Politécnica UFRJ

ESTUDO DE ESCOAMENTOS EM FILTROS DE PARTICULADOS DE EMISSÕES DE BIODIESEL

João Pedro Rodrigues Ferreira

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

Orientador: Gustavo Rabello dos Anjos

Rio de Janeiro

Dezembro de 2023

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Departamento de Engenharia Mecânica

DEM/POLI/UFRJ

ESTUDO DE ESCOAMENTOS EM FILTROS DE PARTICULADOS DE EMISSÕES DE BIODIESEL

João Pedro Rodrigues Ferreira

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

Aprovada por:

Prof. Gustavo Rabello dos Anjos, Ph.D.

Prof. Albino José Kalab Leiroz, Ph.D.

Prof. Nisio de Carvalho Lobo Brum, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

DEZEMBRO DE 2023

Rodrigues Ferreira, João Pedro

Estudo de Escoamentos em Filtros de Particulados de Emissões de Biodiesel/ João Pedro Rodrigues Ferreira. – Rio de Janeiro: UFRJ/Escola Politécnica, 2023.

XII, 90 p.: il.; 29, 7cm.

Orientador: Gustavo Rabello dos Anjos

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia Mecânica, 2023.

Referências Bibliográficas: p. 43 – 44.

1. Método dos Elementos Finitos. 2. Fluidodinâmica. 3. Filtragem. 4. Poluentes ambientais. I. Rabello dos Anjos, Gustavo. II. Universidade Federal do Rio de Janeiro, UFRJ, Curso de Engenharia Mecânica. III. Estudo de Escoamentos em Filtros de Particulados de Emissões de Biodiesel.

Dedico minha graduação ao povo brasileiro.

Agradecimentos

Agradeço a minha mãe Denise Maria, meu maior exemplo de perseverança, que fundou as bases do meu caráter e entregou suporte incalculável na construção da minha vida acadêmica e profissional. Se hoje sei o que sei, é por conta de todos os momentos em que Dona Denise zelou por mim. Agradeço a minha querida vó Janete Muniz, cujo amor e ternura me cercaram durante toda a vida. Agradeço a meu irmão Pedro Henrique, que cumpriu papel de inspiração por muito tempo em minha vida e foi decisivo na escolha pela carreira de engenheiro em minha adolescência. Agradeço a meu pai Eurico José por me incentivar a buscar a excelência em minha vida profissional e pessoal.

Agradeço a meu amigo Bacharel Gabriel Pessanha, cujo companheirismo, astúcia e paixão pelo saber me acompanharão pela vida. Agradeço a meus amigos Pedro Martins e Victor Xavier; meus maiores confidentes, que me ampararam nos momentos mais críticos e trazem leveza a minha vida. Agradeço a minha amiga Carolina Coutinho e minha amiga e companheira de pesquisa Anna Bárbara, cujos apoios e instruções tornaram meu dia e dia e graduação mais felizes. Agradeço a minha amada Carolina, cuja companhia, afeto e ouvidos sempre atentos a ouvir minhas divagações sobre engenharia trouxeram felicidade e leveza a todo esse árduo percurso.

Agradeço ao Professor Gustavo Rabello por toda a orientação e aprendizado durante os dois anos de pesquisa ao seu lado. Suas aulas, solicitude e didática ímpares marcarão para sempre minha carreira de engenheiro. Agradeço a Universidade Federal do Rio de Janeiro, cuja história se confunde com a glória e progresso do Brasil. Por fim, agradeço ao Programa de Recursos Humanos da Agência Nacional do Petróleo, Gás Natural e Biocombustíveis, PRH-ANP, por fornecer todos os recursos, palestras e encontros que tornaram este trabalho possível.

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

ESTUDO DE ESCOAMENTOS EM FILTROS DE PARTICULADOS DE EMISSÕES DE BIODIESEL

João Pedro Rodrigues Ferreira

Dezembro/2023

Orientador: Gustavo Rabello dos Anjos

Programa: Engenharia Mecânica

Este trabalho propõe uma abordagem alternativa para a análise de escoamentos gasosos em DPFs, Diesel Particulate Filters; utilizando a Formulação de Função Corrente-Vorticidade em vez da tradicional Formulação de Navier-Stokes para modelagem da dinâmica do fluido. Também é proposta a substituição da formulação algébrica via Equação de Darcy para modelagem de meios porosos por uma formulação geométrica, impondo diferentes geometrias de obstáculos ao longo da malha do modelo. Com isso, espera-se obter um maior entendimento de como a dinâmica do escoamento gasoso no interior do filtro pode contribuir na retenção de material particulado e, assim, diminuir a emissão de partículas nas combustões de biodiesel e diesel.

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

STUDY OF FLOWS IN BIODIESEL EMISSION PARTICULATE FILTERS

João Pedro Rodrigues Ferreira

December/2023

Advisor: Gustavo Rabello dos Anjos

Department: Mechanical Engineering

This work proposes an alternative approach to the gas flow analysis in DPFs, Diesel Particulate Filters; by modeling the problem through the Stream Function-Vorticity formulation instead of the traditional Navier-Stokes formulation. Along with that, it's also proposed the replacement of the algebraic formulation of porous media through Darcy's Equation by a geometrical one, imposing different geometries of obstacules along the model's mesh. By these means, it is expected to reach a better understanding about how the fluid dynamics inside a filter contribute to the capture of this particulate material and, thus, decrease particulate emissions in biodiesel and diesel combustions.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Biocombustíveis, entre eles o biodiesel, hoje figuram entre as principais alternativas à substituição dos combustíveis fósseis em matéria de estocagem e fonte energética. A presença do biodiesel entre os principais substitutos dos combustíveis fósseis no curto e médio prazo se justifica por uma série de motivos econômicos e ambientais, entre eles: a liderança e vasta experiência brasileira na produção de sua matéria-prima, soja; sua produção em escala industrial ser plenamente dominada [3]; o fato de sua produção ser baseada em Ciclo de Carbono fechado; a presença ínfima de compostos sulfurados em sua queima e reduções de mais de 50% na emissão de particulados [4].

Contudo, apesar de todas as vantagens econômicas e ambientais listadas acima, a emissão de particulados, ainda que em quantidade menores que a do diesel fóssil, persiste como problema no consumo do biodiesel. Este trabalho tem como objetivo lidar com este material particulado, o que será feito a partir de análises do dispositivo filtrante de partículas comumente usado em automóveis, o DPF (Filtro de Partículas Diesel). Avaliando a dinâmica da filtragem que ocorre no interior desses componentes, espera-se compreender melhor como a dinâmica do escoamento gasoso pode contribuir para a retenção de partículas.

Um DPF convencional, ilustrado na figura 1.1, consiste em um cilindro com seu interior repartido em canais de seção transversal quadrada onde membranas atuam como a parede desses canais. Tais canais não são vazados em ambas extremidades, sendo alguns deles vazados para a admissão do filtro e alguns vazados para a exaustão do filtro, como se pode ver na figura 1.2. Essa propriedade gera um gradiente

de pressão através dos canais, fazendo com que o escoamento flua através das membranas, onde o material particulado ficará retido. O objetivo das análises consiste em modelar especificamente a membrana atuante nestes filtros.

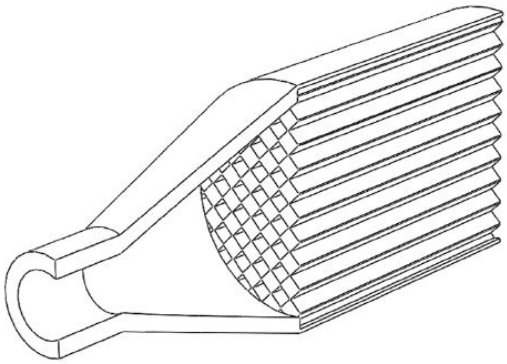

Figura 1.1: DPF convencional.

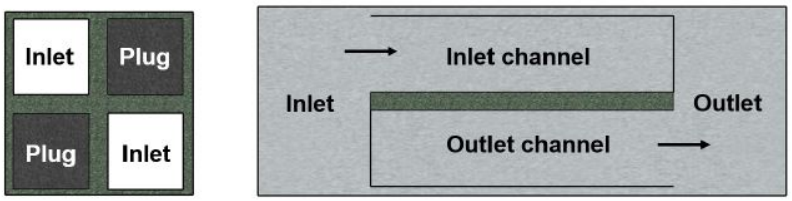

Figura 1.2: Interior de um DPF. A membrana filtrante está destacada em verde.

Para bem descrever essas análises, bem como toda a metodologia, obtenção e avaliação de resultados; este projeto final foi dividido em 5 capítulos: Introdução, Revisão Bibliográfica, Metodologia, Resultados e Conclusões.

Na seção de Metodologia, é proposta a modelagem do escoamento utilizando a Formulação de Função Corrente-Vorticidade e solucionando suas equações através do Método de Elementos Finitos. Nela, a discretização do espaço será dada pelos elementos finitos, utilizando uma malha não-estruturada em um domínio bidimensional, enquanto a discretização do tempo é realizada através de uma aproximação progressiva do operador temporal usando expansão em série de Taylor. O resultado será a solução transiente das equações de movimento do escoamento gasoso, fornecendo o valor dos campos de velocidade em qualquer região da membrana porosa.

Na seção de Revisão Bibliográfica faz-se um breve apanhado da literatura sobre emissões do Biodiesel e DPFs; bem como sobre o MEF, Método dos Elementos

Finitos, e da Formulação da Função Corrente-Vorticidade. Suas particularidades, vantagens e limitações são abordadas nesta seção.

Nas duas últimas seções, Resultados e Conclusões, a partir dos campos de velocidades obtidos são feitas induções acerca do comportamento das partículas durante sua passagem pelo filtro. Com isso, espera-se obter uma melhor compreensão da dinâmica por trás da retenção de partículas, possibilitando uma análise não só global como local do fenômeno de filtragem.

Capítulo 2

Revisão Bibliográfica

2.1 As emissões do biodiesel e o DPF, Filtro de Partículas Diesel

O biodiesel é caracterizado por uma combustão mais completa que a do diesel fóssil, com emissão de particulados menores e em menor quantidade. RODRÍGUEZ-FERNÁNDEZ *et al* [1] argumenta que a presença de compostos aromáticos no diesel fóssil favorece a formação de grandes particulados, o que diminui a superfície de contato entre partículas e moléculas de oxigênio. Já no biodiesel a ausência dos aromáticos contribui para formação de partículas menores e, portanto, mais propensas a oxidação; além da presença do oxigênio na molécula do éster, que também favorece a geração de produtos mais suscetíveis a oxidação no DPF. Essa queda na emissão de particulados e a diminuição do tamanho das partículas provenientes da combustão de biodiesel podem ser vistas na figura 2.1, onde se encontram os resultados de RODRÍGUEZ-FERNÁNDEZ *et al* [1].

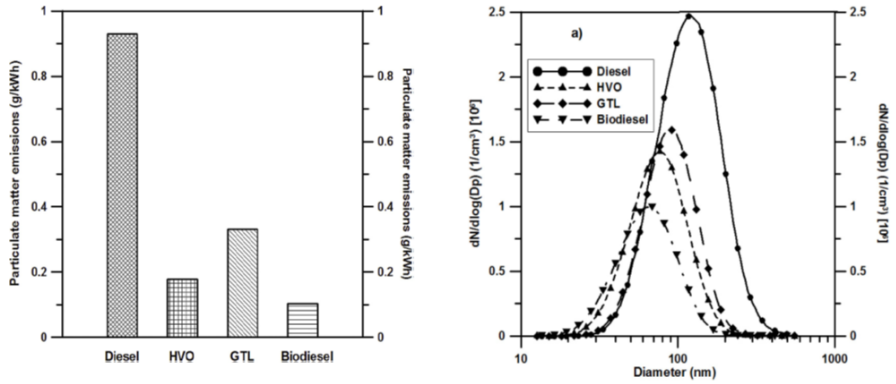

Figura 2.1: Gráficos presentes nos resultados de RODRÍGUEZ-FERNÁNDEZ *et al* [1]. O 1º ilustra a quantidade de particulados emitidos por cada combustível e o 2º ilustra a dimensão das partículas emitidas.

É nesse cenário que o DPF cumpre papel fundamental na retenção e oxidação de material particulado. Mais de 90% das partículas geradas na combustão ficam retidas ou são oxidadas em sua passagem pelo filtro, fenômeno chamado de regeneração, como aponta DU *et al* [5].

Este trabalho se propõe a recriar o interior de um DPF, com um canal de admissão, um elemento filtrante, que corresponderia a parede entre os canais, e um canal de exaustão, e simular diferentes geometrias para a porosidade do elemento filtrante a fim de entender como a movimentação do fluido em seu interior pode contribuir para a filtragem desses particulados.

2.2 Método dos Elementos Finitos

Método dos Elementos Finitos, MEF, constitui hoje uma das principais ferramentas computacionais de análise de problemas de engenharia. O método e suas bases matemáticas surgem no início do século XX como proposta de solução numérica de problemas estruturais a partir de trabalhos de John William Strutt, Walther Ritz, Boris Galerkin e outros cientistas. A partir dos anos 50, com o avanço da capacidade de processamento de computadores, o método passa a ser efetivamente utilizado na solução de análises estruturais e nas décadas seguintes se populariza nas mais diversas áreas de engenharia, da engenharia aeronáutica à bioengenharia [6].

O método parte da abordagem por equações integrais para modelagem matemática de problemas físicos. Tradicionalmente, os problemas da Mecânica do Contínuo que permeiam a engenharia podem ser abordados de duas formas: pela abordagem diferencial da Mecânica Clássica, que descreve o modelo físico através de equações diferenciais, ou pela abordagem integral, que surge com o amadurecimento do Cálculo Variacional e descreve o modelo a partir de equações integrais [6].

Nesse contexto, o Método dos Elementos Finitos consiste na discretização dessas equações integrais, o que geometricamente corresponde à repartição do domínio espacial do modelo em uma malha de quantidade finita de elementos. Esse procedimento de discretização leva ao surgimento de um sistema linear que uma vez solucionado fornece o campo da variável estudada. Nota-se, portanto, que o MEF pode ser descrito como uma aplicação numérica do Cálculo Variacional baseada na divisão de domínio em elementos menores para solução de modelos físicos [2].

Vale ainda ressaltar que este processo de geração da malha do domínio é um procedimento chave no Método dos Elementos Finitos e está diretamente ligado ao surgimento de algoritmos de geração de malha e ordenamento de elementos ao longo do século XX. Esses algoritmos possibilitaram a generalização do MEF para domínios com as geometrias mais variadas possíveis, como a da figura 2.2 abaixo, ampliando enormemente sua capacidade de atuação.

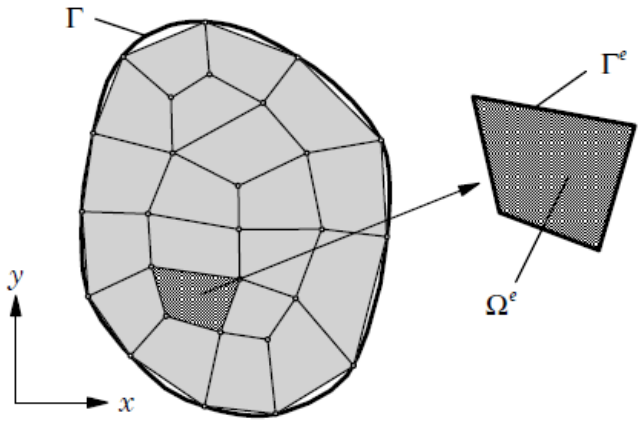

Figura 2.2: Domínio discretizado em elementos quadriláteros. Imagem retirada de GARTLING *et al* [2].

2.2.1 O Método de Galerkin

Ó Método de Galerkin é o mais tradicional para discretização da variável de interesse na aplicação do MEF. Uma vez repartido o domínio em uma malha de elementos com seus respectivos nós, o método substitui a solução real por uma interpolação dos valores da variável de interesse respectivos a cada um dos n nós da malha:

$$f(x,y) \approx \sum_{i=1}^{n} \int_{\Omega^e} N_i(x,y) \cdot f_i \, d\Omega \tag{2.1}$$

Polinômios, por atenderem aos critérios da Interpolação de Lagrange, são comumente usados como as funções de interpolação da discretização. O grau desses polinômios ainda determina a ordem do elemento utilizado: elementos que usam polinômios de $2^{\underline{o}}$ grau, são elementos de $2^{\underline{a}}$ ordem, elementos que usam polinômios de $1^{\underline{o}}$ grau, são elementos de $1^{\underline{a}}$ ordem.

Abaixo, para fins de exemplo, encontram-se as funções de interpolação de um elemento triangular linear genérico reunidas em um vetor:

$$N_i = \begin{bmatrix} N_1^i \\ N_2^i \\ N_3^i \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} X_2Y_3 - X_3Y_2 + (Y_2 - Y_3)x + (X_3 - X_2)y \\ X_3Y_1 - X_1Y_3 + (Y_3 - Y_1)x + (X_1 - X_3)y \\ X_1Y_2 - X_2Y_1 + (Y_1 - Y_2)x + (X_2 - X_1)y \end{bmatrix}_{3\times 1} \tag{2.2}$$

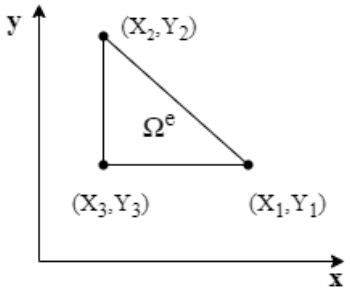

Figura 2.3: Elemento triangular linear Ω^e com área A em eixo de coordenadas local.

Ao olhar somente para um elemento do domínio como o ilustrado na figura 2.3, a substituição da solução analítica por essa interpolação resultará em um sistema linear $A^e\vec{x}$. Neste sistema, a matriz A^e, oriunda de produtos tensoriais dos vetores de interpolação, será composta por funções de interpolação e $\vec{x}$ será a solução numérica a ser descoberta.

Por fim, é necessário realizar a montagem da matriz global do problema. Isto é feito com o auxílio de uma matriz de conectividade, que relaciona cada elemento da malha com os nós que a ele pertencem. Uma vez montada a matriz global, soluciona-se o sistema linear, obtendo, assim, a solução numérica em cada um dos nós da malha.

2.3 Modelagem do escoamento: Formulação Função Corrente-Vorticidade

Para modelar o escoamento no interior do filtro de partículas, foi adotado a formulação de Função Corrente-Vorticidade, uma alternativa a modelagem tradicional de Navier-Stokes. Em The Finite Element Method in Heat Transfer and Fluid Dynamics (GARTLING *et al* [2]), a equação que sustenta esta formulação, a Equação de Transporte de Vorticidade, é descrita e suas vantagens para problemas bidimensionais são apresentadas. Ela é obtida com a aplicação do operador rotacional na equação de momento do fluido e é caracterizada por estar em função da vorticidade e velocidade do escoamento, sem a presença do termo de pressão, como se pode ver abaixo:

$$\frac{\partial \omega_z}{\partial t} + \vec{v} \cdot \nabla \omega_z = \nu \nabla^2 \omega_z \tag{2.3}$$

Onde ν é viscosidade cinemática e v a velocidade do escoamento. As outras duas equações que completam a Formulação $\psi - \omega_z$ são a Equação de Corrente-Vorticidade e a própria definição da Função Corrente:

$$\nabla^2 \psi = -\omega_z \tag{2.4}$$

$$\vec{v} = \left(\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x} \right) \tag{2.5}$$

A principal vantagem desta formulação é o desacoplamento de pressão e velocidade que o transporte de vorticidade apresenta, contornando as dificuldades da obtenção do campo de pressão e viabilizando a utilização de elementos lineares na implementação de Elementos Finitos. A possibilidade de usar elementos lineares facilita muito a modelagem computacional do escoamento.

Por outro lado, algumas dificuldades acompanham a Formulação $\psi - \omega_z$ e cuidados são necessários. A formulação não lida bem com altos números de Reynolds, seu aumento ocasiona o surgimento de erros de dispersão, originados na parcela advectiva da equação. Há maneiras de contornar este problema, como a partir da discretização "upwind"do domínio. Neste trabalho, apenas optou-se por permanecer em baixos números de Reynolds. O fato de se estar trabalhando com escoamento gasosos também contribuiu para essa escolha, baixos valores de Reynolds permitem desconsiderar a compressibilidade dos gases.

A faixa de valores para a viscosidade do fluido também é restrita. Estes elementos são mais limitados para lidar com operadores de 2ª ordem, como o Laplaciano, e a viscosidade multiplica justamente a parcela deste operador na formulação; tornando necessário, portanto, trabalhar com baixas viscosidades para atenuar esta parcela.

Outra limitação é a definição das condições de contorno do problema a ser atacado se torna mais difícil, uma vez que a formulação é dada em termos de função corrente, vorticidade e velocidade e não pressão e velocidade. Para lidar com esse problema, CARNEVALE *et al* [7] propõe a definição das condições de contorno pela função corrente, deixando a vorticidade livre de imposições diretas. Este trabalho seguirá por esse mesmo caminho.

