UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

FERNANDO DATZ DO NASCIMENTO

Análise comparativa de softwares para simulação de camada limite sobre uma superfície com irregularidades

São Carlos

2024

FERNANDO DATZ DO NASCIMENTO

Análise comparativa de softwares para simulação de camada limite sobre uma superfície com irregularidades

Monografia apresentada ao Curso de Engenharia Aeronáutica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do Título de Engenheiro Aeronáutico.
Orientador: Prof. Dr. Marcello Augusto Faraco de Medeiros

São Carlos

2024

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Nascimento, Fernando Datz do
N244a Análise comparativa de softwares para simulação de camada limite sobre uma superfície com irregularidades / Fernando Datz do Nascimento; orientador Marcello Augusto Faraco de Medeiros. São Carlos, 2024.

Monografia (Graduação em Engenharia Aeronáutica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2024.

1. dinâmica dos fluidos computacional. 2. camada limite. 3. escoamento sobre cavidade. 4. método dos elementos espectrais. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO
Approval sheet

Candidato */ Student*: Fernando Datz do Nascimento
Título do TCC */ Title* **:** Análise comparativa de softwares para simulação de camada limite sobre uma superfície com irregularidades
Data de defesa */ Date*: 01/07/2024

Comissão Julgadora */ Examining committee*	**Resultado** */ result*
Professor Associado Marcello Augusto Faraco de Medeiros	Aprovado
Instituição / Affiliation: EESC - SAA	
Professor Doutor Ricardo Afonso Angélico	Aprovado
Instituição / Affiliation: EESC **-** SAA	
Fernando Henrique Tadashi Himeno	Aprovado
Instituição / Affiliation: **-**	

Presidente da Banca / Chair of the Examining Committee:

_____

Professor Associado Marcello Augusto Faraco de Medeiros
(assinatura / *signature*)

USPAssina - Autenticação digital de documentos da USP

Registro de assinatura(s) eletrônica(s)

Este documento foi assinado de forma eletrônica pelos seguintes participantes e sua autenticidade pode ser verificada através do código 61MJ-T2VE-H7XE-B9F3 no seguinte link: https://portalservicos.usp.br/iddigital/61MJ-T2VE-H7XE-B9F3

Marcello Augusto Faraco de Medeiros

Nº USP: 2915414

Data: 04/07/2024 12:28

Ricardo Afonso Angélico

Nº USP: 3691286

Data: 04/07/2024 13:06

Fernando Henrique Tadashi Himeno

Nº USP: 7592992

Data: 04/07/2024 13:40

RESUMO

Nascimento, F. D. Análise comparativa de softwares para simulação de camada limite sobre uma superfície com irregularidades. 2024. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2024.

Este trabalho busca comparar dois programas de simulação numérica de escoamento, sendo um deles desenvolvido internamente pelo grupo de pesquisa do Professor Medeiros e o outro, o *Nek5000*. O escopo da comparação é a facilidade de uso, a precisão e o desempenho na solução dos fenômenos estudados em seu laboratório, visando estudar a qualidade do código escrito internamente e a possibilidade da introdução de outro *software* para simulação. O caso de um escoamento de baixo Mach sobre uma placa plana semi-infinita com uma cavidade retangular foi escolhido como cenário de teste para a comparação, por ser bom representante de um fenômeno de geração de arrasto e ruído, além de ser simples de se descrever e analisar. A malha e as condições de contorno foram replicadas entre os programas, ajustadas de acordo com o formato de descrição do problema exigido por cada um, com um esforço para que se mantivessem idênticas. A simulação foi bem sucedida para o programa *in-house*, em que se pode observar a formação de um escoamento similar àquele apresentado em Mathias e Medeiros (2019) – um estudo que descreve esse programa e o emprega na simulação de um caso similar – e mal sucedida para o *Nek5000*, o qual falhou em convergir a um resultado em diversas simulações testadas. Foi possível avaliar a facilidade de uso e comparar as capacidades de cada programa, embora a comparação de desempenho tenha sido prejudicada em razão desta falha.

Palavras-chave: dinâmica dos fluidos computacional. camada limite. escoamento sobre cavidade. método dos elementos espectrais.

ABSTRACT

Nascimento, F. D. Análise comparativa de softwares para simulação de camada limite sobre uma superfície com irregularidades. 2024. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2024.

This study seeks to compare two computational fluid dynamics programs, one of them being developed by Professor Medeiros' research group, and *Nek5000*. The comparison criteria is ease of use, results precision and computational performance for the simulation of phenomena studied in his laboratory. It aims to assess the quality of the in-house code compared to external software and evaluate the advantages of introducing a new piece of software for running simulations. A scenario of a low-Mach number flow over a semi-infinite plate with a rectangular cavity was chosen as basis for the comparison, since it's a representative sample of the types of drag and noise simulations studied there, and given its simplicity. The mesh and boundary conditions where replicated across the programs, with appropriate adjustments for each program's input formats, with focus on keeping them as close as possible. The simulation was successful using the in-house software, which resulted in a flow similar to the one found in a published review of this tool's results (MATHIAS; MEDEIROS, 2019). It was unsuccessful with *Nek5000*, which failed to converge to a result over a series of attempts. A comparison was made between the use of use and the capabilities of each of them, but no conclusion could be drawn about their relative performances.

Keywords: computational fluid dynamics. boundary layer. flow over gap. spectral elemental method.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A simulação numérica de escoamentos é parte fundamental da Aerodinâmica experimental. Esta simulação pode ter como objetivo selecionar casos relevantes para um experimento a ser montado, buscar condições de escoamento próprias para o estudo de um fenômeno ou avaliar o impacto da variação do parâmetros do escoamento sobre seu comportamento.

Para que os resultados da simulação possam ser utilizados de forma confiável, é necessário que se verifique sua concordância com dados reais do fenômeno a ser estudado. A partir destes dados é possível assegurar a adequação daquele código para situações similares.

A equipe do Professor Medeiros desenvolveu para o uso em seu laboratório um programa capaz de computar um escoamento por meio de uma simulação numérica direta das equações de Navier-Stokes (MATHIAS, 2021). Ele incorpora um modelo que inclui a compressibilidade do fluido, um método de Runge-Kutta de quarta ordem para variação do tempo, a derivação espacial de quarta ordem como em solucionadores espectrais e a filtragem passa-baixa espacial e temporal. Além disso, oferece a geração de refinamento da malha e a criação de zonas de *buffer* nos contornos, entre outras funções. O programa é capaz de computar seus resultados em paralelo pela decomposição do domínio.

Essa equipe também busca estudar a disponibilidade de programas externos que cumpram funções similares às do programa desenvolvido internamente. Esses programas devem ser considerados em termos das funções oferecidas, da facilidade de uso, da rapidez de processamento e da qualidade dos resultados numéricos.

O programa a ser estudado neste trabalho é o *Nek5000*, um solucionador de escoamentos bidimensional, tridimensional e axissimétrico baseado no método dos elementos espectrais. Ele é capaz de simular escoamentos não-estacionários, com convecção natural e forçada, em regime compressível e incompressível, e geometrias de malha variáveis no tempo. Também oferece modelos para baixo Mach e para cenários envolvendo eletromagnetismo.

O cenário escolhido para a comparação foi elaborado a partir da análise da linha de pesquisa desse laboratório, relacionada a ruído e excrescências superficiais. Foi elaborado um modelo de uma placa plana com cavidade, a baixo Mach, já verificado para o programa interno. Com isso, busca-se testar a capacidade do *Nek5000* de obter bons resultados em

um escoamento bem conhecido e relevante para as pesquisas desse laboratório.

2 DESCRIÇÃO DO CASO SIMULADO

2.1 Configuração do caso

A formulação do caso estudado será obtida a partir de Mathias e Medeiros (2019), o qual enuncia as condições em que o software desenvolvido no laboratório foi testado e como foi o procedimento usado para a geração de malha e a avaliação da convergência dos resultados, ilustrando seu desempenho. Em particular, será importante a primeira subseção de resultados, a qual descreve o escoamento que se espera encontrar após a simulação.

O caso a ser simulado contém uma placa plana horizontal semi-infinita imersa em um escoamento na direção de seu comprimento. Esta placa é dimensionada de forma que haja uma cavidade retangular após um comprimento definido, selecionado de modo que a camada limite tenha uma espessura de deslocamento (δ^*) que resulte em um número de Reynolds Re_{δ^*} predeterminado no início da cavidade, encontrado usando

$$\mathrm{Re}_{\delta^*} = \frac{U_\infty \, \delta^*}{\nu} \, ,$$

em que U_∞ é a velocidade do escoamento livre e ν, a viscosidade cinemática.

Segundo a formulação de Blasius, a espessura de deslocamento é

$$\delta^* = \int_0^\infty \left(1 - \frac{u}{U_\infty}\right) dy \approx 1.7208 \sqrt{\frac{\nu x}{U_\infty}} \, ,$$

em que $u = u(x, y)$ é a componente de velocidade paralela à placa. Para $\mathrm{Re}_{\delta_c^*}$, a posição de início da cavidade, x_c, com origem no bordo de ataque da placa, deve ser

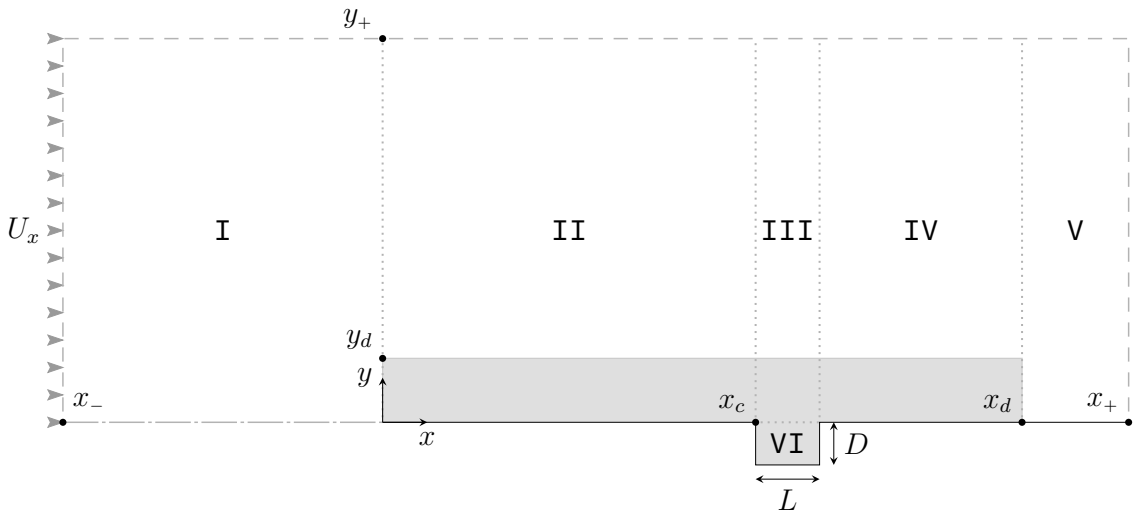
$$x_c = \delta_c^{*2} \, \frac{U_\infty}{1.7208^2 \cdot \nu} = \delta_c^* \, \frac{\mathrm{Re}_{\delta_c^*}}{1.7208^2} \, ,$$

a qual é trivialmente adimensionalizada em termos de δ_c^*, a espessura de deslocamento no início da cavidade.

A escolha do número de Reynolds serviu, em Mathias e Medeiros (2019), para que se pudesse observar a fronteira entre regimes de escoamento estáveis e instáveis. Neste trabalho, a escolha serve para verificar a capacidade do *Nek5000* de produzir resultados comparáveis para uma combinação de parâmetros já estudada pelo laboratório.

A geometria da placa está ilustrada na figura 1. Nela, a região cinza indica o domínio físico do problema e a região completa demarca o domínio de simulação. As regiões numeradas representam zonas retangulares do domínio, que serão importantes durante a

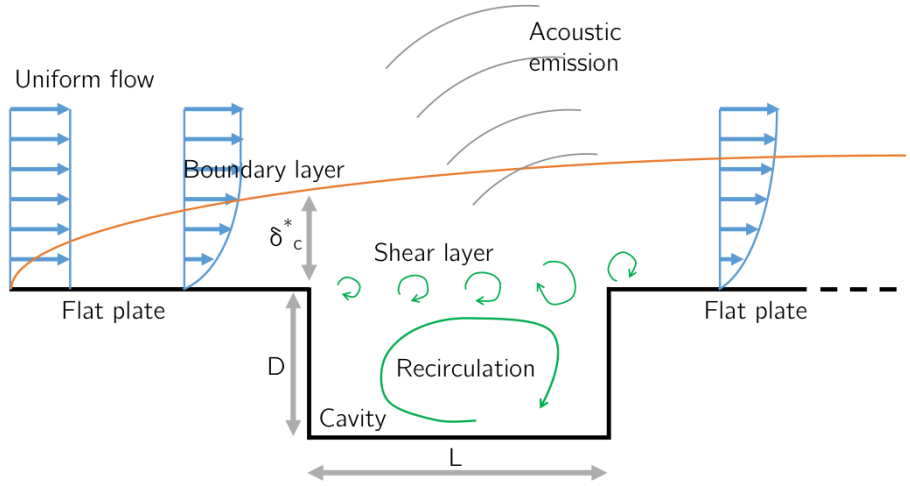
Figura 1 – Esquemático do domínio do problema



Fonte: Elaborada pelo autor

montagem da malha. A linha traço-ponto mostra uma parede com deslizamento; as setas, a fronteira com velocidade fixa; a linha tracejada, a fronteira livre; e a linha sólida, a parede sem deslizamento da placa.

Figura 2 – Esquemático do escoamento esperado



Fonte: Mathias e Medeiros (2019)

O resultado esperado é uma camada limite próxima à formulação de Blasius, porém com diferenças na espessura e com a presença de vórtices e forte recirculação dentro da cavidade, o que gera um fenômeno não-estacionário de natureza acústica, como ilustrado na figura 2. Para este trabalho, a comparação se restringirá à parte estacionária do escoamento.

2.2 Parâmetros relevantes

A definição dos parâmetros do problema foi obtida a partir do *script* de configuração `parameters.m` fornecido junto ao código do laboratório. Seus comprimentos estão adimensionalizados por δ_c^*, com a origem das coordenadas no ponto de início da placa. O cálculo de x_c, além de outros para dimensionar as bordas do domínio a uma distância razoável da camada limite, são usados pelo *script* para gerar uma malha adimensional.

No cenário estudado, a espessura da camada limite no início da cavidade é de $\delta^* = 1\,\mathrm{mm}$, com número de Reynolds $\mathrm{Re}_{\delta_c^*} \approx 1600$. Outros parâmetros relacionados ao escoamento estão apresentados na tabela 1. A cavidade do cenário estudado possui comprimento $L = 20\,\delta_c^*$ e profundidade $D = 4\,\delta_c^*$. Os parâmetros geométricos derivados dessa combinação de Reynolds, formato de cavidade e espessura da camada limite estão mostrados na tabela 2, cujos valores foram adimensionalizados usando como referência δ_c^*.

Tabela 1 – Parâmetros de escoamento para a simulação

Grandeza	Símbolo	Valor	Unidade
Velocidade no *inlet*	U_x	22.6	$\mathrm{m/s}$
Viscosidade dinâmica	μ	1.8444×10^{-5}	$\mathrm{Pa\,s}$
Temperatura	T	299.05	K
Densidade	ρ	1.1845	$\mathrm{kg/m^3}$
Capacidade térmica volumétrica	$\rho\, C_\mathrm{p}$	1.1820×10^3	$\mathrm{J/(K\,m^3)}$
Condutividade térmica	k	2.6036×10^{-2}	$\mathrm{W/(K\,m)}$

Fonte: Elaborada pelo autor

Tabela 2 – Parâmetros geométricos do domínio

	Grandeza	Símbolo	Valor/δ_c^*
Cavidade	Comprimento	L	20
	Profundidade	D	4
	Início	x_c	474.5
Domínio físico	Limite a jusante	x_d	800
	Limite superior	y_d	12
Domínio de simulação	Limite a montante	x_-	-1176.8
	Limite a jusante	x_+	1895.9
	Limite superior	y_+	85.916

Fonte: Elaborada pelo autor

3 CONSIDERAÇÕES SOBRE A MONTAGEM E A SIMULAÇÃO DO CASO

3.1 Visão geral dos programas

Uma comparação rápida entre as capacidades de ambos os programas estudados está apresentada na tabela 3. A discussão detalhada sobre as opções disponíveis em cada um deles e as escolhas tomadas na montagem será feita a seguir.

Tabela 3 – Comparação de funções oferecidas pelos programas

Funcionalidade	*Nek5000*	*in-house*
Formulações	Incompressível e compressível	Compressível
Integração temporal	Fórmula de diferenciação retrógrada (BDF), até a 3ª ordem	Runge-Kutta de 4ª ordem
Derivação espacial	Espectral, até 16ª ordem	4ª ordem similar a espectral
Filtragem temporal	Passa-baixa ou passa-alta no domínio completo	Passa-baixa com ajuste fino de regiões
Filtragem espacial	—	Passa-baixa com ajuste fino de regiões
Condições inicial e de contorno	Personalizáveis por *scripts* de usuário em *Fortran*	Personalizáveis por novas definições em *Matlab*
Geração de malha	Retangulares e deformadas, ou geradas externamente; 2D, 3D e axissimétricas	Retangulares 2D e 3D
Formato dos resultados	Arquivo binário próprio	Arquivos `.mat` ou *HDF5*

Fonte: Elaborada pelo autor

3.2 Modelos de simulação

O funcionamento do *Nek5000* e seus princípios de operação são descritos no manual de uso (Nek5000, 2023). Para o caso escolhido, a seção teórica do manual indica o modelo de baixo Mach para Navier-Stokes, o qual também requer adicionalmente a consideração das propriedades térmicas do fluido.

O modelo de baixo Mach se baseia numa simplificação da equação de Navier-Stokes, separando os termos de temperatura e velocidade da definição de densidade, o que

possibilita que a equação seja resolvida de forma análoga ao caso incompressível, com menor erro de continuidade, enquanto preserva a compressibilidade (TOMBOULIDES; LEE; ORSZAG, 1997).

A escolha deste modelo de equações limita o *Nek5000* à formulação $\mathbb{P}_n - \mathbb{P}_n$ dos elementos espectrais. Nela, a velocidade é primeiro calculada em termos de um termo convectivo; depois, o laplaciano da pressão é computado a partir da convecção; e, por fim, a velocidade é atualizada para levar em conta o gradiente de pressão e o termo viscoso. Esta restrição não está listada diretamente na documentação, mas está explicada nas listas de *email* de suporte ao usuário (Nek5000-users, 2017). Além disso, o cálculo do erro passa a ser ligado a resolução espacial e resulta em valores muito diferentes, o que dificulta a comparação de resultados em função das tolerâncias.

É importante ressaltar que o *Nek5000* possui grandes deficiências na documentação, em especial sobre funções definidas pelo usuário e arquivos de entrada no formato antigo. Muitas vezes, a documentação não descreve o impacto das opções de valores para um parâmetro, ou deixa de alertar incompatibilidades. Grande parte das informações disponíveis sobre o programa estão disponíveis como exemplos de caso ou discussões em listas de *email*, no lugar de um guia organizado.

Não é simples, ademais, buscar entender o funcionamento do programa pela leitura do código, por usar um padrão antigo de *Fortran*, o *Fortran 77*, repleto de más práticas programação – como, por exemplo, o extenso uso de blocos `COMMON` para compartilhamento de memória global entre rotinas, o que é desaconselhado desde *Fortran 90*, há mais de 30 anos (METCALF et al., 2024). Isso torna extremamente difícil clarificar o que não estiver bem descrito no manual de uso. Sendo assim, é importante considerar se o caso estudado já possui um exemplo similar disponível para o *Nek5000* antes de decidir usá-lo para uma simulação.

As formulações fornecidas por esse programa podem ser utilizadas tanto em sua forma dimensional quanto em sua forma adimensional. Porém, a documentação não deixa claro como fazer para adimensionalizar cada parâmetro de forma compatível com as equações. Novamente, é preciso recorrer a listas de *email*, fóruns e notas de palestras relacionadas ao programa. Algumas formulações adimensionais estão descritas na seção de teoria do manual (Nek5000, 2023).

O *Nek5000* possui um mecanismo de passo temporal variável para a integração, exigindo

um valor para o número de Courant que será usado para avaliar a condição de Courant--Friedrichs-Lewy. Para isso, basta especificar o valor alvo e o passo máximo permitido. Este mecanismo também está implementado no código do laboratório.

Há também filtros de estabilização temporal para o escoamento, tanto passa-baixa quanto passa-alta, definidos com parâmetros fornecidos pelo usuário. Estes filtros são necessários para a simulação turbulenta, caso contrário não há garantia de estabilização do resultado, embora diminuam sua exatidão, uma vez que atenuam fenômenos não--estacionários. Mais grave é a exigência de que os filtros sejam aplicados no domínio completo, acentuando a perda de exatidão.

