}^{n_p} \frac{\partial (T_i N_i)}{\partial x} \sum_{j=1}^{n_p} \frac{\partial (\delta_j N_j)}{\partial x} + \sum_{i=1}^{n_p} \frac{\partial (T_i N_i)}{\partial y} \sum_{j=1}^{n_p} \frac{\partial (\delta_j N_j)}{\partial y} \right) d\Omega = 0
\end{aligned} \tag{4.60}$$

É possível remover as componentes v_x e v_y da equuação 4.57, pois, para tornar a equação linear, serão utilizados os componentes da velocidade no último passo de tempo para estes valores. Dessa forma, conseguimos representar de modo implícito a discretização espacial do escoamento por transporte de vorticidade.

4.4 Domínio Computacional

A escolha do tipo de malha computacional varia dependendo da aplicação, seja por questões de refino ou de modeloagem matemática. O comum é a utilização de elementos maiores em regiões simples enquanto se aplica um refino localizado nas partes do domínio que estão sujeitas à instabilidades ou que os efeitos do modelo são mais intensos, necessitando de uma discretização maior para conseguir reproduzir os efeitos do modelo com clareza.

Para as equações de Navier-Stokes, é comum a utilização de elementos retangulares ou cúbicos devido ao acoplamento dos parâmetros de pressão e velocidade que

descreve a equação. Todavia, ao utilizar a formulação função corrente-vorticidade, há a possibilidade de aplicar elementos de geometria triangular no modelo, contornando esse acoplamento tanto computacionalmente quanto matematicamente.

Como é apresentado por Cunha [9] e Carrasco [11], para o problema de função corrente-vorticidade, os elementos utilizados normalmente são do tipo triangular ou retangular. Enquanto que os polinômios interpoladores possuem ordem linear, quadrática ou cúbica.

Neste trabalho, foi adotado a discretização geométrica triangular em todos os elementos do domínio, assim como a utilização de polinômios interpoladores de ordem linear. Dessa forma, cada elemento do domínio é representado da seguinte forma:

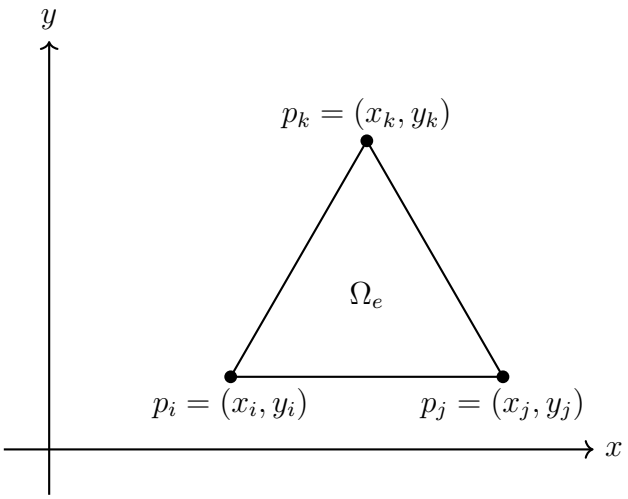


Figura 4.1: Representação geométrica de um domínio elementar triangular.

Como é apresentado pela figura, cada elemento é caracterizado pela presença de 3 nós p_i, p_j e p_k. Considerando a função interpoladora linear, consegue-se reescrever as equações 4.57, 4.58, 4.59 e 4.60 discretizadas espacialmente na sua forma matricial:

$$M\frac{\partial \omega}{\partial t} + v_x G_x \omega + v_y G_y \omega + \nu(K_{xx} + K_{yy})\omega = 0 \tag{4.61}$$

$$-(K_{xx} + K_{yy})\psi + M\omega = 0 \tag{4.62}$$

$$M v_x \omega - G_y \psi = 0 \tag{4.63}$$

$$Mv_y\omega + G_x\psi = 0 \tag{4.64}$$

$$M\frac{\partial T}{\partial t} + v_x G_x T + v_y G_y T + \alpha(K_{xx} + K_{yy})T = 0 \tag{4.65}$$

Em que as matrizes M,G_x, G_y, K_{xx} e K_{yy} são as matrizes globais do problema reescritas na forma matricial com formato $n_p \times n_p$.

Estas matrizes são formadas a partir de um processo chamado de assembly. Nele, passando por cada elemnto, utiliza-se matrizes locais m^e, $g^e y$, $g^e x$, $k^e yy$ e $k^e xx$ para adicionar nas matrizes globais a contribuição de cada elemento. A definição das matrizes locais dos elementos é comumente apresentada na literatura como em Zienkiewicz [8]:

$$m^e = \int_{\Omega^e} N_i^e N_j^e d\Omega^e \tag{4.66}$$

$$g_x^e = \int_{\Omega^e} \frac{\partial N_i^e}{\partial x} N_j^e d\Omega^e \tag{4.67}$$

$$g_y^e = \int_{\Omega^e} \frac{\partial N_i^e}{\partial y} N_j^e d\Omega^e \tag{4.68}$$

$$k_{xx}^e = \int_{\Omega^e} \frac{\partial N_i^e}{\partial x} \frac{\partial N_j^e}{\partial x} d\Omega^e \tag{4.69}$$

$$k_{yy}^e = \int_{\Omega^e} \frac{\partial N_i^e}{\partial y} \frac{\partial N_j^e}{\partial y} d\Omega^e \tag{4.70}$$

As matrizes locais utilizam a formulação obtida por Lewis, Nithiarasu e Seetharamu que é apresentado por Cunha [9]. Dessa forma, é possível escrever para elementos triangulares as matrizes locais como:

$$\mathbf{m^e} = \frac{A^e}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

$$\mathbf{g_x^e} = \frac{1}{6}\begin{bmatrix} b_i & b_j & b_k \\ b_i & b_j & b_k \\ b_i & b_j & b_k \end{bmatrix}$$

$$\mathbf{g_y^e} = \frac{1}{6}\begin{bmatrix} c_i & c_j & c_k \\ c_i & c_j & c_k \\ c_i & c_j & c_k \end{bmatrix}$$

$$\mathbf{k_{xx}^e} = \frac{1}{4A^e}\begin{bmatrix} b_i b_i & b_j b_i & b_k b_i \\ b_i b_j & b_j b_j & b_k b_j \\ b_i b_k & b_j b_k & b_k b_k \end{bmatrix}$$

$$\mathbf{k_{yy}^e} = \frac{1}{4A^e}\begin{bmatrix} c_i c_i & c_j c_i & c_k c_i \\ c_i c_j & c_j c_j & c_k c_j \\ c_i c_k & c_j c_k & c_k c_k \end{bmatrix}$$

Em que A_e é a área do elemnto e os parâmetros b_i, b_j, b_k, c_i, c_j e c_k são denominadas coordenaedas relativas do elemento. Para obter estes valores, utiliza-se das coordenadas globais do elemento

$$p_i$$

, p_j e p_k apresentadas previamente por meio das definições:

$$b_i = y_j - y_k \tag{4.71}$$

$$b_j = y_k - y_i \tag{4.72}$$

$$b_k = y_i - y_j \tag{4.73}$$

$$c_i = x_k - x_j \tag{4.74}$$

$$c_j = x_i - x_k \tag{4.75}$$

$$c_k = x_j - x_i \tag{4.76}$$

Em que os valores (x_i, y_i), (x_j, y_j) e (x_k, y_k) são as coordenadas globais dos nós p_i, p_j e p_k. Por fim, pode-se calcular a área do elemento por meio da fórmula onde xi e yi são as coordenadas do nó pi, xj e yj são as coordenadas do nó pj e xk e yk são

as coordenadas do nó pk. E a área A pode ser calculada através das coordenadas dos nós pela equação:

$$A^e = \frac{1}{2}((x_i y_j - x_j y_i) + (x_j y_k - x_k y_j) + (x_k y_i - x_i y_k)) \tag{4.77}$$

4.5 Discretização Temporal

As equações de função corrente-vorticidade assim como a derivada material da temperatura avaliam o comportamento e o desenvolvimento destas variáveis tanto espacialmente quanto temporalmente como foi apresentado pelas equações 4.1 e 4.4. Para conseguir simular numericamente estas equações, a metodologia de elementos finitos é útil para a descrição espacial do problema, todavia, comumente este método não é utilizado para avaliar os termos influenciados pela derivada temporal.

Para conseguir visualizar os efeitos de desenvolvimento do escoamento, uma possibilidade é utilizir a discretização em diferenças finitas (MDF) para os termos que estão englobados pelas derivadas temporais. Este método apresentado por Cunha [9], Sousa [10] e Lucas [13] por exemplo é uma forma simples e efetiva de tratar a discretização temporal sem haver a necessidade de criar mais uma matriz que futuramente necessitaria ser invertida. Dessa forma, além do método ser eficaz, ele é computacionalmente otimizado para este problema.

O método de diferenaçs finitas ou MDF tem como base em discretizar pequenas diferenças de uma determinada função para aproximar do comportamento derivativo devido à definição da série de Taylor, que infere qualquer função $f(x)$ pode ser representada como:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \tag{4.78}$$

Desta forma, aproximando uma função $f(x)$ somente para seu primeiro termo, consegue-se sua versão truncada para o termo de primeira ordem:

$$f(x) \approx f^{'}(a)(x-a) + O(1) \tag{4.79}$$

Sendo o termo $O(1)$ o erro de primeira ordem.

Esta aproximação quando utilizada nos termos de vorticidade e temperatura torna-se:

$$\frac{\partial \omega}{\partial t} \approx \frac{\omega(t+\Delta t) - \omega(t)}{\Delta t} \tag{4.80}$$

$$\frac{\partial T}{\partial t} \approx \frac{T(t+\Delta t) - T(t)}{\Delta t} \tag{4.81}$$

Em que Δt é a discretização temporal referente ao passo de tempo da simulação.

Para a aplicação do método de diferenças finitas, é possível a aplicação tanto da forma implícita quanto explícita. Para este trabalho, torna-se interressante a utilização do método em sua forma implícita, visto que a forma implícita é mais estável.

Aplicando o método de diferenças finitas nas equações 4.61 e 4.65, determina-se a forma discretizada espacialmente e temporalmente do problema de função corrente-vorticidade.

$$\left[\frac{M}{\Delta t} + \nu(K_{xx} + K_{yy}) + (v_x G_x + v_y G_y)\right] \omega^{n+1} = \frac{M}{\Delta t}\omega^n + c.c. \tag{4.82}$$

$$\left(K_{xx} + K_{yy}\right) \psi^n = M\omega^n + c.c. \tag{4.83}$$

$$M v_x^n = G_y \psi^n + c.c. \tag{4.84}$$

$$M v_y^n = -G_x \psi^n + c.c \tag{4.85}$$

$$\left[\frac{M}{\Delta t} + \alpha(K_{xx} + K_{yy}) + (v_x G_x + v_y G_y)\right] T^{n+1} = \frac{M}{\Delta t} T^n + c.c. \tag{4.86}$$

Em que o parâmetro *c.c.* são as condições de contorno do problema, que como foi apresentado previamente, são condições do tipo Dirichilet ou Neumann zero.

Capítulo 5

Metodologia

O projeto se trata do desenvolvimento de um conjunto de algorítmos em Python para criar e solucionar diferetes casos de CFD (dinâmica de fluido computacional) em geometrias diferentes conforme foi comentado na seção 1.1 e 1.2. Por meio da revisão bibliográfica abordada na secção 2 e a formulação matemática citada no capítulo prévio, é possível assim definir o escopo geométrico do projeto.

Será analisado o comportamento do escomaneto para um trecho plano reto simples. Este será caracterizado por um trecho com 0,1 metros de comprimento simples, com uma entrada e uma saída. Por meio deste caso será analisado o efeito das superfícies intensificadoras de troca de calor.

A decisão para esta forma geométrica advém da facilidade de comparar e validar o modelo computacional com soluções analíticas apresentadas na literatura como em White [3] assim como a aplicação das condições de contorno do trecho reto e a visualização de resultados importantes como a velocidade média e a temperatura

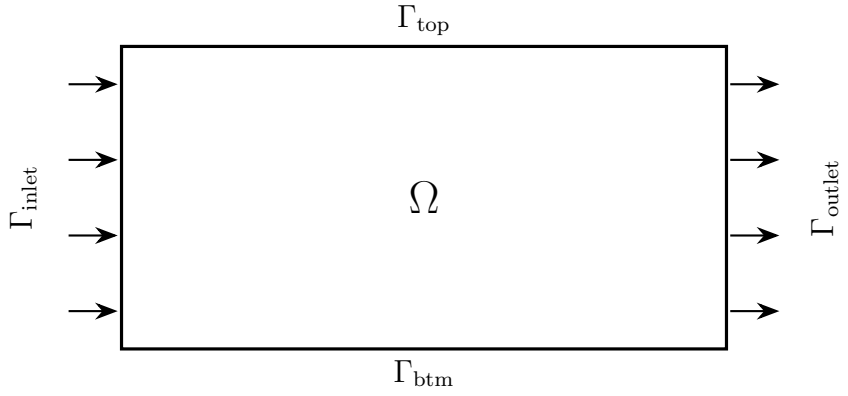


Figura 5.1: Representação geral do trecho reto entre placas.

média energética.

Ao todo, foram separados 5 tipos diferentes de perfis para as paredes, sendo estas do tipo plana, elíptica, trapezoidal, triangular e corrugado. O tipo plano é o perfil sem variações geométricas nas paredes, de modo a servir como parâmetro de controle para as outras configurações geométricas assim como forma de validar o modelo e o código CFD por meio de soluções mais controladas, podendo ou não ser analíticas.

Os perfis seguintes descrevem de modo geral o tipo de superfície intensificadora que será aplicado ao escoamento, podendo ter uma variação periódica de geometria definida (seja triangular, trapezoidal ou utilizando de trechos de elipses) ou uma variação não periódica indefinida como o caso corrugado. Para este último, as paredes do escoamento são geradas por meio de uma função de ruído computacional que é manipulado para gerar assim um perfil rugoso.

Em relação aos perfis com variação geométrica periódica, somente o formato do perfil sofrerá variações, de modo a padronizar parâmetros como distância entre as mudanças de nível e altura máxima destas variações geométricas.

Por fim, para casos alatórios, será também aplicado uma condição de entrada não pertubada. Isto é, para os primeiros 5 porcento da geometria, as placas terão configuração plana, para então seguir com a forma corrugado do perfil.

Assim, ao todo foram tabelados 12 casos diferentes de geometrias para serem analisados, gernado a tabela de casos abaixo.

Tabela 5.1: Parâmetros dos casos simulados

Caso	Tipo de Perfil	n_{pontos}	$L_{\text{reto}}(m)$	offset (m)	$d(m)$	$l(m)$	$a(m)$	tm	tmr	Entrada Limpa
C0	Plano	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0
C1	Trapezoidal	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0
C2	Triangular	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0
C3	Circular	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0
C4	Corrugado	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0
C5	Corrugado	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	1
C6	Trapezoidal	500	0,1	0,0027	0,005	0,001	0,00027	$2,00 \times 10^{-4}$	$2,00 \times 10^{-5}$	0
C7	Triangular	500	0,1	0,0027	0,005	0,001	0,00027	$2,00 \times 10^{-4}$	$2,00 \times 10^{-5}$	0
C8	Circular	500	0,1	0,0027	0,005	0,001	0,00027	$2,00 \times 10^{-4}$	$2,00 \times 10^{-5}$	0

5.1 Parâmetros Físicos

Para esta seção, será discutido as escolhas de parâmetros físicos que influenciam o escoamento.

5.1.1 Condições de Entrada do Fluido

Conforme apresentado nos capítulos 4 e 2, as condições de entrada do fluido no canal são importantes para determinar o comportamento da função corrente nas condições de contorno, que por consequência determina o perfil do escoamento.

Ademais, informações importantes como a velocidade de entrada do fluido são fundamentais para garantir a veracidade e funcionamento do código, pois, conforme apresentado nos trabalhos de Carrasco [11], Gomes [12] e a literatura apresentada por Zienkiewicz [8], o modelo computacional utilizado é valido para regiões de escoamento do tipo laminar caso não seja utilizado métodos computacionais mais robustos para tratar a difusão numérica.

Desta forma, como forma de controlar e garantir um comportamento laminar do escoamento, a decisão feita foi de definir a velocidade de entrada do escoamento a partir do número de Reynolds. Essa decisão é interessante por conta do caráter adimensional do parâmetro, o que permite que seja aplicado à diferentes tipos de fluidos e geometrias.

Assim, o projeto determina a velocidade de entrada do escoamento analisando a altura do canal nos nós de entrada, e o classificando como um parâmetro D. Por meio deste parâmetro, que é próximo numericamente da altura real escolhida entre os canais, conseguimos calcular inversamente a velocidade de entrada da simulação para chegar próximo ao número de reynolds médio desejado pela simulação. Usando a fórmula:

$$\mathbf{v}_{inlet} = \frac{Re \cdot \mu}{\rho \cdot H} \tag{5.1}$$

Sendo o valor v_{inlet} aplicado para todos os nós de entrada do escoamento. Desta forma, consegue-se controlar o tipo de desenvolvimento dinâmico do escoamento. Para este projeto, foi definido um Reynolds de entrada $Re_{inlet} = 50$ para os casos simulados, excetuando-se para a validação do código, que utilizará um Reynolds

inferior de $Re_{inlet} = 20$.

Tomando como exemplo um dos casos de validação, é possível visualizar a forma das condições de entrada como:

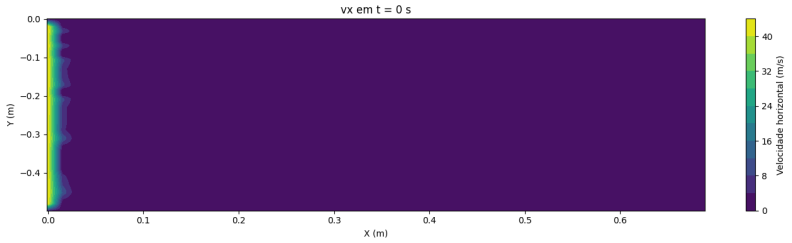


Figura 5.2: Velocidade horizontal no contorno de entrada para um caso plano.

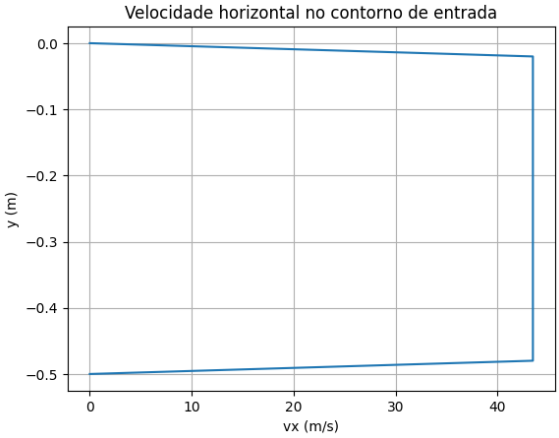


Figura 5.3: Representação Gráfica da velocidade na entrada.

5.1.2 Condições Energéticas

Para as condições térmicas do escaomento, para visualizar o efeito térmico das variações geométricas, foi optado por utilizar condições de temperatura preescrita para as paredes e para a entrada do escoamento para todos os casos analisados como é apresentado no capítulo 3. Esta decisão vem da facilidade de verificar e validar o modelo computacional para este tipo de problema térmico, visto que para condições analíticas, o perfil de temperatura é parabólico conforme apresentado por Incropera [1] e Gomes [12].

Neste projeto, as temperaturas definidas são de $T_{inlet} = 40°C$ e $T_{wall} = 50°C$. Para os nós que não estão nestas regiões de contorno de entrada e paredes, a temperatura definida inicialmente será de $T_{non-cc} = 45°C$. Desta forma, será possível

visualizar a influência da temperatura de entrada do fluido e da parede de modo simultâneo ao longo do escoamento.

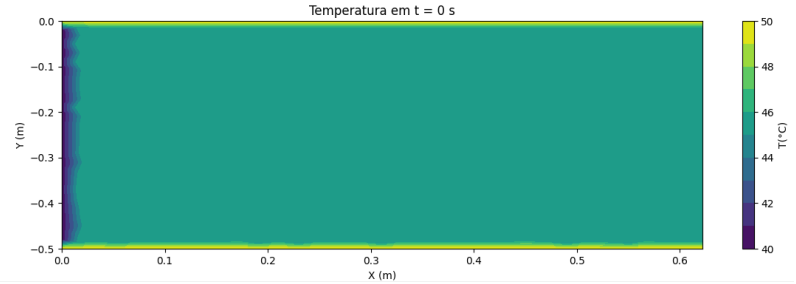


Figura 5.4: Temperatura na região próxiam do contorno de entrada para um caso plano.

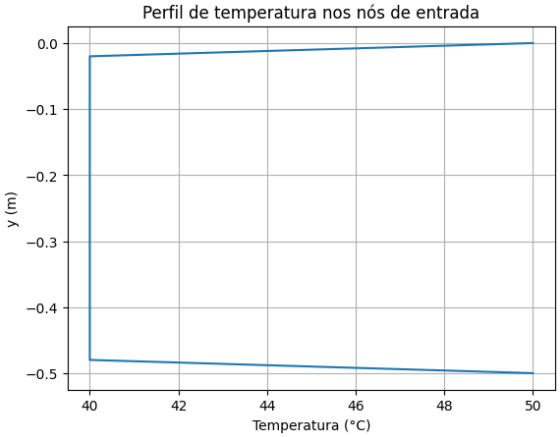


Figura 5.5: Representação Gráfica da temperatura no contorno de entrada

5.1.3 Escolha do fluido

Antes de gerar e simular os casos geométricos, é necessário também inferir qual o fluido de trabalho, visto que, ele determina diretamente parâmetros como viscosidade, massa específica e condutividade térmica, parâmetros estes que alteram os números adimensionais que controlam o comportamento de um escoamento fluidodinâmico, conforme apresentado nas seções 2.3 e 3.5.

Para isto, foi feito uma seleção para diferentes tipos de fluidos, analisando termos gerais de viscosidade e difusividade térmica para reduzir o escopo e assim determinar o fluido de trabalho para a análise. Em geral, como o objetivo é verificar o efeito geométrico sobre o escoamento e a temperatura, torna-se interessante fluidos com alta viscosidade.

Para fins de simplificação e redução do custo computacional, as propriedades termofísicas do fluido serão consideradas constantes ao longo de toda a simulação. Embora propriedades como viscosidade, massa específica, condutividade térmica e calor específico possam variar com a temperatura, especialmente em escoamentos com gradientes térmicos significativos, a hipótese de propriedades constantes é aceita em análises preliminares ou quando as variações são pequenas dentro da faixa de operação. Como a diferença entre a temperatura de entrada ($40°C$) e a temperatura de parede ($50°C$) citadas no capítulo 5.1.2 é de $10°C$, as variações das propriedades ao longo do domínio tornam-se modestas, permitindo assim a aproximação de propriedades constantes tomadas com base na temperatura de filme de $45°C$.

Desta forma, utilizando um script em python, foi calculado utilizando a biblioteca Coolprop as propriedades da água, do fluido refrigerante R-134a e do fluido refrigerante R-410a para esta temperatura de filme. Ademais, utilizando de uma calculadora de propriedades para óleos de diferentes graus APIs e KUOPs seguindo a metodologia da ASTM [23], também foi inserido na analíse uma configuração de óleo grau API 35 e KUOP 12. Desta forma, foi montado a tabela abaixo comparando estes resultados.

Tabela 5.2: Propriedades termodinâmicas dos fluidos a $45°C$

Propriedade	**Óleo Grau API 35**	**Água**	**R134a**	**R410A**
ρ (kg/m^3)	830.5	990.2	1127.2	954.5
ν (m^2/s)	4.797×10^{-6}	6.017×10^{-7}	1.351×10^{-7}	9.669×10^{-8}
μ (Pa $\cdot$ s)	3.989×10^{-3}	5.958×10^{-4}	1.523×10^{-4}	9.229×10^{-5}
k (W/(m $\cdot$ K))	0.1224	0.6347	0.0728	0.0666
C_p (J/(kg $\cdot$ K))	2036.49	4180.36	1523.93	1990.57
α (m^2/s)	7.240×10^{-8}	1.533×10^{-7}	4.239×10^{-8}	3.962×10^{-8}

Conforme a tabela de viscosidades e difusivades térmicas, decidiu-se que, de modo a conseguir visualizar melhor os comportamentos do escoamento em influência das variações geométricas, a melhor opção seria o óleo grau API 35 citado. Desta forma, defini-se as propriedades físicas do fluido para o escoamento por meio da

tabela abaixo.

Tabela 5.3: Propriedades do óleo

Propriedade	**Valor**	**Unidade**
ρ	830.5	kg/m^3
ν	$4{,}797 \times 10^{-6}$	m^2/s
μ	$3{,}989 \times 10^{-3}$	Pa·s
k	0,1224	W/(m·K)
C_p	2036,49	J/(kg·°C)
α	$7{,}24 \times 10^{-8}$	m^2/s
Pr	$66,24$	

5.2 Parâmetros Geométricos

Para esta seção, será discutido as escolhas de parâmetros geométricos que influenciam o escoamento.

5.2.1 Variáveis Geométricas

Assim como é apresentado nos projetos de Yadav [22], há uma necessidade de parametrizar os valores geométricos do perfil tanto para o caso de variações regulares como para casos irregulares como o perfil corrugado.

Deste modo, para os casos periódicos, necessita-se de parâmetros que definem valores geométricos entre os obstáculos. Neste caso, parâmetros que controlam a amplitude e o período dos obstáculos devem ser evidenciados. Definido a, l, d e l_t como parâmetros que controlam a altura ,largura, a distância entre os obstáculos e a espessura do topo para o caso trapezoidal respectivamente, visualiza-se a forma periódica de um perfil conforme abaixo. Para este projeto, foi adotado um valor de $a = 2,7mm$ e $d = 50mm$ para todos os casos. Desta forma, garante-se que a obstrução ao longo do escoamento não ultrapassa um limite que impossibilita o escoamento assim como garante uma distância razoável entre as variações de altura dos perfis. Para o caso trapezoidal, o parâmetro l_t acompanha o número de discre-

tizações feitas para as paredes superiores e inferiores, de modo a equivaler 90% do comprimento total l definido.

Para os casos corrugados, busca-se uma nuvem de pontos não padronizada em torno de um valor médio referente à posição da parede em sua configuração lisa. Desta forma, por meio da inserção de um valor de seed, conseguimos uma nuvem de pontos semi-aleatória que se aproxima do caso ideal.

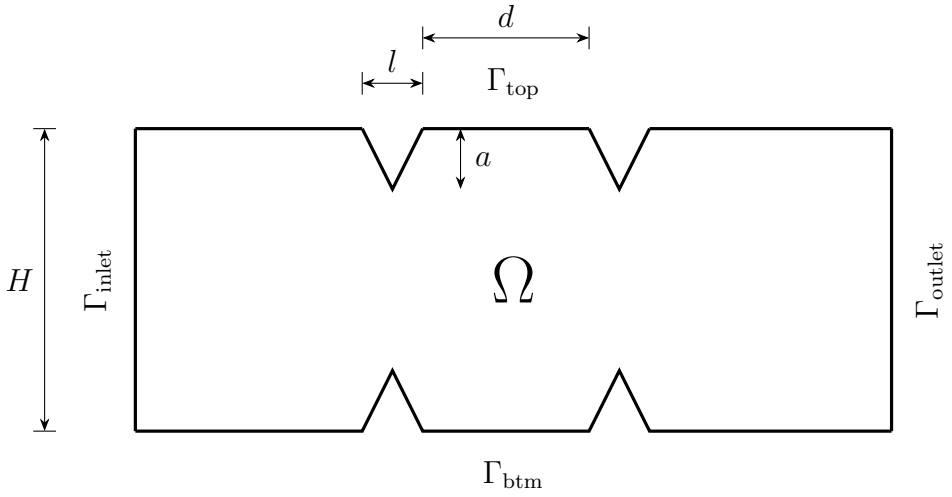


Figura 5.6: Canal com perfis triangulares.

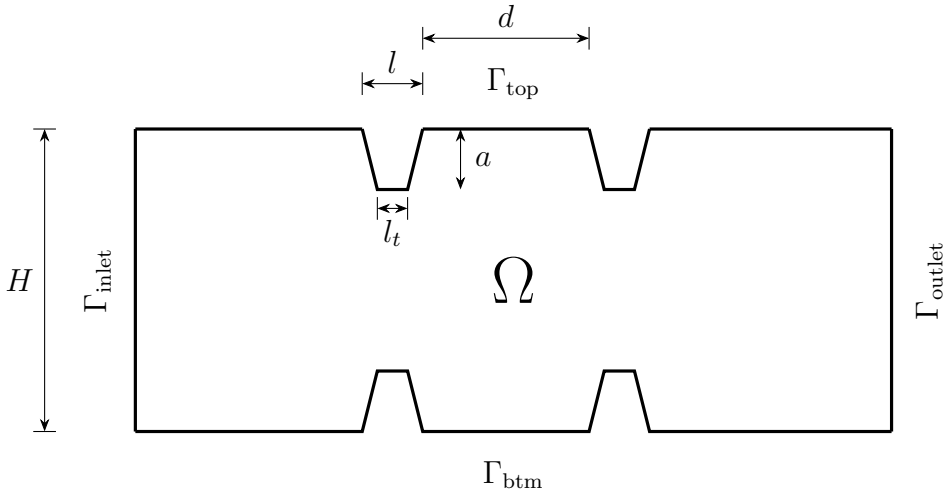


Figura 5.7: Canal com perfis trapezoidais.

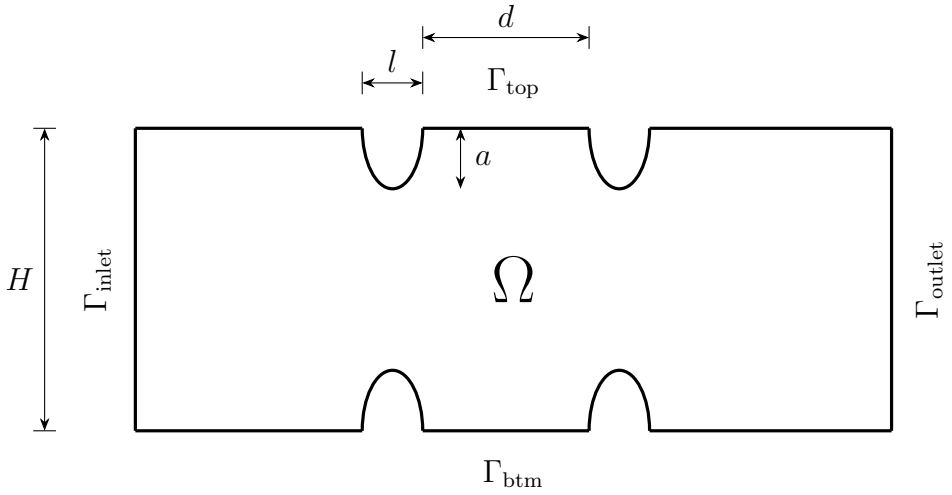


Figura 5.8: Canal com perfis elípticos.

Em relação ao caso corrugado, além da inserção de pontos, também foi adicionado a possibilidade de inserção de um trecho plano ideal, denominado entrada limpa. Essa inserção tem como objetivo verificar possíveis efeitos das condições de entrada do campo de velocidades no resultado dinâmico do escoamento entre placas de um perfil corrugado. Neste projeto, o perfil plano para o caso com a entrada limpa será de 20% do comprimento requisitado, de modo a adicionar ao comprimento requisitado a entrada limpa. Dessa forma, neste trabalho, o caso com entrada limpa não terá a avaliação do comportamento energético do escoamento, visto que o comprimento adicional impacta diretamente na tempertura média final do escoamento.

5.2.2 Altura Entre Placas

Como o projeto pretende utilizar parâmetros reais de fluido para a análise computacional do escoamento, o parâmetro que define a distância entre as placas tem que ser considerado, visto que este valor influencia no número de Reynolds e no tipo de escoamento. Para isto, foi utilizao uma tabela de tubos de diâmetros padronizados seguindo a norma IPS para estimar um valor para a distância entre as placas.

Tabela 5.4: Dimensões e propriedades de tubos de Alumínio IPS

Diâmetro Nominal (pol.)	Diâmetro Externo (mm)	Diâmetro Interno (mm)	Espessura (mm)	Peso (kg/m)
3/8	17,15	12,52	2,31	0,291
1/2	21,34	15,80	2,77	0,436
3/4	26,67	20,93	2,87	0,582
1	33,40	26,64	3,38	0,885
1 1/4	42,16	35,05	3,56	1,170
1 1/2	48,26	40,90	3,68	1,399
2	60,33	52,51	3,91	1,881
2 1/2	73,03	62,71	5,16	2,982
3	88,90	77,92	5,49	3,901
3 1/2	101,60	90,12	5,74	4,689
4	114,30	102,26	6,02	5,555
4 1/2	127,00	114,46	6,27	6,424
5	141,30	128,20	6,55	7,580
6	168,28	154,05	7,11	9,767
8	219,10	202,74	8,18	14,689

Assim, consirando aplicações comuns de serviço, a distância entre as placas foi tomada como constante e igual a 27 mm.

5.3 Geração de Geometria

Para criar as diferentes formas geométricas citadas, torna-se necessário a utilização de um algoritmo que define a curva geométrica para que então seja redirecionado a um algoritmo de geração de malha. Desta forma, tanto para o caso reto quanto para o caso curvo, foi feito um algoritmo específico para os casos periódicos e outro para o caso com pertubação aleatória.

Como deste algoritmo é possível encaminhar a geometria resultante a outro software ou algoritmo para a geração da malha, torna-se interessante o armazenamento das curvas em algum formato que possa ser lido posteriormente. Desta forma, optou-se por armazenar os pontos e as linhas que caracterizam a geometria em um arquivo .geo, que é próprio para arquivos do tipo de forma geométrica para então serem lidos por algoritmos especializados de softwares como o Gmsh.

Para o caso periódico, o algoritmo define a curva ou reta principal do escoamento, e desta curva, é formado as paredes superiores e inferiores por meio de um offset aplicado diretamente ao caso principal. Por meio destas novas curvas que representam as paredes então é inserido as diferentes formas de perfis que cada caso

utiliza. Para isto, é usado uma função degrau periódica sobre cada curva para definir os intervalos em que será aplicado uma função de forma específica que caracteriza o perfil requerido. Tomando por exemplo um caso trapezoidal com perfis finos, a curva obtida seria:

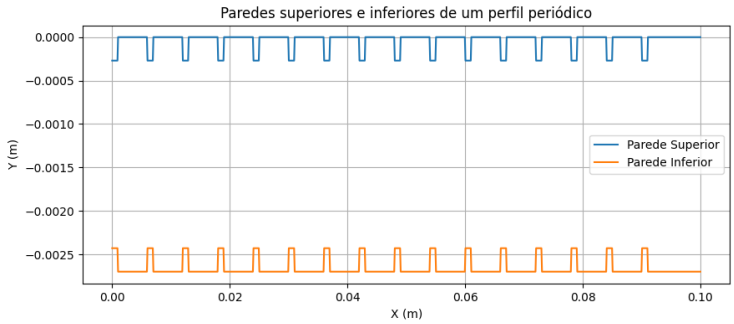


Figura 5.9: Curvas de paredes superiores e inferiores em um perfil trapezoidal periódico.

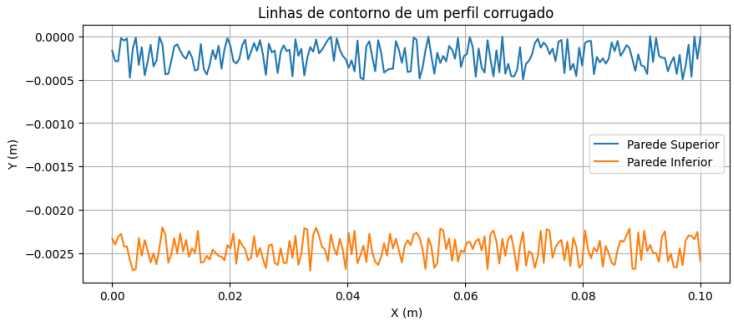


Figura 5.10: Curvas de paredes superiores e inferiores em um perfil corrugado.

Após a geração das paredes, é feito duas retas que ligam os pontos iniciais e os pontos finais das curvas, formando assim um perfil com as linhas de contorno delimitando o domínio para a geração de malha.

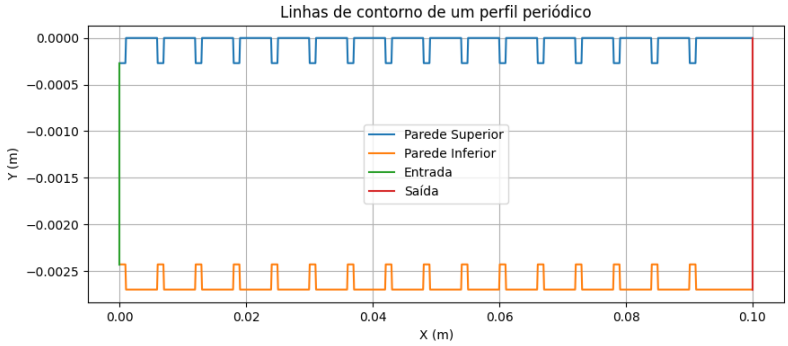


Figura 5.11: Linhas de contorno em um perfil trapezoidal periódico.

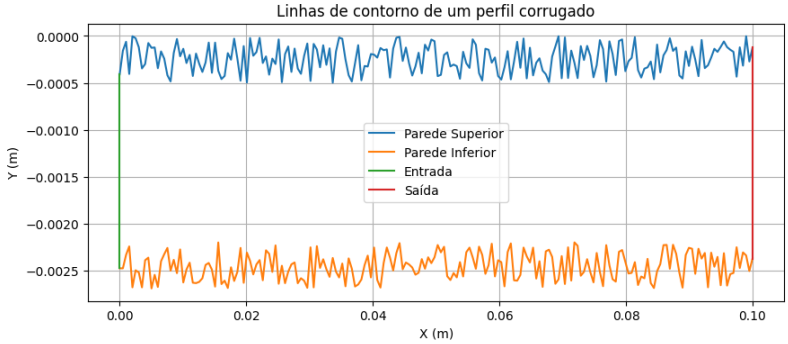


Figura 5.12: Linhas de contorno em um perfil corrugado.

É escrito no formato .geo os pontos, as linhas que conectam os pontos e os nomes físicos de cada curva para facilitar a definição dos tipos de elementos para as condições de contorno. Com o arquivo, é possível visualizar a geometria obtida no software Gmsh:

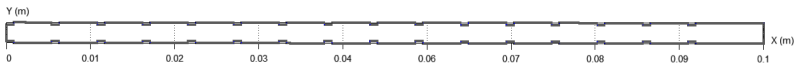


Figura 5.13: Linhas de contorno em um perfil trapezoidal periódico.

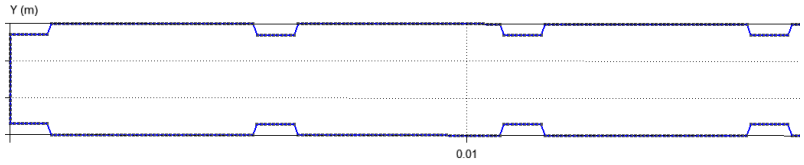


Figura 5.14: Pontos e linhas de contorno próximo da região de entrada em um perfil periódico.

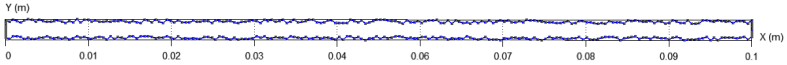


Figura 5.15: Linhas de contorno em um perfil corrugado.

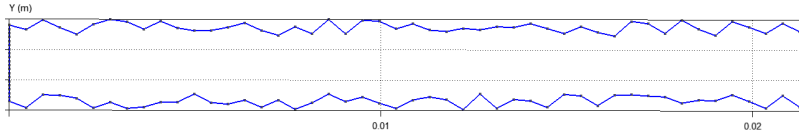


Figura 5.16: Pontos e linhas de contorno próximo da região de entrada em um perfil corrugado.

