Universidade Federal
do Rio de Janeiro
Escola Politécnica

SIMULAÇÃO DOS COEFICIENTES DE ARRASTO E SUSTENTAÇÃO PARA UM ESCOAMENTO LIVRE USANDO A FORMULAÇÃO CORRENTE-VORTICIDADE

João Salgueiro Lage

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.

Orientadores: Gustavo Rabello dos Anjos
Daniel Barbedo Vasconcelos Santos

Rio de Janeiro
Julho de 2022

SIMULAÇÃO DOS COEFICIENTES DE ARRASTO E SUSTENTAÇÃO PARA UM ESCOAMENTO LIVRE USANDO A FORMULAÇÃO CORRENTE-VORTICIDADE

João Salgueiro Lage

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO 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.

Examinado por:

Prof. Gustavo Rabello dos Anjos, Ph.D.

Daniel Barbedo Vasconcelos Santos, M.Sc.

Prof. Daniel Alves Castello, D.Sc.

Prof. Fábio da Costa Figueiredo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2022

Salgueiro Lage, João

Simulação dos Coeficientes de Arrasto e Sustentação para um Escoamento Livre usando a Formulação Corrente-Vorticidade/João Salgueiro Lage. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2022.

XI, 82 p.: il.; 29, 7cm.

Orientadores: Gustavo Rabello dos Anjos

Daniel Barbedo Vasconcelos Santos

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia Mecânica, 2022.

Referências Bibliográficas: p. 60 – 61.

1. Formulação Corrente-Vorticidade. 2. Método de Elementos Finitos. 3. Dinâmica de Fluidos Computacional. I. Rabello dos Anjos, Gustavo *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia Mecânica. III. Título.

A minha namorada, que esteve
comigo durante toda esta
jornada.

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

SIMULAÇÃO DOS COEFICIENTES DE ARRASTO E SUSTENTAÇÃO PARA UM ESCOAMENTO LIVRE USANDO A FORMULAÇÃO CORRENTE-VORTICIDADE

João Salgueiro Lage

Julho/2022

Orientadores: Gustavo Rabello dos Anjos
Daniel Barbedo Vasconcelos Santos

Curso: Engenharia Mecânica

RESUMO

Este projeto de graduação teve por objetivo a elaboração de um algoritmo de em Python para realizar a análise de um escoamento livre ao redor de uma geometria e ser capaz de pós processar os resultados a fim de calcular os Coeficientes de Arrasto e Sustentação. Para isso, foi adotado o Método de Elementos Finitos para a discretização espacial e temporal das equações que compõem a formulação Corrente-Vorticidade, abordagem esta derivada da Equação de Navier-Stokes. Para validar o código desenvolvido, casos já amplamente estudados na literatura, fosse por cálculo analítico ou por outras simulações validadas, serviram de base de alta confiabilidade para comparação dos resultados calculados com a metodologia adotada. Uma vez validado, o código foi usado para simular três casos, distintos um dos outros, para a realização de uma análise comparativa destes. O modelo adotado se mostrou satisfatório quanto à natureza física esperada para cada caso. Assim sendo, este trabalho alcançou seu objetivo.

Palavras-chave: Formulação Corrente-Vorticidade, Método de Elementos Finitos, Fluidodinâmica Computacional.

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

SIMULATION OF LIFT AND DRAG COEFFICIENTS FOR A FREE FLOW USING THE STREAM-VORTICITY FORMULATION

João Salgueiro Lage

July/2022

Advisors: Gustavo Rabello dos Anjos
Daniel Barbedo Vasconcelos Santos

Course: Mechanical Engineering

ABSTRACT

This graduation project aimed to develop a Python algorithm to perform the analysis of a free flow around a geometry and be able to post-process the results in order to calculate the Drag and Lift Coefficients. For this, the Finite Element Method was adopted for the spatial and temporal discretization of the equations that compose the Stream-Vorticity formulation, an approach derived from the Navier-Stokes equation. To validate the developed code, cases already widely studied in the literature, either by analytical calculation or by other already validated simulations, served as a high reliability basis for comparing the results calculated by the adopted methodology. Once validated, the code was used to simulate three cases, each distinct from the others, to carry out a comparative analysis of these. The model adopted proved to be satisfactory in terms of the physical nature expected for each case. Therefore, this work achieved its objective.
Keywords: Streamfunction-Vorticity Formulation, Finite Elements Method, Computacional Fluid Dynamics.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Fluidos e seus comportamentos são parte fundamental de diversos aspectos tanto do cotidiano quanto da pesquisa e da indústria. Dinâmica dos fluidos afeta vários setores do desenvolvimento tecnológico, seja no que toca o resfriamento de equipamentos e componentes, lubrificação, escoamento em dutos dos mais diversos fluidos, ventiladores e bombas, máquinas hidráulicas em geral e, mais intensivamente, a indústria petrolífera que trabalha com equipamentos submersos e a indústria de meios de transporte que está constantemente tentando produzir veículos que consigam se deslocar cortando o ar ou água da forma mais eficiente o possível. Estes desenvolvimentos no campo da dinâmica dos fluidos são movidos pelo interesse em ganho de performance e economia. Reduzir a resistência aerodinâmica, por exemplo, significa menos energia gasta com estas forças dissipativas.

No entanto, estudar e aplicar o comportamento de fluidos em pesquisas e projetos não é uma tarefa simples. Toda a formulação de mecânica dos fluidos existente é relativamente nova e em parte muito complicada, envolvendo muita correlação experimental e matemática muito complexa. Por isso, para estudar o comportamento de um fluido durante um projeto, era muito comum o uso de experimentos em pequena escala que representavam de forma fiel o comportamento em escala real. Porém, a realização destes costuma ser trabalhosa e custosa, dado o uso de diversos equipamentos, sensores, ambientes especializados, mão de obra e tempo de preparo.

Felizmente, graças aos vários progressos na área da computação, computadores cada vez mais potentes oferecem uma alternativa aos complexos ensaios físicos necessários em projetos e pesquisas de engenharia. Aliados a diversos métodos numéricos que surgiram nos séculos XX e XXI, simulações computacionais são capazes de calcular com precisão o comportamento do fluido em uma vasta gama de situações de engenharia.

Estes métodos numéricos implementados por meio de algoritmos computacionais para resolver problemas de fluídica são chamados em inglês de *Computational Fluid Dynamics* ou CFD. Dentre os métodos utilizados, dois se destacam. São eles o Método de Elementos Finitos e o Método de Volumes Finitos. Neste trabalho foi optado por utilizar um desenvolvimento por Método de Elementos Finitos para a elaboração de um algoritmo capaz de resolver problemas bidimensionais de escoamentos de fluidos.

1.1 Introdução à Revisão Bibliográfica e ao Método Adotado

Para tais escoamentos, algumas hipóteses serão desenvolvidas e a formulação matemática será exibida. Serão abordados fluidos monofásicos, incompressíveis e newtonianos, descritos pela formulação Corrente-Vorticidade, advinda das Equações de Navier-Stokes em duas dimensões. Tal formulação simplifica o cálculo uma vez que ela torna o termo da velocidade independente do termo da pressão, reduzindo a quantidade de grandezas a serem calculadas.

Para a adoção do Método de Elementos Finitos, faz-se necessário discretizar o domínio e a evolução temporal. A dicretização espacial do domínio será feita pelo método dos resíduos ponderados por meio do Método de Galerkin enquanto que, por sua vez, a discretização temporal será feita pelo método de diferenças finitas de primeira ordem.

1.2 Introdução à Validação Numérica

No entanto, para concluir que o método está funcionando corretamente, é necessário fazer uma validação numérica do algoritmo adotado e, para isso, dois casos mais simples e cuja solução é conhecida foram empregados. Primeiramente, um escoamento simples entre placas planas fixas e paralelas (Escoamento de Hagen-Poiseuille) e o escoamento em cavidade com tampa deslizante (Escoamento Lid-Driven). Para ambos, a solução é conhecida, seja pela elaboração da solução analítica ou seja por trabalhos anteriores realizados sobre o assunto.

1.3 Introdução ao Ferramental

E, para implementar e analisar os resultados, uma série de ferramentas computacionais é necessária. Primeiramente, para a execução do algoritmo, foi usada a linguagem de programação Python. Python é uma linguagem relativamente simples e de fácil uso porém muito versátil e com diversas bibliotecas prontas para uso. Bibliotecas essas capazes de realizar de forma eficiente operações de álgebra linear (*Numpy*) e de se comunicar com outros softwares, como é o caso da leitura dos arquivos de malha realizada pela biblioteca *Meshio*. Para a criação de malhas, foi usado o software *Gmsh*, capaz de criar malhas de todos os tipos para qualquer geometria, seguindo as mais diversas configurações adotadas pelo usuário. Para o pós-tratamento dos resultados, foi usado o software *Paraview*, que possui uma gama de ferramentas para analisar os mais diversos aspectos do escoamento.

1.4 Introdução ao Problema Proposto

Finalmente, este trabalho visa utilizar todo esse ferramental mencionado para calcular arrasto e sustentação de um corpo imerso em um escoamento em diferentes valores de Reynolds. Uma geometria relativamente simples, um cilindro, será adotada visto que a influência da geometria não entra no escopo deste trabalho. Para esta configuração, serão analisados três casos distintos de escoamento, cada um com um valor de Reynolds.

O objetivo final deste trabalho se trata do cálculo, por meio de todo o ferramental de métodos numéricos, do coeficiente de sustentação e de arrasto para um corpo em escoamento livre.

O coeficiente de sustentação C_L é uma grandeza adimensional que expressa a relação entre a força de sustentação e a densidade e velocidade ao redor de um dado corpo imerso em um escoamento. Este coeficiente, para escoamentos incompressíveis, é função da geometria do corpo e do número de Reynolds.

Semelhantemente, o coeficiente de arrasto C_D é um valor adimensional que relaciona a força de arrasto produzida por um escoamento ao redor de um corpo com a densidade e a velocidade do escoamento. Assim como o coeficiente de sustentação, em um escoamento incompressível, é função da geometria e o número de Reynolds.

Ambos os valores são amplamente utilizados pela indústria, principalmente a aeronáutica, e por isso são de grande importância para a realização de projetos de aeronaves e equipamentos que envolvem escoamentos de fluidos com altos valores de Reynolds.

Neste trabalho, serão apresentados desenvolvimentos e resultados para uma geometria relativamente simples. Será utilizado um cilindro em um escoamento livre. Por se tratar de um desenvolvimento adimensional, todas as medidas serão dadas em relação ao diâmetro D do cilindro.

Como ilustrado na figura abaixo, determinados comportamentos serão esperados do escoamento dependendo do valor de Reynolds utilizado.

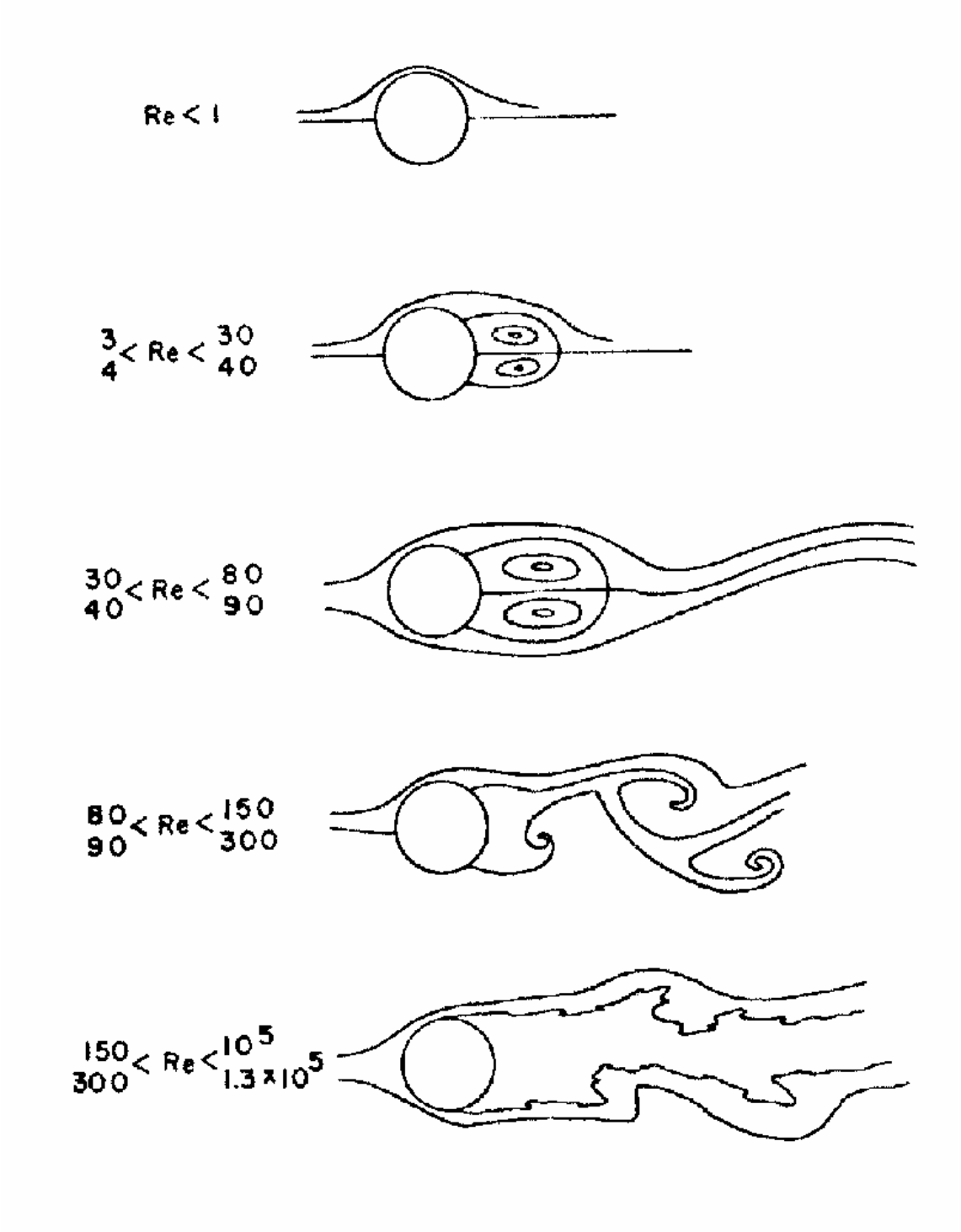


Figura 1.1: Exemplos de Escoamentos Livres ao redor de um cilindro para diferentes valores de Re (Harder, 2012)

1.5 Organização do Trabalho

Este trabalho está organizado em uma estrutura de capítulos. O capítulo 1 introduz o objetivo final deste trabalho. O capítulo 2 faz a revisão bibliográfica da literatura necessária para o desenvolvimento de toda a formulação utilizada. O capítulo 3 introduz o método de Elementos Finitos, principal ferramenta deste trabalho. O capítulo 4 explicita a metodologia adotada para a obtenção dos resultados. O capítulo 5 realiza a validação numérica do algoritmo e demais softwares utilizados. O capítulo 6 apresenta a formulação do problema proposto e seus resultados, bem como uma análise deles. Em seguida, é apresentado um capítulo de conclusão sobre o trabalho, apresentando reflexões e pontos a serem evoluídos futuramente. Finalmente, os anexos contendo todos os materiais pertinentes ao trabalho e as referências são apresentadas, fechando o documento.

Capítulo 2

Revisão Bibliográfica

Para o problema proposto, será necessário primeiramente desenvolver as equações do modelo matemático que representam o comportamento do fluido no escoamento.

Tais equações compõem os fundamentos da mecânica dos fluidos e estão presentes em diversos livros-texto que abordam o assunto. O livro (FOX, 2010), por exemplo, faz um bom trabalho desenvolvendo as formulações que serão apresentadas a seguir.

Para isso, parte-se de algumas hipóteses iniciais. São elas: **Fluido Newtoniano**, **Meio Contínuo** e **Incompressível**. Isto é, sua viscosidade dinâmica μ independe do gradiente de velocidade, dependendo apenas da pressão e temperatura local e a massa específica ρ não varia com o tempo. Para a formulação Corrente-Vorticidade, será uma exigência o escoamento ser incompressível.

Também, para um escoamento suficientemente bem comportado, será adotada a hipótese de que suas propriedades (μ e ρ) serão uniformes em todos os pontos do domínio.

Em seguida, as equações governantes serão desenvolvidas. Essas equações diferenciais originam da Equação do Transporte de Reynolds para um volume de controle definido (PONTES, 2009).

$$\frac{D}{Dt}\int_V F dV = \int_V \frac{\partial F}{\partial t} dV + \int_A F \underset{\sim}{v} \cdot \underset{\sim}{n} dA \tag{2.1}$$

Usando o Teorema de Green no segundo termo do lado esquerdo da Eq. 2.1

$$\frac{D}{Dt}\int_V F dV = \int_V \left(\frac{\partial F}{\partial t} + \nabla \cdot \left(F \underset{\sim}{v} \right) \right) dV \tag{2.2}$$

onde F é uma propriedade escalar qualquer do fluido e $\underset{\sim}{v}$ é o vetor velocidade em um ponto qualquer do escoamento. Eq. 2.2 descreve como que a variação da quantidade total de F dentro de um volume V é igual a variação causada pela variação do prórpio F em V mais a taxa de variação de F causada pelo movimento da propriedade através das bordas do volume de controle.

Assim, dessa equação, serão desenvolvidas as equações da Conservação de Massa, da Conservação da Quantidade de Movimento e, partindo destas, será desenvolvida a Formulação Corrente-Vorticidade.

2.1 Equação da Continuidade

A Equação da Continuidade origina da Eq. 2.2 quando usa-se $F = \rho$ (Yunus A. Çengel, 2017). Isto é, quando a propriedade a ser adotada na equação é a massa no escoamento.

Assim,

$$\frac{D}{Dt}\int_V \rho dV = \int_V \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \underset{\sim}{v})\right) dV \tag{2.3}$$

Porém, adotando as hipóteses iniciais e adotando que não há surgimento ou desaparecimento de massa, Eq. 2.3 pode ser simplicada como

$$\frac{D}{Dt}\int_V \rho dV \overset{0}{\nearrow} = \int_V \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \underset{\sim}{v})\right) dV \tag{2.4}$$

Portanto

$$\nabla \cdot (\rho \underset{\sim}{v}) = 0 \tag{2.5}$$

Ou ainda

$$\underset{\sim}{v}\nabla\rho + \rho\nabla \cdot \underset{\sim}{v} = 0 \tag{2.6}$$

Finalmente, adotando um fluido incompressível, tem-se ρ constante e, portanto, seu gradiente é nulo. isto é, $\nabla\rho = 0$. Assim, a equação anterior se reduz à forma final da Equação da Continuidade

$$\nabla \cdot \underset{\sim}{v} = 0 \tag{2.7}$$

2.2 Equação da Quantidade de Movimento

Seja, para um volume infinitesimal de fluido, a propriedade quantidade de movimento $d\underset{\sim}{G}$ definida $dG = \rho\underset{\sim}{v}$. Dessa forma, usando a Equação do Transporte de Reynolds, definida pela Eq. 2.1 e substituindo a propriedade genérica F pela definição da quantidade de movimento

$$\frac{D}{Dt}\int_V \rho\underset{\sim}{v}dV = \int_V \frac{\partial\rho\underset{\sim}{v}}{\partial t}dV + \int_A \rho\underset{\sim}{v}\underset{\sim}{v}\cdot\underset{\sim}{n}dA = F_S + F_B \tag{2.8}$$

Onde F_S são as forças externas que atuam na superfície do volume e F_B são as forças de corpo que atuam no volume. Dessa forma, essas forças podem ser escritas como

$$F_S = \int_A \boldsymbol{\sigma}\underset{\sim}{n}dA \tag{2.9}$$

$$F_B = \int_V \rho\underset{\sim}{g}dV \tag{2.10}$$

Onde $\boldsymbol{\sigma}$ é o tensor das tensões atuando na superfície do volume infinitesimal e $\underset{\sim}{g}$ é um campo vetorial de forças qualquer atuando no volume infinitesimal. Para este trabalho, $\underset{\sim}{g}$ será a gravidade.