Veja também que não há filtro espacial como aquele implementado no programa *in--house*. Esse filtro possui melhor desempenho no caso simulado, uma vez que estabiliza o resultado sem alterá-lo em excesso, melhorando sua fidelidade com o escoamento real, que exibe um fenômeno acústico. Além disso, pode ser aplicado apenas na região desejada, diminuindo seu impacto no escoamento completo.

Essa falta de opções para filtragem obriga que a simulação utilize valores altos de filtros temporais que, ainda assim, podem não ser suficientes para garantir a convergência, como será mostrado na discussão de resultados.

As simulações em *Nek5000* podem ser customizadas com a adição de rotinas definidas pelo usuário, as quais são executadas em momentos específicos de acordo com seu nome. Essas rotinas podem ser usadas para a inicialização, para a definição de variáveis globais e para a modificação de variáveis ponto a ponto.

Assim como o programa do laboratório, o *Nek5000* consegue particionar o domínio da simulação para resolvê-lo em paralelo. Ambos utilizam o *MPI* (Message Passing Interface Forum, 2023) como interface para comunicação entre processo, o que torna o procedimento de execução muito similar. A carga computacional é, em ambos, distribuída entre núcleos de CPU. Há também uma versão em desenvolvimento que permite o uso de GPUs como núcleos de computação (FISCHER et al., 2022), a qual parece ser mais atualizada e bem documentada.

Para uma visão mais detalhada dos outros modos de operação do *Nek5000*, pode-se consultar Saha, Biswas e Nath (2021). Além disso, a seção de referência do manual de funcionamento possui uma lista extensa de trabalhos em que o código está baseado.

3.3 Geração de malha

A geração de malha no *Nek5000* é simples. Há apenas duas opções para criá-las: um gerador de malhas retangulares e um conversor de malhas externas criadas pelo *gmsh* (GEUZAINE; REMACLE, 2009) ou no formato EXODUS (MILLS-CURRAN; GILKEY; FLANAGAN, 1988).

As malhas utilizadas no *Nek5000* podem ser alteradas e deformadas durante a simulação, assim como as condições de contorno e as propriedades do fluido. Porém, as modificações precisam ser escritas em *Fortran*, dependendo de funções pouco documentadas do código. Deste modo, as customizações disponíveis dependem principalmente da familiaridade do usuário com o programa e de sua proficiência na linguagem.

O programa também oferece malhas periódicas, em que as variáveis de escoamento de duas regiões compatíveis do contorno são interligadas. Para isso, as regiões devem estar discretizadas da mesma forma.

É possível sobrepor malhas distintas durante a simulação, por interpolação das propriedades, também utilizando *Fortran*. Esta função pode ser usada para refinar certas regiões da malha, embora exija duas simulações executadas em paralelo, com arquivos de saída separados. Para que isso seja configurado, é necessário descrever em *Fortran* a interpolação das condições de contorno das malhas.

Também é possível segmentar a malha em trechos de composição distinta, de forma que partes do fluido tenham propriedades distintas, ou que haja uma interface sólida. Esta função pode ser usada para modelar escoamentos multifásicos e sistemas termodinâmicos com trechos sólidos.

Felizmente, as malhas necessárias para o cenário deste trabalho são retangulares e, por isso, é suficiente utilizar apenas o gerador retangular fornecido. Mesmo malhas complexas podem ser feitas com esse procedimento, desde que o usuário esteja disposto a programar a conformação dos elementos.

Uma vez que a discretização espacial apresentada em Mathias e Medeiros (2019) mostrou convergência adequada, ela será tomada como base para as simulações no *Nek5000*. Esta escolha facilitará a comparação direta dos resultados e diminuirá a necessidade de um estudo de convergência novamente.

A ordem dos elementos da malha para o *Nek5000* é, idealmente, por volta de $N = 7$ (Nek5000, 2023). De fato, o programa parece favorecer a escolha de ordens mais altas

no lugar do refinamento simples da malha. Os elementos disponíveis são conformais em formato retangular e hexagonal. Há também um limite para o número máximo de elementos em 350000, embora seja possível ultrapassar o limite com uma configuração mais complexa.

Para a malha fornecida com o programa *in-house*, a ordem recomendada é excessiva em relação à discretização já presente. Desta forma, será usado um polinômio de menor ordem que se aproxime da resolução utilizada originalmente.

3.4 Processamento dos resultados

Diferentemente das etapas anteriores, o processamento dos dados de saída é profundamente diferente entre os dois programas. O principal motivo para isso é o uso pelo *Nek5000* de um formato exótico para armazenar os resultados, antigo e pouco documentado, além da complexidade de procedimentos para obter dados adicionais. Seus arquivos de saída, portanto, são reconhecidos por poucos visualizadores, embora sua licença aberta permita que um leitor seja adicionado a outros programas.

O formato utilizado é compatível com o *VisIt*, um programa de visualização capaz de lidar com grandes quantidades de dados temporais, com capacidade de lê-los em uma ampla gama de formatos. Porém, esse visualizador tem incompatibilidades em alguns sistemas e é distribuído por métodos de instalação complexos em outros, tornando complexa sua utilização. Uma alternativa é o *Paraview*, que também contém um leitor para esse tipo de arquivo. Ambos os visualizadores possuem a capacidade de lidar com arquivos grandes em paralelo e com carregamento sob demanda dos dados.

Enquanto o programa do laboratório cria resultados acessíveis em *Matlab* e *Octave*, onde podem ser manipulados com as ferramentas comuns dessas linguagens, o formato do *Nek5000* força o usuário a utilizar um dos visualizadores citados e manipular os dados a partir deles. Pode-se considerar que, de certo modo, esta complicação pode ser positiva dado que a visualização será feita por ferramentas otimizadas para isso. Ainda assim, será necessário utilizar os aparatos próprios destes programas para manipular as informações de saída.

Para computar propriedades a partir dos resultados do *Nek5000*, há duas opções. A primeira consiste em preparar rotinas adicionais definidas no arquivo `.usr`, em *Fortran*, para que sejam executadas durante a simulação. Com isso, é possível manipular a malha e as variáveis, operando sobre elas, e exportar os resultados, utilizando alguma das funções

que o *Nek5000* oferece ou criando uma forma personalizada de salvar os dados.

A segunda forma depende dos programas de visualização para operar sobre os dados. Nesse caso, é necessário assegurar que o *Nek5000* armazenará todas as informações necessárias nos arquivos de simulação. No caso do *Paraview*, o trabalho sobre os dados é feito em *pipelines* de processamento e visualização, as quais aplicam sobre eles fontes, filtros, transformações e outras operações.

4 ELABORAÇÃO DOS ARQUIVOS DE CASO

Os arquivos de caso para o programa *in-house* fornecidos pelo laboratório possibilitaram a geração da malha e a simulação do caso sem necessidade de ajustes. Sendo assim, apenas foi necessário adaptar os dados para os arquivos de entrada do *Nek5000*.

Como comentado anteriormente, o ponto de partida de uma simulação no *Nek5000* é a busca por casos parecidos, que possam servir de base para a elaboração de uma nova malha. O cenário deste trabalho possui apenas um exemplo similar, simulando uma camada limite de Blasius.[1]

Porém, o código usado é complexo demais para ser aproveitado para a nova simulação. Há nele diversas funções complexas que configuram o estado inicial dos elementos para que iniciem com o perfil já formado, além de emitir dados no instante inicial, tornando-o mais extenso do que o necessário. Além disso, vale lembrar que, caso haja um erro nessa configuração, não é simples determinar sua raiz, uma vez que o processo de criação das condições iniciais e de contorno ocorre durante a execução do programa, sem que seja possível inspecionar visualmente as condições impostas. Dessa forma, os erros só serão descobertos após a execução.

Por esse motivo, os arquivos de entrada para este caso foram criados do início, para que fossem mantidos simples e claros. Foram usados como base apenas dois arquivos distribuídos junto ao *Nek5000*, `SIZE.template`, o qual possui informações básicas para compilação, e `zero.usr`, com rotinas vazias listando as personalizações disponíveis.

O primeiro arquivo a se construir é o `caseC.box`, responsável por descrever a malha para o gerador `genbox`. Este arquivo possui um formato simples, uma vez que apenas permite combinar diversos retângulos para formar uma malha. Os componentes da malha foram selecionados de forma que cada mudança nas condições de contorno ocorresse na divisão entre retângulos. Essas divisões já foram apresentadas na figura 1.

Para reproduzir a malha original usando apenas os parâmetros geométricos do caso, seria necessário também extrair os atratores utilizados para o refinamento e imitar o gerador de malha do programa original, uma vez que a distribuição de nós não é uniforme. No lugar disso, é preferível importar a malha como está, para evitar a introdução de erros na conversão.

[1] <https://github.com/Nek5000/NekExamples/tree/master/blasius>

O *script* `script/make_box_mesh.m`, listado no Apêndice A, lê um arquivo de malha e extrai sua geometria, presumindo que a malha tenha o formato esperado de uma placa plana com cavidade. Para a malha fornecida com o caso estudado, a geometria resultante é composta por 6 retângulos, cujos dados estão dispostos na tabela 4, em que x e y são as posições dos nós e N_e é o número de elementos em cada face. Os índices listados relacionam cada região a sua zona correspondente na figura 1 e o comprimentos foram calculados a partir das medidas indicadas na tabela 2.

Tabela 4 – Parâmetros geométricos adimensionais dos componentes da malha

Região	Índice	x			y		
		x_0	x_n	N_e	y_0	y_n	N_e
`pre_buffer`	`I`	-1176.8	0	97	0	85.916	82
`leading_boundary_layer`	`II`	0	474.5	251	0	85.916	82
`boundary_layer`	`III`	474.5	494.5	276	0	85.916	82
`trailing_boundary_layer`	`IV`	494.5	800.0	461	0	85.916	82
`post_buffer`	`V`	800.0	1895.9	60	0	85.916	82
`cavity`	`VI`	474.5	494.5	276	-4	0	58

Fonte: Elaborada pelo autor

Para que a malha seja completamente descrita, é necessário fornecer condições de contorno que conectam os retângulos entre si ou impõem sobre faces externas alguma condição determinada. Esses dados do contorno estão exibidos na tabela 5, representados usando as abreviações de até 3 letras convencionadas pelo *Nek5000*. As arestas são nomeadas em relação à sua posição relativa ao centro do retângulo – isto é, $-x$ para a aresta esquerda, $+x$ para a direita e analogamente para y.

Tabela 5 – Condições de contorno aplicadas à malha

Região	Aresta			
	$-x$	$+x$	$-y$	$+y$
`pre_buffer`	`v`	`E`	`SYM`	`O`
`leading_boundary_layer`	`E`	`E`	`W`	`O`
`boundary_layer`	`E`	`E`	`E`	`O`
`trailing_boundary_layer`	`E`	`E`	`W`	`O`
`post_buffer`	`E`	`O`	`W`	`O`
`cavity`	`W`	`W`	`W`	`E`

Fonte: Elaborada pelo autor

A tabela 6 relaciona cada abreviação com seu tipo correspondente. Note que, para serem utilizados, devem sempre ser tratados como campos de 3 letras alinhados à esquerda, sendo por vezes necessário preencher as letras faltantes à direita com espaços. A condição de velocidade (`v`) é uma condição essencial simples em que se fornece, a cada instante de tempo, uma velocidade $\vec{u} = [u_x, u_y, u_z]$, podendo ser definida em função das coordenadas ou de outras variáveis no ponto. A saída aberta (`O`) é uma condição natural que permite que o escoamento a atravesse livremente, sem um gradiente de pressão. A fronteira interna (`E`) apenas serve para alertar quando alguma face interna não se uniu a outra face interna de outro retângulo. As fronteiras de parede (`W` e `SYM`) anulam a componente da velocidade normal à superfície, embora a primeira anule componentes tangenciais enquanto a segunda os preserva. Condições representadas por letras minúsculas exigem o fornecimento de parâmetros adicionais por meio do arquivo `.usr`.

Tabela 6 – Abreviações de condições de contorno usadas pelo *Nek5000*

Sigla	Tipo	Condição
`v`	Dirichlet	Velocidade definida pelo usuário
`O`	Neumann	Saída aberta
`E`	–	Fronteira interna
`W`	Dirichlet	Parede sem escorregamento
`SYM`	Mista	Parede com escorregamento

Fonte: Nek5000 (2023)

Com estes dados em mãos, `script/make_box_mesh.m` gera um arquivo `caseC.box`, o qual é dividido em 3 partes, ilustradas a seguir com trechos de um arquivo real. Uma explicação completa está disponível na documentação (Nek5000, 2023).

Primeiro, há um prefácio indicando um arquivo *template* de `.rea`, o formato antigo e textual da malha com parâmetros e elementos juntos. Esta linha só é necessária caso a saída também seja uma malha `.rea`.

```
skeleton.rea
```

Em seguida, indica-se o número de dimensões (2 ou 3). Caso o número de dimensões seja escrito como negativo, *Nek5000* interpreta-o como positivo e altera o formato de saída para `.re2`, uma malha binária.

```
2      2D (negative for binary output)
```

Depois, segue o número de campos a serem computados: `1` para incluir apenas o campo de velocidade e `2`, para incluir o de temperatura também.

```
1      number of fields
```

A partir daí, usa-se o formato

```
box_id
nx ny
X1 X2 X3 ...
Y1 Y2 Y3 ...
AAA,BBB,CCC,DDD
```

em que `box_id` é um nome arbitrário do retângulo; `nx` e `ny` são números de nós na respectiva direção; `Xi` e `Yi` são as posições ordenadas dos nós; e `AAA,...` são siglas de 3 letras para a condição de contorno. Este bloco é repetido até que todos os retângulos sejam declarados.

O leitor atento deve ter notado que a saída usada foi a textual. Isso ocorreu porque, para modelos com muitos elementos, a saída binária falha nos próximos passos de conversão, por conta de um erro no particionador de malha que permanece no *Nek5000* desde 2018.[2] Por qualquer caminho, entretanto, o resultado será idêntico. O arquivo `skeleton.rea` contém uma malha pequena com alguns parâmetros a qual é sobrescrita pela malha gerada. Daí, basta converter esta malha para binário com `reatore2`, resultando em `caseC.re2`.

Com esse arquivo, deve-se gerar um mapa de particionamento com `genmap`. O arquivo resultante `caseC.ma2` será usado para dividir o domínio entre processadores.

Finalmente, basta buscar os parâmetros da simulação necessários para a equação. Nesta simulação, será usada a formulação dimensional do problema, fornecendo os dados já apresentados na tabela 1. Todas estas informações são incluídas em `caseC.par`. Além disso, `caseC.usr` precisa ser alterado para que as funções `useric`, que determina condições iniciais, e `userbc`, que determina condições de contorno a cada passo temporal, atribuam corretamente os valores de velocidade e temperatura fornecidos no arquivo. Também foi exigido um número de Courant próximo a 0.5, o máximo que pode ser pedido para este tipo de modelo.

[2] <https://github.com/Nek5000/Nek5000/issues/562>

Finalmente, deve-se ajustar o arquivo `SIZE` para que leve em conta que esta malha tem aproximadamente 110000 elementos, atribuindo este valor ao parâmetro `lelg`. No mesmo arquivo, para diminuir a ordem dos polinômios, altera-se `lx1` para o número de pontos por elemento, ou seja, $N + 1$, em que N é a ordem do polinômio.

Os arquivos necessários para simular o caso estão disponíveis no Apêndice A, com exceção dos dados originais, os quais são binários e precisam ser obtidos por outro meio. Se os arquivos `*.par`, `*.usr`, `*.ma2`, `*.re2` e `SIZE` estiverem no mesmo diretório, basta utilizar `makenek` para que o caso seja compilado. Um executável nomeado `nek5000` será criado.

Um `Makefile` foi criado com receitas para todos os passos descritos, até a compilação. Para utilizá-lo, use o comando `make all-cases` no mesmo diretório do `Makefile`. Seu resultado estará em `build/case/nek5000`.

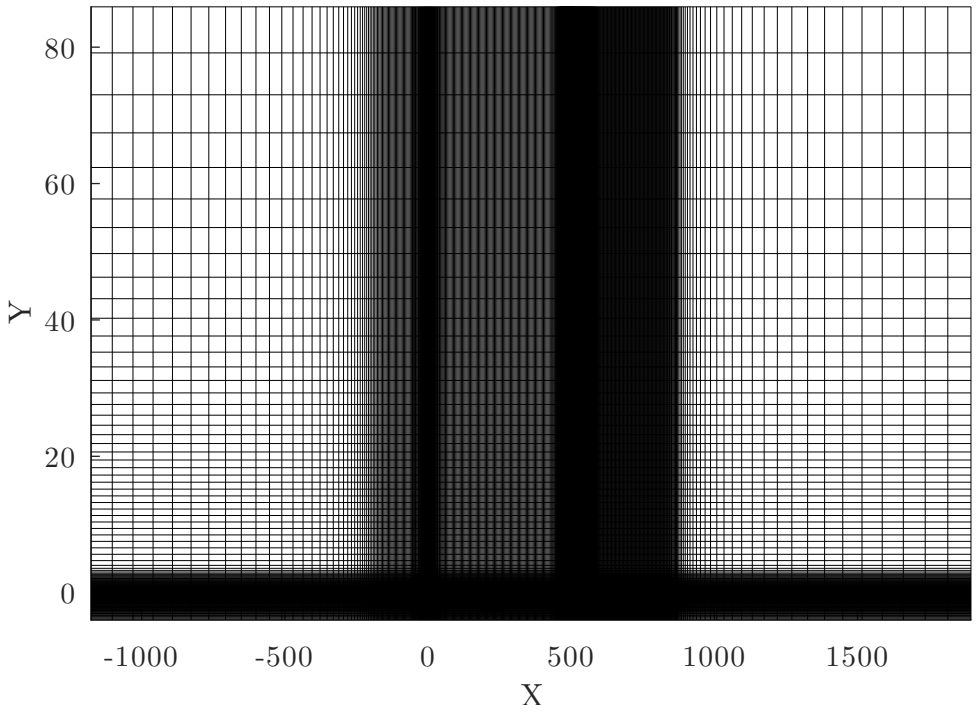
Note que a compilação pode falhar no momento do *linking* dos objetos compilados. Isto ocorre porque o *Nek5000* utiliza alocação estática das matrizes, baseada nos requisitos de memória indicados no arquivo `SIZE`, fazendo com que a especificação do espaço ocupado e dos valores iniciais das variáveis estejam embutidas no executável. Essa característica pode levar a falha na montagem do executável por exigir variáveis cujo tamanho excede o limite do *linker* (por exemplo, de $2\,\mathrm{GiB}$). Tais erros são fáceis de detectar e, para que sejam mitigados, pode-se utilizar a *flag* `-mcmodel=medium`, que muda a forma como a memória é endereçada e permite tamanhos maiores, com impacto no desempenho. Também pode-se aumentar o número mínimo de *ranks* usados pelo MPI, no parâmetro `lpmin`, o que distribui melhor a memória entre processos. Para esta simulação, foi necessário usar o primeiro método.

5 RESULTADOS

5.1 Malha

A malha resultante, idêntica entre os programas, está mostrada na figura 3. Note que os trechos dentro da parede, embora exibam linhas de discretização, não possuem elementos. Note também que há duas zonas particularmente refinadas: uma delas, no bordo de ataque da placa ($x = 0$), para suavizar a mudança de condição de contorno; e a outra na vizinhança da cavidade ($x \approx 500$), para capturar os efeitos da mudança de geometria na espessura da camada limite. A figura pode ser ampliada para se observar a discretização.