5.4 Geração de Malha

Após a formação das geometrias de cada caso, é necessário a formação da malha para então conseguir resolver o escoamento por elementos finitos. Para isto, foi utilizado o software livre Gmsh, que permite por meio de arquivos .geo o desenho direto de geometrias.

Desta forma, as geometrias formadas pelo algoritmo prévio são imediatamente transformadas em malha quando abertas pelo Gmsh. Para isto, foram adotadas padronizações para o tipo de elemento e o tipo de algoritmo que forma os elementos ao longo da geometria. Para o projeto, foi adotado elementros triangulares com formação tipo Galerkin, visto que são elementos simples e não trazem problemas de análise ou de convergência para o tipo de problema analisado.

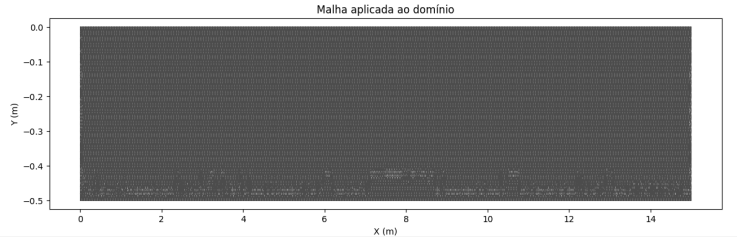


Figura 5.17: Malha Aplicada a um domínio simples.

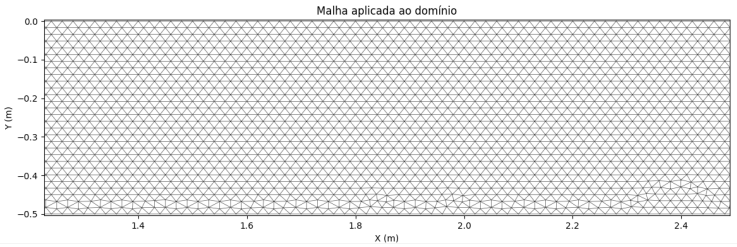


Figura 5.18: Elementos triangulares de uma malha aplicada a um domínio simples.

Por meio da geração da malha, conseguimos acesso à matriz de conectividade, conhecida com IEN. Esta matriz é essencial para a solução de qualquer problema que envolve elementos finitos, visto que por esta conseguimos referencias os elementos e suas conexões de modo próprio.

Para conseguir visualizar melhor comportamentos em determinadas regiões de interesse, é necessário refinar a malha ao redor da região em específico. Assim, visto o objetivo do trabalho, no algoritmo de geração de geometria é analisado a distância entre os pontos conforme estes são colocados no arquivo de modo que, caso a distância ultrapasse um limite que indica uma mudança de nível (sendo esta mudança uma forma de identificar que está em uma região de geração de turbulência devido à superfície intensificadora de troca de calor) o fator que indica o refino posterior da geração da malha é intensificado. Desse modo, estas regiões possuem uma densidade de elementos superior às outras, o que torna os resultados nestes pontos mais precisos.

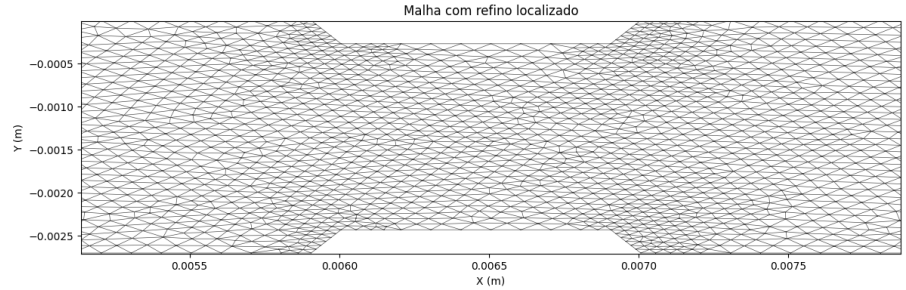


Figura 5.19: Malha com refino localizado aplicado em um perfil com variação periódica trapezoidal.

A etapa de geração de malhas também inclui a geração de malhas com refinos de níveis de refinamento diferentes, com o objetivo de verificar possíveis diferenças nos resultados finais que podem ser influenciados pelo erro existente no método que é controlado pela discretização geométrica do problema. Esta etapa é chamada de analíse de convergência e permite verificar o refino ótimo para uma malha sem que seja computacionalmente a mais custosa, o que implica em simulações mais rápidas.

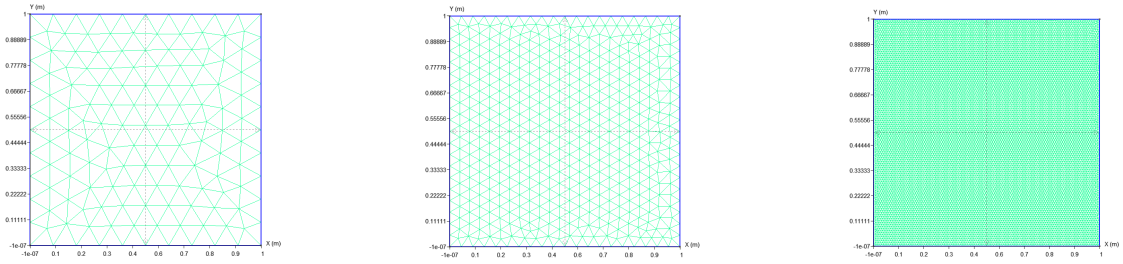


Figura 5.20: Diferentes refinos de malha para uma geometria quadrada simples.

Para este projeto, foi feito uma análise de convergência para a validação do modelo computacional. E, a partir do refino mínimo para que a norma de erro seja suficiente é definido os refinos para os casos de estudo.

5.5 Solucionador de Mecânica dos Fluidos

Um solucionador é necessário para utilizar as malhas geradas e assim simular o escoamento. Para isto, foi gerado um algoritmo em Python devido à sua linguagem de alto nível e bibliotecas diversas que atendem este tipo de problema. Este

algoritmo além de solucionar a equação, o algoritmo também armazena e plota os resultados para análises posteriores necessárias.

Para efetuar a solução, é amarrado em um loop sequencial a ordem que as variáveis de vorticidade, função corrente, velocidade e temperatura devem estar. A partir deste loop, é feito as inversões matriciais e o cálculo dos vetores de solução para os parâmetros, com a aplicação das condições de contorno referente à cada caso geométrico específico.

Após cada solução temporal, é verificado o passo de tempo para assim exportar ou não os resultados em um formato de arquivo próprio para visualização.

5.6 Visualização dos Resultados

Para visualizar os resultados obtidos em cada caso, será utilizado 2 ferramentas em conjunto, snedo estas o software gratuito Para-View e as bibliotecas de visualização disponibilizadas pelo Python.

Para geometrias mais compactas como os casos de validação, será avaliado o comportamento localizado em regiões próximas do contorno pelo Para-View, visto que este software disponibiliza uma série de ferramentas além de uma interface apropriada para a análise e visualização de resultados que envolvem simulações numéricas.

Todavia, devido à proporção da largura com o comprimento dos casos geométricos, a utilização somente do Para-View para os casos do projeto torna-se inviável. Desta forma, será utilizado as bibliotecas de representação gráfica do Python como forma de visualizar em diferentes seções assim como em todo o domínio os resultados obtidos para todos os casos geométricos de análise feito neste projeto.

5.7 Validação dos Resultados

Como os resultados obtidos tratam-se de uma simulação, torna-se necessário validar o modelo e a precisão do código para garantir a efetividade do projeto. Para isso, será comparado o resultado do caso 1, referente ao escoamento entre 2 placas planas, com as soluções analíticas de Hagen-Poiseuille.

Como este tipo de problema é um caso clássico, existem diversas soluções analíticas no meio acadêmico, justificando assim a sua escolha como uma forma

de validar o código tanto do ponto de vista dinâmico quanto térmico. Em suma, o caso que será utilizado para validar o código está descrito abaixo.

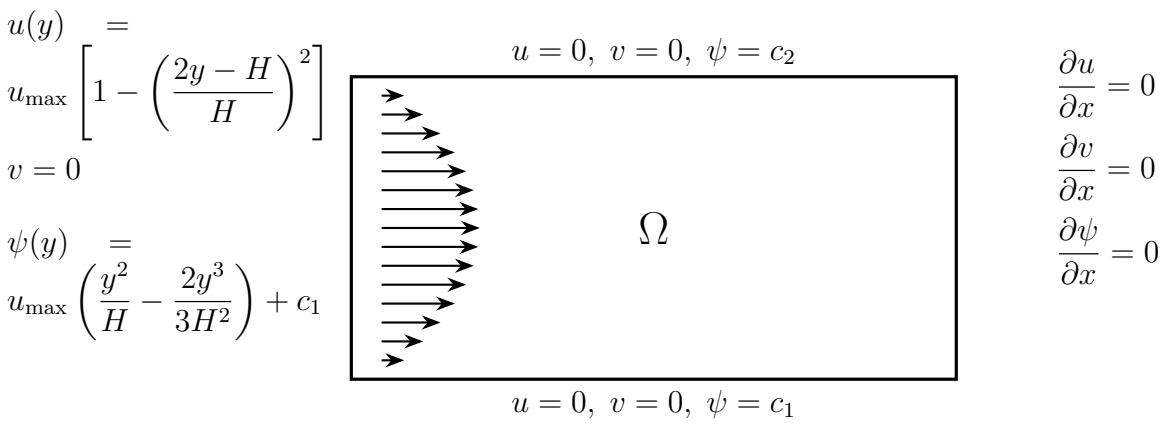


Figura 5.21: Condições de contorno para o escoamento de Hagen–Poiseuille entre placas paralelas.

As soluções analíticas demandam que o escoamento esteja completamente desenvolvido para serem válidas. Dessa forma, considerando a relação do comprimento mínimo de entrada para o desenvolvimento do fluido pela fórmula:

$$L_e = 0,05 \cdot Re \cdot D \tag{5.2}$$

Serão consideradas todas as propriedades do fluido igual a 1 com um número adimensional de Reynolds $Re = 20$. Desta forma, dado um comprimento de entrada de $D = 0.5$, temos que em um comprimento $L_e > 0.5$ permitirá observar um escoamento desenvolvido.

Assim, aplicado o código para diferentes refinos com as condições especificadas,é efetuado a validação do código em conjunto com uma análise de convergência para este caso.

Capítulo 6

Criação de Ferramentas

Conforme citado no capítulo 5, o projeto se utilizou de diversas ferramentas que não necessariamente incluem o algoritmo puro de elementos finitos. Este capítulo explica detalhadamente o motivo e a lógica de cada ferramenta criada ao longo do projeto.

6.1 Geração Automática de perfis

Conforme mencionado no capítulo 5, foi criado uma série de algorítmos que permitem a geração dos arquivos geométricos e posteriormente de malha automaticamente. Como na concepção do projeto era previsto a presença de diveros casos geométricos para comparação, ao invés de desenhar a geometria de cada caso manualmente pelo software Gmsh, a automatização se viabiliza como um método mais eficaz, visto que esta além de reduzir o custo laboral de preparo dos casos, também permite um controle holístico sobre os parâmetros de controle da geometria.

Uma representação do algoritmo geral da geração de um perfil foi gerada conforme a imagem:

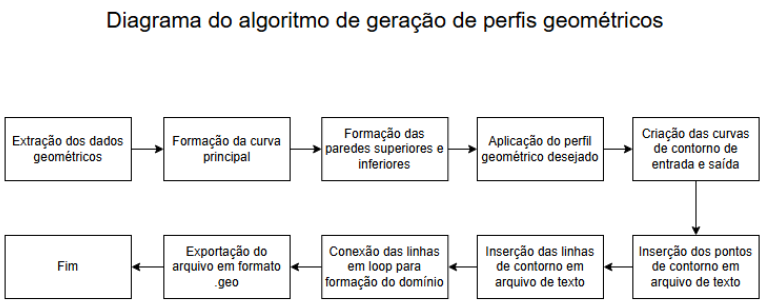


Figura 6.1: Estrutura do Algoritmo de Geração de Perfil.

Nela, conseguimos que temos inicialmente a leitura dos dados de entrada que variam dependendo da forma do perfil para que então seja feita as curvas desejadas. Como foi mencionado no capítulo 5, o método de geração das curvas é a partir de uma curva principal que é utilizada para a geração das curvas referentes às paredes superiores e inferiores.

Para a formação destas curvas, é utilizado geometria diferencial discreta aplicada a curvas parametrizadas. Basicamente, dado uma curva parametrizada no plano:

$$\mathbf{r}(t) = (x(t), y(t)) \tag{6.1}$$

Define-se o vetor tangente à curva é definido como:

$$\mathbf{t}(t) = \frac{d\mathbf{r}}{dt} = \left(\frac{dx}{dt}, \frac{dy}{dt}\right) \tag{6.2}$$

En sua forma discreta, o vetor é escrito como:

$$\mathbf{t} = \frac{(dx, dy)}{\sqrt{dx^2 + dy^2}} \tag{6.3}$$

Por meio do vetor tangente, é obtido o vetor normal unitário por meio de uma rotação 90°. A partir destre vertor normal, define-se a direção de construção das curvas de contorno inferior e superior, de um deslocamento da curva base ao longo da direção normal. Assim, define-se as curvas paramétricas das paredes inferiores e superiores como:

$$\mathbf{r}_{teto}(s) = \mathbf{r}(s) + h \cdot \mathbf{n}(s) \tag{6.4}$$

$$\mathbf{r}_{chao}(s) = \mathbf{r}(s) \tag{6.5}$$

Sendo h a altura do canal especificada no capítulo 5.2.2.

A partir dessa etapa do algoritmo então que há uma separação do processo dependendo do tipo de perfil, sendo um caso específico para perfis com variação periódica e outro com variação aleatória.

6.1.1 Perfis Periódicos

A introdução de rugosidades, na forma de degraus periódicos, é feita a partir dessa parametrização em comprimento de arco. Define-se o comprimento de arco acumulado como:

$$s = \int \sqrt{dx^2 + dy^2} \tag{6.6}$$

No caso discreto:

$$s_i = \sum_{k=1}^{i} \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2} \tag{6.7}$$

Essa escolha tornou-se necessária, pois possibilita definir regiões ao longo da curva com base em distâncias reais, em vez de depender diretamente do parâmetro discreto utilizado na discretização. Assim, torna-se possível identificar trechos específicos da geometria de forma precisa.

Assim, Os perfis periódicos são introduzidos como uma perturbação periódica ao longo do comprimento de arco. Define-se um período espacial:

$$\lambda = d + l \tag{6.8}$$

onde:

- d é o espaçamento entre degraus,
- l é o comprimento do degrau,
- a é a amplitude da perturbação.

Por meio de uma coordenada local dentro de cada período:

$$\xi(s) = s \bmod \lambda \tag{6.9}$$

A geometria modificada é obtida deslocando os pontos na direção normal pela função que caracteriza o perfil desejado:

$$\mathbf{r}(s) = \mathbf{r}_0(s) + \delta(s)\,\mathbf{n}(s) \tag{6.10}$$

No caso da parede inferior, esse deslocamento ocorre no sentido positivo da normal, enquanto no teto ele é aplicado no sentido oposto. Dessa forma, cria-se uma geometria com perturbações alternadas que se projetam para dentro do escoamento, mantendo uma coerência geométrica entre as duas superfícies.

Para definir o número total de estruturas periódicas ao longo da geometria, utiliza-se da relação:

$$N = \left\lfloor \frac{L}{\lambda} \right\rfloor \tag{6.11}$$

Por fim, para garantir qualidade da malha conforme mencionado no capítulo 5.4, define-se um critério baseado na distância entre pontos:

$$\Delta s_i = \|\mathbf{r}_i - \mathbf{r}_{i-1}\| \tag{6.12}$$

Se:

$$\Delta s_i > \frac{a}{3} \tag{6.13}$$

então utiliza-se um tamanho de elemento reduzido por meio de uma redução no parâmetro de refino.

$$h_{\text{local}} = \frac{h}{10} \tag{6.14}$$

6.1.2 Perfis Corrugados

Para o caso corrugado, conforme mencionado previamente, é feito uma introdução de perturbações aleatórias controladas ao longo da direção normal à curva

por meio de funções de geração de números semi-aleatórios. Essa abordagem permite representar superfícies com rugosidade não periódica de forma contínua e suave, evitando descontinuidades geométricas que poderiam comprometer a qualidade da malha numérica ou a veracidade física do caso.

Inicialmente, a partir das curvas bases definidas previamente, aplica-se uma função de suavização do tipo *smoothstep* no trecho inicial da curva para evitar descontinuidades na entrada do domínio

$$\mathbf{r}_0(s) = (x(s), y(s)), \quad s \in [0, L] \tag{6.15}$$

Assim, o trecho inicial da curva é definido como:

$$x(t) = L_{\text{pre}} \cdot t_{\text{smooth}}, \quad y(t) = 0 \tag{6.16}$$

onde L_{pre} representa o comprimento do trecho suavizado. Essa escolha garante continuidade da primeira derivada (C^1), eliminando variações abruptas na geometria. Todavia, quando levamos este trecho para o limite de 0, o código omite a divisão do domínio e segue para a interpolação uniforme.

O restante da curva é construído por interpolação linear uniforme, sendo posteriormente aplicada uma suavização global por meio de um filtro de média móvel:

$$x_i^{\text{novo}} = \frac{1}{2k+1} \sum_{j=i-k}^{i+k} x_j \tag{6.17}$$

A perturbação geométrica é introduzida por meio de um campo de ruído aleatório $\epsilon(s)$, gerado a partir de uma distribuição uniforme:

$$\epsilon(s) \sim \mathcal{U}(-A, A) \tag{6.18}$$

Todavia, foi optado utilizar o valor absoluto do ruído para garantir consistência direcional. Dessa forma, a variação se torna:

$$\delta(s) = |\epsilon(s)| \cdot \alpha \cdot E(s) \tag{6.19}$$

onde α é um fator de amortecimento e $E(s)$ é uma função envelope responsável por controlar a transição do ruído ao longo da geometria.

Esta função envelope teve de ser implementada para evitar descontinuidades exageradas na introdução da rugosidade, o que comprometia a validade da geometria desejada assim como gerava cantos em que o refino de malha necessário para tratar o caso era inviável. Desta forma, define-se uma função envelope baseada em uma interpolação suave:

$$E(s) = \begin{cases} 0, & s < s_0 \\ 3p^2 - 2p^3, & s_0 \leq s \leq s_1 \\ 1, & s > s_1 \end{cases} \tag{6.20}$$

onde:

$$p = \frac{s - s_0}{L_{\text{trans}}} \tag{6.21}$$

Essa função garante a continuidade da primeira derivada, evitando a formação de cantos vivos na geometria. Assim, as perturbações são aplicadas na direção normal à curva base, resultando na geometria modificada:

$$\mathbf{r}_{\text{mod}}(s) = \mathbf{r}_0(s) + \delta(s)\,\mathbf{n}(s) \tag{6.22}$$

onde $\mathbf{r}_0(s)$ representa a curva base original.

6.2 Solucionador de Mecânica dos Fluidos

O solucionador de mecânica dos fluidos apresentado no capítulo 5 possui a seguinte estrutura geral:

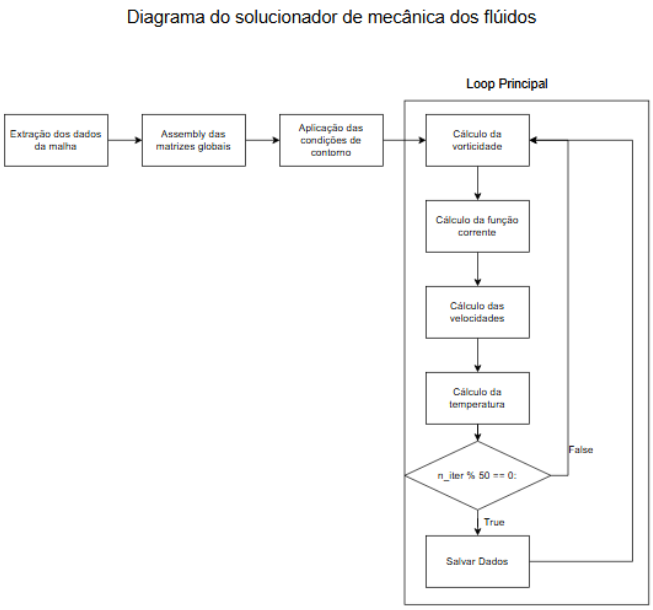


Figura 6.2: Estrutura do código solucionador de mecânica dos fluidos.

Inicialmente, utilizando a biblioteca do meshio, o arquvio da malha é lido pelo código, extraindo as matrizes nodais nos eixos x e y, a IEN e os parâmetros de contorno da IEN. Assim, é formado os arrays referentes à cada região contorno, com esecificicações do tipo de contorno a partir do nome definido no arquivo .geo gerado.

Após a classificação, é montado as matrizes em um processo de assembly, que, por meio das relações geométricas apresentadas no capítulo 4, é formado as matrizes de massa, rigidez e gradiente na direção x e y. Após a geração destas matrizes, é aplicado as condições de contorno às matrizes. Como o problema necessita somente de condições do tipo de Neumann zero e Dirichilet, as alterações nas matrizes são efetuadas nos nós de contorno de modo que, para uma condição de dirichilet, temos que, para uma matriz K qualquer:

1. Zeramento da linha e coluna (exceto o termo diagonal):

$$K_{ij} \leftarrow 0 \quad \forall j \neq i, \qquad K_{ji} \leftarrow 0 \quad \forall j \neq i$$

2. Diagonal principal:

$$K_{ii} \leftarrow 1$$

3. Vetor de forças:

$$F_i \leftarrow g_i$$

e para todos os $j \neq i$:

$$F_j \leftarrow F_j - K_{ji}\, g_i$$

onde K_{ji} é o valor original da matriz (antes do zeramento da coluna).

Enquanto que, para uma condição de Neumann, não é alterado nada na matriz.

Após a geração global das matrizes, então é determinado a velocidade de entrada na direção x conforme o capítulo 5.1.1 enquanto a velocidade em y é mantida igual a 0. A partir disso, o código prossegue criando as matrizes da função corrente e temperatura para as condições de contorno.

Assim, o código aplica as condições de contorno iterando sobre cada array definido previamente conforme a formulação matemática descrita no final do capítulo 3.

Com as matrizes de velocidade, temperatura e função corrente para os nós de contorno definida, é então gerado as matrizes destes mesmos parâmetros que serão utilizados no loop principal de solução. E, por meio da definição da vorticidade descrita no capítulo 2, é determinado também a matriz da vorticidade.

Com a formação inicial do problema, o código segue iterando um mesmo algoritmo que, ao seu fim, atualiza para um novo passo de tempo as novas condições de vorticidade, função corrente, velocidades e temperatura.

Este loop consiste inicialmente no cálculo do termo convectivo da velocidade para então chegar à matriz da vorticidade descrita no cápitulo 3 assim como a matriz de solução da vorticidade. Por meio da solução utilizando a biblioteca scipy, a nova solução da vorticidade ω_z é calculada para então ser aplicada diretamente para o cálculo da nova função corrente ψ. Desta forma, a função ψ atualizada é aplicada para os cálculos das velocidades em x e y que são utilizadas por fim para o cálculo da temperatura, sendo que neste cálculo final o termo convectivo é atualizado antes da formação das matrizes de temperatura descritas no capítulo 3.

Por fim, com a solução completa para este passo de tempo, o código registra e salva os resultados em um arquivo .vtk que é utilizado posteriormente para análise de resultados.

Capítulo 7

Validação do Modelo

Este capítulo apresentará as condições utilizadas no código CFD para validar o modelo em elementos finitos do trabalho. Também será apresentado as referências da literatura utilizadas como base de validação, tomando como base um problema que possua solução analítica unidimensional.

Para analisar o erro numérico do modelo em relação às soluções analíticas, utiliza-se da definição de erro absoluto e relativo de uma variável:

$$e_{abs} = |var_{num} - var_{an}| \tag{7.1}$$

$$e_{rel} = \frac{|var_{num} - var_{an}|}{var_{an}} \tag{7.2}$$

Em que o erro absoluto e_{abs} é o módulo da diferença do valor numérico pelo valor analítico de uma variável e o erro relativo e_{rel} equivale ao valor do erro absoluto divido pelo valor analítico da variável.

Ademais, pode-se definir a qualidade do erro absoluto ou relativo por meio do parâmetro denominado norma do erro. Para este projeto, foi utilizado a norma quadrática para avaliar a validação do código. A norma quadrática é definida por:

$$\|\mathbf{x}\|_2^2 = \sum_{i=1}^{n} x_i^2 = \mathbf{x}^T\mathbf{x} \tag{7.3}$$

7.1 Escoamento de Hagen–Poiseuille

Como é apresentado na literatura de White [3], um dos casos mais simples de escoamento de flúidos é o escoamento de Hagen–Poiseuille. Em um escoamento de-

senvolvido entre placas, temos que a solução analítica unidimensional da velocidade na direção x v_x possui perfil parabólico com forma:

$$v_x(y) = 4 \cdot v_x^{med} \cdot \frac{y}{H} \cdot \left(1 - \frac{y}{H}\right) \tag{7.4}$$

Sendo H a distância entre as placas e v_x^{med} a velocidade média da seção de análise do escoamento.

7.2 Solução Analítica de Temperatura

Segundo Incropera [1], é apresentado diversas soluções analíticas para diferentes condições de escoamento. Considerando que o caso analisado geral consiste em um escoamento entre placas com uma temperatura preescrita na entrada enquanto as 2 paredes possuem temperatura preescita superior à entrada, a solução analítica para o escoamento termicamente desenvolvido também possui perfil polinomial curvo semelhante ao parabólico. Solucionando de forma analítica um escoamento entre placas completamente desenvolvido, obtém-se a seguinte relação:

$$T(\eta) = T_{\text{wall}} + \frac{\mu v_{x,\max}^2}{3k}\left(1 - \eta^4\right) \tag{7.5}$$

Onde η é obtido por meio da definição:

$$\eta = \frac{y - y_c}{h} \tag{7.6}$$

$$y_c = \frac{y_{\max} + y_{\min}}{2} \tag{7.7}$$

7.3 Condições da Validação

Definido as soluções analíticas de temperatura e velocidade escolhidas para validar o modelo computacional desenvolvido, é possível definir no código os parâmetros físicos e geométricos necessários para chegar à solução desejada.

7.3.1 Parâmetros Geométricos

Para avaliar o erro numérico do código com as soluções analíticas, é necessário uma geometria que chegue às condições que permita considerar as hipóteses feitas para chegar à solução analítica. Para chegar às equações de Hagen-Poiseuille em conjunto com a solução analítica da temperatura, o escoamento necessita estar completamente desenvolvido tanto termicamente quanto cinematicamente. Dessa forma, como forma de garantir os efeitos de desevolvimento completo do escoamento, a geometria proposta para validar o código CFD consiste em uma placa plana com uma altura entre placas de $H = 0,5m$ e um comprimento reto de $L = 15m$.

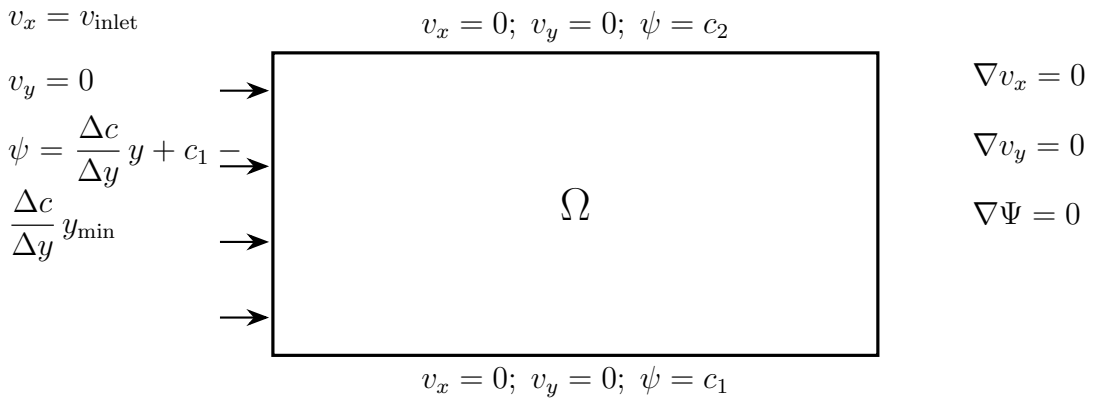


Figura 7.1: Caso Geométrico de Validação do Código CFD.

Para esse tipo de Geometria simples, é possível visualizar e efetuar a construção da malha diretamente no software Gmsh. Nele, inserindo as linhas de contorno que restringem o domínio da simulação, obtém-se:

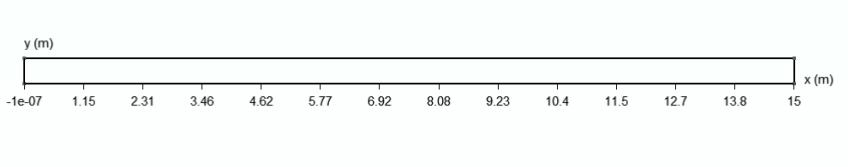


Figura 7.2: Linhas de Contorno do Trecho Reto de Validação no Gmsh.

Efetuando um zoom para a região da entrada, consegue-se verificar o comprimento no eixo y:

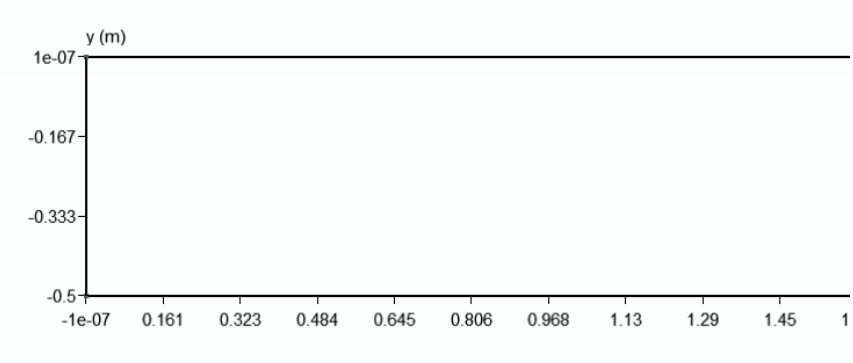


Figura 7.3: Linhas de Contorno na Região de Entrada do Domínio.

7.3.2 Parâmetros Físicos

Para a solução analítica, todos os parâmetros de viscosidade cinemática e difusividade térmica foram definidos como iguais a 1. Desta forma, o comportamento de camada limite é intensificado, permitindo uma verificação melhor dos desvios e erros do código.

Ademais, como as soluções analíticas são válidas somente para escoamentos laminares, para validar o código, foi tomado como velocidade na entrada igual à $v_{inlet} = 40\frac{m}{s}$ para um Reynolds $Re = 20$ devido à conta inversa para o valor da velocidade na entrada apresentada na seção 5.1.1. Dessa forma, o escoamento terá um Reynolds laminar visto que, pela fórmula de Reynolds.

$$Re = \frac{\rho \cdot v_{x,med} \cdot H}{\mu} \tag{7.8}$$

Ademais, para esta validação, será tomado um tempo de simulação de 750 iterações com um passo de tempo de $dt = 0.001s$.

7.3.3 Computador Utilizado

Para a validação do código, foi utilizado um computador do Laboratório de Mecânica dos Flúidos e Aeronáutica (LABMFA) da Universidade Federal do Rio de Janeiro (UFRJ).

O computador possui as seguintes especificações:

- 2x Intel Xeon Silver 4316 com 2.30GHz de frequência
- 256Gb de memória RAM (4x 64Gb RDIMM, Dual Rank)

- Sistema Operacional Ubuntu 22.04.5 LTS
- Compilador Python 3.11

7.3.4 Análise de Convergência

Um teste necessário para verificar a validade dos resultados numéricos com o próprio resultado numérico é a análise de convergência. Nela, é utilizado diferentes versões de um mesmo problema geométrico com diferentes níveis de refino das discretizações. Assim, consegue-se verificar se o modelo funciona corretamente na medida que os parâmetros de erro e norma reduzem conforme a discretização espacial reduz.

Dessa forma, para o caso geométrico de validação do código, foi adotado 3 níveis de refino para verificar a convergência do modelo computacional, sendo cada refinido dotado de um número de nós e elementos diferentes. Para este Trabalho, foi adotado um valor de refino de $refino = 0,05$, $refino = 0,03$ e $refino = 0,01$.

É possível visualizar a diferença que o parâmetro de refino causa pelo software Gmsh. Colocando esses diferentes valores de refino no software, consegue-se assim obter o perfil do domínio.

Figura 7.4: Perfil Geral da Malha com Refino de 0,05.

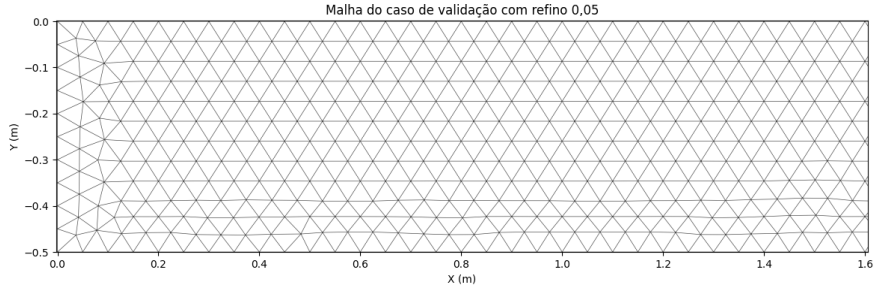


Figura 7.5: Zoom na Malha com Refino de 0,05 na Região de Entrada.

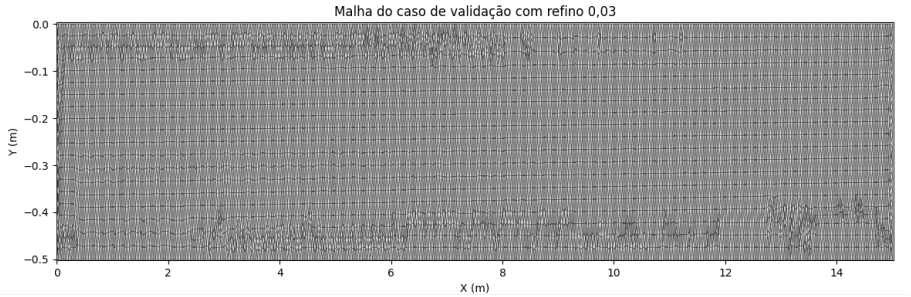


Figura 7.6: Perfil Geral da Malha com Refino de 0,03.

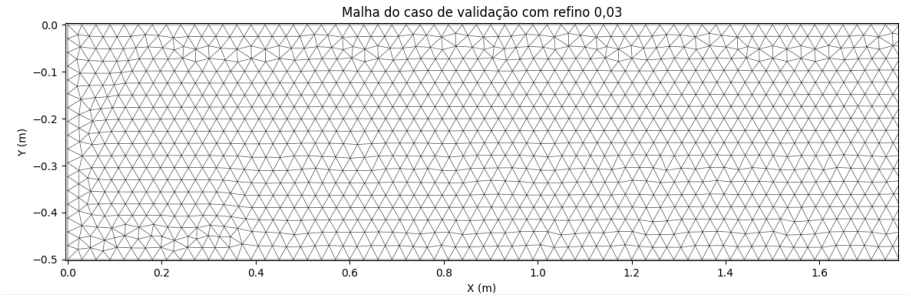


Figura 7.7: Zoom na Malha com Refino de 0,03 na Região de Entrada.

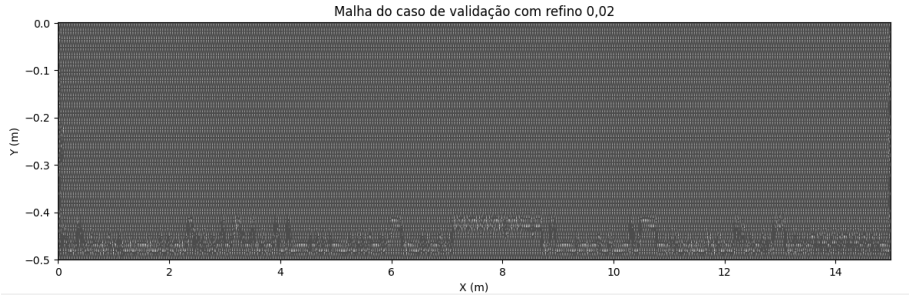


Figura 7.8: Perfil Geral da Malha com Refino de 0,02.

Figura 7.9: Zoom na Malha com Refino de 0,02 na Região de Entrada.

Dessa forma, por meio dessas diferentes malhas geradas, é possível obter a variação do número de nós e elementos para cada nível de refino conforme a tabela abaixo:

Tabela 7.1: Quantidade de elementos e nós em função do refinamento da malha

Identificação da Malha	Caso 1	Caso 2	Caso 3
Refino da malha	0.05	0.03	0.02
Número de nós	3914	10339	22630
Número de elementos	7830	20680	45262

7.4 Resultados

Para a visualização dos resultados, é feita uma avaliação inicial no software ParaView para analisar o formato do perfil encontrado para os parâmetros de velocidade, temperatura, função corrente e vorticidade para assim seguir com a validação com as soluções analíticas.

Será apresentado os resultados para cada nível de refino separadamente para assim efetuar a análise de convergência do modelo computacional.

7.4.1 Validação Caso 1

Abrindo os resultados para a condição de refino de $refino = 0,05$ obtém-se:

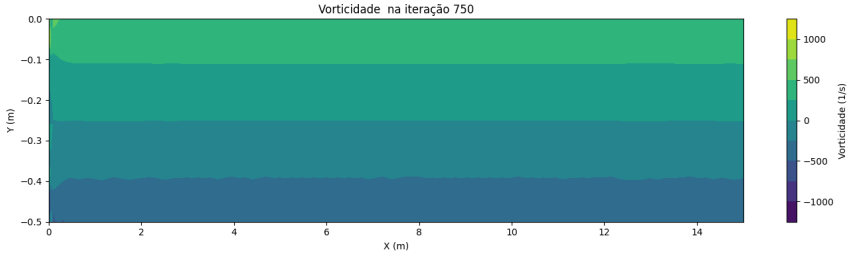


Figura 7.10: Vorticidade para o primeiro caso.

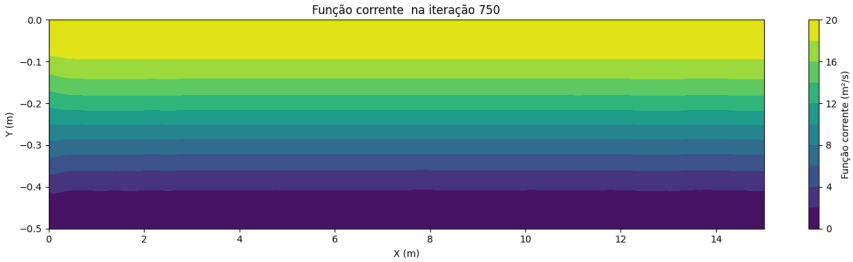


Figura 7.11: Função corrente para o primeiro caso.

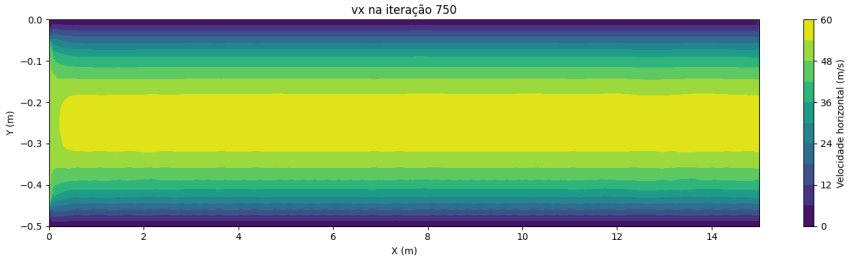


Figura 7.12: Velocidade horizontal para o primeiro caso.