Dessa forma, usando a Eq. 2.8, Eq. 2.9 e Eq. 2.10, é possível escrever

$$\int_V \frac{\partial\rho\underset{\sim}{v}}{\partial t}dV + \int_A \rho\underset{\sim}{v}\underset{\sim}{v}\cdot\underset{\sim}{n}dA = \int_A \boldsymbol{\sigma}\underset{\sim}{n}dA + \int_V \rho\underset{\sim}{g}dV \tag{2.11}$$

Aplicando o Teorema de Gauss na Eq. 2.11, encontra-se finalmente

$$\frac{\partial(\rho\underset{\sim}{v})}{\partial t} + \nabla\cdot(\rho\underset{\sim}{v}\underset{\sim}{v}) = \nabla\cdot\boldsymbol{\sigma} + \rho\underset{\sim}{g} \tag{2.12}$$

Expandindo o lado esquerdo da equação anterior, sendo a massa específica constante

$$\rho\frac{\partial\underset{\sim}{v}}{\partial t} + \underset{\sim}{v}\cancelto{0}{\frac{\partial\rho}{\partial t}} + \rho\underset{\sim}{v}\cancelto{0}{\left(\nabla\cdot\underset{\sim}{v}\right)} + \underset{\sim}{v}\cdot\left[\underset{\sim}{v}\cancelto{0}{\nabla\rho} + \rho\nabla\underset{\sim}{v}\right] \tag{2.13}$$

Assim, o lado esquerdo da equação se torna

$$\rho\frac{\partial\underset{\sim}{v}}{\partial t} + \rho\underset{\sim}{v}\cdot\nabla\underset{\sim}{v}$$

Agora, para o lado direito da Eq. 2.12, é possível expandir $\boldsymbol{\sigma}$ como um termo de tensões normais (Pressão P) e um termo de tensões cisalhantes $\boldsymbol{\tau}$ de forma que

$$\boldsymbol{\sigma} = -P\boldsymbol{I} + \boldsymbol{\tau} \tag{2.14}$$

Sendo $\boldsymbol{I}$ a matriz identidade e $\boldsymbol{\tau}$ pode ser escrito como

$$\boldsymbol{\tau} = \mu \left(\nabla \underset{\sim}{v} + (\nabla \underset{\sim}{v})^T \right) \tag{2.15}$$

De forma que

$$\nabla \cdot \boldsymbol{\sigma} = -\nabla P + \nabla \cdot \boldsymbol{\tau} \tag{2.16}$$

E

$$\nabla \cdot \boldsymbol{\tau} = \mu \nabla^2 \underset{\sim}{v} \tag{2.17}$$

Logo, a Eq. 2.12 pode finalmente ser reescrita como

$$\frac{\partial \underset{\sim}{v}}{\partial t} + \underset{\sim}{v} \cdot \nabla \underset{\sim}{v} = -\frac{1}{\rho} \nabla P + \nu \nabla^2 \underset{\sim}{v} + \underset{\sim}{g} \tag{2.18}$$

Onde $\nu = \mu / \rho$ é a viscosidade cinemática. Tal equação é conhecida como Equação de Navier-Stokes.

2.3 Função Corrente - Vorticidade

Essa formulação é um modo de expressar a equação de Navier-Stokes em termos da função corrente ψ e da vorticidade ω. Mais detalhes podem ser encontrados em (Yunus A. Çengel, 2017) e (Panton, 2013). Essa formulação é possível pois o campo de velocidades de um escoamento pode ser calculado pela função corrente, que é uma função escalar que representa a trajetória das partículas de fluido em um escoamento. A vorticidade é um vetor que representa o movimento circular das partículas de fluido em uma região do domínio. O vetor vorticidade será sempre perpendicular ao campo de velocidades no ponto analisado.

Assim, começa-se representando a relação da função corrente, que é uma função escalar constante ao longo de uma linha de escoamento e da vorticidade com o campo de velocidades. Nos casos a seguir, será adotado um domínio 2D e a hipótese de que todas as propriedades e o campo vetorial $\underset{\sim}{g}$, representando a gravidade, são constantes. Também será a adotada a hipótese de que o campo escalar de Pressão P é tal que P = P(x) e logo $\frac{\partial P}{\partial y} = 0$.

Definindo a Função Corrente, como apresentado em (Yunus A. Çengel, 2017)

$$\frac{\partial \psi}{\partial y} = v_x \ , \ \frac{\partial \psi}{\partial x} = -v_y \tag{2.19}$$

E, para a vorticidade, obtida por meio do operador rotacional aplicado ao vetor da velocidade, sendo o campo de velocidades bidimensional (Panton, 2013)

$$\omega = \nabla \times \underset{\sim}{v} = \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \tag{2.20}$$

Usando a Eq. 2.19 na Eq. 2.20, encontra-se a relação entre ψ e ω.

$$-\omega = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} \rightarrow -\omega = \nabla^2 \psi \tag{2.21}$$

Finalmente, é possível aplicar o operador rotacional na Equação de Navier-Stokes para reescrevê-la em termos da função corrente e vorticidade usando os resultados encontrados acima.

Assim, considerando novamente um domínio 2D, é possível obter a vorticidade na direção Z aplicando o operador rotacional à Eq. 2.18.

Primeiramente, a Eq. 2.18 será escrita para um domínio 2D. Ou seja, será escrita para as direções X e Y respectivamente.

$$\frac{\partial v_x}{\partial t} + v_x \frac{\partial v_x}{\partial x} + v_y \frac{\partial v_x}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \nu \left[\frac{\partial^2 v_x}{\partial x^2} + \frac{\partial^2 v_x}{\partial y^2} \right] + g_x \tag{2.22}$$

$$\frac{\partial v_y}{\partial t} + v_x \frac{\partial v_y}{\partial x} + v_y \frac{\partial v_y}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial y} + \nu \left[\frac{\partial^2 v_y}{\partial x^2} + \frac{\partial^2 v_y}{\partial y^2} \right] + g_y \tag{2.23}$$

Aplicando o Rotacional ao lado esquerdo da Eq. 2.18 e agrupando os termos semelhantes

$$\frac{\partial}{\partial t}\left[\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right] + \left[\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}\right]\left[\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right] + v_x \frac{\partial}{\partial x}\left[\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right] + v_y \frac{\partial}{\partial y}\left[\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right]$$

Porém, usando o resultado da Eq. 2.7 neste caso 2D

$$\nabla \cdot \underset{\sim}{v} = 0 \rightarrow \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} = 0 \tag{2.24}$$

O resultado anterior pode ser reescrito como

$$\frac{\partial}{\partial t}\left[\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right] + v_x \frac{\partial}{\partial x}\left[\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right] + v_y \frac{\partial}{\partial y}\left[\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right]$$

Finalmente, aplicando a Eq. 2.20 ao resultado anterior, o lado esquerdo da Equação de Navier-Stokes é reescrito em termos de ω como

$$\frac{\partial}{\partial t}\omega + v_x \frac{\partial}{\partial x}\omega + v_y \frac{\partial}{\partial y}\omega = \frac{D\omega}{Dt}$$

Em seguida, aplicando o rotacional ao lado direito da Eq. 2.18, sabendo que

$$\frac{\partial^2 P}{\partial x \partial y} = 0 \tag{2.25}$$

O lado direito da Eq. 2.18 pode ser escrito como

$$\nu \left[\frac{\partial^3 v_y}{\partial x^3} + \frac{\partial^3 v_y}{\partial x \partial y^2} - \frac{\partial^3 v_x}{\partial y \partial x^2} - \frac{\partial^3 v_x}{\partial y^3} \right]$$

Explicitando o operador rotacional presente

$$\nu \left[\frac{\partial^2}{\partial x^2} \left(\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right) + \frac{\partial^2}{\partial y^2} \left(\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right) \right]$$

Aplicando novamente a Eq. 2.20 a este novo resultado e, sabendo que o rotacional de uma constante é nulo (Panton, 2013) implicando no desaparecimento do termo da gravidade

$$\nu \left[\frac{\partial^2}{\partial x^2} \omega + \frac{\partial^2}{\partial y^2} \omega \right] = \nu \nabla^2 \omega$$

Por fim, aplicando os resultados na Eq. 2.18, tem-se

$$\frac{D\omega}{Dt} = \nu \nabla^2 \omega \tag{2.26}$$

A Eq. 2.26 apresenta a derivada material da Vorticidade no escoamento e, junto a Eq. 2.21, permite o cálculo da Função Corrente para todos os pontos do escoamento.

Por sua vez, o campo de velocidades pode ser obtido por meio da Função Corrente usando a Eq. 2.19.

Capítulo 3

Discretização do Modelo Matemático pelo Método de Elementos Finitos

3.1 Método de Elementos Finitos - Método dos Resíduos Ponderados e Método de Galerkin

Estes métodos consistem em dividir o domínio Ω e o contorno Γ do problema em partes (elementos) tal que, para elementos suficientemente pequenos, é possível considerar que as propriedades dentro destes elementos são uniformes. Também, visto que elementos diferentes adjacentes compartilham arestas, vértices e faces, será necessário considerar a interação entre elementos distintos nos cálculos (Anjos, 2021).

Tal método é amplamente utilizado na indústria atualmente graças a capacidade computacional capaz de suportar as cargas de trabalho e, principalmente, pois este método oferece uma solução suficientemente próxima da analítica para problemas onde a geometria, número de corpos, contatos, carregamentos e outros aspectos são demasiados complexos para serem resolvidos analiticamente com as ferramentas matemáticas existentes.

A adoção deste método, no entanto, requer uma formulação própria do problema, chamada de Forma Fraca do problema, sendo a formulação matemática acima apresentada a Forma Forte.

A Forma Forte de um problema é a formulação matemática por meio das Equações Diferencias que descrevem analiticamente o problema e que, se resolvidas usando as diversas ferramentas matemáticas, são consideradas a solução que perfeitamente representa o comportamento físico do sistema. Para a construção

da Forma Fraca, obtida da Forma Forte, será utilizado o Método dos Resíduos Ponderados e, posteriormente o Método de Galerkin proverá a discretização espacial.

O Método dos Resíduos Ponderados, como descrito em (ROLAND W. LEWIS, 2004), consiste em considerar que, por se tratar de uma aproximação, a aplicação de cada equação irá gerar um resíduo intrínseco a aproximação utilizada. Isto é, para todo o domínio Ω

$$\begin{aligned} \frac{D\omega^*}{Dt} &= \nu\nabla^2\omega^* + R_1 \\ \nabla^2\psi^* &= -\omega^* + R_2 \\ \underset{\sim}{v}^* &= \left(\frac{\partial\psi^*}{\partial y}, -\frac{\partial\psi^*}{\partial x}\right) + R_3 \end{aligned}$$

Onde R é um resíduo gerado e o sobrescrito $*$ indica que se trata de uma aproximação do valor da variável.

O método, então, irá buscar minimizar este resíduo no domínio Ω do problema, dando a ele seu nome de Método dos Resíduos Ponderados. Assim, é interessante explicitar R como

$$\begin{aligned} R_1 &= \frac{D\omega^*}{Dt} - \nu\nabla^2\omega^* \\ R_2 &= \nabla^2\psi^* + \omega^* \\ R_3 &= \underset{\sim}{v}^* - \left(\frac{\partial\psi^*}{\partial y}, -\frac{\partial\psi^*}{\partial x}\right) \end{aligned}$$

Assim, para minimizar o resíduo R no domínio, faz-se o uso de uma função peso w que multiplica R buscando zerar a integral da ponderação de resíduos no domínio. Isto é

$$\begin{aligned} \int_\Omega w_1 R_1 d\Omega = 0 \rightarrow \int_\Omega w_1 \left[\frac{D\omega^*}{Dt} - \nu\nabla^2\omega^*\right] d\Omega = 0 \\ \int_\Omega w_2 R_2 d\Omega = 0 \rightarrow \int_\Omega w_2 \left[\nabla^2\psi^* + \omega^*\right] d\Omega = 0 \\ \int_\Omega w_3 R_3 d\Omega = 0 \rightarrow \int_\Omega w_3 \left[\underset{\sim}{v}^* - \left(\frac{\partial\psi^*}{\partial y}, -\frac{\partial\psi^*}{\partial x}\right)\right] d\Omega = 0 \end{aligned}$$

Resultando na Forma Fraca do problema exemplificado. Isto é, a Forma Fraca é a Forma Forte multiplicada pela função peso.

Por último, é necessário caracterizar o comportamento de w no contorno do domínio. Dado que este trabalho irá usar apenas condições de contorno de Dirichlet (valores definidos no contorno) e condições de Neumann (fluxo prescrito) nulo, entende-se que o resíduo no contorno será sempre zero e, portanto, a função peso w deverá ser zero no contorno para garantir esta informação na ponderação.

Em seguida, tendo em vista a discretização do domínio estudado, será adotado o uso de funções de interpolação para definir por aproximação as variáveis estudadas ao longo de cada domínio Ω^e pertencente a um elemento específico do domínio global Ω agora discretizado. Isto é, uma variável poderá ser calculada em um ponto qualquer dentro do elemento por meio da interpolação dos valores desta variável em cada vértice do elemento. Assim, para uma variável qualquer F

$$F(\Omega, t) = \sum_{i=1}^{\infty} F_i(t) N_i(\Omega) \ , \ w(\Omega) = \sum_{j=1}^{\infty} w_j N_j(\Omega) \tag{3.1}$$

Onde $N_i(\Omega)$ é uma função de interpolação polinomial e F_i e w_j são pesos para estas funções. Também, é considerado uma variável F que depende de Ω e da evolução temporal do sistema e uma função peso w que depende apenas de Ω. No entanto, exitem alguns critérios impostos pelo para a escolha dessas funções. Tais critérios exigem que a função escolhida seja infinitamente diferenciável, satisfaça as condições de contorno de Dirichlet e que a função N seja não-nula no contorno.

Finalmente, como não é possível trabalhar com somas infinitas, uma vez determinado o número de vértices de um elemento, uma aproximação é imposta. Isto é

$$F(\Omega, t) \approx \sum_{i=1}^{n} F_i(t) N_i(\Omega) \ , \ w(\Omega) \approx \sum_{j=1}^{n} w_j N_j(\Omega) \tag{3.2}$$

Por definição, a função interpoladora deve possuir valor 1 no i-ésimo nó do elemento e 0 nos demais nós.

Finalmente, resta apenas escolher a modelagem das funções de interpolação. Aqui, faz-se o uso do Método de Galerkin. Método este que impõe que qualquer ordem de polinômio interpolador pode ser usada, desde que respeite a seguinte condição

$$N_i(\Omega) = N_j(\Omega)$$

3.2 Elaboração da Forma Fraca

Seja a forma forte do problema

$$\begin{aligned} &\frac{D\omega}{Dt} - \nu\nabla^2\omega = 0 \\ &\nabla^2\psi + \omega = 0 \\ &\frac{\partial\psi}{\partial y} = v_x \ , \ \frac{\partial\psi}{\partial x} = -v_y \end{aligned} \tag{3.3}$$

Onde a seguinte equação auxiliar é usada

$$\omega_{b.c} = \nabla \times \underset{\sim}{v} \ \text{ em } \ \underset{\sim}{v}(\Gamma_v) = \overline{\underset{\sim}{v}} \tag{3.4}$$

definido em um domínio Ω com contorno Γ tal que o contorno do domínio pode ser dividido em Γ_ψ, a porção do contorno onde o valor de ψ é conhecido e Γ_v, a porção do contorno onde $\overline{\underset{\sim}{v}}$ é conhecido.

$$\Gamma = \Gamma_\psi + \Gamma_v \rightarrow \begin{cases} x, y \exists \Gamma_\psi \rightarrow \psi(\Gamma_\psi) = \overline{\psi} \\ x, y \exists \Gamma_v \rightarrow \underset{\sim}{v}(\Gamma_v) = \overline{\underset{\sim}{v}} \end{cases} \tag{3.5}$$

Para não haver confusão com a notação entre a vorticidade e a função peso, a vorticidade será escrita como ω_z. Assim, para obter a forma fraca, faz-se o uso do Método dos Resíduos Ponderados, lembrando da condição de w no contorno

$$\begin{aligned} &\int_\Omega w\left(\frac{D\omega_z}{Dt} - \nu\nabla^2\omega_z\right)d\Omega = 0 \\ &\int_\Omega w(\nabla^2\psi + \omega_z)d\Omega = 0 \\ &\int_\Omega w\left(\frac{\partial\psi}{\partial y} - v_x\right)d\Omega = 0 \ , \ w\left(\frac{\partial\psi}{\partial x} + v_y\right)d\Omega = 0 \\ &w_\Gamma = 0 \end{aligned} \tag{3.6}$$

Começando pela Vorticidade, expandindo a derivada material

$$\int_\Omega w(\frac{\partial\omega_z}{\partial t} + v_x\frac{\partial\omega_z}{\partial x} + v_y\frac{\partial\omega_z}{\partial y} - \nu\nabla^2\omega_z)d\Omega = 0 \ ; \ w_\Gamma = 0 \tag{3.7}$$

Assim, pode-se fazer

$$\underset{I}{\int_\Omega w\dot{\omega_z}d\Omega} \ \ \underset{II}{-\nu\int_\Omega w\nabla^2\omega_z d\Omega} \ \ \underset{III}{-\int_\Omega w\left(v_x\frac{\partial\omega_z}{\partial x} + v_y\frac{\partial\omega_z}{\partial y}\right)d\Omega} = 0 \tag{3.8}$$

Sendo III reescrito como

$$III = -\int_{\Omega} w\underset{\sim}{v} \cdot Grad(\omega_z) d\Omega \tag{3.9}$$

Fazendo a derivada por partes para o termo II

$$\nu \int_{\Omega} w\nabla^2\omega_z d\Omega = \nu \oint_{\Gamma} w\nabla\omega_z \underset{\sim}{n} d\Gamma - \nu \int_{\Omega} \nabla w \nabla\omega_z d\Omega \tag{3.10}$$

no entanto, relembrando da definição apresentada na Eq. 3.6 de que a função peso w é nula no contorno

$$\nu \oint_{\Gamma} \cancel{w}^{0} (\nabla\omega_z \underset{\sim}{n}) d\Gamma = 0 \tag{3.11}$$

logo

$$II = \nu \int_{\Omega} \nabla w \nabla\omega_z d\Omega \tag{3.12}$$

Para resolver as integrais anteriores, adota-se a aproximação introduzida na Eq. 3.2 pois é necessário assumir a forma matemática dessas grandezas e, dado o uso do método de Galerkin, ja possui critérios para a escolha dessas funções.

$$\omega_z(\Omega, t) = \sum_{i}^{\infty} u_i(t) N_i(\Omega) \; ; \; w(\Omega) = \sum_{j}^{\infty} b_j N_j(\Omega) \tag{3.13}$$

Onde u_i é o termo dependente unicamente do tempo e que servirá de peso para a i-ésima função polinomial linear N_i, que depende apenas das coordenadas espaciais e que juntos vão compor a solução aproximada introduzida na Eq. 3.2 para a grandeza estudada. De forma semelhante, b_j é o j-ésimo termo, constante por não depender do tempo, e que multiplica a a j-ésima função polinomial linear N_j para encontrar a solução aproximada para a função peso w.

Agora, resolvendo para I, II e III:

$$I\text{:} \quad \int_{\Omega} w\dot{\omega_z} d\Omega = \int_{\Omega} \sum_{i}^{\infty} \left[\sum_{j}^{\infty} \left(\dot{u_i} N_i\right) \left(b_j N_j\right) \right] d\Omega \tag{3.14}$$

ou, na forma matricial

$$I\text{:} \quad \int_{\Omega} \begin{bmatrix} \dots & b_j^T & \dots \end{bmatrix} \begin{bmatrix} N_j N_i \end{bmatrix} \begin{bmatrix} \vdots \\ \dot{u_i} \\ \vdots \end{bmatrix} d\Omega \tag{3.15}$$

em seguida

$$II\text{: } \nu \int_\Omega \nabla w \nabla \omega_z d\Omega \; ; \; \nabla \omega_z = \frac{\partial u_k N_k}{\partial x_i} = u_k \frac{\partial N_k}{\partial x_i} = \begin{bmatrix} \frac{\partial N_1}{\partial x_1} & \dots & \frac{\partial N_n}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial N_1}{\partial x_n} & \dots & \frac{\partial N_n}{\partial x_n} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = B \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \tag{3.16}$$

de forma similar

$$\nabla w = B \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} b_1 & \dots & b_n \end{bmatrix} B^T \tag{3.17}$$

Então

$$II = \nu \int_\Omega \begin{bmatrix} \dots & b_j^T & \dots \end{bmatrix} B^T B \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} d\Omega \tag{3.18}$$

Finalmente, para III

$$III\text{: } \; - \int_\Omega w \underset{\sim}{v} \cdot Grad(\omega_z) d\Omega = - \int_\Omega \underset{\sim}{v} \sum_{i=1}^{\infty} \left[\sum_{j=1}^{\infty} (u_i \nabla N_i)(b_j N_j) \right] d\Omega \tag{3.19}$$

A Eq. 3.19 pode ser reescrita na forma matricial como

$$\int_\Omega \underset{\sim}{v} \begin{bmatrix} \dots & b_j^T & \dots \end{bmatrix} \begin{bmatrix} N_j \nabla N_i \end{bmatrix} \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} d\Omega$$

Juntando os resultados anteriores e simplificando

$$\int_\Omega \cancel{\begin{bmatrix} \dots & b_j^T & \dots \end{bmatrix}} \begin{bmatrix} N_j N_i \end{bmatrix} \begin{bmatrix} \vdots \\ \dot{u}_i \\ \vdots \end{bmatrix} d\Omega + \nu \int_\Omega \cancel{\begin{bmatrix} \dots & b_j^T & \dots \end{bmatrix}} B^T B \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} - \int_\Omega \underset{\sim}{v} \cancel{\begin{bmatrix} \dots & b_j^T & \dots \end{bmatrix}} \begin{bmatrix} N_j \nabla N_i \end{bmatrix} \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} d\Omega = 0$$

Definindo ainda

$$\int_\Omega \begin{bmatrix} N_j N_i \end{bmatrix} d\Omega = M_{ij} \; , \; \int_\Omega B^T B d\Omega = K \; , \; \int_\Omega \begin{bmatrix} N_j \nabla N_i \end{bmatrix} d\Omega = G \tag{3.20}$$

logo, finalmente

$$M \begin{bmatrix} \vdots \\ \dot{u}_i \\ \vdots \end{bmatrix} + \nu K \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} - \underset{\sim}{v} G \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} = 0 \tag{3.21}$$

é a Forma Fraca do problema.