2.4 Modelagem da porosidade

É comum que a porosidade de um meio seja modelada algebricamente, podendo ser através da tradicional Equação de Darcy ou equações mais modernas, como a Equação de Brinkman-Forchheimer:

$$\nabla p = -\frac{\mu}{K}\vec{v} \tag{2.6}$$

$$\nabla p = -\frac{\mu}{K}\vec{v} - \frac{\rho C_f}{K}|\vec{v}|\vec{v} + \mu\nabla^2\vec{v} \tag{2.7}$$

Onde μ é a viscosidade dinâmica, K é o coeficiente de permeabilidade e C_f é o coeficiente de resistência inercial. Em ambas equações, a porosidade é expressa algebricamente em termos de perda de carga e mensurada por K. A equação Brinkman-Forchheimer ainda apresenta parcelas de difusão e convecção para capturar a não

linearidade da perda de carga durante a passagem do fluido pelo poros.

Essas abordagens algébricas apresentam a grande vantagem de contornar as dificuldades de reprodução da geometria complexa de um meio poroso. Nelas, costuma-se avaliar a capacidade de filtragem do meio a partir da queda de velocidade após a passagem do escoamento pelos poros.

CIMOLIN *et al* [8] em seu artigo acoplam a Equação de Darcy e a Equação de Brinkman-Forchheimer à Equação de Navier-Stokes e comparam os campos de velocidade das dois modelos para mesmos cenários de escoamento. SPESANI [9] por sua vez realiza as devidas transformações na Equação de Navier-Stokes com porosidade de Brinkman-Forchheimer e chega à Equação Adimensionalizada de Transporte de Vorticidade para Meios Porosos. A partir dessa formulação, ele avalia a eficiência de elementos filtrantes de DPFs de diferentes comprimentos e espessuras por meio da queda de velocidade do escoamento após passagem pelo meio poroso.

$$\frac{\partial \omega_z}{\partial t} + \vec{v} \cdot \nabla \omega_z \frac{1}{Re \cdot Da}(w_z + F_o|\vec{v}|w_z) = \frac{1}{Re}\nabla^2 \omega_z \tag{2.8}$$

Já este trabalho tratará o problema da porosidade de uma outra forma: os poros serão modelados geometricamente ao longo dos domínios. Esta abordagem, mesmo que usando domínios bem mais simples que meios porosos reais, viabiliza a obtenção de resultados qualitativos da movimentação do escoamento entre os poros, movimentações que nas formulações algébricas são todas condensadas em uma mera perda de carga. Na seção seguinte, essas geometrias porosas serão apresentadas.

Capítulo 3

Metodologia

Neste capítulo serão descritas a modelagem matemática do escoamento, a solução do modelo por Método dos Elementos Finitos, os cenários e hipóteses adotadas e sua implementação computacional em linguagem de programação Python.

Nos primeiros momentos da pesquisa esperava-se realizar a simulação de um escoamento multifásico, com presença de partículas ao longo do escoamento gasoso. Contudo, no decorrer dos trabalhos, a implementação computacional das partículas se tornou inviável, sobretudo por questões de tempo e dificuldade da elaboração de um algoritmo que fosse capaz de rastrear a posição das partículas a cada incremento no tempo; sendo, portanto, uma tarefa para trabalhos futuros.

Neste contexto, este trabalho se voltou para a realização de uma análise qualitativa da dinâmica das partículas, utilizando os campos de velocidades do fluido para inferir a movimentação das partículas no interior do filtro.

Além disso, como dito acima, a proposta deste trabalho é a modelagem de um escoamento gasoso, portanto, trabalhar em baixos números de Reynolds é fundamental para desprezar efeitos compressíveis. Todas as análises realizadas neste trabalho permaneceram abaixo de $Re = 50$.

Quanto à modelagem da porosidade do filtro, como já abordado na Revisão Bibliográfica, optou-se por uma abordagem geométrica em vez de uma abordagem algébrica como a da Lei de Darcy. Neste modelo, a porosidade será recriada com a imposição de vários obstáculos geométricos ao longo dos domínios bidimensionais, uma vez que há interesse na influência da geometria do filtro na dinâmica do gás a ser filtrado.

Agora serão apresentadas as geometrias de filtro selecionadas. Em seguida, a Formulação $\psi - \omega_z$ será detalhada e por fim a implementação computacional é abordada. A apresentação das geometrias antes da Formulação $\psi - \omega_z$ facilitará o entendimento da definição das condições de contorno do problema.

3.1 Definição das geometrias de domínio e propriedades do escoamento

3.1.1 Geometria para verificação do algoritmo

A primeira geometria a ser avaliada pelo Algoritmo de Elementos Finitos deste trabalho será um retângulo livre de obstáculos, simulando um escoamento incompressível entre placas paralelas infinitas sem deslizamento nos contornos a fim de verificar a validade do modelo construído e sua implementação computacional. O resultado da solução numérica será comparado com a solução permanente do perfil de velocidade analítico deste problema, que é dado pela seguinte equação:

$$u_p(y) = \frac{\partial P}{\partial x}\frac{1}{2\mu}y^2 - \frac{\partial P}{\partial x}\frac{L_c}{2\mu}y \ , \qquad \frac{\partial P}{\partial x} = -\frac{12\mu Q}{L_z L_c^3} \tag{3.1}$$

L_c é o comprimento característico do escoamento (largura do canal), Q é a vazão volumétrica, μ é viscosidade dinâmica, L_z é uma profundidade em Z arbitrada para o canal.

Na figura 3.1 se encontra a geometria gerada para o problema. Seu comprimento L respeita o comprimento de entrada do escoamento, que é dado por:

$$L_e \approx 0.06Re \cdot L_c \tag{3.2}$$

O comprimento L atribuído ao domínio acima é maior que L_e, assim é possível avaliar o escoamento em seu estado permanente e completamente desenvolvido. Na seção de Resultados essas grandezas terão seus valores devidamente apresentados.

3.1.2 Geometrias locais e global

Validado o algoritmo, duas categorias de geometria serão analisadas, uma que retratará o percurso inteiro do escoamento pelo interior de um filtro, adentrando

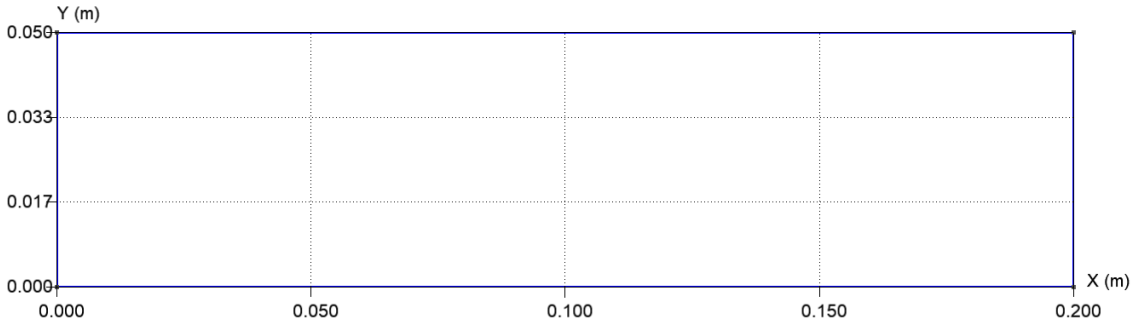

Figura 3.1: Geometria para verificação. Um retângulo 0.05mX0.20m representando um volume de controle de um escoamento entre placas paralelas.

pelo canal de admissão, passando pelo elemento filtrante e saindo pelo canal de exaustão; e uma outra retratando o percurso do fluido apenas pelo elemento filtrante. A primeira geometria, ilustrada na figura 3.2, será denominada global, enquanto a segunda local. Dentro da categoria de geometria local, três perfis de porosidade diferentes foram gerados para análise. Os três estão ilustrados nas figuras 3.3, 3.4 e 3.5:

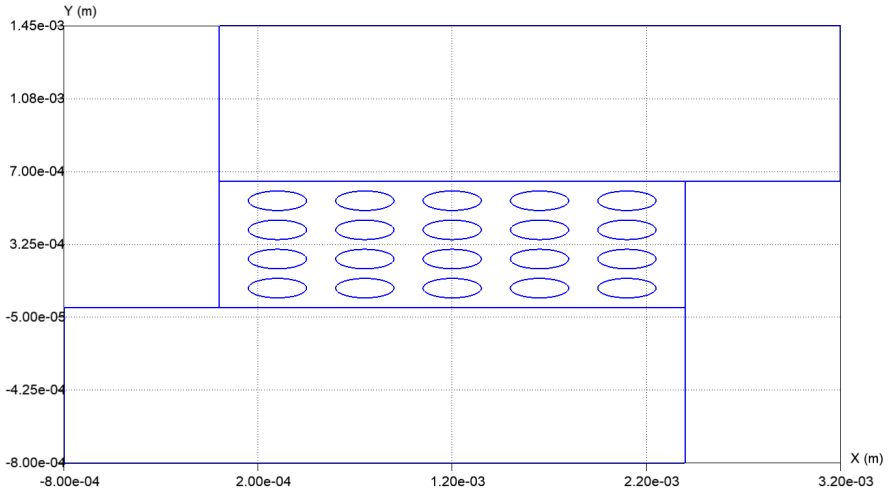

Figura 3.2: Geometria global.

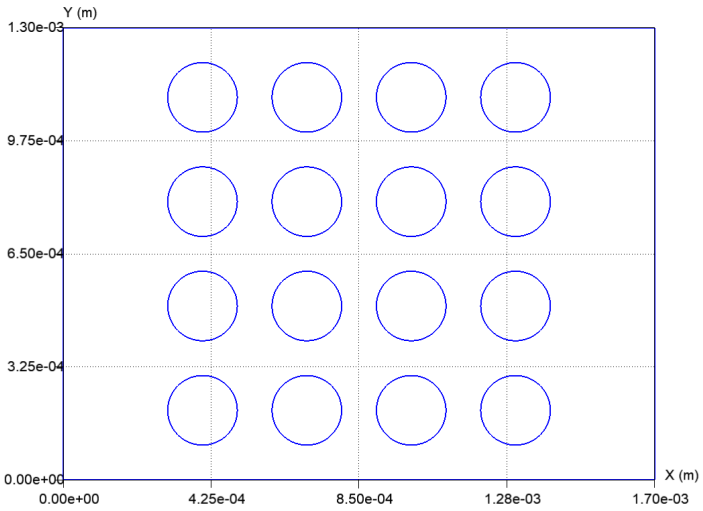

Figura 3.3: Geometria local de obstáculos circulares.

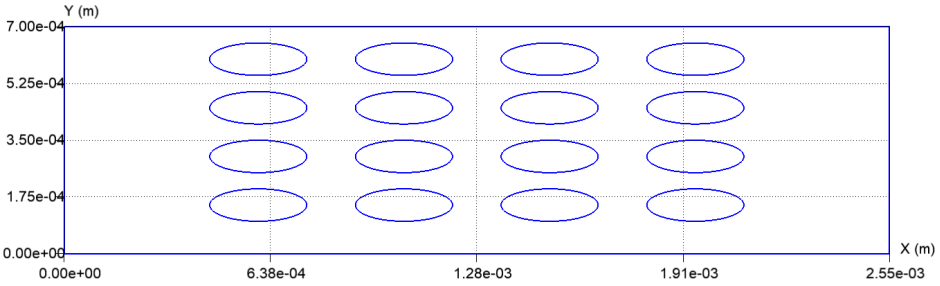

Figura 3.4: Geometria local de obstáculos delgados horizontais.

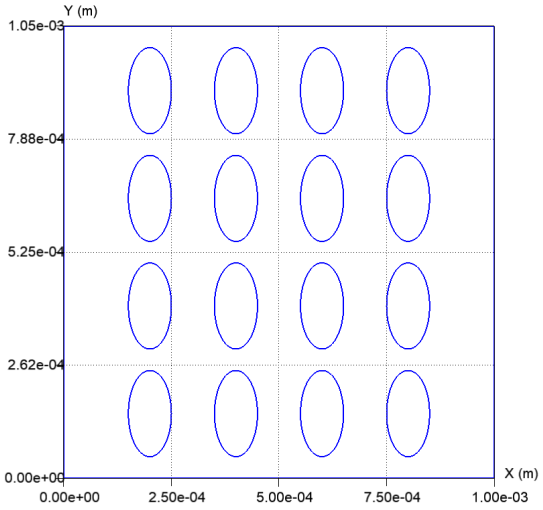

Figura 3.5: Geometria local de obstáculos delgados verticais.

Prezou-se pela fidelidade às dimensões do problema real. Os domínios acima foram gerados tendo como base a espessura da parede de DPFs convencionais. Ao consultar a literatura, foram encontrados experimentos de DPFs com espessura de parede em torno de 0.5mm (YANTING *et al* [5], WANG [10]). Por conta disso, todas as 4 geometrias acima foram geradas apresentando elemento filtrante com espessura entre 0.5mm a 1mm.

Partindo deste comprimento limitador, foram definidas a quantidade de obstáculos que se teria em cada geometria. Como a espessura do elemento filtrante é restrita (até 1mm), a única maneira de aumentar a quantidade de obstáculos na vertical seria diminuindo suas áreas, o que computacionalmente deixaria o algoritmo muito caro. Por essa razão, optou-se por gerar não mais que 4 linhas de obstáculos para cada uma das geometrias.

Quanto ao número de colunas de obstáculos, foi estabelecido arbitrariamente número de 4 colunas nas análises locais. Pela simetria das geometrias selecionadas, não haveria diferença na dinâmica do fluido entre colunas ao realizar a análise em um domínio com mais ou menos colunas de obstáculos.

Definidas a espessura do filtro e a quantidade de obstáculos em seu interior, resta apenas definir a distância entre os obstáculos e seus diâmetros. Na seção de Resultados estão listadas as propriedades dimensionais de cada geometria.

3.2 A Formulação Corrente-Vorticidade

Estabelecida as geometrias a serem avaliadas neste trabalho, pode-se agora abordar o modelo físico-matemático a ser utilizado.

As 3 equações que compõem o modelo de $\psi - \omega_z$ foram brevemente apresentadas na revisão bibliográfica. Esta formulação consiste em um sistema de 3 equações que envolvem a vorticidade, a função corrente e a velocidade do escoamento em análise.

$$\frac{\partial \omega_z}{\partial t} + \vec{v} \cdot \nabla \omega_z = \nu \nabla^2 \omega_z \tag{2.3}$$

$$\nabla^2 \psi = -\omega_z \tag{2.4}$$

$$\vec{v} = \left(\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x} \right) \tag{2.5}$$

Neste sistema, os contornos do domínio e obstáculos apresentam velocidade igual a zero (condição de não deslizamento) e os valores de função corrente ψ são definidos de maneira a se ter um crescimento aproximadamente linear e bem comportado ao longo do domínio. Isto é feito definindo um mesmo valor de ψ para cada coluna de obstáculos, com ψ crescendo linearmente a cada coluna da direita à esquerda, conforme mostram as figuras 3.6 e 3.7. Esse procedimento garante um escoamento bem comportado através dos obstáculos, condição necessária para a Formulação Função Corrente-Vorticidade ser eficaz. Como condição inicial do problema, tem-se um campo de velocidades inicial.

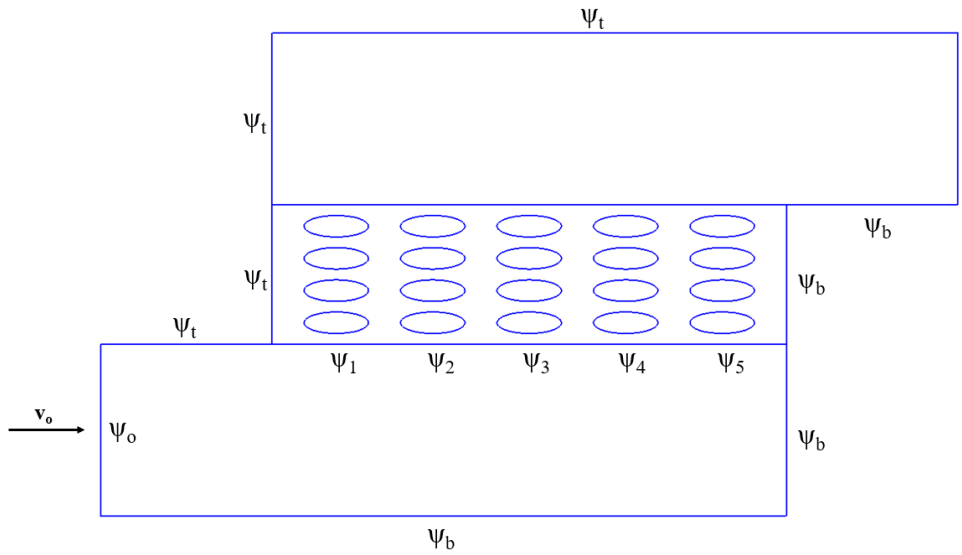

Figura 3.6: Função corrente como condição de contorno da geometria global.

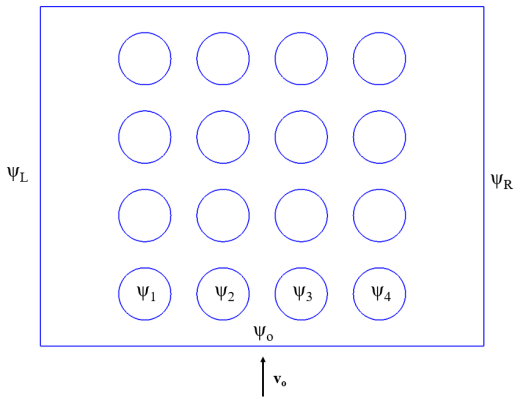

Figura 3.7: Função corrente como condição de contorno de uma das geometrias locais.

Estabelecidas as condições de contorno e inicial, a resolução desse sistema ocorre conforme o fluxograma da figura 3.8: partindo do campo de velocidades inicial, usa-se a equação de transporte de vorticidade (2.3) para obtenção do campo de vorticidade do escoamento. Conhecendo este campo, pode-se obter a função corrente associada ao escoamento por meio da Equação de Corrente-Vorticidade (2.4). Por fim, conhecendo a função corrente, chega-se ao campo de velocidades do fluido (2.5).

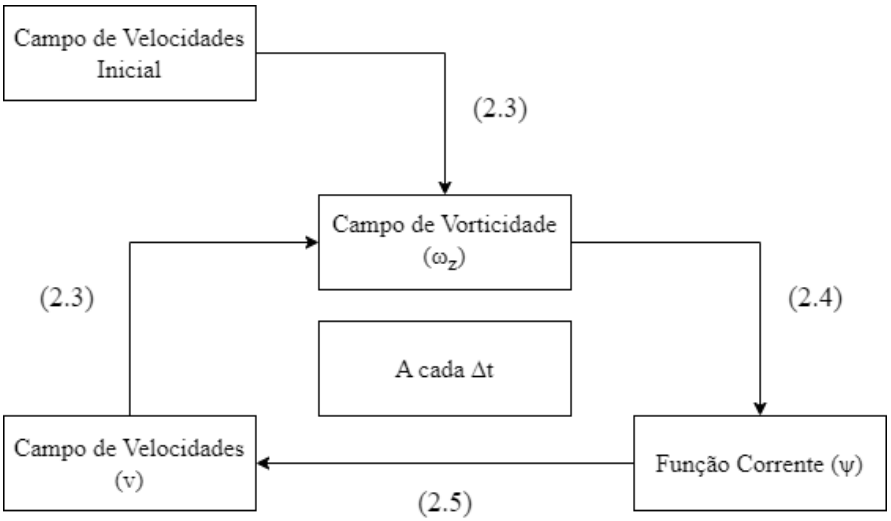

Figura 3.8: Algoritmo de resolução da Formulação de Função Corrente-Vorticidade.

A seguir, serão obtidas, através de ferramentas do Cálculo Variacional, as formas fracas das 3 equações em análise. Uma vez reescritas em sua forma fraca, é realizada sua discretização pelo Método de Galerkin para implementação do Método de Elementos Finitos (MEF):

3.2.1 Formulação MEF para Equação de Transporte de Vorticidade

$$\frac{\partial \omega_z}{\partial t} + \vec{v} \cdot \nabla \omega_z = \nu \nabla^2 \omega_z \tag{2.3}$$

Forma integral:

$$\int_\Omega \left[\frac{\partial \omega_z}{\partial t} + \vec{v} \cdot \nabla \omega_z - \nu \nabla^2 \omega_z \right] \cdot \omega \; d\Omega = 0 \tag{3.3}$$

Onde Ω corresponde ao domínio em análise e ω à função teste, de continuidade

C^1. Em seguida, é feita a aplicação do Teorema de Green na parcela do divergente da vorticidade; obtendo, assim, sua forma fraca:

$$\int_\Omega \omega \frac{\partial \omega_z}{\partial t}\, d\Omega + \int_\Omega \omega \cdot \vec{v} \cdot \nabla \omega_z \, d\Omega - \int_\Gamma \omega \cdot \nu \nabla \omega_z \, d\Gamma + \int_\Omega \nabla \omega \cdot \nu \nabla \omega_z \, d\Omega = 0 \quad (3.4)$$

Onde Γ é a fronteira do domínio em análise.