Figura 3 – Rede de elementos de malha

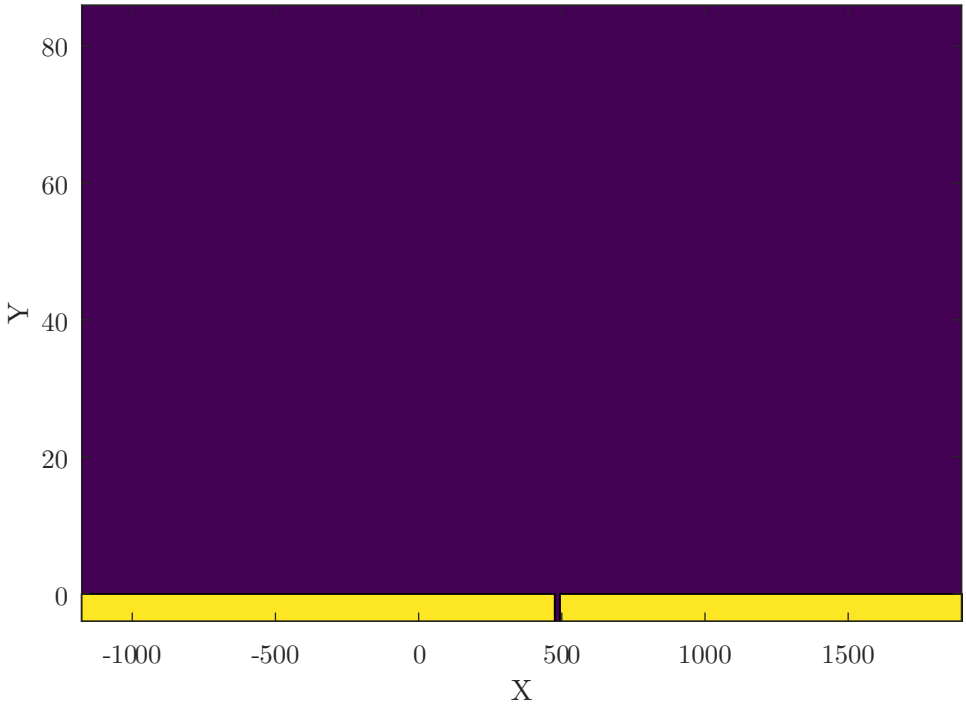


Fonte: Elaborada pelo autor

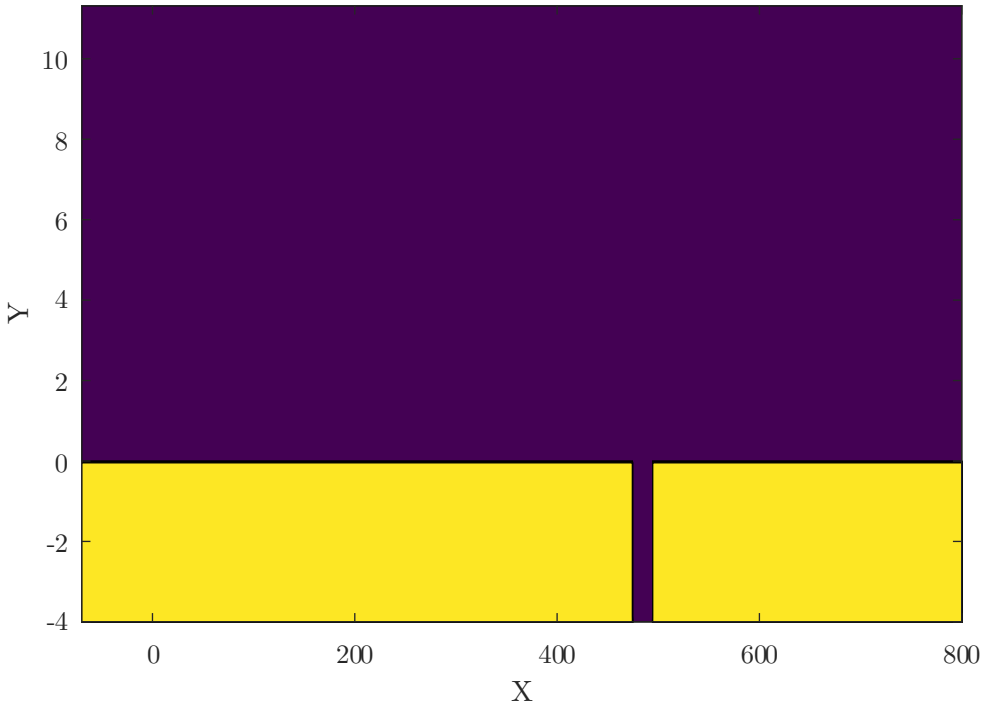
As figuras 4a e 4b ilustram o contorno da malha resultante. Nelas, as regiões em amarelo correspondem ao interior da parede e as roxas, ao fluido. Veja que a escala do gráfico distorce a cavidade, que é maior no sentido do comprimento.

Figura 4 – Contornos da malha convertida

(a) Domínio da simulação



(b) Domínio físico



Fonte: Elaborada pelo autor

5.2 Escoamento

5.2.1 Programa *In-house*

Este programa resultou no fluxo demonstrado nas figuras 5a e 5b para um intervalo de simulação de $\bar{t} = 65000$, em que $\bar{t}$ é o tempo adimensionalizado. Este fluxo é o *base flow* extraído dos resultados após este intervalo de tempo. Note que há uma zona de recirculação próxima à parede final da cavidade, além de uma zona de cisalhamento na altura da superfície da placa, gerando vórtices na interface com a camada limite acima. Também há uma zona de menor velocidade no bordo de ataque da placa, que provavelmente resulta de imprecisões numéricas ligadas à transição repentina entre condições de contorno.

Pode-se perceber que o escoamento não se altera muito na região acima da placa. Veja que a espessura de deslocamento da camada limite no início da cavidade para esse caso passou a ser

$$\delta_c^* = \int_0^\infty \left(1 - \frac{u}{U}\right) dy = 0.978\, \delta^*_{c_{\text{ref}}}$$

utilizando uma integração por partes. Algumas outras espessuras da camada limite estão listadas na tabela 7. Veja que no início da cavidade, por Blasius, $\delta^* \lessapprox 1$ por conta de um arredondamento no número de Reynolds realizado no arquivo de parâmetros `parameters.m`, o qual impactou levemente a posição do início da cavidade. A pequena variação de espessura se estende até o final do domínio, confirmando a adequação da aproximação inicial da camada limite.

Tabela 7 – Espessuras da camada limite dadas pelo programa *in-house*

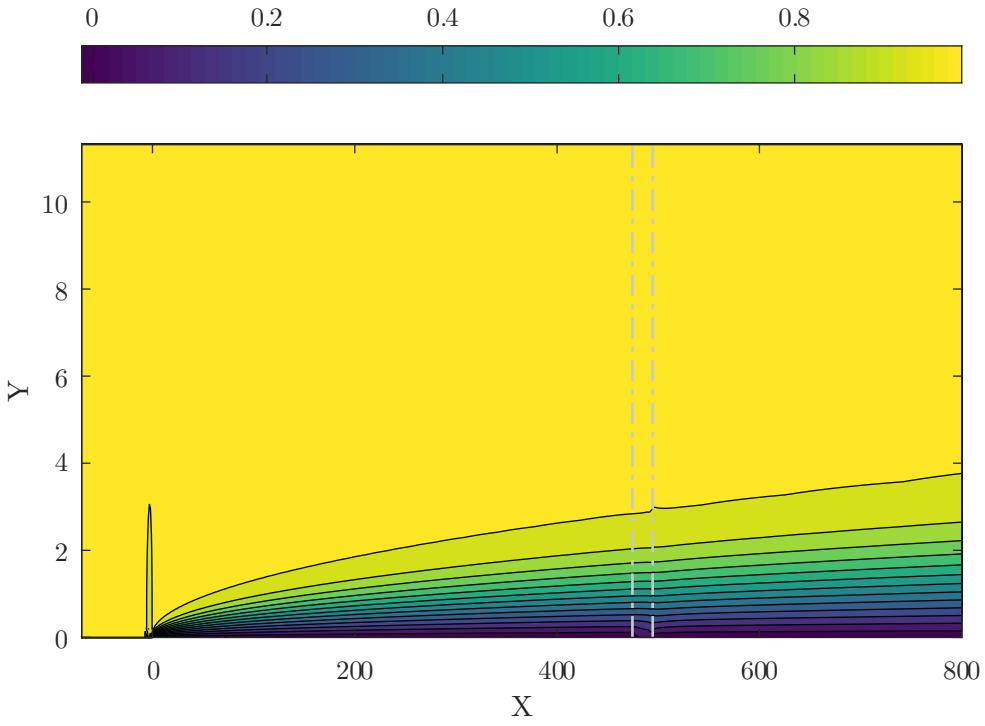
Região	Método	$\frac{\delta_{99}}{\delta^*_{c_{\text{ref}}}}$	$\frac{\delta^*}{\delta^*_{c_{\text{ref}}}}$	$\frac{\theta}{\delta^*_{c_{\text{ref}}}}$
Início da cavidade	Simulação	2.851	0.978	0.365
	Blasius	3.074	0.999	0.386
Fim da cavidade	Simulação	3.042	1.001	0.403
	Blasius	3.138	1.020	0.395
Fim do domínio	Simulação	3.042	1.015	0.400
	Blasius	3.183	1.035	0.400

Fonte: Elaborada pelo autor

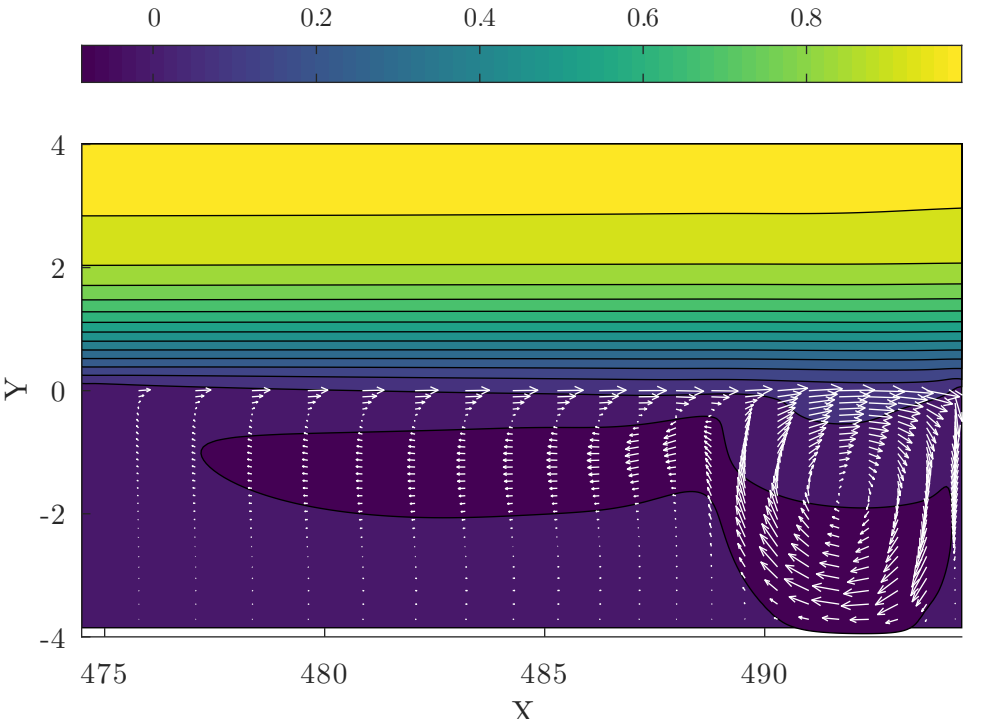
As figuras 6a, 6b e 6c mostram a evolução da camada limite ao longo da placa e as figuras 7a, 7b e 7c exibem o trecho da camada limite na vizinhança da cavidade. As linhas

Figura 5 – Contorno de velocidade total dado pelo programa *in-house*

(a) Domínio físico



(b) Detalhe da cavidade



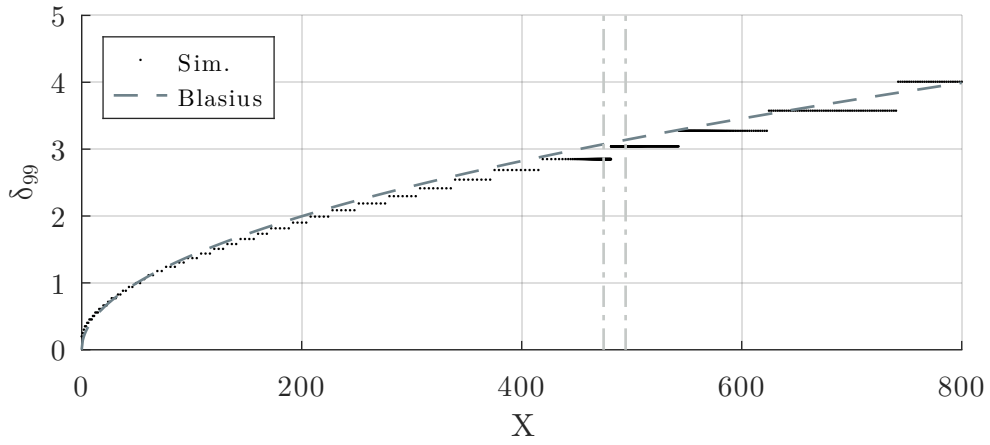
Fonte: Elaborada pelo autor

verticais mostram a posição da cavidade. É importante notar que, por conta da menor discretização na direção vertical, há pouca resolução para o cálculo de δ_{99}, resultando em degraus visíveis correspondentes às faces dos elementos.

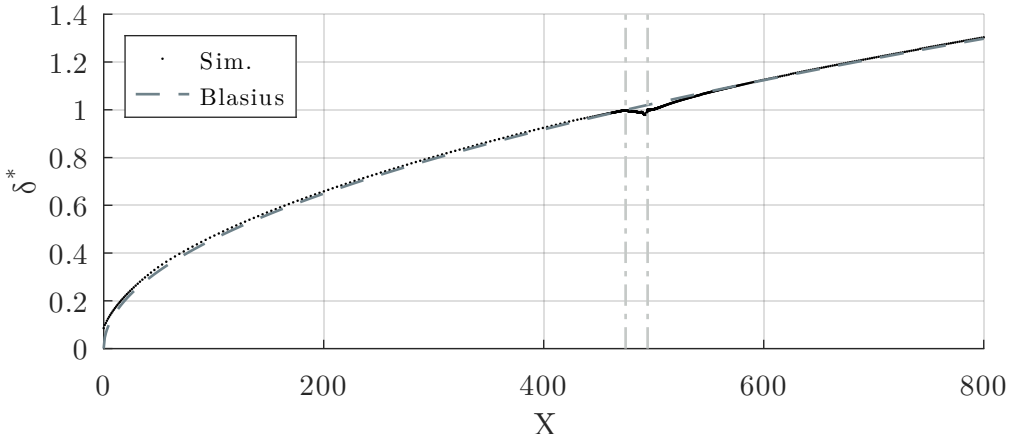
Com base nesses resultados é possível afirmar que a cavidade possui um efeito localizado na camada limite, diminuindo δ^* e aumentando θ em seu entorno, além de uma diminuição geral de sua espessura ao longo da placa. Estes resultados concordam com aqueles expostos em Mathias e Medeiros (2019) e servirão de base para a avaliação dos resultados do *Nek5000*.

Figura 6 – Evolução das camadas limite

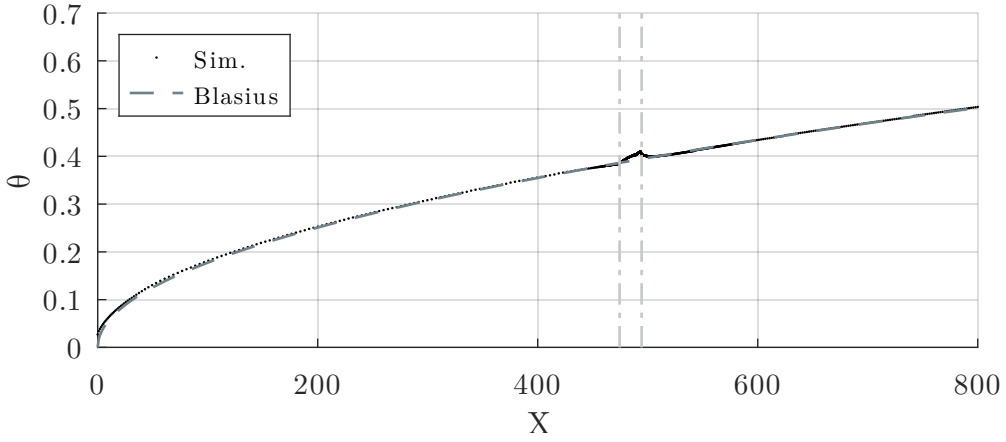
(a) δ_{99}



(b) δ^*



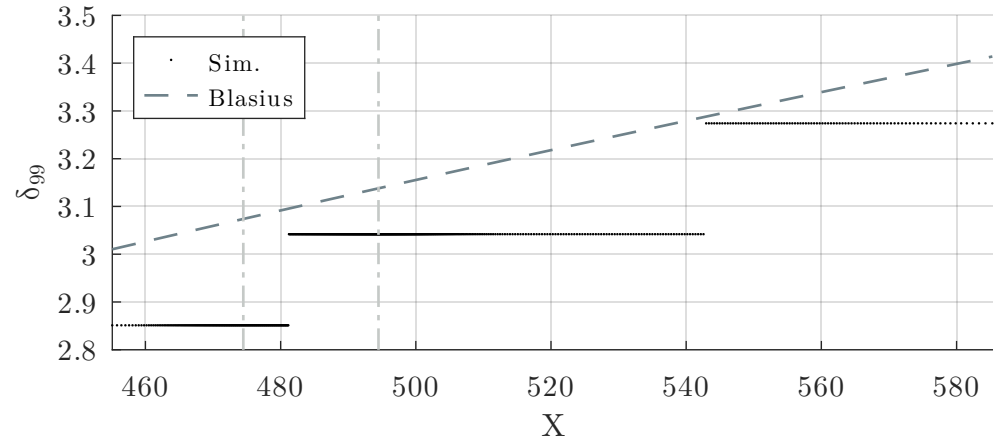
(c) θ



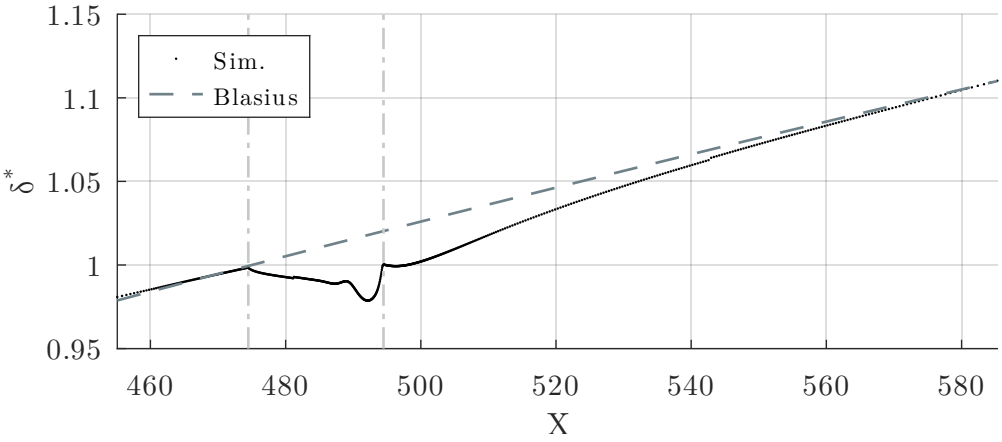
Fonte: Elaborada pelo autor

Figura 7 – Detalhe da camada limite próxima à cavidade

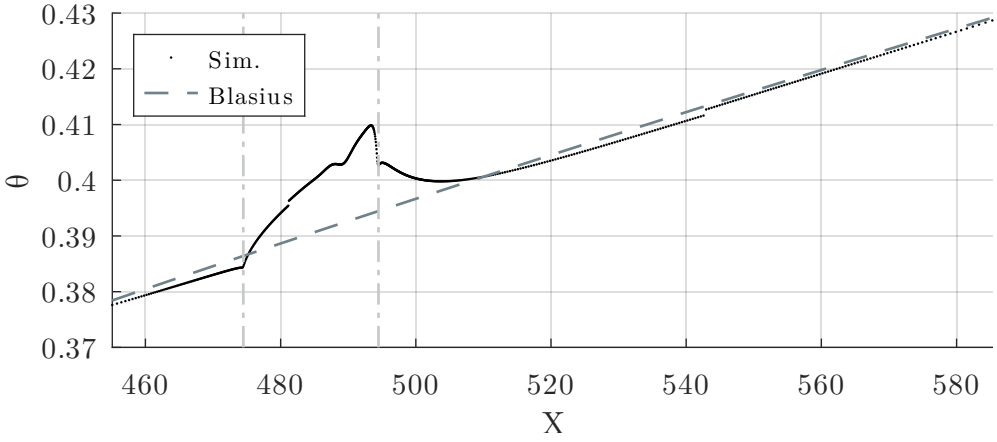
(a) δ_{99}



(b) δ^*



(c) θ



Fonte: Elaborada pelo autor

5.2.2 *Nek5000*

O modelo em *Nek5000* exigiu várias tentativas para que pudesse ser executado. Além das dificuldades em relação à documentação e à compilação do caso, a avaliação da qualidade do modelo só podia ser feita depois de permitir que a simulação fosse executada por algumas horas.

Esses fatores retardaram o desenvolvimento do caso e tornaram mais difícil a correção de erros, o ajuste dos parâmetros de entrada e a mudança de configuração do método numérico. Isso se deve ao fato de que, em geral, só é possível verificar a qualidade do modelo a ser testado tendo em mãos algum resultado intermediário da simulação, já que não há formas simples de inspecionar os arquivos de entrada emitidos pelo programa, nem visualizar as condições de contorno impostas e os parâmetros definidos.