Também, é necessário escrever a Forma Fraca das funções auxiliares.

Partindo das Eq. 3.4,

$$\int_\Omega w \left(v_x - \frac{\partial \psi}{\partial y} \right) d\Omega = \int_\Omega \sum_{i=1}^{\infty} \left[\sum_{j=1}^{\infty} (v_{x_i} N_i - \psi_i \frac{\partial N_i}{\partial y})(b_j N_j) \right] = 0 \tag{3.22}$$

Usando as relações na Eq. 3.20 e definindo

$$\int_\Omega \left[N_j \tfrac{\partial N_i}{\partial y} \right] d\Omega = G_y \tag{3.23}$$

Então

$$M \begin{bmatrix} \vdots \\ v_{x_i} \\ \vdots \end{bmatrix} = G_y \begin{bmatrix} \vdots \\ \psi_i \\ \vdots \end{bmatrix} \tag{3.24}$$

Também, definindo

$$\int_\Omega \left[N_j \tfrac{\partial N_i}{\partial x} \right] d\Omega = G_x \tag{3.25}$$

E seguindo um processo semelhante

$$M \begin{bmatrix} \vdots \\ v_{y_i} \\ \vdots \end{bmatrix} = -G_x \begin{bmatrix} \vdots \\ \psi_i \\ \vdots \end{bmatrix} \tag{3.26}$$

Em seguida, partindo da Eq. 2.20

$$\omega_z = \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \rightarrow \int_\Omega w \left(\omega_z - \frac{\partial v_y}{\partial x} + \frac{\partial v_x}{\partial y} \right) d\Omega = 0 \rightarrow M \begin{bmatrix} \vdots \\ \omega_{z_i} \\ \vdots \end{bmatrix} = G_x \begin{bmatrix} \vdots \\ v_{y_i} \\ \vdots \end{bmatrix} - G_y \begin{bmatrix} \vdots \\ v_{x_i} \\ \vdots \end{bmatrix} \tag{3.27}$$

Esta relação permite calcular ω a partir das velocidades e será útil para o cálculo

das condições de contorno de ω uma vez que as velocidades serão pré-determinadas no contorno do domínio.

Finalmente, para a relação

$$\omega_z + \nabla^2\psi \rightarrow \int_\Omega w(\omega_z + \nabla^2\psi)d\Omega = 0 \tag{3.28}$$

Aproveitando o processo realizado nas Eq. 3.10, Eq. 3.11 e Eq. 3.12 na Eq. 3.28

$$\int_\Omega (w\omega_z + \nabla w \nabla\psi)d\Omega = 0 \tag{3.29}$$

Usando o desenvolvimento das Eq. 3.16, Eq. 3.17 e Eq. 3.18 e mais a definição na Eq. 3.20, a Eq. 3.29 é escrita como

$$M \begin{bmatrix} \vdots \\ \omega_{z_i} \\ \vdots \end{bmatrix} - K \begin{bmatrix} \vdots \\ \psi_i \\ \vdots \end{bmatrix} = 0 \tag{3.30}$$

Para tornar possível o cálculo, $\underset{\sim}{v}G$ será reformulado de forma que

$$G = \int_\Omega \left[N_j \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} \right] d\Omega \ , \ \underset{\sim}{v} = [v_x, v_y] \tag{3.31}$$

Assim, usando a Eq. 3.31 e adotando

$$v_x = \begin{bmatrix} \vdots \\ v_{x_i} \\ \vdots \end{bmatrix} \boldsymbol{I} \ , \ v_y = \begin{bmatrix} \vdots \\ v_{y_i} \\ \vdots \end{bmatrix} \boldsymbol{I} \tag{3.32}$$

onde $\boldsymbol{I}$ é a matriz identidade. Dessa forma, $\underset{\sim}{v}G$ se torna

$$\underset{\sim}{v}G = \begin{bmatrix} \vdots \\ v_{x_i} \\ \vdots \end{bmatrix} \boldsymbol{I} \int_\Omega \left[N_j \frac{\partial N_i}{\partial x} \right] d\Omega + \begin{bmatrix} \vdots \\ v_{y_i} \\ \vdots \end{bmatrix} \boldsymbol{I} \int_\Omega \left[N_j \frac{\partial N_i}{\partial y} \right] d\Omega \tag{3.33}$$

E usando as definições nas Eq. 3.23 e Eq. 3.25

$$\underset{\sim}{v}G = \begin{bmatrix} \vdots \\ v_{x_i} \\ \vdots \end{bmatrix} \boldsymbol{I}G_x + \begin{bmatrix} \vdots \\ v_{y_i} \\ \vdots \end{bmatrix} \boldsymbol{I}G_y \tag{3.34}$$

3.3 Discretização das Equações em um Elemento Triangular

Visto que a malha adotada será composta de elementos triangulares bidimensionais, é interessante calcular as matrizes M, K, G_x e G_y associadas a esses elementos. Tal desenvolvimento é elaborado em (JACOB FISH, 2007) e na apostila (Anjos, 2022) porém uma recapitulação será apresentada abaixo.

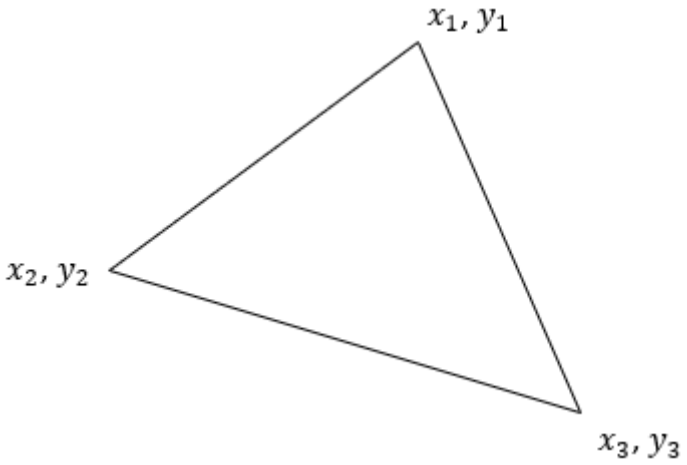


Figura 3.1: Elemento triangular

Definindo

$$\Omega = [x, y] \rightarrow \int_{\Omega} f d\Omega = \int_{y}\int_{x} f dx dy$$

Em seguida, em um elemento triangular, adota-se

$$\omega_z^e = u_1 N_1 + u_2 N_2 + u_3 N_3 \; ; \; w^e = b_1 N_1 + b_2 N_2 + b_3 N_3 \tag{3.35}$$

onde, lembrando das regras impostas a função N

$$\begin{aligned} N_1(x_1, y_1) = 1 \text{ e } N_2(x_1, y_1) = N_3(x_1, y_1) = 0 \\ N_2(x_2, y_2) = 1 \text{ e } N_1(x_2, y_2) = N_3(x_2, y_2) = 0 \\ N_3(x_3, y_3) = 1 \text{ e } N_1(x_3, y_3) = N_2(x_3, y_3) = 0 \end{aligned}$$

e, adotando uma formulação linear (LEWIS, 2004), que oferece performance computacionais mais rápida em detrimento de aproximações normalmente, mas nem sempre, mais grosseiras (Anjos, 2021)

$$N_i = \frac{1}{2A}(a_i + b_i x + c_i y) \tag{3.36}$$

onde A é a área do elemento triangular.

$$A = \frac{1}{2} \left| x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \right| \tag{3.37}$$

Assim, resolvendo o sistema anterior

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

Dessa forma, integrando no domínio do elemento as definições obtidas em Eq. 3.20, Eq. 3.23 e Eq. 3.25

$$M^e = \frac{A}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \; ; \; B^e = \begin{bmatrix} b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \rightarrow K^e = \frac{1}{4A} \cdot B^{eT} B^e \tag{3.38}$$

sendo

$$K^e = \frac{1}{4A}\begin{bmatrix} b_1^2 + c_1^2 & b_1b_2 + c_1c_2 & b_1b_3 + c_1c_3 \\ b_1b_2 + c_1c_2 & b_2^2 + c_2^2 & b_2b_3 + c_2c_3 \\ b_1b_3 + c_1c_3 & b_2b_3 + c_2c_3 & b_3^2 + c_3^2 \end{bmatrix} \tag{3.39}$$

E também

$$G_x = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \ , \ G_y = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_1 & c_2 & c_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \tag{3.40}$$

3.4 Montagem do Sistema Global

Ao montar o sistema, sabe-se que elementos terão uma justaposição de vértices e que, para esses nós, a solução será uma soma dos resultados individuais de cada elemento.

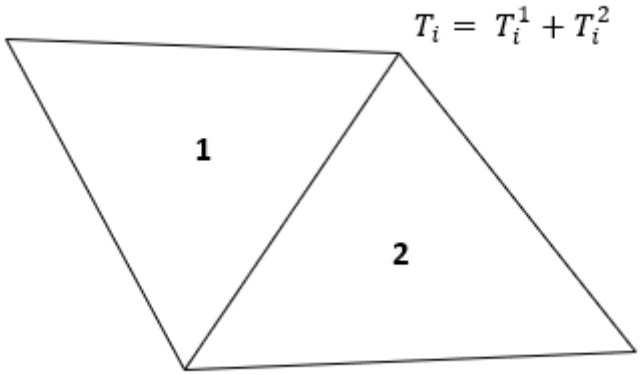


Figura 3.2: Justaposição de elementos adjacentes

Dessa forma, o sistema poderá ser representado por grandes matrizes globais M, K, G_x e G_y as quais representam a interação de um nó global n_i com um nó global n_j como a superposição das interações entre estes pontos i e j (vistos como os nós i e j do elemento) contidos em certos elementos. Isto é

$$M_{n_i n_j} = \sum_e M_{ij}^e$$
$$K_{n_i n_j} = \sum_e K_{ij}^e$$
$$G_{x n_i n_j} = \sum_e G_{x ij}^e$$
$$G_{y_{n_i n_j}} = \sum_e G_{y ij}^e$$

sendo e os elementos que contém os nós n_i e n_j.

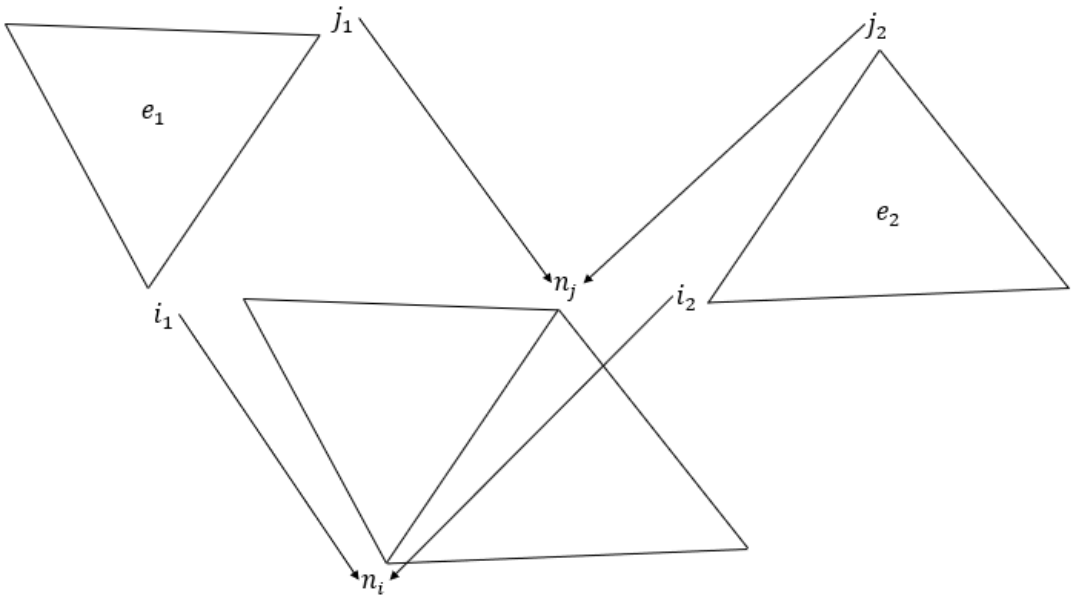


Figura 3.3: Interação entre os nós n_i e n_j como a superposição dessas interações representada em elementos que contém esses nós

3.5 Solução do Sistema

Seja o sistema completo

$$M \begin{bmatrix} \vdots \\ \dot{\omega}_i \\ \vdots \end{bmatrix} + \nu K \begin{bmatrix} \vdots \\ \omega_i \\ \vdots \end{bmatrix} - \underline{\nu} G \begin{bmatrix} \vdots \\ \omega_i \\ \vdots \end{bmatrix} = 0$$

Ou ainda

$$\begin{bmatrix} \vdots \\ \dot{\omega}_i \\ \vdots \end{bmatrix} + M^{-1}(\nu K - \underline{\nu} G) \begin{bmatrix} \vdots \\ \omega_i \\ \vdots \end{bmatrix} = 0 \tag{3.41}$$

Discretizando no tempo de forma que, pelo método de diferenças finitas explícito

$$\begin{bmatrix} \vdots \\ \dot{\omega}_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{\omega_i^{n+1} - \omega_i^n}{\Delta t} \\ \vdots \end{bmatrix} \text{ e } \begin{bmatrix} \vdots \\ \omega_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \omega_i^n \\ \vdots \end{bmatrix} \tag{3.42}$$

pelo método implícito

$$\begin{bmatrix} \vdots \\ \dot{\omega}_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{\omega_i^{n+1}-\omega_i^n}{\Delta t} \\ \vdots \end{bmatrix} \text{ e } \begin{bmatrix} \vdots \\ \omega_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \omega_i^{n+1} \\ \vdots \end{bmatrix} \tag{3.43}$$

ou por Crank-Nicholson, um método que usa diferenças centradas, com aproximação de segunda ordem no tempo e no espaço e que é incondicionalmente estável

$$\begin{bmatrix} \vdots \\ \dot{\omega}_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{\omega_i^{n+1}-\omega_i^n}{\Delta t} \\ \vdots \end{bmatrix} \text{ e } \begin{bmatrix} \vdots \\ \omega_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{\omega_i^{n+1}+\omega_i^n}{2} \\ \vdots \end{bmatrix} \tag{3.44}$$

ou seja

$$\left[\frac{M}{\Delta t} + \nu K + \theta \underset{\sim}{v} G\right] \begin{bmatrix} \vdots \\ \omega_i^{n+1} \\ \vdots \end{bmatrix} = \left[\frac{M}{\Delta t} - (1-\theta)\underset{\sim}{v} G\right] \begin{bmatrix} \vdots \\ \omega_i^{n} \\ \vdots \end{bmatrix} \tag{3.45}$$

Onde θ é uma variável que permitirá escolher a discretização temporal desejada. Ela será 0 para o método explícito, 1 para o método implícito ou $1/2$ para o método de Crank-Nicholson.

Em seguida, definindo

$$\left[\frac{M}{\Delta t} + \nu K + \theta \underset{\sim}{v} G\right] = C \text{ e } \left[\frac{M}{\Delta t} - (1-\theta)\underset{\sim}{v} G\right] \begin{bmatrix} \vdots \\ \omega_i^{n} \\ \vdots \end{bmatrix} = b \tag{3.46}$$

$$C \begin{bmatrix} \vdots \\ \omega_i^{n+1} \\ \vdots \end{bmatrix} = b$$

Para ser possível levar em consideração as condições de contorno

$$\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1m} \\ c_{21} & c_{22} & \dots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{i1} & c_{i2} & \dots & c_{im} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mm} \end{bmatrix} \begin{bmatrix} \omega_1^{n+1} \\ \omega_2^{n+1} \\ \vdots \\ \omega_i^{n+1} \\ \vdots \\ \omega_m^{n+1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_m \end{bmatrix}$$

sendo, no sistema acima

$b_i = \overline{\omega_i}$

$$c_{ij} = \begin{cases} 1; j = i \\ 0; j \neq i \end{cases}$$

Na linha i em que há condição de contorno do i-ésimo ponto.

Finalmente

$$\begin{bmatrix} \omega_1^{n+1} \\ \omega_2^{n+1} \\ \vdots \\ \omega_i^{n+1} \\ \vdots \\ \omega_m^{n+1} \end{bmatrix} = C^{-1} \begin{bmatrix} \omega_1^{n+1} \\ \omega_2^{n+1} \\ \vdots \\ b_i \\ \vdots \\ \omega_m^{n+1} \end{bmatrix}$$

Pode ser usado para calcular a iteração seguinte da Vorticidade em todos os pontos usando como base os valores obtidos na última iteração e as condições de contorno.

Capítulo 4

Metodologia Numérica

Este capítulo irá mostrar o passo a passo utilizado para a resolução numérica das equações de escoamento desenvolvidas até agora para um dado domínio.

4.1 Geometria e Malha

O trabalho de definição de geometria e criação de malha foi feito completamente com o software *Gmsh*. Este software foi escolhido dado sua facilidade de uso e seu código aberto e a versão do programa utilizada foi a 3.0.6, gratuita para uso. Lembrando que todo o descritivo é para uma malha bidimensional.

O primeiro passo para a criação da malha é definir os contornos da geometria. Isso pode ser feito no módulo *Geometry* por meio de pontos com coordenadas passada pelo usuário e, então, ligados por meio de linhas, curvas e arcos definidos pelo usuário. Uma vez que o contorno esteja fechado, cria-se com um comando a superfície limitada por este contorno.

Em seguida, é necessário criar os grupos físicos (*Physical groups*), grupos estes que nomeiam as diferentes arestas e superfícies da geometria. Esta função permite posteriormente que o código identifique os pontos que compõem certas arestas como pontos do contorno e quais condições de contorno são aplicáveis a eles.

Uma vez finalizada a geometria, passa-se à criação da malha. O módulo *Mesh* permite criar uma malha bidimensional a partir da geometria definida de acordo com uma série de parâmetros que podem ser definidos. São estes o tipo de elemento, tamanho da malha, zonas de refino, particionamento do domínio e a escolha do algoritmo de construção da malha principalmente.