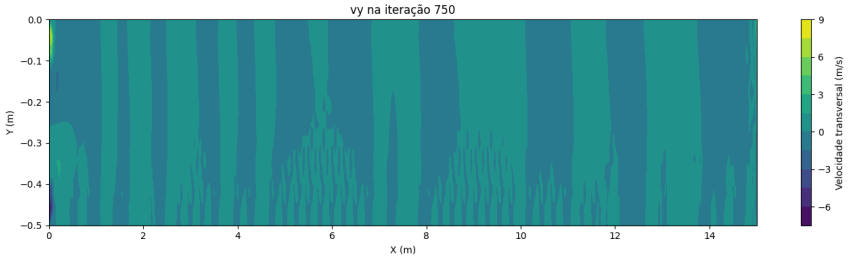


Figura 7.13: Velocidade vertical para o primeiro caso.

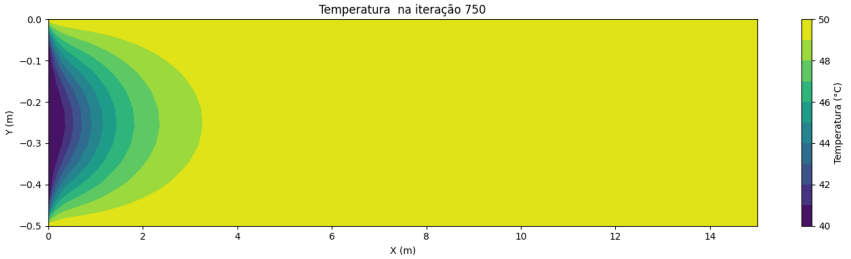


Figura 7.14: Temperatura para o primeiro caso.

Devido à altura pequena do domínio em relação ao seu comprimento, visualizar o domínio em sua totalidade resulta em uma perda de análise e resultados interessantes. Assim, para visualizar o desenvolvimento do escoamento, foi visto também no Para-View a região de entrada até o comprimento em que há o desenvolvimento dinâmico da simulação

Na entrada do escoamento, obtém-se os seguintes resultados:

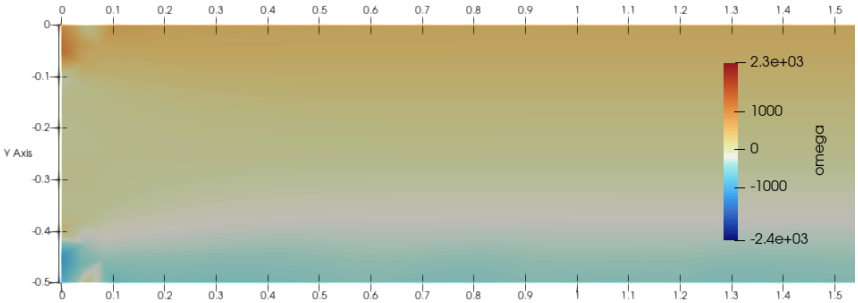


Figura 7.15: Vorticidade na entrada do domínio para o primeiro caso.

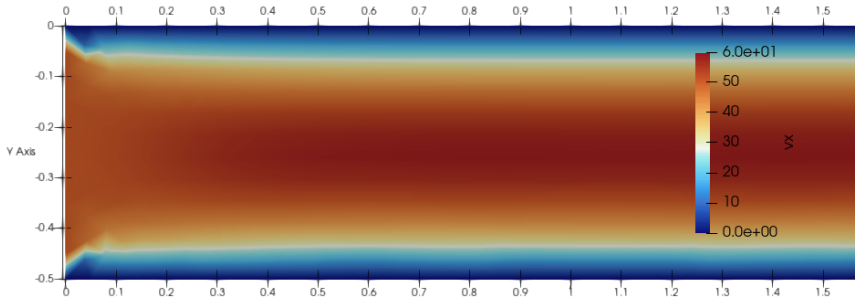


Figura 7.16: Velocidade horizontal na entrada do domínio para o primeiro caso.

Percebe-se que a simulação rapidamente desenvolve o seu escoamento para um perfil em que há a intensificação da velocidade na região média entre as placas. Ademais, é possível vizualizar justamente o efeito de camada limite cinemática da literatura. Ademais, percebe-se que há um salto não suave dos valores de velocidade nos pontos de contorno para os primeiros pontos adjacentes, que é justificado pelo tamanho dos elementos.

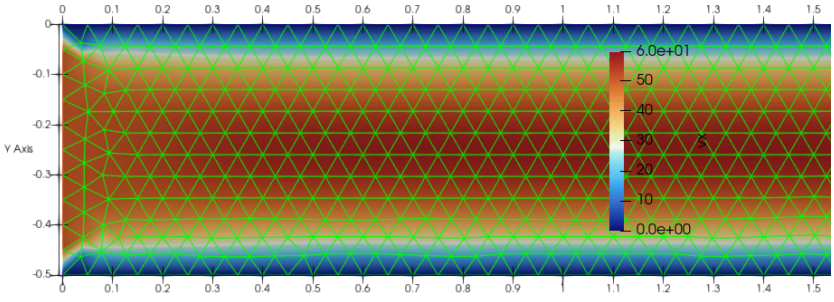


Figura 7.17: Velocidade horizontal com divisão de elementos na entrada do domínio para o primeiro caso.

Observando a velocidade vertical, obtém-se na entrada:

Consegue-se visualizar o mesmo comportamento não suave presente na entrada

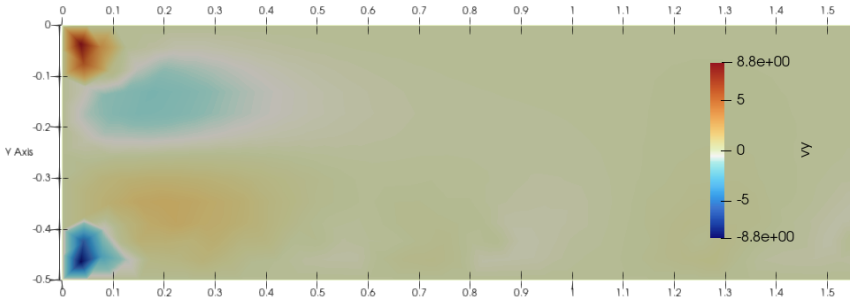


Figura 7.18: Velocidade vertical na entrada do domínio para o primeiro caso.

da velocidade horizontal, mas que rapidamente cede e com o desenvolvimento do escoamento, a velocidade no eixo y torna-se nula.

Para a temperatura, obtém-se:

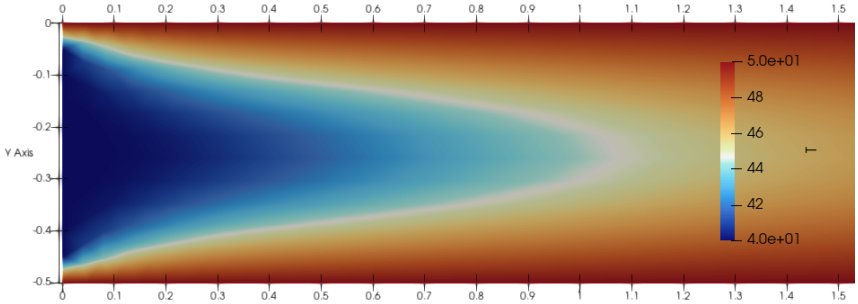


Figura 7.19: Temperatura na entrada do domínio para o primeiro caso.

Consegue-se ver essa curva que vai sendo afinada de modo semelhante à camada limite cinemática apresentada nas imagens da velocidade horizontal. Todavia, com o desenvolvimento do escoamento, a temperatura final tende à ser igual à temperatura nas paredes, como mostra a imagem no final do domínio:

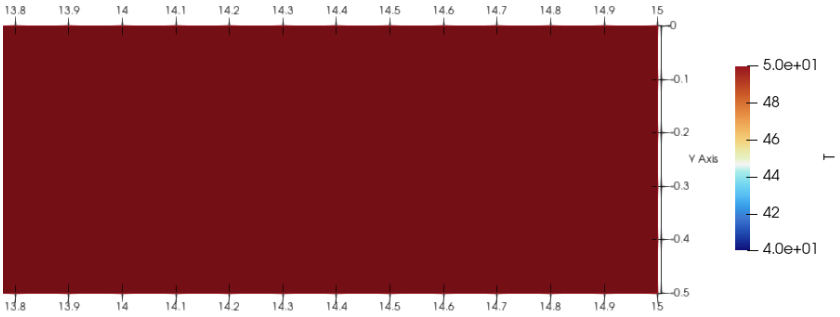


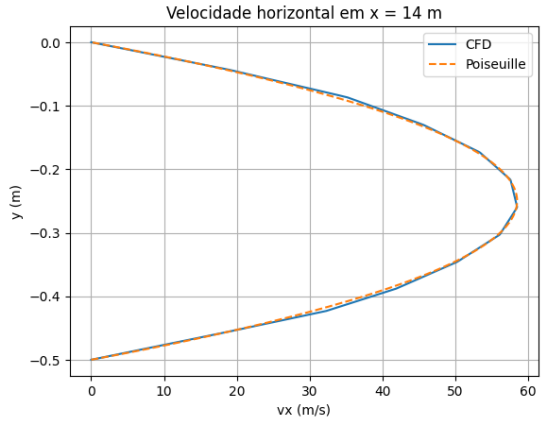
Figura 7.20: Temperatura na saída do domínio para o primeiro caso.

Nessa região, as diferenças de temperatura no eixo y são pequenas comparadas

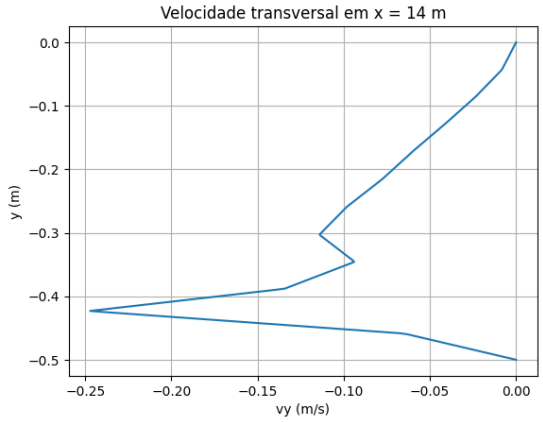
à entrada, porém próximas das soluções analíticas apresentadas.

Para comparar os resultados obtidos com as soluções analíticas, toma-se como referência os pontos na região em que $x = 14m$, visto que nessa região tanto a temperatura quanto as velocidades estão praticamente desenvolvidas.

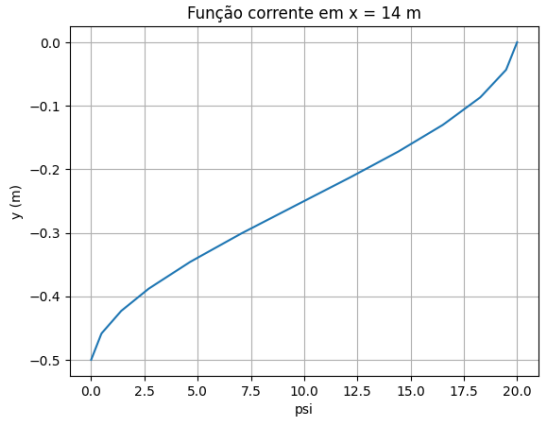
Plotando os resultados utilizando o código de validação é obtido os seguintes gráficos:



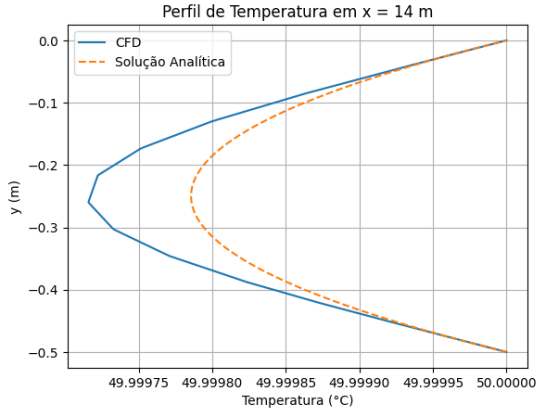
(a) Velocidade horizontal.



(b) Velocidade transversal.



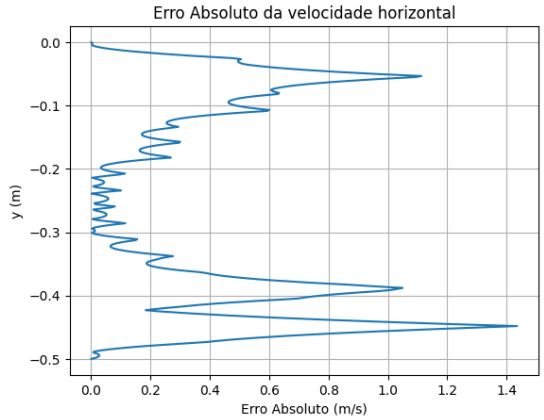
(c) Função corrente.



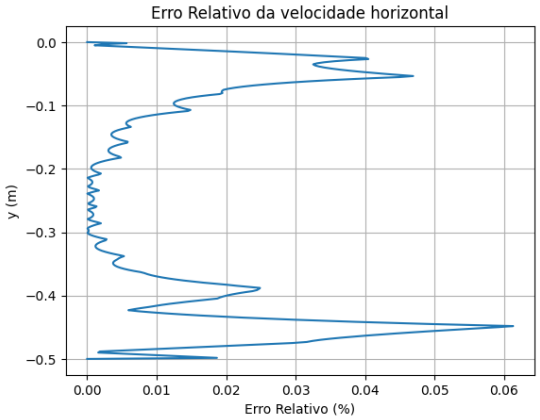
(d) Temperatura.

Figura 7.21: Comparação entre as soluções numérica e analítica em $x = 14\,\mathrm{m}$ para o primeiro caso.

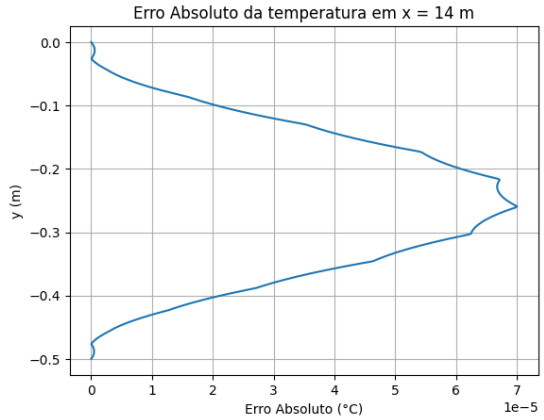
Por fim, gerando calculando os erros relativos e absolutos para os gráficos gerados, obtém-se:



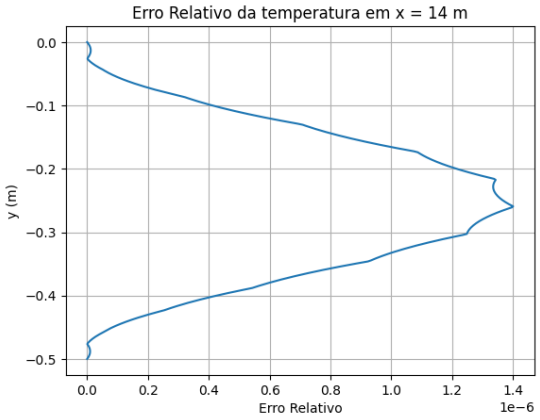
(a) Erro absoluto da velocidade horizontal.



(b) Erro relativo da velocidade horizontal.



(c) Erro absoluto da temperatura.



(d) Erro relativo da temperatura.

Figura 7.22: Erros absolutos e relativos das variáveis avaliadas em $x = 14\,\text{m}$ para o primeiro caso.

Por fim, por meio das fórmulas de norma, obtém-se uma norma do erro absoluto e relativo para as velocidades e temperatura representado pela tabela abaixo:

Tabela 7.2: Normas dos erros para velocidade e temperaturapara o primeiro caso.

Parâmetro	Velocidade	Temperatura
Norma Erro Absoluto	0.6084 m/s	0.000049 $°C$
Norma Erro Relativo	0.02531%	0.0001%

7.4.2 Validação Caso 2

Abrindo os resultados para a condição de refino de $refino = 0,03$ obtém-se:

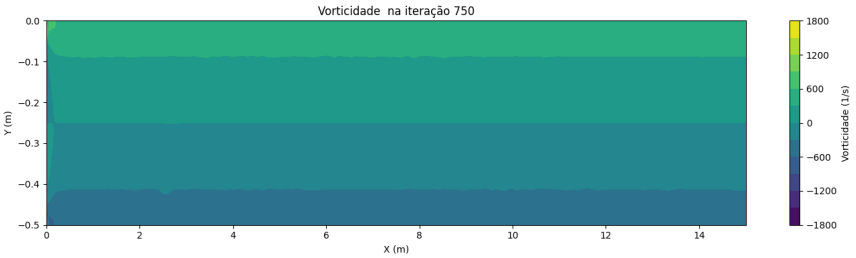


Figura 7.23: Mapa de calor da vorticidade para o segundo caso

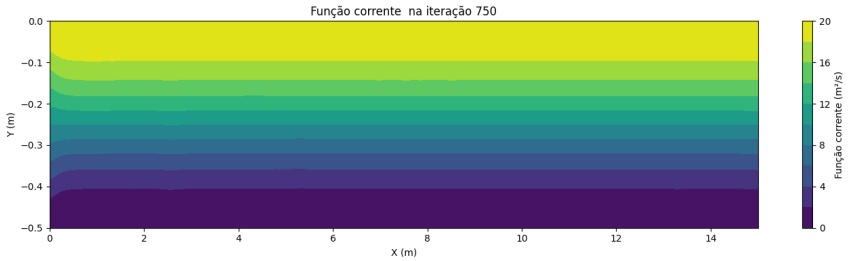


Figura 7.24: Função corrente para o segundo caso.

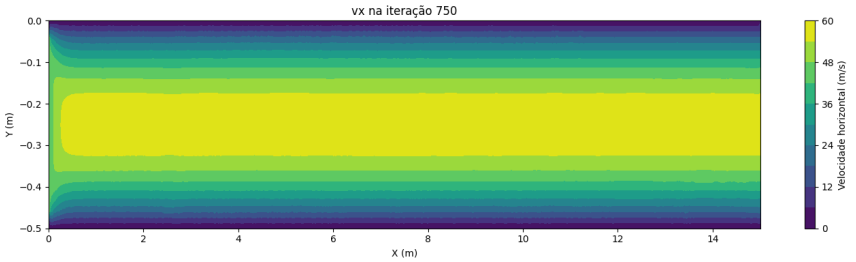


Figura 7.25: Velocidade horizontal para o segundo caso.

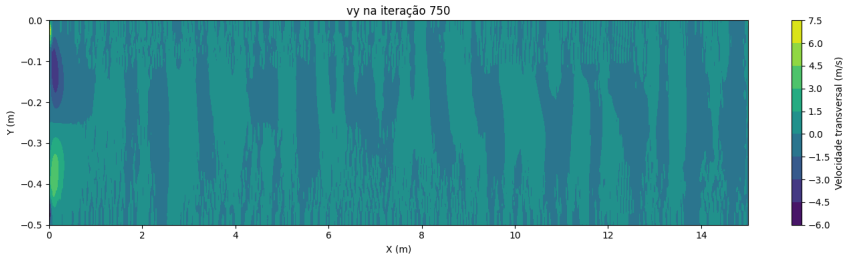


Figura 7.26: Velocidade vertical para o segundo caso.

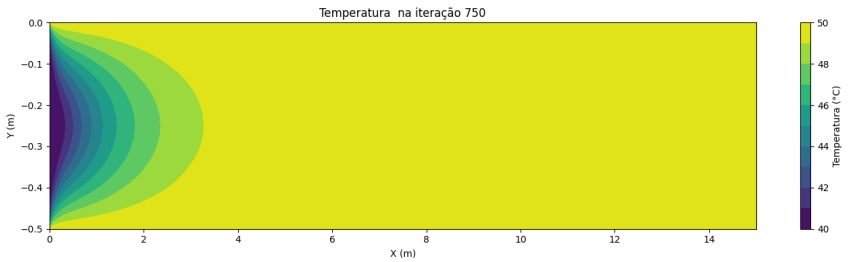


Figura 7.27: Temperatura para o segundo caso.

Aplicando a mesma visualização na entrada feita na seção 7.4.1, é obtido:

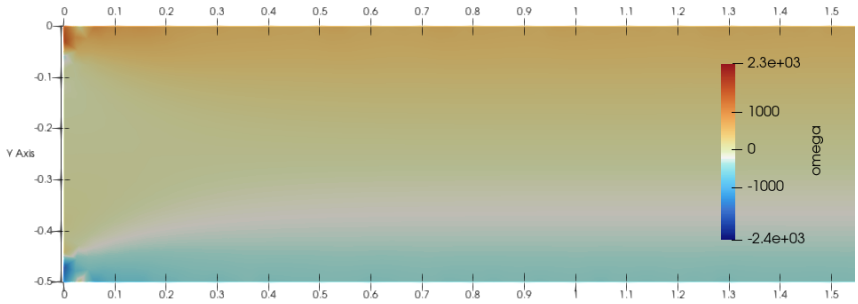


Figura 7.28: Vorticidade na entrada do domínio para o segundo caso.

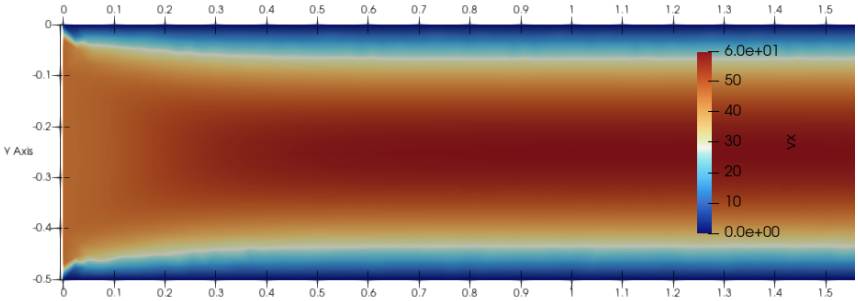


Figura 7.29: Velocidade horizontal na entrada do domínio para o segundo caso.

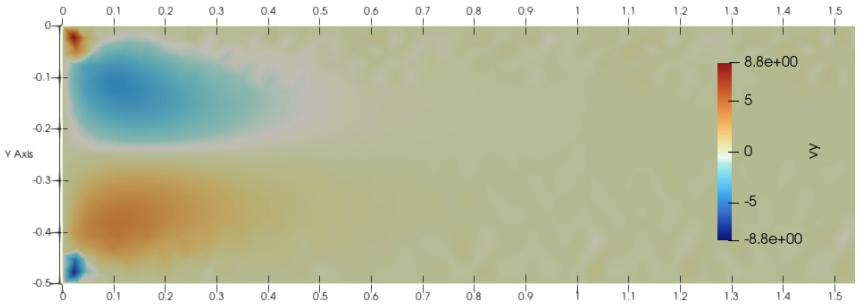


Figura 7.30: Velocidade vertical na entrada do domínio para o segundo caso.

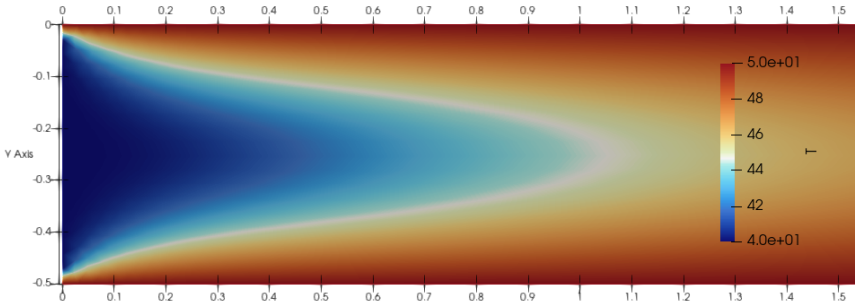
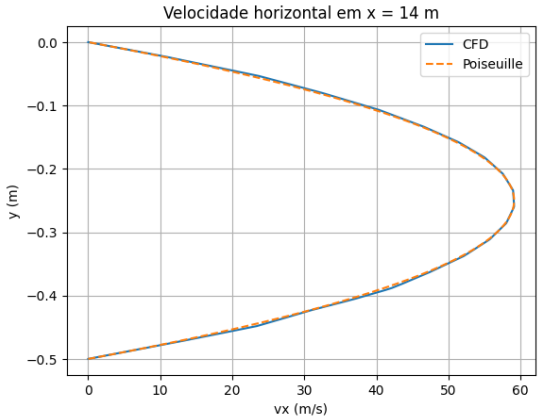


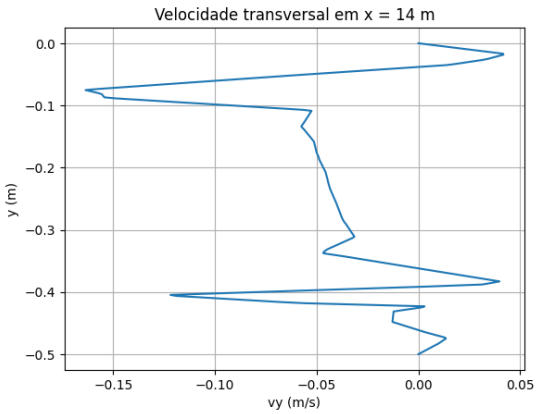
Figura 7.31: Temperatura na entrada do domínio para o segundo caso.

Nele, consegue-se ver que os efeitos de mudança de velocidade bruscos presente na entrada do caso menos refinado são reduzidos devido ao maior número de elementos na entrada. Especialemnte, em relação à velocidade transversal, nota-se que a região com alta intensidade nos nós próximos aos nós de contorno de parede superior e inferior com os nós de contorno da entrada do escoamento são menores se comparado com o primeiro caso.

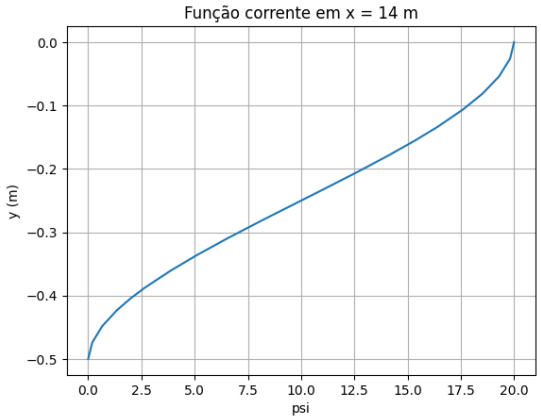
Aplicando o resultado da última iteração no código de validação, os gráficos resultantes são:



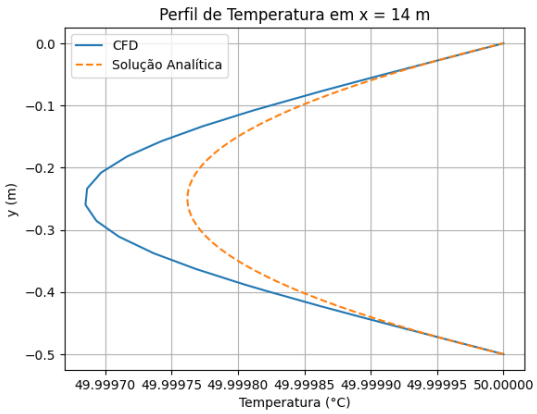
(a) Velocidade horizontal.



(b) Velocidade transversal.



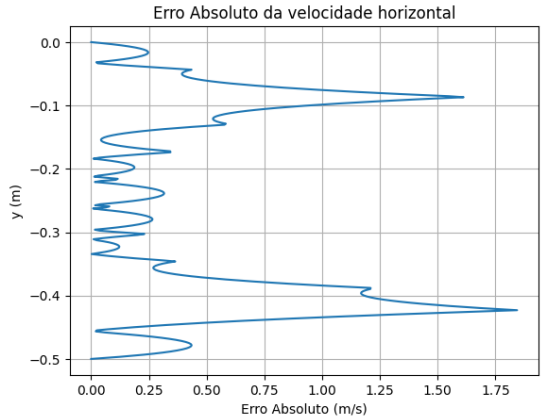
(c) Função corrente.



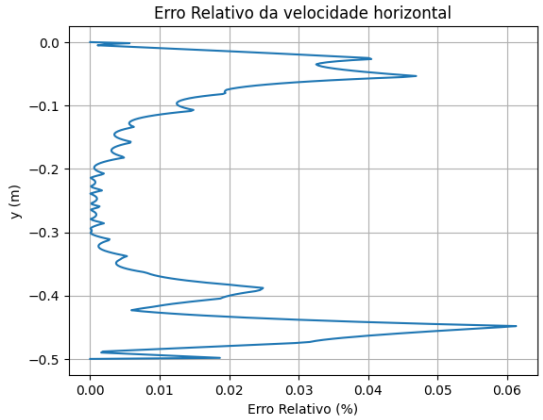
(d) Temperatura transversal.

Figura 7.32: Comparação entre as soluções numérica e analítica em $x = 14\,\text{m}$ para o segundo caso.

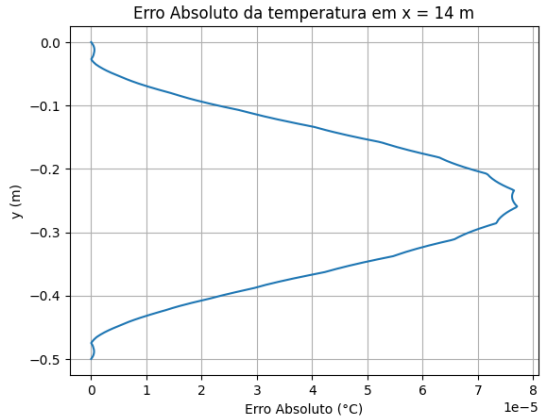
Da mesma forma, calcula-se os erros absolutos e relativos para o segundo caso:



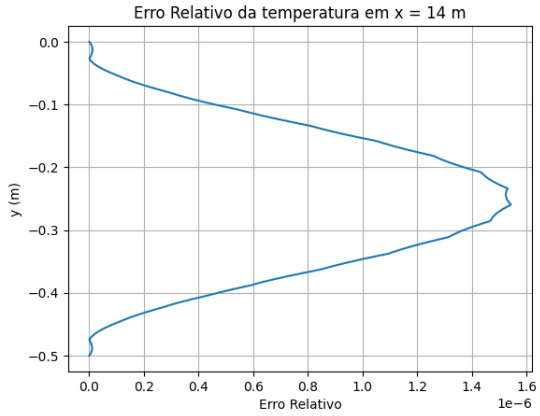
(a) Erro absoluto da velocidade horizontal.



(b) Erro relativo da velocidade horizontal.



(c) Erro absoluto da temperatura.



(d) Erro relativo da temperatura.

Figura 7.33: Erros absolutos e relativos das variáveis avaliadas em $x = 14\,\text{m}$ para o segundo caso.

Da mesma forma, calculando a norma do erro para o segundo caso, a tabela geral é construída:

Tabela 7.3: Normas dos erros para velocidade e temperatura para o segundo caso.

Parâmetro	Velocidade	Temperatura
Norma Erro Absoluto	0.6083 m/s	0.000048 $°C$
Norma Erro Relativo	0.02531%	0.0001%

7.4.3 Validação Caso 3

Abrindo os resultados para a condição de refino de $refino = 0,02$ obtém-se:

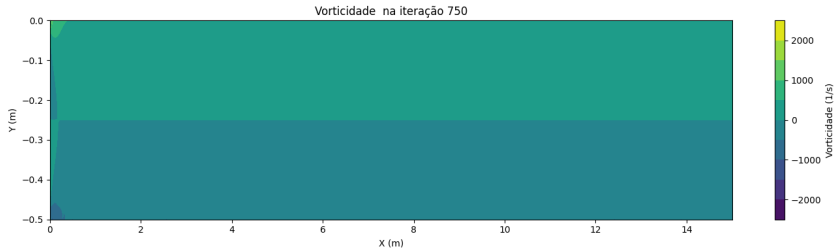


Figura 7.34: Vorticidade para o terceiro caso.

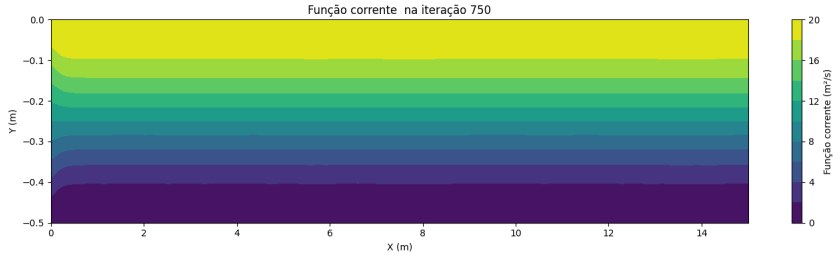


Figura 7.35: Função corrente para o terceiro caso.

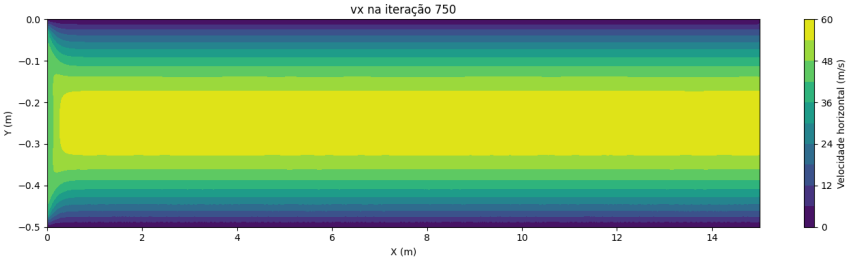


Figura 7.36: Velocidade horizontal para o terceiro caso.

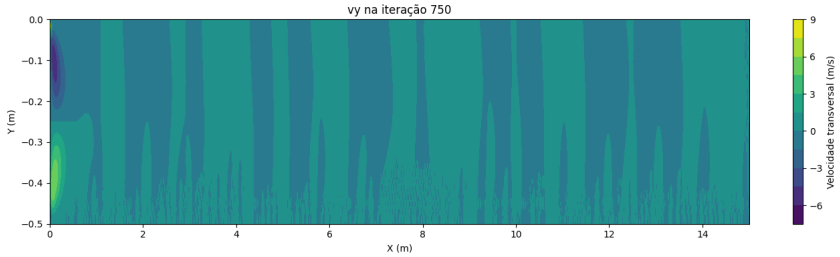


Figura 7.37: Velocidade vertical para o terceiro caso.

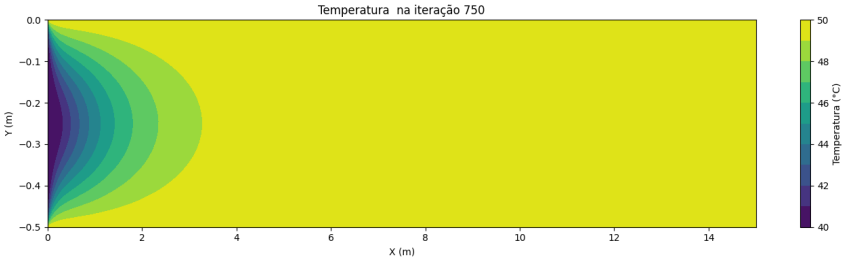


Figura 7.38: Temperatura para o terceiro caso.

Da mesma forma que os casos apresentados previamente, é feita uma visualização na entrada do domínio:

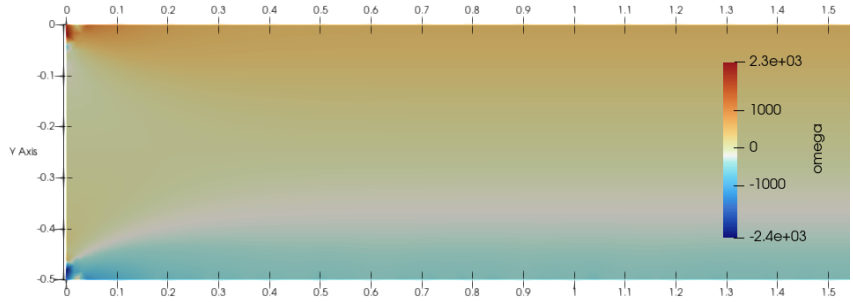


Figura 7.39: Vorticidade na entrada do domínio para o terceiro caso.

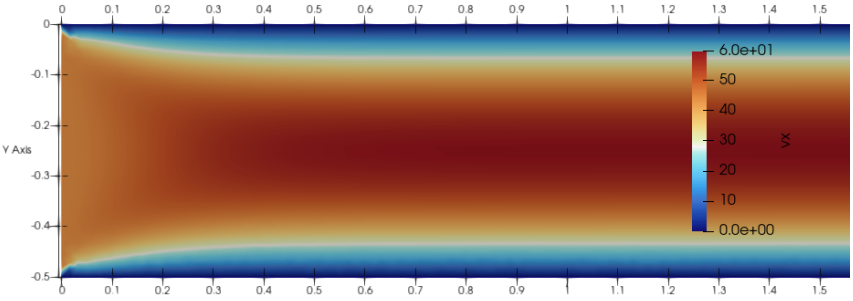


Figura 7.40: Velocidade horizontal na entrada do domínio para o terceiro caso.

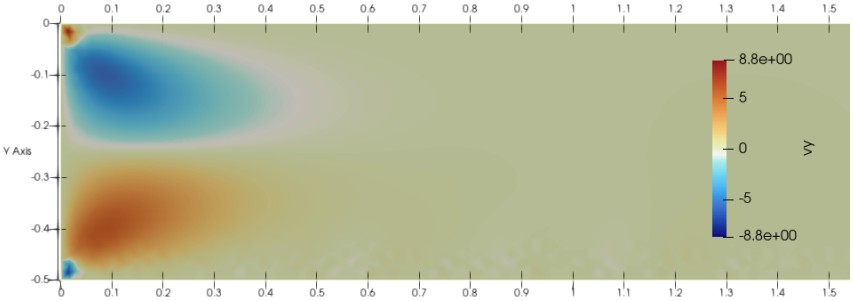


Figura 7.41: Velocidade vertical na entrada do domínio para o terceiro caso.

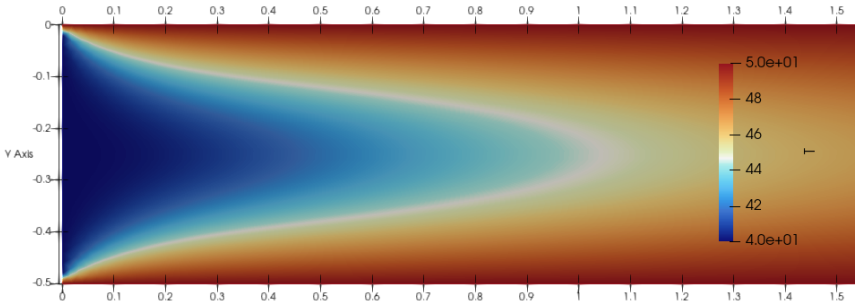
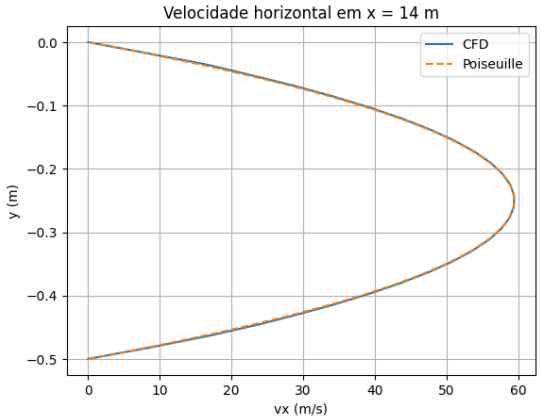
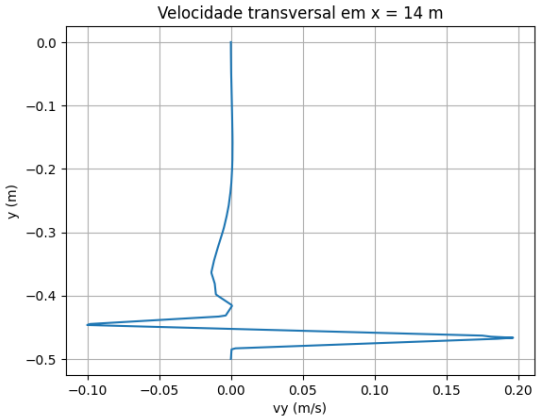


Figura 7.42: Temperatura na entrada do domínio para o terceiro caso.