Agora é necessário discretizar a equação. Como comentado na revisão bibliográfica, uma das vantagens substanciais da Formulação de Corrente-Vorticidade é a possibilidade de usar elementos lineares para solução de problemas de dinâmica dos fluidos. Nesse sentido, este trabalho utilizará elementos triangulares de 1ª ordem para resolução das equações do modelo. Definida a ordem e tipo do elemento, ficam também definidos os vetores de interpolação linear N_i, N_j e de seus gradientes B_i, B_j do Método de Galerkin, que possibilitarão a discretização:

$$N_i = \frac{1}{2A} \begin{bmatrix} X_2Y_3 - X_3Y_2 + (Y_2 - Y_3)x + (X_3 - X_2)y \\ X_3Y_1 - X_1Y_3 + (Y_3 - Y_1)x + (X_1 - X_3)y \\ X_1Y_2 - X_2Y_1 + (Y_1 - Y_2)x + (X_2 - X_1)y \end{bmatrix}_{3\times 1} \quad , \quad N_j = N_i^T \quad (3.5)$$

$$B_i = \frac{1}{2A} \begin{bmatrix} Y_2 - Y_3 & X_3 - X_2 & 0 \\ Y_3 - Y_1 & X_1 - X_3 & 0 \\ Y_1 - Y_2 & X_2 - X_1 & 0 \end{bmatrix}_{3\times 3} \quad , \quad B_j = B_i^T \quad (3.6)$$

Onde A é a área do elemento do domínio em análise e (X_1, Y_1), (X_2, Y_2), (X_3, Y_3) sãos os nós do elemento triangular.

Abaixo se encontra a Equação de Transporte de Vorticidade em sua forma fraca e discretizada em elementos:

$$\begin{aligned} &\sum_i^\infty \sum_j^\infty \int_\Omega N_j \omega_j \cdot \frac{\partial (N_i \omega_{z_i})}{\partial t}\, d\Omega + \sum_i^\infty \sum_j^\infty \int_\Omega N_j \omega_j \cdot \vec{v} \cdot B_i \omega_{z_i}\, d\Omega \\ &- \sum_i^\infty \sum_j^\infty \int_\Gamma N_j \omega_j \cdot (\nu B_i \omega_{z_i})\, d\Gamma + \sum_i^\infty \sum_j^\infty \int_\Omega B_j \omega_j \cdot (\nu B_i \omega_{z_i})\, d\Omega = 0 \end{aligned} \quad (3.7)$$

Dividindo a equação pela função peso ω e retirando ω_{z_i} das integrais:

$$\begin{aligned}\sum_{i}^{\infty}\sum_{j}^{\infty}\int_{\Omega} N_j N_i \, d\Omega \cdot \frac{\partial \omega_{z_i}}{\partial t} + \sum_{i}^{\infty}\sum_{j}^{\infty}\int_{\Omega} \vec{v} \cdot N_j B_i \, d\Omega \cdot \omega_{z_i} \\ - \sum_{i}^{\infty}\sum_{j}^{\infty}\int_{\Gamma} N_j \cdot (\nu B_i) \, d\Gamma \cdot \omega_{z_i} + \sum_{i}^{\infty}\sum_{j}^{\infty}\int_{\Omega} B_j \cdot (\nu B_i) \, d\Omega \cdot \omega_{z_i} = 0\end{aligned} \tag{3.8}$$

Agora, os somatórios duplos serão substituídos por somatórios de elemento e algumas hipóteses serão assumidas:

- velocidade de escoamento $\vec{v}$ constante no elemento arbitrário em análise;
- viscosidade ν constante;
- condições de contorno de Dirichlet, o que torna a função teste ω igual a zero em qualquer ponto pertencente à fronteira Γ, zerando, assim, integral na fronteira Γ.

Assim, a equação assumirá a seguinte forma:

$$\sum_{e}^{\infty}\int_{\Omega} N_j N_i \, d\Omega \cdot \frac{\partial \omega_{z_i}}{\partial t} + \sum_{e}^{\infty}\vec{v}\int_{\Omega} N_j B_i \, d\Omega \cdot \omega_{z_i} + \sum_{e}^{\infty}\nu\int_{\Omega} B_j B_i \, d\Omega \cdot \omega_{z_i} = 0 \tag{3.9}$$

As integrais dos produtos tensoriais das vetores de interpolação correspondem a matrizes elementares de massa, gradiente e viscosidade:

$$M^e = \int_{\Omega} N_j N_i \, d\Omega \;\; , \quad G^e = \int_{\Omega} N_j B_i \, d\Omega \;\; , \quad K^e = \int_{\Omega} B_j B_i \, d\Omega \tag{3.10}$$

Substituindo o somatório de matrizes elementares por suas respectivas matrizes globais:

$$M\frac{\partial \omega_{z_i}}{\partial t} + \vec{v} \cdot G\omega_{z_i} + \nu K \omega_{z_i} = 0 \tag{3.11}$$

Finalmente, é efetuada a discretização avançada e implícita no tempo:

$$M\frac{\omega_{z_i}^{n+1} - \omega_{z_i}^{n}}{\Delta t} + \vec{v} \cdot G\omega_{z_i}^{n+1} + \nu K \omega_{z_i}^{n+1} = 0 \tag{3.12}$$

$$\left(\frac{M}{\Delta t} + \nu K + \vec{v} \cdot G\right)\omega_{z_i}^{n+1} = \frac{M}{\Delta t}\omega_{z_i}^{n} \tag{3.13}$$

O termo de advecção ainda pode ser expandido para identificação das matrizes do gradiente nas direções x e y:

$$\vec{v} \cdot G = (v_x I) \cdot G_x + (v_y I) \cdot G_y \tag{3.14}$$

$$\left(\frac{M}{\Delta t} + \nu K + (v_x I) \cdot G_x + (v_y I) \cdot G_y \right) \omega_{z_i}^{n+1} = \frac{M}{\Delta t} \omega_{z_i}^{n} \tag{3.15}$$

3.2.2 Formulação MEF para Equação de Corrente-Vorticidade

Agora, será feita a discretização da Equação de Corrente-Vorticidade levando em consideração as mesmas hipóteses utilizadas na discretização do Transporte de Vorticidade:

$$\nabla^2 \psi = -\omega_z \tag{2.4}$$

$$\int_\Omega \omega \cdot [\nabla \cdot (\nabla \psi) + \omega_z] \ d\Omega = 0 \tag{3.16}$$

Aplicando Teorema de Green no divergente de ψ:

$$\oint_\Gamma \omega \nabla \psi \cdot \vec{n} \ d\Gamma - \int_\Omega \nabla \omega \cdot \nabla \psi \ d\Omega + \int_\Omega \omega \cdot \omega_z \ d\Omega = 0 \tag{3.17}$$

$$- \oint_\Gamma \omega \nabla \psi \ d\Gamma - \int_\Omega \nabla \omega \cdot \nabla \psi \ d\Omega + \int_\Omega \omega \cdot \omega_z \ d\Omega = 0 \tag{3.18}$$

Dividindo a equação por ω e discretizando novamente as funções por meio das matrizes elementares de interpolação:

$$\sum_i^\infty \sum_j^\infty \left[\int_\Omega B_j B_i \ d\Omega \cdot \psi_i + \int_\Gamma N_j B_i \ d\Gamma \cdot \psi_i \right] - \sum_k^\infty \sum_j^\infty \int_\Omega N_j N_k \ d\Omega \cdot \omega_{z_i} = 0 \tag{3.19}$$

Assumindo condições de contorno de Dirichlet, a integral ψ em Γ é zerada. Por fim, os somatórios das matrizes elementares, representadas pelas integrais das funções de interpolação, são substituídos pelas matrizes globais K e M.

$$K \psi_i = M \omega_{z_i} \tag{3.20}$$

3.2.3 Formulação MEF para Função Corrente

Resta a discretização da definição da função corrente:

$$v_x = \frac{\partial \psi}{\partial y} \ , \ v_y = -\frac{\partial \psi}{\partial x} \tag{2.5}$$

Velocidade horizontal, v_x :

$$\int_\Omega \omega \cdot \left[v_x - \frac{\partial \psi}{\partial y} \right] \ d\Omega = 0 \tag{3.21}$$

$$\sum_i^\infty \sum_j^\infty \int_\Omega N_j N_i \ \ d\Omega \cdot v_{x_i} = \sum_i^\infty \sum_j^\infty \int_\Omega N_j B_i^y \ \ d\Omega \cdot \psi_i \ \ , \quad B^y = \frac{\partial N}{\partial y} \tag{3.22}$$

Velocidade vertical, v_y :

$$\int_\Omega \omega \cdot \left[v_y + \frac{\partial \psi}{\partial x} \right] \ d\Omega = 0 \tag{3.23}$$

$$\sum_i^\infty \sum_j^\infty \int_\Omega N_j N_i \ \ d\Omega \cdot v_{y_i} = -\sum_i^\infty \sum_j^\infty \int_\Omega N_j B_i^x \ \ d\Omega \cdot \psi_i \ \ , \quad B^x = \frac{\partial N}{\partial x} \tag{3.24}$$

$$K v_x = G_y \psi_i \ \ , \quad K v_y = -G_x \psi_i \tag{3.25}$$

Finalmente, o sistema de equações da Formulação Corrente-Vorticidade em sua forma fraca discretizada, pronto para ser implementado computacionalmente:

$$\left(\frac{M}{\Delta t} + \nu K + (v_x I) \cdot G_x + (v_y I) \cdot G_y \right) \omega_{z_i}^{n+1} = \frac{M}{\Delta t} \omega_{z_i}^n \tag{3.15}$$

$$K \psi_i = M \omega_{z_i} \tag{3.20}$$

$$K v_x = G_y \psi_i \ , \ \ K v_y = -G_x \psi_i \tag{3.25}$$

3.3 Implementação computacional

O Método dos Elementos Finitos para este trabalho foi inteiramente implementado em linguagem de programação Python v. 3.10.10 e o editor de código utilizado para programação foi o Visual Studio Code v. 1.82.2. Somente geometria do domínio, malha e visualização dos resultados que foram realizados em dois softwares de código aberto a parte.

Para geração da geometria dos canais e de suas respectivas malhas foi usado o software Gmsh 4.11.0. As malhas, ilustradas nas figuras 3.9, 3.10, 3.12, 3.11 e 3.13, são triangulares não estruturadas e com refinamento localizado, todas geradas a partir do algoritmo de Frontal-Delaunay.

O refinamento em áreas específicas foi uma importante ferramenta para diminuição do custo computacional das simulações, possibilitando refinar a malha somente nas regiões entre os obstáculos, onde se espera não-linearidades e comportamento altamente dinâmico do escoamento. Somente o caso de validação não teve sua malha refinada localmente (figura 3.9), dada a simplicidade da geometria de seu domínio.

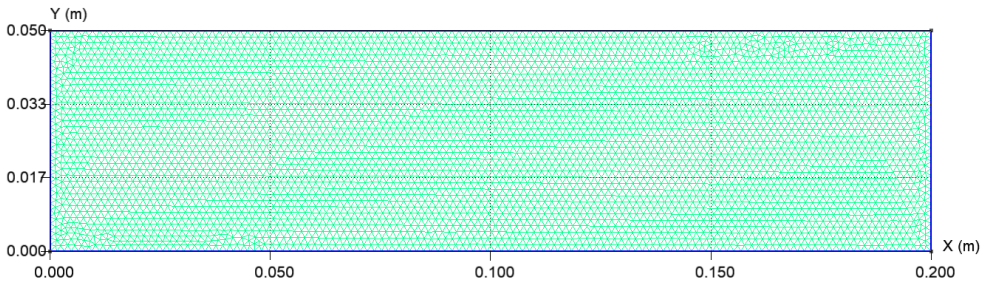

Figura 3.9: Malha para verificação.

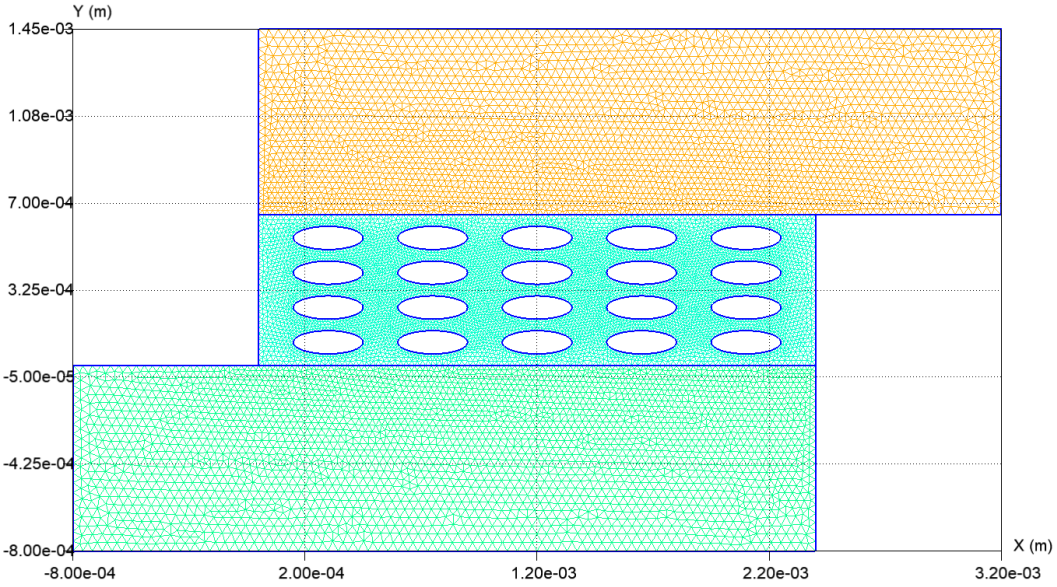

Figura 3.10: Malha global.

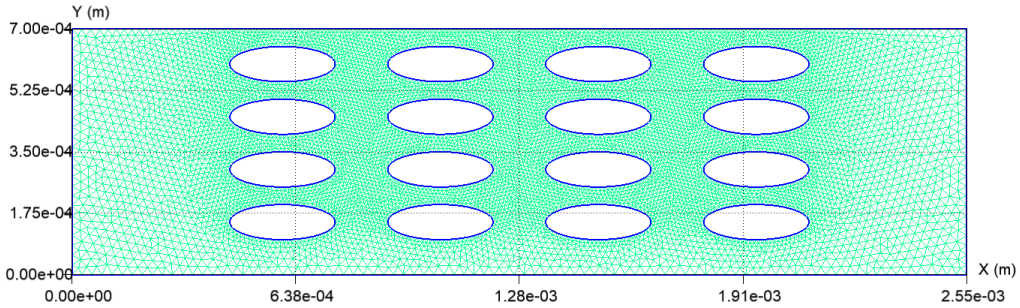

Figura 3.11: Malha de obstáculos delgados horizontais.

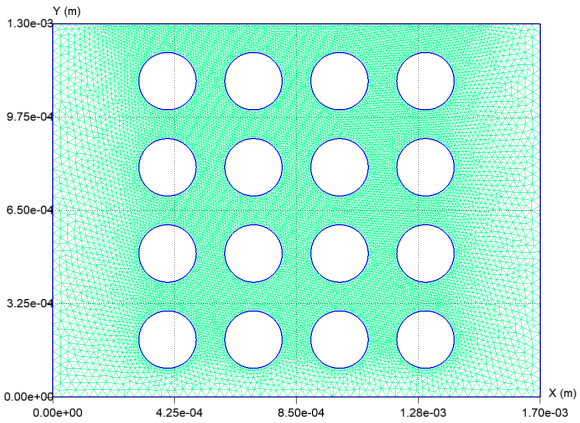

Figura 3.12: Malha de obstáculos circulares.

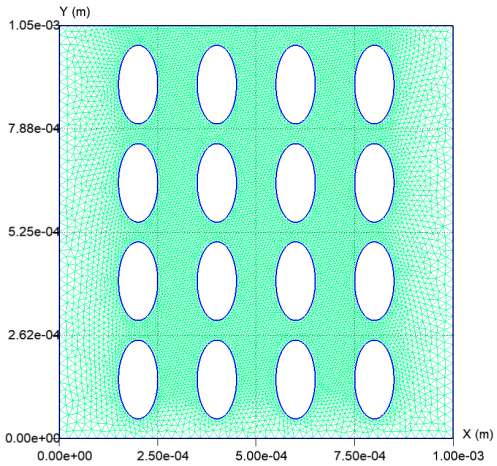

Figura 3.13: Malha de obstáculos delgados verticais.

Criadas as malhas, elas são importadas para o ambiente de trabalho do Python, onde são definidas as características do escoamento e as condições de contorno e inicial do problema. Feito isso, a análise é posta a rodar e os resultados obtidos no algoritmo do MEF, cujo código se encontra no Apêndice A deste trabalho, são importados para o ParaView v. 5.11.1, software onde imagens e animações são gerados para avaliação do problema.

Capítulo 4

Resultados

Todas as análises a seguir foram realizadas em um computador pessoal Lenovo Ideapad S145 Intel® Core™ i7-8565U com 8Gb de memória RAM e 250Gb de memória SSD. As análises mais pesadas levaram cerca de 8 horas para terminar.

As análises a serem apresentadas a seguir foram realizadas com o mesmo fluido e com a mesma velocidade de entrada v_o, exceto pelo caso de verificação, que será visto a seguir. Na tabela 4.1 abaixo se encontram os valores selecionados para viscosidade cinemática do fluido e velocidade inicial:

Grandezas	Valores	Unidades
Velocidade inicial (v_o^y)	25	mm/s
viscosidade cinemática (ν)	$1.0 \cdot 10^{-7}$	m^2/s

Tabela 4.1: Velocidade e viscosidade cinemática selecionadas para o escoamento nas simulações dos filtros.

Inicialmente, se esperava simular o escoamento de um fluido de viscosidade cinemática ν da ordem de $10^{-5}\, m^2/s$, equivalente a do gás carbônico, principal produto da combustão do biodiesel. Contudo, durante o processo percebeu-se que a difusividade numérica impedia a convergência das simulações. O fato dos fenômenos fluidodinâmicos serem da ordem de $10^{-4}\, m$ impediu a utilização de uma viscosidade cinemática da ordem de $10^{-5}\, m^2/s$. Com o objetivo de diminuir a difusividade numérica, tomou-se $\nu = 1.0 \cdot 10^{-7}\, m^2/s$, o que, para mera comparação, equivale a viscosidade do metano.

O primeiro grande desafio enfrentado durante a execução das análises foi a obtenção do critério para convergência temporal do algoritmo. Após a realização de várias simulações com diferentes velocidades de escoamento e incrementos temporais, notou-se que, para o método ser capaz de captar as não linearidades e a dinâmica do fluido, era necessário que o deslocamento do escoamento a cada iteração no tempo fosse pelo menos 4 vezes menor que seu comprimento característico. Isso garantia que a análise capturasse adequadamente a movimentação do fluido e erros não se propagassem no avanço temporal. Matematicamente, esse critério é descrito pela equação 4.1:

$$\Delta t \cdot v_o \leq \frac{L_c}{4} \tag{4.1}$$

Onde Δt é o incremento de tempo, v_o é a velocidade inicial do escoamento e L_c é o comprimento característico do escoamento. Este critério foi um limitador importante na execução das análises. Dado que os domínios utilizados neste trabalho são da ordem de 10^{-4}m, foi necessário optar por incrementos no tempo minúsculos, como 10^{-4}m, ou velocidades de escoamento irreais, como 10^{-4}m/s. Decidiu-se, ao fim, penalizar o incremento temporal em detrimento da velocidade; assim, os modelos não perderiam sua representatividade. Foram utilizados incrementos temporais da ordem de 10^{-3}s e velocidades iniciais da ordem de centímetros 10^{-2}m/s.

4.1 Verificação do algoritmo

Nesta 1ª análise não houve compromisso com a reprodução fiel das dimensões de um DPF nem com as propriedades do gás proveniente da combustão. Foram arbitradas dimensões e propriedades do fluido somente para comparação dos resultados do algoritmo com a solução analítica do escoamento entre placas paralelas. Os valores se encontram na tabela 4.2 abaixo:

	Escoamento	
Grandezas	Valores	Unidades
Comprimento total (L)	0.20	m
Comprimento de entrada (L_e)	0.15	m
Comprimento característico (L_c)	0.05	m
Velocidade inicial (v_o^x)	0.01	m/s
Viscosidade cinemática (ν)	1.00e-5	m/s
N° de Reynolds (Re)	50	-
Tempo de duração (T)	20.0	s
	Algoritmo	
Grandezas	Valores	Unidades
Elementos	9136	-
Incremento temporal (Δt)	0.2	s

Tabela 4.2: Propriedades do escoamento no caso de verificação do algoritmo (escoamento entre placas paralelas).

Abaixo se encontram os campos de vorticidade, função corrente e velocidade (figuras 4.1, 4.2 e 4.3 respectivamente), obtidos com o algoritmo em linguagem de programação Python. Eles correspondem ao último incremento no tempo da análise ($t = 20s$), quando o escoamento já havia atingido o regime permanente.