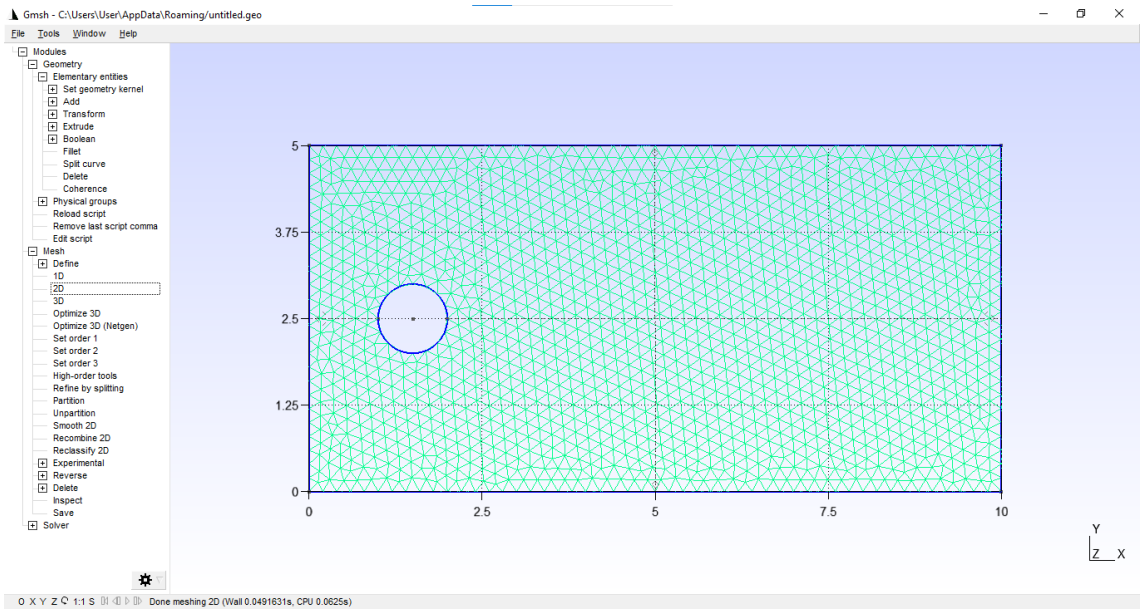


Figura 4.1: Criação de malha usando o *Gmsh*.

As malhas neste trabalho são todas compostas de elementos triangulares de primeira ordem com o algoritmo Frontal-Delaunay.

Uma vez gerada, a malha pode ser exportada como um arquivo .msh contendo as definições dos pontos, elementos e contornos no formato ASCII.

4.2 Código Numérico

O código que resolve estes problemas foi escrito em Python 3 dado a facilidade e o amplo número de bibliotecas disponíveis. Python possui diversas aplicações e guias e documentações são facilmente encontradas na internet.

O código funciona de maneira simples, seguindo uma sequência de etapas

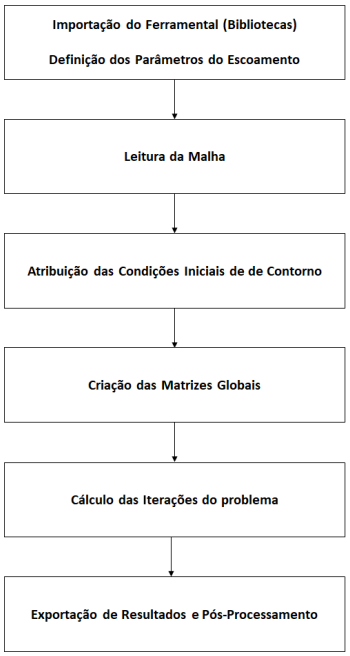


Figura 4.2: Fluxograma da execução do código.

4.2.1 Importação de Bibliotecas e Definição de Parâmetros

Como mencionado, o código faz proveito de todo um ferramental de álgebra linear ja criado e implementado via a biblioteca *Numpy* e também conta com ferramentas próprias para a leitura da malha criada pelo *Gmsh*, a biblioteca *Meshio* na versão 5.0.2.

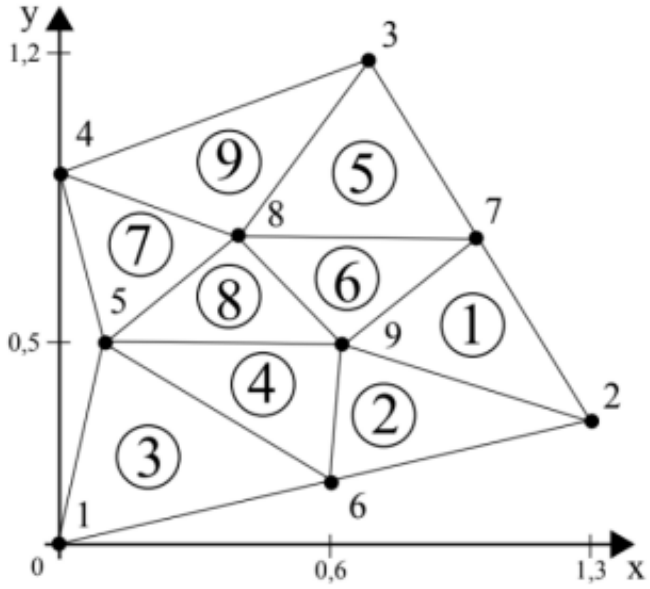
Para os parâmetros, é necessário definir o passo no tempo Δt, ν e o tempo final da simulação. Com isso o número de iterações e o número de Reynolds já poderão ser calculados.

4.2.2 Leitura da Malha

A leitura da malha por meio da biblioteca *Meshio* é feita em etapas.

Primeiramente a malha é lida em sua totalidade e guardada dentro de uma estrutura própria da biblioteca. Dessa estrutura, é possível obter as coordenadas X e Y de todos os pontos que a compõem.

Também, a matriz de conectividade (IEN) é obtida desta estrutura. Esta matriz possui a relação dos pontos (vértices) que compõem cada elemento triangular. Esta matriz é especialmente importante para a construção das matrizes globais.



IEN	v_1	v_2	v_3
1	9	2	7
2	6	2	9
3	1	6	5
4	5	6	9
5	3	8	7
6	7	8	9
7	4	5	8
8	5	9	8
9	3	4	8

Figura 4.3: Matriz IEN para um exemplo de malha (Anjos, 2021)

4.2.3 Atribuição das Condições Inicias e de Contorno

Nesta etapa, um primeiro valor (Condição Inicial) é atribuído para cada grandeza para todos os pontos da malha. Isto é, define-se uma configuração de partida para o sistema.

Em seguida, atribui-se valores de condição de contorno aos elementos já identificados como pertencentes ao contorno. As condições que serão aplicadas dependem do tipo de condição aplicada.

Para os valores de V_x, V_y e ψ, o tipo de condição aplicada é *Dirichlet*. Isto é, para os pontos no contorno, estas variáveis recebem um valor definido pré-estabelecido. Assim, a cada iteração, o valor dessas variáveis para esses pontos não se altera.

Traduzindo isto para o cálculo matricial, a aplicação destas condições pode ser entendida por meio da exemplificação da resolução do sistema genérico $A\underset{\sim}{u} = \underset{\sim}{b}$

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1i} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2i} & \dots & a_{2m} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{i1} & a_{i2} & \dots & a_{ii} & \dots & a_{im} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mi} & \dots & a_{mm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_i \\ \vdots \\ u_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_m \end{bmatrix}$$

Agora, seja um ponto p_i pertencente ao contorno com uma condição de *Dirichlet*. Nesse caso, o valor da variável neste ponto é conhecida. Isto é $u_i = \overline{b_i}$, onde $\overline{b_i}$ é um valor pré-determinado.

Assim, para garantir que o valor de u_i seja sempre este, é possível alterar a i-ésima linha de A e a i-ésima casa de $\underset{\sim}{b}$ de forma a garantir isso. Assim, faz-se

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1i} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2i} & \dots & a_{2m} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \dots & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mi} & \dots & a_{mm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_i \\ \vdots \\ u_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \overline{b_i} \\ \vdots \\ b_m \end{bmatrix}$$

Para a condição de ω, é usada uma condição do tipo *Neumman Homogêneo*. Nesta condição é adotado que o fluxo em dado ponto é conhecido e pré-determinado. Porém, neste caso, o fluxo prescrito é nulo e isso faz com que a integral no contorno seja nula e, portanto, a condição no contorno para ω, cujo cálculo por elementos finitos é descrito pela Eq. 3.45, se resume à adoção do procedimento acima descrito.

Ressaltando apenas que, por depender das velocidades calculadas a cada iteração, o valor de ω no contorno deverá ser constantemente recalculado seguindo a Eq. 3.27.

4.2.4 Montagem das Matrizes Globais

As matrizes globais levam em conta a contribuição, para um dado ponto, de cada elemento triangular que compartilha um vértice (ponto mencionado) com outros elementos. Isto é, um dado elemento triangular possuirá três vértices na forma v_i, sendo $i = 1, 2, 3$. No entanto, cada vértice v_i representa um ponto p_j da malha e, dessa forma, as contribuições dos diferentes elementos que possuem p_j como vértice serão contabilizadas.

Para isso, é preciso primeiramente criar as matrizes elementares para cada elemento, visto que cada elemento terá um conjunto próprio de matrizes elementares.

Assim, seja para um elemento, uma matriz exemplo A^e, 3 x 3, onde a casa ij representa a interação entre os vértices i e j. No entanto, no sistema completo, v_i se traduz em p_m e v_j em p_n.

Logo, a matriz global A pode ser escrita como

$$A_{mn} = \sum A^e_{p_m p_n} \tag{4.1}$$

4.2.5 Loop Temporal

Esta é a porção do método em que se gasta realmente o tempo computacional. Aqui, o computador irá executar o algoritmo repetidas vezes para calcular a evolução temporal das variáveis submetidas às condições de contorno e iniciais do problema.

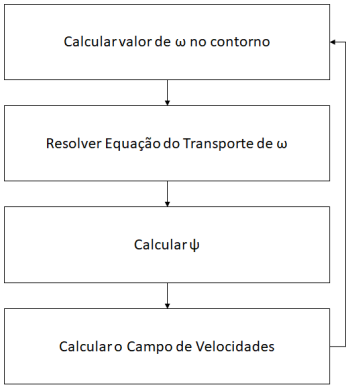


Figura 4.4: Loop temporal para o cálculo da evolução temporal do escoamento

Primeiramente, é necessário calcular as condições de contorno para ω usando a Eq. 3.27 a partir do campo de velocidades obtido na iteração anterior

$$M\omega_{cc}^{n+1} = G_x v_y^n - G_y v_x^n \tag{4.2}$$

Em seguida, após calcular $\underset{\sim}{v}G^n$ pela Eq. 3.34 com o campo de velocidades da última iteração, resolve-se a equação do transporte da vorticidade definida na Eq. 3.45 usando as condições de contorno calculadas

$$\left[\frac{M}{\Delta t} + \nu K + \theta \underset{\sim}{v}G^n\right]\omega^{n+1} = \left[\frac{M}{\Delta t} - (1-\theta)\underset{\sim}{v}G^n\right]\omega^n + c.c. \tag{4.3}$$

Calcula-se então ψ^{n+1} por meio da Eq. 3.30

$$M\omega^{n+1} = K\psi^{n+1} + c.c. \tag{4.4}$$

E por fim, calcula-se os valores de v_x^{n+1} e v_y^{n+1} usando as Eq. 3.24 e Eq. 3.26

$$Mv_x^{n+1} = G_y\psi^{n+1} + c.c.\ ,\ Mv_y^{n+1} = -G_x\psi^{n+1} + c.c. \tag{4.5}$$

4.2.6 Exportação e Visualização de Resultados

Uma vez terminada a execução do código, os resultados calculados para todos os pontos da malha em cada iteração são gravados em arquivos de texto que podem ser utilizados para um pós processamento para o cálculo dos coeficientes de arrasto e sustentação ou então convertidos em arquivos a serem visualizados no software *Paraview*.

Paraview é um software de código aberto gratuito que permite a visualização de resultados de simulações de elementos finitos. Com ele, é possível ver os resultados em cada iteração para todas as variáveis por meio de representações gráficas destas.

Também, este programa oferece uma gama de ferramentas de pós processamento dos resultados, permitindo diversas análises que auxiliam no entendimento dos resultados e na obtenção de outros dados fundamentais ao estudo realizado.

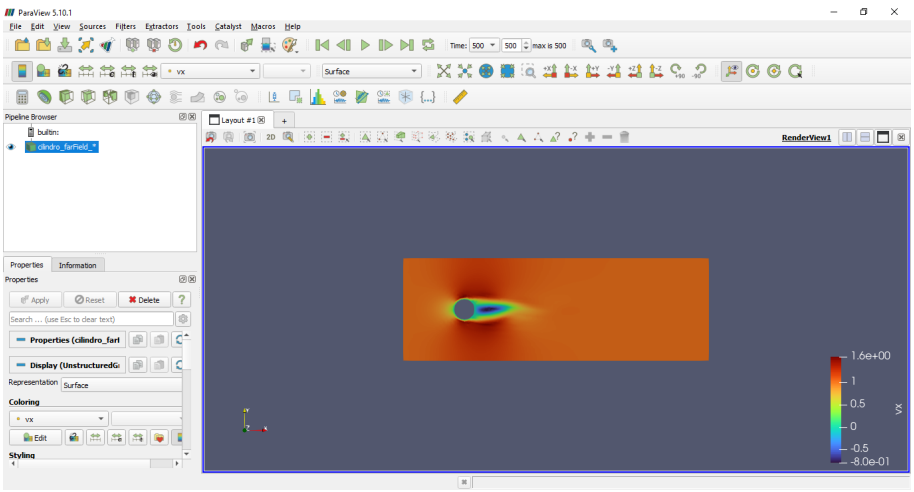


Figura 4.5: Exemplo de visualização de resultados com Paraview

Capítulo 5

Validação Numérica

Em ordem de comprovar o funcionamento do método e a qualidade dos resultados, é necessário aplicá-lo a casos simples e cuja solução analítica é conhecida para que haja uma comparação entre o resultado numérico e a solução analítica ou estudos numéricos validados.

5.1 Caso Escoamento de Poiseuille

O primeiro caso simples para a realização da comparação do resultado numérico será um escoamento entre placas planas tal que o domínio e as condições de contorno são

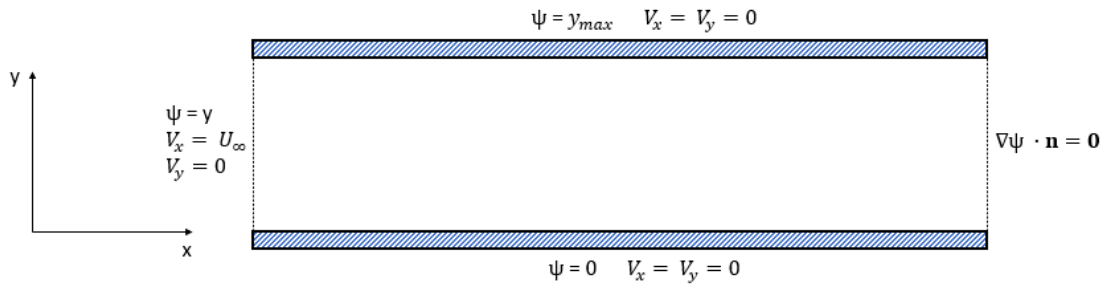


Figura 5.1: Descrição do Escoamento entre Placas Planas

Sendo que, na saída do escoamento, dado que o vetor normal $\underset{\sim}{n}$ ao contorno é o vetor unitário $\underset{\sim}{e_x}$, retomando a relação mostrada na Eq. 2.19, é possível trocar a expressão da condição de contorno por

$$\nabla\psi\underset{\sim}{e_x} = 0 \rightarrow \frac{\partial\psi}{\partial x} = -v_y = 0 \rightarrow v_y = 0 \tag{5.1}$$

5.1.1 Solução Analítica

Para o início da comparação, é preciso desenvolver as soluções analíticas para as variáveis estudadas. Para isso, será considerado uma região do domínio suficientemente longe da entrada do canal de forma que

$$v_y = 0 \ , \ \frac{\partial v_x}{\partial x} = 0 \ , \ \frac{\partial P}{\partial x} = C \ , \ \frac{\partial P}{\partial y} = 0 \tag{5.2}$$

Considerando regime permanente e nenhum campo de forças atuando no escoamento, as Eq. 2.22 e Eq. 2.23 podem ser resolvidas usando as condições de contorno acima apresentadas de forma que

$$v_y = 0 \tag{5.3}$$

$$v_x = \frac{4U_\infty}{y_{\max}^2} y[y_{\max} - y] \tag{5.4}$$

Resolvendo a Eq. 2.19 com o resultado anterior

$$\psi = \frac{4U_\infty}{y_{\max}^2} \left[y_{\max} \frac{y^2}{2} - \frac{y^3}{3} \right] \tag{5.5}$$

Finalmente, resolvendo a Eq. 2.20

$$\omega_z = -\frac{4U_\infty}{y_{\max}^2} [y_{\max} - 2y] \tag{5.6}$$

5.1.2 Comparação com o Resultado Numérico

Para uma malha de tamanho 0.06, Δt de 0.1, com 3588 elementos, adotando $U_\infty = 1.5$ e $y_{\max} = 1$, seguem os resultados obtidos