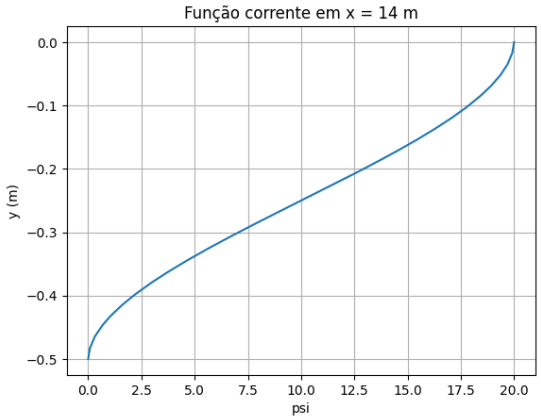
Assim como no segundo caso, as mudanças do terceiro caso para o segundo são intensificadas, com um perfil mais suave na udança de velocidade na direção horizontal enquanto as velocidades transversais possuem as regiões de alta intensidade cada vez mais reduzidas. Dessa forma, avaliando os resultados gráficos do terceiro para a faixa de $x = 14m$:



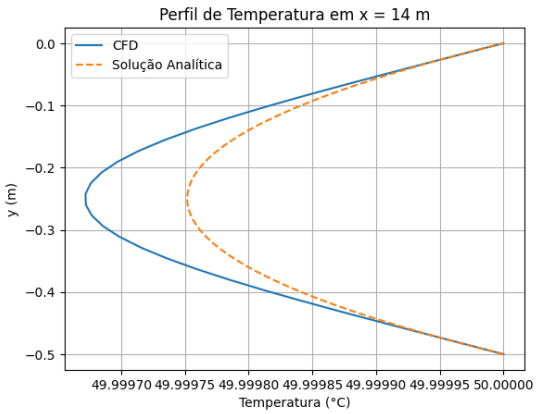
(a) Velocidade horizontal.



(b) Velocidade transversal.



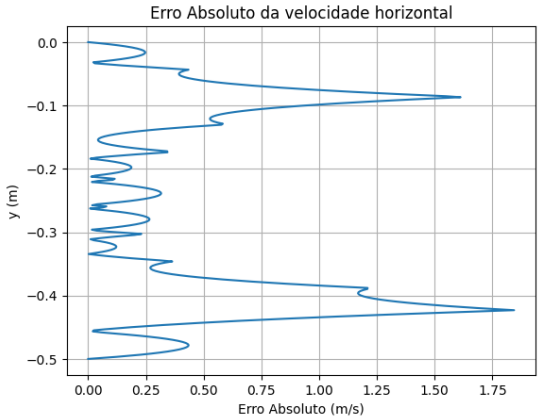
(c) Função corrente.



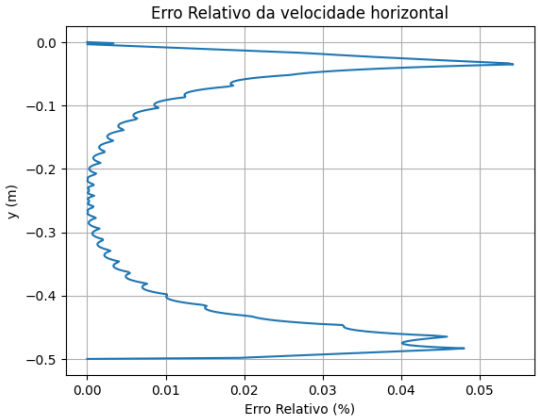
(d) Temperatura.

Figura 7.43: Comparação entre as soluções numérica e analítica em $x = 14\,\mathrm{m}$ para o terceiro caso.

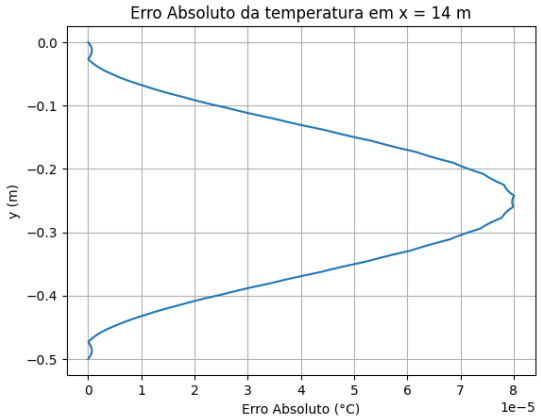
Da mesma forma, calcula-se os erros absolutos e relativos para o segundo caso:



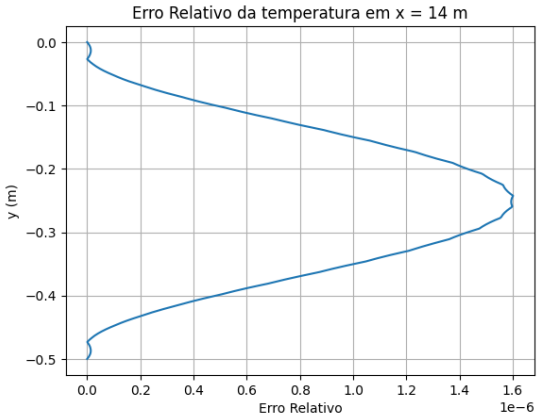
(a) Erro absoluto da velocidade horizontal.



(b) Erro relativo da velocidade horizontal.



(c) Erro absoluto da temperatura.



(d) Erro relativo da temperatura.

Figura 7.44: Erros absolutos e relativos das variáveis avaliadas em $x = 14\,\text{m}$ para o terceiro caso.

Assim, é construída a tabela de normas para o terceiro caso:

Tabela 7.4: Normas dos erros para velocidade e temperatura para o terceiro caso.

Parâmetro	Velocidade	Temperatura
Norma Erro Absoluto	0.6083 m/s	0.000048 $°C$
Norma Erro Relativo	0.02531%	0.0001%

7.4.4 Convergência dos Resultados

Analisando os resultados de norma obtidos para a velocidade e a temperatura, verifica-se que com o aumento do refino da malha, os resultados tornaram-se mais

precisos. Todavia, devido ao tempo de simulação e as condições físicas que controlam a simulação, a norma dos erros absolutos e relativos teve pouca variação para o tempo analisado.

Dessa forma, pode-se concluir que há uma convergência dos resultados para todos os casos de validação, porém, o refino aplicado para o segundo caso é suficiente para obter um resultado satisfatório numericamente.

Capítulo 8

Resultados e Discussões

Neste capítulo, serão apresentados os resultados numéricos dos casos geométricos apresentados no capítulo 5, e a partir disso, será feita uma análise comparativa dos resultados com a configuração perfeitamente plana também apresentada na metodologia.

Para cada tipo de perfil periódico, serão avaliadas 2 configurações de distância entre o perfil parametrizado pela variável l. Dessa forma, poderá ser visualizado também o efeito que a distância entre os perfis ocasiona no campo de velocidades, como no transporte de energia do escoamento.

Para os casos corrugados, será analisado uma configuração completamente variada pelas funções de ruído para verificar o comportamento térmico e dinâmico assim como um caso com um trecho prévio perfeitamente plano para avaliar possíveis mudanças cinemáticas do perfil devido à mudança do perfil de velocidade na entrada do escoamento.

Assim como foi apresentado no capítulo 7, o computador que realizou as simulações numéricas apresentadas neste capítulo de resultados possui as seguintes configurações:

- 2x Intel Xeon Silver 4316 com 2.30GHz de frequência
- 256Gb de memória RAM (4x 64Gb RDIMM, Dual Rank)
- Sistema Operacional Ubuntu 22.04.5 LTS
- Compilador Python 3.11

Para efetuar a análise dos resultados, será considerado além dos valores gerais de cada parâmetro analisado ao longo do domínio os resultados obtidos em uma seção de geometria que esteja próxima dos nós de saída. Como a geometria de todos os casos apresentados possui um comprimento plano de $L_{reto} = 0,1m$, será tomado os valores na posição $x_{data} = 0.0999m$ para este projeto.

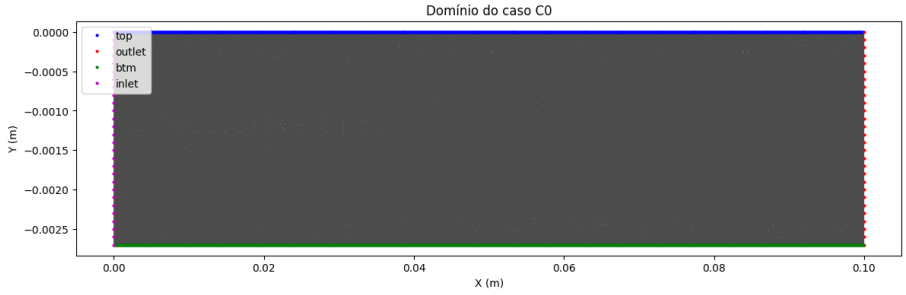
8.1 Perfil Plano

Para perfil reto, temos as características geométricas referenciadas no capítulo 5:

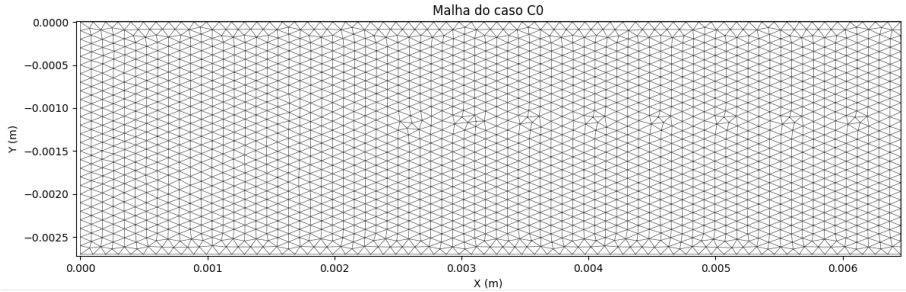
Tabela 8.1: Parâmetros Caso Plano.

Caso	Tipo de Perfil	n_{pontos}	$L_{\text{reto}}(m)$	offset (m)	$d(m)$	$l(m)$	$a(m)$	tm	tmr	Entrada Limpa
C0	Plano	100	0,1	0,0027	0	0	0	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0

Nele, temos uma geometria próxima do caso usado para validar o código computacional, com a diferença sendo as diferenças de comprimento sendo igual à $L_{reto} = 0,1m$, a distância entre placas igual à $H = offset = 27mm$. Dessa forma, formando a malha no Gmsh e visualizando a forma dos elementos, obtém-se os gráficos:



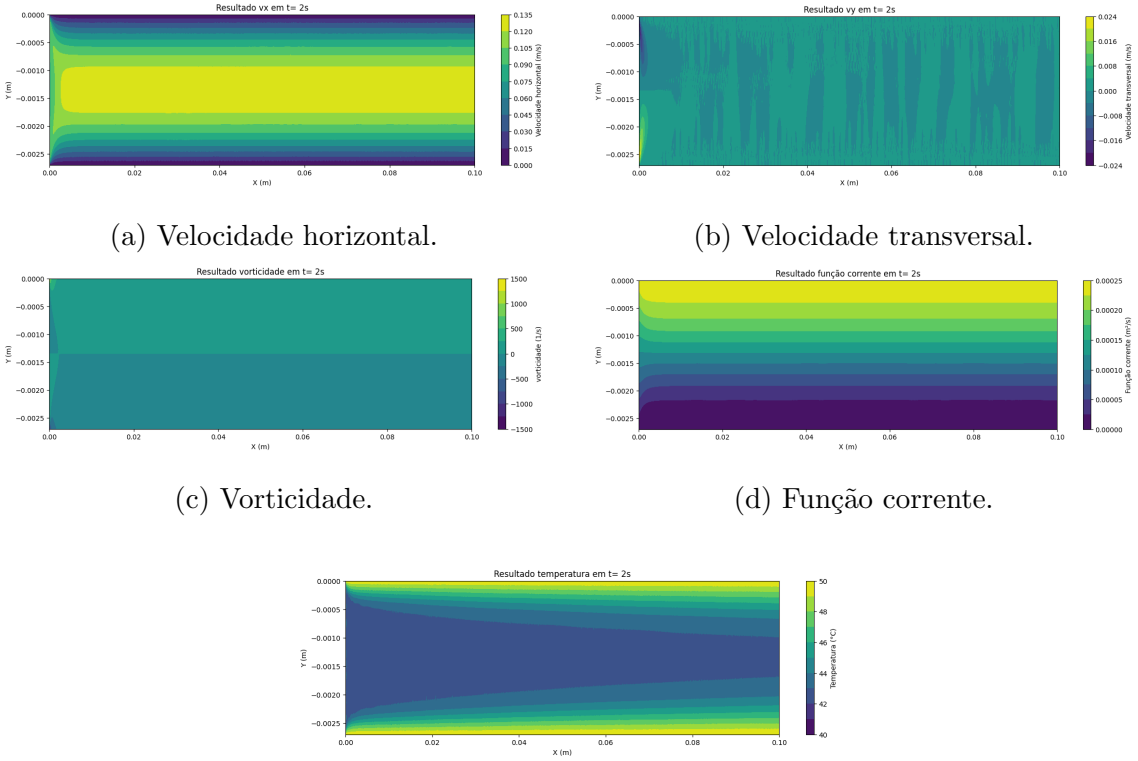
(a) Domínio do caso C0 em conjunto com os nós de contorno.



(b) Elementos triangulares do caso C0.

Devido à geometria simples do problema, Percebe-se que os triangulos formados são em sua maioria equiláteros, o que torna a qualidade da simulação superior. A malha utilizada para simular o caso plano possui um total de 32898 nós formando um total de 65794 elementos.

Simulando para o tempo final de $t_f = 2s$ especificado na metodologia, obtém-se os seguintes resultados de temperatura, velocidades, vorticidade e função corrente.



(a) Velocidade horizontal.

(b) Velocidade transversal.

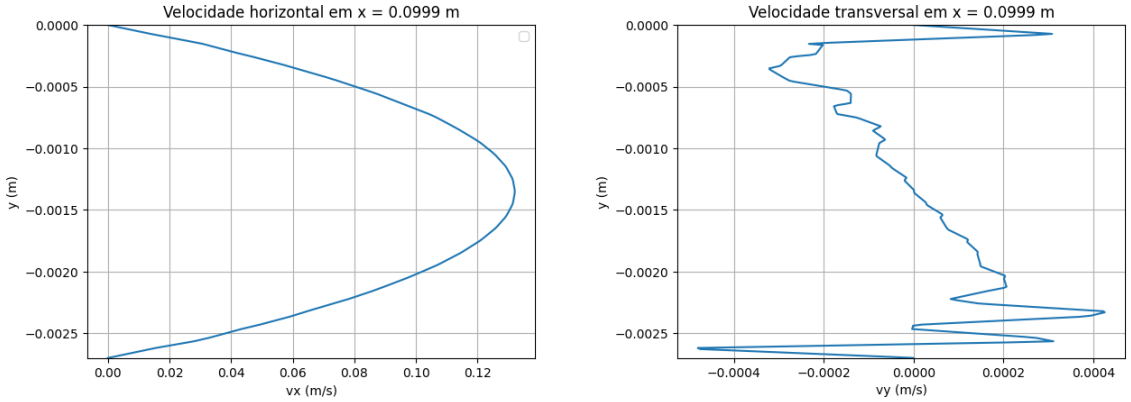
(c) Vorticidade.

(d) Função corrente.

(e) Temperatura.

Figura 8.2: Resultados obtidos para o caso C0 no tempo $t_f = 2s$.

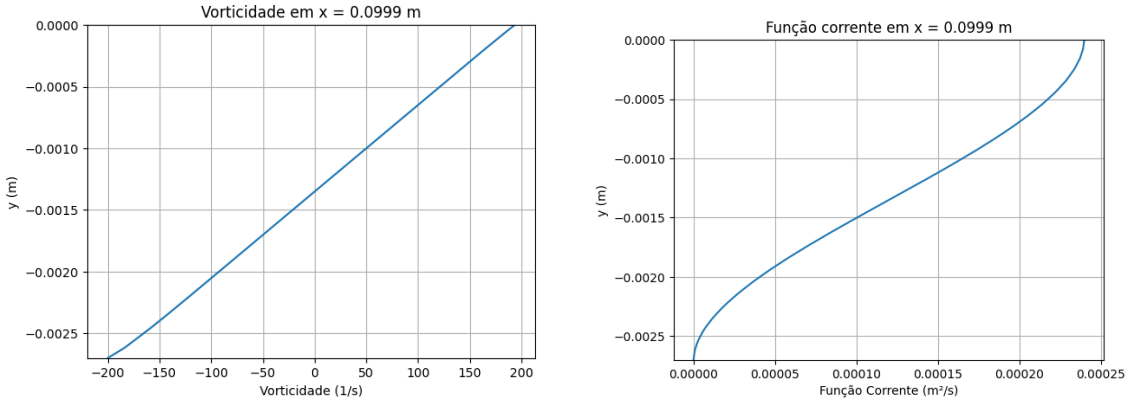
Percebe-se que o comportamento é semelhante ao resultado obtido para efetuar a validação do código computacional, visto que ambos possuem uma geometria próxima. Assim, analisando os resultados obtidos na seção especificada de $x_{data} = 0.0999m$, obtém-se:



(a) Velocidade horizontal.

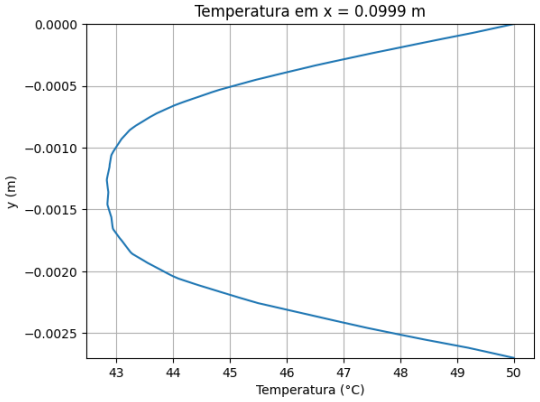
(b) Velocidade transversal.

Figura 8.3: Campo de velocidades horizontais e transversais para o caso C0 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.



(a) Vorticidade.

(b) Função corrente.



(c) Temperatura.

Figura 8.4: Vorticidade, função corrente e temperatura para o caso C0 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.

Analisando os gráficos resultantes,percebe-se justamente para os parâmetros de velocidades, função corrente e vorticidade o comportamento esperado e semelhante ao escoamento de placas planas utilizado para validar o modelo computacional.

Em relação à temperatura, percebe-se que o gráfico para o caso plano é relativamente diferente comparado à simulação de validação. Isso se deve à influência dos parâmetros de difusividade térmica, já que, considerando que o número de Prandtl do flúido de trabalho é de $Pr = 66,24$, o desenvolvimento da camada limite térmica ocorrerá demasiadamente posterior comparado ao caso de validação, onde o número de Prandtl era igual a 1.

8.2 Perfil Trapezoidal

Para perfil trapezoidal, foram avaliados 2 condições geométricas. Sendo estas referenciadas no capítulo 5 como os casos C1 e C6:

Tabela 8.2: Parâmetros Caso Trapezoidal.

Caso	Tipo de Perfil	n_{pontos}	$L_{\text{reto}}(m)$	offset (m)	$d(m)$	$l(m)$	$a(m)$	tm	tmr	Entrada Limpa
C1	Trapezoidal	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0
C6	Trapezoidal	500	0,1	0,0027	0,005	0,001	0,00027	$2,00 \times 10^{-4}$	$2,00 \times 10^{-5}$	0

A diferença principal entre os casos advém do número de pontos e do parâmetro l que, conforme apresentado na metodologia, controla a distância entre os perfis avaliados.

Gerando as malhas para ambos os casos C1 e C6, define-se o domínio computacional utilizado para simular os casos:

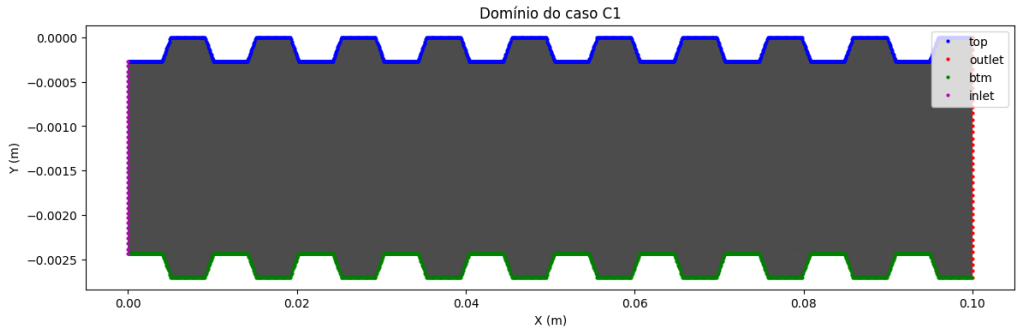
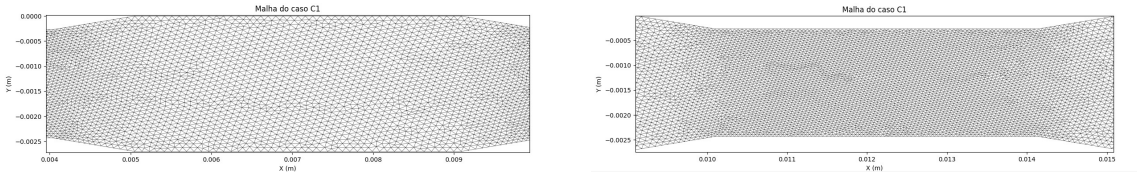


Figura 8.5: Domínio do caso C1 em conjunto com os nós de contorno.



(a) Região não refinada. (b) Região refinada.

Figura 8.6: Diferentes níveis de refino para o domínio do caso C1.

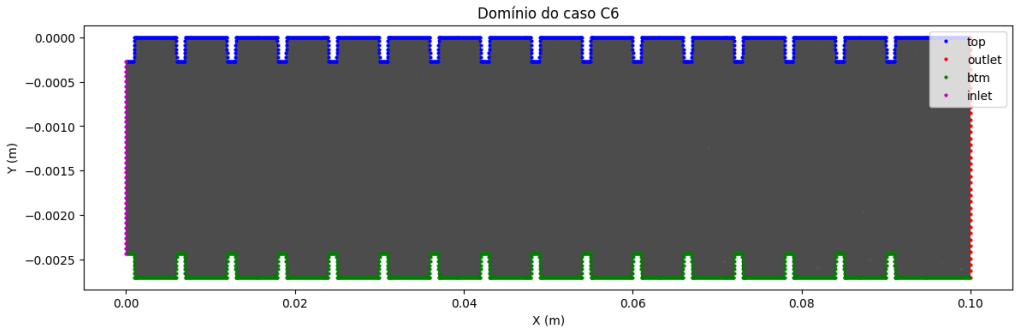
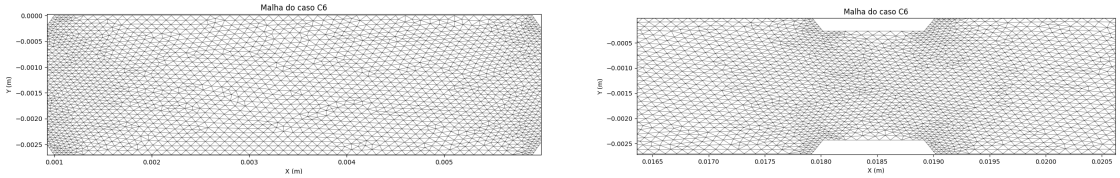


Figura 8.7: Domínio do caso C6 em conjunto com os nós de contorno.



(a) Região não refinada. (b) Região refinada.

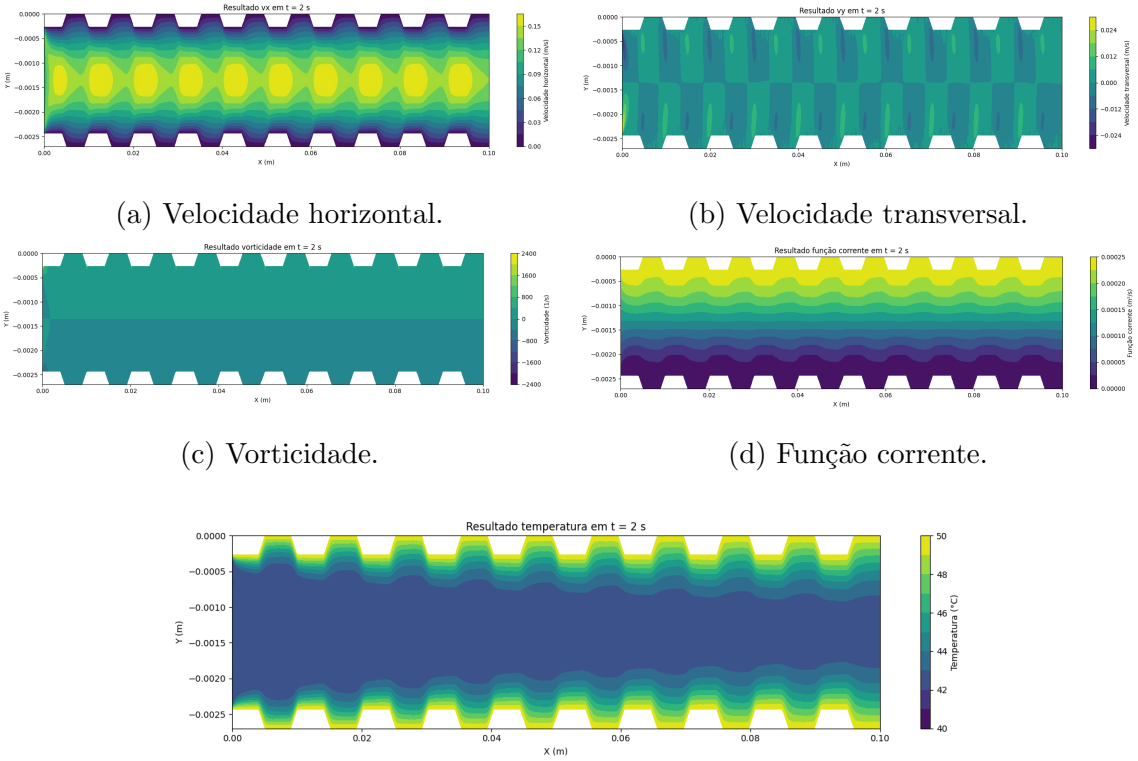
Figura 8.8: Diferentes níveis de refino para o domínio do caso C6.

Devido há mudana brusca de geometria ao longo dos nós de contorno das paredes superiores e inferiores, a implementação do algoritmo de refino mencionado nos capítulos 5 e 6 gera uma região com um refino de malha nos pontos em que ocorre a mudança de altura. Dessa forma, consegue-se visualizar os efeitos das singularidades geométricas com melhor precisão.

Para o caso C1, a malha gerada possui 72570 nós e 145138 elementos, enquanto que para o caso C6, a malha possui 31989 nós e 63976 elementos.

8.2.1 Caso C1

Apresentando os resultados obtidas para a simulação do caso C1 graficamente, é obtido:



(a) Velocidade horizontal.

(b) Velocidade transversal.

(c) Vorticidade.

(d) Função corrente.

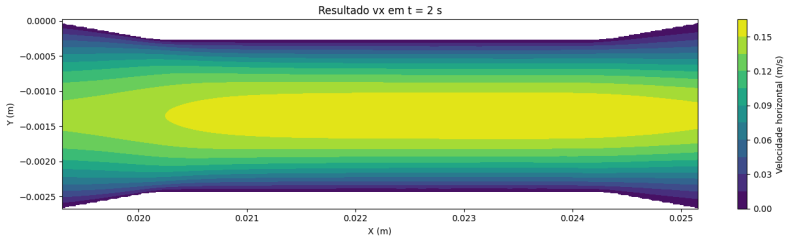
(e) Temperatura.

Figura 8.9: Resultados obtidos para o caso C1 no tempo $t_f = 2s$.

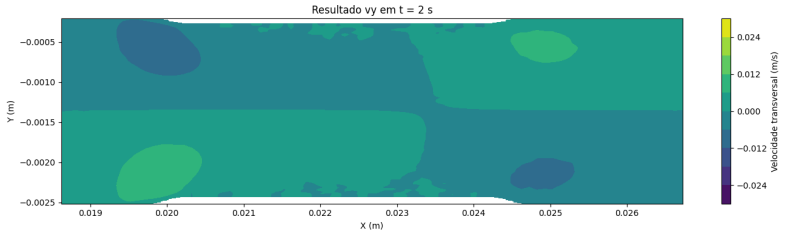
Consegue-se ver a presença de um aumento da magnitude da velocidade horizontal nas regiões em que o escoamento é restringido pelos perfis. Enquanto isso, é possível observar a presença de um padrão no campo de velocidades transversais que caracteriza um movimento de subida e descida do escoamento ao redor dos perfis, o que gera uma variação não só na magnitude, mas no sentido do campo de velocidades ao longo do escoamento.

Em relação à vorticidade, é possível verificar que, enquanto há a formação de regiões com maior intensidade se comparado ao caso completamente plano, a região de maior vorticidade ainda se encontra próxima aos nós de entrada da simulação.

Fazendo um corte em uma região qualquer em que há a intensificação do campo de velocidades, é obtido:



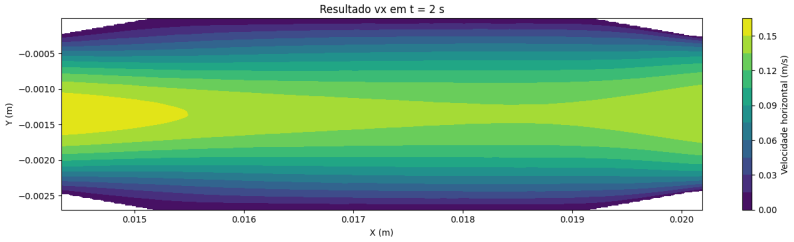
(a) Velocidade horizontal.



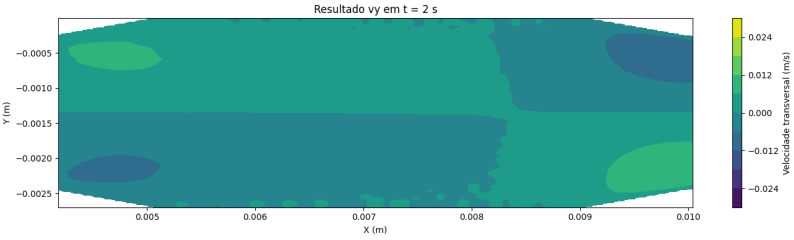
(b) Velocidade transversal.

Figura 8.10: Campo de velocidades em uma região restringida pelo perfil trapezoidal no caso C1

Em comparação, observando o perfil de velocidades nas regiões fora do perfil imposto à geometria, resulta-se nos gráficos:



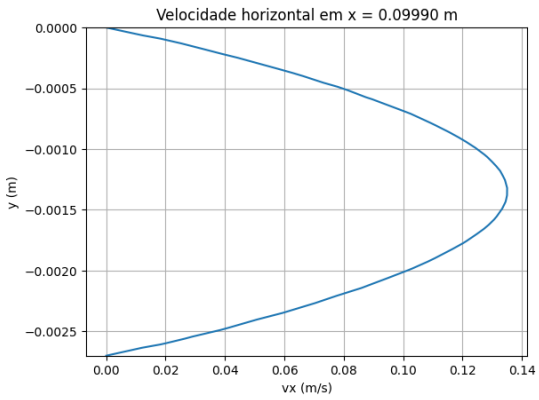
(a) Velocidade horizontal.



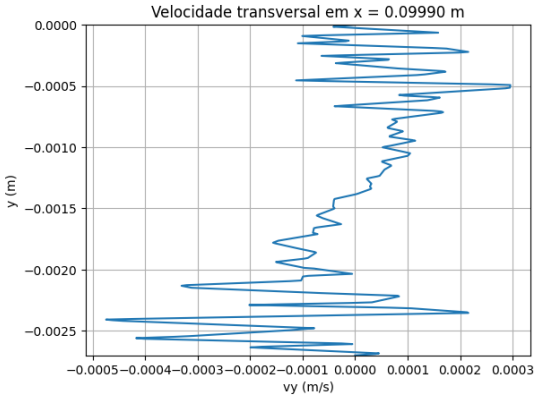
(b) Velocidade transversal.

Figura 8.11: Campo de velocidades em uma região não restringida pelo perfil trapezoidal no caso C1.

Por fim, avaliando os parâmetros de interesse na posição de $x_{data} = 0,0999m$, é obtido:

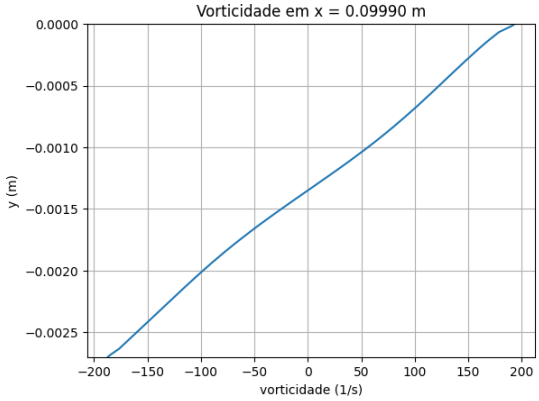


(a) Velocidade horizontal.

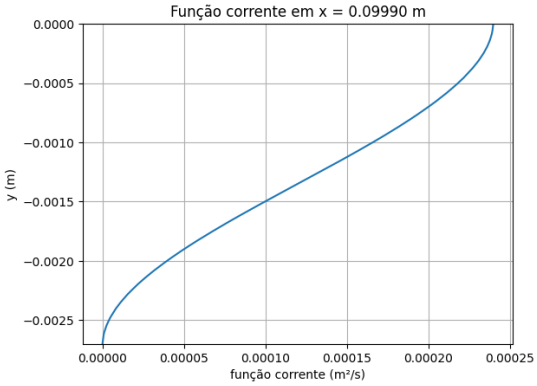


(b) Velocidade transversal.

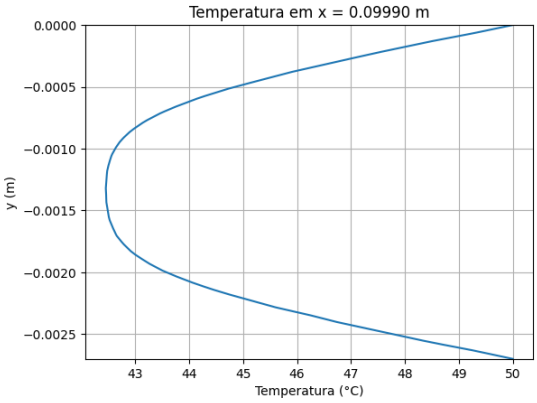
Figura 8.12: Campo de velocidades horizontais e transversais para o caso C1 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.



(a) Vorticidade.



(b) Função corrente.

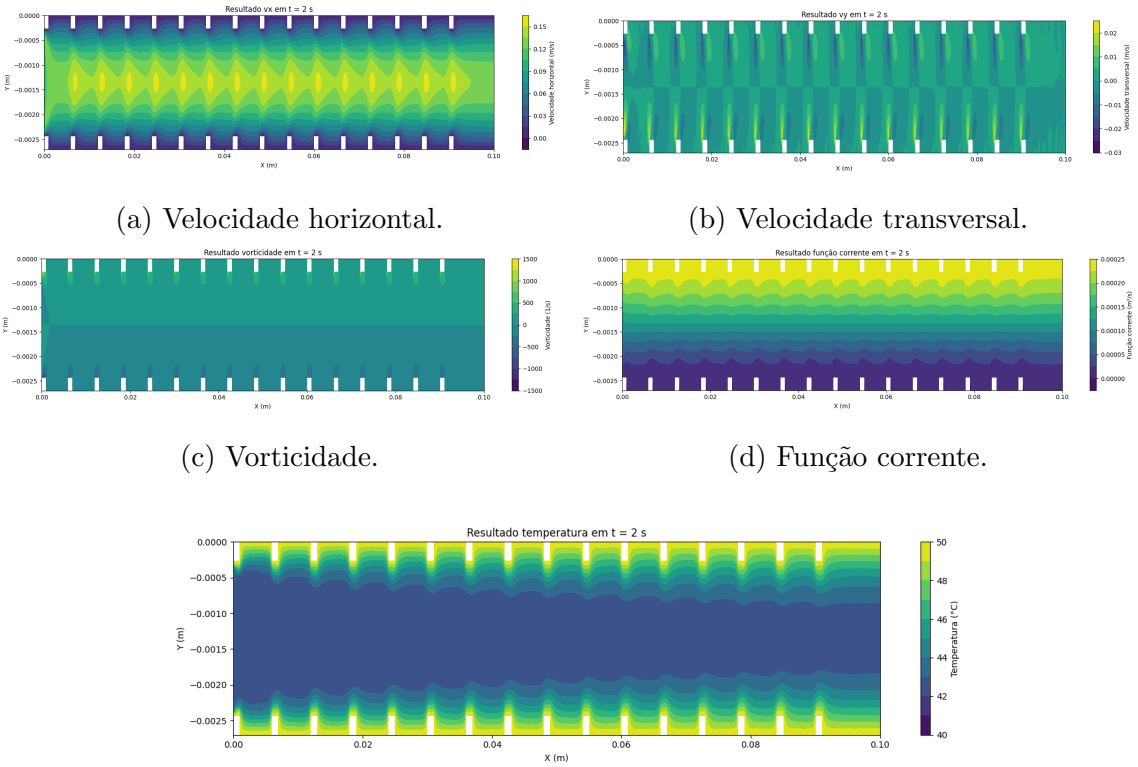


(c) Temperatura.

Figura 8.13: Vorticidade, função corrente e temperatura para o caso C1 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.

8.2.2 Caso C6

Para o caso C6, a simulação resultante possui a seguinte forma:



(a) Velocidade horizontal.

(b) Velocidade transversal.

(c) Vorticidade.

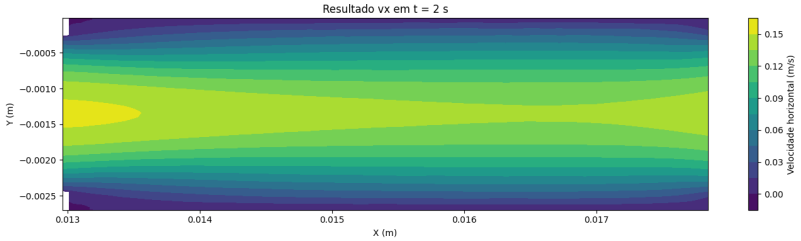
(d) Função corrente.

(e) Temperatura.

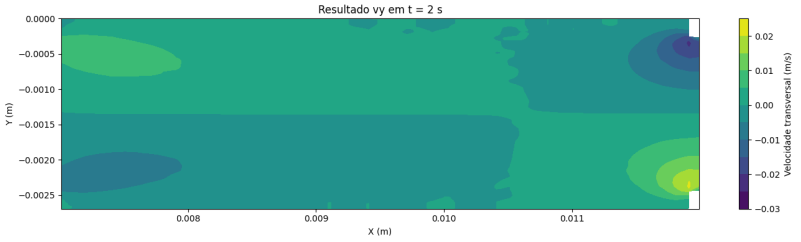
Figura 8.14: Resultados obtidos para o caso C6 no tempo $t_f = 2s$.