As principais causas da demora no desenvolvimento iterativo estão ligadas à necessidade de utilizar funções menos eficientes do *Nek5000* para conduzir a comparação, como explicado anteriormente. Reiterando-as, foi necessário diminuir a ordem dos elementos e modificar o acesso à memória para que se pudesse compilar o mesmo número de elementos e replicar a malha.

Nenhum modelo em *Nek5000* conseguiu convergir para um resultado similar ao que foi apresentado anteriormente, ainda que possuíssem uma abundância de elementos, próxima ao limite do programa. Algumas das tentativas foram causadas por erros em parâmetros de entrada ou dimensões inconsistentes; outras não conseguiram alcançar um regime estacionário. Cada um dos cenários fracassados tomava, em média, várias horas de trabalho, somando o tempo para a procura e correção do erro, compilação, espera por resultados e manipulação dos resultados em um visualizador, o *Paraview*.

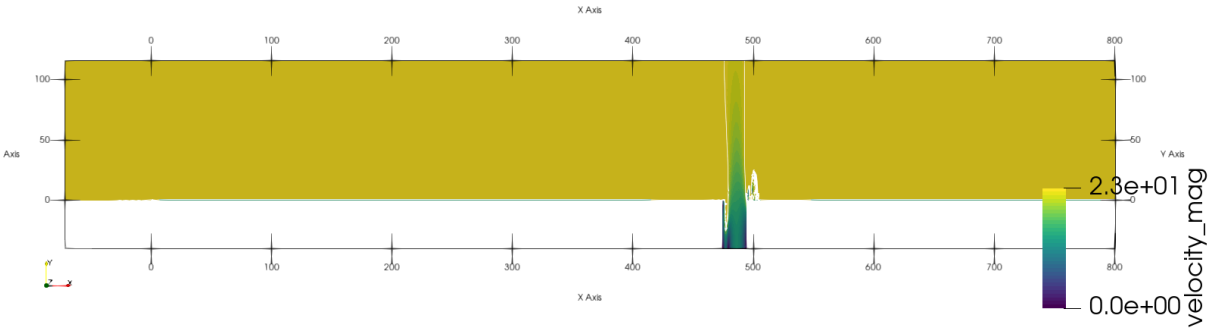
O caso utilizado nas imagens desta subseção servirá para ilustrar alguns dos problemas enfrentados durante a busca por uma simulação comparável àquela do outro programa.

A figure 8a ilustra o escoamento obtido em uma simulação do *Nek5000*, após o mesmo intervalo de tempo que os resultados anteriores, com uma tolerância equivalente. Como não houve estabilização em direção a um regime estacionário, o resultado escolhido foi o de um momento próximo ao fim da simulação. Detalhes da camada limite próxima à cavidade estão mostrados na figura 8b, em que a linha branca representa os pontos em que $U = 0.99\,U_\infty$.

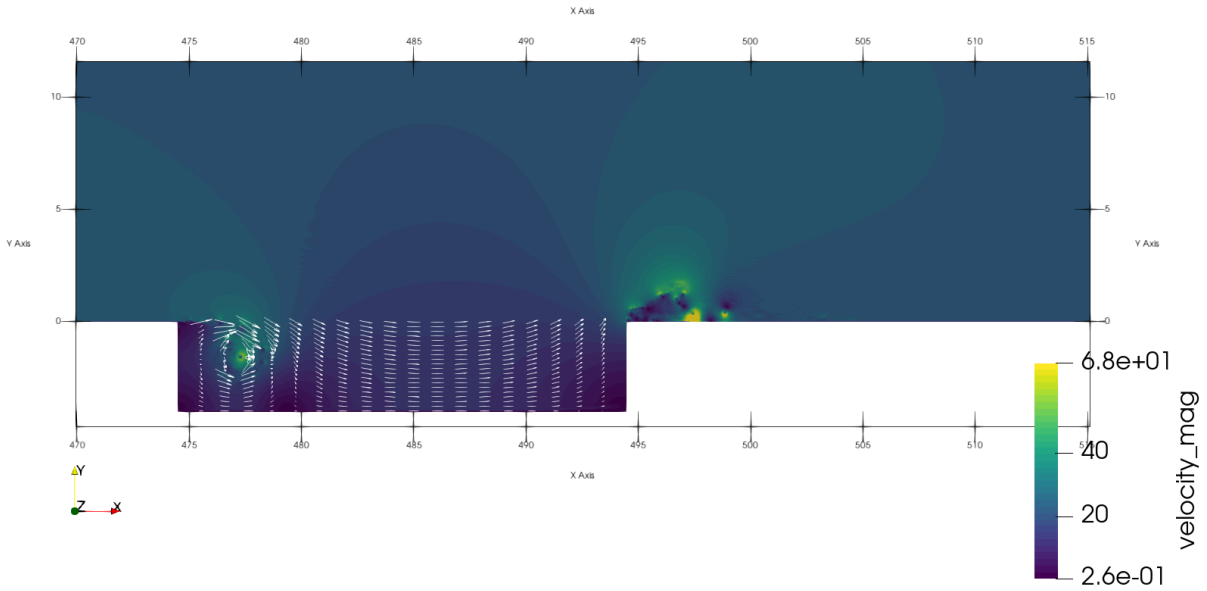
Para uma visão mais detalhada da cavidade, a figura 9 ilustra a camada limite próxima à

Figura 8 – Contorno de velocidade total dado pelo *Nek5000*

(a) Domínio físico



(b) Detalhe da cavidade



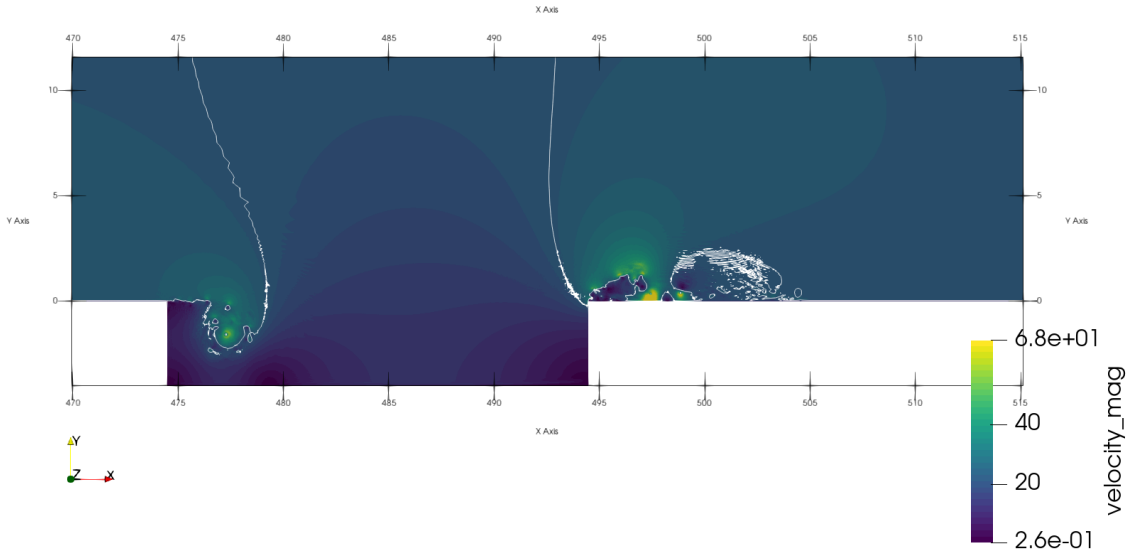
Fonte: Elaborada pelo autor

cavidade, além da magnitude da velocidade. Nota-se que a camada limite é muito menos espessa e que os vórtices foram acentuados. Além disso, há uma zona sobre a cavidade com menor velocidade e vórtices de alta velocidade próximos aos vértices da cavidade. Veja que a velocidade nos vórtices é quase 3 vezes maior que aquela na entrada do domínio, o que não concorda com os resultados da simulação de referência. Mais ainda, as espessuras de camada limite antes e depois da cavidade variam muito além do esperado, com valores completamente diferentes daqueles encontrados anteriormente.

Com base nessas discrepâncias, não é possível fazer uma comparação satisfatória da

precisão e do tempo de execução entre os programas. Uma vez que os resultados não se assemelham e exibem comportamentos distintos, pouco pode se saber sobre a diferença de desempenho. Porém, fica claro que o *Nek5000* apresenta um degrau de dificuldade muito maior para a configuração, o uso e a verificação dos resultados.

Figura 9 – Camada limite próxima à cavidade pelo *Nek5000*



Fonte: Elaborada pelo autor

6 CONCLUSÃO

Refletindo sobre os critérios delineados na Introdução, é preciso dividir a comparação entre os programas em dois campos. No primeiro campo são incluídos os critérios de facilidade de uso, clareza de documentação e simplicidade de especificação dos casos, ou seja, a experiência do usuário e a abundância de informação sobre o programa. No segundo campo estão o desempenho, as possibilidades de uso e a precisão dos resultados, abrangendo as considerações técnicas sobre os cenários possíveis de simulação e a qualidade da solução final.

Na primeira categoria, o programa *in-house* é superior, em todos os critérios. Embora o *Nek5000* pareça ser bem documentado, as informações disponíveis são superficiais e incompletas, como já comentado, e a especificação de casos é complexa, consistindo de várias etapas. O programa do grupo de pesquisa, que também é pouco documentado, permite ao menos que se consiga inspecionar o código-fonte em busca de informações sobre o que ocorre internamente, por ser escrito em uma mistura de *Matlab* com um pouco de *Fortran*. O mesmo não ocorre com o *Nek5000*, que possui uma base de código de mais de 40 anos em um padrão de *Fortran* antigo e ultrapassado.

Em relação à segunda categoria, as simulações feitas neste trabalho não são conclusivas. Como os resultados não são comparáveis, nada pode ser dito a respeito do tempo que levou para alcançá-los ou da divergência entre eles. Ainda assim, é possível concluir que o *Nek5000* possui aplicações mais abrangentes por incluir formulações para solução de uma ampla gama de números de Mach e Reynolds e de temperatura, embora possa não ser apropriado para este caso estudado. Note também que seu sucessor, o *NekRS* (FISCHER et al., 2022), parece possuir melhor documentação e receber maior trabalho no desenvolvimento de novas funções.

No entanto, o longo tempo para aprender a usar o *Nek5000* e aplicar suas funcionalidades diminui sua viabilidade para projetos curtos como iniciações científicas. Adicionalmente, é mais complexo estender o comportamento desse programa em comparação com aquele desenvolvido no laboratório, cujos funcionamento e estrutura são significativamente mais claros e simples de serem modificados.

Com base nestes motivos, é recomendável que se mantenha o uso do programa *in-house* para a simulação de casos como este, buscando o *Nek5000* apenas para situações

onde aquele for deficiente. Também é aconselhável que se evite indicar o *Nek5000* para projetos curtos cujos membros ainda não sabem utilizá-lo. Seu uso pode ser vantajoso em projetos maiores, em que haja tempo para estudá-lo e para solucionar problemas como os deste trabalho, que inevitavelmente ocorrerão em outras simulações.

A comparação entre os programas ainda exige investigações futuras que avaliem se é possível simular o caso deste trabalho e se há outros cenários em que o *Nek5000* possua vantagens. Mesmo sem encontrar uma resolução completa para a questão inicial, este estudo fornece uma introdução ao programa para aqueles interessados em continuar esta comparação no futuro, descrevendo dificuldades e apontando soluções para alguns problemas comuns durante a configuração das simulações.

REFERÊNCIAS

FISCHER, P. et al. NekRS, a GPU-accelerated spectral element Navier–Stokes solver. *Parallel Computing*, v. 114, p. 102982, 2022. ISSN 0167-8191. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167819122000710>.

FISCHER, P.; MULLEN, J. Filter-based stabilization of spectral element methods. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, v. 332, n. 3, p. 265–270, 2001. ISSN 0764-4442. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0764444200017638>.

GEUZAINE, C.; REMACLE, J.-F. Gmsh: A 3-D finite element mesh generator with built–in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, v. 79, n. 11, p. 1309–1331, 2009. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2579>.

MATHIAS, M.; MEDEIROS, M. F. Global instability analysis of a boundary layer flow over a small cavity. In: ____. *AIAA Aviation 2019 Forum*. AIAA, 2019. Disponível em: <https://arc.aiaa.org/doi/abs/10.2514/6.2019-3535>.

MATHIAS, M. S. *Computational study of the hydrodynamic stability of gaps and cavities in a subsonic compressible boundary layer*. Tese (Doutorado) — Escola de Engenharia de São Carlos, 12 2021.

Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.1*. [S.l.], 2023. Disponível em: <https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf>.

METCALF, M. et al. *Modern Fortran Explained: Incorporating Fortran 2023*. OUP Oxford, 2024. ISBN 9780198876588. Disponível em: <https://doi.org/10.1093/oso/9780198876571.001.0001>.

MILLS-CURRAN, W. C.; GILKEY, A. P.; FLANAGAN, D. P. *EXODUS: A finite element file format for pre- and postprocessing*. [S.l.], 1988. Disponível em: <https://www.osti.gov/biblio/6902151>.

Nek5000. *Nek5000 User Guide*. [S.l.], 2023. Disponível em: <https://nek5000.github.io/NekDoc/theory.html>.

Nek5000-users. *Nek5000 documentation details: Pn–Pn pressure solver*. 2017. Disponível em: <https://lists.mcs.anl.gov/pipermail/nek5000-users/2017-June/004367.html>.

SAHA, S.; BISWAS, P.; NATH, S. A review on spectral element solver Nek5000. In: *International Conference on Computational Sciences-Modelling, Computing and Soft Computing (CSMCS 2020)*. [s.n.], 2021. v. 2336, n. 1, p. 030001. ISSN 0094-243X. Disponível em: <https://doi.org/10.1063/5.0045709>.

TOMBOULIDES, A. G.; LEE, J. C. Y.; ORSZAG, S. A. Numerical simulation of low mach number reactive flows. *Journal of Scientific Computing*, v. 12, p. 139–167, 06 1997. ISSN 1573-7691. Disponível em: <https://doi.org/10.1023/A:1025669715376>.

APÊNDICE A: PROGRAMAS SUPLEMENTARES

Os arquivos transcritos correspondem ao programa `caseC-conversion`, desenvolvido para a tradução do problema para os arquivos do *Nek5000*. O executável do caso é construído com `make case`.

Arquivo 6.1 – `Makefile`

```
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause

.SUFFIXES:

# Location of input files and scripts
DATADIR ::= data
SCRIPTDIR ::= script
EXTERNALDIR ::= external

# `SRCDIR` contains one subdirectory per case setup
# Note the trailing slash in each member of SRCDIRS
SRCDIR ::= src
SRCDIRS ::= $(wildcard $(SRCDIR)/*/)
CASENAMES ::= $(SRCDIRS:$(SRCDIR)/%/=%)

# Octave function paths to include
INCLUDEDIR ::= m-files
INCLUDEDIRS ::= $(wildcard $(INCLUDEDIR)/*/)

# `SRCDIR` subdirectory structure is mirrored in each of the `STEPDIRS`
BUILDDIR ::= build
BOXDIR ::= $(BUILDDIR)/box-parts
MESHDIR ::= $(BUILDDIR)/mesh
CASEDIR ::= $(BUILDDIR)/case
IMAGEDIR ::= $(BUILDDIR)/images
VTKDIR ::= $(BUILDDIR)/vtk
STEPDIRS ::= $(BOXDIR) $(MESHDIR) $(CASEDIR) $(IMAGEDIR) $(VTKDIR)

# List of every subdirectory of `BUILDDIR` that may need to be created
BUILDDIRS ::= $(BUILDDIR) $(STEPDIRS) \
              $(foreach STEPDIR,$(STEPDIRS),$(addprefix $(STEPDIR)/,$(CASENAMES)))

# Internal scripts
MAKEIMAGES ::= $(SCRIPTDIR)/make_images.m
MAKEVTK ::= $(SCRIPTDIR)/make_vtk_cavity.m

# Externally-obtained input files
INPUT_MESH ::= $(DATADIR)/blob/mesh.mat
INPUT_PARAMS ::= $(DATADIR)/parameters.m
INPUT_REAFILE ::= $(EXTERNALDIR)/skeleton.rea

# Nektools
NEKROOT ?= $(HOME)/Documents/nek/Nek5000
GENBOX ::= $(NEKROOT)/bin/genbox
GENMAP ::= $(NEKROOT)/bin/genmap
REATORE2 ::= $(NEKROOT)/bin/reatore2
MAKENEK ::= $(NEKROOT)/bin/makenek

```

```
# Octave cli, without GUI, no banner, no octaverc, no history file
OCTAVE ::= /usr/bin/octave
OCTAVEFLAGS ::= -qfH --no-gui
OCTAVEINCLUDE ::= $(addprefix -p,$(INCLUDEDIRS))

.PHONY: all all-meshes all-cases build_dirs images vtk_files

all: all-meshes all-cases images

# Lists of files to be appended to by each case Makefile.inc
ALLMESHES ::=
ALLCASES ::=

# Include each case Makefile stub.
# Each Makefile stub declares recipes for the initial steps of compilation
# and creates the files required by the default recipes below.
# They also create their own PHONY targets for beign individually made
# and append the to `ALLMESHES` and `ALLCASES` to be run on `all`
include src/*/Makefile.inc

### Setup

build_dirs: $(BUILDDIRS)

# Recipe for each builddir
$(BUILDDIRS):
	mkdir -p $@

all-meshes: $(ALLMESHES)
all-cases: $(ALLCASES)

### Images

images: $(MAKEIMAGES) $(INPUT_MESH) | $(IMAGEDIR)
	$(OCTAVE) $(OCTAVEFLAGS) \
		$(OCTAVEINCLUDE) \
		$(MAKEIMAGES) \
		$(INPUT_MESH) \
		$(IMAGEDIR)

### VTK mesh files

vtk_files: $(MAKEVTK) | $(VTKDIR)
	$(OCTAVE) $(OCTAVEFLAGS) \
		$(OCTAVEINCLUDE) \
		$(MAKEVTK) \
		$(INPUT_MESH) \
		$(VTKDIR)

### Default recipes that can be used by all cases

### Meshing
# All Nek tools seem to need interactive input of the case files.
# It is possible to automate this by echoing the input.
# Another way would be to use the Don Libes' `EXPECT` or some derivative.

# Puts the skeleton .rea file alongside the mesh
$(MESHDIR)/%/$(notdir $(INPUT_REAFILE)): $(INPUT_REAFILE) | $(MESHDIR)/%
	cp $< $@

# Generates the mesh in the ASCII variant using a box file
```

```
111 %/this-with-bogus-params.rea: %/this.box %/$(notdir $(INPUT_REAFILE))
112         cd $(<D); printf '$(<F)' | $(GENBOX)
113         mv $(<D)/box.rea $@
114
115 # Converts the ASCII mesh to binary, dropping the old .rea data
116 %/this.re2 %/this.rea.bak: %/this-with-bogus-params.rea %/$(notdir $(INPUT_REAFILE))
117         echo $@ $<
118         # reator2 generates both a .rea and a .re2
119         cd $(<D); printf '$(basename $(<F))\n$(basename $(@F))' | $(REATORE2)
120         # Avoid reading the .rea with genmap later
121         mv $(@:.re2=.rea) $(@:.re2=.rea).bak
122
123 ### Building the partitioning map
124 # This step made me use this convoluted ASCII-to-binary conversion scheme.
125 # For some reason, genmap sometimes fails to read the binary made by genbox:
126 #   https://github.com/Nek5000/Nek5000/issues/562
127 # I switched to the .rea and it did work, but it needs the input in a weird
128 # parameter format in the same file as the mesh, and I only knew how to use
129 # the parameters in the .par file. Luckily, converting back to binary made it
130 # work.
131
132 MESH_TOLERANCE ?= 0.1
133 %/this.ma2: %/this.re2
134         cd $(<D)/; printf '$(basename $(<F))\n$(MESH_TOLERANCE)' | $(GENMAP)
135
136 ### Compiling
137 # As simple as copying the required case files to the case directory
138 # Nek needs adjustments in `makenek` for setting up compiler, compiler flags,
139 # and its own root folder.
140 # For large cases like this, Nek needs `makenek` modified with this:
141 # FFLAGS = "-mcmodel=medium"
142 # to force it to link large arrays.
143 # Compiling with -O9 may cause it to break, or not.
144 # The default rules below presume that `SIZE`, `.usr` and `.par` are in src/*
145
146 $(CASEDIR)/%/SIZE: $(SRCDIR)/%/SIZE | $(CASEDIR)/%
147         cp $< $@
148
149 $(CASEDIR)/%/this.usr: $(SRCDIR)/%/this.usr | $(CASEDIR)/%
150         cp $< $@
151
152 $(CASEDIR)/%/this.par: $(SRCDIR)/%/this.par | $(CASEDIR)/%
153         cp $< $@
154
155 $(CASEDIR)/%/this.re2: $(MESHDIR)/%/this.re2 | $(CASEDIR)/%
156         cp $< $@
157
158 $(CASEDIR)/%/this.ma2: $(MESHDIR)/%/this.ma2 | $(CASEDIR)/%
159         cp $< $@
160
161 $(CASEDIR)/%/nek5000: $(addprefix $(CASEDIR)/%/,SIZE this.usr this.par this.re2 this.ma2)
162         cd $(@D); $(MAKENEK) this
```