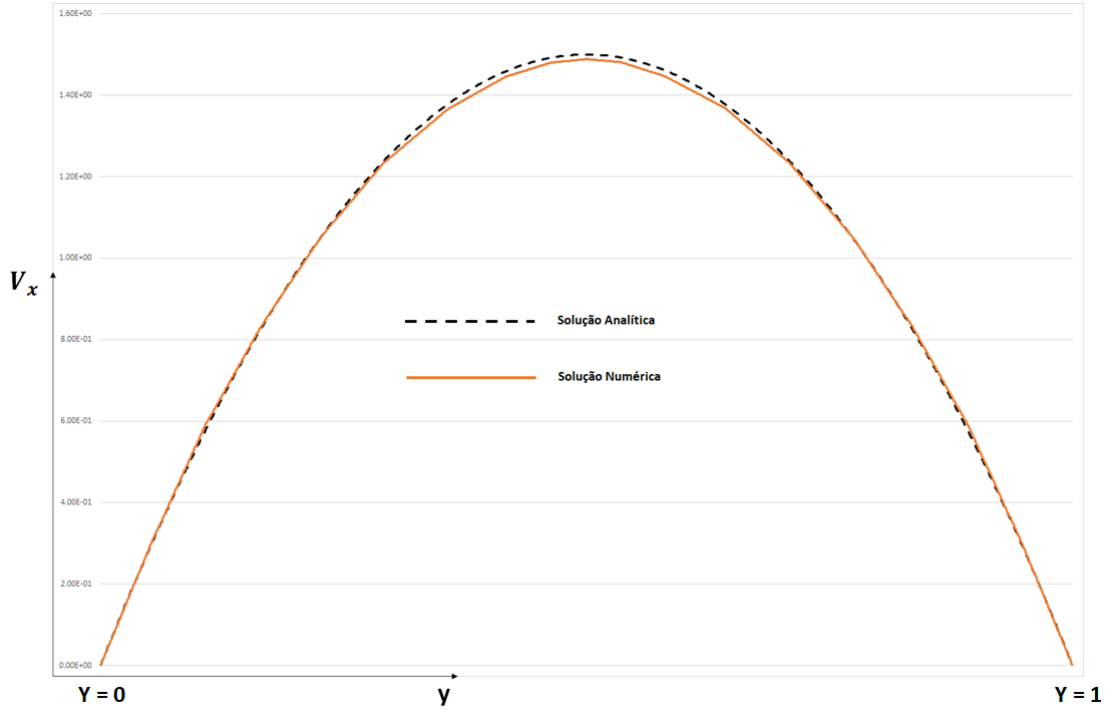


Figura 5.2: Comparação da velocidade horizontal

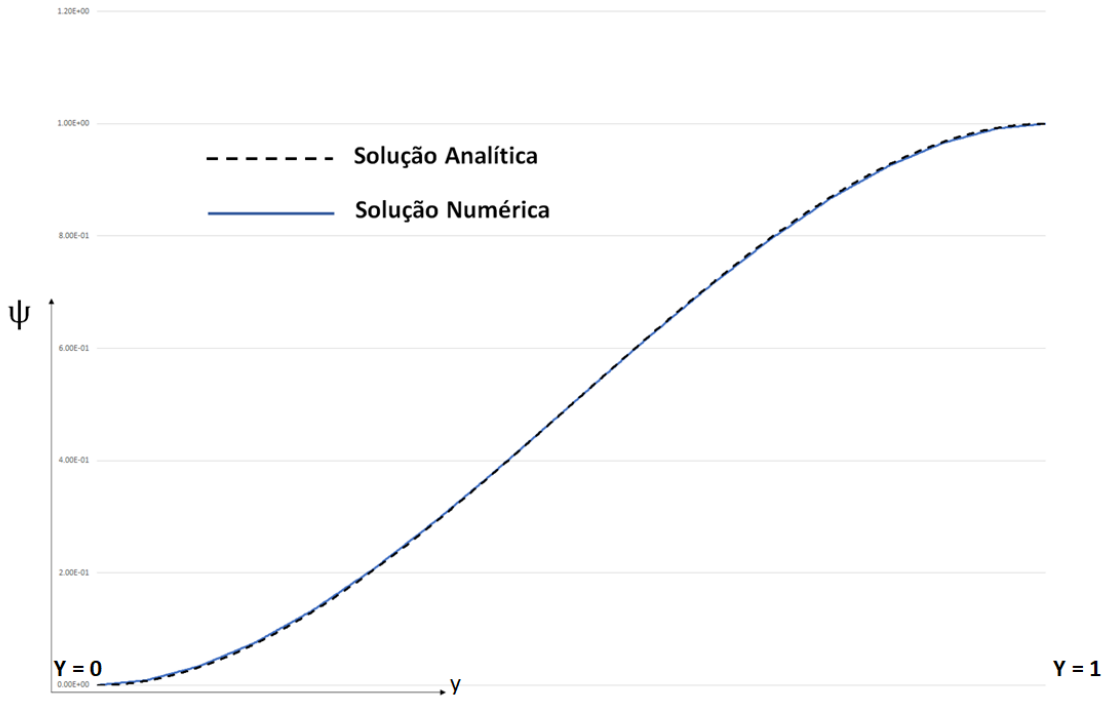


Figura 5.3: Comparação da função corrente

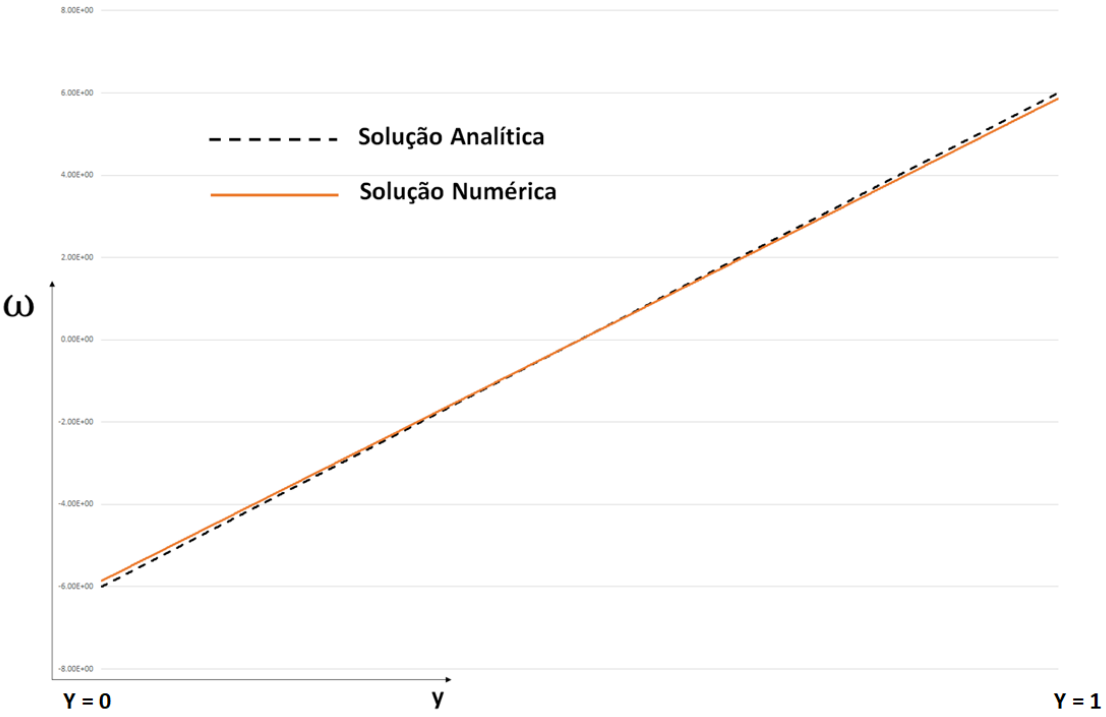


Figura 5.4: Comparação da vorticidade

E fazendo uma análise da convergência da malha, adotando $u(y)$ como o valor da velocidade horizontal para um dado valor de y e o erro relativo para cada tamanho de malha como

$$e = \max\left(\frac{|v_x(y) - u(y)|}{u(y)}\right) \tag{5.7}$$

Tabela 5.1: Erro relativo para diferentes malhas

Tamanho de Malha	Erro Relativo
0.15	0.260
0.12	0.199
0.10	0.050
0.08	0.036
0.06	0.038

O erro levemente maior para a malha mais refinada pode ser explicado por erros numéricos decorrentes da malha muito refinada.

5.2 Caso Lid - Driven

Trata-se de um outro caso cuja solução ja é conhecida. Para esta validação, foi utilizado como referência (U. GHIA, 1982). Neste cenário, adota-se o seguinte domínio e condições de contorno

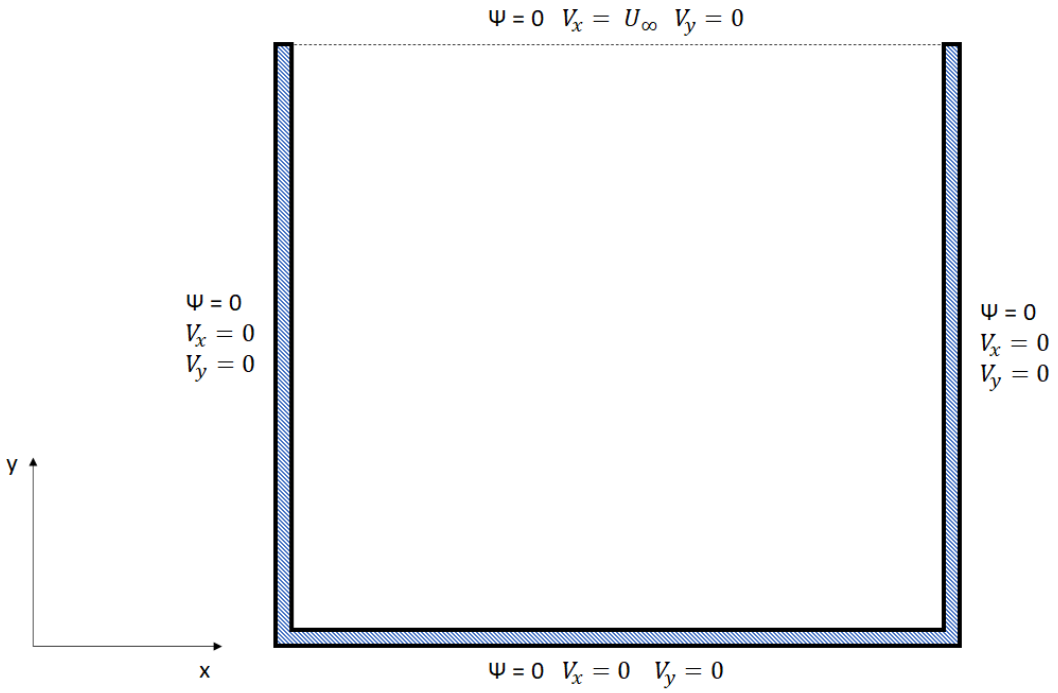


Figura 5.5: Descrição do caso Lid-Driven

Dado que o material de referência (U. GHIA, 1982) contém uma série de soluções numéricas para diferentes valores de Re, será adotado para esta validação os casos de $Re = 100$, $Re = 400$ e $Re = 1000$. Para cada cálculo computacional, o perfil de velocidades v_x sobre uma reta vertical passando pelo centro geométrico do domínio será comparado com o equivalente encontrado na referência.

Para uma malha de tamanho 0.05, Δt de 0.1, com 3360 elementos, adotando $U_\infty = 1$ e $x_{\max} = y_{\max} = 1$, seguem os resultados obtidos em regime permanente (tanto o trabalho atual quanto (U. GHIA, 1982) atingiram regime permanente)

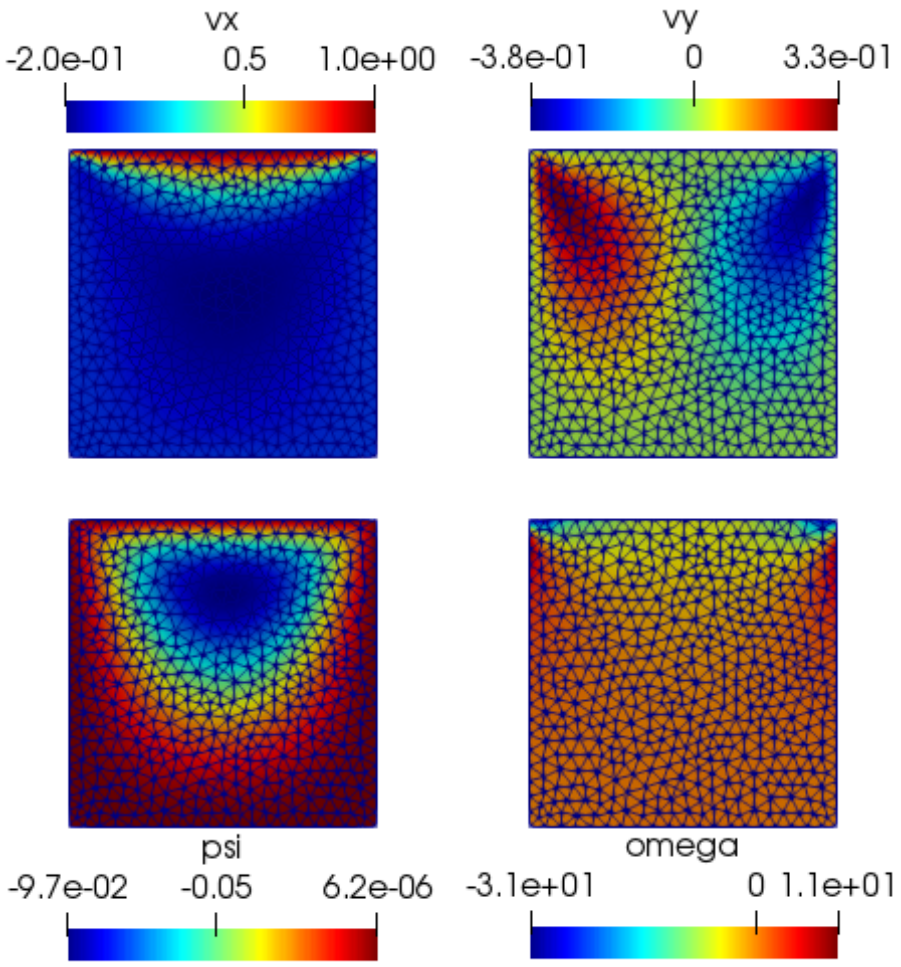


Figura 5.6: Exemplo de Cálculo Computacional para $Re = 10$

Assim, analisando os gráficos de v_x para os valores de Re mencionados acima

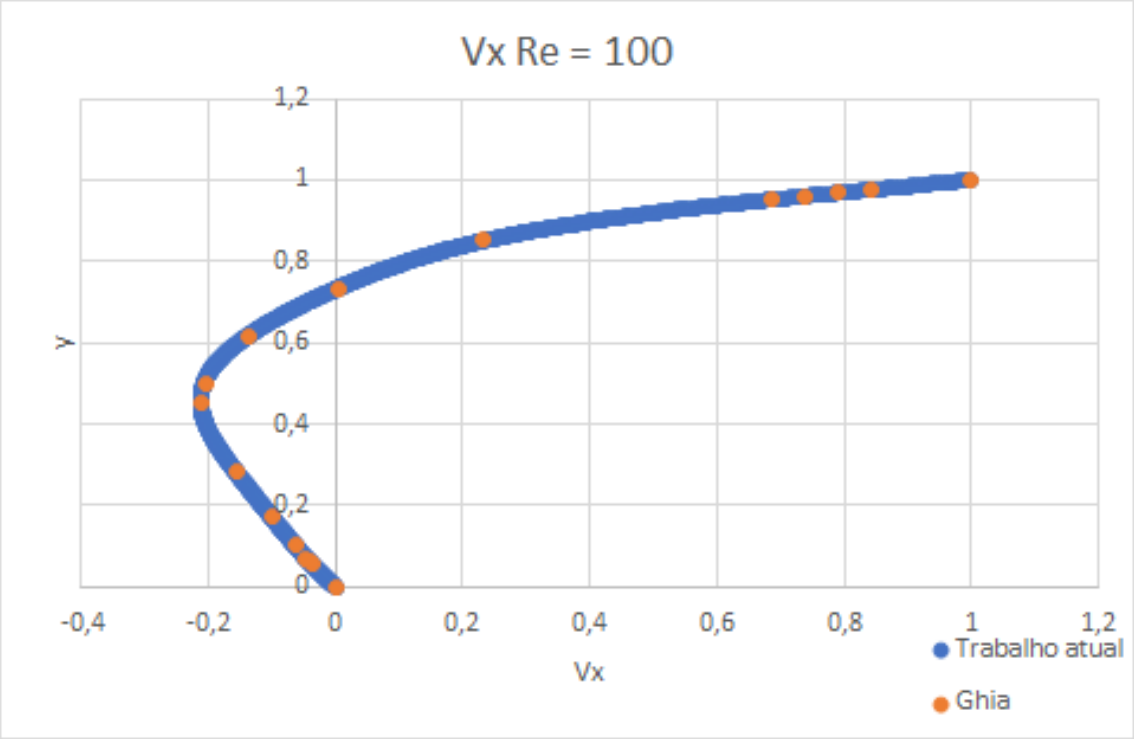


Figura 5.7: Comparação Lid - Driven para $Re = 100$

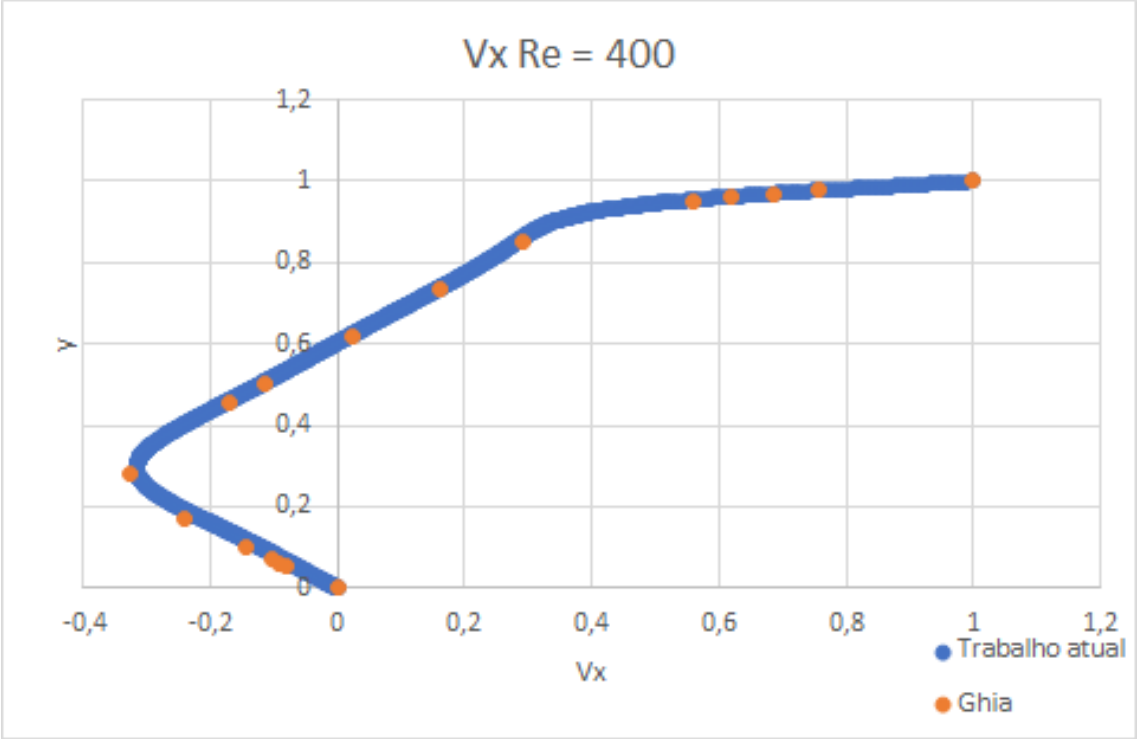


Figura 5.8: Comparação Lid - Driven para $Re = 400$

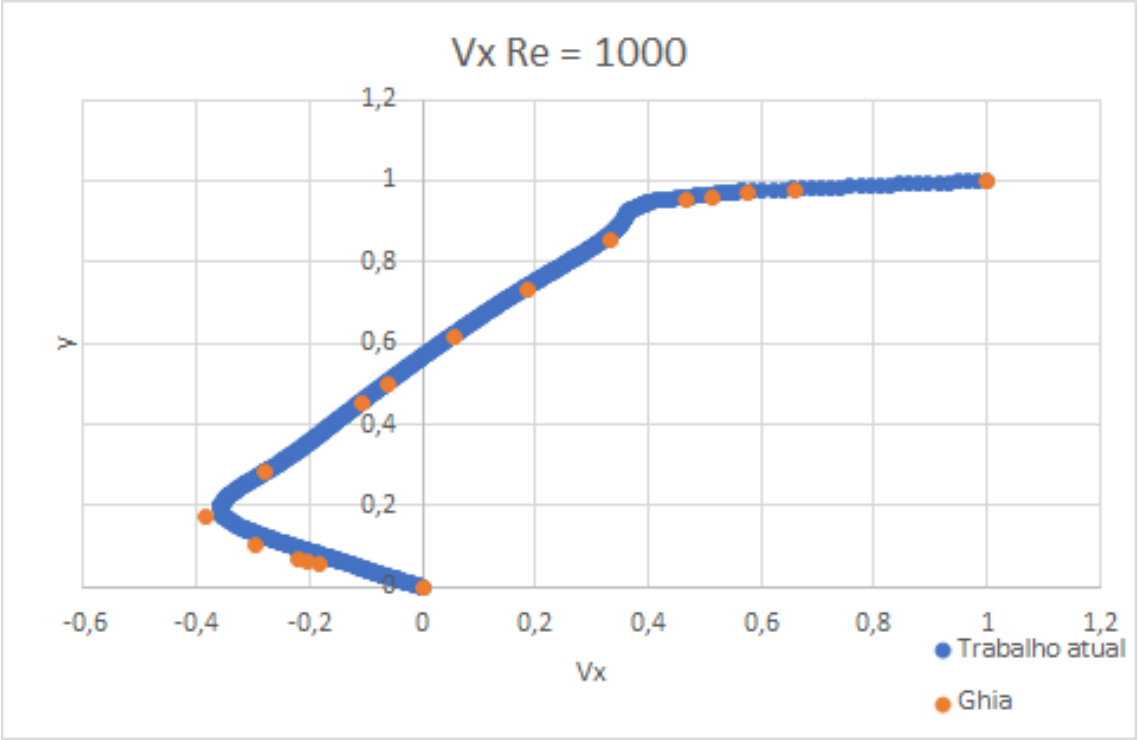


Figura 5.9: Comparação Lid - Driven para $Re = 1000$

Capítulo 6

Cálculo do Arrasto e Sustentação

6.1 Descrição do Escoamento

Como mencionado, este escoamento será caracterizado como livre e ao redor do cilindro. Isto é, o cilindro está muito afastado de quaisquer eventuais bordas do domínio da simulação. Assim, é possível simplificar a descrição do problema como

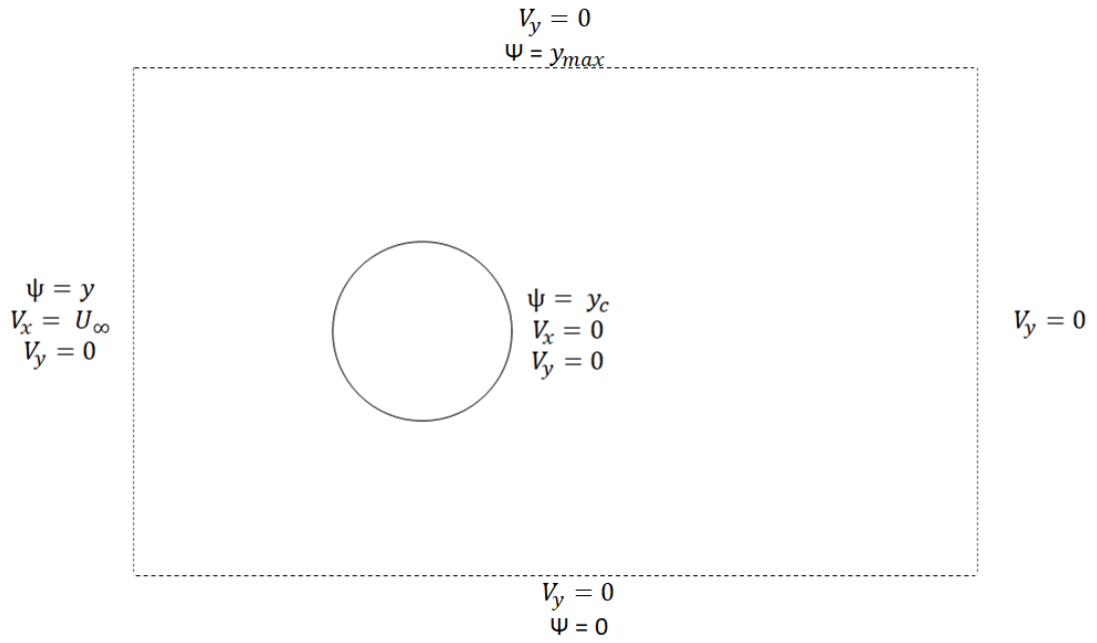


Figura 6.1: Descrição das Condições de Contorno do Escoamento Livre ao redor do Cilindro

Onde y_c é a coordenada y do centro do cilíndro no domínio.