De modo semelhante ao primeiro caso, é visível a variação do campo de velocidades com o aumento da intensidade nas regiões em que o perfil restringe a passagem do escoamento. Todavia, existe uma região próxima imediatamente ao lado dos perfis que possui uma velocidade inferior a 0, o que implica em um movimento de retorno do escoamento, semelhante a um refluxo ou um vórtice formado por conta do perfil. Analisando as velocidades horizontais e verticais nestas regiões, obtém-se:

Em contrapartida, avaliando as velocidades horizontais e transversais nas regiões em que ocorre a restrição:

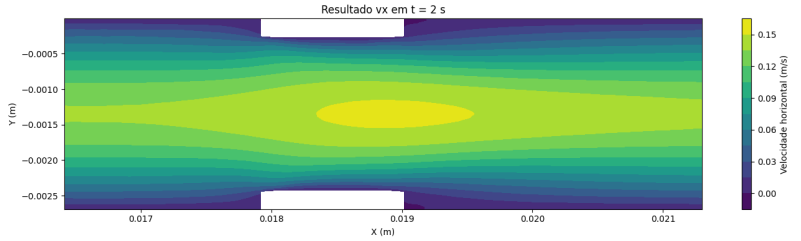


(a) Velocidade horizontal.

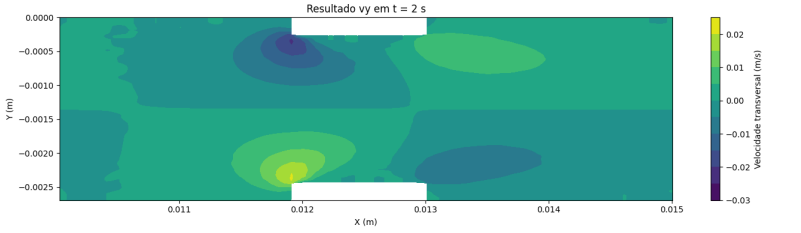


(b) Velocidade transversal.

Figura 8.15: Campo de velocidades em uma região próxima ao perfil trapezoidal no caso C6.



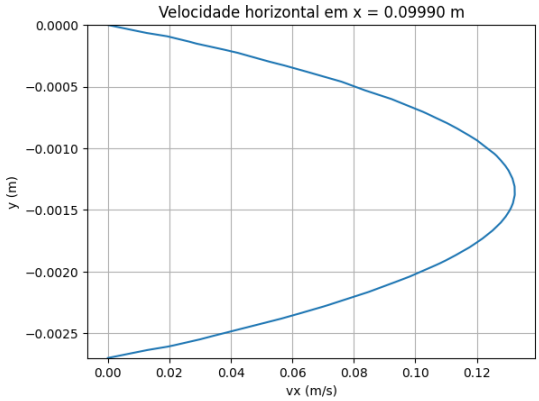
(a) Velocidade horizontal.



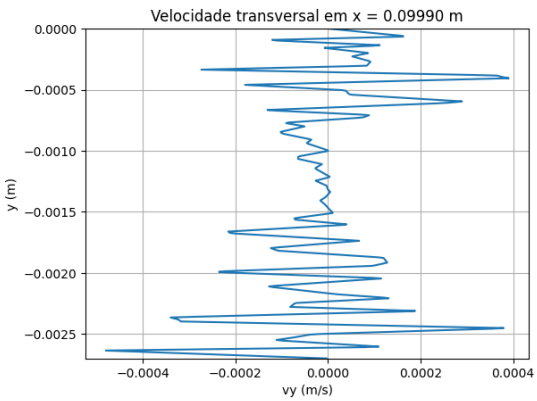
(b) Velocidade transversal.

Figura 8.16: Campo de velocidades em uma região restringida pelo perfil trapezoidal do caso C6.

Ademais, analisando todos os parâmetros na posição $x_{data} = 0,0999m$:

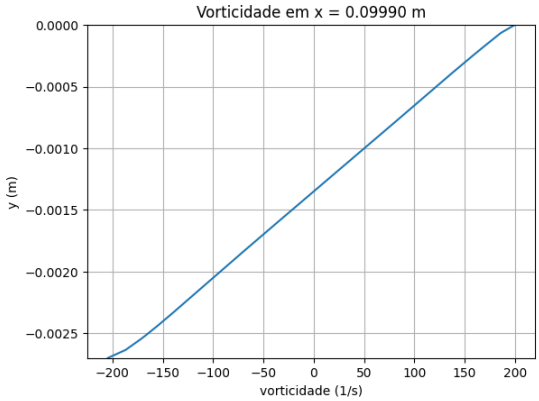


(a) Velocidade horizontal.

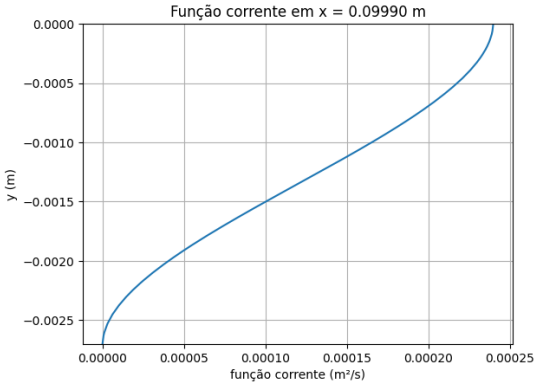


(b) Velocidade transversal.

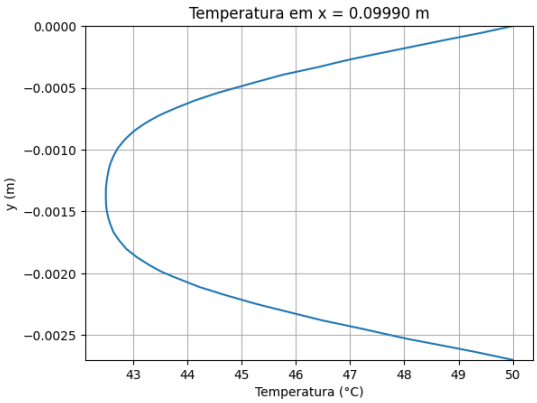
Figura 8.17: Campo de velocidades horizontais e transversais para o caso C6 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.



(a) Vorticidade.



(b) Função corrente.



(c) Temperatura.

Figura 8.18: Vorticidade, função corrente e temperatura para o caso C6 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.

8.3 Perfil Triangular

Para perfil triangular, foram avaliados 2 condições geométricas. Sendo estas referenciadas no capítulo 5 como os casos C2 e C7:

Tabela 8.3: Parâmetros Caso Triangular.

Caso	Tipo de Perfil	n_{pontos}	$L_{\text{reto}}(m)$	offset (m)	$d(m)$	$l(m)$	$a(m)$	tm	tmr	Entrada Limpa
C2	Triangular	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0
C7	Triangular	500	0,1	0,0027	0,005	0,001	0,00027	$2,00 \times 10^{-4}$	$2,00 \times 10^{-5}$	0

A diferença entre os casos C2 e C7 neste caso vai além da distância entre os perfis. Para o caso C2, os parâmetros acabam por gerar um formato de perfil parecido a uma serra, com triangulos colocados justapostamente sem um intervalo entre eles. No caso C7, a distância entre os perfis é tal que se aproxime dos casos trapezoidais apresentados nos casos C1 e C6, com a diferença sendo somente o perfil triangular imposto.

Gerando as malhas para ambos os casos C2 e C7, define-se o domínio computacional utilizado para simular os casos:

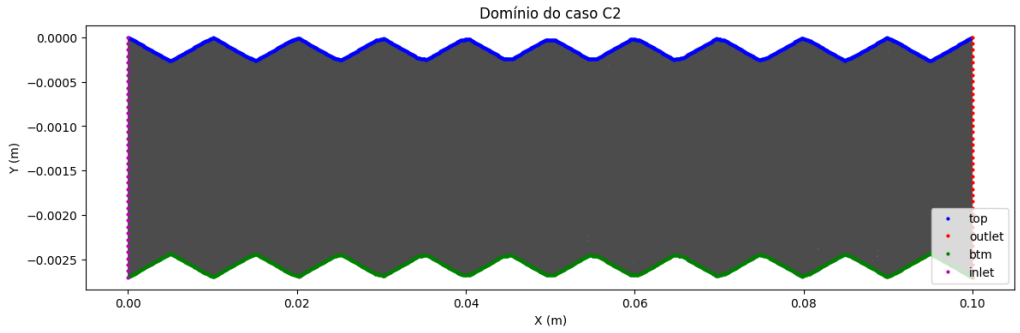
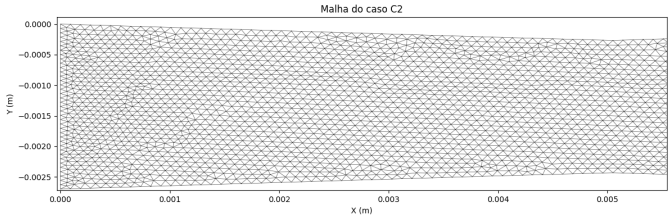
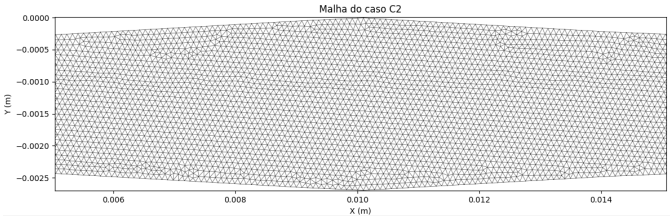


Figura 8.19: Domínio do caso 2 em conjunto com os nós de contorno.



(a) Região não refinada.



(b) Região refinada.

Figura 8.20: Diferentes níveis de refino para o domínio do caso C2.

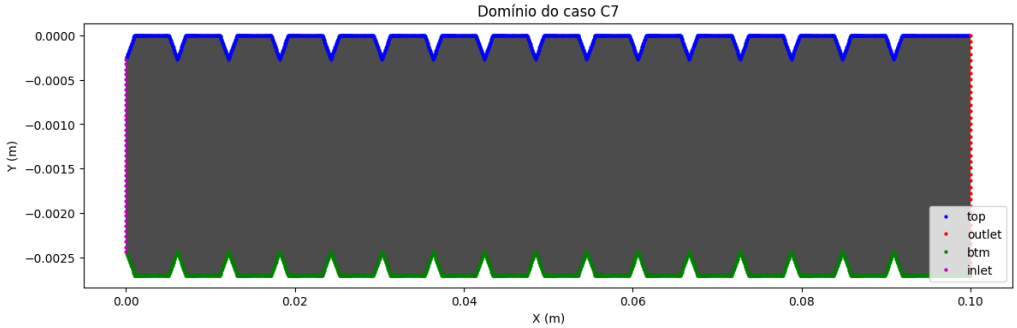
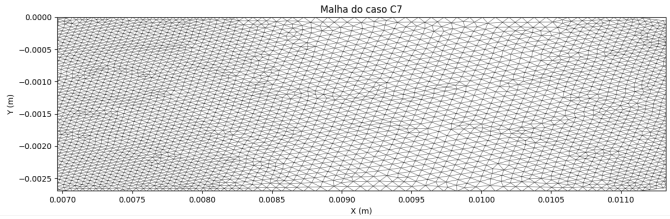
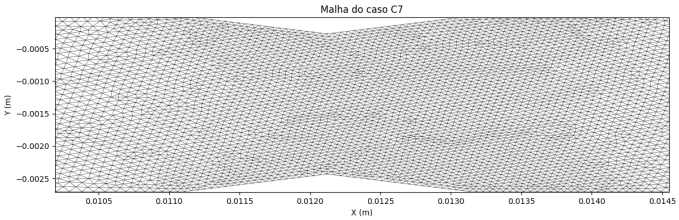


Figura 8.21: Domínio do caso C7 em conjunto com os nós de contorno.



(a) Região não refinada.



(b) Região refinada.

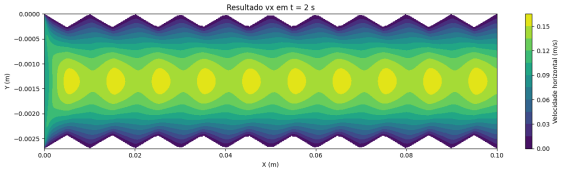
Figura 8.22: Diferentes níveis de refino para o domínio do caso C7.

Para o caso C2, o formato de serra acaba tornando interessante uma analise refinada em todo o domínio, enquanto no caso C7 é aplicado os algorítmos de refino localizado.

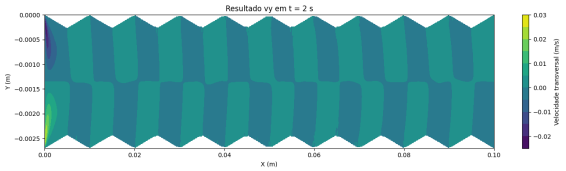
Para o caso C2, a malha gerada possui 35241 nós e 70480 elementos, enquanto que para o caso C7, a malha possui 83032 nós e 166062 elementos.

8.3.1 Caso C2

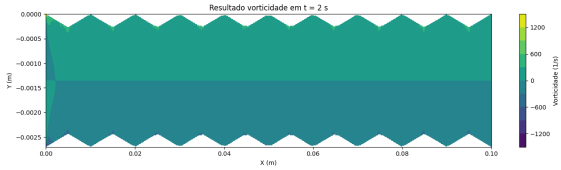
Para o caso C2, o resultado obtido da simulação é:



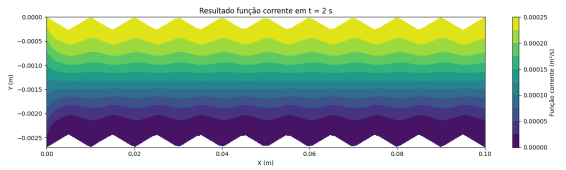
(a) Velocidade horizontal.



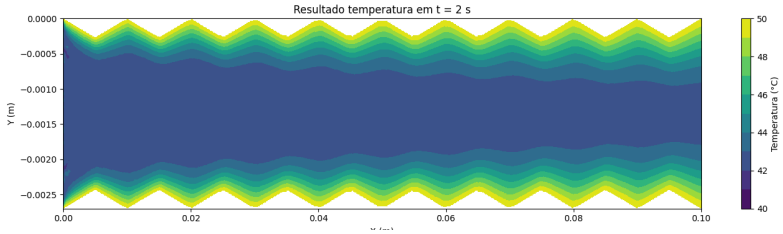
(b) Velocidade transversal.



(c) Vorticidade.



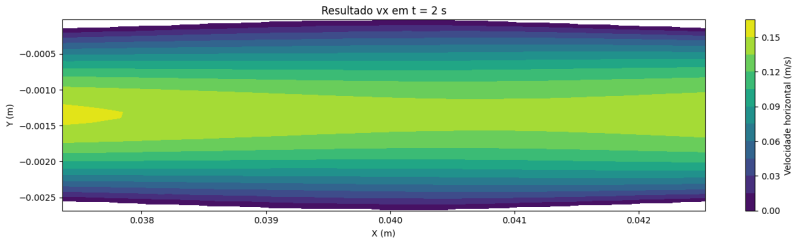
(d) Função corrente.



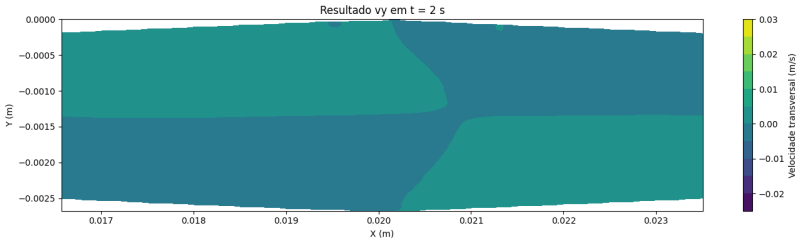
(e) Temperatura.

Figura 8.23: Resultados obtidos para o caso C2 no tempo $t_f = 2s$.

Assim como nos casos trapezoidais, ocorre a intensificação da velocidade na direção horizontal com o mesmo valor máximo. Todavia, o formato geométrico torna o comportamento alternado da velocidade horizontal mais suave, de modo que as regiões de velocidade horizontal alta são mais concentradas e com formato mais compacto se comparado às configurações trapezoidais. Analisando o resultado obtido na região de maior estrangulamento do domínio, resulta-se:



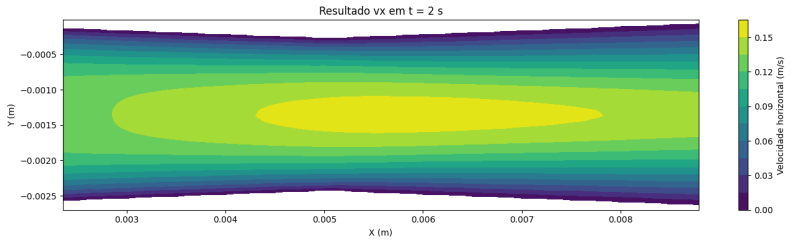
(a) Velocidade horizontal.



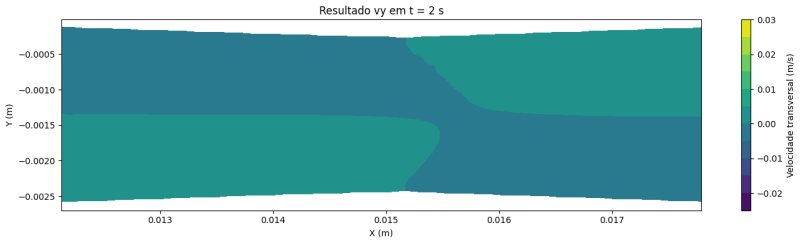
(b) Velocidade transversal.

Figura 8.24: Campo de velocidades em uma região fora da restrição triangular serrilhada no caso C2

Em contrapartida, analisando a velocidade em uma região prévia ao pico do perfil triangular serrilhado, é obtido:



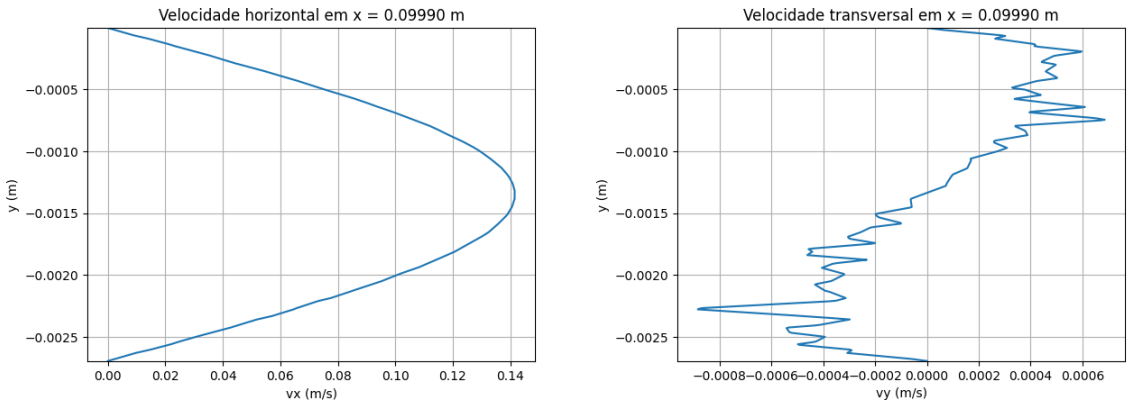
(a) Velocidade horizontal.



(b) Velocidade transversal.

Figura 8.25: Campo de velocidades em uma região restringida pelo perfil triangular serrilhado do caso C2

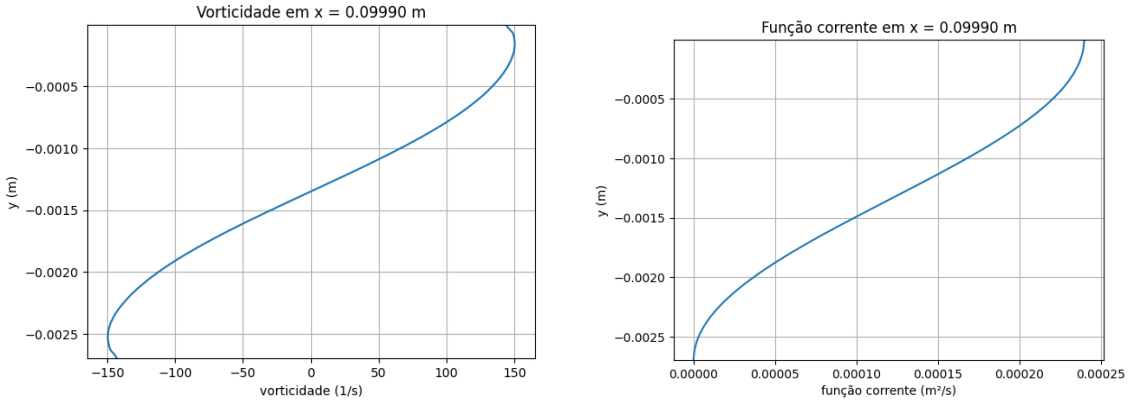
Concluindo, analisando os resultados na posição $x_{data} = 0,0999m$:



(a) Velocidade horizontal.

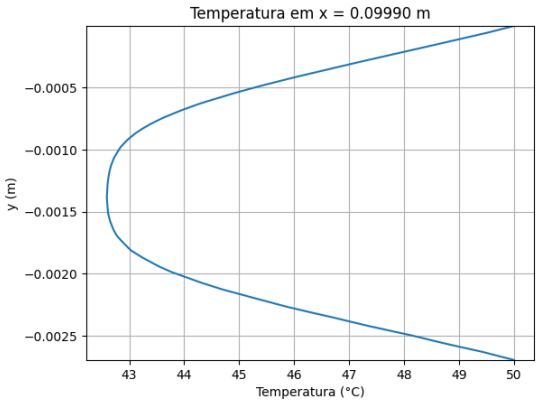
(b) Velocidade transversal.

Figura 8.26: Campo de velocidades horizontais e transversais para o caso C2 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.



(a) Vorticidade.

(b) Função corrente.

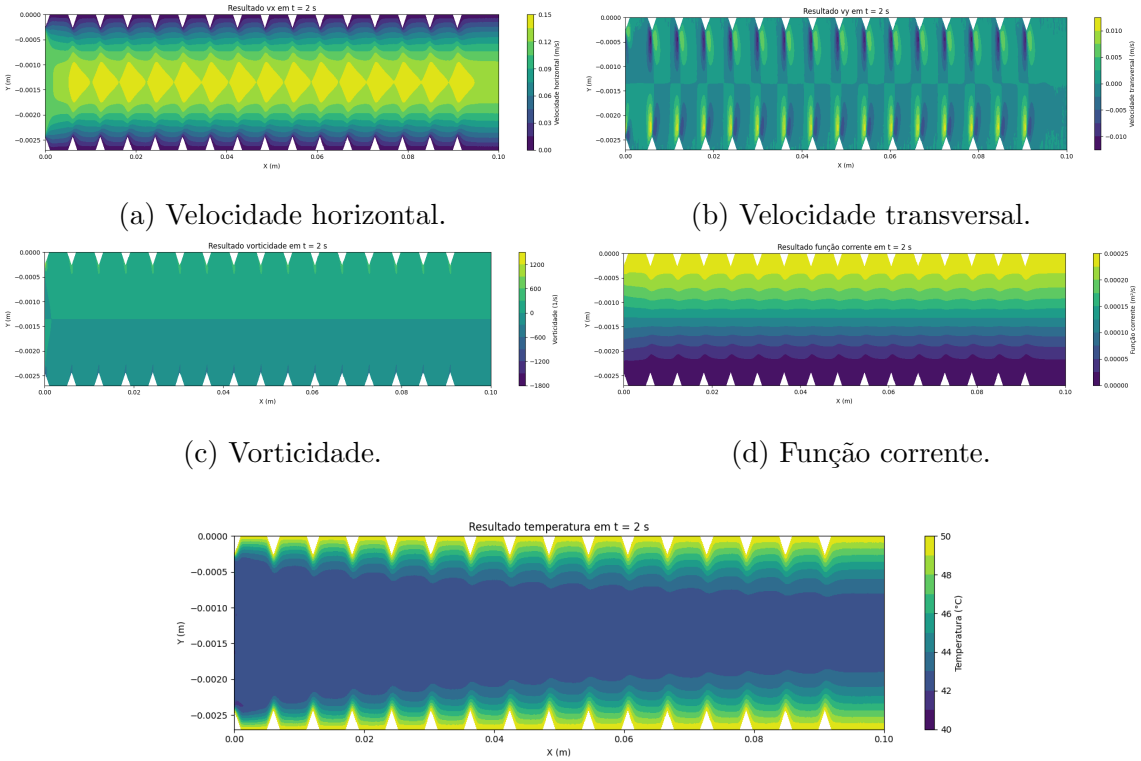


(c) Temperatura.

Figura 8.27: Vorticidade, função corrente e temperatura para o caso C2 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.

8.3.2 Caso C7

Para o caso C7, a simulação resultante possui a seguinte forma:



(a) Velocidade horizontal.

(b) Velocidade transversal.

(c) Vorticidade.

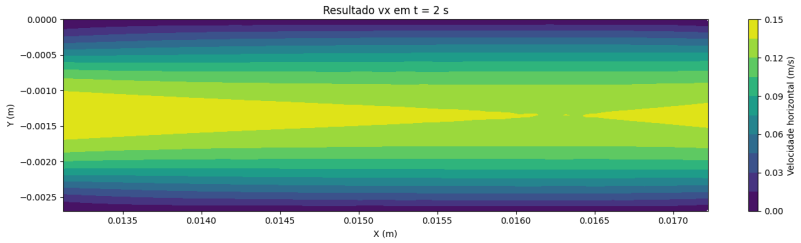
(d) Função corrente.

(e) Temperatura.

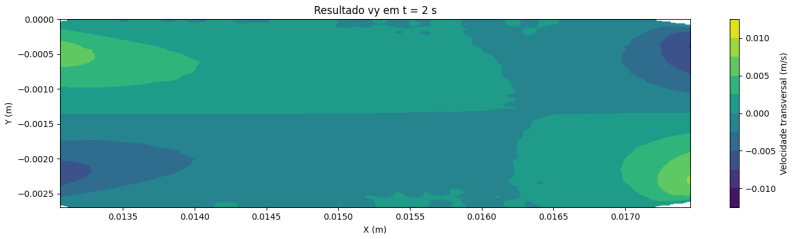
Figura 8.28: Resultados obtidos para o caso C7 no tempo $t_f = 2s$.

Percebe-se que a velocidade máxima horizontal no caso C7 é inferior ao resultado obtido no caso C2, embora a altura do estrangulamento seja a mesma. Ademais, observa-se que o campo de velocidades horizontais se aproxima mais dos casos trapezoidais do que o outro caso triangular C2, com uma região de alta intensidade de velocidade mais espalhada. Em relação à velocidade transversal, os efeitos são bem diferentes quando comparado ao caso C2, já que diferente da configuração anterior, as velocidades transversais possuiem magnitude variável e com intensidade considerável ao redor dos perfis da malha.

Analisando a velocidade obtida próximo do início de um perfil triangular, é encontrado:



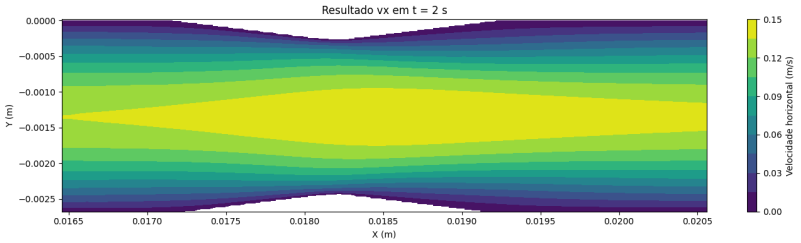
(a) Velocidade horizontal.



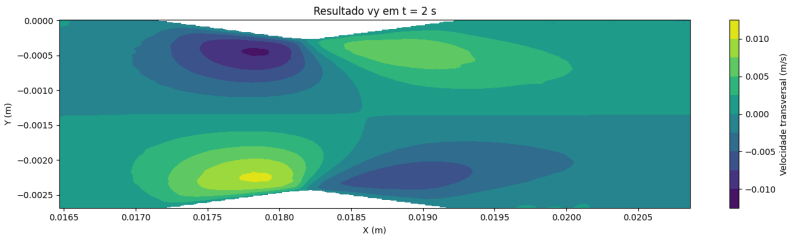
(b) Velocidade transversal.

Figura 8.29: Campo de velocidades em uma região próxima ao perfil triangular no caso C7

Em contrapartida, avaliando as velocidades horizontais e transversais nas regiões em que ocorre a restrição:



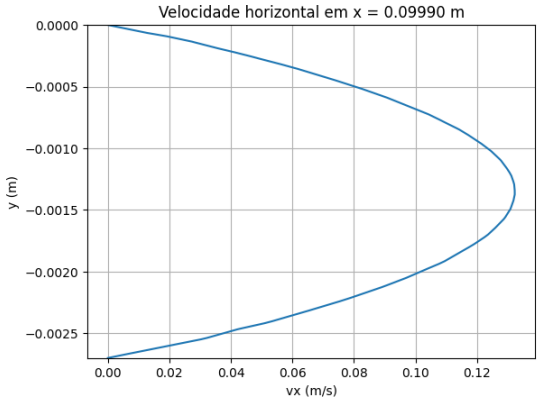
(a) Velocidade horizontal.



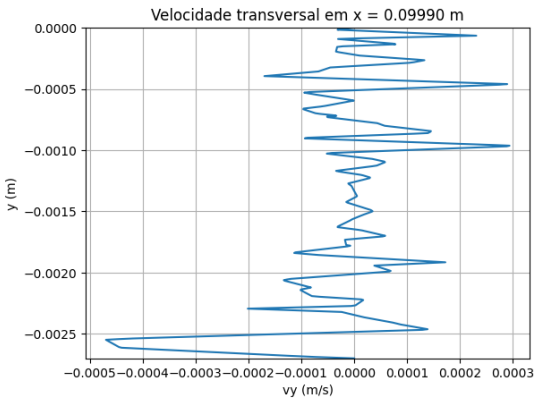
(b) Velocidade transversal.

Figura 8.30: Campo de velocidades em uma região restringida pelo perfil triangular no caso C7.

Ademais, analisando os resultados na posição $x_{data} = 0,0999m$:

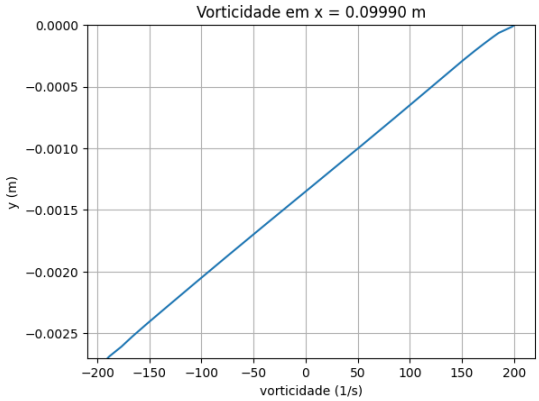


(a) Velocidade horizontal.

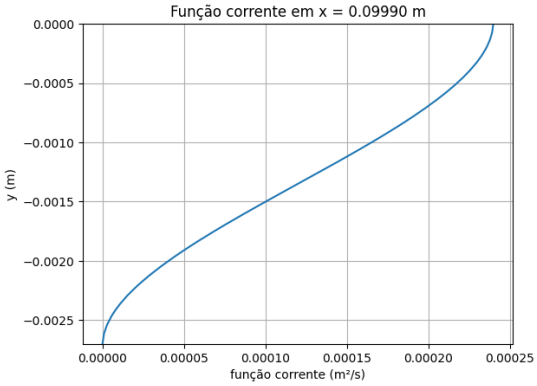


(b) Velocidade transversal.

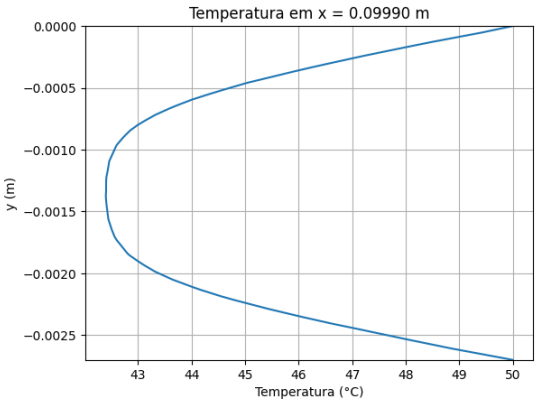
Figura 8.31: Campo de velocidades horizontais e transversais para o caso C7 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.



(a) Vorticidade.



(b) Função corrente.



(c) Temperatura.

Figura 8.32: Vorticidade, função corrente e temperatura para o caso C7 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.

8.4 Perfil Elíptico

Para perfil elíptico, foram avaliados 2 condições geométricas. Sendo estas referenciadas no capítulo 5 como os casos C3 e C8:

Tabela 8.4: Parâmetros Caso Elíptico.

Caso	Tipo de Perfil	n_{pontos}	$L_{\text{reto}}(m)$	offset (m)	$d(m)$	$l(m)$	$a(m)$	tm	tmr	Entrada Limpa
C3	elíptico	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0
C8	elíptico	500	0,1	0,0027	0,005	0,001	0,00027	$2,00 \times 10^{-4}$	$2,00 \times 10^{-5}$	0

Para os casos elípticos, a diferença também é somente o parâmetro de comprimento entre perfis que foi apresentado na seção 8.1, sendo esta distância controlada pelo parâmetro l.

Gerando as malhas para ambos os casos C3 e C8, define-se o domínio computacional utilizado para simular os casos:

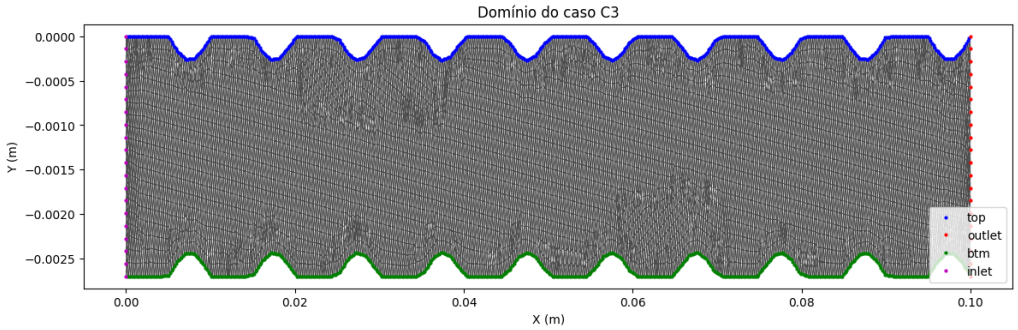
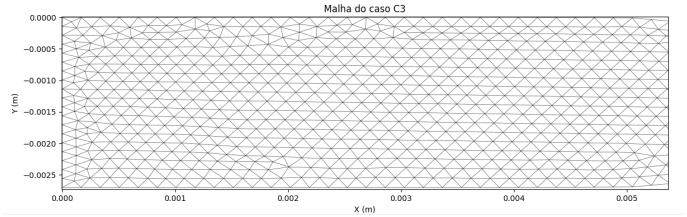
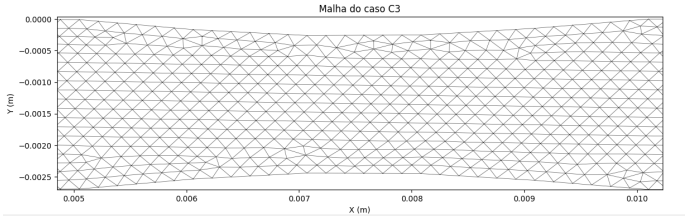


Figura 8.33: Domínio do caso C3 em conjunto com os nós de contorno.



(a) Região não refinada.



(b) Região refinada.

Figura 8.34: Diferentes níveis de refino para o domínio do caso C3.

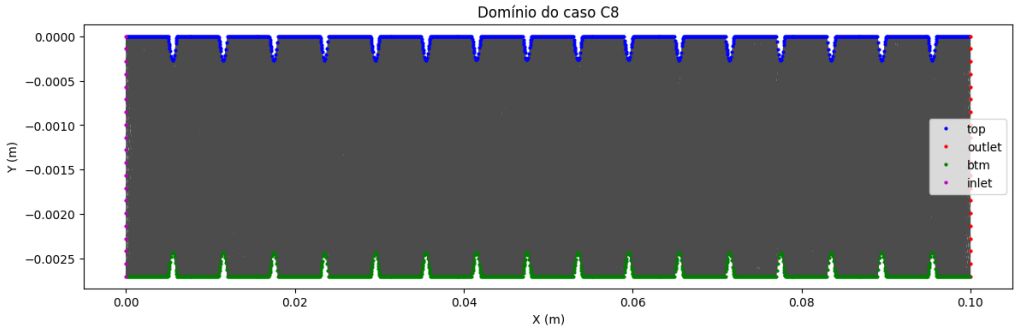
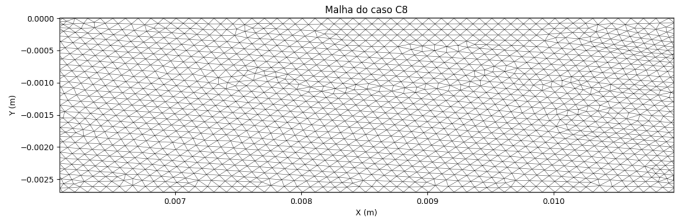
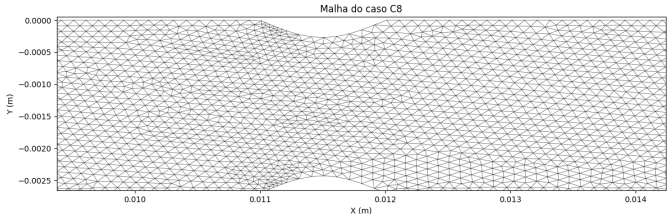


Figura 8.35: Domínio do caso C8 em conjunto com os nós de contorno.



(a) Região não refinada.



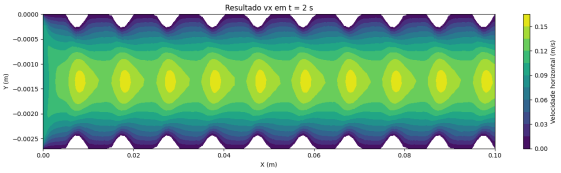
(b) Região refinada.

Figura 8.36: Diferentes níveis de refino para o domínio do caso C8.

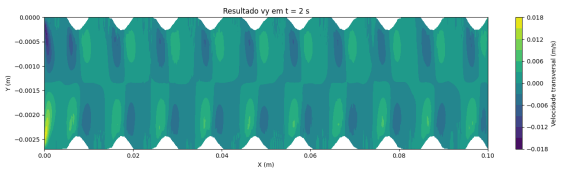
Para o caso C3, a malha gerada possui 11188 nós e 22374 elementos, enquanto que para o caso C8, a malha possui 34448 nós e 68894 elementos.

8.4.1 Caso C3

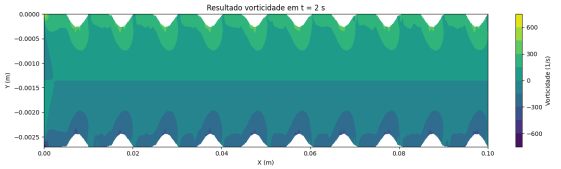
Para o caso C3, a simulação resultante possui a seguinte forma:



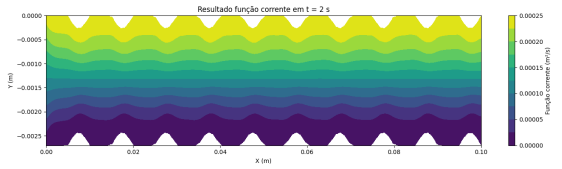
(a) Velocidade horizontal.



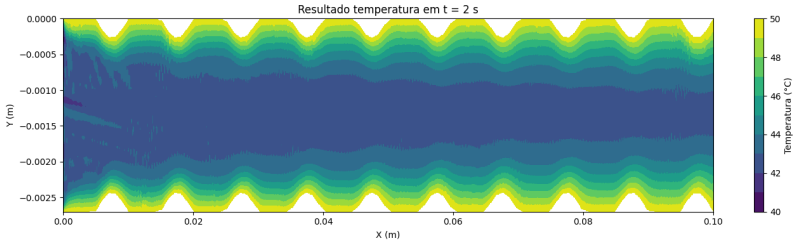
(b) Velocidade transversal.



(c) Vorticidade.