Arquivo 6.2 – `src/caseC-same-mesh/Makefile.inc`

```
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause

### Particular recipes for this case
# Since every variable might be overwritten, and especially since `CASENAME`
# will be, the hacky `%` capture must be used in the recipes so that the
# case name can be extracted on runtime from the recipe target path.
# None of this would be required if there was some form of lexical closure.

# Some overwritable variables for the current case
CASENAME ::= caseC-same-mesh

.PHONY: $(CASENAME)_MESH $(CASENAME)_CASE

# Resulting mesh and case files
$(CASENAME)_MESH: $(addprefix $(MESHDIR)/$(CASENAME)/,this.re2 this.ma2)
$(CASENAME)_CASE: $(addprefix $(CASEDIR)/$(CASENAME)/,nek5000 SIZE this.usr this.par this.re2 this.ma2)

ALLMESHES += $(CASENAME)_MESH
ALLCASES += $(CASENAME)_CASE

# Mesh conversion script
MAKEMESH ::= $(SCRIPTDIR)/make_box_mesh.m

# Builds the genbox mesh file by extracting nodes from Matlab caseC mesh
$(MESHDIR)/$(CASENAME)/this.box: $(MESHDIR)/%/this.box: \
        $(MAKEMESH) $(INPUT_PARAMS) $(INPUT_REAFILE) \
        | $(BOXDIR)/% $(MESHDIR)/%
        $(OCTAVE) $(OCTAVEFLAGS) \
                $(OCTAVEINCLUDE) \
                $(MAKEMESH) \
                $(INPUT_MESH) \
                $(INPUT_PARAMS) \
                $(BOXDIR)/$*/ \
                $(notdir $(INPUT_REAFILE))
        cat $(BOXDIR)/$*/* > $@
```

Arquivo 6.3 – `src/caseC-same-mesh/SIZE`

```
! SPDX-FileCopyrightText: 2024 Fernando Datz
! SPDX-FileCopyrightText: Copyright (c) 2008-2020, UCHICAGO ARGONNE, LLC.
! SPDX-License-Identifier: LicenseRef-Nek5000-BSD-3-Clause
!
c
c     Include file to dimension static arrays
c     and to set some hardwired run-time parameters
c
      integer ldim,lx1,lxd,lx2,lx1m,lelg,lelt,lpmin,ldimt
      integer lpelt,lbelt,toteq,lcvelt
      integer lelx,lely,lelz,mxprev,lgmres,lorder,lhis
      integer maxobj,lpert,nsessmax,lxo
      integer lfdm,ldimt_proj,lelr

      ! BASIC
      parameter (ldim=2)               ! domain dimension (2 or 3)
      parameter (lx1=4)                ! GLL points per element along each direction
      parameter (lxd=12)               ! GL  points for over-integration (dealiasing)
      parameter (lx2=lx1-0)            ! GLL points for pressure (lx1 or lx1-2)

      parameter (lelg=109898)          ! max number of global elements
      parameter (lpmin=16)             ! min number of MPI ranks
      parameter (lelt=lelg/lpmin + 3)  ! max number of local elements per MPI rank
      parameter (ldimt=1)              ! max auxiliary fields (temperature + scalars)

      ! OPTIONAL
      parameter (ldimt_proj=1)         ! max auxiliary fields residual projection
      parameter (lelr=lelt)            ! max number of local elements per restart file
      parameter (lhis=1)               ! max history/monitoring points
      parameter (maxobj=1)             ! max number of objects
      parameter (lpert=1)              ! max number of perturbations
      parameter (toteq=1)              ! max number of conserved scalars in CMT
      parameter (nsessmax=1)           ! max sessions to NEKNEK
      parameter (lxo=lx1)              ! max GLL points on output (lxo>=lx1)
      parameter (mxprev=20)            ! max dim of projection space
      parameter (lgmres=30)            ! max dim Krylov space
      parameter (lorder=3)             ! max order in time
      parameter (lx1m=1)               ! GLL points mesh solver
      parameter (lfdm=0)               ! unused
      parameter (lelx=1,lely=1,lelz=1) ! global tensor mesh dimensions

      parameter (lbelt=1)              ! lelt for mhd
      parameter (lpelt=1)              ! lelt for linear stability
      parameter (lcvelt=1)             ! lelt for cvode

      ! INTERNALS
      include 'SIZE.inc'
```

Arquivo 6.4 – src/caseC-same-mesh/this.usr

```
! SPDX-FileCopyrightText: 2024 Fernando Datz
! SPDX-FileCopyrightText: Copyright (c) 2008-2020, UCHICAGO ARGONNE, LLC.
! SPDX-License-Identifier: LicenseRef-Nek5000-BSD-3-Clause
!
c Only changed userbc and useric
c-----------------------------------------------------------------------
c  nek5000 user-file template
c
c  user specified routines:
c     - uservp  : variable properties
c     - userf   : local acceleration term for fluid
c     - userq   : local source term for scalars
c     - userbc  : boundary conditions
c     - useric  : initial conditions
c     - userchk : general purpose routine for checking errors etc.
c     - userqtl : thermal divergence for lowMach number flows
c     - usrdat  : modify element vertices
c     - usrdat2 : modify mesh coordinates
c     - usrdat3 : general purpose routine for initialization
c
c-----------------------------------------------------------------------
      subroutine uservp(ix,iy,iz,eg) ! set variable properties

c      implicit none

      integer ix,iy,iz,eg

      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      integer e
c     e = gllel(eg)

      udiff  = 0.0
      utrans = 0.0

      return
      end
c-----------------------------------------------------------------------
      subroutine userf(ix,iy,iz,eg) ! set acceleration term
c
c     Note: this is an acceleration term, NOT a force!
c     Thus, ffx will subsequently be multiplied by rho(x,t).
c
c      implicit none

      integer ix,iy,iz,eg

      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      integer e
c     e = gllel(eg)

      ffx = 0.0
```

```
58          ffy = 0.0
59          ffz = 0.0
60
61          return
62          end
63    c-----------------------------------------------------------------------
64          subroutine userq(ix,iy,iz,eg) ! set source term
65
66    c      implicit none
67
68          integer ix,iy,iz,eg
69
70          include 'SIZE'
71          include 'TOTAL'
72          include 'NEKUSE'
73
74          integer e
75    c     e = gllel(eg)
76
77          qvol   = 0.0
78
79          return
80          end
81    c-----------------------------------------------------------------------
82          subroutine userbc(ix,iy,iz,iside,eg) ! set up boundary conditions
83    c
84    c     NOTE ::: This subroutine MAY NOT be called by every process
85    c
86          implicit none
87
88          integer ix,iy,iz,iside,eg
89    c     Inlet speed as defined in the .par file
90          real ux_inlet
91    c     Temperature
92          real t_mean
93
94          include 'SIZE'
95          include 'TOTAL'
96          include 'NEKUSE'
97
98    c      if (cbc(iside,gllel(eg),ifield).eq.'v01')
99
100         ux_inlet = uparam(1)
101         t_mean = uparam(2)
102
103         if (ifield .eq. 1) then          ! velocity
104             ux   = ux_inlet
105             uy   = 0.0
106   c         uz   = 0.0
107         else if (ifield .eq. 2) then      ! Temperature
108             temp = t_mean
109         end if
110
111         return
112         end
113   c-----------------------------------------------------------------------
114         subroutine useric(ix,iy,iz,eg) ! set up initial conditions
115
116         implicit none
117
118         integer ix,iy,iz,eg
```

```
119   c     Inlet speed as defined in the .par file
120         real ux_inlet
121   c     Temperature
122         real t_mean
123
124         include 'SIZE'
125         include 'TOTAL'
126         include 'NEKUSE'
127
128         ux_inlet = uparam(1)
129         t_mean = uparam(2)
130
131         if (ifield .eq. 1) then          ! velocity
132             ux   = ux_inlet
133             uy   = 0.0
134   c         uz   = 0.0
135         elseif (ifield .eq. 2) then      ! Temperature
136             temp = t_mean
137         end if
138
139         return
140         end
141   c-----------------------------------------------------------------------
142         subroutine userchk()
143
144   c      implicit none
145
146         include 'SIZE'
147         include 'TOTAL'
148
149         return
150         end
151   c-----------------------------------------------------------------------
152         subroutine userqtl ! Set thermal divergence
153
154         call userqtl_scig
155
156         return
157         end
158   c-----------------------------------------------------------------------
159         subroutine usrdat()   ! This routine to modify element vertices
160
161   c      implicit none
162
163         include 'SIZE'
164         include 'TOTAL'
165
166         return
167         end
168   c-----------------------------------------------------------------------
169         subroutine usrdat2()  ! This routine to modify mesh coordinates
170
171   c      implicit none
172
173         include 'SIZE'
174         include 'TOTAL'
175
176         return
177         end
178   c-----------------------------------------------------------------------
179         subroutine usrdat3()
```

```
180
181   c      implicit none
182
183         include 'SIZE'
184         include 'TOTAL'
185
186         return
187         end
```

Arquivo 6.5 – `src/caseC-same-mesh/this.par`

```
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause

## Simulation parameters for a dimensional case

[GENERAL]
## Stopping at dimensional time
stopAt=endTime
endTime=65000.0
variableDT=yes
## Maximum time step
dt=1
## Target courant number
targetCFL=0.5
timeStepper=BDF3
writeControl=runTime
## Time interval for file output
writeInterval=100.0
writeDoublePrecision=yes
## U [m/s], towards +x
userParam01=22.6
## T [K]
userParam02=299.05
## Temporal filters taken from:
## https://nek5000.github.io/NekDoc/problem_setup/filter.html#explicit-filter
## Since it didn't converge, it was set to increasingly large weights up to 0.8
filtering=explicit
filterModes=2
filterWeight=0.80
## Enable over-integration
dealiasing=yes

[MESH]
motion=none

[PROBLEMTYPE]
equation=lowMachNS
axiSymmetry=no
cyclicBoundaries=no

[VELOCITY]
## Dynamic viscosity [kg/(m**2 * s)], at T
## A negative value sets the Reynolds number
# viscosity=-1405.0
viscosity=1.8444E-5
## Density [kg/m**3], at T
density=1.1845
## See manual FAQ, "How do I choose solver tolerances?", Pn-Pn
residualTol=1E-06
writeToFieldFile=yes

[PRESSURE]
## See manual FAQ, "How do I choose solver tolerances?", Pn-Pn
residualTol=1E-06
writeToFieldFile=yes

[TEMPERATURE]
```

```
58  ## Volumetric heat capacity [J / (m**3 * K)], at T
59  rhoCp=1182.03967
60  ## Thermal conductivity [W / (m * K)], at T
61  ## A negative value sets the Péclet number (Pe = (U * L * rho * Cp) / k)
62  conductivity=0.026036
63  writeToFieldFile=yes
```

Arquivo 6.6 – `src/caseC-custom-mesh/Makefile.inc`

```
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause

### Particular recipes for this case
# Since every variable might be overwritten, and especially since `CASENAME`
# will be, the hacky `%` capture must be used in the recipes so that the
# case name can be extracted on runtime from the recipe target path.
# None of this would be required if there was some form of lexical closure.

# Overwritable case name
CASENAME ::= caseC-custom-mesh

.PHONY: $(CASENAME)_MESH $(CASENAME)_CASE

# Resulting mesh and case files
$(CASENAME)_MESH: $(addprefix $(MESHDIR)/$(CASENAME)/,this.re2 this.ma2)
$(CASENAME)_CASE: $(addprefix $(CASEDIR)/$(CASENAME)/,nek5000 SIZE this.usr this.par this.re2 this.ma2)

ALLMESHES += $(CASENAME)_MESH
ALLCASES += $(CASENAME)_CASE

# Mesh conversion script
MAKECUSTOMMESH ::= $(SCRIPTDIR)/make_custom_box_mesh.m

# Number of elements in each generated box
THIS_NX ::= 10
THIS_NY ::= 5

# Builds a genbox mesh file by extracting nodes from Matlab caseC mesh
$(MESHDIR)/$(CASENAME)/this.box: $(MESHDIR)/%/this.box: \
        $(MAKECUSTOMMESH) $(INPUT_PARAMS) $(INPUT_REAFILE) \
        | $(BOXDIR)/% $(MESHDIR)/%
        $(OCTAVE) $(OCTAVEFLAGS) \
                $(OCTAVEINCLUDE) \
                $(MAKECUSTOMMESH) \
                $(THIS_NX) \
                $(THIS_NY) \
                $(INPUT_MESH) \
                $(INPUT_PARAMS) \
                $(BOXDIR)/$*/ \
                $(notdir $(INPUT_REAFILE))
        cat $(BOXDIR)/$*/* > $@
```

Arquivo 6.7 – `src/caseC-custom-mesh/SIZE`

```
! SPDX-FileCopyrightText: 2024 Fernando Datz
! SPDX-FileCopyrightText: Copyright (c) 2008-2020, UCHICAGO ARGONNE, LLC.
! SPDX-License-Identifier: LicenseRef-Nek5000-BSD-3-Clause
!
c
c     Include file to dimension static arrays
c     and to set some hardwired run-time parameters
c
      integer ldim,lx1,lxd,lx2,lx1m,lelg,lelt,lpmin,ldimt
      integer lpelt,lbelt,toteq,lcvelt
      integer lelx,lely,lelz,mxprev,lgmres,lorder,lhis
      integer maxobj,lpert,nsessmax,lxo
      integer lfdm,ldimt_proj,lelr

      ! BASIC
      parameter (ldim=2)               ! domain dimension (2 or 3)
      parameter (lx1=7)                ! GLL points per element along each direction
      parameter (lxd=12)               ! GL  points for over-integration (dealiasing)
      parameter (lx2=lx1-0)            ! GLL points for pressure (lx1 or lx1-2)

      parameter (lelg=300)             ! max number of global elements
      parameter (lpmin=8)              ! min number of MPI ranks
      parameter (lelt=lelg/lpmin + 3)  ! max number of local elements per MPI rank
      parameter (ldimt=2)              ! max auxiliary fields (temperature + scalars)

      ! OPTIONAL
      parameter (ldimt_proj=1)         ! max auxiliary fields residual projection
      parameter (lelr=lelt)            ! max number of local elements per restart file
      parameter (lhis=1)               ! max history/monitoring points
      parameter (maxobj=1)             ! max number of objects
      parameter (lpert=1)              ! max number of perturbations
      parameter (toteq=1)              ! max number of conserved scalars in CMT
      parameter (nsessmax=1)           ! max sessions to NEKNEK
      parameter (lxo=lx1)              ! max GLL points on output (lxo>=lx1)
      parameter (mxprev=20)            ! max dim of projection space
      parameter (lgmres=30)            ! max dim Krylov space
      parameter (lorder=3)             ! max order in time
      parameter (lx1m=1)               ! GLL points mesh solver
      parameter (lfdm=0)               ! unused
      parameter (lelx=1,lely=1,lelz=1) ! global tensor mesh dimensions

      parameter (lbelt=1)              ! lelt for mhd
      parameter (lpelt=1)              ! lelt for linear stability
      parameter (lcvelt=1)             ! lelt for cvode

      ! INTERNALS
      include 'SIZE.inc'
```

Arquivo 6.8 – src/caseC-custom-mesh/this.usr

```
! SPDX-FileCopyrightText: 2024 Fernando Datz
! SPDX-FileCopyrightText: Copyright (c) 2008-2020, UCHICAGO ARGONNE, LLC.
! SPDX-License-Identifier: LicenseRef-Nek5000-BSD-3-Clause
!
c Only changed userbc and useric
c-----------------------------------------------------------------------
c  nek5000 user-file template
c
c  user specified routines:
c     - uservp  : variable properties
c     - userf   : local acceleration term for fluid
c     - userq   : local source term for scalars
c     - userbc  : boundary conditions
c     - useric  : initial conditions
c     - userchk : general purpose routine for checking errors etc.
c     - userqtl : thermal divergence for lowMach number flows
c     - usrdat  : modify element vertices
c     - usrdat2 : modify mesh coordinates
c     - usrdat3 : general purpose routine for initialization
c
c-----------------------------------------------------------------------
      subroutine uservp(ix,iy,iz,eg) ! set variable properties

c      implicit none

      integer ix,iy,iz,eg

      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      integer e
c     e = gllel(eg)

      udiff  = 0.0
      utrans = 0.0

      return
      end
c-----------------------------------------------------------------------
      subroutine userf(ix,iy,iz,eg) ! set acceleration term
c
c     Note: this is an acceleration term, NOT a force!
c     Thus, ffx will subsequently be multiplied by rho(x,t).
c
c      implicit none

      integer ix,iy,iz,eg

      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'

      integer e
c     e = gllel(eg)

      ffx = 0.0
```

```
58          ffy = 0.0
59          ffz = 0.0
60
61          return
62          end
63    c-----------------------------------------------------------------------
64          subroutine userq(ix,iy,iz,eg) ! set source term
65
66    c      implicit none
67
68          integer ix,iy,iz,eg
69
70          include 'SIZE'
71          include 'TOTAL'
72          include 'NEKUSE'
73
74          integer e
75    c     e = gllel(eg)
76
77          qvol   = 0.0
78
79          return
80          end
81    c-----------------------------------------------------------------------
82          subroutine userbc(ix,iy,iz,iside,eg) ! set up boundary conditions
83    c
84    c     NOTE ::: This subroutine MAY NOT be called by every process
85    c
86          implicit none
87
88          integer ix,iy,iz,iside,eg
89    c     Inlet speed as defined in the .par file
90          real ux_inlet
91    c     Temperature
92          real t_mean
93
94          include 'SIZE'
95          include 'TOTAL'
96          include 'NEKUSE'
97
98    c      if (cbc(iside,gllel(eg),ifield).eq.'v01')
99
100         ux_inlet = uparam(1)
101         t_mean = uparam(2)
102
103         if (ifield .eq. 1) then          ! velocity
104             ux   = ux_inlet
105             uy   = 0.0
106   c         uz   = 0.0
107         else if (ifield .eq. 2) then      ! Temperature
108             temp = t_mean
109         end if
110
111         return
112         end
113   c-----------------------------------------------------------------------
114         subroutine useric(ix,iy,iz,eg) ! set up initial conditions
115
116         implicit none
117
118         integer ix,iy,iz,eg
```

```
119   c     Inlet speed as defined in the .par file
120         real ux_inlet
121   c     Temperature as defined in the .par file
122         real t_mean
123
124         include 'SIZE'
125         include 'TOTAL'
126         include 'NEKUSE'
127
128         ux_inlet = uparam(1)
129         t_mean = uparam(2)
130
131         if (ifield .eq. 1) then          ! velocity
132             if (y .ge. 0.0) then
133                 ux   = ux_inlet
134             else
135                 ux   = 0.0
136             end if
137             uy   = 0.0
138   c         uz   = 0.0
139         elseif (ifield .eq. 2) then      ! Temperature
140             temp = t_mean
141         end if
142
143         return
144         end
145   c-----------------------------------------------------------------------
146         subroutine userchk()
147
148   c      implicit none
149
150         include 'SIZE'
151         include 'TOTAL'
152
153         return
154         end
155   c-----------------------------------------------------------------------
156         subroutine userqtl ! Set thermal divergence
157
158         call userqtl_scig
159
160         return
161         end
162   c-----------------------------------------------------------------------
163         subroutine usrdat()   ! This routine to modify element vertices
164
165   c      implicit none
166
167         include 'SIZE'
168         include 'TOTAL'
169
170         return
171         end
172   c-----------------------------------------------------------------------
173         subroutine usrdat2()  ! This routine to modify mesh coordinates
174
175   c      implicit none
176
177         include 'SIZE'
178         include 'TOTAL'
179
```

```
180         return
181         end
182   c-----------------------------------------------------------------------
183         subroutine usrdat3()
184   
185   c      implicit none
186   
187         include 'SIZE'
188         include 'TOTAL'
189   
190         return
191         end
```