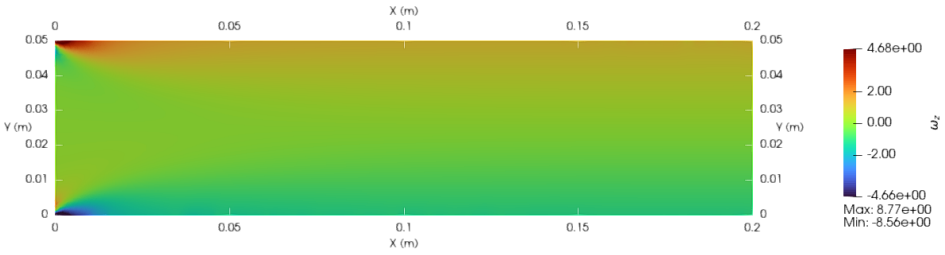

Figura 4.1: Campo de vorticidade (ω_z) do escoamento entre placas paralelas.

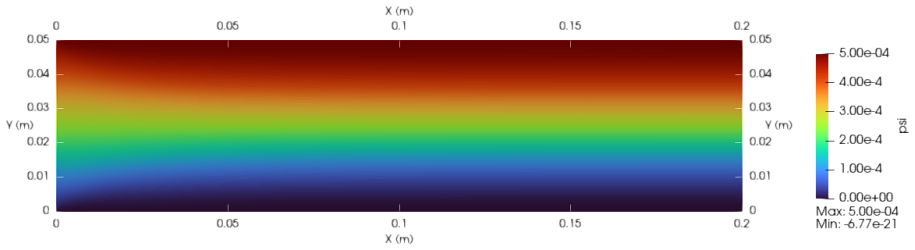

Figura 4.2: Função corrente (ψ) do escoamento entre placas paralelas.

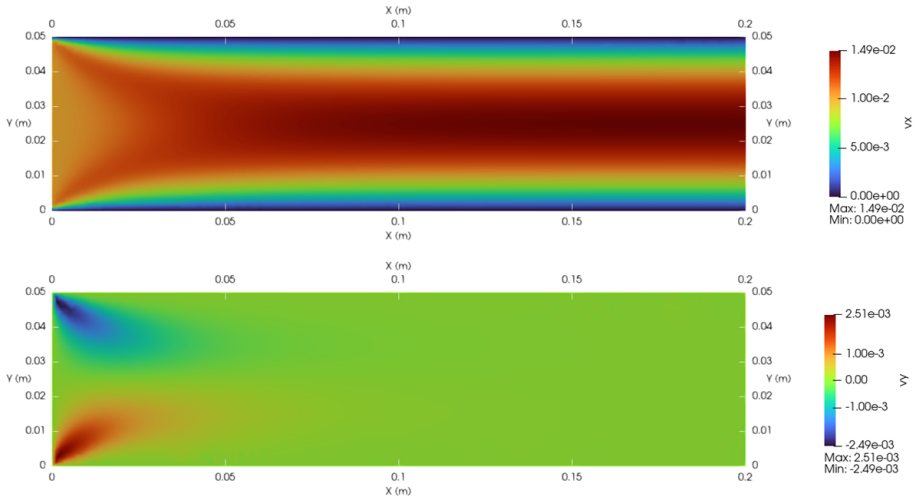

Figura 4.3: Campos de velocidade horizontal e vertical (v_x e v_y) do escoamento entre placas paralelas.

Na figura 4.4 abaixo se encontram os perfis de velocidade horizontal da solução numérica e da solução analítica:

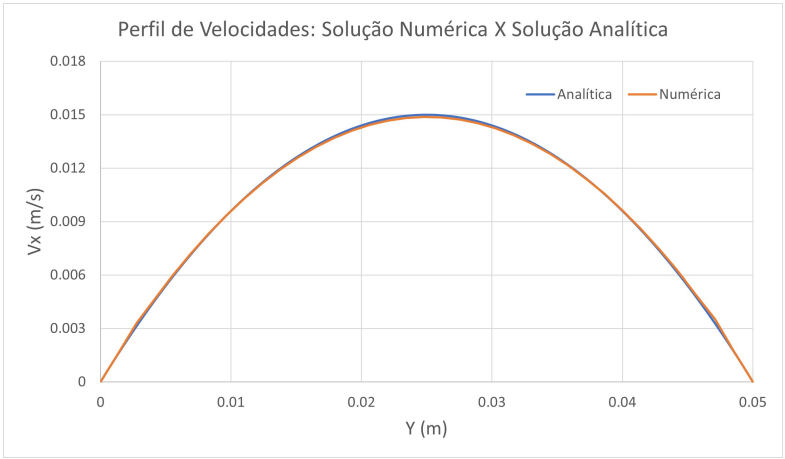

Figura 4.4: Perfis de velocidade horizontal numérico e analítico.

O perfil de velocidade numérico corresponde ao último instante da solução (t = 20.0s) e foi tomado em x = 0.18m, além do comprimento de entrada L_c do escoamento (0.15m), como mostra a figura 4.5.

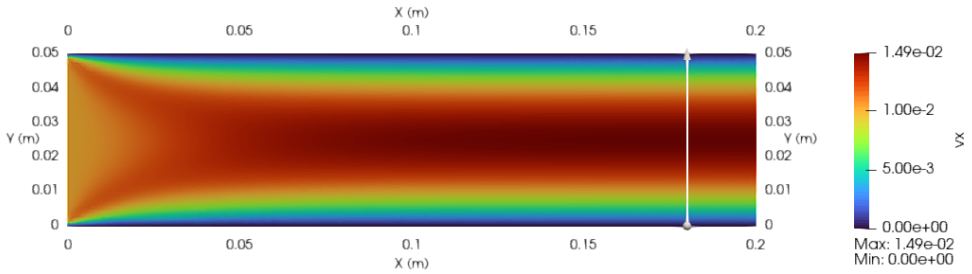

Figura 4.5: Procedimento de tomada do perfil de velocidade horizontal do escoamento no ParaView. Através do traçado vertical branco em x = 0.18m é selecionada a posição de tomada do perfil de velocidade.

Como métrica quantitativa para avaliar a qualidade da solução do algoritmo, foi utilizada a raiz do erro quadrático médio (RMSE):

$$RMSE = \sqrt{\frac{1}{N} \cdot (V^x_{ana} - V^x_{num})^2} = 8.89 \times 10^{-5}\, m/s \tag{4.2}$$

Onde V^x_{ana} e V^x_{num} correspondem respectivamente à velocidade horizontal analítica e velocidade horizontal numérica do escoamento e N ao número de pontos da amostra de velocidades ao longo do eixo y. É notável a coincidência das duas soluções, ratificando a qualidade do algoritmo.

4.2 Modelo global

Abaixo se encontra a tabela 4.3, que contém as propriedades que caracterizam o escoamento na geometria global:

	Escoamento	
Grandezas	Valores	Unidades
Espessura do filtro (L_f)	0.65	mm
Comprimento característico (L_c)	0.15	mm
Velocidade inicial (v_o^y)	25	mm/s
Viscosidade cinemática (ν)	1.00e-7	m/s
N° de Reynolds (Re)	37.5	-
Tempo de duração (T)	0.2	s
	Algoritmo	
Grandezas	Valores	Unidades
Elementos	22254	-
Incremento temporal (Δt)	0.0015	s

Tabela 4.3: Propriedades do escoamento no modelo global.

E a seguir se encontram os campos de vorticidade, função corrente e velocidade (figuras 4.6, 4.7 e 4.8 respectivamente) que caracterizam o escoamento na geometria global:

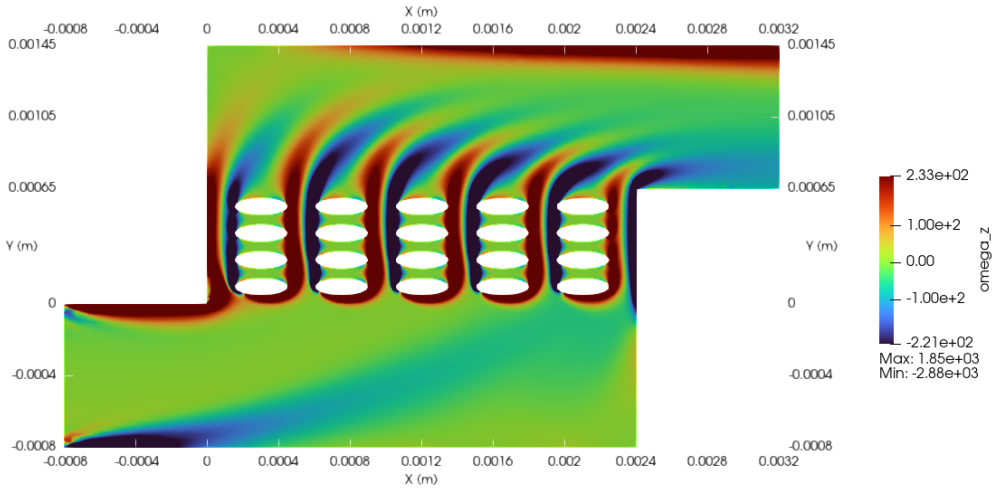

Figura 4.6: Campo de vorticidade (ω_z) na geometria global.

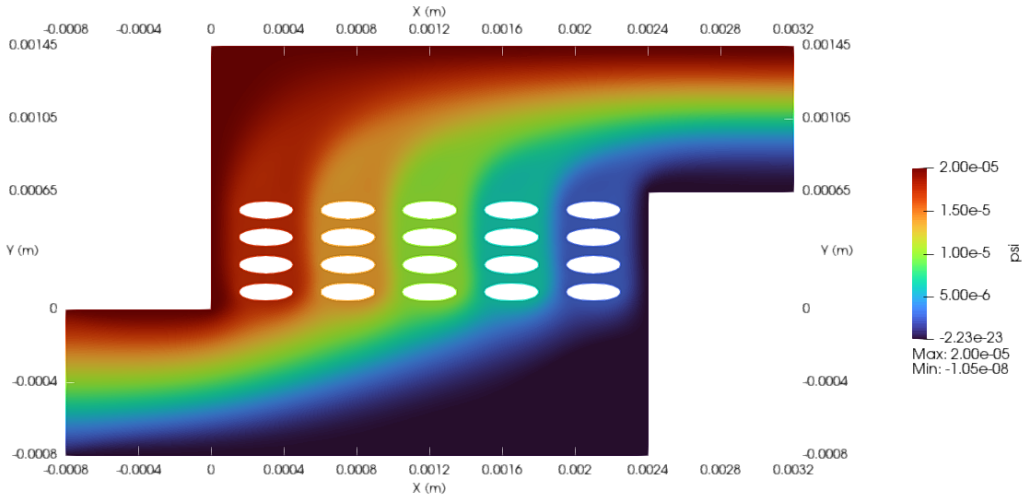

Figura 4.7: Função corrente (ψ) na geometria global.

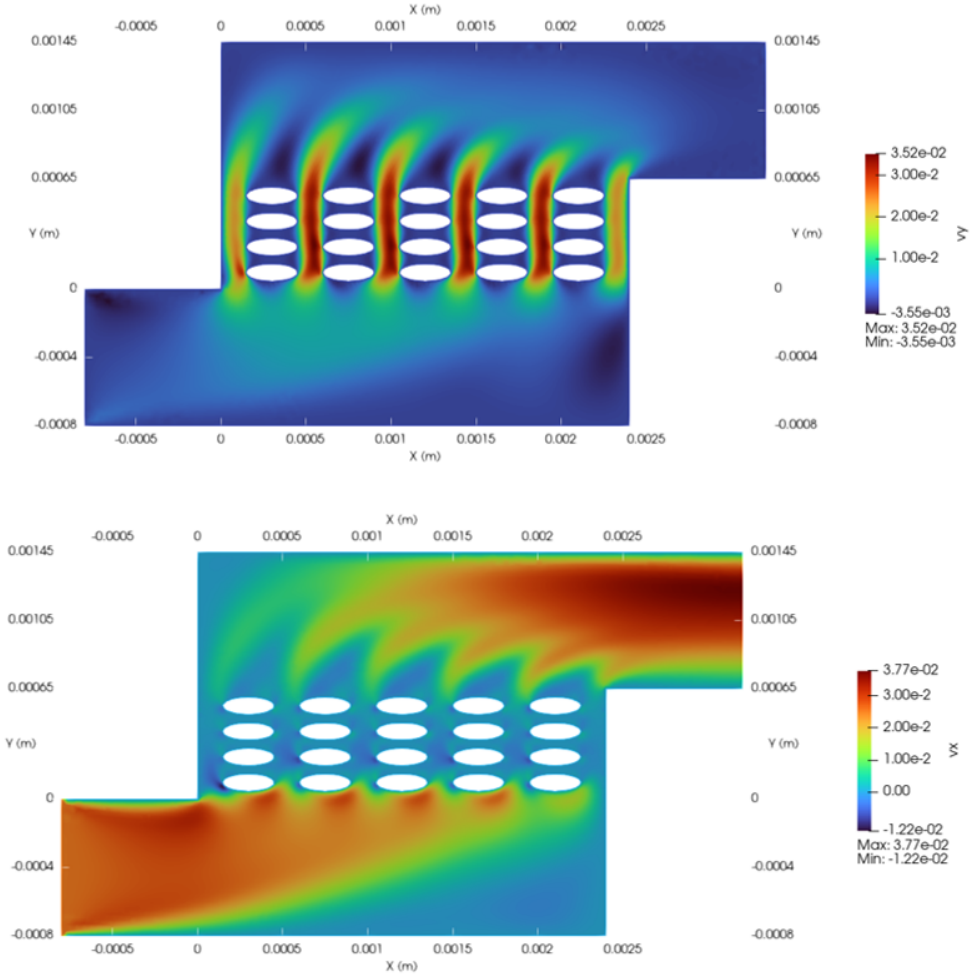

Figura 4.8: Campos de velocidade horizontal e vertical (v_y e v_x) na geometria global.

Na figura 4.8 pode-se notar que há três regiões de retenção de escoamento ao longo do percurso do fluido: Uma primeira e grande zona de recirculação no fim do canal de admissão, onde velocidade horizontal e vertical tendem a zero. Em seguida, uma série de zonas de recirculação se formam no interior do filtro, nos espaços entre as linhas de obstáculo, onde velocidade do escoamento também tende a zero. E, por último, uma terceira zona de recirculação no canal de exaustão, tal como a 1ª no canal de admissão. Essas zonas de recirculação geradas pela dinâmica do fluido podem cumprir um papel relevante na retenção (filtragem) dos particulados presentes no gás.

A responsável pelo surgimento dessas zonas de recirculação é a vorticidade ω_z, apresentada na figura 4.6. Isso fica claro na passagem do escoamento pelo filtro, onde a vorticidade atinge seu máximo nas bordas dos obstáculos próximas às passagens livres do fluido. Essas vorticidades levam o escoamento para trás dos obstáculos, onde ali fica retido.

Note ainda que nessas regiões entre obstáculos onde há literal presença de vórtices, ω_z é muito menor que nas bordas dos obstáculos, onde não há propriamente a presença de vórtices. Isso ocorre porque a velocidade do fluido nesses espaços de retenção é muito pequena, o que torna ω_z também pequena; enquanto nas bordas do obstáculo o fluido vem em alta velocidade e sujeito a alta rotação, o que explica a magnitude de ω_z nessas regiões.

4.3 Modelos locais

Agora serão avaliadas a capacidade de filtragem de 3 geometrias de obstáculos a partir da sua capacidade de geração de zonas de recirculação.

4.3.1 Obstáculos circulares

Abaixo se encontra a tabela 4.4, que contém as propriedades que caracterizam o escoamento na geometria de obstáculos circulares:

	Escoamento	
Grandezas	Valores	Unidades
Espessura do filtro (L_f)	1.3	mm
Comprimento característico (L_c)	0.1	mm
Velocidade inicial (v_o^y)	25	mm/s
Viscosidade cinemática (ν)	1.00e-7	m/s
N° de Reynolds (Re)	25	-
Tempo de duração (T)	0.2	s
	Algoritmo	
Grandezas	Valores	Unidades
Elementos	25782	-
Incremento temporal (Δt)	0.001	s

Tabela 4.4: Propriedades do escoamento no modelo local de obstáculos circulares.

A seguir se encontram os campos de vorticidade, função corrente e velocidade (figuras 4.9, 4.10 e 4.11 respectivamente) que caracterizam o escoamento na geometria circular:

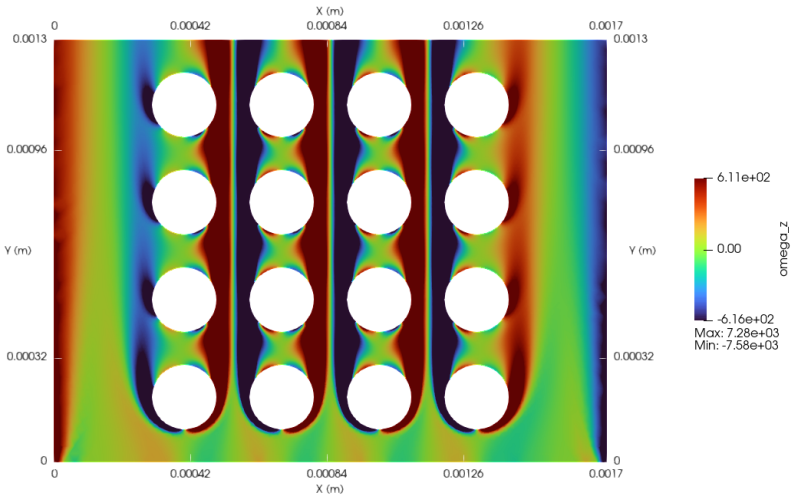

Figura 4.9: Campo de vorticidade (ω_z) na geometria de obstáculos circulares.

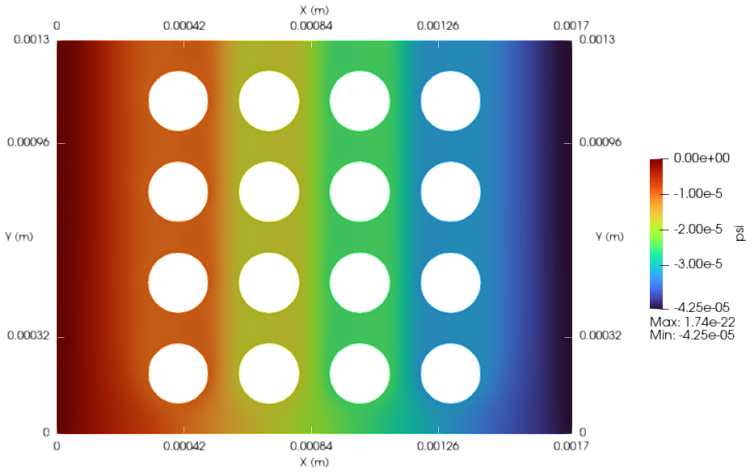

Figura 4.10: Função corrente (ψ) na geometria de obstáculos circulares.

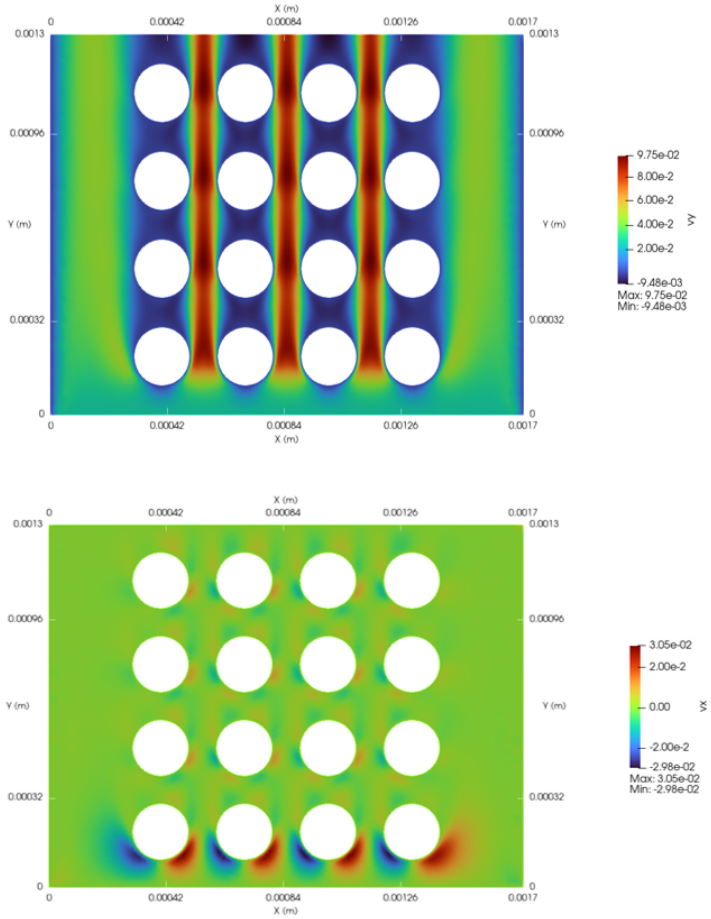

Figura 4.11: Campos de velocidade horizontal e vertical (v_y e v_x) na geometria de obstáculos circulares.