6.2 Equações do Problema

Este problema revolve ao redor de duas principais grandezas, a Força de Sustentação e a Força de Arrasto. O trabalho elaborado em (J.-Z. WU, 2007) mostra que a região próxima ao cilindro oferece a maior contribuição para o cálculo da força sobre o cilindro. Isto é, uma vez que haja formação de vórtices no escoamento, estes não irão contribuir para o valor da força.

Para o cálculo desta força, (L. FIABANE, 2011) adota um subdomínio $\mathbb{F}$ composto pela região do escoamento próxima ao cilindro e delimitada pelo contorno $\boldsymbol{B}$, de forma que a força do escoamento sobre o cilindro pode ser calculada por

$$\boldsymbol{F}(t) = -\mu \int_{\mathbb{F}} \nabla^2 \omega_z \underset{\sim}{x}^{\perp} d\Omega + \mu \oint_{\boldsymbol{B}} ([(\underset{\sim}{n} \cdot \nabla)\omega]\underset{\sim}{x}^{\perp} + \omega \underset{\sim}{n}^{\perp}) d\Gamma \tag{6.1}$$

Porém, como descrito no trabalho de (L. FIABANE, 2011) e confirmado por (J.-Z. WU, 2007), à medida que o domínio $\mathbb{F}$ cresce, a contribuição do segundo termo no lado direito da Eq. 6.1 diminui. Desta forma, para os domínios $\mathbb{F}$ adotados, a contribuição deste segundo termo será de menos de 1% e, portanto, a Eq. 6.1 pode ser simplificada como

$$\boldsymbol{F}(t) \approx -\mu \int_{\mathbb{F}} \nabla^2 \omega_z \underset{\sim}{x}^{\perp} d\Omega \tag{6.2}$$

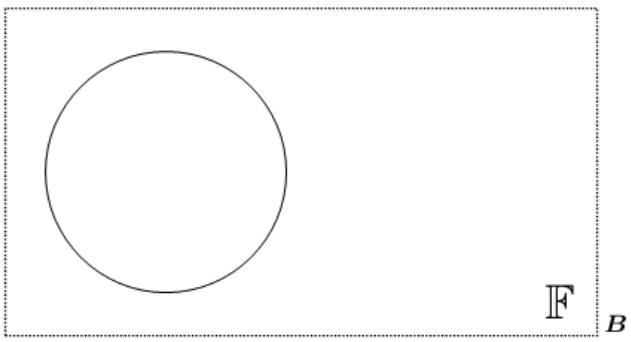


Figura 6.2: Descrição das Subregião ao redor do cilindro

Sendo $\underset{\sim}{x}^{\perp} = y\boldsymbol{e_x} - x\boldsymbol{e_y}$.

As componentes horizontal e vertical de $\boldsymbol{F}$ representam a força de arrasto e a força de sustentação, respectivamente e $\boldsymbol{F}$ representa a força do escoamento sobre o cilindro.

Para ser possível trabalhar com esta equação usando os métodos apresentados, é necessário passar para a forma fraca. Para isso, será refeito o procedimento que resultou na formulação da Eq. 2.28.

Isto é, encontra-se a seguinte formulação

$$F_L = \sum_{\mathbb{F}} -\mu y_e K \begin{bmatrix} \vdots \\ \omega_{z_i} \\ \vdots \end{bmatrix} \qquad F_D = \sum_{\mathbb{F}} \mu x_e K \begin{bmatrix} \vdots \\ \omega_{z_i} \\ \vdots \end{bmatrix} \tag{6.3}$$

Onde o somatório indica que apenas os elementos contidos em $\mathbb{F}$ serão contabilizados em um somatório das contribuições individuais de cada elemento para a força $\boldsymbol{F}$. x_e e y_e representam as coordenadas x e y do centro do elemento, respectivamente.

Os coeficientes de arrasto e sustentação podem ser calculados como

$$C_L = \sum_{\mathbb{F}} \frac{2x_e \nu K \begin{bmatrix} \vdots \\ \omega_{z_i} \\ \vdots \end{bmatrix}}{UD} \qquad C_D = \sum_{\mathbb{F}} \frac{-2y_e \nu K \begin{bmatrix} \vdots \\ \omega_{z_i} \\ \vdots \end{bmatrix}}{UD} \tag{6.4}$$

Dado o nível requerido de refinamento de malha e Δt para obter de forma satisfatoriamente acurada a convergência do cálculo do arrasto em valores mais elevados de Reynolds, será optado por deixá-lo de fora dos resultados de maior número de Reynolds.

Para esta simulação, foi adotada a seguinte malha

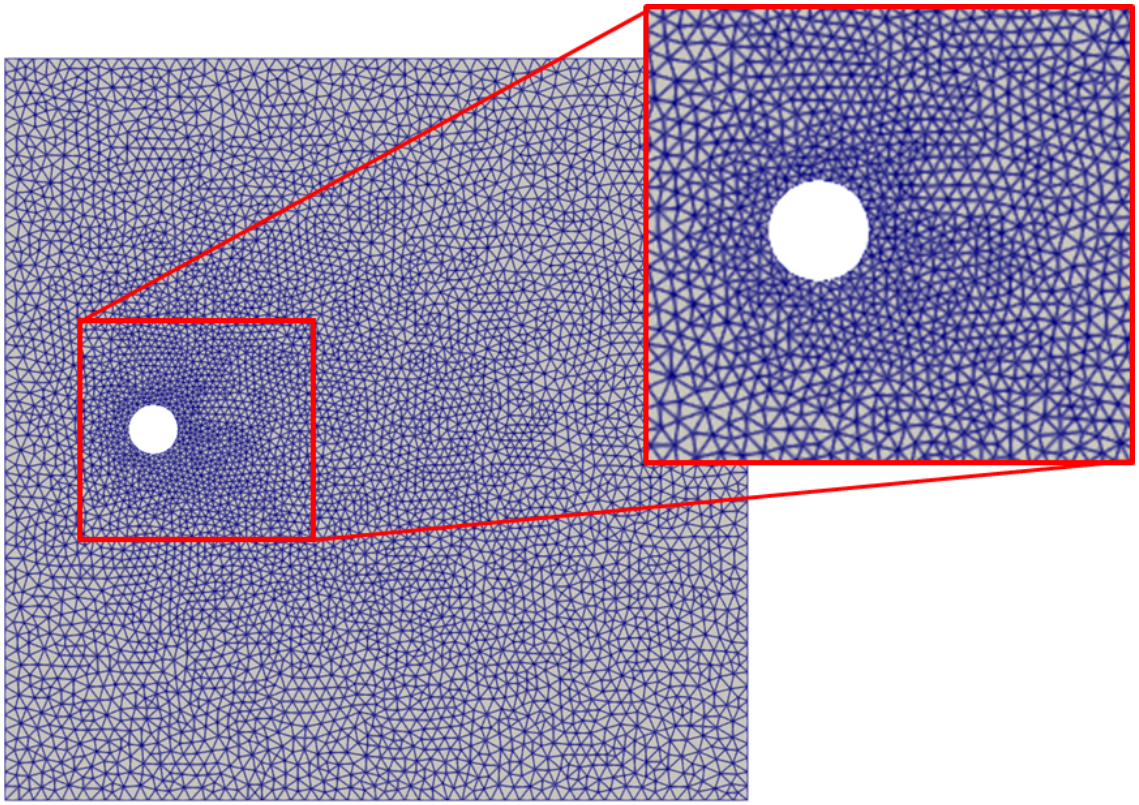


Figura 6.3: Visualização da malha para o escoamento livre ao redor do cilindro

Sendo os limites superiores e inferiores definidos como uma condição de *Far Field* ($v_y = 0$ e $\psi = y_{FF}$), o *inlet* afastado de $3D$ do centro do cilindro e o *outlet* afastado de $12D$ do centro do cilindro. A malha é composta de 10664 elementos triangulares e para todas as simulações foi usado um Δt de 0.01.

Para estas simulações, foi feito um refino maior da malha na vizinhança do cilindro dado que é a região de maior sensibilidade ao cálculo e, portanto, requer um tratamento mais cuidadoso para que haja a convergência do resultado. Isto é, haverá variações maiores de velocidades e, portanto, vorticidade na região mais próxima ao corpo em comparação a pontos mais distantes do escoamento.

Foram simulados os escoamentos para três casos. $Re = 50$, $Re = 200$ e $Re = 400$. Para cada um foram obtidos os valores de v_x, v_y, ψ e ω do escoamento desenvolvido bem como as linhas de corrente e, claro, os coeficientes de sustentação e arrasto. Para o cálculo destes últimos pelo método mencionado, diferentes subdomínios $\mathbb{F}$ foram adotados dependendo do valor de Re utilizado.

6.3 Resultados

6.3.1 $Re = 50$

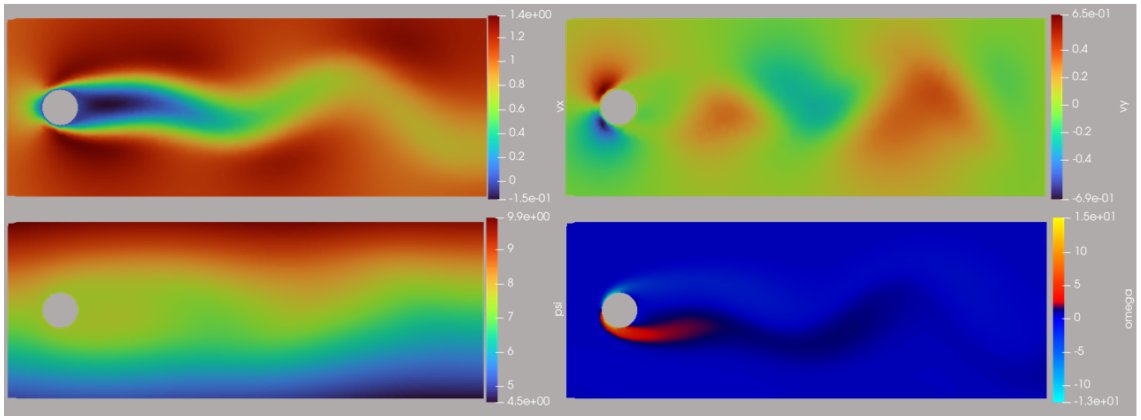


Figura 6.4: Visualização de v_x, v_y, ψ e ω para $Re = 50$.

Nesta imagem, cores quentes indicam valores positivos enquanto que cores frias implicam em valores negativos das grandezas estudadas. Na imagem, é possível ver o resultado da simulação para a componente horizontal da velocidade (v_x), a componente vertical da velocidade (v_y), a Função Corrente (ψ) e a Vorticidade (ω).

Como esperado, observa-se uma região de recirculação bem marcada após o cilindro e uma esteira com oscilação pouco intensa, resultado este esperado para tal valor de Re.

A velocidade em X apresenta um comportamento ainda sem esteiras e com uma zona de valores negativos, características de recirculação, na região de baixa pressão que se forma logo após o cilindro.

Também, analisando em Y, é possível notar valores positivos para v_y acima do centro do cilindro e valores negativos para a região abaixo deste ponto. Tal comportamento mostra, como esperado, a divisão da porção do escoamento que vai diretamente ao encontro do corpo, sendo uma uma parte ganhando uma velocidade vertical para cima, para contornar o cilindro por cima e outra direcionada vertical para baixo, para contornar o cilindro por baixo.

O gráfico da vorticidade, mostrando a geração de vórtices após o cilindro também é um resultado esperado.

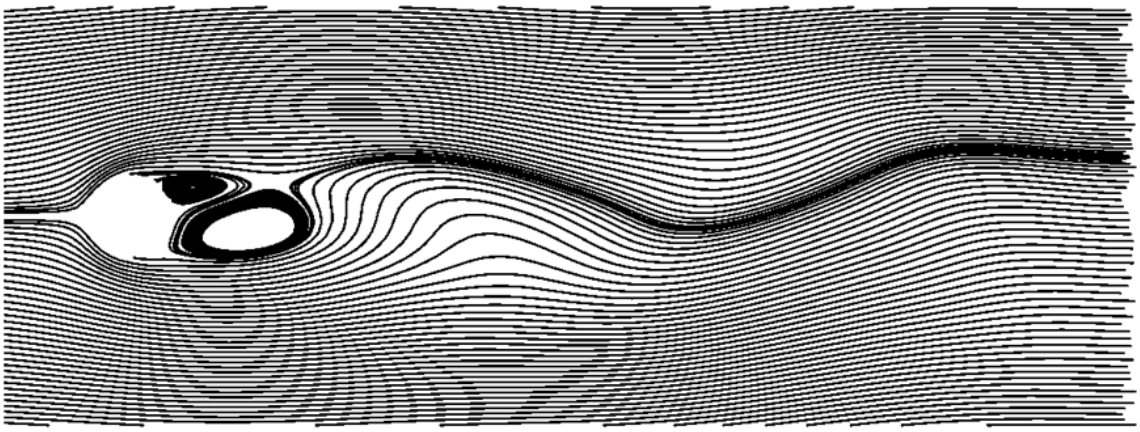


Figura 6.5: Linhas de corrente para $Re = 50$.

Por meio das linhas de corrente, fica bem marcada a região de recirculação e o caráter pouco oscilatório da esteira atrás do cilindro. Com estes gráficos, é possível observar que para este valor relativamente baixo de Re ainda não há o aparecimento das esteiras de von Kármán. Estas esteiras são explicadas na literatura como o descolamento e aparecimento de esteiras de vórtices para valores mais elevados de Reynolds. Estas serão mais notáveis para valores mais elevados de Re adotados neste trabalho.

Figura 6.6: Contribuição de $\nabla^2\omega$ para as forças de sustentação e arrasto.

Seguindo a metodologia adotada em (L. FIABANE, 2011), uma delimitação da região do cálculo dos coeficientes de arrasto e sustentação foi feita em $-0.75D$ na dianteira do cilindro, $1D$ posterior ao cilindro e $\pm 0.95D$ acima e abaixo do cilindro, permitindo a contabilização da porção de $\nabla^2\omega$ que mais influencia a força sobre o cilindro.

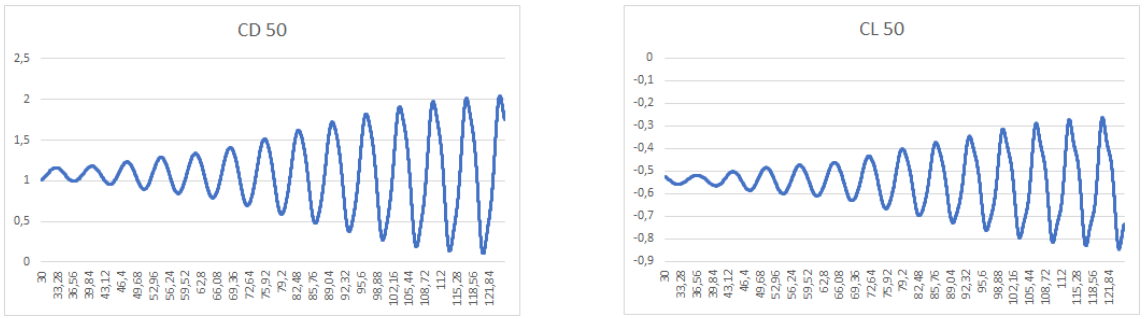


Figura 6.7: Coeficientes de arrasto e sustentação para $Re = 50$ calculados para diversos valores de tempo adimensional.

Para este valor suficientemente baixo de Reynolds, o coeficiente de arrasto apresenta um comportamento dentro do esperado, sendo este valor obtido condizente com a física ($C_D > 0$) e também apresenta caráter oscilatório, já confirmado nas análises anteriores dos outros valores para este escoamento.

Quanto ao coeficiente de sustentação, também apresentando caráter oscilatório, um valor constantemente negativo indica separação laminar do escoamento porém sem transição para o regime turbulento. Dado o valor baixo de Re esta hipótese pode ser confirmada.

Um coeficiente de sustentação negativo indica que a força de sustentação está orientada oposta ao esperado. Assim, um coeficiente de sustentação que oscila entre positivo e negativo implica em uma força de sustentação que constantemente alterna de sentido. Tal comportamento causa vibrações no corpo imerso no escoamento.

Neste caso estudado, o cilindro está fixo e portanto não se move. No entanto, em um caso onde o corpo possua liberdade de movimento, seria possível observá-lo oscilando quando imerso no escoamento em questão.

6.3.2 $Re = 200$

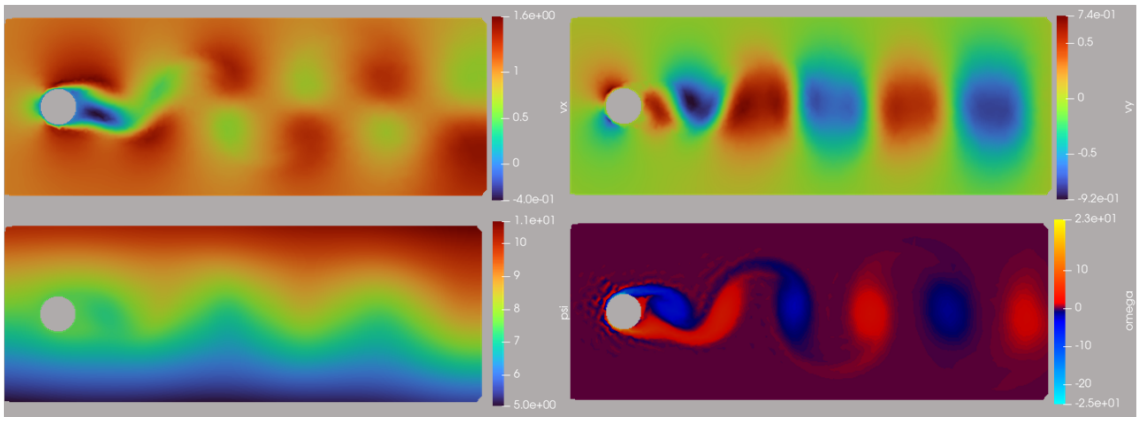


Figura 6.8: Visualização de v_x, v_y, ψ e ω para $Re = 200$.

Observa-se uma região pequena de recirculação após o cilindro e uma esteira que começa a se formar, indicando a transição para o regime turbulento. Resultado este esperado para tal valor de Re.

A velocidade em X apresenta um comportamento de início de formação de esteiras e com uma pequena zona de valores negativos, características ainda um pouco de recirculação na região de baixa pressão que se forma logo após o cilindro.

Também, analisando em Y, é possível notar agora fortes variações na velocidade vertical após o cilindro.

O gráfico da vorticidade, mostrando a geração de vórtices após o cilindro agora também indica o início de formação de esteiras.

Estes resultados são esperados dentro do aspecto de regime de transição entre escoamento laminar e turbulento.

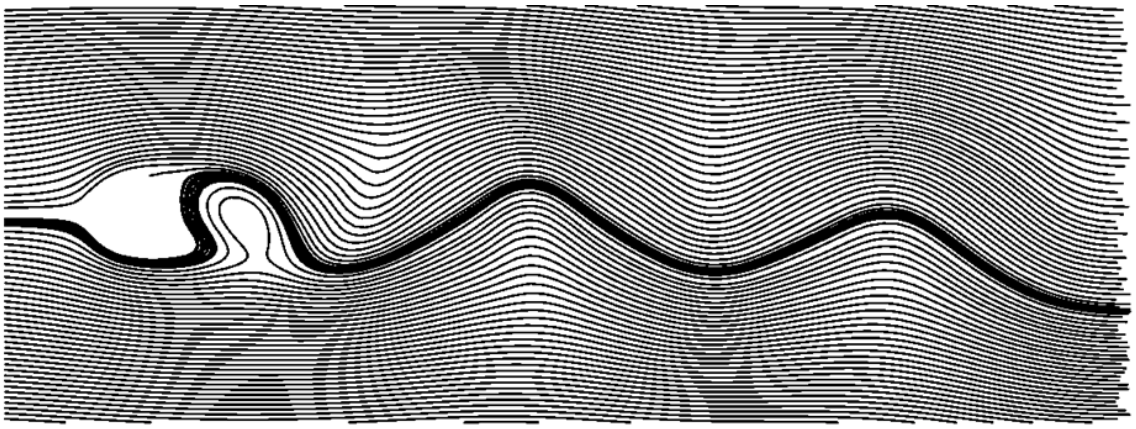


Figura 6.9: Linhas de corrente para $Re = 200$.