Arquivo 6.9 – src/caseC-custom-mesh/this.par

```
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause

## Simulation parameters for a non-dimensional case

[GENERAL]
## Stopping at non-dimensional time
stopAt=endTime
endTime=65000.0
variableDT=yes
## Maximum time step
dt=1
## Target courant number
targetCFL=0.5
timeStepper=BDF3
writeControl=runTime
## Time interval for file output
writeInterval=100.0
writeDoublePrecision=yes
## U / U, towards +x, non-dimensional
## U = 22.6 [m/s]
userParam01=1.0
## (T / T) - 1, non-dimensional
## T = 299.05 [K]
userParam02=0.0
## Temporal filters taken from:
## https://nek5000.github.io/NekDoc/problem_setup/filter.html#explicit-filter
## Since it didn't converge, it was set to increasingly large weights up to 0.8
filtering=explicit
filterModes=2
filterWeight=0.80
## Enable over-integration
dealiasing=yes

[MESH]
motion=none

[PROBLEMTYPE]
equation=lowMachNS
axiSymmetry=no
cyclicBoundaries=no

[VELOCITY]
## Dynamic viscosity [kg/(m**2 * s)], at T
## A negative value sets the Reynolds number
# viscosity=1.8444E-5
viscosity=-1405.0
## rho / rho, at T
## density = 1.1845 [kg/m**3]
density=1.0
## See manual FAQ, "How do I choose solver tolerances?", Pn-Pn
residualTol=1E-06
writeToFieldFile=yes

[PRESSURE]
## See manual FAQ, "How do I choose solver tolerances?", Pn-Pn
residualTol=1E-06
```

```
writeToFieldFile=yes

[TEMPERATURE]
## Volumetric heat capacity [J / (m**3 * K)], at T
# rhoCp = 1182.03967
rhoCp=1.0
## Thermal conductivity [W / (m * K)], at T
## A negative value sets the Péclet number (Pe = (U * L * rho * Cp) / k)
# conductivity=0.026036
conductivity=-1026.045
writeToFieldFile=yes
```

Arquivo 6.10 – `script/make_box_mesh.m`

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {} {} make_box_mesh ()
##
## Read the case mesh and outputs a genbox-compatible mesh description.
##
## The case mesh is sliced into several boxes, each of which correspond to
## a different flow region or boundary condition.
## Namely, it separates the initial buffer zone, prior to the plate,
## which has slippery surface;
## the initial plate surface, prior to the cavity;
## the cavity;
## its upper region, to ensure correct box connectivity;
## and the final plate surface, downstream of the cavity.
##
## It still needs additional input parameters in .usr and .par files
## to define speed and temperature at the beginning for the whole mesh
## and at every timestep for the boundary.
##
## @end deftypefn

function make_box_mesh()
    [mesh_filename, par_filename, output_dir, skeleton_rea_filename] = __parse_args(argv());

    mesh_s = load(mesh_filename);
    par_s = get_parameters(par_filename);

    [boxes_s, nfields] = make_boxes(mesh_s.X, mesh_s.Y, par_s);

    write_box_files(boxes_s, nfields, output_dir, skeleton_rea_filename);
end

### Arguments parser

function [mesh_filename, par_filename, output_dir, rea_filename] = __parse_args(args)
    if 3 <= length(args) && length(args) <= 4
        mesh_filename = args{1};
        par_filename = args{2};
        output_dir = args{3};
    else
        disp("USAGE: ./make-box-mesh.m MESH_FILE PARAMETERS_FILE OUTDIR [BASE_REA_FILE]");
        return
    end

    if not(isfile(mesh_filename))
        err_msg = sprintf("Mesh file '%s' does not exist", mesh_filename);
        error(err_msg);
    end

    if not(isfile(par_filename))
        err_msg = sprintf("Parameters file '%s' does not exist", par_filename);
        error(err_msg);
    end
```

```
58
59      if not(isfolder(output_dir))
60          err_msg = sprintf("Output directory '%s' does not exist", output_dir);
61          error(err_msg);
62      end
63
64      if length(args) == 4
65          rea_filename = args{4};
66      else
67          rea_filename = '';
68      end
69  end
70
71  ### Box translation
72
73  function [boxes, nfields] = make_boxes(X, Y, par_s)
74      ## Slice mesh into boxes and annotate its boundaries conditions
75      ## Uses parameters instead of wall matrix data, since they are equivalent
76      ## The result is a struct with each box as a field.
77      ## Not very extendable as it is.
78
79      ## Slice indices
80      pre_buffer_x_start = 1;
81      pre_buffer_x_end = find(X >= 0, 1);
82
83      cavity_x_start = find(X >= par_s.flowType.cav{1}.x(1), 1);
84      cavity_x_end = find(X >= par_s.flowType.cav{1}.x(2), 1);
85
86      post_buffer_x_start = find(X >= par_s.domain.xf, 1);
87      post_buffer_x_end = length(X);
88
89      plate_y = find(Y >= 0, 1);
90      cavity_y = 1;
91      ceiling_y = length(Y);
92
93      ## Scale X and Y to actual size
94      X = par_s.dref * X;
95      Y = par_s.dref * Y;
96
97      ## Get velocity condition "enum"
98      U_bc = velocity_boundary_conditions();
99
100     ## Number of variable fields
101     nfields = 1;
102
103     ## Box subdivisions
104     boxes = struct(
105         'pre_buffer', struct(
106             'x', X(pre_buffer_x_start:pre_buffer_x_end),
107             'y', Y(plate_y:ceiling_y),
108             ## [-x, +x, -y, +y]
109             'U_boundaries', {{U_bc.vel_u, U_bc.internal, U_bc.slip_free, U_bc.outflow}}
110         ),
111         'leading_boundary_layer', struct(
112             'x', X(pre_buffer_x_end:cavity_x_start),
113             'y', Y(plate_y:ceiling_y),
114             ## [-x, +x, -y, +y]
115             'U_boundaries', {{U_bc.internal, U_bc.internal, U_bc.wall, U_bc.outflow}}
116         ),
117         'boundary_layer', struct(
118             'x', X(cavity_x_start:cavity_x_end),
```

```
            'y', Y(plate_y:ceiling_y),
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.internal, U_bc.internal, U_bc.internal, U_bc.outflow}}
        ),
        'trailing_boundary_layer', struct(
            'x', X(cavity_x_end:post_buffer_x_start),
            'y', Y(plate_y:ceiling_y),
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.internal, U_bc.internal, U_bc.wall, U_bc.outflow}}
        ),
        'cavity', struct(
            'x', X(cavity_x_start:cavity_x_end),
            'y', Y(cavity_y:plate_y),
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.wall, U_bc.wall, U_bc.wall, U_bc.internal}}
        ),
        'post_buffer', struct(
            'x', X(post_buffer_x_start:post_buffer_x_end),
            'y', Y(plate_y:ceiling_y),
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.internal, U_bc.outflow, U_bc.wall, U_bc.outflow}}
        )
    );
end

make_box_mesh();
```

Arquivo 6.11 – `script/make_custom_box_mesh.m`

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {} {} make_custom_box_mesh ()
##
## Read the case mesh and output a modified genbox-compatible mesh description.
##
## As in @code{make_box_mesh}, but with custom number of elements, so that
## simulations can run quicker. Also, the temperature field is enabled.
##
## Unlike @code{make_box_mesh}, this code DOES NOT scale the axis.
## It is intended for a non-dimensional run.
##
## @end deftypefn

function make_custom_box_mesh()
    [NX, NY, mesh_filename, par_filename, output_dir, skeleton_rea_filename] = __parse_args(argv());

    mesh_s = load(mesh_filename);
    par_s = get_parameters(par_filename);

    [boxes_s, nfields] = make_custom_boxes(mesh_s.X, mesh_s.Y, NX, NY, par_s);

    write_box_files(boxes_s, nfields, output_dir, skeleton_rea_filename);
end

### Arguments parser

function [NX, NY, mesh_filename, par_filename, output_dir, rea_filename] = __parse_args(args)
    if 5 <= length(args) && length(args) <= 6
        str_NX = args{1};
        str_NY = args{2};
        mesh_filename = args{3};
        par_filename = args{4};
        output_dir = args{5};
    else
        disp(["USAGE: ./make-custom-box-mesh.m N_ELEM_X N_ELEM_Y " ...
              "MESH_FILE PARAMETERS_FILE OUTDIR [BASE_REA_FILE]"]);
        return
    end

    NX = str2double(str_NX);
    if isnan(NX)
        err_msg = sprintf("NX '%s' is not a number", str_NX);
        error(err_msg);
    end

    NY = str2double(str_NY);
    if isnan(NY)
        err_msg = sprintf("NY '%s' is not a number", str_NY);
        error(err_msg);
    end

    if not(isfile(mesh_filename))
```

```
58          err_msg = sprintf("Mesh file '%s' does not exist", mesh_filename);
59          error(err_msg);
60      end
61
62      if not(isfile(par_filename))
63          err_msg = sprintf("Parameters file '%s' does not exist", par_filename);
64          error(err_msg);
65      end
66
67      if not(isfolder(output_dir))
68          err_msg = sprintf("Output directory '%s' does not exist", output_dir);
69          error(err_msg);
70      end
71
72      if length(args) == 6
73          rea_filename = args{6};
74      else
75          rea_filename = '';
76      end
77  end
78
79  ### Box translation
80
81  ## The mesh translations assumes a certain shape for the surface,
82  ## shown below.
83  ## To simplify variable names, the regions, points and lengths will
84  ## be as follows:
85  ##
86  ## Y   a     X1     b     X2     c   X3   d      X4     e      X5     f
87  ## ^   +            +            +        +             +             + i
88  ## |   > pre_buffer : leading_BL :   BL   : trailing_BL : post_buffer   Y2
89  ## +   **************+------------+........+-------------+-------------+ h
90  ##                               | cavity | Y1
91  ##                   +-> X       +--------+ g
92
93
94  function [boxes, nfields] = make_custom_boxes(X, Y, NX, NY, par_s)
95      ## Slice mesh into boxes and annotate its boundaries conditions
96      ## Uses parameters instead of wall matrix data, since they are equivalent
97      ## The result is a struct with each box as a field.
98
99      ## Slice indices, see chart above
100     ### For X
101     a_i = 1;
102     b_i = find(X >= 0, 1);
103     c_i = find(X >= par_s.flowType.cav{1}.x(1), 1);
104     d_i = find(X >= par_s.flowType.cav{1}.x(2), 1);
105     e_i = find(X >= par_s.domain.xf, 1);
106     f_i = length(X);
107     ### And for Y
108     g_i = 1;
109     h_i = find(Y >= 0, 1);
110     i_i = length(Y);
111
112     ## Simple resampling of the domain
113     ### For X
114     X1 = linspace(X(a_i), X(b_i), NX + 1);
115     X2 = linspace(X(b_i), X(c_i), NX + 1);
116     X3 = linspace(X(c_i), X(d_i), NX + 1);
117     X4 = linspace(X(d_i), X(e_i), NX + 1);
118     X5 = linspace(X(e_i), X(f_i), NX + 1);
```

```
    ### And for Y
    Y1 = linspace(Y(g_i), Y(h_i), NY + 1);
    Y2 = linspace(Y(h_i), Y(i_i), NY + 1);

    ## Get velocity and temperature condition "enums"
    U_bc = velocity_boundary_conditions();
    T_bc = temperature_boundary_conditions();

    ## Number of variable fields
    nfields = 2;

    ## Box subdivisions
    boxes = struct(
        'pre_buffer', struct(
            'x', X1,
            'y', Y2,
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.vel_u, U_bc.internal, U_bc.slip_free, U_bc.outflow}},
            ## [-x, +x, -y, +y]
            'T_boundaries', {{T_bc.temp_u, T_bc.internal, T_bc.insulated, T_bc.insulated}}
        ),
        'leading_boundary_layer', struct(
            'x', X2,
            'y', Y2,
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.internal, U_bc.internal, U_bc.wall, U_bc.outflow}},
            ## [-x, +x, -y, +y]
            'T_boundaries', {{T_bc.internal, T_bc.internal, T_bc.insulated, T_bc.insulated}}
        ),
        'boundary_layer', struct(
            'x', X3,
            'y', Y2,
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.internal, U_bc.internal, U_bc.internal, U_bc.outflow}},
            ## [-x, +x, -y, +y]
            'T_boundaries', {{T_bc.internal, T_bc.internal, T_bc.internal, T_bc.insulated}}
        ),
        'trailing_boundary_layer', struct(
            'x', X4,
            'y', Y2,
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.internal, U_bc.internal, U_bc.wall, U_bc.outflow}},
            ## [-x, +x, -y, +y]
            'T_boundaries', {{T_bc.internal, T_bc.internal, T_bc.insulated, T_bc.insulated}}
        ),
        'cavity', struct(
            'x', X3,
            'y', Y1,
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.wall, U_bc.wall, U_bc.wall, U_bc.internal}},
            ## [-x, +x, -y, +y]
            'T_boundaries', {{T_bc.insulated, T_bc.insulated, T_bc.insulated, T_bc.internal}}
        ),
        'post_buffer', struct(
            'x', X5,
            'y', Y2,
            ## [-x, +x, -y, +y]
            'U_boundaries', {{U_bc.internal, U_bc.outflow, U_bc.wall, U_bc.outflow}},
            ## [-x, +x, -y, +y]
            'T_boundaries', {{T_bc.internal, T_bc.insulated, T_bc.insulated, T_bc.insulated}}
        )
```

```
    );
end

make_custom_box_mesh();
```

Arquivo 6.12 – `script/make_images.m`

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {} {} make_images ()
##
## Create illustrations of the original mesh.
##
## Uses the original mesh file in order to generate some illustrations
## of the mesh wall and grid refinement.
##
## Might consume too much memory on contourf calls.
##
## @end deftypefn

## Mark it as not a function file
1;

function make_images()
    [mesh_filename, output_dir] = __parse_args(argv());

    mesh_s = load(mesh_filename);

    outname = fullfile(output_dir, "mesh-walls.pdf");
    plot_mesh_walls(mesh_s.X, mesh_s.Y, mesh_s.wall, outname);

    outname = fullfile(output_dir, "mesh-regions.pdf");
    plot_mesh_regions(mesh_s.X, mesh_s.Y, mesh_s.wall, outname);

    outname = fullfile(output_dir, "mesh-grid.pdf");
    plot_mesh_lines(mesh_s.X, mesh_s.Y, outname);
end

### Arguments parser

function [mesh_filename, output_dir] = __parse_args(args)
    if length(args) == 2
        mesh_filename = args{1};
        output_dir = args{2};
    else
        disp("USAGE: ./make-images.m MESH_FILE OUTDIR")
        return
    end

    if not(isfile(mesh_filename))
        err_msg = sprintf("Mesh file '%s' does not exist", mesh_filename);
        error(err_msg);
    end

    if not(isfolder(output_dir))
        err_msg = sprintf("Output directory '%s' does not exist", output_dir);
        error(err_msg);
    end
end

```

```
### Plotting helpers

function plot_mesh_walls(X, Y, wall, filename)
    fig_h = my_figure();
    _fig_cleanup_h = onCleanup(@() close(fig_h));
    ax_h = my_axes(fig_h);

    ## Make wall a number
    wall = int8(wall);

    ## Plot region borders using contour
    contourf(ax_h, X, Y, wall');

    ## Add labels
    xlabel(ax_h, "X");
    ylabel(ax_h, "Y");

    ## Print to PDF
    my_print(fig_h, filename);
end

function plot_mesh_regions(X, Y, wall, filename)
    fig_h = my_figure();
    _fig_cleanup_h = onCleanup(@() close(fig_h));
    ax_h = my_axes(fig_h);

    ## Find points marked as wall
    wall_f = (wall' == 1) + 1;

    ## Make grid from ticks
    [mx, my] = meshgrid(X, Y);

    ## Colour with min and max from the Viridis colormap
    colours = viridis(2)(wall_f)(:);
    scatter(ax_h, mx, my, 4, colours, "marker", ".");

    ## Fit axis limits
    axis(ax_h, [X(1) X(end) Y(1) Y(end)]');

    ## Add labels
    xlabel(ax_h, "X");
    ylabel(ax_h, "Y");

    ## Print to PDF
    my_print(fig_h, filename);
end

function plot_mesh_lines(X, Y, filename)
    fig_h = my_figure();
    _fig_cleanup_h = onCleanup(@() close(fig_h));
    ax_h = my_axes(fig_h);

    ## Plot horizontal lines
    for i = Y
        line(ax_h, "xdata", [X(1) X(end)], "ydata", [i i], 'linewidth', 0.05);
    end

    ## Plot vertical lines
    for i = X
        line(ax_h, "xdata", [i i], "ydata", [Y(1) Y(end)], 'linewidth', 0.05);
    end
```

```
119
120     ## Fit axis limits
121     axis(ax_h, [X(1) X(end) Y(1) Y(end)]');
122
123     ## Add labels
124     xlabel(ax_h, "X");
125     ylabel(ax_h, "Y");
126
127     ## Print to PDF
128     my_print(fig_h, filename);
129 end
130
131 ### Printing setup
132
133 function fig_h = my_figure(paper_size = [15.0 10.0])
134     ## Default setup for figures
135     fig_h = figure(
136         "Visible", "off", 'PaperUnits', 'centimeters',
137         'PaperSize', paper_size, 'PaperPosition', [0.0 0.0 paper_size]
138     );
139 end
140
141 function ax_h = my_axes(fig_h, font_size = 11)
142     ## Default setup for axes
143     ax_h = axes(fig_h, "FontUnits", "points", "FontSize", font_size);
144 end
145
146 function my_print(fig_h, filename, font_flag = "-FLatin Modern Math")
147     ## Default setup for print
148
149     ## For some reason Octave only sets the font when it's used in `print`,
150     ## because of the pipeline it uses from svgconvert to ghostscript.
151     ## Otherwise, it ignores it and sets a default of the same family.
152     print(fig_h, filename, "-dPDF", font_flag);
153 end
154
155 make_images();
```

Arquivo 6.13 – `script/make_more_images.m`

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {} {} make_more_images ()
##
## Create additional illustrations based on mesh and flow files.
##
## Uses the original mesh file and a generated flow file from the lab-made
## Matlab tool.
##
## Mostly generates flow illustrations, like contours and flow arrows.
## Also plots boundary layer thickness comparisons between Blasius and
## the simulation.
##
## @end deftypefn

## Mark it as not a function file
1;

function make_more_images()
    [mesh_filename, par_filename, flow_filename, output_dir] = __parse_args(argv());

    ## Wants to plot streamlines, otherwise it's arrows
    stream_lines = false;

    ## Load data
    mesh_s = load(mesh_filename);
    par_s = get_parameters(par_filename);
    flow_s = load(flow_filename);

    ## Plot wall as contour
    plot_domain_wall(mesh_s, par_s.domain, output_dir);

    ## Velocity levels for contour
    N = 15;
    breakpoints = linspace(min(flow_s.U(:)), 0.99, N);

    ## Domain above plate
    upper_domain_s = get_upper_domain(mesh_s, par_s.domain);

    plot_flow_above_plate(
        mesh_s, flow_s, upper_domain_s, breakpoints, output_dir
    );

    ## Domain near cavity
    vslice_domain_s = get_cavity_vslice(mesh_s);
    ## Domain inside cavity
    cavity_domain_s = get_cavity_domain(mesh_s);

    plot_flow_near_cavity(
        mesh_s, flow_s, vslice_domain_s, cavity_domain_s, breakpoints,
        output_dir, stream_lines
    );