Observando a figura 4.9, fica claro como o campo de vorticidade apresenta o

mesmo comportamento observado na geometria global, agora em ampliação. Os valores mais altos de vorticidade se encontram nos corredores principais do fluido, deslocando o fluido desses corredores para os vórtices entre obstáculos.

Nas figuras 4.10 e 4.11 percebe-se também a semelhança e consistência com os campos de função corrente e velocidades gerados para a geometria global. Neles pode-se notar a formação das zonas de recirculação entre as linhas de obstáculos, potenciais contribuintes para a retenção de partículas.

4.3.2 Obstáculos delgados paralelos ao escoamento

Abaixo se encontra a tabela 4.5, que contém as propriedades que caracterizam o escoamento na geometria de obstáculos delgados orientados paralelamente ao escoamento:

	Escoamento	
Grandezas	Valores	Unidades
Espessura do filtro (L_f)	1.0	mm
Comprimento característico (L_c)	0.1	mm
Velocidade inicial (v_o^y)	25	mm/s
Viscosidade cinemática (ν)	1.00e-7	m/s
N° de Reynolds (Re)	25	-
Tempo de duração (T)	0.2	s
	Algoritmo	
Grandezas	Valores	Unidades
Elementos	21980	-
Incremento temporal (Δt)	0.001	s

Tabela 4.5: Propriedades do escoamento no modelo local de obstáculos delgados paralelos ao escoamento.

A seguir se encontram os campos de vorticidade, função corrente e velocidade (figuras 4.12, 4.13 e 4.14 respectivamente) que caracterizam o escoamento na geometria de obstáculos delgados paralelos ao escoamento:

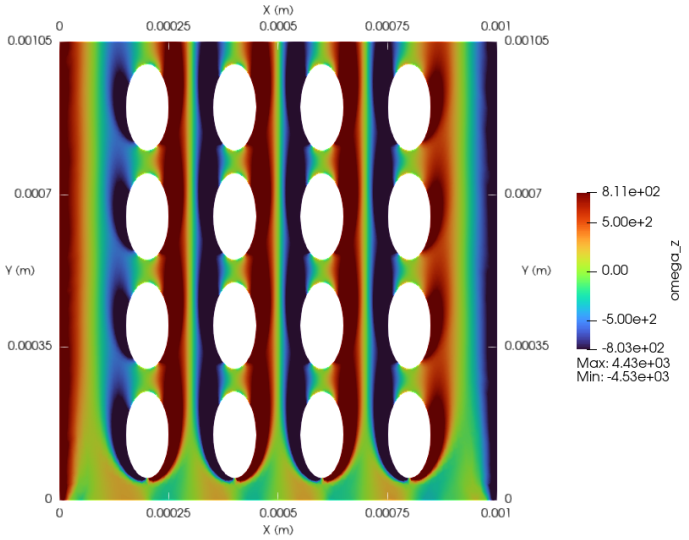

Figura 4.12: Campo de vorticidade (ω_z) na geometria de obstáculos delgados paralelos ao escoamento.

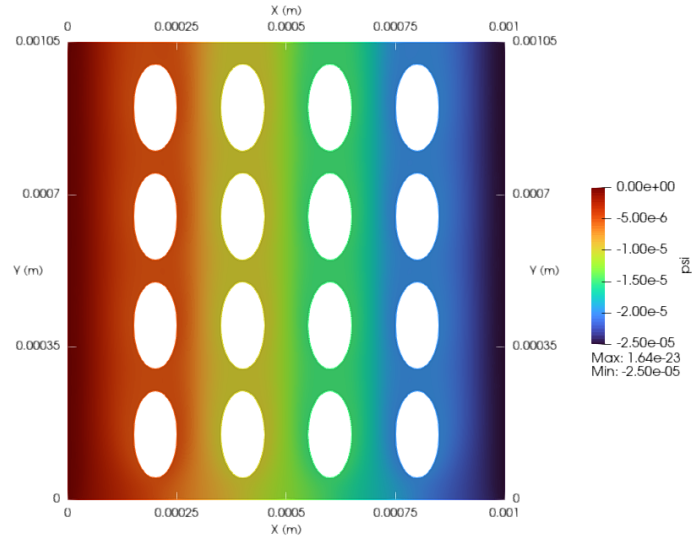

Figura 4.13: Função corrente (ψ) na geometria de obstáculos delgados paralelos ao escoamento.

A partir da figura 4.12, nota-se a diminuição perceptível das zonas de recirculação. O fato dos obstáculos delgados estarem orientados na mesma direção do escoamento prejudica a criação dessas zonas neste tipo de geometria.

No campo de velocidade vertical (v_y) ilustrado na figura 4.14 abaixo essa diminuição se confirma:

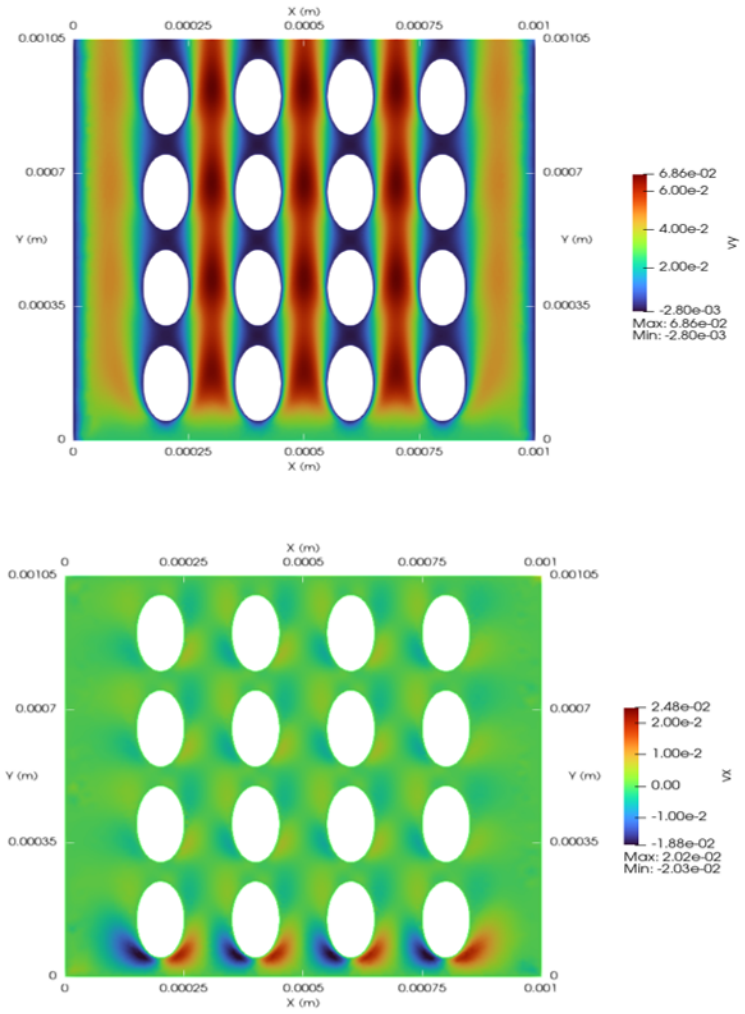

Figura 4.14: Campos de velocidade horizontal e vertical (v_y e v_x) na geometria de obstáculos delgados paralelos ao escoamento.

4.3.3 Obstáculos delgados ortogonais ao escoamento

Abaixo se encontra a tabela 4.6, que contém as propriedades que caracterizam o escoamento na geometria de obstáculos delgados orientados perpendicularmente ao escoamento:

Escoamento		
Grandezas	Valores	Unidades
Espessura do filtro (L_f)	0.7	mm
L_c	0.15	mm
Velocidade inicial (v_o^y)	25	mm/s
Viscosidade cinemática (ν)	1.00e-7	m/s
N° de Reynolds (Re)	37.5	-
Tempo de duração (T)	0.2	s
Algoritmo		
Grandezas	Valores	Unidades
Elementos	17239	-
Incremento temporal (Δt)	0.001	s

Tabela 4.6: Propriedades do escoamento no modelo local de obstáculos delgados ortogonais ao escoamento.

A seguir se encontram os campos de vorticidade, função corrente e velocidade (figuras 4.15, 4.16 e 4.17 respectivamente) que caracterizam o escoamento na geometria de obstáculos delgados ortogonais ao escoamento:

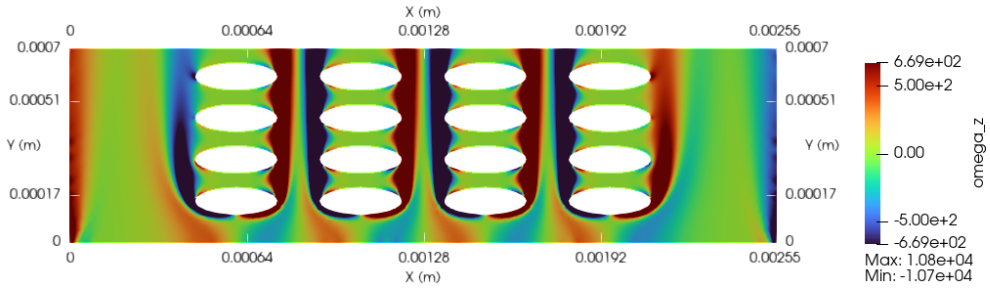

Figura 4.15: Campo de vorticidade (ω_z) na geometria de obstáculos delgados ortogonais ao escoamento.

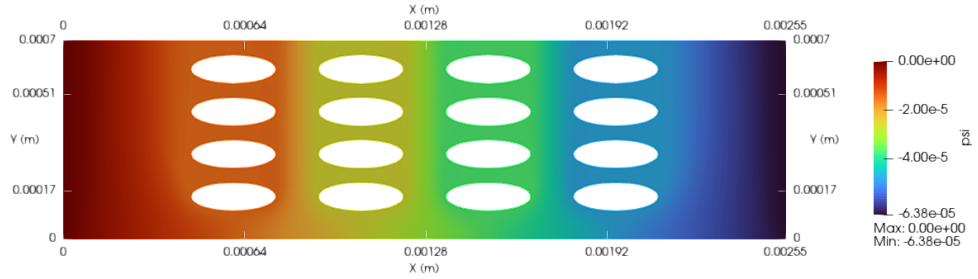

Figura 4.16: Função corrente (ψ) na geometria de obstáculos delgados ortogonais ao escoamento.

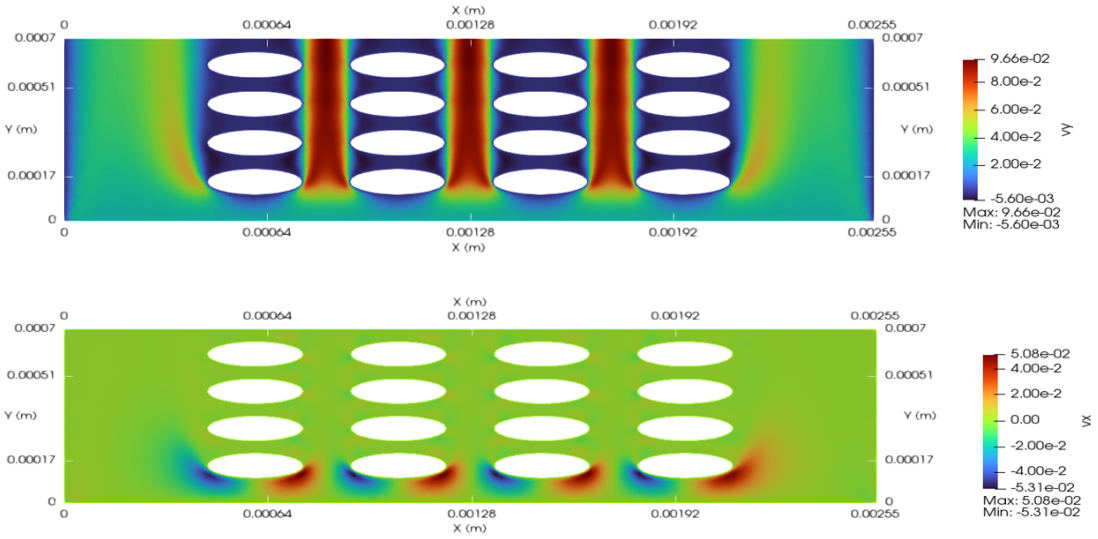

Figura 4.17: Campos de velocidade horizontal e vertical (v_y e v_x) na geometria de obstáculos delgados ortogonais ao escoamento.

Comparando os resultados das 3 geometrias analisadas, é perceptível como os obstáculos delgados na direção ortogonal a do escoamento geram maiores zonas de vorticidade que os demais. Como se pode ver na figura 4.17, o perfil delgado e orientação espacial ortogonal ao escoamento propiciam o crescimento das zonas de recirculação. Além disso, por conta da curvatura acentuada nas regiões de máxima vorticidade, os obstáculos delgados ortogonais ao escoamento apresentaram a maior vorticidade entre os 3 modelos, chegando a valores da ordem 10^4, conforme mostra a figura 4.15.

Os resultados das análises apresentadas indicam as zonas de recirculação, áreas onde a velocidade é próxima de zero, como potenciais aliados no processo de retenção

de partículas. A partir disso, pode-se inferir um maior desempenho das geometrias geradoras de zonas de recirculação no processo de filtragem de particulados em escoamento gasosos.

Capítulo 5

Conclusões

Neste trabalho foi apresentada uma breve introdução às propriedades do Biodiesel e ao filtro de particulados comumente usado em automóveis, o DPF. Nesta introdução, as emissões de particulados do biodiesel e o funcionamento do DPF são brevemente caracterizados.

Em seguida foi feita uma revisão de conceitos e ferramentas utilizados neste trabalho. O Método dos Elementos Finitos, a formulação $\psi - \omega_z$ e as modelagens de porosidade são brevemente explicadas, esclarecendo suas vantagens e desvantagens dentro do escopo deste trabalho.

No capítulo de Metodologia, as geometrias selecionadas para simulação foram introduzidas, seguidas da aplicação do Método dos Elementos Finitos na Formulação $\psi - \omega_z$ e da implementação computacional da formulação, onde também foram ilustradas as malhas de cada geometria.

Nos Resultados, um problema de convergência temporal enfrentado durante as análises e a solução para superá-lo foram apresentados. Percebeu-se que, mesmo em regime estacionário, o incremento temporal, para uma dada velocidade e um dado comprimento característico do escoamento, deve ser pequeno o suficiente a ponto do escoamento se deslocar apenas uma fração do comprimento característico. Matematicamente:

$$\Delta t \cdot v_o \leq \frac{L_c}{4} \tag{4.1}$$

Em seguida, a partir dos resultados das simulações, constatou-se que a modelagem do escoamento gasoso em DPFs através de MEF e formulação $\psi - \omega_z$ foi capaz

de descrever a dinâmica do fluido no interior desses equipamentos. A partir destas modelagens, foi possível observar um potencial papel das zonas de recirculação na retenção das partículas no interior do filtro, bem como o papel da vorticidade na criação dessas zonas de recirculação. Esta melhor compreensão dos mecanismos dinâmicos que atuam no processo de filtragem possibilita pensar a filtragem a partir da geração de zonas de recirculação.

Por fim, melhorias na modelagem do problema ainda podem ser feitas, como a implementação de partículas na simulação e o desenvolvimento de malhas mais representativas de um meio poroso real, mais complexas e desordenadas. A implementação das partículas em especial abriria uma nova frente de avaliação quantitativa da capacidade de filtragem dos elementos filtrantes, complementando a avaliação qualitativa do papel da vorticidade e zonas de recirculação na retenção de particulados. Tais propostas tornariam a abordagem deste trabalho substancialmente mais robusta e podem ser implementadas em trabalhos futuros.

Referências Bibliográficas

[1] RODRÍGUEZ-FERNÁNDEZ, J., LAPUERTA, M., SÁNCHEZ-VALDEPEÑAS, J., "Regeneration of diesel particulate filters: Effect of renewable fuels", *Renewable Energy*, v. 104, pp. 30–39, 2017.

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

[3] MILANEZ, A. Y., "Biodiesel e diesel verde no Brasil: panorama recente e perspectivas", *BNDES Setorial, Rio de Janeiro*, v. 28, n. 56, pp. 41–71, 2022.

[4] BRASIL, "Usos de Biodiesel no Brasil e no Mundo", Ministério da Agricultura, Pecuária e Abastecimento. Disponível em: `https://www.gov.br/agricultura/pt-br/assuntos/sustentabilidade/agroenergia/arquivos-publicacoes-agroenergia/usos-de-biodiesel-no-brasil-e-no-mundo.pdf/view`, acesso em: 25/09/2023.

[5] DU, Y., HU, G., XIANG, S., et al., "Estimation of the Diesel Particulate Filter Soot Load Based on an Equivalent Circuit Model", *Energies*, v. 11, n. 2, 2018.

[6] RIBEIRO, F. L. B., *The Finite Element Method in Heat Transfer and Fluid Dynamics*. 1st ed. Ciência Moderna, 2020.

[7] CARNEVALE, L. H., ANJOS, G. R., MANGIAVACCHI, N., "STREAM FUNCTION-VORTICITY FORMULATION AND HEAT TRANSPORT USING FEM FOR UNSTRUCTURED MESHES AND COMPLEX DOMAINS". In: *II Congresso Brasileiro de Fluidodinâmica Computacional*, 2018, Disponível em: https://proceedings.science/cbcfd-

2018/papers/stream-function-vorticity-formulation-and-heat-transport-using-fem-for-unstructu?lang=pt-br.

[8] CIMOLIN, F., DISCACCIATI, M., "Navier–Stokes/Forchheimer models for filtration through porous media", *Applied Numerical Mathematics*, v. 72, pp. 205–224, 2013.

[9] SPESANI, D. M., "ANÁLISE NUMÉRICA DO ESCOAMENTO DE ´ BIOCOMBUSTÍVEIS EM UM MEIO POROSO", Trabalho de Conclusão de Curso, Escola Politécnica, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2023.

[10] WANG, T. J., "A methodology for estimating the permeability of a soot deposit in a wall-flow diesel particulate filter", *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, v. 228, n. 10, pp. 1154–1169, 2014.

[11] BRASIL, "DESPACHO DO PRESIDENTE DA REPÚBLICA", Diário Oficial da União, Poder Executivo, disponível em: `https://www.in.gov.br/en/web/dou/-/despacho-do-presidente-da-republica-473383252`.

[12] BERGMANN, J. C., TUPINAMBÁ, D. D., COSTA, O. Y. A., et al., "Biodiesel production in Brazil and alternative biomass feedstocks", *Renewable and Sustainable Energy Reviews*, v. 21, pp. 411–420, 2013.

[13] ATABANI, A., SILITONGA, A., BADRUDDIN, I. A., et al., "A comprehensive review on biodiesel as an alternative energy resource and its characteristics", *Renewable and Sustainable Energy Reviews*, v. 16, pp. 2070–2093, 2012.