(d) Função corrente.

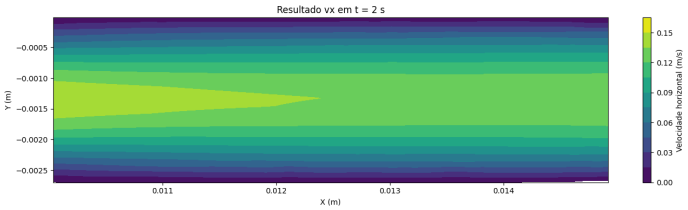


(e) Temperatura.

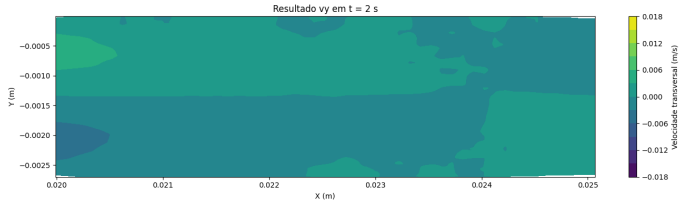
Figura 8.37: Resultados obtidos para o caso C3 no tempo $t_f = 2s$.

Observando o campo de velocidades do caso elíptico C3, nota-se uma proximidade com o caso trapezoidal C1, com a diferença sendo que as velocidades próximas de 0 que ficam em torno da parede são maiores quando comparado ao formato trapezoidal. Todavia, o formato elíptico possui um campo de velocidades também próximo ao resultado obtido na configuração C2, que possui formato triangular serrilhado, no que tange à concentração do campo de velocidades próximo do centro do canal, ainda que possua velocidades transversais semelhantes às encontradas nos casos trapezoidais e no caso C7.

Analisando os valores obtidos de velocidades horizontais e verticais em uma região prévia ao perfil elíptico, obtém-se:



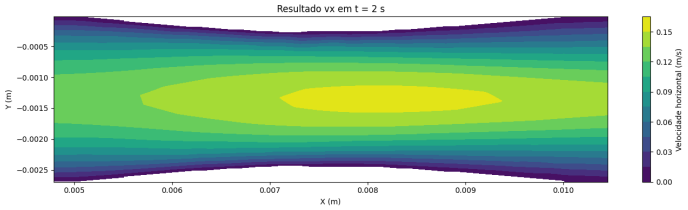
(a) Velocidade horizontal.



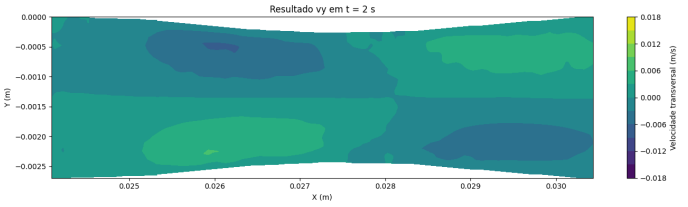
(b) Velocidade transversal.

Figura 8.38: Campo de velocidades em uma região próxima ao perfil elíptico no caso C3

Em contrapartida, avaliando os mesmos parâmetros na seção de velocidade horizontal máxima:



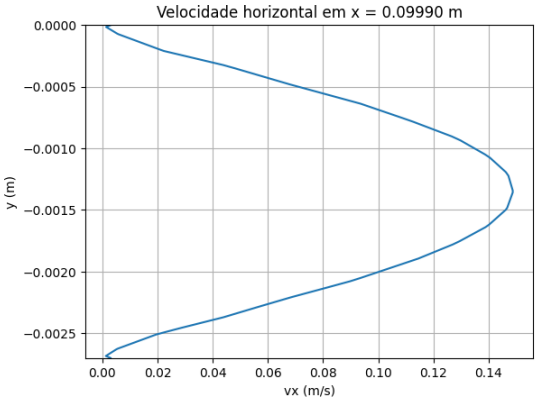
(a) Velocidade horizontal.



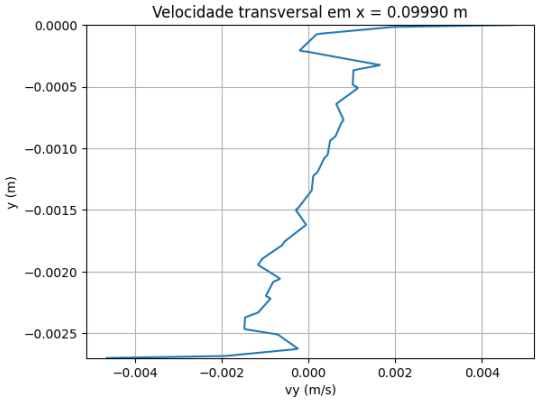
(b) Velocidade transversal.

Figura 8.39: Campo de velocidades em uma região restringida pelo perfil elíptico no caso C3

Por fim, analisando os valores obtidos na posição $x_{data} = 0,0999m$:

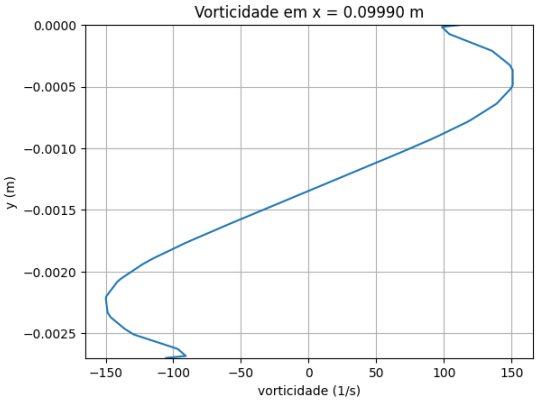


(a) Velocidade horizontal.

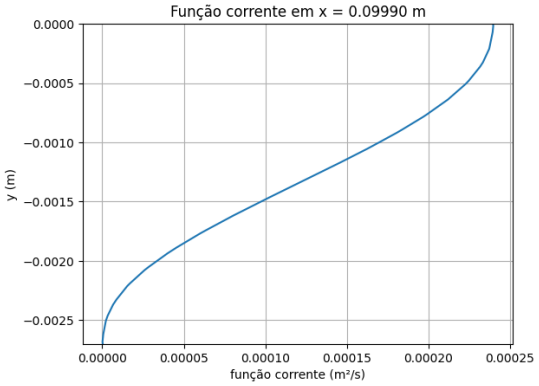


(b) Velocidade transversal.

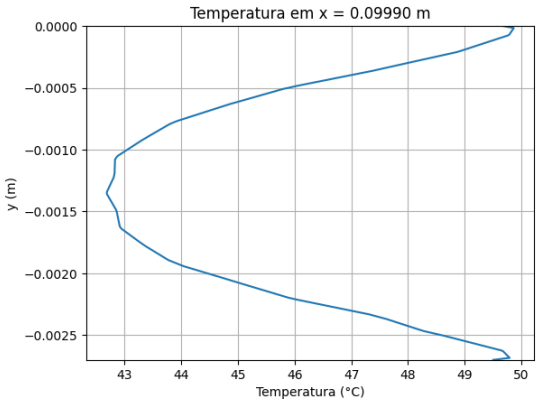
Figura 8.40: Campo de velocidades horizontais e transversais para o caso C3 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.



(a) Vorticidade.



(b) Função corrente.

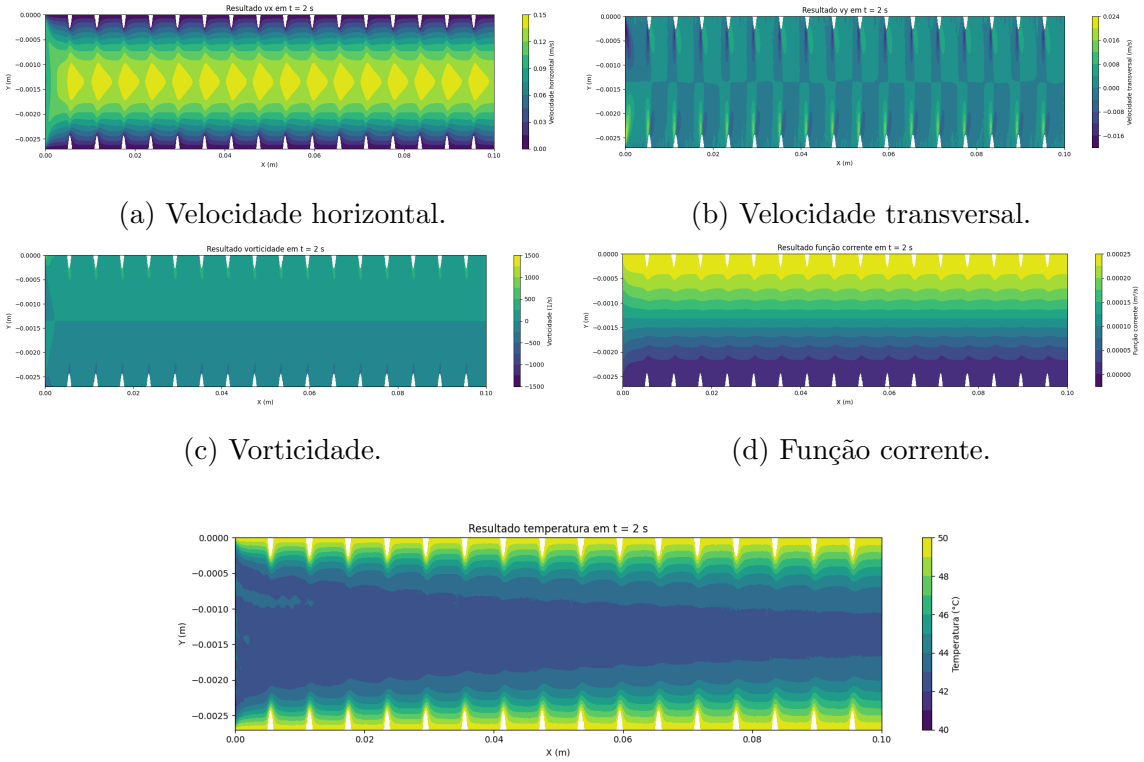


(c) Temperatura.

Figura 8.41: Vorticidade, função corrente e temperatura para o caso C3 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.

8.4.2 Caso C8

Para o caso C8, a simulação resultante possui a seguinte forma:



(a) Velocidade horizontal.

(b) Velocidade transversal.

(c) Vorticidade.

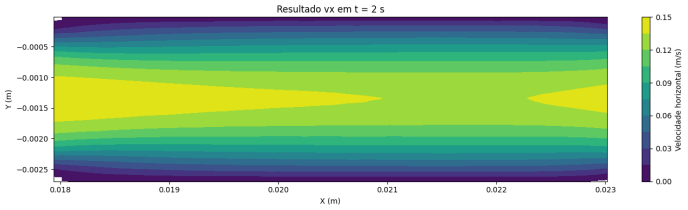
(d) Função corrente.

(e) Temperatura.

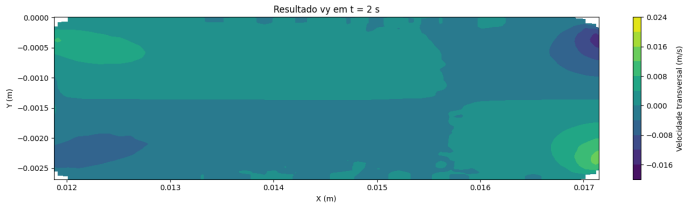
Figura 8.42: Resultados obtidos para o caso C8 no tempo $t_f = 2s$.

Nota-se que, diferente do caso C3, o campo de velocidades horizontal é mais próximo do perfil trapezoidal que o triangular. E este comportamento também é presente no formato dos campos de velocidade transversal e temperatura. Visualizando as regiões próximas ao perfil elíptico deste caso, encontra-se:

Em contrapartida, avaliando as velocidades horizontais e transversais em uma posição de velocidade máxima:

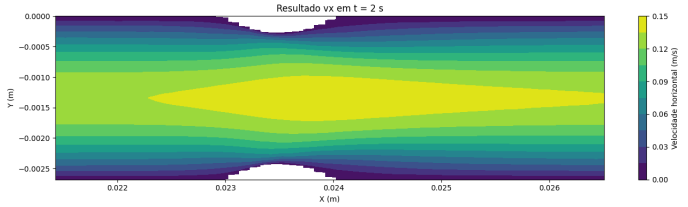


(a) Velocidade horizontal.

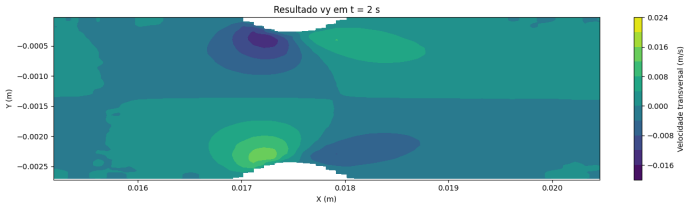


(b) Velocidade transversal.

Figura 8.43: Campo de velocidades em uma região próxima ao perfil elíptico no caso C8



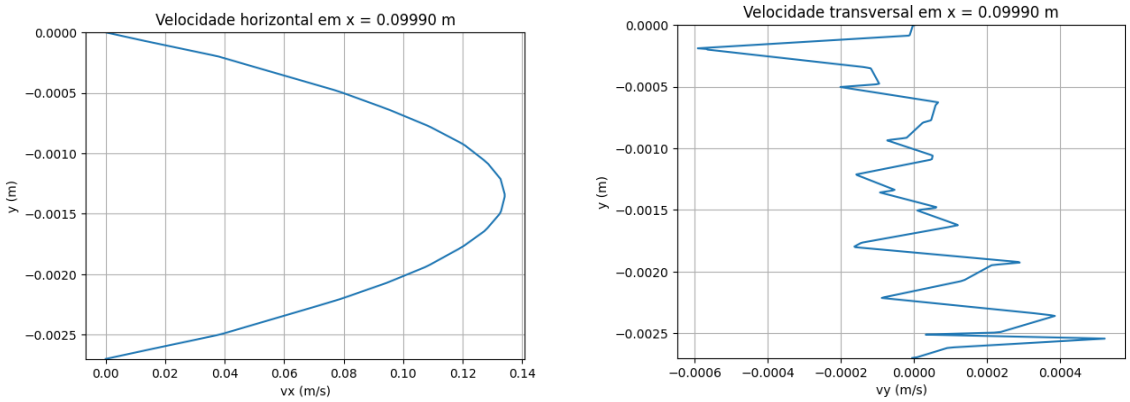
(a) Velocidade horizontal.



(b) Velocidade transversal.

Figura 8.44: Campo de velocidades em uma região restringida pelo perfil elíptico no caso C8

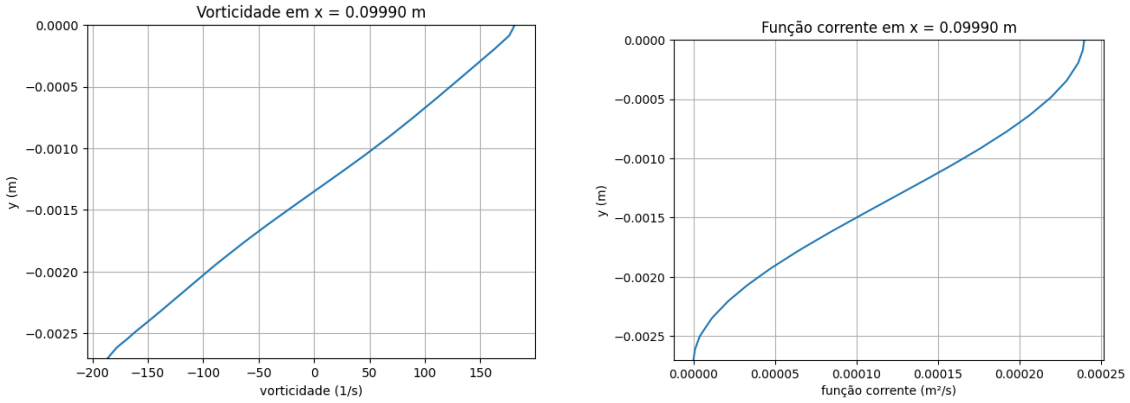
Por fim, analisando os resultados na posição $x_{data} = 0,0999m$:



(a) Velocidade horizontal.

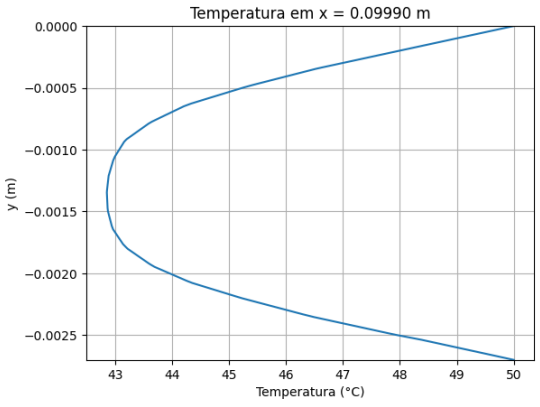
(b) Velocidade transversal.

Figura 8.45: Campo de velocidades horizontais e transversais para o caso C8 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.



(a) Vorticidade.

(b) Função corrente.



(c) Temperatura.

Figura 8.46: Vorticidade, função corrente e temperatura para o caso C8 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.

8.5 Perfil Corrugado

Para o perfil corrugado, foram avaliados 2 condições geométricas. Sendo estas referenciadas no capítulo 5 como os casos C4 e C5:

Tabela 8.5: Parâmetros Caso Plano

Caso	Tipo de Perfil	n_{pontos}	$L_{\text{reto}}(m)$	offset (m)	$d(m)$	$l(m)$	$a(m)$	tm	tmr	Entrada Limpa
C4	Corrugado	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	0
C5	Corrugado	100	0,1	0,0027	0,005	0,005	0,00027	$5,00 \times 10^{-4}$	$5,00 \times 10^{-5}$	1

Para os casos corrugados, optou-se por não utilizar critérios de refino nas partes influenciadas pelas funções de ruído, visto que neste caso, todo o domínio geométrico demanda um refino alto para visualizar de modo efetivo os efeitos das alterações de altura aleatória na perfomance do caso.

Gerando as malhas para ambos os casos C4 e C5, define-se o domínio computacional utilizado para simular os casos:

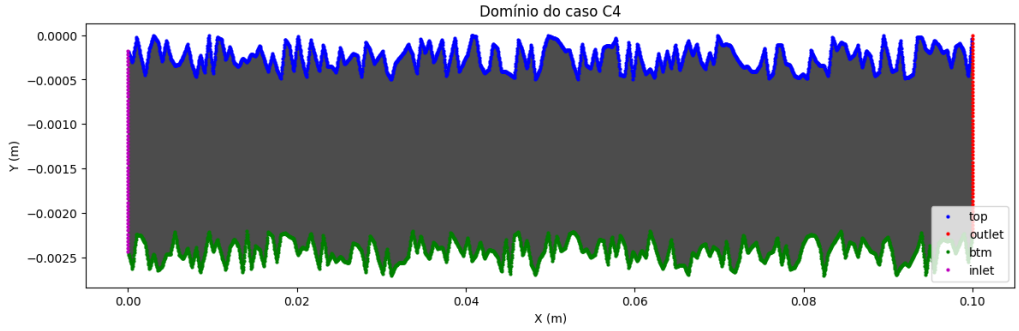


Figura 8.47: Domínio do caso 4 em conjunto com os nós de contorno.

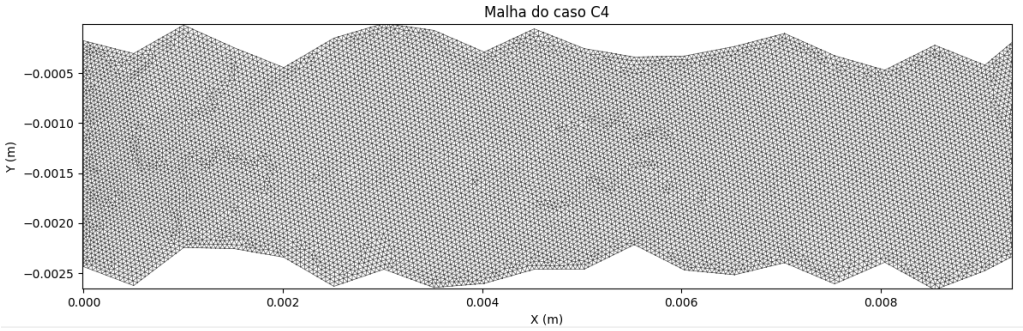


Figura 8.48: Elementos do caso C4 em um trecho do domínio.

Para o caso C5, utiliza-se de um refino especializado nas regiões em que se insere as funções de ruído a partir da coordenada $x = 0.0m$. Assim, é obtido:

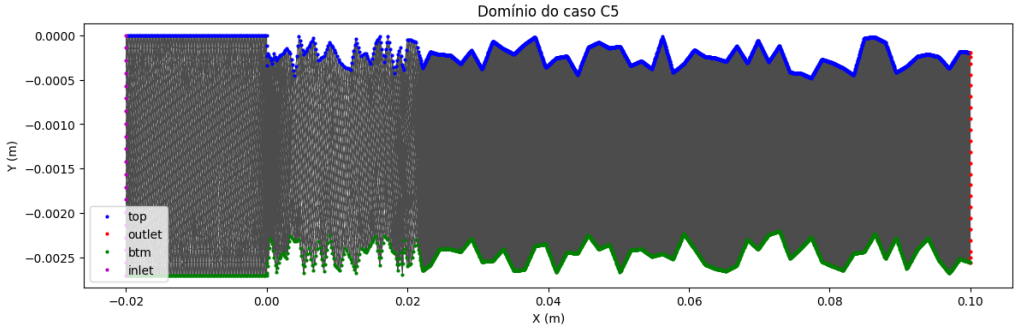
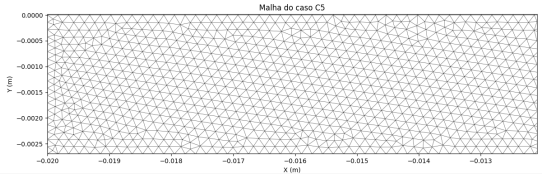
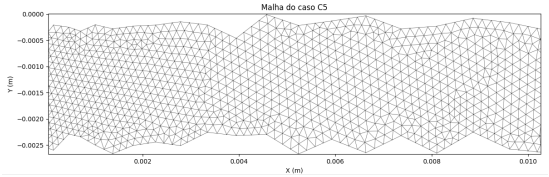


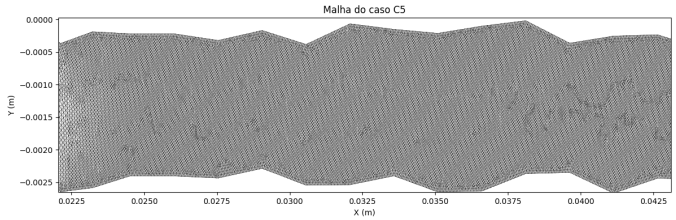
Figura 8.49: Domínio do caso C5 em conjunto com os nós de contorno.



(a) Região não refinada.



(b) Região de transição com refino.



(c) Região com alto refino.

Figura 8.50: Diferentes níveis de refino para o domínio do caso C5.

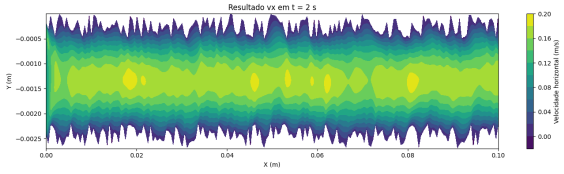
Devido à geometria não padronizada destes casos, é uma forma interessante de comparar os efeitos de performance que possíveis imperfeições podem implicar em um escoamento entre placas.

Como é apresentado na metodologia assim como nas imagens do caso C5, este caso possui um comprimento superior aos outros casos. Desta forma, será avaliado o caso C5 somente as influências das condições de entrada no campo de velocidades e temperatura para uma região aleatória. Assim, este caso não será comparado com outras geometrias.

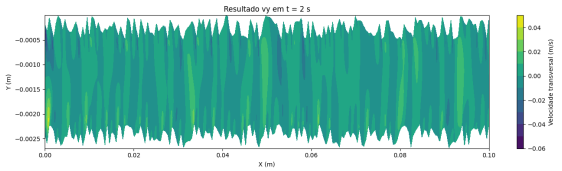
Para o caso C4, a malha gerada possui 118204 nós e 236406 elementos, enquanto que para o caso C5, a malha possui 89334 nós e 176666 elementos.

8.5.1 Caso C4

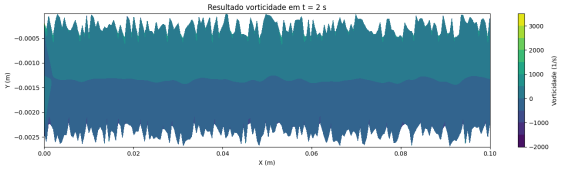
Para o caso C4, a simulação finalizou com os seguintes resultados:



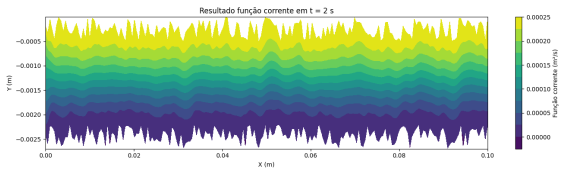
(a) Velocidade horizontal.



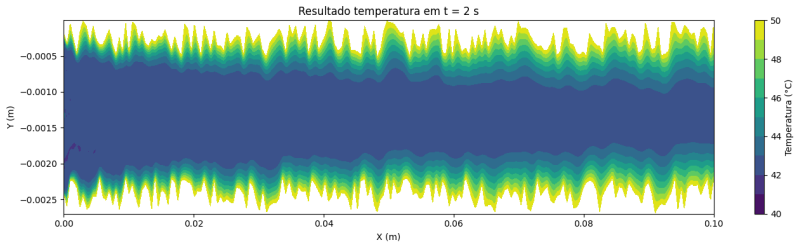
(b) Velocidade transversal.



(c) Vorticidade.



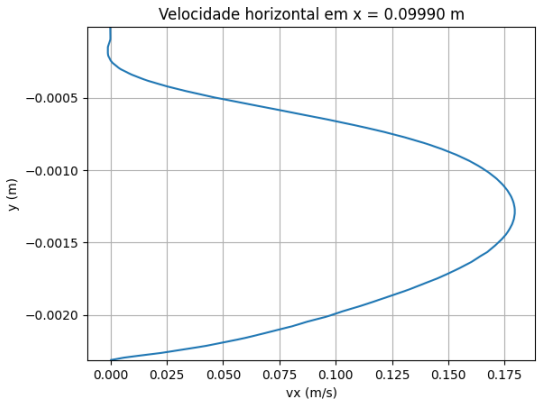
(d) Função corrente.



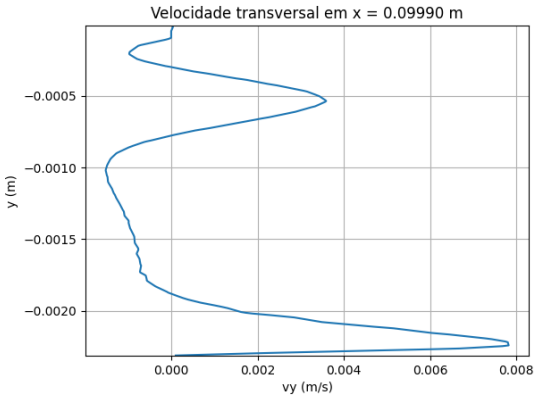
(e) Temperatura.

Figura 8.51: Resultados obtidos para o caso C4 no tempo $t_f = 2s$.

Extraindo os resultados na posição $x_{data} = 0,0999m$:

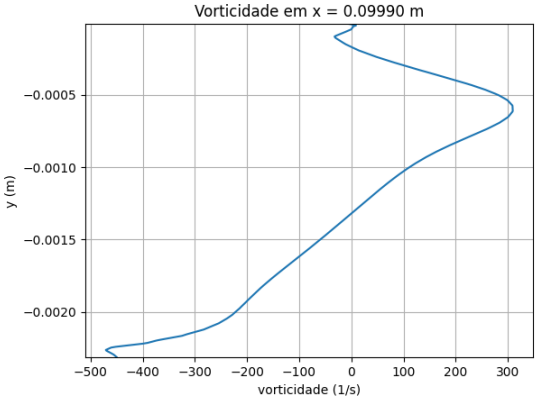


(a) Velocidade horizontal.

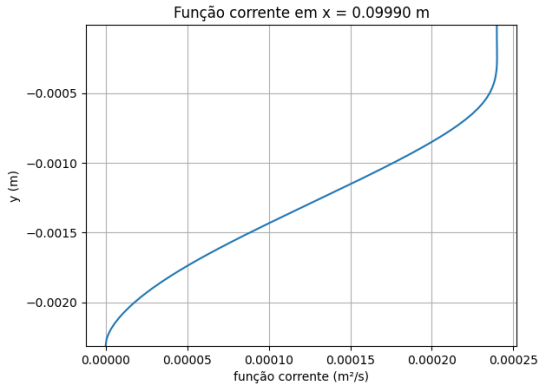


(b) Velocidade transversal.

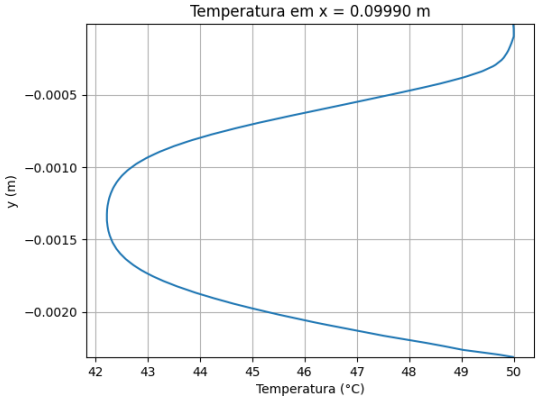
Figura 8.52: Campo de velocidades horizontais e transversais para o caso C4 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.



(a) Vorticidade.



(b) Função corrente.

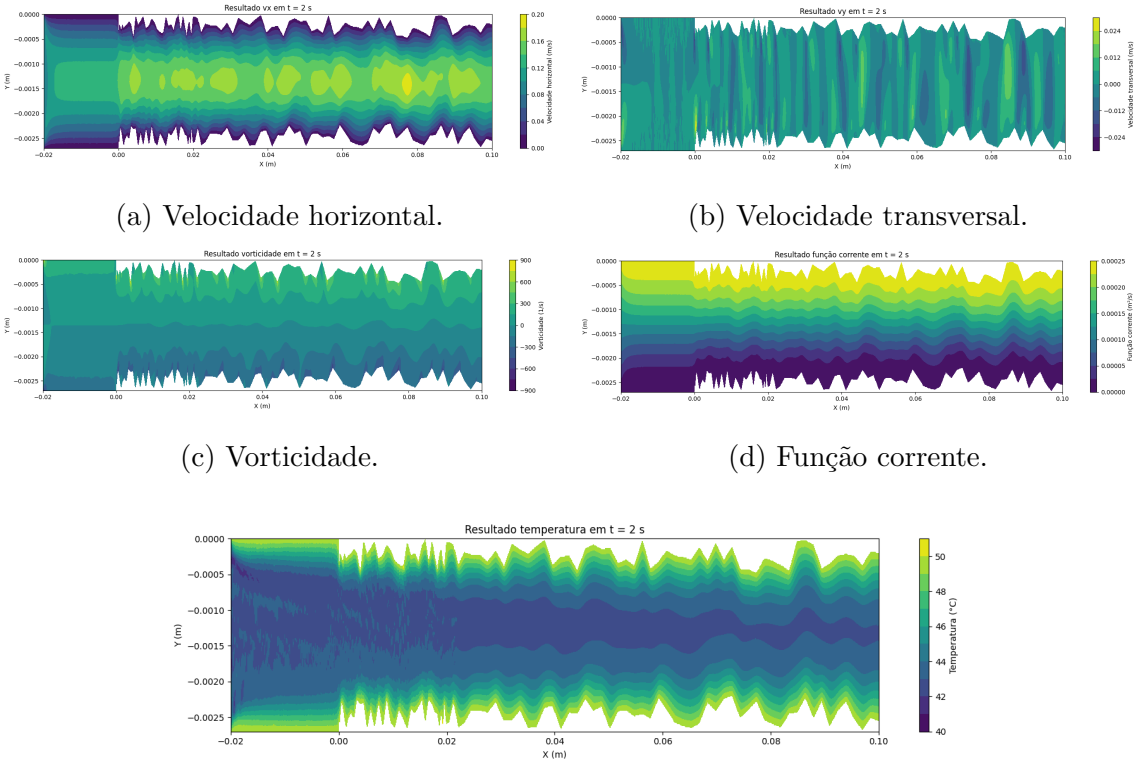


(c) Temperatura.

Figura 8.53: Vorticidade, função corrente e temperatura para o caso C4 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.

8.5.2 Caso C5

No caso C5, obtém-se os seguintes resultados para o tempo final especificado:



(a) Velocidade horizontal.

(b) Velocidade transversal.

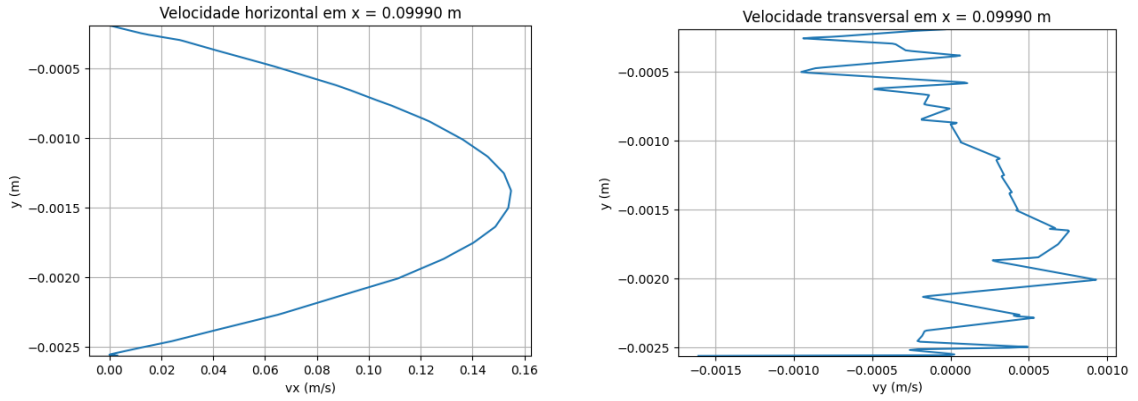
(c) Vorticidade.

(d) Função corrente.

(e) Temperatura.

Figura 8.54: Resultados obtidos para o caso C5 no tempo $t_f = 2s$.

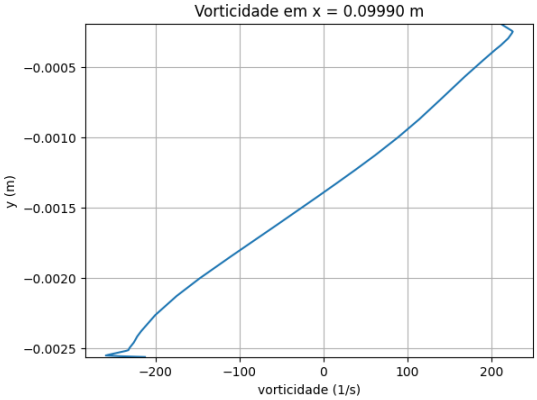
Analisando os resultados na posição $x_{data} = 0,0999m$:



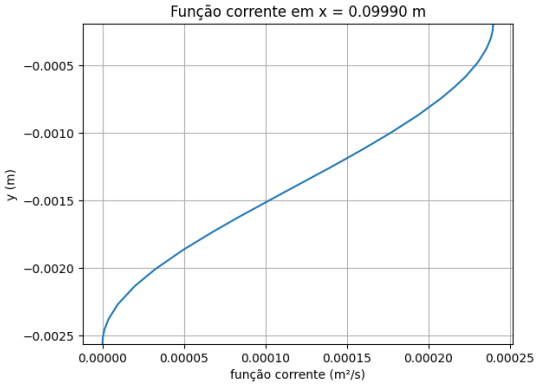
(a) Velocidade horizontal.

(b) Velocidade transversal.

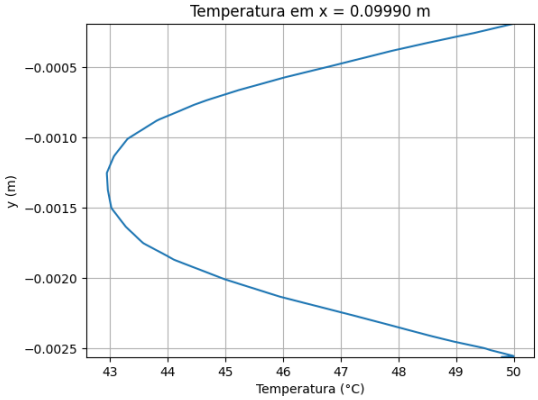
Figura 8.55: Campo de velocidades horizontais e transversais para o caso C5 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.



(a) Vorticidade.



(b) Função corrente.



(c) Temperatura.

Figura 8.56: Vorticidade, função corrente e temperatura para o caso C5 no tempo $t_f = 2s$ na seção de $x_{data} = 0.0999m$.

8.6 Análise Entre Geometrias

Como forma de análise entre as geometrias, utiliza-se do algoritmo de comparação mencionado na seção de criação de ferramentas para avaliar as velocidades médias e temperaturas energéticas médias em diferentes seções ao longo do domínio.

Para este projeto, foram tomado 200 pontos de medição ao longo do comprimento de $x_0 = 0m$ até o comprimento final $x_f = 0.1m$. Assim, é obtido os seguintes resultados:

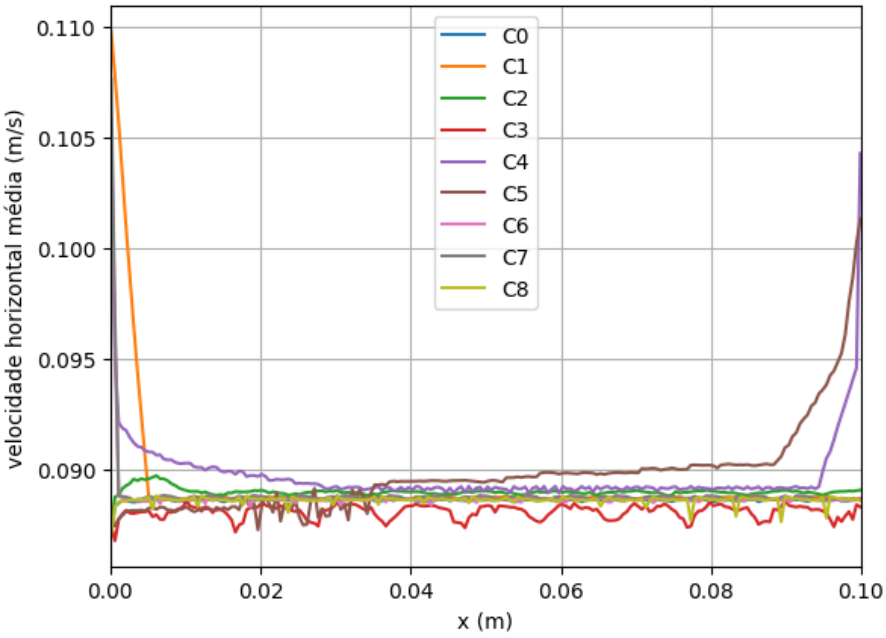


Figura 8.57: Velocidade horizontal média para todos os casos avaliados.