```

```
58      ## Smaller plot size to fit on a page
59      paper_size_bl = [15.0 6.0];
60
61      ## Domain after the boundary layer started
62      bl_domain_s = setfield(upper_domain_s, 'xi', max(0, upper_domain_s.xi));
63
64      ## Domain surrounding the cavity, above the plate
65      zoom_domain_s = get_cavity_surroundings(mesh_s);
66
67      ## Plot boundary layer for the whole domain
68      plot_bl(
69          mesh_s, flow_s, par_s, bl_domain_s,
70          output_dir, "boundary-", paper_size_bl
71      );
72      ## And also for the cavity zoom
73      plot_bl(
74          mesh_s, flow_s, par_s, zoom_domain_s,
75          output_dir, "boundary-cavity-", paper_size_bl
76      );
77  end
78
79  ### Arguments parser
80
81  function [mesh_filename, par_filename, flow_filename, output_dir] = __parse_args(args)
82      if length(args) == 4
83          mesh_filename = args{1};
84          par_filename = args{2};
85          flow_filename = args{3};
86          output_dir = args{4};
87      else
88          disp(["USAGE: ./make-more-images.m " ...
89                "MESH_FILE PARAMETERS_FILE FLOW_FILE OUTDIR"])
90          return
91      end
92
93      if not(isfile(mesh_filename))
94          err_msg = sprintf("Mesh file '%s' does not exist", mesh_filename);
95          error(err_msg);
96      end
97
98      if not(isfile(par_filename))
99          err_msg = sprintf("Parameter file '%s' does not exist", par_filename);
100         error(err_msg);
101     end
102
103     if not(isfile(flow_filename))
104         err_msg = sprintf("Flow file '%s' does not exist", flow_filename);
105         error(err_msg);
106     end
107
108     if not(isfolder(output_dir))
109         err_msg = sprintf("Output directory '%s' does not exist", output_dir);
110         error(err_msg)
111     end
112 end
113
114 ### Plotting helpers
115
116 function plot_domain_wall(mesh_s, domain_s, output_dir)
117     ## Plot wall in the domain
118
```

```
    fig_h = my_figure();
    _fig_cleanup_h = onCleanup(@() close(fig_h));
    ax_h = my_axes(fig_h);

    plot_mesh_walls_in_domain(ax_h, mesh_s, domain_s);

    ## Add labels
    xlabel(ax_h, "X");
    ylabel(ax_h, "Y");

    ## Print to PDF
    outname = fullfile(output_dir, "mesh-walls-domain.pdf");
    my_print(fig_h, outname);
end

function plot_flow_above_plate(
        mesh_s, flow_s, domain_s, breakpoints, output_dir)
    ## Plot flow above plate, ignoring the cavity

    fig_h = my_figure();
    _fig_cleanup_h = onCleanup(@() close(fig_h));
    ax_h = my_axes(fig_h);

    ## Plot flow u_x contour in the upper domain
    plot_flow_contourf(ax_h, mesh_s, flow_s, domain_s, breakpoints);

    ## Mark start and end of cavity
    plot_cavity_xlim_indicator(ax_h, mesh_s);

    ## Add labels
    xlabel(ax_h, "X");
    ylabel(ax_h, "Y");
    colorbar(ax_h, 'Location', 'NorthOutside');

    ## Print to PDF
    outname = fullfile(output_dir, "flow-domain.pdf");
    my_print(fig_h, outname);
end

function plot_flow_near_cavity(
        mesh_s, flow_s, vslice_domain_s, cavity_domain_s, breakpoints,
        output_dir, stream_lines)
    ## Plot flow in the cavity and slightly above it, ignoring the rest

    fig_h = my_figure();
    _fig_cleanup_h = onCleanup(@() close(fig_h));
    ax_h = my_axes(fig_h);
    hold(ax_h, 'on');

    ## Plot flow u_x contour near cavity
    plot_flow_contourf(ax_h, mesh_s, flow_s, vslice_domain_s, breakpoints);

    if stream_lines
        plot_flow_streamlines(ax_h, mesh_s, flow_s, cavity_domain_s);
    else
        Nx = 20;
        Ny = 20;
        plot_flow_arrows(ax_h, mesh_s, flow_s, cavity_domain_s, Nx, Ny);
    end

    ## Add labels
```

```
    xlabel(ax_h, "X");
    ylabel(ax_h, "Y");
    colorbar(ax_h, 'Location', 'NorthOutside');

    ## Print to PDF
    outname = fullfile(output_dir, "flow-cavity.pdf");
    my_print(fig_h, outname);
end

function plot_bl(
        mesh_s, flow_s, par_s, domain_s, output_dir, file_prefix, paper_size)
    ## Plot the developement of each boundary layer over the plate

    ## Plot boundary layer parameters
    figure_names = {{'delta-99', '\delta_{99}'},
                    {'delta-star', '\delta^*'},
                    {'theta', '\theta'}};

    fig_hs = zeros(1, length(figure_names));
    ax_hs = zeros(1, length(figure_names));
    for i = 1:length(figure_names)
        fig_hs(i) = my_figure(paper_size);
        ax_hs(i) = my_axes(fig_hs(i));
    end

    plot_boundary_layers(
        ax_hs(1), ax_hs(2), ax_hs(3), mesh_s, flow_s, par_s, domain_s
    );

    for i = 1:length(figure_names)
        ## Mark start and end of cavity
        plot_cavity_xlim_indicator(ax_hs(i), mesh_s);

        ## Add labels
        xlabel(ax_hs(i), 'X');
        ylabel(ax_hs(i), figure_names{i}{2}, 'Interpreter', 'tex');
        xlim(ax_hs(i), [max([0, domain_s.xi]), domain_s.xf]);
        grid(ax_hs(i), 'on');

        ## Print to tight PDF
        outname = fullfile(output_dir, [file_prefix figure_names{i}{1} ".pdf"]);
        my_print(fig_hs(i), outname);
        close(fig_hs(i));
    end
end

### Printing setup

function fig_h = my_figure(paper_size = [15.0 10.0])
    ## Default setup for figures
    fig_h = figure(
        "Visible", "off", 'PaperUnits', 'centimeters',
        'PaperSize', paper_size, 'PaperPosition', [0.0 0.0 paper_size]
    );
end

function ax_h = my_axes(fig_h, font_size = 11)
    ## Default setup for axes
    ax_h = axes(fig_h, "FontUnits", "points", "FontSize", font_size);
end

```

```
241 function my_print(fig_h, filename, font_flag = "-FLatin Modern Math")
242     ## Default setup for print
243
244     ## For some reason Octave only sets the font when it's used in `print`,
245     ## because of the pipeline it uses from svgconvert to ghostscript.
246     ## Otherwise, it ignores it and sets a default of the same family.
247     print(fig_h, filename, "-dPDF", font_flag);
248 end
249
250 make_more_images();
```

Arquivo 6.14 – `script/make_vtk_cavity.m`

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {} {} make_vtk_cavity ()
##
## Create VTK formatted grid and points files.
##
## Generate a mesh of the cavity inside, less refined than the original.
##
## For use in generating a mesh in paraview by resampling Nek flow data into
## less points and ploting flow arrows in Paraview, or at least that was
## the intention.
##
## It seems Paraview's `Resample to Image` and `Resample to Dataset` work
## while this method does not, for some unknown reason.
## This script is left as a historical artifact, as the VTK files may be
## useful in the future.
##
## Note that STRUCTURED_POINTS seems to result in a grid of points only,
## without connections, while RECTILINEAR_GRID generates the connections
## like in a mesh, and is suitable for extracting a surface.
##
## @end deftypefn

## Mark it as not a function file
1;

function make_vtk_cavity()
    [mesh_filename, output_dir] = __parse_args(argv());

    mesh_s = load(mesh_filename);
    cavity_s = get_cavity_domain(mesh_s);

    N = [20, 20, 1];
    O = [cavity_s.xi, cavity_s.yi, cavity_s.zi];
    E = [cavity_s.xf, cavity_s.yf, cavity_s.zf];
    S = (E - O) ./ N;
    S(isnan(S)) = 0;

    ## Write structured points
    filename = 'mesh-in-cavity_str-pt.vtk';
    full_path = fullfile(output_dir, filename);
    fh = fopen(full_path, 'w+');
    write_vtk_preamble(fh, 'CavityMesh');
    write_structured_points(fh, N, O, S);
    fclose(fh);

    ## Write structured points
    filename = 'mesh-in-cavity_r-grid.vtk';
    full_path = fullfile(output_dir, filename);
    fh = fopen(full_path, 'w+');
    write_vtk_preamble(fh, 'CavityMesh');
    write_rectilinear_grid(
        fh,
```

```
        linspace(cavity_s.xi, cavity_s.xf, N(1)),
        linspace(cavity_s.yi, cavity_s.yf, N(2)),
        linspace(cavity_s.zi, cavity_s.zf, N(3))
    );
    fclose(fh);
end

function [mesh_filename, output_dir] = __parse_args(args)
    if length(args) == 2
        mesh_filename = args{1};
        output_dir = args{2};
    else
        disp("USAGE: ./make_vtk_cavity.m MESH_FILE OUTDIR");
        return
    end

    if not(isfolder(output_dir))
        err_msg = sprintf("Output directory '%s' does not exist", output_dir);
        error(err_msg)
    end
end

function write_vtk_preamble(fh, id)
    ## Write preamble for ASCII .vtk files.
    fprintf(fh, "# vtk DataFile Version 2.0\n");
    fprintf(fh, "%s\n", id);
    fprintf(fh, "ASCII\n");
end

function write_structured_points(fh, N, O, S)
    ## Write a STRUCTURED_POINTS dataset

    fprintf(fh, "DATASET STRUCTURED_POINTS\n");
    fprintf(fh, "DIMENSIONS %d %d %d\n", N(1), N(2), N(3));
    fprintf(fh, "ORIGIN % .16E % .16E % .16E\n", O(1), O(2), O(3));
    fprintf(fh, "SPACING % .16E % .16E % .16E\n", S(1), S(2), S(3));
end

function write_rectilinear_grid(fh, X, Y, Z)
    fputs(fh, "DATASET RECTILINEAR_GRID\n");
    N = [length(X), length(Y), length(Z)];

    fprintf(fh, "DIMENSIONS %d %d %d\n", N(1), N(2), N(3));

    ## X coordinates
    fprintf(fh, "X_COORDINATES %d float\n", N(1));
    write_wrapped_data(fh, X);

    ## Y coordinates
    fprintf(fh, "Y_COORDINATES %d float\n", N(2));
    write_wrapped_data(fh, Y);

    ## Z coordinates
    fprintf(fh, "Z_COORDINATES %d float\n", N(3));
    write_wrapped_data(fh, Z);
end

function write_wrapped_data(fh, X)
    ## Writes a vector as a stream of values wrapped at every N values

    ## Formatting parameters
```

```
    float_fmtstr = "% .16E";
    values_per_line = 3;

    N = length(X);
    for k = 1:N
        fprintf(fh, float_fmtstr, X(k));
        if mod(k, values_per_line) == 0 || k == N
            fputs(fh, "\n");
        else
            fputs(fh, ' ');
        end
    end
end

make_vtk_cavity();
```

Arquivo 6.15 – `external/skeleton.rea`

```
****** PARAMETERS *****
2.610000     NEKTON VERSION
 2  DIMENSIONAL RUN
103 PARAMETERS FOLLOW
     0.0000000E+00     p01 DENSITY
     0.0000000E+00     p02 VISCOS
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00     p07 RHOCP
     0.0000000E+00     p08 CONDUCT
     0.0000000E+00
     0.0000000E+00     p10 FINTIME
     0.0000000E+00     p11 NSTEPS
     0.0000000E+00     p12 DT
     0.0000000E+00     p13 IOCOMM
     0.0000000E+00     p14 IOTIME
     0.0000000E+00     p15 IOSTEP
     0.0000000E+00     p16 PSSOLVER
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00     p22 HELMHOLTZ
 0              p23 NPSCAL
     0.0000000E+00     p24 TOLREL
     0.0000000E+00     p25 TOLABS
     0.0000000E+00     p26 COURANT/NTAU
     0.0000000E+00     p27 TORDER
     0.0000000E+00     p28 TORDER: mesh velocity (0: p28=p27)
     0.0000000E+00     p29 magnetic visc if > 0, = -1/Rm if < 0
     0.0000000E+00     p30 > 0 ==> properties set in uservp()
     0.0000000E+00     p31 NPERT: #perturbation modes
     0.0000000E+00     p32 #BCs in re2 file, if > 0
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00     p42 0=gmres/1=pcg
     0.0000000E+00     p43 0=semg/1=schwarz
     0.0000000E+00     p44 0=E-based/1=A-based prec.
     0.0000000E+00     p45 Relaxation factor for DTFS
     0.0000000E+00      p46 if >0, dont call SETICS
     0.0000000E+00     p47 vnu: mesh matieral prop
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00
     0.0000000E+00     p52 IOHIS
     0.0000000E+00
```

```
58        0.0000000E+00     p54 1,2,3-->fixed flow rate dir=x,y,z
59        0.0000000E+00     p55 vol.flow rate(p54>0) or Ubar(p54<0)
60        0.0000000E+00
61        0.0000000E+00
62        0.0000000E+00
63        0.0000000E+00     p59 !=0 --> use std axhelm for all elem
64        0.0000000E+00     p60 !=0--> init velocity to small nonzero
65        0.0000000E+00
66        0.0000000E+00     p62 >0 --> force byte_swap for output
67        0.0000000E+00     p63 =8 --> force 8-byte output
68        0.0000000E+00     p64 =1 --> perturbation restart
69        0.0000000E+00     p65 #iofile(eg,0 or 64); <0 --> sep. dirs
70        0.0000000E+00     p66 write fmt:ONLY postx uses rea value
71        0.0000000E+00     p67 read fmt: same modes as p66
72        0.0000000E+00     p68 iastep: freq for avg_all
73        0.0000000E+00
74        0.0000000E+00
75        0.0000000E+00
76        0.0000000E+00
77        0.0000000E+00
78        0.0000000E+00     p74 verbose Helmholtz
79        0.0000000E+00
80        0.0000000E+00
81        0.0000000E+00
82        0.0000000E+00
83        0.0000000E+00
84        0.0000000E+00
85        0.0000000E+00
86        0.0000000E+00
87        0.0000000E+00
88        0.0000000E+00     p84 !=0 -->sets initial timestep if p12>0
89        0.0000000E+00     p85 dt ratio if p84 !=0, for timesteps>0
90        0.0000000E+00     p86 reserved
91        0.0000000E+00
92        0.0000000E+00
93        0.0000000E+00     p89 reserved
94        0.0000000E+00
95        0.0000000E+00
96        0.0000000E+00
97        0.0000000E+00     p93 Numbr of prev pressure solns saved
98        0.0000000E+00     p94 start projecting vel. after p94 step
99        0.0000000E+00     p95 start projecting pr after p95 step
100       0.0000000E+00     p96 u0 translational velocity (in .usr)
101       0.0000000E+00      p97 v0 translational velocity (in .usr)
102       0.0000000E+00
103       0.0000000E+00     p99    dealiasing:if <0 disable
104       0.0000000E+00     p100 reserved
105       0.0000000E+00     p101   No. of additional filter modes
106       0.0000000E+00
107       0.0000000E+00     p103   weight of stabilizing filter (.01)
108       4  Lines of passive scalar data follows2 CONDUCT; 2RHOCP
109    1.00000       1.00000       1.00000       1.00000       1.00000
110    1.00000       1.00000       1.00000       1.00000
111    1.00000       1.00000       1.00000       1.00000       1.00000
112    1.00000       1.00000       1.00000       1.00000
113             13  LOGICAL SWITCHES FOLLOW
114    T     IFFLOW
115    T     IFHEAT
116    T     IFTRAN
117    T F F F F F F F F F F IFNAV & IFADVC (convection in P.S. fields)
118    F F T T T T T T T T T IFTMSH (IF mesh for this field is T mesh)
```

```
119     F     IFAXIS
120     F     IFSTRS
121     F     IFSPLIT
122     F     IFMGRID
123     F     IFMODEL
124     F     IFKEPS
125     F     IFMVBD
126     F     IFCHAR
127      8.00000       8.00000     -0.500000      -4.00000     XFAC,YFAC,XZERO,YZERO
128    **MESH DATA** 1st line is X of corner 1,2,3,4. 2nd line is Y.
129              -1  2             1     NELT,NDIM,NELV
130                0 PRESOLVE/RESTART OPTIONS  *****
131                7         INITIAL CONDITIONS *****
132   C Default
133   C Default
134   C Default
135   C Default
136   C Default
137   C Default
138   C Default
139     ***** DRIVE FORCE DATA ***** BODY FORCE, FLOW, Q
140                4                 Lines of Drive force data follow
141   C
142   C
143   C
144   C
145     ***** Variable Property Data ***** Overrrides Parameter data.
146                1 Lines follow.
147                0 PACKETS OF DATA FOLLOW
148     ***** HISTORY AND INTEGRAL DATA *****
149                0   POINTS.  Hcode, I,J,H,IEL
150     ***** OUTPUT FIELD SPECIFICATION *****
151                6 SPECIFICATIONS FOLLOW
152     T      COORDINATES
153     T      VELOCITY
154     T      PRESSURE
155     F      TEMPERATURE
156     F      TEMPERATURE GRADIENT
157                0      PASSIVE SCALARS
158     ***** OBJECT SPECIFICATION *****
159          0 Surface Objects
160          0 Volume  Objects
161          0 Edge    Objects
162          0 Point   Objects
```

Arquivo 6.16 – m-files/plot/plot_boundary_layers.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {} plot_boundary_layers (@
## @var{ax_99_h}, @var{ax_star_h}, @var{ax_theta_h}, @
## @var{mesh_s}, @var{flow_s}, @var{par_s}, @var{domain_s}@
## )
##
## Plot comparison graphs of boundary layer thicknesses.
##
## Uses @code{boundary_layer_thickness} and @code{blasius_parameters} to
## compute simulated and ideal values of thickness on the grid x values.
## The same axis handle may be used to plot all of them, that is
## @code{@var{ax_99_h} = @var{ax_star_h} = @var{ax_theta_h} = @var{ax_h}},
## so that they appear in the same graph.
##
## @var{mesh_s} and @var{flow_s} must have the standard fields generated by
## the lab's Matlab CFD program.
## @var{par_s} is the parameters struct generated with @code{get_parameters}
## from the labcode setup file @code{parameters.m}.
## @var{domain_s} should be generated from one of the @code{get_*_domain}
## functions, or extracted from @var{mesh_s}.
##
## @end deftypefn

function plot_boundary_layers(
        ax_99_h, ax_star_h, ax_theta_h, mesh_s, flow_s, par_s, domain_s)
    ## Plots the boundary layer over x, assuming that the plate is located
    ## at the bottom of the domain.

    X = mesh_s.X;
    Y = mesh_s.Y;

    ## Crop indices
    x_i = find(X >= domain_s.xi, 1);
    x_f = find(X >= domain_s.xf, 1);
    y_i = find(Y >= domain_s.yi, 1);
    y_f = find(Y >= domain_s.yf, 1);

    ## Manually looping for processing flow data
    N = x_f - x_i + 1;
    delta_99 = zeros(1, N);
    delta_star = zeros(1, N);
    theta = zeros(1, N);
    for i = 1:N
        [delta_99(i), delta_star(i), theta(i)] = boundary_layer_thickness(
            X(x_i + i), Y(y_i), mesh_s, flow_s, 1
        );
    end

    ## Ideal parameters
    [delta_99_id, delta_star_id, theta_id] = blasius_parameters(
        par_s.dref * X(x_i:x_f), par_s.Ufs, par_s.nu
    );
```

```
58
59      plot_single_boundary(
60          ax_99_h, X(x_i:x_f), delta_99, delta_99_id / par_s.dref
61      );
62      plot_single_boundary(
63          ax_star_h, X(x_i:x_f), delta_star, delta_star_id / par_s.dref
64      );
65      plot_single_boundary(
66          ax_theta_h, X(x_i:x_f), theta, theta_id / par_s.dref
67      );
68  end
69
70  function plot_single_boundary(ax_h, X, delta, delta_id)
71      ## Resets hold state at function return or error.
72      if not(ishold(ax_h))
73          hold(ax_h, 'on');
74          _ax_cleanup = onCleanup(@() hold(ax_h, "off"));
75      end
76
77      ## Plot experimental data
78      exp_h = plot(ax_h, X, delta,
79                   'LineStyle', 'none', 'MarkerSize', 2, 'Marker', '.',
80                   'Color', 'black',
81                   'DisplayName', 'Sim.');
82
83      ## Plot ideal as lines
84      id_h = plot(ax_h, X, delta_id,
85                  'LineWidth', 1, 'LineStyle', '--',
86                  ## XKCD steel grey
87                  'Color', '#6f828a',
88                  'DisplayName', 'Blasius');
89      legend(ax_h, "Location", 'northwest');
90  end
```

Arquivo 6.17 – m-files/plot/plot_cavity_xlim_indicator.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {} plot_cavity_xlim_indicator (@
## @var{ax_h}, @var{mesh_s}@
## )
##
## Plots two vertical lines demarking the cavity start and end x coordinates.
##
## @var{mesh_s} must have the standard fields generated by
## the lab's Matlab CFD program.
##
## @end deftypefn

function plot_cavity_xlim_indicator(ax_h, mesh_s)
    for k = 1:2
        xk = mesh_s.flowType.cav{1}.x(k);
        line(ax_h,
             ## Theoretically faster syntax; read the help page for `line`.
             'xdata', [xk, xk],
             'ydata', get(ax_h, 'ylim'),
             ## Width in points
             'LineWidth', 1,
             ## Dot-line pattern
             'LineStyle', '-.',
             ## XKCD silver
             'Color', '#C5C9C7',
             ## Prevents `legend` throwing warnings on line addition
             'HandleVisibility', 'off'
        )
    end
end
```

Arquivo 6.18 — m-files/plot/plot_flow_arrows.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {} plot_flow_arrows (@
## @var{ax_h}, @var{mesh_s}, @var{flow_s}, @var{domain_s}, @var{Nx}, @var{Ny}@
## )
##
## Plots velocity-oriented arrows in a grid of @var{Nx} * @var{Ny} points
##
## @var{mesh_s} and @var{flow_s} must have the standard fields generated by
## the lab's Matlab CFD program.
## @var{domain_s} should be generated from one of the @code{get_*_domain}
## functions, or extracted from @var{mesh_s}.
##
## @end deftypefn

function plot_flow_arrows(ax_h, mesh_s, flow_s, domain_s, Nx, Ny)
    ## Crop indices
    x_i = find(mesh_s.X >= domain_s.xi, 1);
    x_f = find(mesh_s.X >= domain_s.xf, 1);
    y_i = find(mesh_s.Y >= domain_s.yi, 1);
    y_f = find(mesh_s.Y >= domain_s.yf, 1);

    ## Force sampling at N points
    rx = floor((x_f - x_i) ./ Nx);
    ry = floor((y_f - y_i) ./ Ny);
    f_x = x_i:rx:x_f;
    f_y = y_i:ry:y_f;

    quiver(ax_h,
           mesh_s.X(f_x), mesh_s.Y(f_y),
           flow_s.U(f_x, f_y)', flow_s.V(f_x, f_y)',
           "Color", "white"
    );
end
```