Apêndice A

Rotina do modelo de verificação

```
# Bibliotecas (nem todas foram usadas nesse modelo em especifico)
import math
import numpy as np
import matplotlib as mplt
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import matplotlib.tri as mtri
import meshio
from scipy.spatial import Delaunay
import scipy as sp
import pandas as pd
import os   # Library used for system interaction (such as listing
    files in a directory,...)

# Criando diretorio para armazenar solucao
## Specify the folder in which results will be exported (mostly graphs)
workingDir = r"C:\Users\Joao Pedro\Downloads\ElementosFinitos"
fileName = "Sol_Retangulo_(0.05x0.20)_0.002_mesh_9k_elem_0.05_L_1e
5_nu_20_T_0.2_dt_0.01_vxcc_50_Re_25_R"
outputPath = workingDir + "\\" + fileName

if not os.path.exists(outputPath):
    os.mkdir(outputPath)
    print("Directory " , outputPath ,  " has been created ")
else:
    print("Directory " , outputPath ,  " already exists")
```

```
27
28 # Propriedades do fluido e dominio:
29
30 nu = 0.00001  #m^2/s
31 ux = 0.01    # m/s
32 to = 0.0    # s
33 tf = 20.0    # s
34 dt = 0.2    # s
35 nt = int((tf-to)/dt+1)
36
37 Lc = 0.05    # m
38 print("Comprimento caracteristico do escoamento: {} ".format(np.around(
       Lc, 6)))
39 print("Deslocamento em um incremento no tempo: {} ".format(np.around(ux
       *dt, 6)))
40 print("Razao: {} ".format(np.around(Lc/(ux*dt), 2)))
41
42 # leitura de malha e classificacao de contorno por nome (ccName)
43 mshname = 'Retangulo_(0.05x0.20)_0.002_mesh_9k_elem_0.05_L.msh'
44 msh = meshio.read('./' + mshname)
45 X = np.array(msh.points[:,0])
46 Y = np.array(msh.points[:,1])
47 npoints = len(X)
48 IEN = msh.cells[1].data # triangles
49 ne = IEN.shape[0]
50 IENbound = msh.cells[0].data # lines
51 IENboundTypeElem = list(msh.cell_data['gmsh:physical'][0] - 1)
52 boundNames = list(msh.field_data.keys())
53 IENboundElem = [boundNames[elem] for elem in IENboundTypeElem]
54
55
56 # cria lista de nos do contorno
57 cc = np.unique(IENbound.reshape(IENbound.size))
58 ccName = [[] for i in range( len(X) )]
59 # prioridade 4
60 for elem in range(0,len(IENbound)):
61  if IENboundElem[elem] == 'Admissao':
62   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
63   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
```

```
64 # prioridade 3
65 for elem in range(0,len(IENbound)):
66  if IENboundElem[elem] == 'Exaustao':
67   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
68   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
69 # prioridade 2
70 for elem in range(0,len(IENbound)):
71  if IENboundElem[elem] == 'CCInferior':
72   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
73   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
74 # prioridade 1
75 for elem in range(0,len(IENbound)):
76  if IENboundElem[elem] == 'CCSuperior':
77   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
78   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
79
80 IENbound = cc
81
82 # Definicao dos vetores de condicoes de contorno para vx,vy e psi
83 vxcc = np.zeros( (npoints),dtype='float' )
84 vycc = np.zeros( (npoints),dtype='float' )
85 PSIcc = np.zeros( (npoints),dtype='float' )
86
87 for i in IENbound:
88  if ccName[i] == 'CCInferior':
89   vxcc[i] = 0.0
90   vycc[i] = 0.0
91   PSIcc[i] = 0.0
92  if ccName[i] == 'CCSuperior':
93   vxcc[i] = 0.0
94   vycc[i] = 0.0
95   PSIcc[i] = Lc*ux
96  if ccName[i] == 'Admissao':
97   vxcc[i] = ux
98   vycc[i] = 0.0
99   PSIcc[i] = Y[i]*ux
100
101 # inicializacao das matrizes globais
102 ne = IEN.shape[0]
```

```
103 K = np.zeros( (npoints,npoints),dtype='float' )
104 M = np.zeros( (npoints,npoints),dtype='float' )
105 Gx = np.zeros( (npoints,npoints),dtype='float' )
106 Gy = np.zeros( (npoints,npoints),dtype='float' )
107
108 for e in range(0,ne):
109  v1,v2,v3 = IEN[e]
110
111  # Calcula a area do triangulo
112  area = 0.5*np.linalg.det([[1.0,X[v1],Y[v1]],
113                            [1.0,X[v2],Y[v2]],
114                            [1.0,X[v3],Y[v3]]])
115
116  b1 = Y[v2]-Y[v3]
117  b2 = Y[v3]-Y[v1]
118  b3 = Y[v1]-Y[v2]
119
120  c1 = X[v3]-X[v2]
121  c2 = X[v1]-X[v3]
122  c3 = X[v2]-X[v1]
123
124  kxelem = np.array([[b1*b1,b1*b2,b1*b3],
125                     [b2*b1,b2*b2,b2*b3],
126                     [b3*b1,b3*b2,b3*b3]])
127  kyelem = np.array([[c1*c1,c1*c2,c1*c3],
128                     [c2*c1,c2*c2,c2*c3],
129                     [c3*c1,c3*c2,c3*c3]])
130  kelem = (1/(4*area))*kxelem + (1/(4*area))*kyelem
131
132  melem = (area/12.0)*np.array([[2.0,1.0,1.0],
133                                [1.0,2.0,1.0],
134                                [1.0,1.0,2.0]])
135
136  gxelem = (1/6)*np.array([[b1,b2,b3],
137                           [b1,b2,b3],
138                           [b1,b2,b3]])
139  gyelem = (1/6)*np.array([[c1,c2,c3],
140                           [c1,c2,c3],
141                           [c1,c2,c3]])
```

```
142
143
144  for ilocal in range(0,3):
145   iglobal = IEN[e,ilocal]
146   for jlocal in range(0,3):
147    jglobal = IEN[e,jlocal]
148
149    K[iglobal,jglobal] += kelem[ilocal,jlocal]
150    M[iglobal,jglobal] += melem[ilocal,jlocal]
151    Gx[iglobal,jglobal] += gxelem[ilocal,jlocal]
152    Gy[iglobal,jglobal] += gyelem[ilocal,jlocal]
153
154
155 # condicao inicial de vx,vy (necessarias para calculo da vorticidade)
156 vx = np.zeros( (npoints),dtype='float' )
157 vy = np.zeros( (npoints),dtype='float' )
158 PSI = np.zeros( (npoints),dtype='float' )
159
160 for i in IENbound:
161  vx[i] = vxcc[i]
162  vy[i] = vycc[i]
163
164 # Vorticidade nos contornos no instante inicial
165 vort_cc = np.linalg.solve(M,(Gx@vy - Gy@vx))
166 vort = vort_cc.copy()
167
168
169 # Gravando condicoes de contorno e iniciais da solucao em .vtk
170 point_data = {'PSIcc' : PSIcc}
171 data_vxcc = {'vxcc' : vxcc}
172 data_vycc = {'vycc' : vycc}
173 data_vort_cc = {'vort_cc' : vort_cc}
174 point_data.update(data_vxcc)
175 point_data.update(data_vycc)
176 point_data.update(data_vort_cc)
177 meshio.write_points_cells(outputPath +'\\'+'condicaoDeContorno.vtk',
178                           msh.points,
179                           msh.cells,
180                           point_data=point_data,
```

```
                                )


CampoVort = np.empty((npoints,nt))
CampoPSI = np.empty((npoints,nt))
CampoVx = np.empty((npoints,nt))
CampoVy = np.empty((npoints,nt))


# Avanco no tempo
for n in range(0,nt):

 ## Solucao do sistema linear para vorticidade

 ### Calculo da condição de contorno da vorticidade (atualizando
    vorticidade nos contornos a cada iteracao)
 vort_cc = np.linalg.solve(M,(Gx@vy - Gy@vx))

 ### Montagem da matriz A
 vx_diag = np.diag(vx)
 vy_diag = np.diag(vy)

 ### Montagem da matriz A e do vetor b de transporte de vorticidade
 A = M/dt + nu*K + vx_diag@Gx + vy_diag@Gy # implicito para conv e
    difusao
 b = (M/dt)@vort

 ### Condicao de contorno para o sistema linear Ax=b
 for i in IENbound:
     if ccName[i] == 'CCSuperior' or \
        ccName[i] == 'CCInferior' or \
        ccName[i] == 'Admissao':

          A[i,:] = 0.0 # zerando a linha
          A[i,i] = 1.0 # colocando 1 na diagonal
          b[i]   = vort_cc[i]

 ### Solucao
 vort = np.linalg.solve(A,b)
```

```
218
219 CampoVort[:,n] = vort
220
221 ## Solucao da Equacao de Corrente-Vorticidade
222
223 Apsi = K.copy()
224
225 bpsi = M@vort
226
227 ### Imposicao das c.c.s de Dirichlet
228 for i in IENbound:
229     if ccName[i] == 'CCSuperior' or \
230        ccName[i] == 'CCInferior' or \
231        ccName[i] == 'Admissao':
232
233         Apsi[i,:] = 0.0 # zerando a linha
234         Apsi[i,i] = 1.0 # colocando 1 na diagonal
235         bpsi[i]   = PSIcc[i]
236
237 ### Solucao
238 PSI = np.linalg.solve(Apsi,bpsi)
239
240 CampoPSI[:,n] = PSI
241
242 ## Obtendo campo de velocidades a partir da funcao corrrente
243
244 vx = np.linalg.solve(M,Gy@PSI)
245 vy = np.linalg.solve(M,-Gx@PSI)
246
247 for i in IENbound:
248  if ccName[i] == 'CCSuperior' or \
249     ccName[i] == 'CCInferior' or \
250     ccName[i] == 'Admissao':
251
252     vx[i] = vxcc[i]
253     vy[i] = vycc[i]
254
255 CampoVx[:,n] = vx
256 CampoVy[:,n] = vy
```

```
257
258
259 # Gravando solucao em .vtk
260 print ("... gravando em VTK passo de tempo: " + str(n))
261 point_data = {'psi' : PSI}
262 data_vx = {'vx' : vx}
263 data_vy = {'vy' : vy}
264 data_vort = {'$\omega_z$' : vort}
265 point_data.update(data_vx)
266 point_data.update(data_vy)
267 point_data.update(data_vort)
268 meshio.write_points_cells(outputPath +'\\'+'solucao-'+str(n)+'.vtk',
269                           msh.points,
270                           msh.cells,
271                           point_data=point_data,
272                           )
```

Apêndice B

Rotina do modelo global

```
# Bibliotecas
import math
import numpy as np
import matplotlib as mplt
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import matplotlib.tri as mtri
import meshio
from scipy.spatial import Delaunay
import scipy as sp
import pandas as pd
import os   # Library used for system interaction (such as listing
    files in a directory,...)

# Criando diretorio para armazenar solucao
## Specify the folder in which results will be exported (mostly graphs)
#workingDir = r"C:\Users\João Pedro\Downloads\ElementosFinitos"
fileName = "Sol_DPF_(0.65e-3x2.40e-3)_filtro_8e-4
    _canal_4x5_obstaculos_3.6e-5&1.2e-5_mesh_32k_elem_3.0e-4_D_1.5e-4
    _L_10e-7_nu_0.0015_dt_0.025_vxcc_37.5_Re_4_R"
outputPath = workingDir + "\\" + fileName

if not os.path.exists(outputPath):
    os.mkdir(outputPath)
    print("Directory " , outputPath ,  " has been created ")
else:
```

```
25      print("Directory " , outputPath ,  " already exists")
26
27 # Propriedades do fluido e domínio:
28
29 nu = 1.0*10**(-7)
30 ux = 0.025
31 to = 0.0
32 tf = 0.3
33 dt = 0.0015
34 nt = int((tf-to)/dt+1)
35
36 # leitura de malha e classificacao de contorno por nome (ccName)
37 mshname = 'DPF_(0.65e-3x2.40e-3)_filtro_8e-4_canal_4x5_obstaculos_3.6e
       -5&1.2e-5_mesh_32k_elem_3.0e-4_D_1.5e-4_L.msh'
38 msh = meshio.read('./' + mshname)
39 print(msh)
40 X = np.array(msh.points[:,0])
41 Y = np.array(msh.points[:,1])
42 npoints = len(X)
43 IEN = msh.cells[1].data # triangles
44 ne = IEN.shape[0]
45 IENbound = msh.cells[0].data # lines
46 IENboundTypeElem = list(msh.cell_data['gmsh:physical'][0] - 1)
47 boundNames = list(msh.field_data.keys())
48 IENboundElem = [boundNames[elem] for elem in IENboundTypeElem]
49
50 # cria lista de nos do contorno
51 cc = np.unique(IENbound.reshape(IENbound.size))
52 ccName = [[] for i in range( len(X) )]
53
54 for elem in range(0,len(IENbound)):
55  if IENboundElem[elem] == 'admissao - esquerda':
56   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
57   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
58  if IENboundElem[elem] == 'admissao - direita':
59   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
60   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
61  if IENboundElem[elem] == 'admissao - lado inferior':
62   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
```

```
63      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
64     if IENboundElem[elem] == 'admissao - lado superior':
65      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
66      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
67     if IENboundElem[elem] == 'filtro - esquerda':
68      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
69      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
70     if IENboundElem[elem] == 'filtro - direita':
71      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
72      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
73     if IENboundElem[elem] == 'obstaculo1':
74      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
75      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
76     if IENboundElem[elem] == 'obstaculo2':
77      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
78      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
79     if IENboundElem[elem] == 'obstaculo3':
80      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
81      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
82     if IENboundElem[elem] == 'obstaculo4':
83      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
84      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
85     if IENboundElem[elem] == 'obstaculo5':
86      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
87      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
88     if IENboundElem[elem] == 'exaustao - esquerda':
89      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
90      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
91     if IENboundElem[elem] == 'exaustao - direita':
92      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
93      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
94     if IENboundElem[elem] == 'exaustao - lado inferior':
95      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
96      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
97     if IENboundElem[elem] == 'exaustao - lado superior':
98      ccName[ IENbound[elem][0] ] = IENboundElem[elem]
99      ccName[ IENbound[elem][1] ] = IENboundElem[elem]
100
101 IENbound = cc
```

```
102
103 # Definição dos vetores de condicoes de contorno para vx,vy e psi
104 vxcc = np.zeros( (npoints),dtype='float' )
105 vycc = np.zeros( (npoints),dtype='float' )
106 PSIcc = np.zeros( (npoints),dtype='float' )
107
108 for i in IENbound:
109  if ccName[i] == 'admissao - lado inferior':
110   vxcc[i] = 0.0
111   vycc[i] = 0.0
112   PSIcc[i] = 0.0
113  if ccName[i] == 'admissao - lado superior':
114   vxcc[i] = 0.0
115   vycc[i] = 0.0
116   PSIcc[i] = ux*0.0008
117  if ccName[i] == 'admissao - esquerda':
118   vxcc[i] = ux
119   vycc[i] = 0.0
120   PSIcc[i] = ux*(Y[i] + 0.0008)   #A origem da malha não esta na quina
      do canal, mas na quina do filtro
121  if ccName[i] == 'admissao - direita':
122   vxcc[i] = 0.0
123   vycc[i] = 0.0
124   PSIcc[i] = 0.0
125  if ccName[i] == 'filtro - esquerda':
126   vxcc[i] = 0.0
127   vycc[i] = 0.0
128   PSIcc[i] = (-ux*0.0008/0.0024)*0 + ux*(0.0008)
129  if ccName[i] == 'filtro - direita':
130   vxcc[i] = 0.0
131   vycc[i] = 0.0
132   PSIcc[i] = (-ux*0.0008/0.0024)*0.00240 + ux*(0.0008)
133  if ccName[i] == 'obstaculo1':
134   vxcc[i] = 0.0
135   vycc[i] = 0.0
136   PSIcc[i] = (-ux*0.0008/0.0024)*0.00030 + ux*(0.0008)
137  if ccName[i] == 'obstaculo2':
138   vxcc[i] = 0.0
139   vycc[i] = 0.0
```

```
140   PSIcc[i] = (-ux*0.0008/0.0024)*0.00075 + ux*(0.0008)
141  if ccName[i] == 'obstaculo3':
142   vxcc[i] = 0.0
143   vycc[i] = 0.0
144   PSIcc[i] = (-ux*0.0008/0.0024)*0.00120 + ux*(0.0008)
145  if ccName[i] == 'obstaculo4':
146   vxcc[i] = 0.0
147   vycc[i] = 0.0
148   PSIcc[i] = (-ux*0.0008/0.0024)*0.00165 + ux*(0.0008)
149  if ccName[i] == 'obstaculo5':
150   vxcc[i] = 0.0
151   vycc[i] = 0.0
152   PSIcc[i] = (-ux*0.0008/0.0024)*0.00210 + ux*(0.0008)
153
154  if ccName[i] == 'exaustao - lado inferior':
155   vxcc[i] = 0.0
156   vycc[i] = 0.0
157   PSIcc[i] = 0.0
158  if ccName[i] == 'exaustao - lado superior':
159   vxcc[i] = 0.0
160   vycc[i] = 0.0
161   PSIcc[i] = ux*0.0008
162  if ccName[i] == 'exaustao - esquerda':
163   vxcc[i] = 0.0
164   vycc[i] = 0.0
165   PSIcc[i] = ux*0.0008
166
167 # condicao inicial de vx,vy (necessárias para cálculo da vorticidade)
        (Agora antes da inicialização das matrizes, pois Kest usa a média
      das velocidades de cada elemento)
168 vx = np.zeros( (npoints),dtype='float' )
169 vy = np.zeros( (npoints),dtype='float' )
170 PSI = np.zeros( (npoints),dtype='float' )
171
172 for i in IENbound:
173  vx[i] = vxcc[i]
174  vy[i] = vycc[i]
175
176
```

```
177 # inicializacao das matrizes globais
178 ne = IEN.shape[0]
179 K = np.empty( (npoints,npoints),dtype='float' )
180 M = np.empty( (npoints,npoints),dtype='float' )
181 Gx = np.empty( (npoints,npoints),dtype='float' )
182 Gy = np.empty( (npoints,npoints),dtype='float' )
183
184 for e in range(0,ne):
185  v1,v2,v3 = IEN[e]
186
187  # Calcula a área do triângulo
188  area = 0.5*np.linalg.det([[1.0,X[v1],Y[v1]],
189                            [1.0,X[v2],Y[v2]],
190                            [1.0,X[v3],Y[v3]]])
191
192  b1 = Y[v2]-Y[v3]
193  b2 = Y[v3]-Y[v1]
194  b3 = Y[v1]-Y[v2]
195
196  c1 = X[v3]-X[v2]
197  c2 = X[v1]-X[v3]
198  c3 = X[v2]-X[v1]
199
200  kxelem = np.array([[b1*b1,b1*b2,b1*b3],
201                     [b2*b1,b2*b2,b2*b3],
202                     [b3*b1,b3*b2,b3*b3]])
203  kyelem = np.array([[c1*c1,c1*c2,c1*c3],
204                     [c2*c1,c2*c2,c2*c3],
205                     [c3*c1,c3*c2,c3*c3]])
206
207  kelem = (1/(4*area))*kxelem + (1/(4*area))*kyelem
208
209  melem = (area/12.0)*np.array([[2.0,1.0,1.0],
210                                [1.0,2.0,1.0],
211                                [1.0,1.0,2.0]])
212
213  gxelem = (1/6)*np.array([[b1,b2,b3],
214                           [b1,b2,b3],
215                           [b1,b2,b3]])
```

```
 gyelem = (1/6)*np.array([[c1,c2,c3],
                          [c1,c2,c3],
                          [c1,c2,c3]])


 for ilocal in range(0,3):
  iglobal = IEN[e,ilocal]
  for jlocal in range(0,3):
   jglobal = IEN[e,jlocal]

   K[iglobal,jglobal] += kelem[ilocal,jlocal]
   M[iglobal,jglobal] += melem[ilocal,jlocal]
   Gx[iglobal,jglobal] += gxelem[ilocal,jlocal]
   Gy[iglobal,jglobal] += gyelem[ilocal,jlocal]


# Vorticidade nos contornos no instante inicial
cscM = sp.sparse.csc_matrix(M)   # Matrix form that improves sparse.
    linalg.solve
vort_cc = sp.sparse.linalg.spsolve(cscM,(Gx@vy - Gy@vx))
vort = vort_cc.copy()

# Gravando condições de contorno e iniciais da solução em .vtk
point_data = {'PSIcc' : PSIcc}
data_vxcc = {'vxcc' : vxcc}
data_vycc = {'vycc' : vycc}
data_vort_cc = {'vort_cc' : vort_cc}
point_data.update(data_vxcc)
point_data.update(data_vycc)
point_data.update(data_vort_cc)
meshio.write_points_cells(outputPath +'\\'+'condicaoDeContorno.vtk',
                          msh.points,
                          msh.cells,
                          point_data=point_data,
                          )


CampoVort = np.empty((npoints,nt))
CampoPSI = np.empty((npoints,nt))
```

```
254 CampoVx = np.empty((npoints,nt))
255 CampoVy = np.empty((npoints,nt))
256
257
258 # Avanço no tempo
259 for n in range(0,nt):
260
261  ## Solução do sistema linear para vorticidade
262
263  ### Cálculo da condição de contorno da vorticidade (atualizando
       vorticidade nos contornos a cada iteração)
264  cscM = sp.sparse.csc_matrix(M)    # Matrix form that improves sparse.
       linalg.solve
265  vort_cc = sp.sparse.linalg.spsolve(cscM,(Gx@vy - Gy@vx))
266
267  ### Montagem da matriz A
268  vx_diag = np.diag(vx)
269  vy_diag = np.diag(vy)
270
271  ### Montagem da matriz A e do vetor b de transporte de vorticidade
272  A = M/dt + nu*K + vx_diag@Gx + vy_diag@Gy # implicito para conv e
       difusao
273  b = (M/dt)@vort
274
275  ### Condição de contorno para o sistema linear Ax=b
276  for i in IENbound:
277      if ccName[i] == 'admissao - lado inferior' or \
278         ccName[i] == 'admissao - lado superior' or \
279         ccName[i] == 'admissao - direita' or \
280         ccName[i] == 'admissao - esquerda' or \
281         ccName[i] == 'filtro - direita' or \
282         ccName[i] == 'filtro - esquerda' or \
283         ccName[i] == 'obstaculo1' or \
284         ccName[i] == 'obstaculo2' or \
285         ccName[i] == 'obstaculo3' or \
286         ccName[i] == 'obstaculo4' or \
287         ccName[i] == 'obstaculo5' or \
288         ccName[i] == 'exaustao - lado inferior' or \
289         ccName[i] == 'exaustao - lado superior' or \
```

```
290         ccName[i] == 'exaustao - esquerda':
291
292          A[i,:] = 0.0 # zerando a linha
293          A[i,i] = 1.0 # colocando 1 na diagonal
294          b[i]   = vort_cc[i]
295
296 ### Solução
297 cscA = sp.sparse.csc_matrix(A)   # Matrix form that improves sparse.
      linalg.solve
298 vort = sp.sparse.linalg.spsolve(cscA,b)
299
300 CampoVort[:,n] = vort
301
302 ## Solução da Equação de Corrente-Vorticidade
303
304 Apsi = K.copy()
305
306 bpsi = M@vort
307
308 ### Imposição das c.c.s de Dirichlet
309 for i in IENbound:
310     if ccName[i] == 'admissao - lado inferior' or \
311        ccName[i] == 'admissao - lado superior' or \
312        ccName[i] == 'admissao - direita' or \
313        ccName[i] == 'admissao - esquerda' or \
314        ccName[i] == 'filtro - direita' or \
315        ccName[i] == 'filtro - esquerda' or \
316        ccName[i] == 'obstaculo1' or \
317        ccName[i] == 'obstaculo2' or \
318        ccName[i] == 'obstaculo3' or \
319        ccName[i] == 'obstaculo4' or \
320        ccName[i] == 'obstaculo5' or \
321        ccName[i] == 'exaustao - lado inferior' or \
322        ccName[i] == 'exaustao - lado superior' or \
323        ccName[i] == 'exaustao - esquerda':
324
325          Apsi[i,:] = 0.0 # zerando a linha
326          Apsi[i,i] = 1.0 # colocando 1 na diagonal
327          bpsi[i]   = PSIcc[i]
```

```
328
329 ### Solução
330 cscApsi = sp.sparse.csc_matrix(Apsi)   # Matrix form that improves
    sparse.linalg.solve
331 PSI = sp.sparse.linalg.spsolve(cscApsi,bpsi)
332
333 CampoPSI[:,n] = PSI
334
335 ## Obtendo campo de velocidades a partir da função corrrente
336
337 cscM = sp.sparse.csc_matrix(M)   # Matrix form that improves sparse.
    linalg.solve
338 vx = sp.sparse.linalg.spsolve(cscM,Gy@PSI)
339 vy = sp.sparse.linalg.spsolve(cscM,-Gx@PSI)
340
341 for i in IENbound:
342     if ccName[i] == 'admissao - lado inferior' or \
343        ccName[i] == 'admissao - lado superior' or \
344        ccName[i] == 'admissao - direita' or \
345        ccName[i] == 'admissao - esquerda' or \
346        ccName[i] == 'filtro - direita' or \
347        ccName[i] == 'filtro - esquerda' or \
348        ccName[i] == 'obstaculo1' or \
349        ccName[i] == 'obstaculo2' or \
350        ccName[i] == 'obstaculo3' or \
351        ccName[i] == 'obstaculo4' or \
352        ccName[i] == 'obstaculo5' or \
353        ccName[i] == 'exaustao - lado inferior' or \
354        ccName[i] == 'exaustao - lado superior' or \
355        ccName[i] == 'exaustao - esquerda':
356         vx[i] = vxcc[i]
357         vy[i] = vycc[i]
358
359 CampoVx[:,n] = vx
360 CampoVy[:,n] = vy
361
362
363 # Gravando solução em .vtk
364 print ("... gravando em VTK passo de tempo: " + str(n))
```

```
365 point_data = {'psi' : PSI}
366 data_vx = {'vx' : vx}
367 data_vy = {'vy' : vy}
368 data_vort = {'omega_z' : vort}
369 point_data.update(data_vx)
370 point_data.update(data_vy)
371 point_data.update(data_vort)
372 meshio.write_points_cells(outputPath +'\\'+'solucao-'+str(n)+'.vtk',
373                           msh.points,
374                           msh.cells,
375                           point_data=point_data,
376                           )
```