É visível que os campos de velocidade média para os casos com alterações padronizadas variam pouco e sempre sobre uma velocidade média que é próxima da velocidade média para o caso das placas planas. Todavia, percebe-se que para os casos corrugados, a velocidade média varia consideravelmente no final, o que é um resultado do formato consideravelmente restringido presente no final de ambos os domínios dos casos C4 e C5. Avaliando separadamente os casos periódicos, é obtido o seguinte gráfico:

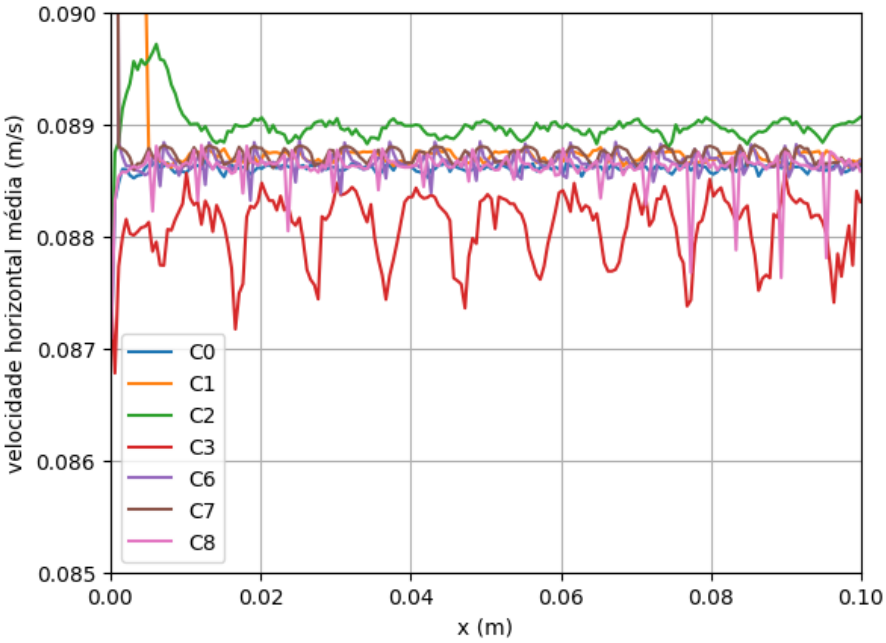


Figura 8.58: Velocidade horizontal média para os casos periódicos.

Analisando os casos periódicos, nota-se que o caso C3 possui uma alteração da velocidade média mais intensa, enquanto que para a mesmo tipo geométrico, o caso C8 se mantém próximo da velocidade média do caso plano.

Percebe-se também que o caso triangular serrilhado C2 possui uma velocidade média sempre acima da velocidade média. Isso também acontece em parte para os casos C7, C1 e C6. Porém, a velocidade média destes outros casos é próxima do caso plano. Dessa forma, pode-se inferir que o caso serrilhado consegue aumentar de modo mais efetivo o transporte de flúido dentre os casos simulados.

Analisando a temperatura, obtém-se o seguinte gráfico:

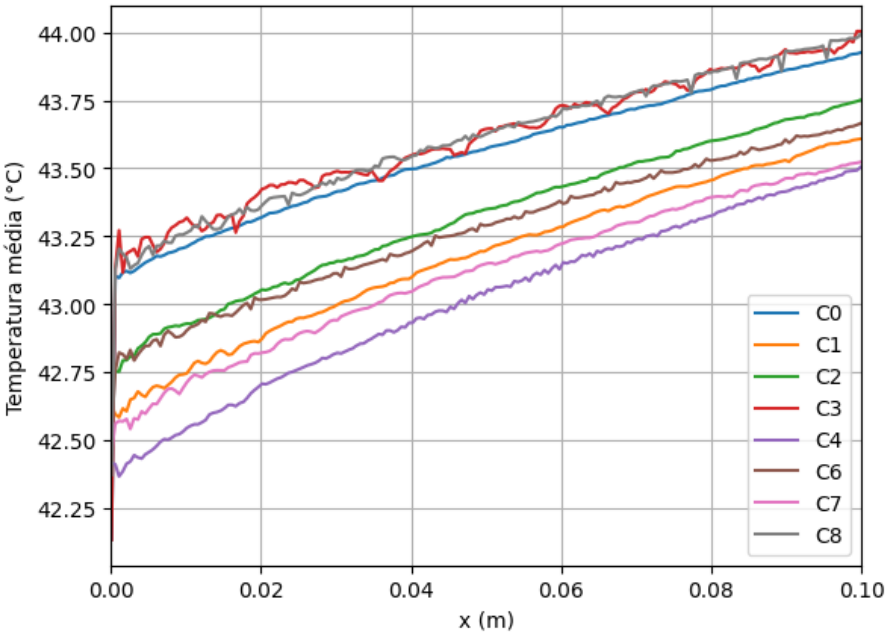


Figura 8.59: Temperatura energética média para todos os casos avaliados.

Neste gráfico, consegue-se visualizar os efeitos tanto de forma quanto de espessura dos perfis periódicos e corrugados na troca de calor.

Inicialmente, verifica-se que os casos elípticos foram os únicos que obtiveram uma temperatura média energética acima do caso plano, ou seja, obtiveram uma troca térmica mais efetiva. Observando os casos trapezoidais, verifica-se que a menor espessura dos perfis impactou positivamente a troca térmica do caso, embora teoricamente a área de troca tenha se reduzido. Para os casos triangulares, verifica-se que a configuração serrilhada é mais interessante se comparada à forma distribuída do caso C7, embora ambas tenham uma temperatura média inferior ao caso da placa plana.

Em relação à configuração aleatória C4, nota-se que embora ela tenha o pior resultado no que tange à temperatura média energética, o desenvolvimento da temperatura ao longo do comprimento é o mais acelerado, de modo a se aproximar da curva C7 no final da simulação em $x_f = 0.1m$. Esse comportamento pode implicar em uma perfomance superior aos casos periódicos em domínios maiores, em que o comprimento dos tubos supera ao comprimento do projeto.

Dessa forma, é possível inferir que para o escoamento laminar entre placas proposto pelo projeto, a configuração eliptica é a configuração que consegue maximizar a troca térmica dentre os casos apresentados. Em contrapartida, dependendo da

forma que os perfis elípticos se distribuem, é possível uma redução considerável da velocidade média do escoamento. Enquanto isso, casos em que há uma possibilidade de aumento da velocidade do escoamento podem acabar por interferir negativamente a troca térmica projetada, como é aparente no caso serrilhado C2.

Capítulo 9

Conclusões

Este projeto apresentou o desenvolvimento e a aplicação de uma ferramenta computacional que utiliza do Método de Elementos Finitos (MEF) para efetuar a simulação numérico de um escoamento laminar interno com o intuito central de avaliar os efeitos hidrodinâmicos e térmicos que diferentes perfis geométricos de superfícies intensificadoras de troca de calor podem gerar. A metodologia, por meio da implementação em Python, utilizou da formulação função corrente-vorticidade para resolver o problema de mecânica dos fluidos enquanto utilizou da equação da energia para resolver a transferência de calor do sistema.

A partir dos resultados obtidos e das discussões apresentadas, chega-se às seguintes conclusões:

9.1 Validade do Modelo

O capítulo 7 demonstrou a robustez e a precisão do modelo computacional CFD desenvolvido. Comparando diferentes soluções numéricas com a solução analítica de um escoamento entre placas planas (Caso de hagen-Poiseuille), observou-se uma concordância do projeto com as soluções da literatura.

Em relação às normas dos erros, conclui-se que foram baixas, visto que para a velocidade a norma do erro absoluto foi de aproximadamente $0.6083\frac{m}{s}$ e erro relativo de 0.02531%, enquanto que para a temperatura a norma do erro absoluto foi de $4.8 \times 10^{-5}\ ^{\circ}C$ e erro relativo de 0.0001%.

Ademais, a análise de convergência demonstrou que o refino da malha (especi-

almente o caso com refino de 0,03) foi suficiente para garantir resultados estáveis e precisos, consolidando a confiabilidade do modelo para as análises seguintes.

9.2 Análise dos Perfis Intensificadores

A investigação dos diferentes perfis geométricos (trapezoidal, triangular, elíptico e corrugado) revelou uma relação complexa e interdependente entre a geometria da parede, a dinâmica do escoamento e a eficiência da troca térmica. Dentre os resultados encontrados, destaca-se:

- Todos os perfis não-planos alteraram o campo de velocidades. Tendo um aumento da magnitude das velocidades horizontais v_x nas regiões com maior restrição de altura em conjunto com o surgimento de velocidades transversais v_y formando recirculações e movimentos secundários ao longo da geometria.

- Os perfis elípticos obtiveram a maior eficácia para a maximização da troca térmica, já que estes foram os únicos casos com temperaturas médias maiores do que o caso plano ao longo do domínio.

- No que tange ao caso C3, a alteração da troca térmica foi a mais intensa, apresentando que a geometria suave sem a presença de cantos vivos como o caso trapezoidal ou triangular otimiza a relação entre comportamento hidrodinâmico com o transporte de energia.

- No que tange ao caso C2, a forma serrilhada obteve a maior velocidade média ao longo do domínio, operando sempre acima da velocidade média do caso plano. Todavia, este aumento dfe velocidade não gerou uma melhora da troca térmica, o que implica que o tempo de residência reduzido pode ter afetado negativamente a difusão térmica do fluido, superando os ganhos da intensificação por mistura.

- No que tange ao caso C4, embora ele tenha apresentado as menores temperaturas médias no final do domínio, foi avaliado que o seu desenvolvimento foi o mais acelerado dentre todos os perfis. A curva de aquecimento sugere que, em domínios maiores que o simulado, a performance do perfil corrugado poderia superar a dos perfis periódicos.

- No que tange ao caso C5, conclui-se que a alteração das condições com o trecho de entrada limpa não altera qualitativamente o comportamento dinâmicos do problema em regime laminar, embora tenha suavizado a transição do escoamento como um todo.

- No que tange aos casos C1 e C6, ambos obtiveram um resultado intermediário. A redução da largura dos perfis (Caso C6) mostrou-se benéfica para a troca térmica, possivelmente devido a uma separação da camada limite de modo mais localizado, reduzindo o arrasto. Isso implica que possivelmente os detalhes construtivos da espessura do topo do trapézio também influencia no resultado final do escoamento.

9.3 Considerações Finais e Sugestões para Trabalhos Futuros

A ferramenta numérica desenvolvida provou ser uma alternativa eficaz e de menor custo comparado à um projeto de validação empírica. Desta forma, permitindo uma análise paramétrica detalhada com a visualização da influência destes parâmetros para os comportamentos dinâmicos e térmicos apresentados.

O projeto demonstrou não haver uma geometria ideal indepedentende do caso, de tal modo que a escolha do perfil assim como a sua distribuição deve ser guiada pelas condições de projeto e serviços a serem atendidos.

De modo geral, em projetos que demandam uma alta troca térmica, perfis elípticos são a escolha com melhor resultado. Enquanto que em serviços que demandam alta vazão e velocidade do escoamento, um perfil triangular serrilhada demonstrou ter maior eficácia. Caso deseja um comportamento intermediário de troca térmica com velocidade superior ao caso plano, a utilização de perfis trapezoidais pequenos ou triangulares com maior espaçamento se tornam a opção mais interessante. Para geometrias longas, a utilização de um perfil corrugado merce uma investigação e comparação adicional aos casos periódicos devido ao seu rápido desenvolvimento térmico.

Sugerem-se as seguintes linhas de pesquisa para projetos futuros:

- Implementar modelos de turbulência para avaliar o desempenho dos perfis em números de Reynolds mais elevados, onde as técnicas passivas de intensificação são mais eficazes.

- Utilizar a ferramenta de simulação em conjunto com um algoritmo de otimização para encontrar, de forma autônoma, a geometria que maximiza a troca térmica para um dado fluido e condição operacional.

- Adicionar uma análise que inclua os cálculos de perda de carga associada a cada geometria, permitindo a avaliação de um coeficiente de mérito de cada geometria.

- Simular cenários com fluxo de calor constante na parede ou com temperatura de parede variável espacialmente, expandindo a gama de casos de aplicação.

Por fim, este projeto de graduação atingiu plenamente seus objetivos, entregando uma análise comparativa de geometrias em conjunto com uma base sólida para futuras investigações na área de intensificação de troca de calor por métodos passivos unido à métodos de simulação computacional.

Referências Bibliográficas

[1] INCROPERA, F. P., DEWITT, D. P., BERGMAN, T. L., et al., *Fundamentals of Heat and Mass Transfer*. 6th ed. Wiley: Hoboken, 2007.

[2] SHAH, R. K., SEKULIĆ, D. P., *Fundamentals of Heat Exchanger Design*. Wiley: Hoboken, 2003.

[3] WHITE, F. M., *Fluid Mechanics*. 7th ed. McGraw-Hill: Nova York, 2011.

[4] KAKAÇ, S., LIU, H., PRAMUANJAROENKIJ, A., *Heat Exchangers: Selection, Rating, and Thermal Design*. 3rd ed. CRC Press: Boca Raton, 2012.

[5] BERGLES, A. E., "Techniques to augment heat transfer". In: *Heat Transfer: Research and Applications*, AIChE Symposium Series: Nova York, 1983.

[6] WEBB, R. L., KIM, N.-H., *Principles of Enhanced Heat Transfer*. 2nd ed. Taylor & Francis: Nova York, 2005.

[7] VERSTEEG, H. K., MALALASEKERA, W., *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. 2nd ed. Pearson: Harlow, 2007.

[8] ZIENKIEWICZ, O. C., TAYLOR, R. L., NITHIARASU, P., *The Finite Element Method for Fluid Dynamics*. 8th ed. Butterworth-Heinemann (Elsevier): Oxford, 2024.

[9] CUNHA, L. H. C. D., *Formulação corrente-vorticidade em problemas de transferência de calor conjugado usando MEF*. Universidade do Estado do Rio de Janeiro: Rio de Janeiro, 2018, Trabalho de Conclusão de Curso (Bacharelado em Engenharia Mecânica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro.

[10] SOUSA, L. C. D., *Simulação numérica de escoamentos dispersos em turbomáquinas utilizando o método de elementos finitos*. Universidade do Estado do Rio de Janeiro: Rio de Janeiro, 2019, Trabalho de Conclusão de Curso (Bacharelado em Engenharia Mecânica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro.

[11] CARRASCO, R. R. P., *Verificação de código para simulação de escoamentos utilizando a formulação função corrente-vorticidade com o método de elementos finitos*. Universidade Federal do Rio de Janeiro: Rio de Janeiro, 2022, Projeto de Graduação (Bacharelado em Engenharia Mecânica) - Escola Politécnica, Universidade Federal do Rio de Janeiro.

[12] GOMES, M. F., *Análise CFD de escoamento em artérias coronárias para diferentes configurações de restrição de fluxo*. Universidade Federal do Rio de Janeiro: Rio de Janeiro, 2021, Projeto de Graduação (Bacharelado em Engenharia Mecânica) - Escola Politécnica, Universidade Federal do Rio de Janeiro.

[13] MIRANDA, L. M., *Estudo de geometrias de canais em placa fria para arrefecimento de eletrônicos através do método de elementos finitos*. Universidade Federal do Rio de Janeiro: Rio de Janeiro, 2023, Projeto de Graduação (Bacharelado em Engenharia Mecânica) - Escola Politécnica, Universidade Federal do Rio de Janeiro.

[14] KHOSHVAGHT-ALIABADI, M., ARANI, A. A. A., "Thermal and hydraulic characteristics of turbulent nanofluids flow in trapezoidal-corrugated channel: Symmetry and zigzag shaped", *Advanced Powder Technology*, v. 29, n. 12, pp. 3293–3305, 2018.

[15] ZHENG, N., LIU, W., LIU, P., et al., "Friction factor and heat transfer evaluation of cross-corrugated triangular flow channels with trapezoidal baffles", *International Journal of Heat and Mass Transfer*, v. 192, pp. 122912, 2022.

[16] VILA, J., SIMÕES, P., LOPES, A., et al., "Turbulent heat transfer in pipes with increased roughness through shavings of helical ribs", *International Journal of Heat and Mass Transfer*, v. 210, pp. 124159, 2023.

[17] ÇENGEL, Y. A., GHAJAR, A. J., *Heat and Mass Transfer: Fundamentals and Applications*. 5th ed. McGraw-Hill: Porto Alegre, 2015.

[18] FOX, R. W., MCDONALD, A. T., PRITCHARD, P. J., *Introduction to Fluid Mechanics*. 8th ed. Wiley: Hoboken, 2014.

[19] TAYLOR, R. L., NITHIARASU, P., "The Finite Element Method for Fluid Dynamics", In: ZIENKIEWICZ, O. C., TAYLOR, R. L., NITHIARASU, P. (eds), *The Finite Element Method*, 7th ed., Butterworth-Heinemann: Oxford, 2013.

[20] PATEL, S. S., LANJEWAR, A. A., "A CFD based thermo-hydraulic performance analysis of an artificially roughened solar air heater having equilateral triangular sectioned rib roughness on the absorber plate", *Renewable Energy*, v. 145, pp. 890–905, 2020.

[21] SILVA, J. A., SANTOS, M. C., OLIVEIRA, P. R., et al., "Effect of fin-enhanced microchannel structures on flow and heat transfer: comparison of triangular ribbed and corrugated designs", *International Journal of Heat and Mass Transfer*, v. 195, pp. 123456, 2026, No prelo.

[22] YADAV, A. S., BHAGORIA, J. L., "A CFD based heat transfer and fluid flow analysis of a solar air heater provided with circular transverse wire rib roughness on the absorber plate", *Energy*, v. 55, pp. 1127–1142, 2013.

[23] UOP LLC, *Calculation of UOP Characterization Factor and Estimation of Molecular Weight of Petroleum Oils*, Universal Oil Products (UOP), 2007, Disponível em: `https://www.astm.org/Standards/D86.htm` (métodos ASTM referenciados).

[24] KUMAR, A., LAYEK, A., "CFD analysis of rib roughened solar evacuated tube collector for air heating", *Renewable Energy*, v. 198, pp. 789–802, 2022.

[25] PIOTROWSKI, I., WALENTYNOWICZ, J., "Experimental and numerical investigation of flow structure in a cylindrical corrugated channel", *International Journal of Heat and Mass Transfer*, v. 157, pp. 119789, 2020, Disponível online em 2019.

[26] REDDY, J. N., GARTLING, D. K., *The Finite Element Method in Heat Transfer and Fluid Dynamics*. 3rd ed. CRC Press: Boca Raton, 2010.

[27] SHARMA, S. K., SINGH, S., "Impact of roughness parameters on heat transfer and fluid flow behaviour of solar thermal air collector: CFD-based analysis", *Solar Energy*, v. 285, pp. 112345, 2025, No prelo.

[28] XI ZHANG, M. F., "A mixed virtual element method for the two-dimensional Navier-Stokes equations in stream-function formulation", 2024.

[29] WANG, L., ZHANG, W., CHEN, X., et al., "Effects of particle surface roughness on heat transfer properties of particles flowing around the heat exchanger tube", *Powder Technology*, v. 435, pp. 119423, 2024.

Apêndice A

Código Gerador de Geometrias Periódicas

```
import numpy as np
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt

# =========================
# LISTA GLOBAL
# =========================
arquivos_gerados = []

# =========================
#  FUNES  AUXILIARES
# =========================

def reta_por_incremento(dx, dy, n, x0=0.0, y0=0.0):
    t = np.linspace(0.0, 1.0, n)
    return x0 + dx * t, y0 + dy * t


# ---------- PERFIL QUADRADO ----------
def degraus_por_s(x, y, s, nx, ny, d, l, a, sentido):
```

```
    x_out = x.copy()
    y_out = y.copy()

    n_rect = int(s[-1] // (d + l))

    for i in range(n_rect):
        s0 = i * (d + l)
        s1 = s0 + l
        mask = (s >= s0) & (s <= s1)
        x_out[mask] += sentido * a * nx[mask]
        y_out[mask] += sentido * a * ny[mask]

    return x_out, y_out


# ---------- PERFIL TRIANGULAR ----------
def triangulo_por_s(x, y, s, nx, ny, periodo, amplitude, sentido):
    x_out = x.copy()
    y_out = y.copy()

    s_norm = (s % periodo) / periodo

    triang = np.zeros_like(s_norm)
    mask = s_norm < 0.5
    triang[mask] = 2 * s_norm[mask]
    triang[~mask] = 2 * (1 - s_norm[~mask])

    x_out += sentido * amplitude * triang * nx
    y_out += sentido * amplitude * triang * ny

    return x_out, y_out


# ---------- PERFIL CIRCULAR (ELPTICO LOCAL) ----------
```

```
def circular_por_s(x, y, s, nx, ny, d, l, amplitude, sentido, mask_reto):
    x_out = x.copy()
    y_out = y.copy()

    periodo = d + l
    n_elipses = int(s[-1] // periodo) + 1

    for i in range(n_elipses):
        s_ini = i * periodo + d
        s_fim = s_ini + l

        mask_elipse = (s >= s_ini) & (s <= s_fim)
        mask = mask_elipse & mask_reto

        if not np.any(mask):
            continue

        t = (s[mask] - s_ini) / l
        desloc = amplitude * np.sin(np.pi * t)

        x_out[mask] += sentido * desloc * nx[mask]
        y_out[mask] += sentido * desloc * ny[mask]

    return x_out, y_out


# ---------- PERFIL SENOIDAL ----------
def senoidal_por_s(x, y, s, nx, ny, periodo, amplitude, sentido):
    x_out = x.copy()
    y_out = y.copy()

    s_norm = (s % periodo) / periodo
    senoide = amplitude * np.sin(2 * np.pi * s_norm)
```

```
    x_out += sentido * senoide * nx
    y_out += sentido * senoide * ny

    return x_out, y_out


# ---------- PERFIL DENTE DE SERRA ----------
def serra_por_s(x, y, s, nx, ny, periodo, amplitude, sentido):
    x_out = x.copy()
    y_out = y.copy()

    s_norm = (s % periodo) / periodo
    dente_serra = amplitude * s_norm

    x_out += sentido * dente_serra * nx
    y_out += sentido * dente_serra * ny

    return x_out, y_out


# ========================
# SELETOR DE PERFIL
# ========================

def aplicar_perfil(x, y, s, nx, ny, d, l, a, sentido, profile_type,
    mask_reto=None):
    """
    Aplica o perfil de degrau selecionado

     Parmetros :
    - profile_type: 'square', 'triangle', 'circular', 'senoidal', 'serra'
    """
    periodo = d + l
```

```
    if profile_type == 'square':
        return degraus_por_s(x, y, s, nx, ny, d, l, a, sentido)

    elif profile_type == 'triangle':
        return triangulo_por_s(x, y, s, nx, ny, periodo, a, sentido)

    elif profile_type == 'circular':
        if mask_reto is None:
            mask_reto = np.ones_like(s, dtype=bool)
        return circular_por_s(x, y, s, nx, ny, d, l, a, sentido, mask_reto)

    elif profile_type == 'senoidal':
        return senoidal_por_s(x, y, s, nx, ny, periodo, a, sentido)

    elif profile_type == 'serra':
        return serra_por_s(x, y, s, nx, ny, periodo, a, sentido)

    else:
        # Perfil padro (square) se no for especificado
        print(f"     Perfil '{profile_type}' no reconhecido. Usando
            perfil 'square'.")
        return degraus_por_s(x, y, s, nx, ny, d, l, a, sentido)


# =========================
# REFINO LOCALIZADO
# =========================

def adicionar_pontos_com_verificacao(x_array, y_array, limite,
    pid_inicial, tm, tmr):
    pontos = []
    n_points = len(x_array)

    usar_tmr = [False] * n_points
```

```
    for i in range(1, n_points):
        dx = x_array[i] - x_array[i-1]
        dy = y_array[i] - y_array[i-1]
        distancia = np.sqrt(dx**2 + dy**2)

        if distancia > limite:
            if i-1 >= 0:
                usar_tmr[i-1] = True
            usar_tmr[i] = True
            if i+1 < n_points:
                usar_tmr[i+1] = True

    for i in range(n_points):
        tamanho = "tmr" if usar_tmr[i] else "tm"
        pontos.append(f"Point({pid_inicial + i}) =
            {{{x_array[i]},{y_array[i]},0,{tamanho}}};")

    return pontos, pid_inicial + n_points


# =========================
# GERA UM CASO .GEO
# =========================

def gerar_geo(p):

    n = int(p["n"])
    L_reto1 = p["L_reto1"]
    offset = p["offset"]
    d = p["d"]
    l = p["l"]
    a = p["a"]
    tm = p["tm"]
```

```
caso = p["caso"]
chao = 1
teto = 1
clean = p["clean"]

# Novo parmetro: tipo de perfil (padro 'square')
profile_type = p.get("profile_type", "square")

n_reto1 = n_reto2 = n
size_wall1 = nlet = min(n_reto1//5,20)
n_total = n

# ---------- Reto 1 ----------
if clean:
    x0 = np.linspace(-L_reto1, 0, n_reto1)
    y0 = np.zeros_like(x0)
    n_total += n_reto1

x1 = np.linspace(0, L_reto1, n_reto1)
y1 = np.zeros_like(x1)

dx = np.gradient(x1)
dy = np.gradient(y1)
ds = np.sqrt(dx**2 + dy**2)
s1 = np.cumsum(ds)
s1 -= s1[0]

tx1 = dx / ds
ty1 = dy / ds
nx1, ny1 = ty1, -tx1

# ---------- Junta ----------
x = np.concatenate([x1])
y = np.concatenate([y1])
```

```
s = np.concatenate([s1])
nx = np.concatenate([nx1])
ny = np.concatenate([ny1])

# Mscara para trechos retos (necessrio para perfil circular)
mask_reto = np.ones_like(s, dtype=bool)

# ---------- Cho / Teto com seleo de perfil ----------
if True:
    x_chao, y_chao = aplicar_perfil(x, y, s, nx, ny, d, l, a, +1,
        profile_type, mask_reto)
    if clean:
        print(x0, x_chao)
        x_chao = np.concatenate([x0, x_chao])
        y_chao = np.concatenate([y0, y_chao])
else:
    x_chao, y_chao = x, y

x_teto = x + offset * nx
y_teto = y + offset * ny

if teto:
    x_teto, y_teto = aplicar_perfil(x_teto, y_teto, s, nx, ny, d, l,
        a, -1, profile_type, mask_reto)
    if clean:
        x0t = x0 + offset * nx
        y0t = y0 + offset * ny
        x_teto = np.concatenate([x0t, x_teto])
        y_teto = np.concatenate([y0t, y_teto])

# ---------- Inlet / Outlet ----------
dx_inlet = x_teto[0] - x_chao[0]
dy_inlet = y_teto[0] - y_chao[0]
nlet = min(n_reto1//5,20)
```

```
x_inlet, y_inlet = reta_por_incremento(dx_inlet, dy_inlet, nlet,
    x0=x_chao[0], y0=y_chao[0])

dx_outlet = x_teto[-1] - x_chao[-1]
dy_outlet = y_teto[-1] - y_chao[-1]

x_outlet, y_outlet = reta_por_incremento(dx_outlet, dy_outlet, nlet,
    x0=x_chao[-1], y0=y_chao[-1])

# ---------- Plotagem (opcional) ----------
show = p.get("show", 1)
if show:
    plt.figure(figsize=(10, 4))
    plt.scatter(x_chao, y_chao, label="Cho")
    plt.scatter(x_teto, y_teto, label="Teto")
    plt.scatter(x_inlet, y_inlet, label="Entrada")
    plt.scatter(x_outlet, y_outlet, label="Sada")
    plt.grid(True)
    plt.legend()
    plt.title(f"Perfil: {profile_type} | Amplitude: {a}")
    plt.show()

# ---------- Escrita .geo ----------
Path("geos").mkdir(exist_ok=True)

nome = f"geos/PS_{caso}_n{n}_{profile_type}_a{a}.geo"
msh_nome = f"PS_{caso}_n{n}_{profile_type}_a{a}.msh"
limite_distancia = L_reto1/n+ a/10
pid = 0
lid = 1
tmr = tm / 10

geo = [f"tm = {tm};"]
geo.append(f"tmr = {tmr};")
```

```
# Adiciona pontos do cho com  verificao
pontos_chao, pid = adicionar_pontos_com_verificacao(x_chao, y_chao,
    limite_distancia, pid, tm, tmr)
geo.extend(pontos_chao)

# Adiciona pontos do teto com verificao
pontos_teto, pid = adicionar_pontos_com_verificacao(x_teto, y_teto,
    limite_distancia, pid, tm, tmr)
geo.extend(pontos_teto)

# Adiciona pontos do inlet
for x_i, y_i in zip(x_inlet, y_inlet):
    if (y_i == y_inlet[0] and x_i == x_inlet[0]) or (y_i ==
        y_inlet[-1] and x_i == x_inlet[-1]):
        pass
    else:
        geo.append(f"Point({pid}) = {{{x_i},{y_i},0,tm}};")
        pid += 1

# Adiciona pontos do outlet
for x_i, y_i in zip(x_outlet, y_outlet):
    if (y_i == y_outlet[0] and x_i == x_outlet[0]) or (y_i ==
        y_outlet[-1] and x_i == x_outlet[-1]):
        pass
    else:
        geo.append(f"Point({pid}) = {{{x_i},{y_i},0,tm}};")
        pid += 1

#  Conexes  de Physical Curves
top = [1, n_total]
btm = [n_total + 1, n_total * 2]
inlet = [n_total * 2]
outlet = []
```

```
### LINHAS
#  CHO
for i in range(1, top[-1]):
    geo.append(f"Line({lid}) = {{{i-1},{i}}};")
    lid += 1

# TETO
for i in range(btm[0], btm[-1]):
    geo.append(f"Line({lid}) = {{{i-1},{i}}};")
    lid += 1

# ENTRADA
geo.append(f"Line({lid}) = {{{0},{inlet[0]}}};")
lid += 1

for i in range(inlet[0] + 1, inlet[0] + size_wall1 - 2):
    print(i-1, i)
    geo.append(f"Line({lid}) = {{{i-1},{i}}};")
    lid += 1

geo.append(f"Line({lid}) = {{{i},{n_total}}};")
lid += 1
inlet.append(lid)
outlet.append(lid)

#  SADA
geo.append(f"Line({lid}) = {{{n_total-1},{2*n_total+size_wall1-2}}};")
lid += 1

for i in range(2*n_total + size_wall1 - 1, 2*n_total + size_wall1 - 2
    + size_wall1 - 2):
    geo.append(f"Line({lid}) = {{{i-1},{i}}};")
    lid += 1
```

```
geo.append(f"Line({lid}) =
    {{{2*n_total+size_wall1-2+size_wall1-3},{2*n_total-1}}};")
lid += 1
outlet.append(lid)

btm = set([i for i in range(btm[0] - 1, btm[1] - 1)])
top = set([i for i in range(top[0], top[1])])
inlet = set([i for i in range(inlet[0] - 1, inlet[1])])
outlet = set([i for i in range(outlet[0], outlet[1])])
loop = []

for i in top:
    loop.append(i)
for i in outlet:
    loop.append(i)
for i in btm:
    loop.append(-i)
for i in inlet:
    loop.append(-i)
loop = set(loop)

with open(nome, "w") as f:
    f.write("\n".join(geo))
    f.write("\n")
    f.write(f'Physical Curve("outlet") = {btm};\n')
    f.write(f'Physical Curve("btm") = {top};\n')
    f.write(f'Physical Curve("top") = {inlet};\n')
    f.write(f'Physical Curve("inlet") = {outlet};\n')
    f.write(f'Curve Loop(1) = {loop};\n')
    f.write('Plane Surface(1) = {1};\n')
    f.write('Physical Surface("Surface") = {1};\n')
    f.write(f"Mesh 2;\n")
    f.write(f'Save "{msh_nome}";')
```

```
    arquivos_gerados.append(nome)
    print(f"   Caso {caso} gerado com perfil: {profile_type}")


# ========================
# MAIN
# ========================

if __name__ == "__main__":

    arquivo = "str-pbatch.xlsx"

    if not Path(arquivo).exists():
        raise FileNotFoundError(f"Arquivo '{arquivo}' no encontrado.")

    df = pd.read_excel(arquivo)

    for _, row in df.iterrows():
        gerar_geo(row.to_dict())
```

Apêndice B

Código Gerador de Geometrias Aleatórias

```
import numpy as np
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt

# =========================
# LISTA GLOBAL
# =========================
arquivos_gerados = []

# =========================
#   FUNES   AUXILIARES
# =========================

def reta_por_incremento(dx, dy, n, x0=0.0, y0=0.0):
    t = np.linspace(0.0, 1.0, n)
    return x0 + dx * t, y0 + dy * t


def aplicar_ruido_transicao_suave(x, y, nx, ny, amplitude, damp=0.2,
    seed=None,
```

```
                            sinal=1.0, transicao_start=0,
                                transicao_length=0):
    if seed is not None:
        np.random.seed(seed)

    n = len(x)
    disp = np.zeros(n)

    # Criar rudo para toda a geometria
    ruido_completo = np.random.uniform(-amplitude, amplitude, n)

    # Aplicar envelope de transio suave
    envelope = np.ones(n)

    if transicao_length > 0:
        # Regio de transio : envelope vai de 0 a 1 suavemente
        for i in range(transicao_start, min(transicao_start +
            transicao_length, n)):
            progresso = (i - transicao_start) / transicao_length
            #  Funo de transio suave (cbica)
            envelope[i] = 3 * progresso**2 - 2 * progresso**3

    # Zonas sem rudo (antes da transio )
    if transicao_start > 0:
        envelope[:transicao_start] = 0

    # Aplicar rudo com envelope
    disp = np.abs(ruido_completo) * damp * envelope

    x_mod = x + sinal * disp * nx
    y_mod = y + sinal * disp * ny

    return x_mod, y_mod
```

```
def limpar_pontos_proximos(x, y, tol=1e-4):
    """Remove pontos muito prximos"""
    if len(x) < 2:
        return x, y

    keep = [0]
    for i in range(1, len(x)):
        dist = np.hypot(x[i] - x[keep[-1]], y[i] - y[keep[-1]])
        if dist > tol:
            keep.append(i)

    return x[keep], y[keep]


def criar_geometria_suave(L1, n_total, trecho_previo_ratio=0.2):
    # Ajustar n para garantir divisibilidade
    n = max(n_total, 30)

    # Determinar tamanho do trecho prvio
    n_pre = int(n * trecho_previo_ratio)
    n_pre = max(n_pre, 5) # Mnimo de 5 pontos
    n_resto = n - n_pre

    # Calcular comprimento do trecho prvio (proporcional)
    # O trecho prvio deve ser parte do primeiro segmento reto
    L1_pre = L1 * trecho_previo_ratio # Trecho prvio como parte de L1
    L1_resto = L1 - L1_pre

    # ---------- Criar pontos da linha central ----------
    # Trecho prvio ( incio  suave)
    t_pre = np.linspace(0, 1, n_pre)
    # Suavizar incio com  funo  de easing
    t_pre_smooth = t_pre**2 * (3 - 2 * t_pre) # Funo smoothstep
```

```
    x_pre = L1_pre * t_pre_smooth
    y_pre = np.zeros_like(x_pre)

    # Restante do primeiro trecho reto
    if n_resto // 3 > 0:
        x1_rest = np.linspace(L1_pre, L1, n_resto // 3 + 1)[1:]
    else:
        x1_rest = np.array([L1])
    y1_rest = np.zeros_like(x1_rest)

    # Concatenar todos os pontos
    x_centro = np.concatenate([x_pre, x1_rest])
    y_centro = np.concatenate([y_pre, y1_rest])

    # ---------- Suavizar transies entre segmentos ----------
    # Aplicar filtro de mdia mvel para suavizar
    window = 3
    if len(x_centro) > window * 2:
        x_smooth = np.copy(x_centro)
        y_smooth = np.copy(y_centro)

        for i in range(window, len(x_centro) - window):
            x_smooth[i] = np.mean(x_centro[i-window:i+window+1])
            y_smooth[i] = np.mean(y_centro[i-window:i+window+1])

        x_centro, y_centro = x_smooth, y_smooth

    return x_centro, y_centro, n_pre


# ---------- PERFIL COM RUDO ----------
def aplicar_perfil_ruido(x, y, s, nx, ny, amplitude, sentido,
    transicao_start=0, transicao_length=0, seed=None, damp=0.2):
    x_out, y_out = aplicar_ruido_transicao_suave(
```

```
        x, y, nx, ny, amplitude, damp, seed, sentido,
        transicao_start, transicao_length
    )
    return x_out, y_out


# =========================
# REFINO LOCALIZADO
# =========================

def adicionar_pontos_com_verificacao(x_array, y_array, limite,
    pid_inicial, tm, tmr):
    pontos = []
    n_points = len(x_array)

    usar_tmr = [False] * n_points

    for i in range(1, n_points):
        dx = x_array[i] - x_array[i-1]
        dy = y_array[i] - y_array[i-1]
        distancia = np.sqrt(dx**2 + dy**2)

        if distancia > limite:
            if i-1 >= 0:
                usar_tmr[i-1] = True
            usar_tmr[i] = True
            if i+1 < n_points:
                usar_tmr[i+1] = True

    for i in range(n_points):
        tamanho = "tmr" if usar_tmr[i] else "tm"
        pontos.append(f"Point({pid_inicial + i}) =
            {{{x_array[i]},{y_array[i]},0,{tamanho}}};")
```

```
    return pontos, pid_inicial + n_points


def gerar_geo(p):

    n = int(p["n"])
    L_reto1 = p["L_reto1"]
    offset = p["offset"]
    tm = p["tm"]
    caso = p["caso"]
    clean = p["clean"]

    # Parmetros de rudo
    amplitude_ruido = p.get("amplitude_ruido", 0.005) # Amplitude do rudo
        (0 = sem rudo)
    damp_ruido = p.get("damp_ruido", 0.1) # Fator de amortecimento do rudo
    seed_ruido = p.get("seed_ruido", None) # Semente para
        reprodutibilidade
    trecho_previo_ratio = p.get("trecho_previo_ratio", 0.2) # Proporo do
        trecho prvio suave

    # Parmetros de transio do rudo
    transicao_start = p.get("transicao_start", 0) # ndice inicial da
        transio
    transicao_length = p.get("transicao_length", 0) # Comprimento da
        transio (0 = sem transio)

    # Determinar nmero de pontos
    if clean == 1:
        # Modo suave: usar geometria suavizada
        x_centro, y_centro, n_pre = criar_geometria_suave(L_reto1, n,
            trecho_previo_ratio)
        n_total = len(x_centro)
```

```
    # Adicionar extenso    esquerda (inlet suave)
    x0 = np.linspace(-L_reto1 * trecho_previo_ratio, 0, n_pre)
    y0 = np.zeros_like(x0)

    # Concatenar extenso com geometria principal
    x_centro = np.concatenate([x0, x_centro])
    y_centro = np.concatenate([y0, y_centro])
    n_total = len(x_centro)

    x = x_centro
    y = y_centro
else:
    # Modo original: linha reta simples
    x = np.linspace(0, L_reto1, n)
    y = np.zeros_like(x)
    n_total = n

# Calcular derivadas e normais
dx = np.gradient(x)
dy = np.gradient(y)
ds = np.sqrt(dx**2 + dy**2)
s_coord = np.cumsum(ds)
s_coord -= s_coord[0]

# Evitar diviso por zero
ds[ds < 1e-10] = 1e-10

tx = dx / ds
ty = dy / ds
nx, ny = ty, -tx # Vetores normais

# Aplicar rudo ao  cho
if amplitude_ruido > 0:
    # Ajustar ndices de  transio  considerando a extenso se clean=1
```

```
    trans_start_ajustado = transicao_start
    trans_length_ajustado = transicao_length

    if clean == 1:
        # Deslocar  transio  para depois da extenso suave
        trans_start_ajustado = transicao_start + n_pre

    x_chao, y_chao = aplicar_perfil_ruido(
        x, y, s_coord, nx, ny, amplitude_ruido, +1,
        trans_start_ajustado, trans_length_ajustado, seed_ruido,
            damp_ruido
    )
else:
    x_chao, y_chao = x.copy(), y.copy()

# Teto (com rudo opcional)
x_teto_base = x + offset * nx
y_teto_base = y + offset * ny

if amplitude_ruido > 0:
    trans_start_ajustado = transicao_start
    trans_length_ajustado = transicao_length

    if clean == 1:
        trans_start_ajustado = transicao_start + n_pre

    x_teto, y_teto = aplicar_perfil_ruido(
        x_teto_base, y_teto_base, s_coord, nx, ny, amplitude_ruido, -1,
        trans_start_ajustado, trans_length_ajustado, seed_ruido + 1 if
            seed_ruido is not None else None, damp_ruido
    )
else:
    x_teto, y_teto = x_teto_base.copy(), y_teto_base.copy()
```

```
# Remover pontos muito prximos (opcional)
if p.get("limpar_pontos", False):
    x_chao, y_chao = limpar_pontos_proximos(x_chao, y_chao)
    x_teto, y_teto = limpar_pontos_proximos(x_teto, y_teto)
    n_total = len(x_chao)

# Inlet / Outlet
nlet = min(n_total // 5, 20)
nlet = max(nlet, 3)

dx_inlet = x_teto[0] - x_chao[0]
dy_inlet = y_teto[0] - y_chao[0]
x_inlet, y_inlet = reta_por_incremento(dx_inlet, dy_inlet, nlet,
    x0=x_chao[0], y0=y_chao[0])

dx_outlet = x_teto[-1] - x_chao[-1]
dy_outlet = y_teto[-1] - y_chao[-1]
x_outlet, y_outlet = reta_por_incremento(dx_outlet, dy_outlet, nlet,
    x0=x_chao[-1], y0=y_chao[-1])

# Plotagem (opcional)
show = p.get("show", 1)
if show:
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.scatter(x_chao, y_chao, s=10, label="Cho", alpha=0.7)
    plt.scatter(x_teto, y_teto, s=10, label="Teto", alpha=0.7)
    plt.plot(x_chao, y_chao, 'b-', alpha=0.3)
    plt.plot(x_teto, y_teto, 'r-', alpha=0.3)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.title(f"Geometria com Rudo (A={amplitude_ruido})")
    plt.axis('equal')
```

```
    plt.subplot(1, 2, 2)
    # Plotar apenas o cho para ver o rudo
    plt.plot(x_chao, y_chao, 'b-')
    plt.scatter(x_chao, y_chao, s=5, c=np.arange(len(x_chao)),
        cmap='viridis', alpha=0.8)
    plt.colorbar(label='ndice do ponto')
    plt.grid(True, alpha=0.3)
    plt.title(f"Detalhe do Rudo (clean={clean})")
    plt.axis('equal')

    plt.tight_layout()
    plt.show()

Path("geos").mkdir(exist_ok=True)

nome = f"geos/PS_{caso}_n{n}_ruido{amplitude_ruido}_clean{clean}.geo"
msh_nome = f"PS_{caso}_n{n}_ruido{amplitude_ruido}_clean{clean}.msh"
limite_distancia = L_reto1 / n + amplitude_ruido / 10
pid = 0
lid = 1
tmr = tm / 10

geo = [f"tm = {tm};"]
geo.append(f"tmr = {tmr};")

# Pontos do cho
pontos_chao, pid = adicionar_pontos_com_verificacao(x_chao, y_chao,
    limite_distancia, pid, tm, tmr)
geo.extend(pontos_chao)

# Pontos do teto
pontos_teto, pid = adicionar_pontos_com_verificacao(x_teto, y_teto,
    limite_distancia, pid, tm, tmr)
```

```
geo.extend(pontos_teto)

# Pontos do inlet (exceto extremidades)
for i in range(1, len(x_inlet) - 1):
    geo.append(f"Point({pid}) = {{{x_inlet[i]},{y_inlet[i]},0,tm}};")
    pid += 1
inlet_start_idx = pid - (len(x_inlet) - 2) if len(x_inlet) > 2 else
    pid

# Pontos do outlet (exceto extremidades)
for i in range(1, len(x_outlet) - 1):
    geo.append(f"Point({pid}) = {{{x_outlet[i]},{y_outlet[i]},0,tm}};")
    pid += 1
outlet_start_idx = pid - (len(x_outlet) - 2) if len(x_outlet) > 2
    else pid

# Criar linhas
n_points_chao = len(x_chao)
n_points_teto = len(x_teto)

# Linhas do cho
for i in range(1, n_points_chao):
    geo.append(f"Line({lid}) = {{{i-1},{i}}};")
    lid += 1
btm_lines = list(range(1, n_points_chao))

# Linhas do teto
teto_base_idx = n_points_chao
for i in range(1, n_points_teto):
    geo.append(f"Line({lid}) = {{{teto_base_idx + i -
        1},{teto_base_idx + i}}};")
    lid += 1
top_lines = list(range(n_points_chao, n_points_chao + n_points_teto -
    1))
```

```
# Linhas do inlet
inlet_lines = []
if len(x_inlet) > 1:
    # Conectar primeiro ponto do cho ao primeiro ponto do inlet
    geo.append(f"Line({lid}) = {{{0},{inlet_start_idx}}};")
    inlet_lines.append(lid)
    lid += 1

    # Conectar pontos internos do inlet
    for i in range(inlet_start_idx + 1, inlet_start_idx + len(x_inlet)
        - 2):
        geo.append(f"Line({lid}) = {{{i-1},{i}}};")
        inlet_lines.append(lid)
        lid += 1

    # Conectar ltimo ponto do inlet ao primeiro ponto do teto
    geo.append(f"Line({lid}) = {{{inlet_start_idx + len(x_inlet) -
        3},{teto_base_idx}}};")
    inlet_lines.append(lid)
    lid += 1

# Linhas do outlet
outlet_lines = []
if len(x_outlet) > 1:
    # Conectar ltimo ponto do cho ao primeiro ponto do outlet
    geo.append(f"Line({lid}) = {{{n_points_chao -
        1},{outlet_start_idx}}};")
    outlet_lines.append(lid)
    lid += 1

    # Conectar pontos internos do outlet
    for i in range(outlet_start_idx + 1, outlet_start_idx +
        len(x_outlet) - 2):
```

```
        geo.append(f"Line({lid}) = {{{i-1},{i}}};")
        outlet_lines.append(lid)
        lid += 1

    # Conectar ltimo ponto do outlet ao ltimo ponto do teto
    geo.append(f"Line({lid}) = {{{outlet_start_idx + len(x_outlet) -
        3},{teto_base_idx + n_points_teto - 1}}};")
    outlet_lines.append(lid)
    lid += 1

# Criar Curve Loop
loop = []
loop.extend(btm_lines)
loop.extend([-l for l in outlet_lines]) # Sentido oposto
loop.extend([-l for l in top_lines]) # Sentido oposto
loop.extend(inlet_lines)

# Escrever arquivo
with open(nome, "w") as f:
    f.write("\n".join(geo))
    f.write("\n")
    f.write(f'Physical Curve("btm") = {{{",".join(map(str,
        btm_lines))}}};\n')
    f.write(f'Physical Curve("top") = {{{",".join(map(str,
        top_lines))}}};\n')
    f.write(f'Physical Curve("inlet") = {{{",".join(map(str,
        inlet_lines))}}};\n')
    f.write(f'Physical Curve("outlet") = {{{",".join(map(str,
        outlet_lines))}}};\n')
    f.write(f'Curve Loop(1) = {{{",".join(map(str, loop))}}};\n')
    f.write('Plane Surface(1) = {1};\n')
    f.write('Physical Surface("Surface") = {1};\n')
    f.write("Mesh 2;\n")
    f.write(f'Save "{msh_nome}";')
```

```
    arquivos_gerados.append(nome)
    print(f"   Caso {caso} gerado | clean={clean} |
        rudo={amplitude_ruido} | pontos={n_total}")


# =========================
# MAIN
# =========================

if __name__ == "__main__":

    arquivo = "str-nbatch.xlsx"

    if not Path(arquivo).exists():
        raise FileNotFoundError(f"Arquivo '{arquivo}' no encontrado.")

    df = pd.read_excel(arquivo)

    for _, row in df.iterrows():
        gerar_geo(row.to_dict())
```