Aqui, as linhas de corrente marcam um caráter oscilatório mais intenso atrás do cilindro.

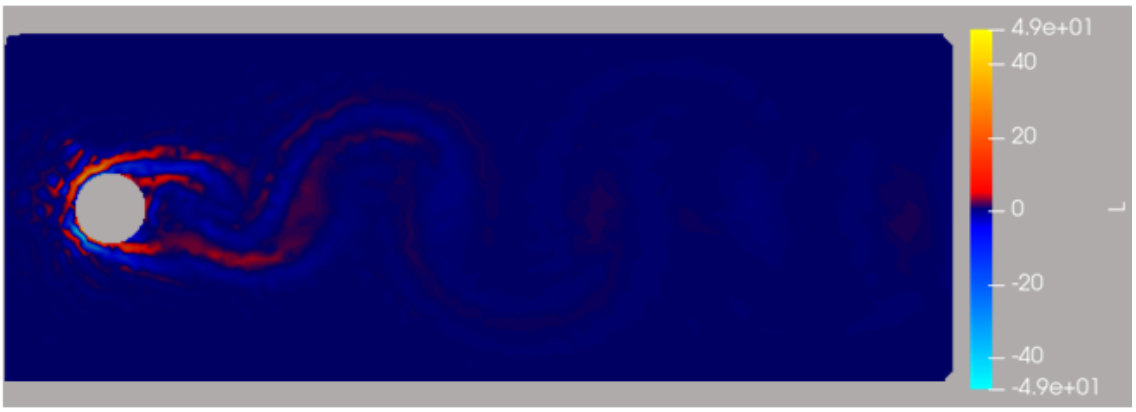


Figura 6.10: Contribuição de $\nabla^2\omega$ para as forças de sustentação e arrasto.

Seguindo a metodologia adotada em (L. FIABANE, 2011), uma delimitação da região do cálculo dos coeficientes de arrasto e sustentação foi feita em $-0.75D$ na dianteira do cilindro, $0.857D$ posterior ao cilindro e $\pm 0.95D$ acima e abaixo do cilindro para calcular a força sobre o corpo.

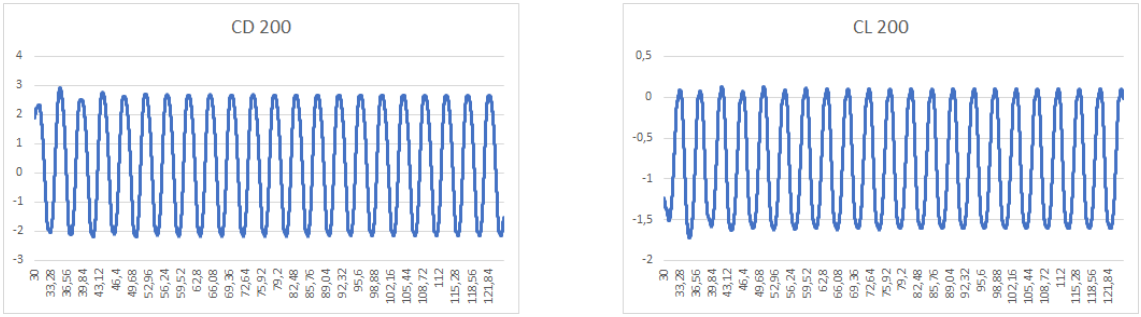


Figura 6.11: Coeficientes de sustentação para $Re = 200$ calculado para diversos valores de tempo adimensional.

Desta vez, o coeficiente de arrasto encontrado apresentou não-convergência deste resultado. Valores negativos para o coeficiente de arrasto indicam redução na força de arrasto e que o corpo está acelerando dentro do escoamento. Porém, dado que o cilindro é fixo e rígido, não há aceleração deste e portanto seu valor jamais poderia ser negativo.

Normalmente, para este caso de não-convergência, uma malha mais refinada, especialmente na região próxima ao corpo e na região de formação de vórtices, poderia ser adotada para tentar obter a convergência deste resultado.

No entanto, o coeficiente de sustentação apresenta oscilação predominantemente no espectro negativo porém atingindo valores positivos, agora. Isto é, o comportamento descrito anteriormente ainda está presente porém, dado o Reynolds de transição, pequenos pontos de turbulência estão contribuindo para valores positivos.

6.3.3 $Re = 400$

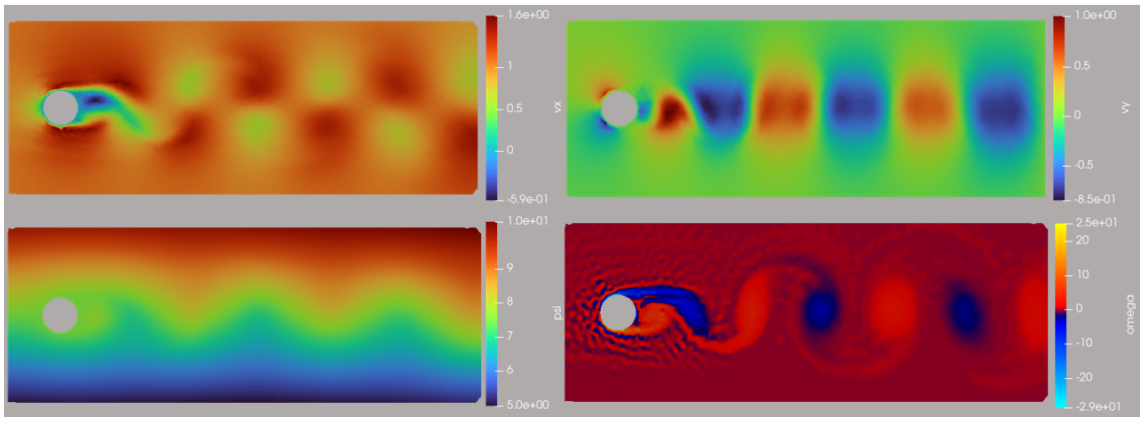


Figura 6.12: Visualização de v_x, v_y, ψ e ω para $Re = 400$.

Agora, para tal valor elevado de Reynolds, é possível observar a formação de esteiras de von Kármán.

O gráfico da vorticidade, mostrando a geração de vórtices após o cilindro agora também indica a presença das esteiras.

Estes resultados são esperados dentro do aspecto de regime de turbulento do escoamento.

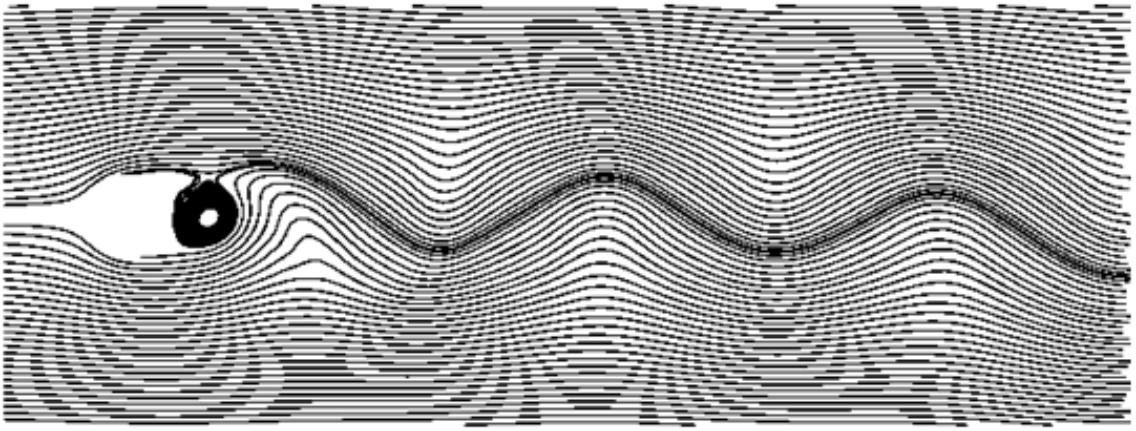


Figura 6.13: Linhas de corrente para $Re = 400$.

Figura 6.14: Contribuição de $\nabla^2\omega$ para as forças de sustentação e arrasto.

Seguindo a metodologia adotada em (L. FIABANE, 2011), uma delimitação da região do cálculo dos coeficientes de arrasto e sustentação foi feita em $-0.75D$ na dianteira do cilindro, $0.75D$ posterior ao cilindro e $\pm 0.95D$ acima e abaixo do cilindro.

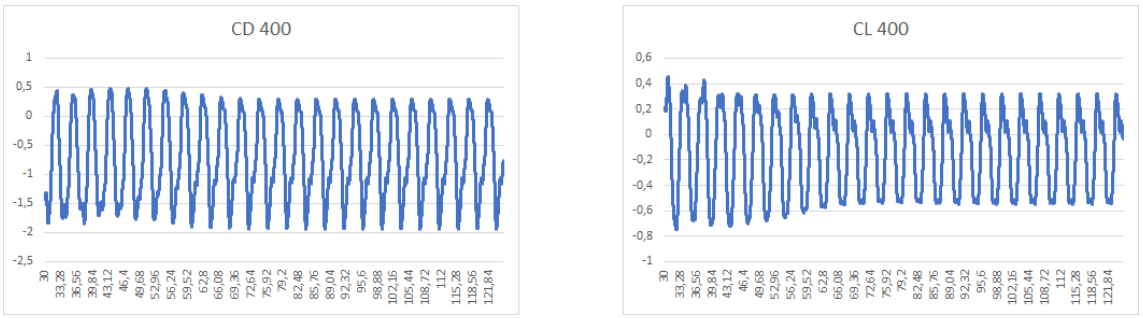


Figura 6.15: Coeficientes de sustentação para $Re = 400$ calculado para diversos valores de tempo adimensional.

Novamente o coeficiente de arrasto não convergiu pelo mesmo motivo do caso anterior. Aqui uma malha mais refinada também poderia ser adotada para tentar contornar este problema.

O coeficiente de sustentação agora oscila com um valor médio próximo ao zero. Agora com os efeitos de descolamento laminar praticamente extintos, é compreensível observar um valor médio nulo já que uma forma geométrica como o cilindro, dado a sua simetria e o fato de estar estático (cilindros rotacionando são capazes de gerar sustentação) implicam na incapacidade de gerar sustentação.

No entanto, o comportamento turbulento do escoamento promove tais oscilações na força de sustentação, mas sempre mantendo um valor médio nulo.

Conclusão

Este trabalho visou resolver computacionalmente um problema de escoamento livre ao redor de um cilindro por meio do método de elementos finitos usando a formulação Corrente-Vorticidade da equação de Navier-Stokes.

Para isto, a discretização espacial contou com o método dos resíduos ponderados e o método de Galerkin com interpolação de primeira ordem.

A execução do grande número de operações matemáticas exigidas para o cálculo foram feitas em computador por meio de um código em Python, que se encontra no Apêndice A deste trabalho e está disponível a todos que desejarem utilizá-lo.

O código foi validado usando casos mais simples ou cuja solução já foi obtida em outros trabalhos. Foram esses casos o escoamento entre placas planas e paralelas (*Hagen-Poiseuille*) e o escoamento em cavidade (*Lid-Driven*) e as respostas encontradas apresentaram comparações muito satisfatórias com a referência.

As simulações finais apresentaram comportamento esperado para tais configurações e permitiram observar diferentes caracterizações para os valores de Reynolds estudados.

Para Reynolds mais baixo, o código identificou a região de recirculação e o caráter levemente oscilatório do escoamento. Os valores dos coeficientes de arrasto e sustentação também foram condizentes com a física deste tipo de problema.

Para valores mais altos de Reynolds, no entanto, embora comportamentos oscilatórios do escoamento, da recirculação e do coeficiente de sustentação tenham sido satisfatórios, o cálculo mais preciso do coeficiente de arrasto requereria uma malha muito mais refinada e um passo de tempo muito menor. Infelizmente, tais tentativas de obter este resultado foram deixadas de lado visto que a capacidade computacional necessária para esta execução não estiveram disponíveis durante a realização deste trabalho.

Felizmente, dentre os diversos resultados obtidos, foi possível observar diversos comportamentos esperados para os coeficientes de sustentação e arrasto para um cilindro em escoamento livre para diferentes valores de Reynolds. O caráter oscilatório destes coeficientes, condizente com valores um pouco mais elevados de Reynolds, o caráter predominantemente negativo do coeficiente de sustentação, compatível com o descolamento laminar para Reynolds mais baixos, um valor constantemente positivo do coeficiente de arrasto para este Reynolds e, para Reynolds mais elevados, o caráter oscilatório com média nula para o coeficiente de sustentação. Todos os valores obtidos foram comparáveis com as referências utilizadas e comportamentos tais como região de recirculação e esteiras de von Kármán foram observados.

Referências Bibliográficas

Gustavo R. Anjos. *Computação Científica para Engenheiros*. PEM/COPPE/UFRJ, 2021.

Gustavo R. Anjos. Projeto final: Mecânica dos fluidos e transmissão de calor computacional. *apostila do Curso-UFRJ*, 2022.

Alan T.; PRITCHARD-Philip J. FOX, Robert W.; MCDONALD. *Introdução a Mecânica dos Fluidos*. Editora LTC, 7 edition, 2010.

Adam Harder. Experimental characterization of turbulent flow around cylinder arrays. 2012.

L.-X. ZHUANG J.-Z. WU, X.-Y. LU. Integral force acting on a body due to local flow structures. *JOURNAL OF FLUID MECHANICS*, 576, 2007.

TED BELYTSCHK JACOB FISH. *A First Course in Finite Elements*. John Wiley & Sons Ltd, 2007.

OLIVIER CADOT L. FIABANE, M. GOHLKE. Characterization of flow contributions to drag and lift of a circular cylinder using volume expression of the fluid force. *European Journal of Mechanics*, 2011.

P.; SEETHARAMU K. N. LEWIS, R. W.; NITHIARASU. *Fundamentals of the Finite Element Method for Heat and Fluid Flow*. John Wiley & Sons Ltd, 2004.

Ronald L. Panton. *Incompressible Flow*. John Wiley & Sons, Ltd, 2013.

N. PONTES, J.; MANGIAVACCHI. *Fenômenos de Transferência com Aplicações a Ciências Físicas e a Engenharia*. Apostila do Curso-UFRJ, 1 edition, 2009.

KANKANHALLY N. SEETHARAMU ROLAND W. LEWIS, PERUMAL NITHIARASU. *Fundamentals of the Finite Element Method for Heat and Fluid Flow*. John Wiley & Sons Ltd, 2004.

C. T. SHIN U. GHIA, K. N. GHIA. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *JOURNAL OF COMPUTATIONAL PHYSICS*, 1982.

John M. Cimbala Yunus A. Çengel. *Mecânica dos Fluidos: Fundamentos e Aplicações*. McGraw-Hill Education, 4 edition, 2017.