Arquivo 6.19 – m-files/plot/plot_flow_contourf.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {} plot_flow_contourf (@
## @var{ax_h}, @var{mesh_s}, @var{flow_s}, @var{domain_s}, @
## @var{breakpoints}@
## )
##
## Plots the flow as a contour, bounded by the specified domain.
##
## @var{breakpoints} may be an integer representing the number of steps
## in the contour, or a vector of contour levels.
##
## @var{mesh_s} and @var{flow_s} must have the standard fields generated by
## the lab's Matlab CFD program.
## @var{domain_s} should be generated from one of the @code{get_*_domain}
## functions, or extracted from @var{mesh_s}.
##
## @end deftypefn

function plot_flow_contourf(ax_h, mesh_s, flow_s, domain_s, breakpoints)
    ## Crop indices
    x_i = find(mesh_s.X >= domain_s.xi, 1);
    x_f = find(mesh_s.X >= domain_s.xf, 1);
    y_i = find(mesh_s.Y >= domain_s.yi, 1);
    y_f = find(mesh_s.Y >= domain_s.yf, 1);

    ## Plot region borders using contour
    contourf(ax_h,
             mesh_s.X(x_i:x_f),
             mesh_s.Y(y_i:y_f),
             flow_s.U(x_i:x_f, y_i:y_f)',
             breakpoints);
end
```

Arquivo 6.20 – m-files/plot/plot_flow_streamlines.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {} plot_flow_streamlines (@
## @var{ax_h}, @var{mesh_s}, @var{flow_s}, @var{domain_s}
## )
##
## Plots the flow streamlines, bounded by the specified domain.
##
## @var{mesh_s} and @var{flow_s} must have the standard fields generated by
## the lab's Matlab CFD program.
## @var{domain_s} should be generated from one of the @code{get_*_domain}
## functions, or extracted from @var{mesh_s}.
##
## @end deftypefn

function plot_flow_streamlines(ax_h, mesh_s, flow_s, domain_s)
    ## Crop indices
    x_i = find(mesh_s.X >= domain_s.xi, 1);
    x_f = find(mesh_s.X >= domain_s.xf, 1);
    y_i = find(mesh_s.Y >= domain_s.yi, 1);
    y_f = find(mesh_s.Y >= domain_s.yf, 1);

    # Square root of the number of streamlines
    N = 7;

    ## Grid of data points
    [mx, my] = meshgrid(mesh_s.X(x_i:x_f), mesh_s.Y(y_i:y_f));

    ## Select N x N probes, discarding points on the wall
    [sx, sy] = meshgrid(linspace(mesh_s.X(x_i), mesh_s.X(x_f), N+2)(2:end-1),
                        linspace(mesh_s.Y(y_i), mesh_s.Y(y_f), N+2)(2:end-1));

    ## Plot the streamlines
    s_h = streamline(
        ax_h,
        ## Slicing the domain
        mx, my,
        flow_s.U(x_i:x_f, y_i:y_f)', flow_s.V(x_i:x_f, y_i:y_f)',
        ## Generated probes
        sx, sy
    );

    ## And colour them
    for i = 1:length(s_h)
        set(s_h(i), 'Color', 'white');
    end
end
```

Arquivo 6.21 – m-files/plot/plot_mesh_walls_in_domain.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {} plot_mesh_walls_in_domain (@
## @var{ax_h}, @var{mesh_s}, @var{domain_s}
## )
##
## Plots the wall as a contour, bounded by the specified domain.
##
## @var{mesh_s} must have the standard fields generated by
## the lab's Matlab CFD program.
## @var{domain_s} should be generated from one of the @code{get_*_domain}
## functions, or extracted from @var{mesh_s}.
##
## @end deftypefn

function plot_mesh_walls_in_domain(ax_h, mesh_s, domain_s)
    X = mesh_s.X;
    Y = mesh_s.Y;
    wall = mesh_s.wall;

    ## Make wall a number
    wall = int8(wall);

    ## Crop indices
    x_i = find(X >= domain_s.xi, 1);
    x_f = find(X >= domain_s.xf, 1);
    y_i = find(Y >= domain_s.yi, 1);
    y_f = find(Y >= domain_s.yf, 1);

    ## Plot region borders using contour
    contourf(ax_h, X(x_i:x_f), Y(y_i:y_f), wall(x_i:x_f, y_i:y_f)');
end
```

Arquivo 6.22 – `m-files/boundary_layer/blasius_parameters.m`

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@
## [@var{delta_99}, @var{delta_star}, @var{theta}] =} blasius_parameters (@
## @var{x}, @var{U}, @var{nu}@
## )
##
## Computes three types of boundary layer thicknesses algebraically.
##
## All values are dimensional. The coefficient values are approximate,
## based on a semi-infinite plate.
##
## @var{x} may be a scalar or a matrix, generating an output with same shape.
##
## @end deftypefn

function [delta_99, delta_star, theta] = blasius_parameters(x, U, nu)
    c = zeros(size(x));
    c(x >= 0) = realsqrt(nu .* x(x >= 0) ./ U);
    delta_99 = 5.29 .* c;
    delta_star = 1.72 .* c;
    theta = 0.665 .* c;
end
```

Arquivo 6.23 – m-files/boundary_layer/boundary_layer_thickness.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@
## [@var{delta_99}, @var{delta_star}, @var{theta}] =} boundary_layer_thickness (@
## @var{x}, @var{y0}, @var{mesh_s}, @var{flow_s}, @var{dref} @
## )
##
## Computes the three types of boundary layer thicknesses using flow data.
##
## @var{x} and @var{y0} must be adimensional, with reference length @var{dref},
## and @var{flow_s} must also have adimensional velocities.
##
## @var{mesh_s} and @var{flow_s} must have the standard fields generated by
## the lab's Matlab CFD program.
##
## All thicknesses are dimensional. The surface y value is considered to be
## the top of the cavity.
##
## Only works on scalars because of `find`, which cannot search more than one
## row.
##
## Only integrates up to the 99% boundary layer for delta_star and theta.
##
## @end deftypefn

function [delta_99, delta_star, theta] = boundary_layer_thickness(
        x, y0, mesh_s, flow_s, dref)
    # Find position indices
    x_i = find(mesh_s.X >= x, 1);
    y0_i = find(mesh_s.Y >= y0, 1);

    ## Dimensionalised viscous thickness at 99% * U_inf
    delta_99_i = find(flow_s.U(x_i, y0_i:end) >= 0.99, 1) + y0_i - 1;
    delta_99 = (mesh_s.Y(delta_99_i) - mesh_s.Y(y0_i)) .* dref;

    ## Vertical slice of horizontal speed starting from (x, y0) up to
    ## the end of the boundary layer
    U_slice = flow_s.U(x_i, y0_i:delta_99_i+1);

    ## Dimensionalised displacement thickness
    delta_star = trapz(mesh_s.Y(y0_i:delta_99_i+1), 1 - U_slice) .* dref;
    ## Dimensionalised momentum thickness
    theta = trapz(mesh_s.Y(y0_i:delta_99_i+1), U_slice .* (1 - U_slice)) .* dref;
end
```

Arquivo 6.24 – m-files/domain/get_cavity_domain.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@var{cavity_domain_s} =} get_cavity_domain (@
## @var{mesh_s}@
## )
##
## Creates a domain-shaped struct containing only the cavity.
##
## Its shape is given in @code{parameters.m}.
## All members are adimensional lengths.
##
## Useful for plotting flow arrows inside the cavity.
##
## @end deftypefn

function cavity_domain_s = get_cavity_domain(mesh_s)
    cavity_domain_s = struct(
        ## Cavity left
        'xi', mesh_s.X(find(mesh_s.X >= mesh_s.flowType.cav{1}.x(1), 1)),
        ## Cavity right
        'xf', mesh_s.X(find(mesh_s.X >= mesh_s.flowType.cav{1}.x(2), 1)),
        ## Cavity bottom
        'yi', mesh_s.Y(find(mesh_s.Y >= mesh_s.flowType.cav{1}.y(1), 1)),
        ## Cavity top
        'yf', mesh_s.Y(find(mesh_s.Y >= mesh_s.flowType.cav{1}.y(2), 1)),
        ## Useless z
        'zi', 0,
        'zf', 1
    );
end
```

Arquivo 6.25 – m-files/domain/get_cavity_surroundings.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@var{zoom_domain_s} =} get_cavity_surroundings (@
## @var{mesh_s}@
## )
##
## Creates a domain-shaped struct closer to the cavity, above the plane surface.
##
## Its shape is given in @code{parameters.m}.
## All members are adimensional lengths.
##
## Useful for plotting boundary layer thicknesses.
##
## @end deftypefn

function zoom_domain_s = get_cavity_surroundings(mesh_s)
    zoom_domain_s = struct(
        ## 20 drefs before cavity start
        'xi', mesh_s.X(find(mesh_s.X >= mesh_s.flowType.cav{1}.x(1) - 20, 1)),
        ## 90 drefs after cavity end
        'xf', mesh_s.X(find(mesh_s.X >= mesh_s.flowType.cav{1}.x(2) + 90, 1)),
        ## Cavity top
        'yi', mesh_s.Y(find(mesh_s.Y >= mesh_s.flowType.cav{1}.y(2), 1)),
        ## 4 drefs above cavity top
        'yf', mesh_s.Y(find(mesh_s.Y >= mesh_s.flowType.cav{1}.y(2) + 4, 1)),
        ## Useless z
        'zi', 0,
        'zf', 1
    );
end
```

Arquivo 6.26 – m-files/domain/get_cavity_vslice.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@var{vslice_domain_s} =} get_cavity_vslice (@
## @var{mesh_s}@
## )
##
## Creates a domain-shaped struct extending from the cavity upwards.
##
## Its shape is given in @code{parameters.m}.
## All members are adimensional lengths.
##
## The motivation for this is to avoid a bug in Octave where NaNs are not
## excluded from plots and generate wrong drawings.
##
## @end deftypefn

function vslice_domain_s = get_cavity_vslice(mesh_s)
    vslice_domain_s = struct(
        ## Cavity left
        'xi', mesh_s.X(find(mesh_s.X >= mesh_s.flowType.cav{1}.x(1), 1)),
        ## Cavity right
        'xf', mesh_s.X(find(mesh_s.X >= mesh_s.flowType.cav{1}.x(2), 1)),
        ## Cavity bottom
        'yi', mesh_s.Y(find(mesh_s.Y >= mesh_s.flowType.cav{1}.y(1), 1)),
        ## 4 drefs above cavity top
        'yf', mesh_s.Y(find(mesh_s.Y >= mesh_s.flowType.cav{1}.y(2) + 4, 1)),
        ## Useless z
        'zi', 0,
        'zf', 1
    );
end
```

Arquivo 6.27 – m-files/domain/get_upper_domain.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@var{upper_domain_s} =} get_upper_domain (@
## @var{mesh_s}, @var{domain_s}@
## )
##
## Create a domain-shaped struct containing only the above-plate flow.
##
## Its shape is given in @code{parameters.m}.
## All members are adimensional lengths.
##
## The motivation for this is to avoid a bug in Octave where NaNs are not
## excluded from plots and generate wrong drawings.
##
## Useful for plotting boundary layer thicknesses and flow contours.
##
## @end deftypefn

function upper_domain_s = get_upper_domain(mesh_s, domain_s)
    upper_domain_s = struct(
        'xi', domain_s.xi,
        'xf', domain_s.xf,
        ## Plane surface
        'yi', max([domain_s.yi, 0]),
        'yf', domain_s.yf,
        ## Useless z
        'zi', 0,
        'zf', 1
    );
end
```

Arquivo 6.28 – `m-files/etc/get_parameters.m`

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@var{cavity_domain_s} =} get_parameters (@
## @var{mesh_s}@
## )
##
## Wrapper for the @code{parameters.m} script.
##
## Runs the parameters file in a separate lexical context and
## returns its most important values wrapped in a struct, as with @code{load}.
##
## @end deftypefn

function P = get_parameters(filename)
    source(filename);
    P = struct(
        "domain", domain,
        "flowParameters", flowParameters,
        "flowType", flowType,
        "dref", dref,
        "nu", nu,
        "Ufs", Ufs
    );
end
```

Arquivo 6.29 – m-files/etc/nondimensionalise_s.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@var{nondim_s} =} nondimensionalise_s (@
## @var{dim_s}, @var{ref_val}@
## )
##
## Nondimensionalises every field of @var{dim_s}, dividing by @var{ref_val}.
##
## Mainly used to nondimensionalise boundary layer thickness structs.
##
## @end deftypefn

function nondim_s = nondimensionalise_s(dim_s, ref_val)
    nondim_s = struct();
    fs = fieldnames(dim_s);
    for i = 1:length(fs)
        f = fs{i};
        nondim_s = setfield(nondim_s, f, getfield(dim_s, f) / ref_val);
    end
end
```

Arquivo 6.30 – m-files/genbox/write_box.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {} write_box (@
## @var{fh}, @var{box_s}, @var{box_id}, @var{nfields}@
## )
##
## Write a single box mesh.
##
## Includes the number of nodes, a wrapped list of coordinates, and
## its boundary conditions.
##
## @end deftypefn

function write_box(fh, box_s, box_id, nfields)
    ## Header
    ### Box identification
    fprintf(fh, "box_%02d\n", box_id);
    ### Number of elements on X and Y (nelem = nnodes - 1)
    fprintf(fh, "%d %d\n", length(box_s.x) - 1, length(box_s.y) - 1);

    ## Formatting parameters
    double_fmtstr = "% .16f";
    nodes_per_line = 4;

    ## X boundary nodes
    fputs(fh, "# X nodes start\n");
    for k = 1:length(box_s.x)
        fprintf(fh, double_fmtstr, box_s.x(k));
        if mod(k, nodes_per_line) == 0 || k == length(box_s.x)
            fputs(fh, "\n");
        else
            fputs(fh, ' ');
        end
    end
    fputs(fh, "# X nodes end\n");

    ## Y boundary nodes
    fputs(fh, "# Y nodes start\n");
    for k = 1:length(box_s.y)
        fprintf(fh, double_fmtstr, box_s.y(k));
        if mod(k, nodes_per_line) == 0 || k == length(box_s.y)
            fputs(fh, "\n");
        else
            fputs(fh, ' ');
        end
    end
    fputs(fh, "# Y nodes end\n");

    ## Boundary conditions
    fputs(fh, "# Velocity boundary conditions\n");
    for k = 1:length(box_s.U_boundaries)
        fprintf(fh, "%s,", box_s.U_boundaries{k});
    end
```

```
58      fputs(fh, "\n");
59
60      if nfields == 2
61          fputs(fh, "# Temperature boundary conditions\n");
62          for k = 1:length(box_s.T_boundaries)
63              fprintf(fh, "%s,", box_s.T_boundaries{k});
64          end
65          fputs(fh, "\n");
66      end
67  end
```

Arquivo 6.31 – `m-files/genbox/write_box_preamble.m`

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {} write_box_preamble (@
## @var{fh}, @var{nfields}@
## )
## @deftypefnx {Function File} {} write_box_preamble (@
## @var{fh}, @var{nfields}, @var{skeleton_rea_filename}@
## )
##
## Write the preamble for @code{.box} files.
##
## Includes the @code{.rea} source (if ASCII output is desired),
## the number of mesh dimensions
## and the number of computed fields @var{nfields}
## (flow, with or without temperature).
##
## I believe the initial @code{.rea} parameters are ignored by @code{reatore2},
## since it seems to put them in a separate @code{.rea} output file after you
## convert it to @code{.re2}.
##
## @end deftypefn

function write_box_preamble(fh, nfields, skeleton_rea_filename = '')
    is_binary_output = strcmp(skeleton_rea_filename, '');

    ## Mesh geometric dimensions (negative gives binary mesh (.re2) output)
    if is_binary_output
        dims = -2;
    else
        dims = 2;
    end

    ## If ASCII, the box needs to include a .rea file with run parameters
    if not(is_binary_output)
        fprintf(fh, "%s\n", skeleton_rea_filename);
    end

    fprintf(fh, "%1d      2D (negative for binary output)\n", dims);
    fprintf(fh, "%d       number of fields\n", nfields);
end
```

Arquivo 6.32 – m-files/genbox/write_box_files.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {} write_box_files (@
## @var{boxes_s}, @var{nfields}, @var{output_dir}, @var{skeleton_rea_filename}@
## )
##
## Write a series of files corresponding to each section of a @code{.box}
##
## The preamble and each box get its own files, which are saved under
## @var{output_dir}. The @var{skeleton_rea_filename} is the name of the
## @code{.rea} file to be used as the template for generating a @code{.rea},
## the textual mesh format.
## Its path should be relative to the resulting @code{.box}.
##
## @end deftypefn

function write_box_files(boxes_s, nfields, output_dir, skeleton_rea_filename = '')
    ## Preamble to box file
    preamble_filename = "00-preamble.part.box";
    full_path = fullfile(output_dir, preamble_filename);
    fh = fopen(full_path, 'w+');
    _fh_cleanup = onCleanup(@() fclose(fh));

    ## File start header
    fprintf(fh, "## %s starts here\n", preamble_filename);
    ## Preamble body
    write_box_preamble(fh, nfields, skeleton_rea_filename);
    ## File end footer
    fprintf(fh, "## %s ends here\n", preamble_filename);

    clear _fh_cleanup;

    ## Names each file according to its field name in the struct
    fields = fieldnames(boxes_s);

    ## Prepares each box as a partial file to be concatenated
    for i = 1:length(fields)
        filename = sprintf("%02d-%s.part.box", i, fields{i});
        full_path = fullfile(output_dir, filename);
        fh = fopen(full_path, 'w+');
        _fh_cleanup = onCleanup(@() fclose(fh));

        ## File start header
        fprintf(fh, "## %s starts here\n", filename);
        ## Box body
        write_box(fh, getfield(boxes_s, fields{i}), i, nfields);
        ## File end footer
        fprintf(fh, "## %s ends here\n", filename);

        clear _fh_cleanup;
    end
end
```

Arquivo 6.33 – m-files/genbox/velocity_boundary_conditions.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@var{bc_s} =} velocity_boundary_conditions ()
##
## Get an enum-like struct with velocity boundary condition abbreviations.
##
## This struct only lists the conditions required by this use case.
##
## Since Octave 8.x still does not support enums, this function serves as
## a way to fetch the boundary conditions as required by Nek.
##
## @seealso{temperature_boundary_conditions}
##
## @end deftypefn

function bc_s = velocity_boundary_conditions()
    ## Nek boundary condition codes, 3-character-padded
    bc_s = struct(
        ## E = internal boundary
        'internal', 'E  ',
        ## W = wall
        'wall', 'W  ',
        ## v = user-defined velocity (in .usr program)
        'vel_u', 'v  ',
        ## SYM = symmetry plane (= only in-plane flow)
        'slip_free', 'SYM',
        ## O = outflow in any direction
        'outflow', 'O  '
    );
end
```

Arquivo 6.34 – m-files/genbox/temperature_boundary_conditions.m

```
## Copyright
## SPDX-FileCopyrightText: 2024 Fernando Datz
## SPDX-License-Identifier: BSD-2-Clause
## Author: Fernando Datz

## -*- texinfo -*-
## @deftypefn {Function File} {@var{bc_s} =} temperature_boundary_conditions ()
##
## Get an enum-like struct with temperature boundary condition abbreviations.
##
## This struct only lists the conditions required by this use case.
##
## Since Octave 8.x still does not support enums, this function serves as
## a way to fetch the boundary conditions as required by Nek.
##
## @seealso{velocity_boundary_conditions}
##
## @end deftypefn

function bc_s = temperature_boundary_conditions()
    ## Nek boundary condition codes, 3-character-padded
    bc_s = struct(
        ## E = internal boundary
        'internal', 'E  ',
        ## I = insulated wall (= no gradient). Aliases: O, SYM
        'insulated', 'I  ',
        ## t = user-defined temperature (in .usr program)
        'temp_u', 't  '
    );
end
```