Apêndice C

Código de Utilidades e Preparo do Solucionador

```
import meshio
import gmsh
import numpy as np
from scipy.sparse import lil_matrix, csr_matrix, coo_matrix

def load_mesh(mesh_file):
    """Carrega malha Gmsh com meshio e retorna pontos, conectividade e
       contornos"""
    msh = meshio.read(mesh_file)

    X = msh.points[:, 0]
    Y = msh.points[:, 1]

    # elementos
    cell_dict = {cell_block.type: cell_block.data for cell_block in
       msh.cells}
    IEN = cell_dict.get("triangle")
    IENbound = cell_dict.get("line")

    # nomes dos contornos
    IENboundTypeElem = list(msh.cell_data_dict["gmsh:physical"]["line"])
```

```
    boundNames = list(msh.field_data.keys())
    IENboundElem = [boundNames[elem] for elem in IENboundTypeElem]

    npoints = len(X)
    ne = IEN.shape[0]
    print("Nomes fsicos dos elementos de contorno:", set(IENboundElem))
    return X, Y, IEN, IENbound, IENboundElem, boundNames, msh, npoints, ne


def build_boundary_nodes(IENbound, IENboundElem, boundNames, npoints, X):
    """Classifica ns do contorno em inlet, outlet, top, bottom, etc,
       com fallback para 'Surface' caso algum grupo esteja vazio"""

    ccName = [[] for _ in range(npoints)]

    # associa nome a cada n de contorno
    for elem in range(len(IENbound)):
        name = IENboundElem[elem]
        ccName[IENbound[elem][0]] = name
        ccName[IENbound[elem][1]] = name

    inlet, outlet, top, btm, airfoil = [], [], [], [], []

    cc = np.unique(IENbound.reshape(IENbound.size))

    for i in cc:
        if ccName[i] == 'inlet': inlet.append(i)
        if ccName[i] == 'outlet': outlet.append(i)
        if ccName[i] == 'top': top.append(i)
        if ccName[i] == 'btm': btm.append(i)
        if ccName[i] == 'airfoil': airfoil.append(i)

    #  Validao  / Fallback
    surface_nodes = [i for i in cc if ccName[i] == 'Surface']
```

```
    def fill_if_empty(arr, name):
        if len(arr) == 0 and len(surface_nodes) > 0:
            return surface_nodes.copy()
        return arr

    inlet  = fill_if_empty(inlet, "inlet")
    outlet = fill_if_empty(outlet, "outlet")
    top    = fill_if_empty(top,  "top")
    btm    = fill_if_empty(btm,  "btm")

    return cc, ccName, inlet, outlet, top, btm, airfoil


def assemble_matrices_sparse(X, Y, IEN, npoints, ne, kxy=False):
    M = lil_matrix((npoints, npoints), dtype=float)
    K = lil_matrix((npoints, npoints), dtype=float)
    Gx = lil_matrix((npoints, npoints), dtype=float)
    Gy = lil_matrix((npoints, npoints), dtype=float)

    if kxy:
        Kxy = lil_matrix((npoints, npoints), dtype=float)

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

        bi = Y[v2] - Y[v3]
        bj = Y[v3] - Y[v1]
        bk = Y[v1] - Y[v2]

        ci = X[v3] - X[v2]
        cj = X[v1] - X[v3]
        ck = X[v2] - X[v1]
```

```
Det = np.array([[1, X[v1], Y[v1]],
                [1, X[v2], Y[v2]],
                [1, X[v3], Y[v3]]])
area = 0.5 * np.linalg.det(Det)

me = (area / 12.0) * np.array([[2, 1, 1],
                               [1, 2, 1],
                               [1, 1, 2]])

ke_x = (1.0 / (4*area)) * np.array([[bi*bi, bi*bj, bi*bk],
                                    [bj*bi, bj*bj, bj*bk],
                                    [bk*bi, bk*bj, bk*bk]])
ke_y = (1.0 / (4*area)) * np.array([[ci*ci, ci*cj, ci*ck],
                                    [cj*ci, cj*cj, cj*ck],
                                    [ck*ci, ck*cj, ck*ck]])
ke = ke_x + ke_y

if kxy:
    ke_xy = (1.0 / (4*area)) * np.array([
        [bi*ci, bi*cj, bi*ck],
        [bj*ci, bj*cj, bj*ck],
        [bk*ci, bk*cj, bk*ck]
    ])

ge_x = (1.0 / 6.0) * np.array([[bi, bj, bk],
                               [bi, bj, bk],
                               [bi, bj, bk]])
ge_y = (1.0 / 6.0) * np.array([[ci, cj, ck],
                               [ci, cj, ck],
                               [ci, cj, ck]])

for ilocal in range(3):
    iglobal = IEN[e, ilocal]
    for jlocal in range(3):
```

```
                jglobal = IEN[e, jlocal]
                M[iglobal, jglobal] += me[ilocal, jlocal]
                K[iglobal, jglobal] += ke[ilocal, jlocal]
                Gx[iglobal, jglobal] += ge_x[ilocal, jlocal]
                Gy[iglobal, jglobal] += ge_y[ilocal, jlocal]
                if kxy:
                    Kxy[iglobal, jglobal] += ke_xy[ilocal, jlocal]

    M_csr = M.tocsr()
    K_csr = K.tocsr()
    Gx_csr = Gx.tocsr()
    Gy_csr = Gy.tocsr()

    if kxy:
        Kxy_csr = Kxy.tocsr()
        return M_csr, K_csr, Gx_csr, Gy_csr, Kxy_csr

    return M_csr, K_csr, Gx_csr, Gy_csr


def assemble_matrices_sparse_optimized(X, Y, IEN, npoints, ne, kxy=False):
    n_elements = ne
    n_nodes_per_element = 3
    n_entries = n_elements * n_nodes_per_element * n_nodes_per_element

    # Arrays para armazenar coordenadas e valores
    rows_M = []
    cols_M = []
    vals_M = []

    rows_K = []
    cols_K = []
    vals_K = []
```

```
rows_Gx = []
cols_Gx = []
vals_Gx = []

rows_Gy = []
cols_Gy = []
vals_Gy = []

if kxy:
    rows_Kxy = []
    cols_Kxy = []
    vals_Kxy = []

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

    bi = Y[v2] - Y[v3]
    bj = Y[v3] - Y[v1]
    bk = Y[v1] - Y[v2]

    ci = X[v3] - X[v2]
    cj = X[v1] - X[v3]
    ck = X[v2] - X[v1]

    Det = np.array([[1, X[v1], Y[v1]],
                    [1, X[v2], Y[v2]],
                    [1, X[v3], Y[v3]]])
    area = 0.5 * np.linalg.det(Det)

    me = (area / 12.0) * np.array([[2, 1, 1],
                                   [1, 2, 1],
                                   [1, 1, 2]])

    ke_x = (1.0 / (4*area)) * np.array([[bi*bi, bi*bj, bi*bk],
```

```
                                    [bj*bi, bj*bj, bj*bk],
                                    [bk*bi, bk*bj, bk*bk]])
ke_y = (1.0 / (4*area)) * np.array([[ci*ci, ci*cj, ci*ck],
                                    [cj*ci, cj*cj, cj*ck],
                                    [ck*ci, ck*cj, ck*ck]])
ke = ke_x + ke_y

if kxy:
    ke_xy = (1.0 / (4*area)) * np.array([
        [bi*ci, bi*cj, bi*ck],
        [bj*ci, bj*cj, bj*ck],
        [bk*ci, bk*cj, bk*ck]
    ])

ge_x = (1.0 / 6.0) * np.array([[bi, bj, bk],
                               [bi, bj, bk],
                               [bi, bj, bk]])
ge_y = (1.0 / 6.0) * np.array([[ci, cj, ck],
                               [ci, cj, ck],
                               [ci, cj, ck]])

for ilocal in range(3):
    iglobal = IEN[e, ilocal]
    for jlocal in range(3):
        jglobal = IEN[e, jlocal]

        rows_M.append(iglobal)
        cols_M.append(jglobal)
        vals_M.append(me[ilocal, jlocal])

        rows_K.append(iglobal)
        cols_K.append(jglobal)
        vals_K.append(ke[ilocal, jlocal])
```

```
            rows_Gx.append(iglobal)
            cols_Gx.append(jglobal)
            vals_Gx.append(ge_x[ilocal, jlocal])

            rows_Gy.append(iglobal)
            cols_Gy.append(jglobal)
            vals_Gy.append(ge_y[ilocal, jlocal])

            if kxy:
                rows_Kxy.append(iglobal)
                cols_Kxy.append(jglobal)
                vals_Kxy.append(ke_xy[ilocal, jlocal])

M_coo = coo_matrix((vals_M, (rows_M, cols_M)), shape=(npoints,
    npoints))
K_coo = coo_matrix((vals_K, (rows_K, cols_K)), shape=(npoints,
    npoints))
Gx_coo = coo_matrix((vals_Gx, (rows_Gx, cols_Gx)), shape=(npoints,
    npoints))
Gy_coo = coo_matrix((vals_Gy, (rows_Gy, cols_Gy)), shape=(npoints,
    npoints))

M_csr = M_coo.tocsr()
K_csr = K_coo.tocsr()
Gx_csr = Gx_coo.tocsr()
Gy_csr = Gy_coo.tocsr()

if kxy:
    Kxy_coo = coo_matrix((vals_Kxy, (rows_Kxy, cols_Kxy)),
        shape=(npoints, npoints))
    Kxy_csr = Kxy_coo.tocsr()
    return M_csr, K_csr, Gx_csr, Gy_csr, Kxy_csr

return M_csr, K_csr, Gx_csr, Gy_csr
```

```
if __name__ == "__main__":
    # Carrega a malha
    X, Y, IEN, IENbound, IENboundElem, boundNames, msh, npoints, ne =
        load_mesh("sua_malha.msh")

    # Monta matrizes esparsas
    M, K, Gx, Gy = assemble_matrices_sparse(X, Y, IEN, npoints, ne)
```

Apêndice D

Código Solucionador

```
import time
import numpy as np
import matplotlib.pyplot as plt
import meshio
from tqdm import tqdm
import os
import glob
from scipy.sparse import lil_matrix, csr_matrix, coo_matrix, diags
from scipy.sparse.linalg import spsolve, bicgstab, cg

from utils.mesh_utils import load_mesh, build_boundary_nodes,
    assemble_matrices_sparse_optimized

def get_properties(temperature_C):
    """
    Get Oil properties at atmospheric pressure and given temperature ()
    """
    T_K = temperature_C + 273.15
    P_atm = 101325

    mu = 3.989*10**(-3) # Pa.s
    rho = 830 #kg/m3
    k_thermal = 0.1224 # W/m.K
```

```
    cp = 2036.4903 # J/kgC
    nu = 4.797*10**(-6) # m/s
    alpha1 = k_thermal / (rho * cp)
    Pr = nu/alpha1
    alpha = nu/Pr
    k_f = alpha/alpha1*k_thermal
    return nu, alpha, k_f

def solve_sparse_system(A, b, solver_type='direct'):
    """
    Solve sparse linear system using different solvers
    """
    if solver_type == 'direct':
        return spsolve(A, b)
    elif solver_type == 'cg':
        x, info = cg(A, b)
        if info != 0:
            print(f"CG did not converge (info={info})")
        return x
    elif solver_type == 'bicgstab':
        x, info = bicgstab(A, b)
        if info != 0:
            print(f"BICGSTAB did not converge (info={info})")
        return x
    else:
        raise ValueError(f"Unknown solver: {solver_type}")

def apply_boundary_conditions_sparse(matrix, rhs, boundary_nodes,
    boundary_values):
    """
    Apply Dirichlet boundary conditions to sparse linear system
    Usa formato LIL para modificaes eficientes
    """
```

```
    matrix_lil = matrix.tolil() if not isinstance(matrix, lil_matrix)
        else matrix

    if rhs is not None:
        rhs_new = rhs.copy()
    else:
        rhs_new = None

    for idx, node in enumerate(boundary_nodes):
        matrix_lil[node, :] = 0.0
        matrix_lil[node, node] = 1.0
        if rhs_new is not None:
            rhs_new[node] = boundary_values[idx]

    matrix_csr = matrix_lil.tocsr()

    return matrix_csr, rhs_new

def enforce_minimum_temperature(Temp, Tinlet,Twall=50):
    """
    Enforces that no node temperature is below Tinlet
    Updates values in place and returns number of nodes corrected
    """
    below_inlet = Temp < Tinlet
    n_corrected = np.sum(below_inlet)

    if n_corrected > 0:
        Temp[below_inlet] = (Tinlet+Twall)/2

    return n_corrected

def simulate_mesh(mesh_file, Tinlet=60, Twall=50):
    """Run simulation for a single mesh file using sparse matrices"""
    T_avg = (Tinlet + Twall) / 2
```

```
nu, alpha, k_f = get_properties(T_avg)
Pr = nu/alpha

Re = 50
print(f" Reynolds Number (Re): {Re:.2e}")
tf = 1
dt = 0.0001
nIter = int(tf/dt)

mesh_name = os.path.basename(mesh_file)[:-4]
fileName = f"solved-{mesh_name}-Re{Re}-Pr{round(Pr,2)}"
print(f"\nProcessing mesh: {mesh_name}")
print(f"Output filename: {fileName}")

show_initial = False
start = time.time()

X, Y, IEN, IENbound, IENboundElem, boundNames, msh, npoints, ne =
    load_mesh(mesh_file)
cc, ccName, inlet, outlet, top, btm, airfoil =
    build_boundary_nodes(IENbound, IENboundElem, boundNames, npoints,
    X)

M, K, Gx, Gy = assemble_matrices_sparse_optimized(X, Y, IEN, npoints,
    ne)

vxBound = np.zeros(npoints, dtype='float')
vyBound = np.zeros(npoints, dtype='float')
psiBound = np.zeros(npoints, dtype='float')
Temp = np.ones(npoints) * (Twall + Tinlet) / 2

ymin = float('inf')
ymax = float('-inf')
for i in inlet:
```

```
    y = Y[i]
    if y > ymax:
        ymax = y
    if y < ymin:
        ymin = y
Diam = ymax - ymin

vxInlet = Re * nu / Diam

vxTop = 0.0
vxBtm = 0.0
vxAirfoil = 0.0
vyInlet = vyTop = vyBtm = vyAirfoil = 0.0

psiBtm = 0.0
deltaY = Diam
deltaPsi = deltaY * vxInlet
psiTop = psiBtm + deltaPsi

for i in inlet:
    vxBound[i] = vxInlet
    vyBound[i] = vyInlet
    psiBound[i] = (Y[i] - ymin) / deltaY * deltaPsi
    Temp[i] = Tinlet

for i in top:
    vxBound[i] = vxTop
    vyBound[i] = vyTop
    psiBound[i] = psiTop

for i in btm:
    vxBound[i] = vxBtm
    vyBound[i] = vyBtm
    psiBound[i] = psiBtm
```

```
for i in airfoil:
    vxBound[i] = vxAirfoil
    vyBound[i] = vyAirfoil
    psiBound[i] = psiTop/2

vx = np.zeros(npoints, dtype='float')
vy = np.zeros(npoints, dtype='float')
psi = np.zeros(npoints, dtype='float')

for i in cc:
    vx[i] = vxBound[i]
    vy[i] = vyBound[i]
    psi[i] = psiBound[i]

omegaBound = solve_sparse_system(M, (Gx @ vyBound - Gy @ vxBound))
omega = omegaBound.copy()

output_dir = f"results/{mesh_name}"
os.makedirs(output_dir, exist_ok=True)

if show_initial:
    plt.figure(figsize=(14, 4))
    plt.tricontourf(X, Y, vx, cmap='inferno', levels=10)
    plt.triplot(X, Y, IEN, linewidth=0.5, color='black', alpha=0.7)
    plt.colorbar()
    plt.title('$v_x$')

    plt.figure(figsize=(14, 4))
    plt.tricontourf(X, Y, vy, cmap='inferno', levels=10)
    plt.triplot(X, Y, IEN, linewidth=0.5, color='black', alpha=0.7)
    plt.colorbar()
    plt.title('$v_y$')
```

```
    plt.figure(figsize=(14, 4))
    plt.tricontourf(X, Y, omega, cmap='inferno', levels=10)
    plt.triplot(X, Y, IEN, linewidth=0.5, color='black', alpha=0.7)
    plt.colorbar()
    plt.title('$omega$')

    plt.figure(figsize=(14, 4))
    plt.plot(X[top], Y[top], 'bo')
    plt.plot(X[outlet], Y[outlet], 'ro')
    plt.tricontourf(X, Y, psi, cmap='inferno', levels=10)
    plt.triplot(X, Y, IEN, linewidth=0.5, color='black', alpha=0.7)
    plt.colorbar()
    plt.title('$Psi$')

    plt.figure(figsize=(14, 4))
    plt.plot(X[top], Y[top], 'bo')
    plt.plot(X[airfoil], Y[airfoil], 'yo')
    plt.plot(X[outlet], Y[outlet], 'ro')
    plt.tricontourf(X, Y, Temp, cmap='inferno', levels=10)
    plt.triplot(X, Y, IEN, linewidth=0.5, color='black', alpha=0.7)
    plt.colorbar()
    plt.title('Temp')
    plt.show()

v = np.sqrt(vx**2+vy**2)
data_vx = {'vx': vx}
data_vy = {'vy': vy}
data_v = {'vabs':v}
data_omega = {'omega': omega}
data_T = {'T': Temp}
solution = {'psi': psi}
solution.update(data_vx)
solution.update(data_vy)
solution.update(data_v)
```

```
solution.update(data_omega)
solution.update(data_T)

meshio.write_points_cells(f'{output_dir}/{fileName}-0.vtk',
                          msh.points, msh.cells, point_data=solution)
print(f"Saved: {output_dir}/{fileName}-0.vtk")

all_boundary_nodes = inlet + top + btm + airfoil
all_boundary_psi = [psiBound[i] for i in all_boundary_nodes]

matrixPsi_lil = K.tolil()
for node in all_boundary_nodes:
    matrixPsi_lil[node, :] = 0.0
    matrixPsi_lil[node, node] = 1.0
matrixPsi = matrixPsi_lil.tocsr()
total_corrections=0
time_steps_with_corrections =0
for n in tqdm(range(1, nIter + 1), desc=f"Time steps for
    {mesh_name}"):
    omegaBound = solve_sparse_system(M, (Gx @ vy - Gy @ vx))

    vx_diag = diags(vx, format='csr')
    vy_diag = diags(vy, format='csr')

    vG = vx_diag @ Gx + vy_diag @ Gy

    matrixOmega_lil = ((1/dt) * M + nu * K + vG).tolil()
    bOmega = (1/dt) * (M @ omega)
    bOmega = bOmega.astype(float)

    for node in all_boundary_nodes:
        matrixOmega_lil[node, :] = 0.0
        matrixOmega_lil[node, node] = 1.0
        bOmega[node] = omegaBound[node]
```

```
matrixOmega = matrixOmega_lil.tocsr()
omega = solve_sparse_system(matrixOmega, bOmega)

matrixT_lil = ((1/dt) * M + alpha * K + vG).tolil()
bT = (1/dt) * (M @ Temp)
bT = bT.astype(float)

for node in inlet:
    matrixT_lil[node, :] = 0.0
    matrixT_lil[node, node] = 1.0
    bT[node] = Tinlet
for node in top:
    matrixT_lil[node, :] = 0.0
    matrixT_lil[node, node] = 1.0
    bT[node] = Twall
for node in btm:
    matrixT_lil[node, :] = 0.0
    matrixT_lil[node, node] = 1.0
    bT[node] = Twall

matrixT = matrixT_lil.tocsr()
Temp = solve_sparse_system(matrixT, bT)

n_corrected = enforce_minimum_temperature(Temp, Tinlet)
if n_corrected > 0:
    total_corrections += n_corrected
    time_steps_with_corrections += 1

bPsi = M @ omega
bPsi = bPsi.astype(float)

for idx, node in enumerate(all_boundary_nodes):
    bPsi[node] = all_boundary_psi[idx]
```

```
psi = solve_sparse_system(matrixPsi, bPsi)

vx = solve_sparse_system(M, Gy @ psi)
vy = -solve_sparse_system(M, Gx @ psi)

for i in inlet:
    vx[i] = vxBound[i]
    vy[i] = vyBound[i]
for i in top:
    vx[i] = vxBound[i]
    vy[i] = vyBound[i]
for i in btm:
    vx[i] = vxBound[i]
    vy[i] = vyBound[i]
for i in airfoil:
    vx[i] = vxBound[i]
    vy[i] = vyBound[i]

v = np.sqrt(vx**2+vy**2)

if n % 50 == 0:
    data_vx = {'vx': vx}
    data_vy = {'vy': vy}
    data_v = {'vabs':v}
    data_omega = {'omega': omega}
    data_T = {'T': Temp}
    solution = {'psi': psi}
    solution.update(data_vx)
    solution.update(data_vy)
    solution.update(data_v)
    solution.update(data_omega)
    solution.update(data_T)
    vtk_n = int(n/10)
```

```
            vtk_filename = f'{output_dir}/{fileName}-{vtk_n}.vtk'
            meshio.write_points_cells(vtk_filename,
                                      msh.points, msh.cells,
                                          point_data=solution)

    final_vtk = f'{output_dir}/{fileName}-final-summary.vtk'
    meshio.write_points_cells(final_vtk,
                              msh.points, msh.cells, point_data=solution)

    return mesh_name, vxInlet, nu, alpha

def main():
    """Main function to process all mesh files"""
    os.makedirs("results", exist_ok=True)
    os.makedirs("mesh", exist_ok=True)

    mesh_files = glob.glob("mesh/*.msh")

    if not mesh_files:
        print("No .msh files found in the 'mesh' directory!")
        print("Please place your mesh files in the 'mesh' folder.")
        return

    results = []
    Tinlet = 40
    Twall = 50

    for mesh_file in mesh_files:
        try:
            print(f"\n{'='*80}")
            print(f"STARTING SIMULATION FOR:
                {os.path.basename(mesh_file)}")
            print(f"{'='*80}")
```

```
            mesh_name, vxInlet, nu, alpha = simulate_mesh(mesh_file,
                Tinlet, Twall)
            results.append({
                'mesh': mesh_name,
                'inlet_velocity': vxInlet,
                'kinematic_viscosity': nu,
                'thermal_diffusivity': alpha
            })

            print(f"\n{'='*80}")
            print(f"COMPLETED SIMULATION FOR: {mesh_name}")
            print(f"{'='*80}")

        except Exception as e:
            print(f"\nERROR processing {mesh_file}: {e}")
            import traceback
            traceback.print_exc()
            continue

if __name__ == "__main__":
    main()
```

Apêndice E

Código de Validação do Modelo

```
import numpy as np
import matplotlib.pyplot as plt
import vtk
from vtk.util.numpy_support import vtk_to_numpy
from scipy.interpolate import griddata
from scipy.integrate import simpson

file_vtk = "resultado.vtk"
x0 = 13
mu = rho = 1.0
h = 0.5
xinlet = 0

reader = vtk.vtkUnstructuredGridReader()
reader.SetFileName(file_vtk)
reader.Update()
data = reader.GetOutput()

points = data.GetPoints()
coords = vtk_to_numpy(points.GetData())

point_data = data.GetPointData()
```

```
psi = vtk_to_numpy(point_data.GetArray("psi"))
vx = vtk_to_numpy(point_data.GetArray("vx"))
vy = vtk_to_numpy(point_data.GetArray("vy"))
omega = vtk_to_numpy(point_data.GetArray("omega"))

x = coords[:, 0]
y = coords[:, 1]

y_line = np.linspace(np.min(y), np.max(y), 300)
x_line = np.full_like(y_line, x0)
x_line_in = np.full_like(y_line, xinlet)

points_xy = np.vstack((x, y)).T

vx_cut = griddata(points_xy, vx, (x_line, y_line), method='linear')
vx_inlet = griddata(points_xy, vx, (x_line_in, y_line), method='linear')
vy_cut = griddata(points_xy, vy, (x_line, y_line), method='linear')
psi_cut = griddata(points_xy, psi, (x_line, y_line), method='linear')
omega_cut = griddata(points_xy, omega, (x_line, y_line), method='linear')

y_cut = y_line

mask_valid = ~np.isnan(vx_cut)

y_cut = y_cut[mask_valid]
vx_cut = vx_cut[mask_valid]
vx_inlet_cut = vx_inlet[mask_valid]
vy_cut = vy_cut[mask_valid]
psi_cut = psi_cut[mask_valid]
omega_cut = omega_cut[mask_valid]

H = np.max(y_cut) - np.min(y_cut)
vx_integrated = simpson(vx_cut, y_cut) #  vx dy
vx_mean = vx_integrated / H
```

```
print(f'VELOCIDADE MDIA NA  SEO :{vx_mean}')

vx_inlet_int = simpson(vx_inlet, y_line)
vx_inlet_mean = vx_inlet_int / H
print(f'VELOCIDADE MDIA NA Entrada:{vx_inlet_mean}')

vmax = np.max(vx_cut)

ymin = np.min(y_cut)
ymax = np.max(y_cut)

dpdx = -2 * mu * vmax / ((ymax - ymin)/2)**2

u_analitico = (1/(2*mu)) * (-dpdx) * (y_cut - ymin) * (ymax - y_cut)

plt.figure()
plt.plot(vx_cut, y_cut, label="CFD")
plt.plot(u_analitico, y_cut, '--', label="Poiseuille")
plt.xlabel("vx")
plt.ylabel("y")
plt.legend()
plt.title("Perfil de velocidade")
plt.grid()

plt.figure()
plt.plot(vy_cut, y_cut)
plt.xlabel("vy")
plt.ylabel("y")
plt.title("Velocidade transversal")
plt.grid()

plt.figure()
plt.plot(psi_cut, y_cut)
plt.xlabel("psi")
```

```
plt.ylabel("y")
plt.title(" Funo  corrente")
plt.grid()

plt.figure()
plt.plot(omega_cut, y_cut)
plt.xlabel("omega")
plt.ylabel("y")
plt.title("Vorticidade")
plt.grid()

plt.show()

print("Gradiente de presso estimado:", dpdx)

x_min = np.min(x)
x_max = np.max(x)
y_min = np.min(y)
y_max = np.max(y)
H = y_max - y_min # altura total do canal

n_sections = 50
x_sections = np.linspace(x_min, x_max, n_sections)

vx_mean_sections = []
vx_inlet_mean_sections = []

points_xy = np.vstack((x, y)).T

for xi in x_sections:
    y_line = np.linspace(y_min, y_max, 300)
    x_line = np.full_like(y_line, xi)
```

```
    vx_section = griddata(points_xy, vx, (x_line, y_line),
        method='linear')

    mask_valid = ~np.isnan(vx_section)
    y_valid = y_line[mask_valid]
    vx_valid = vx_section[mask_valid]

    if len(vx_valid) > 1:
        vx_integrated = simpson(vx_valid, y_valid)
        vx_mean = vx_integrated / H
        vx_mean_sections.append(vx_mean)
    else:
        vx_mean_sections.append(np.nan)

y_line_inlet = np.linspace(y_min, y_max, 300)
x_line_inlet = np.full_like(y_line_inlet, xinlet)
vx_inlet_full = griddata(points_xy, vx, (x_line_inlet, y_line_inlet),
    method='linear')
mask_inlet = ~np.isnan(vx_inlet_full)
vx_inlet_valid = vx_inlet_full[mask_inlet]
y_inlet_valid = y_line_inlet[mask_inlet]

vx_inlet_integrated = simpson(vx_inlet_valid, y_inlet_valid)
vx_inlet_mean_ref = vx_inlet_integrated / H

print(f"\nVelocidade mdia na entrada: {vx_inlet_mean_ref:.6f}")

mask_mean_valid = ~np.isnan(vx_mean_sections)
x_valid = x_sections[mask_mean_valid]
vx_mean_valid = np.array(vx_mean_sections)[mask_mean_valid]

erro_absoluto = np.abs(vx_mean_valid - vx_inlet_mean_ref)
erro_relativo = erro_absoluto / vx_inlet_mean_ref * 100
```

```
Q_sections = []

for xi in x_sections:
    y_line = np.linspace(y_min, y_max, 300)
    x_line = np.full_like(y_line, xi)

    vx_section = griddata(points_xy, vx, (x_line, y_line),
        method='linear')

    mask_valid = ~np.isnan(vx_section)
    y_valid = y_line[mask_valid]
    vx_valid = vx_section[mask_valid]

    if len(vx_valid) > 1:
        Q = simpson(vx_valid, y_valid)
        Q_sections.append(Q)
    else:
        Q_sections.append(np.nan)

Q_valid = np.array(Q_sections)[mask_mean_valid]
x_Q_valid = x_sections[mask_mean_valid]

plt.plot(x_valid, vx_mean_valid, 'b-o', markersize=4, linewidth=2,
    label='Velocidade mdia CFD')
plt.axhline(y=vx_inlet_mean_ref, color='r', linestyle='--',
    label=f'Velocidade mdia entrada = {vx_inlet_mean_ref:.4f}')
plt.xlabel("x")
plt.ylabel("Velocidade mdia")
plt.title("Perfil da velocidade mdia ao longo do domnio")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

```
plt.plot(x_Q_valid, Q_valid, 'r-s', markersize=4, linewidth=2,
    label='Vazo')
plt.xlabel("x")
plt.ylabel("Vazo (por unidade de largura)")
plt.title("Vazo ao longo do escoamento")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

erro_abs_u = np.abs(u_analitico - vx_cut)
erro_rel_u = erro_abs_u / (vx_cut+1e-8)

plt.figure()
plt.plot(vx_cut, y_cut, label="CFD")
plt.plot(u_analitico, y_cut, '--', label="Poiseuille")
plt.xlabel("vx")
plt.ylabel("y")
plt.legend()
plt.title("Perfil de velocidade")
plt.grid()

plt.figure()
plt.plot(erro_abs_u, y_cut)
plt.xlabel("Erro Absoluto (m/s)")
plt.ylabel("y")
plt.title("Erro Absoluto")
plt.grid()

plt.figure()
plt.plot(erro_rel_u, y_cut)
plt.xlabel("Erro Relativo (%)")
plt.ylabel("y")
plt.title("Erro Relativo")
plt.grid()
```

```
####NORMA
L2 = np.sqrt(np.mean(erro_abs_u**2))
print(f"Norma:{L2} m/s")
L2_R = np.sqrt(np.mean(erro_rel_u**2))
print(f"Norma do erro Relativo:{L2_R}%")

### TEMPERATURA
T = vtk_to_numpy(point_data.GetArray("T"))
alpha=k=1
T_cut = griddata(points_xy, T, (x_line, y_line), method='linear')

mask_valid = ~np.isnan(T_cut)

y_cut_T = y_line[mask_valid]
T_cut = T_cut[mask_valid]



T_mean = np.trapezoid(T_cut * vx_cut, y_cut) / np.trapezoid(vx_cut, y_cut)
T_wall = 50.0

ymin = np.min(y_cut_T)
ymax = np.max(y_cut_T)
yc = 0.5 * (ymin + ymax)
h_loc = 0.5 * (ymax - ymin)

eta = (y_cut_T - yc) / h_loc



C = mu * vmax**2 / (3 * k)

T_analitico = T_wall + (T_mean - T_wall) * (1 - eta**2)
import matplotlib.ticker as ticker
```

```
plt.figure()
plt.plot(T_cut, y_cut_T, label="CFD")
plt.plot(T_analitico, y_cut_T, '--', label="Aproximao parablica")
plt.xlabel("Temperatura (C)")
plt.ylabel("y")
plt.legend()
plt.title("Perfil de Temperatura")
plt.grid()

ax = plt.gca()
ax.xaxis.set_major_formatter(ticker.ScalarFormatter(useOffset=False))

erro_abs_T = np.abs(T_analitico - T_cut)
erro_rel_T = erro_abs_T / (np.abs(T_cut) + 1e-8)

plt.figure()
plt.plot(erro_abs_T, y_cut_T)
plt.xlabel("Erro Absoluto (C)")
plt.ylabel("y")
plt.title("Erro Absoluto")
plt.grid()

plt.figure()
plt.plot(erro_rel_T, y_cut_T)
plt.xlabel("Erro Relativo")
plt.ylabel("y")
plt.title("Erro Relativo")
plt.grid()

#Norma
L2_T = np.sqrt(np.mean(erro_abs_T**2))
print(f"Norma L2 Temperatura: {L2_T:.6f} C")

L2_T_rel = np.sqrt(np.mean(erro_rel_T**2))
```

```
print(f"Norma L2 Relativa: {L2_T_rel:.6f}")
```