Apêndice C

Rotina dos modelos locais: obstáculos circulares

```
# Bibliotecas
import math
import numpy as np
import matplotlib as mplt
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import matplotlib.tri as mtri
import meshio
from scipy.spatial import Delaunay
import scipy as sp
import pandas as pd
import os   # Library used for system interaction (such as listing
    files in a directory,...)

# Criando diretorio para armazenar solucao
## Specify the folder in which results will be exported (mostly graphs)
#workingDir = r"C:\Users\João Pedro\Downloads\ElementosFinitos"
fileName = "Sol_DPF_Local_(1.7e-3x1.3e-3)_4x4_obstaculos_4.0e-5&1.25e-5
    _mesh_25k_elem_2.0e-4_D_1.0e-4_L_1e-7_nu_0.001_dt_0.025
    _vycc_25_Re_4_R"
outputPath = workingDir + "\\" + fileName

if not os.path.exists(outputPath):
```

```
22      os.mkdir(outputPath)
23      print("Directory " , outputPath ,  " has been created ")
24  else:
25      print("Directory " , outputPath ,  " already exists")
26
27  # Propriedades do fluido e domínio:
28
29  nu = 1.0*10**(-7)
30  uy = 0.025
31  to = 0.0
32  tf = 0.2
33  dt = 0.001
34  nt = int((tf-to)/dt+1)
35
36  Lc = 1.0*10**(-4)
37  print("Comprimento característico do escoamento: {} ".format(np.around(
        Lc, 6)))
38  print("Deslocamento em um incremento no tempo: {} ".format(np.around(uy
        *dt, 6)))
39  print("Razão: {} ".format(np.around(Lc/(uy*dt), 2)))
40
41  # leitura de malha e classificacao de contorno por nome (ccName)
42  mshname = 'DPF_Local_(1.7e-3x1.3e-3)_4x4_obstaculos_4.0e-5&1.25e-5
        _mesh_25k_elem_2.0e-4_D_1.0e-4_L.msh'
43  msh = meshio.read('./' + mshname)
44  print(msh)
45  X = np.array(msh.points[:,0])
46  Y = np.array(msh.points[:,1])
47  npoints = len(X)
48  IEN = msh.cells[1].data # triangles
49  ne = IEN.shape[0]
50  IENbound = msh.cells[0].data # lines
51  IENboundTypeElem = list(msh.cell_data['gmsh:physical'][0] - 1)
52  boundNames = list(msh.field_data.keys())
53  IENboundElem = [boundNames[elem] for elem in IENboundTypeElem]
54
55  # cria lista de nos do contorno
56  cc = np.unique(IENbound.reshape(IENbound.size))
57  ccName = [[] for i in range( len(X) )]
```

```
58
59 for elem in range(0,len(IENbound)):
60  if IENboundElem[elem] == 'esquerda':
61   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
62   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
63  if IENboundElem[elem] == 'direita':
64   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
65   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
66  if IENboundElem[elem] == 'inferior':
67   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
68   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
69  if IENboundElem[elem] == 'superior':
70   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
71   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
72  if IENboundElem[elem] == 'obstaculo1':
73   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
74   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
75  if IENboundElem[elem] == 'obstaculo2':
76   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
77   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
78  if IENboundElem[elem] == 'obstaculo3':
79   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
80   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
81  if IENboundElem[elem] == 'obstaculo4':
82   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
83   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
84
85 IENbound = cc
86
87 # Definição dos vetores de condicoes de contorno para vx,vy e psi
88 vxcc = np.zeros( (npoints),dtype='float' )
89 vycc = np.zeros( (npoints),dtype='float' )
90 PSIcc = np.zeros( (npoints),dtype='float' )
91
92 for i in IENbound:
93  if ccName[i] == 'inferior':
94   vxcc[i] = 0.0
95   vycc[i] = uy
96   PSIcc[i] = -uy*X[i]
```

```
 if ccName[i] == 'esquerda':
  vxcc[i] = 0.0
  vycc[i] = 0.0
  PSIcc[i] = -uy*0
 if ccName[i] == 'direita':
  vxcc[i] = 0.0
  vycc[i] = 0.0
  PSIcc[i] = -uy*0.0017
 if ccName[i] == 'obstaculo1':
  vxcc[i] = 0.0
  vycc[i] = 0.0
  PSIcc[i] = -uy*0.0004
 if ccName[i] == 'obstaculo2':
  vxcc[i] = 0.0
  vycc[i] = 0.0
  PSIcc[i] = -uy*0.0007
 if ccName[i] == 'obstaculo3':
  vxcc[i] = 0.0
  vycc[i] = 0.0
  PSIcc[i] = -uy*0.0010
 if ccName[i] == 'obstaculo4':
  vxcc[i] = 0.0
  vycc[i] = 0.0
  PSIcc[i] = -uy*0.0013

# condicao inicial de vx,vy (necessárias para cálculo da vorticidade)
      (Agora antes da inicialização das matrizes, pois Kest usa a média
     das velocidades de cada elemento)
vx = np.zeros( (npoints),dtype='float' )
vy = np.zeros( (npoints),dtype='float' )
PSI = np.zeros( (npoints),dtype='float' )

for i in IENbound:
 vx[i] = vxcc[i]
 vy[i] = vycc[i]


# inicializacao das matrizes globais
ne = IEN.shape[0]
```

```
K = np.empty( (npoints,npoints),dtype='float' )
M = np.empty( (npoints,npoints),dtype='float' )
Gx = np.empty( (npoints,npoints),dtype='float' )
Gy = np.empty( (npoints,npoints),dtype='float' )

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

 # Calcula a área do triângulo
 area = 0.5*np.linalg.det([[1.0,X[v1],Y[v1]],
                           [1.0,X[v2],Y[v2]],
                           [1.0,X[v3],Y[v3]]])

 b1 = Y[v2]-Y[v3]
 b2 = Y[v3]-Y[v1]
 b3 = Y[v1]-Y[v2]

 c1 = X[v3]-X[v2]
 c2 = X[v1]-X[v3]
 c3 = X[v2]-X[v1]

 kxelem = np.array([[b1*b1,b1*b2,b1*b3],
                    [b2*b1,b2*b2,b2*b3],
                    [b3*b1,b3*b2,b3*b3]])
 kyelem = np.array([[c1*c1,c1*c2,c1*c3],
                    [c2*c1,c2*c2,c2*c3],
                    [c3*c1,c3*c2,c3*c3]])

 kelem = (1/(4*area))*kxelem + (1/(4*area))*kyelem

 melem = (area/12.0)*np.array([[2.0,1.0,1.0],
                               [1.0,2.0,1.0],
                               [1.0,1.0,2.0]])

 gxelem = (1/6)*np.array([[b1,b2,b3],
                          [b1,b2,b3],
                          [b1,b2,b3]])
 gyelem = (1/6)*np.array([[c1,c2,c3],
                          [c1,c2,c3],
```

```
                              [c1,c2,c3]])


 for ilocal in range(0,3):
  iglobal = IEN[e,ilocal]
  for jlocal in range(0,3):
   jglobal = IEN[e,jlocal]

   K[iglobal,jglobal] += kelem[ilocal,jlocal]
   M[iglobal,jglobal] += melem[ilocal,jlocal]
   Gx[iglobal,jglobal] += gxelem[ilocal,jlocal]
   Gy[iglobal,jglobal] += gyelem[ilocal,jlocal]


# Vorticidade nos contornos no instante inicial
cscM = sp.sparse.csc_matrix(M)   # Matrix form that improves sparse.
    linalg.solve
vort_cc = sp.sparse.linalg.spsolve(cscM,(Gx@vy - Gy@vx))
vort = vort_cc.copy()

# Gravando condições de contorno e iniciais da solução em .vtk
point_data = {'PSIcc' : PSIcc}
data_vxcc = {'vxcc' : vxcc}
data_vycc = {'vycc' : vycc}
data_vort_cc = {'vort_cc' : vort_cc}
point_data.update(data_vxcc)
point_data.update(data_vycc)
point_data.update(data_vort_cc)
meshio.write_points_cells(outputPath +'\\'+'condicaoDeContorno.vtk',
                          msh.points,
                          msh.cells,
                          point_data=point_data,
                          )


CampoVort = np.empty((npoints,nt))
CampoPSI = np.empty((npoints,nt))
CampoVx = np.empty((npoints,nt))
CampoVy = np.empty((npoints,nt))
```

```
211
212
213 # Avanço no tempo
214 for n in range(0,nt):
215
216  ## Solução do sistema linear para vorticidade
217
218  ### Cálculo da condição de contorno da vorticidade (atualizando
       vorticidade nos contornos a cada iteração)
219  cscM = sp.sparse.csc_matrix(M)    # Matrix form that improves sparse.
       linalg.solve
220  vort_cc = sp.sparse.linalg.spsolve(cscM,(Gx@vy - Gy@vx))
221
222  ### Montagem da matriz A
223  vx_diag = np.diag(vx)
224  vy_diag = np.diag(vy)
225
226  ### Montagem da matriz A e do vetor b de transporte de vorticidade
227  A = M/dt + nu*K + vx_diag@Gx + vy_diag@Gy # implicito para conv e
       difusao
228  b = (M/dt)@vort
229
230  ### Condição de contorno para o sistema linear Ax=b
231  for i in IENbound:
232      if ccName[i] == 'inferior' or \
233         ccName[i] == 'direita' or \
234         ccName[i] == 'esquerda' or \
235         ccName[i] == 'obstaculo1' or \
236         ccName[i] == 'obstaculo2' or \
237         ccName[i] == 'obstaculo3' or \
238         ccName[i] == 'obstaculo4':
239          A[i,:] = 0.0 # zerando a linha
240          A[i,i] = 1.0 # colocando 1 na diagonal
241          b[i]   = vort_cc[i]
242
243  ### Solução
244  cscA = sp.sparse.csc_matrix(A)    # Matrix form that improves sparse.
       linalg.solve
245  vort = sp.sparse.linalg.spsolve(cscA,b)
```

```
246
247 CampoVort[:,n] = vort
248
249 ## Solução da Equação de Corrente-Vorticidade
250
251 Apsi = K.copy()
252
253 bpsi = M@vort
254
255 ### Imposição das c.c.s de Dirichlet
256 for i in IENbound:
257     if ccName[i] == 'inferior' or \
258        ccName[i] == 'direita' or \
259        ccName[i] == 'esquerda' or \
260        ccName[i] == 'obstaculo1' or \
261        ccName[i] == 'obstaculo2' or \
262        ccName[i] == 'obstaculo3' or \
263        ccName[i] == 'obstaculo4':
264         Apsi[i,:] = 0.0 # zerando a linha
265         Apsi[i,i] = 1.0 # colocando 1 na diagonal
266         bpsi[i]   = PSIcc[i]
267
268 ### Solução
269 cscApsi = sp.sparse.csc_matrix(Apsi)   # Matrix form that improves
    sparse.linalg.solve
270 PSI = sp.sparse.linalg.spsolve(cscApsi,bpsi)
271
272 CampoPSI[:,n] = PSI
273
274 ## Obtendo campo de velocidades a partir da função corrrente
275
276 cscM = sp.sparse.csc_matrix(M)   # Matrix form that improves sparse.
    linalg.solve
277 vx = sp.sparse.linalg.spsolve(cscM,Gy@PSI)
278 vy = sp.sparse.linalg.spsolve(cscM,-Gx@PSI)
279
280 for i in IENbound:
281     if ccName[i] == 'inferior' or \
282        ccName[i] == 'direita' or \
```

```
283         ccName[i] == 'esquerda' or \
284         ccName[i] == 'obstaculo1' or \
285         ccName[i] == 'obstaculo2' or \
286         ccName[i] == 'obstaculo3' or \
287         ccName[i] == 'obstaculo4':
288          vx[i] = vxcc[i]
289          vy[i] = vycc[i]
290
291 CampoVx[:,n] = vx
292 CampoVy[:,n] = vy
293
294
295 # Gravando solução em .vtk
296 print ("... gravando em VTK passo de tempo: " + str(n))
297 point_data = {'psi' : PSI}
298 data_vx = {'vx' : vx}
299 data_vy = {'vy' : vy}
300 data_vort = {'omega_z' : vort}
301 point_data.update(data_vx)
302 point_data.update(data_vy)
303 point_data.update(data_vort)
304 meshio.write_points_cells(outputPath +'\\'+'solucao-'+str(n)+'.vtk',
305                           msh.points,
306                           msh.cells,
307                           point_data=point_data,
308                           )
```

Apêndice D

Rotina dos modelos locais: obstáculos delgados paralelos ao escoamento

```
# Bibliotecas
import math
import numpy as np
import matplotlib as mplt
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import matplotlib.tri as mtri
import meshio
from scipy.spatial import Delaunay
import scipy as sp
import pandas as pd
import os   # Library used for system interaction (such as listing
    files in a directory,...)

# Criando diretorio para armazenar solucao
## Specify the folder in which results will be exported (mostly graphs)
#workingDir = r"C:\Users\João Pedro\Downloads\ElementosFinitos"
fileName = "Sol_DPFeliptico_DelgadoVertical_Local_(1.05e-3x1.00e-3)
    _4x4_obstaculos_3.0e-5&1.0e-5_mesh_22k_elem_1.0e-4_D_1.0e-4_L_1e-7
    _nu_0.001_dt_0.025_vycc_25_Re_4_R"
outputPath = workingDir + "\\" + fileName
```

```
20
21 if not os.path.exists(outputPath):
22     os.mkdir(outputPath)
23     print("Directory " , outputPath ,  " has been created ")
24 else:
25     print("Directory " , outputPath ,  " already exists")
26
27 # Propriedades do fluido e domínio:
28
29 nu = 1.0*10**(-7)
30 uy = 0.025
31 to = 0.0
32 tf = 0.3
33 dt = 0.001
34 nt = int((tf-to)/dt+1)
35
36 Lc = 1.0*10**(-4)
37 print("Comprimento característico do escoamento: {} ".format(np.around(
       Lc, 6)))
38 print("Deslocamento em um incremento no tempo: {} ".format(np.around(uy
       *dt, 6)))
39 print("Razão: {} ".format(np.around(Lc/(uy*dt), 2)))
40
41 # leitura de malha e classificacao de contorno por nome (ccName)
42 mshname = 'DPFeliptico_DelgadoVertical_Local_(1.05e-3x1.00e-3)
       _4x4_obstaculos_3.0e-5&1.0e-5_mesh_22k_elem_1.0e-4_D_1.0e-4_L.msh'
43 msh = meshio.read('./' + mshname)
44 print(msh)
45 X = np.array(msh.points[:,0])
46 Y = np.array(msh.points[:,1])
47 npoints = len(X)
48 IEN = msh.cells[1].data # triangles
49 ne = IEN.shape[0]
50 IENbound = msh.cells[0].data # lines
51 IENboundTypeElem = list(msh.cell_data['gmsh:physical'][0] - 1)
52 boundNames = list(msh.field_data.keys())
53 IENboundElem = [boundNames[elem] for elem in IENboundTypeElem]
54
55 # cria lista de nos do contorno
```

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

for elem in range(0,len(IENbound)):
 if IENboundElem[elem] == 'esquerda':
  ccName[ IENbound[elem][0] ] = IENboundElem[elem]
  ccName[ IENbound[elem][1] ] = IENboundElem[elem]
 if IENboundElem[elem] == 'direita':
  ccName[ IENbound[elem][0] ] = IENboundElem[elem]
  ccName[ IENbound[elem][1] ] = IENboundElem[elem]
 if IENboundElem[elem] == 'inferior':
  ccName[ IENbound[elem][0] ] = IENboundElem[elem]
  ccName[ IENbound[elem][1] ] = IENboundElem[elem]
 if IENboundElem[elem] == 'superior':
  ccName[ IENbound[elem][0] ] = IENboundElem[elem]
  ccName[ IENbound[elem][1] ] = IENboundElem[elem]
 if IENboundElem[elem] == 'obstaculo1':
  ccName[ IENbound[elem][0] ] = IENboundElem[elem]
  ccName[ IENbound[elem][1] ] = IENboundElem[elem]
 if IENboundElem[elem] == 'obstaculo2':
  ccName[ IENbound[elem][0] ] = IENboundElem[elem]
  ccName[ IENbound[elem][1] ] = IENboundElem[elem]
 if IENboundElem[elem] == 'obstaculo3':
  ccName[ IENbound[elem][0] ] = IENboundElem[elem]
  ccName[ IENbound[elem][1] ] = IENboundElem[elem]
 if IENboundElem[elem] == 'obstaculo4':
  ccName[ IENbound[elem][0] ] = IENboundElem[elem]
  ccName[ IENbound[elem][1] ] = IENboundElem[elem]

IENbound = cc

# Definição dos vetores de condicoes de contorno para vx,vy e psi
vxcc = np.zeros( (npoints),dtype='float' )
vycc = np.zeros( (npoints),dtype='float' )
PSIcc = np.zeros( (npoints),dtype='float' )

for i in IENbound:
 if ccName[i] == 'inferior':
  vxcc[i] = 0.0
```

```
95      vycc[i] = uy
96      PSIcc[i] = -uy*X[i]
97    if ccName[i] == 'esquerda':
98      vxcc[i] = 0.0
99      vycc[i] = 0.0
100     PSIcc[i] = -uy*0
101   if ccName[i] == 'direita':
102     vxcc[i] = 0.0
103     vycc[i] = 0.0
104     PSIcc[i] = -uy*0.001
105   if ccName[i] == 'obstaculo1':
106     vxcc[i] = 0.0
107     vycc[i] = 0.0
108     PSIcc[i] = -uy*0.0002
109   if ccName[i] == 'obstaculo2':
110     vxcc[i] = 0.0
111     vycc[i] = 0.0
112     PSIcc[i] = -uy*0.0004
113   if ccName[i] == 'obstaculo3':
114     vxcc[i] = 0.0
115     vycc[i] = 0.0
116     PSIcc[i] = -uy*0.0006
117   if ccName[i] == 'obstaculo4':
118     vxcc[i] = 0.0
119     vycc[i] = 0.0
120     PSIcc[i] = -uy*0.0008
121
122 # condicao inicial de vx,vy (necessárias para cálculo da vorticidade)
        (Agora antes da inicialização das matrizes, pois Kest usa a média
      das velocidades de cada elemento)
123 vx = np.zeros( (npoints),dtype='float' )
124 vy = np.zeros( (npoints),dtype='float' )
125 PSI = np.zeros( (npoints),dtype='float' )
126
127 for i in IENbound:
128   vx[i] = vxcc[i]
129   vy[i] = vycc[i]
130
131
```