Anexos

6.4 Apêndice A

6.4.1 Código Fonte

Solver

```
#Carregamento das bibliotecas usadas
import numpy as np
import meshio
from time import time

#Funcao que escreve um txt contendo os parametros
   do escoamento e dados da simulacao
def write_data(U,mesh,nu,rho,dt,t_lim,ne,npoints,
   t_calc):
path = r"C:/Users/User/Documents/TCC/results/"
f = open(path+mesh+".txt",'w')
f.write("U = "+str(U) + "\n")
f.write("mesh = "+mesh+".msh; num elementos = "+str
   (ne)+"; num pontos = "+str(npoints)+"\n")
f.write("rho = "+str(rho)+"; nu = "+str(nu)+"; mu =
    "+str(rho*nu)+"\n")
f.write("t = "+str(t_lim)+"; dt = "+str(dt)+";
   tempo elapsado = "+str(t_calc))
f.close()

#Funcao que escreve os valores das variaveis
   calculadas em todos os pontos em cada iteracao
def write_Results(X,Y,Var,name,ne):
path = r"C:/Users/User/Documents/TCC/results/"
f = open(path+name+"_"+str(ne)+".txt",'w')
```

```
for i in range(len(X)):
f.write(str(i + 1) + ";")
f.write(str(X[i]) + ";")
f.write(str(Y[i]) + ";")
for j in range(len(Var)):
f.write(str(Var[j][i]) + ";")
f.write("\n")
f.close()

#Funcao de leitura da malha usando a biblioteca
   Meshio 5.0.2
def get_Mesh(name):
path = r"C:/Users/User/Documents/TCC/mesh/"
file_name = name+".msh"

#Leitura do arquivo contendo a malha
msh = meshio.read(path+file_name)

#Valores das coordenadas X e Y de cada ponto
X = msh.points[:,0]
Y = msh.points[:,1]

#Matriz de conectividade dos elementos
IEN = msh.cells[1][1]

#Matriz de conectividade dos elementos do contorno
IENbound = msh.cells[0][1]

#Tipo dos grupos f[U+FFFD]sicos
IENboundElemType = msh.cell_data['gmsh:physical'
   ][0] - 1

#Nome das condicoes de contorno para cada vertice/
   face do IENbound
boundNames = list(msh.field_data.keys())

#Relaciona elementos ao indice da condicao de
   contorno
```

```
IENboundElem = [boundNames[elem] for elem in
   IENboundElemType]

#Numero de pontos e de elementos da malha
npoints = len(X)
ne = IEN.shape[0]

#Aplicacao do tipo de condicao de contorno em cada
   ponto da malha que for do contorno
cc = np.unique(IENbound.reshape(IENbound.size))
ccName = [[] for i in range(npoints)]
for elem in range(0,len(IENbound)):
if not ccName[IENbound[elem][0]] or ccName[IENbound
   [elem][0]] == "inlet" or ccName[IENbound[elem
   ][0]] == "outlet":
ccName[IENbound[elem][0]] = IENboundElem[elem]
if not ccName[IENbound[elem][1]] or ccName[IENbound
   [elem][1]] == "inlet" or ccName[IENbound[elem
   ][1]] == "outlet":
ccName[IENbound[elem][1]] = IENboundElem[elem]
if IENboundElem[elem] == "tampa":
ccName[IENbound[elem][0]] = IENboundElem[elem]
ccName[IENbound[elem][1]] = IENboundElem[elem]

return npoints,ne,X,Y,IEN,cc,ccName

#Calculo das matrizes Kxx, Kyy, Kxy e Kyx de um
   elemento
def get_Kxye(x1,y1,x2,y2,x3,y3):

#Area do elemento
A = (x1*(y2 - y3) + x2*(y3-y1) + x3*(y1-y2))/2

b1=y2-y3
b2=y3-y1
b3=y1-y2
c1=x3-x2
c2=x1-x3
c3=x2-x1
```

```
b = np.array([[b1,b2,b3]])/(2*A)
c = np.array([[c1,c2,c3]])/(2*A)

kxe = A*np.dot(b.transpose(),b)
kye = A*np.dot(c.transpose(),c)
kxye = A*np.dot(b.transpose(),c)
kyxe = A*np.dot(c.transpose(),b)

return kxe,kye,kxye,kyxe

#Calculo da matriz K de um elemento. Esta funcao
   calcula K de forma direta e mais rapida que a
   soma de Kxx e Kyy
def get_Ke(x1,y1,x2,y2,x3,y3):

#Area do elemento
A = (x1*(y2 - y3) + x2*(y3-y1) + x3*(y1-y2))/2

b1=y2-y3
b2=y3-y1
b3=y1-y2
c1=x3-x2
c2=x1-x3
c3=x2-x1

B = np.array([[b1,b2,b3],[c1,c2,c3]])
BT = B.transpose()

Ke = 1/(4*A)*np.dot(BT,B)

return Ke

#Calculo da matriz M de um elemento
def get_Me(x1,y1,x2,y2,x3,y3):

#Area do elemento
A = (x1*(y2 - y3) + x2*(y3-y1) + x3*(y1-y2))/2
```

```
Me = (A/12.0)*np.array
   ([[2.0,1.0,1.0],[1.0,2.0,1.0],[1.0,1.0,2.0]])

return Me

#Calculo da matriz Gx de um elemento
def get_Gxe(y1,y2,y3):

b1=y2-y3
b2=y3-y1
b3=y1-y2

return (1.0/6.0)*np.array([ [b1, b2, b3],[b1, b2,
   b3],[b1, b2, b3] ])

#Calculo da matriz Gy de um elemento
def get_Gye(x1,x2,x3):

c1=x3-x2
c2=x1-x3
c3=x2-x1

return (1.0/6.0)*np.array([ [c1, c2, c3],[c1, c2,
   c3],[c1, c2, c3] ])

#Calculo da matriz vG do sistema global para cada
   iteracao
def generate_vG(Vx,Vy,Gx,Gy):

return np.dot(np.diag(Vx),Gx) + np.dot(np.diag(Vy),
   Gy)

#Inicializacao do campo de velocidades em X. Neste
   caso, inicializa-se com zero
def get_Initial_Vx(X,Y,minX,maxX,minY,maxY):

n = len(X)

Vx = np.zeros(n,dtype = 'float')
```

```
return Vx

#Inicializacao do campo de velocidades em Y. Neste
    caso, inicializa-se com zero
def get_Initial_Vy(X,Y,minX,maxX,minY,maxY):

n = len(X)

Vy = np.zeros(n,dtype = 'float')

return Vy

#Funcao principal do codigo. Chama as outras
    funcoes
def main():

#Nome da malha (.msh)
name = "cilindro"

#Variaveis que armazenam os valores calculados para
     cada ponto de uma variavel a cada iteracao

W = [] #Vorticidade
Psi = [] #Funcao Corrente
Vx = [] #Componente horizontal da velocidade
Vy = [] #Componente vertical da velocidade
L = [] #Contribuicao para a forca de arrasto e
    sustentacao

#Parametros do tempo inicial final e step
t=0
tlim = 50
dt = 0.1

#definicao do tipo de MDF usado
MDF_w = 1 #0: explicito / 1: implicito / 0.5: crank
    -nicholson
```

```
#Parametros de geometria, input e propriedade do
    fluido
yp = 2.5 #Posicao vertical do centro do cilindro
U = 1 #Velocidade na entrada
nu = 0.01 #Viscosidade cinem[U+FFFD]tica
rho = 1 #Massa especifica
mu = nu*rho #Viscosidade din[U+FFFD]mica

t1 = time() #Inicio da contagem de tempo da
    simulacao

npoints,ne,X,Y,IEN,cc,ccName = get_Mesh(name) #
    Leitura da malha

#Inicializacao do campo de velocidades e das CC
    para Psi, Vx e Vy
Vx.append(get_Initial_Vx(X,Y,min(X),max(X),min(Y),
    max(Y)))
Vy.append(get_Initial_Vy(X,Y,min(X),max(X),min(Y),
    max(Y)))
Psi_cc = np.zeros(npoints,dtype = 'float')

for i in cc:
boundary_type = ccName[i]
if boundary_type == "tampa":
Vx[-1][i] = U
Vy[-1][i] = 0
Psi_cc[i] = 0
elif boundary_type == "paredeInf":
Vx[-1][i] = 0
Vy[-1][i] = 0
Psi_cc[i] = 0
elif boundary_type == "paredeSup":
Vx[-1][i] = 0
Vy[-1][i] = 0
Psi_cc[i] = Y[i]
elif boundary_type == "inlet":
Vx[-1][i] = U
Vy[-1][i] = 0
```

```
Psi_cc[i] = Y[i]
elif boundary_type == "outlet":
Vy[-1][i] = 0
elif boundary_type == "farField":
Vy[-1][i] = 0
Psi_cc[i] = Y[i]
elif boundary_type == "cilindro":
Vx[-1][i] = 0
Vy[-1][i] = 0
Psi_cc[i] = yp

#Construcao das matrizes globais partindo das
   matrizes elementares
K = np.zeros((npoints,npoints),dtype = 'float')
Kx =  np.zeros((npoints,npoints),dtype = 'float')
Ky = np.zeros((npoints,npoints),dtype = 'float')
Kxy =  np.zeros((npoints,npoints),dtype = 'float')
M = np.zeros((npoints,npoints),dtype = 'float')
Gx = np.zeros((npoints,npoints),dtype = 'float')
Gy = np.zeros((npoints,npoints),dtype = 'float')

for e in range(0,ne):

vertices = IEN[e]
x1 = X[vertices[0]]
y1 = Y[vertices[0]]
x2 = X[vertices[1]]
y2 = Y[vertices[1]]
x3 = X[vertices[2]]
y3 = Y[vertices[2]]

Ke = get_Ke(x1,y1,x2,y2,x3,y3)
Kxe,Kye,Kxye,Kyxe = get_Kxye(x1,y1,x2,y2,x3,y3)
Me = get_Me(x1,y1,x2,y2,x3,y3)
Gxe = get_Gxe(y1,y2,y3)
Gye = get_Gye(x1,x2,x3)

for i in range(0,3):
ii = IEN[e,i]
```

```
for j in range(0,3):
jj = IEN[e,j]
Gx[ii,jj] = Gx[ii,jj] + Gxe[i,j]
Gy[ii,jj] = Gy[ii,jj] + Gye[i,j]
M[ii,jj] = M[ii,jj] + Me[i,j]
K[ii,jj] = K[ii,jj] + Ke[i,j]
Kx[ii,jj] = Kx[ii,jj] + Kxe[i,j]
Ky[ii,jj] = Ky[ii,jj] + Kye[i,j]
Kxy[ii,jj] = Kxy[ii,jj] + Kxye[i,j]
Ksum = Kx + Ky #Forma alternativa de calcular K

W.append(np.linalg.solve(M,np.dot(Gx,Vy[-1]) - np.
   dot(Gy,Vx[-1]))) #W inicial gerado com Vx e Vy
   inicial

L.append(np.dot(Ksum,W[-1])) #Contribuicao para as
   forcas na iteracao inicial

#Matrizes especiais para cada variavel. Estas terao
    condicoes de contorno implementadas nelas
K_psi = K.copy()
M_vx = M.copy()
M_vy = M.copy()

#Calculo de Psi na iteracao inicial e aplicacao das
    CC nas matrizes
c = np.dot(M,W[-1])
for i in cc:
boundary_type = ccName[i]
if boundary_type == "tampa" or boundary_type == "
   cilindro" or boundary_type == "inlet" or
   boundary_type == "paredeInf" or boundary_type ==
    "paredeSup": #separar pontos do contorno e
   aplicar CC neles
K_psi[i] = np.zeros(npoints,dtype = 'float')
K_psi[i,i] = 1.0
c[i] = Psi_cc[i]
M_vx[i] = np.zeros(npoints,dtype = 'float')
M_vx[i,i] = 1.0
```

```
M_vy[i] = np.zeros(npoints,dtype = 'float')
M_vy[i,i] = 1.0
if boundary_type == "farField":
K_psi[i] = np.zeros(npoints,dtype = 'float')
K_psi[i,i] = 1.0
c[i] = Psi_cc[i]
M_vy[i] = np.zeros(npoints,dtype = 'float')
M_vy[i,i] = 1.0
if boundary_type == "outlet":
M_vy[i] = np.zeros(npoints,dtype = 'float')
M_vy[i,i] = 1.0
Psi.append(np.linalg.solve(K_psi,c))

t += dt #Contabilizado uma iteracao alem do estado
    inicial

#Loop temporal
while t <= tlim:

#Calculo das CC para a vorticidade na iteracao
Wcc = np.linalg.solve(M,np.dot(Gx,Vy[-1]) - np.dot(
    Gy,Vx[-1]))

#Calculo de vG para a iteracao
vG = generate_vG(Vx[-1],Vy[-1],Gx,Gy)

#Calculo da vorticidade aplicando CC e calculo da
    contribuicao para as forcas
A = M/dt + nu*K + MDF_w*vG
b = np.dot((M/dt - (1 - MDF_w)*vG),W[-1])
for i in cc:
boundary_type = ccName[i]
if boundary_type == "farField" or boundary_type ==
    "tampa" or boundary_type == "cilindro" or
    boundary_type == "inlet" or boundary_type == "
    paredeInf" or boundary_type == "paredeSup":
A[i] = np.zeros(npoints,dtype = 'float')
A[i,i] = 1.0
b[i] = Wcc[i]
```

```
W.append(np.linalg.solve(A,b))
L.append(np.dot(Ksum,W[-1]))

#Calculo de Psi aplicando CC
c = np.dot(M,W[-1])
for i in cc:
boundary_type = ccName[i]
if boundary_type == "farField" or boundary_type ==
   "tampa" or boundary_type == "cilindro" or
   boundary_type == "inlet" or boundary_type == "
   paredeInf" or boundary_type == "paredeSup": #
   separar pontos do contorno e aplicar CC neles
c[i] = Psi_cc[i]
Psi.append(np.linalg.solve(K_psi,c))

#Calculo do novo campo de velocidades aplicanco CC
b_vx = np.dot(Gy,Psi[-1])
b_vy = np.dot(-Gx,Psi[-1])
for i in cc:
boundary_type = ccName[i]
if boundary_type == "tampa" or boundary_type == "
   inlet":
b_vx[i] = U
b_vy[i] = 0
elif boundary_type == "cilindro" or boundary_type
   == "paredeInf" or boundary_type == "paredeSup":
b_vx[i] = 0
b_vy[i] = 0
elif boundary_type == "outlet" or boundary_type ==
   "farField":
b_vy[i] = 0

Vx.append(np.linalg.solve(M_vx,b_vx))
Vy.append(np.linalg.solve(M_vy,b_vy))


t += dt #Contabilizacao da iteracao

t2 = time()
```

```
t_calc = t2 - t1
print("tempo de execucao: " + str(round(t_calc,2))
    + "s") #Tempo elapsado

#Registro dos resultados em txt
write_Results(X,Y,Psi,name+"_Psi",ne)
write_Results(X,Y,W,name+"_w",ne)
write_Results(X,Y,Vx,name+"_Vx",ne)
write_Results(X,Y,Vy,name+"_Vy",ne)
write_Results(X,Y,L,name+"_L",ne)
write_data(U,name,nu,rho,dt,tlim,ne,npoints,t_calc)

if __name__ == "__main__":
main()
```

Conversor para Paraview

```
#Carregamento das bibliotecas a serem usadas
import meshio
import numpy as np
import os
from datetime import datetime

#Leitura da malha
def read_Mesh(path,file_name):
file_name = file_name+".msh"
msh = meshio.read(path+file_name)
num_elements = msh.cells[1][1].shape[0]
return msh,num_elements

#Funcao principal da conversao de txt em formato
    Paraview
def main():

#Paths ate local de escrita dos arquivos vtk, ate
    os resultados calculados e ate a malha
path = r"C:/Users/User/Documents/TCC/paraview/"
path_res = r"C:/Users/User/Documents/TCC/results/"
path_mesh = r"C:/Users/User/Documents/TCC/mesh/"
```

```
mesh = "cilindro_farField"

now = datetime.now() #Momento em que a execucao
    comecou
date_time = now.strftime("-%d_%m_%Y-%H_%M_%S") #
    Data em que a execucao esta ocorrendo

msh,num_elements = read_Mesh(path_mesh,mesh) #
    Leitura da malha
cells = [(msh.cells[1][0],msh.cells[1][1])] #
    Elementos triangulares da malha

#Inicializacao das variaveis que vao guardar o
    campo de velocidades, Funcao Corrente,
    Vorticidade e contribuicao para o calculo das
    forcas
Vx = []
Vy = []
Psi = []
W = []
L = []

#Variavel que vai guardar o numero de iteracoes
num_values = 0

#Leitura de Vx e armazenamento na variavel
    correspondente
try:
file = open(path_res + mesh + "_Vx_"+str(
    num_elements)+".txt","r")
lines  = file.readlines()
n = len(lines)
num_values = len(lines[0].split(";")[3:-1])
for i in range(0,num_values):
Vx.append([])
for line in lines:
line = line.split(";")[0:-1]
values = list(map(float, line[3:]))
for i in range(0,num_values):
```

```
Vx[i].append(values[i])
file.close()
except:
print("Erro lendo Vx")
return

#Leitura de Vy e armazenamento na variavel
   correspondente
try:
file = open(path_res + mesh + "_Vy_"+str(
   num_elements)+".txt","r")
lines  = file.readlines()
n = len(lines)
for i in range(0,num_values):
Vy.append([])
for line in lines:
line = line.split(";")[0:-1]
values = list(map(float, line[3:]))
for i in range(0,num_values):
Vy[i].append(values[i])
file.close()
except:
print("Erro lendo Vy")
return

#Leitura de Psi e armazenamento na variavel
   correspondente
try:
file = open(path_res + mesh + "_Psi_"+str(
   num_elements)+".txt","r")
lines  = file.readlines()
n = len(lines)
for i in range(0,num_values):
Psi.append([])
for line in lines:
line = line.split(";")[0:-1]
values = list(map(float, line[3:]))
for i in range(0,num_values):
Psi[i].append(values[i])
```

```
file.close()
except:
print("Erro lendo Psi")
return

#Leitura da Vorticidade e armazenamento na variavel
    correspondente
try:
file = open(path_res + mesh + "_w_"+str(
   num_elements)+".txt","r")
lines  = file.readlines()
n = len(lines)
for i in range(0,num_values):
W.append([])
for line in lines:
line = line.split(";")[0:-1]
values = list(map(float, line[3:]))
for i in range(0,num_values):
W[i].append(values[i])
file.close()
except:
print("Erro lendo W")
return

#Leitura de L e armazenamento na variavel
   correspondente
try:
file = open(path_res + mesh + "_L_"+str(
   num_elements)+".txt","r")
lines  = file.readlines()
n = len(lines)
for i in range(0,num_values):
L.append([])
for line in lines:
line = line.split(";")[0:-1]
values = list(map(float, line[3:]))
for i in range(0,num_values):
L[i].append(values[i])
file.close()
```

```
except:
print("Erro lendo L")
return

#Criacao de uma pasta para guardar os arquivos
    Paraview
try:
os.makedirs(path+mesh+date_time)
except:
print("Erro criando diretorio")
return

#Conversao dos resultados txt em Paraview
for step in range(0,len(Vx)):
point_data={'vx':Vx[step]}
data_vy={'vy':Vy[step]}
data_psi={'psi':Psi[step]}
data_omega={'omega':W[step]}
data_L={'L':L[step]}
point_data.update(data_vy)
point_data.update(data_psi)
point_data.update(data_omega)
point_data.update(data_L)
meshio.write_points_cells(path+mesh+date_time+"/"+
    mesh+"_"+str(step + 1)+'.vtk',msh.points,cells,
    point_data=point_data,file_format = "vtk51")

if __name__ == "__main__":
main()
```

Pós Processamento da Sustentação e Arrasto

```
#Carregamento das bibliotecas a serem usadas
import numpy as np
import meshio

#Funcao que escreve um txt com todos os valores
    calculados
```

```
def write_Results(lift_vectors_for_each_step,mesh,
    str_val):

path = r"C:/Users/User/Documents/TCC/lift_drag/"

f = open(path+mesh+"_lift_vector.txt",'w')

f.write(str_val+"\n")
f.write("FD    FL    CD    CL"+"\n")

for i in range(len(lift_vectors_for_each_step)):

f.write(str(lift_vectors_for_each_step[i,0])+","+
    str(lift_vectors_for_each_step[i,1])+","+str(
    lift_vectors_for_each_step[i,2])+","+str(
    lift_vectors_for_each_step[i,3])+"\n")

f.close()

#Funcao de leitura da malha
def get_Mesh(name):

path = r"C:/Users/User/Documents/TCC/mesh/"

file_name = name+".msh"

#Abertura do arquivo de malha
msh = meshio.read(path+file_name)

#Coordenadas X e Y de cada ponto
X = msh.points[:,0]
Y = msh.points[:,1]

#Matriz de conectividade dos elementos
IEN = msh.cells[1][1]

#Leitura dos grupos fisicos
IEN_ElemType = msh.cell_data['gmsh:physical'][1] -
    1
```

```
#Leitura dos tipos de grupos fisicos de elementos
    bidimensionais
ElementsNames = list(msh.field_data.keys())

#Relaciona elementos ao indice do tipo de grupo
    fisico
IEN_Elem = [ElementsNames[elem] for elem in
    IEN_ElemType]

#Separacao de um sub grupo de elementos com um
    mesmo nome
subset_elements = []

ne = IEN.shape[0] #numero de elementos

for i in range(ne):

if ElementsNames[IEN_ElemType[i]] == "surface": #
    Apenas elementos caracterizados como surface

#ID dos 3 pontos que compoem o elemento triagular e
     coordenadas X e Y de cada um
subset_elements.append([IEN[i][0],IEN[i][1],IEN[i
    ][2],X[IEN[i][0]],Y[IEN[i][0]],X[IEN[i][1]],Y[
    IEN[i][1]],X[IEN[i][2]],Y[IEN[i][2]]])

return subset_elements,len(subset_elements)

#Funcao principal do calculo dos coeficientes de
    sustentacao e arrasto
def main():

#Leitura da malha
mesh = "cilindro_farField"
subset_elements,ne = get_Mesh(mesh)

path = r"C:/Users/User/Documents/TCC/results/"
```

```
#Variavel que vai conter o numero de iteracoes
num_values = 0

#Variavel que vai guardar a contribuicao para a
   forca em cada iteracao apra todos os pontos
L = []

#Leitura do txt resultado do solver e carregamento
   dos resultados na variavel L
try:
file = open(path + mesh + "_L_"+str(ne)+".txt","r")
lines  = file.readlines()
n = len(lines)
num_values = len(lines[0].split(";")[3:-1])
for i in range(0,num_values):
L.append([])
for line in lines:
line = line.split(";")[0:-1]
values = list(map(float, line[3:]))
for i in range(0,num_values):
L[i].append(values[i])
file.close()
except:
print("Erro lendo L")
return

#Dados do cilindro e velocidade de entrada
xc = 3 #Posicao horizontal do centro do cilindro
yc = 2.5 #Posicao vertical do centro do cilindro
U = 1 #Velocidade na entrada
D = 1 #Diamentro do cilindro

#Parametros do escoamento
nu = 1/200 #Viscosidade cinematica
rho = 1 #Massa especifica
mu = nu*rho #Viscosidade dinamica

#Definicao dos limites da regiao de medicao em
   relacao ao centro e diametro do cilindro
```

```
min_X = xc - 0.75*D
max_X = xc + 0.857*D
min_Y = yc - 0.95*D
max_Y = yc + 0.95*D

#String contendo os limites da regiao de medicao
str_val = "min X: " + str(min_X) + " " + "max X: "
   + str(max_X) + " " + "min Y: " + str(min_Y) + "
   " + "max Y: " + str(max_Y)

#Inicializacao do vetor contendo os valores
   calculados de forca e coeficientes em cada
   iteracao
lift_vectors_for_each_step = np.zeros((num_values
   ,4),dtype = 'float')

#Calculo das forcas e coeficientes
for i in range(num_values):
FL = 0
FD = 0
CL = 0
CD = 0
for j in range(ne):
x1 = subset_elements[j][3]
y1 = subset_elements[j][4]
x2 = subset_elements[j][5]
y2 = subset_elements[j][6]
x3 = subset_elements[j][7]
y3 = subset_elements[j][8]

#Se o elemento estiver completamente dentro dos
   limites, ele entra no calculo
if max(x1,x2,x3) <= max_X and min(x1,x2,x3) >=
   min_X and max(y1,y2,y3) <= max_Y and min(y1,y2,
   y3) >= min_Y:

#Leitura do valor de L de cada vertice do elemento
p1 = subset_elements[j][0]
p2 = subset_elements[j][1]
```

```
            p3 = subset_elements[j][2]
            L1 = L[i][p1]
            L2 = L[i][p2]
            L3 = L[i][p3]

            Le = (L1 + L2 + L3)/3 #L medio do elemento

            Xce = (x1 + x2 + x3)/3 #Posicao X do centro do
               elemento
            Yce = (y1 + y2 + y3)/3 #Posicao Y do centro do
               elemento

            #Somatorio de FL, FD, CL e CD elementares para uma
               iteracao
            FL = FL -mu*Yce*Le
            FD = FD + mu*Xce*Le
            CL = CL + 2*Xce*nu*Le/(U*D)
            CD = CD - 2*Yce*nu*Le/(U*D)

            lift_vectors_for_each_step[i] = [FD,FL,CD,CL]

            #Escreve os resultados em um txt
            write_Results(lift_vectors_for_each_step,mesh,
               str_val)

            if __name__ == "__main__":
            main()
```