```
# inicializacao das matrizes globais
ne = IEN.shape[0]
K = np.empty( (npoints,npoints),dtype='float' )
M = np.empty( (npoints,npoints),dtype='float' )
Gx = np.empty( (npoints,npoints),dtype='float' )
Gy = np.empty( (npoints,npoints),dtype='float' )

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

 # Calcula a área do triângulo
 area = 0.5*np.linalg.det([[1.0,X[v1],Y[v1]],
                           [1.0,X[v2],Y[v2]],
                           [1.0,X[v3],Y[v3]]])

 b1 = Y[v2]-Y[v3]
 b2 = Y[v3]-Y[v1]
 b3 = Y[v1]-Y[v2]

 c1 = X[v3]-X[v2]
 c2 = X[v1]-X[v3]
 c3 = X[v2]-X[v1]

 kxelem = np.array([[b1*b1,b1*b2,b1*b3],
                    [b2*b1,b2*b2,b2*b3],
                    [b3*b1,b3*b2,b3*b3]])
 kyelem = np.array([[c1*c1,c1*c2,c1*c3],
                    [c2*c1,c2*c2,c2*c3],
                    [c3*c1,c3*c2,c3*c3]])

 kelem = (1/(4*area))*kxelem + (1/(4*area))*kyelem

 melem = (area/12.0)*np.array([[2.0,1.0,1.0],
                               [1.0,2.0,1.0],
                               [1.0,1.0,2.0]])

 gxelem = (1/6)*np.array([[b1,b2,b3],
                          [b1,b2,b3],
                          [b1,b2,b3]])
```

```
171  gyelem = (1/6)*np.array([[c1,c2,c3],
172                           [c1,c2,c3],
173                           [c1,c2,c3]])
174
175
176  for ilocal in range(0,3):
177   iglobal = IEN[e,ilocal]
178   for jlocal in range(0,3):
179    jglobal = IEN[e,jlocal]
180
181    K[iglobal,jglobal] += kelem[ilocal,jlocal]
182    M[iglobal,jglobal] += melem[ilocal,jlocal]
183    Gx[iglobal,jglobal] += gxelem[ilocal,jlocal]
184    Gy[iglobal,jglobal] += gyelem[ilocal,jlocal]
185
186
187 # Vorticidade nos contornos no instante inicial
188 cscM = sp.sparse.csc_matrix(M)   # Matrix form that improves sparse.
        linalg.solve
189 vort_cc = sp.sparse.linalg.spsolve(cscM,(Gx@vy - Gy@vx))
190 vort = vort_cc.copy()
191
192 # Gravando condições de contorno e iniciais da solução em .vtk
193 point_data = {'PSIcc' : PSIcc}
194 data_vxcc = {'vxcc' : vxcc}
195 data_vycc = {'vycc' : vycc}
196 data_vort_cc = {'vort_cc' : vort_cc}
197 point_data.update(data_vxcc)
198 point_data.update(data_vycc)
199 point_data.update(data_vort_cc)
200 meshio.write_points_cells(outputPath +'\\'+'condicaoDeContorno.vtk',
201                           msh.points,
202                           msh.cells,
203                           point_data=point_data,
204                           )
205
206
207 CampoVort = np.empty((npoints,nt))
208 CampoPSI = np.empty((npoints,nt))
```

```
209 CampoVx = np.empty((npoints,nt))
210 CampoVy = np.empty((npoints,nt))
211
212
213 # Avanço no tempo
214 for n in range(0,nt):
215
216  ## Solução do sistema linear para vorticidade
217
218  ### Cálculo da condição de contorno da vorticidade (atualizando
       vorticidade nos contornos a cada iteração)
219  cscM = sp.sparse.csc_matrix(M)    # Matrix form that improves sparse.
       linalg.solve
220  vort_cc = sp.sparse.linalg.spsolve(cscM,(Gx@vy - Gy@vx))
221
222  ### Montagem da matriz A
223  vx_diag = np.diag(vx)
224  vy_diag = np.diag(vy)
225
226  ### Montagem da matriz A e do vetor b de transporte de vorticidade
227  A = M/dt + nu*K + vx_diag@Gx + vy_diag@Gy # implicito para conv e
       difusao
228  b = (M/dt)@vort
229
230  ### Condição de contorno para o sistema linear Ax=b
231  for i in IENbound:
232      if ccName[i] == 'inferior' or \
233         ccName[i] == 'direita' or \
234         ccName[i] == 'esquerda' or \
235         ccName[i] == 'obstaculo1' or \
236         ccName[i] == 'obstaculo2' or \
237         ccName[i] == 'obstaculo3' or \
238         ccName[i] == 'obstaculo4':
239          A[i,:] = 0.0 # zerando a linha
240          A[i,i] = 1.0 # colocando 1 na diagonal
241          b[i]   = vort_cc[i]
242
243  ### Solução
244  cscA = sp.sparse.csc_matrix(A)    # Matrix form that improves sparse.
```

```
    linalg.solve
vort = sp.sparse.linalg.spsolve(cscA,b)

CampoVort[:,n] = vort

## Solução da Equação de Corrente-Vorticidade

Apsi = K.copy()

bpsi = M@vort

### Imposição das c.c.s de Dirichlet
for i in IENbound:
    if ccName[i] == 'inferior' or \
       ccName[i] == 'direita' or \
       ccName[i] == 'esquerda' or \
       ccName[i] == 'obstaculo1' or \
       ccName[i] == 'obstaculo2' or \
       ccName[i] == 'obstaculo3' or \
       ccName[i] == 'obstaculo4':
        Apsi[i,:] = 0.0 # zerando a linha
        Apsi[i,i] = 1.0 # colocando 1 na diagonal
        bpsi[i]   = PSIcc[i]

### Solução
cscApsi = sp.sparse.csc_matrix(Apsi)   # Matrix form that improves
    sparse.linalg.solve
PSI = sp.sparse.linalg.spsolve(cscApsi,bpsi)

CampoPSI[:,n] = PSI

## Obtendo campo de velocidades a partir da função corrrente

cscM = sp.sparse.csc_matrix(M)   # Matrix form that improves sparse.
    linalg.solve
vx = sp.sparse.linalg.spsolve(cscM,Gy@PSI)
vy = sp.sparse.linalg.spsolve(cscM,-Gx@PSI)

for i in IENbound:
```

```
281     if ccName[i] == 'inferior' or \
282        ccName[i] == 'direita' or \
283        ccName[i] == 'esquerda' or \
284        ccName[i] == 'obstaculo1' or \
285        ccName[i] == 'obstaculo2' or \
286        ccName[i] == 'obstaculo3' or \
287        ccName[i] == 'obstaculo4':
288         vx[i] = vxcc[i]
289         vy[i] = vycc[i]
290
291 CampoVx[:,n] = vx
292 CampoVy[:,n] = vy
293
294
295 # Gravando solução em .vtk
296 print ("... gravando em VTK passo de tempo: " + str(n))
297 point_data = {'psi' : PSI}
298 data_vx = {'vx' : vx}
299 data_vy = {'vy' : vy}
300 data_vort = {'omega_z' : vort}
301 point_data.update(data_vx)
302 point_data.update(data_vy)
303 point_data.update(data_vort)
304 meshio.write_points_cells(outputPath +'\\'+'solucao-'+str(n)+'.vtk',
305                           msh.points,
306                           msh.cells,
307                           point_data=point_data,
308                           )
```

Apêndice E

Rotina dos modelos locais: obstáculos delgados ortogonais ao escoamento

```
# Bibliotecas
import math
import numpy as np
import matplotlib as mplt
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import matplotlib.tri as mtri
import meshio
from scipy.spatial import Delaunay
import scipy as sp
import pandas as pd
import os   # Library used for system interaction (such as listing
    files in a directory,...)

# Criando diretorio para armazenar solucao
## Specify the folder in which results will be exported (mostly graphs)
#workingDir = r"C:\Users\João Pedro\Downloads\ElementosFinitos"
fileName = "DPFeliptico_DelgadoHorizontal_Local_(0.70e-3x2.55e-3)
    _4x4_obstaculos_3.6e-5&1.4e-5_mesh_17k_elem_3.0e-4_D_1.5e-4_L_1e-7
    _nu_0.001_dt_0.025_vycc_37.5_Re_6_R"
outputPath = workingDir + "\\" + fileName
```

```
20
21 if not os.path.exists(outputPath):
22     os.mkdir(outputPath)
23     print("Directory " , outputPath ,  " has been created ")
24 else:
25     print("Directory " , outputPath ,  " already exists")
26
27 # Propriedades do fluido e domínio:
28
29 nu = 1.0*10**(-7)
30 uy = 0.025
31 to = 0.0
32 tf = 0.3
33 dt = 0.001
34 nt = int((tf-to)/dt+1)
35
36 Lc = 1.5*10**(-4)
37 print("Comprimento característico do escoamento: {} ".format(np.around(
       Lc, 6)))
38 print("Deslocamento em um incremento no tempo: {} ".format(np.around(uy
       *dt, 6)))
39 print("Razão: {} ".format(np.around(Lc/(uy*dt), 2)))
40
41 # leitura de malha e classificacao de contorno por nome (ccName)
42 mshname = 'DPFeliptico_DelgadoHorizontal_Local_(0.70e-3x2.55e-3)
       _4x4_obstaculos_3.6e-5&1.4e-5_mesh_17k_elem_3.0e-4_D_1.5e-4_L.msh'
43 msh = meshio.read('./' + mshname)
44 print(msh)
45 X = np.array(msh.points[:,0])
46 Y = np.array(msh.points[:,1])
47 npoints = len(X)
48 IEN = msh.cells[1].data # triangles
49 ne = IEN.shape[0]
50 IENbound = msh.cells[0].data # lines
51 IENboundTypeElem = list(msh.cell_data['gmsh:physical'][0] - 1)
52 boundNames = list(msh.field_data.keys())
53 IENboundElem = [boundNames[elem] for elem in IENboundTypeElem]
54
55 # cria lista de nos do contorno
```

```
56 cc = np.unique(IENbound.reshape(IENbound.size))
57 ccName = [[] for i in range( len(X) )]
58
59 for elem in range(0,len(IENbound)):
60  if IENboundElem[elem] == 'esquerda':
61   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
62   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
63  if IENboundElem[elem] == 'direita':
64   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
65   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
66  if IENboundElem[elem] == 'inferior':
67   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
68   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
69  if IENboundElem[elem] == 'superior':
70   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
71   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
72  if IENboundElem[elem] == 'obstaculo1':
73   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
74   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
75  if IENboundElem[elem] == 'obstaculo2':
76   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
77   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
78  if IENboundElem[elem] == 'obstaculo3':
79   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
80   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
81  if IENboundElem[elem] == 'obstaculo4':
82   ccName[ IENbound[elem][0] ] = IENboundElem[elem]
83   ccName[ IENbound[elem][1] ] = IENboundElem[elem]
84
85 IENbound = cc
86
87 # Definição dos vetores de condicoes de contorno para vx,vy e psi
88 vxcc = np.zeros( (npoints),dtype='float' )
89 vycc = np.zeros( (npoints),dtype='float' )
90 PSIcc = np.zeros( (npoints),dtype='float' )
91
92 for i in IENbound:
93  if ccName[i] == 'inferior':
94   vxcc[i] = 0.0
```

```
95    vycc[i] = uy
96    PSIcc[i] = -uy*X[i]
97   if ccName[i] == 'esquerda':
98    vxcc[i] = 0.0
99    vycc[i] = 0.0
100   PSIcc[i] = -uy*0
101  if ccName[i] == 'direita':
102   vxcc[i] = 0.0
103   vycc[i] = 0.0
104   PSIcc[i] = -uy*0.00255
105  if ccName[i] == 'obstaculo1':
106   vxcc[i] = 0.0
107   vycc[i] = 0.0
108   PSIcc[i] = -uy*0.00060
109  if ccName[i] == 'obstaculo2':
110   vxcc[i] = 0.0
111   vycc[i] = 0.0
112   PSIcc[i] = -uy*0.00105
113  if ccName[i] == 'obstaculo3':
114   vxcc[i] = 0.0
115   vycc[i] = 0.0
116   PSIcc[i] = -uy*0.00150
117  if ccName[i] == 'obstaculo4':
118   vxcc[i] = 0.0
119   vycc[i] = 0.0
120   PSIcc[i] = -uy*0.00195
121
122 # condicao inicial de vx,vy (necessárias para cálculo da vorticidade)
        (Agora antes da inicialização das matrizes, pois Kest usa a média
      das velocidades de cada elemento)
123 vx = np.zeros( (npoints),dtype='float' )
124 vy = np.zeros( (npoints),dtype='float' )
125 PSI = np.zeros( (npoints),dtype='float' )
126
127 for i in IENbound:
128  vx[i] = vxcc[i]
129  vy[i] = vycc[i]
130
131
```

```
132 # inicializacao das matrizes globais
133 ne = IEN.shape[0]
134 K = np.empty( (npoints,npoints),dtype='float' )
135 M = np.empty( (npoints,npoints),dtype='float' )
136 Gx = np.empty( (npoints,npoints),dtype='float' )
137 Gy = np.empty( (npoints,npoints),dtype='float' )
138
139 for e in range(0,ne):
140  v1,v2,v3 = IEN[e]
141
142  # Calcula a área do triângulo
143  area = 0.5*np.linalg.det([[1.0,X[v1],Y[v1]],
144                            [1.0,X[v2],Y[v2]],
145                            [1.0,X[v3],Y[v3]]])
146
147  b1 = Y[v2]-Y[v3]
148  b2 = Y[v3]-Y[v1]
149  b3 = Y[v1]-Y[v2]
150
151  c1 = X[v3]-X[v2]
152  c2 = X[v1]-X[v3]
153  c3 = X[v2]-X[v1]
154
155  kxelem = np.array([[b1*b1,b1*b2,b1*b3],
156                     [b2*b1,b2*b2,b2*b3],
157                     [b3*b1,b3*b2,b3*b3]])
158  kyelem = np.array([[c1*c1,c1*c2,c1*c3],
159                     [c2*c1,c2*c2,c2*c3],
160                     [c3*c1,c3*c2,c3*c3]])
161
162  kelem = (1/(4*area))*kxelem + (1/(4*area))*kyelem
163
164  melem = (area/12.0)*np.array([[2.0,1.0,1.0],
165                                [1.0,2.0,1.0],
166                                [1.0,1.0,2.0]])
167
168  gxelem = (1/6)*np.array([[b1,b2,b3],
169                           [b1,b2,b3],
170                           [b1,b2,b3]])
```

```
171  gyelem = (1/6)*np.array([[c1,c2,c3],
172                           [c1,c2,c3],
173                           [c1,c2,c3]])
174
175
176  for ilocal in range(0,3):
177   iglobal = IEN[e,ilocal]
178   for jlocal in range(0,3):
179    jglobal = IEN[e,jlocal]
180
181    K[iglobal,jglobal] += kelem[ilocal,jlocal]
182    M[iglobal,jglobal] += melem[ilocal,jlocal]
183    Gx[iglobal,jglobal] += gxelem[ilocal,jlocal]
184    Gy[iglobal,jglobal] += gyelem[ilocal,jlocal]
185
186
187 # Vorticidade nos contornos no instante inicial
188 cscM = sp.sparse.csc_matrix(M)   # Matrix form that improves sparse.
        linalg.solve
189 vort_cc = sp.sparse.linalg.spsolve(cscM,(Gx@vy - Gy@vx))
190 vort = vort_cc.copy()
191
192 # Gravando condições de contorno e iniciais da solução em .vtk
193 point_data = {'PSIcc' : PSIcc}
194 data_vxcc = {'vxcc' : vxcc}
195 data_vycc = {'vycc' : vycc}
196 data_vort_cc = {'vort_cc' : vort_cc}
197 point_data.update(data_vxcc)
198 point_data.update(data_vycc)
199 point_data.update(data_vort_cc)
200 meshio.write_points_cells(outputPath +'\\'+'condicaoDeContorno.vtk',
201                           msh.points,
202                           msh.cells,
203                           point_data=point_data,
204                           )
205
206
207 CampoVort = np.empty((npoints,nt))
208 CampoPSI = np.empty((npoints,nt))
```

```
209 CampoVx = np.empty((npoints,nt))
210 CampoVy = np.empty((npoints,nt))
211
212
213 # Avanço no tempo
214 for n in range(0,nt):
215
216  ## Solução do sistema linear para vorticidade
217
218  ### Cálculo da condição de contorno da vorticidade (atualizando
       vorticidade nos contornos a cada iteração)
219  cscM = sp.sparse.csc_matrix(M)    # Matrix form that improves sparse.
       linalg.solve
220  vort_cc = sp.sparse.linalg.spsolve(cscM,(Gx@vy - Gy@vx))
221
222  ### Montagem da matriz A
223  vx_diag = np.diag(vx)
224  vy_diag = np.diag(vy)
225
226  ### Montagem da matriz A e do vetor b de transporte de vorticidade
227  A = M/dt + nu*K + vx_diag@Gx + vy_diag@Gy # implicito para conv e
       difusao
228  b = (M/dt)@vort
229
230  ### Condição de contorno para o sistema linear Ax=b
231  for i in IENbound:
232      if ccName[i] == 'inferior' or \
233         ccName[i] == 'direita' or \
234         ccName[i] == 'esquerda' or \
235         ccName[i] == 'obstaculo1' or \
236         ccName[i] == 'obstaculo2' or \
237         ccName[i] == 'obstaculo3' or \
238         ccName[i] == 'obstaculo4':
239          A[i,:] = 0.0 # zerando a linha
240          A[i,i] = 1.0 # colocando 1 na diagonal
241          b[i]   = vort_cc[i]
242
243  ### Solução
244  cscA = sp.sparse.csc_matrix(A)    # Matrix form that improves sparse.
```

```
    linalg.solve
vort = sp.sparse.linalg.spsolve(cscA,b)

CampoVort[:,n] = vort

## Solução da Equação de Corrente-Vorticidade

Apsi = K.copy()

bpsi = M@vort

### Imposição das c.c.s de Dirichlet
for i in IENbound:
    if ccName[i] == 'inferior' or \
       ccName[i] == 'direita' or \
       ccName[i] == 'esquerda' or \
       ccName[i] == 'obstaculo1' or \
       ccName[i] == 'obstaculo2' or \
       ccName[i] == 'obstaculo3' or \
       ccName[i] == 'obstaculo4':
        Apsi[i,:] = 0.0 # zerando a linha
        Apsi[i,i] = 1.0 # colocando 1 na diagonal
        bpsi[i]   = PSIcc[i]

### Solução
cscApsi = sp.sparse.csc_matrix(Apsi)   # Matrix form that improves
    sparse.linalg.solve
PSI = sp.sparse.linalg.spsolve(cscApsi,bpsi)

CampoPSI[:,n] = PSI

## Obtendo campo de velocidades a partir da função corrrente

cscM = sp.sparse.csc_matrix(M)   # Matrix form that improves sparse.
    linalg.solve
vx = sp.sparse.linalg.spsolve(cscM,Gy@PSI)
vy = sp.sparse.linalg.spsolve(cscM,-Gx@PSI)

for i in IENbound:
```

```
281     if ccName[i] == 'inferior' or \
282        ccName[i] == 'direita' or \
283        ccName[i] == 'esquerda' or \
284        ccName[i] == 'obstaculo1' or \
285        ccName[i] == 'obstaculo2' or \
286        ccName[i] == 'obstaculo3' or \
287        ccName[i] == 'obstaculo4':
288         vx[i] = vxcc[i]
289         vy[i] = vycc[i]
290
291 CampoVx[:,n] = vx
292 CampoVy[:,n] = vy
293
294
295 # Gravando solução em .vtk
296 print ("... gravando em VTK passo de tempo: " + str(n))
297 point_data = {'psi' : PSI}
298 data_vx = {'vx' : vx}
299 data_vy = {'vy' : vy}
300 data_vort = {'omega_z' : vort}
301 point_data.update(data_vx)
302 point_data.update(data_vy)
303 point_data.update(data_vort)
304 meshio.write_points_cells(outputPath +'\\'+'solucao-'+str(n)+'.vtk',
305                           msh.points,
306                           msh.cells,
307                           point_data=point_data,
308                           )
```